Improving a Cross Entropy Approach to Parameter Estimation for ODEs and DDEs

Jonathan Calver

February 11, 2014

Abstract

We investigate and extend the cross entropy (CE) approach for parameter estimation for ODE and DDE models introduced in [27]. Software is developed to allow models to be easily specified for use with an existing package for working with DDEs. Our software implements CE and is used to explore potential improvements to the method. A 3-stage optimization procedure is presented. First, an inexpensive initial guess for some of the parameters is obtained to reduce the size of the search space. Second, a global search phase is performed to obtain an estimate for the parameters that is 'close' to the optimal values. Third, a local optimizer, that can converge super-linearly, is used to obtain the final estimates for the parameters. The performance of this 3-stage procedure is demonstrated by estimating parameters on several test problems from the literature. Additionally, we look at ways to reduce the computational cost of simulating the ODEs and DDEs during the estimation procedure.

1 Introduction

Differential equation based models are ubiquitous throughout the sciences. Due to model simplifications and measurement noise, models tend not to exactly fit collected data. Parameter estimation seeks to find the set of model parameters such that the model fits the collected data as well as possible, as defined by some objective function. The standard approach is the minimization of the sum of squared errors (SSE), which we describe in the next section.

Our goal is to develop a software package to facilitate least squares parameter estimation for ODE and DDE models. In order to achieve this, we require two things. First, we need a reliable routine for simulating ODEs and DDEs. Second, we need a suitable optimization algorithm. For these two requirements, we draw upon previous work at University of Toronto. The DDEM package [29] provides modules for simulating DDEs, as well as computing sensitivities and even estimating parameters using a sequential quadratic programming (SQP) algorithm. Work in [27] demonstrated how CE could be used for the parameter estimation we seek to perform. In the process of extending this software, we identify areas where the estimation procedure can be improved.

This leads to the development of a 3-stage optimization procedure. First, we solve a relatively inexpensive alternate optimization problem to obtain improved initial guesses for the parameters. Second, we run CE from this improved starting point. Third, we switch to a local optimizer once CE has gotten us 'close' to the optimal parameters. These last two stages form a hybrid optimization technique similar to that discussed in [1, 2].

In section 2, we review background information about parameter estimation for ODE and DDE models. The third section reviews the CE method and motivates the approaches we investigate in this paper. Section 4 outlines the details of the software implementation. Section 5 describes the test problems used and presents results. Section 6 discusses several other ways to improve the performance of our method. The final section concludes and suggests areas for future work.

2 Background

In this section, we present the notation to be used throughout the paper. First, we review ordinary and delay differential equations, then we discuss the method of least squares commonly used for estimating parameters. This is followed by a brief discussion of global and local optimizers, as well as techniques used to compute confidence intervals (CIs).

2.1 Ordinary Differential Equations (ODEs)

We consider a system of ODEs with state vector $\boldsymbol{x} \in \mathbb{R}^{n_x}$. For $t \in [t_0, t_f]$, the dynamics of $\boldsymbol{x}$ are given by,

$$\boldsymbol{x}' = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{p}); \;\; \boldsymbol{x}(t_0) = \boldsymbol{x_0}, \tag{1}$$

where $\boldsymbol{p} \in \mathbb{R}^{n_p}$ are the parameters of the model. We assume $\boldsymbol{x}$ is observed at n_t values of $t \in [t_0, t_f]$. These measurements are denoted by $\bar{X}_{ij}$, where $i = 1, 2, \ldots, n_t$ and $j = 1, 2, \ldots, n_x$ index time and the component of the state vector, respectively. That is, $\bar{X}_{ij}$ is a measurement of $\boldsymbol{x}_j(t_i)$. To quantify the error in the observations, we assume that the measurements are subject to a noise function, $\bar{X}_{ij} = g(\boldsymbol{x}_j(t_i))$. The noise function is commonly modelled by,

$$g(\boldsymbol{x}_j(t_i)) = \boldsymbol{x}_j(t_i) + \sigma_j \epsilon_{ij}, \tag{2}$$

where each ϵ_{ij} come from independent standard normal distributions and each σ_j is the standard deviation of the noise associated with $\boldsymbol{x}_j$. Alternatively, under a relative error model, we might use,

$$g(\boldsymbol{x}_j(t_i)) = \boldsymbol{x}_j(t_i)(1 + \sigma_j \epsilon_{ij}). \tag{3}$$

In (2) and (3), the noise level is determined by σ_j. Depending on the scale of the components of $\boldsymbol{x}$, different values for σ_j can correspond to drastically different levels of noise. To avoid this, we use the signal-to-noise-ratio (SNR) to quantify the level of noise in the observations. We define SNR as,

$$SNR = \frac{\sqrt{P}}{\sigma}, \tag{4}$$

with P being the power of the signal, and σ^2 being the variance of the noise. Note that a higher SNR indicates less noise in the data. Given noiseless data, we can use (4) to determine an appropriate σ to produce noisy data with a specified SNR. The power of the signal is taken to be the average of the squared values of the state variable, where the average is over the independent variable. This average could be computed per component of the state vector or over all components simultaneously. In our experiments, we allow a different σ_j for each $\boldsymbol{x}_j$, so we have,

$$\sigma_j = \frac{\sqrt{\sum_{i=1}^{n_t} \boldsymbol{x}_j(t_i)^2}}{SNR}. \tag{5}$$

For example, if we vary P and SNR, then we obtain the values in Table 1. As we see from the ratio $\frac{\sigma}{\sqrt{P}}$, $SNR = 10$ corresponds to 10% noise in the data, regardless of P. For a fixed P, in order to increase σ by a factor of K, we decrease SNR by a factor of K.

P	SNR	σ	$\sigma/\sqrt{P}$
1	1	1	1
1	10	0.1	0.1
1	100	0.01	0.01
4	1	2	1
4	10	0.2	0.1
4	100	0.02	0.01

Table 1: A simple example of how σ varies with P and SNR.

2.2 Delay Differential Equations (DDEs)

DDEs are used to model systems in which the dynamics not only depend on the current value of the state vector, but also its past values. They have been commonly applied to biological systems, where the dependence on the past is due to the time it takes biological processes to occur. For example, standard ODE population dynamics models are unable to reproduce the rich dynamics obtained using DDE population dynamics models [14]. In this investigation, we only look at a subset of general DDEs, known as retarded DDEs, where the delays only appear in the value of the state vector, not its derivative. A system of retarded DDEs is given by,

$$\boldsymbol{x}' = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{x}(\alpha_1(t, \boldsymbol{x}, \boldsymbol{p})), \ldots, \boldsymbol{x}(\alpha_\rho(t, \boldsymbol{x}, \boldsymbol{p})), \boldsymbol{p})), \boldsymbol{p}),$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x_0}\ ;\ \boldsymbol{x}(t) = \boldsymbol{\phi}(t, \boldsymbol{p}),\ \text{for } t < t_0,$$

where each $\alpha_k(t, \boldsymbol{x}, \boldsymbol{p})$ is a scalar function defining a delay, $\alpha_k(t, \boldsymbol{x}, \boldsymbol{p}) < t$, and $\boldsymbol{\phi}$ specifies the history for the system, which is referenced if the corresponding delay is less than t_0.

2.3 Numerical Solutions to ODEs and DDEs

Many reliable methods exist for solving ODEs, but here we will briefly mention a continuous Runge-Kutta (CRK) solver that is particularly well suited to our application. This solver is a standard RK solver, except that it additionally computes a local interpolant between time steps. This allows for high accuracy off-mesh values of $\boldsymbol{x}$, as well as its derivatives, to be obtained. A user specified tolerance, TOL, controls the defect of the computed solution, such that it is bounded by a small multiple of TOL.

We use the DDEM package to simulate both our ODE and DDE models, since when DDEM is asked to simulate a DDE, with all delays equal to zero, the simulation reduces to approximating the ODE by a reliable CRK solver.

2.4 Least Squares (LSQ)

Given the measurements, $\bar{X}_{ij}$, parameter estimation is performed by minimizing the weighted sum of squared errors,

$$\mathcal{L}(\boldsymbol{p}) = \sum_{i}^{n_t} \sum_{j}^{n_x} w_{ij} (\bar{X}_{ij} - \boldsymbol{x}_j(t_i, \boldsymbol{p}))^2, \tag{6}$$

where each w_{ij} is a weight factor. These weight factors are set to reflect the practitioner's confidence in the observations. If the error in the observations is known, the weights should be inversely proportional to the level of error, so as to reduce the importance of data values with large noise. We refer to (6) as the sum of squared errors (SSE).

We note that numerically performing this minimization requires repeated simulation of the model in order to compute (6). Algorithms for solving this least squares problem can be broadly categorized as being global or local.

2.4.1 Global Optimization Methods

Global optimization methods seek to find the global minimum of an objective function. The search space is usually defined by upper and lower bounds or by means and standard deviations on each parameter. Various heuristics have been suggested to direct such a search. In this paper, we focus on a cross entropy (CE) approach. CE iteratively draws parameter sets from a specified distribution, then uses the corresponding values of the objective function to select the best parameter sets, which are then used to update the probability distribution for subsequent iterations. For continuous optimization, the probability distribution is taken to be a multivariate-normal distribution. The CE approach is fully described in [25]. Application of this method, and several modifications to it, are presented in [27]. We will briefly describe the method and introduce the necessary notation in the next section. A comparison of several global optimizers is presented in [20], where the test problem was performing a least squares parameter estimation for a large system of ODEs.

2.4.2 Local Optimization Methods

Local optimization methods use derivative information to obtain fast convergence to a local minimum. Since they rely on local gradient information, they provide no guarantee that the found minimum will be the global minimum. These methods have been widely studied and implementations are broadly available. For LSQ, the gradient of the associated objective function, (6), is given by

$$\frac{\delta \mathcal{L}(\boldsymbol{p})}{\delta \boldsymbol{p}} = \sum_{i}^{n_t} \sum_{j}^{n_x} 2w_{ij}(\bar{X}_{ij} - \boldsymbol{x}_j(t_i, \boldsymbol{p})) \frac{\delta \boldsymbol{x}_j(t_i, \boldsymbol{p})}{\delta \boldsymbol{p}},$$

where $\frac{\delta \boldsymbol{x}_j(t_i, \boldsymbol{p})}{\delta \boldsymbol{p}}$ must be computed. Note that, although $\boldsymbol{x}_j(t_i, \boldsymbol{p})$ is approximated by a simulation, the values of $\frac{\delta \boldsymbol{x}_j(t_i, \boldsymbol{p})}{\delta \boldsymbol{p}}$ must be approximated as well. This can be done using finite differences or a more sophisticated method. DDEM simultaneously simulates the system and a set of variational equations to provide an approximation to this $n_t \times n_p$ matrix at each t_i. This approach is commonly used to obtain the sensitivities for ODEs [3], but was more recently extended to general DDEs in [29]. We now briefly mention two approaches to simulating the system.

2.4.3 Simple (Single) Shooting (SS)

The most obvious way to obtain the trajectories in (6) is to simulate the system from an initial time and stop the simulation after the last measurement is reached. However, this simple approach has been found to encounter two major issues when used in combination with a local optimization method. First, it often causes the optimizer to converge to local minima. Second, if the initial set of parameters is far from the optimal, the simulation might fail before reaching the last observation. To address these issues, the method of multiple shooting can be used.

