International Journal of Pure and Applied Mathematics

Volume 31 No. 1 2006, 9-22

A PARAMETER SEARCH ALGORITHM BASED ON OPTIMAL LINEAR CODES

Gregory M. Constantine[1], John Bartels[2], Marius Buliga[3] §,
Gilles Clermont[4], Yoram Vodovotz[5]

[1]Department of Mathematics
University of Pittsburgh
Pittsburgh, PA 15260, USA
e-mail: gmc@euler.math.pitt.edu

[2]Immunetrics, Inc.
Pittsburgh, PA 15260, USA

[3]Department of Mathematics
University of Pittsburgh-Bradford
Bradford, PA 16701, USA
e-mail: buliga@pitt.edu

[4]Department of Critical Care Medicine
University of Pittsburgh Medical Center
Pittsburgh, PA 15260, USA

[5] Department of Surgery
University of Pittsburgh Medical Center
Pittsburgh, PA 15260, USA

Abstract: Modeling by way of dynamical systems often leads to systems of ordinary differential equations involving a large number of parameters, such as growth rates and initial conditions. A parameter search algorithm based on optimal linear codes is developed having as aim the identification of different regimes of behaviour of the model and its calibration to data.

AMS Subject Classification: 68R01, 37M05
Key Words: optimization, differential equations, immune system, mathematical modeling

Received: June 7, 2006

§Correspondence author

1. Introduction

Acute systemic inflammation is triggered by stresses on a living organism such as infection or trauma. This response involves a cascade of events mediated by a network of cells and molecules. The process localizes and identifies an insult, strives to eliminate offending agents, and initiates a repair process.

Complexities of the immune system are discussed in Bone [1], Mira et al [8] and Medzhitov [7]. It has been suggested that mathematical modeling might provide an effective tool to grapple with the complexity of the inflammatory response to infection and trauma (Nathan [9], Buchman et al [2], Neugebauer et al [10]). Modeling is increasingly being used to address clinically relevant biological complexity, in some cases leading to novel predictions (Kitano [5], Chow et al [3]). In silico simulations based on mathematical models have recently been shown to be useful at the therapeutic level; cf. Clermont et al [4]. We embarked on an iterative process of model generation, verification and calibration in animal models, and subsequent hypothesis generation. The model, described by a system of differential equations (see Chow et al [3] for explicit lists of the ordinary differential equations involved) aims to explain the reaction of the immune system in several scenarios, by engaging processes that take place at the molecular level. The focus of this paper is on the development of a global search strategy of the parameter space that aids in predicting regime behaviour of the model as well as optimizing its fit to observed data. The general outline of the algorithm is presented in Section 2. Implementational issues are discussed in Section 3, with comparisons with other known algorithms forming the contents of Section 4. The regime classification and prediction, based on a multinomial logistic model, is described in the last section.

2. A Parameter Optimization Algorithm Based on Optimal Codes

The algorithm was developed out of the need to survey a high dimensional parameter space in which varying one or two parameters at a time, while keeping the others fixed, is simply not practical. It has at its core a (binary) code with efficient covering properties of the high dimensional parameter space. For expository transparency we worked with Hamming (binary) codes, but other codes over arbitrary finite fields with optimal covering properties may be used. The setting is that of a finite dimensional vector space V of dimension n over F_2, the binary field, endowed with the usual inner product with values in F_2. We specify the code as the orthogonal complement of the space generated by the

row vectors of a parity check matrix H of full row rank. The code is, therefore,

$$C = \{x : Hx = 0\}.$$

Let the dimension of C be k. Of equal interest are the cosets of the code. A coset C_y is specified as follows:

$$C_y = \{x : Hx = y\},$$

where y is the "syndrome" vector defining the coset in question. Since C has dimension k, the vector y is a column vector of dimension $n-k$. The set $\{C_y\}$ represents the set of 2^{n-k} cosets of C, as y runs over the set of all binary vectors of dimension $n-k$. The reader is refered to MacWilliams and Sloane [6] for material on coding theory.

We write out some of the details for the Hamming code in 7 dimensions, assuming that we have seven parameters, defined by the parity check matrix

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

This matrix H yields a code of dimension 4, consisting of 16 binary vectors.

