Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Manual of spIds, a software package for parameter identification in dynamic systems

C.T.H. Everaars, P.W. Hemker and W. Stortelder

Department of Numerical Mathematics

NM-R9521 1995

Report NM-R9521
ISSN 0169-0388

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

CWI is the National Research Institute for Mathematics and Computer Science. CWI is part of the Stichting Mathematisch Centrum (SMC), the Dutch foundation for promotion of mathematics and computer science and their applications.
SMC is sponsored by the Netherlands Organization for Scientific Research (NWO). CWI is a member of ERCIM, the European Research Consortium for Informatics and Mathematics.

Copyright © Stichting Mathematisch Centrum
P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

Manual of spIds, a Software Package for Parameter Identification in Dynamic Systems

C.T.H. Everaars, P.W. Hemker and W. Stortelder

CWI
P.O. Box 94079, 1090 GB Amsterdam,
The Netherlands
(e-mail: walterst@cwi.nl)

Abstract

This report contains the manual of spIds, version 1.0, a software package for parameter identification in dynamic systems. SpIds is an acronym of simulation and parameter identification in dynamic systems. It can be applied on wide variety of dynamic systems which can be described by a set of ordinary differential equations or differential algebraic equations.
The manual describes briefly the general principles of the underlying mathematics and the structure of the software package. The preparations for the input are described in detail.
The documentation of the Graphical User Interface is also quite explicit.

AMS Subject Classification (1991): 65C20, 93B30, 65L05.

Keywords & Phrases: parameter estimation, dynamic systems, nonlinear system identification, visualisation, computational steering, interaction and direct manipulation.

Note: This report describes part of the work made for the STW project CWI22.2695. This research is partially supported by the Dutch Technology Foundation (STW).

Contents

List of Tables

List of Figures

Preface

The software package "spIds" is a user-friendly system to simulate time-dependent dynamical systems and to estimate unknown parameters in such systems when additional (experimental) data about the system are known. Much of its strength lies in the identification of possible multiple solutions and in the careful statistical analysis of the data and the results.

The software package runs under UNIX and consists of a number of independent processes. These parts communicate with each other by means of a database manager. The different parts of spIds (with the name of the corresponding UNIX process between brackets) are the following:

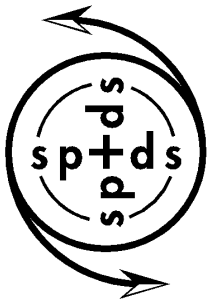
The database manager (dmserver)
This part of spIds takes care for the communication between the different parts of the software package. It is developed by R. v. Liere [5].

The numerical engine (nengine)
This part is responsible for all the numerical work and is developed by W. Stortelder.

The filter (filter)
This part takes care of the communication between the file system and the data manager. It interprets the data file and sends it to the database etc.. It is developed by P.W. Hemker.

The algebraic engine (aengine)
This part interprets the model file and generates the model-dependent parts of the numerical engine. It makes use of the computer algebra system Maple[1]. It is developed by P.W. Hemker.

The Graphical User Interface (GUI)
This Graphical User Interface (or GUI) opens up the possibility to steer the whole spIds application and to visualise the numerical results. It is developed by C.T.H. Everaars.



[1]Maple V Release 3. Copyright (c) 1981-1994 by Waterloo Maple Software and the University of Waterloo. All rights reserved. Maple and Maple V are registered trademarks of Waterloo Maple Software.

Chapter 1

Introduction

The first chapter of this manual gives a short summary of what kind of problems can be solved by means of spIds, how data are prepared and how a model of a process is formulated in such a way that the package can simulate the process and can solve a parameter estimation problem. In Chapter 2 it is explained how the package is started, and in Chapter 3 we describe how the program spIds is used in practice to steer the simulation and the parameter estimation process. The manual describes all (standard) user options of the continuous modelling and parameter estimation program.

In this manual, if the `typewriter font` is used for special words, these words describe notions for which in normal mathematical notation usually a single symbol is used. These words are also used as identifiers in the program, with the same meaning.

1.1 Problem description

The main purpose of the program is to solve a *parameter estimation problem*. I.e., it can be used to validate mathematical models of physical (chemical, biological, biochemical etc.) processes and compute the values of unknown parameters that are used in the description of these processes [1]. Of course, in order to determine such parameters, a model of the process should be available. Also experimental data are needed, and we assume that such data are available. Further we assume that the process can be modelled by a system of ordinary differential equations (*ODEs*) or a system of differential algebraic equations (*DAEs*). In fact, we assume that the process is described by an initial value problem (*IVP*) for a system of differential equations:

$$\begin{aligned} \frac{dy}{dt} &= f(t, y; p), \\ y(t_0; p) &= y_0(p), \end{aligned} \tag{1.1}$$

or, including the algebraic equations, by the system

$$\begin{aligned} \frac{du}{dt} &= f(t, u, v; p), \\ 0 &= g(t, u, v; p), \\ u(t_0; p) &= u_0(p). \end{aligned} \tag{1.2}$$

Here the vector $y(t;p) = \begin{pmatrix} u(t;p) \\ v(t;p) \end{pmatrix}$ represents the variables in the model, which describe the state of the system for $t > t_0$. In the case of the differential algebraic equations the vector $y(t;p)$ combines two vector parts, $u(t;p)$ and $v(t;p)$. For each state variable in the first part, $u(t;p)$, a differential equation is available. For each remaining variable an algebraic equation is given. Of course, all *state variables* in $y(t;p)$ are a function of time, $t > t_0$, and they depend on the (unknown) parameters p. The function $y(t;p)$ is called the *state vector*, as it describes the state of the physical process at time t.

The dimension (i.e. the length) of the vector y is the number of variables in the model, and it is denoted by `noq`. The first part of the vector, $u(t;p)$, corresponding with the variables for which differential equations exist, contains `nodq` elements. The second part, with `noaq` elements, corresponds with the number of algebraic equations. Of course `nodq` + `noaq` = `noq`. The system of ODEs, (1.1), can be seen as a special case of the system of DAEs, with `noaq` = 0.

All unknown parameters are components of the *parameter vector* p. The dimension of the vector p, i.e. the number of unknown parameters, is denoted by `nop`.

To determine a unique solution for the differential equations, an initial vector $u(t_0;p)$ should be known. The program requires to provide a complete initial state $y(t_0;p)$. If algebraic equations are present (`noaq` > 0), this initial state should (approximately) satisfy the conditions determined by these algebraic conditions. The initial state, $y(t_0;p)$, i.e. the state vector at $t = t_0$, may be dependent on the parameter vector p. The number of `noq` initial values (independent initial relations) determines a unique solution of the system of DAEs (ODEs).

Table 1.1: Summary of the symbols in the model

symbol	meaning	dimension
t	time, the independent variable	1
y	the state vector, $y = (u, v)^T$	`noq`
u	the vector of state variables for which a differential equation is given (a part of y)	`nodq`
v	the vector of state variables for which no differential equation is given (a part of y)	`noaq`
p	the vector of unknown parameters	`nop`
c	a vector of known constants	`noc`
f	a vector function of t, y and p, that describes the rate of change of u with respect to t.	`nodq`
g	a vector function of t, y and p, that describes the algebraic relations between the components of y.	`noaq`
y_0	the initial condition of the DAEs (possibly depending on p)	`noq`
r	the (possibly nonlinear) constraints on p	`nosid`

The initial-value problem (1.1) or (1.2) is supposed to give a relevant mathematical description of the process under consideration. The set of equations (1.2), together with possible constraints for the parameters, we call the *model*. Generally, we assume that lower and upper

bounds for the unknown parameters are known, i.e. the parameter vector satisfies:

$$p_{min} \leq p \leq p_{max} \ .$$

Often there are additional *constraints* for the unknown parameters. We denote such (possible non-linear) constraints by a vector inequality

$$r(p) \leq 0 \, . \tag{1.3}$$

By this vector-inequality we denote the element-wise inequality: for a vector $a \leq 0$ means that all elements of a are non-positive. The dimension of the vector $r(p)$ is `nosid`.

For completeness, besides a vector of unknown parameters we introduce a vector of known *constants*. This vector is denoted by c and has dimension `noc`. The reason for the introduction of these extra constants will be explained in Section 1.2.

As mentioned before, besides the model we should have a set of experimental data or measurements. Every new set of data for a model yields another parameter estimation problem. We assume that some (or all) components of the state vector have been measured at certain points in time. We denote the results of these measurements, briefly the *measurements* or *observations*, by:

$$\widehat{y}_i = \widehat{y}_{c_i}(t_i), \qquad i = 1, \ldots, \texttt{nobs}. \tag{1.4}$$

Here `nobs` denotes the number of observations; the subscript i indicates the i-th observation; t_i denotes the time of the i-th measurement[1]; c_i denotes which y-component has been measured at time t_i. In the i-th observation the c_i-th component of y was measured, $1 \leq c_i \leq$ `noq`.

Of course, it is necessary that the number of measurements is not smaller than the number of unknown parameters (`nobs` $\geq$ `nop`). Usually, the number of measurements is much larger. The aim of parameter estimation is to find a parameter vector which gives an optimal fit between the solution of the DAEs and the measurements. We look for an optimal fit in a *weighted least squares* sense. In mathematical notation: we want to determine p for which

$$S(p) = \sum_{i=1}^{\texttt{nobs}} w_i^2 \, \left(y_{c_i}(t_i; p) - \widehat{y}_{c_i}(t_i) \right)^2 \tag{1.5}$$

is minimal. The *weights* w_i can be chosen by the user, based on his knowledge about the experiments. Often, there is a simple recipe to determine w_i. If the error in all observations is assumed to be roughly comparable, $w_i = 1$ is sufficient. How this w_i can be chosen will be explained elsewhere.

The user gets control over the parameter estimation program by means of the *graphical user interface (GUI)*. This means that, by starting-up the program, the user gets some kind of a *dashboard* on the computer screen, and by moving the mouse and clicking the buttons he can steer the actions of the program. The GUI will show the results and it will take care of proper file management, call the necessary numerical routines and show the solution by visualisation on the screen.

Before a numerical experiment can be performed with the program, the user has to supply the model and the measurements. This information should be provided on two files: the *model*

[1] Of course, more measurements are possible at the same time: then $t_i = t_j$ for $i \neq j$, but the sequence t_i should be non-decreasing, i.e. $t_i \leq t_j$ if $i < j$.

file and the *data file*. The model file contains the ODEs or the DAEs, the data file contains the measurements. After we have shown two examples of mathematical models, we shall show in Section 1.2 and 1.3 how the model file and the data file are constructed.

Before we continue we'll give two simple examples of the mathematical formulation of a parameter estimation problem. These examples are only for illustration. By skipping the examples the reader doesn't skip essential information.

1.1.1 Example 1

In this first example we describe one of the simplest possible physical phenomena: a falling object. The corresponding model is described by most elementary dynamics.

Suppose we have an iron ball, with an unknown horizontal velocity, V, which is falling due to gravity. We assume that we can measure the horizontal as well as the vertical position of the ball at several distinct times. From this information we want to retrieve the unknown horizontal velocity as well as the gravity constant, G. We denote the vertical and horizontal position at time t by $S_v(t)$ and $S_h(t)$ respectively. A possible model to describe the position of the ball is the following system of DAEs:

$$\begin{aligned} \frac{dS_v(t)}{dt} &= G * t, && (1.6)\\ S_h(t) &= V * t, && (1.7)\\ S_v(0) &= 0, && (1.8)\\ S_h(0) &= 0. && (1.9) \end{aligned}$$

We want to cast this into the general formulation of a parameter estimation problem:

- The state vector, $y = (u, v)^T$, renders the vertical and the horizontal position, (S_v, S_h), of the ball. In this example both vectors u and v have dimension 1.
- The parameter vector is given by: $p = (G, V)$.
- The differential algebraic equations for this example are written as:

$$\begin{aligned} \frac{du}{dt} &= p_1 * t\ , \\ 0 &= p_2 * t - v\ . \end{aligned}$$

- the corresponding initial condition, $y(t_0)$ in this case, is given at $t_0 = 0$ by:

$$\begin{pmatrix} u(0) \\ v(0) \end{pmatrix} = \begin{pmatrix} S_v(0) \\ S_h(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}\ .$$

- `noq`=2, `nodq`=1, `noaq`=1 and `nop`=2.

In Section 1.2.2 we describe how this information is written on a model file in order to analyse the problem with spIds.

Suppose we have measured both the vertical and the horizontal position of the ball at $t = 1$ and $t = 2$ (thus `nobs`=4), a possibly set of measured values is given below. We denote

these values by (see also equation (1.4)):

$$\begin{aligned}
\widehat{y}_{[1]} = \widehat{y}_1(1) &= S_v(1) = -4.71 \ , \\
\widehat{y}_{[2]} = \widehat{y}_2(1) &= S_h(1) = \ \ 3.02 \ , \\
\widehat{y}_{[3]} = \widehat{y}_1(2) &= S_v(2) = -19.6 \ , \\
\widehat{y}_{[4]} = \widehat{y}_2(2) &= S_h(2) = \ \ 6.48 \ .
\end{aligned}$$

In section 1.3.1 we show how these data can be converted into program input. The goal is now to determine the parameters G and V in such a way that the sum of squared distances between the measured and the computed positions is minimal.

1.1.2 Example 2

The second example describes a simple process from population dynamics. It describes the population density in a two-species population with a predator-prey relation ('foxes' and 'rabbits'). The model is known as the Lotka-Volterra equations.