2.4.4 Multiple Shooting (MS)

A more robust alternative to single shooting is multiple shooting. This method is widely used for the numerical solution of boundary value problems in ODEs (see, for example Krogh et al [13] and Boch et al [5]). It has also been suggested for parameter estimation of ODEs by [23]. We now present the general approach. The interval over which the system is simulated is divided into N_{MS} subintervals, where the first interval is $I_0 = [t_0, \tilde{t}_1]$, the r^{th} interval is $I_r = [\tilde{t}_r, \tilde{t}_{r+1}]$, and $I_{N_{MS}-1} = [\tilde{t}_{N_{MS}-1}, t_f]$. Additional parameters are added to specify the state vector at the beginning of each subinterval after I_0. This means we have $(N_{MS} - 1)n_x$ extra parameters to estimate. Finally, we enforce equality constraints at the boundary of each interval,

$$\boldsymbol{x}(\tilde{t}_r, \boldsymbol{p}; \tilde{t}_{r+1}) = \boldsymbol{x}(\tilde{t}_{r+1}) \ \text{, for } r = 0, \ldots, N_{MS} - 2, \tag{7}$$

where $\boldsymbol{x}(\tilde{t}_r, \boldsymbol{p}; \tilde{t}_{r+1})$ is the value of the state vector at the right end point of I_r, and $\boldsymbol{x}(\tilde{t}_{r+1})$ is the value of the state vector at the left end point, which are the additional parameters to be estimated.

The advantage of this approach is that it allows discontinuities in the trajectory to exist during the optimization, which can allow local minima to be avoided. Also, since it restarts the simulation at the start of each subinterval, it is less likely that the simulation will ever fail. The presence of the additional equality constraints ensures that the trajectory at convergence will be continuous. The downside of MS is that it is more computationally expensive than SS. We note that the simulations on each subinterval can be done in parallel. Multiple shooting was extended in [10] to a class of DDEs with constant delays.

2.5 Confidence Intervals (CIs)

The simplest way to generate confidence intervals is to use the estimated variance of each parameter. For weighted least squares, the variance of $\boldsymbol{p}_i$, $\sigma^2(\hat{\boldsymbol{p}}_i)$, can be approximated by,

$$\hat{\sigma}^2(\hat{\boldsymbol{p}}_i) = \frac{\mathcal{L}(\hat{\boldsymbol{p}})}{n_x n_t}(SWS^T)^{-1}{}_{ii},$$

where W is a diagonal matrix of the weights, w_{ij}, $\hat{\boldsymbol{p}}$ is the vector of estimated best-fit parameters, and S is the matrix of sensitivities. The dimension of matrix S is $n_x n_t$ by n_p, with entry $(in_x + j, k)$ given by $\frac{\delta \boldsymbol{x}_j(t_i, \boldsymbol{p})}{\delta \boldsymbol{p}_k}$. Using this variance, the $(1-\alpha)100\%$ CI is given by the standard formula,

$$\hat{\boldsymbol{p}}_i \pm z_\alpha \hat{\sigma}(\hat{\boldsymbol{p}}_i), \tag{8}$$

where z_α is the critical value of the standard normal distribution for the given value of α.

Alternatively, one can use the minimum value of (6) and the F-distribution to obtain a target value, which (6) should obtain when each parameter is perturbed by an appropriate amount. While this is more computational than the first approach, it has the advantage that it directly references the SSE, so the obtained CI is consistent with the fact that the SSE likely does not depend symmetrically on a given parameter.

For more robust CIs, it is necessary to perform a much more complex computational method, such as a bootstrap or jackknife. Given their high cost, we do not implement them. We used the first two approaches mentioned and obtained similar results for our test problems.

3 Methods

3.1 Cross Entropy (CE)

For a general discussion of CE optimization, we refer the reader to [25]. For the purposes of this paper, we present a minimalist description of the algorithm suggested in [27]. CE is initialized by specifying an initial mean, $\boldsymbol{\mu}^{(0)}$, and initial standard deviation, $\boldsymbol{s}^{(0)}$, for the parameters being estimated and by setting an iteration counter, r, equal to 1. CE proceeds iteratively as follows:

- Draw N samples from $\mathbb{N}(\boldsymbol{\mu}^{(r-1)}, \boldsymbol{s}^{(r-1)})$. These samples are denoted by $\boldsymbol{D}$.
- For each sample, compute the objective function. Sort the samples from smallest to largest objective function, denoting the objective function for the i^{th} smallest sample by $S(\boldsymbol{D}_i)$.
- From the CE parameter $\rho \in (0, 1)$, determine the number of elites, $e = \lceil \rho N \rceil$. The elites are the e samples with the smallest objective function values.

- Store $\gamma_r = S(\boldsymbol{D}_e)$.
- Compute the mean, $\boldsymbol{u}_r$, and the standard deviation, $\boldsymbol{s}_r$, of the elites.
- Update the parameter means using $\boldsymbol{\mu^{(r)}} = \alpha \boldsymbol{u}_r + (1-\alpha)\boldsymbol{\mu^{(r-1)}}$.
- Update the parameter standard deviations using $(\boldsymbol{s}^{(r)})^2 = \beta_r {\boldsymbol{s}_r}^2 + (1-\beta_r)(\boldsymbol{s}^{(r-1)})^2$, where $\beta_r = \frac{\beta_0}{r}$.
- If the stopping criterion is satisfied, terminate. Otherwise, increment r and return to the first step.

Above, α and β_0 are weighting parameters in $[0, 1]$, which determine how quickly the algorithm is to accept changes to the current values of $\boldsymbol{\mu}$ and $\boldsymbol{s}$. Based on results reported in [27], we set $\alpha = 0.8$. The parameters N and ρ specify the number of samples per iteration and the percentage of samples to be used as elites per iteration, respectively. The stopping criterion must be carefully chosen. In [27] the choice is to wait until, for some $r > 4$, the condition $\gamma_r \approx \gamma_{r-1} \approx \ldots \gamma_{r-4}$ is satisfied. Here, $a \approx b$ is taken to mean that $|a - b| < TOL$, where a TOL of 10^{-6} is used. We also note that since CE draws samples from a truncated normal distribution, it can naturally handle upper and lower bounds on parameters.

3.2 Modified Cross Entropy (MCE)

In [27], the MCE algorithm is inspired by the slow convergence of CE, as it is presented above. The principal of local evolution is used to shift non-elites towards the best-seen sample. Each shifted sample is simply a linear combination of the best seen and the original sample. The other key change to CE is that these shifted non-elites are also included in the calculation of the means and standard deviations in each iteration. In this work, we do not adopt MCE, but explore alternative modifications to the original CE algorithm presented above.

3.3 Our Modifications to CE

We adopt several minor modifications to CE. First, we adopt a weighted calculation of the means of the elites in each iteration, where the elites are weighted by the inverse of their objective function values. This helps reduce the impact of potentially poor elites early on. Second, we change how we update $\boldsymbol{s}$ on each iteration. We note that the choice of weight factor, $\beta_r = \frac{\beta_0}{r}$, results in $\boldsymbol{s}^{(r)} \approx \boldsymbol{s}^{(r+1)}$, as r increases. If we want CE to converge to a certain tolerance, this freezing of $\boldsymbol{s}^{(r)}$ is undesirable. To address this concern, we redefine β_r to be given by,

$$\beta_r = \beta_0 + \beta_f min(\frac{r-1}{n_p}, 1).$$

We choose $\beta_0 = 0.2$ and $\beta_f = 0.7$. This strategy initially favours the previous iteration's $\boldsymbol{s}$, but gradually switches to heavily weighting the standard deviations of the elites. Since we are no longer freezing $\boldsymbol{s}^{(r)}$, we have the potential to shrink $\boldsymbol{s}^{(r)}$ too rapidly. To address this, we change $\boldsymbol{s}_r$, the standard deviations of the elites on iteration r, to be

$${\boldsymbol{s}_r}^2 = \frac{1}{N_e} \sum_{k=1}^{N_e} (\boldsymbol{D}_k - \boldsymbol{\mu}^{(r-1)})^2,$$

where $\boldsymbol{D}_k$ is the k^{th} elite and N_e is the number of elites. This is motivated by the updates done in the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm [9]. By looking at the difference between the elites and the means from the previous iteration, this alternative s_r has the advantage that it can result in $\boldsymbol{s}^{(r+1)} > \boldsymbol{s}^{(r)}$. This will occur if the elites are strongly skewed between iterations.

3.3.1 Stopping Criterion

We choose a stopping criterion for CE similar to the one used in [27]. We require that,

$$\frac{|\bar{S}_r - \bar{S}_{r-1}|}{\bar{S}_r} < TOL,$$

where $\bar{S}_r$ is the average objective function value of the elites on iteration r. This must be satisfied for E consecutive iterations, where E is chosen to be 5. We try other stopping criteria and obtain similar results, since the quantities measuring convergence tend to behave similarly to $\bar{S}_r$.

3.3.2 Choice of N

Starting from a simple model with a single parameter, we will discuss several aspects of CE. These aspects of CE focus on the number of samples per iteration, N, and the percentage of samples to be taken as elites on each iteration, ρ. We describe a model with one parameter, p, which has an optimal value that is assumed to be in $[A, B]$. Consider an objective function with two local minima, as shown in Figure 1. We assume CE is initialized with $\mu = \frac{A+B}{2}$ and $\sigma = \frac{B-A}{4}$. The pdf for this distribution is shown in Figure 1.

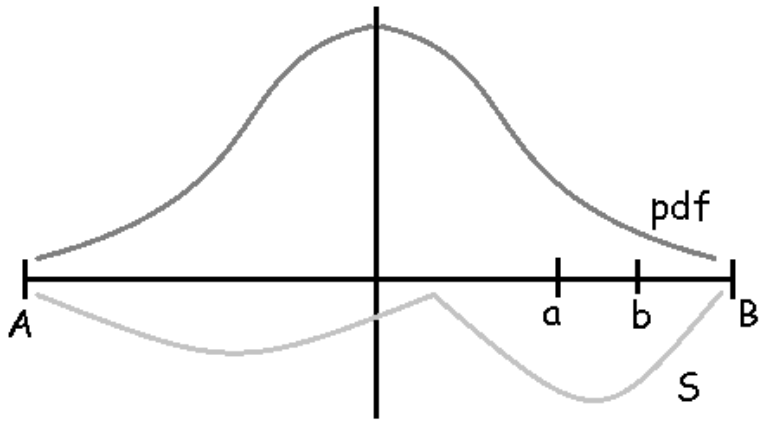

Figure 1: Objective function and probability distribution for a simple model with one parameter and two minima.

Now, we want to determine a crude estimate of how likely we are to sample a value of p, such that it will be close to the global minimum. To do this, we look at the probability that $p \in [a, b]$, which is simply $\hat{\rho} = cdf(b) - cdf(a)$, where cdf is the cumulative distribution function of the distribution the samples are being drawn from. Then, if we want N_e elites in proximity to the global minimum, we expect that $\hat{N} = \frac{N_e}{\hat{\rho}}$ samples must be taken. For example, if we use the values: $A = -1$, $B = 1$, $a = 0.5$, $b = 0.7$, then $\mu = 0$ and $\sigma = \frac{1}{2}$. This gives us that $\hat{\rho} = 0.08$. So, if we want $N_e = 10$ elites, we require about $\hat{N} = 128$ samples.

Given this example, we now discuss several observations we can make regarding the behaviour of CE. First, the values of a and b are usually not known, but must be approximated in order to calculate $\hat{\rho}$. Second, if we choose $\rho = \hat{\rho}$ and $N < \hat{N}$, it is not that CE will not work, but it becomes more likely that a subset of the elites will in fact be in proximity to a local minimum, which could slow the convergence of CE. Similarly, if we use this approach and choose $N = \hat{N}$ and $\rho > \hat{\rho}$, then it becomes more likely that some of the elites will not be in proximity to the global minimum.

Another observation is that it might be better to draw the parameters from a different distribution. For our example, if we had used a uniform distribution over the interval $[-1, 1]$,

then we would obtain $\hat{\rho} = 0.1$. So, if we want $N_e = 10$ elites, we require about $\hat{N} = 100$ samples. This suggests that if little is known about the parameter, it might be advantageous to draw parameters from a uniform distribution, rather than a normal distribution, for the first iteration of CE. Alternatively, what if we had more information and knew that the minimum occurred within $[0.2, 1]$. In this case, we would have $\mu = 0.6$ and $\sigma = \frac{1}{5}$. This gives us that $\hat{\rho} = 0.38$. So, if we want $N_e = 10$ elites, we require about $\hat{N} = 27$ samples. In this case, it is advantageous to use the normal distribution, since the uniform distribution would require $\hat{N} = 40$ samples. Note that the number of samples decreases significantly if we can reduce the search space.

