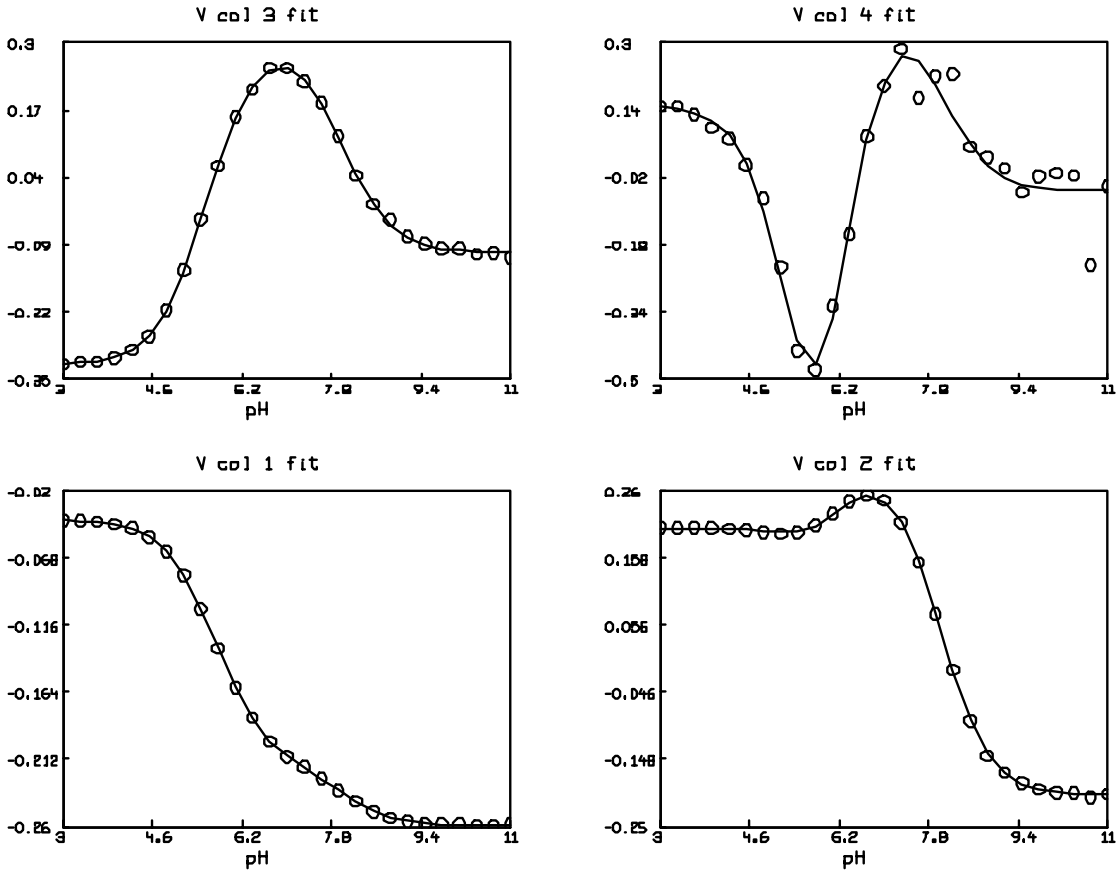
MLAB

A Mathematical Modeling Laboratory
MLAB Applications Manual

Gary Knott and Dan Kerner

Revision Date: May 10, 2004

© Civilized Software, Inc.
8120 Woodmont Ave. #250
Bethesda, MD 20814
phone: +1–301–652–4714
email: csicivilized.com
URL: http://www.civilized.com



Contents

1 INTRODUCTION

MLAB is a computer program whose name is an acronym for "modeling laboratory". It is a tool for experimentation with and evaluation of mathematical models (functions).

MLAB is an interactive system for mathematical and statistical modeling, originally developed at the National Institutes of Health, and now available in enhanced form for most computers, including Macintoshes, Unix machines, and IBM PC's and compatibles, from Civilized Software, Inc.

Although MLAB has hundreds of useful functions, e.g. the discrete Fourier transform function `DFT` and the parametric spline interpolation function SPLINEP, one of the central components of the system is a curve fitting program which will adjust the parameters of a model function to minimize the weighted sum of the errors raised to a specified power. A repertoire of mathematical operators and functions, routines for solving differential equations, a collection of routines for onscreen drawing and for hardcopy plotting, and mechanisms for saving data between sessions provide a powerful and convenient environment for data manipulation, arithmetic calculations, and for building and testing models.

The user communicates with MLAB by typing commands which are executed at once or by executing "script" files containing MLAB "programs". Should the user have questions, typing `HELP` will put the online system documentation at his disposal. The MLAB language is defined in the MLAB reference manual.

One of MLAB's main uses is to fit models to data. Curve-fitting is a useful analytical tool in many diverse disciplines. The basic notion is easily described. Given data, say various points in the plane $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, and a function $y = f(x)$ where f involves some parameters, say a and b, as for example $f(x) = ax^b + 1$, we may wish to calculate values for the parameters a and b so that the function f well-predicts the observed data, that is, so that $f(x_i) = y_i$ for $1 \leq i \leq n$. In this case, we say we have fit the model f to the data by estimating the parameters a and b. The end result is merely the values obtained for the initially unknown parameters. The same principles apply in higher dimensions with arbitrarily many parameters. MLAB can simultaneously fit multiple non-linear model functions, some or all of which may be implicit functions, or may even be defined by a system of differential equations.

The notion of well-prediction which is generally used is that of minimizing the sum of squares,

$$S(a, b) = \sum_{i=1}^{n} (f(x_i) - y_i)^2$$

by appropriately choosing values for a and b. This is because least-squares minimization is a well-studied method which allows various theorems involving error and uniqueness to be invoked. Analogous results do not always exist for other measures of goodness-of-fit.

There are various algorithms which may be employed to minimize a sum-of-squares value. One of the most robust is the carefully-tuned Marquardt-Levenberg method used in MLAB.

This manual is an applications manual for MLAB. Various applications of MLAB are presented in order to exemplify the use of MLAB. For those interested in learning MLAB, the MLAB Reference manual should first be read quickly from front-to-back. Following initial immersion, a more leisurely browsing and study of examples in this manual should follow. Finally, you may find it useful to try the various tutorial sequences which appear throughout.

2 Curve Fitting: Basic Theory

2.1 Modeling By Curve-Fitting

Modeling, as we use the term here, means matching a set of numbers, obtained from some theory, to a set of observed numbers. We will refer to the observed numbers as the observed data or curve and to the theoretical numbers as the computed data or curve. We say curve, because typically the observed data is in the form of a set of points or vectors, some of the component values being independent variables (time, dosage, etc.) and the others dependent variables (concentration, response, spectral intensity, etc.). Our theoretical data is computed from the theory or model, which is a set of functions that compute the theoretical dependent variables. If there are several dependent variables, several functions are required, one for each dependent variable.

Our purpose may be to evaluate the correctness of the model, or to use it to determine otherwise unobtainable values as functions of the observed curve. This ensures that the model be completely known. All the unknown parameters in the model must be established as specific values. Thus the key problem in modeling by curve-fitting is parameter determination. The model functions contain unknown or vaguely known numbers called parameters that affect the shape of the computed curve. The problem is to find those parameter values that make the computed curve most resemble the observed curve in the least-squares sense. This is why we speak of modeling by curve-fitting. Statisticians call this activity linear regression or non-linear regression, depending upon the model. If we wish to test the descriptive quality of a model, the model must first be specialized by choosing parameter values to best-fit the data.

2.2 Input To A Curve-Fit

The input to a modeling problem, in our sense, is listed under four headings below. The definitions and properties of these inputs are discussed just following the list itself.

Observations

Let n be the number of observations (data points). Let k be the number of independent variables. We shall initially assume that we have only a single dependent variable for simplicity. MLAB will fit several functions simultaneously however, so that multi-dimensional models are quite allright. Let $X[i,u] =$ the uth independent variable value at the ith observation and let $Y[i] =$ the observed dependent variable value at the ith observation. $W[i] =$ the statistical weight for the ith observation.

The Model And Parameters

$P_1, P_2, \ldots, P_m = m$ parameters to be estimated.
$x_1, x_2, \ldots, x_k = k$ independent variables.
$F(x_1, x_2, \ldots, x_k) =$ a formula for the computed dependent variable in terms of the arguments $x_1, x_2, \ldots, x_k$; and the parameters $P_1, P_2, \ldots, P_m$. We shall write $F(i)$ as shorthand for the value $F(X[i,1], X[i,2], \ldots, X[i,k])$ of the function F at the point $(X[i,1], \ldots, X[i,k])$.

Constraints

A set of linear equalities and/or inequalities in the parameters which must be satisfied by the final parameter values.

Miscellaneous Control Values

There are two control values, namely the maximum number of iterations: `MAXITER`, and the convergence factor: `TOLSOS`.

Now we may elaborate on the various inputs listed above. The observed independent variable values, $X[i,u]$, are usually the controlling experimental conditions, and the dependent variable values, $Y[i]$, are the measured results. For example:

$X[i,1] =$ observation time after injection of label for the ith experiment.

$X[i,2] =$ initial dosage of label for the ith experiment.

$X[i,3] = 1, 2, 3, 4,$ or 5 depending on whether the label for the ith experiment is detected in plasma, marrow, red cells, urine, or stool.

$Y[i] =$ amount of label detected at time $X[i,1]$ for the ith experiment.

The statistical weight, $W[i]$, reflects the precision of the measurement of $Y[i]$ which is contaminated with random error because of experimental limitations. In light of this probabilistic error in the Y-values, we would like to estimate the reliability of our results (which are the estimated values of the parameters $P_1, P_2, \ldots, P_m$ associated with the best fit function, F.)

The statistical weights are directly related to the variances of the observations. The reciprocal of the variance of the observation $Y[i]$ should be used as the weight $W[i]$ in computing our measure of resemblance, namely the sum-of-squares, as shown below. If we do not know these values, the variances of all observations may be assumed equal or they may be estimated by some mathematical

device such as the EWToperator. Also, if the errors are not independent (for example, if the error is affected by preceding errors), then we should supply covariances. But since there are $n(n+1)/2$ covariances, as compared to only n variances, we compromise with the truth on this issue and use variances alone.

Given the model $F(x_1, \ldots, x_k)$ which computes the theoretical dependent variable given the independent variables $x_1, \ldots, x_k$ and the parameters $P_1, \ldots, P_m$, our goal is to find values for $P_1, \ldots, P_m$ so as to minimize:

$$S(P_1, P_2, \ldots, P_m) = \sum_{i=1}^{n} W[i]\{Y[i] - F(i)\}^2 \quad \text{where} \quad F(i) = F(X[i,1], \ldots, X[i,k]).$$

The sum-of-squares function, S, depends only on the values $P_1, \ldots, P_m$; all other values occuring in S are known constants. S is a measure of disparity between the "curves" Y and F. To accomplish this minimization, we compute $F(i)$ for each i with the initial estimates of $P_1, \ldots, P_m$ supplied by the user. Then based on the differences $Y[i] - F(i)$ and the partial derivatives $\partial F/\partial P_j(i)$, we generate new values of $P_1, \ldots, P_m$; we then recompute each $F(i)$-value and compare them to the corresponding $Y[i]$ values and so on until we can't improve (i.e., reduce) S very much, or until we exceed the allotted number of iterations.

Sometimes, the parameters must be restricted because the best fit may make no physical sense, whereas a slightly worse fit will be perfectly adequate. For example, if the parameters represent fractions of a certain chemical concentration, negative parameters will make no sense. We need a method that will avoid forbidden parameter values, and in addition will find the best permissible set of values.

We deal exclusively with linear constraints. That is, all constraints must have this form: "(coefficient times parameter) plus (coefficient times parameter) plus ..... R constant", where the symbol R can stand for $<$, $=$, or $>$. This class of constraints includes the majority of constraints met in practice. For example, a parameter can be kept within a specified interval, two or more parameters can be kept in a given proportion, two parameters can be kept some minimum distance apart, or all parameters can be required to add up to unity, all with linear constraints.

Geometrically, a set of linear constraints define a possibly unbounded convex region in M-space which has (hyper-)planar borders defined by the various constraint equations. This region is called the constraint region and any acceptable solution (i.e. set of parameter values) must lie within that region.

A definite maximum number of iterations must be specified. An iteration consists of the evaluation of the derivatives of the model with respect to the parameters, followed by several evaluations of the model function itself (each such evaluation is called a subiteration) with various parameter vectors governed by the following factors.

(a) The initial guess or the best parameter vector from the last iteration, i.e., the iteration's starting point.

(b) The residuals $Y[i] - F(i)$ and the partials $\partial F/\partial P_j(i)$ at the starting point $(P_1, P_2, \ldots, P_m)$.

(c) The constraints.

Usually, there are 3 or 4 subiterations per iteration. The parameter vector from the subiteration with the lowest sum-of-squares is chosen as the starting point for the next iteration. The subiterations correspond to moving in the "downhill" direction on the sum-of-squares "surface" by appropriately varying the parameters. The best sum-of-squares value found so far in this process can be reported at the beginning of each subiteration by setting the control variable $LSQRPT$ to 8.

A value called the convergence factor must also be given by setting the control variable `TOLSOS`. This value is used in testing for convergence. Convergence is defined as failure to improve the sum-of-squares by a specified fraction in the current iteration, i.e, $S_{old} - S_{new} <$ (convergence factor) $\times$ S_{old} holds.

A final important control value, set automatically within MLAB, is a scalar called ε. ε assumes several values in each iteration, and it governs the parameter vector as follows. When ε is much less than one, the Marquardt-Levenberg method is essentially a Gauss-Newton iteration, which converges very rapidly when it works but takes "wild leaps" to worse parameters when it fails. When ε is larger than one, the method is essentially a shortened steepest descent step, which converges very slowly, but which is almost guaranteed to improve the sum-of-squares until a local minimum is found. A reasonable default minimum ε value is .0001, which seems to avoid the "wild leap" problem while allowing a reasonable rate of convergence when small ε's are working well; e.g. most linear problems will converge in two iterations using .0001.

2.3 The Method Of Curve-Fitting

The Marquardt-Levenberg curve-fitting method is described below and in [M2] and [S4]. The quadratic programming algorithm, which the curve-fitting algorithm needs to converge within linear constraints is described in [S5]. Both articles contain brief explicit recipes and examples, so one need not hack through the more or less interesting theory. However, some facts which are needed for everyday use, even if you never know the details at all, are listed below.

The method can usually solve a linear problem in one step (although two iterations are recommended for greater precision) with or without constraints. The definition of a linear model is that parameters in a linear model appear only as first power multipliers of terms that contain no other parameters. Thus $F(X) = A \times \sin(X) + B \times \log(X) + C + D \times X^2$ is a linear model with parameters A, B, C, and D.

Even if your model is nonlinear, you may take advantage of the above property. For example, in the model $F(X) = A \times \exp(B \times X) + C \times \exp(D \times X)$, the problem is linear if A and C are used as parameters while B and D are held constant. In one or two iterations, you will get the best possible values of A and C, given the B and D that you have imposed. Then you can fit all four parameters together, knowing that if your B and D estimates were pretty good, so are your A and C estimates.

If the problem is linear, and the output dependency values are reasonable, then the solution is unique. There is no such guarantee in a nonlinear problem. You can try several different first estimates of your parameters. If they all converge to the same solution, this is reassuring but not conclusive. You can also state categorically that your fit has put an upper bound on the sum-of-squares, i.e., the best fit for this model is at least as good as the fit you found. However, the true best fit may have significantly different parameter values. Still, your fit may be so good that no other fit is going to be significantly better.

Now we will describe the curve-fitting process in geometric terms. Consider the sum-of-squares, S, as a function of the parameters $P_1, P_2, \ldots, P_m$ alone:

$$S(P_1, P_2, \ldots, P_m) = \sum_{i=1}^{n} W[i]\{Y[i] - F(i; P_1, \ldots, P_m)\}^2.$$

We can do this because once the model and data points are chosen, the parameters, which are the only free variables in S, determine S. It is helpful to look upon a value of S as one would look upon an altitude-value on a contour map. In fact, for problems in two parameters, such contour maps frequently appear in the literature.

In a linear problem, the contours of the surface defined by S in parameter space are hyperellipsoids in n-space when there are n parameters. It is instructive to look at the two parameter linear cases where the contours of the surface of S, on the (P_1, P_2) map are simply ellipses, all centered at the solution. If we made a plaster model of the surface, it would be a bowl set upon the (P_1, P_2) plane. The lowest point in the bowl would be the solution. Any vertical cross-section of the bowl would be a parabola, and any horizontal cross-section (contour) would be an ellipse.

Because of the simple geometry, we can compute the location of the minimum of (P_1, P_2) directly and go there in one step. If the solution is not unique, we know there is some vertical cross-section that is level rather than parabolic. In this case, our method imposes a minimum second derivative, thus choosing one of the solutions. Large dependency values serve to warn of the non-uniqueness due to a flat section.

If there is a single equality constraint, e.g. $A \times P1 + B \times P2 = C$, then the solution is the minimum of the vertical cross-section (parabola) defined by the constraint line. If that same constraint is of the inequality type, e.g. $A \times P_1 + B \times P_2 < C$, then there are two possibilities: (1) the unconstrained solution lies inside the constraint, so that imposing the constraint really has

no effect; or (2) the unconstrained solution lies outside the constraint, so that equality must be imposed as if the constraint were of the "equality" type.

In order to handle many inequality constraints we must decide which of them must be strictly imposed (these are called active constraints). Again, the geometry is simple, and the linear model with linear constraints (mixed equality and inequality) can be solved in one step.

Nonlinear problems introduce two difficulties. First, they cannot be solved in one step; some iterative scheme must be used. Second, the solutions may be non-unique and separated. The surface S may look, not like a bowl, but more like a series of craters and gullies. Moreover finding the bottom of one depression is no guarantee that it is the deepest depression. Also, introducing constraints may create many solutions, just as building a wall on an inside slope of a gully may cause water to collect in several places when it rains. Nonlinear problems are treated as linear problems, using the partial derivatives of the model function with respect to the parameters as one would use coefficients in a linear regression problem. This is equivalent to pretending that our sum-of-squares surface is always a parabolic bowl with elliptical contours, obviously a dangerous assumption. In particular, a leap to the supposed minimum of that bowl will quite possibly produce an increase in S at that point, rather than the hoped-for minimum. In such cases, one useful feature of the surface is very reliable: the antigradient or steepest descent direction. The value called ϵ, mentioned above, is used to shorten the step-length automatically in bad cases, and to steer the step more toward the antigradient.

2.4 Output From A Curve-Fit

The output from a curve-fit is listed below. The definitions and properties of these outputs are then discussed.

1. Final parameter values.

2. The final sum of squares value corresponding to the final parameter values.

3. The root-mean-square error (rms error) associated with final fit.

4. The mean data error fraction.

5. The determination coefficient, R^2.

6. Error estimates. Each final parameter value has an associated error estimate.

7. Dependency values. Each final parameter has an associated dependency value.

8. Lagrange multipliers. Each constraint has an associated final Lagrange multiplier value.

The final sum-of-squares value is usually a local minimum if there has been convergence in the nonlinear case. In the linear case, it is a global minimum. The sum-of-squares is a useful measure of goodness-of-fit, provided reciprocals of variances have been entered as weights. To tell how probable your fit is, evaluate the chi-square distribution function with the sum-of-squares as the argument: chi-square, and the number of observations minus the number of parameters as the degrees of freedom. The following facts may help. The mean of the chi-square distribution is the degrees of freedom. The event: chi-square $<$ (degrees of freedom)$^{-1}$ always has better than 50 percent probability. For large degrees of freedom (> 30), $T = (2(\text{chi-square}))^{1/2} - ((\text{degrees of freedom}) - 1)^{1/2}$ is normally distributed with mean zero and unit variance.

The root-mean-square error is the average standard deviation about 0 of $W[i] \times (Y[i] - F(i))$ taken over i. It is computed as: $((\text{sum-of-squares})/(\text{degrees of freedom}))^{1/2} = (S/(n-m))^{1/2}$. The mean deviation/data fraction is a weighted average of the fractions $|Y[i] - F(i)|/|Y[i]|$. It is computed as:

$$\sum_{i=1}^{n} W[i]|Y[i] \times (Y[i] - F(i))| / \sum_{i=1}^{n} W[i]Y[i]^2.$$

The determination coefficient, called R^2, is a traditional descriptive statistic for straight line curve-fitting. In this case, R^2 is the fraction of the sum-of-squares which is "accounted for" by fitting a line of non-zero slope. R^2 is discussed further below in section 3.

There are many problems with tests for goodness-of-fit. The variances may be optimistic because of biased observation. (An investigator may take repeated measurements of a blurred phenomenon in a manner that reproduces the error, for example.) The missing covariances may be significant. The errors may be non-normally distributed, and so on. Also, there may be spurious components in the data that the investigator would rather ignore, once he recognizes the problem. So ultimately, a person must decide if the fit is worth anything.

Perhaps the best thing to look at is a graph of Y and F to see how they compare. Such a graph should include the residuals $Y - F$. Look for the maximum residual. There may be a mistaken entry in that observation. Look for long runs of residuals with one sign ($+$ *or* $-$) which indicate non-random disagreement between model and data.

The final parameter values determine the function F which is (hopefully) the closest we can get to Y with our given model. For each parameter there is an associated standard deviation which serves as an error estimate. If the model is linear, and the underlying set of random variables from which the data is a sample are normally distributed, then each parameter divided by its error estimate is Student-t distributed. Each computed parameter value is a maximum likelihood estimate, and the error estimate is the standard error. As the model departs from the linear, and the data error departs from the normal, the parameter error estimates depart from the standard error, but they remain remarkably good in many cases. Constraints are not accounted for in computing error estimates, thus, error estimates are meaningless if there are active constraints.

The size of the error estimates are governed by two factors: the rms error and the dependency values associated with each parameter. The sentence "these parameters are (nearly) dependent" means that a change in any one parameter of the group can be (nearly) compensated for by changes in the other parameters of the group. Thus, appropriate large changes in the parameters will produce no change (or very little change) in the function. Consequently, very little change in the data can produce large changes in the parameters. Dependencies among parameters arise only from the model, the parameter values and the range of the data, not from noise in the data. We need dependency values because we must know, as a guide for corrective action, if a large parameter error stems from data noise or from a poor choice of model or data range.

The dependency values are nondimensional quantities whose ideal values are zero. To be precise, the dependency value for a parameter P is the fraction of the variance of P which can be attributed to uncertainty in other parameters, namely $1-\text{var}(P \text{ with other parameters fixed})/\text{var}(P)$. If all dependency values are zero, there are no dependencies whatever between parameters. If some dependency value is .98 or higher, as a rule of thumb, then the corresponding parameter is nearly dependent on some other parameters which also generally have high dependency values. A dependency value might be slightly less than zero due to round-off error, but if it is very much less than zero, the curve-fitting process has broken down numerically and all your results are meaningless. It is impossible for a dependency value greater than one to occur.

Constraints are as much a part of one's model as the function itself. Some inequality constraints exert no influence on the result, because the best solution lies within the constraint region. Constraints that do affect the result are called active constraints. These are all the inequality constraints that are satisfied by equality (the solution lies on the constraint boundary) and all the equality constraints. Naturally, one wants to know which active constraints are having a strong effect on the fit. The Lagrange multipliers can sometimes help. Each active constraint has an associated value known as its Lagrange multiplier, which in geometric terms, is the change of the sum-of-squares when the constraint boundary is displaced one unit and all other active constraints remain in force. The Lagrange multiplier for a given constraint is the change in the sum-of-squares divided by the corresponding change in the constraint's right hand side. Unfortunately, these values are not nondimensional so their magnitudes will vary from constraint to constraint. Lagrange multipliers can be typed out after each iteration, so you can see which constraints are influencing the trial parameters, but they are meaningful, in the above sense, only at a solution.

The following references are useful sources for learning more about curve-fitting and about on-line interactive mathematics in general.

[A1] Acton, Forman S., *Analysis of Straight-Line Data*, Dover Books, N.Y., 1959.

[C1] Cohen, S., *"Speakeasy, An Evolutionary System"*, SigPlan Notices, Vol. 9, No. 4, pp. 118:126, ACM, 1974.

[C2] Cuthbert, Daniel, and Wood, Fred S., *Fitting Equations to Data*, Wiley-Interscience, N.Y., 1971.

[D1] Davies, R. G., *Computer Programming in Quantitative Biology*, Academic Press, N.Y., 1971.

[D2] Draper, N. R., Smith, H., *Applied Regression Analysis, Wiley*, New York, 1966.

[E1] Ekenberg, Bertil, *"Curve-Fitting by the Use of Graphic Display"*, B.I.T., Vol. 15, pp. 385:393, 1975.

[F1] Fried, B.E., *"Solving Mathematical Problems,"* On-Line Computing, chapter 6, ed. Walter J. Karplus, Mcgraw-Hill, New York 1967. Also see the appendix by G.J. Culler.

[G1] Gear, William C., *"A Generalized Interactive Network Analysis and Simulation System"*, Proc. First USA-JAPAN Computer Conf., pp. 559:566, 1972.

[G2] Golub, G. H., Reinsch, C., *"Handbook Series Linear Algebra: Singular Value Recompositions and Least Squares Solutions"*, Numer. Math., Vol. 14, pp. 403:420, 1970.

[G3] Groner, G. F., Clark, R. L., Berman, R .A., and Deland, E. C., *Biomod: An Interactive Graphics Computer System for Modeling*, Rand Corp. R-617 NIH, July 1971.

[H1] Hobson, Richard F., and Weinkam, James J., *"Curve-Fitting"*, pp. 1:27 in Encyclopedia of Computer Science and Technology, Vol. 7, edited by Belzer, Holzman, and Kent.

[K1] Kahaner, David; Moler, Cleve; Nash, Stephen, *Numerical Methods and Software*, Prentice-Hall, NJ 1989.

[K2] Klerer, Melvin, and Reinfelds, Juris ed. *Interactive Systems for Experimental Applied Mathematics*, Academic Press, New York, 1968.

[K3] Kendall, M. G., Stuart A., *The Advanced Theory of Statistics*, Vol. 2, Second Edition, Hafner Publishing Company, N. Y., 1967.

[M1] Magar, Magar E., *Data Analysis in Biochemistry and Biophysics*, Academic Press, N.Y., 1972.

[M2] Marquardt, D.W., ‘*'An Algorithm for Least-Squares Estimation of Non-linear Parameters"*, J. SIAM, Vol. 11, No. 2, pp. 431:441, 1963.

[R1] Rice, John R. Ed., *Mathematical Software*, Academic Press, New York, 1971.

[R2] Rao, C.Radhakrishna, *Linear Statistical Inference and its Applications*, Wiley, New York, 1965.

[S1] Sall, John, *SAS Regression Applications*, SAS Technical Report A-102, SAS Institute Inc. P.O. Box 10066, Raleigh, N.C., 27605, Aug. 1978.

[S2] Shrager, Richard I., *"Modelaide: A Computer Graphics Program for the Evaluation of Mathematical Models,"* Computer Graphics, Vol. 3, No. 3, pp 17:54, Siggraph-ACM, fall 1969. Also available as technical report no.5, DCRT, NIH.

[S3] Shrager, Richard I., *"Nonlinear Regression with Linear Constraints: An Extension of the Magnified Diagonal Method,"* JACM, Vol. 17, No. 3, pp 446:452, July 1970.

[S4] Smith, Lyle,B., *"The Use of Interactive Graphics to Solve Numerical Problems,"* CACM, Vol. 13, No. 10, pp 625:634, October 1970.

[S5] Shrager, Richard I., *"Quadratic Programming for Non-Linear Regression,"* CACM, Vol. 15, No. 1, pp. 41:44, Jan. 1972.

[S6] Southwell, W. H., *"Fitting Data to Nonlinear Functions with Uncertainties in all Measurement Variables"*, The Computer Journal, Vol. 19, No. 1, pp. 69:73, Feb., 1976.

[S7] Searle, S. R., *Linear Models, Wiley*, New York, 1971.

[T1] Thames, Joe M. Jr., *"Slang: A Problem Solving Language for Continuous-Model Simulation and Optimization,"* Proc. Of the 24th Natl. ACM Conf., pp 23:41, 1969.

3 Curve Fitting: Statistical Theory

3.1 Linear Regression

Suppose we have a vector of random variables, $e = (e_1, e_2, \ldots, e_n)'$ which represent "errors" in some observations. Let y be a vector of random variables, $y = (y_1, y_2, \ldots, y_n)'$, representing the observations such that

$$y_i = b_0 x_{i0} + b_1 x_{i1} + \ldots + b_k x_{ik} + e_i,$$

where $b_0, b_1, \ldots, b_k$ are unknown constants, and $x_{i0}, \ldots, x_{ik}$ are specified values. The random variable y_i is merely a translation of the random variable e_i. Often $x_{i0} = 1$, and in general $x_{i0}, x_{i1}, \ldots, x_{ik}$ may be defined in terms of some lesser number of independent variables. For example, x_{i1} may be an independent variable, and $x_{i0} = 1$ and $x_{i2} = x_{i1}^2$.

Define the vector $b = (b_0, b_1, \ldots, b_k)'$, and the matrix

$$X = \begin{bmatrix} x_{10} & x_{11} & \ldots & x_{1k} \\ x_{20} & x_{21} & \ldots & x_{2k} \\ \ldots & \ldots & \ldots & \ldots \\ x_{n0} & x_{n1} & \ldots & x_{nk} \end{bmatrix}.$$

Thus, X is an $n \times (k+1)$ matrix.

Now in matrix terms, we have $y = Xb + e$. This defines each random variable y_i as a linear expression in $x_{i0}, x_{i1}, \ldots, x_{ik}$ plus an error term e_i.

Any stochastic relationship between the random variable y_i and the values $x_{i0}, \ldots, x_{ik}$ can be expressed by the choice of the random variable e_i. Note that $E(y) = Xb + E(e)$. Suppose that $E(e) = 0$. In this case $E(y) = Xb$ and we say that $y = Xb + e$ is a *linear model* for $E(y)$ since $E(y_i)$ is a linear function of the constants $b_0, \ldots, b_k$.

Now, given X and estimates, $\tilde{y}$, of $E(y)$, we wish to estimate the constants, b, which are the parameters of interest. This is commonly done by least-squares estimation, also known as curve-fitting and regression analysis. We shall summarize the results in this area, based mainly on the text Linear Models by S. R. Searle, Wiley, 1971, and the article "Correlation and Regression" by Neil H. Timm in the Encyclopedia of Computer Science and Technology, Vol. 6, 1977. The linear model $y = Xb + e$ with $E(e) = 0$ encompasses the special case of a stochastic conditional mean linear model where, for $1 \leq i \leq n, (x_{i0}, x_{i1}, \ldots, x_{ik}, \tilde{y})$ is a sample of a random vector $(z_0, z_1, \ldots, z_{k+1})$ which has the property that $E(z_{k+1} | z_0 = x_{i0}, \ldots, z_k = x_{ik}) = x_{i0}b_0 + \ldots + x_{ik}b_k$. Note we do *not* assume that $\text{cov}(z_i, e_j) = 0$, nor that the multivariate distribution of $(z_0, \ldots, z_k)$ does not depend on b.

Let $\mathrm{cov}(e) = V$, so that V is a symmetric $n \times n$ matrix with $V_{ij} = \mathrm{cov}(e_i, e_j)$. Then we may define the sum-of-squares, $S(b) = (y - Xb)'V^{-1}(y - Xb)$. $S(b)$ is a random variable which depends upon the parameters b. Note when b is the parameter vector such that $E(y) = Xb$, then $S(b) = e'V^{-1}e$.

Now define $\hat{b} = (\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_k)'$ as the vector of random variables such that

$$\hat{b} = (X'V^{-1}X)^{-1}X'V^{-1}y.$$

$\hat{b}$ is the solution to the system of equations $\partial S/\partial b = 0$, so that $E(S(\hat{b}))$ is minimal. $\hat{b}$ is our *estimator* of b. Since V or $X'V^{-1}X$ may be singular, we assume throughout that all matrix inverses are Moore-Penrose generalized inverses.

The relation $y = Xb + e$ states that, except for error, y is a vector which is a linear combination of the columns of X, and hence lies in the column-space of X. $X\hat{b}$ is the orthogonal projection of y on the column space of X, and the vector $y - X\hat{b}$, which is orthogonal to $X\hat{b}$, is the vector deviation of y from the column space of X due to the errors e.

Given a vector of sample values, $\tilde{y}$, of y, which are estimates of $E(y)$, we obtain an estimate of b as $\tilde{\hat{b}} = (X'V^{-1}X)^{-1}X'V^{-1}\tilde{y}$. (Note an *estimator* is a random variable; an *estimate* is a number.) The estimator $\hat{b}$ is a peculiar estimator, since it has useful properties only when $X'V^{-1}X$ is non-singular. When V itself is non-singular, this is equivalent to the condition: $r(X) = k + 1$, where $r(X)$ is the rank of the matrix X.

Suppose that $X'V^{-1}X$ is non-singular. Then under the assumptions $E(e) = 0$, and $\mathrm{cov}(e) = V$, which are to hold henceforth, the random vector $\hat{b}$ is the *best linear unbiased estimator* of b, i.e., the *b.l.u.e.* of b, for short. $\hat{b}$ is a linear function of y, and $E(\hat{b}) = b$, so $\hat{b}$ is *unbiased*. Finally, for an arbitrary $(k + 1)$-vector of coefficients q, $\mathrm{var}(q'\hat{b}) = q'(X'V^{-1}X)^{-1}q$ is minimal over all linear estimators of b, so $\hat{b}$ is a *best linear estimator* of b. Also, $\mathrm{cov}(\hat{b}) = (X'V^{-1}X)^{-1}$.

When $X'V^{-1}X$ is singular, we have $E(\hat{b}) = (X'V^{-1}X)^{-1}X'V^{-1}Xb \neq b$, and $\mathrm{cov}(\hat{b}) = (X'V^{-1}X)^{-1}$. Thus, in this case, $\hat{b}$ is not an unbiased estimator of b. Indeed, $\hat{b}$ is not even a consistent estimator of b. There is not enough information to estimate b; the best we can do is to observe that certain linear functions of the b_i-values can be estimated. Such functions are called *estimable* functions. An arbitrary linear expression $q'b$ involving a coefficient vector q, is estimable if and only if $E(q'\hat{b}) = q'b$, so that $q'\hat{b}$ is an unbiased estimator of $q'b$. Let q be a $(k + 1)$-vector of coefficients such that $q'b$ is estimable. q is called an *estimation-admissible* vector of coefficients for b. q lies in the column space of $(X'V^{-1}X)^{-1}X'V^{-1}X$. When $q'b$ is estimable, $q'\hat{b}$ is the b.l.u.e. for $q'b$.

The matrix X may sometimes be chosen in advance and then used to obtain the sample values . In this case, we can choose X so as to reduce the values $\mathrm{var}(\hat{b}_i)$, as follows.

Let X_i denote the ith column of X. Let $c_i^2 = X_i'V^{-1}X_i$. Then $\mathrm{var}(\hat{b}_i) \geq 1/c_i^2$, and $\mathrm{var}(\hat{b}_i) = 1/c_i^2$ when $X_i'V^{-1}X_j = 0$ for $i \neq j$. Thus if the elements of X are chosen so that $X_i'V^{-1}X_j = 0$ for $i \neq j$ and $X_i'V^{-1}X_i$ is large, then $\mathrm{var}(\hat{b}_i)$ is accordingly small. For $V = I\sigma^2$, this implies X should

be chosen with orthogonal column vectors which have large Euclidean norms. Note also that if $X'V^{-1}X$ is diagonal, then the parameter estimators $\hat{b}_i$ are uncorrelated.

Let us now introduce various distributions. Let $N(\mu, V)$ be a random vector which is normally-distributed with vector of means μ and covariance matrix V.

Let $t(n)$ denote a random variable having the Student's$-t$ distribution with n degrees of freedom.

Let $g(x, y)$ denote a random variable having the gamma distribution with mean x and variance y.

Let $\chi^2(n, \lambda)$ denote a random variable having the non-central chi-square distribution with n degrees of freedom and non-centrality parameter λ.

Let $F(n, m, \alpha)$ denote a random variable having the singly non-central F-distribution with (n, m) degrees of freedom and non-centrality parameter α.

The notation $A \sim B$ will be used throughout to mean the random variable A has the same distribution function as the random variable B.

Now, when the stronger assumption: $e \sim N(0, V)$ holds, then $\hat{b}$ is also the maximum likelihood estimator of b, and moreover, when q is an estimation-admissible coefficient vector for b, $q'\hat{b}$ is the best unbiased estimator of $q'b$ among all unbiased estimators of $q'b$, not just the subclass of linear estimators.

Let $\hat{y}$ be the estimator of $E(y)$ defined by $\hat{y} = X\hat{b}$. Then $\hat{y}$ is the b.l.u.e. of $E(y)$, with $E(\hat{y}) = E(y) = Xb$, and $\text{cov}(\hat{y}) = X\text{cov}(\hat{b})X'$.

We may use the estimator $\hat{b}$ for prediction. Suppose $x_s = (x_{s0}, \ldots, x_{sk})$ is given, then we may *predict* a later sample value, $\tilde{y}_s$, of $y_s = x_s b + e_s$, using $\hat{y}_s = x_s\hat{b}$ as an estimator for $\tilde{y}_s$. $\hat{y}_s$ is unbiased, and $\text{var}(\hat{y}_s) = x_s(X'V^{-1}X)^{-1}x_s'$, so $\text{var}(\hat{y}_s - y_s) = x_s(X'V^{-1}X)^{-1}x_s' + \text{var}(e_s)$.

$\hat{y}_s$ is, of course, also an estimator of $E(y_s)$, and $\text{var}(\hat{y}_s - E(y_s)) = Var(\hat{y}_s) \leq \text{var}(\hat{y}_s - y_s)$, so $\hat{y}_s$ is a more precise estimator of $E(y_s)$ than of a sample value $\tilde{y}_s$.

Now suppose $\text{cov}(e) = I\sigma^2$, so that e_i and e_j are uncorrelated for $i \neq j$, and $\text{var}(e_i) = \text{var}(e_j) = \sigma^2$ for $1 \leq i, j \leq n$. This is a very strong assumption, but when it holds, we have a means of estimating σ^2.

For $\text{cov}(e) = I\sigma^2$, we define $Q(b) = (y - Xb)'(y - Xb)$. Note $Q(b) = S(b)$ when $V = I$. The particular sum-of-squares random variable $Q(\hat{b})$ is called SSE, the *sum-of-squares of the residuals.* It can be shown that, when V is taken as a multiple of I, E(SSE)$= (n - r(X))\sigma^2$, where $r(X)$ is the rank of X, which, when $X'X$ is non-singular, is $k + 1$. Thus, the random variable $\hat{\sigma}^2 = \text{SSE}/(n - r(X))$ is an estimator for σ^2. Note $\text{SSE} = (y - \hat{y})'(y - \hat{y})$ does not involve σ^2. $\hat{\sigma}^2$ is the b.l.u.e. of σ^2.

Certain other sums-of-squares are of interest for the analysis of variance. Let the *total sum-of-squares* random variable SST, be defined by $\mathrm{SST} = y'y$, and let the *sum-of-squares due to regression* be the random variable $\mathrm{SSR} = \mathrm{SST} - \mathrm{SSE}$. Note $\mathrm{SSR} = \hat{y}'y$, since $y - \hat{y} = (I - X(X'X)^{-1}X')y$. Also, define $\bar{y} = \sum_{i=1}^{n} y_i/n$, and let the *sum-of-mean-squares* be the random variable $SSM = n\bar{y}^2$. Then we may define the *sum-of-squares due to regression corrected for the mean* as $SSR_m = SSR - SSM$, and the *total sum-of-squares corrected for the mean* $SST_m = SST - SSM$.

Using these random variables, we may define the *coefficient of determination* $R^2 = SSR_m/SST_m$. R^2 is the fraction of the total sum-of-squares corrected for the mean which is accounted for by fitting the model, e.g., by using $\hat{b}$ as an estimator for b. $(1 - R^2)$ is the *fraction due to observation error*. If we think of the points $(x_{i0}, \ldots, x_{ik}, \tilde{y}_i)$ as samples of a normally-distributed random vector, $(z_0, z_1, \ldots, z_k, z_{k+1})$, then $R = \mathrm{cor}(z_{k+1}, E(z_{k+1}|z_0, z_1, \ldots, z_k))$. Thus, the positive square root R is called the *multiple correlation coefficient.*

Now we assume that $e \sim N(0, I\sigma^2)$, so $V = I\sigma^2$. This stringent condition is enough to obtain the distributions of all the random variables introduced above. Define $\bar{1} = (1, 1, \ldots, 1)'$. Then we have:

$$y \sim N(Xb, I\sigma^2).$$

$$\hat{b} \sim N((X'X)^{-1}X'Xb, (X'X)^{-1}\sigma^2).$$

$$\hat{\sigma}^2 \sim g((n - r(X))/2, \sigma^2/(2(n - r(X)))).$$

$$\hat{b} \text{ and } \hat{\sigma}^2 \text{ are independent.}$$

$$SSE/\sigma^2 \sim \chi^2(n - r(X), 0).$$

$$SSM/\sigma^2 \sim \chi^2(1, (\bar{1}'Xb)^2/(2n\sigma^2)).$$

$$SSR/\sigma^2 \sim \chi^2(r(X), b'X'Xb/(2\sigma^2)).$$

$$SSE \text{ and } SSR \text{ are independent. } SSE \text{ and } SSM \text{ are independent.}$$

$$(n - r(X))SSR/(r(X)SSE) \sim F(r(X), n - r(X), b'X'Xb/(2\sigma^2)).$$

$$(n - r(X))(Q(b) - Q(\hat{b}))/(r(X)Q(\hat{b})) \sim F(r(X), n - r(X), 0)$$

$$(n - r(X))SSM/SSE \sim F(1, n - r(X), (\bar{1}'Xb)^2/(2n\sigma^2)).$$

Also, when $e \sim N(0, I\sigma^2)$, q is an estimation-admissible vector of coefficients for b if and only if q lies in the row-space of X, which is all of $(k+1)$-space precisely when $r(X) = k + 1$.

3.2 Confidence Intervals

Let q be an estimation-admissible vector of coefficients with which we form a linear combination of the random variables $\hat{b}_i$. Then, under our assumption $e \sim N(0, I\sigma^2)$, we have $(q'\hat{b} - q'b)/[q'(X'X)^{-1}q\hat{\sigma}^2]^{1/2} \sim t(n - r(X))$.

We have the $(1-\alpha)$-confidence interval:

$$q'\hat{b} - h_\alpha[q'(X'X)^{-1}q\hat{\sigma}^2]^{1/2} \leq q'b \leq q'\hat{b} + h_\alpha[q'(X'X)^{-1}q\hat{\sigma}^2]^{1/2},$$

with probability $1-\alpha$, where h_α is defined by $P(t(n - r(X)) \geq h_\alpha) = \alpha/2$.

Note a confidence interval c is an "interval-valued" random variable. This means that a sample $\tilde{c}$ is an explicit interval which covers the true point with probability $1-\alpha$.

Also, recalling our prediction estimator, $\hat{y}_s = x_s\hat{b}$, for a sample value, $\tilde{y}_s$, of $y_s = x_s b + e_s$, with $e_s \sim N(0, \sigma^2)$, we have $(\hat{y}_s - y_s)/[x_s(X'X)^{-1}x_s'\hat{\sigma}^2 + \hat{\sigma}^2]^{1/2} \sim t(n - r(X))$, so that $x_s\hat{b} - h_\alpha[(x_s(X'X)^{-1}x_s' + 1)\hat{\sigma}^2]^{1/2} \leq y_s \leq x_s\hat{b} + h_\alpha[(x_s(X'X)^{-1}x_s' + 1)\hat{\sigma}^2]^{1/2}$ holds with probability $1-\alpha$, where h_α is defined as before.

Separate confidence intervals for each b_i value can be misleading, since the rectangular region defined by the cross-product of the separate confidence intervals is *not* a joint $(1-\alpha)$ confidence region for b. One type of such a region can be defined in parameter space, as: $\{b|G(b) \leq v\}$ where $P(b \in \{b|G(b) \leq v\}) = 1-\alpha$, and where $G(b)$ is a function (i.e. a random variable) which depends upon an argument vector b, and v is a constant independent of b.

Let us define the random variable, $G(b)$, as follows:

$$G(b) = (n - r(X))(Q(b) - Q(\hat{b}))/(r(X)Q(\hat{b})) = (\hat{b} - b)'(X'X)^{-1}(\hat{b} - b)/(r(X)\hat{\sigma}^2).$$

Then under the assumption $e \sim N(0, I\sigma^2)$, we have $G(b) \sim F(r(X), n-r(X), 0)$, so the distribution $F_{G(b)}(z)$ of $G(b)$ is an F-distribution.

Thus, we may choose v such that $P(G(b) \leq v) = 1-\alpha$, so that $v = F^{-1}_{G(b)}(1-\alpha)$. Then the contour $\{b|G(b) = v\}$ in parameter space is the boundary of a joint $(1-\alpha)$ confidence region for b. Note the contour $\{b|G(b) = v\}$ is a "contour-valued" random variable, since $G(b)$ is defined in terms of the vector of random variables y. The appropriate interpretation of this statement is that, upon repeated sampling of y, the contours thus determined will enclose the point $b, 100(1-\alpha)$ percent of the time.

The contour equation $G(b) = v$ is equivalent to the equation $Q(b) = Q(\hat{b})(1+(r(X)/(n-r(X)))v)$. Treating random variables as constants, this is an ellipsoid in the $(k+1)$-dimensional parameter

space. The center of this ellipsoid is $\hat{b}$, and it has $(k+1)$ principal axes of lengths: $(\rho/\lambda_k)^{1/2} \geq (\rho/\lambda_{k-1})^{1/2} \geq \ldots \geq (\rho/\lambda_0)^{1/2}$, where $\rho = Q(\hat{b})(1 + (r(X)/(n - r(X)))v) + y'X(X'X)^{-1}X'y$, and where $\lambda_0 \geq \lambda_1 \geq \ldots \geq \lambda_k$ are the (positive real) eigenvalues of $X'X$. The principal axes are mutually orthogonal and the direction of the principal axis of length $(\rho/\lambda_i)^{1/2}$ is the same as the direction of the eigenvector of $X'X$ corresponding to the eigenvalue, λ_i.

In general, a joint confidence region, R, is an ellipsoid in $(k+1)$-space. The sum-of-squares function, $Q(b)$, defined on $(k+1)$-space with y treated as a vector of constants, has a graph in $(k+2)$-space which is a bowl-like ellipsoidal "surface" which has a unique minimum at $b = \hat{b}$. The boundary of the region R is a contour of the graph of $Q(b)$ in the domain of Q.

3.3 Calibrated Estimation of x

Given the model $y = Xb + e$, we can estimate b with $\tilde{\hat{b}}$ based on sample values $\tilde{y}$. Then later, given a further value $\tilde{y}_s$, which is a sample of the additional random variable y_s, we may wish to estimate the corresponding vector x, such that $xb = E(y_s)$, or equivalently, $xE(\hat{b}) = E(\hat{y}_s)$.

But x is not estimation-admissible in this situation, unless only a scalar need be estimated, since we have only one equation in hand. Thus suppose $x = [1\ x_1]$ and we have $E(\hat{b}_0) + x_1 E(\hat{b}_1) = E(y_s)$. Then $x_1 = (E(y_s) - E(\hat{b}_0))/E(\hat{b}_1)$, and we may use the random variable $\hat{x}_1 = (y_s - \hat{b}_0)/\hat{b}_1$ as the estimator for x_1. In order to compute an estimate of x, we use our sample $\tilde{y}_s$ and the previously-determined $\tilde{\hat{b}}$ values to obtain $\tilde{\hat{x}}_1 = (\tilde{y}_s - \tilde{\hat{b}}_0)/\tilde{\hat{b}}_1$.

This method of estimating x_1 is called calibrated estimation, because we first use the data X and $\tilde{y}$ to compute $\hat{b}$ which is used to specify or "calibrate" the estimator $\hat{x}_1$, so that given only a further sample, $\tilde{y}_s$, the corresponding value x_1 can be estimated.

In this special linear case, with only the scalar x_1 to be estimated, if $y \sim N(Xb, I\sigma^2)$, so that $\hat{b} \sim N((X'X)^{-1}X'Xb, (X'X)^{-1}\sigma^2)$, and if $y_s \sim N(b_0 + x_1 b_1, \sigma^2)$, and $\mathrm{cov}(y_s, y_i) = 0$ for $1 \leq i \leq n$, then if the estimated slope $\tilde{\hat{b}}_1$ is not too close to zero, we can specify a $1 - \alpha$ confidence interval for x_1. In fact, the distribution of $\hat{x}_1$ can be stated. See "On the ratio of two correlated random variables" by D.V. Hinkley in Vol. 56, pp. 635:639 of Biometrika, 1969.

In particular, let h_α be such that $P(F(1, n-2, 0) \geq h_\alpha) = \alpha$, and define the random-variable $u = n\hat{b}_1/\hat{\sigma}^2 - h_\alpha$. The calibration data X and $\tilde{y}$ determines the sample value $\tilde{u}$. If $\tilde{u} > 0$ then

$$P[((u + h_\alpha)\hat{x}_1 - [h_\alpha((n+1)u + (u + h_\alpha)\hat{x}_1^2)]^{1/2})/u \leq x_1 \leq \\ ((u + h_\alpha)\hat{x}_1 + [h_\alpha((n+1)u + (u + h_\alpha)\hat{x}_1^2)]^{1/2})/u] \geq 1 - \alpha.$$

This follows from the fact that $n\hat{b}_1^2(\hat{x}_1 - x_1)^2/(\hat{\sigma}^2(n + 1 + x_1^2)) \sim F(1, n-2, 0)$.

If $\tilde{u} \leq 0, then|\tilde{\hat{b}}_1|$ is so small that the probability statement above does not hold. Further details appear in "A Bayesian Look at Inverse Linear Regression" by B. Hoadley in Vol. 65., pp. 356:369 of the Journal of the American Statistical Association, March 1970.

Note this confidence interval does not apply jointly for the estimates of a number of x-values based on a common calibrated line. Although a particular x-value lies in its confidence interval with probability $1-\alpha$, the probability that all the x-values being estimated simultaneously lie in their associated intervals is less than $1-\alpha$.

3.4 Influence of Observations

Some of the observations $(x_{i0}, x_{i1}, \ldots, x_{ik}, \tilde{y}_i)$ may be atypical to the degree that $\tilde{\hat{b}}$ is significantly distorted. Such points are often called *outliers*. Detecting outliers involves considering the residual estimator $\hat{e} = y - X\hat{b}$.

Cook has suggested some ways to measure the influence of an observation on the estimator $\hat{b}$ in "Detection of Influential Observations in Linear Regression", Technometrics, Vol. 19, pp. 15:18, 1977.

Let $\hat{b}_{-i}$ denote the maximum-likelihood estimator for b obtained by neglecting the ith observation $(x_{i0}, \ldots, x_{ik}, y_i)$. Define the random variable $M_i(A,c) = (\hat{b} - \hat{b}_{-i})'A(\hat{b} - \hat{b}_{-i})/c$, where A is a positive semidefinite matrix ($\det(A) \geq 0$), and c is a positive scaling factor. One interesting choice for A and c is $A = X'V^{-1}X$ and $c = r(X)\hat{\sigma}^2$.

Let $D_i = M_i(X'V^{-1}X, r(X)\hat{\sigma}^2)$. D_i is a metric-value or measure of distance between $\hat{b}$ and $\hat{b}_{-i}$. The magnitude of D_i indicates the influence of the ith data point in the estimator $\hat{b}$. The random variable D_i can be computed without obtaining $\hat{b}_{-i}$, since $D_i = (A_{ii}/(1-A_{ii})^2)\hat{e}_i^2/(r(X)\hat{\sigma}^2)$, where $A = X(X'V^{-1}X)^{-1}X'$.

One device for controlling the effect of outliers is to weigh the ith point by $1/D_i$, and perform a calculation of the estimate $\tilde{\hat{b}}$ based on these weights.

3.5 Hypothesis Testing

In general, hypothesis testing tests a *null hypothesis* H_0, versus an *alternate hypothesis* H_1, where we have a statistic p, such that H_0 implies $p \sim A$, and H_1 implies $p \sim B$, where the distribution F_A, at least, is known.

Then to test H_0 at the 100α percent significance level, we first compute d as the solution to $P(A \geq d) = \alpha = 1 - F_A(d)$. Then we obtain a sample value of p, $\tilde{p}$, and if $\tilde{p} > d$, we can reject the hypothesis H_0.

The following situations hold:

no error: (H_0 is true and $\tilde{p} \leq d$) or (H_0 is false and $\tilde{p} > d$).

type 1 **(rejection) error:** (H_0 is true and $\tilde{p} > d$).

type 2 **(acceptance) error:** (H_0 is false and $\tilde{p} \leq d$).

We have $P(\text{type 1 error}) = \alpha$, and, if H_0 is true *if and only if* H_1 is false, then we can compute $P(\text{type 2 error}) = P(H_1 \text{ is true and } \tilde{p} \leq d) = P(B \leq d) = F_B(d)$. $1 - F_B(d)$ is called the *power* of the test with respect to H_1. It may be that we have a so-called composite hypothesis, such that F_B is not known, but often suitable approximations can be used.

Now suppose that $e \sim N(0, I\sigma^2)$, and let K be an $s \times (k+1)$ matrix with $r(K) = s \leq k+1$, whose rows are estimation-admissible coefficient vectors for b, and let m be an s-vector of constants. Then $(K\hat{b} - m) \sim N(Kb - m, K(X'X)^{-1}K'\sigma^2)$.

Let the random variable $p = (K\hat{b} - m)'(K(X'X)^{-1}K')^{-1}(K\hat{b} - m)/(r(K)^{\hat{\sigma}^2})$.

Then $p \sim F(r(K), n - r(X), (Kb - m)'(K(X'X)^{-1}K')^{-1}(Kb - m)/(2\sigma^2))$.

If $Kb = m$, then $p \sim F(r(K), n - r(X), 0)$. Thus, p is an F-statistic for testing the hypothesis $Kb = m$. We can contrast the hypothesis $Kb = m$ to an alternate hypothesis $Kb = m^*$, if we use an approximation for the unknown parameter σ^2 in the general non-central F-distribution for p, in order to make the distribution known.

In particular, when $r(X) = k + 1$, then for $K = I$ with $s = k + 1 = r(K)$, we have:

$$(\hat{b} - m)'(X'X)^{-1}(\hat{b} - m)/((k+1)\sigma^2) \sim F(k+1, n - r(X), 0),$$

which is an $F(k+1, n - r(X), 0)$-statistic for testing the hypothesis $b = m$.

3.6 Error in the X-Values

Often the values x_{ij} in the matrix X cannot be precisely known. Thus, we may have the situation $y = E(X)b + e$ and $E(e) = 0$ where X is a matrix of random variables, and we are given sample values $\tilde{y}$ *and* $\tilde{X}$ and we wish to estimate the parameters b. This is called a structural model, as opposed to a functional model, since y is now a random vector defined in terms of a matrix of random variables, X, plus error e.

We now have $E(y) = E(X)b$. Define $D = X - E(X)$. Now we assume $E(Db) = 0$, and *if* $\text{cov}(Db) = \gamma I$ for some scalar γ, and $cov(D_i b, e_j) = 0$, then the expectation of the sum-of-squares

at $\hat{b}$, $E(S(\hat{b}))$, is minimal, as before, where X is now a matrix of random variables. $(X'\mathrm{cov}(e)X)^{-1}$ is now an estimator of $\mathrm{cov}(\hat{b})$, rather than $\mathrm{cov}(\hat{b})$ itself. Note that $y = Xb + (e - Db)$, so that $\hat{\sigma}^2 I$ is now an estimator for $\mathrm{cov}(e - Db)$, when $\mathrm{cov}(e - Db)$ is of the form $\hat{\sigma}^2 I$.

Practically speaking, we often assume the conditions $\mathrm{cov}(Db) = \gamma I$ and $\mathrm{cov}(D_i b, e_j) = 0$ when the error in x_{ij} is independent of $E(x_{ij})$, but these conditions are clearly violated if the expected error in x_{ij} is proportional to $E(x_{ij})$.

We may have a mathematically similar but conceptually distinct situation where X is a matrix of random variables and $y = Xb + e$ with $E(e) = 0$, and we observe, not samples of X and y, but samples of $X + H$ and y, where H is a matrix of random variables with $E(H) = 0$ and $\mathrm{cov}(H_i b, e_j) = 0$, and $\mathrm{cov}(Hb)$ is diagonal. Nevertheless we still have $E(y) = E(X)b$, and linear regression applies as above when these covariance relations hold.

Even when $\mathrm{cov}(y)$ is not diagonal, regression analysis can be applied in certain special circumstances. Let us consider the case where we have $y = [1 \;\; x]b + e$ with $x = (x_1, \ldots, x_k) \sim N(\mu_x, V)$, and $\mathrm{cov}(x_i, e) = 0$, and $e \sim N(0, \sigma^2)$. Let $b = (b_0, b_1, \ldots, b_k)' = [b_0 \; a']$. Then $y \sim N([1 \; \mu_x]b, a'Va + \sigma^2)$. Let $\sigma_{xy} = (\mathrm{cov}(x_1, y), \ldots, \mathrm{cov}(x_k, y))$, and $\mu_y = E(y)$. Then $\sigma_{xy} = a'V$ and $E(y|x) = \mu_y + \sigma_{xy}V^{-1}(x - \mu_x)'$. Thus $a' = \sigma_{xy}V^{-1}$ and $b_0 = \mu_y - \mu_x a$.

Now let $d = (d_1, \ldots, d_k)$ with $d \sim N(0, D)$, and $\mathrm{cov}(x_i, d_j) = \mathrm{cov}(e, d_j) = 0$. Let $w = x + d$. Then $w \sim N(\mu_w, W)$ with $\mu_w = \mu_x$ and $W = V + D$. We have $E(y|w) = \mu_y + \sigma_{wy}W^{-1}(w - \mu_w)'$ where $\sigma_{wy} = (cov(w_1, y), \ldots, cov(w_k, y)) = \sigma_{xy}$. Thus $E(y|w) = [1 \;\; w]h$, where $h = [h_0 \;\; g']'$, with $g' = a'VW^{-1}$ and $h_0 = \mu_y - \mu_w g$. Therefore, if $V = \mathrm{cov}(x)$ is known and we estimate μ_y, μ_w, and W with the estimators $\hat{\mu}_y$, $\hat{\mu}_w$, and $\hat{W}$, based on data $(\tilde{w}, \tilde{y})$, then we can estimate h with the usual least-squares estimator $\hat{h} = [\hat{h}_0 \;\; \hat{g}']'$, and then estimate $b_{1:k}$ with $\hat{a}' = \hat{g}'\hat{W}V^{-1}$ and $\hat{b}0 = \hat{\mu}_y - \hat{\mu}_w\hat{a}$. When $V = diag(\sigma_1^2, \ldots, \sigma_k^2)$ and $D = \mathrm{diag}(\varepsilon_1^2, \ldots, \varepsilon_k^2)$, then we obtain

$$\hat{a} = \hat{g}'diag(\sigma_1^2 + \varepsilon_1^2)/\sigma_1^2, \ldots, (\sigma_k^2 + \varepsilon_k^2)/\sigma_k^2).$$

3.7 Principal Components Analysis

In the general case where the elements of X are random variables, we require a different approach to estimating b, since regression analysis no longer applies. Recall that we were able to incorporate a constant term, b_0, in our linear model $E(y) = Xb$ by permitting the first column of X to be $\bar{1}$. When we consider x_{ij} to be a random variable, it is no longer convenient to use this device for providing a constant term. Thus, we shall consider the linear model $E(y) = E(X)b + \bar{1}d$. Let $U = [-X \; y]$. Then we may write our model as $E(U)a = \bar{1}d$ where $a = [b' \; 1]'$. We wish to estimate the unknown components of a, and the scalar d, given a matrix of samples, $\tilde{U}$ of U, which estimates $E(U)$.

Let $p = k + 2$, and let u_i denote the ith row of U, so that $u_1, u_2, \ldots, u_n$ are p-dimensional random points in p-space, and let $\bar{u} = (u_1 + u_2 + \ldots + u_n)/n$. We make two special assumptions about

the random vectors u_i. They may be correlated, and they need not be identically-distributed, however, we assume that (1) $\mathrm{cov}(u_{is}, u_{it}) = 0$ for $s \neq t$ and $1 \leq i \leq n$, and that (2) the density function of each u_i has ellipsoidal contours. (Note that assumption (1) implies that the axes of these ellipsoids are aligned with the coordinate axes.)

For an arbitrary random vector, w, define $h(w) = v$, such that $v_i = (w_i - \hat{w}_i)(\sigma^2/Var(w_i))^{1/2} + \hat{w}_i$, where $\hat{w}_i$ is either 0 or an unbiased estimator of $E(w_i)$, and σ^2 is a positive constant. Let Q be the $n \times p$ matrix whose ith row is $q_i = h(u_i) - \bar{u}$. We have $E(Q_{ij}) = E(U_{ij}) - E(\bar{u}_j)$, and $E(U)a = \bar{1}d$ implies that $E(Q)a = 0$.

The point of using the transformation, h, to convert the matrix, U, to the matrix, Q, is that, with the assumptions above, the density functions of the elements of Q have spherical contours around the means, so minimizing the sum-of-squares of the perpendicular distances to the model plane in order to determine that plane is now appropriate.

Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p$ be the eigenvalues of $Q'Q = n \cdot \mathrm{cov}(Q)$, and let $P_1, P_2, \ldots, P_p$ be the corresponding eigenvectors of unit length. $P_1, \ldots, P_p$ form an orthonormal basis, and the subspace S_t spanned by $P_1, P_2, \ldots, P_t$ is the best-fitting t-dimensional plane to the points q_1, $\ldots$, q_n, in the sense that the sum-of-squares of the lengths of the perpendiculars from the points q_1, $\ldots$, q_n to S_t is minimal.

Let $P(t)$ be the $p \times t$ matrix whose columns are $P_1', P_2', \ldots, P_t'$. Then $P(t)P'(t)$ is the projection matrix which projects a point into S_t. S_t is thus the null-space of $I - P(t)P'(t)$, so $S_t = \{\, v \mid v(I - P(t)P'(t)) = 0 \,\}$. The sum of the squares of Euclidean distances which is minimal is

$$\sum_{1 \leq i \leq n} \parallel q_i - q_i P(t)P'(t) \parallel^2 = \lambda_{t+1} + \ldots + \lambda_p$$

.

Note $P(t)P'(t) = P_{(p)} diag(1, \ldots, 1, 0, \ldots, 0) P_{(p)}^{-1}$, since projecting to S_t is equivalent to rotating so that S_t has the basis $(1, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0)$, $\ldots$, $(0, \ldots, 0, 1, 0, \ldots, 0)$, projecting by zeroing the last $p - t$ coordinates, and rotating back to the original coordinates.

When $t = p-1$, the subspace S_t is a $(p-1)$-dimensional plane which has a unique normal direction given by the "missing" direction, P_p, which is the eigenvector corresponding to the least eigenvalue, λ_p, so that $S_{p-1} = \{\, v \mid vP_p' = 0 \,\}$. Thus, the equation of the plane S_{p-1} is $qa^* = 0$ where $a^* = P_p'$.

Now define the vector of random variables $\hat{a} = (1/a_p^*)a^*$ (if $a_p^* = 0$, then take $\hat{a} = a^*$). Then $\hat{a}$ is the maximum-likelihood estimator of a in $E(Q)a = 0$. Since $E(Q) = E(U) - \bar{1}E(\bar{u})$, $E(U)a = \bar{1}d$ implies $\bar{u}\hat{a} = \hat{d}$ is the maximum-likelihood estimator of d. Equivalently, define $\hat{b} = \hat{a}_{1:(p-1)}$. $\hat{b}$ is the maximum-likelihood estimator of b and $\hat{d}$ is the maximum-likelihood estimator of d in the model $E(y) = E(X)b + \bar{1}d$.

[What about a linear model with some $x's$ known precisely, and others not: $E(y) = E(Z)a + Xb$?]

This method of obtaining $\hat{a}$ and $\hat{d}$ is based on principal components theory following section 8g of *Linear Statistical Inference and Its Applications* by C. R. Rao, Wiley, 1965. It requires that the variances of the elements of U be known to within a common scale-factor and that the zero-correlation and distribution assumptions above hold. It is, however, robust where these assumptions are violated.

When the points $u_1, u_2, \ldots, u_n$ share a common multivariate distribution, so that $u_i \sim z$, where z is a random p-vector, for $1 \leq i \leq p$, our model, $E(U)a = \bar{1}d$, reduces to $E(z)a = d$. Let $S(\mu, \sigma^2)$ be a random variable with mean μ and variance σ^2. Suppose that $z_j \sim S(\mu, \sigma_j^2)$, and that constants $g_1, g_2, \ldots, g_p$ are known such that $\sigma_1^2/g_1 = \ldots = \sigma_p^2/g_p = \sigma^2$, for some constant value σ^2.

In this case $\bar{u}$ is an unbiased estimator of $E(u_i)$, and $h(u_i)$ may be defined to be $(u_i - \bar{u})L + \bar{u}$, where $L = diag(1/\sqrt{g_1} + \ldots + 1/\sqrt{g_p})$, and $q_i = (u_i - \bar{u})L$. Also, when the random variables U_{1j}, U_{2j}, $\ldots$, U_{nj} are mutually independent, $Q_{ij} \sim S(0, \sigma^2 + \sigma^2/n^2 - 2\sigma^2/n)$. Now the best-fitting subspace S_t is still $\{\, v \mid v(I - P(t)P'(t)) = 0 \,\}$ and in terms of our original points $u_1, u_2, \ldots, u_n$, the best-fitting affine t-plane, A_t, for which the sum of the weighted squares of the distances from the points u_1, $\ldots$, u_n to the t-plane is minimal is $\{\, u \mid (u - \bar{u})L(I - P(t)P'(t)) = 0 \,\}$, where the sum which is minimized is:

$$\sum_{i=1}^{n} \parallel (u_i - \bar{u})LP(t)P'(t) \parallel^2 = \sum_{i=1}^{n} (u_i - \bar{u})LL'(u_i - \bar{u})' = \lambda_{t+1} + \ldots + \lambda_p.$$

In particular, the equation of the plane A_{p-1} is $ua^* = d^*$ where $a^* = LP_p'$ and $d^* = \bar{u}a^*$. However the model $E(z)a = d$ is not very interesting, since there are many values for a and d which satisfy $E(z)a = d$, namely for each affine plane which contains the point $E(z)$, there are corresponding a and d values. The method above selects that plane which neglects the least important principal component in the data.

In the special case where $u_i \sim z$ and $z \sim N(\mu, V)$, L becomes $diag(1/\sqrt{V_{11}} + \ldots + 1/\sqrt{V_{pp}})$, and $q_i = (u_i - \bar{u})L \sim N(0, LVL')$. In this case we can specify an asymptotic covariance matrix for an eigenvector P_i of $Q'Q$ as:

$$cov(P_i) \to (v_i/n) \sum_{\substack{h=1 \\ h \neq i}}^{n} R_i' R_i v_h/(v_h - v_i)^2 \quad \text{as} \quad n \to \infty,$$

where $v_1 > v_2 > \ldots > v_p$ are the eigenvalues of LVL' and $R_1, \ldots, R_p$ are the corresponding eigenvectors. See *Multivariate Analysis* by Morrison, Wiley, 1968. Thus an approximate covariance matrix for P_i can be computed as:

$$cov(P_i) \approx (\lambda_1/n) \sum_{\substack{h=1 \\ h \neq i}}^{p} P_i' P_i \lambda_h/(\lambda_h - \lambda_i)^2$$

so that $cov(a^*) \approx L(cov(P_p))L'$, and $Var(d^*) \approx \bar{u}(cov(a^*))\bar{u}'$.

3.8 Multivariate Stochastic Models

Suppose that we have a matrix of observations, $\tilde{U}$, whose rows are samples of a random vector z. We have observed that in this case, a model such as $E(z)a = d$ is uninteresting. It is most useful to study the joint-distribution of z. Thus we would like to use the data $\tilde{U}$ to estimate the parameters in the joint-distribution function of z, assuming we can choose the form for the distribution function F_z either by physical or probabilistic considerations. The most common choice is the multivariate normal distribution.

Thus, we suppose we have an $n \times p$ matrix, $\tilde{U}$, whose rows $\tilde{u}_1, \ldots, \tilde{u}_n$, are samples of the random vectors $u_1, \ldots, u_n$, which form the rows of the random matrix U, where $u_i \sim z$ for $1 \leq i \leq n$, and $z \sim N(\mu, V)$. We have $V_{ij} = cov(z_i, z_j)$ and $\mu_i = E(z_i)$. Then, when V is non-singular, z has the density function $f_z(t) = (2\pi)^{-p/2} det(V^{-1})^{1/2} \exp(-(t-\mu)V^{-1}(t-\mu)'/2)$. Whether V is non-singular or not, the maximum-likelihood estimators for μ and V are $\hat{\mu} = \sum_{i=1}^{n} u_i = \bar{1}'U/n$ and $\hat{V} = U'U/n - \hat{\mu}\hat{\mu}'$. $\hat{\mu}_t$ and $\hat{V}_{ij}$ are independent random variables.

Moreover $\hat{\mu}_t \sim N(\mu_t, V_{tt})$ and $\hat{V}_{ij} \sim \chi^2(n-1)$. The joint distribution of the elements of $\hat{V}$ is known to be a Wishart$(n-1, V, 0)$ distribution. Thus joint confidence regions for the elements of $\hat{V}$, and, separately, of $\hat{\mu}$, can be established.

When the maximum likelihood estimator $\hat{V}$ is singular, it is most appropriate to transform the data to lie in a space of reduced dimension, wherein the new covariance matrix estimator is non-singular.

Note if the off-diagonal element $V_{ij} = 0$, then the random variables z_i and z_j are independent.

Also there exists a column vector, b_i, such that $E(z_i|(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_p) = x) = [1\ x]b_i$, so the various conditional expectations are linear functions, and the coefficients $(b_{i0}, b_{i1}, \ldots, b_{ik}) = b_i'$ can be estimated using least-squares estimation. Let V_i denote the ith row of V, and let $V|i$ denote the matrix V with the ith column deleted. We have $E(z_i|(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_p) = x) = \mu_i + (V_i|i)((V'|i)'|i)^{-1}(x - \mu|i)' = [1\ x]b_i$. If the parameter vectors $b_1, b_2, \ldots, b_n$ are known, the joint normal distribution, F_z is determined, since μ and V are determined by the p equations:

$$E(z_i|(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_p) = x) = [1\ x]b_i.$$

When V is non-singular, the density function equation $f_z(t) = c$ is an ellipsoid in n-space whose center is at the point μ, and whose principal axes are of lengths:

$$(\rho/\lambda_p)^{1/2} \geq (\rho/\lambda_{p-1})^{1/2} \geq \ldots \geq (\rho/\lambda_1)^{1/2}, \quad \text{where} \quad \rho = -2log(c/((2\pi)^{-p/2} det(V^{-1})^{1/2})),$$

and where $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p$ are the (positive real)eigenvalues of V^{-1} (and the reciprocals of the eigenvalues of V). The principal axes are mutually orthogonal and the direction of the principal axis of length $(\rho/\lambda_i)^{1/2}$ is the same as the direction of the eigenvector of V^{-1} corresponding to the eigenvalue, λ_i.

By using $\tilde{\hat{V}}$ for V, and $\tilde{\hat{\mu}}$ for μ, an approximate $100\alpha\%$ joint confidence ellipsoid for z can be defined by taking the ellipse $\{\, t \mid f_z(t) = c \,\}$, where $\alpha = \int_{t \in \{\, s | f_z(s) \geq c \,\}} f_z(t)$.

For $p = 2$, we have $c = (1-\alpha) det(\tilde{\hat{V}}^{-1})^{1/2}/(2\pi)$.

Also, for $p \geq 1$, we have $(z-\hat{\mu})\hat{V}^{-1}(z-\hat{\mu})' \sim (n+1)pF(p, n-p, 0)/(n-p)$, so we may solve the equation: $P(F(p, n-p, 0) \leq (n-p)r/((n+1)p)) = \alpha$, for r, and take the ellipsoid $\{\, t \mid (t-\tilde{\hat{\mu}})\tilde{\hat{V}}^{-1}(t-\tilde{\hat{u}})' = r \,\}$ as an approximate $100\alpha\%$ joint confidence ellipsoid in p-space.

3.9 Practical Procedures

Given explicit data, we generally do not know which of various hypotheses used in fitting a linear model are in fact true. In spite of this, we can proceed and often obtain suggestive results of practical value.

Suppose we are given an X matrix and corresponding sample values $\tilde{y}$ for y. Then we first assume the linear model $y = Xb + e$ with $E(e) = 0$ and $cov(e) = V$.

If $X'V^{-1}X$ is singular, we use the Moore-Penrose generalized inverse and, although we may compute $\tilde{\hat{b}}$, we can usefully employ it only to estimate estimable functions.

Suppose, instead, that $X'V^{-1}X$ is non-singular. We now must decide if the model is *correct*. This means we must decide whether, in fact, $E(e) = 0$. If the linear model has physical significance, so that accepted physical principles dictate the model, and if the data is not subject to measurement bias, then the model *is* correct. If, however, we have a stochastic situation involving random and unknown factors, then we can never know the model is correct, but we can check to see if it seems obviously incorrect. To do this, examination of the residuals, e_i, determined by the $\tilde{y}_i$-values is most appropriate, where $\tilde{e} = \tilde{y} - X\tilde{\hat{b}}$. Although $\sum_{i=1}^{n} \tilde{e}_i = 0$ necessarily, any other non-normal-random behavior among the $\tilde{e}_i$-values is grounds for suspicion.

If, in fact, $cov(e) = I\sigma^2$, then if $E(e) = 0$ then $E(\hat{\sigma}^2) = \sigma^2$, otherwise $E(\hat{\sigma}^2) > \sigma^2$. Thus, if we can independently determine σ^2, say by repeated experiments, then we can compute $\tilde{\hat{\sigma}}^2$, based on $\tilde{y}$ $\tilde{\hat{\sigma}}^2$, and if $\tilde{\hat{\sigma}}^2 > \sigma^2$, with significance as determined by a hypothesis test, then we may reject the model as being correct.

Another problem arises if the values in the X matrix are not precisely known, but are themselves

sample values of various random variables. In this case, as discussed above, the standard analysis given here is not valid, and another approach is required.

If however, we decide the model is correct, and if the elements of X are precisely determined, and $X'V^{-1}X$ is non-singular, then $\tilde{\hat{b}}$ is the b.l.u.e. estimate of b, and the estimator $\hat{b}$ has $cov(\hat{b}) = (X'V^{-1}X)^{-1}$.

Now, if in addition $e \sim N(0, I\sigma^2)$, then $\hat{b} \sim N(b, (X'X)^{-1}/\sigma^2)$, so we can determine confidence intervals for b_i, and we can test the hypothesis $b = \tilde{\hat{b}}$.

The assumption $e \sim N(0, I\sigma^2)$ can itself be tested by examination of $\tilde{e}$; we may test the hypothesis $e = \tilde{e}$ using e as a test-statistic.

3.10 An MLAB Example

Now let us consider an example using MLAB. This example is set up to be used as a tutorial by typing in the specified commands. Suppose we have the observations:

$$\begin{array}{l} x_1 = 2 \\ x_2 = 3 \\ x_3 = 4 \\ x_4 = 5 \\ x_5 = 6 \\ x_6 = 7 \\ x_7 = 8 \\ x_8 = 9 \end{array}, \quad \text{and} \quad \begin{array}{l} \tilde{y}_1 = 0.678 \\ \tilde{y}_2 = 1.46 \\ \tilde{y}_3 = 2.68 \\ \tilde{y}_4 = 4.16 \\ \tilde{y}_5 = 5.99 \\ \tilde{y}_6 = 8.17 \\ \tilde{y}_7 = 10.7 \\ \tilde{y}_8 = 13.5 \end{array}.$$

Let us use the linear model $y = Xb + e$, with $n = 8$, and with 2 parameters, so that $k = 1$. Let $x_{i0} = 1$, and $x_{i1} = x_i$ for $1 \le i \le 8$. Then

$$X = \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \\ 1 & 6 \\ 1 & 7 \\ 1 & 8 \\ 1 & 9 \end{bmatrix} \quad \text{and} \quad \tilde{y} = \begin{bmatrix} 0.678 \\ 1.46 \\ 2.68 \\ 4.16 \\ 5.99 \\ 8.17 \\ 10.7 \\ 13.5 \end{bmatrix}.$$

We may enter this data in MLAB by typing:

```
*X = 1&'(2:9)
*YS = KREAD(8)
[.678, 1.46, 2.68, 4.16, 5.99, 8.17, 10.7, 13.5]
```

We use the name `YS` (meaning "y samples") for $\tilde{y}$. Suppose also that $cov(e) = I\sigma^2$ with $\sigma = 1$. Now we may compute $\tilde{\hat{b}}$ (which we call `BE` for "b estimates"), and cov($\hat{b}$). Type

```
*BE = (X'*X)^(-1)*X'*YS
*TYPE BE
   BE: a  2 by 1 matrix

  1: -4.18271429
  2: 1.83635714
*COVBE = (X'*X)^(-1)
*TYPE COVBE
   COVBE: a  2 by 2 matrix

  1: .845238095    -.130952381
  2: -.130952381   2.38095238E-2
```

Let us check to see if $X'X$ is non-singular. Type:

```
*TYPE(X'*X)^(-1)*(X'*X)
    : a  2 by 2 matrix
  1: 1  -2.81441537E-13
  2: 6.23112673E-15 1
```

We have a near-identity matrix, so $X'X$ is numerically non-singular. Thus $\hat{b} = X^{-1}y$ is the b.l.u.e. of b.

When $X'X$ is singular, the least-squares solution to $Xb = y$ is not unique, but $X^{-1}y$ is always a solution when we use the Moore-Penrose generalized inverse, and moreover it is the solution of minimum Euclidean norm. Thus, we can always use X^{-1} in MLAB for $(X'X)^{-1}X'$. Of course the MLAB FIT statement will provide the same result. We can check this by typing:

```
*TYPE X^(-1)*YS
    : a  2 by 1 matrix

  1: -4.18271429
```

```
  2: 1.83635714

*FUNCTION YF(X) = B0+B1*X
*B0 = 1; B1 = 1

*FIT(B0,B1), YF TO (X COL 2)&'YS
final parameter values
value               error                 dependency        parameter
   -4.182714279        0.8141114254          0.8521126761   B0
    1.836357142        0.1366374264          0.8521126761   B1
2 iterations
CONVERGED
best weighted sum of squares = 4.70479e+00
weighted root mean square error = 8.85512e-01
weighted deviation fraction = 7.64468e-02
R squared = 9.67850e-01

*TYPE COVP
   COVP: a  2 by 2 matrix

  1: .662777413    -.102683825
  2: -.102683825   1.86697863E-2
```

Note COVP has been scaled by $\tilde{\tilde{\sigma}}^2$. Now let us examine the residuals $\tilde{e}$ (which we call ER). Type:

```
*YE = X*BE
*ER = YE-YS
*TYPE ER
   ER: a  8 by 1 matrix

  1: -1.188
  2: -.133642857
  3: .482714286
  4: .839071429
  5: .845428571
  6: .501785714
  7: -.191857143
  8: -1.1555
```

The results are suspicious. Nevertheless, we may proceed. We may estimate σ^2 as follows:

```
*SIGMA2 = ((YS-YE)'*(YS-YE))(1)/(NROWS(X)-NCOLS(X))
```

```
*TYPE SIGMA2
  SIGMA2 = .784131024
```

$\tilde{\hat{\sigma}}^2$ is not approximately 1 as we assumed. Let us correct our estimate of $\text{cov}(\hat{b})$).

```
*COVBE = COVBE*SIGMA2
```

Also, we may compute Cook's $\hat{b}$-influence measures for each data point. The non-randomness of the residuals should be observed.