We shall make use of a binary code C in the following way. The response function f of n (usually real) parameters $x_1, \ldots, x_n$ is to be investigated over a product space $B = \prod_{i=1}^{n}[a_i, b_i]$, with $a_i \leq x_i \leq b_i$; $1 \leq i \leq n$. For illustrative purposes we assume that the usually vector-valued response function is a scalar function. This function is in practice obtained by integrating the ODE system numerically (which we call *the model*) and assigning a measure of discrepancy between the model and the data. To describe how the algorithm works we assume that we want to numerically find the minimum of f over B. Take a vector $x = (x_1, \ldots, x_n) \in B$. Code x as a binary vector by placing a 0 in coordinate position i if $x_i \in [a_i, \frac{a_i+b_i}{2}]$ and a 1 otherwise; write $\bar{x}$ for the coded x.

By either a deliberate or a random process select a point $x_{i0} \in [a_i, \frac{a_i+b_i}{2}] = B_{i0}$ and a point $x_{i1} \in (\frac{a_i+b_i}{2}, b_i] = B_{i1}$; $1 \leq i \leq n$. A point $(x_{1j_1}, \ldots, x_{nj_n})$, with j_m being either 0 or 1, belongs to the "cell" $\prod_{i=1}^{n} B_{ij_i}$. Consider now the set of coded points of B,

$$\{(\bar{x}_{1j}, \ldots, \bar{x}_{nj}) : j = 0, 1\}.$$

This process defines 2^n binary vectors which we now view as the elements of an $n-$dimensional vector space V over the field with two elements F_2. Indeed,

with the points x_{ij} fixed, the correspondence

$$(x_{1j_1}, \ldots, x_{nj_n}) \leftrightarrow (\bar{x}_{1j_1}, \ldots, \bar{x}_{nj_n}),$$

with $j_m = 0$ or 1, is a bijection by way of which we shall identify the selected points of B, which we denote by $V(B)$, with elements of V. In particular, any subset S of points of V identifies through this bijection a corresponding set of points in B, which we write as $S(B)$. This is the case with the set of vectors in a linear code C of V as well. If C is a linear code of dimension k, the subset $C(B)$ consists of 2^k points in B. By abuse of language, we may on occasion refer to $C(B)$ as the points of the code C.

If the objective is to minimize function f over B, the algorithm first identifies the cells B_{ij} associated with the code C. Within each cell B_{ij} the algorithm replicates itself, that is, it treats cell B_{ij} as a new space B. It selects points in B_{ij} in accordance to (a code equivalent to) C (two codes are called equivalent if one is obtained from the other upon a permutation of coordinates). We evaluate the function f at all points selected in B_{ij}, for all cells B_{ij}. This allows us to identify a subset L of cells that yield the smallest values of f. If the minimum obtained so far is satisfactory, we stop. Else we iterate the procedure within each of the cells in L. This allows evaluation of the function f on a finer local mesh (at a deeper level of iteration). The nested level may be repeated any number of times. We stop when the minimum reached on f is sufficiently low. The list L keeps tabs of the addresses of the cells of interest at the various levels of iteration. All this happens only for cells associated with the code C. We can thus select from list L a point x_C (found in some cell of code C) such that $f(x_C)$ is the smallest value of f found so far. Produce the list of differences $L = \{\frac{f(x)-f(x_C)}{||x-x_C||} : x \in C(B)\}$ and select $x_0 \in C(B)$ such that $\frac{f(x_0)-f(x_C)}{||x_0-x_C||}$ is maximal (in general x in the list L actually runs over the set of cosets selected so far). Along the line $x_C - t(x_0 - x_C)$ find a smallest positive value t_0 of t such that $x_C - t_0(x_0 - x_C)$ is in a cell of $V(B)$ not examined thus far. The coded version of $x_C - t_0(x_0 - x_C) = y$ defines a vector in V not previously considered, and hence identifies a new coset $C_y = y + C$ of C. Before moving to the next coset a local analysis of the best cells found so far is performed as described in detail in the implementational section.

