Hindawi
Journal of Applied Mathematics
Volume 2018, Article ID 9160793, 9 pages
https://doi.org/10.1155/2018/9160793

Research Article

Parameter Estimation in Ordinary Differential Equations Modeling via Particle Swarm Optimization

Devin Akman,[1] Olcay Akman,[2] and Elsa Schaefer[3]

[1]*University of Illinois Urbana-Champaign, USA*
[2]*Illinois State University, USA*
[3]*Marymount University, USA*

Correspondence should be addressed to Olcay Akman; oakman@ilstu.edu

Received 25 May 2018; Accepted 29 July 2018; Published 2 September 2018

Academic Editor: Theodore E. Simos

Researchers using ordinary differential equations to model phenomena face two main challenges among others: implementing the appropriate model and optimizing the parameters of the selected model. The latter often proves difficult or computationally expensive. Here, we implement Particle Swarm Optimization, which draws inspiration from the optimizing behavior of insect swarms in nature, as it is a simple and efficient method for fitting models to data. We demonstrate its efficacy by showing that it outstrips evolutionary computing methods previously used to analyze an epidemic model.

1. Introduction

In mathematical biology, parameter estimation is one of the most important components of model fitting procedure. With poorly estimated parameters, even the most appropriate models perform poorly. Although there is already an abundance of traditional algorithms in the literature, such as Gauss-Newton methods, the Nelder-Mead method, and simulated annealing (see [1] for a thorough review), as models get more complex, researchers need more versatile and capable optimization methods. Evolutionary computing methods are becoming more frequently used tools, reducing the computational cost of model fitting, and starting to attract interest among mathematical biologists.

Compartmental models that are frequently used in infectious disease modeling have not escaped the computational cost versus complex model dilemma either. Since parameter estimation for compartmental models can be tackled by such evolutionary computing methods, it is reasonable to examine the goodness of fit versus the price of fit for these commonly employed models. Therefore, we will focus on the lesser-used evolutionary algorithms in comparison to Particle Swarm Optimization (PSO). In particular, Genetic Algorithms (GA) have been frequently used to optimize the parameters of ordinary differential equations (ODE) models [2–7]. PSO has thus far been underutilized in this area. It is an optimization algorithm inspired by swarms of insects, birds, and fish in nature. Although PSO has been applied in a number of different scenarios [8], its performance on compartmental models has not yet been studied.

In optimization, the computational cost of an algorithm is just as important as the quality of its output. Unfortunately for us, cost increases with quality. Evolutionary computing algorithms are especially susceptible to this effect, wherein small reductions in error must be paid for by disproportionate amounts of additional computation. To reduce cost or improve accuracy without compromising the other requires the application of an innovative technique to the problem. For example, Hallam et al. [9] implemented progressive population culling in a genetic algorithm to hit a given error target in fewer CPU cycles.

Here, we intend to show that PSO is not only a viable technique for fitting ODE models to data, but also that it has the potential to outperform GA, which was proposed in [2] as a superior optimization method to the traditional ones for fitting ODE models. Hence, with this study, we aim to establish an even more viable tool for ODE model fitting. Specifically, we apply PSO to kinetic parameter optimization/fitting in

the context of ODEs, and we contrast the PSO and GA algorithms using the cholera SIRB example as common ground. The organization of this article is as follows: In Section 2, we introduce PSO. Section 3 contains a description of how PSO can be applied to ODE models. We illustrate an implementation of PSO to optimize an ODE model of cholera infections during the recent Haitian epidemic in Section 4. Finally, we provide concluding remarks in Section 5.

2. Particle Swarm Optimization

We are given an objective function $f : \mathbb{R}^D \longrightarrow \mathbb{R}$ with the goal of minimization. Based on reasonable parameter bounds for the problem at hand, we constrain our search space to the Cartesian product

$$\Sigma = \prod_{i=1}^{D} [L_i, U_i] \tag{1}$$

for lower bounds $\{L_i\}$ and upper bounds $\{U_i\}$. We release a "swarm" of point particles within the space and iteratively update their positions. The simulation rules allow the particles to explore and exploit the space all the while communicating with one another about their discoveries. Since PSO does not rely on gradients or smoothness in general, it is well-suited to optimizing chaotic and discontinuous systems. This flexibility is an essential property in the domain of ODE models, whose prediction errors may change drastically with minute parameter variations. The swarm nature of PSO makes it amenable to parallelization, giving it a sizable advantage over sequential methods. The algorithm used for our computations is largely based on SPSO 2011 as found in [10]. We provide implementation details in Section 2. Then, we discuss ODE models, as they are ubiquitous in the realm of modeling natural phenomena. We provide a justification of our procedure for hyperspherical sampling in the Appendix.

