Parameter Estimation in Nonlinear Dynamical Systems

Master's Thesis

Supervisors:
Michael L. Michelsen & Per G. Thomsen

June 30th 2004

Morten Rode Kristensen

Student No. : s991431

Department of Chemical Engineering
Technical University of Denmark

Preface

This report constitutes my M.Sc. thesis. The thesis was prepared at the Center for Phase Equilibria and Separation Processes (IVC-SEP), Department of Chemical Engineering (KT) in collaboration with Informatics and Mathematical Modelling (IMM), both at the Technical University of Denmark. The work presented in the thesis was carried out from February 2nd, 2004, to June 30th, 2004.

First of all, I would like to thank my supervisors, Michael L. Michelsen, KT, and Per G. Thomsen, IMM, for their invaluable inputs during the many fruitful discussions that I have had with them, and for always taking the time to answer my questions. I would also like to thank John Bagterp Jørgensen, 2-Control ApS, who has been a co-supervisor on the project, for being an everlasting source of inspiration and motivation. Also thanks to Niels Rode Kristensen for proofreading the final version of the thesis and to my fellow students Jakob Kjøbsted Huusom, Kent Johansen, Jakob Sloth and Martin Dan Palis Sørensen for their moral support during my work on this thesis, but especially for the many hours that we have shared together during the past five years working on various projects and assignments.

The topic of this thesis is parameter estimation in dynamical systems, which is a wide area involving many different aspects of statistics as well as numerical analysis, at the same time being closely linked to mathematical model building and experimental design. Many of the areas touched upon in this thesis, such as solution of differential-algebraic equations or regularization of ill-conditioned problems, really deserve much more attention. There is much more to be said about these topics than what is included in this thesis. The main goal of this thesis, however, is to tackle parameter estimation problems in dynamical systems by drawing on knowledge about numerical optimization, differential equation solution, etc. It is hoped that the results presented in this thesis, in terms of tools and guidelines for systematic handling of the many difficulties often encountered in parameter estimation, provide a useful methodology for the practical solution of parameter estimation problems.

With respect to the structure of the thesis, it is composed of three consecutive parts plus appendices: A literature review including the objective of the work, a progress report and a final report including the overall conclusion.

Kgs. Lyngby, June 30th, 2004

Morten Rode Kristensen

Summary

The subject of this thesis is parameter estimation in nonlinear dynamical systems. More specifically, attention is restricted to systems modelled by ordinary differential equations and differential-algebraic equations. Estimating parameters in such systems is both computationally intensive as well as numerically challenging due to a variety of undesirable characteristics, such as ill-conditioning and stiffness of model equations, often exhibited by real parameter estimation problems. By gaining a detailed understanding of the numerical algorithms involved, the goal of this thesis is to provide a systematic approach for the practical solution of dynamical parameter estimation problems and, hopefully, contribute to the development of efficient and robust parameter estimation software.

A review of the relevant literature is given emphasizing the numerical aspects of parameter estimation. Based on this review a series of benchmark tests are performed comparing the performance of different optimization algorithms and differential equation solvers. The results from these tests are used to motivate the choice of a Levenberg-Marquardt method for solving the least squares optimization problem and a specifically tailored equation solver of the Runge-Kutta family for joint state and sensitivity integration. These methods are incorporated into a new program called PARFIT designed to facilitate parameter estimation in dynamical systems. Among several features the program implements an adaptive tolerance selection mechanism to improve efficiency and an option for use of regularization to improve robustness in ill-conditioned problems. A method for tracing the optimal solution from a poor initial parameter guess by using regularization is proposed. This framework, consisting of the PARFIT program along with guidelines for systematically addressing the difficulties often encountered in dynamical parameter estimation, is the main result of the thesis. The framework is successfully applied to a notoriously difficult problem from chemical reaction engineering.

Resumé på dansk

Emnet for denne afhandling er parameterestimering i ikke-lineære dynamiske systemer. Specielt betragtes systemer, der kan modelleres med et sæt af ordinære differentialligninger eller differential-algebraiske ligninger. Parameterestimering i sådanne systemer er både beregningsmæssigt krævende og numerisk udfordrende på grund af en række vanskeligheder, såsom dårlig konditionering og stivhed i modelligninger, der ofte optræder i realistiske parameterestimeringsproblemer. Målet med denne afhandling er, gennem en detaljeret forståelse af de numeriske algoritmer, at udvikle en systematisk tilgang til den praktiske løsning af dynamiske parameterestimeringsproblemer, og derigennem forhåbentligt at bidrage til udviklingen af effektive og robuste parameterestimeringsprogrammer.

Der er indledningsvist givet en oversigt over den relevante litteratur med vægt på de numeriske aspekter. Med udgangspunkt heri er der udført en række sammenligninger af forskellige optimeringsalgoritmer og differentialligningsløsere. Resultatet af sammenligningerne er brugt til at motivere valget af en Levenberg-Marquardt metode til løsning af mindste kvadraters optimeringsproblemet samt en specielt designet Runge-Kutta differentialligningsløser til samtidig løsning af model- og sensitivitetsligninger. Optimeringsalgoritmen og differentialligningsløseren er indbygget i et nyt program (PARFIT) til parameterestimering i dynamiske systemer. Programmet benytter bl.a. en adaptiv metode til justering af optimeringstolerancen for at øge effektiviteten. Herudover er der indbygget en mulighed for at benytte regularisering for at øge robustheden ved løsning af dårligt konditionerede problemer. Med udgangspunkt i regularisering er der foreslået en metode til at spore den optimale løsning fra et dårligt begyndelsesgæt på parametrene. Afhandlingens hovedresultat er en metode bestående af PARFIT programmet samt en systematisk fremgangsmåde til håndtering af en række af de problemer, der ofte opstår i forbindelse med dynamisk parameterestimering. Metodens anvendelighed er demonstreret på et erfaringsmæssigt vanskeligt problem hentet fra kemisk reaktionsteknik.

Contents

Progress Report

Final Report

Appendices

Literature Review

1

Introduction

Parameter estimation arises in many different areas of engineering, where mathematical models are used to describe real life phenomena and experiments are performed to validate these models. Advantages of mathematical models include optimization of design and production and the ability to analyze and understand system behaviour subject to conditions that are not readily handled by experiments. Often the models contain a number of parameters that cannot be measured directly or calculated by applying established laws of nature, and therefore must be estimated from experimental data. The basic concept is to determine these parameters such that the differences between the experimental data and the values predicted by the model are minimal in some sense: The predicted, theoretical values should fit the measurements. The choice of fitness criterion depends on the knowledge and the assumptions about the measurement errors.

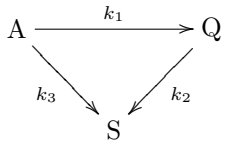
This thesis addresses the problem of estimating parameters in dynamical models, especially those described by ordinary differential equations (ODEs) or differential algebraic equations (DAEs). Methods tailored for partial differential equation (PDE) models are not discussed, but often these models can be reduced to a set of ODEs, which allows the use of techniques developed for ODEs. Estimating parameters in dynamical models is computationally intensive, since it requires the repeated (numerical) solution of the underlying set of differential equations. Efficient and robust methods for solving this problem are important for the development and improvement of process models. For example, a better knowledge of kinetic rate constants in the modelling of chemical reactions can help in choosing operating conditions that favor the desired products.

The purpose of this chapter is to briefly outline different approaches to parameter estimation and to discuss the different aspects involved. In Section 1.1 the basic problem is introduced by means of an example, and the motivation for the work presented in this thesis is given. A mathematical formulation of the problem is introduced, and different methods of estimation are discussed. This preliminary section serves to define basic concepts of parameter estimation and especially to provide a short introduction to the statistical aspects, an area which is not emphasized in the remainder of the thesis. The objective of this work is stated in Section 1.2 and, finally, an outline of the contents of the remainder of the thesis is given in Section 1.2.1 in terms of a schedule with intermediate deadlines for the three parts, which constitute this thesis.

1.1 Preliminaries

To set the stage a simple example is considered, which will be used, whenever possible, to illustrate important concepts throughout this thesis:

Example 1.1 (Catalytic Cracking of Gas-Oil)
Numerous reports on this example can be found in the literature, and it has been used by e.g. Tjoa and Biegler (1991) to evaluate parameter estimation software. The overall reaction of catalytic cracking of gas-oil (A) to gasoline (Q) and other products (S) is considered:



A model for the concentration of species A and Q is described by the following equations:

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} -(k_1 + k_3)y_1^2 \\ k_1 y_1^2 - k_2 y_2 \end{bmatrix}, \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{1.1}$$

in which y_1 and y_2 denote the concentration of A and Q respectively, and k_1, k_2 and k_3 denote the rate constants for the three reactions. Given a set of experimental data, the objective is to estimate k_1, k_2 and k_3 such that the differences between the model predictions and the experimental values are minimized. Since this example only serves to illustrate important concepts, simulated values will be used instead of experimental data. ■

1.1.1 Motivation

The problem of estimating parameters in a model from information about the states is often referred to as an *inverse problem*. Parameter estimation is a part of a larger framework, which includes model formulation and model validation. A typical modelling cycle for the iterative process of building and validating mathematical models is illustrated in Figure 1.1. Elements shown in grey constitute tasks, whereas elements shown in white constitute items that serve as input or output to the individual tasks of the framework. It is assumed that the modelling is based on first engineering principles, where the models are constructed from prior physical knowledge. The information obtained from experiments is then used to estimate the unknown parameters of the model.

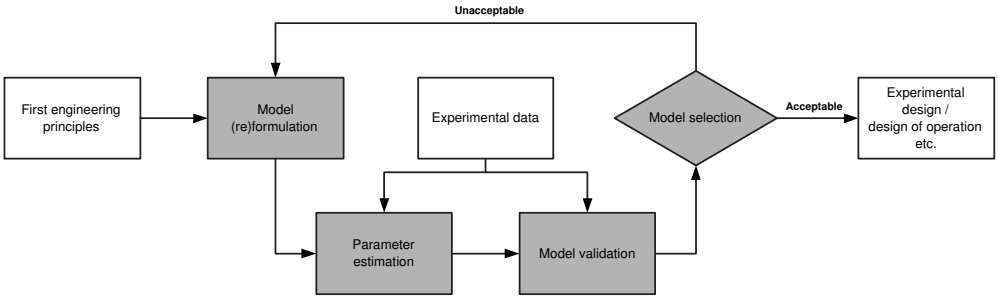


Figure 1.1. Modelling cycle based on first engineering principles.

An alternative strategy is to use data-driven models where both the structure and the parameters of a model are identified from experimental data. These models lack a physically meaningful interpretation, since their structure does not reflect the underlying mechanisms of the system. However, depending on the area of application, data-driven models are sometimes the preferred choice, because systematic tools exist for their development, which renders the modelling process less time-consuming, or simply because the nature of the system is not properly understood. A third possible strategy is to use so-called *grey-box* models which combine first engineering principles modelling with data-driven modelling.

Model formulation, model validation and model selection is a systematic process that eventually leads to the recommendation of one model or a set of models that are:

1. Consistent with the experimental data.
2. In accordance with well established facts concerning the physical process.
3. Not unnecessarily complex.

This process is closely related to experimental design. Once an acceptable model is chosen, advice with respect to additional experiments can be given. Since experiments are often expensive and time-consuming, the insight offered by the model is valuable. Also, the data obtained from the additional experiments can be used to improve the estimates of the model parameters. In fact, the model can be used to devise a set of experiments that yield a parameter estimation with maximum statistical quality, e.g. smallest possible confidence regions for the parameters. This area, known as optimum experimental design, was studied by Bauer *et al.* (2000). Using this approach, the experimental design phase is a more integral part of the overall modelling cycle than suggested in Figure 1.1. The knowledge obtained from statistical tests is used directly to design new experiments providing data that, when used for parameter estimation, result in parameter estimates with improved statistical qualities.

This thesis addresses the part of the overall framework concerned with parameter estimation. Much research has already been carried out to estimate unknown parameters by fitting a numerical solution to a set of experimental data. Some publications emphasize the statistical aspects (Bard, 1974; Seber and Wild, 2003; Stortelder, 1998), while others focus more on the practical implementation and numerical aspects of optimization and solution of differential equation models (Schittkowski, 2002). The latter part is also emphasized in this thesis. Essentially, two components are needed to solve the parameter estimation problem in dynamical systems: An optimization algorithm and a differential equation solver. Although extensive research has been conducted for many years in the areas of optimization and solution of differential equations, unanswered questions and possible improvements still remain concerning the particular interaction between optimizer and equation solver encountered in parameter estimation. Most of the computation time (more than 80% according to Stortelder (1998)) in parameter estimation is used solving the differential equations and generating gradient information for the optimization

algorithm. An efficient solution of the differential equations is therefore crucial to the performance of the overall algorithm.

1.1.2 Mathematical Formulation

The mathematical models considered are assumed to be described by a system of DAEs:

$$\boldsymbol{D}\dot{\boldsymbol{y}} = \boldsymbol{f}(t, \boldsymbol{y}, \boldsymbol{\theta}), \qquad \boldsymbol{y}(t_0, \boldsymbol{\theta}) = \boldsymbol{y}_0(\boldsymbol{\theta}) \tag{1.2}$$

in which $\boldsymbol{\theta} \in \mathbb{R}^{n_p}$ is a vector of unknown parameters and $\boldsymbol{y} \in \mathbb{R}^n$ is a state vector depending on t and $\boldsymbol{\theta}$. $\boldsymbol{f}$ is, in general, a nonlinear function that maps $\mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^{n_p}$ into $\mathbb{R}^n$. $\boldsymbol{D}$ is assumed to be a constant $n \times n$ diagonal matrix with $d_{ii} = 1$ if the ith equation is a differential equation and $d_{ii} = 0$ if the ith equation is algebraic. The above notation for the DAE system is used for convenience to illustrate the basic setup of the parameter estimation problem. Later, when solution strategies are discussed, modifications to this notation are made depending on the specific method considered.

In order to estimate the unknown parameters, a number of measurements are required for the process under consideration. Adopting the notation of Stortelder (1998), each measurement is characterized by a triple:

$$(c_i, t_i, \tilde{y}_i), \qquad i = 1, \ldots, m \tag{1.3}$$

in which c_i indicates which component of the state vector $\boldsymbol{y}$ that has been measured, t_i is the time of the measurement and $\tilde{y}_i$ is the measured value. m denotes the total number of measurements. The solution of the model equations (1.2) for the c_ith component at time t_i, which corresponds to the ith measurement, is denoted by $y_{c_i}(t_i, \boldsymbol{\theta})$. With this notation the ith residual is defined as:

$$r_i(\boldsymbol{\theta}) = y_{c_i}(t_i, \boldsymbol{\theta}) - \tilde{y}_i \tag{1.4}$$

This is a simplified statement, assuming that components of the state vector can be measured directly. More generally, the measurements are associated with the states through the measurement equation. When the true parameter vector $\boldsymbol{\theta}^*$ [1] is used, this equation becomes:

$$\tilde{y}_i = h(t_i, y_{c_i}(t_i, \boldsymbol{\theta}^*), \boldsymbol{\theta}^*) + \varepsilon_i \tag{1.5}$$

in which ε_i denotes the measurement error associated with the ith measurement.

1.1.3 Methods of Estimation

1.1.3.1 Least Squares

The method of estimation depends on the assumptions and knowledge about the measurement errors. One of the most widely used methods of estimation is

[1] In most statistical literature $\boldsymbol{\theta}^*$ is used to denote the "true" value of $\boldsymbol{\theta}$ and $\hat{\boldsymbol{\theta}}$ is used to denote the estimated value of $\boldsymbol{\theta}^*$. In optimization literature, however, an '$*$' indicates an optimal solution. In this thesis, the use of $\boldsymbol{\theta}^*$ as the true parameter vector is limited to this section on statistical aspects, whereas the notation from optimization literature will be used in subsequent chapters.

least squares (LS). In its simplest form the parameters are estimated such that the sum of squared residuals:

$$f(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m} r_i^2(\boldsymbol{\theta}) \tag{1.6}$$

is minimal. The function $f : \mathbb{R}^{n_p} \mapsto \mathbb{R}$ denotes the least squares *objective function*. The factor $\frac{1}{2}$ is introduced for convenience to avoid a factor of 2 when taking the derivative of f. As will be discussed later, the least squares criterion can be regarded as a *maximum likelihood* (ML) criterion, if certain assumptions about the distribution of measurement errors are made. This shows that, provided these assumptions hold, the least squares estimates possess optimal statistical properties.

The fitting criterion (1.6) is also referred to as ordinary least squares (OLS). This criterion is often unsatisfactory for the following reasons (Bard, 1974):

- The measured quantities may have different physical dimensions, or may be measured on different scales. For example, some of the measurements in the measurement vector $\tilde{\boldsymbol{y}}$ may represent concentrations of a chemical species, expressed in mole fractions and falling into the range zero to one. Other measurements, however, may be temperature measured in Kelvin with values in a much higher range. Fitting an OLS criterion will most likely result in the temperature residuals dominating those of the mole fractions, and any information contained in the latter will be lost.
- Some measurements may be known to be less reliable than others. Thus, a tool is needed to make sure that the parameter estimates are less influenced by these measurements relative to the more accurate ones.

The solution to both of these problems is to introduce weight factors into the objective function resulting in a *weighted least squares* (WLS) criterion:

$$f(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m} w_i^2 r_i^2(\boldsymbol{\theta}) \tag{1.7}$$

The weights are chosen small, if the measurements are unreliable or measured on a large scale and large if the opposite is true. An implicit assumption of OLS and WLS is that the errors are only present in the dependent variables. That is, the independent variable (often time) is assumed to be known without error. An approach considering errors in all variables is *total least squares* (TLS), also referred to as orthogonal distance regression (Stortelder, 1998). If the measurement errors in time are denoted ξ_i $(i = 1, \ldots, m)$, the measured time is given by:

$$\tilde{t}_i = t_i + \xi_i \tag{1.8}$$

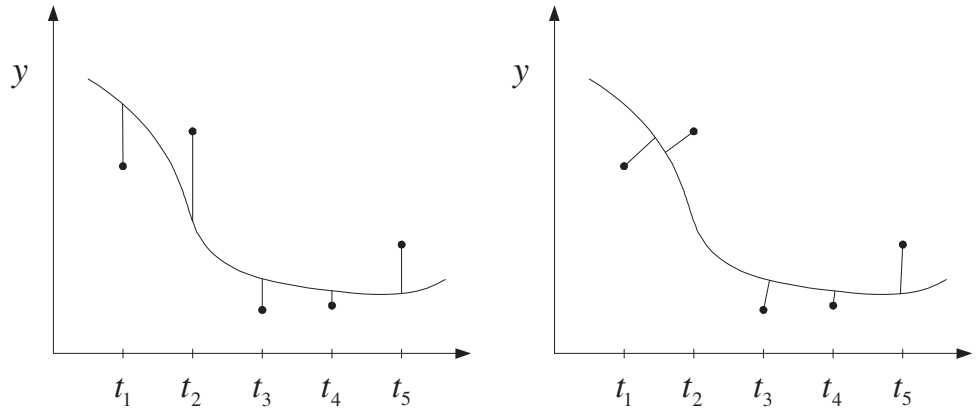
The residuals related to the independent variables are denoted δ_i. Thus, the overall residuals between the measurements and the theoretical values now depend on both $\boldsymbol{\theta}$ and $\boldsymbol{\delta}$:

$$r_i(\boldsymbol{\theta}, \delta_i) = y_{c_i}(t_i + \delta_i, \boldsymbol{\theta}) - \tilde{y}_i \tag{1.9}$$

This leads to the following TLS fitting criterion:

$$f(\boldsymbol{\theta}, \boldsymbol{\delta}) = \frac{1}{2} \sum_{i=1}^{m} w_i^2 r_i^2(\boldsymbol{\theta}, \delta_i) + d_i^2 \delta_i^2 \tag{1.10}$$

in which d_i denotes the weight associated with the ith residual of the independent variable. If the weights in (1.10) are chosen equal, then minimization of the TLS criterion corresponds to minimizing the orthogonal distance between the measurements and the curve $\boldsymbol{y}(t, \boldsymbol{\theta})$, hence the name orthogonal distance regression. The difference between TLS and OLS, in which the vertical distance is minimized, is illustrated in Figure 1.2.



(a) Ordinary least squares.

(b) Total least squares.

Figure 1.2. Graphical comparison between ordinary least squares and total least squares. The OLS estimate is based on minimizing the vertical distance between the measurements and the curve, whereas the TLS estimate minimizes the distance orthogonal to the curve.

1.1.3.2 Maximum Likelihood

The maximum likelihood estimates are derived from the probability density function of the measurement errors. Under certain assumptions these estimates coincide with the OLS or WLS estimates, which will be discussed in the following. In this section the errors are assumed to be present only in the dependent variables.

The measurement errors ε_i are assumed mutually independent and normally distributed with zero mean and variance σ^2, i.e. $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$. The covariance matrix for the vector of measurement errors is:

$$\boldsymbol{V} = E\{\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T\} = \sigma^2 \boldsymbol{I}_m \tag{1.11}$$

where E denotes the expectation operator. The residuals and the measurement errors coincide when the true parameter vector $\boldsymbol{\theta}^*$ is used, c.f. (1.4) and (1.5).

Therefore, if the residuals are assumed to give an adequate representation of the measurement errors, the probability density function for the assumed structure of the measurement errors is given by (Seber and Wild, 2003):

$$\begin{aligned} p(\tilde{\boldsymbol{y}}|\boldsymbol{\theta}) &= \left(2\pi\sigma^2\right)^{-m/2} \exp\left(\frac{-\sum_{i=1}^{m} r_i^2(\boldsymbol{\theta})}{2\sigma^2}\right) \\ &= \left(2\pi\sigma^2\right)^{-m/2} \exp\left(-\frac{1}{2}\boldsymbol{r}(\boldsymbol{\theta})^T \boldsymbol{V}^{-1} \boldsymbol{r}(\boldsymbol{\theta})\right) \end{aligned} \tag{1.12}$$

in which $\boldsymbol{r} : \mathbb{R}^{n_p} \mapsto \mathbb{R}^m$ denotes a vector of assembled residuals. The maximum likelihood estimate is the value of $\boldsymbol{\theta}$ that maximizes the probability density function, i.e. the most likely $\boldsymbol{\theta}$ for a given data set. The *likelihood function* is defined as:

$$L(\boldsymbol{\theta}) \equiv p(\tilde{\boldsymbol{y}}|\boldsymbol{\theta}) \tag{1.13}$$

Taking the logarithm of the likelihood function yields:

$$\ln L(\boldsymbol{\theta}) = -\frac{m}{2} \ln\left(2\pi\sigma^2\right) - \frac{1}{2}\boldsymbol{r}(\boldsymbol{\theta})^T \boldsymbol{V}^{-1} \boldsymbol{r}(\boldsymbol{\theta}) \tag{1.14}$$

The likelihood function reaches its maximum when the latter term in (1.14) is minimal, which corresponds to the minimum of the OLS objective function (1.6). Thus, in the case of independent and identically distributed measurement errors from a normal distribution, the maximum likelihood estimate of $\boldsymbol{\theta}$ coincides with the OLS estimate.

A connection between maximum likelihood and WLS also exists. If the measurement errors are assumed independent and normally distributed, but with non-constant variances, $\varepsilon_i \sim \mathcal{N}(0, \sigma_i)$, the corresponding likelihood function is (Seber and Wild, 2003):

$$L(\boldsymbol{\theta}) = \frac{(2\pi)^{-m/2}}{\sqrt{\det(\boldsymbol{V})}} \exp\left(-\frac{1}{2}\boldsymbol{r}(\boldsymbol{\theta})^T \boldsymbol{V}^{-1} \boldsymbol{r}(\boldsymbol{\theta})\right) \tag{1.15}$$

in which $\boldsymbol{V}$ now has non-constant diagonal elements $V_{ii} = \sigma_i^2$. Again, taking the logarithm yields:

$$\ln L(\boldsymbol{\theta}) = -\frac{m}{2} \ln(2\pi) - \sum_{i=1}^{m} \ln(\sigma_i) - \frac{1}{2} \sum_{i=1}^{m} \left(\frac{r_i(\boldsymbol{\theta})}{\sigma_i}\right)^2 \tag{1.16}$$

Comparing (1.7) and (1.16) shows that the maximum likelihood estimates coincide with the WLS estimates, if the weights in (1.7) are chosen proportional to the reciprocal of the standard deviations:

$$w_i \propto \frac{1}{\sigma_i} \tag{1.17}$$

This establishes the connection between maximum likelihood and WLS. However, in most practical situations the standard deviations of the measurement errors are unknown. The standard deviations can be estimated along with the unknown parameters. If the measurement errors are assumed independent, the covariance matrix is diagonal. In the more general case, the covariance matrix

is a full matrix. To limit the number of unknown elements to be estimated, assumptions such as independence of errors at different points in time and zero expectation of errors are often made. Details on maximum likelihood estimation with unknown covariance can be found in e.g Bard (1974). These cases may be regarded as WLS problems with unknown weights.

As mentioned earlier, the choice of estimation method depends on the assumptions and knowledge about the measurement errors. The conditions required to apply a maximum likelihood criterion are often not present, e.g. an incorrect model structure results in systematic errors. On the other hand, a least squares criterion may lack a sound statistical interpretation, unless assumptions similar to maximum likelihood with known variances are made, in which case the two types of estimation coincide.

1.1.4 Statistical Background

An important part of the mathematical modelling cycle is validation of the proposed model. Various statistical tests and residual analysis tools are available including marginal and simultaneous tests for parameter significance and correlation analysis of residuals computed from validation data sets, i.e. data sets that were not used for parameter estimation. Here, a brief discussion is given on estimation of the covariance and calculation of confidence regions for the parameter estimates.

In this section the errors in the measurements are assumed independent and normally distributed with zero expectation and known variance σ^2. It is important to obtain an estimate of the correlation between the estimated parameters. Correlation between parameters indicates that insufficient information is available in the data to estimate the model parameters uniquely, so that either the model structure should be reconsidered or further experiments performed. If $\hat{\boldsymbol{\theta}}$ denotes the estimated parameter vector, then the covariance matrix for the parameters can be approximated as follows (Seber and Wild, 2003):

$$\begin{aligned} \operatorname{cov}\left(\hat{\boldsymbol{\theta}}\right) &= E\left\{\left(\boldsymbol{\theta}^* - \hat{\boldsymbol{\theta}}\right)\left(\boldsymbol{\theta}^* - \hat{\boldsymbol{\theta}}\right)^T\right\} \\ &\approx \hat{\sigma}^2\left(\boldsymbol{H}(\hat{\boldsymbol{\theta}})\right)^{-1} \end{aligned} \tag{1.18}$$

in which $\boldsymbol{H}(\hat{\boldsymbol{\theta}})$ denotes the Hessian matrix of the objective function evaluated at $\hat{\boldsymbol{\theta}}$. $\hat{\sigma}^2$ is an estimator for σ^2:

$$\hat{\sigma}^2 = \frac{f(\hat{\boldsymbol{\theta}})}{m - n_p} \tag{1.19}$$

The covariance estimate (1.18) can be derived from a linearization of the objective function around $\boldsymbol{\theta}^*$. This will be clear when solution strategies are discussed in the next chapter. An often used approximation to the Hessian matrix is the so-called Gauss-Newton approximation (see Section 2.1.3). Using this approximation, (1.18) is rewritten as:

$$\operatorname{cov}\left(\hat{\boldsymbol{\theta}}\right) \approx \hat{\sigma}^2\left(\boldsymbol{J}(\hat{\boldsymbol{\theta}})^T\boldsymbol{J}(\hat{\boldsymbol{\theta}})\right)^{-1} \tag{1.20}$$

Details on the properties of the covariance estimate are found in Seber and Wild (2003). The only other statistical element considered here is an approximate confidence interval for the parameter estimates. Assuming that the estimator for the parameters is normally distributed:

$$\hat{\boldsymbol{\theta}} \sim \mathcal{N}(\boldsymbol{\theta}^*, \sigma^2 \boldsymbol{C}), \quad \boldsymbol{C} = \left(\boldsymbol{H}(\hat{\boldsymbol{\theta}}) \right)^{-1} \tag{1.21}$$

a $100(1-\alpha)\%$ marginal confidence interval for the ith parameter is given by:

$$\hat{\theta}_i \pm t_{m-n_p}^{\alpha/2} \hat{\sigma} \sqrt{\boldsymbol{C}_{ii}} \tag{1.22}$$

in which $t_{m-n_p}^{\alpha/2}$ denotes a quantile of the t-distribution with $m - n_p$ degrees of freedom. In the case of high correlation between the parameters the marginal intervals might be misleading, since the correlation is not taken into account. Approaches exist for constructing simultaneous intervals (Seber and Wild, 2003).

1.2 Objective

The objective of this thesis is to assess different methods for parameter estimation in nonlinear dynamical systems. More specifically, the objective is to study the numerical aspects involved and to build a framework, consisting of tools and guidelines, capable of handling the many difficulties often encountered in the practical solution of parameter estimation problems. As indicated previously, a major task in solving parameter estimation problems in dynamical systems is the repeated solution of the underlying set of differential equations and, in particular, the generation of gradient information for the optimization algorithm. Therefore, one of the goals of this thesis is to study, develop and test different methods for gradient generation with the ultimate goal of improving the efficiency and robustness of parameter estimation algorithms.

1.2.1 Schedule

The project is divided into three parts. Below is a preliminary schedule with intermediate deadlines for handing in reports and a brief outline of the individual parts:

- **Literature Review** – With respect to contents, the first part of the thesis is devoted to a review of the relevant literature. Important terminology is defined, and basic principles of parameter estimation are discussed. Numerical methods for optimization and solution of differential equation models are emphasized.
  Deadline: March 8th 2004.

- **Progress Report** – Based on the literature review, initial parameter estimation case studies are performed, starting with small scale problems with simulated data and continuing to problems with real data. Numerical experiments with different optimization algorithms and differential equation solvers are performed. Advantages and disadvantages of the

individual approaches are discussed.
Deadline: May 6th 2004.

- **Final Report** – The work from the progress report is continued. Further case studies are performed emphasizing more advanced problems. Effort is put into developing an efficient, robust and flexible parameter estimation tool. The overall conclusions are presented and possible improvements to existing parameter estimation approaches are suggested.
  Deadline: June 30th 2004.

1.3 Notation

Since parameter estimation includes aspects of statistical analysis as well as numerical optimization and solution of differential equations, the notation used in the literature varies depending on which elements are emphasized in the individual publications. The notation used in this thesis is summarized in the tables on pages 133–134.

2

Numerical Methods for Optimization

This chapter focuses on the methods for numerical optimization required to solve parameter estimation problems. With some preliminary comments on optimality conditions, different optimization strategies are discussed both for the unconstrained and the constrained case. Methods specifically tailored for nonlinear least squares problems are emphasized. The methods are treated in a general context, not considering how to solve the underlying set of differential equations, which is the topic of Chapter 3.

2.1 Unconstrained Problems

In order to discuss properties of different optimization strategies, some basic definitions and theorems are needed. All proofs, which can be found in standard textbooks on optimization such as Dennis and Schnabel (1983) or Nocedal and Wright (1999), are omitted from the discussion. The parameter estimation problem can be formulated as:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \tag{2.1}$$

in which $\boldsymbol{\theta} \in \mathbb{R}^{n_p}$ is a vector of parameters, and $f : \mathbb{R}^{n_p} \mapsto \mathbb{R}$ is the objective function. $\boldsymbol{\theta}^*$ is a vector that minimizes f. f is in general some nonlinear function of the problem parameters, for example the least squares function defined in Section 1.1. Thus, the problem is to find $\boldsymbol{\theta}^*$. Ideally, $\boldsymbol{\theta}^*$ would be the *global minimum* of f, but often the objective function has several minimizers, so that $\boldsymbol{\theta}^*$ might only be a *local minimizer*. The optimization methods discussed here all find a local minimum, and it is beyond the scope of this thesis to discuss global methods, which requires more advanced techniques. A local minimizer is defined as (Nocedal and Wright, 1999):

Definition 2.1 (Local Minimizer)
A point $\boldsymbol{\theta}^*$ is a local minimizer if there exists a neighborhood $\mathcal{N}$ of $\boldsymbol{\theta}^*$ such that $f(\boldsymbol{\theta}^*) \leq f(\boldsymbol{\theta})$ for all $\boldsymbol{\theta} \in \mathcal{N}$.

A neighborhood of a point is simply defined as a sufficiently small open set that contains $\boldsymbol{\theta}^*$.

2.1.1 Optimality Conditions

In order to distinguish local minimizers from other points some optimality conditions are needed. If f is twice continuously differentiable, then the following Taylor expansion for f applies:

$$f(\boldsymbol{\theta}+\boldsymbol{h}) = f(\boldsymbol{\theta}) + \boldsymbol{h}^T\boldsymbol{g}(\boldsymbol{\theta}) + \frac{1}{2}\boldsymbol{h}^T\boldsymbol{H}(\boldsymbol{\theta})\boldsymbol{h} + \mathcal{O}\left(\|\boldsymbol{h}\|^3\right) \tag{2.2a}$$

where the gradient $\boldsymbol{g}$ and the Hessian matrix $\boldsymbol{H}$ are defined as:

$$\boldsymbol{g} = \nabla f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \vdots \\ \frac{\partial f}{\partial \theta_{n_p}} \end{bmatrix} \tag{2.2b}$$

$$\boldsymbol{H} = \nabla^2 f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial^2 f}{\partial \theta_i \partial \theta_j} \end{bmatrix} \tag{2.2c}$$

If $\boldsymbol{\theta}$ is a local minimizer, then it is impossible to find an $\boldsymbol{h}$ so that $f(\boldsymbol{\theta}+\boldsymbol{h}) < f(\boldsymbol{\theta})$ with $\|\boldsymbol{h}\|$ small enough, which leads to the *first order necessary condition* for a local minimum:

Theorem 2.1 (First Order Necessary Condition)
If $\boldsymbol{\theta}^*$ is a local minimizer for f, then $\boldsymbol{g}(\boldsymbol{\theta}^*) = \boldsymbol{0}$.

Among the points that satisfy $\boldsymbol{g}(\boldsymbol{\theta}^*) = \boldsymbol{0}$ (stationary points) are also local maximizers and *saddle points*. To distinguish the minimizers, it is necessary to include conditions based on second order information. From the Taylor expansion (2.2a) it is seen that the second term vanishes for all stationary points, which implies that a necessary condition for a local minimum is:

Theorem 2.2 (Second Order Necessary Condition)
If $\boldsymbol{\theta}^*$ is a local minimizer for f, then $\boldsymbol{g}(\boldsymbol{\theta}^*) = \boldsymbol{0}$ and $\boldsymbol{H}(\boldsymbol{\theta}^*)$ is positive semidefinite.

Again, $\|\boldsymbol{h}\|$ is assumed small enough so that the higher order terms in the Taylor expansion are negligible. Positive semidefiniteness implies by definition that $\boldsymbol{h}^T\boldsymbol{H}\boldsymbol{h} \geq \boldsymbol{0}$ for all $\boldsymbol{h} \neq \boldsymbol{0}$. The last condition presented here is the second order sufficient condition for optimality, which guarantees that $\boldsymbol{\theta}^*$ is a local minimum provided some conditions on the derivatives of f:

Theorem 2.3 (Second Order Sufficient Condition)
If $\boldsymbol{g}(\boldsymbol{\theta}^*) = \boldsymbol{0}$ and $\boldsymbol{H}(\boldsymbol{\theta}^*)$ is positive definite, then $\boldsymbol{\theta}^*$ is a local minimizer for f.

These conditions are important when constructing numerical methods to find the local minimizers, which will be evident in the subsequent sections.

2.1.2 Rate of Convergence

The methods presented here to solve the parameter estimation problem are all iterative methods that produce a sequence of iterates $\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots$ that, hopefully, converges to a local minimizer, i.e. $\boldsymbol{\theta}_k \to \boldsymbol{\theta}^*$ for $k \to \infty$. An important

property of a method for optimization is the rate at which this sequence of iterates converges to the minimizer. The convergence is often divided into a *global part* and a *local part* (Frandsen *et al.*, 1999), which represent the parts of the iteration far from and close to the local minimizer, respectively. Normally, the highest rate of convergence is achieved in the local part of the iteration. Some methods do not even converge if the iteration starts in a position too far from the minimizer as illustrated in the example below:

Example 2.1 (Newton's Method on a Nonlinear Equation)
Consider the simple problem of finding the solution to the following scalar equation:

$$f(\theta) = \arctan\theta = 0 \tag{2.3}$$

The unique solution is $\theta^* = 0$, but if Newton's method is applied with a "bad choice" of starting point, the sequence of iterates will diverge as illustrated on Figure 2.1. Newton's method is not globally convergent for this problem.

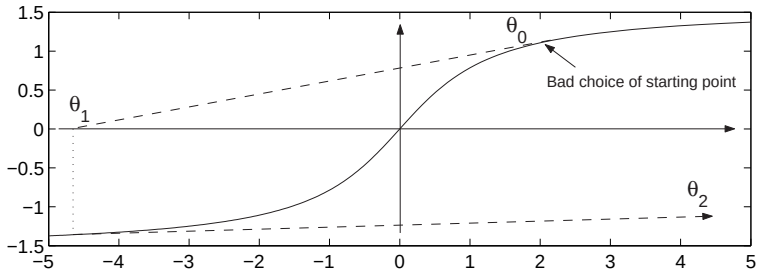


Figure 2.1. Newton's method applied to (2.3) with a starting point too far from the solution.

■

In what follows, three different rates of convergence are encountered, and these are defined below (see Dennis and Schnabel (1983)):

Definition 2.2 (Rates of Convergence)
If there exists $K \geq 0$ *and* $\alpha \in [0, 1[$ *such that for all* $k \geq K$*:*

$$\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}^*\| \leq \alpha \|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\| \tag{2.4a}$$

then the sequence $\{\boldsymbol{\theta}_k\}$ *is said to converge linearly to* $\boldsymbol{\theta}^*$*. If there exist* $K \geq 0$ *and* $\alpha > 0$ *such that for all* $k \geq K$*:*

$$\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}^*\| \leq \alpha \|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|^2 \tag{2.4b}$$

then the sequence of iterates converges quadratically to $\boldsymbol{\theta}^*$*. Finally, if there exists a sequence of scalars* $\{\alpha_k\}$ *converging to zero such that for all* $k \geq 0$*:*

$$\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}^*\| \leq \alpha_k \|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\| \tag{2.4c}$$

then the convergence is said to be superlinear.

To be precise, the form of convergence defined above is the so-called q-order of convergence, where the prefix q stands for "quotient" and is used to differentiate

from r-orders ("root") of convergence, which is a weaker type of convergence (Dennis and Schnabel, 1983). When assessing different optimization methods, it is desirable to have for example linear convergence in the global part, but at least superlinear convergence in the local part. Newton's method is locally quadratic convergent, but not in general globally convergent as seen in Example 2.1.

2.1.3 Nonlinear Least Squares

Many of the methods tailored for nonlinear least squares are based on modifications of Newton's method. They differ in the way the Hessian matrix is approximated. As introduced in Section 1.1 the least squares problem has the following special structure:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m} r_i^2(\boldsymbol{\theta}), \quad \text{where } r_i : \mathbb{R}^{n_p} \mapsto \mathbb{R} \text{ and } m \geq n_p \tag{2.5}$$

in which r_i are the residuals. If the individual residuals are assembled in a vector $\boldsymbol{r} : \mathbb{R}^{n_p} \mapsto \mathbb{R}^m$ defined by:

$$\boldsymbol{r}(\boldsymbol{\theta}) = \begin{bmatrix} r_1(\boldsymbol{\theta}) \\ r_2(\boldsymbol{\theta}) \\ \vdots \\ r_m(\boldsymbol{\theta}) \end{bmatrix} \tag{2.6}$$

then the objective function in (2.5) is rewritten as:

$$f(\boldsymbol{\theta}) = \frac{1}{2}\left\|\boldsymbol{r}(\boldsymbol{\theta})\right\|^2 = \frac{1}{2}\boldsymbol{r}(\boldsymbol{\theta})^T\boldsymbol{r}(\boldsymbol{\theta}) \tag{2.7}$$

A simple way to solve the least squares problem is to use Newton's method, which relies on a quadratic approximation to the objective function. Provided that f is sufficiently smooth, then the following Taylor approximation is valid:

$$f(\boldsymbol{\theta} + \boldsymbol{h}) \simeq f(\boldsymbol{\theta}) + \boldsymbol{h}^T\boldsymbol{g}(\boldsymbol{\theta}) + \frac{1}{2}\boldsymbol{h}^T\boldsymbol{H}(\boldsymbol{\theta})\boldsymbol{h} \tag{2.8}$$

In each iterate this quadratic model is minimized, and the minimizer $\boldsymbol{\theta}^*$ to the least squares problem is approximated by the solution to:

$$\boldsymbol{g}(\boldsymbol{\theta}) + \boldsymbol{H}(\boldsymbol{\theta})\boldsymbol{h} = \boldsymbol{0} \tag{2.9}$$

which states that the gradient of the quadratic model is equal to zero. The quadratic model has a unique minimum when $\boldsymbol{H}$ is positive definite. Each iteration in Newton's method requires the solution of the linear system (2.9), which again requires the computation of the gradient and the Hessian matrix. For the least squares objective function, the gradient and Hessian matrices are:

$$\boldsymbol{g} = \nabla f(\boldsymbol{\theta}) = \sum_{i=1}^{m} r_j(\boldsymbol{\theta})\nabla r_i(\boldsymbol{\theta}) = \boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{r}(\boldsymbol{\theta}) \tag{2.10a}$$

$$\begin{aligned} \boldsymbol{H} = \nabla^2 f(\boldsymbol{\theta}) &= \sum_{i=1}^{m} \nabla r_i(\boldsymbol{\theta})\nabla r_i(\boldsymbol{\theta})^T + \sum_{i=1}^{m} r_i(\boldsymbol{\theta})\nabla^2 r_i(\boldsymbol{\theta}) \\ &= \boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{J}(\boldsymbol{\theta}) + \sum_{i=1}^{m} r_i(\boldsymbol{\theta})\nabla^2 r_i(\boldsymbol{\theta}) \end{aligned} \tag{2.10b}$$

where $\boldsymbol{J}(\boldsymbol{\theta})$ denotes the $m \times n_p$ Jacobian matrix of $\boldsymbol{r}$:

$$\boldsymbol{J}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial r_1}{\partial \theta_1} & \cdots & \frac{\partial r_1}{\partial \theta_{n_p}} \\ \vdots & & \vdots \\ \frac{\partial r_m}{\partial \theta_1} & \cdots & \frac{\partial r_m}{\partial \theta_{n_p}} \end{bmatrix} \tag{2.11}$$

It appears that the first part of the Hessian matrix consists of first order partial derivatives. This observation leads to an approximation forming the basis for the *Gauss-Newton* and the *Levenberg-Marquardt* algorithms, which are discussed in the next sections. Calculation of first and second order derivatives of the objective function often constitutes a major part of the work required in optimization. This is especially true in the case of dynamical systems, where each gradient evaluation is itself a complex procedure requiring the solution of a set of differential equations. Thus, the incentive to use an efficient method that exploits the special structure in the least squares problem is particularly great in the context of parameter estimation in dynamical systems. The need for second order derivatives of the residual functions r_i in Newton's method renders it less attractive for practical applications.

2.1.3.1 Line Search versus Trust Region

Before turning to different modifications of Newton's method the notions of *line search* and *trust region* are briefly explained. In a line search approach each iterate in the optimization is composed of two parts:

1. Compute a descent direction, i.e. a direction where the objective function value decreases. An example is the direction of *steepest descent* $\boldsymbol{h} = -\nabla f(\boldsymbol{\theta})$.

2. Compute the length of the step in the direction chosen such that a "sufficient" decrease in f is gained. The overall step is then computed as $\alpha \boldsymbol{h}$, where α is the steplength parameter determined from the line search.

Given a descent direction, the line search algorithm searches along this direction to determine a steplength that provides a sufficient decrease in f. The situation is depicted on Figure 2.2. The figure illustrates a possible variation of f along the direction $\boldsymbol{h}$ as expressed by:

$$\phi(\alpha) = f(\boldsymbol{\theta} + \alpha \boldsymbol{h}), \quad \text{with fixed } \boldsymbol{\theta} \text{ and } \boldsymbol{h} \tag{2.12}$$

Different strategies are used to ensure that a sufficient decrease is obtained. First of all, one must guard against α being chosen too large such that f increases. However, α should not be chosen too small either, giving an insufficient decrease in f-value. Al-Baali and Fletcher (1986) present a number of line search algorithms and prove convergence for these. The details are omitted here, but standard algorithms can also be found in Nocedal and Wright (1999) as well as in Frandsen *et al.* (1999). Algorithms seeking the exact minimum of $\phi(\alpha)$ in (2.12) are referred to as *exact line searches*, whereas algorithms that accept a sufficient decrease in f are referred to as *soft line searches*. Exact line

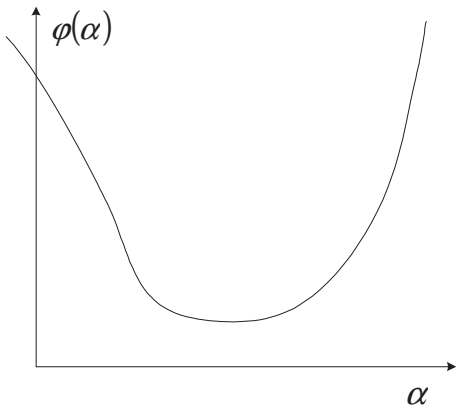


Figure 2.2. Variation of the objective function along the search line (Frandsen *et al.*, 1999).

searches terminate in fewer iterations, but the extra effort spent minimizing $\phi(\alpha)$ in each step makes them overall slower compared to soft line searches (Frandsen *et al.*, 1999).

