Rigorous parameter estimation for noisy mixed-effects models

Alexander Danis, Sebastian Ueckert, Andrew Hooker, and Warwick Tucker
Uppsala University
Sweden

February 28, 2011

Abstract

We describe how set–valued techniques can be used to reconstruct model parameters from noisy data. The main algorithm combines a branch and bound procedure with a data expansion step. The set–valued results are transformed into point clouds, after which statistical properties can be retrieved. We apply the presented method to two mixed-effects models.

1 Introduction

Finitely parameterized mathematical models are used to describe, explain, summarize, and predict the behavior of physical, biological, and economical systems. A parameter estimation problem is a problem of finding a set of parameter values that makes the model function fit the experimental data. Incomplete approaches to this problem search for one solution in parameter space; complete approaches search for all. In this article, we describe a complete approach – a general-purpose solution strategy based on set–valued computations and global search algorithms. We begin by describing the basic computational components which involves set–valued computations, directed acyclic graphs, constraint propagation, and data expansion. We end the article with some examples demonstrating the usefulness of the approach.

2 Interval analysis

The foundation of most computer-aided proofs dealing with continuous problems is the ability to compute with set-valued functions. This not only allows for all rounding errors to be taken into account, but – more importantly – all discretization errors too. Here, we will briefly describe the fundamentals of interval analysis. For a concise reference on this topic, see e.g. [Moo66, AH83, Neu90].

Let $\mathbb{IR}$ denote the set of closed intervals. For any element $\boldsymbol{a} \in \mathbb{IR}$, we adopt the notation $\boldsymbol{a} = [\underline{a}, \overline{a}]$, where $\underline{a}, \overline{a} \in \mathbb{R}$. If $\star$ is one of the operators $+, -, \times, \div$, we define the arithmetic on elements of $\mathbb{IR}$ by

$$\boldsymbol{a} \star \boldsymbol{b} = \{a \star b \colon a \in \boldsymbol{a}, b \in \boldsymbol{b}\},$$

except that $\boldsymbol{a} \div \boldsymbol{b}$ is undefined if $0 \in \boldsymbol{b}$. Working exclusively with closed intervals, we can describe the resulting interval in terms of the endpoints of the operands:

$$\begin{aligned}
\boldsymbol{a} + \boldsymbol{b} &= [\underline{a} + \underline{b}, \overline{a} + \overline{b}] \\
\boldsymbol{a} - \boldsymbol{b} &= [\underline{a} - \overline{b}, \overline{a} - \underline{b}] \\
\boldsymbol{a} \times \boldsymbol{b} &= [\min(\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b}), \max(\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b})] \\
\boldsymbol{a} \div \boldsymbol{b} &= \boldsymbol{a} \times [1/\overline{b}, 1/\underline{b}], \quad \text{if } 0 \notin \boldsymbol{b}.
\end{aligned} \tag{1}$$

Note that the identities (1) reduce to ordinary real arithmetic when the intervals are *thin*, i.e., when $\underline{a} = \overline{a}$ and $\underline{b} = \overline{b}$. When computing with finite precision, however, directed rounding must also be taken into account, see e.g., [Moo66, Moo79].

A key feature of interval arithmetic is that it is *inclusion monotonic*, i.e., if $\boldsymbol{a} \subseteq \boldsymbol{x}$, and $\boldsymbol{b} \subseteq \boldsymbol{y}$, then

$$\boldsymbol{a} \star \boldsymbol{b} \subseteq \boldsymbol{x} \star \boldsymbol{y}, \tag{2}$$

where we demand that $0 \notin \boldsymbol{y}$ for division.

One of the main reasons for passing to interval arithmetic is that this approach provides a simple way of enclosing the range of a function f, denoted by $R(f; D) := \{f(x) \colon x \in D\}$. Except for the most trivial cases, classical mathematics provides few tools to accurately bound the range of a function. To achieve this latter goal, we extend the real functions to *interval functions* which take and return intervals rather than real numbers. Based on (1) we extend rational functions to their interval versions by simply substituting all occurrences of the real variable x with the interval variable $\boldsymbol{x}$ (and the real arithmetic operators with their interval counterparts). This produces a rational interval function $F(\boldsymbol{x})$, called the *natural interval extension* of f. As long as no singularities are encountered, we have the inclusion

$$R(f;\boldsymbol{x}) \subseteq F(\boldsymbol{x}), \tag{3}$$

by property (2). In fact, this type of range enclosure can be achieved for any reasonable function. A higher-dimensional function $f\colon \mathbb{R}^n \to \mathbb{R}$ can be extended to an interval function $F\colon \mathbb{IR}^n \to \mathbb{IR}$ in a similar manner. The function argument is then an interval-vector $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, which we also refer to as a *box*.