Now, we consider what happens if our model has more parameters. In this case, our estimate for ρ is simply the product of the $\hat{\rho}$'s for the parameters in the model. For example, if we take the numbers we used for the initial one-parameter model, and assume that both parameters behave the same, then we have $\hat{\rho} = 0.006$. So, if we want $N_e = 10$ elites, we require about $\hat{N} = 1648$ samples. In a sequential computing environment, this quickly becomes an infeasible number of samples per iteration.

As in the case of one parameter, we can greatly reduce the required number of samples if we can reduce the search space. Ideally, we could independently estimate the two parameters, which would still require 128 samples per parameter, but would only need 256 samples to estimate both of them, instead of 1648 samples.

Lastly, we return to the simple one parameter model to note the usefulness of a 'hybrid' optimization procedure. In [1, 2], the use of a hybrid optimization procedure is used, where 'hybrid' refers to the use of both a global and a local optimizer during the optimization procedure. The idea is that the global optimizer should be used only until we are close enough to the global optimum that a local optimizer will converge. This hybrid approach combines the local optimizer's fast convergence and the global optimizer's more rigorous search to potentially significantly reduce the computational effort, without sacrificing the thoroughness of the search. For our simple example in Figure 1, it is clear that if we start a local optimizer between a and b, then it will converge to the global minimum. This means that we will probably be able to switch to a local optimizer after just one or two iterations of CE.

In this section, we have highlighted two main topics which we explore next. These are reducing the search space and using a hybrid optimizer.

3.4 Obtaining Improved Initial Estimates

Since the run time of a global optimization algorithm is expected to scale exponentially with the number of parameters, it is thus useful if reliable estimates for a subset of $\tilde{n}$ parameters can be used to effectively reduce the number of parameters being estimated from n_p down to $n_p - \tilde{n}$. We detail three possible approaches to do this. The first is applying expert knowledge about the problem. If it is known that a parameter will be roughly a certain value, it should be safe to fix its value and only refine its estimate after the other parameters are estimated for the first time. Without such expert knowledge, we either have to identify parameters for which the objective function is relatively insensitive, or come up with another way to get an estimate on a subset of the parameters. Identifying how sensitive the objective function is to each parameter is difficult, since sensitivity information is generally local, that is, the sensitivities are computed at a point in the parameter space. We are interested in more global sensitivities. This approach is not explored in this investigation.

We now consider how we have chosen to estimate a subset of the parameters. We describe two approaches for ODEs, but they apply to DDEs as well. The idea will be to formulate an alternate optimization problem, based on the given observations and the model.

3.4.1 A Fast Approach - Don't Simulate the Model

As we noted, minimizing (6) requires repeated simulation of the model. Since this is an expensive procedure, we first try to avoid it entirely. We can avoid the costly model simulations by solving an alternate optimization problem. In this case, we use the data, $\bar{X}_{ij}$, to approximate the dynamics of the state variables. This was previously suggested in [26], and more rigorously explored in [24]. We denote the numerically differentiated data by $\bar{X}'_{ij}$. Now, our parameter estimation is done by minimizing,

$$\mathcal{C}(\boldsymbol{p}) = \sum_{i}^{n_t} \sum_{j}^{n_x} w_{ij} (\bar{X}'_{ij} - \boldsymbol{x}_j{}'(t_i, \boldsymbol{p}))^2, \tag{9}$$

where the $\boldsymbol{x}_j{}'(t_i, \boldsymbol{p})$ are obtained directly from (1).

We immediately see several limitations and benefits to this approach. First, we require a sufficient number of observations to compute reasonable divided differences approximations of $\bar{X}'_{ij}$. The noise in the observations must also not be too large or else the numerical derivative will not be accurate. A major benefit in terms of computation is that the numerical derivatives computed using finite differences (FD) are significantly faster to compute than the simulation of an ODE or DDE. We also do not have to worry about what happens when a set of parameters would cause the simulation to fail. As with our original least squares problem, numerous techniques exist for performing the minimization of (9). The other major benefit is that for models with multiple components, this approach can work on a subset of the components. This means we can estimate the subset of parameters that appear in the equations defining the dynamics of a subset of components.

Note that while this approach cannot estimate initial conditions at all, we can estimate initial conditions directly from the observations. The simplest approach is to estimate the initial conditions as being equal to the observation closest to the initial time. We consider this heuristic in this investigation. For most of our test problems, this FD approach is used to obtain our improved initial guesses, as it yields good results and is less expensive than the approach we suggest next.

3.4.2 Data Driven Model Simulation

We introduce a second heuristic to obtain improved initial parameter estimates. Here, we return to simulating the system, but in a special way. Rather than simulate the dynamics of all state variables, we only simulate a single state variable, say $\boldsymbol{x}_k$. For the rest of the states, we will interpolate from the data, $\bar{X}_{ij}$. The modified dynamics for $\boldsymbol{x}_k$ are given by,

$$\tilde{x}'_k = \tilde{f}_k(t, \tilde{x}_k(t), \tilde{\boldsymbol{p}}_{\boldsymbol{k}}, \bar{X}_{ij}); \;\; \tilde{x}_k(t_0) = \boldsymbol{x}_{\boldsymbol{0}k}, \tag{10}$$

where $\tilde{f}_k$ is $\boldsymbol{f}_k$ from (1), except that it uses $\bar{X}_{ij}$ to interpolate the other state variables, and $\tilde{\boldsymbol{p}}_{\boldsymbol{k}}$ is the subset of $\boldsymbol{p}$, which appear in $\boldsymbol{f}_k$. Then to estimate $\tilde{\boldsymbol{p}}_{\boldsymbol{k}}$, we again use least squares. In order to estimate all the model parameters, this must be done for $k = 1, 2, \ldots, n_x$.

When this has been done for each $\boldsymbol{x}_k$, if the minimizations are all successful, then we should intuitively be very close to a solution to (6). This is because the parameters are now set, such that each $\tilde{x}_k$ closely fits the data, subject to the other state variables being interpolated from the data. Since this is true for each $\tilde{x}_k$, we expect that if we now simulate the full dynamics of the system, we should obtain a similarly good fit. Of course, this assumes that data is available for all state variables. Additionally, if the data is too noisy, the underlying simulator may be forced to take small time steps, due to the jagged shape of the linear interpolation of the data. To prevent this, one can smooth the data and possibly relax the tolerance of the simulator.

3.5 Hybrid Approach

As we mentioned earlier, a hybrid optimizer combines a global and a local optimizer to achieve both a high rate of convergence and run times much lower than those of purely global methods. In order to take advantage of this, it is necessary to determine when the global method has moved sufficiently close to the global minimum for the local optimizer to converge. Switching too early can result in a local minimum, while switching too late reduces the benefit of the local method's super-linear rate of convergence.

Determining when to switch can be done in several ways. One approach would be to monitor the values of the parameter means, standard deviations, or objective function values. When one of these values has converged, to some user-defined tolerance, we can move to the local phase. The local search is started from the value of $\boldsymbol{\mu}$ after the last iteration of CE. Another approach would be to run one iteration of the local solver at the end of each iteration of the global method and determine if its progress in that iteration suggests that it is converging to a minimum.

In either case, we are left with the possibility that we switch too soon and find a local minimum. If this is the case, and we can detect that the minimum is not globally optimal, it would be possible to add a penalty term to the objective function, which would avoid converging to the same local minimum. With the penalty term in place, we could return to the global phase and continue the search, on the modified objective function. In practice, it is difficult to create such a penalty function, so we do not currently implement this feature. For our switching criterion, we adopt the same criterion as we do for our stopping criterion in CE, but we use a less stringent tolerance and reduce E from 5 to 3.

4 Implementation Details

Our software is implemented in C++. Additional post-processing scripts are written in MATLAB to provide visualizations of the simulations. C++ is used rather than MATLAB for several reasons. First, the DDE solver used by DDEM is more reliable than the MATLAB solvers. Second, C++ can be optimized at compile time and performs significantly faster than the MATLAB interpreter. The ALGLIB library [4] is used for computing the matrix inverse in (8).

In order for a gradient-based local optimizer to be used, it is necessary to compute the derivative of our objective function, with respect to the model parameters. DDEM can do this either using finite differences or using analytical expressions for first partials of the specified ODE or DDE. If analytical expressions are used, the user has to provide $\frac{\delta \boldsymbol{f}}{\delta \boldsymbol{x}}$, $\frac{\delta \boldsymbol{f}}{\delta \boldsymbol{p}}$, and $\frac{\delta \boldsymbol{f}}{\delta t}$ to DDEM. The software implemented includes a symbolic differentiation module, which automatically computes these first partials for the user. This feature is especially desirable when working with complex models with a large number of parameters.

In future, a more complete symbolic differentiation program should replace the currently used Java implementation, but for most purposes this simple implementation should be all one needs. The implementation uses Dijkstra's shunting yard algorithm to convert the user provided infix expression, written with MATLAB syntax, into a postfix expression, which is in turn converted into an expression tree. The standard rules of differentiation are then applied to the expression tree to construct the expression trees representing the specified partial derivatives. Each expression tree can then be output as an infix expression, in standard C++ syntax.

In order to use this software, several steps are required. The first, and most involved step, is to load the model into the system.

4.1 Specifying the System

In order to load the model, the software requires the user to specify:

- the right hand side (RHS) function, $\boldsymbol{f}$.
- the delays (none if the system is an ODE)
- a set of default parameter values
- the initial and final times

It is anticipated that most users of this software would be most comfortable working with MATLAB syntax, so the specification of each of these pieces is in MATLAB syntax. The program for loading the models is written in Java and contains a simple interface for specifying the model. The syntax of the program calls are based on the ODE and DDE calling sequences used in MATLAB:

- ode(RHS,$[t_0\ t_f]$,$[y_0(t_0)\ y_1(t_0)\ \ldots\ y_n(t_0)]$,$[p_0\ p_1\ \ldots\ p_n]$)
- dde(RHS,$[\tau_0;\tau_1;\ldots;\tau_n]$, $[y_0(t_0)\ y_1(t_0)\ \ldots\ y_n(t_0)]$,$[t_0\ t_f]$,$[p_0\ p_1\ \ldots\ p_n]$)

Above, RHS is the name of a .m file containing the function defining the right-hand-side function for the system.

4.2 Specifying the Data and Parameter Estimation

After the system is loaded, one can perform parameter estimation. This involves specifying the name of the file containing the data points and indicating which method should be used to solve the least-squares optimization. The data file is a csv file, with each line containing the time, followed by the value of each component of the system. Currently, data for all components must be provided. Additionally, the times must be increasing.

4.3 Local Optimizer

The levmar package [17] is used as our local optimizer. It is a freely available implementation of the Levenberg-Marquardt algorithm [16, 18]. We experimented with the SQP optimizer, e04unc from the NAG C Library [21], used in DDEM's parameter estimation module, but found that it generally had more difficulty converging than levmar.

We also implemented a simple version of multiple shooting, using DDEM to simulate the system and levmar to solve the optimization. Lagrange multipliers are added to (6) to enforce the equality constraints in (7),

$$\mathcal{L}_{MS}(\boldsymbol{p}) = \mathcal{L}(\boldsymbol{p}) + \lambda \sum_{r=0}^{N_{MS}-2} \sum_{j=1}^{n_x} (x(\tilde{t}_r, \boldsymbol{p}; \tilde{t}_{i+1}) - x(\tilde{t}_{r+1}))^2, \tag{11}$$

where λ must be chosen. More sophisticated implementations of MS avoid having to arbitrarily choose λ through alternative methods, but for our purposes, a value of λ between 10 and 100 seemed to produce adequate results. Additionally, assuming the equality constraints are close to being satisfied at convergence, we should be able to use SS to remove any error in the estimation due to the crude enforcement of the equality constraints.

