BIOINFORMATICS

Vol. 14 no. 10 1998
Pages 869–883

Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation

Pedro Mendes and Douglas B. Kell

Institute of Biological Sciences, University of Wales Aberystwyth, Aberystwyth, Ceredigion SY23 3DD, UK

Received on July 27, 1998; revised on August 31, 1998; accepted on September 4, 1998

Abstract

Motivation: *The simulation of biochemical kinetic systems is a powerful approach that can be used for: (i) checking the consistency of a postulated model with a set of experimental measurements, (ii) answering 'what if?' questions and (iii) exploring possible behaviours of a model. Here we describe a generic approach to combine numerical optimization methods with biochemical kinetic simulations, which is suitable for use in the rational design of improved metabolic pathways with industrial significance (metabolic engineering) and for solving the inverse problem of metabolic pathways, i.e. the estimation of parameters from measured variables.*

Results: *We discuss the suitability of various optimization methods, focusing especially on their ability or otherwise to find global optima. We recommend that a suite of diverse optimization methods should be available in simulation software as no single one performs best for all problems. We describe how we have implemented such a simulation-optimization strategy in the biochemical kinetics simulator Gepasi and present examples of its application.*

Availability: *The new version of Gepasi (3.20), incorporating the methodology described here, is available on the Internet at http://gepasi.dbs.aber.ac.uk/softw/Gepasi.html.*

Contact: *prm@aber.ac.uk*

Introduction

Computers have been used since the 1940s to simulate the kinetics of biochemical reactions. Given a pathway structure and a kinetic scheme, the admissible steady states of that system and the time courses of the reactions can be computed. This is a well-developed field and the reader is referred to other publications that cover the subject in greater depth (Garfinkel *et al.*, 1970; Heinrich *et al.*, 1977; Garfinkel, 1981; Hayashi and Sakamoto, 1986; Hofmeyr, 1986; Mendes and Kell, 1996). Basically, kinetics simulation software calculates the values of the internal metabolite concentrations based on a set of kinetic functions and values for their parameters and the (constant) external metabolite concentrations. The majority of kinetic models are described in terms of coupled differential equations (rather than explicit algebraic functions) and simulators implement the appropriate methods to solve these systems of ordinary differential equations (ODEs). Two distinct types of simulation can be carried out: (i) time courses, in which the values of the variables are determined as a time series; (ii) steady states where the values of the variables are determined for a state in which no metabolite concentration changes. There are several modern programs designed specifically for this purpose (Cornish-Bowden and Hofmeyr, 1991; Mendes, 1993, 1997; Sauro, 1993; Ehlde and Zacchi, 1995) and most include a facility to investigate the behaviour of the model over a range of values of some of its parameters. This allows one effectively to study the dependency of the model's behaviour on its parameters and is therefore a very effective means of forecasting the effects of large parameter perturbations. The ability to predict the effect of large parameter changes is an essential requisite (Westerhoff and Kell, 1996) for the rational design of engineered organisms (metabolic engineering) (Bailey *et al.*, 1990; Cameron and Tong, 1993; Cameron and Chaplen, 1997; Mendes and Kell, 1997), an activity that is becoming attainable experimentally (via rDNA, site-directed mutagenesis, etc.).

The currently existing simulators use one of two strategies to 'scan' the parameter space of a kinetic model: repeated solution of the ODEs at different values of the scanned parameters, or using continuation algorithms (Hocker, 1994). Both these approaches have major limitations in terms of the dimension of the parameter space. Continuation can be carried out for invariant dynamical states (steady states, limit cycles, etc.) and is limited to two parameters at a time, while the computation time for systematic scans by the repeated solution of ODEs grows exponentially with the number of parameters. This 'curse of dimensionality' (Duda and Hart, 1973) means that the systematic process is really applicable only to rather small numbers of parameters (below 10 and preferably less than five; Garey and Johnson, 1979). The question then arises as to whether there are any other procedures one can use when the model has a large number of

parameters. The answer is yes: numerical and combinatorial optimization algorithms can be used for such purposes. Here, we will describe how these can be implemented in a metabolic simulation package in a generic way so as to be able to address a diversity of problems.

Optimization methods have a number of applications in science and engineering. For example, they are routinely used in parameter fitting problems, as a means of designing improved processes or devices, in search problems or as learning algorithms. In enzyme kinetics, some of these methods are already used in combination with simulation for parameter estimation (fitting). Linear or (now more commonly) non-linear least squares regression is used to estimate kinetic constants from measured rates and concentrations, and several computer programs are available for this purpose (Duggleby, 1984; Holzhütter and Colosimo, 1990; Frieden, 1993; Kuzmic, 1996). It is also possible to apply the same technique to estimate the kinetic parameters of several enzymes simultaneously, a procedure already described in 1972 by Curtis and Chance, but still carried out only infrequently. More recently, numerical optimization has been used for the design of pathway models with particular properties (Bray and Lay, 1994; Gilman and Ross, 1995) or for the improvement of fluxes of biotechnological interest (Torres *et al*., 1997). These applications were carried out with software specially written for them. The fitting software packages cited above are only capable of using one optimization method and usually this is one that can find a minimum only in the vicinity of an initial guess (e.g. the Levenberg–Marquardt method; Levenberg, 1944; Marquardt, 1963). Essentially, numerical optimization has not evolved to a stage in which there is a single method that is the best for all applications, and although the idea that a single method would be the best for all problems is an attractive one, it is in fact incorrect, probably due to the great variety of problem domains and their properties (Wolpert and Macready, 1997). Thus, it is highly desirable to be able to apply a series of methods to each optimization problem (including fitting), in order to achieve the best possible solution. The purpose of this article is to describe a way in which this can be implemented in biochemical kinetics software. We also propose that the use of numerical optimization should be extended such that any item of a kinetic model should be able to take part in optimization, not simply the metabolite concentrations, fluxes or functions of the differences between simulated and measured data.

For the purposes of this simulation-optimization methodology, the strategy that is used for the simulation part is not really important, so long as it is capable of taking values of the model parameters as inputs and providing the (calculated) values of the variables as outputs. Simulation based on differential equations, arguably the most popular method, is quite adequate, but so are other methods such as Monte Carlo (Kibby, 1969) or cellular automata (Ermentrout and Edelstein-Keshet, 1993). In the implementation described below, we have used our own simulation package Gepasi (Mendes, 1993, 1997), which is based on differential equations and is written in the C++ language.

Optimization

Optimization problems are concerned with locating optima (maxima or minima) of functions. Finding a maximum of a function $f(\boldsymbol{x})$ is equivalent to finding a minimum of $-f(\boldsymbol{x})$. Thus, we will refer only to minimization hereafter, although the reader should bear in mind that the discussion applies equally to maximization following the simple transformation above. The problem can then be stated in general terms as follows:

> given a real-valued scalar function $f(\boldsymbol{x})$ of n variables $\boldsymbol{x} = (x_1, \ldots, x_n)$, find a minimum of $f(\boldsymbol{x})$ such that $g_i(\boldsymbol{x}) \geq 0$ with $i = 1, \ldots, m$ (inequality constraints) and $h_j(\boldsymbol{x}) = 0$ with $j = 1, \ldots, m$ (equality constraints).

In general, the objective function $f(\boldsymbol{x})$ and the constraints $g_i(\boldsymbol{x})$ and $h_j(\boldsymbol{x})$ are non-linear, although frequently the only constraints are linear boundaries of the form $a_i \leq x_i \leq b_i$ (these actually translate into two separate constraints: $x_i - a_i \geq 0$ and $b_i - x_i \geq 0$), where a_i and b_i are often positive constants. Some problems might be unconstrained, i.e. $m = m' = 0$. Owing to its non-linearity, $f(\boldsymbol{x})$ will often have several minima. For many applications, one is interested in the global minimum (i.e. the one with the lowest value of $f(\boldsymbol{x})$), although sometimes a local minimum may be sufficient. In fact, it is assumed in many cases such that a local minimum then found is 'close' to the global minimum, an assumption that is widely borne out for hard combinatorial optimization problems (so-called NP-complete; see Garey and Johnson, 1979) of this type. On some rare occasions, one may even wish or need to know all the minima. The existence of several minima or otherwise is an important issue in so far as many optimization methods can only find a local minimum, as will be discussed below. The majority of numerical optimization methods do not rely on the explicit form of $f(\boldsymbol{x})$, and only require its value to be evaluated at several points; some methods do not even require this function to be continuous or differentiable and may still work if its values contain noise. In the next section, we will briefly discuss some of the numerical methods that are available to minimize functions; for the moment, it is enough to bear in mind that these methods only need to evaluate f at specific values of the variables (vector $\boldsymbol{x}$).