There exist several open source programming packages for interval analysis [CXS],[INv], [PrB], as well as commercial products such as [SUN].

We will now illustrate the use of interval techniques, with a special emphasis on parameter estimation problems.

Example 2.1 *Consider the model* $y = f(x;p) = xe^{-px}$*, together with the (exact) data point* $(x,y) = (2,1)$*, and search region* $\boldsymbol{p} = [0,1]$*. A straightforward interval evaluation of the model function yields:*

$$f(x;\boldsymbol{p}) = f(2;[0,1]) = 2e^{-[0,1]\times 2} = 2e^{-[0,2]} = 2 \times [e^{-2},1] = [2e^{-2},2]. \tag{4}$$

This constrains (at $x = 2$*) the value of the model function (*$y = 1$*) to belong to the interval* $[2e^{-2},2] \approx [0.27,2]$*, which it does.*

Had we chosen $\boldsymbol{p} = [1,2]$ *as our search space, we would obtain an inconsistency:* $1 \notin f(2,[1,2]) = [2e^{-4},2e^{-2}] \approx [0.03,0.27]$*. This would allow us to discard the entire set* $\boldsymbol{p}$*.*

This example illustrates how a *divide and conquer* approach can be devised. Starting from a large search space $\boldsymbol{p}$, we adaptively bisect $\boldsymbol{p}$ into smaller subsets, many of which we can discard via inconsistency checks.

3 Noise and data expansion

There are multiple sources of uncertainty, e.g. model simplifications, intrinsic system sensitivities (that causes unpredictable behavior of the system), measurement errors, and conversion errors. In some situations, the only information available regarding a variable can be limited to knowing the range of its realizable values. In most such cases, the variable's probability distribution information is lacking, and the uncertainty of the variable is naturally modeled by specifying an interval enclosing its possible values.

Given a model function $y = f(x; p)$ together with a finite set of noisy data $(x_1, \tilde{y}_1), \ldots, (x_N, \tilde{y}_N)$, we will attempt to find parameters that make the model *consistent* with the data, i.e., we want to find the set

$$\mathcal{S} = \{p \in \boldsymbol{p}\colon f(x_i; p) = \tilde{y}_i \text{ for all } i = 1, \ldots, N\}. \tag{5}$$

In general, this set will be empty, indicating the presence of noise in the data. In order to improve matters, we expand each data value into an interval. This can be done in several ways; assigning a width roughly proportional to the value $\tilde{y}_i$ is a good heuristic choice in many situations:

$$\tilde{y}_i \mapsto \boldsymbol{y}_i = \tilde{y}_i(1 + \alpha[-1, +1]) + \beta[-1, +1]. \tag{6}$$

Here α is a scaling factor, and β is a threshold factor, necessary for situations when $|\tilde{y}_i|$ is very small.

The new requirement for consistency is now formulated in terms of more robust inclusion conditions:

$$\mathcal{S} = \{p \in \boldsymbol{p}\colon f(x_i; p) \in \boldsymbol{y}_i \text{ for all } i = 1, \ldots, N\}. \tag{7}$$

With no data expansion, this reduces to (5). Gradually increasing α (and/or β) will eventually produce a non-empty set of consistent parameters $\mathcal{S}$. The amount of expansion necessary to find consistent parameters can in some cases provide information regarding the level and type of noise present in the data. We will explore this feature in the examples of Section 5.

Given a partition $\mathcal{P} = \{\boldsymbol{p}_i\}_{i=1}^K$ of the search space $\boldsymbol{p} = \boldsymbol{p}_1 \cup \cdots \cup \boldsymbol{p}_K$, the consistent parameters can be enclosed as $\underline{\mathcal{S}} \subset \mathcal{S} \subset \overline{\mathcal{S}}$, where

$$\begin{aligned}\underline{\mathcal{S}} &= \{\boldsymbol{p}_j \in \mathcal{P}\colon f(x_i; \boldsymbol{p}_j) \subset \boldsymbol{y}_i \text{ for all } i = 1, \ldots, N\},\\ \overline{\mathcal{S}} &= \{\boldsymbol{p}_j \in \mathcal{P}\colon f(x_i; \boldsymbol{p}_j) \cap \boldsymbol{y}_i \neq \emptyset \text{ for all } i = 1, \ldots, N\}.\end{aligned}$$

The next example displays how interval-valued data can be contracted, using constraints from both the model function and the search space.