Unlike the line search methods, the direction and steplength in a trust region approach are chosen simultaneously subject to a distance constraint, i.e. the trust region. The trust region methods rely on a model approximating the objective function in a region around the current iterate. The step is then chosen as a minimizer of the model in the trust region. If a step is not acceptable, the size of the trust region is reduced, and a new minimizer of the model is found. In more mathematical terms the model could for instance be a linear model:

$$f(\boldsymbol{\theta}+\boldsymbol{h}) \simeq q(\boldsymbol{h}), \quad \text{where} \quad q(\boldsymbol{h}) = f(\boldsymbol{\theta}) + \boldsymbol{h}^T \boldsymbol{g}(\boldsymbol{\theta}) \tag{2.13a}$$

or a quadratic model as for Newton's method:

$$f(\boldsymbol{\theta}+\boldsymbol{h}) \simeq q(\boldsymbol{h}), \quad \text{where} \quad q(\boldsymbol{h}) = f(\boldsymbol{\theta}) + \boldsymbol{h}^T \boldsymbol{g}(\boldsymbol{\theta}) + \frac{1}{2}\boldsymbol{h}^T \boldsymbol{H}(\boldsymbol{\theta})\boldsymbol{h} \tag{2.13b}$$

In both cases q is only an adequate approximation to $f(\boldsymbol{\theta}+\boldsymbol{h})$ if $\|\boldsymbol{h}\|$ is sufficiently small. Therefore $\boldsymbol{h}$ is chosen as the minimizer to the following problem:

$$\begin{aligned} &\boldsymbol{h} = \arg\min_{\boldsymbol{h}\in\mathcal{D}} q(\boldsymbol{h}), \quad \text{where} \\ &\mathcal{D} = \{\boldsymbol{h} \mid \|\boldsymbol{h}\| \leq \Delta\}, \quad \Delta > 0 \end{aligned} \tag{2.14}$$

in which $\mathcal{D}$ denotes the trust region defined here as a ball with radius Δ. Other shapes of trust regions are also used, including ellipsoids (Nocedal and Wright, 1999). The size of the trust region is adjusted based on the ratio of the actual reduction in f-value to the predicted reduction:

$$\varrho = \frac{f(\boldsymbol{\theta}) - f(\boldsymbol{\theta}+\boldsymbol{h})}{q(\mathbf{0}) - q(\boldsymbol{h})} \tag{2.15}$$

When ϱ is close to one, $q(\boldsymbol{h})$ is a good approximation to $f(\boldsymbol{\theta}+\boldsymbol{h})$, and the trust region can be expanded. Similarly, ϱ is used to reject steps and reduce the trust region. Whenever the size of the trust region is altered, both the direction and length of a step change, which also distinguishes these methods from line search methods, in which a search is performed along a fixed direction.

2.1.3.2 The Gauss-Newton Method

As indicated previously, there exist alternative methods to Newton's method that avoid calculating the exact second order derivatives of the objective function. In developing more efficient algorithms, it is important to understand the different drawbacks of Newton's method, which are summarized below (Frandsen *et al.*, 1999):

1. The method is not globally convergent for many problems, c.f. Example 2.1.
2. It may converge towards a maximum or saddle point of f.
3. The system of nonlinear equations to be solved in each iteration may be ill-conditioned or singular.
4. The method requires analytic second order derivatives[1] of f.

The Gauss-Newton method addresses the last disadvantage in particular. Instead of using the full Hessian (2.10b), the second order terms are neglected and the Gauss-Newton step is found by solving:

$$\left(\boldsymbol{J}^T\boldsymbol{J}\right)\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{r} \tag{2.16}$$

In this way, only first order derivatives need to be implemented, thereby avoiding the trouble of computing the individual Hessians $\nabla^2 r_i$, $i = 1, 2, \ldots, m$. Normally, the Gauss-Newton method is implemented with a line search, and it can be shown to converge, if the Jacobian $\boldsymbol{J}$ has full rank implying that $\boldsymbol{J}^T\boldsymbol{J}$ is positive definite. The rate of convergence depends on how good the Hessian approximation is. From (2.10b) it is observed that the approximation is reasonable for problems with small residuals (r_i is small) or problems where the residual functions are nearly linear ($\left\|\nabla^2 r_i\right\|$ is small). Superlinear or even locally quadratic convergence is observed for these problems (Nocedal and Wright, 1999). However, large residual problems or highly nonlinear problems will cause slow convergence.

2.1.3.3 The Levenberg-Marquardt Method

A drawback of the Gauss-Newton method is its behaviour when the Jacobian is rank-deficient or nearly rank-deficient such that the Hessian approximation is no longer positive definite. The Levenberg-Marquardt method proposed by Levenberg (1944) and later Marquardt (1963) handles this problem by introducing a damping parameter μ that "interpolates" between the Gauss-Newton direction and the steepest descent direction. This can be illustrated by considering (modified versions of) the two algorithms:

Steepest Descent	**Gauss-Newton**
Solve $\boldsymbol{I}\boldsymbol{h} = \boldsymbol{J}^T\boldsymbol{r}$	$\left(\boldsymbol{J}^T\boldsymbol{J}\right)\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{r}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{h}$	$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{h}$

[1] This is the strict definition of Newton's method adopted from Frandsen *et al.* (1999). If, for example, the analytic derivatives are replaced by difference approximations, the resulting method is regarded as a modified Newton's method.

in which α is determined from a line search. Second order components in the Hessian are still ignored, and the Levenberg-Marquardt method appears when combining the steepest descent and Gauss-Newton methods by adding a multiple of the identity matrix to the Hessian:

Levenberg-Marquardt

Solve $\left(\boldsymbol{J}^T\boldsymbol{J} + \mu\boldsymbol{I}\right)\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{r}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{h}$

The damping parameter improves the performance in the global part of the iteration. When μ is small, which is a good approximation in the final part of the iteration, the Levenberg-Marquardt step is equal to the Gauss-Newton step, so the convergence properties of the two methods in this part are the same. The damping parameter is adjusted based on a measure of the actual decrease in f-value and the predicted decrease similar to (2.15). This establishes the connection to trust region methods, and the Levenberg-Marquardt method is often interpreted as a trust region approach (Nocedal and Wright, 1999). Since μ influences both the direction and the length of a step, no line search is required. Marquardt (1963) proposed a simple heuristic approach to adjusting μ. Algorithms based on this approach can be shown to converge (Osborne, 1976). Substantial improvement was obtained on a number of examples by Nielsen (1999) with a strategy in which μ varies smoothly. Since the Levenberg-Marquardt method, or modifications of it, will be used in examples in the present chapter as well as in later chapters of this thesis, an outline of the method is given in Algorithm 1. μ_0 and the tolerance parameters for the stopping criteria ε_1 and ε_2 are specified by the user (see Nielsen (1999) for details on μ_0).

The Levenberg-Marquardt method is one of the most widely used methods for solving nonlinear least squares problems. In the example below the method is tested on the Gas-Oil cracking problem introduced in Section 1.1. These preliminary experiments are conducted in MATLAB using the algorithms described in Madsen *et al.* (1999). When estimating parameters in larger problems in subsequent parts of this thesis, the computations are carried out in Fortran.

Example 2.2 (Parameter Estimation in Gas-Oil Cracking Model)
The purpose of this example is to illustrate the performance of the Levenberg-Marquardt method as outlined in Algorithm 1. The underlying model equations are solved using the solvers in MATLAB, and the objective function gradient is calculated by solving the corresponding set of sensitivity equations. This important topic will be properly introduced and treated in much more detail in Chapter 3. For now, it suffices to state that the objective function value and gradient are available.

Consider again the cracking of gas-oil. The problem is modified slightly assuming that $k_3 = k_2$ to facilitate the demonstration of the performance of the optimization algorithm. The modified problem is:

$$\begin{bmatrix}\dot{y}_1\\ \dot{y}_2\end{bmatrix} = \begin{bmatrix}-(k_1+k_2)y_1^2\\ k_1y_1^2 - k_2y_2\end{bmatrix}, \quad \begin{bmatrix}y_1(0)\\ y_2(0)\end{bmatrix} = \begin{bmatrix}1\\ 0\end{bmatrix}, \quad \begin{bmatrix}k_1\\ k_2\end{bmatrix} = \begin{bmatrix}12\\ 8\end{bmatrix} \tag{2.18}$$

The parameters k_1 and k_2 are estimated from "measurements" generated by the model at 10 equidistant time points using the parameter values listed above. If $y_j(t_i, \theta)$ and $\tilde{y}_{ij}$ denote the prediction and measurement of component j at time t_i, respectively,

Algorithm 1 The Levenberg-Marquardt Method

Set $\boldsymbol{\theta} = \boldsymbol{\theta}_0$, $\mu = \mu_0$, $\nu = 2$ and $k = 0$
Set $found = false$
Compute the Hessian approximation and the gradient:

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{J}^T(\boldsymbol{\theta})\boldsymbol{J}(\boldsymbol{\theta}) \\ \boldsymbol{g} &= \boldsymbol{J}^T(\boldsymbol{\theta})\boldsymbol{r}(\boldsymbol{\theta}) \end{aligned} \tag{2.17a}$$

while not $found$ **and** $k < k_{max}$ **do**
$k = k + 1$
Solve for $\boldsymbol{h}$:

$$(\boldsymbol{A} + \mu \boldsymbol{I})\, \boldsymbol{h} = -\boldsymbol{g} \tag{2.17b}$$

if $\|\boldsymbol{h}\| \leq \varepsilon_2 \|\boldsymbol{\theta}\|$ **then** $found = true$ **end if**
Compute ϱ similar to expression (2.15):

$$\varrho = \frac{f(\boldsymbol{\theta}) - f(\boldsymbol{\theta} + \boldsymbol{h})}{L(\mathbf{0}) - L(\boldsymbol{h})} \tag{2.17c}$$

in which L is the linear model approximating f.
if $\varrho > 0$ **then**
Update $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{h} \tag{2.17d}$$

and recalculate the Hessian and the gradient:

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{J}^T(\boldsymbol{\theta})\boldsymbol{J}(\boldsymbol{\theta}) \\ \boldsymbol{g} &= \boldsymbol{J}^T(\boldsymbol{\theta})\boldsymbol{r}(\boldsymbol{\theta}) \end{aligned} \tag{2.17e}$$

if $\|\boldsymbol{g}\| \leq \varepsilon_1$ **then** $found = true$ **end if**
Update μ by strategy described in Nielsen (1999):

$$\mu = \mu \cdot \max\left\{\frac{1}{3}, 1 - (2\varrho - 1)^3\right\}, \quad \nu = 2 \tag{2.17f}$$

else

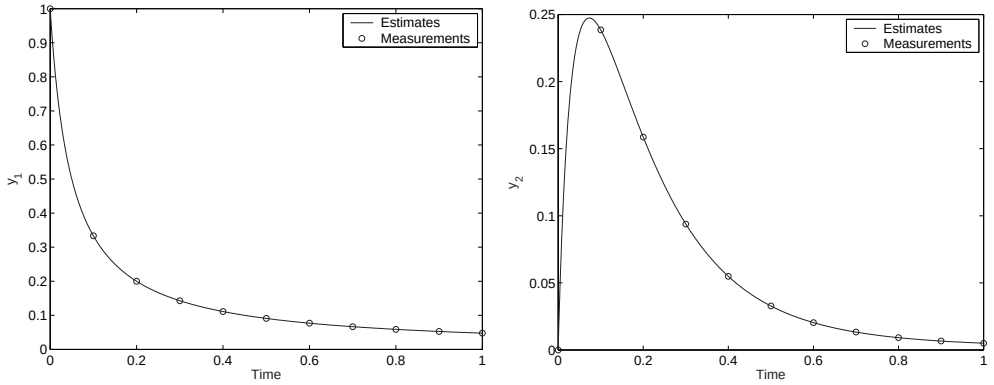
$$\mu = \mu \cdot \nu, \quad \nu = 2 \cdot \nu \tag{2.17g}$$

end if
end while

then the problem may be formulated as:

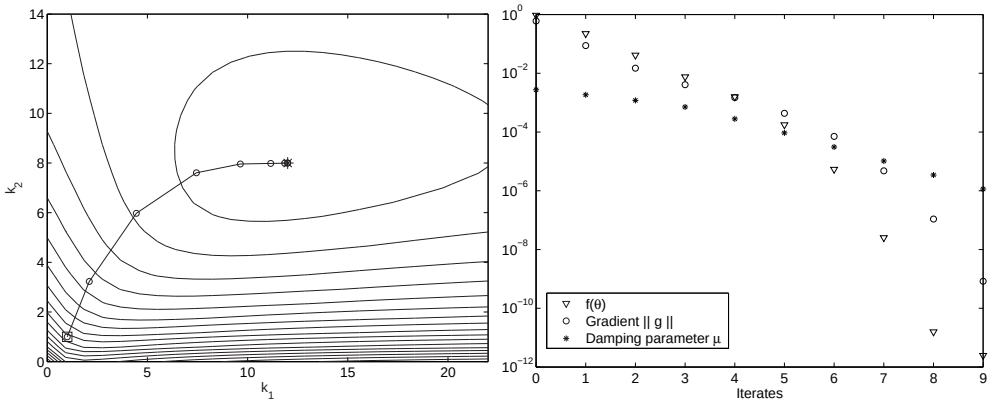
$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{1}{2} \sum_{j=1}^{2} \sum_{i=1}^{10} \left(y_j(t_i, \theta) - \tilde{y}_{ij}\right)^2 \tag{2.19}$$

The tolerance parameters in Algorithm 1 are set to $\varepsilon_1 = \varepsilon_2 = 10^{-8}$ and $k_{max} = 100$. As a starting guess for the parameters $\boldsymbol{\theta}_0 = [k_1,\ k_2]^T = [1,\ 1]^T$ is used. The optimization



(a) y_1. (b) y_2.

Figure 2.3. Measurements and solution trajectories with optimal parameters for the gas-oil cracking problem.



(a) Contours of the objective function plotted along with the steps taken by the optimizer. (□ : initial guess, ∗ : optimal solution).

(b) Values of $f(\boldsymbol{\theta})$, $\|\boldsymbol{g}\|$ and damping parameter μ in each iterate.

Figure 2.4. Performance of the Levenberg-Marquardt method applied to the gas-oil cracking problem.

terminates after 9 iterations with the stopping criterion $\|\boldsymbol{g}\| \leq \varepsilon_1$ satisfied and with optimal parameters $[k_1,\, k_2]^T = [12.0000048,\, 7.9999875]^T$. The measurements and solution trajectories corresponding to the optimal parameters are plotted in Figure 2.3.

Figure 2.4 illustrates the performance of the Levenberg-Marquardt method applied to the current problem. The first plot shows the contours of the objective function and the sequence of steps taken by the optimizer, whereas the second plot shows values of $f(\boldsymbol{\theta})$, $\|\boldsymbol{g}\|$ and the damping parameter μ in each iterate. Good convergence is attained for all iterates. Since this problem is consistent (zero residuals), the Hessian approximation is good for $\boldsymbol{\theta}$ close to $\boldsymbol{\theta}^*$ (c.f. Equation (2.10b)). In agreement with the expected

behaviour, superlinear or quadratic convergence is observed for the last three iterates. ■

2.1.3.4 Hybrid Methods

A good approximation of the Hessian matrix is the key to an efficient solution of the least squares problem. As mentioned earlier, the Gauss-Newton approximation gives an adequate representation of the Hessian when the residuals at the optimum are small, or the residual functions are nearly linear causing the second order terms in Equation (2.10b) to vanish. However, it is desirable to have methods that perform well also on problems with nonlinearities or large residuals. This has led to the construction of *hybrid methods* that are able to switch between Gauss-Newton or Levenberg-Marquardt and other methods with superior performance in case of large residuals. Before discussing these methods, an introduction to *quasi-Newton methods* for general unconstrained minimization is required.

Quasi-Newton methods use a matrix $\boldsymbol{B}$ based on information about first order derivatives to approximate the Hessian. In each iterate the matrix is updated by adding a correction term producing a sequence of matrices $\boldsymbol{B}_0, \boldsymbol{B}_1, \boldsymbol{B}_2, \ldots$ which, under certain conditions, converge towards the true Hessian. Different formulae for updating the approximations have been proposed. The so-called BFGS formula due to Broyden, Fletcher, Goldfarb and Shanno is claimed to be the best (Dennis and Schnabel, 1983):

$$\boldsymbol{B}_{new} = \boldsymbol{B} + \frac{\boldsymbol{y}\boldsymbol{y}^T}{\boldsymbol{h}^T\boldsymbol{y}} - \frac{\boldsymbol{B}\boldsymbol{h}\boldsymbol{h}^T\boldsymbol{B}}{\boldsymbol{h}^T\boldsymbol{B}\boldsymbol{h}} \tag{2.20}$$

in which $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{\theta}_{new}) - \boldsymbol{g}(\boldsymbol{\theta})$ and $\boldsymbol{h} = \boldsymbol{\theta}_{new} - \boldsymbol{\theta}$. Necessary and sufficient conditions for $\boldsymbol{B}_{new}$ to be positive definite are that $\boldsymbol{B}$ is positive definite and that $\boldsymbol{h}^T\boldsymbol{y} > 0$ (the *curvature condition*). The latter condition is automatically satisfied if a line search algorithm is used (Frandsen *et al.*, 1999). Similar to (2.20), formulae for direct updating of the inverse of the Hessian or even its Cholesky factors can be derived, which overall is computationally more efficient. Quasi-Newton methods with BFGS updating are normally implemented with a soft line search, and superlinear final convergence is obtained. This property makes them attractive for nonlinear least squares problems with large residuals, where Gauss-Newton or Levenberg-Marquardt methods only achieve linear convergence.

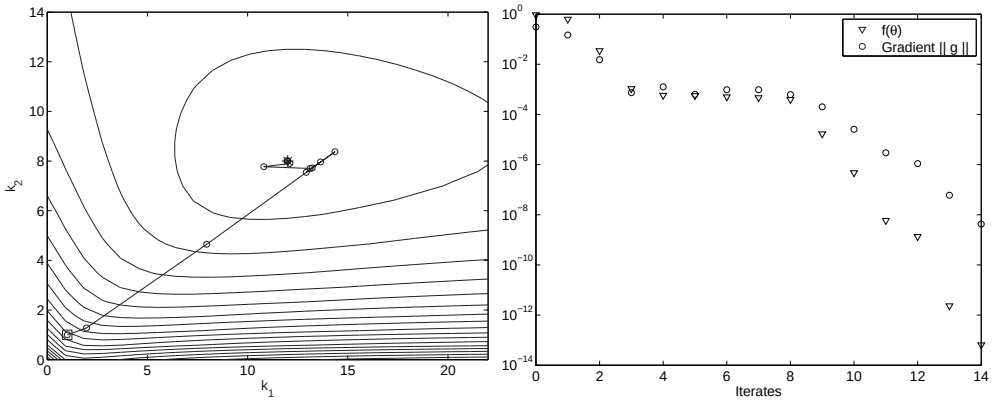
One way to construct a hybrid method is to start out with Gauss-Newton or Levenberg Marquardt, and then switch to quasi-Newton if the performance indicates that $f(\boldsymbol{\theta}^*)$ is significantly nonzero. Methods based on this idea have been proposed by Al-Baali and Fletcher (1985) and Madsen (1988). A different strategy is to use an augmented Gauss-Newton approximation, in which approximations to the second order terms of the Hessian are included. The resulting overall approximation is:

$$\boldsymbol{B} = \boldsymbol{J}^T\boldsymbol{J} + \boldsymbol{S} \tag{2.21}$$

In each iterate $\boldsymbol{S}$ is updated by adding a correction term. One contribution to this class of methods is the work of Dennis *et al.* (1981) where the update

is imbedded in a trust region framework. Also, an extensive review on quasi-Newton methods with application to least squares problems is given by Luksan and Spedicato (2000).

Example 2.3 (Parameter Estimation in Gas-Oil Cracking Model (Continued)) The hybrid method by Madsen (1988) as discussed above was applied to the parameter estimation problem in the gas-oil cracking model. The problem has zero residuals at the optimum, so, as expected, the method never switches to quasi-Newton. Therefore, similar results are obtained as in Example 2.2 with pure Levenberg-Marquardt.



(a) Contours of the objective function plotted along with the steps taken by the optimizer. (□ : initial guess, ∗ : optimal solution).

(b) Values of $f(\boldsymbol{\theta})$ and $\|\boldsymbol{g}\|$ in each iterate.

Figure 2.5. Performance of a quasi-Newton method with BFGS updating and soft line search applied to the gas-oil cracking problem.

Instead, the problem is solved with a pure quasi-Newton method with BFGS updating and soft line search (based on algorithm in Frandsen *et al.* (1999)). The same setup as in Example 2.2 is used with tolerance parameters $\varepsilon_1 = \varepsilon_2 = 10^{-8}$ and initial guess $\boldsymbol{\theta}_0 = [1,\ 1]^T$. This time the optimizer terminates with optimal parameters $\boldsymbol{\theta^*} = [12.0000067,\ 7.9999875]^T$ after 14 iterations, but with 37 function evaluations compared to 10 evaluations (9 iterations plus 1) with Levenberg-Marquardt. Thus, the Levenberg-Marquardt method performs expectedly better, since it exploits the special structure of the least squares problem. Figure 2.5 shows the performance of the quasi-Newton method. Between iterations 4 and 8 the convergence stalls. Comparing the two plots shows that the optimizer misses the target initially and needs to "backtrack" its steps. ■

The hybrid methods constitute a powerful class of methods for efficient and robust solution of least squares problems. They are more efficient than pure quasi-Newton methods with BFGS updating, the reason being that BFGS approximations usually starts with an identity matrix. As the iterations proceed, the updated Hessian eventually resembles the actual Hessian, while the hybrid methods starts with a better approximation for the least squares problem. Thus, the Hessian in the hybrid method resembles the actual Hessian at an earlier stage in the iteration, and therefore better convergence is obtained.

This section concludes the discussion of methods tailored for unconstrained nonlinear least squares problems. The basic principles of the most commonly used methods were emphasized. Aspects of implementation were omitted, e.g. how to solve the (possibly large) linear subproblem arising in each iteration using decompositions to avoid ill-conditioning, etc., which in itself is a huge area of research.

2.2 Problems with Constraints

For many practical reasons restrictions may occur with respect to the parameters being estimated (e.g. rate constants are non-negative). Imposing these restrictions (or constraints) and solving the constrained parameter estimation problem is much more involved than for the unconstrained case. It is outside the scope of this thesis to discuss methods for constrained optimization in general. Instead, the focus of this section is on a particular class of methods based on *sequential quadratic programming* (SQP) that enables general nonlinear constraints to be incorporated into the optimization problem. SQP methods have proved powerful in parameter estimation problems for dynamical systems (Schittkowski, 2002), and they also seem the preferred choice in the context of solving nonlinear optimal control problems (Jørgensen *et al.*, 2004; Leineweber, 1995). At the end of this section a brief discussion is given on a technique to improve robustness of the parameter estimation procedure without increasing the computational load.

2.2.1 Optimality Conditions

The following constrained optimization problem is considered:

$$\begin{aligned} &\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \\ s.t. \quad &c_i(\boldsymbol{\theta}) = 0, \quad i = 1, 2, \ldots, m_e \\ &c_i(\boldsymbol{\theta}) \geq 0, \quad i = m_e + 1, \ldots, m_c \end{aligned} \tag{2.22}$$

in which $\boldsymbol{\theta} \in \mathbb{R}^{n_p}$ is the vector of parameters, $f : \mathbb{R}^{n_p} \mapsto \mathbb{R}$ is the objective function, and $c_i(\boldsymbol{\theta}) : \mathbb{R}^{n_p} \mapsto \mathbb{R}$ are the constraint functions. A vector $\boldsymbol{\theta}$ is said to be feasible if it satisfies both the equality and inequality constraints in (2.22), and the set of feasible vectors is denoted $\mathcal{P}$ (the feasible region). Furthermore, an index set $\mathcal{I}(\boldsymbol{\theta})$ is defined that holds the indices of the active inequality constraints at $\boldsymbol{\theta}$:

$$\mathcal{I}(\boldsymbol{\theta}) = \{i \ : \ c_i(\boldsymbol{\theta}) = 0, \ m_e < i \leq m_c\} \tag{2.23}$$

The *Lagrangian function*, which is central in the development of a solution procedure for constrained nonlinear optimization, is defined as:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = f(\boldsymbol{\theta}) - \sum_{i=1}^{m_c} \lambda_i c_i(\boldsymbol{\theta}) \tag{2.24}$$

where $\lambda_i \in \mathbb{R}^{m_c}$ are the *Lagrange multipliers*. Similar to Section 2.1.1, necessary and sufficient conditions exist for $\boldsymbol{\theta}^*$ to be a constrained local minimizer. Since

these conditions form the basis for every attempted solution of the optimization problem, they are included here, although in a non-rigorous form. For the unconstrained case the first and second order conditions involve the gradient and the Hessian of the objective function. Completely analogous to this, the corresponding conditions for the constrained case involve the gradient and the Hessian (both with respect to $\boldsymbol{\theta}$) of the Lagrangian function.

First, a constraint qualification is needed (Nocedal and Wright, 1999):

Definition 2.3 (Linear Independence Constraint Qualification (LICQ))
The optimization problem (2.22) is said to satisfy a constraint qualification in $\boldsymbol{\theta}^* \in \mathcal{P}$, if the gradients $\nabla c_i(\boldsymbol{\theta}^*)$ for all equality and active inequality constraints are linearly independent.

With this definition the first order necessary conditions, known as *Karush-Kuhn-Tucker* (KKT) necessary conditions, may be stated in the following theorem:

Theorem 2.4 (Karush-Kuhn-Tucker Necessary Condition)
If $\boldsymbol{\theta}^*$ is a local minimizer for f, and the constraint qualification holds, then there exists a Lagrangian multiplier vector $\boldsymbol{\lambda}^* \in \mathbb{R}^{m_c}$ such that:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^*, \boldsymbol{\lambda}^*) = \mathbf{0} \tag{2.25a}$$
$$c_i(\boldsymbol{\theta}^*) = 0, \qquad i = 1, 2, \ldots, m_e \tag{2.25b}$$
$$c_i(\boldsymbol{\theta}^*) \geq 0, \qquad i = m_e + 1, \ldots, m_c \tag{2.25c}$$
$$\lambda_i^* \geq 0, \qquad i = m_e + 1, \ldots, m_c \tag{2.25d}$$
$$\lambda_i^* c_i(\boldsymbol{\theta}^*) = 0, \qquad i = m_e + 1, \ldots, m_c \tag{2.25e}$$

The first equations states that the gradient of the Lagrangian function is equal to zero at the local minimizer, which is quite similar to the first order condition for the unconstrained case (c.f. Theorem 2.1). Vectors $(\boldsymbol{\theta}^*, \boldsymbol{\lambda}^*)$ that satisfy the KKT necessary condition are possible candidates for a local solution of problem (2.22). These vectors, however, also include local maximizers and saddle points. To distinguish the local minimizer, second order conditions are needed, unless the sufficient condition in the next theorem holds:

Theorem 2.5 (Karush-Kuhn-Tucker Sufficient Condition)
If f is convex, all equality constraints are linear, and all inequality constraints are concave, then $\boldsymbol{\theta}^*$ is a global minimizer of (2.22).

This condition that ensures a global solution will be used later in the subproblems arising in the SQP method. For situations where the KKT sufficient condition does not apply, second order information is needed. The second order necessary condition for a local minimizer may be stated as in the following theorem:

Theorem 2.6 (Second Order Necessary Condition)
If $\boldsymbol{\theta}^*$ is a local minimizer for f, if the constraint qualification holds, and if $\boldsymbol{\lambda}^*$ is a vector such that the KKT conditions of Theorem 2.4 are satisfied, then:

$$\boldsymbol{h}^T \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^*, \boldsymbol{\lambda}^*) \boldsymbol{h} \geq 0 \tag{2.26}$$

for all $\boldsymbol{h} \in \mathbb{R}^{n_p}$ *with* $\nabla c_i(\boldsymbol{\theta}^*)^T \boldsymbol{h} = 0$ *for* $i \in \{1, 2, \ldots, m_e\} \cup \mathcal{I}(\boldsymbol{\theta}^*)$ *(all equality and active inequality constraints).* $\nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^*, \boldsymbol{\lambda}^*)$ *denotes the Hessian of* $\mathcal{L}$ *with respect to* $\boldsymbol{\theta}$ *given by:*

$$\nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^*, \boldsymbol{\lambda}^*) = \nabla^2 f(\boldsymbol{\theta}^*) - \sum_{i=1}^{m_c} \lambda_i^* \nabla^2 c_i(\boldsymbol{\theta}^*) \tag{2.27}$$

This condition can be used to prove that a point which satisfies Theorem 2.4 is not a local minimizer. Finally, the second order sufficient condition, which is of more practical importance, since it can be used to prove that a point is a solution to (2.22) without the assumptions of Theorem 2.5, may be stated:

Theorem 2.7 (Second Order Sufficient Condition)
If $\boldsymbol{\theta}^*$ *and* $\boldsymbol{\lambda}^*$ *satisfy the KKT conditions of Theorem 2.4, and if:*

$$\boldsymbol{h}^T \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^*, \boldsymbol{\lambda}^*) \boldsymbol{h} > 0$$

for every nonzero $\boldsymbol{h} \in \mathbb{R}^{n_p}$ *with* $\nabla c_i(\boldsymbol{\theta}^*)^T \boldsymbol{h} = 0$ *for* $i \in \{1, 2, \ldots, m_e\}$ *and for all* $\boldsymbol{h}$ *with* $\nabla c_i(\boldsymbol{\theta}^*)^T \boldsymbol{h} = 0$ *for* $i \in \mathcal{I}(\boldsymbol{\theta}^*)$ *and* $\lambda_i > 0$*, then* $\boldsymbol{\theta}^*$ *is a constrained local minimizer for* f*.*

The last part is rather tricky, because the necessary and sufficient conditions only differ with a strict inequality in (2.26), and in the way the active inequality constraints with zero Lagrange multipliers are treated. A thorough treatment of optimality conditions for constrained optimization can be found in e.g. Nocedal and Wright (1999). Here, the conditions are presented as concise as possible only to facilitate the subsequent discussion of solution strategies.

2.2.2 Sequential Quadratic Programming Methods

The constrained optimization problem (2.22) is considered. The basic principle for its solution is to replace the problem by a sequence of subproblems, which are easier to solve. This is a well-known strategy from unconstrained optimization, where for example a quadratic model is used to approximate the objective function in each iteration. Since the approximating model is only valid in a neighborhood of the current point, either a line search or trust region framework is applied to ensure global convergence. These principles are also used in SQP methods, which are based on solving a quadratic programming (QP) subproblem in each iteration. The QP problem is obtained by linearizing the constraints and approximating the Lagrangian function quadratically.

To illustrate how the QP subproblem arises, the special case with no inequality constraints is considered:

$$\begin{aligned} &\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \\ &s.t. \quad c_i(\boldsymbol{\theta}) = 0, \quad i = 1, 2, \ldots, m_c \end{aligned} \tag{2.28}$$

Finding a local minimizer of (2.28) is done by solving the KKT necessary conditions of Theorem 2.4:

$$\boldsymbol{F}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \begin{bmatrix} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \\ \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \end{bmatrix} = \begin{bmatrix} \nabla f(\boldsymbol{\theta}) - \nabla \boldsymbol{c}(\boldsymbol{\theta})^T \boldsymbol{\lambda} \\ -\boldsymbol{c}(\boldsymbol{\theta}) \end{bmatrix} = \boldsymbol{0} \tag{2.29}$$

in which

$$(\nabla \boldsymbol{c}(\boldsymbol{\theta}))_{ij} = \frac{\partial c_i}{\partial \theta_j} \tag{2.30}$$

If Newton's method is applied to this system of nonlinear equations, a search direction $(\boldsymbol{h}, \boldsymbol{\eta})$ from the current point $(\boldsymbol{\theta}, \boldsymbol{\lambda})$ is obtained by solving:

$$\begin{bmatrix} \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) & -\nabla \boldsymbol{c}(\boldsymbol{\theta})^T \\ -\nabla \boldsymbol{c}(\boldsymbol{\theta}) & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{h} \\ \boldsymbol{\eta} \end{bmatrix} = - \begin{bmatrix} \nabla f(\boldsymbol{\theta}) - \nabla \boldsymbol{c}(\boldsymbol{\theta})^T \boldsymbol{\lambda} \\ -\boldsymbol{c}(\boldsymbol{\theta}) \end{bmatrix} \tag{2.31}$$

in which the Hessian of the Lagrangian was defined in (2.27). Conditions on the coefficient matrix in (2.31) to be non-singular can be found in e.g. Jørgensen *et al.* (2004). In each Newton step the following update is made:

$$\begin{aligned} \boldsymbol{\theta} &= \boldsymbol{\theta} + \boldsymbol{h} \\ \boldsymbol{\lambda} &= \boldsymbol{\lambda} + \boldsymbol{\eta} \end{aligned} \tag{2.32}$$

Elimination of $\boldsymbol{\eta}$ between (2.32) and (2.31) gives:

$$\begin{bmatrix} \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) & -\nabla \boldsymbol{c}(\boldsymbol{\theta})^T \\ -\nabla \boldsymbol{c}(\boldsymbol{\theta}) & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{h} \\ \boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \nabla f(\boldsymbol{\theta}) \\ -\boldsymbol{c}(\boldsymbol{\theta}) \end{bmatrix} \tag{2.33}$$

This is equivalent to:

$$\begin{aligned} \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \boldsymbol{h} - \nabla \boldsymbol{c}(\boldsymbol{\theta})^T \boldsymbol{\lambda} + \nabla f(\boldsymbol{\theta}) &= \mathbf{0} \\ \nabla \boldsymbol{c}(\boldsymbol{\theta}) \boldsymbol{h} + \boldsymbol{c}(\boldsymbol{\theta}) &= \mathbf{0} \end{aligned} \tag{2.34}$$

These equations are exactly the optimality conditions for the following QP subproblem (c.f. Theorem 2.5):

$$\begin{aligned} &\min_{\boldsymbol{h}} \, q(\boldsymbol{h}) = \frac{1}{2} \boldsymbol{h}^T \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \boldsymbol{h} + \nabla f(\boldsymbol{\theta})^T \boldsymbol{h} \\ &s.t. \quad \nabla \boldsymbol{c}(\boldsymbol{\theta}) \boldsymbol{h} + \boldsymbol{c}(\boldsymbol{\theta}) = 0 \end{aligned} \tag{2.35}$$

Thus, solving the KKT necessary conditions by Newton's method is equivalent to solving a sequence of quadratic programs (2.35). This observation motivates the formulation of QP subproblems and hence the name sequential quadratic programming for the overall algorithm. In the case of nonlinear inequality constraints, (2.35) can still be used if the feasible region is modified to account for the extra constraints. However, at all times it is necessary to know which of the inequality constraints that are active. The inequality constraints are "monitored" by keeping a set of active constraints (an *active set* strategy).

The above is a very brief description of the principles in SQP methods, and there still remain important issues to make the methods practical. First, for reasons similar to the discussion in Section 2.1.3.4, an approximation is needed to replace the Hessian of the Lagrangian in (2.35). A BFGS update can be applied, which gives superlinear final convergence (Schittkowski, 2002). For least squares problems a Gauss-Newton type approximation can be made. Secondly, since the quadratic approximation is only valid close to the current point, the methods should be equipped with a line search in order to make the convergence robust. Details on these important topics are given in Nocedal and Wright (1999).

2.2.2.1 Solving Nonlinear Least Squares Problems by SQP

For nonlinear least squares problems special purpose methods such as Gauss-Newton or Levenberg-Marquardt are widely used. However, also SQP methods can be applied to these problems with the additional advantage that constraints can be handled. Schittkowski (2002, 1988) advocates the use of SQP methods strongly, showing that the performance is at least comparable, or in some cases superior, to conventional methods. First, the unconstrained least squares problem is considered:

$$\min_{\boldsymbol{\theta}} \; f(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{m} r_i^2(\boldsymbol{\theta}) \tag{2.36}$$

A transformation of this problem is proposed in Schittkowski (1988), which consists of introducing m additional variables $\boldsymbol{z} = [z_1, \ldots, z_m]^T$ and m additional equality constraints of the form:

$$r_i(\boldsymbol{\theta}) - z_i = 0, \quad i = 1, \ldots, m \tag{2.37}$$

The transformed problem is:

$$\begin{aligned} &\min_{\boldsymbol{\theta},\boldsymbol{z}} \; \frac{1}{2} \boldsymbol{z}^T \boldsymbol{z} \\ &s.t. \quad \boldsymbol{r}(\boldsymbol{\theta}) - \boldsymbol{z} = \boldsymbol{0} \end{aligned} \tag{2.38}$$

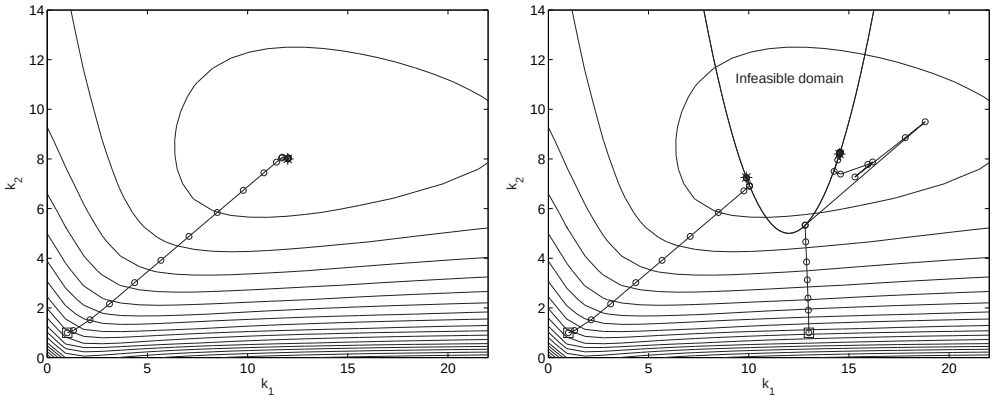
in which $[\boldsymbol{\theta}, \boldsymbol{z}] \in \mathbb{R}^{n_p+m}$. Applying the SQP algorithm, Schittkowski shows that the Hessian of the Lagrangian has a special structure that allows for easy update, and that the underlying structure of the least squares problem is automatically exploited in terms of the Gauss-Newton approximation (details are given in Schittkowski (2002, 1988)). In this way, typical features of special purpose least squares codes are retained. Although m additional variables are introduced, the effort required to solve each QP subproblem is still $\mathcal{O}\left(n_p^3\right)$, which is comparable to the effort required in each iteration of a Gauss-Newton or Levenberg-Marquardt method.

Example 2.4 (Parameter Estimation in Gas-Oil Cracking Model (Continued))
The specialized approach of Schittkowski (1988) for nonlinear least squares is studied in later chapters. Instead, the use of the SQP algorithm `fmincon` available within the Optimization Toolbox in MATLAB is demonstrated on the parameter estimation problem for the gas-oil cracking model. `fmincon` has a medium-scale and a large-scale option. The medium-scale option used in this example, is an SQP method with line search and BFGS update, where each QP subproblem is solved using an active set strategy (Mathworks, 2003). First, a simple optimization is performed with lower bounds on the parameters, which will not influence the optimal solution:

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \; f(\boldsymbol{\theta}) = \frac{1}{2} \sum_{j=1}^{2} \sum_{i=1}^{10} \left(y_j(t_i, \theta) - \tilde{y}_{ij}\right)^2 \\ &s.t. \quad \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned}$$

The starting point $\boldsymbol{\theta}_0 = [1,\ 1]^T$ is used, and a tolerance of 10^{-8} is specified in a stopping criterion based on the magnitude of the gradient. The optimization terminates after 20 iterations with a total of 55 function evaluations and optimal parameters

$\boldsymbol{\theta}^* = [12.001892, 7.999979]^T$. The sequence of steps is illustrated on Figure 2.6a. Compared to the Levenberg-Marquardt and quasi-Newton approaches more function evaluations are needed. However, a careful interpretation should be made, since differences in stopping criteria etc. may exist.



(a) Only lower bounds.

(b) Lower bounds and one additional nonlinear constraint.

Figure 2.6. Performance of the SQP method `fmincon` applied to the gas-oil cracking problem with two different sets of constraints. The contours of the objective function are plotted along with the steps taken by the SQP algorithm. (□ : initial guess, $*$: optimal solution).

To illustrate a situation with active constraints, the problem is changed to:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{1}{2}\sum_{j=1}^{2}\sum_{i=1}^{10}\left(y_j(t_i,\theta) - \tilde{y}_{ij}\right)^2$$

$$s.t. \quad \begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix} \geq \begin{bmatrix}0 \\ 0\end{bmatrix}$$

$$\frac{1}{2}\theta_1^2 - 24\theta_1 + 77 - \theta_2 \geq 0$$

When starting at $\boldsymbol{\theta}_0 = [1,\ 1]^T$ the algorithm terminates after 15 iterations and 45 function evaluations, and the optimal parameters are $\boldsymbol{\theta}^* = [9.876389, 7.254858]^T$. The maximum allowable constraint violation is set to 10^{-8}. The steps are illustrated in Figure 2.6b. The algorithm finds a local minimum on the border of the feasible domain. Changing the starting point to $\boldsymbol{\theta}_0 = [13,\ 1]^T$ gives the second sequence of steps in Figure 2.6b. This time the algorithm terminates after 24 iterations due to too many function evaluations ($k_{max} = 200$). Inspection shows that no progress is made after iteration no. 17. The iterations oscillate near the constraint, and the algorithm fails to satisfy the termination criteria. ■

2.2.3 Multiple Shooting

To conclude this chapter on optimization strategies in parameter estimation problems a brief discussion is given on a technique that improves robustness

of the optimization and more effectively uses the available information. All the methods discussed so far, both constrained and unconstrained, may be regarded as *single shooting* techniques, also referred to as *initial value problem* (IVP) approaches. Starting with some initial guess on the parameters, the basic idea of the IVP approach is:

1. Integrate the underlying model equations over the whole time interval.
2. Evaluate the objective function.
3. Compute a correction to the parameters in order to decrease the value of the objective function.

As pointed out by Bock (1981, 1983) the IVP approach has two major drawbacks. For poor starting guesses on the parameters, the underlying equations may be hard to solve, or a solution may not even exist. Even when the "true" parameters result in a perfectly well conditioned system, poor parameter values may cause excessive stiffness of the equations or instability in the numerical solution. A second drawback is what Bock (1983) calls the *reinversion of the inverse problem*. In the inverse problem information about the states of a system is available through the measurements, and the objective is to estimate unknown parameters. This information, however, is neglected in the IVP approach, since the states are eliminated by solving the model equations. Thus, an approach that more efficiently uses the available information is desirable. Bock (1981, 1983) suggests using a *multiple shooting* strategy, which is also referred to as a *boundary value problem* (BVP) approach, since the problem is reformulated is a multipoint BVP. The basic concept is to introduce an additional set of shooting points along the time axis. If $[t_0, t_f]$ is the time interval for the overall IVP, then a mesh is defined as:

$$t_0 = \tau_0 < \tau_1 < \cdots < \tau_{m_s} = t_f, \quad \Delta\tau_j = \tau_{j+1} - \tau_j, \quad j = 0, 1, \ldots, m_s - 1$$

Instead of integrating over the whole time interval, integration is only performed from one shooting point to the next. Assuming that the model is described by a set of ODEs, the problem is separated into m_s independent IVPs:

$$\dot{\boldsymbol{y}} = \boldsymbol{f}(t, \boldsymbol{y}, \theta), \quad \boldsymbol{y}(t_j) = \boldsymbol{s}_j, \quad t \in [\tau_j, \tau_{j+1}] \tag{2.39}$$

in which the additional shooting parameters $[\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{m_s}]$ are estimates of the states $\boldsymbol{y}(t_j)$. The shooting parameters are unknown parameters which are computed in addition to the model parameters $\boldsymbol{\theta}$. Thus, the total parameter vector is:

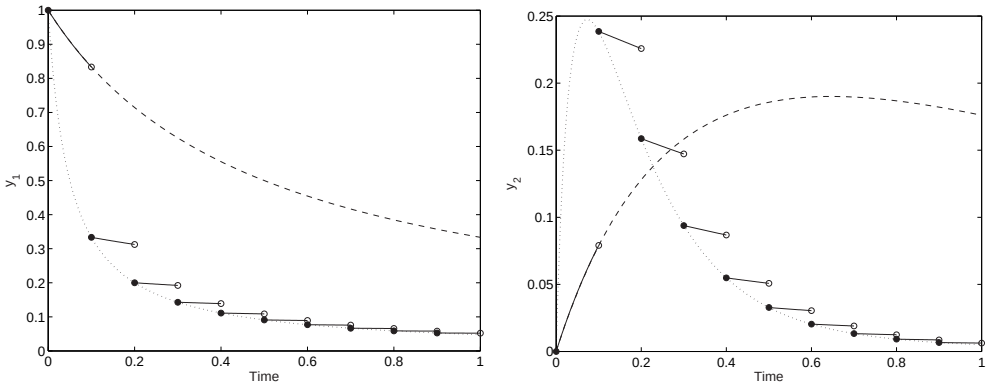
$$\bar{\boldsymbol{\theta}} = [\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{m_s}, \boldsymbol{\theta}]^T \tag{2.40}$$

To ensure that the final solution is continuous, a set of matching conditions are introduced, and assuming that no other constraints are present the overall optimization problem is:

$$\begin{aligned} &\min_{\bar{\boldsymbol{\theta}}} \; f(\bar{\boldsymbol{\theta}}) = \frac{1}{2}\boldsymbol{r}(\bar{\boldsymbol{\theta}})^T\boldsymbol{r}(\bar{\boldsymbol{\theta}}) \\ &s.t. \quad \boldsymbol{y}(\tau_{j+1}, \boldsymbol{s}_j, \boldsymbol{\theta}) - \boldsymbol{s}_{j+1} = 0, \quad j = 0, 1, \ldots, m_s - 1 \end{aligned} \tag{2.41}$$

The additional shooting parameters require starting guesses, and this is actually the main advantage of multiple shooting, since it allows the user to bring in information about the states. That is, if measurement values are available for the states of the system, and if the shooting points coincide with (some of) the experimental times, then the measurement values can be used as starting guesses for the shooting parameters. Hence, the influence of poor starting guesses for the parameters is substantially reduced.