Because the functions to minimize are not required in an explicit form, the results of kinetic simulations (i.e. solutions of ODEs) are suitable candidates for optimization. These include metabolite concentrations and fluxes, but also any

other indices derived from these, including the various coefficients of metabolic control analysis (Kacser and Burns, 1973; Heinrich and Rapoport, 1974; Fell, 1992, 1996; Heinrich and Schuster, 1996), mass action ratios, stability indices, etc. The ability to find maxima or minima of these entities is an extremely desirable feature of a kinetics simulator as it will allow one to characterize any kinetic model extensively and is of great interest to the biotechnology industry. The field of metabolic engineering (Kell and Westerhoff, 1986; Bailey *et al*., 1990; Cameron and Tong, 1993) is concerned with modifying the properties of metabolic pathways with the aim of enhancing the production of some metabolite of interest. Another area where optimization is quite useful is in parameter estimation; here one has a candidate model and experimental data, and one desires to estimate the values of the model parameters from those data. This could typically involve the construction of a sum-of-squares function of the residuals between the measured and the simulated data, the estimates of the true parameters being those values that minimize the sum-of-squares function. The minimization of this sum-of-squares function can be carried out exactly in the same manner and with the same numerical methods as the design problems mentioned above. This extends the usefulness of the approach to the inverse problems of identifying parameter values from measured variables. In this paper, we will describe a general strategy to implement this combined simulation-optimization approach and will briefly describe how we have implemented it in our own biochemical kinetics simulator Gepasi (Mendes, 1993, 1997).

Before proceeding any further, we would like to clarify some potentially confusing issues of language that arise in this combined approach. This is related to the words 'parameter' and 'variable', which are used in simulation and optimization with perhaps confusing meanings. In modelling and simulation, one distinguishes between parameters and variables, the parameters being entities of a model that are either constant, under our direct control or vary independently; examples of parameters are kinetic constants and time. Variables are entities whose values are entirely determined by the parameters; examples are the internal metabolite concentrations and fluxes (and, of course, any other entities calculated from these). Although in optimization these words have the same meaning, the entities that are parameters and variables are now exactly the opposite: the variables of the simulation are now part of the objective function (the function to be minimized) and the parameters to optimize become variables (because they are varied in the course of the optimization)—this is why these are also known as inverse problems. To avoid confusion, we will always refer to the entities that are parameters in the simulation model as 'parameters' and the variables in the simulation model as 'variables'; when we refer to the model parameters that are being optimized, we call them 'adjustable parameters' to stress the fact that their values are going to be adjusted by the optimization method. We must nevertheless point out that the optimization literature refers to these as 'variables' as they are effectively varied during the course of the optimization. To make matters worse, the numerical optimization methods themselves can be tuned by a few parameters that are only involved with the way the method proceeds (e.g. step sizes, desired accuracies, etc.); we refer to these as 'method parameters'.

Numerical optimization methods

The field of numerical optimization is vast and it is outside the present scope to discuss it at any length. However, it is useful to list the various types of methods and their characteristics, simply to show their suitability for our purpose.

Linear programming methods are very popular due to their ability to handle hundreds of thousands of parameters. However, with the exception of metabolite concentrations or fluxes represented in the S-System formalism (as described in Torres *et al*., 1997), they cannot be applied to biochemical problems due to their requirement of objective function linearity in terms of the adjustable parameters. Non-linear optimization methods are thus to be preferred. Those that converge faster to a local minimum are the gradient descent methods, the most popular being the Newton method (e.g. Fletcher, 1987), although for practical reasons sophisticated variants of this (e.g. Nash, 1984; Byrd *et al*., 1995) are preferred. The Levenberg–Marquardt method (Levenberg, 1944; Marquardt, 1963) is particularly suitable in least squares problems based on sum of squares functions; the method can also be used for general functions (Goldfeld *et al*., 1966). Direct search methods do not calculate derivatives and are also local minimizers, the most successful being those described by Hooke and Jeeves (1961) and Nelder and Mead (1965). Stochastic methods are important when the objective function has several optima and one is interested in a global optimum. The most simple is random start, but most often the ones to use are multistart (Rinnooy Kan and Timmer, 1989) and simulated annealing (Kirkpatrick *et al*., 1983). Evolutionary algorithms, also of stochastic nature, can be used for non-linear optimization (Bäck and Schwefel, 1993) and may indeed find global optima. The most popular is the genetic algorithm (Holland, 1975; Goldberg, 1989), but evolutionary programming (Fogel *et al*., 1966; Fogel, 1995) may be better suited for optimization. Finally, there are also deterministic methods for solving global optimization problems: TRUST (Barhen *et al*., 1997) appears to be much faster than the stochastic based methods, but its use is still not widespread. All these methods can be applied in the methodology we describe here.

Table 1. An application programming interface (API) for the optimization routines. The controlling part of the software uses these function calls to run the optimization

Procedure	Input data	Output data	Description
Init	none	none	carries out initialization of private variables
Version	none	a version number	returns the version number of this routine
IsConstrained	none	TRUE/FALSE	returns TRUE if this method is capable of handling general constraints, FALSE otherwise
IsBounded	none	TRUE/FALSE	returns TRUE if this method is capable of handling adjustable parameter boundary constraints, FALSE otherwise
SetCallback	reference to callback function	none	to set a callback function to the front-end; this will be called by the optimization routine regularly to provide information about the progress of the algorithm
MethodParameterNumber	none	total number of method parameters	returns the number of method parameters that have to be set
MethodParameterName	index of parameter	name of parameter	returns the name of one method parameter
CreateArrays	number of adjustable parameters; number of constraints	none	creates storage to hold values of the adjustable parameters and constraints
ReleaseMemory	none	none	releases all the memory temporarily allocated to carry out the optimization
SetAdjustableParameter	index of parameter; reference to parameter	none	stores a reference to one adjustable parameter
SetConstraint	index of constraint; reference to constraint	none	stores a reference to a constraint
GetAdjustableParameter	index of parameter	value of parameter	returns the candidate value of this adjustable parameter at the minimum
GetObjectiveFunction	none	value of the objective function	returns the candidate value of the objective function at the minimum
Optimize	reference to the function that carries out the simulations	none	executes the optimization algorithm calling the simulation routine (passed as argument) when needed. Note: this procedure can give feedback of its progress by the callback function set with SetCallback
SolveLeastSquares	reference to the function that carries out the simulations	none	stores a reference to the procedure that carries out simulations and then executes the optimization algorithm to solve the problem. Note: this procedure must periodically call the callback function set with SetCallback

The optimization-simulation methodology

All numerical optimization methods operate by carrying out a series of procedures in which the value of the objective function has to be calculated for a set of values of the adjustable parameters. Figure 1 shows the generic algorithm which describes the higher level operation of all numerical optimization methods. Where these methods differ from each other is in the details of steps 3 and 4. This high-level algorithm forms the base of our specification of a general method to implement numerical optimization in biochemical kinetics. Each of the optimization methods is programmed as a module consisting of a set of procedures that implement a well-defined application programming interface (API). In this section, we describe a prototype API for such a purpose. Table 1 lists the functions that compose this API, which after implementation for a specific optimization method form an optimization module.

Before initiating the optimization process itself, one has fully to specify a kinetic model which will be the subject of the optimization. This implies defining the reactions, and setting their kinetic types, initial concentrations for metabolites and values for the kinetic constants. Even those parameters that are going to be subject to the optimization have to be given an initial value as the optimization methods have to start (step 1 in Figure 1) at some point of parameter space. All this is carried out by the user at the front-end level, and exactly in the same way as one would do for a simple simulation (e.g. Mendes, 1997). Then the user has to enter the specific information about the optimization: (i) the model variable

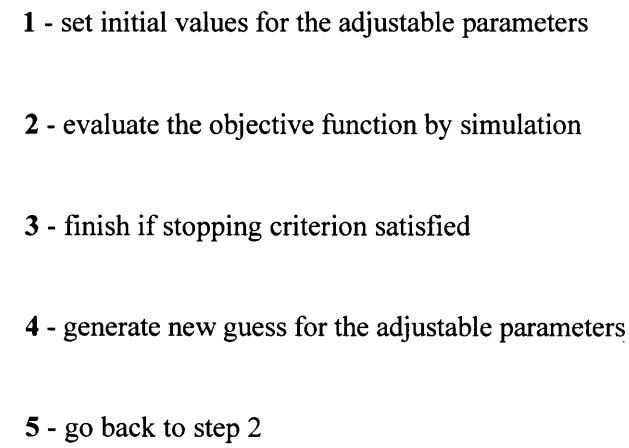

Fig. 1. An overview of the optimization process. Steps 3 and 4 are the only ones whose details differ from method to method.

that will be the objective function, which could be a function of several variables; (ii) whether the process is a minimization or a maximization; (iii) the parameters which will be adjusted in the optimization and optionally upper and lower boundaries for their values; (iv) some constraints that must not be violated in the optimization; (v) which of the available optimization modules to use and values for its own controlling parameters. When the user chooses to start the process, all this information is passed to the relevant optimization module using the appropriate routines of the API (see Table 1). Specifically, *CreateArrays*, once to allocate the appropriate amount of memory for storing intermediate data, iterate *SetAdjustableParameter* to pass a reference to each adjustable parameter and its boundaries, and *SetConstraint* to pass a reference to each constraint, *MethodParameterNumber* to find out how many controlling parameters this module has, and *SetMethodParameter* for each of the module's controlling parameters.

