Technische Universität München

Department of Mathematics

Chair of Mathematical Modeling of Biological Systems

Master's Thesis

Assessment of the integration based method to calculate profile likelihoods for mathematical models in biology

Franziska Frank

in cooperation with

HelmholtzZentrum münchen
Deutsches Forschungszentrum für Gesundheit und Umwelt
ICB Institute of Computational Biology

Supervisor: Prof. Dr. Dr. Fabian J. Theis

Advisor: Sabrina Hock, Dr. Jan Hasenauer

Submission Date: 2013/11/28

Zusammenfassung

In der Biologie werden vermehrt mathematische Modelle verwendet, um komplexe Prozesse zu analysieren. Damit diese Modelle die experimentellen Daten wiederspiegeln knnen, ist die zuverlässige Parameterschätzung von großem Interesse. Allerdings ergeben sich Herausforderungen bei der Schätzung von nicht identifizierbaren Parametern, weshalb die Analyse der Identifizierbarkeit von Parametern von groer Wichtigkeit ist. Diese Analyse kann mit den sogenannten Likelihood Profilen durchgeführt werden, wobei das klassische Vorgehen darin besteht, die Likelihood-Funktion schrittweise zu optimieren.
Diese Methode ist rechnerisch kostspielig, da die wiederholte Optimierung der Likelihood-Funktion notwendig ist. In dieser Arbeit wird daher die potentiell schnellere integrationbasierte Methode von Chen und Jennrich untersucht, welche sich die Optimierung unter Nebenbedingung mit dem Lagrange Multiplikator zu Nutze macht. Diese Methode soll die Berechnung beschleunigen, da sie, im Gegensatz zu der klassischen Methode, keine weiteren Optimierungen benötigt und die, aus der Optimierung unter Nebenbedingung entstehenden, Differentialgleichung mit bereits bewährten Algorithmen gelöst werden kann. Ein weiterer Vorteil ist, dass diese Methode auf eine einfache Art und Weise auch für Profile verwendet werden kann, die für eine Kombination von Parametern berechnet werden, ohne dabei das Modell zusätzlich reparametrisieren zu müssen.
Im Zuge dieser Masterarbeit wurde eine MATLAB Implementierung erstellt, die Likelihood Profile mit der integrationbasierten Methode berechnet. Diese Implementierung beinhaltet auch Methoden um benötigte Ableitungen zu approximieren oder geeignet zu ersetzen, falls diese nicht verfübar sind. Außerdem wurden verschiedene Möglichkeiten, das Anfangswertproblem als gewöhnliche oder algebraische Differentialgleichung zu lösen, implementiert und verglichen.
Zur Evaluierung der implementierten Funktion wurde diese auf eine Reihe von Beispielen angewendet. Die berechneten Profile wurden, im Bezug auf Genauigkeit und Anzahl der Evaluationen des Modells, mit den Ergebnissen der klassischen Methode verglichen. Das Hauptaugenmerk lag dabei auf den Modellen, die nicht-identifizierbare Parameter beinhalteten und vorallem auf einem partiellen Differentialgleichungsmodell, da eine Reduzierung der Auswertungen des Modells insbesonders bei solchen Problemen wünschenswert ist.
Beobachtet wurde, dass die Implementierung die Auswertungen stark reduziert, wobei die erreichte Genauigkeit vergleichbar ist mit der klassischen Methode, wenn die exakten oder gute Approximationen der Ableitungen verfügbar sind. Außerdem wurde die Nebenbedingung des Optimierungsproblems in der logarithmischen Skala definiert, was die Berechnung zusätzlich verbesserte. In diesem Fall kann die Hessematrix sogar mit der Identitätsmatrix ersetzt werden, wenn die Implementierung einer genaueren Approximation umgangen werden soll.
Die Arbeit schließt mit Vorschlägen zu weiteren Verbesserung der Implementierung und zu potentiellen Anwendungsbereichen ab.

Abstract

Together with the advancing use of mathematical models in biology, the importance ascribed to the reliable parameter estimation of the parameters in these models is significantly increasing. But challenges of this estimation are for example, if the model contains non-identifiable parameters. Therefore it is crucial to apply the identifiability analysis. This analysis can be done using the so called profile likelihoods, where the classical method to calculate these profiles is, to apply step-wise optimization.

However, the classical method is computational demanding, due to the repeated optimization. This work will therefore focus on the potential faster integration based approach from Chen and Jennrich [1], using constraint optimization with Lagrange multipliers. This method accelerates the computation, as – in contrast to the classical method – it does not need additional optimization algorithms for the calculation. Another advantage of this method is that it can easily be applied to prediction profile likelihoods without the additional application of a reparametrization.

In the course of this masters thesis a MATLAB implementation of the integration based profile likelihood calculation is provided. This implementation also includes various methods to approximate or substitute the required derivatives, if these are not available. This is especially important for this method, since the analytical calculation of the hessian of the log-likelihood is often limited. Furthermore a selection of different ODE or DAE solvers for the calculation is provided and evaluated concerning their performances for a fast and accurate profile calculation. To evaluate the method, the function was applied to a number of example models and the calculated profiles were then compared to the classical method in terms of accuracy and the number of evaluations of the model.

One novel result is thereby that the profile calculation was successfully accomplished for models with non-identifiable parameters, which was not yet investigated in [1]. Furthermore the method was in particular applied to a real data PDE model, where the reduction of computation time for the profiles receives the biggest benefit.

It was found that the method reduces the evaluations of the model significantly without a great increase of inaccuracy, given that exact derivatives or acceptable approximations are available. Moreover one observation was, that defining the optimization constraint in the logarithmic scale particularly improves the algorithm. It was found that in this case, the hessian can also be substituted with the identity matrix to save the time implementing an accurate approximation of the hessian.

Acknowledgement

At first I would like to thank Fabian J. Theis for the opportunity to write my masters thesis at the ICB, where I was able to meet some excellent scientists, and I am grateful for his counsel and suggestions.

I am also obliged to Sabrina Hock for her sublime guidance during my thesis. Her instructions, I will always keep in mind and appreciate her contribution very much, for that it improved my work not only for academical reasons. Furthermore my sincere thanks go to Dr. Jan Hasenauer for his highly respected advice and respond to every question.

Moreover I would like to express my thanks to all people, which supported me during my studies. Especially I owe this thesis to my parents, which motivated and encouraged me in my academic career as they did during my whole life. Not less important for me, was the help and humour of my sisters.

Last but not least I would like to address my best counsellor in the way he likes it best: Not mentioning it too much.

Contents

Chapter 1

Introduction

The aim of mathematical modelling in biology is, to describe biological processes as accurately as possible. This leads to complex models, which include a large number of free parameters (reaction rates, scaling parameters, etc.). A major challenge is that direct measurements of all model parameters are often unavailable or infeasible. Furthermore, the measurements are noise corrupted or parameters can only be measured in combination with other parameters. Some reaction rates, for instance, cannot directly be measured in experiments. Taking these challenges of direct measurements into account, the reliable estimation of parameters by using observable data, is an issue of great interest.
Additional to the impossibility of direct measurements, models can also include non-identifiable parameters, which means they cannot precisely be estimated. That makes the parameter fitting for these parameters impossible, since different parameter sets produce the same output. This is the case, for example, if two parameters occur only proportional to each other (e.g. $\theta_1 \cdot \theta_2$ or $\frac{\theta_1}{\theta_2}$). It is important to investigate the identifiability of the model parameters and judge the results of the parameter fitting process accordingly. Furthermore, if knowledge about practical non-identifiability is available (or easy to access) it might be possible to resolve this with additional experiments [2].
A common method to describe identifiability is to use confidence intervals, which define a region of parameter values being the most likely ones. These intervals are finite for identifiable parameters [2] and can be calculated using profile likelihoods. A profile likelihood can be compared to a ridge walk on a mountain, i.e. moving along one parameter direction one stays on the highest possible point of the likelihood surface. Considering the theory, the calculation of profiles is quite similar to the calculation of the maximum likelihood estimator (MLE), being the parameter set for which the likelihood attains its maximum, but instead of optimizing over all parameters, one parameter is set constant and then the function is optimized over all other parameters. In the next step the constant parameter is changed slightly and the optimization is repeated.
Profiles have already been used frequently for detecting identifiability and calculating confidence intervals [2, 3, 4, 5, 6], however, the classical calculation of the profiles is computationally demanding due to the large amount of optimizations in the algorithm. The effort increases especially for models which are non-trivial to solve, e.g. a system of partial differential equations.
A fast and easy access to profiles and therefore to information about identifiability of parameters is the goal of this thesis. The work will thereby be focused on the integra-

tion based method from [1], which has the potential to accelerate the profile likelihood calculation compared to the classical approach. This method uses constraint optimization instead of step wise repeated optimization, that means no additional optimization is necessary. Moreover, to solve the differential equation, resulting from the constraint optimization, approved and established methods can be used, which is hypothesized to be additionally faster than the classical method, due to probable wider step sizes.

In this thesis the gain of the integration based method is investigated, by providing a MATLAB implementation of the method, for further use. This implementation is tested with various model examples, to investigate the gain of the new method compared to the classical approach.

In contrast to [1] there are also models with non-identifiable parameters used and challenges for these cases are discussed. This is important, since non-identifiable parameters appear in a wide range of models, but the profile calculation using the classical method takes especially long for these parameters.

In this thesis variations of the integration based method will be compared to the optimization based profile calculation method, concerning different solvers and approximations of derivatives, to find the most favourable choice in terms of accuracy and speed. This exceeds the work in [1], since a larger variety of approximations and solvers are applied. The differential equation, which results from the constraint optimization is also addressed with differential algebraic equation (DAE) solvers.

Particularly, it will be investigated how parameter transformation, i.e. transformation to the logarithmic scale, can be applied to the optimization problem and if it enhances the calculation performance in contrast to the linear scale, like it appears in other applications of the logarithmic scale. In this context it is found that the constraint of the optimization can be defined in the logarithmic scale, which additionally reduces the evaluations of the cost function significantly.

Additionally it will be discussed, if the correction term, introduced by [1], always improves the profile computation in the practical application and specifically in the case of non-identifiable parameters.

Furthermore it is suggested that the integration based approach is superior to the classical method concerning *prediction profile likelihoods*, which are similar to ordinary profile likelihoods, defined for not only for one, but a combination of parameters. In this case not just numerical computation time but applications of analytical methods can be saved, since no reparametrization is necessary.

Finally, the method is applied to a parameter estimation problem in the context of partial differential equations (PDEs), which is an example for a numerically costly model, where also real measurement data will be considered. In other words, the case, where a reduction of evaluations would be most advantageously. A successfully application confirms a possible improvement of the parameter estimation for higher complex models.

Chapter 2

Mathematical Background

This section introduces the mathematical background used in this thesis. It contains definitions and propositions as well as notation, which will be used throughout this work.

2.1 Parameter estimation for biological processes

For parameter estimation a mathematical model to the experimentally observed data is fitted. There are many different model types, but the concerned models in this thesis are either based on ordinary differential equations (ODEs) or partial differential equation (PDE)s.
For an introduction, the concentration of the species $x(t, \theta) \in \mathbb{R}$ should be modelled, depending on time $t \in \mathbb{R}^+$ and n unknown parameters $\theta \in \mathbb{R}^n$ (containing for example kinetic parameters of the model). The model describes the dynamics of the concentration by an ODE:

$$F(t, x, x', \theta, x_0) = 0 \tag{2.1}$$

with $x' := \frac{dx}{dt}$ and the initial conditions $x_0 := x(t_0, \theta)$ at the starting point t_0.
The dynamics of the modelled process is optimally reflected with the so called true parameter set θ^*.
The species can also be modelled with PDEs, then the concentration $x(\tau, \theta)$ does not depend on the time t, but on the vector variable $\tau \in \mathbb{R}^d$ consisting of $d \in \mathbb{N}$ variables, which can be time and/or location coordinates (e.g. $\tau = (t, z_1, z_2)^T$ with t time and z_1, z_2 being the coordinates on a 2-dimensional field).
Then the equation looks as follows:

$$F(\tau_1, ..., \tau_d, x, \frac{\partial x}{\partial \tau_1}, ..., \frac{\partial x}{\partial \tau_d}, \frac{\partial^2 x}{\partial \tau_1 \partial \tau_1}, ..., \frac{\partial^2 x}{\partial \tau_1 \partial \tau_d}, ..., \theta, x_0) = 0$$

with initial condition: $x_0 = x(\tau_0, \theta)$ at the starting point τ_0.
To be general, it is possible that $x(t, \theta)$ or $x(\tau, \theta)$ respectively (In the following τ will be used as a generalization of t) are vectors describing not only one, but various species. Then $x(\tau, \theta) \in \mathbb{R}^q$, with q be the number of species. In this case, potentially not all species are directly observable and to map x onto the observable species y the observation

function ψ is used.
So it is:

$$y(\tau,\theta) = \psi(x(\tau,\theta),\theta) \text{ with } y(\tau,\theta) \in \mathbb{R}^p \text{ and } p \leq q$$

with $y(\tau,\theta)$ being those species, which are directly measurable. Moreover the observation function can also include a scaling factor (e.g. $y(\tau,\theta) = s \cdot x(\tau,\theta)$), which is also part of the parameter set θ if unknown. In the considered examples (see section 2.3) there is only one observable, i.e. $p = 1$.

2.1.1 Maximum likelihood estimation

The experimental measurement at point τ_k for $k = (0,...,m)$ is the observation data $\tilde{y}_k$ for the observable $y(\tau_k,\theta)$. In most cases the observation data is imperfect, because the experiments incorporate measurement noise. In the examples later applied in this thesis the measurements are supposed to be subject to additive normal distributed noise such that:

$$\tilde{y}_k = y\left(\tau_k,\theta\right) + \epsilon_k$$

with $\epsilon_k \sim \varrho_k = \mathcal{N}(0,\sigma_k^2)$ being the error at point τ_k.
In general the true set of parameters θ^* is unknown and therefore the reliable derivation of an estimate is necessary. For the parameter estimation, the parameters are fitted to the observation data, that means an estimator θ' for parameter set is calculated, for which the model reflects the data optimally.
One common estimator is the maximum likelihood estimator (MLE) $\theta' = \hat{\theta}$, which is the set of parameters for that the likelihood function attains its maximum.
The *likelihood function* $\mathrm{L}(\theta)$ is given by

$$\mathrm{L}(\theta) := \prod_{k=1}^{m} \frac{1}{\sqrt{2\pi\sigma_k^2}} \cdot \exp\left(\frac{(\tilde{y_k} - y(\tau_k,\theta))^2}{\sigma_k^2}\right)$$

and the MLE by

$$\hat{\theta} := \underset{\theta}{\operatorname{argmax}}\ \mathrm{L}(\theta)$$

Applications often show, that it is analytically favourable and numerically more efficient to use the *log-likelihood* $\mathrm{l}(\theta)$, because multiplications become sums, which may be easier to evaluate and eventual exponential functions disappear.
The log-likelihood is defined as:

$$\mathrm{l}(\theta) \quad := \quad \log(\mathrm{L}(\theta)) = \sum_{k=1}^{m} \log\left(\frac{1}{\sqrt{2\pi\sigma_k^2}} \cdot \exp\left(\frac{(\tilde{y_k} - y(\tau_k,\theta))^2}{\sigma_k^2}\right)\right) \tag{2.2}$$

In the application examples the variance σ_k^2 is constant and parameter independent for all k, so that $\epsilon_k \sim \mathcal{N}(0,\sigma^2)\ \forall k = \{1,...,m\}$. Inserted in $\mathrm{l}(\theta)$, this results in:

$$\mathrm{l}(\theta) \quad = \quad -\frac{1}{2}\sum_{k=1}^{m}\left(\log(2\pi\sigma^2) + \frac{(\tilde{y_k} - y(\tau_k,\theta))^2}{\sigma^2}\right) \tag{2.3}$$

For the applied method in this thesis, the first and second derivative of the log-likelihood function is necessary. In general θ is a multidimensional parameter in $\mathbb{R}^n$, therefore the first derivative of $\mathrm{l}(\theta)$ with respect to all elements of θ is a multidimensional vector of length n, called the *gradient* and the second derivative is a $n \times n$ matrix, called the *hessian*. The symbols used in this thesis for the operators of the gradient and the hessian are $\nabla_\theta \in \mathbb{R}^n$ and $\nabla_\theta^2 \in \mathbb{R}^{n\times n}$ respectively and are defined as:

$$\nabla_\theta := \begin{pmatrix} \frac{\partial}{\partial\theta_1} \\ \vdots \\ \frac{\partial}{\partial\theta_n} \end{pmatrix} \quad \forall\ \theta \in \mathbb{R}^n \tag{2.4}$$

$$\nabla_\theta^2 := \begin{pmatrix} \frac{\partial^2}{\partial\theta_1\partial\theta_1} & \cdots & \frac{\partial^2}{\partial\theta_1\partial\theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial\theta_n\partial\theta_1} & \cdots & \frac{\partial^2}{\partial\theta_n\partial\theta_n} \end{pmatrix} \quad \forall\ \theta \in \mathbb{R}^n \tag{2.5}$$

Provided that the first and second derivative of the model $y(\tau_k,\theta)$ with respect to θ are available, it is possible to compute the derivatives of the likelihood function, which look as follows:

$$\nabla_\theta \mathrm{l}(\theta) = \tfrac{1}{\sigma^2}\sum_{k=1}^{m}(\tilde{y_k} - y(\tau_k,\theta)) \cdot \nabla_\theta y(\tau_k,\theta) \tag{2.6}$$

$$\nabla_\theta^2 \mathrm{l}(\theta) = \tfrac{1}{\sigma^2}\sum_{k=1}^{m}(\tilde{y_k} - y(\tau_k,\theta)) \cdot \nabla_\theta^2 y(\tau_k,\theta) - \nabla_\theta y(\tau_k,\theta) \cdot \nabla_\theta y(\tau_k,\theta)^T \tag{2.7}$$

The problem with most models is, that an analytical solution is very difficult or impossible to calculate, hence the model needs to be solved numerically. A model, for example, which consists of a system of ODEs can be solved by MATLAB built-in ODE solvers. But these ODE solvers only return the numerical solution of the model at discrete time points, whereas the derivatives have to be computed with additional methods, for example using the sensitivity equations or finite differences (FD) approximations.
But numerical methods as well as approximations generate numerical errors, due to limited machine precision. It must be considered, that these deviations can influence the results, when using approximated derivatives.

2.1.2 Parameter transformation

The transformation of parameters is used to apply different parameter scales in computations. The parameters are therefore mapped onto the new scale using a continuous and strictly monotone function.
One scale, which is widely used in biological models, is the logarithmic scale, since the parameters have often a wide range of values. Due to a better handling of very small or quite big values the logarithmic scale is frequently favoured and is also applied to the models in this thesis.
The idea of the logarithmic scale is to reparametrize the model such that:

$$\xi = \log(\theta) \tag{2.8}$$

$$\Rightarrow \exp(\xi) = \theta \tag{2.9}$$

The logarithm and the exponential function is in this case point-wise applied to the parameter vectors θ and ξ respectively, which results in $\exp(\xi) \in \mathbb{R}^n$ being a vector, too.
The parameter θ is substituted with the function $\exp(\xi)$ and the algorithm is then run in terms of ξ. Applying the logarithmic scales often shows a faster and more accurate computation.
However the calculating of the derivatives require the additional differentiation of the parameter transformation function. This is for the first derivative accomplished by multiplying with $\exp(\xi)$, since

$$\nabla_\xi (\exp(\xi)) = \exp(\xi)$$

The gradient of the log-likelihood function in the logarithmic scale changes to:

$$\nabla_\xi \mathrm{l}(\exp(\xi)) = \frac{1}{\sigma^2} \cdot \sum_{k=1}^{m} (\tilde{y_k} - y(\tau_k, \exp(\xi))) \cdot \begin{pmatrix} \frac{\partial}{\partial \xi_1} y(\tau_k, \exp(\xi_1)) \cdot \exp(\xi_1) \\ \vdots \\ \frac{\partial}{\partial \xi_n} y(\tau_k, \exp(\xi_n)) \cdot \exp(\xi_n) \end{pmatrix}$$

The second differentiation to calculate the hessian is more complicated, but nonetheless feasible, if one consequently follows the product and chain rule.

$$\begin{aligned} \nabla^2_\xi \mathrm{l}(\exp(\xi)) = {} & \tfrac{1}{\sigma^2} \cdot \sum_{k=1}^{m} (\tilde{y}_k - y(\tau_k, \exp(\xi))) \cdot \big(\exp(\xi)^T \nabla^2_\xi y(\tau_k, \exp(\xi)) \cdot \exp(\xi) \, ... \\ & + \begin{pmatrix} \frac{\partial}{\partial \xi_1} y(\tau_k, \exp(\xi))_1 \cdot \exp(\xi_1) & \cdots & 0 \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial}{\partial \xi_n} y(\tau_k, \exp(\xi))_n \cdot \exp(\xi_n) \end{pmatrix} \Bigg) ... \\ & - \nabla_\xi y(\tau_k, \exp(\xi)) \cdot \exp(\xi) \cdot (\nabla_\xi y(\tau_k, \exp(\xi)) \cdot \exp(\xi))^T \end{aligned} \tag{2.10}$$

Note, that the differentiation of the term $\exp(\xi_i)$ is zero, except it is differentiated with respect to ξ_i, so that the second term of the product rule can not be neglected.
In the following no difference will be made between θ and ξ in the linear or logarithmic scale, since there is no restriction in renaming the parameters. The difference is only indicated if necessary.

2.1.3 Confidence intervals

Confidence intervals to the confidence level α define intervals around the parameter estimate $\hat{\theta}_i$ with $i \in (1, ...n)$, in which the true value θ^*_i lies with probability α. They can be defined as *point-wise confidence intervals* or for all n parameters together, which is then called a *simultaneous confidence interval.*
One possible way to define likelihood-based confidence intervals is the *finite sample confidence interval* [2].

Definition 1. ***Finite sample confidence interval***
Let $L(\theta)$ and $l(\theta)$ be the likelihood and log-likelihood function respectively. $\hat{\theta}$ be the MLE,

then a finite sample confidence interval *is defined as:*

$$\left\{\theta \in \mathbb{R} \middle| \mathrm{l}(\hat{\theta}) - \mathrm{l}(\theta) < \frac{\Delta_\alpha}{2}\right\}$$

or

$$\left\{\theta \in \mathbb{R} \middle| \frac{\mathrm{L}(\theta)}{\mathrm{L}(\hat{\theta})} > \exp\left(-\frac{\Delta_\alpha}{2}\right)\right\} \tag{2.11}$$

where $\Delta_\alpha = \chi^2(\alpha, df)$ being the α-th percentile of the χ^2-distribution.

The degree of freedom (df) is thereby the number of parameters for which the confidence interval is calculated. In the case of point-wise confidence intervals only one parameter is considered, therefore is $df = 1$, and for the simultaneous confidence intervals it is $df = n$.

2.2 Identifiability analysis

An important part of the parameter estimation is the identifiability analysis as shown by [2, 3]. Only for identifiable parameters a unique parameter fitting is possible, since the confidence intervals are boundless for non-identifiable parameters.
Identifiability can be divided into structural and practical identifiability. The structural one is a data-free quantity and for big systems infeasible to analyse. It is defined for the model observable $y(\tau, \theta)$ with $\theta \in \Omega \subseteq \mathbb{R}^n$ following [7].

Definition 2. ***Structural Identifiability***
A model is called global structural identifiable *for parameter θ_i, if there exists a $\tau \in \mathbb{R}^i$ s.t.:*

$$y(\tau, \theta) = y(\tau, \theta') \;\Rightarrow\; \theta = \theta'$$

for θ' varies at θ_i.
It is called local structural identifiable *if for almost any $\theta \in \Omega$ there exists an open neighbourhood $V(\theta)$ around θ, such that it is globally identifiable in $V(\theta)$.*

The structural identifiability is comparable to the injectivity of a mapping from the parameter set to the codomain of the likelihood. That means every input has a distinct output.
Due to the implemented approach in this thesis the practical identifiability is of bigger interest, for being the for modelling and prediction relevant one. Practical non-identifiability is defined in [2] in terms of shapes of the confidence intervals.

Definition 3. ***Practical Non-Identifiability***
A parameter estimate is called practically non-identifiable*, if its confidence interval has on at least one side, no finite bound.*

In many cases is the region of the parameter value initially bounded, for example by biological restrictions. Hence the confidence interval is finite, but the parameter still can be non-identifiable. In this case the parameter is also treated as non-identifiable, if the boundaries of the confidence interval coincide with the value restrictions.
It is of great interest to gain knowledge about the identifiability of parameters to detect uncertainties of the predictions. Useful for the identifiability analysis are the profile likelihoods [2, 3], which behave similar to ordinary likelihoods [8].

Definition 4. ***Profile Likelihood***
Be $\mathrm{L}(\theta)$ *the likelihood function of the parameter set* $\theta = \{\theta_1, \theta_2, ...\}$*, then the* profile likelihood for a single parameter $\mathrm{PL}(\theta_i)$ *is defined as*

$$\mathrm{PL}(\theta_i) = \max_{\theta_{j \neq i}} \mathrm{L}(\theta)$$

The profile likelihood can also be defined for a combination of parameters and are then called *prediction profile likelihood* [3]. A definition will be given later in section 3.4, where prediction profile likelihood are addressed in detail.
Continuing, from likelihood profiles the confidence intervals can be directly computed using the likelihood ratio.