We consider an area where the population densities of the foxes and the rabbits are the state variables of interest. The development of the densities depends on the initial state as well as on the birthrate of the rabbits and the appetite and death rate of the foxes.

We denote the state vector of densities (rabbits and foxes) by (y_1, y_2). The parameters for birthrate of the rabbits, appetite and death rate of the foxes are denoted by p_1, p_2 and p_3 respectively. The predator-prey model is now described by the subsequent set of differential equations:

$$\frac{dy_1}{dt} = p_1 y_1 - p_2 y_1 y_2 \ , \tag{1.10}$$

$$\frac{dy_2}{dt} = p_2 y_1 y_2 - p_3 y_2 \ . \tag{1.11}$$

The initial condition of this problem is given by the initial state $(y_1(0),\ y_2(0))$. (In Section 1.2.3 we will treat a specific example with $y_1(0) = 1.0$ and $y_2(0) = 0.3$.) For this model we get: `noq`=`nodq`=2, `noaq`=0 and `nop`=3.

A possible data-set is given in Section 1.3.2, it contains both densities, y_1 and y_2, from $t = 0.5$ to $t = 5.0$ with intervals of $\Delta t = 0.5$, in dimensionless time-units. This means `nobs`=20 and

$$\widehat{y}_{[1]} = \widehat{y}_1(0.5), \quad \widehat{y}_{[2]} = \widehat{y}_2(0.5), \quad \widehat{y}_{[3]} = \widehat{y}_1(1.0), \ldots, \widehat{y}_{[20]} = \widehat{y}_2(5.0)$$

are given.

1.2 The model file

The *model file* contains the mathematical description of the process studied. In this section we describe first the precise form in which the model should be given on the model file. The novice user is advised to read the formal description in combination with the examples at the end of this section to obtain a more concrete picture of the model file.

The model file is written in the *MAPLE language* and it will be interpreted by the MAPLE program. This means that the user has disposal of the complete MAPLE language to express

his problem in a mathematical form. However, generally only a very small part of the language is necessary to specify the differential(-algebraic) equations, the initial condition and the few other data that are necessary to formulate the model.

First we specify the contents of the model file, and after that we give two examples of a model file corresponding with the Examples 1 and 2 from Section 1.1.

Some parts of the model file are obligatory and some are optional. Besides the typical lines that are found in the model file, as given in the template in Section 1.2.1, the user is free to use additional MAPLE language to help the mathematical formulation of the problem.

The lines that appear in the model file are used in order to:

1. Define the **list of state variables** as `Variables`. This list corresponds to the names of the components of the state vector, $y(t;p)$. Instead of the variables $y_1, \ldots, y_{noq}$, the user is free to choose names that are more meaningful for the problem at hand.

   The names of the actual variables (e.g. `vari`, $1 \leq$ `i` $\leq$ `noq`) are free for the user to choose. The number of variables, `noq`, is known to the program by the length of the list `Variables`.

2. Define the **list of unknown parameters** as `Parameters`. The names of the parameters (e.g. `parj`, $1 \leq$ `j` $\leq$ `nop`) are free for the user to choose, but they should be different from the variable names. The number of parameters, `nop`, is known to the program by the length of the list `Parameters`.

3. Define optionally a **list of constants** as `Constants`. The length of this list will be identified as `noc`. The list contains the names for constants, introduced by the user, to help indicate quantities that are known and fixed in each experiment(part), but possibly different in other experiments or experiment parts.

   The optional list `Constants` gives the opportunity to identify constants that have a fixed value for one (part of an) experiment, but that may be different (but still fixed) in an other (part of the) experiment.

   The names of the actual variables are free for the user to choose, but they should be different from the parameter and variable names.

   If such constants are introduced in the model file, each constant should be initialised by the user with a (default) value. For each constant (e.g. named `conk`) this is done by assigning a value to `Cdefault[conk]`. In the data file the user will have the opportunity to overwrite these values with different values for particular (parts of) experiments. In Section 1.2.4 we shall see how these constants can be used.

4. Define optionally a **list of constraints** as `SideConditions` that should be satisfied by the parameter values. The length of this list will be identified by `nosid`. The list contains a name for each constraint of the form $r_i(p) \leq 0$, cf. (1.3), that is specified by the user. Besides these additional (possible non-linear) constraints that are specified by the user, we have constraints of the form

   $$p_{\min} \leq p \leq p_{\max} ,$$

   to indicate a feasible box region of the parameters.

   All names introduced in the above lists should be unique names, appearing only once in all four lists.

5. Define the **right-hand sides**, $f(t, y; p)$, of the differential equations in (1.1) or (1.2), by assigning an algebraic expression (depending on all available `Y[vari]`, `P[parj]`, and `C[conk]`) to the array elements `f[varl]`, for $1 \leq \mathtt{l} \leq \mathtt{nodq}$.

6. Define the **algebraic equations**, $g(t, y; p) = 0$, of the DAEs by assigning an algebraic expression (depending on all available `Y[vari]`, `P[parj]`, and `C[conk]`) to the array elements `g[varl]`, for $\mathtt{nodq} + 1 \leq \mathtt{l} \leq \mathtt{noq}$.

7. Define the **initial states**, $y_0(p)$, in (1.1) or (1.2) by assigning an algebraic expression (depending on `P[parj]`, and `C[conk]`) to the array element `YStart[vari]`, for $1 \leq \mathtt{i} \leq \mathtt{noq}$. It is necessary to assign expressions to all possible `YStart[vari]`, for $1 \leq \mathtt{i} \leq \mathtt{noq}$.

   If the user forgets one of the above, required assignments, he will receive an error message. More assignments are optional. In the case that an optional specification is omitted, the program will use a default setting as given in Table 1.2.

8. Determine the `nop`-dimensional rectangle in parameter space, where the unknown parameter vector resides. In the model file lower- and upper-bounds for the parameter values can be given. Therefore arrays `ParMin` and `ParMax` are introduced, for which

$$\mathtt{ParMin[parj]} \leq p_{\mathtt{j}} \leq \mathtt{ParMax[parj]}; \quad \mathtt{j} = 1, \cdots, \mathtt{nop}\,.$$

   If no arrays `ParMin` and `ParMax` are specified, the default values $\mathtt{ParMin[parj]} = 0$ and $\mathtt{ParMax[parj]} = 1$ are assumed, for $\mathtt{j} = 1, \cdots, \mathtt{nop}$.

9. Define the additional **constraints**, that were introduced in `SideConditions`. These additional (possibly nonlinear) constraints in the parameter space are specified by assigning the expressions $r(p)$, as in equation (1.3), to the array of expressions `r[sidl]`, with $\mathtt{l} = 1, \cdots, \mathtt{nosid}$, where the index `sidl` is the name in the list of `l`-th parameter constraint. We call these additional constraints *side conditions*. Such expressions only depend on the unknown parameters `P[parj]` and the known constants `C[conk]`.

10. Indicate the order of magnitude for the components in the state vector, so that

$$|y_{\mathtt{i}}(t)| \leq \mathtt{YSize[vari]}; \quad \mathtt{i} = 1, \cdots, \mathtt{noq}\,.$$

    These `YSize`-values are used for scaling purposes only and play a minor role in the computations. If no `YSize` is specified, its elements are assumed to be equal to 1.0.

1.2.1 A model file template

In Table 1.3 we give a template of the MAPLE-text on the model file. The choice of most names used in the MAPLE text are at the user's discretion, except for the reserved words `Variables`, `Parameters`, `Constants`, `Y`, `P`, `f`, `g`, `r`, `YStart`, `YSize`, `ParMin`, `ParMax`. [2] See for the reserved words Table 1.2.

Here, `var1, vari, varnoq, par1, parj, parnop, con1, conk, connoc, sid1, sidl` and `sidnosid`, are names that can be selected by the user; `RHSexpressionj`, `ALGexpressioni`

[2] Bug: at the moment the following names should not be used either: `ee, ez, es, ytrans, rr, rn, DF`.

Table 1.2: Summary of reserved names and default values in the model file

Reserved name	Assignment	Default value	Type in MAPLE
`Variables`	yes		list [a]
`Parameters`	yes		list
`Constants`	optional		list
`SideConditions`	optional		list
`t`	no		name
`Y`	no		table [b]
`P`	no		table
`C`	no		table
`f`	yes		table [c]
`g`	for DAEs		table
`r`	optional	`r[sidl]=-1.0`,[d] $1 \leq$ `l` $\leq$ `nosid`	table [e]
`YStart`	yes		table [d]
`YSize`	optional	`YSize[vari]=1.0`, $1 \leq$ `i` $\leq$ `noq`	table [f]
`ParMin`	optional	`ParMin[parj]=0.0`, $1 \leq$ `j` $\leq$ `nop`	table
`ParMax`	optional	`ParMax[parj]=1.0`, $1 \leq$ `j` $\leq$ `nop`	table
`Cdefault`	optional	`Cdefault[conk]=0.0`, $1 \leq$ `k` $\leq$ `noc`	table

[a] list of names

[b] table of variables

[c] table of expressions, depending on `Y, P` and `C`

[d] Can still give rise to an error in case an element of the list `SideConditions` is omitted here

[e] table of expressions, depending on `P` and `C`

[f] table of floating point numbers

are algebraic expressions depending on the independent variable `t`, the dependent variables `Y[vari]`, the parameters `P[parj]`, and the constants `C[conk]` (with `i=1, ...,noq, j=1, ...,nop, k=1, ...,noc`). The `RHSexpressionj` corresponds with $f_j(t, y; p)$, ($j = 1, ..., $ `nodq`), and describes the right hand side of the j-th differential equation; the i-th algebraic equation, $g_i(t, y; p)$, ($i = 1, ...,$ `noaq`), is represented by `ALGexpressioni`. For all $i = 1, ...,$ `noq`, `INITexpressioni` corresponds with the initial condition, $y_i(t_0; p)$, of the i-th component of the differential-algebraic equations, and it may depend on `P[parj]` and `C[conk]`. The assignments to `Cdefault`, `Ysize`, `ParMin` and `ParMax` are expressions for numerical values (floating or fixed point numbers).

Extra model constraints with respect to the unknown parameters, besides $p_{\min} \leq p \leq p_{\max}$, can be added at the end of the model file. These side conditions, which are al-

Table 1.3: The template of a model file

```
Variables:=[var1,vari,varnoq];
Parameters:=[par1,parj,parnop];
Constants:=[con1,conk,connoc];
SideConditions:=[sid1,sidl,nosid];
Cdefault[con1]:= constant1;
Cdefault[conk]:= constantk;
Cdefault[connoc]:= constantnoc;
f[var1]:= RHSexpression1;
g[vari]:= ALGexpressioni;
f[varj]:= RHSexpressionj;
g[noq]:= ALGexpressionnoq;
YStart[var1]:= INITexpression1;
YStart[vari]:= INITexpressioni;
YStart[varnoq]:= INITexpressionnoq;
YSize[var1]:= ysize1;
YSize[vari]:= ysizei;
YSize[varnoq]:= ysizenoq;
ParMin[par1]:= parmin1;
ParMin[parj]:= parminj;
ParMin[parnop]:= parminnop;
ParMax[par1]:= parmax1;
ParMax[parj]:= parmaxj;
ParMax[parnop]:= parmaxnop;
r[sid1]:= SIDEexpression1
r[sidl]:= SIDEexpressionl
r[sidnosid]:= SIDEexpressionnosid;
```

lowed to be non-linear, are supplied in the form `r[sidl]:= SIDEexpressionl` (with `l=1, ...,nosid`). Here, `SIDEexpressionl` is an algebraic expression, depending on the unknown parameters `P[parj]` and the known constants `C[conk]`, representing the expression $r_l(p)$, the l-th component in equation (1.3). The number of side conditions (`nosid`) corresponds with the dimension of $r(p)$. We assume, as we do for `INITexpressioni`, that `SIDEexpressionl` is (MAPLE-) differentiable with respect to p. For `RHSexpressionj` and `ALGexpressioni` we assume (MAPLE-) differentiability with respect to p and to y.

1.2.2 Model file for example 1

In this section we show the model file that corresponds with the example of the falling ball of Section 1.1.1. Comment lines in the model file start with '`#`'.

```
#introduction of the symbol names used
Variables :=[Sv,Sh];
Parameters:=[G,V];
```

```
#specification of the differential equation
f[Sv]:=P[G]*t;
#specification of the algebraic equation
g[Sh]:=P[V]*t-Y[Sh];

#initial values
YStart[Sv]:= 0;
YStart[Sh]:= 0;

#parameter bounds for G
ParMin[G]:=0;
ParMax[G]:=25;

#parameter bounds for V
ParMin[V]:=-10;
ParMax[V]:=15;

#a guess for the upper bounds of the state variables
YSize[Sv]:= 20;
YSize[Sh]:= 10;
```

1.2.3 Model file for example 2

The second example model file corresponds with the Lotka-Volterra model, as described in Section 1.1.2. In particular we refer to the differential equations (1.10) and (1.11) for the model. For the differential equations in the model file, we can simply choose the names: `BirthRate, Appetite, DeathRate, rabbit` and `fox`, instead of the symbols p_1, p_2, p_3, y_1 and y_2. So, the model file reads:

```
#introduction of the symbol names used
Variables :=[rabbit,fox];
Parameters:=[BirthRate,Appetite,DeathRate];

#specification of the differential equations
f[rabbit]:=P[BirthRate]*Y[rabbit]-P[Appetite]*Y[rabbit]*Y[fox];
f[fox]   :=P[Appetite]*Y[rabbit]*Y[fox]-P[DeathRate]*Y[fox];

#the initial conditions
YStart[rabbit]:=1.0;
YStart[fox]   :=0.3;

#bounds for the parameter space
ParMin[BirthRate]:= 0.0;
ParMin[Appetite] := 0.0;
ParMin[DeathRate]:= 0.0;
ParMax[BirthRate]:= 2.0;
```

```
ParMax[Appetite] := 2.0;
ParMax[DeathRate]:= 2.6;

#guess for the size of the state variables
YSize[fox]   :=1.0;
YSize[rabbit]:=1.0;
```

1.2.4 Example of a more extended model file

The setup of the model file gives many more possibilities than those we showed in the previous two examples. Therefore, in this section, we present an extension of example 1 to give an (artificial) example which shows more possibilities for a model file. E.g. we introduce *known constants* and a sidecondition for the unknown parameters.

