CLUSTER NEWTON METHOD FOR SAMPLING MULTIPLE SOLUTIONS OF UNDERDETERMINED INVERSE PROBLEMS: APPLICATION TO A PARAMETER IDENTIFICATION PROBLEM IN PHARMACOKINETICS*

YASUNORI AOKI†, KEN HAYAMI‡, HANS DE STERCK §, AND AKIHIKO KONAGAYA ¶

Abstract. A new algorithm is proposed for simultaneously finding multiple solutions of an underdetermined inverse problem. The algorithm was developed for an ODE parameter identification problem in pharmacokinetics for which multiple solutions are of interest. The algorithm proceeds by computing a cluster of solutions simultaneously, and is more efficient than algorithms that compute multiple solutions one-by-one because it fits the Jacobian in a collective way using a least squares approach. It is demonstrated numerically that the algorithm finds accurate solutions that are suitably distributed, guided by *a priori* information on which part of the solution set is of interest, and that it does so much more efficiently than a baseline Levenberg-Marquardt method that computes solutions one-by-one. It is also demonstrated that the algorithm benefits from improved robustness due to an inherent smoothing provided by the least-squares fitting.

Key words. Inverse Problems, Method of Least Squares, Pharmacokinetics, Underdetermined Problems

AMS subject classifications. 65L09, 92C45

1. Introduction.

Since the information we can obtain clinically from a live patient going through treatment is often much less extensive than the complexity of the internal activity in a patient's body, underdetermined inverse problems naturally appear in the field of mathematical medicine. Our interest in underdetermined inverse problems of this kind was initiated by the parameter identification problem of a pharmacokinetics model for the anti-cancer drug CPT-11 (also known as Irinotecan) [2]. This pharmacokinetics model is an ODE-based mathematical model for the transportation, metabolization and excretion of the drug in a human body. In this problem, a large set of parameters needs to be estimated from a very small number of measurements that correspond to integrated (accumulated) quantities (area-under-the-curve) at a single final time T. Konagaya has proposed a framework called "virtual patient population convergence" [14] (see [15] for the English translation), whose essential idea is to estimate the parameters of a whole body pharmacokinetics model from the clinically observed patient data. The essential difference of this framework with other approaches is that instead of finding a single set of parameters that is suitable for the pharmacokinetics model to reproduce the clinically observed data, its aim is to find multiple sets of such parameters, because the ranges and extremal values of the parameters that can be obtained from multiple solutions are of significant value in the context of the pharmacokinetics problem.

*This work was supported by the Grant-in-Aid for Scientific Research (C) of the Ministry of Education, Sports, Science and Technology, Japan, and by the Natural Science and Engineering Research Council of Canada. This paper is based on technical report [1].

†University of Waterloo, 200 University Ave. West, Waterloo, Ontario, Canada, N2L 3G1, now at Uppsala University, Box 591 75124 Uppsala, Sweden, (yaoki@uwaterloo.ca).

‡National Institute of Informatics 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan, 101-8430 (hayami@nii.ac.jp).

§University of Waterloo, 200 University Ave. West, Waterloo, Ontario, Canada, N2L 3G1 (hdesterck@uwaterloo.ca).

¶Tokyo Institute of Technology 4259 Nagatsuta-cho, Midori-ku, Yokohama, Japan, 226-8503 (kona@dis.titech.ac.jp).

For highly underdetermined inverse problems, which may have a large set of exact solutions for a given right-hand-side, it is customary to add extra constraints to make the solution unique (e.g., one may seek the solution closest to some initial point, or one may add regularization terms). If only one of many solutions is considered, it is often hard to know to which degree the characteristics of that specific solution are representative of all solutions, and in how much they are a consequence of the particular choice of the extra constraints. Hence, we wish to sample many solutions from the solution set of the underdetermined inverse problem for the application at hand. However, for a problem as complicated as a pharmacokinetics model aiming to model whole body drug kinetics, even to find one set of model parameters that fits a clinical observation can be time consuming. Thus, trying to find multiple sets of model parameters one set by one set can take computational time that is unrealistic for practical use, if a traditional method like the single-shooting Levenberg-Marquardt (LM) method is used.

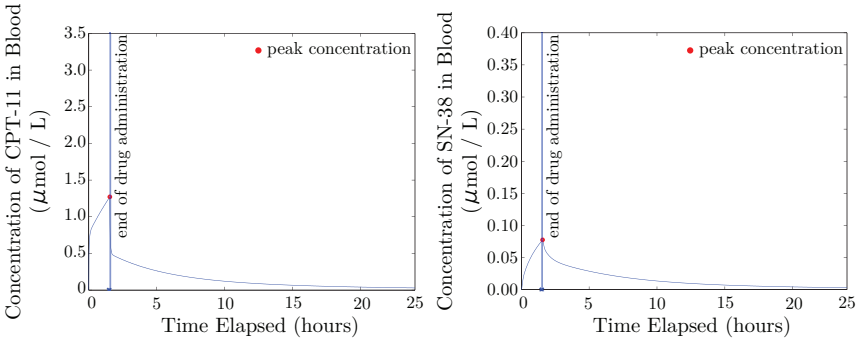
With this motivation, we have constructed an algorithm to simultaneously find multiple solutions of an underdetermined inverse problem, in a new way that is significantly more efficient than solving many separate inverse problems with different initial iterates, and that benefits from improved robustness. Our iterative scheme starts with a set (cluster) of initial points and computes the forward problem at each point. It then fits a hyperplane to the solution values in the sense of least squares and obtains a linear approximation of the function that corresponds to the forward problem. This linear approximation aims to approximate the function in the broad domain covered by the cluster of initial points. Using this single linear approximation for all cluster points, we collectively move the cluster of points closer to the solution of the inverse problem in an update step that is similar to a Newton step. By repeating this iteratively, the cluster of points becomes stationary and the points are close to being solutions of the inverse problem. In the second stage of our algorithm we use Broyden's method to improve the accuracy of the approximate solutions in the cluster by moving each cluster point individually using different approximated Jacobians, until the desired accuracy is achieved. Throughout this paper, we shall refer to this method we have constructed as the Cluster Newton method (CN method).

Through numerical experiments, we have found that the Cluster Newton method requires far less function evaluations than using the LM method to compute multiple solution points separately starting from different initial iterates. The Cluster Newton method is similar to Newton's method in the sense that it iteratively improves the approximation by approximating the forward problem by a linear function and inverting the linear approximation. Aside from moving a cluster of points instead of a single point, the Cluster Newton method differs from the traditional Newton's method in the sense that instead of approximating the Jacobian locally, we estimate it more globally in the domain covered by the cluster of points. The global approximation of the Jacobian acts as a regularization and we have observed that the Cluster Newton algorithm is robust against noise in the function evaluations (e.g., caused by the inaccuracy of solving the forward problem), compared to a method like LM. As can be seen in recent works, for example [11, 19, 20], optimization problems and parameter identification problems with rough functions or noisy data can be challenging and are of interest to the scientific computing community. Such roughness can appear in the coefficient identification problem of a system of ODEs when the system is solved numerically, hence robustness against this type of roughness is important when identifying the parameters of ODE-based pharmacokinetics models.

In this paper, we demonstrate the effectiveness of the CN method on a simple model problem in an abstract setting, and on a real highly underdetermined parameter identification problem from pharmacokinetics for which multiple solutions are of interest [2]. For both problems, we compare the CN method with the LM method, which is a universal nonlinear solver that can be applied to our problem in a general setting. In the context of our parameter identification problem, we combine forward ODE solves with LM in a so-called single-shooting LM approach that obtains solutions one-by-one, and compare this with the collective CN results. While this provides a baseline comparison, it has to be noted that parameter identification methods exist that are more efficient and more robust than single-shooting LM. These more advanced methods may combine approaches like multiple shooting and accurate Jacobian computation via additional ODEs, and may take advantage of the structure of particular ODE systems, resulting in faster and more robust solution methods than single-shooting LM [4, 7]. While it may be possible to adapt some of these advanced methods to obtain solution methods for the problem of finding multiple solutions of underdetermined inverse problems that are more efficient than computing the solutions one-by-one, it appears that this has not been done yet. In this paper, we propose a different and novel approach for this problem based on collectively computing many solutions by fitting a linear approximation using a least-squares approach. As we will show, this approach is much more efficient than computing multiple solutions one-by-one (for example, using LM), and has improved robustness due to an inherent smoothing provided by the least-squares fitting. Our numerical results demonstrate that the collective CN method is much more efficient than computing solutions one-by-one using single-shooting LM, and we will argue that the same conclusion is expected to hold for one-by-one methods that may be faster than single-shooting LM, for the simple reason that the total number of forward ODEs solved in our collective approach is much smaller than the number of ODEs that need to be solved in any one-by-one method. On the other hand, it may be possible to extend our new collective method using techniques from advanced parameter identification methods like multiple shooting and statistical analysis, in order to make our new method more robust and more widely applicable. This is a topic of further research.

1.1. Motivation for seeking multiple solutions of the underdetermined pharmacokinetics problem. The motivation for seeking multiple solutions instead of a single solution for the underdetermined inverse problem of parameter identification for Arikuma et al.'s pharmacokinetics model for the anti-cancer drug CPT-11 [2] can be further illustrated as follows. Figure 1.1 shows the concentrations of CPT-11 and SN-38 (a metabolite of CPT-11) in blood simulated by the pharmacokinetics model using a set of model parameters found by the single-shooting LM method based on clinically measured data. The LM method iteratively finds a solution of the underdetermined inverse problem near the initial iterate. We have chosen the initial iterate as the "typical" values of the model parameters listed in Arikuma et al. [2]. These "typical" values are best available estimates for the average values of the parameters over human populations. From Figure 1.1, we observe that the peak concentration in both CPT-11 and SN-38 occurs at time $t = 1.5$ (we denote this time as T_{max}). Also, observe that the peak concentration is around 1.3 $\mu mol/L$ for CPT-11 and 0.08 $\mu mol/L$ for SN-38. (We denote peak concentrations by C_{max})

In order to investigate further whether these obtained values are specific to the choice of the initial iterate or similar for most of the solutions of this underdetermined inverse problem, we have computed multiple solutions (multiple sets of model



(a) Concentration of the anti-cancer drug CPT-11.

(b) Concentration of the metabolite SN-38.

FIG. 1.1. *Drug and Metabolite concentration simulation based on a single set of model parameters found by the LM Method.*

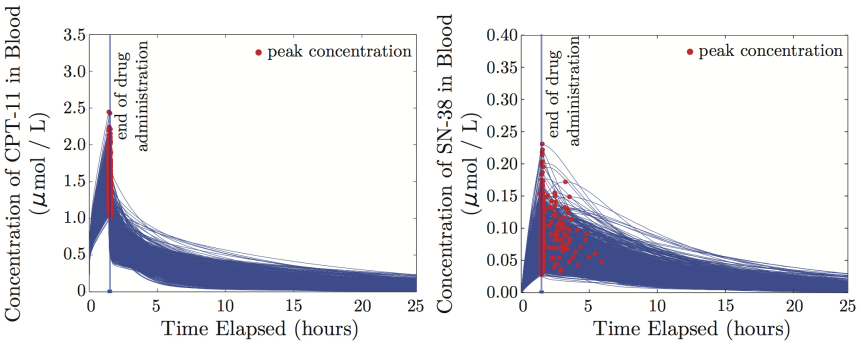
parameters) using the LM method with different initial iterates close to the "typical" values listed in [2]. Figure 1.2 shows the concentrations of CPT-11 and SN-38 in blood simulated by the pharmacokinetics model using 1,000 sets of model parameters found by the LM method with 1,000 different initial iterates. We observe from Figure 1.2 that only the observation that $T_{max} = 1.5$ for CPT-11 seems independent of the choice of the initial iterate and may be a common feature among the solutions in the solution set of this underdetermined inverse problem. Other values (e.g., C_{max} for both CPT-11 and SN-38 and T_{max} for SN-38) are heavily dependent on the choice of the initial iterate. That is to say, these values cannot be determined precisely due to the underdetermined nature of the inverse problem. Although these values cannot be determined precisely, information on the range of C_{max} and T_{max} as obtained from the multiple solutions shown in Figure 1.2 is still of significant value in the context of the application problem.

The values of C_{max} and T_{max} can be measured, even though this is only realistic in a clinical experiment setting (e.g., 16 blood samples are required from a patient in a day) and not necessarily reliable. For example, Slatter et al. have obtained for 8 patients that C_{max} of CPT-11 is on average 2.26 $\mu mol/L$ (with standard deviation of 0.21), C_{max} of SN-38 is on average 0.04 $\mu mol/L$ (with standard deviation of 0.017), T_{max} of CPT-11 is 1.5 hours (with zero standard deviation) and T_{max} of SN-38 is on average 2.3 hours (with standard deviation of 1.0). All of the measured values except T_{max} of CPT-11 are different from what can be predicted from the single solution of Figure 1.1. However, Table 1.1 shows that these measured values for a small set of patients are almost within the range of the values of C_{max} and T_{max} obtained by solving for multiple solutions of the underdetermined inverse problem as shown in Figure 1.2. While the numbers in Table 1.1 indicate that the pharmacokinetics model is not perfect yet, this example does show that obtaining multiple solutions of the underdetermined inverse problem is useful for determining the general characteristics of the solutions in the solution set of the underdetermined inverse problem. Ideally, ODE models and available data should be such that parameter identification problems have unique and well-defined solutions, but in biology and medicine this may not always be the case in practice, and there is a need for reliable numerical methods that can deal with underdetermined parameter identification problems when they arise.