Definition 5. ***Likelihood Ratio***
Be $\mathrm{L}(\hat{\theta})$ *the likelihood function at the MLE and* $\mathrm{PL}(\theta_i)$ *the profile likelihood function, then the* likelihood ratio R_i *is defined as*

$$R_i = \frac{\mathrm{PL}(\theta_i)}{\mathrm{L}(\hat{\theta})}$$

Since $\mathrm{PL}(\theta_i)$ is nothing else than the likelihood for a specific value of θ, the fraction in the confidence interval definition (2.11) can be replaced by the likelihood ratio R_i.

$$\left\{ \theta \in \mathbb{R} \,\middle|\, \frac{\mathrm{PL}(\theta_i)}{\mathrm{L}(\hat{\theta})} = R_i > \exp(-\frac{\Delta_\alpha}{2}) \right\}$$

where $\Delta_\alpha = \chi^2(\alpha, df)$ being the α-th percentile of the χ^2-distribution.
That means, the confidence interval contains all parameters for which the likelihood ratio is bigger than the threshold of the confidence level α.
Using the log-likelihood $\mathrm{l}(\hat{\theta})$ and profile log-likelihood $\mathrm{pl}(\theta_i)$, the likelihood ratio becomes:

$$\begin{aligned} R_i = \frac{\mathrm{PL}(\theta_i)}{\mathrm{L}(\hat{\theta})} &= \frac{\exp(\log(\mathrm{PL}(\theta_i)))}{\exp(\log(\mathrm{L}(\hat{\theta})))} = \\ &= \frac{\exp(\mathrm{pl}(\theta_i))}{\exp(\mathrm{l}(\hat{\theta}))} = \exp(\mathrm{pl}(\theta_i) - \mathrm{l}(\hat{\theta})) \end{aligned}$$

Let $\mathit{\Xi}_\alpha$ in the following be defined as:

$$\mathit{\Xi}_\alpha = \exp(-\frac{\Delta_\alpha}{2})$$

which is used throughout this thesis in the profile plots.

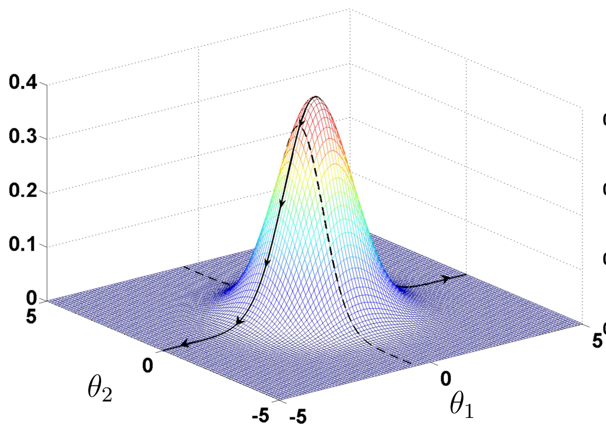
2.2.1 Existing methods for profile likelihood calculation

There are two main concepts to compute profile likelihoods. The first one is referred to as the *classical method* in this thesis. The method is straight forward using the definition,

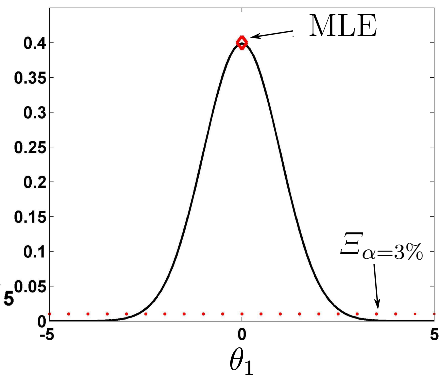
i.e. optimizing at every step.
For a visual explanation of this concept consider Figure 2.1. In 2.1a the two dimensional standard normal distribution is plotted. The starting point is the maximum of the standard normal distribution, which is the MLE. After increasing θ_1 by one step the log-likelihood will be optimized with θ_1 being constant (see Figure 2.1c). This maximum is added to the profile and the next step will be done until the boundaries are reached. Applying this to the forward and backward direction results in the profile of θ_1 (see Figure 2.1b).
This method is computationally demanding, because the optimization requires also repeated evaluations of the model. In most cases the analytical solution of the model is not available, therefore additionally to the optimization, a numerical solution of the model needs to be calculated. The high complexity of biomathematical models makes the calculation of numerical solutions as well as the repeated optimization a computationally demanding task to undertake.
An implementation example of the classical method is the function `computeProfiles` by Jan Hasenauer, which is for example used for the profile calculation in [4]. In contrast to the naive approach with constant step sizes for the profile computation, this algorithm is improved, among other things, due to an adaptive step size.
The function calculates the profiles of every single parameter of the model and requires basically the log-likelihood function, the MLE and parameter intervals. These provided intervals are determining boundaries of the parameter values, for example biological relevant limits. The profiles are calculated within these intervals by mainly following the definition of the profile likelihoods. Therefore the log-likelihood function is optimized at every step, using the optimization toolbox provided by MATLAB.
The difference to the naive approach of the classical method is, that the step size is not constant. The length of the step is calculated using a support function, which requires the update direction, the parameter constraints, the log-likelihood function and a boundary for the maximal allowed relative decrease of the likelihood ratio. The algorithm of the function then decides how wide it is possible to choose the step, so that the decrease of the likelihood ratio does not exceed this boundary.
In section 4 of this thesis this function is used to compute the benchmark for the assessment of the new implementation, concerning the number of evaluations of the model solution and the accuracy of the calculated profile.
The second method is, to treat the profile likelihood as an optimization with constraints and is in this thesis referred to as the integration based method. This idea was introduced by [1] and is the basis of the function implemented and tested in this Mastersthesis. The method will be introduced later in more detail (see section 3). A proof that the solutions of the classical method and this integration based approach are equivalent is given in [9].

2.2.2 Constrained optimization

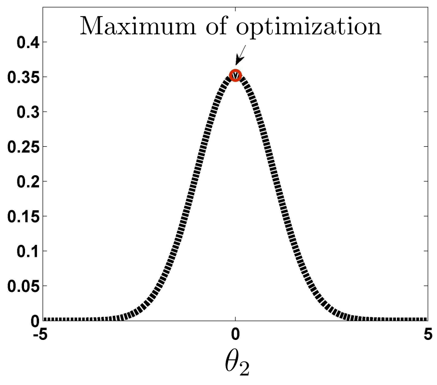
The restriction to fix one parameter (or a combination of several parameters) in the profile calculation can be seen as a constraint. Constraint optimization is generally treated in the context of Lagrange multipliers.



(a)
Standard normal distribution with profile for parameter θ_1 (solid line) and optimization step at $\theta_1 = -0.1$ (dashed line)



(b)
Profile for parameter θ_1 with threshold at 3% of the maximum



(c)
Optimization step at $\theta_1 = -0.1$

Figure 2.1: Profile calculation of two dimensional standard normal distribution with parameters $\theta = (\theta_1, \theta_2)$ for θ_1: Starting from the MLE, θ_1 is increased stepwise then the likelihood function is optimized with fixed θ_1. The maximum is then added to the profile and the next step is applied.

Lagrange multiplier

The approximation method of the profile likelihood from [1] which is investigated in this thesis is based on the Lagrange multiplier for optimization under constraints. The following Proposition from [10] gives a necessary requirement for an extreme point $\hat{\theta}$.

Proposition 1. ***Lagrange multiplier***
Be l *differentiable and* $g = (g_1, ..., g_k)$ *continuous differentiable on the open set* $\Omega \in \mathbb{R}^n$. *If* $\hat{\theta} \in \Omega$ *be an extreme point of* l *under the constraint* $g(\theta) = c$ *with* $c \in \mathbb{R}$ *and be the derivatives* $g_1'(\hat{\theta}), ..., g_k'(\hat{\theta})$ *linearly independent, then*

$$\exists \lambda_1, ..., \lambda_k \in \mathbb{R} \text{ so that } \mathrm{l}'(\hat{\theta}) = \sum_{i=1}^{k} \lambda_i g_i'(\hat{\theta})$$

To visualize the meaning of proposition 1 in the one dimensional case, consider Figure 2.2, where the contour lines from $\mathrm{l}(\theta)$ and $g(\theta)$ are shown.
The solid lines are representing the contour lines of the function $\mathrm{l}(\theta)$ and the dotted one is the contour line of $g(\theta) = c$. The point $\hat{\theta}$ is an extreme point (to be more concrete: maximal point) of $\mathrm{l}(\theta)$ s.t. $g(\theta) = c$, therefore moving along $g(\theta) = c$ away from $\hat{\theta}$ the value of $\mathrm{l}(\theta)$ is only decreasing. At the extreme point $\hat{\theta}$ the gradients of $\mathrm{l}(\theta)$ and $g(\theta)$ are proportional to each other, since the contour lines are parallel.

2.3 Introduction of examples

Throughout this thesis the integration based as well as the classical method will be applied to three model examples to evaluate the performance and investigate the effect of various input combinations. The examples are the standard normal distribution, as a test function of the method, the crystal growth model, which was already used in [1] to evaluate the integration based method, and a simple chemical reaction model, to verify the method in the presence of non-identifiable parameters.

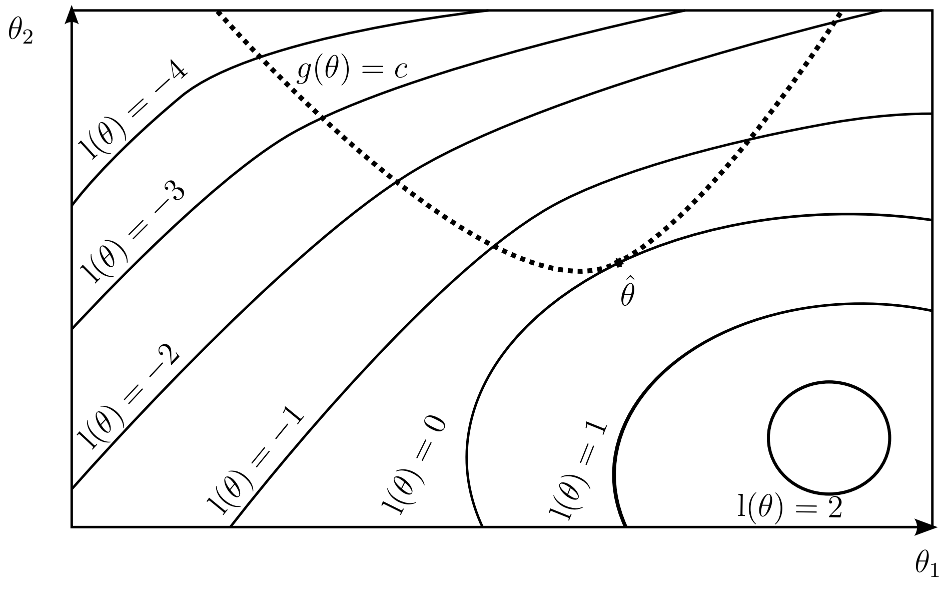


Figure 2.2: Sketch visualizes constraint optimization with Lagrange multipliers. The solid lines are representing the contour lines of the function $\mathrm{l}(\theta)$. The dotted line is the contour line of $g(\theta) = c$. The point $\hat{\theta}$ is an extreme point of $\mathrm{l}(\theta)$ s.t. $g(\theta) = c$, since the value of $\mathrm{l}(\theta)$ is only decreasing, if one is moving along $g(\theta) = c$ away from $\hat{\theta}$. At $\hat{\theta}$ the gradients of $\mathrm{l}(\theta)$ and $g(\theta)$ are proportional to each other, since the contour lines are parallel (idea of the sketch is taken from [10])

2.3.1 Standard normal distribution

For the first verification of the method, the implementation it is tested using the standard normal distribution in two dimensions:

$$\varphi_1(\theta) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}||\theta||^2} \text{ with } \theta \in \mathbb{R}^2$$

The variable θ represent here for the two dimensional parameter set $\theta = (\theta_1, \theta_2)$.
For this distribution the gradient and hessian can be computed analytically:

$$\nabla_\theta \varphi_1(\theta) = -\begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} \tag{2.12}$$

$$\nabla^2_\theta \varphi_1(\theta) = -\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.13}$$

This test function is a data free example for the profile computation. The function $\varphi_1(\theta)$ can directly be implemented representing the log-likelihood and its derivatives as the exact gradient and hessian of the log-likelihood. Due to the negative parameters of the standard normal distribution (SND) (the distribution is centred around the zero point), the logarithmic scale is not applied to this example function.

2.3.2 Crystal growth model

The following model was fitted to data with additive normal noise. Therefore the log-likelihood (2.3) and its derivatives (2.6) and (2.7) are used. The fitting of the data was done using a multi start algorithm also provided by Jan Hasenauer. The algorithm computes a candidate for the MLE by starting at various points in the parameter set and optimizing the log-likelihood.

Model: The crystal growth model (CGM) is an example from the paper [1]. It describes the growth of ice crystals with a non-linear regression model. The experimental data are the crystal masses y_i in nano grams after growing t_i seconds. The variance of the normal noise is $\sigma^2 = 1$.

$$\varphi_2(t, \theta) = \theta_1 \cdot t^{\theta_2} \tag{2.14}$$

with the parameter set $\theta = (\theta_1, \theta_2)$.
Differentiating with respect to θ gives:

$$\nabla_\theta \varphi_2(t, \theta) = \begin{pmatrix} t^{\theta_2} \\ \theta_1 \cdot t^{\theta_2} \cdot \log(t) \end{pmatrix} \tag{2.15}$$

$$\nabla^2_\theta \varphi_2(t, \theta) = \begin{pmatrix} 0 & t^{\theta_2} \cdot \log(t) \\ t^{\theta_2} \cdot \log(t) & \theta_1 \cdot t^{\theta_2} \cdot \log(t)^2 \end{pmatrix} \tag{2.16}$$

Inserting these functions into the log-likelihood function makes it possible to compute the profile with the exact gradient and hessian of the log-likelihood function. This exact solution can be compared to the profile, which is computed with approximations of the gradient and/or hessian.

2.3.3 Chemical reaction model

A simple example for a model with non-identifiable parameters is this model of a chemical reaction. The concentration x is produced with constant reaction rate k_1 and reduced dependent on the current concentration x and the rate k_2. Thereby is the concentration x only observable with the scaling factor s.

Model:

$$\varnothing \xrightarrow{k_1} x \tag{2.17}$$

$$x \xrightarrow{k_2} \varnothing \tag{2.18}$$

ODE System:

$$\begin{aligned} \dot{x} &= k_1 - k_2 \cdot x \\ y &= s \cdot x \\ \Rightarrow \dot{y} &= s \cdot k_1 - k_2 \cdot y \end{aligned} \tag{2.19}$$

with the parameter set $\theta = \{s,\ k_1,\ k_2\}$.

Analytical Solution: The inhomogeneous ODE (2.19) can analytically be solved up to a constant c:

$$y(t, \theta) \ = e^{-k_2(t+c)} + \tfrac{s \cdot k_1}{k_2} \tag{2.20}$$

There is no initial value problem given, but a reasonable initial point is: $x(0) = 0 = s \cdot y(0)$, because the production is independent from the current concentration $x(t, \theta)$.
Then the solution of the differential equation becomes:

$$y(t, \theta) \ = \tfrac{s \cdot k_1}{k_2} \cdot (1 - e^{-k_2 t}) \tag{2.21}$$

The parameters in the parameter set $\theta = (s,\ k_1,\ k_2)$ are treated in this example as being not measurable directly and can only be found by parameter estimation with data. However, for this model no experimental, but artificial data is used. The data is created using the analytical solution (see (2.21)) and adding normal noise with variance $\sigma^2 = 0.1^2$.

$$\tilde{y}_k \ = \ y(t_k, \theta^*) + \epsilon_k \tag{2.22}$$

$$\epsilon_k \ \sim \ \mathcal{N}(0, 0.1^2) \tag{2.23}$$

The true value θ^* for the artificial data is thereby chosen freely in biological relevant boundaries and is set to $\theta^* = [0.1, 2, 1]$ for the simulations.
Considering equation (2.21) it is obvious that the parameters s and k_1 are non-identifiable and the parameter k_2 is identifiable. The non-identifiability of s and k_1 is easy to see if different values for these parameters are compared.
Be $\theta_1 = (0.1, 1, k_2)$ and $\theta_2 = (1, 0.1, k_2)$. Then

$$y(t, \theta_1) = \frac{0.1}{k_2} \cdot (1 - e^{-k_2 t}) = y(t, \theta_2)$$

The product $s \cdot k_1$ has for both parameter sets the same value. In other words: the function $y(t,\theta)$ is not injective, which is necessary for identifiability. However this analytical approach to identify the identifiability of parameters is not always applicable. Especially, because the analytical solution $y(t,\theta)$ is rarely available.
In this model the analytical solution $y(t,\theta)$ is not only available, but continuous differentiable with respect to $\theta = \{s, k_1, k_2\}$. The derivatives are:

$$\nabla_\theta y(t,\theta) = \begin{pmatrix} \frac{k_1}{k_2} \cdot (1 - e^{-k_2 t}) \\ \frac{s}{k_2} \cdot (1 - e^{-k_2 t}) \\ -\frac{s \cdot k_1}{k_2^2} \cdot (1 - e^{-k_2 t}) + \frac{s \cdot k_1}{k_2} \cdot e^{-k_2 t} \cdot t \end{pmatrix} \tag{2.24}$$

$$\nabla_\theta^2 y(t,\theta) = \left(\begin{matrix} 0 & \frac{1}{k_2} \cdot (1 - e^{-k_2 t}) & \dots \\ \frac{1}{k_2} \cdot (1 - e^{-k_2 t}) & 0 & \\ -\frac{k_1}{k_2^2} \cdot (1 - e^{-k_2 t}) + \frac{k_1}{k_2} \cdot e^{-k_2 t} \cdot t & -\frac{s}{k_2^2} \cdot (1 - e^{-k_2 t}) + \frac{s}{k_2} \cdot e^{-k_2 t} \cdot t & \dots \end{matrix} \right. \tag{2.25}$$

$$\left. \begin{matrix} \dots & -\frac{k_1}{k_2^2} \cdot (1 - e^{-k_2 t}) + \frac{k_1}{k_2} \cdot e^{-k_2 t} \cdot t \\ \dots & -\frac{s}{k_2^2} \cdot (1 - e^{-k_2 t}) + \frac{s}{k_2} \cdot e^{-k_2 t} \cdot t \\ \dots & 2 \cdot \frac{s \cdot k_1}{k_2^3} \cdot (1 - e^{-k_2 t}) - 2 \cdot \frac{s \cdot k_1}{k_2^2} \cdot e^{-k_2 t} \cdot t - \frac{s \cdot k_1}{k_2} \cdot e^{-k_2 t} \cdot t^2 \end{matrix} \right) \tag{2.26}$$

The reparametrized chemical reaction model

In the previous section it became obvious by an analytical approach, that s and k_1 are both non-identifiable parameters. Furthermore they appear in the analytical solution $y(t,\theta)$ only as the product $s \cdot k_1$, which might raise the question, if the product of these parameters is identifiable even if the single parameters are not. To investigate this, the ansatz of the classical method is, to reparametrize the model, such that the resulting model contains a new parameter, which is equal to the desired parameter combination.
In this example define

$$(\chi_1, \chi_2, \chi_3) = (s \cdot k_1, k_1, k_2)$$

Analytical solution: Inserting the new parameters into the model solution (2.21) results in a system with only two parameters present:

$$y(t,\chi) = \frac{\chi_1}{\chi_3} \cdot (1 - e^{-\chi_3})$$

The reparametrization is now inverted. The result is a mapping from the new variables on the old ones. Therefore it is still possible to use the initial model. The mapping looks as follows:

$$\begin{aligned} k_2 &= \chi_3 \\ k_1 &= \chi_2 \\ \text{with } \chi_1 &= s \cdot k_1 \text{ and } \chi_2 = k_1 \\ \Rightarrow s &= \chi_1/k_1 = \chi_1/\chi_2 \\ \Rightarrow (s, k_1, k_2) &= (\chi_1/\chi_2, \chi_2, \chi_3) \end{aligned}$$

Because χ_2 does not appear in the system any more, it is arbitrary and will be set to 1 for all model calculations. Hence $(s, k_1, k_2) = \chi = (\chi_1, 1, \chi_3)$ will be used as the input for the model.

2.4 Method of performance evaluation

The implementation of the integration based method is supposed to enhance the profile calculation. To investigate this, the algorithm is evaluated concerning its performance for the profile likelihood calculation and compared to the classical method. The most important aspects of the evaluation are the accuracy and the computational demands.
The standard normal density (see 2.3.1), for example, is a simple function and easy to evaluate, but profiles are often calculated for models, which consist of a system of differential equations. The evaluation of those models is often a computational costly procedure, due to the numerical solving of the models.
Although the integration based method is supposed to be faster than the classical method and the actual time the algorithm needs for the whole profile calculation is finally the value, which interests the user, the time is not measured and compared in this thesis. That is because the length of time changes depending on the machine the algorithm is running on and how long the evaluation of the model itself needs. Already the latter one is various for the considered models in this thesis.
To gain a comparable quantity the number of evaluations (NoE) of the cost function is the figure of merit of the profile computation. This value indicates how often the model system needs to be evaluated. This value should be as little as possible to save computation time.
For the evaluation procedure the number of evaluations (NoE) is plotted against the *error to benchmark*. This error reflects the variance of the results from the classical method, which is the benchmark.
To calculate the error, the approximated profile as well as the benchmark profile were interpolated on a grid of 1001 points, resulting in two vectors of dimension 1001: $\mathrm{pl}_{\mathrm{appr}}$ and $\mathrm{pl}_{\mathrm{bench}}$ for the approximated profile and the benchmark profile respectively.
Then the error is defined as:

$$\varepsilon_{2\mathrm{bench}} := \| \mathrm{pl}_{\mathrm{appr}} - \mathrm{pl}_{\mathrm{bench}} \|_2$$

which is the difference of the elements in both vectors in the l2-norm.

In the examples it occurred for some inputs that the algorithm was not able to calculate the profile and the resulting output was only one point. But for one point the interpolation cannot be applied, so that in this case the error was set to an arbitrary but relatively big value: $\varepsilon_{2\mathrm{bench}} = 42$.

Chapter 3

Integration based profile likelihood calculation

In their integration based profile likelihood calculation, Chen and Jennrich [1] make use of the Lagrange multiplier (see (2.2.2)) for the optimization. For this purpose, the real valued *parameter function* $g(\theta) \in \mathbb{R}$ is defined. This function determines the parameters for which the profile should be computed of. For example, if $g(\theta) = \theta_1$ then the profile will be calculated along the parameter θ_1.

3.1 Method derivation

Recalling the definition of profile likelihoods (see definition 4), the likelihood is optimized for one constant parameter over all the other parameters. Therefore, the constraint for the optimization is $g(\theta) = \theta_i = c$ with $c \in \mathbb{R}$ constant.
The parameter set optimizing $\mathrm{l}(\theta)$ under constraint depends on the constant and can therefore be treated as a function of c: $\theta(c)$. This function defines the likelihood profile with respect to the parameter function. The same argument can be applied to the Lagrange-multiplier: $\lambda(c)$. To save space these functions are written in the following equations with the shorter index notation $\theta_c := \theta(c)$ and $\lambda_c := \lambda(c)$.
Inserted into the Lagrange multiplier formula (see proposition 1), the constraint optimization system looks as follows:

$$\nabla_\theta \mathrm{l}(\theta_c) + \lambda_c \cdot \nabla_\theta g(\theta_c) = 0 \tag{3.1}$$

$$\text{s.t. } g(\theta_c) = c \tag{3.2}$$

with $c \in \mathbb{R}$ and λ_c be the Lagrange-multiplier.
Differentiating this system of equations with respect to c gives then:

$$\begin{pmatrix} \nabla_\theta^2 \mathrm{l}(\theta_c) \cdot \dot{\theta}_c + \lambda_c \cdot \nabla_\theta^2 g(\theta_c) \cdot \dot{\theta}_c + \dot{\lambda}_c \cdot \nabla_\theta g(\theta_c) \\ \nabla_\theta g(\theta_c) \cdot \dot{\theta}_c \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{3.3}$$

The dot over $\dot{\theta}_c$ and $\dot{\lambda}_c$ stands for the derivation with respect to c, which is a one-dimensional differentiation only.

The resulting equation system is a DAE of index 1 in the semi-explicit form [11]. To solve this differential equation numerically, it is useful to rewrite it into a matrix multiplication.

$$\Rightarrow \underbrace{\begin{pmatrix} \nabla_\theta^2 \mathrm{l}(\theta_c) + \lambda_c \cdot \nabla_\theta^2 g(\theta_c) & \nabla_\theta g(\theta_c) \\ \nabla_\theta g(\theta_c)^T & 0 \end{pmatrix}}_{:=A(\theta_c,\lambda_c)} \cdot \begin{pmatrix} \dot{\theta}_c \\ \dot{\lambda}_c \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{3.4}$$

where $A(\theta_c, \lambda_c)$ is called the *mass matrix* of the DAE.
In this equation also the gradient and hessian of the parameter function are used. In the simple case of $g(\theta) = \theta_i$ the derivatives read as follows:

$$\nabla_\theta g(\theta) = e_i \tag{3.5}$$

$$\nabla_\theta^2 g(\theta) = 0_{n\times n} \tag{3.6}$$

with e_i being the i-th unit vector and $0_{n\times n}$ the $n \times n$-zero matrix.
Now, (3.4) can be written in the compact form:

$$A(\zeta) \cdot \dot{\zeta} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \text{ with } \zeta = \begin{pmatrix} \theta_c \\ \lambda_c \end{pmatrix} \text{ and its derivative } \dot{\zeta} \tag{3.7}$$

The solution of this DAE is ζ, which contains with θ_c the parameters along the profile. Inserting them into the log-likelihood results in the profile likelihood. Taking the profile likelihoods into account statements about identifiability can be made.

