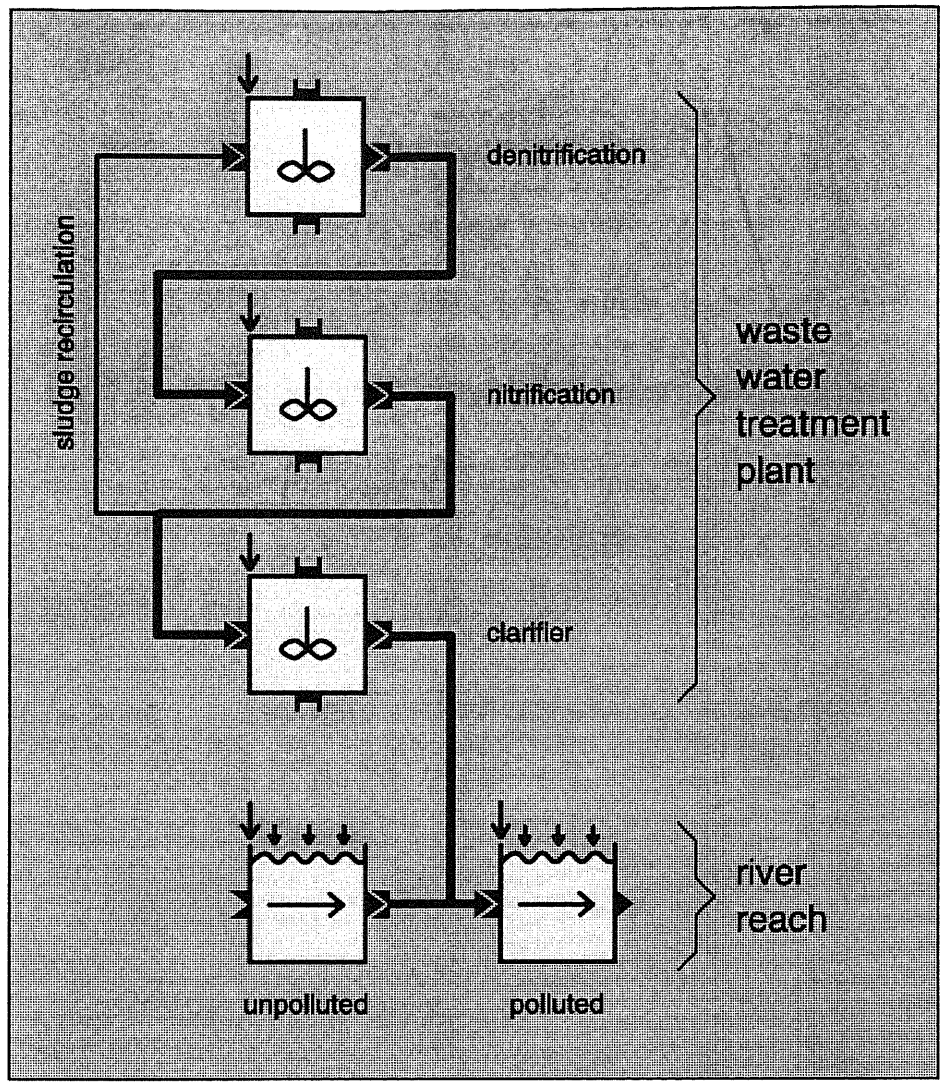
EAWAG

A Research Institution
of the ETH



SCHRIFTENREIHE DER EAWAG Nr. 7

Peter Reichert

Concepts underlying a Computer Program for the Identification and Simulation of Aquatic Systems

ISBN: 3-906484-08-4

Swiss Federal Institute for Environmental Science
and Technology (EAWAG)

CH-8600 Dübendorf (Switzerland)

1994

Acknowledgments

The ideas for the environmental identification and simulation program described in this report grew from the experiences made in a lot of interdisciplinary studies at EAWAG, in which I have been involved during the past eight years. It is not possible to mention all persons, who contributed with the discussion of their data interpretation and modelling problems to the concepts of this program.

By far the largest influence is due to Oskar Wanner. I would like to thank him for the excellent conditions under which I could work, and for the large number of inspiring discussions on data analysis and parameter estimation problems during this time. Without these collaborations, which were the most important driving forces for the design and realization of the program described in this report, this project would not have been realized. These studies led to the desire for a data analysis program, which is not only a highly flexible simulation tool, but also allows its users to perform identifiability analyses, parameter estimations and uncertainty analyses.

The second important influence is due to Jürg Ruchti who raised my interest for object-oriented programming and for the programming language C++. The application of the concepts of object-oriented programming made a much more robust and extensible program design possible than conventional programming techniques. I would also like to thank him for implementing the formula variables and the plotting facilities of the program realized according to the guidelines described in this report.

Furthermore, I would like to thank Hans-Peter Bader, Olaf Cirpka, Dieter Imboden and Werner Simon for important comments to the manuscript, Willi Gujer, Hans-Ruedi Siegrist and Manfred Tschui for their advice with respect to the activated sludge example, Werner Simon and Markus Ulrich for their hints for improving the quality of the user manual, Achim Albrecht, Olaf Cirpka, Maria Fruhen, Ivana Jancarkova, Norbert Mattle, Max Maurer, Reto von Schulthess, Urs Uehlinger, Oskar Wanner and Daniel Wild for using and criticizing preliminary program versions, Gérard Mohler, Bouziane Outiti and Raoul Schaffner for their support in solving technical problems involving various hardware platforms, David Livingstone for very carefully correcting the most important parts of the manuscript, and Alexander Zehnder for making publication of this report as an issue of "Schriftenreihe der EAWAG" possible.

1 Introduction

The interpretation of measured data is a problem common to all natural sciences. In most environmental studies, measured data cannot be used directly to assess the validity of hypotheses about the mechanisms acting within a given system; instead, an indirect test based on the implications of the hypotheses must be employed. Hypotheses about functional relationships in environmental systems are usually stated in the form of mathematical models. With the exception of very simple (but important) models, computer programs are required to calculate the consequences of model assumptions.

Many computer programs are available for the simulation of environmental systems. Most conventional environmental simulation programs implement a specific model (or a limited set of models). For this reason, such programs are not suitable for carrying out the model comparisons needed for the process of system identification. This process consists of finding an "adequate" model which should be as simple as possible, but sufficiently complex to describe the features shown by measured data. System identification programs specifically designed for this purpose are usually not flexible enough to allow their users to formulate models typical of environmental systems. General purpose simulation software which can be combined with identification programs is difficult to use. For these reasons, there is a need for a more universal, user-friendly simulation and identification tool for environmental systems. Such a program should allow its users to define their models in a flexible way, it should be relatively easy to be operated by environmental scientists and engineers, and it should combine simulation with system identification and uncertainty analysis.

In this report, it is shown how such an identification and simulation program for an important class of aquatic systems in the environment, in technical plants, and in the laboratory, can be designed. The proposed solution is based on mutually compatible concepts on structural, methodological, mathematical and technical levels, which are described in detail in this report. The aim of this report is not only to demonstrate the possibility of such an approach and to provide an instrument for scientific research, but also to stimulate discussion of the application of the methods of systems analysis to environmental systems and to encourage other developers to design more universally applicable environmental system identification software than has so far been the case.

Survey of Contents

In chapter 2, the necessity of employing the methods of systems analysis to interpret measurements from environmental systems is discussed by reviewing the role of mathematical models in the environmental sciences. The need for using the methods of systems analysis in the most important field of model application in research, for system identification, is outlined.

As a tool for the application of the methods of systems analysis to environmental systems, a computer program is necessary. In chapter 3, the major capabilities of three categories of existing computer programs which can be used for this purpose are briefly reviewed, and the requirements of a more universal identification and simulation program are discussed.

In chapters 4 - 7, the concepts underlying an identification and simulation program for an important class of aquatic systems, which is designed to take a step in the direction outlined in chapter 3, are discussed. Each of the chapters 4 - 7 treats one of the following modelling levels:

- structural level: A general model structure, which is used for the formulation of aquatic systems is proposed in chapter 4. This structure forms the basis for a flexible formulation of models.
- methodological level: The selection of simulation and data analysis techniques, to be available in the first version of the program, is discussed in chapter 5.
- mathematical level: The numerical algorithms guaranteeing an efficient solution of the equations for time evolution, sensitivity analysis and parameter estimation are described in chapter 6.
- technical level: In chapter 7, the techniques used in implementing the computer program are discussed. This chapter focuses on the advantages of object-oriented program design for the implementation of a program described in the previous chapters.

An important concern of this report is to pay attention to the mutual compatibility of the concepts introduced at the different modelling levels. It is important to note that most of the above-mentioned concepts are not limited to aquatic systems, but can be applied to other systems as well.

In chapter 8, the flexibility and utility of an identification and simulation program implemented according to the guidelines discussed in the previous chapters is demonstrated by summarizing applications of this program to five well-known models of environmental and technical systems.

Finally, in chapter 9, the most important concepts presented in this report are summarized and possible future developments are discussed.

The appendix contains the user manual of version 1.0 of the program AQUASIM, which was implemented according to the guidelines described in this report. Because the user manual is self-contained, some overlap between the main chapters and the appendix is unavoidable.

2 The Role of Models in the Environmental Sciences

Research in natural sciences is an attempt to identify the mechanisms responsible for the observed behavior of natural systems. All ideas on the functional relationships in a system taken together form an abstract representation of the system, called a model, or more precisely, a *conceptual model.* In many cases, such ideas are formulated mathematically. Such *mathematical models* are referred to simply as "models" in this report. The mathematical formulation of models makes quantitative derivation of model behavior possible. ***The comparison of model predictions with measurements of the system to be described is used to test the hypotheses formulated in the model.*** While a failed test invalidates a model, successful tests can never validate model hypotheses, but only increase the confidence in the model. Therefore, model "validation" or "verification" in a strict sense is impossible (e.g. Popper, 1982; Caswell, 1976; Reckhow and Chapra, 1983). Model hypotheses should be tested as directly as possible to avoid ambiguity in the interpretation of the results. This is usually done by limiting the investigation to the isolated subsystem that is most sensitive to the hypotheses to be tested. The more directly a hypothesis can be tested on a subsystem, the less important is the formulation of a model for the complete system to perform such a test.

Due to the simultaneous action of a large number of processes, the investigation of environmental systems is very complicated. For this reason, research in the environmental sciences can best be conducted based on a combination of laboratory and field studies. Laboratory investigations can be used to characterize isolated processes under controlled conditions, whereas field investigations show the behavior of the system as a whole. Without using the knowledge obtained from laboratory experiments, it is very difficult to quantify the contributions of different processes to the net transformation rates observed in the field. ***The complex nature of environmental systems with important external influences and a large number of internal interactions, together with the necessity of taking into account results obtained from laboratory experiments for the evaluation of field data, makes the use of mathematical models for hypothesis testing extremely important in the environmental sciences.***

As an additional difficulty, environmental systems are open systems, and in many cases the detection of functional relationships must be based on natural variations in external variables instead of on well-designed experiments (the most important exception to this is tracer experiments, which can be performed, for instance, with fluorescent dye to investigate the hydraulic properties of aquatic systems, or with radioactively marked substances to investigate the biological metabolisms). The need to sample open systems with simultaneous and correlated variations in many external variables makes the detection of causal dependences more difficult than in the case of carefully designed experiments. Thus, ***the intrinsic nature of environmental systems as multi-input multi-output systems in which inputs often can only be measured but not controlled makes the use of advanced system and parameter identification techniques for the model building process important.***

In this chapter, the role of mathematical models in the environmental sciences is reviewed. In the first section, the importance of various classification criteria for models of environmental systems within the context of this report are discussed. In the second section, an overview is given over the three main fields of model application for planning measurements, for system identification, and for forecasting the future behavior of a system. In the third section, the procedure for system identification or model building is described in more detail. Finally, in the fourth section, the major sources of error of model calculations are discussed and possible methods for error estimation in model-based forecasts are described.

2.1 Classification of Mathematical Models

There are many criteria for the classification of mathematical models. In Table 2.1 pairs of model types, taken mainly from Jørgensen (1992), are listed. Three categories of criteria are distinguished: philosophy of the approach; application area; and mathematical form of the model.

Table 2.1: Classification of Models of Environmental Systems

category of criteria	pairs of model type	
philosophy of the approach	reductionist	holistic
	causal	black-box (phenomenological)
application area	research	management
mathematical form	linear	nonlinear
	autonomous	non-autonomous
	box (discrete space)	continuous space (different dimensionality possible)
	discrete time	continuous time
	deterministic	stochastic

Reductionist models are based on the attempt to include as many details as possible into the model and to describe the behavior of a system as the net effect of all processes. In contrast to this approach, ***holistic*** models are based on a few important global parameters and on general principles. ***Causal*** models describe system response as a consequence of input using the mechanistic structure of the system, whereas ***black-box*** models use empirical relationships between input and output. Most water quality models are a mixture of causal and phenomenological models, using different concepts at different levels of resolution. As an example, microbial growth rates are in most cases parameterized phenomenologically at the cell level, but macroscopic water flow and substance mass balances (even those due to microbial growth) are treated in a causal way. Phenomenological models may even be used to obtain simplified descriptions of situations in which the "validity" of a causal model is widely accepted. As an example, phenomenological parameterizations of turbulent correlations are used in equations describing mean values of turbulent flow, because the solution of the underlying Navier-Stokes equations is too difficult (e.g. Rodi, 1980; Stull, 1988). These model classification criteria characterizing the philosophy of the modelling approach are not of relevance in the present context, because one main objective of the program described in this report is to allow the user as much freedom as possible in model formulation. Therefore, this program can be used for the application of models of any of the types described above.

Whether a model is to be applied for ***research*** or for ***management*** purposes does not influence its mathematical form. For this reason, the program described in this report can on principle be applied in both fields. The emphasis on data evaluation and system identification techniques and the requirement of model definition by the user (a drawback of the *possibility* of model definition by the user), however, evidently make this program most suitable as a research and teaching tool.

The last category of model classification criteria listed in Table 2.1, the mathematical form of the model, is of special interest in the present context, because the model structure presented in chapter 4 does not include all possible model types. The distinction between ***linear*** and ***nonlinear*** models and between ***autonomous*** and ***non-autonomous*** models is not necessary in the present context, because nonlinear and non-autonomous models, as used in this report, include linear and autonomous models as special cases. The model structure described in chapter 4 makes the definition of ***box*** models and of ***continuous space*** models possible. The general model structure also does not limit the dimensionality of continuous space models. Nevertheless, the first version of the program only implements zero- and one-dimensional models. Although model structure would make the implementation of higher dimensional models possible, the numerical algorithms employed in the current version of the program would not be very efficient for this case (cf. chapter 6). For the last two model classification criteria, a decision was made which limits the generality of the approach. For this reason the distinction of ***discrete time*** and ***continuous time*** models and of ***deterministic*** and ***stochastic*** models is discussed in more detail in the following two subsections.

2.1.1 Continuous-Time versus Discrete-Time Models

Continuous-time models are based on formulations of the rates of change of the state variables (e.g. concentrations or masses of substances, densities or sizes of populations, temperature, etc.). The values of the state variables as functions of time are then obtained as the solution of a system of differential equations. In contrast to this approach of continuous evolution with time, discrete-time models are based on a division of the time scale into discrete intervals and specify the state variables in a given time interval as algebraic functions of the values in the immediately previous time interval (and, possibly, of the values in other previous time intervals). Both continuous-time and discrete-time models are of importance for the description of environmental systems (e.g. Rose, 1987; Yozdis, 1989; Wissel, 1989). ***The importance of discrete-time models is mainly for systems dominated by external cyclic variables***, in which the variations in the state variables from cycle to cycle are more important than those within a single cycle. The most important example of such behavior is the population dynamics of macroscopic individuals such as birds, fishes, etc. from year to year. Since for the organisms dominating substance turnover in aquatic systems (algae and bacteria), as well as for the substances important for water quality, the strong variations of their concentrations during the year is very important, ***the large majority of models for water quality in aquatic systems are continuous-time models*** (e.g. Straskraba and Gnauck, 1983; Jørgensen, 1983;

Gromiec et al., 1983; Mauersberger, 1983). This is the reason for limiting the discussion to this type of model in this report.

2.1.2 Deterministic versus Stochastic Models

Deterministic models assume the future behavior of a system to be completely determined by the knowledge of its present state and the future values of external variables which describe the influence of the environment on the modelled (sub)system. This assumption does not exclude uncertain results due to uncertain parameter values, initial state or external variables. Deterministic models are formulated as difference equations for discrete time models and as differential equations for continuous time models (e.g. Rose, 1987; Yozdis, 1989; Wissel, 1989; Jørgensen, 1992). Stochastic models also take into account the random influences of the temporal evolution of the system itself. This makes future behavior uncertain even in the case of exact knowledge of parameters, initial state and external variables. Stochastic models are formulated as stochastic difference or differential equations (Tiwari and Hobbie, 1976a; Kremer, 1983; Gard, 1988; Kloeden and Platen, 1992). ***Although the stochastic description of environmental systems may be more realistic, the large majority of ecological models formulated so far are deterministic models*** (Rose, 1987; Yozdis, 1989; Wissel, 1989; Jørgensen, 1992). The main reasons for this fact may be a lack of data for the characterization of random variables, high requirements of computational resources for solving stochastic differential equations and the success of deterministic models in describing average future behavior. Although for nonlinear systems deterministic calculations and averages of stochastic calculations may deviate significantly from each other (e.g. Tiwari and Hobbie, 1976a), deterministic calculations can be correct if the model is designed to follow the deterministic path of mean values of a more realistic (but possibly intractable) stochastic model (this is done in many areas of applied physics, e.g. in turbulence modelling: cf. Rodi, 1980; Stull, 1988). Due to the reasons given above, the discussion in this report is limited to deterministic models.

2.2 Fields of Model Application in the Environmental Sciences

There are three important fields of model application in the environmental sciences:

- measurement planning,
- system identification,
- forecasting.

These three fields of model application are discussed briefly in this section. As already mentioned in the introduction to chapter 2, system identification is in the center of interest for this report. Section 2.3 therefore treats this topic in more detail. The most difficult problem of model-based forecasting is the estimation of predictive uncertainty. This problem is discussed in more detail in section 2.4.

2.2.1 Measurement Planning

The aim employing mathematical models for measurement planning is the optimization of the sampling process with regard to maximizing the possible gain in information.

Two ranges of model application for measurement planning can be distinguished. The model can simply be used as a didactic tool for testing the completeness of the set of measured variables, or it can be used for optimizing the sampling process. In both cases, the model is used for forecasting the expected behavior of the system to be investigated, based on *a priori* knowledge and assumptions with respect to model structure and model parameters.

The didactic advantage of a predictive simulation of the expected behavior of a system is that such a calculation forces the modeler to provide estimates of all necessary boundary and initial conditions, model parameters and external variables. This diminishes the risk of forgetting to measure important quantities required for a subsequent simulation. It is important to note, however, that such a didactic model application is not necessary, because it is also possible to include all important quantities soley by ensuring that the experimant has been carefully designed. Nevertheless, practical experience shows that such a model prediction is often advantageous (in particular, because it additionally helps to optimize the sampling process, as described in the next paragraph). Note, however, that a didactic model application does not guarantee that all important quantities are measured, because the *a priori*

model may turn out to be incomplete. Therefore, even at the *a priory* model stage, alternative model formulations should be considered.

In many cases, at least for important submodels of the model to be found good *a priori* knowledge already exists. In such cases, this knowledge can be incorporated into the model used for measurement planning purposes in order to optimize the gain of information resulting from the measurements. This can best be done by conducting a sensitivity analysis of the measured quantities with regard to uncertain model parameters or submodels. Such an analysis can lead to the detection of sensitive time or space domains in which an increase in sampling density would lead to an especially large gain in the amount information obtained from the measurements. As an example, the hydraulic submodel of a river quality model can be used to predict travel times, and therefore to fix sampling intervals, at locations downstream of a pulse release site of a substance with badly known transformation rates. If *a priori* knowledge of model parameters is bad, average results of Monte Carlo simulations, as described in Tiwari and Hobbie (1976b), can be used to obtain model estimates. In the case of long-term investigations, an iterative process of modelling and improving measurement strategy is recommended. Updating parameter values in order to take account of new measurements can be done using the method described by Tiwari et al. (1978).

2.2.2 System Identification

The goal of system identification is to find and calibrate an adequate model for the system under investigation.

The iterative process of system identification, or model building, is very complicated. It consists of the following steps:

- model structure evaluation,
- identifiability analysis of model parameters,
- parameter and initial state estimation,
- model confirmation or falsification.

These four steps are described in more detail in section 2.3. Here, only a short overview is given to clarify the notion of model building or system identification.

Model structure evaluation consists of finding adequate model structures and of comparing their quality. The most difficult step is the concretization of the notion of "adequacy" of models and the finding of model structures to be tested ("conjectures"

according to Popper, 1982). There is no systematic approach to this step, but, according to Spriet (1985), guiding principles are that the proposed model describe the major causal mechanisms of the system, that it be possible to decide between competing model structures, and that measured data be appropriate for the identification of unknown model parameters. The most important criterion for the comparison of models is that the deviations between measurements and model calculations should be as small as possible (quality of fit). This criterion cannot be used alone, because it favors the use of complicated models with many parameters which are difficult to identify uniquely. For this reason, this criterion has to be complemented by a criterion of "parsimony" leading to a preference of simple model structures. Possible quantifications of these criteria are discussed in section 2.3.1.

Identifiability analysis of model parameters treats the problem of uniqueness and accuracy in determining the parameters of a given model based on given data. Methods which can be applied for identifiability analysis are reviewed in section 2.3.2.

Parameter estimation methods determine "optimal" choices of the values of the model parameters of a given model based on given data. The problem consists mainly of finding a measure for the deviation between calculated and measured values, which has to be minimized to determine the parameters. Possible techniques for solving this problem are reviewed in section 2.3.3.

The last step of ***model confirmation or falsification*** consists of testing the model with data not used for model calibration. The most important point to remember here, is that a model is disproved if its results do not agree with experimental data to within a reasonable accuracy, whereas the agreement of model calculations with data does not prove its validity (Popper, 1982). Model confirmation and falsification is discussed in section 2.3.4.

2.2.3 Forecasting

The goal of model-based forecasting is to predict the future behavior of a system with the aid of a calibrated model.

Caswell (1976) distinguishes "models for understanding" and "models for prediction". He stresses the point that prediction is possible with calibrated empirical models without understanding the basic mechanisms of a system. Since this report mainly stresses the potential of models for scientific investigations, this subsection discusses the utility of "models for understanding" as tools for prediction. Such models

have to be found by the system identification procedure discussed in the previous subsection. This makes it evident that ***the reliability of forecasts is strongly dependent on model identification.*** Three types of errors for such forecasts can be distinguished (e.g. Beck, 1991):

- uncertainty of model structure,
- uncertainty of model parameters and initial state,
- uncertainty associated with external variables.

These three sources of uncertainty are discussed in more detail in section 2.4. Because a model of an environmental system only covers certain aspects of the system the uncertainty in the model structure is nearly impossible to estimate. For this reason, model predictions should always be treated very carefully in the environmental sciences.

2.3 System Identification

According to Popper (1982), the advance of scientific knowledge consists of the two main steps of stating a new theory or model as a conjecture and then trying to falsify this conjecture. In order to represent scientific progress, such a conjecture has to be conservative with respect to facts already explained by a previous theory, but revolutionary with respect to failures of old theories. The main points Popper (1982) makes are to reject the inductive nature of scientific discovery, to request the possibility of falsification as a criterion for admissible conjectures and to stress the deductive nature of attempts to falsify a conjecture. Popper (1982) does not discuss how reasonable conjectures should be obtained, because this step does not belong to the "logic of scientific discovery". ***Although there will never be a recipe for the process of obtaining new conjectures, the methods of system identification or model building try to give guidelines for the design, and techniques for the selection, of "reasonable" model structures.***

Descriptions of natural systems (including technical and laboratory systems containing living organisms) by mathematical models need drastic simplifications and can only cover certain aspects of real systems (e.g. Rose, 1987; Yozdis, 1989; Wissel 1989). As a consequence, such descriptions cannot be universal, but depend on the class of phenomena to be described (modelling objectives). Therefore, together with the more obvious information sources of *a priori* knowledge and experimental data, the following three ***information sources*** contribute to the model building process (Spriet and Vansteenkiste, 1982; Spriet,1985):

- *a priori* knowledge,
- experimental data,
- modelling objectives.

The task of system identification consists of making optimal use of this information in order to find the most adequate model. As already discussed in section 2.2.2, system identification is an iterative process consisting of the following steps:

- model structure evaluation,
- identifiability analysis of model parameters,
- parameter estimation,
- model confirmation or falsification.

Although these four steps are treated separately in the four subsections of this section, it is very important to note that they are strongly dependent on each other. For example, model structures with unidentifiable parameters are useless, and although *a priori* identifiability analysis is very useful, the estimation of accuracy and correlation of estimated parameters ultimately quantifies parameter identifiability. It is also very important to note that model building is an iterative process. Systematic deviations between values caculated using a calibrated model (with identified parameters) and measured values usually leads to a revision of the model structure, and therefore to a restart of the system identification loop summarized above.

There exists a large literature on system identification techniques. A survey can be found in one of the textbooks Norten (1986), Ljung (1987), Söderström and Stoica (1989) and Bohlin (1991) or in the Systems & Control Encyclopedia (Singh, 1987). The origin of system identification is in the field of control engineering and the majority of books and papers concentrate on linear or discrete time models. Surveys for the application of system identification methods to nonlinear systems are given by Mehra (1980) and Billings (1980). A review of the application of system identification techniques to water quality modelling is given by Beck (1987).

2.3.1 Model Structure Evaluation

Model structure evaluation consists of the following two steps:

- finding models or classes of models to be compared with data,
- deciding between competing model structures.

These two steps are discussed in more detail in the following paragraphs.

The first step, that of finding models or classes of models as conjectures to be tested with data, requires a creative analysis of the information sources discussed in the introduction to this section. This makes it impossible to give a systematic approach for solving this problem. Nevertheless, the following ***guiding principles for proposing model structures*** can be formulated (Spriet, 1985):

- physicality: the structure of a model should be related as directly as possible to the causal mechanisms acting in the system under investigation;
- characterizability: model structures defining competing model classes should lead to a well-defined decision process;
- identifiability: model parameters of a proposed model structure have to be identifiable using the available data; otherwise the model structure is useless.

In the environmental sciences, finding model structures is often a two stage process. In the first stage, preliminary models are suggested using the information sources "a priori knowledge" and "modelling objectives". Much interdisciplinary knowledge is required to make a reasonable suggestion for models at this stage. In a second stage of the process of finding model structures, systematic deviations between measurements and calculations performed with the models proposed in the first stage can lead to revisions of model structure. To avoid basing conclusions for model revisions

too much on current model parameter values, for this second stage, Hornberger and Spear (1981 and 1983) and Young (1983) developed a method for discriminating between important and insignificant model substructures with the aid of Monte Carlo simulations. The advantage of this method in comparison to conventional sensitivity analysis and parameter estimation is, that it can be applied in an early phase of an experiment, when only very few data are available.

The three most important ***techniques for deciding between competing model structures*** are:

- graphical or statistical searching for systematic deviations between calculations and measurements,
- quantitative measures of model adequacy,
- recursive parameter estimation.

These three techniques are reviewed briefly in the following paragraphs.

In most cases, graphical comparisons clearly show the existence or absence of systematic deviations between calculations and measurements. Such deviations can also be detected with the aid of statistical analyses such as residual plots, distribution or correlation tests of residuals, etc.. The advantage of such statistical measures over graphical methods is mainly the fact that they facilitate the partial automation of model structure evaluation.

It is evident that a quantitative measure of the difference between calculated and measured values is an important criterion for the adequacy of a model. Such a measure of quality of fit should be as small as possible. The problem of employing such a measure as a single criterion is that it is possible to make it smaller and smaller by increasing the complexity and the number of parameters of the model. Such a tendency to favor complex models cannot be accepted, because this leads to problems in uniquely identifying model parameters. For this reason, an additional criterion to promote simplicity is necessary. Thus, the criteria for defining model adequacy can be summarised as follows (Spriet, 1985):

- quality of fit: the model structure should be able to represent the measured data in a proper manner,
- parsimony: the model structure should be as simple as possible compatible with the requirement stated above.

In order to be able to assess these criteria objectively and thus to make it possible to automate the process of model discrimination, a quantitative measure of model adequacy is necessary. The first such measure was introduced and discussed in a series of papers by Akaike (1973, 1974, 1976, 1981 and more) and consists of a generalization of the maximum likelihood principle for the parameter estimation of a given model (cf. section 2.3.3). It is called the A, or Akaike, information criterion (AIC) and is formulated thus:

$$AIC(\mathbf{f}_{meas}) = -2\log\left(\max_{\mathbf{p}}\left(L(\mathbf{f}_{meas},\mathbf{p})\right)\right) + 2m \quad , \qquad (2.1a)$$

where **p** is the array of model parameters, $\mathbf{f}_{meas}$ the array of measured values, L the likelihood function of the model (probability density of the model at the measured values; cf. section 2.3.3) and m the number of parameters. Note that the first term on the right-hand side of equation (2.1a) is not non-dimensional because L has the dimension of the reciprocal of the product of the dimensions of the measurements (as it is a probability density). However, a change of units in the measurements results in L being multiplied by a constant factor, i.e. in a constant term being added to AIC. Such an additive term has no influence on model discrimination. Expression (2.1a) formulates the trade-off to be made between quality of fit and model complexity. The first term on the right hand side of equation (2.1a) increases with decreasing quality of fit, while the second term increases with increasing model complexity measured in terms of the number of parameters. The measure (2.1a) for model adequacy is very easy to use, since the first term corresponds to the usual criterion for maximum likelihood parameter estimation (cf. section 2.3.3). The procedure has to be performed as follows (Akaike, 1973; 1974; 1976; 1981). For all competing model structures, a maximum likelihood parameter estimation is performed. Then the model with the smallest value of AIC according to equation (2.1a) is chosen as the best model. The criterion (2.1a) has been criticized by various authors (e.g. Rissanen, 1976; Sawa, 1978; Schwarz, 1978). The main problems associated with criterion (2.1a) are the heuristic arguments for its justification, which have not been extended to a rigorous proof (Schwarz, 1978), and the ignorance of model structure with the exception of the number of parameters (Rissanen, 1976). For independent, identically distributed observations of linear models, Schwarz (1978) derived rigorously the alternative B information criterion

$$BIC(\mathbf{f}_{meas}) = -2\log\left(\max_{\mathbf{p}}\left(L(\mathbf{f}_{meas},\mathbf{p})\right)\right) + m\log(n) \quad , \qquad (2.1b)$$

where n is the number of measured data points. Equation (2.1b) is used in the same way as equation (2.1a). Criterion (2.1b) favors more simple models than does criterion (2.1a) if 8 or more measured data points are available. For large numbers of measured data points, the two procedures differ significantly from one another. The measures of model adequacy discussed above were originally developed for estimating the order of auto-regressive processes. Even more suggestions for such measures exist, and it is not yet clear which measure performs best when confronted with the more general problem of discrimination between nonlinear models such as those used in modelling environmental systems (Spriet, 1985).

In most cases, model parameters are assumed to be constant. In contrast to the usual *en bloc* parameter estimation, which compares the results of calculations performed using different, but fixed, parameters, recursive parameter estimation of time series data allows the parameters to change slowly in time (e.g. Young, 1983; Young, 1984). Recursive parameter estimation thus tests the model hypothesis of constant parameters and can therefore also be regarded as a model identification method (cf. section 2.3.3 on parameter estimation).

2.3.2 Identifiability Analysis of Parameters

Identifiability analysis treats the problem of the uniqueness and accuracy of the determination of model and initial state parameters for a given experiment. It was put on a formal basis by Bellmann and Aström (1970). An excellent review is given by Godfrey and DiStefano (1985, 1987) (see also Walter, 1982, 1987). Two identifiability problems can be distinguished: theoretical identifiability analysis treats the identifiability problem for an input-output experiment with perfect data, whereas practical identifiability analysis treats the problem of the estimation of parameter accuracy with real data for parameters which are known to be theoretically identifiable.

Theoretical Identifiability Analysis

Theoretical identifiability analysis treats the problem of the uniqueness of the determination of model parameters resulting from a given input-output experiment with perfect data. It is also called ***structural, deterministic or a priori identifiability analysis***. The last name refers to the fact that such an analysis can (and should) be performed before the experiment is carried out. For a given model, given initial conditions and external variables and given ideal measurements (continuously measured in time), according to Godfrey and DiStefano (1985, 1987), a parameter is called

- uniquely (globally) identifiable: if there exists a unique solution for the parameter,
- locally identifiable: if there exists a finite number (larger than or equal to one) of parameter values,
- unidentifiable: if there exists an infinite number of parameter values,

which make the model (exactly) reproduce the measurements. In the case of the existence of unidentifiable parameters, there exists a local submanifold in the parameter space which leads to (exactly) the same model behavior. In this case, it is important to know which combinations of parameters are identifiable (e.g. if only the product of two parameters is used in a model, the parameters are not separately identifiable, but the product may be identifiable).

Whereas there exist several methods of identifiability analysis for models formulated as systems of linear differential equations, there is only one universal technique applicable to nonlinear differential equations (Godfrey and DiStefano, 1985 and 1987). This technique, due to Pohjanpalo (1978), is based on a Taylor series expansion of the measured variables with respect to time. The coefficients of the power series contain the model parameters; the decision whether these parameters can be determined from the known Taylor series coefficients is an algebraic problem. If the algebraic equations can be solved for the parameters, they are identifiable. An example of such an identifiability analysis for a Monod or Michaelis-Menten type model, which is often used in ecosystem modeling (e.g. for modelling microbial

growth), is given by Godfrey and Fitch (1984) (see also Godfrey and DiStefano, 1985, 1987). ***The problem of theoretical identifiability analysis by Taylor series expansion of the measured variables is that it leads to a cumbersome computational effort*** for "all but the simplest problems" (Beck, 1987). This problem can partially be overcome by the use of computer algebra programs (Lecourtier and Raksanyi, 1987).

Practical Identifiability Analysis

Practical identifiability analysis treats the problem of parameter identification in the presence of noisy measurements. It is also called ***numerical or a posteriori identifiability analysis***. It is evident, that practical identifiability of parameters requires their theoretical identifiability. ***Since practical identifiability is mainly a problem of the estimation of parameter uncertainty, it is not an objective characteristic of a model for a given experimental situation, but depends instead on the values of the measured data and on the desired accuracy of the parameters***.

There are two main techniques for practical identifiability analysis: sensitivity analysis (linear and nonlinear) and parameter covariance estimation. Because parameter covariance estimation is treated in the next subsection together with parameter estimation, only sensitivity analysis is briefly discussed here.

Linear sensitivity analysis consists of calculating a linear approximation to the change in a variable caused by a given change in a parameter. Depending on whether absolute or relative measures of the variable and of the parameter are used, the following four sensitivity functions can be distinguished:

$$\delta_{f,p}^{a,a} = \frac{\partial f}{\partial p} \quad , \tag{2.2a}$$

$$\delta_{f,p}^{r,a} = \frac{1}{f}\frac{\partial f}{\partial p} \quad , \tag{2.2b}$$

$$\delta_{f,p}^{a,r} = p\frac{\partial f}{\partial p} \quad , \tag{2.2c}$$

$$\delta_{f,p}^{r,r} = \frac{p}{f}\frac{\partial f}{\partial p} \quad . \tag{2.2d}$$

Sensitivity function (2.2a) gives the absolute change in variable f per unit change in parameter p, function (2.2b) the relative change in f per unit change in p, function (2.2c) the absolute change in f for a 100 % change in p, and function (2.2d) the relative change in f for a 100 % change in p (all of these changes are calculated as linear approximations only!). The two most often used sensitivity functions are (2.2c)

and (2.2d), because the units of these functions do not depend on the units of the parameter. This makes the comparison of the sensitivity of a variable to different parameters possible. The use of the sensitivity function (2.2d) is recommended if relative changes in f are of interest. Since this function is non-dimensional, comparison of the effect on different variables f can be compared. Absolute changes in f, as provided by the sensitivity function (2.2c), are more useful if f is close to zero somewhere within the domain of interest. In this case, the division by f in equation (2.2d) arbitrarily increases the relative sensitivity.

The larger the values of the sensitivity functions, the more accurately a single parameter can be identified. In the case of several parameters, the sensitivity functions of the parameters as functions of the independent variable of the measurements (usually time, but a space coordinate is also possible if spatial profiles are used as targets for the fit) have to be linearly independent. Otherwise, the parameters are not individually identifiable, because a change in one parameter can be compensated for by changes in the other parameters. The more different the patterns of the sensitivity functions are (within the ranges of the independent variable for which measured data are available), the better the parameters can be identified. Examples of the application of linear sensitivity analysis to environmental systems are given by Holmberg (1982), who applies linear sensitivity analysis to discuss the practical identifiability of Michaelis-Menten type microbial growth models and by Cirpka et al. (1993), who use linear sensitivity analysis to discuss the physical mechanisms of gas exchange at river cascades.

Nonlinear sensitivity analysis is based on the calculation of the probability distributions of calculated variables from the probability distributions of the parameters. This is done with the aid of Monte Carlo simulation (e.g. Rubinstein, 1981). It is evident that this analysis, which takes the nonlinearity of the model fully into account, gives much better information than does linear sensitivity analysis as described above. The disadvantage of this method is that it requires a large computational effort and (at least approximate) knowledge of the probability distributions of the parameters. Such distributions can be estimated using the methods described by Tiwari and Hobbie (1976b). Due to the large uncertainty generally present in environmental systems, and to the nonlinear nature of these systems on the scale of the uncertainty (Young, 1983), nonlinear sensitivity analysis is preferable to linear sensitivity analysis. Nevertheless, the much less troublesome method of linear sensitivity analysis is in many cases accurate enough to find the major dependences of a model and to understand identifiability problems detected during the parameter estimation process.

2.3.3 Parameter Estimation

Parameter estimation consists of determining the "optimal" values of the parameters of a given model with the aid of measured data. Although this procedure uses a given model structure, it is not completely independent of model structure identification, because the model may degenerate to a simpler structure for particular values of some parameters. Since the initial state of a simulation, the boundary conditions and the external variables can also be formulated with the aid of parameters, all these parameters, together with model parameters, can be combined to yield a single array of parameters to be estimated simultaneously using the same estimation technique.

Type of Estimator

There are four important conventional techniques which can be used for parameter estimation (e.g. Eykhoff, 1974; Beck, 1987):

- Bayesian estimation,
- maximum likelihood estimation,
- weighted least squares estimation,
- least squares estimation.

These four methods are listed in decreasing order of the amount of information that has to be provided by the user of the method, or, equivalently, in increasing order of the number of a priori assumptions already included in the method. For the most complicated case of a Bayesian estimation, the probability distribution of the parameters and the conditional probability distribution of the measurements for given parameter values have to be parameterized, whereas the simplest case of least squares estimation can be performed without any extrinsic information. In fact, as discussed below, weighted least squares estimation and least squares estimation are special cases of the maximum likelihood method in which the measurements are assumed to be uncorrelated and normally distributed. Since distributions of real data are never known exactly, all of these parametric methods are based on the implicit assumption that small deviations from the assumed form of the distributions will result in small errors in the final results. Practical experience has shown, however, that this is not always the case. This experience led to the development of methods of

- robust estimation,

which fulfill the above-mentioned criterion (Hampel, 1973; Huber, 1981; Hampel et al., 1986; Rousseeuw and Leroy, 1987; Birkes and Dodge, 1993). The basic ideas of all five parameter estimation techniques are reviewed briefly in the following paragraphs.

Bayesian Estimation

Bayesian estimation treats both measurements and model parameters as random variables. If an *a priori* probability density $g(\mathbf{p})$ for the occurrence of the parameter values $\mathbf{p} = (p_1, \ldots, p_m)$ and the conditional probability density $g(\mathbf{f}_{meas}|\mathbf{p})$ of the model for measuring the values $\mathbf{f}_{meas} = (f_{meas,1}, \ldots, f_{meas,n})$ for given parameter values $\mathbf{p}$ are known, the probability density of the parameters for given values of the measurements can be written as

$$g(\mathbf{p}|\mathbf{f}_{meas}) = \frac{g(\mathbf{f}_{meas}|\mathbf{p})\, g(\mathbf{p})}{g(\mathbf{f}_{meas})} \tag{2.3}$$

(Bayes' rule). Note that equation (2.3) does not directly specify estimates of the parameters, but it yields a complete description of the distribution of parameter values for given measurements. Additional assumptions are necessary for the choice of parameter estimates $\tilde{\mathbf{p}}$ (e.g. taking the expectation values of the probability density given by equation (2.3)). The central idea of Bayesian estimation is to update prior information on the distribution of parameters by taking measured data into account.

Maximum Likelihood Estimation

In contrast to Bayesian estimation, maximum likelihood estimation treats the parameters not as random variables, but as constant parameters of the distributions of the measurements. Maximum likelihood estimation consists of maximizing the so-called likelihood function, L, which is the probability density of a model for the occurrence of the measurements for given parameters:

$$L(\mathbf{f}_{meas},\mathbf{p}) = g(\mathbf{f}_{meas},\mathbf{p}) \quad . \tag{2.4}$$

For given measurements $\mathbf{f}_{meas}$, the maximum likelihood estimates $\tilde{\mathbf{p}}(\mathbf{f}_{meas})$ of the parameters are those values of $\mathbf{p}$ for which the likelihood function L, given by equation (2.4), has its maximum:

$$\tilde{\mathbf{p}}(\mathbf{f}_{meas}): \quad L(\mathbf{f}_{meas},\tilde{\mathbf{p}}) = \text{maximum (as a function of } \tilde{\mathbf{p}}) \quad . \tag{2.5}$$

In many practical applications of the maximum likelihood method (assuming differentiability of the likelihood function), equation (2.5) is replaced by

$$\tilde{\mathbf{p}}(\mathbf{f}_{meas}): \quad \frac{\partial L}{\partial \mathbf{p}}\left(\mathbf{f}_{meas},\tilde{\mathbf{p}}(\mathbf{f}_{meas})\right) = 0 \quad . \tag{2.6}$$

This expression defines $\tilde{\mathbf{p}}$ as an implicit function (m equations in the m unknowns $p_1, \ldots, p_m$). Although the parameters $\mathbf{p}$ are not random variables, their estimates given by the function $\tilde{\mathbf{p}}$ are uncertain due to the dependence on the random variables $\mathbf{f}_{meas}$. The uncertainty of the estimates $\tilde{\mathbf{p}}$ can be estimated from the uncertainty of the measurements $\mathbf{f}_{meas}$ by applying the general linear error propagation formula of a function $\mathbf{f}(\mathbf{p})$

$$\mathrm{Cov}(\mathbf{f}) = \left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}}\right) \mathrm{Cov}(\mathbf{p}) \left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}}\right)^{T} \tag{2.7}$$

to the estimation function $\tilde{\mathbf{p}}$

$$\mathrm{Cov}(\tilde{\mathbf{p}}) = \left(\frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{f}_{meas}}\right) \mathrm{Cov}(\mathbf{f}_{meas}) \left(\frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{f}_{meas}}\right)^{T} , \tag{2.8}$$

where Cov is the variance-covariance matrix of the specified argument. The derivatives $\partial\tilde{\mathbf{p}}/\partial\mathbf{f}_{meas}$ can be obtained by differentiating equation (2.6), defining $\tilde{\mathbf{p}}$ with respect to $\mathbf{f}_{meas}$:

$$\frac{\partial^2 L}{\partial \mathbf{p}\, \partial \mathbf{f}_{meas}} (\mathbf{f}_{meas}, \tilde{\mathbf{p}}) + \frac{\partial^2 L}{\partial \mathbf{p}^2} (\mathbf{f}_{meas}, \tilde{\mathbf{p}}) \cdot \frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{f}_{meas}} = 0 . \tag{2.9}$$

If the matrix $\partial^2 L/\partial \mathbf{p}^2$ can be inverted, this equation can be solved for $\partial\tilde{\mathbf{p}}/\partial\mathbf{f}_{meas}$:

$$\frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{f}_{meas}} = -\left(\frac{\partial^2 L}{\partial \mathbf{p}^2} (\mathbf{f}_{meas}, \tilde{\mathbf{p}})\right)^{-1} \cdot \frac{\partial^2 L}{\partial \mathbf{p}\, \partial \mathbf{f}_{meas}} (\mathbf{f}_{meas}, \tilde{\mathbf{p}}) . \tag{2.10}$$

This equation completes the estimate for of the variance-covariance matrix of the estimated parameters given in equation (2.8).

Weighted Least-Squares Estimation

If the probability distributions of the measurements are assumed to be uncorrelated normal distributions, the likelihood function $L(\mathbf{f}_{meas}, \mathbf{p})$ is given as the product of these normal distributions:

$$\begin{aligned} L(\mathbf{f}_{meas}, \mathbf{p}) &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} \frac{1}{\sigma_{meas,i}} \exp\left(-\frac{1}{2} \left(\frac{f_{meas,i} - f_i(\mathbf{p})}{\sigma_{meas,i}} \right)^2 \right) \\ &= \frac{1}{\sqrt{2\pi}^{n}} \cdot \prod_{i=1}^{n} \frac{1}{\sigma_{meas,i}} \cdot \exp\left(-\frac{1}{2} \sum_{i=1}^{n} \left(\frac{f_{meas,i} - f_i(\mathbf{p})}{\sigma_{meas,i}} \right)^2 \right) , \end{aligned} \tag{2.11}$$

where $f_i(\mathbf{p})$ is the calculated value of the model corresponding to $f_{meas,i}$ using the parameters $\mathbf{p}$, and $\sigma_{meas,i}$ is the (estimated) standard deviation of $f_{meas,i}$. Maximizing the likelihood function (2.11) for given values $f_{meas,i}$ as a function of $\mathbf{p}$ is equivalent to minimizing the function

$$\chi^2(\mathbf{p}) = \sum_{i=1}^{n} \left(\frac{f_{meas,i} - f_i(\mathbf{p})}{\sigma_{meas,i}} \right)^2 . \tag{2.12}$$

The values $\tilde{\mathbf{p}}$ of $\mathbf{p}$ corresponding to the minimum of the expression (2.12) are called weighted least-squares estimates. The parameter variance-covariance matrix $\mathrm{Cov}(\tilde{\mathbf{p}})$ can be estimated with the aid of equations (2.8) and (2.10). The derivatives of the likelihood function (2.11) are given by

$$\frac{\partial^2 L}{\partial f_{meas,i}\,\partial p_j} = L\,\frac{1}{\sigma_{meas,i}{}^2}\,\frac{\partial f_i}{\partial p_j} \tag{2.13}$$

and

$$\frac{\partial^2 L}{\partial p_i\,\partial p_j} \approx -L\sum_{k=1}^{n}\frac{1}{\sigma_{meas,k}{}^2}\,\frac{\partial f_k}{\partial p_i}\,\frac{\partial f_k}{\partial p_j}\quad , \tag{2.14}$$

where second derivatives of $\mathbf{f}$ with respect to $\mathbf{p}$ are neglected in the second equation (in accordance with the use of the linear error propagation formula). Substitution of these derivatives into equation (2.10) yields

$$\frac{\partial \tilde{p}_i}{\partial f_{meas,j}} \approx \left(\sum_{k=1}^{n}\frac{1}{\sigma_{meas,k}{}^2}\,\frac{\partial f_k}{\partial p_i}\,\frac{\partial f_k}{\partial p_j}\right)^{-1}\cdot\left(\frac{1}{\sigma_{meas,i}}\,\frac{\partial f_i}{\partial p_j}\right)\quad . \tag{2.15}$$

Substituting this expression and

$$\mathrm{Cov}(\mathbf{f}_{meas}) = \begin{pmatrix}\sigma_{meas,1}{}^2 & \cdot & 0\\ \cdot & \cdot & \cdot\\ 0 & \cdot & \sigma_{meas,n}{}^2\end{pmatrix} \tag{2.16}$$

into equation (2.8) yields the desired estimate for the variance-covariance matrix of the estimated parameters:

$$\mathrm{Cov}(\tilde{\mathbf{p}}) = \left(\left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}}\right)^{T}\begin{pmatrix}\frac{1}{\sigma_{meas,1}{}^2} & \cdot & 0\\ \cdot & \cdot & \cdot\\ 0 & \cdot & \frac{1}{\sigma_{meas,n}{}^2}\end{pmatrix}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}}\right)\right)^{-1}\quad . \tag{2.17}$$

This equation expresses the variance-covariance matrix of the estimated parameters as a function of the given standard deviations of the measurements. Usually, parameter estimation programs only use relative magnitudes of the given standard deviations for weighting the measurements, but replace the absolute value by the estimated standard deviation. In this case, the formula given above must be replaced by

$$\mathrm{Cov}(\tilde{\mathbf{p}}) = \frac{\chi^2}{n\text{-}m}\left(\left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}}\right)^{T}\begin{pmatrix}\frac{1}{\sigma_{meas,1}{}^2} & \cdot & 0\\ \cdot & \cdot & \cdot\\ 0 & \cdot & \frac{1}{\sigma_{meas,n}{}^2}\end{pmatrix}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}}\right)\right)^{-1}\quad , \tag{2.18}$$

where $\chi^2/(n-m)$ is the estimated variance of the data. This factor was originally introduced by Rosenfeld et al. (1967); it has no rigorous foundation. It leads to more reasonable (larger) estimates of errors in cases in which the standard deviations of the measurements do not take all sources of error into account. These cases can easily be detected by a failing chi square test. The danger involved in employing equation (2.18) instead of equation (2.17) is that systematic errors may be treated as statistical errors (cf. Brandt, 1992). In the rare case in which χ^2 is smaller than n-m, the factor $\chi^2/(n-m)$ reduces the estimated values in the variance-covariance matrix.

Least-Squares Estimation

If not even the standard deviations $\sigma_{meas,i}$ of the measurements are known, all of them are assumed to be of equal size. In this case, dropping the common factor $1/\sigma^2$ from the sum in equation (2.12) leads to the expression

$$SS = \sum_{i=1}^{n} \left(f_{meas,i} - f_i(\mathbf{p})\right)^2 \qquad (2.19)$$

to be minimized for the simple least-squares estimation. Equation (2.18) then simplifies to

$$Cov(\tilde{\mathbf{p}}) = \frac{SS}{n-m} \left(\left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}}\right)^T \left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}}\right) \right)^{-1} . \qquad (2.20)$$

It is not possible to apply equation (2.17) in this case, because it is necessary to estimate the variance of the measurements (cf. comment between equations (2.17) and (2.18)).

Robust Estimation

The four conventional parameter estimation techniques described above are derived using assumptions concerning the form of the distributions of the measured variables. If real data violate these assumptions slightly, the estimation procedures may give incorrect results. This problem can be diminished by the application robust statistical methods, which, without giving up the advantages of parametric statistics, admit the possibility of alternative probability distributions near to the proposed parametric model or allow a subset of the measurements to belong to an arbitrary distribution (e.g. Huber, 1981; Hampel et al., 1986; Rousseeuw and Leroy, 1987). ***The application of robust estimation techniques makes the results much less dependent on slight violations of the parameterized probability distributions by the data*** (e.g. existence of outliers). A similar improvement in conventional estimation techniques can be achieved by removing outliers "by hand", e.g. by graphical inspection of the data. It is important to note that a category of outlier called "leverage points" cannot be detected by a simple residual analysis (cf. any of the above-mentioned books). The main advantages of robust methods is that they make automation of outlier detection possible and that they also work in the case of several

independent variables, where graphical methods fail. One of the simplest robust parameter estimation methods is the minimization of the median of the squares of the residuals instead of their sum, or, more efficiently, changing the sums in equations (2.12) and (2.19) to include only the smaller half of the squares of the residuals (Rousseeuw and Leroy, 1987). Robust parameter estimation methods for linear models are already available in the form of computer program packages (Marazzi, 1993). The application of such techniques to environmental systems is very desirable, but an extension of the methods to nonlinear models is necessary. The only disadvantage of robust parameter estimation methods compared to conventional techniques is the increase in computation time involved.

Estimation Procedure

If the measured data are in time series form, two different types of estimation can be performed:

- *en bloc* estimation,
- recursive estimation.

En bloc estimation is the straightforward way of applying the estimators (2.3) to (2.19) to time series data. This procedure consists of performing a simulation with constant parameter values over the whole time interval containing measurements, and locating the maximum of equation (2.4), or the minimum of equations (2.12) or (2.19), by repeating such calculations for different sets of parameters.

An alternative to *en bloc* estimation is recursive estimation, where the estimators (2.3) to (2.19) are recursively applied to subsections of the measured time series. Such subsections can be of increasing length or they can be selected with the aid of a window moving along the time series. This procedure and its extensions (Kalman-filter, recursive instrumental variable method, etc.) can be used for estimating time-varying parameters or for testing the model assumption of time-independence of the parameters (Young, 1984; Beck, 1983). For this reason, recursive parameter estimation can also be classified as a method for system identification. The test of time-independence of the model parameters can also be performed using *en bloc* estimation by splitting the time series into subsections and applying the *en bloc* procedure to the subsections or by using a time-dependent reparameterization for the original constant parameters.

Alternative Approach for "Badly Defined" Systems

The statistical methods described so far can only be applied if a reasonable amount of measured data is available. An alternative approach, which can be applied in the case of bad resolution of data or even using semi-quantitative information on system behavior, is the Monte Carlo filtering technique. This method was originally devel-

oped to make model structure evaluation less dependent on currently used parameter values (Hornberger and Spear, 1981 and 1983; Young, 1983), and was later applied for the determination of reasonable sets of parameter values for given models (Hornberger and Cosby, 1985; Rose et al. 1991). The basic idea of the method is to fix ranges of the values of observables which characterize reasonable system behavior, to perform Monte Carlo simulations with the model, and to select sets of parameters which lead to the desired behavior. This method does not lead to a unique set of parameter values, as in the case of the methods described so far, but the sets of parameter values found to be compatible with the data can be used for predictions, which then have to be statistically evaluated.

2.3.4 Model Confirmation or Falsification

The last step in system identification consists of testing the model with independent data sets (data sets not used for model identification and calibration). This step is usually called model "validation" or "verification" (e.g. Thomann, 1982). It is, however, important to note that ***the absence of significant deviations between model calculations and measurements only proves that the model assumptions are compatible with system behavior. Thus, in a strict sense, model "validation" or "verification" is impossible*** (Popper, 1982; Caswell, 1976; Reckhow and Chapra, 1983). Model confirmation tests only "confirm" or "corroborate" a model (Popper, 1982). Because significant deviations between model calculations and measurements disprove a model, the goal of model confirmation should be to attempt to refute the model (Popper, 1982). Then, the confidence in the model assumptions increases as the model passes more and more severe tests. It is very desirable to quantify the result of such hypothesis tests with the aid of statistical criteria (Thomann, 1982; Reckhow and Chapra, 1983; Reckhow et al., 1990), but it is important to note that such rigorous tests do not change the fundamental problem of the impossibility of model verification. Accepting a hypothesis never means that the hypothesis is proved, but only that it was not possible to contradict the hypothesis with the aid of the measurements used.

2.4 Uncertainty Analysis of Simulations

Model structure identification and parameter estimation lead to a calibrated model for the description of the behavior of the system under investigation. If such a model is used to predict future behavior in order to support management decisions, an estimate of the uncertainty of the predictions is very important. O'Neill and Gardner (1979) wrote: "***In spite of the intensive activity in model development and application, relatively little attention has been given to examining the magnitude of uncertainty associated with model predictions***". Since then, more attention has been payed to this subject, but even now, predictions based on the results of simulations are very often used without consideration of their uncertainty. Practically, in water quality management, this often leads to the paradoxical situation that simulation results are accepted without questioning their assumptions, whereas simple estimates are doubted even though they may be based on realistic assumptions.

It is the goal of this section to analyze the sources of uncertainty of ecological model simulations and to review briefly the possibilities of estimating the magnitude of the uncertainties. Three sources of uncertainty are distinguished (Beck, 1991):

- uncertainty in model structure,
- uncertainty in parameter values and initial state,
- uncertainty associated with external variables.

Each of these sources of uncertainty is treated briefly in a subsection of this section. More extensive reviews are given by O'Neill and Gardner (1979) and Beck (1983, 1987).

2.4.1 Uncertainty in Model Structure

As discussed in the introduction to section 2.3, models of environmental systems are drastic simplifications describing only certain aspects of the system. These simplifications limit the validity of the model and are a major source of error. The most important sources of error due to model formulation lie in:

- the consideration of important variables and processes,
- the formulation of processes,
- the resolution of variability in space and time.

It is often difficult to decide whether an incorrect simulation result is due to incorrect assumptions concerning the important processes to be considered in the model

(wrong conceptual model), or to incorrect formulation of the processes (wrong mathematical model). Errors arising due to inadequate resolution of spatial variability are discussed by O'Neill and Gardner (1979); errors due to modelling natural temporal variability using constant model structures and parameters are discussed by O'Neill (1979) and Kremer (1983). In any case, estimation of the reliability of a model of an environmental system requires much interdisciplinary knowledge. ***Uncertainty in model structure is by far the most difficult contribution to uncertainty in model predictions to quantify***, because different model structures may lead to very similar results for some environmental conditions, but to different results for other conditions. Attempts to estimate the uncertainty in model structure are very closely related to the model structure evaluation process discussed in section 2.3.1: comparing predictions of competing model structures tested during the identification process may help in estimating this type of uncertainty. The most important way of avoiding an extreme increase in prediction errors is not to leave the range of validity of the model tested in the model confirmation procedure.

2.4.2 Uncertainty in Parameter Values and Initial State

If a model correctly represents the structure of a system, natural variability and finite measurement accuracy are the causes of uncertainty in model parameters (including parameters characterizing the initial state) estimated with one of the methods described in section 2.3.3. In contrast to the uncertainty in model structure, which is difficult to estimate due to the delicate model identification process, quantification of the uncertainty in simulation results due to uncertainty in the parameters for a given model structure is a purely mathematical problem, that of error propagation. Nevertheless, since the numerical solution to this problem is computationally very intensive, approximate techniques are of importance. There are two main techniques used for estimating the uncertainty in simulations due to uncertainty in the model parameters:

- linear error propagation,
- Monte Carlo simulation.

Linear error propagation assumes the error distributions of the parameters to be small compared to the scale on which the nonlinear effects of the model become important. Then, assuming uncorrelated parameters, the standard deviation σ_f of a simulated variable f, according to equation (2.7), is given by

$$\sigma_f = \sqrt{\sum_{i=1}^{m} \left(\frac{\partial f}{\partial p_i}\right)^2 \sigma_{p_i}^2} \quad , \tag{2.21}$$

where p_i (i=1,...,m) are the uncertain model parameters and σ_{p_i} their standard deviations (e.g. Bevington, 1969). In addition to the total standard deviation σ_f according to equation (2.21), it is often useful to compare the individual contributions of different parameters

$$\delta_{f,pi}^{err} = \frac{\partial f}{\partial p_i} \sigma_{p_i} \tag{2.22}$$

in order to detect major sources of uncertainty. The main problem involved in applying the linear error propagation formula (2.21) is the fact that nonlinear effects are neglected. Since models for ecosystems are in many cases highly nonlinear on the scale of the relatively large errors in the parameters, this assumption is often violated. Because of this problem, some authors reject the linearized error propagation formula (2.21), whereas others accept it at least as an order of magnitude estimator. It is evident that such statements cannot be made universally, but that they depend on the system under investigation and on the accuracy of the parameters.

If exact estimates of simulation uncertainty due to uncertain parameters are desired, and if the computational power for performing hundreds or thousands of simulations (depending on the number of parameters) is available, the probability distribution of simulation results can be approximately calculated from the distributions of the parameters by ***Monte Carlo simulation*** (e.g. Rubinstein, 1981). Such a Monte Carlo simulation evidently gives a much better uncertainty estimate than the linear error propagation formula (2.21) (Gardner and O'Neill, 1983). Tiwari and Hobbie (1976a) demonstrated that the stochastic means of a Monte Carlo simulation can deviate significantly from a deterministic calculation using the mean values of the parameters. Such behavior is typical for "badly defined" nonlinear systems, which are in turn typical for models of environmental systems (Young, 1983). Estimates for probability distributions of parameters needed for Monte Carlo simulations can be obtained using the technique described by Tiwari and Hobbie (1976b).

As argued by Kremer (1983), ***the uncertainty analysis of deterministic models often overpredicts the uncertainty of results compared to stochastic modelling with similar parameter uncertainty.*** This is due to the fact that in deterministic simulations, the parameters remain at their initial, possibly extreme values during time evolution, whereas in stochastic models, averaging occurs already even for a single simulation. This basic criticism applies to both linear and nonlinear uncertainty analysis of deterministic models.

2.4.3 Uncertainty Associated with External Variables

External variables describe the influence of the environment on the system under investigation. Uncertainty due to random changes in the values of external variables can be treated in the same way as uncertainty in model parameters as described in section 2.4.2. Apart from this statistical uncertainty, there may be a lack of knowledge of the future temporal evolution of the mean values of such external variables. ***Such uncertainty can best be estimated by the calculation of several "reasonable" scenarios*** assumed for the behavior of the external variables. The meaning of "reasonable" depends here not on the system described by the model, but on its surroundings, which were not taken into account mechanistically in the model identification process. Therefore, the model identification process does not help in estimating this type of uncertainty.

3 Computer Programs for Model Application

As discussed in the previous chapter, mathematical models can be used for planning measurements, for system identification and for forecasting the future behavior of a system. System identification is particularly important because it helps in obtaining knowledge of the mechanisms acting within a system, which is also of use in the other two areas of model application.

The majority of mathematical models, as well as the steps of system identification, are so complicated, that computer programs are required for their application. A lot of such programs have been developed. In the first section of this chapter, an attempt is made to classify these programs into three main categories. Although this classification is not strict, it correctly reflects the main groups of existing software.

In the second section of this chapter, it is argued that, for environmental system identification, a more universal program is required to perform tasks belonging to all of the categories discussed in the first section. The requirements of such a program are stated. The concepts underlying the implementation of a program that takes a first step in the direction of such a more universal identification and simulation tool for a class of aquatic systems are discussed in chapters 4 - 7.

3.1 Classification of Programs

Computer programs that can be used for environmental simulation and system identification can roughly be classified into the following three categories:

- general purpose simulation software,
- conventional environmental simulation programs,
- tools for model identification.

This classification is not strict, because there exist programs covering tasks belonging to more than one of these categories. Nevertheless, this classification correctly characterizes the majority of currently available programs. In this section, the advantages and disadvantages of the programs in each of these three categories with respect to environmental system identification are briefly discussed.

3.1.1 General Purpose Simulation Packages

Mathematical models of environmental systems are usually formulated as systems of differential and algebraic equations. These equations can be solved numerically using general purpose simulation software. Three levels of such software can be distinguished:

- programming languages in combination with mathematical libraries,
- simulation languages,
- interactive simulation tools.

On the lowest level, it is evident that, with the use of programming languages, any task can be implemented in a computer program. In order to limit the expense involved, and to avoid reinventing the wheel, for this approach it is strongly recommended that mathematical libraries such as LINPACK (Dongarra et al., 1979), ODEPACK (Hindmarsh, 1983) and DASSL (Petzold, 1983), be used for the solution of systems of linear algebraic equations, ordinary differential equations, and differential-algebraic systems of equations, respectively; or that commercially available mathematical libraries such as IMSL (1993) or NAG (1993) be employed. While such libraries allow their users to implement systems of ordinary differential equations easily, there is only marginal support for partial differential equations. For this reason, program packages for the numerical solution of partial differential equations were developed. These packages provide routines for space discretization and use some of the above-mentioned programs for the integration of the resulting system of

ordinary differential equations (method of lines). Two widely used packages of this type are FORSIM VI (Carver et al., 1978) and DSS/2 (Schiesser, 1976, 1991, 1994). These packages require the user to write a FORTRAN subroutine for the definition of the equations, of initial and boundary conditions and for processing results. Most of the other libraries are also written in FORTRAN, but some of them are now being translated to C.

Because the use of mathematical libraries and program packages is not easy, simulation languages were developed with the aim of facilitating the specification of ordinary and, in some cases, of partial differential equations. Such languages require many fewer statements for the specification of the same problem than do the programs described in the previous paragraph. In most implementations, the underlying mathematical techniques are the same. An example of a widely spread simulation language is ACSL (e.g. Breitenecker et al., 1993).

The two classes of general purpose simulation software described above have been in use for more than a decade. They were originally developed and used as batch jobs in large computing centers and were later transferred to workstations and PC's. More recently, the increased availability of powerful workstations and the increase in PC computing power made the development and use of interactive simulation tools possible. Such tools are based on the same mathematical techniques, but they represent an important step forward as far as user-friendliness is concerned by providing an interactive user interface and graphical data processing. An important representative of this class of programs is MATLAB (MathWorks, 1993), which, together with its toolboxes (also for system identification), is a powerful simulation tool, at least for systems of ordinary differential equations.

The programs falling in the category of general purpose simulation software described in this subsection have the advantage of giving the user a large degree of freedom in model formulation (a predecessor of the biofilm model discussed in section 4.3.3 was implemented using FORSIM VI by Wanner and Gujer, 1986). The main disadvantages of these programs are that they require the user to be mathematically adept and to be capable of the abstract formulation of mathematical models, and that a lot of programming is necessary in order to organize input and output and to facilitate changes in model structure. Furthermore, most of these programs do not support model identification and the estimation of predictive uncertainty; in addition, those parts of a model, such as the hydraulics part, which are well-known and understood, still have to be implemented by the user. In recent years, interactive simulation tools have started to eliminate some of these disadvantages. Out of the programs in the category discussed in this subsection, these tools are certainly the most recommendable programs for direct application in environmental models consisting of ordinary differential equations. For models consisting of partial differential equations, simulation languages for partial differential equations are simpler to use, and for programmers implementing a simulation program, the use of public domain mathematical libraries may also remain attractive.

3.1.2 Conventional Environmental Simulation Programs

Most conventional environmental simulation programs (e.g. Ambrose and Barnwell, 1989; Zanetti, 1992) concentrate on a given environmental system and implement a specific model of relevant processes. The only way the user can influence such a program is by specifying the values of model parameters or by selecting a submodel out of a given set of submodels. The trend in the development of models implemented in conventional environmental simulation programs increasingly lies in the direction of models which consider more and more state variables and transformation processes. The reasons for this trend are concerned mainly with the availability of computational resources and with the assumption that the more detail taken into account, the better the model will be. This approach can be called into question because of the increasing difficulty of unambigously identifying the parameters of such models (Beck, 1983; Young, 1983). It is very important to note that such reductionistic models cannot be identified soley on the basis of the data from a given field study, but that they use additional *a priori* knowledge from laboratory experiments and other, similar, field studies. The correct combination of *a priori* knowledge with new data complicates the model calibration process in such a case. For this reason, expert systems have been developed to support the user of such a program in calibrating the model for the system under investigation (e.g. Barnwell et al., 1989; Baffaut and Delleur, 1989; Baffaut and Delleur, 1990). Such expert systems are not of general utility; they can only be used to calibrate the simulation programs for which they are designed (there is no sharp line between a good context-sensitive help system and such an expert system).

The main advantage of conventional environmental simulation programs is that, due to their specific nature and the inclusion of a graphical capability they are much easier to use than the programs described in the previous section. Furthermore, such a program is much easier to make numerically efficient than is a more universal program. The main disadvantage is the lack of freedom to modify the model on which the program is based. This makes it impossible to carry out a system identification process as described in section 2.3. Furthermore, these programs usually do not assist the user in calibrating the model with the aid of measured data, and, in most cases, also do not contain features for estimating predictive uncertainty.

3.1.3 Tools for Model Identification

In contrast to the programs discussed in sections 3.1.1 and 3.1.2, where model identification tools are usually absent, the programs in this class are specialized for

model identification, as described in section 2.3. In most cases, these programs also provide tools for model confirmation or "validation" (cf. section 2.3.4), and sometimes also for estimating predictive uncertainty. A survey of such programs is given by Jamshidi and Herget (1985). Newer development can be studied in the proceedings of the IFAC symposia on "Identification and System Parameter Estimation", which take place every three years (e.g. Haest et al., 1988; Wernstedt et al., 1992). The modern trend of these programs is to combine identification packages with expert systems which assist the user in performing the identification process successfully (Meier zu Fahrwig and Unbehauen, 1992).

The advantage of programs belonging to the category of model identification tools is that they are able to compare a given class of models automatically, and that they support the user in detecting systematic deviations between model calculations and measurements. Although environmental system identification cannot be automated completely, model identification tools would be very useful to scientists, who critically reflect the results. The disadvantages of most currently available programs of this category is their limitation to single-input single-output and/or to linear systems. Because environmental systems are (nearly always) nonlinear multiple-input multiple-output systems, most programs in this category cannot be used to model environmental systems. Together with the program interfaces, which are usually designed for electrical engineers and not for environmental scientists, this is the main reason why such programs have been rarely used in environmental systems analysis so far.

3.2 Requirements of a Computer Program for the Identification and Simulation of Aquatic Systems

As outlined in chapter 2, system identification of environmental systems is a complicated task requiring several analysis steps. In section 3.1, the available computer programs are classified into three categories, in each of which different aspects of simulation or system identification are implemented. Although this classification is not strict, and although there do exist programs which include features belonging to more than one category, no user-friendly program exists which is able to perform a complete system identification for a complex environmental system. There are two ways of solving this problem: existing environmental simulation programs can be coupled to a universal system analysis program, or more universal environmental simulation programs that include system identification can be designed.

An example of the first of these approaches is described by Jannssen et al. (1992). The disadvantage in using such a program is that the user must implement the interface between the programs. This can be a difficult task. For this reason, a standardization of program interfaces, which would make the specification of model parameters and the transfer of calculated results possible, would be very desirable. As long as such a standardization cannot be achieved, the coupling of simulation programs designed for specific environmental systems with general purpose identification programs would appear to be too troublesome.

These coupling problems make apparent the need for a more universal identification and simulation program capable of performing the tasks of all three program categories discussed in section 3.1. In this section, the requirements of such a program for the identification and simulation of a class of aquatic systems are discussed. The purpose of restricting the program to a class of aquatic systems is to limit the amount of program development necessary; however, the key concepts can also be applied to other environmental and technical systems. The program requirements are classified into scientific (S1-S7) and technical (T1-T4) requirements. Each of these classes of requirements is discussed in a subsection of this chapter. The possibility of fulfilling the requirements outlined in this section will be discussed in chapters 4 - 7.

3.2.1 Scientific Requirements

The scientific requirements of a computer program for the identification and simulation of aquatic systems consist of making methods available which will allow the user to perform all system identification and uncertainty analysis tasks discussed in sec-

tions 2.3 and 2.4. Such a program must combine the features of all categories of programs described in section 3.1. The program should be as flexible as possible with regard to model formulation similarly to the programs described in section 3.1.1, but should possess a user interface designed for environmental scientists familiar with aquatic systems which is as easy to handle as those of the programs described in section 3.1.2. Furthermore, the program should provide support for calculating water flow for the most important environmental, technical and laboratory systems. As a special case, such a program allows the user to implement most of the models implemented in the programs described in section 3.1.2 and to compare them with alternative model formulations. As a very important extension to conventional environmental simulation programs, the program should support model identification techniques characteristic of the programs described in section 3.1.3 for the class of nonlinear multiple-input multiple-output systems used for the description of aquatic systems.

Comparing the requirements summarized above with the available methods described in chapter 2, the following list of detailed requirements can be given:

Simulation

The first requirement facilitates model formulation for the case in which transformation processes, and not water flow, are the main object of investigation:

S1: The program should offer the calculation of water flow and substance transport for the most important environmental, technical and laboratory systems.

This requirement does not exclude user influence on water flow and substance transport by the choice of parameters for geometry, friction, diffusion, dispersion, etc., but it makes user-implementation of well-known differential equations that are not the object of investigation unnecessary.

The following requirement allows the user to formulate his or her own transformation models:

S2: The program should give the user high flexibility in defining state variables and as much freedom as possible in specifying transformation processes.

This requirement guarantees the ability to compare different model structures, which is necessary for the model structure evaluation of transformation processes (cf. section 2.3.1).

As a further requirement, it has to be possible to compare the results of model simulations graphically with measured data:

S3: The program should offer basic plotting facilities that make fast graphical comparisons of measured and calculated values for different models possible. Additionally, it should provide an interface which allows the user to transfer calculated results to external programs.

As stated in section 2.3.1, such comparisons are the most important tool for determining systematic deviations between the simulation and the actual system. High quality plots are not required, because final plots can be created with a professional graphics package after transferring the data.

Model Structure Evaluation (section 2.3.1)

Since the model structure evaluation step involves a priori knowledge and depends on modelling objectives, it is the most difficult aspect of the model to automate. The most important requirements here are flexibility in model formulation (requirement S2), plotting facilities (requirement S3), and the following requirement for supporting decisions between competing model structures:

S4: The program should provide a quantitative measure of the deviation between measured and calculated values.

Furthermore, automatic model parameter estimation is required to allow the users to compare different models efficiently. This requirement is formulated below (requirement S6).

Identifiability Analysis of Parameters (section 2.3.2)

Since theoretical identifiability analysis only uses the information of which variables are measured, and neither requires the measured values themselves nor a numerical solution of the model, this analysis step can be performed well using a separate program. Modelers interested in theoretical identifiability analysis usually need a computer algebra program to perform this task. For this reason, theoretical identifiability analysis is excluded from the data analysis program described in this report.

However, to support practical identifiability analysis, the following feature should be provided:

S5: The program should allow linear or nonlinear sensitivity analyses to be performed.

The second method of identifiability analysis mentioned in section 2.3.2, the estimation of parameter covariance, is requested below in requirement S6.

Parameter Estimation (section 2.3.3)

An automatic parameter estimation technique must be provided:

S6: The program should include an automatic parameter and parameter variance-covariance estimation procedure which allows universal and experiment-specific model and initial state parameters, several target variables and several experiments to be combined into a single calculation.

This requirement not only fulfills the needs of efficient model comparison as requested for model structure evaluation, but also gives an *a posteriori* estimate of parameter identifiability.

Model Confirmation (section 2.3.4)

Model confirmation requires a quantitative measure of the deviation between calculated and measured values as already requested in requirement S4.

Uncertainty Estimation (section 2.4)

Since requirement S2 already allows the users to calculate different scenarios, the new point here is the calculation of the propagation of parameter errors:

S7: The program should allow the user to estimate the uncertainty of results due to propagation of parameter uncertainty using either the linearized formula (2.21) or Monte Carlo simulation.

A program which is designed to make a step in the direction of a universal modelling tool for aquatic systems should offer at least the most elementary methods of fulfilling requirements S1 to S7. A possible implementation of the requirements S1 and S2 is discussed in chapter 4 and section 5.1, of S3 in section 7.5, of S5 in section 5.2, of S4 and S6 in section 5.3, and of S7 in section 5.4.

3.2.2 Technical Requirements

The most important technical requirement of a program for scientific simulation and data analysis that is immediately apparent to the user is user-friendliness:

T1: The program should be easy for environmental scientists to operate, both interactively and in batch mode.

This requirement is essential if the goal of making the application of system analytical methods to environmental data more popular to be attained. It is evident that editing model definitions, selecting program tasks and plotting results are best done using the native graphical user interface of a machine. However, on machines with multitasking operating systems it should also be possible to submit long calculations, such as those necessary for sensitivity analyses and parameter estimations of complex systems, as batch jobs. It should also be possible to view the results of such batch calculations with the interactive program version and to plot the results automatically during a batch job. The main implications of this requirement are discussed along with the formulation of model structure in chapter 4 and the implementation of the user interface in section 7.5.

To allow the program to be used in laboratories which are not connected to a computing center, to take into account the high degree of dependence the model complexity and the program task have on the required computing power, and to take into account the habits of the users, the following requirement should be fulfilled:

T2: The program should be transferable to any important computing platform.

This requirement not only makes the program available to a wider range of users, but it guarantees that the investment in program development time is preserved even in the case of changes in important hardware platforms and operating systems. It is evident that this requirement conflicts with requirement T1, which requests use of the native graphical user interface of each machine. The way of fulfilling this requirement is discussed mainly in the introduction to chapter 7 and in section 7.5.

To keep the amount of requested computation time as small as possible and to compensate for unavoidable efficiency losses due to model formulation flexibility, the following requirement should be fulfilled:

T3: Numerical computations should use efficient algorithms and the interfaces between the main program and routines implementing numerical algorithms should be as simple as possible.

The requirement that main program and numerical algorithms be clearly separated facilitates subsequent changes and improvements in numerical methods, and takes into account for the fact that nearly all numerical programs are written in FORTRAN, which, as will be discussed later, cannot be selected as the programming language for this program. The numerical algorithms proposed to fulfill this requirement are discussed in chapter 6.

The attempt to make a program as universal as possible has evident limits. Often, to keep programming effort down to a reasonable size, a pragmatic compromise has to be made between universality and program complexity. If the program is already in use, such compromises obviously depend on the current needs of the users. In order not to limit future development too much by such decisions, and to account for rapidly changing needs during scientific investigations, the following requirement is very important:

T4: The program code should be well structured, well documented, and should be based on an extensible program design.

The main goal of chapter 7 is to demonstrate how it is possible to fulfill this requirement by applying the methods of object-oriented program design.

4 Model Formulation

In this chapter, a general framework for the formulation of models of aquatic systems in nature, in technical plants and in the laboratory is developed. This framework was designed to fulfill the following requirements (the requirement numbers from section 3.2 are given in parentheses):

- the model structure should provide the differential equations for water flow and substance transport for the most important environmental, technical and laboratory systems (S1),
- the general model formulation should allow for an arbitrary number of state variables, and it should restrict formulation of transformation processes as little as possible (S2),
- the model formulation should use a "language" that is easy for environmental scientists to understand (T1),
- the model structure should already consider the possibility of a well structured implementation in a computer program (T4).

The reasons for including environmental, technical and laboratory systems in the same model formalism are to make a single tool available for scientists working in all three fields and to allow the users to evaluate combined laboratory-field studies.

Deterministic continuous-time models are formulated as systems of ordinary and/or partial differential equations (cf. discussion in section 2.1). The type of partial differential equation is determined by the transport terms and is not affected by transformation processes, which are formulated as source terms. This makes the following ***general model design strategy*** for fulfilling the above-mentioned requirements possible: ***The model structure should offer several compartment types with different dominant transport processes. It should be possible to combine such compartments to yield a spatial configuration describing the aquatic system to be modelled, and to specify arbitrary transformation processes for an arbitrary number of state variables within the compartments.*** This general design strategy clearly fulfills requirements S1 and S2. Since the type of partial differential equation is determined by the compartment type, specific techniques can be applied to optimize the numerical solution to the partial differential equations characteristic of the current compartment type without restricting process formulation. This gives a good basis for fulfilling requirement T4. By a good choice of compartment types and a clear interface for the specification of transformation processes, requirement T1 can also be fulfilled.

As mentioned above, the aquatic system to be modelled is divided into ***compartments*** with well defined dominant transport processes. Furthermore, the interactions between such compartments should be limited to a small number of clearly defined interfaces.
Fig. 4.1 shows pictograms of the following examples of compartment types: a *mixed reactor compartment* represents a region of space in which concentration gradients can be neglected (e.g. a stirred laboratory reactor, a mixed basin of a waste water

treatment plant or a circulating lake). Such a compartment leads to a system of ordinary differential equations, because the only transport to be considered is inflow and outflow (the dominant transport process is mixing, which must not be explicitly considered if it is strong enough to eliminate concentration gradients completely). A *biofilm reactor compartment* represents a reactor with a wall on which a film of microorganisms (biofilm) grows. The dominant transport processes are mixing in the bulk water volume outside of the biofilm, diffusion of dissolved substances and advection of particulate substances within the biofilm, and attachment and detachment processes at the interface. A *river section compartment* describes a river reach. In this compartment, water discharge and water level elevation have to be calculated according to the equations of open channel hydraulics, and the dominant transport processes are advection and longitudinal dispersion. An *advective-diffusive compartment* describes advective and diffusive (or dispersive, depending on the object to which the model is being applied) longitudinal transport. A *soil column compartment* describes saturated or unsaturated water flow and advective and dispersive substance transport in a laboratory column or a one-dimensional field site. A *lake compartment* represents a lake or a lake basin consisting of a mixed surface layer (epilimnion), a stratified hypolimnion and sediment. If complete horizontal mixing is assumed, vertical turbulent diffusion and sedimentation are the dominant transport processes.

Note that the compartment types shown in Fig. 4.1 are only examples and that other types of compartments can be realized as well. All of the compartment types shown in Fig. 4.1 are zero- or one-dimensional approximations to three-dimensional reality. The modelling concepts described in this report (with slight modifications) can also be used to implement higher dimensional compartments. The main problem of higher dimensional compartments is the large amount of calculation time necessary for simulation, and, in particular, for the application of system analytical techniques as

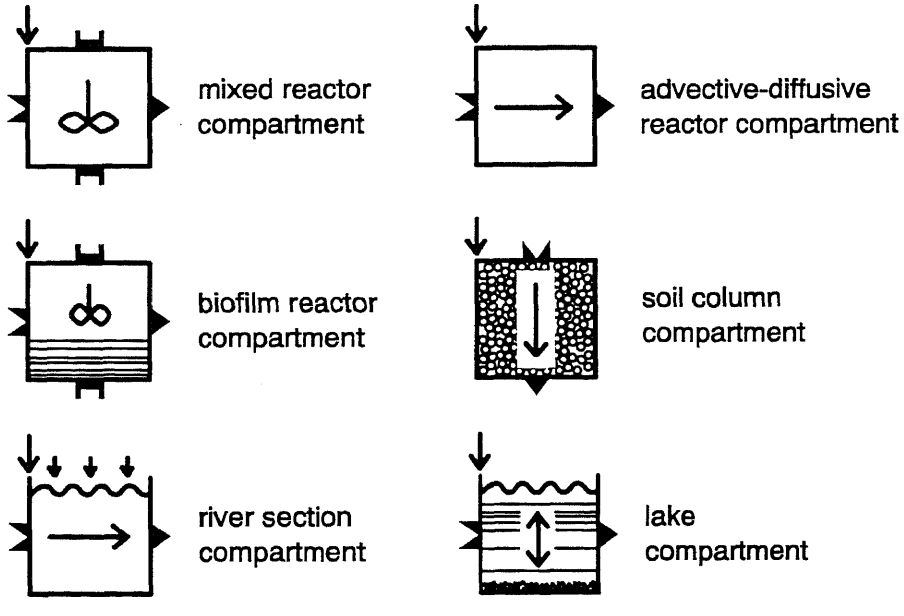


Fig. 4.1: Pictograms of possible compartment types.

discussed in this report. In this report, detailed discussion is limited to the three compartment types shown in the left hand column of Fig. 4.1. These compartment types are sufficient to discuss the key concepts of model structure, systems analysis, numerical methods and implementation techniques. These three compartment types are those considered in the first version of the program AQUASIM, implemented according to the guidelines described in this report.
Aquatic systems consisting of more than one compartment require ***links*** for their connection. Fig. 4.2 shows pictograms of two examples of (zero-dimensional) link types: *Advective links* are used to describe water flow and advective substance transport between compartments and to make the modelling of bifurcations and junctions possible. *Diffusive links* model diffusive boundary layers or membranes between compartments which can be penetrated diffusively by certain substances. Other types of links could be included into the model description as well. The link types shown in Fig. 4.2 are those implemented in the first version of the program AQUASIM, realized according to the guidelines described in this report.

Figs. 4.3 and 4.4 show two examples of aquatic systems composed of compartments from Fig. 4.1 and links from Fig. 4.2. The system shown in Fig. 4.3 represents a waste water treatment plant consisting of two mixed basins with recirculation and sludge export. The treatment plant releases treated waste water into a river modelled by two river sections. The first river section represents an uncontaminated river reach, whereas changes due to the release of the effluent of the waste water treatment plant can be modeled in the second river section. The system shown in Fig. 4.4 represents a laboratory system consisting of a biofilm growing on a diffusively permeable tube in a mixed reactor which is also subject to gas exchange with the atmosphere.

In many cases, ***transformation processes*** are the main object of investigation in an aquatic system. Therefore, the general model structure should restrict formulation of transformation processes as little as possible. Since characteristic times of processes can vary over several orders of magnitude, concentrations depending on fast

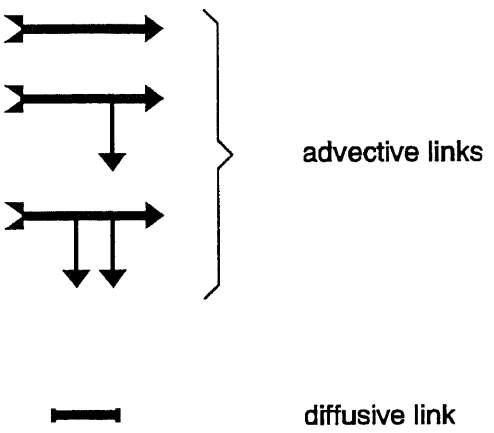


Fig. 4.2: Pictograms of possible link types.

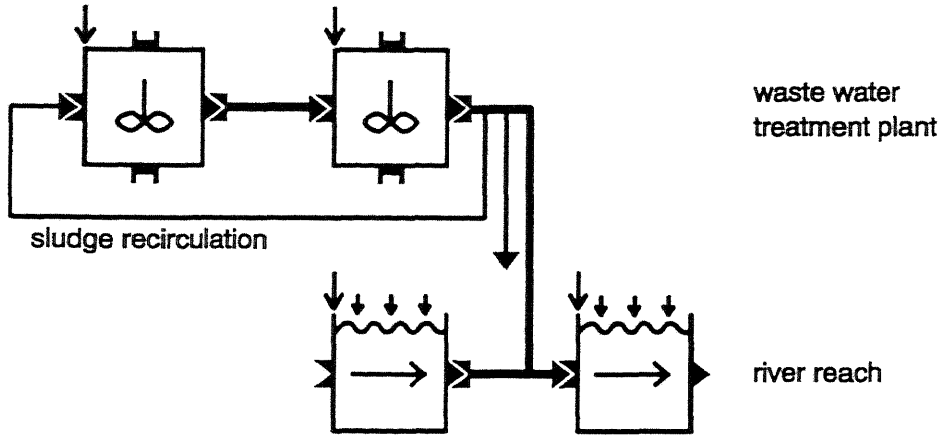


Fig. 4.3: Example of a spatial configuration representing a waste water treatment plant and the river into which the treated water is released.

processes often converge so fast to their current equilibrium values (which temselves depend on slower processes) that the transient phase is not of importance for the overall behavior of the system. In such situations, a separation of time scales describing the equilibria of fast processes in terms of algebraic equations can be advantageous. This leads to two types of processes beeing distinguished: *Dynamic processes* are described in terms of a common process rate and individual stoichiometric

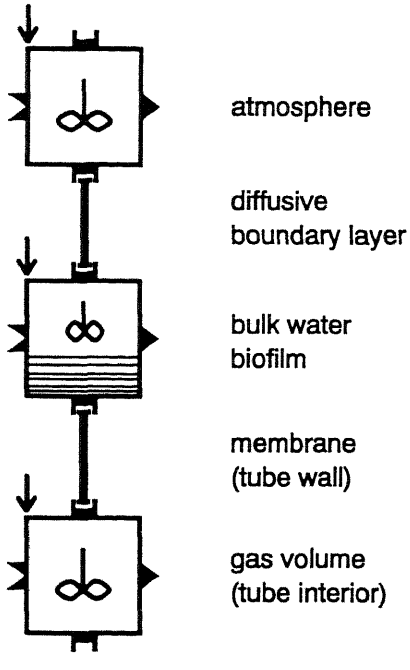


Fig. 4.4: Example of a spatial configuration representing a biofilm growing on a permeable tube in a reactor subject to gas exchange with the atmosphere.

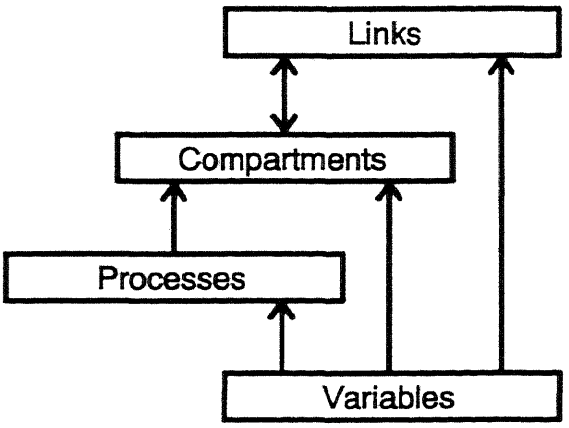


Fig. 4.5: Mutual dependence between the subsystems forming the general model structure.

coefficients which determine the relative effects on different variables. The temporal evolution of variables affected by dynamic processes is determined by the solution of a system of differential equations. The second type of processes are *equilibrium processes*, in which the values of the corresponding variables are determined by solving algebraic equations characterizing the equilibrium values of the variables.

For the formulation of models consisting of compartments, links and processes, ***variables*** are needed. Variables are objects which are characterized by a context-sensitive numeric value. Three main categories of variables are distinguished: *system variables* correspond to quantities to be calculated by the model, *data variables* represent measured data and *function variables* can build functional relations using other variables.

Summarizing the preceding paragraphs, a model consists of the four subsystems of variables, processes, compartments and links. Fig. 4.5 illustrates the mutual dependences existing between these subsystems. It is evident that variables form the basic subsystem required for the formulation of processes, compartments and links. Compartments additionally use processes, and links connect compartments to the desired spatial configuration of the system.

The four subsystems of variables, processes, compartments and links introduced above are discussed in more detail in the four sections of this chapter.

4.1 System of Variables

The basic objects for the formulation of models are variables. Variables are identified by their name. They are characterized by the property of taking a possibly context-sensitive numeric value. This property is important for the object-oriented implementation of variables discussed in chapter 7, which is designed to fulfill requirement T4 of extensible program design (cf. section 3.2). There are three main ranges of application of variables: Variables can be used for quantities to be determined by the model (e.g. by the solution of algebraic or differential equations) or which have a predefined meaning in a compartment (e.g. time and space coordinates), they can be used to provide data (e.g. model parameters or measured data series) or they can be used to build functions depending on other variables (e.g. for the specification of process rates or stoichiometric coefficients). According to these ranges of application, three main categories of variables are distinguished:

- system variables,
- data variables,
- function variables.

These three categories of variables are described in more detail in the following subsections.

4.1.1 System Variables

System variables represent quantities to be determined by a model or which have a predefined meaning in a compartment. This leads to the division of system variables into two types:

- state variables,
- program variables.

These two types of system variables are described in more detail in the following paragraphs.

State variables describe properties of water or of a surface in contact with water (e.g. temperature, masses or concentrations of dissolved or suspended substances or of substances attached to a surface). State variables obtain their meaning indirectly by the processes in which they are involved. There are the following types of state variables:

- dynamic state variables (volume or surface),
- equilibrium state variables.

The values of dynamic state variables are given by the solution to differential equations describing transport and user-defined transpromation processes. Dynamic state variables may either represent substances transported with the water (quantified as mass per unit volume) or substances attached to a surface (quantified as mass, as mass per unit length or as mass per unit surface). Equilibrium state variables represent substances, the process rates of which are much faster than those of other processes, so that they always can be approximated to take the values corresponding to the equilibrium state of their transformation processes. Therefore, the values of equilibrium state variables are given by the solution of algebraic equations.

In contrast to state variables, the meaning of which is implicitly given by their transformation processes, ***program variables*** refer to predefined quantities of the modelled system. From a mathematical point of view, program variables can represent independent variables, parameters, or solutions of differential-algebraic systems. The idea of program variables is to make the corresponding quantities, which anyway are present in model formulation, available for use in the system of variables. Besides program variables for time and space coordinates and for compartment specific quantities, the set of program variables also includes a "Calculation Number", which allows the user to distinguish different calculations. Consult the user manual of the program developed according to the guidelines described in this report, which is given in the appendix, for a list of program variables available in different compartments.

4.1.2 Data Variables

Data variables make measured data available to a model. Single parameters and lists of argument-value pairs for quantities measured as a function of another variable are distinguished. This leads to the following two types of data variables:

- constant variables,
- real list variables.

These two types of variables are described in the following paragraphs.

Constant variables describe single measured quantities consisting of a value, its accuracy characterized by the standard deviation, and a minimum and a maximum which bound the legal range of values. For simulations, only the value is used. The

standard deviation together with the legal range is used during sensitivity analyses. Constant variables can be used as parameters to be estimated by the program. In this case, the optimum value of the parameter within the legal range is found automatically.

Quantities measured as a function of another variable, e.g. time series or spatial profiles, are represented by ***real list variables***. For the definition of a real list variable a variable representing the argument, a list of argument-value data pairs, the standard deviations of the data and an interpolation method must be specified. The standard deviations can be given as global relative and absolute standard deviations or as individual standard deviations for all data values. Real list variables are usually evaluated as follows: In a first step, the variable given as the argument is evaluated. Then, the value of the variable is calculated by employing the selected interpolation method at the value of the argument. Fig. 4.6 shows interpolation of two real list variables with different interpolation and smoothing methods (compare the appendix for a description of the set of implemented interpolation methods). Note that spline interpolation may lead to undesired oscillations in case of very abrupt changes in the data series. An alternative use of real list variables is their use as target variables for

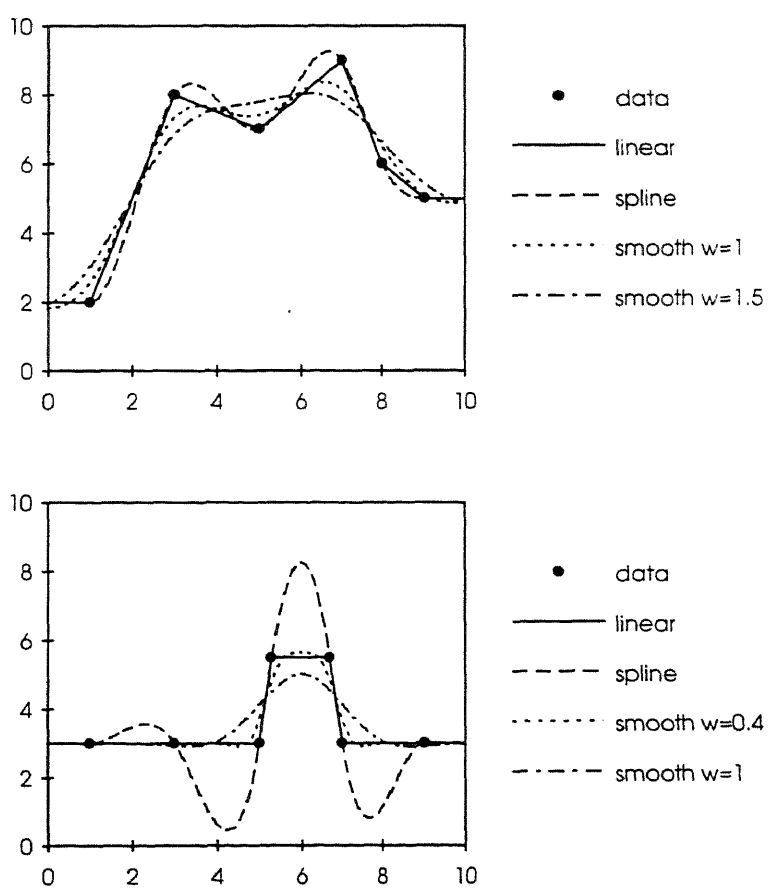


Fig. 4.6: Behavior of different interpolation and smoothing options for real list variables.

parameter estimations. This is only possible, if the argument of the real list variable is either the program variable for time or the program variable describing the space coordinate of the compartment in which the comparison takes place. In this case, no interpolation is performed, but the variable to be compared with the real list variable is evaluated at the positions of the data pairs and the differences between the values of the two variables are summed up according to equation (2.12) using the standard deviations specified for the real list variable.

4.1.3 Function Variables

Function variables allow the users of the program to build functional relationships unsing previously defined variables. There are the following types of function variables:

- variable list variables,
- formula variables.

Variable list variables are similar to real list variables, but instead of a value, another variable corresponds to each value of the argument. Multi-stage application of variable list variables and combination with other variable types makes the construction of very complicated functional dependences possible.

The last, but most versatile variable type is the ***formula variable***. An algebraic expression using the previously defined variables can be given to define the new variable. Cyclic references of variables, however, are not allowed. In the appendix, the formula syntax chosen for the implementation is given. This syntax allows the users to specify any algebraic expression using variables, usual operations and elementary functions, and even logical branching within an expression is possible.

4.1.4 Flexibility of the System of Variables

As will be shown in chapter 7, the object-oriented implementation of variables easily allows the programmer to add additional types of variables (partial fulfillment of requirement T4 discussed in section 3.2). It is the goal of this section, to demonstrate the power of the combination of the six (relatively) simple types of variables described in the preceeding sections. This property of the system of variables forms the basis for fulfilling requirement S2 of universal process formulation (cf. section 3.2), which is discussed in section 4.2.

The most obvious deficiency of the system of variables seems to be the lack of a multi-dimensional interpolation variable. While in some cases such a variable may be advantageous, usually the combination of real list variables with variable list variables is sufficient for multidimensional interpolation. As an example, time series of a quantity measured at different places, can be specified as real list variables with time as the argument, and can then be used within a variable list variable with the space coordinate as the argument to make two-dimensional interpolation in time and space possible. This construction can obviously be extended to more than two dimensions.

Parameter estimations usually have the purpose of estimating the values of some constant parameters of a model. This can easily be realized by defining these parameters as constant variables. However, sometimes the values of a function depending on another variable as its argument should be estimated. The system of variables offers two possibilities for estimating such a function: The first possibility is to parameterize the function by an algebraic expression with unknown parameters. This function is realized as a formula variable using constant variables as parameters. Then, these parameters can be estimated in the usual way. The other possibility is to realize the unknown function as a variable list variable with constant variables corresponding to selected values of the argument. These constant variables, between which the function is interpolated with a method selected by the user, can then again be estimated with the aid of measured data. In both cases, it is important not to exceed a reasonable (problem dependent) number of parameters to be simultaneously estimated to avoid parameter identifiability problems.

As a last point, the utility of the program variable "Calculation Number" must be discussed. If several independent measurements with different environmental conditions were performed for the same system, the calculation number can be used to distinguish between the experiments. As an example, different inflows can be realized as real list variables with time as the argument used in a variable list variable with the calculation number as the argument. In the same way, parameters to be estimated, which depend on the experiment, can be realized as constant variables in a variable list variable with the calculation number as the argument. Such constructions allow the users to perform parameter estimations with global parameters, experiment-specific parameters and even with parameters specific to certain subsets of experiments.

4.2 System of Processes

A biological, chemical or physical process can be defined by a set of process rates, each of which describes the contribution of the process to the temporal change of the concentration of a given substance. Characteristic times of such processes may vary over several orders of magnitude. The consequence of the existence of widely varying time scales within a system is that the concentrations determined by fast processes converge so fast to their current equilibrium values (which themself depend on slower processes) that the transient phase is not important for the behavior of the system on the slower time scale. In such situations, a separation of time scales, which solves concentrations determined by fast processes directly for their equilibrium solution, can be advantageous. This leads to a replacement of differential equatios of fast processes by algebraic equations. Therefore, the following two types of processes are introduced:

- dynamic processes,
- equilibrium processes.

These two types of processes are described in more detail in the following subsections.

4.2.1 Dynamic Processes

A clear presentation of dynamic biochemical processes is very important to facilitate readers of a paper or users of the simulation program to obtain a survey of the interactions between the components of the system. The method of presentation used for the data analysis program described in this report was made popular for technical biochemical systems by Henze et al. (1986). It is based on work by Petersen (1965).

Dynamic processes describe transformations by their contribution to the temporal rate of change of (dynamic) state variables. Usually, a biological or chemical process transforms several substances in fixed stoichiometric proportions. Therefore, it is advantageous to separate a common factor as a reaction rate, and to describe a process by this reaction rate and by stoichiometric coefficients for all substances involved into the process. The contribution of a process to the temporal change of the concentration of a substance is then given as the product of the common reaction rate and the substance-specific stoichiometric coefficient. This decomposition of process rates into a common reaction rate and individual stoichiometric coefficients is not unique; to make it unique, one of the stoichiometric coefficients is usually set to one. Physical processes or transfer processes which due to spatial averaging also

Table 4.1: Representation of a process model with the aid of a process matrix.

Process	Substance				Reaction Rate
	s_1	s_2	s_3	.	
p_1	ν_{11}	ν_{12}	ν_{13}	.	r_{p_1}
p_2	ν_{21}	ν_{22}	ν_{23}	.	r_{p_2}
p_3	ν_{31}	ν_{32}	ν_{33}	.	r_{p_3}
.	.	.	.	.	.
.	.	.	.	.	.

have the mathematical form of a source term of the differential equation can be integrated easily into this general scheme. The notion of a stoichiometric coefficient has then a more general meaning and can include geometric factors as well. With this concept, the total transformation rate of a substance s_j is given by

$$r_j = \sum_i \nu_{ij} r_{p_i} \quad , \tag{4.1}$$

where r_j ($ML^{-3}T^{-1}$) is the total transformation rate of the substance s_j, ν_{ij} (-) the stoichiometric coefficient of the substance s_j for the process p_i and r_{p_i} ($ML^{-3}T^{-1}$) the reaction rate of the process p_i. A clear representation of a process model is given by writing the stoichiometric matrix (ν_{ij}), supplemented by the reaction rates p_i in an additional column. This results in a process matrix as shown in Table 4.1. The nonzero elements of a row of such a matrix show, which substances are affected by a given process, whereas the nonzero elements of a column indicate, which processes have an influence to a given substance.

In consequence to the formulation of dynamic processes as described above, a dynamic process, as used in the program described in this report, consists of a reaction rate and of a list of variables together with their stoichiometric coefficients. The change of models is facilitated if in this list all types of variables are allowed. Only stoichiometric coefficients corresponding to dynamic state variables have an effect. The design of dynamic processes as a row of the process matrix shown in Table 4.1 is familiar to environmental scientists and thus fulfills requirement T1 as stated in section 3.2. Due to the possibility of formulating process rates and stoichiometric coefficients as algebraic expressions, the user of the program can define any possible transformation process and requirement S2 is also fulfilled.

4.2.2 Equilibrium Processes

Equilibrium processes are used for processes, which are much faster than the processes which determine the typical time scale of the simulation. A variable determined by such a process can be treated as taking always the value corresponding to its equilibrium state. Therefore, its value is given as the solution of an algebraic equation:

$$r_{eq} = 0 \quad , \tag{4.2}$$

where r_{eq} depends on the variable involved and on other variables influencing the equilibrium value.

Consequently, an equilibrium process of the program described in this report consists of a variable and of an algebraic expression r_{eq}. As for dynamic processes, all types of variables are allowed. If the variable is an equilibrium state variable, its value is determined by the solution to the equation defined by setting the algebraic expression to zero. If the variable is of another type, the process has no effect. The formulation of chemical equilibria as algebraic equations is familiar to chemical engineers so that this formulation fulfills the requirement T1 stated in section 3.2. Due to the freedom in building algebraic expressions also requirement S2 is fulfilled.

4.3 System of Compartments

As discussed in the introduction to chapter 4, the concept of how to fulfill requirement S1, which demands to provide the solution to important water flow and substance transport equations (cf. section 3.2), is to offer different compartment types, characterized by dominant transport processes. The user then can build the spatial configuration of the system to be modelled using an arbitrary number of such compartments. The restriction to special compartment types has to be compensated for by an implementation strategy which easily allows the programmer to introduce additional compartment types (cf. requirement T4 discussed in section 3.2). In this report, the discussion is limited to the three compartment types

- mixed reactor compartment,
- biofilm reactor compartment,
- river section compartment,

which are implemented in the first version of the program realized according to the guidelines described in this report. While these compartment types do not cover all important environmental and technical systems, they are sufficient to outline the important concepts. More compartment types can be implemented using the same strategy.

After an introductory subsection treating mathematical formulation of conservation laws, the differential equations provided for each of the three compartment types listed above are discussed in a separate subsection of this section.

4.3.1 Mathematical Formulation of Conservation Laws

The basic principle of conservation laws is very simple: If $\mathbf{m}(t)$ ([**m**])is an array of conserved properties (e.g. mass, momentum or energy) in a given region of space, $\mathbf{I}(t)$ ($[\mathbf{m}]T^{-1}$) is the array of fluxes of the properties across the boundary of the region and $\mathbf{R}(t)$ ($[\mathbf{m}]T^{-1}$) is the array of net production rates of the properties in the region, then **m** at the time t_2 is given as the sum of **m** at the time t_1 plus the time integrals of the fluxes $\mathbf{I}(t)$ and production rates $\mathbf{R}(t)$ over the time interval from t_1 to t_2:

$$\mathbf{m}(t_2) = \mathbf{m}(t_1) + \int_{t_1}^{t_2} \mathbf{I}(t)\,dt + \int_{t_1}^{t_2} \mathbf{R}(t)\,dt \quad . \tag{4.3}$$

If the derivative of **m**(t) is defined, (4.3) can be transformed to:

$$\frac{d}{dt}(\mathbf{m}(t)) = \mathbf{I}(t) + \mathbf{R}(t) \quad . \tag{4.4}$$

Equation (4.4) can be directly applied in the zero-dimensional case of a mixed reactor compartment. For higher dimensional compartments, if the conserved quantities are piecewise differentiable functions of space coordinates, it is useful to introduce densities. Because the compartments discussed in this study are at most one-dimensional, the following discussion concentrates on the one-dimensional case only. The array of one dimensional densities $\hat{\rho}$ ($[\mathbf{m}]L^{-1}$) is defined as the vector of the quantities of the properties per unit length (the symbol "^" is used to distinguish the one-dimensional density from the spatial density ρ which is defined as the quantity of a property per unit volume). If the considered region in space is bounded by the coordinates x_1 and x_2, the total quantity **m**(t) can then be expressed by the integral

$$\mathbf{m}(t) = \int_{x_1}^{x_2} \hat{\rho}(x,t)\, dx \quad . \tag{4.5}$$

The flux **I**(t) across a moving boundary $x_i(t)$ is given by

$$\mathbf{I}(t) = \hat{\mathbf{j}}(x_i(t),t) - \frac{dx_i}{dt}(t)\, \hat{\rho}(x_i(t),t) \quad , \tag{4.6}$$

where $\hat{\mathbf{j}}$ ($[\mathbf{m}]T^{-1}$) is the array of one-dimensional fluxes describing the quantities of the properties transported per unit time relative to the resting frame of reference (the symbol "^" is used to distinguish the one-dimensional flux from the spatial flux **j** which is defined as the quantity transported per unit time and per unit surface area perpendicular to the direction of transport). Finally, the production rates **R**(t) can be written as the space integral of the one-dimensional production rates $\hat{\mathbf{r}}$ ($[\mathbf{m}]L^{-1}T^{-1}$) which are defined as the production rates of the properties per unit time and per unit length of the system (the symbol "^" is used to distinguish the one-dimensional production rates from the spatial production rates **r** specifying the properties produced per unit time and per unit volume):

$$\mathbf{R}(t) = \int_{x_1}^{x_2} \hat{\mathbf{r}}(x,t)\, dx \quad . \tag{4.7}$$

Using the expressions (4.5), (4.6) and (4.7) for one-dimensional systems, the general conservation law (4.4) can be written in the form

$$\frac{d}{dt}\left(\int_{x_1}^{x_2} \hat{\rho}(x,t)\,dx\right) = \hat{j}(x_1,t) - \frac{dx_1}{dt}\hat{\rho}(x_1,t) - \left(\hat{j}(x_2,t) - \frac{dx_2}{dt}\hat{\rho}(x_2,t)\right) + \int_{x_1}^{x_2} \hat{r}(x,t)\,dx \quad . \tag{4.8}$$

This is the (semi-) integral form of one-dimensional conservation laws. If all relevant quantities are continously differentiable with respect to the space coordinate x, equation (4.8) can be transformed to the differential form

$$\frac{\partial \hat{\rho}}{\partial t} + \frac{\partial \hat{j}}{\partial x} = \hat{r} \quad . \tag{4.9}$$

In both cases, the definition of a system consists of specifying the functions $\hat{\rho}$, $\hat{j}$ and $\hat{r}$. The first form (4.8) has the advantage of being more general in allowing the solutions to be (spatially) discontinuous. Furthermore, as is shown in chapter 6, equation (4.8) is the basis for the formulation of so-called conservative numerical discretization methods for conservation laws. These methods have the advantage of exactly fulfilling a discretized version of the conservation law.