The coset C_y thus identified has the same optimal covering properties of V as does C. The process followed for code C is now repeated for the coset C_y. After examining $m \leq n-k$ cosets of C, call them $\{C_{y_1}, \ldots, C_{y_m}\}$ (let $y_1 = 0$, so $C_{y_1} = C$), we obtain m points $P = \{x_{C_{y_i}} : 1 \leq i \leq m\}$ from the corresponding m lists. The point in P at which f attains a minimum is the point the algorithm gives as solution to the optimization problem. The algorithm thus combines

local gradient properties with global reach throughout the region B by way of the cosets of the code C.

3. Implementational Issues

This recursive process of subdivision and evaluation provides a general framework for our search algorithm. In order to actually implement this as a procedure, we must make several decisions about the details of our search process. Below we discuss the major considerations that must be addressed, and present our strategies for dealing with them.

3.1. Search Procedure

The search algorithm proceeds in a manner similar to A* search; that is, we generate candidate subregions of the space to be searched, compute an estimated quality score for each candidate, and build a ranked list of these candidates. On each iteration of the algorithm, we remove the most promising region from the front of the list, evaluate sample points within that region to update our quality estimate, re-insert the updated region into the list, and possibly add to the list new candidate regions discovered during evaluation. As discussed later, some additional work may be done to bound the list growth within finite memory limitations. The following discussion assumes the existence of a customizable ranking function $R(c)$, which computes a measure of the quality of a given cell c in the coded space V. While the choice of $R(c)$ is crucial to succesful use of the algorithm, the search procedure itself is independent of how $R(c)$ is defined. Our discussion casts the search as a minimization problem and thus assumes that lower values of the objective function $f(x)$ and the ranking function $R(c)$ are preferred, though this could be inverted for maximization problems. Immediately below we describe the details of the search procedure in terms of an abstract $R(c)$, while discussion of the design considerations for a concrete $R(c)$ are continued later.

We require initial bounds for each parameter P_i, of the form $P_i \in [L_i, H_i]$. These bounds completely specify the n-dimensional search space B, and thus define our initial search cell c_0. We create the empty ranked-cell list L, and insert a record for c_0 into this list. We also define S_b to be the best solution ever seen, and initially set it to null. The routine then selects the most promising cell c from L, such that $R(c) \leq R(c')$ for all c' in L. The next step is to perform an evaluation of c, which requires a design decision on how to apply the code

C to cell c to identify the sample points for evaluation. Since each coset of C defines a minimal set of points which cover the search space at the desired resolution, we consider a coset to be our smallest unit of evaluation; i.e., k sample points are evaluated each time a cell is scored, where k is equal to the number of codewords in (each coset of our) code C. We now introduce a function $NextCoset(c)$, which determines the coset of C that should be applied during a given cell evaluation. The very first time a cell is evaluated, the primary coset of the code is applied. If the same cell c is later selected for re-evaluation (i.e., it becomes the best ranked cell in L), we may wish to evaluate a different coset of C to broaden our exploration of the cell. In our experiments, we tried two simple definitions of $NextCoset(c)$: the gradient method outlined above, and a sequential selection method. The sequential selection method simply indexes all j cosets of C from $\{0, 1, ..., j-1\}$, and selects the coset C_q, where $q = p \bmod j$ on the p-th evaluation of cell c. Once the coset is chosen, we must map each codeword in the coset from the discretized space V to a solution vector in the real-valued space B, as discussed previously. It is important to define this mapping in a non-static manner, so that repeated evaluations of the same coset in cell c sample different points rather than repeating previous samples. Our current implementation introduces this variation by simply making a uniformly distributed, random choice for each parameter P_i in codeword cw_j. If $cw_{j,i}$ is coded as 0, we choose $P_i \in [L_i, \frac{L_i+H_i}{2})$; otherwise we choose $P_i \in [\frac{L_i+H_i}{2}, H_i]$.