Once all the necessary information has been defined and passed to the optimization module and the user has selected to run the optimization, one of the procedures *Optimize* or *SolveLeastSquares* is called and then the optimization is actually carried out (steps 2–5 in Figure 1). *SolveLeastSquares* is only used if solving a fitting problem and the module has a special way of solving least squares problems (such as the Levenberg–Marquardt method which forms an approximation of the first and second derivatives of the sum of squares from the vector of residuals rather than by finite differences). Both these procedures take as argument a reference to one procedure in the simulation subsystem that calculates the objective function. We note that this procedure in the simulation subsystem is where it is taken into account whether the process is a minimization or a maximization. In the latter case, the procedure returns the negative of the objective function value so that the optimization modules only carry out minimizations. This procedure is also responsible for calculating the sum of squares of the residuals in the case of fitting problems, except if *SolveLeastSquares* is being used.

The operation of the procedures *Optimize* and *SolveLeastSquares* is specific to the method the module implements, and will not be discussed here in any detail. It is worth mentioning that all methods require several evaluations of the objective function, each of these consisting of a simulation. Some modules, notably those implementing gradient descent methods, require estimates of the first and/or second derivatives of the objective function. As the objective functions are never explicit functions in this framework, this is achieved by forward finite differences using the formula:

$$\frac{f(p)}{p} \approx \frac{f(p + \Delta h) - f(p)}{\Delta h}$$

where Δh is a small number relative to the parameter value p. The optimization modules regularly pass information about the progress of the optimization back to the controlling subsystem by a callback function (set previously with *SetCallback*). This is typically called at every iteration of the method and allows the front-end to inform the user about the progress of the optimization, in terms of the current estimate of the objective function at the minimum.

Each module has a different criterion for stopping the iterations (step 3, Figure 1) which is, in some cases, strictly related to the process of optimization itself. A common criterion is based on the amount by which the objective function has reduced in the last iteration. If the change has been smaller than a certain tolerance, the method stops. Some methods also monitor the change in a norm of the adjustable parameter vector against another tolerance level. In the particular case of evolutionary algorithms (see above), it is difficult to specify a stopping criterion due to the probabilistic nature of the algorithms; in this case, it is possible to have the criterion based on the number of generations (i.e. leaving the decision to the user).

Step 4 of Figure 1 is the one that is specific for each different method. Some of these methods may already be available as subroutines and so all that is needed is a way of interfacing them to the API described here. This is what has been done using a series of methods in our implementation of this API in Gepasi, described below (see also Table 2).

Table 2. Optimization modules currently available for Gepasi Version 3.20

Module name	Method	Features[a]	Origin of code	References
Random search	Random search with uniform distribution	B C G S	written in C++ by PM	
Steepest descent	Steepest descent with finite differences gradient	B C L D	written in C++ by PM	
L-BFGS-B	Limited memory BFGS quasi-Newton method	B L D	L-BFGS-B Version 2.1,[b] written in FORTRAN by C.Zhu[c]	Byrd *et al*., 1995
Truncated Newton	Truncated Newton (Lanczos method)	B L D	TN,[b] written in FORTRAN by S.Nash[c]	Nash, 1984
Tensor	Derivative tensor method	L D	TOMS[d] Algorithm 739[b], written in FORTRAN by T.Chow *et al*.[c]	Chow *et al*., 1994
Levenberg–Marquardt	Restricted step Newton method with the Marquardt iteration	B L D	written in C++ by PM	Levenberg, 1944; Marquardt, 1963; Goldfeld *et al*., 1966
NL2SOL	Adaptive non-linear least squares (Gauss–Newton)	B L D	DN2FB[e] written in FORTRAN by D.Gay[c]	Dennis *et al*., 1981
Multistart	Multistart with steepest descent-based local optimization	B C G S	written in C++ by PM	
Hooke and Jeeves	Hooke and Jeeves direct search	B L D	written in C by M.Johnson,[b] adapted by PM	Hooke and Jeeves, 1961
Genetic algorithm	Genetic algorithm with floating-point encoding	B C G S	written in C++ by PM	e.g. Michalewicz, 1994
Evolutionary programming	Meta-evolutionary programming	B C G S	written in C++ by PM	Fogel *et al*., 1992
Simulated annealing	Monte Carlo simulated annealing with exponential cooling schedule	B C G S	written in C++ by PM	Corana *et al*., 1987

[a]B, works with bounds on the adjustable parameters; C, works with constraints on the variables; G, can find global optima; L, can only find local optima; S, stochastic algorithm; D, deterministic algorithm.
[b]Obtained from the mathematical software repository Netlib, http://netlib.bell-labs.com/netlib/master/readme.html
[c]Original FORTRAN code automatically converted to C using *f2c* (Feldman *et al*., 1995).
[d]ACM Transactions on Mathematical Software.
[e]Obtained from the PORT library, available in the Netlib archive (see note b).

When the stopping criterion is satisfied, the optimization module gives control back to the main part of the software. If a parameter fitting problem was the objective of the optimization, then some statistics can be calculated, specifically standard deviations for the fit and for the fitted parameters or even better (Johnson, 1992) confidence intervals. The optimization module is interrogated by the controlling subsystem through calls to the routine *GetAdjustableParameter* for the values of the adjustable parameters and *GetObjectiveFunction* for the value of the objective function at the minimum. In the end, the results are output in an appropriate format, often a textual report file. For parameter fitting problems, it is important to plot the distribution of residuals against the free and dependent variables of the measurements, as this provides a good check for the quality of the fitted parameters (Straume and Johnson, 1992).

As described above, the optimization modules and the main controlling part of the software pass information to each other in both directions. This is required for setting up the procedure and to pass back its results, but also to provide the user with feedback about the progress of the computation. The latter is a very important feature as some of the optimization methods are very slow, especially when the biochemical model and/or the number of adjustable parameters are large. Without this feature, the user would soon suspect the software of malfunction and would be tempted to abort the run even though it was working properly, albeit slowly. Another feature which is also important in a practical sense is that the user should be allowed to halt an optimization routine at any time and hopefully still get some results (i.e. a set of parameter values that produces an objective function value closer to the optimum than the original guess). This way in which this is implemented depends on how the main software is designed and so we have left this undefined in the API (Table 1).

Implementation

The implementation of the API defined in Table 1 is tied up with the computer language used. We have implemented this API in the Gepasi software (Mendes, 1993, 1997) which is written in the C++ language in a Microsoft Windows™ environment. Thus, we wrote all optimization modules in the C++ language, making extensive use of pointers (including pointers to functions), dynamic-link libraries (DLLs) and threads: (i) the optimization routine keeps pointers to the function that carries out one simulation and to the variables that hold the values of the adjustable parameters; (ii) the controller starts the optimization as a new thread, of which it has total control, i.e. it can stop it at any time; (iii) the optimization modules implementing the API defined above were built as dynamic-link libraries. The latter point is important in the sense that it makes the program extensible: whenever we implement a new optimization module, we can distribute that module alone without having to rewrite or recompile any other part of Gepasi. In fact, we are still implementing extra optimization modules that will be released later.

When other computer languages or operating systems are used, perhaps there will be other more appropriate architectures to follow. For example, in the increasingly popular language Java™, the optimization modules would be adequately written as a set of classes implementing a specific 'interface'. The interface would be the equivalent to the DLLs and would follow the API of Table 1, just like the Windows™ DLLs.

Table 2 lists the optimization modules currently available for Gepasi Version 3.20 and the numerical optimization methods they implement.

Applications

There are basically two types of applications in which this simulation-optimization approach is useful. The first is concerned with finding sets of parameter values that result in a maximum or minimum of a model variable or function of model variables. This type of application can be generally referred to as 'design' problems and includes both metabolic engineering and biochemical evolution studies. The second type of application is where one has a number of experimentally determined values of some model variables (or functions derived from them) and intends to find the most probable set of values of the model parameters that would produce the behaviour of the experimental system. We will refer to these as 'parameter estimation' problems. These two applications are different only in the form of objective function used: while in the first only items of the model are used, the second also uses values obtained by experiment (the objective function is the sum of squares of residuals). While in design problems one may be interested in either maximization or minimization, parameter estimation always consists of a minimization. However, the two types of applications share common features in all other aspects and this is exactly why we advocate that biochemical optimization software should deal with both.