Direction parameter s

The differential equation is solved starting from the MLE in the forward direction and in the backward direction. For numerical reasons, the differential equation needs to be adjusted for solving the backward direction. This requires only to add a direction parameter s to $\nabla_\theta g(\theta)$.

$$g(\theta_c) = c$$

$\Rightarrow$ Differentiating with respect to c:

$$\nabla_\theta g(\theta_c) \cdot \dot{\theta}_c = \dot{c}$$

For the negative direction is $\dot{c} = -1$, because for decreasing c the differentiation is inverted.

$$\begin{aligned} \Rightarrow \nabla_\theta g(\theta_c) \cdot \dot{\theta}_c &= -1 \\ -\nabla_\theta g(\theta_c) \cdot \dot{\theta}_c &= 1 \\ s \cdot \nabla_\theta g(\theta_c) \cdot \dot{\theta}_c &= 1 \text{ with } s = -1 \end{aligned}$$

3.2 Correction term

Due to the fact that exact gradients and hessians of the considered functions may not be available in all cases, Chen and Jennrich [1] introduced a correction term to reduce the numerical error in the presence of approximations. The error is in [1] defined as the difference of the profile calculated with approximated derivatives and the exact one.

$$\| \theta_c^{\text{appr.}} - \theta_c^{\text{exact}} \| < \epsilon \text{ for } \epsilon > 0$$

The introduced correction term is substituted as the right hand side of (3.4), which becomes:

$$\begin{pmatrix} \nabla_\theta^2 \text{l}(\theta_c) + \lambda_c \cdot \nabla_\theta^2 g(\theta_c) & \nabla_\theta g(\theta_c) \\ \nabla_\theta g(\theta_c)^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \dot\theta_c \\ \dot\lambda_c \end{pmatrix} = \begin{pmatrix} -\gamma \nabla_\theta \text{l}(\theta_c) \\ 1 \end{pmatrix} \tag{3.8}$$

with the factor $\gamma \in \mathbb{R}_+^0$ determining the influence of the correction term.
In [1] a proof is given that this correction term reduces the error in the calculation of profile likelihoods in the presence of an approximated hessian. One assumption of the proof is, that

$$\nabla_\theta^2 \text{l}(\hat\theta) + \lambda \cdot \nabla_\theta^2 g(\hat\theta)$$

is negative definite. However, this is not always the case for systems with non-identifiable parameters, since it occurs, that the MLE $\hat\theta$ is not even near the true value θ^*. That is because the MLE has to be found using optimization algorithms. But it appears that the log-likelihood does not have a unique maximum, if non-identifiable parameters are present. This results in arbitrary parameter values for the MLE.
Furthermore, increasing the factor γ produces a stiff differential equation. This is shown in section 4.2.3, by applying the Euler method, which is sensitive for stiff systems. This stiffness enhances in some cases the computation time but not the accuracy. The correction term can moreover cause other undesired behaviours of the algorithm, as for example a turning of the solving direction (see Figure 5.7).
Therefore, it is suggested to disable the correction term for a faster and reliable computation and to use an accurate hessian approximation instead. Nevertheless will this correction term from now on be part of the differential equation system of the integration based method, since in (3.8) γ can also be set to zero, which results again in (3.4).

3.3 Optimization problem in the logarithmic scale

There are two ways to apply the logarithmic scale to the optimization problem 3.2.
The first one is to define the constraint in terms of the initial problem and then apply the logarithmic scale, that means that also the parameter function is changed in terms of ξ. The optimization problem is then written as:

$$\nabla_\xi \text{l}(\xi) + \lambda \cdot \nabla_\xi g(\xi) = 0 \tag{3.9}$$

$$\text{s.t. } g(\xi) = \exp(\xi_i) = c \tag{3.10}$$

where ξ is the logarithmic parameter with: $\xi = \log(\theta)$.

The other possibility is to use the logarithmic scale for the optimized function (i.e. the log-likelihood) and then define the constraint in the log-parameters.
In this case the optimization problem reads as:

$$\nabla_\xi \mathrm{l}(\xi) + \lambda \cdot \nabla_\xi g(\xi) = 0 \tag{3.11}$$
$$\text{s.t. } g(\xi) = \xi_i = c' \tag{3.12}$$

with $c' \in \mathbb{R}$ being constant.
It should be noted, that the parameter function is in this case a linear function.
These two possibilities will later in this thesis be referred to as the first and second approach of the logarithmic scale.
Both approaches are relevant, since they actually reflect a different scaling of the parameters in the profile calculation regarding the optimization itself and not only the solving of the resulting differential equation. In the first approach the optimization is done in terms of the initial parameters, while the second approach is to optimize directly in terms of the logarithmic parameters. Applications often show a better performance with logarithmic parameters and it will be investigated if this behaviour also occurs with the connection to profile likelihoods.

3.4 Non-linear parameter function

In [1] only cases with a linear parameter function $g(\theta)$ were considered. This is the case if the profile is computed for a single parameter (i.e. $g(\theta) = \theta_1$). The hessian of the parameter function is then a zero matrix and the equation (3.8) reduces to:

$$\begin{pmatrix} \nabla^2_\theta \mathrm{l}(\theta_c) & \nabla_\theta g(\theta_c) \\ \nabla_\theta g(\theta_c)^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \dot{\theta}_c \\ \dot{\lambda}_c \end{pmatrix} = \begin{pmatrix} -\gamma \nabla_\theta \mathrm{l}(\theta_c) \\ 1 \end{pmatrix} \tag{3.13}$$

But this simplification cannot be used in general, because the parameter function is $g(\xi) = \exp(\xi_i)$ using the first approach of the logarithmic scale (see 3.10). Then the hessian $\nabla^2_\xi g(\xi_c)$ is no zero matrix:

$$\nabla_\xi g(\xi) = e_i \cdot \exp(\xi_i) \tag{3.14}$$
$$\nabla^2_\xi g(\xi) = \Lambda_{ii} \cdot \exp(\xi) \tag{3.15}$$

with Λ_{ii} being the matrix with entry $\varpi_{ii} = 1$ and all other entries equal 0.
Since the log-scale is applied to the majority of the example models in this thesis the simplification from [1] cannot be used here.
However it can still be applied for the second approach of the logarithmic scale (see 3.12). In this case the parameter function is linear and the simplification can still be applied.

But there is another reason why non-linear parameter functions have to be considered. If two or more parameters are non-identifiable it is also interesting to know if a combination of parameters is identifiable. For the integration based method this is simply accomplished by defining a new parameter function as this combination (e.g. $g(\theta) = \theta_1 \cdot \theta_2$). In this case the definition of profile likelihoods is generalized to the *prediction profile likelihood* [3].

$$\mathrm{PPL}(z) = \max_{\theta \in \{\theta | g(\theta) = z\}} \mathrm{L}(\theta)$$

The prediction profile likelihood can also be calculated with the classical method using stepwise optimization. In this case a reparametrization is necessary [9].
Concerning this, the method to optimize under constraints is favourable to the classical one, because in complex models it is a crucial advantage to avoid the elaborate reparametrization to investigate identifiability of parameter combinations. The integration based method does not need a reparametrization and therefore relieves the user from the sometimes complicated calculation and prevents thereby probably occurring mistakes.

Example calculation of prediction profile likelihoods

For example, in the model of chemical reactions (introduced in 2.3.3) the reparametrization was shown to investigate the profile likelihood for the combination of the non-identifiable parameters s and k_1. Using the integration based method the reparametrization can be neglected and the parameter function simply defined as:

$$g(\theta) := \chi_1 = s \cdot k_1 \tag{3.16}$$

However, to use the parameter function in the integration based method, it has to be differentiated with respect to θ and reads as:

$$\nabla_\theta g(\theta) = \begin{pmatrix} k_1 \\ s \\ 0 \end{pmatrix} \tag{3.17}$$

$$\nabla_\theta^2 g(\theta) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.18}$$

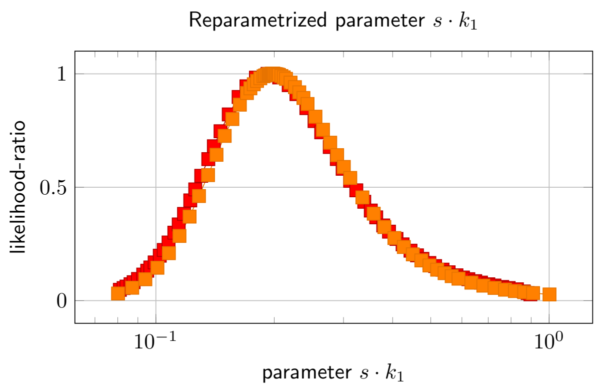
The profile likelihood can now be calculated with the integration based method and is shown in Figure 3.1, where also the profiles of the reparametrization for $\chi_1 = s \cdot k_1$ and $\chi_3 = k_2$ are shown. The profiles match each other nicely, with a considerably small error and the profile calculated with the integration based method does additionally decrease the NoE.

If the logarithmic scale is applied for the combination of parameters, the parameter function changes in the first approach of the logarithmic scale (see 3.10), with $\xi = (\log(s), \log(k_1). \log(k_2))$ simply to:

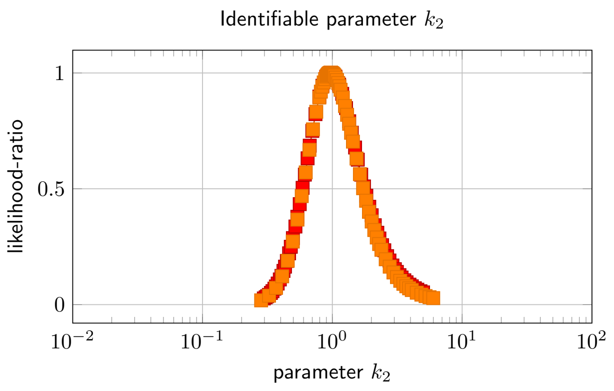
$$g(\xi) := \exp(\xi_1) \cdot \exp(\xi_2) = s \cdot k_1 \tag{3.19}$$

$$\nabla_\xi g(\xi) = \begin{pmatrix} \exp(\xi_1) \cdot \exp(\xi_2) \\ \exp(\xi_1) \cdot \exp(\xi_2) \\ 0 \end{pmatrix} \tag{3.20}$$

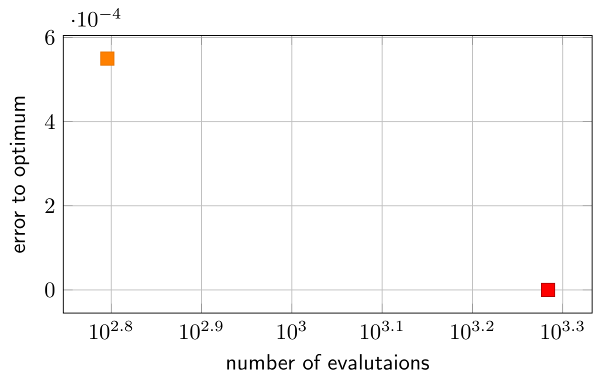
$$\nabla_\xi^2 g(\xi) = \begin{pmatrix} \exp(\xi_1) \cdot \exp(\xi_2) & \exp(\xi_1) \cdot \exp(\xi_2) & 0 \\ \exp(\xi_1) \cdot \exp(\xi_2) & \exp(\xi_1) \cdot \exp(\xi_2) & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.21}$$



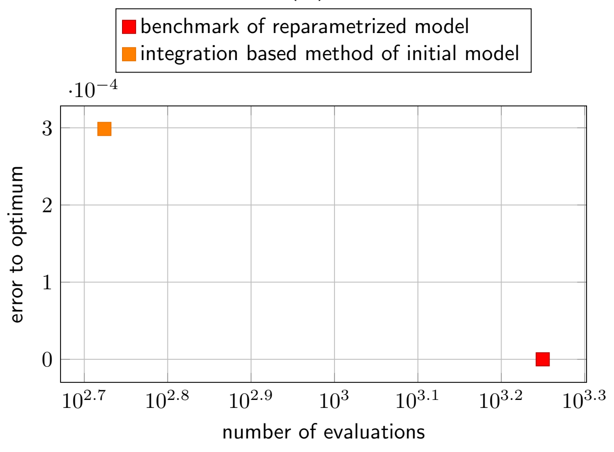
(a)



(b)



(c)



(d)

Figure 3.1: Profiles of the reparametrized chemical reaction model for the parameters $\chi_1 = s \cdot k_1$ and $\chi_3 = k_2$, calculated with `computeProfiles` and the profiles calculated with `computeApproximatedProfiles` using IDAS (default options) and the parameter function $g(\theta) = s \cdot k_1$ for the left profile and $g(\theta) = k_2$ for the right one.

But for the second approach of the logarithmic scale (see 3.12) the constraint must be rewritten in terms of the logarithmic parameters, therefore it becomes

$$g(\xi) := \log(s \cdot k_1) \tag{3.22}$$
$$g(\xi) = \log(s) + \log(k_1) \tag{3.23}$$
$$= \xi_1 + \xi_2 \tag{3.24}$$

and the derivatives will be calculated accordingly with respect to ξ.

In the displayed case, the reparametrization was not very difficult, but if more complicated models and combinations of parameters are to be considered, the set up of the parameter function is a considerably simpler and more direct approach than using reparametrization, because only the parameter function has to be changed to calculate the desired profile. However, for non-linear $g(\theta)$ there are cases where both methods are not appropriate to compute profile likelihoods and confidence intervals. A possible alternative to find a confidence interval, stated by [9], is then to optimizing $g(\theta)$ and use its maximal and minimal point respectively for the interval. But the considered examples in this thesis do not fall within these cases and the used approaches are therefore sufficient.

Chapter 4

Method assessment

The algorithm to calculate profile likelihoods using the integration based method is implemented in MATLAB. This implementation is used to assess the integration based method concerning its performance for the profile likelihood calculation compared to the classical method.
The implemented method is supposed to accelerate the calculation. Thus the most important aspects of the evaluation are the accuracy and the number of evaluations (NoE). The algorithm accepts various inputs, ranging from necessarily provided functions to options, which adjust the algorithm according to preferences of the user. There are too many to test every possible combination. Therefore, only the most interesting ones will be in focus of the evaluation.
At first, the algorithm of the implementation is introduced, followed by the discussion of the different solvers and the evaluation of various approximations of the derivatives.
The structure is thereby that the methods are introduced and subsequently assessed considering the two example models, which are standard normal distribution (SND) and crystal growth model (CGM) (introduced in section 2.3). Finally the introduced methods are evaluated using a model, which includes non-identifiable parameters.

4.1 Implementation details

Before assessing the implementation, the single steps of the algorithm will be explained in more detail.
The illustration is done at a sketch of the implemented function in pseudo code and is split into three parts for reasons of clarity and comprehensibility. The first part is the initialization of the required derivatives and initial condition. The second one refers to the solving of the differential equation and the third one concludes the algorithm with the calculation of the output.

At first consider the algorithm 1, which visualizes the initialization part of the algorithm. The function requires the MLE, for being the initial point of the differential equation, and intervals to determine the maximum and minimum values of the parameters. The log-likelihood function as well as the parameter function must be provided by the user, with or without the derivatives. The user can moreover choose between the implemented solver and decide, which approximation method should be applied for the derivatives.

```
// Initialization
Input: MLE $\hat{\theta}$,
       parameter intervals,
       log-likelihood function,
       parameter function,
       solver, approximation options, $\gamma$
// Approximations of derivatives
if gradient not provided then
    approximate gradient using approximation options for gradient;
end
if hessian not provided then
    approximate hessian using approximation options for hessian;
end
// Initial conditions
calculate $\lambda_0$ s.t. $\nabla_\theta \mathrm{l}(\hat{\theta}) + \lambda_0 \cdot \nabla_\theta g(\hat{\theta}) = 0$;
set $\zeta_0 = (\hat{\theta}, \lambda_0)^T$;
set
    $A(\zeta) = \begin{pmatrix} \nabla^2_\theta \mathrm{l}(\theta_c) + \lambda_c \cdot \nabla^2_\theta g(\theta_c) & \nabla_\theta g(\theta_c) \\ \nabla_\theta g(\theta_c)^T & 0 \end{pmatrix}$
```

Algorithm 1: Sketch of the implemented algorithm, Part I: Initialization of the required inputs. The approximation of the derivatives is applied, if they are not provided by the functions. The missing initial condition λ_0 is calculated and the mass matrix $A(\zeta)$ set.

The MLE $\hat{\theta}$ is the upper part of the initial condition ζ_0 of the differential equation. The missing initial condition λ_0 is calculated using the formula of Lagrange multiplier and then inserted into $\zeta_0 = (\hat{\theta}, \lambda_0)^T$. The mass matrix $A(\zeta)$ is set with the provided or approximated derivatives.

In the next part (consider algorithm 2) the solving of the ODE or the DAE system is applied, depending on the chosen solver. Additional to the initial conditions it is necessary for the IDAS solver to calculate the initial value of the derivative $\dot{\zeta}$. The differential equation is then solved within the parameter interval of the parameter concerned in the profile calculation.

At the end of the function (consider algorithm 3) the log-likelihood function is evaluated at the parameter values of the profile. The likelihood ratio is also calculated at every point of the profile and returned by the function together with the parameter values and the log-likelihood evaluations.

4.2 Solver evaluation

As shown in algorithm 2, there are two basic approaches to solve (3.7). The solution can directly be found using a numerical DAE Solver or it can be rearranged by multiplying

```
// Solve differential equation
if DAE Solver then
    if solver is IDAS then
        // Initial condition for IDAS
        calculate:
            $\dot{\zeta}_0 = A(\zeta_0)^{-1} \cdot \begin{pmatrix} -\gamma\nabla_\theta \mathrm{l}(\theta_c) \\ 1 \end{pmatrix}$
    end
    solve:
        $A(\zeta) \cdot \dot{\zeta} = \begin{pmatrix} -\gamma\nabla_\theta \mathrm{l}(\theta_c) \\ 1 \end{pmatrix}$
    within parameter interval;
else if ODE Solver then
    solve:
        $\dot{\zeta} = A(\zeta)^{-1} \cdot \begin{pmatrix} -\gamma\nabla_\theta \mathrm{l}(\theta_c) \\ 1 \end{pmatrix}$
    within parameter interval;
end
```

Algorithm 2: Sketch of the implemented algorithm, Part II: Depending on the chosen solver is the ordinary differential equation or the algebraic differential equation solved in the provided parameter interval. For the solver IDAS, the initial condition for $\dot{\zeta}$ must be calculated, too.

```
// Calculate output
for j={1,...,k} do
    calculate value of log-likelihood $\mathrm{l}(\theta^j)$;
    calculate likelihood ratio $R_j = \frac{\mathrm{l}(\theta^j)}{\mathrm{l}(\hat{\theta})}$;
end
return parameters along profile $\{\theta^1, \theta^2, ..., \theta^k\}$,
       log-likelihood along profile $\mathrm{l}(\theta^j)\ \forall j \in \{1, ..., k\}$,
       likelihood ratio along profile $R_j\ \forall j \in \{1, ..., k\}$
```

Algorithm 3: Sketch of the implemented algorithm, Part III: At the end the log-likelihood function is evaluated at the parameter values of the profile and the likelihood ratio is calculated.

with the inverse matrix of $A(\zeta)$, as it was done in [1].

$$\dot{\zeta} \;=\; A(\zeta)^{-1} \cdot \begin{pmatrix} -\gamma \nabla_\theta \mathrm{l}(\theta_c) \\ 1 \end{pmatrix} \tag{4.1}$$

The result is a system of ODEs. Obviously this approach is only applicable if the mass matrix $A(\zeta)$ is non-singular.
The crystal growth model used in [1] (see 2.3.2) only included identifiable parameters and the hessian was non-singular, so that there is no problem in inverting the mass matrix $A(\zeta)$. But it can happen that the hessian becomes singular, since the hessian is dependent on the parameter values. Furthermore the scale of the parameter values plays a decisive role for the singularity of the hessian.
The question is, if this is also the case for the mass matrix $A(\zeta)$ and if it is still possible to compute the profiles. Furthermore it is if interest if then the ansatz with the DAEs solver is superior to the method which requires the inversion of the mass matrix.

4.2.1 Concerning the singularity of the hessian

It was already noted that it is possible that the hessian of the log-likelihood is singular. In the example of the chemical model (see 2.3.3) it will be shown that under certain circumstances the hessian becomes singular and it will be investigated if this singularity does interfere with the profile calculation.
In the following the hessian of the log-likelihood function in the logarithmic scale will be considered, see (2.10). Recalling the logarithmic scale, the parameter set changes to:

$$(\xi_1, \xi_2, \xi_3) = (\log(s), \log(k_1), \log(k_2))$$

The parameter transformation $\exp(\xi_i)$ will be substituted with θ_i to shorten the equation, but preserve the difference to the initial parameters (i.e. remember to apply the chain rule for the parameter transformation).

$$(\theta_1, \theta_2, \theta_3) = (\exp(\xi_1), \exp(\xi_2), \exp(\xi_3))$$

Inserting the model solution into the formula for the hessian of the log-likelihood in the logarithmic scale results in:

$$\begin{aligned}
\nabla_\xi^2 \mathrm{l}(\theta) = \frac{1}{\sigma^2}\sum_{k=1}^{m} \Upsilon_k & \left(\left(\begin{matrix} 0 & \frac{\theta_1}{\theta_3}(1-e^{-\theta_3 t})\theta_2 & \theta_1(\frac{-\theta_2}{\theta_3^2}(1-e^{-\theta_3 t}) + \frac{\theta_2}{\theta_3}e^{-\theta_3 t}t)\theta_3 \\ \frac{\theta_2}{\theta_3}(1-e^{-\theta_3 t})\theta_1 & 0 & \theta_2(\frac{-\theta_1}{\theta_3^2}(1-e^{-\theta_3 t}) + \frac{\theta_1}{\theta_3}e^{-\theta_3 t}t)\theta_3 \\ \dots & \dots & \dots \end{matrix} \right. \right) \\
& + \left. \begin{pmatrix} \theta_1\frac{\theta_2}{\theta_3}(1-e^{-\theta_3 t}) & 0 & 0 \\ 0 & \theta_2\frac{\theta_1}{\theta_3}(1-e^{-\theta_3 t}) & 0 \\ 0 & 0 & \dots \end{pmatrix} \right) \\
& - \begin{pmatrix} \frac{\theta_2^2}{\theta_3}(1-e^{-\theta_3 t}) & \frac{\theta_2\theta_1}{\theta_3}(1-e^{-\theta_3 t}) & \frac{\theta_2}{\theta_3}(1-e^{-\theta_3 t})(\frac{-\theta_1\theta_2}{\theta_3^2}(1-e^{-\theta_3 t}) + \frac{\theta_1\theta_2}{\theta_3}e^{-\theta_3 t}t) \\ \frac{\theta_2\theta_1}{\theta_3}(1-e^{-\theta_3 t}) & \frac{\theta_1^2}{\theta_3}(1-e^{-\theta_3 t}) & \frac{\theta_1}{\theta_3}(1-e^{-\theta_3 t})(\frac{-\theta_1\theta_2}{\theta_3^2}(1-e^{-\theta_3 t}) + \frac{\theta_1\theta_2}{\theta_3}e^{-\theta_3 t}t) \\ \dots & \dots & \dots \end{pmatrix}
\end{aligned} \tag{4.2}$$

with $\Upsilon_k = \tilde{y}_k - y(\tau_k, \exp(\xi))$, which is a scalar.
The last row of the matrices is not of interest here and therefore neglected for simplification.
Summing all up, the hessian becomes:

$$\nabla^2_\xi \mathrm{l}(\theta) = \frac{1}{\sigma^2} \cdot ...$$

$$\left(\begin{matrix} \sum\limits_{k=1}^{m}\left(\Upsilon_k\theta_1\frac{\theta_2}{\theta_3}(1-e^{-\theta_3 t}) - \frac{\theta_2^2}{\theta_3}(1-e^{-\theta_3 t})\right) & \sum\limits_{k=1}^{m}\left(\Upsilon_k\theta_2\frac{\theta_1}{\theta_3}(1-e^{-\theta_3 t}) - \frac{\theta_2\theta_1}{\theta_3}(1-e^{-\theta_3 t})\right) & ... \\ \sum\limits_{k=1}^{m}\left(\Upsilon_k\theta_1\frac{\theta_2}{\theta_3}(1-e^{-\theta_3 t}) - \frac{\theta_2\theta_1}{\theta_3}(1-e^{-\theta_3 t})\right) & \sum\limits_{k=1}^{m}\left(\Upsilon_k\theta_2\frac{\theta_1}{\theta_3}(1-e^{-\theta_3 t}) - \frac{\theta_1^2}{\theta_3}(1-e^{-\theta_3 t})\right) & ... \\ \dots & \dots & ... \end{matrix}\right.$$

$$\left.\begin{matrix} ... & \sum\limits_{k=1}^{m}\left(\Upsilon_k\theta_1(\frac{-\theta_2}{\theta_3^2}(1-e^{-\theta_3 t}) + \frac{\theta_2}{\theta_3}e^{-\theta_3 t}t)\theta_3 - \frac{\theta_2}{\theta_3}(1-e^{-\theta_3 t})(\frac{-\theta_1\theta_2}{\theta_3^2}(1-e^{-\theta_3 t}) + \frac{\theta_1\theta_2}{\theta_3}e^{-\theta_3 t}t)\right) \\ ... & \sum\limits_{k=1}^{m}\left(\Upsilon_k\theta_2(\frac{-\theta_1}{\theta_3^2}(1-e^{-\theta_3 t}) + \frac{\theta_1}{\theta_3}e^{-\theta_3 t}t)\theta_3 - \frac{\theta_1}{\theta_3}(1-e^{-\theta_3 t})(\frac{-\theta_1\theta_2}{\theta_3^2}(1-e^{-\theta_3 t}) + \frac{\theta_1\theta_2}{\theta_3}e^{-\theta_3 t}t)\right) \\ ... & \dots \end{matrix}\right) \tag{4.3}$$

It depends – among others – on the parameters and the values of Υ_k, if the matrix (4.3) is singular. But it is obvious that it is singular, if $\theta_1 = \theta_2$, because in this case the first two rows of the matrix are equal and therefore linearly dependent.
Moreover in the numerical case the limited machine precision leads to singular matrices. The following numerical example with inserted parameter values will show that even if $\theta_1 \neq \theta_2$ the hessian can be treated as singular. Subsequently it will be shown how the singularity of the hessian can be resolved.
Let the MLE be

$$\hat{\theta} = \begin{pmatrix} -0.8771 \\ -0.7420 \\ -0.0286 \end{pmatrix}$$

In this case is $\theta_1 = -0.8771 \neq \theta_2 = -0.7420$.
The hessian of the log-likelihood function with insertet MLE is approximately:

$$\nabla^2_\theta \mathrm{l}(\hat{\theta}) \approx \begin{pmatrix} -99.9630 & -99.9630 & 80.4643 \\ -99.9630 & -99.9630 & 80.4643 \\ 80.4643 & 80.4643 & -69.9919 \end{pmatrix}$$

The hessian has a condition number $\mathrm{cond}(\nabla^2_\theta \mathrm{l}(\hat{\theta})) = 2.5341 \times 10^{-17}$, which is smaller than 10^{-15} and is therefore numerically treated as singular. This is obvious, since the first row and the second row of the hessian are equal keeping the rounding error in mind. In other words the hessian is numerical singular even if the parameters θ_1 and θ_2 are not equal.
As the next step, the hessian will be inserted into the mass matrices (recall equation (3.4)) with $\hat{\zeta} = (\hat{\theta}, \lambda_0)^T$ and λ_0 be the solution of:

$$\nabla_\theta \mathrm{l}(\hat{\theta}) + \lambda_0 \cdot \nabla_\theta g(\hat{\theta}) = 0$$

In the case of the profile calculation for s and k_2 these look as follows:

Mass matrix $A(\hat{\zeta})$ for k_2 at the MLE:

$$A(\hat{\zeta}) \approx \begin{pmatrix} -99.9630 & -99.9630 & 80.4643 & 0 \\ -99.9630 & -99.9630 & 80.4643 & 0 \\ 80.4643 & 80.4643 & -69.9919 & 0.9718 \\ 0 & 0 & 0.9718 & 0 \end{pmatrix}$$

with condition number: $\text{cond}(A(\hat{\zeta})) = 2.5341 \times 10^{-17} < 10^{-15}$.