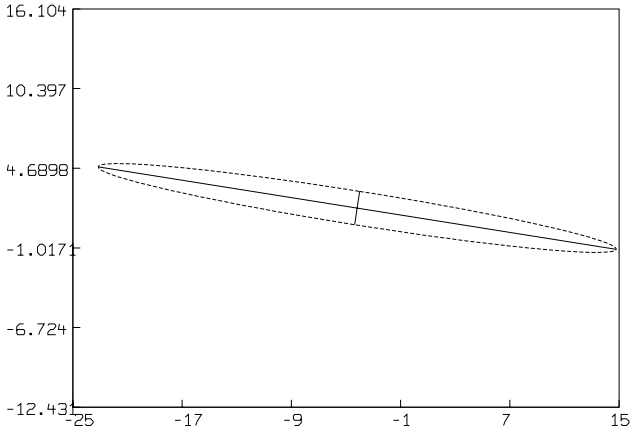
```
*AM = GETDIAG(X*COVBE*X')
*FUNCTION D(I) = (AM[I]/(1-AM[I])^2)*(ER(I)^2)/(SIGMA2/2)
*TYPE POINTS(D,1:NROWS(X))
    : a  8 by 2 matrix

  1: 1   2.59454874
  2: 2   1.58599446E-2
  3: 3   .112524995
  4: 4   .229007796
  5: 5   .232491052
  6: 6   .121592079
  7: 7   .032686326
  8: 8   2.45453286
```

Now we may compute 90 percent confidence intervals for b_0 and b_1, under the further assumption that $e \sim N(0, I\sigma^2)$, with $\sigma^2 = \tilde{\hat{\sigma}}^2$. Note $F^{-1}_{t(6)}(.95)$ is obtained by `stuti`$(.95, 6) = 1.9432$.

```
*HA = STUTI(.95,6)
*L = HA*(SQRT ON GETDIAG(COVBE))
*H = BE+L
*L = BE-L
*TYPE L&'BE&'H
    : a  2 by 3 matrix

  1: -5.76467955 -4.18271429 -2.60074902
  2: 1.57084599 1.83635714 2.1018683
```

Since we have just two parameters, we may graph a 90 percent joint-confidence ellipse in 2-space, based on the data, $\tilde{y}$.

```
*FUNCTION FX(T) = A*COSD(T)
*FUNCTION FY(T) = B*SIND(T)
*P = (ER'*ER*(1+2*HA)/6+YS'*YE)[1]
*L = SVD(X'*X)
*A = SQRT(P/L[1,1]); B = SQRT(P/L[1,2])
*M = (FX ON 0:360:3)&'(FY ON 0:360:3)
*V = L ROW 4:5
*M = (M*V')+((BE')^^NROWS(M))
*DRAW M
*L = V'*'(A & B)
*H = (BE')^^2
*DRAW MESH(H-L,H+L) LINETYPE ALTERNATE
*WINDOW ADJUST WMATCH
*VIEW
```

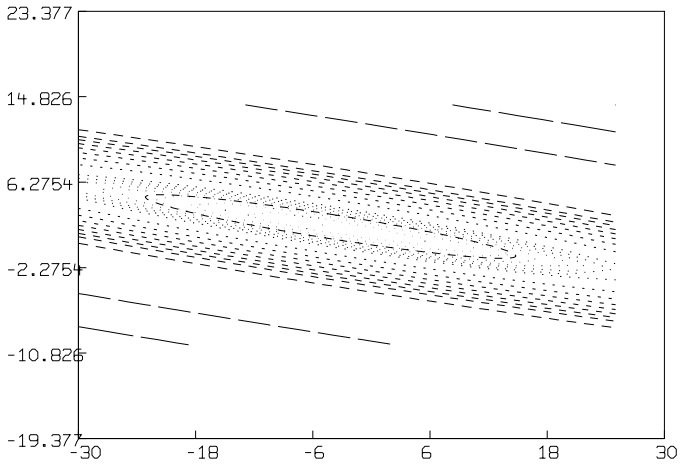


The elliptic bowl-shaped sum-of-squares surface, which has the 90 percent joint confidence region as a contour, can be seen as a contour map as follows.

```
*DELETE W,L,H
*FUNCTION S(B0,B1) = SUM(I,1,NROWS(YS),(X[I,1]*B0+X[I,2]*B1-YS[I])^2)
*CM = POINTS(S,CROSS(-30:25:5, 1:3:.1))
*DRAW CONTOUR(CM,6:1000!8&1200:8000!8&10000:80000!4) LINETYPE VMARKER
```

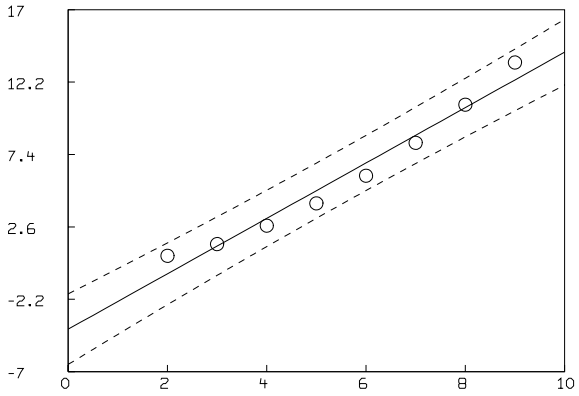
We can add the joint-confidence ellipse as a contour curve

```
*DRAW M LINETYPE DASHED COLOR RED; VIEW
```



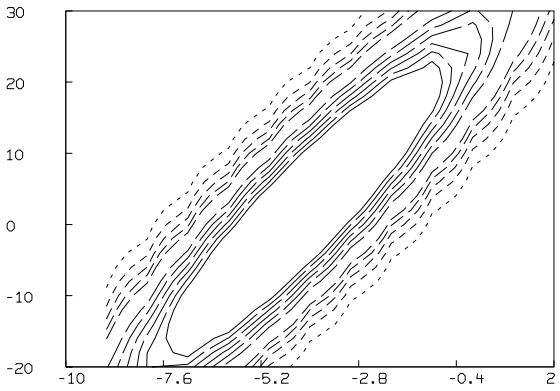
Finally, let us draw a graph of the data and the best-fit regression line with graphs of the bounds of a 90 percent tolerance interval for prediction. Type:

```
*DELETE W
*FUNCTION YC(X) = BE(1)+BE(2)*X
*FUNCTION CX(X) = \
        HA*SQRT(SIGMA2+COVBE[1,1]+X*(COVBE[2,1]+COVBE[1,2])+X*X*COVBE[2,2])
*XV = 0:10:.25
*DRAW (X COL 2)&'YS, LINETYPE 0, POINTTYPE CIRCLE
*DRAW POINTS(YC,XV)
*DRAW LC = XV&'((YC ON XV)-(CX ON XV)) LINETYPE DASHED
*DRAW HC = XV &'((YC ON XV)+(CX ON XV)) LINETYPE DASHED
*VIEW
```



Under the assumption $e \sim N(0, I\tilde{\hat{\sigma}}^2)$, we know that $\hat{b} \sim N(b, (X'X)^{-1}\tilde{\hat{\sigma}}^2)$. We may draw a contour map graph of the density function of $\hat{b}$ as follows, substituting $\tilde{\hat{b}}$ for b.

```
*BLANK W
*V = LIST(COVBE)
*FUNCTION F(S,T) = EXP((V[1]*(S-BE[1])^2+(V[3]+V[2])*(S-BE[1])*(T-BE[2]) \
  +V[4]*(T-BE[2])^2)/(-2))/(2*pi*SQRT(V[1]*V[4]-V[2]*V[3]))
*CM = POINTS(F, CROSS(-11:4:.4,-11:19:1.5)) LINETYPE DASHED
*DRAW CONTOUR(CM,0:2!12) LINETYPE VMARKER
*VIEW
```



Now we may estimate x_s corresponding to a further sample value $\tilde{y}_s = 12.1$, using the calibrated estimator $\hat{x}s = (y_s - \hat{b}_0)/\hat{b}_1$, and establish a 90% confidence interval for $\hat{x}_s$. We assume $y_s \sim N(b_0 + b_1 x_s, \sigma^2)$.

Note $F^{-1}_{F(1,6,0)}(.9)$ is computed in MLAB as `QFI`$(.9, 1, 6)$. It is 3.78.

```
*DELETE W,D
*XS = (12.01-BE[1])/BE[2]
*TYPE XS
  XS = 8.86685596
*HA = QFI(.9,1,6)
*U1 = NROWS(X)*BE[1]^2/SIGMA2
*U = U1-HA
*D = SQRT(HA*((NROWS(X)+1)*U+U1*XS^2))
*TYPE (U1*XS-D)/U,(U1*XS+D)/U
```

```
    = 7.66910227
    = 10.4478702
```

We can compare these bounds to those obtained by a two standard-deviation interval about $\tilde{y}_s$ intersected with the 90% tolerance band for prediction.

```
*FUNCTION XV(Y,M) = LOOKUP(Y,M)
*TYPE XV(12.1+2*SQRT(SIGMA2),CURVEM(HC) COL (2,1))
    = 8.73294709
*TYPE XV(12.1-2*SQRT(SIGMA2),CURVEM(LC) COL (2,1))
    = 9.01899533
```

It is a worthwhile exercise to repeat this entire analysis with the 3-parameter linear model $y_i = b_0 + b_1 x_i + b_2 x_i^2 + e_i$. A joint-confidence region for b is now an ellipsoid in 3-space. The sum-of-squares surface is now graphed in 4-space, and can no longer be drawn.

The question, then, is which is the better model, the 2-parameter or the 3-parameter model? Of course, the 3-parameter model is better, but we are interested in a probabilistic test which can indicate just how much better. Such tests exist, and are the subject of a large amount of literature.

3.11 Non-Linear Regression

Often we have a situation where $E(e) \neq 0$ when we choose $y = Xb + e$. In this case, the linear model is not appropriate, instead, we may have a real-valued function, $f(x; b)$, of vector arguments x and b, such that $y_i = f(x_i; b) + e$ with $E(e) = 0$. The function $f(x_i; b)$ is in general a non-linear function of the parameters $b = (b_0, b_1, \ldots, b_k)'$, as well as the arguments $x_i = (x_{i0}, x_{i1}, \ldots, x_{im})$. Note the number of parameters, $k + 1$, need not be the same as the number of independent variables, $m + 1$.

Our problem is still that of estimating the parameters b, given estimates $\tilde{y}$ of $E(y)$. As before we may define the vector of random variables $\hat{b}$ so that $[\partial S/\partial b](\hat{b}) = 0$ and so that $E(S(\hat{b}))$ is minimal, where $S(b) = (y - y^*)'V(y - y^*)$, with $V = cov(e)$, and $y_i^* = f(x_i; b)$.

There are numerical methods for solving the normal equations $[\partial S/\partial b](\tilde{\hat{b}}) = 0$ for particular estimates $\tilde{\hat{b}}$ based on particular estimates $\tilde{y}$ for $E(y)$. These methods are iterative in nature, and may sometimes fail to converge to a correct solution. Some of these methods are discussed below.

Let us assume, however, that $\hat{b}$ is the random variable such that $E(S(\hat{b}))$ is globally-minimal. Then as before, $\hat{b}$ is the b.l.u.e. of b (is this true?), and if $e \sim N(0, V)$, then $\hat{b}$ is the maximum-likelihood estimator of b as well. The covariances, $cov(\hat{b})$, however, are not obtainable, except by linear

approximations, which may be very poor. Similarly, all the other results for linear regression are merely unreliable approximations in the non-linear case.

For $\mathrm{cov}(e) = I\sigma^2$, we still have the contour of a joint confidence region for b defined by $Q(b) = Q(\hat{b})(1+(k+1)/(n-k-1)v)$ where $Q(b) = S(b)/\sigma^2$, and $v = F^{-1}_{F(k+1,n-k-1,0)}(1-\alpha)$; however the region so defined is not a $1-\alpha$ joint confidence region, but is, instead, a $1-\alpha'$ joint confidence region where α' is unknown. Moreover, the boundary of such a region is not necessarily an ellipse.

If we have a set-valued random-variable, C, which takes subsets of $(k+1)$-space as values, such that $P(b \in C) \geq 1-\alpha$, then the true model function, $f(x;b)$ satisfies $P(\min_{a\in C} f(x;a) \leq f(x;b) \leq \max_{a\in C} f(x;a)) \geq 1-\alpha$ for all vectors x. As a practical matter, we can often use this result to draw suggestive confidence bands for prediction of a value $\tilde{y}s = f(x_s;b)$, given the vector x_s, but of course, a value of C corresponds to an unknown value of $1-\alpha$ in general.

Also, if we have, in addition to C, a further interval-valued random variable, J, such that $P(b \in C$ and $E(y_s) \in J) \geq 1-\alpha$, then for any function $g(y;b)$ of the scalar argument y and the vector of parameters b, we have

$$P(\min_{\substack{z\in J\\ a\in C}} g(z,a) \leq g(E(y_s);b) \leq \max_{\substack{z\in J\\ a\in C}} g(z,a)) \geq 1-a.$$

This provides a $1-\alpha$ confidence interval for $g(E(y_s);b)$, in terms of C and J. In particular, if we wish to estimate $g(E(y_s);b)$ by using $\tilde{\hat{b}}$ to calibrate g, and then use a later sample value $\tilde{y}_s$ to compute the estimate $g(\tilde{y}_s, \tilde{\hat{b}})$, then the inequality above can be used with C and J taken as estimated $1-\alpha$ confidence regions for b and y_s to obtain suggestive bounds for $g(E(y_s);b)$. The result is a $1-\alpha'$ confidence interval, but α' is generally unknown.

The most common choice of g is the inverse of the model f, with $g(y_s;b) = root_{v_1\leq x\leq v_2}(f(x;b)-y_s)$, where x is a scalar, and a unique root in the interval v_1 to v_2 is assumed to exist. Note for any choice of g, the random variable $g(y_s;\hat{b})$ is a consistent estimator of $g(E(y_s);b)$. If $cov([y\ y_s]) = I\sigma^2$ or if g is a linear function of y_1, y_2, ..., y_n, and y_s, then $g(y_s;\tilde{\hat{b}})$ is unbiased.

3.12 Solving Non-Linear Normal Equations

To solve $\partial S/\partial b = 0$, treating the random vector y as constant, we may expand $\partial S/\partial b$ in a Taylor series about a point $b^{(j)}$. Thus, we linearize the normal equations, to obtain:

$$0 = [\partial S/\partial b](b^{(j)} + \beta^{(j)}) = M^{(j)}\beta^{(j)} + w^{(j)} + O(\beta^{(j)\prime}\beta^{(j)})$$

where $M^{(j)} = [\partial^2 S/\partial b^2](b^{(j)}), w^{(j)} = [\partial S/\partial b](b^{(j)})$, and $\beta^{(j)} = (\beta_0^{(j)}, \ldots, \beta_k^{(j)})'$.

Now the *Newton-Raphson procedure* is the iteration formula:

$b^{(j+1)} = b^{(j)} - \pi(M^{(j)})^{-1}w^{(j)}$, where π is a parameter, usually near 1. Under appropriate conditions this iteration will converge to a vector, b^*, such that $[\partial S/\partial b](b^*) = 0$.

The *Method of Steepest Descent* is: $b^{(j+1)} = b^{(j)} - \pi w^{(j)}$, where π is the minimum of S along the antigradient, given by $\partial S(b^{(j)} - \pi w^{(j)})/\partial\pi = 0$.

Davidon's method is: $b^{(j+1)} = b^{(j)} - \pi H^{(j)}w^{(j)}$, where π is determined as in the method of steepest descent, and $H^{(j)}$ is a positive-definite matrix such that $H^{(j)}w^{(j)} - w^{(j-1)} = b^{(j)} - b^{(j-1)}$, and $H^{(0)} = I$.

Another approach to solving $\partial S/\partial b = 0$ is to *linearize the model* to obtain linear normal equations. The *Gauss-Newton procedure* uses this idea iteratively. We have the model $y_i = f(x_i; b) + e_i$, and we may expand f in a Taylor series to obtain:

$f(x; b + \beta) \approx f(x, b) + (\partial f(x; b)/\partial b)'\beta$, where $\beta = (\beta_0, \ldots, \beta_k)'$.

Define $g(x; \beta) = (\partial f(x; b)/\partial b)'\beta$. Now, for any point b, we suppose β is the correction vector such that $f(x_i, b + \beta) = E(y_i)$. Thus $g(x_i; \beta) \approx E(y_i) - f(x_i; b)$, so $\beta \approx (X'V^{-1}X)^{-1}X'V^{-1}(E(y) - (f(x_1; b), \ldots, f(x_n; b))')$, where $X_{st} = \partial f(x_s; b)/\partial b_t$, with $1 \le s \le n$, and $0 \le t \le k$. Now β depends upon the guess, b. Thus we have the iteration: $b^{(j+1)} = b^{(j)} + \beta^{(j)}$, where $\beta^{(j)} = (X'V^{-1}X)^{-1}X'V^{-1}(E(y) - (f(x_1; b^{(j)}), \ldots, f(x_n; b^{(j)}))')$, with $X_{st} = \partial f(x_s; b^{(j)})/\partial b_t$.

The matrix $X'V^{-1}X$ may be singular or ill-conditioned; moreover the approximation $f(x; b+\beta) \approx f(x; b) + g(x; \beta)$ may hold only for vectors, β, which are very small. The *Marquardt-Levenberg method* is a variation of the Gauss-Newton procedure which deals with these difficulties.

Let D be a diagonal $n \times n$ matrix with $D_{ii} = (X'V^{-1}X)_{ii}$. Now define:

$$\beta = (X'V^{-1}X + \varepsilon D)^{-1}X'V^{-1}(E(y) - (f(x_1; b), \ldots, f(x_n; b))').$$

The matrix $X'V^{-1}X + \varepsilon D$ is non-singular for $\varepsilon > 0$ and its condition depends upon ε.

Now, the Marquardt-Levenberg iteration is:

$$b^{(j+1)} = b^{(j)} + \beta^{(j)},$$

where $\beta^{(j)} = (X'V^{-1}X + \varepsilon D)^{-1}X'V^{-1}(E(y) - (f(x_1; b^{(j)}), \ldots, f(x_n; b^{(j)}))')$,
with $X_{st} = \partial f(x_s; b^{(j)})/\partial b_t$, and $D_{st} =$ if $s = t$ then $(X'V^{-1}X)_{st}$ else 0.

For $\varepsilon = 0$, we have the Gauss-Newton procedure, while for $\varepsilon \to \infty$, we obtain a vanishing correction vector, $\beta^{(j)}$, in the direction of steepest descent. In the linear case, this diagonal modification is a form of so-called ridge regression, used to promote numerical stability.

MLAB employs the Marquardt-Levenberg method, with $\varepsilon = 10^{-10}$ initially. Rather than keep ε fixed, at each iteration ε is varied so as to seek out a smaller $S(b)$ for b lying on an appropriate

curve between the current estimate of b and the next estimate predicted by the Gauss-Newton iteration. This is a form of line-search with a curved "line". The evaluations of $S(b)$ involved for different trial values of ε constitute subiterations.

When the matrix $X'X$ is ill-conditioned or of deficient rank, the magnified-diagonal property of the Marquardt-Levenberg method usually overcomes this difficulty. This is not always the case however, and we shall consider an example below.

Consider fitting the coefficients of the fifth-degree polynomial $Ax^5+Bx^4+Cx^3+Dx^2+Ex+F = y(x)$ to the exact data points POINTS$(p, 1:10)$, where $p(x) = -2x^5+4x^4+2x^3+4x^2+x+1$.

Let us use the guesses $A = B = C = D = E = 0$ and $F = .5$. Then MLAB will require 18 iterations to converge when fitting for A, B, C, D, E, and F. This is because the matrix $X'X$ is ill-conditioned

In this case the matrix X should be used directly, and its Moore-Penrose inverse employed to estimate the coefficients. In MLAB we have, for this example:

```
*FCT P(X) = -2*X^5+4*X^4+2*X^3+4*X^2+X+1
*V = 1:10
*X = 1&'V&'V*'V
*X = X&'(V*'V*'V)*'X
*F = P ON V
*TYPE (X^-1)*F
```

The typed-out vector elements are the estimates of the parameters F, E, D, C, B, and A, which match the true coefficients exactly.

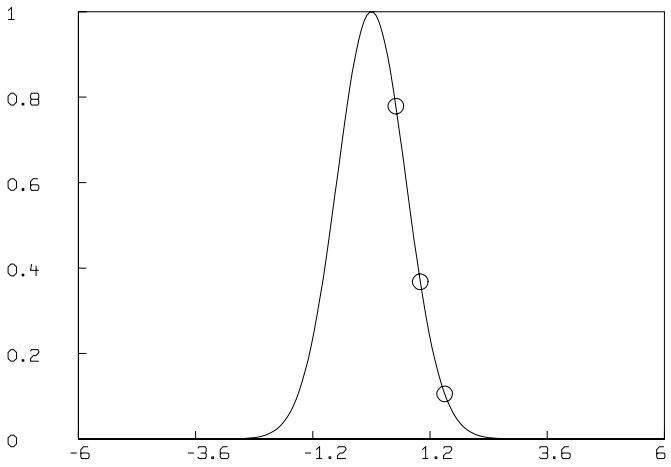
No method can guarantee, in general, that it will produce a solution, b^*, to $\partial S/\partial b = 0$ such that $S(b^*)$ is a global minimum. Indeed, b^* may be produced for which $S(b^*)$ is not even locally-minimal. We shall discuss these difficulties below.

3.13 Gaussian Fit Example

Let us consider a simple model with exact data which has a sum-of-squares function with two local minima. The function $F(X) = EXP(-(X-M)\text{^}2)$ is a Gaussian error function with one parameter, namely M, the point at which F is maximum. We shall define this function in MLAB, and generate three data points with $M = 0$.

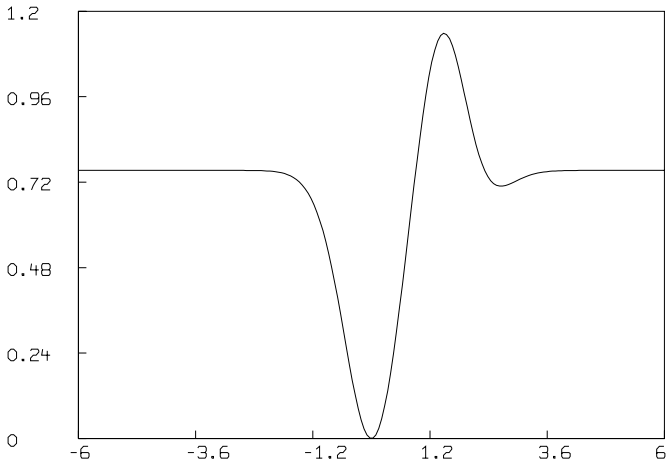
```
*FUNCTION F(X)=EXP(-(X-M)^2)
*M = 0
```

```
*D = POINTS(F,.5:1.5!3)
*TYPE D

D: A 3 BY 2 MATRIX
1: 0.5 0.7788008
2: 1.0 0.3678794
3: 1.5 0..1053992

*DRAW D POINTTYPE CIRCLE LINETYPE NONE
*V = -6:6!200
*DRAW POINTS(F,V)
*VIEW
```



Now, given the function F and the data, D, we can define the sum-of-squares function, SSQ. This is a function of the single parameter, M, so the sum-of-squares surface is merely a curve in the xy-plane.

```
*DELETE W
*FUNCTION SSQ(M)=SUM(I,1,NROWS(D),(D(I,2)-F1(D(I,1),M))^2)
*FUNCTION F1(X,M)=EXP(-(X-M)^2)
*DRAW POINTS(SSQ,V)
*VIEW
```

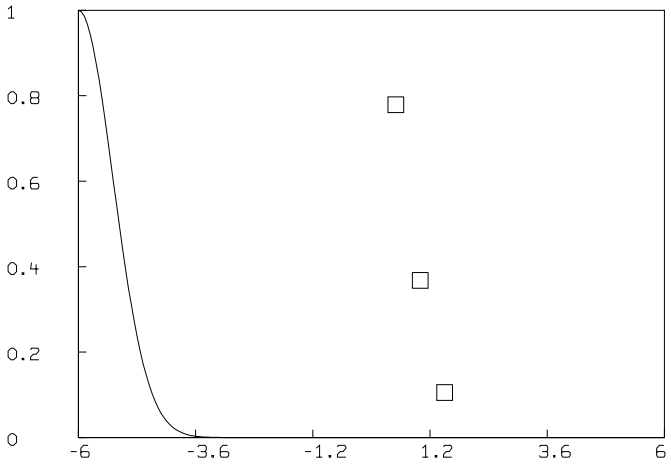



```
*DELETE W
```

Let us guess $M = -6$, and fit to find the true value of M (which we know is 0).

```
*M = -6; MAXITER = 5
*FIT(M),F TO D
matherr: underflow error in exp
         arg = -1.47961e+34
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -1.47961e+34
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -1.47961e+34
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -2.27135e+32
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -2.27135e+32
         return value: 0.00000e+00
final parameter values
value   error            dependency   parameter
 -6     9.577661785e+16       0       M
```

```
1 iterations
CONVERGED
best sum of squares = 7.52975e-01
root mean square error = 6.13586e-01
deviation fraction = 1.00000e+00

*DRAW D,LINETYPE  NONE POINTTYPE SQUARE
*DRAW C = POINTS(F,V)
*VIEW
```



Note the curve-fitter did not change M at all! The reason is that the slope of the sum-of-square surface, SSQ, is so close to zero at $M = -6$ that, within the convergence factor specified, the program treats the slope as zero and stops at a false minimum. Let us try decreasing the convergence factor and see if the slope can be detected.

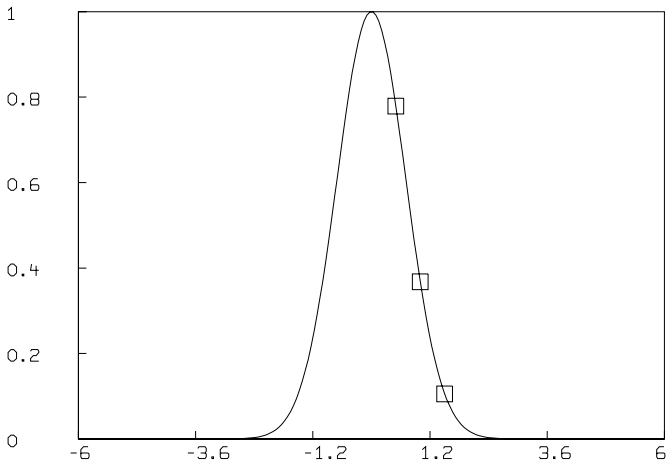
```
*TOLSOS = 1E-7; MAXITER = 12
*FIT(M),F TO D
matherr: underflow error in exp
         arg = -1.47961e+34
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -1.47961e+34
         return value: 0.00000e+00
matherr: underflow error in exp
```

```
          arg = -1.47961e+34
          return value: 0.00000e+00
 matherr: underflow error in exp
          arg = -2.27135e+32
          return value: 0.00000e+00
 matherr: underflow error in exp
          arg = -2.27135e+32
          return value: 0.00000e+00
 final parameter values
 value   error           dependency    parameter
 -6      9.577661785e+16 0             M
 1 iterations
 CONVERGED
 best sum of squares = 7.52975e-01
 root mean square error = 6.13586e-01
 deviation fraction = 1.00000e+00
```

The change in the slope of SSQ at M = -6 is so small as to be unrepresentable in 64-bit floating point form. Let us try a better guess.

```
 *M = -2; TOLSOS = 0.001
 *FIT(M),F TO D
 matherr: underflow error in exp
          arg = -6.45673e+03
          return value: 0.00000e+00
 matherr: underflow error in exp
          arg = -6.37663e+03
          return value: 0.00000e+00
 matherr: underflow error in exp
          arg = -6.29702e+03
          return value: 0.00000e+00
 final parameter values
 value              error   dependency         parameter
 1.637806653e-17      0     -.110223025e-16    M
 8 iterations
 CONVERGED
 best sum of squares = 0.00000e+00
 root mean square error = 0.00000e+00
 deviation fraction = 0.00000e+00
 R squared = 1.00000e+00

 *DRAW C = POINTS(F,V)
```

```
*VIEW
```



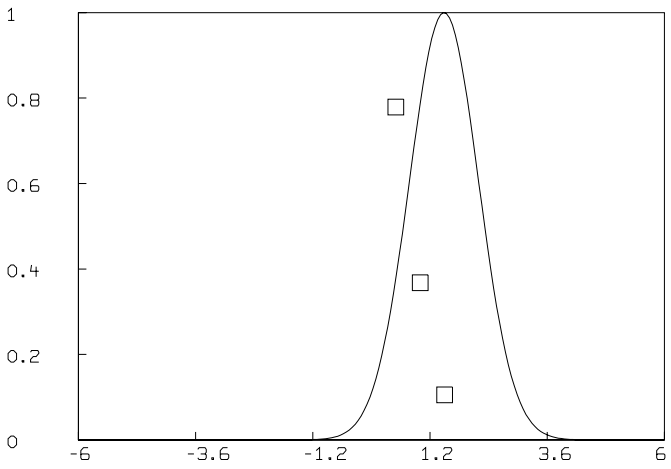
This time we did obtain the solution. Now let's try an even better guess.

```
*M = 1; MAXITER = 12
*FIT(M),F TO D
final parameter values
value             error  dependency          parameter
 9.257147744e-18  0      -.110223025e-16     M
7 iterations
CONVERGED
best sum of squares = 0.00000e+00
root mean square error = 0.00000e+00
deviation fraction = 0.00000e+00
R squared = 1.00000e+00
```

Here we again rolled easily along the sum-of-squares curve to fall into the correct "pocket". Now let us try $M =$ the value which maximizes the sum-of-squares function. This is, in some sense, a worse guess.

```
*M = 1.48609
*FIT(M),F TO D
final parameter values
```

```
value            error            dependency        parameter
1.486062058      0.7047323882     -1.110223025e-16  M
1 iterations
CONVERGED
best sum of squares = 1.13827e+00
root mean square error = 7.54410e-01
deviation fraction = 7.45560e-01
*DRAW C = POINTS(F,V)
*VIEW
```



This time we failed to move away from the region of the initial guess. The reason is that the slope of SSQ at $M = 1.48609$ is (nearly) 0. MLAB will never move to a maximum, but if a maximum is given as the initial guess, MLAB will erroneously stay there, since second derivatives of SSQ are not used.

Now, let us try a guess for M nearby 1.48609.

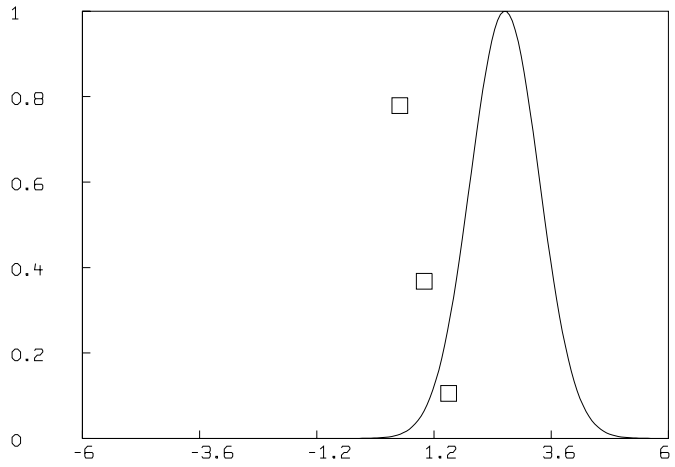
```
*M = 1.5
*FIT(M),F TO D
final parameter values
value          error           dependency    parameter
1.515379992    0.7039011029    0             M
1 iterations
CONVERGED
```

```
best sum of squares = 1.13718e+00
root mean square error = 7.54050e-01
deviation fraction = 7.56686e-01
```

This time we changed M a little bit and then stopped. The reason is because the sum-of-squares changed by less than a factor of 0.001 (i.e. by less than .1 percent), and this is our criterion for "flatness", and hence for stopping.

Let us try this fit again with a more stringent convergence criterion.

```
*M = 1.5; MAXITER = 16; TOLSOS = 1E-5
*FIT(M),F TO D
final parameter values
value          error           dependency    parameter
2.651114315    0.9125040853    0             M
13 iterations
CONVERGED
best sum of squares = 7.08567e-01
root mean square error = 5.95217e-01
deviation fraction = 9.65594e-01

*DRAW C = POINTS(F,V);
VIEW
```



Now we moved alright, but we moved to the false solution $M = 2.65147$. Our guess, 1.5, was so bad, it was "nearer" the second minimum of SSQ than the first (correct) minimum. Note the resulting fit has no distinguishing characteristics; it is neither the worst nor the best. Indeed, it is a "mathematical accident"!

Let us now try a guess back on the "right" side of 1.48609, and again use a stringent convergence factor.

```
*M = 1.475; MAXITER = 16; TOLSOS = 1E-5
*FIT(M),F TO D
final parameter values
value           error   dependency       parameter
1.80080061e-17  0       -.110223025e-16  M
12 iterations
CONVERGED
best sum of squares = 0.00000e+00
root mean square error = 0.00000e+00
deviation fraction = 0.00000e+00
R squared = 1.00000e+00
```

As expected, we did converge to the correct answer, but initially the convergence was very slow. Later, as we got sufficiently near $M = 0$, the sum-of-squares values rapidly decreased at each iteration, exhibiting the quadratic convergence properties of the Gauss-Newton procedure. (You can check this behavior by setting `lsqrpt` = 15 and redoing the fit.)

One can fall into the false minimum from many places. Indeed, for any initial guess greater than 1.487, where the slope of SSQ is still sufficiently large, we will converge to the false result, $M = 2.65147$. If M is guessed to be too large however, we have the same problem as with our guess of $M = -6$ before. Consider guessing $M = 6$.

```
*M = 6
*FIT(M),F TO D
matherr: underflow error in exp
         arg = -5.20816e+13
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -5.20816e+13
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -5.20816e+13
         return value: 0.00000e+00
```

```
matherr: underflow error in exp
         arg = -7.99498e+11
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -7.99499e+11
         return value: 0.00000e+00
final parameter values
value   error           dependency    parameter
6       40619416.6      0             M
1 iterations
CONVERGED
best sum of squares = 7.52975e-01
root mean square error = 6.13586e-01
deviation fraction = 1.00000e+00
```

As predicted, we stick at $M = 6$ because the slope is too small. One way to deal with this is to transform the model and the data as shown below, and then fit, hoping that the new sum-of-squares function is more nicely behaved

```
*FUNCTION LOGF(X)=-(X-M)^2
*LOGD = (D COL 1)&'(LOG ON D COL 2)
*TYPE LOGD
LOGD: A 3 BY 2 MATRIX
1: .5000000 -.2500000
2: 1.000000 -1.000000
3: 1.500000 -2.250000
*M = 6; MAXITER = 16
*FIT(M),LOGF TO LOGD
final parameter values
value             error  dependency    parameter
-3.83356279e-17   0      0             M
14 iterations
CONVERGED
best sum of squares = 0.00000e+00
root mean square error = 0.00000e+00
deviation fraction = 0.00000e+00
R squared = 1.00000e+00
```

The transformation did its job (But note the slow rate of convergence.) If we were dealing with real data with error, we would use this device to "refine" our guess, and then fit the original model and data to finish off because we usually want to keep the error in the data unmodified. It is an

interesting exercise to draw a graph of the sum-of-squares function for the model, `LOGF`, and the data, `LOGD`.

You may wish to experiment to answer the following questions. (1)Take $D = 0\&'0$. What does the sum-of-squares curve look like as a function of M? What happens when you try to fit F to this one-point matrix? (2) Set $M = 0$ and then set `D=POINTS(F,-1&-.5&.5&1)`. Again, what does the sum-of-squares curve look like? Try fitting F to this matrix, D. What happens and why?

The problem of errors in the x-values can be handled in the non-linear model case when we may assume each observation point $[x_i, y_i] = (z_{i,1}, \ldots, z_{i,m+1}, z_{i,m+2})$ to be distributed as $N(0, V)$. Then, following H.I. Britt and R.N. Luecke, "The estimation of parameters in nonlinear implicit models", Technometrics, Vol.15, No.2, pp.233:247, May 1973, we may use maximum likelihood estimation to obtain an estimate of b. This involves solving a minimization problem with non-linear constraints which may be iteratively attacked by an approach given by Britt and Luecke.

An often-satisfactory approximation may be employed which allows us to use weighted least-squares estimation. Suppose that $y_i = f(x_i - h_i; b) + e_i$, where h_i is a random vector $(h_{i0}, h_{i1}, \ldots, h_{im})$ with $E(h_{ij}) = 0$ for $0 \leq j \leq m, \mathrm{cov}(h_i) = \gamma I$, and e_i is a random variable with $E(e_i) = 0$, and $cov(e_i, h_{ij}) = 0$ for $1 \leq i \leq n$ and $0 \leq j \leq -m$. Also suppose we are given the data $(x_1, \tilde{y}_1), \ldots, (x_n, \tilde{y}_n)$ and we wish to estimate the parameters $b_0, b_1, \ldots, b_k$.

Then expanding f in a multivariate Taylor series about x_i yields:

$$y_i = e_i + f(x_i) + (\partial_1 f(x_i), \partial_2 f(x_i), \ldots, \partial_{m+1} f(x_i))((x_i - h_i) - x_i)' + O(||h_i||2),$$

where $\partial_j f$ denotes the partial derivative of f with respect to its jth argument. Thus $y_i \approx e_i + f(x_i) - \partial_1 f(x_i) h_{i0} - \partial_2 f(x_i) h_{i1} - \ldots - \partial_{m+1} f(x_i) h_{im}$, and $Var(y_i) \approx Var(e_i) + \sum_{j=0}^{m} Var(h_{ij})(\partial_{j+1} f(x_1))^2$ and these values can be used in the covariance matrix used to form the sum-of-squares function $S(b) = (y - y^*)' V (y - y^*)$, where $y_i^* = f(x_i; b)$ and $V_{ij} = \delta_{ij} / Var(y_i)$. When $Var(h_{ij}) = \gamma$, we have $Var(y_i) \approx Var(e_i) + \gamma ||g_f(x_i)||^2$, where g_f is the gradient of f. Now estimating b can be done by minimizing $S(b)$.

3.14 Empirical Regression

It may be that we do not know the form of the true model for the data $(x_i, \tilde{y}_i)$. We may assume a stochastic conditional mean model, so that $E(y|x_i) = f(x_i)$. The function f itself can be approximated by fitting a spline function defined as a piecewise conjunction of polynomials to the data points. Other more elaborate approximations of f can be developed. See, for example: "Nearest neighbor estimators of a nonlinear regression function" by C.J. Stone, in the Proceedings of Computer Science and Statistics: 8th Annual Symposium on the Interface, UCLA, 1975, and "On the nonparametric estimation of regression functions" by Jacqueline K. Benedetti in the Journal of the Royal Statistical Society, series B, Vol.39, No.2, pp.248:253, 1977, and "Non-Parametric Estimation of a Smooth Regression Function" by R. M. Clark in the same journal, Vol. 39, No. 1, pp. 107:113.

4 Multiple Site Binding

The study of how a ligand material, such as a hormone or antibody, binds to one or more kinds of molecular complexes, called sites, is of fundamental importance in biochemistry. Sites are often embedded in cell membranes, and the binding serves to control the behavior of the cell itself. Typically we are interested in the number of distinct kinds of sites and their frequency of occurrence, and also the equilibrium constants for each ligand-site binding reaction which indicates the absolute strength of each such binding reaction.

For example, quantitative analysis of hormone-receptor binding can be easily performed using appropriate software such as MLAB. MLAB is a computer program whose name is an acronym for "modeling laboratory"; it is an interactive system for mathematical modeling, originally developed at the National Institutes of Health. MLAB can fit multiple non-linear models to data points obtained from standard direct-binding or competitive displacement assays. Typical assays involve measuring the competition between radiolabelled and cold ligand in detergent-solubilized membrane preparations or on whole cells. Affinity constants and limit values of binding protein concentrations for single or multiple sites can be computed by fitting saturation curves in MLAB. Output can include Scatchard plots obtained by a suitable transformation.

There are two common categories of data. The first category results from cold displacement of bound labeled ligand where the labelled ligand concentration is held constant, the unlabelled ligand concentration is varied, and the ligand binding is calculated as a function of the proportion of labelled to total ligand concentration. The second category is a straight saturation analysis where the ligand binding is measured as a function of increasing labelled ligand concentration. This latter category of data also arises in fluorescent, ultraviolet, or electrical current studies. In general, MLAB can handle any measurements modeled by linear or nonlinear saturation plots.

In practice, the basic data must be adjusted to account for an experimentally-determined non-specific binding level, specific activity of the labelled ligand, the values for maximal binding of the cellular preparations, and defective non-binding ligand. Replicate data points need not be averaged; they are entered as individual data points which are each included in the curve fitting process. In contrast, curve fitting based on dose averages may require careful attention to weights to obtain a reasonable fit. Experimental researchers rarely have *a priori* knowledge of the validity of individual data points, so each point is generally assigned an unbiased equal weight. However, weighting functions can be differentially applied in MLAB to adjust for differing error percentages, such as may occur in measurements of low levels of radioactivity. Because of the options for flexible weights, MLAB addresses some of the problems inherent in the treatment of experimental data where the number of samples is usually low and the error can be relatively high.

MLAB can be used to analyze ligand-binding studies of the regulation of the number of receptors on the cell surface, where the binding capacity or receptor concentration is the limiting x-intercept on the Scatchard plot. The MLAB program is also useful in determining whether changes in binding by different ligand agonists/antagonists can be attributed to reduced binding affinity of

the defined site/sites (seen as changes in the affinity constant), or to the total loss of binding ability of a class of sites (seen as changes in the binding capacity, and changes in the fit from an $n+1$ to n site model). Similarly, MLAB can aid in comparison studies of a single ligand against different binding proteins that vary by site-directed mutagenesis, or alternative splicing; such analyses can be useful in exploring mechanisms of ligand/receptor interaction, including amino acid and charge requirements.

In addition, post-translational processing and folding of the nascent protein in the endoplasmic reticulum can be studied by estimating the number of active receptor molecules that have reached the cell surface. These studies are of current interest to biochemists, and an accurate method of quantification is important since conclusions about the biological system, which in turn determine the future direction of research, are based on the calculated binding parameters.

In addition to saturation and Scatchard analyses, MLAB can be used in many other curve fitting applications such as kinetic, Michaelis-Menten, Lineweaver-Burke plots, and various statistical analyses. Kinetic analysis involves fitting differential equation models, which is an important core capability of MLAB. Indeed, when there is any doubt about equilibrium data really being taken at equilibrium, one can employ a kinetic model for this data and often obtain useful results, including verifying or rejecting the equilibrium assumption.

4.1 The Mathematics of Multiple Site Binding

Suppose we have a ligand, F, which binds to each of N *independently-acting* sites, S_1, S_2, ..., S_N, which are present in the various concentrations of $S_{10}\,\mu$M, $S_{20}\,\mu$M, ..., and $S_{N0}\,\mu$M respectively. Each reaction $F + S_i \rightleftharpoons B_i$ forms a bound complex B_i, and we shall define κ_i to be the associated equilibrium constant. Let F_0 denote the concentration of ligand present initially, and let F be the concentration of free ligand at equilibrium. Similarly, let S_i denote the concentration of free sites of type i at equilibrium, and let B_i denote the concentration of bound site i complex at equilibrium. Note we use the symbols F, S_i, and B_i for the numerical quantities in μM units of these materials, as well as for the names of the materials themselves. Then:

$$\begin{aligned} \kappa_i &= B_i/(FS_i) \quad \text{and} \quad B_i + S_i = S_{i0} \quad \text{for} \quad 1 \le i \le N, \quad \text{and} \\ F &= F_0 - B_1 - B_2 - \ldots - B_N. \end{aligned}$$

Note that sk_i is specified in liters/μMole units, so that $10^6\kappa_i$ is the corresponding equilibrium constant in liters/Mole units.

Often there is a fictitious $(N+1)$-st site, X, which binds F molecules, which is introduced to describe the *non-specific* binding of F molecules with weak affinity to many locations, other than the N specific sites of interest. Let κ_0 be the equilibrium constant for the fictitious non-specific binding reaction $F + X \rightleftharpoons Y$. Then we have:

$$\kappa_i = B_i/(FS_i) \quad \text{and} \quad B_i + S_i = S_{i0} \quad \text{for} \quad 1 \le i \le N,$$

$$\begin{aligned} \kappa_0 &= Y/(FX) \quad \text{and} \quad Y + X = X_0, \quad \text{and} \\ F &= F_0 - B_1 - B_2 - \ldots - B_N - Y, \end{aligned}$$

where Y is the concentration of non-specifically bound ligand, X_0 is the concentration of the fictitious non-specific binding site, and X is the concentration of the free fictitious site at equilibrium. Then $B_i = \kappa_i S_{i0} F/(1 + \kappa_i F)$ for $1 \le i \le N$, and $Y = \kappa_0 X_0 F/(1 + \kappa_0 F)$.

Now, to capture the notion of non-specific binding as a weak sticking of F molecules almost everywhere, let κ_0 tend to zero and let X_0 tend to infinity such that $\kappa_0 X_0 = c$, where c is a fixed constant. Then $Y \to cF$, and we have:

$$B_i = \kappa_i S_{i0} F/(1 + \kappa_i F) \quad \text{for} \quad 1 \le i \le N, \quad Y = cF, \quad \text{and} \quad F = F_0 - B_1 - \ldots - B_N - Y.$$

Note that the mathematical form $B_i = S_{i0} F/(1/\kappa_i + F)$ is difficult to handle when κ_i approaches 0, or when $1/\kappa_i + F$ approaches 0, so that the form given above is more convenient.

Usually values of F and/or $B_1 + B_2 + \ldots + B_N + Y$ are measured for different values of F_0 and we wish to use this data, which is generally of the form (F_0, F) or $(F_0, B_1 + \ldots + B_N + Y)$, to estimate κ_1, κ_2, ..., κ_N, c, and, if not already known, S_{10}, S_{20}, ..., S_{N0}. When only the total bound concentration $B_1 + B_2 + \ldots + B_N + Y$ is measured, rather than B_1, B_2, ..., B_N and Y separately, only a few sites can be distinguished by curve-fitting. In order to determine how many kinds of sites appear to be present, we must try each of the 1-site, 2-site, etc. models and choose among them on the basis of how well the data is fit. If the equilibrium constants obtained for two species of sites are close, then there are no grounds for considering them to be distinct species, based on this analysis alone.

Often people use the model $B(F) = \kappa_1 S_{10} F/(1 + \kappa_1 F) + \cdots + \kappa_N S_{N0} F/(1 + \kappa_N F) + cF$ and fit it to data points of the form (F, B) where F is the free ligand concentration at equilibrium and B is the total bound ligand concentration at equilibrium, with one such point for each experiment. *The difficulty with this approach is that we must compute F as $F_0 - B$ so that there is correlated error in* **both** *the dependent and independent values used to form the data points.* Our approach uses data points of the form (F_0, B), and since F_0 can be accurately determined, we avoid the aforementioned difficulty of error in the independent variable.

It is possible to model the simultaneous use of *several* distinct types of ligands having distinct binding-affinities interacting with multiple classes of sites. This can be done by a straightforward elaboration of the equations used above. Although we shall not provide an example here, this is an important situation to be kept in mind.

For example, the general two-site model with non-specific binding for a single ligand type can be defined in MLAB with the following dialog. Here and hereafter, the text shown following the MLAB prompt asterisk is an MLAB command statement entered by the user.

```
* FUNCTION B(F)  = k1*S10*F/(1+k1*F) + k2*S20*F/(1+k2*F)
* FUNCTION F(F0) = ROOT(Z,0,F0,F0-B(Z)-Z*(1+c))
* FUNCTION Y(F0) = c*F(F0)
```

Here $A(F_0) = B_1 + \ldots + B_N$ for $N = 2$.

These commands exemplify the MLAB `FUNCTION` statement, which is used to define a function or differential equation. Note that arguments of functions must be explicitly specified. Variables, such as `k1` and `k2`, which appear in the body of a function, but not in its argument list, are called parameters. Parameters must be assigned values before an associated function can be evaluated.

`ROOT` is an operator which is built-in in MLAB. `ROOT(Z,A,B,E)` is a value between `A` and `B` which, when taken as the value of the dummy variable, `Z`, makes the expression, `E`, which involves `Z`, equal to zero. Thus `ROOT(Z,A,B,E)` is a solution of $E(Z) = 0$. The model given above, involving a so-called *implicit* function, deserves careful study; it is easily extendible to more than two sites. The amount F of free-ligand at equilibrium as a function of F_0 satisfies $F_0 - F = B(F) + c \cdot F$.

4.2 An Example

Suppose we have measured F in μM units as a function of F_0 in μM units, as follows:

F_0	F
.58668	.036
1.1734	.096
2.3467	.385
2.9334	.61
4.1068	1.15
4.6934	1.46
5.8668	2.11
7.0402	2.73

and we have $S_{10} = S_{20} = 1.7121\,\mu$M. (If S_{10} and S_{20} were unknown, we could include them as fitting parameters.)

Then, we can introduce the appropriate constraints (which should always be used for this particular model), guess K_1, K_2, and c and then estimate them, as follows.

```
* constraints q ={k1>=0,k2>=0,c>=0,S10>=0,S20>=0}
```

The `CONSTRAINTS` statement permits the user to specify successive linear inequalities or equations involving the parameters (or potential parameters). Now we may specify values for the parameters, guessing when necessary.

```
* k1 = 10; k2 = 1; c = 0; S10 = 1.7121; S20 = S10;
```

Here the `ASSIGNMENT` statement is exemplified. In this case all the above assignments are assigning values to scalar variables, but the assignment statement is used to assign values to matrices as well. This can be seen in the next assignment statement which defines a matrix, D.

```
* D = Kread(8,2)
.58668 .036
1.1734 .096
2.3467 .385
2.9334 .61
4.1068 1.15
4.6934 1.46
5.8668 2.11
7.0402 2.73
```

The `KREAD` operator takes optional array size arguments; in this case an 8 row by 2 column matrix is specified, and reads in numbers from the keyboard to construct an appropriately-dimensioned matrix as the result. An analogous function is used to read numbers from a file. This matrix is then, in this case, assigned to `D`. Note the entry of the numbers which follow. Now we may examine `D` by "typing it out" using the `TYPE` statement.

```
* type D
D:  8 by 2 matrix
1: .58668 .36E1
2: 1.1734 .96E-1
3: 2.3467 .385
4: 2.9334 .61
5: 4.1068 1.15
6: 4.6934 1.46
7: 5.8668 2.11
8: 7.0402 2.73
```

We have established a model function, F, and entered data, D. We expect that $f(D[i,1]) \approx D[i,2]$ would hold, if only the parameters K_1, K_2, and c were set to their "correct" values. The following `FIT` statement requests MLAB to estimate K_1, K_2, and c by assigning them values which minimize the sum-of-squares objective function $S(\mathrm{K}_1, \mathrm{K}_2, c) = \sum_{i=1}^{8} \left(F(D[i,1]) - D[i,2]\right)^2$.

```
* maxiter = 30; TOLSOS = .001
* fit(k1,k2,c), F to d, constraints q
final parameter values
value           error           dependency      parameter
13.86352736     2.331373145     0.665073064     K1
0.5321372226    0.06602915638   0.9432600296    K2
0.5874015019    0.02229743988   0.9171690302    C
5 iterations
CONVERGED
best sum of squares = 9.78763e-04
root mean square error = 1.39912e-02
deviation fraction = 6.82654e-03
R squared = 9.99854e-01
no active constraints
```

The behavior of the fit statement depends upon the supplied constraints q, as well as upon the MLAB control variables: maxiter, the maximum number of iterations and tolsos, the requested convergence factor.

MLAB uses a carefully-tuned version of the Marquardt-Levenberg magnified-diagonal algorithm which is, in turn, a form of the Gauss-Newton procedure for minimizing a function which is in the form of a sum-of-squares. This process estimates the value of the parameter vector $b = (\mathrm{K}_1, \mathrm{K}_2, c)'$ by successive approximations $b^{(0)}$, $b^{(1)}$, ..., $b^{(n)}$, where $b^{(0)}$ is the vector of initial guesses for K_1, K_2, and c, and $b^{(j+1)} = b^{(j)} + \beta^{(j)}$, where

$$\begin{aligned}
\beta^{(j)} &= (X'V^{-1}X + \varepsilon G)^{-1}X'V^{-1}(y - (f(x_1; b^{(j)}), \ldots, f(x_8; b^{(j)}))'), \quad \text{with} \\
X_{st} &= \partial f(x_s; b^{(j)})/\partial b_t \quad \text{and} \\
G_{st} &= \text{if } s = t \text{ then } (X'V^{-1}X)_{st} \text{ else } 0 \quad \text{and} \\
x_s &= D[s, 1] \quad \text{for} \quad 1 \le s \le 8, \quad \text{and} \\
y &= (D[1, 2], \ldots, D[8, 2])',
\end{aligned}$$

where V is the estimated covariance matrix of the observations. In our example, $V = I$, the identity matrix. In general V is determined from weight-values supplied by the user.

An iteration consists of computing $b^{(j+1)}$ from $b^{(j)}$. Note that this requires the partial derivatives of the model function with respect to the parameters evaluated at $b^{(j)}$, since these values form the matrix X. In MLAB, these derivatives are automatically computed symbolically and evaluated to form X. The convenience thus obtained is considerable and the parameter estimation process is provided with more accurate derivative values. For example the derivative of F with respect to K_1 can be explicitly displayed in MLAB as follows.

```
* type F'k1
FUNCTION F'K1(F0) = EVAL(Z,ROOT(Z,0,F0,(F0-B(Z))-Z*(1+c)),
        -B'K1(Z)/(B'F(Z)-(1+c)))
```

Indeed derivatives are full-fledged members of the class of functions and can be used in graphics or curve-fitting in MLAB just as can any other user-defined function.

A sub-iteration consists of computing $b^{(j+1)}$ with a particular value of ε which specifies the amount of diagonal magnification. At each iteration, the value ε starts at 10^{-9} and is increased until the corresponding value of $b^{(j+1)}$ results in a smaller sum-of-squares value, whereupon this vector is taken to be the final $b^{(j+1)}$ iterate.

The parameter estimation process stops when the limit of the number of iterations is reached or, more usually, when the decrease in the sum-of-squares value between successive iterations is less than a specified fractional amount determined by the user-specified convergence factor in `TOLSOS`. For `TOLSOS` = .001, the sum-of-squares must change by less than .1 percent for the curve-fitting process to stop based on this criterion.

When the estimation process does stop, the parameters are reset to their computed estimates, and they and their estimated standard deviations are typed out. Associated values called dependency values, which lie between 0 and 1, are also typed out. It suffices here to remark that large dependency values above .99 usually indicate a non-unique solution; that is, other parameter estimates exist which would provide a nearly equally-small sum-of-squares.

MLAB also types-out the root-mean-square error which is the estimate of the standard deviation in each observation, given that they are identically distributed. This quantity should roughly equal the experimental error in the data. There are, of course, many caveats and restrictions which must hold to insure the validity of the supporting statistics provided.

The material typed out above shows that the vector of parameters $(\mathrm{K}_1, \mathrm{K}_2, c)$ has been estimated to be $(13.8633 \pm 2.3315, .532151 \pm .0660296, .587396 \pm .0222973)$, with reasonably small dependency values, and with an RMS error of about .014, which should be comparable with the experimental error in our data, D. The sum-of-squares was reduced from an initial value of 2.64314 to .000978763 at the final parameter values.

In order to visually see how our model with its parameters set to their best-estimated values corresponds to the data, we may draw a graph of the data points and the model function. Although we are drawing only the simplest and most direct kind of picture here, it should be noted that MLAB provides facilities for many types of point-symbols and types of lines, axes with arbitrarily-placed numeric labels in various formats, titles in the form of text strings in arbitrary sizes and various fonts with subscripts and superscripts, color, and a number of other special features. It is quite possible to prepare more or less elaborate publication-quality graphs with a modest amount of effort. Indeed this is one of MLAB's most-used facilities. The desired graph can be constructed as follows.

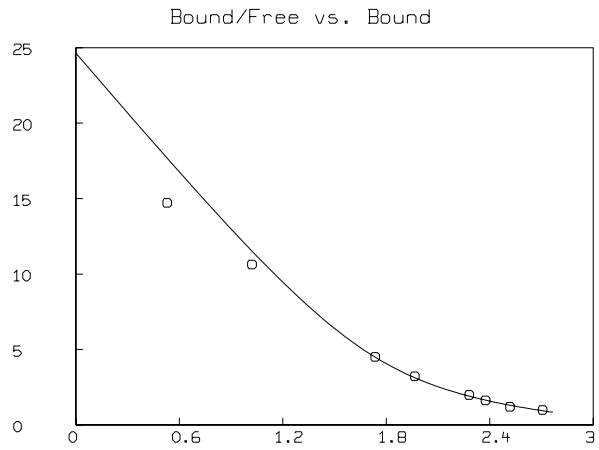
```
* draw D, linetype none,pointtype circle
* draw c2 = points(F,0:8:.2)
* VIEW
```

The first DRAW command above plots the data points, while the second draw command constructs a curve called C2, which is a graph consisting of solid straight-lines connecting the points which are the rows of the 2-column matrix which is the value of the expression POINTS(F,0:8:.2). This matrix has the values 0 through 8 in steps of .2 in its first column and corresponding values of the function F evaluated at 0 through 8 in steps of .2 in the second column. The POINTS operator is very useful for graphing functions. Both these curves are drawn in the default MLAB 2D-window called W (since no other window is specified) which has predefined labeled axes already present. The picture finally appears when its display is requested with a view statement and a plot can be obtained if desired using the PLOT statement.

Often a Scatchard graph of Bound/Free vs. Bound (*i.e.*, $B(F(F_0))/F(F_0)$ vs. $B(F(F_0))$) is desired. Although such a formulation should not be used for curve-fitting due to the non-normal error introduced by computing Bound/Free, it is quite straightforward to draw the data and model Scatchard plots as follows. Note the current picture in W is saved and restored.

```
* SAVE W IN GW
* DELETE W
* FUNCTION R(X,Y)=IF Y=0 THEN (k1*s10+k2*s20) ELSE X/Y
* MF = F ON 0:8:.2
* M = B ON MF
* M = M&'(R ON M&'MF)
* DRAW M;
* MF = D COL 2
* M = (D COL 1)-(1+C)*MF
```

```
* M = M&'(R ON M&'MF)
* DRAW M, LINETYPE NONE, POINTTYPE "o"
* TOP TITLE "Bound/Free vs. Bound"
* VIEW
```




```
* DELETE W,MF,M
* USE GW
```

The `&'` operator denotes column concatenation, while the `ON` operator, as in `F ON H`, applies the function `F` to each row of the matrix `H` treated as an argument list for `F` and returns the column vector of result values.

It is an enlightening exercise to fit the Scatchard model to the corresponding transformed data and to then draw a graph of the original data points and the function f using the parameter values obtained. An example of this comparison is given in the following section.

If we know we have non-cooperative two-site specific binding, with our given S_{10} and S_{20} values, together with non-specific binding with no degradation (degradation could be due to the presence of various enzymes for example), then this curve-fit of the correct model gives us the best available estimates of κ_1, κ_2, and c. But if the model is possibly not correct, we have a more difficult problem of deciding what the correct model is. This often cannot be resolved without further experimentation.

The non-specific binding constant, c, can be independently estimated by adding a large amount of unlabeled ligand, L, which is distinguishable from F, to a mixture of F and the site-bearing material. Then virtually all the bound F which results will be non-specifically bound and taking this value as Y yields $c \approx Y/(F_0 - Y)$. With a value of c known from one or more such experiments,

c can be fixed in subsequent curve-fitting, or alternatively the data can be modified by using $F - cF$ for F and fitting an appropriate model with $c = 0$.

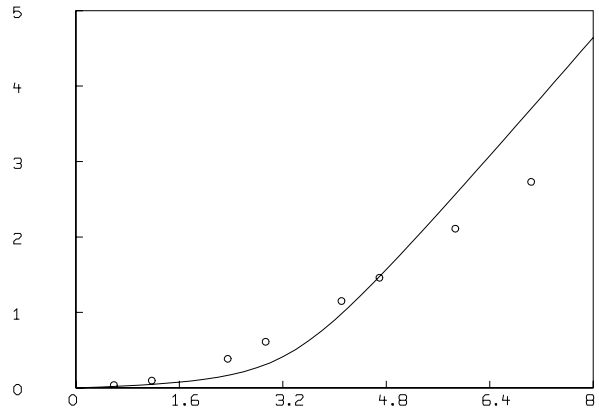
If we have no such independent estimate of c, and we are unsure if non-specific binding is occurring, we may test the possibility that there is no non-specific binding as follows.

```
* tolsos = .0002
* c = 0
* fit(k1,k2), F to d, constraints q
final parameter values
value           error           dependency      parameter
10.42883512     881.9654883     0.9989885554    K1
10.87461407     712.3997932     0.9989885554    K2
27 iterations
CONVERGED
best sum of squares = 1.31885e+00
root mean square error = 4.68838e-01
deviation fraction = 2.57765e-01
R squared = 8.02989e-01
no active constraints
```

`LSQRPT` is a sum of zero or more of the values 1, 2, 4, and 8. When `LSQRPT` contains 1, the best sum of squares obtained at each iteration is typed-out. When `LSQRPT` = 1, the fit command directs MLAB to type-out only the final results. All intermediate reports are suppressed. It is also possible to use `LSQRPT` = 0 which suppresses all output, but the parameters are reset as usual. In any event, a matrix called `COVP` (the estimated covariance matrix of the parameters) and a scalar called `SOSQ` (the final sum-of-squares value) are created, replacing any previous data objects with these names. The user may examine these values, or employ them in further calculations as desired.

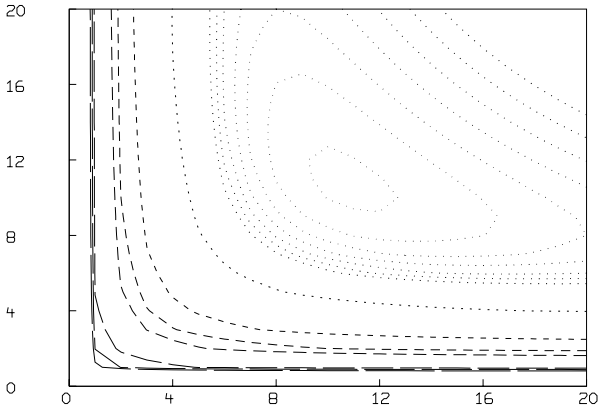
We may examine our fit by redefining the curve `C2`.

```
* draw c2 = points(F,0:8:.2); view
```



This is not an adequate fit, as we could have predicted by noting the large parameter standard deviations and the large RMS error. Although the dependency values are not extremely close to one, we may consider the possibility that our fit is bad because we fell into an unfortunate local minimum. A crude way to study this possibility is to draw a contour map of the sum-of-squares surface to see if several local minima are apparent. This is possible in this case, since we have fewer than three parameters, and MLAB can be used to view the sum-of-squares surface as follows. Drawing this contour map is time-consuming due to the large number of root calculations required.

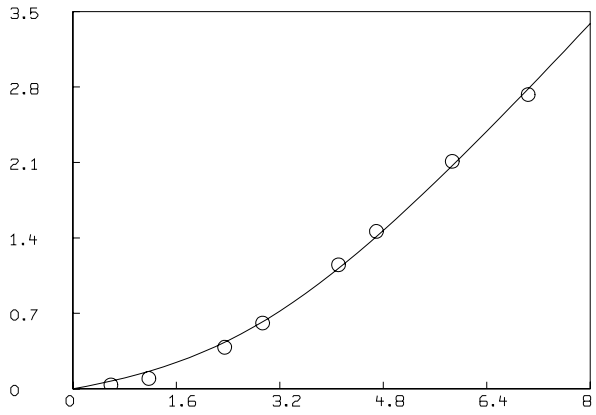
```
* FUNCTION FH(F0,K1,K2)=ROOT(Z,0,F0,F0-(K1*S10*Z/(1+K1*Z) \
      +K2*S20*Z/(1+K2*Z))-Z*(1+C))
* FUNCTION S(k1,k2)=SUM(I,1,8,(FH(D[I,1],k1,k2)-D[I,2])^2)
* M=contour(points(S,cross(0:20,0:20)),1.3:1.35:.005&1.4:1.8:.2&2:5)
* DELETE W
* DRAW M, LINETYPE VMARKER; VIEW
```



The CROSS operator expression above constructs a 441 by 2 matrix whose rows are all the pairs (i, j) with $0 \leq i, j \leq 20$. The CONTOUR operator expression constructs a specially-coded two-column matrix which is suitable for drawing with line-type VMARKER. Contour lines are formed for the surface height values in the list 1.3:1.35:.005 & 1.4:1.8:.2 & 2:5 where & denotes row concatenation. Evidently, our solution is unique.

We may try a one-site model with S_{10} equal to 3.4242 μM by using our two-site model where both sites are identical as follows. If S_{10} were not known, we could add S10 to the parameter list to be adjusted.

```
* S10 = S10+S20; k2 = 0;
* fit(k1,c), F to d, CONSTRAINTS Q
final parameter values
value           error           dependency      parameter
2.325176617     0.2567377717    0.5948957102    K1
0.4642651058    0.03442515131   0.5948957102    C
15 iterations
CONVERGED
best sum of squares = 1.55348e-02
root mean square error = 5.08835e-02
deviation fraction = 2.31134e-02
R squared = 9.97679e-01
no active constraints

*delete w
*draw D, line none, pointtype circle
*draw points(f,0:8:.2); VIEW
```



This fit is not as good as the fit obtained with the two-site model, and we can probably reject it. We should have an estimate of the variance of the data in order to test the goodness-of-fit. Even with curve-fitting evidence, it is best if there is independent evidence of multiple sites. Also, of course, a three or more site model, with or without non-specific binding may be the correct model.

Thus, in general, given the correct model, curve-fitting is a powerful device for estimating parameters; but curve-fitting is not always a very good way to distinguish the correct model from alternate impostor models. Usually we must resort to discriminating experiments.

MLAB can handle simultaneous curve-fitting, where several functions are to be fit to corresponding sets of data points and where, moreover, these several functions share some parameters in common. MLAB model functions can be functions of more than one formal argument, so that data points may lie in n-space for arbitrary n, but we do not provide an example here. However we shall provide an example of simultaneous curve-fitting.

Suppose we not only have the data given above for 2-site binding, but also are given the following observations of the concentrations Y of F-ligand which is non-specifically bound, for various amounts of ligand provided initially, denoted by F_0.

F_0	Y
1.1734	.3
2.5	.5
3	.3
3.98	.7
6.5	1.6

Then we may enter this data as the rows of a matrix N and estimate `k1`, `k2`, and `c` based on all the available data.

```
* N=KREAD(5,2)
1.1734 .3
2.5 .5
3 .3
3.98 .7
6.5 1.6
```

In general, we should use weights which specify the relative accuracy of each data point. For each data matrix, a vector of weight values may be given as a clause in the `FIT` statement. The weight-value, g_i, that corresponds to an observation (x_i, y_i) should be $1/\operatorname{var}(y_i)$. Of course, guesses or estimates must generally suffice. The value $1/y_i^2$ for y_i corresponds approximately to a constant percentage error in the observed value y_i. Weight functions are permitted in MLAB, and a more

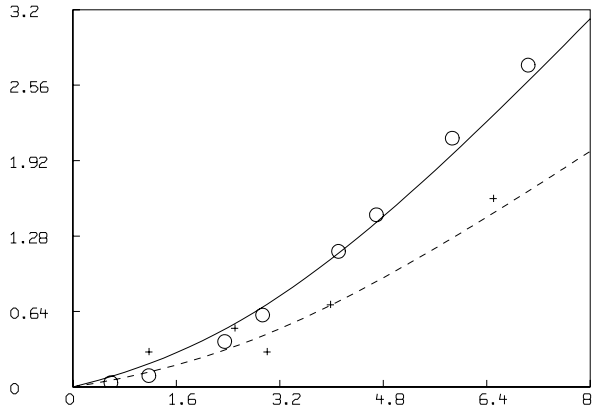
accurate device is to use the reciprocal of the square of the model function itself as the weighting function. This results in an iterative reweighting process.

The use of weights permits data points of high reliability to exert greater influence on the parameter estimates than data points of lesser reliability. When simultaneous fitting is being done, weights have the additional function of compensating for differing units or magnitudes of the observations from various data sets. Without weights, the curve-fitting process would tend to favor one model component, fitting it at the expense of others, if the deviations there were much larger than the others, even though this may be an artifact of the use of different units.

MLAB includes an operator, `EWT`, which computes a vector of weight-values for a given set of data points, `M`, by estimating the standard deviation at each point by the difference between the data curve and a smoothed form of it, which effectively assumes the error is due to white noise. These standard-deviation estimates are, in turn, smoothed, and then used to form reciprocal variance weight values. In our example, we shall use `EWT` to obtain the necessary weight vectors, although this is not totally appropriate for chemical binding data, which is often taken in a range where the measurement accuracy improves as the amount of bound ligand increases. Also, `EWT` is unreliable for small numbers of data points. Nevertheless, we shall proceed as follows.

```
* DW = EWT(D); DN = EWT(N)
* TYPE D&'DW,N&'DN
: a  8 by 3 matrix
1:  0.58668    0.036    340.092919
2:  1.1734     0.096    340.092919
3:  2.3467     0.385    271.140209
4:  2.9334     0.61     256.511957
5:  4.1068     1.15     193.017413
6:  4.6934     1.46     171.441311
7:  5.8668     2.11     93.14747
8:  7.0402     2.73     93.14747
 a  5 by 3 matrix
1:  1.1734     0.3      159.445918
2:  2.5        0.5      159.445918
3:  3          0.3      156.341715
4:  3.98       0.7      134.665471
5:  6.5        1.6      134.665471

* fit (k1,k2,c), F to d with weight dw, Y to n with weight dn, CONSTRAINTS q
Begin iteration 1 bestsosq=5.86711e+01
Begin iteration 2 bestsosq=2.76531e+01
Begin iteration 3 bestsosq=2.69881e+01
final parameter values
value           error           dependency      parameter
```

```
1.681292363     0.3984166729    0.6190759098    K1
1.905413519e-17 0.06987097833   0.6367292361    K2
0.6401358429    0.057666687     0.09280130233   C
3 iterations
CONVERGED
best sum of squares = 2.69879e+01
root mean square error = 1.64280e+00
deviation fraction = 6.88260e-02
lagrange multiplier[1] = -0
lagrange multiplier[2] = -73.23991748
lagrange multiplier[3] = -0
* delete w
* draw d line none pt circle
* draw points(F,0:8:.2)
* draw n line NONE pointtype "+"
* draw points(Y,0:8:.2) linetype dashed
* VIEW
```



MLAB also allows weights to be the reciprocals or other functions of the actual squared deviations themselves. This device is called curve-fitting with iterative reweighting and is invoked in MLAB by specifying an appropriate weight function. Moreover MLAB employs an iterative reweighting technique to allow the general form of a sum of *pth* powers to be minimized, rather than just a sum-of-squares. This so-called L_p-norm fitting is sometimes useful for $p = 1$ or 1.5 when the data is not normally-distributed, and tends to contain some outliers. The value p is specified as the value of the MLAB control variable `fitnorm`.

4.3 Practical Direct and Competitive Ligand Binding Analysis

Given a radioactively-labeled ("hot") ligand material F, we wish to mix and incubate varying amounts of hot ligand each with the same fixed amount of site material, and obtain a table of the amounts of bound ligand corresponding to the total amounts of ligand added. This requires that we be able to separate the bound ligand from the free ligand and then measure the amount of bound ligand by the numbers of radioactive-decay events per minute for each distinct experiment. The separation process must be quick compared to the incubation time needed to approach equilibrium.

There is an alternate device which sometimes has an advantage in simplicity over this direct method. Let us add a single fixed amount of hot ligand to the site material and then add varying amounts of unlabeled ("cold") ligand. The cold ligand is often an unlabeled form of F, but the cold ligand may be a reactant distinct from F. The cold ligand competes with the hot ligand in binding to the sites, and the effect is that, as we add more cold ligand, the observed concentration of bound hot ligand decreases. We may sometimes be able to add an incremental amount of cold ligand to our mixture, incubate and then separate and measure the amount of bound hot ligand, and then remix and add the next incremental amount of cold ligand; when this is possible, the variability due to differing concentrations of sites and hot ligand is reduced. Since replicate experiments suffer from using slightly-differing amounts of site material and of hot ligand, it is often better to expend effort getting more points for a single sample of site material than to run replicate experiments (except as a gross check on the experimental process.) In any event, the results of replicate experiments need not be averaged; the MLAB curve-fitting process will, in effect, give each point equal weight as is appropriate. Note this approach uses a smaller amount of hot-ligand than direct-binding studies generally require.

We have a further complication to deal with. Both the hot and cold ligand materials usually bind weakly and non-specifically throughout our mixture, in addition to specifically binding the site material. The amount Y_H of such non-specifically bound hot ligand is generally taken to be proportional to the total amount of hot ligand present less the amount bound, *i.e.* $Y_H = cF_H$ for some constant c where F_H denotes the concentration of free hot ligand at equilibrium. Let A_H denote the concentration of bound hot ligand, both specifically and non-specifically, and let B_H denote the concentration of specifically-bound hot ligand. The concentration of hot ligand in all forms is $L_H = A_H + F_H$. Then $A_H = B_H + Y_H$. If we add a very large amount of cold ligand to our mixture, almost all the specific binding sites will be occupied by cold ligand. The amount of observed bound hot ligand A_H may then taken to be the amount of non-specifically bound hot ligand Y_H, whence $c \approx Y_H/F_H = A_H/(L_H - A_H)$. We can thus use our other observed A_H values and compute corresponding B_H values by subtracting $c(L_H - A_H)$ from the A_H values, where c has been estimated as indicated above. Alternatively, it is possible, and generally preferable, to curve-fit for the unknown non-specific binding constant c, as is done below.

In any event, for each observation point, we know the constant Molar concentration L_H of hot ligand present, the concentration L_C of cold ligand present and the observed concentration $A_H = B_H + Y_H$ of bound hot ligand (with due regard for the volume of the mixture). Computing the

concentration A_H requires a computation based on the following parameters.

1. the *molecular weights* of hot and cold ligand (often these are taken to be the same.)
2. the *half-life* of the radioactive-label material and the *age* of the hot ligand being used, measured in days.
3. the *counting efficiency* of the radioactive decay detector device, measured as the ratio of observed counts per minute to the actual number of decay-events per minute.
4. the *counting time period* (minutes) during which decay events are monitored.
5. the *background radioactivity* (in counts per time-period) in the site material and the cold ligand taken together.
6. the *specific activity* of the hot ligand measured as the expected number of decays per minute per mole of hot ligand of age 0.
7. the *total counts* over the counting time period for the total amount of hot ligand material.
8. the *bindable fraction* of the hot ligand defined as the fraction of the radioactive label material attached to non-inert ligand which is actually capable of binding.
9. the *volume* (milliliters) of the mixture.
10. the *mass amounts* (nanograms) of the cold ligand and the hot ligand in our mixture.
11. the *observed counts* during the counting time-period for the bound ligand component.

Let WH be the molecular weight of hot ligand.
Let WC be the molecular weight of cold ligand.
Let HL be the half-life in days of the label material.
Let AL be the age in days of the label material.
Let CE be the counting efficiency in (cpm/dpm) units.
Let CT be the counting time period in minutes.
Let BR be the background radioactivity in counts per time period.
Let SA be the specific activity of the hot ligand in dpm/mole.
Let TC be the total counts per time-period of the hot ligand material.
Let BD be the bindable fraction of hot ligand material.
Let V be the volume in milliliters.
Let MH be the mass in nanograms of the hot ligand material.
Let MC be the mass in nanograms of the cold ligand material.
Let AO be the observed counts per time-period for the bound ligand component.

Then we have $L_H = MH/[WH \cdot V \cdot 10^6]$ Molar units (M) of hot ligand. As a check, $L_H = (1/SA)[TC/(CT \cdot CE)] \cdot 2^{AL/HL} \cdot (10^3/V)$. This equation can alternately be used to determine any of the other parameters involved, given L_H in Molar units. In particular, if the observed value of TC is not close to the predicted value from this equation, there is some difficulty in the experimental parameters. The corrected value of L_H we use in subsequent calculations is then obtained by multiplying L_H by the bindable fraction factor BD. The concentration in Molar units of the bound hot ligand is computed as $A_H = (1/SA)[(AO - BR)/(CT \cdot CE)] \cdot 2^{AL/HL} \cdot (10^3/V)$.

The concentration of cold ligand is $L_C = MC/[WC \cdot V \cdot 10^6]$ M. (If a fraction of cold ligand is inert, the corresponding correction should be made to L_C.) A series of competition experiments consists of varying MC and observing the differing values of AO (and potentially BR) that result. The final outcome is a sequence of (L_C,L_H,A_H) triples, where L_C and A_H vary, and L_H is generally fixed. For a series of direct-binding experiments, MC is 0, and we vary MH directly. The result is a sequence of (L_C,L_H,A_H) triples where $L_C = 0$ and L_H and A_H vary. In either case, it is important to include observations at very high levels of total hot and cold ligand to insure that we have made at least one observation near the saturation limit where almost all of the sites are bound with ligand. The danger and consequences of failing to do this are covered briefly by Michael Johnson and Susan Frasier in *"Analysis of Hormone Binding Data", pp. 45-61 of Methods in Diabetes Research Vol. 1 (eds. J. larner and S. Pohl), Wiley, NY 1984.*

The error in A_H can usually be mainly attributed to the error in AO, which is, in turn, due to the random variation seen in counting radioactive decay events over the counting time-period CT. In order to minimize this error, CT should be as large as is practicable. Johnson and Fraiser (cited above) suggest that counting times be adjusted for each observation so that the number of counts seen, AO, is about the same value, say 3000, for each observation. This practice has the benefit that the standard-error of each measured AO-value is nearly the same, so weights need not be used in fitting. The drawback to this practice is that the counting time can become excessively large when only a small amount of hot ligand is bound. Note the observed-counts value AO is a sample from a Poisson-distributed random variable which we treat as a normally-distributed random variable for the purpose of curve-fitting.

Let us consider a single-site material S. Suppose our mixture contains a concentration of S_0 M of this site material together with L_C M of cold ligand and L_H M of hot ligand. Our goal is to estimate the equilibrium constant K_C of the reaction $L_C + S \rightleftharpoons B_C$. Let B_C denote the concentration of bound cold ligand at equilibrium and let F_C denote the concentration of free (unbound) cold ligand at equilibrium, just as B_H and F_H denote these quantities for hot ligand. Suppose the reactions $L_H + S \rightleftharpoons B_H$ and $L_C + S \rightleftharpoons B_C$ have the not-necessarily-equal equilibrium constants K_H and K_C, so that $K_H = B_H/(F_H S_F)$ and $K_C = B_C/(F_C S_F)$, where S_F denotes the concentration of free sites at equilibrium. Also, $B_H + F_H + Y_H = L_H$, $B_C + F_C + Y_C = L_C$, and $S_F + B_H + B_C = S_0$. All concentrations are in Molar units (M).

Our goal is to estimate K_C, given the varying observed concentrations of bound hot ligand associated with the sequence of cold ligand concentrations used. This cannot be done without knowing K_H, or alternately, the ratio $p := K_H/K_C$. Thus $K_C = K_H/p$, and either K_H or p, but not both,

is to be determined. Often we may assume $p = 1$.

A pre-computer-style Scatchard-plot analysis method is often employed to estimate the equilibrium constant K_C. When $L_C = 0$, suppose that $K_C = K_H/p$, where p is known. Also we have $K_H = B_H/(F_H S_F)$ and $B_H + S_F = S_0$. Then $B_H = S_0 - \frac{1}{K_H} \cdot \frac{B_H}{F_H}$, or $\frac{B_H}{F_H} = -K_H(B_H - S_0)$. This last relationship is a linear relation between B_H/F_H and B_H. If we graph B_H/F_H vs. B_H for varying amounts of L_H, we obtain a straight-line graph with slope $-K_H$ and x-axis intercept S_0 and y-axis intercept $K_H S_0$. Note that B_H/F_H is a dimensionless quantity, so we could compute B_H and F_H separately in any desired units (such as cpm), and obtain the same ratio when B_H/F_H is formed. This is the so-called Scatchard plot commonly used when a more advanced analysis is not available; the unknown parameters K_H and S_0 are estimated directly from looking at a straight-line fit of the experimental points (B_H,B_H/F_H). Then $K_C = K_H/p$. As L_H increases, both B_H and F_H increase, but F_H increases faster; this is reflected in the fact that the Scatchard plot line has a negative slope. If the plotted points do not approximately lie on a negatively-sloped straight line, the one-site binding model that has been assumed is probably incorrect. (What do you think about the example presented below?)

For a sequence of competitive binding experiments, the crucial point is that *when we have* 100α *% hot ligand and* $100(1-\alpha)$ *% cold ligand, then both forms of ligand participate in every chemical state in this proportion (modified by the equilibrium proportionality constant p.)*

When $p = 1$, so that $K_C = K_H$, we then have $B_H/F_H = -K_H[B_H + B_C - S_0]$. And $B_H = \alpha(B_H + B_C)$ and $B_C = (1-\alpha)(B_H + B_C)$, where $\alpha := L_H/(L_H + L_C)$. Thus $B_C = ((1-\alpha)/\alpha)B_H$, so $B_H/F_H = -K_H[B_H/\alpha - S_0]$, analogous to the non-competitive binding case!

In general, however, it is preferable to use the data (F_0,B), where $F_0 = L_C + L_H$ and $B = A_H/\alpha = (B_H + Y_H)/\alpha = (B_H + Y_H)F_0/L_H$, and fit the model given above for multiple-site binding with a single ligand where the independent variable is the total concentration of (hot and cold) ligand supplied, and the dependent variable is the concentration of (hot and cold) ligand bound (specifically and non-specifically). When $p \neq 1$, using this model is required.

Thus we may construct the data points (F_0,B) where $F_0 = L_C + L_H$ and $B = A_H F_0/L_H$ for each experimental data point (L_C,L_H,A_H), and then fit the function A to this data in order to estimate the parameters c, S_{10}, K_1, (and S_{20}, K_2, if a two-site model is desired). When $p \neq 1$, we introduce the constraints $K_2 = K_1/p$ and $S_{10} = S_{20}$ and use the two-site model for the one-site situation with $K_C \neq K_H$.