The MATLAB implementation of the LM method we used took 3.3 minutes to

TABLE 1.1
Summary of C_{max} and T_{max} predicted from Figures 1.1 and 1.2, and clinically measured values.

	Predicted value Figure 1.1	Range from Figure 1.2	Measured value in [17] (avg ± sd)
C_{max} of CPT-11 ($\mu mol/L$)	1.3	1.0∼2.5	2.26 ±0.21
C_{max} of SN-38 ($\mu mol/L$)	0.08	0.02∼0.23	0.04 ± 0.017
T_{max} of CPT-11 (hours)	1.5	1.5	1.5 ± 0
T_{max} of SN-38 (hours)	1.5	1.5∼6	2.3 ± 1.0



(a) Concentration of the anti-cancer drug CPT-11.

(b) Concentration of the metabolite SN-38.

FIG. 1.2. *1,000 model parameter sets found by multiple application of the LM Method.*

compute the single set of model parameters that was used to simulate the concentration of CPT-11 in blood plotted in Figure 1.1, using one core of an Intel Xeon X7350 3GHz processor. It took about 7 hours with a server machine with two quad-core Intel Xeon X7350 3GHz processors to find the 1,000 model parameters used to produce Figure 1.2. Our goal in this paper is to develop an algorithm that can find such sets of parameters with significantly less computational cost.

1.2. Problem Statement. In this paper, we consider the following underdetermined inverse problem:
Find parameter vector $\boldsymbol{x}$ such that

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}^* , \tag{1.1}$$

where $\boldsymbol{y}^*$ is a given constant vector in $\mathbb{R}^n$, and $\boldsymbol{f}$ is a C^1 vector function from $\mathcal{X} \subset \mathbb{R}^m$ to $\mathbb{R}^n$ with $m > n$. We seek m parameters from n equations with $m > n$, so the solution is in general not unique. (See Appendix A for an explanation of the matrix and vector notation used in this paper.) We assume this inverse problem has the following properties:

- The evaluation of $\boldsymbol{f}$ (solving the forward problem) is computationally expensive. Thus, we would like to minimize the number of function evaluations.
- The Jacobian of $\boldsymbol{f}$ is not explicitly known.

In the pharmacokinetics application we target (see Section 4), the right-hand-side quantity $\boldsymbol{y}^*$ contains measurements that are subject to measurement errors. In this particular context it is reasonable to assume that the standard deviations of the measured values are proportional to the quantities measured [2]. In general, other

error models can also be considered. Note that, for the highly undetermined inverse problems we target, Equation (1.1) typically has a large set of exact solutions for any fixed value of $\boldsymbol{y}^*$. It is our goal to compute many solutions to problem (1.1) simultaneously, with an accuracy that is dictated by the measurement errors on the right-hand side. We thus seek solutions in the subset $\mathcal{X}_\epsilon^*$ of $\mathcal{X}$, which is the set containing all the values of $\mathcal{X}$ which approximately satisfy (1.1) with maximum norm relative residual less than ϵ, i.e.,

$$\mathcal{X}_\epsilon^* := \{\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^m \, : \, \max_{i=1,\ldots,n} |\left(f_i(\boldsymbol{x}) - y_i^*\right)/y_i^*| < \epsilon\}\,. \tag{1.2}$$

Here, we have used a relative error in (1.2) because for the application we target the standard deviations of measured values can be assumed proportional to the quantities measured. More generally, a scaling by standard deviation could be used to define $\mathcal{X}_\epsilon^*$ [4]. We note that in most cases $\mathcal{X}_\epsilon^*$ is an infinite set and often it is an unbounded set. In practice, we are interested in a part of this set $\mathcal{X}_\epsilon^*$, namely the part that is relevant in the context of the problem and that corresponds to a range of reasonable physiological parameters. For real physiological parameter identification applications, experiments described in the literature often provide a typical value for each of the parameters in the parameter vector $\boldsymbol{x}$, and a range in which each parameter can be expected to vary. The goal is to find multiple sets of parameter values in $\mathcal{X}_\epsilon^*$ guided by these physiologically relevant values in the following sense. We use these typical values and ranges to define a box in $\mathcal{X}$ in which the initial cluster of points for our method is chosen in a uniformly random manner. This cluster is used to initialize the algorithm, and the location and size of this initial box will also influence the location and size of the final solution cluster obtained by the algorithm. As explained below, the algorithm moves the initial cluster points towards the solution set by solving least squares problems in a way that minimizes the distance between successive iterates for each cluster point. The algorithm thus targets finding cluster points in the solution set that remain close to the centre of the initial box, and have a range that is similar to the range of the initial box. In this way, the selection of cluster points in the solution set is guided by the physiologically relevant parameter values that define the initial box. To summarize, we assume that we know the following regarding the physiologically relevant values of $\boldsymbol{x} \in \mathcal{X}$:

- a typical value of $\boldsymbol{x}$ is known (we denote this value as $\hat{\boldsymbol{x}}$, and it is used as the centre of the initial cluster of points for our algorithm)
- the typical relative ranges of the parameter values in $\boldsymbol{x}$ are known and we denote the typical relative range of the ith parameter in the parameter vector as v_i (see Equation 2.1 below; the ranges v_i are used to define the size of the initial cluster centred about $\hat{\boldsymbol{x}}$).

The remainder of this paper is structured as follows. In Section 2, we give a detailed description of the Cluster Newton method, and explain its relation to existing methods. In Section 3 we illustrate the method using a simple model problem. Section 4 explains how the method can be applied to a large pharmacokinetics model, and compares performance with the LM method and other algorithms. Section 5 briefly describes two extensions of the Cluster Newton method. Conclusions are formulated in Section 6 followed by an Appendix on notation.

2. Algorithm: Cluster Newton Method. The Cluster Newton method for finding a cluster of parameter vectors $\boldsymbol{x}$ in the desired solution set $\mathcal{X}_\epsilon^*$ proceeds in two stages.

Starting from an initial cluster of points chosen uniformly randomly in the initial box centred about the typical value $\hat{\boldsymbol{x}}$ for the parameter vector, the first stage of the algorithm iteratively moves the initial cluster of points towards the desired solution set $\mathcal{X}_\epsilon^*$. In each iteration, a linear approximation of $\boldsymbol{f}(\boldsymbol{x})$ is constructed in the neighbourhood of the current cluster using the solutions of the forward problem at all the points in the cluster, and then this linear approximation is used to move the cluster closer to the desired solution set $\mathcal{X}_\epsilon^*$ in an update step that is similar to a Newton step. The linear approximation is created using least squares fitting of a hyperplane to the solutions of the forward problem. A visual illustration of this iterative process is provided for a simple example problem in Section 3 (see Figure 3.3), and detailed pseudocode for all the steps of the algorithm is provided below.

Since Stage 1 of our algorithm moves all the points in the cluster collectively using one linear approximation, the improvement in the accuracy of the approximation eventually stalls after a number of iterations have been conducted. If the accuracy achieved in the first stage of the algorithm is not sufficient, we proceed to Stage 2 of the algorithm. In the second stage of the algorithm, we further move each point towards the desired solution set $\mathcal{X}_\epsilon^*$ but now individually, using Broyden's method. For each point, we take the collective linear approximation of $\boldsymbol{f}(\boldsymbol{x})$ from Stage 1 to obtain an initial approximation for the Jacobian in Broyden's method. This avoids the large number of function evaluations that would be required for approximating the initial Jacobian for Broyden's method for each solution, for example by a finite difference scheme.

2.1. Algorithm pseudocode (see also Figure 3.3 for graphical illustration of the algorithm). Recall that we seek multiple solutions of the underdetermined inverse problem $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}^*$ with m parameters and n equations ($m > n$).

Stage 1

1: Set up the initial cluster points and the target values.

1-1: Uniformly randomly generate an initial cluster of l points $\{\boldsymbol{x}_{\cdot j}^{(0)}\}_{j=1}^{l}$ in $\mathbb{R}^m$ in a box defined by the following inequalities:

$$\left| \frac{x_{ij}^{(0)} - \hat{x}_i}{\hat{x}_i} \right| < v_i \qquad \text{for } i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, l\,, \tag{2.1}$$

where $\hat{x}_i$ is the typical value of the ith parameter and v_i is the typical relative range of the ith parameter. We require the number of points in the cluster to satisfy $l \geq m + 1$ so that we can construct a linear approximation to $\boldsymbol{f}(\boldsymbol{x})$ using the values of $\boldsymbol{f}$ at all the points in the cluster. We typically choose l to be much larger than $m + 1$ in order to make the algorithm more robust against small scale roughness in the function values and because we are interested in obtaining many more solutions than $m + 1$. The vectors $\boldsymbol{x}_{\cdot j}^{(0)}$ are stored in the columns of matrix $X^{(0)} = [\boldsymbol{x}_{\cdot 1}^{(0)}, \boldsymbol{x}_{\cdot 2}^{(0)}, \ldots, \boldsymbol{x}_{\cdot l}^{(0)}] \in \mathbb{R}^{m \times l}$.

1-2: Generate randomly perturbed target vectors $\{\boldsymbol{y}_{\cdot j}^*\}_{j=1}^{l}$ near $\boldsymbol{y}^*$. We choose each vector $\boldsymbol{y}_{\cdot j}^*$ so that

$$\max_{i=1,2,\ldots,n} \left| \frac{y_{ij}^* - y_i^*}{y_i^*} \right| < \eta\,, \tag{2.2}$$

with $\eta = 0.1$. The random perturbation is necessary in line 2-3 to keep line 2-2 of the algorithm well-defined, as explained in Section 2.2.3.

2: For $k = 0, 1, 2, \ldots, K_1$

2-1: Solve the forward problem for each point $\boldsymbol{x}_{\cdot j}^{(k)}$, i.e., compute

$$\boldsymbol{y}_{\cdot j}^{(k)} = \boldsymbol{f}(\boldsymbol{x}_{\cdot j}^{(k)}) \qquad \text{for } j = 1, 2, \ldots, l\,, \tag{2.3}$$

with $Y^{(k)} = [\boldsymbol{y}_{\cdot 1}^{(k)}, \boldsymbol{y}_{\cdot 2}^{(k)}, \ldots, \boldsymbol{y}_{\cdot l}^{(k)}]$.

2-2: Construct a linear approximation of $\boldsymbol{f}$, i.e.,

$$\boldsymbol{f}(\boldsymbol{x}) \approx A^{(k)}\boldsymbol{x} + \boldsymbol{y}_o^{(k)}\,, \tag{2.4}$$

by fitting a hyperplane to $\{(\boldsymbol{x}_{\cdot j}^{(k)}, \boldsymbol{y}_{\cdot j}^{(k)})\}_{j=1}^{l}$. Recalling that we have chosen the number of the points in the cluster to be l, where $l \geq m+1$, the slope matrix $A^{(k)} \in \mathbb{R}^{n\times m}$ and the shift constant $\boldsymbol{y}_o^{(k)} \in \mathbb{R}^n$ can be found as the least squares solution of an overdetermined system of linear equations:

$$\min_{A^{(k)}\in\mathbb{R}^{n\times m}\,,\,\boldsymbol{y}_o^{(k)}\in\mathbb{R}^n} ||Y^{(k)} - (A^{(k)}X^{(k)} + Y_o^{(k)})||_F\,, \tag{2.5}$$

where $Y_o^{(k)}$ is a $n \times l$ matrix whose columns are all $\boldsymbol{y}_o^{(k)}$.

2-3: Find an update vector $\boldsymbol{s}_{\cdot j}$ for each $\boldsymbol{x}_{\cdot j}^{(k)}$ using the linear approximation, i.e., find $\boldsymbol{s}_{\cdot j}$ s.t.

$$\boldsymbol{y}_{\cdot j}^{*} = A^{(k)}(\boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)}) + \boldsymbol{y}_o{}^{(k)} \qquad \text{for } j = 1, 2, \ldots, l\,, \tag{2.6}$$

and $\boldsymbol{x}_{\cdot j}^{(k+1)} = \boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)}$. As can be seen from the fact that matrix $A^{(k)} \in \mathbb{R}^{n\times m}$ is a rectangular matrix with more columns than rows, this is an underdetermined system of linear equations. Hence, we cannot uniquely determine $\boldsymbol{s}_{\cdot j}^{(k)}$ that satisfies Equation (2.6). Instead, we choose the vector $\boldsymbol{s}_{\cdot j}^{(k)}$ with the shortest scaled length, among all the solutions of Equation (2.6), as follows:

$$\min_{\boldsymbol{s}_{\cdot j}^{(k)}\in\mathbb{R}^m} ||(diag(\hat{\boldsymbol{x}}))^{-1}\boldsymbol{s}_{\cdot j}^{(k)}||_2 \tag{2.7}$$

$$\text{s.t.} \qquad \boldsymbol{y}_{\cdot j}^{*} = A^{(k)}(\boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)}) + \boldsymbol{y}_o^{(k)}\,, \tag{2.8}$$

for $j = 1, 2, \ldots, l$, where $\hat{\boldsymbol{x}} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m]^{\mathrm{T}}$.

We note that we have scaled Expression (2.7) with the diagonal matrix $diag(\hat{\boldsymbol{x}})^{-1}$ since the order of magnitude of the parameter values in vector $\hat{\boldsymbol{x}}$ can vary significantly, and finding the vector with shortest scaled length leads to a more robust method.