Example 2.5 (Multiple Shooting for Gas-Oil Cracking Model)
This example illustrates the effect of using multiple shooting on the initial trajectories in the parameter estimation problem for the gas-oil cracking model. Assume that a



(a) y_1. (b) y_2.

Figure 2.7. Initial multiple shooting and single shooting trajectories compared to the final trajectory. (solid lines : multiple shooting; dashed lines : single shooting; dotted lines : final trajectory).

mesh is constructed with 10 subintervals such that the shooting points coincide with the experimental times. If the measurement values are used as starting guesses for the shooting parameters and if $\boldsymbol{\theta}_0 = [1, 1]^T$, then the initial trajectories are as illustrated in Figure 2.7. The trajectories are much closer to the final solution compared to the initial single shooting trajectories. For this example the measurements all coincide with the final solution. Of course, this is not the case in general, but the measurements can still provide very good starting guesses for the shooting parameters. ■

A drawback of multiple shooting is the increased complexity of the problem caused by the large set of extra parameters that are introduced. However, due to the special structure of the problem the computational effort is comparable to single shooting (Bock, 1981, 1983; Schittkowski, 2002).

An alternative to the multiple shooting technique that also treats the problem as a multipoint BVP is to discretize each subinterval using orthogonal collocation. Details on orthogonal collocation are found in Villadsen and Michelsen (1978), and Tjoa and Biegler (1991) have used this approach in parameter estimation problems.

3

Numerical Solution of the Model Equations

As indicated earlier, essentially two components are needed for the practical solution of parameter estimation problems: An optimization algorithm and a differential equation solver. So far, no considerations have been made regarding the numerical solution of the model equations and the calculation of gradients required by the optimization algorithm. Since these key aspects are extremely important for an efficient and robust solution of parameter estimation problems, they are addressed in this chapter.

A brief discussion on solving ODEs and DAEs is given in Section 3.1, emphasizing two specific classes of methods, which will be used in subsequent chapters when solving real problems. By far the most computationally intensive part of parameter estimation in dynamical systems is the calculation of derivative information for the optimizer (Bard, 1974; Stortelder, 1998), especially when optimization is required with respect to a large set of parameters. Therefore, the main focus of this chapter is on different methods specifically tailored for calculation of derivative information. This is the topic of Section 3.2.

3.1 Solving Differential Equations

The available literature on the numerical solution of ODEs and DAEs is very rich, and no attempt is made here to review the entire field. An excellent treatment is found in the books by Hairer, Nørsett and Wanner (1992) and Hairer and Wanner (1996). Instead, a brief discussion is given on two specific classes of methods, which also seem the most popular choices: Runge-Kutta methods and BDF (backward differentiation formula) methods.

3.1.1 Runge-Kutta Methods

For simplicity, the methods are introduced assuming that no algebraic equations are present. Thus, the following system of equations is considered:

$$\dot{\boldsymbol{y}} = \boldsymbol{f}(t, \boldsymbol{y}, \boldsymbol{\theta}), \qquad \boldsymbol{y}(t_0, \boldsymbol{\theta}) = \boldsymbol{y}_0(\boldsymbol{\theta}) \tag{3.1}$$

in which $\boldsymbol{\theta} \in \mathbb{R}^{n_p}$ is the vector of unknown parameters, $\boldsymbol{y} \in \mathbb{R}^n$ is the state vector depending on t and $\boldsymbol{\theta}$, and $\boldsymbol{f}$ is a function mapping $\mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^{n_p}$ into $\mathbb{R}^n$. The solution to this *initial value problem* (IVP) by most numerical schemes proceeds in a stepwise fashion from a given initial value. Methods that use one starting

value at each step are denoted *one-step methods*, whereas methods using several values of the solution are denoted *multi-step methods*. All numerical methods for ordinary differential equations can be classified into one of these groups. One of the most widely used classes of one-step methods are the Runge-Kutta methods. The following scheme represents a general s-stage Runge-Kutta method:

$$\boldsymbol{Y}_i = \boldsymbol{y}_n + h \sum_{j=1}^{s} a_{ij} \boldsymbol{f}(t_n + c_j h, \boldsymbol{Y}_j, \boldsymbol{\theta}) \tag{3.2a}$$

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h \sum_{i=1}^{s} b_i \boldsymbol{f}(t_n + c_i h, \boldsymbol{Y}_i, \boldsymbol{\theta}) \tag{3.2b}$$

$$\boldsymbol{e}_{n+1} = h \sum_{i=1}^{s} d_i \boldsymbol{f}(t_n + c_i h, \boldsymbol{Y}_i, \boldsymbol{\theta}) \tag{3.2c}$$

in which $\boldsymbol{Y}_i$ designates the solution at the ith ($i = 1, 2, \ldots, s$) internal stage of integration step n and $\boldsymbol{e}$ is the error vector. The coefficients are defined by the so-called Butcher tableau (Hairer *et al.*, 1992):

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \ldots & a_{1s} \\ c_2 & a_{21} & a_{22} & \ldots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \ldots & a_{ss} \\ \hline \boldsymbol{y}_{n+1} & b_1 & b_2 & \ldots & b_s \\ \hline \boldsymbol{e}_{n+1} & d_1 & d_2 & \ldots & d_s \end{array} \quad = \quad \begin{array}{c|c} \boldsymbol{c} & \boldsymbol{A} \\ \hline & \boldsymbol{b}^T \\ \hline & \boldsymbol{d}^T \end{array} \tag{3.3}$$

where $\boldsymbol{A}$ is an $s \times s$ matrix and $\boldsymbol{b}$, $\boldsymbol{d}$ and $\boldsymbol{c}$ are $s \times 1$ vectors. Since the numerical solution of (3.1) is closely linked to quadrature, the elements of $\boldsymbol{c}$ are referred to as the quadrature nodes and the elements of $\boldsymbol{b}$ are the quadrature weights. The coefficient vector $\boldsymbol{d}$ contains the quadrature weights used for error estimation.

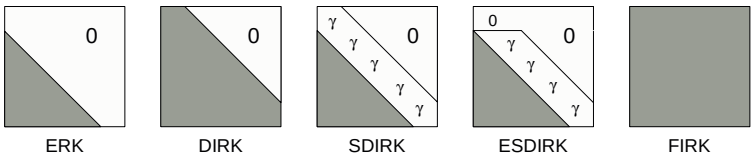


Figure 3.1. Structure of the $\boldsymbol{A}$ matrix in the Butcher tableau for different classes of Runge-Kutta methods.

Runge-Kutta methods are classified according to the structure of the $\boldsymbol{A}$-matrix in the Butcher tableau as illustrated in Figure 3.1. For explicit methods (ERK), $\boldsymbol{A}$ is strictly lower triangular, implying that all calculations can be done explicitly, making these methods computationally fast and straightforward to implement. However, in general, ERK methods have poor stability properties, which make them unsuited for stiff problems (Hairer *et al.*, 1992). The four remaining subclasses of Runge-Kutta methods in Figure 3.1 are all implicit. The values of the internal stages in (3.2) can no longer be calculated explicitly from the values of the previous stages, but depend nonlinearly on all the stages (for FIRK methods). Each integration step of an implicit method requires the

solution of a system of ns nonlinear equations. Normally, an iterative method, such as Newton's method, is applied. For diagonal implicit methods (DIRK) the iterations in the internal stages are carried out for one stage at a time, lowering the computational cost compared to fully implicit methods (FIRK). If all diagonal elements of $\boldsymbol{A}$ are identical and upper diagonal elements are zero, the method is said to be singly diagonal implicit (SDIRK). Finally, if the first stage of an SDIRK method is explicitly given, the method is said to be explicit singly diagonal implicit (ESDIRK).

It might seem strange at first that diagonal Runge-Kutta methods draw so much attention, but they constitute a good compromise between stability, order and computational efficiency. To illustrate the latter part, the iterations required in the solution of ith internal stage for an SDIRK method are outlined. From (3.2) the ith internal stage is:

$$\boldsymbol{Y}_i = \boldsymbol{y}_n + h\sum_{j=1}^{i-1} a_{ij}\boldsymbol{f}(t_n + c_j h, \boldsymbol{Y}_j, \boldsymbol{\theta}) + h\gamma\boldsymbol{f}(t_n + c_i h, \boldsymbol{Y}_i, \boldsymbol{\theta}) \tag{3.4}$$

In residual form this becomes:

$$\boldsymbol{R}(\boldsymbol{Y}_i) = \boldsymbol{Y}_i - \boldsymbol{y}_n - h\sum_{j=1}^{i-1} a_{ij}\boldsymbol{f}(t_n + c_j h, \boldsymbol{Y}_j, \boldsymbol{\theta}) - h\gamma\boldsymbol{f}(t_n + c_i h, \boldsymbol{Y}_i, \boldsymbol{\theta})$$

The residual is equated to zero and solved via Newton iterations:

$$\nabla\boldsymbol{R}\left(\boldsymbol{Y}_i^{(k)}\right)\cdot\Delta\boldsymbol{Y}_i = -\boldsymbol{R}\left(\boldsymbol{Y}_i^{(k)}\right)$$
$$\boldsymbol{Y}_i^{(k+1)} = \boldsymbol{Y}_i^{(k)} + \Delta\boldsymbol{Y}_i$$

in which the gradient of $\boldsymbol{R}$ is:

$$\nabla\boldsymbol{R}(\boldsymbol{Y}_i) = \boldsymbol{I} - h\gamma\nabla_{\boldsymbol{y}}\boldsymbol{f}(t_n + c_i h, \boldsymbol{Y}_i, \boldsymbol{\theta}) \tag{3.5}$$

This shows the importance of identical diagonal elements. If the Jacobian of the ODE system $\nabla_{\boldsymbol{y}}\boldsymbol{f}$ is approximated by its value at the first internal stage, then only one evaluation of the Jacobian and one factorization of $\nabla\boldsymbol{R}$ is needed in each integration step (some codes even reuse the factorization in several successive steps).

A short introduction of the concepts of order and stability is given in the following. The order of a method is a measure of the rate, at which the error incurred in the numerical integration from t_n to $t_n + h$ decreases with the step length h. The concept of order is defined through the Taylor expansion (Hairer *et al.*, 1992):

Definition 3.1 (Order of Runge-Kutta Methods)
A method has order p, if the difference between a Taylor series for the exact solution through the point $\boldsymbol{y}_n$ *evaluated at* $\boldsymbol{y}(t_n + h)$ *and the numerical solution* $\boldsymbol{y}_{n+1}$ *is* $\mathcal{O}\left(h^{p+1}\right)$*:*

$$\left\|\boldsymbol{y}(t_n + h) - \boldsymbol{y}_{n+1}\right\| \leq Kh^{p+1} \tag{3.6}$$

i.e., the two expansions coincide up to and including the term h^p*.*

General order conditions for Runge-Kutta methods can be found in Hairer *et al.* (1992). If high accuracy in the solution is required, a method with high order will normally be advantageous.

Instability in the numerical solution of a problem arises if large differences in the magnitudes of the eigenvalues exist. A problem with this characteristic is said to be *stiff*. Many problems in chemical engineering are known to be stiff due to the presence of fast and slow phenomena in the processes they originate from. As mentioned in Section 2.2.3, poor starting guesses for the parameters in a parameter estimation problem may cause stiffness of the equations, even though the problem is non-stiff with the final parameter estimates. Consequently, to ensure robustness of the parameter estimation process, an equation solver with good stability properties should be applied. Assessment of stability is associated with the scalar linear equation $\dot{y} = \lambda y,\ \ y(0) = y_0$, where $\lambda \in \mathbb{C}$. For all Runge-Kutta methods, advancing one step in the solution of this equation is equal to the multiplication with a rational function, $R(z)$:

$$\begin{aligned} y_{n+1} &= R(h\lambda) y_n \\ R(z) &= 1 + z\boldsymbol{b}^T \left(\boldsymbol{I} - z\boldsymbol{A}\right)^{-1} \mathbf{1} \end{aligned} \tag{3.7}$$

in which $\mathbf{1}$ is an s-dimensional unity vector $(1, \ldots, 1)^T$. The method is said to be *A-stable*, if $|R(z)| < 1$ for $\mathrm{Re}(z) < 0$. Furthermore, A-stable methods that satisfy $\lim_{z\to\infty} R(z) = 0$ are called *L-stable*. This property is particularly important when solving DAEs. The order conditions and the desired stability properties jointly define the coefficients of the Butcher tableau for the individual Runge-Kutta methods. The actual choice of method depends on the desired application, but diagonal methods with orders 2–4 have been proposed by many researchers (Nørsett, 1974; Alexander, 1977, 2003; Cameron, 1983; Thomsen, 2002) due to their good stability properties and easy implementation.

3.1.1.1 ESDIRK34: An Implicit Runge-Kutta Solver of Order 3

Several Runge-Kutta codes are available including a five-stage SDIRK method with order 4 and a three-stage FIRK method with order 5 (RADAU5), both described in Hairer and Wanner (1996). One of the methods used in subsequent chapters is a four-stage ESDIRK method with order 3 (Alexander, 2003) implemented in the code ESDIRK34 (see Kristensen *et al.* (2004a) for details). The code is tailored for sensitivity integration, the details of which are explained in Section 3.2. Since this method will be used later, an introduction is given here.

The Butcher tableau of the ESDIRK method is:

0	0			
c_2	a_{21}	γ		
c_3	a_{31}	a_{32}	γ	
1	b_1	b_2	b_3	γ
$\boldsymbol{y}_{n+1}$	b_1	b_2	b_3	γ
$\boldsymbol{e}_{n+1}$	d_1	d_2	d_3	d_4

(3.8)

The coefficients are provided in Table 3.1. For later reference, 'ESDIRK34' denotes this specific method. In its present form ESDIRK34 solves DAEs of

γ	0.435866521508	b_1	0.102399400620
c_2	0.871733043017	b_2	-0.376878452256
c_3	0.468238744852	b_3	0.838612530127
a_{21}	0.435866521508	d_1	0.054625497240
a_{31}	0.140737774725	d_2	0.494208893626
a_{32}	-0.108365551381	d_3	-0.221934499735
		d_4	-0.326899891131

Table 3.1. Coefficients for ESDIRK34 with Butcher tableau (3.8).

the form:

$$\begin{aligned} \dot{\boldsymbol{y}} &= \boldsymbol{f}(t, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\theta}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \\ \boldsymbol{0} &= \boldsymbol{g}(t, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\theta}), \quad \boldsymbol{z}(t_0) = \boldsymbol{z}_0 \end{aligned} \tag{3.9}$$

in which $\boldsymbol{y} \in \mathbb{R}^{n_d}$ and $\boldsymbol{z} \in \mathbb{R}^{n_a}$. n_d and n_a are introduced to denote the number of differential and algebraic equations, respectively. The total dimension of the DAE system is $n = n_d + n_a$. The extra algebraic equations are solved along with the implicit equations in each internal stage in (3.2). Only DAE problems of differential index 1 (implying that $\nabla_{\boldsymbol{z}}\boldsymbol{g}$ is nonsingular) are treated.

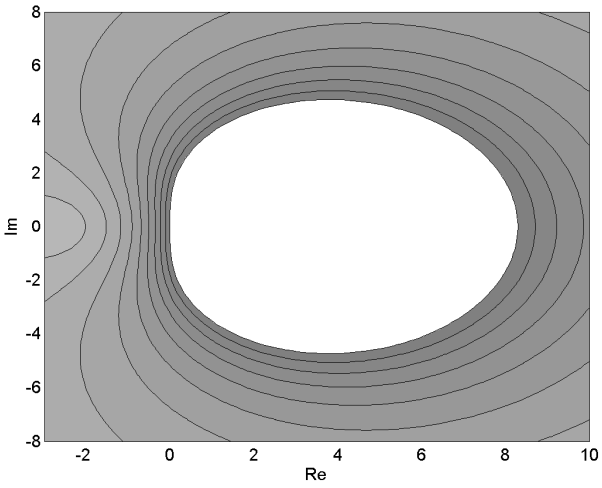


Figure 3.2. Stability domain for ESDIRK34. The method is stable for all $h\lambda$ in the shaded area. The lines are contours of the function $|R(h\lambda)|$ (c.f. (3.7)).

The ESDIRK34 method has several desirable properties. It is A-stable as well as L-stable, and the basic order of the method is 3 with an imbedded method of order 4 for error estimation. The stability domain for the method is illustrated in Figure 3.2. Since $c_4 = 1$ and $a_{4i} = b_i$ for $i = 1, \ldots, 4$ with $b_4 = \gamma$, the last stage calculation is equal to the final solution at the end of the current step. Hereby the final evaluation in (3.2b) can be omitted and in the case of DAE problems, the algebraic variables are solved without additional computation. A- and L-stable methods with this additional characteristic are referred to as *stiffly accurate*. Stiffly accurate methods avoid the order reduction as observed by Prothero and Robinson (1974) when applied to stiff differential equations.

With the explicit first stage in (3.8), the last stage value of the current step becomes equal to the first stage value of the next step. In this way, only three intermediate values need to be calculated in each step. Also, the explicit first stage ensures high internal stage order. The stage values are themselves approximations to the solution of the DAE system, and with the design of ESDIRK34 the first and fourth stages have order 3, whereas the second and third stages have order 2. The importance of high stage order is related to DAE problems, in which the stage order influences the order of convergence in the algebraic variables (Hairer *et al.*, 1987). For ESDIRK34 the same order of convergence is obtained in the differential and algebraic variables. More details on the method are found in Alexander (2003), and important implementational aspects are discussed in Kristensen *et al.* (2004a).

3.1.2 BDF Methods

The BDF methods belong to the class of multi-step methods. They are based on numerical differentiation instead of quadrature as for Runge-Kutta methods. They have been used extensively for the solution of both ODEs and DAEs, largely due to two powerful codes: DASSL by Linda R. Petzold and LSODE by Alan C. Hindmarsh. In later chapters of this thesis, an extended version of DASSL for sensitivity integration will be used, so a brief introduction is given here.

DASSL (Petzold, 1982) solves systems of DAEs of a more general form than ESDIRK34:

$$\begin{aligned} \boldsymbol{F}(t, \boldsymbol{y}, \dot{\boldsymbol{y}}) &= \boldsymbol{0} \\ \boldsymbol{y}(t_0) &= \boldsymbol{y}_0 \\ \dot{\boldsymbol{y}}(t_0) &= \dot{\boldsymbol{y}}_0 \end{aligned} \tag{3.10}$$

in which $\boldsymbol{F}$, $\boldsymbol{y}$ and $\dot{\boldsymbol{y}}$ are n-dimensional vectors. The basic idea of BDF methods is to replace the derivative in (3.10) by a difference approximation and solve the resulting nonlinear system by an iterative method such as Newton's method. Replacing the derivative by the first order backward difference, the implicit Euler formula is obtained:

$$\boldsymbol{F}\left(t_n, \boldsymbol{y}_n, \frac{\boldsymbol{y}_n - \boldsymbol{y}_{n-1}}{h}\right) = \boldsymbol{0} \tag{3.11}$$

in which $h = t_n - t_{n-1}$. Instead of always using a first order approximation, DASSL approximates the derivative using a kth order backward differentiation formula (BDF), where k ranges from one to five. That is, the kth order BDF consists of replacing $\dot{\boldsymbol{y}}$ by the derivative of the polynomial which interpolates the computed solution at $k+1$ times $t_n, t_{n-1}, \ldots, t_{n-k}$, evaluated at t_n. Substitution in (3.10) gives:

$$\boldsymbol{F}\left(t_n, \boldsymbol{y}_n, \frac{\boldsymbol{\rho}\boldsymbol{y}_n}{h}\right) = \boldsymbol{0} \tag{3.12}$$

in which $\boldsymbol{\rho}\boldsymbol{y}_n = \sum_{i=0}^{k} \alpha_i \boldsymbol{y}_{n-i}$, where α_i, $i = 0, 1, \ldots, k$ are the coefficients of the BDF method. In principle, the resulting nonlinear system is then solved for $\boldsymbol{y}_n$ via Newton iterations. A much more detailed discussion on BDF methods and especially on the DASSL code is found in Brenan *et al.* (1996).

BDF methods such as DASSL are often implemented using an adaptable order strategy. Being multi-step methods, they start out at low order and successively build up higher order information as the solution process proceeds. This works very efficiently as long as the solution trajectories are smooth. However, one class of problems where multi-step methods are outperformed by one-step methods is that of ODEs or DAEs exhibiting frequent discontinuities. Multi-step methods must be restarted at low order after each discontinuity, whereas one-step methods such as Runge-Kutta methods recommence at higher orders rendering these methods better suited for this class of problems.

To summarize: The integration method chosen for a specific application should reflect the nature of the problem at hand. If the problem is stiff, an implicit integrator with strong stability properties should be used. If it is not stiff, the use of an explicit method is adequate. When frequent discontinuities are present, one-step methods should be used, whereas multi-step methods are advantageous for problems with long and smooth intervals. If high accuracy in the solution is required, a method with high order should be chosen. These guidelines, however, are not always useful, since for example the stiffness characteristics of a specific problem are often not known beforehand. Consequently, for reasons of reliability and robustness, implicit methods are often the default choice for many practical purposes. For the parameter estimation problem it should be stressed that the choice of equation solver depends heavily on the choice of optimization algorithm. If a multiple shooting technique is used, the obvious conclusion is not to use a multi-step integration method, since these methods suffer particularly from the frequent restarts of the integration. Instead, a one-step method should be applied.

3.2 Efficient Gradient Generation

Generation of gradient information for the optimization algorithm is a computationally demanding task. For parameter estimation problems in 'stationary' models, the gradient of an explicit function is required with respect to the parameters. For dynamical models, however, the gradient of the solution to a set of differential equations is required. This circumstance increases the complexity of the problem, and it is one of the main characteristics distinguishing parameter estimation in dynamical models from parameter estimation in stationary models.

To show how the need for derivatives of the solution to differential equations arises, the ordinary least squares fitting criterion introduced in Section 1.1.3 is considered:

$$f(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m} r_i^2(\boldsymbol{\theta}) \tag{3.13}$$

in which the residuals are defined as:

$$r_i(\boldsymbol{\theta}) = y_{c_i}(t_i, \boldsymbol{\theta}) - \tilde{y}_i \tag{3.14}$$

where $\boldsymbol{y}(t, \boldsymbol{\theta})$ and $\tilde{\boldsymbol{y}}$ denote the solution to the model equations and the measurements, respectively. If a Gauss-Newton type of method is used for the

optimization, only the first order derivative of $f(\boldsymbol{\theta})$ is required. In Section 2.1.3 it was shown that the gradient and the Hessian approximation could be expressed as $\boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{r}(\boldsymbol{\theta})$ and $\boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{J}(\boldsymbol{\theta})$, in which $\boldsymbol{J}(\boldsymbol{\theta})$ refers to the Jacobian of the residuals. The ijth element of $\boldsymbol{J}(\boldsymbol{\theta})$ is:

$$\boldsymbol{J}(\boldsymbol{\theta})_{ij} = \frac{\partial r_i(\boldsymbol{\theta})}{\partial \theta_j} = \frac{\partial y_{c_i}(t_i, \boldsymbol{\theta})}{\partial \theta_j} \tag{3.15}$$

which shows that for every measurement $(c_i, t_i, \tilde{y}_i)$, $i = 1, \ldots, m$, the derivative of the model output with respect to the parameters is required. To calculate these derivatives, one possibility is to solve the so-called *sensitivity equations*. Proceeding from the case with only ODEs present[1], the sensitivity equations can be derived by differentiating the ODE system (3.1) with respect to $\boldsymbol{\theta}$ using the chain rule and subsequently exchanging the order of differentiation. This leads to an additional set of nn_p ODEs, written in compact matrix notation as:

$$\frac{\partial}{\partial t}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}}, \qquad \frac{\partial \boldsymbol{y}(t_0, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{y}_0(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{3.16}$$

The existence of derivatives of the solution to (3.1) is given by Gronwall's theorem (Gronwall, 1919):

Theorem 3.1 (Gronwall's Theorem)
If the partial derivatives $\partial \boldsymbol{f}/\partial \boldsymbol{y}$ and $\partial \boldsymbol{f}/\partial \boldsymbol{\theta}$ exist, and are continuous in the neighborhood of the solution $\boldsymbol{y}(t, \boldsymbol{y}_0(\boldsymbol{\theta}), \boldsymbol{\theta})$, then the derivatives of the solution with respect to $\boldsymbol{\theta}$ exist, are continuous, and satisfy the linear inhomogeneous matrix differential equation (3.16).

Thus, the requirement of the optimization algorithm for derivative information has introduced n additional differential equations into the problem for each unknown parameter to be estimated. With a large number of parameters, solving the entire system of model equations plus sensitivity equations with an off-the-shelf ODE solver, disregarding the special structure of the system, is highly inefficient. To obtain some insight into possible solution strategies for the sensitivity equations, the full system is written explicitly to reveal its special structure. If $\boldsymbol{s}_i$ denotes the vector of sensitivities with respect to the ith element of $\boldsymbol{\theta}$, then the full system of model equations and sensitivity equations is:

$$\begin{aligned}
\dot{\boldsymbol{y}} &= \boldsymbol{f}(t, \boldsymbol{y}, \boldsymbol{\theta})\,, & \boldsymbol{y}(t_0, \boldsymbol{\theta}) &= \boldsymbol{y}_0(\boldsymbol{\theta}) \\
\dot{\boldsymbol{s}}_1 &= \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}\boldsymbol{s}_1 + \frac{\partial \boldsymbol{f}}{\partial \theta_1}\,, & \boldsymbol{s}_1(t_0, \boldsymbol{\theta}) &= \frac{\partial \boldsymbol{y}_0(\boldsymbol{\theta})}{\partial \theta_1} \\
&\vdots & &\vdots \\
\dot{\boldsymbol{s}}_{n_p} &= \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}\boldsymbol{s}_{n_p} + \frac{\partial \boldsymbol{f}}{\partial \theta_{n_p}}\,, & \boldsymbol{s}_{n_p}(t_0, \boldsymbol{\theta}) &= \frac{\partial \boldsymbol{y}_0(\boldsymbol{\theta})}{\partial \theta_{n_p}}
\end{aligned} \tag{3.17}$$

[1] With algebraic equations present, an extra set of algebraic sensitivity equations are obtained.

The system of equations (3.17) contains one subsystem of n nonlinear ODEs and n_p subsystems of the same size of linear sensitivity equations. Direct solution of (3.17) with an implicit solver requires the Jacobian of the system:

$$Jac = \begin{bmatrix} \dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}} & 0 & \cdots & 0 \\ \dfrac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{y}^2}\boldsymbol{s}_1 + \dfrac{\partial^2 \boldsymbol{f}}{\partial \theta_1 \partial \boldsymbol{y}} & \dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}} & 0.. & 0 \\ \vdots & 0 & \ddots & 0 \\ \dfrac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{y}^2}\boldsymbol{s}_{n_p} + \dfrac{\partial^2 \boldsymbol{f}}{\partial \theta_{n_p} \partial \boldsymbol{y}} & 0 & ..0 & \dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}} \end{bmatrix} \tag{3.18}$$

This special structure of the Jacobian is exploited in several of the solution strategies discussed in the following. One observation is made from (3.18), namely that the overall system has the same eigenvalues as $\partial \boldsymbol{f}/\partial \boldsymbol{y}$, just with different multiplicity. This means that the sensitivity equations inherit the stiffness character of the model equations.

It should be mentioned that part of a parameter estimation problem may be to estimate the initial conditions of the states in a model. Since the right-hand-side function in an ODE system (normally) has no explicit dependence on the initial conditions, the corresponding sensitivity equations are homogeneous. The requirement for sensitivities with respect to initial conditions also arises if a multiple shooting optimization strategy is used, since the initial conditions of each shooting interval are introduced as extra parameters in the problem.

Example 3.1 (Sensitivity Calculation in Gas-Oil Cracking Model)
To illustrate the derivation of sensitivity equations, the gas-oil cracking model is considered again. In the previous examples when testing optimization algorithms, the solution and the derivatives with respect to parameters were assumed available. The (modified) gas-oil cracking model is:

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} -(k_1 + k_2) y_1^2 \\ k_1 y_1^2 - k_2 y_2 \end{bmatrix}, \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3.19}$$

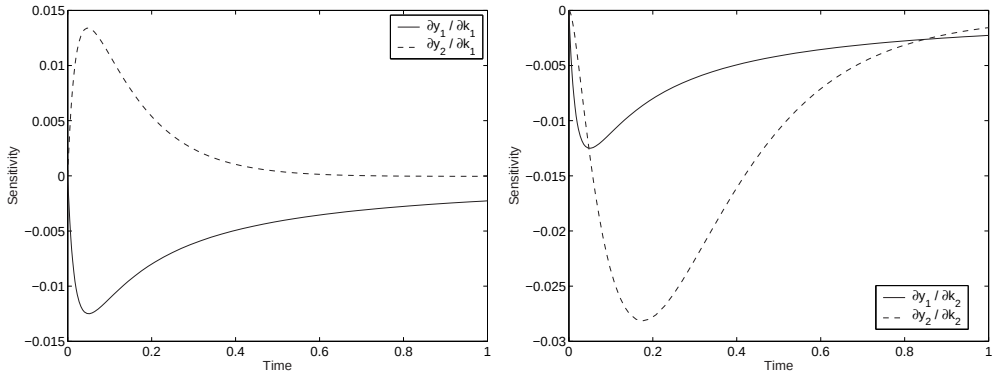
k_1 and k_2 are considered unknown. Applying (3.17), the sensitivity equations are readily derived:

$$\begin{aligned} \dot{s}_{11} &= -2(k_1 + k_2) y_1 s_{11} - y_1^2 \,, & s_{11}(0) &= 0 \\ \dot{s}_{21} &= 2k_1 y_1 s_{11} - k_2 s_{21} + y_1^2 \,, & s_{21}(0) &= 0 \\ \dot{s}_{12} &= -2(k_1 + k_2) y_1 s_{12} - y_1^2 \,, & s_{12}(0) &= 0 \\ \dot{s}_{22} &= 2k_1 y_1 s_{12} - k_2 s_{22} - y_2 \,, & s_{22}(0) &= 0 \end{aligned}$$

in which s_{ij} denotes the sensitivity of state i with respect to parameter j. These equations need to be integrated each time the gradient of the objective function is required by the optimization algorithm. The sensitivity trajectories for the final set of parameters ($k_1 = 12$ and $k_2 = 8$) are plotted in Figure 3.3.

As the name indicates, the sensitivities measure the 'sensitivity' of the model states with respect to changes in the parameters. For example, Figure 3.3a shows that an increase in k_1 at $t = 0$ leads to an initial decrease in y_1 as expected from the model equations (3.19).

■



(a) Sensitivities with respect to k_1.

(b) Sensitivities with respect to k_2.

Figure 3.3. Plots of trajectories for parameter sensitivities in gas-oil cracking model.

3.2.1 Internal versus External Numerical Differentiation

Before attempting to solve the sensitivity equations, some alternative approaches are considered. The most straightforward approach is to approximate the sensitivities using a finite difference formula. The parameters of the system are perturbed successively by a small tolerance and the model equations are integrated again:

$$\frac{\partial \boldsymbol{y}}{\partial \theta_j} \approx \frac{\boldsymbol{y}\left(t, (\theta_1, \ldots, \theta_j + \Delta\theta_j, \ldots, \theta_{n_p})\right) - \boldsymbol{y}\left(t, (\theta_1, \ldots, \theta_j, \ldots, \theta_{n_p})\right)}{\Delta\theta_j} \tag{3.20}$$

This approach is known as *external numerical differentiation* (END). With a first order approximation like (3.20) it requires $n_p + 1$ solutions of the model equations each time the derivatives are required. The choice of the perturbation $\Delta\theta_j$ is important for the accuracy obtained. If it is chosen too large, the difference quotient no longer provide a good approximation to the derivative, since the truncated higher order terms in the Taylor expansion become significant. However, if it is too small, the subtraction in the numerator of (3.20) is subject to loss of significance, since only a finite number of digits are stored in the computation.

Another problem of END arises due to the adaptive nature of modern differential equation solvers. The output from such solvers is usually a discontinuous function of the initial values and parameters (Leineweber, 1995). If, for example, one of the parameters is varied, jumps of the order of the integrator tolerance have to be expected, which can cause severe errors in the numerical estimation of derivatives. These discontinuities are due to the adaptive nature of the stepsize selection mechanisms. They arise when a small change of an initial condition or parameter causes the integration algorithm to follow a different path of execution, e.g. employ another sequence of stepsizes. Thus, unless the integrator tolerance is set extremely low (close to machine precision), poor estimates of the derivatives are obtained (Leineweber, 1995). Solving the

model equations with a tolerance close to machine precision, however, is computationally very expensive, which renders the whole END approach inefficient.

To avoid the discontinuity problem, a finite difference approach for computation of the derivatives should adopt an *internal numerical differentiation* (IND) strategy (Bock, 1981). The basic principle of IND is to "freeze" the discretization such that the unperturbed and perturbed trajectories are calculated using the same sequence of stepsizes. That is, the same path of execution (same sequence of stepsizes) is used in the two solutions differenced to estimate the derivatives. Using this approach, it is no longer necessary to perform the integrations with an extreme tolerance in order to obtain reliable finite difference approximations (Leineweber, 1995). If an adaptive stepsize algorithm is used, the stepsizes chosen by the algorithm for the calculation of the unperturbed trajectories are used again for the calculation of the perturbed trajectories. The IND approach computes an approximation to the well-defined, smooth derivative of one single discretization scheme, whereas the END approach combines two (generally different) discretizations to obtain a finite difference approximation, which is meaningful only if the discretization errors involved are very small.

The IND approach is much more efficient than END. However, joint integration of the model equations and sensitivity equations with a specifically tailored algorithm is even more efficient. If an explicit integration algorithm is used, IND may be advantageous since some of the methods discussed in the next section for solution of the sensitivity equations rely on using information only required by implicit integrators (Jacobian of the equation system etc.).

3.2.2 Solving the Sensitivity Equations

In this section efficient strategies for solving the sensitivity equations are discussed. Solving these equations is referred to as *forward* or *direct* sensitivity analysis as opposed to *adjoint* sensitivity analysis. The adjoint approach considers calculating the sensitivities of a scalar response function:

$$f(\boldsymbol{\theta}) = \int_{t_0}^{t_f} g(t, \boldsymbol{y}(t, \boldsymbol{\theta}), \boldsymbol{\theta}) dt \tag{3.21}$$

Forward sensitivity analysis is best suited for the situation of finding the sensitivities of a potentially large number of variables with respect to a small number of parameters, whereas adjoint sensitivity analysis is best suited to the complimentary situation of finding the sensitivities of a scalar response function with respect to a large number of parameters (Cao *et al.*, 2002). Calculating the sensitivities of a scalar response function is exactly the requirement in parameter estimation problems, so the adjoint methods seems promising at first. However, in Gauss-Newton type optimization methods the sensitivity coefficients obtained in the forward approach are used directly to approximate the Hessian. This observation implies that the adjoint sensitivity method must be used in conjunction with an optimization method which uses an alternative approximation to the Hessian (e.g. a quasi-Newton method with BFGS updating). Law and Sharma (1997) have found by numerical comparison that a Gauss-Newton

strategy with forward sensitivity analysis is superior to a quasi-Newton approach with adjoint sensitivity analysis. The adjoint method is not discussed further in this review, but details can be found in the relevant literature (Cao *et al.*, 2002, 2003; Sandu *et al.*, 2003).

As mentioned earlier, the naive approach in forward sensitivity analysis is to solve the combined system (3.17) directly, which in the case of implicit integrators requires calculation and factorization of the Jacobian (3.18) with dimension $n(n_p + 1) \times n(n_p + 1)$. More efficient approaches exist that exploit the linearity of the sensitivity equations. Many of the existing algorithms for joint integration of the model and sensitivity equations are extensions to the BDF based methods LSODE and DASSL, and the terminology used to classify the sensitivity methods was developed in the context of these methods. The approaches used for forward sensitivity analysis may roughly be categorized into three groups:

Staggered direct method. This method, presented by Caracotsios and Stewart (1985), is also referred to as the *decoupled direct method*. A similar approach was proposed by Leis and Kramer (1988). The state and sensitivity equations are integrated jointly, but in each step of the integration the nonlinear state equations are solved first, observing the one-way coupling of the Jacobian (3.18) for the combined system. Once the solution to the state equations has been updated, the n_p subsystems of sensitivity equations are solved directly exploiting the linearity. The sequence of actions performed in each integration step is:

1. Integrate state equations from t_n to t_{n+1}.
2. Perform initial error control on state variables.
3. If error control accepted: Evaluate the Jacobian[2] at t_{n+1} and LU factorize it.
4. Integrate each subsystem of sensitivity equations from t_n to t_{n+1}. With a fresh factorization of the Jacobian, this is equivalent to one Newton iteration, since the sensitivity equations are linear.
5. If required: Perform additional error control on sensitivity variables.

Because of the direct solution of the sensitivity equations, a factorization of the Jacobian is required in each step. The error control can either include state variables only (partial error control), or both state and sensitivity variables (full error control). This is not specific to the staggered direct method.

Simultaneous corrector method. The simultaneous corrector method, as presented by Maly and Petzold (1996), solves the combined system (3.17) as one nonlinear system without exploiting linearity. The Jacobian of the combined system (3.18) is approximated by its block diagonal part. The individual steps are:

1. Integrate state and sensitivity equations simultaneously from t_n to t_{n+1}. Each overall iteration consists of:

[2] Unless otherwise stated, the Jacobian refers to $\partial \boldsymbol{f}/\partial \boldsymbol{y}$.

Perform one iteration for the state equations.

Perform one iteration for each of the n_p sensitivity subsystems.

2. Perform error control on state variables.
3. If required: Perform additional error control on sensitivity variables.

Since an iterative procedure is applied to solve the sensitivity equations, an adaptive strategy can be used to choose when to refactorize the Jacobian. Arguing that one factorization per integration step is too expensive for large scale problems, Maly and Petzold (1996) claim that the simultaneous corrector method is more efficient than the staggered direct method. A drawback of a simultaneous approach compared to a staggered approach is, however, that potential error control failures on the state variables render the effort spent integrating the sensitivity equations fruitless.

Staggered corrector method. This method, proposed by Feehery *et al.* (1997), constitutes a compromise between the staggered direct and the simultaneous corrector methods. As for the staggered direct method, the state equations are solved first. Once an acceptable solution has been obtained, a separate iteration procedure is used to converge the sensitivity equations:

1. Integrate state equations from t_n to t_{n+1}.
2. Perform initial error control on state variables.
3. If error control accepted: Converge sensitivity equations to obtain a solution at t_{n+1} using an iterative scheme.
4. If required: Perform additional error control on sensitivity variables.

This method avoids solving the sensitivity equations in steps where the error control fails for the state variables, but still maintains the need for relatively few Jacobian updates and factorizations.

Software packages incorporating the ideas described above include:

DASPK – (Maly and Petzold, 1996) Extension of DASSL for sensitivity analysis. The package includes all three methods described above with an option for full or partial error control.

CVODES – (Hindmarsh and Serban, 2002) Package for sensitivity analysis of ODEs. Employs a BDF method for stiff problems and an Adams-Moulton method for non-stiff problems. The package includes all three methods described above with an option for full or partial error control.

ODESSA – (Leis and Kramer, 1988) Extension of LSODE for sensitivity analysis. Uses the staggered direct method with an option for full or partial error control.

ESDIRK34 – (Kristensen *et al.*, 2004a) Based on one-step method of the Runge-Kutta family. Uses the staggered direct method with partial error control.

sLIMEX – (Schlegel *et al.*, 2004) Based on one-step extrapolation method. Uses the simultaneous corrector method with an option for full or partial error control.

The work of Caracotsios and Stewart (1985) and Feehery *et al.* (1997) on the staggered direct and the staggered corrector methods, respectively, was also based on the DASSL code. The ESDIRK34 code was described in Section 3.1.1.1. The sensitivity method employed is essentially a staggered direct method, but the numerical implementation differs from the BDF based approaches. In ESDIRK34 the sensitivities are derived by differentiating the numerical scheme with respect to the initial conditions and parameters (see outline in Appendix A.1). Efficiency is obtained by reusing the Jacobian $\partial \boldsymbol{f}/\partial \boldsymbol{y}$ and its factorization, which is computed anyway for the integration of the state equations. Since DASPK solves DAE problems of the more general form (3.10), a similar reuse of information is not possible. DASPK requires user-implemented sensitivity equations (although with an option to use automatic differentiation), whereas ESDIRK34 only requires implementation of the Jacobian and partial derivatives with respect to parameters, $\partial \boldsymbol{f}/\partial \boldsymbol{\theta}$. Further details on implementation of ESDIRK34 are found in Kristensen *et al.* (2004a).

Most of the approaches reported in the literature are based on multi-step methods. As discussed in Section 3.1, one-step methods have an inherent advantage over multi-step method when frequent discontinuities are present, since multi-step methods have to revert to low order at each restart of the integration. A discontinuous solution trajectory arises in parameter estimation problems when a multiple shooting based optimization algorithm is used. In later parts of this thesis comparison is made between the one-step method ESDIRK34 and the multi-step method DASPK.

4

Summary

In this thesis a practical approach to parameter estimation is taken emphasizing the numerical aspects involved. With some preliminary comments on different methods of estimation and statistical aspects, the two main elements of parameter estimation in dynamical systems were discussed: The optimization algorithm and the differential equation solver. Chapter 2 covered the basic principles of unconstrained and constrained optimization focusing on methods for nonlinear least squares. All the methods presented are gradient-based methods requiring the gradient of the objective function. Efficient methods for solving the underlying differential equations and generating gradient information were discussed in Chapter 3.

Ideally, an optimization algorithm and an equation solver applicable to all possible problems would be available. This is, however, not the case. Therefore, it is important to understand the advantages and disadvantages of the individual algorithms such that a qualified choice of optimizer and differential equation solver can be made for each desired application. Numerous reports are found in the literature (e.g. Khorasheh *et al.*, 2002; Issanchou *et al.*, 2003), in which the numerical algorithms apparently have been applied purely as black boxes with no particular reasoning regarding the choice of method. The computationally most intensive task in parameter estimation is the generation of gradient information, a task that often seems to suffer from naive approaches, in which the gradients are either approximated by finite difference schemes or calculated by solving the sensitivity equations with an off-the-shelf ODE solver, disregarding the special structure of these equations, and therefore ending up with an inefficient overall algorithm.

In conclusion: Possible improvements still remain in the development of efficient and robust algorithms for parameter estimation in dynamical systems, particularly regarding the role of the differential equation solver.

Progress Report

5

Benchmarking

The main purpose of this progress report is to assess different methods for parameter estimation in dynamical systems. As mentioned in the previous report, the optimization algorithm and the differential equation solver are the essential components required for the practical solution of parameter estimation problems. The performance of different optimization algorithms are compared through benchmark tests. Also, comparison is made between the different methods for sensitivity computation discussed in Section 3.2.2. Based on these comparative studies, advantages and disadvantages of the individual approaches are identified, and progress is made towards an efficient tool applicable to general parameter estimation problems in dynamical systems described by ODEs or DAEs.

In Section 5.1 the individual comparisons are made. The gas-oil cracking problem introduced in the previous report is used as test example, and simulated values corrupted with random noise are used as "measurements". Comparison is made between three different optimization algorithms with ESDIRK34 as the differential equation solver. Furthermore, using a general purpose SQP method the performance of a standard least squares formulation is compared to the transformed formulation proposed by Schittkowski (1988).

Section 5.2 covers the comparison between the two differential equation solvers: The Runge-Kutta method ESDIRK34 and the BDF method DASPK. The staggered direct, staggered corrector and simultaneous corrector methods for sensitivity integration are compared, still using the gas-oil cracking problem as test example.