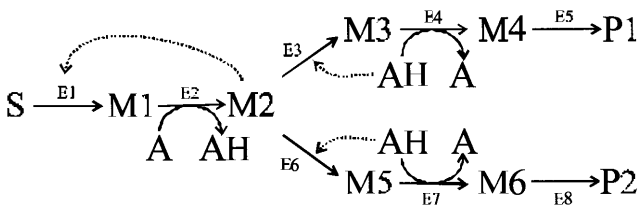

Fig. 2. A branched metabolic pathway with feedback. All steps are chemically reversible, the arrows only represent the positive direction of flux. Steps 1, 3 and 6 follow reversible Hill kinetics with one modifier (Hofmeyr and Cornish-Bowden, 1997), for all three the Hill coefficient for substrate is 4, the forward limiting rate 10, the parameter α is 0.1 (the modifier is an inhibitor) and all other parameters unity. Steps 2, 4 and 7 follow the ordered bi bi mechanism (Cleland, 1963) where the cofactor A/AH is always the first to bind the enzyme and the last to be released, all three have forward and reverse limiting rates of 10, all other parameters equal to unity. Steps 5 and 8 follow reversible Michaelis–Menten kinetics (Haldane, 1930) with forward limiting rate equal to 10 and all Michaelis constants unity. All steps have an equilibrium constant of unity. S, P1 and P2 are external metabolites, their concentrations kept constant at 1, 0.1 and 0.1, respectively. The total amount of the conserved moiety A is 0.1. Units are arbitrary.

As a matter of example we will describe two applications of this combined approach: the first is a design problem, while the second is a parameter estimation problem. They were both carried out with Gepasi Version 3.20 (briefly described above) which implements this methodology.

Rational design for flux enhancement

Figure 2 depicts a hypothetical branched biochemical pathway with a conserved cofactor and feedback. It is similar to the pathway analysed by Cornish-Bowden *et al*. (1995), except for the presence of the cofactor and the kinetics of some of the steps. Steps 2, 4 and 7 follow the ordered bi bi mechanism (Cleland, 1963) (similar to many NADH-dependent dehydrogenases), steps 5 and 8 follow reversible Michaelis–Menten kinetics (Haldane, 1930), and steps 1, 3 and 6 follow reversible Hill kinetics with one modifier (Hofmeyr and Cornish-Bowden, 1997). The latter mechanism allows the modifier (M2 for step 1 and AH for steps 3 and 6) to be an inhibitor or activator, depending on the values of the parameter α. We set arbitrary values for the parameters of the model (see the legend to Figure 2) to build a reference state which we will call wild type for obvious reasons.

The aim of this simulation-optimization exercise is then to find the optimal conditions in order to manipulate the pathway to achieve two independent objectives: (i) maximize the

flux towards P1 without constraints on the flux towards P2; (ii) maximize the flux towards P1 keeping the flux towards P2 unaltered from its original value. This is thus a metabolic engineering application and at this level one is determining what are the manipulations that need to be carried out in a later stage.

The fact that the reversible Hill kinetics equation (Hofmeyr and Cornish-Bowden, 1997) we used allows for the modifier to be either activator or inhibitor is very advantageous for these purposes: the optimization process can simultaneously explore the effect of the modifiers as inhibitors or activators to achieve the optimal states required.

We define the model of Figure 2 in Gepasi setting all the appropriate numerical values of the wild-type state. Then, to achieve aim (i), we set up a maximization of the flux of step 5 (J5, the objective function) with 16 adjustable parameters: the eight forward limiting rates between 0.001 and 1000, $M_{0.5}$ (half-saturating concentrations of modifier) for steps 1, 3 and 6 between 10^{-7} and 10^{7}, the parameter α (see above) of steps 1, 3 and 6 between 10^{-5} and 10^{5}, and the total amount of cofactor moiety ([A] + [AH]) between 10^{-6} and 10. To achieve aim (ii), we added the additional constraint that the flux J8 be limited to the interval $0.01 \leq J8 \leq 0.0118$, which is roughly its wild-type value ± 8%. In both problems, we forced the ratio of forward to reverse limiting rates of all steps to be constant to comply with the Haldane relationship (the Michaelis constants for substrate and product are not changed). This was achieved by defining 'links' (Mendes, 1993) that set the reverse rates according to the value of the forward ones. Not doing so would mean that the thermodynamic properties of the pathway would change in the course of the optimization.

Several optimization modules were applied to solve these two problems and their performance is summarized in Table 3 and 4. Full numerical results are available on the Internet at http://gepasi.dbs.aber.ac.uk/metab/opt/branched.html.

Table 3. Performance of optimization modules on the maximization of J5 (Figure 2) without constraints on J8

Optimization module	J5	J1	J8	Characteristics of solution	No. of simulations
None (wild type)	0.0109	0.0218	0.0109	A + AH = 0.1; A/AH = 0.27 steps 1, 3 and 6: all weak feedback inhibition	1
L-BFGS-B	0.0131	0.0261	0.0131	A + AH = 0.22; A/AH = 0.24 steps 1, 3 and 6: all weak feedback inhibition V_m: no significant changes in any step	672
Levenberg–Marquardt	0.0132	0.0262	0.0130	A + AH = 0.22; A/AH = 0.24 steps 1, 3 and 6: all weak feedback inhibition V_m: unaltered	31 030
Steepest descent	0.0295	0.0331	0.0036	A + AH = 0.24; A/AH = 0.23 steps 1, 3 and 6: all feedback inhibition, stronger on 6 V_m: step 3 up, step 6 down	4454
Simulated annealing	0.2276	0.2276	2×10^{-7}	A + AH = 0.72; A/AH = 0.16 steps 1, 2 and 3 no feedback V_m: steps 1, 2, 3, 4 and 5 at maximum, 6 at minimum, 7 and 8 increased	326 731
Multistart	0.2538	0.2578	0.0040	A + AH = 1.30; A/AH = 0.09 step 1 strong feedback activation, 3 feedback activation, 6 weak feedback activation V_m: steps 1, 2, 3, 4, 5 and 6 up, 7 unaltered, 8 decreased	29 045
Random search	0.2942	0.2942	3×10^{-5}	A + AH = 1.13; A/AH = 0.13 step 1 no feedback, 3 feedback activation, 6 very weak feedback inhibition V_m: steps 1, 2, 3, 4, 5 and 7 increased, 6 and 8 decreased	3 000 000
Truncated Newton	0.3727	0.3727	$<10^{-9}$	A + AH = 1.82; A/AH = 0.09 step 1 weak feedback inhibition, 3 strong feedback activation, 6 strong feedback inhibition V_m: steps 1, 2, 3, 4 and 5 close to maximum, 7 and 8 unaltered, 6 at minimum	27 041
Evolutionary programming	0.3771	0.3771	3×10^{-8}	A + AH = 1.84; A/AH = 0.09 step 1 feedback activation, 3 feedback activation, 6 weak feedback activation V_m: steps 1,2, 3, 4 and 5 at maximum, 7 and 8 5-fold increase, 6 at minimum	45 150
Genetic algorithm	0.3771	0.3771	$<10^{-9}$	A + AH = 1.80; A/AH = 0.09 step 1 weak feedback activation, 3 very strong feedback activation, 6 very strong feedback inhibition V_m: steps 1, 2, 3, 4 and 5 at maximum, 6, 7 and 8 decreased	25 500

Table 4. Performance of optimization modules on the maximization of J5 (Figure 2) constrained to 0.01 ≤ J8 ≤ 0.0118

Optimization module	J5	J1	J8	Characteristics of solution	No. of simulations
None (wild type)	0.0109	0.0218	0.0109	A + AH = 0.1; A/AH = 0.27 steps 1, 3 and 6: all weak feedback inhibition	1
Random search	0.0723	0.0824	0.0100	A + AH = 2.69; A/AH = 0.06 step 1 feedback inhibition, 3 strong feedback activation, 6 feedback activation V_m: steps 1, 2, 6 and 7 increased, 3, 4 and 7 no significant changes, 8 decreased	100 000
Simulated annealing	0.2218	0.2321	0.0103	A + AH = 0.73; A/AH = 0.16 steps 1, 2 and 3 no feedback V_m: steps 2, 3, 4, 5 and 7 at maximum, 1, 6 and 8 increased	95 434
Genetic algorithm	0.2918	0.3018	0.0100	A + AH = 1.06; A/AH = 0.12 step 1 feedback activation, 3 very strong feedback activation, 6 no feedback V_m: steps 1, 2, 4, 5, 6, 7 and 8 increased, 3 no significant change	101 100
Evolutionary programming	0.3740	0.3840	0.0100	A + AH = 1.83; A/AH = 0.09 steps 1 and 6 weak feedback activation, 3 feedback activation V_m: step 6 increased 10-fold, all others at maximum	100 100

For the problem without constraints, the highest value of J5 obtained was 0.3771, a 30-fold increase over the 'wild-type' value. This solution was obtained with the genetic algorithm (GA) and evolutionary programming (EP) methods. We note that even though the value of J5 is the same in both cases, the two solutions are not the same: the feedback interaction at step 6 is a very strong inhibition in the GA solution while it is a weak activation in the EP solution. Indeed, the second-best solution obtained with the truncated Newton method (with J5 just slightly smaller than in the best case) was obtained with yet another combination of feedback interactions as in this case step 1 is inhibited rather than activated by M2. Being able to have obtained these three solutions is already an advantage: the metabolic engineer can choose the one that is easier to produce in practice. Interestingly, this result possibly reflects the natural (as opposed to computational) phenomenon of divergent evolution.