2-4: Find new points approximating the solution set $\mathcal{X}^*$ by updating $X^{(k)}$. If necessary, we first shrink the length of the vector $\boldsymbol{s}_{\cdot j}^{(k)}$ until the point $\boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)}$ is in the domain of the function $\boldsymbol{f}$ by the following simple procedure:

For $j = 1, 2, \ldots, l$

While $(\boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)}) \notin \mathcal{X}$

$$\boldsymbol{s}_{\cdot j}^{(k)} = \frac{1}{2}\boldsymbol{s}_{\cdot j}^{(k)} \tag{2.9}$$

End while
End for
Then

$$\boldsymbol{x}_{\cdot j}^{(k+1)} = \boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)} \qquad \text{for } j = 1, 2, \ldots, l\,, \tag{2.10}$$

with $X^{(k+1)} = [\boldsymbol{x}_{\cdot 1}^{(k+1)}, \boldsymbol{x}_{\cdot 2}^{(k+1)}, \ldots, \boldsymbol{x}_{\cdot l}^{(k+1)}]$.
End for.

Stage 2: Broyden's method

3: Set up the initial Jacobian approximation for each point $\boldsymbol{x}_{\cdot j}^{(k)}$:

$$J_{(j)}^{(K_1+1)} = A^{(K_1)} \qquad \text{for } j = 1, 2, \ldots, l\,. \tag{2.11}$$

4: For $k = K_1 + 1, \ldots, K_2$

4-1: Solve the forward problem for each point $\boldsymbol{x}_{\cdot j}^{(k)}$, i.e.,

$$\boldsymbol{y}_{\cdot j}^{(k)} = \boldsymbol{f}(\boldsymbol{x}_{\cdot j}^{(k)}) \qquad \text{for } j = 1, 2, \ldots, l\,. \tag{2.12}$$

4-2: If $k \neq K_1 + 1$ then update the Jacobian for each point using Broyden's method (see [12] or [7]) as follows:

$$J_{(j)}^{(k)} = J_{(j)}^{(k-1)} + (\boldsymbol{y}_{\cdot j}^{(k)} - \boldsymbol{y}^*)\frac{(\boldsymbol{s}_{\cdot j}^{(k-1)})^{\mathrm{T}}}{||\boldsymbol{s}_{\cdot j}^{(k-1)}||^2} \qquad \text{for } j = 1, 2, \ldots, l. \tag{2.13}$$

4-3: Find the search direction vector $\boldsymbol{s}_{\cdot j}^{(k)}$ for each $\boldsymbol{x}_{\cdot j}^{(k)}$ using the approximate Jacobian, i.e., $\boldsymbol{s}_{\cdot j}^{(k)}$ is given by the minimum norm solution of an underdetermined linear system:

$$\min_{\boldsymbol{s}_{\cdot j}^{(k)} \in \mathbb{R}^m} ||(diag(\hat{\boldsymbol{x}}))^{-1}\boldsymbol{s}_{\cdot j}^{(k)}||_2 \tag{2.14}$$

$$\text{s.t.} \qquad \boldsymbol{y}^* - \boldsymbol{y}_{\cdot j}^{(k)} = J_{(j)}^{(k)}\boldsymbol{s}_{\cdot j}^{(k)} \tag{2.15}$$

for $j = 1, 2, \ldots, l$.

4-4: Find new points approximating the solution set $\mathcal{X}^*$ by updating $X^{(k)}$, i.e.,

For $j = 1, 2, \ldots, l$
While $(\boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)}) \notin \mathcal{X}$

$$\boldsymbol{s}_{\cdot j}^{(k)} = \frac{1}{2}\boldsymbol{s}_{\cdot j}^{(k)} \tag{2.16}$$

End while
End for
Then

$$\boldsymbol{x}_{\cdot j}^{(k+1)} = \boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)} \qquad \text{for } j = 1, 2, \ldots, l\,. \tag{2.17}$$

End for.

2.2. Further details on some key steps in the algorithm.

2.2.1. Finding the linear approximation (line 2-2). When constructing the linear approximation in Stage 1, we solve for the least squares solution of the overdetermined system of linear equations (2.5). To show how this computation can be done, we first rewrite (2.5) in standard matrix-vector multiplication form and then solve a set of least squares problems. We first rewrite the matrix in expression (2.5), by considering its n rows:

$$\begin{aligned} & Y^{(k)} - (A^{(k)}X^{(k)} + Y_o^{(k)}) \\ &= \begin{bmatrix} \boldsymbol{y}_{1\cdot}^{(k)} \\ \boldsymbol{y}_{2\cdot}^{(k)} \\ \vdots \\ \boldsymbol{y}_{n\cdot}^{(k)} \end{bmatrix} - \left(\begin{bmatrix} \boldsymbol{a}_{1\cdot}^{(k)} \\ \boldsymbol{a}_{2\cdot}^{(k)} \\ \vdots \\ \boldsymbol{a}_{n\cdot}^{(k)} \end{bmatrix} X^{(k)} + \begin{bmatrix} y_{o1}^{(k)} \dots y_{o1}^{(k)} \\ y_{o2}^{(k)} \dots y_{o2}^{(k)} \\ \vdots \\ y_{on}^{(k)} \dots y_{on}^{(k)} \end{bmatrix} \right), \end{aligned} \tag{2.18}$$

where $\boldsymbol{y}_{i\cdot}^{(k)} \in \mathbb{R}^{1\times l}$ and $\boldsymbol{a}_{i\cdot}^{(k)} \in \mathbb{R}^{1\times m}$. By taking the transpose of both sides we obtain the following expression:

$$\begin{aligned} & (Y^{(k)} - (A^{(k)}X^{(k)} + Y_o^{(k)}))^{\mathrm{T}} \\ &= \left[\boldsymbol{y}_{1\cdot}^{(k)\mathrm{T}} \dots \boldsymbol{y}_{n\cdot}^{(k)\mathrm{T}} \right] - \left(X^{(k)\mathrm{T}} \left[\boldsymbol{a}_{1\cdot}^{(k)\mathrm{T}} \dots \boldsymbol{a}_{n\cdot}^{(k)\mathrm{T}} \right] + \begin{bmatrix} y_{o1}^{(k)} \dots y_{on}^{(k)} \\ y_{o1}^{(k)} \dots y_{on}^{(k)} \\ \vdots \\ y_{o1}^{(k)} \dots y_{on}^{(k)} \end{bmatrix} \right), \end{aligned} \tag{2.19}$$

where $\boldsymbol{y}_{i\cdot}^{(k)\mathrm{T}} \in \mathbb{R}^{l\times 1}$, $X^{(k)\mathrm{T}} \in \mathbb{R}^{l\times m}$ and $\boldsymbol{a}_{i\cdot}^{(k)\mathrm{T}} \in \mathbb{R}^{m\times 1}$. We now observe that expression (2.5) is equivalent to n independent least squares problems:

$$\min_{\boldsymbol{a}_{i\cdot}^{(k)} \in \mathbb{R}^m,\, y_{oi}^{(k)} \in \mathbb{R}} \left|\left| \boldsymbol{y}_{i\cdot}^{(k)\mathrm{T}} - \left(X^{(k)\mathrm{T}} \boldsymbol{a}_{i\cdot}^{(k)\mathrm{T}} + \begin{bmatrix} y_{oi}^{(k)} \\ y_{oi}^{(k)} \\ \vdots \\ y_{oi}^{(k)} \end{bmatrix} \right) \right|\right|_2 \qquad \text{for } i = 1, 2, \dots, n. \tag{2.20}$$

This can be concisely written as follows

$$\min_{\tilde{\boldsymbol{a}}_{i\cdot}^{(k)} \in \mathbb{R}^{m+1}} ||\boldsymbol{y}_{i\cdot}^{(k)\mathrm{T}} - \tilde{X}^{(k)\mathrm{T}} \tilde{\boldsymbol{a}}_{i\cdot}^{(k)\mathrm{T}}||_2 \qquad \text{for } i = 1, 2, \dots, n, \tag{2.21}$$

where

$$\tilde{x}_{ij}^{(k)} = \begin{cases} x_{ij}^{(k)} & \text{for } i \le m \\ 1 & \text{for } i = m+1 \end{cases} \qquad \tilde{a}_{ij}^{(k)} = \begin{cases} a_{ij}^{(k)} & \text{for } j \le m \\ y_{oi}^{(k)} & \text{for } j = m+1\,. \end{cases} \tag{2.22}$$

Noting that $\tilde{X}^{(k)\mathrm{T}}$ is an $l \times (m+1)$ matrix, the solutions of the least squares problems (2.21) are the least squares solutions of overdetermined systems of linear equations if $l > m+1$. Assuming $\mathrm{rank}(\tilde{X}^{(k)\mathrm{T}}) = m+1$ (see Section 2.2.3), the normal equations of the first kind (see [3]) provide the least squares solution:

$$\tilde{\boldsymbol{a}}_{i\cdot}^{(k)\mathrm{T}} = (\tilde{X}^{(k)}\tilde{X}^{(k)\mathrm{T}})^{-1}(\tilde{X}^{(k)}\boldsymbol{y}_{i\cdot}^{(k)\mathrm{T}}) \qquad \text{for } i = 1, 2, \dots, n. \tag{2.23}$$

The actual computation is done using QR decomposition for numerical stability reasons. As a result we obtain the matrix $A^{(k)}$ and the vector $\boldsymbol{y}_o^{(k)}$ such that $\boldsymbol{y}_{\cdot j}^{(k)} \approx A^{(k)}\boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{y}_o^{(k)}$ for $j = 1, 2, \ldots, l$.

2.2.2. Solving the underdetermined system of linear equations (line 2-3). The minimum norm solution of the underdetermined systems of linear equations (2.7)-(2.8) can be computed as the solution of the normal equation of the second kind (see [3]). Equation (2.8) can be rewritten as follows:

$$\boldsymbol{y}_{\cdot j}^{*} - A^{(k)}\boldsymbol{x}_{\cdot j}^{(k)} - \boldsymbol{y}_o^{(k)} = A^{(k)}(diag(\hat{\boldsymbol{x}}))(diag(\hat{\boldsymbol{x}}))^{-1}\boldsymbol{s}_{\cdot j}^{(k)} \tag{2.24}$$

for $j = 1, 2, \ldots, l$. Thus, by using the normal equations of the second kind, $\boldsymbol{s}_{\cdot j}^{(k)}$ can be expressed as follows:

$$\boldsymbol{s}_{\cdot j}^{(k)} = (diag(\hat{\boldsymbol{x}}))^2 A^{(k)\mathrm{T}} \left(A^{(k)}(diag(\hat{\boldsymbol{x}}))^2 A^{(k)\mathrm{T}}\right)^{-1} (\boldsymbol{y}_{\cdot j}^{*} - A^{(k)}\boldsymbol{x}_{\cdot j}^{(k)} - \boldsymbol{y}_o^{(k)}), \tag{2.25}$$

for $j = 1, 2, \ldots, l$. Again, the actual computation is done using QR decomposition.

2.2.3. Randomly perturbed target values (line 1-2). In line 1-2 of the algorithm, we generate randomly perturbed target values $\{\boldsymbol{y}_{\cdot j}^{*}\}_{j=1}^{l}$. This step is necessary in order to iteratively repeat Stage 1 of the algorithm: matrix $X^{(k)}$ in line 2-2 is required to be full rank to make the overdetermined system (2.5) uniquely solvable, and the randomly perturbed target values $\{\boldsymbol{y}_{\cdot j}^{*}\}_{j=1}^{l}$ guarantee that $X^{(k)}$ computed in line 2-3 is indeed full rank, as we now explain. A graphical interpretation of this requirement is given in Section 3.3 (Figures 3.3(c) and 3.3(d)) for a simple model problem. We first observe that each of the overdetermined systems (2.21) has a unique least squares solution if and only if rank $\tilde{X}^{(k)} = m + 1$. Further, rank $\tilde{X}^{(k)} = m + 1$ requires rank $X^{(k)} = m$. On the other hand, when solving the underdetermined system of linear equations (2.7)-(2.8), if we had not randomly perturbed the target (i.e., if we take $\boldsymbol{y}_{\cdot j}^{*} = \boldsymbol{y}^{*}$ for all j), then we would have the following relationships:

$$A^{(k)}\left(\boldsymbol{x}_{\cdot i}^{(k)} + \boldsymbol{s}_{\cdot i}^{(k)}\right) = A^{(k)}\left(\boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)}\right) \equiv \boldsymbol{b} \qquad \text{for any } i, j = 1, 2, \ldots l. \tag{2.26}$$

In other words,

$$[A^{(k)}, -\boldsymbol{b}] \begin{pmatrix} \boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)} \\ 1 \end{pmatrix} = \boldsymbol{0} \qquad \text{for } j = 1, 2, \ldots, l\,. \tag{2.27}$$

Hence, if we define $\tilde{A}^{(k)} = [A^{(k)}, -\boldsymbol{b}] \in \mathbb{R}^{n\times(m+1)}$, then

$$\begin{pmatrix} \boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)} \\ 1 \end{pmatrix} \in \mathcal{N}(\tilde{A}^{(k)}) \qquad \text{for } j = 1, 2, \ldots l, \tag{2.28}$$

where $\mathcal{N}(\tilde{A}^{(k)})$ denotes the null space of $\tilde{A}^{(k)}$. Also note that $\dim \mathcal{N}(\tilde{A}^{(k)}) = m + 1 - \text{rank } \tilde{A}^{(k)} < m + 1$ unless $\tilde{A}^{(k)} = 0$. (Note that rank $\tilde{A}^{(k)} = \text{rank } A^{(k)}$.) Hence, if $\boldsymbol{x}_{\cdot j}^{(k+1)}$ is chosen to be $\boldsymbol{x}_{\cdot j}^{(k)} + \boldsymbol{s}_{\cdot j}^{(k)}$, rank $\tilde{X}^{(k+1)} \le \dim \mathcal{N}(\tilde{A}^{(k)}) < m + 1$. This would imply that each least squares problem (2.21) in the the $k + 1$th iteration is rank deficient and does not have a unique solution. On the other hand, if we seek the minimum norm solution of the underdetermined system of linear equations (2.7)-(2.8) for randomly perturbed targets $\{\boldsymbol{y}_{\cdot j}^{*}\}_{j=1}^{l}$, then generically rank $\tilde{X}^{(k+1)} = m + 1$ and each least squares problem (2.21) in the $k+1$th iteration will have a unique solution.