Initial values for each $\boldsymbol{x}(t_i)$ are chosen through linear interpolation of the measurements. We have not implemented it, but using a similar approach as [10], we could extend our implementation of MS to work for DDEs with constant delays. In this case, the history of each subinterval is interpolated from the measurements. Once the equality constraints are reasonably satisfied, it is then possible to obtain the history directly from previous subintervals. Of course, if this is done, then it would not be possible to simulate each subinterval in parallel.

4.4 Example of an ODE

Here, we show how our first test problem, which will be described in the next section, would be loaded into our software. First, we would write the matlab function B.m, shown in Figure 2. Then, we would enter the command: ode(B,[0 10],[p(4) p(5)],[1 1 1 1 0.3]). Note that we use the symbol 'p' when we want to refer to the parameter vector. So, we have specified that the right-hand-side function is B.m, the system is to be simulated from 0 to 10, the initial conditions are the fourth and fifth parameters, and the default parameter vector for the model is [1 1 1 1 0.3]. This command generates the necessary .cc files to specify the system in our software. These files are B.cc, B.h, B_0.cc, B_0.h, B_1.cc, and B_1.h. The B_0 files specify the modified system defined in (10), where all components except the first are being interpolated from the data. The B.cc file is shown in Figure 3. After this, the software must be recompiled for this model to be available for use.

```
function f = barnes(t,y,Z,p)
    f(1) = p(1)*y(1) - p(2)*y(1)*y(2);
    f(2) = p(2)*y(1)*y(2) - p(3)*y(2);
```

Figure 2: B.m

```
#include "B.h"
#include "math.h"
void getSystem_B(ddemSystem* system)
{
    system->create(2,5,1,0);
    system->setF(f_B);
    system->setHistory(historyFunc_B);
    system->setT0(0.0);
    system->setDelayArguments(1,0,delayArgumentsFunc_B, DDEM_FALSE);
}

void getSimulationA_B(ddemSimulationAttributes* simA)
{
    simA->tEnd = 10.0;
    simA->parameters[0] = 1.0;
    simA->parameters[1] = 1.0;
    simA->parameters[2] = 1.0;
    simA->parameters[3] = 1.0;
    simA->parameters[4] = 0.3;
}

void delayArgumentsFunc_B(double t, int m, double* y,
                          int l, double* p, int nu, int omega, double* alpha,
                          ddemUserProvidedJac* userProvidedJac, void* comm)
{
    alpha[0] = -1;
    if(userProvidedJac->needed)
    {
        userProvidedJac->tIndependent = DDEM_TRUE;
        userProvidedJac->yIndependent = DDEM_TRUE;
        userProvidedJac->pIndependent = DDEM_TRUE;
        userProvidedJac->zIndependent = DDEM_TRUE;
    }
}
```

```
void f_B(int state, double t, int m, double* y,
         int nu, int omega, double* z, int l, double* p,
         double* f, ddemUserProvidedJac* userProvidedJac, void* comm)
{
    f[0] = ((p[0] * y[0]) - ((p[1] * y[0]) * y[1]));
    f[1] = (((p[1] * y[0]) * y[1]) - (p[2] * y[1]));

    if(userProvidedJac->needed)
    {
        userProvidedJac->tIndependent = DDEM_TRUE;
        userProvidedJac->zIndependent = DDEM_TRUE;
        zero_Jac(m, l, nu, omega, userProvidedJac);

        userProvidedJac->yJac[0] = (p[0] - (p[1] * y[1]));
        userProvidedJac->yJac[1] = (-(p[1] * y[0]));
        userProvidedJac->yJac[2] = (p[1] * y[1]);
        userProvidedJac->yJac[3] = ((p[1] * y[0]) - p[2]);
        userProvidedJac->pJac[0] = y[0];
        userProvidedJac->pJac[1] = (-(y[0] * y[1]));
        userProvidedJac->pJac[6] = (y[0] * y[1]);
        userProvidedJac->pJac[7] = -y[1];
    }
}
void historyFunc_B(int historySegmentIndex, double t, int l, double* p,
     int m, double* h, ddemUserProvidedJac* userProvidedJac, void* comm)
{
    h[0] = p[3];
    h[1] = p[4];

    if(userProvidedJac->needed)
    {
        userProvidedJac->tIndependent = DDEM_TRUE;
        userProvidedJac->yIndependent = DDEM_TRUE;
        userProvidedJac->zIndependent = DDEM_TRUE;
        zero_Jac(m, l, 0, 0, userProvidedJac);
        userProvidedJac->pJac[3] = 1;
        userProvidedJac->pJac[9] = 1;
    }
}
```

Figure 3: The file B.cc generated by the command ode(B,[0 10],[p(4) p(5)],[1 1 1 1 0.3])

5 Experiments and Results

Experiments are performed to explore the issues we introduced earlier. First, we use our software to obtain parameter estimates on a series of six test problems, in order to demonstrate the basic functionality of our implementation. We then quantify the runtime of our method and compare it with simple and multiple shooting. Our final set of experiments look at how much time can potentially be saved when simulating the ODE or DDE, as this is the majority of the computational effort in a CE.

5.1 Additional Details of CE Heuristics

To start CE, we specify upper and lower bounds, $\boldsymbol{U}$ and $\boldsymbol{L}$, for each parameter. If an initial $\boldsymbol{\mu}$ is not additionally specified, the first iteration of CE draws from a uniform distribution between L and U. In this case, we do a special modification to the first update of $\boldsymbol{s}^{(1)}$ to ensure we don't shrink $\boldsymbol{s}$ too quickly. We use $\boldsymbol{s}^{(1)} = max\left(\boldsymbol{U} - \boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(1)} - \boldsymbol{L}\right)$. For initial conditions being treated as parameters, their initial means are taken to be the observations at t_0. We use $\alpha = 0.8$ as our weighting factor in updating the mean in each iteration.

For our initial search space, we take upper and lower bounds to be $\boldsymbol{L} = max(\boldsymbol{p} - i\boldsymbol{p}, \boldsymbol{0})$ and $\boldsymbol{U} = \boldsymbol{p} + i\boldsymbol{p}$, with $i = 3$. We also adopt a simple heuristic for choosing the number of samples per iteration of CE. Based on the discussion of section (3.3.2), we expect N must grow exponentially with n_p to ensure enough elites are close to the global minimum. In a sequential environment, this exponential scaling is not feasible, so we adopt the following heuristic for choosing N. We choose $N_e = 5$ and let N for the r^{th} iteration be given by,

$$N = max\left(\left(\sum_{k=1}^{n_p} min(1, \frac{2{\boldsymbol{s}_k}^{(r-1)}}{|\boldsymbol{U}_k - \boldsymbol{L}_k|})\right) N_{max}, n_p N_{min}\right), \tag{12}$$

where $N_{max} = 200$ and $N_{min} = 20$ indicate the maximum and minimum number of samples per parameter. This heuristic is chosen to allow the number of samples per iteration to decrease as $\boldsymbol{s}$ decreases. For large search spaces, or if more elites are used, N_{max} and N_{min} should be increased in order to improve the quality of the global search.

In the first iteration of CE, $\boldsymbol{s}_k = \frac{|U_k - L_k|}{2}$. So, for the first iteration, we have $N = N_{max} n_p$. The minimum number of samples ensures that we are always taking at least a reasonable number of samples per iteration.

5.2 Heuristic for Obtaining an Improved Initial Guess

To obtain our improved initial guess, we attempt to efficiently find a set of parameters that comes close to minimizing (9). This is either done using a local optimizer or a simple global optimizer, depending on the difficulty of the minimization. For this set of parameters, we crudely form an upper and lower bound based on its value of (6), by using (8). Half the width of this region is used as the initial $\boldsymbol{s}$ in CE.

5.3 Test Problems

For each test problem, we provide the model and the parameters used to generate simulated data. One to two sets of data are generated per problem, for the specified signal-to-noise ratio (SNR). Parameter estimates obtained for each data set are reported, as well as plots of the corresponding best-fit trajectories and the corresponding value of SSE scaled by $n_x n_t$, which we refer to as the mean squared error (MSE), for the best-fit trajectories. We perform a weighted least squares estimation, with the weights in (6) set to be the true ${\sigma_i}^2$ for each observation of the i^{th} component. With these weights, the value of the optimal MSE should be close to 1. We define the target MSE to be the MSE obtained with the parameters set to their true values

used to generate the noiseless data. The value of σ_i for each component is specified for each test problem.

In practice, if we do not know what the weights should be, we could first perform the optimization without weights and then use the estimated values of each σ_i to perform the weighted least squares estimation. Unless otherwise stated, we use 100 uniformly spaced data points in $[t_0, t_f]$ and a simulation tolerance of 10^{-5}. For each problem, we report CIs computed using (8).

5.3.1 Barnes Problem

The Barnes Problem is often used in the literature on parameter estimation for ODE models [27, 26]. It refers to a specific parameterization of the predator-prey model, given by,

$$y_1{}' = ay_1 - by_1y_2,$$

$$y_2{}' = by_1y_2 - cy_2,$$

where y_1 is the population of predators, y_2 is the population of prey. The parameters and initial conditions are specified to be $a = b = c = 1$, $y_1(0) = 1$, and $y_2(0) = 0.3$, for $t \in [0, 20]$. The true solution with these parameters is shown in Figure 4, along with the noisy data and the estimated best-fit trajectory. Parameter estimates are given in Table 2 and a summary of the true and estimated noise is presented in Table 3.

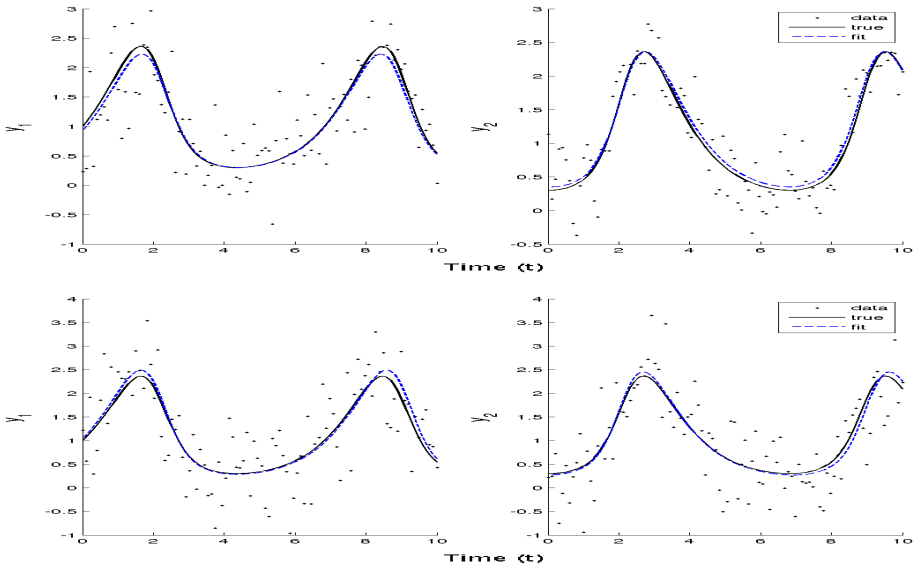

Figure 4: Barnes Problem: Noisy data, true curve, and fitted curve for two levels of noise - SNR $= \sqrt{10}$ (top) and SNR $= \sqrt{5}$ (bottom)

Parameter	True	Est	low	high
a	1	1.049	0.9836	1.114
b	1	0.9946	0.9258	1.063
c	1	0.9607	0.897	1.024
$y_1(0)$	1	0.9276	0.8031	1.052
$y_2(0)$	0.3	0.3512	0.2793	0.423
a	1	0.9775	0.8937	1.061
b	1	0.9876	0.8888	1.086
c	1	1.006	0.9115	1.1
$y_1(0)$	1	1.041	0.8602	1.221
$y_2(0)$	0.3	0.271	0.183	0.359

Table 2: Parameter Estimates with 95% Confidence Intervals for the Barnes Problem

SNR	target MSE	fit MSE	σ_1	σ_2
$\sqrt{10}$	1.103	1.080	0.438	0.409
$\sqrt{5}$	1.095	1.086	0.620	0.579

Table 3: Summary of actual noise (values of SNR and each σ_i), the MSE for the best-fit parameters (fit MSE), and the MSE for the true parameters (target MSE) in the Barnes Problem. Each σ_i is the standard deviation of the normally distributed noise for the i^{th} component of the model.