To obtain this we add:

- an extra unknown parameter, `Sv0`, which represents the unknown initial position in the vertical direction.
- a known constant, `Sh0`, which is the known initial position in the horizontal direction. For this horizontal position we usually have `Sh0`=3, however in some experiments its value may be different.
- a known constant, `Vv0`, corresponds with the initial speed in the vertical direction, which may take different values in different experiments.
- a known constant, `M`, the mass of the ball.
- a known constant, `MaxKin3`, which is an upper bound for the kinetic energy at $t = 3.0$.
- a side condition, `KinEner3`, which prevents the kinetic energy at $t = 3.0$ from exceeding the upper bound `MaxKin3` by making a proper choice of the unknown parameters. The horizontal speed is equal to `P[V]` for all t and the vertical speed at $t = 3.0$ is $3 * \texttt{P[G]} + \texttt{C[Vv0]}$. The kinetic energy at $t = 3.0$ is given by half the mass times the squared speed at $t = 3.0$:
$$\tfrac{1}{2}\texttt{C[M]} * ((3 * \texttt{P[G]} + \texttt{C[Vv0]})^2 + (\texttt{P[V]})^2) \ .$$

The extended model file becomes:

```
#introduction of the symbol names used
Variables :=[Sv,Sh];
Parameters:=[G,V,Sv0];
Constants :=[Sh0,Vv0,M,MaxKin3];
SideConditions:=[KinEner3];

#setting of the default values for the known constants
Cdefault[Sh0]:=1.0;
Cdefault[Vv0]:=3.0;
Cdefault[M]:=2.0;
Cdefault[MaxKin3]:=800;
```

```
#introduction of the DAEs
f[Sv]:=-P[G]*t+C[Vv0];
g[Sh]:=P[V]*t-Y[Sh]+C[Sh0];

#the initial conditions (depending on an unknown parameter
#or a known constant)
YStart[Sv]:= P[Sv0];
YStart[Sh]:= C[Sh0];

#bounds for the unknown parameters
ParMin[G]  :=0;
ParMin[V]  :=-10;
ParMin[Sv0]:=-5;
ParMax[G]  :=25;
ParMax[V]  :=15;
ParMax[Sv0]:=5;

#an additional restriction for the unknown parameters
r[KinEner3]:= 0.5*C[M]*((3*P[G]+C[Vv0])**2+P[V]**2)-C[MaxKin3];

#a guess for the upper bounds of the state variables
YSize[Sv]:= 20;
YSize[Sh]:= 10;
```

1.3 The data file

The less experienced user is advised to read this formal description of the data file in combination with the examples in Section 1.3.1, 1.3.2 and 1.3.3.

The data file contains the measured values (the *observations*) obtained from the process studied. From equation (1.4) we see that the measured value $\widehat{y}_{c_i}(t_i)$ is related with the point of time t_i and the component c_i of the state vector, $1 \leq c_i \leq$ `noq`. All information about a single measured value should be on the same line in the data file. So the *data part* of the simplest data file consists of `nobs` lines, with on each line three numbers: t_i, c_i and $\widehat{y}_i$. The numbers t_i and $\widehat{y}_i$ are floating point numbers. The number c_i is an integer that corresponds with the c_i-th variable in the list `Variables`. This number c_i can also be replaced by the symbolic name that appears in the list `Variables`. The lines corresponding with a single experiment should appear in the order of increasing (more precisely: non-decreasing) t_i.

A single experiment

In the simplest possible data file, the *data part* is preceded by two lines: (1) a line containing some identification of this data set: an arbitrary string of at most 24 characters, and (2) a line containing only the word START and the value t_0. This obligatory line denotes that the initial value problem should start for $t = t_0$. The data part is closed by a single line, containing the word STOP and the value for t_{end}, the time at which the initial value problem ends.

If the user wants to provide a weight w_i for the weighted sum of squares (1.5), he can do this by adding the real number w_i as the 4-th number on the line for the i-th observation. If no weight is specified, this has the same effect as $w_i = 1.0$.

In case the user wants to skip a measurement, he can inactivate the measurement by putting a 0 as the 5-th number at the end of the corresponding line. We give this number the name `active`. The default setting is 1, which means that the measurement is *active*, i.e. is taken into account in the computation.

Multiple experiments

Another important option is to take several experiments into account for the same model and the same parameters, but possibly with different values of the model constants as given in `Constants`. In this manual we use the word *experiment* for a sequence of observations (measurements) ordered in time. In case of a parameter estimation problem with a series of experiments the user should provide a series of *data parts*, each of which is preceded by a line containing the value t_0 (to denote that a new initial value problem is considered, starting at $t = t_0$). In order to specify what values for the constants are used, the restart line can be immediately preceded by a number of lines which contain the word `CONSTANT`, the constants name, and the constants value.

In this way, a data file can contain measurements from many different experiments corresponding to the same model file. If some constants change from one experiment to the other, the corresponding measurements have to be separated by a constant block. [3]

It is also possible to change the model constants at distinct values t with $t_0 < t < t_{\text{end}}$. At such times, t_{cont}, a discontinuity in the process of the experiment occurs and the change of constants is specified in the data file. So, each experiment may consist of different, distinct parts in time, where the constants have fixed values. Such periods during experiments are called *experiment parts*.

At the beginning of every experiment the constants are set equal to their default values from the model file and adaptation will be performed after every constant line in the datafile.

Data file syntax

Summarising we find the following syntax for the information on the data file.

DATA_FILE:	identification_line ; EXPERIMENT_BLOCK
EXPERIMENT_BLOCK:	EXPERIMENT [; EXPERIMENT_BLOCK]
EXPERIMENT:	START_PART [; CONTINUATION] ; stop_line
CONTINUATION:	CONTINUATION_PART [; CONTINUATION]
START_PART:	start_line [; CONST_PART] ; DATA_PART
CONTINUATION_PART:	continue_line [; CONST_PART] ; DATA_PART
CONST_PART:	constants_line [; CONST_PART]
DATA_PART:	data_line [; DATA_PART]
identification_line:	`DATASET` , data-set-name
start_line:	`START` , t_0 [, experiment-name]
continue_line:	`CONTINUE` , t_{cont}

[3] It is important to note that by measurement we mean one single measured value and by experiment part we refer to a set of measurements which are performed under the same conditions.

stop_line:	`STOP` , t_{end}
constants_line:	`CONSTANT` , `conk` , b_j
data_line:	t_i , c_i , $\widehat{y}_i$ [, w_i [, 0 | 1 [, 0 | 1]]]
comment_line:	#, a sequence of characters ending with carriage return

In this syntax description, ';' means 'followed on the next line by',
',' means 'followed on the same line by', '[ ]' means 'optional',
and '|' means 'or'.

t_i, $\widehat{y}_i$, w_i and b_j are floating point numbers;
c_i, is a natural number;
c_i can be replaced by `vari` from `Variables`;
`conk` is an element from `Constants` in the model.

data-set name and experiment name are sequences of at most 24 characters. The first binary flag (0|1) on the data line denotes that the observation is active; the second flag denotes whether the observation is a *break-point* All data lines in an experiment block, following a "START ; t_0 ; name"-line, should be ordered in time: $i < j \Rightarrow t_i \leq t_j$. Also the possible t_{cont} should satisfy this ordering. Any such sequence beginning with a t_0 is called an *experimental sequence* and can be identified by a name.

1.3.1 Data file of Example 1

In this first example we give about the simplest possible data file, which corresponds with the assumed measurements on page 5, with `nobs`=4. Instead of the variable names in the second column of each data line, `Sv` and `Sh`, also a number 1 or 2 can be used (i.e. its number in the list [`Sv, Sh`]). (Again, lines starting with '#' are comment lines.)

```
# identification line
DATASET test_ball1
#
# starting line of the first experiment; exp1 starts at t=0.0
START  0.0 exp1
#
# data lines
# t_i     c_i        obs_i
1.0       Sv         -4.71
1.0       Sh          3.02
2.0       Sv         -19.6
2.0       Sh          6.48
#
# stop line; exp1 ends at t=3.0
STOP   3.0
```

1.3.2 Data file for Example 2

In this data file for Example 2, compared to the previous data file for Example 1, we start adding a *weight* and a value (0 or 1) for the variable that may (in)activate a data line. For each data line the time, the component and the measured value (observation) are indispensable,

the numbers on the fourth and fifth position are optional. Note that the several entries on a line are separated by spaces: the exact position on the line is irrelevant.

The numbers in the fourth column correspond to the weight of that particular measurement. In case this number is omitted the weight of the measurement on that line is set equal to the default value 1.0. By taking larger weights, as we do in this example for the `rabbit`, we assume these data to be more accurate. The weights, w_i in formula (1.5), indicate the relative importance of that particular measurement.

The binary flag in the fifth column shows if the corresponding measurement is used for the parameter estimation problem. In case of 1, which is also the default value, the measurement is active. If the fifth column contains 0, the corresponding measurement is neglected. For the parameter estimation problem this has the same effect as deleting this line from the data file.

In case the fourth and fifth column of a data line are empty, the corresponding measurement is active and has weight 1.0. If we want to inactivate this data line we can do this by putting a value in the fourth column (this has to be done because, otherwise no distinction between the fourth and the fifth column can be made) and 0 in the fifth or just by setting the weight equal to zero. In the latter case the number of observations, `nobs`, is larger. If `active` is set to zero the value of `nobs` is 1 less. If measurements have to be inactivated we prefer to take 0 for the corresponding values of `active`, because the value of `nobs` is the number of measurements which is really of interest for the parameter estimation problem. (It has its influence in the computation of the statistics.) Although we have 20 observations in the subsequent data set, `nobs` = 17 because `active` is 0 on three data lines. Moreover only 16 observations influence the weighted sum of squares (equation (1.5)), as the weight on the sixth data line is zero.

```
# identification line
DATASET Barnes_problem
#
# starting line; the experiment starts at t=0.0
START   0.0
#
# data lines
# t_i    c_i        obs_i        w_i     active_i
0.5     rabbit      1.1          2.0
0.5     fox         0.35         1.0
1.0     rabbit      1.3          2.0       0
1.0     fox         0.4          1.0       0
1.5     rabbit      1.1          2.0       0
1.5     fox         0.5          0.0
2.0     rabbit      0.9          2.0
2.0     fox         0.5          1.0
2.5     rabbit      0.7          2.0
2.5     fox         0.4          1.0
3.0     rabbit      0.5          2.0
3.0     fox         0.3          1.0
3.5     rabbit      0.6          2.0
3.5     fox         0.25
4.0     rabbit      0.7          2.0
```

```
4.0     fox         0.25
4.5     rabbit      0.8          2.0
4.5     fox         0.3
5.0     rabbit      1.0          2.0
5.0     fox         0.35
#
# stopline; the experiment stops at t=5.0.
STOP  5.0
```

1.3.3 Example of a more extended data file

In this section we will show all the possibilities of setting up a more complex data file. We show a data file that contains the data from two experiments and the second experiment contains two parts with different constants.

At the start of an experiment, a named constant that appears in the list `Constants` takes the value that is specified in the `Cdefault` list of the model file. This value can be overruled by a value specified in a constants_line of the data file. This value holds only for that experiment (in the data file) in which it appears till the value is overruled again.