2.1. Particle Swarm Optimization Implementation

2.1.1. Initialization. First, we define a new objective function $g : [0,1]^D \longrightarrow \mathbb{R}$ by

$$g(x) = f(b + Mx), \tag{2}$$

where $b = (L_1, \cdots, L_D)^T$ and $M = \text{diag}(U_1 - L_1, \cdots, U_D - L_D)$. Hence, the particles search within the unit hypercube, which provides better hyperspherical sampling performance than does a search space whose dimensions could be orders of magnitude apart. The particles' positions are then affinely mapped to S before evaluation by the objective function.

A swarm consists of an ordered list of S particles. This value is manually adjusted to fit the problem at hand, although more advanced methods may attempt to select it with a metaoptimizer or adaptively adjust it during execution. If S is too low, then the particles will not be able to explore the space effectively; if it is too high, then computational costs will rise, without yielding a significant improvement in the final fitness. Each particle has attributes which are updated at every iteration. The attributes are shown in Table 1, and their initial values ($t = 0$) are given. We define $U(a, b)$ to mean a value sampled uniformly from $[a, b]$. The particles are initially placed in the search space via Latin Hypercube Sampling (LHS). This helps avoid the excessive clustering that can occur with truly random placement, as we want all areas of the search space to be considered. The reader is directed to [11] for more details on LHS.

The PSO algorithm derives its power from the communication of personal best positions among particles. During the initialization phase, each particle randomly and uniformly selects K natural numbers in $[1, S]$ (with replacement) and adds its own index j to the list. Once again, K is a tunable parameter of the algorithm; one might use $K = 3$ as a good starting point. These numbers, with repeats discarded, form a set N_j. They tell the particle which of its peers to communicate with when it discovers lower values of the objective function during the execution of the algorithm. This communication is also performed once to initialize the l_j values before any motion has taken place. In addition, N_j is continually updated based on the performance of the current configuration; this aspect of the algorithm is known as its *adaptive random topology*.

2.2. Pragmatic Considerations

2.2.1. Exploration and Exploitation. The degree of *exploration* in a given PSO configuration refers to the propensity of the particles to travel large distances within the search space and visit different regions. This must be properly balanced with *exploitation*, which is the tendency of particles to stay within a given region and examine local features to seek out even lower values of the objective function. The PSO specification provides a parameter to this end, and it may have to be manually or automatically tuned for best results on a particular function landscape.

2.3. Algorithm. We present a description of the algorithm after initialization below. Each step is followed by a comment that may describe its motivation, practical implementation details, or pitfalls. Table 1 contains descriptions of the particle attributes. The parameters c and ω (not to be confused with the ω in the SIRB model) used in the algorithm must be selected beforehand. They control the exploration-exploitation balance and the "inertia" of the particles, respectively.

(1) Apply a random permutation to the indices of the particles.

Permutations can be selected uniformly at random with the simple Fisher-Yates shuffle.

(2) For $j = 0$ to $S - 1$,

(a) Set $G_j = x_j + c(p_j + l_j - 2x_j)/3$.
We define G_j to be the "center of gravity" of these points: x_j, $x_j + c(p_j - x_j)$, and $x_j + c(l_j - x_j)$. This provides a mean position around which the particle can search for new locations. The larger c is, the farther the particle will venture on average. Previous versions of SPSO had detectable biases

Table 1: Particle attributes indexed by position in list.