Example 3.1 *Repeating the calculations from Example 2.1, with $\boldsymbol{p} = [0, 1]$, but with the data $(x, \boldsymbol{y}) = (2, [1, 3])$, we can contract the data range according to*

$$\boldsymbol{y} \mapsto \boldsymbol{y} \cap f(x; \boldsymbol{p}) = [1, 3] \cap [2e^{-2}, 2] = [1, 2].$$

4 Constraint propagation for pedestrians

In this section, we will outline the main ingredients of our parameter estimation procedure. As mentioned in Section 1, our method is global. As such, it attempts to find *all* parameters consistent with the data. Of course, when the data is noisy, there are no consistent parameters, in general. This forces us to expand the data – a relaxation process that decrease the noisy data information. Once sufficiently expanded, the set of consistent parameters is non-empty and bounded. We shrink the bounding set (and the expand data) by constraint propagation techniques. To be fully effective, these techniques require that the model function be represented in a special form.

4.1 The DAG representation

We use a directed acyclic graph (DAG) representation of the model function to automate constraint propagations. This representation captures the natural way of decomposing a (possibly complicated) function into its basic building blocks. The graph nodes represent variables, constants or simple functions, while the edges represent dependencies between them.

Example 4.1 *Returning to the model function of Example 2.1, $f(x;p) = xe^{-px}$, it can be decomposed into the following code list:*

$$\begin{aligned}
n_1 &= x \\
n_2 &= p \\
n_3 &= n_1 \times n_2 \\
n_4 &= -n_3 \\
n_5 &= e^{n_4} \\
n_6 &= n_1 \times n_5.
\end{aligned}$$

This list is equivalent to the DAG illustrated in Figure 4.1.

The DAG representation is used to obtain numeric and symbolic information of the function, such as derivatives, slopes, which are used to obtain good range evaluations. Information on these matters can be found in [GW08, SN05].

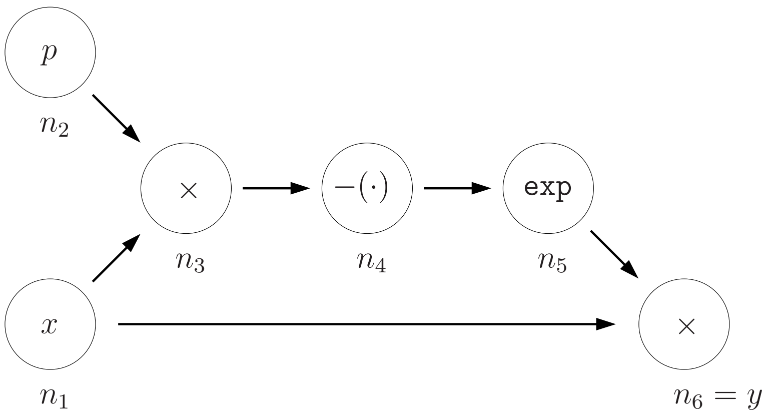

Figure 1: The DAG representation of a forward sweep of $xe^{-px} \mapsto y$.

4.2 Constraint propagation on DAGs

To each elementary mathematical operation one associates two operations, *forward* and a *backward* operations. The forward operator evaluates the range based on the range of its arguments and intersect it with the current range. The backward operator evaluates the ranges of its predecessors and intersect it with their current ranges.

Example 4.2 *Once more, we will use the model function of Example 2.1, $f(x;p) = xe^{-px}$. As we saw in Example 4.1, it can be decomposed into a code list as well as a DAG. We will now show how we can use these objects to propagate constraints from data to the parameter. Moving backwards in the code list of Example 4.1, starting from $(x, y) = (n_1, n_6)$, and ending in $p = n_2$, we obtain a new code list:*

$$\begin{aligned} n_5 &= n_6 \div n_1 \\ n_4 &= \log n_5 \\ n_3 &= -n_4 \\ n_2 &= n_3 \div n_1. \end{aligned}$$

This list is equivalent to the DAG illustrated in Figure 4.2.