Mass matrix $A(\hat{\zeta})$ for s at the MLE:

$$A(\hat{\zeta}) \approx \begin{pmatrix} -99.9630 & -99.9630 & 80.4643 & 0.4160 \\ -99.9630 & -99.9630 & 80.4643 & 0 \\ 80.4643 & 80.4643 & -69.9919 & 0 \\ 0.4160 & 0 & 0 & 0 \end{pmatrix}$$

with condition number: $\text{cond}(A(\hat{\zeta})) = 7.4068 \times 10^{-4} > 10^{-15}$.
It is interesting that the mass matrix for the identifiable parameter is singular, while it is non-singular for the non-identifiable parameter. That is because, concerning the mass matrix of s, the gradient of the parameter function adds a value in the row, which is differentiated with respect to s. This results in a row that is no longer linearly dependent to the second one and the matrix is invertible. The same applies of course for the parameter k_1 (not shown). For k_2 the gradient of the parameter function only adds a value in the row which is already linearly independent from the other ones and therefore the mass matrix is, like the hessian, singular.
However, to still be able to invert the mass matrix and use the ODE based integrations one can add a term to the singular mass matrix like:

$$A'(\hat{\zeta}) = A(\hat{\zeta}) + \epsilon \cdot I_{n+1} \tag{4.4}$$

with $\epsilon > 0$, but relatively small (for example $\epsilon = 10^{-4}$) and I_{n+1} being the identity matrix of dimension $n+1$.
This resolves the singularity of the mass matrix without being too imprecise in the calculation of the profiles, because it is only a comparably small change of the matrix.
For the example with k_2 this results in a new mass matrix:

$$A'(\hat{\zeta}) \approx \begin{pmatrix} -99.9629 & -99.9630 & 80.4643 & 0 \\ -99.9630 & -99.9629 & 80.4643 & 0 \\ 80.4643 & 80.4643 & -69.9918 & 0.9718 \\ 0 & 0 & 0.9718 & 0.0001 \end{pmatrix}$$

with condition number: $\text{cond}(A'(\hat{\zeta})) = 9.1664 \times 10^{-7} > 10^{-15}$, which is no longer treated as singular and therefore the profile calculation can also be applied using the ODE based solver.

4.2.2 DAE based integration

As mentioned before, the system (3.7) can be solved directly using DAE solvers. That means the solver searches for a solution to the DAE system. This system does not

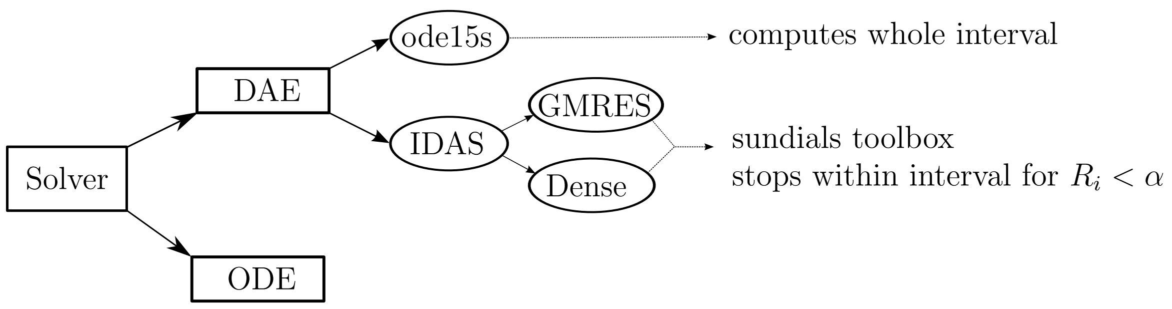


Figure 4.1: Possible DAE solvers to be used in `computeApproximatedProfiles` to calculate the profile likelihood. The `ode15s` solver provided by MATLAB is computing the profile on the whole parameter interval, while the IDAS solver provided by *sundials* stops as soon as the likelihood ratio drops under a certain threshold.

necessarily be of index one.
There are two DAE solvers included in the implemented function (see Figure 4.1), one of them is the MATLAB built-in solver `ode15s`, which solves ODEs but also DAEs with a – potentially singular – mass matrix $M(y)$ of the form:

$$M(y) \cdot \dot{y} = f(y) \tag{4.5}$$

This form is exactly the same as (3.7) if y is substituted with ζ and $M(y)$ with $A(\zeta)$. The algorithm of the solver is based on numerical differentiation formulas and is a multi-step solver [12]. Unfortunately, this DAE solver can not solve DAEs of index greater than one. But since the considered DAE is of index equal one, this solver suffices for the profile calculation.
The other solver is the IDAS solver of the *sundials* [13] toolbox. For this solver it is necessary to rearrange the equation, such that there are only zeros on the left hand side.

$$0 = A(\zeta) \cdot \dot{\zeta} - \begin{pmatrix} -\gamma \nabla_\theta \mathrm{l}(\theta_c) \\ 1 \end{pmatrix} \tag{4.6}$$

IDAS has moreover the option to change the applied linear solver. In the applications only two of them are used frequently: `GMRES` and `Dense`.
The inversion of the matrix is not necessary for the solver itself, but the IDAS solver requires a suggestion for $\dot{\zeta}_0$ at the initial point. If there is no sufficient information of the derivative at this point the computation of the inverse at the initial point cannot be avoided.

$$A(\zeta_0) \cdot \dot{\zeta}_0 = \begin{pmatrix} -\gamma \nabla_\theta \mathrm{l}(\theta_c) \\ 1 \end{pmatrix} \tag{4.7}$$

$$\dot{\zeta}_0 = A(\zeta_0)^{-1} \cdot \begin{pmatrix} -\gamma \nabla_\theta \mathrm{l}(\theta_c) \\ 1 \end{pmatrix} \tag{4.8}$$

If $A(\zeta)$ is singular and if this can not be avoided using the adjustments of the section before (see 4.4), it is not possible to calculate $\dot{\zeta}_0$, properly. In the implementation the singularity of $A(\zeta)$ results in `NaN` or `Inf` (if the matrix is only badly scaled) as entries of

$\dot{\zeta}_0$. In this case it occurs that the solver does not converge in its first step and to prevent this, a contrived solution has to replace these entries with finite – but favourable big – values.
The implementation sets these entries to ± 100, depending on whether they are positive or negative infinite. For `NaN` always the positive value is chosen.

4.2.3 ODE based integration

To solve (3.7) the standard ODE solver always requires the inversion of the matrix $A(\zeta)$ as in (4.1). If the matrix is singular, which for example happens in systems with non-identifiable parameters, the numerical inversion of the matrix leads to entries, which are set to `NaN` or `Inf`.
However, it is shown in section 4.2.1 with reference to an example that this integration method is still possible, if certain adjustments are applied to the matrix.
For the numerical solving of ODEs many numerical methods exists. The used ones are summarized in Figure 4.2. MATLAB already provides a wide range of ODE solvers for various problems. Apparently the considered system of ODEs can become stiff, which is the reason for choosing `ode15s` and `ode23s` as examples from the MATLAB built-ins, for being able to solve stiff ODEs.
In the subsection before the `ode15s` was introduced as a DAE solver, but can also applied to ODEs. The solver `ode23s` is based on the Rosenbrock formula and is a one-step method [12]. This solver was already used in [1] and it will be investigated if this solver also suffices for the examples considered in this thesis.
Furthermore these two solvers will be compared to the Euler method – a quite simple approach to solve ODE systems, introduced subsequently – and the toolbox CVODE also provided by *sundials*. CVODE does – like IDAS – allow different inputs for the linear solver, which additionally can be compared in terms of accuracy and number of evaluations (NoE).

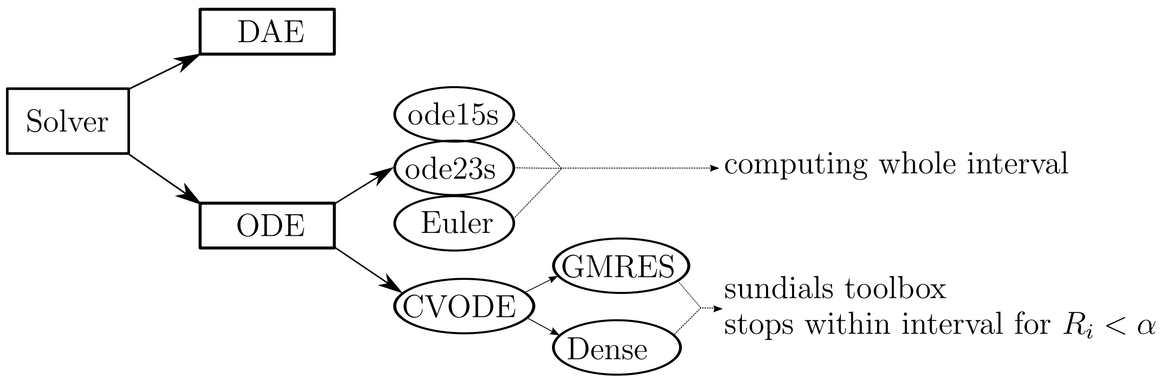


Figure 4.2: Possible ODE solvers to be used in `computeApproximatedProfiles` to calculate the profile likelihood. `ode15s` and `ode23s` provided by MATLAB as well as the Euler solver are computing the profile on the whole parameter interval, while the CVODE solver provided by *sundials* stops as soon as the likelihood ratio drops under a certain threshold.

Euler method

The explicit Euler method is used to solve ODEs [11]. Given an ODE of the form:

$$\dot{y} = f(y, t, \theta) \tag{4.9}$$

where $\dot{y}$ being the first derivative of $y(t, \theta)$.
Moving along the path of y is like adding the value of its derivative in an infinitely small step δ to the value of y at the point before. This results in the iterative function:

$$y_{n+1} = y_n + \delta \cdot f(y_n, t_n)$$

where δ is the step size.
This is nothing else than the rearranged FD:

$$\dot{y}_n = \frac{y_{n+1} - y_n}{\delta} \tag{4.10}$$

$$y_{n+1} = y_n + \delta \cdot \dot{y}_n = y_n + \delta \cdot f(y_n, t_n) \tag{4.11}$$

This solution is exact for an infinitely small step size. However for the applied computation a finite step size has to be set, which causes a numerical error. In the implementation, the default step size of the Euler method is set to: $\delta = 10^{-3}$.
The considered Euler method is an explicit method and therefore sensitive for stiff problems [11]. Adding the correction term to the profile approximation with a high factor γ, results in a stiff problem and in these cases it is not appropriate to solve the ODE with the Euler method.
This is displayed by Figure 4.3. For $\gamma = 1$ the Euler method with step size $\delta = 10^{-3}$ calculates the likelihood profile of the parameter k_2 quite well (the corresponding model can be found in 2.3.3). However, increasing the factor γ to 100 results in such a stiff system that the Euler method is not able to sufficiently calculate the profile. Only by decreasing the step size of the method to $\delta = 10^{-4}$ results again in an accurate profile.

4.2.4 Solver performance evaluation

After the introduction of the solvers it will be evaluated how they perform on the profile calculation. Therefore, the implemented function is applied to the standard normal distribution (SND) and the crystal growth model (CGM).
At first consider the profiles for the SND calculated with various solvers shown in Figure 4.4a.
All in all, the approximated profiles fit quite well on the benchmark, which is computed by the classical method. This is confirmed by the evaluation figures (Figure 4.4c), where the error to the benchmark is equal zero for all computation variations. Attention should also be given to the NoE.
The benchmark needs around 4×10^2 evaluations of the cost function, which is undercut by all solvers except the Euler method. Why the Euler needs so much more NoE is easy to explain, due to the fact that this method – in contrast to the others – uses a fixed step size. Moreover the profile with the CVODE seems inaccurate, but this is only because it

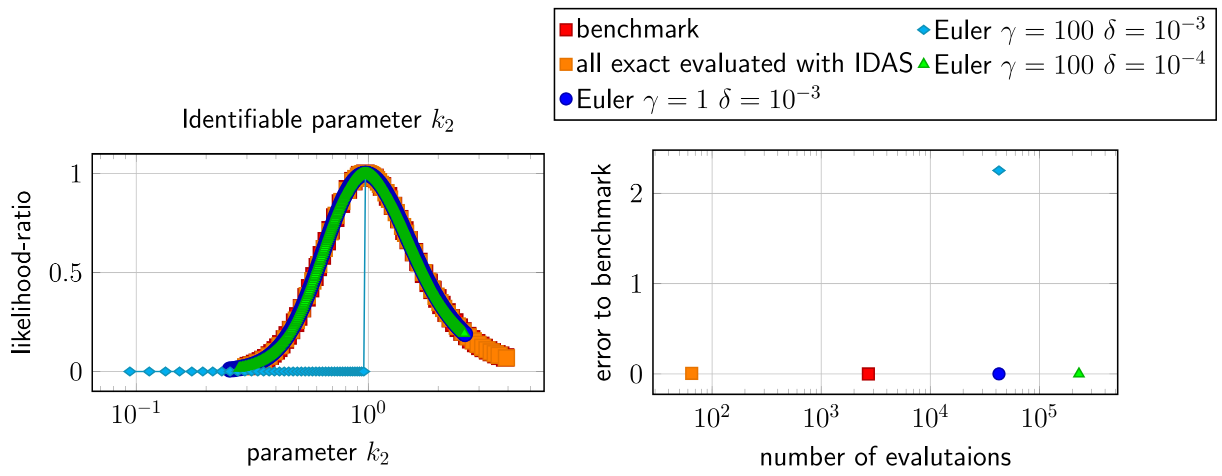


Figure 4.3: Profiles of the chemical reaction model for the parameters k_2, calculated by `computeProfiles` and by `computeApproximatedProfiles`. The latter one uses IDAS (orange squares matching the benchmark nearly exactly) and the Euler method. The Euler method calculates for $\gamma = 1$ the profile (blue circles) but fails for a high factor $\gamma = 100$ and step size $\delta = 10^{-3}$ (light blue diamonds), since it does not match the profile but drops to zero. This can only be resolved by reducing the step size δ to 10^{-4} (the green triangles match the benchmark), which shows the stiffness of the system in these cases.

uses partially quite big step sizes.
4.4b shows the profile for the crystal growth model (CGM) and the profiles are quite accurate for all solvers, as it is in the other example with the SND. However, once more, the CVODE uses large step sizes and is in this case producing an increased error, compared to the other solvers. But considering the profiles in Figure 4.4b this error is not visible to the naked eye.
It is interesting that `ode15s` for ODEs needs a similar NoE of the cost function as IDAS, despite it is computing a wider interval, and much less than the `ode23s`. This one was used in [1] for the computation of the profiles of the CGM and needs even more NoE than the benchmark.
Additionally, there is one main difference between the two MATLAB built-in solvers as well as the Euler method on the one hand and the ones provided by *sundials* on the other. The latter ones are executed stepwise and therefore it is possible to evaluate the likelihood-ratio in every step. This evaluation can be used as a stopping criterion of the algorithm, when, for example, the ratio falls under the desired threshold.
The other solvers, however, are calculated in a fixed interval and do not stop even if the ratio is zero. Furthermore the interval passed to the solvers does not exactly reflect the boundaries of the concerned profile parameter, but defines the boundaries of c. Recalling equation (3.2) c is defined to be equal to the profile parameter, but is not assigned to its value after the differentiation any more. The ODE solver changes the value of c and computes the solution for ζ. It appears that the values of c do not correspond exactly to the profile parameter.
Therefore it can happen that the calculated profile is too short or evaluated at unnecessary many points. That means the profile calculation with these solvers is dependent on the passed interval, which is then actually independent of the considered parameter

interval. Although it is possible to assign `ode15s` with a stopping criterion, this does not prevent the issue of the calculation of a too short interval.

Concluding, the best results are provided by the *sundials* solvers CVODE and IDAS. The MATLAB built-in solver `ode15s` for DAEs and ODEs performs well, too, but the disadvantage is crucial that the whole interval is computed or only parts of the desired profile, which can lead to wrong assumptions about the confidence intervals. Therefore the solvers with stopping criteria dependent on the actual parameter intervals should be preferred.

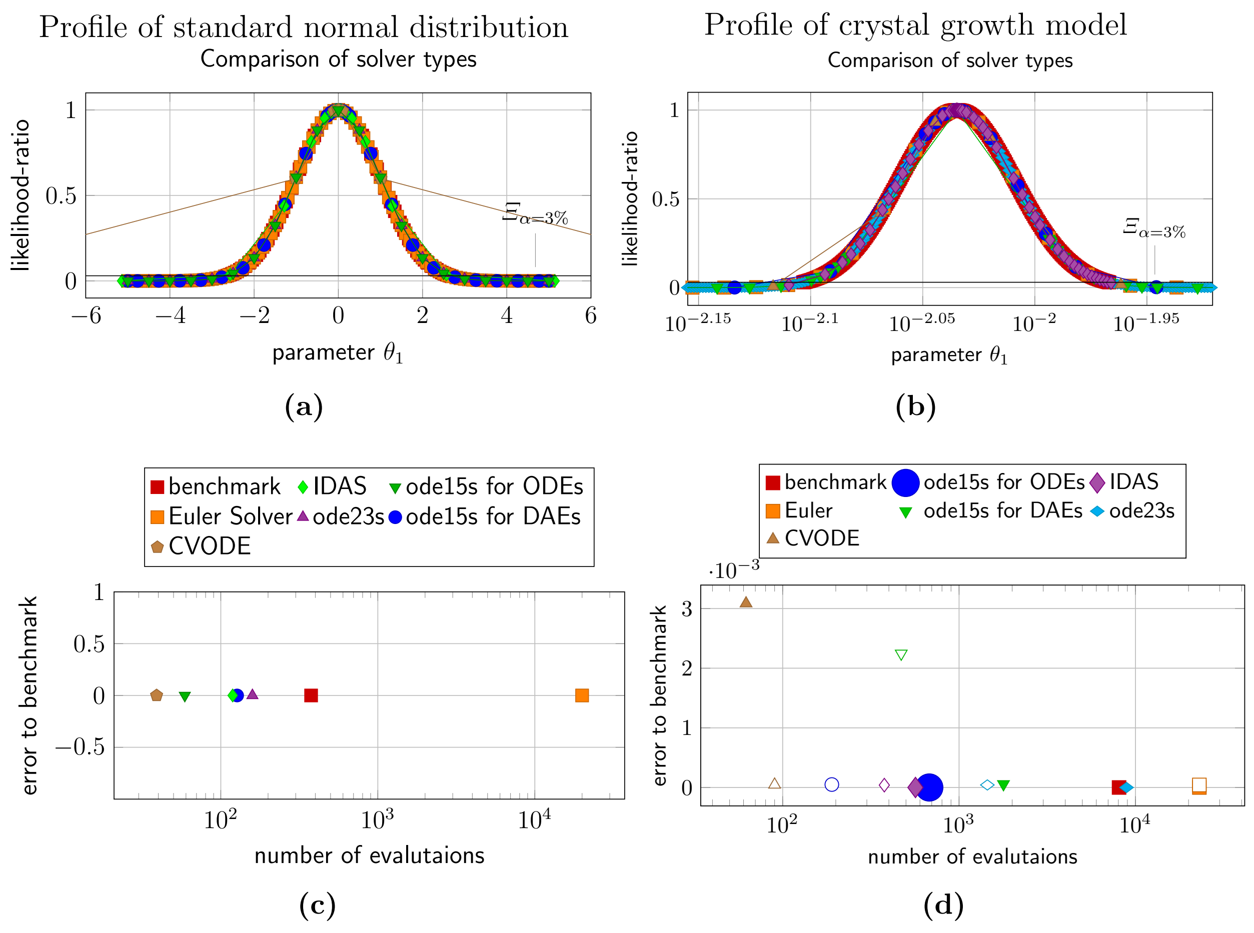


Figure 4.4: Profiles calculated with the classical method (benchmark) and the integration based method using different solvers: Euler method, `ode15s` for ODEs, `ode15s` for DAEs, `ode23s`, IDAS and CVODE. (a) & (b) show the profiles and (c) & (d) the evaluation plots, with NoE plotted against the error to the benchmark. (a) & (c): Profiles for the standard normal distribution for parameter θ_1. (b) & (d): Profiles of the crystal-growth model for the parameter θ_1. (d): The unfilled markers determine the calculation if the parameter function is defined in terms of the logarithmic parameters and therefore linear. Omitted in (b), since there is no difference visible.

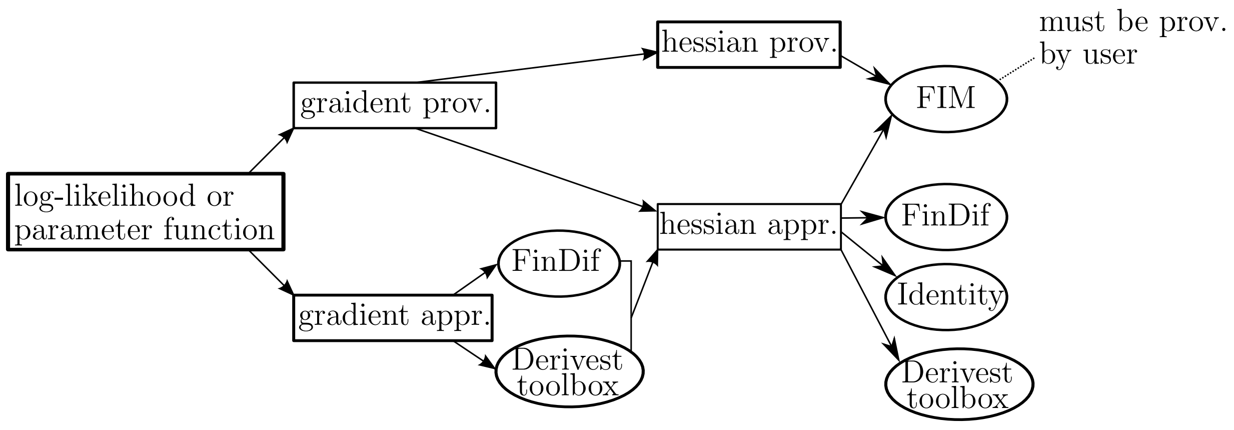


Figure 4.5: Possible combination of log-likelihood and parameter function derivatives, either provided (prov.) by the function or approximated (appr.). The possible approximation methods have a oval frame. If the fisher information matrix (FIM) is chosen to be the approximation method for the hessian, it must be provided by the user, because the calculation of the FIM requires the gradient of the underlying model.

4.3 Approximation methods and substitutes of derivatives

To obtain the profile likelihood, the DAE (3.4) must be solved. First of all this requires that the derivatives of the log-likelihood function and the parameter function are available.
For $g(\theta)$ it is not very difficult to calculate the gradient and hessian, since it is a linear function. But circumstances are different for the log-likelihood function, for there is often no analytical differentiation possible and no exact solution of the profile can be calculated. This section introduces methods to approximate the derivatives and will moreover show possibilities to substitute the log-likelihood derivatives.
The function `computeApproximatedProfiles` accepts various combinations of the derivatives and their approximations (summary shown in Figure 4.5). Some of these combinations will be evaluated at the end of this section.

4.3.1 Approximation methods of derivatives

As long as the analytical solutions of the gradient and hessian of the log-likelihood are not available they have to be replaced by appropriate approximations. The following section introduces methods used for the numerical approximation in the application examples and evaluates their accuracy.

Finite Differences

One simple and common method to approximate derivatives are the FD, defined as follows:

$$\frac{d\mathrm{l}(\theta)}{d\theta} := \frac{\mathrm{l}(\theta + h) - \mathrm{l}(\theta)}{h}$$

where the accuracy depends on the step size h. This step size is comparable to the step size of the Euler method δ, but a different notation is used to distinguish between the two.
For h being infinitely small, the FD is similar to the definition of the differentiation. For a practical application a numerical error of the approximation must be accepted, since step size h can not be set infinitesimally small.
In the case of the hessian the FD is applied two times. This increases the numerical error as well as the computation time and is therefore not the favourable approximation method for the hessian.

Sensitivity equations

Using the sensitivity equations is a common method to derive the gradient of the solution of differential equation systems with respect to the parameters θ. This method is based on the sensitivity equations, which can numerically be solved simultaneously with the corresponding system of differential equations.
Consider for example the explicit ODE:

$$\dot{y} = f(y, t, \theta)$$

with $\theta \in \mathbb{R}^n$.
Let Z be the new sensitivity variable, which is defined as:

$$Z := \nabla_\theta y$$

Now continue point wise with $Z_i = \frac{\partial y}{\partial \theta_i} \quad \forall i \in (1, ..., n)$.
The derivative of Z_i with respect to t is:

$$\dot{Z}_i = \frac{\partial}{\partial t} Z_i = \frac{\partial}{\partial t}\frac{\partial y}{\partial \theta_i} = \frac{\partial}{\partial \theta_i}\frac{\partial y}{\partial t} \tag{4.12}$$

where $\frac{\partial y}{\partial t} = \dot{y} = f(y, t, \theta)$.
Applying the chain rule to $f(y, t, \theta)$ i.e.:

$$\dot{Z}_i = \frac{\partial}{\partial \theta_i} f(y, t, \theta) = \frac{\partial f}{\partial \theta_i} + \frac{\partial f}{\partial y} \cdot \underbrace{\frac{\partial y}{\partial \theta_i}}_{Z_i} \tag{4.13}$$

$$\Rightarrow \dot{Z}_i = \frac{\partial f}{\partial \theta_i} + \frac{\partial f}{\partial y} \cdot Z_i \tag{4.14}$$

The resulting n ODEs (for $Z_1, ..., Z_n$)) can be solved together with the initial ODE, using common integration solvers to calculate the solution of the differential equation and the gradient of $y(t, \theta)$ at once [14].
The previous derivation was done for an explicit ODE, but sensitivity equations can also be calculated to systems of ODEs [14] and under certain conditions to DAEs and PDEs [15].
However, this approach needs further knowledge about the underlying model and is therefore not part of the implemented gradient approximation.