Through this process we obtain a real-valued solution vector S_j in B, corresponding to each codeword cw_j in our evaluating coset. We now evaluate our objective function on each of these points and obtain a score f(S_j) for each point. As these scores are calculated, they are compared with the best-ever score f(S_b), and S_b is updated if a better solution has been found. The procedure then updates its history information for the cell c, by adding the newly obtained set of $(S_j, f(S_j))$ pairs into the data structure associated with c in list L. Once the cell record has been updated, the quality score $R(c)$ of the cell is recomputed in light of these latest findings, and the cell's record is inserted back into L so as to maintain ranked ordering of cells. It is at this stage that the algorithm now opens the possibility for recursive searching of subregions of c. Since each codeword cw_j of the evaluating coset denotes a subregion of c, the point S_j sampled within that region serves as a crude measure of the quality of this subregion. If the score computed for some S_j was particularly promising, we may wish to narrow the focus of our search from c to the subcell containing S_j. We define a function $NewCells(cw)$ which abstracts the decision of whether to create new candidate search regions by adding entries to L for subcells of cell c. Our current implementation defines $NewCells(cw)$ in

the following way: a new cell c_i' is created in L for the subcell of c associated with codeword cw_i iff $f(c_i) \leq f(c_b)$, where c_b denotes the best solution known within c prior to evaluating this coset. To control the rate at which cells are added, we also include an upper bound $MaxCells$, which limits insertion to only the best-ranked $MaxCells$ subcells of c that showed improvement. The quality measure $R(c')$ is computed for each new subcell c', and these cells are inserted into L in ranked order. This completes a single iteration of the search, and the algorithm begins another iteration by by removing the best-ranked cell from the front of the list and repeating the process described above. This loop may be terminated either when a sufficiently good score has been obtained, a certain number of evaluations have been performed, or a fixed runtime limit has elapsed. Upon termination of the routine, the best solution S_b and its score f(S_b) are returned.

3.2. Memory Considerations

The description above assumes that we can track a potentially infinite number of cells and evaluated points in the list L as the search continues. In practice the algorithm must run on a machine with finite memory resources, and we must decide how to meet these limitations while sacrificing as little as possible of our valuable evaluation histories. We draw on the idea of Simplified Memory-Bounded A* search (SMA*). At the outset, an upper bound is set for the amount of memory available to the algorithm, and the algorithm must check that this limit is not exceeded when updating L. While the limit is not exceeded, the algorithm proceeds exactly as described above. Once the limit is reached, we cannot add or update cells in the list L without first removing others. We wish to lose the least valuable evaluation histories, and to avoid biasing our search by discarding regions we may wish to return to later. It is useful to visualize the cells in L as a tree, where each cell in L is considered a child of the smallest cell in L that fully contains it. From this point of view, our goals are best satisified by rejecting the poorest-ranked leaf-nodes of the tree. The algorithm is least interested in revisiting poorly-ranked cells, so sacrificing is a minimal loss. Furthermore, by rejecting these deeper cells of the tree and retaining their parents, we ensure that the algorithm has not forever abandoned certain regions of the search space; it can return to these regions later if subsequent evaluations of parent cells make them look promising once again. Our policy is then to discard the cell d such that $d \in LeafNodes(L)$ and $R(d) \geq R(l)$ for all $l \in LeafNodes(L)$. This process is repeated as necessary until the required amount of memory has been freed, then the algorithm completes its insertions

and resumes.

3.3. Design of Ranking Functions

The ranking function $R(c)$ serves as our estimate of how worthwhile it is to continue searching within a given cell c. The design of this function is crucial to the performance of the algorithm, as it largely determines the course the search will follow. We outline several considerations that influence the choice of $R(c)$, suggest some implementation strategies, and report comparisons of the results for each.

3.4. Scoring

The simplest measure of a cell's quality is the set of objective function scores obtained at points within that cell. Since the ultimate goal is to report the best solution ever discovered, we are not concerned if the cell containing the optimal point also contains suboptimal points. One might then recommend ranking a cell according to the best score ever found inside of it. However, we are also concerned about the efficiency of our search; a cell which requires vast amounts of evaluation to discover a good solution may not be as useful as one that finds a slightly worse solution much faster. Furthermore, using score as the sole ranking criteria always confines the search to the best cell known at any given time, ensuring that the search will become stuck at a local minimum. To address both of these concerns, we suggest that the best score found in a cell should contribute very significantly to the cell's rank, but other factors must be weighed as well. In particular, it is important that once-promising cells should decay in rank if more evaluations do not yield improved solutions. These concerns motivate the following additions to the ranking function.

