PARSING PARAMETER ESTIMATION PROBLEMS FROM EASY-FIT TO $\mathbb{SOCS}$

by
Matthew W. Donaldson

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE, HONOURS

AT

DALHOUSIE UNIVERSITY
HALIFAX, NOVA SCOTIA
APRIL 8, 2005

DALHOUSIE UNIVERSITY

DEPARTMENT OF MATHEMATICS STATISTICS AND COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**PARSING PARAMETER ESTIMATION PROBLEMS FROM EASY-FIT TO SOCS**" by **Matthew W. Donaldson** in partial fulfillment of the requirements for the degree of **Bachelor of Science, Honours**.

Dated: April 8, 2005

Supervisor: _____
Dr. Raymond Spiteri

Reader: _____
Dr. Jason Brown

DALHOUSIE UNIVERSITY

Date: **April 8, 2005**

Author: **Matthew W. Donaldson**

Title: **PARSING PARAMETER ESTIMATION PROBLEMS FROM EASY-FIT TO $\mathbb{SOCS}$**

Department: **Mathematics Statistics and Computer Science**

Degree: **B.Sc. (Hon)** Convocation: **May** Year: **2005**

Table of Contents

List of Tables

List of Figures

Abstract

In many cases, mathematical models involve parameters that must be fit to experimental data. These so-called parameter estimation problems have many applications that may involve differential equations, optimization, and control theory. In this thesis we consider only parameter estimation problems that involve explicit model functions, ordinary differential equations, and differential-algebraic equations.

This thesis reviews two software packages, EASY-FIT and $\mathbb{SOCS}$, which are used to solve parameter estimation problems [5], [3]. We discuss the design of a parser used to translate EASY-FIT input into $\mathbb{SOCS}$ input so that is it possible to quickly test $\mathbb{SOCS}$ on a number of parameter estimation problems varying both in size and difficulty.

After parsing a small subset of parameter estimation problems from each of the three categories given above, we find that the parser performs very well. We are able to test $\mathbb{SOCS}$ on this subset of problems in a matter of seconds. This is a small fraction of the time it would take to code each problem separately in $\mathbb{SOCS}$. Although there were differences in some of the solutions found by EASY-FIT and $\mathbb{SOCS}$, they do not appear to be a result of the parser.

Acknowledgements

I would like to acknowledge the following people for all the help they have given me and which has led to making this thesis a great success:
Dr. Raymond J. Spiteri, Dr. John T. Betts, and Dr. Klaus Schittkowski.

Chapter 1

Introduction

Optimal control problems are most often governed by ordinary differential equations (ODEs) and arise in a wide range of applications. One particular field where optimal control problems are common is the aerospace industry. Aerospace engineers have been solving optimal control problems for trajectory optimization, spacecraft altitude control, jet thruster control, missile guidance, and many other applications for decades [2].

The particular type of optimal control problem that is considered in this thesis is the parameter estimation problem. Models that are used to match experimental data belong to an area of mathematics called parameter estimation. Parameter estimation is most important in models where a finite number of unknown parameters are difficult to measure directly from an experimental data set.

Methods for obtaining solutions to these parameter estimation problems are almost as numerous as the applications themselves. In this thesis we consider two software packages, EASY-FIT and $\mathbb{SOCS}$, which approximate the solutions of such problems.

The Sparse Optimal Control Software ($\mathbb{SOCS}$) package is a collection of FORTRAN 77 subroutines designed to solve optimal control problems [3]. The package implements the direct transcription or collocation method to convert the continuous control problem into a discrete one. The discretization gives a finite-dimensional nonlinear programming (NLP) problem, which is solved by $\mathbb{SOCS}$ using, for example, a sequential programming (SQP) method. A detailed description of the methods used by $\mathbb{SOCS}$ to solve NLP problems is given in [2], [3]. $\mathbb{SOCS}$ is supported on most UNIX and Windows systems. Perhaps one of the major disadvantages of $\mathbb{SOCS}$ is that the software does not have a graphical user interface (GUI).

EASY-FIT is an interactive software system used to estimate parameters in explicit model functions and dynamical systems of equations [5]. A dynamical system is usually described mathematically by a set of ODEs. The mathematical background of the numerical algorithms implemented in EASY-FIT is described in [8] in the form of a comprehensive textbook. EASY-FIT is implemented in the form of a relational database under Microsoft Access running under Microsoft Windows (2000 or higher). The underlying numerical algorithms are coded in FORTRAN and are executable independently from the interface.

In this thesis, we wish to test $\mathbb{SOCS}$ on a large number and also a wide variety of parameter estimation problems. To accomplish this goal, we take advantage of the over 1000 parameter estimation problems based on academic and real life examples included in EASY-FIT. To limit the amount of work required to test $\mathbb{SOCS}$ on such a large number of problems, we design a parser that allows for translation of EASY-FIT input into $\mathbb{SOCS}$ input. Specifically, the parser's job is to take a parameter estimation problem from the EASY-FIT database and output the executable $\mathbb{SOCS}$ code. As a result of this parser, we have access to a large database of parameter estimation problems without the need to consult multiple sources for problems, such as journals, textbooks, and other references.

The remainder of thgis thesis is as follows. In Chapter 2, we give details on the problem formulations of parameter estimation problems involving explicit model functions, ordinary differential equations, and differential-algebraic equations. In Chapter 3, the focus then is shifted to the input requirements for $\mathbb{SOCS}$ and EASY-FIT. We also give a simple example of how the parser works on a parameter estimation problem involving an explicit model function. Finally, in Chapter 4 we present the results for a number of each of the three types of parameter estimation problems.

Chapter 2

Parameter Estimation Problems

Parameter estimation is very important in areas such as natural science, engineering, and many other disciplines. The key idea is to estimate unknown parameters $p_1, \ldots, p_n$ of a mathematical model that often describes a real life situation by minimizing the distance of known experimental data from theoretically predicted values of a model function at certain time values. Therefore, model parameters that cannot be measured directly can be identified by a least-squares fit [8].

In simplified notation, we want to solve a least-squares problem of the form

$$\begin{aligned} \min_{\mathbf{p}} \sum_{i}^{l} (\mathbf{h}(\mathbf{p}; \mathbf{y}(\mathbf{p}; t_i), t_i) - y_i)^2 \ , \\ p_{lb} \leq \mathbf{p} \leq p_{ub} \ , \ \mathbf{p} \in \mathbb{R}^n \ , \end{aligned} \tag{2.1}$$

where $\mathbf{h}(\mathbf{p}; \mathbf{y}, t)$ is a fitting function depending on the unknown parameter vector $\mathbf{p}$, the time t, and the solution $\mathbf{y}(\mathbf{p}; t)$ of a dynamical system. A typical dynamical system is given by differential equations that describe a time-dependent process and that depend on the parameter vector $\mathbf{p}$. In this chapter, we summarize in detail how the model functions depend on the solution of a dynamical system. Moreover, we describe a couple of extensions of the data-fitting problem and the dynamical system that allow us to treat more complex practical models.

In general, parameter estimation problems often involve one the following categories,

- explicit model functions,
- Laplace transformations,
- dynamical systems of equations,
- systems of ordinary differential equations,

- systems of differential-algebraic equations,
- systems of one-dimensional, time-dependent partial differential equations,
- systems of one-dimensional partial differential-algebraic equations.

These are the sets of problems that can be found in the EASY-FIT software package. Of these 7 categories, the parser thus far has been designed to deal only with explicit model functions, systems of ordinary differential equations, and systems of differential-algebraic equations. We now turn to a more in-depth look at the first of these categories.

2.1 Explicit Model Functions

Problems where the vector-valued model function is available in explicit form belong to the class of parameter estimation problems known as explicit model functions. Associated with explicit models is an additional variable called time, and optionally another variable called concentration.

As described by [8], we begin with r experimental data sets

$$(t_i, c_j, y_{ij}^k),\ i = 1, \ldots, l_t,\ j = 1, \ldots, l_c,\ k = 1, \ldots, r, \tag{2.2}$$

where l_t time values, l_c concentration values, and $l = l_t l_c r$ corresponding measurement values are defined. We may also have nonlinear restrictions in the form of equality or inequality constraints on the parameters to be estimated,

$$\begin{aligned} g_j(\mathbf{p}) &= 0, \quad j = 1, \ldots, m_e\ , \\ g_j(\mathbf{p}) &\geq 0, \quad j = m_e + 1, \ldots, m_r\ . \end{aligned} \tag{2.3}$$

We assume that all constraint functions are continuously differentiable with respect to $\mathbf{p}$. Together with a vector-valued model function

$$\mathbf{h}(\mathbf{p}; t, c) = (h_1(\mathbf{p}; t, c), \ldots, h_r(\mathbf{p}; t, c))^T\ , \tag{2.4}$$

we get the resulting least-squares problem of the form

$$\begin{aligned} \min_{\mathbf{p}} \ & \sum_{k=1}^{r} \sum_{i=1}^{l_t} \sum_{j=1}^{l_c} (w_{ij}^k (h_k(\mathbf{p}; t_i, c_j) - y_{ij}^k))^2 \\ & g_j(\mathbf{p}) = 0, \ j = 1, \ldots, m_e \ , \\ & g_j(\mathbf{p}) \geq 0, \ j = m_e + 1, \ldots, m_r \ . \\ & p_{lb} \leq \mathbf{p} \leq p_{ub} \ , \ \mathbf{p} \in \mathbb{R}^n \ . \end{aligned} \tag{2.5}$$

We assume that there are weight factors $w_{ij}^k \geq 0$ given by the user that reflect the individual influence of a measurement value. Weights may be set to zero if there are measurements not available for a particular time value.

We wish to minimize the distance between the model function at certain time and concentration values and the corresponding measurement values. This distance is called the residual.