Toolboxes for Matlab

Rather than approximate the derivatives 'by hand' one can use already implemented toolboxes for a numerical approximation. There are various types of toolboxes available. The one used in the implementation part of this thesis is the `Derivest` toolbox [16]. It calculates the gradient and hessian, requiring only the considered function. Although this MATLAB function does approximate the gradient and particularly the hessian quite well, it needs more evaluation of the differentiated function than the FD.

Assessment

The question is, which method to approximate the derivatives is the best choice for the profile calculation. To answer this questions the different methods are applied in the function and compared in the following.
For the profiles in Figure 4.6 only the IDAS solver from *sundials* is used with different inputs for the derivatives.
In the case of the SND (Figure 4.6a and 4.6c) the approximation of the hessian with FD increases the NoE, while being nearly independent of the gradient approximation. The increase of the NoE is enormous when the toolbox `Derivest` is applied. This is already the case, even if only the hessian is calculated with the toolbox and this approximation is furthermore not reducing the error.
All in all, in the case of the SND no reduction of the NoE is achieved, if derivatives are approximated. But the better performance of the benchmark can be explained by the fact that the `computeProfiles` function uses an adaptive step size which is based on the standard normal distribution and is therefore producing the best results when applied to it.
However, in the example of the CGM (see 4.6b and 4.6d), only the approximation with `Derivest` does increase the NoE compared to the benchmark. The recurrence of the bad performance with the `Derivest` toolbox suggests that more accurate approximations are not improving the calculation compared to more simple approaches like the FD and is more likely to increase the NoE.
Despite the results of the SND the application of FD decreases the NoE for the CGM. This result is important, since the exact derivatives are often not available and the profile calculation is more likely be done relying solely on approximations.

4.3.2 Substitutes for the hessian

Instead of a numerical approximation it is also possible to substitute the hessian with surrogate matrices. The difference to an approximation is that the substitute does not intended to be near the exact value of the hessian but is chosen to show the same behaviour as the hessian.
There are two possibilities suggested in [1], which are applied to the example models. The first one is the identity matrix and the second one the FIM.

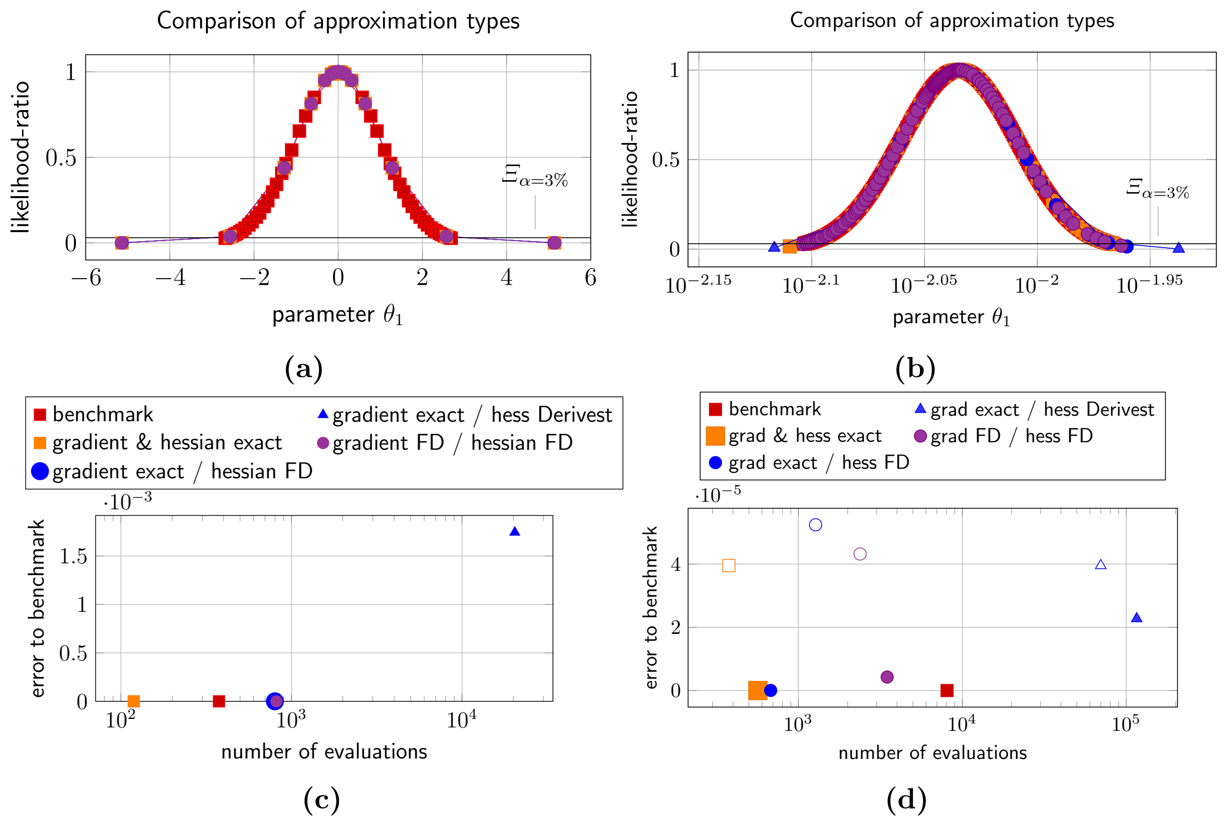


Figure 4.6: Profiles calculated with the classical method (benchmark) and the integration based method using the solver IDAS with default options and $\gamma = 0$. The comparison concerns the various combinations of true and approximated derivatives. (a) & (b) show the profiles and (c) & (d) the evaluation plots, with NoE plotted against the error to the benchmark. (a) & (c): Profiles for the standard normal distribution for parameter θ_1. (b) & (d): Profiles of the crystal-growth model for the parameter θ_1. (d): The unfilled markers determine the calculation if the parameter function is defined in terms of the logarithmic parameters and therefore linear. Omitted in (b), since there is no difference visible.

Identity

To completely avoid the calculation of the hessian or an approximation of it, the authors of [1] suggested to use the negative identity matrix of the same dimension as the hessian. In their paper and selected examples this choice works quite well.
However, their examples contained only identifiable parameters and it was necessary to use the correction term. Later in this chapter (see section 4.5) it will be shown that this is not appropriate for non-identifiable parameters, even with a high value of γ.
The advantage of using the identity is, of course, that the cost function does not need to be evaluated at all, which reduces the necessary NoE.

Fisher information matrix

The second substitute is the FIM and is defined as the matrix with entries:

$$\mathcal{I}(\theta)_{ij} = E\left[\frac{\partial}{\partial\theta_i}\log\varrho_\theta(X)\cdot\frac{\partial}{\partial\theta_i}\log\varrho_\theta(X)\right]$$

where X is a random variable distributed with the density ϱ_θ following [17].
It is for example used in the *information inequality* [17] and its inverse is a common variance estimator [18].
But this definition is far from the practical approach in this thesis. For the data-based parameter estimation the FIM can be used as an approximation for the hessian of the log-likelihood, as it was used in [1].
If the variance is independent of the parameters – hence known – the FIM is the inverse of the covariance. In this case it is defined following [19]:

$$\mathcal{I}(\theta) = \sum_{k=1}^{m}\frac{1}{\sigma_k}\nabla_\theta y(\tau_k,\theta)\nabla_\theta y(\tau_k,\theta)^T \tag{4.15}$$

This definition of the FIM requires the derivative of the model solution $y(\tau,\theta)$ with respect to the parameter θ. If this is not available, because there is for example no analytical solution, also an approximated gradient can be used.
Since the FIM needs the gradient of the underlying model, this substitute can not automatically be generated by the implemented function, but has to be provided by the user with the log-likelihood function.

Assessment

Profiles are calculated using the previously introduced substitutes and compared in Figure 4.7. The Subfigures 4.7a and 4.7c show again the profiles of the SND.
The substitution with the negative identity reduces the NoE significantly and needs even less evaluations than the calculation with the exact derivatives. That is because the hessian of the SND is exactly the negative identity matrix. Therefore the substitution of the hessian with it, does not increase the error. But it takes lesser evaluations because the log-likelihood function and therefore the model is not evaluated for the hessian, because the hessian is substituted with a constant matrix.

This stands in contrast to the CGM, where it is obvious from Figure 4.7b that the identity matrix is inadvisable to use, as long as the correction term is not present.
The substitution with the FIM for the CGM are well enough. Solely if the gradient is approximated with FD, the calculation has a visible error considering the profile plot (see Figure 4.7a), but is still reflecting the identifiability of the parameter.
The substitution with the FIM is not applied for the SND, because it is only a test function for the algorithm without an underlying model. To calculate the FIM it is necessary to have access to the gradient of the model solution.

To conclude, the assessment of the method confirms the hypothesis that the implemented method does compute the right profiles and can reduce the NoE significantly, without being more inaccurate. It should be noted that the NoE are reduced compared to the benchmark even if approximations of the derivatives are used. Although, it is possible that approximations of the derivatives, especially FD, increase the NoE. But for the CGM example, this value is still less than that of the benchmark.

4.4 Assessment concerning the logarithmic scale

The logarithmic scale is only applied to the CGM, since the SND has negative parameter values.
Both approaches of the logarithmic scale work in the considered example. The results of the calculation with the second approach are added to the evaluation plots of the profiles in Figure 4.4d, 4.6d and 4.7d in the form of unfilled markers. The profiles are neglected in the profile plot, because there is no difference visible to the other profiles.
Defining the constraint in terms of the logarithmic parameters leads to an additional reduction of the NoE and the error to the benchmark in most cases (see 4.4d, 4.6d and 4.7d). There is only one exception in Figure 4.7d, where in the calculation with exact gradient and hessian approximated with FD the second approach of the logarithmic scale takes more NoE than the first one.
These reductions are positively noted, but are not yet very significantly in the example with the CGM. Defining the optimization constraint in terms of the logarithmic parameters has a much higher influence in the presence of non-identifiable parameters, as the next section will show.

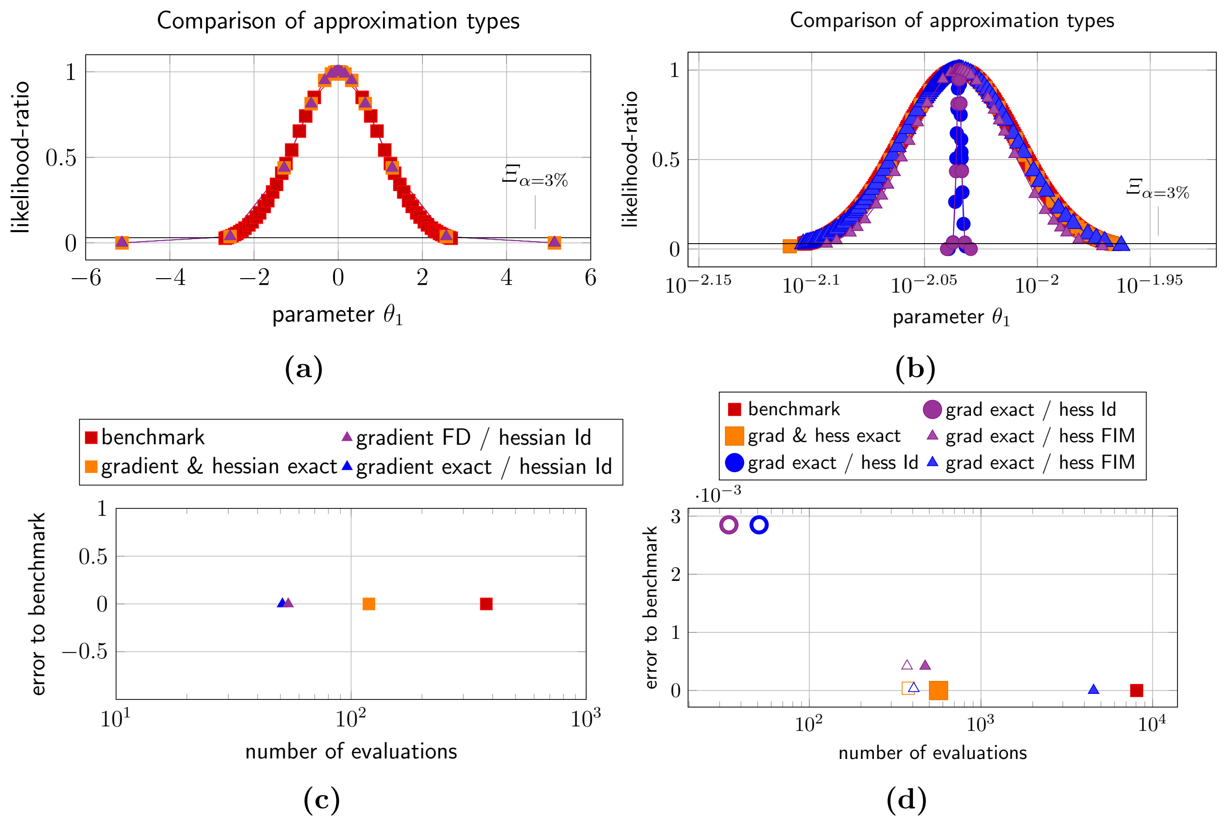


Figure 4.7: Profiles calculated with the classical method (benchmark) and the integration based method using the solver IDAS with default options and $\gamma = 0$ to investigate the influence of the various combinations of exact and substituted derivatives. (a) & (b) show the profiles and (c) & (d) the evaluation plots, with NoE plotted against the error to the benchmark. (a) & (c): Profiles for the standard normal distribution for parameter θ_1. (b) & (d): Profiles of the crystal-growth model for the parameter θ_1. (d): The unfilled markers determine the calculation if the parameter function is defined in terms of the logarithmic parameters and therefore linear. Omitted in (b), since there is no difference visible.

4.5 Presence of non-identifiable parameters

Previously, the implementation was assessed with models, which included only identifiable parameters. In this section the profile calculation will be applied to the chemical reaction model (introduced in 2.3.3), where the parameters s and k_1 have already been detected to be non-identifiable.
The profiles of the chemical-reaction model are computed for the parameters s and k_2, but there are no profiles shown for the parameter k_1 since its profile is basically the same as for the parameter s. As before the different solvers will be evaluated as well as various approximation methods. Furthermore the influence of the correction term will be investigated.

4.5.1 Solver performance evaluation

The Figure 4.8 shows the approximation of the profiles using different solvers.
For the parameter s (profile 4.8a) all solvers of the approximation method are as accurate as the benchmark. Only the CVODE solver seems like an outlier, but the error of $1.5 \cdot 10^{-10}$ is still too small to be noteworthy.
The profile in Figure 4.8 shows two difficulties in the calculation. The first one concerns the MATLAB built in solvers, which calculate far more than the desired interval on the lower side of Figure 4.8b. The profile, where the ratio is zero is of no interest. They are also stopping before the threshold is reached at approximately $k_2 = 3 \times 10^0$.
The second one is that, IDAS with the linear solver *Dense* as well as the `ode15s` for DAEs does not compute the whole profile. After a couple of iterations the solvers do not converge any more and stop. For the IDAS Solver this can be resolved by changing the linear solver to `GMRES`. For this reason the linear solver `GMRES` is used as the default value in the implementation.

All in all the CVODE and `ode15s` for ODEs show again the best performance, which suggests that the ODE solvers are superior to the DAE solver, even if the inversion of the mass matrix is necessary in every step.
However IDAS showed a bit higher accuracy if approximations are used and is therefore used for the following profile calculations in this section.

4.5.2 Performance evaluation concerning the derivatives

There are various possibilities to substitute the exact hessian of the log-likelihood. Figure 4.9 shows four of them.
Instantly, the inaccuracy of the identity matrix, especially for the non-identifiable parameter, leaps to the eye. In absence of the correction term, the identity matrix is not even a good choice for the identifiable parameter k_2.
The toolbox `Derivest` for calculating derivatives holds the accuracy, but increases the NoE substantially. Also the FD needs more evaluations than the benchmark. However using the FIM for the hessian results in nearly as many or only a little more NoE as if all derivatives are exact. This corresponds to earlier results with the FIM and allows a preliminary recommendation for the substitute of the hessian.
The next evaluation regards the influence of the gradient approximation. As long as the

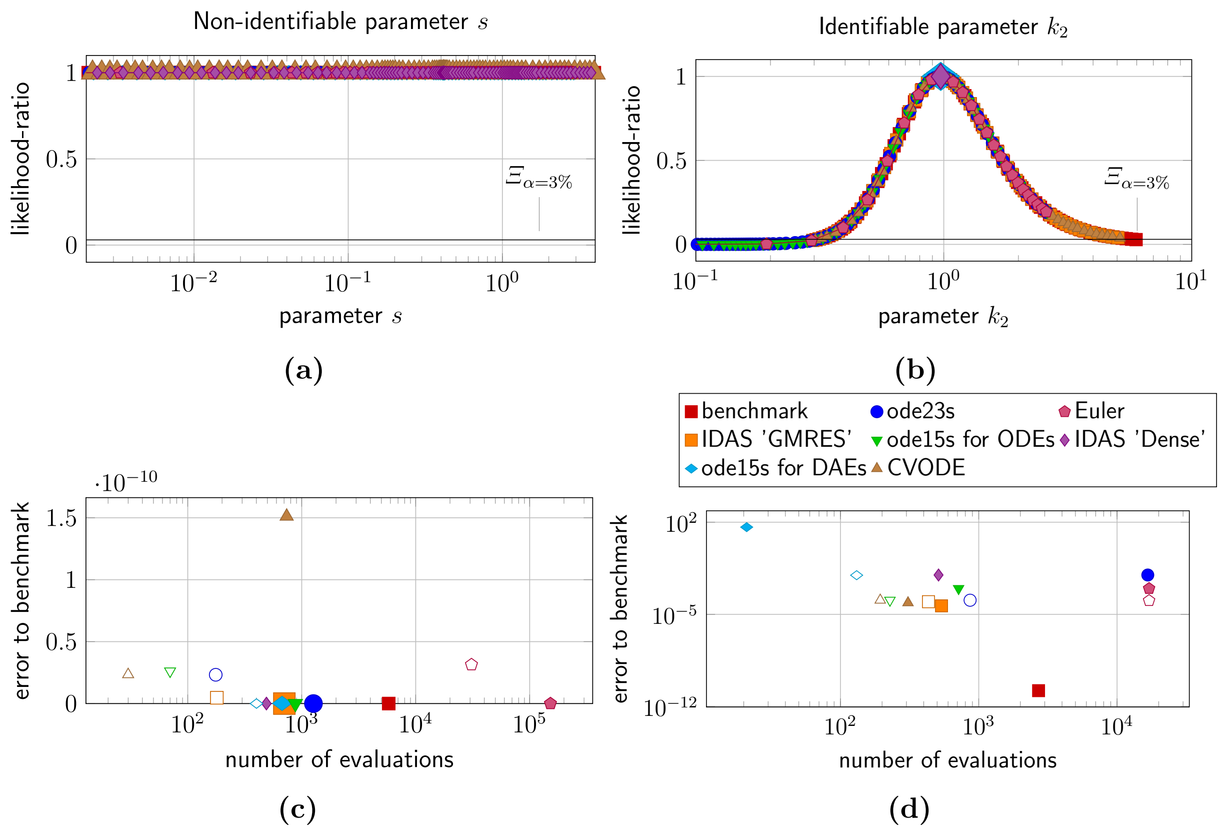


Figure 4.8: Profiles of the parameters s and k_2 in the chemical-reaction model calculated with the classical method (benchmark) and the integration based method with exact derivatives and $\gamma = 0$ to investigate the different performances of the solvers: Euler method, `ode15s` for ODEs, `ode15s` for DAEs, `ode23s`, IDAS and CVODE. (a) & (b) show the profiles and (c) & (d) the evaluation plots, with NoE plotted against the error to the benchmark. (b): `ode15s` for DAEs and IDAS with linear solver `Dense` fail to compute the profile for parameter k_2 (cyan and purple diamond). (c) & (d): The unfilled markers determine the calculation if the parameter function is defined in terms of the logarithmic parameters. Omitted in (a) & (b), since there is no difference visible.

correction term is 'disabled' (i.e. $\gamma = 0$) it would make no difference, because the gradient does not directly appear in the profile calculation. Therefore the correction term is now included by setting $\gamma = 1$. For comparison the gradient is either exact or approximated via FD with the step size $h = 10^{-4}$. The hessian is approximated with the FIM or with the identity.

The corresponding profiles can be seen in Figure 4.10. In this figure it is obvious that the substitution of the hessian with the identity works better with the correction term. There is still a serious error for s (although the forward direction of the calculation almost matches the benchmark), but for k_2 the profile is nearly exactly approximated.

It is important, that the approximation of the gradient does not influence the profile at all (Figure 4.10a: both lines – gradient approximated and exact – with hessian be the identity, matching each other).

In the other case, where the FIM is the substitute for the hessian, the profile is fine

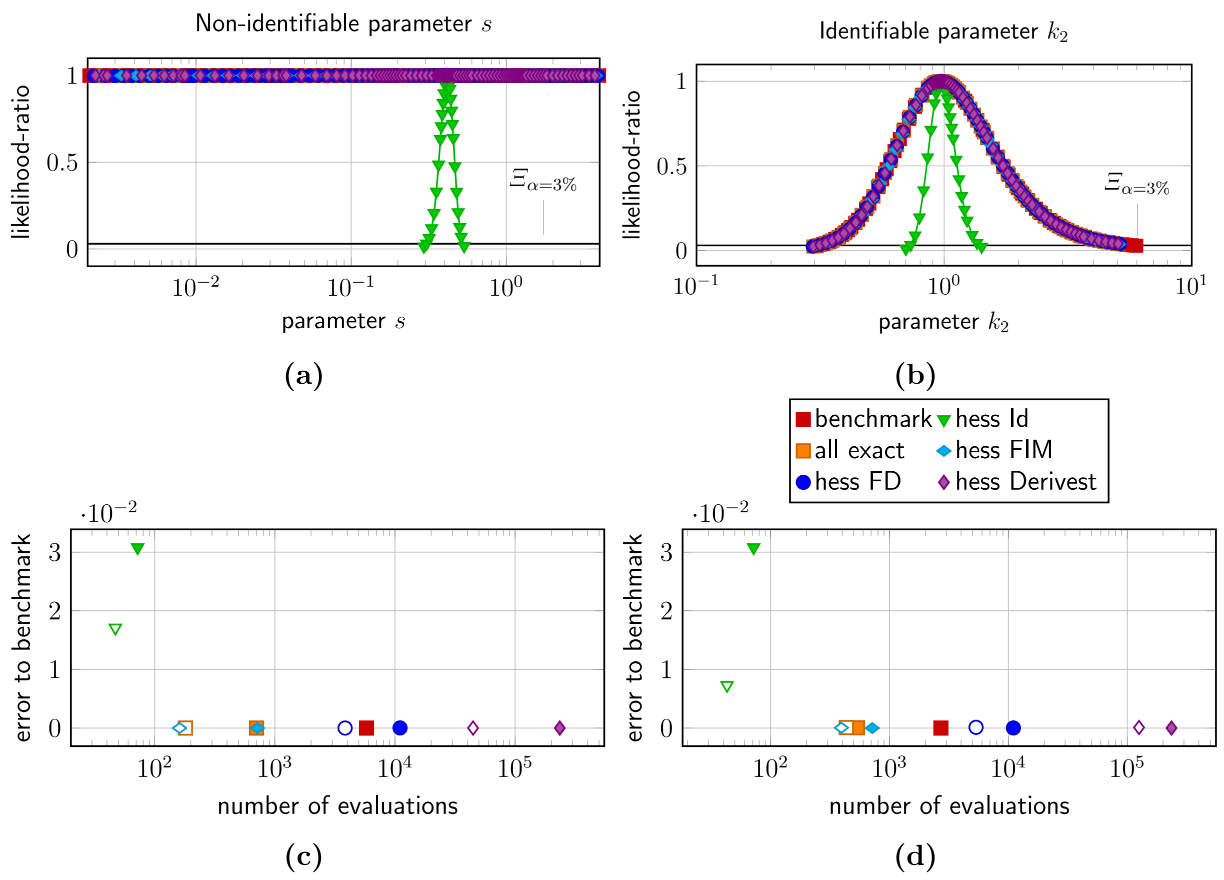


Figure 4.9: Profiles of the parameters s and k_2 calculated with the classical method (benchmark) and the integration based method using IDAS with default options and $\gamma = 0$ to investigate the influence of the different hessian substitutes in the chemical-reaction model with gradient exact and various approximations for the hessian (hess.). (a) & (b) show the profiles and (c) & (d) the evaluation plots, with NoE plotted against the error to the benchmark. (a): The identity matrix as substitute for the hessian does not reflect the non-identifiability of parameter s (green triangle). (c) & (d): The unfilled markers determine the calculation if the parameter function is defined in terms of the logarithmic parameters. Omitted in (a) & (b) for there is no difference visible.