5.1 Optimizer Performance

The first comparison made is between three different optimization algorithms. Below the individual algorithms are briefly introduced along with the performance characteristics measured in the test. The algorithms are all available as Fortran codes. Details on implementation of the parameter estimation problems are given in Appendix A. For this preliminary comparison no alterations are made to the optimization algorithms, but possible improvements are identified. The algorithms considered in the test are:

LMDER: This code, which is part of the Minpack package available from the Netlib repository (www.netlib.org), was first described in Moré (1977). The code is an implementation of the Levenberg-Marquardt algorithm

(c.f. Section 2.1.3.3) solving the standard nonlinear least squares problem:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{r}(\boldsymbol{\theta})^T\boldsymbol{r}(\boldsymbol{\theta}) \tag{5.1}$$

in which $\boldsymbol{r} : \mathbb{R}^{n_p} \mapsto \mathbb{R}^m$ is the vector of residual functions. LMDER uses the subroutine `FCN` supplied by the user, which calculates the residual functions and the Jacobian. `FCN` is called with a flag indicating wether to calculate the residuals or the Jacobian. When estimating parameters in systems described by differential equations, it is inefficient to separate these two tasks, since the information required for the Jacobian calculation is already present when calculating the residuals. Thus, a modification of LMDER is desirable, which is addressed in Chapter 6.

LMDER uses three different termination criteria and it terminates whenever one of them is met. The first criterion measures the relative error of the objective function. Termination occurs if both the actual and the predicted relative reductions are less than a user specified tolerance ε_1. That is:

$$\frac{f(\boldsymbol{\theta}) - f(\boldsymbol{\theta} + \boldsymbol{h})}{f(\boldsymbol{\theta})} \leq \varepsilon_1 \quad \text{and} \quad \frac{L(\mathbf{0}) - L(\boldsymbol{h})}{L(\mathbf{0})} \leq \varepsilon_1 \tag{5.2}$$

in which $L(\cdot)$ is the local linear model used in the Levenberg-Marquardt algorithm (c.f. Algorithm 1 in Section 2.1.3.3). The second termination criterion measures the relative error of the desired solution. Termination occurs when the relative error between two consecutive iterates is at most ε_2:

$$\|\boldsymbol{h}\| \leq \varepsilon_2 \|\boldsymbol{\theta}\| \tag{5.3}$$

Finally, the third termination criterion measures the orthogonality between the residual vector and the columns of the Jacobian. Termination occurs when the cosine of the angle between the residual vector and any column of the Jacobian is at most ε_3 in absolute value:

$$|\cos(\varphi)| = \frac{|\boldsymbol{r}(\boldsymbol{\theta}) \cdot \boldsymbol{J}(\boldsymbol{\theta})_{:,i}|}{\|\boldsymbol{r}(\boldsymbol{\theta})\| \cdot \|\boldsymbol{J}(\boldsymbol{\theta})_{:,i}\|} \leq \varepsilon_3 \tag{5.4}$$

in which the notation $\boldsymbol{J}(\boldsymbol{\theta})_{:,i}$ denotes the ith column of the Jacobian and φ is the angle between the residual vector and the ith column of the Jacobian. The inequality (5.4) should hold for all columns of the Jacobian. This criterion is equivalent to taking the infinity norm of the gradient of the objective function, $\boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{r}(\boldsymbol{\theta})$, except from some scaling by the norms of $\boldsymbol{r}(\boldsymbol{\theta})$ and $\boldsymbol{J}(\boldsymbol{\theta})_{:,i}$.

LMDER allows the user to provide scaling factors for the parameters. By default the parameters are scaled by the norms of the columns of the initial Jacobian.

NPSOL: This code (Gill *et al.*, 1986) is designed for minimization of a general smooth function subject to constraints. It is the only code in the test, which is not designed specifically for nonlinear least squares problems.

NPSOL solves nonlinear programming problems of the form:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$$
$$s.t. \quad \boldsymbol{l} \leq \begin{Bmatrix} \boldsymbol{\theta} \\ \boldsymbol{A}_L\boldsymbol{\theta} \\ \boldsymbol{c}(\boldsymbol{\theta}) \end{Bmatrix} \leq \boldsymbol{u} \tag{5.5}$$

in which $\boldsymbol{A}_L$ is an $m_L \times n_p$ constant matrix defining the general linear constraints and $\boldsymbol{c}(\boldsymbol{\theta})$ is a vector of dimension m_N of nonlinear constraint functions. That is, simple bound constraints, general linear constraints, and general nonlinear constraints are treated separately by NPSOL. The algorithm implemented in NPSOL is an active set SQP method with line search and BFGS updating. Each major iteration consists of computing a search direction, determining a step length that produces a sufficient decrease in an augmented Lagrangian merit function[1], and, finally, updating the approximation to the Hessian of the Lagrangian function. Details on the algorithm can be found in Gill *et al.* (1986).

NPSOL uses two subroutines, `CONFUN` and `OBJFUN`, provided by the user, which calculate the nonlinear constraint functions and the objective function (and its gradient), respectively. Since NPSOL is not designed specifically for nonlinear least squares problems, the special structure of these problems is not exploited. The first comparison considers application of NPSOL to a standard formulation of the least squares problem, whereas the comparison performed in Example 5.2 considers a simple transformation of the least squares problem as proposed by Schittkowski (1988).

The NPSOL package has a large number of advanced settings, the details of which are not explained here. NPSOL terminates with an optimal solution if three different criteria are met simultaneously: (i) the sequence of iterates has converged (criterion similar to (5.3)), (ii) the norm of the gradient is sufficiently small, and (iii) the norm of the residuals of the constraints in the predicted active set is small enough. Thus, the mechanism for termination is different from LMDER in that three criteria must be satisfied at the same time.

NL2SOL: This code, described in Dennis *et al.* (1981), was developed for nonlinear least squares problems with the specific motivation of developing an algorithm, which in the large residual case would be more reliable than the Gauss-Newton or Levenberg-Marquardt methods. NL2SOL solves least squares problems of the form (5.1). Remembering that the Hessian of the least squares objective function is:

$$\boldsymbol{H} = \boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{J}(\boldsymbol{\theta}) + \sum_{i=1}^{m} r_i(\boldsymbol{\theta})\nabla^2 r_i(\boldsymbol{\theta}) \tag{5.6}$$

the Gauss-Newton approximation consists of neglecting the second order terms in (5.6). NL2SOL uses an augmented Gauss-Newton approximation, in which approximations to the second order terms are included.

[1] See Nocedal and Wright (1999) for details.

Thus, the resulting overall approximation is:

$$\boldsymbol{B} = \boldsymbol{J}^T\boldsymbol{J} + \boldsymbol{S} \tag{5.7}$$

In each iteration $\boldsymbol{S}$ is updated by adding a correction term containing information about the problem at the new point. By keeping an approximation to the second order terms of the Hessian, improved performance is achieved on problems with large residuals or strong nonlinearities.

NL2SOL uses two subroutines, `CALCR` and `CALCJ`, supplied by the user for calculation of the residual functions and the Jacobian. Again, this separation of the computational tasks is inefficient for parameter estimation problems in differential equation systems.

Although not as extensive as NPSOL, the NL2SOL package has many advanced settings that allow the user to tune the optimization algorithm. Except for the tolerance parameters, default settings (see Dennis *et al.*, 1981) are used in the comparison. NL2SOL uses five different termination criteria: *absolute function convergence*, *relative function convergence*, $\boldsymbol{\theta}$–*convergence*, *singular convergence* and *false convergence*. The specific mechanisms used for the individual criteria to be satisfied are detailed in Dennis *et al.* (1981). Relative function convergence and $\boldsymbol{\theta}$–convergence are comparable to (5.2) and (5.3), respectively. NL2SOL uses a quadratic model of the objective function in each step of the iteration, so instead of the linear model used by LMDER in (5.2) a quadratic model is used to compute the predicted decrease. Absolute function convergence is simply a measure of the value of the objective function, and termination occurs with this criterion satisfied if:

$$f(\boldsymbol{\theta}) \leq \varepsilon_0 \tag{5.8}$$

Singular convergence and false convergence are used to accommodate situations, where none of the previously described tests are satisfied. For example, false convergence can occur if the specified tolerances are too strict for the accuracy to which $f(\boldsymbol{\theta})$ and $\boldsymbol{J}(\boldsymbol{\theta})$ are being computed.

Making a fair comparison between the three optimization algorithms just described is a difficult task, since differences in, for example, termination criteria exist. Based on a single test example, it is impossible to draw certain conclusions about the advantages and disadvantages of the three algorithms in question. However, the intention here is not to give a thorough comparison considering all possible applications, but rather to demonstrate the use of the algorithms and to provide guidelines for further investigation. In order to give as detailed a comparison as possible, a number of different performance characteristics are included for the optimizer as well as for the differential equation solver. Descriptions of the measured quantities are listed in Table 5.1. A few comments are required: NJAC is the number of evaluations of the Jacobian required to solve the nonlinear equations arising due to the use of an implicit integration method (DASPK or ESDIRK34). NJACT is the total number of Jacobian evaluations, i.e. evaluations required to solve the nonlinear equations

Abbreviation	Meaning
NIT	Number of major iterations. Iterations performed in an "inner loop" are not included.
NOBJ	Number of objective function evaluations.
NGRD	Number of gradient evaluations.
NSTEP	Total number of integration steps (successful + rejected).
NFUN	Number of function evaluations.
NJAC	Number of Jacobian evaluations due to nonlinear iterations.
NJACT	Total number of Jacobian evaluations, i.e. evaluations for iteration matrices and for sensitivity residuals.
NLU	Number of *LU* factorizations.
NBACK	Number of back substitutions.
NSENS	Number of right hand side evaluations of sensitivity equations.
CPU Time	The CPU time (measured in seconds) applies to an average of 100 runs on a 450MHz AMD K6-2 computer. The time is measured using the function `CPU_TIME`, which is part of the Compaq Visual Fortran compiler.
NCLOCK	Total number of clock cycles required (millions).

Table 5.1. Performance characteristics measured in the benchmark tests. The first section of the table holds characteristic quantities for the optimizer, the second section holds characteristics for the equation solver (accumulated values for all calls made to the equation solver during optimization), whereas the last two entries relate to the overall performance. Not all quantities are reported in each comparison.

plus evaluations required when computing the right-hand-side of the sensitivity equations[2]. The CPU time is measured with the function `CPU_TIME`, which is part of the Compaq Visual Fortran compiler. Simple tests show that this measurement is subject to "noise", so the reported value is an average of 100 runs. Furthermore, the total number of clock cycles is listed, which allows for a validation of the CPU time measurement.

In the following example a comparison is made between LMDER, NPSOL and NL2SOL:

Example 5.1 (Performance Comparison Between Optimization Algorithms) The gas-oil cracking problem encountered several times in the previous report is used again as test example:

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} -(k_1 + k_3)y_1^2 \\ k_1 y_1^2 - k_2 y_2 \end{bmatrix}, \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{5.9}$$

This time all three parameters are estimated. The initial conditions are assumed to be known without error. "Measurements" are obtained at 10 equidistant points in time by simulation using the following parameter values:

$$k_1 = 12, \quad k_2 = 8, \quad k_3 = 8 \tag{5.10}$$

[2] In ESDIRK34 NJAC and NAJCT are the same, which is not the case in DASPK.

The data are perturbed with random noise drawn from a normal distribution with zero mean and variance $\sigma^2 = 0.05^2$. The data are plotted in Figure 5.1 along with the solution trajectories corresponding to the optimal parameters computed in the optimization and the trajectories corresponding to the original parameters (5.10).

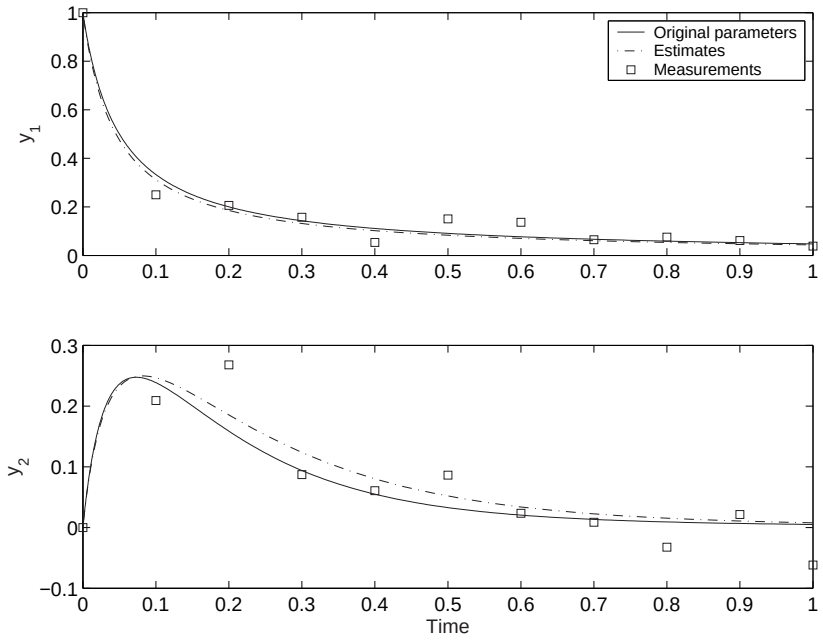


Figure 5.1. Measurements plotted along with the solution trajectories computed using the original and the optimal parameters, respectively. The parameter set from LMDER was used.

The least squares problem may now be formulated as:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{1}{2} \sum_{j=1}^{2} \sum_{i=1}^{10} w_{ij}^2 \left(y_j(t_i, \theta) - \tilde{y}_{ij} \right)^2 \tag{5.11}$$

Each residual function has an associated weight. In this comparison all weights are chosen to 1. The differential equations are solved using ESDIRK34 with the absolute and relative tolerances set to 10^{-6}. The choice of tolerance in the equation solver is studied in much more detail in Chapter 6. ESDIRK34 uses the staggered direct method for calculating the sensitivities. The sensitivity equations need not be implemented directly, only the Jacobian and the partial derivatives with respect to the parameters must be specified:

$$\frac{d\boldsymbol{f}}{d\boldsymbol{y}} = \begin{bmatrix} -2(k_1 + k_3)y_1 & 0 \\ 2k_1 y_1 & -k_2 \end{bmatrix}, \qquad \frac{d\boldsymbol{f}}{d\boldsymbol{\theta}} = \begin{bmatrix} -y_1^2 & 0 & -y_1^2 \\ y_1^2 & -y_2 & 0 \end{bmatrix} \tag{5.12}$$

The tolerance parameters in the individual optimization algorithms are as far as possible specified such that the same accuracy is required in the solution for each algorithm. For LMDER $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 10^{-4}$ in (5.2)–(5.4). The relative function convergence and $\boldsymbol{\theta}$-convergence in NL2SOL are set to 10^{-4}, and in NPSOL the criteria corresponding to (5.3) and (5.4) are both set to 10^{-4} [3].

[3] Due to the particular implementation of the termination criteria, the setting "Optimality Tolerance" must be set to 10^{-8} (see Gill *et al.*, 1986).

Code	**LMDER**	**NPSOL**	**NL2SOL**
NIT	7	36	7
NOBJ	8	37	8
NGRD	7	37	8
NSTEP	791	2058	836
NFAIL	0	0	0
NFUN	8998	23503	9524
NJAC	791	2058	836
NLU	791	2058	836
NBACK	7570	23133	8110
$f(\boldsymbol{\theta}^*)$	$0.177566 \cdot 10^{-1}$	$0.177569 \cdot 10^{-1}$	$0.177568 \cdot 10^{-1}$
k_1	11.6866	11.7382	11.7323
k_2	5.8897	5.9314	5.9284
k_3	10.3454	10.3002	10.3029
CPU Time	0.215	0.591	0.220
NCLOCK	101.3	328.0	103.0

Table 5.2. Performance statistics for three different optimization algorithms when applied to the parameter estimation problem in the gas-oil cracking model. ESDIRK34 was used for solving the equations and generating gradient information.

Internal scaling is allowed in all algorithms. The following starting guesses for the parameters are provided:

$$\boldsymbol{\theta_0} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{5.13}$$

The performance statistics for the three optimization algorithms are summarized in Table 5.2. LMDER and NL2SOL terminate with the relative function convergence criterion satisfied, whereas all criteria are satisfied in NPSOL (condition for termination). The most distinctive feature of Table 5.2 is the apparent inefficiency of NPSOL compared to LMDER and NL2SOL. All statistics for LMDER and NL2SOL are comparable. All three final objective functions share the first 5 decimals in common suggesting that the relative accuracy required has been achieved. Small differences exist in the parameter estimates, probably caused by correlation of the parameters as seen from Table 5.3, which shows a fairly strong correlation between k_1 and k_2 and between k_2 and k_3. The estimated values of the parameters differ from the true parameter values. Further experiments show that the true parameter values are approached, as expected, if the variance of the measurement errors is reduced.

	k_1	k_2	k_3
k_1	1.000		
k_2	0.718	1.000	
k_3	-0.552	-0.774	1.000

Table 5.3. Estimated correlation matrix for the parameters. The estimates are based on the final Jacobian and objective function returned from LMDER.

Regarding the statistics for the equation solver, one particular observation is made. Although NPSOL needs 5 times the number of iterations compared to LMDER and NL2SOL, the work performed by the equation solver is only increased by a factor of about 2.5. The reason behind this is that NPSOL, unlike the two other algorithms, evaluates the objective function and its gradient during one call to the user supplied subroutine. As mentioned earlier, this is a desirable property, since the information required for solving the sensitivity equations is already available during the solution of the system equations.

The poor performance of NPSOL is expected since it is not designed specifically for least squares problems, and hence does not exploit the special structure of the least squares objective function. Compared to LMDER and NL2SOL, the initial approximation to the Hessian does not resemble the actual Hessian and poorer performance is therefore expected. A simple transformation, however, of the least squares problem into a general NLP problem will improve the efficiency of the SQP approach as illustrated in the next example. ■

The above example showed that straightforward application of a general purpose SQP method to a least squares problem can be inefficient. In Section 2.2.2 a simple transformation proposed by Schittkowski (1988) was briefly introduced. Proceeding from the unconstrained least squares problem:

$$\min_{\boldsymbol{\theta}} \; f(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m} r_i^2(\boldsymbol{\theta}) \tag{5.14}$$

the transformation consists of introducing m additional variables $\boldsymbol{z} = [z_1, \ldots, z_m]^T$ and m equality constraints of the form:

$$r_i(\boldsymbol{\theta}) - z_i = 0, \quad i = 1, \ldots, m \tag{5.15}$$

The transformed problem then reads:

$$\begin{aligned} &\min_{\bar{\boldsymbol{\theta}}} \; \frac{1}{2}\boldsymbol{z}^T\boldsymbol{z} \\ &s.t. \quad \boldsymbol{r}(\boldsymbol{\theta}) - \boldsymbol{z} = \boldsymbol{0} \end{aligned} \tag{5.16}$$

in which $\bar{\boldsymbol{\theta}} = [\boldsymbol{\theta}, \boldsymbol{z}]^T \in \mathbb{R}^{n_p+m}$. Thus, the parameter vector has been augmented with m additional elements that must be estimated along with the original parameters. Intuitively, it might seem strange to increase the complexity of the problem in this way, but Schittkowski (1988) shows that the computational requirements are comparable to tailored least squares codes. Applying an SQP method to (5.16), the QP subproblem may be written as:

$$\begin{aligned} &\min_{\bar{\boldsymbol{h}}} \; \frac{1}{2}\bar{\boldsymbol{h}}^T\nabla^2_{\bar{\boldsymbol{\theta}}}\mathcal{L}(\bar{\boldsymbol{\theta}}, \boldsymbol{\lambda})\bar{\boldsymbol{h}} + \nabla\bar{f}(\bar{\boldsymbol{\theta}})^T\bar{\boldsymbol{h}} \\ &s.t. \quad \nabla\bar{\boldsymbol{c}}(\bar{\boldsymbol{\theta}})\bar{\boldsymbol{h}} + \bar{\boldsymbol{c}}(\bar{\boldsymbol{\theta}}) = \boldsymbol{0} \end{aligned} \tag{5.17}$$

in which $\bar{\boldsymbol{h}} = [\boldsymbol{h}, \boldsymbol{d}]^T$ denotes the computed direction in $\boldsymbol{\theta}$ and $\boldsymbol{z}$, $\bar{f}(\bar{\boldsymbol{\theta}}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{z}$ and $\bar{\boldsymbol{c}}(\bar{\boldsymbol{\theta}}) = \boldsymbol{r}(\boldsymbol{\theta}) - \boldsymbol{z}$. The bar is introduced to avoid confusion with the notation of Section 2.2.2. In practice, an approximation is used for the Hessian of the Lagrangian function in (5.17). The Lagrangian function is defined as:

$$\begin{aligned} \mathcal{L}(\bar{\boldsymbol{\theta}}, \boldsymbol{\lambda}) &= \bar{f}(\bar{\boldsymbol{\theta}}) - \boldsymbol{\lambda}^T\bar{\boldsymbol{c}}(\bar{\boldsymbol{\theta}}) \\ &= \frac{1}{2}\boldsymbol{z}^T\boldsymbol{z} - \boldsymbol{\lambda}^T\left(\boldsymbol{r}(\boldsymbol{\theta}) - \boldsymbol{z}\right) \end{aligned} \tag{5.18}$$

The first and second derivatives with respect to $\bar{\boldsymbol{\theta}}$ are:

$$\nabla_{\bar{\boldsymbol{\theta}}}\mathcal{L}(\bar{\boldsymbol{\theta}},\boldsymbol{\lambda}) = \begin{bmatrix} -\nabla \boldsymbol{r}(\boldsymbol{\theta})\boldsymbol{\lambda} \\ \boldsymbol{z}+\boldsymbol{\lambda} \end{bmatrix}, \qquad \nabla^2_{\bar{\boldsymbol{\theta}}}\mathcal{L}(\bar{\boldsymbol{\theta}},\boldsymbol{\lambda}) = \begin{bmatrix} \boldsymbol{S}(\boldsymbol{\theta}) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{5.19}$$

in which:

$$\boldsymbol{S}(\boldsymbol{\theta}) = -\sum_{i=1}^{m} \lambda_i \nabla^2 r_i(\boldsymbol{\theta}) \tag{5.20}$$

With this structure of the Hessian matrix, the following approximation seems reasonable:

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{S} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{5.21}$$

in which $\boldsymbol{S} \in \mathbb{R}^{n_p \times n_p}$ is a positive definite approximation to $\boldsymbol{S}(\boldsymbol{\theta})$. Replacing the actual Hessian in (5.17) with this approximation gives:

$$\begin{aligned} &\min_{\boldsymbol{h},\boldsymbol{d}} \frac{1}{2}\boldsymbol{h}^T\boldsymbol{S}\boldsymbol{h} + \frac{1}{2}\boldsymbol{d}^T\boldsymbol{d} + \boldsymbol{z}^T\boldsymbol{d} \\ &s.t. \quad \nabla \boldsymbol{r}(\boldsymbol{\theta})^T\boldsymbol{h} - \boldsymbol{d} + \boldsymbol{r}(\boldsymbol{\theta}) - \boldsymbol{z} = \boldsymbol{0} \end{aligned} \tag{5.22}$$

It is readily shown that the minimization problem (5.22) is equivalent to the linear system:

$$\left(\nabla \boldsymbol{r}(\boldsymbol{\theta})\nabla \boldsymbol{r}(\boldsymbol{\theta})^T + \boldsymbol{S}\right)\boldsymbol{h} + \nabla \boldsymbol{r}(\boldsymbol{\theta})\boldsymbol{r}(\boldsymbol{\theta}) = \boldsymbol{0} \tag{5.23}$$

If $\boldsymbol{S}$ is equal to $\boldsymbol{S}(\boldsymbol{\theta})$ in (5.20), then solving (5.23) is identical to performing one Newton step in the solution of the standard least squares problem (5.14)[4]. That is, one SQP iteration with $\boldsymbol{S} = \boldsymbol{S}(\boldsymbol{\theta})$ is equivalent to one Newton step in the solution of (5.14). It is noted that $\boldsymbol{S}(\boldsymbol{\theta})$ in (5.20) coincides with the second order terms in the Hessian (5.6) at an optimal solution, since $\boldsymbol{r}(\boldsymbol{\theta}) = \boldsymbol{z} = -\boldsymbol{\lambda}$. In practice, a quasi-Newton updating scheme is used to update the Hessian. Since $\boldsymbol{S}$ is an approximation to the second order terms in the Hessian, Schittkowski (1988) suggests initializing the approximation with:

$$\boldsymbol{B}_0 = \begin{bmatrix} \mu\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{5.24}$$

in which μ is chosen small, if small residuals are expected, or close to one if large residuals are expected. This allows the user to bring in information about the problem. Thus, by introducing a simple transformation and applying a general purpose SQP method, typical features of tailored least squares codes are retained, i.e. the combination of a Gauss-Newton search direction with a quasi-Newton correction (as in NL2SOL).

Example 5.2 (Comparing Two Formulations of the Least Squares Problem) In this example a comparison is made between the standard formulation of the least squares problem and the transformed NLP formulation. The same setup is used as in Example 5.1. The NLP problem may be expressed as:

$$\begin{aligned} &\min_{\boldsymbol{\theta},\boldsymbol{z}} \; f(\bar{\boldsymbol{\theta}}) = \frac{1}{2}\sum_{i=1}^{20} z_i^2 \\ &s.t. \quad y_j(t_i,\boldsymbol{\theta}) - \tilde{y}_{ij} - z_{10(j-1)+i} = 0, \quad i = 1,\ldots,10, \quad j = 1,2 \end{aligned} \tag{5.25}$$

[4]Note that $\boldsymbol{J}(\boldsymbol{\theta}) = \nabla \boldsymbol{r}(\boldsymbol{\theta})^T$.

With this transformed problem, the objective function and its gradient are evaluated at a very low cost. The gradient is simply given by:

$$\nabla f(\bar{\boldsymbol{\theta}}) = \begin{bmatrix} 0 & 0 & 0 & z_1 & \cdots & z_{20} \end{bmatrix}^T \tag{5.26}$$

All computational work lies in the evaluation of the nonlinear constraints and the constraint gradients. The Jacobian, $\boldsymbol{J}_c(\bar{\boldsymbol{\theta}})$, of the constraint functions is:

$$\boldsymbol{J}_c(\bar{\boldsymbol{\theta}}) = \begin{bmatrix} \frac{\partial y_1(t_1,\boldsymbol{\theta})}{\partial k_1} & \frac{\partial y_1(t_1,\boldsymbol{\theta})}{\partial k_2} & \frac{\partial y_1(t_1,\boldsymbol{\theta})}{\partial k_3} & -1 & 0 & \cdots & 0 \\ \frac{\partial y_1(t_2,\boldsymbol{\theta})}{\partial k_1} & \frac{\partial y_1(t_2,\boldsymbol{\theta})}{\partial k_2} & \frac{\partial y_1(t_2,\boldsymbol{\theta})}{\partial k_3} & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & & \ddots & \vdots \\ \frac{\partial y_2(t_{10},\boldsymbol{\theta})}{\partial k_1} & \frac{\partial y_2(t_{10},\boldsymbol{\theta})}{\partial k_2} & \frac{\partial y_2(t_{10},\boldsymbol{\theta})}{\partial k_3} & 0 & \cdots & 0 & -1 \end{bmatrix} \tag{5.27}$$

A feasible starting point for the optimization is obtained by setting $\boldsymbol{z}_0 = \boldsymbol{r}(\boldsymbol{\theta}_0)$. The nonlinear feasibility tolerance in NPSOL is set to 10^{-4}. The "warm start" option is used supplying an initial Hessian approximation of the form (5.21) (a few test runs indicate that $\mu = 10^{-4}$ is a reasonable choice). All other settings are unchanged from the previous example.

Formulation	Standard	Transformed
NIT	36	10
NOBJ	37	11
NGRD	37	11
NSTEP	2058	600
NFAIL	0	0
NFUN	23503	6834
NJAC	2058	600
NLU	2058	600
NBACK	23133	6714
$f(\boldsymbol{\theta}^*)$	$0.177569 \cdot 10^{-1}$	$0.177569 \cdot 10^{-1}$
k_1	11.7382	11.7381
k_2	5.9314	5.9313
k_3	10.3002	10.3006
CPU Time	0.591	0.272
NCLOCK	328.0	124.1

Table 5.4. Performance statistics for NPSOL applied to the parameter estimation problem in the gas-oil cracking model. Comparison is made between a standard least squares formulation and the NLP formulation proposed by Schittkowski (1988). ESDIRK34 was used for solving the equations and generating gradient information.

Table 5.4 shows the computational statistics for the comparison. Almost identical solutions are obtained, but the NLP formulation requires only 10 iterations compared to 36 using the standard formulation. Thus, substantial improvements are possible by introducing the simple transformation, which retains typical features of special purpose least squares codes. The performance using the NLP formulation is fully comparable to the performance of LMDER and NL2SOL (c.f. Table 5.2). ■

The conclusions from Example 5.1 and 5.2 are highlighted below:

- All CPU times listed are insignificant for practical applications, so, basically, any of the three algorithms would be satisfactory. For larger problems, however, it is inadvisable to use NPSOL with a standard formulation of the least squares problem.
- LMDER and NL2SOL show comparable performance, but the codes are not tailored for problems described by differential equations in that they separate the evaluation of the residual functions and the Jacobian of the residual functions. A modification is desirable. Of all three codes, LMDER is by far the least complex and therefore relatively easy to modify.
- The NLP formulation of the least squares problem using NPSOL seems promising. The performance is comparable to special purpose codes. Although the complexity of the problem is increased, the important advantage is that additional constraints can be handled. Often, for reasons of robustness, it is desirable to introduce simple bound constraints on the parameters.

In addition to the above, potential savings are expected if an adaptive scheme is used to adjust the tolerance in the equation solver. When far from the optimal solution a crude approximation to the solution of the differential equations would be sufficient. Then, when progress is made towards the solution, the accuracy in the approximation is increased.

5.2 Sensitivity Computation

Since the computationally dominating tasks in parameter estimation problems are the repeated solution of the differential equations and, especially, the generation of gradient information, efficient algorithms for these tasks are crucial to the overall performance. This section compares the performance of the codes DASPK and ESDIRK34, both of which were described in Section 3.1. The characteristics of the two codes are summarized below:

ESDIRK34 : This code implements a 3rd order semi-implicit Runge-Kutta method with a 4th order embedded method for error estimation. Systems of DAEs of index up to 1 are treated. Sensitivities are calculated using the staggered direct method with partial error control (only state variables).

DASPK : This code implements a variable order BDF method. Systems of DAEs of index up to 2 are treated. Sensitivities are calculated using any of the methods previously described, i.e. staggered direct, staggered corrector or simultaneous corrector. All sensitivity algorithms have an option for including sensitivity variables in the error test (full error control).

Compared to ESDIRK34, DASPK is a much larger package that treats a more general class of problems. In addition, a number of advanced features are

included such as an iterative solver for the linear system and an option for use of automatic differentiation. These features are, however, not considered in this test.

Example 5.3 (Performance Comparison of Sensitivity Algorithms)
ESDIRK34 and DASPK are applied to the parameter estimation problem in the gas-oil cracking model using LMDER for the optimization. Again, the same setup is used as in Example 5.1. Since DASPK treats problems of the form:

$$\begin{aligned} \boldsymbol{F}(t, \boldsymbol{y}, \dot{\boldsymbol{y}}) &= \boldsymbol{0} \\ \boldsymbol{y}(t_0) &= \boldsymbol{y}_0 \\ \dot{\boldsymbol{y}}(t_0) &= \dot{\boldsymbol{y}}_0 \end{aligned} \tag{5.28}$$

the user supplied subroutine must return the residual of each equation (including the sensitivity equations). An option exists to perform an initial consistency calculation internally, but this option is not used in order to avoid confusing the performance statistics. Instead, a separate subroutine is used to make the equations consistent.

All tolerance parameters in LMDER are set to 10^{-4}, and the relative and absolute tolerances in ESDIRK34 and DASPK are set to 10^{-6}. Collected statistics for the comparison are shown in Table 5.5. The simultaneous corrector method in DASPK fails to produce any results. LMDER terminates after two iterations claiming that no progress can be made. Closer inspection reveals that the initial sensitivity trajectories are completely wrong. The integration process seems unstable. DASPK is a very extensive code, so no further investigations are carried out.

Looking at the statistics in Table 5.5, various observations are made. In general, ESDIRK34 needs more function evaluations, but fewer Jacobian evaluations than DASPK. ESDIRK34 reuses the information obtained during state integration when evaluating right-hand-sides of the sensitivity equations, which is not the case for DASPK. This difference is reflected in the numbers for NJAC and NJACT.

When estimating parameters in large scale systems, the *LU* factorizations are expected to dominate, since these are the only $\mathcal{O}\left(n^3\right)$ operations performed, n being the dimension of the system. In fact, this is part of the motivation behind the staggered corrector method. As seen from the table, this method requires only a small number of *LU* factorizations. Since the Jacobian is not re-evaluated and factorized in each integration step, the staggered corrector method relies on a Newton iteration to converge the linear sensitivity equations. For the small scale example considered here, the differences between the staggered direct and staggered corrector methods in terms of computational efficiency are negligible. Interestingly, it is observed that, although the staggered corrector method applies an iterative scheme to the sensitivity equations, on average convergence is obtained with only one iteration (compare NSTEP and NSENS).

Regarding full or partial error control, the test results show only minor differences between the two strategies. One could argue that including the sensitivity variables in the error test would improve the robustness of the overall algorithm. As mentioned earlier, the sensitivity equations inherit the stiffness character of the state equations, so it seems fair to assume that the errors incurred in the integration of the sensitivity equations are of comparable size to the integration errors incurred in the state equations, thereby rendering the extra error test on sensitivity variables unnecessary. The statistics in Table 5.5 support this assumption showing that full error control has only a small impact on the total number of integration steps and error test failures. However, more test examples are needed in order to draw any certain conclusions. ■

Code	**DASPK**				**ESDIRK34**
Method	Staggered direct		Staggered corrector		Staggered direct
Error control	Full	Partial	Full	Partial	Partial
NIT	8	8	8	8	7
NOBJ	9	9	9	9	8
NGRD	8	8	8	8	7
NSTEP	1739	1727	1791	1759	791
NFAIL	22	20	30	30	0
NFUN	3845	3809	3932	3846	8998
NJAC	1761	1747	349	355	791
NJACT	3500	3474	2150	2124	791
NSENS	1739	1727	1801	1769	791
NLU	1761	1747	349	355	791
NBACK	7323	7263	7534	7384	7570
$f(\boldsymbol{\theta}^*)$	$0.177565 \cdot 10^{-1}$	$0.177565 \cdot 10^{-1}$	$0.177565 \cdot 10^{-1}$	$0.177565 \cdot 10^{-1}$	$0.177566 \cdot 10^{-1}$
k_1	11.6737	11.6737	11.6737	11.6737	11.6866
k_2	5.8930	5.8930	5.8930	5.8930	5.8897
k_3	10.3476	10.3476	10.3476	10.3477	10.3454
CPU Time	0.261	0.247	0.260	0.240	0.215
NCLOCK	123.6	113.7	122.8	111.1	101.3

Table 5.5. Performance statistics for LMDER applied to the parameter estimation problem in the gas-oil cracking model. Comparison is made between different methods for solving the differential equations and generating gradient information.

In summary: The differences in performance among the different sensitivity methods are negligible. DASPK has three sensitivity methods to choose from, one of which failed on this simple example. In general, the DASPK package has many advanced features, which make it suitable for a wide range of problems. ESDIRK34, on the other hand, implements only the staggered direct method, but the implementation is simple, easy to modify and works efficiently. For many typical parameter estimation problems ESDIRK34 would be an adequate choice.

6

PARFIT (I) : Design Considerations and Initial Development

Based on the comparative study of the previous chapter, the aim of this chapter is to address some of the potential difficulties arising when estimating parameters in dynamical systems and to make progress towards an efficient tool applicable to general parameter estimation problems in systems described by DAEs. To facilitate the process of switching between different models and testing different parameter estimation scenarios, effort is put into developing a routine (PARFIT) for parameter estimation that basically acts as an interface to the optimization algorithm and the differential equation solver, but allows for easy switching between models. In addition, the routine implements an option for adaptive tolerance selection as discussed in the following. The work initiated in this chapter is continued in the final report.

6.1 A Tool for DAE Parameter Estimation

The benchmark tests in Examples 5.1, 5.2 and 5.3 showed comparable performance of LMDER, NL2SOL and NPSOL using the NLP formulation of the problem. Furthermore, no considerable differences were observed in overall performance between the different sensitivity algorithms. Basing any conclusions on a single, fairly well behaved example is questionable. It seems fair to argue in favour of NPSOL in terms of robustness, since the algorithm handles constraints on the parameters. A combination of NPSOL for the optimization and DASPK for the state and sensitivity integration would provide an extensive framework capable of handling a wide range of problems. The main drawback of this approach, however, is that NPSOL and DASPK are both extremely complicated algorithms consisting of thousands of lines of code. To provide more flexibility in terms of being able to modify the algorithms, a decision is made here to base the PARFIT routine on LMDER and ESDIRK34, both of which are compact codes, which are easy to read.

To explain the considerations made when developing PARFIT, some of the potential difficulties frequently encountered in parameter estimation problems are briefly discussed. Schittkowski (2002) lists the following reasons why least squares algorithms sometimes slow down convergence or even terminate unsuccessfully:

- Overdetermined parameter sets.
- Internal round-off errors.
- Badly scaled parameters.
- Approximation errors in fitting criteria.
- Measurement errors in experimental data.
- Bad starting guesses.

In realistic problems many of these complications are present at the same time making the parameter estimation problem a nontrivial task. Errors in experimental data are always present, and parameters are often badly scaled (e.g. kinetic rate constants often differ by several orders of magnitude). Also common are overdetermined parameter sets, meaning that insufficient information is available in the data to identify the parameters uniquely resulting in high correlation between the parameters. Especially for systems described by differential equations, for which numerical solutions are required, the fitting criteria are subject to approximation errors. The influence of these errors is investigated in the next subsection. Finally, the user may provide starting guesses that are too far from the optimal solution causing the optimizer to progress extremely slowly or terminate unsuccessfully. As mentioned in the literature review, bad starting guesses may also cause stiffness of the model equations even though the equations are non-stiff with the optimal parameters.

It is probably debatable who is responsible for which of the "trouble makers" listed above, or more importantly, who should do something about it: The model maker building the mathematical models, the experimentalist carrying out the experiments, or the numerical analyst designing the parameter estimation software. From a numerical analyst's point of view it seems reasonable to concentrate on round-off errors, scaling of parameters and approximation errors in fitting criteria. Measurement errors are inevitable, so the software must be able to handle noisy data. Poor starting guesses are impossible to guard against, but measures exist to reduce the influence of such guesses (c.f. multiple shooting).

6.1.1 Approximation Errors in Fitting Criteria

Dynamical systems require the repeated numerical solution of a system of differential equations. However, the accuracy in the approximate solution is under the disposal of the user. Inaccurate state variables and sensitivities will affect the optimization process and may lead to an unsuccessful termination. Since the main bulk of computational labour in DAE parameter estimation is expended solving the differential equations and calculating derivative information, controlling the integration accuracy is one of the key challenges. Too accurate integration will slow down computation, whereas inaccurate integration may prevent a successful optimization. The accuracy requirement will probably also vary during the optimization. When far from the optimal solution, a crude approximation to the state and sensitivity variables may be sufficient. Then, when

the optimal solution is approached, higher accuracy is required. This suggests using an adaptive strategy to adjust the tolerance in the equation solver.

Before addressing the tolerance selection strategy implemented in PARFIT, an example is given demonstrating the effect of integration accuracy on the optimal solution.

Example 6.1 (Influence of Integration Accuracy on Optimization)
Consider the parameter estimation problem in the gas-oil cracking model. Unlike the previous examples, exact experimental data are used generated with $k_1 = 12$, $k_2 = 8$ and $k_3 = 8$. LMDER is started with $\boldsymbol{\theta_0} = [50\ \ 1\ \ 0.01]^T$ and all tolerance parameters set to 10^{-4}. The differential equations are solved using ESDIRK34 with varying tolerance (absolute and relative tolerances are set equal).

Tolerance	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-6}
NOBJ	25	19	16	13	12
NGRD	25	19	16	13	12
$f(\boldsymbol{\theta^*})$	$0.52 \cdot 10^{-5}$	$0.48 \cdot 10^{-6}$	$0.75 \cdot 10^{-7}$	$0.28 \cdot 10^{-8}$	$0.23 \cdot 10^{-10}$
k_1	11.5458	11.8442	11.9213	11.9798	11.9990
k_2	7.9010	7.9672	7.9859	7.9972	7.9998
k_3	8.0991	8.0377	8.0183	8.0029	8.0001

Table 6.1. Performance results for LMDER and ESDIRK34 using fixed optimization tolerance, but varying absolute and relative tolerance in the equation solver.

Table 6.1 contains parameter values, final objective function value, the number of objective function evaluations, and the number of gradient evaluations[1]. All optimizations terminate successfully with the termination criterion (5.3) satisfied. Even with a very loose tolerance in ESDIRK34, the true solution is approached. The numbers for NOBJ and NGRD decrease with increasing accuracy. Even though the true solution is approached in each situation, the results indicate that small changes to the accuracy of the state and sensitivity variables cause the optimization algorithm to follow a different path of execution. Thus, it seems fair to imagine that in a less well behaved example a too loose tolerance may prevent a successful optimization.

Although well behaved, a scenario is easily constructed that, regardless of the tolerance specified, causes an unsuccessful optimization. For example, starting with $\boldsymbol{\theta_0} = [1\ \ 50\ \ 1]^T$ produces an error from ESDIRK34 in the first iteration saying that the stepsize has been reduced below the minimum limit. Inspection reveals that LMDER in the inner loop of the first iteration calls the objective function subroutine with the parameter vector $\boldsymbol{\theta} = [-27.27\ \ 50\ \ 1]^T$. With these parameters the system has a positive eigenvalue, which forces the stepsize down in ESDIRK34. Introducing a logarithmic transformation of all parameters prevents negative parameter values.

As will be explained in much more detail in Appendix A ESDIRK34 calculates sensitivities, which are approximations compared to the integration scheme used for the state integration. Table 6.1 shows the effect of varying integration tolerance on the optimal parameter estimates, but the influence of approximation errors in the sensitivity variables alone cannot be distinguished. Therefore, a comparison is made in which the sensitivity equations are integrated exactly[2]. The result is shown in Table 6.2. In all

[1] LMDER has been modified to perform a gradient evaluation whenever the objective function is evaluated. This modification is explained later.

[2] That is, the integration is exact based on the integration scheme applied to the state equations.

Tolerance	10^{-4}		10^{-6}	
Sensitivities	Exact	Approximate	Exact	Approximate
NOBJ	11	13	10	12
NGRD	11	13	10	12
$f(\boldsymbol{\theta^*})$	$0.59 \cdot 10^{-8}$	$0.28 \cdot 10^{-8}$	$0.51 \cdot 10^{-10}$	$0.23 \cdot 10^{-10}$
k_1	11.9724	11.9798	11.9985	11.9990
k_2	7.9958	7.9972	7.9997	7.9998
k_3	8.0049	8.0029	8.0001	8.0001

Table 6.2. Performance results for LMDER and ESDIRK34 using fixed optimization tolerance (10^{-4}), but varying absolute and relative tolerance in the equation solver. Comparison is made between the approximate sensitivities calculated by ESDIRK34 and the exact sensitivities obtained by integrating the sensitivity equations directly.

cases the optimization terminates with the criterion (5.3) satisfied. Using exact sensitivities the number of iterations is slightly reduced. Considering the higher objective function value obtained with the exact sensitivities, this difference seems negligible. ■

It seems an appealing idea to use a tolerance selection mechanism that tightens the tolerance when the optimal solution is approached. Of course, there is a risk that the "extra slack" allowed in the initial phase of optimization will lead to an unsuccessful termination. Surprisingly little information is found in the literature on this kind of tolerance adaptation in parameter estimation. Bock (1983) has a few comments on the subject, noting that, although the idea seems straightforward, a computationally effective and safe implementation that works for highly nonlinear as well as for ill-conditioned problems is not trivial. The simple strategy used in PARFIT consists of making multiple calls to the optimization algorithm with successively tightening tolerance. If the final optimization tolerance required by the user is, say, $\varepsilon = 10^{-6}$, then the first call is made with $\varepsilon = 10^{-2}$. After termination with this tolerance, a second call is made with $\varepsilon = 10^{-4}$ and, finally, a third call with $\varepsilon = 10^{-6}$. The absolute and relative tolerances in the equation solver are adapted accordingly. In general, the optimization interval is divided into n_{tol} subintervals. The optimization tolerance for the ith subinterval is:

$$\varepsilon_i = \varepsilon_{final}^{i/n_{tol}} \tag{6.1}$$

and the corresponding tolerances for the equation solver are:

$$atol_i = rtol_i = scale \cdot \varepsilon_i \tag{6.2}$$

Numerical experiments suggest that $scale = 0.01$ is a reasonable value. Details on implementation of PARFIT are provided in Appendix A. The tolerances associated with the termination criteria (5.2)–(5.4) are given the default value 10^{-4}. In case the user supplies individual tolerances for the three criteria, the corresponding equation solver tolerances are scaled according to the minimum of the optimization tolerances.