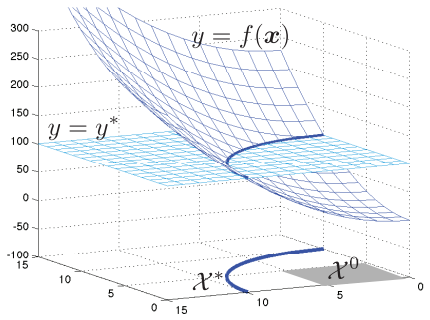


FIG. 3.1. *Example 1: the function f and the solution set $\mathcal{X}^*$ (in the first quadrant).*

2.2.4. Some further implementation details. Note that the function evaluations at each point in the cluster in lines 2-1 and 4-1 are independent of each other. Hence, these lines can be implemented in an embarrassingly parallel way. Since most of the computational cost is spent on computing the function values in these two lines, the computation time required by the Cluster Newton method is almost inversely proportional to the number of CPU cores that can be utilized. Also, for simplicity of presentation, we have fixed the number of iterations for each of the stages (via parameters K_1 and K_2). However, one can easily modify the implementation so that the iteration stops once a desired accuracy has been achieved (e.g., $\boldsymbol{x} \in \mathcal{X}^*_\epsilon$ in Equation (1.2)).

3. Simple model problem (Example 1). Before we attempt to solve the parameter identification problem of the pharmacokinetics model, we illustrate our algorithm using a simple example that is easy to visualize.

3.1. Model problem description. Our model problem is as follows: find a set of l points in $\mathbb{R}^2$ near a box $\mathcal{X}^0$, such that

$$f(\boldsymbol{x}) = y^*\,, \tag{3.1}$$

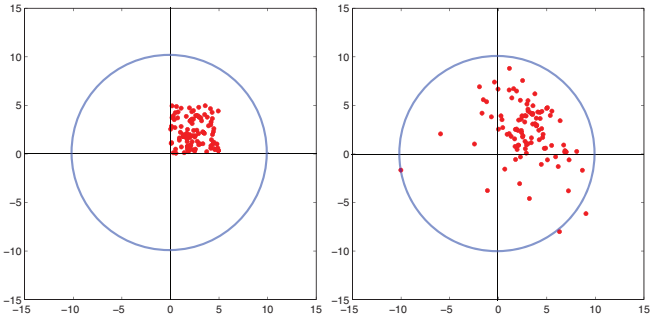
where

$$f(\boldsymbol{x}) = (x_1^2 + x_2^2) + \frac{1}{100}\sin(10000x_1)\cdot\sin(10000x_2)\,, \tag{3.2}$$

$$y^* = 100\,, \tag{3.3}$$

$$\mathcal{X}^0 = \{\boldsymbol{x} \in \mathbb{R}^2 \,:\, \max_{i=1,2} |\,(x_i - 2.5)\,/2.5| < 1\}\,. \tag{3.4}$$

The function is a paraboloid perturbed by a wildly oscillatory perturbation with a small amplitude. As depicted in Figure 3.1, the solution set of this inverse problem $\mathcal{X}^*$ is approximately a circle of radius 10 centred at the origin in the x_1-x_2 plane. Thus, we aim to find the points on this curve $\mathcal{X}^*$ near the box $\mathcal{X}^0$. The perturbation mimics 'roughness' or 'noise' that can be found in many realistic high-dimensional applications. For example, as we will illustrate in the following section, when the forward problem involves numerical solution of a system of ODEs, a similar kind of 'roughness' can be observed for the function evaluation, caused by numerical error. The initial box $\mathcal{X}^0$ may signify some *a priori* knowledge about where physically relevant solutions are expected. We choose $l = 100$ in the numerical examples below.



(a) Initial points in $\mathcal{X}^0$. (b) Final Points obtained by the LM method.

FIG. 3.2. *Example 1: an attempt with the LM method.*

3.2. Levenberg-Marquardt Method. We first discuss how the well-known LM method (see, e.g., [3]) performs when applied to this problem. We create l random points in $\mathcal{X}^0$ and then apply the LM method using each random point as an initial point. We have used the LM implementation in the MATLAB optimization toolbox (version 2010b) with default parameters for our numerical experiment. We observe that the algorithm terminates with the error "Algorithm appears to be converging to a point that is not a root" for all initial points we tried. As can be seen in Figure 3.2, we fail to find points close to the solution set $\mathcal{X}^*$.

3.3. Cluster Newton (CN) Method. We now use the Cluster Newton method to find multiple points on the level curve. Noting that this example is a special case of (1.1) with $m = 2$ and $n = 1$, we directly apply the algorithm presented in Section 2 with the following user-defined parameters:

$$\text{typical values for } \boldsymbol{x} \qquad \hat{\boldsymbol{x}} = [2.5, 2.5]^{\mathrm{T}}, \tag{3.5}$$

$$\text{relative ranges for } \boldsymbol{x} \qquad \boldsymbol{v} = [1, 1]^{\mathrm{T}}, \tag{3.6}$$

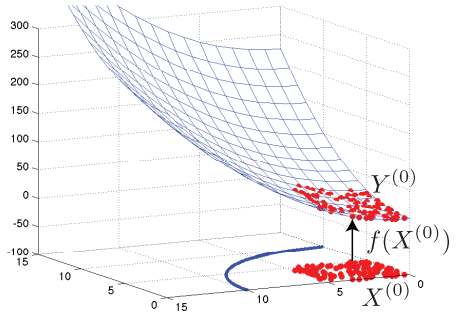
$$\text{domain of } \boldsymbol{f} \qquad \mathcal{X} = \mathbb{R}^2, \tag{3.7}$$

$$\text{number of Stage 1 iterations} \qquad K_1 + 1 = 6 \quad (\text{so } K_1 = 5), \tag{3.8}$$

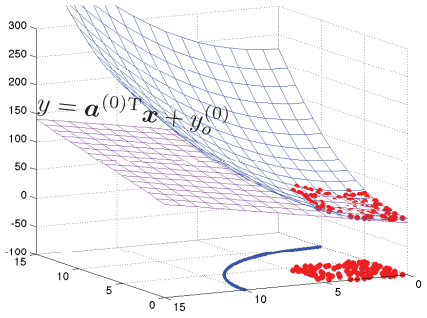
$$\text{number of total iterations} \qquad K_2 + 1 = 24 \quad (\text{so } K_2 = 23). \tag{3.9}$$

In order to illustrate the fundamental idea of the algorithm, Stage 1 of the Cluster Newton method is graphically explained in Figure 3.3 using this example. As can be seen in Figure 3.3(b), the Cluster Newton method constructs a linear approximation from points in the cluster and computes the gradient using all points in the cluster. Also, it constructs only one linear approximation in each iteration, so only one function evaluation per iteration per point in the cluster is required. In contrast, when using MATLAB's LM with the default finite-difference Jacobian computation, each iteration requires $m + 1 = 3$ function evaluations per point. This illustrates the first major point in comparing our collective CN method with one-by-one LM: the one-by-one approach requires about $m+1$ times more function evaluations per nonlinear iteration than the collective approach.

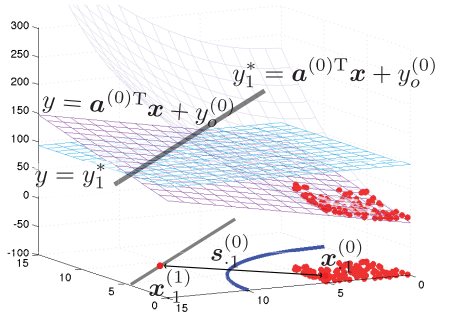
Panels (c) and (d) of Figure 3.3 show how new points $\boldsymbol{x}_{\cdot j}^1$ are generated on the intersections of the linear approximation $y = \boldsymbol{a}^{(0)\mathrm{T}}\boldsymbol{x} + y_o^{(0)}$ and the constant planes $y = y_j^*$, by determining update vectors $\boldsymbol{s}_{\cdot j}$ that are orthogonal to those intersections (line



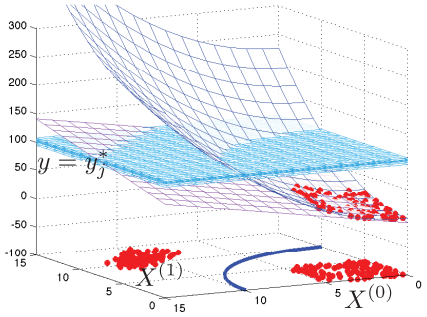
(a) Line 2-1: Evaluate f at each point in $X^{(0)}$ to generate $Y^{(0)}$.



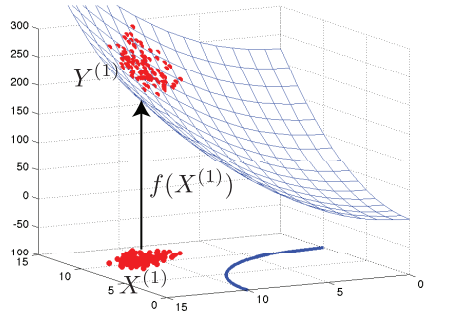
(b) Line 2-2: Construct a linear approximation $y = \boldsymbol{a}^{(0)\mathrm{T}}\boldsymbol{x} + y_o^{(0)}$.



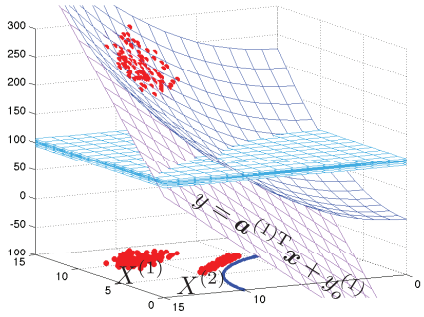
(c) Line 2-3: Find the update vectors $\boldsymbol{s}_{\cdot j}$.



(d) Line 2-4: Find a new cluster of points $X^{(1)}$.



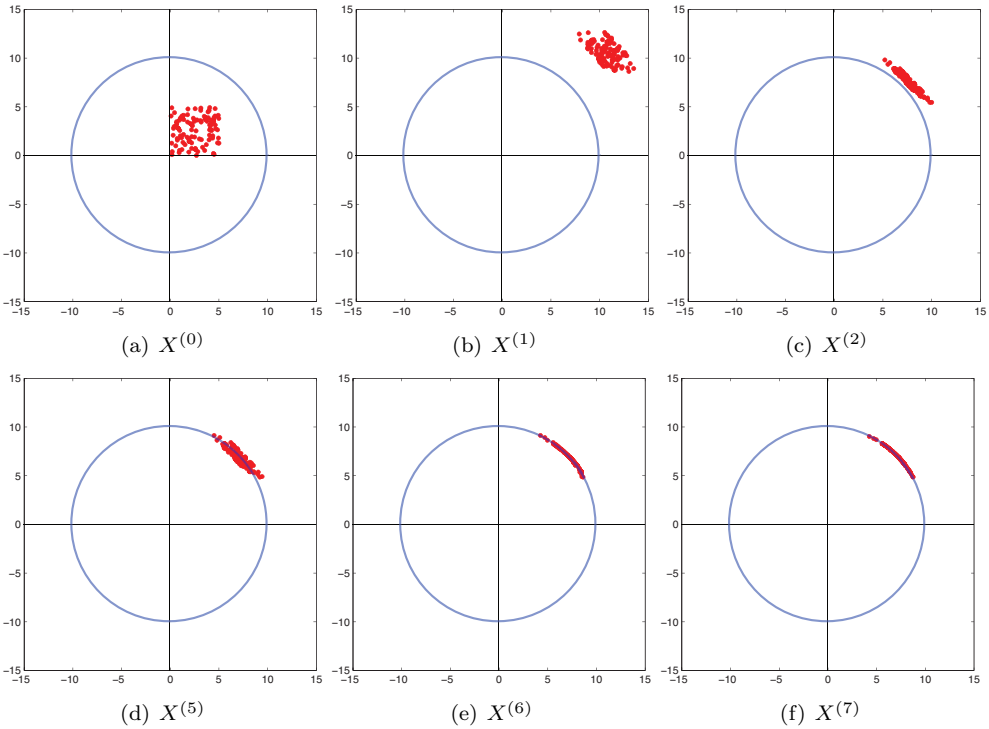
(e) Line 2-1: Evaluate f at each point in $X^{(1)}$ to generate $Y^{(1)}$.