The initial conditions for the second experiment are different from the initial conditions for the first experiment with respect to initial horizontal position and the initial vertical speed. During the second experiment, at $t = 3$, the speed of the ball in the vertical direction is changed instantaneously. In case of an instantaneous change of one or more of the known constants during an experiment, a new *experiment part* is introduced.

```
# identification line
DATASET test_ball2
#
# starting line; experiment1 starts at t=0.0
START  0.0 experiment1
#
# constant lines
CONSTANT   Sh0   0.0
CONSTANT   Vv0   0.0
#
# data lines
# t_i     c_i        obs_i    w_i   active_i
0.5       Sv         -1.22    0.5
0.5       Sh          1.51    0.6
1.0       Sv         -4.71    0.8
1.0       Sh          3.02    0.9
1.5       Sv         -11.1    1.3
1.5       Sh          4.54
2.0       Sv         -19.6
2.0       Sh          6.48
2.5       Sv         -30.5    1.8
2.5       Sh          8.11    1.0     0
3.0       Sv         -44.3    1.9
```

```
3.0       Sh          9.51
3.5       Sv         -59.9    1.4
3.5       Sh          11.2
4.0       Sv         -78.6    1.6
4.0       Sh          12.7
4.5       Sv         -99.2    1.4
4.5       Sh          14.5
5.0       Sv         -122     1.7
5.0       Sh          16.2
#
# stop line of the first experiment; experiment1 stops at t=5.0
STOP   5.0
#
# starting line of the second experiment; experiment2 starts at t=0.0
START  0.0 experiment2
#
# constant lines
CONSTANT   Sh0   2.0
CONSTANT   Vv0   1.0
#
# data lines
# t_i     c_i        obs_i    w_i   active_i
1.0       Sv         -5.83    0.9     1
1.0       Sh          3.48    2.5     1
2.0       Sv         -20.72   1.1     0
2.0       Sh          8.56    2.8     1
#
# continuation line; proceed from (experiment2) part 1 to part 2 at t=3.0
CONTINUE  3.0
#
# constant line
CONSTANT   Vv0   2.0
#
# data lines
# t_i     c_i        obs_i    w_i   active_i
4.5       Sh          16.4     5.0    1
4.5       Sv         -111.4    5.0
#
# stop line of the second experiment; experiment2 stops at t=4.8
STOP   4.8
```

Chapter 2

Starting spIds

2.1 Starting the program

To start the program, we should have both a model file and a data file available. The program is started in a UNIX-environment by typing the line:

```
spIds dmserverMACH filterMACH nengineMACH ModelFile DataFile      (2.1)
```

dmserverMACH filterMACH and nengineMACH are the names of the machines on which we want to run the server of the data base manager, the filter and numerical engine (nengine) respectively. The GUI always runs on the local machine.

For a description of the model and data file we refer to Section 1.2 and 1.3 respectively. The model-file and the data-file in line (2.1) are only initial settings. With the GUI we can switch to another problem by selecting another model and/or data file. As a result of (2.1) the following takes place:

- A numerical engine is generated corresponding with the model and data file.
- The database manager, numerical engine, filter and GUI are coming up in separate windows. Give them a place on the screen.

Now the different processes start to communicate. After some seconds everything is silent in the windows and the word is with the user. At that moment we can start to investigate our problem. We can steer the numerical engine, have a look into the solution space to see graphs of the solution components and the measurements or have a look into the parameter space and follow the path of a local search, see the quality of parameter estimates, and study the confidence region of a solution found.

2.2 The structure of spIds

The global structure of the program package spIds together with the files that contain the information which is exchanged between the modules, is illustrated in Figure B.1 in Appendix B.

Chapter 3

The graphical user interface (GUI) of spIds

In Section 3.1 we give a general description of the widgets we use in the GUI (widgets are objects that appear on the screen, such as buttons, places in which we can type information, etc.). In Section 3.2 we describe the options in the main window of the spIds. When such an option leads to another window we devote a separate section to it. An overview of all the options in the GUI of spIds is given in appendix C.

3.1 The widgets used in the GUI

In this section we will give an overview of the widgets used in the GUI of spIds.

Text widget

Text widgets are rectangles which are used to show us information from the application. It is a static widget which means that it takes no information from the user.

Input field widget

The input field is a widget which can be used to give textual input to the application. We can edit the input field by using the keyboard. The GUI can take only float and integer input fields, so that we can only type this kind of information into it. If we try to type in something else it is not shown and a bell will ring. Whenever we press the mouse inside an input field a cursor will appear there and the colour of the input field change. We have the following edit possibilities:

- The <Backspace> key characters erases in front of the cursor.
- The <Delete> key erases characters behind the cursor.
- The <Esc> erases the whole input field.
- The left and right arrows move the cursor in the input field. It is also possible to place the cursor at a different position in the input field by using the mouse.

- We can also use the mouse to select a part of the input field which will be removed when we type the <Backspace> key or be replaced by whatever we type.

When we press the <Tab> or <Return> the input field is returned to the application program and input is directed to the next input field (if there is one in the window).

Button widget

We can push buttons with the mouse. Buttons of the type we use have the property that they return to its normal position when we release the mouse. On the screen they look as rectangles which come out of the background. Most buttons in the GUI are grey and when we point to a button with the mouse, its colour will change into light grey. When we push the button, it will change colour, go into the background and come back from it immediately.

Slider widget

This widget is useful for letting us indicate a value between some bounds. A slider has a minimum, maximum and current value which are all floats. The current value is changed by moving the mouse inside the slider area. For finer control, hold down the left or right <shift> key while moving the slider. Whenever the value of a slider is changed, this is reported to the GUI.

Menu widget

Menus can be used to choose from different options. Each menu widget has a box with a label. Whenever we press the mouse — with the left or the right mouse button — inside the box the pop-up menu will appear. Then we can make a selection from this menu. This selection has to be made with the right mouse button. (Therefore the easiest way to make a selection in a menu is to use the right mouse button for both getting the pop-up menu and making the selection.) When we click outside the menu box, no selection is made.

To use the menus, act as follows:

1. Press the right mouse button over the pull-down menu. We see a list of menu entries.
2. While holding the mouse button down, move the cursor up and down over the entries in the menu. When we move the cursor individual entries will be highlighted.
3. Highlight one of the choices; then release the mouse button to activate a command.

Choice widget

A choice widget is an object that allows us to choose among a number of possible choices. The current choice is shown in a box. We can either cycle through the list of choices using the left or middle mouse button or get the list as a menu using the right mouse button. The choice widget is a kind of menu widget that shows its choice.

Browser widget

A browser widget is a box that contains a number of lines of text. We use browser widgets in the GUI for showing information. If the whole text does not fit into the box, automatically a scroll bar is added so that we can scroll through the information.

Colour-map widget

In a colour-map widget we can select a colour. Figure 3.1 shows a colour-map widget. In that figure we can see the first 64 entries. we can scroll through the colour-map to see more entries. We select a colour by pressing the left mouse button on some entry. After selection the widget is automatically removed from the screen We can also decide not to change the colour by pressing the cancel button.

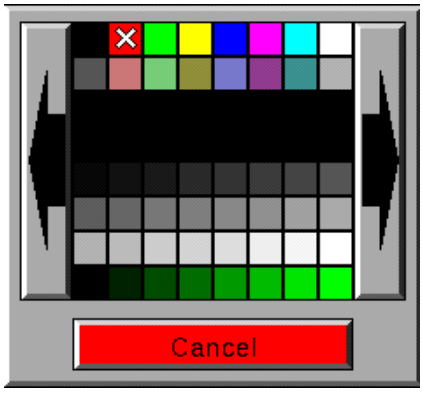


Figure 3.1: The colour-map widget

File selector widget

A file selector widget provides an easy way to select a file. Figure 3.2 shows a file selector.

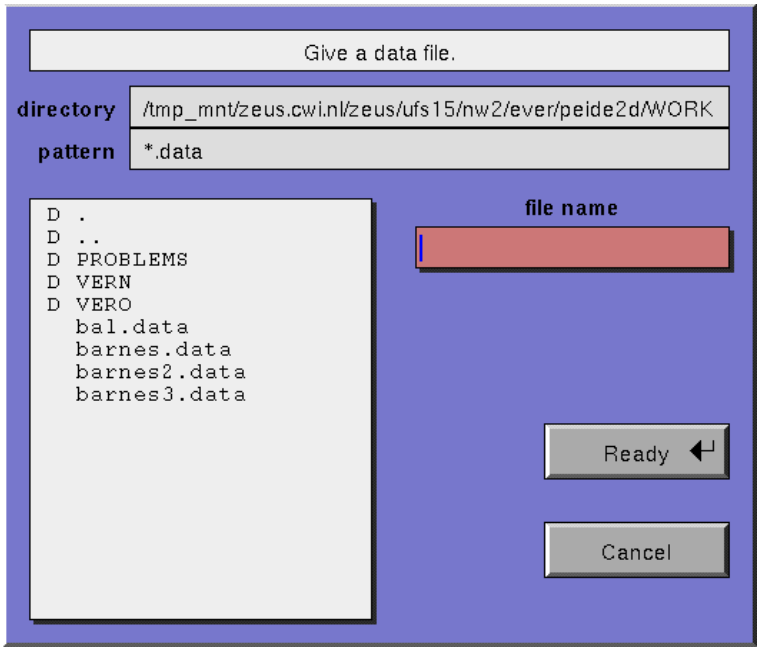


Figure 3.2: The file selector widget

In the window of the file selector all the files in a directory are listed that satisfy a certain

pattern. A pattern can be any kind of regular expression, e.g. `[a-f]*.data`, which gives all files starting with a letter from a to f and ending with .data. In the top of the window we see a message as "Give a $\cdots$ file". We can choose by mouse a file from the list given or type a file name directly. We can also walk through the directory structure, either by changing the directory string by pressing the mouse on it, or by pressing the mouse on a directory name (shown with a D in front of it). It is also possible to change the pattern by pushing the mouse on it. Note that directories are shown independent of whether they satisfy the pattern. When we are satisfied, i.e. found the correct directory and indicated the file name required, we can press the button labelled "Ready" or press the <Return> key. We can also double click on the file name in the browser. If we press the "cancel" button no file is selected and the file selector disappears

3.2 Main window of the GUI

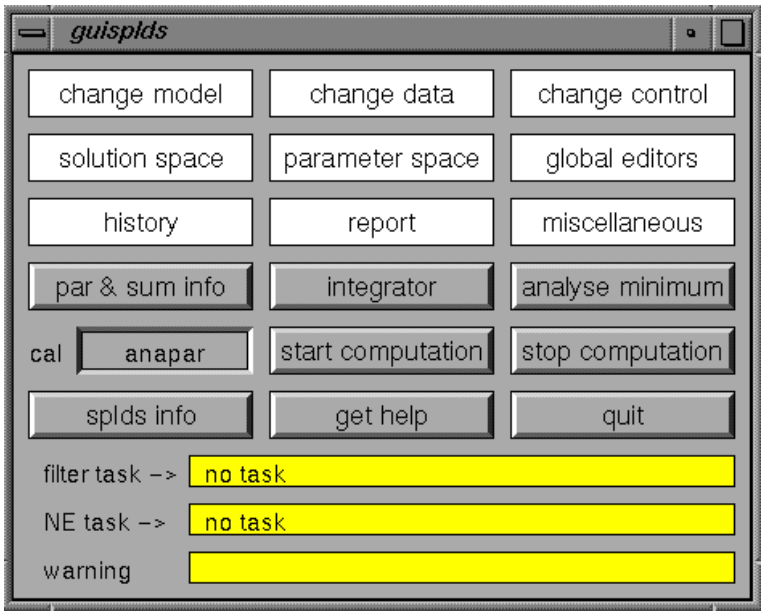


Figure 3.3: The main window of the GUI

Before we give a summary of the options in the main window (see Figure 3.3), we will explain some general principles and terminology.

For spIds we use the words "active" and "activate" to express respectively that something (this can be a file, a variable or a parameter) is or becomes the current value known to the application (i.e. to the data base manager). So when we activate a model file, this file contains the model that is studied, and the active parameter vector is the one that is currently used for the computation (see also the description of the ok button below).

When we speak about the y and p vector we mean the vector of state variables (the solution vector) and the vector of unknown parameters respectively.

Windows, which communicate with the data base manager, have "cancel" "ok" and "done" buttons. These buttons can raise a blush (they turn pink; see also the description of the ok button below). Windows without communication with the data base manager only have

quit buttons. To prevent describing them each time in the sections devoted to the different windows, we describe them here. These buttons perform the following actions:

quit With this button a window that has no communication facilities with the data base manager can be closed.

cancel With this button a window, containing communication facilities with the data base manager, can be closed, without sending the data shown to the data base manager.

ok This button closes the window and sends the data shown in the window to the database manager. We say that we *activate* the data or that the data become *active*. When we adapt data shown in some window, the adapted data are only known to the GUI and not known by the data base manager. To put extra emphasis on the different values of data in the GUI and in the database manager, the buttons "cancel" "ok" and "done" raise a blush when we adapt data in a window. After pressing the "ok" button this blush disappears and the data shown in the window is sent to the data base manager (becomes active). As a result of this activation, windows can be updated.

done This button performs an "ok" action followed by a "cancel" action.

3.2.1 "change model" menu

With the "change model" menu we can modify the model or switch to another problem, by selecting or changing a model file and activate it. To be sure that the current active data are consistent with the model, after activating a model file (offering the modified or new model to the application) we also have to activate a data file (first select one).

In the "change model" menu we can make a selection from the following options.

select file With this option a file selector comes up, which enables us to select a model file.

edit file With this option we start up an editor (at this moment we use the "vi" editor) on the selected model file and modify it.

activate file With this option we make the selected model file active, which means that the model file is read and the problem described in this file will be the one studied. Notice that activating a model file automatically changes the program used to solve the differential equations etc., however activating the model file does not automatically check the consistency with the current data. To make this check, the current data have to be activated again.

We can only activate a file when we first select one. When we forget it, we get a message in a separate, dialogue window. The message disappears when we press the "ok" button in this window.

3.2.2 "change data" menu

In the "change data" menu we can switch to a new data set by selecting or changing a data file and make it active.

In this menu we can make a selection from the following options.

select file With this option a file selector comes up, which enables us to select a data file.

edit file With this option we start up an editor (at this moment we use the "vi" editor) on the selected data file and modify it.

activate file With this option the selected data file is read and the corresponding data are activated, which means that these data are used for the analysis.