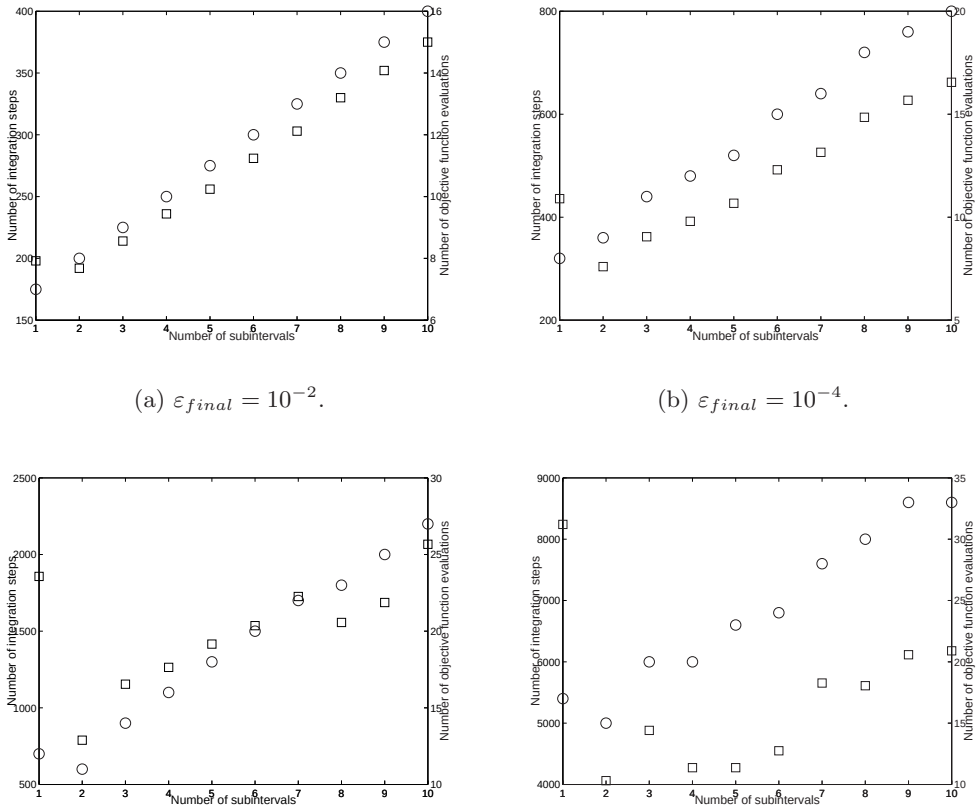
Example 6.2 (Adaptive Tolerance Selection)
The performance of PARFIT using the tolerance selection mechanism is compared to the case with fixed tolerances. Again, the gas-oil cracking problem with noisy data is used as test example (same setup as in Example 5.1). The final optimization tolerance is 10^{-4}, 2 subintervals are used, and for the fixed tolerance case $atol = rtol = 10^{-6}$. As indicated earlier, a modification has been made to LMDER. Originally, the evaluations of the residual functions and the Jacobian of the residual functions were separated, but this is inefficient for dynamical systems. Therefore, LMDER has been modified to perform a Jacobian evaluation whenever the residuals are evaluated. Table 6.3 shows performance statistics for PARFIT using (i) the original LMDER, (ii) the modified LMDER, and (iii) the modified LMDER with adaptive tolerance selection.

Various observations are made. The small modification of LMDER doubles the efficiency. Instead of 15 repeated solutions of the differential equations, only 8 are required. With 2 subintervals the adaptive tolerance strategy reduces the total number of integration steps by 25%. 7 calls are made to the objective function subroutine with the loose tolerance and only 2 with the tighter tolerance. It seems fair to assume that the optimal number of subintervals is problem–dependent. Experiments with this example indicate that 2 is a reasonable choice. Figure 6.1 shows the number of objective function evaluations and the corresponding total number of integration steps as a function of the number of subintervals. Plots are shown for increasing values of the final optimization tolerance. All plots show that the optimal number of subintervals is 2. Too many subintervals increase the total number of iterations, and hence the number of integration steps. The relative reduction in the number of integration steps experienced when going from 1 to 2 subintervals is much greater for the tighter tolerances (compare Figure 6.1a with c or d). Part of the explanation for this observation is that ESDIRK34, when working at very loose tolerances, has a lower bound on the computational work, which means that changing the integration tolerance from 10^{-2} to 10^{-3} is not comparable to changing it from 10^{-6} to 10^{-7}.

Finally, some rather irregular behaviour is observed in Figure 6.1d. As mentioned earlier, small changes in tolerances cause the optimization algorithm to follow a different path of execution, which might be the reason for the observed behaviour. Also, the

LMDER version	Original	Modified	Adaptive tolerance
NOBJ	8	8	9
NGRD	7	8	9
NSTEP	791	436	320
NFAIL	0	4	2
NFUN	8998	4951	3527
NJAC	791	432	318
NLU	791	436	320
NBACK	7570	4859	3431
$f(\boldsymbol{\theta}^*)$	$0.177566 \cdot 10^{-1}$	$0.177566 \cdot 10^{-1}$	$0.177569 \cdot 10^{-1}$
CPU Time	0.215	0.120	0.112
NCLOCK	101.3	53.6	51.1

Table 6.3. Performance statistics for PARFIT applied to the gas-oil cracking problem. Comparison is made between the original LMDER (similar to results from Example 5.1), the modified LMDER, and the modified LMDER with an adaptive tolerance scheme.



(a) $\varepsilon_{final} = 10^{-2}$.

(b) $\varepsilon_{final} = 10^{-4}$.

(c) $\varepsilon_{final} = 10^{-6}$.

(d) $\varepsilon_{final} = 10^{-8}$.

Figure 6.1. Adaptive tolerance selection. The number of objective function evaluations and the corresponding total number of integration steps are plotted as a function of the number of subintervals. Plots are shown for four different values of the final optimization tolerance. (□ : left axis, ◯ : right axis).

figure does not reflect the number of objective function evaluations performed in each subinterval. The trend seems to be that the majority of the evaluations are performed with loose tolerances and only a few with the tight tolerances. This is in good agreement with the convergence properties of the Levenberg-Marquardt method. When close to the solution, superlinear convergence is expected (for small residual problems). ■

6.1.2 Scaling of Data and Parameters

PARFIT estimates parameters using a weighted least squares criterion. The minimization problem is:

$$\min_{\boldsymbol{\theta}} \; f(\boldsymbol{\theta}) = \frac{1}{2} \sum_{j=1}^{l_c} \sum_{i=1}^{l_t} w_{ij}^2 \left(y_j(t_i, \boldsymbol{\theta}) - \tilde{y}_{ij} \right)^2 \tag{6.3}$$

in which l_c and l_t denote the number of measured components and the number of experimental time values, respectively. Thus, the total number of measurements is $m = l_c l_t$. l_c components measured at l_t experimental times are referred

to as a data set. The extension to multiple data sets, which are characterized by an additional set of independent model variables (e.g. temperature or concentration) is treated in the next report. Each measurement $\tilde{y}_{ij}$ has an associated weight w_{ij} that must be specified by the user. An obvious reason for introducing weight factors is the possibility to set $w_{ij} = 0$ if measurement $\tilde{y}_{ij}$ is not available. Another reason is that the measured components may have different physical meanings leading to different dimensions. In this case, Schittkowski (2002) suggests scaling the residuals by the sum of squares of the corresponding measurement values:

$$w_{ij} = \left(\sum_{k=1}^{l_t} \tilde{y}_{kj}^2 \right)^{-1/2} \tag{6.4}$$

for $j = 1, \ldots, l_c$. If, however, the order of magnitude of the individual measured components differ within the time interval considered, a more individual scaling procedure is needed. If large function and measurement values lead to large residuals, then these values will be fitted better than those with small residuals, if no scaling is applied. To ensure that all measured values are equally taken into account, the following scaling can be applied:

$$w_{ij} = \frac{1}{|\tilde{y}_{ij}|} \tag{6.5}$$

for $i = 1, \ldots, l_t$ and $j = 1, \ldots, l_c$. Basically, the choice of scaling depends on the knowledge about the measurement errors. In PARFIT, individual weight factors allow the user to choose the type of scaling.

The parameters of a problem also frequently have different dimensions. By default, the parameters are scaled internally in PARFIT, an option that is part of the LMDER optimization algorithm. LMDER scales the parameters according to the norms of the columns of the initial Jacobian of the residual functions. That is, the ith parameter is scaled by $\|\boldsymbol{J}(\boldsymbol{\theta_0})_{:,i}\|$, in which $\boldsymbol{\theta_0}$ denotes the initial guess for the parameter vector. An option exists for the user to provide scale factors.

6.1.3 The PARFIT Algorithm

The discussion above covered some of the considerations made when developing PARFIT. Details of implementation as well as code listings including a driver for the gas-oil cracking problem are given in Appendix A. Current features and limitations of PARFIT are highlighted below:

- PARFIT estimates problem parameters in index one DAEs. An option exists for estimating initial conditions in ODE systems.
- An adaptive tolerance selection mechanism is used to improve performance.
- Data are scaled using individual weight factors supplied by the user. Possible scaling strategies are introduced above.
- Parameters are scaled internally. An option exists for user supplied scale factors.

- The parameter covariance and correlation matrices are computed.

Current limitations, some of which are addressed in the next report, include:

- Multiple data sets are not handled. The extension is fairly straightforward requiring multiple solutions of the differential equations in each call to the objective function subroutine. Additional independent model variables characterizing each data set are introduced.

- The user must provide the Jacobian of the differential equations as well as the partial derivatives with respect to the parameters being estimated. Numerical difference approximations or the use of automatic differentiation techniques are possible extensions.

- If the states of the system are not measured directly, the user must modify the objective function subroutine to account for the measurement equation.

- The initial conditions for the differential equations are assumed to be independent from the parameters appearing in the right-hand-side functions of the DAEs.

- In case of algebraic equations, an auxiliary subroutine must be used to ensure consistency in the equations. Automatic consistency calculation is not implemented in the equation solver.

6.2 The Dow Chemicals Problem

The PARFIT algorithm is now tested on a notoriously difficult problem with real data, which was formulated in 1981 by the Dow Chemical Company. The problem was distributed to 165 researchers at a conference, but only 5 research groups submitted acceptable solutions. The results from these groups are presented and compared in Biegler *et al.* (1986).

Example 6.3 (The Dow Chemicals Problem)
The parameter estimation problem is based on a kinetic model of an isothermal batch reactor. The model is described by a system of 10 stiff DAEs with 9 unknown parameters to be estimated. A formulation of the problem and a description of the chemical background and the assumptions made can be found in Biegler *et al.* (1986). Here, only the resulting model equations are presented. The model consists of 6 differential

equations and 4 algebraic equations:

$$\frac{dy_1}{dt} = -k_2 y_2 y_8 \tag{6.6a}$$
$$\frac{dy_2}{dt} = -k_1 y_2 y_6 + k_{-1} y_{10} - k_2 y_2 y_8 \tag{6.6b}$$
$$\frac{dy_3}{dt} = k_2 y_2 y_8 + k_1 y_4 y_6 - \frac{1}{2} k_{-1} y_9 \tag{6.6c}$$
$$\frac{dy_4}{dt} = -k_1 y_4 y_6 + \frac{1}{2} k_{-1} y_9 \tag{6.6d}$$
$$\frac{dy_5}{dt} = k_1 y_2 y_6 - k_{-1} y_{10} \tag{6.6e}$$
$$\frac{dy_6}{dt} = -k_1 y_6 \left(y_2 + y_4\right) + k_{-1} \left(y_{10} + 1/2 y_9\right) \tag{6.6f}$$
$$y_7 = -\left[\mathrm{Q}^+\right] + y_6 + y_8 + y_9 + y_{10} \tag{6.6g}$$
$$y_8 = \frac{K_2 y_1}{K_2 + y_7} \tag{6.6h}$$
$$y_9 = \frac{K_3 y_3}{K_3 + y_7} \tag{6.6i}$$
$$y_{10} = \frac{K_1 y_5}{K_1 + y_7} \tag{6.6j}$$

The equilibrium constants K_i, $i = 1, 2, 3$, are assumed to be temperature independent, whereas the rate constants k_1, k_{-1} and k_2 are assumed temperature dependent via Arrhenius' law:

$$k_i = \alpha_i \exp\left(-E_i / RT\right), \qquad i = 1, -1, 2 \tag{6.7}$$

in which α_i and E_i denote the preexponential factor and the activation energy, respectively. Thus, the 9 unknown parameters in the original formulation of the problem are:

$$\boldsymbol{\theta} = \begin{bmatrix} \alpha_1 & E_1 & \alpha_{-1} & E_{-1} & \alpha_2 & E_2 & K_1 & K_2 & K_3 \end{bmatrix}^T \tag{6.8}$$

The quantity $[\mathrm{Q}^+]$ in (6.6g) is a concentration, which is assumed to be constant during the reactions.

The available data originate from three different experiments, in which the temperature is varied from experiment to experiment (40°C, 67°C and 100°C). During each experiment, data from four different components corresponding to the first four elements of the state vector in (6.6) are observed. Three species are measured and the measured values are adjusted according to a conservation law. The value of the fourth component is derived from an additional relation. This history of the data suggests that the measurements are correlated. The initial conditions corresponding to the low temperature data set are:

$$\begin{aligned}
y_1(0) &= 1.7066 \\
y_2(0) &= 8.32 \\
y_3(0) &= 0.01 \\
y_4(0) &= 0.0 \\
y_5(0) &= 0.0 \\
y_6(0) &= \left[\mathrm{Q}^+\right] = 0.0131 \\
y_7(0) &= 1/2 \cdot \left(-K_2 + \sqrt{K_2^2 + 4 K_2 y_1(0)}\right) \\
y_8(0) &= y_7(0) \\
y_9(0) &= 0.0 \\
y_{10}(0) &= 0.0
\end{aligned} \tag{6.9}$$

The initial parameter estimates are given in Table 6.4. The activation energies have

Parameter	Value
α_1	$2.0 \cdot 10^{13}$
E_1	$2.0 \cdot 10^{4}$
α_{-1}	$4.3 \cdot 10^{15}$
E_{-1}	$2.0 \cdot 10^{4}$
α_2	$2.0 \cdot 10^{13}$
E_2	$2.0 \cdot 10^{4}$
K_1	$1.0 \cdot 10^{-17}$
K_2	$1.0 \cdot 10^{-11}$
K_3	$1.0 \cdot 10^{-17}$

Table 6.4. Initial parameter estimates for the batch reactor problem (Biegler *et al.*, 1986).

dimension [cal/mol], the equilibrium constants have dimension [mol/kg], and the pre-exponential factors have dimensions corresponding to the individual rate constants.

Results

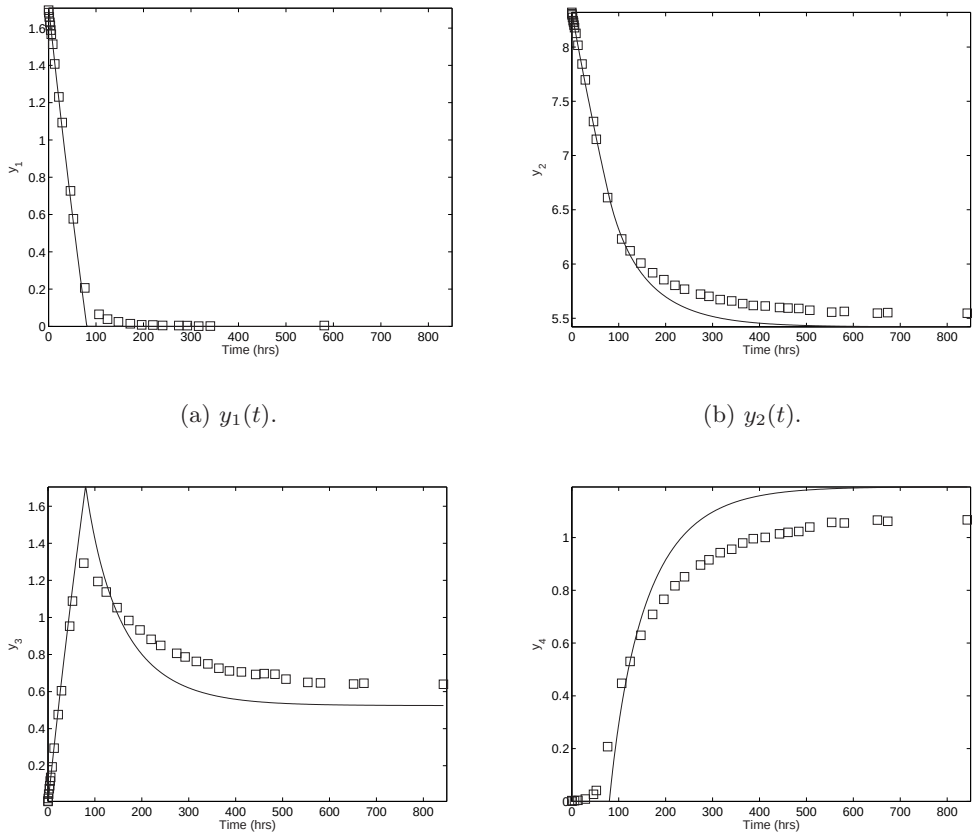
In this example a simplified version of the parameter estimation problem is treated. Instead of estimating Arrhenius parameters, the rate constants k_1, k_{-1} and k_2 are estimated directly from isothermal data. This simplification reduces the number of parameters to 6. The low temperature data are used for estimation.

Parameter	Value	Scaled parameter	Value
k_1	0.219	$\ln k_1$	-1.52
k_{-1}	47.15	$\ln k_{-1}$	3.85
k_2	0.219	$\ln k_2$	-1.52
K_1	$1.0 \cdot 10^{-17}$	$\ln K_1$	-39.1
K_2	$1.0 \cdot 10^{-11}$	$\ln K_2$	-25.3
K_3	$1.0 \cdot 10^{-17}$	$\ln K_3$	-39.1

Table 6.5. Initial parameter estimates for the simplified batch reactor problem. The scaled parameters used in the estimation are listed in the last column.

The initial parameter estimates for the simplified problem are given in Table 6.5. These parameters vary by many orders of magnitude. To obtain a better scaling of the numerical problem, it is preferable to have the parameters within approximately the same order of magnitude. To achieve this, a logarithmic transformation is introduced. The transformed parameters are listed in the second part of Table 6.5.

Before attempting a solution, the problem is studied by simulation with the initial parameter values. Figure 6.2 shows the low temperature data plotted along with the solution trajectories computed using the initial parameter estimates from Table 6.5. Sharp changes are observed in the concentration profiles. y_1 decreases rapidly to zero before the species represented by y_4 starts to form. These rapid changes also appear in the sensitivity trajectories, which are shown in Figure 6.3. Looking at the sensitivities,



(a) $y_1(t)$. (b) $y_2(t)$.

(c) $y_3(t)$. (d) $y_4(t)$.

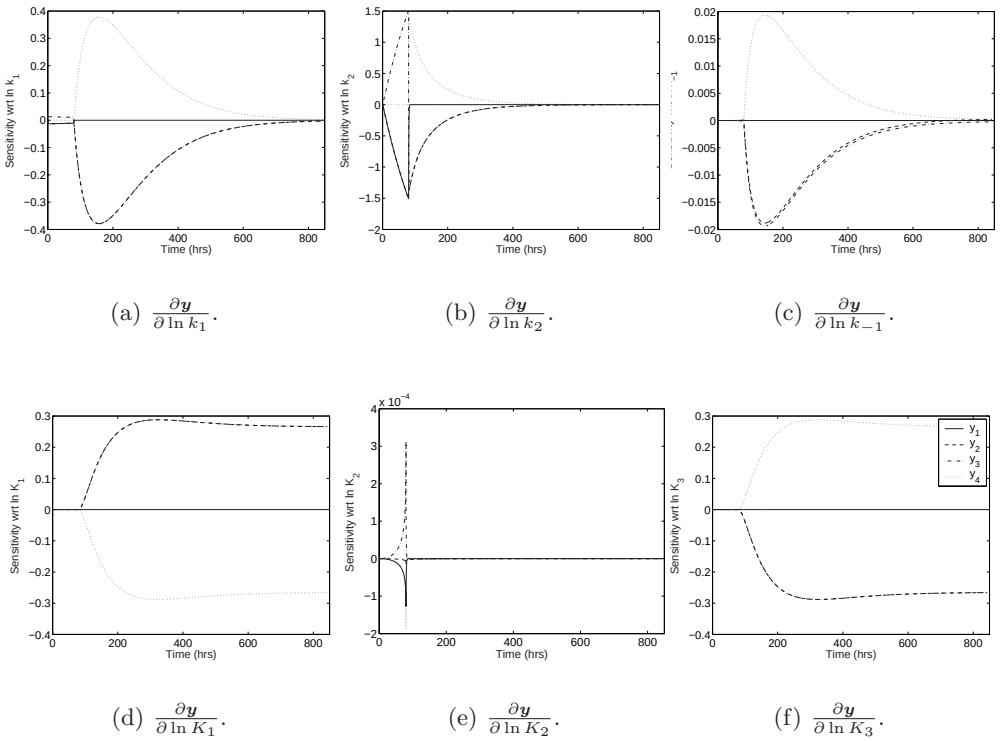
Figure 6.2. The low temperature data (□) are plotted along with the solution trajectories computed using the initial parameter estimates from Table 6.5.

various observations are made. As expected, no changes in the parameters affect y_1 after $t \simeq 80$, since y_1 has decreased to zero. Before $t \simeq 80$, changes in k_{-1}, K_1 and K_3 do not affect any of the states. Close inspection shows that changes in K_1 and K_3 affect y_2 and y_3 in the same way (the curves for y_2 and y_3 coincide in Figure 6.3d and f). In general, sensitivity plots can be of considerable use. The physical interpretation of the trajectories is, however, not considered here. The following linear relation:

$$\frac{\partial \boldsymbol{y}}{\partial \ln K_1} = -\frac{\partial \boldsymbol{y}}{\partial \ln K_3} \tag{6.10}$$

is observed from Figure 6.3d and f. Thus, K_1 and K_3 cannot be estimated independently from data on $\boldsymbol{y}(t)$. Therefore, it is chosen to fix K_3 at its initial value.

Having made the preliminary adjustments, the first optimization run is performed using the default values in PARFIT. PARFIT terminates unsuccessfully with an error from ESDIRK34 saying that the step length has been decreased below the lower limit. Previous experience suggests that this behaviour is forced by a poor parameter set computed by LMDER during optimization, which renders the integration algorithm unstable. Further inspection reveals that the initial Jacobian of the residuals is rank deficient leading to a singular Hessian approximation. Thus, an additional linear relation between the parameter sensitivities seems to exist, which could not be identified



(a) $\frac{\partial \boldsymbol{y}}{\partial \ln k_1}$. (b) $\frac{\partial \boldsymbol{y}}{\partial \ln k_2}$. (c) $\frac{\partial \boldsymbol{y}}{\partial \ln k_{-1}}$.

(d) $\frac{\partial \boldsymbol{y}}{\partial \ln K_1}$. (e) $\frac{\partial \boldsymbol{y}}{\partial \ln K_2}$. (f) $\frac{\partial \boldsymbol{y}}{\partial \ln K_3}$.

Figure 6.3. Sensitivity trajectories with initial parameter values. The plots show the sensitivities of the first four states in the batch reactor model with respect to (wrt) the logarithm of the parameters.

from the sensitivity plots. This supposition is supported by the commentary provided by one of the research groups, whose results are presented in Biegler *et al.* (1986). The researchers note a "two-dimensional dependency" among the parameters k_{-1}, K_1, K_2 and K_3. A decision is therefore made also to fix K_2 at its initial value during the optimization.

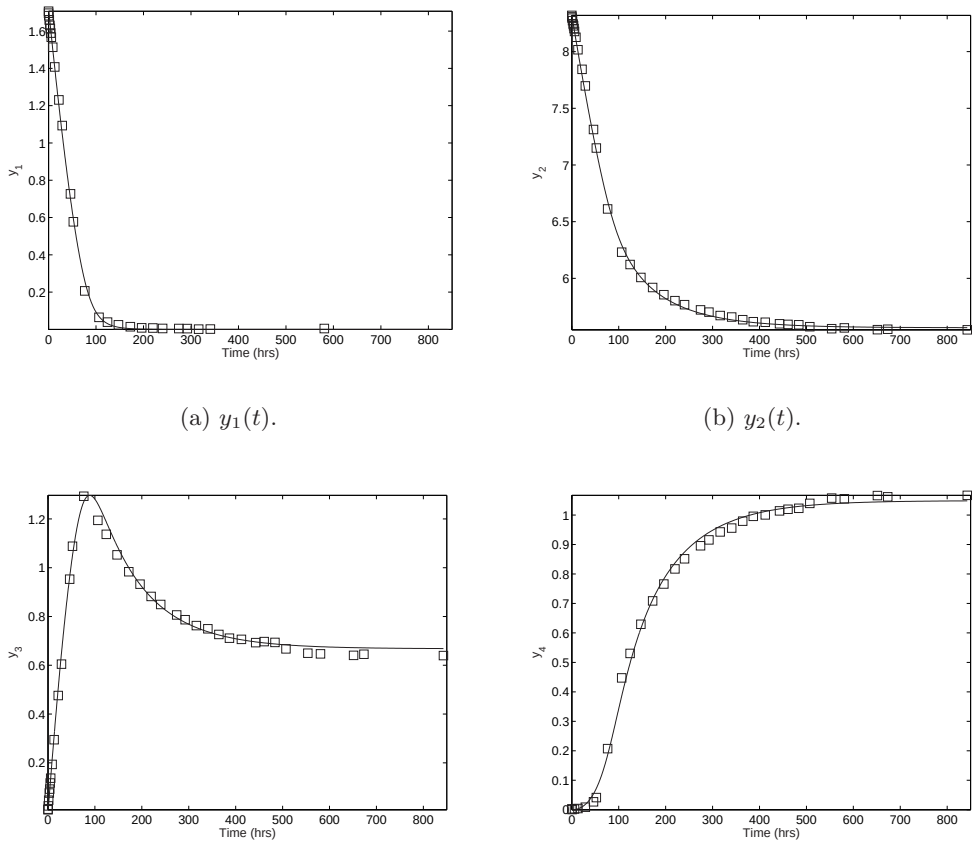
Using a very conservative value of the parameter controlling the initial step bound in LMDER, tightening the optimization tolerances to 10^{-7}, and modifying the initial parameter guesses slightly[3], the optimization finally terminates with an acceptable solution using 15 iterations. The solution trajectories corresponding to the optimal parameters are plotted in Figure 6.4. The parameters are listed in Table 6.6 along

Parameter	Final estimate	Marginal confidence interval
$\ln k_1$	-1.56	$\pm 4.93 \cdot 10^{-2}$
$\ln k_2$	-1.46	$\pm 1.85 \cdot 10^{-2}$
$\ln k_{-1}$	12.8	$\pm 1.11 \cdot 10^{-1}$
$\ln K_1$	-38.6	$\pm 3.09 \cdot 10^{-2}$

Table 6.6. Final parameter estimates plus 95% marginal confidence intervals.

with 95% confidence intervals, which all seem satisfactory. The correlation matrix is

[3]The initial guess for K_1 was changed to 10^{-15}.



(a) $y_1(t)$. (b) $y_2(t)$.

(c) $y_3(t)$. (d) $y_4(t)$.

Figure 6.4. Solution trajectories for the batch reactor problem computed using the optimal parameters from Table 6.6.

given in Table 6.7. The highest correlation is observed between k_1 and K_1. The final objective function value is 0.036.

This problem requires a great deal of "tuning" in order to produce acceptable results. First, a transformation of the parameters was introduced to take care of bad scaling. Secondly, linear dependencies among the parameters were identified, and the parameter set subsequently reduced. Finally, adjustments to the optimizer settings were made and one of the initial parameter guesses was changed. By experimenting a little further, the following observations are made:

- If the initial guess for K_1 is kept at 10^{-17}, the optimizer terminates successfully after 7 iteration, but at a different optimum with a final objective function value of 0.201. Inspection of the corresponding solution trajectories shows that only little progress has been made towards the solution.
- If the optimization tolerance is loosened to 10^{-6} (or higher), the first step computed by the optimizer causes an unstable system, and termination occurs because of the integrator taking too small steps. The same happens if the initial step bound in the optimizer is chosen too large. Allowing extra slack in the early iterations by using several tolerance intervals is clearly a bad idea in this problem.

	k_1	k_2	k_{-1}	K_1
k_1	1.000			
k_2	-0.557	1.000		
k_3	-0.136	0.516	1.000	
K_1	0.761	-0.308	-0.702	1.000

Table 6.7. Estimated correlation matrix for the parameters in the batch reactor problem.

- If the integrator tolerance is fixed at a very tight level (say 10^{-9}), the optimization tolerance can be loosened to 10^{-3} and the optimization still terminates successfully at the optimal solution. This is an interesting observation showing that the integration accuracy in this problem has great influence on the robustness of the overall algorithm.
- No scaling of the data has been used so far, since all components are measured on the same scale and no prior knowledge is available regarding the error structure of the data. Scaling according to (6.4) or (6.5) does not result in any improvements.

Two issues seem to be of concern: The optimizer "gets stuck" and terminates because of too small steps, or the optimizer produces poor parameter guesses in the initial phase of optimization, which causes the integrator to fail. The latter seems to be triggered by a loose integration tolerance.

Indeed, this batch reactor problem possesses many of the undesirable characteristics discussed in Section 6.1. It is badly scaled, the model equations are stiff, and the initial parameter set is overdetermined. PARFIT was able to solve the simplified problem in its standard formulation with 6 differential equations and 4 algebraic equations. However, a great deal of user intervention was still required to obtain acceptable results. ■

7

Summary

The purpose of this progress report was to identify and address some of the key challenges faced when solving parameter estimation problems in dynamical systems. Chapter 5 covered a series of benchmark tests. The performance of three different optimization algorithms was compared in terms of computational efficiency by measuring a number performance indicators. The LMDER and NL2SOL codes showed comparable performance when applied to a simple test problem with three unknown parameters. By transforming the nonlinear least squares problem into a general nonlinear programming problem, the NPSOL code was able to solve the test problem just as efficiently as LMDER and NL2SOL, which, unlike NPSOL, are specifically tailored for least squares problems. NPSOL has the additional advantage that constraints are handled.

Comparison was also made between two differential equation solvers in terms of (i) solving the differential equations and (ii) solving the corresponding set of sensitivity equations. In particular, the staggered direct and staggered corrector methods for sensitivity integration were compared. The motivation for using the staggered corrector method is a reduction in the number of LU factorizations required for the joint state and sensitivity integration. However, for problems of moderate scale this is insignificant. DASPK and ESDIRK34 were comparable in performance.

In Chapter 6 effort was put into developing the PARFIT routine based on LMDER for the optimization and ESDIRK34 for the state and sensitivity integration. These codes performed satisfactory in the tests, and they are both compact and easy to modify thereby providing a flexible parameter estimation routine. Several numerical pitfalls encountered in parameter estimation were discussed. Of particular interest when estimating parameters in dynamical systems are the approximation errors incurred in the objective function due to the numerical solution of the differential equations. Controlling the accuracy in the state and sensitivity variables is a key issue. Experiments on a batch reactor problem showed that inaccurate state and sensitivity variables easily prevent a successful optimization. Unless a tight integration tolerance was used, poor parameter guesses were computed by the optimizer in the initial iterates, which ultimately forced the integrator to fail. Achieving both robustness and efficiency in parameter estimation software is not a trivial task. Good results were obtained on a more "well behaved" example using a simple tolerance selection mechanism to improve the efficiency. However, the batch reactor example showed that using loose tolerances in the initial iterations may compromise the robustness.

Final Report

8

PARFIT (II) : Efficiency, Robustness and Flexibility

The primary goal of this final report is to extend the PARFIT algorithm, the work on which was initiated in the previous report. Based on the experience gained, a number of extensions are introduced in order to improve the flexibility, robustness and efficiency of the algorithm.

Sections 8.1–8.4 provide the details of the individual extensions. The previous report showed that accurate state and sensitivity variables were crucial to the success of the optimization. Other problems arose due to linear dependencies among the parameters, which resulted in the optimizer computing poor parameter guesses. To improve the robustness in such situations, regularization of the problem is introduced. An extension is also made to allow multiple data sets in the estimation. Each data set is characterized by an additional set of independent model variables. To account for these variables, PARFIT is equipped with an input functionality that allows the user to specify several inputs, which can vary not only between different data sets, but also within each data set. An example is given demonstrating this functionality.

An important problem for systems containing algebraic equations is the consistent initialization and reinitialization of the algebraic variables. This issue is addressed in Section 8.4.

This report concludes the thesis and the overall conclusions are presented in Chapter 9 along with suggestions for future work.

8.1 Multiple Data Sets

Until now, estimation based on only a single data set has been considered. As introduced earlier, a data set is defined as a set of measurements of l_c components at l_t experimental times, which gives a total of $m = l_c l_t$ measurements. Often several experiments are performed, in which the same components are measured, but the experimental conditions are varied between the experiments (e.g. varying temperature). To incorporate all the available information in the estimation, the PARFIT algorithm is extended to handle l_d different data sets each containing l_c components measured at l_t^k experimental times where the superscript 'k' refers to the kth data set. That is, it is assumed that each data

set contains measurements of l_c components. If in one experiment a different number of components are measured, this must be handled by specifying zero weights for the missing components. Varying experimental conditions are handled by introducing an input vector $\boldsymbol{u} \in \mathbb{R}^{n_u}$. For example, in the batch reactor problem (c.f. Example 6.3) data are available at three different temperatures. Thus, the temperature is treated as an input, which is constant within each data set. Each data set must contain the same number of inputs.

Following the notation from Chapter 1, each measurement is characterized as:

$$(c_i, t_i, \tilde{y}_i, \boldsymbol{u}_i)\,, \qquad i = 1, \ldots, m \tag{8.1}$$

in which c_i indicates which component of the state vector $\boldsymbol{y}$ that has been measured, t_i is the time of the measurement and $\tilde{y}_i$ is the measured value. $\boldsymbol{u}_i$ does not denote the ith component of the input vector, but the combination of inputs applying to the ith measurement. PARFIT supports piecewise constant inputs, which may be of relevance in e.g. fed-batch experiments, in which step changes are made to the feed flow rate as illustrated in Example 8.2.

The weighted least squares criterion may be formulated as:

$$\min_{\boldsymbol{\theta}} \; f(\boldsymbol{\theta}) = \frac{1}{2} \sum_{k=1}^{l_d} \sum_{j=1}^{l_c} \sum_{i=1}^{l_t^k} {w_{ij}^k}^2 \left(y_j(t_i, \boldsymbol{\theta}, \boldsymbol{u}_{ik}) - \tilde{y}_{ij}^k \right)^2 \tag{8.2}$$

Details on implementation are given in Appendix A along with guidelines for the use of PARFIT.

8.1.1 The Dow Chemicals Problem Revisited

Example 8.1 (The Dow Chemicals Problem Revisited)
Consider again the Dow Chemicals problem from Example 6.3. The model equations are given by (6.6). Measurements of four concentrations are available at three different temperatures. In this example all measurements are included in the estimation. The 9 unknown model parameters are:

$$\boldsymbol{\theta} = \begin{bmatrix} \alpha_1 & E_1 & \alpha_{-1} & E_{-1} & \alpha_2 & E_2 & K_1 & K_2 & K_3 \end{bmatrix}^T \tag{8.3}$$

The initial conditions are assumed known without error, and the conditions corresponding to the low temperature data are given by (6.9). The initial conditions for the second and third experiment are:

67°C:	100°C:
$y_1(0) = 1.6497$	$y_1(0) = 1.5608$
$y_2(0) = 8.2262$	$y_2(0) = 8.3546$
$y_3(0) = 0.0104$	$y_3(0) = 0.0082$
$y_4(0) = 0.0017$	$y_4(0) = 0.0086$

The initial parameter estimates as provided in Biegler *et al.* (1986) were listed in Table 6.4. To obtain a better scaling of the problem the activation energies are scaled by a factor of 1/10000, $\tilde{E}_i = E_i/10000$, and a logarithmic transformation is introduced for

the equilibrium constants. To reduce the intercorrelation between the parameters in the rate constants, a reparametrisation of the Arrhenius' law (6.7) is applied:

$$\begin{aligned} k_i &= \alpha_i \exp\left(-\frac{E_i}{RT}\right) \\ &= k_{0,i} \exp\left(-\frac{E_i}{R}\left(\frac{1}{T}-\frac{1}{T_0}\right)\right), \qquad i = 1, -1, 2 \end{aligned} \tag{8.4}$$

in which:

$$k_{0,i} = \alpha_i \exp\left(-\frac{E_i}{RT_0}\right)$$

The reference temperature T_0 is chosen as the average temperature over all the performed experiments, $T_0 = 69°\text{C}$. The reparametrisation (8.4) does not change the model responses, but the correlation between the two Arrhenius parameters is reduced.

Parameter	Initial estimate	Final estimate	Marginal confidence interval
$\ln k_{0,1}$	1.194	0.796	±0.102
$\ln k_{0,-1}$	6.565	27.29	±19.37
$\ln k_{0,2}$	1.194	1.107	±0.108
$\tilde{E}_1$	2.000	1.847	±0.074
$\tilde{E}_{-1}$	2.000	2.636	±0.078
$\tilde{E}_2$	2.000	1.882	±0.028
$\ln K_1$	-34.54	-38.76	±0.024
$\ln K_2$	-25.33	-14.26	±19.48
$\ln K_3^*$	-39.14	-39.14	–

Table 8.1. Initial and final parameter estimates for the batch reactor problem using data at three different temperatures. A 95% marginal confidence interval is listed for each parameter. ($*$: fixed during optimization).

As in Example 6.3 K_1 and K_3 are linearly dependent, and therefore K_3 is fixed at its initial value during the optimization. With a tight optimization tolerance (10^{-7}) and a conservative initial step bound in the optimizer, the optimization terminates successfully with the final objective function value 0.207[1] using 16 iterations. Since each iteration requires three consecutive solutions of the DAEs (one for each temperature), the computations are quite demanding.

The initial and final parameter estimates are listed in Table 8.1 and the optimal solution trajectories are plotted in Figure 8.1–8.3. Inspection of the plots shows that systematic errors exist between the data and the model responses, thus the model does not capture all of the variation in the data. The estimated correlation matrix in Table 8.2 shows that $k_{0,-1}$ and K_2 are fully correlated, which supports the observation made by one of the research groups, whose results are presented in Biegler *et al.* (1986). They note that only two of the parameters among $k_{0,-1}$, K_1, K_2 and K_3 can be estimated independently, which in Example 6.3 led to the decision of fixing K_2 in the estimation. In this case the optimization was successful, but the estimates of $k_{0,-1}$ and K_2 both have wide confidence intervals.

This problem still poses many of the numerical difficulties observed in Example 6.3. Loose integration tolerances prevent a successful optimization, and unless the initial step bound in the optimizer is chosen very small, the first correction computed to the

[1] The initial guess for K_1 was changed to 10^{-15} as in Example 6.3.

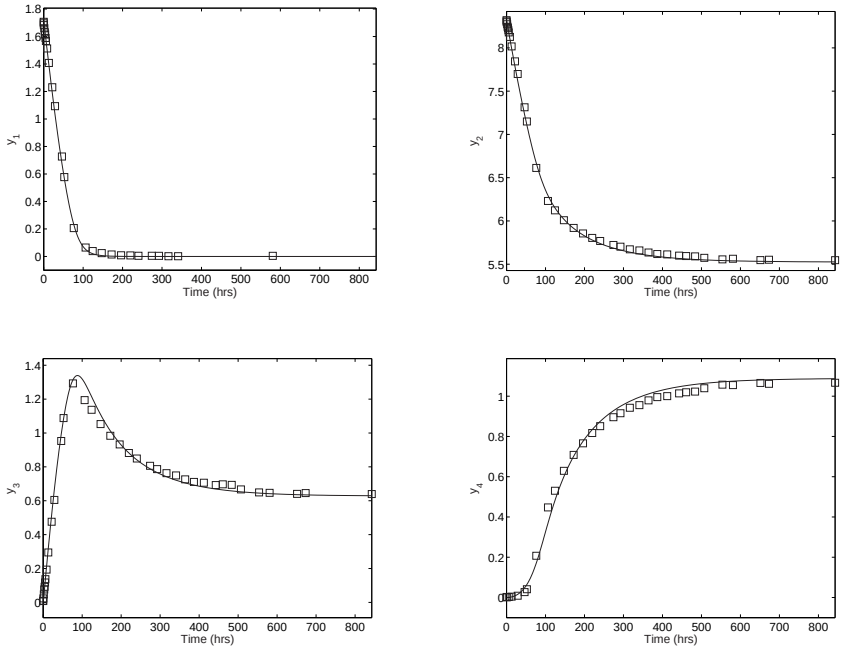


Figure 8.1. Experimental measurements and solution trajectories for the batch reactor problem. The trajectories are computed using the optimal parameters from Table 8.1. Low temperature data ($T = 40°\mathrm{C}$).

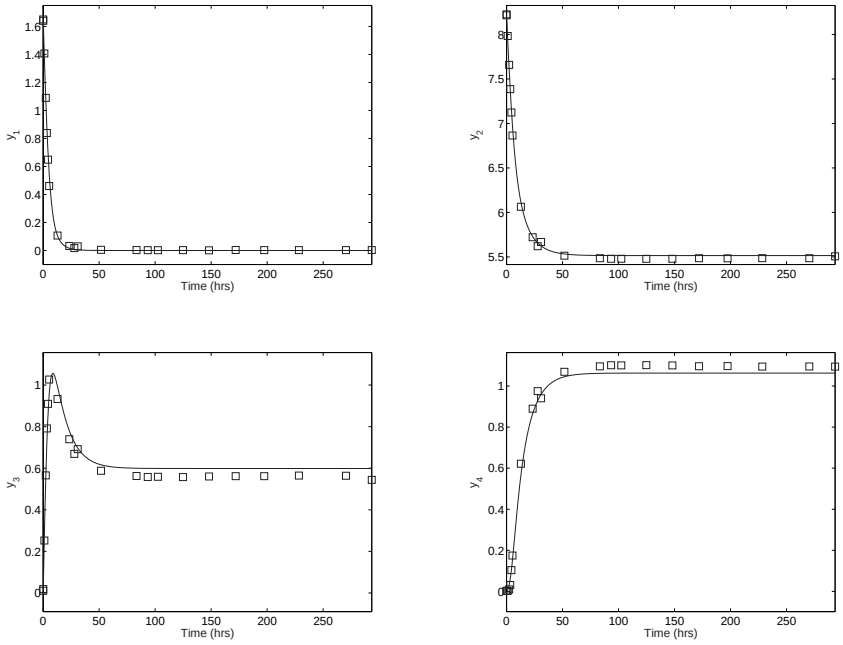


Figure 8.2. Experimental measurements and solution trajectories for the batch reactor problem. Medium temperature data ($T = 67°\mathrm{C}$).

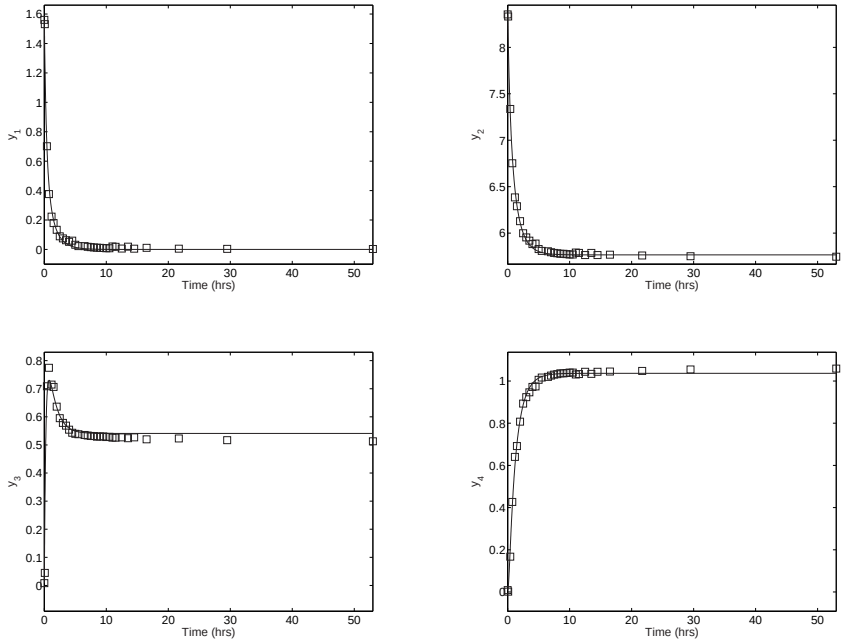


Figure 8.3. Experimental measurements and solution trajectories for the batch reactor problem. High temperature data ($T = 100°\mathrm{C}$).

parameter vector causes the integrator to fail. Inspection shows that, with a large step bound, $\tilde{E}_{-1}$ becomes negative after the first iteration. The problem also illustrates that initial parameter guesses are often far from the optimal solution. The initial estimate for $k_{0,-1}$ is a factor of 10^9 smaller than its final value. The sequence of objective function values is plotted in Figure 8.4, which shows the slow progress made in the initial phase of optimization, in which the optimizer takes small steps in the steepest descent direction.

To conclude this example a comparison is made to the results presented in Biegler *et al.* (1986). As mentioned earlier, five research groups submitted acceptable solution. They tackled the problem in very different ways. Four groups optimized a maximum likelihood criterion using either a full unknown covariance matrix or an unknown diagonal

	$k_{0,1}$	$k_{0,2}$	$k_{0,-1}$	E_1	E_2	E_{-1}	K_1	K_2
$k_{0,1}$	1.000							
$k_{0,2}$	0.753	1.000						
$k_{0,-1}$	-0.873	-0.948	1.000					
E_1	0.879	0.798	-0.887	1.000				
E_2	-0.163	0.207	0.028	-0.246	1.000			
E_{-1}	-0.061	-0.126	0.160	-0.109	0.418	1.000		
K_1	0.089	-0.258	0.209	-0.160	-0.065	0.051	1.000	
K_2	-0.873	-0.949	1.000	-0.887	0.026	0.161	0.209	1.000

Table 8.2. Estimated correlation matrix for the parameters. The numerically largest elements of the correlation matrix are marked in grey (threshold value = 0.9).