A general do-file based on the MLAB statements given above can be constructed which queries the user for the experimental parameters: HL, AL, CE, CT, BR, SA, TC, BD, V, MH and for the sequence of value-pairs (MC, BO), and then fits the desired model, reports the estimated values of c, S_{10}, and $K_1(= K_C)$, and graphs the corresponding saturation plot and Scatchard plot for the data. A simplified example of such a do-file (called *liganal.do* is shown below, together with an example of its use.

```
"liganal.do = competitive single site binding analysis"
echodo=0 /* echodo=0 inhibits screen and log-file output */
reset
namesw=0

type "Specify the COLD ligand molecular weight"
wc=kread("wc=")

type "Specify the HOT ligand molecular weight"
wh=kread("wh=")

type "Specify the half-life in days of the HOT ligand label"
hl=kread("hl=")

type "Specify the age in days of the HOT ligand label"
al=kread("al=")

type "Specify the counting-efficiency (cpm/dpm) of the label"
ce=kread("ce=")

type "Specify the counting time-period in minutes"
ct=kread("ct=")

type "Specify the background radioactivity in cpp(counts per period)"
br=kread("br=")

type "Specify the specific activity of the HOT ligand(in dpm/mole)"
sa=kread("sa=")

type "Specify the total counts per period of HOT ligand(in cpp)"
tc=kread("tc=")

type "Specify the HOT ligand bindability fraction"
bd=kread("bd=")

type "Specify the Volume (in milliliters)"
v=kread("v=")

type "Specify the mass in ng of the HOT ligand
 (or enter -1 to have the mass computed for you.)"
mh=kread("mh=")
```

```
/*Compute the mass in ng of the HOT ligand! */
mhx=(v*wh*1e6)*(1e3/sa)*(tc/(ct*ce))*(2^(al/hl))/v
type "specified mh value (in ng):", mh
mh=mhx
type "computed mh (hot mass) value (in ng) based on tc:",mh

type "Specify the number of experimental observation points"
nb=kread("nb=")
type "type-in the sequence of "+strval(nb)+" data points"
type  "as: ([COLD ligand mass in ng], [observed bound cpp]) pairs"
type  "on successive lines"
namesw=1
d=kread(nb,2); d=sort(d)

/* 2 classes-of-sites binding model.
   We use k2=0 and s20=0 for the 1-class-of-sites specialization*/
FCT A(F0)=G(F(F0))
FCT G(F)=B(F)+C*F
FCT B0(F0)=B(F(F0));
FCT B(F)=K1*S10*F/(1+K1*F)+K2*S20*F/(1+K2*F);
FCT F(F0)=ROOT(Z,0,F0,F0-B(Z)-Z*(1+C));
FCT Y(F0)=C*F(F0);
CONSTRAINTS CQ={K1>=0,K2>=0,C>=0,S10>=0,S20>=0};

lh=mh/(wh*v*1e6)
lhck=(tc/(ct*ce))*(1e3/sa)*(2^(al/hl))/v
namesw=0
type "Molar conc. of HOT ligand based on supplied molecular wt.",lh
type "Molar conc. of HOT ligand based on total cpp supplied",lhck
namesw=1

lh=lh*bd /* compensate for inert hot ligand */

fct m0(x)=if x>0 then x else 0
fct cconc(mc)=mc/(wc*v*1e6)
fct bconc(a0)=((a0-br)/(ct*ce))*(1e3/sa)*(2^(al/hl))/v

dm col 1 = cconc on (d col 1)  /*total cold (M)*/
dm col 2= m0 on bconc on (d col 2)   /*bound + non-spec. bound (M)*/

/*type-out the raw and converted input data*/
type "D=input data [Cold Ligand (ng) | Hot counts]", d
```

```
type "DM=[Cold Ligand (M) | Hot Spec.+Non-Spec.bound(M)]",dm

/* compute da col 1= LH+LC, da col 2= AH/alpha */
da=lh+(dm col 1)            /*total (hot+cold) */
alpha=lh/'da                /* vector of scale-factors*/
da col 2=(dm col 2)/'alpha  /* scaled bound+non-spec. bound */

/* compute guesses for K1 and S10 from the Scatchard fit.
   estimate C from the largest cold ligand data point. */

i=maxrow(dm col 1);
c=m0(dm[i,2]/(lh-dm[i,2]))

fre=lh-(dm col 2)           /*free hot (FH) in Molar units */
yns=c*fre                   /*non-spec. bound hot */
sd =m0 on ((Dm COL 2)-yns); /* spec. bound hot*/
sd2 = sd/'fre               /*bound/free */
sd col 2=sd2
sd col 1 = (sd col 1)/'alpha /* spec. bound (all =H+C) */
sd=compress(sd);

type "directly-estimated non-spec. binding coef.", c

ltot=(da col 1)
ftot=((da col 1)-(da col 2))
ytot=yns/'alpha
btot = sd col 1

mmt=ltot&'btot&'ytot&'ftot
type "MMT=[L Total | B Total | Y Total | F Total]", mmt
mmh=mmt/'(alpha^^'4)
type "MMH=[L Hot | B Hot | Y Hot | F Hot]", mmh
mmcOA=mmt-mmh
type "MMC=[L Cold | B Cold | Y Cold | F Cold]", mmt

/*Remember BH/FH = -K1*BH/alpha +K1*S10 */

fct lin0(b)=-k1*b+k1s10 /* simple linear form */
fct lin(b) = -k1*(b-s10)
k1=1; k1s10=lh

lsqrpt=0
```

```
fit(k1,k1s10), lin0 to sd
s10=k1s10/k1
fit(k1,s10),lin to sd
type "Scatchard model linear-regession fit estimates",k1,s10
ok1=k1; os10=s10

draw sd lt none pt circle color orange
draw points(lin,0:s10!4) lt dashed
top title "Scatchard plot"
title "[dashed=regression-fit line]" at (.5,.75) fract size .14 inches
title "[hashed=euclidean-fit line]" at (.5,.7) fract size .14 inches

/* compute k1 and s10 for the best euclidean-fitting line */
qb=mean(sd)
q=sd-(qb')^^nrows(sd)
m=q'*q
z=eigen(m) row (1,3,5)
rno=if z[1,1]>z[1,2] then 2 else 3
v1=(z row rno)'
k1=-v1[2]/v1[1]
s10=(qb[2]-v1[2]*qb[1]/v1[1])/k1
type "Scatchard model euclidean-fit estimates;", k1,s10
draw points(lin,0:s10!50) pt nbar ptsize .01
unview;
ws=w; blank ws;

k2=0; s20=0;
/*define our weighting function (fixed-percentage error assumed) */
fct wvf(x)=1/x^p
p=2;
wv1=.00000001*(wvf on (da col 2));

lsqrpt=8
maxiter=90
FIT(K1,s10,C),A TO da with wt (wv1), CONSTRAINTS CQ;

/* Draw the (full) relative saturation curve log-plot form ------------*/
FL = (da[1,1]*.9):((da[nrows(da),1])*1.1)!100;
am= POINTS(a,FL);
lda= (log on da col 1)&'((da col 2)/'(da col 1))
draw lda lt none pt circle ptsize .01
lam =(log on am col 1)&' ((am col 2)/'(am col 1))
```

```
draw lam

nk1=k1; ns10=s10; k1=ok1; s10=os10
am = points(a,fl)
k1=nk1; s10=ns10
lam =(log on am col 1)&' ((am col 2)/'(am col 1))
draw lam lt dashed
top title "Log-Percent Saturation Plot"
bottom title "log(TOTAL LIGAND)"
left title "(spec. bound)/(total)"
title "[dashed=regression-fit line]" at (.5,.8) fract size .15 inches
title "[solid=MLAB model-fit line]" at (.5,.75) fract size .15 inches
VIEW;

/*Now draw the Scatchard form of the data +fit-curves -------*/
w = ws; unblank w
draw points(lin,0:s10!4) color yellow
TOP TITLE "Scatchard Plot: Bound/Free vs. Bound" color green;
title "[solid=MLAB model-fit line]" at (.5,.8) fract size .14 inches
left title "(bound)/(free)"
bottom title "bound ligand"
lowy=minv(sd col 2); hiy=maxv(sd col 2)
window 0 to 2*s10, lowy to hiy adjust wnice
VIEW;
```

The MLAB log-file result of running the above do-file on some actual experimental data is shown below.

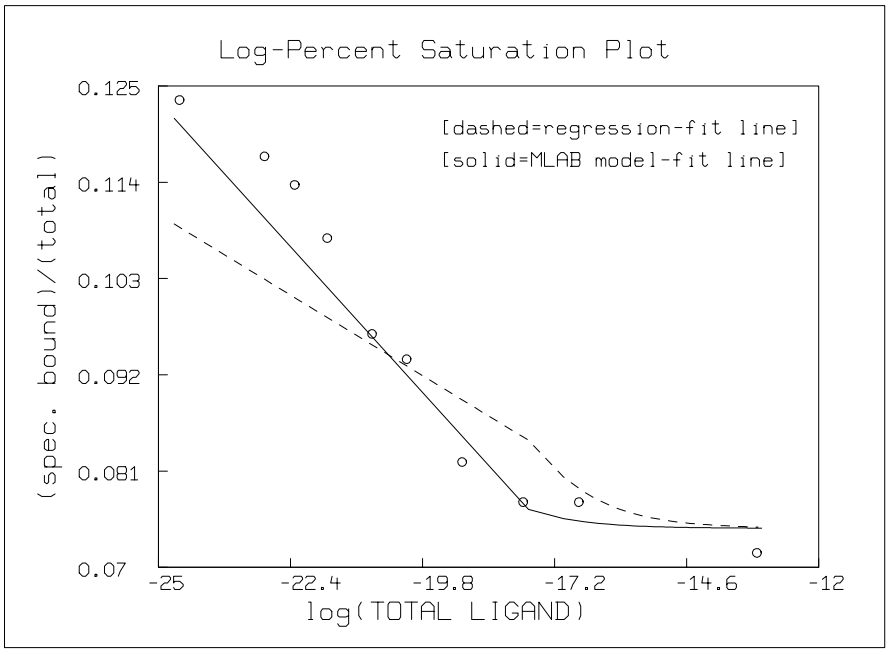
```
   MLAB Mathematical Modeling System, Revision: April 24, 1996
   Copyright: Civilized Software, Inc. (301)652-4714, email: csi@civilized.com

Sat May 18 15:29:38 1996
'* ' is the command prompt

* do liganal

Specify the COLD ligand molecular weight
wc=  1100
Specify the HOT ligand molecular weight
wh=  1130
Specify the half-life in days of the HOT ligand label
hl=  2
```

```
Specify the age in days of the HOT ligand label
al=  0
Specify the counting-efficiency (cpm/dpm) of the label
ce=  .5
Specify the counting time-period in minutes
ct=  5
Specify the background radioactivity in cpp(counts per period)
br=  0
Specify the specific activity of the HOT ligand(in dpm/mole)
sa=  2.5086e18
Specify the total counts per period of HOT ligand(in cpp)
tc=  65789
Specify the HOT ligand bindability fraction
bd=  1
Specify the Volume (in milliliters)
v=  .5
Specify the mass in ng of the HOT ligand
(or enter -1 to have the mass computed for you.)
mh=  -1
     specified mh value (in ng):
-1
     computed mh (hot mass) value (in ng) based on tc:
1.18538739E-2

Specify the number of experimental observation points
nb=  10
     type-in the sequence of 10 data points
     as: ([COLD ligand mass in ng], [observed bound cpp]) pairs
     on successive lines
  Type in numbers for KREAD.  End lines with <Enter>
  0 8118
  .05 7694
  .1 7480
  .2 7080
  .5 6360
  1 6170
  3 5397
  10 5096
  30 5096
  1000 4714

     Molar conc. of HOT ligand based on supplied molecular wt.
```

```
2.09803077E-11
     Molar conc. of HOT ligand based on total cpp supplied
2.09803077E-11

     D=input data [Cold Ligand (ng) | Hot counts]

   D: a  10 by 2 matrix

  1: 0      8118
  2: 0.05   7694
  3: 0.1    7480
  4: 0.2    7080
  5: 0.5    6360
  6: 1      6170
  7: 3      5397
  8: 10     5096
  9: 30     5096
 10: 1000   4714

     DM=[Cold Ligand (M) | Hot Spec.+Non-Spec.bound(M)]

   DM: a  10 by 2 matrix

  1: 0                2.58885434E-12
  2: 9.09090909E-11   2.45363948E-12
  3: 1.81818182E-10   2.38539424E-12
  4: 3.63636364E-10   2.25783305E-12
  5: 9.09090909E-10   2.02822291E-12
  6: 1.81818182E-9    1.96763135E-12
  7: 5.45454545E-9    1.72111935E-12
  8: 1.81818182E-8    1.62512955E-12
  9: 5.45454545E-8    1.62512955E-12
 10: 1.81818182E-6    1.50330862E-12

     directly-estimated non-spec. binding coef.
 C = 7.71837904E-2
     MMT=[L Total | B Total | Y Total | F Total]

   MMT: a  10 by 4 matrix

  1: 2.09803077E-11   1.16933226E-12   1.41952208E-12   1.83914534E-11
  2: 1.11889399E-10   5.45935993E-12   7.62606519E-12   9.88039735E-11
```

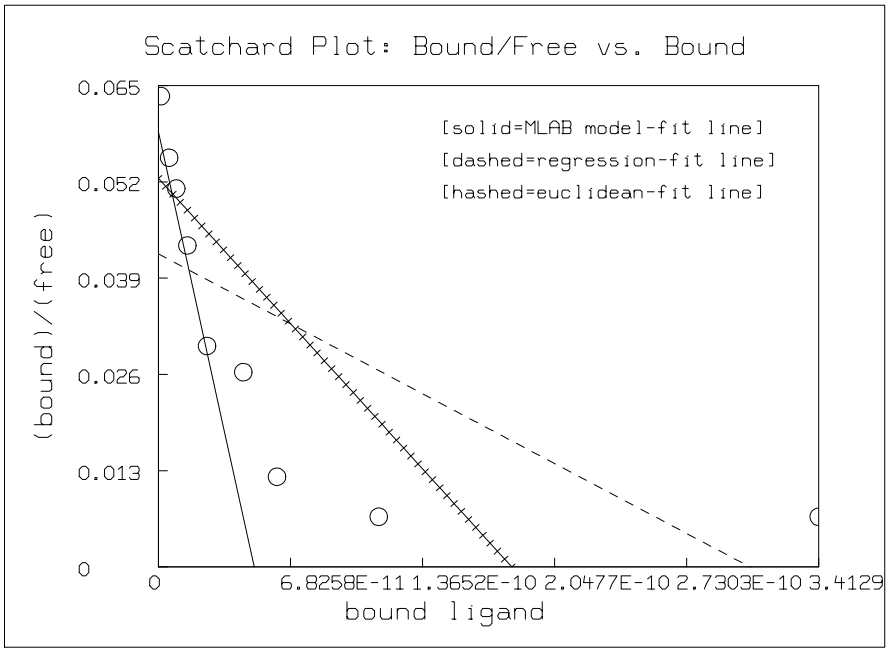
```
 3: 2.0279849E-10    9.18445554E-12   1.38730875E-11   1.79740946E-10
 4: 3.84616671E-10   1.48997633E-11   2.64914424E-11   3.43225466E-10
 5: 9.30071217E-10   2.50658571E-11   6.48466349E-11   8.40158725E-10
 6: 1.83916213E-9    4.38447819E-11   1.28640441E-10   1.6666769E-9
 7: 5.47552576E-9    6.12326499E-11   3.87952055E-10   5.02634106E-9
 8: 1.82027985E-8    1.13851314E-10   1.29613305E-9    1.67928141E-8
 9: 5.45664349E-8    3.41291496E-10   3.88541133E-9    5.0339732E-8
10: 1.8182028E-6     1.16933226E-12   1.30280259E-7    1.68792254E-6

   MMH=[L Hot | B Hot | Y Hot | F Hot]

  MMH: a  10 by 4 matrix

 1: 2.09803077E-11   1.16933226E-12   1.41952208E-12   1.83914534E-11
 2: 5.96713722E-10   2.91151354E-11   4.06703209E-11   5.26928266E-10
 3: 1.9602776E-9     8.87781883E-11   1.34099139E-10   1.73740028E-9
 4: 7.05089676E-9    2.73146488E-10   4.85648281E-10   6.29210199E-9
 5: 4.12306854E-8    1.11118638E-9    2.87469514E-9    3.72448039E-8
 6: 1.61223437E-7    3.84349283E-9    1.12767949E-8    1.46103149E-7
 7: 1.42902491E-6    1.59807452E-8    1.012493E-7      1.31179486E-6
 8: 1.57929939E-5    9.87789387E-8    1.12454254E-6    1.45696724E-5
 9: 1.41918596E-4    8.87644758E-7    1.01053353E-5    1.30925616E-4
10: 0.15756973       1.01337083E-7    1.12903936E-2    .146279336

   MMC=[L Cold | B Cold | Y Cold | F Cold]

  MMT: a  10 by 4 matrix

 1: 2.09803077E-11   1.16933226E-12   1.41952208E-12   1.83914534E-11
 2: 1.11889399E-10   5.45935993E-12   7.62606519E-12   9.88039735E-11
 3: 2.0279849E-10    9.18445554E-12   1.38730875E-11   1.79740946E-10
 4: 3.84616671E-10   1.48997633E-11   2.64914424E-11   3.43225466E-10
 5: 9.30071217E-10   2.50658571E-11   6.48466349E-11   8.40158725E-10
 6: 1.83916213E-9    4.38447819E-11   1.28640441E-10   1.6666769E-9
 7: 5.47552576E-9    6.12326499E-11   3.87952055E-10   5.02634106E-9
 8: 1.82027985E-8    1.13851314E-10   1.29613305E-9    1.67928141E-8
 9: 5.45664349E-8    3.41291496E-10   3.88541133E-9    5.0339732E-8
10: 1.8182028E-6     1.16933226E-12   1.30280259E-7    1.68792254E-6

   Scatchard model linear-regession fit estimates
 K1 = 138222119
 S10 = 3.0576542E-10
```

```
     Scatchard model euclidean-fit estimates;
  K1 = 287268836
  S10 = 1.82633621E-10

final parameter values
      value               error                dependency    parameter
     1187347821         228157865.8          0.9547016996   K1
 4.947046514e-11      8.636211996e-12          0.9633732247   S10
   0.0804516126      0.001324755233          0.5724726409   C
19 iterations
CONVERGED
best weighted sum of squares = 4.378871e-11
weighted root mean square error = 2.501106e-06
weighted deviation fraction = 1.716122e-02
R squared = 9.982448e-01
no active constraints
```



Note that the log-percent saturation plot using the estimated values for `K1` and `S10` obtained by linear-regression with the bound/free vs. bound Scatchard-plot form of the data is decidedly inferior to the graph using the direct saturation model-based estimates. Of course, as the following picture shows, this situation is not so clear in the Scatchard plot itself. But the log-percent

saturation plot is the more faithful form of the data with regard to minimal error in the x-values, and thus the associated estimated parameters are likely to be the most accurate.



You can see that MLAB is an extremely flexible and general tool for curve-fitting. Moreover, it has a broad range of other useful functions, only a few of which have been alluded to here.

5 Ultracentrifuge Data Analysis

One common application of curve-fitting occurs in the analysis of ultracentrifuge data. One of the most common uses of an ultracentrifuge is for molecular weight determination. Suppose we load the cell (a tube-like container) of an ultracentrifuge with a solution of a substance with an unknown molecular weight, and then spin the cell until the molecules in solution distribute themselves in the cell according to the forces that act upon them. This equilibrium arrangement will have the solute distributed in the cell in an exponential fashion with relatively more of the solute at the bottom (which is the outer end) of the cell. Therefore the concentration from the center of the solute varies at different positions along the cell (i.e., at different radial positions from the center of the ultracentrifuge.)

Let us denote a given radial position by r, measured in centimeters from the center of rotation, where the cell ranges from the top position at r_m to the bottom position at the outer end of the cell at r_b. Then the concentration of the solute, c, measured in grams per liter, at the radial position r is a function of r given as:

$$c(r) = c_h e^{AN(r^2 - r_h^2)},$$

where

c_h is the concentration in grams per liter of the solute at a given reference radial position, r_h. c_h can be computed from N and c_0, the total amount of solute in the cell measured in grams per liter. In practice c_h can be determined by curve-fitting.

r_h is the given radial reference position (it is not necessary that r_h be r_b or r_m).

N is the apparent molecular weight of the solute.

A is a certain constant, defined as: $(1 - \bar{v}\rho)w^2/(2RT)$, where $\bar{v}$ is the partial specific volume of the solute ($1/\bar{v}$ is the density of the anhydrous solute, so $\bar{v}$ is measured in cm^3/gram), ρ is the solution density in grams/cm^3, w is the angular velocity of the rotor in radians per second, R is the gas constant ($\approx 8.314 \times 10^7$ ergs/degree/mole), and T is the absolute temperature.

Due to the non-ideal behavior of most materials, the observable or apparent molecular weight N, at a given radial position r, is approximately the following non-linear function of the actual molecular weight, M : $N \approx M/(1 + BMc(r))$, where the parameter B is the so-called virial coefficient, with $B \geq 0$.

Thus N is a function of r, and we have:

$$\begin{aligned} c(r) &= c_h \exp(AM(r^2 - r_h^2)/(1 + BMc(r))), \quad \text{or} \\ c(r) &= root(x, 0, c_{\max}, (c_h \exp(AM(r^2 - r_h^2)/(1 + MBx)) - x). \end{aligned}$$

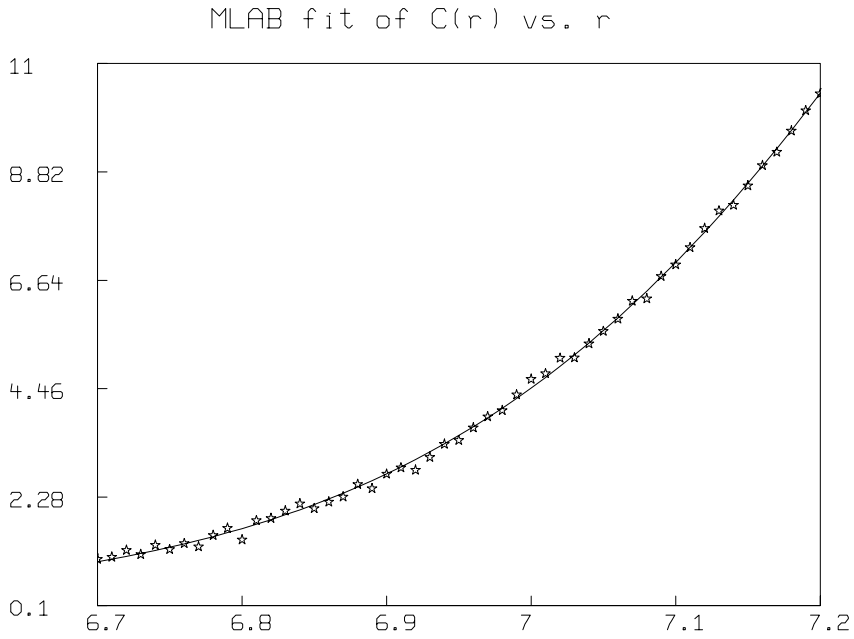
The notation $root(x, a, b, f(x))$ denotes a value, x_0, in the interval $[a, b]$ which is a root of the expression $f(x)$, i.e., x_0 is a value such that $f(x_0) = 0$. MLAB provides the root operator as a built-in function `ROOT`. The value $c_{\max}$ is any value greater than $c(r_b)$, which is an upper bound for $c(r)$.

Now, if we measure the concentration distribution with a UV scanner or with a Rayleigh interferometer attached to the ultracentrifuge and express the results as a table of points (r_i, c_i), which gives the observed concentration c_i at the radial position r_i in the cell, then the root-form equation for $c(r)$ given above can be used as a model to fit these points. In this way, M can be determined by curve-fitting. The virial coefficient B is also generally treated as a fitting parameter. Often c_h is not exactly known, and can also be a fitting parameter. Concentration can be measured in any of a number of different units. Fundamentally, we have fringe numbers or optical densities, and such observations are readily convertible by scaling to grams-per-liter concentration units.

Here is an example showing how the fit is done in *MLAB*. We first define the model, and then read-in some data, set initial-guess values of the parameters to be fitted, and fit the model to the given data. Finally, we draw pictures of the data and the fitted model.

```
fct c(r)=root(x, 0.5,20,(ch*exp(A*M*(r^2-rh^2)/(1+M*B*x))-x))
data = read(centri,51,2) /* read in the data from file centri.dat */

q = 0.26 /* this term is 1-zbar*rho */
omega = 1047.2 /* corresponding to 10000 rpm machine rotation */
R = 8.314*10^7 /* gas concentration */
t = 298.15 /* absolute temprature */
A = (q*omega^2)/(2*R*t); /* A = 5.75119093E-6 */
rh = 6.7   /* reference radius */

/* set initial guesses for B, ch, M */
B = 10^(-7)
ch = 1.25;
M = 60000

fit(M,B,ch), c to data

final parameter values
      value               error                dependency    parameter
 68743.0466061385      1414.898785691          0.9928704243   M
 2.283270139e-07      1.901124729e-08          0.9729368851   B
   0.9799669867        0.0201841151          0.9789628624   ch
4 iterations
CONVERGED
best weighted sum of squares = 3.946440e-01
```

```
weighted root mean square error = 9.067387e-02
weighted deviation fraction = 9.884748e-03
R squared = 9.989925e-01

draw data, lt none, pt star, ptsize 0.01
draw points(c,(data col 1))
top title "MLAB fit of C(r) vs. r"
view
```



Note that our model $c(r)$ is an *implicit* function. One of the features of *MLAB* is the ability to handle such models. The *MLAB* curve-fitting process involves computing the derivatives of c with respect to B, M and c_h. Here are the derivative functions generated by *MLAB*. When differentiating an implicit function, *MLAB* makes use of the *eval* operator defined so that $eval(x, E, F)$ denotes the value of the expression F with the value E *subbstituted* for x within F.

```
* type c'B
FUNCTION c'B(r) = EVAL(x,ROOT(x,0.5,20,ch*EXP((A*M*(r^2-rh^2))/(1+M*B*x))-x),
-(((M*x*A*M*(r^2-rh^2))/((1+M*B*x)*(1+M*B*x)))*EXP((A*M*(r^2-rh^2))
/(1+M*B*x))*ch)/(((M*B*A*M*(r^2-rh^2))/((1+M*B*x)*(1+M*B*x)))
*EXP((A*M*(r^2-rh^2))/(1+M*B*x))*ch+1))
```

```
*
* type c'M
FUNCTION c'M(r) = EVAL(x,ROOT(x,0.5,20,ch*EXP((A*M*(r^2-rh^2))/(1+M*B*x))-x),
(((A*(r^2-rh^2)*(1+M*B*x)-B*x*A*M*(r^2-rh^2))/((1+M*B*x)*(1+M*B*x)))
*EXP((A*M*(r^2-rh^2))/(1+M*B*x))*ch)/(((M*B*A*M*(r^2-rh^2))
/((1+M*B*x)*(1+M*B*x)))*EXP((A*M*(r^2-rh^2))/(1+M*B*x))*ch+1))
*
* type c'ch
FUNCTION c'ch(r) = EVAL(x,ROOT(x,0.5,20,ch*EXP((A*M*(r^2-rh^2))/(1+M*B*x))-x),
EXP((A*M*(r^2-rh^2))/(1+M*B*x))/(((M*B*A*M*(r^2-rh^2))/((1+M*B*x)*(1+M*B*x)))
*EXP((A*M*(r^2-rh^2))/(1+M*B*x))*ch+1))
```

If we have several solutes with unknown molecular weights, then the equation for the combined concentration distribution is merely the sum of the separate concentration distributions. Thus, curve-fitting can be used to determine several molecular weights simultaneously.

In general, we have: $c(r) = c_1(r) + c_2(r) + \ldots + c_n(r)$, where the concentration of the ith solute is $c_i(r) = c_{hi} \exp(A_i N_i (r^2 - r_h^2))$, N_i is the apparent molecular weight of the ith solute, and c_{hi} is the grams per liter concentration of the ith solute at the reference position r_h, and A_i is the usual constant $(1 - \bar{v}\rho)w^2/(2RT)$ described above, particularized for the ith solute.

As before, N_i is a function of r and the true molecular weight, M_i, namely: $N_i \approx M_i/(1 + B_i M_i c(r))$. In fact this is an approximation to the relationship:

$$N_i = M_i/(1 + \sum_{k=1}^{n} \sum_{j=1}^{\infty} B_{ikj} (M_i c_k(r))^j)$$

Usually assuming each value B_{ik1} is the same, namely B_i/n, for $1 \leq k \leq n$ and $B_{ikj} = 0$ for $1 < j$ is adequate, and this results in the simple form used above.

Suppose, for example, we have two solutes with molecular weights M_1 and M_2. Then

$$c(r) = root(x, 0, c_{\max}, (c_{h1} \exp(AM_1(r^2 - r_h^2)/(1 + B_1 M_1 x)) + c_{h2} \exp(AM_2(r^2 - r_h^2)/(1 + B_2 M_2 x)) - x).$$

Also, when several solutes are present, it is of interest to compute the amounts of each component present. When the non-ideality constants B_i are nearly 0, this can be done by using the equation:

$$c_{0i} = c_{hi}(\exp(AN_i(r_h^2 - r_b^2)) - \exp(AN_i(r_h^2 - r_m^2)))/(AN_i(r_b^2 - r_m^2)),$$

where c_{0i} is the initial gram-per-liter concentration of a given solute i (before spinning); c_{hi}, A, and N_i are as before, and r_b is the radial position of the bottom of the cell and r_m is the radial position of the top, or meniscus, of the solution column in the cell.

It is often of interest to compute the amounts of each component present in the ultracentrifuge cell directly from the gram-per-liter concentration profile function. This can be done by using the equation:

$$q_i = \int_{r_m}^{r_b} c_i(r) v dr,$$

where q_i is the total mass in grams of the given solute i, v is a constant based on the geometry of the cell, r_b is the radial position of the bottom of the cell, and r_m is the radial position of the top, or meniscus, of the solution in the cell. For a cell 1.2 cm deep with a chord length of .33 cm at the radial position $r_b = 7.2$ cm while spinning, we have $v = 1.2 \cdot 2 \cdot \arcsin(.33 \cdot 7.2/2)$. It is important to use the most accurate possible geometry constant v and bottom radial position r_b; when spinning, r_b can be approximately 1% greater than the value measured at rest. Finally note that this computation can be confounded when the solute sticks to the the interior walls of the cell or builds up excessively at the bottom of the cell.

The mathematical modeling software MLAB can be used to compute the equilibrium constant K for the simple binding reaction $X + Y \rightleftharpoons Z$ occurring within the ultracentrifuge cell in many cases.

Let $c_1(r)$ denote the grams-per-liter concentration of the first solute substance X at radial position r, let $c_2(r)$ denote the grams-per-liter concentration of the second solute substance Y at radial position r, and let $c_3(r)$ denote the grams-per-liter concentration of the composite third solute substance Z at radial position r. Then the combined concentration distribution function is $c(r) = c_1(r) + c_2(r) + c_3(r)$, where the grams-per-liter concentration distribution of the ith solute is $c_i(r) = c_{hi} \exp(A_i N_i (r^2 - r_h^2))$ for $i = 1, 2, 3$.

Also, at any fixed radial position r, the stochiometric relationship $K = (\frac{c_3(r)}{M_3})/((\frac{c_1(r)}{M_1})(\frac{c_2(r)}{M_2}))$ holds, where M_i denotes the molecular weight of solute i for $i = 1, 2, 3$. Note that $M_3 = M_1 + M_2$. Thus,

$$c_3(r) = \frac{M_1 + M_2}{M_1 M_2} K c_1(r) c_2(r).$$

Let $g_i(r, x) := c_{hi} \exp(A_i M_i (r^2 - r_h^2)/(1 + B_i M_i x))$ for $i = 1, 2, 3$. Note $g_i(r, c(r)) = c_i(r)$. Now define $f(c_1, c_2) := c_1 + c_2 + K c_1 c_2 (M_1 + M_2)/(M_1 M_2)$. Then, $c(r) = root(x, 0, c_{\max}, f(g_1(r, x), g_2(r, x)) - x)$ where $c_{\max}$ satisfies $c_{\max} > \max_r c(r)$. We thus have an implicit model for the c vs. r data corresponding to the simple binding reaction $X + Y \rightleftharpoons Z$ occurring in the ultracentrifuge cell. The potentially-unknown parameters of this model are c_{1h}, c_{2h}, M_1, M_2, B_1, B_2, A_1, A_2 and K. MLAB curve-fitting may now in principle be employed to estimate these parameters. In the case of a self-associative dimerization reaction, this model simplifies with $c_{1h} = c_{2h}$, $M_1 = M_2$, $A_1 = A_2$ and $B_1 = B_2$.

Now, if we measure the total concentration distribution with a UV scanner or with a Rayleigh interferometer attached to the ultracentrifuge and express the results as a table of (r, c) points, each of which gives the observed concentration c at the corresponding specified radial position r in the cell, then the root-form equation for $c(r)$ given above can be used as a model to fit

these points. In this way, various parameters such as K, M_1, M_2, B_1, or B_2 can, in principle, be determined by curve-fitting. Often c_{h1} and c_{h2} are not exactly known, and are also taken as fitting parameters. Generally, the number of distinct fitting parameters must be minimized, however, in order to obtain physically reliable estimates. Concentration can be measured in any of a number of different units. Fundamentally, we have fringe numbers or optical densities, and such observations are readily convertible by scaling to grams-per-liter concentration units. Note if $K = 0$, we have two non-interacting substances in the cell, and estimating M_1, M_2, B_1, and B_2 becomes feasible, In the example given here, however, we are interested in estimating the equilibrium constant K.

Generally we cannot successfully estimate the equilibrium constant K when K is small (say $< .5 \cdot 10^4$,) because our model tends to become increasingly ill-conditioned as K decreases. The exact limit depends on the molecular weights, the initially-loaded amounts, and the virial coefficients of the reactants.

There are several more "rules of thumb" that we should remember when estimating the equlibrium constant K.

(1) The molecular weights M_1 and M_2 should be predetermined and should not be fitting parameters. Any of a number of methods can be used to determine M_1 and M_2; if sequence composition information is known, of course, it should be used to compute the corresponding molecular weight exactly. Similarly, the virial coefficients B_1 and B_2 should likewise be predetermined, if possible. (Note that once we have obtained M_1, M_2, B_1, B_2, c_{1h}, and c_{2h}, we can subtract $c_1(r) + c_2(r)$ from our data, and then fit g_3 to the result in order to estimate B_3 and c_{3h} if this is desired.)

(2) The value $c_{\max}$ should be as small as possible so as to minimize the occurrence of overflows in the derivatives of c. This implies that relatively dilute solutions of the reactants should be used. On the other hand, we need to produce an adequate amount of Z material to be represented in the $c(r)$-data. The speed of the ultracentrifuge should be chosen to obtain a distribution of mass which is neither too "flat" nor too "sharp", *i.e.*, material should not "pile-up" at the bottom of the cell. It may sometimes be convenient to discard the data for radial positions near the bottom of the cell in order to be able to reduce the value of c_{max}. It is generally wise to discard the data near the bottom of cell in any event because the error in measuring mass concentration is large near the boundary of the cell.

(3) It is generally useful to replace K by $exp(L)$, where $L = log(K)$, and then fit to estimate L. This does not generally provide a more exact estimate of K, but the normal theory standard errors computed for L are more likely to be meaningful than are the same quantities for K when estimated directly.

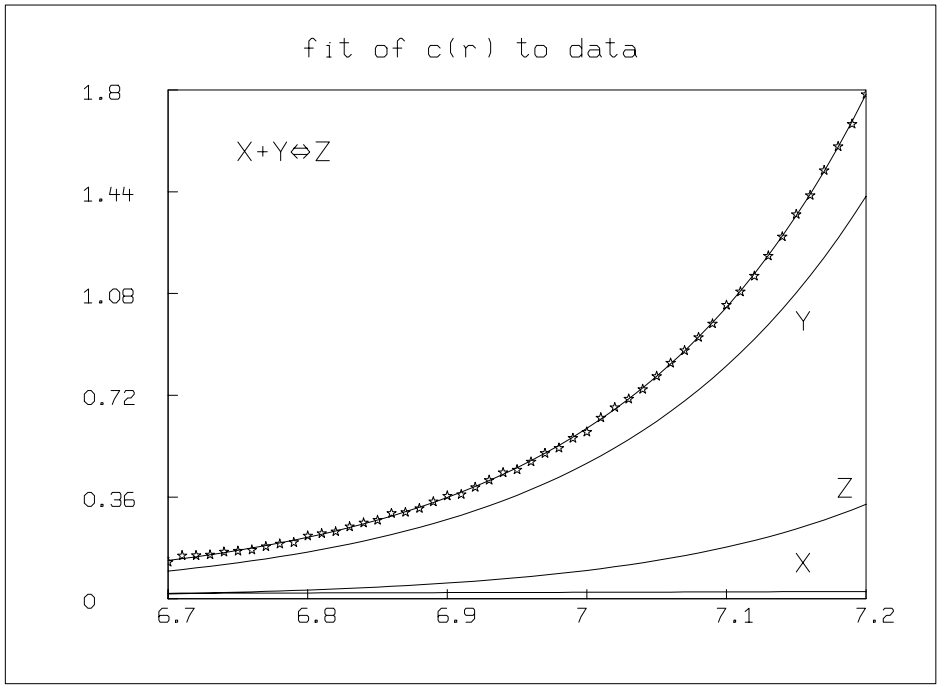
Here is an example of using MLAB to estimate the equilibrium constant for a simple binding reaction. Note we have also used the conservation-of-mass relations for substance 1 (X) and for substance 2 (Y). Choosing parameters to fit the computed masses of X and Y to the known initially-loaded masses is a valuable device to improve our parameter estimates. This can only be done, however, when accurate values of v and r_b are available. (Another device that can help

in obtaining more accurate estimates is to simultaneously fit gram-per-liter concentration-profile data obtained at equilibrium for several different rotor speeds.)

```
* rm = 6.7  /* minimum radius in cm */
* rb = 7.2  /* maximum radius in cm */
* rh = rm   /* reference radius in cm */
* vc = 2*asin(.33/(2*rb))  /*angle of the cell wedge */
* vc=vc*1.2 /*volume constant, total solution vol.= (vc/2)*(rb^2-rm^2) cm^3 */

* r1 = rm:rb:.01 /* radial positions for curve-plotting */

* qv = 0.26 /* this term is 1 - zbar*rho */
* omega = 1047.2 /* corresponding to 10000 rpm machine rotation */
* R = 8.314*10^7 /* gas concentration */
* t = 298.15 /* absolute temperature */
* A1 = (qv*omega^2)/(2*R*t); A2=A1 /*Both A1 and A2 = 5.75119093E-6 */

* /* predetermined values for known parameters */
* B1 = 2*10^(-8); B2=10^(-9);
* M1 = 67000;  M2=6800

* constraints q={9<lk, lk<40, ch1>0, ch2>0, ch1+ch2<1.4}

* fct g1(r,x)=ch1*exp(A1*M1*(r^2-rh^2)/(1+M1*B1*x))
* fct g2(r,x)=ch2*exp(A2*M2*(r^2-rh^2)/(1+M2*B2*x))
* fct f(c1,c2)=c1+c2+exp(lk)*c1*c2*(M1+M2)/(M1*M2)
* fct c(r) = root(x,(ch1+ch2)/3,100000, f(g1(r,x),g2(r,x))-x)

* fct c1(r)=g1(r,c(r))
* fct c2(r)=g2(r,c(r))
* fct c3(r)=c3h(r,c(r))
* fct c3h(r,x)=x-g1(r,x)-g2(r,x)

* /*tm1, tm2  = total mass in grams of substances 1 and 2 */
* fct tm1()=integral(r,rm,rb,tm1z(r,c(r))*vc*r)
* fct tm1f(c1,c2,c)=c1+(M1/(M1+M2))*(c-c1-c2)
* fct tm1z(r,x) = tm1f(g1(r,x),g2(r,x),x)

* fct tm2()=integral(r,rm,rb,tm2z(r,c(r))*vc*r)
* fct tm2f(c1,c2,c)=c2+(M2/(M1+M2))*(c-c1-c2)
* fct tm2z(r,x) = tm2f(g1(r,x),g2(r,x),x)
```

```
* /*specify the total mass in grams of substances 1 and 2 */
* mass1[1]= .113689839;
* mass2[1]= .00613855051

* /* read-in the r vs. c(r) data to be modeled. */
* data = read(data.dat,1000,2)
* n=nrows(data);
*
* /* define the weights for fitting and normalize them to sum to 1 */
* zw=ewt(data); zw=zw/rowsum(zw)

* /*set initial guesses for k, ch1,and ch2 */
* k=10000; lk=log(k)
* ch1=.15;
* ch2=.035

* fit(lk,ch1,ch2), c to data with weight zw, tm1 to mass1, tm2 to mass2,\
: constraints q

 final parameter values
      value               error                dependency    parameter
 10.92859862         0.1533653124            0.9980490348   LK
  0.09847696449      0.003231742687          0.9986427864   CH1
  0.01982622037      0.001226384415          0.9914474897   CH2
4 iterations
CONVERGED
best weighted sum of squares = 2.386287e-05
weighted root mean square error = 6.908382e-04
weighted deviation fraction = 5.217909e-03
R squared = 9.998792e-01
no active constraints

* exp(lk)  /* k=exp(lk) */
    = 55748.1007
* tm1()    /* estimated total mass in grams of X */
    = .113740813
* tm2()    /* estimated total mass in grams of Y */
    = 6.25476406E-3

* draw data, lt none, pt star, ptsize 0.01
* draw points(c,r1)
* top title "fit of c(r) to data"
```

```
* draw points(c1,r1), color red
* draw points(c2,r1), color green
* draw points(c3,r1), color yellow
* title "Y" at (7.15,.95) world
* title "Z" at (7.18,.36) world
* title "X" at (7.15,.095) world
* title "X+Y'2T='RZ" at (6.75,1.55) world
* view
```



The equations above are examples of the type of models involved in the analysis of data obtained from the use of an ultracentrifuge. There are many other useful experiments, with other equations (or differential equations) describing the phenomena involved. A good reference on the use of the ultracentrifuge and the associated mathematical modeling is the chapter by K.E. Van Holde in the book The Proteins, 3rd edition, Vol. 1, 1975. Another source is Foundations of Ultracentrifugical Analysis, by Hiroshi Fujita, Wiley, 1975. MLAB is often useful in analyzing such experiments, and may well succeed where other software fails.

Another example is the situation when we have self-associating molecules in an ultracentrifuge, and we wish to determine the monomer molecular weight, and the equilibrium constants of the known association reactions by curve-fitting.

For example, suppose we have the self-associating reactions:

$$\begin{aligned} c_1 + c_1 &\rightleftharpoons c_2, \quad \text{molar equilibrium constant} = K_{12} \\ c_2 + c_2 &\rightleftharpoons c_4, \quad \text{molar equilibrium constant} = K_{24} \\ c_4 + c_4 &\rightleftharpoons c_8, \quad \text{molar equilibrium constant} = K_{48}. \end{aligned}$$

If we let these reactions take place in an ultracentrifuge cell, and measure the weight-average molecular weight, M_w, of the resulting mixture in grams for different values of c, the total concentration in grams per liter of all the species c_1, c_2, c_4, and c_8, then we can simultaneously determine the monomer molecular weight of the species c_1: M_1, and the equilibrium constants K_{12}, K_{24}, K_{48}, by curve-fitting.

We have $M_w = (c_1M_1 + c_2M_2 + c_4M_4 + c_8M_8)/c$, where M_i is the molecular weight of c_i, and the symbol c_i, as a function, denotes the gram-per-liter equilibrium concentration of the species c_i. c is the total gram-per-liter concentration, so $c = c_1 + c_2 + c_4 + c_8$.

Note that $M_2 = 2M_1$, $M_4 = 4M_1$, and $M_8 = 8M_1$. Also,

$$\begin{aligned} K_{12} &= (c_2/M_2)/(c_1/M_1)^2, \\ K_{24} &= (c_4/M_4)/(c_2/M_2)^2, \quad \text{and} \\ K_{48} &= (c_8/M_8)/(c_4/M_4)^2. \end{aligned}$$

Thus, the corresponding equilibrium constants for gram-per-liter concentrations can be defined as:

$$\begin{aligned} E_{12} &= 2K_{12}/M_1 \\ E_{24} &= K_{24}/M_1, \quad \text{and} \\ E_{48} &= K_{48}/(2M_1), \end{aligned}$$

so that,

$$\begin{aligned} c_2 &= E_{12}c_1^2, \\ c_4 &= E_{24}c_2^2 = E_{24}E_{12}^2c_1^4, \quad \text{and} \\ c_8 &= E_{48}c_4^2 = E_{48}E_{24}^2E_{12}^4c_1^8. \end{aligned}$$

Define $E_{14} = E_{24}E_{12}^2$, and $E_{18} = E_{48}E_{24}^2E_{12}^4$.

Now, $c = c_1 + c_2 + c_4 + c_8$, so $c = c_1 + E_{12}c_1^2 + E_{14}c_1^4 + E_{18}c_1^8$, and thus, $c_1 = root(x, 0, c, (x + E_{12}x^2 + E_{14}x^4 + E_{18}x^8 - c))$.

Also, $M_w = M_1(c_1 + 2E_{12}c_1^2 + 4E_{14}c_1^4 + 8E_{18}c_1^8)/c$.

Due to the non-ideal gas behavior of most materials, the observed weight-average molecular weight N is, in fact, more nearly the following non-linear function of the actual weight-average molecular weight: $N = M_w/(1 + BM_wc)$, where the parameter B is the so-called virial coefficient.

Thus, given data: c vs. N, we can fit for the parameters E_{12}, E_{24}, E_{48}, B, and M_1 in the following model, as defined in MLAB.

```
*FUNCTION MA(C) = NID(C,M1*MWX(C,E12,E14,E18),B)
*FUNCTION NID(C,MW,B) = MW/(1+B*C*MW)
*FUNCTION MWX(C,A1,A2,A3) = P(C1(C,A1,A2,A3),A1,A2,A3)/C
*FUNCTION P(C1,A1,A2,A3) = C1+2*A1*C1^2+4*A2*C1^4+8*A3*C1^8
*FUNCTION C1(C,A1,A2,A3) = ROOT(X,0,C,X+A1*X^2+A2*X^4+A3*X^8-C)
```

In order to obtain the data c vs. N, preliminary analysis of the actual observations is required. What is, in fact, observed is r vs. c, where r is a radial position in the cell and where the total gram-per-liter concentration c is observed as a function of r. We have the fundamental relation:

$$c(r) = c_h \exp(AN(r^2 - r_h^2)),$$

where A, c_h, and r_h are defined above.

Now one can take a few neighboring observations, say five points $(r_1, c(r_1))$, $(r_2, c(r_2))$, ..., $(r_5, c(r_5))$, and fit the above equation to these points with $c_h = c(r_3)$, and $r_h = r_3$, and thus determine the value of N which corresponds to $c(r_3)$. Sliding this operation over all the observations transforms the data so that the model above for c vs. N data is now applicable.

Alternatively, one may use the r vs. c observations directly by using a model based on the following equations.

$$\begin{aligned} c_1(r) &= c_{1h} \exp(AN_1(r_2 - r_h^2)), \\ N_1 &= M_1/(1 + BM_1c(r)), \quad \text{and} \\ c(r) &= c_1(r) + E_{12}c_1(r)^2 + E_{14}c_1(r)^4 + E_{18}c_1(r)^8. \end{aligned}$$

In MLAB we have:

```
*FUNCTION C(R) = P(C1(R,B,M1,C1H),E12,E14,E18)
*FUNCTION P(X,A1,A2,A3) = X+A1*X^2+A2*X^4+A3*X^8
*FCT C1(R,B,M1,CH1) = ROOT(Y,0,MAXC1,Y-CH1*EXP(A*(R*R-RH*RH)*M1/(1+B*M1*Y)))
```

Although using this model to fit r vs. c data to obtain the parameters B, M_1, E_{12}, E_{14}, E_{18}, and possibly c_{h1}, is probably more accurate than the indirect method above, it has the disadvantage that when, as is common, different experiments with different c_{h1} values (and perhaps different A values) are used to cover a wider range of c values, then a curve-fit involving several models being fit simultaneously to adjust shared parameters must be employed, and this is impractical when too many experiments are involved.

In either case, K_{12}, K_{24}, and K_{48} can then be determined from E_{12}, E_{14}, E_{18}, and M_1. Similar models can be obtained for many other specific forms of self-association. The special case where all possible species can arise due to sequential binding, with each binding requiring the same change of free energy as any other is discussed below.

Suppose we have a non-specific isodesmic "stacking" reaction involving a monomer c_1.

$$c_1 + c_1 \rightleftharpoons c_2, \quad \text{molar equilibrium constant} = K \tag{0.1}$$
$$c_1 + c_2 \rightleftharpoons c_3, \quad \text{molar equilibrium constant} = K \tag{0.2}$$
$$\vdots \qquad \vdots \tag{0.3}$$

An isodesmic association is one where the free energy for adding one more monomer, c_1, to an oligomer, c_n, is the same for all n.

As before, let c_i also denote the number of grams-per-liter of the species c_i in the cell at equilibrium, thus c_i is the grams-per-liter concentration of the identically-named species. Also let $c = c_1 + c_2 + \ldots$, and $M_w = (c_1M_1 + c_2M_2 + \ldots)/c$, where M_i is the molecular weight of the species c_i; thus c is the total grams-per-liter concentration of all species present, and M_w is the weight-average molecular weight of all species.

Now, $K = (c_{i+1}/M_{i+1})/((c_1/M_1)(c_i/M_i))$ so $c_{i+1}/M_{i+1} = K(c_1/M_1)(c_i/M_i)$, and hence by successive substitution $c_{i+1}/M_{i+1} = K_i(c_1/M_1)^{i+1}$.

Thus $c_i = i(K/M_1)^{i-1}c_1^i$, and hence $c = c_1(1 + 2Kc_1/M_1 + 3(Kc_1/M_1)^2 + \ldots)$. But $1 + 2x + 3x^2 + \ldots = 1/(1-x)^2$, so $c = c_1/(1 - Kc_1/M_1)^2$, where K is the molar equilibrium constant, subject to the constraint that $Kc_1/M_1 < 1$.

Also, $M_w = (M_1c_1 + 2M_1(2(Kc_1/M_1)c_1) + 3M_1(3(Kc_1/M_1)2c_1) + \ldots)/c$, so $M_w = c_1M_1(1 + 2^2(Kc_1/M_1) + 3^2(Kc1/M1)2 + \ldots)/c$; and $1^2 + 2^2x + 3^2x^2 + \ldots = (1+x)/(1-x)^3$, so $M_w = c_1M_1(1 + Kc_1/M_1)/(1 - Kc_1/M_1)^3$/c. Substituting $c_1/(1 - Kc_1/M_1)^2$ for c, we have $M_w = M_1(M_1 + Kc_1)/(M_1 - Kc_1)$.

Now, c_1 is the function of c given by $c_1(c) = root(x, 0, c, x/(1 - Kx/M_1)^2 - c)$, and in terms of this function $M_w = M_1(M_1 + Kc_1(c))/(M_1 - Kc_1(c))$. As before the apparent weight-average

molecular weight N_w is approximately given by $M_w/(1 + BM_wc)$, so we may use the model

$$N_w(c) = M_1/((M_1 - Kc_1(c))/(M_1 + Kc_1(c)) + BM_1c)$$

to fit c vs. N_w data.

Also, since $c(r) = c_h \exp(AN_w(r_2 - r_h^2))$, we can use the model

$$c(r) = root(x, 0, c_{max}, (x - c_h \exp(AN_w(x)(r_2 - r_h^2))))$$

to fit r vs. c data.

6 Second-Order Binding

Consider the reaction $F + G \underset{K_2}{\overset{K_1}{\rightleftharpoons}} B$.

The study of this reaction is common in chemistry and biochemistry. For example, F could be a hormone or drug and G the associated receptor sites. The symbol B represents the bound complex.

Let us also allow $F(t)$, $G(t)$, and $B(t)$ to be functions which specify the concentrations of F, G, and B respectively at time t. Then we have the differential equation model:

$$\begin{aligned} dB/dt(t) &= K_1F(t)G(t) - K_2B(t), \quad B(0) = B_0, \\ F(t) &= F_0 - (B(t) - B_0), \\ G(t) &= G_0 - (B(t) - B_0), \end{aligned}$$

where F_0, G_0, and B_0 are the initial concentrations of F, G, and B respectively, and the molar association and dissociation rate constants K_1 and K_2 appear as the proportionality constants for the terms which occur in dB/dt.

Note that $K_1 = (dB/dt(0) + K_2B_0)/(F_0G_0)$, so K_1 may be estimated from the initial velocity $dB/dt(0)$, which can, in turn, be estimated from a few points.

The solution to our differential equation is:

$$B(t) = (S(B_0 - R) - R(B_0 - S)e^{d\cdot K_1\cdot t})/(B_0 - R - (B_0 - S)e^{d\cdot K_1\cdot t}),$$

where

$$\begin{aligned} S &= A + d/2 \\ R &= A - d/2 \\ d &= 2(A^2 - (F_0 + B_0)(G_0 + B_0))^{1/2} \\ A &= (F_0 + G_0 + 2B_0 + K_2/K_1)/2 \end{aligned}$$

An appropriate definition of this function in MLAB involves using auxiliary functions as follows:

```
fct B(T) = H1((F0+G0+2*B0+K2/K1)/2,T)
fct H1(A,T) = H2(A,SQRT(4*(A^2-(F0+B0)*(G0+B0))),T)
fct H2(A,D,T) = H3(A+D/2,A-D/2,EXP(D*K1*T))
fct H3(S,R,E) = ((B0-R)*S-(B0-S)*R*E)/(B0-R-(B0-S)*E)
```

It is useful to study this example carefully. Many occasions will arise where functions will need to be expressed with sub-functions in a similar manner.

Suppose we have specific kinetic data, time versus B-concentration, appearing as the rows of a two-column matrix, BM. Note time vs. G-concentration or time vs. F-concentration can be easily converted to time vs. B-concentration. In MLAB, if GM is a 2 column matrix of time vs. G-concentration for example, we merely type

```
*BM = (GM COL 1)&'(GO-BO-(GM COL 2))
```

and BM is then the desired time vs. B-concentration matrix of data points.

We may use the curve-fitting facility of MLAB to compute estimates of K_1 and K_2, and even F_0, G_0, and/or B_0 if necessary.

The equilibrium constant, K, of the reaction, is defined as $K = K_1/K_2$. At equilibrium (say at time t_e) we have $dB/dt(t_e) = 0$, and hence $K_1F(t_e)G(t_e) - K_2B(t_e) = 0$.

Thus, $K = B(t_e)/(F(t_e)G(t_e)) = B(t_e)/((F_0 + B_0 - B(t_e))(G_0 + B_0 - B(t_e)))$.

Define $B_e(F_0, G_0, B_0) = B(t_e)$, the amount of B at equilibrium or "saturation". Then solving for $B_e(F_0, G_0, B_0)$, we have the following equation, called the *saturation equation*

$$B_e(F_0, G_0, B_0) = [(F_0 + G_0 + 2B_0 + 1/K) - ((F_0 + G_0 + 2B_0 + 1/K)^2 - 4(F_0 + B_0)(G_0 + B_0))^{1/2}]/2.$$

If we have data points for a so-called saturation curve consisting of pairs of F_0 values with associated B_e values for fixed values of G_0 and B_0, then curve-fitting can be used with the above function in order to estimate K. Indeed K can be estimated even when we have points (F_0, G_0, B_e) from the saturation surface in 3-space, where arbitrary values of F_0 and G_0 have been paired, and B_0 is fixed.

Define $B_e = B(t_e)$, $F_e = F(t_e)$, and $G_e = G(t_e)$. Then from the basic relations: $K = B_e/(F_eG_e)$, $F_e + B_e = F_0 + B_0$, and $G_e + B_e = G_0 + B_0$, we may write a number of equivalent relationships.

Michaelis-Menten Equation(1) (B_e vs. F_e):

$$B_e = K(G_0 + B_0)F_e/(1 + KF_e)$$

Michaelis-Menten Equation(2) ($B_e/(B_0 + G_0)$ vs. F_e):

$$B_e/(B_0 + G_0) = KF_e/(1 + KF_e)$$

Lineweaver-Burke Equation ($1/B_e$ vs. $1/F_e$):

$$1/B_e = (1/(K(G_0 + B_0))(1/F_e) + 1/(G_0 + B_0)$$

Eadie-Wilkinson-Dixon Equation (F_e/B_e vs. F_e):

$$F_e/B_e = F_e/(G_0 + B_0) + 1/(K(G_0 + B_0))$$

Scatchard Equation (B_e/F_e vs. B_e):

$$B_e/F_e = -KB_e + K(G_0 + B_0)$$

Hill Equation ($\log[(B_e/(G_0 + B_0))/(1 - B_e/(G_0 + B_0)]$ vs. $\log F_e$):

$$\log((B_e/(G_0 + B_0)/(1 - B_e/(G_0 + B_0))) = \log F_e + \log K$$

(Also see the "direct linear plot" of Cornish-Bowden in Biochem. J. Vol 137, p. 143, 1974)

Some of these relationships are inspired by analogous relations for enzyme reactions where they arise in different forms. Most are linear relations in K or $1/K$ for simple binding, and this accounts for their popularity—they are easy to use as models with linear regression methods in order to estimate K. For the non-linear Michaelis-Menten forms, constraints are often necessary. *In spite of their traditional use, however, the errors introduced when transforming data to the appropriate form limits the accuracy obtainable when using any of these models; and in general biased estimates of K will result.* See Rodbard, D., "Mathematics of Hormone-Receptor Interaction", in Receptors for Reproductive Hormones, Plenum Pub. Corp.

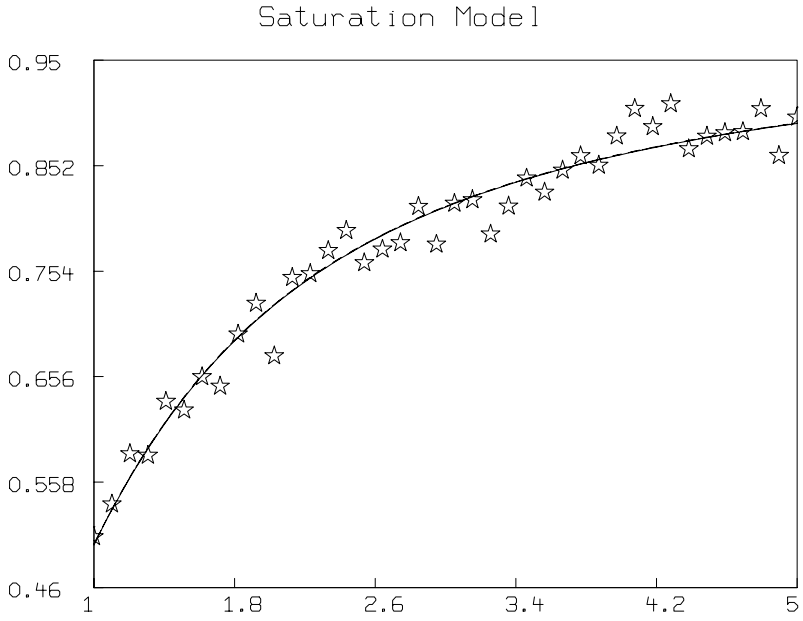
The saturation equation above may thus be preferred, although usually there is little difference. At any rate, the values of K obtained using various models should be checked by computing theoretical predicted values for B_e vs. F_0 and comparing them to the observed values. The major difficulty for the various linear forms is that both the independent and the dependent-variable data values have non-normally-distributed error. As a result, linear Euclidean curve-fitting (with appropriate weights) should be employed with these models. The results should be checked in the saturation equation. That value of K which yields the lowest sum-of-squares in the saturation model should be used.

Here is an example comparing the saturation equation model with the Michaelis-Menten (1) model and the Scatchard model.

```
FCT B(F0)=((F0+G0+2*B0+1/K)-SQRT((F0+G0+2*B0+1/K)^2-4*(F0+B0)*(G0+B0)))/2
FCT BE(FE) = (G0+B0)*FE/(1/K+FE)
FCT BS(BE) = -K*BE+K*(G0+B0)

B0 = 0; G0 = 1;

/* M = F0 values, BE values */
M = read(data,100,2)
```

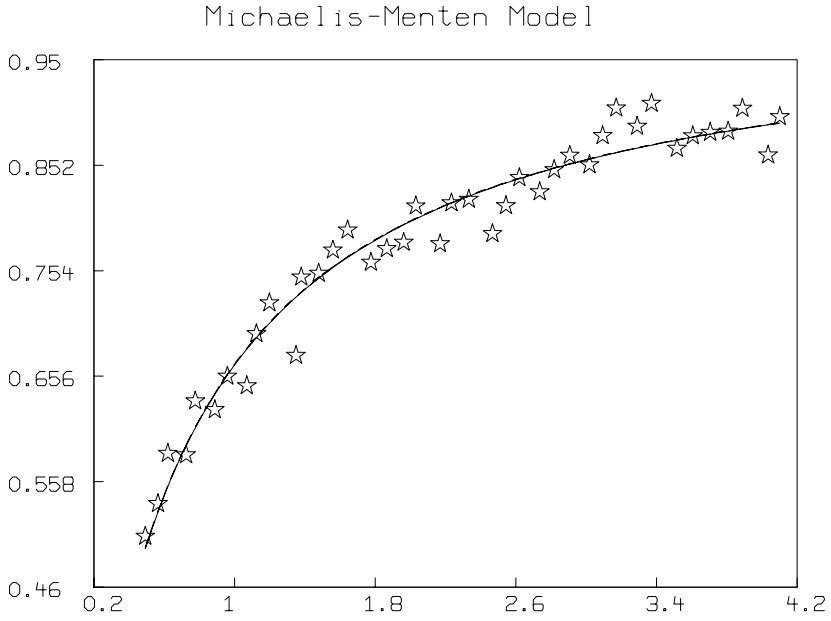
```
/* generate M1 = corresponding data (FE,BE) for the Michaelis-Menten model */
M1 = (M COL 1) - (M COL 2) + B0
M1 COL 2 = M COL 2

/* generate M2 = corresponding data (BE/FE,BE) for the Scatchard model */
M2 = (M COL 2)
M2 COL 2 = (M1 COL 2)/'(M1 COL 1)

K = 2
FIT(K), B TO M
final parameter values
      value               error                dependency    parameter
    1.996602727        0.0420400256                     0   K
1 iterations
CONVERGED
best weighted sum of squares = 1.446128e-02
weighted root mean square error = 1.925622e-02
weighted deviation fraction = 1.849765e-02
R squared = 9.691570e-01
KS = K

FIT(K), BE TO M1
final parameter values
      value               error                dependency    parameter
   2.0043749954        0.0397621839                     0   K
1 iterations
CONVERGED
best weighted sum of squares = 1.769429e-02
weighted root mean square error = 2.130023e-02
weighted deviation fraction = 2.026365e-02
R squared = 9.622617e-01
KM = K

FIT(K), BS TO M2
final parameter values
      value               error                dependency    parameter
   2.0008431423        0.0360417006                     0   K
1 iterations
CONVERGED
best weighted sum of squares = 1.242737e-01
weighted root mean square error = 5.644913e-02
```

```
weighted deviation fraction = 8.426590e-02
R squared = 9.358569e-01
KC = K

/* draw MM model + data with the above K */
K = KS
TOP TITLE "Saturation Model"
DRAW M, LT NONE, PT STAR
DRAW POINTS(B, 1:5!101)
K = KM
DRAW POINTS(B, 1:5!101) LT DASHED
K = KC
DRAW POINTS(B, 1:5!101) LT ALTERNATE
VIEW
```

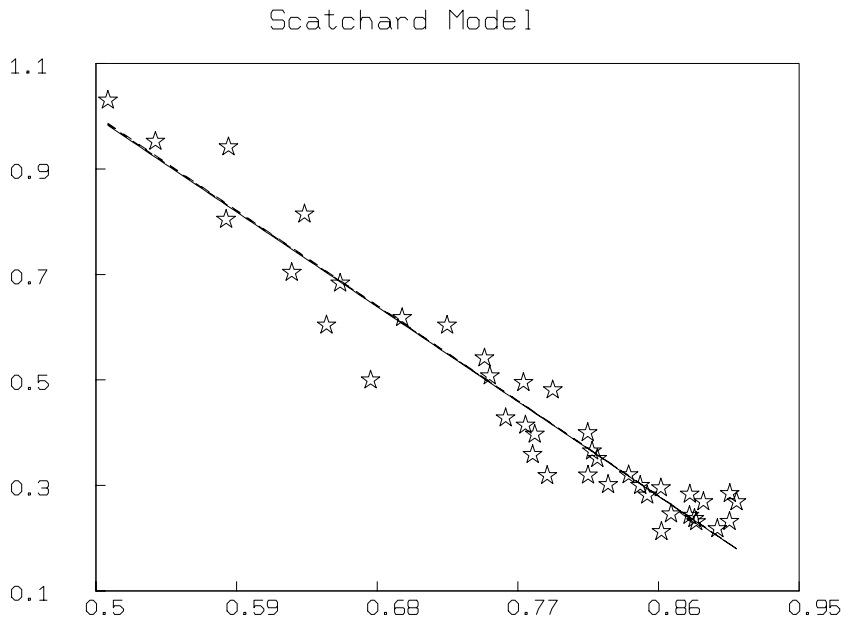



```
K = KS
TOP TITLE "Michaelis-Menten Model"
DRAW M1, LT NONE, PT STAR
DRAW POINTS(BE, MINV(M1 col 1):MAXV(M1 col 1)!101)
K = KM
DRAW POINTS(BE, MINV(M1 col 1):MAXV(M1 col 1)!101) LT DASHED
```

```
K = KC
DRAW POINTS(BE, MINV(M1 col 1):MAXV(M1 col 1)!101) LT ALTERNATE
VIEW
```




```
/* draw Scatchard model + data with the above K */
K = KS
TOP TITLE "Scatchard Model"
DRAW M2, LT NONE, PT STAR
DRAW POINTS(BS, MINV(M2 col 1):MAXV(M2 col 1)!101)
K = KM
DRAW POINTS(BS, MINV(M2 col 1):MAXV(M2 col 1)!101) LT DASHED
K = KC
DRAW POINTS(BS, MINV(M2 col 1):MAXV(M2 col 1)!101) LT ALTERNATE
VIEW
```



Note that for the data studied above, all the models give comparable results. Indeed the estimated K-values are so close, the curves are almost superimposed.

The second Michaelis-Menten equation is useful when the amount, $G_0 + B_0$, is not known. Then measuring a quantity proportional to $B_e/(G_0 + B_0)$ vs. F_e is commonly done, and K can be determined in unknown units. Indeed, by introducing another parameter, D, to obtain $B_e/(B_0 + G_0) = DKF_e/(1+KF_e)$ and computing D and K by fitting this model to data points $(F_e, B_e/(B_0+G_0))$, where F_e and B_e are measured in moles and B_0+G_0 is measured in grams, then $D(B_0+G_0)$ has the unit moles, and so $(B_0+G_0)/(D(B_0+G_0)) = 1/D$ is the molecular weight of a G molecule. (This device for computing molecular weight assumes that $B_0 = 0$ or that an F molecule is much lighter than a G molecule.) Fitting the saturation equation to obtain both K and $G_0 + B_0$ may be a better approach.

Cooperative Binding

Often the binding of F and G is complicated by cooperative effects. Namely, K_1 and/or K_2 appear to be dependent upon the relative amount of B. This can be due to allosteric shape changes in the molecules or sites G which occur during binding. Various other explanations, including multiple classes of sites, are also possible. If K_1/K_2 increases as B increases, we have positive cooperativity; if K_1/K_2 decreases as B increases we have negative cooperativity.

Suppose, then, that K_1 and K_2 are functions of B. Thus,

$$dB/dt(t) = K_1(B(t)) \cdot F(t) \cdot G(t) - K_2(B(t)) \cdot B(t),$$

with $F(t) = F_0 - (B(t) - B_0)$, $G(t) = G_0 - (B(t) - B_0)$, and $B(0) = B_0$, as before.

Now suppose that all cooperative effects are due to changes in K_2, in particular, suppose

$$K_1(B) = K_1^0, \quad \text{and} \quad K_2(B) = K_2^0(1 + p \cdot B/(G_0 + B_0)).$$

This is the same as saying $K_2(B) - K_2(0) = pB/(G_0 + B_0)$, thus we assume that the change in K_2 from the "ground state" $K_2(0) = K_2^0$, is proportional to the fraction of occupied sites, $B/(G_0 + B_0)$, with the proportionality-constant p.

Note then, that $dK_2/dt(B(t)) = (p/(G_0 + B_0))dB/dt(t)$, so $dK_2/dB(B) = p/(G_0 + B_0)$.

There are of course many other functional relationships which could be postulated. For example, we could assume that $dK_2/dt(B(t)) = A(dB/dt)h$, or we could assume K_1 and K_2 vary together in certain ways. Indeed, changes in K_1 will give cooperative effects unobtainable by changes in K_2 alone. K_1 and K_2 need not change monotonically; we may have variation which results in intervals of positive cooperativity and other intervals of negative cooperativity.

Since cooperativity, without qualification as to its cause, is merely a mathematical description, and not a structural description, the choice of how $K_1(B)$ and $K_2(B)$ are defined is dependent upon the actual physical situation and the desired uses of the mathematical model. The particular choice here has the same effect as that made by DeMeyts in his analysis of cooperativity (DeMeyts, P., Woebroeck, M., "The structural basis of insulin-receptor binding and cooperative interactions", in Membrane Proteins (ed. P.Nicholls et al.) FEBS 11th Meeting, Vol. 45 Symposium A4, Pergamon Press, pp. 319–323, 1977).

Now let t_e be the time when equilibrium is approached, and let $B(t_e) = B_e$, $F(t_e) = F_e$, and $G(t_e) = G_e$. Then, at equilibrium, we have the equilibrium constant K as a function of B_e, $K(B_e) = K_1(B_e)/K_2(B_e) = B_e/(F_eG_e)$. Thus, $K(B_e) = K_1^0/(K_2^0(1 + pB_e/(G_0 + B_0)))$, or, $K(B_e) = K_0/(1 + pB_e/(G_0 + B_0))$, where $K_0 = K(0)$.

Note for $-1 < p < 0$, we have positive cooperativity, for $p = 0$, we have no cooperativity ($K(B_e) = K(0)$), and for $p > 0$, we have negative cooperativity.

It is convenient to define p in terms of another parameter, a, called the F, G interaction factor, so that $p = (1 - a)/a$, and hence $a = 1/(1 + p)$. Note for $0 < a < 1$, we have negative cooperativity, for $a = 1$, we have no cooperativity, and for $a > 1$, we have positive cooperativity.

Indeed, if ΔG_0 is the energy needed (or released) (i.e., the change in free energy) for binding the first F molecule to a G molecule, and if ΔG_1 is the energy used (or released) for binding an F molecule to the last unoccupied G molecule (whereupon $B = G_0 + B_0$), then we have the classical thermodynamic relations: $\Delta G_0 = -RT \log K(0)$, and $\Delta G_1 = -RT \log K(G_0 + B_0)$, where R is the gas constant (about 1.987 calories/degree/mole) and T is the absolute temperature. Thus, $K(G_0 + B_0)/K(0) = \exp(-(\Delta G_1 - \Delta G_0)/RT)$, and, by assumption, $K(G_0 + B_0)/K(0) =$

$1/(1+p) = a$, so a has the interpretation: $a = K(B_0+G_0)/K(0)$; it is the ratio of the equilibrium constant K with all G-sites occupied, to K with no occupied G-sites.

DeMeyts has observed that the relation $K(B_e) = K_0/(1+((1-a)/a)\cdot B_e/(G_0+B_0))$ may be used in the various equilibrium models given before to obtain the corresponding cooperative models. Thus, we substitute $K(B_e)$ for K to obtain models which now involve the parameters F_0, G_0, B_0, a, and K_0. Curve-fitting which yields a value for a obviously different from one, should be followed by fitting with a fixed to one. If the latter fit is clearly inferior, cooperative binding phenomena may be present.

In particular, for $a \neq 1$, the cooperative Michaelis-Menten(2) relation is:

$$B_e/(B_0+G_0) = (-a(1+K_0F_e) + [a^2(1+K_0F_e)^2 + 4a(1-a)K_0F_e]^{1/2})/(2(1-a)).$$

The cooperative Scatchard equation is:

$$B_e/F_e = K_0(B_0+G_0-B_e)/(1+(1-a)B_e/(a(G_0+B_0))).$$

The cooperative Hill equation is obtainable by substitution from the cooperative Michaelis-Menten (2) equation above, however, it can be plotted in MLAB, without using explicit algebra, as follows, assuming B_0, G_0, K_0, and a are already set, with $a \neq 1$.

```
FCT BE(FE) = (B0+G0)*(-A*(1+K0*FE)+ \
     SQRT(A*A*(1+K0*FE)^2+4*A*(1-A)*K0*FE))/(2*(1-A))
FUNCTION HILLT(BE) = LOG((BE/(G0+B0))/(1-BE/(G0+B0)))
LFEV = -12:2:.2
M = LFEV &' (HILLT ON BE ON EXP ON LFEV)
DRAW M
```

The Hill-plot matrix M has rows which are points of the form:

$$\log((B_e/(G_0+B_0))/(1-B_e/(G_0+B_0))) \quad \text{vs.} \quad \log F_e.$$

Normally a Hill-plot, as defined above, is a straight line with slope 1, however, this is not the case for $a \neq 1$. DeMeyts has shown that, in general, the slope

$$d(\log((B_e/(G_0+B_0))/(1-B_e/(G_0+B_0))))/d(\log F_e)$$

decreases when a decreases and increases when a increases, and attains its minimum value when $F_e = 1/K_0$, independently of a.

7 Step-Wise Multiple-Site Binding

Cooperative effects often arise when there are multiple binding sites for a ligand that are located spatially close to one another. Some of the issues that arise are discussed in "Mixed and uniform cooperativity of ligand binding to multisite proteins: The cooperativity types allowed by the Adair equation and conditions for them" by Edward P. Whitehead, in the Journal of Theoretical Biology, pp. 153:170, vol. 87, 1980.

A particular situation of interest is that where many F molecules bind step-wise through a series of reactions to a G molecule with distinct affinities. For example, oxygen binds to hemaglobin in this manner. This is one explanation for apparent cooperative binding. Thus we may consider the situation:

$$\begin{aligned}
F + G_0 &\underset{D_1}{\overset{A_1}{\rightleftharpoons}} G_1, \\
F + G_1 &\underset{D_2}{\overset{A_2}{\rightleftharpoons}} G_2, \\
&\vdots \\
F + G_{i-1} &\underset{D_i}{\overset{A_i}{\rightleftharpoons}} G_i, \\
&\vdots \\
F + G_{N-1} &\underset{D_N}{\overset{A_N}{\rightleftharpoons}} G_N,
\end{aligned}$$

where G_0 is defined as free G.

When N (the number of F-binding sites on each G molecule) is not large, the kinetic model differential equations can be used in curve-fitting; however, in general, we deal with the equilibrium model instead.

Define the molar equilibrium constants $K_i = A_i/D_i$. Let $F(t)$ be the concentration of (unbound) F at time t, let $G_i(t)$ be the concentration of G_i at time t, and let F_e and G_{ie} be these concentrations at $t = t_e$, the time when our system approaches equilibrium. Then: $K_i = G_{ie}/(G_{i-1,e}F_e)$, so $K_1K_2 \ldots K_i = G_{ie}/(G_{0e}F_e^i)$. Now we may define $B_i = K_1K_2 \ldots K_i$.

Now, let F_b be the concentration of bound F molecules at equilibrium, so $F_b = G_{1e} + 2G_{2e} + \ldots + NG_{Ne}$. Note that $F_b + F_e$ is the total concentration of F present, *i.e.* $F_b + F_e = F(0)$. Also, let H be the concentration of G molecules in either a bound or free state, so $H = G_{0e} + G_{1e} + \ldots + G_{Ne}$.

Now, define v as the mean number of F molecules bound to each G molecule. Then $v = F_b/H$, or

$$v = (G_{1e} + 2G_{2e} + \ldots + NG_{Ne})/(G_{0e} + G_{1e} + \ldots + G_{Ne}).$$

But, $G_{ie} = B_i G_{0e} F_e^i$, so

$$v = (B_1 G_{0e} F_e + 2B_2 G_{0e} F_e^2 + \ldots + N B_N G_{0e} F_e^N)/(G_{0e} + B_1 G_{0e} F_e + B_2 G_{0e} F_e^2 + \ldots + B_N G_{0e} F_e^N),$$

or

$$v = (B_1 F_e + 2B_2 F_e^2 + \ldots + N B_N F_e^N)/(1 + B_1 F_e + B_2 F_e^2 + \ldots + B_N F_e^N).$$

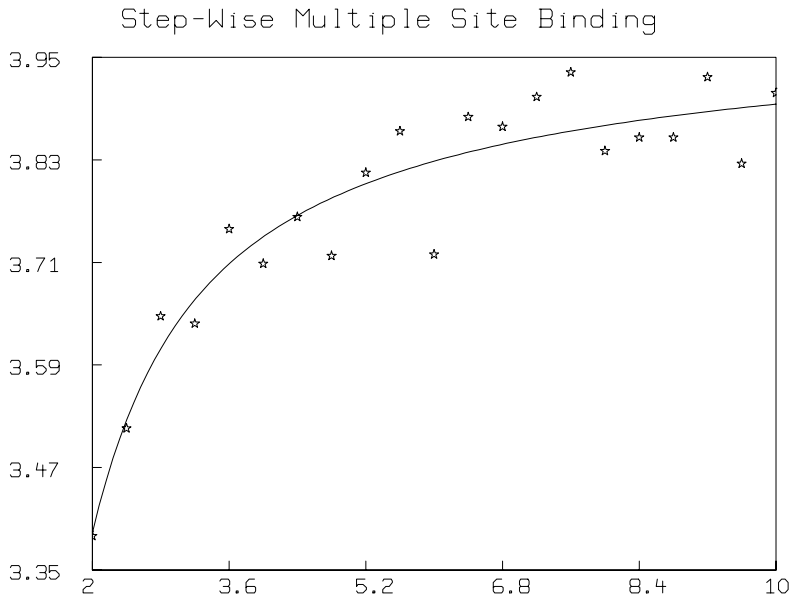
This equation is known as the Adair-Klotz stepwise equilibrium model.

Now given data points (F_e, F_b), each based on different initial values of H and $F_e + F_b$, corresponding data points of the form $(F_e, F_b/H)$ can be constructed, and v can be treated as a function of F_e and fit to such data thus allowing the parameters B_1, B_2, ..., B_N to be estimated (which hence provides estimates of the equilibrium constants K_1, K_2, ..., K_N). If the number of sites, N, is not known, N can be set to 1, 2, 3, etc., and that value of N which yields the best fit can be taken as the estimate of the true N-value. Note the model for data points of the form (F_e, G_{0e}) can be expressed in terms of a `ROOT` expression in $G_0(0)$ and K_1, ..., K_N; but no nice general form is apparent. Note also that data of the form $(F_e, F_b/H)$ has error in both the first and second components. This means that, at a minimum, correct weights should be used in fitting.

The following is a *MLAB* dialog that demonstrates the above mentioned curve fitting for $N = 4$. We first define the model function $v(F_e)$, read-in the data, set the initial guesses for B_1, B_2, B_3 and B_4, and then fit the data to the model.

```
fct p(x) = 1+sum(I,1,N, B[I]*x^I)
fct v(fe) = fe*p'x(fe)/p(fe) /* model function */
N = 4 /* number of reaction steps */
data = read(msb, 21, 2)
t = minv((data col 1)):maxv(data col 1)!100

/* initial guesses */
B = 2:5

fit(B), v to data with weight ewt(data)

final parameter values
      value               error                dependency    parameter
   2.7807664141      164.6641428061          0.9999816051   B[1]
   0.0702266931       16.0436904339           0.999253623   B[2]
   3.1455369331      133.7340925626          0.9999952965   B[3]
   2.7828358122      116.4894630769          0.9999972398   B[4]
3 iterations
CONVERGED
best weighted sum of squares = 3.047449e+01
```

```
weighted root mean square error = 1.338886e+00
weighted deviation fraction = 9.451356e-03
R squared = 8.959082e-01

draw data lt none pt star ptsize 0.01
draw points(v,t)
top title "Step-Wise Multiple Site Binding"
view
```



In order to curve-fit for $B_1, \ldots, B_N$, this model often requires accurate guesses, and these may be difficult to obtain. One method is to obtain such guesses following section 10 of "Models in Regression", by Peter Sprent. The idea is to convert our model to a related linear model which admits a unique solution with arbitrary guesses.

We have $v = B_1F_e(1-v) + B_2F_e^2(2-v) + \ldots + B_NF_e^N(N-v)$, so we can establish the data matrix whose columns are values of $F_e(1-v), F_e^2(2-v), \ldots, F_e^N(N-v), v$, and fit the above model to obtain $B_1, \ldots, B_N$, using as weights the values $1 + B_1F_e + \ldots + B_NF_e^N$, based on the initial guesses for $B_1, \ldots, B_N$. The resulting B-values should then be used to recompute the weights $1 + B_1F_e + \ldots + B_NF_e^N$, and then this fitting process should be reiterated in this manner. *MLAB* can provide such iterative reweighting by specifying a weight function. In general constraints must be used, or special deviations from this process must be introduced to cope with negative B-values; for example each value B_i may be replaced with $\max(B_i, 0)$. Finally, the results obtained may be used as initial guesses for a non-linear regression analysis using the original model for v.

Consider the polynomial $p(x) = 1 + B_1x + B_2x^2 + \ldots + B_Nx^N$. Note $v(F_e) = F_e\frac{dp}{dx}(F_e)/p(F_e)$. Fletcher, Spector, and Ashbrook have shown in their paper, "Analysis of macromolecule-ligand binding by determination of stepwise equilibrium constants", Biochemistry, Vol. 9, pp. 4580:4587, 1970, that if $p(x)$ has N real roots, R_1, R_2, ..., R_M of multiplicities N_1, N_2, ..., N_M respectively, then by partial-fraction decomposition, v can be written as:

$$v(F_e) = N_1r_1F_e/(1 + r_1F_e) + \ldots + N_Mr_MF_e/(1 + r_MF_e),$$

where $r_i = -1/R_i$, the negative reciprocals of the roots of p.

This form was suggested by Scatchard as a generalization of the form $rF_e/(1+rF_e)$ which arises for a single-site simple binding reaction. Its interpretation is that there are M classes of binding sites, with N_i sites in the ith class, having an average equilibrium constant r_i. If the number of classes is not too large, this model can be used to describe $(F_e, v(F_e))$ data by curve-fitting to find the parameters N_1, N_2, ..., N_M, and r_1, r_2, ..., r_M, under the constraints $N_1 + N_2 + \ldots + N_M = N$, and $N_i > 0$, and $r_i > 0$. (Then the values of N_i can be judiciously adjusted to integer-values so as to preserve $N_1 + N_2 + \ldots + N_M = N$; alternatively, M can be taken to be N, the N_i's can be dispensed with (*i.e.* each $N_i = 1$), and the number of r_i's increased accordingly.) Constraints are usually needed to get a satisfactory fit.

Fletcher, et.al., have shown that, when $v(F_e)$ can be expressed in Scatchard's form, the following relations hold. Let L_1, L_2, ..., L_N be defined by: $L_j = r_i$ for $N_{i-1} + 1 \le j \le N_i$, with $N_0 = 0$.

Then: $v(F_e) = \sum_{i=1}^{n} L_iF_e/(1 + L_iF_e)$ and $B_j = \sum_{1\le i_1<i_2<\ldots i_j\le N} L_{i_1}L_{i_2}\ldots L_{i_j}$.

Note, B_j is the jth elementary symmetric function on L_1, ..., L_N. It is the coefficient of z^{N-j} in the polynomial $(z + L_1)(z + L_2)\cdots(z + L_N)$. The symmetric functions can be defined in MLAB by the following recursive function definitions.

```
*FUNCTION S(A,Z) = IF Z=N THEN SUM(I,A,Z,L[I]) ELSE \
       SUM(I,A,Z,L[I]*S(I+1,Z+1))
*FUNCTION B(J) = IF J<1 OR J>N THEN 1 ELSE S[1,N-J+1]
```

Another relationship which could be used is:

$B_j(i) = B_j(i-1) + L_iB_{j-1}(i-1)$, where $B_j(i)$ is the jth symmetric function on L_1, L_2, ..., L_i. Then $B_j = B_j(N)$.

Now, the original equilibrium constants may be obtained as: $K_i = B_i/B_{i-1}$ where $B_0 = 1$.

When our polynomial, $p(x)$, does not have N real roots, the Scatchard form is not equivalent to the Adair form. Rather a partial-fraction expansion yields a sum of rational terms with quadratic denominators. Sometimes, there is a "near-by" polynomial with N real roots, and if a Scatchard

form is fit to data, the resulting estimates of $K_1, \ldots, K_N$ can be useful, even though we are using an inexact model. One usually would check the obtained K_i-values in the Adair model in any event. Usually useful results can be had only when $N < 6$, unless highly-accurate data over a wide range of F_e-values is available.

The general analysis of multiple-site binding is quite complex. An excellent survey, including many practical tips, is: "The analysis of equlibrium binding data by the fitting of models" by John E. Fletcher, DCRT, N.I.H., May 1982.