Note that for each of the three types of parameter estimation problems described in this Chapter it is possible to have a global scaling strategy in addition to the individual weight factors for each measurement value. The four options are:

- 0: no additional scaling
- 1: division of residuals by square root of sum of squares of all measurement values
- -1: division of each single residual by corresponding absolute measurement value
- -2: division of each single residual by corresponding squared measurement value

2.2 Ordinary Differential Equations

As with the explicit data-fitting model and as described by [8], we begin with r data sets,

$$(t_i, c_j, y_{ij}^k), \ i = 1, \ldots, l_t, \ j = 1, \ldots, l_c, \ k = 1, \ldots, r, \tag{2.6}$$

where l_t time values, l_c concentration values, and $l = l_t l_c r$ corresponding measurement values are defined. We may also have nonlinear restrictions in the form of equality or

inequality constraints on the parameters to be estimated,

$$\begin{aligned} g_j(\mathbf{p}) &= 0, \quad j = 1, \ldots, m_e \ , \\ g_j(\mathbf{p}) &\geq 0, \quad j = m_e + 1, \ldots, m_r \ . \end{aligned} \tag{2.7}$$

We assume that all constraint functions are continuously differentiable with respect to $\mathbf{p}$. The vector-valued model function

$$\mathbf{h}(\mathbf{p}; \mathbf{y}(\mathbf{p}; t, c), t, c) = (h_1(\mathbf{p}; \mathbf{y}(\mathbf{p}; t, c), t, c), \ldots, h_r(\mathbf{p}; \mathbf{y}(\mathbf{p}; t, c), t, c))^T \ , \tag{2.8}$$

depends on the concentration parameter c and in addition on the solution $\mathbf{y}(\mathbf{p}; t, c)$ of a system of m ordinary differential equations with initial values,

$$\begin{aligned} \dot{y}_1 &= F_1(\mathbf{p}; y, t) \quad , \ y_1(0) = y_1^0(\mathbf{p}) \ , \\ &\vdots \\ \dot{y}_m &= F_m(\mathbf{p}; y, t) \quad , \ y_m(0) = y_m^0(\mathbf{p}) \ . \end{aligned} \tag{2.9}$$

Without loss of generality, we assume that the initial time is zero. The initial values of the system of differential equations $y_1^0(\mathbf{p}; c), \ldots, y_m^0(\mathbf{p}; c)$ may depend on one or more of the system parameters to be estimated and on the concentration parameter c.

We then get the resulting least-squares problem of the form

$$\begin{aligned} \min_{\mathbf{p}} \ & \sum_{k=1}^{r} \sum_{i=1}^{l_t} \sum_{j=1}^{l_c} (w_{ij}^k (h_k(\mathbf{p}; \mathbf{y}(\mathbf{p}; t_i, c_j), t_i, c_j) - y_{ij}^k))^2 \\ & g_j(\mathbf{p}) = 0, \ j = 1, \ldots, m_e \ , \\ & g_j(\mathbf{p}) \geq 0, \ j = m_e + 1, \ldots, m_r \ . \\ & p_{lb} \leq \mathbf{p} \leq p_{ub} \ , \ \mathbf{p} \in \mathbb{R}^n \ . \end{aligned} \tag{2.10}$$

2.3 Differential-Algebraic Equations

Assuming the same problem formulation as given by (2.6) and (2.7), we now add algebraic equations to the system of differential equations (2.9). The resulting fitting

criterion $\mathbf{h}(\mathbf{p};\mathbf{y}(\mathbf{p};t,c),\mathbf{z}(\mathbf{p};t,c),t,c)$ depends on m_d differential variables $\mathbf{y}(\mathbf{p};t,c)$ and m_a algebraic variables $\mathbf{z}(\mathbf{p};t,c)$. The system of equations is now

$$\begin{aligned}
\dot{y}_1 &= F_1(\mathbf{p};\mathbf{y},\mathbf{z},t) \quad , \; y_1(0) = y_1^0(\mathbf{p}) \; , \\
&\vdots \\
\dot{y}_{m_d} &= F_{m_d}(\mathbf{p};\mathbf{y},\mathbf{z},t) \quad , \; y_{m_d}(0) = y_{m_d}^0(\mathbf{p}) \; , \\
0 &= G_1(\mathbf{p};\mathbf{y},\mathbf{z},t) \quad , \; z_1(0) = z_1^0(\mathbf{p}) \; , \\
&\vdots \\
0 &= G_{m_a}(\mathbf{p};\mathbf{y},\mathbf{z},t) \quad , \; y_{m_a}(0) = y_{m_a}^0(\mathbf{p}) \; ,
\end{aligned} \tag{2.11}$$

Again without loss of generality, we assume that the initial time is zero. The initial values of the differential equations $y_1^0(\mathbf{p};c),\ldots,y_{m_d}^0(\mathbf{p};c)$ and of the algebraic equations $z_1^0(\mathbf{p};c),\ldots,z_{m_a}^0(\mathbf{p};c)$ may depend on the system parameters to be estimated and on the concentration parameter c.

The least-squares problem for differential-algebraic equations has the form

$$\begin{aligned}
&\min_{\mathbf{p}} \sum_{k=1}^{r}\sum_{i=1}^{l_t}\sum_{j=1}^{l_c}(w_{ij}^k(h_k(\mathbf{p};\mathbf{y}(\mathbf{p};t_i,c_j),\mathbf{z}(\mathbf{p};t_i,c_j),t_i,c_j) - y_{ij}^k))^2 \\
&g_j(\mathbf{p}) = 0, \; j = 1,\ldots,m_e \; , \\
&g_j(\mathbf{p}) \geq 0, \; j = m_e+1,\ldots,m_r \; . \\
&p_{lb} \leq \mathbf{p} \leq p_{ub} \; , \; \mathbf{p} \in \mathbb{R}^n \; .
\end{aligned} \tag{2.12}$$

$\mathbf{y}(\mathbf{p};t,c)$ and $\mathbf{z}(\mathbf{p};t,c)$ are solution vectors of a system of $m_d + m_a$ differential-algebraic equations (DAEs). The system is called an index-1 problem or an index-1 DAE if the algebraic equations can be solved for $\mathbf{z}$ for all t. For simplicity, we consider only problems of index-1. EASY-FIT has an implicit solver that is able to solve index-2 DAEs and index-3 DAEs by transforming the higher-index problem to index-1 by successive differentiation of the algebraic equations. This transformation is performed because of the numerical instability of high-index DAEs. For more on DAEs with higher-index, see [6].

Chapter 3

Parser Design

Given that EASY-FIT has 780 parameter estimation problems of the sort described in Chapter 2, the design of a parser to translate these problems into $\mathbb{SOCS}$ input is essential when considering the amount of time required to code even the simplest of parameter estimation problems in $\mathbb{SOCS}$. This automated process of reading input from EASY-FIT and writing an output file which is directly executable by $\mathbb{SOCS}$ saves time as well as coding errors on the users part when it comes to solving such a large number of problems with $\mathbb{SOCS}$.

In this Chapter, we consider the input requirements for EASY-FIT and $\mathbb{SOCS}$. A list of function, variable, and other declarations used in the EASY-FIT input that are not supported by the parser are given. Finally, we present a sample EASY-FIT input file for an explicit model function along with the output file generated by the parser.

3.1 Input Format for EASY-FIT

This section is the exact description as given by [8] of the modelling language used by EASY-FIT called PCOMP. For a more complete description of PCOMP, see [8], [4]. All model functions are defined in the PCOMP modelling language, and, they are interpreted and evaluated during run time. The PCOMP-language is a subset of FORTRAN with a few extensions. In particular, the declaration and executable statements must satisfy the usual FORTRAN input format; e.g., they must start at column 7 or subsequently. A statement line is read in until column 72. Comments, denoted with `C` in the first column, may be included in a program text wherever needed. Statements may be continued on subsequent lines by including a continuation mark in column 6. Either capital or small letters are allowed for identifiers of the user and key words of the language, i.e., PCOMP is not case sensitive. The length of an

identifier must be smaller than 20 tokens.

In contrast to FORTRAN, however, most variables are declared implicitly by their assignment statements. Variables and functions must be declared separately only if they are used for automatic differentiation. PCOMP possesses special constructs to identify program blocks.

* `PARAMETER`
  Declaration of constant integer parameters to be used throughout the program, particularly for dimensioning index sets.

* `SET OF INDICES`
  Definition of index sets that can be used to declare data, variables, and functions or to define `sum` or `prod` statements.

* `INDEX`
  Definition of an index variable, which can be used in a FUNCTION program block.

* `REAL CONSTANT`
  Definition of real data, either without index or with one- or two- dimensional index. An index may be a variable or a constant number within an index set. Also arithmetic expressions may be included.

* `INTEGER CONSTANT`
  Definition of integer data, either without index or with one- or two-dimensional index. An index may be a variable or a constant number within an index set. Also arithmetic integer expressions may be included.

* `TABLE <identifier>`
  Assignment of constant real numbers to one- or two-dimensional array elements. In subsequent lines, one has to specify one or two indices followed by one real value per line in a free format (starting at column 7 or later).

* `VARIABLE`
  Declaration of variables with up to one index, with respect to which automatic

differentiation is to be performed.

* `CONINT <identifier>`
  Declaration of a piecewise constant interpolation function.

* `LININT <identifier>`
  Declaration of a piecewise linear interpolation function.

* `SPLINE <identifier>`
  Declaration of a spline interpolation function.

* `MACRO <identifier>`
  Definition of a macro function, an arbitrary set of PCOMP statements that define an auxiliary function to be inserted into subsequent function declaration blocks. Macros are identified by a name that can be used in any right-hand side of an assignment statement.