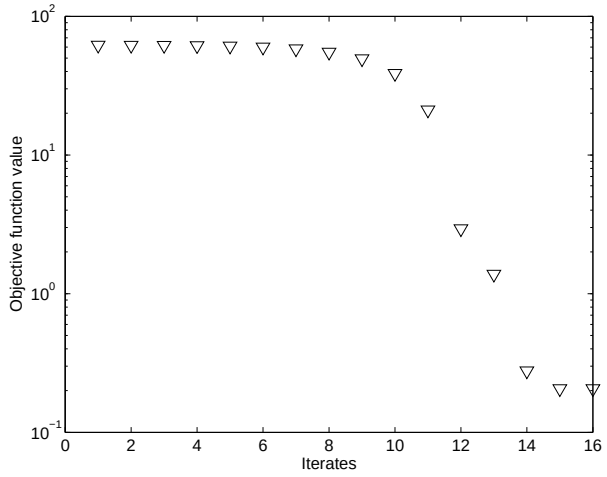


Figure 8.4. The value of the objective function plotted as a function of the iteration number.

covariance matrix, whereas one group used a least squares criterion. Only one group used the original formulation with 6 differential equations and 4 algebraic equations. Furthermore, the five approaches differed in the number of parameters optimized, the number of components regressed, and in the numerical algorithms used for optimization and differential equation solution. Only two groups computed derivative information by solving the sensitivity equations, whereas three groups relied on difference approximations. In general, much effort was put into rearranging the model equations and transforming the parameters before an optimization was attempted.

The results obtained here are compared to the results of the five research groups in Table 8.3. Good agreement is observed.

The batch reactor problem was posed by the Dow Chemical Company to review the state of the art in parameter estimation for reaction engineering problems. Biegler *et al.* (1986) conclude that an interactive approach and a great deal of experience is required to solve difficult parameter estimation problems, and that user friendly software is not yet available that automatically handles all of the difficulties encountered. Now, 20

	Research group no.					
Transformed parameter	(1)	(2)	(3)	(4)	(5)	(6)
$k_1(69°\mathrm{C})$	1.88	1.84	1.94	2.04	2.21	2.22
$k_2(69°\mathrm{C})$	2.73	2.89	2.38	2.68	2.78	3.02
$E_1 \cdot 10^{-3}$	18.74	18.48	18.84	18.26	17.84	18.47
$E_2 \cdot 10^{-3}$	18.88	19.07	17.87	18.41	18.85	18.81
$E_3 \cdot 10^{-3}$	25.67	26.05	25.15	21.90	25.20	26.36
K_1/K_3	1.44	1.44	1.42	1.44	1.43	1.46
$k_{-1}(69°\mathrm{C})K_3/K_2$	9.84	10.92	8.52	9.98	10.14	11.12

Table 8.3. Summary of optimal solutions to the batch reactor problem. The results presented in Biegler *et al.* (1986) (research groups 1–5) are compared to the results obtained in this example (research group no. 6).

years later, computer power has increased dramatically, but estimating parameters in differential equation systems is still a non-trivial task requiring experience and careful attention from the user of parameter estimation software. User friendly software is available (e.g. EASYFIT (Schittkowski, 2002) or gPROMS (Pantelides and Barton, 1993)), but to the knowledge of the author none of the packages automatically handle the difficulties encountered in e.g. the batch reactor problem. Probably the type of difficulty most frequently arising in parameter estimation is for the model maker to "balance" the complexity of the model with the information available in the data such that all of the parameters can be identified. Thus, obtaining fully automated software packages that handles all possible scenarios seems an unattainable goal. Parameter estimation is, inherently, an interactive and iterative process. ■

8.1.2 A Fed-Batch Fermentation Problem

In the above example the initial conditions were assumed known. In some systems this is not a realistic assumption and the initial conditions must be estimated along with the unknown parameters. For systems described by ODEs, PARFIT has an option for estimating initial conditions for all data sets. The extension is straightforward. Since the initial conditions are assumed not to appear explicitly in the right-hand-side functions of the ODEs, the corresponding sensitivity equations, which must be solved to provide the gradient of the objective function, take the form:

$$\frac{\partial}{\partial t}\frac{\partial \boldsymbol{y}}{\partial y_{0i}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial y_{0i}}, \quad \frac{\partial \boldsymbol{y}}{\partial y_{0i}}(t_0) = \boldsymbol{e}_i, \quad i = 1, \ldots, n \tag{8.5}$$

in which y_{0i} denotes the initial condition of state y_i, and $\boldsymbol{e}_i$ denotes the ith unit vector. The sensitivities with respect to initial conditions are already computed in ESDIRK34 (see the outline of the sensitivity algorithm in Appendix A.1). Thus, no extra user intervention is required for initial condition estimation.

Example 8.2 (Parameter Estimation in a Fed-Batch Fermentation Model) The purpose of this example is to (i) illustrate initial condition estimation in multiple data sets and (ii) to show a problem with a time-varying input. The problem considered is a simple model of a fed-batch fermentation process. The problem was studied by e.g. Kuhlmann *et al.* (1998) in the context of robust control and by Kristensen (2003) in the context of parameter estimation in stochastic grey-box models. The process consists of a stirred tank reactor containing water, substrate and biomass, which is fed with a stream containing water and substrate. The model describes growth of biomass on a single substrate with Monod kinetics and substrate inhibition (Kristensen, 2003). The model equations are:

$$\frac{dX}{dt} = \mu(S)X - \frac{FX}{V} \tag{8.6a}$$

$$\frac{dS}{dt} = -\frac{\mu(S)X}{Y} + \frac{F(S_F - S)}{V} \tag{8.6b}$$

$$\frac{dV}{dt} = F \tag{8.6c}$$

in which X [g/L] is the biomass concentration, S [g/L] is the substrate concentration, V [L] is the reactor volume, F [L/h] is the feed flow rate, Y is a yield coefficient and S_F $[g/L]$ is the feed concentration of substrate. The biomass growth rate, $\mu(S)$, is given by the following expression:

$$\mu(S) = \mu_{max}\frac{S}{K_2S^2 + S + K_1} \tag{8.7}$$

in which μ_{max}, K_1 and K_2 are kinetic parameters. Before considering the parameter estimation problem, a brief aside is made on optimal operation of the fed-batch fermentation. Kuhlmann *et al.* (1998) study the problem of optimizing the production of biomass by manipulating the feed flow rate. The problem is to determine the initial conditions and the feed flow rate trajectory that gives optimal productivity in terms of the amount of biomass at the end of the batch. As shown in Kristensen (2003), the problem can be solved by observing that the productivity is maximized when the biomass growth rate is maximized, which leads to the following condition:

$$0 = \frac{d\mu(S)}{dS} = \mu_{max} \frac{K_1 - K_2 S^2}{(K_2 S^2 + S + K_1)^2} \quad \Rightarrow \quad S = \sqrt{\frac{K_1}{K_2}} = S^* \tag{8.8}$$

If the initial substrate concentration is chosen to $S_0 = S^*$, and if the feed flow rate is chosen such that $\frac{dS}{dt} = 0$, then S can be kept at S^*:

$$0 = \frac{dS}{dt} = -\frac{\mu(S_0)X}{Y} + \frac{F(S_F - S_0)}{V} \quad \Rightarrow \quad F = \frac{\mu(S_0)XV}{Y(S_F - S_0)} \tag{8.9}$$

By substituting this expression into (8.6a) and (8.6c), the equations for X and V can be solved (see Kristensen, 2003):

$$X = \frac{ac\exp(at)}{1 + bc\exp(at)} \tag{8.10}$$

$$V = \frac{1 + bc\exp(at)}{1 + bc} V_0, \qquad t \in [t_0, t_f] \tag{8.11}$$

in which t_0 and t_f denote the initial and final times, respectively, and:

$$a = \mu(S_0), \qquad b = \frac{\mu(S_0)}{Y(S_F - S_0)}, \qquad c = \frac{X_0}{a - bX_0}$$

Inserting (8.10) and (8.11) in the expression for the feed flow rate (8.9) gives:

$$F = bX_0V_0\exp(at) \tag{8.12}$$

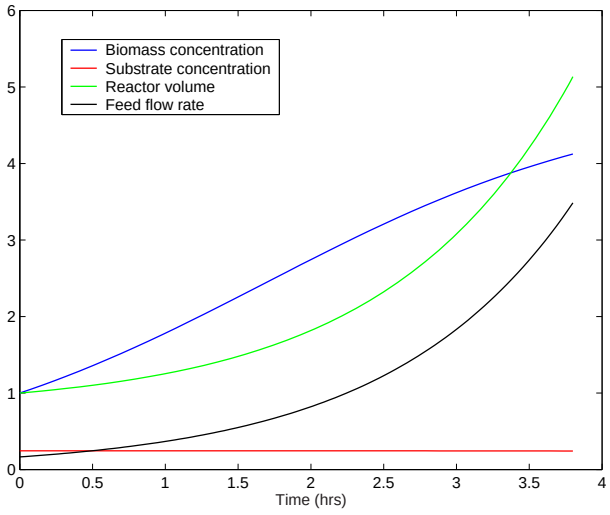


Figure 8.5. Input trajectory and solution trajectories corresponding to optimal operation of the fed-batch fermentation.

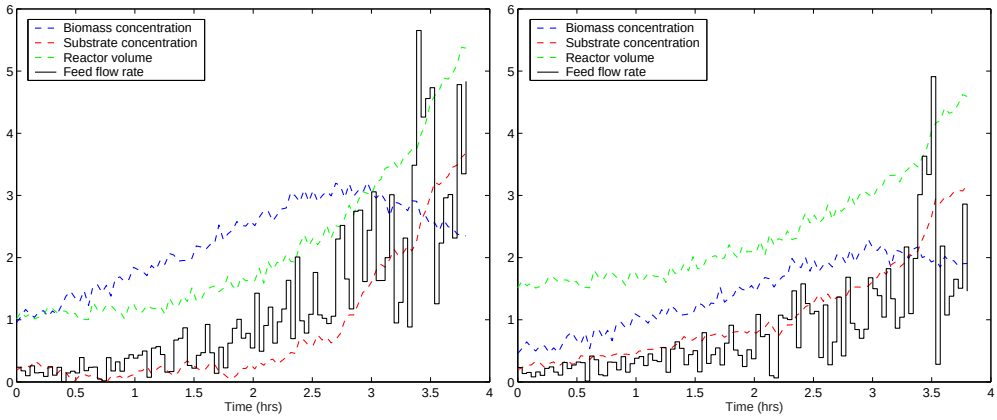
Parameter	Value
Y	0.5
S_F	10g/L
μ_{max}	1h^{-1}
K_1	0.03g/L
K_2	0.5L/g

Table 8.4. Parameter values for the fed-batch fermentation example. The values correspond to the values used by Kuhlmann *et al.* (1998).

Thus, an analytic expression has been derived for the optimal feed flow trajectory. The solution corresponding to optimal operation is depicted in Figure 8.5. The initial conditions $X_0 = 1\text{g/L}$, $S_0 = S^*$ and $V_0 = 1\text{L}$ are used along with the parameters listed in Table 8.4. When operated at optimum the growth rate of biomass is constant meaning that any measurements taken in this state of operation will not contain information that enables the estimation of the kinetic parameters in (8.7). Thus, to uncover the dynamics of the system, two simulated identification experiments are performed, in which the optimal input profile is perturbed with random values drawn from a normal distribution with zero mean and variance $\sigma^2 = 0.25$. Measurements are taken at 100 equidistant points in time and added with random noise according to the following measurement equation:

$$\begin{pmatrix} \tilde{X} \\ \tilde{S} \\ \tilde{V} \end{pmatrix}_j = \begin{pmatrix} X \\ S \\ V \end{pmatrix}_j + \boldsymbol{\varepsilon}_j, \quad \boldsymbol{\varepsilon}_j \in \mathcal{N}(0, \boldsymbol{V}), \quad \boldsymbol{V} = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \tag{8.13}$$

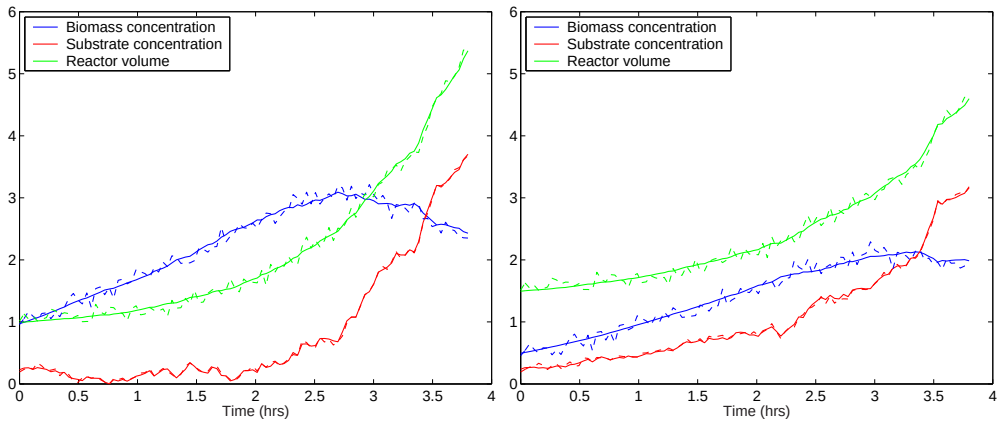
The two experiments are started at different initial conditions. The parameter estimation problem considered here consists of estimating the kinetic parameters μ_{max} and K_1 along with the initial conditions corresponding to each data set. The measurements and perturbed input profiles for the two experiments are plotted in Figure 8.6. In both experiments the perturbations in the input cause the substrate concentration



(a) Experiment no. 1.

(b) Experiment no. 2.

Figure 8.6. Fed-batch data from two identification experiments.



(a) Experiment no. 1.

(b) Experiment no. 2.

Figure 8.7. Experimental measurements and optimal solution trajectories for the fed-batch problem.

to increase, which inhibits the growth of biomass. The unknown parameters in the parameter estimation problem are:

$$\boldsymbol{\theta} = \begin{bmatrix} X_{0,1} & S_{0,1} & V_{0,1} & X_{0,2} & S_{0,2} & V_{0,2} & \mu_{max} & K_1 \end{bmatrix}^T \tag{8.14}$$

in which $X_{0,i}$, $S_{0,i}$ and $V_{0,i}$ refer to the initial values of experiment no. i. Using an optimization tolerance of 10^{-6} the parameter estimates listed in Table 8.5 are obtained. All estimates, except from $X_{0,1}$, are within a 95% confidence interval. The optimal solution trajectories are plotted in Figure 8.7. The estimated correlation matrix is given in Table 8.6. The highest correlation is observed between μ_{max} and K_1.

This example illustrates the importance of handling time-varying inputs. A discontinuous input profile, as in Figure 8.6, puts certain demands on the differential equation solver. Many solvers (e.g. DASPK) have an option for providing continuous output, which does not interfere with the overall integration and time step selection. However, if discontinuities are present, the integration must be restarted at each discontinuity.

Parameter	True value	Final estimate	Marginal confidence interval
$X_{0,1}$	1.000	0.972	$\pm$0.023
$S_{0,1}$	0.245	0.229	$\pm$0.034
$V_{0,1}$	1.000	0.987	$\pm$0.013
$X_{0,2}$	0.500	0.495	$\pm$0.009
$S_{0,2}$	0.245	0.237	$\pm$0.027
$V_{0,2}$	1.500	1.494	$\pm$0.013
μ_{max}	1.000	0.997	$\pm$0.005
K_1	0.030	0.025	$\pm$0.012

Table 8.5. Estimation results for the fed-batch fermentation problem using two data sets. A 95% marginal confidence interval is listed for each parameter. All estimates, except from $X_{0,1}$, are within the confidence interval.

	$X_{0,1}$	$S_{0,1}$	$V_{0,1}$	$X_{0,2}$	$S_{0,2}$	$V_{0,2}$	μ_{max}	K_1
$X_{0,1}$	1.000							
$S_{0,1}$	0.818	1.000						
$V_{0,1}$	0.503	0.141	1.000					
$X_{0,2}$	0.267	0.002	-0.178	1.000				
$S_{0,2}$	0.304	-0.137	0.257	0.240	1.000			
$V_{0,2}$	-0.175	-0.528	0.227	0.206	0.396	1.000		
μ_{max}	-0.223	-0.222	-0.080	-0.033	-0.021	0.561	1.000	
K_1	-0.177	-0.276	0.019	0.039	0.100	-0.156	0.330	1.000

Table 8.6. Estimated correlation matrix for the parameters in the fed-batch fermentation problem. The numerically largest elements of the correlation matrix are marked in grey (threshold value = 0.8).

Being a one-step method ESDIRK34 does not suffer particularly from these restarts. A variable order multi-step method, on the other hand, would need to revert to low order at each restart making the integration inefficient. Thus, by using a one-step method, discontinuous input profiles are handled with only small additional costs. ■

The above example touches on the experimental design aspect. By manipulating the inputs in order to excite the dynamics of the system, the information content in the data obtained is increased, thereby increasing the quality of the resulting parameter estimates. However, the questions remain how to choose the input profiles and possibly also when to place the samples, provided that only a limited number of samples can be made. Consider, for example, a batch experiment, in which the concentration of a chemical species is measured at 10 discrete time points. Should the samples be placed equidistantly, concentrated near the starting point, or concentrated near the end in order to gain as much information as possible? Also, if the experiment can be manipulated (e.g. by varying the temperature), how should the input profile be chosen? Bauer *et al.* (2000) have shown that these questions can be answered by solving a dynamic optimization problem, in which a performance criterion related to the size or shape of the ellipsoidal confidence region for the parameters is minimized. Thus, by perturbing the system and placing the samples "in the right way" the statistical quality of the estimates is maximized. Although a very important and interesting issue, it is beyond the scope of this thesis to address experimental design in more detail.

8.2 Regularization

One of the difficulties often encountered in parameter estimation problems is ill-conditioning, which arises due to e.g. high correlation between the parameters. If no measures are taken to reduce the ill-conditioning, the optimization will often terminate unsuccessfully. The solution of the linear subproblems arising during optimization will be sensitive towards perturbations in the data, and the search directions computed will often be seriously in error. One possible remedy for this problem is to introduce *regularization* into the parameter es-

timation problem to stabilize the solution. Computing regularized solutions to least squares type problems is a huge area of research in itself. The type of regularization considered here is known as *Tikhonov regularization* (Hansen, 1998). The basic idea is to define the regularized solution as the minimizer of the following problem:

$$\min_{\boldsymbol{\theta}} \; f(\boldsymbol{\theta}) = \underbrace{\frac{1}{2}\sum_{i=1}^{m} w_i^2 r_i(\boldsymbol{\theta})^2}_{\text{residual term}} + \underbrace{\frac{1}{2}\sum_{i=1}^{n_p} w_{reg}^2 \left(\frac{\theta_i - \theta_{i,0}}{\theta_{i,ref}}\right)^2}_{\text{regularization term}} \tag{8.15}$$

in which $\theta_{i,0}$ denotes the initial guess for parameter θ_i and $\theta_{i,ref}$ denotes a reference value for θ_i, which should be chosen to the expected order of magnitude[2]. w_{reg} denotes the regularization weight. Thus, the objective function is composed of a residual term and a regularization term. Differentiating the regularization term twice with respect to the parameters gives a diagonal contribution to the Hessian matrix, thus improving the conditioning of the matrix. Specifying $w_{reg} = 0$ reduces the problem to the standard least squares problem. By changing w_{reg} the amount of regularization can be adjusted. The regularization term acts as a penalty to the objective function for any movement of the parameters away from the initial guesses. This action causes bias in the resulting parameter estimates, and the key issue when using regularization is to balance the trade-off between improved conditioning of the system and bias in the estimates.

A useful tool when analyzing ill-conditioned problems is the singular value decomposition (SVD). The SVD extracts information from a matrix by decomposing it into left singular vectors, singular values and right singular vectors. Let $\boldsymbol{A} \in \mathbb{R}^{m \times n_p}$ be a rectangular matrix with $m \geq n_p$, then:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \sum_{i=1}^{n_p} \boldsymbol{u}_i \sigma_i \boldsymbol{v}_i^T \tag{8.16}$$

in which $\boldsymbol{U} = \begin{bmatrix}\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n_p}\end{bmatrix}$ and $\boldsymbol{V} = \begin{bmatrix}\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n_p}\end{bmatrix}$ are matrices with orthonormal columns, and $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_{n_p})$ has non-negative diagonal elements appearing in non-increasing order:

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{n_p} \geq 0 \tag{8.17}$$

The condition number of $\boldsymbol{A}$ is defined as the ratio between the largest and smallest singular values, σ_1/σ_{n_p}. Furthermore, from (8.16) the following relations are obtained:

$$\|\boldsymbol{A}\boldsymbol{v}_i\|_2 = \sigma_i, \qquad i = 1, \ldots, n_p \tag{8.18}$$

If a small singular value exists compared to σ_1, then (8.18) shows that the columns of $\boldsymbol{A}$ are nearly linearly dependent meaning that $\boldsymbol{A}$ is nearly rank deficient. Thus, by performing an SVD, information about the conditioning of the system can be extracted. More specifically, by inspecting the right singular vectors, the linearly dependent parameters in the parameter estimation problem can be identified. This is illustrated in the next example.

[2]The implementation in PARFIT uses $\theta_{i,ref} = \theta_{i,0}$ as default. Thus, the reference values need not be specified unless values different from $\theta_{i,0}$ are desired.

Example 8.3 (Using Regularization in the Dow Chemicals Problem (I)) Consider the Dow Chemicals problem with 3 data sets and 8 parameters as treated in Example 8.1. Using the initial parameter guesses as provided in Biegler *et al.* (1986), the initial Hessian approximation is computed. Figure 8.8 shows the singular values of the Hessian approximation using (i) no regularization and (ii) using regularization with $w_{reg} = 1$. By introducing regularization the condition number of the initial Hessian matrix is reduced by a factor of 10^4.

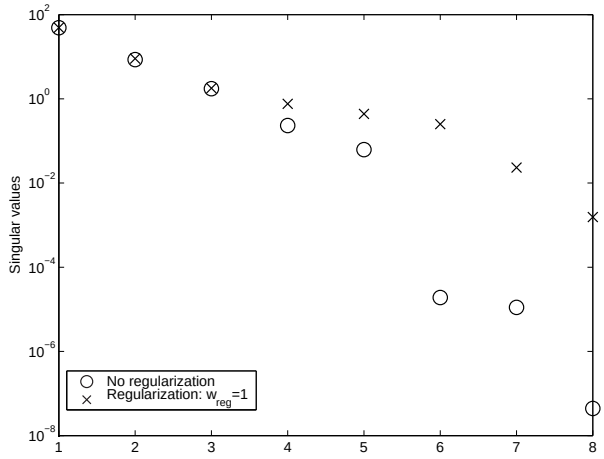


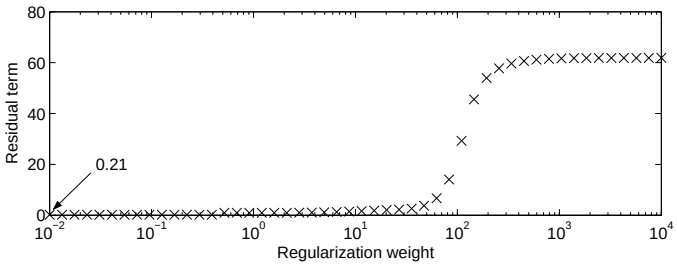
Figure 8.8. Plot of singular values for the initial Hessian approximation in the Dow Chemicals problem.

Inspection of the last right singular vector in the SVD of the Hessian matrix in the unregularized case shows that the smallest singular value can be identified with a linear combination consisting mainly of columns 3 and 8 of the Hessian. These columns correspond to the parameters $k_{0,-1}$ and K_2, which is in good agreement with the observations made regarding correlation in Example 8.1. ■

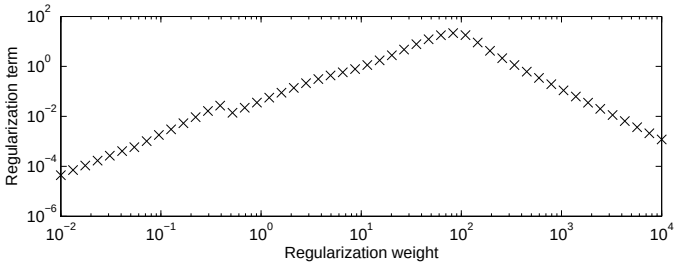
The above example illustrates the effect of introducing regularization. The strong correlation between some of the parameters is reduced at the expense of bias in the resulting parameter estimates. The important question remains how to choose an "adequate" level of regularization. This question is studied in the next example.

Example 8.4 (Using Regularization in the Dow Chemicals Problem (II)) Consider again the Dow Chemicals problem. In Example 8.1 it was found that 8 parameters could be optimized, only if the initial guess for K_1 was changed from 10^{-17} to 10^{-15}. In this example the influence of the regularization weight on the objective function value, and hence on the optimal solution, is studied, and it is shown that the optimal solution to the Dow Chemicals problem can be traced from the unaltered starting guess by successively lowering the regularization weight.

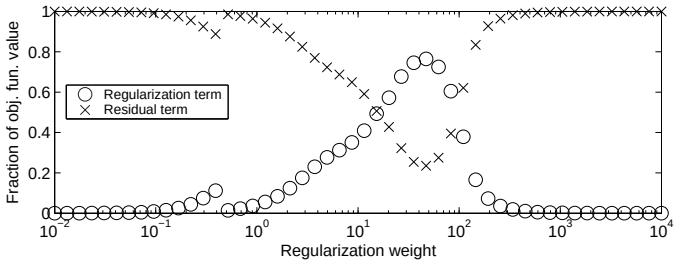
Using the same starting guess as in Example 8.1 the parameter estimation problem with 3 data sets and 8 parameters is solved for varying values of the regularization weight. The results are illustrated in Figure 8.9. Figure 8.9a–b show the value of the residual term and the regularization term in the objective function, respectively, whereas Figure 8.9c shows the fraction of the objective function value corresponding to each of these two terms. Finally, the condition number of the Hessian approximation evaluated at the optimal solution is shown in Figure 8.9d. Figure 8.10 shows the values of the transformed parameters as a function of the regularization weight.



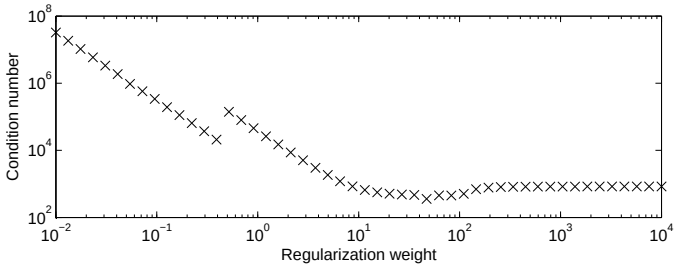
(a) Residual term in objective function.



(b) Regularization term in objective function.



(c) Fraction of objective function corresponding to the residual and regularization terms, respectively.



(d) Condition number for the Hessian matrix evaluated at optimum.

Figure 8.9. Influence of regularization on the objective function in the Dow Chemicals problem. The values of the residual and regularization terms in (8.15) are plotted as a function of the regularization weight along with the condition number for the Hessian approximation.

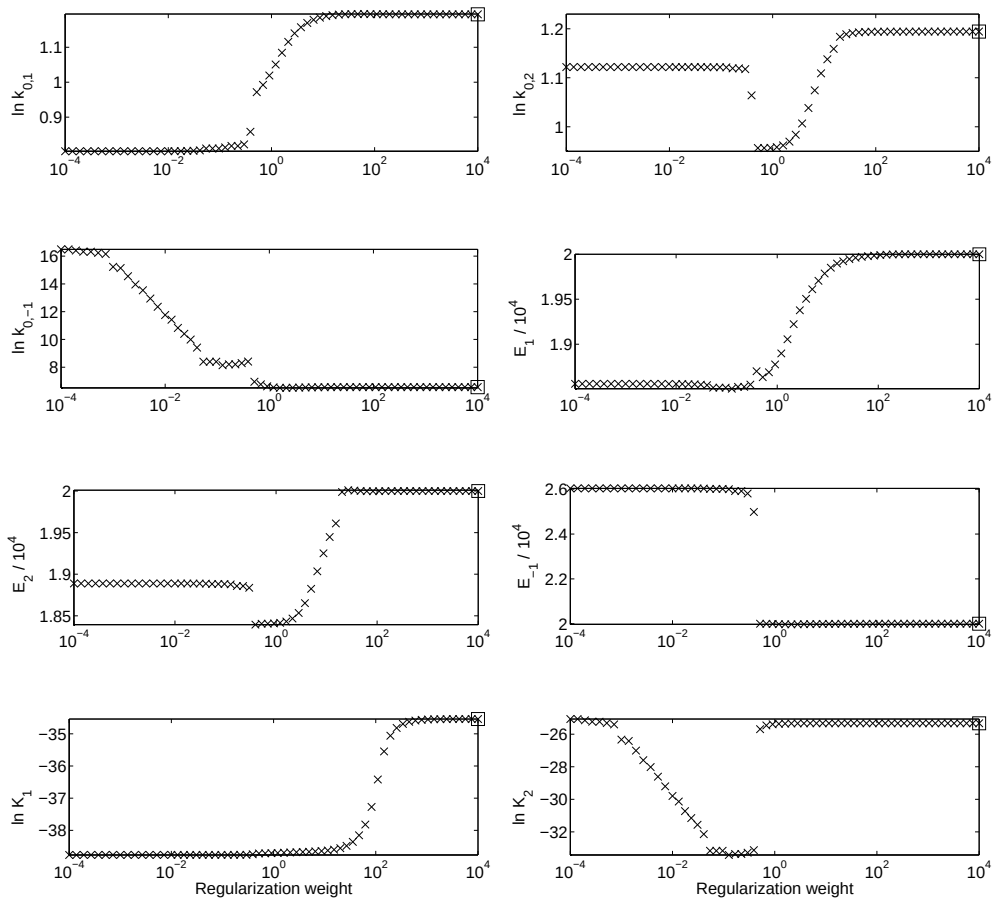


Figure 8.10. Optimal parameter values in the Dow Chemicals problem plotted as a function of the regularization weight. The interval for the regularization weight is expanded compared to the plots in Figure 8.9 in order to show the behaviour with only a small amount of regularization. (□ : starting guess for the parameters).

Looking at the plots, several interesting observations are made. The plots are generated by successively lowering the regularization weight. For $w_{reg} > 100$ the optimization problem is dominated by the regularization part. The information in the actual problem is lost leaving a large residual part and only the regularization part is minimized. For $w_{reg} \simeq 100$ the regularization part has a maximum, whereas a rapid decrease occurs in the residual part. This rapid decrease coincides with a decrease in K_1 (c.f. Figure 8.10), whereas the major changes in the remaining parameters occur around $w_{reg} = 1$. For w_{reg} approximately between 10 and 100 the objective function value is dominated by the regularization part. In this interval the optimization problem gradually changes from being dominated by regularization to being dominated by residuals. For $w_{reg} < 10$ the residual part levels off at a value around 0.21, whereas the regularization part decreases, as expected, with decreasing regularization weight.

The key question is which regularization weight to choose. The goal is to balance the trade-off between improved conditioning and too much regularization giving bias in the estimates. The regularization problem may be regarded as a filtering problem. Enough filtering should be applied to stabilize the solution without loosing information in the actual problem. Comparing Figure 8.9c and d indicates that $w_{reg} \simeq 0.2$ is a reasonable choice for this specific problem. The conditioning is improved substantially (almost)

without affecting the objective function value. The jump in the condition number at $w_{reg} \simeq 0.5$ is caused by the jumps in the parameters.

Inspection of the parameter plots shows that all parameters reach a "steady state" for $w_{reg} < 0.2$ except from $k_{0,-1}$ and K_2, which are drifting due to their high correlation (c.f. Example 6.3).

To conclude this example an experiment is performed, in which the unaltered initial parameter guesses, as provided in Biegler *et al.* (1986), are used. Earlier attempts without regularization showed that the optimization algorithm terminated unsuccessfully after a few iterations unable to compute a step giving a sufficient decrease in the objective function. By experimenting it is found that performing three consecutive optimizations with $w_{reg} = 1.00, 0.20, 0.05$ the optimal solution obtained in Example 6.3 is reproduced (final objective function value = 0.21). Hence, using regularization can be a strong tool for tracing the optimal solution from poor initial parameter guesses. Further experiments show that the optimal solution can be traced, even when the initial parameter guesses are perturbed "in the wrong direction". ■

To summarize, two important observations are made in the Dow Chemicals problem regarding the use of regularization:

- It is possible to choose the regularization weight such that the conditioning is improved substantially without affecting the objective function value. That is, there seems to be a level for the amount of regularization that allows the benefits without "suffering" (too much) bias in the estimates.
- The optimal solution can be traced by performing a sequence of optimizations, in which the amount of regularization is reduced. This technique has great potential in terms of improving the global properties of the algorithm, thereby improving the robustness towards poor initial parameter guesses.

8.3 Numerical Difference Approximations

The use of PARFIT requires, besides specification of the model equations, a user provided subroutine for computation of the Jacobian of the DAE system along with the partial derivatives with respect to the parameters being estimated:

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial y_1} & \cdots & \frac{\partial f_n}{\partial y_n} \end{bmatrix}, \qquad \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \cdots & \frac{\partial f_1}{\partial \theta_{n_p}} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial \theta_1} & \cdots & \frac{\partial f_n}{\partial \theta_{n_p}} \end{bmatrix} \tag{8.19}$$

Experience tells that users tend to make erroneous implementations of the Jacobian, so in order to provide more user friendliness an option is included for approximating the derivatives by finite difference quotients. A simple forward difference approximation is used. The jth column of the Jacobian is approximated as:

$$\frac{\partial \boldsymbol{f}}{\partial y_j} \approx \frac{\boldsymbol{f}\left(t, (y_1, \ldots, y_j + \Delta y_j, \ldots, y_n), \boldsymbol{\theta}\right) - \boldsymbol{f}\left(t, (y_1, \ldots, y_j, \ldots, y_n), \boldsymbol{\theta}\right)}{\Delta y_j} \tag{8.20}$$

in which Δy_j is a suitably chosen small perturbation. A similar approximation is used for the matrix of partial derivatives with respect to the parameters.

Some care has to be exercised in choosing the small perturbations Δy_j. If they are chosen too large, the truncated higher order terms in the Taylor expansion become significant rendering the approximations inaccurate. On the other hand, if they are chosen too small, the subtraction in the numerator of (8.20) is subject to loss of significance. As a rule of thumb, to balance these two phenomena, the perturbations should be chosen as (Nielsen, 2000):

$$\Delta y_j = \sqrt{\epsilon_m}|y_j| \tag{8.21}$$

in which ϵ_m denotes the machine precision[3]. The option for numerical difference derivatives has been tested in the gas-oil cracking problem and the Dow Chemicals problem. All results were reproducible with only minor differences in the computational statistics, such as total number of integration steps, etc.

8.4 Consistent Initialization of DAEs

An important issue, which so far has been silently passed over, is the consistent initialization of DAEs. In case of algebraic equations, the initial conditions for a subset of the dependent variables cannot be specified arbitrarily. In earlier chapters a compact notation for the DAE system was used for convenience (c.f. (1.2) in Chapter 1). The class of problems actually treated by PARFIT is index one DAEs in *semi-explicit* from:

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\theta}), \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \tag{8.22a}$$

$$\boldsymbol{0} = \boldsymbol{g}(t, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\theta}), \qquad \boldsymbol{z}(t_0) = \boldsymbol{z}_0 \tag{8.22b}$$

in which the state vector has been separated into differential variables, $\boldsymbol{y} \in \mathbb{R}^{n_d}$, and algebraic variables, $\boldsymbol{z} \in \mathbb{R}^{n_a}$. $\boldsymbol{\theta} \in \mathbb{R}^{n_p}$ denotes the vector of parameters to be estimated. $\boldsymbol{f}(\cdot)$ and $\boldsymbol{g}(\cdot)$ are functions mapping $\mathbb{R} \times \mathbb{R}^{n_d} \times \mathbb{R}^{n_a} \times \mathbb{R}^{n_p}$ into $\mathbb{R}^{n_d}$ and $\mathbb{R}^{n_a}$, respectively. Given $\boldsymbol{y}_0$, the problem of computing consistent initial values consists of determining $\boldsymbol{z}_0$ such that (8.22b) is satisfied. If started from arbitrary initial values the DAE integrator is likely to terminate unsuccessfully due to repeated error test failures in the first step.

Inconsistencies in the algebraic equations arise in chemical process models for many reasons. For example, a discontinuous input profile causes an inconsistency at each discontinuity. Other examples include process inherent discontinuities such as phase changes. For the class of parameter estimation problems treated by PARFIT inconsistencies arise in three different places:

1. The first call made to the integrator in the first objective function evaluation requires a consistent *initialization* of the algebraic variables. The user must provide an initial guess for $\boldsymbol{z}_0$.
2. Each correction to the parameter vector computed by the optimizer calls for a *reinitialization* of the algebraic variables. The consistent values computed during the previous objective function evaluation are used as starting guesses.

[3]PARFIT is a double precision Fortran implementation, for which $\epsilon_m = 2^{-53} \simeq 10^{-16}$ on most computers.

3. Discontinuities in the inputs require reinitializations. PARFIT supports piecewise constant inputs. Thus, whenever a change occurs in one of the inputs, a reinitialization is performed.

Process inherent discontinuities or *events* are not supported in PARFIT.

Several researchers have studied the problem of computing consistent initial values (Brown *et al.*, 1998; Kröner *et al.*, 1992; Majer *et al.*, 1995; Najafi *et al.*, 2004). The nonlinear equations (8.22b) are often solved with a Newton type method, but much of the work presented concerns DAE problems of a more general class than (8.22) such as fully implicit DAEs or DAEs of index greater than one. Majer *et al.* (1995) and Najafi *et al.* (2004) suggest using a *continuation method* instead of a Newton type method to overcome the problems arising from the limited region of convergence for Newton's method. Continuation methods are not considered here, but the basic idea for solving $\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{0}$ is to introduce a continuation parameter, λ. Instead of solving $\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{0}$ directly, an augmented system defined by:

$$\boldsymbol{h}(\boldsymbol{z}, \lambda) = (1 - \lambda)\boldsymbol{g}^0(\boldsymbol{z}) + \lambda \boldsymbol{g}(\boldsymbol{z}), \qquad \lambda \in [0, 1] \tag{8.23}$$

is solved, in which $\boldsymbol{g}^0(\boldsymbol{z})$ is chosen to be a simple function such that $\boldsymbol{h}(\boldsymbol{z}, 0) = \boldsymbol{0}$ has a known solution. At $\lambda = 1$ the original problem is recovered. Continuation methods trace the solution from $\lambda = 0$ to $\lambda = 1$.

The initializations and reinitializations are handled in PARFIT by using the LMDER optimization algorithm which is already used for parameter optimization. When applied to a set of nonlinear equations, the Levenberg-Marquardt method reduces to Newton's method, if the Levenberg-Marquardt parameter is chosen to 0. One step in the solution of $\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{0}$ consists of solving the linear system:

$$\left(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}^T + \mu \boldsymbol{I} \right) \cdot \Delta \boldsymbol{z} = -\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}} \boldsymbol{g}(\boldsymbol{z}) \tag{8.24a}$$

and updating the solution vector:

$$\boldsymbol{z}^{(k+1)} = \boldsymbol{z}^{(k)} + \Delta \boldsymbol{z} \tag{8.24b}$$

A conservative initial value is used for the trust region parameter, μ. Since the DAE system is assumed to be in semi-explicit form, in which the differential equations precede the algebraic equations, the values of $\boldsymbol{g}(\boldsymbol{z})$ and $\partial \boldsymbol{g}/\partial \boldsymbol{z}$ are readily obtained form the user provided subroutines. Hence, no additional information is required from the user for the initialization calculation. An option is included for approximating $\partial \boldsymbol{g}/\partial \boldsymbol{z}$ numerically.

Example 8.5 (Consistent Initialization in the Dow Chemicals Problem)
In earlier examples dealing with the Dow Chemicals problem consistent initial values were supplied and analytic expressions were used for the reinitializations due to the simple structure of the algebraic equations. This example illustrates the use of LMDER for the consistency calculations.

The parameter estimation problem with 8 unknown parameters and 3 data sets, as in Example 8.1, is considered. The following inconsistent initial values are used for each

data set:

$$y_7(0) = 1.0$$
$$y_8(0) = 1.0$$
$$y_9(0) = 1.0$$
$$y_{10}(0) = 1.0$$

Each objective function evaluation requires three (re)initializations, one for each data set. The average number of iterations required for the initializations during one objective function evaluation is plotted as a function of the overall iteration number in Figure 8.11 along with the corresponding value of the objective function. The observed perfor-

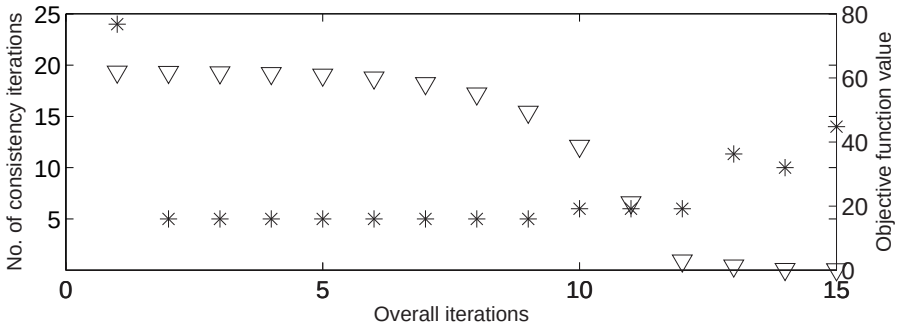


Figure 8.11. Performance statistics for the consistency calculations in the Dow Chemicals parameter estimation problem. The average number of iterations required for the (re)initializations are plotted as a function of the overall iteration number along with the corresponding value of the objective function. ($*$: left axis, $\bigtriangledown$: right axis).

mance seems reasonable. The initialization requires 24 iterations, which is much higher than the number required for the subsequent reinitializations. The corrections computed to the parameter vector are small in the beginning due to a conservative value of the trust region parameter which allows for easy reinitializations. The major "jumps" in the parameters occur near the end of the optimization requiring more iterations in the consistency calculations. Further experiments indicate that the initializations are insensitive towards the starting guesses for the algebraic variables. ■

8.5 Noise Corrupted Data

As shown in Section 8.2, using regularization improves the robustness of the parameter estimation algorithm towards poor initial parameter guesses. The effect of measurement noise on the robustness, and especially on the quality of the estimates, is investigated in this section.

Example 8.6 (Influence of Measurement Noise on Parameter Estimates)
The gas-oil cracking model is used as test problem. For completeness, it is restated here:

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} -(k_1 + k_3)y_1^2 \\ k_1 y_1^2 - k_2 y_2 \end{bmatrix}, \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{8.25}$$

A number of simulated experiments are performed, in which measurements are obtained at 10 equidistant points in time and perturbed with random noise according to the

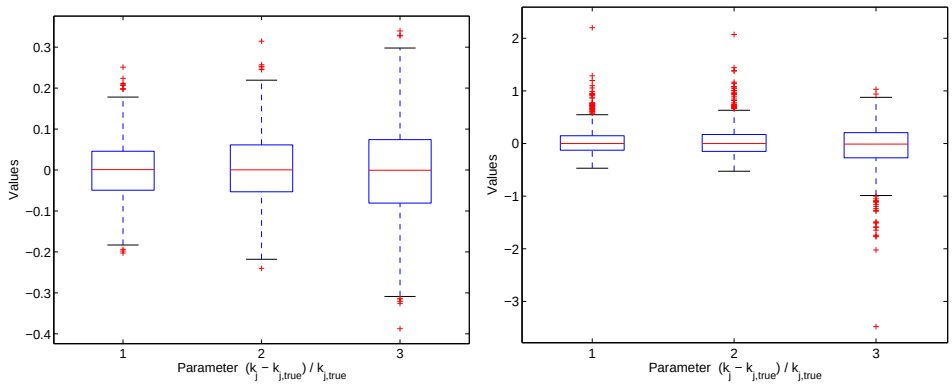
following measurement equation:

$$\begin{bmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{bmatrix}_j = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}_j + \boldsymbol{\varepsilon}_j, \quad \boldsymbol{\varepsilon}_j \in \mathcal{N}(0, \boldsymbol{V}), \quad \boldsymbol{V} = \begin{bmatrix} \sigma_{11} & 0 \\ 0 & \sigma_{22} \end{bmatrix} \tag{8.26}$$

The following parameter values are used when generating data:

$$k_1 = 12, \quad k_2 = 8, \quad k_3 = 8 \tag{8.27}$$

Ideally, when reestimating k_1, k_2 and k_3 from noise corrupted data, the true values would be recovered on average. To investigate this[4], 1000 data sets are generated with $\sigma_{11} = \sigma_{22} = 10^{-4}$ and another 1000 data sets with $\sigma_{11} = \sigma_{22} = 10^{-3}$. One optimization is performed for each data set using default settings in PARFIT (no regularization). Initial conditions are not estimated. The variations in the resulting parameter estimates are illustrated in Figure 8.12 in terms of boxplots. To obtain comparable plots, the true parameter values, $k_{j,true}$, were subtracted from the estimates and the results were then scaled by $k_{j,true}$. The red line in the boxplots indicate the median, whereas the upper and lower bounds of the boxes define the upper and lower quartiles. The height of the box is referred to as the *interquartile range* (IQR). The length of the whiskers extending from each end of the box define the values within $1.5 \cdot \text{IQR}$ of the box. The values beyond the end of the whiskers are regarded as outliers.