4.3.2 Mixed Reactor Compartment

The simplest type of a compartment is a mixed reactor compartment, in which spatial concentration gradients are neglected. Such a compartment can be used to describe a stirred laboratory reactor, a mixed basin of a waste water treatment plant or even a circulating lake. In this subsection, the equations determining water flow and the behavior of the different types of state variables within a mixed reactor compartment are discussed.

Fig. 4.7 shows the pictogram of a mixed reactor compartment showing inflow (arrow), advective inflow and outflow connections (symbols on the left and right of the pictogram) and diffusive connections (symbols at the top and bottom of the pictogram).

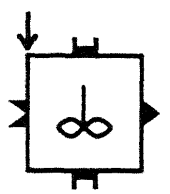


Fig. 4.7: Pictogram of a mixed reactor compartment.

Water Flow

As can be seen in Fig. 4.7, a mixed reactor compartment has two types of inflows: Input directly specified for the compartment (arrow in the top left corner) and input via the advective links connected to the input connection (symbol on the left) of the mixed reactor compartment (there is no water flow through a diffusive link):

$$Q_{in} = Q_{in,comp} + \sum Q_{in,advlink} \quad . \tag{4.10}$$

The discharge Q (L^3T^{-1}) through the compartment is equal to the inflow:

$$Q = Q_{in} \quad . \tag{4.11}$$

In the case of a reactor with *constant volume*, the discharge of the effluent is given by

$$Q_{ef} = Q \quad . \tag{4.12a}$$

In a reactor with *variable volume*, the discharge of the effluent can be specified by the user (e.g. depending on the current value of the reactor volume V_R or of time t):

$$Q_{ef} = Q_{ef}(V_R,t,...) \quad . \tag{4.12b}$$

In this case, the reactor volume V_R (L^3) is given by the solution of the differential equation

$$\frac{dV_R}{dt} = Q_{in} - Q_{ef} \quad . \tag{4.13}$$

The user has to be careful in specifying Q_{ef} in a way that the reactor volume calculated according to (4.13) does not become negative.

Dynamic Volume State Variables

Dynamic volume state variables represent concentrations C_i (ML^{-3}) of quantities transported with water flow. Temporal evolution of these concentrations is given by the conservation law (4.4) which, in this case, can be written in the form

$$\frac{d}{dt}\left(V_R C_i \right) = I_i + V_R r_i \quad , \tag{4.14}$$

where C_i is the concentration of substance i specified as mass per unit volume, I_i (MT^{-1}) is the total mass flux of substance i into the compartment and r_i ($ML^{-3}T^{-1}$) is the net production rate of substance i per unit volume. I_i has four types of contributions

$$I_i = I_{i,in,comp} + \sum I_{i,in,advlink} - I_{i,ef,advlink} + \sum I_{i,difflink} \quad . \tag{4.15}$$

The first contribution can be specified by the user in an arbitrary form $I_{i,in,comp}(t)$ (because often concentration measurements are available instead of flux measurements, the form will usually be $I_{i,in,comp}(t) = Q_{in}(t) \cdot C_{i,in,advlink}(t)$; it is, however, possible to specify substance mass input without water inflow). The second contribution is determined by the advective links connected to the input connection of the mixed reactor compartment. The third term, the output flux of substance i, is given as the product of discharge of the effluent times concentration within the mixed reactor:

$$I_{i,ef,advlink} = Q_{ef} C_i \quad . \tag{4.16}$$

Finally, the fluxes through diffusive links depend on the concentrations at both sides of the link and are given in section 4.4.

Dynamic Surface State Variables

Dynamic surface state variables may represent masses or surface densities C_i (M or ML^{-2}) of substances attached to a surface within the mixed reactor compartment. Consequently, there is no exchange of such variables through links. The thickness of the film built by these variables is assumed to be thin enough so that transport pro-

cesses within the film can be neglected and the substances are affected by the mean substance concentrations within the reactor (see section 4.3.3 for a generalization of this concept). The time evolution of dynamic surface state variables is given by the process rate only:

$$\frac{dC_i}{dt} = r_i \quad . \tag{4.17}$$

Note that the process rate r_i (MT^{-1} or $ML^{-2}T^{-1}$) can depend on concentrations of substances dissolved or suspended in the water as well as on surface densities of attached substances. This allows the user to model interactions between attached and free substances such as growth of attached bacteria on dissolved substrate or attachment and detachment processes.

Equilibrium State Variables

Equilibrium state variables represent substances the reaction rates of which are so fast that their concentrations C_i can be described approximately by the equilibrium concentration, given as the solution of an algebraic equation, which can be expressed in the following form:

$$r_{eq}(C_i) = 0 \quad . \tag{4.18}$$

Note that this equation can depend on external variables and on other state variables as well, so that the solution of this equation can depend on time.

4.3.3 Biofilm Reactor Compartment

A biofilm consists of a solid matrix of particles attached to a surface and water filling the pore volume in between. A significant fraction of these particles consists of microorganisms, transforming substances dissolved in the water. If the layer of these particles is thick and dense enough and if the biochemical processes performed by the microorganisms are fast enough, transport of substances into and out of the film can become a limiting process. In this case, it is important to resolve the spatial gradients in the film perpendicular to the substratum. This results in a one-dimensional biofilm model for concentrations averaged over planes parallel to the substratum. Auch a biofilm model was originally formulated by Wanner and Gujer (1984, 1986), and was extended by Wanner and Reichert (1994). This latter formulation is used for the implementation of the biofilm reactor compartment in the program described in this report. It would seem to be natural to construct a compartment consisting of a biofilm alone. The approach chosen for the biofilm reactor compartment is to additionally include the water flowing over the film into the same compartment. The reason for this decision was to make the implementation of biofilm surface processes, such as attachment and detachment of particles possible, without the need of taking into account such biofilm specific processes in the general design of links. Therefore, the biofilm reactor compartment consists of a zone of mixed water called bulk volume and a biofilm zone, which resolves the film in the direction perpendicular to the underlying substratum. These two zones are separated by a boundary layer of negligible volume, which manifests its existence by a diffusive mass transfer resistance. The equations determining water flow and substance concentrations in the bulk volume as well as in the biofilm are summarized briefly in the following paragraphs. A more detailed discussion of these equations is given in Wanner and Reichert (1994).

Fig. 4.8 shows the pictogram of the biofilm reactor compartment. This pictogram shows the two zones, inflow (arrow), advective inflow and outflow connections (symbols at the left and right of the pictogram) and diffusive connections to the biofilm base (bottom) and to the bulk volume (top).

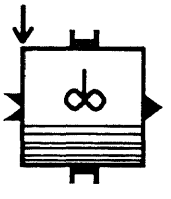


Fig. 4.8: Pictogram of a biofilm reactor compartment.

Water Flow

Because of the high particle density in biofilms, their volume cannot be neglected. Therefore, a distinction between the total volume of water plus particles and the pure volume of the water phase alone must be made within the biofilm reactor compartment. The variable Q (L^3T^{-1}) is used to describe total volumetric discharge of water and particles (this definition is necessary to make the equations to be developed rigorous; for the bulk water phase outside the film, however, the distinction of water volume and total volume can be neglected in nearly all cases of practical relevance). The inflowing discharge has the same contributions as in the case of the mixed reactor compartement described in section 4.3.2:

$$Q_{in} = Q_{in,comp} + \sum Q_{in,advlink} \quad . \tag{4.19}$$

The discharge Q through the compartment is equal to the inflow:

$$Q = Q_{in} \quad . \tag{4.20}$$

Because the main constituent of microorganisms growing in water is water itself, the concept of growth of microorganisms used in the biofilm model is that the cell membrane shifts the border between pure water phase and particulate phase. Therefore, growth of microorganisms does not change total volume of water plus particles. If this concept is applied to a reactor with constant volume (*confined* reactor), the volumetric outflow is equal to the inflow and therefore to the discharge Q through the reactor

$$Q_{ef} = Q \quad . \tag{4.21a}$$

Another type of biofilm reactor, the *unconfined* reactor, assumes a layer of water with fixed bulk volume flowing over the growing biofilm. In this case, due to the change of total volume of film plus water, the outflow is given by

$$Q_{ef} = Q - A\,u_L \quad , \tag{4.21b}$$

where A (L^2) is the surface area of the biofilm and u_L (LT^{-1}) is its interface velocity. Note that, in most applications, the correction Au_L is small compared to the discharge Q. Nevertheless, the small rate of change of reactor volume V_R (L^3)

$$\frac{dV_R}{dt} = Q_{in} - Q_{ef} \tag{4.22}$$

can lead to siginficant changes of total reactor volume.

Dynamic Volume State Variables

Dynamic volume state variables represent concentrations of particles and of dissolved substances in water. Because these two types of substances behave differently in the biofilm reactor compartment (the biofilm solid matrix consists of attached particles, whereas dissolved substances circulate in the water in between), the dynamic volume state variables must be divided into particulate and dissolved variables for the biofilm reactor compartment. The concentration of particles of type i averaged over planes parallel to the substratum and measured as mass per unit of total volume (including water and particles) is denoted by X_i (ML^{-3}), the concentration of the dissolved substance i averaged over planes parallel to the substratum and measured as mass per unit water phase volume by C_i (ML^{-3}). If ρ_{s_i} (ML^{-3}) denotes the density of particles of type i (mass per unit of particle volume), the volume fraction of the water phase ε_l (-) of total volume is given by

$$\varepsilon_l = 1 - \sum_{i=1}^{n_X} \frac{X_i}{\rho_{s_i}} \quad , \tag{4.23}$$

where n_X is the number of particulate variables. The equations governing temporal evolution of bulk water concentrations of particulate and dissolved substances outside the film is similar to the mixed reactor compartment (4.14) with the difference, that the factor ε_l has to be introduced into the equation for dissolved variables to correct for the water phase volume:

$$\frac{d}{dt}\left(V_B X_{B,i} \right) = I_{B,X_i} + V_B r_{X_i} \quad , \tag{4.24}$$

$$\frac{d}{dt}\left(V_B \varepsilon_{l\,B} C_{B,i} \right) = I_{B,S_i} + V_B r_{C_i} \quad , \tag{4.25}$$

where the index "B" indicates values within the bulk water volume V_B (L^3) (the factor ε_l is not needed in the term describing transformation processes, because the rates r_{C_i} are interpreted as rates per total volume). For the confined reactor, where biofilm and bulk water phase share constant total reactor volume, bulk volume V_B and total reactor volume V_R are given by

$$V_B = V_R - \int_0^{L_F} A \, dz' \quad , \quad V_R = \text{const.} \tag{4.26a}$$

where L_F (L) is the biofilm thickness, A (L^2) is the area of biofilm as a function of the space coordinate z perpendicular to the film-substratum interface (zero at the film-substratum interface) and the integration is over the whole biofilm thickness. For the unconfined reactor, bulk volume and reactor volume are given by

$$V_B = \text{const.} \quad , \qquad V_R = V_B + \int_0^{L_F} A \, dz' \quad . \tag{4.26b}$$

In this case, expansion of the film increases total reactor volume, because the volume of the bulk water phase flowing over the film is assumed to be constant.

Total mass flux of substances into bulk water volume of the biofilm reactor compartment has similar terms as the influx into the mixed reactor compartment given by equation (4.15):

$$I_{B,X_i} = I_{X_i,in,comp} + \sum I_{X_i,in,advlink} - I_{X_i,ef,advlink} + I_{L,X_i} \quad . \tag{4.27}$$

and

$$I_{B,C_i} = I_{C_i,in,comp} + \sum I_{C_i,in,advlink} - I_{C_i,ef,advlink} + \sum I_{B,C_i,difflink} + I_{L,C_i} \quad . \tag{4.28}$$

The contributions $I_{X_i,in,comp}$ and $I_{C_i,in,comp}$ are given by the user, $I_{X_i,in,advlink}$ and $I_{C_i,in,advlink}$ are determined from the advective links connected to the inflow connection of the compartment and $I_{B,C_i,difflink}$ is determined from the diffusive links connected to the bulk water volume of the compartment. Due to the different definitions of concentrations, the formulation of the output flux depends on the type of substances considered. For particulate components it is given by

$$I_{X_i,ef,advlink} = Q_{ef} \, X_{B,i} \quad , \tag{4.29}$$

for dissolved components

$$I_{C_i,ef,advlink} = Q_{ef} \, \varepsilon_{l\,B} \, C_{B,i} = Q_{ef} \, S_{B,i} \quad . \tag{4.30}$$

where

$$S_{B,i} = \varepsilon_{l\,B} \, C_{B,i} \tag{4.31}$$

is the concentration of dissolved substances per unit of total volume instead of the concentration $C_{B,i}$ per unit of water phase volume In equation (4.30), the distinction between these two concentrations can usually be neglected; it is only essential within the biofilm. The additional terms I_{L,X_i} and I_{L,C_i} in equations (4.29) and (4.30) (in comparison to equation (4.15) for the mixed reactor compartment) are the fluxes through the diffusive boundary layer out of the biofilm, which are discussed later.

Whereas the particulate substances are suspended in the bulk water volume, they form the solid matrix in the biofilm. Therefore, growth of particles in the depth of a biofilm leads to an advective displacement of the biofilm, if additional particulate volume is not completely at the expense of volume of the water phase. The displacement velocity u_F (LT^{-1}) at the location z is given as the total volume production between the substratum-film interface at z=0 and the position z:

$$u_F = \frac{1}{A}\int_0^z \left(\sum_{k=1}^{n_X} \frac{r_{X_k}}{\rho_{s_k}} + r_{\varepsilon_{l\,F}} \right) A \, dz' \quad . \tag{4.32}$$

The first term in the integrand is the volume production due to growth of particulate substances, the second term is the volume production of free water volume between the particles. It is a usual assumption, that the volume fraction of the water phase within the film ε_l remains constant. In this case, free water volume production is proportional to particulate volume production. Therefore, it is reasonable to write the volume production rate as a sum of this generic term and an excess rate $r_{\varepsilon_{l\,F}}'$ in the following way:

$$r_{\varepsilon_{l\,F}} = \frac{\varepsilon_{l\,F}}{1-\varepsilon_{l\,F}} \sum_{k=1}^{n_X} \frac{r_{X_k}}{\rho_{s_k}} + r_{\varepsilon_{l\,F}}' \quad . \tag{4.33}$$

With this notation, $r_{\varepsilon_{l\,F}}' = 0$ leads to a constant volume fraction $\varepsilon_{l\,F}$ in the film. This special case corresponds to the original biofilm model described in Wanner and Gujer (1986). With the aid of this equation, the advective velocity of the biofilm solid matrix can be rewritten in the form:

$$u_F = \frac{1}{A}\int_0^z \left(\frac{1}{1-\varepsilon_{l\,F}} \sum_{k=1}^{n_X} \frac{r_{X_k}}{\rho_{s_k}} + r_{\varepsilon_{l\,F}}' \right) A \, dz' \quad . \tag{4.34}$$

The mass balances of particulate substances, dissolved substances and water volume in the biofilm form a system of conservation laws of the form of equation (4.8) or (4.9). A complete definition of such a system requires specification of the three functions $\hat{\rho}$, $\hat{\mathbf{j}}$ and $\hat{\mathbf{r}}$. In the following, these quantities are written showing three components for particulate and dissolved substances and for the water fraction of the biofilm. Note that the first two of these components themself represent a series of different substances. The one-dimensional densities of the conservation law are given by

$$\hat{\rho} = \begin{pmatrix} A\, X_{F,i} \\ A\, \varepsilon_{l\,F}\, C_{F,i} \\ A\, \varepsilon_{l\,F} \end{pmatrix} \quad . \tag{4.35}$$

The first component of this expression is the one-dimensional density of particulate substances which is given by the product of the area A and the concentration $X_{F,i}$ in the biofilm averaged over a surface parallel to the substratum (index "F" means in the biofilm). The second component describes the one-dimensional density of dissolved substances given by the product of the area A, the water fraction $\varepsilon_{l\,F}$ and the concentration $C_{F,i}$ in the water averaged over a surface parallel to the substratum. The last component is the one-dimensional density of water volume given by the

product of teh area A and the water fraction $\varepsilon_{l\,F}$ averaged over a surface parallel to the substratum. The fluxes corresponding to the densities defined above are

$$\hat{\mathbf{j}} = \begin{pmatrix} A\,u_F\,X_{F,i} - A\,D_{F,X_i}\dfrac{\partial X_{F,i}}{\partial z} \\ -A\,(1-\varepsilon_{l\,F})\,u_F\,C_{F,i} - A\,\varepsilon_{l\,F}\,D_{F,C_i}\dfrac{\partial C_{F,i}}{\partial z} \\ A\,u_F\,\varepsilon_{l\,F} + A\sum_{k=1}^{n_X}\dfrac{D_{F,X_k}}{\rho_{s_k}}\dfrac{\partial X_{F,k}}{\partial z} \end{pmatrix} . \qquad (4.36)$$

The first component of this equation represents the flux of particulate substances in the biofilm. The first term of this component describes the advective motion of the biofilm matrix due to growth processes between the actual location and the biofilm base with velocity u_F as given by equation (4.34). The second term describes changes in a *diffusive* biofilm solid matrix by an effective diffusion process with diffusion coefficient D_{F,X_i} (L^2T^{-1}). Note that D_{F,X_i} is not molecular diffusion of particles in water, but it represents an effective diffusion process which models changes of the biofilm matrix due to detachment and reattachment of particles within the film. This effective diffusion process is only active for a *diffusive* solid matrix, it is zero in the case of a *rigid* solid matrix. The second component of equation (4.36) represents the flux of dissolved components in the biofilm. The first term of this component describes advective transport due to water flowing into the film to compensate for volume produced by growth processes. This term is necessary to make the equations rigorous; it can usually be neglected because, due to the slowness of microbial growth, diffusive processes are much faster. The second term of the second component describes diffusive flux of dissolved substances in the film. If turbulence is shielded in the film and if tortuosity is neglected, the diffusion coefficient D_{F,C_i} (L^2T^{-1}) is equal to the molecular diffusion coefficient of the substance in water. To model effects of turbulence penetrating into upper biofilm layers, the diffusion coefficient D_{F,C_i} can be increased near the biofilm surface. The last component of equation (4.36) represents the flux of free water volume (not the water flux!) in the biofilm. The first term describes advective motion together with the solid matrix with velocity u_F, the second term describes volume changes due to the effective diffusion process of particulate substances. This term is only in effect for a diffusive biofim matrix; it is zero for a rigid biofilm matrix. To complete the definition of the conservation law, the transformation rates must be specified:

$$\hat{\mathbf{r}} = \begin{pmatrix} A\,r_{X_i} \\ A\,r_{C_i} \\ A\,r_{\varepsilon_{l\,F}} \end{pmatrix} . \qquad (4.37)$$

All of these rates specify transformation as mass (or volume) per unit of total biofilm volume including the volume of particles. A more extended discussion of the concepts of the biofilm model and of the derivation of the functions $\hat{\rho}$, $\hat{\mathbf{j}}$ and $\hat{\mathbf{r}}$ in biofilms are given in Wanner and Reichert (1994).

Substitution of equations (4.35), (4.36) and (4.37) into equation (4.8) leads to the integral form of the conservation laws governing biofilm dynamics. The differential form is obtained by substituting equations (4.35), (4.36) and (4.37) into equation (4.9). With the aid of equations (4.33) and (4.34) the resulting set of equations can be transformed to give for the conservation of particulate substances

$$\frac{\partial X_{F,i}}{\partial t} = - u_F \frac{\partial X_{F,i}}{\partial z} + \frac{1}{A}\frac{\partial}{\partial z}\left(A\, D_{F,X_i} \frac{\partial X_{F,i}}{\partial z}\right) - r_{\varepsilon_{l\,F}}' X_{F,i} + \left(r_{X_i} - \frac{X_{F,i}}{1-\varepsilon_{l\,F}} \sum_{k=1}^{n_X} \frac{r_{X_k}}{\rho_{s_k}}\right) , \tag{4.38}$$

for the conservation of dissolved substances

$$\frac{\partial C_{F,i}}{\partial t} = \frac{1-\varepsilon_{l\,F}}{\varepsilon_{l\,F}} u_F \frac{\partial C_{F,i}}{\partial z} + \frac{1}{\varepsilon_{l\,F}} \sum_{k=1}^{n_X} \frac{r_{X_k}}{\rho_{s_k}} C_{F,i} + \frac{1}{\varepsilon_{l\,F}}\frac{1}{A}\frac{\partial}{\partial z}\left(A \sum_{k=1}^{n_X} \frac{D_{F,X_k}}{\rho_{s_k}} \frac{\partial X_{F,k}}{\partial z}\right) + \frac{1}{\varepsilon_{l\,F}}\frac{1}{A}\frac{\partial}{\partial z}\left(\varepsilon_{l\,F}\, D_{F,C_i} \frac{\partial C_{F,i}}{\partial z}\right) + \frac{1}{\varepsilon_{l\,F}} r_{C_i} , \tag{4.39}$$

and for the conservation of free water volume

$$\frac{\partial \varepsilon_{l\,F}}{\partial t} = - u_F \frac{\partial \varepsilon_{l\,F}}{\partial z} - \frac{1}{A}\frac{\partial}{\partial z}\left(A \sum_{k=1}^{n_X} \frac{D_{F,X_k}}{\rho_{s_k}} \frac{\partial X_{F,k}}{\partial z}\right) + (1-\varepsilon_{l\,F})\, r_{\varepsilon_{l\,F}}' . \tag{4.40}$$

These equations have the following meaning: Equation (4.38) describes time evolution of the concentration of a particulate species at a fixed position in the biofilm. The first term on the right hand side of this equation is the change in concentration due to advective motion of the biofilm matrix (caused by growth in underlying layers), the second term corresponds to changes due to a possible effective diffusion process of particulate substances in the biofilm, the third term models dilution of particles caused by growth of free water volume and the last term describes concentration changes due to growth processes of particulate components. This last term is given by the difference of the production rate of the substance and its dilution due to the growth of the other particulate species. Equation (4.39) describes time evolution of the concentration of a dissolved species at a fixed position in the biofilm. The first term on the right hand side of this equation corresponds to the advective flux of water transported into the biofilm to compensate for volume produced due to growth in underlaying biofilm layers. The second term describes growth of concentration due to growth of particulate species (since bacteria are treated as consisting mainly of water, movement of a cell membrane due to bacterial growth increases concentration of a dissolved substance not consumed in the growth process, i.e. not crossing the membrane). The next two terms describe changes in concentration of dissolved substances due to water flow caused by effective solid matrix diffusion of particles and due to diffusion of dissolved substances in the water phase, respectively. Finally, the last contribution is due to transformation processes consuming or producing the

dissolved component under consideration. Equation (4.40) describes changes of free volume between the biofilm solid matrix. The first term on the right hand side of this equation is due to advection of the solid matix, the second term due to effective diffusion of particles and the last term due to given changes in free volume.

In order to completely define the right-hand sides of the equations (4.38) to (4.40) the geometry of the biofilm must be defined by specifying the surface area of the film as a function of the distance from the substratum A(z). Examples of functions A(z) for important geometries are

$$\begin{aligned} &\text{plane film:} && A(z) = \text{const} \\ &\text{film on a cylinder:} && A(z) = 2\pi\,(z_{cy} + z)\,L_{cy} \\ &\text{film inside of a cylinder:} && A(z) = 2\pi\,(z_{cy} - z)\,L_{cy} \\ &\text{film on spheres:} && A(z) = 4\pi\,n_{sp}\,(z_{sp} + z)^2 \end{aligned} \qquad (4.41)$$

where z_{cy} (L) is the radius and L_{cy} (L) the length of a cylinder and z_{sp} (L) is the radius and n_{sp} the number of the spheres.

Time evolution of biofilm thickness is given by

$$\frac{dL_F}{dt} = u_L \quad , \qquad (4.42)$$

where due to attachment and detachment processes, the interface velocity u_L (LT^{-1}) is not equal to $u_F(L_F)$:

$$u_L = u_F(L_F) - u_{de} + u_{at} \quad , \qquad (4.43)$$

where u_{de} (LT^{-1}) and u_{at} (LT^{-1}) are the detachment and attachment velocities.

A *global detachment velocity* u_{de} can be specified by the user of the program (e.g. as a function of biofilm thickness, time, biofilm growth velocity, etc.):

$$u_{de} = u_{de}(L_F, t, u_F(L_F), \ldots) \quad . \qquad (4.44a)$$

In the case of a diffusive biofilm matrix, *individual detachment coefficients* k_{de,X_i} (LT^{-1}) for the variables X_i can be specified. In this case, the detachment velocity is given by

$$u_{de} = \frac{1}{1-\varepsilon_{l\,F}} \sum_{k=1}^{n_X} \frac{k_{de,X_k}\, X_{F,k}}{\rho_{s_k}} \quad . \qquad (4.44b)$$

The attachment velocity u_{at} is given by:

$$u_{at} = \frac{1}{1-\varepsilon_{l\,F}} \sum_{k=1}^{n_X} \frac{k_{at,X_k}\, X_{L,k}}{\rho_{s_k}} \quad , \qquad (4.45)$$

where k_{at,X_i} (LT^{-1}) are the attachment coefficients of the variables X_i. The following boundary conditions are necessary to define the connection of the film to the diffusive link at the biofilm base and to the bulk water compartment:

There is no flux of particulate substances into and out of the substratum:

$$- A D_{F,X_i} \frac{\partial X_{F,i}}{\partial z} (z=0) = 0 \quad . \tag{4.46}$$

This is only a boundary condition if the effective diffusivities D_{F,X_i} of particulate substances in the film are not zero, otherwise no boundary condition for particulate species at the biofilm base is necessary. The flux of dissolved substances at the biofilm base is given by the flux through the diffusive link connected to the biofilm base. This leads to the boundary condition

$$- A \varepsilon_{i,F} D_{F,C_i} \frac{\partial C_{F,i}}{\partial z} (z=0) = I_{F,C_i} \quad . \tag{4.47}$$

The calculation of the flux I_{F,C_i} through a diffusive link depends on the concentrations at both sides of the interface and is described in section 4.4. If there is no diffusive link connected to the base of the biofilm, I_{F,C_i} is zero.

Due to attachment and detachment processes, the boundary condition at the biofilm-bulk interface is more complicated. The net rate of transfer of particulate substances is given by

$$R''_{X_i} = A \left(k_{de,X_i} X_{F,i} (z=L_F) - k_{at,X_i} X_{L,i} \right) \quad , \tag{4.48}$$

where $X_{L,i}$ is the concentration of the suspended particulate substance i between the film and the boundary layer and k_{de,X_i} is equal to u_{de} for all X_i in the case of the specification of a global detachment velocity. Thus, the boundary conditions for particulate substances are that the flux into the film is equal to this transfer rate

$$A (u_F - u_L) X_{F,i} (z=L_F) - A D_{F,X_i} \frac{\partial X_{F,i}}{\partial z} (z=L_F) = R''_{X_i} \tag{4.49}$$

and that the flux through the boundary layer is equal to this transfer rate:

$$I_{L,X_i} = A \frac{X_{L,i} - X_{B,i}}{\kappa_{X_i}} = R''_{X_i} \quad , \tag{4.50}$$

where κ_{X_i} ($L^{-1}T$) is the empirical mass transfer resistance of the particulate substance i in the boundary layer (if κ_{X_i} is zero, this last boundary condition simplifies to $X_{L,i} = X_{B,i}$). In many cases, the mass transfer resistance κ_{X_i} is given as the ratio of the boundary layer thickness L_L divided by the diffusion coefficient D_{L,X_i} of the substance in the boundary layer:

$$\kappa_{X_i} = \frac{L_L}{D_{L,X_i}} \quad . \tag{4.51}$$

The two boundary conditions for the dissolved substances are the continuity of concentrations

$$C_{F,i}(z=L_F) = C_{L,i} \tag{4.52}$$

and the continuity equation for fluxes

$$I_{L,C_i} = \frac{C_{L,i} - C_{B,i}}{\kappa_{C_i}} = \left(- u_F + \varepsilon_{l\,F}(u_F - u_L) \right) C_{F,i}(z=L_F) - \varepsilon_{l\,F}\, D_{F,C_i} \frac{\partial C_{F,i}}{\partial z}(z=L_F) \ , \tag{4.53}$$

where κ_{C_i} ($L^{-1}T$) is the empirical mass transfer resistance of the dissolved substance i in the boundary layer (if κ_{C_i} is zero, this last boundary condition simplifies to $C_{L,i} = C_{B,i}$). In many cases, the mass transfer resistance κ_{C_i} is given as the ratio of the boundary layer thickness L_L divided by the diffusion coefficient D_{L,C_i} of the substance in the boundary layer:

$$\kappa_{X_i} = \frac{L_L}{D_{L,C_i}} \ . \tag{4.54}$$

Dynamic Surface State Variables

Dynamic surface state variables in a biofilm reactor compartment have the same meaning as in the mixed reactor compartment described in section 4.3.2. Concentrations of dynamic state variables and of equilibrium state variables used in transformation rates of dynamic surface state variables are evaluated in the bulk water volume. The simple concept of dynamic surface state variables is not of great interest in biofilm reactor compartments, because the biofilm itself describes similar phenomena at a much higher spatial resolution. If this spatial resolution is not adequate, it is better to use a mixed reactor compartment.

Equilibrium State Variables

Equilibrium state variables represent substances the reaction rates of which are so fast, that their concentrations C_i can be described approximately by the equilibrium concentration given as the solution of an algebraic equation:

$$r_{eq}(C_i) = 0 \ . \tag{4.55}$$

Note that this equation can depend on external variables and on other state variables as well so that the solution of this equation becomes dependent on time. As it is the case for dynamic volume state variables also equilibrium state variables are calculated in the bulk water volume and as a function of the space coordinate z in the biofilm.

4.3.4 River Section Compartment

A river section compartment models a reach of an open channel not containing significant abrupt hydraulic structures. The basic equations of one-dimensional open channel hydraulics are the equations of St. Venant (1871) carefully reviewed by Yen (1973, 1979). These equations can also be found in standard text books on open channel hydraulics, such as Chow (1959), Henderson (1966) or French (1985). Two approximations to these equations, the kinematic and diffusive wave equations, are used to describe water flow in the river section compartment. These equations together with the equations governing substance transport are reviewed briefly in this subsection.

Fig. 4.9 shows the pictogram of a river section compartment showing upstream (large arrow) and lateral inflow (small arrows) and the advective inflow and outflow connections (sylbols on the left and on the right).

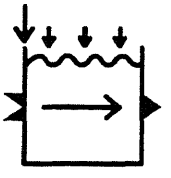


Fig. 4.9: Pictogram of a river section compartment.

Water Flow

Water flow in open channels is determined by the one-dimensional conservation laws of mass (or volume) and momentum. Derivation of these conservation laws for open channel flow can be found in one of the above mentioned text books or in Reichert (1992). Conservation of volume is given by a conservation law with

$$\hat{\rho} = A \quad , \quad \hat{j} = Q \quad , \quad \hat{r} = q_{lat} \quad , \tag{4.56}$$

where A (L^2) is the cross-sectional area, Q (L^3T^{-1}) is the discharge and q_{lat} (L^2T^{-1}) is the rate of lateral inflow per unit river length. The integral form of this conservation law is given by substituting equations (4.56) into equation (4.8). The differential form, obtained by substituting equations (4.56) into equation (4.9), has the form:

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = q_{lat} \quad . \tag{4.57}$$

This equation must be combined with an equation which approximates the momentum balance.

The simplest description fo open channel hydraulics, the kinematic wave approximation to the St. Venant equations, assumes the accelerating gravitational force due to channel slope to be equilibrated by friction always and everywhere:

$$S_o = S_f \quad . \tag{4.58a}$$

In this equation, S_o (-) is the slope of the channel and S_f (-) is the so-called friction slope, defined as the non-dimensional ratio of the friction force and the gravity force. This approximation makes the description of backwater effects due to hydraulic controls impossible. The kinematic wave approximation of open channel flow can only be employed, if the geometry of the river bed is regularly enough to make the definition of a local channel slope S_o possible (especially, this slope must have the same sign everywhere). The classical application of the kinematic wave approximation is to prismatic channels without important influences of hydraulic controls.

The approximation of the momentum equation corresponding to the diffusive wave approximation sets the slope of water level elevation equal to the friction slope:

$$-\frac{\partial z_o}{\partial x} = S_f \quad . \tag{4.58b}$$

Here, z_o (L) is the water level elevation. Due to the consideration of hydrostatic pressure differences, this equation makes the calculation of backwater effects due to weirs possible. Equations (4.57) and (4.58a) or (4.58b) are usually solved with the independent variable z_o or

$$h_o = z_o - z_B \quad , \tag{4.59}$$

where h_o (L) is the maximum water depth and z_B (L) is the elevation of the deepest point of the river bed. The cross-sectional area becomes then a function $A(z_o,x)$ or $A(h_o,x)$, which characterises the geometry of the river bed. This function must be substituted into equation (4.57) together with an expression for discharge Q (L^3/T) obtained by solving one of the equations (4.58a) or (4.58b) for Q. If the functions Q_N and Q_D are defined as the solutions of these implicit equations:

$$Q_N(z_o,x): \qquad S_o(x) = S_f(z_o,Q_N(z_o,x),x) \tag{4.60a}$$

and

$$Q_D\left(z_o,\frac{\partial z_o}{\partial x},x\right): \qquad -\frac{\partial z_o}{\partial x} = S_f\left(z_o,Q_D\left(z_o,\frac{\partial z_o}{\partial x},x\right),x\right) \tag{4.60b}$$

equation (4.57) can be written in the form

$$\frac{\partial A}{\partial z_o}\frac{\partial z_o}{\partial t} + \frac{\partial Q_N}{\partial z_o}\frac{\partial z_o}{\partial x} + \frac{\partial Q_N}{\partial x} = q_{lat} \tag{4.61a}$$

for the kinematic case and

$$\frac{\partial A}{\partial z_o}\frac{\partial z_o}{\partial t} + \frac{\partial Q_D}{\partial z_o}\frac{\partial z_o}{\partial x} + \frac{\partial Q_D}{\partial\left(\frac{\partial z_o}{\partial x}\right)}\frac{\partial^2 z_o}{\partial x^2} + \frac{\partial Q_D}{\partial x} = q_{lat} \qquad (4.61b)$$

for the diffusive case (the formulation of these equations with z_o instead of h_o eliminates the problem of determining S_o for the diffusive case: note that $\partial h_o/\partial x = \partial z_o/\partial x - dz_B/dx = \partial z_o/\partial x + S_o$). The impossibility of describing backwater effects with the kinematic approximation is reflected by the fact, that equation (4.61a) only requires an upstream boundary condition for uniquely defining a solution, whereas for the solution to equation (4.61b) for the diffusive case, upstream and downstream boundary conditions are required. The upstream bondary condition for both equations (4.61a) and (4.61b) is total discharge into the river section, which is the sum of inflow explicitly specified for the river section and contributions from other compartments connected with an advective link to the upstream end of the river section compartment. The second boundary condition necessary only for the diffusive case (4.61b) consists in the specification of water level at the downstream boundary of the river section. This downstream water level is given as the maximum of the water level selected by the user, the critical water level and the start water level of a possible river section compartment connected with an advective link to the downstream end of the river section compartment. This procedure of finding the downstream boundary condition makes the definition of a downstream hydraulic control specific for the river section possible, but it also takes into account backwater effects from downstream river sections, which influence the end water level of the current section. There are three possibilities for defining the hydraulic control at the end of a section: It is possible to explicitly specifying downstream water level or to use normal or critical water level, $z_{o,N}$ or $z_{o,C}$, defined by

$$z_{o,N}(Q,x): \quad S_f(z_{o,N},Q,x) = S_o(x) \qquad (4.62)$$

and

$$z_{o,C}(Q,x): \quad Fr^2 = \frac{Q^2\, w(z_{o,C},x)}{g\, A(z_{o,C},x)^3} = 1 \quad , \qquad (4.63)$$

where Fr is the Froude number, w surface width and g gravitational acceleration. Explicit specification of water level allows the user to implement arbitrary hydraulic controls by their water level - discharge relationship, critical water depth describes a drop and normal water level (approximately) describes an end of a river section without influence of downstream hydraulic controls. Note that coupling of water level between adjacent river sections couples equations (4.61b) for the corresponding sections with two algebraic equations (for discharge and water level) and makes individual solution of the differential equations for the river sections impossible.

As the initial conditions for solving the equations of open channel hydraulics, the steady state solution corresponding to inflow at the initial time is used.

To complete the definition of the set of equations (4.57) and (4.58a) or (4.58b), an empirical parameterization of the friciton slope as a function of mean flow and river

geometry characteristics is needed. The expression most often used for the friction slope is given by

$$S_f = \frac{1}{K_{st}^2} \frac{1}{R^{4/3}} \frac{Q^2}{A^2} = n^2 \frac{1}{R^{4/3}} \frac{Q^2}{A^2} \quad , \tag{4.64}$$

where K_{st} ($L^{1/3}T^{-1}$) is the empirical friction coefficient according to Strickler, n ($L^{-1/3}T$) is an alternative friction coefficient according to Manning, R = A/P (L) is the hydraulic radius and P (L) is the wetted perimeter length. An alternative expression due to Darcy and Weisbach is

$$S_f = \frac{f}{8g} \frac{1}{R} \frac{Q^2}{A^2} \quad , \tag{4.65}$$

with the non-dimensional friction factor f.

To calculate river hydraulics, a set of three equations must be solved at each grid point: The conservation law (4.56) leads to a differential equation (4.61a) or (4.61b) for A, equations (4.60a) or (4.60b) together with an expression for the friction slope form an algebraic equation for Q and, finally, definition of river geometry specified as the function $A(z_o,x)$ is an implicit equation for z_o.

Dynamic Volume State Variables

One-dimensional transport of substances in a river consists of the processes advection with the mean velocity (calculated with the techniques described above) and longitudinal dispersion. Longitudinal dispersion is the process of spreading of a substance pulse released into a river. Molecular and turbulent diffusion would also lead to such a spreading effect, but it is the dispersion mechanism of the transverse velocity profile which is the dominant process responsible for longitudinal spreading: Due to transverse turbulent diffusion, the substance released into the river moves between fast flowing zones near the middle of the river and stagnant zones near the river banks. This leads to the observed longitudinal spreading. According to Fischer (1967) or Fischer et al. (1979), after an initial period which needs two-dimensional description, the net effect of longitudinal dispersion can be described by an effective diffusion process. Thus river transport of substance i is given by a conservation law with

$$\hat{\rho} = A C_i \quad , \quad \hat{j} = Q C_i - A E \frac{\partial C_i}{\partial x} \quad , \quad \hat{r} = A r_i + q_{lat} C_{i,lat} \quad , \tag{4.66}$$

where C_i (ML^{-3}) is the cross-sectionally averaged concentration of substance i, E is the longitudinal dispersion coefficient and $C_{i,lat}$ is the concentration of substance i in lateral inflow ($q_{lat} > 0$) or the concentration C_i in the river in the case of lateral outflow ($q_{lat} < 0$). Substituting equations (4.66) into equation (4.9) leads to the differential form of the substance transport and transformation equation:

$$\frac{\partial(AC_i)}{\partial t} = -\frac{\partial(QC_i)}{\partial x} + \frac{\partial}{\partial x}\left(AE\frac{\partial C_i}{\partial x}\right) + A\,r_i + q_{lat}\,C_{i,lat} \quad . \tag{4.67}$$

According to Fischer (1967) and Fischer et al. (1979), an estimate of the dispersion coefficient E is given by

$$E = c_F \frac{w^2\,(Q/A)^2}{u^*\,d} \quad , \tag{4.68}$$

where $c_F \approx 0.011$ (-) is a non-dimensional coefficient not varying too much from one river to another, w (L) is the width of the river, $u^* = \sqrt{\tau_o/\rho} \approx \sqrt{g\,d\,S_f}$ (LT^{-1}) is the shear velocity (τ_o is the bottom shear stress) and d = w/A is the mean river depth.

Equation (4.67) is solved using the upstream boundary condition

$$I_i = Q\,C_i - A\,E\frac{\partial C_i}{\partial x} \quad , \tag{4.69}$$

where I_i is the total mass flux of substance i into the river section and the right-hand side of this equation is evaluated at the upstream boundary of the reach. This mass flux is the sum of the mass flux explicitly specified as input into the river section and the contributions of all advective links coupled to the upstream end of the river section:

$$I_i = I_{i,in,comp} + \sum I_{i,in,advlink} \quad . \tag{4.70}$$

An additional downstream boundary condition is necessary due to the second order space derivative in equation (4.67). The physical meaning of this downstream boundary condition is difficult to understand because longitudinal dispersion is caused by differences of advective velocities over the cross section and needs no downstream boundary condition in a mechanistic three dimensional description (if longitudinal molecular and turbulent diffusion is neglected). The need for this boundary condition is caused by the approximation of the effect of longitudinal dispersion by an effective diffusion process. As discussed by Danckwerts (1953), Wehner and Wilhelm (1956), Pearson (1959) and Bischoff (1961), a possible solution of this problem is the use of the downstream boundary condition

$$\frac{\partial C_i}{\partial x} = 0 \quad . \tag{4.71a}$$

An alternative approach, which seems to be more reasonable in the case of an advection-dominated transport, is the "transmission boundary condition"

$$\frac{\partial^2 C_i}{\partial x^2} = 0 \tag{4.71b}$$

originally due to Shamir and Harleman (1967). This second equation is employed by the program described in this report. Note that in contrast to the hydraulic case of the diffusive wave approximation of open channel flow, both of these boundary conditions avoid mutual coupling of the transport equations of different river sections. The mass flux over the downstream boundary, which is required for further downstream connections, is given by equation (4.70) in which now the right-hand side is evaluated at the downstrem boundary of the reach.

As the initial condition for solving equation (4.67), a concentration specified by the user or the steady state concentration corresponding to water flow and substance input at the initial time can be used.

Dynamic Surface State Variables

Dynamic surface state variables represent substances attached to the river bed (sediment, sessile microorganisms or substances adsorbed to the bed material). To facilitate formulation of mass balances, it is useful to interpret values of dynamic surface state variables not as surface densities but as mass per unit river length (otherwise there are problems with temporal changes of wetted perimeter length or surface width). The time evolution of such attached masses per unit river length C_i (ML^{-1}) is given by the process rate only:

$$\frac{\partial C_i}{\partial t} = r_i \quad . \tag{4.72}$$

Note that the process rate r_i ($ML^{-1}T^{-1}$) can depend on concentrations of substances dissolved or suspended in the water as well as on dynamic surface state variables. This allows the users of the program to model interactions between attached and free substances, such as sedimentation, resuspension or growth of sessile microorganisms on substrate dissolved in the river.

Equilibrium State Variables

Equilibrium state variables represent substances the reaction rates of which are so fast, that their concentrations C_i can be described approximately by the equilibrium concentration given as the solution of an algebraic equation:

$$r_{eq}(C_i) = 0 \quad . \tag{4.73}$$

Note that this equation can depend on external variables and on other state variables as well so that the solution of this equation becomes dependent on time. As is the case for dynamic volume and surface state variables also equilibrium state variables are calculated as a function of the space coordinate x along the river.

4.4 System of Links

Links are used to connect different compartments to the desired spatial configuration representing the aquatic system to be investigated. In this report, only zero-dimensional links, which represent point connections of compartments, are considered. Two types of them are distinguished:

- advective links,
- diffusive links.

Advective links model water flow and advective substance transport between compartmets, diffusive links represent diffusive boundary layers or membranes between compartments, which can be penetrated diffusively by some substances. These two link types are discussed in the two subsection of this section.

4.4.1 Advective Link

Fig. 4.10 shows pictograms of possible advective links. The form of the connections indicate at which connections these links can be connected to the compartments the pictograms of which are shown in Figs. 4.7 - 4.9.

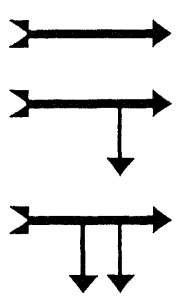


Fig. 4.10: Pictograms of advective links with different numbers of bifurcations.

Water and substance inflow to an advective link is determined by the output of the compartment connected to the input connection of the link. Advective links allow the users of the program to define bifurcations, where a part of water or substance flow bifurcates to another compartment or leaves the system. The rest of water and substance (dynamic volume state variables) flow given by

$$Q_{ef} = Q_{in} - \sum Q_{bif} \tag{4.74}$$

and by

$$I_{i,ef} = I_{i,in} - \sum I_{i,bif} \tag{4.75}$$

(the sums extend over all bifurcations) leaves the link at its main effluent, which itself can be connected to a compartment or can leave the system. Junctions can be modelled using advective links by connecting more than one output connection of advective links to the input connection of a compartment.

4.4.2 Diffusive Link

Fig. 4.11 shows the pictogram of a diffusive link. The form of the connections indicates at which points these links can be connected to the compartments, the pictograms of which are shown in Figs. 4.7 and 4.8.

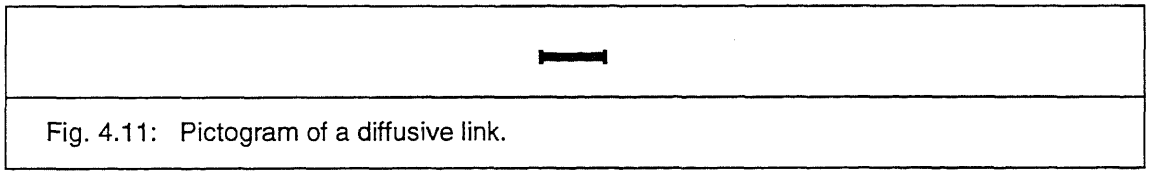


Fig. 4.11: Pictogram of a diffusive link.

A diffusive link models a diffusive boundary layer or a membrane between two compartments, which can be penetrated by some substances represented by dynamic volume state variables. There is no water flow through diffusive links. The flux of substance i through a diffusive link is given by the product of the mass exchange coefficient $q_{ex,i}$ (L^3T^{-1}) of the substance i and the difference of the concentrations at both sides of the membrane:

$$I_i = q_{ex,i} (f_i C_{i,1} - C_{i,2}) \quad , \tag{4.76}$$

where $C_{i,1}$ and $C_{i,2}$ are the concentrations in the two compartments connected by the link at the place of the connection and f_i is a conversion factor converting concentration $C_{i,1}$ to an equivalent concentration corresponding to the medium in compartment 2. For the usual case of a diffusive link between two compartements filled with water, this conversion factor is equal to unity. If compartment 1, however, is a mixed reactor compartment modelling a gas phase, the conversion factor has to be set to the inverse of the non-dimensional Henry coefficient of substance i:

$$f_i = \frac{1}{H_i} \quad . \tag{4.77}$$

In many cases, the mass exchange coefficient $q_{ex,i}$ of the substance i is given as the product of a surface area A (L^2) and a mass transfer coefficient k_i (LT^{-1})

$$q_{ex,i} = A\, k_i \quad , \tag{4.78}$$

which itself is often given as the ratio of the diffusion coefficient D_i of the substance in the boundary layer and the thickness L_M of the membrane

$$k_i = \frac{D_i}{L_M} \quad . \tag{4.79}$$

Instead of these expressions, an empirical exchange coefficient can be used.

5 Selection of Program Tasks

In chapter 2, a review is given on methods of model-based systems analysis of environmental systems. A comparison of the analyses required with the capabilities of commonly employed computer programs was given in section 3.1. This comparison showed that such programs only partially cover the needs of environmental systems analysis. The requirements of a more universal identification and simulation program, as needed for the analysis of environmental systems, are stated in section 3.2. In the present chapter, it is shown that the methodological requirements can be satisfied by providing the four program capabilities of

- simulation,
- identifiability analysis,
- parameter estimation,
- uncertainty analysis.

For each of these tasks, a technique is selected from the system analytical methods described in chapter 2. This selection is based mainly on practical considerations. In each of the sections of this chapter, the method selected for one of the program tasks listed above is discussed. The goal is not to develop a general system identification tool, but to design a "minimal" computer program fulfilling requirements S1 to S7, listed in section 3.2. The criteria for the selection of techniques are:

- wide acceptability and universality,
- flexibility of application and suitability of the technique for designing a user interface that is easy to handle,
- numerical efficiency.

The reason for designing a "minimal" program satisfying the requirements stated in section 3.2 is to gain practical experience with the new program concept before continuing development. Those techniques not selected in the present chapter which are thought to be most important for future program improvements are further discussed in chapter 9.

5.1 Simulation

Simulations are calculations of the temporal evolution of models. It is an evident task of a system identification program to provide the capability to perform simulations. All of the requirements S1 to S7 listed in section 3.2 are implicitly based on simulations. ***On the methodological level discussed in this chapter, there is no choice of methods for performing simulations, because the deterministic nature of the models uniquely defines this task*** (the numeric level is discussed in chapter 6). In order to make system identification possible, the most important feature of simulations is that models can easily be modified and that a large class of models can be formulated (requirements S1 and S2). The design of a general model structure satisfying this desire is discussed in chapter 4. The provision of plotting facilities for comparing the results of different models and for comparing model calculations with measurements is briefly discussed in chapter 7.

5.2 Identifiability Analysis

Since theoretical identifiability analysis is not based on measurements and uses completely different techniques than the other components of an identification and simulation program, it is excluded from the goals of such a program (cf. sections 2.3.2 and 3.2). As described in section 2.3.2, practical identifiability analysis can be conducted using sensitivity analysis or parameter covariance estimation. Both capabilities should be provided by the program (cf. requirements S5 and S6 in section 3.2). Since parameter covariance estimation is intimately related to parameter value estimation, it is treated in the next section. In this section, therefore, the decision to employ either linear or nonlinear sensitivity analysis must be made. Linear sensitivity analysis has important practical advantages in comparison to nonlinear sensitivity analysis. In order to perform linear sensitivity analysis with respect to m parameters, m+1 simulations must be carried out (one basic simulation and m additional simulations, each with one parameter slightly changed). If all states of the system for all of these m+1 simulations are stored, any of the sensitivity functions (2.2a) to (2.2d) for any combination of variables and parameters (out of the set of m parameters previously specified) can be calculated at any location. Nonlinear sensitivity analysis for several parameters, carried out using the Monte Carlo technique, requires execution of hundreds or thousands of simulations (depending on the number of parameters and on the desired accuracy of the calculated distributions). Because complete storage of such a number of states is in most cases impossible, the variables and locations in space and time for which and where results are desired, must be specified in advance. The whole calculation must be repeated when the desire for results at other locations arises later. These problems make linear sensitivity analysis the much more efficient and flexible of the two techniques. These practical advantages of linear sensitivity analysis must be compared with the theoretical disadvantage of not considering the nonlinearity of dependences present in the model. The unidentifiable case, as discussed in section 2.3.2, is characterized by a local submanifold in parameter space leading to the same model results. Because such a submanifold is manifested by its tangent plane in the linear approximation of the model, linear identifiability analysis is sufficient for the detection of combinations of unidentifiable parameters. Usual estimates of parameter covariances, as described in chapter 2, are also based on a linear approximation to the model. ***Because of its practical advantages, and since it is suitable for interpreting identifiability problems, linear sensitivity analysis is selected as the adequate method for identifiability analysis.*** Two important comments, however, must be made on this selection:

- Either linear or nonlinear sensitivity analysis can be used to detect the existence of unidentifiable parameters manifested by a local submanifold passing through the current parameter values in parameter space. Both methods test local identifiability as defined in section 2.3.2. There are no general methods for testing global identifiability. It is recommended to restart the numerical algorithm which performs parameter estimation with different sets of initial values to check for global convergence of the procedure. Such global convergence would make global identifiability probable (cf. discussion in the next section).
- It is important to note that, although linear sensitivity analysis is sufficient for local identifiability analysis, it can yield incorrect results for other applications of

sensitivity analysis (e.g. uncertainty analysis, parameter-result covariance estimation, etc.; cf. Gardner et al., 1981).

Because of its utility, efficiency and flexibility, linear sensitivity analysis must certainly be implemented in an identification program. The additional availability of nonlinear techniques is desirable, but in the sense of a minimal first version, linear sensitivity analysis must be preferred.

5.3 Parameter Estimation

The possibility of including several target variables in a single parameter estimation, as stated in requirement S6 (cf. section 3.2), makes the application of the simplest parameter estimation technique, i.e. minimizing the sum of squared residuals according to equation (2.19), impossible (different variables may have different dimensions). Of the remaining parameter estimation techniques, the weighted least squares estimation method, which determines parameter values by minimizing the function χ^2 according to equation (2.12), is by far the most widely applied technique. As described in section 2.3.3, this method makes it possible to estimate the parameter variance-covariance matrix, which gives useful information on parameter identifiability (standard deviations and correlations). ***Because weighted least squares estimation is universally applicable and makes it possible to estimate the parameter variance-covariance matrix, this method is selected for parameter estimation in the program.*** The application of the more general methods of maximum likelihood estimation and of Bayesian estimation usually fail because the required probability distributions are unavailable. As is discussed in chapter 9, robust estimation techniques, which allow slight violations of assumed distributions, are in most cases a very useful generalization. Equation (2.18) is used for the estimation of the parameter variance-covariance matrix instead of equation (2.17). Although the derivation of this equation is less rigorous, it gives more reasonable variance and covariance estimates if the errors of the data are not well known. The problem of interpreting systematic errors as statistical errors intrinsic to this equation can be diminished by performing a χ^2-test on the results of the fit and by checking the residuals for correlation (furthermore, because the value of χ^2 as well as the number of data points and parameters are known, an interested user can easily calculate the variance-covariance matrix according to equation (2.17)).

Note that the model formulation framework described in chapter 4 provides a universal basis for specifying parameter estimations. Any constant variable which can occurs in model formulation, in the formulation of initial state or in the formulation of variables describing external infuences, can be used as a parameter to be estimated. Because parameters (and other experiment-specific variables) can be made dependent on the program variable "Calculation Number" (e.g. with the aid of a variable list variable with the argument "Calculation Number"), which distinguishes different calculations, experiment-specific and universal parameters can easily be realized. Even parameters characterizing subgroups of experiments can be defined. Unknown functions can either be parameterized as formula variables using constant variables as parameters, or they can be formulated as a variable list variable with constant variables corresponding to the values of the argument. Due to this universality in model structure, most of the properties demanded by requirement S6 are already present. The implementation of parameter estimation must allow the user to combine several calculations and several target variables for each calculation into one parameter estimation in order to fulfill the rest of requirement S6.

Selection of the parameter estimation technique consists of selecting the "loss" function (dependent on data and parameters), the minimum of which (as a function

of the parameters for given data) determines the "optimal" parameters. Due to the possible existence of several local minima or the existence of "valleys" of minima, such a minimization problem can be very difficult to solve. The numerical techniques applied for solving this problem are discussed in section 6.5. It should be noted, however, that numerical problems in finding the minimum of the loss function for parameter estimation almost always indicate a badly posed estimation problem. The high flexibility in formulating models and using parameters as discussed in the previous paragraph may tempt one to overparameterize a model (this is a wide-spread habit in the environmental sciences, cf. Beck, 1983; Young, 1983). The program user is responsible for formulating an identifiable problem. The methods selected in section 5.2 and a careful check for large magnitudes of correlation coefficients (near +1 or -1) may help in detecting local identifiability problems. Checking for global identifiability is very difficult; restarting the parameter estimation process with different sets of initial values for the parameters is recommended to obtain increased confidence in global identifiability.

5.4 Uncertainty Analysis

The mathematical techniques of uncertainty analysis discussed in section 2.4.2 are very similar to those of sensitivity analysis discussed in section 2.3.2. Therefore, the practical advantages and disadvantages of linear and nonlinear techniques discussed in section 5.2 are also the same. As a further argument, the implementation of linear uncertainty analysis according to equations (2.21) and (2.22) is extremely simple if the derivatives $\partial y/\partial p_i$ are already available from performing linear sensitivity analysis according to equations (2.2a) to (2.2d). ***For practical reasons, linear uncertainty analysis is selected as the method to be implemented in the first version of the program.***

It should be noted that the selection of linear uncertainty analysis represents a much more severe restriction in program generality than does the selection of linear sensitivity analysis to accomplish identifiability analysis. The linear sensitivity functions (2.2a) to (2.2d) are correct and possess reasonable validity even in the nonlinear case. They suffice to detect locally unidentifiable parameters, whereas in the case of strong nonlinearities (defined by the fact that nonlinear effects are siginificant on the scale of the uncertainty of model parameters), equation (2.21) or parameter-result correlation coefficients can give incorrect results (e.g. Gardner et al., 1981). For this reason, ***the decision to implement linear uncertainty analysis is provisional and will be revised in chapter 9.*** If a sufficient number of Monte Carlo simulations are performed, nonlinear uncertainty analysis gives the correct probability distributions of calculated results caused by uncertain model parameters (for given distributions and correlations of the parameters). It should, however, be remembered that uncertainty in the model parameters are only one source of total uncertainty, as discussed in section 2.4. Model predictions, even when uncertainty calculated using one of the methods discussed above is taken into account of, should be very carefully interpreted, because the quantification of a (possibly) major source of uncertainty, the uncertainty in the model structure, is extremely difficult to quantify (cf. section 2.4).

6 Numerical Algorithms

In this chapter the methods are discussed which were chosen for numerically solving the equations given in chapter 4 in order to perform the program tasks selected in chapter 5. These methods should fulfill the following requirements (numbers of requirements from section 3.2 are given in parantheses):

- transport equations must be solved with methods accounting for different types of partial differential equations in different compartments (S1),
- the flexible formulation of transformation processes by the user of the program can lead to stiff and nonlinear differential equations that must be solved with the numerical algorithm (S2),
- ease of operation of the program requires a robust numerical algorithm, which does not need much tuning by the user (T1),
- because sensitivity analyses and parameter estimations require many integrations to be executed, an efficient integration algorithm can save much computation time (T3),
- the choice of the method for numerically solving the model equations should already consider an extension of the program to more compartment types (T4).

In contrast to space discretization, which must be made in a compartment-specific way in order to account for different types of partial differential equations, time discretization cannot profit from compartment-specific knowledge, because transformation processes can be specified by the user of the program. For this reason, the ***method of lines solution technique*** was chosen for numerically solving the partial differential equations (e.g. Schiesser, 1991). In this technique, the partial differential equations of the model are converted into a set of ordinary differential equations and algebraic equations (due to boundary equations) by discretization in space only. In this discretization step, different techniques can be applied to different compartment types. The spatially discretized set of equations together with the ordinary differential equations of the model and the algebraic equations due to equilibrium processes can be written in the general form of a (ordinary) ***differential-algebraic system of equations***

$$\mathbf{G}\left(\mathbf{y}, \frac{\partial \mathbf{y}}{\partial t}, t\right) = 0 \quad , \tag{6.1}$$

where $\mathbf{y}$ is the state vector of the system and $\mathbf{G}$ is a vector-valued function characterizing the system. This set of equations is integrated in time in a second step using a common time discretization technique for all compartments.

The implicit form of differential-algebraic systems according to equation (6.1) describes a large class of problems, but it is more difficult to specify initial conditions for equations of the form of (6.1) than for a system of explicit ordinary differential equations, because starting temporal integration requires consistent initial conditions, which fulfill the algebraic equations contained in (6.1) (and also possible implicit

differential equations, which do not occur in the present context). The algorithms used for making user-specified initial conditions consistent are described in section 6.1. Although spatial discretization techniques are chosen to be compartment-specific, the resulting set of ordinary differential and algebraic equations can be solved using a universal temporal integration algorithm. In section 6.2 the choice of this algorithm is discussed. The methods applied for spatial discretization are discussed in section 6.3. After the introduction of general concepts concerning conservative discretization methods for conservation laws, flux limiter methods and adaptive grid techniques used for the discretization of the biofilm reactor compartment and of the river section compartment are described. In section 6.4 it is described how the partial derivatives needed for identifiability and uncertainty analysis are calculated, and in section 6.5, the algorithms used for parameter estimation are discussed.

6.1 Finding Consistent Initial Conditions

To start time integration of a differential-algebraic system of equations defined by a function $\mathbf{G}(\mathbf{y},\partial\mathbf{y}/\partial t,t)$ and formulated according to equation (6.1), a consistent initial condition

$$\mathbf{y}(t_o) \ , \quad \frac{\partial \mathbf{y}}{\partial t}(t_o) \tag{6.2}$$

fulfilling the system of nonlinear equations

$$\mathbf{G}\left(\mathbf{y}(t_o),\frac{\partial \mathbf{y}}{\partial t}(t_o),t_o\right) = 0 \tag{6.3}$$

is required. In the present context, the differential-algebraic system to be solved results from discretization of a system of explicit partial differential equations combined with explicit ordinary differential equations and with (implicit) algebraic equations. Such a system is explicit in the differential equations and, according to Brenan et al. (1989), is called semi-implicit. For such a system, the vector-valued function **G** contains only components of one of the following two forms:

$$G_i\left(\mathbf{y},\frac{\partial \mathbf{y}}{\partial t},t\right) = F_i(\mathbf{y},t) - \frac{\partial y_i}{\partial t} \tag{6.4a}$$

(implicitly formulated explicit ordinary differential equations) and

$$G_i\left(\mathbf{y},\frac{\partial \mathbf{y}}{\partial t},t\right) = G_i'(\mathbf{y},t) \tag{6.4b}$$

(algebraic equations). Components of the form of equations (6.4a) correspond to discretized partial differential equations or to ordinary differential equations, components of the form of equation (6.4b) result from equilibrium processes or from boundary conditions of partial differential equations.

Initial conditions given by the user of the program may violate the consistency condition formulated by equation (6.3). The special form of the function **G** given by the equations (6.4a) and (6.4b) makes the following strategy possible to make the given inititial condition consistent: For the indices i for which G_i has the form of the equations (6.4b), the set of equations (6.4b) is solved for the variables $y_i(t_o)$ belonging to the same index set. This is a well-posed algebraic problem for the algebraic equations resulting from boundary conditions of partial differential equations; it is in the responsibility of the users of the program that this is also the case for their formulation of equilibrium processes. When the set of equations (6.4b) has been solved, the initial conditions for the time derivatives $\partial y_i/\partial t(t_o)$ for indices corresponding to equations of the form (6.4a) can be determined by evaluating these equations with $\partial y_i/\partial t(t_o)$ set to zero. The initial conditions for the derivatives $\partial y_i/\partial t(t_o)$ for indices corresponding to equations of the form (6.4a) are irrelevant, because **G** does not

depend on these values. Because the strategy of making initial conditions consistent as described in this paragraph only changes values $y_i(t_o)$ corresponding to algebraic equations, this strategy leads to the ***consistent initial condition with as few changes as possible to the initial condition specified by the user***.

There is another strategy leading to an alternative consistent initial condition: Setting all components of $\partial\mathbf{y}/\partial t(t_o)$ to zero and solving the whole set of equations (6.3) for $\mathbf{y}(t_o)$ leads to a ***steady state initial condition***. In some cases, such a steady state solution can be a reasonable initial condition. The user who wants to use this type of initial condition must carefully check, if such a solution really exists. As an additional problem, numerically finding this initial condition is much more difficult than in the case of the alternative initial condition discussed before. Therefore, the use of the second type of initial condition is reserved to special situations.

For both possibilities of finding consistent initial conditions discussed in the preceeding paragraphs, a set of nonlinear algebraic equations must be solved numerically. A modified Newton algorithm as described by Stoer (1983) is used to perform this task. There are two essential changes to the original Newton algorithm: Global convergence behavior is improved by a flexible step reduction mechanism guaranteeing monotonic decrease of the error term of the equations, and calculation of the matrix of partial derivatives at each step is simplified by an updating mechanism due to Broyden (1965). All solutions of systems of linear equations are solved using the LINPACK library due to Dongarra et al. (1979).

6.2 Temporal Discretization

Spatial discretization of a system of partial differential equations leads to a system of ordinary differential equations, which is often coupled with algebraic equations describing boundary conditions of the partial differential equations. Due to the large range of time scales and/or the nonlinearity of natural processes and due to effects of spatial discretization such a system of ordinary differential equations is usually stiff. Solutions to stiff systems of differential equations are characterized by a short transient phase during which fast convergence to a much slower varying solution occurs (e.g. Gear, 1969). This behavior of the solutions causes difficulties for the numerical solution. For this reason, a general purpose simulation program must use a time integration algorithm which is designed to overcome this problem.

Fig. 6.1 shows a typical example of the behavior of the solutions to a stiff system of differential equations

$$\frac{\partial \mathbf{y}}{\partial t} = \mathbf{F}(\mathbf{y},t) \quad . \tag{6.5}$$

If such a system is solved with an explicit numerical technique, such as the forward Euler algorithm given by

$$\mathbf{y}(t_{j+1}) = \mathbf{y}(t_j) + (t_{j+1} - t_j)\,\mathbf{F}(\mathbf{y}(t_j),t_j) \quad , \tag{6.6}$$

and with a time step in the order of the time scale of the slowly varying solution, the numerical solution shows oscillations as indicated in the left-hand diagram of Fig. 6.1. These oscillations are caused by the linear extrapolation of the slope of the transient solution which, for stiff systems of differential equations, is a poor approximation to the slope of the slowly varying solution. This problem leads to the need for

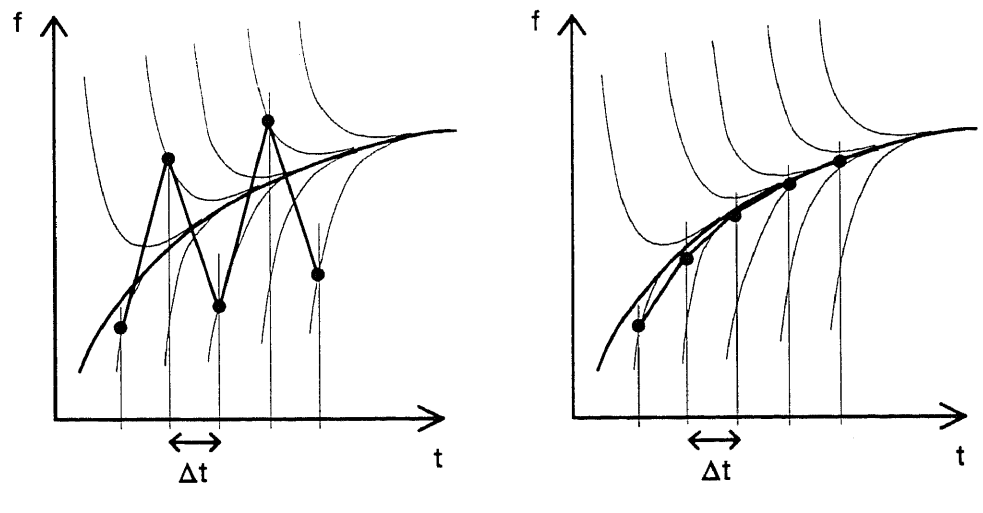


Fig. 6.1: Numerical solution of a stiff system of differential equations by the explicit (left) and the implicit (right) Euler algorithm with the same size of the time step.

a very short time step in the order of magnitude of the time scale of the transient solution in order to obtain a stable integration. This makes explicit integration techniques very inefficient for solving stiff systems of differential equations. In contrast to explicit integration techniques, implicit techniques are based on a backward extrapolation of the slope of the solution at the end point of the time interval. An example of such a technique is the backward Euler algorithm given by

$$\mathbf{y}(t_{j+1}) = \mathbf{y}(t_j) + (t_{j+1} - t_j)\, \mathbf{F}(\mathbf{y}(t_{j+1}),t_{j+1}) \quad . \tag{6.7}$$

As shown in the right-hand diagram of Fig. 6.1, although this algorithm does not exactly reproduce the fast varying transient solution, it leads to a good approximation of the slowly varying solution to be found (the numerical solution is now tangent to the true solution at the end of the time step instead of at the beginning). Note that the system of equations given by (6.7) is much more difficult to be solved than equation (6.6) because the unknown values $\mathbf{y}(t_{j+1})$ occur on both sides of the equation. Due to possible nonlinearities of the function **F**, the solution of the equation (6.7) must be performed iteratively. This additional effort must be compensated by the larger time step possible for the algorithm (6.7) as compared to the algorithm (6.6). If the system of equations is stiff enough, this larger time step by far over-compensates the additional computational expense of the implicit technique required for executing a single time step. Therefore, ***implicit integration techniques are much more efficient for the integration of stiff systems of differential equations than are explicit techniques***. The size of the time step of an implicit technique is limited by the desired accuracy of the integration, by restrictions due to the stability range of the algorithm that are much less severe as compared to explicit techniques, and by possible convergence problems of the algorithm solving the nonlinear implicit system of algebraic equations at each time step. For a detection of causes of numerical integration problems, it is important to know, if integration failed due to convergence problems or due to error test failures.

The most popular numerical technique for the integration of stiff systems of ordinary differential equations is originally due to Gear (Gear, 1969, 1971a-c; cf. Byrne and Hindmarsh, 1987). This method is a generalization of the backward Euler algorithm described by equation (6.7). It is given by

$$\mathbf{y}(t_{j+1}) = \sum_{k=0}^{q-1} \alpha_k\, \mathbf{y}(t_{j-k}) + \Delta t\, \beta_o\, \mathbf{F}(\mathbf{y}(t_{j+1}),t_{j+1}) \quad , \tag{6.8}$$

where $\Delta t = t_{i+1}-t_i$ is the (constant) time step, $1 \le q \le 6$ is the order of integration and the coefficients α_k and β_o are determined by the condition that a polynomial of order q passes through the q points $\mathbf{y}(t_{j-q+1})$ to $\mathbf{y}(t_j)$ and has a derivative of $\mathbf{F}(\mathbf{y}(t_{j+1}),t_{j+1})$ at t_{j+1}. These coefficients are listed in Table 6.1. Note that for q=1 the algorithm (6.8) degenerates to the backward Euler algorithm given by equation (6.7). Because it uses the derivative at the new point $\mathbf{y}(t_{j+1})$ it is called a ***backward differentiation formula*** (this is the main reason for its stability in solving stiff systems of differential equations as shown above) and because of the use of more than one past value of the solution it is called a ***multistep method***. Gear (1969 and 1971c) proved that this

Table 6.1: Coefficients α_k and β_0 of the Gear backward differentiation formula (6.8) for q = 1 to 6 according to Gear (1971c).

q	β_0	α_0	α_1	α_2	α_3	α_4	α_5
1	1	1					
2	$\frac{2}{3}$	$\frac{4}{3}$	$-\frac{1}{3}$				
3	$\frac{6}{11}$	$\frac{18}{11}$	$-\frac{9}{11}$	$\frac{2}{11}$			
4	$\frac{12}{25}$	$\frac{48}{25}$	$-\frac{36}{25}$	$\frac{16}{25}$	$-\frac{3}{25}$		
5	$\frac{60}{137}$	$\frac{300}{137}$	$-\frac{300}{137}$	$\frac{200}{137}$	$-\frac{75}{137}$	$\frac{12}{137}$	
6	$\frac{60}{147}$	$\frac{360}{147}$	$-\frac{450}{147}$	$\frac{400}{147}$	$-\frac{225}{147}$	$\frac{72}{147}$	$-\frac{10}{147}$

algorithm is unconditionally stable for linear stiff systems and orders 1 and 2 and that the range of stability slightly decreases for orders 3 to 6. The equation (6.8) for a single time step is solved by Newton iteration. The iteration step is given by

$$\mathbf{y}^{(n+1)} = \mathbf{y}^{(n)} + \left(I - \Delta t\, \beta_o \frac{\partial \mathbf{F}}{\partial \mathbf{y}} \right)^{-1} \left(\sum_{k=0}^{q-1} \alpha_k\, \mathbf{y}(t_{j-k}) + \Delta t\, \beta_o\, \mathbf{F}(\mathbf{y}^{(n)}, t_{j+1}) - \mathbf{y}^{(n)} \right) \qquad (6.9)$$

where $\mathbf{y}^{(n)}$ and $\mathbf{y}^{(n+1)}$ are the approximations n and (n+1) to $\mathbf{y}(t_{j+1})$, I is the unity matrix, and $\partial\mathbf{F}/\partial\mathbf{y}$ is the jacobian matrix of the right-hand side of the differential equation (6.5). Note that the equation (6.9) has the remarkable property, that if the iteration converges, it converges to the correct solution even if the jacobian matrix $\partial\mathbf{F}/\partial\mathbf{y}$ is incorrect or inaccurate. This property is used by the implementation (see below).

The first implementation of the algorithm described above was done by Gear (1971a,b) as a ***variable-step, variable-order*** method. The implementation was later improved and extended by Hindmarsh (1983). A further generalization of the method to differential-algebraic systems was implemented by Petzold (1983). Her program DASSL (Petzold, 1983) is based on adapted error control criteria taking into account problems of conventional error control criteria occurring for differential-algebraic systems (Petzold, 1982). A comprehensive report on differential-algebraic systems including a description of the program DASSL is given by Brenan et al. (1989). The program DASSL was selected for the numerical solution of the differential-algebraic system (6.1) obtained after spatially discretizing the partial differential equations.

All of the algorithms described above start the integration with the first-order method (q=1; because no function values in the past are available) and with a very small time step. Later on the time step and the order are adapted according to the behavior of the solution using an effective, semi-empirical strategy. Due to the property of the iteration formula (6.9) described above, that if the iteration converges it converges to the correct solution even if the jacobian matrix is inaccurate, the expensive process

of updating the jacobian matrix is only done if convergence fails. The time step of the algorithm is varied according to internal criteria, output at user defined points of time is interpolated using the interpolation polynomial also used for the calculation of the solution at the end of the time step. The most important parameters of the algorithm are the maximum size of the internal time step, maximum order of the algorithm (1 to 5 for DASSL) and the number of codiagonals used for the calculation of the jacobian matrix. Maximum time step may be reduced to avoid skipping over too large time domains, reduction of maximum order increases stability and decreases the time step and the number of codiagonals can only be reduced for linearly arranged systems without recirculation or branching.

The experiences with the use of the time integration algorithm described in this section were very good. The two most severe disadvantages are

- There is a fast increase of memory requirements and decrease of efficiency for large systems in which the jacobian matrix is not banded. This limits somewhat the application of the technique to two- and three-dimensional systems unless sparse matrix techniques become available.
- Due to the approximation of the solution by relatively high-order polynomials, the algorithm is unable to pass discontinuities of external variables and it must decrease the time step considerably at discontinuities of the slope of external variables (as they occur by linear interpolation of measured time series) if the algorithm is not restarted at any value of the argument.

6.3 Spatial Discretization

In this section methods used for the spatial discretization of one-dimensional conservation laws are discussed. Section 6.3.1 reviews conservative spatial discretization techniques for conservation laws, which are based on the formulation of conservation laws given in section 4.3.1. The problem of avoiding numerical diffusion and oscillating numerical solutions is treated in section 6.3.2. In section 6.3.3 it is explained how the methods introduced before can be extended to adaptive grids.

6.3.1 Conservative Methods for Conservation Laws

The idea behind conservative discretization methods is to construct a numerical scheme, which exactly fulfills a discretized approximation to the original conservation law (e.g. LeVeque, 1990). The simplest derivation of a conservative method is to discretize the integral form (4.8) of the conservation law instead of the differential form (4.9). A straightforward discretization of the (semi-)integral form of the conservation laws (4.8) is the conservative finite difference discretization given by dividing the real axis into cells as shown in Fig. 6.2 and solving the set of ordinary differential equations

$$\begin{aligned} \frac{d}{dt}\left((x_{i+1/2} - x_{i-1/2})\, \hat{\rho}(x_i,t) \right) = & \left(\hat{j}_{num}(x_{i-1/2}) - \frac{dx_{i-1/2}}{dt}\, \hat{\rho}(x_{i-1/2},t) \right) \\ & - \left(\hat{j}_{num}(x_{i+1/2}) - \frac{dx_{i+1/2}}{dt}\, \hat{\rho}(x_{i+1/2},t) \right) \\ & + (x_{i+1/2} - x_{i-1/2})\, \hat{r}(x_i,t) \quad , \end{aligned} \tag{6.10}$$

where $\hat{j}_{num}(x_{i\pm1/2})$ is a numerical approximation to the flux $\hat{j}$ at the cell boundaries, $\hat{\rho}(x_i,t)$ represents the average one-dimensional density within a cell, $\hat{\rho}(x_{i\pm1/2},t)$ is the one-dimensional density at the cell boundaries (interpolated values of $\hat{\rho}(x_i,t)$), and time derivatives of the location of cell boundaries $dx_{i\pm1/2}/dt$ have to be given according to a strategy discussed in section 6.3.3. To complete the system of equations (6.10), the flux from the left-hand side into the first cell and the flux leaving the last cell to the right must be specified. Usual conservative methods do not solve the discretization equation (6.10) of the general conservation law (4.8), but instead assume a static division of the x-axis into cells. This corresponds to setting the derivatives dx_i/dt in equation (4.8) or the derivatives $dx_{i\pm1/2}/dt$ in equation (6.10), to zero. For such a static numerical grid, the discretization equation (6.10) simplifies to

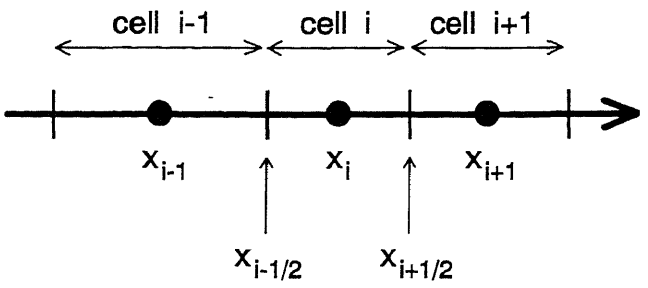


Fig. 6.2: Division of the real axis into cells for discretizing the integral form (4.8) of the conservation law to obtain the numerical approximations (6.10) or (6.11).

$$\frac{d}{dt}\hat{\rho}(x_i,t) = \frac{\hat{j}_{num}(x_{i-1/2}) - \hat{j}_{num}(x_{i+1/2})}{x_{i+1/2} - x_{i-1/2}} + \hat{r}(x_i,t) \quad . \tag{6.11}$$

Since both of the discretizations given by equations (6.10) and (6.11) use exactly the same flux for the decrease of mass in the cell on the left of a cell boundary as they do for the increase of mass in the neighboring cell to the right, these discretizations exactly conserve the quantities described by the one-dimensional densities $\hat{\rho}$. The accuracy and the behavior of the numerical algorithm is determined by the method of approximating the true flux $\hat{j}$ at the cell boundaries by the numerical flux $\hat{j}_{num}$ using the densities at neighboring grid points x_i. The order of discretization of such a method is given by the smallest exponent of the cell length that is neglected in the Taylor expansion of the flux. A first-order method approximates the flux at $x_{i-1/2}$ by its value at x_{i-1} or at x_i (depending on the direction of the flux: upstream differentiation), whereas a second-order method uses a weighted average of these two values. ***The disadvantage of a first-order method is the appearance of numerical diffusion, which artificially smears out sharp fronts of flux. A second-order flux solves the problem of numerical diffusion, but it leads to oscillations in the numerical solution near sharp density structures*** (e.g. LeVeque, 1990). In the next section, it is shown how the positive features of first- and second-order methods can be combined to yield an accurate and stable space discretization technique.

6.3.2 Flux Limiter Methods

As described in the previous subsection, a first-order flux behaves very well even near discontinuities of the solution, but it smoothes the solution with numerical diffusion. In contrast to this behavior, a second-order flux approximates a smooth solution much better but it leads to oscillations near discontinuities or in zones where the solution varies strongly. ***The idea of high resolution flux limiter methods is to switch continuously between a first-order flux near discontinuities or zones of strong variations of the solution and a second-order flux in zones where the solution is smooth*** (e.g. LeVeque, 1990). Such a procedure makes the discretization scheme dependent on the solution. A high resolution flux can be written in the form

$$\hat{j}_{num,high} = (1-\Phi)\,\hat{j}_{num,1} + \Phi\,\hat{j}_{num,2} \quad , \tag{6.12a}$$

where $\hat{j}_{num,high}$ is the high resolution numerical flux (used in one of the discretizations given by equation (6.10) or (6.11)), $\hat{j}_{num,1}$ is a first-order numerical flux, $\hat{j}_{num,2}$ is a second-order numerical flux and Φ is a weighting function of the two methods. Setting Φ to zero obviously lets the numerical flux (6.12a) degenerate to a first-order flux, whereas setting Φ to unity yields a second-order flux. Values in between take into account contributions of both first- and second-order fluxes. Equation (6.12a) can alternatively be written in the form

$$\hat{j}_{num,high} = \hat{j}_{num,1} + \Phi\,(\hat{j}_{num,2} - \hat{j}_{num,1}) \quad . \tag{6.12b}$$

In this form the high resolution flux can be interpreted as the sum of a first-order flux plus an anti-diffusive flux, the contribution of which is limited by the factor Φ ("flux limiter"), which becomes small in strongly varying zones of the solution. An alternative interpretation of equation (6.12a) is given by writing the high resolution flux in the form

$$\hat{j}_{num,high} = \hat{j}_{num,2} + (1-\Phi)\,(\hat{j}_{num,1} - \hat{j}_{num,2}) \quad . \tag{6.12c}$$

In this form, the high resolution flux can be interpreted as a second-order flux, to which a sufficient contribution (controlled by the factor $(1-\Phi)$) of numerical diffusion is added to avoid numerical oscillations. Many formulations of the dependence of the flux limiter Φ on the solution have been proposed (see surveys given by Sweby, 1984 and LeVeque, 1990). The most often used measure of "smoothness" of the solution is the ratio of consecutive gradients

$$\Theta_i = \frac{\hat{\rho}_i - \hat{\rho}_{i-1}}{x_i - x_{i-1}} \left(\frac{\hat{\rho}_{i+1} - \hat{\rho}_i}{x_{i+1} - x_i} \right)^{-1} \quad . \tag{6.13}$$

(Θ_i and Φ are calculated separately for each component of $\hat{\rho}$). Sweby (1984) showed that functions $\Phi(\Theta)$ that lie in the shaded area shown in Fig. 6.3 lead to non-oscil-

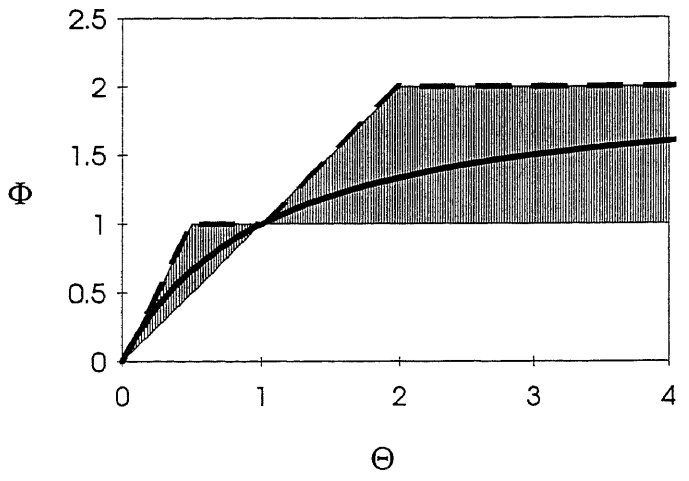


Fig. 6.3: Dependence of the flux limiters on the ratio of consecutive gradients of the solutions. Shaded area: region of non-oscillating second-order limiters (Sweby, 1984). Solid line: van Leer limiter according to equation (6.14a). Dashed line: superbee limiter of Roe according to equation (6.14b).

lating second-order discretization techniques (note that values of Φ larger than one are allowed). Two of the most common limiters are the limiter of van Leer (1974)

$$\Phi(\Theta) = \frac{|\Theta| + \Theta}{1 + |\Theta|} \qquad (6.14a)$$

and the "superbee" limiter of Roe (1985)

$$\Phi(\Theta) = \max\big(0 , \min(1,2\Theta) , \min(\Theta,2) \big) \quad , \qquad (6.14b)$$

which follows the upper boundary of the shaded area in Fig. 6.3. These two flux limiters are also shown in Fig. 6.3. Fig. 6.4 shows a comparison of longitudinal profiles of the transport of a conservative substance spilled into a dispersion-free river section as a rectangular pulse, calculated with a first-order method (Φ=0), using the van Leer limiter according to equation (6.14a) and the superbee limiter according to equation (6.14b). The superiority of the high resolution flux limiter methods over the first-order method, which shows significant numerical diffusion, is evident. In agreement with the results of Sweby (1984), the superbee limiter, which uses the maximum possible contribution of the second-order flux, gives the best results. Nevertheless, the van Leer limiter was chosen as default flux limiter for the program, because the non-differentiable nature of the superbee limiter decreases the step size of the time integrator and leads to a significant increase in computation time (cf. comment at the end of section 6.2).

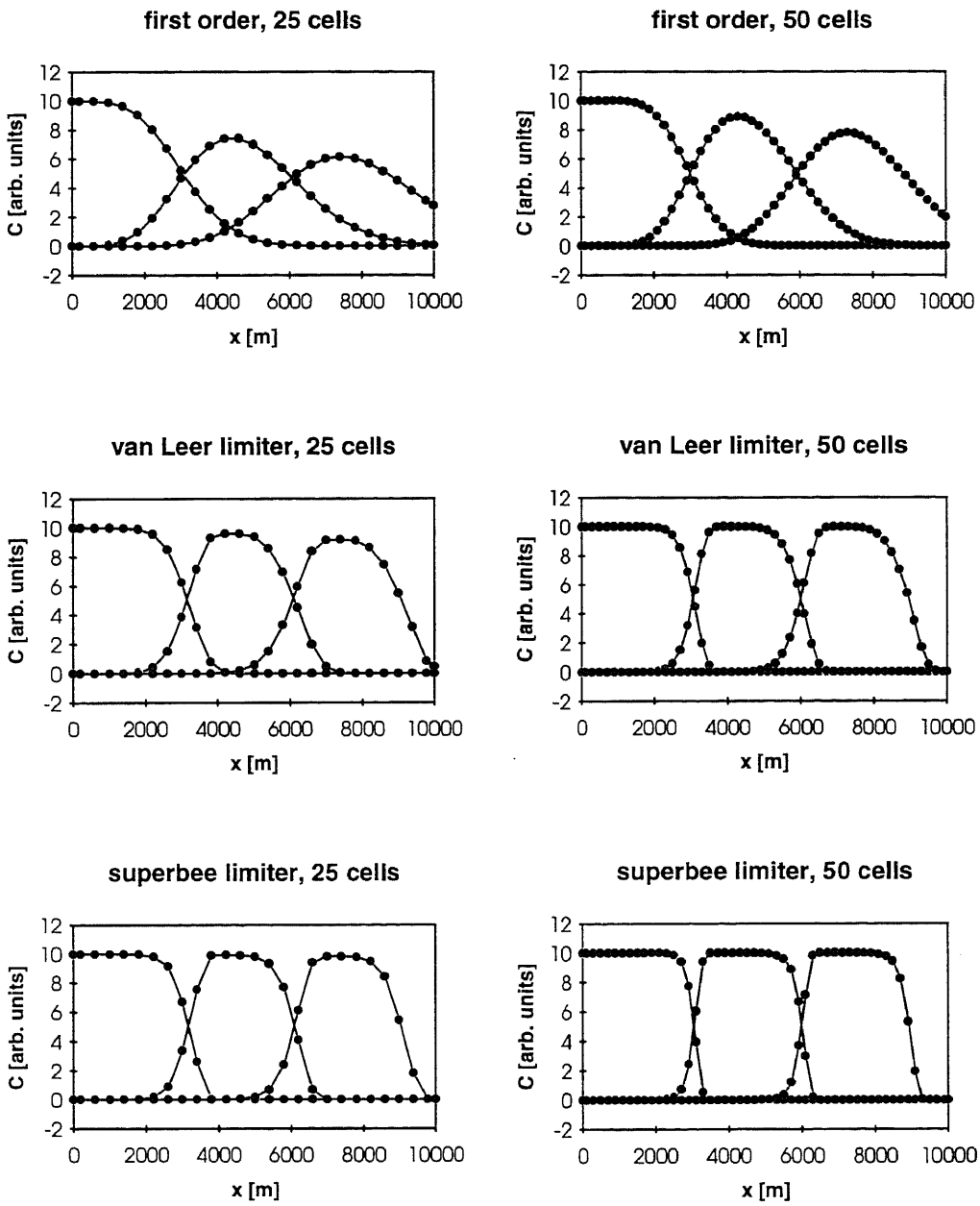


Fig. 6.4: Comparison of longitudinal profiles of a conservative rectangular pulse transported along a river. The first-order method (Φ=0) is compared with flux limiter methods using the limiters of van Leer according to equation (6.14a) and of Roe according to equation (6.14b) for two different spatial resolutions.

6.3.3 Adaptive Grids

Adaptive grids, as discussed in this report, use moving grid points in order to describe regions in space of varying size or to dynamically resolve areas of high curvature of the solutions. Grid refinements by addition of grid points are not taken into account in this report. The generalized integral form of conservation laws (4.8), which includes moving boundary points $x_i(t)$, is the basis for the implementation of adaptive grid techniques. If the fluxes across the boundaries (4.6)

$$I(t) = \hat{j}(x_i(t),t) - \frac{dx_i}{dt}(t)\,\hat{\rho}(x_i(t),t) \qquad (6.15)$$

are approximated by a discretization scheme instead of the fluxes in the resting coordinate system $\hat{j}$, a moving grid technique is implemented. The discrete form of the equations is then given by equation (6.10) instead of equation (6.11). The goal then consists of finding reasonable differential equations for the temporal evolution of the boundary points $x_i(t)$ to follow and resolve areas of high curvature or of discontinuities of the solutions. In the current version of the program only moving grid points describing variation of biofilm size in the biofilm reactor compartment are taken into account. In this case the position of a grid point x_i is given as the solution to the equation

$$\frac{dx_i}{dt} = \frac{x_i}{L_F} u_L \qquad (6.16)$$

(L_F (L) is the biofilm thickness and u_L (LT^{-1}) the velocity of the interface between biofilm and bulk volume). This strategy of moving grid points keeps relative distances between grid points. As a special case, an initially equidistributed grid remains equidistributed although the biofilm thickness changes. It is planned to use the technique described above also for improving grid resolution in regions of high curvature of the solutions in later program versions, by changing the strategy of movement of grid points given above.

6.4 Uncertainty and Identifiability Analyses

The derivatives required for calculating the sensitivity functions (2.2a) to (2.2d), the standard deviation of a calculated variable according to equation (2.21) and the contribution of a parameter to the uncertainty of a variable according to equation (2.22) are calculated using the ***finite difference approximation***

$$\frac{\partial f}{\partial p_i} \approx \frac{f(p_i + \Delta p_i) - f(p_i)}{\Delta p_i} \quad . \tag{6.17}$$

In order to make the program as flexible as possible, the solutions for the basic parameter values ($p_1,...,p_m$) and for all m sets of parameter values, where one of the p_i's is replaced by $p_i+\Delta p_i$ are stored simultaneously. Using these m+1 stored states, all the expressions (2.2a) to (2.2d), (2.21) and (2.22) can be evaluated easily with the aid of the approximation (6.17) for arbitrary variables and all m parameters of the set selected for the calculation. The increment Δp_i is chosen to be one percent of the standard deviation σ_{p_i} of the variable p_i.

6.5 Parameter Estimation

As described in sections 5.3 and 2.3.3, parameters are estimated by minimizing the sum of the squares of the weighted deviations between measured and calculated values given by the expression (2.12)

$$\chi^2(\mathbf{p}) = \sum_{i=1}^{n} \left(\frac{f_{meas,i} - f_i(\mathbf{p})}{\sigma_{meas,i}} \right)^2 , \qquad (6.18)$$

where $\mathbf{p} = (p_1, \ldots, p_m)$ is the array of m parameters, $f_{meas,i}$ is the measured and $f_i(\mathbf{p})$ the calculated value corresponding to a data point, $\sigma_{meas,i}$ is the standard deviation of $f_{meas,i}$ and the sum extends over the data points of all data series of all calculations included into the parameter estimation. Since parameters are represented by constant variables of model structure, which are bounded by the minimum and maximum of their legal range (cf. section 4.1.2), minimization of equation (6.18) together with the conditions

$$p_{min,i} \leq p_i \leq p_{max,i} \qquad (6.19)$$

is a ***constrained minimization problem*** to be solved numerically.

There are a lot of numerical methods for localizing the minimum of an arbitrary function or of a function of the special form of equation (6.18) (Bevington, 1969; Gill et al., 1981; Press et al., 1989). Most of these algorithms use first- or second-order derivatives of the function $\chi^2(\mathbf{p})$ to be minimized. In most cases, these derivatives must be approximated numerically. This is also the case for the data analysis program developed in this report, since the values $f_i(\mathbf{p})$ used in (6.18) result from the numerical solution of a system of partial differential equations, ordinary differential equations and algebraic equations. The numerical errors due to an approximative solution of such a system of equations are much larger than machine precision and make the theoretically smooth χ^2 hypersurface in $\mathbf{R}^{m+1}$ (m is the number of parameters and χ^2 is plotted against $p_1, \ldots, p_m$ on the axis m+1) to a "crumpled" hypersurface. This fact obviously makes the choice of the step size of a numerical finite difference approximation of derivatives delicate. This is the reason why ***derivative-free algorithms were chosen*** for application by the identification and simulation program for aquatic systems described in this report. Two in some sense (as explained later) complementary algorithms are selected. This allows the user to change the algorithm in case of numerical problems of minimization. Since some changes were made to the algorithms as described in the literature, they are described in more detail than the other algorithms discussed in this chapter.

6.5.1 Simplex Algorithm

The first algorithm used for parameter estimation is the simplex function minimization method of Nelder and Mead (1965) (also described in Press et al., 1989). This algorithm starts with a set of m+1 arrays of parameters building a simplex of nonvanishing volume in the parameter space $\mathbf{R}^m$. This start simplex is generated from the initial parameter array specified by the user by incrementing (or decrementing) each parameter by 10 % of the length of its legal interval. At each iteration step, the point with the highest value of the function to be minimized (in our case χ^2 according to equation (6.18)) and in some cases also other points of the simplex are replaced by new points according to the four basic steps illustrated for two parameters in Fig. 6.5. As a first step, the point with the highest function value is reflected at the plane defined by the other points (this is a natural guess for the downhill direction). If the function value at this point is between the lowest and the highest value at the points of the simplex, it is accepted and replaces the point with the highest function value (cf. first diagram of Fig. 6.5). If it is smaller than the lowest value at the points of the simplex, an increase of the step size by an additional extrapolation in the same direction is tried. If the extrapolated point has a function value smaller than the lowest value at the points of the simplex, it is accepted (cf. second diagram of Fig. 6.5), otherwise the original reflected point (without extrapolation) replaces the point with the highest function value. If the function value at the reflected point is higher than the highest value at the points of the simplex, a contraction of the simplex away from the highest point is tried (cf. third diagram of Fig. 6.5). If also the function value at this point is higher than the highest value of the simplex, a contraction of all other points towards the point with the lowest function value is performed (cf. fourth diagram of Fig. 6.5). This procedure is continued until all function values at the points

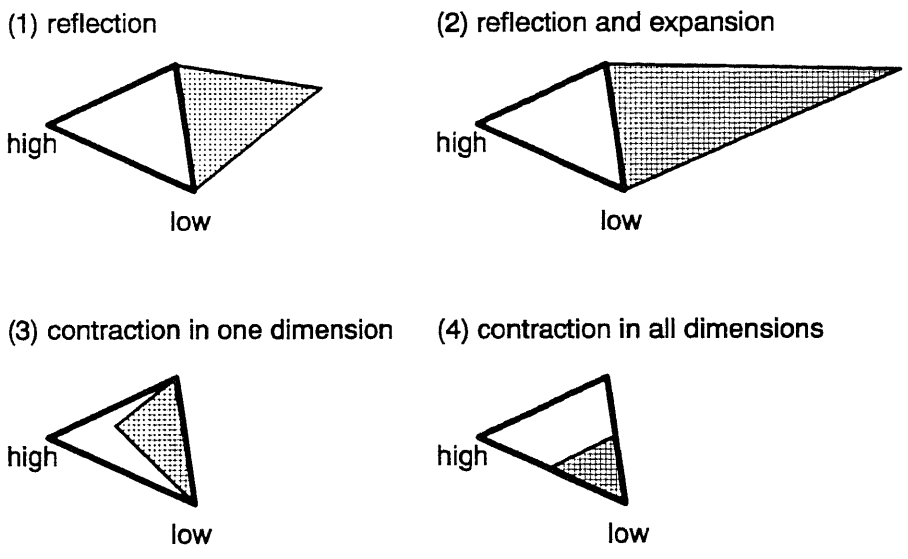


Fig. 6.5: Basic steps of the simplex algorithm for function minimization illustrated for the case of two parameters. The solid line simplex is replaced by one of the hashed simplexes.

of the simplex are close enough together to fulfill the convergence criterion.

As described so far, this is the original algorithm of Nelder and Mead (1965) for unconstrained function minimization. We now have to take into account the constraints (6.19). Nelder and Mead recommend to consider inequality constraints by returning an extremely high function value for parameter values not fulfilling the constraints (penalty or barrier function method). This procedure obviously leads to an unconstrained minimization problem with the same solution as in the constrained case. Practical experiences showed, however, that this algorithm had problems to proceed a descending direction along a constraint. Therefore, a parameter transformation illustrated in Fig. 6.6 was chosen for the elimination of the constraints. The parameter p_i within the interval between $p_{min,i}$ and $p_{max,i}$ is mapped to the whole real axis by the transformation

$$p_i' = \tan\left(\frac{\pi}{2}\,\frac{2p_i - p_{max,i} - p_{min,i}}{p_{max,i} - p_{min,i}}\right) \quad . \tag{6.20}$$

The minimization with the unconstrained algorithm as described before is performed in the coordinates p_i' and the solution in the original coordinates is obtained by the inverse transformation

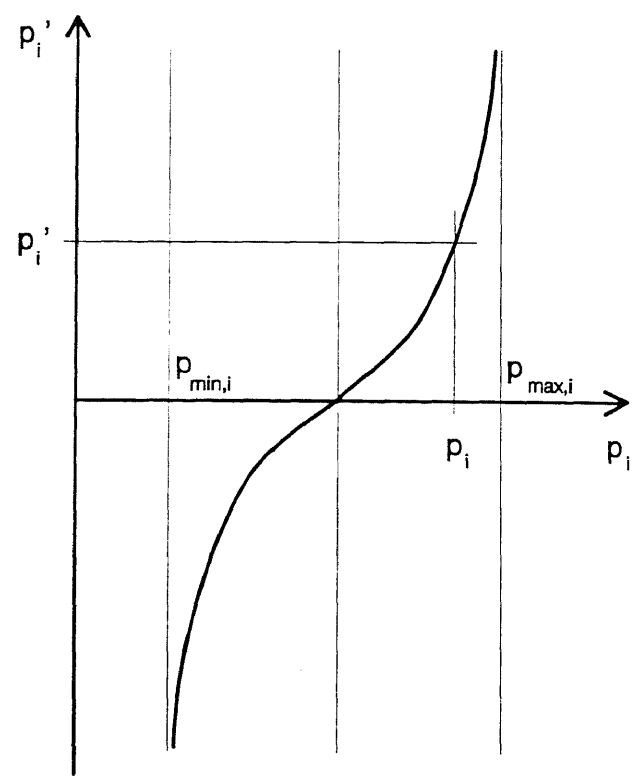


Fig. 6.6: Scaled tan - function used to transform the bounded interval ($p_{min,i}$,$p_{max,i}$) of the parameter p_i to the whole real axis for the parameter p_i'.

$$p_i = \frac{1}{2}\left(p_{max,i} - p_{min,i}\right) + \left(p_{max,i} - p_{min,i}\right)\frac{\arctan(p_i')}{\pi} \quad , \qquad (6.21)$$

which maps the real axis to the interval between $p_{min,i}$ and $p_{max,i}$. To avoid divergence of transformed parameters, an initial point lying exactly on a boundary is moved away from the boundary by a small fraction of the interval length before start of the iteration. Whereas such a parameter transformation can be dangerous for Newton-like algorithms with unlimited step size (divergence of calculated solution), due to the limited growth rate of the step size, it works very well for the algorithm described above. The simplex algorithm of Nelder and Mead (1965) together with the implementation of the inequality constraints (6.19) by the transformations (6.20) and (6.21) leads to a very robust method for estimating parameters of differential-algebraic systems of equations. There are, however, two important disadvantages of this algorithm:

1) ***There is no straightforward method for estimating standard deviations and correlations of parameters***. The method proposed by Nelder and Mead (1965) requires a lot of additional function evaluations.

2) ***The method converges very slowly***, so that many function evaluations are necessary to locate the minimum. This is a very serious problem, since in the present case, function evaluations are by far the most expensive part of the algorithm (each function evaluation needs performing a model simulation).

Especially this second point makes the additional implementation of a more efficient algorithm desirable.

6.5.2 Secant Algorithm

The idea for a more efficient algorithm for minimizing the function (6.18) is to make use of the special form of this function and to assume differentiability of the function f in (6.18). Differentiability of **f** implies that the accuracy of the approximation of **f** by a secant increases for decreasing distance of the points defining the secant. The selected algorithm is an extension of the secant method "Dud" of Ralston and Jennrich (1978). The extensions are to take into account individually weighted data points as formulated in equation (6.18) and simple inequality constraints of the parameters as given by the equations (6.19). In this section, the extended algorithm is described.

Similarly to the algorithm of Nelder and Mead described in the previous paragraph, the algorithm starts with a set of m+1 arrays of the parameters $\mathbf{p}^{(1)},\ldots,\mathbf{p}^{(m+1)}$ in $\mathbf{R}^m$.

Again, this set is generated from the initial parameter array specified by the user by incrementing (or decrementing) each parameter by 10 % of the length of its legal interval. There is exactly one linear function $\mathbf{F}$: $\mathbf{R}^m \rightarrow \mathbf{R}^n$ which passes through the function values $f_i(\mathbf{p}^{(j)})$ at the points $\mathbf{p}^{(j)}$ for all i=1,...,n and all j=1,...,m+1. This function describes a secant to the nonlinear function f used in expression (6.18). If $\mathbf{f}$ is replaced by this secant approximation $\mathbf{F}$, χ^2 according to (6.18) becomes the positive definite quadratic form

$$Q(\mathbf{p}) = \sum_{i=1}^{n} \left(\frac{f_{meas,i} - F_i(\mathbf{p})}{\sigma_{meas,i}} \right)^2 \quad . \tag{6.22}$$

This function has exactly one extremal point at its minimum. The idea of the algorithm of Ralston and Jennrich (1978) is to exactly locate this minimum at each iteration step. The new parameter array resulting from the solution of this problem replaces one of the m+1 parameter arrays of the previous step. The new set of parameter arrays defines a new secant function F and the algorithm can proceed until convergence is achieved.

To avoid unnecessary matrix inversions, the secant function $\mathbf{F}$ is not formulated as a function of the parameters $\mathbf{p} = (p_1,...,p_m)$ but with auxiliary coordinates $\alpha = (\alpha_1,...,\alpha_m)$ in the following way:

$$F_i(\alpha) = f_i(\mathbf{p}^{(m+1)}) + \sum_{j=1}^{m} \left(f_i(\mathbf{p}^{(j)}) - f_i(\mathbf{p}^{(m+1)}) \right) \alpha_j \quad , \tag{6.23}$$

where α is connected to $\mathbf{p}$ by

$$p_i(\alpha) = p_i^{(m+1)} + \sum_{j=1}^{m} \left(p_i^{(j)} - p_i^{(m+1)} \right) \alpha_j \quad . \tag{6.24}$$

For j=1,...,m, the linear function $\mathbf{F}$ obviously passes through the function value $\mathbf{f}(\mathbf{p}^{(j)})$ at the point α with α_j=1 and α_i=0 for i≠j. It passes through $\mathbf{f}(\mathbf{p}^{(m+1)})$ at α=0. With the aid of the matrices

$$(\Delta F)_{i,j} = f_i(\mathbf{p}^{(j)}) - f_i(\mathbf{p}^{(m+1)}) \qquad i=1,...,n \;,\; j=1,...,m \tag{6.25}$$

and

$$(\Delta p)_{i,j} = p_i^{(j)} - p_i^{(m+1)} \qquad i=1,...m \;,\; j=1,...,m \tag{6.26}$$

the equations (6.23) and (6.24) can simpler be written in matrix notation as

$$\mathbf{F}(\alpha) = \mathbf{f}(\mathbf{p}^{(m+1)}) + \Delta F \cdot \alpha \tag{6.27}$$

and

$$\mathbf{p}(\alpha) = \mathbf{p}^{(m+1)} + \Delta p \cdot \alpha \quad . \tag{6.28}$$

The quadradic form Q defined in (6.22) expressed as a function of the coordinates α is then given as

$$Q(\alpha) = (\mathbf{f}_{meas} - \mathbf{F}(\alpha))^T \odot (\mathbf{f}_{meas} - \mathbf{F}(\alpha)) \quad , \tag{6.29}$$

where T means transposed and "$\odot$" is the scalar product in $\mathbf{R}^n$ with division of component i by $\sigma_{meas,i}^2$. The gradient of Q with respect to α is given by

$$\mathbf{grad}_\alpha Q = 2 \left(\Delta F^T \odot \Delta F \cdot \alpha - \Delta F^T \odot (\mathbf{f}_{meas} - \mathbf{f}(\mathbf{p}^{(m+1)})) \right) \quad . \tag{6.30}$$

The unique minimum of Q is given by the solution of the linear equation

$$\mathbf{grad}_\alpha Q = 0 \quad , \tag{6.31}$$

which is solved for α using LINPACK routines of Dongarra et al. (1979). The parameter values at the solution are obtained by applying equation (6.24) or (6.28) to the solution α.