Property	Domain	Meaning	Initial Value
x_j	$[0,1]^D$	position	determined via LHS
v_j	$\mathbb{R}^D$	velocity	$v_{j,d}(0) = U(-x_{j,d}(0), 1 - x_{j,d}(0))$
p_j	$[0,1]^D$	personal best position	$x_j(0)$
q_j	$\mathbb{R}$	personal best fitness	$g(x_j(0))$
N_j	$2^{\mathbb{N}}$	neighborhood	described below
l_j	$[0,1]^D$	local best position	described below
m_j	$\mathbb{R}$	local best fitness	$g(l_j(0))$

along the axes of the coordinate system. This coordinate-free definition elegantly avoids such a problem.

(i) If $p_j = l_j$, let $G_j = x_j + c(p_j - x_j)/2$ instead.
If the local best position is equal to the particle's personal best, then the original formula would give a weight of $2/3$ to $x_j + c(p_j - x_j)$ and only $1/3$ to x_j. To avoid this, we revise the definition of G_j to be the mean of only x_j and $x_j + c(p_j - x_j)$ in this case. Here, we are testing the coordinates for bitwise equality, as opposed to the usual floating point comparison. This is because a particle will have p_j exactly equal to l_j when it has the lowest personal best fitness in its network of peers.

(b) Sample x'_j uniformly from the volume of a hypersphere with center G_j and radius $r = \|x_j - G_j\|$.
One might naively attempt this by generating a vector representing displacement from G_j whose coordinates are $U(0,1)$ each and scaling it to a length of $U(0,r)$. However, this does not produce a constant probability density. The proper method is to generate a vector whose coordinates are standard normal variates and scale it to a length of $rU(0,1)^{1/D}$. Normal variates can be generated from a uniform source with methods such as the Box-Muller transform, and Ziggurat algorithm.

(c) Set $v_j(t+1) = \omega v_j(t) + [x'_j(t) - x_j(t)]$.
The position x'_j represents where the particle intends to travel next. The second term of $v_j(t+1)$ contains the corresponding displacement from its current position. However, we would also like to retain some of the particle's old velocity to lessen the chance of getting trapped in local minimum. To this end, we add an inertial term with a "cooldown" factor ω that functions analogously to friction.

(d) Set $x_j(t+1) = x_j(t) + v_j(t+1)$.
The next position of the particle is its current position plus its updated velocity.

(e) Clamp position if necessary.
If any component of a particle's position falls outside $[0,1]$, we clamp it to the appropriate endpoint and multiply the corresponding component of its velocity by -0.5. This inelastic "bouncing" contains particles, while still allowing exploitation of the periphery of the search space.

(f) Increment t.

(g) If $g(x_j(t)) < q_j(t-1)$, set $p_j(t) = x_j(t)$ and $q_j(t) = g(x_j(t))$. Else, carry forward values from previous iteration.
If the value of the objective function at the current position is better than the personal best value from the previous iteration, set the personal best position to the current position and the personal best value to the current value.

(h) Communicate with neighbors.
When a particle finds a value of the objective function lower than its m_j (which corresponds to the position l_j), it informs all of its neighbors by updating their values of these attributes to its current objective function value and position.

(i) Regenerate topology if global fitness $\min_k q_k$ did not improve from previous step.
If the global fitness of the swarm has not improved at the end of the iteration, then the sets N_j are updated by regenerating them as during the initialization phase.

(j) Check stopping condition. See below for example conditions.

A multitude of conditions can be used to decide when to stop. Some common ones are as follows:

(i) The iteration count exceeds a threshold T. For a fixed threshold, this method gives more computation time to larger swarms.

(ii) The product of the iteration count and the swarm size (the number of objective function evaluations) exceeds T. In this case, the threshold is a good representation of how much total computation time the swarm utilizes, as the computational resource limit is nearly independent of swarm size.

(iii) The globally optimal objective function value ($\min_j q_j$) falls below some threshold.