8 Chemical Kinetics: Simple Enzyme Reactions

Let S be a "substrate" material which is converted by the addition of an enzyme, E, into a molecular complex C which dissociates to yield a product P. We have:

$$S + E \underset{K_2}{\overset{K_1}{\rightleftharpoons}} C \overset{K_3}{\to} E + P.$$

This is, of course, an idealization of a more complex schema, since, in fact, even when S and P have the same molecular weight, energy must be used to, at least momentarily, change and restore the conformation of the enzyme E. But for many situations, our simple model will suffice.

Let $S(t)$, $E(t)$, $C(t)$, and $P(t)$ be the amounts of S, E, C, and P respectively at time t. Then we have:

$$\begin{aligned} dC/dt(t) &= K_1S(t)E(t) - K_2C(t) - K_3C(t), \quad C(0) = 0, \\ dP/dt(t) &= K_3C(t), \quad P(0) = 0, \\ S(t) &= S_0 - C(t) - P(t), \quad \text{with} \quad S_0 = S(0), \\ E(t) &= E_0 - C(t), \quad \text{with} \quad E_0 = E(0). \end{aligned}$$

Typically the reaction $C \to P+E$ is much slower than the complex-formation reaction $S+E \rightleftharpoons C$. Thus after the initial rise of C within time t_0, we have $dC/dt \approx 0$, so that then

$$\begin{aligned} C(t) &\approx K_1S(t)(E_0 - C(t))/(K_2 + K_3), \quad \text{or} \\ C(t) &\approx E_0S(t)/[(K_2 + K_3)/K_1 + S(t)], \quad \text{for} \quad t > t_0. \end{aligned}$$

Thus, from the equation for dP/dt, we have the so-called Michaelis-Menten equation:

$$dP/dt(t) \approx K_3E_0S(t)/((K_2 + K_3)/K_1 + S(t)), \quad \text{for} \quad t > t_0.$$

Now the slope $dP/dt(t)$ is measurable by fitting a straight-line to a segment of the kinetic curve for $P(t)$ with $t_0 < t \ll t_e$, where t_e is the time such that $S(t_e) \approx 0$, e.g., when almost all the substrate material has been consumed. Note $S(t) \approx S_0$ for $t_0 < t \ll t_e$ if S_0 is large enough, so in this case one can fit the Michaelis-Menten equation to the single data point $(S_0, dP/dt)$ to try to obtain K_3 and $K_m = (K_2 + K_3)/K_1$, the so-called Michaelis-Menten constant.

The parameter K_3 may be independently resolved by observing that when S_0 is very large we have $S(t)/(K_m + S(t)) \approx 1$ for $t_0 < t \ll t_e$, so $dP/dt(t) \approx K_3E_0$. Thus the maximum rate of formation of P for a fixed E_0 value is K_3E_0. Let $K_3E_0 = V_m$. V_m can be determined by measuring the slope, $dP/dt(t)$, of the linear region of the kinetic curve obtained when S_0 is very large. Now, the Michaelis-Menten equation becomes:

$$dP/dt(t)/V_m \approx S(t)/(K_m + S(t)) \quad \text{for} \quad t_0 < t \ll t_e.$$

Note $dP/dt(t)/V_m$ is the relative rate of formation of P, i.e., the proportion of the maximum rate which is achieved. Hence, having measured V_m, one can then measure $dP/dt(t)$ for a relatively-large amount of S, and then obtain K_m from the Michaelis-Menten equation as $S_0V_m/(dP/dt(t))-S_0$.

The Michaelis-Menten constant,K_m, is the amount of substrate which will yield a product formation rate of $V_m/2$. It thus is the point at which the formation of product becomes increasingly sensitive to a decreasing amount of substrate. The activity of the enzyme for a given amount of substrate is determined directly as $dP/dt(t)$ computed from the Michaelis-Menten equation for a given K_m.

The Lineweaver-Burke form of the Michaelis-Menten equation is often used because of its linear form. It is:

$$1/(dP/dt(t)) \approx (K_m/V_m)(1/S(t)) + (1/V_m).$$

The Eadie and Dixon form is also often used. It is:

$$S(t)/(dP/dt(t)) \approx S(t)/V_m + K_m/V_m.$$

Actually, as noted above, the reaction $S + E \rightleftharpoons C \rightarrow P + E$ is a fiction. It is commonly used to approximate the situation:

$$S + E \underset{K_2}{\overset{K_1}{\rightleftharpoons}} C \underset{K_4}{\overset{K_3}{\rightleftharpoons}} D \underset{K_6}{\overset{K_5}{\rightleftharpoons}} P + F$$

$$U + F \rightleftharpoons B \rightleftharpoons A \rightleftharpoons V + E$$

where C is ES-complex, and D is EP-complex and E and F are co-factors. Typically K_4 is negligible, but as the amount of P increases, it may block an appreciable amount of enzyme if K_6 is not nearly zero, and this can render the Michaelis-Menten equation useless.

The following is an MLAB tutorial sequence for studying the Michaelis-Menten model relative to the simplified kinetic model.

First define the kinetic model by typing:

```
*FUNCTION C DIFF T(T) = K1*(S0-C-P)*(E0-C)-(K2+K3)*C
*FUNCTION P DIFF T(T) = K3*C
*INITIAL C(0)=0
*INITIAL P(0)=0
*S0 = 10; E0 = 1
*K1 = .2; K2 = .025; K3 = .025
```

Thus we have assumed the true situation is:

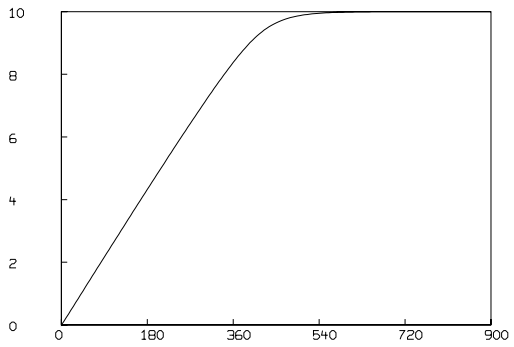
$$S + E \underset{.025}{\overset{.2}{\rightleftharpoons}} C \overset{.025}{\rightarrow} P + E$$

starting with 10 moles of substrate and 1 mole of enzyme. We may look at the kinetic behavior of this system over 900 seconds by typing:

```
*Q = INTEGRATE(P DIFF T, C DIFF T, 0:900:5)
*TYPE Q ROW 1:180:10
```

The first column of Q is time, t, the second is $P(t)$, the third is $dP/dt(t)$, the fourth is $C(t)$, and the fifth is $dC/dt(t)$. We can look at the graph of P vs. t by typing:

```
*DRAW Q COL 1:2, LINETYPE dashed
*VIEW
```

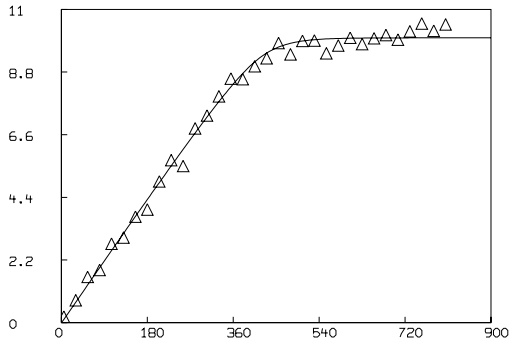


Now, let us generate some "laboratory data" about our reaction. We shall use the MLAB normal random number generator to generate normally distributed random numbers. Type:

```
*M = Q COL 1:2 ROW 2:162:5
*E = (NORMRAN ON 0^^NROWS(M))/4
*TYPE E
```

E is a vector of "normal" errors. Now type:

```
*M COL 2 = (M COL 2) + E
*DRAW M, POINTTYPE triangle LINETYPE none
*view
```

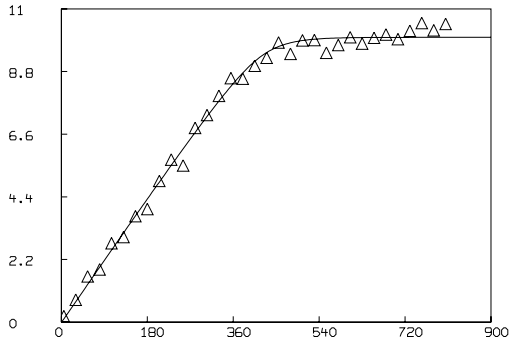


M is a matrix of P vs. t points "with error", as might have been measured in an actual laboratory situation. Now, let us "guess" K_1, K_2, and K_3 and try to determine them as functions of M. Type:

```
*K1 = .3; K2 = .01; K3 = .02
*CONSTRAINTS CX={K1>K2, K2>0, K3>0}
*METHOD = GEAR; ERRFAC = .002; MAXITER = 12
*FIT(K1,K2,K3), P to M, CONSTRAINTS CX
```

The control variables `METHOD` and `ERRFAC` are set based on prior experience; this problem is stiff and runs slowly! Our curve-fit "predicts" that K_1, K_2, and K_3 are 2.4295, .7975, and .025296 respectively, and resets them accordingly. Note K_1 and K_2 are not even close to .2 and .025, but K_3 is approximately correct. We may observe the graph of this fit by typing:

```
*Q1 = INTEGRATE(P DIFF T, C DIFF T, 0:900:5)
*DRAW Q1 COL 1:2 color green
*VIEW
```

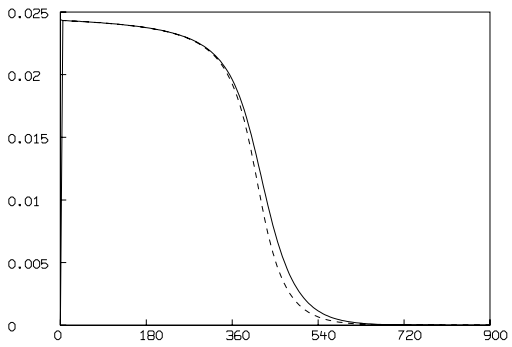


`Q1` is now a matrix of "kinetic concentration and velocity curves" for our reaction as determined by curve-fitting. Let us discard our picture by typing:

```
*DELETE W
```

Now, let us analyze the same reaction using the Michaelis-Menten model. Type:

```
*K1 = .2; K2 = .025; K3 = K2
*FUNCTION MM(S)=VM*S/(KM+S)
*KM = (K2+K3)/K1; VM = K3*E0
*DRAW Q COL (1,3)
*SM = S0-(Q COL 2)-(Q COL 4)
*DRAW (Q COL 1)&'(MM ON SM), LINETYPE dashed
*view
```

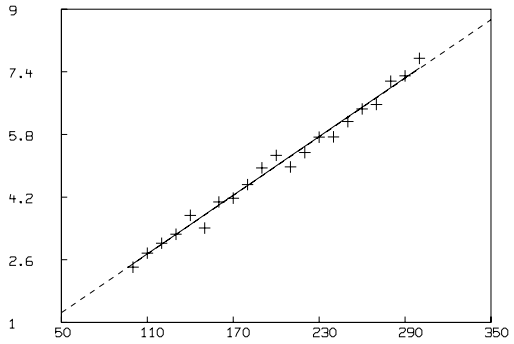


Recall Q is the matrix of true curves corresponding to $K_1 = .2$, $K_2 = .025$, and $K_3 = .025$. The matrix SM is computed as the amounts of S at time $t = 0, 5, 10, \ldots, 900$. The curves we see are the rate of change, dP/dt vs. t and its Michaelis-Menten approximation.

Now, let us generate two runs of "laboratory" data using the error vector E and obtain the two constants V_m and K_m. Type:

```
*DELETE W
*S0 = 500
*Z = INTEGRATE (P DIFF T, C DIFF T, 100:300:10) COL 1:2
*DRAW Z
```

```
*Z COL 2 = (Z COL 2)+(NORMRAN ON 0^^NROWS(Z))/4
*DRAW Z, LINETYPE none, POINTTYPE crosspt
*FUNCTION Y(T)=A*T+B
*CONSTRAINTS QS = {A > 0, B > 0}
*A = 1; B = 1
*FIT(A,B), Y TO Z, CONSTRAINTS QS
*VM =A
*DRAW POINTS (Y, 50:350!2), LINETYPE dashed
*VIEW
```



We have generated a "straight-line" segment of the P vs. t curve for $S_0 = 500$, drawn it, added some "noise", shown the simulated points obtained, fit a straight line to these points, set V_m as the slope of this line, and drawn the straight-line fit.

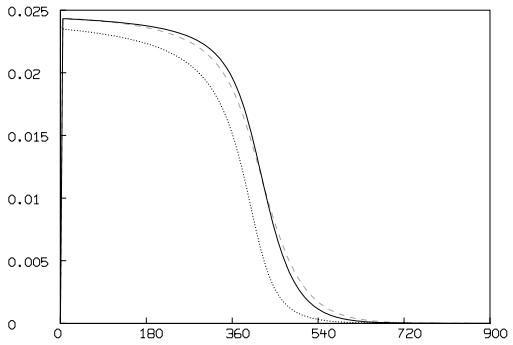
Now, we proceed in the same manner to do another "experiment" to help compute K_m. Type:

```
*DELETE W
*S0 = 10
*Z = INTEGRATE(P DIFF T, C DIFF T,100:300:10) COL 1:2
*Z COL 2 = (Z COL 2)+(NORMRAN ON 0^^NROWS(Z))/4
*FIT(A,B),Y TO Z, CONSTRAINTS QS
*KM = VM*S0/A-S0
*TYPE VM,KM
```

Now, V_m and K_m are computed. Let us look at the result. Type:

```
*DRAW Q COL (1,3)
*DRAW (Q COL 1)&'(MM ON SM) LINETYPE dotted color red
```

```
*DRAW Q1 COL (1,3) LINETYPE DASHED color green
*VIEW
```



Note the dP/dt curve predicted from the kinetic differential equation model is much better than the consistent underestimate predicted by the Michaelis-Menten model.

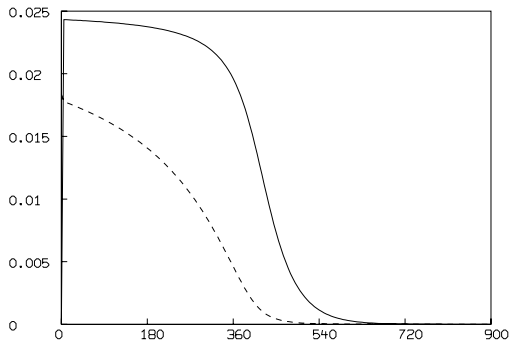
There is another approach to estimating the Michaelis-Menten constants, V_m and K_m, based on the intersections of various linear plots. This scheme is due to R. Eisenthal and A. Cornish-Bowden (Biochemistry Journal, Vol. 139, pp. 715:730). It is robust and, at the cost of more experiments, allows a confidence region for V_m and K_m to be obtained, without the usual restrictive assumptions. Unfortunately, it often produces poor extimates of V_m and K_m.

Given observations (S_{0i}, H_i) of substrate concentrations and corresponding product-formation velocities (obtained by linear-regression), we can construct lines defined by $V_m/H_i + K_m/S_{0i} = 1$, which may be plotted in K_m, V_m space. The line $\{\,(K_m, V_m) \mid V_m/H_i + K_m/S_{0i} = 1\,\}$ is the locus of all (K_m, V_m) pairs which could produce the observation (S_{0i}, H_i). Each of the (K_m, V_m) points obtained by the intersections of all pairs of these lines is an estimate of the "true" K_m, V_m values. The arithmetic median of the K_m-estimates is the Eisenthal-Cornish-Bowden estimate of K_m, and the arithmetic median of the V_m-estimates is the Eisenthal-Cornish-Bowden estimate of V_m.

We shall simulate ten experiments for $S_0 = 50 : 500 : 50$ and compute the Michaelis-Menten velocity curve based on K_m and V_m as estimated by the Eisenthal-Cornish-Bowden procedure.

```
*FUNCTION VMF(I,J) = (SV[I]-SV[J])/(SV[I]/VV[I]-SV[J]/VV[J])
*FUNCTION KMF(I,J) = (VV[J]-VV[I])/(VV[I]/SV[I]-VV[J]/SV[J])
*FOR I = 1:10 DO \
{S0 = 50*I;
  Z = INTEGRATE(P DIFF T, C DIFF T, 100:300:10) COL 1:2;
  Z COL 2  = (Z COL 2) + (NORMRAN ON 0^^NROWS(Z))/4;
```

```
  LSQRPT = 8;
  FIT(A,B),Y to Z, CONSTRAINTS QS;
  SV[I] = S0; VV[I] = A;
};
*D = 1:9^^'9
*D = COMPRESS((LIST(D)&'LIST(D'))*'LIST(D'<=D))
*D  COL 1 = (D COL 1) +1
*VM = MEDIAN(VMF ON D)
*KM =MEDIAN(KMF ON D)
*TYPE VM,KM
*DELETE W
*DRAW Q COL (1,3)
*DRAW (Q COL 1)&'(MM ON SM),LINETYPE DASHED
*VIEW
```



Overall the best approach to enzyme kinetics is to try to measure enough points on the kinetic curves of several species, so that direct curve-fitting using the appropriate differential equation model can permit the association and dissociation constants to be found. The Michaelis-Menten equation is used only due to the difficulty of obtaining data other than $P(t)$ for $t_0 < t < t_2$. Even then, concurrent use of the kinetic model is useful. An excellent source for mathematical models in enzyme kinetics is: Enzyme Kinetics by Kent Plowman, published by McGraw-Hill. Another is Enzyme Kinetics by Irwen Segal, published by Wiley.

9 Radioimmunoassay Analysis

Introduction

The study of how a ligand material, such as a hormone or antigen, binds to one or more kinds of molecular complexes, called sites, is of fundamental importance in biochemistry. Sites are often embedded in cell membranes, and the binding may serve to control the behavior of the cell itself. Typically we are interested in the number of distinct kinds of sites and their frequency of occurrence, and also the equilibrium constants for each ligand-site binding reaction which indicates the absolute strength of each such binding reaction.

For example, quantitative analysis of ligand-receptor binding can be easily performed using appropriate software such as the MLAB system discussed in this paper. MLAB is a computer program whose name is an acronym for "modeling laboratory"; it is an interactive system for mathematical modeling, originally developed at the National Institutes of Health. MLAB can fit multiple nonlinear models to data points. We are interested here in *designing* standard displacement assays. Typically such assays involve measuring the competition between radiolabelled and cold ligand in detergent-solubilized membrane preparations or on whole cells. Affinity constants and limit values of binding protein concentrations for single or multiple sites can be computed by fitting saturation curves in MLAB. Output can include Scatchard plots obtained by a suitable transformation.

One specific elaboration of the study of simple binding equilibrium states arises in radioimmunoassay analysis. Suppose P is a substance (a ligand or antigen) to be assayed which binds with a material Q (possibly, but not necessarily, an antibody) to form a complex B, and suppose further that H is a radioactive material which competes with P in binding to Q molecules. H is called the labeled or "hot" ligand. Thus, we have:

$$H + Q \underset{K_2}{\overset{K_1}{\rightleftharpoons}} L$$

$$P + Q \underset{K_4}{\overset{K_3}{\rightleftharpoons}} B.$$

Let P_0, H_0, and Q_0 be the initial amounts of P, H, and Q. If a number of experiments with different values of P_0, but with H_0 and Q_0 fixed, are done, and the bound labeled material L is measured, we obtain a set of data points (P_0, L) which, except for error, define a so-called dose-response curve. Generally, L falls as P_0 increases, because more P is competing with the hot ligand, so more B and less L are formed. Given such a dose-response curve, we can mix an unknown amount (possibly zero) of P (the dose) with H_0 moles of H and Q_0 moles of Q and measure L (the response) at equilibrium, and then read off from our dose-response curve how much P there must have been. This is an assay for the material P. The radioactivity is merely a device that allows us to measure the amount of L that is formed. Any other method that allows the amount of L to be determined can alternately be used.

Developing an assay entails providing a suitably-accurate dose-response curve. Thus, we wish to refine our dose-response curve by curve-fitting the experimentally-obtained data points (P_0, L).

Let the equilibrium constant $A_1 = K_1/K_2 = L/(HQ)$, and let the equilibrium constant $A_2 = K_3/K_4 = B/(PQ)$, with all species measured at equilibrium. Then $Q_0 = Q + B + L$, $H_0 = H + L$, and $P_0 = P + B$, so with some manipulation, we have:

$$L = A_1 H_0 Q/(1 + A_1 Q), \quad \text{and} \quad Q_0 = Q(1 + A_2 P_0/(1 + A_2 Q) + A_1 H_0/(1 + A_1 Q)).$$

We can define this model in MLAB as follows:

```
*FUNCTION L(P0) = LV(Q(P0))
*FUNCTION LV(QV) = A1*H0*QV/(1+A1*QV)
*FUNCTION Q(P0) = ROOT(Z,0,Q0,Z*(1+A1*H0/(1+A1*Z)+A2*P0/(1+A2*Z))-Q0)
```

These commands exemplify the MLAB FUNCTION statement, which is used to define a function or differential equation. Note that arguments of functions must be explicitly specified. Variables, such as `A1` and `A2`, which appear in the body of a function, but not in its argument list, are called *parameters*. Parameters must be assigned values before an associated function can be evaluated.

ROOT is an operator which is built-in in MLAB. `ROOT(Z,A,B,E)` is a value between `A` and `B` which, when taken as the value of the dummy variable, `Z`, makes the expression, `E`, which involves `Z`, equal to zero. Thus `ROOT(Z,A,B,E)` is a solution of $E(Z) = 0$. The model given above, involving a so-called *implicit* function, deserves careful study.

Now we can use *MLAB* to fit L to the (P_0, L)-data by adjusting A_1 and A_2 (and even H_0 and Q_0 if desired). Often we assume $A_1 = A_2$, but this is not necessary. This model demands that the data points be taken at equilibrium, and that the reaction is adequately described by the simple competitive scheme given above. In practice this model is very sensitive to the initial guesses, and constraints are often required. Usually H_0 is fixed at 1 and the L-data is expressed as a percent of total H. The P_0 data should then be in percent of total H units, as well.

The MLAB Mathematical Modeling System

MLAB has hundreds of useful functions, e.g., the discrete Fourier transform function `dft` and the parametric spline interpolation function `splinep`. One of the central components of the system is a curve fitting program which will adjust the parameters of a model function to minimize the weighted sum of the errors raised to a specified power. A repertoire of mathematical operators and functions, routines for solving differential equations, a collection of routines for onscreen drawing and for hardcopy plotting, and mechanisms for saving data between sessions provide a powerful and convenient environment for data manipulation, arithmetic calculations, and for building and testing models.

The user communicates with MLAB by typing commands which are executed at once or by providing a script to be executed. Should the user have questions, typing `HELP` will put the on-line system documentation at his disposal. The MLAB language is defined in the MLAB reference manual.

One of MLAB's main uses is to fit models to data. Curve-fitting is a useful analytical tool in many diverse disciplines. The basic notion is easily described. Given data, say various points in the plane (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), and a function $y = f(x)$ where f involves some parameters, say a and b, as for example $f(x) = ax^b + 1$, we may wish to calculate values for the parameters a and b so that the function f well-predicts the observed data, that is, so that $f(x_i) = y_i$ for $1 \leq i \leq n$. In this case, we say we have fit the model f to the data by estimating the parameters a and b. The end result is merely the values obtained for the initially unknown parameters. The same principles apply in higher dimensions with arbitrarily many parameters. MLAB can simultaneously fit multiple non-linear model functions, some or all of which may be implicit functions, or may even be defined by a system of differential equations.

The curve-fitting and graphics display facilities of MLAB make it an ideal tool for the estimation of equilibrium constants from data, which typically consists of observed amounts of ligand bound for various specified amounts of ligand provided for binding.

Dose-Response Curve Computation in MLAB

Here is an example showing the calculation of the dose-response curve for certain data which is read-in-below.

To begin we introduce the appropriate constraints (which should always be used for this particular model), guess the equilibrium constants A_1 and A_2, and then estimate them, as follows.

```
* CONSTRAINTS N ={A1>0,A2>0}
```

The `CONSTRAINTS` statement permits the user to specify successive linear inequalities or equations involving the parameters (or potential parameters). Now we may specify values for the parameters, guessing when necessary.

```
A1 = 1; A2 = 2; H0 = 2; Q0 = 4
```

Here the `ASSIGNMENT` statement is exemplified. In this case all the above assignments are assigning values to scalar variables, but the `ASSIGNMENT` statement is used to assign values to matrices as well. This can be seen in the next `ASSIGNMENT` statement which defines a matrix, M.

```
M = READ(assay,21,2)
```

The READ operator takes optional array size arguments; in this case a 21 row by 2 column matrix is specified, and reads in numbers from the specified file to construct an appropriately-dimensioned matrix as the result. This matrix is then, in this case, assigned to M.

We have established a model function, L, and entered data, M. We expect that $L(M[i,1]) \approx M[i,2]$ would hold, if only the parameters A_1 and A_2 were set to their "correct" values. The following FIT statement requests MLAB to estimate A_1 and A_2 by assigning them values which minimize the sum-of-squares objective function $S(A_1, A_2) = \sum_{i=1}^{8} (L(M[i,1]) - M[i,2])^2$.

```
FIT(A1,A2), L TO M WITH WEIGHT EWT(M), CONSTRAINTS N

final parameter values
      value                error                 dependency    parameter
   6.1567727039        0.9593487946          0.7470305183   A1
  60.5719353014       12.2219429924          0.7470305183   A2
6 iterations
CONVERGED
best weighted sum of squares = 9.870322e+01
weighted root mean square error = 2.279234e+00
weighted deviation fraction = 5.480005e-02
R squared = 9.873349e-01
no active constraints
```

The behavior of the FIT statement depends upon the supplied constraints as well as upon the MLAB control variables: MAXITER, the maximum number of iterations and TOLSOS, the requested convergence factor.

MLAB uses a carefully-tuned version of the Marquardt-Levenberg magnified-diagonal algorithm which is, in turn, a form of the Gauss-Newton procedure for minimizing a function which is in the form of a sum-of-squares. This process estimates the value of the parameter vector ($b = (A_1, A_2)$ in our case) by successive approximations $b^{(0)}$, $b^{(1)}$, ..., $b^{(n)}$, where $b^{(0)}$ is the vector of initial guesses for A_1 and A_2, and $b^{(j+1)} = b^{(j)} + \beta^{(j)}$, where

$$\begin{aligned}
\beta^{(j)} &= (X'V^{-1}X + \varepsilon G)^{-1}X'V^{-1}(y - (f(x_1; b^{(j)}), \ldots, f(x_8; b^{(j)}))'), \quad \text{with} \\
X_{st} &= \partial f(x_s; b^{(j)})/\partial b_t \quad \text{and} \\
G_{st} &= \text{if } s = t \text{ then } (X'V^{-1}X)_{st} \text{ else } 0 \quad \text{and} \\
x_s &= M[s,1] \quad \text{for} \quad 1 \le s \le 8, \quad \text{and} \\
y &= (M[1,2], \ldots, M[8,2])',
\end{aligned}$$

where V is the estimated covariance matrix of the observations. In our example, $V = I$, the identity matrix. In general V is determined from weight-values supplied by the user.

An iteration consists of computing $b^{(j+1)}$ from $b^{(j)}$. Note that this requires the partial derivatives of the model function with respect to the parameters evaluated at $b^{(j)}$, since these values form the matrix X. In MLAB, these derivatives are automatically computed symbolically and evaluated to form X. The convenience thus obtained is considerable and the parameter estimation process is provided with more accurate derivative values. For example the derivative of L with respect to A_1 can be explicitly displayed in MLAB as follows.

```
* TYPE L'A1, LV'A1, Q'A1
FUNCTION L'A1(P0) = LV'A1(Q(P0))+LV'QV(Q(P0))*Q'A1(P0)
FUNCTION LV'A1(QV) = (H0*QV*(1+A1*QV)-QV*A1*H0*QV)/((1+A1*QV)*(1+A1*QV))
FUNCTION Q'A1(P0) = EVAL(Z,ROOT(Z,0,Q0,
  Z*(1+(A1*H0)/(1+A1*Z)+(A2*P0)/(1+A2*Z))-Q0),
  -(((H0*(1+A1*Z)-Z*A1*H0)/((1+A1*Z)*(1+A1*Z)))*Z)/
  ((1+(A1*H0)/(1+A1*Z)+(A2*P0)/(1+A2*Z))-((A2*A2*P0)/
  ((1+A2*Z)*(1+A2*Z))+(A1*A1*H0)/((1+A1*Z)*(1+A1*Z)))*Z))
```

Indeed derivatives are full-fledged members of the class of functions and can be used in graphics or curve-fitting in MLAB just as can any other user-defined function.

A sub-iteration consists of computing $b^{(j+1)}$ with a particular value of ε which specifies the amount of diagonal magnification. At each iteration, the value ε starts at 10^{-9} and is increased until the corresponding value of $b^{(j+1)}$ results in a smaller sum-of-squares value, whereupon this vector is taken to be the final $b^{(j+1)}$ iterate.

The parameter estimation process stops when the limit of the number of iterations is reached or, more usually, when the decrease in the sum-of-squares value between successive iterations is less than a specified fractional amount determined by the user-specified convergence factor in `TOLSOS`. For `TOLSOS` = .001, the sum-of-squares must change by less than .1 percent for the curve-fitting process to stop based on this criterion.

When the estimation process does stop, the parameters are reset to their computed estimates, and they and their estimated standard deviations are typed out. Associated values called dependency values, which lie between 0 and 1, are also typed out. It suffices here to remark that large dependency values above .99 usually indicate a non-unique solution; that is, other parameter estimates exist which would provide a nearly equally-small sum-of-squares.

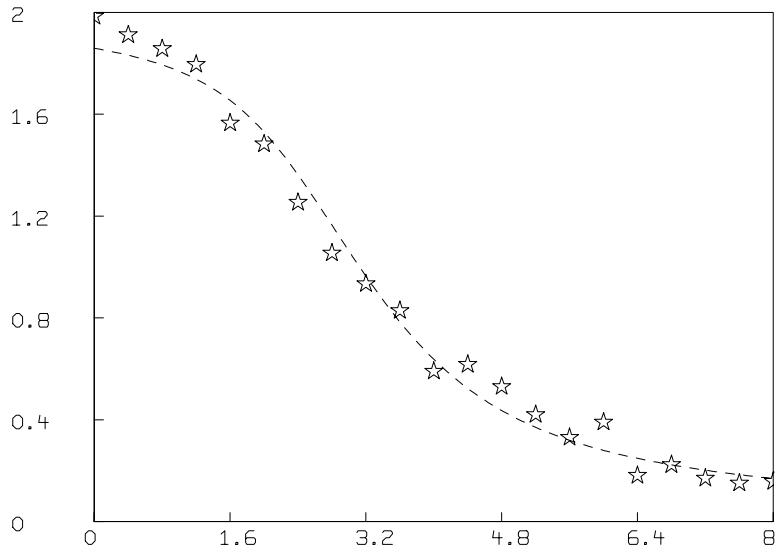
MLAB also types-out the root-mean-square error which is the estimate of the standard deviation in each observation, given that they are identically distributed. This quantity should roughly equal the experimental error in the data. There are, of course, many caveats and restrictions which must hold to insure the validity of the supporting statistics provided.

The material typed out above shows that the vector of parameters (A_1, A_2) has been estimated to be $(6.15677 \pm 0.95935, 60.5719 \pm 12.2219)$, with reasonably small dependency values, and with an

RMS error of about .987, which should be comparable with the experimental error in our data, M. The sum-of-squares was reduced to 98.7 at the final parameter values.

In order to visually see how our model with its parameters set to their best-estimated values corresponds to the data, we may draw a graph of the data points and the model function. Although we are drawing only the simplest and most direct kind of picture here, it should be noted that MLAB provides facilities for many types of point-symbols and types of lines, axes with arbitrarily-placed numeric labels in various formats, titles in the form of text strings in arbitrary sizes and various fonts with subscripts and superscripts, color, and a number of other special features. It is quite possible to prepare more or less elaborate publication-quality graphs with a modest amount of effort. Indeed this is one of MLAB's most-used facilities. The desired graph can be constructed as follows.

```
* DRAW M, LINETYPE NONE,POINTTYPE STAR
* DRAW C2 = POINTS(L,0:8:.2)
* VIEW
```



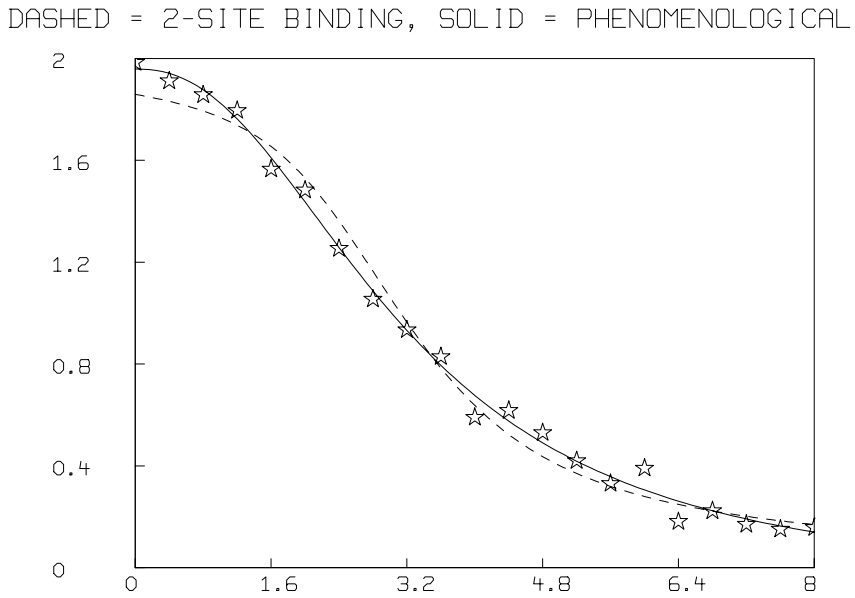
The first `DRAW` command above plots the data points, while the second `DRAW` command constructs a curve called `C2`, which is a graph consisting of solid straight-lines connecting the points which are the rows of the 2-column matrix which is the value of the expression `POINTS(L,0:8:.2)`. This matrix has the values 0 through 8 in steps of .2 in its first column and corresponding values of the

function L evaluated at 0 through 8 in steps of .2 in the second column. The `POINTS` operator is very useful for graphing functions. Both these curves are drawn in the default MLAB 2D-window called `W` (since no other window is specified) which has predefined labeled axes already present. The picture finally appears when its display is requested with a `VIEW` statement and a plot can be obtained if desired using the `PLOT` statement.

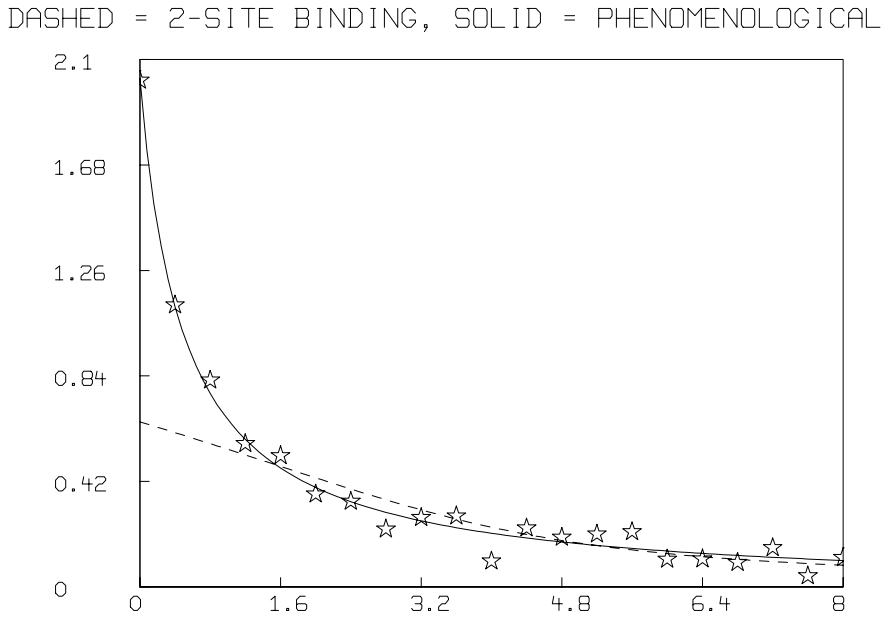
Often a "phenomenological" model for dose-response curves is preferred. Experience has shown that the function $L(P_0) = (a-d)/(1+(P_0/c)^b)+d$, where b is the "slope-factor" approximately equal to 1, a is the zero dose response, d is the infinite dose response, and c is the dose for a 50 percent response, often fits well. In either case, the fitting should be done with weights (usually the MLAB estimated weights, `ewt(m)`, where M is the (P_0, L) data matrix, suffices).

Now let us fit the phenomenological model and draw the comparison below.

```
FCT EL(p) = (a-d)/(1+(p/c)^b)+d

a = 2; b = 2.5; c = 3.0; d = 0;

FIT(a,b,c,d), EL TO M WITH WEIGHT EWT(M)

final parameter values
      value               error                dependency    parameter
   1.9589377175        0.0253123625          0.6168286269   a
    2.300705405        0.1574645119          0.8828919509   b
    3.176751857        0.0979352743          0.8541792257   c
  -0.0773479916        0.0539555619          0.9449998881   d
3 iterations
CONVERGED
best weighted sum of squares = 2.767049e+01
weighted root mean square error = 1.275804e+00
weighted deviation fraction = 2.149476e-02
R squared = 9.958668e-01

DRAW M LT NONE, PT STAR
DRAW POINTS(L,0:8!101) LT DASHED
DRAW POINTS(EL,0:8!101)
TOP TITLE "DASHED = 2-SITE BINDING, SOLID = PHENOMENOLOGICAL"
VIEW
```



Both the direct second-order competitive binding model and the phenomenological model fit well here. In other cases, however, the second-order competitive binding model is inferior to the phenomenological model as is demonstrated in the example shown in fig. 3. The reasons for this include the likelihood that the actual binding reactions are more complex than the simple competitive binding scheme postulated above.

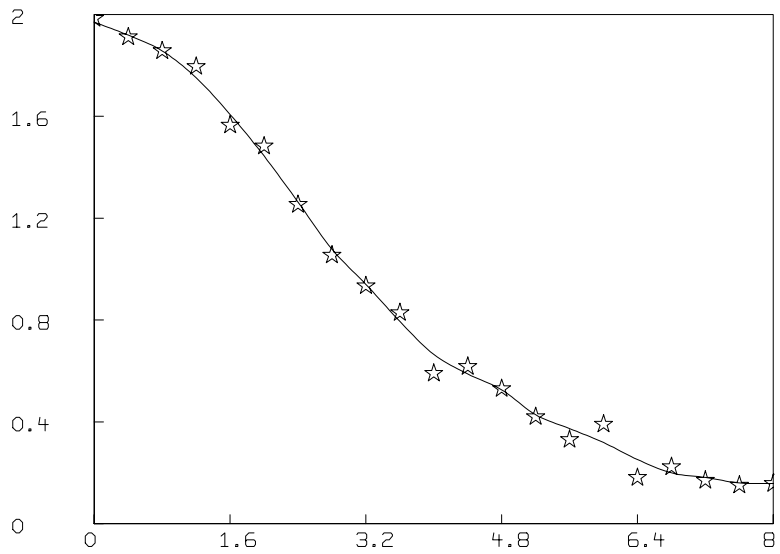


Once the dose-response function L is determined, the assay value for P_0, given an observed amount, L_1, of the material L, is just $V(L_1)$, where in MLAB, we have:

```
*FUNCTION V(L1) = ROOT(X,0,PMAX, L(X)-L1).
```

Finally, as another approach, the function L can be determined entirely empirically from the data matrix M as follows, where the L-unit is fraction of total H.

```
*G = SMOOTH(M,3)
*G = INTERPOLATE(G,0:8:.04)
*DRAW M LT NONE PT STAR
*DRAW G
*VIEW
```



Any of a number of non-parametric regresion schemes can be used; here we have used a weighted moving average smoothing function.

A confidence interval for P_0 can be established, at least approximately, by following the scheme for so-called calibrated estimation. A useful reference in this regard, as well as for other purposes, is: "The Application of Robust Calibration to Radioimmunoassay" by James Tiede and Marcello Pagano in Biometrics, Vol. 35, pp. 567:574, Sept. 1979.

10 Infectious Disease Dynamics

Suppose we have an infectious disease agent which infects a susceptible individual and remains untransmittable by that individual for a latent time period q. Such an infected individual will be called a latent individual. After q days, a latent individual becomes an infective individual and can transmit the disease to susceptibles. An infective individual also becomes vulnerable to an increased death risk. The death rate for non-infectives is u percent per day, but for infectives, the death rate is $(u + a)$ percent per day. Finally, after a time period of p days, an infective who survives becomes an immune individual, who cannot transmit the disease, and whose risk of death drops back to the population death rate u.

Suppose also that the birth rate is b percent per day. We may define the following functions.

$$\begin{aligned} S(t) &= \text{the number of suceptible individuals at time } t. \\ L(t) &= \text{the number of latent infected individuals at time } t. \\ F(t) &= \text{the number of infective individuals at time } t. \\ I(t) &= \text{the number of immune individuals at time } t. \end{aligned}$$

Now, of the individuals, say $D(t)$ of them, who enter the latent state at time t, $D(t)(1 - e^{-uq})$ will die during the time period q, so that $D(t)e^{-uq}$ individuals will enter the infective state at time $t + q$. Similarly, of the individuals, say $E(t)$ of them,who enter the infective state at time t, $E(t)e^{-(u+a)p}$ will survive to enter the immune state at time $t + p$.

Let c be the mean number of susceptible individuals contacted by an individual per day; c might be about .2. Let h be the probability that a contact of a suceptible with an infective results in transmission of the disease. The probability that a contact at time t is with an infective is, of course, $F/(S + L + F + I)$.

To begin, we will "infuse" our population with F0 newly-arriving infectives over one day, as indicated by the infusion rate function, R, given below. Let

$$\begin{aligned} R(t) &= \text{if } 1 > t \text{ and } t > 0 \text{ then } F0 \text{ else } 0, \\ D(t) &= \text{if } t < 0 \text{ then } 0 \text{ else } chF(t)/(S(t) + L(t) + F(t) + I(t))S(t), \\ E(t) &= D(t - q)e^{-uq}, \quad \text{and} \\ G(t) &= (E(t - p) + R(t - p))e^{-(u+a)p}. \end{aligned}$$

Thus we have the following differential equations:

$$\begin{aligned} dS/dt &= b(S+L+F+I) - uS - D(t), \\ dL/dt &= D(t) - uL - E(t), \\ dF/dt &= R(t) + E(t) - (u+a)F - G(t), \quad \text{and} \\ dI/dt &= G(t) - uI, \end{aligned}$$

where $S(0) = S0$, $L(0) = 0$, $F(0) = 0$, and $I(0) = 0$.

The latency and infective periods lead to delay terms in the differential equations. MLAB deals with delay by linearly interpolating in the table of previous results. Thus, the time interval between requested answer points should be less than $\min(p/2, q/2)$ to allow the backward interpolation to apply.

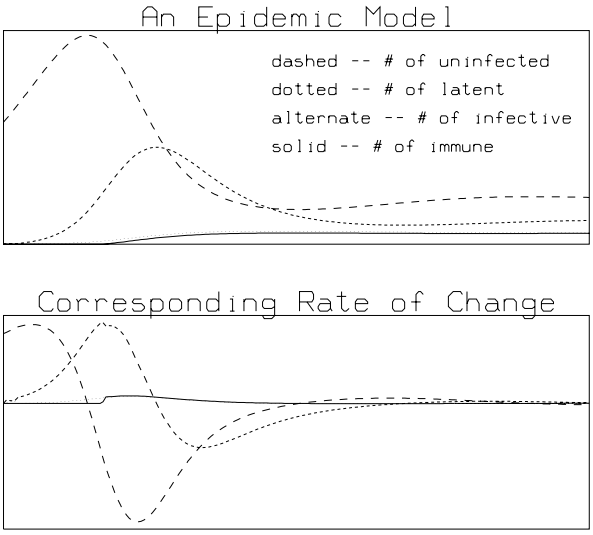
It is interesting to experiment with values for c, h, a, p, q, and $F0$ to see under what circumstances a disease will die out, oscillate, or infect the entire population. One can replace the factor $chF/(S+L+F+I)$ with a more abstract constant, k, when the disease is not transmitted by contact alone. Also note that epidemiological data can be fit with these equations and the values of a, q, p, h and/or $F0$ can thus be estimated.

For more material on such models, see: "The mathematical theory of infectious diseases and its applications", by N.T.J.Bailey, published in 1975 by Charles Griffin in London.

An example using MLAB which results in a new equilibrium being obtained from an initial perturbation is presented below.

```
"define the epidemic differential equations"
fct S't(t) = b * (S(t) + L(t) + F(t) + I(t)) - u * S(t) - D(t)
fct L't(t) = D(t) - u * L(t) - E(t)
fct F't(t) = R(t) + E(t) - (u+a) * F(t) - G(t)
fct I't(t) = G(t) - u * I(t)

"define ancillary functions (note the delay)"
fct R(t) = if (t < t0 and t > 0) then f0 else 0
fct D(t) = if t < 0 then 0 else c*h*S(t)*F(t)/(S(t) + L(t) + F(t) + I(t))
fct E(t) = D(t-q) * exp(-u*q)
fct G(t) = (E(t-p) + R(t-p)) * exp(-(u+a)*p)

"define system constants"
f0 = 0.3; s0 = 200; c = 0.5; h = 0.5; p = 30; q = 4;
u = 0.0072; a = 0.11; b = 0.035; t0 = 40;
```

```
"define the initial values"
init S(0) = s0; init L(0) = 0;
init F(0) = 0; init I(0) = 0
"integrate the ODEs"
M = integrate(S't, L't, F't, I't, 0:200)

frame 0 to 1, .5 to 1 in w1

"graph the number of susceptibles, S(t)"
draw M col (1,2), linetype dashed, in w1

"graph the number of latents, L(t)"
draw M col (1,4), linetype dotted, in w1

"graph the number of infectives, F(t)"
draw M col (1,6), linetype alternate, in w1

"graph the number of immunes, I(t)"
draw M col (1, 8) linetype solid, in w1

top title "An Epidemic Model", size 0.025, in w1
title "dashed -- # of uninfected", size 0.015, at (.47,.75), in w1
title "dotted -- # of latent", size 0.015, at (.47,.65), in w1
title "alternate -- # of infective", size 0.015, at (.47,.55), in w1
title "solid -- # of immune", size 0.015, at (.47,.45), in w1

frame 0 to 1, 0 to .5 in w2

"graph the derivative of the number of susceptibles, S't(t)"
draw M col (1,3), linetype dashed, in w2

"graph the derivative of the number of latents, L't(t)"
draw M col (1,5), linetype dotted, in w2

"graph the derivative of the number of infectives, F't(t)"
draw M col (1,7), linetype alternate, in w2

"graph the derivative of the number of immunes, I't(t)"
draw M col (1, 9), linetype solid, in w2

top title "Corresponding Rate of Change", size 0.025, in w2
view
```


An Epidemic Model
dashed -- # of uninfected
dotted -- # of latent
alternate -- # of infective
solid -- # of immune
Corresponding Rate of Change

11 The Hodgkin-Huxley Nerve Axon Model

The prevailing model of a nerve axon membrane is a pair of theories concerning the nature of the axon membrane with respect to active ("pumping") and passive (diffusion) steady-state transport of various ions across the membrane and with respect to time-dependent "gate" opening and closing which controls the active passage of ions through such "open gates".

It is postulated that the membrane in a given state has a certain permeability for each given ion, and that this permeability is determined by the electrochemical potential across the membrane.

The permeability, P_C, of a membrane for a particular chemical species, C, is a measure of the ease of diffusion of C across the membrane in the presence of a concentration difference on either side of the membrane. In particular, upon entering the membrane, one C-molecule will travel, on the average, $(P_C aF/(\beta RT))E_m$ cm/sec transversely across the membrane, where $P_C = u\beta RT/(aF)$, and E_m, u, β, R, T, a, and F are defined as follows.

$E_m =$ the electric potential difference across the membrane, measured in volts.

$u =$ the electric mobility of C in the membrane $\approx .001\,(\text{cm/sec})/(\text{volt/cm})$ for sodium or potassium. This is the average velocity of a C-molecule due to an electric potential difference of one volt/cm (e.g., for a 100 Angstrom membrane, the reference potential difference is $1\,\mu$volt between the inside and the outside). As the potential difference, E_m, changes, the mobility of C changes. For a -60 mv potential difference across a 100 Angstrom membrane, $u \approx 600\,(\text{cm/sec})/(\text{volt/cm})$.

$\beta =$ the partition coefficient between the membrane and solution $\approx .003$. β is the ratio of the concentration of C in the solution to the concentration of C in the membrane itself in steady-state. β is a measure of the relative affinity of the membrane substance for C-molecules with respect to the neighboring solution.

$a =$ the thickness of the membrane, about 100 Angstroms.

$F =$ Faraday's constant, 96486.7 coulomb/mole. F is Avogadro's number times the charge in coloumbs of a proton.

$R =$ the gas constant, 8.31434 joule/(degree/mole). R is Avogadro's number times Boltzmann's constant.

$T =$ temperature in degrees Kelvin.

For a given ionic species, C, which is placed in unequal concentrations on either side of an idealized membrane, there will be a diffusion of C in the direction of lower concentration. This diffusion changes any existing electric potential field, since charge is being redistributed. Other species

may be diffusing as well, but regardless of the total complexity, the ionic current of C ions will be zero just when the electric potential difference across the membrane is sufficient to induce a compensating counter drift of C-ions in the direction of lower concentration. The potential difference of the potential in the "in"-compartment minus the potential difference in the "out"-compartment which leads to a net C-ion current of zero is given by the so-called Nernst potential difference, $E_C = ZRT/(ZF)\log(C_{out}/C_{in})$ volts, where Z is the valence of C ($+1$ for sodium and -1 for chlorine) and C_{in} and C_{out} are the concentrations of C in the inside and outside compartments. The actual electrical potential across the membrane is a function of the relative concentrations of each species present, and it is this total potential which determines how each ion would move with respect to its diffusion tendency. The total potential can be computed using the Goldman equation given below. Potential is defined so that positive charge flows towards points of lower potential, while negative charge flows towards points of higher potential. The membrane potential difference at a point is defined as the potential at the point minus the potential at a reference point in the outside solution.

For any given initial concentrations of various species and given time-course changes of membrane permeabilities for these species, there will be an associated change in total potential and a varying ionic current flow which, in principle, we can determine with appropriate equations. (We ignore the added complication of concentrations changing due to the diffusion of the solute, or of hydrostatic pressure gradients arising when solute diffusion is not possible.)

A full discussion of nerve membranes is given in "Ionic channels of excitable membranes" by Bertil Hille (Sinauer Associates, Sunderland, Mass, 1984).

For a giant squid nerve axon it has been experimentally determined that the chemical species of interest are potassium ions, sodium ions, chloride ions, and various other ions of lesser importance. Ions of all these less-significant species are collectively lumped together and called "leakage" ions. Each of the various leakage ions have differing permeabilities; the result is that the permeability for leakage ions is an "average" which compensates for the various differing permeabilities actually involved. Moreover, some of the leakage ions are negative and some are positive. We shall assume that our "average" leakage ion is negative, and that the strength of an ionic current of leakage ions is diminished to compensate for this fiction. The resting axon membrane of an "average" squid has permeabilities for these species which are given in the table below with the resting concentrations in mmoles/liter.

	inside	outside	resting permeability	potential difference
$K(+)$	400	10	6×10^{-6}	$-72\,\mathrm{mV}$
$Na(+)$	50	460	8×10^{-9}	$55\,\mathrm{mV}$
leakage$(-)$	45	540		$-50\,\mathrm{mV}$

These values lead to a zero net ionic current in the resting state. The total resting state potential difference, E_r, based on the Goldman equation works out to $-60\,\mathrm{mv}$. The Goldman equation in

this case is:

$$E_r = log((P_K K_{out} + P_{Na} Na_{out} + P_L L_{in})/(P_K K_{in} + P_{Na} Na_{in} + P_L L_{out}))RT/F$$

volts, where L denotes "leakage" ions. This is valid only when there is zero net ionic current.

The above state cannot be indefinitely maintained by a passive membrane which allows each species to pass at a non-zero rate. Even though the net ionic current is zero, there are non-zero sodium, potassium, and leakage currents in the resting state. An additional mechanism is postulated, namely a "sodium pump" which actively transports sodium ions from the inside to the outside to balance the resting state inward-diffusion, and a "potassium pump" which transports potassium ions from the outside to the inside to balance the resting state outward-diffusion.

There is certainly such a mechanism present in the axon membrane but its precise chemical nature is not yet known. The energy to drive the pumps depends upon the conversion of ATP to ADP. It is not known how the two pumps are interrelated. It may be they are entirely independent, or they may be coupled in some way. There is probably not a single charge-balanced pump which brings one K ion inside for each sodium ion ejected outside, since the pump is required to compensate for unequal K and Na fluxes. A discussion of current research results about the pumping mechanism can be found in "The $(Na^+ + K^+)$ activated enzyme system and its relationship to transport of Sodium and Potassium" by Jens Chr. Skou, pp. 401:434 of Quarterly Reviews of Biophysics, Vol. 7, No. 3, July 1974.

Conductance is merely the reciprocal of resistance. It is a measure of the ability of a material to carry a particular ionic or electric current. Conductance is measured in Ω^{-1}/cm^2 units. The potassium conductance of one cm^2 of axon membrane is denoted by g_K, The sodium conductance of one cm^2 of axon membrane is denoted by g_{Na}, and the leakage ion conductance of one cm^2 of axon membrane is denoted by g_L. *These values are, in general, functions of the potential difference history of the membrane.*

Ohm's law holds for ionic current flow, but since there is zero current for the ionic species, C, when the voltage is exactly the Nernst potential, E_C, for C, we have the C-ion current is given by $I_C = g_C(E_m - E_C)$, where E_m is the total membrane potential. Ohm's law applies here by convention, since g_C is determined in any state, just so $I_C = g_C(E_m - E_C)$ is true.

The conductance, g_C, and the permeability, P_C, for a given ionic species, C, with charge Z, are related as follows.

$$P_C = g_C RT(E_m - E_C)(exp(ZE_m F/(RT)) - 1)/((C_{out} - C_{in} exp(ZE_m F/(RT)))/Z/(F^2 E_m)),$$

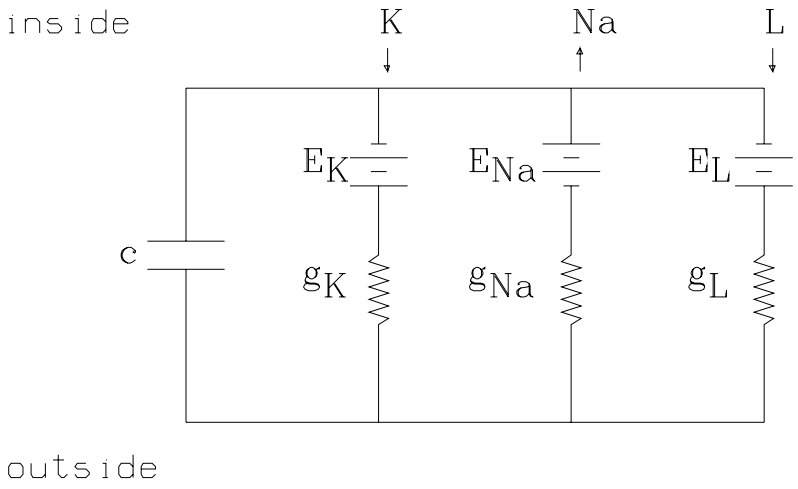
for $C_{out} - C_{in} \exp(ZFE_m/(RT)) \neq 0$, where E_m, E_C, F, R, T, C_{out}, and C_{in} are as defined above. E_m and E_r are here measured in volts.

The Hodgkin-Huxley model of an axon membrane is a mathematical description of the potential difference or voltage across the membrane which changes as a function of time in response to various

perturbations of potential established with an associated applied current. It was presented in the Journal of Physiology, Vol. 117, pp. 500:544, 1952.

This description is in the form of an analogous circuit which in turn can be characterized by a set of non-linear differential equations. These equations contain an empirically adequate description of the feedback system which changes the conductances (or equivalently, the permeabilities) of the membrance as a function of time and potential difference.

The circuit merely contains the three main ionic current circuits in parallel together with the observed membrane capacitance.



Empirically we have the emf values $E_K \approx -72\,\text{mV}$, $E_{Na} \approx 55\,\text{mV}$, and $E_L \approx -50\,\text{mV}$.

The values g_K, g_{Na}, and g_L are the conductances of the membrance for potassium, sodium, and leakage ions respectively. g_K and g_{Na} are functions of time and voltage.

The sign convention for current is such that a net flow of positive ions from inside to outside is a positive current. Thus a net flow of positive ions from outside to inside, or a net flow of negative ions from inside to outside is a negative current. Here current is measured in μAmperes per cm^2, and E_m, E_K, E_{Na}, and E_L are measured in mvolts.

The values of E_K, E_{Na}, and E_L given above imply that independently in the resting state, we would have a nearly-zero sodium current, a nearly-zero potassium current, and a zero leakage ion current. The sodium and potassium currents just balance the corresponding pump currents.

These circuits are connected in parallel however, and thus the total driving potential applied to sodium ions is $E_m - E_{Na}$, the total driving potential applied to potassium ions is $E_m - E_K$, and

the total driving potential applied to leakage ions is $E_m - E_L$. According to Kirchoff's law, the currents into any point must sum to zero, so these three ionic currents must sum to 0. From Ohm's law as discussed above, the sodium current, I_{Na}, is $g_{Na}(E_m - E_{Na})$, the potassium current, I_K, is $g_K(E_m - E_K)$, and the leakage ion current, I_L, is $g_L(E_m - E_L)$. But, in the resting state, $I_K + I_{Na} + I_L = 0$, so $E_m = (g_{Na}E_{Na} + g_K E_K + g_L E_L)/(g_{Na} + g_K + g_L)$. In the resting state, E_m has been determined experimentally to be about $-60\,\mathrm{mV}$, g_{Na} has been determined to be about .0106092 and gK is about .3666445, so we may define

$$g_L = (g_{Na}(E_{Na} + 60) + g_K(E_K + 60))/(-60 - E_L) \approx .3179676.$$

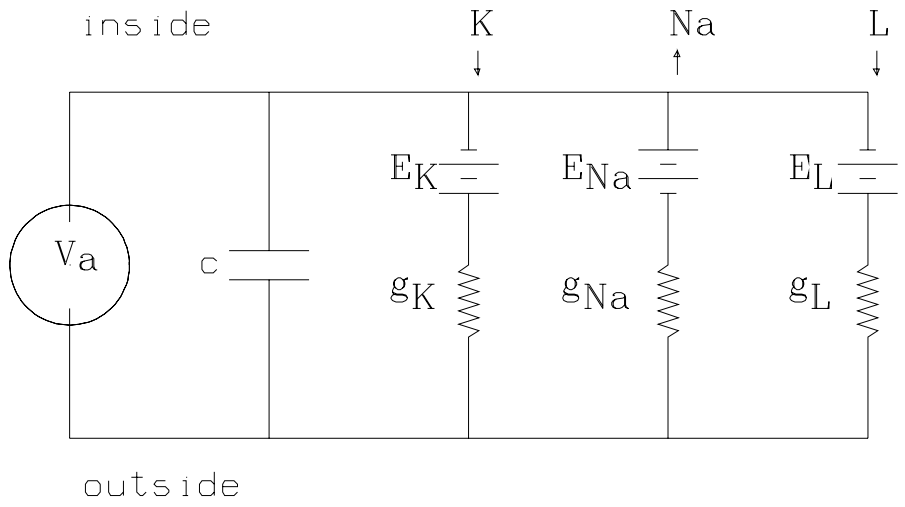
(Usually, g_L is set to .3, and E_L is computed to yield a zero current, but we follow the opposite convention.)

Thus, for our "average" axon, the total sodium driving potential is $-115\,\mathrm{mV}$ and we have a negative sodium current, the total potassium driving potential is $12\,\mathrm{mV}$ and we have a positive potassium current, and the total leakage ion driving potential is $-10\,\mathrm{mV}$ and we have a negative leakage ion current.

The sodium current is approximately $-1.22\,\mathrm{mA}$, the potassium current is about $4.4\,\mathrm{mA}$, and the leakage ion current is about $-3.18\,\mathrm{mA}$, and, of course, the sum is zero.

Of course, as these currents flow, the concentration-based emf's E_K, E_{Na}, and E_L change; however the pumping devices postulated before are assumed to maintain a $1.22\,\mu\mathrm{A}$ sodium current and a $-4.4\,\mu\mathrm{A}$ potassium current which forces leakage ions to remain in place in order to keep charge balanced in both the inside and the outside compartments. Thus in the resting state there is zero net ionic current across the membrane, and zero individual ionic currents as well. The Hodgkin-Huxley model does not include the pump current, so although there is zero net current in the resting state, the individual ionic currents are presented as computed above.

We may apply a time-varying current, I_a, across the membrane as a stimulus. This can be achieved by the feedback-controlled circuit shown below which applies whatever time-varying voltage, V_a, across the membrane is needed to achieve the current $I_a(t)$ at time t. Here time is measured in milliseconds.



Thus, in general, we have $I_a = I_K + I_{Na} + I_L + I_c$, where I_c is the current across the membrane due to the capacitance c. Now $I_c = c(dE_m/dt)$, where $c \approx 1\,\mu\text{farad/cm}^2$. Thus, we obtain

$$c(dE_m/dt) = I_a - g_K(E_m - E_K) - g_{Na}(E_m - E_{Na}) - g_L(E_m - E_L).$$

The Hodgkin-Huxley model assumes that the emf's E_K, E_{Na}, and E_L, and the conductance g_L, are constant, but that the conductances g_K and g_{Na} are not constant, but rather are functions of the membrane potential difference history.

A change in g_K corresponds to a change in the permeability of the membrane for potassium and similarly for g_{Na}. The physical idea stated below underlying the Hodgkin-Huxley specification of how g_K and g_{Na} change is, by itself, incorrect, (see Armstrong, *C.*, Physiological Reviews, Vol. 61, pp. 644:683, 1981). However, the associated equations adequately serve the purpose of defining g_K and g_{Na} as functions of time and membrane potential difference.

The Hodgkin-Huxley definition of g_K and g_{Na} can be "explained" as follows.

Let the opening of a potassium "channel" which allows potassium to pass through the membrane involve j events. When $j = 4$, for example, we have (inside) $K \Leftrightarrow K' \Leftrightarrow K'' \Leftrightarrow K''' \Leftrightarrow K''''$ (outside), and let us imagine that each such event is enabled by a K-gate molecule in the on-state. If $n(t, E)$ is the proportion of K-gate molecules which are in the on-state at time t and membrane potential difference E, then n^j is the probability that any given potassium channel is open; this probability is proportional to the probability that a potassium ion approaching the membrane will succeed in passing through. Note that j K-gate molecules might actually be one K-gate molecule which assumes 2^j states, only one of which is "open".

Suppose that when all the K-gate molecules are in the on-state, the associated potassium conductance of the membrane is $\bar{g}_K$. In general, then, the potassium conductance of the membrane at voltage E and time t is $\bar{g}_K n^j$.

Suppose further, that K-gate molecules switch reversibly between the on-state and off-state following simple first-order kinetics; so that: $dn/dt = \phi(\alpha_n(1-n) - \beta_n n)$, where ϕ is a temperature compensating factor, and where the "rate constants" α_n and β_n are functions of voltage only! The factor $\phi = 3^{(T-6.3)/10}$, where T is the temperature in degrees Celsius. Hodgkin and Huxley "guessed" that $\alpha_n(E) = -.01(50+E)/(e^{-(50+E)/10}-1)$ and $\beta_n(E) = .125e^{-(60+E)/80}$ by extensive numerical simulation and curve-fitting. The functions α_n and β_n are chosen to "work"; their basic form was obtained from theoretical considerations, assuming that the potential across the membrane acts on charged particles (the gate molecules) to move them in and out of the way within the channels in the membrane.

For sodium, suppose there are both $Na(1)$-gate molecules and $Na(2)$-gate molecules, and let $m(t,E)$ be the proportion of $Na(1)$-gate molecules in the on-state, and let $h(t,E)$ be the proportion of $Na(2)$-gate molecules in the on-state, and finally, suppose opening a sodium channel across the membrane requires p $Na(1)$-events and q $Na(2)$-events. Thus, $g_{Na} = \bar{g}_{Na} m^p h^q$, where g_{Na} is the sodium conductance when all the $Na(1)$ and $Na(2)$ gate molecules are in the on-state.

As before suppose that the $Na(1)$ and $Na(2)$ gate molecules are switching between their on and off states following first order kinetics, so that $dm/dt = \phi(\alpha_m(1-m) - \beta_m m)$, and $dh/dt = \phi(\alpha_h(1-h) - \beta_h h)$, where again the rate constants α_m, β_m, α_h, and β_h are functions of the potential difference, E, which have been empirically chosen as:

$$\begin{aligned}
\alpha_m(E) &= -.1(35+E)/(e^{-(35+E)/10}-1), \\
\beta_m(E) &= 4e^{-(60+E)/18}, \\
\alpha_h(E) &= .07e^{-(60+E)/20}, \quad \text{and} \\
\beta_h(E) &= 1/(e^{-(30+E)/10}+1).
\end{aligned}$$

The constants $\bar{g}_K$ and $\bar{g}_{Na}$ are approximately $36\,\mathrm{m}\Omega^{-1}/\mathrm{cm}^2$ and $120\,\mathrm{m}\Omega^{-1}/\mathrm{cm}^2$ respectively.

For a discussion of current knowledge and theory about the nature of the ionic channels, see "Ionic channels and gating currents in excitable membranes" by Werner Ulbricht, pp. 7:31 in Annual Review of Biophysics and Bioengineering, Vol. 6, 1977, and "Kinetics of channel gating in excitable membranes" by Leon Goldman, pp. 491:526 of Quarterly Reviews of Biophysics, Vol. 9, No. 4, Nov. 1976.

In general we may have an initial state where a "preconditioning" ionic current, I_0, has been applied for a time sufficient to establish a new steady-state. For any I_0-value we have the resulting steady-state voltage, E_{ss}, given by $E_{ss} = \underset{E}{\mathrm{root}}(I_K + I_{Na} + I_L - I_0)$.

Also, initially we assume n, m, and h are such that $dn(0)/dt = dm(0)/dt = dh(0)/dt = 0$, when α_n, β_n, α_m, β_m, α_h, and β_h are determined by the steady-state voltage, E_{ss}.

Thus, we have: $n(0) = n_{ss}(E_{ss})$, $m(0) = m_{ss}(E_{ss})$, and $h(0) = h_{ss}(E_{ss})$ where,

$$\begin{aligned} n_{ss}(E) &= \alpha_n(E)/(\alpha_n(E) + \beta_n(E)), \\ m_{ss}(E) &= \alpha_m(E)/(\alpha_m(E) + \beta_m(E)), \quad \text{and} \\ h_{ss}(E) &= \alpha_h(E)/(\alpha_h(E) + \beta_h(E)). \end{aligned}$$

Also we have:

$$E_{ss}(I_0) = \underset{E}{\text{root}}(\bar{g}_{Na} m_{ss}(E)^p h_{ss}(E)^q (E - E_{Na}) + \bar{g}_K n_{ss}(E)^j (E - E_K) + g_L(E - E_L) - I_0),$$

and $E_m(0) = E_{ss}(I_0)$.

The Hodgkin-Huxley model has $j = 4$, $p = 3$, and $q = 1$.

Note that it is possible to choose I_0 so as to have $E_m(0)$ be any desired steady-state voltage, V, namely: $I_0 = root_I(E_{ss}(I) - V)$. Such a choice for I_0 represents a so-called voltage-clamp situation.

The stimulus current, $I_a(t)$, is a current which varies as desired from the steady-state applied current I_0. $I_a(0)$ need not equal I_0. The interpretation of this is that we have accomodated the axon membrane to the current I_0, and then instantaneously switched the applied current to be $I_a(0)$. Of course the variables, n, m, and h are accomodated to $E_{ss}(I_0)$, not $E_{ss}(I_a(0))$, so they will begin to change in response to this new voltage. Slightly more realism can be had by choosing $I_a(t) = I_0 + (I_1(t) - I_0)$(if $t < s$ then $(1 - exp(-kt))$ else $(1 - exp(-ks))\ exp(-k(t - s))$), for some new current function, I_1, which starts at time 0 and replaces I_0 at some "rise time" rate k, and then is cutoff at time s and decays to zero at the rate k.

Finally, it is possible to apply an impulse current which changes $E_m(0)$ by some desired amount, V_i, without allowing time for $n(0)$, $h(0)$ and $m(0)$ to change. Thus our initial condition becomes $E_m(0) = E_{ss}(I_0) + V_i$.

We have now given the complete Hodgkin-Huxley model for the giant squid axon membrane. There are various improvements; notably the Adelman-FitzHugh extension; however we shall suppose here that a "naked" axon having no Schwann cells is being modeled.

The following MLAB do-file, called `HHF.DO`, can be used to establish the Hodgkin-Huxley equations by typing `DO HHF`.

```
"file: HHF.DO";
EL=-50; ENA=55; EK=-72;
GKBAR=36; GNABAR=120; GL=.3179676;

FUNCTION PHI(TEMP)=3^((TEMP-6.3)/10);
FUNCTION F(X)=IF ABS(X)<.00002 THEN 1 ELSE X/(EXP(X)-1);
```

```
FUNCTION BM(E)=4*EXP((-E-60)/18);
FUNCTION BH(E)=1/(1+EXP(-3-.1*E));
FUNCTION BN(E)=EXP((-E-60)/80)/8;
FUNCTION AM(E)=F((-35-E)/10);
FUNCTION AH(E)=.07*EXP((-E-60)/20);
FUNCTION AN(E)=.1*F((-50-E)/10);
FUNCTION MINF(E)=AM(E)/(AM+BM(E));
FUNCTION HINF(E)=AH(E)/(AH+BH(E));
FUNCTION NINF(E)=AN(E)/(AN+BN(E));

K=25; S=.2;
FUNCTION IA(T)=I0+(I1-I0)*(IF T<S THEN 1-EXP(-K*T) ELSE \
   (1-EXP(-K*S))*EXP(-K*(T-S)));
FUNCTION IONIC(E,M,H,N)=GNABAR*M^3*H*(E-ENA)+GKBAR*N^4*(E-EK)+GL*(E-EL);
FUNCTION EM DIFF T(T)=IA(T)-IONIC(EM,M,H,N);
FUNCTION M DIFF T(T)=PHITEMP*(AM(EM)*(1-M)-BM(EM)*M);
FUNCTION H DIFF T(T)=PHITEMP*(AH(EM)*(1-H)-BH(EM)*H);
FUNCTION N DIFF T(T)=PHITEMP*(AN(EM)*(1-N)-BN(EM)*N);
FUNCTION ISS(E)=IONIC(E,MINF(E),HINF(E),NINF(E));
FUNCTION ESS(I)=ROOT(E,-246,830,ISS(E)-I);
"FOR -62<I<32751, ESS(I) LIES BETWEEN -246 AND 830";

TEMP=6.3; PHITEMP=PHI(TEMP);
V0=-60; I0=0; ESS0=ESS(I0);
VI=0; I1=0;

INITIAL EM(0)=ESS0+VI;
INITIAL M(0)=MINF(ESS0);
INITIAL H(0)=HINF(ESS0);
INITIAL N(0)=NINF(ESS0);
```

The following MLAB do-file, called `HHDO.DO`, can be used to allow the user to establish any desired boundary conditions and then solve the Hodgkin-Huxley equations established with `HHF.DO` given above.

```
"file: HHDO.DO";
I0=0;
TYPE "
IF AN INITIAL VOLTAGE CLAMP OF X mV (-245<X<-12) IS \
DESIRED, TYPE: V0=X.
OTHERWISE SET ANY DESIRED PRECONDITIONING uA CURRENT,Y, \
(-62<Y<1100) BY TYPING: I0=Y.
```

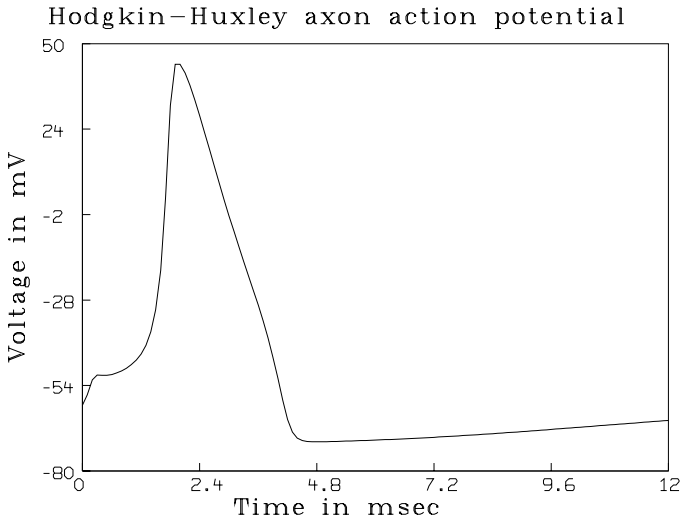
```
OTHERWISE TYPE A RETURN (V0 WILL BE -60 AND I0 WILL BE 0).";
DO KLINE;
IF V0 NOT=-60 THEN (I0=ISS(V0));
ESS0=ESS(I0);

VI=0;
TYPE "
ENTER INITIAL IMPULSE POTENTIAL CHANGE,X, BY TYPING: VI=X.
OTHERWISE TYPE A RETURN (VI WILL BE 0).";
DO KLINE;

I1=0;
TYPE "
DEFINE AN APPLIED CURRENT FUNCTION,IA, BY TYPING: \
FUNCTION IA(T)=desired expression. \
OTHERWISE, IF DESIRED, MERELY SET AN INITIAL .2 MSEC \
DURATION STEP CURRENT, s, (IN uA) BY TYPING: I1=s.\
OTHERWISE TYPE A RETURN (I1 WILL BE 0).";
DO KLINE;

TV=0:12:.1;
TYPE "
SET the TIME-VECTOR, TV, BY TYPING: TV=0:ft:dt (TV=0:12:.1 BY DEFAULT).";
DO KLINE;

METHOD = GEAR;
Q=INTEGRATE(EM DIFF T,M DIFF T,H DIFF T,N DIFF T,TV);
TYPE "
Q COL 1=TV, COL 2=EM, COL 3=EM', COL 4=M, COL 5=M', \
COL 6=H, COL 7=H', COL 8=N, COL 9=N'.";
```

For example, we can compute the famous "action potential" curve as follows.

```
*DO HHF
*DO HHDO

IF AN INITIAL VOLTAGE CLAMP OF X mV (-245<X<-12) IS DESIRED, TYPE: V0
= X. OTHERWISE SET ANY DESIRED PRECONDITIONING uA CURRENT,Y,
(-62<Y<1100) BY TYPING: I0 = Y. OTHERWISE TYPE A RETURN (V0 WILL BE
-60 AND I0 WILL BE 0).

 *
```

```
ENTER INITIAL IMPULSE POTENTIAL CHANGE,X, BY TYPING: VI=X. OTHERWISE
TYPE A RETURN (VI WILL BE 0).
 *

DEFINE AN APPLIED CURRENT FUNCTION,IA, BY TYPING: FUNCTION
IA(T)=desired expression. OTHERWISE, IF DESIRED, MERELY SET AN INITIAL
.2 SECOND DURATION STEP CURRENT,s, (IN uA) BY TYPING: I1 =
s. OTHERWISE TYPE A RETURN (I1 WILL BE 0).

 *I1 = 50

 SET the TIME-VECTOR, TV, BY TYPING: TV = 0:ft:dt (TV=0:12:.1 BY DEFAULT).

 *

 Q COL 1=TV,COL 2=EM,COL 3=EM',COL 4=M,COL 5=M',COL 6=H,COL 7=H',COL 8=N,COL 9=N'

*DRAW Q COL 1:2
*VIEW
```

The resulting picture with added titles is reproduced below.



Many familiar observations can be simulated and the model's response can be investigated by solving the Hodgkin-Huxley equations with various initial conditions. See "Computer analyses of

the excitable membrane" by T. Hironaka and S. Morimoto in Computers and Biomedical Research, Vol. 13, pp. 36:51, 1980, for a number of interesting numerical experiments which can be performed.

One problem with the Hodgkin-Huxley model is that current can flow in each loop of the circuit analog, but this implies sodium and potassium change into each other! This defect is not serious since the "impossible currents" are small, but it is a conceptual error.

One of the main agreements of the Hodgkin-Huxley model with experiment is the suitability of the function $E_m(t)$ as a solution to the cable equation which involves the velocity of an action potential wave propagating along an axon.

Consider an infinitely-long sheathed cable of diameter δ cm with r_i Ω resistance for 1 cm. Let the exterior medium have a resistance of r_o Ω for a 1 cm distance along the cable. Let $I_e(x,t)$ be the current in μA applied to the surface of the short band of cable centered at the point at distance x along the cable and at time t with an electrode with a feedback voltage control which develops whatever voltage is required to obtain the specified current. The circuit is from ground through the electrode, then through the cable sheathing and, via leakage, back to ground. Let $I(x,t)$ be the current in μA flowing across the short band of cable sheathing centered at position x, at time t. $I(x,t)$ arises due to capacitance-driven current, leakage, and, for an axon, active forces in the membrane sheathing itself. Finally let $V(x,t)$ be the voltage across the sheathing (inside potential minus outside potential) at position x and time t.

The cable equation relates these functions as:

$$\partial^2 V/\partial x^2 = (r_i + r_o)\pi\delta I - r_0 I_e.$$

For a constant sheathing resistance of R_m Ω per cm^2, we have

$$I = c((\partial V/\partial t) + V/R_m/(\pi\delta^2/4).$$

If our cable is a giant squid axon, however, the membrane resistance varies, and we may define I by the Hodgkin-Huxley equations as

$$I(x,t) = c(\partial V/\partial t) + g_K(V - E_K) + g_{Na}(V - E_{Na}) + g_L(V - E_L).$$

Thus, we have

$$\partial^2 V/\partial x^2 = (r_i + r_o)\pi\delta[c(\partial V/\partial t) + g_K(V - E_K) + g_{Na}(V - E_{Na}) + g_L(V - E_L)] - r_0 I_e,$$

with $V(x,0) = -60$ mV for $-\infty \leq x \leq \infty$. All the state variables m, h, and n are now functions of x and t, and assumed to be at their normal resting values for all x at time 0. I_e is a perturbing current which will cause potential changes which will propagate along the axon according to the cable equation above.

Another way to begin is to assume that $I_e = 0$, but that $V(x,0)$ is known as an initial condition, where the axon can be assumed to have been charged by a current pulse function which established $V(x,0)$ sufficiently rapidly that m, h, and n have not responded. For example, we may have $V(x,0) = V_0\, exp(-x^2/s^2)$ for some spatial decay constants. We have the additional constraint that $V(x,t) \to -60\,\mu V$ as $t \to \infty$, which will be necessarily satisfied.

The nature of the solution surface V for the initial condition $V(x,0) = V_0\, exp(-x^2/s^2)$ where V_0 is above threshold, is that the curve $V(0,t)$ is an action potential-like curve which is propagated along the axon (we ignore propagation towards $x = -\infty$) so that approximately the same curve appears as $V(x_1, t + x_1/\theta)$.

The parameter θ is the propagation velocity. The initial part of the curve $V(0,t)$ is artificially established by the initial condition and gradually dies out as time progresses, thus we do not have a pure traveling wave. The curve $V(x_1,t)$ for a large value of x_1 is almost the same shape as $V(x_1 + \varepsilon, t + (x_1 + \varepsilon)/\theta)$ however.

If we ignore the transient behavior and assume that t is translated so that $V(0,t)$ is propagated at velocity θ cm/sec in a nearly unchanged form, then we may take $V(x,t) = V(x - \theta t, 0) = V(0, t - x/\theta)$. Let $s = t - x/\theta$, and let $U(s) = V(x,t) = V(0, t - x/q) = V(0,s)$. Then $\partial^2 V/\partial x^2 = (d^2U/ds^2)/\theta^2$ and $\partial V/\partial t = dU/ds$, and we have:

$$d^2U/ds^2 = \theta^2(r_i + r_o)\pi\delta[c(dU/ds) + g_K(U - E_k) + g_{Na}(U - E_{Na}) + g_L(U - E_L)].$$

For $U(0) = -60\,\mu V$ and $dU(0)/ds = 0$, the solution is 0 but since $V(x,t) \neq -60$ mV, even for $t = 0$ and very large x, when the surface $V(x,t)$ is not flat at -60 mV, the appropriate initial conditions for the steady-state wave U is $U(0) = U_0$ and $dU(0)/ds = w_0$ where U_0 and w_0 are very small. $m(0)$, $h(0)$, and $n(0)$ may revert to nearly the resting state values. To obtain a physically meaningful solution, the initial values U_0 and w_0 must be nearly correct. Also an acceptable solution must have $U(s) \to -60$ mV as $s \to \pm\infty$. This only occurs for certain discrete values of θ. The parameter θ is thus analogous to an eigenvalue of a linear operator.

It has been shown that, when U_0, w_0, and θ are carefully chosen, then $U(s)$ approximates the almost steady-state traveling wave observed in a real axon, and moreover that θ is approximately the propagation velocity of about $168000/(\pi\delta c(r_i + r_o))$ cm/sec which is actually observed. This fact constitutes one of the coroborations of the Hodgkin-Huxley model for the squid axon membrane.

Since θ cannot be precisely represented, the numerically computed solution $U(s)$ will not approach -60 mV as $s \to \pm\infty$, but rather will approach $+\infty$ or $-\infty$, depending upon the error in θ (and U_0 and w_0).

12 Adaptive Delta Modulation

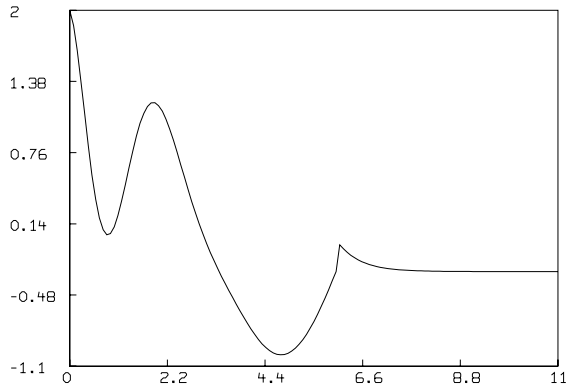
Given an analog signal $f(t)$, for $s_0 \leq t \leq s_1$, we may wish to transmit this curve to a remote site. For example, f may be a voice signal, or a continuous time series measurement reading of some transducer. Digital encoding generally allows data compression, permitting fewer bits per second to be transmitted, and error-correcting codes can be used as desired to further improve accuracy. We may transmit a train of rectangular pulses, using repeaters as needed, or we may use any of the various amplitude, frequency, or phase modulation methods for transmission as a band-limited oscillating signal. Digital encoding for data compression is also a useful device in storing digitally-encoded curves.