5.3.2 Fitz-Hugh Nagumo

The Fitz-Hugh Nagumo equations model potentials of squid neurons [6]. The model is given by,

$$V'(t) = c\left(V(t) - \frac{V(t)^3}{3} + R(t)\right),$$

$$R'(t) = \frac{1}{c}\left(V(t) - a + bR(t)\right),$$

where V is a voltage across the neuron and R is a response current, inversely related to V. The initial conditions and parameters are chosen to be $V(0) = -1$, $R(0) = 1$, $a = 0.25$, $b = 0.5$, and $c = 3.5$, for $t \in [0, 20]$. This parameterization of the model is considered a challenging test problem due to the observation that its objective function has many local minima [27]. The true solution with these parameters is shown in Figure 5, along with the noisy data and the estimated best-fit trajectory. Parameter estimates are given in Table 4 and a summary of the true and estimated noise is presented in Table 5.

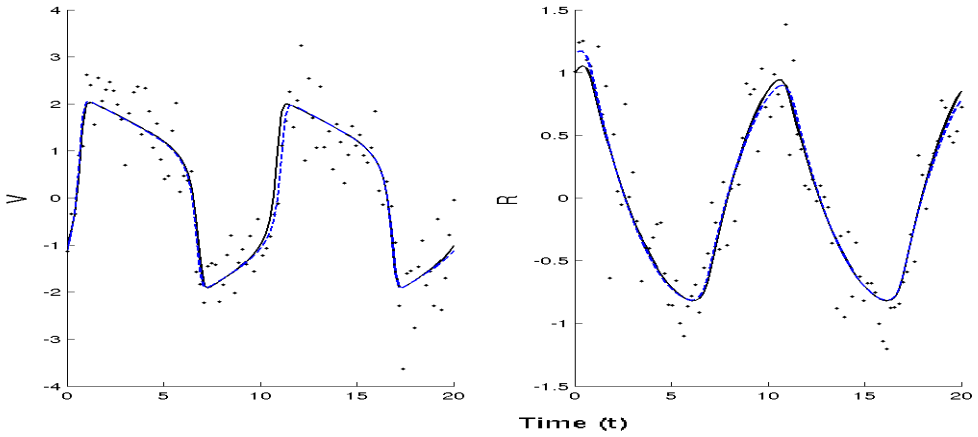

Figure 5: Noisy data, best-fit trajectory (dashed blue), and true trajectory (solid black) for the Fitz-Hugh model.

P	True	Est	Low	High
a	0.25	0.1664	0.1019	0.2309
b	0.5	0.6249	0.5053	0.7445
c	3.5	3.529	3.183	3.876
V(0)	-1	-1.143	-1.272	-1.015
R(0)	1	1.145	0.8906	1.4

Table 4: Parameter Estimates with 95% Confidence Intervals for the Fitz-Hugh model

SNR	target MSE	fit MSE	σ_1	σ_2
$\sqrt{5}$	0.870	0.833	0.652	0.274

Table 5: Summary of actual noise (values of SNR and each σ_i), the MSE for the best-fit parameters (fit MSE), and the MSE for the true parameters (target MSE) in the Fitz-Hugh model. Each σ_i is the standard deviation of the normally distributed noise for the i^{th} component of the model.

5.3.3 Kermack-McKendrick Model with Delays

This DDE system models the spread of disease within a population, using a Susceptible-Infected-Recovered (SIR) compartment model. It is a more complicated version of the standard ODE Kermack-McKendrick model [12]. The model is specified by the system of DDEs,

$$y_1{}'(t) = -y_1(t)y_2(t-\tau) + y_2(t-\rho),$$

$$y_2{}'(t) = y_1(t)y_2(t-\tau) - y_2(t),$$

$$y_3{}'(t) = y_2(t) - y_2(t-\rho),$$

where y_1 is the number of susceptible individuals, y_2 is the number of infected individuals, and y_3 is the number of recovered individuals. For convenience, we denote the initial conditions as $\boldsymbol{y}(0) = [a, b, c]$. As in [27], the parameters are chosen to be $a = 5$, $b = 0.1$, $c = 1$, $\tau = 1$, and $\rho = 10$, for $t \in [0, 55]$. The true solution with these parameters is shown in Figure 6, along with the noisy data and the estimated best-fit trajectory. Parameter estimates are given in Table 6 and a summary of the true and estimated noise is presented in Table 7.

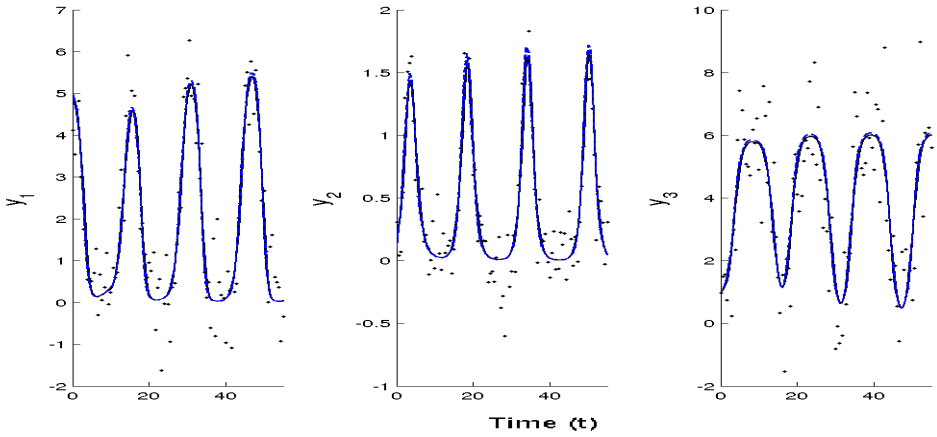

Figure 6: Kermack McKendrick model: Noisy data, true curve (solid black), and fitted curve (dashed blue).

P	True	Est	Low	High
τ	1	0.9328	0.6411	1.225
ρ	10	10.04	9.999	10.08
a	5	5.013	4.547	5.479
b	0.1	0.1003	0.06489	0.1357
c	1	1.047	0.982	1.111

Table 6: Parameter Estimates with 95% Confidence Intervals for the Kermack-McKendrick model.

SNR	target MSE	fit MSE	σ_1	σ_2	σ_3
$\sqrt{10}$	0.9217	0.8987	0.823	0.213	1.373

Table 7: Summary of actual noise (values of SNR and each σ_i), the MSE for the best-fit parameters (fit MSE), and the MSE for the true parameters (target MSE) in the Kermack-McKendrick model. Each σ_i is the standard deviation of the normally distributed noise for the i^{th} component of the model.

5.3.4 Hutchinson's Model

This models the population dynamics of a population, by the DDE,

$$x'(t) = rx(t)\left(1 - \frac{x(t-\tau)}{K}\right),$$

where r is the growth rate, K is the carrying capacity, x(t) is the population at time t, and τ is the time lag. This model was introduced in [11] and is commonly studied in ecology. We use parameters similar to those used in [28]. The values are $r = 1$, $\tau = 1.9$, $K = 2000$, and $x(0) = 1000$, for $t \in [0, 20]$. The true solution with these parameters is shown in Figure 7, along with the noisy data and the estimated best-fit trajectory. Parameter estimates are given in Table 8 and a summary of the true and estimated noise is presented in Table 9.

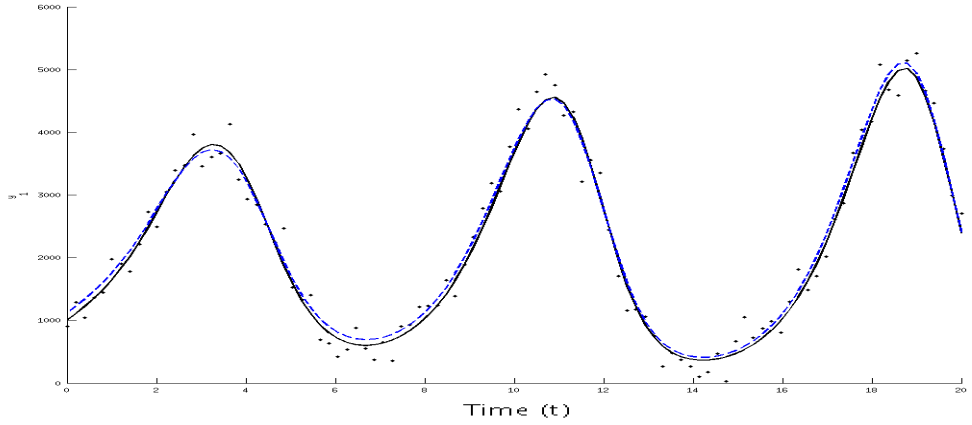

Figure 7: Noisy data, best-fit trajectory (dashed blue), and true trajectory (solid black) for Hutchinson's model

P	True	Est	Low	High
τ	1.9	1.915	1.899	1.931
r	1	0.9953	0.9838	1.007
K	2000	2052	2004	2101
$x(0)$	1000	1117	1037	1197

Table 8: Parameter Estimates with 95% Confidence Intervals for Hutchinson's model

SNR	target MSE	fit MSE	σ
10	0.8759	0.8880	262.6671

Table 9: Summary of actual noise (values of SNR and σ), the MSE for the best-fit parameters (fit MSE), and the MSE for the true parameters (target MSE) in Hutchinson's model. σ is the standard deviation of the normally distributed noise on x

5.3.5 Goodwin Model

This system of ODEs models a biological oscillator [8], which has been investigated in applications including enzyme kinetics and circadian clocks [7]. The delayed negative feedback loop model is given by,

$$x'(t) = \frac{a}{A + z(t)^\sigma} - bx(t),$$
$$y'(t) = \alpha x(t) - \beta y(t),$$
$$z'(t) = \gamma y(t) - \delta z(t),$$

where x is the product being synthesized, y is an intermediate product, and z exerts a negative feedback on the synthesis of x. Initial parameters and parameters are chosen to be $a = 3.4884$, $A = 2.15$, $b = 0.0969$, $\alpha = 0.0969$, $\beta = 0.0581$, $\gamma = 0.0969$, $\sigma = 10$, $\delta = 0.0775$, $x(0) = 0.3617$, $y(0) = 0.9137$, and $z(0) = 1.3934$, for $t \in [0, 20]$. These parameters are chosen to exhibit oscillatory behaviour [1]. The true solution with these parameters is shown in Figure 8, along with the noisy data and the estimated best-fit trajectory. Parameter estimates are given in Table 10 and a summary of the true and estimated noise is presented in Table 11.