Thus, viewing a function in terms of its code list or DAG allows us to compute its formal inverses, without knowing the formulae for these. All we need is the code list for f. Traversing the list backward produces the desired information. This is extremely useful for parameter estimation problems, as we illustrate in the following example.

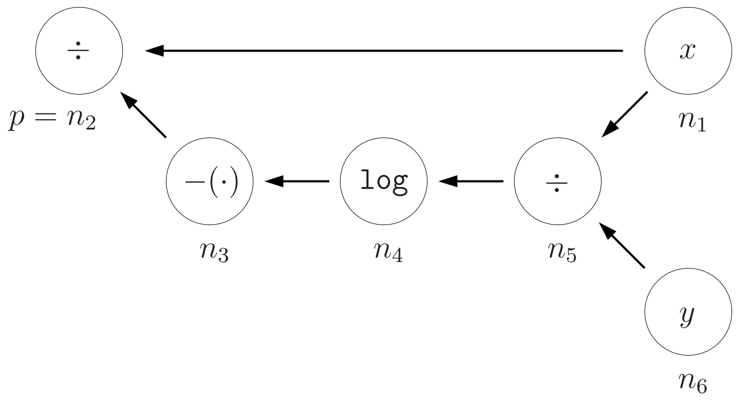

Figure 2: The DAG representation of a backward sweep of $xe^{-px} \mapsto y$.

Example 4.3 *Given the model function $y = f(x;p) = xe^{-px}$, together with the data $(x,y) = (2,1)$, we can generate, and evaluate the code list of Example 4.2:*

$$\begin{aligned} n_5 &= n_6 \div n_1 = 1 \div 2 \\ n_4 &= \log n_5 = \log \tfrac{1}{2} \\ n_3 &= -n_4 = \log 2 \\ n_2 &= n_3 \div n_1 = \tfrac{1}{2}\log 2 \approx 0.34657359. \end{aligned}$$

The conclusion is that the only parameter that corresponds to the data $(x,y) = (2,1)$ is $p = \frac{1}{2}\log 2$. Of course, for this simple example, we can find the explicit inverse: $p = \frac{1}{x}\log\frac{x}{y}$, which gives the sought result. The point is, however, that we never use this formula; we only use the code list of f.

Continuing Example 3.1, where we have interval-valued data $(x, \boldsymbol{y})$, and a parameter domain $\boldsymbol{p}$ to examine, we can combine the forward and backward sweeps to contract both $\boldsymbol{y}$ and $\boldsymbol{p}$.

Example 4.4 *Again, we work on the model function $y = f(x;p) = xe^{-px}$, but now with the data $(x,\boldsymbol{y}) = (2,[1,3])$, together with the parameter domain $\boldsymbol{p} = [0,1]$. The forward sweep, performed in Example 3.1, contracts the interval data to $\boldsymbol{y} = [1,2]$. Performing a backward sweep, as in Example 4.3, contracts the interval parameter to $\boldsymbol{p} = [0, \frac{1}{2}\log 2]$:*

$$\begin{aligned} n_5 &= n_6 \div n_1 = [1,2] \div 2 = [\tfrac{1}{2}, 1] \\ n_4 &= \log n_5 = \log[\tfrac{1}{2}, 1] = [-\log 2, 0] \\ n_3 &= -n_4 = [0, \log 2] \\ n_2 &= n_3 \div n_1 = \tfrac{1}{2}[0, \log 2] \approx [0, 0.34657359]. \end{aligned}$$

Note that, in one forward/backward sweep, we managed to exclude over 65% *of the parameter domain, at the same time reducing the data uncertainty by* 50%*.*

In general, we can iterate the procedure of taking forward/backward sweeps, until no further contraction occurs. In this example, however, we have already reached the optimal state: for $x = 2$, there is a one-to-one correspondence between parameters in $\boldsymbol{p} = [0, \frac{1}{2}\log 2]$ and function values in $\boldsymbol{y} = [1, 2]$.

In most cases, the described constraint propagation techniques do not result in a complete contraction to the optimal state. Rather, a stage is reached where no further contraction can be obtained, even though there are inconsistencies present. In order to proceed, some type of partitioning must be employed. The partitioning can be performed at any node of the DAG. Great care should be taken to make the split at the most effective place, as repeated splitting leads to exponential complexity[1]. Once a node has been selected for partitioning, its domain is split, resulting in two new DAGs, differing only in the domain of the split node. Each DAG is updated through forward/backward sweeps, possibly generating more contraction.