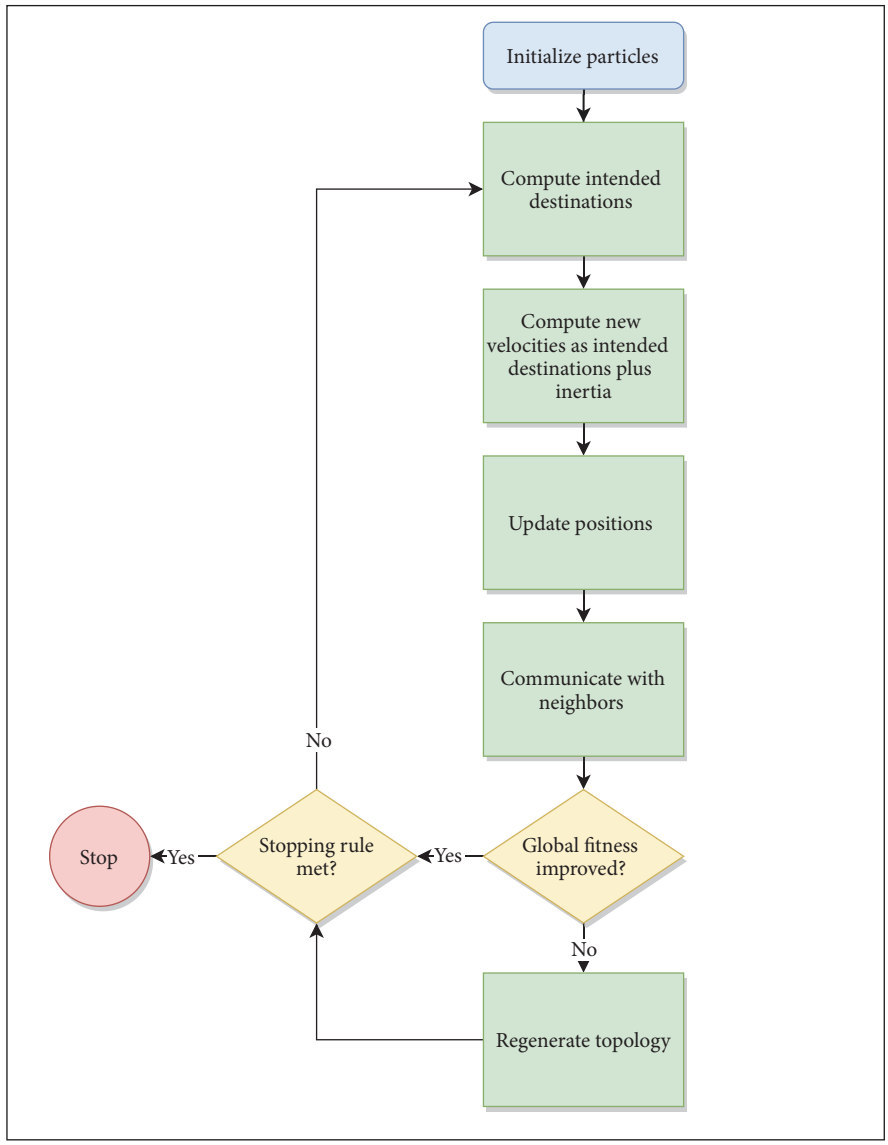

Figure 1: Particle Swarm Optimization process.

(iv) The globally optimal objective function value has not improved by more than ϵ within the last N iterations.

(v) Any combination of the above.

The output of the algorithm is then the p_j value corresponding to the minimum q_j value.

The flowchart (Figure 1) summarizes the process of Particle Swarm Optimization.

2.3.1. Discrete Parameters. If some (not necessarily all) of the parameters in the problem are discrete, PSO can be adapted by snapping the appropriate continuous coordinates in the search space to the nearest acceptable values during each iteration. This feature gives PSO great versatility, in that it can solve problems with entirely continuous, mixed, or entirely discrete inputs.

2.3.2. Parallel Computing. Most real-world objective functions are sufficiently complex so that PSO will spend the vast majority of its time evaluating them, as opposed to executing the control logic. In our case, profiling showed that numerically solving an ODE at many points consumed more than 95% of the total CPU time. Therefore, in order to parallelize the algorithm, one needs only to evenly distribute these evaluations among different cores or possibly computers. The remaining logic of the algorithm can be executed in a single control thread that collates the fitness values and updates particle attributes before offloading evaluations for the next iteration.

2.3.3. Randomness. Far too often in our experience, probabilistic numerical simulations are run with little thought to the underlying random number generator (RNG). As Clerc

Table 2: Parameter ranges for cholera model.