So far, only the extension to individually weighted data points was included into the original algorithm of Ralston and Jennrich (1978). It is more difficult to include the inequality constraints (6.19). In the case of the secant algorithm it is not possible to eliminate the constraints with the parameter transformation (6.20) and (6.21), which was successfully applied to the simplex method, because the unlimited step size of the secant method would lead to rapid divergence of parameters lying on boundaries. Instead, the active set method described conceptually in Gill et al. (1981) is applied. In this method, the constraints are divided into active constraints, where one of the bounding equalities of (6.19) hold, and inactive constraints, which must not be taken into account (temporarily). For a given active set, the inequality constrained optimization problem is reduced to an equality constrained problem which is easier to solve. If the equality constraints are formulated as

$$G_k(\alpha) = 0 \qquad k=1,\ldots,l \tag{6.32}$$

and the rank of the matrix $(\partial G_i/\partial \alpha_k)$ is l (what is clearly true for equality constraints due to (6.19)), the minimum of the constrained problem consisting of minimizing the expression (6.29) under the constraints (6.32) is given by the simultaneous solution of (6.32) and

$$\mathbf{grad}_\alpha Q + \sum_{k=1}^{l} \lambda_k \, \mathbf{grad}_\alpha G_k = 0 \quad , \tag{6.33}$$

where the factors λ_k must be determined simultaneously with α out of the $m+l$ equations (6.32) and (6.33) are called Lagrange multipliers (see any text book on analysis, e.g. Blatter, 1974). There is a very simple geometrical interpretation of equation (6.33): $\mathbf{grad}_\alpha G_k$ is a direction perpendicular to the constraint $G_k=0$. The

minimum of the constrained problem needs only to be an extremal point for directions parallel to the constraints. To allow the component of $\mathbf{grad}_\alpha Q$ perpendicular to a constraint to take an arbitrary value, an arbitrary multiple of the gradient of the constraint is added. The Lagrange multiplier λ_k is the multiple of the gradient of the constraint G_k. This geometrical interpretation of Lagrange multipliers is used below for checking the correctness of the active set of constraints. For the inequality constraints (6.19), possible equality constraints of the active set have the form

$$G_k(\alpha) = p_{i_k}^{(m+1)} + \sum_{j=1}^{m} \left(p_{i_k}^{(j)} - p_{i_k}^{(m+1)}\right) \alpha_j - p_{given,i} = 0 \quad , \tag{6.34}$$

where $p_{given,i}$ is either $p_{min,i}$ or $p_{max,i}$ depending on which of the interval boundary is active and $i_1,...,i_\ell$ are the indices of the active set. With the aid of the submatrix

$$(\Delta\hat{p})_{j,k} = p_{i_k}^{(j)} - p_{i_k}^{(m+1)} \tag{6.35}$$

and the subarray

$$\hat{p}_k = p_{i_k} \tag{6.36}$$

corresponding to the active set of constraints and using equation (6.30), the equations (6.33) and (6.34) can simpler be written in matrix notation as

$$\Delta F^T \odot \Delta F \cdot \alpha + \Delta\hat{p} \cdot \lambda = \Delta F^T \odot \left(\mathbf{f}_{meas} - \mathbf{f}(\mathbf{p}^{(m+1)})\right) \tag{6.37}$$

and

$$\Delta\hat{p}^T \cdot \alpha = \hat{\mathbf{p}}_{given} - \hat{\mathbf{p}}^{(m+1)} \quad . \tag{6.38}$$

These two equations can be combined to a single matrix equation of dimension $m+\ell$ as follows:

$$\begin{pmatrix} \Delta F^T \odot \Delta F & \Delta\hat{p} \\ \Delta\hat{p}^T & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \lambda \end{pmatrix} = \begin{pmatrix} \Delta F^T \odot \left(\mathbf{f}_{meas} - \mathbf{f}(\mathbf{p}^{(m+1)})\right) \\ \hat{\mathbf{p}}_{given} - \hat{\mathbf{p}}^{(m+1)} \end{pmatrix} \quad . \tag{6.39}$$

This linear equation is solved for $(\alpha,\lambda)^T$ using LINPACK routines of Dongarra et al. (1979) to give the solution of the equality constrained minimization problem with the currently active set of constraints. The solution α is transformed to the solution of the parameters $\mathbf{p}$ using equation (6.28). If one of the components of this solution is outside of its legal interval, the violated constraint is added to the active set and solution of the linearized equality constrained problem is repeated. This procedure is repeated until a legal solution is found. This solution may include unnecessary equality constraints. Therefore, the sign of the gradient of $Q(\alpha)$ perpendicular to each constraint must be checked. According to the geometrical interpretation of Lagrange multipliers discussed after equation (6.33), only the signs of the Lagrange multipliers λ calculated together with α by solving equation (6.39) must be checked. Since a po-

sitive gradient of G_k corresponds to increasing values of the corresponding component of **p**, a necessary lower bound corresponds to a negative value of λ_k, a necessary upper bound to a positive value of λ_k. An unnecessary bound is removed and the solution of the linearized equality constrained problem is repeated. This procedure finally (after a finite number of inexpensive steps which do not require function evaluations) leads to the exact solution of the linearized inequality constrained problem.

The solution to the linearized problem usually replaces the point with the highest value of χ^2 of the set of m+1 points defining the secant function **F**. If, however the solution is too close to the best solution found so far (all components of α smaller than 10^{-3}), the point with the highest value of χ^2 is contracted by a factor of ten towards the best point to improve the approximation of the function **f** by its secant function **F** at the next iteration step. If the solution of the linearized problem has a larger value of χ^2 than any value of the actual set of solutions, convergence is improved by a step reduction mechanism. This should not be necessary near the minimum where the algorithm converges very fast.

As described by Ralston and Jennrich (1978), the unconstrained method easily makes the estimation of the variance-covariance matrix of the parameters possible without requiring additional function evaluations. The estimate is given as a numerical approximation to equation (2.18) by the expression

$$\mathrm{Cov}(\mathbf{p}) = \frac{Q(\mathbf{p})}{n-m} \Delta p \cdot \left(\Delta F^T \circledcirc \Delta F \right)^{-1} \cdot \Delta p^T \quad . \tag{6.40}$$

This formula is only applied if the active set is empty.

6.5.3 Practical Recommendations

The two algorithms described in this section are complementary in their search strategy since the simplex method slowly moves down the "gradient" of χ^2 (in a generalized sense for non-differentiable functions), whereas the secant method rapidly jumps to the position of a suggested solution of the problem found by parabolic extrapolation. This search strategy makes the secant method much more efficient concerning function evaluations especially near the minimum. Therefore, it is recommended to start a parameter estimation with the secant algorithm and only in case of numerical problems switch to the simplex method. This method is more robust far away from the solution and with reference to

non-differentiability of χ^2. After roughly localizing the solution with the simplex method, it may be advantageous to switch back to the secant method to accelerate final convergence. If problems as mentioned above will frequently occur, an additional algorithm, which switches between gradient search far from the minimum and parabolic extrapolation near the minimum similar to the method of Marquardt (1963) will be provided later.

As a general recommendation for function minimizing, ***it is always advantageous to restart the algorithm with different initial conditions to confirm the minimum found at the first trial.***

7 Object-Oriented Implementation Concepts

An implementation of an identification and simulation program as described in the preceeding chapters should fulfill the technical requirements T1 - T4 stated in section 3.2. In this chapter, it is discussed how this can be done. An even shorter survey of object-oriented design concepts of this program is given in Reichert (1994b).

Requirement T4 states, that ***the program has to be designed in a well structured and extensible way***. This is necessary in order to be able to account for future desires of the users. The model structure described in chapter 4, which can easily be extended, e.g. by additional compartment types, is the basis for such an implementation. It is the main goal of this chapter to demonstrate that object-oriented program design techniques very much simplify the implementation of such a model structure, and that this implementation at the same time is a good basis for future program extensions.

Since memory requirements and computational speed needed for model calculations can vary over several orders of magnitude for different problems, and since hardware platforms rapidly change, requirement T2 states that ***the program should be transferable to any important computing platform***. There are two problems to be taken into account in writing a portable program. The first problem is that different compiler implementations are accompanied by different libraries extending the language; the second problem is that different dialects of programming languages are implemented on different hardware platforms. The first problem can only be solved by avoiding calls to such libraries, the second by chosing a well standardized programming language. Since object-oriented programming is a relatively new approach, there are no ANSI standards for object-oriented languages (at the start time of this project in 1991; there are ANSI proposals for C++ now). At the time the programming language was selected for this project, C++ was by far the most standardized object-oriented programming language. Furthermore, C++ is efficiently implemented and it is easy to link a C++ program with a graphical user interface library written in C (as is usually the case). For these reasons, C++ was selected for realizing this project. C++ is a so called hybrid language, that is an object-oriented extension of a conventional programming language (in this case C). The language definition is given in Ellis and Stroustrup (1990); a very recommendable introduction is Lippman (1989). As a consequence of this choice, all examples used in this chapter to demonstrate the key elements of the data structures are given in C++. Knowledge of C++ is advantageous but not necessary to understand this chapter, but knowledge of elementary programming concepts is required. It is important to note that the reasons for the choice of C++ are of practical nature (portability, efficiency, large base of available C-libraries) and that all programming concepts explained in this chapter could be implemented in a similar way with any other object-oriented programming language (but not with a conventional programming language).

Requrement T1 states, that ***the program should be easy to operate for environmental scientists***. Two strategies are combined to fulfill this requirement. The first

strategy is to use a "language" for model formulation that is familiar to environmental scientists. As described in chapter 4, this is done by allowing the user to define his or her system using "compartments", "links", "processes" and "variables" instead of requesting him or her to specify differential equations. The second strategy is to implement a user-friendly program interface. Obviously, it is best to use the native graphical user-interface of a machine. The problem of this choice is that there does not (yet?) exist a standardized application programmer interface for the graphical user-interfaces of different hardware platforms and operating systems. This makes this choice to be in conflict with the requirement of portability stated above. Because of this problem, it is advisable to separate as far as possible general program tasks from program parts needed for the user-interface. Programming the user-interface then consists only of performing dialogs with the user and by calling functions of the general (and easily portable) part of the program. This strategy not only facilitates adaptation of the program to new platforms, but it makes it also possible to implement different user-interfaces. Nevertheless, because of the large number of dialog boxes needed for the program, adaptation to a new platform is troublesome. Therefore, this strategy was combined with the use of a graphical user interface library, which is available on various systems.

To facilitate later changes of numerical algorithms, requirement T3 states, that ***the interfaces between the program and routines implementing numerical algorithms should be as simple as possible*** (the selection of efficient algorithms is discussed in chapter 6). Since nearly all numerical algorithms are implemented in FORTRAN, this means that FORTRAN-like interfaces have to be used.

This chapter is structured as follows: In section 7.1 the key concepts of object-oriented programming are reviewed briefly. Then, in section 7.2 *technical* advantages of object-oriented program design are demonstrated by comparing an object-oriented implementation of a list of symbols (which is the basic data structure of the model subsystems) with a conventional implementation. In section 7.3 *conceptual* advantages of object-oriented program design are discussed by showing some techniques used for the implementation of model subsystems. In section 7.4 the solution of possible consistency problems resulting from editing the model are discussed. Implementation of user-interfaces, which form the top layer of the program between the portable core and the user interface library, is discussed in section 7.5. Finally, in section 7.6, the main implementation concepts are summarized.

7.1 Fundamentals of Object-Oriented Programming

In this section, the key concepts of object-oriented programming

- information hiding,
- abstract data types,
- inheritance,
- polymorphism,

are reviewed briefly. For a more detailed treatment see e.g. Marti (1988), Meyer (1990) or Budd (1991).

Information hiding

Information hiding is the concept of shielding unnecessary information from the programmer using a program segment and limiting communication to a well defined interface.

Subroutines and procedures, as used in Fortran (e.g. Müller and Streker, 1984), Pascal (Jensen and Wirth, 1975) or C (e.g. Haribson and Steele, 1991), were originally introduced to shorten program code by separating and reusing code which is used at different places within a program. Such subroutines and procedures are a first step towards information hiding, because they shield their mechanism by the header which defines the interface. The availability of large subroutine libraries, e.g. for solving numerical problems or for graphics (e.g. IMSL, 1993; NAG, 1993), demonstrate the success of this concept. Program modules as used in Modula-2 (Wirth, 1985) are a further step towards information hiding. Modules make it possible to design much more general interfaces to a program segment than procedure headers and they separate the interface (definition module) from its implementation (implementation module). An interface of a module typically uses procedures to manipulate the hidden data. By a clear separation of interfaces and implementations and by the elimination of side effects (global variables!) readability and ease of maintenance of a program can be extensively increased.

Abstract data types

Abstract data types combine the definition of a data structure together with the operations designed for manipulating and using the data.

Procedures or functions integrated into a data structure are called methods or member functions. Usually, dedicated methods may be provided for initialization and destruction of the data structure. The important difference to conventional data structures as used in Pascal, Modula-2 or C is the inclusion of the methods into the type definition. The technique of accessing variables indirectly by member functions makes it possible to include information hiding by shielding data from direct external

access. Data can then only be modified by a call to a member function, which checks consistency of the data. The important difference to modules is that abstract data types are not realizations of data structures but are new data types available for general use in the program. Any number of objects of such a type can be declared within the program. Some languages (e.g. C++) allow usual operators like +, -, *, /, ^, <, >, <=, >=, ==, =, [], etc. to be overloaded with a procedure specific for such data types. This feature simplifies usage of abstract data types for complex numbers, vectors, matrices, etc..

Inheritance

Inheritance makes it possible to derive new abstract data types from previously defined types so that the features of the old type remain usable and only the modifications necessary for the new type have to be implemented.

Abstract data types with the possibility of inheritance are called classes. Realizations of classes are objects. The concept of inheritance increases reusability of code by making information hiding available not only to a programmer using an unmodified class but also to a programmer modifying the class. The goal of object-oriented programming is to construct hierarchies of classes which can optimally reuse tested (and unmodified) classes as base classes within the derivation hierarchy and which only need (relatively) few specific modifications at each derivation step.

Polymorphism (dynamic binding)

Polymorphism is the capability of class hierarchies, that calling the same method of a base class can result in different actions for different derived classes.

Polymorphism is technically realized by dynamic binding (binding of such functions is not done during compilation but during execution). Polymorphism increases flexibility and extensibility of a program mainly by eliminating conditional branching. It makes it possible to introduce new classes without changing those parts of existing code which only use dynamically bound methods of base classes. Large parts of the program can then be realized independently of the exact type of objects used. This main advantage of object-oriented programming is used and discussed in detail in section 7.3.

7.2 Implementation of a List of Symbols

The objects of the four subsystems of model structure described in chapter 4, variables, processes, compartments and links, are identified by their names. Therefore, a data structure implementing a list of names (symbols) has to be designed. Since this data structure has an important position in the class hierarchy of the program, and because its implementation is a good example for demonstrating object-oriented programming techniques, it is described in detail in this section. The section starts with the description of a conventional implementation of such a list. Then the disadvantages of this implementation are discussed, and it is demonstrated how these disadvantages can be overcome with an object-oriented implementation. The comparison shows, that the superiority of the object-oriented implementation consists of a lot of details, which all together lead to a significant improvement in robustness of the program with regard to programming errors. Although this is a technical aspect of the program, this advantage should not be underestimated for large programming projects. ***The discussion in this section concentrates on technical advantages of object-oriented program design.*** In the following sections, conceptual advantages of object-oriented programming (polymorphism) are discussed.

7.2.1 Conventional Implementation

The program fragments (7.1) and (7.2) show a C implementation of a conventional list of symbols as described e.g. in Wirth (1986) for Modula-2:

```
struct SYMNODE
{
   char*            symbol;

   struct SYMNODE* prev;
   struct SYMNODE* next;
};
```

(7.1)

```
struct SYMLIST
{
   struct SYMNODE* first;
   struct SYMNODE* last;
   struct SYMNODE* current:
};
```

(7.2)

The structure `SYMNODE` contains a string (in C a pointer to the first character) and two pointers to the previous node and to the next node (a "*" following a type name declares a pointer to this type). For the first node of the list, `prev` is set to zero, for the last node, `next` is set to zero. The structure `SYMLIST` contains pointers to the

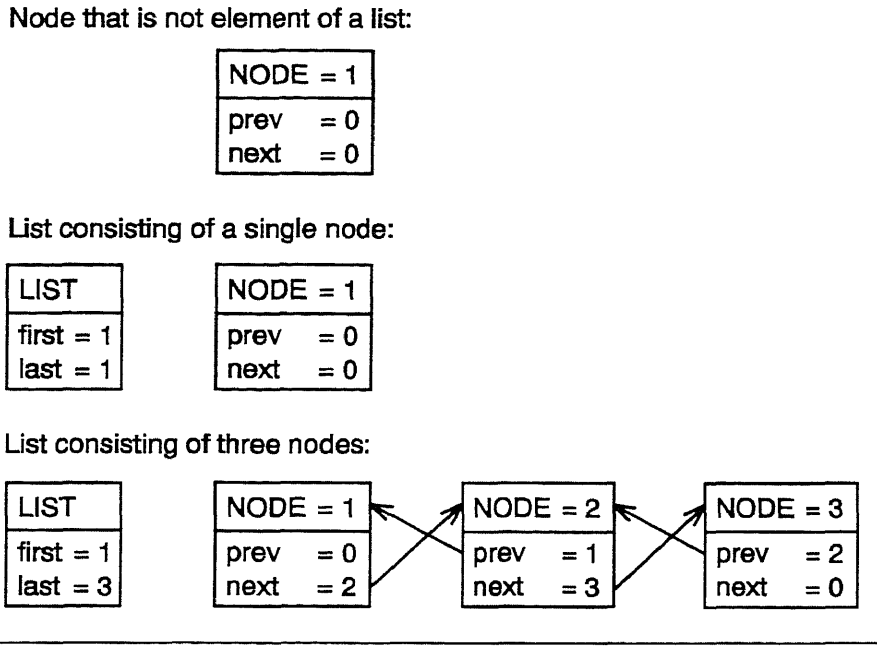


Fig. 7.1: Conventional implementation of a list ("1", "2" and "3" symbolize addresses of nodes).

first, the last and the current node. Functions have to be provided which change the pointers in a node and its neighboring nodes when a node is inserted or removed. Fig. 7.1 illustrates this mechanism of lists. The conventional implementation of a list given by the program fragments (7.1) and (7.2) has the following disadvantages:

1a The structures `SYMNODE` and `SYMLIST` are specifically designed for lists of symbols. If one also wants to have lists of something else (e.g. of variables, processes, compartments or links), the insertion and removal functions have to be rewritten (or copied and modified), since they use different structures as arguments.

2 The list given in program fragment (7.2) cannot protect its nodes against erroneous manipulations of the pointers to the neighboring nodes (e.g. it cannot be decided, if a node is already element of a list consisting of only one element (see Fig. 7.1), and therefore, a node which is element of a list, can be inserted into a second list, an operation which corrupts the first list).

3 The data of the node, in the case of program fragment (7.1) the string `symbol`, is not protected against illegal changes. Such accidental changes may lead to inconsistencies within the program.

4 The data structure is not automatically initialized when it is created, and the memory area allocated for the string is not automatically freed if the structure gets out of scope.

The first of these disadvantages can be eliminated with the following modification of the data structures given in program fragments (7.1) and (7.2):

```
struct DATANODE
{
   void*             data;

   struct DATANODE* prev;
   struct DATANODE* next;
};
```
(7.3)

```
struct DATALIST
{
   struct DATANODE* first;
   struct DATANODE* last;
   struct DATANODE* current:
};
```
(7.4)

In comparison to (7.1) in the program fragment (7.3) the data entry `char* symbol` of a string is replaced by a pointer to an arbitrary data type `void* data`. Now, the list (7.4) can be used for nodes connected to arbitrary data types. This advantage, however, leads to the following new disadvantage:

1b The type checking mechanism of the data entry is inactivated by using a pointer of type `void` to a data structure of an arbitrary type. Therefore, handling nodes of a wrong type can no more be recognized by the compiler.

Most of the disadvantages of the list constructions of this section cannot be eliminated using C or another non object-oriented programming language.

7.2.2 Object-Oriented Implementation

Elementary data types are implemented differently on different machines. This may have important consequences concerning memory requirements and accuracy of calculations. Therefore, for the implementation of a portable program, it is advantageous to use its own data types. In C++ this can be realized with the following statements:

```
typedef unsigned int CARDINAL;
typedef int          INTEGER;
typedef double       REAL;
```
(7.5)

These statements define a nonnegative integer type `CARDINAL`, an integer type `INTEGER` and a real number type `REAL`. Using only these new elementary data types has the advantage, that actually used data types can be changed by changing only one statement or by using machine dependent preprocessor directives for these definitions. To eliminate a deficiency of the languages C and C++, a boolean data type is added:

```
enum BOOLEAN { FALSE, TRUE };                                    (7.6)
```

Finally, to handle numerical problems and user interrupts, the data type

```
enum JOBSTATUS { OK, INTERRUPT, PROBLEM };                       (7.7)
```

is introduced. The five data types defined in (7.5), (7.6) and (7.7) are used in the following program segments illustrating the implementation techniques of the program.

In an object-oriented implementation of a list, a base node can be realized which does not contain a data entry. A node for a particular data type can then be derived from such a base node inheriting the connection mechanism. The program fragment (7.8) shows a possible implementation of a base node in C++:

```
class NODE
{
      friend class LIST;

   public:

      NODE();
      NODE(const NODE& node);
      NODE(const NODE* node);                                    (7.8)

      BOOLEAN InList() const;

      NODE*   Prev() const;
      NODE*   Next() const;

   private:

      NODE*   prev;
      NODE*   next;
};
```

This class definition contains data entries as well as member functions acting on the data. The keyword `public` starts the section of functions or variables, which are available to a programmer using the class. The functions and variables after the keyword `private` can only be accessed by member functions or by friends of the class. The line `friend class LIST` allows the member functions of the class `LIST` to access the private members of the class `NODE`. The declaration of the pointers `prev` and `next` to the neighboring nodes as private members and the declaration of the class `LIST` as a friend of the class `NODE` make a change of these pointers from outside of the member functions of the classes `NODE` and `LIST` impossible. This mechanism protects these pointers from unintentional changes by a programmer using these classes.

The member function `NODE()` with the same name as the class, a so-called constructor, is automatically called when a node is created. It performs initialization (in this case, it sets the pointers to the neighboring nodes to zero). The copy constructors `NODE(const NODE& node)` and `NODE(const NODE* node)`, which use a reference of a node or a pointer to a node as the argument, do not copy the pointers to the neighboring nodes (as would be the default), but set these pointers of the new node to zero. This saves the integrity of the lists by guaranteeing, that new nodes are not element of a list. A destructor with the name `~NODE()` would be called if a

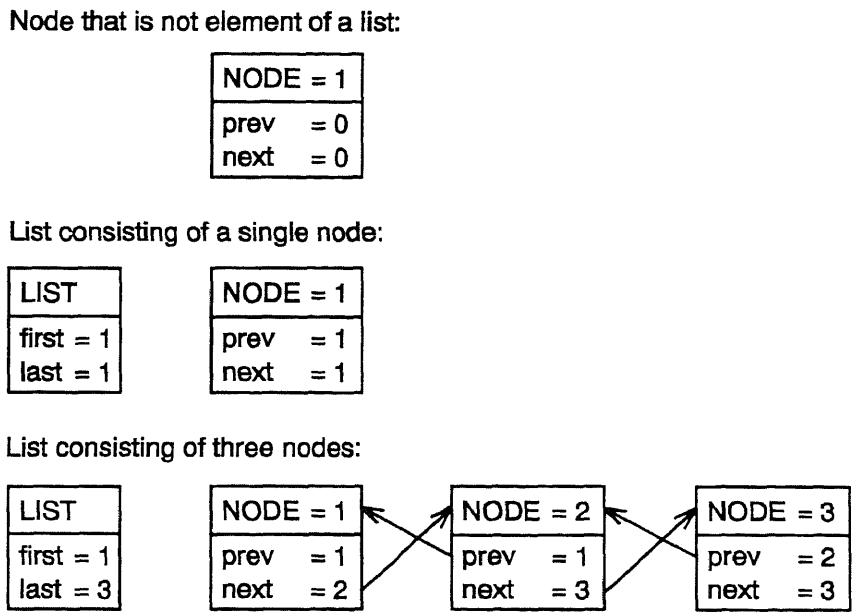


Fig. 7.2: Modified list implementation (compare with Fig. 7.1) allowing to decide if a node is element of a list ("1", "2" and "3" symbolize addresses of nodes).

node is deallocated or if it gets out of scope. In this case, such a function is not needed.

The following modification to the usual treatment of the first and last nodes of the list is realized: The pointer `prev` in the first node of the list and the pointer `next` in the last node of the list are not set to zero, but to the address of the node itself (this is a value which is distinguishable from zero and from all possible addresses of other nodes). Fig. 7.2 illustrates this small modification which can also be realized in a conventional programming language. This modification allows the member function `InList()` of the class `NODE` to decide, if the node is an element of a list (by testing if `next` or `prev` is zero). As is shown later, this member function can be used to avoid insertion of a node, which is element of a list, into a second list, an operation which corrupts the first list. Furthermore, this member function can be used to protect entries in a class derived from the class `NODE` from being changed, if the class is element of a list. Such changes can lead to inconsistencies as is discussed in section 7.4. The keyword `const` after the function `InList()` indicates, that this function does not change elements in the node, so that it may also be applied to a node which is declared to be constant. The member functions `Prev()` and `Next()`, also declared constant, return the current values of the pointers `prev` and `next`, if these values are not identical to the address of the node itself, in which case zero is returned. This allows a programmer to browse through a list with the usual address convention (note that the pointers `prev` and `next` cannot be accessed directly from outside of the classes `NODE` and `LIST`).

The following program fragment (7.9) shows a possible implementation of a list class.

```
class LIST
{
   public:

      LIST();

      BOOLEAN  Empty() const;
      CARDINAL Size() const;
      CARDINAL Position() const;
      CARDINAL Position(const NODE* node) const;
      NODE*    First() const;
      NODE*    Last() const;
      NODE*    Current() const;

      BOOLEAN  Goto(const NODE* node);
      BOOLEAN  Goto(CARDINAL pos);
      BOOLEAN  GotoFirst();
      BOOLEAN  GotoLast();
      BOOLEAN  GotoNext();
      BOOLEAN  GotoPrev();
      BOOLEAN  GotoNextOrFirst();
      BOOLEAN  GotoPrevOrLast();

      BOOLEAN  InsertFirst(NODE* node);
      BOOLEAN  InsertLast(NODE* node);
      BOOLEAN  Insert(NODE* node, CARDINAL pos):
      BOOLEAN  Insert(NODE* node);
      BOOLEAN  InsertAfter(NODE* node);
      BOOLEAN  Replace(NODE* node);
      BOOLEAN  RemoveFirst();
      BOOLEAN  RemoveLast();
      BOOLEAN  Remove();
      BOOLEAN  Remove(NODE* node);
      BOOLEAN  Remove(CARDINAL pos);
      BOOLEAN  RemoveAll();

   private:

      NODE*    first;
      NODE*    last;
      NODE*    current;
};
```

(7.9)

This class definition contains mainly member functions for giving information about the state of the list and for manipulating the list. The constructor `LIST()` sets all variables (`first`, `last`, `current` and `size`) to zero. The group of member functions from `Empty()` to `Current()` gives information about the current state of the list without doing any changes. Therefore, these functions can be declared `const` and can also be applied to a list, which is declared to be constant. Note that in C++ functions identifiable by different types of the arguments may be overloaded using the same name (`Position()` without an argument returns the position of the current node, whereas `Position(const NODE* node)` with the address of a node as the argument returns the position of this node, or zero if the node is not element of the list). The member functions from `Goto(...)` to `GotoPrevOrLast()` change the pointer `current`, which marks the current node in the list. It is possible to set the current node to a given node, to a given position in the list or to move it noncyclic or cyclic forward or backward. The last group of member functions from `InsertFirst(...)` to `RemoveAll()` makes it possible to insert, replace and remove

nodes from the list. Each of the insertion functions checks by a call to `InList()` if the node is already element of a list and rejects insertion if this is the case. The pointers to the neighboring nodes of a node removed from a list are set to zero, so that this node can be reinserted into any list. The data entries of the list are declared private to protect them from being unintentionally changed. Changes of the pointers to the first, last or current node can only be done by the member functions of the class `LIST`.

The list defined by the program fragments (7.8) and (7.9) does not contain a data entry. To realize a list of symbols, a new list node containing a string is needed. Object-oriented programming techniques allow such a node to be derived from the universal node given in program fragment (7.8) with inheritance of the connection mechanism. The program fragment (7.10) shows the implementation of such a node:

```
class SYMBOL : public NODE
{
   public:

      SYMBOL();
      SYMBOL(const SYMBOL& sym);
      SYMBOL(const SYMBOL* sym);
      virtual ~SYMBOL();

      SYMBOL*      Prev() const;
      SYMBOL*      Next() const;

      const char*  Symbol() const;
      BOOLEAN      Symbol(const char* sym);

   private:

      char*        symbol;
};
```

(7.10)

The first line of this class definition indicates, that the class `SYMBOL` is derived from the class `NODE`, inheriting its elements and member functions.

The constructor `SYMBOL()` without arguments initializes the symbol with a default text; the alternative copy constructors `SYMBOL(const SYMBOL& sym)` and `SYMBOL(const SYMBOL* sym)`, which take a reference of a symbol node or a pointer to a symbol node as the argument, initialize the symbol with the same text as is used in the symbol node given as the argument (note that the pointers to the neighboring nodes are not copied, but set to zero to guarantee integrity of the lists as described for the copy constructors of the class `NODE` given in program fragment (7.8) which do this job automatically). The destructor `~SYMBOL()` frees memory when the symbol node is deallocated or when it gets out of scope. It is declared `virtual` to ensure that the destructor of the actual derived class is called.

The member functions `Prev()` and `Next()` only return the results of the inherited member functions `Prev()` and `Next()` of the class `NODE` with a type conversion to `SYMBOL*`.

The member function `Symbol()` makes the symbol available to a user of the class. The first keyword `const` in the declaration of this function indicates that this function returns a constant string, which cannot be changed by the programmer obtaining its

address with this function. This is a very useful feature of C++ because if it would not exist, the whole protection mechanism of class elements would be useless for strings, which are pointers to its first character (in order to maintain compatibility with C). The second keyword `const` in the declaration of the function `SYMBOL()` indicates, that this function does not change the symbol node, so that it can also be called for a constant symbol node. The only way to change the symbol is by a call to the member function `Symbol(const char* sym)` which has a constant string as the argument. This function only replaces the old symbol if the symbol is not element of a list (this feature is realized by a check using the member function `InList()` inherited from the class `NODE` (7.8) and saves integrity of the lists of symbols; cf. section 7.4 for a discussion of editing concepts) and if the syntax of the new symbol is correct (the use of symbols in algebraic expressions limits their syntax).

In a similar way as the derivation of the class `SYMBOL` from the class `NODE`, a class `SYMBOLLIST` can be derived from the class `LIST`. This class makes a new interface to the list which only allows symbol nodes to be inserted into the list. This interface uses the member functions inherited from the class `LIST` to do the pointer manipulations and can therefore not introduce new errors concerning the connections within lists (this is possible, because a pointer to an object of the class `SYMBOL` is also a pointer to an object of the class `NODE` contained in the symbol node and can therefore be given as an argument to the member functions of the class `LIST` described in program fragment (7.9)). As an additional feature, the class `SYMBOL-LIST` can provide a member function for an alphabetically sorted insertion of symbol nodes.

A reexamination of the disadvantages of the classical C implementations discussed at the beginning of this section leads to the following conclusions:

1a A class containing any data type can be derived from an object of the type `NODE` (7.8). Therefore, such nodes are universally usable.

1b Strong type checking is guaranteed, since different data types derived from the class `NODE` are distinguishable to the compiler.

2 The nonconstant member functions of the classes `NODE` (7.8) and `LIST` (7.9) are the sole functions, which are allowed to manipulate the pointers realizing the connections of the list. When these functions are correctly implemented and a test to `InList()` is performed before inserting a node, it is not possible to erroneously manipulate the pointers. A list can no longer become corrupted.

3 The symbol of an object or the class `SYMBOL` (7.10) is only accessible as a constant string and changes can only be made by a call to a member function which checks the legality of the new symbol. Illegal symbols can no longer be assigned.

4 After implementation of correct constructors and destructors, all objects are initialized and freed automatically.

These features make the program much more robust against programming errors than a conventional implementation according to the program fragments (7.1) - (7.4).

They are implemented using information hiding and inheritance. In section 7.3 it is shown how polymorphism can increase the extensibility of the program.

To provide input and output procedures for loading, saving and printing, the definition of the class SYMBOL according to (7.10) is revised to not only being derived from the class NODE but also from a class FILEIO, the main elements of which are shown in the program fragment (7.11):

```
class FILEIO
{
   public:

      FILEIO();
      FILEIO(const FILEIO& fio);
      FILEIO(const FILEIO* fio);
      virtual ~FILEIO();

      static BOOLEAN    Print(ostream& out, const char* string,
                              CARDINAL pos=0)
      static BOOLEAN    Print(ostream& out, CARDINAL num,
                              CARDINAL pos=0)
      static BOOLEAN    Print(ostream& out, INTEGER num,
                              CARDINAL pos=0)
      static BOOLEAN    Print(ostream& out, REAL num,
                              CARDINAL pos=0)
      static BOOLEAN    SaveString(ostream& out, const char* str);
      static BOOLEAN    SaveNumber(ostream& out, CARDINAL num);
      static BOOLEAN    SaveNumber(ostream& out, INTEGER num);
      static BOOLEAN    SaveNumber(ostream& out, REAL num);
      static BOOLEAN    SaveBoolean(ostream& out, BOOLEAN b);
      static BOOLEAN    LoadString(istream& in, char* str);
      static BOOLEAN    LoadNumber(istream& in, CARDINAL* num);
      static BOOLEAN    LoadNumber(istream& in, INTEGER* num);
      static BOOLEAN    LoadNumber(istream& in, REAL* num);
      static BOOLEAN    LoadBoolean(istream& in, BOOLEAN* b);

      BOOLEAN           Saved() const;                          (7.11)
      void              Changed();

      virtual JOBSTATUS Save(ostream& out);
      virtual JOBSTATUS Write(ostream& out) const;

   private:

      BOOLEAN           saved;
};
```

The member functions Print(...) of this class are used for writing the system definitions to a print file, declared as a reference to the C++ standard output stream data type ostream. An argument pos is included into the definition of these functions in order to define the start position in the currently written output line. The default value of pos is equal to zero. This results in starting at the current position. The member functions SaveString(...), SaveNumber(...), SaveBoolean(...), LoadString(...), LoadNumber(...) and LoadBoolean(...) serve to support a standard file format for system files of the program. The files are declared as references to the C++ standard input and output stream types istream and ostream. All of the above-mentioned functions are declared to be static member functions of the class to make their use from code external to the class possible. The member function Saved() is used to ask, if the class has been saved since the last

changes have been made, and with the function `Changed()`, the status can be changed when the object is edited. The member functions `Save(ostream& out)` and `Write(ostream& out)` are declared here to be overwritten in derived classes. The implementation of these functions in derived classes will write the data of the derived classes to the output stream given as the argument. The keyword `virtual` guarantees, that the instances of the actual derived class of these functions are called. The function `Save(ostream& out)` cannot be declared constant, because this function changes the boolean variable `saved` of the class.

To increase the possibilities of documentation of the symbol nodes, an arbitrary description is allowed in addition to the symbol. With the additional derivation from the class `FILEIO` and the inclusion of a description, the definitive implementation of the class `SYMBOL` changes from (7.10) to:

```
class SYMBOL : public NODE, public FILEIO
{
   public:

      SYMBOL();
      SYMBOL(const SYMBOL& sym);
      SYMBOL(const SYMBOL* sym);
      virtual ~SYMBOL();

      SYMBOL*      Prev() const;
      SYMBOL*      Next() const;

      const char*  Symbol() const;
      BOOLEAN      Symbol(const char* sym);

      const char*  Description() const;
      void         Description(const char* des);

   private:

      char*        symbol;
      char*        description;
};
```

(7.12)

Fig. 7.3 shows the derivation hierarchy of the class `SYMBOL`. From this class the classes for variables, processes, compartments and links are derived. The main elements of these classes are described in the following section of this chapter.

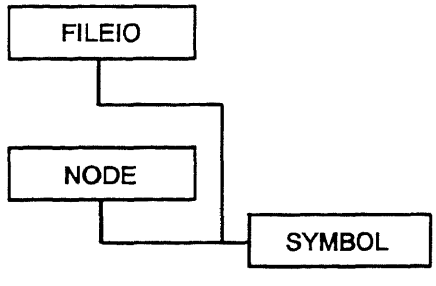


Fig. 7.3: Derivation hierarchy of the class `SYMBOL`.

7.3 Derivation Hierarchies of Model Subsystems

As described in chapter 4, the general structure of mathematical models of aquatic systems as used in this report is based on a division of the system into the four subsystems of variables, processes, compartments and links. In the following subsections, the implementation concepts of these four subsystems and their integration to a data analysis program are discussed. ***The central point of the implementation of model subsystems is the use of polymorphism for subclass independent implementation of important features. This technique not only saves development time by the reuse of tested code inherited from base classes, but it is the main concept guaranteeing the extensibility of the program.***

7.3.1 System of Variables

As characterized in section 4.1, variables have the common feature of being able to return a numeric value. ***The main idea of the derivation hierarchy of variables is to construct a base class*** `VAR` ***with the virtual function*** `Evaluate()` ***which is designed for calculating and returning the value of the variable. This function is not implemented in the base class, but it is realized specifically for each subclass derived from the class*** `VAR`***, which represent one of the variable types described in section 4.1. By the use of polymorphism (dynamic binding), numerical calculations and plotting and listing of results can then be implemented independently of the actual types of variables. By providing virtual functions for consistency checks and editing of the system of variables, also these tasks can be implemented independent of the variable type.*** These features make future extensions by additional types of variables quite simple. Since variables are identified by their names and they have to be inserted into a list of variables, their base class `VAR` is derived from the class `SYMBOL` (7.12) described in the previous section.

The program fragment (7.13) shows the main elements of the base class for variables:

```
class VAR : public SYMBOL
{
   public:

      VAR();
      VAR(const VAR& var);
      VAR(const VAR* var);
      ~VAR();
```

```
        VAR*              Prev() const;
        VAR*              Next() const;

        virtual VARTYPE   Type() const = 0;

        virtual REAL      Evaluate() = 0;
        REAL              Value();
        void              Reset();

        const char*       Unit() const;
        void              Unit(const char* u);

        virtual CARDINAL  NumVarArg() const;
        virtual VAR*      VarArg(CARDINAL index) const;
        BOOLEAN           Arg(const VAR* var) const;

        virtual BOOLEAN   ReplaceVarArg(VAR* oldvar, VAR* newvar);

        JOBSTATUS         Load(istream& in);
        JOBSTATUS         Save(ostream& out);
        JOBSTATUS         Write(ostream& out) const;

    private:

        char*             unit;

        REAL              lastvalue;
        BOOLEAN           evaluated;
};
```

(7.13)

The class VAR is derived from the class SYMBOL inheriting the member functions of the classes SYMBOL (7.12), NODE (7.8) and FILEIO (7.11). This gives objects of this class a symbol and a description, makes it possible to insert them into a list, and provides functions for input and output to a file.

The constructor VAR() without arguments initializes the variable with default data; the alternative copy constructors VAR(const VAR& var) and VAR(const VAR* var) initialize the variable with data copied from the variable referenced as the argument. As described for the copy constructors of the class NODE (7.8), which are automatically called by these constructors, the addresses of the neighboring nodes are not copied, so that the new variable is not member of a list. The destructor ~VAR() frees memory allocated by the variable.

The member functions Prev() and Next() only return the results of the inherited member functions Prev() and Next() of the class SYMBOL (7.12) (and therefore of the class NODE (7.8)) with a type conversion to VAR*.

The virtual member functions Type() (returning the variable type) and Evaluate() are set to zero to indicate that the implementation will be given in derived classes. This makes it impossible to realize an object of type VAR. It is only possible to realize objects of derived classes which implement these functions.

The member function Evaluate() implemented in the derived class, evaluates and returns the value of the variable. Since this member function not obviously changes data of the class, it seems to be a good idea to declare this function to be constant, to let the compiler detect possible harm caused by an erroneous implementation of this function. This is not done for efficiency reasons: real list variables and variable list variables, which interpolate in data lists, have to search the list for the neighboring elements of a given argument. Because succeeding calls usually have neigh-

boring arguments, the efficiency of this search process can strongly be increased by storing the position of the last call. This procedure requires to change this position variable of the data list and therefore makes a constant declaration of the function `Evaluate()` impossible.

Because of functional dependences, variables can appear several times as arguments within other variables. Such dependences cause multiple evaluation of variables in the same state of the system of variables. For variables the evaluation of which is computationally intensive (because of long dependence chains or because of interpolation or smoothing), multiple evaluations unnecessarily diminish efficiency of the program. To avoid multiple evaluations, the member function `Value()` is used instead of `Evaluate()` in the numerical part of the program. At the first call after a reset of the system of variables, this function evaluates the variable by a call to `Evaluate()`, saves the value in the variable `lastvalue`, and sets the variable `evaluated` to `TRUE`. Subsequent calls just return the value stored in `lastvalue` until the variable is reset by a call to `Reset()` which sets `evaluated` to `FALSE`. This whole mechanism of remembering old values can be implemented in the base class `VAR` of the variables and works independent of variable type (for this reason, the function `Value()` must not be declared as a virtual function, in contrast to the function `Evaluate()`).

The member functions `Unit()` and `Unit(const char* u)` serve to inquire and set the unit of the variable. The unit is stored as a string in the variable `unit`.

The virtual member function `NumVarArg()` returns the number of direct arguments of the variable, the addresses of which can be obtained with the aid of the virtual member function `VarArg(CARDINAL index)`. The member function `Arg(const VAR* var)` checks recursively, if the current variable depends on the variable `var` given as the argument. Both of these functions are required for checks of consistency of the system of variables, which can be performed with the aid of member functions of the class `VAR` independent of the type of the variable to be checked. The virtual member function `ReplaceVarArg(...)` is used for editing the system of variables. It is designed to replace a direct argument of a variable by another variable. Its use is explained in more detail in section 7.4.

The member function `Load(...)` is used to read general data for variables from the system file. The member functions `Save(...)` and `Write(...)` implement virtual member functions of the class `FILEIO` (7.11). All these three functions will be called by the more specific functions of the derived classes for different variable types. File input and output operations performed by these functions use the functions provided for this purpose by the base class `FILEIO`.

The different variable types required for the formulation of a model as described in section 4.1, are represented by classes derived from the base class `VAR` given in program fragment (7.13). Fig. 7.4 gives an overview of the derivation hierarchy of variables. For each derived class, the virtual member functions `Type()`, `Evaluate()`, `NumVarArg()`, `VarArg(...)` and `ReplaceVarArg(...)` of the class `VAR`, the virtual member functions `Save(...)` and `Write(...)` of the class `FILEIO`, a member function `Load(...)` (which calls `VAR::Load(...)` for loading

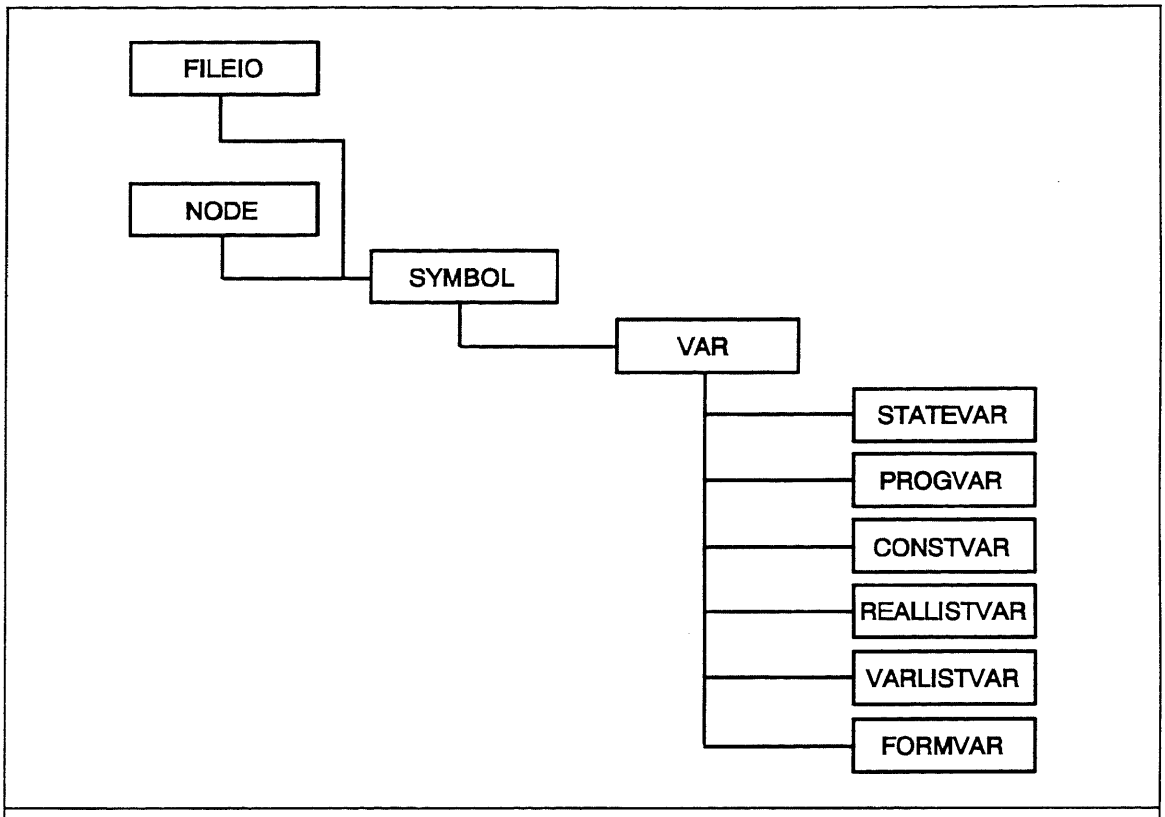


Fig. 7.4: Derivation hierarchy the classes implementing variables.

general data) and member functions for editing type specific data have to be provided.

The additional data entries of the derived class `STATEVAR` for state variables are the type of the state variable, its relative and absolute accuracy and a pointer to the real number to be returned by the member function `Evaluate()`. The data entries for the derived class `PROGVAR` for program variables are its type and a pointer to the real number which is returned by the member function `Evaluate()`. Both these pointers of state variables and of program variables have to be set by the member function of the compartment, which implements the differential-algebraic system of equations to be solved (cf. section 7.3.3). The data entries of the class `CONSTVAR` representing constant variables are its value, standard deviation, minimum, maximum and two boolean variables indicating if the variable is active for sensitivity analysis and for parameter estimation. The data entries for the derived class `REALLISTVAR` for real list variables are arrays of the values of the argument, of the variable and of individual standard deviations, relative and absolute global standard deviations, minimum and maximum and variables indicating, if global or individual standard deviations have to be used, which type of interpolation has to be applied (including smoothing width) and if the variable is active for sensitivity analysis. The derived class `VARLISTVAR`, which represents variable list variables, needs arrays of the values of the argument and of variables corresponding to the values of the argument and information on the interpolation method to be used. Finally, as an example,

program fragment (7.14) shows the definition of the class FORMVAR, which implements formula variables:

```
class FORMVAR : public VAR
{
   public:

      FORMVAR();
      FORMVAR(const FORMVAR& var);
      FORMVAR(const VAR* var);
      ~FORMVAR();

      VARTYPE   Type() const;

      REAL      Evaluate();

      CARDINAL  NumVarArg() const;
      VAR*      VarArg(CARDINAL index) const;

      BOOLEAN   ReplaceVarArg(VAR* oldvar, VAR* newvar);

      int       Parse(const char* from_line,
                      const VARLIST* varlist,
                      char* errorline);
      int       UnParse(char* to_line, int maxch) const;

      JOBSTATUS Load(istream& in, const VARLIST* varlist);
      JOBSTATUS Save(ostream& out);
      JOBSTATUS Write(ostream& out) const;

   private:

      Fnode*    root;
};
```

(7.14)

In this case, Evaluate() calculates the result of the algebraic expression parsed by the function Parse(...) and stored in a tree data structure, the starting address of which is given by the variable root (the type Fnode of this variable is not documented in this report). For parsing an algebraic expression passes as a string, a pointer to the list of variables already defined has to be given as an argument. The member function UnParse(...) recovers the algebraic expression as a string, which is used for saving, printing and for writing the expression into the edit field of the dialog box used for editing the variable.

All variables building the system of variables are collected in a list of variables of the type VARLIST. As shown in Fig. 7.5 this class is derived from the class LIST (7.9) via the intermediate class SYMBOLLIST. The insertion and removal functions of the class VARLIST are made private, so that they can only be used by the member functions of the class VARSYS, which represents the system of variables and which is a friend of the class VARLIST. The class VARSYS consists of a VARLIST and of member functions for adding, replacing, exchanging and deleting variables:

```
class VARSYS : public FILEIO
{
   public:

      VARSYS();
      ~VARSYS();

      void      Reset();
```

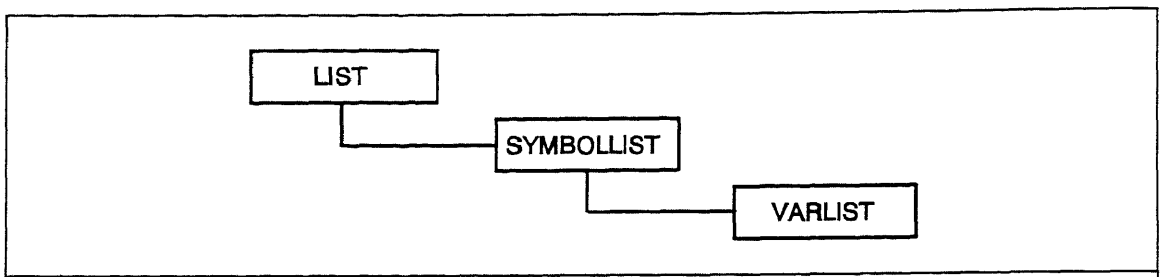


Fig. 7.5: Derivation hierarchy of the class VARLIST.

```
        VAR*        Get(const char* sym) const;

        BOOLEAN     Add(VAR* var);
        BOOLEAN     Replace(VAR* oldvar, VAR* newvar);
        BOOLEAN     Exchange(VAR* var1, VAR* var2);
        BOOLEAN     Delete(VAR* var);

        BOOLEAN     Arg(const VAR* var) const;

        VARLIST*    Varlist();

        JOBSTATUS   Load(istream& in);
        JOBSTATUS   Save(ostream& out);
        JOBSTATUS   Write(ostream& out) const;

    private:

        VARLIST   varlist;
};
```

(7.15)

The member function `Reset()` resets all variables of the system of variables (compare comment to the member function `Reset()` of the class `VAR` defined in program fragment (7.13)). The member function `Get(const char* sym)` returns a pointer to the variable with the symbol given as the argument. The member functions `Add(VAR* var)`, `Replace(VAR* oldvar, VAR* newvar)`, `Exchange(VAR* var1, VAR* var2)` and `Delete(VAR* var)` are used for editing the system of variables. The functions `Replace(...)` and `Exchange(...)` not only replace the variable in the list of variables, but additionally, by a call to the member function `ReplaceVarArg(...)` of the class `VAR`, in all variables in which it appears as an argument. The member function `Arg(const VAR* var)` checks, if the variable given as the argument is an argument of another variable. Such checks are necessary for maintaining the consistency of the system of variables during edit processes as described in more detail in section 7.4 (e.g. a variable which is an argument of another variable must not be deleted). Finally, the member functions `Load(...)`, `Save(...)` and `Write(...)` are used for loading and saving the system of variables and for writing it to a print file. Each of these functions mainly calls the functions with the same name of all variables in the list (cf. program fragment (7.13)).

7.3.2 System of Processes

As described in section 4.2, processes calculate transformation rates of dynamic state variables or equilibrium conditions corresponding to given equilibrium state variables. ***The main purpose of the base class PROC for processes is to provide a type-independent interface for consistency checks and model editing.*** Furthermore, because transformation rates of different processes have to be summed to the total transformation rates for all substances, a virtual function can be provided by this base class to add the rate of the actual process to the sum of rates calculated before. In the case of an equilibrium process, this rate is interpreted as the deviation of the equilibrium condition from zero. Since processes are identified by their names and they have to be inserted into a list, their base class PROC is derived from the class SYMBOL (7.12) described in section 7.2.

The program fragment (7.16) shows the main elements of the base class for processes:

```
class PROC : public SYMBOL
{
   public:

      PROC();
      PROC(const PROC& proc);
      PROC(const PROC* proc);
      ~PROC();

      PROC*            Prev() const;
      PROC*            Next() const;

      virtual PROCTYPE Type() const = 0;

      virtual void     SumRates(STATEVAR* const* statevar,
                                CARDINAL numstatevar,
                                REAL* rates) = 0;

      virtual CARDINAL NumVarArg() const;
      virtual VAR*     VarArg(CARDINAL index) const;
      BOOLEAN          Arg(const VAR* var) const;

      virtual BOOLEAN  ReplaceVar(VAR* oldvar, VAR* newvar);
      virtual BOOLEAN  ExchangeVar(VAR* var1, VAR* var2);

      JOBSTATUS        Load(istream& in);
      JOBSTATUS        Save(ostream& out);
      JOBSTATUS        Write(ostream& out) const;
};
```

(7.16)

The class PROC is derived from the class SYMBOL inheriting the member functions of the classes SYMBOL (7.12), NODE (7.8) and FILEIO (7.11). This gives objects of this class a symbol and a description, makes it possible to insert them into a list, and provides functions for input and output to a file.

The constructor PROC() without arguments initializes the process with default data; the alternative copy constructors PROC(const PROC& proc) and PROC(const PROC* proc) initialize the process with with data copied from the process referenced as the argument. As described for the copy constructors of the class NODE (7.8), which are automatically called by these constructors, the addresses of the

neighboring nodes are not copied, so that the new process is not member of a list. The destructor `~PROC()` frees memory allocated by the process.

The member functions `Prev()` and `Next()` only return the results of the inherited member functions `Prev()` and `Next()` of the class `SYMBOL` (7.12) (and therefore of the class `NODE` (7.8)) with a type conversion to `PROC*`.

The virtual member functions `Type()` (returning the process type) and `SumRates()` are set to zero to indicate that the implementation will be given in derived classes. This makes it impossible to realize an object of the type `PROC`. It is only possible to realize objects of derived classes which implement these functions. `SumRates(...)` adds the process rates of the state variables given as arguments to the corresponding components of the array of rates referenced as the last argument.

The virtual member functions `NumVarArg()`, `VarArg(CARDINAL index)` and `Arg(const VAR* var)` do the same job as do the member functions with the same name of the class `VAR` given in program fragment (7.13). The member functions `ReplaceVar(...)` and `ExchangeVar(...)` are used to update changes made in the system of variables.

The member functions `Load(...)`, `Save(...)` and `Write(...)` are used for loading from and saving to system files and for writing the system to a print file. All these three functions will be called by the more specific functions of the derived classes for different variable types. File input and output operations performed by these functions use the functions provided for this purpose by the base class `FILEIO`.

The two different process types as described in section 4.2 are represented by classes derived from the base class `PROC`. Fig. 7.6 gives an overview of the derivation hierarchy of processes. In addition to the implementation of the virtual functions of the class `PROC`, the class `DYNPROC` representing dynamic processes stores a common rate law and a list of stoichiometric coefficients, whereas the class `EQUPROC` for equilibrium processes has data entries for a variable and an algebraic expression

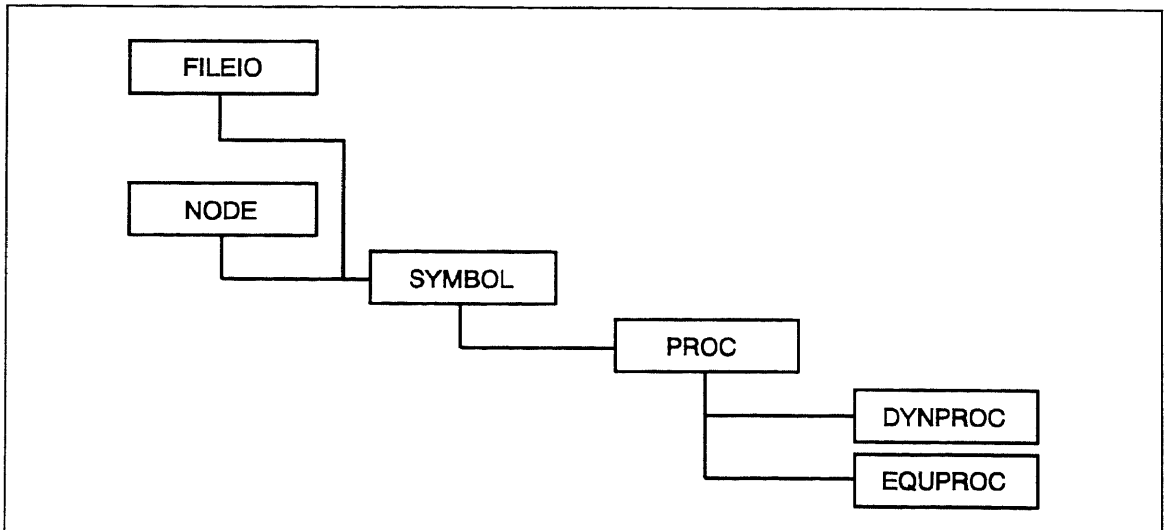


Fig. 7.6: Derivation hierarchy of the classes implementing processes.

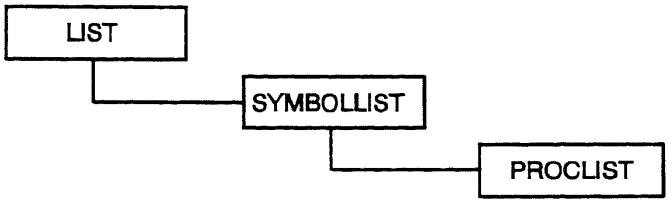


Fig. 7.7: Derivation hierarchy of the class PROCLIST.

describing the equation to be solved.

All processes building the system of processes are collected in a list of processes of the type PROCLIST. Fig. 7.7 shows the derivation hierarchy of this class which is analogous to the list of variables shown in Fig. 7.5. The insertion and removal functions of this class are made private, so that they can only be used by the member functions of the class PROCSYS, which represents the system of processes and which is a friend of the class PROCLIST. The class PROCSYS consists of a PROCLIST and of member functions for adding, replacing and deleting processes and for replacing and exchanging variables:

```
class PROCSYS : public FILEIO
{
   public:

      PROCSYS();
      ~PROCSYS();

      PROC*      Get(const char* sym) const;

      BOOLEAN    Add(PROC* proc);
      BOOLEAN    Replace(PROC* oldproc, PROC* newproc);
      BOOLEAN    Delete(PROC* proc);

      BOOLEAN    Arg(const VAR* var);

      void       ReplaceVar(VAR* oldvar, VAR* newvar);
      void       ExchangeVar(VAR* var1, VAR* var2);

      PROCLIST*  Proclist();

      JOBSTATUS  Load(istream& in, const VARLIST* varlist);
      JOBSTATUS  Save(ostream& out);
      JOBSTATUS  Write(ostream& out) const;

   private:

      PROCLIST   proclist;
};
```

(7.17)

The member function Get(const char* sym) returns a pointer to the process with the symbol given as the argument. The member functions Add(PROC* proc), Replace(PROC* oldproc, PROC* newproc) and Delete(PROC* proc) are used for editing the system of processes. The member function Arg(const VAR* var) checks if one of the processes depends on the variable given as the argument. Such checks are necessary for maintaining the consistency of the system during edit

operations as described in more detail in section 7.4. The member functions `ReplaceVar(VAR* oldvar, VAR* newvar)` and `ExchangeVar(VAR* var1, VAR* var2)` are necessary to update changes made in the system of variables. These functions reflect the dependence of the system of processes on the system of variables as shown in Fig. 4.5. Finally, the member functions `Load(...)`, `Save(...)` and `Write(...)` are used for loading and saving the system of processes and for writing it to a print file. Each of these functions mainly calls the functions with the same name of all processes in the list (cf. program fragment (7.16)).

7.3.3 System of Compartments

As outlined in chapter 6, spatial discretization of partial differential equations should use compartment specific techniques that account for the type of the equations. According to equation (6.1), the resulting system of ordinary differential equations and algebraic equations can be expressed by a vector valued function **G**(**y**,∂**y**/∂t,t), where the array **y** characterizes the current state at time t. Numerical software, as described in chapter 6, is usually written in Fortran and uses real arrays for information interchange with the problem-specific function **G**. Although object-oriented programming and problem-specific data structures facilitate the design of an identification and simulation program, the interface to the numerical integrator should follow usual standards to facilitate changes of numerical algorithms. Therefore, the main task of a class representing a compartment of an aquatic system is to provide a function **G** defining the spatially discretized equations of the compartment and returning its values as a real array. Each of the compartments defines a section of the function **G** of the whole aquatic system. ***The main idea of the implementation hierarchy of compartments is to provide a base class COMP, which provides a common interface of functions for the following purposes: Definition of the differential-algebraic system of spatially discretized equations, communication with links and consistency checks and model editing.*** The use of polymorphism for implementing these functions (implementation as virtual functions) makes the program extensible with regard to the addition of new compartment types in later program versions. The implementation of temporal integration algorithms, links, and model editing procedures need not be changed when new compartment types are added (with the exception of the need for editing the items specific for the new compartment). Besides these virtual functions, the base class COMP contains universal data for compartments such as lists of active variables and processes, input specifications and initial conditions. Since compartments are identified by their names and they have to be inserted into a list of compartments, their base class COMP is derived from the class SYMBOL (7.12) described in section 7.2.

The program fragment (7.18) shows the main elements of the base class for compartments:

```
class COMP : public SYMBOL
{
   public:

      COMP();
      COMP(const COMP& comp);
      COMP(const COMP* comp);
      ~COMP();

      COMP*              Prev() const;
      COMP*              Next() const;

      virtual COMPTYPE Type() const = 0;

      CARDINAL           NumActVar() const;
      VAR*               ActVar(CARDINAL index) const;
      BOOLEAN            AddActVar(VAR* var);
      BOOLEAN            RemoveActVar(VAR* var);

      CARDINAL           NumActProc() const;
```

```
PROC*              ActProc(CARDINAL index) const;
BOOLEAN            AddActProc(PROC* proc);
BOOLEAN            RemoveActProc(PROC* proc);

const FORMVAR*     Inflow() const;
BOOLEAN            Inflow(const char* parseline,
                          const VARLIST* varlist);
CARDINAL           NumInput() const;
VAR*               InputVar(CARDINAL index) const;
const FORMVAR*     InputFlux(CARDINAL index) const;
BOOLEAN            AddInput(VAR* var, const char* parseline,
                            const VARLIST* varlist);
BOOLEAN            DeleteInput(CARDINAL index);

CARDINAL           NumInitCond() const;
CARDINAL           InitZone(CARDINAL index) const;
VAR*               InitVar(CARDINAL index) const;
const FORMVAR*     InitVal(CARDINAL index) const;
BOOLEAN            AddInitCond(VAR* var, CARDINAL zone,
                               const char* parseline,
                               const VARLIST* varlist);
BOOLEAN            DeleteInitCond(CARDINAL index);

virtual CARDINAL NumEq() const;
virtual BOOLEAN  InitCond(REAL* Y) = 0;
virtual BOOLEAN  Delta(const REAL* Y, const REAL* YT,
                       REAL* G) = 0;
virtual BOOLEAN  GridValue(CARDINAL calcnum, REAL t,
                           const REAL* Y, CARDINAL j,
                           VAR* var, REAL& value) = 0;
                                                          (7.18)
virtual CARDINAL NumAdvInConn() const;
virtual CARDINAL NumAdvOutConn() const;
virtual CARDINAL NumDiffConn() const;
virtual BOOLEAN  AdvInExZ(CARDINAL conn) const;
virtual REAL     AdvInZ(const REAL* Y, CARDINAL conn);
virtual REAL     AdvOutQ(const REAL* Y, CARDINAL conn);
virtual REAL     AdvOutJ(const REAL* Y, CARDINAL conn,
                         const VAR* var);
virtual REAL     DiffC(const REAL* Y, CARDINAL conn,
                       const VAR* var);

virtual BOOLEAN  AllowedVar(const VAR* var) const;
virtual BOOLEAN  AllowedReplaceVar(const VAR* oldvar,
                                   const VAR* newvar) const;
virtual BOOLEAN  AllowedExchangeVar(const VAR* var1,
                                    const VAR* var2) const;
virtual BOOLEAN  AllowedProc(const PROC* proc) const;
virtual BOOLEAN  AllowedReplaceProc(const PROC* oldproc,
                                    const PROC* newproc) const;

virtual CARDINAL NumVarArg() const;
virtual VAR*     VarArg(CARDINAL index) const;
BOOLEAN          Arg(const VAR* var) const;

virtual CARDINAL NumProcArg() const;
virtual PROC*    ProcArg(CARDINAL index) const;
BOOLEAN          Arg(const PROC* proc) const;

virtual BOOLEAN  ReplaceVar(VAR* oldvar, VAR* newvar);
virtual BOOLEAN  ExchangeVar(VAR* var1, VAR* var2);
BOOLEAN          ReplaceProc(PROC* oldproc, PROC* newproc);

JOBSTATUS        Load(istream& in,
                      const VARLIST* varlist,
                      const PROCLIST* proclist);
JOBSTATUS        Save(ostream& out);
```

```
        JOBSTATUS           Write(ostream& out) const;

    protected:

        void                CalcRates(REAL* rates);

        CARDINAL            numactvar;
        VAR**               actvar;
        CARDINAL            numactproc;
        PROC**              actproc;
        FORMVAR*            inflow;
        CARDINAL            numinput;
        VAR**               inputvar;
        FORMVAR**           inputflux;
        CARDINAL            numinit;
        VAR**               initvar;
        FORMVAR**           initval;
};
```

The class COMP is derived from the class SYMBOL inheriting the member functions of the classes SYMBOL (7.12), NODE (7.8) and FILEIO (7.11). This gives objects of this class a symbol and a description, makes it possible to insert them into a list, and provides functions for input and output to a file.

The constructor COMP() without arguments initializes the compartment with default data; the alternative copy constructors COMP(const COMP& comp) and COMP(const COMP* comp) initialize the compartment with with data copied from the compartment referenced as the argument. As described for the copy constructors of the class NODE (7.8), which are automatically called by these constructors, the addresses of the neighboring nodes are not copied, so that the new compartment is not member of a list. The destructor ~COMP() frees memory allocated by the compartment.

The member functions Prev() and Next() only return the results of the inherited member functions Prev() and Next() of the class SYMBOL (7.12) (and therefore of the class NODE (7.8)) with a type conversion to COMP*.

The virtual member function Type() (returning the compartment type) is set to zero to indicate that the implementation will be given in derived classes. This makes it impossible to realize an object of type COMP. It is only possible to realize objects of derived classes which implement all member functions which are set to zero in a base class.

The member functions from NumActVar() to DeleteInitCond(VAR* var) are used to specify and change active variables and processes, external input into the compartment and initial conditions. These functions are common to all compartments.

The most important functions to be implemented for new compartment types are the virtual functions InitCond(...), Delta(...) and GridValue(...), which are set to zero in the base class COMP. The universal interface of these functions declared in the base class COMP guarantees the extensibility of the program to future compartment types. InitCond(...) calculates initial conditions, Delta(...) defines the differential-algebraic system of space discretized differential equations by returning the array G corresponding to the evaluation of the equation (6.1), and GridValue(...) returns the calculated result of a given variable at a given grid

point. All these functions use the real array `Y` corresponding to the array **y** in equation (6.1) to describe the current state of the system.

The virtual member functions from `NumAdvInConn()` to `DiffC(...)` are used for communication with links. Calls to these functions allow the links to calculate hydraulic coupling of water levels and water and substance fluxes. The virtual declaration of these functions in the base class `COMP` of compartments makes it possible to link connections of compartments of different type. Most of these functions use the state vector `Y` as an argument. The function `AdvInExZ()` decides, if there is hydraulic coupling of water levels at the specified advective input connection, and `AdvInZ(...)` returns the current water level elevation if this is the case. `AdvOutQ(...)` and `AdvOutJ(...)` return the water discharge and the mass flux of the specified variable at the specified advective output connection. `DiffC(...)` returns the concentrations of the specified variable at the interface to a diffusive link at the specified diffusive connection. These functions require the state vector `Y` for calculating their return values.

The virtual member functions from `AllowedVar(...)` to `AllowedReplaceProc(...)` are used to check legality of a planned addition or replacement of a variable or process. Such functions are necessary because not all program variables are meaningful in all compartments. The virtual member functions from `NumVarArg()` to `Arg(const PROC* proc)` are used to check dependences on variables and processes. The member functions `ReplaceVar(...)`, `ExchangeVar(...)` and `ReplaceProc(...)` are used to update changes made in the systems of variables and processes, respectively.

The member functions `Load(...)`, `Save(...)` and `Write(...)` are used for loading from and saving to system files and for writing the system to a print file. All these three functions will be called by the more specific functions of the derived classes for different variable types. File input and output operations performed by these functions use the functions provided for this purpose by the base class `FILEIO`.

The protected member function `CalcRates(REAL* rates)` calculates the transformation rates of the active processes within the compartment. This function uses the function `SumRates(...)` of the class `PROC` for the active processes (cf. program fragment (7.16)). The member function `CalcRates(REAL* rates)` can be declared as a protected member function because it is only used within the member function `Delta(...)` of the derived classes wihich is used for the formulation of the differential and algebraic equations of the compartment.

The protected data entries form `numactvar` to `initval` are used to store data for active variables and processes, and input and initial conditions common to all compartment types.

The classes representing the different compartment types described in section 4.3 are derived from the base class `COMP` given (partially) in the program fragment (7.18). Fig. 7.8 gives an overview of the derivation hierarchy of compartments. The class `MIXCOMP` represents a mixed reactor compartment. Besides the implementation of the virtual member functions of `COMP` (with an advective input connection,

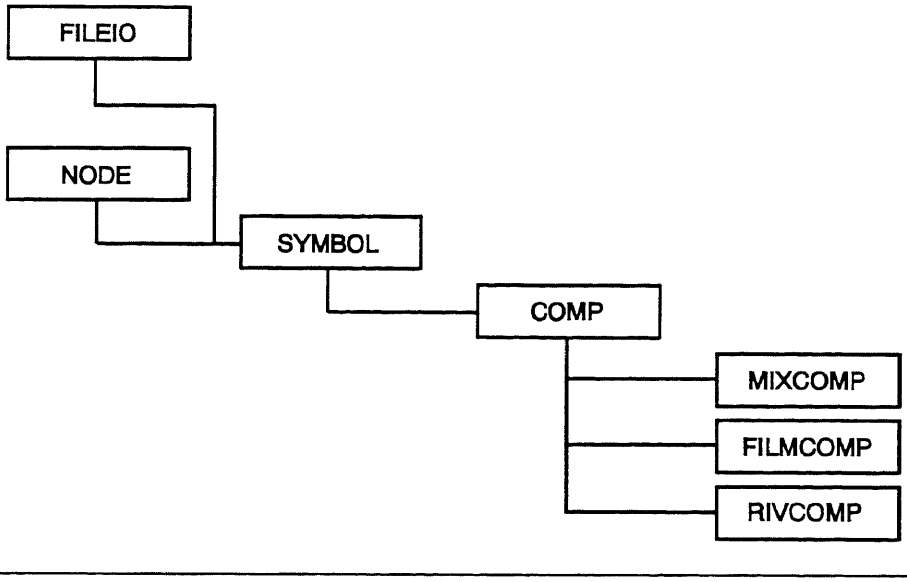


Fig. 7.8: Derivation hierarchy of the classes implementing compartments.

an advective output connection and a diffusive connection), variables for selecting the reactor type (constant or variable volume) and for specifying the volume and the outflow are provided. A biofilm reactor compartment is represented by the class `FILMCOMP`. This compartment has an advective input connection, an advective output connection and two diffusive connections (to the biofilm base and to the bulk fluid volume). This compartment needs the following data entries: Properties of particulate (density, attachment and detachment coefficients, boundary layer resistance and biofilm matrix diffusivity) and dissolved (boundary layer resistance and diffusivity in the pore water of the biofilm) variables, reactor type (confined or unconfined), biofilm matrix type (rigid or diffusive), detachment type (individual or global) and velocity, biofilm area as a function of the distance from the substratum, growth rate of free volume between the particles, number of grid points for spatial discretization and resolution of the discretization method (low or high). A river section compartment is represented by the class `RIVCOMP`. It has an advective input connection at the upstream end and an advective output connection at the downstream end. It needs data for start and end coordinates of the river section, the cross sectional area, perimeter and surface width of the river bed as a function of distance along the river and of water level elevation, empirical expressions for friction slope and for longitudinal dispersion, the type of hydraulic equations to be solved (kinematic or diffusive), the number of grid points for spatial discretization and the resolution of the discretization method (low or high).

All compartments building the system of compartments are collected in a list of the type `COMPLIST`. Fig. 7.9 shows the derivation hierarchy of this class which is analogous to the list of variables shown in Fig. 7.5. The insertion and removal functions of this class are made private, so that they can only be used by the member functions of the class `COMPSYS`, which represents the system of

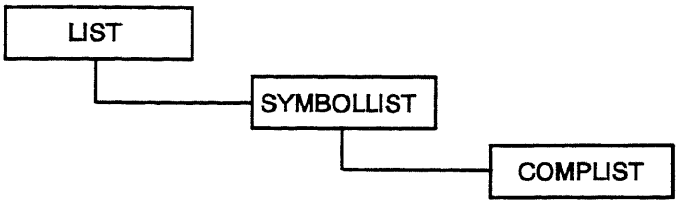


Fig. 7.9: Derivation hierarchy of the class COMPLIST.

compartments and which is a friend of the class COMPLIST. The class COMPSYS consists of a COMPLIST and of member functions for adding, replacing and deleting compartments, for raplacing processes, for replacing and exchanging variables, and for defining the complete differential-algebraic system of equations for all compartments:

```
class COMPSYS : public FILEIO
{
   public:

      COMPSYS();
      ~COMPSYS();

      COMP*      Get(const char* sym) const;

      BOOLEAN    Add(COMP* comp);
      BOOLEAN    Replace(COMP* oldcomp, COMP* newcomp);
      BOOLEAN    Delete(COMP* comp);

      BOOLEAN    AllowedReplaceVar(const VAR* oldvar,
                                   const VAR* newvar) const;
      BOOLEAN    AllowedExchangeVar(const VAR* var1,
                                    const VAR* var2) const;
      BOOLEAN    AllowedReplaceProc(const PROC* oldproc,
                                    const PROC* newproc) const;

      BOOLEAN    Arg(const VAR* var);
      BOOLEAN    Arg(const PROC* proc);

      void       ReplaceVar(VAR* oldvar, VAR* newvar);
      void       ExchangeVar(VAR* var1, VAR* var2);
      void       ReplaceProc(PROC* oldproc, PROC* newproc);

      COMPLIST*  Complist();

      BOOLEAN    InitCond(REAL* Y);
      BOOLEAN    Delta(const REAL* Y, const REAL* YT, REAL* G);
      BOOLEAN    GridValue(CARDINAL calcnum, REAL t,
                           const REAL* Y, COMP* comp,
                           CARDINAL i, VAR* var, REAL& value);

      JOBSTATUS  Load(istream& in);
      JOBSTATUS  Save(ostream& out);
      JOBSTATUS  Write(ostream& out) const;

   private:

      COMPLIST   complist;
};
```

(7.19)

The member function `Get(const char* sym)` returns a pointer to the compartment with the symbol given as the argument. The member functions `Add(COMP* comp)`, `Replace(COMP* oldcomp, COMP* newcomp)` and `Delete(COMP* comp)` are used for editing the system of compartments. The member functions `AllowedReplaceVar(...)`, `AllowedExchangeVar(...)` and `AllowedReplaceProc(...)` test the legality of planned edits of the systems of variables and of processes. The member functions `Arg(const VAR* var)` and `Arg(const PROC* proc)` are used to check if one of the compartments depends on a variable or process given as argument. Such checks are necessary for maintaining the consistency of the system. The member functions `ReplaceVar(...)`, `ExchangeVar(...)` and `ReplaceProc(...)` are necessary to update changes made in the systems of variables and processes, respectively. These functions reflect the dependence of the system of compartments on the systems of variables and of processes shown in Fig. 4.5. The strategy used for editing the system and for maintaining its consistency is discussed in more detail in section 7.4. The member functions `InitCond(...)`, `Delta(...)` and `GridValue(...)` combine the functions of the individual compartments with the same names (described in (7.18)) to global functions defining the whole composite system. Finally, the member functions `Load(...)`, `Save(...)` and `Write(...)` are used for loading and saving the system of compartments and for writing it to a print file. Each of these functions mainly calls the functions with the same name of all compartments in the list (cf. program fragment (7.18))

7.3.4 System of Links

As described in section 4.4, links are used to connect the compartments to the desired spatial configuration of the aquatic system to be modelled. ***The base class for links serves as a common interface for consistency checks and for updating changes made in the subsystems of variables and compartments.*** Since links are identified by their names and they have to be inserted in a list of links, their base class `LINK` is derived from the class `SYMBOL` (7.12) described in section 7.2. Program fragment (7.20) shows the main elements of the base class for links:

```
class LINK : public SYMBOL
{
   public:

      LINK();
      LINK(const LINK& link);
      LINK(const LINK* link);
      ~LINK();

      LINK*             Prev() const;
      LINK*             Next() const;

      virtual LINKTYPE  Type() const = 0;

      virtual BOOLEAN   AllowedReplaceVar(const VAR* oldvar,
                                          const VAR* newvar) const;
      virtual BOOLEAN   AllowedExchangeVar(const VAR* var1,
                                           const VAR* var2) const;
      virtual BOOLEAN   AllowedReplaceComp(const COMP* oldcomp,
                                           const COMP* newcomp) const;

      virtual CARDINAL  NumVarArg() const;                          (7.20)
      virtual VAR*      VarArg(CARDINAL index) const;
      virtual BOOLEAN   Arg(const VAR* var) const;

      virtual CARDINAL  NumCompArg() const;
      virtual COMP*     CompArg(CARDINAL index) const;
      virtual BOOLEAN   Arg(const COMP* comp) const;

      virtual BOOLEAN   ReplaceVar(VAR* oldvar, VAR* newvar);
      virtual BOOLEAN   ExchangeVar(VAR* var1, VAR* var2);
      virtual BOOLEAN   ReplaceComp(COMP* oldcomp, COMP* newcomp);

      JOBSTATUS         Load(istream& in);
      JOBSTATUS         Save(ostream& out);
      JOBSTATUS         Write(ostream& out) const;
};
```

The class `LINK` is derived from the class `SYMBOL` inheriting the member functions of the classes `SYMBOL` (7.12), `NODE` (7.8) and `FILEIO` (7.11). This gives objects of this class a symbol and a description, makes it possible to insert them into a list, and provides functions for input and output to a file.

The constructor `LINK()` without arguments initializes the link with default data; the alternative copy constructors `LINK(const LINK& link)` and `LINK(const LINK* link)` initialize the link with data copied from the link referenced as the argument. As described for the copy constructors of the class `NODE` (7.8), which are automatically called by these constructors, the addresses of the neighboring nodes

are not copied, so that the new link is not member of a list. The destructor `~LINK()` frees memory allocated by the link.

The member functions `Prev()` and `Next()` only return the results of the inherited member functions `Prev()` and `Next()` of the class `SYMBOL` (7.12) (and therefore of the class `NODE` (7.8)) with a type conversion to `LINK*`.

The virtual member function `Type()` (returning the link type) is set to zero to indicate that the implementation will be given in derived classes. This makes it impossible to realize an object of type `LINK`. It is only possible to realize objects of derived classes which implement all member functions which are set to zero in a base class.

The virtual member functions from `AllowedReplaceVar(...)` to `Arg(const COMP* comp)` are used for consistency checks of planned operations and for dependence checks. The virtual member functions `ReplaceVar(...)`, `ExchangeVar(...)` and `ReplaceComp(...)` are used to update changes made in the systems of variables and of compartments.

The member function `Load(...)` is used to read general data for links from the system file. The member functions `Save(...)` and `Write(...)` implement virtual member functions of the class `FILEIO` (7.11). All these three functions will be called by the more specific functions of the derived classes for different variable types. File input and output operations performed by these functions use the functions provided for this purpose by the base class `FILEIO`.

The two classes `ADVLINK` and `DIFFLINK` representing advective and diffusive links as described in section 4.4 are derived from the base class `LINK` as shown in Fig. 7.10.

The class `ADVLINK` needs an additional class `BIF` implementing bifurcations. This class, which is not documented here, contains information on the compartment to

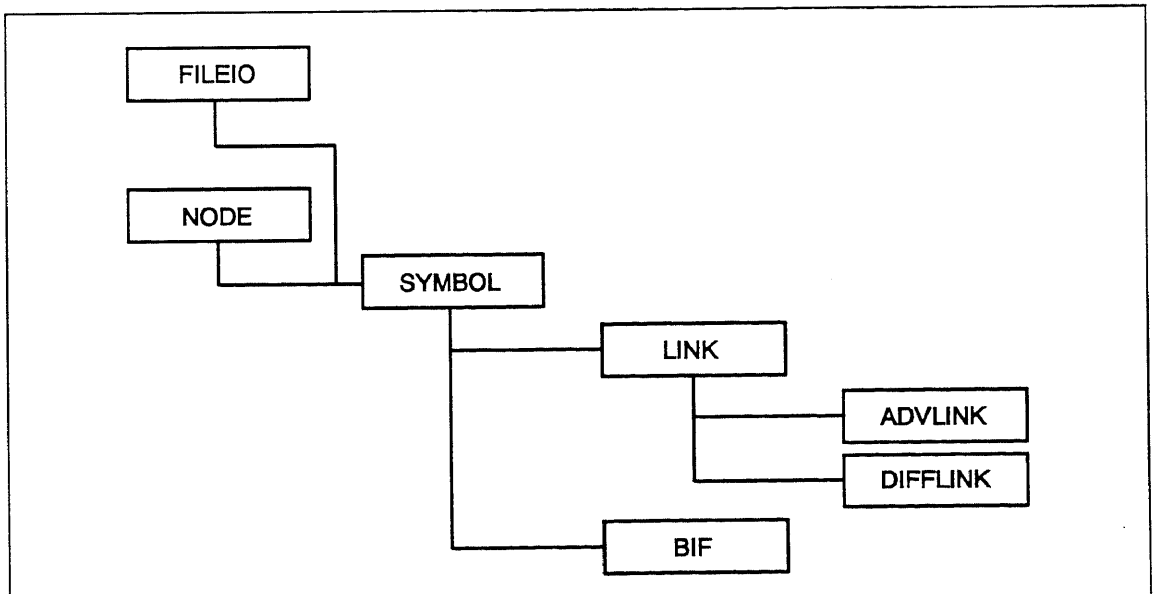


Fig. 7.10: Derivation hierarchy of the classes implementing links.

which the bifurcation is connected and on water flow and substance fluxes through the bifurcation. The class BIF is derived from the class SYMBOL similarly to the classes VAR (7.13), PROC (7.16), COMP (7.18) and LINK (7.20). As described for these classes, also for the class BIF a class BIFLIST derived from the class SYMBOLLIST is defined to maintain the bifurcations of an advective link. The main elements of the class ADVLINK are given in program fragment (7.21):

```
class ADVLINK : public LINK
{
   public:

      ADVLINK();
      ADVLINK(const ADVLINK& link);
      ADVLINK(const LINK* link);
      ~ADVLINK();

      LINKTYPE          Type() const;

      BOOLEAN           SetCompIn(COMP* comp, CARDINAL conn);
      COMP*             GetCompIn() const;
      CARDINAL          GetConnIn() const;
      BOOLEAN           SetCompOut(COMP* comp, CARDINAL conn);
      COMP*             GetCompOut() const;
      CARDINAL          GetConnOut() const;
      BIF*              GetBif(const char* sym) const;
      BOOLEAN           AddBif(BIF* bif);
      BOOLEAN           ReplaceBif(BIF* oldbif, BIF* newbif);
      BOOLEAN           DeleteBif(BIF* bif);
      const BIFLIST*    Biflist() const;

      BOOLEAN           AdvInExZ() const;
      REAL              AdvInZ(const REAL* Y) const;
      REAL              AdvOutQ(const REAL* Y, CARDINAL index);
      REAL              AdvOutJ(const REAL* Y, CARDINAL index,
                                const VAR* var);                    (7.21)
      BOOLEAN           AllowedReplaceVar(const VAR* oldvar,
                                          const VAR* newvar) const;
      BOOLEAN           AllowedExchangeVar(const VAR* var1,
                                           const VAR* var2) const;
      BOOLEAN           AllowedReplaceComp(const COMP* oldcomp,
                                           const COMP* newcomp) const;

      CARDINAL          NumVarArg() const;
      VAR*              VarArg(CARDINAL index) const;
      BOOLEAN           Arg(const VAR* var) const;

      CARDINAL          NumCompArg() const;
      COMP*             CompArg(CARDINAL index) const;
      BOOLEAN           Arg(const COMP* comp) const;

      BOOLEAN           ReplaceVar(VAR* oldvar, VAR* newvar);
      BOOLEAN           ExchangeVar(VAR* var1, VAR* var2);
      BOOLEAN           ReplaceComp(COMP* oldcomp, COMP* newcomp);

      JOBSTATUS         Load(istream& in);
      JOBSTATUS         Save(ostream& out);
      JOBSTATUS         Write(ostream& out) const;

   private:

      COMP*             compin;
      CARDINAL          connin;
      COMP*             compout;
      CARDINAL          connout;
```

```
        BIFLIST         biflist;
};
```

Besides the constructors and the implementation of virtual member functions of the base class `LINK`, this class contains two important groups of member functions:

The member functions from `SetCompIn(...)` to `Biflist()` are used for defining the link and for giving information on link definition. The compartments and the connections of the compartments of the inflow and the default outflow can be specified. Furthermore, bifurcations of type `BIF` (see above) can be added, replaced and deleted by the member functions within this group.

The other group of member functions from `AdvInExZ()` to `AdvOutJ(...)` serves for communication with the compartments. A compartment the inflow of which is connected to the outflow of an advective link determines by calls to `AdvOutQ(...)` and `AdvOutJ(...)` the water and substance fluxes at its inflow. These functions themselves determine these quantities by calls to functions with the same name of the compartments feeding the link (compare with the definition of the class `COMP` given in program fragment (7.18)). These functions require the state vector `Y` as an argument for calculating these quantities.

The class `DIFFLINK` has a similar structure as the class `ADVLINK`. It is somewhat simpler due to the absence of bifurcations. The program fragment (7.22) shows the main elements of this class:

```
class DIFFLINK : public LINK
{
   public:

      DIFFLINK();
      DIFFLINK(const DIFFLINK& link);
      DIFFLINK(const LINK* link);
      ~DIFFLINK();

      LINKTYPE          Type() const;

      BOOLEAN           SetComp1(COMP* comp, CARDINAL conn);
      COMP*             GetComp1() const;
      CARDINAL          GetConn1() const;
      BOOLEAN           SetComp2(COMP* comp, CARDINAL conn);
      COMP*             GetComp2() const;
      CARDINAL          GetConn2() const;

      CARDINAL          NumExch() const;
      VAR*              ExchVar(CARDINAL index) const;
      const FORMVAR*    ExchCoeff(CARDINAL index) const;
      const FORMVAR*    ConvFact(CARDINAL index) const;
      BOOLEAN           AddExch(VAR* var, const VARLIST* varlist,
                                const char* coeff, const char* fact);
      BOOLEAN           ReplaceExch(VAR* var, const VARLIST* varlist,
                                const char* coeff, const char* fact);
      BOOLEAN           DeleteExch(VAR* var);

      REAL              DiffJ(const REAL* Y, CARDINAL index,
                              const VAR* var);
                                                                  (7.22)
      BOOLEAN           AllowedReplaceVar(const VAR* oldvar,
                                          const VAR* newvar) const;
      BOOLEAN           AllowedExchangeVar(const VAR* var1,
                                           const VAR* var2) const;
```

```
        BOOLEAN         AllowedReplaceComp(const COMP* oldcomp,
                                           const COMP* newcomp) const;

        CARDINAL        NumVarArg() const;
        VAR*            VarArg(CARDINAL index) const;
        BOOLEAN         Arg(const VAR* var) const;

        CARDINAL        NumCompArg() const;
        COMP*           CompArg(CARDINAL index) const;
        BOOLEAN         Arg(const COMP* comp) const;

        BOOLEAN         ReplaceVar(VAR* oldvar, VAR* newvar);
        BOOLEAN         ExchangeVar(VAR* var1, VAR* var2);
        BOOLEAN         ReplaceComp(COMP* oldcomp, COMP* newcomp);

        JOBSTATUS       Load(istream& in);
        JOBSTATUS       Save(ostream& out);
        JOBSTATUS       Write(ostream& out) const;

    private:

        COMP*           comp1;
        CARDINAL        conn1;
        COMP*           comp2;
        CARDINAL        conn2;

        REAL            area;
        CARDINAL        numtrans;
        VAR**           exchvar;
        FORMVAR**       exchcoeff;
        FORMVAR**       convfact;
};
```

In addition to the constructors and destructors and the implementation of the virtual member functions of the class `LINK`, this class contains member functions for editing the link by setting the compartments and connections to be linked, the area of the membrane and the exchange coefficients and conversion factors of various substances. The member function `DiffJ(...)`, which uses the state vector `Y` of the system as the argument, is called by the compartments connected to the link for determining the flux of a given substance represented by a variable. This function itself calls the function `DiffC(...)` of the compartments connected to the link for determining the concentrations at both sides of the boundary layer or membrane to be able to calculate the flux using the exchange coefficient and the conversion factor.

All links building the system of links are collected in a list of links of the type `LINKLIST`. Fig. 7.11 shows the derivation hierarchy of this class which is analogous to the list of variables shown in Fig. 7.5. The insertion and removal functions of this class are made private, so that they can only be used by the member functions of the class `LINKSYS`, which represents the system of links and which is a friend of the class `LINKLIST`. The class `LINKSYS` consists of a `LINKLIST` and of member functions for adding, replacing and deleting links, for replacing compartments and for replacing and exchanging variables:

```
class LINKSYS : public FILEIO
{
   public:

      LINKSYS();
      ~LINKSYS();
```

```
        LINK*       Get(const char* sym) const;

        BOOLEAN     Add(LINK* link);
        BOOLEAN     Replace(LINK* oldlink, LINK* newlink);
        BOOLEAN     Delete(LINK* link);

        BOOLEAN     AllowedReplaceVar(const VAR* oldvar,
                                      const VAR* newvar) const;
        BOOLEAN     AllowedExchangeVar(const VAR* var1,
                                       const VAR* var2) const;
        BOOLEAN     AllowedReplaceComp(const COMP* oldcomp,
                                       const COMP* newcomp) const;
                                                                    (7.23)
        BOOLEAN     Arg(const VAR* var);
        BOOLEAN     Arg(const COMP* comp);

        void        ReplaceVar(VAR* oldvar, VAR* newvar);
        void        ExchangeVar(VAR* var1, VAR* var2);
        void        ReplaceComp(COMP* oldcomp, COMP* newcomp);

        LINKLIST*   Linklist();

        JOBSTATUS   Load(istream& in);
        JOBSTATUS   Save(ostream& out);
        JOBSTATUS   Write(ostream& out) const;

    private:

        LINKLIST    linklist;
};
```

The member function `Get(const char* sym)` returns a pointer to the link with the symbol given as the argument. The member functions `Add(LINK* link)`, `Replace(LINK* oldlink, LINK* newlink)` and `Delete(LINK* link)` are used for editing the system of links. The member functions `AllowedReplaceVar(...)`, `AllowedExchangeVar(...)` and `AllowedReplaceComp(...)` test the legality of planned edits of the systems of variables and compartments. The member functions `Arg(const VAR* var)` and `Arg(const COMP* comp)` are used to check if one of the links depends on the variable or compartment given as the argument. Such checks are necessary for maintaining the consistency of the system. The member functions `ReplaceVar(...)`, `ExchangeVar(...)` and `ReplaceComp(...)` are necessary to update changes made in the systems of variables and compartments, respectively. These functions reflect the dependence of the system of links on the systems of variables and of compartments shown in Fig. 4.5. The strategy used for editing the system and for maintaining its consistency is discussed in more detail in section 7.4. Finally, the member functions `Load(...)`,

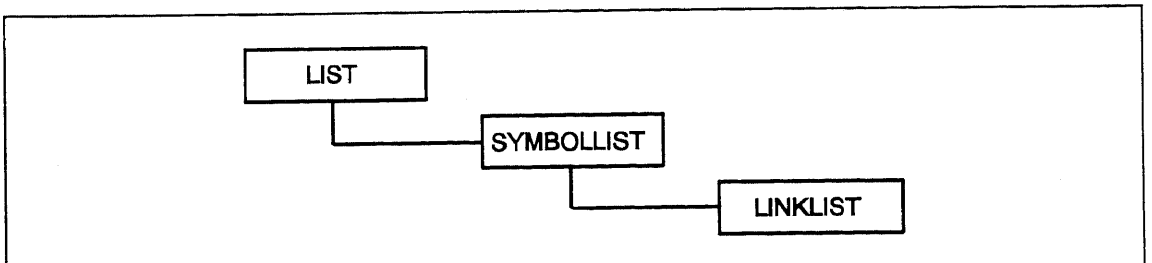


Fig. 7.11: Derivation hierarchy of the class `LINKLIST`.

`Save(...)` and `Write(...)` are used for loading and saving the system of links and for writing it to a print file. Each of these functions mainly calls the functions with the same name of all links in the list (cf. program fragment (7.20))

7.3.5 Integration to an Extensible Simulation Program

In the preceeding subsections, the main implementation concepts of the four model subsystems of variables, processes, compartments and links are discussed. These subsystems are sufficient to define the model. The program, however, needs additional information for the definition of sensitivity analyses, of parameter estimations and of plots of results. These definitions are stored in objects of the classes SENS, FIT and PLOT, respectively. These classes are implemented in analogy to the classes VAR (7.13), PROC (7.16), COMP (7.18) and LINK (7.20), with the exception, that there are no derived classes of the classes SENS, FIT and PLOT (such derived classes are not necessary, because there is no division of sub-types for sensitivity analyses, parameter estimations and plots). The definitions of sensitivity analyses, parameter estimations and plots are collected in subsystems implemented as classes SENSSYS, FITSYS and PLOTSYS in analogy to the model subsystems VARSYS (7.15), PROCSYS (7.17), COMPSYS (7.19) and LINKSYS (7.23). Furthermore, the class STATESYS implements storage of calculated states of the program. ***The integration of all subsystems to the complete system is done in a class AQUASYS, which is the main interface to the user interface program layer.*** The main elements of this class are given in the following program fragment (7.24):

```
class AQUASYS : public FILEIO
{
   public:

      AQUASYS();
      ~AQUASYS();

      BOOLEAN     ExistVar(const VAR* var) const;
      BOOLEAN     ExistVar(const char* sym) const;
      VAR*        GetVar(const char* sym) const;
      BOOLEAN     AddVar(VAR* var);
      BOOLEAN     ReplaceVar(VAR* oldvar, VAR* newvar);
      BOOLEAN     ExchangeVar(VAR* var1, VAR* var2);
      BOOLEAN     DeleteVar(VAR* var);
      VARLIST*    Varlist();

      BOOLEAN     ExistProc(const PROC* proc) const;
      BOOLEAN     ExistProc(const char* sym) const;
      PROC*       GetProc(const char* sym) const;
      BOOLEAN     AddProc(PROC* proc);
      BOOLEAN     ReplaceProc(PROC* oldproc, PROC* newproc);
      BOOLEAN     DeleteProc(PROC* proc);
      PROCLIST*   Proclist();

      BOOLEAN     ExistComp(const COMP* comp) const;
      BOOLEAN     ExistComp(const char* sym) const;
      COMP*       GetComp(const char* sym) const;
      BOOLEAN     AddComp(COMP* comp);
      BOOLEAN     ReplaceComp(COMP* oldcomp, COMP* newcomp);
      BOOLEAN     DeleteComp(COMP* comp);
      COMPLIST*   Complist();

      BOOLEAN     ExistLink(const LINK* link) const;
      BOOLEAN     ExistLink(const char* sym) const;
      LINK*       GetLink(const char* sym) const;
      BOOLEAN     AddLink(LINK* link);
      BOOLEAN     ReplaceLink(LINK* oldlink, LINK* newlink);
      BOOLEAN     DeleteLink(LINK* link);
      LINKLIST*   Linklist();
```

```
    BOOLEAN     ExistSens(const SENS* sens) const;
    BOOLEAN     ExistSens(const char* sym) const;
    SENS*       GetSens(const char* sym) const;
    BOOLEAN     AddSens(SENS* sens);
    BOOLEAN     ReplaceSens(SENS* oldsens, SENS* newsens);
    BOOLEAN     DeleteSens(SENS* sens);
    SENSLIST*   Senslist();

    BOOLEAN     ExistFit(const FIT* fit) const;
    BOOLEAN     ExistFit(const char* sym) const;
    FIT*        GetFit(const char* sym) const;
    BOOLEAN     AddFit(FIT* fit);
    BOOLEAN     ReplaceFit(FIT* oldfit, FIT* newfit);
    BOOLEAN     DeleteFit(FIT* fit);
    FITLIST*    Fitlist();

    BOOLEAN     ExistPlot(const PLOT* plot) const;
    BOOLEAN     ExistPlot(const char* sym) const;
    PLOT*       GetPlot(const char* sym) const;
    BOOLEAN     AddPlot(PLOT* plot);
    BOOLEAN     ReplacePlot(PLOT* oldplot, PLOT* newplot);
    BOOLEAN     DeletePlot(PLOT* plot);
    PLOTLIST*   Plotlist();                                          (7.24)

    void        CalcNum(CARDINAL num);
    CARDINAL    CalcNum() const;
    void        InitTime(REAL time);
    REAL        InitTime() const;
    void        InitSteady(BOOLEAN steady);
    BOOLEAN     InitSteady() const;
    void        StepSize(REAL size);
    REAL        StepSize() const;
    void        NumSteps(CARDINAL num);
    CARDINAL    NumSteps() const;
    FITMETHOD   FitMeth() const;
    void        FitMeth(FITMETHOD meth);
    void        FitIter(CARDINAL maxiter);
    CARDINAL    FitIter() const;

    JOBSTATUS   InitCalc();
    JOBSTATUS   Calculate();
    JOBSTATUS   SensAnal(ostream& out);
    JOBSTATUS   Fit(ostream& out);

    JOBSTATUS   GetAllCurvesData(PLOT* plot, PLOTDATA* plotdata);
    JOBSTATUS   ListResults(ostream& out, PLOT* plot);
    JOBSTATUS   PlotResults(ostream& out, PLOT* plot);

    LOADRESULT  Load(istream& in);
    JOBSTATUS   Save(ostream& out);
    JOBSTATUS   Write(ostream& out);

private:

    VARSYS      varsys;
    PROCSYS     procsys;
    COMPSYS     compsys;
    LINKSYS     linksys;
    SENSSYS     senssys;
    FITSYS      fitsys;
    PLOTSYS     plotsys
    STATESYS    statesys;

    CARDINAL    calcnum;
    REAL        inittime;
    BOOLEAN     initsteady;
```

```
        REAL        stepsize;
        CARDINAL    numsteps;
        FITMETHOD   fitmeth;
        CARDINAL    fititer;
};
```

The group of member functions from `ExistVar(...)` to `Plotlist()` are used to edit the subsystems of variables, processes, compartments, links, sensitivity analysis definitions, parameter estimation definitions, and plot definitions stored in the variables `varsys`, `procsys`, `compsys`, `linksys`, `senssys`, `fitsys` and `plotsys`. Due to the functions provided by the classes `VARSYS` (7.15), `PROCSYS` (7.17), `COMPSYS` (7.19), `LINKSYS` (7.23), `SENSSYS`, `FITSYS` and `PLOTSYS`, the implementation of these functions is very simple: First, the legality of the proposed change is checked in all subsystems. In case of a positive answer, the change is performed in the corresponding subsystem and updated in the other subsystems. As an example, the function `ReplaceVar(...)` of `AQUASYS` first checks by a call to the functions `AllowedReplaceVar(...)` of the objects `compsys` and `linksys` if the replacement is allowed, and then calls the functions `ReplaceVar(...)` of `varsys`, `procsys`, `compsys`, `linksys`, `senssys`, `fitsys` and `plotsys` to perform the replacement. The function `DeleteVar(VAR* var)` checks by a call to `Arg(var)` of the subsystems, if the variable is an argument of another variable or of any of the subsystems, and rejects deletion if this is the case.

The member functions from `CalcNum(CARDINAL num)` to `FitIter()` serve to store and recall parameter values for data analysis calculations of the following group of member functions.

The member functions from `InitCalc()` to `Fit()` perform the calculations requested by the user. `InitCalc()` makes the initial state defined by the user consistent as described in section 6.1. `Calculate()` performs a simulation, `SensAnal()` a sensitivity analysis and `Fit(ostream& out)` a parameter estimation. As described in section 6.2, for the numerical solution of the system of differential-algebraic equations, the Fortran program DASSL due to Petzold (1983) is used. To facilitate portability of the program, and to eliminate compiler specific directives, this program was converted to C++ by an automatic conversion program (Cobalt Blue, 1992). The other numerical algorithms described in chapter 6 were directly implemented in C++ (non object-oriented).

The next group of member functions are used to process results. The function `GetAllCurvesData(...)` collects data required for the plot with the plot definition given as the argument and returns this data in a class of type `PLOTDATA`, which is not described in this report. The two functions `ListResults(...)` and `PlotResults(...)` write the data corresponding to a plot definition as a listing of numbers (for external postprocessing) or as a plot in PostScript format (for transfer to a printer) to a file.

The last group of member functions `Load(...)`, `Save(...)` and `Write(...)` is used for loading and saving the whole system and for writing it to a print file. The functions of this group mainly call the corresponding functions of the subsystems.

The return type `LOADRESULT` makes it possible to differentiate between successful load and error due to wrong file type, wrong file version and read error.

7.4 Concepts for Editing Models

Two main problems of editing a model are distinguished in this section. The first problem is to not destroy consistency of the system by editing operations. The second problem is to allow the user undoing elementary edits by pressing the "Cancel" button of a dialog box without too much programming effort. These two problems are treated separately in the following two subsections.

7.4.1 Solving Consistency Problems

In the preceeding section, functions are discussed which make construction of a model possible (member functions `AddVar(...)`, `AddProc(...)`, `AddComp(...)` and `AddLink(...)` of the class `AQUASYS` (7.24)). Additionally, simple editing functions are provided for deleting (member functions `DeleteVar(...)`, `DeleteProc(...)`, `DeleteComp(...)` and `DeleteLink(...)` of the class `AQUASYS` (7.24)) and replacing (member functions `ReplaceVar(...)`, `ReplaceProc(...)`, `ReplaceComp(...)` and `ReplaceLink(...)` of the class `AQUASYS` (7.24)) variables, processes, compartments and links. All these functions check maintenance of consistency before the operation is performed. ***There are three levels of inconsistency that may occur during editing a model***:

1 Illegal entries in data fields of variables, processes, compartments or links (e.g. illegal variable names containing spaces or special characters (which are not allowed to make it possible to parse algebraic expressions), values outside their legal ranges, etc.).

2 Uniqueness of symbols within subsystems and circular references in formula variables (e.g. two variables with the same name cannot be distinguished in parsing an algebraic expression, a variable depending on itself cannot be evaluated, etc.).

3 Mutual consistency of subsystems such as dependences on illegal variables (e.g. a variable used within a mixed reactor compartment is not allowed to depend on the program variable "Space Coordinate X", because this variable has only a sense in a compartment which resolves the x-axis).

All of these consistency problems can relatively easily be checked for the elementary editing operations introduced so far:

1 Illegal entries in objects are already avoided at the class level by using information hiding and by accessing data members of the class only via member functions which perform class-internal consistency checks (e.g. the member

function `Symbol(const char* sym)` of the class `SYMBOL` defined in the program fragments (7.10) and (7.12), which is used to rename objects, rejects illlegal names).

2 Uniqueness of symbols and circular references are checked at the subsystem level by the functions `AddVar(VAR* var)`, `ReplaceVar(...)` and `ExchangeVar(...)` of the class `VARSYS` (7.15) and by similar functions of the classes `PROCSYS` (7.17), `COMPSYS` (7.19) and `LINKSYS` (7.23).

3 Mutual consistency of the subsystems is the most severe consistency problem which can only be solved at the level of the class `AQUASYS` (7.24). This problem is solved by checking legality of an action by a call to the functions of all subsystems provided for this purpose before performing the desired action (e.g. as explained in section 7.3.5, the function `ReplaceVar(...)` of the class `AQUASYS` checks with a call to the functions `AllowedReplaceVar(...)` of the subsystems `compsys` and `linksys` legality of the change and updates all subsystems only in case of legality. Similar checks are performed for the other elementary edit operations).

It would be very difficult to implement similar consistency checks for all possible edit actions. For this reason, the following concept for editing the system, which makes use of the copy constructors described in section 7.3 and of the list concept described in section 7.2, is applied: To save integrity of the subsystems, all manipulations on variables, processes, compartments and links which may violate the consistency of the system (e.g. changing the name or argument of a variable) are locked for objects which are element of the subsystems (this can easily be done using the member function `InList()` of the class `NODE` (7.8) (cf. section 7.2) in the member function which change the data entry). Editing objects of subsystems is performed according the following strategy: ***With the aid of the copy constructor of the object, a copy of the object to be edited is made. Since this copy is not element of a list (cf. section 7.2) all edits can be performed. Then, the original object is replaced by its edited copy with the aid of the elementary editing function described above, which guarantees consistency of the edited system.*** This method makes it also possible to implement easily a duplication operation, by calling the add function instead of the replacement function after the copy of the object has been edited.