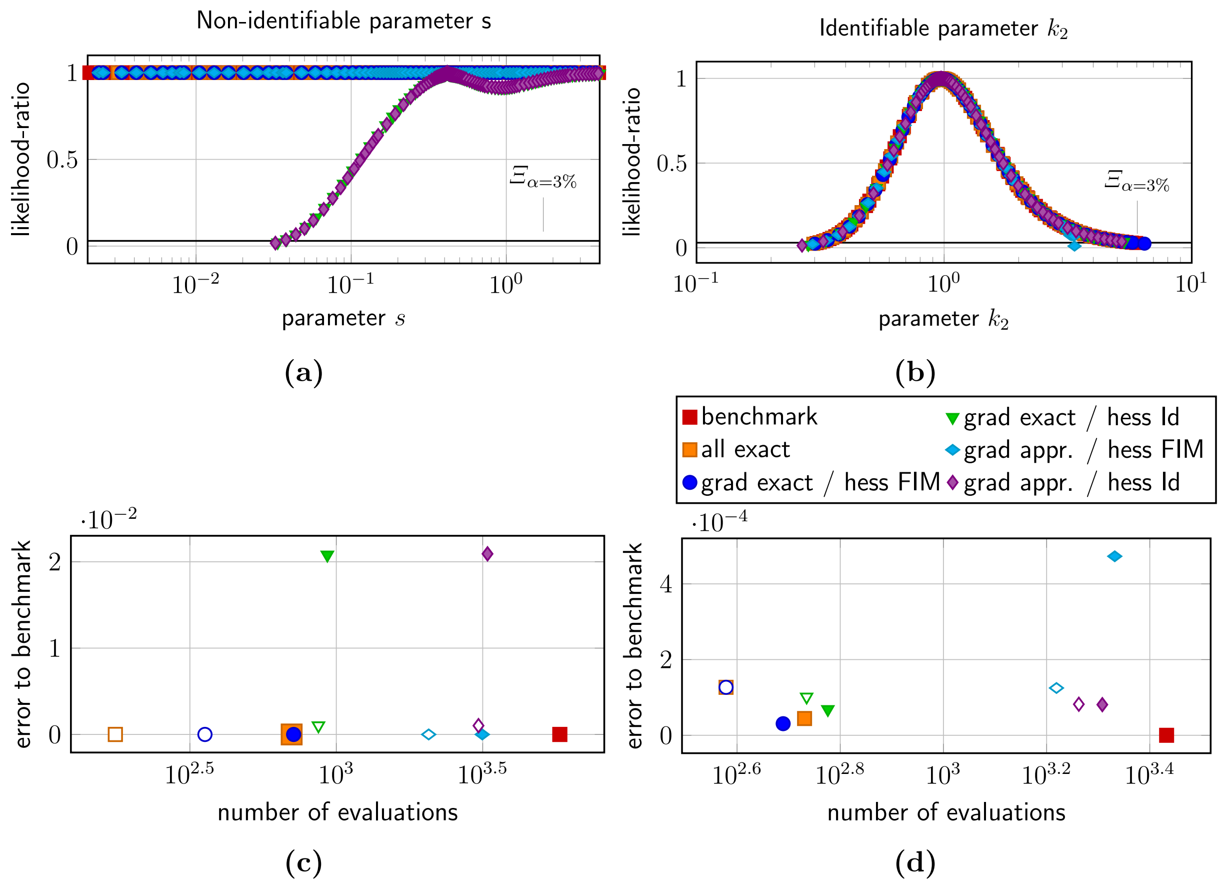


Figure 4.10: Profiles of the parameters s and k_2 calculated with the classical method (benchmark) and the integration based method using IDAS with default options and $\gamma = 1$ to investigate the influence of approximated gradient (grad appr.) in the chemical-reaction model: gradient exact or FD, hessian FIM or Identity (Id). (a) & (b) show the profiles and (c) & (d) the evaluation plots, with NoE plotted against the error to the benchmark. (c) & (d): The unfilled markers determine the calculation if the parameter function is defined in terms of the logarithmic parameters. Omitted in (a) & (b) for there is no difference visible.

for the non-identifiable parameter, but the method needs more NoE if the gradient is approximated. The same applies for the identifiable parameter, but here the increase of NoE and of the error is significant, if the gradient is approximated.
Following these results it is investigated how the accuracy of the gradient approximation influences the profile calculation with the FIM. For calculating the FIM only the gradient of the model is necessary. If this gradient is not available it needs to be approximated, too, even if the gradient of the likelihood does not appear in the differential equation.
Considering Figure 4.11a and 4.11c, it is obvious that the use of an approximated gradient does increase the NoE but an improvement of the approximation with smaller step size does not change anything.
This stands in contrast to Figure 4.11b and 4.11d, where the computation works well for FIM with the gradient approximated with step size $h = 10^{-2}$, but takes more NoE than the benchmark for step size $h = 10^{-4}$, and fails completely for step size $h = 10^{-6}$. That is because, if the tolerance of the solver is bigger than the accuracy of the approximation, the solver only 'sees' its own numerical error, which causes wrong results and the profiling fails.

4.5.3 Influence of the correction term

In Figure 4.10 an improvement is shown, if the correction term is present, regarding the calculation with the identity matrix as the hessian. To visualize the influence of the correction term, its impact is further increased in Figure 4.12 by changing the magnitude of the factor γ. In the figure γ is set to 0, 1 and 100 and again the bad performance for $\gamma = 0$ is obvious for the non-identifiable parameter.
In the Figure 4.12b, setting $\gamma = 1$ is sufficient that the profile for the identifiable parameter is relatively precise. For the non-identifiable parameter, though, it is necessary to further increase γ to calculate a profile that at least reflects the non-identifiability (see Figure 4.12a). But there is still a drop visible at the lower end. The application in chapter 5 will show that an increasing γ alone does not always produces valid results.
It also should be noted that the increase of the factor significantly increases the NoE, since the stiffness of the resulting system makes the computation more difficult for the applied solvers.
However, if the optimization constraint is applied in terms of the logarithmic parameters the performance of the method changes significantly (see Figure 4.12c and 4.12d). Already with the correction term factor $\gamma = 1$, the non-identifiability of the parameter s is visible and by further increasing the factor, this approach even leads to an accurate profile, which needs even less evaluations than with $\gamma = 1$.
This is a crucial advantage in models where the hessian is not available and the FIM is initially not implemented or can not be used as an approximation for the hessian (e.g. if the variance is not known).

4.5.4 Evaluation of the performance in the logarithmic scale

In this example with non-identifiable parameters there is a significantly gain, when using the second approach of the logarithmic scale. The NoE can be reduced by at least one magnitude in comparison to the benchmark. To visualize this with numbers, consider the

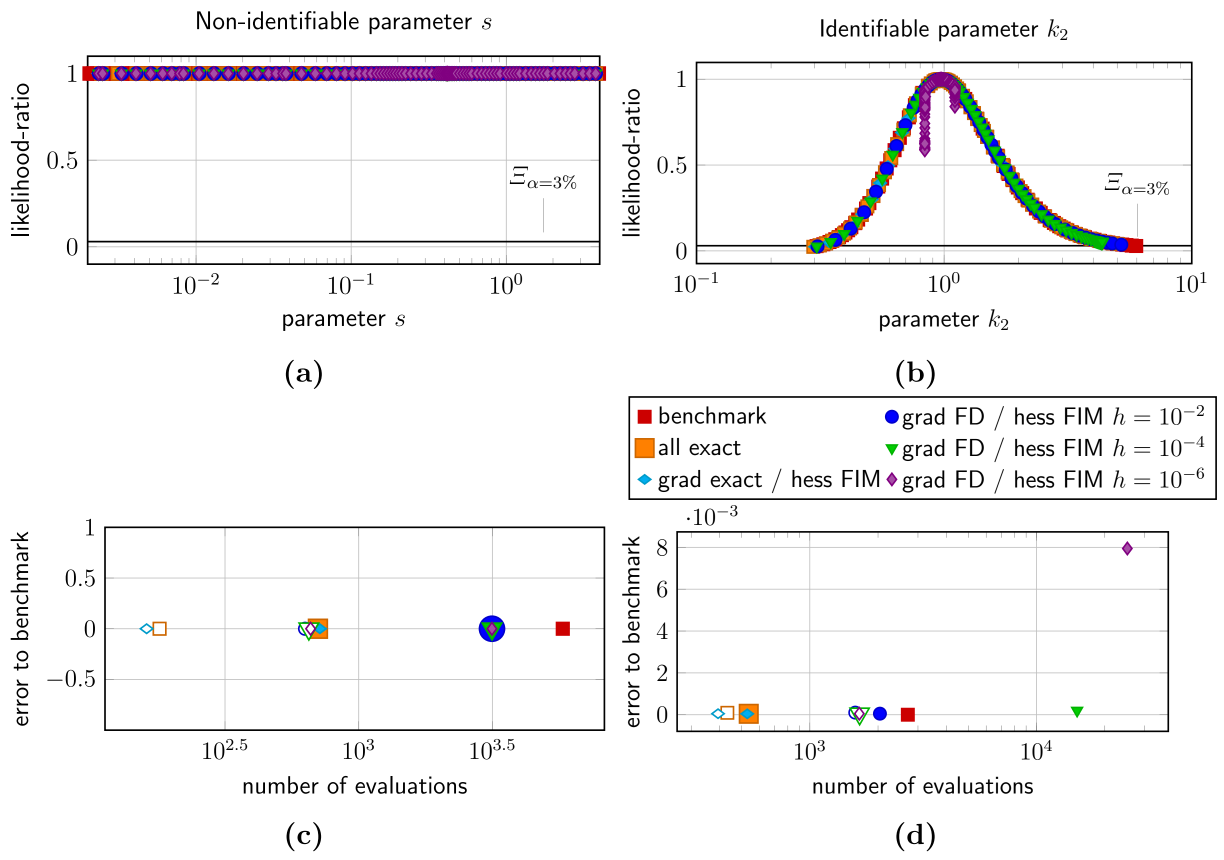


Figure 4.11: (a) & (b): Profiles of the parameters s and k_2 calculated with the classical method (benchmark) and the integration based method using IDAS with default options and $\gamma = 0$ to investigate the influence of the approximation step size h for the gradient (grad) of the model, if the hessian (hess) is substituted with the FIM in the chemical-reaction model: gradient exact or FD, hessian FIM. (a) & (b) show the profiles and (c) & (d) the evaluation plots, with NoE plotted against the error to the benchmark. (c) & (d): The unfilled markers determine the calculation if the parameter function is defined in terms of the logarithmic parameters. Omitted in (a) & (b) for there is no difference visible.

case where all derivatives are exact. The solver CVODE needs then only 30 evaluations in contrast to the benchmark, which needs over 5000.
This is remarkable, since the benchmark takes for non-identifiable parameters always longer than for identifiable ones. That is because the benchmark is in the case of non-identifiable parameters computing the whole interval, whereas the algorithm can stop in the case of identifiable parameters, if the ratio drops under the threshold. That means the computing time of the classical method is for non-identifiable parameters dependent on the interval length. The same disadvantage does, of course, occur for the integration based method, but is resolved by immense huge step sizes.
Therefore is the performance of the integration based method reversed compared to the classical method. The NoE for non-identifiable parameters is mostly less than the NoE for identifiable parameters, because a wider step size can be applied.
Moreover the example, where the hessian is substituted with the identity matrix shows

an additional advantage of the constraint in logarithmic parameters. In the first approach of the logarithmic scale the identity matrix is not a favourable substitute for the hessian, since the method is then not able to sufficiently reflect the non-identifiability of the parameters and an increase of the factor leads to a stiff problem. This however is not the case for the second approach, where the non-identifiability is already visible for a small factor and a higher factor even improves the calculation in terms of NoE.

Concluding, defining the constraint in the logarithmic scale is favourable for the profile calculation, not only considering the NoE, but also in terms of accuracy. It is especially advantageous for non-identifiable parameters.

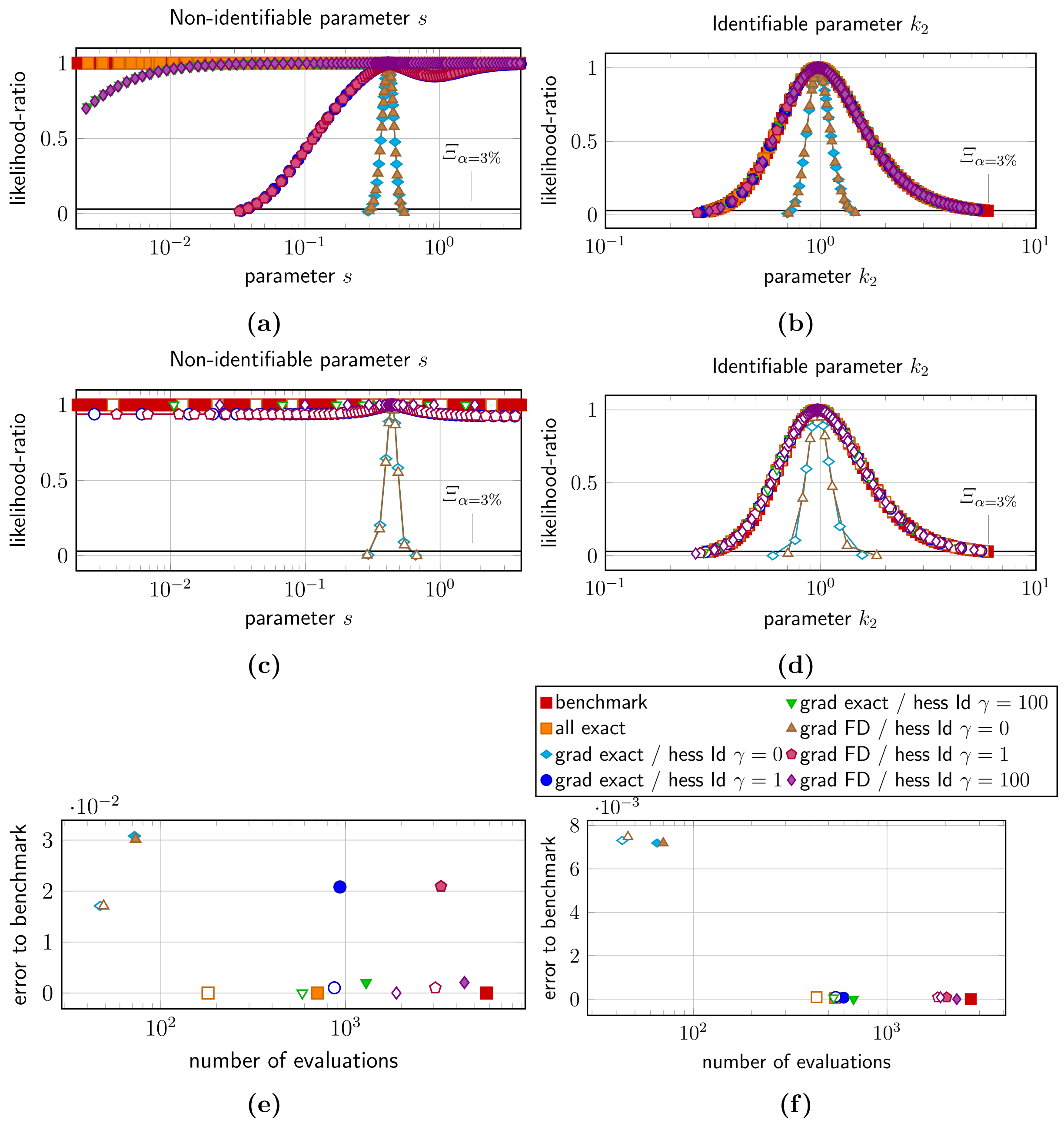


Figure 4.12: Profiles of the parameters s and k_2 calculated with the classical method (benchmark) and the integration based method using IDAS with default options to investigate the influence of the correction term with an improper hessian substitute in the chemical-reaction model: gradient is exact or FD, hessian is the identity matrix and $\gamma \in (0, 1, 100)$. (a), (b), (c) & (d) show the profiles (e) & (f) the evaluation plots, with NoE plotted against the error to the benchmark. (a) & (b): optimization constraint in terms of initial parameters (see 3.10). (c) & (d): optimization constraint in terms of logarithmic parameters (see 3.12). (e) & (d): Evaluation figure compares error with NoE. The unfilled markers correspond to (c) & (d).

Chapter 5

Evaluation of the integration based method applied to a PDE model

In most cases PDE based models are computational more costly to simulate than ODE systems. Therefore a reduction of the NoE in the profile computation is in this context particularly desirable. As an application example of a real data PDE, the *pom1p* model from [20] is used. Additionally it contains with five, the highest number of parameters of the considered models.

5.1 The *pom1p* model

The model describes concentration gradients of the protein *pom1p*. Within cells, concentration gradients are responsible for many biological processes. In the yeast cell *Schizosaccharomyces pombe*, the *pom1p* protein – together with two landmark proteins – regulates, among other things, the bipolar growth of the cell and the cell mitosis. *Pom1p* forms a gradient at both ends of the rod-shaped cell and produces negative signals for the division, while in the middle of the cell positive signals are produced to divide the cell at that point [21].
To observe the protein in a living cell fluroscence microscope images were made (see Figure 5.1). *Pom1p* is visible with a higher fluorescence in the pictures and one can see, that it is congregating at the cell tips.

PDE system

Considered is a source-diffusion-disassociation (SDD) model for the concentration of the *pom1p* protein [20]. It describes the emerging of the protein at the membrane (source), the diffusion on it and the unbinding (disassociating) from the membrane.
The PDE for the concentration $\rho(t, z)$ of *pom1p* at the time t on the singular spatial dimension, denoted by the variable d, is:

$$\partial_t \rho = D \cdot \nabla_d^2 \rho - \mu \cdot \rho + J \cdot e^{d^2/2w_{tea}^2}$$

with D being the diffusion parameter, μ the unbinding rate and J the maximum amplitude in $\frac{\mu m^2}{s}$ with which the *pom1p* molecules are emerging at the membrane. The parameter

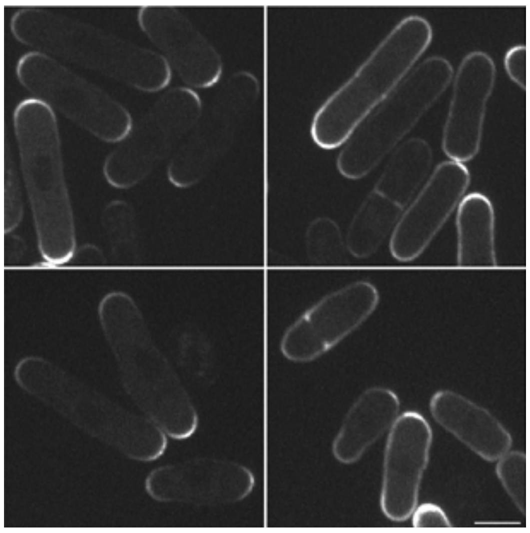


Figure 5.1: Fluorescence microscope images of the yeast cell *Schizosaccharomyces pombe*. The protein *pom1p* has a higher fluorescence and congregate at the cell tips. Image taken from [21]

w_{tea} denotes the width of the associated region and the observable of the concentration ρ is scaled with an additional parameter s:

$$y(t, d, \theta) = s \cdot \rho(t, d, \theta)$$

with θ being the set of all parameters together: $\theta = (D, J, \mu, w_{tea}, s)$.
To obtain the solution of the model, the PDE is solved by a FD approach on a spatial grid with 1401 points.

Fitting of the model parameters

The model was fitted to data, which was obtained by fluorescence microscope images. The concentration was measured over one spatial axis by the intensity of the fluorescence including a possible variance of the measurement.
A fit of the model parameters is shown in Figure 5.2. For the optimization of the log-likelihood the multi start algorithm by Jan Hasenauer was executed 20 times with the initial guess visualized in Figure 5.2a. The result is a candidate for the MLE. The simulation of the model using the fitted parameter set is shown in Figure 5.2b and it is obvious that the results fit to the experimental data quite well.

What does not become obvious with these results is, that in this model all parameters are non-identifiable except the scaling parameter s. Even if the profile of the parameter w_{tea} has a unique maximum (see for example Figure 5.3b) it is a practically non-identifiable parameter, because the definition only requires one infinite bound and the likelihood ratio of this parameter does not drop under the threshold at its lower end, which is therefore treated as infinite. But it depends on the significance level α, if w_{tea} is treated as identifiable or not, therefore this parameter will in the following be referred to as quasi-identifiable.

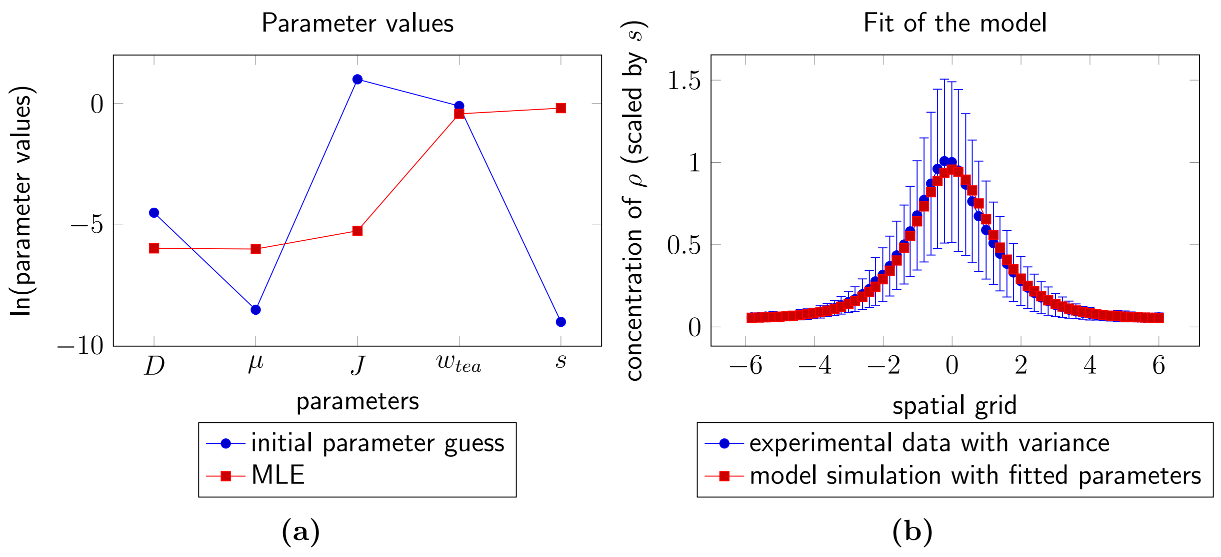


Figure 5.2: Model fitting of the *pom1p* PDE model to experimental data from fluorescence microscope images using a multi start algorithm. (a) shows the initial guess of the parameter values compared to the MLE. (b) shows the experimental data with its variance and a simulation of the model using the MLE for the parameter set.

In this application example the implemented log-likelihood function provides both derivatives. The gradient is evaluated using sensitivity equations and the hessian is substituted with the FIM, whereas the exact derivatives are not available.

5.2 Evaluation concerning solvers and hessian

The previous example with chemical reactions and non-identifiable parameters shows that defining the optimization constraint in the logarithmic scale does improve the integration based method significantly. Therefore this approach is used for the calculation of the profiles in the PDE model to gain the maximal possible reduction of NoE.

Solver performance evaluation

The corresponding profiles calculated with different solver are shown in Figure 5.3 for the parameters D, w_{tea}, J and s (μ is neglected because it does not show new dynamics).
Most solvers are calculating the profiles correctly and with less evaluations than the benchmark. Only the Euler method (and `ode23s` in one case) needs more NoE. But this is again explained by the static step size.
However the DAE solver IDAS shows in this case some sensitivities concerning the identifiable parameters, because it does not calculate the whole profile of the identifiable parameters. In this case the step size of the solver converges to zero until it stops due to a too small step size. In the figure it seems as if only one point is calculated. These convergence issues of the IDAS solver might be explained by an inconsistent initialization of $\dot{\zeta}_0$ due to the resolving of the occurrence of `NaN` and/or `Inf` (see subsection 4.2.2).
This stands in contrast to the suggestion that the DAE solvers are superior to the ODE

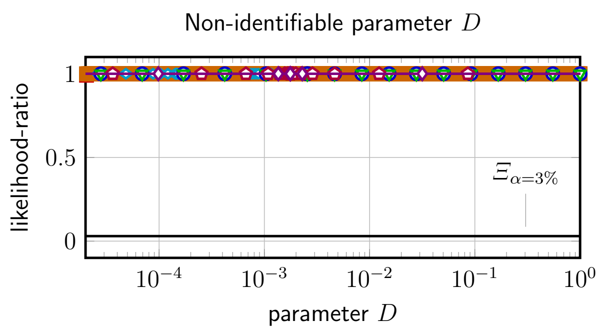
solvers, because the inversion of the matrix is only necessary once and not in every step. Interesting is, that again the ODE solver CVODE and `ode15s` for ODEs show the best performance. They moreover seem stable despite the singularity of the mass matrix.
Significant is the reduction of the NoE which is achieved in the case of the non-identifiable parameters. The benchmark needs $10,000$ evaluations to calculate the profile of the parameter D. For the same parameter the CVODE only needs 27, which is an immense saving of computational resources, especially in PDEs.
Even if the profiles for the parameters w_{tea} and s do not show such a big step size, they match the benchmark nicely and reduce the NoE at least by one magnitude. The gain is not as significant as for the non-identifiable parameters, because the classical method already does not need as many evaluations as for the non-identifiable parameters.

Substituting the identity for the hessian

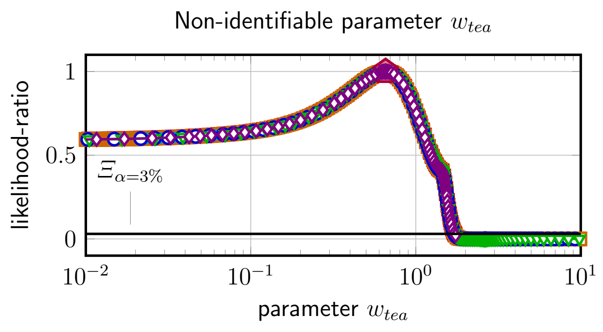
As it is already shown in 4.5.3, if the optimization constraint is defined with logarithmic parameters, the hessian can be substituted with the identity and the integration based method is still producing accurate results.
This is again proven by the profiles in Figure 5.4. However there is no actual gain in comparison to the case, where the profile is calculated with the provided hessian, which is the FIM. Especial in the non-identifiable case does the calculation with the FIM need less NoE than with the identity matrix as the substitute.
Although the calculation with the FIM is in this case better, the identity matrix should not be neglected. Because it can be applied in the cases where the FIM can not be used or if an investigation of the profiles is wanted instantly, without the previous implementation of the FIM as the hessian of the log-likelihood.