Name	Description	Lower bound	Upper bound
b	Natural birth rate of humans	0.000072	NA
d	Natural death rate of humans	0.000044	NA
κ	Half-saturation constant of vibrios	10^6	NA
B_0	Starting bacterial concentration	0	1
h	Proportion of hospitalizations	0.001	1
β_B	Ingestion rate of vibrios	0.001	1
β_I	Infectivity for susceptible	0.001	0.5
η	Rate of contribution by infected	0.001	1
γ	Recovery rate of infected	1/14	1/2
δ	Decay rate of vibrios to non-HI state	1/40	1/3
ω	Rate of waning immunity	0.0001	1/360

states in [10], the performance of PSO on some problems is highly dependent on the quality of the RNG. The default RNG in one's favorite programming language is likely not up to the task. By default, our code uses the fast, simple, and statistically robust `xorshift128+` generator from [12]. It outputs random 64-bit blocks, which are transformed using standard algorithms into the various types of random samples required. The use of a seed allows for exactly reproducible simulations (this is more difficult to ensure if a single generator is being accessed by multiple threads).

3. ODE Models

3.1. Introduction. Suppose we have data and a corresponding system of ordinary differential equations which we believe models the underlying process. The system takes parameters and initial values (some known, some unknown) as input and outputs functions, which can then be compared to the data using some goodness of fit (GOF) metric (e.g., mean squared error). Competing models can be evaluated using frameworks introduced in [3].

If we take the vector of unknown inputs to be a position within search space, then we can apply PSO in order to minimize the model's error. The choice of error function is an important consideration. Those GOF metrics that are superlinear in the individual deviations will preferentially reduce larger deviations over smaller ones. Similarly, sublinear functions will produce good correspondence on most data points at the expense of some larger errors.

3.2. Compartmental Models. In epidemiology, a common tool for studying the spread of an infectious disease is the compartmental model. The prototypical example is the SIR model, in which the population is divided into three compartments: *Susceptible*, *Infected*, and *Recovering*. Differential equations describe how individuals flow between the compartments over time. The behavior of a specific disease is captured by parameters, and it is here that we use model fitting techniques to derive these parameters from data. The simplest form of SIR does not include birth or death (the dynamics of the disease are assumed to be sufficiently rapid), so it holds that the population size $N = S + I + R$ is constant. More advanced models include more compartments and incorporate factors such as demography. In the field, it is often easiest to measure the number of infected individuals; thus a viable GOF metric may compute the distance between time series infection data and the corresponding points on the $I(t)$ curve for a given set of parameters.

Chubarova [13] implemented GA for fitting ODE models to cholera epidemic data. Although better errors were achieved than similar studies that used traditional methods, the computational cost required was substantial. This led to our consideration of a technique which could achieve similar accuracy with fewer cycles, namely PSO. One of the models studied in [13] was SIRB, which is a refinement of SIR: it includes birth and death rates for the population and a compartment B representing the concentration of bacteria in the water supply. We present equations for SIRB and describe the roles of its parameters in Section 4.

4. Application

4.1. SIRB. The SIRB model is a system of four nonlinear first-order ODEs that capture how quickly individuals progress through the susceptibility-infection-recovery cycle and how bacteria are exchanged between the population and the water supply:

$$\begin{aligned}
\frac{dS}{dt} &= bN - dS - \beta_B \frac{BS}{\kappa + B} - \beta_I \frac{SI}{N} + \omega R \\
\frac{dI}{dt} &= -dI + \beta_B \frac{BS}{\kappa + B} + \beta_I \frac{SI}{N} - \gamma I \\
\frac{dR}{dt} &= -dR + \gamma I - \omega R \\
\frac{dB}{dt} &= \eta I - \delta B
\end{aligned} \tag{3}$$

The parameters and their ranges are listed in Table 2 (derived from [2] with some later corrections from the authors). An upper bound of NA means that the value is constant and not optimized by PSO. The bacteria being studied in this case are of the genus *Vibrio*. They decay from a state of hyperinfectivity (HI) to a non-HI state (only the