7.4.2 Undoing Elementary User Edits

A user of the program should be allowed to press a "Cancel" button also after a multi-stage editing procedure. ***The concept of editing objects by duplication, editing the copy of the original object and replacing the original object by the edited copy, as described in the preceeding subsection, very easily allows such cancelling operations to be implemented. Pressing the "Ok" button of a dialog box leads to a call to the replacement function of the class AQUASYS, whereas pressing the "Cancel" button just leads to destroying the edited copy of the original object.***

7.5 User Interfaces

As mentioned in the introduction, a user-friendly interface very much facilitates learning to use a program. Obviously, it is best to work with the native graphical user interface of a machine. The problem of this choice of a user interface is, that the application programmer interfaces of the window systems of different hard- and software platforms are completely different. Therefore, a program using such an interface loses its portability, what was stated to be a main requirement of the program.

Since it is not clear, if one of the actually used application programmer interfaces will become a standard, it is advisable to separate as far as possible general program tasks from program parts needed for the user interface. ***The general program structure consists of a large, portable core program and a relatively thin user-interface layer, which translates user input to calls of functions provided by the core program***. Such a strategy of program design facilitates adaptation of the program to new platforms. Nevertheless, because of the large number of dialog boxes required by the program, such an adaptation to a new user-interface causes a lot of work.

The following solution of the dilemma between portability and use of a user-friendly graphical interface was chosen: ***Three versions of the user-interface layer were implemented.*** The first version, the ***window interface version***, uses the native graphical user-interface of a machine. This version is most recommendable for editing models, for performing short calculations and for viewing results. To not unnecessarily limit portability, this window interface version was realized using the application programmer interface of the XVT library (Rochkind, 1989-92; Carpenter, 1991; Côté, 1992, Apiki, 1994). This library is available for all important window systems and makes the program portable to any machine for which the developer has purchased the library. The compiled program, linked with the library, remains freely distributable. A second version of the user-interface layer, the ***character interface version***, only uses standard input and output procedures as available for all C++ compilers. This version can be used with simple terminals without graphical capabilities. It is easily transferable to any platform because no external library is required. This makes the character interface version best suited for testing new machines. The third program version is a ***batch version***, which is designed for submitting long calculations as batch jobs (especially sensitivity analyses and parameter estimations with large models need a lot of computation time and therefore are best submitted as batch jobs). The batch version uses a model created with one of the interactive program versions and performs a simulation, a sensitivity analysis or a parameter estimation or results can be listed or plotted. The task to be performed can be specified with the aid of command line arguments of the program. It is not possible to edit a model with the batch version. Due to the absence of the user-interface, this version has the smallest size of the executable program.

To enable graphical output for all program versions, the following strategy was employed: Only the window interface version has the possibility of plotting results directly to the screen. All three program versions can list results to a text file and write plots to a PostScript file. List-files are used for transfering results to external

postprocessing programs. PostScript-files can be transfered to a printer or, if a PostScript interpreter is available, they can also be displayed on the screen.

7.6 Summary of Implementation Concepts

Object-oriented techniques were applied for the implementation of the program to guarantee extensibility, to facilitate robust program design and to shorten development time.

Fig. 7.12 reviews the main derivation hierarchy of classes derived from the classes FILEIO (7.11) or NODE (7.8) described in section 7.2. The classes representing the model subsystems of variables (VARSYS (7.15)), processes (PROCSYS (7.17)), compartments (COMPSYS (7.19)) and links (LINKSYS (7.23)) which are described in section 7.3, as well as the classes for definitions of sensitivity analyses and parameter estimations (SENSSYS and FITSYS), for plot definitions (PLOTSYS) and for storage of calculated states (STATESYS) are derived from the class FILEIO (7.11), which provides functions for loading, saving and printing the elements of these subsystems. The class AQUASYS (7.24), representing the whole system, is also derived from the class FILEIO. The base classes for variables (VAR (7.13)), processes (PROC (7.16)), compartments (COMP (7.18)), links (LINK (7.20)), bifurcations (BIF), and for definitions of sensitivity analyses (SENS), parameter estimations (FIT) and plots (PLOT) are derived from the class SYMBOL (7.12), which handles the identifiers of these objects. The class SYMBOL is again derived from the class FILEIO, but additionally, it is derived from the class NODE (7.8), to make it possible to insert these objects into lists. The different variable types are derived from their base class VAR, the process types from PROC, the compartment types from COMP and the link types from LINK. This derivation hierarchy makes it possible to implement common features in these base classes and to declare similar features as virtual functions in base classes. The following features of variables, processes, compartments, links, bifurcations and definitions of sensitivity analyses, parameter estimations and plots are inherited from implementations in base classes:

all classes:
- list connections (inherited from the class NODE),
- basic functions for file input and output (inherited from the class FILEIO),
- storing and recalling the name (symbol) and description (inherited from the class SYMBOL),

class VAR:
- storing and recalling the unit,
- mechanism of storing previously calculated values,

class COMP:
- maintaining the list of active variables,
- maintaining the list of active processes,
- handling of water inflow and substance input,
- handling of initial conditions,
- calculation of rates of active processes.

The implementation of these features in base classes saves development time and guarantees a robust implementation by reusing tested code.

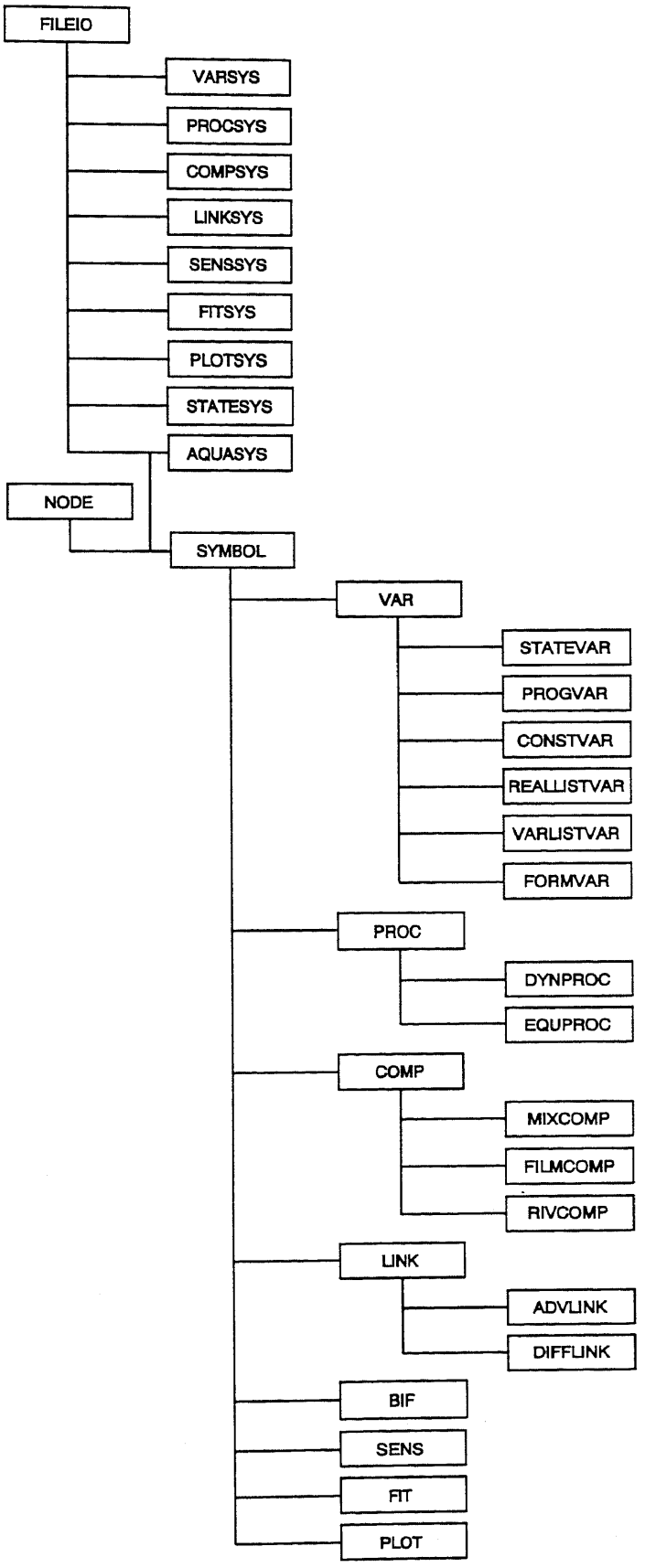


Fig. 7.12: Hierarchy of classes derived from the classes `FILEIO` or `NODE`.

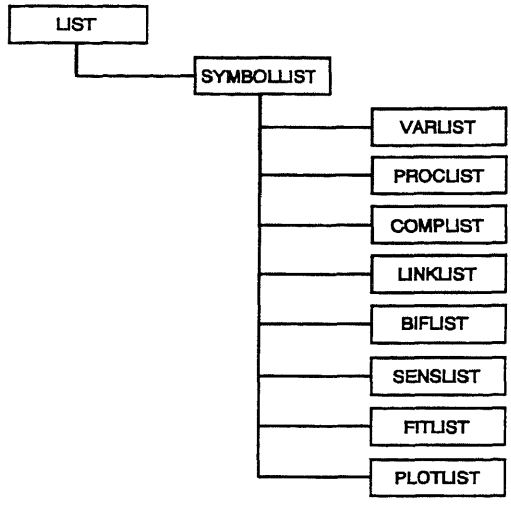


Fig. 7.13: Hierarchy of classes derived from the class `LIST`.

Even more important is the use of polymorphism by virtual declaration of member functions in base classes, which perform similar, but not the same tasks in derived classes. This makes it possible to write the overwhelming part of the program independent of the types of variables, processes, compartments or links and therefore facilitates later extensions of the program. The following important features are virtually declared in base classes:

all classes:
- saving and printing,
- consistency checks and updates,

class `VAR`:
- calculation of the value of the variable,

class `COMP`:
- formulation of the differential-algebraic equations of temporal evolution of the spatially discretized model equations,
- communication with links via connections,
- evaluation of calculated results.

Addition of a new variable or compartment type requires the implementation of all virtual functions (which are quite a lot in the case of a compartment), of type-specific functions for input and output, and of additional dialog boxes in the interface layer for editing type specific data. Due to the object-oriented program design, the overwhelming rest of the program including temporal integration, sensitivity analysis, parameter estimation and listing and plotting of results needs no change at all.

As described in more detail in section 7.4, the consequent implementation of copy constructors, which do not copy the list connections, facilitates editing of the model. Editing of objects is done on local copies of the objects to be edited, created with these constructors. The implementation of this concept requires only a few replacement functions which do consistency checks and facilitates cancelling of edit operations.

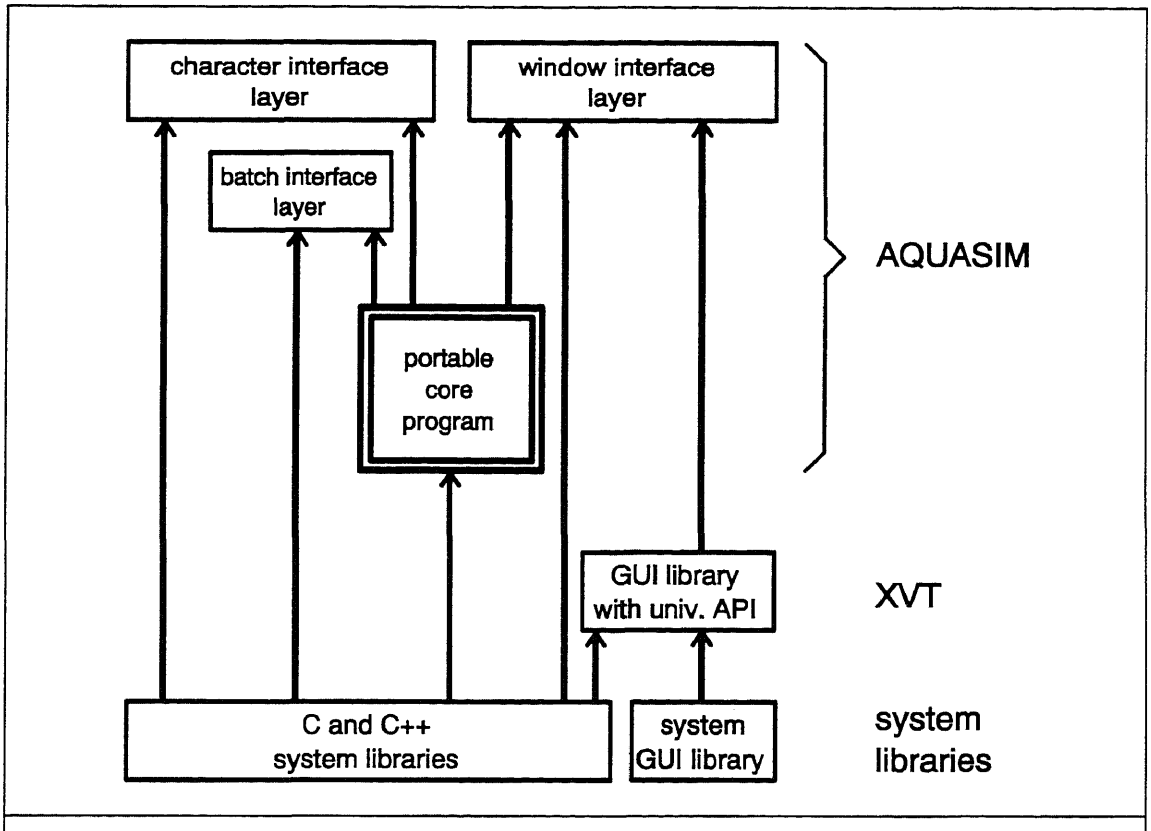


Fig. 7.14: Architecture of the program AQUASIM (arrows indicate dependences).

Besides the derivation hierarchy of classes shown in Fig. 7.12, there is a corresponding derivation hierarchy of list classes, which collect the objects derived from the class `NODE`. As is shown in Fig. 7.13, the root of this derivation hierarchy is the class `LIST` described in program fragment (7.9).

The core program represented by the derivation hierarchies shown in Figs. 7.12 and 7.13 is written in C++ using standard C and C++ libraries. This makes this core program easily portable. The complete program needs an additional relatively thin user-interface layer, which forwards communication between the user and the portable core of the program. As described in more detail in section 7.5, three versions of such a user-interface layer are provided: A window interface layer uses the application programmer interface (API) of the graphical user-interface (GUI) ligrary XVT, which is available on different hard- and software platforms, a character interface layer, which runs on primitive teletype terminals, makes the program easily portable, and a batch interface layer makes it possible to submit a job in batch by specifying a single command line. Fig. 7.14 visualizes this program architecture. Note that direct calls of the program to the system specific window library, which would make portability impossible, are avoided.

The different parts of the program have the following number of lines of source code (version 1.0; including comments and empty lines):

portable core program:	50'000	lines
character interface layer:	7'500	lines
window interface layer:	17'000	lines
batch interface layer:	500	lines
initializations and resource definitions:	6'500	lines

This list shows, that in spite of the attempt of making the user-interface layer as thin as possible, due to the large number of dialog boxes (more than 60), the user interface layer also requires a large programming effort. The batch interface layer is much smaller than the layers of the interactive versions, because editing a model is not supported by this version. The batch interface layer only parses the command line, then it calls the function `Load(...)` of the class `AQUASYS` given in program fragment (7.24), then, depending on the program task, it calls the functions `InitCalc()` and `Calculate()`, `SensAnal(...)`, `Fit(...)`, `ListResults(...)` or `PlotResults(...)` (member functions of the class `AQUASYS`), and, if a calculation was performed, the program terminates after calling the function `Save(...)` of the class `AQUASYS`.

8 Examples of Applications

In this chapter, examples of applications of the program AQUASIM, which was developed according to the guidelines described in this report, are discussed. The goal of this chapter is to illustrate the flexibility of the model formulation framework of AQUASIM by demonstrating how this program can be used to perform simulations and data analyses using well-known models from different areas of environmental science and technology. Although it was not the main goal of the development of AQUASIM to implement very complicated models, which are extremely difficult to identify, the possibility of using such models with AQUASIM demonstrates the flexibility of this program. The data analysis features of AQUASIM are designed to motivate the users to question model assumptions and to check model identifiability.

Each of the sections of this chapter treats the application of AQUASIM to an environmental or technical system. Although different examples are used to introduce different features of AQUASIM, there is a general structure used for all sections: First, the model chosen as the example of an AQUASIM application is briefly reviewed and references to more complete descriptions are given. Then, the spatial configuration of compartments and links used for model application is described. As a next step, it is shown how the model together with the spatial configuration can be implemented as an AQUASIM system. All AQUASIM variables, processes, compartments and links are listed, and important concepts are explained. Finally, the presentation of the results of a typical simulation (if possible, the reproduction of results of the original model publication), in some cases supplemented by a sensitivity analysis or a parameter estimation, concludes each example. Because the AQUASIM implementation of each example is given in detail, this chapter is of considerable length. A shorter survey of the capabilities of AQUASIM is given in Reichert (1994a). In order to obtain a unique notation for all examples, the names of variables may differ slightly from those used in the original publication of the models. This difficulty cannot be avoided, because the authors of different models use different symbols for the same quantities. It should, however, be simple to understand the notation used in this chapter.

The first example illustrates the simulation, parameter identifiability and parameter estimation capabilities of AQUASIM by an application to a dynamic model for the rate of photosynthesis of algae. It is first shown how this model, which is formulated as a system of ordinary differential equations, can be mapped to an AQUASIM system. After reproduction of a simulation with the original values of the model parameters, a sensitivity analysis is performed to help assessing the identifiability of model parameters. Finally, the parameter estimation algorithm of AQUASIM is used to try to determine optimal parameter values and to check identifiability.

The second example uses modeling of activated sludge waste water treatment to demonstrate the implementation of a biochemical process system. In addition, sludge recirculation illustrates how substance specific splitting of mass fluxes can be realized and the implementation of a waste water treatment plant with several reactors demonstrates advective coupling of compartments. At the beginning of the section, a simple activated sludge model serves as an introduction for readers not familiar with

activated sludge waste water treatment. Then, it is shown, how the much more complex IAWPRC activated sludge model no. 1 can be used with the aid of AQUASIM. In contrast to most other programs which implement this model, with the aid of AQUASIM, model extensions can be tested and optimal values of parameters and the uncertainty of calculated results can be estimated.

The third example demonstrates the application of AQUASIM to biofilm systems by the implementation of a problem of competition between a heterotrophic and an autotrophic population in a biofilm. The dependence of the spatial distribution of microorganisms in the biofilm as a function of substrate concentration in the water phase outside the film discussed in the original publication is reproduced with AQUASIM.

The fourth example demonstrates connection of compartments by a diffusive link by discussing xylene degradation in a membrane-bound biofilm. In this example, oxygen is supplied through a membrane, which builds the substratum of the film, whereas xylene is contained in the water phase at the other side of the film. Xylene conversion takes place in the interior of the film, where both substances are available. The model selected for the AQUASIM example is a simplification of the model presented in the original publications for simulating this experiment.

The fifth example demonstrates the application of AQUASIM to rivers by an implementation of the predecessor of all river water quality models, the Streeter-Phelps model. To demonstrate two possibilities for the specification of the river bed geometry, in this example, two river sections are combined to the river reach to be modelled and different techniques for river geometry specification are applied to different river sections. The calculation of backwater effects and weir-reaeration is illustrated by considering a weir between the two river sections.

The set of examples of AQUASIM applications described in this chapter illustrates the utility of the program for usage in research in the environmental sciences. It should be noted, however, that not all program features are demonstrated in this chapter.

8.1 Dynamic Modelling of Algal Photosynthesis

The simplest type of continuous time models consists of one or several ordinary differential equations without explicit reference to any of the spatial compartments of AQUASIM. Models of this type were not the central focus for the development of AQUASIM, because programs of the class described in section 3.1.1 are already available for their implementation. Nevertheless, models of this class are especially well suited for demonstrating the capabilities of AQUASIM with respect to flexibility in process formulation, to system identification tasks and to parameter estimation, because there is no complicating spatial structure.

It is well known that the rate of photosynthesis of algae depends on light in a dynamic way (e.g. Reynolds, 1984). The response of photosynthesis to increasing light is delayed by a characteristic time on the order of minutes, whereas light inhibition decays on a time scale of hours. In ignorance of this fact, most mathematical models of primary production assume a static relationship between the rate of production and light intensity (e.g. Collins and Park, 1989). An example of an exception to these static models is the phenomenological dynamic production model DYPHORA of Pahl-Wostl and Imboden (1990) (see also Pahl-Wostl, 1992). This model is used as a first example of an AQUASIM application.

The model is described in Pahl-Wostl and Imboden (1990). To demonstrate its implementation in AQUASIM, the equations are reviewed briefly in the following paragraphs (with a small change due to Pahl-Wostl, unpublished).

The primary production rate $\hat{P}$ ($ML^{-3}T^{-1}$) of is given as the product of a maximum production rate $\hat{P}_{max}$ (in the units as measured in the experiment) and a non-dimensional relative production rate P (-), which takes values between zero and one:

$$\hat{P} = \hat{P}_{max} P \quad . \tag{8.1}$$

The relative production rate is given by the quotient of the potential production rate, which neglects light inhibition, and a term $(1+\psi)$ which expresses the attenuation due to light inhibition ($0 \leq \psi$). The potential production rate is given as the minimum of the equilibrium potential production rate P^*_{eq} ($ML^{-3}T^{-1}$) and a delayed potential production rate P^* which takes into account the response time to light intensity changes. This leads to the following expression for the relative production rate P:

$$P = \begin{cases} \dfrac{P^*}{1+\psi} & \text{for} \quad P^* \leq P^*_{eq} \\ \dfrac{P^*_{eq}}{1+\psi} & \text{for} \quad P^* > P^*_{eq} \quad . \end{cases} \tag{8.2}$$

The equilibrium potential production rate is given by

$$P^*_{eq} = \tanh(I/I_k) \quad , \tag{8.3}$$

where I_k ($M^{-1}T^{-3}$) is a characteristic light intensity for saturation. The delayed potential production rate P^* and the light inhibition function ψ are given as the solutions of the following two differential equations:

$$\frac{\partial P^*}{\partial t} = -\frac{1}{\tau_r}(P^* - P^*_{eq}) \tag{8.4}$$

$$\frac{\partial \psi}{\partial t} = K I_{ex} - \frac{1}{\tau_i}\psi \quad , \tag{8.5}$$

where τ_r (T) is the response time of P^*, K ($M^{-1}T^2$) is the growth coefficient of light inhibition and the inhibiting excess light intensity I_{ex} ($M^{-1}T^{-3}$) is given as the amount by which the current light intensity exceeds the critical light intensity for onset of light inhibition I_{crit}:

$$I_{ex} = \begin{cases} 0 & \text{for } I \le I_{crit} \\ I - I_{crit} & \text{for } I > I_{crit} \end{cases} , \tag{8.6}$$

and τ_i (T) is the light inhibition decay time. The model described by equations (8.1) - (8.6) is compared with the data of Harris and Piccinin (1977) as shown in Fig. 1 of Pahl-Wostl and Imboden (1990). The two data series of Harris and Piccinin (1977) correspond to a step increase of light intensity from zero to 90 and 1300 $\mu E/m^2/s$, respectively. For such a special situation, the model (8.1) - (8.6) can analytically be solved to give

$$\hat{P} = \hat{P}_{max} \frac{\tanh(I/I_k)\left(1 - e^{-t/\tau_r}\right)}{1 + K I_{ex} \tau_i \left(1 - e^{-t/\tau_i}\right)} \quad , \tag{8.7}$$

where I is the light intensity after the step, which takes place at t=0. With such an analytical solution, the model parameters could be fitted to a data series with the aid of a usual statistical program package. In the following, it is shown, how a parameter sensitivity analysis and a simultaneous parameter fit to both data series can be performed with AQUASIM without using the analytical solution (it is important that this is possible, because in most realistic cases, there is no analytical solution).

In order to implement the model (8.1) - (8.6) in AQUASIM, the differential equations (8.4) and (8.5) have to be formulated with the aid of dynamic processes as described in section 4.2.1. This can be done by declaring P^* and ψ as dynamic state variables and the right hand sides of (8.4) and (8.5) as process rates of two dynamic processes. Using the notation introduced with Table 4.1, a possible process matrix describing equations (8.4) and (8.5) is given in Table 8.1. Because each process does only in-

Table 8.1: Process matrix of DYPHORA.

Name	Variables P^*	Variables ψ	Rate
Process for P^*	-1		$\frac{1}{\tau_r}(P^* - P^*_{eq})$
Process for ψ		1	$K I_{ex} - \frac{1}{\tau_i}\psi$

fluence one variable, the splitting of the rate of a variable into a stoichiometric coefficient and a process rate common to all variables would not be necessary for this model. Table 8.1 demonstrates, however, that this general process formulation easily can be applied to the simpler case. The advantages of the process formulation according to Table 4.1 will become obvious in the other examples given in this chapter. Summarizing, DYPHORA can be mapped to the following "biochemical" model:

Compounds:

P^* : delayed potential production rate (-),
ψ : light inhibition function (-).

Processes:

Process for P^* : relaxation of P^* towards P^*_{eq},
Process for ψ : production and decay of light inhibition.

Kinetic Parameters:

$\hat{P}_{max}$: maximum production rate ($ML^{-3}T^{-1}$),
τ_i : light inhibition decay time (T),
τ_r : response time of P^* (T),
K : growth coefficient of light inhibition ($M^{-1}T^2$),
I_k : characteristic light intensity for saturation ($M^{-1}T^{-3}$),
I_{crit} : critical light intensity for onset of light inhibition ($M^{-1}T^{-3}$).

In order to completely specify the system to be implemented in AQUASIM, the spatial configuration within which the effects of the biochemical model shown in Table 8.1 should be investigated has to be specified. In this case, a single mixed reactor compartment without inflow obviously is adequate to describe the experimental setup. This simple AQUASIM configuration is illustrated in Fig. 8.1.

The variables used to describe the model and the data sets with AQUASIM are listed in Table 8.2. Light intensity is made to depend on the program variable calculation number to make it possible to perform a simultaneous fit of production rates at both light intensities measured by Harris and Piccinin (1977). It is not necessary to implement light intensities as step functions over time, because initial conditions of zero for P^* and ψ account for the darkness before start of integration (implementa-

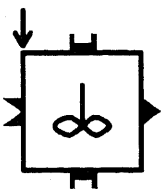


Fig. 8.1: Spatial configuration used for the implementation of DYPHORA.

Table 8.2: Variables of model DYPHORA (Type: VSV = dynamic volume state variable, PV = program variable, CV = constant variable, RLV = real list variable, VLV = variable list variable, FV = formula variable).

Name	Type	Meaning	Unit	Value	Comment
calcnum	PV	calculation number	-	1 or 2	for distinguishing the experiments
I	RLV	I	$\mu E/m^2/s$	(90 or 1300 for calculation number 1 or 2, respectively)	conditions for two experiments
I_crit	CV	I_{crit}	$\mu E/m^2/s$	50	kinetic parameter
I_ex	FV	I_{ex}	$\mu E/m^2/s$	max(I-I_crit,0)	equation (8.6)
I_k	CV	I_k	$\mu E/m^2/s$	790	kinetic parameter
K	CV	K	$m^2/\mu E$	$3.6 \cdot 10^{-6}$	kinetic parameter
P	FV	P	-	if P_star<P_stareq then P_star/(1+Psi) else P_stareq/(1+Psi) endif	equation (8.2)
Prod	FV	$\hat{P}$	mgO/l/h	P_max*P	equation (8.1)
Prod1	RLV	$\hat{P}$ (meas 1)	mgO/l/h	(data of Harris and Piccinin, 1977)	data series for fit
Prod2	RLV	$\hat{P}$ (meas 2)	mgO/l/h	(data of Harris and Piccinin, 1977)	data series for fit
Psi	VSV	ψ	-	-	solution of eq. (8.5)
P_max	CV	P_{max}	mgO/l/h	1.65	kinetic parameter
P_star	VSV	P^*	-	-	solution of eq. (8.4)
P_stareq	FV	P^*_{eq}	-	tanh(I/I_k)	equation (8.3)
t	PV	t	s	-	makes time available to be used as an argument for Prod1 and Prod2
tau_i	CV	τ_i	s	1200	kinetic parameter
tau_r	CV	τ_r	s	240	kinetic parameter

tion of time dependent light intensities corresponding to different calculation numbers would be possible by implementing these time dependences as formula variables or asreal list variables and changing the type of the variable I from a real list variable to a variable list variable, in which the values of light intensity are replaced by the names of the variables. The extension to more than two different light intensities can obviously be done by adding additional elements to the real list variable or variable list variable I. The 6 model parameters $\hat{P}_{max}$ (P_max), τ_i (tau_i), τ_r (tau_r), K (K), I_k (I_k) and I_{crit} (I_crit) are implemented as constant variables to allow them being estimated by the program. The variables P_stareq and Prod show two simple examples of algebraic expressions possible in formula variables. The possibility of logical branching in formula variables either using maximum or minimum functions or if-then-else-endif constructs is demonstrated by the variables I_ex and P. To allow P_star and Psi to be calculated as the solutions of the differential equations (8.4) and (8.5) these variables are implemented as dynamic volume state variables.

Table 8.3: Dynamic processes of model DYPHORA.

Name	Rate	Stoichiometry	Comment
ProcPsi	K*I_ex-1/tau_i*Psi	Psi: 1	r.h.s. of equation (8.5)
ProcPstar	1/tau_r*(P_star-P_stareq)	P_star: -1	r.h.s. of equation (8.4)

Table 8.4: Compartment of model DYPHORA (Type: MRC = mixed reactor compartment).

Name	Type	Property	Unit	Value	Comment
Reactor	MRC	inflow	1/s	0	no inflow of water
		input fluxes	-	-	no substance inflow
		initial cond.	- -	P_star: 0 Psi: 0	all initial conditions zero (would not be necessary to specify, because zero is the default)
		volume	-	1	fixed volume

The dynamic AQUASIM processes representing the process matrix shown in Table 8.1 are listed in Table 8.3. Each process corresponds to a row of the process matrix. In the special case of this model (as already mentioned above), the stoichiometry of each process consists of only one variable-stoichiometric coefficient pair.

The compartment definition implementing the single mixed reactor compartment shown in Fig. 8.1 is given in Table 8.4. Within this compartment all input fluxes and initial conditions are set to zero. Setting the reactor volume V_R to a constant value of 1 (without unit, because P and ψ are non-dimensional) and the input fluxes I_i to zero makes the equation (4.14) degenerate to the equations (8.4) and (8.5) for the two state variables P_star and Psi (replacing C_i in eqation (4.14)) and the two processes ProcPsi and ProcPstar shown in Table 8.3 (the sum of the products of the stoichiometric coefficient of a state variable times the rate of the process replaces r_i in equation (4.14)).

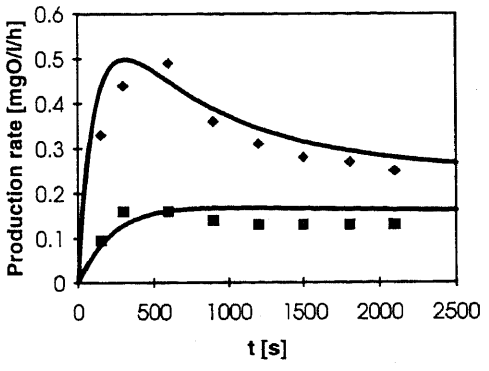


Fig. 8.2: Primary production rates calculated for a step increase of light to two different intensities with model DYPHORA with the parameter values of Pahl-Wostl and Imboden (1990) (solid lines) and measurements of Harris and Piccinin (1977) (symbols).

Performing two simulations from t = 0 s to t = 3000 s (e.g. 150 steps of 20 s) with given initial state (specified in the compartment definition given in Table 8.4 to be $P^* = \psi = 0$) for calculation number 1 (corresponding to I = 90 $\mu E/m^2/s$) and for calculation number 2 (corresponding to I = 1300 $\mu E/m^2/s$) leads to the production curves shown as solid lines in Fig. 8.2. These curves together with the data points of Harris and Piccinin (1977) are a reproduction of Fig. 1 of Pahl-Wostl and Imboden (1990).

As shown by the solid lines in Fig. 8.2, the model (8.1) - (8.6) with the parameter values as chosen by Pahl-Wostl and Imboden (1990) represents the measured data series with an adequate accuracy. It is an interesting question to ask, if these parameter values can be uniquely determined (within a certain accuracy) out of the measured data shown in Fig. 8.2. The methods available for such an identifiability analysis are discussed in section 2.3.2. Theoretical identifiability analysis would usually be performed using Taylor series expansion of the solution (Pohjanpalo, 1978; Godfrey and DiStefano, 1985 and 1987). In this special case, where with equation (8.7) an analytical solution of the model for the given experimental situation is available, this solution much faster gives the answer to the theoretical identifiability problem. Equation (8.7) clearly shows, that using a single light intensity, the parameter pair $\hat{P}_{max}$ and I_k and the parameter pair K and I_{crit} cannot uniquely be identified

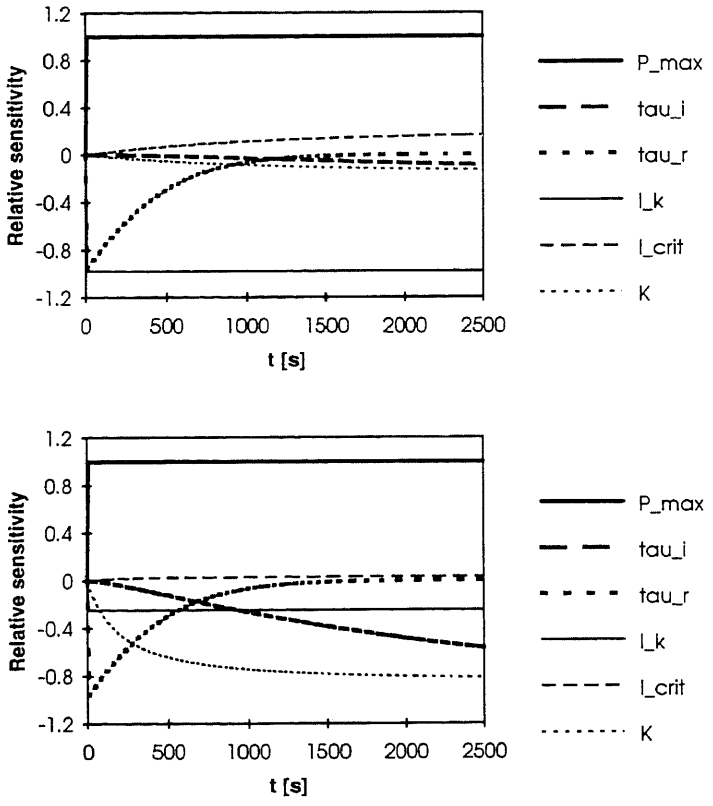


Fig. 8.3: Relative sensitivity functions of primary production rate with respect to all six model parameters according to equation (2.2d) for light intensities of 90 $\mu E/m^2/s$ (top) and 1300 $\mu E/m^2/s$ (bottom).

(there are infinitely many solutions for $\hat{P}_{max}$ and I_k giving the same value of the product $\hat{P}_{max} \tanh(I/I_k)$ for a single light intensity I; the same problem occurs with the product $K\ I_{ex}$ for the parameters K and I_{crit}). Using data from two experiments at different light intensities ($I > I_{crit}$ assumed) solves this identifiability problem, and all six parameters become theoretically identifiable. This means, that it is not possible to find two sets of parameters, which lead to exactly the same solutions at two different light intensities.

Theoretical identifiability is a necessary, but not a sufficient condition for practical identifiability using real noisy data. To judge practical identifiability, it is very useful to look at the relative sensitivity functions $\delta^{r,r}_{f,p_i} = p_i/f \cdot \partial f/\partial p_i$ (according to equation (2.2d)). Fig. 8.3 shows the relative sensitivity functions of primary production rate with respect to all six model parameters at both light intensities for the parameter set of Pahl-Wostl and Imboden (1990) given in Table 8.2 and also listed in Table 8.5 as "original parameter set". The similarity in the shape of the majority of these curves and the small magnitudes (< 0.25 for 5 of 12 curves over the whole time domain and for 7 of 12 curves over a large fraction of the time domain) indicates bad practical identifiability. Similar behavior of the sensitivity functions has the consequence, that changes in production rates due to changes in a parameter can to a large amount be compensated by changes of other parameters.

To verify the bad identifiability suggested from linear sensitivity analysis, parameter fits were performed for several subsets of the whole parameter set. Table 8.5 lists some of the resulting parameter sets together with the total sum of squares (SS) of the deviation between the calculation and both data sets according to equation (2.19) (parameters with the same value as in the original parameter set were excluded from the fit). Fig. 8.4 shows the production rate curves obtained by the parameter sets shown in Table 8.5. The sum of squares values shown in Table 8.5 as well as the plotted curves shown in Fig. 8.4 clearly demonstrate that similarly good agreement between data and calculation can be obtained for various parameter sets. This confirms the practical non-identifiability of the model with these two data sets alone. The last data set, in which the value of τ_i reaches its upper limit, indicates that a simpler model without regeneration of light inhibition would be sufficent to explain the data. It is clear, that such a model is unrealistic; the data analysis only demonstrates that it is not possible to calibrate light inhibition regeneration with the two data sets of Harris and Piccinin (1977). In order to fit also regeneration of light inhibition,

Table 8.5: Parameter values resulting from a fit of different subsets of model parameters to the data.

Parameter set	$\hat{P}_{max}$ [mgO/l/h]	τ_i [s]	τ_r [s]	K [$m^2/\mu E$]	I_k [$\mu E/m^2/s$]	I_{crit} [$\mu E/m^2/s$]	SS [$(mgO/l/h)^2$]
original	1.65	1200	240	$3.6 \cdot 10^{-6}$	790	50	0.028
fit 1	1.41	3100	300	$2.3 \cdot 10^{-6}$	790	50	0.011
fit 2	0.92	1200	195	$1.9 \cdot 10^{-6}$	540	50	0.012
fit 3	0.73	∞	190	$0.8 \cdot 10^{-6}$	410	0	0.007

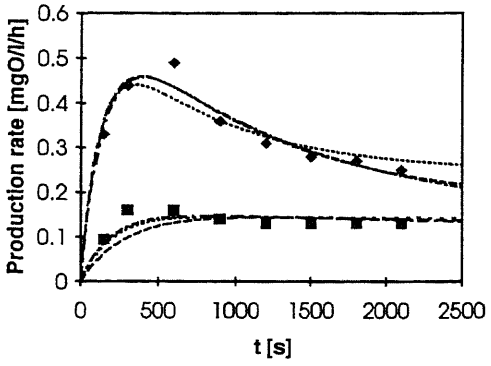


Fig. 8.4: Primary production rates calculated for a step increase of light to two different intensities with the model DYPHORA for the parameter sets 1 to 3 shown in Table 8.5 (lines) and measurements of Harris and Piccinin (1977) (symbols).

experiments including decreases in light intensity would be necessary.

8.2 Modelling Activated Sludge Waste Water Treatment

Mathematical modelling is common practice for design, operation and data analysis of waste water treatment systems since about 10 years. Early modelling experiences lead to the development of the IAWPRC activated sludge model no. 1 published by the IAWPRC task group on modeling for design and operation of biological waste water treatment (Henze et al., 1986). Since then, this model has obtained great acceptance and has been implemented in several computer programs. Because this model consists of a complex biochemical process system with 8 processes transforming 13 substances, it is well suited to demonstrate the process formulation flexibility of AQUASIM. Furthermore, in comparison to other implementations of this model, an AQUASIM implementation has the advantage, that model extensions can be tested, that sensitivity analyses can be performed and that parameters can be estimated automatically using measured data. Before discussing the complex IAWPRC activated sludge model no. 1, as an introduction to activated sludge modelling, a very much simplified model is discussed.

A Simple Activated Sludge Model

In their didactic paper, Gujer and Henze (1991) discuss several modeling levels of the activated sludge process. All of these models are simplifications of the IAWPRC activated sludge model no. 1 mentioned above. The simplest model, which is able to represent the main features of oxygen uptake rates, is model B of Gujer and Henze (1991). The compounds, processes and parameters of this model can be summarized as follows:

Compounds:

C_{O_2} : dissolved oxygen concentration (ML^{-3}),
C_S : readily biodegradable substrate concentration (ML^{-3}),
X_H : heterotrophic biomass concentration (ML^{-3}),
X_S : slowly biodegradable substrate concentration (ML^{-3}).

Processes:

Aerobic growth of heterotrophs :	growth of heterotrophic biomass with consumption of oxygen and readily biodegradable substrate,
Decay of heterotrophs :	decay of heterotrophic biomass into slowly biodegradable substrate,
Hydrolysis of organics :	dissolution of particulate slowly biodegradable substrate into dissolved readily biodegradable substrate.

Stoichiometric Parameters:

Y_H : yield for heterotrophic biomass (-).

Table 8.6: Process matrix of the activated sludge model B.

Name	Variables C_{O_2}	C_S	X_H	X_S	Rate
Aerobic growth of heterotrophs	$-\frac{1-Y_H}{Y_H}$	$-\frac{1}{Y_H}$	1		$\mu_H \frac{C_S}{K_S+C_S} \frac{C_{O_2}}{K_{O_2,H}+C_{O_2}} X_H$
Decay of heterotrophs			-1	1	$b_H X_H$
Hydrolysis of organics		1		-1	$k_h \frac{X_S/X_H}{K_X+X_S/X_H} X_H$

Kinetic Parameters:

μ_H : maximum specific growth rate for heterotrophic biomass (T^{-1}),
K_S : substrate half-saturation coefficient for heterotrophic biomass (ML^{-3}),
$K_{O_2,H}$: oxygen half-saturation coefficient for heterotrophic biomass (ML^{-3}),
b_H : decay coefficient for heterotrophic biomass (T^{-1}),
k_h : maximum specific hydrolysis rate (T^{-1}),
K_X : half-saturation coefficient for hydrolysis of slowly biodegradable substrate (-).

The biochemical process matrix of this model is given in Table 8.6. This table uses the notation introduced in section 4.2.1 (cf. Table 4.1).

The experimental situation, to which the model is applied, consists of a single mixed reactor which is fed each day between 02^{oo} and 14^{oo} with constant input of C_S and X_S and where 85% of the outflowing particles (X_S and X_H) are recirculated. The spatial configuration of this experiment is shown in Fig. 8.5. It consists of a mixed reactor compartment and an advective link, a bifurcation of which models sludge recirculation.

Table 8.7 shows the AQUASIM variables used for implementing the model and the experimental situation. Model parameters, inflow concentrations, inflow discharge and initial conditions are implemented as constant variables. Since the reactor is sufficiently aerated, oxygen concentration is held constant and is not treated as a state variable. Therefore, the 3 state variables represent concentrations of C_S, X_S

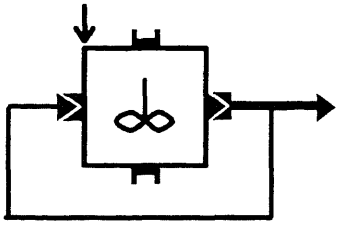


Fig. 8.5: Spatial configuration of a mixed reactor with sludge recirculation as used for the implementation of the activated sludge model B.

Table 8.7: Variables of the activated sludge model B (Type: VSV = dynamic volume state variable, PV = program variable, CV = constant variable, RLV = real list variable, VLV = variable list variable, FV = formula variable).

Name	Type	Mean.	Unit	Value	Comment
b_H	CV	b_H	1/d	0.6	kinetic parameter
C_O2	FV	C_{O_2}	gO/m^3	2	given oxygen conc.
C_S	VSV	C_S	$gCOD/m^3$	-	conc. calculated by prog.
C_Sin	CV	$C_{S,in}$	$gCOD/m^3$	110	experimental parameter
k_h	CV	k_h	1/d	2	kinetic parameter
K_O2H	CV	$K_{O_2,H}$	gO/m^3	0.1	kinetic parameter
K_S	CV	K_S	$gCOD/m^3$	5	kinetic parameter
K_X	CV	K_X	-	0.04	kinetic parameter
mue_H	CV	μ_H	1/d	4	kinetic parameter
Q	PV	Q	m^3/d	-	makes discharge available for use in advective link
Q_in	CV	Q_{in}	m^3/d	0.036	experimental parameter
recircX	CV		-	0.85	particle fraction to be recirculated
r_HetGro	FV	r_{GroHet}	$gCOD/m^3/d$	mue_H*C_S/(K_S+C_S) *C_O2/(K_O2H+C_O2)*X_H	het. spec. growth rate (used in r_O2 and in processes)
r_O2	FV	r_{O_2}	$gO/m^3/d$	(1-Y_H)/Y_H*r_HetGro	respiration rate to be compared with measurements
r_O2meas	RLV	r_{O_2}	$gO/m^3/d$	(data of Ekama and Marais, 1978)	respiration measurements to be compared with calculation
switch	RLV		-	(1 between thour=2 and thour=14, else 0)	switch for turning inflow on and off
t	PV	t	d	-	makes time available for use in h and in r_O2meas
thour	FV	t	h	((t+10) mod 1)*24	hours of the day for formulating periodic inflow (cf. switch)
X_H	VSV	X_H	$gCOD/m^3$	-	conc. calculated by program
X_Hini	CV	$X_{H,ini}$	$gCOD/m^3$	100	experimental parameter
X_S	VSV	X_S	$gCOD/m^3$	-	conc. calculated by program
X_Sin	CV	$X_{S,in}$	$gCOD/m^3$	430	experimental parameter
Y_H	FV	Y_H	-	0.67	stoichiometric parameter

and X_H. Because oxygen uptake rates have to be compared with measured data, a variable r_O2 was defined to calculate this quantity.
Since the rate of heterotrophic growth is used twice, for formulating r_O2 as well as for the process definition, to avoid redundance (which can lead to errors in possible

Table 8.8: Dynamic processes of the activated sludge model B.

Name	Rate	Stoichiometry	Comment
HetDecay	b_H*X_H	X_H: -1 X_S: 1	
HetGro	r_GroHet	C_O2: -(1-Y_H)/Y_H C_S: -1/Y_H X_H: 1	the rate is directly available as a variable (cf. Table 8.7)
HydrolOrg	if X_S>0 then k_h*X_S/(K_X*X_H+X_S)*X_H else 0 endif	C_S: 1 X_S: -1	this formulation of the rate is more robust as compared to Table 8.6 because values of X_H or X_S of zero do not cause a division by zero

model changes), it is also formulated as a variable (r_HetGro). A variable (switch) is used to turn the inflow on and off periodically each day. This variable is formulated with the aid of the hours of the day (thour) which are calculated with the aid of a formula variable using the modulo function (10 is added to time t to make the use of negative times up to -10 days possible).

Table 8.8 shows the dynamic processes used for the formulation of the process matrix given in Table 8.6. The correspondence between these two tables is obvious. Since oxygen (C_O2) is not treated as a state variable (cf. Table 8.7), the stoichiometric coefficient of the process HetGro for C_O2 has no effect. Even though, it is advantageous to include this stoichiometric coefficient into the process definition, to prepare the system to a possible change of C_O2 into a state variable.

The experimental setup of Ekama and Marais (1978), the data of which is used for a comparison with the model calculation, is shown in Fig. 8.5. It consists of a single mixed reactor compartment and an advective link, a bifurcation of which is used for modelling sludge recirculation. The AQUASIM definition of the mixed reactor compartment is given in Table 8.9. The water inflow and the substance input fluxes are formulated as products of the values during operation of the reactor with the variable switch, which turns inflow on and off. A nonzero initial condition for X_H is necessary to start growth of heterotrophic bacteria. State variables without explicitly specified initial conditions are initially set to zero. The reactor has a fixed volume. The definition of the advective link used for modelling sludge recirculation is given in Table 8.10. The inflow to this link is connected to the outflow of the reactor, whereas the outflow of the link leaves the modelled system. A bifurcation, which makes it

Table 8.9: Compartments used for the application of the activated sludge model B (Type: MRC = mixed reactor compartment).

Name	Type	Property	Unit	Value	Comment
Reactor	MRC	inflow	m^3/d	switch*Q_in	periodic inflow
		input fluxes	gCOD/d gCOD/d	C_S: switch*Q_in*C_Sin X_S: switch*Q_in*X_Sin	periodic feed specified as mass per unit time
		initial cond.	$gCOD/m^3$	X_H: X_Hini	nonzero initial condition required for starting growth
		volume	m^3	0.00673	fixed volume

Table 8.10: Advective link of the configuration for applying the activated sludge model B.

Name	Connection	Connected to			Comment
Outflow	In	outflow of compartment "Reactor"			reactor outflow feeds the link
	Out	-			link outflow leaves the modelled system
	Recirc	inflow of compartment "Reactor"			bifurcation back to the reactor input
		Property	Unit	Value	
		water flow	m^3/d	0	no water recirculated
		mass fluxes	gCOD/d gCOD/d	X_H: recirc_X*Q*X_H X_S: recirc_X*Q*X_S	fraction of recirculated particulate mass flux defined by variable recircX

possible to separate substances by the specification of substance specific mass fluxes is connected to the inflow of the reactor and thus models recirculation of 85% (cf. variable recircX in Table 8.7) of the flux of particulate material represented by the state variables X_H and X_S.

The measured respiration data of Ekama and Marais (1978) corresponds to periodic feeding of the reactor. In order to approximately reach periodic behavior of the state variables in the reactor interior (concentrations of C_S, X_H and X_S), some days of transient operation with periodic feed is necessary. This transient stage is shown in Fig. 8.6. This figure clearly shows that the initial condition (X_Hini in Table 8.7) of the heterotrophic bacteria (X_H) is much smaller than the concentration adapted to the feeding conditions and that growing up of these bacteria needs several days.

After 6 days of operation, when the periodic state is reached with a reasonable accuracy, the respiration rate can be compared with the data of Ekama and Marais (1978). Fig. 8.7, which exactly corresponds to Fig. 3 of Gujer and Henze (1991), demonstrates the agreement between calculation and measurement. Since C_O2 is constant and X_H has only a small variation (cf. Fig. 8.6), the shape of the respira-

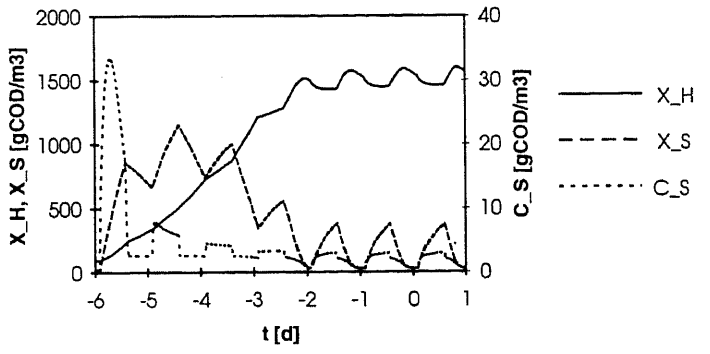


Fig. 8.6: Transient phase of the reactor required for growth of equilibrium heterotrophic population calculated with model B of Gujer and Henze (1991).

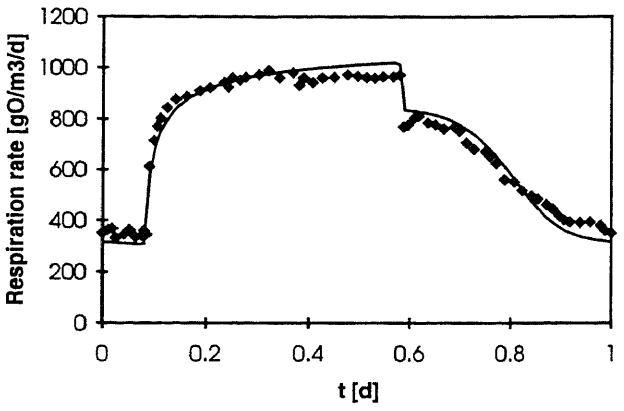


Fig. 8.7: Comparison of respiration rate in the periodic equilibrium state after 6 days of operation (compare Fig. 8.6) with data measured by Ekama and Marais (1978).

tion curve is determined by the magnitude of available substrate C_S (cf. first row of Table 8.6). The feed of substrate between 0.083 d (02^{00}) and 0.583 d (14^{00}) causes a rapid increase of respiration rate due to heterotrophic growth. After stopping the feed at 0.583 days, respiration decreases very fast to a new dynamic equilibrium between substrate consumption by heterotrophic biomass and substrate production by hydrolysis. Between 0.583 d and 1 d the decrease of the respiration rate is much slower because hydrolytic decay of X_S is now the rate limiting process. This decrease goes parallel to the decrease in slowly biodegradable substrate (X_S) visible in Fig. 8.6.

Sensitivity analysis of respiration with respect to all 6 model parameters shown in Fig. 8.8 leads to the following results: Although heterotrophic growth is the cause of respiration, the respiration rate is extremely insensitive to the parameters mue_H, K_S and K_O2H of this process. This clearly shows, that also during the feeding period of the reactor, hydrolysis is the rate-limiting process. It is interesting to look also at the sign of the sensitivity function of respiration with respect to maximum specific hydrolysis rate k_h. During the feeding period of the reactor, this function is positive, indicating an increase of respiration due to an increase of the specific rate of hydrolysis of slowly biodegradable substrate. This is possible because the feed of slowly biodegradable substrate prevents the concentration of this compound to decrease. After turning off the feed (at t = 0.583 d), an increase of the specific rate of hydrolysis only temporarily increases respiration. After a short time, the higher specific hydrolysis rate leads to a decrease of the concentration of slowly biodegradable substrate, which is large enough to decrease the effective hydrolysis rate. This causes the relative sensitivity function to become negative. At the same time, due to the production of slowly biodegradable substrate, decay becomes the important source for hydrolizable material during operation without feed. This is seen in Fig. 8.8 by an increase of the sensitivity function of the parameter b_H during this time.

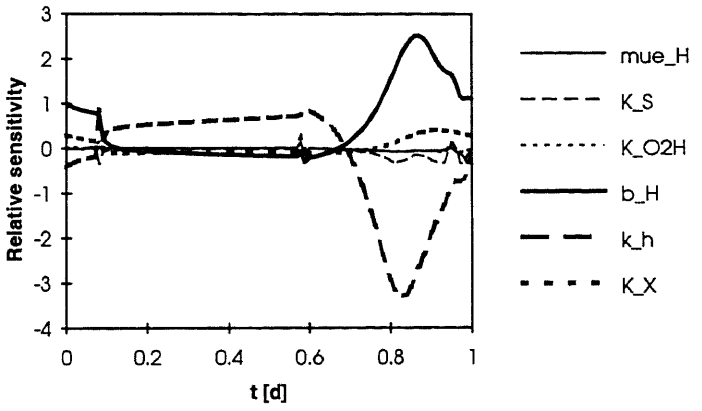


Fig. 8.8: Relative sensitivity functions of respiration with respect to all model parameters according to equation (2.2d).

IAWPRC Activated Sludge Model No. 1

The IAWPRC activated sludge model no. 1 takes into account much more compounds and processes than the simple model described above. The most important extension is the description of the nitrogen cycle in addition to the carbon balance as formulated in the simple activated sludge model B. The additional processes are anoxic growth of heterotrophic bacteria using of nitrate instead of dissolved oxygen (denitrification) and growth (nitrification) and decay of autotrophic bacteria. To close the nitrogen cycle, particulate organic nitrogen can be hydrolised and subsequently ammonificated to become available for nitrifying bacteria. To avoid the introduction of a variable stoichiometry of organic material, organic carbon (measured as COD) and organic nitrogen (measured as N) are treated as different model compounds although they are in reality parts of the same organic material. For a more detailed description of this model, consult the original publication by Henze et al. (1986). The model compounds, processes and parameters can be summarized as follows:

Compounds:

C_I : concentration of dissolved inert organic matter (ML^{-3}),
C_S : concentration of readily biodegradable substrate (ML^{-3}),
X_I : concentration of particulate inert organic matter (ML^{-3}),
X_S : concentration of slowly biodegradable substrate (ML^{-3}),
X_H : heterotrophic biomass concentration (ML^{-3}),
X_A : autotrophic biomass concentration (ML^{-3}),
X_P : concentration of particulate inert products arising from biomass decay (ML^{-3}),
C_{O_2} : dissolved oxygen concentration (ML^{-3}),
$C_{NO_3^-}$: sum of nitrate and nitrite nitrogen concentration (ML^{-3}),
$C_{NH_4^+}$: ammonium nitrogen concentration (ML^{-3}),
C_{ND} : concentration of dissolved biodegradable organic nitrogen (ML^{-3}),
X_{ND} : concentration of particulate biodegradable organic nitrogen (ML^{-3}),
C_{ALK} : alkalinity (ML^{-3}).

Processes:

Aerobic growth of heterotrophs :	growth of heterotrophic biomass with consumption of oxygen, readily biodegradable substrate and ammonia,
Anoxic growth of heterotrophs :	growth of heterotrophic biomass in absence of dissolved oxygen using nitrate instead of oxygen as electron acceptor,
Aerobic growth of autotrophs :	growth of autotrophic biomass using the oxidation of ammonia to nitrate as an energy source (nitrification),
Decay of heterotrophs :	decay of heterotrophic biomass into slowly biodegradable substrate and inert particulate organic matter,
Decay of autotrophs :	decay of autotrophic biomass into slowly biodegradable substrate and inert particulate organic matter,

Ammonification : conversion of organic nitrogen into ammonia,
Hydrolysis of organics : dissolution of particulate slowly biodegradable substrate into dissolved readily biodegradable substrate,
Hydrolysis of organic nitrogen : dissolution of particulate biodegradable organic nitrogen into dissolved biodegradable organic nitrogen.

Stoichiometric Parameters:

Y_H : yield for heterotrophic biomass (-),
i_{CI} : nitrogen fraction of dissolved inert organic matter (-),
i_{XB} : nitrogen fraction of biomass (-),
i_{XI} : nitrogen fraction of particulate inert organic matter (-),
i_{XP} : nitrogen fraction of particulate products arising from biomass decay (-),
f_p : inert fraction of biomass decay (-).

Kinetic Parameters:

μ_H : maximum specific growth rate for heterotrophic biomass (T^{-1}),
K_S : substrate half-saturation coefficient for heterotrophic biomass (ML^{-3}),
$K_{O2,H}$: oxygen half-saturation coefficient for heterotrophic biomass (ML^{-3}),
K_{NO3^-} : nitrate half-saturation coefficient for denitrifying heterotrophic biomass (ML^{-3}),
η_g : correction factor for μ_H under anoxic conditions (-),
μ_A : maximum specific growth rate for autotrophic biomass (T^{-1}),
K_{NH4^+} : ammonia half-saturation coefficient for autotrophic biomass (ML^{-3}),
$K_{O2,A}$: oxygen half-saturation coefficient for autotrophic biomass (ML^{-3}),
b_H : decay coefficient for heterotrophic biomass (T^{-1}),
b_A : decay coefficient for autotrophic biomass (T^{-1}),
k_a : ammonification rate (T^{-1}),
k_h : maximum specific hydrolysis rate (T^{-1}),
K_X : half-saturation coefficient for hydrolysis of slowly biodegradable substrate (-),
η_h : correction factor for hydrolysis under anoxic conditions (-),
θ_{bA} : coefficient for temperature dependence of b_A (θ^{-1}),
θ_{bH} : coefficient for temperature dependence of b_H (θ^{-1}),
θ_{ka} : coefficient for temperature dependence of k_a (θ^{-1}),
θ_{kh} : coefficient for temperature dependence of k_h (θ^{-1}),
θ_{KX} : coefficient for temperature dependence of K_X (θ^{-1}),
$\theta_{\mu A}$: coefficient for temperature dependence of μ_A (θ^{-1}),
$\theta_{\mu H}$: coefficient for temperature dependence of μ_H (θ^{-1}).

The biochemical process matrix of this model is given in Table 8.11. Note that the inert fractions S_I and X_I are not affected by the model processes. They were included into the model to make the calculation of global parameters such as total COD and total particle concentration possible. The temperature dependence of kinetic parameters is calculated with an exponential function according to

$$p(T) = p(20^{\circ}C)\, e^{\theta\,(T-20^{\circ}C)} \quad , \tag{8.8}$$

where $p(20^{\circ}C)$ and θ are chosen to match the values of the parameters at 10 °C and at 20 °C given by Henze et al. (1986).

The model as defined above is applied to a typical situation of a waste water treatment plant. The spatial configuration of the plant to be modelled consists of 3 reactors for biological treatment (one for denitrification and two for nitrification) and a clarifier. All input is released into the first reactor. Sludge is recirculated from the outlet of reactor 3 to the inlet of reactor 1. Fig. 8.9 shows a pictogram of the spatial configuration of the plant.

To start a simulation with the model defined in Table 8.11 for the configuration shown in Fig. 8.9 the initial concentrations of all 13 substances in all 4 reactors and the inflows of water and all 13 substances have to be given. The initial conditions are only important for a transient phase an can therefore be chosen arbitrarily, if a simulation long enough to establish equilibrium operation is performed. Because chemical analysis of waste water composition is difficult, detailed knowledge of this composition is only available in special cases of intensive investigations. An engineer planning a waste water treatment plant has to be able to perform simulations with an estimated waste water composition. Such an estimation consists of distributing measured global COD and nitrogen concentrations in a reasonable way (based on experience with the particular type of waste water) into the substance classes considered in the model. Although AQUASIM is designed as a tool for scientific investigations, in which input concentrations have to be measured, to demonstrate how AQUASIM variables can be used to support the distribution of global input concentrations to all model variables using given waste water characteristics, this way of specifying input is chosen for the AQUASIM example implementation of the IAWPRC activated sludge model no. 1. The method bases on the ideas realized in the computer program AS40 selled by Aqua System AG, Winterthur, Switzerland, which commercially implements this model. Fig. 8.10 illustrates the fractions of total carbon and nitrogen taken into account to estimate model compound concentrations. Note that C_I, X_I, X_A and X_H contain carbon and nitrogen, whereas S_S and X_S are the carbon fractions and C_{ND} and X_{ND} the nitrogen fractions of dissolved and particulate substrate, respectively.

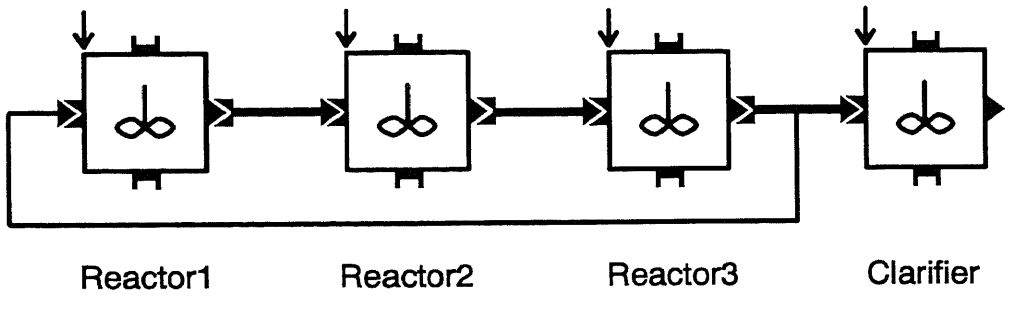


Fig. 8.9: Spatial configuration of the waste water treatment plant to be modelled with the IAWPRC activated sludge model no. 1.

Table 8.11: Process matrix of the IAWPRC activated sludge model no. 1.

Name	Variables													Rate
	C_I	C_S	X_I	X_S	X_H	X_A	X_P	C_{O_2}	$C_{NO_3^-}$	$C_{NH_4^+}$	C_{N_D}	X_{N_D}	C_{ALK}	
Aerobic growth of heterotrophs		$\frac{-1}{Y_H}$			1			$-\frac{1-Y_H}{Y_H}$		$-i_{XB}$			$-\frac{i_{XB}}{14}$	$\mu_H \frac{C_S}{K_S+C_S} \frac{C_{O_2}}{K_{O_2,H}+C_{O_2}} X_H$
Anoxic growth of heterotrophs		$\frac{-1}{Y_H}$			1				$\frac{-(1-Y_H)}{2.86Y_H}$	$-i_{XB}$			$\frac{1-Y_H}{14 \cdot 2.86 Y_H} - \frac{i_{XB}}{14}$	$\mu_H \frac{C_S}{K_S+C_S} \frac{K_{O_2,H}}{K_{O_2,H}+C_{O_2}} \frac{C_{NO_3^-}}{K_{NO_3^-}+C_{NO_3^-}} \eta_g X_H$
Aerobic growth of autotrophs						1		$\frac{4.57-Y_A}{-Y_A}$	$\frac{1}{Y_A}$	$-i_{XB} - \frac{1}{Y_A}$			$-\frac{i_{XB}}{14} - \frac{1}{7Y_A}$	$\mu_A \frac{C_{NH_4^+}}{K_{NH_4^+}+C_{NH_4^+}} \frac{C_{O_2}}{K_{O_2,A}+C_{O_2}} X_A$
Decay of heterotrophs				$1-f_p$	-1		f_p					$i_{XB}-f_p i_{XP}$		$b_H X_H$
Decay of autotrophs				$1-f_p$		-1	f_p					$i_{XB}-f_p i_{XP}$		$b_A X_A$
Ammonification										1	-1		$\frac{1}{14}$	$k_a S_{N_D} X_H$
Hydrolysis of organics		1		-1										$k_h \frac{X_S/X_H}{K_X+X_S/X_H} \left(\frac{C_{O_2}}{K_{O_2,H}+C_{O_2}} + \eta_h \frac{K_{O_2,H}}{K_{O_2,H}+C_{O_2}} \frac{C_{NO_3^-}}{K_{NO_3^-}+C_{NO_3^-}} \right) X_H$
Hydrolysis of org. nitrogen											1	-1		$k_h \frac{X_S/X_H}{K_X+X_S/X_H} \left(\frac{C_{O_2}}{K_{O_2,H}+C_{O_2}} + \eta_h \frac{K_{O_2,H}}{K_{O_2,H}+C_{O_2}} \frac{C_{NO_3^-}}{K_{NO_3^-}+C_{NO_3^-}} \right) X_H \frac{X_{N_D}}{X_S}$

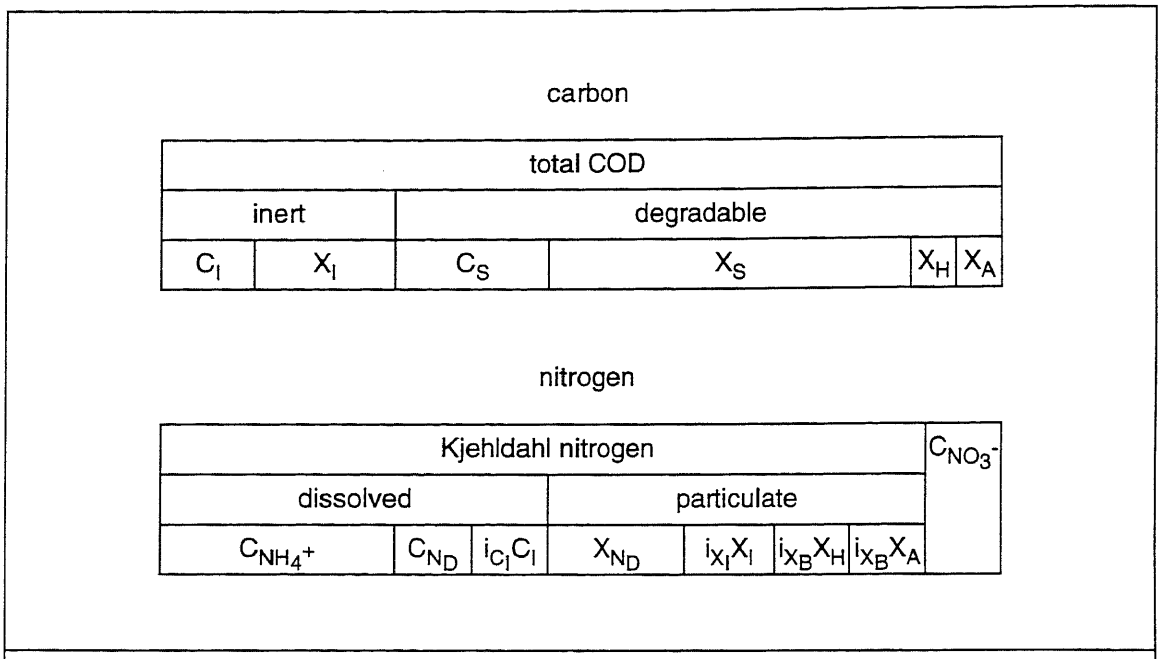


Fig. 8.10: Division of total carbon and nitrogen into model compounds.

Table 8.12 shows the AQUASIM variables used to implement the IAWPRC activated sludge model no. 1 together with the characterization of input and some variables used later for formulating sludge recirculation.

Table 8.12: Variables of the IAWPRC activated sludge model no. 1 (Type: VSV = state variable, PV = program variable, CV = constant variable, RLV = real list variable, VLV = variable list variable, FV = formula variable).

Name	Type	Meaning	Unit	Value	Comment
b_A	FV	b_A (T)	d^{-1}	b_A20 *exp(the_bA*(T-20))	kinetic parameter (with temperature dependence)
b_A20	CV	b_A (20°C)	d^{-1}	0.15	kinetic param. (at 20°C)
b_H	FV	b_H (T)	d^{-1}	b_H20 *exp(the_bH*(T-20))	kinetic parameter (with temperature dependence)
b_H20	CV	b_H (20°C)	d^{-1}	0.62	kinetic param. (at 20°C)
C_ALK	VSV	C_{ALK}	mol/m^3	-	conc. calculated by prog.
C_ALKin	CV	$C_{ALK,in}$	mol/m^3	InCALK*varNin	inflow conc. of C_{ALK}
C_I	VSV	C_I	$gCOD/m^3$	-	conc. calculated by prog.
C_Iin	RLV	$C_{I,in}$	$gCOD/m^3$	InCOD*(1-wwDegCOD) *wwDissI*varCODin	inflow conc. of C_I
C_ND	SV	C_{N_D}	gN/m^3	-	conc. calculated by prog.
C_NDin	RLV	$C_{N_D,in}$	gN/m^3	max(InKN*wwDissKN *(1-wwNH4dissKN) *varNin-i_CI*C_Iin,0)	inflow conc. of C_{N_D}
C_NH4	VSV	$C_{NH_4^+}$	gN/m^3	-	conc. calculated by prog.

C_NH4in	RLV	$C_{NH_4^+,in}$	gN/m^3	InKN*wwDissKN *wwNH4dissKN*varNin	inflow conc. of $C_{NH_4^+}$
C_NO3	SV	$C_{NO_3^-}$	gN/m^3	-	conc. calculated by prog.
C_NO3in	RLV	$C_{NO_3^-,in}$	gN/m^3	InCNO3*varNin	inflow conc. of $C_{NO_3^-}$
C_O2	VSV	C_{O_2}	gO/m^3	-	conc. calculated by prog.
C_O2aera	FV	$C_{O_2,aera}$	gO/m^3	2	regulated oxygen conc. in aeration tanks
C_O2sat	FV	$C_{O_2,sat}$	gO/m^3	exp(7.7117-1.31403*log(T+45.93))	oxygen saturation concentration
C_S	VSV	C_S	$gCOD/m^3$	-	conc. calculated by prog.
C_Sin	RLV	$C_{S,in}$	$gCOD/m^3$	InCOD*wwDegCOD *wwDissS*varCODin	inflow conc. of C_S
eta_g	CV	η_g	-	0.8	kinetic parmeter
eta_h	CV	η_h	-	0.4	kinetic parmeter
f_CALK	FV	$f_{C_{ALK}}$	$^oC^{-1}$	C_ALK/(0.01+C_ALK)	factor to be used in rate laws of biochemical processes to prevent concentrations C_{ALK} to become negative
f_CNH4	FV	$f_{C_{NH_4^+}}$	-	C_NH4/(0.01+C_NH4)	factor to be used in rate laws of biochemical processes to prevent concentrations $C_{NH_4^+}$ to become negative
f_p	CV	f_p	-	0.08	stoichiometric parameter
InCALK	FV	$C_{ALK,in}$	mol/m^3	5	mean inflow conc.
InCNO3	FV	$C_{NO_3^-,in}$	gN/m^3	1	mean inflow conc.
InCOD	FV	$C_{I,in}$ $+X_{I,in}$ $+C_{S,in}$ $+X_{S,in}$ $+X_{H,in}$ $+X_{A,in}$	$gCOD/m^3$	250	mean inflow concentration of total COD (cf. Fig. 8.10)
InKN	FV	$C_{NH_4^+,in}$ $+C_{N_{Org},in}$ $+i_{S_I}C_{I,in}$ $+X_{N_{Org},in}$ $+i_{X_I}X_{I,in}$ $+i_{X_B}X_{H,in}$ $+i_{X_B}X_{A,in}$	gN/m^3	30	mean inflow concentration of Kjehldahl nitrogen (cf. Fig. 8.10)
InQ	FV	Q_{in}	m^3/d	10000	mean water inflow
i_CI	CV	i_{C_I}	-	0.06	stoichiometric parameter
i_XB	CV	i_{X_B}	-	0.086	stoichiometric parameter
i_XI	CV	i_{X_I}	-	0.04	stoichiometric parameter
i_XP	CV	i_{X_P}	-	0.06	stoichiometric parameter
k_a	FV	k_a (T)	d^{-1}	k_a20 *exp(the_ka*(T-20))	kinetic par. (with temperature dependence)
k_a20	CV	k_a (20^oC)	d^{-1}	0.08	kinetic param. (at 20^oC)

K_exO2	CV	K_{ex,O_2}	1/d	5	aeration coefficient
k_h	FV	k_h (T)	d^{-1}	k_h20 *exp(the_kh*(T-20))	kinetic par. (with temperature dependence)
k_h20	CV	k_h (20°C)	d^{-1}	3	kinetic param. (at 20°C)
K_NH4	CV	$K_{NH_4^+}$	gN/m^3	1	kinetic parameter
K_NO3	CV	$K_{NO_3^-}$	gN/m^3	0.5	kinetic parameter
K_O2A	CV	$K_{O_2,A}$	gO/m^3	0.4	kinetic parameter
K_O2H	CV	$K_{O_2,H}$	gO/m^3	0.2	kinetic parameter
K_S	CV	K_S	$gCOD/m^3$	20	kinetic parameter
K_X	FV	K_X (T)	-	K_X20 *exp(the_KX*(T-20))	kinetic parameter
K_X20	CV	K_X (20°C)	-	0.03	kinetic parameter
mue_A	FV	μ_A (T)	d^{-1}	mue_A20 *exp(the_mueA*(T-20))	kinetic par. (with temperature dependence)
mue_A20	CV	μ_A (20°C)	d^{-1}	0.8	kinetic param. (at 20°C)
mue_H	FV	μ_H (T)	d^{-1}	mue_H20 *exp(the_mueH*(T-20))	kinetic par. (with temperature dependence)
mue_H20	CV	μ_H (20°C)	d^{-1}	6	kinetic param. (at 20°C)
Q	PV	Q	m^3/d	-	makes actual water flow available
Q_in	RLV	Q_{in}	m^3/d	InQ*varQin	water inflow
Q_recirc	FV	Q_{rec}	m^3/d	15000	recirculating water flow
SludgeAge	FV		d	15	system retention time of particulate material
T	RLV	T	°C	15	temperature
t	PV	t	d	-	makes time available
the_bA	CV	θ_{b_A}	$°C^{-1}$	0.0981	temp. dep. coefficient
the_bH	CV	θ_{b_H}	$°C^{-1}$	0.1132	temp. dep. coefficient
the_ka	CV	θ_{k_a}	$°C^{-1}$	0.0693	temp. dep. coefficient
the_kh	CV	θ_{k_h}	$°C^{-1}$	0.1098	temp. dep. coefficient
the_KX	CV	θ_{K_X}	$°C^{-1}$	0.1098	temp. dep. coefficient
the_mueA	CV	θ_{μ_A}	$°C^{-1}$	0.0981	temp. dep. coefficient
the_mueH	CV	θ_{μ_H}	$°C^{-1}$	0.0693	temp. dep. coefficient
varCODin	RLV		-	(data series)	variation of COD input concentration
varNin	RLV		-	(data series)	variation of nitrogen input concentration
varQin	RLV		-	(data series)	variation of water inflow
wwDegCOD	FV		-	0.75	fraction of biodegradable COD on total COD in inflow (cf. Fig. 8.10)
wwDissI	FV		-	0.4	fraction of dissolved inert COD on total inert COD in inflow (cf. Fig. 8.10)

wwDissKN	FV		-	0.666667	fraction of dissolved nitrogen on Kjehldahl nitrogen in inflow (cf. Fig. 8.10)
wwDissS	FV		-	0.2	fraction of dissolved biodegradable COD on total biodegradable COD in inflow (cf. Fig. 8.10)
wwNH4dissKN	FV		-	0.8	fraction of C_{NH_4}+ on dissolved Kjehldahl nitrogen in inflow (cf. Fig. 8.10)
wwXA	FV		-	0.001	fraction of autotrophic bacteria on particulate biodegradable COD in inflow (cf. Fig. 8.10)
wwXH	FV		-	0.01	fraction of heterotrophic bacteria on particulate biodegradable COD in inflow (cf. Fig. 8.10)
X_A	VSV	X_A	gCOD/m^3	-	conc. calculated by prog.
X_Ain	FV	$X_{A,in}$	gCOD/m^3	lnCOD*wwDegCOD *(1-wwDissS)*wwXA *varCODin	inflow conc. of X_A
X_Aini	CV	$X_{A,ini}$	gCOD/m^3	50	initial concentration of X_A
X_H	VSV	X_H	gCOD/m^3	-	conc. calculated by prog.
X_Hin	FV	$X_{H,in}$	gCOD/m^3	lnCOD*wwDegCOD *(1-wwDissS)*wwXH *varCODin	inflow conc. of X_H
X_Hini	CV	$X_{H,ini}$	gCOD/m^3	1500	initial concentration of X_H
X_I	VSV	X_I	gCOD/m^3	-	conc. calculated by prog.
X_Iin	CV	$X_{I,in}$	gCOD/m^3	lnCOD*(1-wwDegCOD) *(1-wwDissI)*varCODin	inflow conc. of X_I
X_Iini	FV	$X_{I,ini}$	gCOD/m^3	1000	initial concentration of X_I
X_ND	VSV	X_{N_D}	gN/m^3	-	conc. calculated by prog.
X_NDin	CV	$X_{N_D,in}$	gN/m^3	max(lnKN *(1-wwDissKN)*varNin -i_XI*X_Iin -i_XB*(X_Ain+X_Hin) ,0)	inflow conc. of X_{N_D}
X_P	VSV	X_P	gCOD/m^3	-	conc. calculated by prog.
X_Pini	FV	$X_{P,ini}$	gCOD/m^3	500	initial concentration of X_P
X_S	VSV	X_S	gCOD/m^3	-	conc. calculated by prog.
X_Sin	RLV	$X_{S,in}$	gCOD/m^3	lnCOD*wwDegCOD *(1-wwDissS) *(1-wwXA-wwXH) *varCODin	inflow conc. of X_S
Y_A	CV	Y_A	-	0.24	stoichiometric parameter
Y_H	CV	Y_H	-	0.67	stoichiometric parameter

A user of this AQUASIM system file has only to specify the mean water inflow InQ, the mean inflow concentrations InCOD, InKN, InCALK and InCNO3, and the sizes of the reactors. The global carbon and nitrogen concentrations are then distributed to the model compounds using typical fractions of municipal waste water. These fractions are defined in the variables wwDegCOD to wwXH and can also be edited, if measurements indicate other values. A small fraction of autotrophic and heterotrophic bacteria is assumed to be contained in the inflow to allow the population to regenerate after a spill out of the organisms in the waste water treatment plant has occured.

Table 8.13 shows the dynamic processes used for the formulation of the process matrix given in Table 8.11 and for aeration of the reactors for biological water treatment. It is assumed that reactor 1 is aerated with a given aeration coefficient K_{ex,O_2}, whereas oxygen within the reactors 2 and 3 is regulated to a constant value of $C_{O_2,aera}$ (= 2 g/m^3). The factors f_CALK and f_CNH4 were introduced into the process rates of AutGroAero, HetGroAero and HetGroAnox to avoid negative alcalinity and ammonium concentration, if one of these substances becomes rate limiting. The rates of HydrolOrg and of HydrolOrgN were formulated in a way that avoids division by zero even in cases, in which the concentrations of X_S and X_H are zero. An additional process for excess sludge removal was introduced. For this process the stoichiometric coefficients were used to describe the rates for the different particulate substances. This makes the description of sludge removal with a single process possible.

The definition of AQUASIM compartments and links used to describe the spatial configuration shown in Fig. 8.9 are given in Tables 8.14 and 8.15, respectively. Three mixed reactor compartments are used to model biological treatment, an additional mixed reactor compartment is used to model the clarifier. The reactors are sequentially coupled by advective links. The advective link between the third reactor and the clarifier has a bifurcation, which models water and sludge recirculation.

Typical daily variation of water flow and of the concentrations of some compounds in the inflow and in all reactors are shown in Fig. 8.11. The elimination of dissolved (C_S) and particulate (X_S) substrate is clearly seen as well as the nitrification of ammonia (C_NH4) to nitrate (C_NO3). Partial denitrification makes nitrate output concentration smaller than ammonium concentration in the inlet.

Table 8.13: Dynamic processes of the IAWPRC activated sludge model no. 1.

Name	Rate	Stoichiometry	Comment
AerationR1	K_exO2*(C_O2sat-C_O2)	C_O2: 1	aeration with given K_{ex,O_2}
AerationR23	10000*(C_O2aera-C_O2)	C_O2: 1	regulation of concentration
Ammonific	k_a*C_ND*X_H	C_NH4: 1 C_ND: -1 C_ALK: 1/14	
AutDecay	b_A*X_A	X_S: 1-f_p X_A: -1 X_P: f_p X_ND: i_XB-f_p*i_XP	
AutGroAero	mue_A*C_NH4/(K_NH4+C_NH4) *C_O2/(K_O2A+C_O2)*X_A*f_CALK	X_A: 1 C_O2: -(4.57-Y_A)/Y_A C_NO3: 1/Y_A C_NH4: -i_XB-1/Y_A C_ALK: -i_XB/14-1/(7*Y_A)	
HetDecay	b_H*X_H	X_S: 1-f_p X_H: -1 X_P: f_p X_ND: i_XB-f_p*i_XP	
HetGroAero	mue_H*C_S/(K_S+C_S) *C_O2/(K_O2H+C_O2) *X_H*f_CNH4*f_CALK	C_S: -1/Y_H X_H: 1 C_O2: -(1-Y_H)/Y_H C_NH4: -i_XB C_ALK: -i_XB/14	
HetGroAnox	mue_H*C_S/(K_S+C_S) *K_O2H/(K_O2H+C_O2) *C_NO3/(K_NO3+C_NO3) *eta_g*X_H*f_CNH4	C_S: -1/Y_H X_H: 1 C_NO3: -(1-Y_H) /(2.86*Y_H) C_NH4: -i_XB C_ALK: (1-Y_H) /(14*2.86*Y_H) -i_XB/14	
HydrolOrg	if X_S>0 then k_h*X_S/(K_X*X_H+X_S) *(C_O2/(K_O2H+C_O2) +eta_h*K_O2H/(K_O2H+C_O2) *C_NO3/(K_NO3+C_NO3)) *X_H else 0 endif	C_S: 1 X_S: -1	rate formulation avoiding division by zero in case of X_S=X_H=0
HydrolOrgN	if X_H>0 then k_h*X_ND/(K_X*X_H+X_S) *(C_O2/(K_O2H+C_O2) +eta_h*K_O2H/(K_O2H+C_O2) *C_NO3/(K_NO3+C_NO3)) *X_H else 0 endif	C_ND: 1 X_ND: -1	rate formulation avoiding division by zero in case of X_S=X_H=0
Sludge-Removal	1/SludgeAge	X_A: -X_A X_H: -X_H X_I: -X_I X_ND: -X_ND X_P: -X_P X_S: -X_S	stoichiometry is used to formulate correct removal rates for different substances

Table 8.14: Compartments used for application of the IAWPRC activated sludge model no. 1 (Type: MRC = mixed reactor compartment).

Name	Type	Property	Unit	Value	Comment
Clarifier	MRC	inflow	m^3/d	0	no water inflow (besides link)
		input fluxes	-	-	no input fluxes (besides link)
		initial cond.	-	-	initial conditions zero
		volume	m^3	4000	fixed volume
Reactor1	MRC	inflow	m^3/d	Q_in	water inflow given by Q_in
		input fluxes	mol/d gCOD/d gN/d gN/d gN/d gCOD/d gCOD/d gCOD/d gCOD/d gN/d gCOD/d	C_ALK: Q_in*C_ALKin C_I: Q_in*C_Iin C_ND: Q_in*C_NDin C_NH4: Q_in*C_NH4in C_NO3: Q_in*C_NO3in C_S: Q_in*C_Sin X_A: Q_in*X_Ain X_H: Q_in*X_Hin X_I: Q_in*X_Iin X_ND: Q_in*X_NDin X_S: Q_in*X_Sin	input fluxes specified as products of water inflow and concentration
		initial cond.	$gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$	X_A: X_Aini X_H: X_Hini X_I: X_Iini X_P: X_Pini	initial conditions of major particulated compounds; other initial conditions are zero
		volume	m^3	2000	constant volume
Reactor2	MRC	inflow	m^3/d	0	no water inflow (besides link)
		input fluxes	-	-	no input fluxes (besides link)
		initial cond.	$gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$	X_A: X_Aini X_H: X_Hini X_I: X_Iini X_P: X_Pini	initial conditions of major particulated compounds; other initial conditions are zero
		volume	m^3	2000	constant volume
Reactor3	MRC	inflow	m^3/d	0	no water inflow (besides link)
		input fluxes	-	-	no input fluxes (besides link)
		initial cond.	$gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$	X_A: X_Aini X_H: X_Hini X_I: X_Iini X_P: X_Pini	initial conditions of major particulated compounds; other initial conditions are zero
		volume	m^3	2000	constant volume

Table 8.15: Advective links used for the configuration for application of the IAWPRC activated sludge model no. 1.

Name	Connect.	Connected to			Comment
Link12	In	outflow of compartment "Reactor1"			connection from Reactor1 to Reactor2
	Out	inflow of compartment "Reactor2"			
Link23	In	outflow of compartment "Reactor2"			connection from Reactor2 to Reactor3
	Out	inflow of compartment "Reactor3"			
Link3C	In	outflow of compartment "Reactor3"			connection from Reactor3 to Clarifier
	Out	inflow of compartment "Clarifier"			
	Recirc	inflow of "Reactor1"			recirculation to Reactor1
		Property	Unit	Value	
		water flow	m^3/d	Q_recirc	water flow Q_recirc
		mass fluxes	mol/d	C_ALK: Q_recirc*C_ALK	dissolved compounds are recirculated according to their current concentration in water (flux = recirculating discharge times concentration) total mass flux of particulate compounds is recirculated (flux = discharge into the link times concentration)
			gCOD/d	C_I: Q_recirc*C_I	
			gN/d	C_ND: Q_recirc*C_ND	
			gN/d	C_NH4: Q_recirc*C_NH4	
			gN/d	C_NO3: Q_recirc*C_NO3	
			gO/d	C_O2: Q_recirc*C_O2	
			gCOD/d	C_S: Q_recirc*C_S	
			gCOD/d	X_A: Q*X_A	
			gCOD/d	X_H: Q*X_H	
			gCOD/d	X_I: Q*X_I	
			gN/d	X_ND: Q*X_ND	
			gCOD/d	X_P: Q*X_P	
			gCOD/d	X_S: Q*X_S	

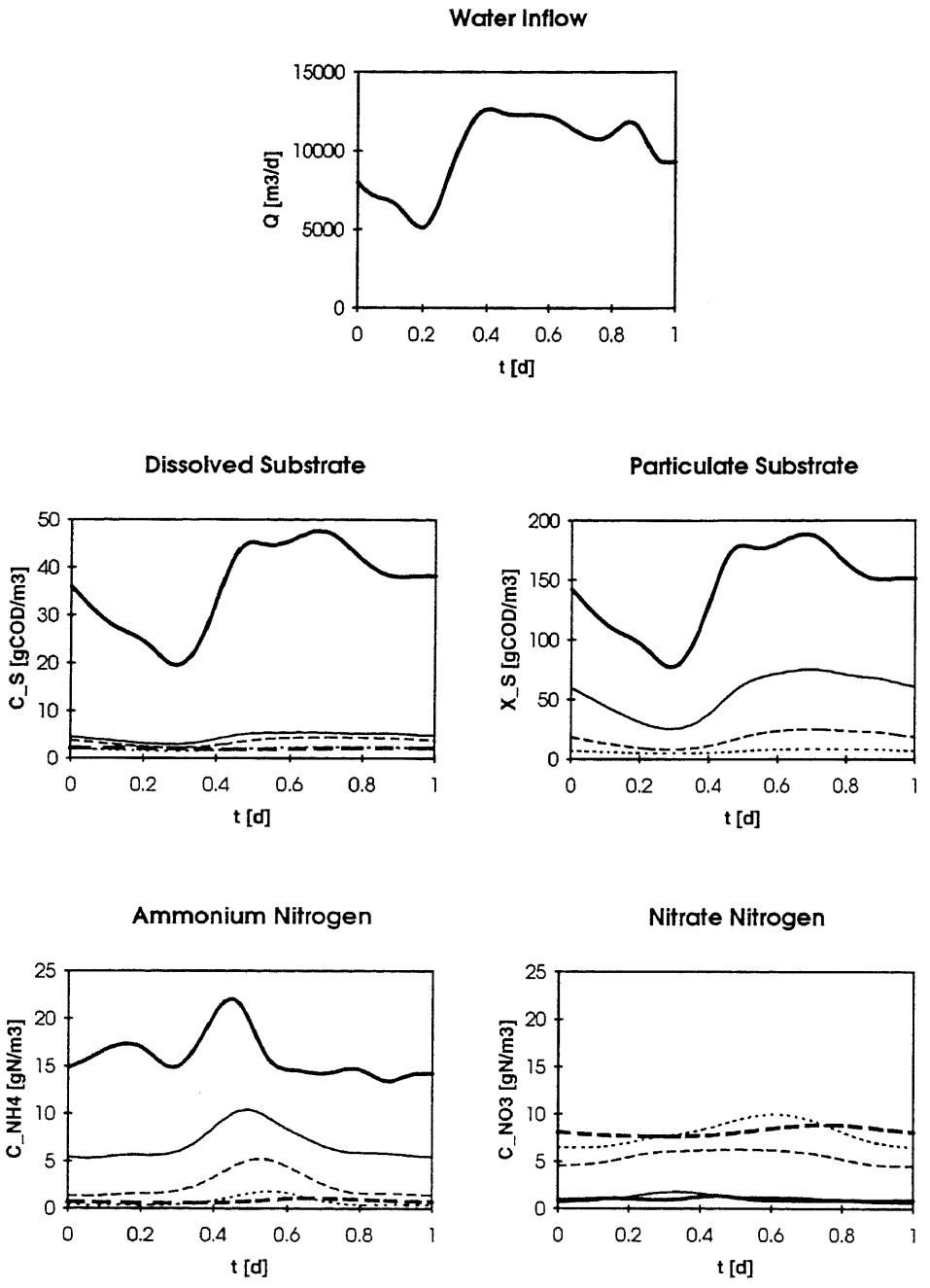


Fig. 8.11: Water inflow and concentrations of C_S, X_S, C_{NH4+} and C_{NO3-} in the inflow (thick, solid), in reactor 1 (thin, solid), in reactor 2 (thin, dashed), in reactor 3 (thin, dotted) and in the outlet of the clarifier (thick, dashed).

8.3 Heterotrophic-Autotrophic Competition in a Biofilm

In order to qualitatively explain systematic changes of the elimination rates of organic substrate and ammonium in a series of biological contactors, Wanner and Gujer (1984) studied competition between heterotrophic and autotrophic bacteria in a biofilm. These calculations are reproduced in this example to demonstrate how biofilm systems can be modelled with AQUASIM.

The biofilm matrix of Wanner and Gujer (1984) is built of active heterotrophic and autotrophic bacteria and inactivated inert biomass. Heterotrophic bacteria grow on organic substrate (acetate), autotrophic bacteria on ammonium. Both growth processes consume oxygen. Besides growth, the processes of endogenous aerobic respiration and inactivation of bacteria are considered. The biochemical model can be summarized as follows:

Compounds:

X_H : heterotrophic biomass concentration (ML^{-3}),
X_A : autotrophic biomass concentration (ML^{-3}),
X_I : concentration of inert particulate material (ML^{-3}),
C_S : dissolved organic substrate concentration (ML^{-3}),
$C_{NH_4^+}$: ammonium concentration (ML^{-3}),
C_{O_2} : dissolved oxygen concentration (ML^{-3}).

Processes:

Heterotrophic growth : growth of heterotrophic biomass with consumption of oxygen and dissolved organic substrate,

Heterotrophic endogenous respiration : endogenous aerobic respiration of heterotrophic organisms,

Heterotrophic inactivation : conversion of active heterotrophic biomass to inert particulate material inde-

Table 8.16: Process matrix of the heterotrophic-autotrophic competition model.

Name	Variables						Rate
	X_H	X_A	X_I	C_S	$C_{NH_4^+}$	C_{O_2}	
Heterotrophic growth	1			$-\frac{1}{Y_H}$		$-\frac{\alpha_H - Y_H}{Y_H}$	$\mu_H \frac{C_{O_2}}{K_{O_2,H}+C_{O_2}} \frac{C_S}{K_S+C_S} X_H$
Heterotrophic endogenous respiration	-1					-1	$b_H \frac{C_{O_2}}{K_{O_2,H}+C_{O_2}} X_H$
Heterotrophic inactivation	-1		1				$k_H X_{HET}$
Autotrophic growth		1			$-\frac{1}{Y_A}$	$-\frac{\alpha_A - Y_A}{Y_A}$	$\mu_A \frac{C_{O_2}}{K_{O_2,A}+C_{O_2}} \frac{C_{NH_4^+}}{K_{NH_4^+}+C_{NH_4^+}} X_A$
Autotrophic endogenous respiration		-1				-1	$b_A \frac{C_{O_2}}{K_{O_2,A}+C_{O_2}} X_A$
Autotrophic inactivation		-1	1				$k_A X_A$

pendent of oxygen,

Autotrophic growth : growth of autotrophic biomass with consumption of oxygen and ammonium,

Autotrophic endogenous respiration : endogenous aerobic respiration of autotrophic organisms,

Autotrophic inactivation : conversion of active autotrophic biomass to inert particulate material independent of oxygen.

Stoichiometric Parameters:

Y_H : yield for heterotrophic biomass (-),
Y_A : yield for autotrophic biomass (-),
α_H : conversion factor for oxygen to substrate (-),
α_A : conversion factor for oxygen to ammonium (-).

Kinetic Parameters:

μ_H : maximum specific growth rate for heterotrophic biomass (T^{-1}),
$K_{O_2,H}$: oxygen half-saturation coefficient for heterotrophic biomass (ML^{-3}),
K_S : organic substrate half-saturation coefficient for heterotrophic biomass (ML^{-3}),
b_H : endogeneous respiration coefficient for heterotrophic biomass (T^{-1}),
k_H : inactivation coefficient for heterotrophic biomass (T^{-1}),
μ_A : maximum specific growth rate for autotrophic biomass (T^{-1}),
$K_{O_2,A}$: oxygen half-saturation coefficient for autotrophic biomass (ML^{-3}),
$K_{NH_4^+}$: ammonia half-saturation coefficient for autotrophic biomass (ML^{-3}),
b_A : endogeneous respiration coefficient for autotrophic biomass (T^{-1}),
k_A : inactivation coefficient for autotrophic biomass (T^{-1}).

Table 8.16 shows the process matrix of the biochemical model used to demonstrate the effects of competition between heterotrophic and autotrophic bacteria in the biofilm.

The calculations of Wanner and Gujer (1984) were performed for a biofilm of constant thickness (produced biomass is detatched at the surface of the film) with given bulk fluid concentrations and without including the bulk water volume into consideration. Such a situation can be simulated with the aid of a single biofilm reactor

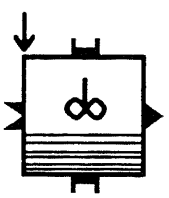


Fig. 8.12: The spatial configuration used for the heterotrophic-autotrophic competition model consists of a single biofilm reactor compartment.

compartment as shown in Fig. 8.12 by specifying a very high discharge through the bulk volume to avoid significant concentration changes due to mass fluxes into the film.

Table 8.17 shows the variables used for the AQUASIM implementation of this model.

Table 8.17: Variables of the heterotrophic autotrophic competition model (Type: VSV = dynamic volume state variable, PV = program variable, CV = constant variable, RLV = real list variable, VLV = variable list variable, FV = formula variable).

Name	Type	Meaning	Unit	Value	Comment
alp_A	CV	α_A	gO/gCOD	4.57	stoichiometric parameter
alp_H	CV	α_H	gO/gCOD	1	stoichiometric parameter
b_A	CV	b_A	d^{-1}	0.05	kinetic parameter
b_H	CV	b_H	d^{-1}	0.2	kinetic parameter
calcnum	PV		-	-	makes calculation number avalable for use in C_Sin)
C_NH4	SV	$C_{NH_4^+}$	gN/m^3	-	conc. calculated by program
C_NH4in	FV	$C_{NH_4^+,in}$	gN/m^3	13	inflow conc. of $C_{NH_4^+}$
C_O2	SV	C_{O_2}	gO/m^3	-	conc. calculated by program
C_O2in	FV	$C_{O_2,in}$	gO/m^3	8	inflow conc. of C_{O_2}
C_S	SV	C_S	$gCOD/m^3$	-	conc. calculated by program
C_Sin	RLV	$C_{S,in}$	$gCOD/m^3$	(0, 3, 13 or 30 depending on calculation number)	inflow conc. of C_S
D_NH4	FV	$D_{NH_4^+}$	m^2/d	$1.86 \cdot 10^{-4}$	diffusion coefficient of $C_{NH_4^+}$
D_O2	FV	D_{O_2}	m^2/d	$2.19 \cdot 10^{-4}$	diffusion coefficient of C_{O_2}
D_S	FV	D_S	m^2/d	$1.04 \cdot 10^{-4}$	diffusion coefficient of C_S
eps_A	FV	ε_A	-	X_A/rho_X	makes volume fraction available for plotting
eps_Aini	FV	$\varepsilon_{A,ini}$	-	0.04	initial volume fraction of X_A
eps_H	FV	ε_H	-	X_H/rho_X	makes volume fraction available for plotting
eps_Hini	FV	$\varepsilon_{H,ini}$	-	0.16	initial volume fraction of X_H
eps_I	FV	ε_I	-	X_I/rho_X	makes volume fraction available for plotting
k_A	CV	k_A	d^{-1}	0.1	kinetic parameter
k_H	CV	k_H	d^{-1}	0.1	kinetic parameter
K_NH4	CV	$K_{NH_4^+}$	gN/m^3	1	kinetic parameter
K_O2A	CV	K_{A,O_2}	gO/m^3	0.1	kinetic parameter
K_O2H	CV	K_{H,O_2}	gO/m^3	0.1	kinetic parameter
K_S	CV	K_S	$gCOD/m^3$	5	kinetic parameter
L_F	PV	L_F	m	-	makes film depth available for formulating initial conditions (cf. Table 8.19)
L_Fini	FV	$L_{F,ini}$	m	$5 \cdot 10^{-4}$	initial condition for L_F
mue_A	CV	μ_A	d^{-1}	0.95	kinetic parameter

mue_H	CV	μ_H	d^{-1}	4.8	kinetic parameter
Q_in	FV	Q_{in}	m^3/d	10	high discharge through bulk volume makes concentrations of dissolved components remain at inflow values
rho_X	FV	ρ_X	$gCOD/m^3$	25000	density of particulate variables
u_F	PV	u_F	m/d	-	makes film velocity available for specifying detachment velocity (cf. Table 8.19)
X_A	SV	X_A	$gCOD/m^3$	-	conc. calculated by program
X_H	SV	X_H	$gCOD/m^3$	-	conc. calculated by program
X_I	SV	X_I	$gCOD/m^3$	-	conc. calculated by program
Y_A	FV	Y_A	-	0.22	stoichiometric parameter
Y_H	FV	Y_H	-	0.4	stoichiometric parameter
z	PV	m	m	-	makes distance from substratum available for formulating initial conditions (cf. Table 8.19)

This list implements in a straightforward manner the variables necessary to formulate the model shown in Table 8.16. Some variables not necessary for model formulation are introduced for making grapical illustration of results possible.

The AQUASIM processes listed in Table 8.18 obviously implement the process matrix shown in Table 8.16.

Table 8.19 shows the AQUASIM implementation of the biofilm reactor compartment. Note that the retention time of bulk water of V_B/Q_{in} is sufficiently short to make the bulk water concentrations of dissolved substances approximately equal to the inflow concentrations. The detachment velocity is chosen to completely compensate growth processes. The case of negative growth velocity is considered by setting the detachment velocity to zero (no detachment if the film shrinks). A rigid solid matrix is assumed and effects of a boundary layer are neglected to make the calculations to

Table 8.18: Dynamic processes of the heterotrophic-autotrophic competition model.

Name	Rate	Stoichiometry	Comment
AutGro	mue_A*C_O2/(K_O2A+C_O2) *C_NH4/(K_NH4+C_NH4)*X_A	X_A: 1 C_O2: -(alp_A-Y_A)/Y_A C_NH4: -1/Y_A	
AutInact	k_A*X_A	X_A: -1 X_I: 1	
AutResp	b_A*C_O2/(K_O2A+C_O2)*X_A	X_A: -1 C_O2: -1	
HetGro	mue_H*C_O2/(K_O2H+C_O2) *C_S/(K_S+C_S)*X_H	X_H: 1 C_S: -1/Y_H C_O2: -(alp_H-Y_H)/Y_H	
HetInact	k_H*X_H	X_H: -1 X_I: 1	
HetResp	b_H*C_O2/(K_O2H+C_O2)*X_H	X_H: -1 C_O2: -1	

Table 8.19: Compartment for application of the heterotrophic autotrophic competition model (Type: BRC = biofilm reactor compartment).

Name	Type	Property	Unit	Value	Comment
Reactor	BRC	inflow	m^3/d	Q_in	
		input fluxes	gN/d gO/d gCOD/d	C_NH4: Q_in*C_NH4in C_O2: Q_in*C_O2in C_S: Q_in*C_Sin	
		initial cond.	m $gCOD/m^3$ $gCOD/m^3$	L_F(BF): L_Fini X_A(BF): rho_X*eps_Aini X_H(BF): rho_X*eps_Hini	initial film thickness BF = biofilm zone BF = biofilm zone
		part. comp.	$gCOD/m^3$ m/d m/d d/m m^2/d $gCOD/m^3$ m/d m/d d/m m^2/d $gCOD/m^3$ m/d m/d d/m m^2/d	X_A: density: rho_X att. coeff.: 0 det. coeff.: 0 layer resist.: 0 diffusivity: 0 X_H: density: rho_X att. coeff.: 0 det. coeff.: 0 layer resist.: 0 diffusivity: 0 X_I: density: rho_X att. coeff.: 0 det. coeff.: 0 layer resist.: 0 diffusivity: 0	 no attachment not used; global det. no boundary layer not used; rigid matrix no attachment not used; global det. no boundary layer not used; rigid matrix no attachment not used; global det. no boundary layer not used; rigid matrix
		diss. comp.	d/m m^2/d d/m m^2/d d/m m^2/d	C_S: layer resist.: 0 diffusivity: D_S C_NH4: layer resist.: 0 diffusivity: D_NH4 C_O2: layer resist.: 0 diffusivity: D_O2	no boundary layer molecular diffusion no boundary layer molecular diffusion no boundary layer molecular diffusion
		reactor type		unconfined	
		bulk volume	m^3	$1 \cdot 10^{-4}$	
		biofilm matrix		rigid	no matrix diffusion
		detach. vel.	m/d	if u_F>0 the u_F else 0 endif	global det. velocity compensates growth
		film surface	m^2	0.1	panar geometry (cf. equation (4.41))
		rate epsFl	d^{-1}	0	constant $\varepsilon_{l\,F}$
		grid points	-	10 (low resolution)	

be comparable with those of Wanner and Gujer (1984).

Fig. 8.13 shows steady state profiles (after a dynamic calculation of 100 days) of particle volume fractions and dissolved substrate concentrations for four bulk water substrate concentrations. These profiles reproduce Fig. 9 of Wanner and Gujer (1984). Major differences between the model as implemented in AQUASIM (Wanner and Reichert, 1994) and the original model introduced by Wanner and Gujer (1984) were compensated by changes in the model parameters involved (rho_X, D_NH4,

D_O2, D_S), whereas minor model changes are the cause of small differences between Fig. 8.13 and Fig. 9 of Wanner and Gujer (1984).

Fig. 8.13 clearly shows significant influence of bulk water concentrations on biofilm composition. It is evident that without organic substrate a completely autotrophic film develops (row 1 of Fig. 8.13). For very high substrate feed, due to the higher growth velocity, the autotrophic population is washed out by the heterotrophic bacteria (row 4 of Fig. 8.13). For a range of substrate concentrations in between, coexistence of both populations is possible (rows 2 and 3 of Fig. 8.13). Consult Wanner and Gujer (1984) for a more extensive discussion of this point.