Clearly, not all methods were capable of solutions as good as the three already mentioned. Indeed, two methods (the quasi-Newton methods L-BFGS-B and Levenberg–Marquardt) barely increased the value J5 at all. The former seems to have converged prematurely to a region of parameter space where J5 is flat (and so these gradient methods could not find the 'downhill direction').

A major factor in obtaining high flux seems to be the total amount of cofactor, which is positively correlated with J5. The cofactor ratio has an even stronger (though negative) correlation with J5, but unlike the total amount this cannot be identified with a cause, as it is a variable of the model just like J5. What this means is that, in a state of maximal J5, this ratio is lower than otherwise. Based on this and other unpublished simulation results, we suspect that for real organisms it may well be beneficial to consider the pathways producing and degrading cofactors as these are the ones that will effectively determine the total amount of the cofactor moiety.

In the second optimization problem, we required the competing flux J8 to be within 8% of its wild-type value. Adding this constraint meant that we could now apply only four optimization modules, as the others in our implementation have not yet been made to work with constraints on variables. The best results were again obtained with evolutionary algorithms, but this time EP performed better than the GA. We are surprised that it was possible to obtain a solution with a value of J5 as high as 37-fold greater than in the wild type. This was achieved without decreasing the limiting rate of any enzyme in the competing branch (in the previous unconstrained problem at least one of these was reduced in all solutions). No other features in this solution are strikingly different from the best solutions of the unconstrained case. This stresses the importance of applying quantitative methods, and indeed reinforces our view that the models chosen for optimization purposes must be capable of reasonable extrapolation abilities. Similarly, the real quality of the simulation-optimization method in the context of metabolic engineering can be verified only after comparison with the performance of the engineered organisms.

Parameter estimation

In this example, we want to estimate a number of rate constants of the mechanism of irreversible inhibition of HIV proteinase (Figure 3; see also Kuzmic, 1996). Data from five time courses at four different inhibitor concentrations measured fluorimetrically were used for this analysis (further details can be found at http://gepasi.dbs.aber.ac.uk/metab/opt/hivfit.html). We defined the reaction mechanism

$$
\begin{aligned}
M+M &\rightleftharpoons E & k_{11}\ k_{12}\\
S+E &\rightleftharpoons ES & k_{21}\ k_{22}\\
ES &\rightarrow E+P & k_3\\
E+P &\rightleftharpoons EP & k_{41}\ k_{42}\\
E+I &\rightleftharpoons EI & k_{51}\ k_{52}\\
EI &\rightarrow EJ & k_6
\end{aligned}
$$

Fig. 3. Mechanism of irreversible inhibition of HIV proteinase. The enzyme is only active in a dimer form, the product is a competitive inhibitor for the substrate and the inhibitor is irreversible. For more details, see Kuzmic (1996).

of Figure 3 in Gepasi and set the parameters to initial guess values as in Kuzmic (1996). The software was set to fit the rate constants k_{22}, k_3, k_{42}, k_{52} and k_6 to the experimental data. In this fit, it was also assumed that there is a certain degree of uncertainty (± 50%) in the value of the initial concentrations of substrate and enzyme, and that the offset (baseline) of the fluorimeter was not exactly zero. Therefore, there are a total of 20 adjustable parameters: the five rate constants, five initial concentrations of enzyme, five initial concentrations of substrate and five offset values (given that there are five time course curves).

We fit the model to the experimental data using a series of optimization methods. The best solution was obtained with simulated annealing with a very good quality of fit, as judged by the lack of correlation between the residuals and time (Figure 4). The next best solution was obtained with the Levenberg–Marquardt method and despite the sum of squares being only some 1% worse than that obtained with simulated annealing, some parameters have considerably different values. Indeed, some of them have a better standard deviation (see Table 5). We do not know whether both solutions are two different minima of the least squares function or if this function is very flat in this region of parameter space. A significant aspect is that the solution obtained with simulated annealing took considerably longer to obtain than the second best (roughly 3 million simulations against 4000). This is a severe disadvantage and one would be tempted to dismiss the usage of this technique and favour the method of Levenberg–Marquardt on these grounds. The latter is guaranteed to converge to the global minimum only if started in its vicinity, so given a good initial guess for the parameters this method is definitely the best choice. However, if the initial guess is poor (and unfortunately there is no way of knowing this *a priori*), then Levenberg–Marquardt will not converge to the global

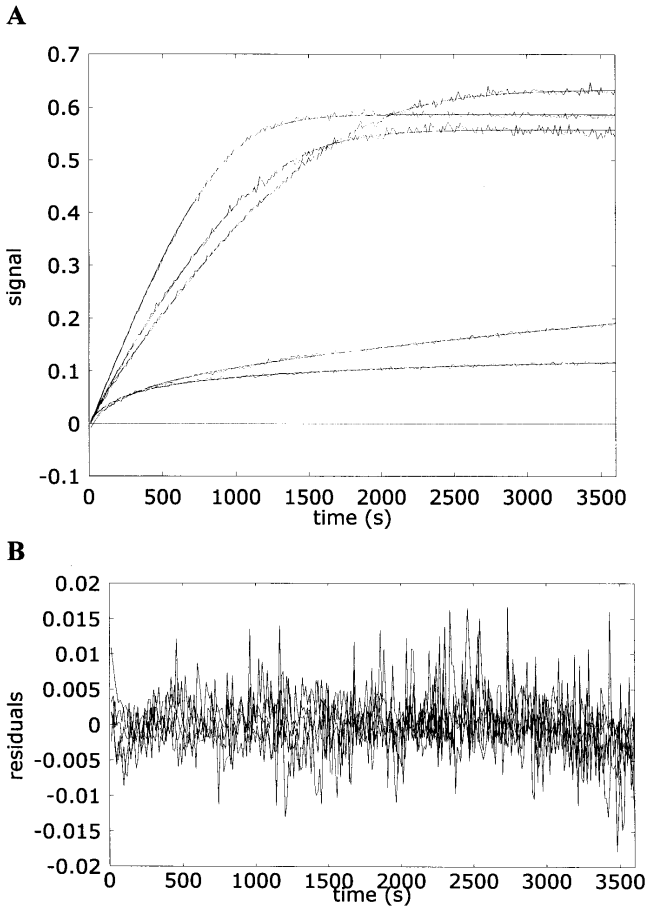

Fig. 4. Least squares fit of progress curves of HIV proteinase in the presence of an irreversible inhibitor. Results of the best fit obtained with simulated annealing. (**A**) Simulated (smooth curves) and experimental data (noisy curves). (**B**) Residuals as a function of time.

minimum, but simulated annealing will, and possibly the evolutionary algorithms will too. We think that this subject deserves more detailed study, although it would not be appropriate to do so here and at all events the implementation that we described here is appropriate for such a study.

Discussion

We have described an approach for combining biochemical kinetics simulations with numerical optimization methods. We illustrated its usefulness with two examples, but, of course, these do not remotely cover the whole universe of applications. We therefore discuss several other types of problems which may be solved (or at least their solution aided) by this strategy. As these problems are of two distinct classes, optimization and fitting, we shall discuss them separately, but we remind the reader that the way in which software is implemented to solve them is essentially the same.