5.3 Influence of the logarithmic scale

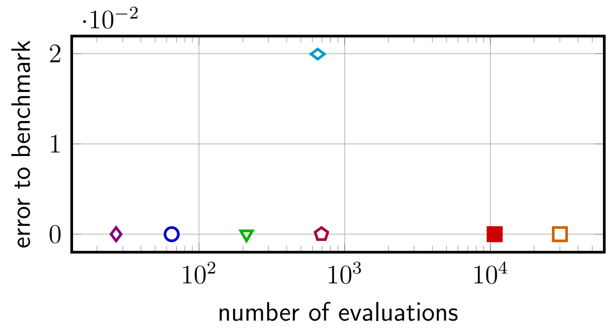
Defining the optimization constraint directly in the logarithmic scale was previously shown to be faster and more accurate. However there is another reason, which speaks for this approach.
In the case of the *pom1p* model the calculation actually fails, if the first approach of the logarithmic scale is applied.
Figure 5.5-5.6 show the profiles of the *pom1p* model, evaluated with various solver types and the constraint defined in the initial parameters. At first consider the Euler method (in Figure 5.5 for example). In this model the Euler method does compute the profile with step size $= 10^{-3}$ with a reasonable error, but needs more evaluations than the benchmark. But the step size of this solver can be increased to 10^{-2} without a great increase in the error, which results in a smaller number of evaluations than the benchmark.
However, the Euler method is the only one of the implemented solver types, which calculates the profile complete and correctly. The performance of the other solvers is various considering the different parameters.
So it seems that in Figure 5.5a the solvers `ode15s` for DAEs, `ode15s` for ODEs and `ode23s` all reflect the non-identifiability of the parameters D, but in the negative direction they all stop before they were reaching the interval end approximately at the same



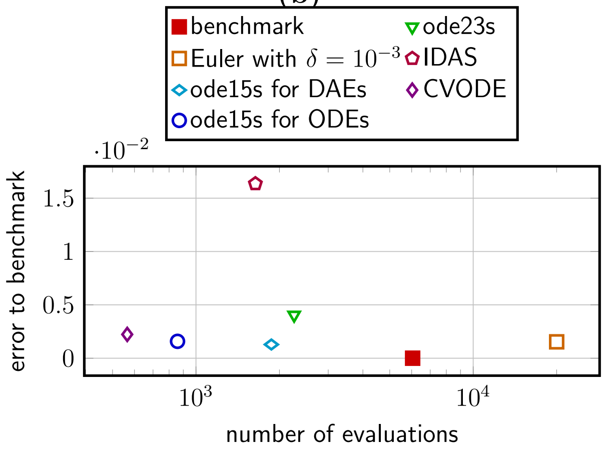
(a)



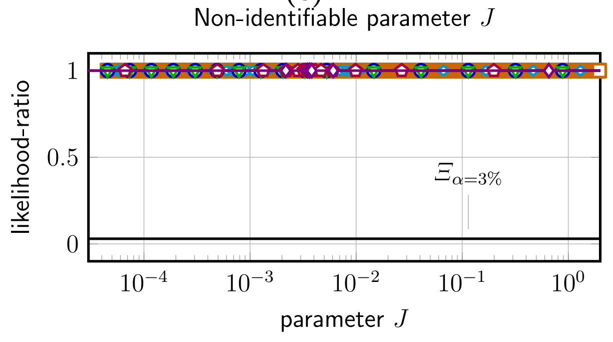
(b)



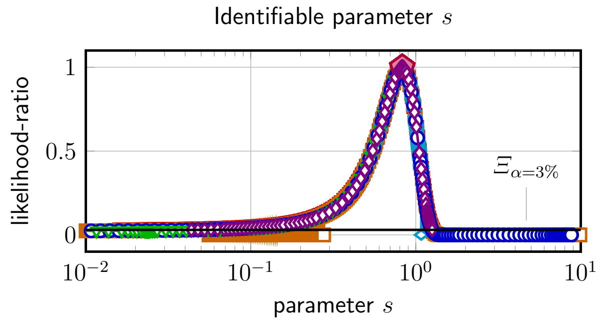
(c)



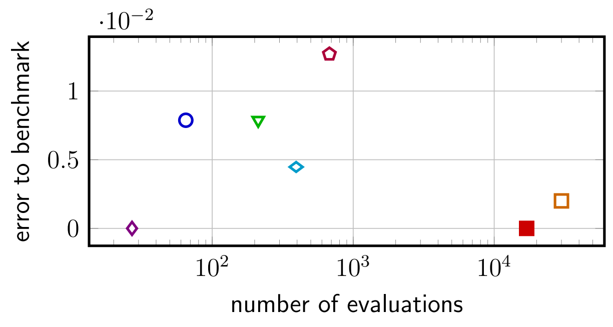
(d)



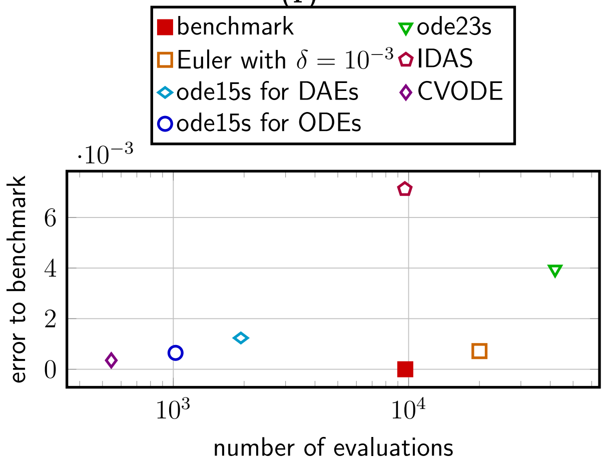
(e)



(f)

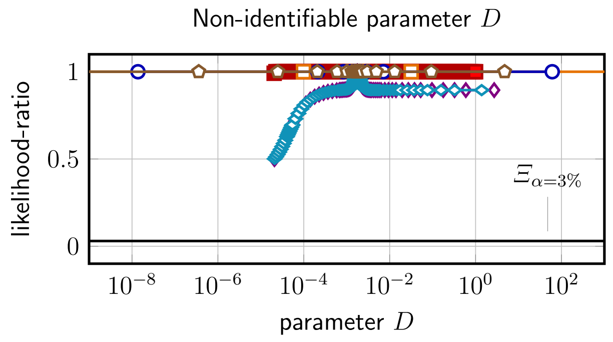


(g)

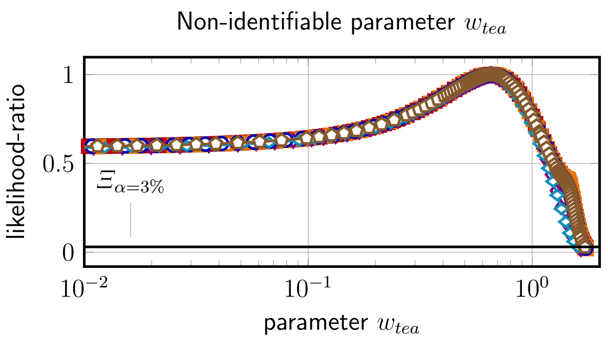


(h)

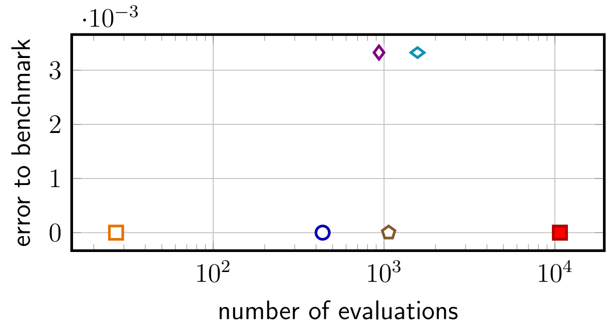
Figure 5.3: Profiles of the *pom1p* model for the parameters D, w_{tea}, J and s, calculated with the classical method (benchmark) and the integration based method with $\gamma = 0$ using various solvers to investigate their performance. The optimization constraint is in terms of logarithmic parameters (see 3.12).



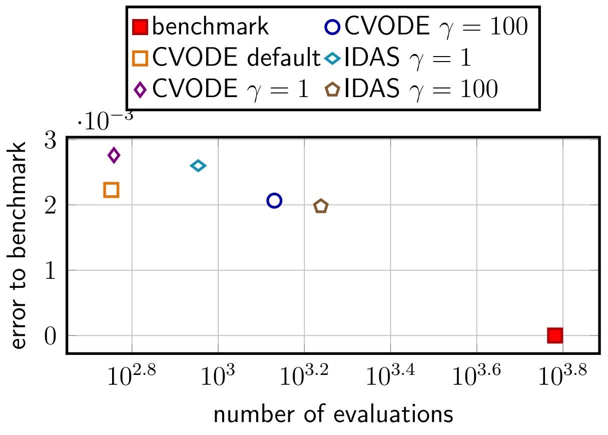
(a)



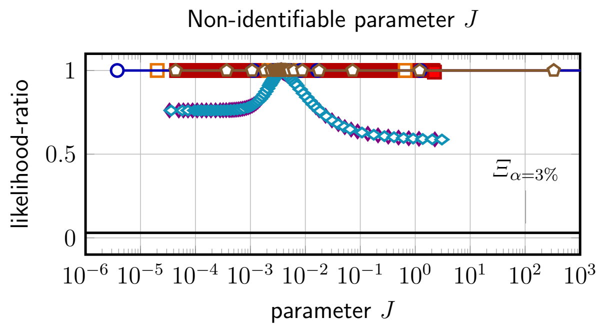
(b)



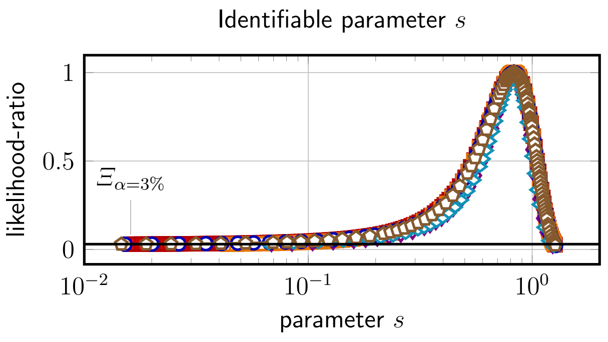
(c)



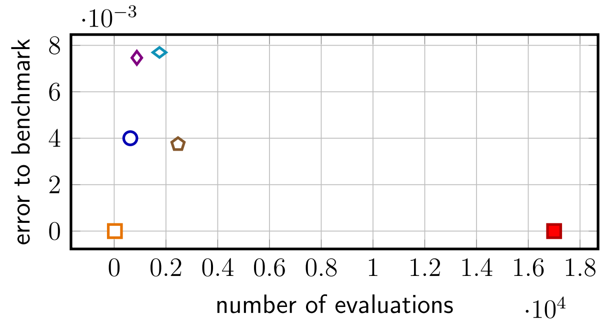
(d)



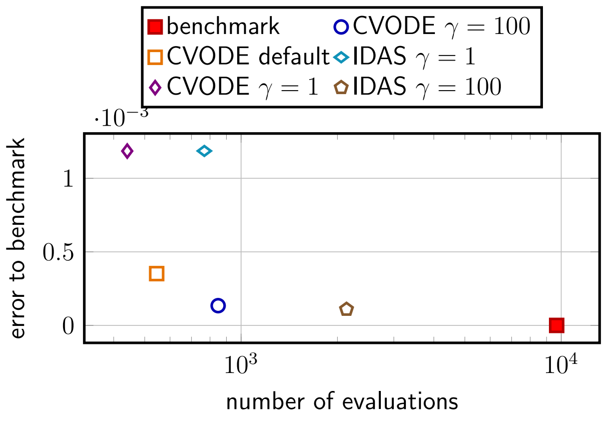
(e)



(f)



(g)



(h)

Figure 5.4: Profiles of the *pom1p* model for the parameters D, w_{tea}, J and s, calculated with the classical method (benchmark) and the integration based method with $\gamma = 0$ using CVODE (options: default) and IDAS (options: default). The calculation is done with the provided gradient, the identity matrix as the hessian and $\gamma = 100$. The optimization constraint is in terms of logarithmic parameters (see 3.12).

point. The same behaviour occurs at certain points for the other non-identifiable parameters (see Figure 5.5e) as well as for the quasi- and identifiable parameter (5.5b and 5.5f). Also the profiles calculated with the CVODE at the same points do not converge (see Figure 5.6a, 5.6b, 5.6e and 5.6f).
Only IDAS with tolerances set to 10^{-10} and the linear solver `Dense` is calculating the profile for the non-identifiable parameters D, J and μ, however it fails completely for the parameters s and w_{tea}. IDAS with the default options only calculates the profile for w_{tea}, but stops at the same point as the CVODE. Using another linear solver or different tolerances only results in the general non convergence of the solver.
Concerning the point, where the MATLAB built-in solvers fail, it happens, that the constraint optimization reaches a set of parameters which does not suffice for the model, since it is producing `NaN` or `Inf` as the output at that point.
For the CVODE it does not happen that `NaN` or `Inf` occur, but it seems as if the calculation reaches an invisible bound. That means the step-wise calculation of the CVODE jumps back and forth, not stopping and not continuing either. This suggests that the calculation does not have a definite direction.
The question is, where does these convergence issues come from.
Recalling section 3.1 the gradient of the parameter function defines the direction of the calculation. Now, this gradient is at the particular point quite small, so that the direction pointer is compared to the other entries in the mass matrix not of big influence any more (e.g. $\| \nabla_\theta g(\theta) \| < 10^{-4}$ in the case of the parameter D).
However, that the gradient of the parameter function does have such a small value is only because the parameter function was rewritten in the logarithmic scale (see 3.14) and therefore the gradient is dependent on the parameter values.

This suggests that this approach of the logarithmic scale is not recommendable in the integration based calculation of the profile likelihood and the other method, where the constraint is defined in the logarithmic scale with a linear parameter function should be used.

Influence of the correction term

It was already shown with the comparison of various solvers that the second approach of the logarithmic scale is favourable, but it was additional tested how the profile calculation of the *pom1p* model performs with the first approach of the logarithmic scale, if the hessian is substituted with the identity matrix.
The results in chapter 4 and in [1] suggested, that the factor γ only has to be increased to a certain extent to gain an accurate profile.
However, in this case the model *pom1p* shows that this approach is not always advisable. The corresponding results are shown in Figure 5.7 and suggest that the identity with a high correction factor γ is only usable for identifiable or at least quasi-identifiable parameters, because the calculation produces quite different results for the parameters D, J and μ (μ not shown). For the parameters D and J (in Figure 5.7a and 5.7e) the calculation even turns and runs backwards.

All in all the substitution of the hessian with the identity does in this case not improve

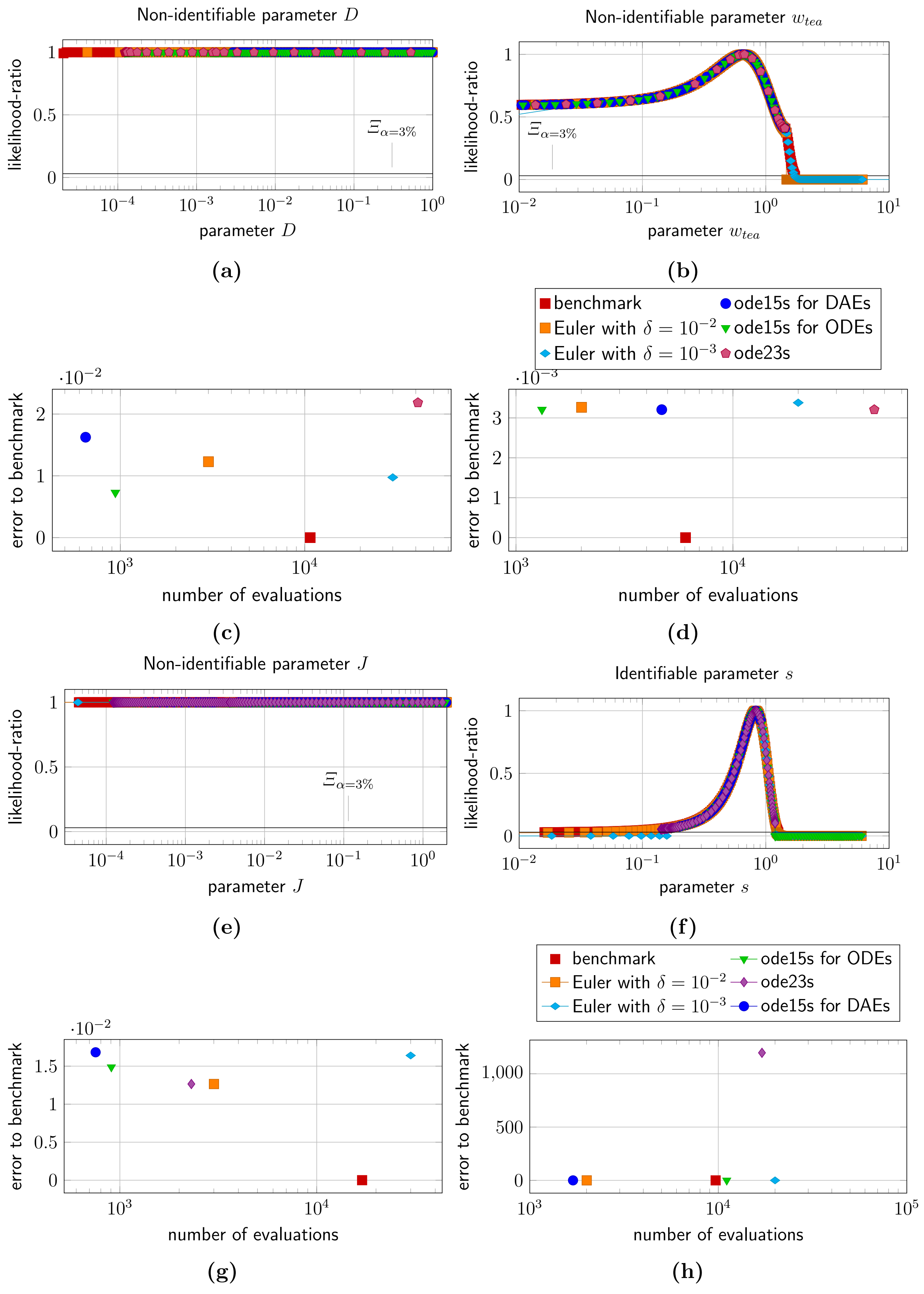


Figure 5.5: Profiles of the *pom1p* model for the parameters D, w_{tea}, J and s, calculated with the classical method (benchmark) and the integration based method with $\gamma = 0$ using the solvers: Euler method (step sizes $\delta = 10^{-2}$ and 10^{-3}), `ode15s` for DAEs, `ode15s` for ODEs and `ode23s`. The optimization constraint is in terms of initial parameters (see 3.10).

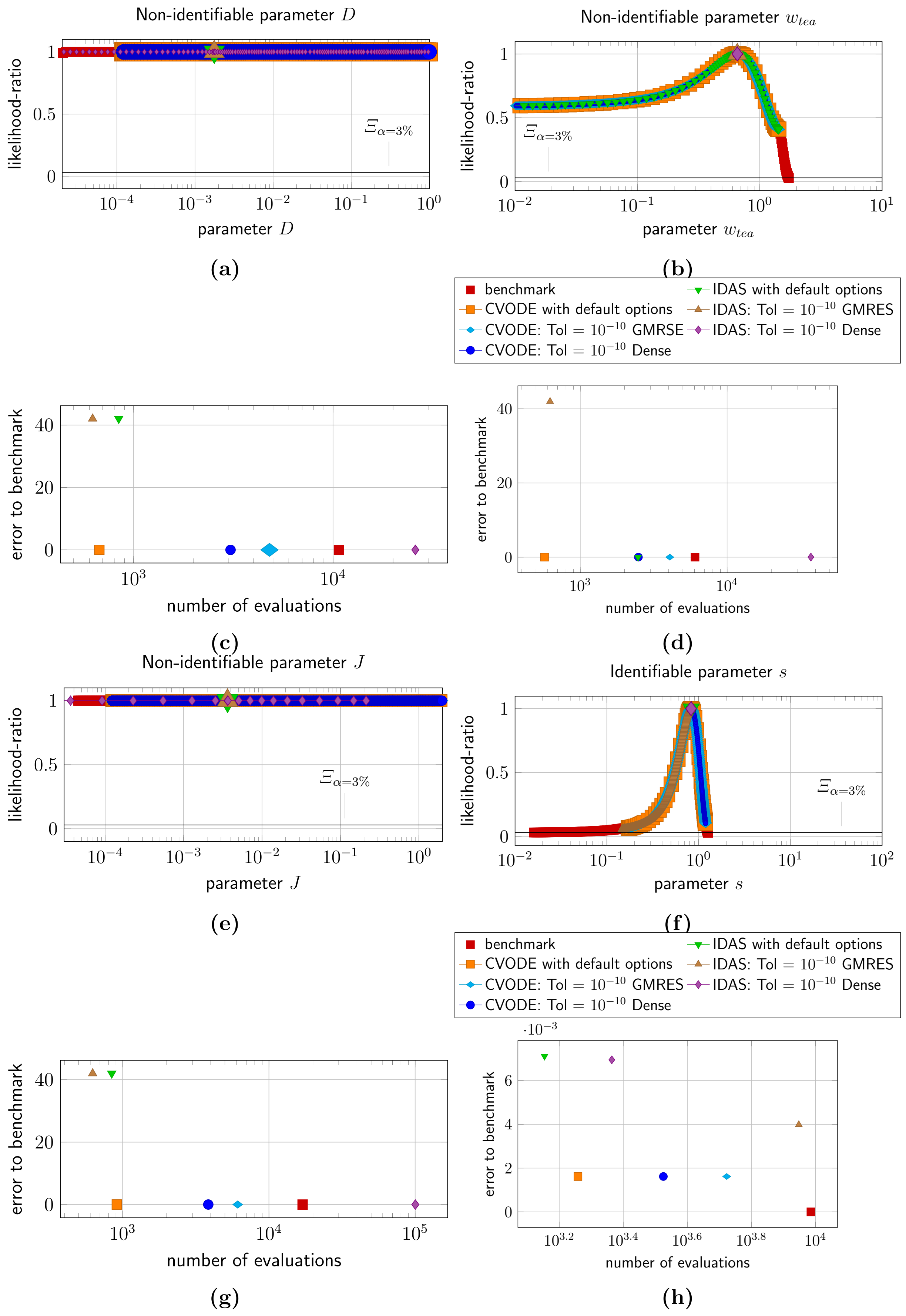
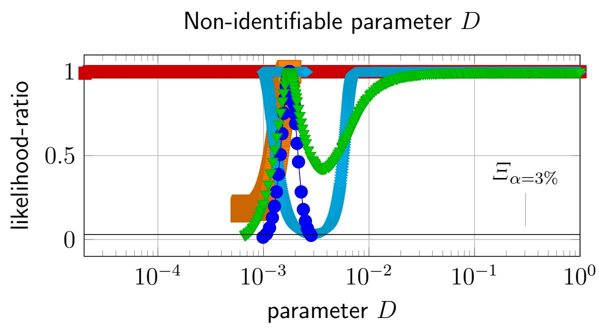


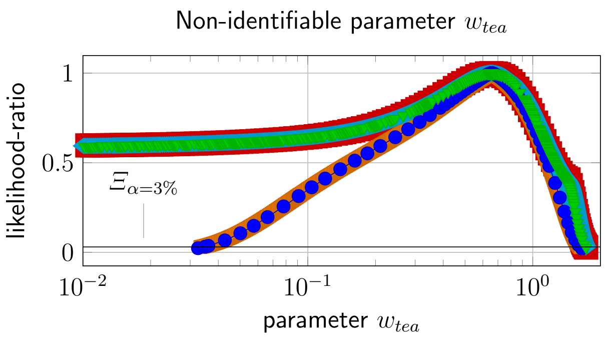
Figure 5.6: Profiles of the *pom1p* model for the parameters D, w_{tea}, J and s, calculated with the classical method (benchmark) and the integration based method with $\gamma = 0$ using the solvers: IDAS and CVODE with default options and the absolute and relative tolerances set to 10^{-10} either with the linear solver `GMRES` and `Dense`. The optimization constraint is in terms of initial parameters (see 3.10).

the calculation significantly even with a high correction term and that this correction term can also badly corrupt the calculation. Therefore this is no alternative to the second approach of the logarithmic scale, which calculates the profile likelihood in a vastly more accurate and faster way.

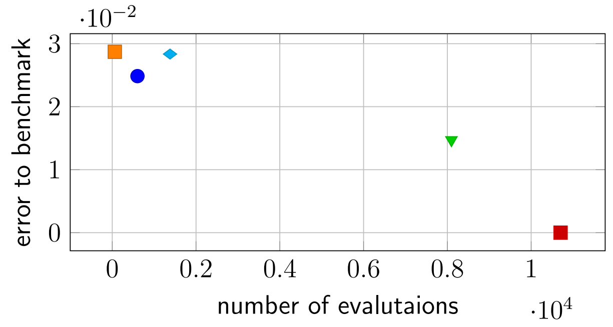
Hence, the *pom1p* model confirms the earlier recommendation of the second approach of the logarithmic scale, due to the sensitivities in the profile calculation, which can occur if the first one is applied.