3.5. Ranking Function Inputs

The algorithm as described works by successively narrowing in on regions of the search space that appear to contain more promising solutions. This is founded on the assumption that there is considerable smoothness in the distribution of scores in the search space. As the space becomes less smooth, our confidence in generalizing scores from sparse sample points to surrounding regions must decrease. Applying a finer-grained code may help alleviate this problem, as would visiting more cosets before ranking cells. Regardless of the policy pursued, the algorithm cannot avoid the possibility of finding local minima. We therefore would like to both reduce the likelihood of confining the search to a region of

local minima, and allow the search to escape from such regions when it becomes stuck. The problem can be stated formally as follows: when a coset is evaluated for cell c, we produce a vector of scores S = $\{f(S_0), f(S_1), ..., f(S_{k-1})\}$, where S_i denotes the real-valued point sampled for codeword cw_i in the coset. Define $Sample_{best}$ s.t. $f(Sample_{best}) \leq f(S_i)$; $i \in \{0, 1, ..., k-1\}$. An algorithm which naively assumes that the subcell represented by cw_i is therefore the best cell may find some improved solutions, but is likely to become trapped there. Clearly one measurement is not an accurate assessment of a large (or non-smooth) cell's quality, so we may wish to sample more thoroughly before electing to focus on a subcell. For this reason, we would like a ranking function that does not penalize a certain number of initial evaluations while we are initially exploring a cell. As we take more measures, we build confidence in our assessment of the cell. On the other hand, the search may enter cells which show very little promise, or it may enter a promising cell, but exhaust of all its good solutions and settle at a minimum. For this reason, we would like to impose a penalty on fruitless evaluations. We experimented with measures that applied penalties based on count of both total evaluations, and consecutive fruitless evaluations.

The above discussion suggests another important factor in budgeting our sample point evaluations: smooth regions require fewer evaluations to estimate cell quality, while non-smooth regions require many more evaluations. We would therefore like a way to assess the smoothness of a region, and choose our samples accordingly. We propose a "heterogeneity" metric, which measures the variability of scores found within the cell. In our experiments, we chose the simplest possible measure of heterogeneity: the variance of the scores obtained for all points evaluated within a cell. Cells with low variance give us more confidence in our estimates, and thus can impose higher penalties on the number of evaluations done within a cell. Cells with high-variance, however, may contain both very good solutions as well as very poor solutions. We therefore wish to apply a lesser penalty to evaluations performed in such cells, in an attempt to better gauge the cell's true promise.

3.6. Final Scoring Function

Based on the results of these experiments, we defined our our ranking function to be:

$$R(c) = \frac{f(Best_c) * \max(1, Searched_c - Delay)}{(1 + ln(1 + SampleVariance_c)}.$$

Here, $Best_c$ denotes the real-valued solution in B which produced the best score ever seen in any search of cell c, while $Searched_c$ denotes the total number of times c was selected for search by algorithm. The $Delay$ term is a mechanism for promoting early exploration of new cells; the first $Delay$ searches of a cell will not incur the penalties normally incurred by repeat evaluations. The $Variance_c$ is the variance of in objective function scores over all points evaluated during the most recent search of the cell; variance is not currently measured over the entire history of the cell. As a monotonic function of the heterogeneity of the scores within the cell, the denominator term serves to improve the rankings of more heterogeneous cells, and prevent the search from prematurely rejecting them.

3.7. Algorithm Comparisons

Table 1 compare the performance of the optimal code algorithm to that of several other search algorithms applied to the same problem. Each algorithm is run 10 times to account for variability due to inherent stochasticity.

3.8. The Seven Dimensional ODE Problem

In this problem, we attempt to tune the set of parameters P of a system of ordinary differential equations to obtain an optimal fit to a set of training data. For each equation e in a set of training equations TE, we are given two vectors: $T_e = \{t_0, t_1, ..., t_{e\_\max}\}$ and $D_e = \{d_0, d_1, ..., d_{e\_\max}\}$. Let the function $M(e, t, p)$ represent the value for equation e at time t, obtained by numerically integrating the ODE system with parameter values specified by the candidate solution vector p. Our objective function is then defined as: $f(p) = \sum_{e \in TE} \sum_{i=0}^{|T_e|} (M(e, T_{e,i}, p) - D_{e,i})^2$. The system of equations and the parameter bounds are listed below. For comparison purposes, each optimization method was allowed to perform 100,000 evaluations of the objective function.