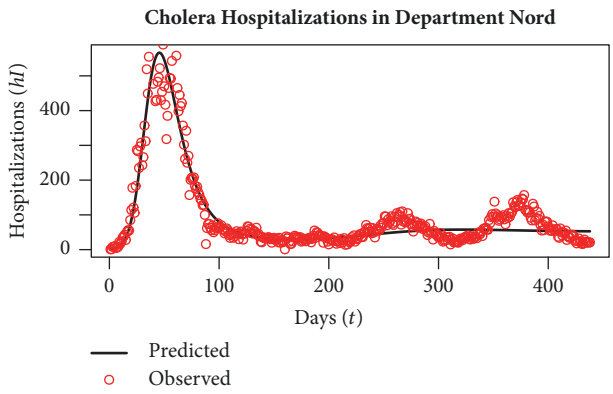

Figure 2: Example run.

former is considered infectious in this model) at a rate of δ. The parameter κ has units of cells per milliliter, while the rest have units of inverse days.

4.2. Model Fitting. We apply PSO to fit the SIRB model to epidemic data from [14] using our own C code and the *rkf45* ODE solver in the GNU Scientific Library. Since the data were obtained from hospitals, we must correct for the fact that not every infected person received care. We introduce the parameter h as the proportion of recorded infections. Instead of presuming a certain value, we allow h to be optimized by PSO. The initial values for the system for Department Nord are $S(0) = 549{,}486$, $I(0) = 1/h$, $R(0) = 0$, and $B(0) = \kappa B_0$, where B_0 is another parameter to be optimized by PSO. The particles explore an 8-dimensional search space with generic position vector

$$x = \left(B_0, h, \beta_B, \beta_I, \eta, \gamma, \delta, \omega\right)^T. \tag{4}$$

We measure GOF using the mean absolute deviation:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} \left|hI(t_i) - y_i\right|, \tag{5}$$

where $\{(t_i, y_i)\}$ is the time series infection data. Figure 2 shows a plot of $hI(i)$ and y_i versus t for an example run of the PSO algorithm. The tail behavior of our model is typical in that it fails to account for cyclical recurrence of the disease. Akman et al. address this phenomenon in [2].

We performed 12 runs of the algorithm with the parameters $c = 1.193$, $\omega = 0.721$, $S = 40$, and $K = 3$ (suggested values from [10]). We chose a stopping condition of two million objective function evaluations. We found this condition experimentally by observing that fitness values rarely, if ever, improved beyond this point. Our results are given in Table 3. An individual run only takes around 8 minutes on a quad-core Intel Celeron J1900 @ 1.99 GHz (compiled with `icc -O2`). The same optimization problem, approached using GA in [13], took between approximately 1 and 6.4 days to run on two hexa-core Intel Xeon X5670s @ 2.93 GHz (see Table 4). Considering the similarity of the fitness values to the evolutionary computing results, PSO appears to be an attractive method for model fitting, especially for the case of expensive objective functions. It uses smaller populations and fewer timesteps due to the fast convergence of the fitness values, which distinguishes it from GA.

Code for PSO and the SIRB model with sample data is available at [11].

5. Concluding Remarks

The use of PSO is a novel approach in ODE modeling, particularly in the field of epidemiology. Where previous algorithms such as GA have taken excessively long to run in the face of complex models, PSO demonstrates its effectiveness by providing comparable results in an impressively small fraction of the time. It has a structural advantage over GA that cannot be attributed to the complexity of the control logic; PSO requires many fewer objective function calls to reach the same level of accuracy. We implemented PSO to optimize an ODE model given cholera data. Although there are other global optimization methods that are also used in fitting parameters of ODE models, such as simulated annealing or scatter search, we restricted our comparison only to the performance of GA, since it was extensively used for the same infectious disease model. We achieved similar model errors in a substantially shorter period of time, as explained above, due to the fundamental differences of operation between PSO and GA. Additionally, we compared the performance of PSO with that of DIRECT [15–19] per the suggestion of a referee. DIRECT evaluated the objective function approximately six thousand times in the same amount of time it took PSO to perform two million evaluations, and it appeared to converge to a worse fitness value.