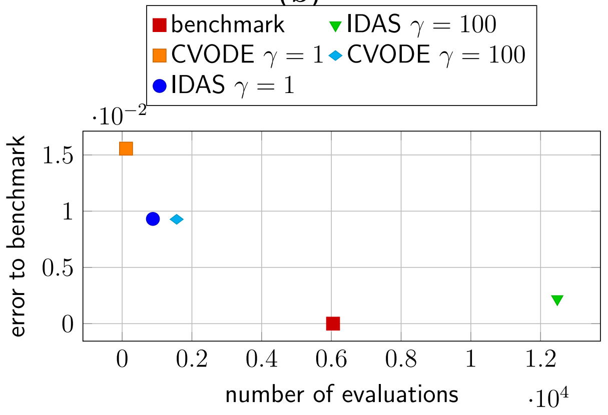
(a)



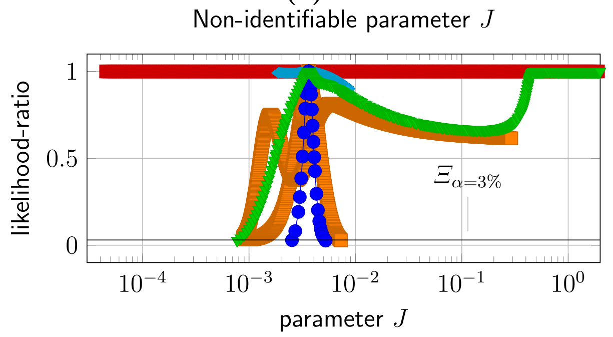
(b)



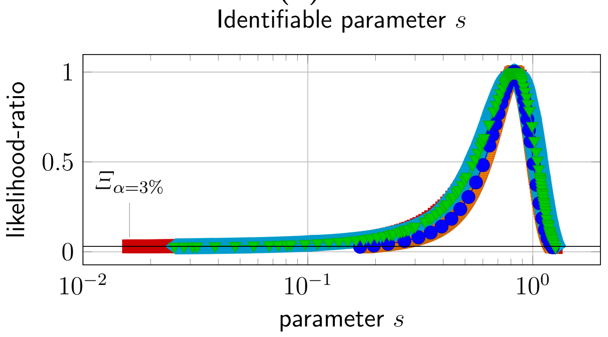
(c)



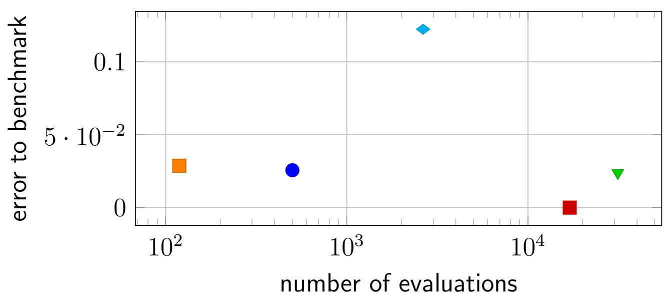
(d)



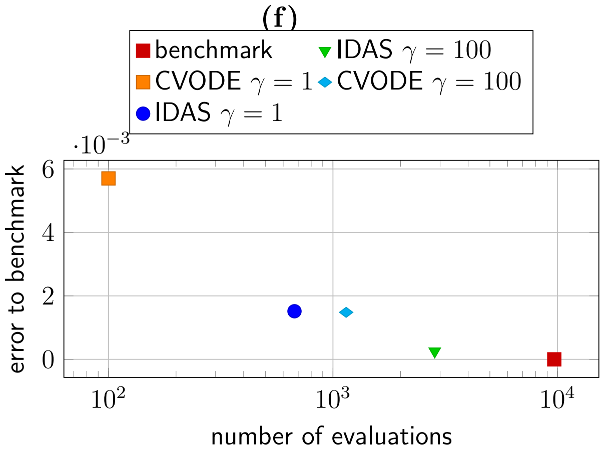
(e)



(f)



(g)



(h)

Figure 5.7: Profiles of the *pom1p* model for the parameters D, w_{tea}, J and s, calculated with the classical method (benchmark) and the integration based method with $\gamma = 0$ using the solvers: CVODE or IDAS (linear solver `Dense` and tolerances set to 10^{-10}) with the hessian substituted by the identity and $\gamma = [0, 1, 100]$. The optimization constraint is in terms of initial parameters (see 3.10). In (a) CVODE and $\gamma = 100$ (cyan diamonds) turns at about 10^{-3} and runs backwards. In (e) CVODE and $\gamma = 1$ (orange squares) turns at about 10^{-3} and runs backwards.

Chapter 6

Conclusion and Outlook

The focus of this thesis is the assessment of the integration based method suggested by [1], which was proposed to improve the profile calculation compared to the classical method in terms of computation time. One main part was thereby to implement the algorithm with various solvers for the differential equation and different approximations methods or substitute candidates for the derivatives. The implementation was then tested with various models with different number of parameters and especially models, which included non-identifiable parameters. The considered models were incorporating additive normal noise, but the method is independent on the actual derivation of the log-likelihood function, which was shown by using the SND as a test function completely independent of an underlying model. Moreover the advantage just to change the parameter function in the case of prediction profile likelihoods was exposed, which can reduce the risk of analytical mistakes during the reparametrization.
The goal of the thesis, to provide a function, which calculates the profile likelihoods for the identifiability analysis faster than the classical method, is fully achieved. The implemented method needs significantly less NoE in the shown examples and calculates the profiles as accurate as the classical method, given that sufficient substitutes for the derivatives were available. Thereby is a realization, that an accurate approximation of the gradient is not as important as the approximation of the hessian. Applications show good results with the use of the FIM and a rough approximation for the gradient of the model.
The most significantly reduction of the NoE is achieved in the case, when the optimization constraint is defined with the logarithmic parameters (the second approach of the logarithmic scale). Particularly noteworthy is the gain for non-identifiable parameters, where the classical method needs the most NoE. In some cases the NoE can even be reduced to as few as 1% of the evaluations of the benchmark.
Even if the actual time of the computation was not part of the method assessment, an example considering the runtime of the algorithm should visualize the benefit of the integration based method:
Using the classical method (i.e. `computeProfiles` by Jan Hasenauer) the computation of the likelihood profiles for all parameters in the *pom1p* model takes on a certain machine around 25 minutes. That is on average 5 minutes per parameter, in contrast to the implemented integration based method, which needs less than half a minute for all parameters together using the best setting.

The second approach of the logarithmic scale does also improve the accuracy of the algorithm in a way, that the identity as the substitute of the hessian is useful, too. The correction factor γ still needs to be adjusted to a high value, but the logarithmic scale resolves the challenges of the resulting stiff system and is then calculating the profile likelihood comparable fast and accurate.
Even if a wide range of different solvers and optional inputs were applied to the system, the optimal choice, is difficult to state, since there are infinitely many combinations of solver, derivation approximations, and other options, e.g. solver tolerances. For future users a recommendation is, to apply the second approach of the logarithmic scale for the parameter function and use the CVODE with the default options, because this combination performed best in the applied examples. If convergence issues occur IDAS or an other DAE solver can be applied.
The identity as the hessian, should only be used, with the second approach of the logarithmic scale and a high factor γ for the correction term. For a faster computation it is, however, favourable to neglect the correction term completely and give more attention to an accurately approximation of the hessian.
A decision guidance to choose the inputs to apply the algorithm is shown in Figure 6.1, however it depends on the model, which solver options do work the best and the guidance does not reflect all possible variations.
The documentary of the implemented function can be found in the appendix and the complete MATLAB code is appended on a CD for further use.
In conclusion, the implemented function `computeApproximatedProfiles` can be used as a replacement for the classical method, because it saves evaluations of the cost function. Moreover it can be used to investigate the confidence intervals for a combination of parameters, without requiring the reparametrization.

What still needs to be done is, to investigate why the ODE solvers show a better performance, than the DAE solvers, especially in the case of a singular mass matrix and even if for the DAE approach the matrix does not need to be inverted in every step.
Additionally there are infinitely many possible input variations, due to the mere fact that the tolerances of the solvers can be set freely and there might be combinations for which the algorithm is optimal, but this search would exceed this masters thesis. Especially an automatic generated decision for the solver and its tolerances would be a further improvement of the algorithm.
Furthermore, the considered models with a maximum of 5 parameters were still of a small dimension. It would be interesting to know, how the performance of the algorithm changes for higher dimensional models.

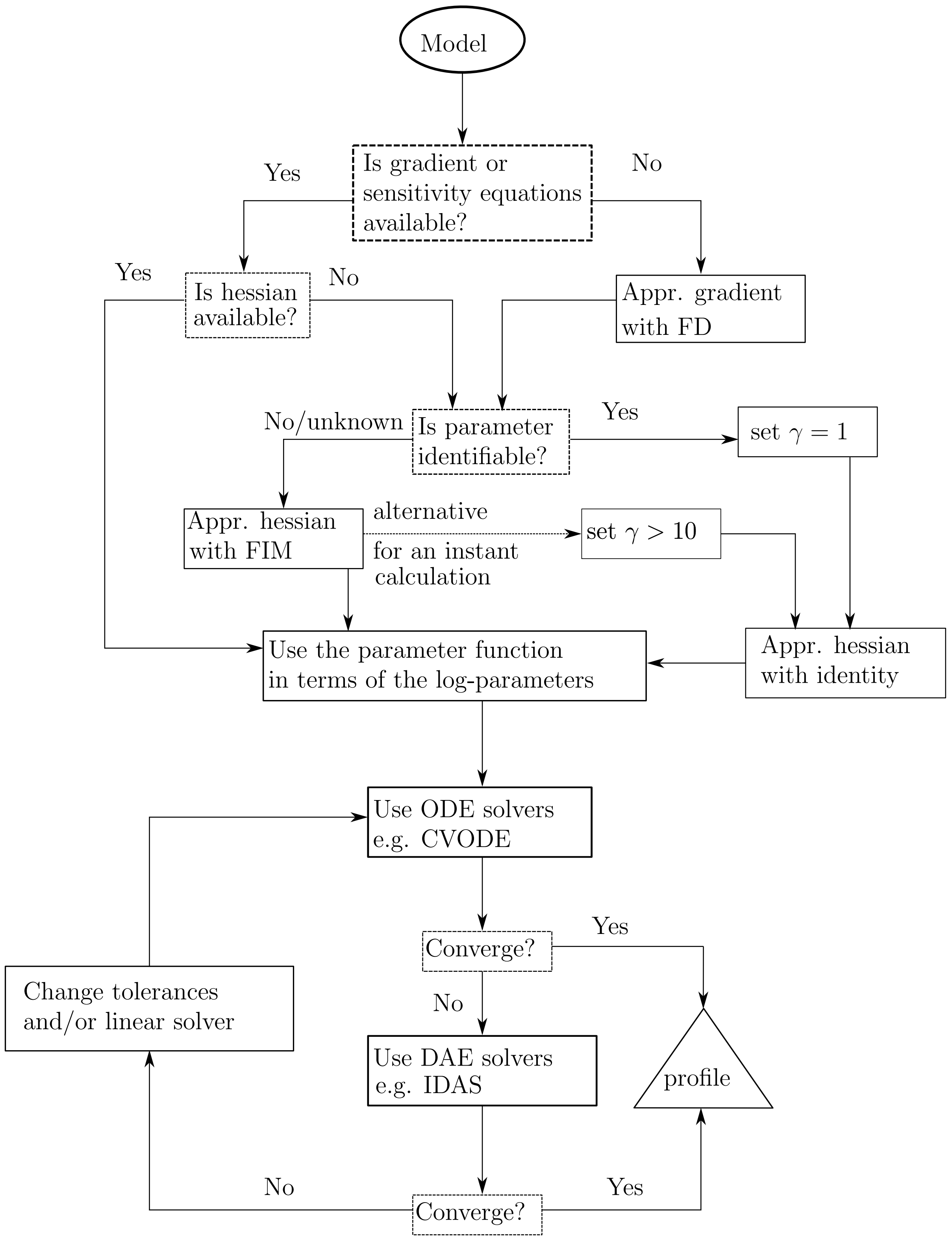

Model
Is gradient or sensitivity equations available?
Yes
No
Is hessian available?
Yes
No
Appr. gradient with FD
Is parameter identifiable?
No/unknown
Yes
set $\gamma = 1$
Appr. hessian with FIM
alternative
for an instant calculation
set $\gamma > 10$
Use the parameter function in terms of the log-parameters
Appr. hessian with identity
Use ODE solvers e.g. CVODE
Converge?
Yes
No
Change tolerances and/or linear solver
Use DAE solvers e.g. IDAS
profile
Converge?
No
Yes

Figure 6.1: Guidance to choose inputs and solvers of the implemented function `computeApproximatedProfiles`

Bibliography

[1] J.-S. Chen and R. I. Jennrich, “Simple accurate approximation of likelihood profiles,” *Journal of Computational and Graphical Statistics*, vol. 11, no. 3, pp. 714–732, 2002.

[2] A. Raue, C. Kreutz, T. Maiwald, J. Bachmann, M. Schilling, U. Klingmüller, and J. Timmer, “Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood,” *Bioinformatics*, vol. 25, no. 15, pp. 1923 – 1929, 2009.

[3] A. Raue, C. Kreutz, F. J. Theis, and J. Timmer, “Joining forces of bayersian and frequentist methodology: A study for inference in the presence of non-identifiability,” *Article to appear in Philosophical Transactions of the Royal Society A*, 2013.

[4] S. Hock, J. Hasenauer, and F. J. Theis, “Modeling of 2d diffusion processes based on microscopy data: Parameter estimation and practical identifiability analysis,” *BMC Bioinformatics*, vol. 14(Suppl 10), p. S7, 2013.

[5] C. Kreutz, A. Raue, D. Kaschek, and J. Timmer, “Profile likelihood in systems biology,” *FEBS Journal*, vol. 280, pp. 2564–2571, 2013.

[6] P. Weber, J. Hasenauer, F. Allgöwer, and N. Radde, “Parameter estimation and identifiability of biological networks using relative data,” *Proceedings IFAC World Congress*, vol. 18(1), pp. 11648–11653, 2011.

[7] L. Ljung and T. Glad, “On global identifiability for arbitrary model parametrizations,” *Automatica*, vol. 30, no. 2, pp. 265–276, 1994.

[8] S. A. Murphy and A. W. van der Vaart, “On profile likelihood,” *Journal of the American Statistical Association*, vol. 95, no. 450, pp. 449–465, 2000.

[9] S. L. Quinn, D. W. Bacon, and T. J. Harris, “Notes on likelihood intervals and profiling,” *Communications in Statistics - Theory and Methods*, vol. 29, no. 1, pp. 109 – 129, 2000.

[10] K. Königsberger, *Analysis 2.* Springer-Verlag, 1993.

[11] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* Society for Industrial and Applied Mathematics, 1998.

[12] The MathWorks, Inc., 3 Apple Hill Drive; Natick, MA 01760-2098, *MATLAB Documentation Mathematics.*

[13] C. Woodward, A. Hindmarsh, and R. Serban, "*sundials*: Suite of nonlinear and differential algebraic equation solver." Web, March 2012. Nonlinear Solvers and Differential Equations Project.

[14] R. P. Dickinson and R. J. Gelinas, "Sensitivity analysis of ordinary differential equation systems – a direct method," *Journal of Computational Physics*, vol. 21, pp. 123–143, 1976.

[15] M. Caracotsios and W. E. Stewart, "Sensitivity analysis of initial-boundary-value problems with mixed pdes and algebraic equations: Applications to chemical and biochemical systems," *Computers & Chemical Engineering*, vol. 19, pp. 1019–1030, 1995.

[16] J. D'Errico, "Derivestsuite: Adaptive robus numerical differentiation." Web, June 2011.

[17] E. L. Lehmann, *Theory of Point Estimation.* Wadsworth & Books/Cole, 1991.

[18] B. Efron and D. V. Hinkley, "Assessing the accuracy of the maximum likelihood estimator: Observed versus expected fisher information," *Biometrika*, vol. 65, no. 3, pp. 457–487, 1978.

[19] D. Faller, U. Klingmüller, and J. Timmer, "Simulation methods for optimal experimental design in systems biology," *Simulation*, vol. 79, pp. 717–725, 2003.

[20] T. E. Saunder, K. Z. Pan, A. Angel, Y. Guan, J. V. Shah, M. Howard, and F. Chang, "Noise reduction in the intracellular *Pom1p* gradient by a dynamic clustering mechanism," *Developmental Cell*, vol. 22, pp. 558–572, 2012.

[21] O. Hachet, M. Berthelot-Grosjean, K. Kokkoris, V. Vincenzetti, and J. Moosbrugger, "A phosphorylation cycle shapes gradients of the dyrk family kinase pom1 at the plasma membrane," *Cell*, vol. 145, pp. 1116–1128, 2011.

Appendix A

Documentation of implemented functions

Library of the implemented functions to calculate the profile likelihood using the integration based method.

A.1 computeApproximatedProfiles.m

Documentation of the function `computeApproximatedProfiles`.

Syntax

```
parameters = computeApproximatedProfiles(parameters, logPosterior, ...
                parameterFunction)
parameters = computeApproximatedProfiles(parameters, logPosterior, ...
                parameterFunction, options)
[parameters, fh] = computeApproximatedProfiles(...)
```

Description

`computeApproximatedProfiles` is calculating the profile likelihood of a parameter or a combination of parameters starting at the MLE.

Usage:

```
parameters = computeApproximatedProfiles(parameters, logPosterior, ...
                parameterFunction)
```

The function starts at the MLE and calculates the profile likelihood within the interval given by the minimal and maximal values of the parameters using the default options. It returns the updated parameters struct.

```
parameters = computeApproximatedProfiles(parameters, logPosterior, ...
                parameterFunction, options)
```

The function starts at the MLE and calculates the profile likelihood within the interval given by the minimal and maximal values of the parameters using the specified optional arguments. It returns the updated parameters struct.

```
[parameters, fh] = computeApproximatedProfiles(...)
```

The function returns additional to the parameters struct the figure handle `fh`

Input Arguments

The input arguments describes the arguments passed to `computeApproximatedProfiles` except the options values, see A.1 for these.

`parameters`	Struct containing parameter values. Must consist of `parameters.MS.MAP.par`, `parameters.min` and `parameters.max`
`logPosterior`	Positive log-posterior of model as function of the parameters. Returns additionally if possible the gradient or the gradient and hessian. Allows for a second input which determines sign and scale of the log-posterior. If gradient and hessian of the log-posterior can also be provided by the `logPosterior` function, it needs to be provided like: `[y,dy,ddy]=@logPosterior`
`parameterFunction`	Parameter function as function of the parameters. It allows for a second input `optionsPF`, which defines scale and other options of the parameter function. It determines the desired parameter or function of parameters for which the profile is to be calculated.

Output Arguments

`parameters`	Updated parameter object, containing:
`.profile.par`	Parameters along profile; the first entry is the profiled parameter, for a non-linear parameter function this the combination of parameters).
`.profile.ratio`	Likelihood ratio along the profile.
`.profile.logPost`	Maximum log-posterior along profile.
`fh`	Figure handle of figure where the profile was plotted

Options

`options`	Options struct containing the options:
`.solver`	Defines the used solver:
`.type`	'ode23s' |'ode15s' |'ode15sODE' |{'IDAS'} |'CVODE'
`.options`	Sets options of the solver (optional)
`.gm`	Defines the influence of the correction term, can be set to any real number, 1
`.plot`	Determines if a plot during the computation is made 'false' |{'true'}
`.fh`	Figure handle. If no figure handle is provided, a new figure is created
`.P.min`	Lower bound for profiling parameters, having same dimension as the parameter vector (default = `parameters.min`)
`.P.max`	Upper bound for profiling parameters, having same dimension as the parameter vector (default = `parameters.max`)
`.R_min`	Minimal ration down to which the profile is calculated (default = 0.03).
`.stepMin`	Minimal step size of the DAE Solver, stops if smaller (default = 0)
`.opt_logPosterior`	Options for the log-posterior (log-likelihood) function
`.sign`	Determines whether the value of the positive (.sign = 'positive') or the negative (.sign = 'negative') log-posterior is provided. (default = 'positive')
`.scale`	Determines scale of the computation (default = 'lin')
`.grad`	Determines if gradient is provided by the log-posterior (log-likelihood) function `false` |{'true'}
`.grad_appr`	Provides approximation details of the gradient if it is not provided.
`.type`	Determines how the gradient is approximated: {'FinDif'} |'Derivest'
`.stepsize`	Approximation step size in the case of FD (default = 10^-4)
`.hess`	Determines if hessian is provided by the log-posterior (log-likelihood) function `false` |{`true`}
`.hess_appr`	Provides approximation details of the hessian if it is not provided.
`.type`	Determines how the hessian is approximated {'FinDif'} |'Identity' |'Derivest'
`.stepsize`	Approximation step size in the case of FD (default = 10^-4)

`.opt_paramFunc`	Options for the parameter function
`.optionsPF`	Determines the options of the parameter function; if the standard parameter function is used optionsPF.number defines the number of parameters, optionsPF.index the parameter the profile is calculated of and optionsPF.scale determines if the function is linear '`lin`' or logarithmic '`log`' (if the parameter function is directly defined in the logarithmic parameters it is a linear function ⇒ '`lin`')
`.grad`	Determines if gradient is provided by the log-posterior (log-likelihood) function '`false`' |{'`true`'}
`.grad_appr`	Provides approximation details of the gradient if it is not provided.
`.type`	Determines how the gradient is approximated: {'`FinDif`'} |'`Derivest`'
`.stepsize`	approximation step size in the case of FD (default = 10^-4)
`.hess`	Determines if hessian is provided by the log-posterior (log-likelihood) function '`false`' |{'`true`'}
`.hess_appr`	Provides approximation details of the hessian if it is not provided.
`.type`	Determines how the hessian is approximated {'`FinDif`'} |'`Identity`' |'`Derivest`'
`.stepsize`	Approximation step size in the case of FD (default = 10^-4)

A.2 approximatehessian.m

Documentation of the function `approximatehessian`.

Syntax

```
ddy = approximatehessian(theta, logPosterior)
ddy = approximatehessian(theta, logPosterior, ...
                options)
```

Description

`approximatehessian` is approximating the hessian at the point `theta` using different approximations.

Usage:

```
ddy = approximatehessian(theta, logPosterior)
```

The function approximates the hessian `ddy` of the function `logPosterior` at the point `theta` and is using the default options.

```
ddy = approximatehessian(theta, logPosterior, options)
```

The function approximates the hessian `ddy` of the function `logPosterior` at the point `theta` and is using the user supplied options.

Input Arguments

The input arguments describes the arguments passed to `approximatehessian` except the options values, see A.2 for these.

`theta`	Parameter vector; point where the hessian is evaluated.
`logPosterior`	Log-posterior of model as function of the parameters. Returns additionally if possible the gradient. Allows for a second input which determines sign and scale of the log-posterior.

Output Arguments

`ddy`	Square matrix with dimension equal the length of `theta` Approximated hessian of the function `logPosterior`

Options

`options`	Options struct to determine additional options
`.type`	Defines the type of the approximation: ’`Identity`’ |’`Derivest`’ |{’`FinDif`’}
`.stepsize`	Sets the step size for the approximation with FD, {`1e−2`}

A.3 approximategradient.m

Documentation of the function `approximategradient`.

Syntax

```
dy = approximategradient(theta, logPosterior)
dy = approximategradient(theta, logPosterior, options)
```

Description

`approximategradient` is approximating the gradient at the point `theta` using different approximations.

Usage:

```
dy = approximategradient(theta, logPosterior)
```

The function approximates the gradient `dy` of the function `logPosterior` at the point `theta` and is using the default options.

```
dy = approximategradient(theta, logPosterior, options)
```

The function approximates the gradient `dy` of the function `logPosterior` at the point `theta` and is using the user supplied options.

Input Arguments

The input arguments describes the arguments passed to `approximategradient` except the options values, see A.3 for these.

`theta`	Parameter vector; point where the gradient is evaluated.
`logPosterior`	Log-posterior of model as function of the parameters. Returns additionally if possible the gradient. Allows for a second input which determines sign and scale of the log Posterior.

Output Arguments

`dy`	Vector of the length of `theta`. Approximated gradient of the function `logPosterior`

Options

`options`	Options struct to determine additional options
`.type`	Defines the type of the approximation: ’`Derivest`’ |{’`FinDif`’}
`.stepsize`	Sets the step size for the approximation with FD, {`1e−2`}

A.4 parameterFunction.m

Documentation of the function `parameterFunction`.

Syntax

```
g = parameterFunction(theta, index)
g = parameterFunction(theta, index, options)
[g, dg] = parameterFunction(...)
[g, dg, ddg] = parameterFunction(...)
```

Description

`parameterFunction` is determining the parameter, for which the profile is computed using the function `computeApproximatedProfile`.

Usage:

```
g = parameterFunction(theta, index)
```

The function determines the parameter, for which the profile is computed, using default options.

```
dy = parameterFunction(theta, index, options)
```

The function determines the parameter, for which the profile is computed, using the user supplied options.

Input Arguments

The input arguments describes the arguments passed to `parameterFunction` except the options values, see A.4 for these.

`theta`	Parameter vector; point where the parameter function is evaluated.
`index`	Index of parameter, of which the profile is computed.

Output Arguments

`g`	One-dimensional output, equal the parameter value of profiled parameter.
`dg`	Gradient vector of parameter function, is of same dimension as `theta`.
`ddg`	Hessian matrix of parameter function, square matrix is of same dimension as `theta`.

Options

`options`	Options struct to determine additional options
`.number`	Defines the number of parameters and therefore the dimension of the derivatives, {`length(theta)`}
`.scale`	Sets the scale of the parameter function, allowed to be different to the `logPosterior` scale in `computeApproximatedProfile`, {’`lin`’}

A.5 explicitEuler.m

Documentation of the function `explicitEuler`.

Syntax

```
[t,x] = explicitEuler(fun, int, x_0, lam)
```

Description

`explicitEuler` is applying the explicit Euler method to solve an explicit ODE system.

Usage:

```
g = explicitEuler(fun, [t_0, t_end], x_0, lam)
```

The function calculates in the interval `int` and with the step size `lam`, the solution of the initial value problem consisting of the explicit ODE system `fun` and the initial point `x_0`.

Input Arguments

The input arguments describes the arguments passed to `explicitEuler`.

`fun`	explicit ODE system of the form: `x_dot = fun(t, x)`
`int`	Interval where the solution is calculated of the form `[t_0, t_end]` with `t_0` be the initial point and `t_end` the end point.
`x_0`	Initial condition vector of the initial value problem at `t_0`
`lam`	Step size of the Euler method

Output Arguments

`t`	Column vector of time point .
`x`	Solution array. Each row of `x` corresponds to the solution at the corresponding row in `t`.