We can only activate a file when we first select one. When we forget it, we get a message in a separate window. The message disappears when we press the "ok" button in this window.

edit current data With this option the "Edit current data" window comes up (see Section 3.3). In this window we see the active data, which can be edited interactively (do not forget to send the adapted data to the database manager by the "ok" or "done" button (see Section 3.2)). In this way we can change the active data without changing the original data file.

save current data With this option the file "datf" is created which contains the active data.

3.2.3 "change control" menu

In the "change control" menu we can change the control parameters by selecting another or changing the current control file and make it active. The control file contains all technical parameters that control the numerical process. At default a convenient set of control parameters is active, but we can change it, if this is beneficial for the computations in the numerical process.

In the "change control" menu we can make a selection from the following options.

select file With this option a file selector comes up, which enables us to select an existing control file.

edit file With this option we start up an editor (at this moment we use the "vi" editor) on the selected control file and modify it.

activate file With this option the selected control file is read and activated, which means that the control parameters from this file are used for the subsequent computations.

We can only activate a file when we first select one. When we forget it, we get a message in a separate window. The message disappears when we press the "ok" button in this window.

edit current control With this option the "Edit current control" window comes up (see Section 3.4). In this window we see the active control parameters. We can edit these control parameters interactively. Do not forget to send the adapted data to the database manager (make them active, see Section 3.2). In this way we can change the active control parameters without changing the control file.

save current control With this option the file "ctrlfile" is created which contains the active control parameters.

3.2.4 "solution space" menu

In the "solution space" menu we can make a selection from the following options.

create solution window This option opens the "Create solution window" window (see Section 3.5). In this window we specify what kind of "Solution ···" window (see Section 3.6) we want to make.

merge solution window Not yet implemented.

3.2.5 "parameter space" menu

In the "parameter space" menu we can make a selection from the following options.

create parameter window This option opens the "Create parameter window" window (see Section 3.8). In this window we specify what kind of "Parameter ···" (see Section 3.9) window we want to make.

3.2.6 "global editors" menu

In the "global editors" menu we can make a selection from the following options.

global edit solution windows This option opens the "Global edit solution windows" window (see Section 3.11).

global edit parameter windows This option opens the "Global edit parameter windows" window (see Section 3.12).

3.2.7 "history" menu

This menu is still under construction

In the "history" menu we can make a selection from the following options.

save Not yet implemented.

save & quit Not yet implemented.

load Not yet implemented.

3.2.8 "report" menu

This menu is still under construction.

3.2.9 "miscellaneous" menu

In the "miscellaneous" menu we can make a selection from the following options.

draw grid in solution windows This option draws a green 10 by 10 grid in all the "Solution ···" windows (see Section 3.6).

draw grid in parameter windows This option draws a green 10 by 10 grid in all the "Parameter ···" windows (see Section 3.9).

MCsim With this option the "MCsim" window comes up (see Section 3.13).

3.2.10 "comp" choice

At the moment spIds supports four types of computations. After selection the computation is started by pressing the "start computation" button.

anapar In this mode, spIds analyses a given parameter vector; a simulation of the model equations with active parameter vector will be performed. The parameter analysed will be the "current" or "active" parameter vector. This parameter vector can be set by the left mouse in a "Parameter $\cdots$" window (see Section 3.9) or in the "Parameter & sum info" window (see Section 3.14). The result of a computation "anapar" is:

- The numerical solution of the system of DAEs for the active parameter vector. This can be shown for every particular experiment as a graph in the solution windows of the GUI (see Section 3.10)
- The computation of the discrepancies with the measurements.
- The computation of the weighted sum of squared discrepancies[1] (see the text widget ActSos in Section 3.14).

locmin In this mode, spIds calculates a local minimum of the weighted sum of squares by a Levenberg-Marquardt (LM) method starting from the "current" or "active" parameter vector as the initial estimate. During this iterative method the vector p and its corresponding sum of squares will change, but the parameter vector stays within the active parameter space (see Section 3.14). These changes are reported in the "Parameter $\cdots$" windows (see Section 3.9) and the "Parameter & sum info" window (see Section 3.14). The initial parameter vector can be set in the same way as by the "anamin" option.

glomin In this mode, spIds searches for a global minimum in a rectangular sub-domain of the parameter space. This sub-domain is specified with the `ParMin` and `ParMax` vector in the "Parameter & sum info" window (see Section 3.14). The algorithm for this computation is still under construction

MCsim In this mode spIds performs a Monte Carlo simulation. For every simulation the original measurements are perturbed with noise from a Gaussian distribution. This perturbation is followed by a local minimisation to compute the optimal parameter vector corresponding to the perturbed data. The optimal parameter values for every simulation are plot in the parameter windows. The input quantities for a Monte Carlo simulation can be set in "MCsim" window (see Section 3.13).

3.2.11 "info" button

The "info" button opens the "Info" window (see Section 3.19) which shows some information about spIds. This option is still under construction.

3.2.12 "help" button

The "help" button opens the "Help" window (see Section 3.20) which shows us the help facility of spIds. This option is still under construction.

[1] Henceforth we will shortly speak about the sum of squares.

3.2.13 Other main window options

We give a summary of the remaining widgets in the main window of the GUI.

start computation This button starts the computation selected in the "comp" choice (see Section 3.2.10).

stop computation This button stops the present computation in the numerical engine. The button is red when a stop flag is on. When the numerical engine actually stops the computation the button becomes grey.

par & sum info This button opens the "Parameter & sum info" window (see Section 3.14).

analyse minimum This button performs an analysis of the minimum found in a local minimisation process (an "anamin" computation) and opens the "Analyse minimum" window (see Section 3.16).

integrator The "integrator" button opens the "Integrator" window (see Section 3.19).

filter task In this text widget the activity of the filter is reported.

NE task In this text widget the activity of the numerical engine is reported.

warning In this text widget possible warnings are shown.

quit This button stops all the processes of the spIds application.

3.3 "Edit current data" window

In the "Edit current data" window we can edit the current data in a limited way; it is not possible to add new observations, but an existing observation can be made inactive. This latter action is equivalent to removing an observation from the corresponding data file. Figure 3.4 shows the "Edit current data" window.

We give a summary of the widgets in this window.

time In this float input field we can change the time t_i of the measurement. Notice the requirement that the sequence t_i should be non-decreasing for every experiment (see Section 1.3).

component In this choice widget we can adapt the component, c_i, of the state vector corresponding to the measurement (see eq. (1.4)).

measurement In this float input field we can change the measured value ($\widehat{y}_i$ in eq. (1.4)).

weight With this float input field we specify the weights, w_i, of the sum of squares (see eq. (1.5)). An observation with a zero weight does no longer contribute. The weight is proportional to the accuracy of the corresponding measurement.

active With this choice widget we can make an observation active or inactive. The default value is active. An inactivated observation does no longer contribute to the sum of squares. Contrary to a zero weight, an inactive measurement doesn't contribute to the number of measurements or observations (`nobs`), which is used for the statistical

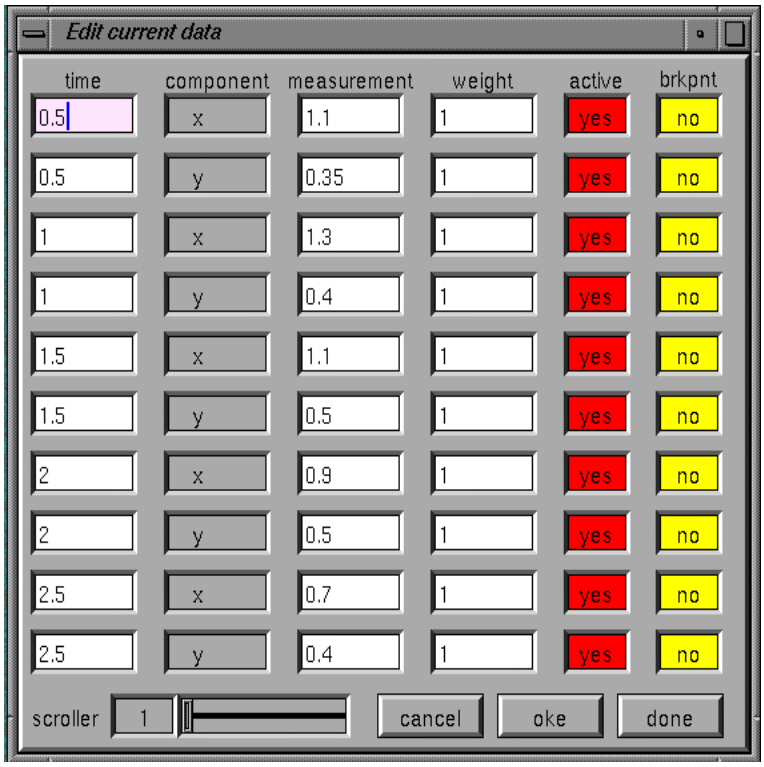


Figure 3.4: The "Edit current data" window

analyses. For most situations an inactive measurement is preferred to a zero weight. The inactive observations are drawn in the "Solution ···" windows with yellow markers. (Notice that we now use the word "active" in a different way than in "The active model ···" (see Section 3.2). We do not think that this is confusing).

brkpnt With this choice widget we add or remove breakpoints. It is not used at the moment.

scroller With this slider we can scroll the observations. Only a maximum of ten observations are shown at the same time.

3.4 "Edit current control" window

In the "Edit current control" window we can adapt the control parameters of the numerical process to our own wishes. spIds starts with a default setting of these parameters. Figure 3.5 shows the "Edit current control" window.

We give a summary of the widgets.

SqSabt Not in use now.

SqSret Not in use now.

IVPtol With this real input field we can adapt the relative, local error bound for the numerical solver of the initial value problem (IVP).

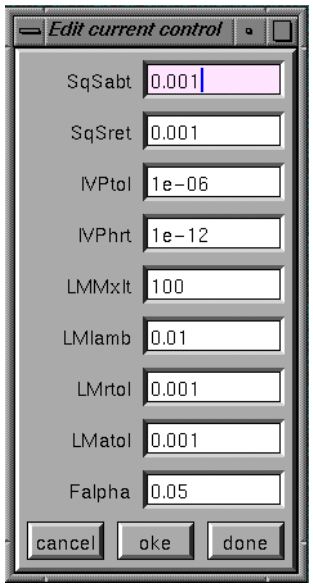


Figure 3.5: The "Edit current control" window

IVPhrt With this real input field we can adapt the allowed minimal ratio of the smallest and the largest step-length during the numerical integration of the IVP.

LMMxIt With this integer input field we can adapt the maximum number of iterations of the Levenberg-Marquardt process.

LMlamb With this real input field we can adapt the initial setting of the parameter λ in the Levenberg-Marquardt process (for details see [2], page 7).

LMrtol With this real input field we can adapt the relative tolerance of the Levenberg-Marquardt process.

LMatol With this real input field we can adapt the absolute tolerance of the Levenberg-Marquardt process.

Falpha With this real input field we can adapt the α-point of the Fishers distribution (for details see [2], page 8).

3.5 "Create solution window" window

The visualisation of the observations and the numerical solution of IVP is done in the "Solution ..." windows (see Section 3.6). In the "Create solution window" window we specify the quantities (time or a component of the solution vector) along the axes of the graphs for these visualisations. Figure 3.6 shows the "Create solution window" window.

We give a summary of the widgets.

exp This choice denotes to which experiment number the graph will refer.

x-axis This choice denotes the solution component displayed along the horizontal or x-axis of the graph.

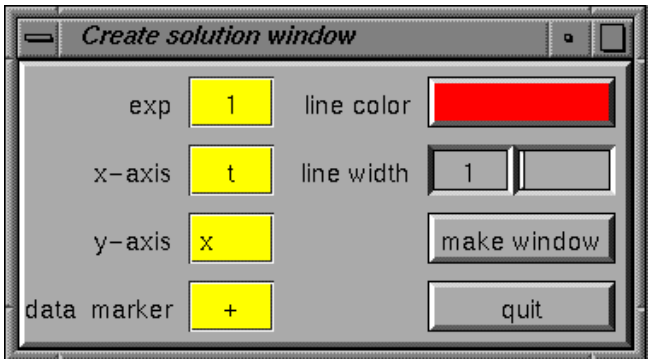


Figure 3.6: The "Create solution ···" window

y-axis This choice denotes the solution component displayed along the vertical or y-axis of the graph.

data marker This choice selects the marker for the observations.

line colour With this button a colour selector comes up to select the colour of a graph.

line width With this slider we can select a line width for the graph.

make window With this button a "Solution ···" window comes up (see Section 3.6).

3.6 "Solution ···" window

A "Solution ···" window gives a two dimensional graph, along the axes we find the quantities from the previous ("Create solution window", section 3.5) window. Figure 3.7 shows a "Solution ···" window.

We give a summary of the widgets.

exp This text widget denotes the experiment number to which the graph refers.

xmin With this slider we can adjust the minimum on the x-axis.

xmax With this slider we can adjust the maximum on the x-axis.

ymin With this slider we can adjust the minimum on the y-axis.

ymax With this slider we can adjust the maximum on the y-axis.

menu In this menu we can perform some additional actions in the solution windows namely:

edit With this option an "Edit solution window ···" window comes up (see Section 3.7).

print Not in use. Use an UNIX utility for making a bitmap of the screen.

quit The window is closed.

The text widgets at the bottom and the left side of the graph denote the quantity along the x-axis and y-axis respectively.