As with most optimization algorithms, the remaining challenge of PSO is algorithm parameter selection. In the context of our study, this is similar to selecting the right GA parameters, such as population size, number of generations, cross-over rate, and mutation rates; that all may have an impact on the accuracy and runtime. One possible improvement is to add a metaoptimizer to choose the algorithm parameters without human intervention. A fruitful potential topic of investigation is to extend PSO to approximating unknown functions (such as probability distributions) rather than merely selecting parameters.

Appendix

Hyperspherical Sampling

In this appendix, we justify our procedure for uniform hyperspherical sampling. Our goal is to provide a practical construction for a random variable W which takes on values from the unit D-ball with a uniform probability density. Then rW will provide the appropriate distribution for the PSO algorithm. Let Z be a random variable distributed according to the D-variate normal distribution of identity covariance, centered at the origin. We can sample from Z in our code by generating a vector with D independent standard normal components. It is well known that the distribution of Z is rotationally symmetric. Therefore, defining $f(z) = z/\|z\|$, we

Table 3: Parameter values from PSO and corresponding fitness.

B_0	h	β_B	β_I	η	γ	δ	ω	Fitness	Time (s)
0.006170	0.003569	0.0.129567	0.001522	0.875338	0.080457	0.078863	0.002336	27.19	481
0.006424	0.008251	0.102802	0.077925	0.796273	0.182466	0.048924	0.002660	28.26	484
0.007413	0.00319664	0.159124	0.021619	0.617604	0.073014	0.100941	0.002739	27.18	477
0.005186	0.003286	0.150995	0.042600	1.000000	0.071429	0.227019	0.002769	28.25	477
0.007859	0.003576	0.096382	0.138211	0.321430	0.079145	0.049010	0.002703	27.76	473
0.053613	0.00334816	0.009927	0.020090	0.664410	0.071454	0.026494	0.002743	27.69	477
0.009432	0.004611	0.122929	0.033397	581153	0.103619	0.048074	0.002384	27.54	482
0.005751	0.003202	0.184915	0.001033	0.933074	0.071438	0.222235	0.002777	27.73	485
0.008324	0.003625	0.100021	0.059395	0.781264	0.081375	0.072428	0.002535	27.29	478
0.010934	0.00354955	0.065875	0.148408	426006	0.078190	0.048143	0.002676	27.80	475
0.005570	0.003256	0.247551	0.009451	337538	0.073270	0.075757	0.002577	27.33	476
0.019549	0.003365	0.032144	0.180211	461570	0.073522	0.040767	0.0027560	27.88	477

Table 4: GA run statistics.

Time to make elites (s)	Time to make parameter sets (s)	Total time (s)	Sample size	Stopping threshold	Number of elites	Max generations	Parameter sets created	Fitness
8680	550	9230	5000	10	400	100	20	27.55
8643	528	9171	5000	10	400	100	20	27.55
8615	546	9161	5000	10	400	100	20	27.54
7446	543	7989	4000	10	400	100	20	27.51
7480	518	7998	4000	10	400	100	20	27.55
6306	571	6877	3000	10	400	100	20	27.54
6192	528	6720	3000	10	400	100	20	27.52
5115	550	5665	2000	10	400	100	20	27.54
5074	500	5574	2000	10	400	100	20	27.55
4285	517	4802	1000	10	400	100	20	27.48
4160	463	4623	1000	10	400	100	20	27.56
2359	562	2921	500	10	400	100	20	27.58
2780	556	3336	500	10	400	100	20	27.59
1963	652	2615	100	10	400	100	20	27.51
1892	612	2504	100	10	400	100	20	27.57
748	949	1697	50	10	400	100	20	27.88
561	941	1502	50	10	400	100	20	27.62
386	1052	1438	50	10	200	100	20	27.97
381	1118	1499	50	10	200	100	20	27.73
960	960	1920	100	10	200	100	20	27.56
951	849	1800	100	10	200	100	20	27.67

find that $f(Z)$ must be uniformly distributed on the surface of the unit D-sphere. Since a D-ball with radius $s \leq 1$ has volume proportional to s^D, we would like to find a random variable Y supported on $[0, 1]$ such that

$$P\left(\left\|Yf\left(Z\right)\right\| \leq s\right) \propto s^D \Longrightarrow$$
$$P\left(Y \leq s\right) \propto s^D. \tag{A.1}$$

It is easily shown that $Y = U(0,1)^{1/D}$ has a cumulative distribution function of $G(s) = s^D$. Setting $W = Yf(Z)$ produces the desired result.