(f) Line 2-2,...,2-4: Obtain $X^{(2)}$

FIG. 3.3. *Example 1: Graphical description of Stage 1 of the Cluster Newton method.*

2.3 of the pseudocode). This also explains graphically why it is necessary to use target values y_j^* that are randomly perturbed from y^*: without random perturbation, all points in the cloud $X^{(1)}$ would lie on the line $y^* = \boldsymbol{a}^{(0)\mathrm{T}}\boldsymbol{x} + y_o^{(0)}$, and matrix $X^{(1)}$ would be rank-deficient. This would preclude the computation of a linear approximation to $f(X^{(1)})$ in line 2-2 and the algorithm would break down.



(a) $X^{(0)}$ (b) $X^{(1)}$ (c) $X^{(2)}$

(d) $X^{(5)}$ (e) $X^{(6)}$ (f) $X^{(7)}$

FIG. 3.4. *Example 1: Plots of the points in* $X^{(k)}$. $X^{(0)}$ *to* $X^{(5)}$ *correspond to Stage 1, and* $X^{(6)}$ *and* $X^{(7)}$ *to Stage 2.*

Figure 3.4 shows how the two stages of our algorithm locate solutions in the solution set guided by the initial cluster. The size of the final cluster in the direction parallel to the solution set is similar to the size of the initial cluster as specified by the typical relative range $\boldsymbol{v}$.

Recall from Figure 3.2 that the LM method was not able to find solutions for this problem. In fact, this is not surprising due to the wildly oscillating nature of the function $f(x)$, and it would not be difficult to make LM converge better by applying some kind of smoothing to $f(x)$. However, the fact that the CN method can find solutions without such smoothing points to an important advantage of the CN method in addition to its low computational cost per nonlinear iteration: the CN method benefits from an inherent smoothing provided by the least-squares fitting, which results in improved robustness.

4. Pharmacokinetics ODE Coefficient Identification Problem (Example 2). We now introduce the original problem that led us to construct the Cluster Newton algorithm for simultaneously finding multiple solutions of an underdetermined inverse problem. This inverse problem can be categorized as a coefficient identification problem of a system of ODEs.

4.1. Forward Problem: Physiologically Based Pharmacokinetics Model. Physiologically Based Pharmacokinetics (PBPK) models are ODE-based mathematical models for transportation, metabolization, and elimination of a drug in the human body. In this paper, we consider the whole-body PBPK model of the anti-cancer drug

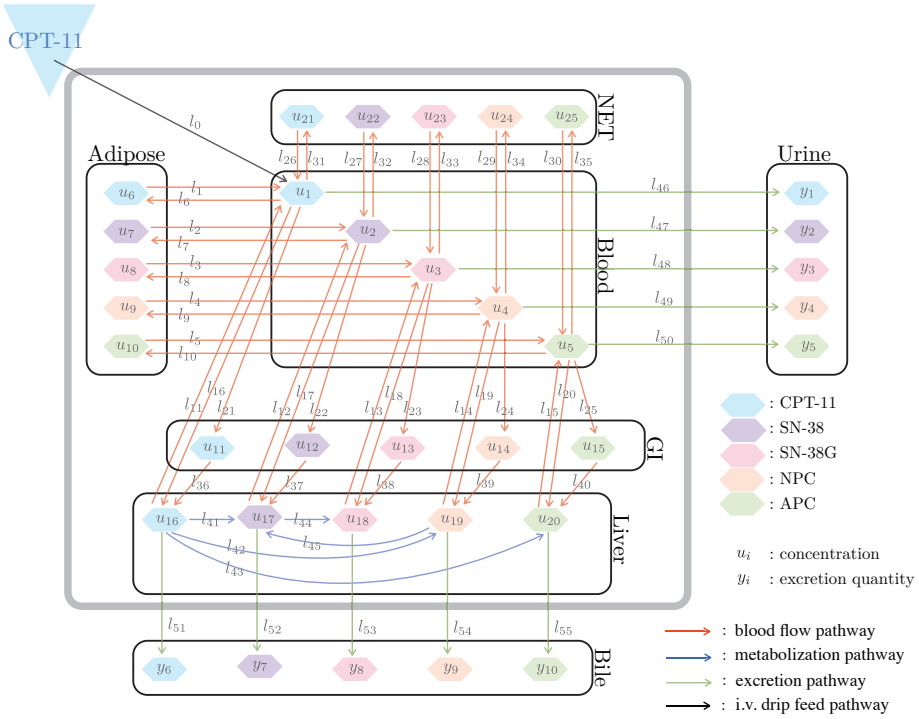


FIG. 4.1. *Schematic diagram of the PBPK model.*

CPT-11 by Arikuma et al. [2]. This PBPK model is presented as a schematic diagram in Figure 4.1. The model considers five body compartments and the concentration of the drug (CPT-11) and its metabolites (SN-38, SN-38G, NPC, and APC) in the five organs are represented as the time-dependent variables $u_1(t), u_2(t), \ldots, u_{25}(t)$. The blood flow pathway links between the organs are represented by $l_1, l_2, \ldots, l_{40}$ and these flows are modelled by diffusion processes. The metabolic reactions in the liver are represented by $l_{41}, l_{42}, \ldots, l_{45}$ and these reactions are modelled by Michaelis-Menten kinetics. The excretion pathways are represented by $l_{46}, l_{47}, \ldots, l_{55}$ and these pathways are also modelled by diffusion processes. For more details about the PBPK model we refer the reader to Appendix B of technical report [1], which is freely available online. In particular, explicit expression for the interaction terms $l_1, l_2, \ldots, l_{55}$ of Figure 4.1 are given in Equations B.2-B.4 of [1].

Since drug and metabolites have different diffusion rates for different organs and different metabolic reaction rates for different enzymes, the model contains many kinetic parameters. In addition to these parameters, the PBPK model contains parameters associated with the physiological characteristics of the patient and the drug administration schedule. In the PBPK model of interest, there are 60 parameters and we denote these parameters by $x_1, x_2, \ldots, x_{60}$. These parameters have some constraints originating from the biological quantities that they are representing, which are discussed in Section 4.2.2. The model parameters with their typical values are described in Tables B.1-B.5 of [1]. To save space we refer the reader to [1] for the precise expressions of the ODEs for the unknown functions $u_1(t), u_2(t), \ldots, u_{25}(t)$, but we fully specify the ODE here for one of the 25 unknown functions, $u_{18}(t)$, the

TABLE 4.1
Computational costs of numerically solving the forward problem for Example 2. The computational time was measured on a server machine with Intel Xeon X7350 3GHz processors.

Absolute/Relative tolerance δ_{ODE}	Computational time (sec)
10^{-3}	0.12
10^{-6}	0.37
10^{-9}	0.76

concentration of SN-38G in the Liver, as an illustrative example:

$$\frac{\mathrm{d}u_{18}(t)}{\mathrm{d}t} = \frac{(l_{18} + l_{38} + l_{44}) - (l_{13} + l_{53})}{x_{57}}\,, \tag{4.1}$$

where

$$l_{18} = x_{53}\,u_3(t)\,, \qquad l_{38} = \frac{x_{52}}{x_8}\,u_{13}(t)\,, \qquad l_{44} = \frac{x_{45}\,x_{50}\,x_{57}}{\frac{x_{40}\,x_{12}}{x_{22}\,u_{17}(t)} + 1}\,, \tag{4.2}$$

$$l_{13} = \frac{x_{52} + x_{53}}{x_{13}}\,u_{18}(t)\,, \qquad l_{53} = \frac{x_{33}\,x_{23}}{x_{13}}\,u_{18}(t)\,. \tag{4.3}$$

With ODE expressions similar to (4.1) for all variables, we obtain the ODE system

$$\frac{\mathrm{d}\boldsymbol{u}(t;\boldsymbol{x})}{\mathrm{d}t} = \boldsymbol{h}(\boldsymbol{u}(t;\boldsymbol{x});\boldsymbol{x})\,, \tag{4.4}$$

where $\boldsymbol{u}$ is a 25-dimensional vector-valued function, $\boldsymbol{h}$ is the nonlinear function associated with the right-hand side of the ODE system (see Equation B.5 of [1] for a complete description of $\boldsymbol{h}$) and $\boldsymbol{x}$ is a parameter vector in $\mathbb{R}^{60}$.

Next, we construct a function $\boldsymbol{f}$ that maps the parameters of the PBPK model $\boldsymbol{x}$ to the total quantities of CPT-11 and its metabolites excreted in urine and bile (faeces), which are the quantities that are experimentally measured and correspond to $\boldsymbol{y}^*$ in our inverse problem $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}^*$. We shall refer to a set of total quantities of excreted CPT-11 and its metabolites as *an excretion profile*. As we will discuss in the next section, these quantities are clinically measurable through mass-balance studies, and the main problem of this chapter boils down to estimating the model parameters from a clinically measured excretion profile. The excretion profile values (which are labeled $y_1, y_2, \ldots, y_{10}$ in Figure 4.1) are obtained by long-time integration of a subset of the ODE functions, so the function that maps the parameters to the excretion profile are defined by the following integrals:

$$f_i(\boldsymbol{x}) = \begin{cases} \int_0^T x_{i+25} \cdot x_{i+20} \cdot u_i(t;\boldsymbol{x})\,\mathrm{d}t & \text{for } i = 1,\ldots,5 \\ \int_0^T (x_{i+25} \cdot x_{i+15})/x_{i+5} \cdot u_{i+10}(t;\boldsymbol{x})\,\mathrm{d}t & \text{for } i = 6,\ldots,10 \end{cases} \tag{4.5}$$

where $\boldsymbol{f}$ maps from $\mathbb{R}^{60}$ to $\mathbb{R}^{10}$ and $T = 10^5\,s$. Note that $f_1, f_2, \ldots, f_5$ are proportional to the area under the curves (AUCs) of the drug and metabolites concentrations in the blood compartment, and $f_6, f_7, \ldots, f_{10}$ are proportional to the AUCs of the drug and metabolites concentrations in the Liver compartment. We shall refer to the map $\boldsymbol{f}$ to be the *forward problem*.

In practice, we compute the integrals in Equation (4.5) numerically by adding ten additional ODEs to the ODE system for u. In this way, we end up with a forward

TABLE 4.2
The amount of drug and its metabolite in excreta in units [nmol/kg] calculated from published data of Slatter et al. [17].

		Patient 1
CPT-11 in Urine	y_1^*	859.0
SN-38 in Urine	y_2^*	35.5
SN-38G in Urine	y_3^*	473.9
NPC in Urine	y_4^*	3.55
APC in Urine	y_5^*	305.0
CPT-11 in Bile + Faeces	y_6^*	975.4
SN-38 in Bile + Faeces	y_7^*	127.1
SN-38G in Bile + Faeces	y_8^*	105.4
NPC in Bile + Faeces	y_9^*	24.5
APC in Bile + Faeces	y_{10}^*	219.4
total dosage	$\sum_{i=1}^{10} y_i^*$	3946

ODE system with $q = 35$ equations and $m = 60$ parameters, and with $n = 10$ measured values that are all measured at a single time $t = T$, resulting in a highly underdetermined inverse problem to compute the $m = 60$ parameters from the $n = 10$ measured values. In each forward computation we solve this system of $q = 35$ ODEs using the MATLAB 2010b stiff ODE solver ODE15s [16] with error tolerance δ_{ODE}, i.e., we set RelTol= δ_{ODE} and AbsTol= δ_{ODE} for MATLAB ODE solver ODE15s, where δ_{ODE} is chosen as a function of the required numerical accuracy, see below. The computational costs of numerically solving this system of ODEs with various integration tolerances δ_{ODE} are tabulated in Table 4.1.

4.2. Inverse Problem: Parameter Identification of the Pharmacokinetics Model. We now wish to identify model parameters $x_1, x_2, \ldots, x_{60}$ using clinically measured excretion profiles. Since there are $m = 60$ model parameters and $n = 10$ clinically measurable excretion profile quantities, we have a highly underdetermined inverse problem.

4.2.1. Clinically measured excretion profile. We use the clinical data published by Slatter et al. [17]. Based on their data, we calculate the excretion profile of a patient as shown in Table 4.2. We use the clinically measured data of Patient 1 as the target for the output of the pharmacokinetics model: the goal is to determine multiple sets of parameter values that are consistent with the excretion profile of Patient 1.

4.2.2. Model parameters: Constraints, Typical value, and relative range. Model parameters $x_1, x_2, \ldots, x_{60}$ correspond to the biological quantities of a patient, and there are some constraints on these values. Firstly, all the parameters have to be positive real numbers. Secondly, as the volume of the adipose compartment is calculated by $1000-(x_{55}+x_{56}+x_{57}+x_{58})$, and the volume cannot be negative, we require $x_{55} + x_{56} + x_{57} + x_{58} \leq 1000$. We use the typical values of the model parameters derived through literature search and educated estimates by Arikuma et al. [2] and denote them as $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{60}$. These values are used to define the centre of the initial cluster in $\mathcal{X} \subset \mathbb{R}^m$, see Equation (2.1) These values are re-tabulated in Tables B.1-B.5 of [1]. We choose the typical relative ranges vi for the initial cluster in $\mathcal{X} \subset \mathbb{R}^m$ (see Equation (2.1)) to be $\pm 50\%$ for the kinetic parameters $(x_1, x_2, ..., x_{50})$, $\pm 30\%$ for the physiological parameters $(x_{51}, x_{52}, ..., x_{58})$ and $\pm 5\%$ for the drug administration

parameters (x_{59}, x_{60}), i.e., the relative ranges are

$$v_i = \begin{cases} 0.5 & \text{for } i = 1, 2, \ldots, 50 \\ 0.3 & \text{for } i = 51, 52, \ldots, 58 \\ 0.05 & \text{for } i = 59, 60. \end{cases} \tag{4.6}$$

The typical relative range of the kinetic parameters was chosen guided by the fact that the inter-subject variability of these values is usually less than $\pm 50\%$ as shown in [8, 9, 10, 18]. The typical relative range of the physiological parameters was motivated by [21]. The typical relative range of the drug administration parameters is chosen to be small since it is only influenced by the experimental precision of the drug administration procedure.