(a) Variance of measurement noise : $\sigma_{11} = \sigma_{22} = 10^{-4}$.

(b) Variance of measurement noise : $\sigma_{11} = \sigma_{22} = 10^{-3}$.

Figure 8.12. Boxplots illustrating the variation in the parameter estimates obtained from 1000 optimizations. Results are shown for two different noise levels in the measurements used for estimation. (Notice the different scaling of the y-axis in the two plots).

The estimates obtained from the low noise level data are centered around the true values with a few outliers on each side. However, as the noise level is increased, the variation in the estimates increases dramatically, and the outliers are distributed unevenly around the true values. Thus, the plots indicate that the estimates are increasingly biased when the measurement noise increases. The medians are still close to the true values, but the mean values differ. This apparent drifting of the mean values of the parameter estimates is investigated in Figure 8.13, which shows mean values as a function of the measurement noise level. For each noise level 100 optimizations are performed. The

[4]Inspection of a few plots of data points indicated that $10^{-4} - 10^{-3}$ is a reasonable interval for the variance.

mean parameter estimates are plotted along with error bars indicating a 95% confidence interval. The implicit assumption here, which has also been used in previous examples when constructing confidence intervals, is that the parameter estimates are normally distributed. Due to the variation in the estimates the condidence intervals are wide and the mean parameter estimates are not significantly different from the true values in a statistical sense. Close inspection of the plots in Figure 8.13 shows a slight indication of the drifting phenomenon, but all mean values are clearly within the confidence intervals.

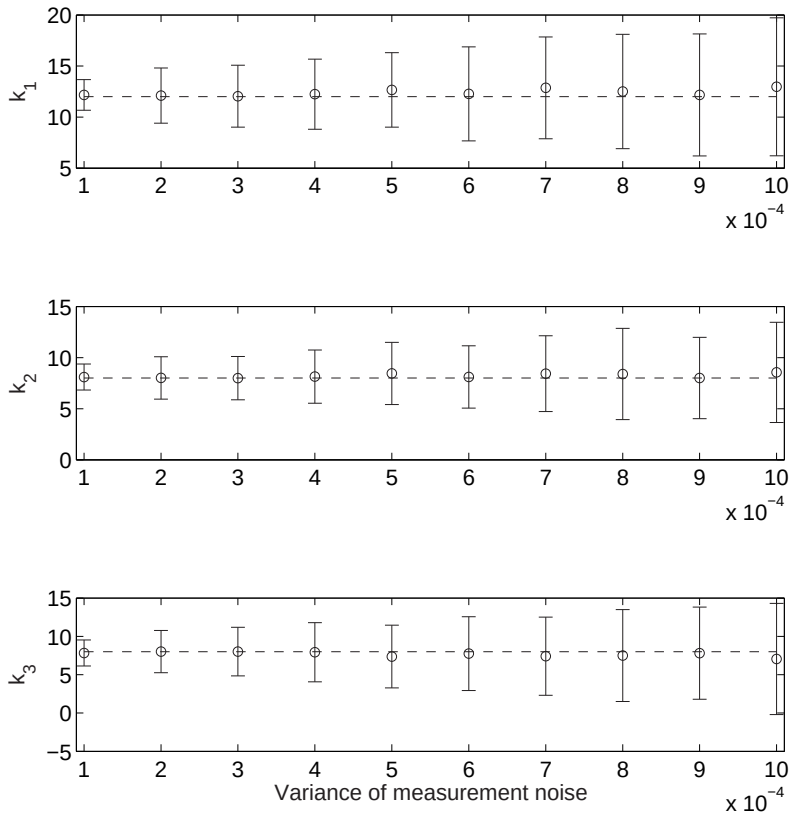


Figure 8.13. Mean value of parameter estimates plotted as a function of the measurement noise level. Each mean value is based on 100 estimations. The errorbars indicate a 95% confidence interval. (Dashed line : true parameter value).

Regarding robustness in the above estimations, all optimizations terminate successfully. Estimation using the high noise level data requires, on average, more iterations compared to using the low noise level data. Moreover, greater variation is observed in the number of iterations required for convergence when using the high noise level data. ■

The validity of the above assumption about normally distributed measurement noise may be questionable in many realistic applications, but the example still illustrates possible effects of measurement noise. The numerical algorithms seem robust towards noise corrupted data, keeping in mind the assumption about normally distributed noise. A different noise structure may lead to a different result.

9

Conclusion

The primary focus of the work presented in this thesis has been on the practical solution of parameter estimation problems in nonlinear dynamical systems. More specifically, parameter estimation problems in systems described by ordinary differential equations or differential-algebraic equations have been considered.

Essentially two numerical algorithms are needed when solving parameter estimation problems in dynamical systems: A differential equation solver for solving the underlying model equations along with the corresponding sensitivity equations and an optimizer for optimizing the parameters. Separately, these two areas of numerical analysis are well understood, but possible improvements remain regarding the particular interaction of differential equation solution and optimization encountered in dynamical parameter estimation, which has motivated the work presented in this thesis. The literature abounds with parameter estimation case studies severely lacking a qualified motivation for the numerical algorithms employed. In particular, the generation of gradient information for the optimizer often seems to suffer from naive approaches, in which the gradients are either approximated by finite difference schemes or calculated by solving the sensitivity equations with an off-the-shelf ODE solver, causing unnecessary inefficiency of the overall algorithm.

By understanding the advantages and disadvantages of the numerical algorithms involved, and by gaining experience through numerous numerical experiments with varying choices of differential equation solvers and optimization algorithms, the goal of this thesis has been to provide a systematic framework, consisting of tools and guidelines, for the practical solution of parameter estimation problems in dynamical systems.

The first part of the thesis was devoted to a review of the relevant literature emphasizing numerical methods for optimization and differential equation solution. The second part of the thesis covered a series of benchmark tests, in which the efficiency of different numerical algorithms were compared. A significant part of the work presented in this thesis has been put into developing a flexible tool for parameter estimation (PARFIT) based on the knowledge gained from the benchmark tests. This work was initiated in the second part and continued throughout the final part.

The different issues of dynamical parameter estimation investigated in this thesis fall into three groups: *Efficiency* issues, *robustness* issues and *flexibility* issues. Regarding efficiency, a specifically tailored equation solver of the Runge-Kutta family was used for the state and sensitivity integration. The solver

exploits the special structure of the sensitivity equations and reuses information computed for the state integration to speed up the sensitivity integration. It was shown that this method was comparable in performance to the much larger package DASPK, which is based on a BDF method. Furthermore, it was shown that the efficiency could be improved by modifying the Levenberg-Marquardt method used for optimization to make it more suitable to dynamical systems, and by using a simple tolerance selection mechanism, which adjusts the optimization and equation solver tolerances depending on the distance from the optimal solution. In one example, however, it was observed that using the tolerance selection mechanism could prevent a successful optimization.

The robustness of parameter estimation software is a key issue. Regularization was introduced to stabilize the solution process for ill-conditioned problems. A simple technique, in which a sequence of optimizations are performed with a successively lowered value of the regularization weight, was proposed for tracing the optimal solution from a poor initial parameter guess. The technique proved useful when applied to a notoriously difficult problem from reaction engineering. Moreover, an automatic consistency calculation was implemented for systems with algebraic equations to ensure robustness towards inconsistencies arising due to inconsistent initial values provided by the user or discontinuities in some of the inputs.

The PARFIT program was developed to provide a flexible environment facilitating the setup of parameter estimation problems. The choices and decisions made at the various stages of the design were motivated throughout the thesis. The class of problems treated is limited to systems modelled by differential-algebraic equations, but flexibility is provided in terms of allowing multiple data sets in the estimation and time varying inputs. In general, much attention was paid to building an easy-to-use program. Default settings have been supplied, which limits the amount of expertise required from the user. The current features in PARFIT are summarized below:

- PARFIT estimates unknown problem parameters from experimental data using a weighted least squares fitting criterion. Systems described by index one differential-algebraic equations (DAEs) in semi-explicit form are handled.
- Multiple data sets are allowed in the estimation. Each data set consists of experimental time values, measurements, weight factors and inputs. Piecewise constant inputs are supported.
- For pure ODE systems an option exists for estimating initial conditions corresponding to each data set.
- Tikhonov regularization can be used for suspected ill-conditioned problems.
- An adaptive tolerance selection mechanism is used to improve efficiency.
- Numerical difference approximations can be used for the DAE system Jacobian and the partial derivatives of the right-hand-side functions of the DAE with respect to the parameters being estimated.
- Inconsistent algebraic equations are automatically initialized. A reinitialization is performed after each discontinuity, which arises due to either a

correction computed to the parameter vector or a discontinuity in one of the inputs.

- Parameters are scaled internally. An option exists for user provided scale factors.
- The data must be scaled using the individual weight factors in the data sets. Different scaling strategies were suggested in Section 6.1.
- Approximate covariance and correlation matrices for the parameters are computed.

Current limitations of PARFIT are discussed in the next section. Assuming that PARFIT, or a similar program, is available, the following systematic approach to parameter estimation can be applied:

1. Obtain initial parameter values using prior physical knowledge.
2. Study the problem by simulation using initial parameter values. Ensure physically meaningful results.
3. Scale all parameters to be within approximately the same order of magnitude. Apply logarithmic transformations to parameters that are known to be positive.
4. Scale the data if some of the components are measured on different scales or if information is available about the error structure in the data.
5. Compute sensitivities. Diagnose possible linear relations among the parameters from sensitivity plots or by performing an SVD analysis of the initial Hessian approximation.
6. If the problem is ill-conditioned:

   either eliminate dependencies by fixing one or several parameters in the optimization or reformulate the model.

   or apply regularization.
7. Perform an optimization using default settings.
8. If the optimization is unsuccessful, possible actions are:

   use more conservative settings by tightening the optimization tolerances and the initial trust region parameter.

   perform multiple optimizations, in which the regularization weight is reduced successively, thereby trying to trace the optimal solution.

   inspect the correlation matrix at the point of termination. Identify linear dependencies.

In conclusion, the main result of the work presented in this thesis is the framework, consisting of the PARFIT tool along with guidelines for a systematic approach to the many difficulties often encountered, which facilitates parameter estimation in dynamical systems. The suggested framework does not guarantee success, but it certainly increases the chance of success. Due to the close link to model building and experimental design, estimating parameters is inherently an iterative and interactive process, in which models are reformulated, experiments redesigned and parameters reestimated.

9.1 Suggestions for Future Work

During the course of the work presented in this thesis a number of related problems were encountered, the treatment of which was beyond the scope of the work. Some of these problems are presented in this section as suggestions for future work.

The most obvious suggestion for future work is to address some of the current limitations of PARFIT. In the current version of PARFIT it is assumed that the states of the model are measured directly. A more general formulation would include a measurement equation relating the measured quantities to the states of the model. The extension in PARFIT is straightforward requiring an extra user supplied subroutine, in which the functions mapping the states to the outputs are provided. Given the parametric sensitivities of the model states, the same subroutine should return the overall derivatives of the measurement functions for use when building the gradient of the objective function. Another possible extension is to implement a functionality for continuous output in the integrator. Currently the integrator is stopped at each sample to provide the model output which interrupts the stepsize controller. Instead an interpolation formula having the same order of accuracy as the integration scheme (Enright *et al.*, 1986) can be used to provide output at the desired points in time. The ESDIRK34 integrator is already prepared for this extension, the necessary coefficients for the interpolant being provided in Kristensen *et al.* (2004a). Along these same lines, PARFIT could be extended to handle systems with discrete events. In this case the continuous extension is required to accurately detect when an event occurs. A complete extension to discrete event systems is, however, not as straightforward as the continuous output functionality.

With respect to robustness a possible topic for future work is to implement a mechanism for rejection of a step computed by the optimizer before the objective function is fully evaluated. In some circumstances the optimizer computes a correction to the parameter vector that is far off causing severe stiffness of the model equations or introducing positive eigenvalues. To improve both efficiency and robustness in these situations it would be desirable to be able to reject the correction before the entire objective function is evaluated. One possibility is to stop the objective function evaluation once the value from the previous evaluation is exceeded. Another issue concerning robustness is the use of regularization. Currently it is left for the user to diagnose linear dependencies and take the necessary actions. A topic for future work is to build automatic tools for detection and proper treatment of linear dependencies by clever manipulation of the regularization weights. In general, much can be done in designing intelligent parameter estimation software that alerts the user of possible pitfalls and advises possible solutions.

Among the more comprehensive extensions to PARFIT it may be worthwhile to consider extending the class of problems treated to fully implicit DAEs or systems described by PDEs. Moreover, other methods of estimation such as maximum likelihood could be considered. Finally, the experimental design issue could be addressed, the motivation being the dependence of parameter estimation on sufficiently informative experimental data.

Appendices

A

Description of PARFIT

A significant part of the work presented in this thesis has been put into developing the PARFIT program for easy estimation of parameters in systems described by differential-algebraic equations. In this appendix a description of PARFIT is given. A few details are explained regarding the solution of the DAEs and computation of sensitivities. Special attention is given to the sensitivity algorithm implemented in ESDIRK34 to supplement the introduction of ESDIRK34 in Section 3.1.1.1.

PARFIT estimates parameters using a weighted least squares criterion. The minimization problem is:

$$\min_{\boldsymbol{\theta}} \; f(\boldsymbol{\theta}) = \frac{1}{2} \sum_{k=1}^{l_d} \sum_{j=1}^{l_c} \sum_{i=1}^{l_t^k} {w_{ij}^k}^2 \left(y_j(t_i, \boldsymbol{\theta}, \boldsymbol{u}_{ik}) - \tilde{y}_{ij}^k \right)^2 \tag{A.1}$$

in which l_c and l_d denote the number of measured components and the number of data sets, respectively. The kth data set contains measurements of l_c components at l_t^k experimental time instants. $y_j(t_i, \boldsymbol{\theta}, \boldsymbol{u}_{ik})$ denotes the model prediction of component y_j at time t_i with the combination of inputs given by $\boldsymbol{u}_{ik}$, whereas $\tilde{y}_{ij}^k$ denotes the corresponding measurement. The model predictions are obtained by solving the DAEs, which are assumed to be in semi-explicit form:

$$\begin{aligned} \frac{d\boldsymbol{y}}{dt} &= \boldsymbol{f}(t, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\theta}), \qquad & \boldsymbol{y}(t_0) &= \boldsymbol{y}_0 \\ \boldsymbol{0} &= \boldsymbol{g}(t, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\theta}), \qquad & \boldsymbol{z}(t_0) &= \boldsymbol{z}_0 \end{aligned} \tag{A.2}$$

in which $\boldsymbol{y} \in \mathbb{R}^{n_d}$ denotes the vector of differential variables, $\boldsymbol{z} \in \mathbb{R}^{n_a}$ denotes the vector of algebraic variables, and $\boldsymbol{\theta} \in \mathbb{R}^{n_p}$ denotes the vector of parameters to be estimated. $\boldsymbol{f}(\cdot)$ and $\boldsymbol{g}(\cdot)$ are functions mapping $\mathbb{R} \times \mathbb{R}^{n_d} \times \mathbb{R}^{n_a} \times \mathbb{R}^{n_p}$ into $\mathbb{R}^{n_d}$ and $\mathbb{R}^{n_a}$, respectively. PARFIT handles DAEs of index up to one.

A.1 DAE Solution and Sensitivity Computation

PARFIT uses the optimization algorithm LMDER (Moré, 1977) described in Section 5.1 and the DAE solver ESDIRK34 (Kristensen *et al.*, 2004a) described in Section 3.1.1.1. ESDIRK34 implements the staggered direct method for sensitivity computation. Below, an outline is given of the sensitivity implementation.

For simplicity, only differential equations are assumed to be present. In case of algebraic equations, an extra set of algebraic sensitivity equations need to

be solved[1] (the sensitivity extension of ESDIRK34 to DAEs is described in Kristensen *et al.* (2004b)). The ESDIRK34 discretization scheme is given by:

$$\boldsymbol{Y}_i = \boldsymbol{y}_n + h\sum_{j=1}^{i} a_{ij}\boldsymbol{f}(t_n + c_j h, \boldsymbol{Y}_j, \boldsymbol{\theta}) \tag{A.3a}$$

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\sum_{i=1}^{4} b_i\boldsymbol{f}(t_n + c_i h, \boldsymbol{Y}_i, \boldsymbol{\theta}) \tag{A.3b}$$

$$\boldsymbol{e}_{n+1} = h\sum_{i=1}^{4} d_i\boldsymbol{f}(t_n + c_i h, \boldsymbol{Y}_i, \boldsymbol{\theta}) \tag{A.3c}$$

in which $\boldsymbol{Y}_i$ designates the solution at the ith ($i = 1, \ldots, 4$) internal stage of integration step n. The coefficients are defined by the Butcher tableau:

$$\begin{array}{c|cccc} 0 & 0 & & & \\ c_2 & a_{21} & \gamma & & \\ c_3 & a_{31} & a_{32} & \gamma & \\ 1 & b_1 & b_2 & b_3 & \gamma \\ \hline \boldsymbol{y}_{n+1} & b_1 & b_2 & b_3 & \gamma \\ \hline \boldsymbol{e}_{n+1} & d_1 & d_2 & d_3 & d_4 \end{array} \tag{A.4}$$

The following notation is established:

$$\begin{aligned} T_i &= t_n + c_i h \\ \boldsymbol{Y}_i &= \boldsymbol{Y}_i(\boldsymbol{y}_n, \boldsymbol{\theta}), \qquad i = 2, 3, 4 \\ \boldsymbol{F}_i(\boldsymbol{y}_n, \boldsymbol{\theta}) &= \boldsymbol{f}(T_i, \boldsymbol{Y}_i(\boldsymbol{y}_n, \boldsymbol{\theta}), \boldsymbol{\theta}) \end{aligned} \tag{A.5}$$

The computation of sensitivities in ESDIRK34 is separated into a *state* sensitivity computation and a *parameter* sensitivity computation. The state sensitivities represent the sensitivities with respect to initial conditions. The equations describing these sensitivities are simpler compared to the sensitivity equations for parameters in that the initial conditions do not enter explicitly into the right-hand-side functions of the DAE system. When deriving the sensitivity algorithm, the basic idea is to differentiate the discrete equations (A.3) with respect to both the initial conditions and the parameters. First, a single integration step is considered, i.e. the sensitivities are derived at t_{n+1} with respect to the states and parameters at t_n. A simple recursion is then used to update the sensitivities between successive integration steps.

The sensitivities with respect to the initial conditions are derived as:

$$\begin{aligned} \nabla_{\boldsymbol{y}_n}\boldsymbol{y}_{n+1} &= \boldsymbol{I} + h\sum_{i=1}^{4} b_i \nabla_{\boldsymbol{y}_n}\boldsymbol{F}_i(\boldsymbol{y}_n, \boldsymbol{\theta}) \\ &= \boldsymbol{I} + hb_1\nabla_{\boldsymbol{y}}\boldsymbol{f}(t_n, \boldsymbol{y}_n, \boldsymbol{\theta}) + h\sum_{i=2}^{3} b_i \nabla_{\boldsymbol{y}_n}\boldsymbol{Y}_i \nabla_{\boldsymbol{y}}\boldsymbol{f}(T_i, \boldsymbol{Y}_i, \boldsymbol{\theta}) \\ &\quad + h\gamma\nabla_{\boldsymbol{y}_n}\boldsymbol{Y}_4\nabla_{\boldsymbol{y}}\boldsymbol{f}(T_4, \boldsymbol{Y}_4, \boldsymbol{\theta}) \end{aligned}$$

[1] ESDIRK34 handles index one DAEs and the corresponding sensitivity equations.

The ith stage gradient (for $i = 2, 3, 4$) $\nabla_{\boldsymbol{y}_n} \boldsymbol{Y}_i$ is calculated as:

$$\begin{aligned}
\nabla_{\boldsymbol{y}_n} \boldsymbol{Y}_i &= \boldsymbol{I} + h \sum_{j=1}^{i} a_{ij} \nabla_{\boldsymbol{y}_n} \boldsymbol{F}_i(\boldsymbol{y}_n, \boldsymbol{\theta}) \\
&= \boldsymbol{I} + h a_{i1} \nabla_{\boldsymbol{y}} \boldsymbol{f}(t_n, \boldsymbol{y}_n, \boldsymbol{\theta}) + h \sum_{j=2}^{i} a_{ij} \nabla_{\boldsymbol{y}_n} \boldsymbol{Y}_j \nabla_{\boldsymbol{y}} \boldsymbol{f}(T_j, \boldsymbol{Y}_j, \boldsymbol{\theta}) \\
\nabla_{\boldsymbol{y}_n} \boldsymbol{Y}_i &= \left[\boldsymbol{I} + h a_{i1} \nabla_{\boldsymbol{y}} \boldsymbol{f}(t_n, \boldsymbol{y}_n, \boldsymbol{\theta}) + h \sum_{j=2}^{i-1} a_{ij} \nabla_{\boldsymbol{y}_n} \boldsymbol{Y}_j \nabla_{\boldsymbol{y}} \boldsymbol{f}(T_j, \boldsymbol{Y}_j, \boldsymbol{\theta}) \right] \cdot \\
&\quad [\boldsymbol{I} - h\gamma \nabla_{\boldsymbol{y}} \boldsymbol{f}(T_i, \boldsymbol{Y}_i, \boldsymbol{\theta})]^{-1}
\end{aligned}$$

Since $\boldsymbol{Y}_4 = \boldsymbol{y}_{n+1}$, the stage gradient $\nabla_{\boldsymbol{y}_n} \boldsymbol{Y}_4$ is equal to the desired sensitivity. The parameter sensitivities are derived in a similar fashion:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \boldsymbol{y}_{n+1} &= h \sum_{i=1}^{4} b_i \nabla_{\boldsymbol{\theta}} \boldsymbol{F}_i(\boldsymbol{y}_n, \boldsymbol{\theta}) \\
&= h b_1 \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(t_n, \boldsymbol{y}_n, \boldsymbol{\theta}) + h \sum_{i=2}^{3} b_i [\nabla_{\boldsymbol{\theta}} \boldsymbol{Y}_i \nabla_{\boldsymbol{y}} \boldsymbol{f}(T_i, \boldsymbol{Y}_i, \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(T_i, \boldsymbol{Y}_i, \boldsymbol{\theta})] \\
&\quad + h\gamma [\nabla_{\boldsymbol{\theta}} \boldsymbol{Y}_4 \nabla_{\boldsymbol{y}} \boldsymbol{f}(T_4, \boldsymbol{Y}_4, \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(T_4, \boldsymbol{Y}_4, \boldsymbol{\theta})]
\end{aligned}$$

The ith stage gradient $\nabla_{\boldsymbol{\theta}} \boldsymbol{Y}_i$ is calculated as:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \boldsymbol{Y}_i &= h \sum_{j=1}^{i} a_{ij} \nabla_{\boldsymbol{\theta}} \boldsymbol{F}_i(\boldsymbol{y}_n, \boldsymbol{\theta}) \\
&= h a_{i1} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(t_n, \boldsymbol{y}_n, \boldsymbol{\theta}) + h \sum_{j=2}^{i} a_{ij} [\nabla_{\boldsymbol{\theta}} \boldsymbol{Y}_j \nabla_{\boldsymbol{y}} \boldsymbol{f}(T_j, \boldsymbol{Y}_j, \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(T_j, \boldsymbol{Y}_j, \boldsymbol{\theta})] \\
\nabla_{\boldsymbol{\theta}} \boldsymbol{Y}_i &= \Big[h a_{i1} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(t_n, \boldsymbol{y}_n, \boldsymbol{\theta}) + h \sum_{j=2}^{i-1} a_{ij} [\nabla_{\boldsymbol{\theta}} \boldsymbol{Y}_j \nabla_{\boldsymbol{y}} \boldsymbol{f}(T_j, \boldsymbol{Y}_j, \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(T_j, \boldsymbol{Y}_j, \boldsymbol{\theta})] \\
&\quad + h\gamma \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(T_i, \boldsymbol{Y}_i, \boldsymbol{\theta}) \Big] \cdot [\boldsymbol{I} - h\gamma \nabla_{\boldsymbol{y}} \boldsymbol{f}(T_i, \boldsymbol{Y}_i, \boldsymbol{\theta})]^{-1}
\end{aligned}$$

The stage gradient $\nabla_{\boldsymbol{\theta}} \boldsymbol{Y}_4$ is now equal to the desired parameter sensitivity. The matrix $\boldsymbol{I} - h\gamma \nabla_{\boldsymbol{y}} \boldsymbol{f}(T_i, \boldsymbol{Y}_i, \boldsymbol{\theta})$ used in the calculations corresponds to the iteration matrix when $i = 1$. The iteration matrix is already calculated and *LU* factorized by ESDIRK34 in the state integration. To avoid calculating and factorizing the matrix at each internal stage ($i = 2, 3, 4$), the following approximations are used:

$$\nabla_{\boldsymbol{y}} \boldsymbol{f}(T_i, \boldsymbol{Y}_i, \boldsymbol{\theta}) \simeq \nabla_{\boldsymbol{y}} \boldsymbol{f}(t_n, \boldsymbol{y}_n, \boldsymbol{\theta}), \quad i = 2, 3, 4 \tag{A.6}$$

A similar approximation is made for the derivative with respect to the parameters:

$$\nabla_{\boldsymbol{\theta}} \boldsymbol{f}(T_i, \boldsymbol{Y}_i, \boldsymbol{\theta}) \simeq \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(t_n, \boldsymbol{y}_n, \boldsymbol{\theta}), \quad i = 2, 3, 4 \tag{A.7}$$

Thus, the sensitivities calculated by ESDIRK34 are approximations compared to the integration scheme used for the state integration. The strong advantage, however, is that no additional evaluations of the Jacobian are required for the sensitivity integration, which allows for a very efficient implementation.

The recursion used to update the sensitivities between successive integration steps is readily derived. Observing that $\boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+2}(\boldsymbol{y}_{n+1}, \boldsymbol{\theta})$ and differentiating with respect to $\boldsymbol{y}_n$ and $\boldsymbol{\theta}$, gives:

$$\begin{aligned} \nabla_{\boldsymbol{y}_n} \boldsymbol{y}_{n+2} &= \nabla_{\boldsymbol{y}_n} \boldsymbol{y}_{n+1} \nabla_{\boldsymbol{y}_{n+1}} \boldsymbol{y}_{n+2} \\ \nabla_{\boldsymbol{\theta}} \boldsymbol{y}_{n+2} &= \nabla_{\boldsymbol{\theta}} \boldsymbol{y}_{n+1} \nabla_{\boldsymbol{y}_{n+1}} \boldsymbol{y}_{n+2} + \nabla_{\boldsymbol{\theta}} \boldsymbol{y}_{n+2} \end{aligned} \tag{A.8}$$

These equations establish the updating between steps. The state sensitivities are initialized with the identity matrix, whereas the parameter sensitivities are assumed initially to be zero corresponding to no parameter dependence of the initial conditions.

A.2 Algorithmic Outline

The different features of PARFIT were summarized in Chapter 9. This section provides a brief algorithmic overview. With the tolerance selection mechanism discussed in Section 6.1.1 the PARFIT algorithm consists mainly of a single loop, in which LMDER is called with a successively tightening tolerance. The algorithm is outlined below:

Require: initial guess $\boldsymbol{\theta_0}$
  Check inputs
  Provide default settings if not provided by the user
  Load data
  for $i = 1$ **to** *number_of_tolerance_intervals* **do**
    Compute optimization tolerances
    Compute ESDIRK34 tolerances
    Call LMDER
  end for
  Compute the covariance and correlation matrices
  Print results
Return: optimal solution $\boldsymbol{\theta^*}$

The objective function subroutine used by LMDER requires no modifications from the user, provided that the states are measured directly (no measurement equation) and that the l_c components regressed correspond to the first l_c components of the state vector. An outline of the objective function subroutine is given below:

Require: current iterate $\boldsymbol{\theta_k}$, array of data, weight factors and inputs
  Set initial values for the DAEs
  for j = 1 **to** *number_of_data_sets* **do**
    Initialize sensitivity arrays
    Compute initial step length for ESDIRK34
    for $i = 1$ **to** *number_of_experimental_time_values* **do**
      Set t and t_{final}
      Integrate DAEs + sensitivity equations from t to t_{final} using ESDIRK34
      Compute residuals
      Compute Jacobian of residuals
    end for
  end for
  if *regularization* **then**
    Compute contribution to residual vector
    Compute contribution to Jacobian
  end if
Return: vector of residuals $\boldsymbol{r}(\boldsymbol{\theta_k})$, Jacobian of residuals $\boldsymbol{J}(\boldsymbol{\theta_k})$

At each sampling instant the integrator is stopped to provide the model output in order to form the residual. Frequent sampling of the system requires frequent interruption of the integrator. Being a one-step method, ESDIRK34 does not suffer particularly from these restarts. Moreover, the last normal stepsize[2] computed in the current integration interval is used as the first stepsize in the next interval, which greatly reduces the number of failed steps.

A.3 Documentation

This section serves as documentation for PARFIT. The interconnections between the subroutines are illustrated in Figure A.1. Subroutines in grey must be supplied by the user. The DFAULT subroutine provides default settings for the optimization. DFAULT may, optionally, be called by the user before calling PARFIT, if changes to the default settings are to be made. PARFIT calls LMDER, which in each iteration calls OBJFUN to evaluate the residual functions and the Jacobian of the residual functions. For each data set OBJFUN calls BUILD_OBJ that provides the contribution to the residual vector and the Jacobian. ESDIRK34 is used to integrate the model equations and to calculate derivative information. The user must provide the FUN and JAC subroutines computing the right-hand-side functions of the DAEs and the Jacobian of the DAE system, respectively. JAC must also return the partial derivatives with respect to the parameters being estimated. An option exists for approximating the Jacobian of the DAEs and the partial derivatives with respect to the parameters by finite difference quotients. In this case JAC is just a dummy subroutine.

Before termination, PARFIT computes the covariance matrix for the parameters. The Cholesky factorization of the final Hessian approximation is available

[2]That is, the second to last in the stepsize sequence, since the last step is computed as $h = t_{final} - t$.

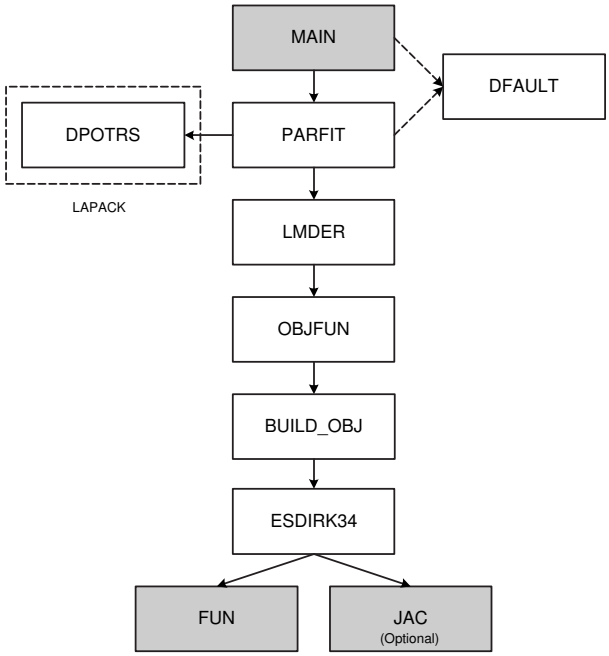


Figure A.1. Hierarchy of subroutines. The user must supply the MAIN program as well as the FUN and JAC (optional) subroutines computing the right-hand-side functions of the DAEs and the Jacobian of the DAE system, respectively.

from LMDER. The LAPACK subroutine DPOTRS is used to invert the Hessian.

The user must provide data files containing measurements, weight factors and inputs. Each data file must be organized as follows:

$$\begin{array}{cccccccccc} t_1 & \tilde{y}_1(t_1) & w_{11} & \tilde{y}_2(t_1) & w_{12} & \cdots & \tilde{y}_{l_c}(t_1) & w_{1l_c} & u_{11} & \cdots & u_{1l_u} \\ t_2 & \tilde{y}_1(t_2) & w_{21} & \tilde{y}_2(t_2) & w_{22} & \cdots & \tilde{y}_{l_c}(t_2) & w_{2l_c} & u_{21} & \cdots & u_{2l_u} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \ldots & \ldots & \cdots & \vdots \\ t_{l_t} & \tilde{y}_1(t_{l_t}) & w_{l_t1} & \tilde{y}_2(t_{l_t}) & w_{l_t2} & \cdots & \tilde{y}_{l_c}(t_{l_t}) & w_{l_tl_c} & u_{l_t1} & \cdots & u_{l_tl_u} \end{array}$$

Thus, the first column contains the values of the independent variable. The next $2 \cdot l_c$ columns contain measurements and weights, whereas the last l_u columns contain inputs. The inputs are optional. Each data file must contain l_u different inputs along with measurements of l_c components. The number of sampling instants, l_t, can vary from data set to data set.

PARFIT uses a derived data type to store the data. In the MAIN program the user must specify an array of type `EXPERIMENT` according to the number of data sets, and subsequently specify the entries `FILE_NAME` and `LT` in each `EXPERIMENT`. For example, if two data sets are available containing measurements at 12 and 16 sampling instants, respectively, then the following lines must be included in the MAIN program:

```
TYPE(EXPERIMENT) EX(2)
...
EX(1)%FILE_NAME = 'DATA_SET_1.TXT'
EX(1)%LT = 12
EX(2)%FILE_NAME = 'DATA_SET_2.TXT'
EX(2)%LT = 16
```

The maximum length of the `FILE_NAME` character string is 30. The `EX` data structure is passed to PARFIT in the argument list.

Below, a description is given of the individual subroutines requiring attention from the user. LMDER and ESDIRK34 are not described, but a description can be found in the header of each subroutine. For each subroutine described, the syntax when called is given, together with a short description of the subroutine functionality and the arguments (names, intent and types) involved when calling the subroutine.

PARFIT

Syntax : `CALL PARFIT(N,X,YO,EX,FUN,JAC,COV,IW,W,RPAR,IPAR)`

Description : PARFIT estimates parameters and initial conditions in systems described by index one differential-algebraic equations using a weighted least squares criterion.

Dependencies : LMDER, DFAULT, DPOTRS (LAPACK)

Arguments :

- `N` (input) INTEGER. Number of parameters to estimate. If initial conditions are estimated, `N` should include the number of initial conditions.
- `X` (input/output) DOUBLE PRECISION ARRAY, dimension (`N`). On entry `X` holds the initial guess for the parameters, and on exit `X` holds the final parameter estimates.
- `YO` (input) DOUBLE PRECISION ARRAY, dimension (`NN`,`LD`). Initial values for dependent variables in the DAE system (in case of algebraic equations, use the switch `IW(15)` to specify wether the initial values are consistent). The initial conditions corresponding to each data set are stored in the columns of `YO`. If the initial conditions are to be estimated from data, `YO` will hold the estimated values on output. Initial condition estimation is only an option for pure ODE systems. If estimated, a starting guess for the initial conditions must be provided in `X` following the other parameters. That is, the first `N-NN` elements of `X` are reserved for "normal" parameters, whereas the last `NN` elements are used for the initial conditions.

`EX` (input) Derived data structure of type 'EXPERIMENT', dimension (LD). The user must specify the name and length of each data file. The following lines should be included in the calling program:

```
TYPE(EXPERIMENT) EX(ld)
...
EX(1)%FILE_NAME  = '<< NAME_OF_DATA_FILE_1 >>'
EX(1)%LT  = LT_1
...
EX(LD)%FILE_NAME = '<< NAME_OF_DATA_FILE_LD >>'
EX(LD)%LT = LT_LD
```

`FUN` (input) Name of subroutine computing the right-hand-side functions of the DAE system. See specification of `FUN`.

`JAC` (input) Name of subroutine computing the Jacobian of the DAE system as well as the partial derivatives with respect to the parameters being estimated. See specification of `JAC`. A dummy subroutine is supplied, if the option for numerical difference derivatives is used.

`COV` (output) DOUBLE PRECISION ARRAY, dimension (N,N). Estimated covariance matrix for the parameters.

`IW` (input/output) INTEGER ARRAY, dimension (NN**2+N+30). Integer working space. The first 30 elements of `IW` are used to specify settings for PARFIT and to store computational statistics. The user must specify `IW(1)-IW(6)` and optionally `IW(7)-IW(15)`:

`IW(1)` (=0) Default settings are used in the parameter estimation. PARFIT calls DFAULT(IW,W) that specifies defaults values.
(=1) User has made separate call to DFAULT and subsequently changed some of the values in `IW` or `W`.

`IW(2)` (=NN) Dimension of the DAE system.

`IW(3)` (=ND) Number of differential equations. If pure ODE system : `ND = NN`.

`IW(4)` (=LU) Number of inputs. Each file must contain the same number of inputs. If no inputs : `LU = 0`.

`IW(5)` (`=LC`) Number of components regressed. It is assumed in PARFIT that the first `LC` components of the state vector are measured. That is, the first column of measurements in the datafile contains measurements of the first component of the state vector

`IW(6)` (`=LD`) Number of data sets. The required structure is specified above.

The remaining entries in `IW` can be altered after call to DFAULT. The default values are marked with an `'*'`:

`IW(7)` (`=1*`) Internal scaling of parameters.
(`=2`) Scale factors are provided in elements `W(31+N)-W(30+2*N)`.

`IW(8)` (`=0`) No computation of the covariance and correlation matrices.
(`=1*`) The covariance and correlation matrices are computed.

`IW(9)` (`=500*`) Maximum number of objective function evaluations.

`IW(10)` (`=0`) No printing of results.
(`=1*`) Results and computational statistics are printed.

`IW(11)` (`=2*`) Number of tolerance subintervals.

`IW(12)` (`=0*`) Initial conditions are not estimated.
(`=1`) Initial conditions are estimated along with the problem parameters (only for ODE systems).

`IW(13)` (`=0`) Numerical approximation of the DAE system Jacobian and the partial derivatives with respect to parameters.
(`=1*`) Analytic Jacobian and partial derivatives with respect to parameters are provided in subroutine JAC.

`IW(14)` (`=0*`) Initial conditions for DAEs are consistent.
(`=1`) An internal consistency calculation is performed by ESDIRK34 in the first call in each objective function evaluation, and whenever a discontinuity is present (due to a discontinuous input profile).

`IW(15)` (`=0`*) No regularization is used.
(`=1`) Regularization weights are provided in `W(31)-W(30+N)`. The regularized optimization problem is:

```
min  sum { 1/2 * W_i**2 * R_i**2 } +
 P    i
sum { 1/2 * WREG_j * ( (X_j - X0_j)/X0_j,ref )**2 }
 j
```

in which `WREG_j` denotes the regularization weights, whereas `X0_j,ref` denotes reference values for `X_j` chosen as the order of magnitude for the individual parameter. By default `X0_j,ref = X0_j` meaning that the `X0_j,ref`'s need not be specified.
(`=2`) As for `IW(15)=1`, but with user specified reference values. The `X0_j,ref`'s must be stored in `W(31+2*N)-W(30+3*N)` before calling PARFIT.

On exit, the following statistics are stored in `IW`:

`IW(21)` Number of objective function evaluations.

`IW(22)` Number of Jacobian evaluations (of residual functions).

`IW(23)` Number of integration steps (accepted + rejected).

`IW(24)` Number of rejected steps.

`IW(25)` Number of times diverged in Newton iteration.

`IW(26)` Number of rhs function evaluations.

`IW(27)` Number of Jacobian evaluations (of DAE system).

`IW(28)` Number of *LU* factorizations.

`IW(29)` Number of back substitutions.

`W` (input/output) DOUBLE PRECISION ARRAY, dimension (`6*NN**2+11*NN+7*N*NN+24+8*N+N*M+2*M`).
Work space for LMDER and ESDIRK34. User may optionally specify `W(1)-W(5)` (else specified by DFAULT):

`W(1)` (`1.D-4*`) Tolerance related to relative error in the objective function value (see Eq. (5.2)).

`W(2)` (`1.D-4*`) Tolerance related to relative tolerance in the solution (see Eq. (5.3)).

`W(3)` (`1.D-4*`) Orthogonality between residual vector and columns of Jacobian (see Eq. (5.4)).

`W(4)` (`1*`) Factor related to initial step bound in optimizer. If `W(4)` is decreased, a more conservative step is taken initially.

`RPAR`, `IPAR` (input) Real and integer parameter arrays, which can be used for communication between the calling program and the FUN and JAC subroutines. The first `N` elements of `RPAR` are used to store those parameters being estimated and therefore they need not be set before calling PARFIT. The inputs from the data sets are passed in `RPAR(N+1)-RPAR(N+LU)`. (The initial conditions are not stored in `RPAR`, meaning that the inputs are stored following the problem parameters). Any parameters to be used in FUN or JAC, which are not estimated, must be stored after `RPAR(N+LU)`. `RPAR` and `IPAR` must be declared in the calling program with the appropriate sizes.

FUN

Syntax : `CALL FUN(NN,ND,T,Y,WOUT,F,RPAR,IPAR)`

Description : Subroutine for computing the right-hand-side functions of the DAE system.

Arguments :

`NN` (input) INTEGER. Dimension of the DAE system.

`ND` (input) INTEGER. Number of differential equations.

`T` (input) DOUBLE PRECISION. Current value of independent variable.

`Y` (input) DOUBLE PRECISION ARRAY, dimension (`NN`). Current value(s) of dependent variable(s).

`WOUT` (output) DOUBLE PRECISION ARRAY. Optional array for additional output. Must be declared with the desired length in the calling program.

`F` (output) DOUBLE PRECISION ARRAY, dimension (`NN`). The first `ND` elements of `F` are the right-hand-side function values of the differential equations. The last `NN-ND` elements are the residuals of the algebraic equations.

RPAR, IPAR — See description above.

JAC

Syntax : `CALL JAC(NN,ND,N,T,Y,DFUN,DFDP,RPAR,IPAR)`

Description : Subroutine for computing Jacobian of the DAE system as well as the partial derivatives with respect to the parameters being estimated.

Arguments :

`NN` (input) INTEGER. Dimension of the DAE system.

`ND` (input) INTEGER. Number of differential equations.

`N` (input) INTEGER. Number of parameters being estimated.

`T` (input) DOUBLE PRECISION. Current value of independent variable.

`Y` (input) DOUBLE PRECISION ARRAY, dimension (`NN`). Current value(s) of dependent variable(s).

`DFUN` (output) DOUBLE PRECISION ARRAY, dimension (`NN,NN`). Jacobian of DAE system. The structure of the Jacobian is:

```
       |----- ND -----|---- NA -----|
        _                           _  ___
       |                             |  |
       |   dF / dY        dG / dY    |  ND
       |                             | _|_
J  =   |                             |  |
       |   dF / dZ        dG / dZ    |  NA
       |_                           _| _|_
```

in which `NA=NN-ND`. `F` and `G` denote the vectors of functions describing the differential and algebraic equations, respectively, whereas `Y` and `Z` denote differential and algebraic variables, respectively.

`DFDP` (output) DOUBLE PRECISION ARRAY, dimension (`N,NN`). Partial derivatives with respect to the parameters being estimated.

`RPAR`, `IPAR` — See description above.

DFAULT

Syntax	:	`CALL DFAULT(IW,W)`
Description	:	Subroutine for specifying default settings to be used in PARFIT. If not called by the user, DFAULT is called by PARFIT. The value of `IW(1)` determines wether PARFIT will call DFAULT.
Arguments	:	`IW` (output) INTEGER ARRAY. See description above.
		`W` (output) DOUBLE PRECISION ARRAY. See description above.

A.3.1 List of Subroutines

	Parameter estimation	**Optimization**	**Differential equation solution**
Files	parfit.for	lmder.for lmderconsist.for	esdirk34dae.for
Subroutines	parfit objfun build_obj dfault	lmder lmderconsist	esdirk34 esdirk34core sens hstart consist fcn_con
Auxiliary files	dpotrs.for	lmder_aux.for	esdirk34_aux.for
Subroutines/ functions	dpotrs	qrfac qrsolv lmpar enorm dpmpar	dgetrf dgetrs dlaswp dgetf2 dgemm dtrsm dger dscal dnrm2 idamax lsame xerbla

Table A.5. Complete list of files and subroutines/functions in the PARFIT project. Subroutines developed as part of the work presented in this thesis are marked in grey. The subroutines marked in blue are MINPACK routines, which have been modified, whereas the subroutines marked in red are routines for the ESDIRK34 sensitivity tailored differential equation solver presented in Kristensen *et al.* (2004a). The remaining subroutines are LAPACK/BLAS/MINPACK library routines.

The PARFIT program was developed using the Compaq Visual Fortran 6.1.A compiler. A complete list of the files and subroutines/functions in the PARFIT project is given Table A.5. All source files are included on the CD-ROM at the back of this thesis along with driver routines for the gas-oil cracking problem and the Dow Chemicals problem. The CD contains a Visual Fortran workspace with two projects, one for each test problem.