* `FUNCTION <identifier>`
  Declaration of functions either with up to one index, for which function and derivative values are to be evaluated. The subsequent statements must assign a numerical value to the function identifier.

* `END`
  End of the program.

From the above list, the parser recognizes only the following declarations: `PARAMETER`, `REAL CONSTANT`, `VARIABLE`, `FUNCTION <identifier>`, and `END`. Although this appears to be a small subset of the possible declarations in the PCOMP language, most EASY-FIT input files do fall into this category and therefore can be handled by the parser. If one of the declarations that is not recognized by the parser is used, a warning is given and this part of the input file is ignored, likely resulting in an output file that is missing crucial information.

The following 3 subsections deal with the input format of the model functions that must be defined in EASY-FIT using the PCOMP language. The parser has been designed to follow the same rules as PCOMP, and so based on the importance

of understanding these rules, we provide them as a reference. Again, for a more complete description see [8].

3.1.1 Input of Explicit Model Functions

To define explicit fitting functions in PCOMP, certain guidelines for the declaration of parameters and functions must be followed. The order in which these items are defined is essential for the interface between the input file and the data-fitting code. For defining variables, we need the following rules:

- The first variable names are identifiers for the n independent parameters to be estimated, i.e., for $p_1, \ldots, p_n$.
- If a concentration variable c exists, then a corresponding variable name must be added next.
- The last variable name identifies the independent time variable t for which measurements are available.
- Any other variables are not allowed to be declared.

Similarly, there are rules for the order in which model functions are defined:

- First, r fitting criteria $h_1(\mathbf{p}; t, c), \ldots, h_r(\mathbf{p}; t, c)$ must be defined depending on p, t, and optionally on c.
- The subsequent m_r functions are the constraints $g_1(\mathbf{p}), \ldots, g_{m_r}(\mathbf{p})$, if they exist. They may depend only on the parameter vector $\mathbf{p}$ to be estimated.
- No other functions are allowed to be declared.

The constants n, r, and m_r are defined in the database of EASY-FIT. These constants along with many other problem-specific constants are contained in a file separate from the PCOMP code. Each problem has associated with it a PCOMP file as well as a file defining these constants.

3.1.2 Input of Ordinary Differential Equations

For defining variables, we have the following rules:

- The first variables are identifiers for the n independent parameters to be estimated, $p_1, \ldots, p_n$.
- The subsequent m names identify the state variables of the system of ordinary differential equations, $y_1, \ldots, y_m$.
- If a concentration variable exists, then an identifier name must be added next that represents c.
- The last variable name identifies the independent time variable t, for which measurements are available.
- No other functions are allowed to be declared.

Similarly, we have rules for the order in which model functions are defined:

- The first m functions are the right-hand sides of the system of differential equations, the functions $F_1(\mathbf{p}; \mathbf{y}, t, c), \ldots, F_m(\mathbf{p}; \mathbf{y}, t, c)$.
- The subsequent m functions define the initial values, which may depend on the parameters to be estimated, and the concentration variable, $y_1^0(\mathbf{p}; c), \ldots, y_m^0(\mathbf{p}; c)$.
- Next, r fitting functions $h_1(\mathbf{p}; \mathbf{y}, t, c), \ldots, h_r(\mathbf{p}; \mathbf{y}, t, c)$ are defined depending on $\mathbf{p}$, $\mathbf{y}$, t, and c, where $\mathbf{y}$ denotes the state variable of the system of differential equations.
- The final m_r functions are the constraints $g_j(\mathbf{p})$ for $j = 1, \ldots, m_r$, if they exist at all, depending on the parameter vector $\mathbf{p}$ to be estimated.
- Any other functions are not allowed to be declared.

The constants n, m, r, and m_r are defined in the database of EASY-FIT. The last of the n parameters to be estimated are considered as switching points if they have been declared to describe certain model changes. Also n_b, the number of constant or

variable break points, must be defined beforehand.
Note: Presently, the parser is not designed to accommodate the use of switching or break points.

3.1.3 Input of Differential-Algebraic Equations

The following order of PCOMP variables is required:

- The first variable names are identifiers for n parameters to be estimated, $p_1, \ldots, p_n$.
- The subsequent m_d names identify the differential variables $y_1, \ldots, y_{m_d}$.
- The subsequent m_a names identify the algebraic variables $z_1, \ldots, z_{m_a}$.
- If a concentration variable exists, another identifier must be added next to represent c.
- The last variable name defines the independent time variable t for which measurements are available.
- No other functions are allowed to be declared.

Similarly, we have rules for the order in which the model functions are defined:

- The first m_d functions define the differential equations,
  $F_1(\mathbf{p}; \mathbf{y}, \mathbf{z}, t, c), \ldots, F_{m_d}(\mathbf{p}; \mathbf{y}, \mathbf{z}, t, c)$.
- The subsequent m_a functions are the right-hand sides of the algebraic equations, i.e., functions $G_1(\mathbf{p}; \mathbf{y}, \mathbf{z}, t, c), \ldots, G_{m_a}(\mathbf{p}, \mathbf{y}, \mathbf{z}, t, c)$.
- Subsequently, m_d functions define initial values for the differential equations, which may depend on the parameters to be estimated, and the concentration variable, $y_1^0(\mathbf{p}; c), \ldots, y_{m_d}^0(\mathbf{p}; c)$.
- Then m_a functions define initial values for the algebraic equations, which may depend on the parameters to be estimated, and the concentration variable, $z_1^0(\mathbf{p}; c), \ldots, z_{m_a}^0(\mathbf{p}; c)$.

- Next r fitting functions $h_1(\mathbf{p};\mathbf{y},\mathbf{z},t,c),\ldots,h_r(\mathbf{p};\mathbf{y},\mathbf{z},t,c)$ must be defined depending on $\mathbf{p}$, $\mathbf{y}$, $\mathbf{z}$, t, and c, where $\mathbf{y}$ and $\mathbf{z}$ are the differential and algebraic state variables of the DAE.

- The final m_r functions are the constraints $g_j(\mathbf{p})$, $j = 1,...,m_r$, if they exist. They may depend on the parameter vector $\mathbf{p}$ to be estimated.

- Any other functions are not allowed to be declared.

The constants n, m_d, m_a, r, and m_r are defined in the database of EASY-FIT and must coincide with the corresponding numbers of variables and functions, respectively. The last n_b fitting variables are considered as switching points, if they have been declared beforehand to describe certain model changes.
Note: At present, the parser is not designed to accommodate the use of switching points.

Many problem-specific constants are contained in a file separate from the PCOMP code (see Appendix B). For the specific details of the content of this file, see [5]. We provide only a brief summary of the important parts of the file used by the parser. Note that the line numbers have been added to easier identify the important parts of the file.

- Line 2: Model type (1 for explicit model, 4 for ordinary differential equations, 5 for differential-algebraic equations)

- Line 10: Number of unknown parameters

- Line 11: Number of inequality constraints

- Line 12: Number of equality constraints

- Line 13: Two integers (if present) defining concentration values to which the constraints are applied (equality then inequality constraints are listed)

- Line 14: Number of differential equations

- Line 15: Number of concentration values

- Line 16: Number of time measurements
- Line 17: Number of measurement sets (dimension of fitting function)
- Line 34: Parameter data (parameter names, lower bound for parameter, initial value for parameter, upper bound for parameter)
- Line 35: Scale type for weight factors
- Line 36: Data (time values, concentration values (if any), observation values, weights)

3.2 Input Format for $\mathbb{SOCS}$

In this section we give a brief description of the subroutines used by $\mathbb{SOCS}$ to solve a parameter estimation problem. For complete details of these and other subroutines in $\mathbb{SOCS}$, see [3]. The software for solving optimal control and parameter estimation problems can be divided into four classes:

1. the optimal control routine HDSOPE, which is called by the user to solve parameter estimation problems, and the input routine HHSOCS;
2. the user supplied subroutines needed to define the parameter estimation problem;
3. the optimization software needed to solve a sparse nonlinear programming subproblem;
4. the optimal control utility software available for special analysis and applications.

The following subroutines have generic names which may be changed by the user. In this description we use names that are consistent with those in the $\mathbb{SOCS}$ manual [3]. The user must define the problem using a routine called as ODEINP. This subroutine defines the phase-dependent problem input. It will be called once for each phase. ODEIGS is an optional user-supplied subroutine which defines the initial guess of the solution to an optimal control problem. Subroutine ODERHS permits the user

to define the right-hand sides of the differential-algebraic equations, and nonlinear boundary conditions can be constructed in subroutine ODEPTF. Optional output can be constructed in subroutine ODEPRT. Parameter estimation problems require input of the measurement data using subroutine DDLOAD. The following subroutines describe a collection of useful utility procedures available in the SOCS library, that are commonly needed for many applications. In particular the subroutine AUXOUT is an auxiliary output utility that can be used to display the optimal control solution produced by SOCS at either a fixed step size during the phase or at a specified number of points. The subroutine OCSEVL is used to evaluate the the optimal control solution at a few points. Subroutine AUXOUT may be more appropriate when the user wishes to display a complete time history of the solution. The primary function of OCSRNG is to construct estimates for the upper and lower bounds for the dynamic variables produced by SOCS. This information is often useful when constructing scale information as well as for display purposes. The subroutine LINKST is useful for linking dynamic variables across a phase boundary. Subroutines PHSLNG, PNTCON, and PTHCON are utility routines to simplify the specification of phase duration constraints, point functions, and path constraints, respectively. The organization of the subroutines used by SOCS to solve parameter estimation problems is illustrated in Figure 3.1. User-supplied subroutines are shown with double boxes. The optional subroutines are indicated by an asterisk. The user must call the SOCS algorithm HDSOPE and define the problem using the subroutine ODEINP. All other information is optional and may be supplied by the user or by using the dummy routines included in the SOCS library.