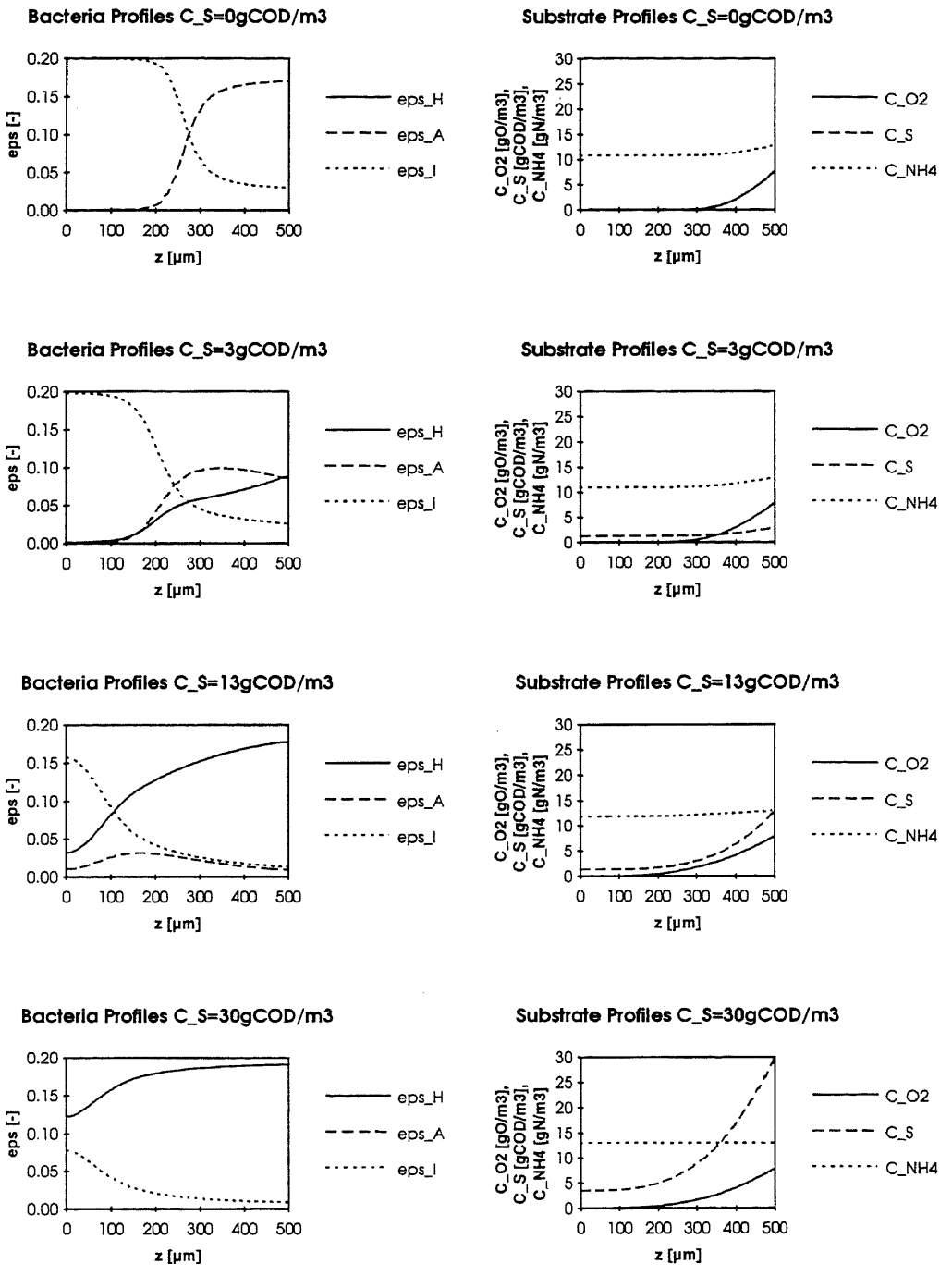


Fig. 8.13: Steady state particle (left) and substrate profiles (right) in the biofilm for four different substrate bulk water concentrations.

8.4 Xylene Degradation in a Membrane-Bound Biofilm

The problem of aerobic degradation of volatile organic chemicals dissolved in water is that the chemicals may escape to the gas phase necessary for oxygen supply. To overcome this problem, experiments on xylene elimination were performed, in which degradation takes place in a biofilm growing on a membrane, through which oxygen is supplied to the biofilm base, whereas xylene diffuses into the film from the bulk water flowing over the film surface (Debus and Wanner, 1992). In this case, the biofilm acts as an additional diffusion barrier between water and gas phase and prevents xylene from escaping into the gas phase. In order to better understand the processes in the biofilm, a model has been developed to describe this experiment (Debus and Wanner, 1992; Wanner et al. 1994). A simplified version of this model is used to demonstrate, how such an experimental situation can be modelled with AQUASIM.

This example is used to demonstrate in more detail, how the stoichiometric coefficients of the process matrix can be derived. The main path of xylene degradation is a two step process, in which xylene is first converted to methylcatechol, which then is consumed as a substrate by heterotrophic bacteria (Debus and Wanner, 1992). Conversion of xylene (C_8H_{10}) to methylcatechol ($C_7H_8O_2$) can be formulated as follows:

$$2\,C_8H_{10} + 5\,O_2 \longrightarrow 2\,CO_2 + 2\,H_2O + 2\,C_7H_8O_2 \quad . \tag{8.9}$$

Heterotrophic growth on methylcatechol is given by

$$\begin{aligned} &5\,C_7H_8O_2 + 8Y_H\,NH_4^+ + 40(1\text{-}Y_H)\,O_2 \\ &\longrightarrow 5(7\text{-}8Y_H)\,CO_2 + 8Y_H\,H^+ + 4(5\text{-}4Y_H)\,H_2O + 8Y_H\,C_5H_7O_2N \quad , \end{aligned} \tag{8.10}$$

where Y_H is the heterotrophic yield coefficient and $C_5H_7O_2N$ is assumed to represent the mean composition of bacterial biomass (McCarty, 1972). The additional biological processes considered in the model of Debus and Wanner (1992) and Wanner et al. (1994) are endogenous decay

$$C_5H_7O_2N + 5\,O_2 + H^+ \longrightarrow NH_4^+ + 5\,CO_2 + 2\,H_2O \tag{8.11}$$

and inactivation

$$C_5H_7O_2N \longrightarrow C_5H_7O_2N \quad . \tag{8.12}$$

In order to use chemical oxygen demand as units to measure xylene, methylcatechol and biomass, mineralisation of these compounds has to be formulated. Mineralisation of xylene is given as

$$\begin{aligned} &2\,C_8H_{10} + 21\,O_2 \longrightarrow 10\,H_2O + 16\,CO_2 \\ &\text{conversion: } 336\ \text{gCOD/mol Xylene} \quad , \end{aligned} \tag{8.13}$$

mineralization of methylcatechol as

$$C_7H_8O_2 + 8\,O_2 \longrightarrow 4\,H_2O + 7\,CO_2 \qquad (8.14)$$

conversion: 256 gCOD/mol Methylcatechol

and mineralization of biomass as

$$C_5H_7O_2N + 5\,O_2 + H^+ \longrightarrow NH_4^+ + 2\,H_2O + 5\,CO_2 \qquad (8.15)$$

conversion: 160 gCOD/"mol" bacterial biomass .

These equations make it possible to determine the stoichiometric coefficients of the four biochemical processes taken into account.

The model can be summarised as follows:

Compounds:

X_H : heterotrophic biomass concentration (ML^{-3}),
X_I : concentration of inert particulate material (ML^{-3}),
C_{XYL} : xylene concentration (ML^{-3}),
C_{MET} : methylcatechol concentration (ML^{-3}),
C_{O_2} : dissolved oxygen concentration (ML^{-3}).

Processes:

Xylene conversion : conversion of xylene to methylcatechol with consumption of oxygen according to equation (8.9),
Heterotrophic growth : growth of heterotrophic bacteria on methylcatechol with consumption of oxygen according to equation (8.10),
Endogenous decay : endogenous respiration of heterotrophic bacteria with consumption of oxygen according to equation (8.11),
Inactivation : Inactivation of heterotrophic biomass independent of oxygen according to equation (8.12).

Stoichiometric Parameters:

Y_H : yield for heterotrophic biomass (-).

Table 8.20: Process matrix of the xylene degradation model

Name	Variables X_H	X_I	C_{XYL}	C_{MET}	C_{O_2}	Rate
Xylene conversion			-1	$\frac{256}{336}$	$-\frac{80}{336}$	$k_{XYL}\frac{C_{XYL}}{K_{XYL}+C_{XYL}}\frac{C_{O_2}}{K_{O_2,XYL}+C_{O_2}}X_H$
Heterotrophic growth	1			$-\frac{1}{Y_H}$	$-\frac{1-Y_H}{Y_H}$	$\mu_H\frac{C_{MET}}{K_{MET}+C_{MET}}\frac{C_{O_2}}{K_{O_2,MET}+C_{O_2}}X_H$
Endogenous decay	-1				-1	$b_H\frac{C_{O_2}}{K_{O_2}+C_{O_2}}X_H$
Inactivation	-1	1				$k_H X_H$

Kinetic Parameters:

k_{XYL} : maximum specific xylene conversion rate (T^{-1}),
K_{XYL} : half-saturation concentration for xylene (ML^{-3}),
$K_{O_2,XYL}$: half-saturation concentration of xylene conversion for oxygen (ML^{-3}),
μ_H : maximum specific growth rate of heterotrophic bacteria (T^{-1}),
K_{MET} : half-saturation concentration for methylcatechol (ML^{-3}),
$K_{O_2,MET}$: half-saturation concentration of heterotrophic growth on methylcatechol for oxygen (ML^{-3}),
b_H : endogeneous respiration coefficient for heterotrophic biomass (T^{-1}),
K_{O2} : half-saturation concentration of endogenous decay for oxygen (ML^{-3}),
k_H : inactivation coefficient for heterotrophic biomass (T^{-1}).

The process matrix of this model is given in Table 8.20. Note that this matrix is the submatrix of the four most important processes of the process matrix of the more detailed model given in Tables 1 and 2 of Wanner et al. (1994).

The experimental setup described in Debus and Wanner (1992) consists of a stirred, air-tight reactor in which a biofilm is grown on a gas-permeable silicone tubing. Oxygen is supplied as a gas flow through the interior of the tubing, whereas xylene is dissolved in the water flowing through the reactor. This experimental setup can be mapped to the AQUASIM system configuration shown in Fig. 8.14. The interior of the tubing is modelled as a mixed reactor compartment, the volume of the reactor filled with water as a biofilm reactor compartment and the wall of the tubing as a diffusive link permeable for oxygen and xylene.

Table 8.21 lists the variables used for the AQUASIM implementation of this model.

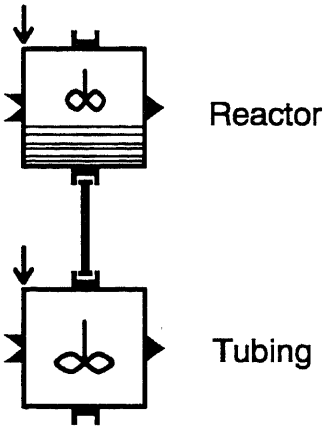


Fig. 8.14: Spatial configuration used for modelling the setup of the xylene degradation experiment described in Debus and Wanner (1992).

Table 8.21: Variables of the xylene degradation model (Type: VSV = dynamic volume state variable, PV = program variable, CV = constant variable, RLV = real list variable, VLV = variable list variable, FV = formula variable).

Name	Type	Meaning	Unit	Value	Comment
A	FV	A	m^2	0.0578	link area
b_H	CV	b_H	d^{-1}	0.66	kinetic parameter
c_de	CV	c_{de}	-	0.2	detachment parameter (cf. Table 8.23)
C_MET	VSV	C_{MET}	$gCOD/m^3$	-	concentration calculated by program
C_O2	VSV	C_{O_2}	gO/m^3	-	concentration calculated by program
C_O2inR	FV	$C_{O_2,in,R}$	gO/m^3	3.5	inflow conc. into reactor of C_{O_2}
C_O2inT	FV	$C_{O_2,in,T}$	gO/m^3	1308	inflow conc. into tubing of C_{O_2}
C_XYL	VSV	C_{XYL}	$gCOD/m^3$	-	concentration calculated by program
C_XYLinR	FV	$C_{XYL,in,R}$	$gCOD/m^3$	200	inflow conc. into reactor of C_{XYL}
D_H	FV	D_H	m^2/d	$1 \cdot 10^{-4}$	diffusion coeff. of X_H (cf. Table 8.23)
D_I	FV	D_I	m^2/d	$1 \cdot 10^{-4}$	diffusion coeff. of X_I (cf. Table 8.23)
D_MET	FV	D_{MET}	m^2/d	$0.8 \cdot 10^{-4}$	diff. coeff. of C_{MET} (cf. Table 8.23)
D_O2	FV	D_{O_2}	m^2/d	$2.1 \cdot 10^{-4}$	diff. coeff. of C_{O_2} (cf. Table 8.23)
D_XYL	FV	D_{XYL}	m^2/d	$0.78 \cdot 10^{-4}$	diff. coeff. of C_{XYL} (vf. Table 8.23)
eps_H	FV	ε_H	-	X_H/rho_H	makes vol. fraction available for plotting
eps_Hini	CV	$\varepsilon_{H,ini}$	-	0.2	initial volume fraction of X_H
eps_I	FV	ε_I	-	X_I/rho_I	makes vol. fraction available for plotting
eps_Iini	CV	$\varepsilon_{I,ini}$	-	0	initial volume fraction of X_I
f	FV	f	-	0.8	reduction of diff. in film (cf. Table 8.23)
H_O2	CV	H_{O_2}	-	32.7	Henry coeff. of C_{O_2} (vf. Table 8.24)
H_XYL	CV	H_{XYL}	-	0.245	Henry coeff. of C_{XYL} (cf. Table 8.24)
k_H	CV	k_H	d^{-1}	0.2	kinetic parameter
K_MET	CV	K_{MET}	$gCOD/m^3$	2.9	kinetic parameter
k_MO2	CV	k_{M,O_2}	m/d	2.57	membrane trans. coeff. (cf. Table 8.24)
k_MXYL	CV	$k_{M,XYL}$	m/d	10	membrane trans. coeff. (cf. Table 8.24)
K_O2	CV	K_{O_2}	gO/m^3	0.11	kinetic parameter
K_O2MET	CV	$K_{O_2,MET}$	gO/m^3	0.13	kinetic parameter
K_O2XYL	CV	$K_{O_2,XYL}$	gO/m^3	0.09	kinetic parameter
K_XYL	CV	K_{XYL}	$gCOD/m^3$	0.5	kinetic parameter
k_XYL	CV	k_{XYL}	d^{-1}	40	kinetic parameter
L_F	PV	L_F	m	-	makes film depth available for formulating initial conditions (cf. Table 8.23)
L_Fini	CV	$L_{F,ini}$	m	$1 \cdot 10^{-6}$	initial condition for L_F
L_L	FV	L_L	m	0.0002	thickness of liquid boundary layer (used in Table 8.23)
mue_H	CV	μ_H	d^{-1}	7	kinetic parameter
Q_inR	FV	$Q_{in,R}$	m^3/d	0.005	water inflow to Reactor
Q_inT	FV	$Q_{in,T}$	m^3/d	0.2	oxygen inflow to Tubing
rho_H	CV	ρ_H	$gCOD/m^3$	64200	density of X_H
rho_I	CV	ρ_I	$gCOD/m^3$	64200	density of X_I

t	PV	t	d	-	makes time available for possible formulation of input time series
u_F	PV	u_F	m/d	-	makes film velocity available for specifying detachment velocity (cf. Table 8.23)
X_H	VSV	X_H	gCOD/m^3	-	concentration calculated by program
X_I	VSV	X_I	gCOD/m^3	-	concentration calculated by program
X_tot	FV	$X_H + X_I$	gCOD/m^3	X_H+X_I	total particule concentration
Y_H	CV	Y_H	-	0.35	stoichiometric parameter
z	PV	z	m	-	makes distance from substratum available for formulating initial conditions (cf. Table 8.23)

The AQUASIM processes listed in Table 8.22 obviously implement the process matrix shown in Table 8.20.

The AQUASIM implementations of the compartments and the link forming the spatial configuration to be modelled, which is shown in Fig. 8.14, are given in Tables 8.23 and 8.24, respectively. Note that the fact, that the tubing is filled with gas instead of water is only relevant for the link, where the inverse of the Henry coefficient has to be specified as a conversion factor for the concentrations in the compartment "Tubing" connected to connection 1 of the link. The default value of this conversion factor of 1 describes diffusion between two compartments filled with water.

Table 8.22: Dynamic processes of the xylene degradation model

Name	Rate	Stoichiometry	Comment
ConvXyl	k_XYL*C_XYL/(K_XYL+C_XYL) *C_O2/(K_O2XYL+C_O2)*X_H	C_XYL: -1 C_MET: 16/21 C_O2: -5/21	
HetDecay	b_H*C_O2/(K_O2+C_O2)*X_H	X_H: -1 C_O2: -1	
HetGro	mue_H*C_MET/(K_MET+C_MET) *C_O2/(K_O2MET+C_O2)*X_H	X_H: 1 C_MET: -1/Y_H C_O2: 1-1/Y_H	
HetInact	k_H*X_H	X_H: -1 X_I: 1	

Table 8.23: Compartments of the xylene degradation model (Types: BRC: = biofilm reactor compartment, MRC = mixed reactor compartment).

Name	Type	Property	Unit	Value	Comment
Reactor	BRC	inflow	m^3/d	Q_inR	
		input fluxes	gCOD/d gO/d	C_XYL: Q_inR*C_XYLinR C_O2: Q_inR*C_O2inR	
		initial cond.	m $gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$ $gCOD/m^3$	L_F(BF): L_Fini X_H(BF): rho_H*eps_Hini X_I(BF): rho_I*eps_Iini C_XYL(BF): C_XYLinR C_XYL(BV): C_XYLinR C_MET(BF): 5 C_MET(BV): 5 C_O2(BF): C_O2inT/H_O2 C_O2(BV): C_O2inT/H_O2	init. thickness. biofilm zone biofilm zone biofilm zone bulk vol. zone biofilm zone bulk vol. zone biofilm zone bulk vol. zone
		part. comp.	$gCOD/m^3$ m/d m/d d/m m^2/d $gCOD/m^3$ m/d m/d d/m m^2/d	X_H: density: rho_H att. coeff.: 0 det. coeff.: 0 layer resist.: L_L/D_H diffusivity: 0 X_I: density: rhoI att. coeff.: 0 det. coeff.: 0 layer resist.: L_L/D_I diffusivity: 0	 no attachment global detachm. resist eq. (4.54) rigid matrix no attachment global detachm. resist eq. (4.54) rigid matrix
		diss. comp.	d/m m^2/d d/m m^2/d d/m m^2/d	C_XYL: layer resist.: L_L/D_XYL diffusivity: f*D_XYL C_MET: layer resist.: L_L/D_MET diffusivity: f*D_MET C_O2: layer resist.: L_L/D_O2 diffusivity: f*D_O2	resist eq. (4.54) reduced diffus. resist. eq. (4.54) reduced diff. resist. eq. (4.54) reduced diff.
		reactor type		confined	
		reactor vol.	m^3	$1.7 \cdot 10^{-3}$	
		biofilm matrix		rigid	no matrix diff.
		detach. vel.	m/d	if u_F>0 the c_de*u_F else 0 endif	global detach. velocity as a fraction of u_F
		film surface	m^2	28.9*(0.002+z)	cyl. geometry (cf. eq. (4.41))
		rate epsFl	d^{-1}	0	constant $\varepsilon_{l,F}$
		grid points	-	12 (low resolution)	
Tubing	MRC	inflow	m^3/d	Q_inT	
		input fluxes	gO/d	C_O2: Q_inT*C_O2inT	
		initial cond.	gO/m^3	C_O2: C_O2inT	
		volume	m^3	$3.25 \cdot 10^{-5}$	

Table 8.24: Diffusive link of the xylene degradation model.

Name	Connection	Connected to			Comment
Outflow	Connect. 1	bulk volume of compartment "Tubing"			diffusive connection between tube interior and biofilm base
	Connect. 2	biofilm base of compartment "Reactor"			
		Property	Unit	Value	
		exch. coeff.	m^3/d m^3/d	C_XYL: A*k_MXYL C_O2: A*k_MO2	mass flux according to equations (4.76) to (4.79)
		conv. fact. 1	- -	C_XYL: 1/H_XYL C_O2: 1/H_O2	

Fig. 8.15 shows the results of a simulation of this system with pure oxygen gas supply into the tubing. The plots in the first row show biofilm thickness and oxygen and xylene concentrations in the bulk water phase as functions of time. As long as oxygen can diffuse through the thin film into the bulk volume (up to day 35), xylene is partially degraded in the bulk water volume of the reactor and the film has a small growth rate. After about 35 days, the film is thick enough to prevent oxygen from diffusing through the biofilm. During this phase of the simulation, xylene conversion takes only place in a very thin film layer, where oxygen and xylene are present (cf. left hand plot in the second row of Fig. 8.15). Decreasing gradients of oxygen and xylene caused by increasing film thickness diminish total xylene removal rate and therefore lead to an increase of xylene bulk concentration (cf. right hand plot in the first row of Fig. 8.15). The last plot of Fig. 8.15 shows profiles of heterotrophic bacteria and inert biomass after 20 and 80 days of simulation. The small active layer mentioned above causes significant population profiles over film depth. The significance of these spatial profiles for xylene elimination is discussed extensively in Wanner et al. (1994).

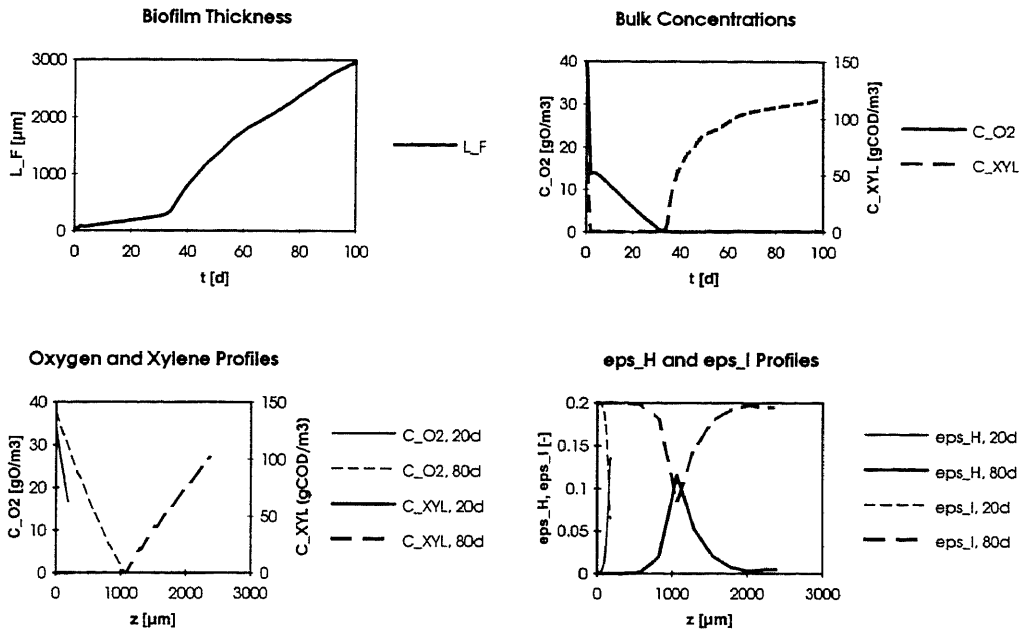


Fig. 8.15: Biofilm thickness and bulk concentrations of xylene and oxygen in the bulk water (first row) and profiles of dissolved and particulate substances after 20 and 80 days of simulation (second row).

8.5 Oxygen Balance in a Heavily Polluted River

The Streeter-Phelps model (Streeter and Phelps, 1925) was the first approach to river water quality modeling. It is well suited as a simple example to demonstrate, how water quality in rivers can be modelled with AQUASIM.

The Streeter-Phelps model treats a situation in which organic pollutants are spilled into a river and in which consumption of oxygen by bacteria degrading these pollutants is the biochemical process dominating the oxygen balance. Oxygen concentration along the river is then determined by oxygen consumtion for degradation of pollutants and by reaeration through the water surface. The model compounds, processes and parameters considered in this model can be summarized as follows:

Compounds:

C_S : organic pollutants measured as biochemical oxygen demand (BOD) (ML^{-3}),
$C_{\Delta O_2}$: oxygen saturation deficit (ML^{-3}).

Processes:

Degradation : degradation of organic pollutants using oxygen,
Reaeration : oxygen transfer from the atmosphere to the river water.

Kinetic Parameters:

K_{deg} : specific degradation rate (T^{-1}),
K_{ex,O_2} : reaeration rate coefficient (T^{-1}).

The process matrix of this model is given in Table 8.25. Both processes are formulated with a rate linear in the concentration of the substance consumed. Degradation consumes organic pollutants and produces oxygen saturation deficit at the same rate (this is the case because both substances, C_S and $C_{\Delta O_2}$, are measured in oxygen units). Reaeration decreases the oxygen saturation deficit.

Table 8.25: Process matrix of the Streeter-Phelps Model.

Name	Variables C_S	Variables $C_{\Delta O_2}$	Rate
Degradation	-1	1	$K_{deg}\, C_S$
Reaeration		-1	$K_{ex,O_2}\, C_{\Delta O_2}$

The biochemical model described above defines the process terms r_i of equation (4.67) describing substance transport and transformation in a river. For each substance, one transport equation has to be solved. These two transport equations are coupled by their process terms:

$$\frac{\partial(AC_S)}{\partial t} = -\frac{\partial(QC_S)}{\partial x} + \frac{\partial}{\partial x}\left(AE\frac{\partial C_S}{\partial x}\right) - A\,K_{deg}\,C_S$$

$$\frac{\partial(AC_{\Delta O_2})}{\partial t} = -\frac{\partial(QC_{\Delta O_2})}{\partial x} + \frac{\partial}{\partial x}\left(AE\frac{\partial C_{\Delta O_2}}{\partial x}\right) + A\,K_{deg}\,C_S - A\,K_{ex,O_2}\,C_{\Delta O_2} \quad . \tag{8.16}$$

These two equations have to be solved simultaneously with one of the equations (4.61a) or (4.61b) describing water flow. In the case of steady flow in a prismatic channel far away from controls, the cross-sectional area A, the discharge Q and the mean velocity

$$v = \frac{Q}{A} \tag{8.17}$$

become constant (given as the solutions of equation (4.58a)). If dispersion can be neglected, equations (8.16) degenerate to

$$\begin{aligned} \frac{\partial C_S}{\partial t} &= -v\frac{\partial C_S}{\partial x} - K_{deg} C_S \\ \frac{\partial C_{\Delta O_2}}{\partial t} &= -v\frac{\partial C_{\Delta O_2}}{\partial x} + K_{deg} C_S - K_{ex,O_2} C_{\Delta O_2} \quad . \end{aligned} \tag{8.18}$$

For constant rate coefficients K_{deg} and K_{ex,O_2}, and constant upstream concentrations $C_{S,o}$ and $C_{\Delta O_2,o}$ at x=0, and assuming an infinite length or the river, these equations can be solved analytically. The solution to these equations is given by

$$C_S(x) = C_{S,o} \exp\left(-\frac{K_{deg}}{v} x\right) \tag{8.19}$$

and

$$C_{\Delta O2}(x) = \begin{cases} \dfrac{K_{deg} C_{S,o}}{K_{ex,O_2} - K_{deg}} \exp\left(-\dfrac{K_{deg}}{v} x\right) + \left(C_{\Delta O_2,o} - \dfrac{K_{deg} C_{S,o}}{K_{ex,O_2} - K_{deg}}\right) \exp\left(-\dfrac{K_{ex,O_2}}{v} x\right) & \text{for } K_{deg} \neq K_{ex,O_2} \\ \left(C_{\Delta O_2,o} + \dfrac{K_{deg} C_{S,o}}{v} x\right) \exp\left(-\dfrac{K_{deg}}{v} x\right) \quad . & \text{for } K_{deg} = K_{ex,O_2} \end{cases} \tag{8.20}$$

Fig. 8.16 shows the typical behavior of longitudinal C_S and $C_{\Delta O_2}$ profiles calculated with the analytical solution (8.19) and (8.20) of the equations (8.18). The organic pollutant concentration decreases exponentially due to degradation, whereas the oxygen saturation deficit first increases because of the large initial rate of oxygen consumption and then decreases due to reaeration, when oxygen consumption becomes small.

For the demonstration of an AQUASIM implementation of the Streeter-Phelps model, in order to make the example more realistic, a more complex river geometry is used and K_{ex,O_2} is not longer assumed to be constant. The river reach to be modelled consists of two reaches separated by a weir (cf. Fig. 8.17). To demonstrate different possibilities for the definition of the geometry of the river bed, the first reach is a rectangular channel, whereas the second reach has an irregular geometry, which is approximated by linear interpolation between measured cross-sectional profiles. In

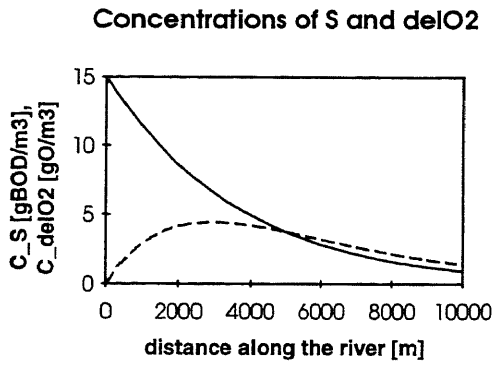


Fig. 8.16: Typical longitudinal profiles of C_S (solid) and $C_{\Delta O_2}$ (dashed) along a river according to the equations (8.19) and (8.20) for v = 0.2 m/s, $K_{deg} = 0.2\ h^{-1}$, $K_{ex,O_2} = 0.3\ h^{-1}$, $C_{S,o} = 15$ gCOD/m^3 and $C_{\Delta O_2,o} = 0$ gO/m^3.

order to be able to describe reaeration along the river and reaeration at the weir with the same reaeration parameter, the reaeration formula of Tsivoglou and Neal (1976), in which reaeration only depends on water level difference, is applied. If the water surface slope is approximated by the friction slope (this is exactly true for the kinematic and diffusive wave equations according to (4.58a) and (4.58b)), K_{ex,O_2} is given by

$$K_{ex,O_2} = c_T\, v\, S_f \quad , \tag{8.21}$$

where S_f (-) is the friction slope, v (LT^{-1}) the mean stream velocity according to (8.17) and c_T (L^{-1}) the reaeration coefficient according to Tsivoglou and Neal (1976). The deficit ratio at the weir (ratio of upstream to downstream oxygen deficit; cf. Gameson, 1957), is then given by

$$r = \exp(c_T\, \Delta z) \quad , \tag{8.22}$$

where Δz is the water level difference across the weir.

Table 8.26 shows the variables used for formulating the Streeter-Phelps model and

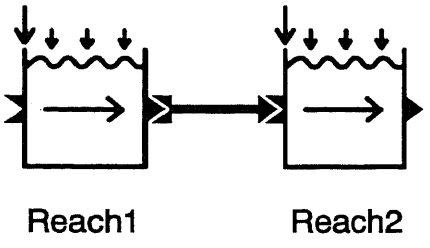


Fig. 8.17: The spatial configuration for applying the Streeter-Phelps model consists of two river reaches.

the river system mentioned above. The variables ending with R1 define geometry, water inflow and friction for reach 1, those ending with R2 do the same for reach 2. In both cases, the geometric quantities cross-sectional area, A (L^2), perimeter length, P (L), and surface width, w (L), have to be formulated as functions of the coordinate along the river, x (L), and of water level elevation, z_o (L). Since section 1 is of rectangular geometry, these functions can be formulated as algebraic expressions in the river section compartment definition (cf. Table 8.28). This is not the case for section 2, in which geometry has to be interpolated between profiles measured at irregular distances along the river. In this case, for each profile, A, P and w has to be converted to a real list with maximum water depth, h_o (L), as the argument. The names of these real lists end with P1R2, P2R2 and P3R2 for profiles 1, 2 and 3, respectively. The variable lists A_R2, P_R2 and w_R2 with argument x interpolate between these profiles along the river.

Table 8.26: Variables of the Streeter-Phelps model (Type: VSV = dynamic volume state variable, PV = program variable, CV = constant variable, RLV = real list variable, VLV = variable list variable, FV = formula variable).

Name	Type	Meaning	Unit	Value	Comment
A	PV	A	m^2	-	makes cross-sectional area available for use in variables d and v and for formulating friction slope (cf. Table 8.28)
A_P1R2	RLV	$A(x_1,z)$	m^2	(list of data pairs)	definition of A for profile 1
A_P2R2	RLV	$A(x_2,z)$	m^2	(list of data pairs)	definition of A for profile 2
A_P3R2	RLV	$A(x_3,z)$	m^2	(list of data pairs)	definition of A for profile 3
A_R2	VLV	A(x,z)	m^2	(list of value-variable pairs)	definition of A in reach 2
C_delO2	VSV	$C_{\Delta O_2}$	gO/m^3	-	conc. calculated by prog.
C_delO2in	FV	$C_{\Delta O_2,in}$	gO/m^3	0	inflow concentration
C_S	VSV	C_S	$gBOD/m^3$	-	conc. calculated by prog.
C_Sin	RLV	$C_{S,in}$	$gBOD/m^3$	15	inflow concentration
c_T	CV	c_T	1/m	0.18	kinetic parameter
d	FV	d	m	A/w	mean water depth
del_z	FV	Δz	m	1.5	height of weir at the end of reach 1
h_0R1	FV	h_o	m	z_0-z_BR1	max. water depth in reach 1
h_0R2	FV	h_o	m	z_0-z_BR2	max. water depth in reach 2
K_deg	CV	K_{deg}	1/h	0.2	kinetic parameter
K_exO2	FV	K_{ex,O_2}	1/h	3600*c_T*v*S_f	equation (8.21) (with conversion of time unit)
K_stR1	CV	K_{st}	$m^{1/3}/s$	20	K_{st} within reach 1
K_stR2	CV	K_{st}	$m^{1/3}/s$	20	K_{st} within reach 2
P	PV	P	m	-	makes perimeter length available for formulating friction slope (cf. Table 8.28)

P_P1R2	RLV	$P(x_1,z)$	m	(list of data pairs)	definition of P for profile 1
P_P2R2	RLV	$P(x_2,z)$	m	(list of data pairs)	definition of P for profile 2
P_P3R2	RLV	$P(x_3,z)$	m	(list of data pairs)	definition of P for profile 3
P_R2	VLV	P(x,z)	m	(list of value-variable pairs)	definition of P in reach 2
Q	FV	Q	m^3/s	Qhour/3600	discharge in more common units than Qh
Qhour	PV	Q	m^3/h	-	makes discharge available for use in Q (in units as used for calculations)
Q_inR1	RLV	Q_{in}	m^3/s	2	inflow to reach 1
Q_inR2	RLV	Q_{in}	m^3/s	0	inflow to reach 2 (in addition to inflow from link)
r	FV	r	-	exp(c_T*del_z)	equation (8.22)
S_0R1	CV	S_o	-	0.0005	definition of S_o in reach 1
S_f	PV	S_f	-	-	makes friction slope available for use in K2
t	PV	t	h	-	makes time available (unit is decided to be hours) for formulating input time series
v	FV	v	m/s	Q/A	equation (8.17)
w	PV	w	m	-	makes surface width available for variable d
w_P1R2	RLV	$w(x_1,z)$	m	(list of data pairs)	definition of w for profile 1
w_P2R2	RLV	$w(x_2,z)$	m	(list of data pairs)	definition of w for profile 2
w_P3R2	RLV	$w(x_3,z)$	m	(list of data pairs)	definition of w for profile 3
w_R1	FV	w(x,z)	m	8	definition of w in reach 1
w_R2	VLV	w(x,z)	m	(list of value-variable pairs)	definition of w in reach 2
x	PV	x	m	-	makes space coordinate along the river available for formulating river geometry
z_0	PV	z_o	m	-	makes water level elavation available for formulating river geometry
z_BR1	FV	z_B	m	-S_0R1*(x-5000)	definition of z_B in reach 1
z_BR2	RLV	z_B	m	(list of data pairs)	definition of z_B in reach 2

Table 8.27 shows the realization of the process matrix given in Table 8.25 as a system of dynamic AQUASIM processes. The correspondence between the two tables is obvious.

Table 8.27: Dynamic processes of the Streeter-Phelps model.

Name	Rate	Stoichiometry	Comment
Degradation	K_deg*C_S	C_S: -1 C_delO2: 1	C_S and C_delO2 are converted with the same rate but opposite sign since both are measured in oxygen units
Reaeration	K_exO2*C_delO2	C_delO2: -1	oxygen deficit is diminished by reaeration

Table 8.28 shows the definition of the two river reaches as two river section compartments. The input of organic pollutants is defined for reach 1. No input of oxygen saturation deficit into reach 1 makes calculation start with an upstream oxygen deficit of zero. Reaeration at the weir is simulated with a negative inflow of oxygen saturation deficit into reach 2. The actual version of the program requires reasonable initial conditions for discharge and water level elevation in order to find the initial conditions. In reach 1, the geometric quantities cross-sectional area, perimeter length and surface width are given as algebraic expressions, whereas in reach 2, the variable list variables prepared for this purpose are used. In both cases, the expression for the friction slope of Strickler is used according to equation (4.64). In reach 1, a fixed end level models regulation by the weir, whereas in reach 2 normal end level simulates an open end of the reach without a hydraulic control.

Table 8.29 completes the system definition by specification of an advective link connecting the two river reaches.

Fig. 8.18 shows longitudinal profiles of some important variables of the model. River bed and water level profiles clearly show the backwater effect of the weir dividing the two river sections at x = 5000 m. Mean velocity demonstrates the significant effect of the weir slowing down water flow. The effect of river geometry on the reaeration coefficient is even more drastic. Finally, degradation of organic pollutants and increase and decrease of oxygen saturation deficit is shown. The effect of the weir in diminishing oxygen exchange in the backwater zone and decreasing oxygen saturation deficit due to weir reaeration is clearly visible.

Table 8.28: Compartments of the Streeter-Phelps model (Type: RSC = river section compartment).

Name	Type	Property	Unit	Value	Comment
Reach1	RSC	inflow	m^3/h	3600*Q_inR1	conversion of time units for inflow
		input fluxes	gCOD/h	C_S: 3600*Q_inR1*C_Sin C_delO2:3600*Q_inR1*C_delO2in	conversion of time units for input flux
		initial cond.	m^3/h m	Qhour: 3600*Q_inR1	reasonable initial conditions required for Q
		start coord.	m	0	reach from 0
		end coord.	m	5000	to 5000 m
		cross sect.	m^2	h_0R1*w_R1	geometry of rect.
		perimeter	m	2*h_0R1+w_R1	channel defined
		surf. width	m	w_R1	with algebraic expr.
		frict. slope	-	1/K_stR1^2*(P/A)^(4/3)*v^2	equation (4.64)
		end level	m	z_BR1+del_z	regulated water level
		method	-	diffusive wave equation	allows to simulate backwater effects due to the weir
		grid points	-	27 (low resolution)	res. into 25 boxes
Reach2	RSC	inflow	m^3/h	3600*Q_inR2	conversion of time units for inflow
		input fluxes	gO/h	C_delO2:-1/r*Qhour*C_delO2	conversion of time units for input flux
		initial cond.	m^3/h m	Qhour: 3600*(Q_inR1+Q_inR2)	reasonable initial conditions required for Q
		start coord.	m	5000	reach from 5000
		end coord.	m	10000	to 10000 m
		cross sect.	m^2	A_R2	geometry defined by
		perimeter	m	P_R2	the variables A_R2,
		surf. width	m	w_R2	P_R2 and w_R2
		frict. slope	-	1/K_stR2^2*(P/A)^(4/3)*v^2	equation (4.64)
		end level	m	normal level	open end of channel
		method	-	diffusive wave equation	backwater effects could be calculated
		grid points	-	27 (low resolution)	res. into 25 boxes

Table 8.29: Advective link of the Streeter-Phelps model.

Name	Connection	Connected to	Comment
Link12	In	outflow or Reach1	connection from reach
	Out	inflow of Reach2	1 to reach 2

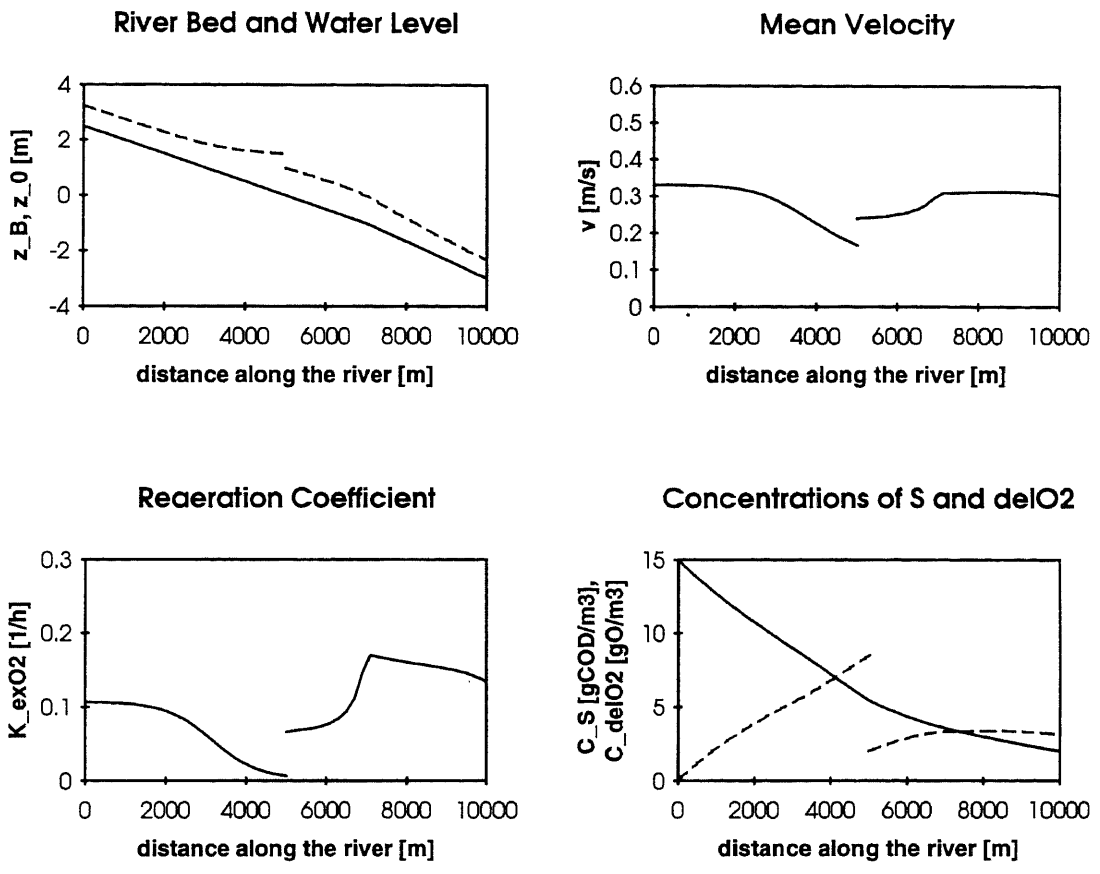


Fig. 8.18: Longitudinal profiles of important variables of the Streeter-Phelps model: River bed and water surface elevation, mean flow velocity, reaeration coefficient and concentrations of organic pollutants (C_S, solid) and oxygen deficit (C_delO2, dashed).

9 Summary and Conclusions

Summary

In this report, the concepts underlying a computer program for the identification and simulation of aquatic systems in nature, in technical plants and in the laboratory are presented. This program offers its users the possibility of formulating their own models, supports them in identifying an adequate model using measured data, and allows them to estimate the uncertainty of calculated results.

As an introduction, in chapter 2, the role of models in the environmental sciences is briefly reviewed. It is pointed out that, due to the difficulty of conducting experiments on well-defined isolated subsystems, the indirect testing of hypotheses concerning functional relationships in natural systems with the aid of mathematical models is even more important in the environmental sciences than in discipline-oriented natural sciences. The use of complex mathematical models, and the impossibility of avoiding simultaneous and uncontrolled changes in external variables, makes the application of system analytical methods for model and parameter identification and for uncertainty analysis necessary. The most important of these methods are briefly reviewed.

For the application of system analytical methods to environmental systems, a computer program is required. In Chapter 3, existing computer programs which can be used for this purpose are classified into three categories. The advantages and disadvantages of the programs belonging to each category are discussed. It is then argued that a more universal program, capable of performing tasks of all three categories, is required in order better to support the application of system analytical methods to environmental systems. The requirements of such a computer program are discussed. Two categories, those of scientific and technical requirements, are distinguished. The main scientific requirements are high flexibility in model formulation and graphical presentation of measured data and calculated results, and the provision of methods for parameter identifiability analysis, parameter estimation and uncertainty analysis. These features make such a program a much more useful tool than conventional environmental simulation programs, which usually only allow their users to perfom simulations with a given model selected by the programmer. The technical requirements are user-friendliness, extensibility, portability and efficiency. User-friendliness is very important if the goal of making the application of system analytical methods more popular for the investigation of environmental systems is to be achieved. In this context, user-friendliness means not only the provision of a graphical user interface, but also the use of a communication "language" familiar to environmental scientists. While user-friendliness is experienced directly by the program user, extensibility and portability are internal program qualities which are necessary for limiting the effort required for program maintenance and extension, and for supporting new computing platforms. Numerical algorithm efficiency is desirable to compensate for the efficiency losses associated with an enhanced flexibility of model formulation, and to make the execution of sensitivity analyses and parameter estimations possible, both of which require a large number of simulations to be performed.

In chapter 4, a general structure for the mathematical modelling of aquatic systems is proposed. Within this modelling framework, an aquatic system is decomposed into four functional subsystems, i.e. variables, processes, compartments and links. Spatial structures are described in terms of compartments, which can be connected by links. In the first program version, the set of compartment types consists of mixed reactors, biofilm reactors and river sections; links can be of advective or of diffusive type. Within the compartments, an arbitrary number of state variables can be defined and the user is free to specify any system of transformation processes. The differential equations solved for the different compartment types are discussed briefly. Both dynamic processes formulated as differential equations and algebraic equilibrium processes are supported. Model formulation is based on variables, which are used for accessing system properties, for providing measured data and for formulating functional relationships using other variables (e.g. for specifying process rates). This chapter provides the foundation for the flexible formulation of models in a "language" familiar to environmental scientists.

The selection of simulation and data analysis techniques available in the first version of the program are discussed in chapter 5. From the discussion in chapter 2, it becomes evident that, in addition to simulations, at least elementary methods of identifiability analysis, parameter estimation and uncertainty analysis must be provided. For practical reasons, it was decided to implement linear sensitivity analysis and parameter covariance estimation in order to help the user in assessing the identifiability of model parameters. Parameter estimation, necessary for the efficient comparison of models, which is required for model structure identification, is implemented using the weighted least-squares technique. It is shown that high flexibility is needed for combining different experiments, universal parameters, parameters specific to individual experiments and several target variables into a single parameter estimation. The uncertainty in the calculated results is estimated with the aid of the linear error propagation formula.

In chapter 6, the numerical algorithms guaranteeing an efficient solution of the equations for temporal evolution, sensitivity analysis and parameter estimation are discussed. It is shown that the solution of partial differential equations by the method of lines makes it possible to combine a general time-discretization method with compartment-specific spatial discretization schemes. Temporal integration of the differential-algebraic system of equations, obtained after spatial discretization, is carried out using the multistep backward differencing technique originally due to Gear, using an implementation of Petzold, which generalizes the original method from pure differential to differential-algebraic systems of equations. Spatial discretization is carried out by employing either a first-order upstream scheme or a conservative high-resolution scheme using flux limiters to avoid oscillations in the numerical solutions. For the biofilm reactor compartment, this technique is combined with a technique using moving grid points. The constrained optimization problem formulated for parameter estimation is solved using two derivative-free algorithms to avoid problems due to the inaccuracy of numerical integration.

In chapter 7, it is demonstrated how object-oriented program design techniques can be used for saving development time and for guaranteeing the extensibility of the

program. After a short general introduction to object-oriented program design, some important highlights of implementation techniques at the technical and conceptual levels are discussed in more detail, and the most important classes of the derivation hierarchy of the program are discussed. The advantage of a program structure consisting of a large portable core program that provides functions for all program tasks, and a relatively thin user-interface layer, which only forwards user input to core program procedures, is discussed. Such a program structure makes the implementation of different user-interface layers possible and reduces the amount of code which becomes useless if a user-interface library is changed.

Finally, in chapter 8, the utility of the program is demonstrated by applying it to five well-known models of environmental and technical systems. These systems are: algal photosynthesis, an activated sludge waste water treatment plant; competition between heterotrophic and autotrophic bacteria in a biofilm; xylene degradation in a membrane-bound biofilm; and oxygen balance in a heavily polluted river according to the Streeter-Phelps model.

The appendix contains the user manual of AQUASIM version 1.0, which has been implemented according to the guidelines developed in this report.

Conclusions

In this report, the concepts underlying a program for the identification and simulation of aquatic systems are discussed. This program combines features of environmental simulation programs, of universal simulation software and of system identification tools. For this reason, it represents a step away from specific environmental simulation programs in the direction of more universally applicable environmental system identification software.

Goals Achieved

The program described in in this report has the following features which make it a much more universal tool for research in the environmental sciences than conventional environmental simulation programs:

- The program calculates water flow and substance transport in various environmental and technical compartments; however, it does not contain a fixed, predefined biochemical transformation model. Program users can define their own transformation models in a very flexible way and can therefore compare simulations performed with different models.
- The program is relatively easy to operate because it uses the graphical user-interface of the machine on which it is installed, and it communicates with the user in a "language" familiar to environmental scientists and engineers. The user can specify a system using "compartments", "links", "processes" and "variables", and it is not necessary to specify differential equations explicitly.

- Due to the combination of the tasks of simulation and model identification (sensitivity analysis, parameter estimation and uncertainty analysis), the program supports scientists in finding "adequate" models for their systems.

For the following reasons, the program is also very suitable for use by students in exercises accompanying lectures on environmental modelling:

- Models can be built up step by step. Simple models can gradually be extended by including additional substances and processes.
- The user is required to enter the uncertainties associated with measured quantities. Based on this information, the program can estimate the uncertainty in the results caused by uncertainties in the input data. Due to the ease with which model formulations can be changed, the program also supports its users in the comparison of different models and therefore in estimating the uncertainty resulting from uncertainty in the model structure.
- The well-structured user interface and the extensive manual (including tutorial exercises) facilitates learning to use the program.

As a result of these features, AQUASIM can be said to belong to a new generation of environmental software.

Limitations in the Universality of the Program

The program is not completely universal. Its most important limitations are the following:

- The transport processes within each compartment are given by the program. The user can influence transport processes only by selecting from various predefined models and by specifying the values of model parameters. This contrasts with the high degree of flexibility available in the formulation of transformation processes. This program design makes AQUASIM an excellent tool for the investigation of transformation processes, whereas its use for the investigation of transport processes is limited to the determination of the parameters of given models. This limitation can be diminished by extending the set of available compartment types, which is intended in future program versions. However, the program will remain a tool designed mainly for the investigation of transformation processes.
- The program concentrates on a class of relatively simple aquatic systems which are thought to be suited for the identification of transformation processes. The main concepts described in this report can be applied to other systems as well. However, the numerical methods employed would not be very efficient for complex two- or three-dimensional systems. This is not a very serious problem, because relatively simple systems are required to achieve the goal of identifying transformation processes.
- Only zero-dimensional links, representing point-to-point connections between compartments, are available in the current program version. An extension to line or surface connections is possible, but would require a considerable further programming effort.

- The program currently supports only elementary statistical data evaluation methods. This limitation is not due to the structure of the program, but is a result of the fact that AQUASIM represents a first attempt at implementing a program which takes a step in the direction of a more universal environmental system identification tool. It is planned to extend the set of available statistical methods in future program versions (see below).

Practical experience has shown, that in spite of these limitations, the program is an extremely useful tool, and represents a large step taken in the direction of higher universality in environmental software. We hope that the publication of the concepts underlying this program will provide an impulse for the development of even more universal programs being based on similar concepts.

Future Developments

The program limitations listed above outline the main directions for future program extensions. The most important of these are are:

- extension of the set of available compartment types,
- inclusion of more statistical data analysis methods.

Additional compartment types could include:

- an advective-diffusive compartment,
- a soil column compartment,
- a lake compartment.

As evident from the discussions in chapters 2 and 5, the statistical techniques most urgently required in the program are:

- nonlinear sensitivity and uncertainty analysis using Monte Carlo simulation,
- robust parameter estimation techniques, which are less sensitive to assumed (normal) distributions of data than the techniques currently employed.

Additional statistical techniques could include:

- an automatic search for systematic deviations between calculated and measured values (e.g. outlier detection and distribution and correlation tests of residuals),
- additional support for model identification (e.g. knowledge-based methods),
- parameter screening and predictions based on Monte Carlo filtering techniques.

To guarantee their practical applicability, all future program extensions will be designed in collaboration with groups evaluating real data. The order in which new features will be implemented will therefore depend on the requirements of experimentally-oriented projects involving the use of AQUASIM.

Appendix

AQUASIM

Computer Program for the Identification and Simulation of Aquatic Systems

Version 1.0

User Manual

June 1994

Contents

A1 Introduction

This manual describes the usage of the program AQUASIM, which was designed for the identification and simulation of aquatic systems in the laboratory, in technical plants and in nature. A survey of the capabilities of the program is also given. It is, however, not the intent of this manual to discuss the equations solved by the program, to describe program design concepts (such as numerical algorithms or object-oriented implementation techniques) or to give examples of applications. All these features are discussed in the main chapters of this report.

Program Design

Comparison of measurements with model calculations is the most important method of testing theories in the natural sciences. Due to the complicated nature of most mathematical models of environmental systems as sets of nonlinear ordinary or partial differential equations, in many cases a computer program is required for performing model calculations. Most programs available for this purpose can be put into one of three categories: universal simulation software; environmental simulation programs; and system identification programs. *Universal simulation software* is very flexible with regard to model formulation, but it is difficult to use, especially for non-specialists. *Environmental simulation programs* are much easier to handle, but they usually implement a specific model selected by the designer of the program. This makes their use for the comparison of different models impossible. Finally, *system identification programs* provide important tools for model comparison and parameter estimation, but the class of models considered in these programs is in most cases restricted to linear or algebraic models, and models cannot be formulated in a way familiar to environmental scientists. Although the classification of simulation programs into these categories is not strict and there are also (a few) programs that cover tasks belonging to two, or even all three, of these categories, a universal identification and simulation program is not yet available. ***The intention behind the design of the program AQUASIM was to provide a more universal identification and simulation tool for a class of aquatic systems important in the environmental sciences. An additional important program design criterion was user-friendliness, which was achieved not only by providing a graphical user interface, but also by utilizing a communication "language" familiar to environmental scientists. AQUASIM is extremely flexible in allowing the user to specify transformation processes, and, in addition to allowing simulations to be performed with the user-specified model, it provides elementary methods for parameter identifiability analysis, for parameter estimation and for uncertainty analysis.*** AQUASIM was developed from 1991-94 in the Computer and System Sciences department of the Swiss Federal Institute for Environmental Science and Technology (EAWAG), CH-8600 Dübendorf, Switzerland. The program was designed mainly for internal use in research and teaching, but is now also available to the public.

Program Tasks

AQUASIM is a program for the identification and simulation of aquatic systems. It performs the four tasks of

- simulation,
- identifiability analysis,
- parameter estimation,
- uncertainty analysis.

Due to the similarity of the mathematical techniques involved, identifiability and uncertainty analyses are combined to yield sensitivity analysis.

The first task of AQUASIM is to allow the user to perform model ***simulations***. By comparing calculated results with measured data, such simulations reveal whether certain model assumptions are compatible with measured data. The existence of systematic deviations between calculations and measurements provides a hint that additional important processes may have to be considered, or corrections made in the way processes are formulated. AQUASIM allows the user to change model structure and parameters values easily.

AQUASIM's second task is to perform ***sensitivity analyses*** with respect to a set of selected variables. This feature allows the user to calculate linear sensitivity functions of arbitrary variables with respect to each of the parameters included in the analysis. These sensitivity functions help in assessing the identifiability of model parameters. Furthermore, sensitivity analysis allows the user to estimate the uncertainty in any variable according to the linear error propagation formula. The calculation of the contribution of each parameter to the total uncertainty facilitates the detection of major sources of uncertainty.

The third important task of AQUASIM is to perform ***parameter estimations*** automatically for a given model structure using measured data. This is not only important for obtaining neutral estimates of parameters, but is also a main prerequisite for efficiently comparing different models. Several calculations with several target variables, as well as universal and calculation-specific parameters, can be combined to yield a single parameter estimation. The quantitative measure of the deviation between model calculations and measurements, which is minimized by the parameter estimation algorithm, is useful for statistically assessing the adequacy of the model.

User Interfaces

Three versions of the program AQUASIM with different user interfaces are provided. The ***window interface version***, `aquasimw`, uses the machines own graphical user interface. The use of this version is strongly recommended for editing models, for defining sensitivity analyses and parameter estimations, for specifying plot definitions, for performing short calculations and for viewing results. The second version is the ***character interface version***, `aquasimc`. This version is intended for users having a

simple terminal without graphical capabilities. It provides all features available in the window interface version except the capability of plotting results directly on the screen (listing results and preparing plots for printing, however, is also possible with this program version). The third version is the ***batch version***, `aquasimb`, which is designed for submitting long calculations as batch jobs (it should be noted that simulations, and especially sensitivity analyses and parameter estimations, may require much computation time). This version allows the user to start a calculation for an AQUASIM system, defined with one of the interactive program versions, by specifying one simple command line. Most operating systems allow several such command lines to be written into a command file, so that calculation and listing or plotting of results can be combined into a single batch job. In the batch version of AQUASIM, models cannot be modified.

Hardware Platforms and Operating Systems

AQUASIM is written in the standardized object oriented programming language C++. The window interface version of the program uses the graphical user interface library XVT (Rochkind, 1989-94; Apiki, 1994), which is available for various hardware platforms and operating systems. ***This program design makes AQUASIM highly portable***. The character interface version and the batch version can be compiled on nearly any platform and operating system without the need of special libraries; the window interface version requires the XVT library for program compilation and linking, but not for program execution. A list of currently supported hardware platforms and operating systems is available on request. Distributed with the program are several pages of platform-specific information including instructions for program installation. The information provided in this manual is valid for all hardware platforms and operating systems.

Organization of this Manual

This manual id dedicated mainly to a description of the window interface version of the program AQUASIM. Sketches of menus and dialog boxes of this version are used to illustrate the text. The exact appearance of these menus and dialog boxes depends on the window system of the machine running the program, which also determines the handling of windows, list boxes, buttons, etc.. In this manual it is assumed that the user is familiar with the window system on his or her own platform. Although this manual is mainly cencerned with the window interface version of the program, it can be used as a reference for the character interface version also, because the menu and dialog box structure is the same for both program versions. The difference between the two versions is that in the window interface version, the user has an overview over all the items in a dialog box and can specify the definitions in any order, whereas in the character interface version the definitions have to be specified in a given order one after the other. The functionality of both versions is the same, with the above-mentioned exception that it is not possible to produce screen plots with the character interface version.

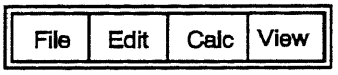


Fig. A1.1:
AQUASIM menu bar.

Fig. A1.1 shows the menu bar of AQUASIM. It contains four menus described in chapters A2 - A5 of this user manual. The menu ***File*** (chapter A2) is used for saving, loading and printing the AQUASIM system, which consists of the mathematical model, measured data, definitions of sensitivity analyses and parameter estimations, plot definitions and calculated states. With the aid of the menu ***Edit*** (chapter A3), the mathematical model and measured data can be entered and edited. The menu ***Calc*** (chapter A4) is used to define and perform simulations, sensitivity analyses and parameter estimations. Finally, the menu ***View*** (chapter A5) is used to specify plot definitions and to list and plot results. Chapter A6 contains a tutorial explaining program usage with a detailed discussion of the solution of six elementary exercises with the aid of AQUASIM. The appendix (chapter A7) contains additional information specific to the different program versions and some hints for troubleshooting.

How to Proceed

The following procedure is recommended for learning to use the program AQUASIM:

1 Read the introduction to this manual (chapter A1) to obtain a general idea of program concepts and capabilities.

2 Skim over chapters A2 - A5 to increase your knowledge of program concepts and to start learning to use the program interface. Look at section A7.1 for keeping track of the program menus. If you plan to work with the character interface version, read section A7.2 also.

3 Study the exercises discussed in the tutorial (chapter A6) thoroughly, and carefully read the corresponding sections of chapters A2 - A5 if you have problems in understanding the solutions.

4 Look at the examples of AQUASIM applications provided as AQUASIM system files, which are documented in chapter A8 of this report. You may concentrate on the example closest to your field of interest. Read section A7.3 describing the batch version of AQUASIM and try to run the examples in batch mode using the command files supplied.

5 Start using the program as a scientific and/or didactic tool. It may be helpful to implement the first model by modifying one of the examples or one of the solutions of the exercises of the tutorial instead of starting from scratch. If problems arise, consult section A7.4.

A2 Manipulating Files

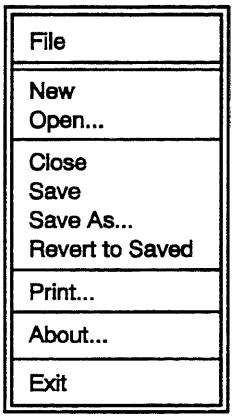


Fig. A2.1: File menu of AQUASIM.

Fig. A2.1 shows the menu **File** of AQUASIM. The first 6 items of this menu are used to save and load AQUASIM systems, consisting of the user-specified mathematical model, measured data, definitions of sensitivity analyses and parameter estimations, plot definitions and calculated states. The item ***New*** is used to free memory from the actual system, to allow the user to enter a new system. With the item ***Open***, a system previously stored with *Save* or *Save As* can be reloaded. Selection of the item ***Close*** results in deletion of the actual system from memory; saved versions are not changed. The item ***Save*** is used to save a system, overwriting its old version on the disk. With the item ***Save As*** the actual system can be saved under a new name. By selecting the item ***Revert to Saved***, the saved version of a system edited interactively can be reloaded. Note, that the items *Close*, *Save*, *Save As* and *Revert to Saved* are inactive, if no system has been opened or interactively entered. For a system interactively entered and not yet saved, the item *Save* results in the same operation as *Save As* and therefore allows to specify a file name. Before specification of a file name by loading or saving an AQUASIM system, the item *Revert to Saved* is inactive. Furthermore, as long as a loaded system is not yet modified, the items *Save* and *Revert to Saved* are inactive. AQUASIM system files should not be edited with other programs, because such an attempt can result in inconsistent or unreadable files. AQUASIM system files can be transfered between all supported platforms using text (ASCII) data transfer. The file format is also compatible with electronic mail; mailed system files can directly be opened on any platform together with their mail headers. To keep a reasonable file size, it is recommended to delete calculated states before saving to an AQUASIM system file, which is planned to be included in a mail message.

To facilitate program portability, the menu item ***Print*** does not directly print the system definitions, but only writes them to a text file, the name of which can be specified by the user. Such text files have then to be submitted to a printer by the user either directly or after loading them into an editor or a text processing program. Printing an AQUASIM system is very useful, because the clear arrangement of all system definitions facilitates checking user input or understanding the meaning of objects loaded from an AQUASIM system file made by someone else. Note, however, that an AQUASIM system cannot be reloaded from a print file.

The menu item ***About*** gives information on the installed program version. This information is also written to any output file written by AQUASIM.

Finally, selection of the menu item ***Exit*** results in program termination. If system definitions have been edited or if states have been calculated or deleted, the user is asked to save the changes.

For each interactive AQUASIM session, a log file with the name `aquasim.log` is written to the startup directory of the program (in the batch version, the name of the log file can be specified by the user). This log file contains information on the progress of calculations. In the case of normal program termination, the log file can be ignored (and deleted); in the case of problems during calculation, the information provided in the log file may help locating the problem. Restart of AQUASIM in the same directory as before, leads to overwriting the old version of the log file, so that only the log files of the most recent sessions (in different directories) are available (unless an old log file has been renamed).

A3 Editing the Model

In the program AQUASIM, a model consists of a system of ordinary and/or partial differential equations and algebraic equations, which deterministically describes the behaviour of a given subset of the observable quantities of an aquatic system. The differential equations for water flow and substance transport can be selected by the choice of environmental compartments, which can be connected by links. The source terms of these equations, which describe the effect of transformation processes, can be freely specified by the user. The definition of such transformation processes follows closely the notation of biochemical processes, as it is familiar to environmental scientists and engineers. The definition of processes, compartments and links is done with the aid of variables, which represent objects taking a possibly context-sensitive numerical value. Fig. A3.1 visualizes the mutual depedences between the four subsystems of variables, processes, compartments and links. It is evident, that the variables form the base subsystem required for the formulation of processes, compartments and links. After a short overview of the four subsystems of AQUASIM model structure, in this chapter, the definition of objects of these subsystems is explained in detail. All these definitions together form the model used by AQUASIM for simulation and analysis of data. The differential equations solved by AQUASIM and examples of AQUASIM applications are given in chapter 4 of this report.

The base subsystem of AQUASIM model structure is the ***system of variables***. Variables are objects which are characterized by the property of taking a numerical value. This value may depend on the values of other variables. Three main categories of variables are distinguished: *System variables* correspond to quantities to be calculated by the program, *data variables* are used to provide measured data and *function variables* can build functional relations using other variables. The system of variables serves as a pool of variables for the formulation of the other subsystems.

The next subsystem of AQUASIM model structure is the ***system of processes***. Two types of processes are distinguished: *Dynamic processes* implement transformation or transfer processes, which are characterized by a common process rate and by individual stoichiometric coefficients describing the relative effect to different variables. Time evolution of variables affected by dynamic processes is determined by the solution of differential equations. The second type of processes are *equilibrium processes*, which determine the value of the corresponding variables by the solution of algebraic equations. Such processes are used to model processes which are so fast, that the corresponding variables can always be approximated to take their actual equilibrium values. The variables of the system of variables may

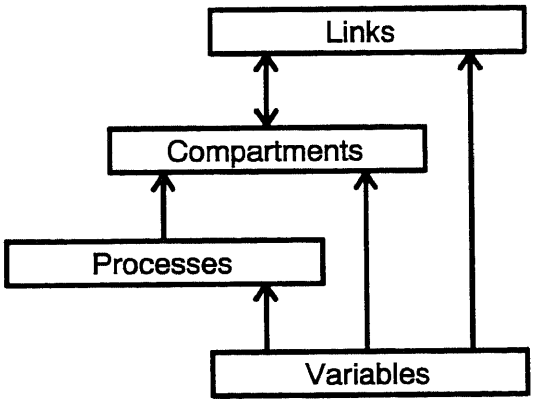


Fig. A3.1: Visualization of mutual dependence of subsystems building AQUASIM model structure.

be used (and are needed) to formulate processes.

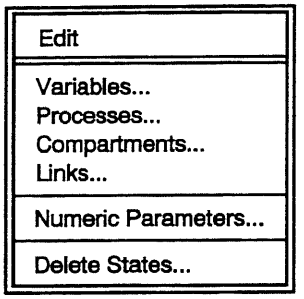


Fig. A3.2: Edit menu of AQUASIM.

The next subsystem of AQUASIM model structure is the ***system of compartments***. This subsystem is designed to spatially divide the system under investigation. The following types of compartments are implemented in the actual version of the program: *Mixed reactor compartments* and *biofilm reactor compartments* are mainly used to describe technical or laboratory systems, whereas *river section compartments* are used to describe natural systems. It is planned to extend this set of compartments in future versions of the program.

The last subsystem of AQUASIM model structure is the ***system of links***. The objects of this subsystem are used to connect the compartments to the desired spatial configuration. To connect the compartments listed above, two types of links are distinguished: *Advective links* describe water flow and advective substance transport between compartments. These links can not only directly connect compartments, but also bifurcations and junctions can be built. *Diffusive links* model diffusive boundary layers or membranes between compartments. These elements can be diffusively penetrated by certain substances.

The menu **Edit** of AQUASIM (cf. Fig. A1.1) shown in Fig. A3.2 bases on the model structure described above. Each of the four menu items *Variables*, *Processes*, *Compartments* and *Links* opens (or activates if it is already open) a modeless dialog box containing a list of objects already defined and controls for defining new objects and for editing and deleting objects of the corresponding subsystem. If the screen is large enough, it is recommended to open all these dialog boxes together to accelerate editing the model. The hierarchy of dialogs controlled by each of these dialog boxes is described in one of the following subsections A3.1 to A3.4. As additional possibilities, the edit menu allows the user to change the values of *Numeric Parameters* and to *Delete States* calculated by the program. The dialog boxes for these editing tasks are described in the sections A3.5 and A3.6, respectively.

A3.1 Variables

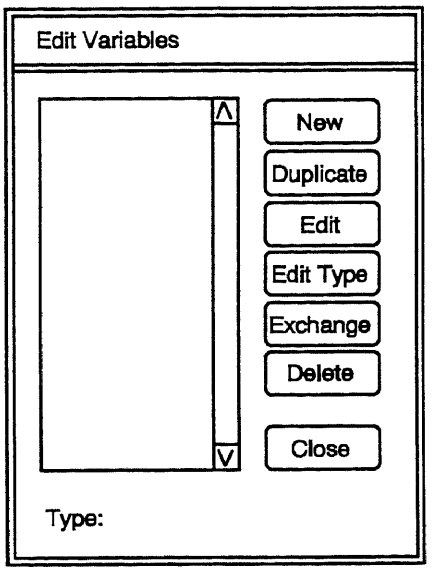


Fig. A3.3: Dialog box for editing the system of variables.

Fig. A3.3 shows the dialog box opened with the edit variables command shown in Fig. A3.2. This dialog box is of modeless type to accelerate editing the system of variables. The names of all variables already defined are listed alphabetically in the list box of this dialog box. The type of the currently selected variable is indicated at the bottom of the dialog box. The buttons of this dialog box allow the user to perform the following operations with variables: ***New*** variables may be created directly or old variables may be ***duplicated*** (in both cases the new variable needs a new name). The data items of a variable can be ***edited*** preserving the type of the variable, but it is also possible to ***edit the type*** of a variable (losing type specific data). Furthermore, it is possible to ***exchange*** two variables in all other variables, processes, compartments, links and definitions of sensitivity analyses and parameter estimations, where they occur as arguments. This feature allows the user to quickly change models without losing data. Finally, it is possible to ***delete*** variables. Deletion of a variable is only possible, if the variable is not an argument of another variable, of a process, of a compartment, of a link or of a definition of a sensitivity analysis or of a parameter estimation. The buttons *Duplicate*, *Edit*, *Edit Type*, *Exchange* and *Delete* are inactive as long as no variable is selected. Pressing the ***Close*** button results in closing of this dialog box. It can be reopened by choosing the edit variables item of the menu shown in Fig. A3.2.

The system of variables defined with the subdialogs assigned to the dialog box shown in Fig. A3.3 serves as a pool of variables for use in the other subsystems. A new variable may depend on any variables already defined (circular references, however, are not allowed). It is important to define all necessary variables before starting to define an object of one of the other subsystems.

A3.1.1 Types of Variables

The dialog box shown in Fig. A3.4 allows the user to select the type of a variable after selecting one of the buttons *New* or *Edit Type* of the dialog box shown in Fig. A3.3. Six types of variables are distinguished. These belong to three categories described in more detail in the following subsections. *State variables* and *program variables* are ***System Variables*** calculated by the program. *Constant variables* and *real list variables* are ***Data Variables***, which correspond to measured quantities. The values and accuracies of data variables have to be provided by the user of the program. *Variable list variables* and *formula variables* are ***Function Variables***, which are used to build functional relations depending on other variables.

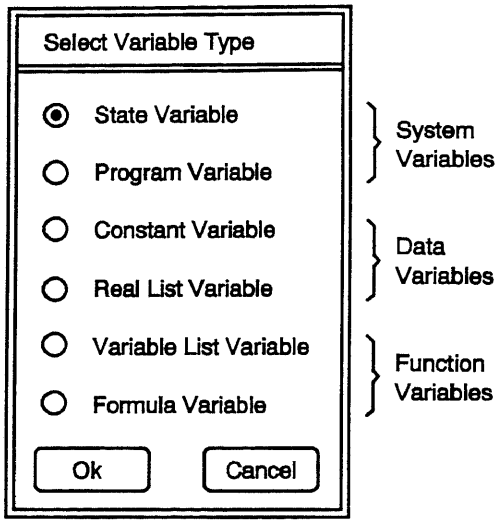


Fig. A3.4: Possible types of variables.

After selecting the *Edit* button of the dialog box shown in Fig. A3.3 or selecting the *New* or *Edit Type* button of this dialog box and specifying the desired variable type with the dialog box shown in Fig. A3.4, a type-specific dialog box appears, which allows the user to define or edit the variable. Fig. A3.5 shows the type independent entries of these dialog boxes. Each variable needs a unique ***Name*** as an identifier. A name of a variable consists of a sequence of letters (A-Z,a-z), digits (0-9) and underline characters (_). The first character may not be a digit. Special characters and blanks are illegal to allow the program to identify variable names in algebraic expressions. For the same reason, the following reserved symbols are not allowed: div, mod, and, or, not, if, then, else, endif, pi, sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, deg, rad, exp, log, ln, log10, sign, abs, sqrt, min, max. To improve documentation of variables, a ***Description*** and a ***Unit*** can be given optionally. In the present version of the program, the units of most variables are only comments, which improve documentation. The user himself is responsible for a consistent use of units. The units of some program variables are used to determine the values of universal constants. Default length unit is meter; the following length units are recognized: mm, cm, dm, m, km. Default time unit is day; the following time units are recognized: s, sec, Sec; m, min, Min; h, std, Std; d, day, days, Tag, Tage. In the present version of the program, the length unit of the program variable "Space Coordinate X" and the time unit of the program variable "Time" are used to calculate the gravitational acceleration needed for

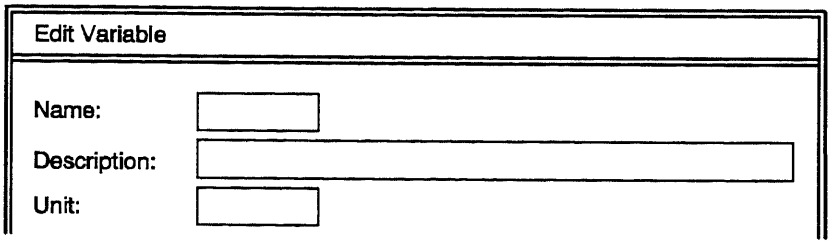


Fig. A3.5: Common entries of edit variable dialog boxes.

the solution of the diffusive wave approximation of one-dimensional open-channel hydraulics in the river section compartment (cf. section A3.3.4).

A3.1.2 System Variables

System variables represent quantities, the actual values of which are provided by the program during the simulation.

A **State Variable** describes a property of water or of a surface in contact with water (e.g. temperature, a concentration of a dissolved or suspended substance or of a substance attached to a surface). Fig. A3.6 shows the required data for the definition of a state variable (refer to section A3.1.1 for a description of the first three items in this dialog box). There are two main types of state variables: The values of ***Dynamic State Variables*** are calculated as solutions of differential equations according to the transport processes determined by the choice of the compartment type and to the transformation rates given by the user. ***Equilibrium State Variables*** are used to describe quantities, the transformation processes of which are much faster than those of other variables, so that they can always be approximated to take the value corresponding to the equilibrium state of their transformation processes. These equilibrium states depend on the values of the other variables and are given as the solution of algebraic equations provided by the user of the program. Dynamic state variables are further divided into ***Dynamic Volume State Variables*** and ***Dynamic Surface State Variables***. Dynamic volume state variables are used to describe concentrations of substances transported with water flow and quantified as mass per unit volume of water, whereas dynamic surface state variables are used to describe substances which are not transported with water flow. Usually, this type of state variables is used to describe substances attached to a surface, which are quantified as total mass, as mass per unit length or as mass per unit area (surface density). The distinction into volume and surface

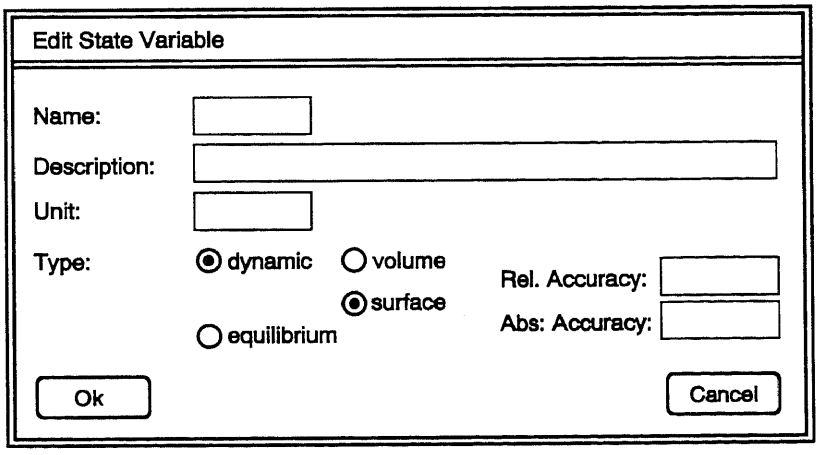


Fig. A3.6: Dialog box for the definition of a state variable.

variables is not needed for equilibrium state variables. Since state variables are calculated as numeric solutions of a system of algebraic and differential equations, ***Relative and Absolute Accuracies*** are needed as further parameters. The integration algorithm uses the absolute accuracy plus the relative accuracy times the actual value as an error criterion to control the size of the time step. Therefore, not both of these accuracies are allowed to be zero, but pure absolute or pure relative error criteria are possible. It is important to specify reasonable values for these accuracies in order to obtain good behaviour of the integration algorithm (in most cases, good behaviour is achieved, if the absolute accuracy plus the product of a reasonable value of the state variable times the relative accuracy is 2 to 6 orders of magnitude smaller than realistic values of the state variable; if the variable can become very small, a significant contribution to total error should be due to absolute accuracy).

The meaning of a state variable is implicitly given by its transformation processes defined by the user. This is in contrast to a **Program Variable**, which refers to a predefined quantity within the program and makes it available for use in the system of variables. Consequently, the user cannot create more than one program variable referring to the same quantity. Fig. A3.7 shows the required data for the definition of a program variable (refer to section A3.1.1 for a description of the first three items in this dialog box). The user can select to which quantity the program variable ***refers to***. Table A3.1 lists the meaning of all program variables considered in the actual version of the program. The program variable "Calculation Number" is a non-negative integer used for distinguishing different calculations (cf. chapter A4), the program variable "Zone Index" is used similarly for distinguishing zones within compartments; all other program variables have a physical meaning. Note, that it depends on the compartment type, which program variables are defined. Program variables always return actual values of the corresponding physical quantity as a function of simulation time and space coordinate within a compartment.

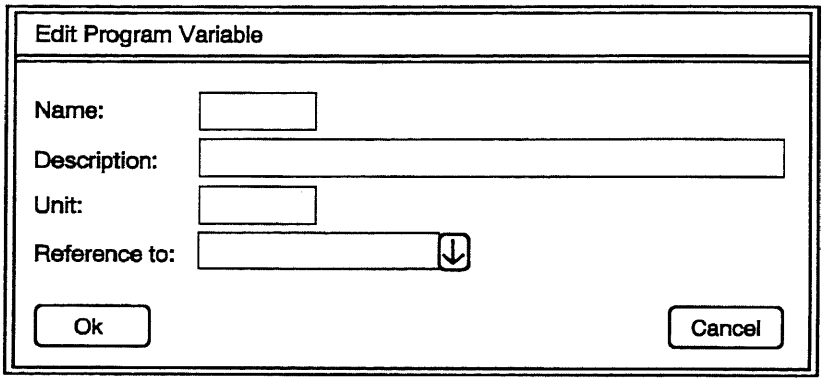


Fig. A3.7: Dialog box for the definition of a program variable.

Table A3.1: Meaning of program variables in different compartments

Program Variable	Mixed Reactor Compartment	Biofilm Reactor Compartment	River Section Compartment
Calculation Number	Identifier for calculations	Identifier for calculations	Identifier for calculations
Zone Index	Identifier for zones: 0: Bulk Volume	Identifier for zones: 0: Bulk Volume 1: Biofilm	Identifier for zones: 0: Water Column

Time	Simulation time	Simulation time	Simulation time
Discharge	Total volumetric inflow into reactor	Total volumetric inflow into reactor	Volumetric flow rate along the river
Water Fraction	Volumetric fraction of water in the reactor = 1	Volumetric fraction of water in the reactor, the total volume of which is filled with water and particles	Volumetric fraction of water in the river = 1
Space Coordinate X	-	-	Space coordinate along the river
Space Coordinate Z	-	Space coordinate in the biofilm, increasing from zero at the biofilm - substratum interface to the thickness of the biofilm	-
Reactor Volume	Total volume of the reactor	Total volume of the reactor including biofilm and bulk water volume	-
Bulk Volume	Total volume of the reactor	Volume of mixed water outside of the biofilm	-
Biofilm Thickness	-	Thickness of the biofilm	-
Growth Velocity of Biofilm	-	Advective velocity of the biofilm solid matrix	-
Interface Velocity of Biofilm	-	Velocity of biofilm - bulk volume interface	-
Detachment Velocity of Biofilm	-	Shrinking velocity of biofilm due to detachment processes	-
Attachment Velocity of Biofilm	-	Growth velocity of biofilm due to attachment processes	-
Water Level Elevation	-	-	Elevation of water level relative to an absolute reference point (e.g. sea level)
Cross Sectional Area	-	-	Area of water body perpendicular to flow direction
Perimeter Length	-	-	Length of the interface between water and river bed perpendicular to flow direction
Surface Width	-	-	Length of the interface between water and the atmosphere perpendicular to flow direction
Friction Slope	-	-	Slope of energy grade line

A3.1.3 Data Variables

Data variables are used to provide measured quantities for use within the system of variables.

A single measured quantity is described by a **Constant Variable**. Fig. A3.8 shows the data required for the definition of a constant variable (refer to section A3.1.1 for a description of the first three items in this dialog box). The user has to specify its ***Value*** which is used for simulations. In sensitivity analyses the ***Standard Deviation*** is used to investigate the influence of uncertainty of model parameters to simulation results. The ***Minimum*** and ***Maximum*** bound the range of legal values. These bounds also hold for internal changes during sensitivity analyses and parameter estimations. For each constant variable, it can be decided, if it is ***Active for Sensitivity Analysis*** and if it is ***Active for Parameter Estimation***. These states can also be accessed with the aid of the dialog boxes shown in Figs. A4.5 and A4.8.

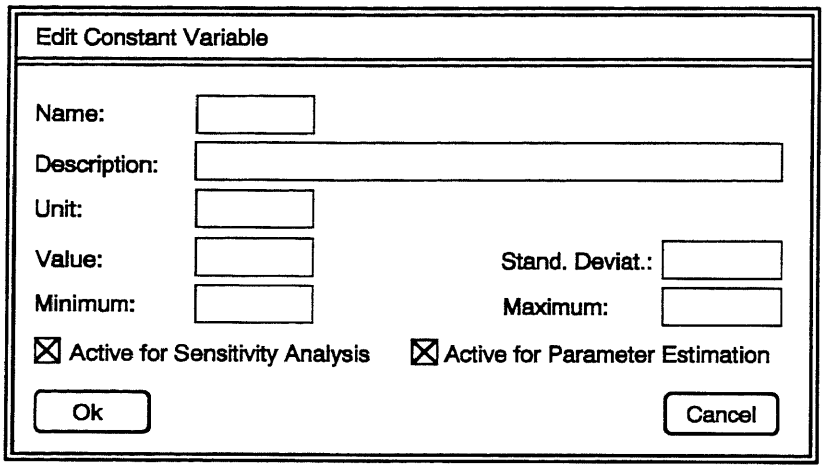


Fig. A3.8: Dialog box for the definition of a constant variable.

Quantities, which are measured as a function of another variable, e.g. time series or spatial profiles, are represented by **Real List Variables**. Fig. A3.9 shows the data required for the definition of a real list variable (refer to section A3.1.1 for a description of the first three items in this dialog box). The ***Argument*** may be any other variable already defined. Standard deviations can either be given as ***Individual Standard Deviations*** for all data values or as ***Global Relative and Absolute Standard Deviations***. In the latter case, the standard deviation of a data value is calculated as the square root of the sum of the square of the absolute standard deviation plus the square of the product of the relative standard deviation times the actual value of the variable. The absolute standard deviation may not be zero if some data elements of the list are zero. As for constant variables, the ***Minimum*** and ***Maximum*** bound the range of legal values. These bounds also hold for internal changes during sensitivity analyses. The list of data is built by elements consisting of a value of the argument, a value of the variable, and, in case of individual standard deviations, the standard deviation of the value. The data pairs are sorted with increasing value of the argument. All data pairs have to differ in their argument.

It is possible to ***add***, ***replace*** and ***delete*** data pairs, to ***read*** them from text files (tab, space or comma delimited; missing lines and values as well as text columns allowed) and to ***write*** them to text files. Fig. A3.10 shows the dialog box used for reading data

pairs from a text file. The user can specify the data area by the ***Start Line*** and ***End Line*** and by the ***Column Numbers of Argument, Value and Standard Deviation*** (Standard deviation only if individual standard deviations are selected in the dialog box shown in Fig. A3.9) Furthermore, the user can choose, if the ***existing data pairs are deleted*** or if the data pairs read from the file are added to existing data.

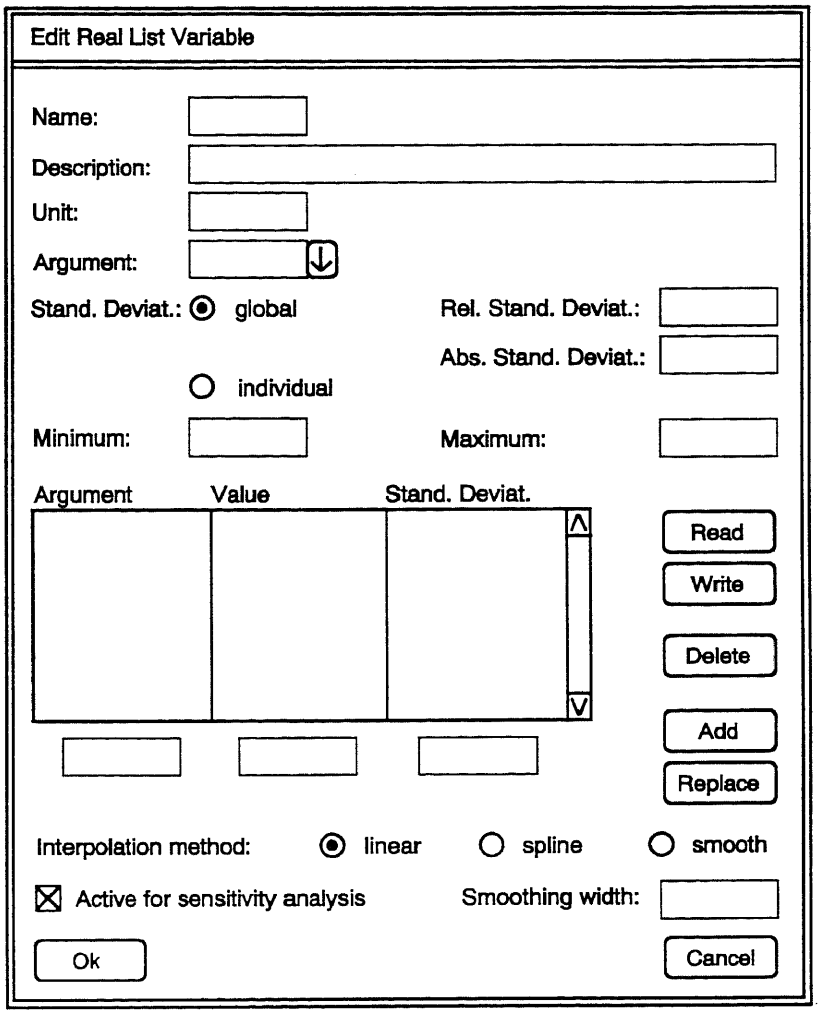


Fig. A3.9: Dialog box for the definition of a real list variable.

Real list variables are evaluated in the following way: As a first step, the argument variable is evaluated. Then, the value of the real list variable is determined from the list of data pairs using one of the following three ***Interpolation Methods***:

- ***Linear interpolation***: If the value of the argument is smaller than the argument of the first list element, the value of the first list element is returned. If the value of the argument is larger than the value of the argument of the last list element, the value of the last list element is returned. If the value of the argument is within the range of the arguments of the list, the value on the connecting straight line between neighbouring data points corresponding to the argument is returned.
- ***Cubic spline interpolation***: If the value of the argument is smaller than the argument of the first list element, the value of the first list element is returned. If the value of the argument is larger than the value of the argument of the last list element, the value of the last list element is returned. If the value of the argument is within the range of the arguments of the list, the interpolated value is given by cubic polynomials between neighbouring data points, which are determined by the conditions of continuous first and second derivatives at inner data points and by zero first derivative at the end points.

- ***Smoothing***: The values are defined by a curve smoothing the data points. This curve is given by the fit of a parabola through neighboring data points. For this fit, the data points are weighted with a normal distribution with a standard deviation chosen by the user (***smoothing width***) and centered at the actual value of the argument. The larger the width of this distribution, the smoother the behavior of the curve.

Fig. A3.11 shows two comparisons of these interpolation and smoothing methods. The second picture in Fig. A3.11 demonstrates, that spline interpolation may lead to undesired oscillations in case of very abrupt changes of data.

As a last point of the definition of a real list variable, as shown in Fig. A3.9, the user has to specify if the real list is ***Active for Sensitivity Analysis***. This state can also be accessed with the aid of the dialog box shown in Fig. A4.5.

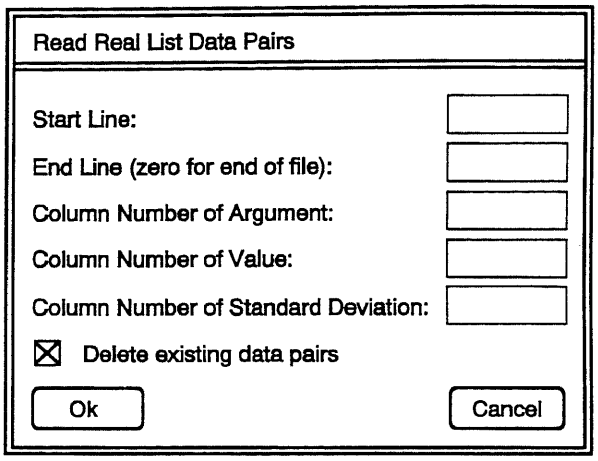


Fig. A3.10: Dialog box for reading data pairs from ASCII files.

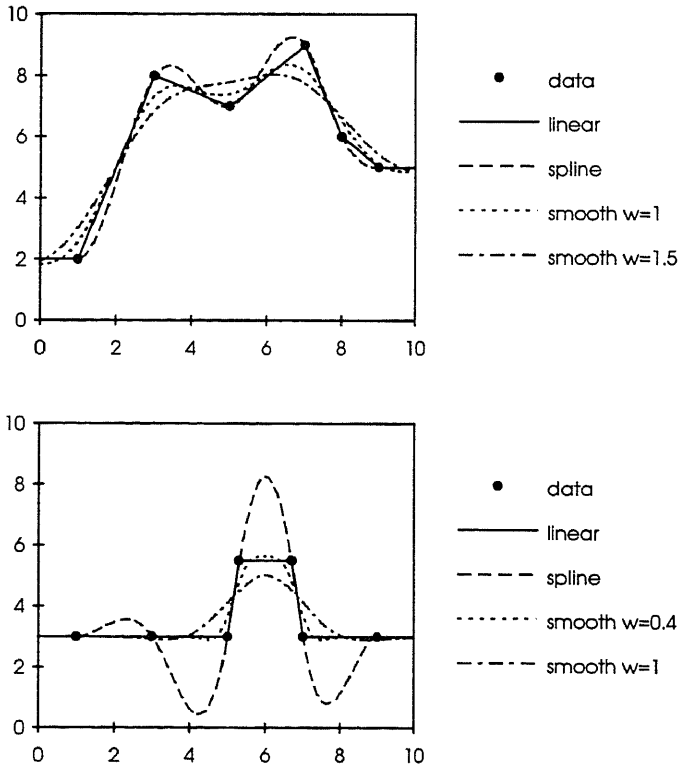


Fig. A3.11: Comparison of interpolation and smoothing methods.

A3.1.4 Function Variables

Function variables allow the user to build functional relations using previously defined variables (cyclic references are not allowed).

Variable List Variables are similar to real list variables, but instead of a value, another variable is given corresponding to each value of the argument. If these variables are variable list variables or real list variables, variable list variables can be used for multidimensional interpolation; if they are constant variables, parameter estimations of time series or of spatial profiles are possible. Fig. A3.12 shows the data required for the definition of a variable list variable (refer to section A3.1.1 for a description of the first three items in this dialog box). For an explanation of the items compare with the description of the real list variable in the preceeding section. Note, that for variable list variables, spline interpolation and smoothing are not very efficient, because these methods need evaluation of all variables of the list for each interpolation.

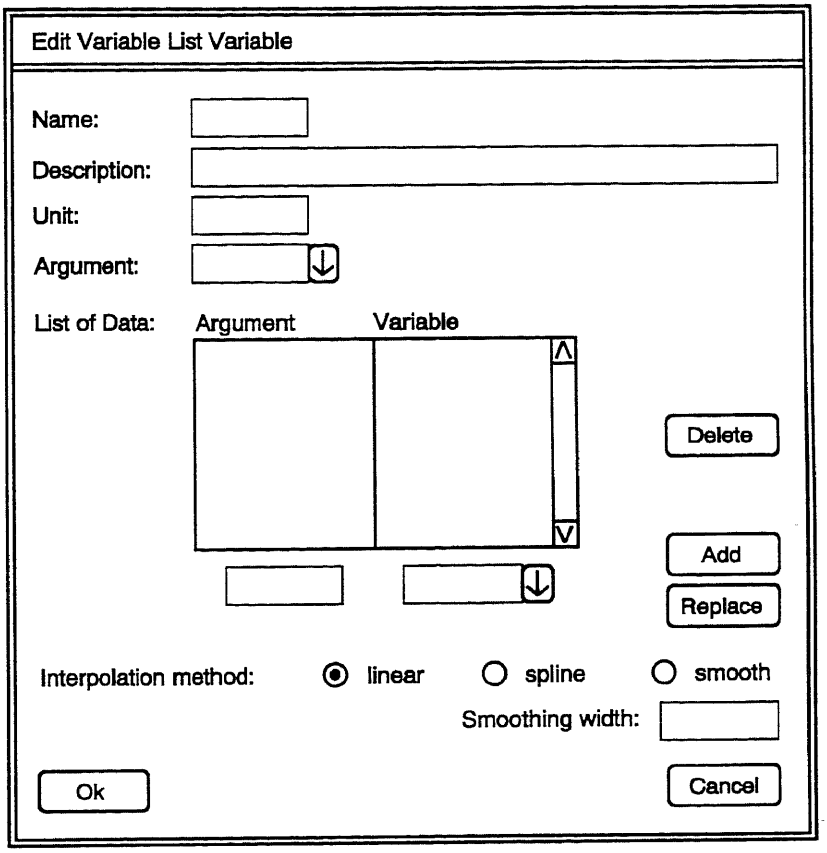


Fig. A3.12: Dialog box for the definition of a variable list variable.

The last, but most versatile variable type is the **Formula Variable**. Fig. A3.13 shows the data required for the definition of a formula variable (refer to section A3.1.1 for a description of the first three items in this dialog box). An algebraic ***Expression*** using the previously defined variables can be given to define the new variable. The formula syntax is given by an <expression> defined as given below, where <varident> has to be the name of a variable already defined:

<varident> = <letter> { <letter_or_digit> }.
<letter> = A | ... | Z | a | ... | z | _ .
<letter_or_digit> = <letter> | <digit>.
<digit> = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.
<expression> = <simple_expression> |

```
                                <simple_expression> <relop> <simple_expression>.
<relop>                 =       = | # | <> | <= | < | >= | >.
<simple_expression>     =       <term> | <sign> <term> |
                                <simple_expression> <addop> <term>.
<term>                  =       <factor> | <term> <mulop> <factor>.
<sign>                  =       + | -.
<addop>                 =       + | - | or.
<factor>                =       <varident> | <unsigned_constant> |
                                ( <expression> ) | <function> | not <factor>.
<unsigned_constant>     =       <unsigned_number> | <predefined_constant>.
<unsigned_number>       =       <unsigned_integer>
                                [.<unsigned_integer> [E [<sign>]
                                <unsigned_integer>]].
<predefined_constant>   =       pi.
<unsigned_integer>      =       <digit> { <digit> }.
<mulop>                 =       ^ | * | / | div | mod | and.
<function>              =       <arith_func_1_arg> |
                                <arith_func_2_arg> |
                                <if_function>.
<arith_func_1_arg>      =       <ident_func_1_arg> ( <expression> ).
<ident_func_1_arg>      =       sin | cos | tan |
                                asin | acos | atan |
                                sinh | cosh | tanh |
                                deg | rad |
                                exp | log | ln | log10 |
                                sign | abs | sqrt.
<arith_func_2_arg>      =       <ident_func_2_arg> ( <expression> , <expression> )
<ident_func_2_arg>      =       min | max.
<if_function>           =       if <condition> then <expression>
                                else <expression> endif.
<condition>             =       <expression>.
```

Note, that this syntax makes it possible to specify algebraic expressions using variables, usual operations and elementary functions, and that even conditional branching with if-then-else-endif constructions is possible. The trigonometric functions use radians as the unit of the argument.

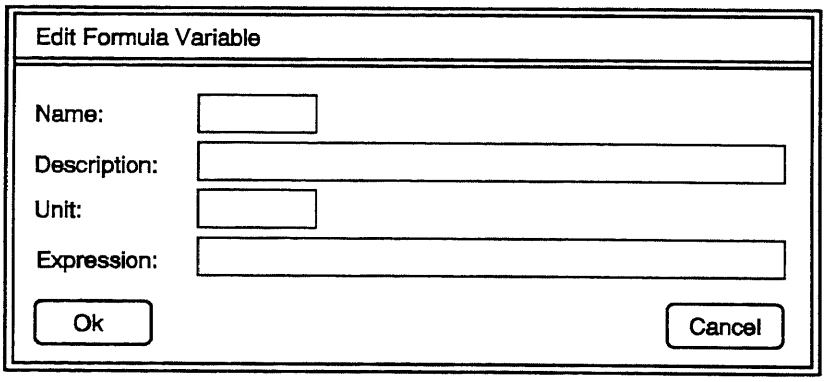


Fig. A3.13: Dialog box for the definition of a formula variable.

A3.2 Processes

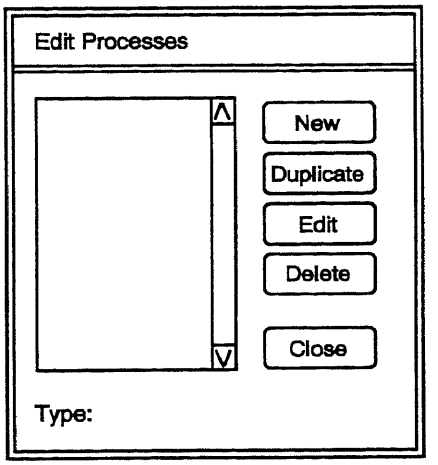


Fig. A3.14: Dialog box for editing the system of processes.

Fig. A3.14 shows the dialog box opened with the edit processes command shown in Fig. A3.2. This dialog box is of modeless type to accelerate editing the system of processes. The names of all processes already defined are listed alphabetically in the list box of this dialog box. The type of the currently selected process is indicated at the bottom of the dialog box. The buttons of this dialog box allow the user to perform the following operations with processes: ***New*** processes may be created directly or old processes can be ***duplicated*** (in both cases the new process needs a new name). It is possible to ***edit*** and to ***delete*** processes. Deletion of a process is only possible, if it is not active within a compartment. The buttons *Duplicate*, *Edit* and *Delete* are inactive as long as no process is selected. Pressing the ***Close*** button results in closing of this dialog box. It can be reopened by choosing the edit processes item of the menu shown in Fig. A3.2.

For the definition of transformation processes with the subdialogs assigned to the dialog box shown in Fig. A3.14, the variables of the system of variables (cf. section A3.1) may be used. The processes of the system of processes serve as a pool for use in the system of compartments: Each process can be activated or inactivated in each compartment (if it does not contain illegal dependences; cf. section A3.3).

A3.2.1 Types of Processes

The dialog box shown in Fig. A3.15 allows the user to select the type of a process after selecting the *New* button of the dialog box shown in Fig. A3.14. Two types of processes are distinguished: ***Dynamic Processes*** implement dynamic transformation and transfer processes as differential equations. ***Equilibrium Processes*** describe very fast processes, for which the corresponding variables can be approximated to always take the values of their equilibrium states, given as the solution of algebraic equations.

After selecting the *Edit* button of the dialog box shown in Fig. A3.14 or selecting the *New* button of this dialog box and specifying the desired process type with the dialog box shown in Fig. A3.15, a type-specific dialog box appears, which allows the user to define or edit the process. Fig. A3.16 shows the type independent entries of these dialog boxes. Each process needs a unique ***Name*** as an identifier. A name of a process consists of a sequence of letters (A-Z,a-z), digits (0-9) and underline characters (_). The first character may not be a digit. To improve documentation of processes, a ***Description*** can be given optionally.

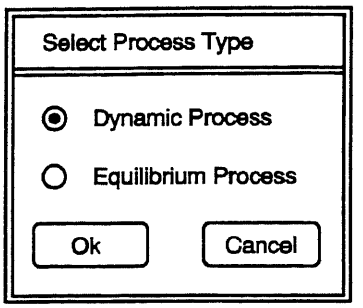


Fig. A3.15: Possible types of processes.

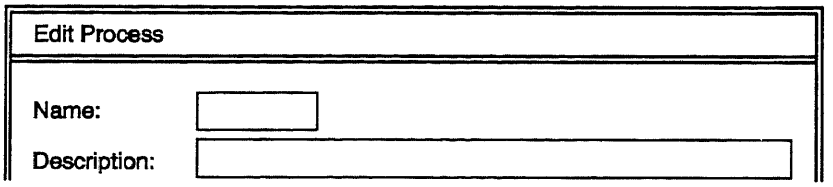


Fig. A3.16: Common entries of edit process dialog boxes.

A3.2.2 Dynamic Processes

Fig. A3.17 shows the data required for the definition of a dynamic process (refer to section A3.2.1 for a description of the first three items in this dialog box). The ***Rate*** contains the common factor of the transformation rates of all variables involved. As shown in Fig. A3.18, for each ***Variable*** involved in the process, an individual ***Stoichiometric Coefficient***, given as an algebraic expression according to the syntax of formula variables (cf. section A3.1) has to be specified. The contribution of the process to the transformation rate of a variable is given as the product of the common rate with the individual stoichiometric coefficient. Note, that splitting of transformation rates into a common process rate and individual stoichiometric coefficients is not unique. Usually, the rate is chosen in such a way, that one of the stoichiometric coefficients becomes unity. During simulations, a dynamic process has only an effect to variables, which are of the type of dynamic state variables. The fact, that in the list of stoichiometric coefficients, any type of variables is allowed, makes it easier to switch between different models (e.g. if variables are changed from calculated dynamic state variables to measured real list variables, the processes have not to be changed).

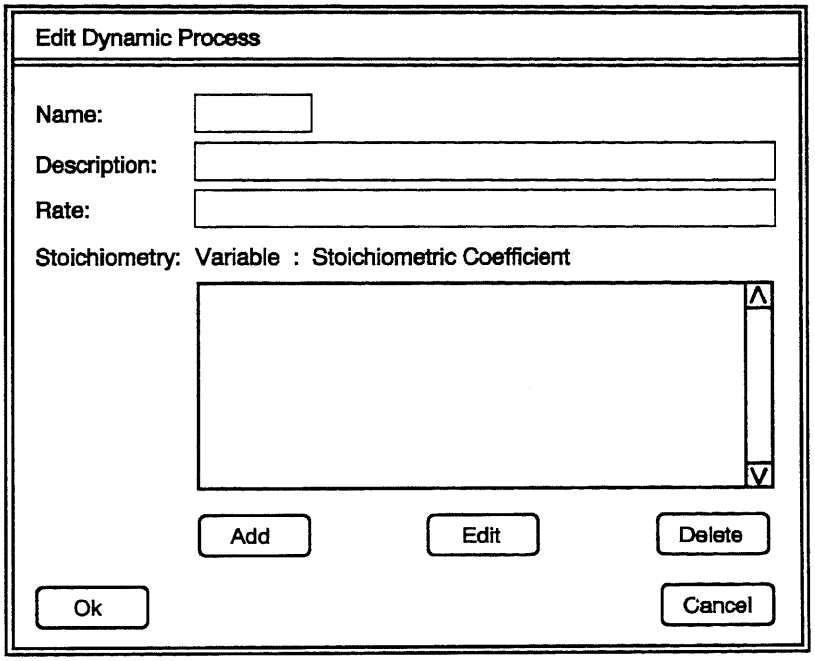


Fig. A3.17: Dialog box for the definition of a dynamic process.

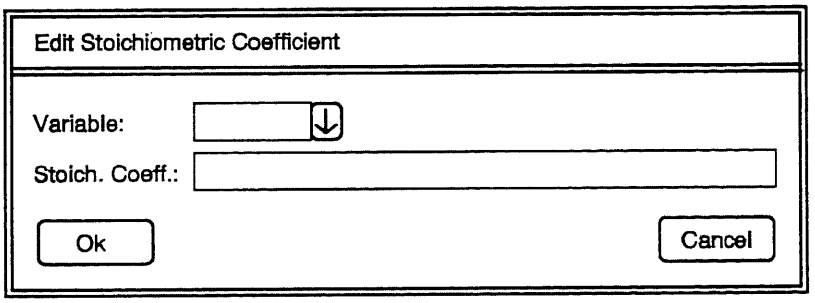


Fig. A3.18: Dialog box the definition of a stoichiometric coefficient.

A3.2.3 Equilibrium Processes

Fig. A3.19 shows the data required for the definition of an equilibrium process (refer to section A3.2.1 for a description of the first three items in this dialog box). For the selected ***Variable***, the ***Equation*** to be solved can be given as an algebraic expression, which is set equal to zero. The syntax of this algebraic expression is the same as that of formula variables described in section A3.1. The variable itself has to be an argument of this expression. In a similar way as in the case of dynamic processes,

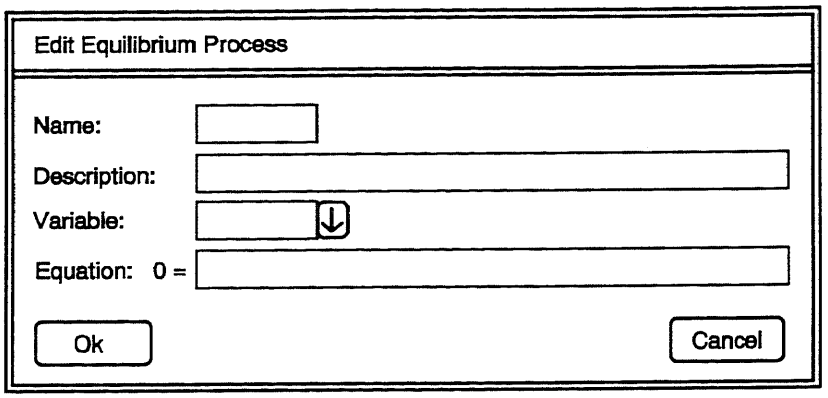


Fig. A3.19: Dialog box for the definition of an equilibrium process.

an equilibrium process has only an effect, if the selected variable is of the type of an equilibrium state variable.