One common scheme is so-called delta-modulation encoding. This method can be used for sound and/or video digital radio transmissions and recordings as well as other types of signals. If the signal, f, is to be digitized for computational purposes, then delta-modulation encoding serves as a cheap and efficient analog-to-digital converter.

The basic idea is quite simple. Given the initial value, $h = f(s_0)$, a sampling interval, α, and an increment value, δ, we may interpret a string of bits b_1, b_2, ..., b_n to obtain a specification of estimated signal values $\hat{f}(t_1)$, $\hat{f}(t_2)$, ..., $\hat{f}(t_n)$, where $t_i = s_0 + i\alpha$ for $0 \leq i \leq n$, as follows. Each bit b_i indicates adding the constant δ if $b_i = 1$ and subtracting δ if $b_i = 0$, so that

$$\hat{f}(s_0 + k\alpha) = h + \sum_{i=1}^{k} (2b_i - 1)\delta.$$

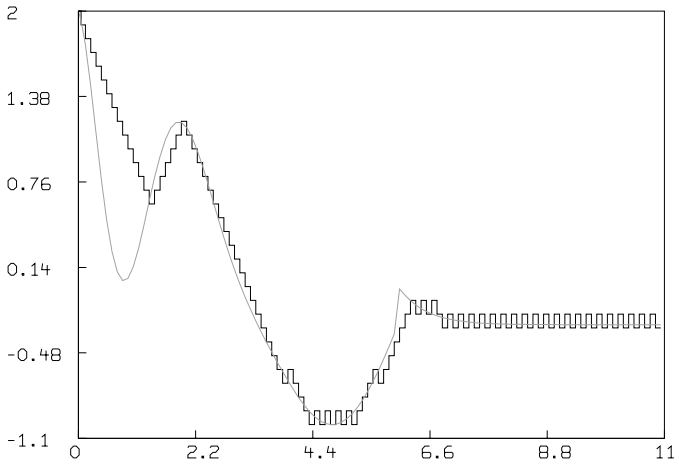
For example, consider $f(t) =$if $t \leq 6$ then $\sin(t)+2\cos(3t)\exp(-t)$ else $-.27614(1-\exp(-2(t-6)))$. A graph of f on the interval $[0, 11]$ is shown below



Choosing the parameter values: $s_0 = 0$, $s_1 = 11$, $\delta = .1$ and $\alpha = .1$, we have $h = 2$ and the binary digital sequence:

```
00000000000000111111000000000000000000010000101010101111011111110101001010101
10101010101010101010101010101010101.
```

This sequence specifies $\hat{f}$ as shown below, drawn with the traditional step-function interpolation which would be seen on an oscilloscope. The smooth curve is the true signal, and the stepped curve is the step-function form of $\hat{f}$. Thus the function $\hat{f}$ on the interval 0 to 11 is represented with 110 bits.



The encoding process computes the ith bit, b_i, by predicting $f(t_i)$ to be some value P_i, and then b_i is taken as one if $f(t_i) > P_i$ and zero otherwise. The choice of P_i used here is merely the previous estimate value $\hat{f}(t_{i-1})$.

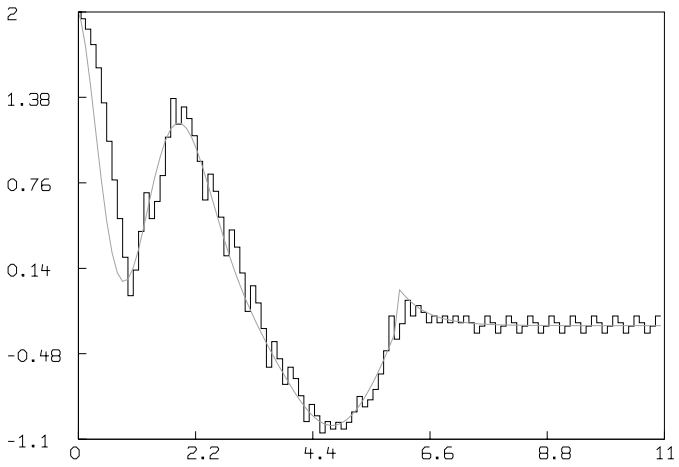
In general, to obtain a reasonable estimate for f, the sampling interval, α, must be such that $(s_1 - s_0)/\alpha$ is greater than the Nyquist frequency, which is twice the frequency of the highest-frequency component present in the signal f, and δ must be small enough to track high-frequency oscillations without undue shape distortion. A practical way to choose α and δ is to choose δ as the largest absolute error by which $\hat{f}$ may deviate from f, and then choose $\alpha = \delta/w$, where $w = \max_{s_0 \le t \le s_1} |df(t)/dt|$, the maximum absolute slope of f. In the previous picture, α is clearly too large for the chosen δ; the result is the large error in the initial part of the signal, called slope overload error, where the slope is too large in magnitude for $\delta = .1$ and $\alpha = .1$ to track the signal.

Even when the signal is being appropriately followed, the estimate oscillates about it. This so-called granular noise is unavoidable, although its size is controlled by δ. Note that the error characteristics of the estimator $\hat{f}$ are given by $|f(t) - \hat{f}(t)| < \delta$ for $s_0 \le t \le s_1$, assuming α is small

enough. This is an absolute error criterion rather than a relative error criterion, and $\hat{f}$ behaves like a Chebychev approximation to f.

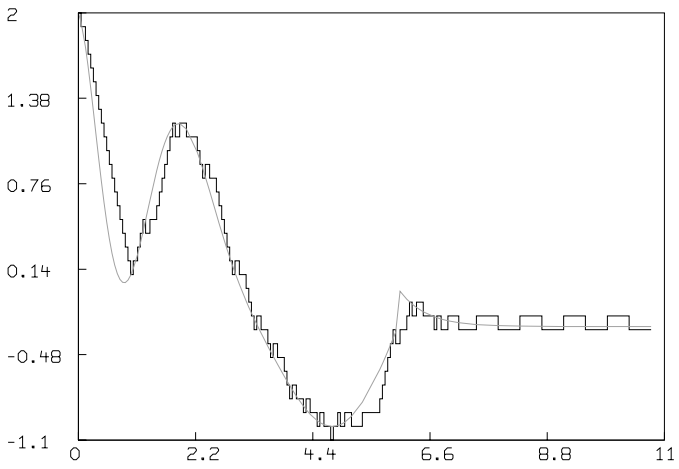
Note that a delta-modulation signal is very sensitive to transmission error. Changing a burst of a dozen bits or so during transmission can destroy the validity of the remaining bits. However, higher sampling rates mean short burst errors are less harmful, and methods to periodically restart the delta-modulation process can be included in a practical transmission system. In general, delta-modulation is a very efficient way to encode a signal. It is not clear how to define the notion of the efficiency of an approximation (as opposed to an exact encoding) in a precise information-theoretic manner, but this is an intriguing direction for investigation.

We can elaborate on the basic idea of delta-modulation in several ways. First, it has been proposed that the increment δ can assume various values, depending upon the past tracking of the signal. If we output m ones or zeros in a row (indicating a region of large absolute slope), we can increase δ, replacing δ with $3\delta/2$ for example. If we output m alternating ones and zeros, we can then decrease δ, say to $2\delta/3$. The new value of δ is to apply to the current bit being output which forms the mth bit of the change-triggering pattern. This device is called adaptive delta-modulation. Changing δ, however, is not always an improvement. Indeed the signal may be such that a closely-tracking, but lagging, estimate becomes an oscillating estimate with greater absolute error when adaptive delta-modulation is employed. For example, for the signal shown above, with $\alpha = .1$ (too large), and δ varying within the limits .05 to .28, based on $m = 2$, so that two zeros or ones in a row increases δ, while two different values decreases δ, we obtain the approximation shown below.

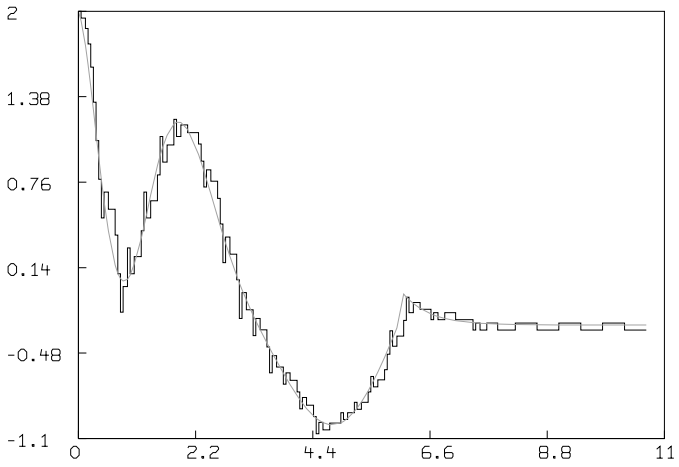


Another idea is to allow the sampling interval, α, to change. This is not very useful for hardware, which is more conveniently designed to use a fixed clock rate, but for data-compression for digital

storage purposes, varying α may allow fewer bits to be used to encode a given curve. We may increase α when the previous m bits have alternated in value, but when m ones or zeros in a row occur we reduce α to re-establish the fastest sampling rate. This permits large steps in slowly-varying regions, but it allows relatively large deviations in the estimate to occur at turning points where f changes from being flat to sharply rising or falling. Choosing $m = 2$ minimizes this effect, but it is still noticable. Lagging tracking at turning points is the major flaw in most delta-modulation schemes. The step-function estimate of our example signal is shown below where we have replaced α by 1.6α up to a limit of .41 whenever the previous two bits were the same and reset α to .05 otherwise. We have fixed $\delta = .1$ (which is too small to be completely reasonable for our range of α). Note we now have as our estimated points: $(s_0, f(s_0)), (t_1, \hat{f}(t_1)), ..., (t_n, \hat{f}(t_n))$for some irregular mesh$s_0 < t_1 < ... < t_n \leq s_1$.



If we allow δ and α to both vary as described above with δ in the interval $[.05, .28]$ and α in the interval $[.05, .41]$, we obtain the following approximation.



In order to compute the bit b_i which determines the point $(t_i, \hat{f}(t_i))$ when encoding the signal, f, we form an estimate of $f(t_i)$, called P_i, where P_i predicts $f(t_i)$ given the previous estimate points $(t_0, \hat{f}(t_0)), (t_1, \hat{f}(t_1)), ..., (t_{i-1}, \hat{f}(t_{i-1}))$. Then if P_i is less than $f(t_i)$ we output $b_i = 1$ and otherwise for $P_i \geq f(t_i)$, b_i is output as zero.

This same predictor must be used in decoding the bitstring, b, in order to compute $\hat{f}$; this is why P_i depends on $\hat{f}$-values, and not on f-values.

In the discussion above, we have used the simple predictor $P_i = \hat{f}(t_{i-1})$. Other predictor schemes are possible, and may provide better performance, allowing smaller δ and/or larger α values to be used. Of course other predictors do not necessarily have the property that $b_i = 1 \text{iff} \hat{f}(t_{i-1}) < f(t_i)$.

In general, then, our decoding calculation for obtaining $\hat{f}(t_i) \text{is} \hat{f}(t_i) = P_i + \delta_i(2b_i - 1) \text{for} 1 \leq i \leq n$, where δ_i is the particular value of the increment used when b_i is computed; δ_i is a function of $b_1, b_2, ..., b_{i-1}$.

We could choose P_i to be determined by extrapolation on the linear or quadratic curve passing through the previous two or three points of the estimate, but experimentation shows the error in this form of predictor to be worse than the simple constant form. A more promising idea is to use an extrapolation on a polynomial which fits the previous k estimate points in the least-squares (or better yet L^1-norm) sense. This works adequately although the predictor is slow to track f on turning intervals. A more elaborate filter predictor might be worth exploring, but weighted averages of the previous k points and weighted averages of the linear predictors based on the k previous points taken two at a time also perform no better than the simple constant predictor. Thus finding a better predictor seems difficult and the constant form seems to be the best practical choice as well as the simplest.

If a good predictor is used, however, the adaptive variation of δ and α becomes less useful. Indeed, with a perfect predictor $P_i = f(t_i)$, our output is all zero bits, and α will foolishly stay at its minimum, while δ will stay at its maximum. The resulting $\hat{f}$ curve tracks below f with the maximum allowable error. Even using merely a good predictor means we should sharply decrease δ and perhaps increase α.

Algorithms for encoding and decoding an adaptive delta-modulation signal are presented below, with $m = 2$ and with the simple constant predictor function, $P_i = \hat{f}(t_{i-1})$, used above. The constants $C, \delta_{min}, \delta_{max}, g, \alpha_{min}, \text{and} \alpha_{max}$ are global parameters which may be "tuned" to the situation. We have used $c = 1.5, \delta_{min} = .05, \delta_{max} = .28, g = 1.6, \alpha_{min} = .05, \text{and} \alpha_{max} = .41$ in the example above.

```
ADMencode(s0, s1, d, a, f):
begin  real x1,x2,t,h,b;
real procedure P: [h <-h+(2b-1)d; return(h)];
b,x2.5; ts0; hf(s0);
send (t0, h,d, a) to ADMdecode; "This send starts the decoder."
while t <=  s1 do
      [tt+a; x1x2; if f(t) > P then b1 else b0;
       x2b; output(b);
       if x1=x2 then (aamin; dmin(cd,dmax))
       else (dmax(d/c,dmin); amin(ga,amax))
     ];
output(end-message); "This will stop the decoder."
end.

ADMdecode(t, h, d, a):
begin  real x1,x2,h,b;
real procedure P: [hh+(2b-1)d; return(h)];
b,x2.5;
repeat
     [output (t,P);
      "Initially, the output point (t,P) is the point (s0,(s0));
        then subsequent points on the -curve are output."
       receive a bit in b, on end-message, goto exit;
       x1x2; x2b; tt+a;
       if x1=x2 then (aamin; dmin(cd,dmax))
       else (dmax(d/c,dmin); amin(ga,amax))
      ];
exit: end.
```

There are many variations possible, based on different ranges for δ and α, and different formulas for changing them. In our example, we actually do about as well with even fewer bits than used

above when we let δ assume just the values .1 and .2 and let α assume just the values .1 and .2. Another idea is to compute b by the test: if$f(t) > P - a(log(1 + |f'(t)|)/k)$then$b \leftarrow 1$else$b \leftarrow 0$. This use of slope information can perhaps be marginally useful, even though it produces "lies" about the location of f. Some suggestions have been made that an "intelligent" encoder could hold m signal values in a memory, and carefully compute the best block of one or more output bits based on this "look-ahead". Perhaps just provisional output bits could be held, so that we would hold m bits in a local memory and output each bit based upon the $m-1$ future bits which have been provisionally computed and stored, but it seems difficult to make a useful scheme out of this idea.

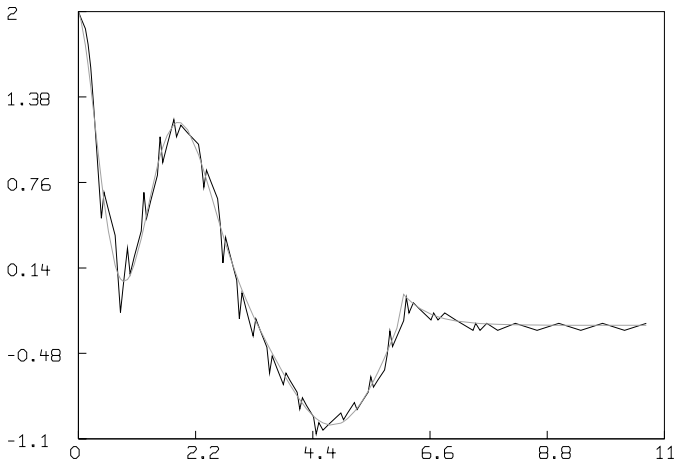
Also, when we use $m > 2$ to adaptively change δ and/or α, we could use a $2^m -$ bit decision tree to pick carefully-chosen δ and α modifications; this scheme does work well, but at a high cost in complexity.

All the pictures in this example were produced with MLAB. The precise statements needed to do such computer experiments with MLAB are shown below. The following text is a do-file, which when invoked with an appropriate MLAB do-command, is executed to produce matrices suitable for graphing.

```
FUNCTION F(T)=IF T<=6 THEN SIN(T)+2*COS(3*T)*EXP(-T) ELSE J-J*EXP(-2*(T-6));
J = -.27614;MINA = .05; MAXA = .41; MAXD = .28; MIND = .05;
T0 = 0; T1 = 11; D = MIND; A = MINA; G = 1.6; C = 1.5;
TYPE "SET T0,T1,A,D, ETC. AS DESIRED."; DO CON;

FUNCTION ADM(I)=
IF T+A<=T1 THEN (B_((PV_P(ME[I]_OLDP(ADM)))<=F(MT[I+1]_(T_T+A))))
 +0*(A_NEWA(X1_X2,X2_B))+0*(D_NEWD()) ELSE 1000-I;

FUNCTION OLDP(B)=PV+D*(2*B-1);
FUNCTION P(H1)=H1;

FUNCTION NEWA(X1,X2)=IF X1=X2 THEN MINA ELSE MIN(G*A,MAXA);
FUNCTION NEWD()=IF X1=X2 THEN MIN(D*C,MAXD) ELSE MAX(D/C,MIND);

X2 = 1; ADM = .5; T = T0; IF T1 <= T0 THEN TYPE ("null interval");  PV = F(T0);
"PRE-ALLOCATE THE ARRAYS MT, ME.";
MT = 0^^360; ME[360] = mt; MT[1] = T0;

MB = ADM ON 1:360;
N = MAXI(MB); IF N >=360 THEN TYPE "TOO MANY POINTS.";

ME(N) = OLDP(B); ME = ME ROW 1:N; MT = MT ROW 1:N; MB = MB ROW 1:(N-1);
SME = MESH(MT,MT)&'ROTATE(MESH(ME,ME),1); DELETE SME ROW (1,N+N);
```

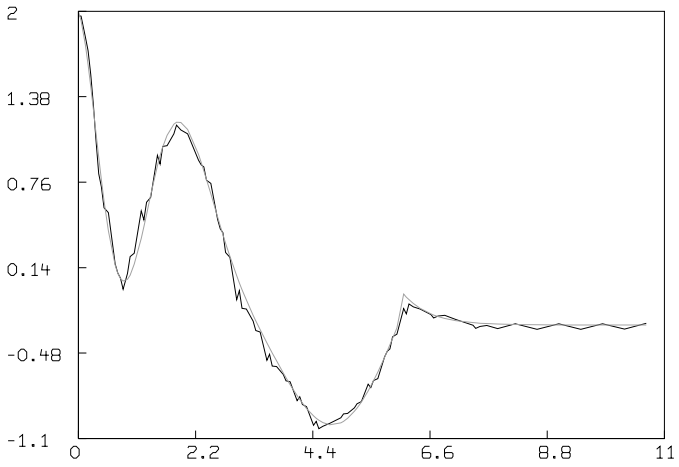
```
ME = MT&'ME; MF = POINTS(F,MT);
"MB IS THE DELTA-MODULATION BIT-VECTOR, MF IS THE SAMPLED
SIGNAL POINTS, ME IS THE ESTIMATED SIGNAL POINTS, AND SME
IS THE STEP-FUNCTION ESTIMATE.";
```

It is worth observing that the step-function approximation form of drawing $\hat{f}$ is somewhat deceiving. A simple straight-line interpolation is a more reasonable presentation. For example, the (δ, α)-varying estimate shown above is seen again here using linear connecting segments.



Viewing a picture such as this suggests that we might estimate f more exactly if we compute the midpoints of the line-segments in the graph of f which cross the graph of $\hat{f}$. But this set of crossing points is only marginally useful when filtering is used. Generally the input, f, should be prefiltered to pass only frequencies below an appropriate cutoff point. In any event, the output points, $(t, \hat{f}(t))$, have both sampling and encoding error, and the output should be filtered to remove some of the noise. The filtering can be done with a distance-weighted smoothing transformation in software, or with an appropriate low-pass filter in hardware.

The smoothed variable δ and α estimate is shown below. A doubly-smoothed estimate would be even better in regions of slowly-changing slope.


2
1.38
0.76
0.14
-0.48
-1.1
0
2.2
4.4
6.6
8.8
11

13 Fitting Exponential Models

Fitting models of the form $f(x) = a_1 \cdot \exp(b_1 x) + a_2 \exp(b_2 x) + \cdots + a_k \exp(b_k x)$ to given data points (x_i, y_i), $i = 1, 2, \ldots, n$ is a common problem. Unfortunately, the results are often unsatisfying, even when they can be obtained. It will often be preferable to use a differential-equation-based model, *e.g.* a compartmental model, which has some physical basis. We may illustrate the difficulties with a particular example. Let us consider the following data due to Dr. Paul Schloerb of the University of Rochester.

x_i	y_i
1	39.814
2	32.269
3	29.431
4	27.481
5	26.086
6	25.757
7	24.932
8	23.928
9	22.415
10	22.548
11	21.900
12	20.527
13	20.695
14	20.105
15	19.516
16	19.640
17	19.346
18	18.927
19	18.857
20	17.652

We will use the model $f(x) = a_1 \exp(b_1 x) + a_2 \exp(b_2 x) + a_3$. Our goal is to estimate the values a_1, a_2, a_3, b_1, and b_2, so that $f(x_i) \approx y_i$, employing the least-squares minimization method. We may enter this data and the specified model in MLAB as follows.

```
* M = (1:20)&'kread(20)
39.814 32.269 29.431 27.481 26.086 25.757 24.932 23.928 22.415 22.548 21.9
20.527 20.695 20.105 19.516 19.64 19.346 18.927 18.857 17.652
* FUNCTION F(X)=A1*EXP(B1*X)+A2*EXP(B2*X)+A3
* CONSTRAINTS Q = {A1 > 0, A2 > 0, A3 > 0, B1 < 0, B2 < 0}
```

We often need reasonable guesses for the parameters a_1, a_2, a_3, b_1 and b_2, which are sufficiently close to the "true" values. Otherwise, we may obtain a solution which corresponds to an undesirable local minimum rather than that local minimum associated with the most physically-meaningful solution, as in the following example.

```
* A1 = 1; A2 = .5; A3 = 1; B1 = -1; B2 = -.5
* FIT(A1,A2,A3,B1,B2), F TO M ,CONSTRAINTS Q
matherr: underflow error in exp
         arg = -7.409512e+02
         return value: 0.000000e+00
matherr: underflow error in exp
         arg = -8.150463e+02
         return value: 0.000000e+00
matherr: underflow error in exp
         arg = -8.891414e+02
         return value: 0.000000e+00
matherr: underflow error in exp
         arg = -9.632365e+02
         return value: 0.000000e+00
matherr: underflow error in exp
         arg = -1.037332e+03
         return value: 0.000000e+00
final parameter values
      value               error         dependency    parameter
  3.6182537e-16      1.797693135e+308         1           A1
 2.42048135e-17      1.797693135e+308         1           A2
    23.5913          1.372938916              0           A3
   -5141744.558      1.797693135e+308         1           B1
   -176780.4082      1.797693135e+308         1           B2
5 iterations
CONVERGED
best weighted sum of squares = 5.654884e+02
weighted root mean square error = 6.139969e+00
weighted deviation fraction = 1.854011e-01
R squared = 6.031254e-15
no active constraints
```

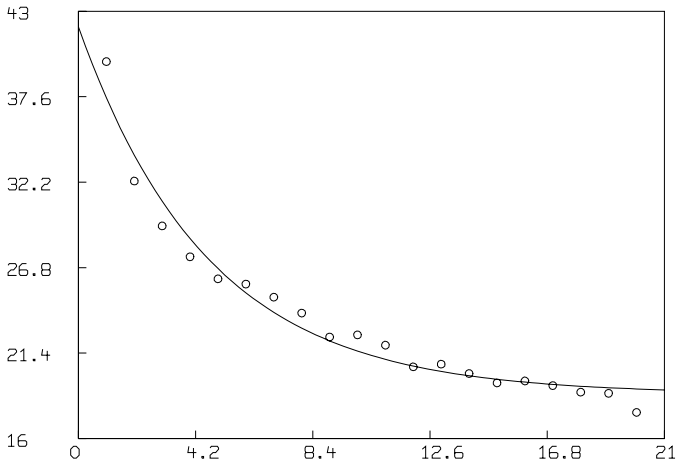
High dependency values near 1 implies an ill-conditioned Jacobian matrix. There may be too many parameters, a lack of data in an important range, or poorly-chosen initial parameter guesses.

It is possible to make bad initial guesses in many ways. If we start with `A1=A2=A3=1` and `B1=B2=-1` for example, then the two exponential terms in our model are identical, and, in fact, the model

has degenerated into a one-exponential model. The curve-fitting process may or may not be able to re-separate the two distinct exponential components, depending upon whether there are lucky intermediate parameter values generated at the beginning of an iteration or not. Often the "wild" parameter vectors which arise for a singular Jacobian matrix with a nearly-unmagnified diagonal are "unlucky". We shall call such initial guesses degenerate.

The correct approach is to make reasonable non-degenerate guesses to start with. If we fit a one-exponential term model to the data, the result may help in starting to fit the two-exponential term model. By looking at the data, we may proceed as follows.

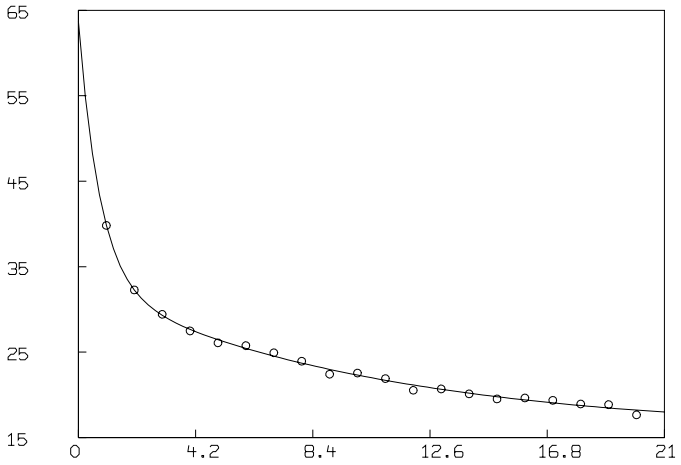
```
* A1=20; A2=0; A3=20; B1=-.1; B2=0;
* FIT (A1,B1,A3), F TO M, CONSTRAINTS Q
Begin iteration 1 bestsosq=4.770403e+02
Begin iteration 2 bestsosq=1.287325e+02
Begin iteration 3 bestsosq=1.904221e+01
Begin iteration 4 bestsosq=1.892617e+01
final parameter values
      value               error        dependency  parameter
    23.19717945         1.240156312    0.6125079766   A1
  -0.2148403998       0.02481714336    0.8665292689   B1
     18.8176717        0.5477893054    0.8145001634   A3
4 iterations
CONVERGED
best weighted sum of squares = 1.892560e+01
weighted root mean square error = 1.055116e+00
weighted deviation fraction = 3.492313e-02
R squared = 9.665323e-01
no active constraints
* DRAW M, POINTTYPE  CIRCLE PTSIZE .01, LINETYPE NONE
* DRAW p = POINTS(F, 0:21:.25); VIEW;
```



Now we may guess introductory values for `A2` and `B2`, and proceed to fit the complete two-exponential term model.

```
* A2 = 10; B2 = -2
* FIT(A1,A2,A3,B1,B2), F TO M, CONSTRAINTS Q
Begin iteration 1 bestsosq=1.529741e+01
Begin iteration 2 bestsosq=1.001018e+01
Begin iteration 3 bestsosq=7.869168e+00
Begin iteration 4 bestsosq=6.509648e+00
Begin iteration 5 bestsosq=5.557916e+00
Begin iteration 6 bestsosq=4.874465e+00
Begin iteration 7 bestsosq=4.371166e+00
Begin iteration 8 bestsosq=3.992295e+00
Begin iteration 9 bestsosq=2.826510e+00
Begin iteration 10 bestsosq=2.444951e+00
Begin iteration 11 bestsosq=2.305400e+00
Begin iteration 12 bestsosq=2.228857e+00
Begin iteration 13 bestsosq=2.179124e+00
Begin iteration 14 bestsosq=2.143836e+00
Begin iteration 15 bestsosq=2.115133e+00
Begin iteration 16 bestsosq=2.100721e+00
Begin iteration 17 bestsosq=2.005968e+00
final parameter values
      value           error        dependency    parameter
```

```
    17.35185308   0.5924659802    0.9172348481   A1
    30.65781056    6.440647997    0.9597145815   A2
    15.67738647    1.020820115    0.9935853439   A3
  -0.0961929828  0.01697024506    0.9926829646   B1
   -1.297370234   0.2737645734    0.9810419889   B2
17 iterations
CONVERGED
best weighted sum of squares = 2.005363e+00
weighted root mean square error = 3.656376e-01
weighted deviation fraction = 9.897291e-03
R squared = 9.964537e-01
no active constraints
* DRAW p = POINTS(F,0:21:.25); VIEW;
```



14 A Pharmacological Model

This example shows several important features of MLAB. These features include simultaneously fitting several functions with shared parameters to different data sets. The functions which make up the model are defined by a set of differential equations. These differential equations turn out to be stiff and thus require a suitable implicit method such as *Gear*'s method to solve them numerically in a reasonable amount of time. The data used here was provided by Nicholas Holford as a challenge for modelers; it is widely disparate in scale, and we show how to use weight vectors to handle this. There is also missing data at different time points; *MLAB* handles this problem automatically (zero weights are generated internally to correspond to missing data).

The problem setting is as follows: 48.15 milligrams of a drug D is given by mouth, and blood concentrations of the drug D and also of its only metabolite M are measured. Also the cumulative amounts of D and M in the urine are measured. Thus, we have the following data.

Note: In the data table below, blanks represent missing data. In order to prepare the data for input, some value must be supplied at each place where a number is missing. Any unique value may be used for these missing values since we will remove them later. For this example, zero will be entered for the missing table values.

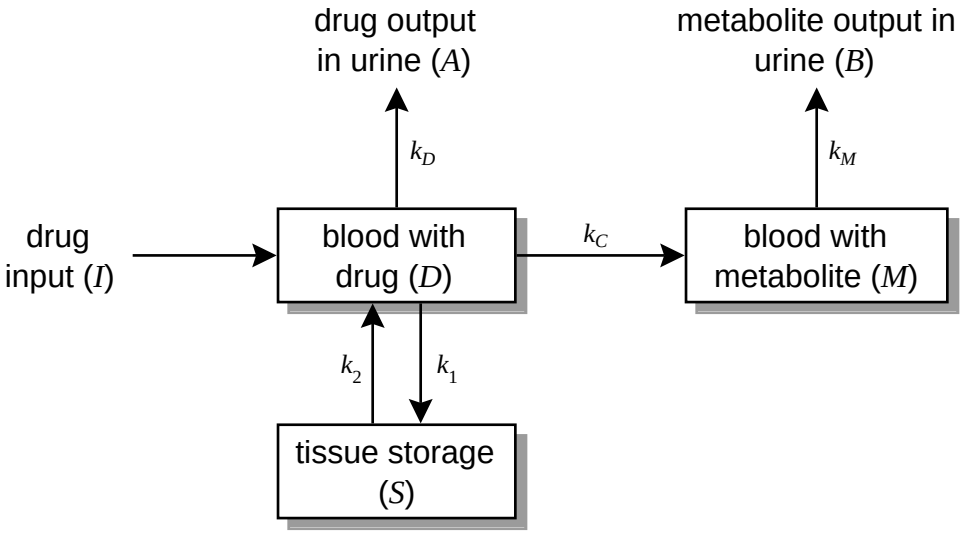
time (hours)	blood-D (mg/liter)	blood-M (mg/liter)	urine-D (mg)	urine-M (mg)
.82	.1746	.822051		
1			1.87	7.23
1.2	.166	1.143786		
1.4	.1264	1.152462		
2	.1092	.859647	3.23	15.53
2.4	.0904	.648531		
2.9	.0828	.601536		
3			4.02	21.15
3.38	.0704	.381744		
3.92	.0591	.402711		
4			4.59	25.88
4.42	.0511	.30366		
5.18	.0355	.252327		
6			5.77	32.42
6.35	.0148	.143154		
8			6.3	34.89
8.3	.0081	.063624		
10			6.51	36.16
10.28	.0047	.033258		
12			6.65	37.06
12.4	.0026	.020967		
24			6.92	38.7
24.57	.0009	.006507		
48			7.3	40.29
72			7.38	40.77

We wish to devise a model for the uptake, metabolic conversion and excretion of this drug, and curve-fit to adjust the model to fit the observed data.

The error in the blood concentration measurements has a variance which is roughly proportional to the square of the true measurement value. The error in the urine amounts has a more-rearly constant variance. Whatever model we use to predict $D(t)$ (blood drug concentration at time t), $M(t)$ (blood metabolite concentration at time t), $A(t)$ (urine drug amount at time t), and $B(t)$ (urine metabolite amount at time t), we will want to weight our observations by weights which are proportional to the reciprocals of the variances.

We will use the *MLAB* `EWT` operator which employs the deviations from a smoothed spline to estimate the errors in data values. Using `EWT` on the various data sets produces error estimates scaled comparably to the underlying error of the data sets themselves. This has the effect that the deviations will each be approximately sized so that each of our sets of observed data is given more or less correct weight in the sum of squares to be minimized.

As a model, let us consider the following compartmental form.



This is a highly simplified model; the path from the blood drug compartment to the blood metabolite compartment should probably include a metabolic conversion compartment, and perhaps the metabolite should go in and out of tissue as does the drug `D`, but this model is already near the limit of what we can usefully fit to the data.

Let V_B be the volume in liters of the blood and let V_S be the volume in liters of the tissue of the subject being studied. We let V_M denote the volume in liters of the blood-metabolite compartment; we would expect that $V_M = V_B$, but we may obtain a better fit when this constraint is not honored.

Let $D(t) =$ the concentration of the drug D in the blood at the time t, let $M(t)$ be the concentration of the metabolite M in the blood at time t, let $S(t)$ be the concentration of the drug D in the tissue at time t. Also, let $A(t)$ be the cumulative amount of drug D which has appeared in the urine by time t, and let $B(t)$ be the cumulative amount of the metabolite M which has appeared in the urine by time t.

We can write the following model involving a first-order ordinary differential equation for each compartment.

$$\begin{aligned} D' &= (I(t) - (k_1 + k_D + k_C)D + k_2 S)/V_B \\ S' &= (k_1 D - k_2 S)/V_S \\ M' &= (k_C D - k_M M)/V_M \\ A' &= k_D D \\ B' &= k_M M \end{aligned}$$

with $D(0) = 0$, $M(0) = 0$, $S(0) = 0$, $A(0) = 0$, and $B(0) = 0$.

The choice of the input function, I, is somewhat arbitrary. However, if the drug is absorbed as fast as it passes at a constant rate into the small intestine, we may choose $I(t) =$ if $t < E_T$ then $48.15/E_T$ else 0, so 48.15 mg. of the drug is introduced at a constant rate over E_T hours.

We could instead use the form $I(t) = 48.15 \cdot H \cdot \exp(-H \cdot t)$, and introduce the constraint $H > 0$. It turns out this makes little difference in the final results.

Note that k_M, K_D, k_C, k_1 and k_2 are in units of liters/hour, the derivatives D', S', and M' are in units of mg/liter/hour, A' and B' are in units of mg/hour, D, S, and M are in units of mg/liter, A and B are in units of mg, V_B, V_S and V_M are in units of liters, and $I(t)$ is in units of mg/hour, and these units are dimensionally consistent.

It is necessary to use constraints for fitting this model; without them, the parameters may well be assigned foolish values where the differential equations cannot be integrated numerically. Let us assume the following constraints.

$\{ .1 < E_T < 70,\ 0 < k_1,\ 0 < k_2,\ 0 < k_D,\ 0 < k_C,\ 0 < k_M,\ 3 < V_B < 7,\ 10 < V_S < 100,\ V_M > .01\}$

Now we need initial guesses for all the parameters. These guesses must be suitable; arbitrary guesses can lead to unreasonable final fit values, or even cause the fitting process to be unable to proceed due to excessive stiffness of the differential equations!

Suppose a unit amount of drug diffuses from blood into tissue so that half of it is transfered in one hour. Then if y is the amount of drug in the blood, we have $y' = -k_1 y$ with $y(0) = 1$, and $y(1) = .5$, and so $k_1 \approx .7$. Let us also guess that $k_2 = .7$.

If half of a unit amount of drug is cleared from the blood and transferred to the urine by the kidneys in about 4 hours, then $k_D \approx .17$. Let us also guess that $k_M = .17$. Similarly, let us guess $k_C = .17$.

Finally we choose $V_B = 5$, $V_M = 5$, $V_S = 40$, and $E_T = 1$.

Now we may proceed in MLAB as follows. First we enter the data listed above, with zeros for missing values, and then we construct the corresponding weight vectors `WD`, `WM`, `WA`, and `WB`.

```
n = read(dataf, 100, 5)
tv = n col 1;  "tv = time values"
dv = n col 2;  "dv = blood drug data."
mv = n col 3;  "mv = blood metabolite data."
av = n col 4;  "av = urine drug data."
bv = n col 5;  "bv = urine metabolite data."
```

```
dv = tv &' dv;  dv = compress(dv,2); wd =ewt(dv)
mv = tv &' mv;  mv = compress(mv,2); wm =ewt(mv)
av = tv &' av;  av = compress(av,2); wa =ewt(av)
bv = tv &' bv;  bv = compress(bv,2); wb =ewt(bv)
```

Now we enter our model, our constraints, and our inital guesses.

```
function d't(t) = (i(t) - (k1 + kd + kc)*d + k2*s)/vb
function s't(t) = (k1*d - k2*s)/vs
function m't(t) = (kc*d - km*m)/vm
function a't(t) = kd*d
function b't(t) = km*m
function i(t) = if t<et then dose/et else 0

initial d(0) = d0
initial s(0) = 0
initial m(0) = 0
initial a(0) = 0
initial b(0) = 0

d0 = 0; dose = 48.15

k1=.7;k2=k1;kd=.17;km=kd;kc=kd;vb=5;vs=40;et=1;vm=5

constraints c = {k1>0, k2>0, kc>0, km>0, et>.1, et<70, vb>3, \
: vb<7, vs>10, vs<100, vm>.01}
```

Now we proceed to fit. Due to the large amount of time needed to fit this stiff model, we use Gear's method with a tolerance of .01.

```
method = gear;
maxiter = 100
errfac = 0.01

fit(k1,k2,kc,kd,km,et,vb,vs,vm), \
: d to dv with weight wd, m to mv with weight wm, \
: a to av with weight wa, b to bv with weight wb, constraints c

final parameter values
      value              error               dependency    parameter
    19.55670378         12.41128315          0.7277144348   K1
```

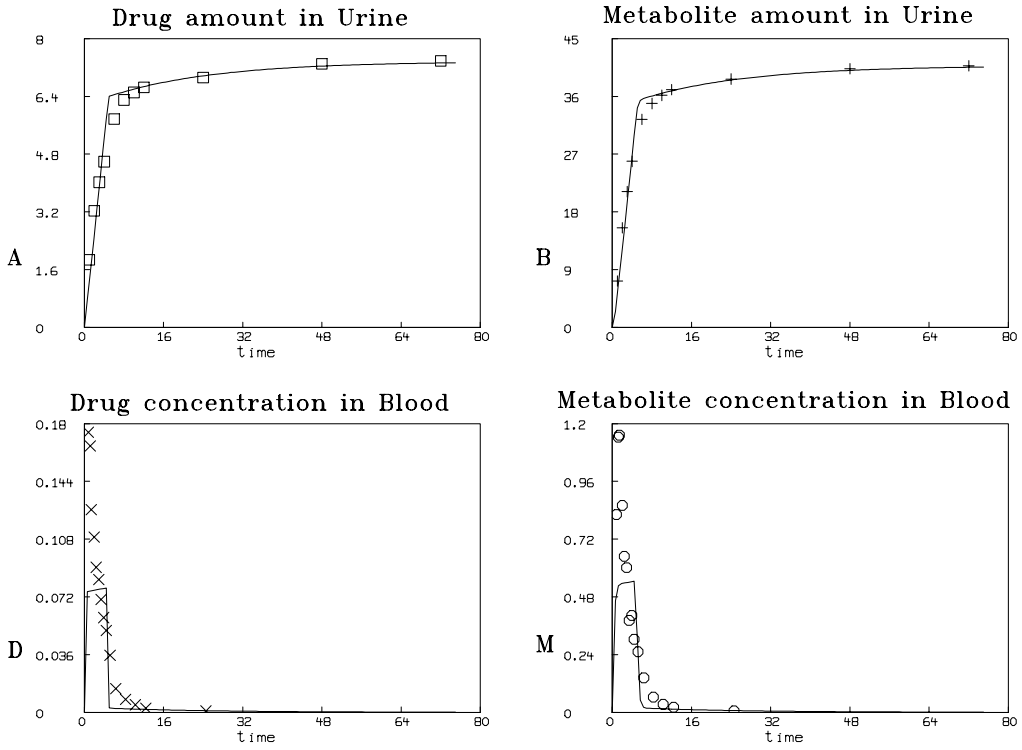
```
    4.829029615         56.05648908          0.9970952774   K2
    95.71231349         18.05875113          0.9611351847   KC
    17.29009355         3.174400652          0.9601255658   KD
    13.57478156         2.526143866          0.6637941471   KM
    4.835844445        0.8222361079           0.945128974   ET
    4.168221663          13.3558721          0.5651377681   VB
      88.309337         930.8309603           0.997317646   VS
    3.929260682         8.161760496          0.9041479451   VM
22 iterations
CONVERGED
best weighted sum of squares = 3.054944e+02
weighted root mean square error = 2.665431e+00
weighted deviation fraction = 5.505088e-02
R squared = 9.959130e-01
no active constraints
```

Now we will graph our four data sets together with the best-fit curves produced by solving our system of differential equations with the parameter values obtained above.

Note that we must beware of assuming that our obtained parameters have any physical significance. It is unlikely, for example, that the actual compartment volumes are close to the values we have for V_B, V_M and V_S. Our model may be useful for prediction purposes, but it is not useful for gaining insight into any actual physiological mechanisms.

```
tv=0:75!120
draw points(d,tv) color brown
draw dv color red pt xpt lt none
image color white
top title " Drug concentration in Blood" font 11 size .03
left title "'-90AD" font 11 size .03
bottom title "time"
frame 0 to .5, 0 to .5
w1=w

draw points(m,tv) color blue
draw mv color blue pt octagon lt none
image color yellow; frame color green
top title "Metabolite concentration in Blood" font 11 color brown size .03
left title "'-90AM" font 11 size .03
bottom title "time"
frame .5 to 1, 0 to .5
w2=w
```

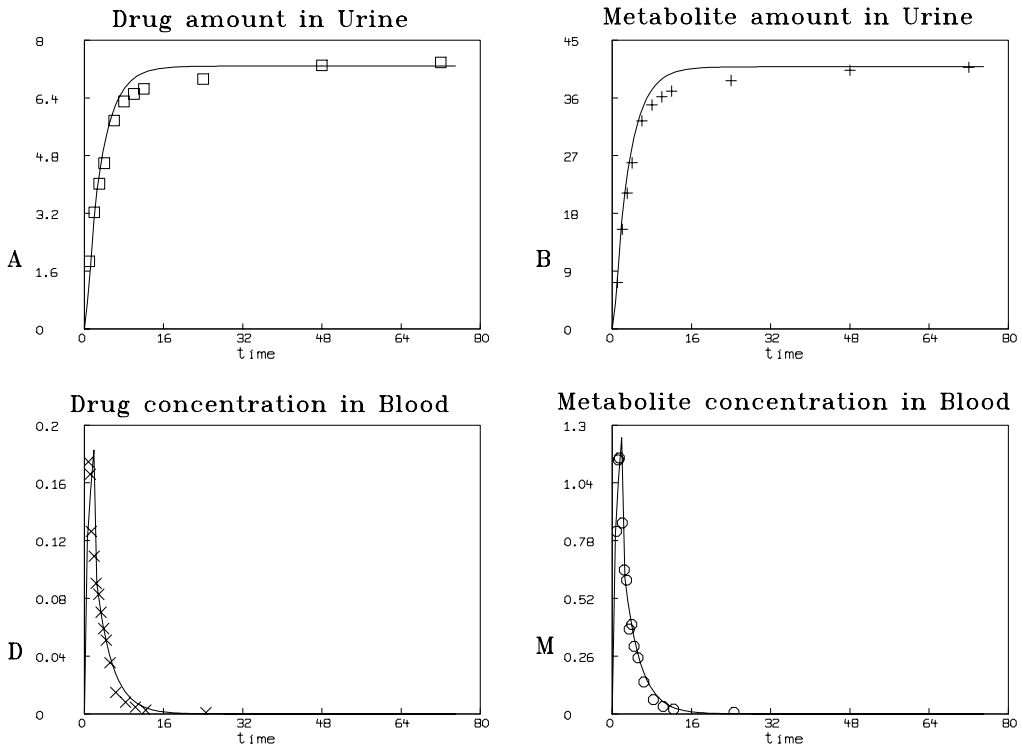
```
draw points(a,tv) color purple
draw av color red pt square lt none
image color grey; frame color brown
top title " Drug amount in Urine" font 11 size .03
left title "'-90AA" font 11 size .03
bottom title "time"
frame 0 to .5, .5 to 1
w3=w

draw points(b,tv) color brown
draw bv color blue pt crosspt lt none
image color aqua; frame color red
top title " Metabolite amount in Urine" font 11 size .03
left title "-90AB" font 11 size .03
bottom title "time"
window 0 to 80, 0 to 45
frame .5 to 1, .5 to 1
w4=w

view
```



This is not the only reasonable fit. Starting from other guesses, for example: $K_1 = 223$, $K_2 = 14.7$, $K_C = 66$, $K_D = 11.7$, $K_M = 9.7$, $E_T = 1.95$, $V_B = 6.24$, $V_S = 12.35$, and $V_M = 0.04$, we can obtain other, quite different, results. The large dependency values of the parameters indicate that this problem does not have a unique answer. Probably the problem is over-parameterized.

```
k1 = 223; k2 = 14.7
kd = 11.7; km = 9.7; kc = 66
vb = 6.24; vs = 12.35; vm =.04; et = 1.95;

fit(k1,k2,kc,kd,km,et,vb,vs,vm), \
: d to dv with weight wd, m to mv with weight wm, \
: a to av with weight wa, b to bv with weight wb, constraints c

final parameter values
      value               error                dependency    parameter
    222.3296966         27.76132266          0.8337053755   K1
     14.7359648         3117.341849          0.9999999383   K2
    65.98333895         3.836887237          0.9399933157   KC
     11.7407663         0.692533352           0.938315949   KD
    9.697523939        0.6754487655          0.6928345824   KM
    1.960006851         0.059767016          0.9654962599   ET
    5.964737228         3.990424231           0.866044195   VB
    12.31946483         2605.653695          0.9999999384   VS
  0.04208395842       0.01012174536          0.9703620604   VM
3 iterations
CONVERGED
best weighted sum of squares = 4.491649e+01
weighted root mean square error = 1.022042e+00
weighted deviation fraction = 2.993774e-02
R squared = 9.941458e-01
no active constraints
```

The graphical results of this fit are shown below. Note we fit the blood-drug and blood-metabolite concentrations more closely at the expense of urine-drug and urine-metabolite fitting.



The existence of multiple local minima corresponding to different parameter values is further indication that our model is not physically accurate. Thus, we should choose our parameter values so that the four curves are adequately predicted without concern for the physical meanings of the parameters.

15 A Pharmacological Model with Delay (Ernest Feytmans)

This example shows how to formulate and fit a compartmental model. It also shows how MLAB can handle a delay term in a differential equation. Finally, the `EWT` function is exemplified.

At time 0, a dose of radioactive-labeled taurocholate, Y, is injected in a rat. This material, Y, is passed through the liver and enters the bile. The bile is collected from the bile duct and the cumulative amount of Y present is measured at various times over a period of 1202 seconds. The level of Y seen in the bile at time t is actually the amount we measure in the output of a catheter at time $t+k$, since a delay of time k is needed for the bile to travel through the catheter. Also each blood level measurement involves removing a small sample of blood for analysis. Thus a simple model for the incorporation of Y in the bile which includes the effects of measurement takes the following compartmental form.

$Y1(t)$ is the concentration of Y in the blood at time t. $Y2(t)$ is the amount of Y in the liver at time t. $Y3(t)$ is the cumulative amount of Y in the collected bile at time t. V is the volume of blood in milliliters. U is the volume of the liver compartment in milliliters. k is the time-delay of the flow of bile from the liver into the bile-collection vial. We have:

$$\begin{aligned} Y_1'(t) &= (h_{21}Y_2/U - (h_{10}+h_{12})Y_1)/V, \\ Y_2'(t) &= h_{12}Y_1 - (h_{20}+h_{21})Y_2/U, \\ Y_3'(t) &= h_{20}Y_2(t-k)/U, \end{aligned}$$

with $Y_1(0) = 100/V$, $Y_2(0) = 0$, and $Y_3(0) = 0$.

The initial value $100/V$ for $Y_1(0)$ denotes the concentration of 100 percent of the dose of Y, and all measurements are given in terms of percent of the initial dose. Y_1 is in units of percent-of-dose per V milliliters and Y_2 and Y_3 are in percent-of-dose units.

Let $k_{21} = h_{21}/U$, $k_{12} = h_{12}/V$, $k_{20} = h_{20}/U$, and $k_{10} = h_{10}/V$. Then we have

$$Y_1'(t) = k_{21}Y_2/V - (k_{10}+k_{12})Y_1,$$

$$\begin{aligned} Y_2'(t) &= k_{12}V \cdot Y_1 - (k_{20} + k_{21})Y_2, \\ Y_3'(t) &= k_{20}Y_2(t-k), \end{aligned}$$

with $Y_1(0) = 100/V$, $Y_2(0) = 0$, and $Y_3(0) = 0$.

We may guess the following values (these are fairly good guesses.)

$V \approx 15\,\text{ml}$; $k \approx 1.7\,\text{hsec}$; $k_{12} \approx 1.7\,\text{hsec}^{-1}$; $k_{21} \approx .1\,\text{hsec}^{-1}$; $k_{10} \approx .25\,\text{hsec}^{-1}$; $k_{20} \approx .5\,\text{hsec}^{-1}$;

The time units used are hectoseconds, denoted hsec; one hsec is 100 seconds. The data below is given in units of hectoseconds.

The blood data is in a text-file called $D1$, as follows.

time	Y_1
.2	3.972
.3	3.484
.4	2.928
.5	2.317
.6	1.988
1	.2 .481
1	.8 .321
2	.4 .217
3	.6 .146
4	.8 .086
6	.0 .066
7	.2 .047
9	.0 .033
12	.0 .047

The bile data is in a text-file called $D2$, as follows.

t	Y_3	t	Y_3	t	Y_3
.35	0	.60	0	.90	0
1.17	.004	1.44	.114	1.78	1.095
2.05	2.248	2.32	4.689	2.62	6.542
2.91	7.652	3.12	5.937	3.40	6.580
3.60	4.687	3.85	6.663	4.20	4.587
4.44	3.687	4.77	3.215	5.06	3.378
5.30	2.458	5.60	2.679	5.84	1.854
6.10	2.059	6.42	1.691	6.63	1.181
6.97	1.648	7.25	1.235	7.56	1.183
7.83	.955	8.12	.976	8.37	.706
8.60	.498	8.82	.665	9.13	.792
9.32	.395	9.55	.462	9.80	.470
10.00	.478	10.27	.407	10.60	.426
10.84	.285	11.15	.438	11.46	.396
11.74	.273	12.02	.318		

The bile data in $D2$ is the incremental percentage-of-dose amounts of Y found in drops of bile collected at the specified times. To obtain cumulative bile measurements we must compute the successive partial sums of the given values.

We wish to fit our model to the data by estimating values for k_{12}, k_{21}, k_{20}, k_{10}, V, and k.

We may run MLAB and proceed as follows.

```
* M1 = READ(D1,200,2)
* M2 = READ(D2,200,2)
```

First we must compute the partial sums of the incremental bile data.

```
* FUNCTION PS(j) = (IF j=1 THEN 0 ELSE PS) + M2[j,2]
* M2 COL 2 = PS ON 1:NROWS(M2)
```

In general, the weight associated with a given observation should be the reciprocal of the variance of the random variable of which the observation is a sample. When using correct weights, we not only account for differing amounts of error, but also automatically compensate for differing scales which may be used in distinct sets of data being fit simultaneously by model functions with shared parameters. We can estimate the standard deviations, and hence the desired weight values, non-parametrically, using the `EWT` operator.

```
* W1 = EWT(M1)
* W2 = EWT(M2)
```

Now we normalize both $W1$ and $W2$ to sum to 1/2, since we wish to give each curve the same total weight.

```
* W1 = W1/ROWSUM(W1)/2
* W2 = W2/ROWSUM(W2)/2
```

Now $M1$ and $M2$ are our data matrices for the blood and cummulative bile data respectively, and $W1$ and $W2$ are the associated weight vectors. The differential equations which define the model functions Y_1, Y_2, and Y_3 are given below. Note one differential equation contains a delay term.

```
* FUNCTION Y1'T(T) = K21*Y2/V - (K10 + K12)*Y1
* FUNCTION Y2'T(T) = K12*V*Y1 - (K20 + K21)*Y2
* FUNCTION Y3'T(T) = K20*Y2(T-K)
* INITIAL Y1(0)= 100/V
* INITIAL Y2(0)= 0
* INITIAL Y3(0)= 0
* V=15; K=1.7; K12=1.7; K21=.1; K10=.25; K20=.5
```

Now it is wise to save our data and model in an MLAB save-file for later reuse.

```
* SAVE IN FEYT
```

At this point we may start fitting our model. However, first let us consider what the delay term implies. In MLAB, delay terms in differential equations are evaluated during the numerical solution process by looking back in the table of previously-computed results and obtaining a value for the delay term by interpolation. If the previously-generated results do not extend far enough into the past, the earliest available value is used! In order for this to be the initial value, it is necessary for the initial time to be present in the vector of time values at which we are maintaining results. It is also necessary that the time vector consist of sufficiently closely-spaced values so that the interpolation process is adequately accurate. In our case, our time values are reasonably closely-spaced, but, we must add the initial time value (with weight 0) as follows.

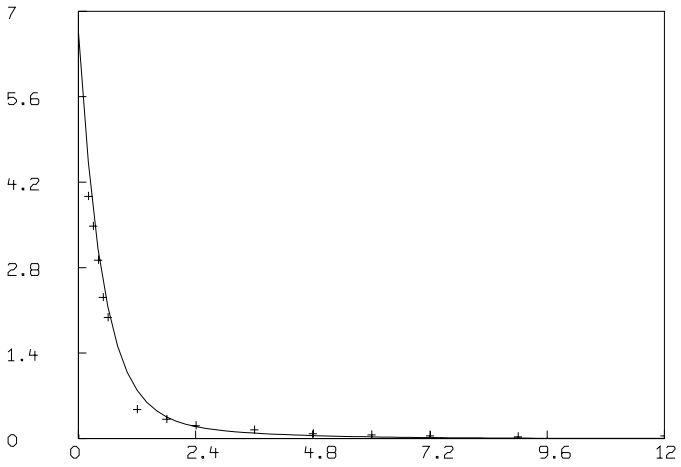
```
* M1 = 0&M1
* M2 = 0&M2
* W1 = 0&W1
* W2 = 0&W2
```

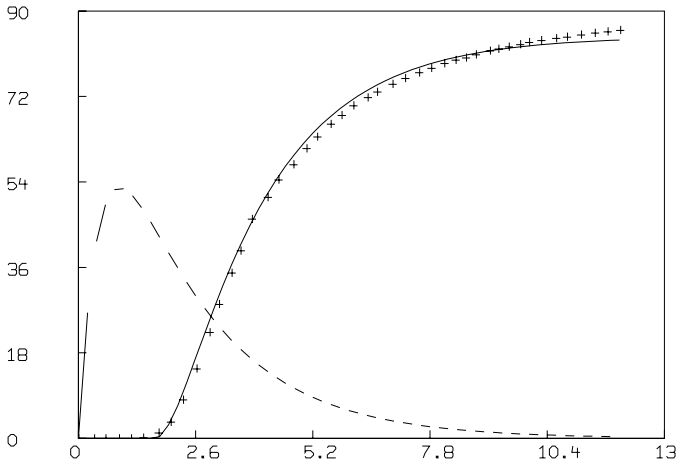
In MLAB, the delay expression $Y2(t-k)$ is effectively treated as $Y2(\text{if } t < k \text{ then } 0 \text{ else } t-k)$. Thus the delay enters gradually. We start with zero delay, and the delay increases until time $t = k$, whereupon the delay remains constant. This interpretation of a delay is forced by MLAB, but it is often the appropriate way to handle delays, given initial conditions only.

Before fitting, it is a good idea to check the model and the quality of our guesses.

```
* R=INTEGRATE (Y1'T, Y2'T, Y3'T, 0:12:.2)
t = 1.7, errfac = 0.001, eqn # 3, truncation error = 111.111
tolerance will be increased in order to proceed with the deq solution.
t = 1.7, errfac = 0.01, eqn # 3, truncation error = 11.1111
tolerance will be increased in order to proceed with the deq solution.
solution of deq system forced past a possible singularity near 1.7.
accuracy may be lost
dy[1]: -0.486184
dy[2]: -16.1942
dy[3]: 0
* DRAW M1, LINE NONE PT "+"
* DRAW R COL 1:2
* VIEW
* DELETE W
* DRAW R COL (1,4) LINETYPE ALTERNATE
* DRAW M2, LINETYPE NONE POINTTYPE "o"
* DRAW R COL (1,6)
* VIEW
* DELETE W
```

These graphs are shown below. Note how the delay effects $Y3$; it remains zero even when $Y2$ is positive.





The tolerance violations which occurred during the integration above are harmless. What happened is that for the chosen value of k, the particular time points where the differential-equation-solver obtained solution values included a time, t_1, which was very close to k. When a step, s, from t_1 to $t_1 + s$ was attempted, the value of $Y2(t - k)$, and hence of $Y3't$, jumped from zero to a positive value. The change in a derivative value must be sufficiently gentle or the differential-equation-solver will reduce the step-size s until it is! But in this case, t_1 was so close to k that $t_1 + s$ could not achieve a value at which $Y3't$ was sufficiently close to zero without s being too

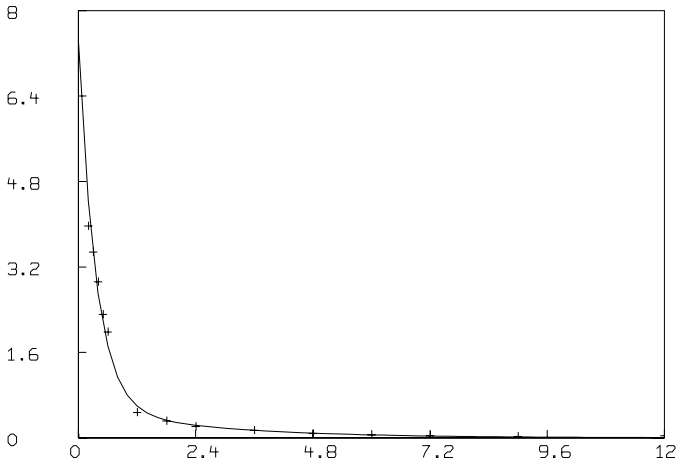
small! The tolerance violation is complaining of this fact. The effect of a tolerance violation is to advance t and assume the current value of $Y1$, $Y2$, and $Y3$ hold at the new time. This introduces an error which we hope is small and whose effect dies out over time.

Now we may fit our data. In order to see the progress of the curve-fitting process, we set the MLAB control variable `lsqrpt` to 9. Note the parameter V occurs in the initial condition for $Y1$ as well as in the derivative functions. In order to avoid tolerance violation messages, we shall set the MLAB control variable disastersw to -2.

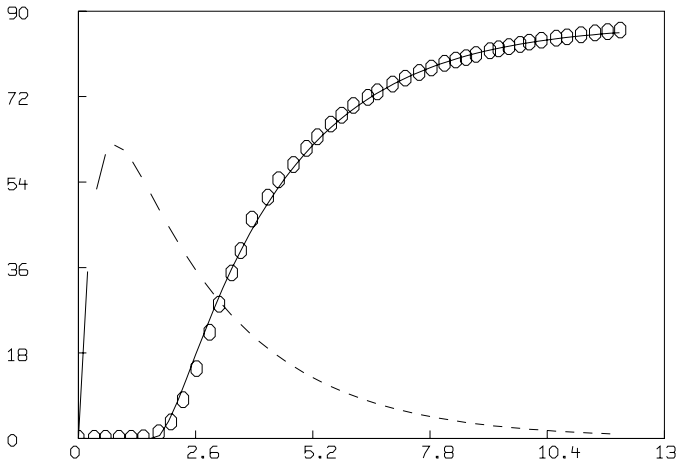
```
* lsqrpt = 9
* disastersw = -2
* FIT (K12, K21, K20, K10, K, V), Y1 TO M1 WITH WEIGHT W1,Y3 TO M2 WITH WEIGHT W2
Begin iteration 1 bestsosq=6.06291e-01
Begin iteration 2 bestsosq=3.37174e-01
Begin iteration 3 bestsosq=2.80705e-01
Begin iteration 4 bestsosq=6.13384e-02
Begin iteration 5 bestsosq=6.01915e-02
Begin iteration 6 bestsosq=6.01306e-02
Begin iteration 7 bestsosq=6.00476e-02
Begin iteration 8 bestsosq=5.95795e-02
Begin iteration 9 bestsosq=5.94568e-02
Begin iteration 10 bestsosq=5.91193e-02
Begin iteration 11 bestsosq=5.41983e-02
Begin iteration 12 bestsosq=5.39791e-02
Begin iteration 13 bestsosq=5.31954e-02
final parameter values
      value               error                dependency    parameter
    2.404830506        0.5089934386          0.9996545161   K12
   0.1788429902       0.03364502419          0.9984644263   K21
   0.4046451617       0.01567822569           0.995752869   K20
   0.2352243999         0.050102745          0.9988267993   K10
    1.667994505       0.06660192075          0.9643014847   K
    13.44151307         2.411338814           0.855048444   V
13 iterations
CONVERGED
best weighted sum of squares = 5.31954e-02
weighted root mean square error = 3.13863e-02
weighted deviation fraction = 2.74048e-03
R squared = 9.98708e-01
```

We may observe the result below.

```
* R = INTEGRATE(Y1'T,Y2'T,Y3'T,0:12:.2)
* DRAW M1,LINETYPE 0,POINTTYPE "+"
* DRAW R COL 1:2; VIEW
```




```
* DELETE W
* DRAW M2,LINETYPE NONE POINTTYPE "o"
* DRAW R COL (1,4),LINETYPE ALTERNATE
* DRAW R COL (1,6); VIEW;
```



```
* DELETE W
```

This is a good fit. In this case we did not need to impose constraints to keep the parameter values within reasonable limits, but such constraints are usually necessary. We can get a slightly better fit by using the explicit formulas for $Y1$ and $Y3$, which are obtainable in this case since $Y1$ and $Y2$ do not depend on $Y3$.

The explicit solution functions are given below. It is important to use the gradual delay expression (if $t < k$ then t else k) in solving the differential equations. Otherwise the solutions will not correspond to the results obtained by numerical integration.

$$\begin{aligned} Y_1(t) &= A_1 \exp(-A_2 t) + B_1 exp(-B_2 t), \\ Y_2(t) &= A_3[\exp(-A_2 t) - \exp(-B_2 t)], \quad \text{and} \\ Y_3(t) &= \text{if } t < k \text{ then } 0 \text{ else } A_4[1 - \exp(-A_2(t-k))] + A_5[1 - \exp(-B_2(t-k))], \end{aligned}$$

where

$$\begin{aligned} A_2 &= (S+Q)/2, \\ B_2 &= (S-Q)/2, \\ S &= K_{12} + K_{21} + K_{20} + K_{10}, \\ Q &= [S_2 - 4(K_{12}K_{20} + K_{10}K_{21} + K_{10}K_{20})]^{1/2}, \\ A_1 &= (100/V)[A_2 - K_{21} - K_{20}]/(A_2 - B_2), \\ B_1 &= (100/V)[K_{21} + K_{20} - B_2]/(A_2 - B_2), \\ A_3 &= 100K_{20}K_{12}/(B_2 - A_2), \\ A_4 &= 100K_{20}K_{12}/[A_2(B_2 - A_2)], \quad \text{and} \\ A_5 &= 100K_{20}K_{12}/[B_2(A_2 - B_2)]. \end{aligned}$$

Now, an important point arises. How are these functions to be defined in MLAB? The straightforward process of substituting to eliminate the auxilary variables A_1, A_2, A_3, A_4, A_5, B_1, and B_2 results in huge formulas, which, when defined in MLAB would cause the required symbolic derivative functions $Y1'K_{12}$, $Y1'K_{21}$, ..., etc. to be so large they might not all fit in memory!

One approach to representing these functions is to make A_1, A_2, A_3, A_4, A_5, B_1, B_2, S, and Q functions of no arguments which have the appropriate parameters merely by virtue of their appearence in the definitions. Thus we could type:

```
FUNCTION Y1(T) = A1()*EXP(-A2()*T)+B1()*EXP(-B2()*T)
FUNCTION A1() = (100/V)*(A2()-K12-K20)/(A2()-B2())
FUNCTION A2() = (S() + Q())/2
FUNCTION S() = K12 + K21 + K20 + K10 ..., etc.
```

This process results in functions which waste much time computing the same values many times during one invocation. For example computing $Y1(2)$ involves computing $A1()$, $A2()$, $B1()$, and $B2()$; but computing $A1()$ also involves computing $A2()$ and $B2()$ (again!) and so on.

A better approach is to discover common sub-expressions, and when they occur, make them actual arguments to be passed to a function which returns the value of the formula in which they occur. This process results in the following formulation for defining our model. This example merits careful study since the principle involved should usually be employed in using MLAB!

```
* DELETE Y1, Y2, Y3, Y1'T, Y2'T, Y3'T
* FUNCTION F3(J,T,A2,B2) = (100/(A2-B2))*(IF J=1 THEN \
          ((A2-K21-K20)*EXP(A2*T)+(K21+K20-B2)*EXP(B2*T))/V \
          ELSE K20*K12*(IF J=2 THEN (EXP(B2*T)-EXP(A2*T)) ELSE \
          ((1-EXP(B2*T))/B2-(1-EXP(A2*T))/A2)))
* FUNCTION F2(J,T,S,Q) = F3(J, -T, .5*(S+Q), .5*(S-Q))
* FUNCTION F1(J,T,S) = F2(J,T,S,SQRT(S*S-4*(K12*K20+K10*K21+K10*K20)))
* FUNCTION Y1(T) = F1(1, T, K12+K21+K10+K20)
* FUNCTION Y2(T) = F1(2, T, K12+K21+K10+K20)
* FUNCTION Y3(T) = IF T<K THEN 0 ELSE F1(3, T-K, K12+K21+K10+K20)
```

Now given these functions, you may see the derivative forms which result by typing them out; although they are very large, they are easily handled within MLAB. For example:

```
* TYPE Y1'K12
FUNCTION Y1'K12(T) = F1'K12(1,T,K12+K21+K10+K20)+ \
      F1'S(1,T,K12+K21+K10+K20)
* TYPE F1'S
FUNCTION F1'S(J,T,S) = F2' S(J,T,S, \
       SQRT(S*S-4*(K12*K20+K10*K21+K10*K20))) \
       +F2' Q(J,T,S,SQRT(S*S-4*(K12*K20+K10*K21+K10*K20))) \
       *((S+S)/(2*SQRT(S*S-4*(K12*K20+K10*K21+K10*K20))))
* TYPE F2'S
 FUNCTION F2'S(J,T,S,Q) = F3'A2(J,-T,.5*(S+Q),.5*(S-Q))*.5 \
        +F3'B2(J,-T,.5*(S+Q),.5*(S-Q))*.5
* TYPE F3'A2
 FUNCTION F3 DIFF A2(J,T,A2,B2) = (100/(A2-B2))*( IF J=1 \
       THEN (((A2-K21)-K20)*T*EXP(A2*T)+EXP(A2*T))/V \
       ELSE K20*K12*( IF J=2 THEN -T*EXP(A2*T)  \
       ELSE -((-T*EXP(A2*T))/A2+(1-EXP(A2*T))*(-A2^(-2)))))  \
       +( IF J=1 THEN (((A2-K21)-K20)*EXP(A2*T)+((K21+K20)-B2)*EXP(B2*T))/V \
       ELSE K20*K12*( IF J=2 THEN EXP(B2*T)-EXP(A2*T) \
       ELSE (1-EXP(B2*T))/B2-(1-EXP(A2*T))/A2))*100*(-(A2-B2)^(-2))
```

Now to do our fit, we may use the following MLAB fit-statement.

```
FIT(K12, K21, K20, K10, K), Y1 TO M1 WITH WEIGHT W1,Y3 TO M2 WITH WEIGHT W2
Begin iteration 1 bestsosq=1.03081e-01
Begin iteration 2 bestsosq=6.66487e-02
Begin iteration 3 bestsosq=6.45033e-02
Begin iteration 4 bestsosq=6.39085e-02
final parameter values
      value               error                dependency    parameter
    2.486325129        0.2919415767          0.9986809093   K12
   0.2181076867         0.175390433          0.9997631512   K21
   0.4060674476       0.01517223395          0.9949641321   K20
   0.2362370486       0.04912624607          0.9993555185   K10
    1.636370209       0.05358085176          0.9369202287   K
4 iterations
CONVERGED
best weighted sum of squares = 6.38532e-02
weighted root mean square error = 3.40730e-02
weighted deviation fraction = 2.94842e-03
R squared = 9.98282e-01
```

We can do even better if we account for the laminar flow in the catheter which is sampling the bile. If this tube is of length L with interior diameter $2R$, then

$$Y_3(t) = \text{if } t = 0 \text{ then } 0 \text{ else } (1/t) \int_0^t g(u)\, du,$$

where $g(u)$ is the concentration of labeled taurocholate entering compartment Y_3 at time u. This is just an average of earlier concentrations which have entered the catheter; the following function g expresses the delayed laminar flow and we have:

$$g(u) = \int_0^R Y_2(u - L/(a(R-x)^2)) \cdot (2x/R^2)\, dx,$$

where $Y2(t) = 0$ for $t < 0$, and a is defined by the equation

$$t \int_0^R a(R-z)^2 2\pi x\, dx - \pi R^2 L = V_t,$$

where V_t is the total volume of fluid accumulated in t seconds of flow through the catheter, with to be completely filled.

We can thus reformulate our model using the MLAB integral operator:

```
fct Y3(T) = if T=0 then 0 else \
     INTEGRAL(U,0,T,INTEGRAL(X,0,R,Y2(U-L/(A*(R-X)^2))*(2*X/R^2)))/T
```

where R and L are known values, and A is computed initially (before fitting) using a function which employs the MLAB `ROOT` operator.

16 An Oscillating Chemical Reaction

The possibility of oscillations in chemical kinetics depends upon the existence of sources and sinks for catalytic intermediates with very different time scales, resulting in very wide swings in their concentrations. The Belusov-Zhabotinsky reaction is perhaps the best known of the oscillatory chemical reactions. The following example is a gross simplification of the kinetics of the BZ reaction, retaining, however, the main features and properties.

In this example, there are five chemical reactions involving the chemicals y_1, y_2, y_3, y_4, y_5, and y_6.

$$y_1 + y_3 \xrightarrow{k_1} y_2 \quad y_2 + y_3 \xrightarrow{k_2} y_5 \quad y_1 + y_2 \xrightarrow{k_3} y_4 + 2y_2 \quad y_2 + y_2 \xrightarrow{k_4} y_6 \quad y_4 \xrightarrow{k_5} y_3$$

In these reactions, the rate constants k_1, k_2, k_3, k_4, and k_5 take on the following values.

$$k_1 = 1.34, \quad k_2 = 1.6 \cdot 10^9, \quad k_3 = 8000, \quad k_4 = 8 \cdot 10^7, \quad \text{and} \quad k_5 = 1.$$

The disposition of the products y_5 and y_6 of the second and fourth reactions is ignored. The initial concentrations of the four remaining chemicals are:

$$y_1(0) = .06, \quad y_2(0) = 5.01 \cdot 10^{-11}, \quad y_3(0) = 3.3 \cdot 10^{-7}, \quad y_4(0) = 2.4 \cdot 10^{-8}.$$

These reactions imply the following system of differential equations, (ignoring the products y_5 and y_6).

$$\begin{aligned} dy_1/dt &= -y_1(k_1 y_3 + k_3 y_2) \\ dy_2/dt &= -y_2(k_2 y_3 - k_3 y_1 + k_4 y_2) + k_1 y_1 y_3 \\ dy_3/dt &= -y_3(k_1 y_1 + k_2 y_2) + k_5 y_4 \\ dy_4/dt &= -k_5 y_4 + k_3 y_1 y_2 \end{aligned}$$

These equations, initial conditions, and rate constants are defined by the MLAB statements below.

```
k1 = 1.34; k2 = 1.6e9; k3 = 8e3; k4 = 8e7; k5 = 1
init y1(0) = .06; init y2(0) = 5.01e-11; init y3(0) = 3.3e-7; init y4(0) = 2.4e-8
fct y1't(t) = -y1*(k1*y3+k3*y2)
fct y2't(t) = -y2*(k2*y3-k3*y1+k4*y2)+k1*y1*y3
fct y3't(t) = -y3*(k1*y1+k2*y2)+k5*y4
fct y4't(t) = -k5*y4+k3*y1*y2
```

Because of the wide variation in the rate constants, these equations are very "stiff" in some regions. They are also oscillatory.

There are two other properties of these equations that make them difficult to solve numerically. First, there are very sharp "spikes" in y_2 and y_3; if the stepsize is not sufficiently small, the solver can skip over these spikes without "seeing" them. Thus we must either set `errfac` to a sufficiently small value to cause the stepsize used to be appropriately small or we must explicitly choose where the integrator should "land" and specify these times to be so-called "mandatory" points by setting `mandsw` to 1, or both. Second, these equations are unstable with respect to the numerical methods employed to solve them in certain regions of their common period. (This means that the eigenvalues of the Jacobian matrix of the linearized system are positive at some points during the solution time course and this implies "explosive" error is injected in the solution at these points.) This episodic instability can only be combatted by using a method with the largest-possible region of stability, together with a small step-size. This means we should use the Gear-Shrager (gear2) method with a small step-size as produced by choosing a sufficiently-small value for errfac.

```
method = gear2; errfac = 1e-8
```

This system has an oscillatory period of about 50 time units, so the integration period specified below in the list t includes that interval. MLAB generates its integration output only at the time-points that are specified by the user. Such values are generated by interpolation at the user-specified time points, unless mandatory points are specified, in which case, values are computed exactly at the specified points. Thus, in order to "catch" the very steep spikes in the output matrix, we must specify output time points during the spikes; otherwise, the sharp spikes would not be present in the output matrix, even though the integrator generated them correctly. Appropriate points to catch the spikes for the parameter values used in the differential equations above are listed in the 1-column matrix t below.

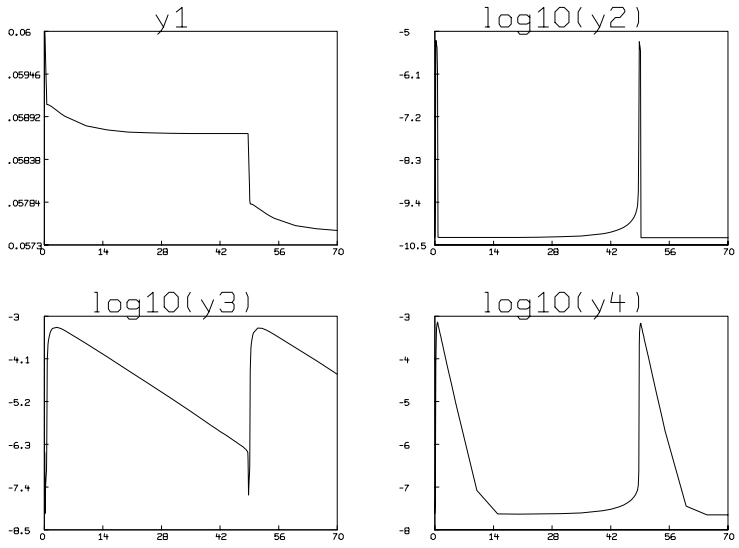
```
t =list(0:.9:.1,1:1.8:.2,2:5,10:40:5,41:46,46.5:48:.5,48.1:50:.1,50:55,60:70:5)
```

The solution is obtained using the integrate function.

```
m = integrate(y1't,y2't,y3't,y4't,t)
```

The solutions for y_1, y_2, y_3, and y_4 are in columns 2, 4, 6, and 8 of `m`, respectively. Because of the wide range of concentrations assumed in this problem, we take the base-10 logarithm of the y_2, y_3, and y_4 values before plotting.

```
l2 = log10 on m col 4; l3 = log10 on m col 6; l4 = log10 on m col 8

draw m col (1:2); top title "y1" size .05; frame 0 to .5, .5 to 1; w1=w
draw t&'l2; top title "log10(y2)" size .05; frame .5 to 1, .5 to 1; w2 = w
draw t&'l3; top title "log10(y3)" size .05; frame 0 to .5, 0 to .5; w3 = w
draw t&'l4; top title "log10(y4)" size .05; frame .5 to 1, 0 to .5; w4 = w
view
```

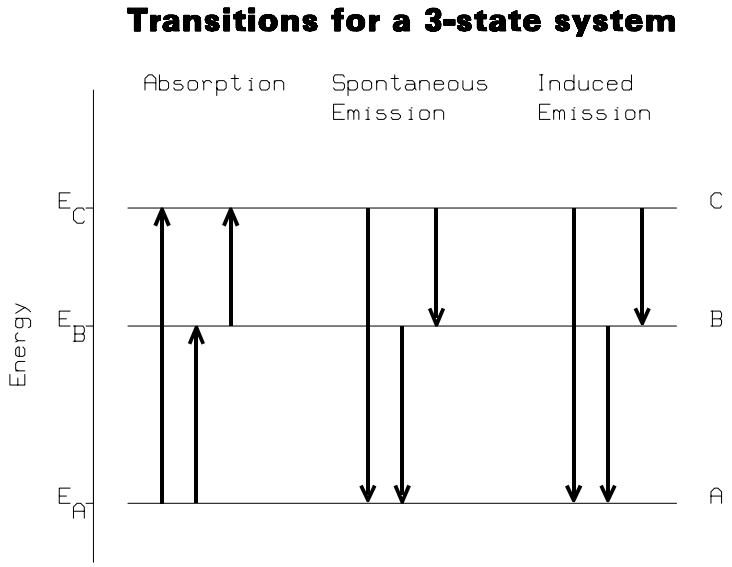


17 Two Chemical Kinetics Models of a Laser

Since the 1950's, many devices have been discovered which produce coherent, monochromatic, electromagnetic radiation by irradiating a sample of solid, liquid, or gas that is contained in a chamber having dimensions equal to some multiple of the radiation's wavelength. Known as *MASER*s or *LASER*s, these devices can be modelled in terms of chemical reactions where photons, atoms, and molecules occupying different energy states are the reactants and products. In this paper we use the MLAB mathematical modelling program to develop chemical kinetics models of systems that amplify electromagnetic radiation by stimulated emission.

17.1 The Three State Model

This section refers to a system of atoms which can undergo transitions between three states. Figure 1 shows the energy levels of the three-state atom, and the types of transitions that can occur between them. The states, in order of increasing energy, are designated A, B, and C. The energies of the states are designated E_A, E_B, and E_C, respectively.



Three types of reactions are seen to occur: absorption reactions, spontaneous emission reactions, and induced (also known as stimulated) emission reactions. The three spontaneous emission

reactions are:

$$C \xrightarrow{K^s_{CA}} A + \omega_{CA}$$

$$B \xrightarrow{K^s_{BA}} A + \omega_{BA}$$

$$C \xrightarrow{K^s_{CB}} B + \omega_{CB}$$

The three absorption reactions are:

$$A + \omega_{CA} \xrightarrow{K^a_{AC}} C$$

$$A + \omega_{BA} \xrightarrow{K^a_{AB}} B$$

$$B + \omega_{CB} \xrightarrow{K^a_{BC}} C$$

The three induced emission reactions are:

$$C + \omega_{CA} \xrightarrow{K^i_{CA}} A + 2\omega_{CA}$$

$$C + \omega_{CB} \xrightarrow{K^i_{CB}} B + 2\omega_{CB}$$

$$B + \omega_{BA} \xrightarrow{K^i_{BA}} A + 2\omega_{BA}$$

In these reactions, ω represents a photon; its subscript specifies its frequency. For example, ω_{CA} is a photon with frequency $(E_C - E_A)/\hbar$ where $\hbar$ is Planck's constant. We use K to represent the rate constant for the reaction. The superscripts a, s, and i on K identify whether the rate constant is for an absorption, spontaneous emission, or induced emission reaction, respectively. The subscripts on K identify the reactant and product states involved in the reaction. For example K^i_{BA} is the rate constant for the induced emission reaction in which an atom in state B reacts with a photon to produce an atom in state A and two photons.

In a collection of N three-state atoms at equilibrium, the number of atoms in the j^{th} atomic state is given by the Boltzmann distribution,

$$N_j = N \frac{\mathrm{e}^{-E_j/kT}}{\sum_{i=A}^{C} \mathrm{e}^{-E_i/kT}}$$

where j is the state label A, B, or C; T is the temperature in degrees Kelvin; N is the total number of atoms in all states; and k is Boltzmann's constant. At room temperature and with the energies we are considering here, the populations of states B and C are negligible when the system is at equilibrium.

In order to generate monochromatic, electromagnetic radiation of frequency ω_{BC} by stimulated emission, the collection of three-state atoms must get energy in the form of photons of frequency $(E_C - E_A)/\hbar$. This *pumping* of the system disrupts the equilibrium distribution of states and creates a *population inversion*; the number of atoms in state C becomes greater than the number of atoms in state B. When this condition exists, the presence of a photon of frequency ω_{BC} catalyzes the the transition from C to B and produces another coherent photon. If the population inversion can be maintained, significant numbers of coherent photons can be produced.