The same distribution can also be produced in a simpler fashion by repeatedly generating a vector z with components of $U(-r, r)$ and rejecting it if $\|z\| > r$. However, the expected number of tries per sample is equal to the ratio of the volume of a D-cube to that of an inscribed D-ball, which grows too rapidly with D.

Data Availability

The cholera infection data used to support the findings of this study are readily available on the Haiti Department of Health website.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] J. M. Krueger, *Parameter Estimation Methods for Ordinary Differential Equation Models with Applications to Microbiology [PhD thesis]*, Virginia Tech, 2017.

[2] O. Akman and E. Schaefer, "An evolutionary computing approach for parameter estimation investigation of a model for cholera," *Journal of Biological Dynamics*, vol. 9, pp. 147–158, 2015.

[3] O. Akman, M. R. Corby, and E. Schaefer, "Examination of models for cholera: insights into model comparison methods," *Letters in Biomathematics*, vol. 3, no. 1, pp. 93–118, 2016.

[4] R. Scitovski and D. Jukić, "A method for solving the parameter identification problem for ordinary differential equations of the second order," *Applied Mathematics and Computation*, vol. 74, no. 2, pp. 273–291, 1996.

[5] N. Tutkun, "Parameter estimation in mathematical models using the real coded genetic algorithms," *Expert Systems with Applications*, vol. 36, no. 2, pp. 3342–3345, 2009.

[6] M. A. Khalik, M. Sherif, S. Saraya, and F. Areed, "Parameter identification problem: Real-coded GA approach," *Applied Mathematics and Computation*, vol. 187, no. 2, pp. 1495–1501, 2007.

[7] E. K. Nyarko and R. Scitovski, "Solving the parameter identification problem of mathematical models using genetic algorithms," *Applied Mathematics and Computation*, vol. 153, no. 3, pp. 651–658, 2004.

[8] A. Lazinica, *Particle Swarm Optimization*, Kirchengasse, 2009.

[9] J. W. Hallam, O. Akman, and F. Akman, "Genetic algorithms with shrinking population size," *Computational Statistics*, vol. 25, no. 4, pp. 691–705, 2010.

[10] M. Clerc, Standard Particle Swarm Optimisation (Rep.). Retrieved July 1, 2016, from http://clerc.maurice.free.fr/pso/SPSO_descriptions.pdf.

[11] R. L. Iman, "Latin hypercube sampling," in *Encyclopedia of Quantitative Risk Analysis and Assessment*, Wiley Online Library, 2008.

[12] S. Vigna, "Further scramblings of Marsaglia's generators," *Journal of Computational and Applied Mathematics*, vol. 315, pp. 175–181, 2017.

[13] I. N. Chubarova, *Modeling cholera using engineered genetic algorithm [PhD thesis]*, Illinois State University, 2001.

[14] "Republique d'haiti ministère de la santè publique et de la population: Rap-ports journaliers du mspp sur l'èvolution du cholèra en haiti," Retrieved September 6, 2014, from http://mspp.gouv.ht/site/downloads/.

[15] D. E. Finkel, *DIRECT Optimization Algorithm User Guide*, vol. 2, Center for Research in Scientific Computation, North Carolina State University, 2003.

[16] D. E. Finkel and C. T. Kelley, "Additive scaling and the DIRECT algorithm," *Journal of Global Optimization*, vol. 36, no. 4, pp. 597–608, 2006.

[17] J. M. Gablonsky and C. T. Kelley, "A Locally-Biased form of the DIRECT Algorithm," *Journal of Global Optimization*, vol. 21, no. 1, pp. 27–37, 2001.

[18] R. Grbić, E. K. Nyarko, and R. Scitovski, "A modification of the DIRECT method for Lipschitz global optimization for a symmetric function," *Journal of Global Optimization*, vol. 57, no. 4, pp. 1193–1212, 2013.

[19] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant," *Journal of Optimization Theory and Applications*, vol. 79, no. 1, pp. 157–181, 1993.