A3.3 Compartments

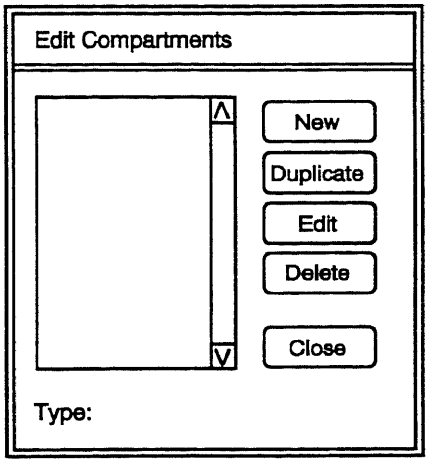


Fig. A3.20: Dialog box for editing the system of compartments.

Fig. A3.20 shows the dialog box opened with the edit compartments command shown in Fig. A3.2. This dialog box is of modeless type to accelerate editing the system of compartments. The names of all compartments already defined are listed alphabetically in the list box of this dialog box. The type of the currently selected compartment is indicated at the bottom of the dialog box. The buttons of this dialog box allow the user to perform the following operations with compartments: ***New*** compartments may be created directly or old compartments can be ***duplicated*** (in both cases the new compartment needs a new name). It is possible to ***edit*** and to ***delete*** compartments. Deletion of a compartment is only possible if it is not an argument of a link or of a definition of a parameter estimation. The buttons *Duplicate*, *Edit* and *Delete* are inactive as long as no compartment is selected. Pressing the ***Close*** button results in closing of the dialog box. It can be reopened by choosing the edit compartments item of the menu shown in Fig. A3.2.

For the definition of compartments with the subdialogs assigned to the dialog box shown in Fig. A3.20, variables of the system of variables (cf. section section A3.1) may be used. Note, however, that variables that depend on program variables which are not defined within the compartment to be edited, are illegal (cf. Table A3.1). Furthermore, there are compartment-specific limitations on dependences of variables that will be discussed in this section. Any of the processes of the system of processes (cf. section A3.2), that do not depend on illegal variables (in the sense defined above), may be activated in any compartment.

A3.3.1 Types of Compartments

Variables and processes have been designed as general as possible (cf. sections A3.1 and A3.2). This is not the case for compartments. The reason for the restriction to specific compartment types is that discretization of partial differential equations is very delicate and needs different methods for different types of compartments (cf. chapter 6 for a discussion of mathematical methods used). The dialog box shown in

Fig. A3.21 allows the user to select the type of a compartment after selecting the button *New* of the dialog box shown in Fig. A3.20. Three types of compartments are considered in the actual version of the program. A ***Mixed Reactor Compartment*** allows to model processes in mixed systems of variable volume, a ***Biofilm Reactor Compartment*** considers diffusive mass transfer limitiations within biofilms growing on a surface within a reactor filled with water and a ***River Section Compartment*** models hydraulics and transport in a river reach. It is planned to extend this set of compartments in future versions of the program.

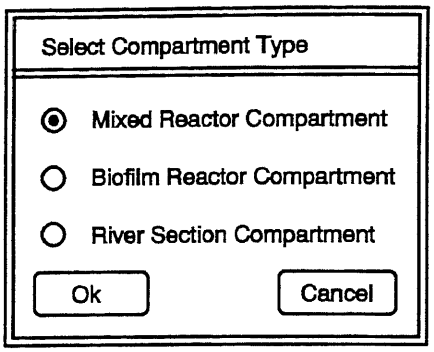


Fig. A3.21: Possible types of compartments.

After selecting the *Edit* button of the dialog box shown in Fig. A3.20 or selecting the *New* button of this dialog box and specifying the desired compartment type with the dialog box shown in Fig. A3.21, a type-specific dialog box appears, which allows the user to define or edit the compartment. Fig. A3.22 shows the type independent entries of these dialog boxes. Each compartment needs a unique ***Name*** as an identifier. A name of a compartment consists of a sequence of letters (A-Z,a-z), digits (0-9) and underline characters (_). The first character may not be a digit. To improve documentation of compartments, a ***Description*** can be given optionally. The four buttons in the ***Options*** line of Fig. A3.22 allow the user to activate state variables and processes and to specify initial conditions and input. These four tasks are described in the following paragraphs.

For each compartment, the user has to specify which state variables are active. This can be done by choosing the ***Variables*** button shown in Fig. A3.22. This action opens the dialog box shown in Fig. A3.23. The two list boxes of this dialog box show the active variables and all available variables, respectively. An inactive variable can be activated by selecting it in the right hand list box and pressing the *Activate* button. The variable is added at the end of the list, if no active variable is selected, otherwise it is inserted before the selected variable (the order of active variables is irrelevant, but a user-defined order may increase clearness of presentation). An active variable can be inactivated by selecting it in the left hand list box and pressing the *Inactivate* button. Note, that the *Activate* button is inactive, as long as no inactive variable is selected in the right hand list box. Similarly, the *Inactivate* button is inactive, if no variable is selected in the left hand list box. The list of active variables may contain any types of variables, but activation and inactivation has only an effect to state variables. Inactive state variables return a value of zero. The reason to allow other variable types within

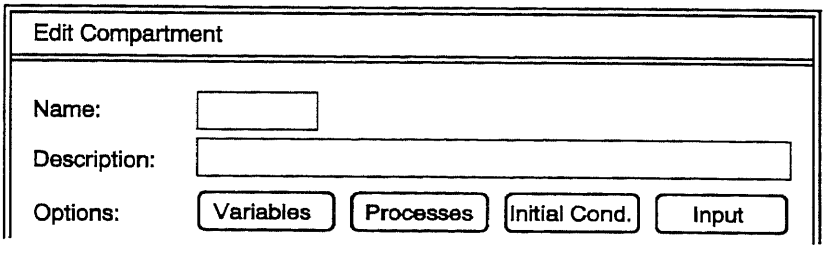


Fig. A3.22: Common entries of edit compartment dialog boxes.

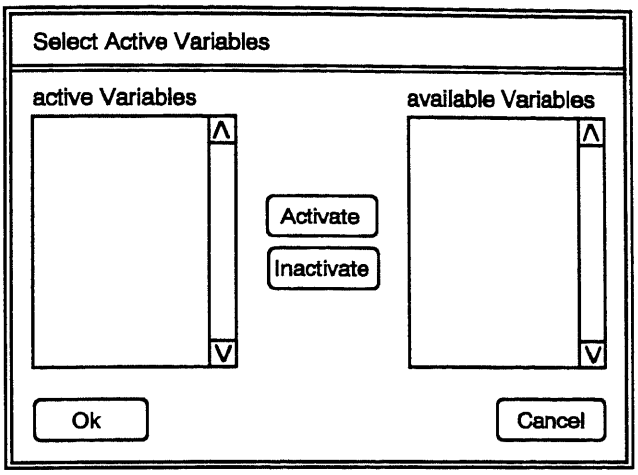


Fig. A3.23: Dialog box for activating variables.

the list of active variables is to facilitate changes of model structure. A user can change the type of a state variable to another variable type, without editing the lists of active variables of the compartments. Active variables which are not state variables are ignored.

As a second point, the user has to select the active processes. Each compartment has its own set of active processes. Activation and inactivation of processes can be done by choosing the ***Processes*** button shown in Fig. A3.22. This action opens the dialog box shown in Fig. A3.24. The two list boxes of this dialog box show the active processes and all available processes, respectively. An inactive process can be activated by selecting it in the right hand list box and pressing the *Activate* button. The process is added at the end of the list, if no active process is selected, otherwise it is inserted before the selected process (the order of active processes is irrelevant, but a user-defined order may increase clearness of presentation). An active process can be inactivated by selecting it in the left hand list box and pressing the *Inactivate* button. Note, that the *Activate* button is inactive, as long as no inactive process is selected in the right hand list box. Similarly, the *Inactivate* button is inactive, if no process is selected in the left hand list box.

The third option common to all compartments, it the definition of initial conditions. This is done by choosing the ***Initial Cond.*** button shown in Fig. A3.22. This action opens the dialog box shown in Fig. A3.25. The initial conditions already defined for various variables are listed in the list box of this dialog. The set of initial conditions can be edited using the buttons *Add*, *Edit* and *Delete* of the dialog box. Selection of *Add* leads to addition of the new initial condition at the end of the list if no initial condition is selected, otherwise the new initial condition is inserted before the selected initial condition (the order of initial conditions is irrelevant, but a user-defined order may increase clearness of presentation). The buttons *Edit* and *Delete* are only active, if an initial condition in the list box is selected.

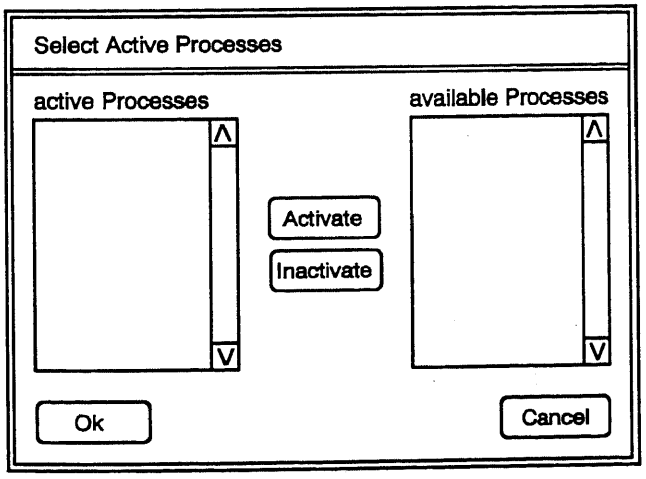


Fig. A3.24: Dialog box for activating processes.

After choosing *Add* or *Edit*, the dialog box shown in Fig. A3.26,

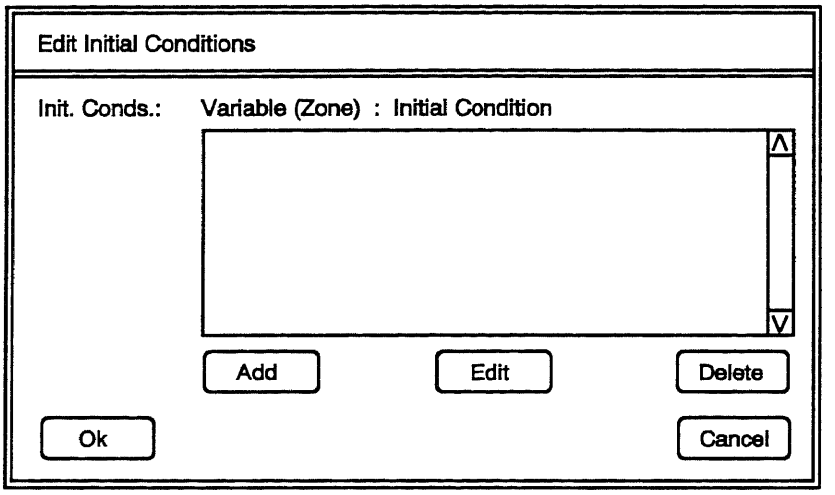


Fig. A3.25: Dialog box for editing initial conditions.

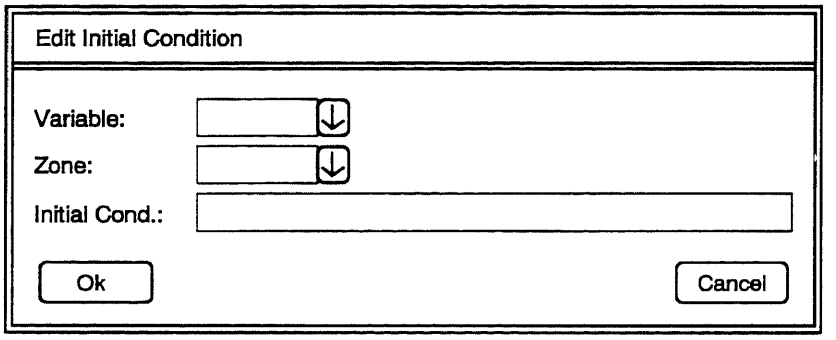


Fig. A3.26: Dialog box for editing a single initial condition.

allows the user to define or edit an initial condition. A *Variable* and a *Zone* have to be selected. If the compartment only consists of one zone, the zone item is inactive. Note that for a given combination of variable and zone only one initial condition can be specified. Initial conditions may be defined for any type of variables, but only initial conditions for state variables and for some program variables (depending on the compartment) are used by the program. In the third line of the dialog box shown in Fig. A3.26, the *initial condition* corresponding to the variable and zone chosen above, can be specified as an algebraic expression. An initial condition may not depend on state variables, but some program variables are allowed (cf. detailed description of compartments given in the following subsections). In particular, an initial condition may depend on the space coordinate of a compartment and on the program variables "Calculation Number" and "Time". The expression given as initial condition is used as the initial condition of the simulation, if the user selects to start a calculation with given initial data; if the user selects to start with a steady state solution, it is used to start the iteration algorithm calculating the steady state solution (cf. section A4.1). Therefore, the user can help the program finding the steady state solution by providing a reasonable initial condition. Note, that the units of initial conditions are the same as those of the corresponding state variables.

As a last option common to all compartments, is the definition of inputs into the compartment. This is done by choosing the ***Input*** button shown in Fig. A3.22. This action opens the dialog box shown in Fig. A3.27. The inputs defined here are external inputs, which add to a possible input due to a link from another compartment. Input definition consist of specification of *water inflow* and *input fluxes* of substances. Water inflow can be specified as an algebraic expression. The input fluxes already

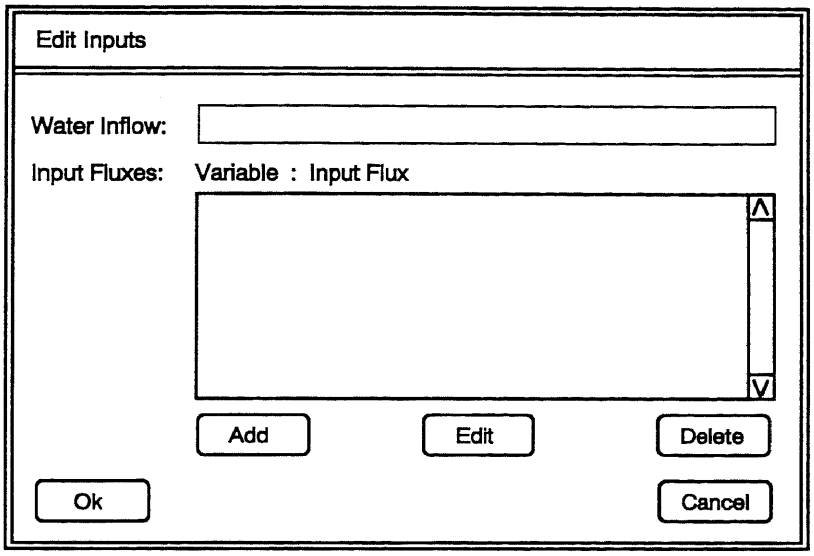


Fig. A3.27: Dialog box for editing inputs.

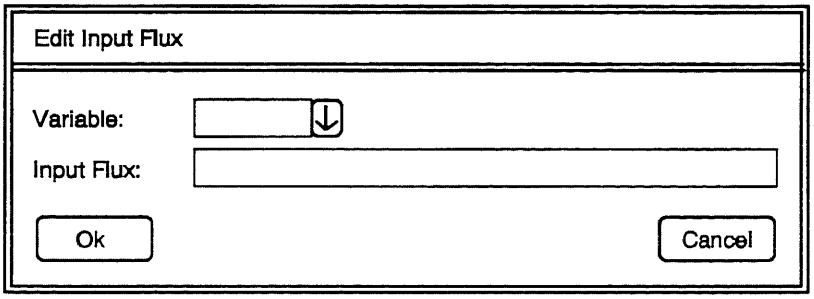


Fig. A3.28: Dialog box for editing a single input flux.

defined for various variables are listed in the list box of the dialog box shown in Fig. A3.27. The set of input fluxes can be edited using the buttons *Add*, *Edit* and *Delete* of the dialog box. Selection of *Add* leads to addition of the new input flux at the end of the list, if no input flux is selected, otherwise the new input flux is inserted before the selected input flux (the order of input fluxes is irrelevant, but a user-defined order may improve clearness of presentation). The buttons *Edit* and *Delete* are only active, if an input flux in the list box is selected.

After choosing *Add* or *Edit*, the dialog box shown in Fig. A3.28 allows the user to define or edit an input flux. A *Variable*, for which no input flux has been defined yet, has to be selected. Input fluxes may be defined for any type of variables, but only inputs for dynamic volume state variables are used by the program. This feature again facilitates switching between different levels of modelling without the need of changing input definitions. In the second line of the dialog box shown in Fig A3.28, the *input flux* can be specified as an algebraic expression. Note, that the flux (mass per unit of time) and not the concentration has to be specified. If water inflow is specified as a variable, input fluxes of variables transported with water inflow can be expressed as products of water inflow and concentration. The specification of flux, however, permits the user also to describe point sources of substances at locations, where water inflow is zero. In contrast to water inflow, input fluxes may also be negative and are then treated as outflow fluxes (this is only meaningful, if there is a link connected to the inflow of the compartment and some of the inflowing mass is lost). Note, that mass units must correspond to those of the corresponding dynamic volume state variables and time units to those used for simulation time. For the formulation of water inflow and input fluxes, the program variable "Discharge" and dynamic volume state variables may be used. They take the values resulting from water flow and substance flux of all links

connected to the inflow connection of the compartment. Furthermore, the inputs may depend on the program variables "Calculation Number" and "Time".

A3.3.2 Mixed Reactor Compartment

A mixed reactor compartment models a reactor in which concentration gradients may be neglected. It consists of only one zone (bulk volume).

Fig. A3.29 shows the data required to define a mixed reactor compartment (refer to section A3.3.1 for a description of the items in the first three lines in this dialog box)). Two ***Reactor Types*** are distinguished: For a reactor with *constant volume*, the ***Volume*** has to be specified, whereas for the reactor with *variable volume*, the ***Outflow*** has to be given as an algebraic expression (especially the program variables "Time", "Reactor Volume" and "Discharge" are useful for the specification of outflow). The outflow has to be carefully specified to avoid that the reactor volume becomes negative. Note, that the units of volume and outflow have to be the same as those used for the definition of concentrations of dynamic volume state variables.

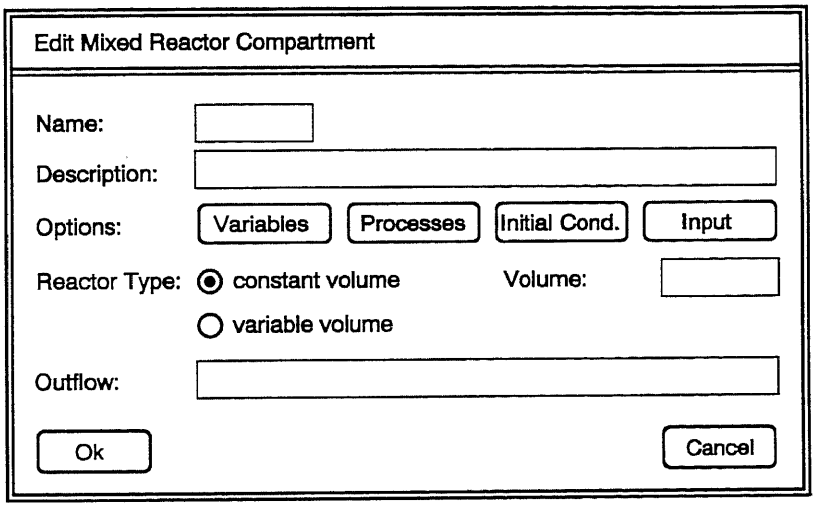


Fig. A3.29: Dialog box for the definition of a mixed reactor compartment.

Consult Table A3.1 for the availability and meaning of program variables within a mixed reactor compartment.

For the case of a mixed reactor compartment with variable volume, an initial condition for the program variable "Reactor Volume" can be given. If no such initial condition is given, the value specified for the reactor with constant volume is used as initial condition for volume. Additional initial conditions can be given for all state variables (state variables without explicitly specified initial conditions are initially set to zero).

A mixed reactor compartment has three connections for linking it to other compartments. The *Inflow* can be connected to any number of outflow connections of advective links, the *Outflow* can be connected to a single inflow connection of an advective link. Furthermore, any number of diffusive links can be connected to the *Bulk Volume* of a mixed reactor. The pictogram shown in Fig. A3.30 visualizes the possible connections of a mixed reactor compartment. The connections on the left and right hand sides symbolize advective inflow and outflow connections, respectively. The arrow shows the point of input to the compartment. The top and bottom connections both are connections to a diffusive link. In the case of the mixed reactor compartment, the two diffusive connections are equivalent. Two of them are indicated to facilitate the geometrical arrangement of compartments and links.

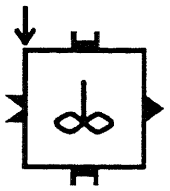


Fig. A3.30: Pictogram of a mixed reactor compartment.

A3.3.3 Biofilm Reactor Compartment

A biofilm reactor compartment models a reactor containing a zone of mixed water (bulk volume) and a zone consisting of a film of attached living microorganisms, dead particles and water (biofilm). These two zones are separated by a boundary layer of negligible volume, which manifests its existence by a diffusive mass transfer resistance. Dissolved substances can diffuse into the film and may be produced and consumed by the microorganisms building the biofilm matrix or suspended in the bulk volume. The biofilm is mathematically described by the equations originally derived by Wanner and Gujer (1984) and (1986) and discussed and extended in Wanner and Reichert (1994). This model averages biofilm structure and properties over planes parallel to the substratum and only resolves the direction perpendicular to the substratum, where the largest gradients occur.

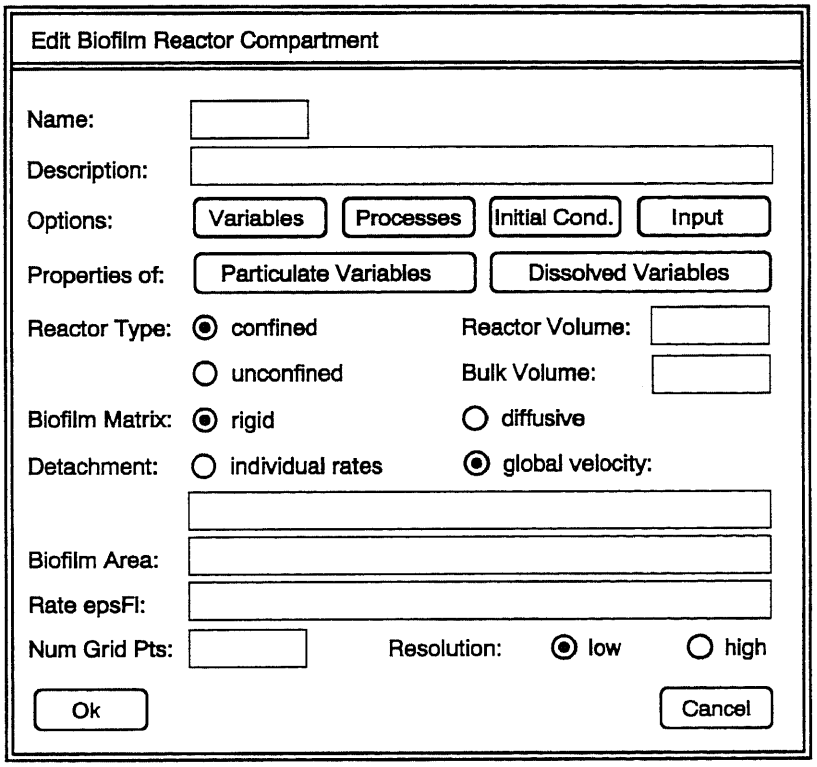


Fig. A3.31: Dialog box for the definition of a biofilm reactor compartment.

Fig. A3.31 shows the data required to define a biofilm reactor (refer to section A3.3.1 for a description of the items in the first three lines in this dialog box). Particulate variables forming the biofilm matrix and variables representing substances dissolved in water have to be distinguished.

Dynamic volume state variables can be made to particulate variables with the dialog box shown in Fig. A3.32, which is opened by pressing the ***Particulate Variables*** button in the dialog box shown in Fig. A3.31. In the list box of the dialog box shown in Fig. A3.32, all variables are listed, for which particulate properties are defined. This set of particulate variables can be edited with the buttons *Add*, *Edit* and *Delete*. The buttons *Edit* and *Delete* are inactive as long as no variable in the list box is selected. *Delete* only deletes particulate properties of a variable and not the variable itself. Choosing *Add* or *Edit* results in opening of the dialog box shown in Fig. A3.33, which allows the user to define or edit the properties of a particulate variable. The ***Density*** corresponds to the weight per volume of the particulate species (not per volume of biofilm including the water fraction). Density of a particulate species has to be constant in time and space. Therefore, density is not allowed to depend on state

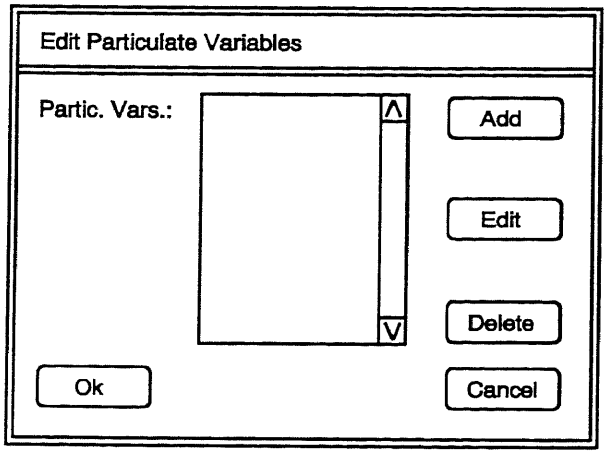


Fig. A3.32: Dialog box for editing particulate variables.

variables and on program variables other than "Calculation Number". ***Attachment and Detachment Coefficients*** have the dimension of velocities (length per unit of time) and control the processes at the biofilm surface. The detachment coefficient is only in effect, if individual detachment rates are selected in the dialog box shown in Fig. A3.31. The ***Boundary Layer Resistance*** makes it possible to limit mass transfer between film and bulk fluid. It has the dimension of an inverse velocity and may be written as the quotient of the thickness of a boundary layer and a (molecular) diffusion coefficient. The ***Diffusivity*** empirically parametrizes the movement of attached particles within the solid matrix of the biofilm. This quantity is only in effect if a diffusive biofilm matrix is selected in the dialog box shown in Fig. A3.31. If particulate properties are defined for a variable of another type than a dynamic volume state variable, these properties are ignored. This feature facilitates switching between different models. Concentrations of particulate species in the biofilm reactor compartment are interpreted as mass per total volume consisting of particles and water.

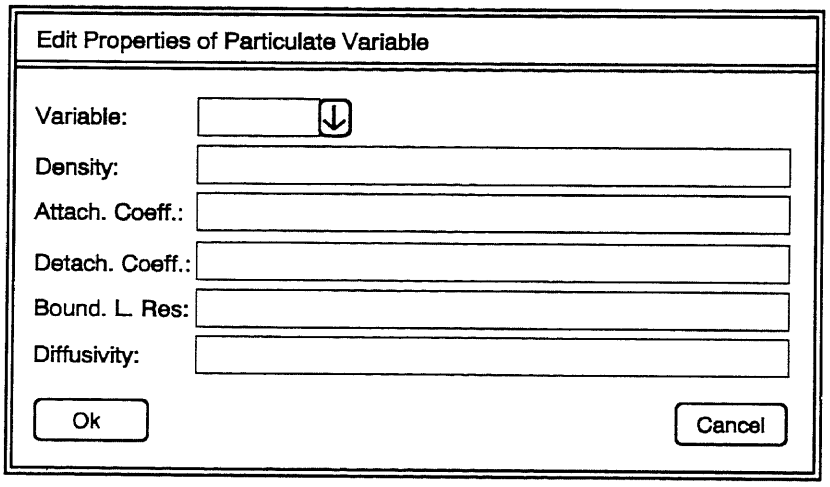


Fig. A3.33: Properties of a particulate variable.

Dynamic volume state variables can be made to dissolved variables with the dialog box shown in Fig. A3.34, which is opened by pressing the ***Dissolved Variables*** button of the dialog box shown in Fig. A3.31. In the list box of the dialog box shown in Fig. A3.34, all variables are listed, for which dissolved properties are defined. This set of dissolved variables can be edited with the buttons *Add*, *Edit* and *Delete*. The buttons *Edit* and *Delete* are inactive as long as no variable in the list box is selected. *Delete* only deletes dissolved properties of a variable and not the variable itself. Choosing *Add* or *Edit* results in opening of the dialog box shown in Fig. A3.35, which allows to define or edit properties of a dissolved variable. The ***Boundary Layer***

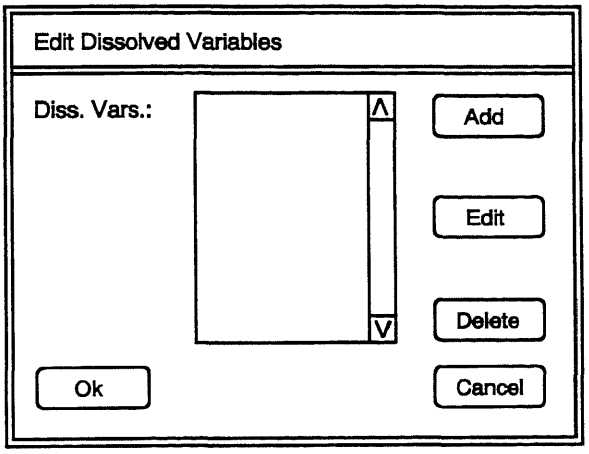


Fig. A3.34: Dialog box for editing dissolved variables.

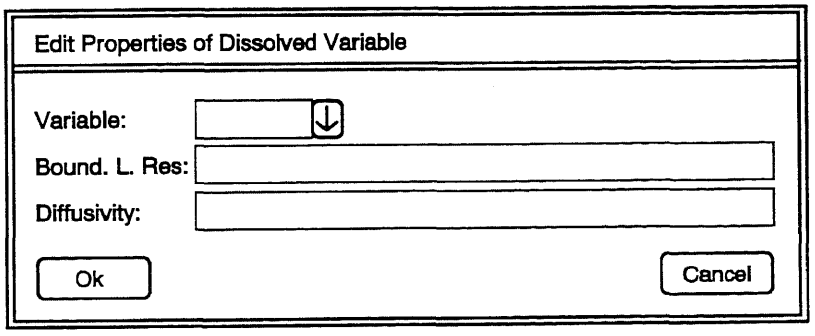


Fig. A3.35: Properties of a dissolved variable.

Resistance is interpreted in the same way as for particulate variables. The ***Diffusivity*** is the diffusion coefficient of the dissolved substance in the water between the particles forming the biofilm matrix. If tortuosity and penetration of turbulence into the film can be neglected, the diffusivity should be set to the molecular diffusion coefficient of the substance. Tortuosity leads to a decrease of the value of diffusivity, turbulence to an increase. It is possible to make diffusivity dependent on time and on the space coordinate in the film, to model the penetration of turbulence into the upper biofilm layers as a function of turbulence in the bulk water phase. If dissolved properties are defined for a variable of another type than a dynamic volume state variable, these properties are ignored. Dynamic volume state variables without particulate or dissolved properties behave like dissolved variables with boundary layer resistance and diffusivity zero. Concentrations of dissolved species in the biofilm reactor compartment are interpreted as mass per unit volume of the water phase excluding particles.

There are more items in the dialog box for the definition of a biofilm reactor compartment shown in Fig. A3.31:

Two ***Reactor Types*** are distinguished: *Confined* reactors are characterized by a constant ***Reactor Volume*** which is dynamically divided into film and bulk fluid volume. *Unconfined* reactors have a fixed value of the ***Bulk Volume***, so that the total reactor volume changes with a change of the film volume. These latter reactors model fluid flowing with a free surface over the biofilm.

The particulate components of the biofilm build the ***Biofilm Matrix***. This matrix can either be *rigid* or *diffusive*. In a rigid biofilm matrix, changes of biofilm structure at a given distance from the substratum is only due to advection caused by growth (or decay) in underlaying biofilm layers and by growth and decay processes at the location itself, as described in the original biofilm model of Wanner and Gujer (1984)

and (1986). The case of a diffusive biofilm matrix is a generalization of this model described by Wanner and Reichert (1994), which accounts for additional structural changes by an effective diffusion process. The diffusion coefficients defined for particulate variables in the dialog box shown in Fig. A3.33 are only used for the case of a diffusive biofilm matrix.

Detachment processes can be simulated by *individual detachment rates* for various components (only in case of a diffusive biofilm matrix), or by a *global detachment velocity.*

The ***Biofilm Area*** is used to define the geometry of the reactor. It is defined as the surface area parallel to the substratum as a function of the distance from the film-substratum interface, which is available with the aid of the program variable "Space Coordinate Z". The biofilm area may not depend on any other program variable with the exception of the program variable "Calculation Number" and it may also not depend on state variables. A constant surface area

$$A = \text{const} \qquad \text{(A3.1a)}$$

describes a plane film, a surface area of the form

$$A(z) = 2\pi L_{cy} (z_{cy}+z) \qquad \text{(A3.1b)}$$

or

$$A(z) = 2\pi L_{cy} (z_{cy}-z) \qquad \text{(A3.1c)}$$

describes a film growing outside or inside of a cylinder of length L_{cy} and radius z_{cy}, respectively, and a surface of the form

$$A(z) = 4\pi n_{sp} (z_{sp}+z)^2 \qquad \text{(A3.1d)}$$

describes a film growing on n_{sp} spherical particles of radius z_{sp} (in these examples, z corresponds to the program variable "Space Coordinate Z").

The ***Rate of*** ε_{Fl} corresponds to the excess rate of growth of water volume between the particles building the biofilm matrix (cf. Wanner and Reichert, 1994). This rate allows to model changes in the water content of the film. The original model of Wanner and Gujer (1986), in which the water fraction of the film remained constant, corresponds to a rate of zero.

Finally, the ***Number of Grid Points*** together with the ***Resolution*** defines the level of discretization. If the number of grid points is n, the film is resolved into two surface points and n-2 biofilm layers. *Low* resolution corresponds to a first order discretization, *high* resolution to a second order discretization with flux limiters to avoid oscillations.

Note, that for all specifications shown in Figs. A3.31, A3.33 and A3.35, mass and space units have to agree with those used for the definition of concentrations with

dynamic volume state variables and time units have to agree with the unit of simulation time.

To enable the simulation of microbial growth on dissolved substrates, process rates are interpreted as mass change per unit time and per unit of total volume consisting of particles and water. Rates of processes which only take place within the water fraction between the particles (as is true for chemical reactions not affected by the particles) should be multiplied by the program variable "Water Fraction" to give a correct mass balance (cf. more detailed discussion in Wanner and Reichert, 1994).

Consult Table A3.1 for the availability and meaning of program variables within a biofilm reactor compartment. The program variable "Zone Index" can be used to make processes different in the film and in the bulk volume (e.g. aeration of bulk water can be modelled by a process the rate of which is zero for zone index equal to two and takes the desired value for zone index equal to one).

Initial film thickness can be specified by defining an initial condition for the program variable "Biofilm Thickness" in the zone "Biofilm". Additional initial conditions can be given for all state variables in both zones (state variables without explicitly specified initial conditions are initially set to zero). The program variables "Space Coordinate Z" and "Biofilm Thickness" can be used to specify initial conditions.

A biofilm reactor compartment has four connections for linking it to other compartments. The *Inflow* can be connected to any number of outflow connections of advective links, the *Outflow* can be connected to a single inflow connection of an advective link. Furthermore, any number of diffusive links can be connected to the *Bulk Volume* or to the *Biofilm Base* of a biofilm reactor. The pictogram shown in Fig. A3.36 visualizes the possible connections of a biofilm reactor compartment. The connections on the left and right hand sides symbolize advective inflow and outflow connections to the bulk volume, respectively. The arrow shows the point of input to the compartment. The top connection is the diffusive connection to the bulk volume of the reactor and the bottom connection is the diffusive connection to the base of the biofilm.

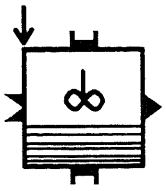


Fig. A3.36: Pictogram of a biofilm reactor compartment.

A3.3.4 River Section Compartment

A river section compartment models a reach of an open channel not containing significant hydraulic structures. The user has the choice of modeling hydraulics with the kinematic or diffusive approximation to the equations of open channel flow (cf. Yen, 1973 and 1979). A river section compartment consists of a single zone (water column).

Fig. A3.37 shows the data required to define a river section compartment (refer to section A3.3.1 for a description of the items in the first three lines in this dialog box). Input to a river section compartment consists of upstream input and lateral input. This leads to a modification of the dialog sequence started with the *Input* button shown in Fig. A3.37 and described in section A3.3.1 (Figs. A3.27 and A3.28). After pressing the *Input* button, the dialog box shown in Fig. A3.38 allows the user to select the type of input. Selection of *Upstream Input* activates the dialog box shown in Fig. A3.27 common to all compartment types. Selection of *Lateral Input* opens the dialog box shown in Fig. A3.39. This dialog box allows the user to specify lateral *Water Inflow* as volume per time and unit river length. Positive inflow leads to increase of river discharge (e.g. exfiltration of groundwater or approximate description of small tributaries), negative inflow decreases river discharge (e.g. infiltration into groundwater). *Inflow Concentrations* of dynamic volume state variables can be specified in the dialog box shown in Fig. A3.39. The set of currently defined inflow concentrations is given in the list box of this dialog box. This set can be edited using the buttons *Add*, *Edit* and *Delete* of the dialog box. Selection of *Add* leads to addition of the new inflow concentration at the end of the list, if no inflow concentration is selected, otherwise the new inflow concentration is inserted before the selected inflow concentration (the order of inflow concentrations is irrelevant, but a user-defined order may improve clear-

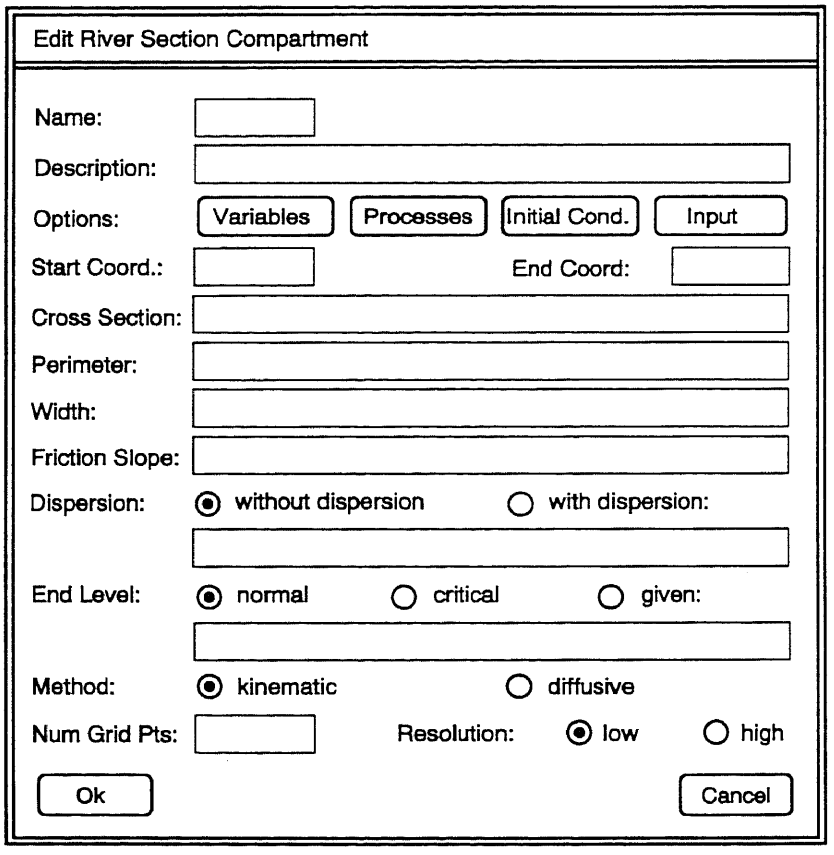


Fig. A3.37: Dialog box for the definition of a river section compartment.

ness of presentation). The buttons *Edit* and *Delete* are only active, if an inflow concentration in the list box is selected. After choosing *Add* or *Edit*, the dialog box shown in Fig. A3.40 allows the user to define or edit an inflow concentration. A *Variable*, for which no inflow concentration has been defined yet, has to be selected. Inflow concentrations may be defined for any type of variables, but only inputs for dynamic volume state variables are used by the program. This feature facilitates switching between different levels of modelling without the need of changing input definitions. In the second line of the dialog box shown in Fig A3.40, the *Inflow Concentration* can be specified as an algebraic expression. Inflow concentrations lead to a mass flux into the river per unit river length given as the product of lateral water inflow and inflow concentration if water inflow is positive, if water inflow is negative, the mass flux is determined by the current concentration in the river and inflow concentrations are ignored. Note, that mass units must correspond to those of the corresponding dynamic volume state variables and time units to those used for simulation time. For the formulation of water inflow and inflow concentrations, the program variables "Calculation Number", "Time", "Space Coordinate X" and "Discharge" and dynamic volume state variables may be used (cf. Table A3.1).

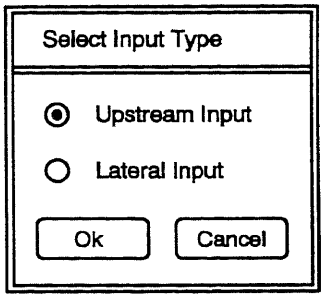


Fig. A3.38: Dialog box for selecting input type.

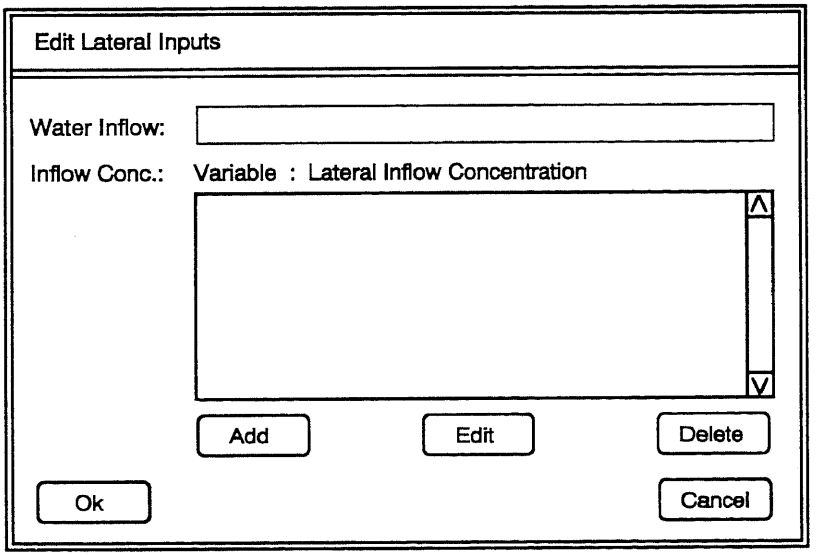


Fig. A3.39: Dialog box for editing lateral inputs.

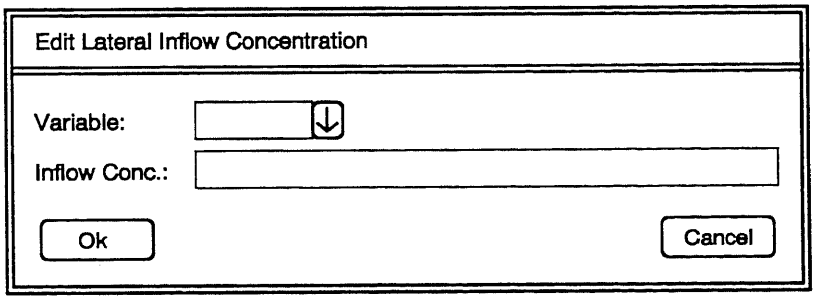


Fig. A3.40: Dialog box for editing a lateral inflow concentration.

The range of the space coordinate x along the river is between ***Start Coordinate*** and ***End Coordinate***. Downstream discharge (from start to end), accessible through the program variable "Discharge" (cf. Table A3.1), is always positive, irrelevant if the value of the start coordinate is smaller or larger than that of the end coordinate.

The expressions specified for the ***Cross Section***, the ***Perimeter*** and the ***Width*** define the geometry of the river bed as functions of distance along the river represented by the program variable "Space Coordinate X" and of water surface elevation (with respect to an absolute basis point, e.g. sea level) represented by the program variable "Water Level Elevation" (cf. Table A3.1). The cross sectional area has to be an increasing function of the program variable "Water Level Elevation". As an example, a rectangular channel with slope S_o can be described by the expression

$$A(x,z_o) = (z_o+S_ox)\, w \qquad \text{(A3.2)}$$

for the cross-sectional area, by the expression

$$P(x,z_o) = w + 2\,(z_o+S_ox) \qquad \text{(A3.3)}$$

for the length of the wetted perimeter and by

$$w(x,z_o) = w \qquad \text{(A3.4)}$$

for the (constant) surface width. In these expressions, x represents the program variable "Space Coordinate X", z_o the program variable "Water Level Elevation", w the width of the rectangular channel and (z_o+S_ox) corresponds to water depth.

The expression specified for the ***Friction Slope*** defines the non-dimensional friction force, defined as the ratio of friction force on a water segment divided by the gravitational force. It has to be a function of the program variables "Discharge" and "Cross Sectional Area" (cf. Table A3.1). The most often used expression for empirically parametrizing friction in open channel flow is given by

$$S_f = \frac{1}{K_{st}^2}\,\frac{1}{R^{4/3}}\,\frac{Q^2}{A^2} = n^2\,\frac{1}{R^{4/3}}\,\frac{Q^2}{A^2} \qquad \text{(A3.5)}$$

where R=A/P is the hydraulic radius, Q is discharge and K_{st} and n $(=1/K_{st})$ are two alternative friction coefficients (cf. French, 1985).

The ***Dispersion*** coefficient determines spreading of transported substances. According to Fischer et al. (1979), it can be estimated as

$$E = 0.011\,\frac{w^2\,(Q/A)^2}{u^*\,d} \qquad \text{(A3.6)}$$

where

$$u^* = \sqrt{g\,d\,S_f} \qquad \text{(A3.7)}$$

is the friction velocity and d the mean river depth.

The ***Method*** of calculating hydraulics can be chosen between the *kinematic* and the *diffusive* approximation of the equations of open channel flow. Because the kinematic approximation uses the local slope of the river bed, it cannot be used for very irregular geometry of the bed.

The ***End Level*** is used to specify the downstream boundary condition. *Normal* depth is used to simulate an open boundary with no influence of a hydraulic control, *critical* depth models a drop and any *given* expression for a water level discharge relation can be used to model hydraulic controls such as weirs. The downstream boundary condition is not required for kinematic calculations.

The ***Number of Grid Points*** together with the ***Resolution*** defines the level of discretization. If the number of grid points is n, the river section is resolved into two end points and n-2 cells of equal length. *Low* resolution corresponds to a first order upstream discretization, *high* resolution to a second order discretization with flux limiters to avoid oscillations.

Note, that for all specifications shown in Fig. A3.37, mass and space units have to agree with those used for the definition of concentrations with dynamic volume state variables and time units have to agree with the unit of simulation time. Furthermore, the units of the program variables "Time" and "Space Coordinate X" (cf. Table A3.1) are used to calculate the gravitational acceleration needed for the solution of the diffusive approximation to the equations of open channel flow. Furthermore, before starting to define a river section compartment, the program variables "Space Coordinate X", "Water Level Elevation", "Cross Sectional Area" and "Discharge" have to be introduced.

Consult Table A3.1 for the availability and meaning of program variables within a river section compartment.

The user has to provide initial conditions for the program variable "Discharge". Approximate initial conditions for the program variable "Water Level Elevation" may help accelerating initialization of simulations. The program variables "Calculation Number", "Time" and "Space Coordinate X" may be used for the formulation of initial conditions.

A river section compartment has two connections for linking it to other compartments. The *Upstream End* can be connected to any number of outflow connections of advective links, the *Downstream End* can be connected to a single inflow connection of an advective link. The pictogram shown in Fig. A3.41 visualizes the possible connections of a river section compartment. The connections on the left and right hand sides symbolize advective inflow at the upstream end and outflow at the downstream end of the river section, respectively. The arrows show the points of input to the compartment.

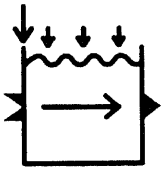


Fig. A3.41: Pictogram of a river section compartment.

A3.4 Links

Fig. A3.42 shows the dialog box opened with the edit links command shown in Fig. A3.2. This dialog box is of modeless type to accelerate editing the system of links. The names of the links already defined are listed alphabetically in the list box of this dialog box. The type of the currently selected link is indicated at the bottom of the dialog box. The buttons of this dialog box allow the user to perform the following operations with links: ***New*** links may be created directly or old links can be ***duplicated*** (in both cases the new link needs a new name). It is possible to ***edit*** and to ***delete*** links. The buttons *Duplicate*, *Edit* and *Delete* are inactive as long as no link is selected. Pressing the ***Close*** button results in closing of this dialog box. It can be reopened by choosing the edit links item of the menu shown in Fig. A3.2.

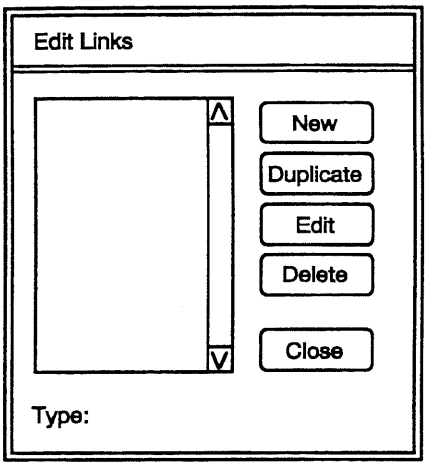


Fig. A3.42: Dialog box for editing the system of links.

For the definition of links with the subdialogs assigned to the dialog box shown in Fig. A3.42, all variables of the system of variables (cf. section A3.1), that do not depend on illegal variables for the edited link (cf. subsections of this section) can be used.

A3.4.1 Types of Links

The dialog box shown in Fig. A3.43 allows the user to select the type of a link after choosing the button *New* of the dialog box shown in Fig. A3.42. Two types of links are considered. ***Advective Links*** are used to describe water flow and advective substance transport between compartments. ***Diffusive Links*** model diffusive boundary layers or membranes between compartements, which can be diffusively penetrated by certain substances. There is no water flow through diffusive links.

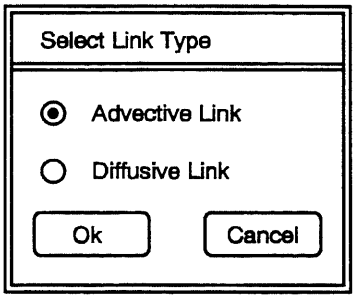


Fig. A3.43: Possible types of links.

After selecting the *Edit* button of the dialog box shown

in Fig. A3.42 or selecting the *New* button of this dialog box and specifying the desired link type with the dialog box shown in Fig. A3.43, a type-specific dialog box appears, which allows the user to define or edit the link. Fig. A3.44 shows the type independent entries of these dialog boxes. Each link needs a unique ***Name*** as an identifier. A name of a link consists of a sequence of letters (A-Z,a-z), digits (0-9) and underline characters (_). The first character may not be a digit. To improve documentation of links, a ***Description*** can be given optionally. The two ***Compartments*** and their ***Connections*** which are to be linked must be specified. If the compartment to be connected to the link has only one available connection of the correct type, the connection is automatically filled in by the program and the user cannot change this entry.

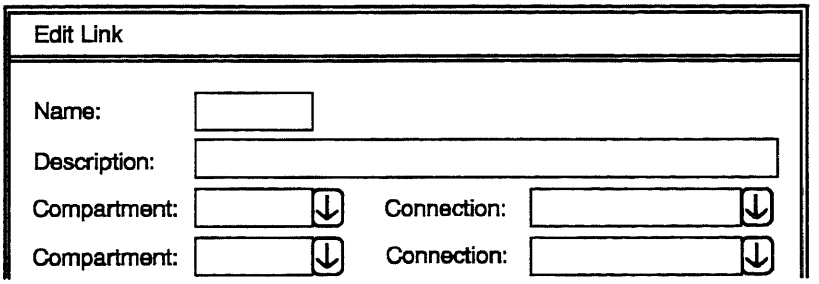


Fig. A3.44: Common entries of edit link dialog boxes.

A3.4.2 Advective Link

An advective link models water flow and advective substance transport between compartments. The discharge entering the link is determined from the outflow of the compartment connected to the inflow connection of the link. Advective links also allow to describe bifurcations. Junctions can be realized using advective links by connecting output connections of more than one advective link to an input connection of a compartment.

Fig. A3.45 shows the data required to define an advective link (refer to section A3.4.1 for a general description of the items in the first four lines in this dialog box). The compartment ***Comp. In*** with its ***Connection*** specifies the inflow to the link. The fraction of all mass fluxes not bifurcating at the link leave the link to the compartment ***Comp. Out*** or exit the system if this compartment is not specified. The names of all bifurcations already defined are listed alphabetically in the list box of the dialog box shown in Fig. A3.45. The buttons *Add*, *Edit* and *Delete* allow to edit the set of bifurcations. The buttons *Edit* and *Delete* are only active, if a bifurcation in the list box is selected.

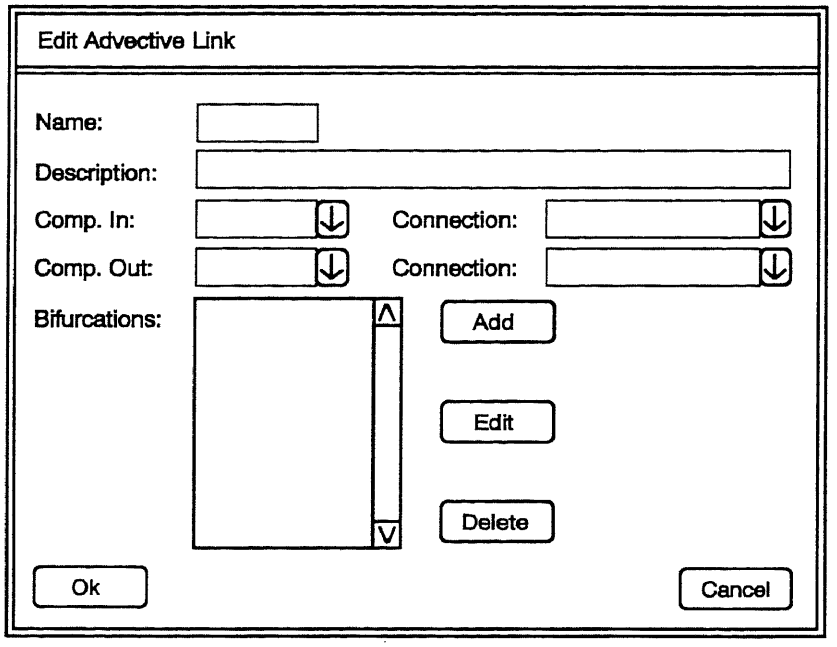


Fig. A3.45: Dialog box for the definition of an advective link.

Pressing one of the buttons *Add* or *Edit* results in opening of the dialog box shown in Fig. A3.46. This dialog box allows the user to define or edit the properties of the bifurcation. Each bifurcation within an advective link needs a unique ***Name*** as an identifier. A name of a bifurcation consists of a sequence of letters (A-Z,a-z), digits (0-9) and underline characters (_). The first character may not be a digit. To improve documentation of bifurcations, a ***Description*** can be given optionally. If the bifurcation does not exit the system, a ***Compartment*** and, if this compartment has more than one input connection, a ***Connection*** of this compartment have to be specified. The bifurcating ***Water Flow*** has to be given as an algebraic expression. Then, it has to be decided, if the ***Fluxes*** of substances are due to passive transport with water flow, or if substances are transported selectively. In the latter case, the mass fluxes of all transported substances have to be specified.

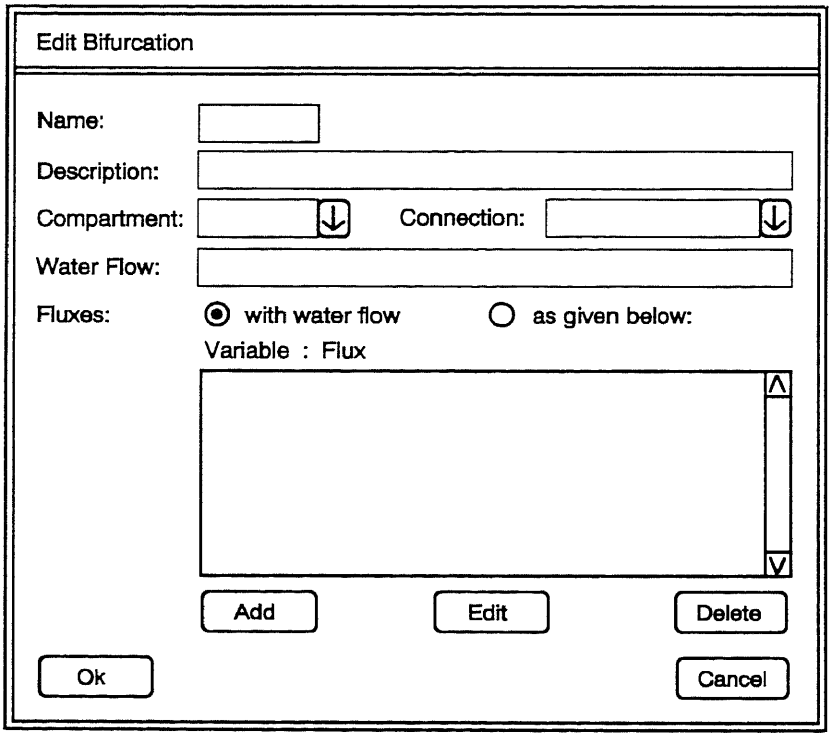


Fig. A3.46: Dialog box for editing a bifurcation.

Fig. A3.47 shows the dialog box used for defining a bifurcation flux of a substance. As a first point, the ***Variable*** corresponding to the substance has to be selected, then the mass ***Flux*** has to be specified as an algebraic expression. The program variables "Time",

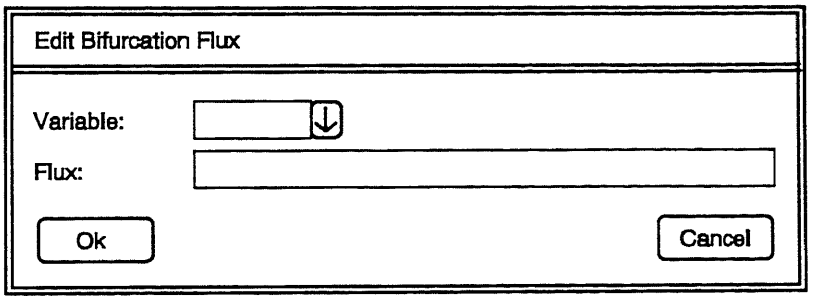


Fig. A3.47: Dialog box for editing a bifurcation flux.

"Calculation Number" and "Discharge" (representing water inflow to the link) and the dynamic volume state variables (representing concentrations of substances in the inflow to the link) can be used to formulate water flow and substance mass fluxes. Note, that total water flow and total mass fluxes bifurcating should not exceed the inflow to the link. In contrast to water flow, mass fluxes are allowed to be negative.

The pictograms shown in Fig. A3.48 visualize examples of advective links with different numbers of bifurcations.

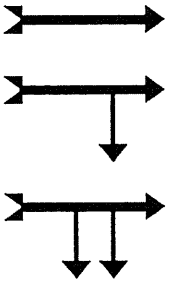


Fig. A3.48: Pictograms for advective links.

A3.4.3 Diffusive Link

A diffusive link models a diffusive boundary layer or a membrane of negligible volume, which can be diffusively penetrated by some substances.

Fig. A3.49 shows the data required to define a diffusive link (refer to section A3.4.1 for a general description of the items in the first four lines in this dialog box). Both ***Comp. 1*** and ***Comp. 2*** and their ***Connections*** connected to the diffusive link have to be specified. The mass ***Exchange Coefficients*** of the substances penetrating the link and a ***Conversion Factor 1*** for the concentration of the substance in compart-

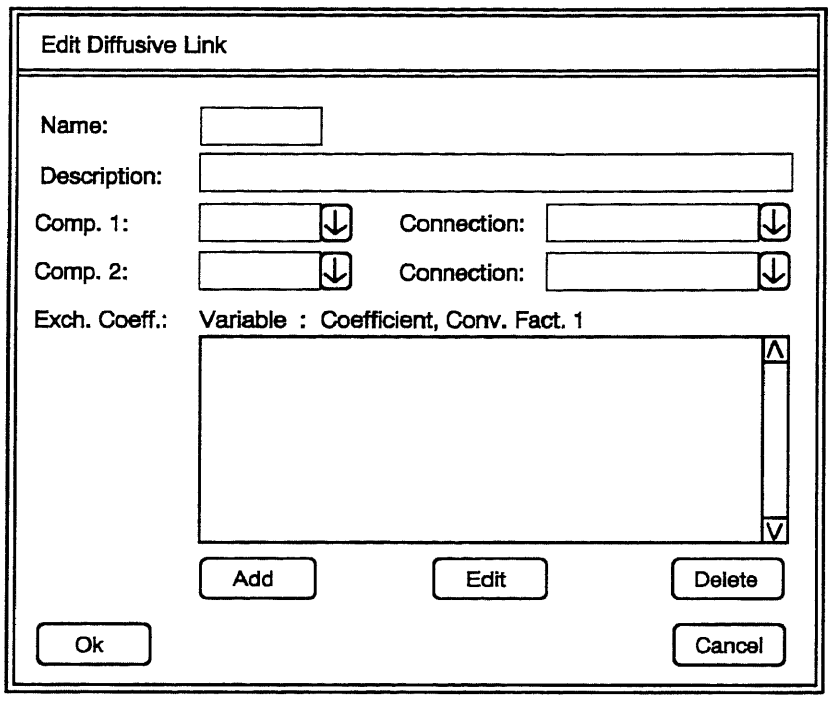


Fig. A3.49: Dialog box for the definition of a diffusive link.

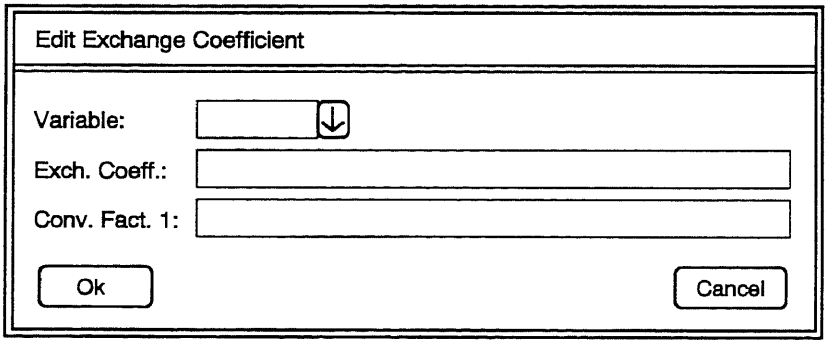


Fig. A3.50: Dialog box for editing an exchangecoefficient.

ment 1 have to be specified (mass flux is proportional to the product of exchange coefficient and concentration difference across the link). For the connection of compartments filled with water, the conversion factor takes its default value of unity. If a mixed reactor compartment, which models a reactor filled with gas, is connected to connection 1, the conversion factor should be set to the inverse of the non-dimensional Henry coefficient of the substance. Substances not present in the dialog box of Fig. A3.49 are not penetrated through the diffusive link. Exchange coefficients have only an effect to dynamic volume state variables. Fig. A3.50 shows the data required to define a mass exchange coefficient and a conversion factor. A variable has to be selected and the corresponding exchange coefficient and conversion factor have to be specified as algebraic expressions. The program variables "Time" and "Calculation Number" can be used to formulate exchange coefficients and conversion factors. A typical expression of an exchange coefficient is the product of the surface area of the link and the quotient of the diffusion coefficient of the substance in the membrane and the thickness of the membrane.

The pictogram shown in Fig. 3.51 visualizes a diffusive link.

Fig. A3.51: Pictogram of a diffusive link.

A3.4.4 Examples of Spatial Configurations

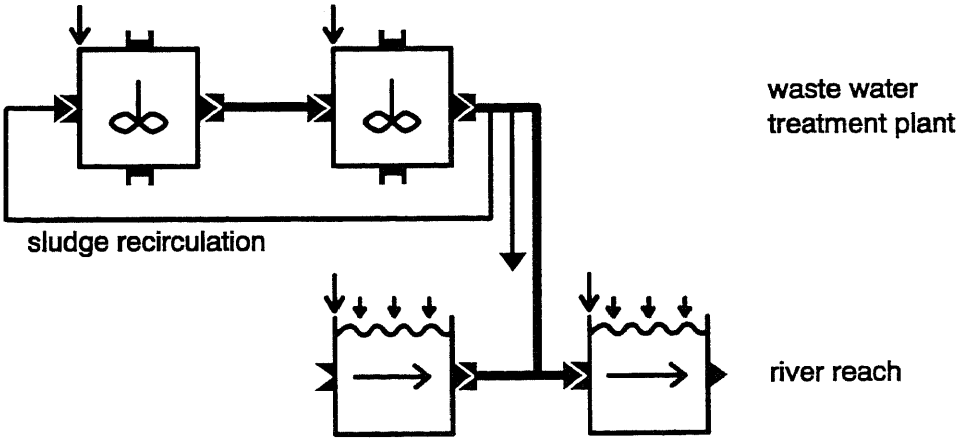


Fig. A3.52: Spatial configuration representing a waste water treatment plant and the river into which the treated waste water is released.

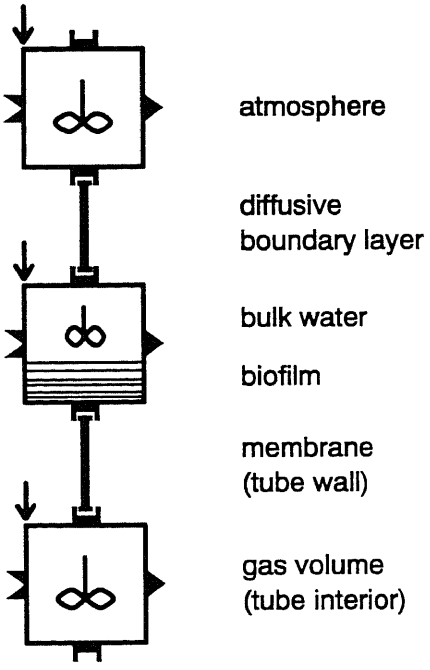


Fig. A3.53: Spatial configuration representing a biofilm growing on a permeable tube in a reactor with gas exchange to the atmosphere.

As examples of possible spatial configurations built with the elements described in the previous sections, Fig. A3.52 shows the configuration of a waste water treatment plant, the outflow of which is released into a river. The waste water treatment plant is modeled using two mixed reactor compartments connected by an advective link. Recirculation and export of sludge are described by bifurcations of the advective link to the river. Two river sections upstream and downstream of the point of release of purified waste water are distinguished. An example of a laboratory system is shown in Fig. A3.53. It consists of a biofilm growing on a diffusively permeable tube in a reactor with gas exchange to the atmosphere. Both, the atmosphere and the interior of the tube, on which the film grows, are described by mixed reactor compartments. The reactor containing the tube and the biofilm is modeled with a biofilm reactor compartment.

A3.5 Numerical Parameters

Fig. A3.54 shows the parameters of the numeric algorithms accessible to the user of the program. Because the time step of the integration algorithm changes dynamically, it is possible that a very short input pulse in an otherwise smooth input curve may be ignored. With the parameter ***Maximum Internal Time Step***, the integration time step can be bounded to avoid this problem. Similarly to the time step, also the integration order is varied dynamically. ***Maximum Integration Order*** can be bounded to increase the stability of the algorithm at the expense of smaller time steps. In order to increase the efficiency of the algorithm, the ***Number of Codiagonals of Jacobian*** can be decreased for linearly arranged systems (river sections coupled linearly or linear arrangements of mixed reactors without recirculation). The number of codiagonals should be chosen as small as possible, but large enough to guarantee convergence of the integration algorithm. In case of repeated convergence failures of the algorithm, increasing this parameter can solve the problem. In order to stop in the case of serious integration problems, the algorithm tests for a ***Maximum Number of Time Steps*** between output intervals. This maximum number can be changed by the user of the program. The ***Grid Relaxation Time Constant*** has the dimension of an inverse time. It does only influence the biofilm reactor compartment. With this coefficient set to zero the grid points within the biofilm are kept equidistant, otherwise relaxation with the inverse of this parameter as relaxation time to a better grid point distribution takes place (this feature is not yet implemented in version 1.0).

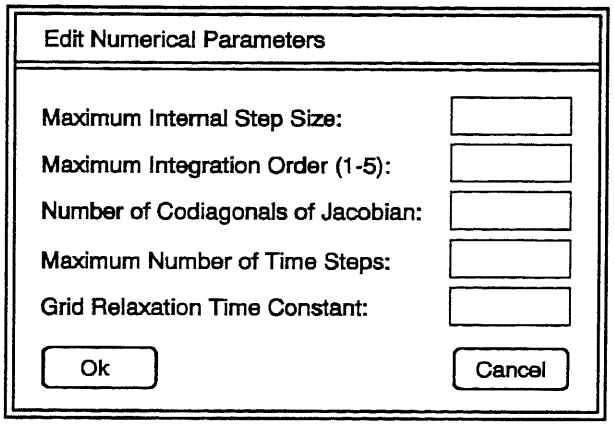


Fig. A3.54: Numerical parameters of AQUASIM.

A3.6 Deleting States

Calculations, as described in chapter A4, lead to storage of calculated states of the user-defined system. To free memory from states not longer needed, the dialog box shown in Fig. A3.55 allows the user to selectively delete calculated states. The left hand list box in this dialog box shows the calculation numbers, for which calculations exist; the right hand list box shows the times of the calculated states corresponding to the calculation number selected in the left list box. The left ***Delete*** button allows the user to delete all calculated states corresponding to the selected calculation number, whereas the right ***Delete*** button only deletes a single selected state. Note, that initializing a calculation also deletes all states with the corresponding calculation number and that editing the system leads to deletion of all calculated states.

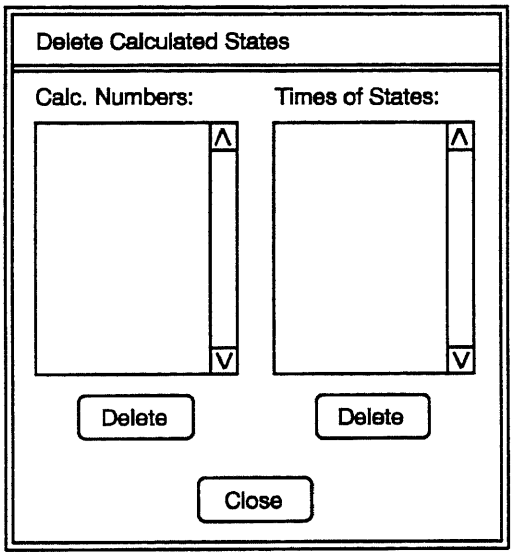


Fig. A3.55: Dialog box for deleting calculated states.

A4 Analyzing Data

Fig. A4.1 shows the menu **Calc** of AQUASIM (cf. Fig. A1.1). This menu reflects the three program tasks of ***Simulation***, ***Sensitivity Analysis*** and ***Parameter Estimation*** mentioned in chaper A1. Each of these tasks is described in more detail in one of the sections of this chapter.

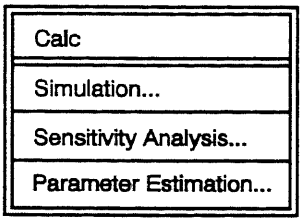


Fig. A4.1:
Calculate menu of AQUASIM

Each calculation performed with the aid of one of the above-mentioned program tasks, has to be identified by a user-specified, non-negative integer number, the calculation number. The actual value of this number is available in the system of variables with the aid of the program variable "Calculation Number". This makes the implementation of calculation-dependent models or model parameters possible.

Execution of calculations leads to storage of calculated states. For each value of the calculation number, a set of states is stored. Initialization of a calculation leads to deletion of all previously stored states corresponding to the same calculation number, whereas performing dynamic calculations leads to an extension of the set of stored states. Calculated states can selectively be deleted as described in section A3.6. Furthermore, editing an object of one of the four subsystems of AQUASIM model structure as described in sections A3.1 to A3.4 leads to deletion of all calculated states.

During calculations, information on the progress of the job and on behavior of the numeric algorithms is written to a log file. For interactive AQUASIM sessions (using the window or character interface version), this log file has the name `aquasim.log` and is stored in the startup directory of the program (previous versions are overwritten); the batch version of AQUASIM allows the user to specify the name of the log file. The information available in this file can be helpful for diagnosing the cause of numeric problems.

A4.1 Simulation

Fig. A4.2 shows the dialog box used for defining and starting simulations. This dialog box is opened by choosing the item *Simulation* of the calculation menu shown in Fig. A4.1. The first item within this dialog box allows the user to identify a calculation by a non-negative integer number, the ***Calculation Number***. The value of this number is irrelevant, if not variables have been made to depend on the program variable "Calculation Number", which makes the actual value of the calculation number available for model formulation.

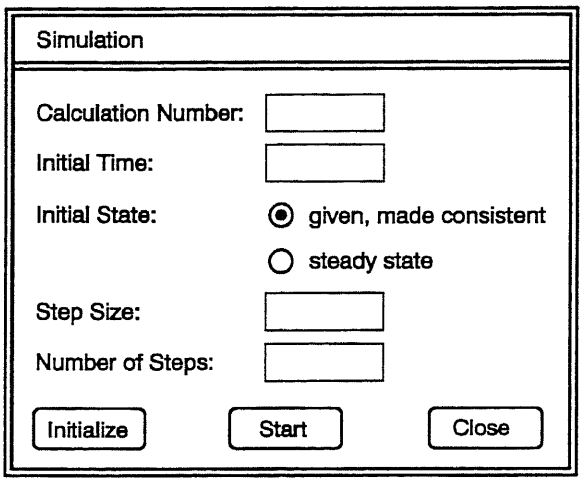


Fig. A4.2: Dialog box for performing simulations.

Each simulation has to be initialized before it can be started. The ***Initial Time*** is the time at which initial conditions (which may depend on time) are evaluated and at which a dynamic simulation can start. The user has the choice between two types of the ***Initial State***: The first choice uses the initial conditions *given* in the definitions of the compartments as described in section A3.3.1. These initial conditions may violate algebraic equations describing boundary conditions or equilibrium processes. For this reason, these initial conditions are *made consistent* during initialization by making the minimum of changes necessary for fulfilling all algebraic equations. The alternative choice for the initial state consists of using the *steady state* solution of the equations corresponding to the system defined as described in chapter A3. Note, however, that not all models have a steady state solution and that even if such a solution exists, it can be too difficult for the numeric algorithm to find it directly during initialization of the simulation (steady state solutions can always be found by "relaxation", i.e. by performing dynamic simulations with constant boundary conditions). The iterative algorithm, which tries to find a steady state solution, uses the initial conditions as defined for the compartments (cf. section A3.3.1) as initial values. Therefore, the user can help increase convergence rate of the algorithm by specifying reasonable initial conditions.

Dynamic simulations require the specification of a ***Step Size*** and a ***Number of Steps*** to be performed. The step size specified in the dialog box shown in Fig. A4.2 is not identical to the step size of the integration algorithm, which is chosen dynamically according to the accuracy of the state variables, but it determines the points of time, at which the solution is stored to be available for plotting and listing results.

Initialization of a simulation is started by pressing the button ***Initialize*** shown in Fig. A4.2. Only the items *Calculation Number*, *Initial Time* and *Initial State* of the dialog box shown in Fig. A4.2 influence initialization. Initialization leads to deletion of all previously calculated states corresponding to the same calculation number. During cal-

culation, the dialog box shown in Fig. A4.3 appears on the screen. This dialog box allows to interrupt the calculation.

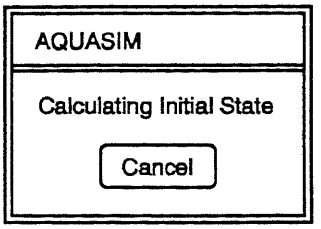


Fig. A4.3:
Dialog box for interrupting calculation of initial state.

Dynamic simulations are performed by pressing the button ***Start*** shown in Fig. A4.2. Only the items *Calculation Number*, *Step Size* and *Number of Steps* directly influence dynamic simulations. Pressing the button *Start* results in performing *Number of Steps* steps of size *Step Size* starting from the last available state corresponding to the same calculation number. Therefore, the first calculation after initialization starts at the *Initial Time* defined for initialization. Succeeding dynamic simulations continue previous simulations. During calculation of a dynamic solution, the dialog box shown in Fig. A4.4 appears on the screen. By displaying the ***Current Time*** of simulation, this dialog box allows the user to follow the progress of the calculation. Furthermore, this dialog box allows the user to interrupt the calculation. Note, that the user is responsible to use the same time unit for simulation time as that used for the definition of process rates, water inflow, substance input fluxes and any time dependent variable.

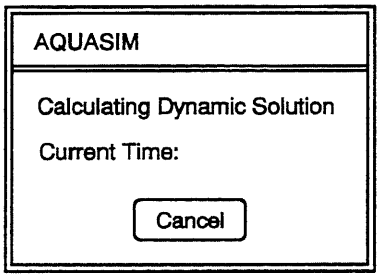


Fig. A4.4:
Dialog box for controlling and interrupting dynamic calculations.

A4.2 Sensitivity Analysis

As mentioned in the introduction, sensitivity analysis combines tasks of identifiability and uncertainty analysis.

The goal of identifiability analysis is to check if model parameters can uniquely be determined and to estimate the accuracy of model parameters. This is done by estimating the standard deviations and correlations of parameters during parameter estimations as discussed in the next section, and by comparing the behavior of one of the following sensitivity functions of measured variables with respect to estimated model parameters

$$\delta_{f,p}^{a,a} = \frac{\partial f}{\partial p} \quad , \tag{A4.1a}$$

$$\delta_{f,p}^{r,a} = \frac{1}{f}\frac{\partial f}{\partial p} \quad , \tag{A4.1b}$$

$$\delta_{f,p}^{a,r} = p\frac{\partial f}{\partial p} \quad , \tag{A4.1c}$$

$$\delta_{f,p}^{r,r} = \frac{p}{f}\frac{\partial f}{\partial p} \quad . \tag{A4.1d}$$

In these functions, f is an arbitrary variable and p is a parameter represented as a constant variable or a real list variable. The sensitivity function (A4.1a) measures absolute effects due to absolute changes of the parameter, (A4.1d) relative changes due to relative changes of the parameter, whereas (A4.1b) and (A4.1c) are mixed absolute-relative sensitivity functions. The most recommendable sensitivity functions are (A4.1c) and (A4.1d), because their units do not depend on the units of the parameter. This makes quantitative comparisons of the influence of different parameters on a common variable possible. If a variable f becomes very small during the simulation, the absolute sensitivity function with respect to the variable given by equation (A4.1c) should be used to avoid division by very small numbers. The larger the values of the sensitivity functions of the measured variables and the more different (within the range of measured data) the shapes, the better are the parameters identifiable.

Linear uncertainty analysis consists of estimating the standard deviation of a variable f according to the formula

$$\sigma_f = \sqrt{\sum_{i=1}^{m}\left(\frac{\partial f}{\partial p_i}\right)^2 \sigma_{p_i}^{\,2}} \tag{A4.2}$$

where p_i are the uncertain parameters, σ_{p_i} their standard deviations and the sum extends over all parameters considered. AQUASIM also allows its users to calculate the individual contributions

$$\delta_{f,pi}^{err} = \frac{\partial f}{\partial p_i} \sigma_{pi} \qquad \text{(A4.3)}$$

to this sum. A plot of these functions for all relevant parameters helps quickly finding the major sources of uncertainty.

Due to the mathematical similarity of the tasks, identifiability analysis with the aid of the linear sensitivity functions given by equations (A4.1a) to (A4.1d) and linear uncertainty analysis according to equations (A4.2) and (A4.3) can be done by the same program task of sensitivity analysis. Sensitivity analysis does not calculate one of the expressions (A4.1) to (A4.3), but it allows the user to select a set of parameters and performs the simulations necessary for calculating all of these expressions. After performing such a sensitivity analysis, the user can plot any of these expressions for arbitrary variables and for arbitrary parameters belonging to the set specified for sensitivity analysis. The necessary expression out of (A4.1) to (A4.3) are then only calculated for the variables and parameters selected by the user. In this section, only definition and performing of sensitivity analysis calculations is described. Presentation of results, including the functions (A4.1) to (A4.3), is treated in chapter A5.

Fig. A4.5 shows the dialog box used for defining and stating a sensitivity analysis. This dialog box is opened by choosing the item *Sensitivity Analysis* of the calculation menu shown in Fig. A4.1.

The two upper list boxes of this dialog box show the active and the available ***Parameters*** (constant variables and real list variables), respectively. Variables can be activated, by selecting them in the right list box and pressing the *Activate* button; they can be inactivated by selecting them in the left list box and pressing the *Inactivate* button (equivalently, this selection can be done with the check box *Active for Sensitivity Analysis* of one of the dialog boxes shown in Fig. A3.8 and A3.9).

The two lower list boxes of the dialog box shown in Fig. A4.5 show the active and the available definitions of ***Calculations*** for sensitivity analysis. A sensitivity analysis may consist of several active calculations with different calculation numbers. Calculations

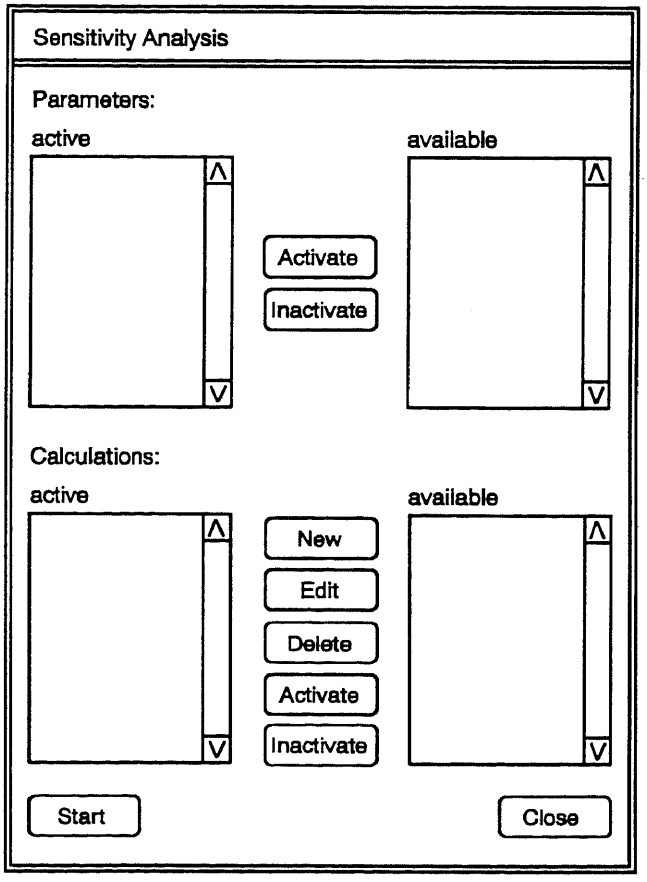


Fig. A4.5: Dialog box for defining and starting sensitivity analyses.

can be activated and inactivated in the same way as parameters. The set of available calculations can be edited using the buttons *New*, *Edit* and *Delete*. Fig. A4.6 shows the dialog box used for defining a single calculation. This dialog box appears, when one of the buttons *New* or *Edit* of the dialog box shown in Fig. A4.5 is pressed. A calculation is identified by a unique ***Name***. A name of a sensitivity analysis calculation consists of a sequence of letters (A-Z,a-z), digits (0-9) and underline characters (_). The first character may not be a digit. To improve documentation of calculations, a ***Description*** can be given optionally. The items required for the definition of a calculation are similar to those needed for a simulation as shown in Fig. A4.2. For each calculation, its ***Calculation Number*** has to be specified. Calculation numbers of active calculations have to be different from each other. As for simulations, the initial condition has to be characterized by the ***Initial Time*** and by the type of the ***Initial State***, and the dynamic simulation is determined by the ***Step Size*** and the ***Number of Steps*** to be performed. Any calculation can be ***Active for Sensitivity Analysis*** or inactive (this is the same decision as is made with the *Activate* and *Inactivate* buttons for sensitivity analysis calculations shown in Fig. A4.5).

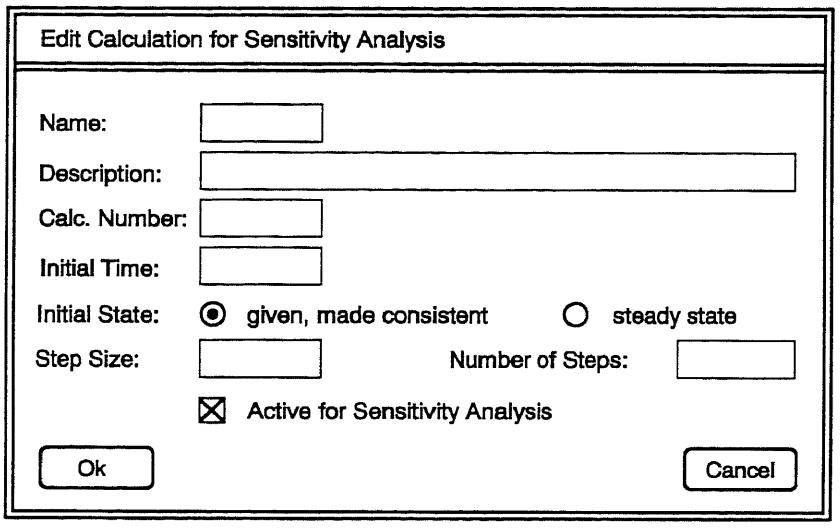


Fig. A4.6: Dialog box for editing a calculation for sensitivity analysis.

After starting a sensitivity analysis by pressing the ***Start*** button of the dialog box shown in Fig. A4.5, the dialog box shown in Fig. A4.7 appears on the screen. This dialog box shows, how many simulations are required for performing the sensitivity analysis and which is the current simulation. Furthermore, this dialog box allows the user to interrupt the calculation.

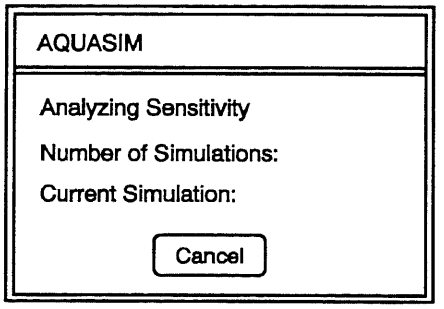


Fig. A4.7: Dialog box for controlling and interrupting sensitivity analysis.

A4.3 Parameter Estimation

Model parameters represented by constant variables can be estimated by minimizing the sum of the squares of the weighted deviations between measurements and calculation according to

$$\chi^2(\mathbf{p}) = \sum_{i=1}^{n} \left(\frac{f_{meas,i} - f_i(\mathbf{p})}{\sigma_{meas,i}} \right)^2 \qquad \text{(A4.4)}$$

where $f_i(\mathbf{p})$ is any variable calculated by the program at a given time and location in a compartment as a function of the parameters $\mathbf{p} = (p_1, \ldots, p_m)$, and $f_{meas,i}$ and $\sigma_{meas,i}$ are values and standard deviations of a measured quantities represented by real list variables. The sum may extend over several variables, compartments, zones and locations specified in the definition of the parameter estimation, and over all data pairs of the measured quantities involved.

Fig. A4.8 shows the dialog box used for specifying parameter estimations. All constant variables of the system of variables can be used as ***Parameters*** to be estimated. The two upper list boxes of the dialog box shown in Fig. A4.8 show the active and the available parameters, respectively. An inactive parameter can be activated by selecting it in the right list box and pressing the *Activate* button; an active parameter can be inactivated by selecting it in the left list box and pressing the *Inactivate* button (equivalently, this selection can be done with the check box *Active for Parameter Estimation* of the dialog box shown in Fig. A3.8). The subset used for the current parameter estimation has to be chosen as a first step.

A parameter estimation may consist of several ***Calculations*** with different calculation numbers for which the parameters are estimated simultaneously (individual parameters may be realized as variable lists of parameters using the calculation number as the argument). The two lower list boxes of the dialog box shown in Fig. A4.8 show the active and the available

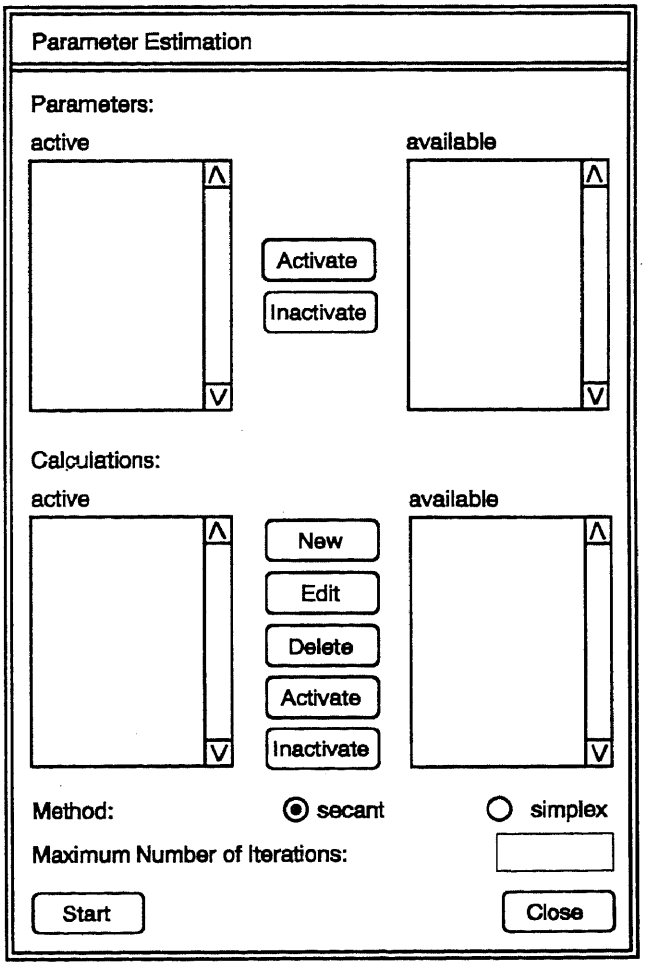


Fig. A4.8: Dialog box for defining and starting parameter estimation.

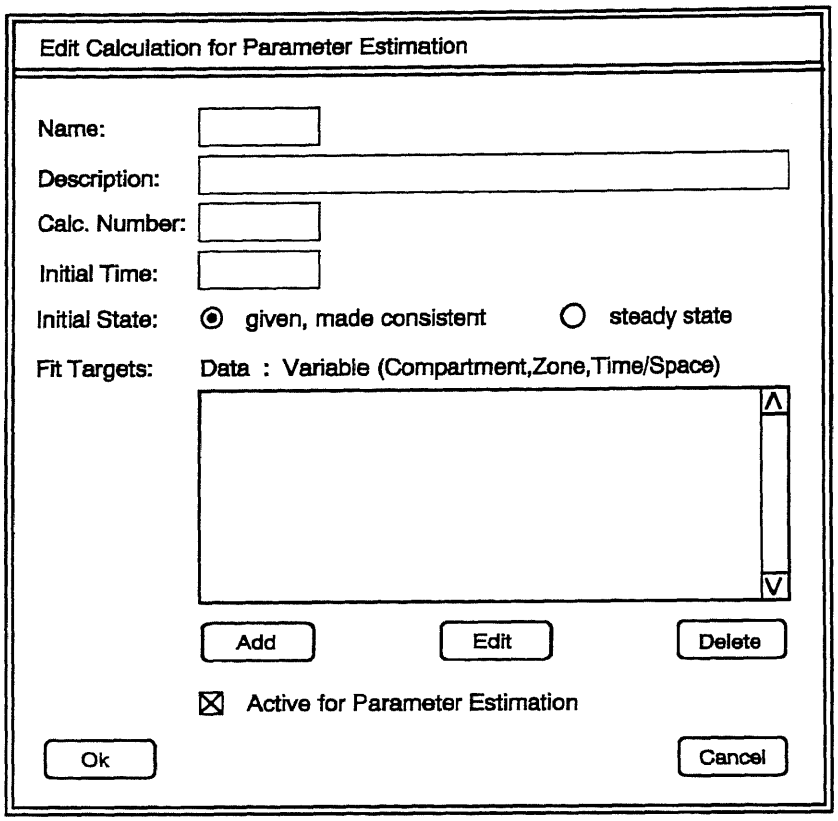


Fig. A4.9: Dialog box for defining a calculation for parameter estimation.

calculations for parameter estimation. Calculations can be activated and inactivated in the same way as parameters. The set of available calculations can be edited using the buttons *New*, *Edit* and *Delete.*

Fig. A4.9 shows the required data for the definition of each calculation. This dialog box appears, when one of the buttons *New* or *Edit* of the dialog box shown in Fig. A4.8 is pressed. A calculation is identified by a unique ***Name***. A name of a calculation consists of a sequence of letters (A-Z,a-z), digits (0-9) and underline characters (_). The first character may not be a digit. To improve documentation of calculations, a ***Description*** can be given optionally. For each calculation its ***Calculation Number*** has to be specified. Calculation numbers of active calculations have to be different from each other. As for simulations, the initial condition has to be characterized by the ***Initial Time*** and by the type of the ***Initial State***. Then, the quantities to be included into the sum given by equation (A4.4) have to be specified as ***Fit Targets***. The current set of such fit targets is provided in the list box of the dialog box shown in Fig. A4.9. This set can be edited using the buttons *Add*, *Edit* and *Delete.*

Fig. A4.10 shows the dialog box used for defining a single fit target. This dialog box is opened by pressing one of the buttons *Add* or *Edit* of the dialog box shown in Fig A4.9. A fit target consistes of measured data and of a calculated variable to be compared with the data. The measured ***Data*** may consist of any real list variable, the argument of which is either time or the space variable of the compartment, in which the comparison takes place. The calculated ***Variable*** to be compared with the data is characterized by its name, by the ***Compartment*** and the ***Zone*** where the comparison takes place, and either by the value of the ***Space*** coordinate, where the comparison with the data time series takes place, or by the value of ***Time***, at which the comparison with the spatial profile data takes place.

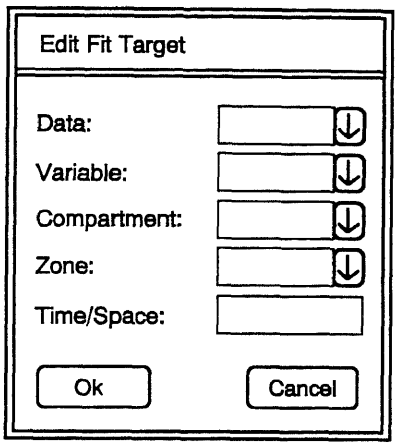
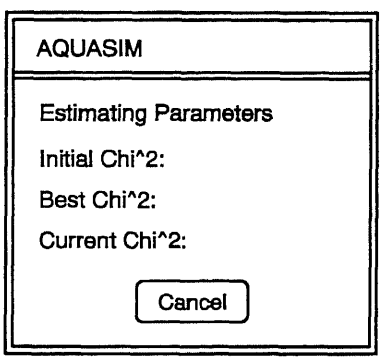


Fig. A4.10: Dialog box for the definition of a fit target.



ig. A4.11: Dialog box for controlling and interrupting parameter estimations.

Any calculation can be ***Active for Parameter Estimation*** or inactive (this is the same decision as is made with the *Activate* and *Inactivate* buttons for parameter estimation calculations shown in Fig. A4.8).

Before starting a parameter estimation, the numerical ***Method*** has to be selected and the ***Maximum Number of Iterations*** has to be given as is shown in Fig. A4.8. If a maximum of zero iterations is specified, the program only calculates the initial value of χ^2. After starting parameter estimation by pressing the button ***Start*** shown in Fig. A4.8, the user is asked for the name of an output file, where detailed results of the parameter estimation are written. Note, that parameter estimations may need a considerable amount of calculation time. During estimation of parameters, the dialog box shown in Fig. A4.11 appears on the screen. This dialog box allows to control progress of calculation by displaying the ***Initial Chi^2*** at start of estimation, the ***Best Chi^2*** obtained so far and the ***Current Chi^2*** of the last calculation completed. Furthermore, this dialog box allows the user to interrupt the calculation.

In case of normal termination of the parameter estimation, the dialog box shown in Fig. A4.12 appears to inform the user about the results. The ***Status*** may either be convergence of the algorithm or reaching the maximum number of iterations specified in the dialog box shown in Fig. A4.8. The ***Number of Iterations performed*** is shown in this dialog box. Then, for each ***Parameter***, its start value and its final value are indicated together with the limits of its legal range. Finally, the ***Initial and Final Chi Squared Values*** are shown to allow the user to judge the achieved progress and to statistically interpret the quality of the fit. More detailed results are written to the output file chosen for the parameter estimation as described above. These results include the number of data points and of active parameters of the selected parameter estimation, a listing of all sets of parameters which have been evaluated by the algorithm with the corresponding values of χ^2 according to equation (A4.4) and the total number of iterations and of simulations performed. In case of successful termination of the algorithm, the contribution of each data series to the total value of χ^2 is given for the initial and final sets of parameters. This is a very useful information for determining the data series which have dominant influence on the result. If the user selected the secant method for estimating the parameters and if none of the estimated parameters is on one of the boundaries of its legal range, an estimation of the standard deviations and of the correlation matrix of the parameters

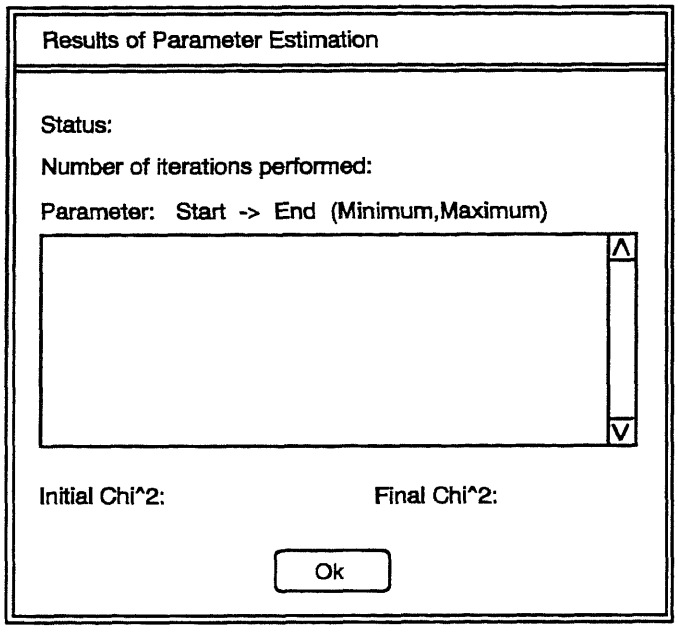


Fig. A4.12: Dialog box summarizing the results of the parameter estimation.

is given additionally. Note, that a parameter estimation changes the values of the constant variables used as parameters to the best estimates.

A5 Viewing Model and Results

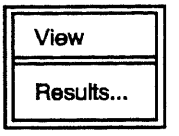


Fig. A5.1:
View menu of AQUASIM

Fig. A5.1 shows the menu **View** of AQUASIM (cf. Fig. A1.1). The sole item ***Results*** of this menu allows the user to specify plot definitions, to plot calculated results to the screen or to a PostScript file, which can be submitted to a printer, and to list plot data to a text file for external postprocessing. It is possible to plot time series or spatial profiles of the values of arbitrary variables (including uncertainty bounds, if a sensitivity analysis has been performed), of contributions of certain parameters to the uncertainty of arbitrary variables and of sensitivity functions of arbitrary variables with respect to certain parameters. These tasks are described in more detail in the section of this chapter.

A5.1 Plot Results

Plots of results base on the existence of calculated states. The strategy of AQUASIM is to allow the user to specify and store plot definitions, i.e. specifications of which property of which variable should be plotted when and where. Such plot definitions are independent of the existence of the corresponding states. Only when a plot definition is selected for plotting the data to the screen or to a file or for listing the data to a file, the variables and functions used in the plot definition are evaluated using calculated states. This strategy allows the user, after carefully preparing his or her plot definitions, to very efficiently produce results for different model versions to be compared.

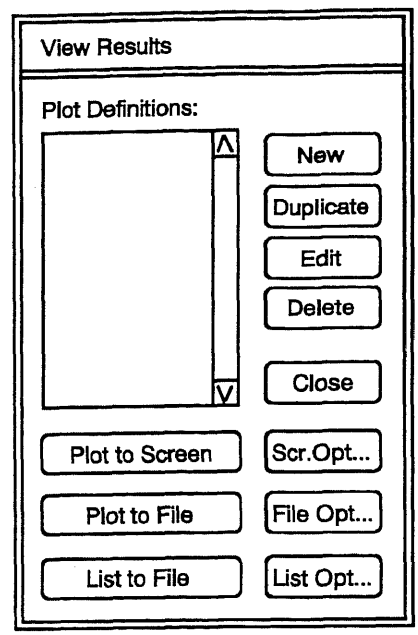


Fig. A5.2: Dialog box for editing plot definitions and plotting and listing results.

Fig. A5.2 shows the dialog box used for editing the set of plot definitions and for realizing plots or listings of plot data with the aid of these definitions. The list box in this dialog box contains an alphabetical list of the names of all plot definitions already specified. The set of plot definitions can be edited using the buttons *New*, *Duplicate*, *Edit* and *Delete*. The buttons *Duplicate*, *Edit* and *Delete* are only active, if a plot definition is selected in the list box. Pressing one of the buttons *New*, *Duplicate* or *Edit* results in opening of the dialog box used for editing the definition of a single plot shown in Fig. A5.3. Each plot definition is identified by its ***Name***. A name of a plot defintion consists of a sequence of letters (A-Z,a-z), digits (0-9) and underline characters (_). The

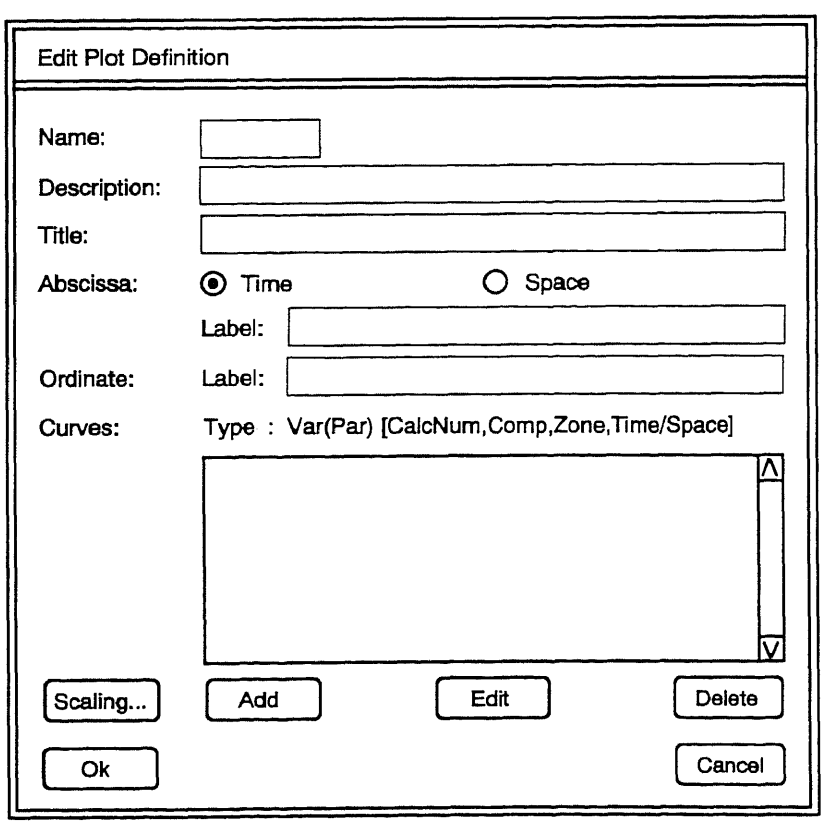


Fig. A5.3: Dialog box for editing a single plot.

first character may not be a digit. To improve documentation of plots, a ***Description*** can be given optionally. For each plot, a ***Title*** can be specified. The ***Abscissa*** can be selected to be *Time* or the *Space* coordinate of the compartment. For the abscissa as well as for the ordinate, an arbitrary ***Label*** can be specified. A plot definition can contain an arbitrary number of curve definitions. The list box of the dialog box shown in Fig. A5.3 shows all curve definitions specified so far. Each line shows the most important attributes of the corresponding curve. The set of curve definitions can be edited using the buttons *Add*, *Edit* and *Delete*. The buttons *Edit* and *Delete* are inactive as long as no curve is selected in the list box. If a curve is selected, pressing the button *Add* inserts a new curve immediately before the selected curve; if no curve is selected, the new curve is appended at the end of the list. As a last possibility, the button ***Scaling*** allows to select ranges and ticks of abscissa and ordinate (cf. more detailed description below).