4.2.3. Statement of the inverse problem. We now state the PBPK model parameter identification problem as follows: find a set of l points in $\mathcal{X} \subset \mathbb{R}^{60}$ near a box $\mathcal{X}^0$, such that

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}^* \,, \tag{4.7}$$

where

$$\boldsymbol{f} \;:\; \mathcal{X} \subset \mathbb{R}^{60} \to \mathbb{R}^{10} \qquad \text{a function that maps the model parameters to the excretion profile, as defined in Section 4.1} \,, \tag{4.8}$$

$$\boldsymbol{y}^* \;:\; \text{clinically measured data from patient 1 as in Table 4.2} \,, \tag{4.9}$$

$$\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^{60} : x_i > 0 \text{ and } \sum_{i=55}^{58} x_i \leq 1000\} \,, \tag{4.10}$$

$$\mathcal{X}^0 = \{\boldsymbol{x} \in \mathbb{R}^{60} \;:\; \max_{i=1,2,\ldots,60} |\left(x_i - \hat{x}_i\right)/(\hat{x}_i v_i)| < 1\} \,. \tag{4.11}$$

So $m = 60$ and $n = 10$ for this problem. We note that evaluating the function value $\boldsymbol{f}(\boldsymbol{x})$ involves numerically solving the system of ODEs.

4.3. Levenberg-Marquardt Method. We first create random points in $\mathcal{X}^0$ and then apply the single-shooting LM method using each point as an initial guess. We do this computation in parallel as each run of the LM method is independent of the others. Due to a limitation of the MATLAB Parallel Computing Toolbox, we utilize at most 8 cores. A visual representation of the 1,000 randomly chosen points in $\mathcal{X}^0$ is given in Figure 4.2. Each red $\times$ indicates the average of the normalized parameter over all points. We have used the LM method implementation in the MATLAB optimization toolbox (version 2010b) with default parameters to find the root of the following function in our numerical experiment:

$$\tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}}) = \begin{cases} (diag(\boldsymbol{y}^*))^{-1} \boldsymbol{f}(diag(\hat{\boldsymbol{x}})\tilde{\boldsymbol{x}}) - 1 & \text{if } (diag(\hat{\boldsymbol{x}})\tilde{\boldsymbol{x}}) \in \mathcal{X} \\ 10^5 \left[1, 1, 1, 1, 1, 1, 1, 1, 1, 1\right]^{\mathrm{T}} & \text{otherwise} \end{cases} \,, \tag{4.12}$$

where $\tilde{\boldsymbol{x}}$ is a normalized model parameter vector. Note that we have used the normalization by $diag(\boldsymbol{y}^*)$ because it improved convergence. The way adopted in (4.12) to force solutions to lie in the domain of the function $\boldsymbol{f}$ turns out to work satisfactorily.

4.3.1. Visual representation of the solution found by the Levenberg-Marquardt method. Figure 4.3 visually represents the final set of points found by the LM method after 469,439 function evaluations using an error tolerance of

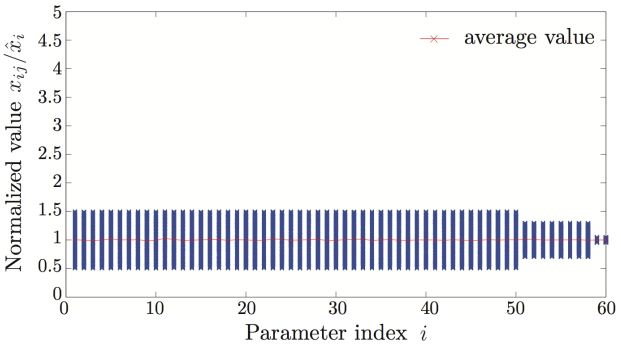


FIG. 4.2. *Example 2: Initial set of points in* $\mathcal{X}^0$.

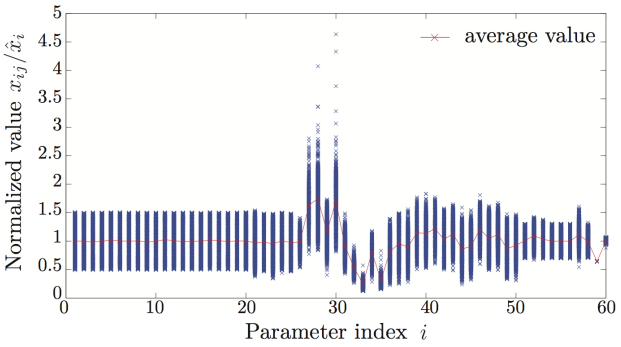


FIG. 4.3. *Final set of points found by the LM method.*

$\delta_{ODE} = 10^{-9}$ when solving the system of ODEs. Note that this small tolerance is required for the LM method to converge, while the Cluster Newton method converges with a significantly less restrictive tolerance (see below). Note also that each function evaluation corresponds to one forward solve of the ODE system with $q = 35$ equations.

4.3.2. Speed and accuracy obtained with the single-shooting Levenberg-Marquardt method. In Figure 4.4(a), the relative residual was plotted against the number of iterations. For Example 2, we define the relative residual of parameter vector $\boldsymbol{x}_{\cdot j}^{(k)}$ as

$$r(\boldsymbol{x}_{\cdot j}^{(k)}) = \max_{i=1,2,\ldots,10} \left| \frac{y_{ij}^{(k)} - y_i^*}{y_i^*} \right|, \tag{4.13}$$

where $\boldsymbol{y}_{\cdot j}^{(k)} = \boldsymbol{f}(\boldsymbol{x}_{\cdot j}^{(k)})$ with $\delta_{ODE} = 10^{-11}$. We use $\delta_{ODE} = 10^{-11}$ for computing $\boldsymbol{y}_{\cdot j}^{(k)}$ in these residuals to make sure we obtain a residual that is close to the true residual that would be obtained when the function $\boldsymbol{f}$ is evaluated exactly. As can be seen in Figure 4.4(a), it takes on average seven iterations to find solutions accurate up to the accuracy of the function evaluation ($\delta_{ODE} = 10^{-9}$). We note that since the Jacobian of the function is not explicitly given, the MATLAB implementation of the LM method estimates the Jacobian by finite differences. Hence, in each iteration, the function is evaluated at least $m + 1 = 61$ times. In each function evaluation, we solve

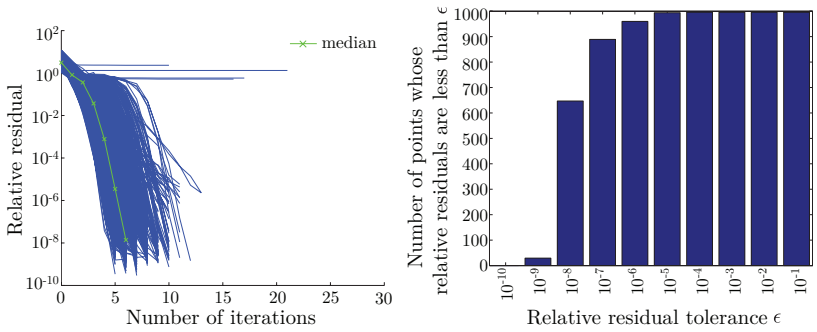
the system of ODEs to high accuracy. Thus, this method can be computationally very expensive (e.g., to obtain the solution presented in Figure 4.4(b), about 8 hours of computation is used on a server machine with two quad-core Intel Xeon X7350 3GHz processors).

In Figure 4.4(b), the number of points in the final set obtained by the LM method (after 469,439 function evaluations) whose relative residual is less than the relative residual tolerance ϵ is plotted. As can be seen in Figure 4.4(b), about 95% of the points achieve a relative residual less than 10^{-6} and about 65% of the points achieve a relative residual less than 10^{-8}.

As already stated in Section 1.1, we can obtain 1,000 different model parameter vectors of interest through multiple application of the LM method. However, this requires accurate function evaluation with tolerance $\delta_{ODE} = 10^{-9}$ and a large number of function evaluations per iteration. Thus, multiple application of the LM method to obtain multiple model parameter vectors is computationally very expensive.

It has to be noted, however, that more efficient and more robust parameter estimation methods exist than single-shooting LM: over the past decades many significant advances have brought the field of ODE parameter estimation methods to a high level of sophistication; see, for example, the works by Deuflhard [7] and Bock et al. [4]. In particular, one of the techniques that improve robustness is to consider a multiple-shooting strategy, which results in methods with increased robustness for cases where ODE solutions may not exist over the entire interval, and for cases where numerical stability issues may occur when solutions grow fast. A second important technique that may increase robustness is to compute the Jacobian numerically by adding qm additional equations (the so-called variational equations) to the set of ODEs to be solved. While this approach generally results in more accurate Jacobian evaluation than when using finite differences, it may be cumbersome to derive the additional ODEs (automatic integration may be used), and the resulting right-hand sides may be expensive to evaluate. For example, for our PBPK model, computing the Jacobian in this way would require solving an ODE system with $q(m+1) = 35 \times 61 = 2,135$ ODEs for each forward solve for the solution and the Jacobian. Even though these more advanced methods may be significantly more efficient and accurate than single-shooting LM with finite difference Jacobians, they would still require the equivalent of $m+1$ forward solves per solution point and per nonlinear iteration if applied in a one-by-one approach. Since our collective CN method only requires 1 forward solve per solution point and per nonlinear iteration, it is clear that the one-by-one approach would still be significantly slower even if more advanced methods than single-shooting LM are used.

4.4. Cluster Newton Method. We now show that our Cluster Newton method is a much more computationally efficient way to find multiple solutions of the underdetermined inverse problem than multiple applications of the LM method in a one-by-one fashion. The computational efficiency of the Cluster Newton method is due to the significantly smaller number of required function evaluations per iteration, with the added benefit of good robustness against 'roughness' (numerical error) in the forward evaluation. These characteristics follow from the collective way in which the points are updated and linear approximations are computed in Stage 1 of the Cluster Newton method. We directly apply the algorithm presented in Section 2 with $m = 60$



(a) Relative residual reduction.

(b) Accuracy of the final set of points after 469,439 function evaluations with error tolerance $\delta_{ODE} = 10^{-9}$.

FIG. 4.4. *Speed and accuracy of the LM method applied to Example 2.*

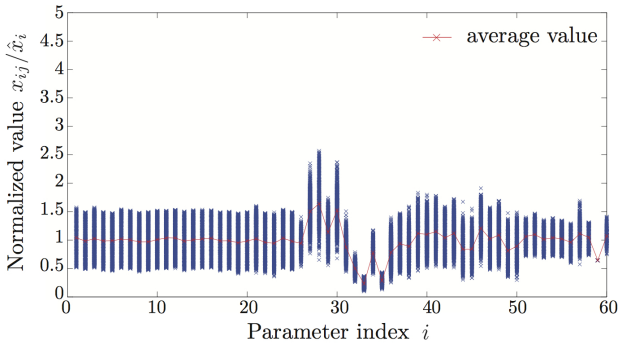


FIG. 4.5. *Example 2: Set of points $X^{(30)}$ found by the Cluster Newton method.*

and $n = 10$ and with the following user-defined parameters:

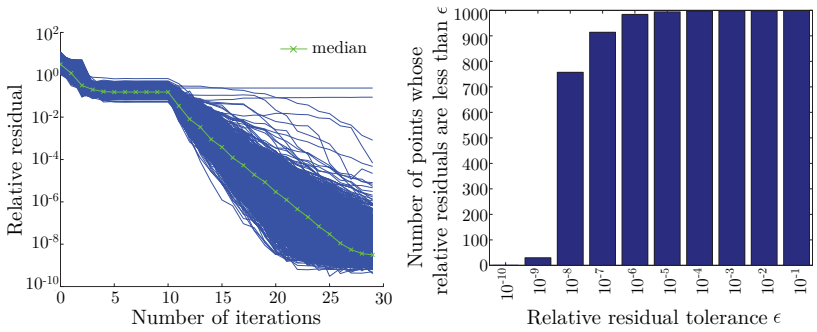
$$\text{number of points in the cluster} \qquad l = 1{,}000\,, \tag{4.14}$$

$$\text{number of Stage 1 iterations} \qquad K_1 + 1 = 11 \quad (\text{so } K_1 = 10)\,, \tag{4.15}$$

$$\text{number of total iterations} \qquad K_2 + 1 = 30 \quad (\text{so } K_2 = 29)\,. \tag{4.16}$$

4.4.1. Visual representation of the solution found by the Cluster Newton method. Figure 4.5 visually represents the final set of points found by the Cluster Newton method using a total of 30,000 function evaluations with $\delta_{ODE} = 10^{-9}$. The 30,000 function evaluations correspond to 30,000 forward solves of the ODE system with q = 35 ODEs: there are 1,000 solution points and 30 nonlinear iterations, with one forward solve of the ODE system per solution point and per nonlinear iteration. By comparing Figure 4.5 with Figure 4.3, one can observe that the sets of parameters found by the Cluster Newton method are generally similar to the sets of parameters found by multiple applications of the LM method.

4.4.2. Speed and accuracy obtained with the Cluster Newton method. In Figure 4.6(a), the relative residual calculated as in Equation (4.13) is plotted against the number of iterations. It can be seen that 30 iterations are sufficient to find



(a) Relative residual reduction.