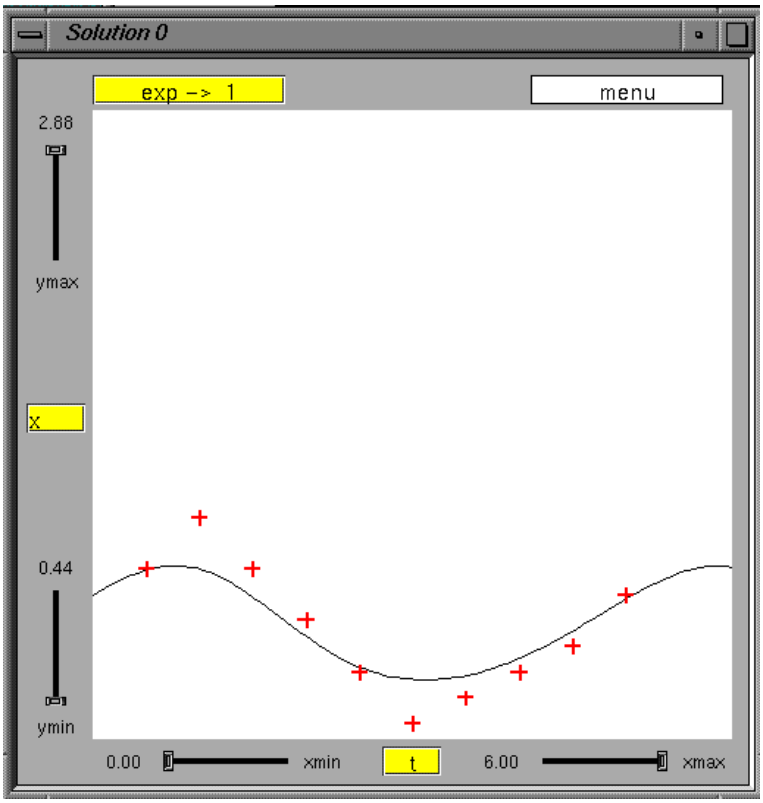


Figure 3.7: A "Solution ···" window

By the left mouse we can inquire a position in a "Solution ···" windows. When we press the left mouse button in this window a text widget will come up, containing the position in the solution space. When we release the left mouse button the text disappears.

In the banner of this window we see an index number. When we start up an editor in this window (see the option in the menu "menu") this leads to an "Edit solution window ···" window. This window has the same index as the corresponding "Solution ···" window. In this way we can see which editor belongs to which solution window.

3.7 "Edit solution window ···" window

In an "Edit solution window ···" window (see Figure 3.8) we can change some settings of the corresponding "Solution ···" window. For the meaning of the index see Section 3.6.

The widgets in this window have the same meaning as those in the "Create solution window" window (see Section 3.5). However we have an extra widget in this window.

show y **win** This button is only visible when we close the "Solution ···" window with the same index as the "Edit solution window ···" window. With this button the "Solution ···" window reappears. We need this button when we merge pictures (see Section 3.2.4).

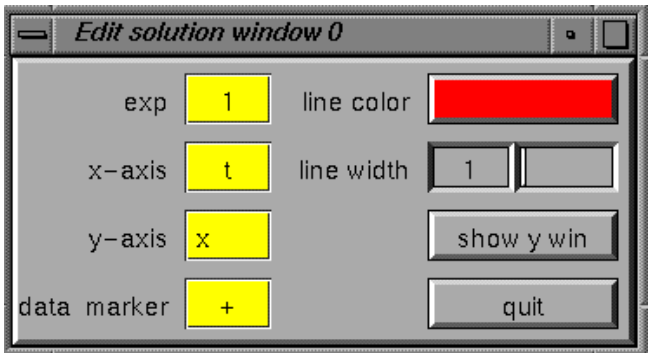


Figure 3.8: An "Edit solution window ···" window

3.8 "Create parameter window" window

In the "Create parameter window" window (see Figure 3.9) we can specify which two-dimensional subspace of the parameter space we want to see in the "Parameter ···" windows (see Section 3.9).

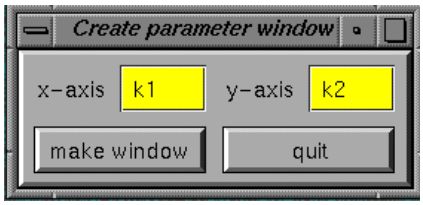


Figure 3.9: The "Create parameter window" window

We give a summary of the widgets.

x-axis This choice denotes the parameter component which is on the x-axis of the graph.

y-axis This choice denotes the parameter component which is on the y-axis of the graph.

make window With this button a "Parameter ···" window comes up (see Section 3.9).

3.9 "Parameter ···" window

In a "Parameter ···" window (see Figure 3.10) we observe a two-dimensional subspace of the parameter space. It is also possible to change the current parameter vector by mouse and start a "locmin", "steplocmin" or "anapar" computation (see Section 3.2.10). We can also investigate the character around the final parameter estimates, follow the path of a local search and study the confidence region of a final estimate of the parameter vector. Further it is possible to put the sum of squares along one of the axis (as a function of a parameter component).

In this window we can make a selection from the following options.

xmin With this slider we can adjust the minimum on the x-axis.

xmax With this slider we can adjust the maximum on the x-axis.

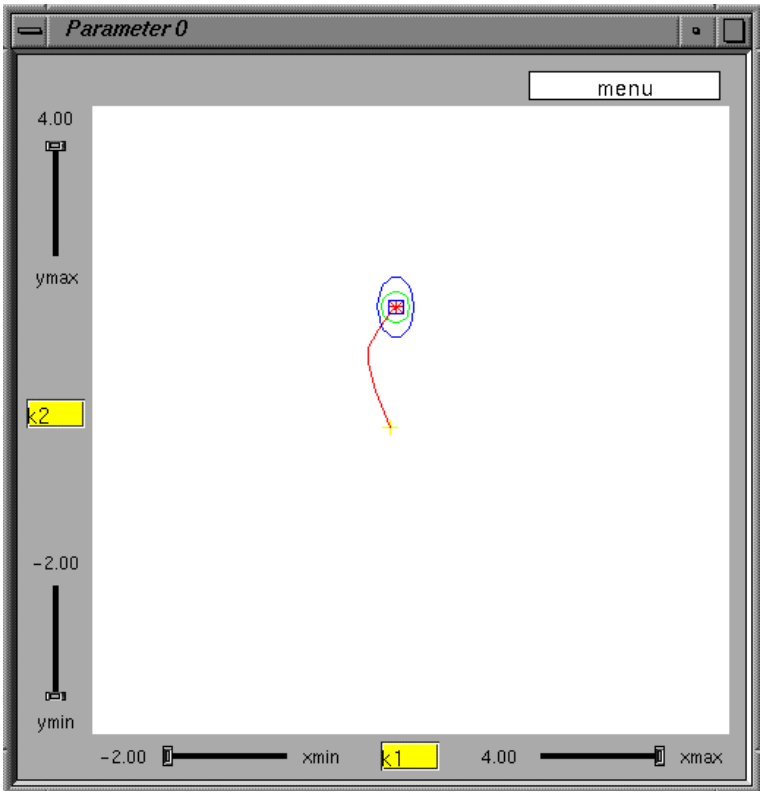


Figure 3.10: A "Parameter ···" window

ymin With this slider we can adjust the minimum on the y-axis.

ymax With this slider we can adjust the maximum on the y-axis.

menu With this menu we can perform some additional actions in the "Parameter ···" windows. We will enumerate them.

edit With this option the corresponding "Edit parameter window ···" window comes up (see Section 3.10).

print Not in use. Use an UNIX utility for making a bitmap of the screen.

quit The window is closed.

The text widgets at the left side and bottom of the window denote the component of the parameter vector or the sum of squares along the x-axis and y-axis respectively.

In a "Parameter ···" window the mouse is used in the following way:

- When we press the *left mouse button* in a "Parameter ···" window a text widget will come up in which we can read the position in the parameter space. When we release the mouse this text disappears and we see a marker "+" moving to the position of the cursor. This position represents the values of the components of the current parameter vector. In the "Parameter & sum info" window (see Section 3.14) we see the corresponding change in the "IniPar" text widget. The "cancel" "ok" and "done" raise a blush to indicate that the change `IniPar` is not known by the database manager. By pressing

the *right mouse button* the `IniPar` is send to the database manager and the blushing of the buttons will stop. This can also be done by pressing the "ok" or "done" button in the "Parameter & sum info" window (see Section 3.14).

- When the computation type is "locmin" (see Section 3.2.10) we can start a local minimisation by pressing the *right mouse button*. The widget "NE task" (see Section 3.2.13) indicating the task of the numerical engine, shows the text "locmin". For the other computation types the right mouse button does not work.
- When the computation type is "locmin" (see Section 3.2.10) we can also use the *middle mouse button* for doing a single step of a local minimisation. We call this computation type a "steplocmin". A "steplocmin" is a special case of a "locmin" computation. In the widget "NE task" (see Section 3.2.13) the text "steplocmin" appears. For the other computation types the middle mouse button performs no task.
- It is also possible to delete a graphical object. This can be a point, a path of a local search or a confidence interval. With the key "d" pressed and pointing to the object with the *left mouse button* we can delete an object in a "Parameter $\cdots$" window (see Section 3.9). Notice that the corresponding objects in the other "Parameter $\cdots$" windows are also deleted.

In the banner of every "Parameter $\cdots$" window we see an index number. When we start up an editor in this window (see the option in the menu "menu") this leads to an "Edit parameter window $\cdots$" window with the same index as the "Parameter $\cdots$" window we started from. In this way we can see which editor belongs to which "Solution $\cdots$" window.

3.10 "Edit parameter window $\cdots$" window

In an "Edit parameter window $\cdots$" window (see Figure 3.11) we can change some settings of the corresponding "Parameter $\cdots$" window. For the meaning of the index see Section 3.9.

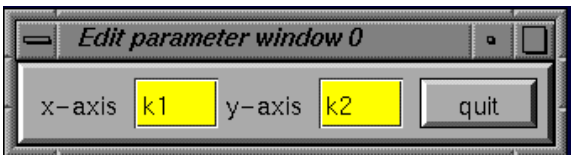


Figure 3.11: An "Edit parameter window $\cdots$" window

The meaning of the widgets in this window is the same as in the "Create parameter window" window see Section 3.9.

3.11 "Global edit solution windows" window

In the window "Global edit solution windows" we can adjust the minimum and maximum values of the solution components. For drawing the solution components in the solution windows we need the minimum and maximum values of these components. When, for example, a "locmin" computation is performed, we get sequences of sets of calculated solution vectors. In these sets we have per experiment, all the calculated solution vectors at different t values.

To determine per experiment the minimum and maximum values of the components, we use the first set received from the numerical engine (via the database manager). The length of the intervals (maximum - minimum values) is enlarged with 20%. When we are not satisfied with the minimum and maximum values for some experiment we can change them in this window. Figure 3.12 shows a "Global edit solution windows" window.

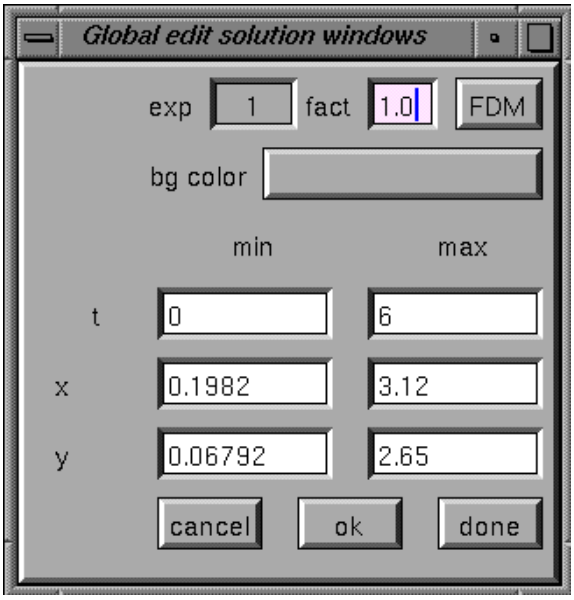


Figure 3.12: The "Global edit solution windows" window

We give a summary of the widgets.

exp This choice widget denotes the experiment number for which we want to adjust minimum and maximum values.

fact This float input field denotes the factor by which the length of an interval, formed by the minimum and maximum of a solution component, is enlarged.

FDM With this button we retrieve the last computed set of solution vectors from the data base (FDM = From Data base Manager).

bg colour With this button a colour selector comes up to select a colour for the background of all the "Solution $\cdots$" windows.

bounds for t In this two float input field we can specify the minimum and maximum for t.

bounds for y In these $2 \star noq$ float input fields we can specify the minimum and maximum for the solution components.

3.12 "Global edit parameter windows" window

In the "Global edit parameter windows" window we can change the colour and width of the lines drawn in the "Parameter $\cdots$" windows. We can also delete the points drawn after an

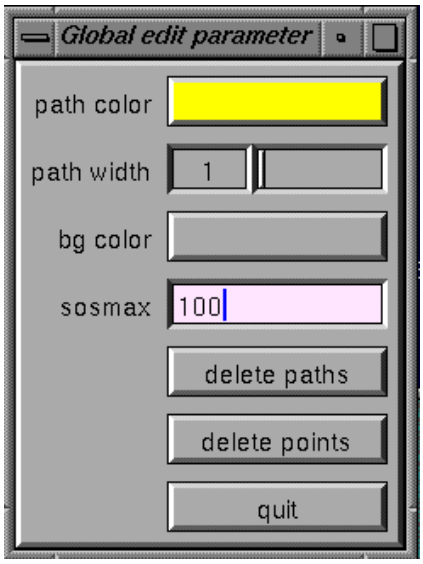


Figure 3.13: The "Global edit parameter windows" window

"anapar" and the paths drawn during a "locmin" (see Section 3.2.10). Figure 3.13 shows the "Global edit parameter windows" window.