4.3 Data expansion strategy

Search for consistent points by expansion Here we discuss a strategy to find consistent parameters by alternating contractions and expansions. The basic idea is to expand data as long as it is detected inconsistent. The expansions increase the probability of finding consistent parameters. The input to Algorithm 1 is a box $\boldsymbol{x}^{\text{in}}$, a branch and bound procedure BAB, and an expansion rate r. Its main loop is controled by an inconsistency test based on the result of BAB. As long as BAB can reduce the current box $\boldsymbol{x}$ to the empty set, it is reapplied to an expansion of $\boldsymbol{x}$. The algorithm also adjusts the expansion to prevent over- and under-expansions. The expansion rate r is increased if the number of consecutive rejections exceed the parameter nRejection. If BAB does not find the initial box $\boldsymbol{x}^{\text{in}}$ inconsistent, the algorithm terminates. If BAB finds $\boldsymbol{x}^{\text{in}}$ inconsistent, expansion rate is enabled, and the rate is larger than a parameter rateMin, then, in order to limit over-expansion, the rate will be decreased, and the procedure is recursively called with the last inconsistent box and the decreased expansion rate as input arguments.

[1]A simple splitting strategy is to select the widest node domain for bisection. Sophisticated splitting strategies make use of monotonicity or contraction information, and can be a priori or a posteriori; for details see e.g. [CD97, CKD00].

Algorithm 1: Reject And Expand

```
Input: Box $\boldsymbol{x}^{\text{in}}$, branch and bound BAB, expansion rate $r$.
Output: Output of BAB.
1  $\boldsymbol{x} \leftarrow \boldsymbol{x}^{\text{in}},\ \boldsymbol{x}^0 \leftarrow \emptyset,\ \boldsymbol{x}^1 \leftarrow \boldsymbol{x}$
2  count $\leftarrow 0$
3  while isInconsistent$_{\text{BAB}}(\boldsymbol{x})$ do
       // increase expansion
4      if adjustEnable and (count $>$ nRejection) then
5          increase $r$
6          count $\leftarrow 0$
7      $\boldsymbol{x}^0 \leftarrow \boldsymbol{x}^1$
8      Expand($\boldsymbol{x}^1, r$)
9      count $\leftarrow$ count $+ 1$
10     $\boldsymbol{x} \leftarrow \boldsymbol{x}^1$
   // decrease expansion
11 if adjustEnable and (count $> 0$) and ($r >$ rateMin) then
12     decrease $r$
13     RejectAndExpand($\boldsymbol{x}^0$, BAB, $r$)
```

The algorithm operates with any expansion function and without any statistical information of the data. An expansion function should however mimick known properties of the data perturbations, for instance, if data perturbations are symmetric and dependent on the value of the data, then so should the expansion.

Expansion of prediction Given that a consistent parameter $p^\star \in \mathcal{S}$ for data $\boldsymbol{y}$ has been found, its images $y_i^\star = f(x_i; p^\star)$, $i = 1, \ldots, N$, provide a prediction of the noise-free data. The consistent parameter and the data prediction could be sensitive to noise. To decrease this sensitivity, and at the same time increase the probability of enclosing the noise-free data, the data prediction can be expanded to intervals $\boldsymbol{y}^\star$ that contain the noisy data. A new search is then performed, this time for all parameters $\mathcal{S}^\star$ that are consistent with $\boldsymbol{y}^\star$. The result of this search is non-empty, seeing that $p^\star$ is consistent. A similar but more global strategy to robustify the estimates involves estimation of noise properties of the data relative to the prediction.

The noise estimates are then used to define the expansion of the prediction.

4.4 Data gridding

In order to extract traditional statistics from the set-valued results, we discretize the set $\mathcal{S}^\star$ into a collection of points. Recall the the outcome of our parameter estimation is a collection of boxes $\boldsymbol{p}_1, \dots, \boldsymbol{p}_n$, whose union $\mathcal{S}^\star$ may or may not be a connected set. We form the hull of this collection by taking the smallest box $\hat{\boldsymbol{p}}$ that contains $\mathcal{S}$. Next, we introduce m equally spaced nodes along each side of $\hat{\boldsymbol{p}}$. This defines a grid of size m^d where d is the dimension of $\hat{\boldsymbol{p}}$. From this grid, we discard all nodes that are not sufficiently close to the set $\mathcal{S}^\star$. This leaves us with a set of points amenable to statistical tests.