3.9. The Seven Parameter ODE System

In the system below, L denotes the pathogen level in the patient, M denotes the macrophage level in the patient, and D stands for the damage or dysfunction observed in the patient.

$$\begin{aligned} \text{Equations:} \quad & L' = k_{pg} \cdot L \cdot (1 - L) - k_{pm} \cdot M \cdot L \,, \\ & M' = (k_{mp} \cdot L + D) \cdot M \cdot (1 - M) - M \,, \end{aligned}$$

Run	OptimalCode	GA(16,6250)	Hillclimber	Random
1	0.001369	0.0344754	0.456643	0.0305824
2	0.002135	0.071979	0.437013	0.0201192
3	0.005527	0.0465479	0.226696	0.0260119
4	0.002664	0.196176	0.418331	0.0135728
5	0.001448	0.335479	0.00387431	0.0183359
6	0.001673	0.0657115	0.316007	0.025964
7	0.001386	0.0792147	0.399875	0.0059441
8	0.001440	0.0602518	0.416516	0.0100706
9	0.005316	0.0714629	0.393411	0.0214453
10	0.005042	0.023839	0.418944	0.00358366
Mean	0.002800	0.09851372	0.348731031	0.017562986
Best	0.001369	0.023839	0.00387431	0.00358366

Table 1: The 7 dimensional ODE system

a0=0.0601	a1=0.5290	a2=0.0007	a3=0.5887	a4=0.9444
a5=0.0622	a6=0.1902	a7=0.3401	a8=0.8010	a9=0.4683
a10=0.8855	a11=0.9299	a13=0.0718	a14=0.5603	a15=0.6151

$$D' = (k_{dma} \cdot (1 + \tanh(\frac{M - t_{ma}}{w_{ma}}))) - k_d \cdot D\,,$$

Initial Conditions: $L(0) = 0.05\,, \quad M(0) = 0.001\,, \quad D(0) = 0.15\,,$

Parameter Bounds: $k_{pg} = [0, 0.1]\,,$

$k_{pm} = [0, 1]\,, \quad k_{mp} = [0, 10]\,, \quad t_{ma} = [0, 0.5]\,,$

$w_{ma} = [0, 0.5]\,, \quad k_{dma} = [0, 0.02]\,, \quad k_d = [0, 0.1]\,.$

3.10. The 15-Parameter Quadratic

The function to be minimized was: $f(p) = \sum_{i=1}^{15}(a_i - p_i)^2$. All a_i and p_i were constrained within [0, 1]. Values for the constants a_i were chosen a priori at random within these intervals and then held fixed throughout the entire battery of tests.

The optimal code algorithm reported here behaves as described previously. The GA columns refer to a genetic algorithm variant, where P is the population size used, and N is the number of generations performed. The steepest descent algorithm sampled 5 points in each dimension of the space (with stochastic noise) and chose the solution whose combined changes produced the greatest score improvement. The random algorithm simply evaluates points at random within the search space.

Run	OptimalCode	GA(2048,44)	Hillclimber	Random
1	0.0832361	1.83559	1.97709e-05	68.379
2	0.217762	1.68968	2.22564e-05	60.54
3	0.0772086	1.68611	1.3165e-05	68.1213
4	0.220271	2.62141	1.1364e-05	66.9669
5	0.540058	1.72501	9.60819e-06	69.3401
6	0.229307	2.22178	1.08077e-05	79.7582
7	0.169243	1.23769	3.66222e-05	55.4995
8	0.0960766	3.02225	1.51147e-05	74.71
9	0.299546	2.87475	5.29135e-06	94.2987
10	0.320636	4.24115	2.10776e-05	72.1723
Mean	0.22533443	2.315542	1.6507804e-05	70.9786
Best	0.0772086	1.23769	5.29135e-06	55.4995