Table 5. Results of least squares fits of HIV proteinase progress curves in the presence of an irreversible inhibitor. Further details are available at http://gepasi.dbs.aber.ac.uk/metab/opt/hivfit.html

Parameter	Simulated annealing[a] (sum of squares = 0.0211)	Levenberg–Marquardt[a] (sum of squares = 0.0213)
k_{22}	201.1 (± 135.7)	180.6 (± 25.70)
k_3	7.352 (± 0.6785)	10.39 (± 1.020)
k_{42}	1171 (± 1070)	1072 (± 202.5)
k_{52}	13 140 (± 26870)	2.821×10^{-16} ($\pm 1.13\times10^{-15}$)
k_6	30 000 (± 89780)	2.486×10^{-16} ($\pm 2.585\times10^{-15}$)
$[S]_i$ (curve 1)	24.79 (± 0.03285)	24.76 (± 0.03321)
$[S]_i$ (curve 2)	23.43 (± 0.03882)	23.56 (± 0.04232)
$[S]_i$ (curve 3)	26.79 (± 0.05742)	27.09 (± 0.06503)
$[S]_i$ (curve 4)	32.10 (± 2.103)	28.17 (± 1.365)
$[S]_i$ (curve 5)	26.81 (± 1.773)	23.51 (± 1.159)
$[E]_i$ (curve 1)	0.004389 (± 0.0003404)	0.003068 (± 0.0002724)
$[E]_i$ (curve 2)	0.004537 (± 0.0002353)	0.003622 (± 0.0001862)
$[E]_i$ (curve 3)	0.005470 (± 0.0001906)	0.004728 (± 0.0001543)
$[E]_i$ (curve 4)	0.004175 (± 0.00001338)	0.004135 (± 0.00001119)
$[E]_i$ (curve 5)	0.003971 (± 0.00009885)	0.003994 (± 0.000004843)
offset (curve 1)	–0.008013 (± 0.0007571)	–0.007377 (± 0.0007672)
offset (curve 2)	–0.003911 (± 0.0008744)	–0.007180 (± 0.0009906)
offset (curve 3)	–0.008962 (± 0.001338)	–0.01605 (± 0.001527)
offset (curve 4)	–0.01600 (± 0.001819)	–0.02281 (± 0.002130)
offset (curve 5)	–0.003789 (± 0.001613)	–0.009926 (± 0.001862)

[a]Parameter value and standard deviation.

Design problems are common in biotechnology (Kell and Westerhoff, 1986; Cornish-Bowden *et al*., 1995; Westerhoff and Kell, 1996), where one normally seeks to maximize a flux, a yield or the concentration of an interesting product, and to minimize the concentration of undesired compounds, or a combination of several of these things. Here it is implied that we have the technology to carry out substantial changes in the pathways, be it in the amount of each enzyme present or indeed in altering the properties of the enzymes. This activity has become known as metabolic engineering (Bailey *et al*., 1990; Cameron and Tong, 1993; Mendes and Kell, 1997). To be able to change pathways in such a way, we will need to apply recombinant DNA (Skatrud *et al*., 1989) and traditional (Ulmer, 1983), active site-based (Fersht, 1985) or more advanced (Kuchner and Arnold, 1997) protein engineering technologies (forced evolution). However, the existence of appropriate technology is not sufficient on its own, since there are many ways in which to change a pathway, but, in general, only a very small proportion of these will result in the desired effect. What has been in high demand in the biotechnology field (Kell and Westerhoff, 1986; Cornish-Bowden *et al*., 1995; Westerhoff and Kell, 1996; Mendes and Kell, 1997) are rational methods of designing the improved pathway. It is exactly at the design stage that the simulation-optimization methodology proposed here can be useful for metabolic engineering.

The first step is to build a kinetic model of the pathway to optimize that has reasonable extrapolation qualities (as the optimization process is most frequently an extrapolation). Thus, we stress that models should be as complete as possible; it is important to keep assumptions to a minimal level and especially not to ignore metabolites of the pathway by assuming them constant (as, for example, the pair NAD^+ and NADH in Galazzo and Bailey (1990)). Doing so implies that results outside the restricted conditions used to build the model will be meaningless. At times, it may be necessary to adopt phenomenological rather than mechanistic descriptions. While the latter are preferred due to their better extrapolation ability, the former are not unreasonable as long as they have been constructed with a sufficiently large set of different experimental conditions and are able to explain the whole range of behaviour observed (Kell and Sonnleitner, 1995).

Once a good kinetic model is available, it is then necessary (i) to construct an objective function reflecting the requirements for the engineered organism and (ii) to determine with which parameters and within what boundaries the manipulations can be made. Then the kinetic model is used in a simulator coupled with optimization methods so as to obtain the values of the parameters at the desired optimum. As we have pointed out previously, the fact that objective functions are usually non-linear in terms of the model parameters, and that there may be a large number of these, means that probably there are several local optima far from the global one. To find the global optimum, one may need to use the slower probabilistic methods rather than the faster gradient descent ones. This is a strong argument for including several optimization methods in software of this kind. Only in that way can users 'experiment' with several methods applied to the same problem. We have found in our own experience with Gepasi that no single method is the best for all problems, a conclusion established in the field of combinatorial optimization (e.g. Wolpert and Macready, 1997).

Although metabolic engineering is an obvious application of this approach, we think that other areas, perhaps more theoretical, may also benefit from it. A few groups, notably that of Heinrich, have indeed applied analytical optimization methods (e.g. Heinrich *et al*., 1987, 1997; Schuster and Heinrich, 1987, 1991; Savinell and Palsson, 1992; Klipp and Heinrich, 1994) to several pathway schemes to investigate the conditions for maximal flux, minimal concentrations, and a series of other criteria. The idea is that these optimal states are ultimately targets of evolution. Even setting aside evolutionary arguments, it is always very useful to find the limits within which a certain pathway may behave, an activity of theoretical biochemistry which we like to call 'exploratory modelling'. Alone, this is useful to increase our knowledge of basic biochemical behaviour and it appears that it may become very important in the analysis of whole-organism behaviour. As a result of the already fully sequenced genomes, there is an ongoing activity (Karp *et al*., 1996; Tatusov *et al*., 1996; Selkov *et al*., 1997; Bono *et al*., 1998) of reconstructing the metabolic pathways of these organisms based on all their (identified) enzymes. The problem here is that the kinetic parameters of the large majority of the enzymes of these organisms are unknown. This approach could be used to reveal the limits of such whole-organism 'superpathways' (though high-performance implementations and fast hardware are probably required). While the analytical approach is better than numerical solutions due to the approximate character of the latter, the domain of problems that are solvable analytically is unfortunately rather small. Thus, the numerical approach proposed here is an important complement to the analytical one, and has a place in theoretical biochemistry too.

Parameter estimation is a well-established area of biochemical kinetics and enzyme kinetic parameters are routinely estimated in many laboratories. Non-linear least squares fitting is a standard technique for the estimation of enzyme kinetic parameters (Cleland, 1979; Duggleby, 1985; Beechem, 1992; Johnson, 1992; Johnson and Faunt, 1992; Kuzmic, 1996) and it could appear that in this aspect our proposal contains no novelty. In reality, though, non-linear least squares fitting in enzyme kinetics has been almost exclusively carried out with gradient descent methods (mainly the Levenberg–Marquardt method), we would argue that for many complex enzyme mechanisms there are benefits in being able to select global optimization methods because the error hyper-surfaces (the sum-of-squares functions) often have several minima (e.g. Markus *et al*., 1980; Esposito and Floudas, 1998). If this is the case, gradient descent methods can provide very poor estimates of the parameter values and may be simply wrong (rather than just inaccurate). Furthermore, most commercial packages for enzyme kinetics do not use the differential equations of the model, but rather some form of integrated equations that are, in general, only approximate and are tainted with unnecessary assumptions. The approach we suggest here can use both methods and it is up to the user to describe the kinetics of the reactions using integrated equations (such as the Michaelis–Menten equation) or by considering the detailed enzyme mechanism, in which case the solution will present the values of the elementary rate constants. Additionally, the fitting can be carried out with time course or steady-state data. We know of only a few computer programs that follow this approach for enzyme kinetic data fitting: SIMFIT (Holzhütter and Colosimo, 1990), FITSIM (Frieden, 1993) and DynaFit (Kuzmic, 1996), but all are limited to using the Levenberg–Marquardt method for the minimization of the sum of squares. Kuzmic (1996) pointed out that in order to obtain good fits, the software needs to consider that concentrations of the various substances in the experiment contain some degree of error and so these initial concentrations should also be included as adjustable parameters. Our implementation follows this recommendation and so Gepasi allows the user to define any initial concentration as an adjustable parameter. Rather than being limited to fitting the parameters of one single reaction, we argue that it is now becoming very important to consider systems of reactions as required by *in situ* and *in vivo* experiments (Mendes *et al*., 1995). In these experiments, one is in general not able to 'silence' competing or side reactions and therefore the numerical model must include them, this also goes towards a 'global analysis' (Beechem, 1992). With the ability to carry out experiments in increasingly complex systems, there is now a greater demand for matching software capable of analysing the complex data sets generated. The approach we have described here offers a significant step in this direction

by defining a flexible framework to incorporate sophisticated optimization methods and deal with metabolic models of arbitrary complexity.