5 Methods and examples

We demonstrate our method on two mixed-effects models. A *mixed–effects* model is a model that includes a mixture of fixed and random factors. In the current setting, we will consider model functions of the form $f(t, \vec{p})$, where t denotes time, and $\vec{p} = (p_1, p_2, p_3)$ is a three-dimensional parameter vector. In our models, we will introduce the notion of populations. A *population* parameter is a parameter that is shared by every individual in the population. An *individual* parameter is a parameter that is unique to each individual. In what follows, we let p_1 be an individual parameter, whereas p_2 and p_3 act as population parameters. Thus p_1 corresponds to a random factor (sampled from some underlying distribution), whereas p_2 and p_3 correspond to fixed factors.

5.1 Methods

Starting from a given parameter vector $\vec{p} = (p_1, p_2, p_3)$, we perturb the individual parameter according to

$$p_1^i = p_1 + \eta_i \quad \text{where} \quad \eta_i \sim N(0, \sigma^2) \qquad (i = 1, \dots, N_p). \tag{8}$$

Here $N(a, b)$ denotes the normal distribution with mean a and variance b. This produces N_p different parameter vectors $\vec{p}^1, \dots, \vec{p}^{N_p}$ (each corresponding to a different individual), where $\vec{p}^i = (p_1^i, p_2, p_3)$. For each of the N_p subjects, we generate exact data $y_{ij} = f(t_j; \vec{p^i})$ for $j = 1, \dots, N_d$. Next, the exact data is perturbed relatively, according to

$$\tilde{y}_{ij} = y_{ij}(1 + \theta_{ij}), \; i = 1, \dots, N_p, \, j = 1, \dots, N_d, \tag{9}$$

where, for instance, each θ_{ij} is either a uniform or a normally distributed random variable. This produces the data set $(t_j, \tilde{y}_{ij})$, which, together with the model function, is all information we have.

Given a search region $\boldsymbol{p}$ in parameter space, the goal is to find consistent model parameters for the entire subject population. To be more precise, we want to produce N_p estimates $\mathbb{E}(\vec{p^i})$ – one for each subject. Our estimation procedure ensures that these estimates only differ in the parameter p_1, and so from the values $\mathbb{E}(p_1^i)$, we will attempt to estimate the distribution of this parameter according to (8). Given a sufficiently large subject population (and of course a sufficiently accurate parameter estimation procedure), we should be able to accurately determine the mean value $\mu(\mathbb{E}(p_1^i))$ and the standard deviation σ for the parameter p_1.

In addition to this, we would like to be able to infer the data perturbation parameters. In order to find the typical performance of our method, we repeat the entire estimation process $N_t = 200$ times, and report the average results.

5.2 Examples

We now present two examples; the first one is well-posed in the sense that the model system is *identifiable*. This means that, given sufficient amounts of unperturbed data, we can reconstruct a *unique* parameter vector from the data. The data is sparse however. The second example, however, is not identifiable. There is an entire continuum of parameters that equally well fit any amount of unperturbed data. This situation is much harder to deal with for traditional (not set-valued) parameter estimation methods.

Example 5.1 *Consider the following function*

$$f(t; \vec{p}) = \frac{p_1}{1 + p_2 e^{p_3 t}} \tag{10}$$

used to model the growth of orange tree trunks, see [LB90, DS98].

For this specific example, we assume $N_p = 5$ individuals, with $N_d = 2$ data points per individual at $t = 100, 1600$. The true parameter values are

$\vec{p} = (191.84, 8.153, -0.0029)$. The individual p_1 parameter is perturbed absolutely with a normal random variable with zero mean and standard deviation 20. The exact data is perturbed relatively with a normal random variable with zero mean and standard deviation $0.1, 0.2$, and 0.3. We search in the parameter region $\boldsymbol{p} = ([0, 300], [0, 9], [-1, 0])$. The table shows average results from NONMEM and the set-valued method for Example 5.1, all using $N_t = 200$ trial runs. A table entry contains the pair $\mu(p_1)$ $\mu(\sigma)$ – average estimates of the distribution parameters for p_1.