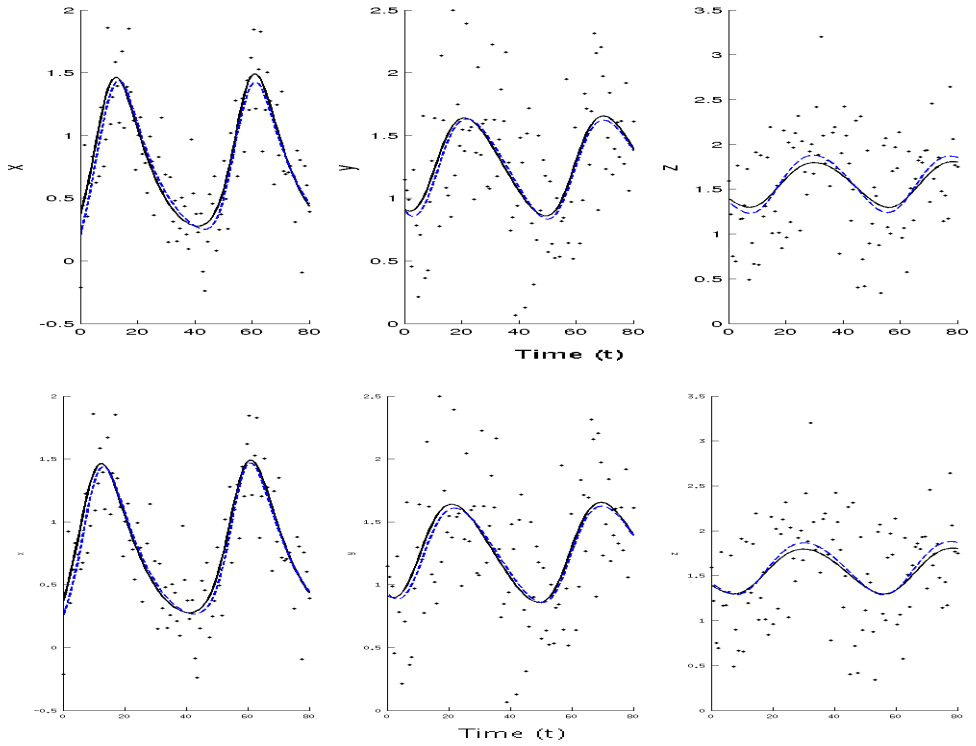

Figure 8: Noisy data, best-fit trajectory (dashed blue), and true trajectory (solid black) for the Goodwin model - all parameters varying (Top), a and A fixed (Bottom)

P	True	Est	Low	High	Est	Low	High
a	3.4884	18.02	-3.976e7	3.976e7	3.4884	-	-
A	2.15	71.91	-2.264e8	2.264e8	2.15	-	-
b	0.0969	0.07356	0.03848	0.1086	0.08367	0.0543033	0.113037
α	0.0969	0.1026	0.07513	0.1301	0.0965	0.0683086	0.124691
β	0.0581	0.06053	0.03496	0.0861	0.05664	0.0392324	0.0740476
γ	0.0969	0.139	0.06203	0.216	0.1276	0.0744458	0.180754
σ	10.0	15.21	-8.108e5	8.108e5	10.42	9.97129	10.8687
δ	0.0775	0.1093	0.05568	0.163	0.09915	0.0566658	0.141634
x(0)	0.3617	0.1905	-0.3775	0.7584	0.2472	-0.0933632	0.587763
y(0)	0.9137	0.9085	0.7433	1.074	0.9361	0.75098	1.12122
z(0)	1.3934	1.357	1.14	1.573	1.42	1.26115	1.57885

Table 10: Parameter Estimates with 95% Confidence Intervals for the Goodwin model. The left set of estimates is for all parameters and the right set of estimates is with a and A fixed at their true values.

SNR	target MSE	fit MSE	fit MSE (a,A fixed)	σ_1	σ_2	σ_3
$\sqrt{10}$	1.019	0.994	0.9976	0.292	0.416	0.492

Table 11: Summary of actual noise (values of SNR and each σ_i), the MSE for the best-fit parameters (fit MSE), and the MSE for the true parameters (target MSE) in the Goodwin model. Each σ_i is the standard deviation of the normally distributed noise for the i^{th} component of the model.

We note that the confidence intervals are extremely large for a, A, and σ. This indicates an

issue with identifiability, which is not surprising when we see that all three of these parameters appear in the same term. As a result, the quality of the other parameter estimates is also affected, but they are still reasonably close to their true values. We repeated the estimation with a and A fixed to their true values and report the improved estimates.

5.3.6 Calcium Ion model

This system of ODEs describes the oscillations of Ca^{2+} in the cytoplasm of eukaryotic cells, which play a role in cellular information processing. For a complete description of this model, see [15] where this model was first proposed. The model is given by,

$$G^*{}_\alpha{}' = k_1 + k_2 G^*{}_\alpha - k_3 PLC^* \frac{G^*{}_\alpha}{G^*{}_\alpha + Km_1} - k_4 Ca_{cyt} \frac{G^*{}_\alpha}{G^*{}_\alpha + Km_2},$$

$$PLC^{*\prime} = k_5 G^*{}_\alpha - k_6 \frac{PLC^*}{PLC^* + Km_3},$$

$$Ca_{cyt}{}' = k_7 PLC^* Ca_{cyt} \frac{Ca_{er}}{Ca_{er} + Km_4} + k_8 PLC^* + k_9 G^*{}_\alpha - k_{10} \frac{Ca_{cyt}}{Ca_{cyt} + Km_5} - k_{11} \frac{Ca_{cyt}}{Ca_{cyt} + Km_6},$$

$$Ca_{er}{}' = -k_7 PLC^* Ca_{cyt} \frac{Ca_{er}}{Ca_{er} + Km_4} + k_{11} \frac{Ca_{cyt}}{Ca_{cyt} + Km_6},$$

where the state variables are concentrations of four compounds, which interact in the calcium-signaling pathway. Parameters are chosen to be $k_1 = 0.09$, $k_2 = 2$, $k_3 = 1.27$, $k_4 = 3.73$, $k_5 = 1.27$, $k_6 = 32.24$, $k_7 = 2$, $k_8 = 0.05$, $k_9 = 13.58$, $k_{10} = 153$, $k_{11} = 4.85$,$Km_1 = 0.19$,$Km_2 = 0.73$,$Km_3 = 29.09$,$Km_4 = 2.67$,$Km_5 = 0.16$,$Km_6 = 0.05$. Initial conditions are treated as known, and given by $G^*{}_\alpha(0) = 0.12$, $PLC^*(0) = 0.31$, $Ca_{cyt}(0) = 0.0058$, and $Ca_{er}(0) = 4.3$. The model is simulated for $t \in [0, 20]$. For this specific parameterization, the solution exhibits a limit cycle [23]. The true solution with these parameters is shown in Figure 9, along with the noisy data and the estimated best-fit trajectory. Parameter estimates are given in Table 12 and a summary of the true and estimated noise is presented in Table 13.

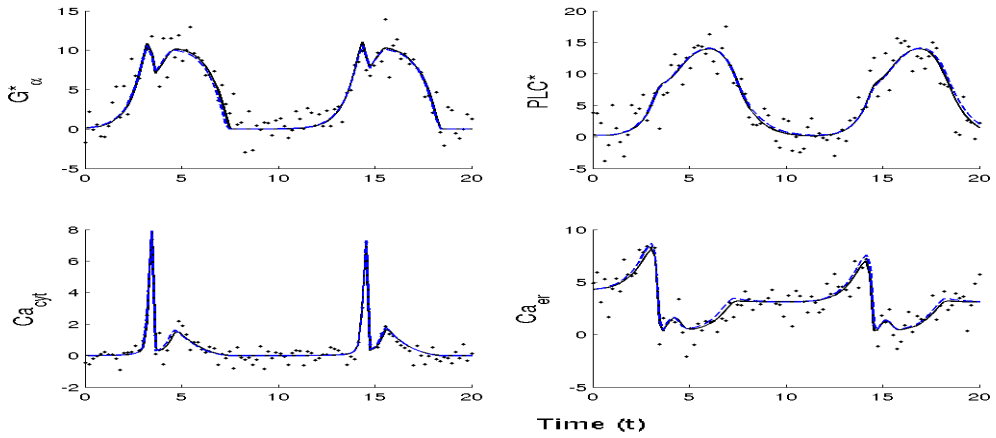

Figure 9: Noisy data, best-fit trajectory (dashed blue), and true trajectory (solid black) for the Calcium Ion model.

P	True	Est	Low	High
k_1	0.09	0.04403	-0.003735	0.09179
k_2	2	2.136	1.926	2.346
k_3	1.27	1.402	1.369	1.435
k_4	3.73	4.024	3.125	4.923
k_5	1.27	1.341	1.134	1.548
k_6	32.24	35.64	27.38	43.9
k_7	2	1.717	1.511	1.923
k_8	0.05	-0.1571	-0.9802	0.666
k_9	13.58	14.38	8.481	20.27
k_{10}	153	161.2	90.59	231.9
k_{11}	4.85	4.549	4.289	4.809

Table 12: Parameter Estimates with 95% Confidence Intervals for the Calcium Ion model

SNR	target MSE	fit MSE	σ_1	σ_2	σ_3	σ_4
$\sqrt{10}$	0.9038	0.8905	1.8318	2.5893	0.3935	1.1544

Table 13: Summary of actual noise (values of SNR and each σ_i), the MSE for the best-fit parameters (fit MSE), and the MSE for the true parameters (target MSE) in the Calcium Ion model. Each σ_i is the standard deviation of the normally distributed noise for the i^{th} component of the model.

We note that the confidence intervals on k_3, k_7, and k_{11} are probably tighter than they should be, as we would expect the true parameter to fall within this range. It is well known that confidence intervals computed in this way can be overly optimistic, as they assume a linear approximation to the gradient with respect to the parameters.

As we can see, our software is able to effectively estimate the parameters for our six test problems, which verifies the basic functionality of our implementation. We now look at quantifying the runtime of our 3-stage procedure.

5.4 Number of Simulations Per Estimation

To quantify the runtime, we use one model simulation as our unit of time, rather than reporting machine specific times. We will call this unit a SIM. In SIMs, we assume one iteration of CE takes N SIMs, where N is the number of samples per iteration. For our local Levenberg-Marquardt optimizer, we directly count the number of calls to our simulation routine and assume that the sensitivities can be computed in n_p SIMs. For MS, we again count all calls to our simulation routine and assume the sensitivities take $(n_p + n_x)$ SIMs, where the additional n_x SIMs are for obtaining the gradients for the additional parameters specifying the initial conditions on each subinterval. Our initial estimation procedure using FD is not included in the times we report, as it is found to be negligible. The times we report are all lower bounds of the actual time, since we neglect the additional computational costs incurred within the optimization routines. For example, the cost of the linear algebra performed in the local optimizer is proportional to the cube of the number of parameters being estimated. For MS, the number of parameters is $n_p + n_x(n_{MS} - 1)$, so while this cost is ignored in our analysis, it will become quite large as N_{MS} increases.

For our first four test problems, we report the time of three versions of our code. The results are given in Tables 14-17. Our three runs consist of only running CE (global), running CE and switching to levmar (hybrid), and our full 3-stage procedure. When CE is used on its own, its tolerance is set to TOL= ϵ, as listed in the tables. For the hybrid, we use a switching tolerance

$10^2\epsilon$. The simulation tolerance is set at $10^{-2}\epsilon$. The stopping criterion of the local solver is when the gradient or change in the parameters is less than 10^{-6}.

For these experiments, we vary SNR and TOL. We use an SNR of $\sqrt{20}$, which is about 20% noise, and an SNR of $10\sqrt{20}$, which is about 2% noise. TOLs of 10^{-2} and 10^{-3} are used. For each pair of SNR and TOL, we perform the estimation procedure at least 10 times and report the average time spent in the local and global optimizers. Additionally, we indicate how many of the runs converged to the global minimum. We consider an estimation to have found the global minimum if the MSE of the fit is less than the target MSE for the true parameters used to generate the data.

When running CE, it is often the case that it terminates with a best-fit MSE slightly above the target MSE. In this case, as long as the parameters are close enough that it is clear CE would eventually find the global minimum, we count it as converging to the global minimum. This means that when we say that CE is converging for two different tolerances, it is the case that the estimate obtained using the more stringent tolerance is more accurate.

In our experiments, we also compare the performance of our method against a multi-start shooting approach, where a local optimizer is started from several sets of parameters uniformly drawn from the specified parameter ranges. The following is used as our local optimizer for these multi-start experiments. First, we try using SS. If this does not converge to the global minimum, we try MS with $N_{MS} = 5$, then $N_{MS} = 10$, and finally $N_{MS} = 20$. If the local optimization still has not found the global minimum with $N_{MS} = 20$, then we give up.

5.4.1 Timing Results for Small ODE problems

From Tables 14 and 15, we see that the Barnes and Fitz-Hugh problems are relatively simple problems for the initial range on the parameters that we used, with our local optimizer converging most of the time from a random initial guess. All three of our global variants converged on every run, but we see significant reductions in time when we add each stage of our 3-stage procedure. We do note that when the hybrid optimizer is used, we see a significant reduction in the time spent in the local optimizer, since we are close enough to the solution that just a few iterations is usually enough to converge.