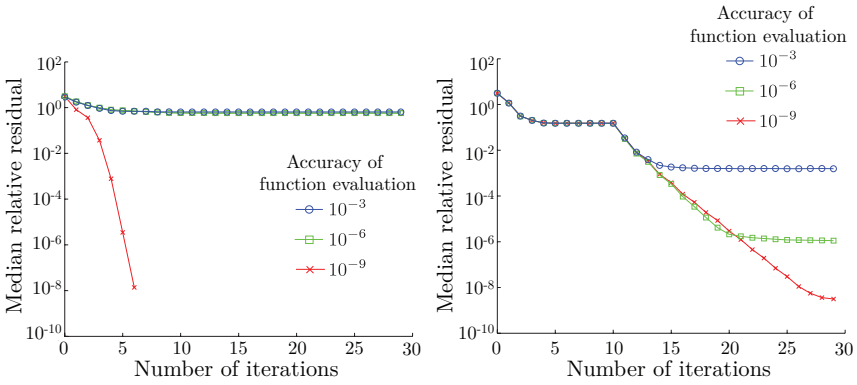
(b) Accuracy of the final set of points after 30,000 function evaluations with error tolerance $\delta_{ODE} = 10^{-9}$.

FIG. 4.6. *Example 2: Speed and accuracy of the Cluster Newton method.*

the solution accurate up to the accuracy of the function evaluation ($\delta_{ODE} = 10^{-9}$). We note that the Cluster Newton method only requires one function evaluation per point in the cluster per iteration. Thus, to get the final solution, only 30,000 function evaluations are necessary (recall that the LM method required 469,439 function evaluations). In Figure 4.6(b), the number of points in the final set obtained by the Cluster Newton method whose relative residuals are less than the relative residual tolerance ϵ is plotted. As can be seen in Figure 4.6(b), almost all the points achieve relative residual less than 10^{-6} and about 75% of the points achieve relative residual less than 10^{-8}. As the Cluster Newton method requires only one function evaluation per point in the cluster per iteration, we can obtain the solution presented in Figure 4.6(b) in about 30 minutes on a server machine with two quad-core of Intel Xeon X7350 3GHz processors, which is a factor of 16 faster than the LM method.

4.5. Comparison between the Levenberg-Marquardt method and the Cluster Newton method. We now further compare the LM method and the Cluster Newton method.

4.5.1. Robustness against small errors in the function evaluation. In Figure 4.7(a), we have plotted the median relative residual against the number of iterations with different accuracy of the function evaluation (δ_{ODE}). As can be seen from Figure 4.7(a), the function needs to be evaluated accurately in order for the LM method to find solutions with small relative residual. These numerical experiments show that $\delta_{ODE} = 10^{-9}$ or smaller is required for the MATLAB implementation of the LM method to stably find the solution. Note that this high accuracy of the forward problem ($\delta_{ODE} = 10^{-9}$) is required for single-shooting LM even when the solution to the inverse problem is sought with low accuracy, e.g., $\epsilon = 10^{-3}$ or larger. This is so because with larger δ_{ODE} the nonlinear solver may not converge to an approximate solution at all, due to numerical robustness issues. On the other hand, as can be seen from Figure 4.7(b), the Cluster Newton method finds solutions with accuracy close to the accuracy of the function evaluation for all values of δ_{ODE}. Thus, we observe that the Cluster Newton method is robust against small errors in the function evaluation caused by the numerical solution of the system of ODEs. This characteristic is especially advantageous if the desired accuracy for the solution of the inverse problem is not very high, so that we can reduce the accuracy of the numerical



(a) Levenberg-Marquardt method. (b) Cluster Newton method.

FIG. 4.7. *Influence of the accuracy of the function evaluation (δ_{ODE}).*

solution of the ODEs to save computational cost. Note again, however, that more advanced parameter identification methods could be used than single-shooting LM for computing solutions one-by-one, and they would be more efficient and accurate per solution than single-shooting LM [7, 4].

4.5.2. Computational cost. In Figure 4.8, we have plotted the relative residual versus the computational time. We have conducted this numerical experiment on a server machine with two quad-core Intel Xeon X7350 3GHz processors with fully-parallelized code using the MATLAB parallel computing toolbox for both methods.

In order for the Cluster Newton method to take advantage of the robustness against the numerical error of the function evaluation, we have implemented the Cluster Newton method so that the error tolerance of the function evaluation is initially set to 10^{-3} and then decreases as the number of iterations increases. The LM method requires 61 times more function evaluations per iteration in order to estimate the Jacobian by finite differences, and the function evaluation tolerance has to be less than 10^{-9} in order for the method to converge. Thus the computational time required by the LM method is significantly greater than the time required for the Cluster Newton method when seeking multiple solutions of the underdetermined inverse problem. This difference in computational time becomes prominent if the desired accuracy of the solution is not very high. For example, if one only requires to find solutions whose relative residual is around 10^{-3}, then the Cluster Newton method takes only about 5 minutes to find 1,000 solution vectors. However, the MATLAB implementation of the LM method requires over 7 hours in order to find a similar set of solutions. Here too, however, we have to caution that the execution time for the one-by-one approach may be reduced significantly if more advanced methods than single-shooting LM are used [7, 4]. Nevertheless, it is expected that our collective CN approach will still be more efficient, because the collective approach requires only 1 forward solve per solution point and per nonlinear iteration, while the one-by-one approaches require the equivalent of $m + 1$ forward solves per solution point and per nonlinear iteration. It is also interesting to say a few more words about the robustness of the CN method compared to advanced parameter estimation methods. The CN method is robust against numerical noise in the forward ODE solves due to the natural smoothing induced by the least-squares estimation of the Jacobian. This is in some sense a mech-

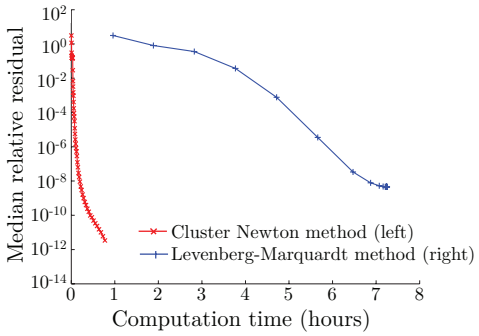


FIG. 4.8. *Comparison of computational cost between the Levenberg-Marquardt method and the Cluster Newton method.*

anism for increased robustness that is an alternative to computing the Jacobian using additional ODEs in multiple shooting methods (rather than using finite differences), which improves robustness for those types of methods. On the other hand, our current implementation of the CN method is still subject to other types of robustness issues that may arise in the forward problem in cases where ODE solutions may not exist over the entire interval, and in cases where numerical stability may be compromised due to fast growth of the solution. (These are the issues that multiple-shooting methods are more robust against.) These issues did not occur for the PBPK problem we considered, but they may arise for other problems. It may be possible to improve the robustness of our collective CN method using ideas from multiple shooting methods, or, alternatively, to derive collective versions of multiple-shooting methods making use of the principles behind the CN method, and this is subject of further research.

4.6. A brief numerical comparison with other algorithms. In addition to the LM method, we have conducted numerical experiments using two other nonlinear solvers combined with single shooting: Broyden's method and a Genetic Algorithm.

When the Cluster Newton method is run without Stage 1, it is essentially Broyden's method. We first use a finite difference scheme to approximate the Jacobian at each initial point. Using this as the initial Jacobian approximation, we have conducted a numerical experiment using Stage 2 of the Cluster Newton method (Broyden's method). From the numerical experiments, we have observed that, even after as many as 50 iterations, Broyden's method finds only about 350 points (out of 1,000 points) in $\mathcal{X}^*_{10^{-8}}$ compared to about 750 points for the Cluster Newton method. Also, as this approach requires approximating the initial Jacobian by a finite difference scheme, it takes 110,000 function evaluations in total (compared to 30,000 for the fully converged Cluster Newton result). Thus, we observe that Stage 1 of the Cluster Newton method is essential for its accuracy and computational efficiency.

Also, we have attempted to solve this inverse problem using the Genetic Algorithm implemented in the MATLAB optimization toolbox. Even after much trial and error with different parameters for the algorithm, we were only able to obtain solutions with relative residual larger than 10^{-1} after 72,000 function evaluations, and the method was not able to find any solution in $\mathcal{X}^*_{10^{-1}}$. Thus the Genetic Algorithm was not able to obtain accurate solutions.

5. Extensions of the Cluster Newton method. In this section we describe two extensions of the Cluster Newton method, namely, a nonlinear preconditioning

approach to improve convergence for highly nonlinear problems, and an approach to obtain solution points spread out over the entire solution set.

5.1. Preconditioned Cluster Newton method. Since our algorithm tries to approximate the function $\boldsymbol{f}$ by a hyper-plane in a large domain, the accuracy of the solution obtained by our algorithm is influenced by how close $\boldsymbol{f}$ is to a linear function. By choosing an appropriate preconditioning function $\mathbf{g}$ so that $\boldsymbol{f} \circ \boldsymbol{g}^{-1}$ is close to linear, and applying the Cluster Newton method to the composite function $\boldsymbol{f} \circ \boldsymbol{g}^{-1}$, we can increase the speed of the algorithm and the chance of finding a solution. It is often not trivial to find such a function. However, for our ODE coefficient identification problem, it is not very difficult to find one. For example, from Equation (4.1), we can observe that the x_i interact with each other mostly by multiplications and divisions. Thus, by redefining the parameters x_i by $x_i = e^{\check{x}_i}$ we can make them interact by additions and subtractions, i.e.,

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}u_{18} = &\left(e^{\check{x}_{53}-\check{x}_{57}} \cdot u_3(t) + e^{\check{x}_{52}-\check{x}_8-\check{x}_{57}} \cdot u_{13}(t) + \frac{e^{\check{x}_{45}+\check{x}_{50}}}{e^{\check{x}_{40}+\check{x}_{12}-\check{x}_{22}}/u_{17}(t)+1}\right) \\ &- \left((e^{\check{x}_{52}-\check{x}_{13}-\check{x}_{57}} + e^{\check{x}_{53}-\check{x}_{13}-\check{x}_{57}}) \cdot u_{18}(t) - e^{\check{x}_{33}+\check{x}_{23}-\check{x}_{13}} \cdot u_{18}(t)\right) .\end{aligned}$$

In addition to making the parameters of the function $\boldsymbol{f}$ interact almost linearly, this choice of redefinition of $\boldsymbol{x}$ has the benefit that the values of $\boldsymbol{x}$ remain positive. This helps us reduce the complication of points going outside of the domain where the function $\boldsymbol{f}$ is defined. This redefinition of $\boldsymbol{x}$ can be done easily for our algorithm by setting the preconditioning function as $\boldsymbol{g}(\boldsymbol{x}) = \ln(\boldsymbol{x})$ and directly applying the Cluster Newton method to $\boldsymbol{f} \circ \boldsymbol{g}^{-1}$.

The effect of this preconditioning function can be demonstrated by solving Example 2 with an initial set of points with larger than usual relative range that brings the initial box close to the boundary of the domain $\mathcal{X}$, i.e.,

$$v_i = \begin{cases} 0.95 & \text{for } i = 1, 2, \ldots, 50 \\ 0.3 & \text{for } i = 51, 52, \ldots, 58 \\ 0.05 & \text{for } i = 59, 60. \end{cases} \tag{5.1}$$

Figure 5.1 plots the median of the relative residual versus the number of iterations for the Cluster Newton method with or without preconditioning. As can be seen from Figure 5.1, without preconditioning, the Cluster Newton method fails to find the parameters of interest. However, with preconditioning, the Cluster Newton method finds a set of points with small relative residual. For the numerical results presented in Sections 3 and 4 preconditioning was not necessary and was thus not used.

5.2. Global Cluster Newton method. Finding multiple solutions of the underdetermined inverse problem near the initial cluster is useful when there is *a priori* knowledge about which part of the solution set is of interest. However, it is a natural question to ask whether we can sample multiple solutions in the solution set not necessarily close to the initial cluster. We can do so using our Cluster Newton method by small modifications to line 4-4 of the algorithm. We demonstrate this idea using Example 1, that is to say, instead of just seeking points on the contour curve near the initial cluster in box $\mathcal{X}^0$, we aim to find points on the entire contour curve. In order to achieve the above goal, we simply slide points in the direction tangent to the solution set once they are sufficiently close to the solution set, as illustrated in Figure 5.2. This can be done by replacing line 4-4 in Stage 2 of the algorithm with the following lines.