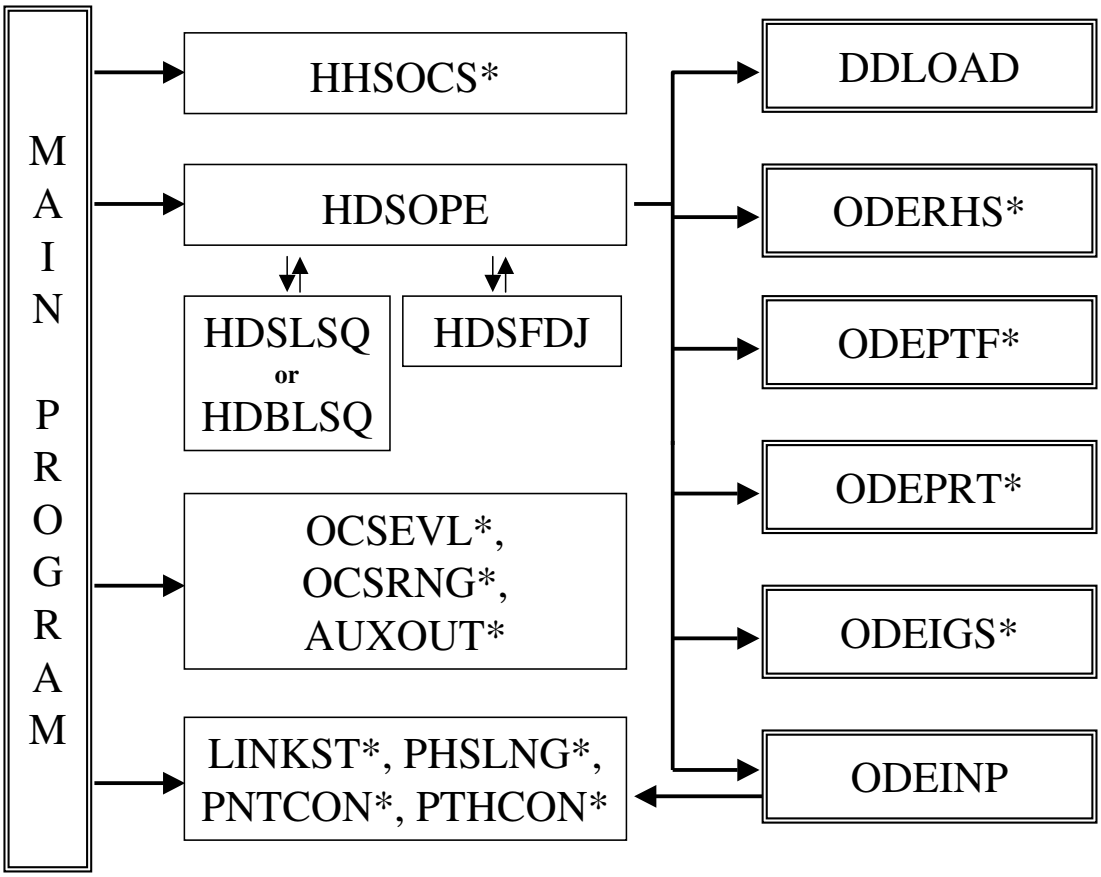

Figure 3.1: Organization of $\mathbb{SOCS}$ for Sparse Optimal Parameter Estimation

3.3 Sample Problem

We now have sufficient background to fully understand the translation of an EASY-FIT input file into a corresponding $\mathbb{SOCS}$ input file. As an example, we consider the simplest type of parameter estimation problem, an explicit model function with no constraint functions. For more examples, including parameter estimation problems involving differential equations, see Chapter 4.

The explicit model function we describe is called TP333 in the EASY-FIT software package. The experimental data can be found in Appendix A. The two integers on the first line give the size of the measurement set. The first column of data represents the time values, the second represents the observation values, and the third represents the weights associated with each measurement value. Other problem-specific constants are found in the data file in Appendix B. Following the problem formulation given in Section 2.1, we have $l_t = 8$ time values, $l_c = 1$ concentration value, $r = 1$ measurement set, and $l = l_t l_c r = 8$ corresponding measurement values. We wish to fit parameters $\mathbf{p} = (x_1,\ x_2,\ x_3)^T$ so that the data in Appendix A are approximated by the function

$$h(\mathbf{p}; t) = x_1 \exp(-x_2 t) + x_3 \ . \tag{3.1}$$

From Appendix B we see that $\mathbf{p}(0) = (30, 0.04, 3)^T$ and $0 \le x_1 \le 1000$, $0 \le x_2 \le 1000$, and $0 \le x_3 \le 1000$. The least-squares data-fitting problem is

$$\begin{aligned} &\min_{\mathbf{p}} \ \sum_{i=1}^{l} (h(\mathbf{p}; t_i) - y_i)^2 \ , \\ &0 \le \mathbf{p} \le 1000 \ , \ \mathbf{p} \in \mathbb{R}^3 \ . \end{aligned} \tag{3.2}$$

The corresponding EASY-FIT PCOMP file is the following:

```
C
C-----------------------------------------------------
C
C     Problem:    TP333
C
C     Date:       02.03.1994
```

```
C
C-----------------------------------------------------
C
C   - Independent variables in the following order:
C     1. parameters to be estimated (x)
C     2. concentration variable, if exists (c)
C     3. time variable (t)
C
*     VARIABLE
      x1, x2, x3, t
C
C-----------------------------------------------------
C
C   - Fitting criteria:
C
*     FUNCTION y
      y = x1*exp(-x2*t) + x3
C
C-----------------------------------------------------
C
C   - Constraints (if exist):
C
c*     FUNCTION G
c      G = ...
C
C-----------------------------------------------------
C
*     END
C
C-----------------------------------------------------
C
```

The parser-generated input file for $\mathbb{SOCS}$ is given in Appendix C. The parser also creates a data file containing the measurement data (Appendix A). As we will see, this data file is used by $\mathbb{SOCS}$ when solving the parameter estimation problem TP333. Along with the call to the parameter estimation solver in $\mathbb{SOCS}$, HDSOPE, the program also makes use of the subroutines EXPRHS, EXPINP, and EXPDDL.

In the subroutine EXPRHS, we define the model function. Because $\mathbb{SOCS}$ can only evaluate residuals on the state and/or algebraic variables and not functions of them, we need to introduce an algebraic variable into the equation defining the right-hand side of the fitting function. This equation then becomes a path constraint which we define in the input subroutine. EXPINP is used to define initial and final times, initial parameter values, parameter bounds, and the objective function. The user-defined subroutine INIEXP is called from EXPINP so that the data in Appendix A are loaded into the program. The data are then appropriately assigned to the correct variables in the subroutine EXPDDL. Here, the time values, the measurement values, and weights are assigned. If scaling of the residuals is needed, this is also done by calculating the correct values for the weight array based on which scaling option is chosen.

Chapter 4

Results and Discussion

To test the parser as well as the $\mathbb{SOCS}$ software package, we run the parser on explicit model functions, ODEs, and DAEs. Given that EASY-FIT contains over 700 problems of these three types and that the parser cannot translate some of these into $\mathbb{SOCS}$ input, we only summarize the results for 5 explicit model functions, 5 ODEs, and 2 DAEs. We describe in detail a single problem from each of these categories and present solution plots.

We begin with an example of an explicit model function along with the results obtained after using the parser to translate the EASY-FIT input file into $\mathbb{SOCS}$ input.

In the EASY-FIT problem INTEG_X, there are $l_t = 5$ time values, $l_c = 5$ concentration values, $r = 1$ measurement set, and $l = l_t l_c r = 25$ corresponding measurement values. The constant a is given the value 5.0. We wish to fit parameters $\mathbf{p} = (b_1,\ b_2,\ b_3)^T$ and concentration variable c so that the data in Table 4.1 are approximated by the function

$$h(\mathbf{p}; c, t) = \frac{b_1 \exp(-c)}{(1 - b_2 \exp(-a) + b_3 \exp(-t))} \ . \tag{4.1}$$

We also have that $\mathbf{p}(0) = (10, 10, 10)^T$ and $0 \le b_1 \le 10$, $0 \le b_2 \le 20$, and $0 \le b_3 \le 20$. The least-squares data-fitting problem is

$$\begin{aligned} \min_{\mathbf{p}} \ & \sum_{i=1}^{l_t} \sum_{j=1}^{l_c} (h(\mathbf{p}; c_j, t_i) - y_{ij})^2 \ , \\ & 0 \le b_1 \le 10 \ , \\ & 0 \le b_2 \le 20 \ , \\ & 0 \le b_3 \le 20 \ , \ \mathbf{p} \in \mathbb{R}^3 \ . \end{aligned} \tag{4.2}$$

The solution is summarized in Table 4.2. A plot of the experimental data and

$\mathbb{SOCS}$ solution for this problem is given in Figure 4.1.

t_i	c_i	y_i	w_i
1	1	0.183400496840477	1
2	1	0.26373416185379	1
3	1	0.338522046804428	1
4	1	0.353820204734802	1
5	1	0.368568271398544	1
1	1	0.061604518443346	1
2	2	9.24699977040291E-02	1
3	2	0.120332285761833	1
4	2	0.133212581276894	1
5	2	0.134147644042969	1
1	3	2.48860493302345E-02	1
2	3	0.034145575016737	1
3	3	4.56701554358006E-02	1
4	3	4.88160811364651E-02	1
5	3	4.85896691679955E-02	1
1	4	9.15580242872238E-03	1
2	4	1.29939131438732E-02	1
3	4	0.01550810970366	1
4	4	1.84268802404404E-02	1
5	4	1.84248797595501E-02	1
1	5	3.28793190419674E-03	1
2	5	4.76185046136379E-03	1
3	5	5.66617911681533E-03	1
4	5	6.43186643719673E-03	1
5	5	6.62047695368528E-03	1

Table 4.1: Experimental Data for problem INTEG_X

In the EASY-FIT ODE parameter estimation problem COMPET, we wish to fit simulated data that model a competition between two, as species described in [1]. The exact parameter values $p_1, \ldots, p_4$ are known to be $\mathbf{p}^* = (1.0, 1.0, 1.0, 0.99)^T$. The model functions are evaluated at $l = l_t r = 50$ points and a uniformly distributed error of 5% is added to the function values, giving the simulated data y_i, for $i = 1, \ldots, 25$.