Fig. A5.4 shows the dialog box used for the definition of a curve in a plot. This dialog box appears after pressing one of the buttons *Add* or *Edit* shown in Fig. A5.3. The ***Type*** of data to be plotted can be the *Value* of the *Variable* (in this case, the *Parameter* entry is irrelevant and may be left blank), the *Error Contribution* of a selected *Parameter* to the uncertainty of the *Variable* according to equation (A4.3) or one of the *Sensitivity Functions* (A4.1a) to (A4.1d) of the *Variable* with respect to a selected *Parameter*. The choice of the sensitivity function is done in the following way: The radio button *AbsAbs* corresponds to the sensitivity function (A4.1a), *RelAbs* to (A4.1b), *AbsRel* to (A4.1c) and *RelRel* to (A4.1d). If the value of the variable is selected to be plotted, uncertainty intervals according to equation (A4.2) are added automatically to the plotted curves, if a sensitivity analysis has been performed. The data plotted on the ordinate belongs to a ***Variable***, a ***Parameter*** (not necessary for curves of type value), a calculation identified by its ***Calculation Number***, a ***Compartment*** and a ***Zone***, in which the variable has to be evaluated and a ***Time*** at which a spatial profile has to be evaluated or a ***Space*** coordinate at which a time series has to be evaluated (depending on the type of abscissa selected in the dialog box shown in Fig. A5.3). For each curve, a ***Legend*** entry can be

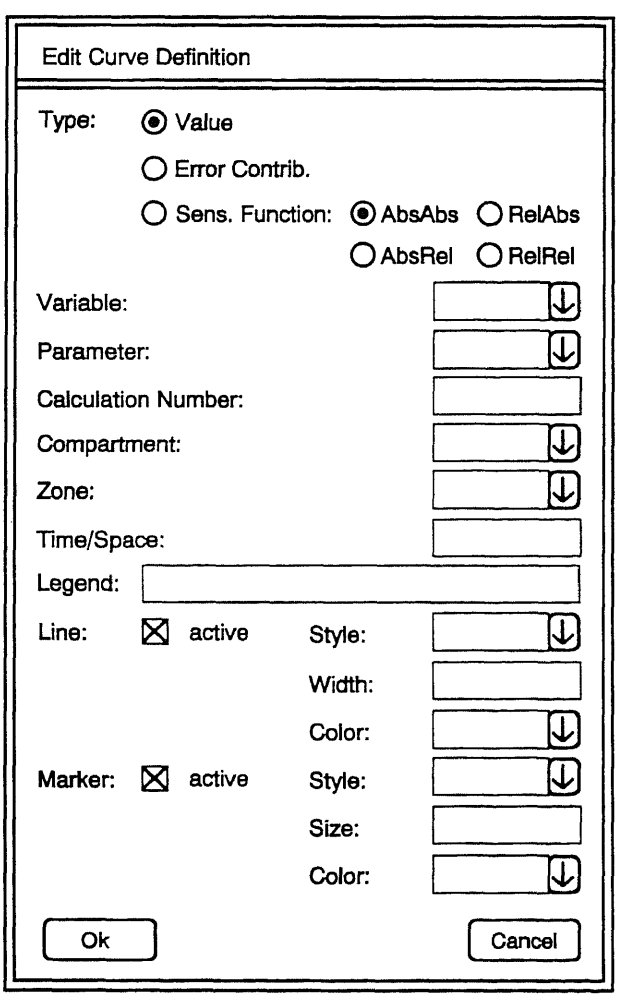


Fig. A5.4: Dialog box for editing a curve in a plot.

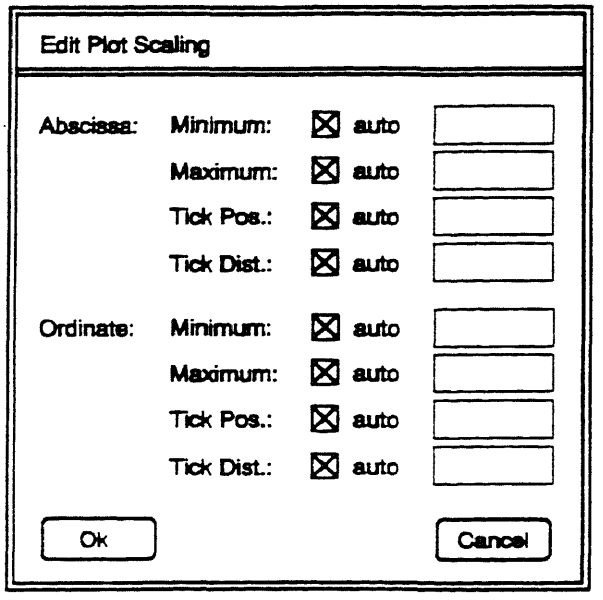


Fig. A5.5: Dialog box for editing plot scaling.

given. The appearance of the curve can be determined by the *Style*, *Width* and *Color* attributes of the ***Line*** linearly connecting calculated or measured values and by the *Style*, *Size* and *Color* attributes of the ***Markers*** showing the calculated or measured data pairs. Line and markers can individually be activated or inactivated. For a real list variable with the same argument as the abscissa, the original data pairs given in the list are plotted; all other data is plotted at calculated points in time or space (for a comparison of original data of such a real list variable with interpolated values, a formula variable, the algebraic expression of which consists of the name of the real list variable can be defined; two curves plotting the values of these two variables lead to the desired result, if time or space intervals of calculations are much smaller than time or space differences of the real list variable).

The last option of a plot definition shown in Fig. A5.3 allows the user to specify ***Scaling*** of plot axes. Fig. A5.5 shows the dialog box, which is opened when the *Scaling* button of the dialog box shown in Fig. A5.3 is pressed. For both axes, abscissa and ordinate, the user can specify its ***Minimum*** and ***Maximum***, ***Tick Position*** and ***Tick Distance***. It is also possible to let the program *automatically* search for reasonable values for these options.

The upper part of the dialog box shown in Fig. A5.2 is used for editing the set of plot definitions, which is stored on the system file together with model definition. If the simulations required for calculating the curves of a plot definition have been performed, the plot selected in the list box shown in Fig. A5.2 can be plotted to the screen, plotted to a file or listed to a file. For each of these tasks, options can be specified.

Pressing the button ***Plot to Screen*** results in displaying axes, legend and all available curves of a plot in a separate window on the screen. Pressing the

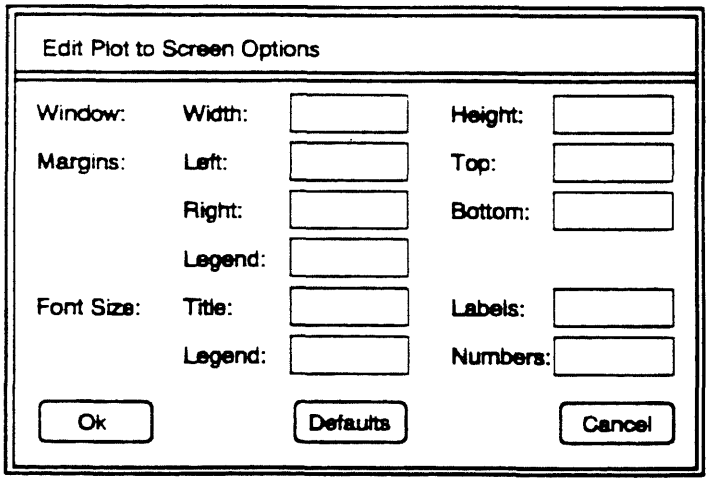


Fig. A5.6: Dialog box for defining options for plotting to the screen.

Options button activates the dialog box shown in Fig. A5.6. The first two options allow the user to specify the initial *Width* and *Height* of the plot window in screen pixels (the size of the windwo can later be changed). Then, the *Margins* and the *Legend Width* can be selected. The legend width should not be too large to let enough space for the plot. The next set of options allows the user to specify the *Font Size* of various classes of text. The button *Defaults* resets all definitions to their default values. Note, that the task of plotting to the screen is not available in the character interface version of AQUASIM, because this version is designed for use on a terminal without graphical capabilities.

The button ***Plot to File*** allows the user to specify the name of a file, to which the plot is written in PostScript format. This file has then to be submitted to a PostScript printer manually by the user of the program. The ***Options*** available for plotting to a file are shown in Fig. A5.7. The first two options allow the user to select the *Paper Size* and *Orientation*. Then, the *Margins* can be specified. As a next option, the numbers of *Columns* and *Rows* of plots to be written on one sheet of paper and the *Legend Width* can be selected. The legend width should not be too large to let enough space for the plots. The next option allows the user to specify the *Font Size* of various classes of text. The next option is used to select between color and black/white *Printing*. Color plots can also be printed on a black/white printer; the color of a curve is then imitated by a greyscale of the line. The last option gives the user the choice of units for the options *Margins* and *Legend Width*. The button *Defaults* resets all definitions to their default values. Consecutive plots during an AQUASIM session written to the same file are plotted on the same page until the number of plots given as the product of *Columns* times *Rows* is reached, if no change of plot options has been done (even opening the options dialog box and pressing *Ok* leads to starting of a new page).

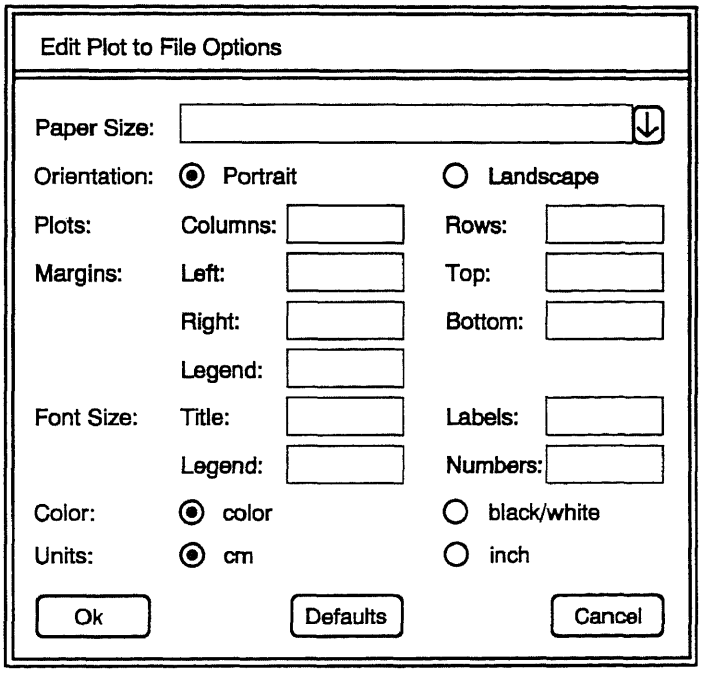


Fig. A5.7: Dialog box for defining options for PostScript output plot files.

The last task of processing results can be performed by pressing the button ***List to File*** of the dialog box shown in Fig. A5.2. This action allows the user to specify a name of a file to which plot data is written in text format. Plot data is appended to this file, so that data of several plots can be written to the same file. This task has been designed to allow the user to postprocess AQUASIM results with the aid of other programs. The ***Options*** available for listing of results are shown in Fig. A5.8. The

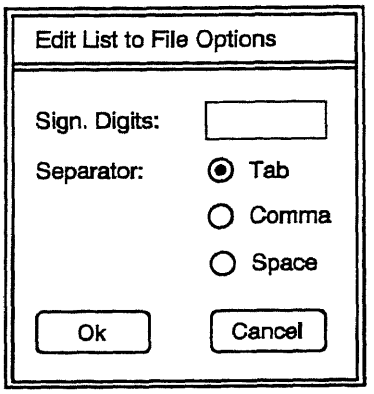


Fig. A5.8: Dialog box for defining text output format.

number of *Significant Digits* of the numbers written to the file and the *Separator* between columns of numbers can be selected. These options should allow the user to select the data format appropriate for the program used for postprocessing the results.

A6 Tutorial

This tutorial consists of a set of exercises, each of which solves a simple modeling problem with the aid of AQUASIM. For each exercise, after a short description of the problem, the solution is explained in detail. All steps required for model specification, simulation and visualization of results are listed and comments are given if necessary. All of the exercises are self-contained; each of them explaining selected AQUASIM elements. The following Table A6.1 gives a survey on which AQUASIM elements are discussed in which exercise. This table allows an interested user to select the example which uses the program elements to be learned. For beginners, it is recommended to study all exercises. Note that this tutorial only treats the most important features of AQUASIM. Consult chapters A2 - A5 for a complete description of all features.

Table A6.1: Survey of AQUASIM elements treated in the examples

Category	Element	Exercise					
		1	2	3	4	5	6
Model Elements	Dynamic Volume State Variables	x	x	x	x	x	x
	Dynamic Surface State Variables			x			
	Equilibrium State Variables			x			
	Program Variables	x		x	x	x	x
	Constant Variables				x		
	Real List Variables			x	x		x
	Variable List Variables				x		
	Formula Variables	x	x	x		x	x
	Dynamic Processes		x	x	x	x	
	Equilibrium Processes			x			
	Mixed Reactor Compartments	x	x	x	x		
	Biofilm Reactor Compartments					x	
	River Section Compartments						x
	Advective Links	x					
	Diffusive Links		x				
Program Tasks	Simulation	x	x	x	x	x	x
	Sensitivity Analysis				x		
	Parameter Estimation				x		
Results	Plot	x	x	x	x	x	x

The descriptions of the solutions to the following exercises only treat AQUASIM-specific features. It is assumed that the reader is familiar with the window system of the platform he or she is using. Additionally, elementary knowledge of chemical process engineering is assumed. All solutions are given for the window interface version of AQUASIM. In section A7.2 the solution of exercise 2a is also given for the character interface version.

A6.1 Exercise 1: Reservoir Series

This exercise introduces into elementary program handling by comparing water flow and substance transport in reservoirs in series.

Part A: Five reactors with variable volume are advectively connected in series (the outflow of the first reactor is connected to the inflow of the second reactor etc.). The outflow of each reactor is proportional to its volume with a rate constant of 1/hour. The initial volume of the first reactor is 1 liter, the initial volumes of the other reactors are negligibly small (10^{-6} liters). Plot the time courses of the volumes of all reactors for a time period of 10 hours.

Part B: Five reactors with fixed volumes of 1 liter each are advectively connected in series. Initially, the first reactor contains 1 mg/l of a substance, the other reactors contain pure water. The inflow to the first reactor is 1 l/h of pure water. Plot the time courses of the concentration of the substance in all reactors for a time period of 10 hours.

Compare the results of parts A and B and discuss similarities and differences.

Solution to Part A:

Start the window interface version of AQUASIM or select File → New to remove previously entered data from memory.

Definition of Variables

For the formulation of this model the actual reactor volume must be made available and it is recommendable to specify the outflow rate constant as a variable:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **V** Description: **Reactor volume** Unit: **l** Reference to: **Reactor Volume** **Ok**	this variable makes reactor volume available
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **k** Description: **Rate constant** Unit: **1/h** Expression: **1** **Ok**	formula variables can be used to define constant parameters
Menu Bar	A1.1/A2.1	**File → Save**	
File Save Dialog	-	File Name: **ex1a.aqu** **Ok**	save definitions to "ex1a.aqu"
Confirmation Message	-	**Ok**	

Definition of Compartments

Five mixed reactor compartments with variable volume and correct initial conditions are required:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	**New**	
Select Compartment Type	A3.21	**Mixed Reactor Comparment** **Ok**	
Edit Mixed React. Comp.	A3.29	Name: **Res_1** Description: **Reservoir 1** **Init. Cond.**	
Edit Initial Conditions	A3.25	**Add**	
Edit Initial Condition	A3.26	Variable: **V** Init. Cond.: **1** **Ok**	initial reactor volume is 1 liter
Edit Initial Conditions	A3.25	**Ok**	
Edit Mixed React. Comp.	A3.29	React. Type: **variable volume** Outflow: **k*V** **Ok**	specify type and outflow
Edit Compartments	A3.20	Select **Res_1** **Duplicate**	
Edit Mixed React. Comp.	A3.29	Name: **Res_2** Description: **Reservoir 2** **Init. Cond.**	define reservoir 2 by duplication and editing
Edit Initial Conditions	A3.25	Select **V(Bulk Volume) : 1** **Edit**	
Edit Initial Condition	A3.26	Init. Cond.: **1e-6** **Ok**	initial reactor volume is 1e-6 l
Edit Initial Conditions	A3.25	**Ok**	
Edit Mixed React. Comp.	A3.29	**Ok**	
Edit Compartments	A3.20	Select **Res_2** **Duplicate**	
Edit Mixed React. Comp.	A3.29	Name: **Res_3** Description: **Reservoir 3** **Ok**	define reservoir 3 by duplication and editing
Edit Compartments	A3.20	Select **Res_2** **Duplicate**	
Edit Mixed React. Comp.	A3.29	Name: **Res_4** Description: **Reservoir 4** **Ok**	define reservoir 4 by duplication and editing
Edit Compartments	A3.20	Select **Res_2** **Duplicate**	
Edit Mixed React. Comp.	A3.29	Name: **Res_5** Description: **Reservoir 5** **Ok**	define reservoir 5 by duplication and editing
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Definition of Links

The reactors defined in the previous step must be connected by advective links:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Links**	
Edit Links	A3.42	**New**	
Select Link Type	A3.43	**Advective Link** **Ok**	
Edit Advective Link	A3.45	Name: **Link_12** Comp. In: **Res_1** Comp. Out: **Res_2** **Ok**	connect outflow of Res_1 to in-flow of Res_2
Edit Links	A3.42	Select **Link_12** **Duplicate**	
Edit Advective Link	A3.45	Name: **Link_23** Comp. In: **Res_2**	connect outflow of Res_2 to in

		Comp. Out: **Res_3** **Ok**	flow of Res_3
Edit Links	A3.42	Select **Link_12** **Duplicate**	
Edit Advective Link	A3.45	Name: **Link_34** Comp. In: **Res_3** Comp. Out: **Res_4** **Ok**	connect outflow of Res_3 to inflow of Res_4
Edit Links	A3.42	Select **Link_12** **Duplicate**	
Edit Advective Link	A3.45	Name: **Link_45** Comp. In: **Res_4** Comp. Out: **Res_5** **Ok**	connect outflow of Res_4 to inflow of Res_5
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Definition of Plots

Graphical presentation of results requires a plot definition. Once a plot definition has been specified, it can be used for several calculations with different parameters or model structures. It is advantageous to specify the plot definitions before performing the calculation:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	**New**	
Edit Plot Definition	A5.3	Name: **Volumes** Title: **Reservoir Volumes** Absc. Label: **time [h]** Ord. Label: **volume [l]** **Add**	define general plot attributes
Edit Curve Definition	A5.4	Variable: **V** Compart.: **Res_1** Legend: **Res_1** Line Style: **solid** Line Color: **black** **Ok**	curve for volume of Res_1 (let the other items at their default values)
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **V** Compart.: **Res_2** Legend: **Res_2** Line Style: **long dashed** Line Color: **red** **Ok**	curve for volume of Res_2
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **V** Compart.: **Res_3** Legend: **Res_3** Line Style: **dashed** Line Color: **green** **Ok**	curve for volume of Res_3
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **V** Compart.: **Res_4** Legend: **Res_4** Line Style: **short dashed** Line Color: **blue** **Ok**	curve for volume of Res_4
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **V** Compart.: **Res_5** Legend: **Res_5** Line Style: **dotted** Line Color: **cyan** **Ok**	curve for volume of Res_5
Edit Plot Definition	A5.3	**Ok**	

Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation over 10 hours can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **0.1** Num. Steps: **100** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Volumes** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **Volumes** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex1a.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex1a.ps" can be submitted to a printer in a system dependent way. Fig. A6.1 shows the plot as it is printed by processing the file "ex1a.ps". This plot shows the time courses of the volumes of all five reactors.

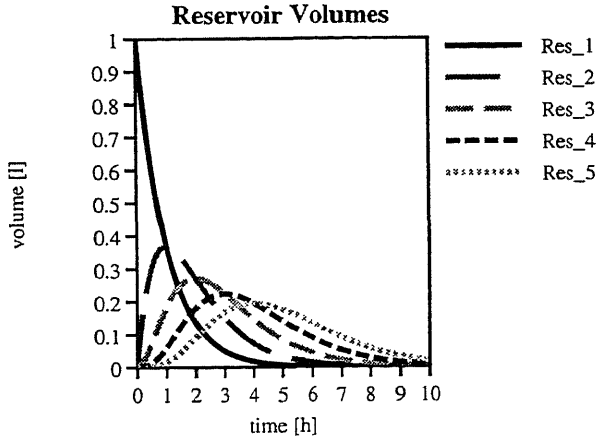


Fig. A6.1: Plot printed by processing the file "ex1a.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	

File Save Dialog	-	File Name: **ex1a.prn** **Ok**	write system definitions to "ex1a.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex1a.aqu". The system definitions have been written to the text file "ex1a.prn" which can be submitted to a printer in order to check system definitions.

Solution to Part B:

Start the window interface version of AQUASIM or select File → New to remove previously entered data from memory.

Definition of Variables

For the formulation of this model a dynamic volume state variable representing concentration of substance A is required:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **C** Description: **Concentration** Unit: **mg/l** **Ok**	dynamic volume state variable describing concentration
Menu Bar	A1.1/A2.1	**File → Save As**	
File Save Dialog	-	File Name: **ex1b.aqu** **Ok**	save definitions to "ex1b.aqu"
Confirmation Message	-	**Ok**	

Definition of Compartments

Five mixed reactor compartments with fixed volume and correct initial conditions are required:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	**New**	
Select Compartment Type	A3.21	**Mixed Reactor Comparment** **Ok**	
Edit Mixed React. Comp.	A3.29	Name: **Res_1** Description: **Reservoir 1** **Variables**	
Select Active Variables	A3.23	Select **C** **Activate** **Ok**	activate the state variable C
Edit Mixed React. Comp.	A3.29	**Init. Cond.**	
Edit Initial Conditions	A3.25	**Add**	
Edit Initial Condition	A3.26	Variable: **C** Init. Cond.: **1** **Ok**	initial concentration of C in Res_1 is 1 mg/l
Edit Initial Conditions	A3.25	**Ok**	
Edit Mixed React. Comp.	A3.29	**Input**	
Edit Inputs	A3.27	Water Inflow: **1** **Ok**	water inflow to Res_1 is 1 l/h
Edit Mixed React. Comp.	A3.29	Volume: **1** **Ok**	specify volume
Edit Compartments	A3.20	Select **Res_1** **Duplicate**	
Edit Mixed React. Comp.	A3.29	Name: **Res_2** Description: **Reservoir 2** **Init. Cond.**	define reservoir 2 by duplication and editing

Edit Initial Conditions	A3.25	Select **C(Bulk Volume) : 1** **Delete** **Ok**	initial concentration in Res_2 is 0
Edit Mixed React. Comp.	A3.29	**Input**	
Edit Inputs	A3.27	Water Inflow: **0** **Ok**	water inflow to Res_2 is 0
Edit Mixed React. Comp.	A3.29	**Ok**	
Edit Compartments	A3.20	Select **Res_2** **Duplicate**	
Edit Mixed React. Comp.	A3.29	Name: **Res_3** Description: **Reservoir 3** **Ok**	define reservoir 3 by duplication and editing
Edit Compartments	A3.20	Select **Res_2** **Duplicate**	
Edit Mixed React. Comp.	A3.29	Name: **Res_4** Description: **Reservoir 4** **Ok**	define reservoir 4 by duplication and editing
Edit Compartments	A3.20	Select **Res_2** **Duplicate**	
Edit Mixed React. Comp.	A3.29	Name: **Res_5** Description: **Reservoir 5** **Ok**	define reservoir 5 by duplication and editing
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Definition of Links

The reactors defined in the previous step must be connected by advective links:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Links**	
Edit Links	A3.42	**New**	
Select Link Type	A3.43	**Advective Link** **Ok**	
Edit Advective Link	A3.45	Name: **Link_12** Comp. In: **Res_1** Comp. Out: **Res_2** **Ok**	connect outflow of Res_1 to inflow of Res_2
Edit Links	A3.42	Select **Link_12** **Duplicate**	
Edit Advective Link	A3.45	Name: **Link_23** Comp. In: **Res_2** Comp. Out: **Res_3** **Ok**	connect outflow of Res_2 to inflow of Res_3
Edit Links	A3.42	Select **Link_12** **Duplicate**	
Edit Advective Link	A3.45	Name: **Link_34** Comp. In: **Res_3** Comp. Out: **Res_4** **Ok**	connect outflow of Res_3 to inflow of Res_4
Edit Links	A3.42	Select **Link_12** **Duplicate**	
Edit Advective Link	A3.45	Name: **Link_45** Comp. In: **Res_4** Comp. Out: **Res_5** **Ok**	connect outflow of Res_4 to inflow of Res_5
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Definition of Plots

Graphical presentation of results requires a plot definition. Once a plot definition has been specified, it can be used for several calculations with different parameters or model structures. It is advantageous to specify the plot definitions before performing the calculation:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	**New**	
Edit Plot Definition	A5.3	Name: **Conc** Title: **Concentrations** Absc. Label: **time [h]** Ord. Label: **C [mg/l]** **Add**	define general plot attributes
Edit Curve Definition	A5.4	Variable: **C** Compart.: **Res_1** Legend: **Res_1** Line Style: **solid** Line Color: **black** **Ok**	curve for concentration in Res_1
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C** Compart.: **Res_2** Legend: **Res_2** Line Style: **long dashed** Line Color: **red** **Ok**	curve for concentration in Res_2
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C** Compart.: **Res_3** Legend: **Res_3** Line Style: **dashed** Line Color: **green** **Ok**	curve for concentration in Res_3
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C** Compart.: **Res_4** Legend: **Res_4** Line Style: **short dashed** Line Color: **blue** **Ok**	curve for concentration in Res_4
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C** Compart.: **Res_5** Legend: **Res_5** Line Style: **dotted** Line Color: **cyan** **Ok**	curve for concentration in Res_5
Edit Plot Definition	A5.3	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation over 10 hours can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **0.1** Num. Steps: **100** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	

View Results	A5.2	Select **Conc** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **Conc** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex1b.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex1b.ps" can be submitted to a printer in a system dependent way. Fig. A6.2 shows the plot as it is printed by processing the file "ex1b.ps". This plot is identical to that created in part A because, despite of the different physical interpretation, the same differential equations are solved in parts A and B of this example.

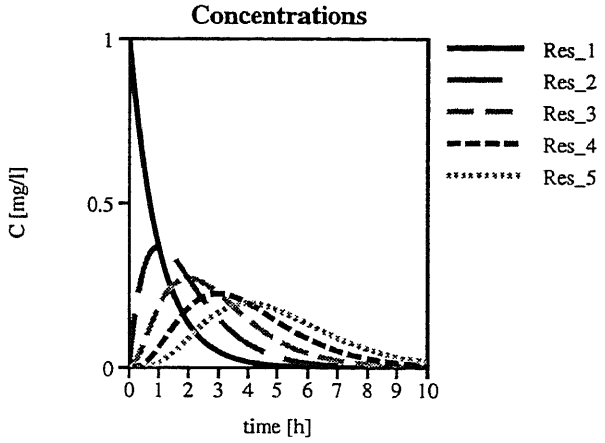


Fig. A6.2: Plot printed by processing the file "ex1b.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex1b.prn** **Ok**	write system definitions to "ex1b.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex1b.aqu". The system definitions have been written to the text file "ex1b.prn" which can be submitted to a printer in order to check system definitions.

A6.2 Exercise 2: Chemical Processes in a Batch Reactor

This exercise introduces implementation of chemical processes.

Part A: In a stirred batch reactor with a volume of 10 liters a substance A is degraded by a first order process with a rate constant of 1/hour. The initial concentration of substance A is 1 mg/l. Plot the time courses of the concentration of substance A for a degradation experiment with a duration of 10 hours.

Part B: Substance A is not completely degraded, but it is converted to substance B with a fixed stoichiometry of 2:1 (1 mg of A is converted to 0.5 mg of B). Substance B is converted to substance C with a nonlinear transformation rate according to Monod. Maximum conversion rate is 0.25 mg/l/h and half-saturation concentration is 0.5 mg/l of substance B. The stoichiometric ratio of this conversion step is 1:2 (1 mg of B is converted to 2 mg of C). Plot the time courses of the substances A, B and C for 10 hours assuming initial concentrations of B and C to be zero.

Part C: The water volume of the reactor of 10 liters is in connection to a gas volume of additional 10 liters. Substances A and B cannot be present in the gas phase but substance C escapes to the gas phase. The non-dimensional Henry coefficient of substance C is 2 (equilibrium concentration in the gas phase is twice the concentration in water). The mass exchange coefficient between water and air is increased to 1 m^3/h with the aid of a stirrer. Plot the time courses of the concentrations of the substances A, B and C in the water phase and of substance C in the gas phase for 30 hours assuming initial concentration of C in the gas phase to be zero.

Solution to Part A:

Start the window interface version of AQUASIM or select File → New to remove previously entered data from memory.

Definition of Variables

For the formulation of this model a state variable representing the concentration of substance A in the reactor is required and it is recommendable to specify the degradation rate constant as a variable:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **C_A** Description: **Concentration of A** Unit: **mg/l** **Ok**	dynamic volume state variable describing concentration of A
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **k_A** Description: **First order degr. rate constant of A**	formula variables can be used to define

		Unit: **1/h** Expression: **1** **Ok**	constant parameters
Menu Bar	A1.1/A2.1	**File → Save**	
File Save Dialog	-	File Name: **ex2a.aqu** **Ok**	save definitions to "ex2a.aqu"
Confirmation Message	-	**Ok**	

Definition of Processes

Degradation of substance A is formulated as a dynamic process:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Processes**	
Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Dynamic Process** **Ok**	
Edit Dynamic Process	A3.17	Name: **Degr_A** Description: **Degradation of A** Rate: **k_A*C_A** **Add**	dynamic process with first order rate
Edit Stoich. Coefficient	A3.18	Variable: **C_A** St. Coeff.: **-1** **Ok**	A is degraded with the specified rate
Edit Dynamic Process	A3.17	**Ok**	

Definition of Compartments

A single mixed reactor compartment representing the batch reactor is required:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	**New**	
Select Compartment Type	A3.21	**Mixed Reactor Comparment** **Ok**	
Edit Mixed React. Comp.	A3.29	Name: **Reactor** Description: **Batch reactor** **Variables**	
Select Active Variables	A3.23	Select **C_A** **Activate** **Ok**	activate the state variable C_A
Edit Mixed React. Comp.	A3.29	**Processes**	
Select Active Processes	A3.24	Select **Degr_A** **Activate** **Ok**	activate the process Degr_A
Edit Mixed React. Comp.	A3.29	**Init. Cond.**	
Edit Initial Conditions	A3.25	**Add**	
Edit Initial Condition	A3.26	Variable: **C_A** Init. Cond.: **1** **Ok**	initial concentration of A is 1 mg/l
Edit Initial Conditions	A3.25	**Ok**	
Edit Mixed React. Comp.	A3.29	Volume: **10** **Ok**	specify volume

Definition of Plots

Graphical presentation of results requires a plot definition. Once a plot definition has been specified, it can be used for several calculations with different parameters or model structures. It is advantageous to specify the plot definitions before performing the calculation:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	**New**	

Edit Plot Definition	A5.3	Name: **Conc** Title: **Degradation of A** Absc. Label: **time [h]** Ord. Label: **C [mg/l]** **Add**	define general plot attributes
Edit Curve Definition	A5.4	Variable: **C_A** Legend: **A** Line Style: **solid** Line Color: **red** **Ok**	curve for concentration of A (let the other items at their default values)
Edit Plot Definition	A5.3	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation over 10 hours can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **0.1** Num. Steps: **100** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **Conc** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex2a.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex2a.ps" can be submitted to a printer in a system dependent way. Fig. A6.3 shows the plot as it is printed by processing the file "ex2a.ps". This plot shows the exponential decay of substance A.

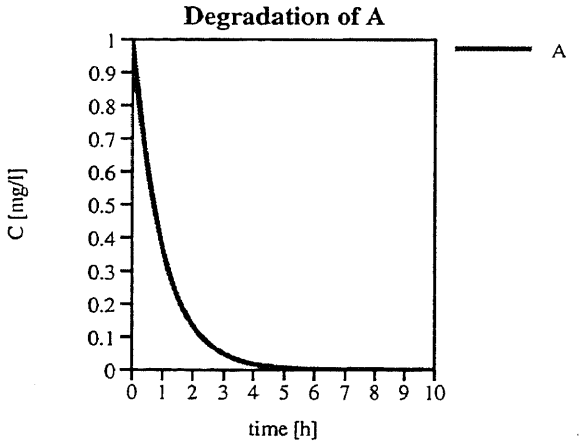


Fig. A6.3: Plot printed by processing the file "ex2a.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex2a.prn** **Ok**	write system definitions to "ex2a.prn"
Confirmation Message	-	**Ok**	

Now, the system can be reloaded from the file "ex2a.aqu". The system definitions have been written to the text file "ex2a.prn" which can be submitted to a printer in order to check system definitions.

Solution to Part B:

Continue your work after you have performed part A or start the window interface version of AQUASIM and load the file "ex2a.aqu" saved in part A with the commands File → Open, File Name: exp2a.aqu, Ok, Ok.

Definition of Variables

Two additional state variables for the concentrations of substances B and C must be added and it is recommendable to specify the two new parameters of the conversion process from B to C as variables:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **C_B** Description: **Concentration of B** Unit: **mg/l** **Ok**	dynamic volume state variable describing concentration of B
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **C_C** Description: **Concentration of C** Unit: **mg/l** **Ok**	dynamic volume state variable describing concentration of C
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **rmax_B** Description: **Maximum degradation rate of B** Unit: **mg/l/h** Expression: **0.25** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	

Edit Formula Variable	A3.13	Name: **K_B** Description: **Half-saturation concentration of B** Unit: **mg/l** Expression: **0.5** **Ok**	formula variables can be used to define constant parameters
Menu Bar	A1.1/A2.1	**File → Save As**	
File Save Dialog	-	File Name: **ex2b.aqu** **Ok**	save definitions to "ex2b.aqu"
Confirmation Message	-	**Ok**	

Definition of Processes

Degradation of substance A must be revised to include the production of substance B and an additional process converting substance B into substance C must be added:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Processes**	
Edit Processes	A3.14	Select **Degr_A** **Edit**	
Edit Dynamic Process	A3.17	**Add**	
Edit Stoich. Coefficient	A3.18	Variable: **C_B** St. Coeff.: **0.5** **Ok**	B is produced with half of the specified rate
Edit Dynamic Process	A3.17	**Ok**	
Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Dynamic Process** **Ok**	
Edit Dynamic Process	A3.17	Name: **Degr_B** Description: **Degradation of B** Rate: **rmax_B *C_B/(K_B+C_B)** **Add**	dynamic process with Monod type rate
Edit Stoich. Coefficient	A3.18	Variable: **C_B** St. Coeff.: **-1** **Ok**	B is degraded with the specified rate
Edit Dynamic Process	A3.17	**Add**	
Edit Stoich. Coefficient	A3.18	Variable: **C_C** St. Coeff.: **2** **Ok**	C id produced with twice the specified rate
Edit Dynamic Process	A3.17	**Ok**	

Definition of Compartments

New state variables and processes defined in the previous steps must be activated:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **Reactor** **Edit**	
Edit Mixed React. Comp.	A3.29	**Variables**	
Select Active Variables	A3.23	Select **C_B** **Activate** Select **C_C** **Activate** **Ok**	activate the state variables C_B and C_C
Edit Mixed React. Comp.	A3.29	**Processes**	
Select Active Processes	A3.24	Select **Degr_B** **Activate** **Ok**	activate the process Degr_B
Edit Mixed React. Comp.	A3.29	**Ok**	

Definition of Plots

The plot definition must be modified to include the concentrations of the additional substances:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc** **Edit**	
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C_B** Legend: **B** Line Style: **long dashed** Line Color: **green** **Ok**	curve for concentration of B
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C_C** Legend: **C** Line Style: **dashed** Line Color: **blue** **Ok**	curve for concentration of C
Edit Plot Definition	A5.3	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation over 10 hours can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **0.1** Num. Steps: **100** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **Conc** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex2b.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex2b.ps" can be submitted to a printer in a system dependent way. Fig. A6.4 shows the plot as it is printed by processing the file "ex2b.ps". This plot shows decay of substance A, production and decay of substance B and production of substance C. At the end of the simulation nearly all of the substance A is completely converted to substance C.

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete** **Ok**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	

Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex2b.prn** **Ok**	write system definitions to "ex2b.prn"
Confirmation Message	-	**Ok**	

Now, the system can be reloaded from the file "ex2b.aqu". The system definitions have been written to the text file "ex2b.prn" which can be submitted to a printer in order to check system definitions.

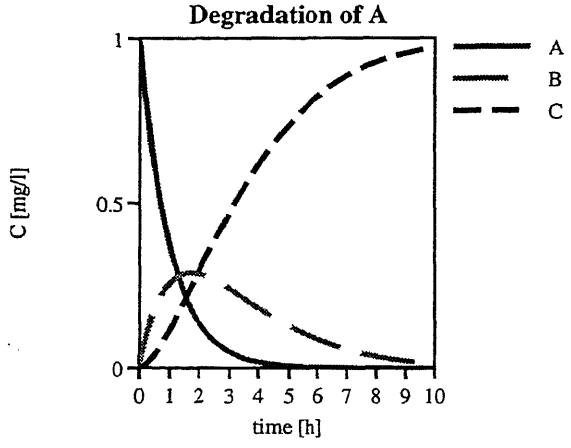


Fig. A6.4: Plot printed by processing the file "ex2b.ps" created in this example

Solution to Part C:

Continue your work after you have performed part B or start the window interface version of AQUASIM and load the file "ex2b.aqu" saved in part B with the commands File → Open, File Name: exp2b.aqu, Ok, Ok.

Definition of Variables

It is recommendable to specify the Henry coefficient of substance C as an additional variable:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **H** Description: **Henry coeff. of C** Unit: Expression: **2** **Ok**	formula variables can be used to define constant parameters
Menu Bar	A1.1/A2.1	**File → Save As**	
File Save Dialog	-	File Name: **ex2c.aqu** **Ok**	save definitions to "ex2c.aqu"
Confirmation Message	-	**Ok**	

Definition of Compartments

A new compartment describing gas volume is required:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	**New**	

Select Compartment Type	A3.21	**Mixed Reactor Comparment** **Ok**	
Edit Mixed React. Comp.	A3.29	Name: **GasVolume** Description: **Gas volume** **Variables**	mixed reactor compartment for gas volume
Select Active Variables	A3.23	Select **C_C** **Activate** **Ok**	activate the state variable C_C
Edit Mixed React. Comp.	A3.29	Volume: **10** **Ok**	specify volume

Definition of Links

A diffusive link represents the molecular boundary layer at the water surface:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Links**	
Edit Links	A3.42	**New**	
Select Link Type	A3.43	**Diffusive Link** **Ok**	
Edit Diffusive Link	A3.49	Name: **Layer** Comp. 1: **GasVolume** Comp. 2: **Reactor** **Add**	connect the two compartments diffusively
Edit Transfer Coefficient	A3.50	Variable: **C_C** Exch. Coeff.: **1** Conv. Fa. 1: **1/H** **Ok**	specify exchange coefficient and conversion factor
Edit Diffusive Link	A3.49	**Ok**	

Definition of Plots

The plot definition must be modified to include the concentration of substance C in the gas phase:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc** **Edit**	
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C_C** Compart.: **GasVolume** Legend: **Cgas** Line Style: **short dashed** Line Color: **blue** **Ok**	curve for concentration of C in the gas phase
Edit Plot Definition	A5.3	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation over 10 hours can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **0.1** Num. Steps: **300** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc** **Plot to Screen**	plot results to the screen

View Results	A5.2	Select **Conc** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex2c.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex2c.ps" can be submitted to a printer in a system dependent way. Fig. A6.5 shows the plot as it is printed by processing the file "ex2c.ps". At the end of the simulation, A is nearly completely converted to C and 2/3 of C is in the gas phase, 1/3 in the water phase.

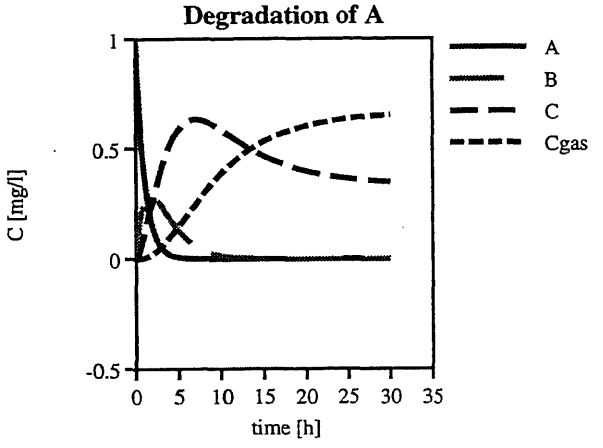


Fig. A6.5: Plot printed by processing the file "ex2c.ps" created in this example

Saving and Printing System Definitions

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete** **Ok**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex2c.prn** **Ok**	write system definitions to "ex2c.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex2c.aqu". The system definitions have been written to the text file "ex2c.prn" which can be submitted to a printer in order to check the system definitions.

A6.3 Exercise 3: Growth of Sessile Organisms

This exercise introduces interactions between dissolved, suspended and attached substances.

Part A: Water flowing into a mixed reactor with a volume of 10 liters at a rate of 10 l/h contains 1 mg/l of substance A. This substance is the substrate of organisms B, which are attached to surfaces within the reactor. 5 mg of consumption of substrate A leads to production of 1 mg of organisms B. Growth rate of the organisms is assumed to be proportional to the mass of organisms B multiplied by a Monod term describing substrate dependence. Maximum growth rate of the organisms is assumed to be 0.1/h and half-saturation concentration of substrate dependence is 0.5 mg/l. Plot the time courses of the concentration of substrate A and of the mass of attached organisms B for initial conditions of zero for substance A and of 1 mg for organisms B for an experiment with a duration of 50 hours.

Part B: Consider the effect of a detachment process which transfers attached organisms into suspension (neglect substrate consumption of suspended organisms). To consider weakening of connecting forces with increasing biofilm depth, assume the detachment process rate to be proportional to the square of the mass of organisms B with a coefficient of 0.01 1/mg/h. Redo the calculation prepared for section A.

Part C: Substrate A exists in two different species. The ratio of the concentration of species 1 to that of species 2 depends linearly (factor of 2.3) on a property P measured as a time series during the experiment (0 h: 1.1; 10 h: 1.3; 20 h: 1.4; 50 h: 1.5). Interpolate linearly between these data points and assume the organisms to grow on species 1 only. Add the concentrations of species 1 and 2 to the plot and redo the calculation.

Solution to Part A:

Start the window interface version of AQUASIM or select File → New to remove previously entered data from memory.

Definition of Variables

For the formulation of this model, state variables for the concentration of substance A and for the mass of organisms B must be defined. Furthermore, the reactor volume must be made available for converting transformation rates from dissolved concentrations to attached masses. It is recommendable to specify the model parameters, water inflow and inflow concentration as variables:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **C_A** Description: **Concentration of A** Unit: **mg/l** **Ok**	dynamic volume state variable describing concentration of A
Edit Variables	A3.3	**New**	

Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **M_B** Description: **Mass of B** Unit: **mg** Type: **dynamic, surface** **Ok**	dynamic surface state variable describing mass of B
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **V** Description: **Reactor volume** Unit: **l** Reference to: **Reactor Volume** **Ok**	this variable makes reactor volume available
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **rmax_B** Description: **Max. growth rate** Unit: **1/h** Expression: **0.1** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **K_A** Description: **Half-saturation concentration of A** Unit: **mg/l** Expression: **0.5** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **Y** Description: **Yield** Unit: **gB/gA** Expression: **0.2** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **Qin** Description: **Water inflow** Unit: **l/h** Expression: **10** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **C_Ain** Description: **Inflow conc. of A** Unit: **mg/l** Expression: **1** **Ok**	formula variables can be used to define constant parameters
Menu Bar	A1.1/A2.1	**File → Save**	
File Save Dialog	-	File Name: **ex3a.aqu** **Ok**	save definitions to "ex3a.aqu"
Confirmation Message	-	**Ok**	

Definition of Processes

Growth of organisms B on substrate A is formulated as a dynamic process:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Processes**	
Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Dynamic Process** **Ok**	
Edit Dynamic Process	A3.17	Name: **Gro_B** Description: **Growth of B** Rate: **C_A/(K_A+C_A) *M_B** **Add**	dynamic process with Monod rate law
Edit Stoich. Coefficient	A3.18	Variable: **M_B** St. Coeff.: **1** **Ok**	B grows with the specified rate
Edit Dynamic Process	A3.17	**Add**	
Edit Stoich. Coefficient	A3.18	Variable: **C_A** St. Coeff.: **-1/V/Y** **Ok**	A is consumed with 1/Y times the specified rate (factor 1/V for conversion of units)
Edit Dynamic Process	A3.17	**Ok**	

Definition of Compartments

A single mixed reactor compartment is required for this exercise:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	**New**	
Select Compartment Type	A3.21	**Mixed Reactor Comparment** **Ok**	
Edit Mixed React. Comp.	A3.29	Name: **Reactor** Description: **Reactor** **Variables**	
Select Active Variables	A3.23	Select **C_A** **Activate** Select **M_B** **Activate** **Ok**	activate the state variables C_A and M_B
Edit Mixed React. Comp.	A3.29	**Processes**	
Select Active Processes	A3.24	Select **Gro_B** **Activate** **Ok**	activate the process Gro_B
Edit Mixed React. Comp.	A3.29	**Init. Cond.**	
Edit Initial Conditions	A3.25	**Add**	
Edit Initial Condition	A3.26	Variable: **M_B** Init. Cond.: **1** **Ok**	initial mass of B is 1 mg
Edit Initial Conditions	A3.25	**Ok**	
Edit Mixed React. Comp.	A3.29	**Input**	
Edit Inputs	A3.27	Water Inflow: **Qin** **Add**	water inflow as specified in Qin
Edit Input Flux	A3.28	Variable: **C_A** Input Flux: **Qin*C_Ain** **Ok**	input flux equal discharge times concentration
Edit Inputs	A3.27	**Ok**	
Edit Mixed React. Comp.	A3.29	Volume: **10** **Ok**	specify volume

Definition of Plots

Graphical presentation of results requires a plot definition. Once a plot definition has been specified, it can be used for several calculations with different parameters or model structures. It is advantageous to specify the plot definitions before performing the calculation:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	**New**	
Edit Plot Definition	A5.3	Name: **ConcA** Title: **Concentration of A** Absc. Label: **time [h]** Ord. Label: **C_A [mg/l]** **Add**	define general plot attributes
Edit Curve Definition	A5.4	Variable: **C_A** Line Style: **solid** Line Color: **red** **Ok**	curve for concentration of A (let the other items at their default values)
Edit Plot Definition	A5.3	**Ok**	
View Results	A5.2	**New**	
Edit Plot Definition	A5.3	Name: **MassB** Title: **Mass of B** Absc. Label: **time [h]** Ord. Label: **M_B [mg]** **Add**	define general plot attributes
Edit Curve Definition	A5.4	Variable: **M_B** Line Style: **solid** Line Color: **green** **Ok**	curve for mass of B
Edit Plot Definition	A5.3	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation over 50 hours can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **0.5** Num. Steps: **100** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **ConcA** **Plot to Screen** Select **MassB** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **ConcA** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex3a.ps** **Ok**	
View Results	A5.2	Select **MassB** **Plot to File**	
File Save Dialog	-	File Name: **ex3a.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex3a.ps" can be submitted to a printer in a system dependent way. Fig. A6.3 shows the plot as it is printed by processing the file "ex3a.ps".

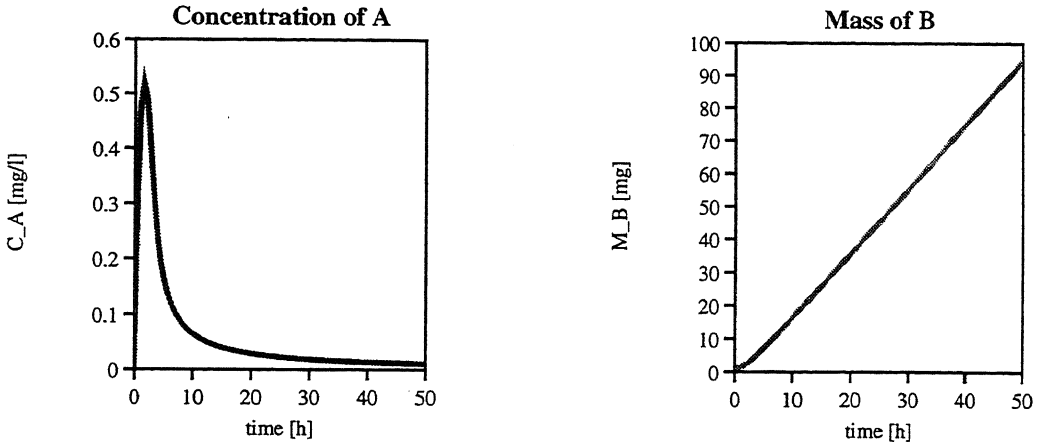


Fig. A6.6: Plot printed by processing the file "ex3a.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex3a.prn** **Ok**	write system definitions to "ex3a.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex3a.aqu". The system definitions have been written to the text file "ex3a.prn" which can be submitted to a printer in order to check the system definitions.

Solution to Part B:

Continue your work after you have performed part A or start the window interface version of AQUASIM and load the file "ex3a.aqu" saved in part A with the commands File → Open, File Name: exp3a.aqu, Ok, Ok.

Definition of Variables

It is advantageous to specify the detachment rate constant as a variable:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **kdet** Description: **Detachment rate constant**	formula variables can be used to define

		Unit: **1/h/mg** Expression: **0.01** **Ok**	constant parameters
Menu Bar	A1.1/A2.1	**File → Save As**	
File Save Dialog	-	File Name: **ex3c.aqu** **Ok**	save definitions to "ex3c.aqu"
Confirmation Message	-	**Ok**	

Definition of Processes

Detachment must be added as an additional dynamic process:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Processes**	
Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Dynamic Process** **Ok**	
Edit Dynamic Process	A3.17	Name: **Detach_B** Description: **Detachment of B** Rate: **kdet*M_B^2** **Add**	dynamic process describing detachment
Edit Stoich. Coefficient	A3.18	Variable: **M_B** St. Coeff.: -1 **Ok**	sessile organisms B are eliminated with the specified rate
Edit Dynamic Process	A3.17	**Ok**	

Definition of Compartments

The new process must be activated:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **Reactor** **Edit**	
Edit Mixed React. Comp.	A3.29	**Processes**	
Select Active Processes	A3.24	Select **Detach_B** **Activate** **Ok**	activate the process Detach_B
Edit Compartments	A3.20	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation over 50 hours can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **0.5** Num. Steps: **100** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **ConcA** **Plot to Screen** Select **MassB** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **ConcA** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex3b.ps** **Ok**	

View Results	A5.2	Select **MassB** **Plot to File**	
File Save Dialog	-	File Name: **ex3b.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex3b.ps" can be submitted to a printer in a system dependent way. Fig. A6.3 shows the plot as it is printed by processing the file "ex3b.ps".

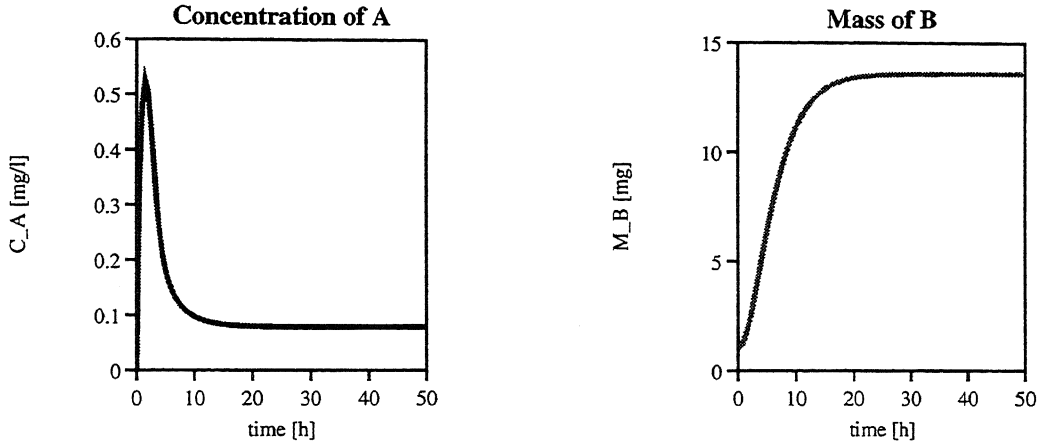


Fig. A6.7: Plot printed by processing the file "ex3b.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex3b.prn** **Ok**	write system definitions to "ex3b.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex3b.aqu". The system definitions have been written to the text file "ex3b.prn" which can be submitted to a printer in order to check the system definitions.

Solution to Part C:

Continue your work after you have performed part B or start the window interface version of AQUASIM and load the file "ex2b.aqu" saved in part B with the commands File → Open, File Name: exp2b.aqu, Ok, Ok.

Definition of Variables

State variables for the two species of A must be defined, time must be made available and the measured time series of property P must be entered in order to solve this part of the exercise:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **C_A1** Description: **Conc. of species 1** Unit: **mg/l** Type: **equilibrium** **Ok**	equilibrium state variable describing concentration of species 1
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **C_A2** Description: **Conc. of species 2** Unit: **mg/l** Type: **equilibrium** **Ok**	equilibrium state variable describing concentration of species 2
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **t** Description: **Time** Unit: **h** Reference to: **Time** **Ok**	time is required for the definition of input time series for property P
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Real List Variable** **Ok**	
Edit Real List Variable	A3.9	Name: **P** Description: **Property P** Unit: Argument: **t** Stand. Dev.: **global** Rel. St.Dev.: **0** Abs. St.Dev.: **1** Minimum: **0** Maximum: **10** Argument: **0** Value: **1.1** **Add** Argument: **10** Value: **1.3** **Add** Argument: **20** Value: **1.4** **Add** Argument: **50** Value: **1.5** **Add** Int. Method: **linear** **Ok**	definition of time series of property P (below the list box)
Menu Bar	A1.1/A2.1	**File → Save As**	
File Save Dialog	-	File Name: **ex3c.aqu** **Ok**	save definitions to "ex3c.aqu"
Confirmation Message	-	**Ok**	

Definition of Processes

The chemical equilibrium conditions for species 1 and 2 of substance A have to be specified as equilibrium processes and the growth process has to be corrected to depend on species 1:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Processes**	
Edit Processes	A3.14	Select **Gro_B** **Edit**	
Edit Dynamic Process	A3.17	Rate: **C_A1/(K_A+C_A1) *M_B** **Ok**	rate depends on C_A1 instead of C_A, but stoichiometric coefficient remains for C_A
Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Equilibrium Process** **Ok**	
Edit Equilibrium Process	A3.19	Name: **Mass_A1A2** Description: **Total mass as sum of species 1 and 2** Variable: **C_A1** Equation: 0= **C_A1+C_A2-C_A** **Ok**	
Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Equilibrium Process** **Ok**	
Edit Equilibrium Process	A3.19	Name: **Equi_A1A2** Description: **Equilibrium cond.** Variable: **C_A2** Equation: 0= **C_A1-2.3*P*C_A2** **Ok**	

Definition of Compartments

The new state variables and processes must be activated:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **Reactor** **Edit**	
Edit Mixed React. Comp.	A3.29	**Variables**	
Select Active Variables	A3.23	Select **C_A1** **Activate** Select **C_A2** **Activate** **Ok**	activate the state variables C_A1 and C_A2
Edit Mixed React. Comp.	A3.29	**Processes**	
Select Active Processes	A3.24	Select **Mass_A1A2** **Activate** Select **Equi_A1A2** **Activate** **Ok**	activate the processes Mass_A1A2 and Equi_A1A2
Edit Mixed React. Comp.	A3.29	**Ok**	

Definition of Plots

The plot definition specified in part A must be changed to include curved for the two species of substance A:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **ConcA** **Edit**	
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C_A1** Legend: **Species 1**	curve for concentration of

		Line Style: **dashed** Line Color: **blue** **Ok**	species 1 of A
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C_A2** Legend: **Species 2** Line Style: **dotted** Line Color: **blue** **Ok**	curve for concentration of species 2 of A
Edit Plot Definition	A5.3	**Ok**	
Menu Bar Confirmation Message	A1.1/A2.1 -	**File → Save** **Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation over 50 hours can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **0.5** Num. Steps: **100** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **ConcA** **Plot to Screen** Select **MassB** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **ConcA** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex3c.ps** **Ok**	
View Results	A5.2	Select **MassB** **Plot to File**	
File Save Dialog	-	File Name: **ex3c.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex3c.ps" can be submitted to a printer in a system dependent way. Fig. A6.3 shows the plot as it is printed by processing the file "ex3c.ps".

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	

File Save Dialog	-	File Name: **ex3c.prn** **Ok**	write system definitions to "ex3c.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex3c.aqu". The system definitions have been written to the text file "ex3c.prn" which can be submitted to a printer in order to check the system definitions.

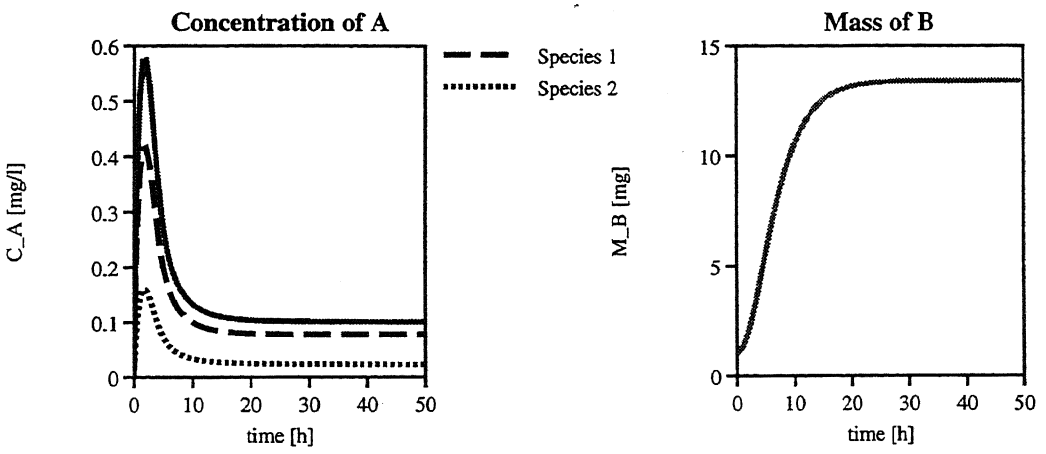


Fig. A6.8: Plot printed by processing the file "ex3c.ps" created in this example

A6.4 Exercise 4: Parameter Estimation

This exercise introduces parameter estimation and sensitivity analysis by evaluations of batch degradation experiments.

Part A: Use the measured time series of decreasing substrate concentration of a substance in a batch reactor with a volume of 10 liters to estimate the parameters maximum conversion rate and half-saturation concentration of a Monod-type degradation process. Start the simulations with the measured concentration at time zero. Use the following data

time [h]:	0	1.5	3	4.5	6	7.5	9	10.5	12	13.5	15
concentration [mg/l]:	10.5	8.3	7.4	6.2	4.9	3.2	2.4	1.3	0.7	0.1	0.1

and use a relative standard deviation of 3 % and an absolute standard deviation of 0.1 mg/l for the accuracy of the measurements. Start the estimation with an initial value of 2 mg/l for the half-saturation concentration (range from 0 to 10 mg/l) and an initial value of 2 mg/l/h for the maximum conversion rate (range from 0 to 10 mg/l/h).

Part B: Treat the initial concentration as an additional model parameter (initial value 10.5 mg/l, range from 0 to 20 mg/l) and redo the fit performed before.

Part C: Use the data from a second experiment with a different population of the same species to increase confidence of the estimated Monod coefficient. The data is given by

time [h]:	0	1	2	3	4	5	6	7	8	9	10
concentration [mg/l]:	0.95	0.83	0.59	0.39	0.25	0.17	0.13	0.08	0.06	0.02	0.02

with a relative standard deviation of 3 % and an absolute standard deviation of 0.02 mg/l.

Part D: Perform a sensitivity analysis of both experiments with respect to the model parameters.

Solution to Part A:

Start the window interface version of AQUASIM or select File → New to remove previously entered data from memory.

Definition of Variables

For the formulation of this model, a state variable for the concentration of the substance, a program variable for making time available, a real list variable for measured data and constant variables for the model parameters must be defined (Note that only constant variables can be estimated by the program. If a model parameter to be estimated is defined as a formula variable, it can be converted to a constant variable by selecting it in the list box of the dialog box shown in Fig. A3.3 and selecting "Edit Type"):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **C** Description: **Concentration**	dynamic volume state variable

		Unit: **mg/l** **Ok**	describing concentration
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **t** Description: **Time** Unit: **h** Reference to: **Time** **Ok**	this variable makes time available for the definition of time series
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Real List Variable** **Ok**	
Edit Real List Variable	A3.9	Name: **Cmeas** Unit: **mg/l** Argument: **t** Stand. Dev.: **global** Rel. St.Dev.: **0.03** Abs. St.Dev.: **0.1** Minimum: **0** Maximum: **20** Argument: **0** Value: **10.5** **Add** Argument: **1.5** Value: **8.3** **Add** Argument: **3** Value: **7.4** **Add** Argument: **4.5** Value: **6.2** **Add** Argument: **6** Value: **4.9** **Add** Argument: **7.5** Value: **3.2** **Add** Argument: **9** Value: **2.4** **Add** Argument: **10.5** Value: **1.3** **Add** Argument: **12** Value: **0.7** **Add** Argument: **13.5** Value: **0.1** **Add** Argument: **15** Value: **0.1** **Add** **Ok**	real list variable for measured concentrations (below the list box) instead of manually entering the data pairs, the data could be read from a file using the button Read
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Constant Variable** **Ok**	
Edit Constant Variable	A3.8	Name: **K** Description: **Half-saturation concentration** Unit: **mg/l** Value: **2** Stand. Dev.: **1** Minimum: **0** Maximum: **10** **Ok**	

Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Constant Variable** **Ok**	
Edit Constant Variable	A3.8	Name: **rmax** Description: **Maximum conversion rate** Unit: **mg/l/h** Value: **2** Stand. Dev.: **1** Minimum: **0** Maximum: **10** **Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
File Save Dialog	-	File Name: **ex4a.aqu** **Ok**	save definitions to "ex4a.aqu"
Confirmation Message	-	**Ok**	

Definition of Processes

Degradation of the substance is formulated as a dynamic process:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Processes**	
Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Dynamic Process** **Ok**	
Edit Dynamic Process	A3.17	Name: **Degr** Description: **Degradation** Rate: **rmax*C/(K+C)** **Add**	dynamic process with Monod rate law
Edit Stoich. Coefficient	A3.18	Variable: **C** St. Coeff.: **-1** **Ok**	A is degraded with the specified rate
Edit Dynamic Process	A3.17	**Ok**	

Definition of Compartments

A single mixed reactor compartment representing the batch reactor is required:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	**New**	
Select Compartment Type	A3.21	**Mixed Reactor Comparment** **Ok**	
Edit Mixed React. Comp.	A3.29	Name: **Reactor** **Variables**	
Select Active Variables	A3.23	Select **C** **Activate** **Ok**	activate the state variable C
Edit Mixed React. Comp.	A3.29	**Processes**	
Select Active Processes	A3.24	Select **Degr** **Activate** **Ok**	activate the process Degr
Edit Mixed React. Comp.	A3.29	**Init. Cond.**	
Edit Initial Conditions	A3.25	**Add**	
Edit Initial Condition	A3.26	Variable: **C** Init. Cond.: **Cmeas** **Ok**	initial concentration as measured
Edit Initial Conditions	A3.25	**Ok**	
Edit Mixed React. Comp.	A3.29	Volume: **10** **Ok**	

Definition of Plots

Graphical presentation of results requires a plot definition. Once a plot definition has been specified, it can be used for several calculations with different parameters or model structures. It is advantageous to specify the plot definitions before performing the calculation:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	**New**	
Edit Plot Definition	A5.3	Name: **Conc** Title: **Concentration** Abscissa: **Time** Absc. Label: **time [h]** Ord. Label: **C [mg/l]** **Add**	
Edit Curve Definition	A5.4	Variable: **Cmeas** Line active: **uncheck** Marker act.: **check** Marker Size: **4** **Ok**	markers for measurements
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C** **Ok**	curve for concentration
Edit Plot Definition	A5.3	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Parameter Estimation and Viewing Results

Now the parameter estimation can be defined and performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Parameter Estimation**	
Parameter Estimation	A4.8	Select **K** **Activate** Select **rmax** **Activate** **New**	activate model parameters to be estimated
Edit Calculation	A4.9	Name: **Fit1** **Add**	
Edit Fit Target	A4.10	Data: **Cmeas** Variable: **C** **Ok**	specify data series and variable to be compared with data
Edit Calculation	A4.9	**Ok**	
Parameter Estimation	A4.8	Select **Fit1** **Activate** **Start**	
File Save Dialog	-	File Name: **ex4a.fit** **Ok**	
Results of Par. Estim.	A4.12	**Ok**	

Now the parameters are changed to their optimal values and the calculated states are available for being plotted or listed. Note that the resolution of the plots corresponds to that of measured data; in order to obtain plots with a better resolution, a simulation with smaller time steps must be executed before plotting. The detailed results of the parameter estimation can be found in the file "ex4a.fit".

Now the plot should be visible on the screen and the PostScript file "ex4a.ps" can be submitted to a printer in a system dependent way. Fig. A6.9 shows the plot as it is printed by processing the file "ex4a.ps".

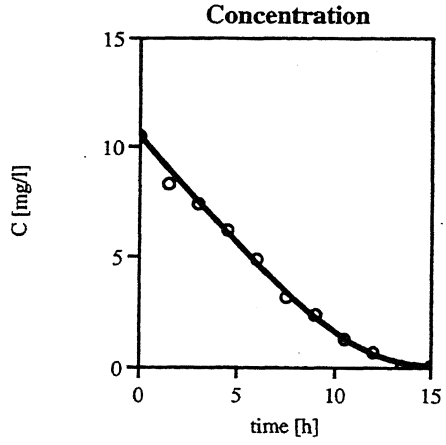


Fig. A6.9: Plot printed by processing the file "ex4a.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex4a.prn** **Ok**	write system definitions to "ex4a.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex4a.aqu". The system definitions have been written to the text file "ex4a.prn" which can be submitted to a printer in order to check the system definitions.

Solution to Part B:

Continue your work after you have performed part A or start the window interface version of AQUASIM and load the file "ex4a.aqu" saved in part A with the commands File → Open, File Name: exp4a.aqu, Ok, Ok.

Definition of Variables

An additional constant variable is necessary for the initial condition of concentration:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Constant Variable** **Ok**	
Edit Constant Variable	A3.8	Name: **Cini** Description: **Initial conc.**	

		Unit: **mg/l** Value: **10.5** Stand. Dev.: **1** Minimum: **0** Maximum: **20** **Ok**	
Menu Bar	A1.1/A2.1	**File → Save As**	
File Save Dialog	-	File Name: **ex4b.aqu** **Ok**	save definitions to "ex4b.aqu"
Confirmation Message	-	**Ok**	

Definition of Compartments

The new variable must be made to the initial concentration of the modelled substance:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **Reactor** **Edit**	
Edit Mixed React. Comp.	A3.29	**Init. Cond.**	
Edit Initial Conditions	A3.25	Select **C(Bulk Volume) : Cmeas** **Edit**	
Edit Initial Condition	A3.26	Init. Cond.: **Cini** **Ok**	
Edit Initial Conditions	A3.25	**Ok**	
Edit Mixed React. Comp.	A3.29	**Ok**	

Performing the Parameter Estimation and Viewing Results

Now the parameter estimation can be modified and redone:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Parameter Estimation**	
Parameter Estimation	A4.8	Select **Cini** **Activate** **Start**	
File Save Dialog	-	File Name: **ex4b.fit** **Ok**	
Results of Par. Estim.	A4.12	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **Conc** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex4b.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex4b.ps" can be submitted to a printer in a system dependent way. Fig. A6.10 shows the plot as it is printed by processing the file "ex4b.ps". Note that the calculated initial concentration is now slightly different from the measured value.

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states for calculation number zero

Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex4b.prn** **Ok**	write system definitions to "ex4b.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex4b.aqu". The system definitions have been written to the text file "ex4b.prn" which can be submitted to a printer in order to check the system definitions.

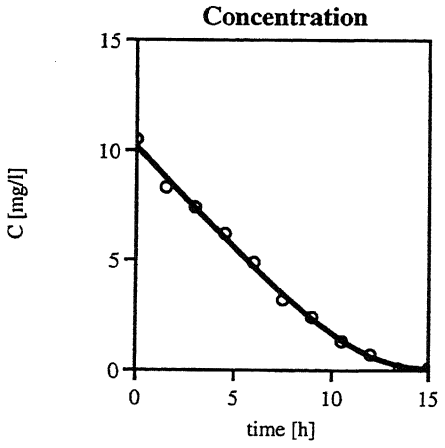


Fig. A6.10: Plot printed by processing the file "ex4b.ps" created in this example

Solution to Part C:

Continue your work after you have performed part B or start the window interface version of AQUASIM and load the file "ex4b.aqu" saved in part B with the commands File → Open, File Name: exp4b.aqu, Ok, Ok.

Definition of Variables

Additional variables are necessary for the consideration of the additional experiment. The two experiments are distinguished by different values of the calculation number defined at calculation time. Experiment-specific parameters must be made different for different calculation numbers (Cini and rmax; K is assumed to be universal):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	Select **Cmeas** **Edit**	
Edit Real List Variable	A3.9	Name: **Cmeas1** **Ok**	change of name
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Real List Variable** **Ok**	
Edit Real List Variable	A3.9	Name: **Cmeas2** Unit: **mg/l** Argument: **t** Stand. Dev.: **global** Rel. St.Dev.: **0.03** Abs. St.Dev.: **0.02** Minimum: **0** Maximum: **20**	real list variable for the second data series

		Argument: 0 Value: **0.95** **Add** Argument: 1 Value: **0.83** **Add** Argument: 2 Value: **0.59** **Add** Argument: 3 Value: **0.39** **Add** Argument: 4 Value: **0.25** **Add** Argument: 5 Value: **0.17** **Add** Argument: 6 Value: **0.13** **Add** Argument: 7 Value: **0.08** **Add** Argument: 8 Value: **0.06** **Add** Argument: 9 Value: **0.02** **Add** Argument: 10 Value: **0.02** **Add** **Ok**	(below the list box) instead of manually entering the data pairs, the data could be read from a file using the button Read
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **calcnum** Description: **Calculation number** Unit: Reference to: **Calculation Number** **Ok**	this variable makes the current value of the calculation number available
Edit Variables	A3.3	Select **Cini** **Duplicate**	
Edit Constant Variable	A3.8	Name: **Cini1** **Ok**	
Edit Variables	A3.3	Select **Cini** **Duplicate**	
Edit Constant Variable	A3.8	Name: **Cini2** Value: **1** Stand. Dev.: **1** Minimum: **0** Maximum: **2** **Ok**	
Edit Variables	A3.3	Select **Cini** **Edit Type**	
Select Variable Type	A3.4	**Variable List Variable** **Ok**	
Edit Variable List Var.	A3.12	Argument: **calcnum** Argument: **0** Variable: **Cini1** **Add** Argument: **1** Variable: **Cini2** **Add** **Ok**	Cini is equal to Cini1 for calculation number 0; equal to Cini2 for calculation number 1

Edit Variables	A3.3	Select **rmax** **Duplicate**	
Edit Constant Variable	A3.8	Name: **rmax1** **Ok**	
Edit Variables	A3.3	Select **rmax** **Duplicate**	
Edit Constant Variable	A3.8	Name: **rmax2** **Ok**	
Edit Variables	A3.3	Select **rmax** **Edit Type**	
Select Variable Type	A3.4	**Variable List Variable** **Ok**	
Edit Variable List Var.	A3.12	Argument: **calcnum** Argument: **0** Variable: **rmax1** **Add** Argument: **1** Variable: **rmax2** **Add** **Ok**	rmax is equal to rmax1 for calculation number 0; equal to rmax2 for calculation number 1
Menu Bar	A1.1/A2.1	**File → Save As**	
File Save Dialog	-	File Name: **ex4c.aqu** **Ok**	save definitions to "ex4c.aqu"
Confirmation Message	-	**Ok**	

Definition of Plots

An additional plot definition is required in order to plot calculation and data of the second data set:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc** **Edit**	
Edit Plot Definition	A5.3	Name: **Conc1** Title: **Conc. Exp. 1** **Ok**	change name
View Results	A5.2	Select **Conc1** **Duplicate**	
Edit Plot Definition	A5.3	Name: **Conc2** Title: **Conc. Exp. 2** Select **Value : Cmeas1 ...** **Edit**	
Edit Curve Definition	A5.4	Variable: **Cmeas2** Calc. Num.: **1** **Ok**	markers for measurements
Edit Plot Definition	A5.3	Select **Value : C ...** **Edit**	
Edit Curve Definition	A5.4	Calc. Num.: **1** **Ok**	curve for concentration
Edit Plot Definition	A5.3	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Parameter Estimation and Viewing Results

Now the parameter estimation can be defined and performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Parameter Estimation**	
Parameter Estimation	A4.8	**New**	
Edit Calculation	A4.9	Name: **Fit2** Calc. Num.: **1** Initial Time: **0** **Add**	

Edit Fit Target	A4.10	Data: **Cmeas2** Variable: **C** **Ok**	
Edit Calculation	A4.9	**Ok**	
Parameter Estimation	A4.8	Select **Fit2** **Activate** **Start**	
File Save Dialog	-	File Name: **ex4c.fit** **Ok**	
Results of Par. Estim.	A4.12	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc1** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **Conc2** **Plot to Screen**	
View Results	A5.2	Select **Conc1** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex4c.ps** **Ok**	
Confirmation Message	-	**Ok**	
View Results	A5.2	Select **Conc2** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex4c.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex4c.ps" can be submitted to a printer in a system dependent way. Fig. A6.11 shows the plot as it is printed by processing the file "ex4c.ps".

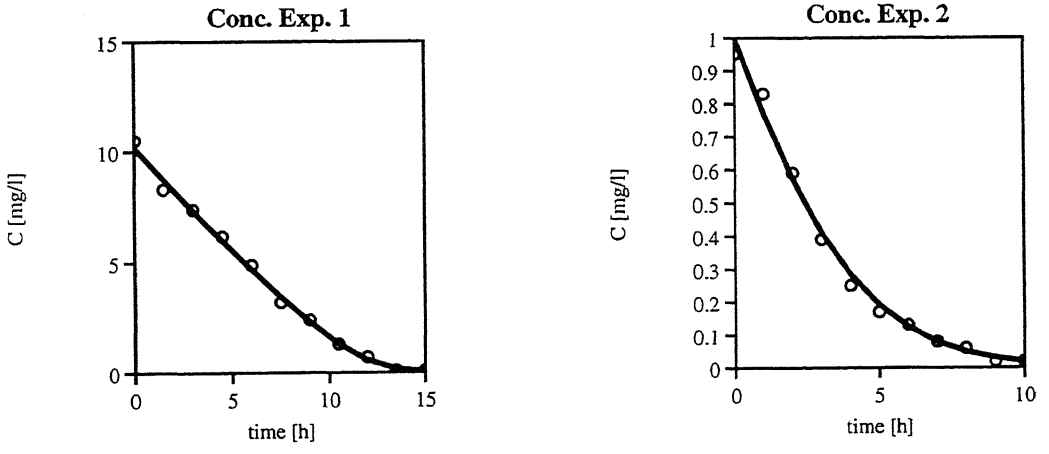


Fig. A6.11: Plot printed by processing the file "ex4c.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete** Select CalcNum **1** **Delete**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	

File Save Dialog	-	File Name: **ex4c.prn** **Ok**	write system definitions to "ex4c.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex4c.aqu". The system definitions have been written to the text file "ex4c.prn" which can be submitted to a printer in order to check the system definitions.

Solution to Part D:

Continue your work after you have performed part C or start the window interface version of AQUASIM and load the file "ex4c.aqu" saved in part C with the commands File → Open, File Name: exp4c.aqu, Ok, Ok.

Definition of Plots

Additional plots are required in order to plot the sensitivity functions:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	**New**	
Edit Plot Definition	A5.3	Name: **Sens1** Title: **Sens. Functions for Exp. 1** Abscissa: **Time** Absc. Label: **time [h]** Ord. Label: **delta_ar [mg/l]** **Add**	general plot definitions
Edit Curve Definition	A5.4	Type: **Sens. Fun., AbsRel** Variable: **C** Parameter: **Cini1** Calc. Num.: **0** Legend: **Cini1** Line Style: **solid** Line Color: **red** **Ok**	sensitivity function of C with respect to Cini1 (1st exp.)
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Type: **Sens. Fun., AbsRel** Variable: **C** Parameter: **rmax1** Calc. Num.: **0** Legend: **rmax1** Line Style: **dashed** Line Color: **green** **Ok**	sensitivity function of C with respect to rmax1 (1st exp.)
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Type: **Sens. Fun., AbsRel** Variable: **C** Parameter: **K** Calc. Num.: **0** Legend: **K** Line Style: **dotted** Line Color: **blue** **Ok**	sensitivity function of C with respect to K (1st exp.)
Edit Plot Definition	A5.3	**Ok**	
View Results	A5.2	Select **Sens1** **Duplicate**	
Edit Plot Definition	A5.3	Name: **Sens2** Title: **Sens. Functions for Exp. 2** Select **SensAR C(Cini1) ...** **Edit**	general plot definitions

Edit Curve Definition	A5.4	Parameter: **Cini2** Calc. Num.: **1** Legend: **Cini2** **Ok**	sensitivity function of C with respect to Cini2 (2nd exp.)
Edit Plot Definition	A5.3	Select **SensAR C(rmax1) ...** **Edit**	
Edit Curve Definition	A5.4	Parameter: **rmax2** Calc. Num.: **1** Legend: **rmax2** **Ok**	sensitivity function of C with respect to rmax2 (2nd exp.)
Edit Plot Definition	A5.3	Select **SensAR C(K) ...** **Edit**	
Edit Curve Definition	A5.4	Calc. Num.: **1** **Ok**	sensitivity function of C with respect to K (2nd exp.)
Edit Plot Definition	A5.3	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save As**	
File Save Dialog	-	File Name: **ex4d.aqu** **Ok**	save definitions to "ex3c.aqu"
Confirmation Message	-	**Ok**	

Performing the Sensitivity Analysis and Viewing Results

Now, the sensitivity analysis can be defined and performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Sensitivity Analysis**	
Sensitivity Analysis	A4.5	Select **K** **Activate** Select **rmax1** **Activate** Select **rmax2** **Activate** Select **Cini1** **Activate** Select **Cini2** **Activate** **New**	activate model parameters
Edit Calculation	A4.6	Name: **Sens1** Calc. Num.: **0** Initial Time: **0** Step Size: **0.1** Num. Steps: **150** **Ok**	specify calculation number and temporal resolution
Sensitivity Analysis	A4.5	**New**	
Edit Calculation	A4.6	Name: **Sens2** Calc. Num.: **1** Initial Time: **0** Step Size: **0.1** Num. Steps: **150** **Ok**	specify calculation number and temporal resolution
Sensitivity Analysis	A4.5	Select **Sens1** **Activate** Select **Sens2** **Activate** **Start**	
Confirmation Box	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Sens1** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **Sens2** **Plot to Screen**	
View Results	A5.2	Select **Sens1** **Plot to File**	plot results to a PostScript file

File Save Dialog	-	File Name: **ex4d.ps** **Ok**	
View Results	A5.2	Select **Sens2** **Plot to File**	
File Save Dialog	-	File Name: **ex4d.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex4d.ps" can be submitted to a printer in a system dependent way. Fig. A6.12 shows the plot as it is printed by processing the file "ex4d.ps".

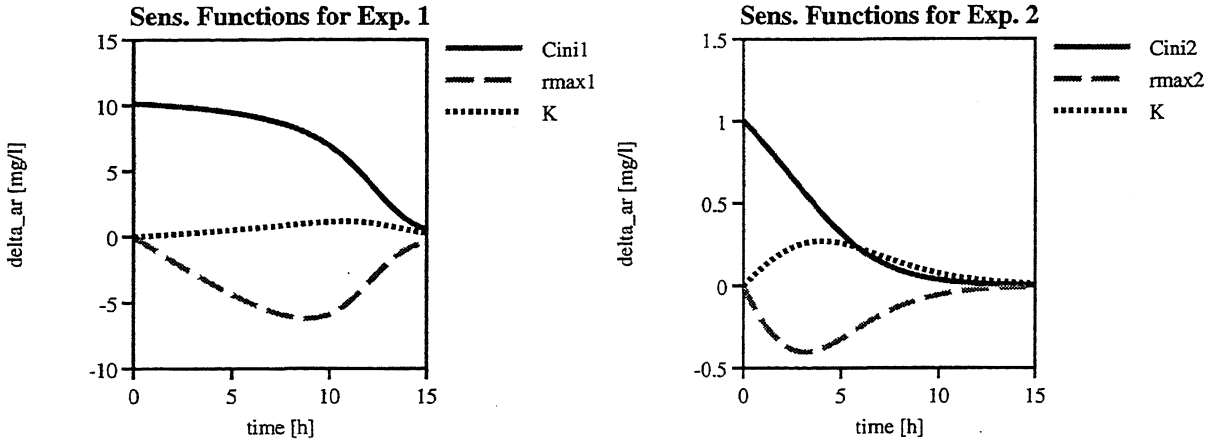


Fig. A6.12: Plot printed by processing the file "exp4d.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete** Select CalcNum **1** **Delete**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex4d.prn** **Ok**	write system definitions to "ex4d.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex4d.aqu". The system definitions have been written to the text file "ex4d.prn" which can be submitted to a printer in order to check the system definitions.

A6.5 Exercise 5: Biofilm Growth

This exercise introduces the biofilm reactor compartment.

Part A: Growth of a biofilm consisting of two microbial species is investigated. The volume of the water flowing over the film surface of 1 m^2 is assumed to be constant (10^{-4} m^3). This water, provided at a rate of 1 m^3/d contains two substrates at the concentrations of 10 $gCOD/m^3$ each. The density of both microorganism species is 25000 $gCOD/m^3$. It is assumed that each species grows on one substrate only. Growth occurs with Monod-type rate laws with maximum growth rates of 0.4 d^{-1} and 0.1 d^{-1}, half-saturation concentrations of 5 $gCOD/m^3$ and yield coefficients of 0.01 and 0.5, respectively. Respiration occurs with specific rates of 0.05 d^{-1} and 0.002 d^{-1}. Initial film depth is 10^{-4} m, initial volume fractions of both species is 10 % (80 % of the film consists of water). The diffusion coefficients of both substrates are assumed to be $2 \cdot 10^{-5}$ m^2/d. Perform a simulation for 50 days and plot the concentration profiles of the dissolved substances and the volume fraction profiles of the particulate substances at the days 10, 25 and 50 and the film thickness as a function of time.

Part B: Assume at the biofilm surface a mass transfer resistance corresponding to a molecular boundary layer of 10^{-4} m (diffusivity of particles equal to $5 \cdot 10^{-6}$ m^2/d) and a detachment velocity equal to half of the growth velocity of the film and redo the calculation.

Solution to Part A:

Start the window interface version of AQUASIM or select File → New to remove previously entered data from memory.

Definition of Variables

For the formulation of this model, state variables for the concentrations of substrates and microorganisms must be provided and the program variable "Biofilm Thickness" must be made available for the specification of the initial condition in the next step. Furthermore, it is recommendable to introduce various model parameters as variables.

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **S1** Description: **Conc. of substr. 1** Unit: **gCOD/m^3** **Ok**	dynamic volume state variable describing substrate 1
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **S2** Description: **Conc. of substr. 2** Unit: **gCOD/m^3** **Ok**	dynamic volume state variable describing substrate 2
Edit Variables	A3.3	**New**	

Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **X1** Description: **Conc. of organ. 1** Unit: **gCOD/m^3** **Ok**	dynamic volume state variable describing microorganisms 1
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **X2** Description: **Conc. of organ. 2** Unit: **gCOD/m^3** **Ok**	dynamic volume state variable describing microorganisms 2
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **LF** Description: **Biofilm thickness** Unit: **m** Reference to: **Biofilm Thickness** **Ok**	this variable makes biofilm thickness available for the specification of the initial condition
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **Qin** Description: **Water inflow** Unit: **m^3/d** Expression: **1** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **S1in** Description: **Inflow conc. of S1** Unit: **gCOD/m^3** Expression: **10** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **S2in** Description: **Inflow conc. of S2** Unit: **gCOD/m^3** Expression: **10** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **rho** Description: **Bacterial density** Unit: **gCOD/m^3** Expression: **25000** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **mue_X1** Description: **Maximum specific growth rate of X1** Unit: **1/d** Expression: **0.4** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	

Edit Formula Variable	A3.13	Name: **mue_X2** Description: **Maximum specific growth rate of X2** Unit: **1/d** Expression: **0.1** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **K_S1** Description: **Half-saturation concentration of S1** Unit: **gCOD/m^3** Expression: **5** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **K_S2** Description: **Half-saturation concentration of S2** Unit: **gCOD/m^3** Expression: **5** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **Y1** Description: **Yield of X1 on S1** Expression: **0.01** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **Y2** Description: **Yield of X2 on S2** Expression: **0.5** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **b_X1** Description: **Resp. rate of X1** Unit: **1/d** Expression: **0.05** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **b_X2** Description: **Resp. rate of X2** Unit: **1/d** Expression: **0.002** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **D_S1** Description: **Diffusivity of S1** Unit: **m^2/d** Expression: **2e-5** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	

Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **D_S2** Description: **Diffusivity of S2** Unit: **m^2/d** Expression: **2e-5** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **eps_X1** Description: **Volume fract. of X1** Expression: **X1/rho** **Ok**	volume fraction defined as variable for plot
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **eps_X1** Description: **Volume fract. of X1** Expression: **X1/rho** **Ok**	volume fraction defined as variable for plot
Menu Bar	A1.1/A2.1	**File → Save**	
File Save Dialog	-	File Name: **ex5a.aqu** **Ok**	save definitions to "ex5a.aqu"
Confirmation Message	-	**Ok**	

Definition of Processes

Growth and respiration of both microorgansim species must be formulated as dynamic processes:

Window	Figure	Action	Comment
Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Dynamic Process** **Ok**	
Edit Dynamic Process	A3.17	Name: **Gro_X1** Description: **Growth of X1** Rate: **mue_X1 *S1/(K_S1+S1)*X1** **Add**	dynamic process of first order in X1 and of Monod-type in S1
Edit Stoich. Coefficient	A3.18	Variable: **X1** St. Coeff.: **1** **Ok**	X1 grows with the specified rate
Edit Dynamic Process	A3.17	**Add**	
Edit Stoich. Coefficient	A3.18	Variable: **S1** St. Coeff.: **-1//Y1** **Ok**	S1 is consumed with 1/Y1 times the specified rate
Edit Dynamic Process	A3.17	**Ok**	
Menu Bar	A1.1/A3.2	**Edit → Processes**	
Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Dynamic Process** **Ok**	
Edit Dynamic Process	A3.17	Name: **Gro_X2** Description: **Growth of X2** Rate: **mue_X2 *S2/(K_S2+S2)*X2** **Add**	dynamic process of first order in X2 and of Monod-type in S2
Edit Stoich. Coefficient	A3.18	Variable: **X2** St. Coeff.: **1** **Ok**	X2 grows with the specified rate
Edit Dynamic Process	A3.17	**Add**	
Edit Stoich. Coefficient	A3.18	Variable: **S2** St. Coeff.: **-1//Y2** **Ok**	S2 is consumed with 1/Y2 times the specified rate
Edit Dynamic Process	A3.17	**Ok**	

Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Dynamic Process** **Ok**	
Edit Dynamic Process	A3.17	Name: **Resp_X1** Description: **Respiration of X1** Rate: **b_X1*X1** **Add**	dynamic process with first order rate law
Edit Stoich. Coefficient	A3.18	Variable: **X1** St. Coeff.: **-1** **Ok**	X1 decays with the specified rate
Edit Dynamic Process	A3.17	**Ok**	
Edit Processes	A3.14	**New**	
Select Process Type	A3.15	**Dynamic Process** **Ok**	
Edit Dynamic Process	A3.17	Name: **Resp_X2** Description: **Respiration of X2** Rate: **b_X2*X2** **Add**	dynamic process with first order rate law
Edit Stoich. Coefficient	A3.18	Variable: **X2** St. Coeff.: **-1** **Ok**	X2 decays with the specified rate
Edit Dynamic Process	A3.17	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Definition of Compartments

A single mixed reactor compartment is required for this exercise:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	**New**	
Select Compartment Type	A3.21	**Biofilm Reactor Comparment** **Ok**	
Edit Biofilm React. Com.	A3.31	Name: **Reactor** Description: **Biofilm reactor** **Variables**	
Select Active Variables	A3.23	Select **S1** **Activate** Select **S2** **Activate** Select **X1** **Activate** Select **X2** **Activate** **Ok**	activate the state variables S1, S2, X1 and X2
Edit Biofilm React. Com.	A3.31	**Processes**	
Select Active Processes	A3.24	Select **Gro_X1** **Activate** Select **Gro_X2** **Activate** Select **Resp_X1** **Activate** Select **Resp_X2** **Activate** **Ok**	activate the processes Gro_X1, Gro_X2, Resp_X1 and Resp_X2
Edit Biofilm React. Com.	A3.31	**Init. Cond.**	
Edit Initial Conditions	A3.25	**Add**	
Edit Initial Condition	A3.26	Variable: **LF** Zone: **Biofilm** Init. Cond.: **0.0001** **Ok**	initial biofilm thickness
Edit Initial Conditions	A3.25	**Add**	
Edit Initial Condition	A3.26	Variable: **X1** Zone: **Biofilm**	initial conc. of X1 correspon-

		Init. Cond.: **0.1*rho** **Ok**	ding to 10 % of volume
Edit Initial Conditions	A3.25	**Add**	
Edit Initial Condition	A3.26	Variable: **X2** Zone: **Biofilm** Init. Cond.: **0.1*rho** **Ok**	initial conc. of X2 correspon-ding to 10 % of volume
Edit Initial Conditions	A3.25	**Ok**	
Edit Biofilm React. Com.	A3.31	**Input**	
Edit Inputs	A3.27	Water Inflow: **Qin** **Add**	water inflow as specified in Qin
Edit Input Flux	A3.28	Variable: **S1** Input Flux: **Qin*S1in** **Ok**	input flux equal discharge times concentration
Edit Inputs	A3.27	**Add**	
Edit Input Flux	A3.28	Variable: **S2** Input Flux: **Qin*S2in** **Ok**	input flux equal discharge times concentration
Edit Inputs	A3.27	**Ok**	
Edit Biofilm React. Com.	A3.31	**Particulate Variables**	
Edit Particulate Vars.	A3.32	**Add**	
Edit Part. Properties	A3.33	Variable: **X1** Density: **rho** **Ok**	
Edit Particulate Vars.	A3.32	**Add**	
Edit Part. Properties	A3.33	Variable: **X2** Density: **rho** **Ok**	
Edit Particulate Vars.	A3.32	**Ok**	
Edit Biofilm React. Com.	A3.31	**Dissolved Variables**	
Edit Dissolved Vars.	A3.34	**Add**	
Edit Diss. Properties	A3.35	Variable: **S1** Diffusivity: **D_S1** **Ok**	
Edit Dissolved Vars.	A3.34	**Add**	
Edit Diss. Properties	A3.35	Variable: **S2** Density: **D_S2** **Ok**	
Edit Dissolved Vars.	A3.34	**Ok**	
Edit Biofilm React. Com.	A3.31	React. Type: **unconfined** Bulk Volume: **0.001** Biof. Matrix: **rigid** Det. Veloc.: **0** Biofilm Area: **0.1** Rate epsFl: **0** **Ok**	unconfined re-actor with given bulk volume and film surface area
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Definition of Plots

Graphical presentation of results requires a plot definition. Once a plot definition has been specified, it can be used for several calculations with different parameters or model structures. It is advantageous to specify the plot definitions before performing the calculation:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	**New**	
Edit Plot Definition	A5.3	Name: **LF_t** Title: **Biofilm Thickness** Absc. Label: **time [h]** Ord. Label: **LF [m]** **Add**	define general plot attributes

Edit Curve Definition	A5.4	Variable: **LF** Zone: **Biofilm** **Ok**	curve for biofilm thickness
Edit Plot Definition	A5.3	**Ok**	
View Results	A5.2	**New**	
Edit Plot Definition	A5.3	Name: **S_z** Title: **Substrate Profiles** Abscissa: **Space** Absc. Label: **z [m]** Ord. Label: **S [gCOD/m^3]** **Add**	define general plot attributes
Edit Curve Definition	A5.4	Variable: **S1** Zone: **Biofilm** Time/Space: **10** Legend: **S1, 10 d** Line Style: **solid** Line Color: **black** **Ok**	curve for profile of S1 within the biofilm after 10 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S1** Zone: **Bulk Volume** Time/Space: **10** Line active: **uncheck** Marker act.: **check** Mark. Color: **black** **Ok**	marker for value of S1 in the bulk volume after 10 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S1** Zone: **Biofilm** Time/Space: **25** Legend: **S1, 25 d** Line Style: **long dashed** Line Color: **red** **Ok**	curve for profile of S1 within the biofilm after 25 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S1** Zone: **Bulk Volume** Time/Space: **25** Line active: **uncheck** Marker act.: **check** Mark. Color: **red** **Ok**	marker for value of S1 in the bulk volume after 25 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S1** Zone: **Biofilm** Time/Space: **50** Legend: **S1, 50 d** Line Style: **dashed** Line Color: **green** **Ok**	curve for profile of S1 within the biofilm after 50 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S1** Zone: **Bulk Volume** Time/Space: **50** Line active: **uncheck** Marker act.: **check** Mark. Color: **green** **Ok**	marker for value of S1 in the bulk volume after 50 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S2** Zone: **Biofilm** Time/Space: **10** Legend: **S2, 10 d** Line Style: **short dashed** Line Color: **blue** **Ok**	curve for profile of S2 within the biofilm after 10 days of simulation

Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S2** Zone: **Bulk Volume** Time/Space: **10** Line active: **uncheck** Marker act.: **check** Mark. Color: **blue** **Ok**	marker for value of S2 in the bulk volume after 10 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S2** Zone: **Biofilm** Time/Space: **25** Legend: **S2, 25 d** Line Style: **dotted** Line Color: **cyan** **Ok**	curve for profile of S2 within the biofilm after 25 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S2** Zone: **Bulk Volume** Time/Space: **25** Line active: **uncheck** Marker act.: **check** Mark. Color: **cyan** **Ok**	marker for value of S2 in the bulk volume after 25 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S2** Zone: **Biofilm** Time/Space: **50** Legend: **S2, 50 d** Line Style: **long dashdot** Line Color: **magenta** **Ok**	curve for profile of S2 within the biofilm after 50 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **S2** Zone: **Bulk Volume** Time/Space: **50** Line active: **uncheck** Marker act.: **check** Mark. Color: **magenta** **Ok**	marker for value of S2 in the bulk volume after 50 days of simulation
Edit Plot Definition	A5.3	**Ok**	
View Results	A5.2	**New**	
Edit Plot Definition	A5.3	Name: **X1_z** Title: **Bacteria Profiles X1** Abscissa: **Space** Absc. Label: **z [m]** Ord. Label: **eps_X1 [-]** **Add**	define general plot attributes
Edit Curve Definition	A5.4	Variable: **eps_X1** Zone: **Biofilm** Time/Space: **10** Legend: **10 d** Line Style: **solid** Line Color: **red** **Ok**	curve for profile of eps_X1 within the biofilm after 10 days of simulation
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **eps_X1** Zone: **Biofilm** Time/Space: **25** Legend: **25 d** Line Style: **dashed** Line Color: **green** **Ok**	curve for profile of eps_X1 within the biofilm after 25 days of simulation
Edit Plot Definition	A5.3	**Add**	

Edit Curve Definition	A5.4	Variable: **eps_X1** Zone: **Biofilm** Time/Space: **50** Legend: **50 d** Line Style: **dotted** Line Color: **blue** **Ok**	curve for profile of eps_X1 within the biofilm after 50 days of simulation
Edit Plot Definition	A5.3	**Ok**	
View Results	A5.2	Select **X1_z** **Duplicate**	
Edit Plot Definition	A5.3	Name: **X2_z** Title: **Bacteria Profiles X2** Ord. Label: **eps_X1 [-]** Select **Value : X1 [. . . ,10]** **Edit**	define general plot attributes
Edit Curve Definition	A5.4	Variable: **eps_X2** **Ok**	curve for profile of eps_X2
Edit Plot Definition	A5.3	Select **Value : X1 [. . . ,25]** **Edit**	
Edit Curve Definition	A5.4	Variable: **eps_X2** **Ok**	curve for profile of eps_X2
Edit Plot Definition	A5.3	Select **Value : X1 [. . . ,50]** **Edit**	
Edit Curve Definition	A5.4	Variable: **eps_X2** **Ok**	curve for profile of eps_X2
Edit Plot Definition	A5.3	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation over 50 days can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **1** Num. Steps: **50** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **LF_t** **Plot to Screen** Select **S_z** **Plot to Screen** Select **X1_z** **Plot to Screen** Select **X2_z** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **LF_t** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex5a.ps** **Ok**	
Confirmation Message	-	**Ok**	
View Results	A5.2	Select **S_z** **Plot to File**	
File Save Dialog	-	File Name: **ex5a.ps** **Ok**	
Confirmation Message	-	**Ok**	
View Results	A5.2	Select **X1_z** **Plot to File**	

File Save Dialog	-	File Name: **ex5a.ps** **Ok**	
Confirmation Message	-	**Ok**	
View Results	A5.2	Select **X2_z** **Plot to File**	
File Save Dialog	-	File Name: **ex5a.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex5a.ps" can be submitted to a printer in a system dependent way. Fig. A6.13 shows the plot as it is printed by processing the file "ex5a.ps". Substrate S1, which is consumed at a much higher rate decays very fast from the bulk value at the surface of the biofilm to zero in the depth of the film. For this reason the depth of the film is dominated by species X2, whereas due to its larger growth rate, near the surface species X1 dominates.

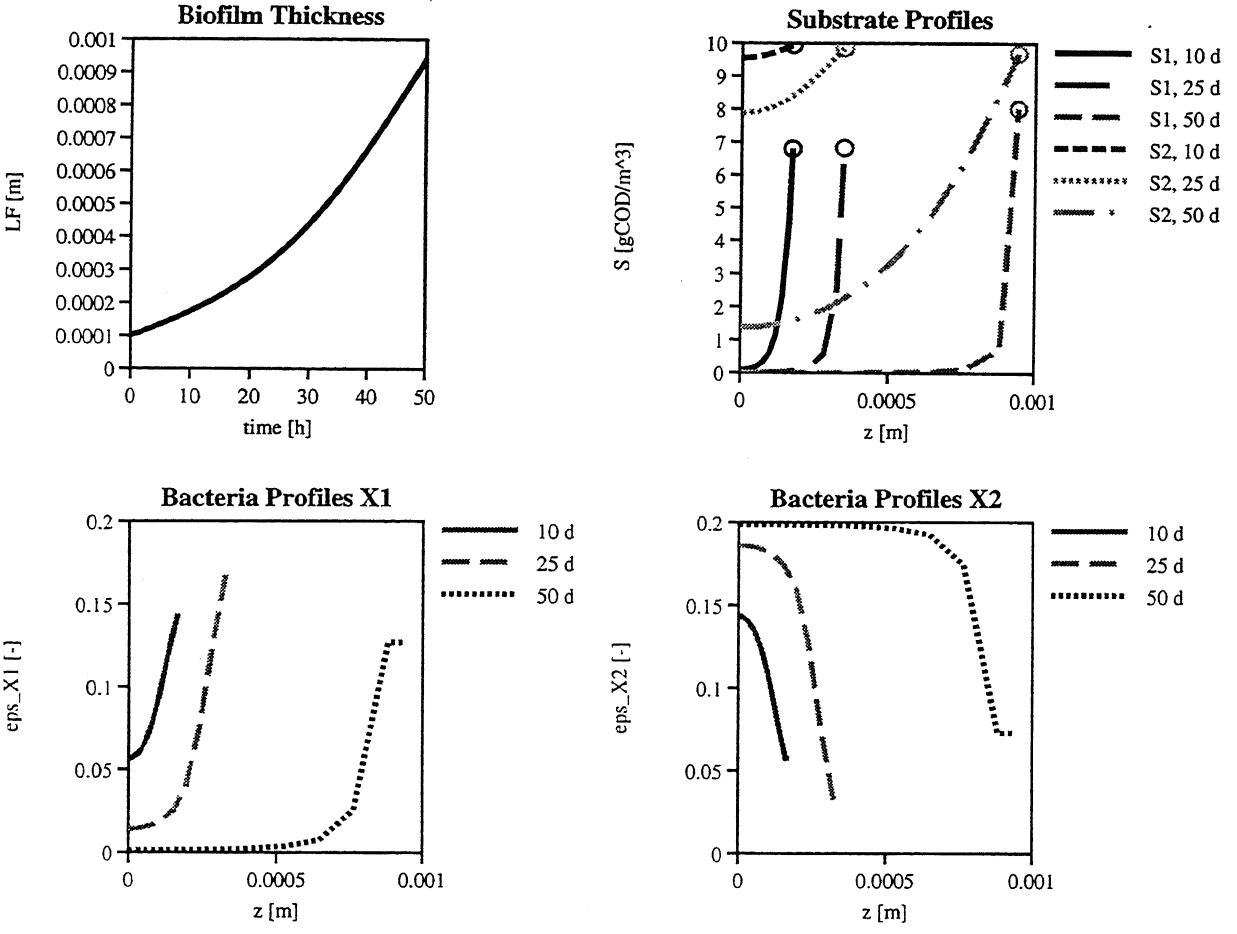


Fig. A6.13: Plot printed by processing the file "ex5a.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states for calculation number zero

Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex5a.prn** **Ok**	write system definitions to "ex5a.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex5a.aqu". The system definitions have been written to the text file "ex5a.prn" which can be submitted to a printer in order to check the system definitions.

Solution to Part B:

Continue your work after you have performed part A or start the window interface version of AQUASIM and load the file "ex5a.aqu" saved in part A with the commands File → Open, File Name: exp5a.aqu, Ok, Ok.

Definition of Variables

For the formulation of this model, biofilm growth velocity must be made available and it is recommendable to specify the diffusivity of particulate variables and the thickness of the boundary layer as variables.

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **uF** Description: **Growth velocity of biofilm** Unit: **m/d** Reference to: **Growth Velocity of Biofilm** **Ok**	this variable makes biofilm growth velocity available for the specification of the detachment rate
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **LL** Description: **Thickness of boundary layer** Unit: **m** Expression: **5e-4** **Ok**	volume fraction defined as variable for plot
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **D_X** Description: **Diffusivity of X1/2** Unit: **m^2/d** Expression: **5e-6** **Ok**	diffusivity of particulate variables in boundary layer
Menu Bar	A1.1/A2.1	**File → Save As**	
File Save Dialog	-	File Name: **ex5b.aqu** **Ok**	save definitions to "ex5b.aqu"
Confirmation Message	-	**Ok**	

Definition of Compartments

The boundary layer resistance must be introduced for all substances and detachment velocity must be specified:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **Reactor** **Edit**	
Edit Biofilm React. Com.	A3.31	**Particulate Variables**	
Edit Particulate Vars.	A3.32	Select **X1** **Edit**	
Edit Part. Properties	A3.33	Bound. Res.: **LL/D_X** **Ok**	
Edit Particulate Vars.	A3.32	Select **X2** **Edit**	
Edit Part. Properties	A3.33	Bound. Res.: **LL/D_X** **Ok**	
Edit Particulate Vars.	A3.32	**Ok**	
Edit Biofilm React. Com.	A3.31	**Dissolved Variables**	
Edit Dissolved Vars.	A3.34	Select **S1** **Edit**	
Edit Diss. Properties	A3.35	Variable: **S1** Bound. Res.: **LL/D_S1** **Ok**	
Edit Dissolved Vars.	A3.34	Select **S2** **Edit**	
Edit Diss. Properties	A3.35	Bound. Res.: **LL/D_S2** **Ok**	
Edit Dissolved Vars.	A3.34	**Ok**	
Edit Biofilm React. Com.	A3.31	Detachment: **if uF > 0 then 0.5*uF else 0 endif** **Ok**	detachment must be non-negative
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation over 50 days can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **1** Num. Steps: **50** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **LF_t** **Plot to Screen** Select **S_z** **Plot to Screen** Select **X1_z** **Plot to Screen** Select **X2_z** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **LF_t** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex5a.ps** **Ok**	
Confirmation Message	-	**Ok**	
View Results	A5.2	Select **S_z** **Plot to File**	

File Save Dialog	-	File Name: **ex5a.ps** **Ok**	
Confirmation Message	-	**Ok**	
View Results	A5.2	Select **X1_z** **Plot to File**	
File Save Dialog	-	File Name: **ex5a.ps** **Ok**	
Confirmation Message	-	**Ok**	
View Results	A5.2	Select **X2_z** **Plot to File**	
File Save Dialog	-	File Name: **ex5b.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex5b.ps" can be submitted to a printer in a system dependent way. Fig. A6.14 shows the plot as it is printed by processing the file "ex5b.ps". Note that the plot definition can be reused for additional calculations.

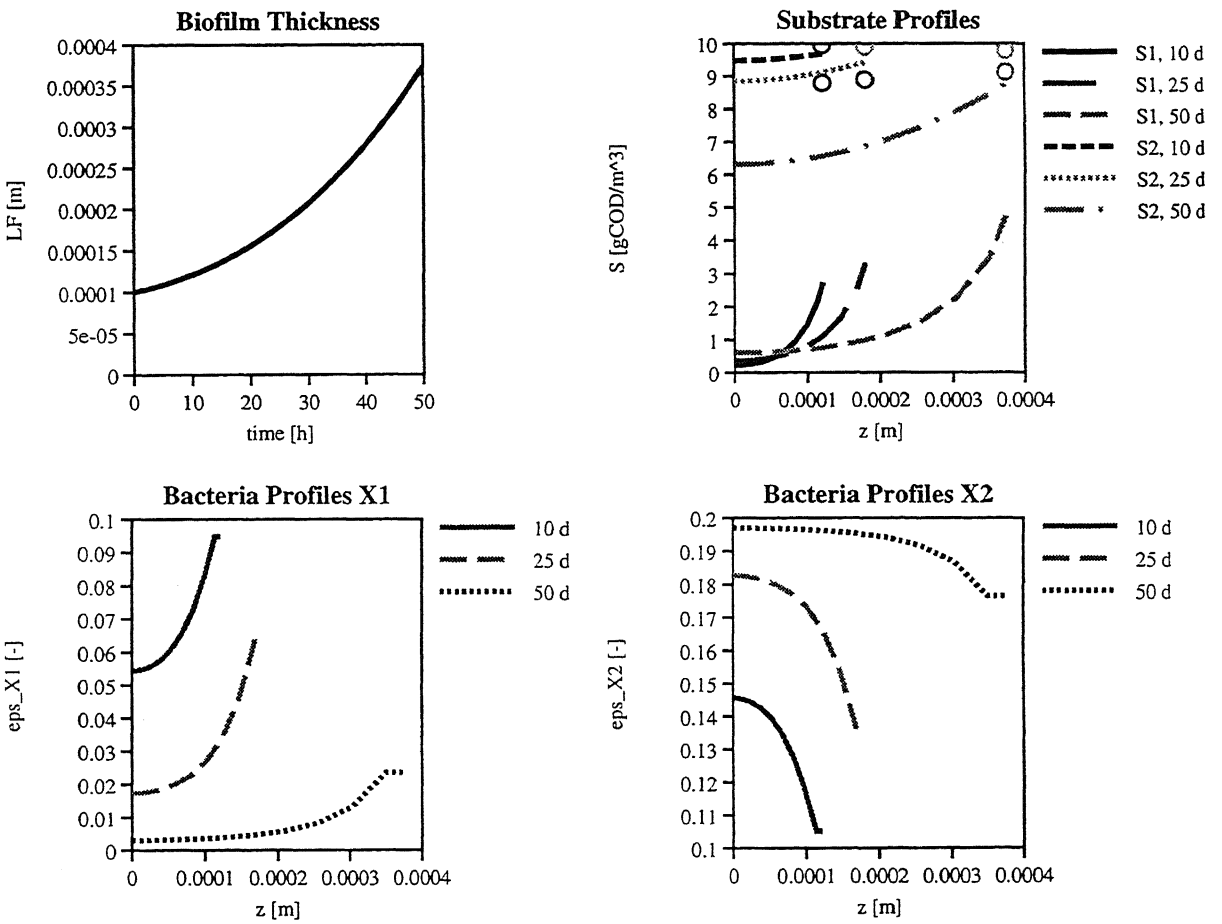


Fig. A6.14: Plot printed by processing the file "ex5b.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	

Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex5b.prn** **Ok**	write system definitions to "ex5b.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex5b.aqu". The system definitions have been written to the text file "ex5b.prn" which can be submitted to a printer in order to check the system definitions.