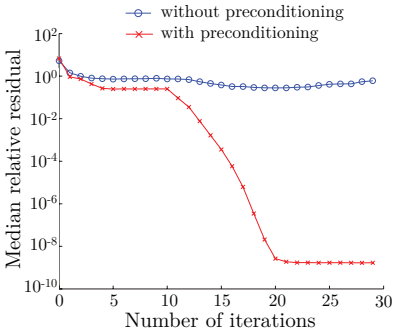


FIG. 5.1. *Median relative residual with or without preconditioning.*

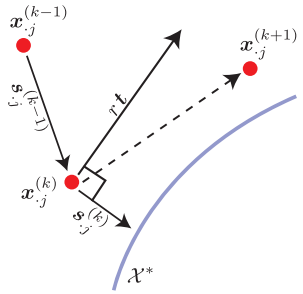
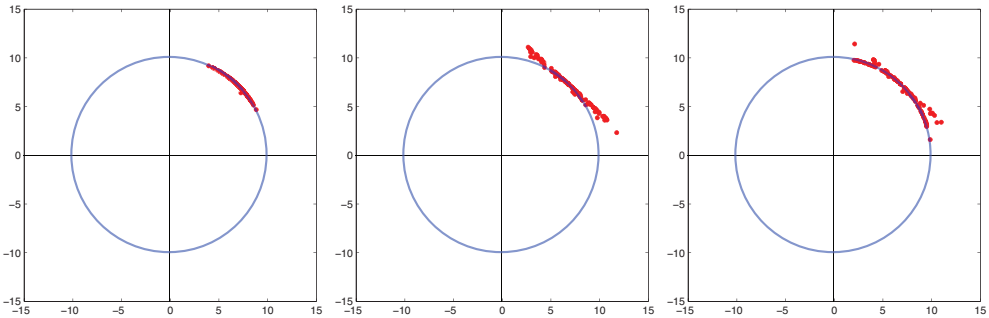


FIG. 5.2. *Main idea of the Global Cluster Newton method.*



(a) $X^{(6)}$ (b) $X^{(7)}$ (c) $X^{(8)}$

(d) $X^{(40)}$ (e) $X^{(79)}$ (f) $X^{(80)}$

FIG. 5.3. *Example 1: Plots of the points $X^{(k)}$ found by the Global Cluster Newton method. $X^{(0)}$ to $X^{(5)}$ are the same as in Figure 3.4 so the plots were omitted.*

4-4: If $k < K_3$

For $j = 1, 2, \ldots, l$

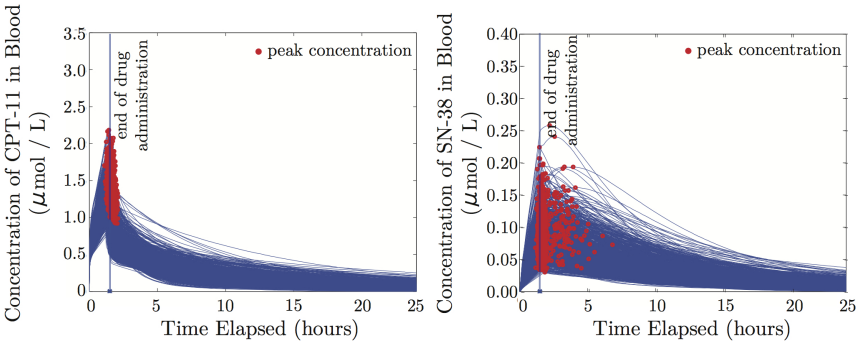
If $||\boldsymbol{s}_{\cdot j}^{(k)}||_2 < \xi$

$t_1 = -\boldsymbol{s}_{2j}^{(k)} / ||\boldsymbol{s}_{\cdot j}^{(k)}||_2$

$t_2 = \boldsymbol{s}_{1j}^{(k)} / ||\boldsymbol{s}_{\cdot j}^{(k)}||_2$

$\boldsymbol{s}_{\cdot j}^{(k)} = \boldsymbol{s}_{\cdot j}^{(k)} + r\boldsymbol{t}$

End if.



(a) Concentration of the anti-cancer drug CPT-11 (b) Concentration of the metabolite SN-38

FIG. 6.1. *1,000 model parameter sets found by the Cluster Newton method.*

End for.
End if.

$$X^{(k+1)} = X^{(k)} + S^{(k)}\,. \tag{5.2}$$

Hence, the algorithm spreads the points in the direction tangential to the solution set up to iteration K_3, ξ is a parameter that selects points sufficiently close to the solution set, and r is a randomly generated real number in a certain range symmetrically about 0 (see below) which is different every time it appears in the for-loop.

In Figure 5.3, we show the solutions found by the Global Cluster Newton method. For this numerical experiment, we have chosen $l = 100$, $K_1 = 5$, $K_2 = 100$, $K_3 = 80$, $\xi = 10^{-2}$ and r is a uniformly randomly generated number between -5 and 5. We can observe that the points are placed on the entire contour curve.

An extension of this idea is possible for higher-dimensional problems (e.g., Example 2). However, the tangential direction cannot be uniquely determined if the dimention of the solution set is greater than one. In such a case, we can choose the direction of the tangential vector $\boldsymbol{t}$ randomly within the tangent hyperplane each time it appears in the for loop.

6. Conclusion. We have introduced a new computationally efficient, easy to parallelize, and robust algorithm for simultaneously sampling multiple points from the solution set of an underdetermined inverse problem, for problems for which multiple solutions are of interest. Our algorithm was applied to a coefficient identification problem of a system of nonlinear ODEs modelling the drug kinetics of an anti-cancer drug, and we demonstrated that it efficiently traces the part of the solution set of interest. The efficiency and robustness of the algorithm follow from the collective way in which a linear approximation to the forward function is computed simultaneously for all the points in the cluster in Stage 1 of the algorithm. Multiple parameter vectors are of interest in this application because the ranges and extremal values of the parameters and solution profiles that can be obtained from multiple parameter vectors can potentially be used, for example, to assess or design treatment plans.

Using our algorithm, 1,000 sets of model parameters can be estimated with relative accuracy 10^{-3} from clinically observed data in half an hour using a five year old laptop computer (MacBook Pro with 2.33 GHz Intel Core2Duo processor with 4 GB of RAM). Figure 6.1 shows the predicted concentration of CPT-11 and SN-38 in

blood calculated using the model parameters found by the Cluster Newton method. Detailed comparison verifies that this solution and the solution obtained through multiple applications of the single-shooting LM method (cf. Figure 1.2), after 8 hours of computation using a server machine, are very similar. It is important to note that more efficient parameter identification methods may be used to compute the multiple solutions one-by-one than single-shooting LM, but it is expected that the performance advantage of the CN method would still be significant for computing many solutions at once, because CN requires only 1 ODE system solve (1 solve of the forward problem) per solution point and per nonlinear iteration, whereas one-by-one methods require the equivalent of $m + 1$ ODE system solves per solution point and per nonlinear iteration, where m is the number of parameters to be determined. We have demonstrated numerically that the CN method is highly efficient for an area-under-the-curve parameter identification problem with very few measurements for which many solutions are sought, but we believe that the CN method is a general framework for solving underdetermined inverse problems that may prove useful in many other problem areas.

We recognize that there are many ways to further improve our algorithm. For example, we can choose the number of Stage 1 iterations based on the residual, or we can use more sophisticated selection algorithms for the step size (i.e., the size of the update vectors $\boldsymbol{s}_{\cdot j}^{(k)}$ in lines 2-4 and 4-4). In the PBPK problem that motivated our work, statistical properties of the computed solution set are not a primary concern. However, in many other parameter identification problems statistical properties are important. We believe that there is significant potential for extending our collective CN method such that statistical properties can be taken into account (as is done in advanced parameter identification methods, see, e.g., [4, 5]), but this would require significant additional research and is subject of further work. Also, it would be interesting to explore making the CN method more efficient for ODE parameter identification problems by taking advantage of the structure of the ODE systems. Similarly to what has been done for the inverse problem of the drug kinetics model, we expect that applying our algorithm to other underdetermined inverse problems will efficiently provide useful information, and will also lead to new insights about the applicability of our algorithm.

7. Acknowledgements. We would like to thank Professor Kunio Tanabe of Waseda University for his extensive advice and useful comments on our research. We would like to thank Mr. Keiichi Morikuni of the Graduate University of Advanced Studies for useful discussions. The first author would like to thank the National Institute of Informatics and the University of Waterloo as the research opportunity which led to this paper was given to him by their memorandum of understanding internship program.

Appendix A. Matrix Notation. In general, we use a capital letter for a matrix, a bold lowercase letter for a vector and a lower case letter for a scalar quantity. Also,

we introduce the following matrix-related notations:

$$\boldsymbol{y} : \text{column vector}, \tag{A.1}$$
$$y_i : i\text{th component of column vector } \boldsymbol{y}, \tag{A.2}$$
$$x_{i\,j} : \text{element of matrix } X \text{ in column } i \text{ and row } j, \tag{A.3}$$
$$\boldsymbol{x}_{\cdot j} : j\text{th column of matrix } X \text{ as a column vector}, \tag{A.4}$$
$$\boldsymbol{x}_{i\cdot} : i\text{th row of matrix } X \text{ as a row vector}, \tag{A.5}$$
$$diag(\boldsymbol{y}) : \text{diagonal matrix whose } i\text{th column, } i\text{th row entry is } y_i, \tag{A.6}$$
$$||X||_F : \text{Frobenius norm of matrix } X. \tag{A.7}$$

REFERENCES

[1] Y. Aoki, K. Hayami, H. De Sterck, and A. Konagaya. Cluster Newton Method for Sampling Multiple Solutions of an Underdetermined Inverse Problem: Parameter Identification for Pharmacokinetics. *NII Technical Report*, National Institute of Informatics, Tokyo, 2011. Available at http://www.nii.ac.jp/TechReports/11-002E.html.

[2] T. Arikuma, S. Yoshikawa, R. Azuma, K. Watanabe, K. Matsumura, and A. Konagaya. Drug interaction prediction using ontology-driven hypothetical assertion framework for pathway generation followed by numerical simulation. *BMC Bioinformatics* , **9**(suppl 6), (2008), S11.

[3] Å. Björck, Numerical methods for least squares problems. Society for Industrial and Applied Mathematics, Philadelphia, 1996.

[4] H. G. Bock, E. Kostina, and J. P. Schloeder. Numerical Methods for Parameter Estimation in Nonlinear Differential Algebraic Equations. *GAMM-Mitteilungen*, **30**, (2007), pp. 376-408.

[5] M. Chung and E. Haber. Experimental Designfor Biological Systems. *SIAM J.ControlOptim.*, **50**, (2012), pp. 471-489.

[6] F. A. de Jong, J. J. Kitzen, P. de Bruijn, J. Verweij, and W. J. Loos. Hepatic transport, metabolism and biliary excretion of irinotecan in a cancer patient with an external bile drain. *Cancer Biology and Therapy* **5**(9) (2006), pp. 1105-1110.

[7] P. Deuflhard. Newton methods for nonlinear problems: affine invariance and adaptive algorithm. Springer-Verlag, Berlin-Heidelberg, 2004.

[8] J. F. Gagne, V. Montminy, P. Belanger, K. Journault, G. Gaucher, and C. Guillemette. Common human UGT1A polymorphisms and the altered metabolism of irinotecan active metabolite 7-ethyl-10-hydroxycamptothecin (SN-38). *Molecular Pharmacology* , **62**(3), (2002), pp. 608-617.

[9] M. Haaz, L. Rivory, C. Rich, L. Vernillet, and J. Robert. Metabolism of Irinotecan (CPT-11) by Human Hepatic Microsomes: Participation of Cytochrome P-450 3A and Drug Interactions. *Cancer Research*, **58**(3), (1998) , pp. 468-472.

[10] M. Haaz, C. Riche, L. P. Rivory, and J. Robert. Biosynthesis of an aminopiperidino metabolite of irinotecan [7-ethyl-10-[4-(1-piperidino)-1-piperidino]carbonyloxycamptothecine] by human hepaticmicrosomes. *Drug Metabolism and Disposition*, **26**(8), (1998), pp. 769-774.

[11] C. T. Kelley, Implicit Filtering. Society for Industrial and Applied Mathematics, Philadelphia, 2011.

[12] C. T. Kelley, Iterative Methods for Optimization. Society for Industrial and Applied Mathematics, Philadelphia, 1999.

[13] L. Kirkwood, R. Nation, and A. Somogri. Characterization of the human cytochrome P450 enzymes involved in the metabolism of dihydrocodeine. *British Journal of Clinical Pharmacology*, **44**(6), (2003), pp. 549-555.

[14] A. Konagaya, Bioinformatics (in Japanese). The Institute of Electronics, Information and Communication Engineering: Tokyo; 2009.

[15] A. Konagaya, Towards an In Silico Approach to Personalized Pharmacokinetics, in Meghea, Molecular Interactions. InTech, (2012), pp. 263-282.

[16] L. F. Shampine, and M. W. Reichelt. The MATLAB ODE suite. *SIAM Journal on Scientific Computing*, **18**(1), (1997), pp. 1-22.

[17] J. G. Slatter, L. J. Schaaf, J. P. Sams, K. L. Feenstra, M. G. Johnson, P. A. Bombardt et al. Pharmacokinetics, Metabolism, and Excretion of Irinotecan (CPT-11) Following I.V. Infusion of [14C]CPT-11 in Cancer Patients. *Drug Metabolism and Disposition*, **28**(4), (2000) , pp. 423-433.

[18] J. G. Slatter, P. Su, J. Sams, L. Schaaf, and L. Wienkers. Bioactivation of the Anticancer agent CPT-11 to SN-38 by human hepatic microsomal carboxylesterases and the in vitro sssessment of potential drug interactions. *Drug Metabolism and Disposition*, **25**(10), (1997), pp. 1157-1164.

[19] B. Wang. Parameter Estimation for ODEs using a Cross-Entropy Approach. Master's Degree Thesis, University of Toronto, Toronto, Ontario, 2012.

[20] J. J. Moré and S. M. Wild. Estimating Computational Noise. *SIAM Journal of Scientific Computing*, **33**(3), (2011), pp. 1292-1314.

[21] S. Willmann, K. Hohn, A. Edginton, M. Sevestre, J. Solodenko, W. Weiss et al. Development of a physiology-based whole-body population model for assessing the influence of individual variability on the pharmacokinetics of drugs. *Journal of Pharmacokinetics and Pharmacodynamics*, **34**(3), (2007), pp. 401-431.