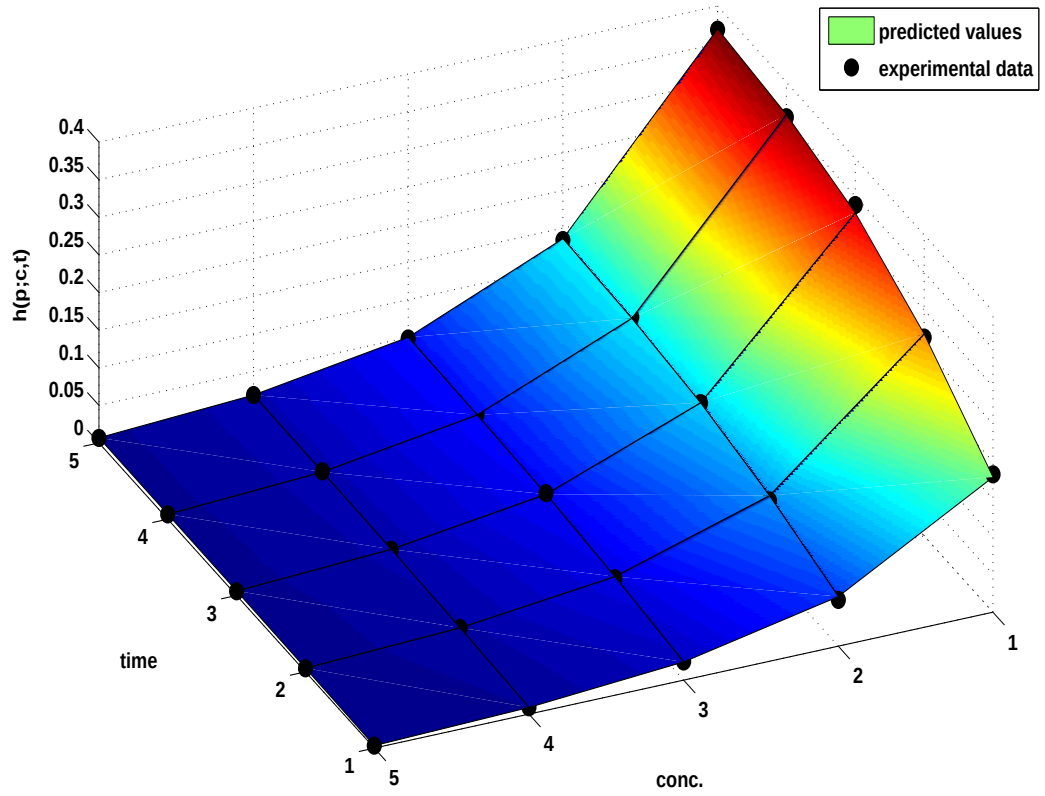

Figure 4.1: Plot of experimental data versus theoretically predicted model values obtained from $\mathbb{SOCS}$ for explicit model function INTEG_X.

The two ODEs used to model the simulated data are

$$\begin{aligned} \dot{y}_1 &= \frac{p_3 y_1(1-y_1)}{2} - p_1 y_1 y_2 \quad , \; y_1(0) = 0.02 \; , \\ \dot{y}_2 &= \frac{p_4 y_2(1-y_2)}{2} - p_2 y_1 y_2 \quad , \; y_2(0) = 0.02 \; . \end{aligned} \tag{4.3}$$

We then get the least-squares problem of the form

$$\begin{aligned} \min_{\mathbf{p}} \; & \sum_{k=1}^{r} \sum_{i=1}^{l_t} (w_i^k (h_k(\mathbf{p}; \mathbf{y}(\mathbf{p}; t_i), t_i) - y_i^k))^2 \\ & 0 \leq \mathbf{p} \leq 100 \; , \; \mathbf{p} \in \mathbb{R}^4 \; . \end{aligned} \tag{4.4}$$

Problem	**p** and obj. values	EASY-FIT	$\mathbb{SOCS}$
INTEG_X	p_1	0.967	0.972
	p_2	8.372	7.599
	p_3	2.819	2.834
	obj.	2.339×10^{-4}	4.242×10^{-4}
TP70	p_1	11.966	11.853
	p_2	4.770	4.880
	p_3	0.298	0.291
	p_4	2.091	2.120
	obj.	1.880×10^{-3}	1.834×10^{-3}
TP333	p_1	96.970	90.935
	p_2	0.074	0.067
	p_3	0.885	0.475
	obj.	3.02×10^{-4}	4.400×10^{-2}
EXP_P7	p_1	0	2.037×10^{-5}
	p_2	0	1.435×10^{-3}
	obj.	0	6.672×10^{-12}
DNS	p_1	70.257	47.162
	p_2	0.050	1.358×10^{4}
	p_3	2.978×10^{-3}	3.730×10^{-3}
	obj.	9.869×10^{-3}	3.280×10^{-2}

Table 4.2: Results obtained from EASY-FIT and $\mathbb{SOCS}$ for explicit model functions.

Beginning with the initial guess $\mathbf{p}(0) = (0.5, 0.5, 0.5, 0.5)^T$, we get the results summarized in Table 4.3. A plot of the experimental data and $\mathbb{SOCS}$ solution to this ODE parameter estimation problem is given in Figure 4.2.

The sample EASY-FIT parameter estimation problem involving a differential-algebraic equation is called BOND. The model represents the transition of a photon in a hydrogen-hydrogen bond [7]. There are $r = 3$ measurement sets with $l_t = 8$ time values, and $l = l_t r = 24$ measurement values. The unknown parameter vector is $\mathbf{p} = (k_{s1}, k_2, k_3, k_{s4})^T$. The model functions

$$\begin{aligned} h_1(\mathbf{p}; \mathbf{x}(\mathbf{p}; t), y(\mathbf{p}, t)) &= x_1, \\ h_2(\mathbf{p}; \mathbf{x}(\mathbf{p}; t), y(\mathbf{p}, t)) &= x_2, \\ h_3(\mathbf{p}; \mathbf{x}(\mathbf{p}; t), y(\mathbf{p}, t)) &= y, \end{aligned}$$

Problem	**p** and obj. values	EASY-FIT	$\mathbb{SOCS}$
COMPET	p_1	0.985	0.984
	p_2	0.999	0.999
	p_3	0.996	0.996
	p_4	0.991	0.991
	obj.	0.0219	0.0212
STAR	p_1	100.741	14.027
	p_2	1.311	1.306
	p_3	-1.713	-1.717
	obj.	0.173	0.172
EXP_SIN	p_1	1.566×10^{-33}	1.102×10^{-5}
	p_2	1.954	1.972×10^{-2}
	obj.	4.685	4.628
MOON	p_1	1.000×10^{-8}	1.246×10^{-8}
	obj.	8.288×10^{-2}	8.286×10^{-2}
WEIBEL	p_1	0.706	0.706
	p_2	0.456	0.457
	p_3	0.135	0.135
	obj.	1.349×10^{-3}	1.351×10^{-3}

Table 4.3: Results obtained from EASY-FIT and $\mathbb{SOCS}$ for ordinary differential equation models.

depend on the solution $x(\mathbf{p};t)$ and $y(\mathbf{p};t)$ of a system of 2 ordinary differential equations and 1 differential-algebraic equation

$$\begin{aligned} \dot{x_1}(\mathbf{p};x,y,t) &= -k_1x_1 + k_2y \quad , \quad x_1(0) = 0 \ , \\ \dot{x_2}(\mathbf{p};x,y,t) &= -k_4x_1 + k_3y \quad , \quad x_2(0) = 1 \ , \\ 0 &= k_1x_1 + k_4x_2 - (k_2+k_3)y \quad , \quad y(0) = 0 \ , \end{aligned}$$

where $k_1 = k_{s1}\cdot 1\times10^{-10}$ and $k_4 = k_{s4}\cdot 1\times10^{-10}$. The initial guess for the parameters is $\mathbf{p}(0) = (10, 0.5, 5, 1\times10^5)^T$ with lower and upper bounds given by $0 \le \mathbf{p} \le 1\times10^6$. The least-squares data-fitting problem is

$$\min_{\mathbf{p}} \ \sum_{k=1}^{r}\sum_{i=1}^{l_t}(w_i^k(h_k(\mathbf{p};\mathbf{x}(\mathbf{p};t_i),\mathbf{y}(\mathbf{p};t_i),t_i) - y_i^k))^2.$$