Spontaneous emission reactions follow rate laws that depend only on the number of atoms in the reactant state. Absorption and induced emission reactions follow rate laws that depend on both the number of atoms in the reactant state and the number of reactant photons. From these rules and the nine reactions listed above, we can write six coupled chemical rate equations; one for the population of each of the three states and one for the number of photons at each of the three different frequencies. Using $N_A(t)$, $N_B(t)$, and $N_C(t)$ as the populations of states A, B, and C, respectively, and $N_{AB}(t)$, $N_{BC}(t)$, and $N_{AC}(t)$ as the number of photons having frequency $(E_B - E_A)/\hbar$, frequency $(E_C - E_B)/\hbar$, and frequency $(E_C - E_A)/\hbar$, respectively, the equations are:

$$\begin{aligned}
\frac{dN_A}{dt} &= -K^a_{AC} N_{AC}(t) N_A(t) - K^a_{AB} N_{AB}(t) N_A(t) + K^s_{CA} N_C(t) + K^s_{BA} N_B(t) + \\
&\quad K^i_{CA} N_{AC}(t) N_C(t) + K^i_{BA} N_{AB}(t) N_B(t) \\
\frac{dN_B}{dt} &= -K^a_{BC} N_{BC}(t) N_B(t) + K^a_{AB} N_{AB}(t) N_A(t) + K^s_{CB} N_C(t) - K^s_{BA} N_B(t) + \\
&\quad K^i_{CB} N_{BC}(t) N_C(t) - K^i_{BA} N_{AB}(t) N_B(t) \\
\frac{dN_C}{dt} &= -\frac{dN_A}{dt} - \frac{dN_B}{dt} \\
\frac{dN_{AB}}{dt} &= -k^a_{AB} N_{AB}(t) N_A(t) + k^s_{BA} N_B(t) + k^i_{BA} N_{AB}(t) N_B(t) - k^d_{AB} N_{AB}(t) \\
\frac{dN_{AC}}{dt} &= pump_{AC}(t) - k^a_{AC} N_{AC}(t) N_A(t) + k^s_{CA} N_C(t) + k^i_{CA} N_{AC}(t) N_C(t) - k^d_{AC} N_{AC}(t)
\end{aligned}$$

$$\frac{dN_{BC}}{dt} = -k^a_{BC}N_{BC}(t)N_B(t) + k^s_{CB}N_C(t) + k^i_{CB}N_{BC}(t)N_C(t) - k^d_{BC}N_{BC}(t)$$

The function $pump_{AC}(t)$ represents the number of photons per second that are put into the system to pump the ground state A atoms to the state C. The terms $k^d_{AB}N_{AB}(t)$ and $k^d_{AC}N_{AC}(t)$ represent the number of photons per second that are lost due to destructive interference in the chamber and the term $k^d_{BC}N_{BC}(t)$ represents the number of ω_{BC} photons per second that escape from the chamber.

Using MLAB, we can assign values to the different rate constants, compute the initial Boltzmann distribution of states, define the rate equations, and integrate the rate equations for a continuous wave (CW) laser by executing a script containing the following commands (comments are delimited by `/*` and `*/`):

```
/* assign values for constants */
k = 1.3806E-16   /* Boltzmann's constant in erg/degree Kelvin */
h = 6.6261E-27   /* Planck's constant in erg sec */
EA =  0.1E-11    /* energy of state A in ergs */
EB =  1.05E-11   /* energy of state B in ergs */
EC =  1.06E-11   /* energy of state C in ergs */
N =   1000       /* total number of A,B,C, state atoms */
Np = 10^10       /* total number of photons per second pumped into system */
tmp = 293        /* initial temperature in degrees Kelvin */
kaAB =  1.1e7 /* A->B absorption rate constant in 1/(second*photon) */
kaBC =  1.1e7 /* B->C absorption rate constant in 1/(second*photon) */
kaAC =  1.1e7 /* A->C absorption rate constant in 1/(second*photon) */
ksCA = 1.1e7  /* C->A spontaneous emission rate constant in 1/second */
ksCB = 1.1e7  /* C->B spontaneous emission rate constant in 1/second */
ksBA = 1.1e7  /* B->A spontaneous emission rate constant in 1/second */
kiCA = .1e7   /* C->A induced emission rate constant in 1/(second*atom) */
kiCB = .2e7   /* C->B induced emission rate constant in 1/(second*atom) */
kiBA = .3e7   /* B->A induced emission rate constant in 1/(second*atom) */
kdAB = 1e9    /* B->A photon loss from chamber in 1/second */
kdBC = 1e7    /* C->B photon loss from chamber in 1/second */
kdAC = 1e9    /* C->A photon loss from chamber in 1/second */

/* initial A,B, and C populations from Boltzmann distribution */
fct g(e) = exp(-e/(k*tmp))
bf = g(EA)+g(EB)+g(EC)
init NA(0) = N*g(EA)/bf  /* atoms */
init NB(0) = N*g(EB)/bf  /* atoms */
```

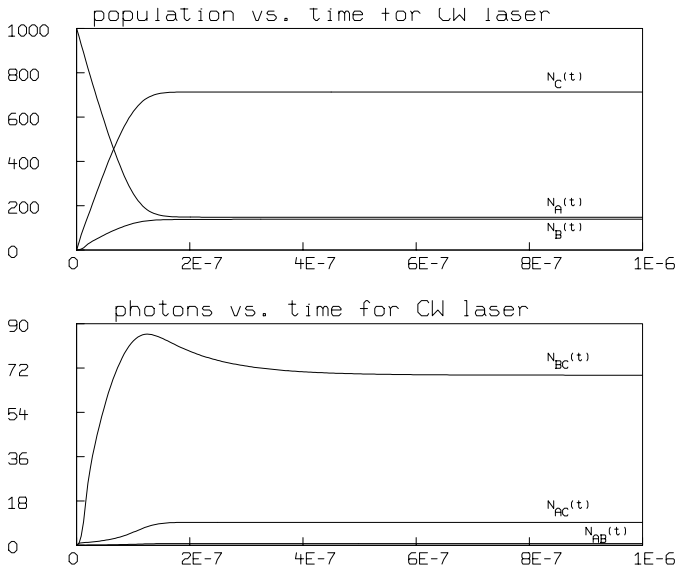
```
init NC(0) = N*g(EC)/bf  /* atoms */

/* initial BA, CB, and CA numbers of photons */
init NAB(0) = 0     /* photons */
init NBC(0) = 0     /* photons */
init NAC(0) = 0     /* photons */

/* photon rate due to pumping A->C transition */
fct pumpAC(t) = Np /* photons/second */

/* rate equations */
fct NA't(t) = -kaAC*NAC(t)*NA(t)-kaAB*NAB(t)*NA(t)+ksCA*NC(t)\
              +ksBA*NB(t)+kiCA*NAC(t)*NC(t)+kiBA*NAB(t)*NB(t)

fct NB't(t) = -kaBC*NBC(t)*NB(t)+kaAB*NAB(t)*NA(t)+ksCB*NC(t)\
              -ksBA*NB(t)-kiBA*NAB(t)*NB(t)+kiCB*NBC(t)*NC(t)

fct NC't(t) = -NA't(t)-NB't(t)

fct NAB't(t) = -kaAB*NAB(t)*NA(t)+ksBA*NB(t)+kiBA*NAB(t)*NB(t)-kdAB*NAB(t)

fct NAC't(t) = pumpAC(t)-kaAC*NAC(t)*NA(t)+ksCA*NC(t)+kiCA*NAC(t)*NC(t)\
               -kdAC*NAC(t)

fct NBC't(t) = -kaBC*NBC(t)*NB(t)+ksCB*NC(t)+kiCB*NBC(t)*NC(t)-kdBC*NBC(t)

/* integrate the kinetics equations for 1 microsecond */
p = integrate(NA,NB,NC,NAC,NBC,NAB,0:.000001!250)
/* columns of p are t NA NA' NB NB' NC NC' NAC NAC' NBC NBC' NAB NAB' */
```

Here the number of atoms, number of photons, and rate constants have been conveniently chosen so that population inversions are achieved in the allotted time scale. The time dependent population levels of each state and the number of photons at each frequency are graphed by the following commands:

```
/* draw the time dependence of the number of A's, B's, and C's */
draw p col 1:2
draw p col (1,4) color green
draw p col (1,6) color red
no framebox
top title "population vs. time for CW laser"
title t1 = "N'1DA'1U(t)" at (.8,.275) ffract size .01
```

```
title t2 = "N'1DB'1U(t)" at (.8,.19) ffract color green size .01
title t3 = "N'1DC'1U(t)" at (.8,.7) ffract color red size .01
frame 0 to 1, .5 to 1
w1=w

draw p col (1,8)
draw p col (1,10) color green
draw p col (1,12) color red
no framebox
top title "photons vs. time for CW laser"
title t5 = "N'1DAC'1U(t)" at (.8,.25) ffract size .01
title t6 = "N'1DBC'1U(t)" at (.8,.75) ffract color green size .01
title t7 = "N'1DAB'1U(t)" at (.85,.16) ffract color red size .01
frame 0 to 1, 0 to .5
view
```

Figure 2 shows the resulting graph.



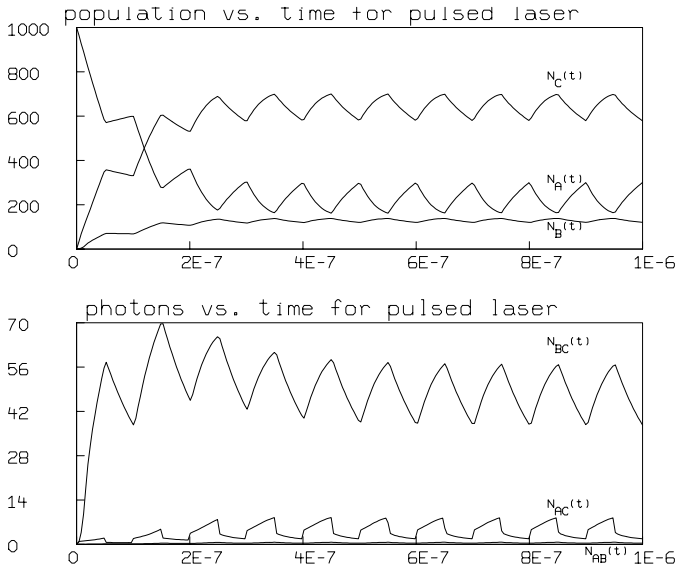
Note that the populations reach steady state after 4×10^{-7} seconds and that the steady state populations are then greater for state C than state B; this is the expected population inversion in

a laser.

The MLAB commands above model a laser that operates in a continuous mode; the pump function is constant in time. We can model a pulsed laser by using the previous MLAB commands but with the following definition of the pumping function:

```
/* define the pulsed input pump function */
fct pumpAC(t) = if (t*10000000-floor(t*10000000) < .5) then 10^10 else 0
```

This effectively modulates the input so that the pump alternates between 0 and 1×10^{10} photons per second 10^7 times a second. The resulting populations and numbers of photons are graphed in Figure 3.



In this figure, the population inversion is seen to be modulated at the frequency of the pump pulses.

An important feature of this three-state model is that one can easily investigate how the population levels change with variations in any of the input constants: N, Np, T, K^a_{AB}, K^a_{BC}, K^a_{AC}, K^s_{CA}, K^s_{CB}, K^s_{BA}, K^i_{CA}, K^i_{CB}, K^i_{BA}, K^d_{AB}, K^d_{BC}, and K^d_{AC}. This makes the model given above an important pedagogical tool.

17.2 The Atomic Hydrogen Laser

Another model is based on the lowest 10 levels of basic hydrogen. By the lowest 10 energy levels of basic hydrogen, we mean the 1s, 2s, 2p, 3s, 3p, 3d, 4s, 4p, 4d, and 4f levels one learns about in beginning and intermediate chemistry and physics courses. Note, this is *not* a hydrogen MASER, which operates on the two hyperfine levels of the 1s state. Hydrogen MASERs have been known experimentally since the late 1950's and are described in Reference 1.

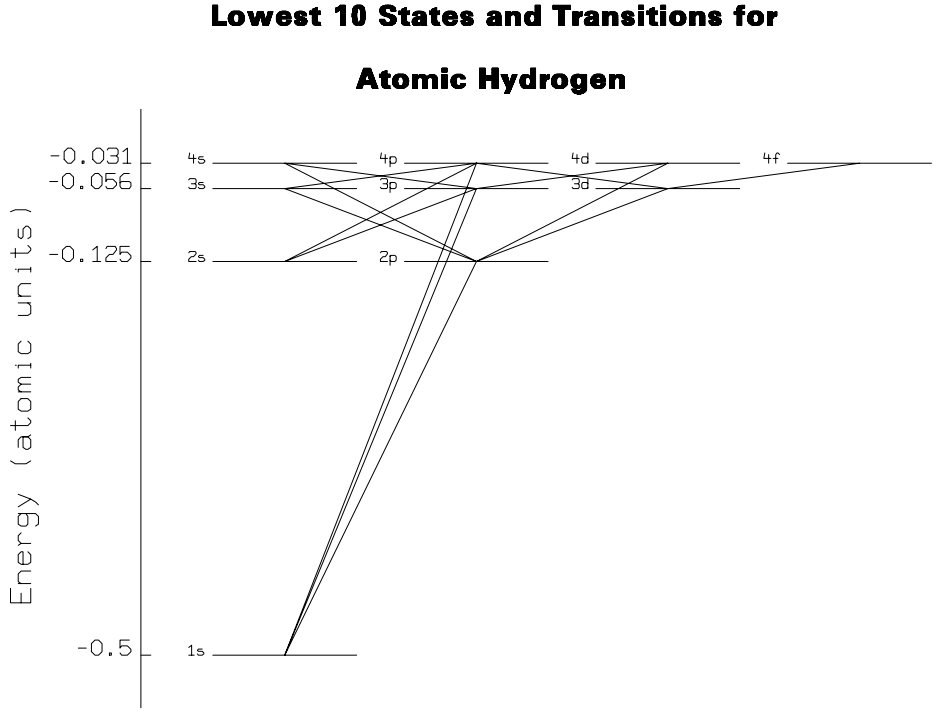


Figure 4 shows the lowest 10 levels of atomic hydrogen and the transitions that are allowed by dipole selection rules. Each line connecting two states represents three reactions: an absorption reaction, a spontaneous emission reaction, and an induced emission reaction.

The spontaneous emission rate constants for this system are calculated exactly from equations (59.11), (63.1), (63.2), and (63.3) of Reference 2. The absorption and induced emission rate constants are calculated from the values of the spontaneous emission rate constants and equations (8.5-1), (8.5-9), and (8.5-10) of Reference 3.

In this computational model, we assume only the ground state is initially populated (Boltzmann factors are not computed); and we continuously pump the 1s $\rightarrow$ 4p transition. The decay rate constants, number of photons, and number of atoms, are conveniently selected so that lasing occurs at a frequency of $(E_4 - E_3)/\hbar$. Here are the MLAB commands:

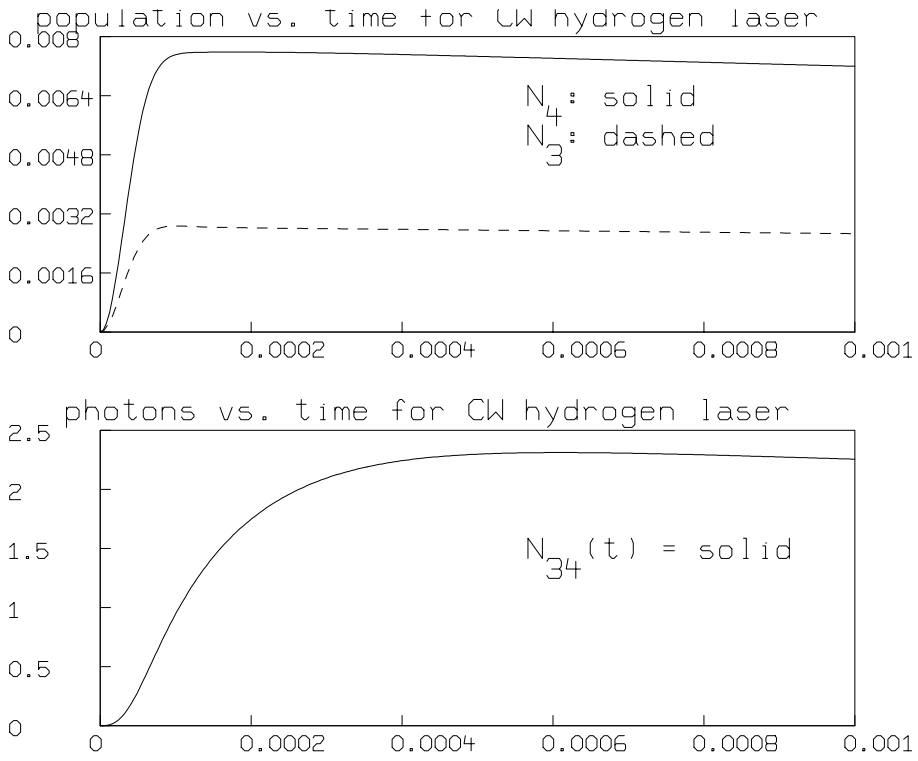
```
/* assign values for constants */
hb = 1.0546E-27  /* Planck's constant divided by 2 pi in erg sec */
c = 2.9979E10    /* speed of light in vacuum (cm/sec)*/
ep = 4.8029E-10  /* proton charge in esu */
em = 9.1096E-28  /* mass of electron in grams */
a0 = 5.2917E-9   /* radius of 1st Bohr orbit in cm */
Z = 1            /* nuclear charge of hydrogen atom */
epau = 4.360E-11 /* ergs per atomic unit of energy */
N = 1e3          /* total number of atoms */
Np = 1e10        /* total number photons per second pumped */
mlt = 1E-15      /* volume proportionality constant relating
                    spontaneous to induced processes */

/* function for evaluating energy of n-th atomic level in ergs */
fct E(n) = -epau*(Z^2)/(2*(n^2))

/* function for evaluating the degeneracy of state with l-orbital angular
   momentum */
fct g(l) = 2*l+1

/* functions for evaluating spontaneous emission rate constants */
fct f(a,b,c,d) = fact(c-a-b-1)*fact(c-1)*sum(k,max(0,1-c),min(-a,-b),\
     hgeomd(k,-a,c-a-b-1,-b)*d^k)/(fact(c-b-1)*fact(c-a-1))
fct R(n1,n2,l) = (-1)^(n1-l)*sqrt(fact(n2+l)*fact(n1+l-1))*(4*n1*n2)^(l+1)*\
     (n2-n1)^(n2+n1-2*l-2)*(f(-n2+l+1,-n1+l,2*l,-4*n2*n1/(n2-n1)^2)-\
     (((n2-n1)/(n2+n1))^2)*f(-n2+l-1,-n1+l,2*l,-4*n2*n1/(n2-n1)^2))/\
     ((4*fact(2*l-1))*sqrt(fact(n2-l-1)*fact(n1-l))*(n2+n1)^(n2+n1))
fct ks(ni,li,nf,lf) = 4*ep^2*((E(ni)-E(nf))/hb)^3*\
     ((a0*(if (lf>li) then R(ni,nf,lf) else R(nf,ni,li)))^2)/(3*hb*g(li)*c^3)

/* spontaneous emission rate constants */
ks2p1s = ks(2,1,1,0); ks3p1s = ks(3,1,1,0); ks4p1s = ks(4,1,1,0)
```

```
ks3p2s = ks(3,1,2,0); ks4p2s = ks(4,1,2,0); ks4p3s = ks(4,1,3,0)
ks3s2p = ks(3,0,2,1); ks4s2p = ks(4,0,2,1); ks4s3p = ks(4,0,3,1)
ks3d2p = ks(3,2,2,1); ks4d2p = ks(4,2,2,1); ks4d3p = ks(4,2,3,1)
ks4p3d = ks(4,1,3,2); ks4f3d = ks(4,3,3,2)

/* induced emission rate constants */
pch = mlt*PI^2*c^3*hb^2
ki2p1s = pch*ks2p1s/(E(2)-E(1))^3; ki3p1s = pch*ks3p1s/(E(3)-E(1))^3
ki4p1s = pch*ks4p1s/(E(4)-E(1))^3; ki3p2s = pch*ks3p2s/(E(3)-E(2))^3
ki4p2s = pch*ks4p2s/(E(4)-E(2))^3; ki4p3s = pch*ks4p3s/(E(4)-E(3))^3
ki3s2p = pch*ks3s2p/(E(3)-E(2))^3; ki4s2p = pch*ks4s2p/(E(4)-E(2))^3
ki4s3p = pch*ks4s3p/(E(4)-E(3))^3; ki3d2p = pch*ks3d2p/(E(3)-E(2))^3
ki4d2p = pch*ks4d2p/(E(4)-E(2))^3; ki4d3p = pch*ks4d3p/(E(4)-E(3))^3
ki4p3d = pch*ks4p3d/(E(4)-E(3))^3; ki4f3d = pch*ks4f3d/(E(4)-E(3))^3

/* absorption rate constants */
ka1s2p = ki2p1s*g(1)/g(0); ka1s3p = ki3p1s*g(1)/g(0)
ka1s4p = ki4p1s*g(1)/g(0); ka2s3p = ki3p2s*g(1)/g(0)
ka2s4p = ki4p2s*g(1)/g(0); ka3s4p = ki4p3s*g(1)/g(0)
ka2p3s = ki3s2p*g(0)/g(1); ka2p4s = ki4s2p*g(0)/g(1)
ka3p4s = ki4s3p*g(0)/g(1); ka2p3d = ki3d2p*g(2)/g(1)
ka2p4d = ki4d2p*g(2)/g(1); ka3p4d = ki4d3p*g(2)/g(1)
ka3d4p = ki4p3d*g(1)/g(2); ka3d4f = ki4f3d*g(3)/g(2)

/* decay rate constants */
kd12 = 1e12   /* 1<->2 photon loss from resonator in 1/second */
kd13 = 1e12   /* 1<->3 photon loss from resonator in 1/second */
kd14 = 1e12   /* 1<->4 photon loss from resonator in 1/second */
kd23 = 1e12   /* 2<->3 photon loss from resonator in 1/second */
kd24 = 1e12   /* 2<->4 photon loss from resonator in 1/second */
kd34 = 1e4    /* 3<->4 photon loss from resonator in 1/second */

/* initial 1s,2s,2p,3s,3p,3d,4s,4p,4d, and 4f populations */
init N1s(0) = N; init N2s(0) = 0; init N2p(0) = 0
init N3s(0) = 0; init N3p(0) = 0; init N3d(0) = 0
init N4s(0) = 0; init N4p(0) = 0; init N4d(0) = 0
init N4f(0) = 0

/* initial 1<->2, 1<->3, 1<->4, 2<->3, 2<->4, and 3<->4 photons */
init N12(0) = 0; init N13(0) = 0; init N14(0) = 0
init N23(0) = 0; init N24(0) = 0; init N34(0) = 0
```

```
/* photon rate due to pumping A->C transition */
fct pump14(t) = Np /* photons/second */\
   *(if t<0 then 0 else (1-(gaussd(t,0,1e-9)/gaussd(0,0,1e-9))))

/* define the rate equations */
fct N1s't(t) = -ka1s2p*N1s*N12-ka1s3p*N1s*N13-ka1s4p*N1s*N14\
               +ki2p1s*N2p*N12+ki3p1s*N3p*N13+ki4p1s*N4p*N14\
               +ks2p1s*N2p    +ks3p1s*N3p    +ks4p1s*N4p
fct N2s't(t) = -ka2s3p*N2s*N23-ka2s4p*N2s*N24\
               +ki3p2s*N3p*N23+ki4p2s*N4p*N24\
               +ks3p2s*N3p    +ks4p2s*N4p
fct N2p't(t) = +ka1s2p*N1s*N12-ka2p3s*N2p*N23-ka2p4s*N2p*N24\
               -ka2p3d*N2p*N23-ka2p4d*N2p*N24\
               -ki2p1s*N2p*N12+ki3s2p*N3s*N23+ki4s2p*N4s*N24\
               +ki3d2p*N3d*N23+ki4d2p*N4d*N24\
               -ks2p1s*N2p    +ks3s2p*N3s    +ks4s2p*N4s\
               +ks3d2p*N3d    +ks4d2p*N4d
fct N3s't(t) = +ka2p3s*N2p*N23-ka3s4p*N3s*N34\
               -ki3s2p*N3s*N23+ki4p3s*N4p*N34\
               -ks3s2p*N3s    +ks4p3s*N4p
fct N3p't(t) = +ka1s3p*N1s*N13+ka2s3p*N2s*N23-ka3p4s*N3p*N34-ka3p4d*N3p*N34\
               -ki3p1s*N3p*N13-ki3p2s*N3p*N23+ki4s3p*N4s*N34+ki4d3p*N4d*N34\
               -ks3p1s*N3p    -ks3p2s*N3p    +ks4s3p*N4s    +ks4d3p*N4d
fct N3d't(t) = +ka2p3d*N2p*N23-ka3d4p*N3d*N34-ka3d4f*N3d*N34\
               -ki3d2p*N3d*N23+ki4p3d*N4p*N34+ki4f3d*N4f*N34\
               -ks3d2p*N3d    +ks4p3d*N4p    +ks4f3d*N4f
fct N4s't(t) = +ka2p4s*N2p*N24+ka3p4s*N3p*N34\
               -ki4s2p*N4s*N24-ki4s3p*N4s*N34\
               -ks4s2p*N4s    -ks4s3p*N4s
fct N4p't(t) = +ka1s4p*N1s*N14+ka2s4p*N2s*N24+ka3s4p*N3s*N34+ka3d4p*N3d*N34\
               -ki4p1s*N4p*N14-ki4p2s*N4p*N24-ki4p3s*N4p*N34-ki4p3d*N4p*N34\
               -ks4p1s*N4p    -ks4p2s*N4p    -ks4p3s*N4p    -ks4p3d*N4p
fct N4d't(t) = +ka2p4d*N2p*N24+ka3p4d*N3p*N34\
               -ki4d2p*N4d*N24-ki4d3p*N4d*N34\
               -ks4d2p*N4d    -ks4d3p*N4d
fct N4f't(t) = +ka3d4f*N3d*N34-ki4f3d*N4f*N34-ks4f3d*N4f
fct N12't(t) = -ka1s2p*N1s*N12+ki2p1s*N2p*N12+ks2p1s*N2p-kd12*N12
fct N13't(t) = -ka1s3p*N1s*N13+ki3p1s*N3p*N13+ks3p1s*N3p-kd13*N13
fct N14't(t) = pump14(t)-ka1s4p*N1s*N14+ki4p1s*N4p*N14+ks4p1s*N4p-kd14*N14
fct N23't(t) = -ka2s3p*N2s*N23+ki3p2s*N3p*N23+ks3p2s*N3p\
               -ka2p3s*N2p*N23+ki3s2p*N3s*N23+ks3s2p*N3s\
               -ka2p3d*N2p*N23+ki3d2p*N3d*N23+ks3d2p*N3d-kd23*N23
```

```
fct N24't(t) = -ka2s4p*N2p*N24+ki4p2s*N4p*N24*ks4p2s*N4p\
               -ka2p4s*N2p*N24+ki4s2p*N4s*N24+ks4s2p*N4s\
               -ka2p4d*N2p*N24+ki4d2p*N4d*N24+ks4d2p*N4d-kd24*N24
fct N34't(t) = -ka3s4p*N3s*N34+ki4p3s*N4p*N34+ks4p3s*N4p\
               -ka3p4s*N3p*N34+ki4s3p*N4s*N34+ks4s3p*N4s\
               -ka3p4d*N3p*N34+ki4d3p*N4d*N34+ks4d3p*N4d\
               -ka3d4p*N3d*N34+ki4p3d*N4p*N34+ks4p3d*N4p\
               -ka3d4f*N3d*N34+ki4f3d*N4f*N34+ks4f3d*N4f-kd34*N34

/* integrate the kinetics equations for .001 seconds */
disastersw = -1
p = integrate(N1s,N2s,N2p,N3s,N3p,N3d,N4s,N4p,N4d,N4f,N12,N13,N14,N23,N24,\
      N34,0:.001!250)
/* cols  p: T  N1s   N1s'T   N2s   N2s'T   N2p   N2p'T   N3s   N3s'T
               N3p   N3p'T   N3d   N3d'T   N4s   N4s'T   N4p   N4p'T
               N4d   N4d'T   N4f   N4f'T   N12   N12'T   N13   N13'T
               N14   N14'T   N23   N23'T   N24   N24'T   N34   N34'T */
```

Here is the graph of the total populations of the $N = 4$ and $N = 3$ levels versus time:



The population inversion is seen to occur after .0005 seconds.

Mathematical models of *molecular* lasers can be implemented in a manner similar to that used here to model an *atomic* laser. Additional terms in the rate equations may be introduced that account for various radiationless relaxation processes, such as rotational-to-vibrational relaxation and electronic-to-vibrational relaxation. A computational limitation does arise in such mathematical models when there are many coupled rate equations; the equations become difficult to integrate without numerical errors accruing at each time step. Such systems of equations are referred to as *ill-conditioned* and one can see erroneous exponential growth in one or more of the quantities of interest. Although MLAB employs several ordinary differential equation-solving algorithms, including Gear's method for integrating stiff equations, solving ill-conditioned differential equations over too large an interval will produce unreliable results.

17.3 References

1. Norman F. Ramsey, "The Atomic Hydrogen Maser", *American Scientist* **56** (1968) 420-438.

2. Hans A. Bethe and Edwin E. Salpeter, *Quantum Mechanics of One- and Two-Electron Atoms* (NY: Plenum Publishing Corp, 1977) 248-269.

3. Amnon Yariv, *Quantum Electronics* (NY: John Wiley and Sons, 1989) 171-173.

18 Telephone Calls to Ordered Ports

This example shows how MLAB can be used to solve a statistical simulation and display problem. Imagine that telephone calls arrive at a bank of n ports and each call is assigned to the first open port found by scanning, in order, ports 1, 2, ... We assume n is large enough so that no call will be denied a port. Suppose that the length of the ith call, l_i, follows a lognormal distribution, so that, for all i, the length, l_i, of the ith call has the probability density function

$$f(x) = \exp(-\log(x/m)^2/(2\sigma^2))/(x\sigma(2\pi)^{1/2}),$$

and

$$P(l_i \leq h) = \int_0^h f(x)\,dx.$$

The average length of a call is then $m \cdot \exp(\sigma^2/2)$, and we will choose $m = 10$ and $\sigma = .5$. The call lengths l_1, l_2, ... are all independent. Also, suppose calls arrive at varying times with exponentially-distributed inter-arrival times. Let A_{0j} be the time between the arrivals of the jth and $(j+1)$st calls. Then we assume that the probability that there are s or fewer minutes between the arrival of the jth call and the arrival of the $(j+1)$st call is $P(A_{0j} \leq s) = 1 - e^{-as}$, where a is, say, 1 minute.

It is of interest to use MLAB to simulate this process and to graph the simulated use of the various ports as follows. In the graph below, the time of arrival of calls are marked on the t-axis, and each use of port i is shown by a solid line, starting at the indicated time and extending for the duration of the call.

A graph of this form is determined by a list of triples, one for each arriving call; so that for call i, we have (t_i, l_i, p_i) where t_i is the time at which call i arrives, l_i is the length of call i, and p_i is the port to which call i is assigned. Then we need only draw the line segments (t_i, p_i) to $(t_i + l_i, p_i)$ for $i = 1, 2, \ldots$, and, of course, mark the t-axis at the points $(t_i, 0)$ to indicate the times at which the calls arrive.

Given data of this form, we can, for example, easily estimate the percentage of time port i is busy, or estimate the number of ports needed to service 95% of the incoming calls.

Let A_{ij} denote the interarrival time, A_{ij}, between the jth and $(j+1)$st calls assigned to port i; A_{ij} is a random variable. An analysis of interest is to state the distribution of A_{ij} in terms of i, j, and the distribution functions of the length and interarrival time of incoming calls. It often suffices to determine the limiting form for the distribution of A_{ij} as $j \to \infty$.

In the case where the times of arrival of incoming calls forms a Poisson process as assumed above, we can explicitly state the desired distributions.

Let D_{ij} be the non-busy waiting time between the end of the jth call on port i and the arrival of the next call assigned to port i. Let $L \sim l_1$, so L is lognormally-distributed, as is l_1. Note the inter-arrival time $A_{ij} \sim L + D_{ij}$ for $i \geq 1$, and also $D_{1j} \sim A_{0j}$.

Let $D_i = \lim_{j\to\infty} D_{ij}$. Indeed $D_{ij} \sim D_i$ for $j > 1$. Then, in the limit, port i is busy the fraction f_i of the time, where $f_i = E(L/(L + D_i))$.

Now, when port i is free, an arriving call will be assigned to port i when ports 1, 2, ..., $i - 1$ are all busy. So if a call arrives during an interval of time which satisfies this condition, it will be assigned to port i. Thus, in the limit, D_i is the waiting time until the occurrence of the event that ports 1, 2, ..., $i - 1$ are busy and a call arrives, and this can be interpreted as the waiting time until the next event of a Poisson process with density $f_1 f_2 \cdots f_{i-1}/a$, where $1/a$ is the density of incoming calls. Thus D_i is exponentially-distributed with mean $a/(f_1 f_2 \cdots f_{i-1})$, for $i > 1$, and $D_1 \sim A_{0j}$.

Moreover, in the limit, the number of busy ports, N, at a fixed time is a Poisson random variable with mean $u = E(L)/a$; that is $P(N = k) = u^k e^{-u}/k!$.

Note this model is of interest where "calls" are sessions with a time-sharing computer, people arriving for service at a system of queues which are scanned linearly, and other isomorphic situations. Let the vector $(V_1, V_2, \ldots, V_n)$ denote the state of the n ports at the current time, t, so that $V_i = 0$ if port i is not busy, and $V_i =$the length of time remaining for the call using port i otherwise. Now, given that the next new call of length l arrives in s minutes from the current time t, we can update the state vector V to be valid at time $t + s - \varepsilon$, where ε is very small, by changing each V_i so that $V_i \leftarrow \max(V_i - s, 0)$. Then the state vector V is updated to be valid at time $t + s + \varepsilon$ by setting V_j to l, where $j = X(V)$, and $X(V)$ is the least index in $\{1, 2, \ldots, n\}$ such that $V_{X(V)} = 0$. Let us denote the updating process used to make V the current state vector for time $t + s - \varepsilon$, so that we may write $V \leftarrow u_s(V)$ to denote updating V to time $t + s - \varepsilon$.

Now our simulation can be programmed as follows. Note a total of d calls are simulated.

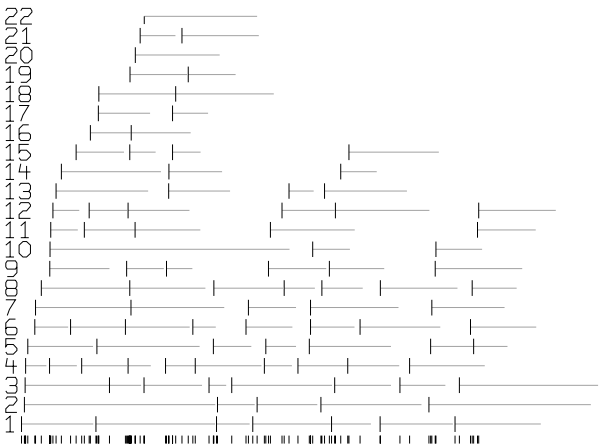
begin
  $t_0 \leftarrow 0$; $j \leftarrow 1$; $V \leftarrow 0$; $\alpha : t_j \leftarrow random{-}exponential + t_j - 1$;
  $l_j \leftarrow random{-}lognormal$;
  $V \leftarrow u_{t_j - t_{j-1}}(V)$; $p_j \leftarrow X(V)$; $V_{p_j} \leftarrow l_j$; $j \leftarrow j + 1$; if $j \leq d$ then goto α;
end.

Upon completion, we have the three vectors, t, l, and p needed to draw the required graph.

We may use MLAB to carry out the simulation sketched above and then draw the generated graph of port usage over time. A general do-file to do the simulation and draw the graph is given below, together with a sample graph generated by using it.

```
"calls.do"
type "enter n: the number of calls";
n = kread();

type "enter a random number seed";
r = kread(); r = ran(r);

type "enter np: number of telephone ports";
np = kread();

v = 0^^np; "(np ports. Port i is busy for v[i] minutes)";

"generate t vector in order: partial sums of samples of an exponential
distribution with mean 1";
t = -(log on (ran on 0^^n));
t = 0 & psum(t);

"generate l vector: lognormal with mean 10*exp(-(1/8)).";
l = normran on 0^^n; l = 10*(exp on .5*L);

"compute the p vector";
function pv(i) = if i > np then np else if v[i] = 0 then i else pv(i+1);
function u(w) = if w > s then w-s else 0;
p[n] = 0;
for j = 1:n do \
   {i = minrow(v); if v[i] not= 0 then {np=np+1; i=np;};
    p[j] = i; v[i] = l[j];
    s = (if j = n then 0 else t[j+1])-t[j];
    v = u on v}

"given t, l, and p, draw the graph.";
m1 = t&'p; m2 = (t+l)&'p;
draw m1 line none pointtype vbar color red in w1
draw mesh(m1,m2) line alternate color green in w1
draw t&'0 line none pointtype vbar in w1
draw 0&'(1:np) line none label 1:np offset (-.02,-.01) color yellow in w1
no imagebox in w1
view
```

Here is a graph of a simulation run with 100 calls produced by the above do-file.



As an exercise, you might try do the simulation again, assuming that the mean of the exponential distribution for the inter-arrival times of incoming calls varies smoothly with the time of day, thus mimicking a heavier load during working hours, with peaks at, say, 10am and 3pm. This can be done by assuming the mean inter-arrival time, $E(A_{0j})$, is an appropriate doubly-peaked function of time.

19 Balls in Cells: An MLAB Tutorial Demonstration Sequence

Below is an annotated sequence of MLAB commands to solve an interesting curve-fitting and graphics display problem. If you execute the MLAB statements and read the notes scattered throughout, you should get an appreciation of MLAB and some of its abilities.

The problem we have to solve is a simple occupancy problem which might arise in quantum physics or elementary statistics. We are given data derived from a sequence of experiments, and we shall attempt to fit a model to this data to determine the best estimates for certain parameters.

There are N cells or buckets and K balls, where N and K are fixed but unknown. We wish to estimate N and K. An experiment consists of randomly distributing the K balls among the N buckets and then counting the number of buckets which hold exactly I balls.

The catch is that, for each experiment, we can do this for just one value of I which we choose a priori (due to fundamental quantum limitations). I may be different for each experiment.

Thus each experiment yields a point, $(I, OCCI)$, which indicates that $OCCI$ buckets held exactly I balls after the experiment was done.

We are given the following collection of experimental points (observations) and the problem is to estimate the values of N and K which underlie this data. We shall use curve-fitting to do so.

The data is given below, where each column represents a point corresponding to a single experiment:

```
   I: 0 0 1 2 2 3 3 4 4 5 6 7 7 8 8 9 10 10 11 14
OCCI: 0 1 1 1 2 3 1 0 2 2 0 1 2 1 1 0  0  1  0  0
```

The first step is to input our data and build an appropriate matrix of observation points, with each row being such a point. If we had lots of data, we could set up a file and then read it , but here we can just type in the data directly and thus define our matrix, B. Now type: `B COL 1 = KREAD(20).`

MLAB will then ask you to enter a sequence of 20 numbers. Type:

```
0,0,1,2,2,3,3,4,4,5,6,7,7,8,8,9,10,10,11,14.
```

Now type: `B COL 2 = KREAD(20)`, and enter the second column of data by typing:

```
0,1,1,1,2,3,1,0,2,2,0,1,2,1,1,0,0,1,0,0.
```

Verify the matrix B by typing: `TYPE B`.

If you made any mistakes, you can correct any incorrect entries with explicit matrix assignment statements, e.g. `B[20,1]=14`.

The model is a function $OCC(I)$ which has parameters N and K and which computes the expected number of i-full buckets, i.e., the expected number of buckets with i balls in them, given N, K, and I. OCC is defined in terms of the binomial coefficient $\binom{K}{I}$ which is the number of combinations of I things taken out of K things; we shall define $COMB(K,I) = \binom{K}{I}$. (The same computation is performed by the builtin MLAB function `bincoef`, but we need the symbolic derivatives of $COMB$, so we need a form that MLAB knows how to differentiate.) $COMB$ can be defined in terms of the factorial power function, $LBAR$, defined recursively below.

To define $LBAR$ type:

```
FUNCTION LBAR(K,I) = IF I=0 THEN 1 ELSE K*LBAR(K-1,I-1).
```

$LBAR(K,I)$ computes the falling factorial power $K(K-1)\cdots(K-I+1)$.

Note that $LBAR(I,I)$ is just the factorial function, $I!$. To check this, let's build a table of factorials.

First we set up M, a 2-column matrix of arguments, $(0,0)$ through $(10,10)$, for $LBAR$. Type `M = (0:10)^^'2`. Now type: `TYPE LBAR ON M`. The first 11 factorials are the result.

Now let us define $COMB$ and then OCC. Type in:

```
FUNCTION COMB(K,I) = LBAR(K,I)/LBAR(I,I)
FUNCTION OCC(I) = N*COMB(K,I)*(1/N)^I*(1-1/N)^(K-I)
```

A derivation of OCC is given later for those who wish to read it. Here let us experiment with OCC to see that it is at least plausible. Hence, let us set N and K, and look at the theoretical distribution of K balls in N buckets. Type: `N = 10; K = 60`. To compute the points $(J, OCC(J))$ for $J = 0, 1, 2, \ldots, 15$, type: `T = POINTS(OCC,0:15)` and see what this is by typing: `TYPE T`.

To see a graph of the points in T drawn in the MLAB default window W, type:

```
DRAW T linetype none pointtype circle; view
```

See how the balls are spread among the buckets (really OCC shows us how the buckets are filled by the balls). The graph is a scaled binomial distribution. Press the enter key to proceed.

The matrix T and the graph in W have served their purpose, so, to save space, type: `DELETE T,W`.

Now we shall use curve-fitting to estimate N and K. What we shall do is find those values of N and K which makes OCC best fit the data B. This N and this K, rounded to integer values, will be our estimates for the true N and K.

Before we proceed however, we may note that in order to fit OCC to B by adjusting N and K, we need to be able to compute the derivative of OCC with respect to N and also the derivative with respect to K. MLAB computes these derivatives automatically. To see what they are, just for fun, type: `TYPE OCC'N, OCC'K`.

For $OCC'K$, the chain rule was used. To see the partial derivatives created, type the command: `TYPE COMB'K, LBAR'K`.

It turns out that differentiating the recursive form of $LBAR$ gives the correct K-derivative for $LBAR$ in recursive form. This is indeed good fortune. (This always works for purely multiplicative recursive forms.)

Let's begin our curve-fitting by guessing that the right answers for N and K are 8 and 8. Thus type: `N=8; K=8`.

Now to fit OCC to B, type: `LSQRPT = 15; FIT(N,K), OCC TO B`

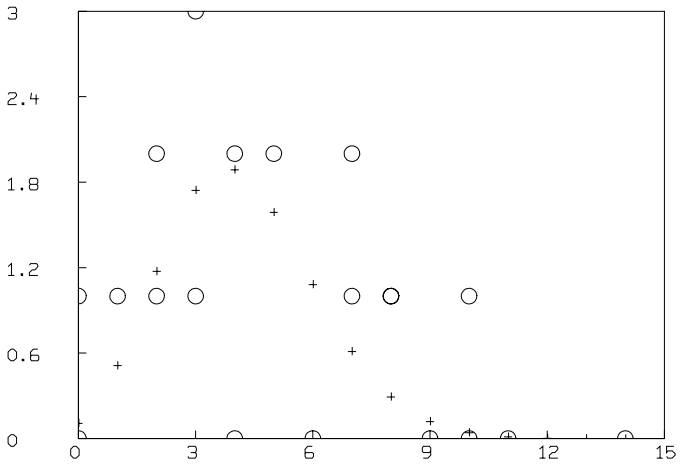
Watch carefully how the fitting process adjusts N and K as it proceeds. The final answer will be $N = 9.1996$, and $K = 38.509$. Thus, $N = 9$ and $K = 39$ is a reasonable solution to our problem.

Type: `TYPE N,K`. These are the desired raw estimates.

In order to see how we did, let's draw a graph of the data and the theoretical best fit on the display screen. All "drawing" is done in one or more "windows". If a window is not specified, a particular window called `W` is used. The default window `W` contains a number of pre-specified curves and labels which form an "axis-box".

Type:

```
DRAW B POINTTYPE CIRCLE LINETYPE NONE
DRAW POINTS(OCC,0:15) LINETYPE NONE POINTTYPE "+"
VIEW
```



Look at the display. The picture above should be showing. This shows how the data and the curve-fit model compare. When you are finished looking at the picture, press the ENTER key to return to MLAB command mode.

MLAB is capable of composing quite elaborately-specified pictures suitable for direct publication. Let us experiment just a little with these graphic facilities by composing a picture from scratch, showing the original data, B, the graph of the initial OCC function with $N = K = 8$, and the graph of the final OCC function after adjusting N and K. To free the memory ocupied by the default window `W`, type: `DELETE W`.

Let us set up our own window and its image. Type:

```
WINDOW -1 to 16, -.25 TO 3.25 IN G
FRAME 0 TO 1, 0 TO 1 IN G
IMAGE .15 TO .85, .15 TO .85 IN G.
```

The `WINDOW` statement above defines a patch of the cartesian plane called G within which drawn curves will be visible; the frame statement defines the screen region where graphics will be displayed; and the `IMAGE` statement specifies a patch in the frame region within which the material within the patch G will be seen. The to-clauses in the window statement use “world” units, i.e., the actual coordinates of whatever is drawn. The to-clauses in the frame statement use “screen fraction” units, i.e., the fraction of the horizontal and vertical dimension of the display screen. The to-clauses in the image statement use ”frame fraction” units, i.e., the fraction of the horizontal and vertical dimension of the frame.

Set up a horizontal axis for G by typing:

```
AXIS XAX=(0:15)&'(-.5) IN G POINTTYPE UTICK LABEL (0:15:4)&'(1:16:4) OFFSET (0,-.05)
```

Set up a vertical axis for G by typing:

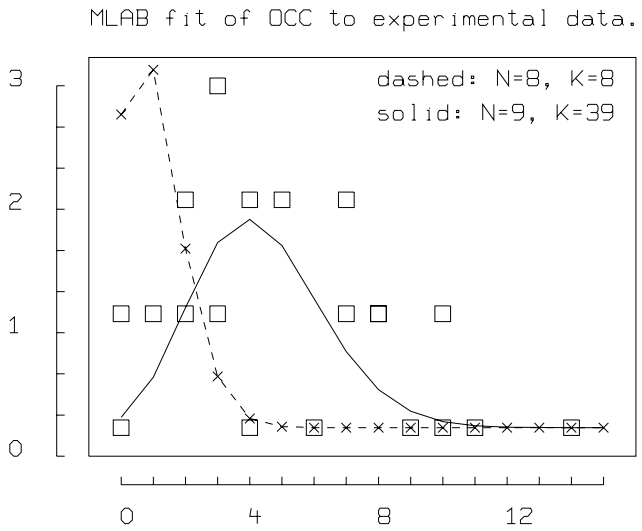
```
AXIS YAX=(-2)&'(3:-.25!10) IN G POINTTYPE RTICK LABEL (3:0)&'(1:10:3) OFFSET (-.06,0)
```

Now we'll draw our curves by typing:

```
DRAW BC=B IN G POINTTYPE SQUARE LINETYPE NONE
N=8;K=8;
DRAW INITOCC=POINTS(OCC,0:15) IN G LINETYPE DASHED POINTTYPE "X"
N=9;K=39
DRAW FITOCC=POINTS(OCC,0:15) IN G
view
```

Ok. Let's label these curves with some descriptive text. Press the enter key and then type:

```
TITLE "N=8,K=8" AT (.15,.8) IN G
TITLE"N=9,K=39", AT (.15,.75) IN G
TOP TITLE "MLAB fit of OCC to experimental data" COLOR YELLOW IN G
view.
```



Remember to press the enter key to return to MLAB command mode. This process you have just gone through, of creating your own "window", is very useful.

Finally we shall get an offline plot of this picture. Type: `PLOT G`. A disk file in the Postscript language containing the information to be plotted is thus created on your disk area. (Alternatively an H-P LaserJet II-compatible file may be created by setting the control variable `PLOTDEV` appropriately.)

Now clean up by typing `DELETE G`.

Now type `EXIT` to exit from MLAB. You are now speaking to DOS. MLAB has written a log-file called `MLAB.LOG` containing a record of your MLAB session. It is just an ASCII file, and may be printed in the usual way. The plot file may be printed using a Postscript printing utility program (e.g., the DOS utility print program), which sends the plot-file to a Postscript printer.

Here is a derivation of OCC:

In order to see that $OCC(I)$ is indeed the expected number of I-full buckets, given K balls and N buckets, consider the following.

1. Each ball has probability $p = 1/N$ of occupying any particular arbitrary but fixed bucket, and probability $q = 1 - 1/N$ that it occupies some one of the other buckets.
2. Therefore the probability that exactly i balls out of K occupy any particular arbitrary but fixed bucket is given by the binomial distribution, which yields

$$Pr(\text{exactly } I \text{ balls out of } K \text{ land in an arbitrary fixed bucket}) = \binom{K}{I} p^I q^{K-I} =: h(I).$$

3. Thus the expected number of I-full buckets altogether is: $N \cdot h(I) =: OCC(I)$.

Note an alternative MLAB definition for the function OCC is:

```
fct OCC(I) = IF I=0 THEN N*(1-1/N)^K ELSE OCC(I-1)*((K-I+1)/I)/(N-1).
```

It might be preferable to estimate N and K using maximum-likelihood estimation. This can be done in MLAB by formulating an equivalent least-squares minimization problem. An interesting unanswered question (which is perhaps unanswerable due to Bayes theorem) is that of specifying quantitatively how good our answer is. It is clear that N and K as computed above are reasonable estimates, but we would like to be able to say that N is such that N is within t of the true answer with probability P(t), and similarly for K. It is possible that under appropriate assumptions such reliability results can be obtained.

20 Kaplan-Meier Survival Curve Estimation

Suppose we have survival data for a group of n similar patients, with censoring present, so that the data consists of pairs of values (y_1, e_1), (y_2, e_2), ..., (y_n, e_n) where each e_i is either 0 or 1. When $e_i = 1$, y_i is the time until death of patient i, counting from the study starting point, and when $e_i = 0$, y_i is the censoring time for patient i, indicating that patient i was lost to follow-up with an unknown fate after y_i time-units from the study starting point. The value e_i is called the result-code for patient i. Note survival time can be, in fact, the time until a "response" of some kind occurs; thus survival modeling has more general application than may be apparent.

Suppose the patients in the group have survival times, X_1, X_2, ... X_n, which are independent identically-distributed random variables, a realization of which is, except for censoring, given by our data. Let $F(t)$ be the common distribution function of X_1, ..., X_n. We wish to estimate the survival function $S(t) = P(X_i > t) = 1 - F(t)$.

Associated with each X_i, we postulate a censoring-time random variable, C_i. The random variables C_1, ..., C_n are assumed to be independent and identically-distributed, with the distribution function $G(t) = P(C_i \leq t)$. For any realization, $(\tilde{X}_i, \tilde{C}_i)$ where $\tilde{X}_i$ is a sample of X_i and $\tilde{C}_i$ is a sample of C_i, if $\tilde{X}_i \leq \tilde{C}_i$, we have $y_i = \tilde{X}_i$, and $e_i = 1$, while when $\tilde{X}_i > \tilde{C}_i$, we have $y_i = \tilde{C}_I$ and $e_i = 0$. Thus, the value y_i is a sample of $\min(X_i, C_i)$.

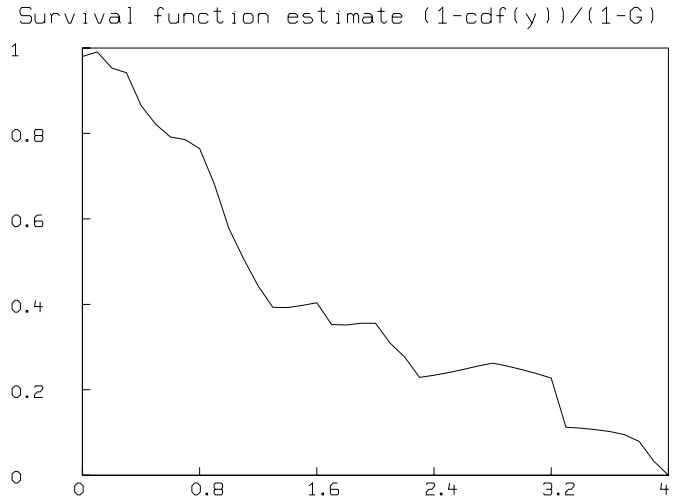
If we assume a specific formula for $G(t)$, then we can estimate $S(t)$ by $(1-H(t))/(1-G(t))$ where $H(t)$ is the empirical cumulative distribution function of the data-values y_1, ..., y_n.

For example, suppose the censoring-time distribution is uniform on the interval $[0, 4]$; then, given the column vector Y listed below, where $Y_i = y_i$, we can estimate the survival function S in MLAB with the function SE given below. The result-code vector, E, is listed below together with Y for later reference.

```
*TYPE Y&'E
MATRIX :
1.635686 0
.3113416 0
.4963642 1
1.131409 0
2.221448 0
1.668752 1
2.785097 0
.3379156 0
.4495545 1
3.020983 1
.3994093 0
```

```
.9359379 1
.9697097 0
.6193121 0
.1234482 0
.8931070 1
.2233199 1
.8303535 0
3.926499 0
2.012626 1
.3871494 1
.9381460 1
1.210837 1
1.704919 0
1.006140 1
1.998524 1
.3763392 0
1.184550 0
.7832847 1
.9971150 0
.0452861 0
2.145687 1
.8396369 1
1.319498 0
3.288359 1
.7552886 1
1.017174 0
.8238844 1
.5893285 1
.2916714 0
3.266290 0
2.215458 1
.5546782 0
.1355474 1
.1247086 0
.9045237 1
.4080161 1
1.188316 1
1.243582 1
1.043640 1

 *L = 4
 *H = CDF(Y)
```

```
 *FUNCTION SE(T)=IF T>=L THEN 0 ELSE (1-LOOKUP(h,t))/(1-T/L)
 *DRAW POINTS(SE,0:L:.1)
*top title "Survival function estimate (1-cdf(y))/(1-G)"
*VIEW
```

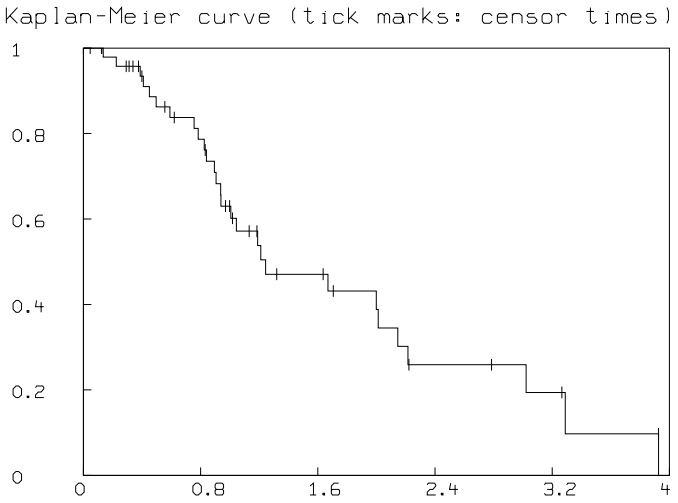


One drawback of this estimator is the fact that it is not monotonically decreasing, as is seen above. Alternatively, we can avoid assuming a specific censoring-time distribution by using the asymptotically-unbiased Kaplan-Meier product-limit estimator function, $\hat{S}$ as the estimator function for the survival function S. This can be computed in MLAB using the `KMSURV` function and graphed using the `STEPGRAPH` function.

For example, given the data, Y, listed above, together with the associated result-code vector, E, we can produce a graph of $\tilde{S}$ as shown below. The tic-marks show the points where censored observations occur.

```
*DELETE W
*D = Y&'E
*D = SORT(SORT(D,2,-1),1)
*H1 = KMSURV(D)
*H = STEPGRAPH(H1 COL 1:2)
*R = (0 &' 1) & H & (H[NROWS(H),1] &' 0)
*DRAW R, COLOR RED
*Y1 = COMPRESS(D,2,1) COL 1
*FCT F(X) = LOOKUP(H,X)
```

```
*H2 = POINTS(F,Y1)
*DRAW H2 LINE NONE, PT VBAR, PTSIZE .015, COLOR GREEN
*WINDOW 0 TO 4, 0 TO 1
*TOP TITLE "Kaplan-Meier curve (tick marks: censor times)"
*VIEW
```

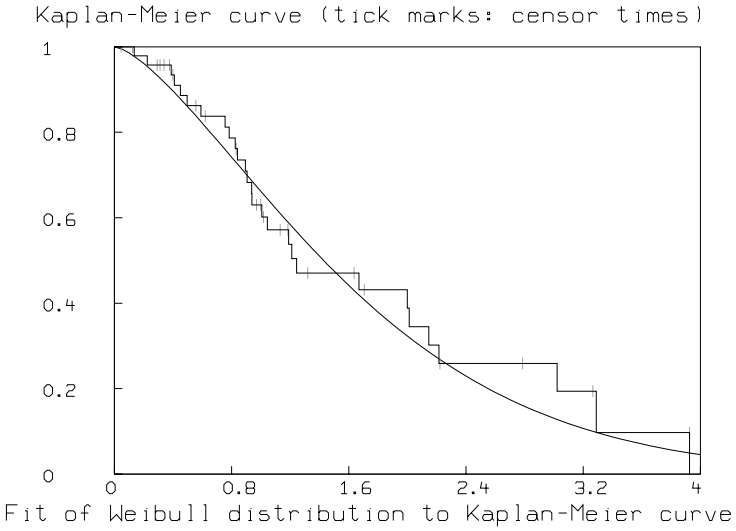


We may postulate a specific form for the distribution of the X_i random variables. For example, if the survival time distribution function is a Weibull distribution with group-specific parameters, a and b, then the survival curve is given by $SW(t) = exp(-(t/a)^b)$.

Now, we may estimate the parameters a and b using MLAB to fit the model function $SW(t)$ to a matrix of points lying on the Kaplan-Meier estimated survival curve. This is demonstrated for the matrix $H1$ of points on the estimated survival curve obtained above. Greenwood's variance approximation for $\hat{S}$ computed by `KMSURV()` in `H1 col 3` is used to compute appropriate weights for the curve-fit.

```
*FUNCTION SW(T)=EXP(-((IF T=0 THEN .000001 ELSE T)/A)^B)
*A = 1;B = 1; H1[1,3] = 1
*FIT(A,B), SW TO H1 COL 1:2 WITH WEIGHT 1/(H1 COL 3)
final parameter values
   value            error             dependency    parameter
    1.837800012    0.05250877741     0.3849266596   A
    1.451240067    0.05755968774     0.3849266596   B
4 iterations
```

```
CONVERGED
best weighted sum of squares = 8.691990e+00
weighted root mean square error = 5.673848e-01
weighted deviation fraction = 1.869069e-02
R squared = 9.756895e-01
*DRAW POINTS(SW,0:4:.05)
*bottom title "Fit of Weibull distribution to Kaplan-Meier curve"
*view
```



Often, we wish to compare the survival curves of several dissimilar groups of patients, to determine, for example, which of several distinct treatments is superior. This can be done by comparing the estimated survival curves directly with an appropriate statistical test (such as the Mantel-Haensel test, MHT) provided by MLAB. We may also want to fit explanatory models to the survival curves, as above, but with embedded expressions which depend upon the various auxillary parameters, such as age, sex, or treatment. These fits then numerically characterize how the auxillary parameters determine survival time.

It may be of interest to compute the expected survival time of a subject who has already survived k time-units. This is just $M(k) := \int_k^\infty tq(t|k)\,dt$, where $q(t|k)$ is the conditional density function, $d[P(X_1 \geq t|X_1 \geq k)]/dt$. But given the survival time distribution function F and the corresponding density function $f(t) = dF(t)/dt$, we can compute $q(t|k) =$ if $t < k$ then 0 else $f(t)/a$, where

$$a = \int_k^\infty f(s)\,ds = 1 - F(k) = S(k).$$

Note $q(t|0) = 0$ and $q(t|t)$ is just the hazard function $f(t)/S(t)$.

The expected additional survival time function $M(k)$ can be easily computed and graphed in MLAB. One use for the function $M(k)$ is to estimate lifetimes for subjects with censored survival times. By thus "completing" a data set, we obtain uncensored data which is amenable to a variety of otherwise inapplicable statistical procedures.

One interesting use of survival curve modeling is as follows. Suppose we have a survival function $s(t)$, possibly obtained by curve-fitting observed survival data. Let $s(t)$ be the fraction of machines (or components, or people, or "items") which survives at least to time t. Suppose that we wish to replace these machines on a regular schedule so as to maintain the constant level of a_0 machines in service. Note that each newly-introduced replacement batch of machines follows the same survival behavior as the original machines. We wish to compute the replacement schedule function. Note the replacement function can be used for budget projection purposes.

First we shall look at the discrete formulation of the problem. Set $s_0 = 1$ and in general let s_i be the fraction of machines which survive for at least i weeks. Let a_0 be the initial number of machines at time 0, and let a_i denote the number of machines placed in service at week i. The number of machines operating starting at week k is $a_0 s_k + a_1 s_{k-1} + \cdots + a_k s_0$. Since we wish to maintain the constant level of a_0 machines at each week, we equate a_0 and $a_0 s_k + a_1 s_{k-1} + \cdots + a_k s_0$. From this we get the following recursion equation for a_k, $0 \leq k$.

$$a_k = (a_0 - \sum_{j=1}^{k-1} a_j s_{k-j})/s_0$$

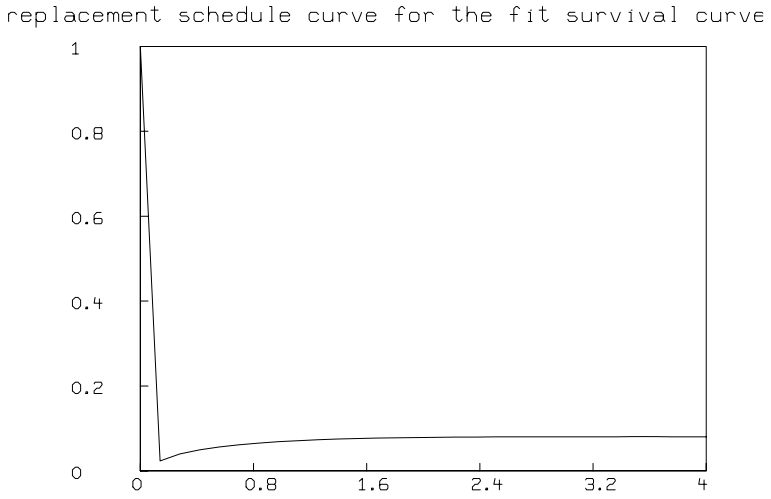
with a_0 and s_0, s_1, ..., s_k given.

This represents the solution vector of the following lower-triangular Toeplitz system.

$$\begin{bmatrix} s_0 & 0 & 0 & 0 \\ s_1 & s_0 & 0 & 0 \\ \vdots & s_1 & s_0 & \vdots \\ & \vdots & & \\ s_k & s_{k-1} & \cdots & s_0 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} a_0 \\ a_0 \\ \vdots \\ a_0 \end{bmatrix}.$$

The MLAB function `DECONV` may be used to compute the solution to this linear system.

For example, here is the replacement schedule function computed by using `DECONV` based on the fit Weibull survival curve for the data above.



In the continuous formulation we have the given survival function $s(t)$ and we wish to compute the replacement function $a(t)$ with $a(0) = a_0$ given. The value $a(t)$ is the amount of machinery to be put into service at time t to maintain the constant level $a(0)$.

Here we get the convolution equation $a * s = a_0$ (using zero-extension). Using the Fourier convolution theorem, we get $a = ((a_0)^\wedge / s^\wedge)^\vee$. Since the Fourier transform costs $O(k \log k)$ time for k points when k is highly composite, this deconvolution method is also a method for solving the lower-triangular system given above (when k is highly composite).

21 Sums of Gaussians: A Tutorial Sequence

A common curve-fitting problem is that of decomposing a curve, often called a spectrum or a signal, into a sum of instances of a particularly-chosen form of component curve. This arises in spectral analysis for example.

Thus, we assume we have a function, $f(t)$, such that $f(t) = \sum_{i=1}^{n} a_i c(u_i, v_i, t)$, where $c(u, v, t)$ is some specified function. Now given points (t_i, f_i) from the graph of f, measured with error, we wish to determine the parameters $a_1, \ldots, a_n, u_1, \ldots, u_n, v_1, \ldots, v_n$, and perhaps n itself, such that best fits our data points (t_i, f_i).

In this tutorial sequence, we shall consider a special case of this problem, namely, that where $c(u, v, t)$ is the Gaussian peak function. Thus suppose $c(u, v, t) = \exp(-((t-u)/v)2)$, and consider the data:

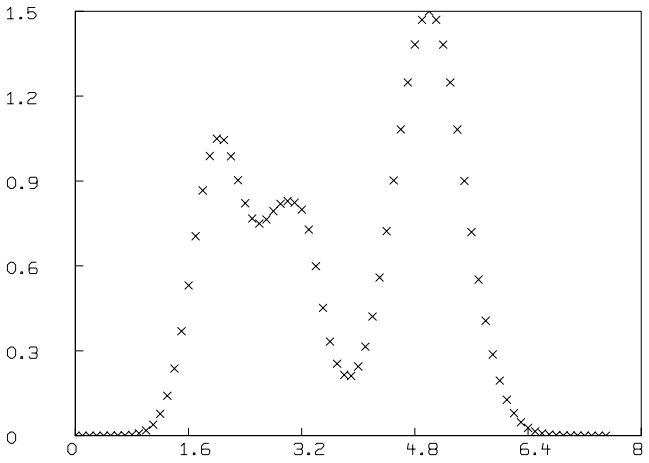
t	f	t	f	t	f
0	.11e-6	2.5	.76736	5.0	1.5000
.1	.53e-6	2.6	.74988	5.1	1.4697
.2	.235e-5	2.7	.76394	5.2	1.3824
.3	.954e-5	2.8	.79344	5.3	1.2483
.4	.3572e-4	2.9	.81926	5.4	1.0821
.5	.1234e-3	3.0	.82928	5.5	.90056
.6	.3938e-3	3.1	.82372	5.6	.71948
.7	.1159e-2	3.2	.79892	5.7	.55182
.8	.3152e-2	3.3	.72831	5.8	.40630
.9	.7911e-2	3.4	.59929	5.9	.28719
1	.1833e-1	3.5	.45159	6.0	.19488
1.1	.3920e-1	3.6	.33235	6.1	.12695
1.2	.7740e-1	3.7	.25461	6.2	.7939e-1
1.3	.14112	3.8	.21480	6.3	.4767e-1
1.4	.23758	3.9	.21129	6.4	.2747e-1
1.5	.36942	4.0	.24462	6.5	.1520e-1
1.6	.53075	4.1	.31495	6.6	.8075e-2
1.7	.70499	4.2	.42095	6.7	.4118e-2
1.8	.86680	4.3	.55913	6.8	.2016e-2
1.9	.98854	4.4	.72293	6.9	.9473e-3
2.0	1.0497	4.5	.90210	7.0	.4274e-3
2.1	1.0451	4.6	1.0828	7.1	.1851e-3
2.2	.98735	4.7	1.2486	7.2	.770e-4
2.3	.90278	4.8	1.3825	7.3	.307e-4
2.4	.82160	4.9	1.4697	7.4	.118e-4
				7.5	.433e-5

Let us begin by entering the data. We type:

```
*D = KREAD(76)
```

When MLAB responds, we will enter, in order, all the numbers in the "f" columns above, separated by blanks. We must strike carriage return as needed. This is a bit tedious, but bearable. (Alternatively, we may put our data in a text-file via the MLAB EDIT command, or by any other desired means, and then read the file using the MLAB `READ` operator.) Now we will type the following.

```
*TX = 0:7.5:.1
*D = TX&'D
*DRAW D LINETYPE none POINTTYPE XPT PTSIZE .01
*VIEW
```

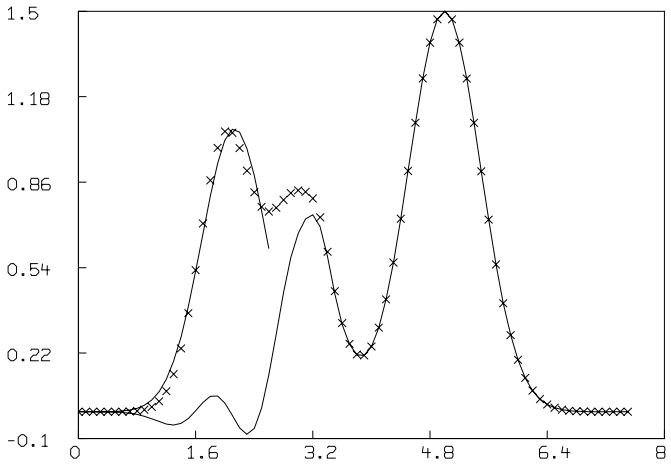


Now we shall define our model function F. Note how the summation bounds, G and N, are initially set.

```
*FUNCTION C(T) = A1*EXP(-((T-U1)/V1)^2)
*FUNCTION F(T) = SUM(I,G,N,A[I]*EXP(-((T-U[I])/V[I])^2))
*G = 1
*N = 0
```

Fitting sums of Gaussians to data is a tricky affair. One approach is to fit the major peaks seen in the data, one by one from left to right, say, and check the difference curve at each step. Our first peak seems to be at $t = 2.1$. Thus, taking data to cover that range we may fit our first peak as follows. After fitting, we will look at the difference between our data and the single peak best-fit model curve.

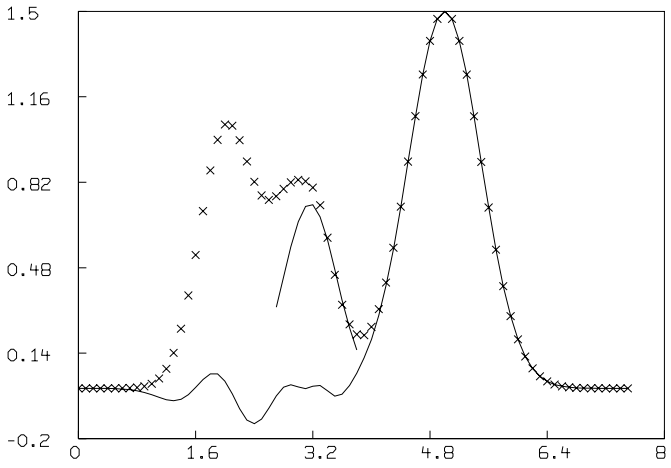
```
*A1 = D[22,2]
*U1 = D[22,1]
*V1 = 1
*FIT(A1,U1,V1), C to D ROW 1:27

final parameter values
      value          error           dependency    parameter
    1.060558658   0.02130292129     0.3571362089   B1
    2.129693372   0.01241488141     0.1802563756   U1
   0.6332395033   0.01957979132        0.4523625   S1
4 iterations
CONVERGED
best weighted sum of squares = 5.27967e-02
weighted root mean square error = 4.69027e-02
weighted deviation fraction = 6.05651e-02
R squared = 9.88524e-01

*N = N+1
*A(1) = A1; U[1] = U1;V[1] = V1
*D1 = TX&'((D COL 2)-(F ON TX))
*DRAW CV = D1
*DRAW CP = POINTS(C,TX) ROW 1:27
*VIEW
```



Upon viewing the difference curve, we may guess that the second peak is at $t = 3$. The apparent peak at $t = 1.8$ is too small to deal with now. Thus, we may fit for the next major peak as follows:

```
*A1 = D[31,2]
*U1 = D[31,1]
*V1 = 1
*FIT(A1,U1,V1), C to (D1 ROW 28:39)

final parameter values
      value          error           dependency    parameter
   0.7341057505   0.01404712323       0.3891979759   B1
    3.161955267   0.008360292672     0.03801268929   U1
   0.5095982134   0.01422069167        0.407114449   S1
7 iterations
CONVERGED
best weighted sum of squares = 6.75526e-03
weighted root mean square error = 2.73968e-02
weighted deviation fraction = 2.37131e-02
R squared = 9.83382e-01

*N = N+1
*A[N] = A1;U[N] = U1;V[N] = V1
*D1 = TX&'((D COL 2)-F ON TX)
*DRAW CV = D1
```

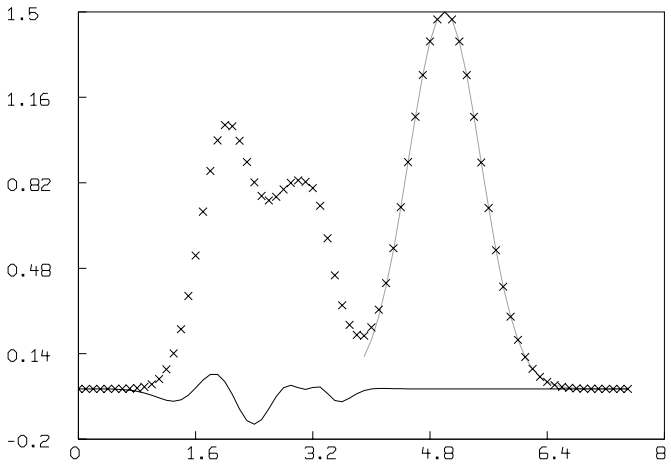
```
*DRAW CP = POINTS(C,TX) ROW 28:39
*VIEW
```



Now the difference curve shows the next large peak to be at $t = 5$. There may be other peaks at $t = 1.8$ or 3 but we cannot be sure. We will now proceed to eliminate the third major peak at $t = 5$.

```
*A1 = D[51,2]
*U1 = D[51,1]
*V1 = 2
*FIT (A1,U1,V1), C to (D1 ROW 40:76)

final parameter values
      value          error           dependency    parameter
    1.499986167   0.0005350719899     0.3386395982   B1
    4.999677239   0.0002037921758  0.0002332067344   U1
   0.7003779925   0.0002924826921     0.3387556571   S1
5 iterations
CONVERGED
best weighted sum of squares = 5.64817e-05
weighted root mean square error = 1.28889e-03
weighted deviation fraction = 3.52845e-04
R squared = 9.99995e-01
```

```
*N = N+1
*A[N] = A1; U[N] = U1;V[N] = V1
*D1 = TX&'((D COL 2)-F ON TX)
*DRAW CV = D1
*DRAW CP = POINTS(C,TX) ROW 40:76
*VIEW
```



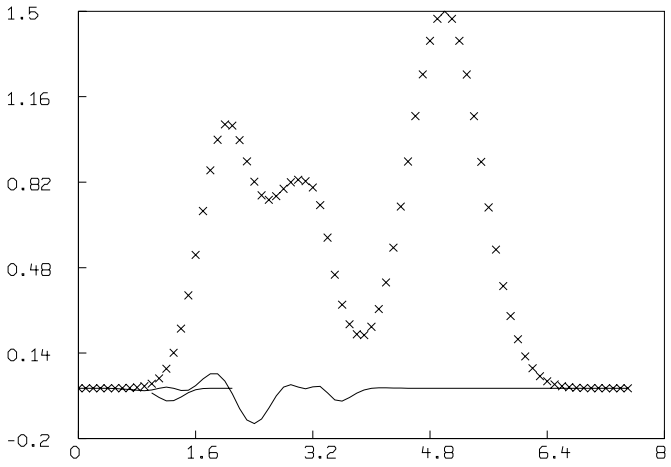
There is no foolproof way to decompose a sum of Gaussians into its components. For example, the small "spike" at $t = 1.8$ might be a peak, or it could merely be noise. Let us suppose there is a peak there, and add it to our ensemble.

```
*A1 = D1[10,2]
*U1 = D1[10,1]
*V1 = 2
*FIT(A1,U1,V1), C to D1 ROW 11:22

matherr: underflow error in exp
         arg = -4.20509e+03
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -2.88606e+03
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -1.81455e+03
```

```
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -9.90549e+02
         return value: 0.00000e+00
matherr: underflow error in exp
         arg = -1.24428e+03
         return value: 0.00000e+00
final parameter values
      value          error           dependency    parameter
 -0.05160238476    0.02327471342      0.3579202025   A1
    1.244999038    0.09344315243     0.01484083378   U1
   0.2442087571      0.142601028      0.3652390741   V1
22 iterations
CONVERGED
best weighted sum of squares = 9.63528e-03
weighted root mean square error = 3.27198e-02
weighted deviation fraction = 5.64757e-01
R squared = 4.59633e-01
```

Note the underflow errors reported above are harmless. They occured because, during curve-fitting, we gave an argument to the exponential function embedded in the model function c which led to a number too small to be represented. Zero was used for this value and the calculation continued successfully.

```
*N = N+1
*A[N] = A1;U[N] = U1;V[N] = V1
*D1 = TX&'((D COL 2)-F ON TX)
*DRAW CV = D1
*DRAW CP = POINTS(C,TX) ROW 11:22
*VIEW
```



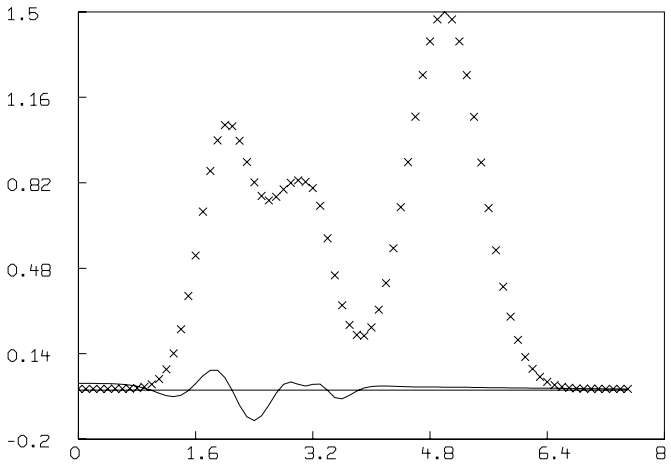
This peak is fit by an almost-flat curve. Thus, we shall not include it, even though a sharper peak might actually be present.

```
*DELETE A ROW N, U ROW N, V ROW N;
*N = N-1
*D1 = TX&'((D COL 2)-F ON TX)
```

There seems to be a slight linear rise in the difference curve. Let us assume a linear baseline correction term is appropriate. Thus, we will type:

```
*FUNCTION Y(T) = P*T+Q
*P = .05; Q = 0
*FIT(P,Q), Y to D1

final parameter values
      value          error           dependency    parameter
  0.00309195025   0.001692261267     0.7450331126   P
 -0.01693381919   0.007352091349     0.7450331126   Q
2 iterations
CONVERGED
best weighted sum of squares = 7.75088e-02
weighted root mean square error = 3.23638e-02
weighted deviation fraction = 9.34090e-01
```

```
R squared = 4.31654e-02

*D1 = D1-(0&'(Y ON (D COL 1)))
*DRAW CV = D1
*DRAW CP = POINTS(C,TX)
*view
```



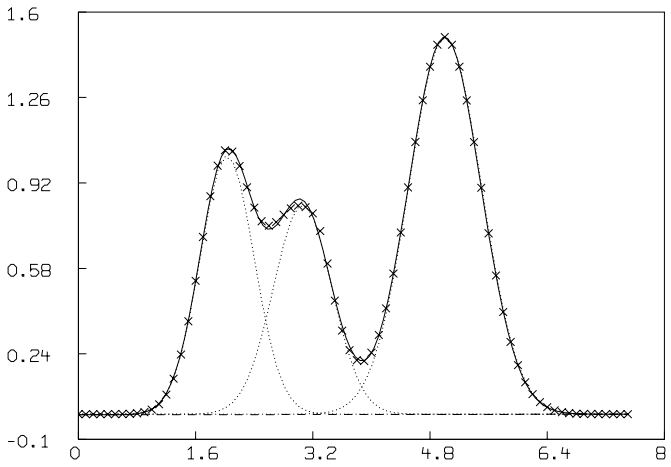
Now we shall modify our model to incorporate the baseline correction and then fit the grand ensemble. Note the matrix parameters being used in the fit command!

```
*FUNCTION F1(T) = F(T)+Y(T)
*FIT(A,U,V,P,Q), F1 to D

final parameter values
      value          error          dependency    parameter
    1.027192982   0.005189389666     0.5889517841   A[1]
   0.8337813868   0.004513925746      0.496821345   A[2]
     1.50138784    0.00382828971     0.4375886466   A[3]
    2.023177649    0.00396324563      0.818228814   Z[1]
    3.053968715   0.005164260712      0.824329662   Z[2]
    4.999939042   0.001392999253    0.07176843552   Z[3]
   0.5213274709   0.004748894868     0.8314409908   V[1]
   0.5627748271   0.006356665639     0.8458505343   V[2]
   0.7000560594   0.002283300581     0.5416655576   V[3]
```

```
0.0002937414279  0.0005374471602      0.825464455   P
-0.002455080808   0.002732830484     0.8725860427   Q
4 iterations
CONVERGED
best weighted sum of squares = 4.70077e-03
weighted root mean square error = 8.50409e-03
weighted deviation fraction = 5.86288e-03
R squared = 9.99710e-01
```

Now we may draw the final fit and the corresponding decomposition into individual Gaussians.

```
*DRAW CV = POINTS (F1,0:7.5!120)
*DRAW CP = POINTS (Y,0:7.5!2) LINETYPE dashed
*FOR G = 1:N DO {N = G; DRAW POINTS(F,0:7.5!120),LINETYPE dotted}
*VIEW
```



This is an excellent fit. Still, we may try to decide if the peak we saw at 1.8 is spurious. In general, this is difficult to do, but one way is to fit both with and without the suspect peak and consider whether or not the total sum of squares is materially different. The details of this process are left as an exercise. Please proceed on your own and fit the model with the suspect peak Draw the final decomposition and the final difference curve. Do you think the peak at 1.8 is "real"? (The data was generated by adding normal random errors to a particular sum of Gaussians. You will never know, however, from this analysis whether the peak at 1.8 was there or not!) Determining

peaks can be aided by using specialized machinery. A good reference is: Eklundh, Jan-Olof and Rosenfeld, A., Peak Detection Using Difference Operators, U. of Md. Computer Science Dept. report TR-651, May 1978.

22 Estimating Weights for Curve-Fitting

It is usually appropriate and often necessary to fit a model to data with associated weight values. The weight for an observed value, y, should be $1/var(Y)$ where y is a sample of the random variable Y. Using the correct weights automatically gives each observation in each separate data set the expected right over-all weight, even if the expected sizes of observations vary a great deal within or among various data sets.