In conclusion, we hope that the methodology that we describe here, and indeed our own implementation of it, will be useful for the theoretical study of pathway kinetics, the rational design of improved pathways in metabolic engineering and for enzyme kinetic parameter estimation, including *in situ* and *in vivo* set-ups. As we enter the post-genomic era, emerging techniques of functional analysis (Lockhart *et al*., 1996; Winston and Fitzgerald, 1997; Wodicka *et al*., 1997; Ducret *et al*., 1998) mean that soon large amounts of gene expression kinetic data (transcriptome, proteome and metabolome; Oliver *et al*., 1988) will become available. The approach described here will be very important in constructing and analysing kinetic models of gene expression with which such data may be compared. This is currently the only way (Brenner, 1997) to rationalize all the non-linear interactions occurring in such complex metabolic systems. The possibility of applying several optimization methods to obtain the best possible solution is essential for the proper workings of this methodology.

Acknowledgements

We thank Jacky Snoep (Free University of Amsterdam) for testing beta versions of Gepasi and Petr Kuzmic (Biokin) for discussions about the Levenberg–Marquardt method. Kinetic data of irreversible inhibition of HIV proteinase were kindly supplied by Drs Sergei Gulnik, Leonid Suvorov and John W.Erickson (SAIC Frederick, NCI-FCRDC). We are grateful to the Engineering and Biological Systems Committee of BBSRC for financial support.

References

Bäck,T. and Schwefel,H.-P. (1993) An overview of evolutionary algorithms for parameter optimization. *Evol. Comput*., **1**, 1–23.

Bailey,J.E., Birnbaum,S., Galazzo,J.L., Khosla,C. and Shanks,J.V. (1990) Strategies and challenges in metabolic engineering. *Ann. NY Acad. Sci*., **589**, 1–15.

Barhen,J., Protopopescu,V. and Reister,D. (1997) TRUST: a deterministic algorithm for global optimisation. *Science*, **276**, 1094–1097.

Beechem,J.M. (1992) Global analysis of biochemical and biophysical data. *Methods Enzymol*., **210**, 37–54.

Bono,H., Ogata,H., Goto,S. and Kanehisa,M. (1998) Reconstruction of amino acid biosynthesis pathways from the complete genome sequence. *Genome Res*., **8**, 203–210.

Bray,D. and Lay,S. (1994) Computer simulated evolution of a network of cell-signaling molecules. *Biophys. J*., **66**, 972–977.

Brenner,S. (1997) *Loose Ends*. Current Biology, London.

Byrd,R.H., Lu,P., Nocedal,J. and Zhu,C. (1995) A limited memory algorithm for bound constrained optimisation. *SIAM J. Sci. Comput*., **16**, 1190–1208.

Cameron,D.C. and Chaplen,F.W.R. (1997) Developments in metabolic engineering. *Curr. Opin. Biotechnol*., **8**, 175–180.

Cameron,D.C. and Tong,I. (1993) Cellular and metabolic engineering: an overview. *Appl. Biochem. Biotechnol*., **38**, 105–140.

Chow,T., Eskow,E. and Schanbel,R. (1994) Algorithm 739: A software package for unconstrained optimization using tensor methods. *ACM Trans. Math. Softw*., **20**, 518–530.

Cleland,W.W. (1963) The kinetics of enzyme-catalyzed reactions with two or more substrates or products. I. Nomenclature and rate equations. *Biochim. Biophys. Acta*, **67**, 104–137.

Cleland,W.W. (1979) Statistical analysis of enzyme kinetic data. *Methods Enzymol*., **63**, 103–138.

Corana,A., Marchesi,M., Martini,C. and Ridella,S. (1987) Minimizing multimodal functions of continuous variables with the 'simulated annealing' algorithm. *ACM Trans. Math. Softw*., **13**, 262–280.

Cornish-Bowden,A. and Hofmeyr,J.H. (1991) METAMODEL—A program for modeling and control analysis of metabolic pathways on the IBM pc and compatibles. *Comput. Applic. Biosci*., **7**, 89–93.

Cornish-Bowden,A., Hofmeyr,J.-H.S. and Cárdenas,M.L. (1995) Strategies for manipulating metabolic fluxes in biotechnology. *Bioorg. Chem*., **23**, 439–449.

Curtis,A.R. and Chance,E.M. (1972) Numerical methods for simulation and optimization. In Hemker,H.C. and Hess,B. (eds), *Analysis and Simulation of Biochemical Systems*. North-Holland Publishing Co., Amsterdam, pp. 39–57.

Dennis,J.E., Gay,D.M. and Welsch,R.E. (1981) An adaptive nonlinear least-squares algorithm. *ACM Trans. Math. Softw*., **7**, 348–368.

Ducret,A., van Oostveen,I., Eng,J.K., Yates,J.R. and Aebersold,R. (1998) High throughput protein characterization by automated reverse-phase chromatography electrospray tandem mass spectrometry. *Protein Sci*., **7**, 706–719.

Duda,R.O. and Hart,P.E. (1973) *Pattern Classification and Scene Analysis*. John Wiley, London.

Duggleby,R.G. (1984) Regression analysis of nonlinear Arrhenius plots: an empirical model and a computer program. *Comput. Biol. Med*., **14**, 447–445.

Duggleby,R.G. (1985) Estimation of the initial velocity of enzyme-catalysed reactions by non-linear regression analysis of progress curves. *Biochem. J*., **228**, 55–60.

Ehlde,M. and Zacchi,G. (1995) MIST: a user-friendly metabolic simulator. *Comput. Applic. Biosci*., **11**, 201–207.

Ermentrout,G.B. and Edelstein-Keshet,L. (1993) Cellular automata approaches to biological modeling. *J. Theor. Biol*., **160**, 97–133.

Esposito,W.R. and Floudas,C.A. (1998) Global optimization in parameter estimation of nonlinear algebraic models via the error-in-variables approach. *Ind. Eng. Chem. Res*., **37**, 1841–1858.

Feldman,S.I., Gay,D.M., Maimone,M.W. and Schryer,N.L. (1995) A Fortran-to-C converter. Computing Science Technical Report 149, AT&T Bell Laboratories.

Fell,D.A. (1992) Metabolic control analysis—a survey of its theoretical and experimental development. *Biochem. J*., **286**, 313–330.

Fell,D.A. (1996) *Understanding the Control of Metabolism*. Portland Press, London.

Fersht,A. (1985) *Enzyme Structure and Mechanism*, 2nd edn. Freeman, San Francisco, CA.

Fletcher,R. (1987) *Practical Methods of Optimization*, 2nd edn. John Wiley and Sons, Chichester.

Fogel,D.B. (1995) *Evolutionary Programming: Toward A New Philosophy of Machine Intelligence*. IEEE, Piscataway.

Fogel,D.B., Fogel,L.J. and Atmar,J.W. (1992) Meta-evolutionary programming. In Chen,R.R. (ed.), *25th Asilomar Conference on Signals, Systems and Computers*. IEEE Computer Society, Asilomar, pp. 540–545.

Fogel,L.J., Owens,A.J. and Walsh,M.J. (1966) *Artificial Intelligence Through Simulated Evolution*. Wiley, New York.

Frieden,C. (1993) Numerical integration of rate equations by computer. *Trends Biochem. Sci*., **18**, 58–60.

Galazzo,J.L. and Bailey,J.E. (1990) Fermentation pathway kinetics and metabolic flux control in suspended and immobilized *Saccharomyces cerevisae*. *Enzyme Microb. Technol*., **12**, 162–172.

Garey,M. and Johnson,D. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

Garfinkel,D. (1981) Computer modeling of metabolic pathways. *Trends Biochem. Sci*., **6**, 69–71.

Garfinkel,D., Garfinkel,L., Pring,M., Green,S.B. and Chance,B. (1970) Computer applications to biochemical kinetics. *Annu. Rev. Biochem*., **39**, 473–498.

Gilman,A. and Ross,J. (1995) Genetic-algorithm selection of a regulatory structure that directs flux in a simple metabolic model. *Biophys. J*., **69**, 1321–1333.

Goldberg,D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA.

Goldfeld,S.M., Quant,R.E. and Trotter,H.F. (1966) Maximisation by quadratic hill-climbing. *Econometrica*, **34**, 541–555.

Haldane,J.B.S. (1930) *The Enzymes*. Longmans Green, London.

Hayashi,K. and Sakamoto,N. (1986) In *Dynamic Analysis of Enzyme Systems. An Introduction*. Springer-Verlag, Berlin.

Heinrich,R. and Rapoport,T.A. (1974) A linear steady-state treatment of enzymatic chains. General properties, control and effector strength. *Eur. J. Biochem*., **42**, 89–95.

Heinrich,R. and Schuster,S. (1996) *The Regulation of Cellular Systems*. Chapman and Hall, New York.