The solution is summarized in Table 4.4. A plot of the experimental data and

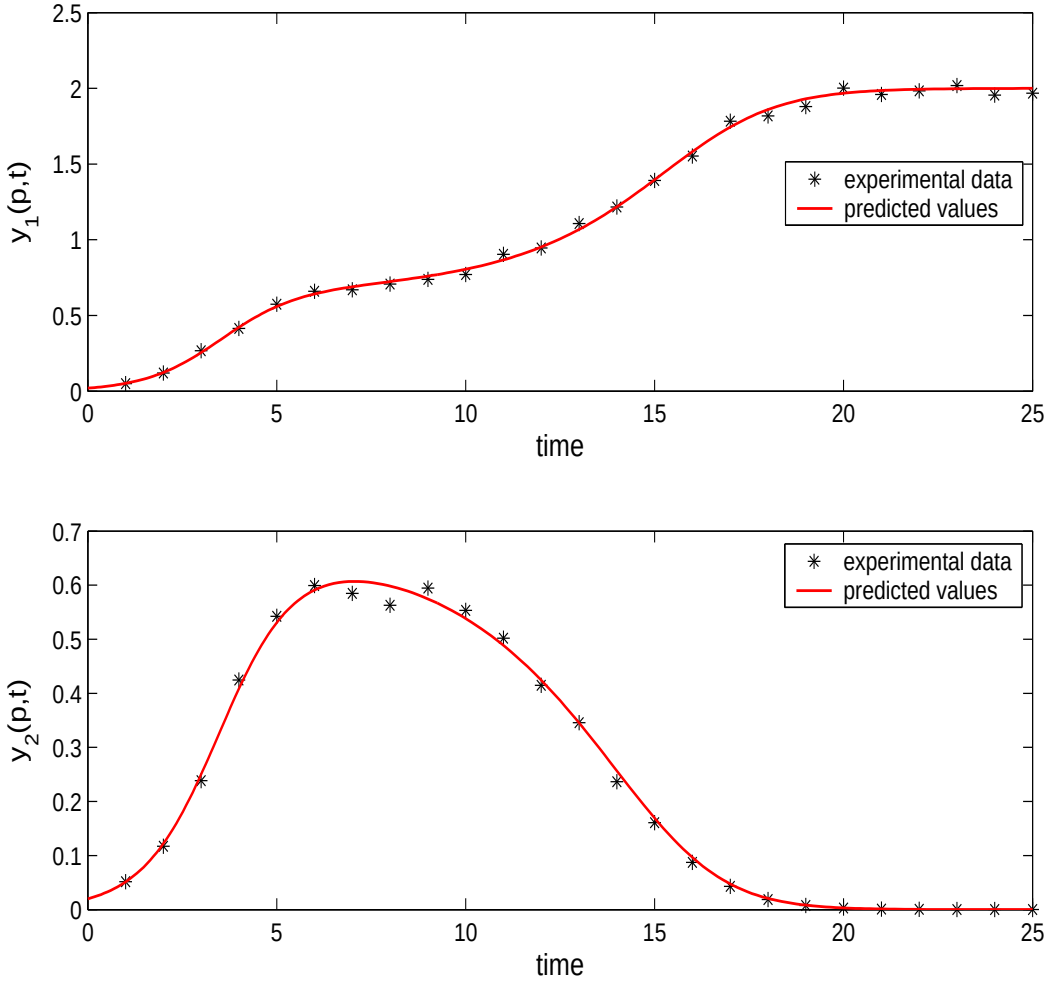

Figure 4.2: Plot of experimental data versus theoretically predicted model values obtained from $\mathbb{SOCS}$ for ODE parameter estimation problem COMPET.

$\mathbb{SOCS}$ solution to the EASY-FIT problem BOND is given in Figure 4.3.

Although it appears as if the solutions to the problems given by $\mathbb{SOCS}$ and EASY-FIT always agree, with the exception of a few parameter values, this is certainly not the case. There are many problems where $\mathbb{SOCS}$ finds a sub-optimal solution or simply returns with an error. These two issues are rare when parsing explicit model functions. However, for ordinary differential equation models, $\mathbb{SOCS}$ was able to find an optimal solution with no errors in only less than half of the problems that we tried. Because only a few differential-algebraic equations were attempted, it is difficult to comment on the efficacy of $\mathbb{SOCS}$ when using parser input files. Early indication is

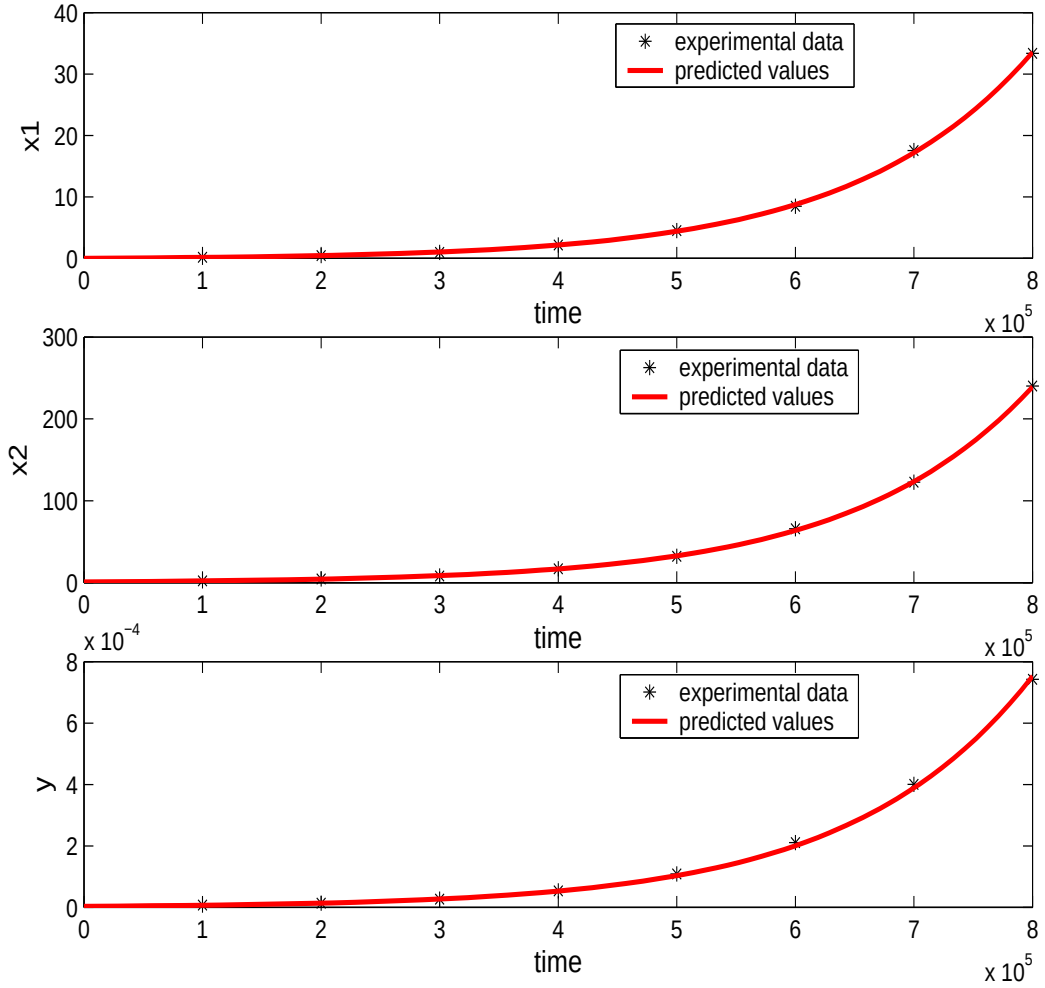

Figure 4.3: Plot of experimental data versus theoretically predicted model values obtained from SOCS for DAE parameter estimation problem BOND.

that the same errors encountered when parsing ordinary differential equations also occur for differential algebraic equations. Perhaps some of the errors reported by SOCS are because a local solution has been found rather than a global solution. Also, in most cases we are using the default values in SOCS.

As for the parser itself, the only case where errors occurred in the output file generated by the parser was when the EASY-FIT input file used a parameter or variable name that was previously used in the SOCS subroutine. No other errors appeared to be a direct result of the parser, assuming the conditions in Chapter 3.1 are valid.

Problem	**p** and obj. values	EASY-FIT	$\mathbb{SOCS}$
BOND	p_1	2.364×10^{5}	0
	p_2	0.307	0.297
	p_3	2.494	2.489
	p_4	8.780×10^{4}	8.765×10^{4}
	obj.	7.617×10^{-4}	7.233×10^{-4}
RESPIR	p_1	1.969	1.971
	p_2	0.246	9.110×10^{-3}
	p_3	4.969	4.968
	obj.	1.859×10^{-3}	1.829×10^{-3}

Table 4.4: Parser results obtained from EASY-FIT and $\mathbb{SOCS}$ for differential algebraic equation models.

A few additions are needed to the parser so that a larger subset of the differential-algebraic equation models can be parsed. The most important addition is that of accommodating parameters within the initial values of the differential variables. Also, adding any of the PCOMP declarations given in Chapter 3.1 that are not already part of the parser will allow for more types of problems to be parsed. It is unknown if all PCOMP declarations can be successfully included as part of the parser. There may be limitations of $\mathbb{SOCS}$ that will prevent the use of some PCOMP declarations.

Chapter 5

Conclusions and Future Work

The goal of this thesis was to describe the process of translating the input for EASY-FIT into the corresponding input for $\mathbb{SOCS}$. We began by introducing these two software packages that are used for solving parameter estimation problems. We also comment on the effectiveness of the translation in terms of comparing the solutions produced by both software packages.

The goal was made possible by the design of a parser. Only a subset of the parameter estimation problems included in EASY-FIT were parsed. Some problems could not be handled by the parser because they contained statements or declarations that were of an unknown format to the parser. Because a number of problems were correctly translated by the parser, it is believed that the errors produced by $\mathbb{SOCS}$ for some problems are not a result of incorrect translation. More research is needed to verify that this is indeed the case. Solving optimal control problems numerically can be challenging. Often very large parameter bounds can be the result of sub-optimal solutions. Local solutions may also be found; therefore it would be wise to use software with global optimization. However, the added computational expense associated with global optimization methods generally makes them infeasible except on small problems.