We give a summary of the widgets.

path colour With this button a colour selector comes up to select a colour for drawing the path of a local search in the "Parameter $\cdots$" windows.

path width With this slider we can change the line width of the path of the local search.

bg colour With this button a colour selector comes up, in which we can select a colour for the background of the "Parameter $\cdots$" windows.

sosmax With this real input field we can change the upper bound of the sum of squares we used for drawing the "Parameter $\cdots$" windows.

In a "Parameter $\cdots$" window we can set one of the components of the p vector or the sum of squares along an axis. For the drawing in the "Parameter $\cdots$" windows we need the upper and lower bounds of the variables along the axes. We use respectively `ParMin` and `ParMax` (see Section 3.14) for the default values of the upper and lower bounds of the parameter components. For the upper bound of the sum of squares we use the real input field "sosmax". The lower bounds is fixed on zero. After adapting the "sosmax" input field all the "Parameter $\cdots$" windows are redrawn with the new bounds. This will also happen when we adapt the `ParMin` and `ParMax` in the "Parameter & sum info" window (see 3.14).

delete points With this button we can delete the points drawn after an "anapar" computation.

delete paths With this button we can delete the paths drawn after a "locmin" computation.

There is also a possibility to delete globally (that means for all "Parameter $\cdots$" windows together) a graphical object (see Section 3.9).

3.13 "MCsim" window

In the "MCsim" window we can specify a Monte Carlo Simulation. Figure 3.14 shows the "MCsim" window.

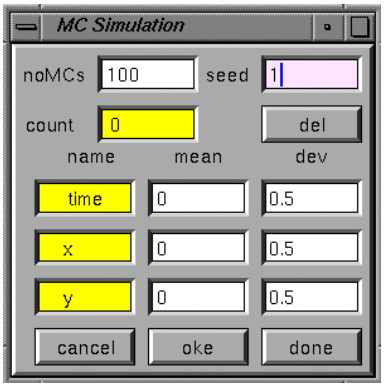


Figure 3.14: The "MCsim" window

We give a summary of the widgets.

noMCs This integer input field denotes the number of the Monte Carlo simulations we want to perform.

seed This integer input field denotes the seed of the random number generator.

count In this text widget the number of Monte Carlo simulations is counted.

del With this button the memory necessary for storing the Monte Carlo simulation is given free.

name In these $noq + 1$ text widgets we see respectively the time and the $noq + 1$ names of the solution components

mean In these noq float input fields we can specify the mean of the Gaussian distribution, from which the perturbation of the time or specified solution component will be taken.

dev In these $noq + 1$ float input fields we can specify the deviation of the Gaussian distribution, from which the perturbation of the time or specified solution component will be taken.

3.14 "Parameter & sum info" window

In the "Parameter & sum info" window we can set our initial guess of the parameter vector. This current parameter vector can act as the starting point of a local minimisation. During the computation of a local minimum we can follow the path of the changing parameter and its corresponding sum of squares. Figure 3.15 shows the "Parameter & sum info" window.

We give a summary of the widgets.

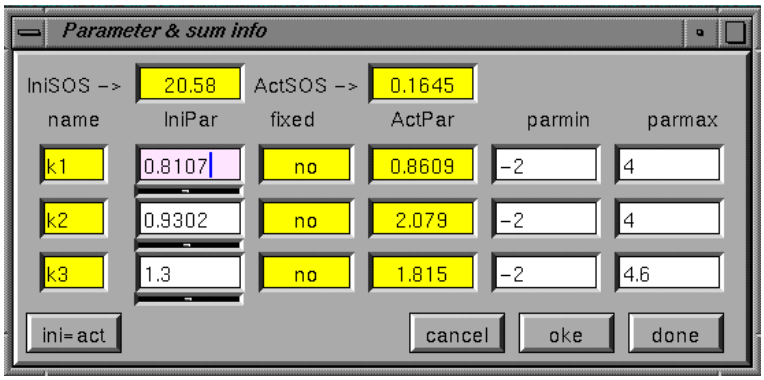


Figure 3.15: The "Parameter & sum info" window

IniSOS This text widget contains the initial sum of squares corresponding to the initial parameters. Before any calculation has been done, we see a question mark in this field. Having started the minimisation process it receives a value after the first iteration in the process. During the minimisation process this value will not change any more.

ActSOS This text widget shows us the current value of the sum of squares during the minimisation process (the Active Sum Of Squares).

name In these *nop* text widgets we see the names of the unknown parameters.

IniPar In these *nop* real input field we give the initial value of the parameter vector. These values can also be adapted by using the sliders under the real input fields. The initial values of the unknown parameters should be in the rectangular parameter domain ([`ParMin, ParMax`]).

fixed In these *nop* choice widgets we indicate which components of the parameter vector are constant during the minimisation process. These fixed parameters are treated as a known constant.

ActPar These *nop* text widgets show the parameter vector during the present (or most recent) calculation of the numerical engine. During an "anapar" and the first iteration of a "locmin" `ActPar` and `IniPar` are equal. `ActPar` is always inside the rectangular parameter domain (denoted by `ParMin` and `ParMax`).

ParMin In these *nop* float input fields we give the lower bound of the current parameter domain.

ParMax In these *nop* float input fields we give the upper bound of the current parameter domain.

Ini=act With this button we assign `ActPar` to `IniPar`.

done After pressing the done button the default action of done is performed (see Section 3.2) followed by a re-drawing of the "Parameter $\cdots$" windows when `ParMin` or `ParMax` has been changed or an update of `ParIni`.

ok Note the remark by the previous option.

3.15 "Integrator" window

During the minimisation process we can obtain information about the actions of the numerical engine. In each step of the minimisation process the set of DAEs has to be integrated at least once. These computations can be time consuming and may give reason to adapt the control parameters of the numerical engine on the fly (see Section 3.4). For every step of the numerical integration some variables, which give inside information about the actions of the IVP solver, are reported. These variables are shown in the "Integrator" window. Figure 3.16 shows the "Integrator" window.

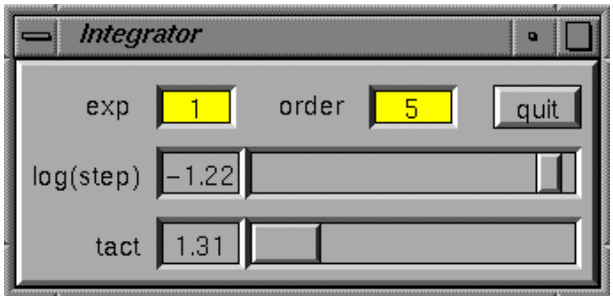


Figure 3.16: The "Integrator" window

We give a summary of the widgets.

exp This text widget shows the experiment number for which the IVP solver is active.

time This text widget shows the actual value of the integration variable.

log(step) This text widget shows the logarithm of the actual step-length in the IVP solver.

order This text widget shows the order of the integration (`order` $\in \{1, \ldots, 5\}$).

3.16 "Analyse minimum" window

In the "Analyse minimum" window we can choose how we want to see the results of an analysis of the minimum found in a local minimisation process. This can be as numerical data in a separate window (see Section 3.18), or as drawn confidence regions in the "Parameter $\cdots$" windows. Figure 3.17 shows the "Analyse minimum" window.

In this window we can make a selection from the following options.

show dependent confidence region When this choice is in the "yes" mode an analysis of the local minimum will be made ("anamin") when a computation of a local minimum ("locmin" or "steplocmin") has been completed. It shows the *dependent* confidence region as a green ellipse in the "Parameter $\cdots$" windows. A square green marker is drawn around the corresponding parameter.

show independent confidence region As previous option, but now it applies to the *independent* confidence region. The line of the independent confidence region and the marker around the parameter are blue.

show singular vectors Option is under construction.

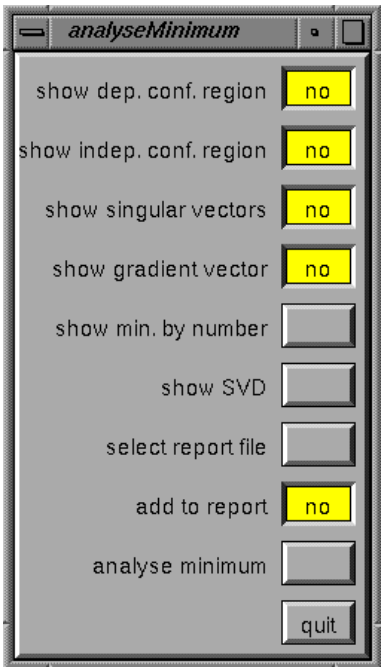


Figure 3.17: The "Analyse minimum" window

show gradient vectors Option is under construction.

show minimum by number This button opens the "Show minimum by number" window (see Section 3.17).

show SVD This button opens the "Show SVD" window (see Section 3.18).

select report file Option is under construction.

add to report Option is under construction.

3.17 "Show minimum by number" window

In the "Show minimum by number" window (see Figure 3.18) we give some numerical information about confidence regions.

This window shows a browser in which some variables are shown (see Figure 3.18). These are the names of the parameters (parnam), the parameter values for which an analysis of the minimum ("anamin") has been made, the variables `parICI`, `parDCI` and the condition number of the right orthogonal matrix V^T (see Section 3.18). The sizes of the independent and the dependent confidence regions are given by `parICI[i]` and `parDCI[i]` respectively. In Section 3.4 the quantity `Falpha` was mentioned; now we can determine the (1-Falpha) independent and dependent confidence regions by `parvis[i]` $\pm$ `parICI[i]` and `parvis[i]` $\pm$ `parDCI[i]` respectively. In this, `parvis[i]` is the i-th component of the parameter vector. (As long as the visualisation of the GUI is synchronous with the calculation in the numerical engine, Actpar and parvis are equal.)

clear This button clears all the information stored in the browser.

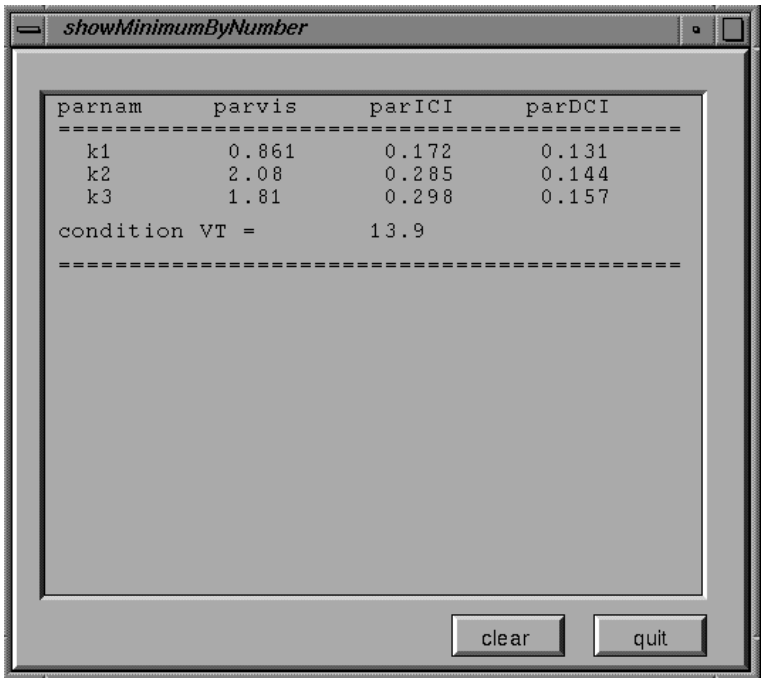


Figure 3.18: The "Show minimum by number" window

3.18 "Show SVD" window

The "Show SVD" window gives graphical insight with respect to the parameters which can be estimated relatively well. Figure 3.19 shows the "Show SVD" window.

We give a summary of the widgets.

$^{10}\log(sigma[i])$ The picture shown under $^{10}\log(sigma[i])$ is a histogram of $^{10}\log(\sigma_i)$. The σ_i (with $i \in \{1, \ldots, \texttt{nop}\}$) are the singular values of the Jacobian matrix $J = \partial y_j / \partial p_i$, evaluated at the local minimum (see [2] on page 6). When a p_i is fixed (see Section 3.14) we see a "f" in the corresponding place in the histogram.

`V[j,i]**2` The picture shown under `V[j,i]**2` is a `nop` by `nop` grid. Each cell of the grid corresponds with an entry in the matrix V from the singular value decomposition (SVD) $J = U\Sigma V^T$. So the i-th column is associated with the i-th singular value and the j-th row is associated with the j-th parameter. As the colour of a entry is associated with v_{ji}^2, it indicates what parameters are associated with what singular vectors. In this way we can immediately see what parameters are well determined. A $v_{ji}^2 \leq shift$ (see next option) is mapped onto a grey colour. A higher value corresponds with a lighter grey colour. A $v_{ji}^2 > shift$ is mapped onto red. When a p_i is fixed (see Section 3.14) the corresponding entries v_{ji}^2 are black with a green cross in it.

shift With this slider we can determine the shift value. This value plays a role in the way the v_{ji}^2 entries are coloured.