Heinrich,R., Rapoport,S.M. and Rapoport,T.A. (1977) Metabolic regulation and mathematical models. *Prog. Biophys. Mol. Biol*., **32**, 1–82.

Heinrich,R., Holzhutter,H.G. and Schuster,S. (1987) A theoretical approach to the evolution and structural design of enzymatic networks. Linear enzymatic chains, branched pathways and glycolysis of erythrocytes. *Bull. Math. Biol*., **49**, 539–595.

Heinrich,R., Montero,F., Klipp,E., Waddell,T.G. and Meléndez-Hevia,E. (1997) Theoretical approaches to the evolutionary optimization of glycolysis: thermodynamic and kinetic constraints. *Eur. J. Biochem*., **243**, 191–201.

Hocker,C.G. (1994) Applying bifurcation theory to enzyme kinetics. *Methods Enzymol*., **240**, 781–816.

Hofmeyr,J.H.S. (1986) Steady-state modeling of metabolic pathways. A guide for the prospective simulator. *Comput. Applic. Biosci*., **2**, 5–11.

Hofmeyr,J.-H.S. and Cornish-Bowden,A. (1997) The reversible Hill equation: how to incorporate cooperative enzymes into metabolic models. *Comput. Applic. Biosci*., **13**, 377–385.

Holland,J.H. (1975) *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI.

Holzhütter,H.G. and Colosimo,A. (1990) SIMFIT: a microcomputer software-toolkit for modelistic studies in biochemistry. *Comput. Applic. Biosci*., **6**, 23–28.

Hooke,R. and Jeeves,T.A. (1961) 'Direct search' solution of numerical and statistical problems. *J. Assoc. Comput. Mach*., **8**, 212–229.

Johnson,M.L. (1992) Why, when and how biochemists should use least squares. *Anal. Biochem*., **206**, 215–225.

Johnson,M.L. and Faunt,L.M. (1992) Parameter estimation by least-squares methods. *Methods Enzymol*., **210**, 1–37.

Kacser,H. and Burns,J.A. (1973) The control of flux. *Symp. Soc. Exp. Biol*., **27**, 65–104.

Karp,P.D., Ouzounis,C. and Paley,S. (1996) HinCyc: a knowledge base of the complete genome and metabolic pathways of *H. influenzae*. In States,D.J., Agarwal,P., Gaasterland,T., Hunter,L. and Smith,R. (eds), *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, Menlo Park, CA, pp. 116–124.

Kell,D.B. and Sonnleitner,B. (1995) GMP—Good Modelling Practice: an essential component of good manufacturing practice. *Trends Biotechnol*., **13**, 481–492.

Kell,D.B. and Westerhoff,H.V. (1986) Towards a rational approach to the optimization of flux in microbial biotransformations. *Trends Biotechnol*., **4**, 137–142.

Kibby,M.R. (1969) Stochastic method for the simulation of biochemical systems on a digital computer. *Nature*, **222**, 298–299.

Kirkpatrick,S., Gelatt,C.D. and Vecchi,M.P. (1983) Optimization by simulated annealing. *Science*, **220**, 671–680.

Klipp,E. and Heinrich,R. (1994) Evolutionary optimization of enzyme kinetic parameters: effect of constraints. *J. Theor. Biol*., **171**, 309–323.

Kuchner,O. and Arnold,F.H. (1997) Directed evolution of enzyme catalysts. *Trends Biotechnol*., **15**, 523–530.

Kuzmic,P. (1996) Program DYNAFIT for the analysis of enzyme kinetic data: application to HIV proteinase. *Anal. Biochem*., **237**, 260–273.

Levenberg,K. (1944) A method for the solution of certain nonlinear problems in least squares. *Q. Appl. Math*., **2**, 164–168.

Lockhart,D.J. *et al*. (1996) Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature Biotechnol*., **14**, 1675–1680.

Markus,M., Plesser,T., Boiteux,A., Hess,B. and Malcovati,M. (1980) Analysis of progress curves. Rate law of pyruvate kinase type I from *Escherichia coli*. *Biochem. J*., **189**, 421–433.

Marquardt,D.W. (1963) An algorithm for least squares estimation of nonlinear parameters. *SIAM J*., **11**, 431–441.

Mendes,P. (1993) GEPASI: a software package for modelling the dynamics, steady states and control of biochemical and other systems. *Comput. Applic. Biosci*., **9**, 563–571.

Mendes,P. (1997) Biochemistry by numbers: simulation of biochemical pathways with Gepasi 3. *Trends Biochem. Sci*., **22**, 361–363.

Mendes,P. and Kell,D.B. (1996) Computer simulation of biochemical kinetics. In Westerhoff,H.V., Snoep,J.L., Sluse,F.E., Wijker,J.E. and Kholodenko,B.N. (eds), *BioThermoKinetics of the Living Cell*. BioThermoKinetics Press, Amsterdam, pp. 254–257.

Mendes,P. and Kell,D.B. (1997) Making cells work—metabolic engineering for everyone. *Trends Biotechnol*., **15**, 6–7.

Mendes,P., Kell,D.B. and Welch,G.R. (1995) Metabolic channelling in organized enzyme systems: experiments and models. *Adv. Mol. Cell. Biol*., **12**, 1–19.

Michalewicz,Z. (1994) *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edn. Springer-Verlag, Berlin.

Nash,S.G. (1984) Newton-type minimization via the Lanczos method. *SIAM J. Numer. Anal*., **21**, 770–788.

Nelder,J.A. and Mead,R. (1965) A simplex method for function minimization. *Comput. J.*, **7**, 308–313.

Oliver,S.G., Winson,M.K., Kell,D.B. and Baganz,F. (1988) Systematic functional analysis of the yeast genome. *Trends Biotechnol.*, **16**, 373–378.

Rinnooy Kan,A.H.G. and Timmer,G.T. (1989) Global optimization: a survey. *Int. Ser. Numer. Math.*, **87**, 133–155.

Sauro,H.M. (1993) SCAMP: a general-purpose simulator and metabolic control analysis program. *Comput. Applic. Biosci.*, **9**, 441–450.

Savinell,J.M. and Palsson,B.O. (1992) Network analysis of intermediary metabolism using linear optimisation. I. Development of mathematical formalism. *J. Theor. Biol.*, **154**, 421–454.

Schuster,S. and Heinrich,R. (1987) Time hierarchy in enzymatic reaction chains resulting from optimality principles. *J. Theor. Biol.*, **129**, 189–209.

Schuster,S. and Heinrich,R. (1991) Minimization of intermediate concentrations as a suggested optimality principle for biochemical networks. I. Theoretical analysis. *J. Math. Biol.*, **29**, 425–442.

Selkov,E., Maltsev,N., Olsen,G.J., Overbeek,R. and Whitman,W.B. (1997) A reconstruction of the metabolism of *Methanococcus jannaschii* from sequence data. *Gene*, **197**, GC11–GC26.

Skatrud,P.L., Tietz,A.J., Ingolia,T.D., Cantwell,C.A., Fisher,D.L., Chapman,J.L. and Queener,S.W. (1989) Use of recombinant DNA to improve production of cephalosporin C by *Cephalosporium acremonium*. *Bio/Technology*, **7**, 477–485.

Straume,M. and Johnson,M.L. (1992) Analysis of residuals: criteria for determining goodness-of-fit. *Methods Enzymol.*, **210**, 87–105.

Tatusov,R.L., Mushegian,A.R., Bork,P., Brown,N.P., Hayes,W.S., Borodovsky,M., Rudd,K.E. and Koonin,E.V. (1996) Metabolism and evolution of *Haemophilus influenzae* deduced from a whole-genome comparison with *Escherichia coli*. *Curr. Biol.*, **6**, 279–291.

Torres,N.V., Voit,E.O., Glez-Alcón,C. and Rodriguez,F. (1997) An indirect optimization method for biochemical systems: Description of method and application to the maximization of the rate of ethanol, glycerol and carbohydrate production in *Saccharomyces cerevisiae*. *Biotech. Bioeng.*, **55**, 758–772.

Ulmer,K.M. (1983) Protein engineering. *Science*, **219**, 666–671.

Westerhoff,H.V. and Kell,D.B. (1996) What BioTechnologists knew all along…? *J. Theor. Biol.*, **182**, 411–420.

Winston,R.L. and Fitzgerald,M.C. (1997) Mass spectrometry as a readout of protein structure and function. *Mass Spectr. Rev.*, **16**, 165–179.

Wodicka,L., Dong,H.L., Mittmann,M., Ho,M.H. and Lockhart,D.J. (1997) Genome-wide expression monitoring in *Saccharomyces cerevisiae*. *Nature Biotechnol.*, **15**, 1359–1367.

Wolpert,D.H. and Macready,W.G. (1997) No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.*, **1**, 67–82.