A6.6 Exercise 6: Tracer Transport in a River

This exercise introduces the river section compartment.

Part A: Calculate spreading of a pulse of a conservative tracer in a river. The geometry of the river bed is approximated as a rectangular channel with a width of 10 m and a slope of 0.001. Friction should be calculated according to equation (A3.5) due to Strickler with a friction coefficient K_{st} of 25 $m^{1/3}/s$. The estimate of Fischer given by equation (A3.6) should be used for longitudinal dispersion. The length of the river section is 2000 m, river discharge is 2 m^3/s. Upstream input mass flux of the tracer increases from 0 to 2 mg/s during the first 100 s, remains constant during 900 s and then decreases to zero within 100 s. Use 20 cells for numerical discretization and use the low resolution method. Plot the time course of the tracer pulse at the locations 0 m, 1000 m and 2000 m.

Part B: Repeat the above calculation for all combinations of

with dispersion ↔ without dispersion

20 cells discretization ↔ 50 cells discretization

low resolution ↔ high resolution

and discuss the results.

Solution to Part A:

Start the window interface version of AQUASIM or select File → New to remove previously entered data from memory.

Definition of Variables

A dynamic state variable representing tracer concentration and a series of program variables needed for the definition of river geometry, friction and dispersion within the river section compartment must be defined. Furthermore, it is recommendable to specify model parameters as variables:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Variables**	
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**State Variable** **Ok**	
Edit State Variable	A3.6	Name: **C** Description: **Conc. of tracer** Unit: **mg/m^3** **Ok**	dynamic volume state variable describing conc. of tracer
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **t** Description: **Time** Unit: **s** Reference to: **Time** **Ok**	time is required for the definition of input time series Fin
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	

Edit Program Variable	A3.7	Name: **x** Description: **Space coordiate along the river** Unit: **m** Reference to: **Space Coord. X** **Ok**	the space coordinate along the river is required for formulating the elevation of the river bed zB
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **z0** Description: **Water level elevat.** Unit: **m** Reference to: **Water Level Elevat.** **Ok**	water level elevation is needed for the formulation of river bed geometry
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **A** Description: **Cross sect. area** Unit: **m^2** Reference to: **Cross Sect. Area** **Ok**	cross sectional area is needed for the formulation of friction and dispersion
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **P** Description: **Perimeter length** Unit: **m** Reference to: **Perimeter length** **Ok**	perimeter length is needed for the formulation of friction
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **Q** Description: **River discharge** Unit: **m^3/s** Reference to: **Discharge** **Ok**	river discharge is needed for the formulation of friction and dispersion
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **w** Description: **Surface width** Unit: **m** Reference to: **Surface Width** **Ok**	surface width is needed for the formulation of dispersion
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Program Variable** **Ok**	
Edit Program Variable	A3.7	Name: **Sf** Description: **Friction slope** Reference to: **Friction Slope** **Ok**	friction slope is necessary for the formulation of shear velocity (for dispersion)
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **wB** Description: **River bed width** Unit: **m** Expression: **10** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	

Edit Formula Variable	A3.13	Name: **S0** Description: **River slope** Expression: **0.001** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **Kst** Description: **Strickler coefficient** Unit: **m^(1/3)/s** Expression: **25** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **cF** Description: **Fischer coefficient** Expression: **0.011** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **Qin** Description: **Discharge of water** Unit: **m^3/s** Expression: **2** **Ok**	formula variables can be used to define constant parameters
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Real List Variable** **Ok**	
Edit Real List Variable	A3.9	Name: **Fin** Description: **Tracer input flux** Unit: **mg/s** Argument: **t** Stand. Dev.: **global** Rel. St.Dev.: **0** Abs. St.Dev.: **1** Minimum: **0** Maximum: **10** Argument: **0** Value: **0** **Add** Argument: **100** Value: **2** **Add** Argument: **1000** Value: **2** **Add** Argument: **1100** Value: **0** **Add** Int. Method: **linear** **Ok**	definition of a time series representing the desired time course of the input mass flux (below the list box)
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **zB** Description: **Bed elevation** Unit: **m** Expression: **-S0*x** **Ok**	bed elevation assuming zero elevation at x = 0 m
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	

Edit Formula Variable	A3.13	Name: **dmax** Description: **Maximum water depth** Unit: **m** Expression: **z0-zB** **Ok**	maximum water depth as difference of water and bed elevations
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **dmean** Description: **Mean water depth** Unit: **m** Expression: **A/w** **Ok**	river depth is useful the formulation of shear velocity, and dispersion
Edit Variables	A3.3	**New**	
Select Variable Type	A3.4	**Formula Variable** **Ok**	
Edit Formula Variable	A3.13	Name: **ustar** Description: **Shear velocity** Unit: **m/s** Expression: **sqrt(9.8*dmean*Sf)** **Ok**	shear velocity according to equation (3.7)
Menu Bar	A1.1/A2.1	**File → Save**	
File Save Dialog	-	File Name: **ex6a.aqu** **Ok**	save definitions to "ex6a.aqu"
Confirmation Message	-	**Ok**	

Definition of Compartments

A single river section reactor compartment can be used to solve the problem:

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	**New**	
Select Compartment Type	A3.21	**River Section Comparment** **Ok**	
Edit River Section Comp.	A3.37	Name: **River** **Variables**	
Select Active Variables	A3.23	Select **C** **Activate** **Ok**	activate the state variable C
Edit River Section Comp.	A3.37	**Init. Cond.**	
Edit Initial Conditions	A3.25	**Add**	
Edit Initial Condition	A3.26	Variable: **Q** Init. Cond.: **Qin** **Ok**	initial condition of discharge equal to inflow
Edit Initial Conditions	A3.25	**Ok**	
Edit River Section Comp.	A3.37	**Input**	
Select Input Type	A3.38	**Upstream Input** **Ok**	
Edit Inputs	A3.27	Water Inflow: **Qin** **Add**	water inflow as specified in Qin
Edit Input Flux	A3.28	Variable: **C** Input Flux: **Fin**	input flux as specified above
Edit Inputs	A3.27	**Ok**	
Edit River Section Comp.	A3.37	Start Coord.: **0** End Coord.: **2000** Cross Sect.: **dmax*wB** Perimeter: **wB+2*dmax** Width: **wB** Frict. Slope: **1/Kst^2*(P/A)^(4/3)*(Q/A)^2** Dispersion: **with dispersion: cF*w^2*(Q/A)^2/ustar/dmean**	length of 2000 m, rectangular profile, friction according to equation (3.5) and dispersion according to equation (3.6); 22 grid points resolve the river

		Method: **kinematic** Num. Grid.: **22** Resolution: **low** **Ok**	section into two end points and 20 cells.
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Numerical Parameters

Since seconds have been used for the unit of time, it is necessary to increase the maximum time step of the integration algorithm. For a linear arrangement of river section compartments, it is recommeded to reduce the number of codiagonals of the jacobian matrix of the system of differential equations to increase computation speed (if more state variable were included, it would be necessary to slightly increase this number):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Numerical Parameters**	
Numerical Parameters	A4.2	Max. Time Step: **100** Number of Codiag: **10** **Ok**	

Definition of Plots

Graphical presentation of results requires a plot definition. Once a plot definition has been specified, it can be used for several calculations with different parameters or model structures. It is advantageous to specify the plot definitions before performing the calculation:

Window	Figure	Action	Comment
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	**New**	
Edit Plot Definition	A5.3	Name: **Conc_t** Title: **20 cells, with disp., low res** Absc. Label: **time [s]** Ord. Label: **C [mg/m^3]** **Add**	define general plot attributes
Edit Curve Definition	A5.4	Variable: **C** Time/Space: **0** Legend: **0 m** Line Style: **solid** Line Color: **red** **Ok**	curve for concentration at 0 m
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C** Time/Space: **1000** Legend: **1000 m** Line Style: **dashed** Line Color: **green** **Ok**	curve for concentration at 1000 m
Edit Plot Definition	A5.3	**Add**	
Edit Curve Definition	A5.4	Variable: **C** Time/Space: **2000** Legend: **2000 m** Line Style: **dotted** Line Color: **blue** **Ok**	curve for concentration at 2000 m
Edit Plot Definition	A5.3	**Scaling**	
Edit Plot Scaling	A5.5	Ord. Min.: **0** auto: **uncheck** Ord. Max.: **1** auto: **uncheck** Ord. T. Pos.: **0** auto: **uncheck** Ord. T. Dist.: **0.2** auto: **uncheck** **Ok**	guarantee the same scaling for all plots

Edit Plot Definition	A5.3	**Ok**	
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Performing the Simulation and Viewing Results

Now initialization and then simulation can be performed:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	Step Size: **20** Num. Steps: **500** **Initialize**	let Calculation Number and Initial Time at their default values of zero
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc_t** **Plot to Screen**	plot results to the screen
View Results	A5.2	Select **Conc_t** **Plot to File**	plot results to a PostScript file
File Save Dialog	-	File Name: **ex6a.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the plot should be visible on the screen and the PostScript file "ex6a.ps" can be submitted to a printer in a system dependent way. Fig. A6.15 shows the plot as it is printed by processing the file "ex6a.ps". Note that the plot definition can be reused for additional calculations.

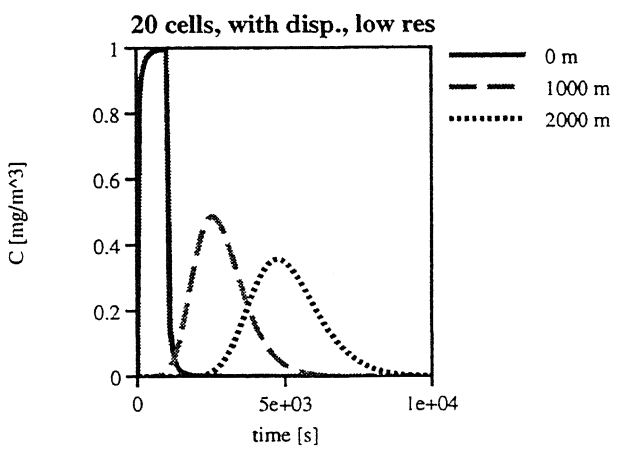


Fig. A6.15: Plot printed by processing the file "ex6a.ps" created in this example

Saving and Printing System Definitions

In order to save disk space, the calculated states can be deleted before saving the definitive version of the system definitions (this could also be done by editing an object of model definition; it is also possible to save system definitions together with calculated states):

Window	Figure	Action	Comment
Menu Bar	A1.1/A3.2	**Edit → Delete States**	
Delete Calculated States	A3.55	Select CalcNum **0** **Delete**	delete all calculated states for calculation number zero
Menu Bar	A1.1/A2.1	**File → Save**	
Confirmation Message	-	**Ok**	

Menu Bar	A1.1/A2.1	**File → Print**	
File Save Dialog	-	File Name: **ex6a.prn** **Ok**	write system definitions to "ex6a.prn"
Confirmation Message	-	**Ok**	

The system can be reloaded from the file "ex6a.aqu". The system definitions have been written to the text file "ex6a.prn" which can be submitted to a printer in order to check the system definitions.

Solution to Part B:

Continue your work after you have performed part A or start the window interface version of AQUASIM and load the file "ex6a.aqu" saved in part A with the commands File → Open, File Name: exp6a.aqu, Ok, Ok.

Editing, Calculating and Plotting various Versions

A cyclic procedure involving modification of compartment definition, simulation and plotting of results has to be performed to solve part B of the exercise:

Window	Figure	Action	Comment
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	**Initialize**	
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc_t** **Plot to File**	
File Save Dialog	-	File Name: **ex6b.ps** **Ok**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **River** **Edit**	
Edit River Section Comp.	A3.37	Resolution: **high** **Ok**	
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	**Initialize**	
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc_t** **Edit**	
Edit Plot Definition	A5.3	Title: **20 cells, with disp., high res** **Ok**	
View Results	A5.2	Select **Conc_t** **Plot to File**	
File Save Dialog	-	File Name: **ex6b.ps** **Ok**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **River** **Edit**	
Edit River Section Comp.	A3.37	Dispersion: **without dispersion** Resolution: **low** **Ok**	
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	**Initialize**	

Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc_t** **Edit**	
Edit Plot Definition	A5.3	Title: **20 cells, without disp., low res** **Ok**	
View Results	A5.2	Select **Conc_t** **Plot to File**	
File Save Dialog	-	File Name: **ex6b.ps** **Ok**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **River** **Edit**	
Edit River Section Comp.	A3.37	Resolution: **high** **Ok**	
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	**Initialize**	
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc_t** **Edit**	
Edit Plot Definition	A5.3	Title: **20 cells, without disp., high res** **Ok**	
View Results	A5.2	Select **Conc_t** **Plot to File**	
File Save Dialog	-	File Name: **ex6b.ps** **Ok**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **River** **Edit**	
Edit River Section Comp.	A3.37	Dispersion: **with dispersion** Num. Grid.: **52** Resolution: **low** **Ok**	
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	**Initialize**	
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc_t** **Edit**	
Edit Plot Definition	A5.3	Title: **50 cells, with disp., low res** **Ok**	
View Results	A5.2	Select **Conc_t** **Plot to File**	
File Save Dialog	-	File Name: **ex6b.ps** **Ok**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **River** **Edit**	
Edit River Section Comp.	A3.37	Resolution: **high** **Ok**	
Menu Bar	A1.1/A4.1	**Calc → Simulation**	

Simulation	A4.2	**Initialize**	
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc_t** **Edit**	
Edit Plot Definition	A5.3	Title: **50 cells, with disp., high res** **Ok**	
View Results	A5.2	Select **Conc_t** **Plot to File**	
File Save Dialog	-	File Name: **ex6b.ps** **Ok**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **River** **Edit**	
Edit River Section Comp.	A3.37	Dispersion: **without dispersion** Resolution: **low** **Ok**	
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	**Initialize**	
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc_t** **Edit**	
Edit Plot Definition	A5.3	Title: **50 cells, without disp., low res** **Ok**	
View Results	A5.2	Select **Conc_t** **Plot to File**	
File Save Dialog	-	File Name: **ex6b.ps** **Ok**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A3.2	**Edit → Compartments**	
Edit Compartments	A3.20	Select **River** **Edit**	
Edit River Section Comp.	A3.37	Resolution: **high** **Ok**	
Menu Bar	A1.1/A4.1	**Calc → Simulation**	
Simulation	A4.2	**Initialize**	
Confirmation Message	-	**Ok**	
Simulation	A4.2	**Start**	
Confirmation Message	-	**Ok**	
Menu Bar	A1.1/A5.1	**View → Results**	
View Results	A5.2	Select **Conc_t** **Edit**	
Edit Plot Definition	A5.3	Title: **50 cells, without disp., high res** **Ok**	
View Results	A5.2	Select **Conc_t** **Plot to File**	
File Save Dialog	-	File Name: **ex6b.ps** **Ok**	
Confirmation Message	-	**Ok**	

Now the PostScript file ex6b.ps containing all 8 plots can be submitted to a printer. Fig. A6.16 shows the plot as it is printed by processing the file "ex6b.ps". This plot shows that high resolution discretization strongly decreases numercal diffusion, but that nevertheless 50 cells are necessary in order to keep numerical diffusion for the sharp step input function small.

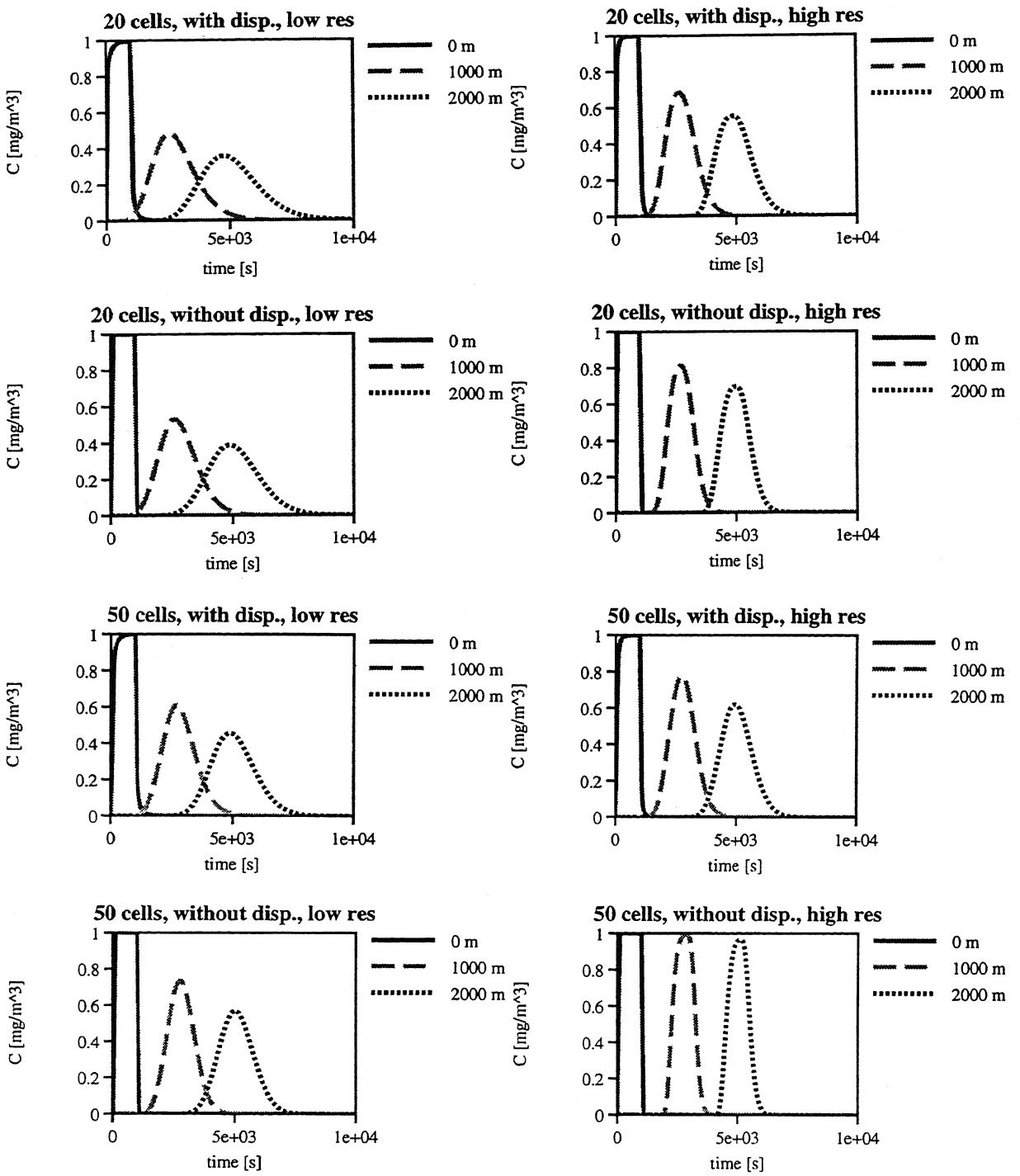


Fig. A6.16: Plot printed by processing the file "ex6b.ps" created in this example

A7 Appendix

A7.1 Window Interface Version - Main Dialog Hierarchy

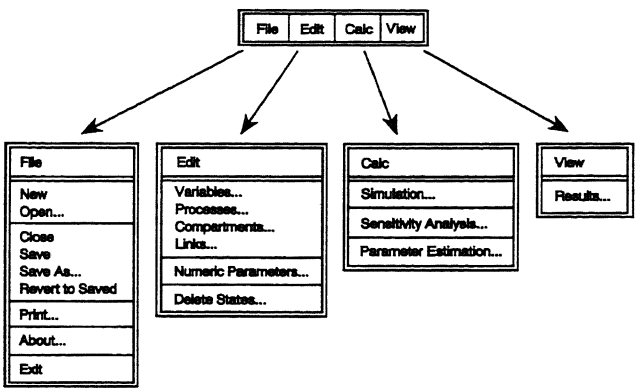


Fig. A7.1: Menu bar.

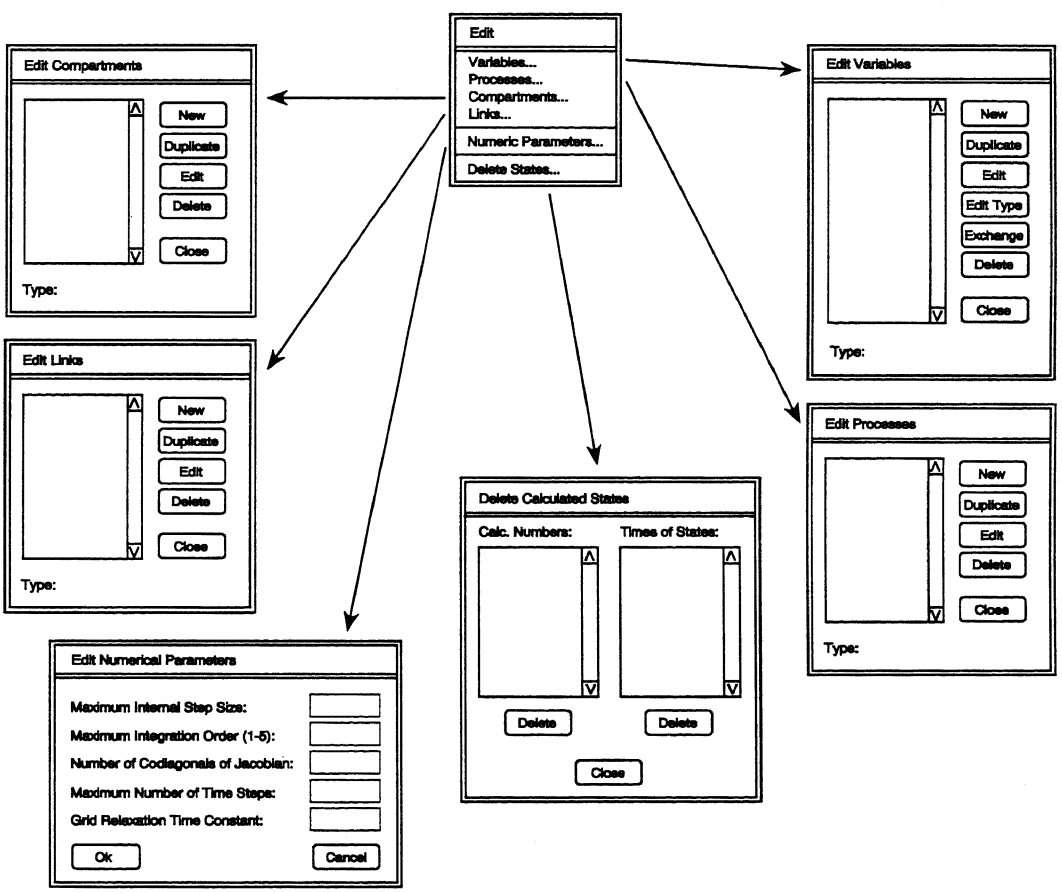


Fig. A7.2: Edit menu.

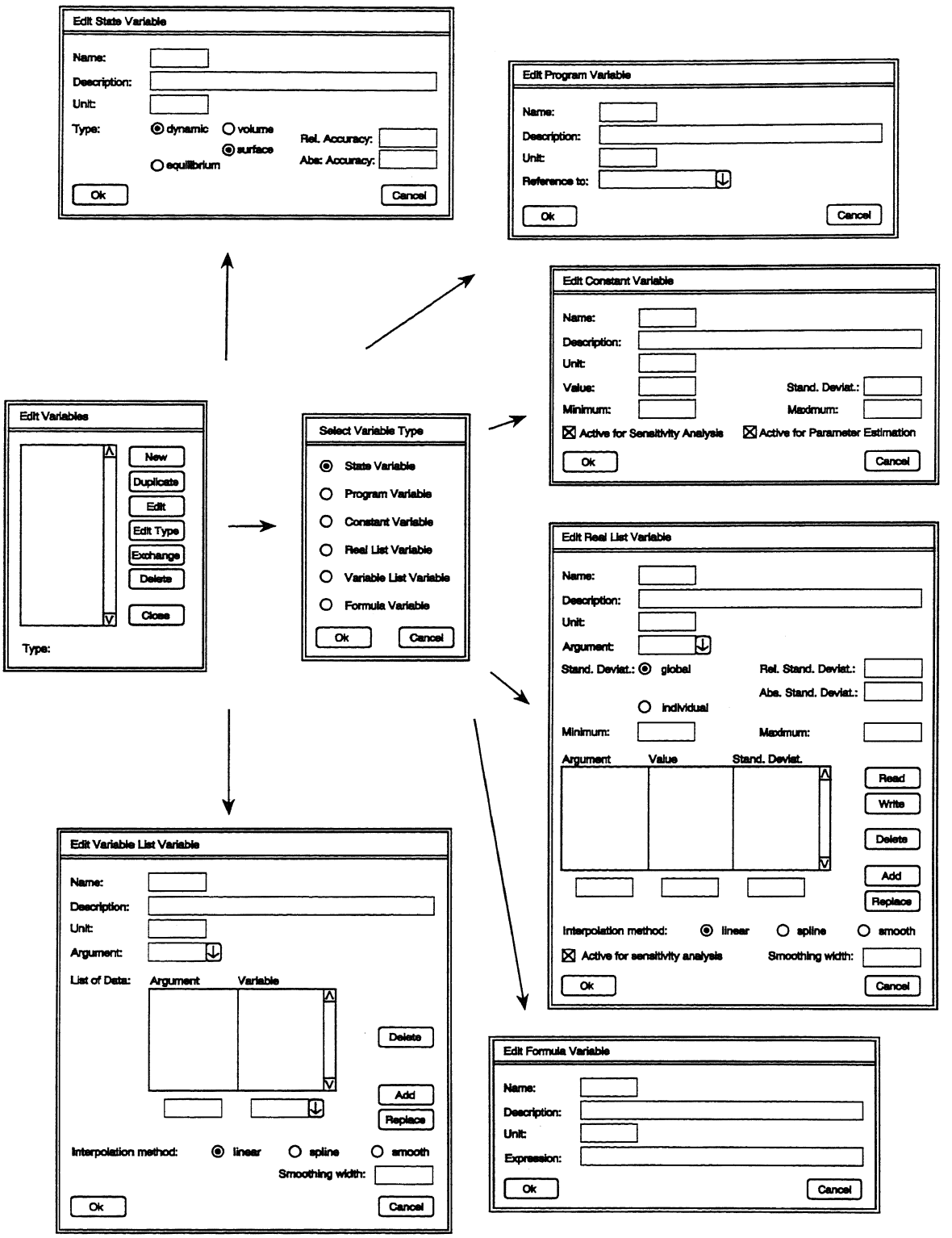


Fig. A7.3: Edit variables dialog boxes.

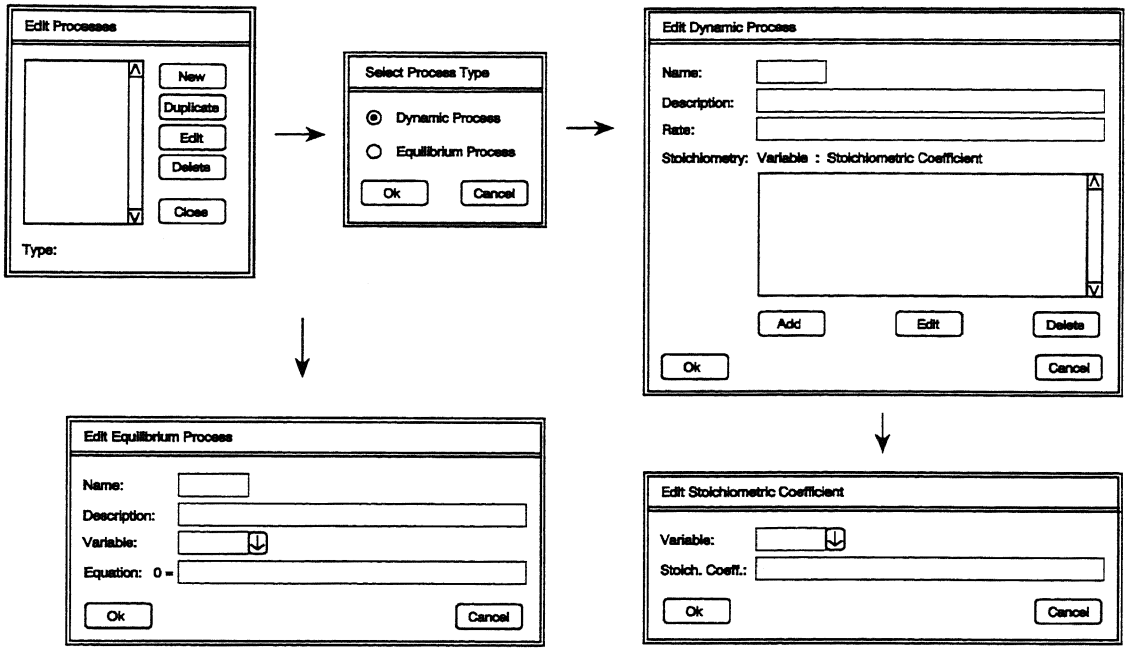


Fig. A7.4: Edit processes dialog boxes.

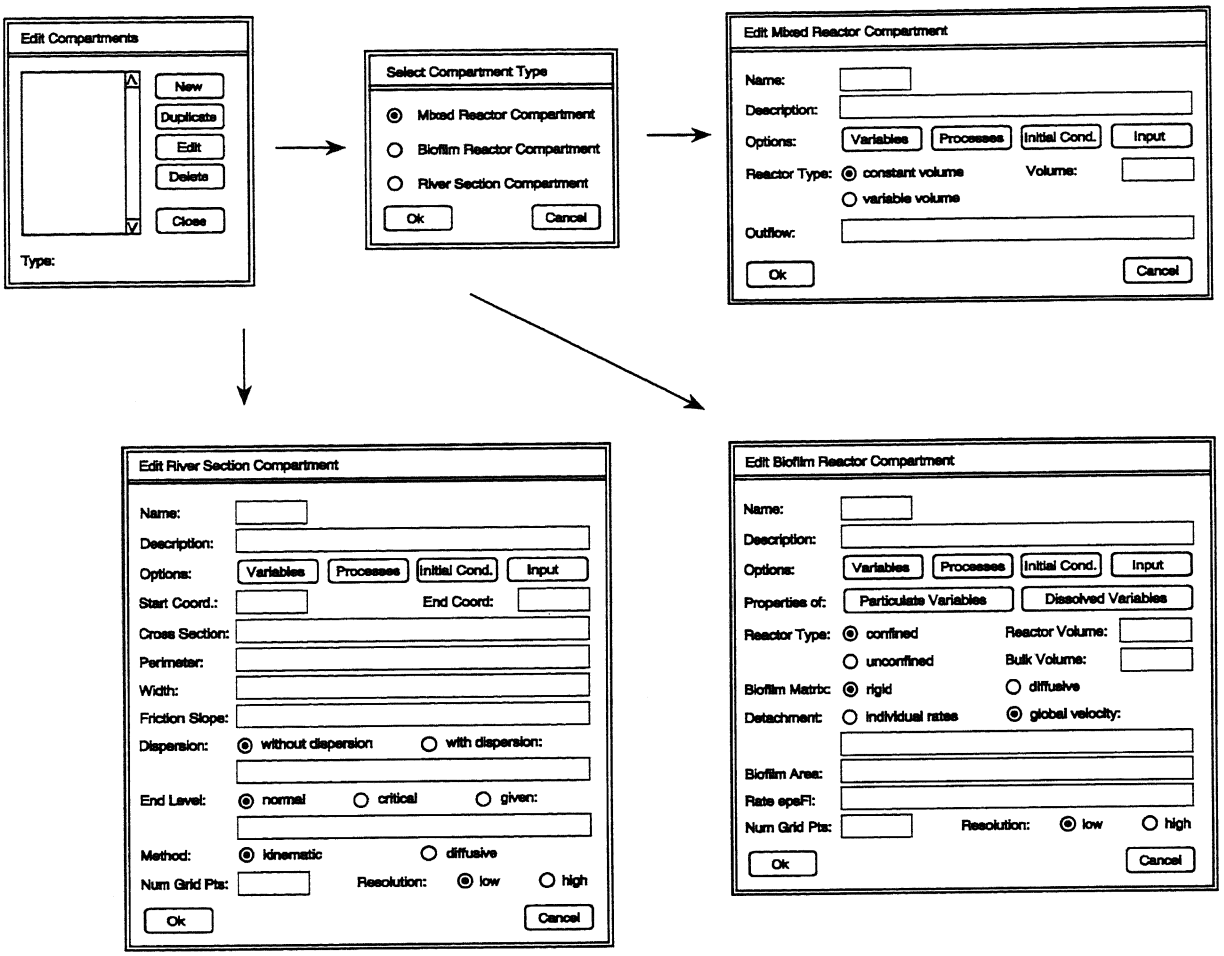


Fig. A7.5: Edit compartments dialog boxes.

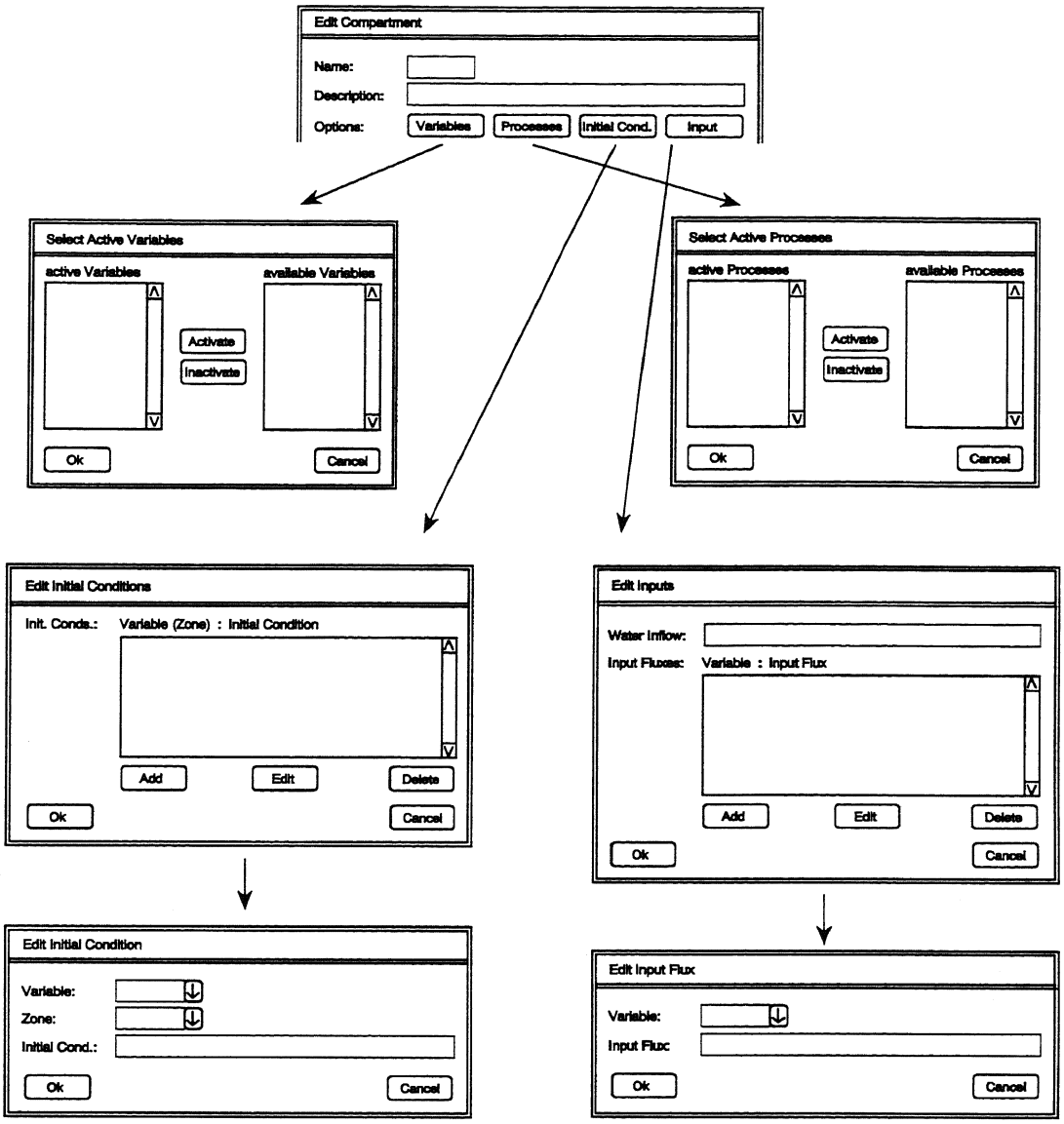


Fig. A7.6: Edit compartment options dialog boxes.

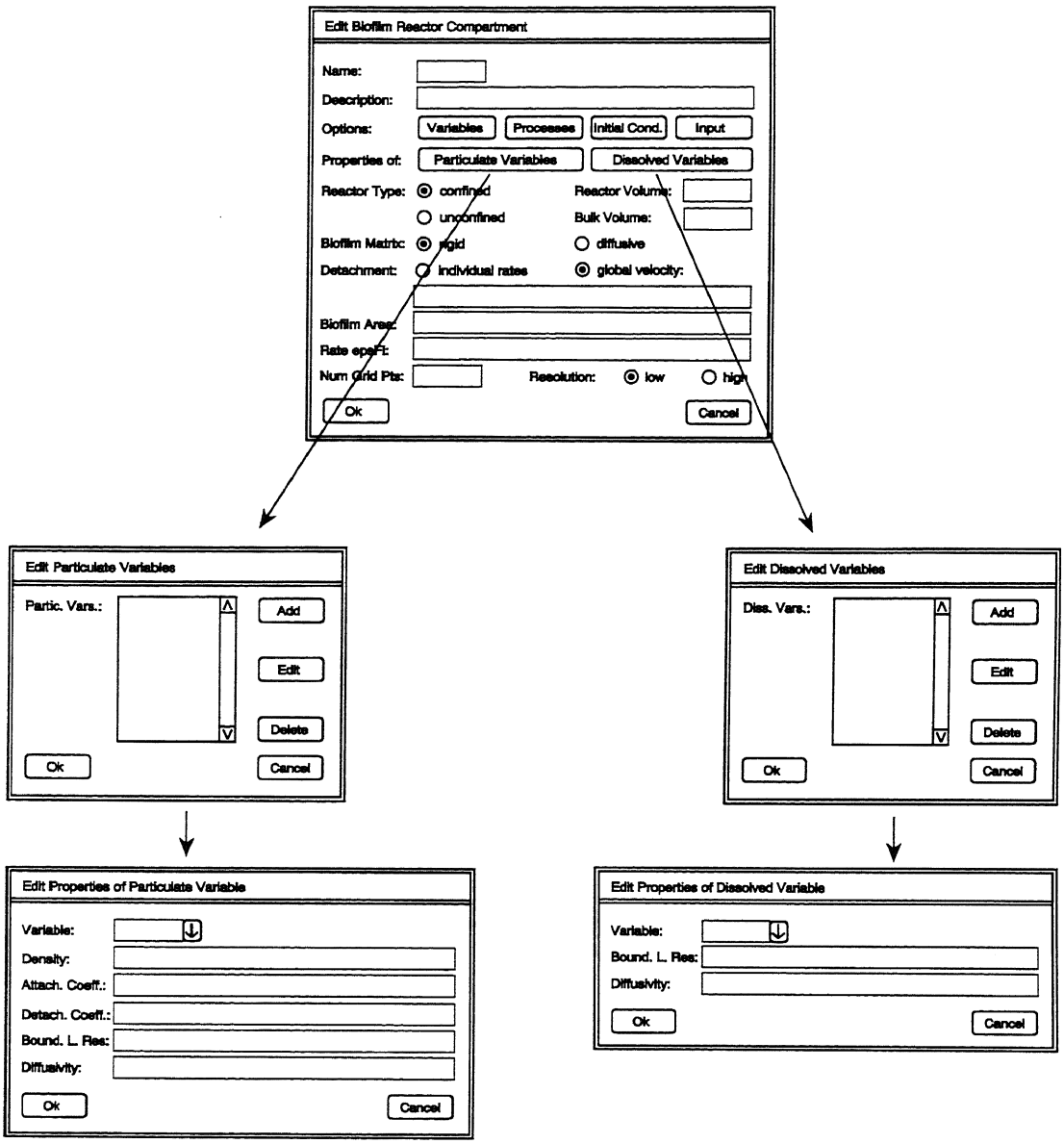


Fig. A7.7: Edit biofilm reactor compartment dialog boxes.

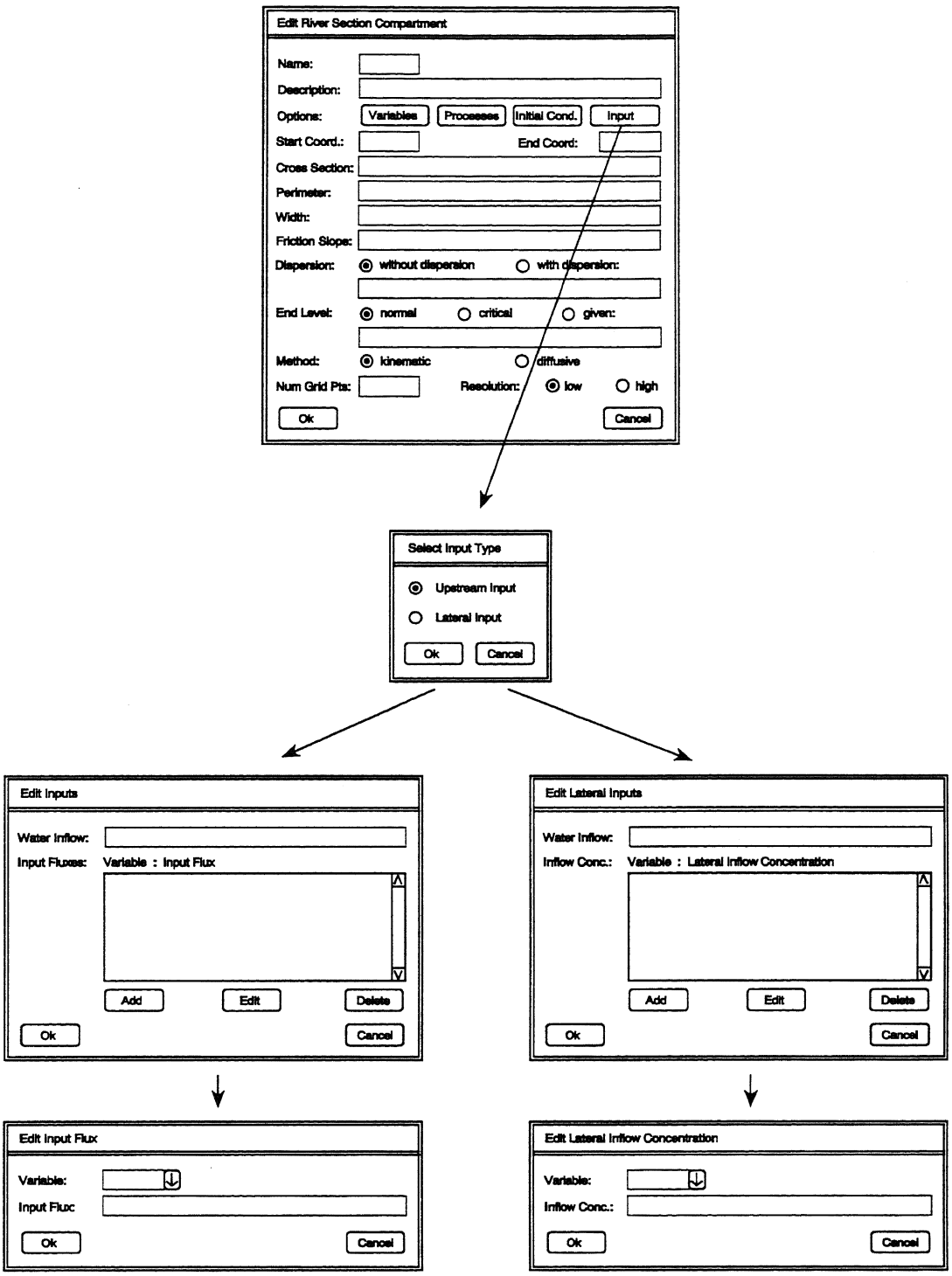


Fig. A7.8: Input definition dialog boxes for the river section compartment.

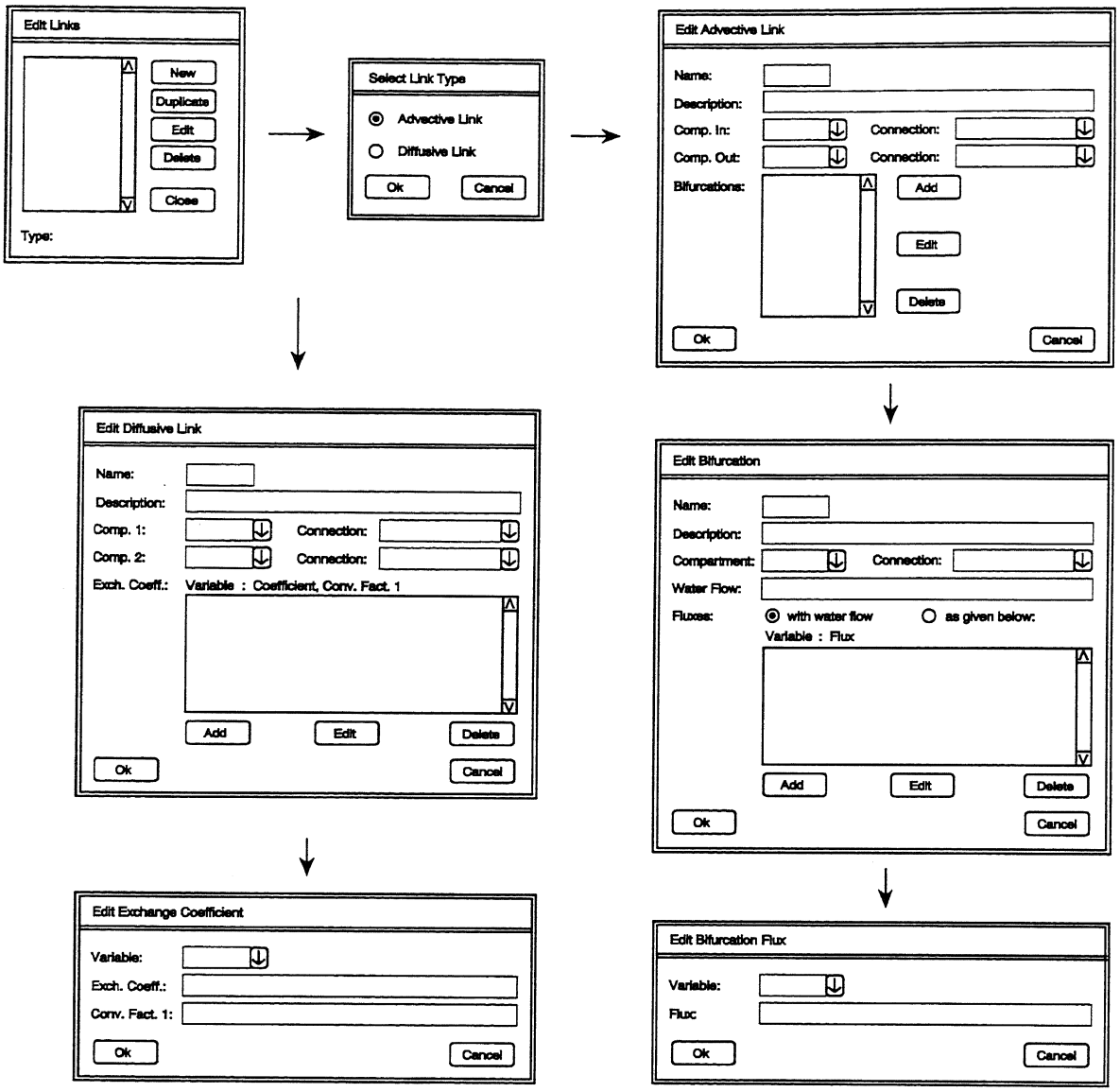


Fig. A7.9: Edit links dialog boxes.

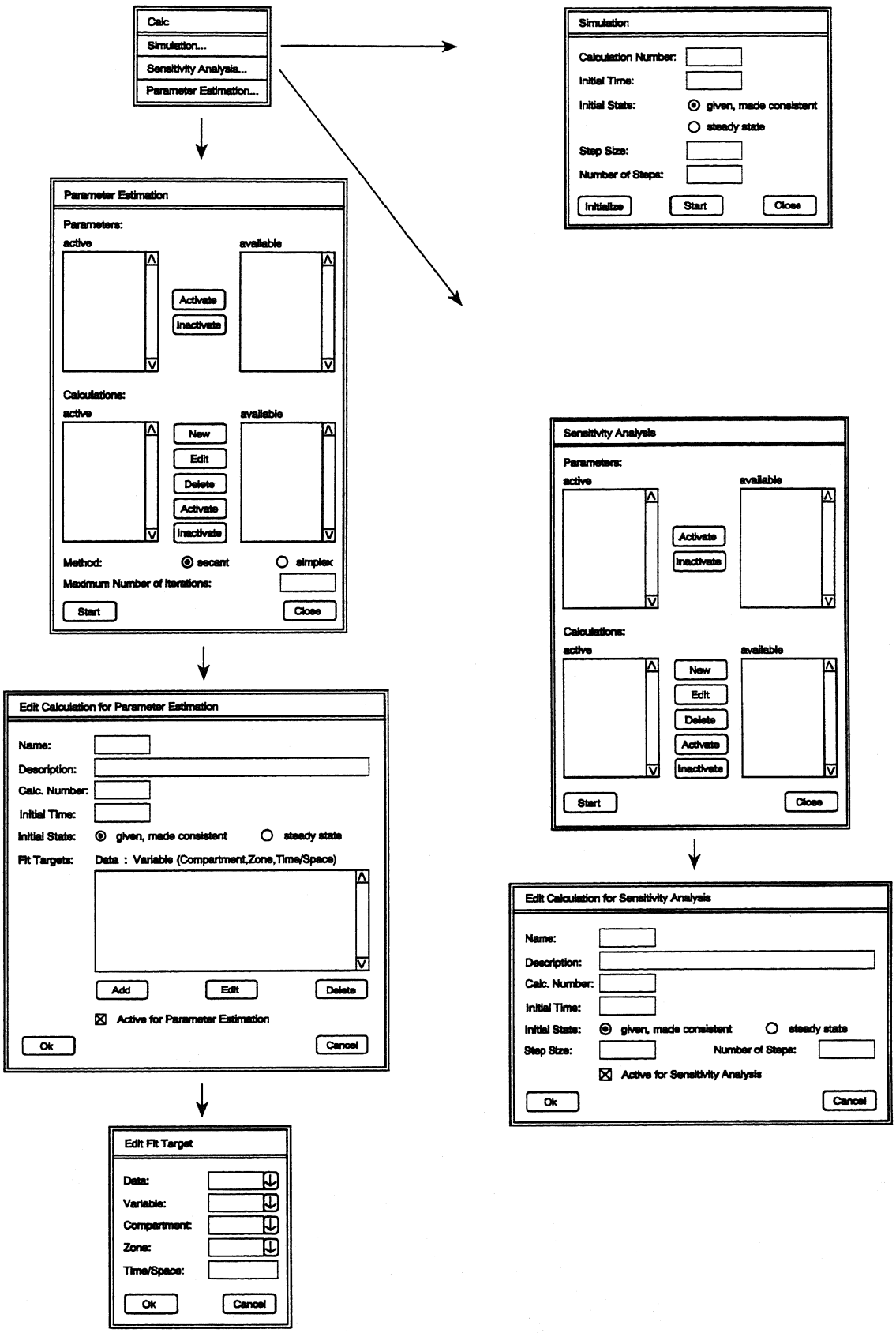


Fig. A7.10: Calc menu.

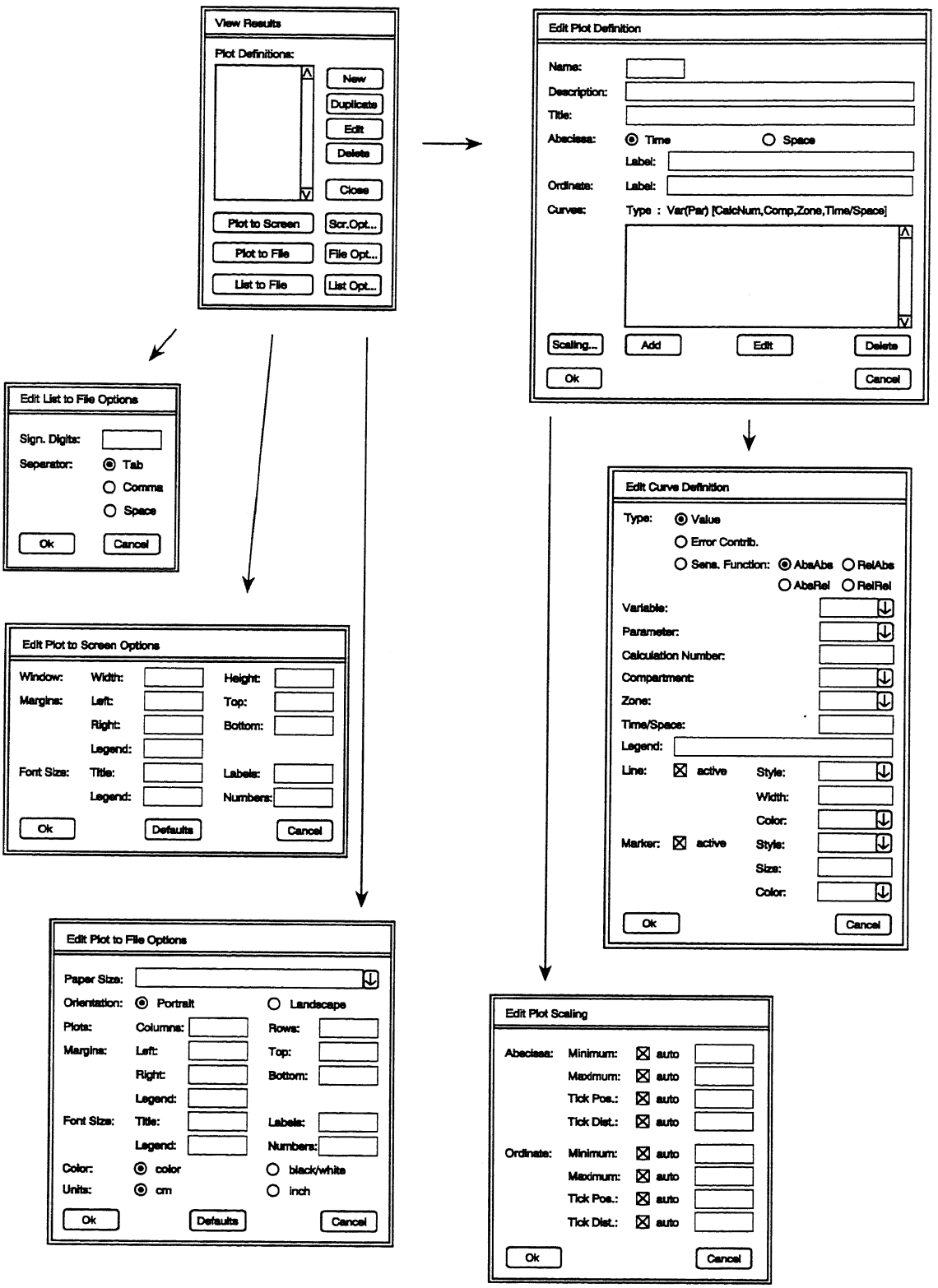


Fig. A7.11: View results dialog boxes.

A7.2 Character Interface Version - Example of a Dialog

In chapters A2 - A5 of this manual, operation of the window interface version of AQUASIM is described in detail. Because the structure of all menus and dialogs is the same in the character interface version, these chapters can also be used as a reference for the character interface version. Fig. A7.12 shows a dialog to give an impression of the behaviour of this version. User input always follows the prompt ">" of AQUASIM. If a default value is given in square brackets "[...]", this default value can be accepted by simply pressing "Return"; if no default is given, or if the default is not to be used, the user can type the desired expression followed by "Return". Selection of menu items is done by typing the number of the desired item and pressing "Return". In the case of a menu, the option "Back", selected by "B" or "b" followed by "Return" leads back to the previous menu of the hierarchy. The same action is performed by simply pressing "Return" without selecting a menu item. List boxes of the window interface version are replaced by listings of the entries of the list box. To clarify the dialog, in the example of a short session of the character interface version of AQUASIM shown in Fig. A7.12, user inputs are written in bold and comments are added in italics. This example session corresponds to the solution of Exercise 2A discribed in section A6.2 of the tutorial for the window interface version.

```
*********************************************
*                                           *
*   AQUASIM - Simulation of Aquatic Systems   *
*                                           *
*********************************************

AQUASIM - Main Menu
-------------------

   1   =   File
   2   =   Edit
   3   =   Calc
   4   =   View
   >2

AQUASIM - Edit Menu
-------------------

   1   =   Variables
   2   =   Processes
   3   =   Compartments
   4   =   Links
   5   =   Numeric Parametes
   6   =   Delete States

   B   =   Back
   >1

AQUASIM - Edit Variables Menu
-----------------------------

   1   =   New
   2   =   Duplicate
   3   =   Edit
   4   =   Edit Type
   5   =   Exchange
   6   =   Delete

   B   =   Back
   >1

   Select Variable Type
   1   =   State Variable
   2   =   Program Variable
   3   =   Constant Variable
```

```
    4   =   Real List Variable
    5   =   Variable List Variable
    6   =   Formula Variable

    B   =   Back
    >1

    Enter name [var1]
    >C_A

    Enter description []
    >Concentration of A

    Enter unit []
    >mg/l

    Select type
    1   =   dynamic volume
    2   =   dynamic surface
    3   =   equilibrium
    [1]
    >                                          "Return" accepts selection "1"

    Enter relative accuracy [1e-06]
    >                                          "Return" accepts proposed accuracy

    Enter absolute accuracy (mg/l) [1e-06]
    >                                          "Return" accepts proposed accuracy

    The variable has successfully been added

 Actual list of variables:

    C_A

 AQUASIM - Edit Variables Menu
 -----------------------------

    1   =   New
    2   =   Duplicate
    3   =   Edit
    4   =   Edit Type
    5   =   Exchange
    6   =   Delete

    B   =   Back
    >1

    Select Variable Type
    1   =   State Variable
    2   =   Program Variable
    3   =   Constant Variable
    4   =   Real List Variable
    5   =   Variable List Variable
    6   =   Formula Variable

    B   =   Back
    >6

    Enter name [var2]
    >k_A

    Enter description []
    >First order degr. rate constant of A

    Enter unit []
    >1/h

    Enter formula
    [0]
    >1

    The variable has successfully been added

 Actual list of variables:

    C_A          k_A

 AQUASIM - Edit Variables Menu
 -----------------------------

    1   =   New
    2   =   Duplicate
    3   =   Edit
```

```
   4   =   Edit Type
   5   =   Exchange
   6   =   Delete

   B   =   Back
   >                                          "Return" leads back to the Main Menu

AQUASIM - Main Menu
-------------------

   1   =   File
   2   =   Edit
   3   =   Calc
   4   =   View
   >1

AQUASIM - File Menu
-------------------

   1   =   New
   2   =   Open
   3   =   Close
   4   =   Save
   5   =   Save As
   6   =   Revert to Saved
   7   =   Print
   8   =   About

   E   =   Exit

   B   =   Back
   >4

   Enter name of file to save to []
   >ex2a.aqu

   System successfully saved to file "ex2a.aqu"

AQUASIM - Main Menu
-------------------

   1   =   File
   2   =   Edit
   3   =   Calc
   4   =   View
   >2

AQUASIM - Edit Menu
-------------------

   1   =   Variables
   2   =   Processes
   3   =   Compartments
   4   =   Links
   5   =   Numeric Parametes
   6   =   Delete States

   B   =   Back
   >2

AQUASIM - Edit Processes Menu
-----------------------------

   1   =   New
   2   =   Duplicate
   3   =   Edit
   4   =   Delete

   B   =   Back
   >1

   Select Process Type
   1   =   Dynamic Process
   2   =   Equilibrium Process

   B   =   Back
   >1

   Enter name [proc1]
   >Degr_A

   Enter description []
   >Degradation of A
```

```
    Enter rate
    []
    >k_A*C_A

    Stoichiometry of process Degr_A:

    Select edit option
    1   =   Add stoichiometric coefficient
    2   =   Edit stoichiometric coefficient
    3   =   Delete stoichiometric coefficient

    B   =   Back
    >1

    Enter desired position of new stoichiometric coefficient [1]
    >                                          "Return" accepts default position

    Enter name of (potential) differential state variable []
    >C_A

    Enter expression of stoichiometric coefficient []
    >-1

    Stoichiometric coefficient successfully added

    Stoichiometry of process Degr_A:

    Index   Variable  :  Stoichiometric coefficient
      1     C_A  :  -1

    Select edit option
    1   =   Add stoichiometric coefficient
    2   =   Edit stoichiometric coefficient
    3   =   Delete stoichiometric coefficient

    B   =   Back
    >                                          "Return" concludes editing stoich. coeff.

    The process has successfully been added

Actual list of processes

    Degr_A

AQUASIM - Edit Processes Menu
-----------------------------

    1   =   New
    2   =   Duplicate
    3   =   Edit
    4   =   Delete

    B   =   Back
    >                                          "Return" leads back to the Main Menu

AQUASIM - Main Menu
-------------------

    1   =   File
    2   =   Edit
    3   =   Calc
    4   =   View
    >2

AQUASIM - Edit Menu
-------------------

    1   =   Variables
    2   =   Processes
    3   =   Compartments
    4   =   Links
    5   =   Numeric Parametes
    6   =   Delete States

    B   =   Back
    >3

AQUASIM - Edit Compartments Menu
--------------------------------

    1   =   New
    2   =   Duplicate
    3   =   Edit
    4   =   Delete
```

```
B   =   Back
>1

Select Compartment Type
1   =   Mixed Reactor Compartment
2   =   Biofilm Reactor Compartment
3   =   River Section Compartment

B   =   Back
>1

Enter name [comp1]
>Reactor

Enter description []
>Batch reactor

List of available variables:

C_A          k_A

List of active variables:

Select edit option:
1   =   Activate variable
2   =   Inactivate variable
B   =   Back
>1

Enter name of variable to be activated []
>C_A

Enter desired position of new active variable [1]
>                                          "Return" accepts default position

Variable successfully activated

List of available variables:

C_A          k_A

List of active variables:

C_A

Select edit option:
1   =   Activate variable
2   =   Inactivate variable
B   =   Back
>                                          "Return" concludes activating variables

List of available processes:

Degr_A

List of active processes:

Select edit option:
1   =   Activate process
2   =   Inactivate process
B   =   Back
>1

Enter name of process to be activated []
>Degr_A

Enter desired position of new active process [1]
>                                          "Return" accepts default position

Process successfully activated

List of available processes:

Degr_A

List of active processes:

Degr_A

Select edit option:
1   =   Activate process
2   =   Inactivate process
B   =   Back
```

```
    >                                              "Return" concludes activating processes

    Initial conditions of the compartment Reactor:

    Select edit option
    1   =   Add initial condition
    2   =   Edit initial condition
    3   =   Delete initial condition

    B   =   Back
    >1

    Enter desired position of new initial condition [1]
    >                                              "Return" accepts default position

    Enter name of (potential) state variable []
    >C_A

    Enter initial condition []
    >1

    Initial condition successfully added

    Initial conditions of the compartment Reactor:

    Index   Variable(Zone)   :  Initial condition
      1     C_A (Bulk Volume)  :  1

    Select edit option
    1   =   Add initial condition
    2   =   Edit initial condition
    3   =   Delete initial condition

    B   =   Back
    >                                          "Return" concludes editing initial conditions

    Enter expression for inflow
    [0]
    >                                          "Return" accepts default inflow of zero

    Input fluxes into the compartment Reactor:

    Select edit option
    1   =   Add input flux
    2   =   Edit input flux
    3   =   Delete input flux

    B   =   Back
    >                                          "Return" concludes editing input fluxes

    Select reactor type
    1   =   constant volume
    2   =   variable volume
    [1]
    >                                          "Return" accepts default type

    Enter volume [1]
    >10

    The compartment has successfully been added

Actual list of compartments:

    Reactor

AQUASIM - Edit Compartments Menu
--------------------------------

    1   =   New
    2   =   Duplicate
    3   =   Edit
    4   =   Delete

    B   =   Back
    >                                          "Return" leads back to the Main Menu

AQUASIM - Main Menu
-------------------

    1   =   File
    2   =   Edit
    3   =   Calc
    4   =   View
    >4
```

```
AQUASIM - View Menu
-------------------

   1    =    New Plot Definition
   2    =    Duplicate Plot Definition
   3    =    Edit Plot Definition
   4    =    Delete Plot Definition
   5    =    Options for Plot to File
   6    =    Options for List to File
   7    =    Plot to File
   8    =    List to File

   B    =    Back
   >1

   Enter name [plot1]
   >Conc

   Enter description []
   >                                          "Return" skips description

   Enter Title []
   >Degradation of A

   Select Abscissa
   1    =    Time
   2    =    Space
   [1]
   >                                          "Return" accepts default abscissa time

   Enter Abscissa Label []
   >time [h]

   Enter Ordinate Label []
   >C [mg/l]

   Select edit opiton:
   1    =    Edit curves
   2    =    Edit scaling

   B    =    Back
   >1

   Curves:


   Select edit option:
   1    =    New curve
   2    =    Edit curve
   3    =    Delete curve

   B    =    Back
   >1

   Enter position where to insert new curve [1]
   >                                          "Return" accepts default position

   Select curve type
   1    =    Value
   2    =    Error contribution
   3    =    Absolute absolute sensitivity function
   4    =    Relative absolute sensitivity function
   5    =    Absolute relative sensitivity function
   6    =    Relative relative sensitivity function
   [1]
   >1

   Enter name of variable []
   >C_A

   Enter calculation number [0]
   >                                          "Return" accepts default calculation number

   Enter space or time [0]
   >                                          "Return" accepts default space coordinate

   Enter legend []
   >A

   Select line status
   1    =    active
   2    =    inactive
   [1]
```

```
   >                                              "Return" accepts line status

   Select line style
   1   =   solid
   2   =   long dashed
   3   =   dashed
   4   =   short dashed
   5   =   dotted
   6   =   long dashdot
   7   =   short dashdot
   [1]
   >                                              "Return" accepts line style

   Enter line width [2]
   >                                              "Return" accepts line width

   Select line color
   1   =   black
   2   =   red
   3   =   green
   4   =   blue
   5   =   cyan
   6   =   magenta
   7   =   yellow
   8   =   white
   [1]
   >2

   Select marker status
   1   =   active
   2   =   inactive
   [2]
   >                                              "Return" accepts marker status

   Curve successfully added

   Curves:

   Index   Type   :  Variable [CalcNum, Comp, Zone, Space/Time
     1     Value  :  C_A [0, Reactor, Bulk Volume, 0]

   Select edit option:
   1   =   New curve
   2   =   Edit curve
   3   =   Delete curve

   B   =   Back
   >                                              "Return" concludes editing curves

   Select edit opiton:
   1   =   Edit curves
   2   =   Edit scaling

   B   =   Back
   >                                              "Return" concludes editing plot definition

   The plot definition has successfully been added

Actual list of plot definitions

   Conc

AQUASIM - View Menu
-------------------

   1   =   New Plot Definition
   2   =   Duplicate Plot Definition
   3   =   Edit Plot Definition
   4   =   Delete Plot Definition
   5   =   Options for Plot to File
   6   =   Options for List to File
   7   =   Plot to File
   8   =   List to File

   B   =   Back
   >                                              "Return" leads back to the Main Menu

AQUASIM - Main Menu
-------------------

   1   =   File
   2   =   Edit
   3   =   Calc
   4   =   View
```

```
   >1

AQUASIM - File Menu
-------------------

   1    =    New
   2    =    Open
   3    =    Close
   4    =    Save
   5    =    Save As
   6    =    Revert to Saved
   7    =    Print
   8    =    About

   E    =    Exit

   B    =    Back
   >4

   System successfully saved to file "ex2a.aqu"

AQUASIM - Main Menu
-------------------

   1    =    File
   2    =    Edit
   3    =    Calc
   4    =    View
   >3

AQUASIM - Calc Menu
-------------------

   1    =    Simulation
   2    =    Sensitivity Analysis
   3    =    Parameter Estimation

   B    =    Back
   >1

   Select simulation task:
   1    =    Initialize Simulation
   2    =    Start Simulation

   B    =    Back
   >1

   Enter calculation number [0]
   >                                          "Return" accepts default calculation number

   Enter initial time [0]
   >                                          "Return" accepts default initial time

   Select type of initial condition to be calculated
   1    =    Given initial condition made consistent
   2    =    Steady state initial condition
   [1]
   >                                          "Return" accepts default initial state

   Calculating ...
   .

   Initial state successfully calculated

   Select simulation task:
   1    =    Initialize Simulation
   2    =    Start Simulation

   B    =    Back
   >2

   Enter calculation number [0]
   >                                          "Return" accepts default calculation number

   Enter step size [0.1]
   >0.1

   Enter number of time steps [1]
   >100

   Calculating ...
   ..........................

   Solution successfully calculated
```

```
   Select simulation task:
   1   =   Initialize Simulation
   2   =   Start Simulation

   B   =   Back
   >                                          "Return" leads back to the Calculation Menu

AQUASIM - Calc Menu
-------------------

   1   =   Simulation
   2   =   Sensitivity Analysis
   3   =   Parameter Estimation

   B   =   Back
   >                                          "Return" leads back to the Main Menu

AQUASIM - Main Menu
-------------------

   1   =   File
   2   =   Edit
   3   =   Calc
   4   =   View
   >4

Actual list of plot definitions

   Conc

AQUASIM - View Menu
-------------------

   1   =   New Plot Definition
   2   =   Duplicate Plot Definition
   3   =   Edit Plot Definition
   4   =   Delete Plot Definition
   5   =   Options for Plot to File
   6   =   Options for List to File
   7   =   Plot to File
   8   =   List to File

   B   =   Back
   >7

   Enter name of plot to be written to file []
   >Conc

   Enter name of file for writing [ex2a.ps]
   >                                          "Return" accepts default file name

   Data successfully written to file  "ex2a.ps"

Actual list of plot definitions

   Conc

AQUASIM - View Menu
-------------------

   1   =   New Plot Definition
   2   =   Duplicate Plot Definition
   3   =   Edit Plot Definition
   4   =   Delete Plot Definition
   5   =   Options for Plot to File
   6   =   Options for List to File
   7   =   Plot to File
   8   =   List to File

   B   =   Back
   >                                          "Return" leads back to the Main Menu

AQUASIM - Main Menu
-------------------

   1   =   File
   2   =   Edit
   3   =   Calc
   4   =   View
   >2

AQUASIM - Edit Menu
-------------------
```

```
   1   =   Variables
   2   =   Processes
   3   =   Compartments
   4   =   Links
   5   =   Numeric Parametes
   6   =   Delete States

   B   =   Back
   >6

   There are states to the following calculation numbers:
   0

   Select edit option
   1   =   Delete all states corresponding to a calculation number
   B   =   Back
   >1

   Enter calculation number for deleting states [0]
   >                                          "Return" accepts default calculation number

   States successfully deleted

   There are no calculated states

AQUASIM - Main Menu
-------------------

   1   =   File
   2   =   Edit
   3   =   Calc
   4   =   View
   >1

AQUASIM - File Menu
-------------------

   1   =   New
   2   =   Open
   3   =   Close
   4   =   Save
   5   =   Save As
   6   =   Revert to Saved
   7   =   Print
   8   =   About

   E   =   Exit

   B   =   Back
   >4

   System successfully saved to file "ex2a.aqu"

AQUASIM - Main Menu
-------------------

   1   =   File
   2   =   Edit
   3   =   Calc
   4   =   View
   >1

AQUASIM - File Menu
-------------------

   1   =   New
   2   =   Open
   3   =   Close
   4   =   Save
   5   =   Save As
   6   =   Revert to Saved
   7   =   Print
   8   =   About

   E   =   Exit

   B   =   Back
   >e

End of program AQUASIM (normal termination)
```

Fig. A7.12: Dialog with the character interface version of AQUASIM.

A7.3 Batch Version - Command Line Arguments

In contrast to the window and character interface versions of AQUASIM, the batch version does not provide full functionality. The batch version of AQUASIM is designed to be used in combination with one of the two interactive versions. It provides the following five tasks:

- performing a simulation,
- performing a sensitivity analysis,
- performing a parameter estimation,
- plotting results to a PostScript file,
- listing results to a text file.

The technical or environmental system, for which these operations have to be executed, has to be specified with an interactive version of AQUASIM and stored on an AQUASIM system file. The results can be viewed interactively by opening the result AQUASIM system file with an interactive program version, or output files can be opened with a file editor, PostScript interpreter or submitted to a printer. The five above mentioned tasks can be started with the following five command lines:

```
aquasimb -s logfile loadfile savefile [scmdfile]
aquasimb -a logfile loadfile savefile
aquasimb -e logfile loadfile savefile fitfile
aquasimb -p logfile loadfile pcmdfile
aquasimb -l logfile loadfile lcmdfile
```

The parameters used in these command lines have the following meaning:

`-s`	perform a **s**imulation.
`-a`	perform a sensitivity **a**nalysis.
`-e`	perform a parameter **e**stimation.
`-p`	**p**lot results (to a PostScript file).
`-l`	**l**ist results (to a text file).
`logfile`	name of output text file for job log information (CAUTION: an existing file with the same name is overwritten without warning!).
`loadfile`	name of input AQUASIM system file containing model definitions.
`savefile`	name of output AQUASIM system file for storing the results of the calculation (CAUTION: an existing file with the same name is overwritten without warning!). This file has to be loaded for viewing the results.
`fitfile`	name of output text file containing detailed results of the parameter estimation (CAUTION: an existing file with the same name is overwritten without warning!).

`scmdfile` name of input text file (simulation command file) containing the instructions for initialization and step sizes of simulations. This file is optional; if no simulation command file is provided, simulation is initialized and performed according to the entries of the dialog box shown in Fig. 4.2 stored on the system file. If a simulation command file is provided, for each initialization, a line providing the two numbers

```
calcnum inittime
```

(`calcnum` = calculation number, `inittime` = initial time) has to be provided. Note, that two initializations for the same calculation number are useless, because the second initialization deletes all states calculated for the first one (cf. section 4.1). For each dynamic simulation, a line providing the numbers

```
calcnum timestep numsteps
```

(`calcnum` = calculation number, `timestep` = step size for time integration, `numsteps` = number of steps to be performed) has to be given in the simulation command file. Note, that an arbitrary number of such lines allows to perform simulations for different calculation numbers with varying step sizes. In order to increase numerical efficiency, it is recommendable to group the simulations according to their calculation numbers and not to mix calculations for different calculation numbers.

`pcmdfile` Name of an input text file (plot command file) containing the instructions of which plots to plot where. A plot command file contains an arbitrary number of lines of the form:

```
plot psfile
```

(`plot` = name of an existing plot definition, `psfile` = name of a PostScript output file written by the program). Note, that several plots may be written to the same PostScript file. If a file with the name `psfile` already exists, the plot is appended to the file.

`lcmdfile` Name of an input text file (list command file) containing the instructions of which plots to list where. A list command file contains an arbitrary number of lines of the form:

```
plot lisfile
```

(`plot` = name of an existing plot definition, `lisfile` = name of a text output file written by the program). Note, that several plots may be listed to the same text file. If a file with the name `lisfile` already exists, the listing is appended to the file.

A7.4 Troubleshooting

As mentioned in the introduction, AQUASIM was primarily designed for internal use for research and teaching at the Swiss Federal Institute for Environmental Science and Technology (EAWAG), Dübendorf, Switzerland. Since the Computer and System Sciences Department of EAWAG, where this program was designed and implemented, is not a commercial software developer, it is not possible to provide technical support concerning use of the program. We recommend the following procedure if you have problems with the program.

1 Reread the section of the manual corresponding to the field of your problem.

2 Perform the tutorial exercise which treats a similar job as you want to perform.

3 If you have problems during a calculation, consult the file `aquasim.log` in the startup directory of the program for further hints.

4 If you really think you have problems because of a program error, prepare a report including the following information

   Name:
   Affiliation:
   Address:
   Phone:
   Fax:
   Email:
   User Interface:
   Program Version:
   Hardware:
   Operating System:
   Window System:

   Submit the data file for which the error occurs and describe exactly, how it is possible to reproduce the error (in case of not systematically reproducible errors, try to indicate critical situations, in which the error often occurs). Please send this information by mail, email or fax (please do not call) to:

   Peter Reichert, EAWAG, Ueberlandstr. 133, CH - 8600 Dübendorf, Switzerland, Fax: +41 1 823 50 28, Email: reichert@eawag.ch.

Suggestions for program improvement are also welcome at the same address, but there is no guarantee that they will be considered in future program versions.

References

Akaike, H. (1973), "Information Theory and an Extension of the Maximum Likelihood Principle", in: Petrov, B.N. and Caski, F., eds., *Proc. 2nd Int. Symp. Information Theory, Supp. to Problems of Control and Information Theory*, pp. 267-281.

Akaike, H. (1974), "A New Look at the Statistical Model Identification", *IEEE Transactions on Automatic Control* 19(6), pp. 716-723.

Akaike, H. (1976), "Canonical Correlation Analysis of Time Series and the use of an Information Criterion", in: Mehra, R.K. and Lainiotis, D.G., eds., *System Identification: Advances and Case Studies*, Mathematics in Science and Engineering Vol. 126, Academic Press, New York, pp. 27-96.

Akaike, H. (1981), "Modern Development of Statistical Methods", in: Eykhoff, P., ed., *Trends and Progress in System Identification*, Pergamon Press, Oxford, pp. 169-184.

Apiki, S. (1994), "Paths to Platform Independence", *Byte* 19(1), pp. 172-178.

Ambrose, R.B., Jr. and Barnwell, T.O., Jr. (1989), "Environmental Software at the U.S. Environmental Protection Agency's Center for Exposure Assessment Modeling", *Environ. Software* 4(2), pp. 76-92.

Baffaut, C. and Delleur, J.W. (1989), "Expert system for calibrating SWMM", *J. Wat. Resourc. Plann. Management* 115(3), pp. 278-298.

Baffaut, C. and Delleur, J.W. (1990), "Calibration of SWMM runoff quality model with expert system", *J. Wat. Resourc. Plann. Management* 116(2), pp. 247-261.

Barnwell, T.O., Jr., Brown, L.C. and Marek, W. (1989), "Application of expert systems technology in water quality modeling", *Wat. Sci. Technol.* 21, pp. 1045-1056.

Beck, M.B. (1983), "Uncertainty, System Identification and the prediction of Water Quality", in: Beck, M.B. and van Straten, G., eds, *Uncertainty and Forecasting of Water Quality*, Springer, Berlin, pp. 3-68.

Beck, M.B. (1987), "Water Quality Modeling: A Review of the Analysis of Uncertainty", *Wat. Resourc. Res.* 23(8), pp. 1393-1442.

Beck, M.B. (1991), "Principles of Modelling", *Wat. Sci. Tech.* 24(6), pp. 1-8.

Bellmann, R. and Aström, K.J. (1970), "On Structural Identifiability", *Math. Biosci.* 7, pp. 329-339.

Bevington, P.R. (1969), *Data reduction and error analysis for the physical sciences*, McGraw-Hill, New York.

Billings, S.A. (1980), "Identification of nonlinear systems - a survey", *Proceedings of IEE* 127(6), part D, pp. 272-285.

Birkes, D. and Dodge, Y. (1993), *Alternative Methods of Regression*, John Wiley & Sons, New York.

Bischoff, K.B. (1961), "A note on boundary conditions for flow reactors", *Chem. Eng. Sci. 16*, pp. 131-133.

Blatter, Ch. (1974), *Analysis I-III*, Heidelberger Taschenbücher, Springer, Berlin.

Bohlin, T. (1991), *Interactive System Identification: Prospects and Pitfalls*, Springer, Berlin.

Brandt, S. (1992), *Datenanalyse: mit statistischen Methoden und Computerprogrammen*, 3rd ed., BI-Wissenschafts-Verlag, Leipzig.

Breitenecker, F., Echer, H. and Bausch-Gall, I. (1993), *Simulation mit ACSL: eine Einführung in die Modellbildung, numerische Methoden und Simulation*, Vieweg, Braunschweig.

Brenan, K.E., Campbell, S.L. and Petzold, L.R. (1989), *Numerical solution of initial-value problems in differential algebraic equations*, North-Holland, New York.

Brown, L.C. (1987), "Uncertainty analysis in water quality modeling using QUAL2E", in: Beck, M.B., ed., *Systems analysis in water quality management*, Pergamon Press, Oxford, pp. 309-319.

Brown, L.C. and Barnwell, T.O., Jr. (1987), *The Enhanced Stream Water Quality Models QUAL2E and QUAL2E-UNCAS: Documentation and User Manual*, Environmental Research Laboratory, Office of Research and Development, U.S. Environmental Protection Agency, Athens, Georgia, Report No. EPA/600/3-87/007.

Broyden, C.G. (1965), "A class of methods for solving nonlinear simultaneous equations", *Math. Comp.* 21, pp. 368-381.

Budd, T. (1991), *An Introduction to Object-Oriented Programming*, Addison Wesley, Reading.

Byrne, G.D. and Hindmarsh, A.C. (1987), "Stiff ODE Solvers: A Review of Current and Coming Attractions", *J. Comput. Phys.* 70, pp. 1-62.

Cale, W.G., Jr., O'Neill, R.V. and Shugart, H.H. (1983), "Development and Application of Desirable Ecological Models", *Ecological Modelling*, 18, pp. 171-186.

Carpenter, S. (1991), "A GUI for all systems", *Byte* 16(6), p. 144.

Carver, M.B., Stewart, G.D., Blair, J.M. and Selander, W.N. (1978), *The FORSIM VI simulation package for the automated solution of arbitrarily defined and partial and/or ordinary differential equation systems*, Report No. AECL-5821, Atomic Energy of Canada Ltd., Chalk River, Ontario, Canada.

Caswell, H. (1976), "The Validation Problem", in: Patten, B.C., ed., *Systems Analysis and Simulation in Ecology*, Volume IV, Academic Press, New Yoork, pp. 313-325.

Chow, V.T. (1959), *Open-channel hydraulics*, McGraw-Hill, New York.

Churchill, M.A., Elmore, H.L. and Buckingham, R.A. (1962), "The Prediction of Stream Reaeration Rates", *Int. J. Air Water Pollution* 6, pp. 467-504.

Cirpka, O., Reichert, P., Wanner, O., Müller, S.R. and Schwarzenbach, R.P. (1993), "Gas Exchange at River Cascades: Field Experiments and Model Calculations", *Environ. Sci. Technol.* 27(10), pp. 2086-2097.

Cobalt Blue (1992), *FOR_C++ - FORTRAN to C++ Translator*, Cobalt Blue, Inc., 875 Old Roswell Road, Suite D-400, Roswell, GA 30076, USA.

Collins, C.D. and Park, R.A. (1989), "Primary Productivity", chapter 15 of: Jorgensen, S.E. and Gromiec, M.J., *Mathematical Submodels in Water Quality Systems*, Elsevier, Amsterdam, pp. 299-330.

Côté, R.G. (1992) "Code on the Move", *Byte* 17(7), pp. 206-224.

Danckwerts, P.V. (1953), "Continuous Flow Systems - Distribution of Residence Times", *Chem. Eng. Sci.* 2, pp 1-.

Debus, O. and Wanner, O. (1992), "Degradation of xylene by a biofilm growing on a gas-permeable membrane", *Wat. Sci. Tech.* 26, pp. 607-616.

Dongarra, J.J., Moler, C.B., Bunch, J.R. and Stewart, G.W. (1979), *LINPACK user's guide*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia.

Ekama, G.A. and Marais, G.v.R. (1978), *The dynamic behabior of the activated sludge process*, Research report no. W27, University of Cape Town, Dept. of Civil Eng. (3 Volumes).

Ellis, M. and Stroustrup, B. (1990), *The annotated C++ reference manual*, Addison-Wesley.

Eykhoff, P. (1974), *System Identification - Parameter and State Estimation*, John Wiley & Sons, London.

Fischer, H.B. (1967), "The mechanics of dispersion in natural streams", *J. Hyd. Div.* ASCE 93(6), pp. 187-216.

Fischer, H.B., List, E.J., Koh, R.C.Y., Imberger, J. and Brooks, N.H. (1979), *Mixing in inland and coastal waters*, Academic Press, New York.

French, R.H. (1985), *Open-channel hydraulics*, McGraw-Hill, New York.

Gameson, A.L.H., "Weirs and the Aeration of Rivers" *J. Inst. Water Eng.* 11, pp. 477-490.

Gard, T.C. (1988), *Introduction to Stochastic Differential Equations*, Marcel Dekker, New York.

Gardner, R.H., O'Neill, R.V., Mankin, J.B. and Carrey, J.H. (1981), "A comparison of sensitivity analysis and error analysis based on a stream ecosystem model", *Ecological Modelling* 12, 173-190.

Gardner, R.H. and O'Neill, R.V. (1983), "Parameter Uncertainty and Model Predictions: A Review of Monte Carlo Results", in: Beck, M.B. and van Straten, G. ed., *Uncertainty and Forecasting of Water Quality*, Springer, Berlin.

Gear, C.W. (1969), "The Automatic Integration of Stiff Ordinary Differential Equations", *Information Processing 68*, Morell, A.J.H., ed., North Holland, Amsterdam, pp. 187-193.

Gear, C.W. (1971a), "The Automatic Integration of Ordinary Differential Equations", *Communications ACM* 14(3), pp. 176-179.

Gear, C.W. (1971b), "Algorithm 407, DIFSUB for Solution of Ordinary Differential Equations", *Communications ACM* 14(3), pp. 185-190.

Gear, C.W. (1971c), *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall, Engelwood Cliffs, N.J..

Gill, P.E., Murray, W. and Wright, M.H. (1981), *Practical optimization*, Academic Press, London.

Godfrey, K.R. and DiStefano, J.J. (1985), "Identifiability of Model Parameters", in: Barker, H.A. and Young, P.C., eds., *Identification and System Parameter Estimation 1985*, Proceedings of the seventh IFAC/IFORS Symposium, Vol. 1, pp. 89-114.

Godfrey, K.R. and DiStefano, J.J. (1987), "Identifiability of Model Parameters", Chapter 1 of Walter, E., ed., *Identifiability of Parametric Models*, Pergamon Press, Oxford, pp. 1-20.

Godfrey, K.R. and Fitch, W.R. (1984), "The deterministic identifiability of nonlinear pharmacokinetic models", *J. Pharmacokinetics Biopharmaceutics* 12(2), pp. 177-191.

Gromiec, M.J., Loucks, D.P. and Orlob, G.T. (1983), "Stream Quality Modeling", in: Orlob, G.T., ed., *Mathematical Modeling of Water Quality*, John Wiley & Sons, Chichester, pp. 176-226.

Gujer, W. and Henze, M. (1991), "Activated Sludge Modelling and Simulation", *Wat. Sci. Tech.* 23, pp. 1011-1023.

Haest, M., Bastin, G., Gevers, M. and Wertz, V. (1988), "An expert workstation for system identification", in: Han-Fu, C., ed., *Identification and System Parameter Estimation 1988*, Selected papers from the eighth IFAC/IFORS symposium, Beijing, 1988, Volume 2, Pergamon Press, Oxford, pp. 1355-1360.

Hampel, F.R. (1973), "Robust Estimation: A Condensed Partial Survey", *Z. Wahrscheinlichkeitstheorie verw. Geb.* 27, pp. 87-104.

Hampel, F.R., Ronchetti, E.M., Rousseeuw, P.J. and Stahel, W.A. (1986), *Robust Statistics - The Approach Based on Influence Functions*, John Wiley & Sons, New York.

Haribson, S.P. and Steele, G.L., Jr. (1991), *C - A Reference Manual*, Prentice Hall, Engelwood Cliffs, Third ed..

Harris, G.P. and Piccinin, B.B. (1977), "Photosynthesis by natural phytoplankton populations", *Arch. Hydrobiol.* 80, pp. 405-456.

Henderson, F.M. (1966), *Open channel flow*, Macmillan, New York.

Henze, M., Grady, C.P.L., Jr., Gujer, W., Marais, G.v.R. and Matsuo, T. (1986), *Activated Sludge Model No. 1*, IAWPRC Task Group on Mathematical Modelling for Design and Operation of Biological Wastewater Treatment, Scientific and Technical Report No. 1, IAWPRC, London.

Hindmarsh, A.C. (1983), "ODEPACK, a Systematized Collection of ODE Solvers", in: Stepleman, R. et al., eds., *Scientific Computing*, IMACS / North-Holland Publishing Company, pp. 55-64.

Holmberg, A. (1982), "On the practical identifiability of microbial growth models incorporating Michaelis-Menten type nonlinearities", *Mathematical Biosciences* 62, pp. 23-43.

Hornberger, G.M. and Cosby, B.J. (1985), "Selection of parameter values in environmental models using sparse data: a case study", *Applied Mathematics and Computation* 17, pp. 335-355.

Hornberger, G.M. and Spear, R.C. (1981), "An approach to the preliminary analysis of environmental systems", *J. Env. Management* 12, pp. 7-18.

Hornberger, G.M. and Spear, R.C. (1983), "An approach to the analysis of behavior and sensitivity in environmental systems", in: Beck, M.B. and van Straten, G., eds., *Uncertainty and Forecasting of Water Quality*, Springer, Berlin, pp. 101-116.

Huber, P.J. (1981), *Robust Statistics*, John Wiley & Sons, New York.

IMSL (1993), *IMSL Libraries*, IMSL Inc., 2500 Permian Tower, 2500 City West Boulevard, Houston, Texas 77042-3020, USA.

Jamshidi, M. and Herget, C.J. (1985), *Computer-Aided Control Systems Engineering*, North-Holland, Amsterdam.

Jannssen, P.H.M., Heuberger, P.S.C. and Sanders, R. (1992), "UNCSAM: A Useful Tool for Sensitivity and Uncertainty Analysis of Mathematical Models", in: Zannetti, P., ed., *Computer Techniques in Environmental Studies IV*, Computational Mechanics Publications, Southampton and Elsevier, London, pp. 335-350.

Jensen, K. and Wirth, N. (1975), *PASCAL - User manual and report*, Springer, New York.

Jørgensen, S.E. (1983), "Modeling the Ecological Processes", in: Orlob, G.T., ed., *Mathematical Modeling of Water Quality*, John Wiley & Sons, Chichester, pp. 116-149.

Jørgensen, S.E. (1992), *Integration of Ecosystem Theories: A Pattern*, Kluwer Academic Publishers, Dordrecht.

Kloeden, P.E. and Platen, E. (1992), *Numerical Solution of Stochastic Differential Equations*, Springer, Berlin.

Kremer, J.N. (1983), "Ecological implications of parameter uncertainty in stochastic simulation", *Ecological Modelling* 18, pp. 187-207.

Langbien, W.B. and Durum, W.H. (1967), "The Aeration Capacity of Streams", *U.S. Geological Survey*, Washington, DC, Circ. 542.

Lecourtier, Y. and Raksanyi, A. (1987), "The Testing of Structural Properties through Symbolic Computation", Chapter 7 of Walter, E., ed., *Identifiability of Parametric Models*, Pergamon Press, Oxford, pp. 75-84.

LeVeque, R.J. (1990), *Numerical solution of conservation laws*, Birkhäuser, Basel.

Lippman, S.B. (1989), *C++ primer*, Addison-Wesley.

Ljung, L. (1987), *System Identification - Theory for the User*, Prentice-Hall, Englewood Cliffs, New Jersey.

Marazzi, A. (1993), *Algorithms, Routines and S Functions for Robust Statistics*, Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, California.

Marquardt, D.W. (1963), "An algorithm for least-squares estimation of nonlinear problems", *J. Soc. Ind. Appl. Math.* 11(2), pp. 431-441.

Marty, R. (1988), *Von der Subroutinentechnik zu Klassenhierarchien - Eine Schrittweise Hinführung zu objektorientierter Programmierung*, Institut für Informatik der Universität Zürich, Nr. 88.04.

MathWorks (1990), *MATLAB*, The MathWorks Inc., Cochituate Place, 24 Prime Park Way, Natick, Massachusetts 01760, USA.

Mauersberger, P. (1983), "General Principles in Deterministic Water Quality Modeling", in: Orlob, G.T., ed., *Mathematical Modeling of Water Quality*, John Wiley, pp. 42-115.

McCarty, P.L. (1972), "Stoichiometry of biological reactions", International Conference "Towards a Unified Concept of Biological Waste Treatment Design", Atlanta, Georgia, October 6, 1972.

McCutcheon, S.C. (1989), *Water Quality Modeling Vol. I: Transport and Surface Exchange in Rivers*, CRC, Boca Raton.

Mehra, R.K. (1980), "Nonlinear system identification: Selected survey and recent trends", in: Isermann, R. ed., *Identification and System Parameter Estimation*, Proceedings of the fifth IFAC Symposium, Darmstadt, 1979, pp. 77-83.

Meier zu Fahrwig, H. and Unbehauen, H. (1992), "Knowledge-based system identification", in: *Identification and System Parameter Estimation 1991*, IFAC Symposia Series, number 3(92), pp. 21-29.

Meyer, B. (1990), *Objektorientierte Softwareentwicklung*, Hanser, München.

Müller, K.H. und Streker, I. (1984), *FORTRAN 77 - Programmieranleitung*, B.I. Hochschultaschenbücher, Mannheim.

NAG (1993), *NAG Numerical Libraries*, NAG Ltd., Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK.

Nelder, J.A. and Mead, R. (1965), "A simplex method for function minimization", *Computer Journal* 7, pp. 308-313.

Norton, J.P. (1986), *An Introduction to Identification*, Academic Press, London.

O'Connor, D.J. and Dobbins, W.E. (1958), "Mechanism of Reaeration in Natural Streams", *Trans. ASCE* 123, pp. 641-884.

O'Neill, R.V. (1979), "Natural Variability as a Source of Error in Model Predictions", in: Innis, G.S. and O'Neill, R.V., eds., *System Analysis of Ecosystems*, International Co-operative Publishing House, Fairland, pp. 23-32.

O'Neill, R.V. and Gardner, R.H. (1979), "Sources of Uncertainty in Ecological Models", in: Zeigler, B.P., Elzas, M.S., Klir, G.J. and Ören, T.I., eds., *Methodology in Systems Modelling and Simulation*, North-Holland, Amsterdam, pp. 447-463.

Owens, M., Edwards, R.W. and Gibbs, J.W. (1964), "Some Reaeration Studies in Streams", *Int. J. Air Water Pollution* 8, pp. 469-486.

Pahl-Wostl, C. and Imboden, D.M. (1990), "DYPHORA - a dynamic model for the rate of photosynthesis of algae", *Journal of Plankton Research* 12(6), pp. 1207-1221.

Pahl-Wostl, C. (1992), "Dynamic versus static models for photosynthesis", *Hydrobiologia* 238, pp. 189-196.

Pearson, J.R.A. (1959), "A note on the Danckwerts boundary conditions for continuous flow reactors", Chem. Eng. Sci. 10, pp. 281-284.

Petersen, E.E. (1965), *Chemical Reaction Analysis*, Prentice-Hall, Englewood-Cliffs, NJ.

Petzold, L. (1982), "Differential/Algebraic Equations are not ODE's", *SIAM J. Sci. Stat. Comput.* 3 (3), pp. 367-384.

Petzold, L. (1983), "A Description of DASSL: A Differential/Algebraic System Solver", in: Stepleman, R. et al., eds., *Scientific Computing*, IMACS / North-Holland Publishing Company, pp. 65-68.

Pohjanpalo, H. (1978), "System identifiability based on power series expansion of the solution", *Mathematical Biosciences* 41, pp. 21-33.

Popper, K. (1982), *Logik der Forschung*, 7th edition, J.C.B. Mohr, Tübingen (first edition 1934).

Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T. (1989), *Numerical Recipes - The Art of Scientific Computing*, Cambridge University Press, Cambridge.

Ralston, M.L. and Jennrich, R.I. (1978), "Dud, a derivative-free algorithm for nonlinear least squares", *Technometrics* 20(1), pp. 7-14.

Reckhow, K.H. and Chapra, S.C. (1983), "Confirmation of Water Quality Models", *Ecological Modelling* 20, pp. 113-133.

Reckhow, K.H., Clements, J.T. and Dodd, R.C. (1990), "Statistical Evaluation of Mechanistic Water-Quality Models", *J. Environ. Eng.* 116(2), pp. 250-268.

Reichert, P. (1992), *Mathematische Modellierung aquatischer Systeme - Teil A: Fliessgewässer*, Skriptum zur Vorlesung 03-426, ETH Zürich, Switzerland.

Reichert, P. (1994a), "AQUASIM - A tool for simulation and data analysis of aquatic systems", *Wat. Sci. Technol.*, in press.

Reichert, P. (1994b), "Implementation Strategy of a Computer Program for the Identification of Processes and the Simulation of Water Quality in Aquatic Systems", to be published.

Reynolds, C.S. (1984), *The Ecology of Freshwater Phytoplankton*, Cambridge University Press, Cambridge.

Rissanen, J. (1976), "Minmax Entropy Estimation of Models for Vector Processes", in: Mehra, R.K. and Lainiotis, D.G., eds., *System Identification: Advances and Case Studies*, Mathematics in Science and Engineering Vol. 126, Academic Press, New York, pp. 97-119.

Rochkind, M.J. (1989-94), *Technical overview of the extensible virtual toolkit (XVT)*, XVT Software Inc., Box 18750, Boulder, CO 80308, USA.

Rodi, W. (1980), *Turbulence Models and their Application in Hydraulics - A State of the Art Report*, International Association for Hydraulic Research (IAHR), Delft.

Roe, P.L. (1985), "Some contributions to the modelling of discontinuous flows", *Lect. Notes Appl. Math.* 22, pp. 163-193.

Rose, M.R. (1987), *Quantitative Ecological Theory - An Introduction to Basic Models*, Croom Helm, London.

Rose, K.A., Smith, E.P., Gardner, R.H., Brenkert, A.L. and Bartell, S.M. (1991), "Parameter Sensitivities, Monte Carlo Filtering and Model Forecasting under Uncertainty", *J. Forecasting* 10, pp. 117-133.

Rosenfeld, A.H., Barbaro-Galtieri, A., Podosky, W.J., Price, L.R., Soding, P., Wohl, C.G., Roos, M. and Willies, W.J. (1967), "Data on Particles and Resonant States", *Rev. Mod. Phys.* 39(1), pp. 1-51.

Rousseeuw, P.J. and Leroy, A.M. (1987), *Robust regression and outlier detection*, John Wiley & Sons, New York.

Rubinstein, R.Y. (1981), *Simulation and the Monte Carlo Method*, John Wiley & Sons, New York.

Sawa, T. (1978), "Information Criteria for Discriminating among Alternative Regression Models", *Econometica* 46(6), pp. 1273-1291.

Schiesser, W.E. (1976), *DSS/2 - An Introduction to the Numerical Solution of Lines Integration of Partial Differential Equations*, Lehigh University, Bethlehem, Pennsylvania, USA.

Schiesser, W.E. (1991), *The Numerical Method of Lines Integration of Partial Differential Equations*, Academic Press, San Diego.

Schiesser, W.E. (1994), *Computational Mathematics in Engineering and Applied Science: ODEs, DAEs and PDEs*, CRC Press, Boca Raton.

Schwarz, G. (1978), "Estimating the Dimension of a Model", *The Annals of Statistics* 6(2), pp. 461-464.

Shamir, U.Y. and Harleman, D.R.F. (1967), "Numerical Solutions for Dispersion in Porous Mediums", *Wat. Resourc. Research* 3(2), pp.557-581.

Singh, M.G., ed. (1987), *Systems & Control Encyclopedia*, 8 volumes + supplements, Pergamon Press, Oxford.

Smith, D.N. (1991), *Concepts of Object-Oriented Programming*, McGraw-Hill, New York.

Söderström, T. and Stoica, P. (1989), *System Identification*, Prentice Hall, New York.

Spriet, J.A. and Vansteenkiste, G.C. (1982), *Computer-aided modelling and simulation*, Academic Press, London.

Spriet, J.A. (1985), "Structure Characterization - An Overview", in: Barker, H.A. and Young, P.C., eds., *Identification and System Parameter Estimation 1985*, Proceedings of the seventh IFAC/IFORS Symposium, Volume 1, pp. 749-756.

St. Venant, M., de (1871), "Théorie du movement non permanent des eaux, avec application aux crues des rivières et à l'introduction des marées dans leur lit", Acad. Sci. (Paris), *Comptes Rendus* 73, pp. 147-154 & 237-240.

Stoer, J. (1983), *Einführung in die numerische Mathematik*, Heidelberger Taschenbücher, Springer, Berlin, 4. ed..

Straskraba, M. and Gnauck, A. (1983), *Aquatische Ökosysteme - Modellierung und Simulation*, Gustav Fischer Verlag, Stuttgart.

Streeter, H.W. and Phelps, E.B. (1925), "A study of the pollution and natural purification of the Ohio River", *US Public Health Service*, Washington, DC, Bulletin 146.

Stull, R.B. (1988), *An Introduction to Boundary Layer Meteorology*, Kluver Academic Publishers, Dordrecht.

Stumm, W. and Morgan, J.J. (1981), *Aquatic Chemistry - An Introduction Emphasizing Chemical Equilibria in Natural Waters*, John Wiley, New York, 2nd ed..

Sweby, P.K. (1984), "High resolution schemes using flux limiters for hyperbolic conservation laws", *SIAM J. Numer. Anal.* 21(5), pp. 995-1011.

Thackston, E.L. and Krenkel, P.A. (1969), "Reaeration Prediction in Natural Streams", *J. San. Eng. Div. ASCE* 95(1), pp. 65-94.

Thomann, R.V. (1982), "Verification of Water Quality Models", *J. Environ. Eng. Div.* ASCE 108(5), pp. 923-940.

Tiwari, J.L. and Hobbie, J.E. (1976a), "Random Differential Equations as Models of Ecosystems: Monte Carlo Simulation Approach", *Mathematical Biosciences* 28, pp. 25-44.

Tiwari, J.L. and Hobbie, J.E. (1976b), "Random Differential Equations as Models of Ecosystems - II. Initial Condition and Parameter Specifications in Terms of Maximum Entropy Distributions", *Mathematical Biosciences* 31, pp. 37-53.

Tiwari, J.L., Hobbie, J.E. and Peterson, B.J. (1978), "Random Differential Equations as Models of Ecosystems - III. Bayesian Inference for Parameters", *Mathematical Biosciences* 38, pp. 247-258.

Tsivoglou, E.C. and Neal, L.A. (1976), "Tracer measurement of reaeration: III. predicting the reaeration capacity of inland streams", *JWPCF* 48(12), pp. 2669-2689.

van Leer, B. (1974), "Towards the ultimate conservative difference scheme II. Monotonicity and conservation combined in a second order scheme", *J. Comput. Phys.* 14, pp. 361-370.

Walter, E. (1982), *Identifiability of State Space Models*, Springer, Berlin.

Walter, E., ed. (1987), *Identifiability of parametric models*, Pergamon Press, Oxford.

Wanner, O. and Gujer, W. (1984), "Competition in Biofilms", *Wat. Sci. Tech.* 17, pp. 27-44.

Wanner, O. and Gujer, W. (1986), "A Multispecies Biofilm Model", *Biotech. Bioeng.* 28, pp. 314-328.

Wanner, O., Debus, O. and Reichert, P. (1994), "Modelling the spatial distribution and dynamics of a Xylene-degrading microbial population in a membrane-bound biofilm", *submitted to Wat. Sci. Technol.*.

Wanner, O. and Reichert, P. (1994), "Mathematical Modeling of Mixed Culture Biofilms", to be published.

Wehner, J.F. and Wilhelm, R.H. (1956), "Boundary conditions of flow reactor", *Chem. Eng. Sci.* 6, pp. 89-93.

Wernstedt, J., Otto, P., Puhlmann, R. and Ross, F. (1992), "DIOPRAN-EXPERT - A consulting/expert system for experimental process analysis", in: *Identification and System Parameter Estimation 1991*, IFAC Symposia Series, number 3(92), pp. 641-646.

Wirth, N. (1985), *Programmieren in Modula-2*, Springer Verlag, Berlin.

Wirth, N. (1986), *Algorithmen und Datenstrukturen mit Modula-2*, Teubner, Stuttgart, 4. Auflage.

Wissel, Ch. (1989), *Theoretische Ökologie - Eine Einführung*, Springer, Berlin.

Yen, B.C. (1973), "Open-channel flow equations revisited", *J. Eng. Mech. Div. ASCE* 99, pp. 979-1009.

Yen, B.C. (1979), "Unsteady flow mathematical modeling techniques", Chapter 13 of Shen, H.W., ed. *Modeling of Rivers*, John Wiley, New York.

Yodzis, P. (1989), *Introduction to Theoretical Ecology*, Harper & Row, New York.

Young, P. (1983), "The validity and credibility of models for badly defined systems", in: Beck, M.B. and van Straten, G., eds., *Uncertainty and Forecasting of Water Quality*, Springer, Berlin, pp. 69-98.

Young, P. (1984), *Recursive Estimation and Time-Series Analysis - An Introduction*, Springer, Berlin.

Zannetti, P., ed. (1992), *Computer Techniques in Environmental Studies IV*, Computational Mechanics Publications, Southampton and Elsevier, London.