When we press the *left mouse button* on a particular entry v_{ji}^2 a text widget comes up containing the values v_{ji}^2 and σ_i or the text "fixed", when the entry v_{ji}^2 corresponds with a fixed parameter component. When we release the mouse on the entry this text widget disappears.

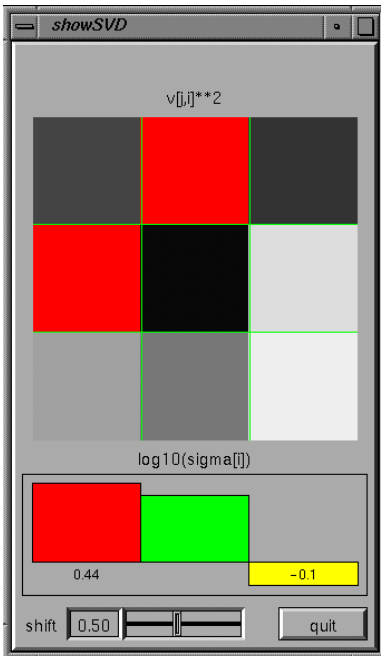


Figure 3.19: The "Show SVD" window

3.19 "Info" window

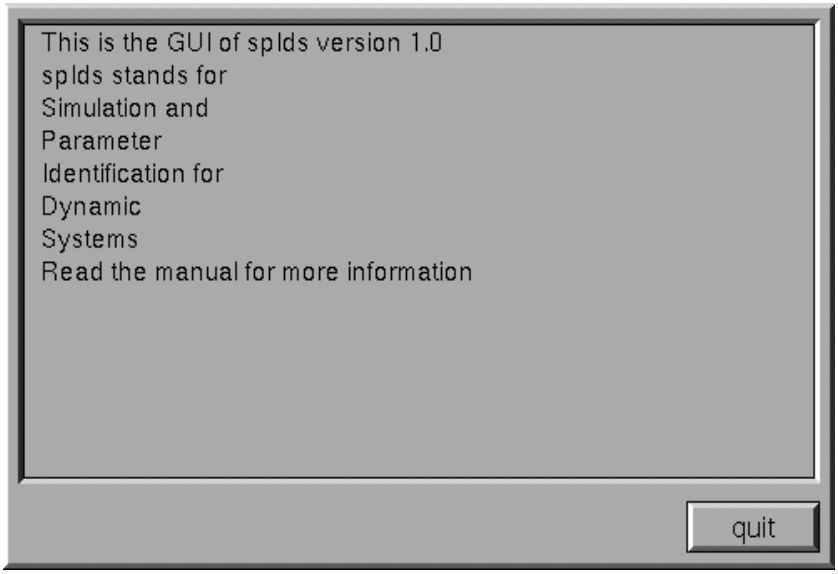


Figure 3.20: The "Info" window

Figure 3.20 shows the "Info" window. This browser contains information about the spIds application. This option is still under construction.

3.20 "Help" window

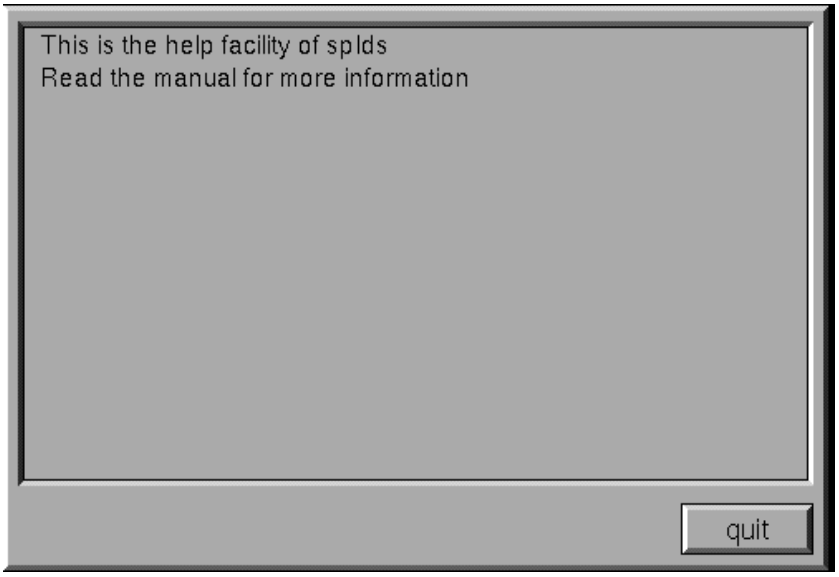


Figure 3.21: The "Help" window

Figure 3.21 shows the "Help" window . In this browser we want to offer some help. This option is still under construction.

Acknowledgement

The authors would like to thank Robert van Liere of the department of interactive systems at CWI for the many discussions concerning the data base manager and the adaptations he made for this application.

Tobias Baanders made the spIds logo look more professional on the basis of a draft from one of the authors.

Appendix A

List of Abbreviations

Table A.1: Summary of used abbreviations

abbreviation	meaning	page
Act	active	13,24
Actpar	active parameter vector	40
ActSOS	active sum of squares discrepancies	40
AE	algebraic engine	vi
aengine	algebraic engine	vi
anapar	analyse parameter	28
bg	background	37,38
brkpnt	breakpoint	30
comp	computation	28
count	counter for the number of Monte Carlo simulations	39
DAE(s)	differential algebraic equation(s)	1
DCI	dependent confident interval	42
del	delete	39
dev	deviation	39
dmserver	data base manager server	vi,19
exp	experiment	31
fact	factor	37
Falpha	α-point of the Fishers distribution	31
FDM	from data base manager	37
glomin	global minimisation	28
GUI	graphical user interface	21
ICI	independent confidence interval	42
Ini	initial	35,40
Inipar	initial parameter vector	35
IniSOS	initial sum of squared discrepancies	40
IVP	initial value problem	1
IVPhrt	ratio of the minimum and maximum step size of the initial value problem solver	31

abbreviation	meaning	page
IVPtol	local error bound of the initial value problem solver	30
LM	Levenberg-Marquardt	28
LMatol	absolute tolerance for the Levenberg-Marquardt minimisation	31
LMlamb	tuning parameter (λ) of the Levenberg-Marquardt minimisation	31
LMMxIt	maximum number of Levenberg-Marquardt minimisation steps	31
LMrtol	relative tolerance for the Levenberg-Marquardt minimisation	31
locmin	local minimisation	28
log	logarithm (base 10)	41,43
MC	Monte Carlo	28
MCsim	Monte Carlo simulation	28
NE	numerical engine	29
nengine	numerical engine	vi,19
`noaq`	number of algebraic equations	2
`nobs`	number of observations	3
`noc`	number of constants	3
`nodq`	number of differential equations	2
noMC	number of Monte Carlo simulations	39
`nop`	number of parameters	2
`noq`	number of number of equations	2
`nosid`	number of side relations	3
ODE(s)	ordinary differential equation(s)	1
par	parameter or parameter vector	29
parDCI	dependent confidence interval of active parameter vector	42
parICI	independent confidence interval of active parameter vector	42
parnam	user defined name of a component of the parameter vector	42
parvis	parameter vector used for visualisation purposes	42
spIds	simulation and parameter identification for dynamic systems	1
SqSabt	absolute tolerance with respect to the sum for squared discrepancies during the minimisation	30
SqSret	relative tolerance with respect to the sum for squared discrepancies during the minimisation	30

abbreviation	meaning	page
sosmax	upper bound for the sum of squared discrepancies, (only used for visualisation purposes)	38
sum	sum of squared discrepancies	28
SVD	singular value decomposition	43
win	window	33
x-axis	horizontal axis of a solution or parameter window	31
xmax	upper bound for visualisation purposes along the horizontal axis	32
xmin	lower bound for visualisation purposes along the horizontal axis	32
y-axis	vertical axis of a solution or parameter window	32
ymax	upper bound for visualisation purposes along the vertical axis	32
ymin	lower bound for visualisation purposes along the vertical axis	32

Appendix B

The structure of spIds

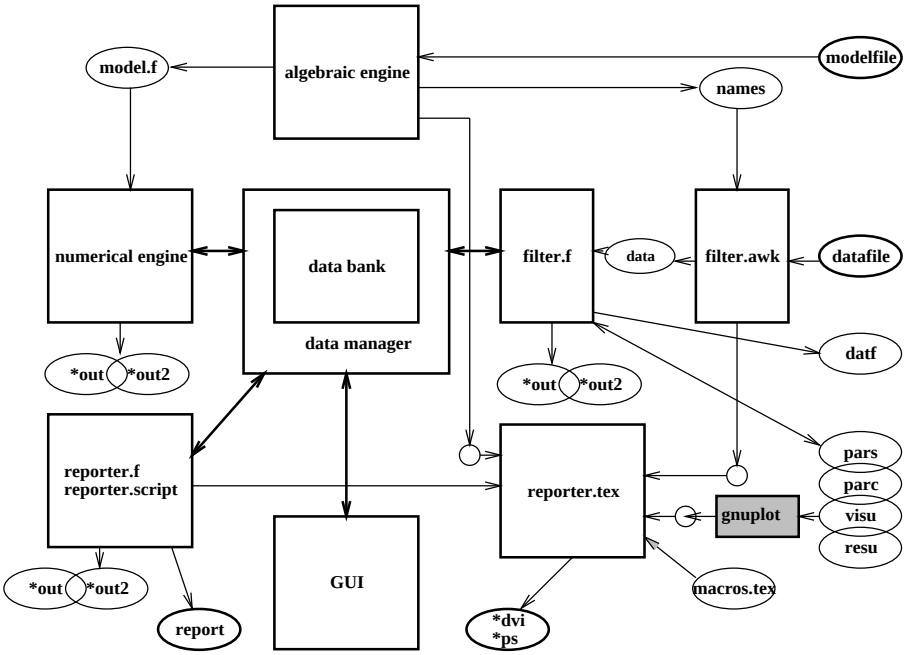


Figure B.1: The structure of the spIds application

Appendix C

Survey of all options in the GUI

In this section we will give a tree structure of all the options (actions) in the GUI. When such an option leads to another window a separate section is devoted to it in Chapter 3. In the subsequent overview these sections are listed between brackets. All the other options are explained in Section 3.2.

- change model
  - select file
  - edit file
  - activate file
- change data
  - select file
  - edit file
  - activate file
  - edit current data (3.3)
    - time
    - component
    - measurement
    - weight
    - active
    - brkpnt
    - scroller
    - cancel
    - ok
    - done
  - save current data
- change control
  - select file
  - edit file
  - activate file
  - edit current control (3.4)
    - SqSabt
    - SqSret
    - IVPtol

IVPhrt
LMMxIt
LMlamb
LMrtol
LMatol
Falpha
cancel
ok
done
save current control
solution space
create solution window (3.5)
exp
x-axis
y-axis
data marker
line colour
line width
make window (3.6)
exp
xmin
xmax
ymin
ymax
menu
edit window (3.7)
exp
x-axis
y-axis
data marker
line colour
line width
show y win
quit
print
quit
cancel
merge solution window
parameter space
create parameter window (3.8)
x-axis
y-axis
make window (3.9)
xmin
xmax
ymin

- ymax
- menu
  - edit window (3.10)
    - x-axis
    - y-axis
    - quit
  - print
  - quit
- cancel

global editors
- global edit solution window (3.11)
  - exp
  - fact
  - fdm
  - bg colour
  - bounds for t
  - bounds for y
  - cancel
  - ok
  - done
- global edit parameter windows (3.12)
  - path colour
  - path width
  - bg colour
  - sosmax
  - delete paths
  - delete points
  - quit

history
- save
- load

report

miscellaneous
- draw grid in solution windows
- draw grid in parameter windows
- MCsim (3.13)

parameter & sum info (3.14)
- IniSOS
- ActSOS
- IniPar
- fixed
- ActPar
- ParMin
- ParMax
- ini = act
- cancel

- ok
- done

integrator (3.15)
- exp
- time
- step
- order
- quit

analyse minimum (3.16)
- show dependent confidence region
- show independent confidence region
- show singular vectors
- show gradient vectors
- show minimum by number (3.17)
  - clear
  - quit
- show SVD (3.18)
  - `V[j,i]**2`
  - $^{10}\log(sigma[i])$
  - shift
  - quit
- select report file
- add to report
- quit

start computation

stop computation

spIds info (3.19)
- quit

spIds help (3.20)
- quit

quit

filter task

NE task

warning

Bibliography

[1] P.W. Hemker. Numerical methods for differential equations in system simulation and in parameter estimation. In H.C. Hemker and B. Hess, editors, *Analysis and Simulation of Biochemical Systems*, pages 59–80. North Holland Publ. Comp., 1972.

[2] P.W. Hemker and J. Kok. *A Project on Parameter Identification in Reaction Kinetics.* Technical Report NM-R9301, Centrum voor Wiskunde en Informatica, P. O. Box 4079, 1009 AB Amsterdam, The Netherlands, 1993.

[3] M.H. Overmars. *The Forms Library version 2.2*, 1993.

[4] SGI. *Graphics Library programming guide*, 1991.

[5] J.J. van Wijk and R. van Liere. *An Environment for Computational Steering.* Technical Report CS-R9448, Centre for Mathematics and Computer Science (CWI), 1994. Presented at the Dagstuhl Seminar on Scientific Visualization, 23-27 May 1994, Germany, proceedings to be published.

Index