It would also be helpful to have more problems parsed to further investigate if any parameter estimation problems produce errors during the parsing process. By accommodating more PCOMP declarations than the current number, it will be possible to expand the range of problems that can currently be parsed. Furthermore, a careful review of why $\mathbb{SOCS}$ is not able to solve some problems will be crucial in designing a fast, reliable, and efficient parser.

It would also be useful to design a graphical user interface (GUI) for $\mathbb{SOCS}$. This would include a similar program such as the parser to convert user input into $\mathbb{SOCS}$

input. An addition of this sort would greatly enhance the usability of $\mathbb{SOCS}$. Based on the amount of code in Appendix C that solves an explicit model problem, it is quite obvious that a user new to the $\mathbb{SOCS}$ software will have significant difficulty in coding even the simplest of problems. On the other hand, when using a GUI, the user would need to know much less about the FORTRAN programming language and spend less time writing the $\mathbb{SOCS}$ code to solve the same problem. To solve a given problem, the user would only be required to input the problem into the GUI and could avoid lengthy FORTRAN code.

Overall, the parser performs well on all EASY-FIT problems that contain only the known declarations.

Appendix A

Measurement Data for Model TP333

```
8 3

4.00000E+00 7.210000000000E+01 1.00E+00
5.75000E+00 6.560000000000E+01 1.00E+00
7.50000E+00 5.590000000000E+01 1.00E+00
2.40000E+01 1.710000000000E+01 1.00E+00
3.20000E+01 9.800000000000E+00 1.00E+00
4.80000E+01 4.500000000000E+00 1.00E+00
7.20000E+01 1.300000000000E+00 1.00E+00
9.60000E+01 6.000000000000E-01 1.00E+00
```

Appendix B

Problem Data for Model TP333

```
1. problems\TP333
2. TP333         1
3. Exponential data fitting
4. Demo
5. Schittkowski
6. Experimental
7. Null
8. Null
9. t
10. NPAR  =     003    0  000
11. NRES  =     000
12. NEQU  =     000
13.
14. NODE  =       0
15. NCONC =     000
16. NTIME =    0008    0
17. NMEAS =     001
18. NPLOT =    0050
19. NOUT  =       0
20. METHOD=      01  000    2   -1
21. OPTP1 =   00110
22. OPTP2 =   00030
23. OPTP3 =      02
24. OPTE1 =   1.0000E-12
25. OPTE2 =   1.0000E-12
26. OPTE3 =   1.0000E+00
27. ODEP1 =       0
28. ODEP2 =       1    0    0    0   0
29. ODEP3 =      00
30. ODEP4 =       0
31. ODEE1 =   0.0
```

```
32. ODEE2 =   0.0
33. ODEE3 =   0.0
34.
x1               0.000000E+00        3.000000E+01        1.000000E+03
x2               0.000000E+00        4.000000E-02        1.000000E+03
x3               0.000000E+00        3.000000E+00        1.000000E+03
35. SCALE =      -1
36.
4.00000E+00 7.210000000000E+01 1.00E+00
5.75000E+00 6.560000000000E+01 1.00E+00
7.50000E+00 5.590000000000E+01 1.00E+00
2.40000E+01 1.710000000000E+01 1.00E+00
3.20000E+01 9.800000000000E+00 1.00E+00
4.80000E+01 4.500000000000E+00 1.00E+00
7.20000E+01 1.300000000000E+00 1.00E+00
9.60000E+01 6.000000000000E-01 1.00E+00
37. NLPIP         0
38. NLPMI         0
39. NLPAC         0.0
40. NDISCO=       0
y(t)
"Schittkowski K. (1987): More Test Examples for Nonlinear Programming,
Lecture Notes in Economics and Mathematical Systems, Vol. 282, Springer
Initial values:
3.00E+01 4.00E-02 3.00E+00"
```