SNR	TOL	Local Time	Global Time	Total Time	Converged
$\sqrt{20}$	10^{-3}	0	5391	5391	10/10
$\sqrt{20}$	10^{-3}	19	4176	4194	10/10
$\sqrt{20}$	10^{-3}	19	1057	1076	10/10
$\sqrt{20}$	10^{-2}	0	4894	4894	10/10
$\sqrt{20}$	10^{-2}	25	2902	2927	10/10
$\sqrt{20}$	10^{-2}	18	578	596	10/10
$10\sqrt{20}$	10^{-3}	0	6476	6476	10/10
$10\sqrt{20}$	10^{-3}	18	4985	4997	10/10
$10\sqrt{20}$	10^{-3}	18	770	782	10/10
$10\sqrt{20}$	10^{-2}	0	5884	5884	10/10
$10\sqrt{20}$	10^{-2}	53	2880	2933	10/10
$10\sqrt{20}$	10^{-2}	18	400	418	10/10
$\sqrt{20}$	-	186	0	186	39/40
$10\sqrt{20}$	-	143	0	143	20/20

Table 14: Timing Results for Barnes Problem. For each value of SNR and TOL, the first row is for CE, the second row is for the hybrid, and the last row is for the full 3-stage procedure. The last two rows are for shooting from a random initial point.

We see that the full procedure takes only about 20-50% of the time taken by the hybrid.

This is largely due to the fact that our initial estimation allows us to start CE with a fairly accurate μ and a smaller s on each parameter, which means that we take significantly fewer samples per iteration of CE, as given by (12). For these two problems, the initial estimates become very accurate as the noise is reduced, so it is actually possible to skip the global phase entirely and directly use the local optimizer.

SNR	TOL	Local Time	Global Time	Total Time	Converged
$\sqrt{20}$	10^{-3}	0	5420	5420	10/10
$\sqrt{20}$	10^{-3}	24	4165	4189	10/10
$\sqrt{20}$	10^{-3}	32	1871	1903	10/10
$\sqrt{20}$	10^{-2}	0	4934	4934	10/10
$\sqrt{20}$	10^{-2}	25	3018	3043	10/10
$\sqrt{20}$	10^{-2}	40	1040	1080	10/10
$10\sqrt{20}$	10^{-3}	0	6152	6152	10/10
$10\sqrt{20}$	10^{-3}	22	5016	5028	10/10
$10\sqrt{20}$	10^{-3}	24	1733	1745	10/10
$10\sqrt{20}$	10^{-2}	0	5863	5863	10/10
$10\sqrt{20}$	10^{-2}	34	3004	3038	10/10
$10\sqrt{20}$	10^{-2}	34	574	608	10/10
$\sqrt{20}$	-	315	0	315	14/20
$10\sqrt{20}$	-	247	0	247	18/20

Table 15: Timing Results for Fitz-Hugh Problem. For each value of SNR and TOL, the first row is for CE, the second row is for the hybrid, and the last row is for the full 3-stage procedure. The last two rows are for shooting from a random initial point.

5.4.2 Timing Results for DDE problems

For our two DDE test problems, we do not have MS available, so the multi-start local optimizer is less effective than it is for our first two ODE problems. We have convergence from only about 20% of the runs. In terms of timing, we see the same kind of trend we have for the ODE problems, with both the hybrid and full methods showing significant reductions in run time. We note that for the Hutchinson problem, the hybrid method has some difficulty converging. This is most visible when the level of noise is increased and TOL is more relaxed. The full method avoids this issue, since its improved initial guesses help it to be close enough to the optimal parameters before switching to the local optimizer.

For the Kermack-McKendrick model, we do the initial estimation slightly differently, than we do for the Barnes and Fitz-Hugh problems. For the previous two problems, optimization of (9) is sufficiently smooth that the optimization is insensitive to the initial point, so we can start a local optimizer from an arbitrary point and be confident it will converge. This is not the case for the KM problem, where the only parameters we can estimate by our crude FD approach are the lags and we find that several local minima exist, usually corresponding to lags that are multiples of the true lags. Since there are only two lag parameters, we simply try 200 uniformly random pairs of lags and choose the pair with the lowest value of (9). We do the same for the other DDE and our last two ODEs, but we linearly scale the number of parameter sets we try by the number of parameters we are estimating. For our larger ODEs, this is a little bit expensive, but it is still not expensive enough to significantly contribute to the total run time.

SNR	TOL	Local Time	Global Time	Total Time	Converged
$\sqrt{20}$	10^{-3}	0	7066	7066	10/10
$\sqrt{20}$	10^{-3}	31	4434	4465	10/10
$\sqrt{20}$	10^{-3}	34	704	738	10/10
$\sqrt{20}$	10^{-2}	0	5700	5700	10/10
$\sqrt{20}$	10^{-2}	48	3058	3106	10/10
$\sqrt{20}$	10^{-2}	34	531	565	10/10
$10\sqrt{20}$	10^{-3}	0	10790	10790	10/10
$10\sqrt{20}$	10^{-3}	26	5761	5787	10/10
$10\sqrt{20}$	10^{-3}	21	850	871	10/10
$10\sqrt{20}$	10^{-2}	0	8935	8935	10/10
$10\sqrt{20}$	10^{-2}	35	3254	3289	10/10
$10\sqrt{20}$	10^{-2}	21	400	421	10/10
$\sqrt{20}$	-	112	0	112	16/50
$10\sqrt{20}$	-	135	0	135	11/50

Table 16: Timing Results for Kermack-McKendrick Problem. For each value of SNR and TOL, the first row is for CE, the second row is for the hybrid, and the last row is for the full 3-stage procedure. The last two rows are for shooting from a random initial point.

SNR	TOL	Local Time	Global Time	Total Time	Converged
$\sqrt{20}$	10^{-3}	0	4198	4198	8/10
$\sqrt{20}$	10^{-3}	85	2530	2616	5/10
$\sqrt{20}$	10^{-3}	62	564	626	10/10
$\sqrt{20}$	10^{-2}	0	3638	3638	10/10
$\sqrt{20}$	10^{-2}	127	1981	2108	4/10
$\sqrt{20}$	10^{-2}	61	329	390	10/10
$10\sqrt{20}$	10^{-3}	0	4148	4148	10/10
$10\sqrt{20}$	10^{-3}	60	3202	3262	10/10
$10\sqrt{20}$	10^{-3}	54	872	926	10/10
$10\sqrt{20}$	10^{-2}	0	3810	3810	10/10
$10\sqrt{20}$	10^{-2}	63	1952	2015	8/10
$10\sqrt{20}$	10^{-2}	55	320	375	10/10
$\sqrt{20}$	-	91	0	91	6/50
$10\sqrt{20}$	-	139	0	139	8/50

Table 17: Timing Results for Hutchinson's model. For each value of SNR and TOL, the first row is for CE, the second row is for the hybrid, and the last row is for the full 3-stage procedure. The last two rows are for shooting from a random initial point.

5.4.3 Timing Results for Larger ODE models

For our first four test problems, we only switched from CE to the local solver once and then terminated. For the Goodwin model and the Calcium Ion model, we run our 3-stage procedure with the feature that it will return to CE if the local solver fails to find the global minimum, since we know what the global minimum is for our problems. Each time we return to CE, we reduce the tolerance by a factor of 10. If the tolerance falls below ϵ, we stop and return the local minimum. As before, our initial switching tolerance is $10^2\epsilon$. The results are given in Tables 18-19.

SNR	TOL	Local Time	Global Time	Total Time	Converged
$\sqrt{20}$	10^{-2}	313	11099	11412	11/20
$\sqrt{20}$	10^{-2}	195	3507	3702	20/20
$10\sqrt{20}$	10^{-2}	252	10051	10303	20/20
$10\sqrt{20}$	10^{-2}	81	1511	1592	20/20
$\sqrt{20}$	-	651	0	651	20/20
$10\sqrt{20}$	-	545	0	545	20/20

Table 18: Timing Results for Goodwin Problem. For each value of SNR and TOL, the first row is for the hybrid, and the second row is for the full 3-stage procedure. The last two rows are for our multi-start local solver.

Due to the identifiability issues with the 3 parameters in the Goodwin problem, we fix the values of a and A to their true values for these experiments. We note that our 3-stage procedure is able to get close enough to the solution that usually only one run of the local optimizer is required to converge to the global minimum. The hybrid often is not close enough after the first switch and must return to CE in order to get close enough to converge. In the case of more noise, the hybrid often ends up converging to a local minimum. It turns out that MS is quite effective for this problem, achieving global convergence on every run and taking much less time than our 3-stage procedure, although our method does significantly reduce the time spent in the local optimizer.

SNR	TOL	Local Time	Global Time	Total Time	Converged	Time When Converged
$100\sqrt{20}$	10^{-2}	194	12707	12900	9/10	11578
$100\sqrt{20}$	10^{-2}	216	12013	12229	9/10	11489
$10\sqrt{20}$	10^{-2}	204	17488	17693	7/10	16902
$10\sqrt{20}$	10^{-2}	242	14082	14323	8/10	14422
$100\sqrt{20}$	10^{-2}	196	9302	9497	8/10	4751
$100\sqrt{20}$	-	540	0	540	4/20	546
$10\sqrt{20}$	-	487	0	487	1/20	899

Table 19: Timing Results for Calcium Ion Problem. For each value of SNR and TOL, the first row is for the hybrid, and the second row is for the full 3-stage procedure. The third last row is the full method with the data-driven approach used to obtain improved guesses for a subset of the parameters. The last two rows are for our multi-start local solver, but only using SS and MS with 5 subintervals.

For the Calcium Ion model, our local optimizers had difficulty converging when ran under the same settings as the other test problems, so we increased the number of observations up to 200, used higher values of SNR, and reduced the width of the search space to be $(\mathbf{0}, 2\boldsymbol{p})$. We note that our initial estimation procedure does not perform as well on this problem, as there are more parameters and the trajectories are more complex. This poor performance is because there are many different parameter sets which give similar values of (9), most of which are not actually near the optimal set to minimize (6). Due to the limited number of samples being taken and the use of SS as our local optimizer, both the hybrid and the 3-stage procedure end up at local minima on several runs.

We also try using our data-driven approach to obtain a suitable starting guess for a subset of the parameters, instead of our FD approach. The results are given in the third last row of Table 19. For this problem, it turns out that the four parameters in the second and fourth components of the state vector can be estimated very well using the data-driven approach and the local optimizer. We use the estimated minimum value of the least squares objective function

for the data-driven modified system and (8) to crudely form an upper and lower bound on each of these four parameters. Half the width of this region is used as the initial $\boldsymbol{s}$ in CE. For the other seven parameters, we do not get a distinct solution, so we start CE with these parameters being drawn from a uniform distribution, as we do with the hybrid. We see that this approach is competitive with the hybrid and 3-stage procedure using FD to obtain an improved initial guess. We do note that when we only consider the average time taken if the global minimum is found, the data-driven approach takes much less time. This is because we are able to get accurate initial guesses on 4/11 parameters, so we are more likely to converge to the global optimum when we switch from CE to the local optimizer for the first time.

5.5 Number of Steps Per Simulation

For each test problem, we report the number of steps the simulation takes to run, for the true set of parameters. This is reported for simulation tolerances ranging from 10^{-6} to 10^{-3}. This is tabulated in Table 20. These numbers suggest that it is often the case that the runtime of the optimization can be significantly reduced if we are able to justify a higher tolerance. Additionally, we note the large number of steps required to simulate the Calcium Ion model, even when the tolerance is relaxed. This behaviour indicates that the problem is likely stiff, which it turns out to be. Using the stiff ODE solver in MATLAB, we verified that the number of steps should be around 188 when TOL $= 10^{-6}$.