	NONMEM	set-valued method
0.1	187.9205 (17.94)	198.262 (22.85)
0.2	189.8293 (21.39)	192.971 (32.42)
0.3	191.2361 (24.22)	187.088 (41.41)

Example 5.2 *Consider the following function*

$$f(x; \vec{p}) = e^{-(p_3^2+p_2^2)x} - e^{-p_1(p_3^2+p_2^2)x}, \tag{11}$$

in which the subexpression $p_3^2+p_2^2$ causes non-identifiability. For this example, we will use $N_p = 5$ individuals and $N_d = 10$ data points per individual and evenly spaced data time points within $[10^{-2}, 1]$. The true parameter values are $\vec{p} = (3, 2, 1)$. The individual parameter p_1 is perturbed absolutely by a normal with zero mean and standard deviation 1. The true data is perturbed relatively by a uniform with support on $[-\epsilon, \epsilon]$, $\epsilon \in \{0.1, 0.2, 0.3\}$. We search in the parameter region $\boldsymbol{p} = [-100, 100]^3$. The table shows average results of $N_t = 200$ experiments for Example 5.2. A table entry contains the pair $\mu(p_1)$ $\mu(\sigma)$ – average estimates of the distribution parameters for p_1.

	set-valued method
0.1	2.9831 (0.988)
0.2	2.8531 (0.915)
0.3	3.7310 (1.031)

6 Discussion

We have described a method to solve parameter estimation problems for noisy data. It is a deterministic global search method based on set–valued computational methods. We gave two examples of population parameter estimation for five individuals. Any number of individuals can be estimated by grouping individuals into small groups whose total amount of data excedes the number of unknown parameters, and then performing averaging over group estimates. The method can be used for non-identifiable systems. When using a set-valued method like this, a parameter search domain should be large enough to includes all possible consistent parameter values. Large search domains generally require more work than small search domains. However, often constraint propagation efficiently rejects large inconsistent regions and estimation time can be quite insensitive to the size of the search domain. The parameter estimates produced will depend on the amount of noise in the data. The precision of the estimates can be read off as a measure of how much the inner and outer parameter solution sets agree. Also, goodness of fit is a measure of how much the prediction agrees with the original (noisy) data and any correlation measure is applicable to describe the parameter estimates. As with any estimator, there are several likely reasons for unsuccessful estimations, such as too little data or too much noise in data, or, having an inappropriate model. A set-valued estimator can be used in model selection. Here, models can be sorted according to how well they perform in, for instance, producing identifiable parameter estimates or making good predictions. The models that perform well are kept and the rest are rejected.

References

[AH83] Göltz Alefeld and Jürgen Herzberger. *Introduction to Interval Computations*. 1983.

[CD97] T. Csendes and D.Ratz. Subdivision direction selection in interval methods for global optimization. *SIAM J. Numer. Anal.*, 34(3):922–938, 1997.

[CKD00] T. Csendes, R. Klatte, and D.Ratz. A posteriori direction selection rules for interval optimization methods. *Central European Journal of Operations Research*, 8(3):225–236, 2000.

[CXS] CSXC, C++ eXtension for Scientific Computation, version 2.2.3. Available from `http://www.math.uni-wuppertal.de/org/WRST/xsc/cxsc.htmlh`.

[DS98] Norman P. Draper and Harry Smith. *Applied Regression Analysis*. John Wiley and Sons, New York, 3rd edition, 1998.

[GW08] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, PA, 2nd edition, 2008.

[INv] INTLAB, INTerval LABoratory, version 5.5.
Available from `http://www.ti3.tu-harburg.de/\~{}rump/intlab/`.

[LB90] Mary J. Lindstrom and Douglas M. Bates. Nonlinear mixed effects models for repeated measures data. *Biometrics*, 46(3):673–687, 1990.

[Moo66] Ramon E. Moore. *Interval analysis*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1966.

[Moo79] Ramon E. Moore. *Methods and Applications of Interval Analysis*. SIAM Studies in Applied Mathematics, Philadelphia, 1979.

[Neu90] Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, 1990.

[PrB] PROFIL/BIAS, Programmer's Runtime Optimized Fast Interval Library / Basic Interval Arithmetic Subroutines, version 2.04.
Available from `http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html`.

[SN05] Hermann Schichl and Arnold Neumaier. Interval analysis on directed acyclic graphs for global optimization. *J. of Global Optimization*, 33(4):541–562, 2005.

[SUN] Forte Developer 7: C++ Interval Arithmetic Programming Reference.
Available from `http://docs.sun.com/app/docs/doc/816-2465`.