In order to determine the weights for a data set of observations given as the rows of a matrix, M, you can use the EWT operator provided in MLAB when M has two or three columns (EWT stands for estimated weights).

The EWT operator can be defined in terms of other MLAB operations. Indeed the following expository description of EWT is useful in learning about EWT and various other MLAB operators.

The idea is to estimate the standard deviation, s_y, of each observation, y, as some value $D(y)$, and then use $1/D(y)^2$ as the associated weight value. The value $D(y)$ can be based on the error in the observation y. But the deviations or errors in 2- or 3-dimensional data appearing as the rows of a 2-column or 3-column matrix, M, can be estimated by computing `(M COL c) - (F ON (M COL 1:(c-1)))` where `c=ncols(m)`, and F is the correct model function for the data M. F is unknown, so we estimate `(F ON M COL 1:(c-1))` by smoothing the data points, M.

This idea assumes the noise in the data is predominantly white; so that most of the noise power is at high-frequencies which are filtered by the smoothing operation.

In order to demonstrate the calculation of $EWT(M)$, let us construct a specific two-dimensional example matrix to be employed in the demonstration. Suppose we pick the model function to be $F(x) = 3\sin(x) - x + 5$. Then we can generate a two-column data matrix M, with normally-distributed proportional 20% error, as follows.

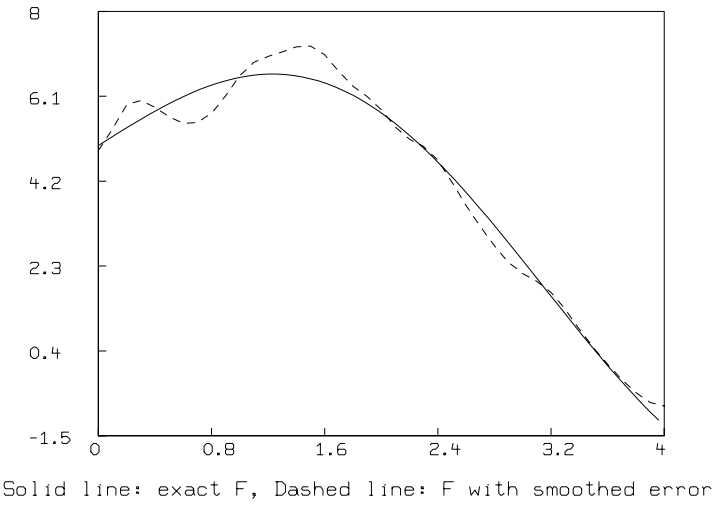
```
*FUNCTION F(X) = 3*SIN(X)-X+5
*M = POINTS (F,0:4:.06)
*E = NORMRAN ON 0^^NROWS(M)
*E = E*'(.2*ABS ON M COL 2)
*M COL 2 = (M COL 2) + E
```

Now we begin by computing the vector of the estimated errors, D, with:

```
*D = ABS ON ((M COL 2) - (SMOOTH(SMOOTH(M)) COL 2))
```

By drawing the following graph, we may see how well `SMOOTH(SMOOTH(M))` estimates `POINTS(F,0:4:.06)`.

```
*DRAW POINTS (F,0:4:.06)
*DRAW SMOOTH(SMOOTH(M)), LINETYPE DASHED
*BOTTOM TITLE "Solid line: exact F, Dashed line: F with smoothed error"
*BOTTOM TITLE SIZE .018
*VIEW
```



Solid line: exact F, Dashed line: F with smoothed error

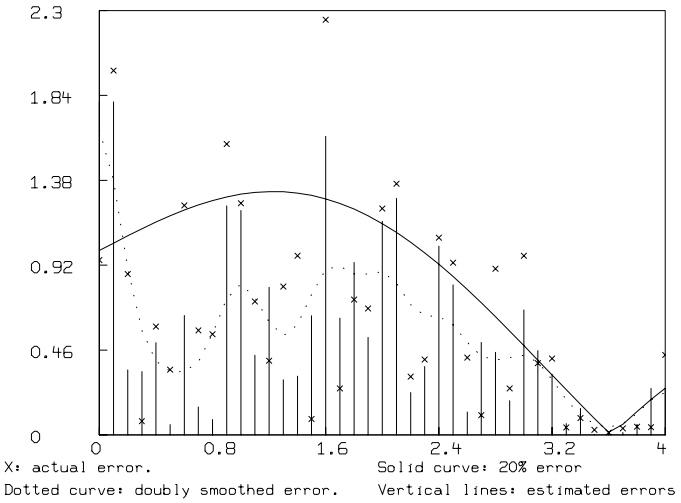
Now the first and last values in the vector of estimated errors D are always zero. It is more appropriate to let D_1 and D_N inherit their neighbor values. Also we shall have need of the average estimated error, S, which we compute below.

```
*N = NROWS(D)
*S = MEAN(D)
*S = IF S = 0 THEN 1 ELSE S
*D[1] = D[2]
*D[N] = D[N-1]
```

These estimates of the standard deviations are themselves very noisy, so we shall smooth them to obtain our final estimates. A graph of the true standard deviations and our unsmoothed and final estimates is drawn below.

```
*DELETE W
*DRAW MESH(M COL 1,M COL 1)&'MESH(D,0^^N), LINETYPE ALTERNATE
*FUNCTION SD(X) = .2*ABS(F(X))
```

```
*DRAW POINTS(SD,M COL 1)
*D = SMOOTH(SMOOTH((M COL 1)&'D))
*DRAW D, LINETYPE DOTTED
*DRAW (M COL 1) &' ABS(E), LINETYPE NONE, POINTTYPE XPT PTSIZE .007
*TITLE "X: actual error. Solid curve: 20% error.", SIZE .013 AT (.05,.045)
*TITLE "Dotted curve: doubly-smoothed error. Vertical lines: estimated errors.",\
        SIZE .013 AT (.05,.02)
*VIEW
```

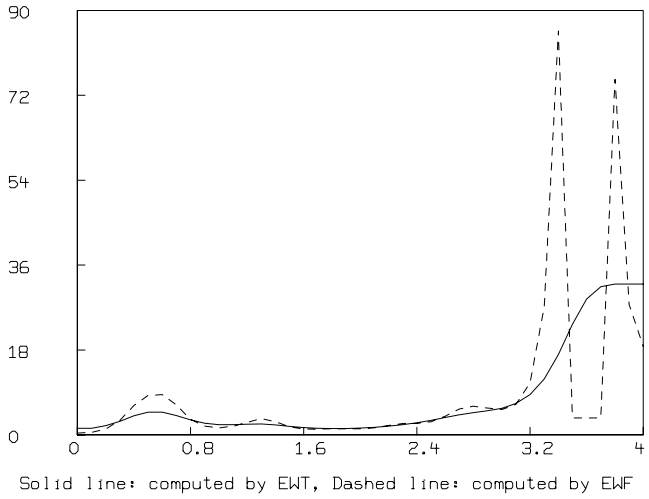


The resulting weight vector, *EW*, is computed as follows.

```
*FUNCTION u(x) = if x < s then x+(s-x)/4 else x
*FUNCTION EWF(X) = 1/u(x)^2
*EW = EWF ON (D COL 2)
```

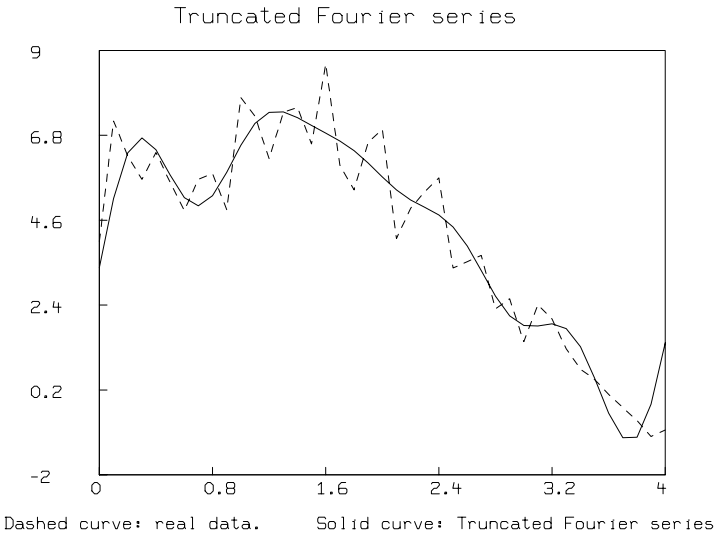
The EWT operator embodies the process shown by the example above. Let us compare EW and ewt(m).

```
*DELETE W
*DRAW (M COL 1)&'EWT(M)
*DRAW (M COL 1)&' EW LINETYPE DOTTED
*BOTTOM TITLE "Solid line: computed by EWT, Dashed line: computed by EWF"
*BOTTOM TITLE SIZE .015
*VIEW
```



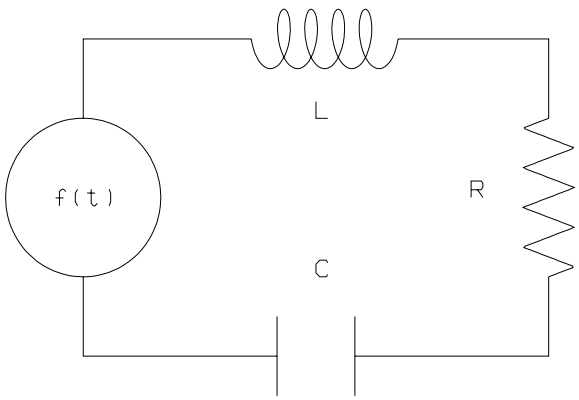
Rather than use `SMOOTH`, it is sometimes better to use explicit lowpass filtering by estimating `(F on M COL 1)` as a truncated Fourier series. This can be done as follows.

```
*R = REALDFT(M)
*FUNCTION EF(X) = SUM(I,1,P,R[I,2]*COS(R[I,1]*PI*2*X+R[I,3]))
*P = 6
*Q = POINTS(EF,M COL 1)
*DELETE W
*DRAW M LINETYPE DOTTED
*DRAW Q
*TOP TITLE "Truncated Fourier series" size .02
*BOTTOM TITLE "Dashed curve: real data, Solid curve: Truncated Fourier series", SIZE .015
*VIEW
```


Truncated Fourier series
9
6.8
4.6
2.4
0.2
-2
0
0.8
1.6
2.4
3.2
4
Dashed curve: real data.
Solid curve: Truncated Fourier series

23 Electric Circuit Dynamics

This example demonstrates MLAB's differential equation-solving facilities, the use of MLAB's Fourier transform operations, and MLAB graphics in the context of the classic problem of analyzing an LRC circuit. A circuit containing a coil with an inductance of L henries, a resistor with a resistance of R ohms, and a capacitor with a capacitance of C farads in series is traditionally called an LRC circuit. Consider the LRC circuit shown below which also contains a voltage source component which exhibits a voltage drop of $-f(t)$ at time t across the voltage source component. (This picture was constructed using MLAB.)



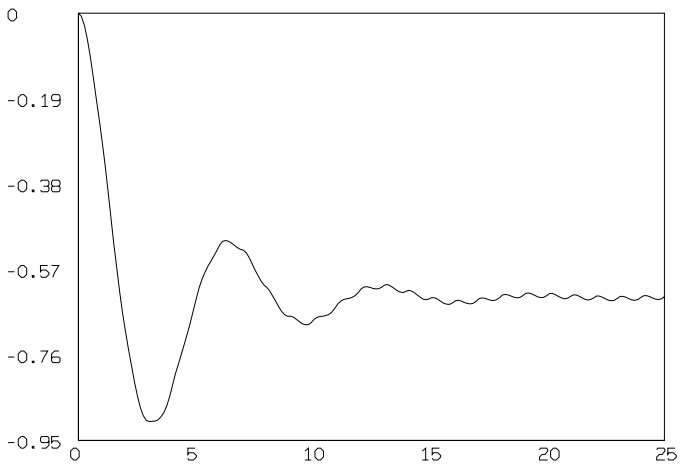
When the switch is closed at time 0, a current will begin to flow. Let $I(t)$ be the current flowing in the circuit at time t, measured in amperes. Let $Q(t)$ be the charge on the capacitor at time t, measured in coulombs. We have $dQ(t)/dt = I(t)$.

The voltage drop across the resistor at time t is given by Ohm's law as $R \cdot I(t)$. The voltage drop across the capacitor at time t is $Q(t)/C$. The voltage drop across the coil at time t is $L(dI(t)/dt)$, and by Kirchhoff's first law, the sum of the voltage drops across each of the circuit components is 0. Thus, we have

$$\begin{aligned} L\frac{dI(t)}{dt} + \frac{Q}{C} + RI(t) - f(t) &= 0, \qquad \text{or} \\ L\frac{d^2I(t)}{dt^2} + R\frac{dI(t)}{dt} + \frac{I(t)}{C} - \frac{df(t)}{dt} &= 0. \end{aligned}$$

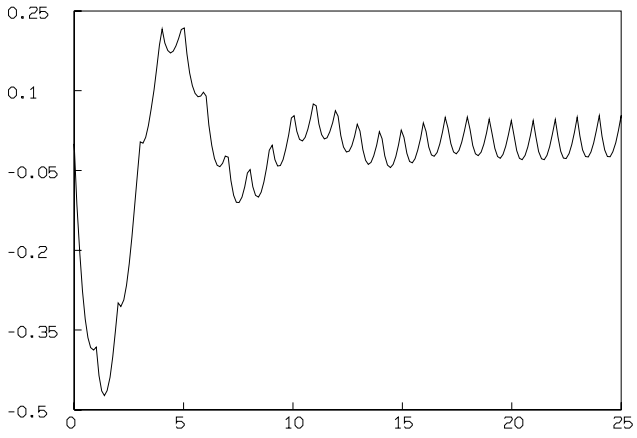
Take the initial conditions to be $I(0) = 0$ and $dI(0)/dt = 0$, and define $f(t) = \exp(-(t \bmod 1))$. Fix $L = 1$, $C = 1$, and $R = .5$. Now we may solve the differential equation defining the current flow function $I(t)$ and produce a graph of this function as shown below.

```
*function i''t(t)=(f't(t)-r*i't -i/c)/l
*initial i(0) = 0
*initial i't(0) = 0
*function f(t)=exp(-mod(t,1))
*l=1; c=1; r=.5
*m = integrate(i't,i''t,0:25!200)
*type odestr
   odestr = t i't i't't i i't
*draw m col (1,4)
*view
```



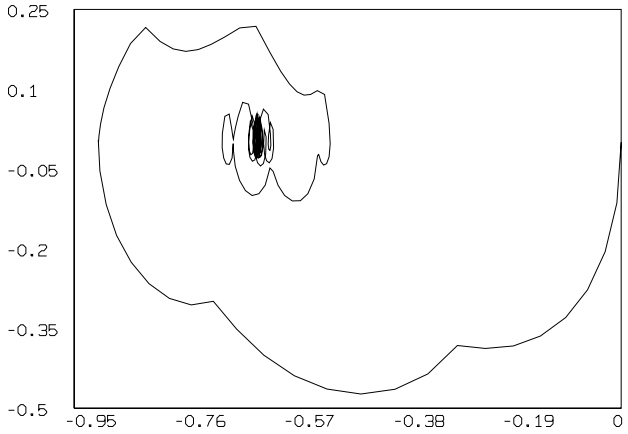
The value of the string variable ODESTR tells us what functions are numerically tabulated in the successive columns of M. The graph of the rate-of-change of the current function $I't(t)$ can also be plotted from the data in M.

```
*delete w
*draw m col 1:2
*view
```



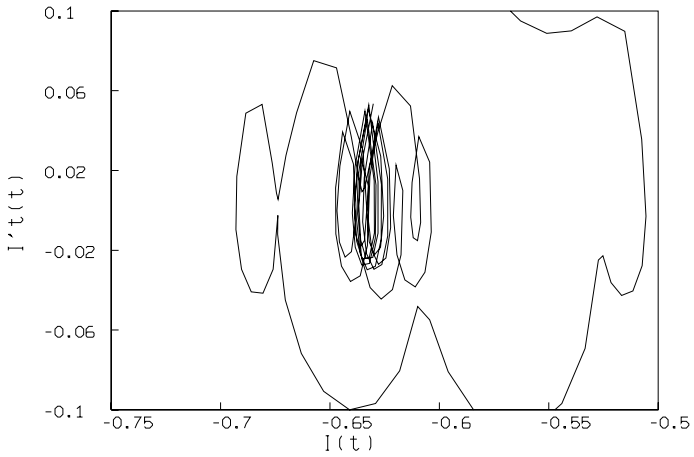
We can display the phase diagram graph for $I(t)$ by plotting $I't(t)$ vs. $I(t)$ as follows

```
*delete w
*draw m col (4,2)
*view
```



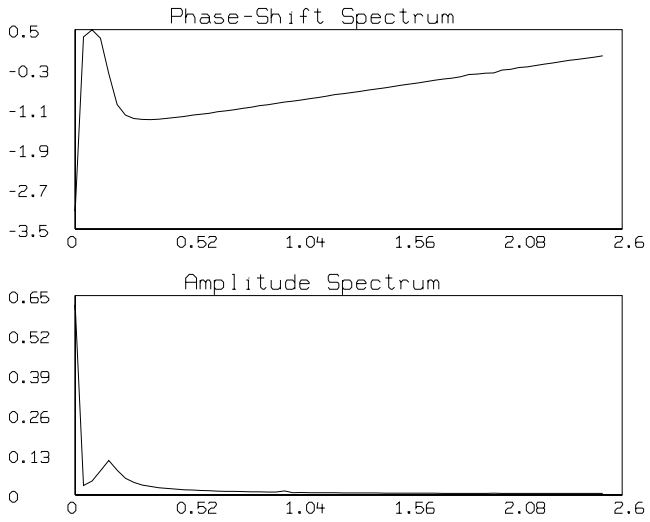
We may "zoom-in" to see the neighborhood of the limit cycle more clearly as follows.

```
*WINDOW -.75 TO -.5, -.1 TO .1
*VIEW
```



We can use the particular MLAB Fourier transform operator `realdft` to compute the amplitude and phase-shift spectra of the current function $I(t)$ tabulated in `M COL (1,4)`.

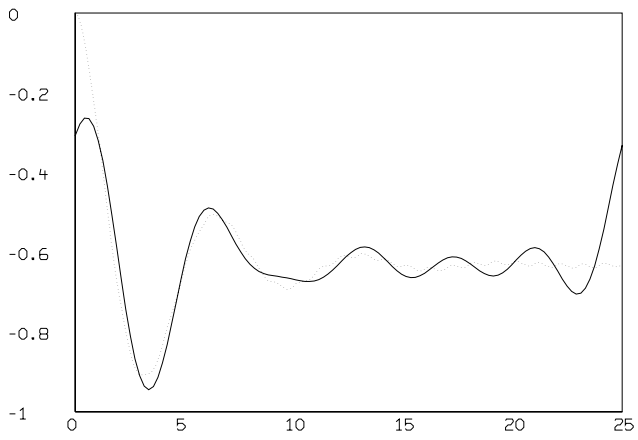
```
*d=realdft(m col(1,4))
*delete w
*draw d col 1:2
*frame 0 to 1, 0 to .5
*top title "Amplitude Spectrum" size .2 inches
*w1=w

*draw d col(1,3)
*frame 0 to 1, .5 to 1
*top title "Phase-Shift Spectrum" size .2 inches
*view
```



Ignoring the large DC value at frequency zero, we see that the maximum amplitude occurs at about .2 Hertz; this is the resonant frequency of the circuit. To see the amplitude spectrum at higher "resolution", we should subtract the DC-value from our signal $I(t)$ and compute the amplitude spectrum of this shifted signal whose mean value is now zero.

The Fourier transform of $I(t)$ contains the information to construct $I(t)$ as a periodic function via its Fourier series. If the Fourier series is truncated, the resulting sum is a filtered form of $I(t)$ omitting the high-frequency components corresponding to the truncated terms. Below we show a graph of the Fourier series of $I(t)$ truncated to 7 terms. Note that Gibbs' phenomenon is exhibited, showing non-uniform convergence to the mid-point of the discontinuities occuring at the points between successive periods of $I(t)$.

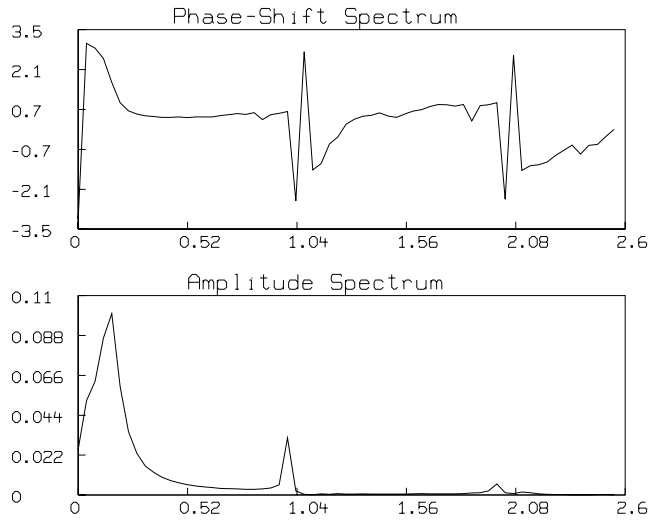
```
*fct s(t)=sum(i,1,n, d(i,2)*cos(2*pi*d(i,1)*t + d(i,3)) )
*n=7
*q=points(s,0:25!120)
*delete w,w1
*draw m col 1:2 lt dotted
*draw q
*view
```



We can also use the MLAB Fourier transform operator to compute the amplitude and phase-shift spectra of the current rate-of-change function $I't(t)$ tabulated in `M COL (1,2)`.

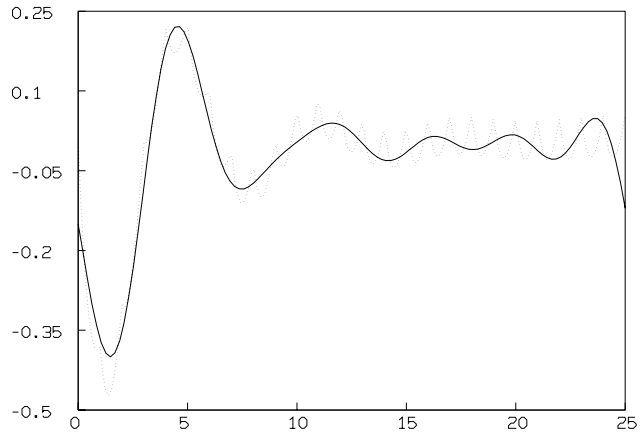
As above, this amplitude spectrum shows the resonant frequency of the circuit to be about .2 Hertz. The peak at 1 Hertz corresponds to the frequency of oscillation of the forcing function $f(t)$.

```
*d=realdft(m col 1:2)
*delete w,w1
*draw d col 1:2
*frame 0 to 1, 0 to .5
*top title "Amplitude Spectrum" size .2 inches
*w1=w

*draw d col(1,3)
*frame 0 to 1, .5 to 1
*top title "Phase-Shift Spectrum" size .2 inches
*view
```



Just as before, the Fourier transform of $I't(t)$ contains the information to construct $I't(t)$ via its Fourier series. If the Fourier series is truncated, the resulting sum is a filtered form of $I't(t)$ omitting the high-frequency components corresponding to the truncated terms. Below we show a graph of the Fourier series of $I't(t)$ truncated to 7 terms, superimposed on a graph of $I't(t)$ plotted as a dotted line.

```
*fct s(t)=sum(i,1,n, d(i,2)*cos(2*pi*d(i,1)*t + d(i,3)) )
*n=7
*q=points(s,0:25!120)
*delete w,w1
*draw m col 1:2 lt dotted
*draw q
*view
```



24 Amortization Schedule Computation

One aspect of MLAB is that it is a spreadsheet for the mathematically literate. For example, the MLAB statement sequence to construct a level payment schedule for a loan of a dollars at r percent interest, to be repaid over n months is given below in a do-file called pay.do. For example, to compute the level payment schedule for $9,000 to be repaid over 16 months at 9.5% annual interest , we type: `do pay` , and enter the values 9000,16, and 9.5 in response to the corresponding prompts. The payment schedule shown below is produced. This do-file can also be used to compute any of either the loan amount, the number of months of the loan, the annual percentage interest rate, or the monthly payment, where the other three values are known.

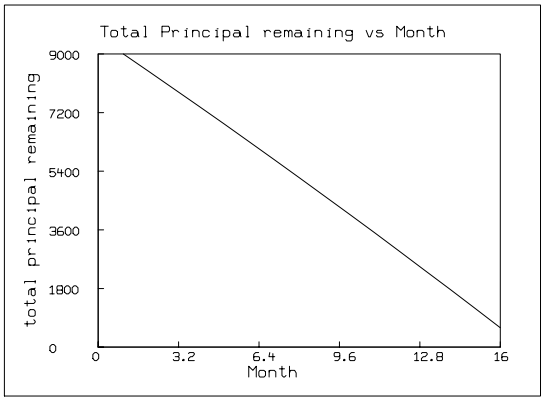
```
"filename:pay.do = computation of amortization schedule."
reset
echodo=0; namesw=0

type "You will be requested to enter 4 quantities:"
type "a=loan amount,"
type "n=number of months of the loan,"
type "r=annual percentage interest rate, and"
type "q=monthly payment amount."
type "If any one of these input-values is specified to be -1,"
type "then MLAB will compute that value based on the other values."
a=kread("enter loan amount a=");
n=kread("enter number of months n=");
r=kread("enter annual percentage interest rate r=")/1200;
q=kread("enter monthly payment amount q=");

function p(j)=a*(1+r)^j+q*(1-(1+r)^j)/r; "total principal due at month j";
function av()=q*((1+r)^n-1)/(r*(1+r)^n); "loan amount";
function nv()=log(q/(q-r*a),1+r); "number of months of the loan"
function rv()=root(r,1e-6,100,p(n)); "monthly interest rate";
function qv()=a*(1+r)^n/((1+r)^n-1)*r; "monthly payment amount";

if a<0 then a=av(); /*compute the loan amount*/
if n<0 then n=nv(); /*compute the number of months of the loan*/
if r<0 then r=rv(); /*compute the monthly interest rate*/
if q<0 then q=qv(); /*compute the monthly payment amount*/

type " "
type "the loan amount:"+a
type "the number of months of the loan:"+n
type "the annual percentage interest rate:"+(r*1200)
```

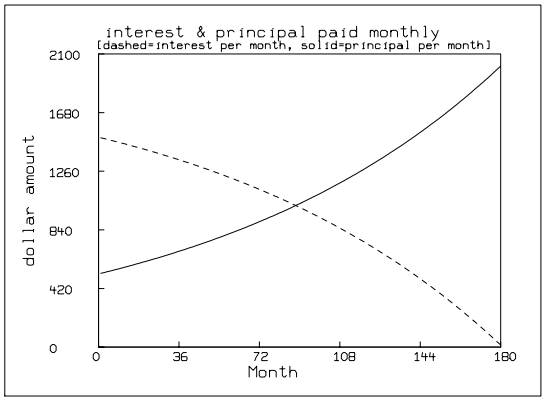
```
type "the monthly payment amount:"+q

d=p on 0:(n-1); /*compute the principal due */
i=r*d; /*compute the interest payment*/
nl=1:n
m=nl&'d&'i&'(q-i)&'q
type " "; type "Do you want to print out the payment schedule?"
y=kread("type 1 for YES, 0 for NO:");
if y=1 then \
{type "   [prin. due | int. paid  |  prin. paid  | monthly pay.]",m}

type "Total interest paid:"+(n*q-a)
type "Total amount paid:"+(n*q)

draw nl&'d
top title "Total Principal remaining vs Month"
left title "total principal remaining"
bottom title "Month"
yaxis w.yaxis format (-3,6,0,0,2,0)
view
w1=w; blank w1

draw nl&'i lt dashed
draw nl&'(q-i)
top title "interest & principal paid monthly"
left title "dollar amount"
bottom title "Month"
title "[dashed=interest per month, solid=principal per month]" \
 at (0,1.018) ifract size .14 inches
view
```

Here is an example of running `pay.do`

```
* do pay
     You will be requested to enter 4 quantities:
     a=loan amount,
     n=number of months of the loan,
     r=annual percentage interest rate, and
     q=monthly payment amount.
     If any one of these input-values is specified to be -1,
     then MLAB will compute that value based on the other values.
     enter loan amount a=  9000
     enter number of months n=  16
```

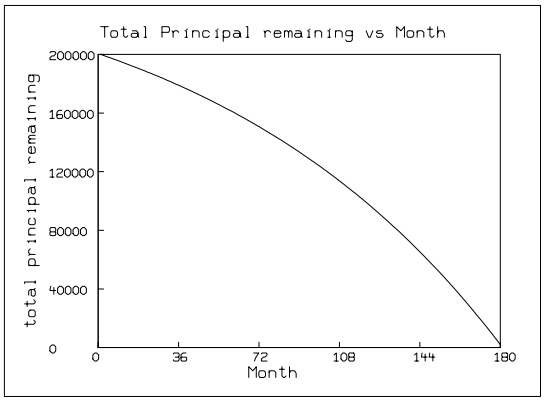
```
     enter annual percentage interest rate r=  9.5
     enter monthly payment amount q=  -1

     the loan amount: 9000
     the number of months of the loan: 16
     the annual percentage interest rate: 9.5
     the monthly payment amount: 601.09756

     Do you want to print out the payment schedule?
type 1 for YES, 0 for NO  1
        [prin. due | int. paid  |  prin. paid  | monthly pay.]
1    9000         71.25        529.84756    601.09756
2    8470.15244   67.0553735   534.042186   601.09756
3    7936.11025   62.8275395   538.27002    601.09756
4    7397.84023   58.5662352   542.531324   601.09756
5    6855.30891   54.2711955   546.826364   601.09756
6    6308.48255   49.9421535   551.155406   601.09756
7    5757.32714   45.5788399   555.51872    601.09756
8    5201.80842   41.1809833   559.916576   601.09756
9    4641.89184   36.7483104   564.349249   601.09756
10   4077.5426    32.2805455   568.817014   601.09756
11   3508.72558   27.7774109   573.320149   601.09756
12   2935.40543   23.2386263   577.858933   601.09756
13   2357.5465    18.6639098   582.43365    601.09756
14   1775.11285   14.0529767   587.044583   601.09756
15   1188.06827   9.40554044   591.692019   601.09756
16   596.376248   4.72131196   596.376248   601.09756
     Total interest paid: 617.560952
     Total amount paid: 9617.56095
* exit
```

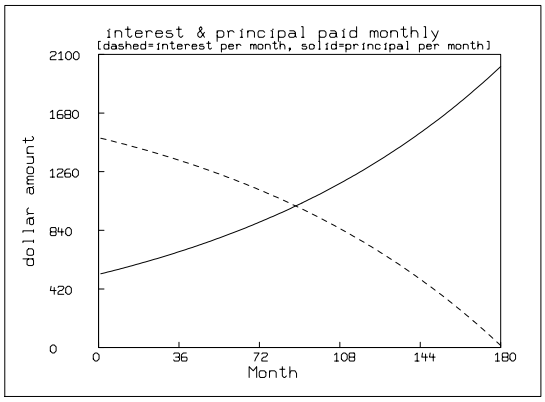




Here is another example of running `pay.do`

```
* do pay
     You will be requested to enter 4 quantities:
     a=loan amount,
     n=number of months of the loan,
     r=annual percentage interest rate, and
     q=monthly payment amount.
     If any one of these input-values is specified to be -1,
     then MLAB will compute that value based on the other values.
     enter loan amount a=  200000
     enter number of months n=  180
     enter annual percentage interest rate r=  9
     enter monthly payment amount q=  -1

     the loan amount: 200000
     the number of months of the loan: 180
     the annual percentage interest rate: 9
     the monthly payment amount: 2028.53317

     Do you want to print out the payment schedule?
type 1 for YES, 0 for NO  0
     Total interest paid: 165135.97
     Total amount paid: 365135.97
* exit
```





25 Buoyancy-Driven Boundary-Layer Flow over a Vertical Plate

In the area of heat transfer and fluid flow, it commonly occurs that the fluid adjacent to a vertical heated plate rises due to the buoyancy force corresponding to the difference in pressure below and above the heated fluid in the vicinity of the vertical plate. This pressure difference is due to the gravity-induced density-gradient in the fluid, taking into account the lower density of the heated fluid; in the absence of gravity, a hot ball of fluid has no tendency to move.

Let us place the origin at the bottom of the vertical plate, with the plate extending along the vertical x-axis of a left-handed coordinate system. Generally, the heated fluid will flow such that the streamwise flow velocity component in the vertical x-direction is much greater than the transverse velocity in the horizontal y-direction (since the y-direction motion is primarily due to the stream deflection caused by the boundary layer of fluid accreted along the plate.) Also, the y-direction gradient, $\partial\phi/\partial y$, of a field variable ϕ is much greater than the x-direction counterpart. As a result, most of the flow activity takes place within a thin region adjacent to the plate. The velocity of the heated fluid is small immediately adjacent to the heated plate due to the "friction" associated with the thermal noise in the heated fluid; thus we have non-uniform-velocity laminar flow about the heated plate. As the horizontal distance from the plate increases, the induced velocity rises; and as the horizontal distance increases still more, the induced velocity diminishes due to the increasingly-small pressure difference in excess of the force of gravity.

Nondimensional continuum partial-differential equations governing the boundary-layer flow driven by the buoyancy force mentioned above can be written as

mass conservation (continuity):

$$\partial u/\partial x + \partial v/\partial y = 0,$$

x-direction momentum conservation:

$$u\partial u/\partial x + v\partial u/\partial y = (\partial^2 u/\partial y^2)/re - \partial p/\partial x + g_x(1/\rho - 1),$$

y-direction momentum conservation:

$$u\partial v/\partial x + v\partial v/\partial y = (\partial^2 v/\partial y^2)/re - \partial p/\partial y + g_y(1/\rho - 1),$$

energy conservation:

$$u\partial T/\partial x + v\partial T/\partial y = \partial^2 T/\partial y^2/pe.$$

The relevant quantities in these equations are defined below.

$u(x, y)$ = dimensionless x-direction flow velocity at the point (x, y).

$v(x,y)$ = dimensionless y-direction flow velocity at the point (x,y).

$\rho(x,y)$ = dimensionless density at the point (x,y).

u_0 = reference flow-velocity value used for dimensionless conversion.

ρ_0 = reference density value far from the plate.

T_0^* = reference temperature value far from the plate.

g_x^* = acceleration due to gravity in the vertical x-direction.

g_y^* = acceleration due to gravity in the horizontal y-direction $(= 0)$.

$g_x = g_x^* L/u_0^2$ = dimensionless x-direction gravity acceleration.

$g_y = g_y^* L/u_0^2$ = dimensionless y-direction gravity acceleration.

$p^*(x,y)$ = the pressure in the fluid at the point (x,y).

$p(x,y)$ = dimensionless pressure, $(p^*(x,y) + \rho_o g_x L y)/(\rho(x,y)u_o^2)$ at the point (x,y).

α = thermal diffusivity.

ν = kinematic viscosity.

pe = Peclet number, $u_0 L/\alpha$.

re = Reynolds number, $u_0 L/\nu$.

$T^*(x,y)$ = the temperature in the fluid at the point (x,y).

$T(x,y) = (T^*(x,y) - T_0^*)/(T^*(x,0) - T_0^*)$.

It is possible and sometimes convenient to transform these partial differential equations into two ordinary differential equations [1].

Let $s := y(gr/(4x))^{1/4}$ where gr is the Grashof number $g_x^* L(T^*(0,0) - T_0^*)/(u_0^2 T_0^*)$. Now define $f(s) := \psi(x,y)/\psi_0(x)$, where the streamfunction $\psi(x,y)$ is defined to be $\int_0^y u(x,\delta)d\delta$ and $\psi_0(x) := 4\nu(gr/4)^{1/4}x^{3/4}$. The function f is a dimensionless streamfunction whose derivative f' is the x-velocity of the fluid in arbitrary units at each point (x,y) in the vicinity of the heated plate that satisfies $s = y(gr/(4x))^{1/4}$. Also define $h(s) := T(1, s/(gr/4)^{1/4})$. The dimensionless temperature defined by h is the constant value $h(s)$ along the curve defined by $y(gr/(4x))^{1/4} = s$, where the x-velocity is similarly constant.

With these definitions, and knowing that $u = \partial\psi/\partial y$ and $v = -\partial\psi/\partial x$, we can follow Ostrach [1] to obtain the following system of differential equations. The symbol pr denotes the Prandtl number, ν/α.

$$f''' + 3ff'' - 2(f')^2 + h = 0,$$

$$h'' + 3prfh' = 0,$$

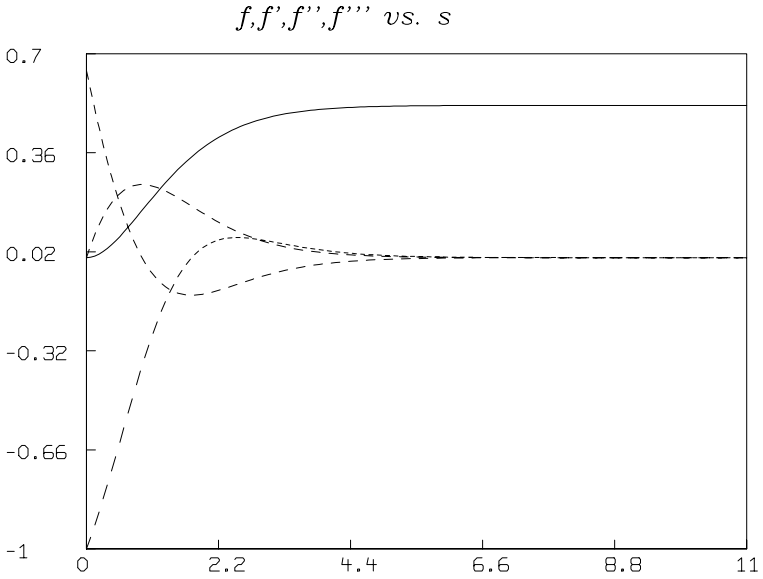
subject to $f(0) = f'(0) = 0, h(0) = 1$, and $f'(\infty) = 0$ and $h(\infty) = 0$.

This is a boundary-value problem with two boundary-value conditions corresponding to the two unknown initial conditions: $f''(0) = v_1$ and $h'(0) = v_2$. For practical computational purposes, f' may be taken to be nearly zero at a large finite horizontal distance from the plate; we shall use the finite boundary conditions $f'(10) = 0$ and $h(10) = 0$ in place of the infinite boundary conditions given above.

The MLAB mathematical modeling system [2] may be employed to solve this double-shooting problem. The required input, and the corresponding results are shown below. We have constructed an MLAB script-file of MLAB commands entitled *hotair.do*, and executed that script to obtain the results below. (The commands in the script file are echoed in the log-file listing displayed below.)

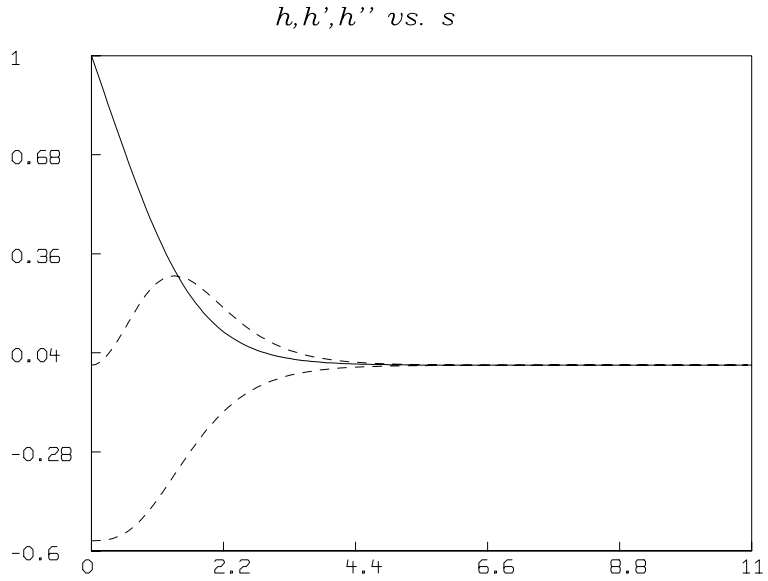
```
    MLAB: Mathematical Modeling System, Revision: January 25, 1996
    Copyright: Civilized Software, Inc. (301)652-4714

Fri Mar 29 14:03:42 1996
'* ' is the command prompt

* do "hotair.do"
*
* fct f''’s(s)=2*(f’s)^2-h-3*f*f''s
* fct h''s(s)=-3*pr*f*h’s
*
* init f(0)=0
* init f’s(0)=0
* init f''s(0)=vf
*
* init h(0)=1
* init h’s(0)=vh
*
* pr=1; vf=.5; vh=-1
*
* d=10&’0
*
* constraints q={0<vf,vf<1,vh>-1,vh<0}
```

```
* fit(vf,vh), h to d, f's to d, constraints q
final parameter values
      value              error                dependency    parameter
   0.6421470108      3.210436076e-14          0.8268185934   VF
  -0.5671057549      1.688437213e-14          0.8268185934   VH
12 iterations
CONVERGED
best weighted sum of squares = 4.232665e-27
weighted root mean square error = 6.505893e-14
weighted deviation fraction = 1.797693e+308
R squared = 1.000000e+00
no active constraints
*
* m=integrate(f'''s,h''s,0:11!160)
* odestr
     ODESTR = S  F'S'S  F'S'S'S  F'S  F'S'S  F  F'S  H'S  H'S'S  H  H'S
*
* draw m col (1,6)
* draw m col (1,4) color red lt dashed
* draw m col (1,2)  lt (.01,004,.01,0,0,0,-1) color green
* draw m col (1,3) color blue lt alternate
* top title "f,f'',f'''',f'''''' vs. s" font 17
* view
```



Note that f is drawn as a solid line, while the derivative functions f', f'', and f''' are drawn with dashed lines. It is easy to see which is f' by looking at the shape of f in the graph.

```
* delete w
* draw m col (1,10)
* draw m col (1,8) color red lt dashed
* draw m col (1,9) lt (.01,004,.01,0,0,0,-1) color green
* top title "h,h'',h'''' vs. s" font 17
* view
```




```
* exit
```

1. S. Ostrach, An analysis of laminar free convection flow and heat transfer about a flat parallel to the direction of the generating body force, NACA, Report 1111, 1953.

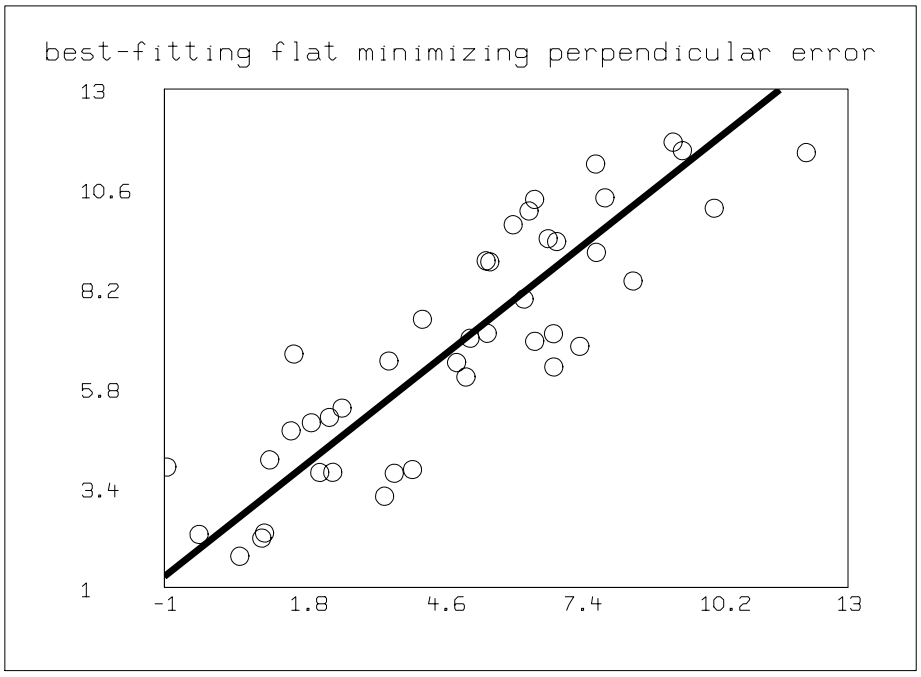
2. Civilized Software home-page: http://www.civilized.com

26 Computation of the Best-fitting Flat for a Set of Points

In order to compute the best-fitting $k-1$ dimensional flat for a set of n points in k-space, we need only compute the eigenvector corresponding to the largest eigenvalue of the "covariance" matrix of the data points. This eigenvector is the unique normal direction which defines the desired flat. This flat is the hyperplane for which the sum of the squared perpendicular distances from the data points to the flat is minimal. The process of computing this flat is also known as principal components analysis, where we seek the subspace spanned by the last $k-1$ "principal component" eigenvectors. The eigenvector corresponding to the least eigenvalue defines the best-fitting line for the n data points; this line is known as the *principal axis* of the set of data points. Note that linear regression does not define the principal axis line of a set of data points; the linear regression line is that line that minimizes the sum of the squares of the vertical (y) distances from the data points to the line.

Below we show an example using MLAB to compute the best-fitting flat for the case $k = 2$. In this case there are only two orthogonal eigenvectors.

```
* q=read(data,50,2)
* n=nrows(q)
* qb=mean(q)
*
* q= q-(qb')^^n
*
* m=q'*q
* z=eigen(m) row (1,3,5)
* type z

   Z: a  3 by 2 matrix

  1: 681.733746    45.5137215
  2: .732144702    .681149128
  3: -.681149128   .732144702

*
* v=(z row 2)'
*
* fct g(x)=qb[2]+v[2]*(x-qb[1])/v[1]
* draw points(g,-1:11.6!2) color green lt (1,0,0,0,0,.0075,0)
* draw q lt none pt circle
* top title "best-fitting flat minimizing perpendicular error"
* view
* exit
```


best-fitting flat minimizing perpendicular error
13
10.6
8.2
5.8
3.4
1
-1
1.8
4.6
7.4
10.2
13

27 Free Induction Decay

Consider the following experiment: a 1 cubic centimeter sample of water at room temperature is placed between the pole faces of a 10,000 Gauss (1 Tesla) electromagnet. As the system reaches equilibrium, a small magnetic dipole of magnitude 3.4E-6 Gauss is induced in the sample parallel to the applied field. Designate the axis passing from the south pole to the north pole of the magnet as the z-axis with the origin at the center of the sample. The sample is surrounded by two thin wire coils such that the axes of the coils are perpendicular to each other and to the z-axis. The axes passing through the center of the coils are designated the x and y axes. A brief 42 megahertz alternating current pulse is applied to the x-axis coil. If the pulse is of the appropriate amplitude and duration, it causes the induced magnetic dipole to tilt away from the z-axis. The magnetic dipole then precesses around the z-axis, gradually returning to its equilibrium position parallel to the z-axis. The component of the magnetic dipole which rotates in the xy-plane induces 42 megahertz oscillating voltages—90 degrees out of phase—across the terminals of the two coils which die away in time.

These signals are termed *free induction decay signals* and they are the basic physical effect which underlies modern day magnetic resonance imaging (MRI) and nuclear magnetic resonance (NMR) experiments. Free induction decay can be described by a model developed by F. Bloch in the paper "Nuclear Induction" Phys. Rev. 70 (1946) 460. The mathematical formulation of the model is called *Bloch's equations* and consists of 3 ordinary differential equations:

$$\frac{dM_x}{dt} = \gamma(M_y H_z - M_z H_y) - \frac{M_x}{T_2} \tag{0.4}$$

$$\frac{dM_y}{dt} = \gamma(M_z H_x - M_x H_z) - \frac{M_y}{T_2} \tag{0.5}$$

$$\frac{dM_z}{dt} = \gamma(M_x H_y - M_y H_x) + \frac{(M_0 - M_z)}{T_1} \tag{0.6}$$

These equations predict the evolution of the induced magnetic dipole, $\mathbf{M}(t) = (M_x(t), M_y(t), M_z(t))$, in the presence of a uniform magnetic field $\mathbf{H} = (H_x, H_y, H_z)$, where M_0 is the equilibrium magnitude of the induced magnetic dipole, γ is the gyromagnetic ratio for protons, T_1 is the spin-lattice relaxation constant, and T_2 is the spin-spin relaxation constant. The free induction decay signals observed on the x-axis coil and y-axis coil are proportional to the components $M_x(t)$ and $M_y(t)$, respectively.

This paper will demonstrate how the MLAB mathematical modeling system can be used to solve Bloch's equations to predict free induction decay signals in single and double pulse experiments.

27.1 A Single $\frac{\pi}{4}$ Pulse Experiment

We begin by simulating a single pulse experiment which rotates the magnetic dipole of the sample by $\frac{\pi}{4}$ radians about the x-axis. First define Bloch's equations by typing:

```
* function mx't(t) = g*(my*hz(t)-mz*hy(t))-mx/t2
* function my't(t) = g*(mz*hx(t)-mx*hz(t))-my/t2
* function mz't(t) = g*(mx*hy(t)-my*hx(t))+(m0-mz)/t1
```

The `function` statement allows the user to define any algebraic or differential equation model desired. The apostrophe, `'`, denotes the differentiation operation.

Next assign values to the constants appearing in the `function` statements by typing

```
* g = 2.66E4    /* gyromagnetic ratio for protons */
* m0 = 3.4E-6   /* equilibrium magnitude of magnetic dipole in Gauss */
* t1 = 1.E-6    /* spin-lattice relaxation time in seconds */
* t2 = 1.E-6    /* spin-spin relaxation time in seconds */
* beta1 = pi/4  /* rotation angle for first pulse in radians */
* h0 = 10000    /* magnitude of external magnetic field in Gauss */
* fct hx(t) = 0 /* x-component of external magnetic field in Gauss */
* fct hy(t) = 0 /* y-component of external magnetic field in Gauss */
* fct hz(t) = h0/* z-component of external magnetic field in Gauss */
```

The magnetic field is defined as having only a z-component of 10000 Gauss. For a normal water sample, T_1 is roughly 1 second and T_2 is roughly a millisecond. Fictitious values for the spin-lattice and spin-spin relaxation constants have been selected here so that the effect of relaxation can be observed in the time scale of several precessions of the induced magnetic dipole.

The magnetic dipole vector will precess about the z-axis at the Larmor frequency, $\omega = \frac{\gamma H_0}{2\pi} = 42$ megahertz. We will solve Bloch's equations for ten precessions sampled at 300 time points. Define a vector `t` with components equal to the times of observation.

```
* nsteps = 300
* tau = 10*2*pi/(g*h0)
* t = 0:tau!nsteps
```

The `-:-!-` construct in the last statement instructs MLAB to make `t` a row vector with components 0 to `tau` in 300 steps. We can now simulate the single pulse experiment by defining a so-called *macro* which initializes the components of the magnetic dipole vector at time 0 after the pulse and then integrates the differential equation model.

```
* pulse1 = "initial mx(0) = 0;\
           initial my(0) = sin(beta1)*m0;\
           initial mz(0) = cos(beta1)*m0;\
           m = points(mx,my,mz,t);"
```

Here the `initial` statements provide the initial conditions for the differential equations—the equilibrium dipole moment is rotated by an angle `beta1` about the x-axis. The `points` operator solves the differential equations for `mx`, `my`, and `mz` at the times listed in `t`. The `points` operator

returns a 4 column array with time in column 1, and the x, y, and z components of the magnetic dipole vector in columns 2, 3, and 4, respectively. The macro is executed as follows:

```
* do pulse1
```

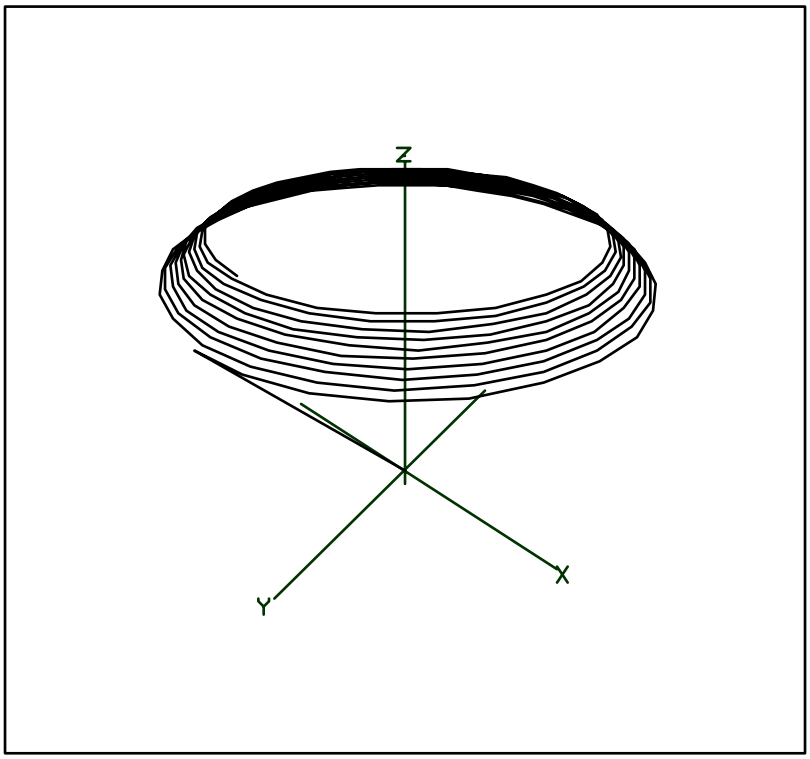
We can generate a 3 dimensional perspective figure of the time evolution of the magnetic dipole vector by defining another macro:

```
* mdip = "delete m col 1;\
         m = (0^^'3)&m;\
         draw m lt sequence;"
```

This macro instructs MLAB to throw away the time information in column 1 of the matrix `m`, add the origin $(0, 0, 0)$ to the list of magnetic dipole vector components, and draw the resulting sequence of points. We execute the macro with some 3 dimensional perspective positioning commands as follows:

```
* do mdip
* cmd3d("raise 1")      /* raises the point of view */
* cmd3d("truck 1")      /* moves the point of view to the right */
* cmd3d("track")        /* points the direction of view to the center */
* cmd3d("dolly 1")      /* moves the point of view toward the subject */
* cmd3d("twist -20")
* cmd3d("axes")         /* draws coordinate axes */
* view
```

The following picture then appears on the display:



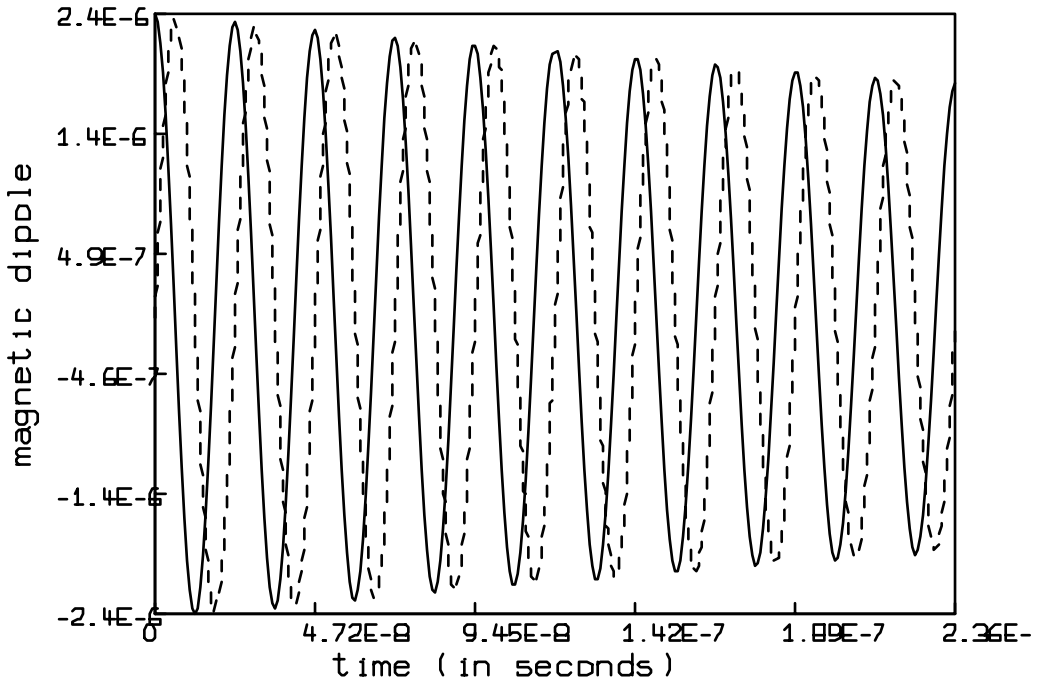
Note that the magnetic dipole at time 0 is drawn as a line segment at a 45 degree angle in the yz plane and that as time progresses, the head of the magnetic dipole vector precesses around the z-axis. As the vector precesses, the magnitude of the component in the xy plane (the transverse component) decreases in time at a rate determined by T_2, while the component along the z axis (the logitudinal component) increases in time at a rate determined by T_1. If the MLAB calculation of the magnetic dipole's time evolution were continued to longer times, the transverse component of the magnetic dipole would completely vanish and the magnetic dipole would fully recover along the z-axis.

The transverse component of the magnetic dipole can be decomposed into x and y components. It is the variations of the components of the magnetic dipole along the x and y axes which give rise to the free induction decay signals. These can be drawn by defining and executing another macro.

```
* unview                /* remove the 3d picture */
* delete w3
* btitle = "time (in seconds)"
* ltitle = "magnetic dipole"
* emfs = "delete m row 1;\
        draw t &' m col 1 lt dashed;\
```

```
        draw t &' m col 2;\
        bottom title btitle;\
        left title ltitle;"
* do emfs
* view
```

The macro `emfs` draws a graph of the x-component of the magnetic dipole versus time with a dashed line and the y-component of the magnetic dipole versus time with a solid line. It also places titles on the bottom and left axes. The following picture is then seen on the display:

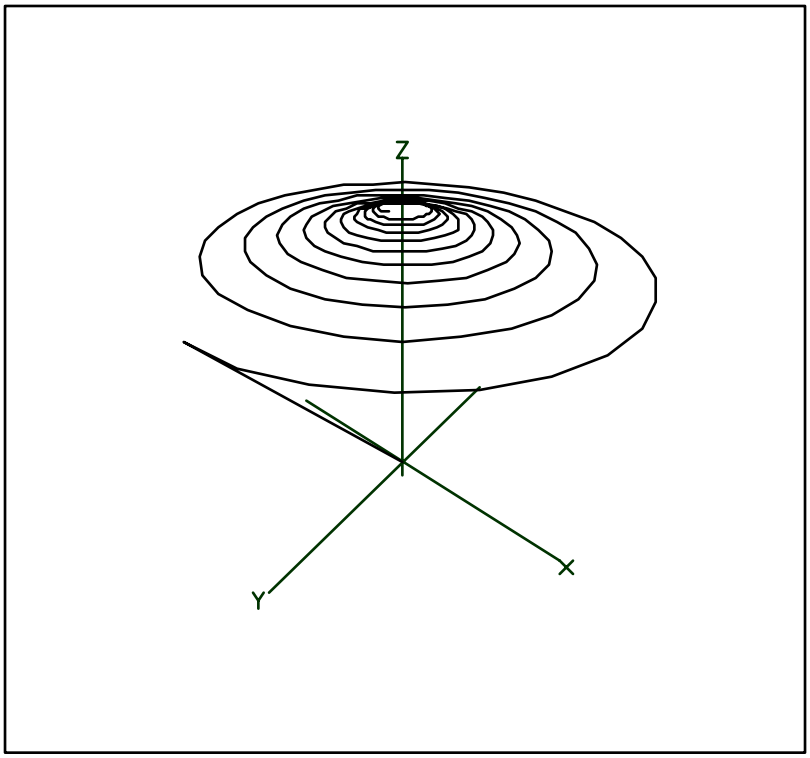


The reduction in amplitude of the oscillations is characteristic of the spin-spin relaxation time, T_2.

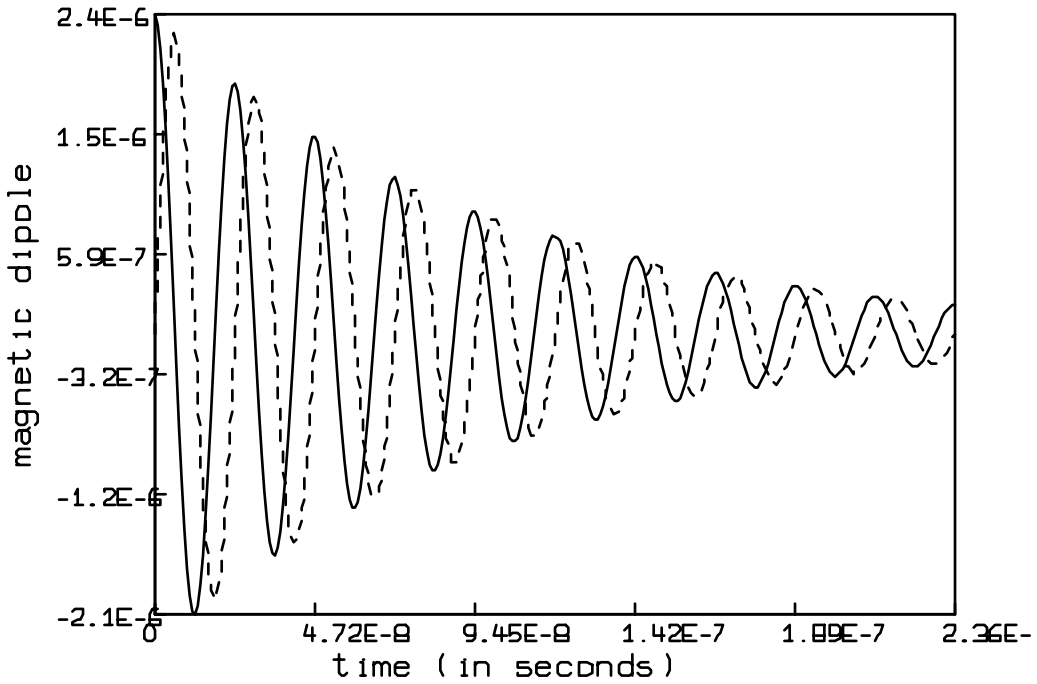
With the macros `pulse1`, `mdip`, and `emfs` defined above, it is a simple matter to change the relaxation constants, the pulse's angle of rotation, the strength and direction of the external magnetic field, or the gyromagnetic ratio and run the single pulse experiment again. For example, suppose the spin-spin relaxation constant, T_2, is smaller by a factor of 10. The following commands generate the 3 dimensional perspective picture and the 2 dimensional time plot of the components of the magnetic dipole vector:

```
* unview           /* remove the previous picture */
```

```
* delete w
* t2 = 1.E-7       /* re-define the spin-spin relaxation constant */
* do pulse1        /* run the experiment */
* do mdip          /* draw the 3d picture */
* cmd3d("raise 1")
* cmd3d("truck 1")
* cmd3d("track")
* cmd3d("dolly 1")
* cmd3d("twist -20")
* cmd3d("axes")
* view
```




```
* unview           /* remove the previous picture */
* delete w3
* do emfs          /* draw the time-dependent magnetic dipole components */
* view
```




```
* unview            /* remove the previous picture */
* delete w
```

As expected, the smaller value of the spin-spin relaxation time, T_2 causes the free induction decay signal to die away faster than in the previous example.

27.2 A Double Pulse Experiment

If the external magnetic field is not homogeneous, *double* pulse experiments can be performed which give rise to resurgences of amplitude in the free induction decay signals. This effect was described by E.L. Hahn in the article "Spin Echoes" Phys. Rev. 80 (1950) 580. Here we simulate a double pulse experiment consisting of a pulse sequence in which the first pulse rotates the magnetic dipole by $\frac{\pi}{2}$ radians about the x-axis and the second pulse—τ units of time later—rotates the magnetic dipole by π radians about the x-axis. Owing to the inhomogeneities in the external magnetic field, the resurgence of amplitude in the free induction decay signal is observed τ units of time after the second pulse.

The inhomogeneity of the external magnetic field is simulated by running the pulse sequence on 10 magnetic dipoles, each subject to a different constant, external magnetic field. The Larmor

frequency of precession is different for each magnetic dipole. The spin echo is then observed in the *net* magnetic dipole obtained by summing the time dependent free induction decay signals from each of the magnetic dipoles.

First we show the free induction decay resulting from the pulse sequence applied to a magnetic dipole in a *homogeneous* magnetic field. The macro `pulse1` from the previous section can be used to generate the time evolution of the magnetic dipole from the moment of the first pulse to the moment before the second pulse. We define a vector of times `tt` which holds times for the rest of the experiment.

```
* tt = tau:(2.5*tau)!(1.5*nsteps)
```

This statement assigns `tt` the values `tau` to `2.5*tau` in 450 steps. Then we define a new macro, `pulse2`, which determines the effect of the second pulse.

```
* pulse2 = "initial mx(tau) = m[nsteps,2];\
           initial my(tau) = m[nsteps,3]*cos(beta2)+m[nsteps,4]*sin(beta2);\
           initial mz(tau) = -m[nsteps,3]*sin(beta2)+m[nsteps,4]*cos(beta2);\
           m = m&points(mx,my,mz,tt);"
```

The `initial` statements in this macro find the components of the magnetic dipole at time `tau` after the second pulse has rotated the magnetic dipole about the x axis by an angle `beta2`. The `points` operator in the fourth statement solves Bloch's equations for the times in the vector `tt` and the ampersand operator concatenates the four column solution matrix from the `points` operator to the existing matrix `m` where the time evolution of the magnetic dipole from 0 to `tau` has been stored.

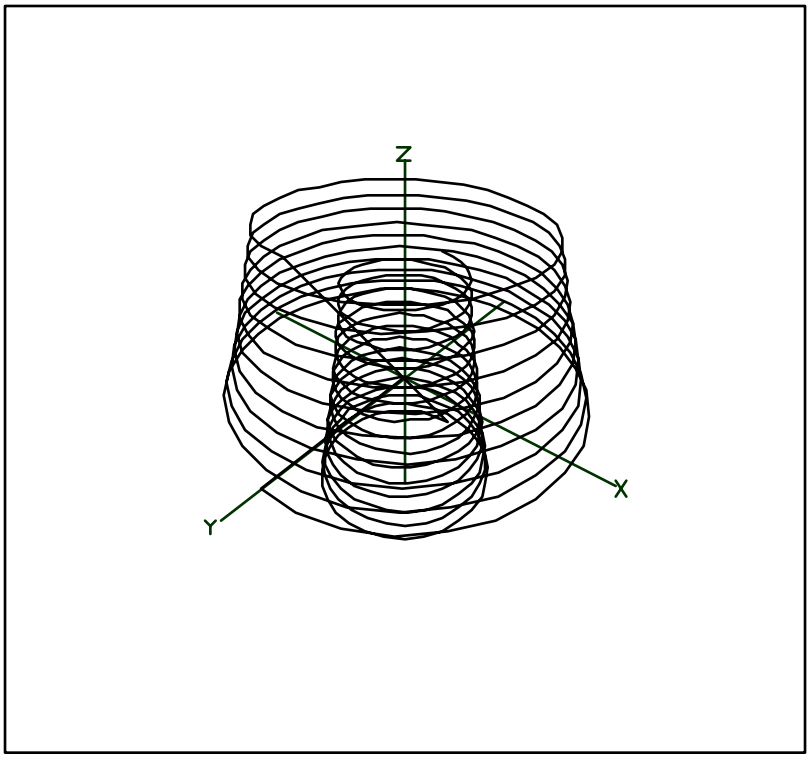
The following MLAB commands perform the complete double pulse sequence on a magnetic dipole in a 10000 Gauss homogeneous magnetic field:

```
* beta1 = pi/2
* beta2 = pi
* t2 = 5.E-7
* do pulse1
* do pulse2
```

To see the time evolution of the magnetic dipole in a 3 dimensional perspective, type:

```
* do mdip
* cmd3d("raise 1")
* cmd3d("truck 1")
* cmd3d("track")
* cmd3d("dolly 1")
* cmd3d("twist -20")
* cmd3d("axes")
```

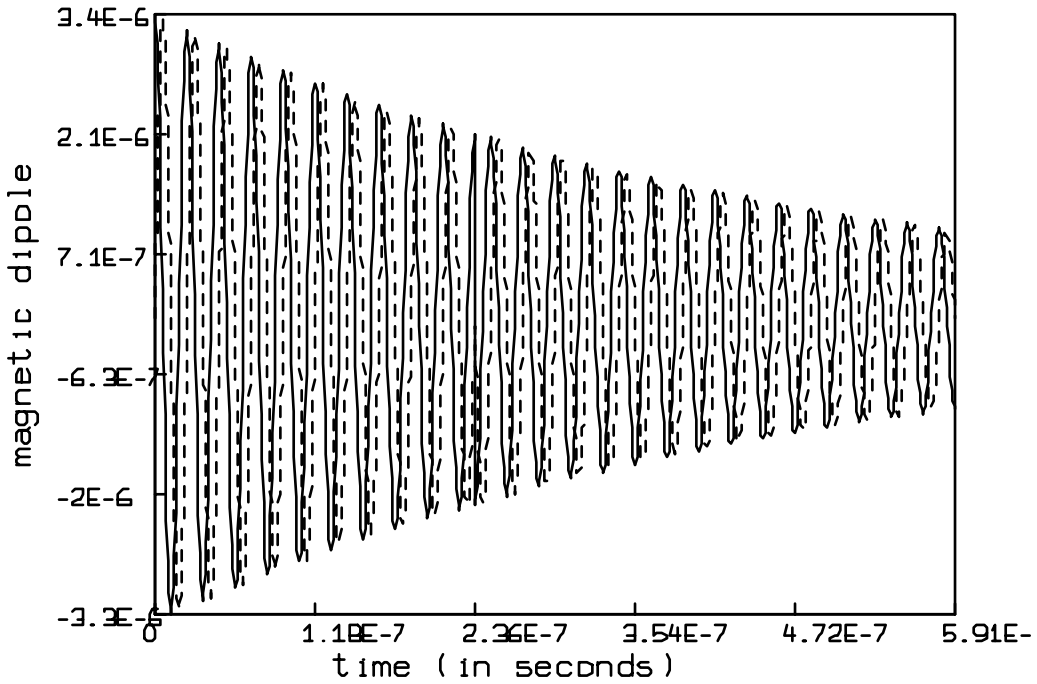
```
* view
```



This figure shows the evolution of the magnetic dipole in a homogeneous magnetic field through the double pulse sequence. The magnetic dipole after the first $\frac{\pi}{2}$ pulse is seen as a line segment along the positive y-axis. As time progresses, the magnetic dipole precesses around the z-axis until the moment of the second pulse at time τ. When the second pulse is applied at time `tau`, the y and z components of the magnetic dipole are reversed. The magnetic dipole then continues to precess about the z axis, returning to its equilibrium configuration. Throughout the entire process, the magnitude of the transverse component of the magnetic dipole is seen to decrease at a rate proportional to the spin-spin relaxation constant, T_2.

The x and y components of the magnetic dipole, which are proportional to the free induction decay signals, are shown by the commands

```
* unview
* delete w3
* ttt = t
* t = t & tt
* do emfs
* view
```



This figure shows that in the homogeneous magnetic field, the x and y components of the magnetic dipole and, therefore, the free induction decay signals, simply die off with a characteristic time constant of T_2. There is a change in phase of the signal when the second pulse occurs at t equals 2.36E-7 seconds.

```
* unview
* delete w
* t = ttt
```

To demonstrate the spin echo effect, we determine the time evolution of the average of 10 distinct magnetic dipoles—each evolving in a different magnetic field. This is accomplished with the following commands:

```
* h00 = 9600:10400!10     /* the different magnetic fields */
* netp = shape(750,3,0^^2250);
* for i = 1:10 do {
>    h0 = h00[i]
>    do pulse1
>    do pulse2
>    delete m col 1
```

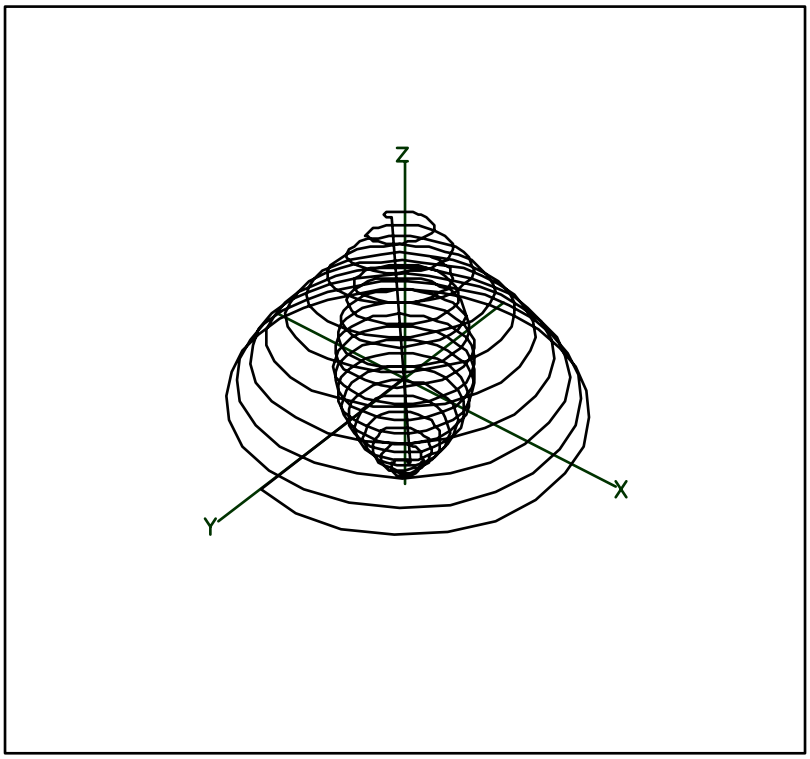
```
>    netp = netp+m
>  }
```

First we set `h00` to be a vector of magnetic field strengths ranging in value from 9600 Gauss to 10400 Gauss in 10 steps. Actual inhomogeneities in the external magnetic field are on the order of .01 Gauss. Here, the inhomogeneity in the 10000 Gauss external magnetic field is greatly exaggerated so that the spin echo effect can be demonstrated in a short time interval.

The `shape` operator is then used to initialize the 750 row by 3 column matrix `netp` to zero. The `for...do` loop runs the double pulse experiment ten times. Each time, the magnetic field is different and the time evolution of the magnetic dipole computed in `m` is accumulated in the array `netp`.

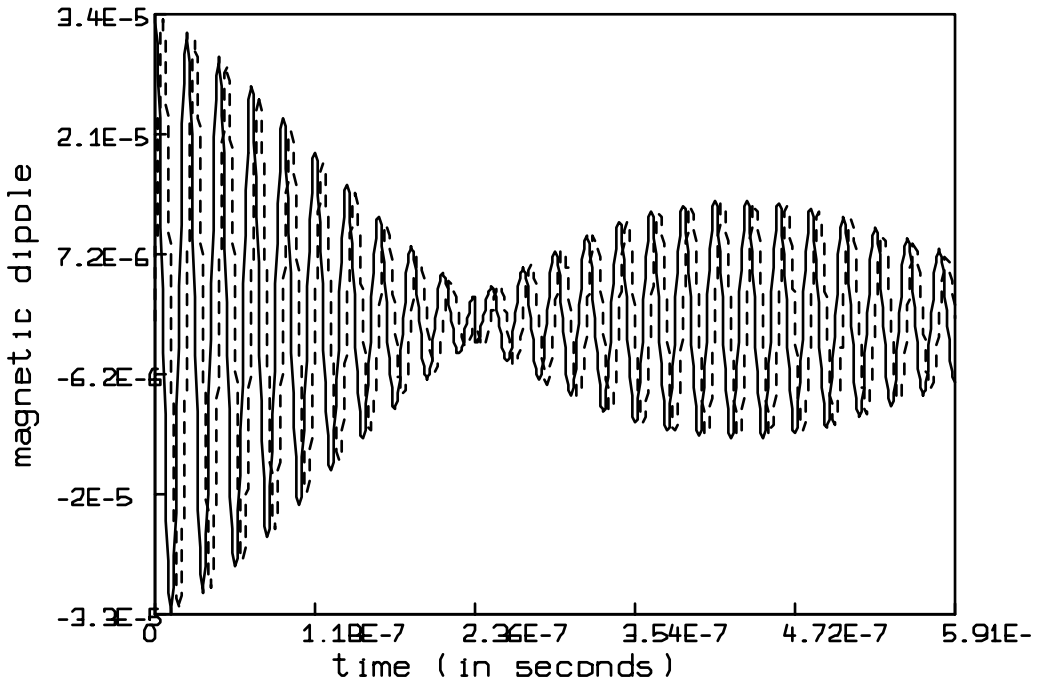
The 3 dimensional perspective view of the evolution of the net magnetic dipole is shown by the following commands:

```
* netp = (0^^'3)&netp
* draw netp lt sequence
* cmd3d("raise 1")
* cmd3d("truck 1")
* cmd3d("track")
* cmd3d("dolly 1")
* cmd3d("twist -20")
* cmd3d("axes")
* view
```



This figure shows that the net magnetic dipole lies along the y-axis at time 0 after the first $\frac{\pi}{2}$ pulse. The net magnetic dipole then precesses around the z-axis. The transverse component of the net magnetic dipole is seen to decrease *faster* than the magnetic dipole in the homogeneous magnetic field experiment, owing to destructive interference between the constituent magnetic dipoles precessing at different Larmor frequencies. Immediately after the second pulse, the y and z components of the net magnetic dipole are reversed. As the net magnetic dipole precesses about the z axis during subsequent evolution, there is a temporary increase in the amplitude of the transverse component of the net magnetic dipole. This is the spin echo. It is more readily seen in graphs of the x and y components of the net magnetic dipole.

```
* unview
* delete w3
* delete netp row 1
* draw (t & tt) &' netp col 1 lt dashed
* draw (t & tt) &' netp col 2
* bottom title btitle
* left title ltitle
* view
```




```
* unview
* delete w
```

This paper has demonstrated how MLAB can be used to explore a specific differential equation model: Bloch's equations for a magnetic dipole in an external magnetic field.

28 Solving Equations Involving the Catenary with MLAB

A 10 foot rope hangs in the form of a catenary with one end attached to a pier at a point 3 feet above the water's surface, and the other end attached at the same height to a boat. If the lowest point on the catenary just touches the water, how far is the boat from the pier? If that distance is fixed, does the minimum of the catenary submerge or clear the surface of the water as the tide ebbs? as the tide flows?

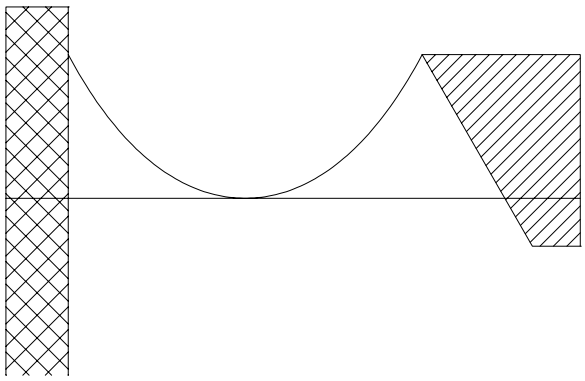


Figure 1: A rope fixed to a pier and a boat. The lowest point touches the water's surface.

It has been known since Leibnitz and Huygens' time, circa 1691, that a rope or chain tethered at each end hangs in the shape of a catenary curve. Reference 1 gives the mathematical form of a catenary as

$$y(x) = \beta + \gamma \cosh(\frac{x}{\gamma} + \alpha).$$

α, β, and γ must be chosen so that the catenary passes through two designated endpoints and the total length of the catenary has the proper value. The length of the catenary with abscissae ranging from x_0 to x_1 is given by

$$L(x_0, x_1) = \gamma \ [\sinh(\frac{x_1}{\gamma} + \alpha) - \sinh(\frac{x_0}{\gamma} + \alpha)].$$

Choosing a coordinate system that has an x-axis co-linear with the line segment that is parallel to the water surface and connects the endpoints of the rope, and a y-axis that is perpendicular to the x-axis and passes through the minimum of the catenary gives rise to 4 equations:

$$\begin{aligned} 0 &= \beta + \gamma \cosh(\alpha) \\ 3 &= \beta + \gamma \cosh(\frac{\bar{x}}{2\gamma} + \alpha) \\ 3 &= \beta + \gamma \cosh(-\frac{\bar{x}}{2\gamma} + \alpha) \\ 10 &= \gamma \left[\sinh(\frac{\bar{x}}{2\gamma} + \alpha) - \sinh(-\frac{\bar{x}}{2\gamma} + \alpha)\right] \end{aligned}$$

where $\bar{x}$ is the distance separating the pier and the boat.

The MLAB mathematical modeling program provides several ways of solving sets of equations such as this. Here we describe 2 methods: one using the `ROOT` operator and the other using the `MINIMIZE` operator.

To use the MLAB `ROOT` operator effectively, a little more human thinking is required to reduce the 4 equations in 4 unknowns to 1 equation in 1 unknown. With some algebraic manipulation of the four equations, one finds $\alpha = 0$, $\beta = -(\gamma + 3)$, $\bar{x} = 2\gamma \sinh^{-1}(\frac{5}{\gamma})$, and that γ must satisfy the following transcendental equation:

$$0 = -3 + \gamma \left[\cosh(\sinh^{-1}(\frac{5}{\gamma}) - 1)\right].$$

The `ROOT` operator finds that value of a variable within a specified interval of values, where a given function of that variable equals zero. The form of the `ROOT` operator is `ROOT(X,A,B,E)` where `X` is the name of the unknown variable, `A` and `B` are numbers or expressions that specify the range of values within which the root-value lies, and `E` is a mathematical expression involving the unknown variable. The equation given above involving γ can be solved and the other unknowns evaluated, with the following MLAB commands:

```
FCT F() = ROOT(G,.001,100,-(3+G)+G*COSH(ASINH(5/G)))
A = 0; C = F(); XB = 2*C*ASINH(5/C); B = -(C+3);
TYPE A,B,C,XB
```

The `ROOT` operator finds a zero of the given mathematical expression by using a hybrid algorithm that mixes bracketing, secant, Newton, and bisection methods. These methods are described separately in Reference 2. `COSH` and `ASINH` evaluate the hyperbolic cosine and inverse hyperbolic sine, respectively. The `TYPE` command prints the following results to the screen:

```
A = 0
B = -5.66666667
```

```
C = 2.66666667
XB = 7.39356993
```

Another method of solving the four equations consists of using the MLAB `MINIMIZE` operator to find the values of α, β, γ, and $\bar{x}$ that make a least-squares objective function evaluate to the minimum value of 0. The least-squares objective function in this problem is given by

$$\text{obj}() = [10 - L(-\frac{\bar{x}}{2}, \frac{\bar{x}}{2})]^2 + [y(\frac{\bar{x}}{2})]^2 + [y(-\frac{\bar{x}}{2})]^2 + [3 + y(0)]^2.$$

Here are the MLAB commands that use the `MINIMIZE` operator to solve the problem:

```
FCT Y(X) = B+C*COSH((X/C)+A)
FCT L(X0,X1) = C*(SINH((X1/C)+A)-SINH((X0/C)+A))
FCT OBJ() = (10-L(-XB/2,XB/2))^2+(Y(XB/2))^2+(Y(-XB/2))^2+(3+Y(0))^2
A = 1; C = 1; XB = 1; B = 1
MINIMIZE(OBJ,XB,A,B,C)
TYPE A,B,C,XB
```

With the first 3 commands, we define `Y(X)` to be the general form of the catenary function, `L(X0,X1)` to be the distance between the endpoints along the catenary, and `OBJ()` to be the least-squares objective function for the system of equations. The assignment statements provide initial guesses for the parameters `A`, `B`, `C`, and `XB` that appear in the definitions of the the functions. Then the `MINIMIZE` operation does a variable metric method search (as described in Reference 2) of the (α,β,γ,$\bar{x}$)-parameter space from the point (1,1,1,1). The first argument to `MINIMIZE` is the name of the function for which a minimum is to be found; the remaining arguments are the names of the parameters to be varied. The `MINIMIZE` operator changes the values of the parameters listed in the argument list. The results printed to the screen by the last `TYPE` command are

```
A = 8.86949601E-10
B = -5.66666669
C = 2.66666669
XB = 7.39356995
```

Both the `ROOT` and `MINIMIZE` methods determined the separation of the pier and the boat to be approximately 7.39 feet.