TOL	Barnes	Kermack-McKendrick	Fitz-Hugh	Goodwin	Calcium Ion	Hutchinson
10^{-6}	50	139	195	45	3727	120
10^{-5}	34	105	137	42	3233	85
10^{-4}	25	79	101	42	2945	63
10^{-3}	17	67	75	42	2792	45

Table 20: Steps Per Simulation at True Parameters for each test problem

5.6 Early Termination

Since CE is only concerned with whether a sample is an elite or not, we add a simple check inside the solver, which terminates the simulation if the objective function is over a threshold. In practice, this threshold would be the boundary between the elites and non-elites from the previous iteration. Alternatively, the threshold could be user-defined and we would modify CE to continue drawing samples until at least enough simulations had run to completion that we had the specified number of elites. Note that in order to do this, we modify the solver so that it is cumulatively calculating the objective function each time that it passes an observation, as it steps through the simulation.

To investigate the potential of using a threshold to terminate simulations early, we simply draw parameters from a uniform distribution, centered at the true value of each parameter. We then perform a series of model simulations, where we vary the width of our uniform distribution. Our uniform distribution is over the range $(\boldsymbol{p} - \frac{\boldsymbol{p}}{i}, \boldsymbol{p} + \frac{\boldsymbol{p}}{i})$, where i determines the width of the distribution. Varying i allows us to simulate the different stages of CE, as it converges to the true solution. We perform 100 model simulations for each parameter range and for each threshold. The threshold is taken to be a multiple of the objective function at the true value of the parameters. The results are given in Table 21. This is done for the Barnes problem.

At best, we can hope to save about 20 out of every 100 time steps. However, this requires a good estimate of the expected minimum being known. For $i = 0.5$, which corresponds to earlier in CE, we see slightly better potential savings of up to 26 out of every 100 time steps. Another option would be to terminate the simulation based on the size of the accumulated partial value of the objective function up to the current time in the simulation. This approach will further

γ \ i	0.5	1	2
1	0.26	0.17	0.19
1.1	0.21	0.14	0.15
1.2	0.15	0.10	0.11
1.3	0.11	0.07	0.07
1.4	0.07	0.03	0.04

Table 21: Early Termination Results for Barnes Problem. The numbers across the top indicate the value for i and the numbers in the left column indicate the threshold factor, γ. Each entry in the table indicates the fraction of steps saved, for the given threshold factor and value for i.

reduce the number of steps being taken, but it may also prematurely terminate simulations which start with a relatively large partially accumulated objective function value, but match the data very well further into the simulation.

6 Discussion

6.1 Unobservable Components of the Sate Vector

For the two methods proposed for obtaining improved initial guesses, we require the state vector to be observable. If only a subset of the components are observed, it might be possible to use these methods on a subset of parameters. If it is not possible to use either of these two methods, then the hybrid can still be used, but it might be that the missing data makes the optimization difficult. In this case, it has been demonstrated that a method like the Unscented Kalman Filter (UKF) can provide an effective alternative [22].

6.2 Other Ways to Improve Performance

Our improved initial guesses and the use of the local optimizer improve the performance of our CE optimization algorithm, but there is still room for further improvement. We briefly mention other modifications to the CE algorithm and other general ways to improve performance.

6.2.1 More CE modifications

The original implementation of CE assumes independence of parameters, which is obviously not realistic. The question is whether or not the additional effort to calculate the covariance and use it to draw samples from a multivariate Gaussian distribution can be justified. We looked into this a little, but it did not seem to improve convergence significantly for the problems we investigated.

We also found that even with our modification to how CE updated $\boldsymbol{s}$ in each iteration, the algorithm is unable to efficiently shift $\boldsymbol{\mu}$, since samples are drawn from a normal distribution. One potential way to address this would be to draw samples from a skewed normal distribution, which would make better use of samples when it is clear which way $\boldsymbol{\mu}$ is trying to move.

A major drawback of CE is having to decide on a reasonable value for N, given the number of elites we hope to obtain per iteration. This problem can potentially be solved by restructuring the algorithm. We propose keeping a global record of the elites, which is updated after every sample. An advantage of this alternative is that it no longer relies on sampling any true elites per iteration, as long as some have been found previously. In the current iterative setting, true elites must be sampled every iteration, which can be difficult to guarantee, especially when the sampling space is large, without choosing an undesirably high value for N. This also means that we can choose a larger number of elites, since we are no longer as concerned with N being too large. With more elites, it adds the possibility for analyzing the elites in order to identify

clusters within the elites, which could allow CE to branch and explore the two minimums, rather than getting caught between the two and eventually exploring only one of them.

6.2.2 Step size Selection

We look at how we might set the CRK solver's step size in order to reduce the time needed to perform a simulation of the model. First, we try setting the initial step size to that of the average step size on a previous simulation. This shows practically no change in performance. We next look at setting all the step sizes taken, using the step sizes taken by the simulation corresponding to the lowest objective function value found so far during CE. This should reduce the number of failed steps, since we are telling the solver to use a smaller step in cases where it would normally attempt too large of a step. However, the setup of the CRK solver does not allow the trial step size to be set directly, but only a starting step size, which is manipulated to compute the trial step size. While it is certainly possible to bypass these manipulations and set the trial step size directly, we decide it is not worth the effort to do so.

6.2.3 Parallel Processing

CE can be parallelized very naturally, since the simulation of the N samples in each iteration independent of one another. A parallel version of the code is prototyped using MPI [19], but we leave further investigating this to future work. As mentioned, the subinterval simulations in MS could also be computed in parallel.

7 Conclusions and Future Work

7.1 Conclusion

We present several modifications to the CE algorithm for parameter estimation of ODE and DDE models. Software is developed building on the existing DDEM package for working with DDEs. The CE modifications are implemented and the performance of a 3-stage parameter estimation procedure is demonstrated for six test problems. Using a hybrid optimizer significantly reduces the number of simulations required by CE, without reducing the reliability of the estimates obtained. With the addition of our inexpensive initial estimation stage, we obtain a further reduction in the number of simulations needed to perform the estimation.

For our three non-stiff ODEs, we compare our approach with multi-start multiple shooting and find that our method is slightly more expensive, but has less difficulty converging on one of our test problems. For DDEs, our method performs better than simple shooting, with improved convergence that outweighs the additional cost. Our 3-stage procedure is not as effective on our final test problem, which is stiff, but using our data-driven approach in the first stage, we observe a reduction in computation time in the cases where the global minimum is found. For stiff problems like this, a stiff ODE solver should be used to reduce the cost per simulation.

Besides looking at the number of simulations needed, we also looked at ways to reduce the cost of the simulations. We quantify the potential savings of using a threshold objective function value to allow for model simulations to be terminated early. For each of our test problems, we show how the number of steps per simulation of the model decreases as the simulation tolerance is relaxed. If we can justify relaxing the simulation tolerance, we can further reduce the cost of performing parameter estimation.

7.2 Future Work

Further work is needed to make this method of parameter estimation available as an easily usable set of MATLAB routines and C++ classes. Applying these approaches to real data sets

is necessary to further demonstrate their validity. In future, more focus will be on working with larger systems in a highly parallel environment, where the computer time to compute large numbers of simulations is not as restrictive as in a sequential environment. Another direction to take is to apply CE to other classes of problems, such as partial differential equations (PDEs) or stochastic differential equations (SDEs).

References

[1] E. Balsa-Canto, M. Pfeifer, J. Banga, J. Timmer, C. Fleck. Hybrid optimization method with general switching strategy for parameter estimation. BMC Systems Biology. Vol. 2 no. 1, pp. 26, 2008.

[2] M. Rodriguez-Fernandez, P. Mendes, J.R. Banga. A hybrid approach for efficient and robust parameter estimation in biochemical pathways. BioSystems. 83:248-265, 2006.

[3] R. Barrio. Sensitivity Analysis of ODEs/DAEs Using The Taylor Series Method. SIAM J. Sci. Comput., 27(6):1929-1947, 2006.

[4] S. Bochkanov. ALGLIB (www.alglib.net).

[5] H.G. Bock and K.J Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In Proceedings 9th IFAC World Congress Budapest, pages 243-247. Pergamon Press, 1984.

[6] FitzHugh R. Impulses And Physiological States in Models of Nerve Membrane. Biophysical Journal. 1:445–466, 1961.

[7] D. Gonze, W. Abou-Jaoudé. The Goodwin Model: Behind the Hill Function. PLoS ONE 8(8): e69573, 2013. doi:10.1371/journal.pone.0069573

[8] B. Goodwin. Oscillatory behavior in enzymatic control processes. Advances in Enzyme Regulation, 3: 425-428, 1965.

[9] N. Hansen, A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation, 9(2) pp. 159–195. 2001.

[10] W. Horbelt, J. Timmer, and H.U. Voss. Parameter estimation in nonlinear delayed feedback systems from noisy data. Physical Letters A, 299:513–521, 2002.

[11] G. E. Hutchinson. Circular causal systems in ecology. Annals of the New York Academy of Sciences 50: 221–246, 1948.

[12] W. Kermack and A. McKendrick, Contributions to the mathematical theory of epidemics, Part I, Proc. R. Slat. Soc. A115, 700-721 (1927).

[13] F.T. Krogh, J.P. Keener, W.H. Enright. Reducing the number of variational equations in the implementation of multiple shooting. In Numerical Boundary Value ODEs, pages 121-135. 1985.

[14] Y. Kuang, ed. Delay differential equations: with applications in population dynamics. Academic Press, 1993.

[15] U. Kummer, L. F. Olsen, C. J. Dixon, A. K. Green, E. Bornber-Bauer, G. Baier. Switching from simple to complex oscillations in calcium signaling. Biophys. J., 79:1188-1195, 2000.

[16] K. Levenberg. A Method for the Solution of Certain Non-Linear Problems in Least Squares. Quarterly of Applied Mathematics 2: 164–168. 1944.

[17] M.I.A. Lourakis, levmar: Levenberg-Marquardt nonlinear least squares algorithms in C/C++, http://www.ics.forth.gr/ lourakis/levmar/, 2004.

[18] D. Marquardt. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. SIAM Journal on Applied Mathematics 11 (2): 431–441. doi:10.1137/0111030. 1963.

[19] Message P Forum. Mpi: a Message-Passing Interface Standard. Technical Report. University of Tennessee, Knoxville, TN, USA. 1994.

[20] C.G. Moles, P. Mendes, J.R. Banga. Parameter estimation in biochemical pathways: a comparison of global optimization methods. Genome research, 13:2467-2474, 2003.

[21] The NAG Library (2013), The Numerical Algorithms Group (NAG), Oxford, United Kingdom www.nag.com

[22] M. Quach, N. Brunel, and F. d'Alché-Buc. Estimating parameters and hidden variables in non-linear state-space models based on ODEs for biological networks inference. Bioinformatics 23 (23): 3209-3216. doi: 10.1093/bioinformatics/btm510. 2007.

[23] M. Peifer, J. Timmer. Parameter estimation in ordinary differential equations for biochemical processes using the method of multiple shooting. IET System Biology, 1:78-88, 2007.

[24] J.O. Ramsay, G. Hooker, D. Campbell, and J. Cao. Parameter estimation for differential equations: a generalized smoothing approach. Journal Of The Royal Statistical Society Series B, 69(5):741–796, 2007.

[25] R.Y. Rubinstein, D.P. Kroese. The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning, Springer-Verlag, New York, 2004.

[26] J.M. Varah. A spline least squares method for numerical parameter estimation in differential equations. SIAM, Journal of Scientific and Statistic Computation, 3:28–46, 1982.

[27] B. Wang. Parameter Estimation for ODEs using a Cross-Entropy Approach. M.Sc. Thesis. University of Toronto: Canada, 2012.

[28] L. Wang and J. Cao. Estimating Parameters in Delay Differential Equation Models. Journal of Agricultural, Biological, and Environmental Statistics, 17, 68-83, 2012.

[29] H. Zivaripiran. Efficient Simulation, Accurate Sensitivity Analysis and Reliable Parameter Estimation for Delay Differential Equations. PhD thesis, Computer Science Department, Univeristy of Toronto, 2009.