Table 2: The 15-dimensional quadratic

4. Classifying Parameter Sets Yielding Different Regimes

Though minimizing an objective function occurs any time one fits a model to data, one of our main objectives is to predict the correct regime for a given vector of parameters. The first step is to identify originally a set of regimes that the system of ODE's could produce; see also Neugebauer [10]. While several potential regimes may be suspected to exist a priori, not all possible regimes will usually be known. However, each time we evaluate the model at a parameter set (i.e., numerically integrate it) we either classify the resulting function in one of the known regimes or decide to create a new regime. We can thus eventually expand the set of regimes as the computer simulation progresses. Comparison of regimes involves defining a measure that compares two regimes and computes a distance between them. We decided to use a measure that matches only "qualitatively defining features" of the functions in questions, such as notable peaks, tail behaviour, or other qualitatively relevant traits. We shall then say that two functions are in the same regime if they have qualitatively similar features; else they represent different regimes (this measure is quite different for the more usual least squares metric, but it addresses better the qualitative features we seek in the model).

We now describe how we can use the algorithm in the previous section in conjunction with a multinomial logistic distribution to classify any point in the parameter space. The outcomes of the logistic distribution are the r distinct regimes. We now start the algorithm. At each parameter point selected, by integrating the ODE system, we obtain the response function. We compute distances to each of the r regimes and classify that parameter point in one of

them. The code ensures that the parameter space is optimally covered each time we select a new coset of parameter points. After a sufficiently large number of iterations we may notice points in the parameter space, where bifurcations seem to occur. In the neighborhood of such points (should they prove of investigative interest) we can *locally replicate* the search just described. We refer to Seydel [11] for detailed information on issues related to bifurcation.

To classify an arbitrary parameter point θ we use the multinomial logistic as follows: think of the points in the parameter space that we evaluated and classified as *data* for the logistic model. Using maximum likelihood estimation, in the usual way, estimate from this data the coefficients in the logistic function. Use the multinomial logistic model thus created to predict that θ yields a response in regime i with probability p_i (with the p_i summing to 1). The probabilities p_i are estimated from the predictive equations generated by the multinomial logistic model. As is commonly done, we now classify θ in the class corresponding to regime j, where j is defined by $p_j = \max_{1 \leq i \leq r} p_i$.

Acknowledgments

The work is funded under the NIH grant GM67240.

References

[1] R.C. Bone, Immunologic dissonance: a continuing evolution in our understanding of the systemic inflammatory response syndrome (SIRS) and the multiple organ dysfunction syndrome (MODS), *Ann. Intern. Med.*, **125** (1996), 680-687.

[2] T.G. Buchman, J.P. Cobb, A.S. Lapedes, et al, Complex systems analysis: a tool for shock research, *Shock*, **16** (2001), 248-251.

[3] C.C. Chow, et al, Quantitative dynamics of the acute inflammatory response in shock states, *American Journal of Physiology* (2004), To Appear.

[4] G. Clermont, et al, In silico modeling of clinical trials: a method coming of age, *Critical Care Medicine* (2004), To Appear.

[5] H. Kitano, Systems biology: a brief overview, *Science*, **295** (2002), 1662-1664.

[6] F.J. MacWilliams, N.J.A. Sloane, *The Theory of Error Correcting Codes*, North Holland, Amsterdam (1978).

[7] R. Medzhitov, C.J. Janeway, Innate immunity, *N. Engl. J. Med.*, **343** (2000), 338-344.

[8] J.P. Mira, A. Cariou, F. Grall, et al, Association of TNF2, a TNF-a promoter polymorphism, with septic shock susceptibility and mortality. A multicenter study, *JAMA*, **282** (1999), 561-568.

[9] C. Nathan, Points of control in inflammation, *Nature*, **420** (2002), 846-852.

[10] E.A. Neugebauer, C. Willy, S. Sauerland, Complexity and non-linearity in shock research: reductionism or synthesis?, *Shock*, **16** (2001), 252-258.

[11] R. Seydel, *Practical Bifurcation and Stability Analysis: From Equilibrium to Chaos*, Second Edition, Springer Verlag, New York (1994).