A.3.2 Code Listings

The source code for the key subroutines PARFIT, OBJFUN, BUILD_OBJ and DFAULT is listed below:

```
      MODULE MODUL
***************************************************************************
***                                                                     ***
***            MODULE WITH DERIVED DATA TYPE USED TO STORE DATA         ***
***                                                                     ***
***************************************************************************
      TYPE EXPERIMENT
          CHARACTER* 30 FILE_NAME
          INTEGER LT
          DOUBLE PRECISION, DIMENSION(:), POINTER :: PTRT
          DOUBLE PRECISION, DIMENSION(:,:), POINTER :: PTRY
          DOUBLE PRECISION, DIMENSION(:,:), POINTER :: PTRW
          DOUBLE PRECISION, DIMENSION(:,:), POINTER :: PTRU
      END TYPE
      PUBLIC EXPERIMENT
      END MODULE MODUL

      SUBROUTINE PARFIT(N,X,Y0,EX,FUN,JAC,COV,IW,W,RPAR,IPAR)
***************************************************************************
***                                                                     ***
***                    PARAMETER ESTIMATION SUBROUTINE                  ***
***                                                                     ***
***************************************************************************
      USE MODUL
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

      INTEGER IW(*),IPAR(*),IODE(20),ISENSPAR(N)
      DOUBLE PRECISION X(N),Y0(IW(2),*)
      DOUBLE PRECISION W(*),COV(N,*),RPAR(*)
      LOGICAL FIRST
      TYPE(EXPERIMENT) EX(IW(6))
      EXTERNAL FUN,JAC,OBJFUN,LMDER
C----- LOCAL VARIABLES:
      DOUBLE PRECISION CORR(N,N),X0(N)

C----- PROVIDE DEFAULT VALUES:
      IF (IW(1) .EQ. 0) CALL DFAULT(IW,W)
      NN   = IW(2)
      ND   = IW(3)
      LU   = IW(4)
      LC   = IW(5)
      LD   = IW(6)
      IC   = IW(12)
      IREG = IW(15)

C----- SPECIFY TOLERANCES FOR LMDER AND ESDIRK34:
      NTOL  = IW(11)
      FTOLL = W(1)
      XTOLL = W(2)
      GTOLL = W(3)

C----- CHECK INPUTS:
      IF ((IC .EQ. 1) .AND. (NN .NE. ND)) THEN
          WRITE(*,'(1A,X,I2)') 'ERROR : IW(12) =',IC
          WRITE(*,'(1A,X)') 'INITIAL CONDITION ESTIMATION IN SYSTEMS
     & WITH ALGEBRAIC EQUATIONS IS NOT SUPPORTED'
          RETURN
      END IF
      IF ((IREG .NE. 0) .AND. (IREG .NE. 1) .AND. (IREG .NE. 2)) THEN
          WRITE(*,'(1X,A,I2)') 'INVALID VALUE : IW(15) = ',IW(15)
          RETURN
      END IF
      M = 0
      DO I = 1,LD
          M = M + EX(I)%LT
      END DO
      M = LC*M
      IF (IREG .GE. 1) M = M+N

```

```
C-----SETTINGS FOR DAE SOLVER:
      DO I = 1,20
         IODE(I) = 0
      END DO
      IF (IW(13) .EQ. 0) THEN
         IODE(2) = 0    ! (=0) : NUMERICAL JACOBIAN
         DO I = 1,N-IC*LD*ND
            ISENSPAR(I) = I
         END DO
      ELSE
         IODE(2) = 1    ! (=1) : ANALYTIC JACOBIAN
      END IF
      IODE(4) = 1              ! (=1) : SENSITIVITIES ARE CALCULATED
      IODE(5) = N-IC*LD*ND     !       : NUMBER OF SENSITIVITY PARAMETERS

C-----LOAD DATA:
      DO I = 1,LD
         OPEN(UNIT=20,FILE=EX(I)%FILE_NAME)
         LT = EX(I)%LT
         ALLOCATE (EX(I)%PTRT(LT))
         ALLOCATE (EX(I)%PTRY(LT,LC))
         ALLOCATE (EX(I)%PTRW(LT,LC))
         IF (LU .GT. 0) THEN
            ALLOCATE (EX(I)%PTRU(LT,LU))
         END IF
         DO J = 1,LT
            READ(20,*) EX(I)%PTRT(J),
     &                 (EX(I)%PTRY(J,K), EX(I)%PTRW(J,K), K=1,LC),
     &                 (EX(I)%PTRU(J,K), K=1,LU)
         END DO
         CLOSE(20)
      END DO

C-----STORE INITIAL PARAMETER VECTOR:
      DO I = 1,N
         X0(I) = X(I)
      END DO

C-----PREPARE CALL TO OPTIMIZER:
      N0  = 1                  !
      N1  = N0+30              ! wreg (LMDER)
      NN1 = N1+N               ! diag (LMDER)
      NN2 = NN1+N              ! xref (LMDER)
      N2  = NN2+N              ! fvec (LMDER)
      N3  = N2+M               ! fjac (LMDER)
      N4  = N3+N*M             ! qft  (LMDER)
      N5  = N4+N               ! wa1  (LMDER)
      N6  = N5+N               ! wa2  (LMDER)
      N7  = N6+N               ! wa3  (LMDER)
      N8  = N7+N               ! wa4  (LMDER)
      N9  = N8+M               ! A    (ESDIRK34)
      N10 = N9+NN**2           ! B    (ESDIRK34)
      N11 = N10+N*NN           ! WORK (ESDIRK34)
      CALL CPU_TIME(T2)
      DO I = 1,NTOL
         IF (IW(10) .EQ. 1) THEN
            WRITE(*,'(1X,A,I1)') 'INTERVAL NO. ',I
         END IF
         FTOL = FTOLL**(DBLE(I)/DBLE(NTOL))
         XTOL = XTOLL**(DBLE(I)/DBLE(NTOL))
         GTOL = GTOLL**(DBLE(I)/DBLE(NTOL))
         RTOL = 1.D-4*MIN(FTOL,MIN(XTOL,GTOL))
         ATOL = RTOL
         CALL LMDER(OBJFUN,FUN,JAC,M,N,X,X0,Y0,EX,W(N2),W(N3),M,FTOL,
     &        XTOL,GTOL,IW(9),W(NN1),W(N1),IW(7),W(4),1,IFLAG,NFEV,NJEV,
     &        IW(N1),W(N4),W(N5),W(N6),W(N7),W(N8),W(N9),W(N10),W(N11),
     &        IW(NN1),IW(N0),W(NN2),IODE,ISENSPAR,RTOL,ATOL,RPAR,IPAR)
         FIRST    = .FALSE.
         IW(21) = IW(21) + NFEV
         IW(22) = IW(22) + NJEV
         IF (IFLAG .GT. 4) THEN
            WRITE(*,'(1X,A,I1,1X,A)') 'ERROR CODE: ',IFLAG,
     &         '- SEE HEADER OF LMDER FOR EXPLANATION OF ERROR CODE'
            RETURN
         END IF
      END DO
      CALL CPU_TIME(T1)
      DO I = 1,N-ND*LD*IC
         RPAR(I) = X(I)
      END DO
      IF (IC .EQ. 1) THEN
      NTMP = N-LD*LC
      DO J = 1,LD
         DO I = 1,ND
            Y0(I,J) = X(NTMP+(J-1)*ND+I)
         END DO
      END DO
      END IF

C-----COMPUTE COVARIANCE AND CORRELATION MATRIX:
      IF (IW(8) .EQ. 1) THEN
         DO I = 1,N
            DO J = 1,N
```

```
                    COV(J,I) = 0.D0
165                 IF (I .EQ. J) COV(I,I) = 1.D0
                ENDDO
                ENDDO
                CALL DPOTRS('U',N,N,W(N3),M,COV,N,INFOO)
                SCALE = ENORM(M,W(N2))**2/(M-N)
170             DO I = 1,N
                    DO J = 1,N
                        COV(I,J) = COV(I,J)*SCALE
                        CORR(I,J) = COV(I,J)
                    ENDDO
175             ENDDO
                DO I = 1,N
                    SQ = SQRT(COV(I,I))
                    DO J = 1,N
                        CORR(I,J) = CORR(I,J)/SQ
180                 CORR(J,I) = CORR(J,I)/SQ
                    ENDDO
                ENDDO
            ENDIF

185 C-----PRINT STATISTICS:
            IF (IW(10) .EQ. 1) THEN
                WRITE(*,*)
                WRITE(*,*) 'SUMMARY OF RESULTS FROM PARFIT     : '
                WRITE(*,*)
190             WRITE(*,21)'TOTAL COMPUTATION TIME             : ',T1-T2,'sec.'

                WRITE(*,*)
                WRITE(*,*) 'EXECUTION STATUS                   : ',IFLAG
                WRITE(*,22)'FINAL OBJECTIVE FUNCTION VALUE     : ',
195         &                   ENORM(M,W(N2))**2/2.D0
                IF (IREG .EQ. 1) THEN
                    WRITE(*,22)'      DUE TO SQUARED RESIDUALS     : ',
            &                       ENORM(M-N,W(N2))**2/2.D0
                    WRITE(*,22)'      DUE TO REGULARIZATION        : ',
200         &                       ENORM(N,W(N2+M-N))**2/2.D0
                ENDIF
                WRITE(*,*) 'FINAL PARAMETERS                   : '
                WRITE(*,*)
                DO I = 1,N
205                 WRITE(*,'(1X,A,I2,1X,A,E13.6)')
            &        'k',I,' = ',X(I)
                ENDDO
                WRITE(*,*)
                WRITE(*,*) 'COVARIANCE MATRIX                  :'
210             DO I = 1,N
                    WRITE(*,'(1X,A,I2,100E12.4)') 'row',I,(COV(I,J), J=1,I)
                ENDDO
                WRITE(*,*)
                WRITE(*,*) 'CORRELATION MATRIX                 :'
215             DO I = 1,N
                    WRITE(*,'(1X,A,I2,100E12.4)') 'row',I,(CORR(I,J), J=1,I)
                ENDDO

                WRITE(*,*)
220             WRITE(*,*) 'OBJECTIVE FUNCTION EVALUATIONS     : ',IW(21)
                WRITE(*,*) 'JACOBIAN EVALUATIONS (LMDER)       : ',IW(22)
                WRITE(*,*) 'TOTAL NUMBER OF STEPS              : ',IW(23)
                WRITE(*,*) 'NUMBER OF REJECTED STEPS           : ',IW(24)
                WRITE(*,*) 'NUMBER OF TIMES DIVERGED           : ',IW(25)
225             WRITE(*,*) 'NUMBER OF FUNCTION EVALUATIONS     : ',IW(26)
                WRITE(*,*) 'NUMBER OF JACOBIAN EVALUATIONS     : ',IW(27)
                WRITE(*,*) 'NUMBER OF LU FACTORIZATIONS        : ',IW(28)
                WRITE(*,*) 'NUMBER OF BACK SUBSTITUTIONS       : ',IW(29)
    21        FORMAT(1X,A,F12.4,1X,A)
230 22        FORMAT(1X,A,E14.6)
            ENDIF
            END ! SUBROUTINE PARFIT

            SUBROUTINE OBJFUN(M,LU,LC,LD,N,NN,ND,X,X0,Y0,EX,WREG,FVEC,FJAC,
235         &                 LDJAC,IFLAG,FUN,JAC,WORK,IWORK,A,B,ISTAT,XREF,
            &                 IODE,ISENSPAR,RTOL,ATOL,RPAR,IPAR)
    ************************************************************************
    ***                                                                  ***
    ***                   OBJECTIVE FUNCTION SUBROUTINE                  ***
240 ***                                                                  ***
    ************************************************************************
            USE MODUL

            IMPLICIT DOUBLE PRECISION (A-H,O-Z)
245
            INTEGER IWORK(NN**2),IODE(20),ISTAT(30),IPAR(*),ISENSPAR(*)
            DOUBLE PRECISION X(N),X0(N),FVEC(M),FJAC(LDJAC,N),WREG(N)
            DOUBLE PRECISION WORK(6*NN**2+11*NN+4 + 5*N*NN),XREF(N)
            DOUBLE PRECISION Y0(NN,*),A(NN,NN),B(N-ISTAT(12)*ND*LD,NN),RPAR(*)
250         TYPE(EXPERIMENT) EX(LD)
    C-----LOCAL VARIABLES:
            DOUBLE PRECISION Y0TEMP(NN,LD),Y0CON(NN)
            EXTERNAL FUN,JAC

255         IC   = ISTAT(12) ! SWITCH FOR INITIAL CONDITION ESTIMATION
            ICON = ISTAT(14) ! SWITCH FOR CONSISTENCY CALCULATION
```

```
      IREG = ISTAT(15) ! SWITCH FOR USE OF REGULARIZATION

      DO I = 1,N-ND*LD*IC
260        RPAR(I) = X(I)
      END DO

      IF ((IC .EQ. 1) .AND. (IFLAG .EQ. 2)) THEN
         NTMP = N-ND*LD
         DO J = 1,LD
            DO I = 1,ND
               Y0TEMP(I,J) = X(NTMP+(J-1)*ND+I)
            END DO
         END DO
         DO J = NTMP+1,N
            DO I = 1,M
               FJAC(I,J) = 0.D0
            END DO
         END DO
      ELSE
         DO J = 1,LD
            DO I = 1,NN
               Y0TEMP(I,J) = Y0(I,J)
            END DO
         END DO
      END IF
      IF (IREG .EQ. 1) THEN
         DO I = 1,N
            XREF(I) = X0(I)
         END DO
      END IF
      IENTRY = 0
      IF (IFLAG .EQ. 2) THEN ! CALCULATE RESIDUALS AND JACOBIAN AT CURRENT X
         DO I = 1,LD
            CALL BUILD_OBJ(M,N,LD,I,IENTRY,LC,LU,IC,NN,ND,X,X0,
     &           Y0TEMP(1,I),FVEC,FJAC,LDJAC,FUN,JAC,WORK,IWORK,
     &           A,B,ISTAT,IODE,ISENSPAR,RTOL,ATOL,RPAR,IPAR,
     &           EX(I)%PTRT,EX(I)%PTRY,EX(I)%PTRW,EX(I)%PTRU,EX(I)%LT,
     &           WREG,XREF,Y0CON)
            IENTRY = IENTRY + LC*EX(I)%LT
            IF ((ND .LT. NN) .AND. (ICON .EQ. 1)) THEN
               DO J = ND+1,NN ! SAVE CONSISTENT INITIAL VALUES
                  Y0(J,I) = Y0CON(J)
               END DO
            END IF
         END DO
         IF (IREG .GE. 1) THEN
            CALL BUILD_OBJ(M,N,LD,0,IENTRY,LC,LU,IC,NN,ND,X,X0,
     &           Y0TEMP,FVEC,FJAC,LDJAC,FUN,JAC,WORK,IWORK,
     &           A,B,ISTAT,IODE,ISENSPAR,RTOL,ATOL,RPAR,IPAR,
     &           EX(1)%PTRT,EX(1)%PTRY,EX(1)%PTRW,EX(1)%PTRU,EX(1)%LT,
     &           WREG,XREF,Y0CON)
         END IF
      ELSEIF (IFLAG .EQ. 0) THEN
         IF (ISTAT(10) .EQ. 1) THEN
            WRITE(*,'(1X,A,4E12.5)')'F( X )   = ',ENORM(M,FVEC)**2/2.D0
        END IF
      END IF

      RETURN
      END ! SUBROUTINE OBJFUN

      SUBROUTINE BUILD_OBJ(M,N,LD,ILD,IENTRY,LC,LU,IC,NN,ND,X,X0,Y0,
     &                     FVEC,FJAC,LDJAC,FUN,JAC,WORK,IWORK,A,B,
     &                     ISTAT,IODE,ISENSPAR,RTOL,ATOL,RPAR,IPAR,
     &                     T,Y,W,U,LT,WREG,XREF,Y0CON)
*****************************************************************************
***                                                                       ***
***          AUXILIARY SUBROUTINE USED BY OBJFUN FOR BUILDING             ***
***          THE OBJECTIVE FUNCTION + GRADIENT FOR ONE DATA SET.          ***
***                                                                       ***
*****************************************************************************
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

      DOUBLE PRECISION X(N),X0(N),FVEC(M),FJAC(LDJAC,N),WREG(N)
      DOUBLE PRECISION WORK(6*NN**2+11*NN+4 + 5*N*NN),XREF(N)
      DOUBLE PRECISION Y0(NN),A(NN,NN),B(N-IC*LD*ND,NN),RPAR(*)
      DOUBLE PRECISION Y0CON(NN)
      DOUBLE PRECISION T(LT),Y(LT,LC),W(LT,LC),U(LT,*)
      INTEGER IWORK(NN**2),IODE(20),ISTAT(30),IPAR(*),ISENSPAR(*)
      EXTERNAL FUN,JAC

      LWORK  = 6*NN**2+11*NN+4 + 5*N*NN
      LIWORK = NN**2
      NTMP   = LD*ND*IC
      NTMP2 = N+(ILD-1-LD)*ND
      CALL HSTART(NN,ND,FUN,T,Y0,H,WOUT,RTOL,ATOL,0,RPAR,IPAR)

C-----RESET A AND B AND FJAC:
      DO J = 1,ND
         DO I = 1,ND
            A(I,J) = 0.D0
            IF (I .EQ. J) A(I,I) = 1.D0
         END DO
```

```
      END DO
      DO J = 1,ND
         DO I = 1,N-NTMP
            B(I,J) = 0.D0
         END DO
      END DO
      IF ((ISTAT(14) .EQ. 1) .AND. (ND .LT. NN)) THEN
         IODE(6) = 1 ! PERFORM INTERNAL CONSISTENCY CALCULATION IN DAEs
      END IF
C----- BUILD RESIDUAL FUNCTIONS + JACOBIAN:
      IF (ILD .EQ. 0) THEN ! CONTRIBUTION FROM REGULARIZATION
         DO I = 1,N
            JJ = IENTRY+I
            FVEC(JJ) = WREG(I)*(X(I) - X0(I))/XREF(I)
         END DO
         DO J = 1,N
            DO I = 1,N
               II = IENTRY+I
               FJAC(II,J) = 0.D0
               IF (I .EQ. J) FJAC(II,J) = WREG(J)/XREF(J)
            END DO
         END DO
      ELSE ! CONTRIBUTION FROM RESIDUALS
      DO I = 1,LT
         IF (I .EQ. 1) THEN ! FIRST MEASUREMENT
            DO J = 1,LC
               JJ = I+LT*(J-1)+IENTRY
               FVEC(JJ)=W(I,J)*(Y0(J)-Y(I,J))
            END DO
            DO J = 1,LC ! PROVIDE SENSITIVITIES AT INITIAL T
               JJ = I+LT*(J-1)+IENTRY
               DO K = 1,N-NTMP
                  FJAC(JJ,K) = 0.D0
               END DO
               IF (IC .EQ. 1) THEN
               DO K = 1,ND
                  IF (K .EQ. J) THEN
                     FJAC(JJ,NTMP2+K) = W(I,J)
                  ELSE
                     FJAC(JJ,NTMP2+K) = 0.D0
                  END IF
               END DO
               END IF
            END DO
            GOTO 22
         END IF
         IF (LU .GT. 0) THEN
            ICON = 0
            DO J = 1,LU
               RPAR(N-NTMP+J) = U(I-1,J)
               IF ((I .GT. 2) .AND. (U(I-1,J) .NE. U(I-2,J))) ICON=1
            END DO
            IF (ICON .EQ. 1) IODE(6) = 2 ! INPUT HAS CHANGED => REINITIALIZE
         END IF
         TT = T(I-1)
         TF = T(I)
         CALL ESDIRK34(NN,ND,FUN,TT,TF,Y0,Y0CON,WOUT,IODE,H,
     &                 RTOL,ATOL,JAC,SOLOUT,ISENSPAR,A,B,WORK,LWORK,
     &                 IWORK,LIWORK,RPAR,IPAR)
         ! ONLY CONS. CALC. IN FIRST CALL OR IF INPUT HAS CHANGED:
         IF (ISTAT(14) .EQ. 1) IODE(6) = 0
         DO J = 21,27
            ISTAT(J+2) = ISTAT(J+2) + IODE(J-10)
         END DO
         DO J = 1,LC
            JJ = I+LT*(J-1)+IENTRY
            FVEC(JJ)=W(I,J)*(Y0(J)-Y(I,J))
         END DO
         DO J = 1,LC
            WW = W(I,J)
            JJ = I+LT*(J-1)+IENTRY
            DO K = 1,N-NTMP
               FJAC(JJ,K) = WW*B(K,J)
            END DO
            IF (IC .EQ. 1) THEN
            DO K = 1,ND
               FJAC(JJ,NTMP2+K) = WW*A(K,J)
            END DO
            END IF
         END DO
22    END DO
      END IF ! ILD .EQ. 1

30    RETURN
      END ! SUBROUTINE BUILD_OBJ
```

```
      SUBROUTINE DFAULT(IW,W)
*****************************************************************************
***                                                                       ***
***              SUBROUTINE PROVIDING DEFAULT SETTINGS TO PARFIT          ***
***                                                                       ***
*****************************************************************************
C
C  Output
C  ======
C
C     IW            INTEGER ARRAY, DIMENSION ( ) : INTEGER VALUES FOR PARFIT.
C
C     W             DOUBLE PRECISION ARRAY, DIMENSION ( ) : DOUBLE PRECISION
C                   VALUES FOR PARFIT.
C
C=============================================================================
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INTEGER IW(*)
      DOUBLE PRECISION W(*)

C-----IW - VALUES:
      IW(1)  = 1      ! DEFAULT VALUES HAVE BEEN SPECIFIED
      IW(7)  = 1      ! (=1) : INTERNAL SCALING
                      ! (=2) : SCALING PROVIDED IN ELEMENTS OF W(31+N)-W(30+2*N)
      IW(8)  = 1      ! (=0) : NO COMPUTATION OF COVARINCE OR CORRELATION MATRIX
                      ! (=1) : COMPUTE COVARIANCE MATRIX + CORRELATION MATRIX
      IW(9)  = 500    ! MAXIMUM NUMBER OF OBJECTIVE FUNCTION EVALUATIONS
      IW(10) = 1      ! (=0) : NO PRINTING OF RESULTS
                      ! (=1) : PRINT SUMMARY
      IW(11) = 2      ! (=1) : NUMBER OF TOLERANCE INTERVALS
      IW(12) = 0      ! (=0) : DO NOT ESTIMATE INITIAL CONDITIONS
                      ! (=1) : ESTIMATE INITIAL CONDITION
      IW(13) = 1      ! (=0) : NUMERICAL APPROXIMATION TO JACOBIAN + dF/dP
                      ! (=1) : ANALYTIC JACOBIAN + dF/dP
      IW(14) = 0      ! (=0) : INITIAL CONDITIONS FOR DAEs ARE CONSISTENT.
                      ! (=1) : AN INTERNAL CONSISTENCY CALCULATION IS PERFORMED
                      !        BY ESDIRK34 IN THE FIRST CALL IN EACH OBJ EVALUATION
      IW(15) = 0      ! (=0) : NO REGULARIZATION
                      ! (=1) : REGULARIZATION WEIGHTS ARE PROVIDED IN W(31)-W(30+N)

      DO I = 21,30    ! INITIALIZE PART OF IW-ARRAY (FOR STATISTICAL SUMMARY)
         IW(I) = 0
      END DO

C-----W - VALUES :
      W(1)  = 1.D-4   ! RELATIVE TOLERANCE IN OBJECTIVE FUNCTION VALUE
      W(2)  = 1.D-4   ! RELATIVE TOLERANCE IN SOLUTION
      W(3)  = 1.D-4   ! ORTHOGONALITY BETWEEN RESIDUAL AND COLUMNS OF JACOBIAN
      W(4)  = 1.D0    ! FACTOR RELATED TO INITIAL STEP BOUND IN OPTIMIZER

      END ! SUBROUTINE DFAULT
```

A Sample Program

The following is a driver for the parameter estimation problem in the gas-oil cracking model, which was treated extensively in the first two parts of this thesis. Default settings are used, so no separate call is made to DFAULT.

```
      PROGRAM PARAMETER_ESTIMATION
*****************************************************************************
***                                                                       ***
***              MAIN DRIVER FOR PARFIT                                   ***
***              EXAMPLE : GAS-OIL CRACKING PROBLEM                       ***
***                                                                       ***
*****************************************************************************
      USE MODUL

      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

      PARAMETER N    = 3      ! NUMBER OF PARAMETERS TO ESTIMATE
      PARAMETER LD   = 1      ! NUMBER OF DATA SETS
      PARAMETER LC   = 2      ! NUMBER OF REGRESSED COMPONENTS
      PARAMETER LU   = 0      ! NUMBER OF INPUTS

      PARAMETER LT1 = 11      ! NUMBER OF TIME POINTS IN DATA SET 1
      PARAMETER M    = LC*LT1 ! NUMBER OF RESIDUAL FUNCTIONS

      PARAMETER NN   = 2      ! DIMENSION OF DAE SYSTEM
      PARAMETER ND   = 2      ! NUMBER OF DIFFERENTIAL EQUATIONS

      PARAMETER LW   = 6*NN**2+11*NN+7*N*NN+24+6*N+N*M+2*M+100 ! SIZE OF REAL WORKING ARRAY
      PARAMETER LIW  = NN**2+N+30                               ! SIZE OF INTEGER WORKING ARRAY

      INTEGER IW(LIW),INFO_ODE(20)
      DOUBLE PRECISION X(N),X0(NN,LD),W(LW),RPAR(4),COV(N,N)
```

```
      DOUBLE PRECISION A(NN,NN),B(N,NN),Y0CON(NN)

30    TYPE(EXPERIMENT) EX(LD)

      EXTERNAL FUN,JAC,SOLOUT

C-----SPECIFY DATA FILES:
35    EX(1)%FILE_NAME = 'PARFIT_DATA_SET1.txt'
      EX(1)%LT = LT1

C-----INITIAL PARAMETER GUESSES:
      X(1) = 1.D0 ! K1
40    X(2) = 1.D0 ! K2
      X(3) = 1.D0 ! K3

C-----INITIAL CONDITIONS:
      X0(1,1) = 1.D0
45    X0(2,1) = 0.D0

C-----REQUIRED SETTINGS FOR PARFIT:
      IW(1) = 0    ! (=0) -> SUBROUTINE DFAULT IS CALLED BY PARFIT
      IW(2) = NN
50    IW(3) = ND
      IW(4) = LU
      IW(5) = LC
      IW(6) = LD

55    CALL PARFIT(N,X,X0,EX,FUN,JAC,COV,IW,W,RPAR,IPAR)
      END ! MAIN PROGRAM PARAMETER_ESTIMATION

*************************************************************************
***                                                                   ***
60    ***                  SUBROUTINES FOR ODE/DAE SOLVER               ***
***                                                                   ***
*************************************************************************
      SUBROUTINE FUN(N,ND,T,Y,WOUT,F,RPAR,IPAR)
C-----RIGHT-HAND-SIDE FUNCTIONS:
65    IMPLICIT NONE
      INTEGER N,ND,IPAR(*)
      DOUBLE PRECISION Y(*),WOUT(*),F(*),RPAR(*),T
C-----LOCAL VARIABLES
      DOUBLE PRECISION P1,P2,P3
70
      P1 = RPAR(1)
      P2 = RPAR(2)
      P3 = RPAR(3)

75    F(1) = -(P1+P3)*Y(1)**2
      F(2) = P1*Y(1)**2 - P2*Y(2)
      RETURN
      END ! SUBROUTINE FUN

80    SUBROUTINE JAC(N,ND,NPAR,T,Y,DFUN,DFDU,RPAR,IPAR)
C-----JACOBIAN OF ODE/DAE SYSTEM:
      IMPLICIT NONE
      INTEGER N,ND,NPAR,IPAR(*)
      DOUBLE PRECISION Y(*),DFUN(N,*),DFDU(NPAR,*),RPAR(*),T
85 C-----LOCAL VARIABLES
      DOUBLE PRECISION P1,P2,P3

      P1 = RPAR(1)
      P2 = RPAR(2)
90    P3 = RPAR(3)

      DFUN(1,1) = -2.D0*(P1+P3)*Y(1)
      DFUN(2,1) = 0.D0
      DFUN(1,2) = 2.D0*P1*Y(1)
95    DFUN(2,2) = -P2

      DFDU(1,1) = -Y(1)**2
      DFDU(2,1) = 0.D0
      DFDU(3,1) = -Y(1)**2
100   DFDU(1,2) = Y(1)**2
      DFDU(2,2) = -Y(2)
      DFDU(3,2) = 0.D0
      RETURN
      END ! SUBROUTINE JAC
105
  SUBROUTINE SOLOUT(N,ND,NPAR,T,Y,WOUT,A,B)
C-----OUTPUT SUBROUTINE FOR ESDIRK34
      IMPLICIT NONE
      INTEGER N,ND,I,NPAR,J
110   DOUBLE PRECISION Y(*),T,WOUT(*),A(N,*),B(NPAR,*)

      WRITE(14,99) T, (Y(I), I=1,N), ((B(J,I), I=1,N), J=1,3)
 99   FORMAT(F12.6, 8F16.8)
      RETURN
115   END
```

Abbreviations

BDF	Backward differentiation formula
BVP	Boundary value problem
DAE	Differential-algebraic equation
DIRK	Diagonal implicit Runge-Kutta
END	External numerical differentiation
ERK	Explicit Runge-Kutta
ESDIRK	Explicit singly diagonal implicit Runge-Kutta
FIRK	Fully implicit Runge-Kutta
IND	Internal numerical differentiation
IQR	Interquartile range
IVP	Initial value problem
LS	Least squares
ML	Maximum likelihood
NLP	Nonlinear programming
ODE	Ordinary differential equation
OLS	Ordinary least squares
PDE	Partial differential equation
QP	Quadratic programming
SDIRK	Singly diagonal implicit Runge-Kutta
SQP	Sequential quadratic programming
SVD	Singular value decomposition
TLS	Total least squares
WLS	Weighted least squares

Nomenclature

Symbol	Meaning	Dimension
$\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$	Matrix/vectors with Runge-Kutta coefficients	-
$\boldsymbol{B}$	Hessian approximation	$m \times n_p \times n_p$
$\boldsymbol{c}(\cdot)$	Vector of constraint functions	m_c
$\boldsymbol{C}$	Covariance matrix for parameter estimates	$n_p \times n_p$
$\boldsymbol{D}$	Diagonal matrix	$n \times n$
E	Expectation operator	-
$f(\cdot)$	Objective function	1
$\boldsymbol{f}(\cdot)$	Vector of right-hand-side functions for DAE system	n
$\boldsymbol{F}(\cdot)$	Vector function for fully implicit DAEs	n
h	Step size in ODE/DAE solver	1
$\boldsymbol{h}$	Direction in optimization algorithm	n_p
$\boldsymbol{h}(\cdot)$	Vector of functions in measurement equation	m
$\boldsymbol{H}$	Hessian matrix	$m \times n_p \times n_p$
i, j, k	Indices	1
$\boldsymbol{I}_m$	Identity matrix	m
$\boldsymbol{J}$	Jacobian matrix	$m \times n_p$
l_c	Number of components regressed	1
l_d	Number of data sets	1
l_t^k	Number of measurements of each component in data set k	1
$L(\cdot)$	Likelihood function	1
m	Number of measurements ($m = l_c l_t$)	1
m_c	Total number of constraints	1
m_e	Number of equality constraints	1
m_s	Number of multiple shooting intervals	1
n	Number of dependent variables	1
n_a	Number of algebraic variables	1
n_d	Number of differential variables	1
n_p	Number of parameters	1
n_u	Number of inputs	1
$p(\cdot)$	Probability density function	1
$\boldsymbol{r}(\cdot)$	Vector of residual functions	m
$R(\cdot)$	Stability function for Runge-Kutta methods	1
$\boldsymbol{s}_i$	ith multiple shooting parameter vector *or* sensitivities with respect to parameter i	n n
t	Independent variable (time)	1

Symbol	Meaning	Dimension
t_{m-n_p}	Quantile of the t-distribution with $m - n_p$ degrees of freedom	1
$\boldsymbol{u}$	Vector of inputs	n_u
$\boldsymbol{v}$	Vector of weights for independent variable	m
$\boldsymbol{V}$	Covariance matrix of measurement errors	$m \times m$
$\boldsymbol{w}$	Vector of weights for dependent variables	m
w_{reg}	Regularization weight	1
$\boldsymbol{y}(\cdot)$	Vector of dependent variables	n
$\tilde{\boldsymbol{y}}$	Vector of measurements	m
$\boldsymbol{z}$	Vector of algebraic variables	n_a
$\mathcal{D}$	Trust region	-
$\mathcal{I}(\cdot)$	Index set for active inequality constraint	-
$\mathcal{L}(\cdot)$	Lagrangian function	1
$\mathcal{N}(\mu, \boldsymbol{V})$	Gaussian distribution with mean μ and covariance matrix $\boldsymbol{V}$	-
α	Step length parameter	1
$\boldsymbol{\delta}$	Vector of residuals related to independent variable	m
Δ	Radius of trust region	1
$\boldsymbol{\varepsilon}$	Vector of measurement errors in dependent variables	m
ϵ_m	Unit round-off	1
$\boldsymbol{\theta}$	Parameter vector	n_p
λ	Continuation parameter	1
$\boldsymbol{\lambda}$	Lagrange multiplier vector	m_c
μ	Levenberg-Marquardt parameter *or* mean value	1
$\boldsymbol{\xi}$	Vector of measurement errors in independent variable	m
σ, σ^2	Standard deviation and variance	-
ϱ	Trust region parameter	1
τ_i	ith multiple shooting node	1
$\phi(\cdot)$	Line search function	1
∇	Differential operator	-

References

Al-Baali, M. and Fletcher, R. (1985). Variational Methods for Nonlinear Least Squares. *Journal of the Operational Research Society*, **36**(5), 405–421.

Al-Baali, M. and Fletcher, R. (1986). An Efficient Line Search for Nonlinear Least Squares. *Journal of Optimization Theory and Applications*, **48**(3), 359–377.

Alexander, R. (1977). Diagonal Implicit Runge-Kutta Methods for Stiff O.D.E.'s. *SIAM Journal of Numerical Analysis*, **14**(6), 1006–1021.

Alexander, R. (2003). Design and Implementation of DIRK Integrators for Stiff Systems. *Applied Numerical Mathematics*, **46**, 1–17.

Bard, Y. (1974). *Nonlinear Parameter Estimation*. Academic Press, New York, USA.

Bauer, I.; Bock, H. G.; Körkel, S. and Schlöder, J. P. (2000). Numerical Methods for Optimum Experimental Design in DAE Systems. *Journal of Computational and Applied Mathematics*, **120**, 1–25.

Biegler, L. T.; Damiano, J. J. and Blau, G. E. (1986). Nonlinear Parameter Estimation: A Case Study Comparison. *AIChE Journal*, **32**(1), 29–43.

Bock, H. G. (1981). Numerical Treatment of Inverse Problems in Chemical Reaction Kinetics. In K. H. Ebert; P. Deuflhard and W. Jäger, editors, *Modelling of Chemical Reaction Systems*, pages 102–125. Springer Series in Chem. Phys.

Bock, H. G. (1983). Recent Advances in Parameter Identification Techniques for O.D.E. In P. Deuflhard and E. Hairer, editors, *Numerical Treatment of Inverse Problems in Differential and Integral Equations*, pages 95–121. Birkhäuser, Boston, USA.

Brenan, K. E.; Campbell, S. L. and Petzold, L. R. (1996). *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, Philadelphia, USA.

Brown, P. N.; Hindmarsh, A. C. and Petzold, L. R. (1998). Consistent Initial Condition Calculation for Differential-Algebraic Systems. *SIAM Journal of Scientific Computing*, **19**(5), 1495–1512.

Cameron, I. T. (1983). Solution of Differential-Algebraic Systems Using Diagonal Implicit Runge-Kutta Methods. *IMA Journal of Numerical Analysis*, **3**, 273–289.

Cao, Y.; Li, S. and Petzold, L. (2002). Adjoint Sensitivity Analysis for Differential-Algebraic Equations: Algorithms and Software. *Journal of Computational and Applied Mathematics*, **149**, 171–191.

Cao, Y.; Li, S.; Petzold, L. and Serban, R. (2003). Adjoint Sensitivity Analysis for Differential-Algebraic Equations: The Adjoint DAE System and its Numerical Solution. *SIAM Journal of Scientific Computing*, **24**(3), 1076–1098.

Caracotsios, M. and Stewart, W. E. (1985). Sensitivity Analysis of Initial Value Problems with Mixed ODEs and Algebraic Equations. *Computers and Chemical Engineering*, **9**(4), 359–365.

Dennis, J. E. and Schnabel, R. B. (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, USA.

Dennis, J. E.; Gay, D. M. and Welsch, R. E. (1981). An Adaptive Nonlinear Least Squares Algorithm. *ACM Transactions on Mathematical Software*, **7**(3), 348–368.

Enright, W. H.; Jackson, K. R.; Nørsett, S. P. and Thomsen, P. G. (1986). Interpolants for Runge-Kutta Formulas. *ACM Transactions on Mathematical Software*, **12**(3), 193–218.

Feehery, W. F.; Tolsma, J. E. and Barton, P. I. (1997). Efficient Sensitivity Analysis of Large-Scale Differential-Algebraic Systems. *Applied Numerical Mathematics*, **25**, 41–54.

Frandsen, P. E.; Jonasson, K.; Nielsen, H. B. and Tingleff, O. (1999). Unconstrained Optimization. Informatics and Mathematical Modelling, Technical University of Denmark.

Gill, P. E.; Murray, W.; Saunders, M. A. and Wright, M. H. (1986). *User's Guide for NPSOL: A Fortran Package for Nonlinear Programming*. Systems Optimization Laboratory, Department of Operations Research, Stanford University.

Gronwall, T. H. (1919). Note on the Derivatives with Respect to a Parameter of the Solutions of a System of Differential Equations. *Annals of Mathematics*, **20**, 292–296.

Hairer, E. and Wanner, G. (1996). *Solving Ordinary Differential Equations II*. Springer, second revised edition.

Hairer, E.; Lubich, C. and Roche, M. (1987). Error of Runge-Kutta Methods for Stiff Problems Studied Via Differential Algebraic Equations. Technical report, Dept. de Mathématiques, Université de Genève.

Hairer, E.; Nørsett, S. and Wanner, G. (1992). *Solving Ordinary Differential Equations I*. Springer, second revised edition.

Hansen, P. C. (1998). Regularization Tools. A Matlab Package for Solution of Discrete Ill-Posed Problems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark.

Hindmarsh, A. C. and Serban, R. (2002). *User Documentation for CVODES, An ODE Solver with Sensitivity Analysis Capabilities*. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory.

Issanchou, S.; Cognet, P. and Gabassud, M. (2003). Precise Parameter Estimation for Chemical Batch Reactions in Heterogeneous Medium. *Chemical Engineering Science*, **58**, 1805–1813.

Jørgensen, J. B.; Rawlings, J. B. and Jørgensen, S. B. (2004). Numerical Solution of Unconstrained Nonlinear Optimal Control Problems. Submitted.

Khorasheh, F.; Kheirolomoom, A. and Mireshghi, S. (2002). Application of an Optimization Algorithm for Estimating Intrinsic Kinetic Parameters of Immobilized Enzymes. *Journal of Bioscience and Bioengineering*, **94**(1), 1–7.

Kristensen, M. R.; Jørgensen, J. B.; Thomsen, P. G. and Jørgensen, S. B. (2004a). An ESDIRK Method with Sensitivity Analysis Capabilities. *Submitted to Computers and Chemical Engineering*.

Kristensen, M. R.; Jørgensen, J. B.; Thomsen, P. G. and Jørgensen, S. B. (2004b). Extension of ESDIRK34 to DAE Systems. Technical report, Department of Chemical Engineering, Technical University of Denmark.

Kristensen, N. R. (2003). *Fed-Batch Process Modelling for State Estimation and Optimal Control*. Ph.D. thesis, Technical University of Denmark.

Kröner, A.; Marquardt, W. and Gilles, E. D. (1992). Computing Consistent Initial Values for Differential-Algebraic Equations. *Computers and Chemical Engineering*, **16**, 131–138.

Kuhlmann, C.; Bogle, I. D. L. and Chalabi, Z. S. (1998). Robust Operation of Fed Batch Fermenters. *Bioprocess Engineering*, **19**, 53–59.

Law, V. J. and Sharma, Y. (1997). Computation of the Gradient and Sensitivity Coefficients in Sum of Least Squares Minimization Problems with Differential Equation Models. *Computers and Chemical Engineering*, **21**(12), 1471–1479.

Leineweber, D. B. (1995). *Analyse und Restrukturierung Eines Verfahrens Zur Direkten Lösung Von Optimal-Steuerungsproblemen*. Master's thesis, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Universität Heidelberg.

Leis, J. R. and Kramer, M. A. (1988). The Simultaneous Solution and Sensitivity Analysis of Systems Described by Ordinary Differential Equations. *ACM Transactions on Mathematical Software*, **14**(1), 45–60.

Levenberg, K. (1944). A Method for the Solution of Certain Nonlinear Problems in Least Squares. *Quarterly of Applied Mathematics*, **2**, 164–168.

Luksan, L. and Spedicato, E. (2000). Variable Metric Methods for Unconstrained Optimization and Nonlinear Least Squares. *Journal of Computational and Applied Mathematics*, **124**, 61–95.

Madsen, K. (1988). A Combined Gauss-Newton and Quasi-Newton Method for Nonlinear Least Squares. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark.

Madsen, K.; Nielsen, H. B. and Tingleff, O. (1999). Methods for Nonlinear Least Squares. Informatics and Mathematical Modelling, Technical University of Denmark.

Majer, C.; Marquardt, W. and Gilles, E. D. (1995). Reinitialization of DAE's After Discontinuities. *Computers and Chemical Engineering*, **19**, 507–512.

Maly, T. and Petzold, L. R. (1996). Numerical Methods and Software for Sensitivity Analysis of Differential-Algebraic Systems. *Applied Numerical Mathematics*, **20**, 57–79.

Marquardt, D. W. (1963). An Algorithm for Least Squares Estimation of Nonlinear Parameters. *SIAM Journal of Applied Mathematics*, **11**(2), 431–441.

Mathworks (2003). MATLAB *Optimization Toolbox User's Guide*. Mathsworks Inc., Natick, MA, USA.

Moré, J. J. (1977). The Levenberg-Marquardt Algorithm: Implementation and Theory. In G. A. Watson, editor, *Numerical Analysis*, pages 105–116. Springer Verlag, Berlin, Germany.

Najafi, M.; Nikoukhah, R. and Campbell, S. L. (2004). Computation of Consistent Initial Values for Multi-Mode DAEs: Application to Scicos. Proc. Computer Aided Control Design, Taipei, To appear.

Nielsen, H. B. (1999). Damping Parameter in Marquardts Method. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark.

Nielsen, H. B. (2000). Checking Gradient. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer-Verlag, New York, USA.

Nørsett, S. P. (1974). Semi-Explicit Runge-Kutta Methods. Technical Report 6, Department of Mathematics, University of Trondheim. ISBN-82-7151-009-6.

Osborne, M. R. (1976). Nonlinear Least Squares – the Levenberg Algorithm Revisited. *Journal of the Australian Mathematical Society, series B*, **19**(3), 343–357.

Pantelides, C. C. and Barton, P. I. (1993). Equation-Oriented Dynamic Simulation: Current Status and Future Perspectives. *Computers and Chemical Engineering*, **17**, 263–285.

Petzold, L. R. (1982). DASSL: A Differential/Algebraic System Solver. 10th IMACS World Congress on System Simulation and Scientific Computation.

Prothero, A. and Robinson, A. (1974). On the Stability and Accuracy of One-Step Methods for Sloving Stiff Systems of Ordinary Differential Equations. *Mathematics of Computation*, **28**(125), 145–162.

Sandu, A.; Daescu, D. N. and Carmichael, G. R. (2003). Direct and Adjoint Sensitivity Analysis of Chemical Kinetic Systems with KPP: Part 1 – Theory and Software Tools. *Atmospheric Environment*, **37**, 5083–5096.

Schittkowski, K. (1988). Solving Nonlinear Least Squares Problems by General Purpose SQP-Method. In K.-H. Hoffmann; J.-B. Hiriart-Urruty; C. Lemarechal and J. Zowe, editors, *Trends in Mathematical Optimization*, pages 295–309. Birkhäuser, Berlin.

Schittkowski, K. (2002). *Numerical Data Fitting in Dynamical Systems*. Kluwer Academic Publishers, The Netherlands.

Schlegel, M.; Marquardt, W.; Ehrig, R. and Nowak, U. (2004). Sensitivity Analysis of Linearly-Implicit Differential-Algebraic Systems by One-Step Extrapolation. *Applied Numerical Mathematics*, **48**(1), 83–102.

Seber, G. A. F. and Wild, C. J. (2003). *Nonlinear Regression*. John Wiley & Sons, New Jersey.

Stortelder, W. J. H. (1998). *Parameter Estimation in Nonlinear Dynamical Systems*. Ph.D. thesis, National Research Institute for Mathematics and Computer Science, University of Amsterdam.

Thomsen, P. G. (2002). A Generalised Runge Kutta Method of Order Three. Technical report, Informatics and Mathematical Modelling, Technical University of Denamrk.

Tjoa, I.-B. and Biegler, L. T. (1991). Simultaneous Solution and Optimization Strategies for Parameter Estimation of Differential-Algebraic Equation Systems. *Industrial and Engineering Chemistry Research*, **30**, 376–385.

Villadsen, J. and Michelsen, M. L. (1978). *Solution of Differential Equation Models by Polynomial Approximation*. Prentice-Hall Inc., Englewood Cliffs, New Jersey.