Appendix C

$\mathbb{SOCS}$ Code for Problem TP333

```
      PROGRAM EXPLICIT_MODEL
C
C     ----THIS IS A PROGRAM FOR A PARAMETER ESTIMATION PROBLEM CHOSEN FROM
C         THE EASY-FIT SOFTWARE PACKAGE (TP333)
C
C     ****************************************************************
      IMPLICIT DOUBLE PRECISION  (A-H,O-Z)
      PARAMETER  (MXIW=10000000,MXRW=10000000,MXC=100000)
      PARAMETER (MXDP=100, MXPHS=1)
C
      COMMON /ODEWRK/ WORK(MXRW)
      COMMON /ODEIWK/ IWORK(MXIW)
      COMMON /ODESPL/ CSTAT(MXC)
      DIMENSION  IPCPH(MXPHS+1),DPARM(MXDP),IPDPH(MXPHS+1)
      PARAMETER (MAXRWD=8, MAXCLD=3)
C
      EXTERNAL  DUMYPF,  DUMYPR,  EXPINP,  EXPRHS,  EXPDDL,  DUMYIG
C
C     ****************************************************************
C
C     ----WORKING ARRAYS USED BY HDSOPE ROUTINE
C
      NIWORK = MXIW
      NWORK  = MXRW
      MAXCS = MXC
      MAXDP = MXDP
      MAXPHS = MXPHS
```

```
C
C     ****************************************************************
C
C     ----SETS EVERY OPTIONAL PARAMETER FOR THE SUBROUTINES HDSOCS AND
C         HDSOPE TO ITS DEFAULT VALUE
C
      CALL HHSOCS('DEFAULT')
C
C     ----MAXIMUM NUMBER OF DISCRETE DATA VALUES PER PHASE
C
      CALL HHSOCS('MXDATA=1000')
C
C     ----MAXIMUM NUMBER OF PARAMETERS PER PHASE
C
      CALL HHSOCS('MXPARM=3')
C
C     ----OPTIMAL CONTROL OUTPUT LEVEL (HIGHEST)
C
      CALL HHSOCS('IPGRD=20')
C
C     ----SPARSE NLP OUTPUT LEVEL (SET TO INTERPRETIVE OUTPUT)
C
      CALL HHSNLP('IOFLAG=20')
C
C     ----MAXIMUM NUMBER OF ITERATIONS
C
      CALL HHSNLP('NITMAX=500')
C
C     ----OUTPUT UNIT NUMBER
C
```

```
      IPU = 6
C
C     ----CALL HDSOPE WITH THE APPROPRIATE ARGUMENTS FOR THE PROBLEM
C
      CALL HDSOPE(EXPINP,DUMYIG,EXPRHS,DUMYPF,DUMYPR,EXPDDL,
     &            IWORK,NIWORK,WORK,NWORK,MAXPHS,
     &            CSTAT,MAXCS,IPCPH,DPARM,MAXDP,IPDPH,NEEDED,IER)
C
      STOP
      END
C
C
      SUBROUTINE EXPRHS(IPHASE,t^M,YVEC,NYVEC,PARM,NPARM,FRHS,NRHS,
     & IFERR)
C
C     ----EVALUATE RIGHT HAND SIDE OF DATA FITTING FUNCTION(S)
C
C     ****************************************************************
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
      DIMENSION YVEC(NYVEC),FRHS(NRHS),PARM(NPARM)
C
      PARAMETER (NCONC=1)
C
      DOUBLE PRECISION  x1, x2, x3, t
C
      COMMON /CONPAR/ CONC(NCONC)
C
C     ****************************************************************
C
C     ----LOAD PARAMETER VECTOR
C
      x1 = PARM(1)
```

```
      x2 = PARM(2)
      x3 = PARM(3)
C
C     ----COMPUTE VALUES FOR FITTING FUNCTION(S)
C
      y = x1*exp(-x2*t) + x3
C
      FRHS(1) = YVEC(1) - y
C
      IFERR = 0
C
      RETURN
      END
C
C
      SUBROUTINE EXPINP(IPHASE,NPHS,METHOD,NSTG,NCF,NPF,NPV,NAV,NGRID,
     &                  INIT,MAXMIN,MXPARM,P0,PLB,PUB,PLBL,
     &                  MXSTAT,Y0,Y1,YLB,YUB,STSKL,STLBL,MXPCON,CLB,CUB,
     &                  CLBL,MXTERM,COEF,ITERM,TITLE,IER)
C
C     ----INITIALIZE DATA FITTING PROBLEM
C
C     ****************************************************************
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
C  ARGUMENTS:
      INTEGER    IPHASE,NPHS,METHOD,NSTG,NCF(5),NPF(2),NPV,NAV,NGRID,
     &           INIT,MAXMIN,MXPARM,MXSTAT,MXPCON,MXTERM,
     &           ITERM(4,MXTERM),IER
      DIMENSION  P0(MXPARM),PLB(MXPARM),PUB(MXPARM),Y0(0:MXSTAT),
     &           Y1(0:MXSTAT),YLB(-1:1,0:MXSTAT),YUB(-1:1,0:MXSTAT),
     &           STSKL(0:MXSTAT+MXPARM,2),CLB(MXPCON),CUB(MXPCON),
     &           COEF(MXTERM)
```

```
      CHARACTER  TITLE(3)*60,PLBL(MXPARM)*80,STLBL(0:MXSTAT)*80,
     &           CLBL(0:MXPCON)*80
C
      COMMON /PESTCM/ TFINAL
      PARAMETER (MAXRWD=8,MAXCLD=3,NTIME=8)
      COMMON /EXPCM/ STDAT(MAXRWD,MAXCLD),NROWS
      PARAMETER (NCONC=1)
      COMMON /CONPAR/ CONC(NCONC)
C
      CHARACTER*60 LABEL
      DATA LABEL(1:60) /' '/
C
C     ******************************************************************
C
C     ----DEFINE INITIAL AND FINAL TIME
C
      TINITIAL = 4.000000D0
      TFINAL = 96.000000D0
C
C     ----NUMBER OF PHASES
C
      NPHS = 1
C
C     ----NUMBER OF GRID POINTS
C
      NGRID = NTIME
C
C     ----SUCCESS/ERROR CODE
C
      IER = 0
C
      NTERM = 0
      NKON = 0
```

```
      TITLE(1) = 'PARAMETER ESTIMATION PROBLEM: TP333'
      TITLE(2) = 'SOLVED USING LEAST SQUARES'
      STLBL(0) = 'TIME    Time'
C
C     ----SET INTREGRATION METHOD TO HERMITE-SIMPSON
C
      NSTG = 1
      METHOD = 3
C
C     ----INITIALIZE PROBLEM DATA
C
      CALL INIEXP
C
      PLBL(1) = 'p1       Parameter 1'
      PLBL(2) = 'p2       Parameter 2'
      PLBL(3) = 'p3       Parameter 3'
C
C     ----GUESS FOR INITIAL PARAMETER VALUES AND BOUNDS
C
      P0(1) = 30.000000D0
      P0(2) = 0.040000D0
      P0(3) = 3.000000D0
      PLB(1) = 0.000000D0
      PUB(1) = 1000.000000D0
      PLB(2) = 0.000000D0
      PUB(2) = 1000.000000D0
      PLB(3) = 0.000000D0
      PUB(3) = 1000.000000D0
C
C     ----INITIALIZE PROBLEM VARIABLES
C            NAV - NUMBER OF ALGEBRAIC VARIABLES
C            NPV - NUMBER OF DISCRETE PARAMETERS
C            NDE - NUMBER OF DIFFERENTIAL EQUATIONS
```

```
C            NDF - NUMBER OF DATA FITTING FUNCTIONS
C
      NAV = 1
      NPV = 3
      NDE = 0
      NDF = 1
C
C     ----NUMBER OF DIFFERENTIAL EQUATIONS
C
      NCF(1) = NDE
C
C     ----NUMBER ALGEBRAIC EQUATIONS
C
      NCF(2) = 0
C
C     ----NUMBER OF DISCRETE DATA FUNCTIONS
C
      NCF(5) = NDF
C
C     ----INITIAL GUESS TYPE FOR INTERNAL STATES: LINEAR GUESS
C         SEE SOCS MANUAL FOR DESCRIPTION AND OTHER OPTIONS
C
      INIT = 1
C
C     ----SET INITIAL AND FINAL TIME
C
      Y0(0) = TINITIAL
      Y1(0) = TFINAL
C
C     ----FIX INITIAL AND FINAL TIME
C
      YLB(-1,0) = Y0(0)
      YUB(-1,0) = Y0(0)
```

```
      YLB(1,0) = Y1(0)
      YUB(1,0) = Y1(0)
C
C     ----DEFINE GUESSES FOR INITIAL CONDITIONS FOR STATE VARIABLES
C
      Y0(1) = 0.D0
C
C     ----DEFINE GUESSES FOR FINAL CONDITIONS FOR STATE VARIABLES
C
      Y1(1) = 0.D0
C
C     ----DEFINE OBSERVATION PATH CONSTRAINT
C
      CALL PTHCON(NTERM,NKON,NCF,IPHASE,ITERM,MXTERM,COEF,
     $    CLB,CUB,CLBL,MXPCON,0.D0,0.D0,1.D0,'ACON1',
     $    'Algebraic constraint 1: u1 = 0',IERPTH)
C
C     ----DEFINE LEAST SQUARES OBJECTIVE
C
      MAXMIN = 2
      CLBL(0) = 'LSQ     DISCRETE DATA'
C
      DO KK = 1,NDF
         NTERM = NTERM + 1
C
C          TERM KK IS PART OF NLP OBJECTIVE FUNCTION
C
         ITERM(1,NTERM) = 0
C
C          TERM KK IS COMPUTED IN IPHASE
C
         ITERM(2,NTERM) = IPHASE
C
```

```
C          TERM KK IS ASSIGNED TO DISCRETE DATA
C
         ITERM(3,NTERM) = 2
C
C          TERM KK IS DISCRETE DATA FUNCTION KK
C
         ITERM(4,NTERM) = KK
      ENDDO
C
      RETURN
      END
C
C
      SUBROUTINE INIEXP
C
C     ----DATA FOR PARAMETER EXTIMATION PROBLEM
C         PROBLEM DATA IS READ IN FROM FILE TP333.DAT
C
C     *****************************************************************
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
      COMMON /PESTCM/ TFINAL
      PARAMETER (MAXRWD=8,MAXCLD=3,NTIME=8)
      COMMON /EXPCM/ STDAT(MAXRWD,MAXCLD),NROWS
      CHARACTER*80 DATFL1
      SAVE DATFL1
      DATA DATFL1 /'../Datasets/datafiles/TP333.dat'/
C
C     *****************************************************************
C
      IPN = 3
      OPEN(IPN,FILE=DATFL1,STATUS='UNKNOWN')
      READ(IPN,*) NROWS,NCOLS
```

```
      IF(NROWS.GT.MAXRWD) THEN
        PRINT *,'NROWS GT MAXRWD; NROWS =',NROWS
        STOP
      ENDIF
      DO I = 1,NROWS
         READ(IPN,*) (STDAT(I,JCOL),JCOL=1,NCOLS)
      ENDDO
      CLOSE(IPN)
C
      RETURN
      END
C
C
      SUBROUTINE EXPDDL(IPHASE,NDD,NDDST,MXDATA,TDATA,DATA,WTDATA,
     &                  NDATA,IER)
C
C     ----LOAD DISCRETE DATA FOR PARAMETER ESTIMATION PROBLEM
C
C     ***************************************************************
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
      PARAMETER (MAXRWD=8,MAXCLD=3,NTIME=8)
      COMMON /EXPCM/ STDAT(MAXRWD,MAXCLD),NROWS
      DIMENSION TDATA(MXDATA),DATA(MXDATA),WTDATA(MXDATA)
      PARAMETER (NCONC=1)
      COMMON /CONPAR/ CONC(NCONC)
C
C     ***************************************************************
C
C     ----RESET ERROR/SUCCESS CODE TO 0
C
      IER = 0
C
```

```
C     ----LOAD STATE NDD TARGET VALUES
C
      NDATA = NTIME
      SIZEDATA = NROWS
      IF(MXDATA.LT.SIZEDATA) THEN
        IER = -1
        RETURN
      ENDIF
      NDDST = NDD+0
C
      SUM_OF_SQRS = 0.D0
C
C     ----SET TIME VALUES TO FIRST COLUMN IN DATA FILE,
C         CONCENTRATION VALUES TO SECOND, FUNCTION VALUES TO THIRD,
C         AND WEIGHTS TO FOURTH. . .
C
      DO 310 I = 1,NTIME
        TDATA(I) = STDAT(I+(IPHASE-1)*NTIME,1)
        DATA(I) = STDAT(I+(IPHASE-1)*NTIME,2*NDD)
        WTDATA(I) = STDAT(I+(IPHASE-1)*NTIME,1+2*NDD)
        SUM_OF_SQRS = SUM_OF_SQRS + DATA(I)**2
 310  CONTINUE
C
C     ----SCALING OF RESIDUALS (IF NEEDED)
C
C            0 - NO ADDITIONAL SCALING
C            1 - DIVISION OF RESIDUALS BY SQUARE ROOT OF SUM
C                OF SQUARES OF ALL MEASUREMENTS VALUES
C           -1 - DIVISION OF EACH SINGLE RESIDUAL BY CORRESPONDING
C                ABSOLUTE MEASUREMENT VALUE
C           -2 - DIVISION OF EACH SINGLE RESIDUAL BY CORRESPONDING
C                SQUARED MEASUREMENT VALUE
C
```

```
      SCALE = -1
C
      DO 320 I = 1,NTIME
        IF(SCALE.EQ.1) THEN
          IF(SUM_OF_SQRS.NE.0.D0) THEN
            WTDATA(I) = 1.D0/SQRT(SUM_OF_SQRS)
          ENDIF
        ELSEIF(SCALE.EQ.-1) THEN
          IF(DATA(I).NE.0.D0) THEN
            WTDATA(I) = 1.D0/ABS(DATA(I))
          ELSE
            WTDATA(I) = 1.D0
          ENDIF
        ELSEIF(SCALE.EQ.-2) THEN
          WTDATA(I) = 1.D0/(DATA(I)**2)
        ENDIF
320   CONTINUE
C
      RETURN
      END
```

Bibliography

[1] E. Beltrami, *Mathematics for Dynamic Modeling*, Academic Press, Orlando, 1987.

[2] J.T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*, SIAM, United States of America, 2001.

[3] J.T. Betts and W.P. Huffman, *Sparse Optimal Control Software* $\mathbb{SOCS}$, Mathematics and Engineering Analysis Tech. Document MEA-LR-085, Boeing Information and Support Services, The Boeing Company, PO Box 3707, Seattle, WA 98124-2207, July 1997.

[4] M. Dobmann, M. Liepelt, K. Schittkowski, C. Traßl, *PCOMP: A Fortran code for automatic differentiation, language description and users guide*, Report, Dept. of Mathematics, University of Bayreuth, Germany, 1995.

[5] EASY-FIT, *Software for parameter estimation.* http://www.uni-bayreuth.de/departments/math/kschittkowski/easy_fit.htm.

[6] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*, Springer Series Computational Mathematics, Vol. 14, Springer, Berlin, 1991.

[7] L. Lapidus, R.C. Aiken, Y.A. Liu, *The occurence and numerical solution of physical and chemical systems having widely varying time constants*, in: Willoughby E.A. (ed.): Stiff Differential Systems, Plenum Press, 187-200, 1973.

[8] K. Schittkowski, *Numerical Data Fitting in Dynamical Systems - A Practical Introduction with Applications and Software*, Kluwer Academic Publishers, Dordrecht, Boston, London, 2002.