The MLAB `FIT` and `MINIMIZE` operators can now be used to investigate how the minimum of the catenary changes as the right endpoint moves down or up vertically, i.e. as the tide ebbs or flows. Although the minimum of the catenary formed when the boat is above or below the pier is not the same as the minimum when the boat and the pier are at the same level, we continue to use the coordinate system described above in which the y-axis passes through the minimum of the catenary formed when the boat and the pier *are* at the same level. Note that the boat is always three feet above the water level.

```
/* compute the maximum height that the boat can attain if the rope
   is 10 feet long and the pier and boat are separated by xb feet */
MH = SQRT(100-XB^2);

/* compute a vector of 21 equally-spaced height values ranging from
   -MH to MH, excluding mh and -mh */
D = -MH:MH!23
DELETE D ROW (1,23)

/* L(-XB/2,XB/2) = 10, and ML = the corresponding data point */
ML = LIST(-XB/2,XB/2,10)'

/* define linear constraints for XMIN, B, C */
CONSTRAINTS Q1 = {B<0,C>0}
CONSTRAINTS Q2 = {XMIN >= -XB/2, XMIN <= XB/2}

XMIN = 0; /* initial guess at minimum's abscissa */

/* loop over 21 different heights, D[J], J = 1,2,...,21 */
FOR J = 1:21 DO {
    /* MC = 2X2 matrix containing endpoints of the catenary
       when the boat is at (XB/2,D[J]) */
    MC = SHAPE(2,2,LIST(-XB/2,0,XB/2,D[J]));

    /* adjust (A,B,C) to fit the catenary to the endpoints and length */
    FIT (A,B,C), Y TO MC, L TO ML, CONSTRAINTS Q1;

    /* find the minimum point (XMIN,YMIN) on the catenary, leaving
       A,B, and C fixed, constraining (-XB/2)<=XMIN<=(XB/2) */
    YMIN = MINIMIZE(Y,XMIN,Q2);

    /* store the results in the matrix MRES */
    MRES ROW J = LIST(D[J],XMIN,YMIN,A,B,C)';
   }
```

The MLAB `FIT` operator uses the Marquardt-Levenberg curve-fitting method described in References 3 and 4. The quadratic programming algorithm, which the curve-fitting algorithm needs to converge within linear constraints is described in Reference 5.

Five of the resulting catenaries are graphed via MLAB in Figure 2.

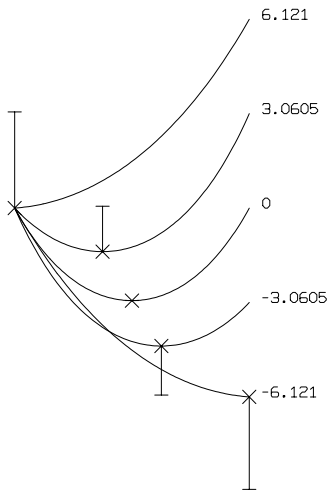


Figure 2: Catenaries for 5 different tide levels. The minimum of each catenary is marked with × and connected to a horizontal dash mark which represents the water level.

The computed minimum of each of the 21 catenaries and the corresponding water levels are shown in Figure 3.

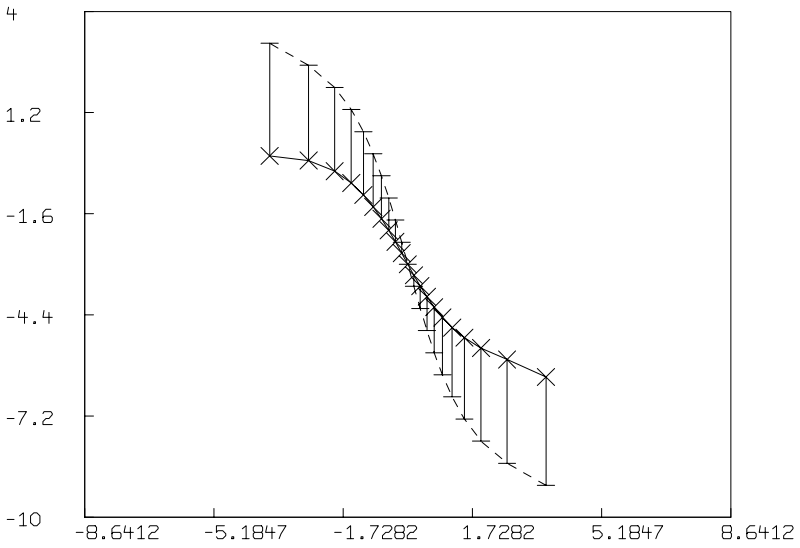


Figure 3: The minima of 21 catenaries marked with × connected to horizontal dash marks which represent water level.

If the tide ebbs, thereby lowering the boat with respect to the pier, the rope clears the water. If the tide rises, thereby raising the boat with respect to the pier, the rope is partially submerged.

29 Analysis of Sunspot Data with MLAB

This paper will demonstrate some of the data visualization and signal processing capabilities available with the MLAB mathematical modeling computer program . These capabilities will be applied in consideration of solar sunspot data. As periods of intense sunspot activity may affect many terrestial phenomena including climate, quality of radio communications, occurrence of auroras, and overloads in public utility electrical power grids, understanding and predicting sunspot activity is of interest.

Daily sunspot numbers compiled by the American Association of Variable Star Observers (AAVSO) can be found each month in *Sky and Telescope* magazine, or requested from AAVSO Solar Division chairman, Peter O. Taylor (e-mail: ptaylor@ngdc.noaa.gov).

The MLAB operator `MMEAN` can be used to calculate moving averages of the sunspot data over periods greater than a day. For example, if the daily sunspot numbers from 1986 to 1995 are stored in the ASCII text file *recssd.dat*, the following MLAB commands will generate Figure 1 which shows the daily data along with month averages and year averages.

```
* "store AAVSO's daily data between 1986 and 1995 in a 2 column matrix"
* m4 = read(recssd,4000,2)
*
* "draw the daily data"
* draw m4
* title "daily data" at (1994,208) world size .01
* window 1986 to 1996, 0 to 260
* frame 0 to 1, .67 to 1
* w1 = w
*
* "smooth the data using moving mean operator with 30 day window"
* m2 = m4 col 1 &' mmean(m4 col 2,30)
* draw m2
* title "30-day average" at (1994,208) world size .01
* window 1986 to 1996, 0 to 260
* frame 0 to 1, .33 to .67
* w2 = w
*
* "smooth the data using moving mean operator with 365 day window"
* m3 = m4 col 1 &' mmean(m4 col 2,365)
* draw m3
* title "365-day average" at (1994,208) world size .01
* window 1986 to 1996, 0 to 260
* frame 0 to 1, 0 to .33
*
```

```
* view
```

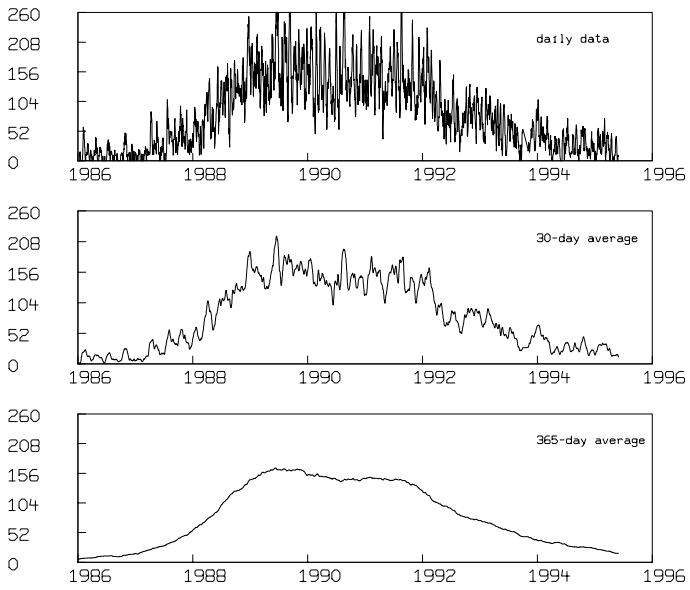


Figure 1: Averaging data with the MLAB moving mean operator.

The `MMEAN` operator in MLAB takes a sequence of numbers and a window width and computes the average of the numbers in the window as the center of the window proceeds from the first number in the sequence to the last number in the sequence. For windows extending beyond the first or last number in the sequence, a variety of extension options are provided; the default extension method—which prepends copies of the first number to the beginning of the sequence and appends copies of the last number to the end of the sequence, as needed—was used in this example. The general effect of the moving average is to smooth noisy data. Larger windows are seen to yield smoother results.

Annual mean sunspot numbers for the years 1750 through 1994 are composed of Zurich/International and American Relative Sunspot Numbers. They are available through the CompuServe Information Service (GO SUNSPOT) and other sources. Reference 1 provides a series expansion for computing the Sun-Jupiter distance in astronomical units (1 astronomical unit = 93 million miles = 150 million kilometers) over the same period. If the Zurich data is stored in an ASCII text file called *sunspots.dat* and the necessary expansion coefficients from Reference 1 are stored in an ASCII text file called *juprad.dat*, the following MLAB commands generate Figure 2:

```
* "enter the AAVSO's annual averages from 1750 to 1994, make a 2"
* "column matrix with time increasing in the first column"
* m1 = read(sunspots,1000,1)
* m1 = shape(246,2,m1)
* m1 = sort(m1,1)
*
* "draw the AAVSO data"
* top title "Annual Average Sunspot Number"
* draw m1
* frame 0 to 1,.5 to 1
* w1 = w
*
* "compute Sun-Jupiter distance at Jan 1, 0:00, for each year from"
* "1750 to 1994"
* juprad = shape(244,2,0^^490)
* jrcoeff = read(juprad,53,8);
* ad1800 = jdate(1,1,1800)-.5
* fct dt(u) = (-15+32.5*(((u-ad1800)/36525)-.1)^2)/(60*60*24)
* fct vc(et) = (et-ti0)/2000
* fct f(r,t) = sum(j,0,6,jrcoeff[r,j+1]*t**j)
* k = 1;
* for i = 1750:1990:5 do {
> rw = 1+(i-1750)/5
> ti0 = jdate(1,1,jrcoeff(rw,8))-.5;
> for j = 0:4 do {
> t = jdate(1,1,i+j); "determine the Julian date of Jan. 1, year = i+j"
> v = vc(t+dt(t)); "determine the expansion's independent variable value"
> juprad[k,1] = i+j; "store the year in column 1"
> juprad[k,2] = f(rw,v); "store the Sun-Jupiter distance in column 2"
> k = k+1;
> }
> }
*
* "draw the Sun-Jupiter distance versus time"
* top title "Sun-Jupiter distance"
* draw juprad
* frame 0 to 1, 0 to .5
* w2 = w
* view
```

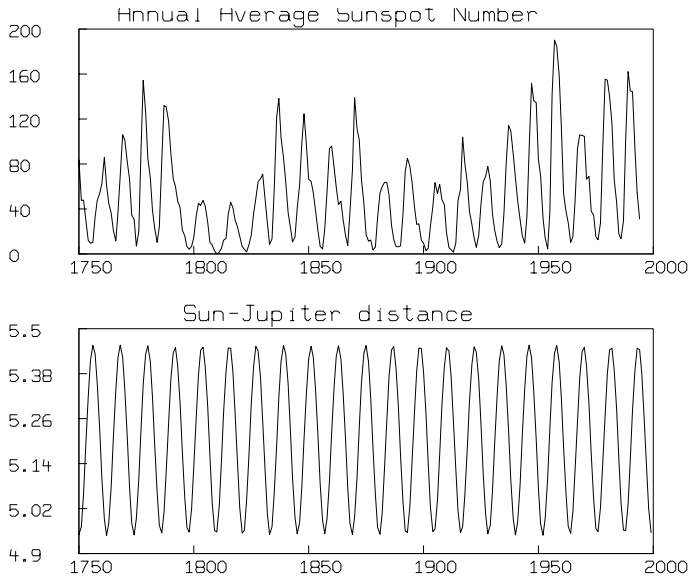


Figure 2: Annual average sunspot numbers (top) and Sun-Jupiter distance (bottom) from 1750 to 1995.

The two curves are similar in that both show about twenty three regularly spaced maxima and minima over the roughly 250 year interval.

Using the `REALDFT` function in MLAB, we can compute and graph the amplitude spectrum of each of the curves in Figure 2. Assuming a signal, $S(t)$, is sampled n times and repeats with period p, it can be expanded in a Fourier series as

$$S(t) = \sum_{j=0}^{n-1} a_j \cos(2\pi \frac{j}{p} t + \phi_j)$$

.

The amplitude spectrum is then the set of points $(\frac{p}{j}, a_j)$ for $j = 0, 1, 2, \ldots, n-1$. Figure 3 results from the following MLAB commands:

```
* "compute the amplitude spectrum of the Sun-Jupiter distance signal and"
* "annual average sunspot signal"
* p1 = realdft(m1)
```

```
* p2 = realdft(juprad)
*
* "convert frequencies to periods by taking reciprocal of FT's first column"
* "delete first rows to eliminate divide-by-zeros"
* delete p1 row 1, p2 row 1
* p1 col 1 = 1/(p1 col 1)
* p2 col 1 = 1/(p2 col 1)
*
* "draw the amplitude spectrum of the Sunspot data"
* top title "Amplitude Spectrum of Sunspot Numbers"
* draw p1 col 1:2
* frame 0 to 1, .5 to 1
* w3 = w
*
* "draw the amplitude spectrum of the Sun-Jupiter distance"
* top title "Amplitude Spectrum of Sun-Jupiter distance"
* draw p2 col 1:2
* frame 0 to 1, 0 to .5
* w4 = w
*
* blank w1,w2
* view
```

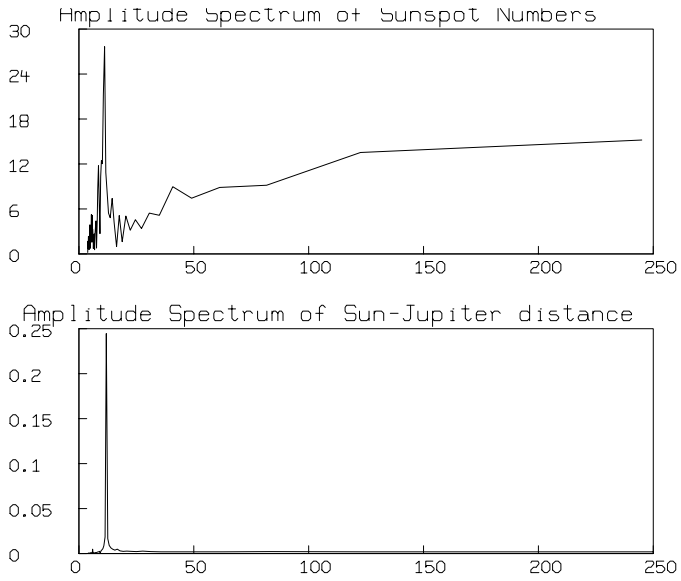


Figure 3: The amplitude spectrum of annual average sunspot data (top) and Sun-Jupiter distance (bottom).

The dominant frequency component in both the annual average sunspot number data and the Sun-Jupiter distance data is seen to have a period of roughly 11 years.

If the Sun-Jupiter distance data is considered to be a periodic *input* function, $I(t)$, which is transformed by some unknown periodic *transfer* function, $T(t)$, to yield the annual average sunspot data as a periodic *output* function, $S(t)$, by the following relation,

$$S(t) = \int I(\tau) \cdot T(t - \tau) d\tau$$

then the MLAB function `DECONV` can be used to estimate the unknown transfer function as follows:

```
* "Sun-Jupiter distance signal is the input signal and the"
* "annual average sunspot data is the output signal; compute the"
* "transfer function"
* z = deconv(juprad col 2,m1 col 2,2)
*
```

```
* "draw the transfer function with the input and output signals"
* draw (m1 col 1) &' z
* frame 0 to 1, 0 to .33
* top title "Transfer function"
* w5 = w
* blank w3,w4
* unblank w1,w2
* frame 0 to 1,.67 to 1 in w1
* frame 0 to 1,.33 to .67 in w2
* view
```

The resulting transfer function is shown with the input and output functions in Figure 4.

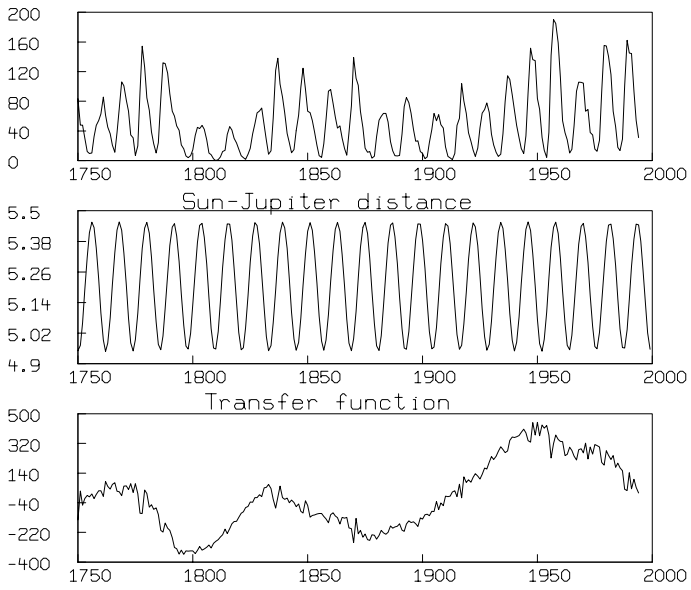


Figure 4: Annual sunspot average as output signal (top), Sun-Jupiter distance as input signal (middle), and transfer function as deconvolution of input and output (bottom).

Note that the computed transfer function corresponds roughly to the modulation in amplitude of the input signal observed in the output signal. It would be interesting to research if the transfer function corresponds to some physical process occurring in the sun.

MLAB has a non-linear least squares curve fitting operation that can be used to extrapolate the

sunspot data to the future. We can find values for the parameters a,b,c, and d in the function

$$f(t) = a\cos(b \cdot (t - c)) + d$$

which minimize

$$g(t) = \sum_{i=1}^{n} |f(t_i) - y(t_i)|^2$$

where $(t_i, y(t_i)), i = 1, 2, \ldots, n$ are the sunspot data of interest; here we consider monthly average sunspot data from 1900 to 1985 and the yearly average sunspot data from 1750 to 1995. If an ASCII file called *slm.dat* contains the monthly average sunspot data from 1900 to 1985 given in the appendix of Reference 2, and the ASCII file called *recssm.dat* contains the monthly average sunspot data from 1985 to 1995 from AAVSO, the following MLAB commands will result in Figure 5 which shows the daily sunspot data, month average sunspot data, year average sunspot data, and extrapolated fitted curves.

```
* "enter Marple's monthly averages from 1900 to 1985"
* m2 = read(slm,1020,2)
* m2 = sort(m2,1)
*
* "enter AAVSO's monthly averages from 1985 to 1995 and concatenate with"
* "Marple's data"
* m3 = read(recssm,1000,2)
* m2 = m2&m3
*
* "fit to find the offset in each data set"
* fct f1(t) = d1
* d1 = 40
* fit d1, f1 to m1
final parameter values
      value               error                dependency    parameter
     52.4004065         2.654515798                     0   D1
2 iterations
CONVERGED
best weighted sum of squares = 4.246898e+005
weighted root mean square error = 4.163445e+001
weighted deviation fraction = 5.040686e-001
* fct f2(t) = d2
* d2 = 40
* fit d2, f2 to m2
```

```
final parameter values
      value               error                dependency    parameter
    61.90062665         1.524683901                     0   D2
2 iterations
CONVERGED
best weighted sum of squares = 2.981508e+006
weighted root mean square error = 5.132096e+001
weighted deviation fraction = 5.160940e-001
R squared = 2.342744e-015
*
* "using the offset, fit a cosine curve to find the period in each data set"
* fct f1(t) = a1*cos(b1*(t-c1))+d1
Redefining F1
* a1 = 40; b1 = 2*pi/11; c1 = -1749;
* fit (a1,b1,c1), f1 to m1
final parameter values
      value               error                dependency    parameter
    29.02077265         3.269184404        4.382000966e-005   A1
   0.5695259285      0.001604153238          0.9996200793   B1
   -1759.006943          10.2264283          0.9996200791   C1
3 iterations
CONVERGED
best weighted sum of squares = 3.204926e+005
weighted root mean square error = 3.631666e+001
weighted deviation fraction = 4.226415e-001
R squared = 2.453489e-001
* fct f2(t) = a2*cos(b2*(t-c2))+d2
Redefining F2
* a2 = 40; b2 = 2*pi/11; c2 = -1900;
* fit (a2,b2,c2), f2 to m2
final parameter values
      value               error                dependency    parameter
    56.46748013          1.36633036        0.000234975336   A2
   0.6012346023      0.0008759293804          0.9999436943   B2
   -1708.911894         5.327618133          0.9999436942   C2
18 iterations
CONVERGED
best weighted sum of squares = 1.186679e+006
weighted root mean square error = 3.240614e+001
weighted deviation fraction = 3.232770e-001
R squared = 6.019871e-001
*
```

```
* "draw the annual averaged data with circles, the monthly averages with"
* "crosses, the fit of the annual averaged data with a solid line,"
* "and the fit of the monthly averaged data with a dashed line"
* draw m1 lt none pt circle ptsize .01 color red
* draw m2 lt none pt crosspt ptsize .01 color green
* draw m4 lt none pt dotpt color orange
* draw points(f1,1900:2010!500) lt dashed color yellow
* draw points(f2,1900:2010!500)
* window 1986 to 2006, 0 to 320
* draw 1993 &' 300 pt dotpt color orange
* draw 1993 &' 280 pt crosspt ptsize .01 color green
* draw 1993 &' 260 pt circle ptsize .01 color red
* draw (1992.5 & 1993.5) &' (240 & 240)
* draw (1992.5 & 1993.5) &' (220 & 220) lt dashed color yellow
* title "day average" at (1995,300) world size .01 color orange
* title "month average" at (1995,280) world size .01 color orange
* title "year average" at (1995,260) world size .01 color red
* title "cosine fit of month averages since 1900" at (1995,240) world size .01
* title "cosine fit of year averages since 1750" at (1995,220) world \
: size .01 color yellow
* top title "Time Variation of Sunspots"
* left title "Relative Sunspot No."
* bottom title "Year"
* view
```

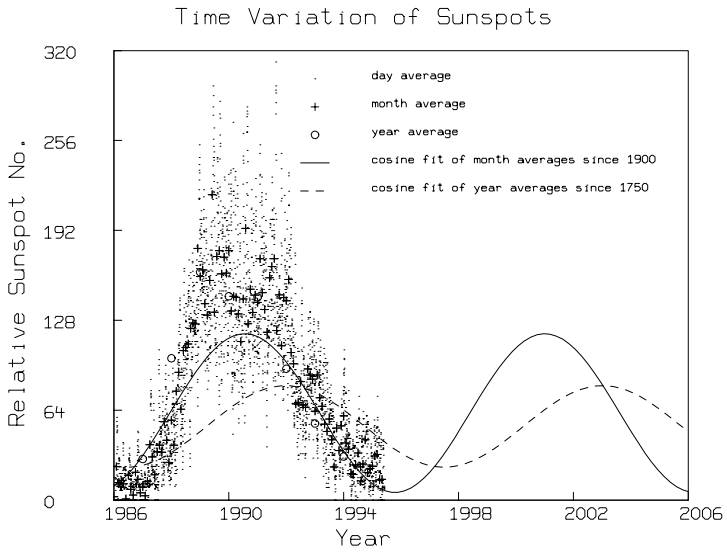


Figure 5: Daily, monthly, and yearly average sunspot numbers and fitted cosine models with projection to 2006.

According to these simple cosine models, the next maximum in sunspot activity is to be expected between 2001 and 2003.

A visual method for finding periodic trends in annual average sunspot number data is to create a surface by stacking sunspot data between successive maxima and examining contours of constant value on the resulting surface. The following MLAB commands use the `CONTOUR` operator to find trends in 11 year periods of both the annual average sunspot data and the best fitting cosine model.

```
* "draw contour diagram of folded time series"
* cnst = 2*pi/b1; nx = 22; ny = 11;
* fct f(x,y) = lookup(m1,1750+y*cnst/ny+x*cnst)
* z1 = cross(0:21,0:10)
* z1 col 3 = f on z1
* m1 = points(f1,m1 col 1)
* z2 = z1 col 1:2
* z2 col 3 = f on z2
* mz = maxv(z2 col 3)
```

```
*
* "find contour levels for the annual averaged sunspot data and cosine model"
* cc8 = contour(z2,10:(mz-10)!4)
* cc7 = contour(z1,10:(mz-10)!4)
* draw cc7 lt 7 in w
* top title "Sunspot data"
* window 0 to 21,0 to 10*cnst/11
* xaxis 0:21:3&'0 label 1750:1981:33 labelsize .015 ffract offset(-.01,-.025)\
:  ffract pt utick format(-3,5,0,0,2,0)
* yaxis 0&'(0:(cnst*10/11)!6) label 0:(cnst*10/11)!6 labelsize .015 ffract \
: offset(-.09,-.01) ffract pt rtick format(-3,5,0,0,2,0)
* frame 0 to .5, 0 to 1
* w1 = w
*
* "draw contour diagram of cosine model"
* draw cc8 lt 7 in w
* top title "Cosine model"
* window 0 to 21,0 to 10*cnst/11
* xaxis 0:21:3&'0 label 1750:1992:33 labelsize .015 ffract offset(-.01,-.025)\
:  ffract pt utick format(-3,5,0,0,2,0)
* yaxis 0&'(0:(cnst*10/11)!6) label 0:(cnst*10/11)!6 labelsize .015 ffract \
: offset(-.09,-.01) ffract pt rtick format(-3,5,0,0,2,0)
* frame .5 to 1, 0 to 1
* view
```

The resulting graphs are shown in Figure 6.

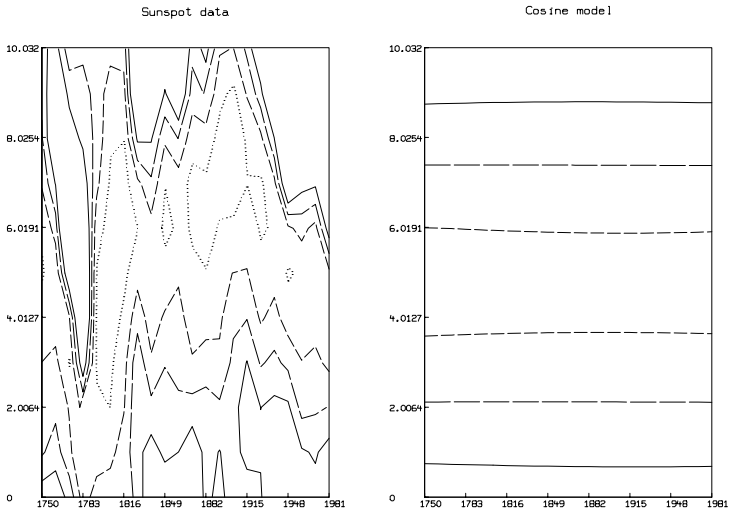


Figure 6: Contours of folded annual average sunspot data and cosine model.

The contours in these graphs that correspond to the highest level are drawn with solid lines; those contours that correspond to the lower levels are drawn with successively shorter dashed lines; the contours corresponding to the lowest level are drawn with dotted lines. Note that each period of sunspot data is treated as a vertical slice of the composed surface.

A three dimensional perspective view of these surfaces looking down the sunspot minimum valley with time increasing as one progresses up the valley toward the point of perspective can be generated in MLAB with the following commands:

```
* "draw 3d perspective views of the 2 surfaces"
* blank w1,w
* draw z1 lt hidden
* cmd3d("surface zrotate 90")
* cmd3d("raise 1");
* cmd3d("track")
* cmd3d("dolly .5");
* cmd3d("box")
* cmd3d("colorlist 0,1");
* frame 0 to .5, 0 to 1 in w3
```

```
* w3d = w3
* draw z2 lt hidden
* cmd3d("surface zrotate 90")
* cmd3d("raise 1");
* cmd3d("track")
* cmd3d("dolly .5");
* cmd3d("box")
* cmd3d("colorlist 0,1");
* frame .5 to 1, 0 to 1 in w3
* view
```

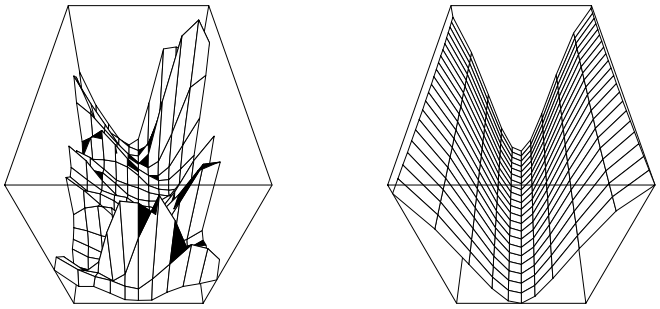


Figure 7: 3D-perpective views of folded annual average sunspot data and cosine model.

It is apparent from the contour map and 3D-perspective view that although the sunspot data follows an 11 year cycle from maximum to maximum (or minimum to minimum), there is considerable drift in the occurrence of the minimum (maximum) from one period to the next.

30 Analysis of Absorption Spectra-Titration Data

Suppose we have a mixture of substances which we carry through a sequence of changes by varying some state variable. This process is called a *titration* with respect to the state variable, and the successive values of the state variable are called the titration levels. At each titration level, we measure some properties of the mixture; this data can then be modeled by a suitable mathematical description of the chemistry involved. When the measured properties are themselves a spectrum of responses to some sequence of stimuli, such as an absorption spectrum sampled over a range of wavelengths, a particularly elegant method of analysis called the "Singular-Value Decomposition" (SVD) method due to Richard Shrager and Richard Hendler can be used. (see Analysis of the spectra and redox properties of pure cytochromes aa3, Biophysical J. Vol. 49, pp. 717–729, March, 1986.) The stimulus could even consist of the passage of time; thus kinetics data could be analyzed with this method. We will describe this method below in the context of an example with a titration with respect to hydrogen ion level, pH. (Hendler's actual experiments involved a titration with respect to electron content (i.e., voltage) corresponding to changes in the component substances of the respiratory chain in a mitochondrial membrane.)

For our example, suppose we have data values in a matrix $A[1:m, 1:n]$ such that A_{ik} is the absorbance at wavelength wl_i of the mixture at titration pH-level pH_k. We will take $m = 70$ and $n = 27$. The 70 wavelengths are equally spaced in the range from 350 nm to 650 nm. The 27 pH levels are equally-spaced in the range from pH 3 to pH 11.

We suppose there are $t-1$ substances in the mixture which appear in greater concentration as the pH level is increased. The value $t-1$ is unknown and is to be estimated from the data. The relative fraction of the jth substance is given by the Henderson-Hasselbalch function: $f_j(pH) = 1/(1+10(pKj - pH))$, where pKj is the pH-value at which substance j's concentration is 50% of its maximum value, and f_j is the function of the transition curve of substance j. With a more general choice of f_j, it would be possible to model the disappearance of substance j, as well as its appearance. Let c_j denote the maximum concentration of substance j that occurs at "infinite" pH. We also suppose there is an unchanging "background" complex of materials that we call "substance t". Its transition function is just $ft(pH) = 1$, and its limiting concentration is the constant c_t.

Substance t and the other $t-1$ substances are each assumed to have a characteristic absorption spectrum; $g_j(wl)$ denotes the absorbance seen for 1 unit of substance j at wavelength wl. Often, but not always, g_j is approximately a gaussian bell-shaped curve centered at a characteristic central wavelength. Absorption spectra are assumed to be additive; the spectrum for several substances is the sum of the spectra for each individual substance.

Now we can compute the overall absorbance at wavelength wl_i and pH-level pH_k as:

$$b_{ik} = \sum_{j=1}^{t} g_j(wl_i) c_j f_j(pH_k), \quad \text{for } 1 \le i \le m, \text{ and } 1 \le k \le n.$$

Here, $c_j f_j(pH_k)$ is the concentration of substance j at pH-level pH_k, and $g_j(wl_i)c_j f_j(H_k)$ is the absorbance at wavelength wl_i due to this concentration of substance j. When the pK_j values appearing in the f_j functions are correctly specified, b_{ik} should match the data value A_{ik}.

We can express this model in matrix form as follows. Let F be an $n \times t$ matrix with $F_{kj} = f_j(pH_k)$. Note that F col $t = 1$ identically. Let D be an $m \times t$ matrix with $D_{ij} = g_j(wl_i)c_j$. Note that D col t is the baseline spectrum at the m wavelengths being monitored. Let B be the $m \times n$ matrix with $B_{ik} = b_{ik}$. Then $B = DF^T$, and $B \approx A$. (B would equal A exactly, except for measurement error.)

Given the data A, we wish to estimate t, and $pK_1, \ldots, pK_t - 1$, and D_{ij} for $1 \leq i \leq m$ and $1 \leq j \leq t$, and F_{kj} for $1 \leq k \leq n$ and $1 \leq j \leq t$. This may be done using the matrix singular-value decomposition. Note that if we know $pK_1, \ldots, pK_t$, we then know the elements of the matrix F, and conversely.

Any $m \times n$ real matrix A can be written as $A = USV^T$, where U is an $m \times m$ orthogonal matrix, S is an $m \times n$ diagonal matrix with decreasing non-negative diagonal values called the singular values of A, and V is an $n \times n$ orthogonal matrix. The number of positive singular values exhibited in S is the rank of A.

In our example, we thus use MLAB to write $A = USV^T$. Now we discard the zero or almost zero singular values from S, leaving t positive singular values in a $t \times t$ diagonal matrix $\underline{S}$. We have thus determined the value t; recall that the number of non-background substances is $t - 1$. We also discard the last $m - t$ columns of U and the last $n - t$ columns of V that correspond to the discarded singular values to obtain $\underline{U} = U$ col $1 : t$ and $\underline{V} = V$ col $1 : t$. We now have $A \approx \underline{USV}^T$. "Almost zero" can be defined as less than about 1% to 2% of the maximum singular value, however, this is a dangerous assumption. Shrager suggests we check the noise apparent in the corresponding columns of V; only singular values corresponding to significantly noisy columns of V should be discarded.

Now we may factor $\underline{V}$ as $\underline{V} = FH$ where H is a $t \times t$ matrix, so $A = \underline{US}H^T F^T$, and hence we may obtain $D = \underline{US}H^T$, since $A = DF^T$. In order to factor $\underline{V}$ into explicitly-known matrices F and H, we use curve-fitting. Let:

$$q_j(x) = H_{tj} + \sum_{r=1}^{t-1} H_{rj} f_r(x; pH_r)$$

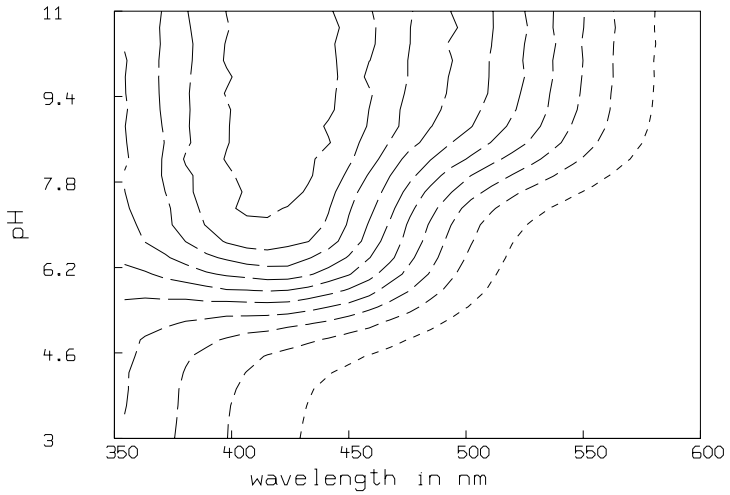
Note that $[q_j(pH_1, \ldots, q_j(pH_k)]^T = \underline{V}$ col j. Thus we may use MLAB to estimate H_{1j}, H_{2j}, $\ldots$, $H_{(t-1)j}$, H_{tj}, and $pK_1, \ldots, pK_{t-1}$ by fitting the model q_j to the $n \times 2$ data matrix $X_j :=$ (3 : 11!27)&$'$($\underline{V}$ col j), for $j = 1, \ldots, t$. (Recall that in our example we are using 27 equally-spaced pH-values between 3 and 11.) Shrager points out that this is best done by simultaneously fitting q_j to X_j using the weight $(\underline{S}_{jj})^2$ for $1 \leq j \leq t$.

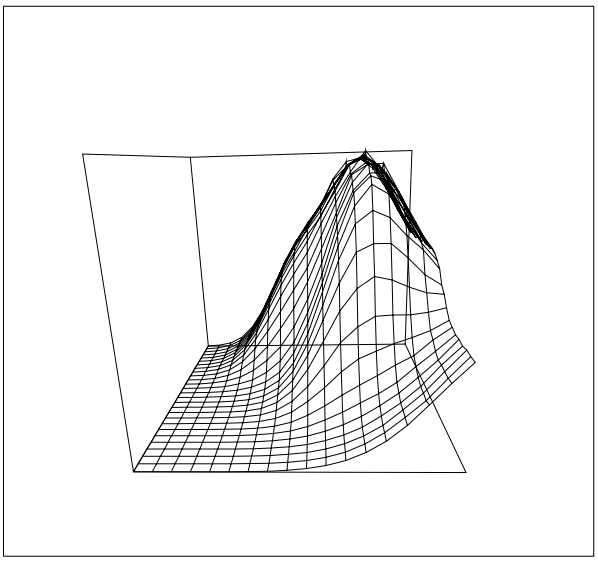
Now having estimated t, and $pK_1, \ldots, pK_{t-1}$, and the t^2 elements of H, we may directly obtain the matrix F and then D. The columns of D are the spectra of the t substances taken at their maximum concentrations; thus $(D \text{ col } j)^T = c_j[g_j(wl_1), \ldots, g_j(wl_m)]$. If we know these spectral curves independently, we can deduce the concentration scale factors $c_1, \ldots, c_t$. Looking at these curves also serves as a check on the entire modeling process. Below is a particular MLAB dialog showing the SVD method applied for our example.

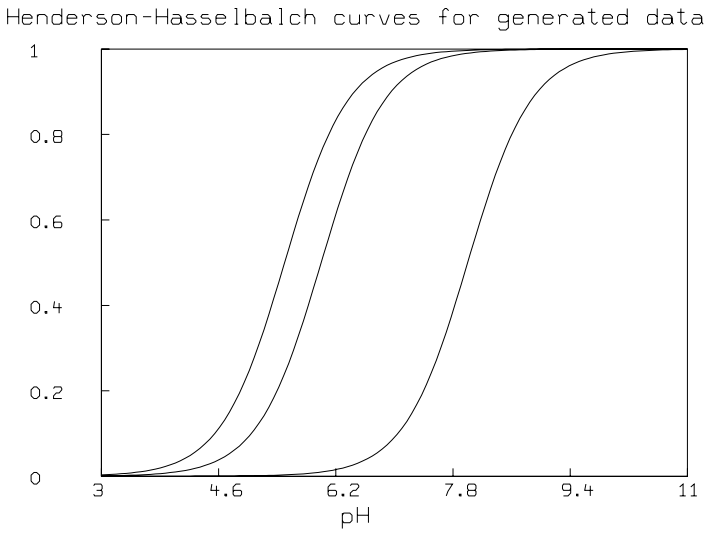
First we read in our data matrix A, where A is a 70×27 matrix whose rows correspond to the wavelengths $350 : 600!70$ (nm), and whose columns correspond to the pH values $3 : 11!27$. A_{ij} is the absorbance observed at wavelength-level i and pH-level j.

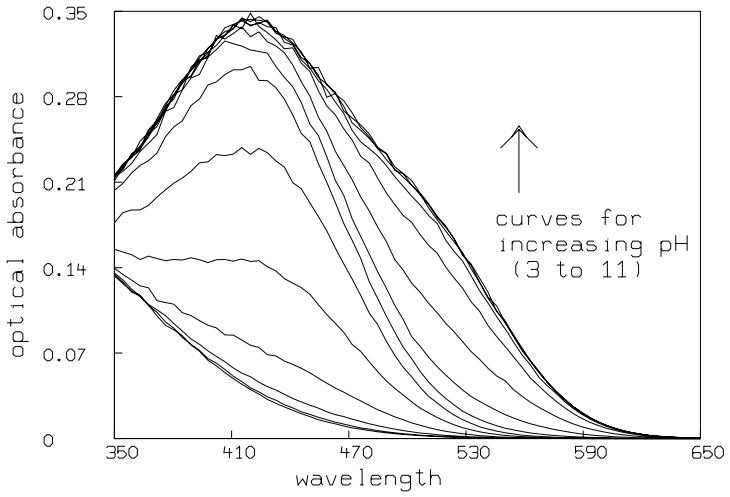
```
*a = read(dataf,70,27)
```

Some graphs which show the nature of the data A are given below. All of these graphs were drawn with MLAB.






Henderson-Hasselbalch curves for generated data
1
0.8
0.6
0.4
0.2
0
3
4.6
6.2
7.8
9.4
11
pH



Now we compute the singular value decomposition of A and decide the value of t.

```
* m=nrows(a); n=ncols(a)
*
* u=svd(a)
* s=(u row 1)'
* v=u row (m+2):(n+m+1)
* u = u row 2:(m+1)
* t=nrows(compress(((s col 1)>.025)&'s));
* type s row 1:(t+3); "type the singular values 3 beyond the t-th"

    : a  7 by 1 matrix

  1: 6.90549919
  2: .964371539
  3: .458168815
  4: 4.74121723E-2
  5: .023760869
  6: 2.21695634E-2
  7: .020740888

* v=v col 1:t; u= u col 1:t
*
* fct f1(ph)=1/(1+10^(pk1-ph))
```

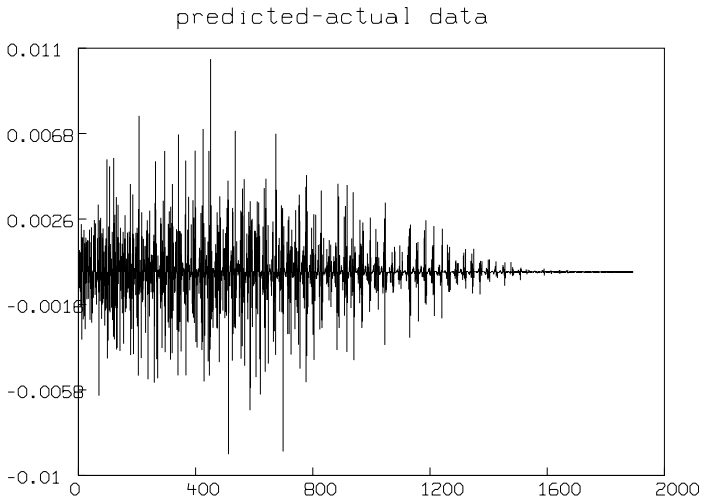
```
* fct f2(ph)=1/(1+10^(pk2-ph))
* fct f3(ph)=1/(1+10^(pk3-ph))
* fct b1(ph)=h[t,1]+h[1,1]*f1(ph)+h[2,1]*f2(ph)+h[3,1]*f3(ph)
* fct b2(ph)=h[t,2]+h[1,2]*f1(ph)+h[2,2]*f2(ph)+h[3,2]*f3(ph)
* fct b3(ph)=h[t,3]+h[1,3]*f1(ph)+h[2,3]*f2(ph)+h[3,3]*f3(ph)
* fct b4(ph)=h[t,4]+h[1,4]*f1(ph)+h[2,4]*f2(ph)+h[3,4]*f3(ph)
*
* h=.3*shape(4,4,ran on 0^^16); "generate random guesses for h"
* pk1= 7; pk2 = 7.2; pk3= 6; "choose guesses for pk1,pk2,pk3"
*
* wl=350:650!70;
* phv=3:11!27
*
* lsqrpt=9;maxiter=30; symdsw=0
* fit(h,pk1,pk2,pk3),b1 to phv&'(v col 1) with wt s[1]^^n,\
> b2 to phv&'(v col 2) with wt s[2]^^n,\
> b3 to phv&'(v col 3) with wt s[3]^^n,\
> b4 to phv&'(v col 4) with wt s[4]^^n
Begin iteration 1 bestsosq=5.511887e+01
Begin iteration 2 bestsosq=2.741844e+00
Begin iteration 3 bestsosq=6.954775e-01
Begin iteration 4 bestsosq=4.391224e-01
Begin iteration 5 bestsosq=2.770091e-01
Begin iteration 6 bestsosq=1.525704e-01
Begin iteration 7 bestsosq=6.386165e-02
Begin iteration 8 bestsosq=3.205128e-02
Begin iteration 9 bestsosq=2.200377e-02
Begin iteration 10 bestsosq=1.412370e-02
Begin iteration 11 bestsosq=1.203038e-02
Begin iteration 12 bestsosq=1.098136e-02
Begin iteration 13 bestsosq=1.006403e-02
Begin iteration 14 bestsosq=9.237729e-03
Begin iteration 15 bestsosq=8.498725e-03
Begin iteration 16 bestsosq=8.261531e-03
Begin iteration 17 bestsosq=3.447120e-03
Begin iteration 18 bestsosq=3.094425e-03
Begin iteration 19 bestsosq=3.082968e-03
Begin iteration 20 bestsosq=3.078296e-03
final parameter values
      value              error               dependency    parameter
 -0.04313096632      0.002075088787          0.8681882157   H[1]
  -0.5091112274      0.006380218701          0.9001594031   H[2]
```

```
  -0.4141844706      0.008460681648          0.880494897   H[3]
  -0.5124442081       0.02935544387         0.9040697608   H[4]
 -0.06083337034       0.01144532059         0.9974810125   H[5]
  0.2027353714        0.03290432573         0.9978176388   H[6]
 0.00747866897        0.07009215024         0.9989876919   H[7]
    2.848373276         0.4970101962         0.9998054387   H[8]
  -0.1143730484       0.01060867204         0.9973495801   H[9]
 -0.09979224416       0.03273355056          0.998006569   H[10]
  0.6236741131        0.06485462555         0.9989311313   H[11]
  -2.542220886         0.5055039097         0.9998299828   H[12]
 -0.04089842681      0.0009856221674          0.8092088134   H[13]
  0.2031173917       0.002508541316           0.78909474   H[14]
  -0.3226308862      0.003843686277          0.8109168102   H[15]
   0.152892387        0.01220313549          0.8187234648   H[16]
   8.006947711        0.01676401014           0.720544985   PK1
   6.058377382        0.05911112374          0.9828623017   PK2
   5.553356498        0.04694757727          0.9790372858   PK3
20 iterations
CONVERGED
best weighted sum of squares = 3.075587e-03
weighted root mean square error = 5.878534e-03
weighted deviation fraction = 3.081091e-03
R squared = 9.844744e-01
*
* s=diag(s row 1:t);
* d=u*s*h'
* del u,s
*
* f=(f1 on phv)&'(f2 on phv)&'(f3 on phv)&'1
* e=a-d*f'; "e is the error in estimating a"
* del fcovp,sosq,depvals,stdest
```

We have estimated t $(= 4)$ and pK_1, pK_2, and pK_3 ($pK_1 = 8.007$, $pK_2 = 6.058$, $pK_3 = 5.553$). We have also computed the matrix D. Now we present some pictures drawn in MLAB showing the results after this analysis of the data.

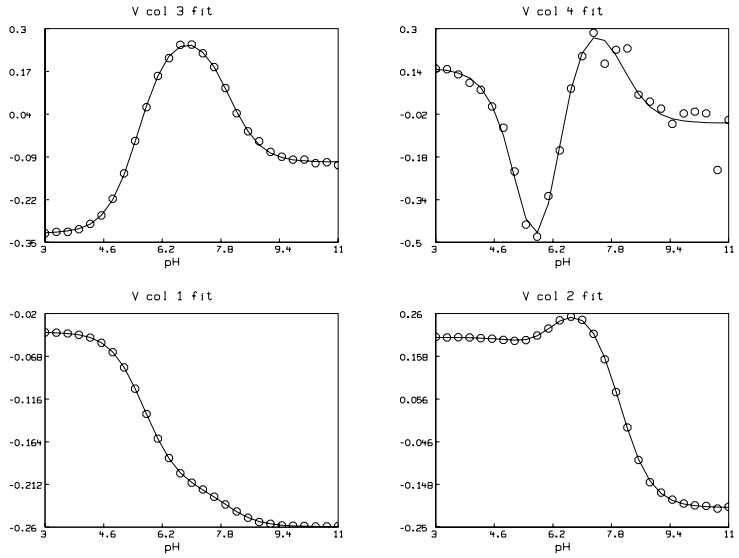
```
*e=list(e)
*vl=1:nrows(e)
*e = mesh(vl&'0,vl&'e)
*del vl
*draw e lt alternate
*top title "predicted-actual data"
```

```
*view
```



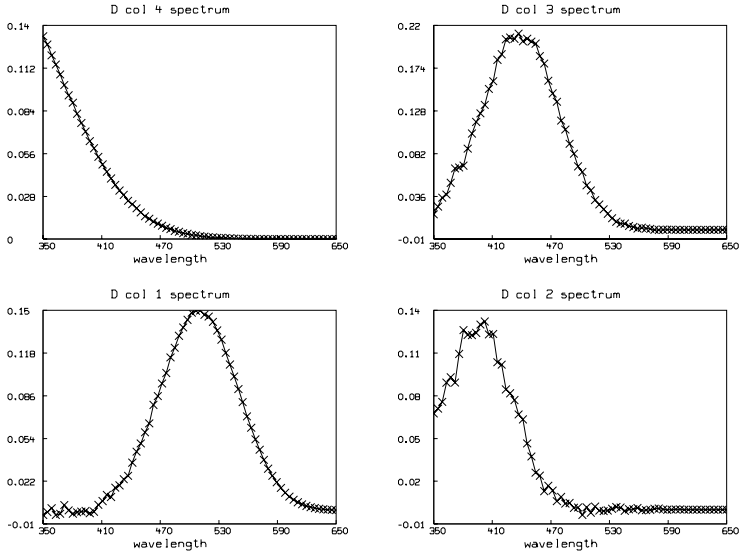

```
*del w
*mz=phv&'(v col 1)
*draw mz lt none pt octagon
*draw points(b1,phv)
*top title "V col 1 fit"
*bottom title "pH"
*frame 0 to .5, 0 to .5
*w1=w

*mz=phv&'(v col 2)
*draw mz lt none pt octagon
*draw points(b2,phv)
*top title "V col 2 fit"
*bottom title "pH"
*frame .5 to 1, 0 to .5
*w2=w

*mz=phv&'(v col 3)
*draw mz lt none pt octagon
*draw points(b3,phv)
*top title "V col 3 fit"
*bottom title "pH"
*frame 0 to .5, .5 to 1
```

```
*w3=w

*mz=phv&'(v col 4)
*draw mz lt none pt octagon
*draw points(b4,phv)
*top title "V col 4 fit"
*bottom title "pH"
*frame .5 to 1, .5 to 1
*w4=w
*view
```




```
*del w1,w2,w3,w4
*mz=wl&'(d col 1)
*draw mz pt xpt
*top title "D col 1 spectrum"
*bottom title "wavelength"
*frame 0 to .5, 0 to .5
*w1=w

*mz=wl&'(d col 2)
*draw mz pt xpt
*top title "D col 2 spectrum"
*bottom title "wavelength"
*frame .5 to 1, 0 to .5
*w2=w
```

```
*mz=wl&'(d col 3)
*draw mz pt xpt
*top title "D col 3 spectrum"
*bottom title "wavelength"
*frame .5 to 1, .5 to 1
*w3=w

*mz=wl&'(d col 4)
*draw mz pt xpt
*top title "D col 4 spectrum"
*bottom title "wavelength"
*frame 0 to .5, .5 to 1
*w4=w
*view
```



Although the fit we show here is very good, in general the fit we obtain can be very sensitive to initial guesses for the parameters. Shrager and Hendler have developed a collection of heuristic techniques to acquire reasonable initial guesses. Also, the fact that the H_{ij} parameters appear linearly means that a two-stage fitting approach consisting of separately fitting just the linear and then the nonlinear parameters could be used. (See Golub and Peryra, SIAM J. of Numerical Analysis, Vol. 10, No. 2, pp. 413:432, April 1973.)

31 Chemical Kinetics Modeling

The MLAB advanced mathematical and statistical modeling system is an unparalleled tool for mathematical modeling; take a look at the following example. Only MLAB can solve a kinetics-modeling problem like this so easily! (It's just as easy to solve enzyme kinetics, multiple site binding equilibrium, or any other of a wide variety of problems.)

Below we show an interesting example of MLAB modeling for dimer kinetics. Suppose we have two substances, A and B which bind to form a complex C, and the substance C, in turn, binds with itself to form a dimer D. We thus have:

$$A + B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C, \qquad C + C \underset{k_4}{\overset{k_3}{\rightleftharpoons}} D.$$

Suppose further we mix 2 mmoles of A and 3 mmoles of B and measure the concentration in mmoles of both C and D at ten equally-spaced times between 7 and 70 seconds. From this data we wish to estimate the association and dissociation constants k_1, k_2, k_3, and k_4. We may proceed in MLAB as follows.

First we read in the data consisting of values of $c(t)$ and $d(t)$ given at the common times $7 : 70!10$. Although common times are used here, this is *not* required.

```
*data = (7:70!10) &' read(ddata,10,2)
*type data

     time    c      d

  1: 7       1.065   0.0058
  2: 14      1.383   0.2203
  3: 21      0.9793  0.4019
  4: 28      1.107   0.3638
  5: 35      0.7289  0.456
  6: 42      0.7236  0.5014
  7: 49      0.4674  0.715
  8: 56      0.6031  0.4723
  9: 63      0.6149  0.7219
 10: 70      0.3369  0.7294

*cdata = data col 1:2
*ddata = data col (1,3)
```

Now we define our kinetic model so that $c(t)$ is the concentration of c in mmoles at time t and $d(t)$ is the concentration of d in mmoles at time t.

```
* fct c't(t)=k1*(a0-c-2*d)*(b0-c-2*d)-k2*c-2*d't(t)
* fct d't(t)=k3*c*c-k4*d
* initial c(0)=0
* initial d(0)=0
* a0=2;b0=3
```

Now we guess the values of k_1, k_2, k_3, and k_4. We may use the results of equilibrium studies, analyzed by MLAB, to know values for the ratios k_1/k_2 and k_3/k_4.

```
* k1=.02;k2=.002; k3=.02;k4=.002
* constraints q={k1>0,k2>0,k3>0,k4>0}
```

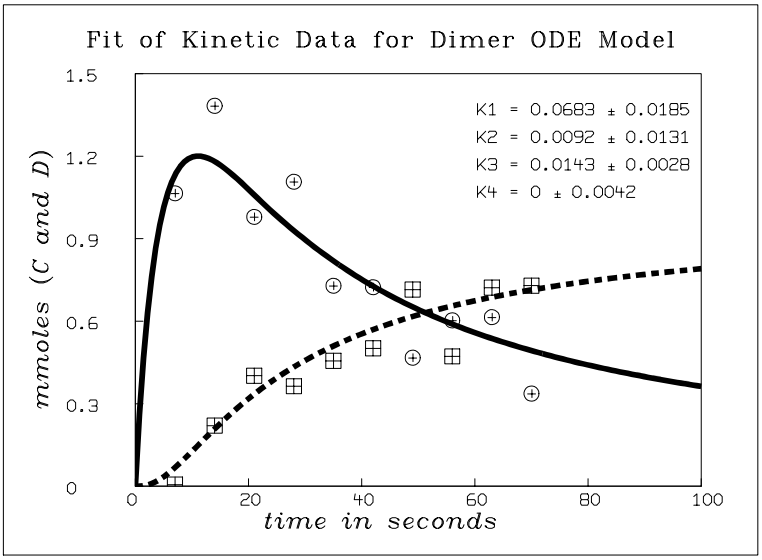
Now we may curve-fit the two ode-system-defined functions, c and d, to estimate k_1, k_2, k_3, and k_4.

```
* fit(k1,k2,k3,k4), c to cdata, d to ddata, constraints q
final parameter values
      value               error                dependency    parameter
 0.06830069826        0.01853684638           0.5090866122   K1
 0.009215472845       0.01307266967           0.5011226583   K2
 0.01429036692        0.002818099621          0.7828534111   K3
 1.616224063e-19      0.004158477431          0.7620523838   K4
6 iterations
CONVERGED
best weighted sum of squares = 2.222001e-01
weighted root mean square error = 1.178453e-01
weighted deviation fraction = 1.203378e-01
lagrange multiplier[4] = -3.928493584
```

Now we may draw the results of the curve-fit.

```
* m=integrate(c't,d't,0:100!140)
* draw m col (1,2) color red
* draw m col (1,4) color green lt dashed
* draw cdata pt circle lt none color red
* draw ddata pt circle lt none color green
* bottom title "time in seconds"
* left title "mmoles (C and D)"
*
```

```
* oformat = nformat; nformat ="%4.4lf"
* v=strval(stdest[1]); s=strval(k1)+" '25TF'R "+substr(v,7:strlen(v))
* title s at (.6,.8) ffract size .015
* v=strval(stdest[2]); s=strval(k2)+" '25TF'R "+substr(v,7:strlen(v))
* title s at (.6,.75) ffract size .015
* v=strval(stdest[3]); s=strval(k3)+" '25TF'R "+substr(v,7:strlen(v))
* title s at (.6,.7) ffract size .015
* v=strval(stdest[4]); s=strval(k4)+" '25TF'R "+substr(v,7:strlen(v))
* title s at (.6,.65) ffract size .015
* nformat=oformat

* view
```



32 Hypothesis Testing

Hypothesis testing is a major paradigm in statistics. It is closely linked with the computation of specified probability distribution functions. The basic notion is simple. We obtain a sample value v of a random variable T and we ask how probable it is that the sample value v would appear *under a hypothesis H which makes the distribution of T well-defined*. If the probability that, under the hypothesis H, v or a more extreme value of T appears is small, we take this as evidence that the hypothesis H is unlikely to be true. In other words, we conclude that the test of the hypothesis H has not supported H.

The random variable T is called the *test statistic*. If many samples of various random variables are taken, they are often combined in some, possibly quite elaborate, manner to obtain a single sample of a derived test statistic T. In other cases, the test statistic may be a vector-valued random variable with a multivariate distribution function. For example, the test statistic associated with the famous t-test for testing the hypothesis that two normally-distributed random variables have the same mean is the difference between the means, or variance-adjusted means, of two sets of sample values corresponding to the two random variables being studied.

In order to compute the probability p that, under the hypothesis H, the sample value v or a more extreme value of T appears, we must be able to compute $P(T \leq v \mid H)$, which is the distribution of the test statistic T under the hypothesis H. We shall denote a random variable with this distribution by T_H. The hypothesis H must be such that the distribution function of T_H is known; this means that H is often of the form: "there is no difference between two sets of samples", since it is generally easier to deduce the distribution of T_H in this case. Thus, H is called the *null hypothesis*, meaning the "no difference" hypothesis.

Suppose that the distribution function of T_H is $G(x) := P(T_H \leq x)$. Also suppose that the density function $dG(x)/dx$ is a unimodal "bell-shaped" curve, so that the extreme sample values of T_H lie toward $+\infty$ and $-\infty$. Suppose the value v is given as a sample value of T. We may compute, for example, $p = P(|T_H - E(T_H)| \geq |v - E(T_H)|)$. This is a particular form of a so-called two-tailed test. p is the probability that the value v or a "more extreme" value occurs as a sample value of T, given H. If p is sufficiently small, we may reject the null hypothesis H as implausible in the face of the "evidence" v. We call such a probability p the *plausibility probability* of H, given v.

If the test statistic T_H were known to be non-negative and the density function $dG(x)/dx$ were a function, such as the exponential density function, which decreases on $[0, \infty)$, then we might use a so-called one-tail test, where we compute the probability $p = P(T_H \geq v)$.

In general, we may specify a particular value α as our criterion of "sufficiently small" and we may choose any subset S of the range of T such that $P(T_H \notin S) = \alpha$. Then if $v \notin S$, the null hypothesis H may be judged implausible. S is called the acceptance set, because, when $v \in S$, the null hypothesis H is not rejected. The value $\alpha = P(T_H \notin S)$ is the probability that we make a mistake if we reject H when $v \notin S$.

How should the acceptance set S be chosen? S should be chosen to minimize the chance of making the mistake of accepting H when H is, in fact, false. But, this can only be done rigorously with respect to an alternate hypothesis H_a such that the distribution of T given H_a is known. We must postulate that H and H_a are the only non-negligable possibilities. Sometimes, $H_a = \neg H$ is a suitable alternate hypothesis, but more often, this is not suitable. Given H_a, the probability we falsely accept H when the alternate hypothesis H_a is true is $P(T_{H_a} \in S) =: \beta$, and we can choose S such that $P(T_H \notin S) = \alpha$ while $P(T_{H_a} \in S) = \beta$ is minimized.

The value $P(T_{H_a} \notin S) = 1 - \beta$ is called the *power* of the test of the null hypothesis H versus the alternate hypothesis H_a. Choosing S to minimize β is the same as choosing S to maximize the power $1 - \beta$.

If we don't care about achieving the optimal power of the test with respect to a specific alternate hypothesis, but merely wish to compute the plausibility probability that v or a more extreme sample value of T would occur given H, in a fair manner, then we may proceed as follows.

Let $m = median(T_H)$; thus, $P(T_H \geq m) = 0.5$. Now, if $v < m$, choose $r_1 = v$ and r_2 as the value such that $P(m < T_H < r_2) = P(v < T_H < m)$, otherwise choose $r_2 = v$ and choose r_1 as the value such that $P(r_1 < T_H < m) = P(m < T_H < v)$. Then the two-tail plausibility probability $p = 1 - P(r_1 < T_H < r_2)$. If $v < m$, $p = 2P(T_H \leq v)$, otherwise, if $v \geq m$, $p = 2(1 - P(T_H \leq v))$.

If we know that the only values more extreme than v which we wish to consider as possible are those in the same tail of the density function that v lies in, then we may compute the one-tail plausibility probability as $p = P(T_H \leq v)$ if $v \leq m$ and $p = P(T_H \geq v)$ if $v > m$.

Consider testing the null hypothesis H versus the alternate hypothesis H_a using a sample value v of the test random variable T with the acceptance set S. We have the following outcomes.

	$v \in S$	$v \notin S$
H	accept H prob $1 - \alpha$ correct	reject H prob α rejection error
H_a	accept H prob β acceptance error	reject H prob $1 - \beta$ correct

$$\begin{aligned}
\alpha &= P(T_H \notin S) = P(\text{we falsely reject } H \mid H) && \text{(rejection error)} \\
1 - \alpha &= P(T_H \in S) = P(\text{we correctly accept } H \mid H) && \text{(acceptance power)} \\
\beta &= P(T_{H_a} \in S) = P(\text{we falsely accept } H \mid H_a) && \text{(acceptance error)} \\
1 - \beta &= P(T_{H_a} \notin S) = P(\text{we correctly reject } H \mid H_a) && \text{(rejection power)}
\end{aligned}$$

Let Q be the sample space of the test statistic T. The hypothesis H and the alternate hypothesis H_a may each hold at different points of Q, so that H and H_a define corresponding complementary Bernouilli random variables on Q. Thus $H(q) = 1$ if H holds at the sample point $q \in Q$ and $H(q) = 0$ if H does not hold at the sample point q; H_a is defined on Q in the same manner. We assumed above that either $H(q) = 1$ for all $q \in Q$ or $H(q) = 0$ for all $q \in Q$, but this universal applicability of H or H_a may be relaxed. Let $P(\{q \in Q \mid H(q) = 1\})$ be denoted by $P(H)$ and let $P(\{q \in Q \mid H_a(q) = 1\})$ be denoted by $P(H_a)$.

$P(H)$ is called the *incidence* probability of H and $P(H_a)$ is called the *incidence* probability of H_a. As before, we postulate that $P(H) = 1 - P(H_a)$. Often $P(H)$ is 0 or 1 as we assumed above, but it may be that $0 < P(H) < 1$. In this latter case, our test of hypothesis can be taken as a test of whether $H(q) = 1$ or $H_a(q) = 1$ for the particular sample point q at hand for which $T(q) = v$; the test statistic value v may be taken as evidence serving to increase or diminish the probability of $H(q) = 1$.

Note that we cannot compute the "posterior" probability that $H(q) = 1$ (and that $H_a(q) = 0$), or conversely, *unless* we have the "prior" incidence probability of H being true in the sample space Q. In particular,

$$\begin{array}{lcll} P(H(q)=1 & \& & T(q)\in S) & = & (1-\alpha)P(H) \\ P(H(q)=1 & \& & T(q)\notin S) & = & \alpha P(H) \\ P(H(q)=0 & \& & T(q)\in S) & = & \beta(1-P(H)) \\ P(H(q)=0 & \& & T(q)\notin S) & = & (1-\beta)(1-P(H)) \end{array}$$

Let us look at a particular case of an hypothesis test, namely the so-called F-test for equal variances of two normal populations.

Suppose $X_{11}, X_{12}, \ldots, X_{1n_1}$ are independent identically-distributed random variables distributed as $N(\mu_1, \sigma_1^2)$, and $X_{21}, X_{22}, \ldots, X_{2n_2}$ are independent identically-distributed random variables distributed as $N(\mu_2, \sigma_2^2)$. The corresponding sample-variance random variables are

$$S_1^2 = \sum_{j=1}^{n_1} (X_{1j} - \bar{X}_1)^2/(n_1 - 1) \quad \text{and} \quad S_2^2 = \sum_{j=1}^{n_2} (X_{2j} - \bar{X}_2)^2/(n_2 - 1),$$

where $\bar{X}_1 = \sum_{j=1}^{n_1} X_{1j}/n_1$ and $\bar{X}_2 = \sum_{j=1}^{n_2} X_{2j}/n_2$.

Let R denote the sample variance ratio S_1^2/S_2^2. Then $R \sim (\sigma_1^2/\sigma_2^2)F_{n_1-1,n_2-1}$, where F_{n_1-1,n_2-1} is a random variable having the F-distribution with $(n_1 - 1, n_2 - 1)$ degrees of freedom.

We take the null hypothesis H to be $\sigma_1/\sigma_2 = 1$, so that, given H, the test statistic R is known to be distributed as F_{n_1-1,n_2-1}. In order to determine the acceptance region S with maximal power for α fixed, we take the alternate hypothesis H_a to be $\sigma_1/\sigma_2 = a$. Then S is the interval $[r_1, r_2]$ where $P(r_1/a^2 \leq F_{n_1-1,n_2-1} \leq r_2/a^2) = \beta$ is minimal, subject to $1 - P(r_1 \leq F_{n_1-1,n_2-1} \leq r_2) = \alpha$.

Let $G(z) = P(F_{n_1-1,n_2-1} \leq z)$, the distribution function of F_{n_1-1,n_2-1}, and let $g(z) = G'(z)$, the probability density function of F_{n_1-1,n_2-1}. Then, we have $r_1 = root_z[g(z)g(h(z)/a^2) - g(h(z))g(z/a^2)]$ and $r_2 = h(r_1)$, where $h(z) = G^{-1}(1 - \alpha + G(z))$.

A simplified, slightly less powerful, way to choose the acceptance region S is to take $S = [r_1, r_2]$ where r_1 is the value such that $P(F_{n_1-1,n_2-1} \leq r_1) = \alpha/2$ and r_2 is the value such that $P(F_{n_1-1,n_2-1} \geq r_2) = \alpha/2$. Another way to select the acceptance region is to take $S = [1/r, r]$, where r is the value such that $P(1/r \leq F_{n_1-1,n_2-1} \leq r) = 1 - \alpha$. When $n_1 = n_2$, the acceptance region $[r_1, r_2]$ and the acceptance region $[1/r, r]$ are identical.

The foregoing clearly exemplifies the fact that there is a trade-off among the acceptance error probability β, the rejection error probability α, and the sample sizes (n_1, n_2). If we wish to have a smaller α, then we must have a greater β or greater values of n_1 and n_2. Similarly, β can only be reduced if we allow α or n_1 and n_2 to increase. In general, given any two of the test parameters α, β, or (n_1, n_2), we can attempt to determine the third, although a compatible value need not exist. Actually, in most cases, a fourth variable representing the distinction between the null hypothesis and the alternate hypothesis, such as the value a above, enters the trade-off balancing relations.

For the simplified two-tailed F-test with $S = [r_1, r_2]$, the relations among α, β, a, n_1, n_2, r_1, and r_2 are listed below.

$$\begin{aligned} P(F_{n_1-1,n_2-1} \leq r_1) &= \alpha/2, \\ P(F_{n_1-1,n_2-1} \geq r_2) &= \alpha/2, \\ P(r_1 \leq a^2 F_{n_1-1,n_2-1} \leq r_2) &= \beta. \end{aligned}$$

For the alternate case with $S = [1/r, r]$, the relations among α, β, a, n_1, n_2, and r are:

$$\begin{aligned} P(1/r \leq F_{n_1-1,n_2-1} \leq r) &= 1 - \alpha \\ P(1/r \leq a^2 F_{n_1-1,n_2-1} \leq r) &= \beta. \end{aligned}$$

In order to reduce the number of unknowns, we may postulate that n_1 and n_2 are related as $n_2 = \theta n_1$, where θ is a fixed constant.

The value of β is determined above by the distribution of $a^2 F_{n_1-1,n_2-1}$, because for the F-test, the test statistic, assuming the alternate hypothesis $\sigma_1/\sigma_2 = a$, is the random variable $a^2 F_{n_1-1,n_2-1}$ whose distribution function is just the F-distribution with the argument scaled by $1/a^2$. In other cases, the distribution of the alternate hypothesis test statistic T_{H_a} is more difficult to obtain.

If we take seriously the dichotomy $\sigma_1/\sigma_2 = 1$ vs. $\sigma_1/\sigma_2 = a$ as the *only* two possibilities, then a one-tailed F-test is most appropriate. Suppose $a > 1$. Then we take $S = [-\infty, r_2]$, and we have the relations: $P(F_{n_1-1,n_2-1} \geq r_2) = \alpha$, and $P(a^2 F_{n_1-1,n_2-1} \leq r_2) = \beta$. If $a < 1$, then with the null hypothesis $\sigma_1/\sigma_2 = 1$ and the alternate hypothesis $\sigma_1/\sigma_2 = a$, we should take $S = [r_1, \infty]$. Then, $P(F_{n_1-1,n_2-1} \leq r_1) = a$, and $P(a^2 F_{n_1-1,n_2-1} \geq r_1) = \beta$.

Generally, hypothesis testing is most useful when a decision is to be made. Instead, for example, suppose we are interested in the variance ratio $(\sigma_1/\sigma_2)^2$ between two normal populations for computational purposes. Then it is preferable to use estimation techniques and confidence intervals to characterize (σ_1/σ_2), rather than to use a hypothesis test whose only useful outcome is "significantly implausible", or "not significantly implausible" with the significance level α (which is the same as the rejection error probability).

Let r_1 satisfy $P(F_{n_1-1,n_2-1} \leq r_1) = \alpha_1$, and let r_2 satisfy $P(F_{n_1-1,n_2-1} \leq r_2) = 1 - \alpha_2$, with $\alpha_1 + \alpha_2 = \alpha < 1$. Then $P((\sigma_1/\sigma_2)^2 r_1 > R$ or $R > (\sigma_1/\sigma_2)^2 r_2) = \alpha_1 + \alpha_2$, and $P((\sigma_1/\sigma_2)^2 r_1 < R$ and $R < (\sigma_1/\sigma_2)^2 r_2) = 1 - \alpha_1 - \alpha_2 = P((\sigma_1/\sigma_2)^2 < R/r_1$ and $R/r_2 < (\sigma_1/\sigma_2)^2) = P(R/r_2 < (\sigma_1/\sigma_2)^2 < R/r_1)$.

Thus, $[R/r_2, R/r_1]$ is a $(1-\alpha)$-*confidence interval* which is an interval-valued random variable that contains the true value (σ_1/σ_2) with probability $1 - \alpha$. The length of this interval is minimized for $n_2 > 2$ by choosing α_1 and α_2, subject to $\alpha_1 + \alpha_2 = \alpha$, such that $G^{-1}(\alpha_1)^2 g(G^{-1}(1 - \alpha_2)) - G^{-1}(1 - \alpha_2)^2 g(G^{-1}(\alpha_1)) = 0$, where $G(x) = P(F_{n_1-1,n_2-1} \leq x)$ and where $g(x) = G'(x)$, the probability density function of F_{n_1-1,n_2-1}. Then $\alpha_1 = root_z(G^{-1}(z)^2 g(G^{-1}(1+\alpha-z)) - G^{-1}(1-\alpha+z)^2 g(G^{-1}(z))$, and $\alpha_2 = \alpha - \alpha_1, r_1 = G^{-1}(\alpha_1)$, and $r_2 = G^{-1}(1 - \alpha_2)$.

Let v denote the observed sample value of R. Then $[v/r_2, v/r_1]$ is a sample $(1 - \alpha)$-confidence interval for $(\sigma_1/\sigma_2)^2$.

The MLAB mathematical and statistical modeling system contains functions for various statistical tests and also functions to compute associated power and sample-size values. Let us consider an example focusing on the simplified F-test discussed above. We are given the following data:

```
x1:  -1.66, 0.46, 0.15, 0.52, 0.82, -0.58, -0.44, -0.53, 0.4, -1.1
x2:  3.02, 2.88, 0.98, 2.01, 3.06, 2.95, 3.4, 2.76, 3.92, 5.02, 4, 4.89, 2.64, 3.08
```

We may read this data into two vectors in MLAB and test whether the two data sets x_1 and x_2 have equal variances by using the MLAB F-test function `QFT`, which implements the $[1/r, r]$ simplified F-test specified above. The MLAB dialog to do this is exhibited below

```
*x1 = read(x1file); x2 = read(x2file);
*qft(x1,x2)
```

```
[F-test: are the variances v1 and v2 of 2 normal populations
 plausibly equal?]

null hypothesis H0: v1/v2 = 1. Then v1/v2 is F-distributed with
(n1-1,n2-1) degrees of freedom. n1 & n2 are the sample sizes.
The sample value of v1/v2 = 0.577562, n1 = 10, n2 = 15

The probability P(F < 0.577562) = 0.205421
This means that a value of v1/v2 smaller than 0.577562 arises about
20.542096 percent of the time, given H0.

The probability P(F > 0.577562) = 0.794579
This means that a value of v1/v2 larger than 0.577562 arises about
79.457904 percent of the time, given H0.

The probability: 1-P(0.577562 < F < 1.731416) = 0.377689
This means that a value of v1/v2 more extreme than 0.577562 arises
about 37.768896 percent of the time, given H0.
```

The $\alpha = .05$ simplified F-test acceptance set of the form $[1/r, r]$ can be computed directly as follows. `QFF` is the name of the F-distribution function in MLAB.

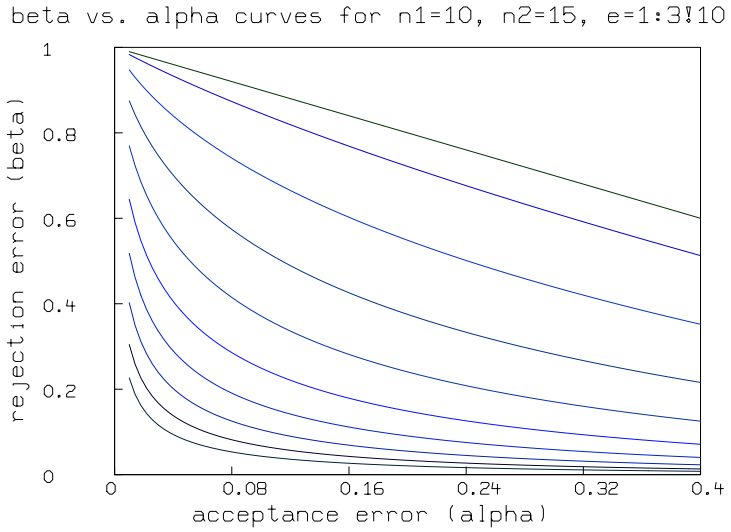
```
* n1 = nrows(x1)-1; n2 = nrows(x2)-1;
* fct rv(a) = root(z,.001,300,qff(1/z,n1,n2)+1-qff(z,n1,n2)-a)
* r = rv(.05)
* type 1/r,r
    = .285471776
  R = 3.50297326
```

Thus, a sample of $F_{9,14}$ will lie in $[.2855, 3.503]$ with probability .95.

The rejection error probability β can be plotted as a function of the acceptance error probability α for the sample sizes 10 and 15 by using the builtin function `QFB` as follows. The function `QFB`(α, n, θ, e) returns the rejection error probability value β that corresponds to the sample sizes n and θn, with the acceptance error probability α and the alternate hypothesis variance ratio e.

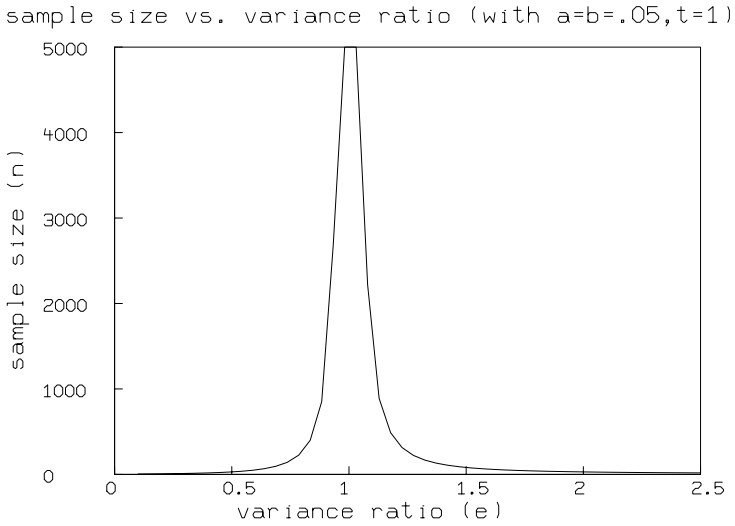
```
* fct b(a) = qfb(a,10,3/2,e)
* for e = 1:3!10 do {draw points(b,.01:.4!100)}
* left title "rejection error (beta)"
```

```
* bottom title "acceptance error (alpha)"
* top title "beta vs. alpha curves for n1=10, n2=15, e=1:3!10"
* view
```



Suppose we want to take n samples from each of two populations to be used to test whether these populations have the variance ratio 1 versus the variance ratio e, with acceptance error probability $\alpha = .05$ and rejection error $\beta = .05$. We can use the builtin function `QFN` to compute the sample size n as a function of e as follows. The function `QFN`$(\alpha, \beta, \theta, e)$ returns the sample size n that corresponds to the variance ratio numerator sample size, assuming the denominator sample size θn, and given that the acceptance error probability is α, the rejection error probability is β, and the alternate hypothesis variance ratio value is e.

```
* fct n(e) = qfn(.05,.05,1,e)
* draw points(n,.1:2.5!50)
* top title "sample size vs. variance ratio (with a=b=.05,t=1)"
* left title "sample size (n)"
* bottom title "variance ratio (e)"
* view
```


sample size vs. variance ratio (with a=b=.05,t=1)
sample size (n)
5000
4000
3000
2000
1000
0
0
0.5
1
1.5
2
2.5
variance ratio (e)

33 Sensitivity Analysis in MLAB

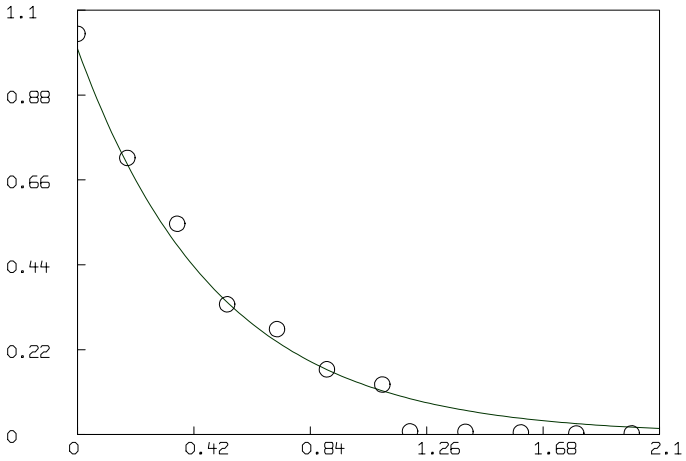
It may be interesting to look at the rate of change of a model function with respect to a parameter. This derivative function is a kind of sensitivity indicator that shows locally in a neighborhood of a fixed value of the paramter how sensitive that model is to a change in the specified parameter. This shows, among other things, where a new observation would be most valuable for determining the parameter of interest. There are other kinds of sensitivity measures, but we shall initially focus on this rate-of-change sensitivity.

If our model function is explicitly known as an algebraic formula, then it is a simple matter to use MLAB to compute the desired derivative function symbolically and evaluate and graph it over the desired range.

For example, suppose our model function is a simple one-component decaying exponential with the parameter k that has been determined to best fit the data points in the matrix d as shown below.

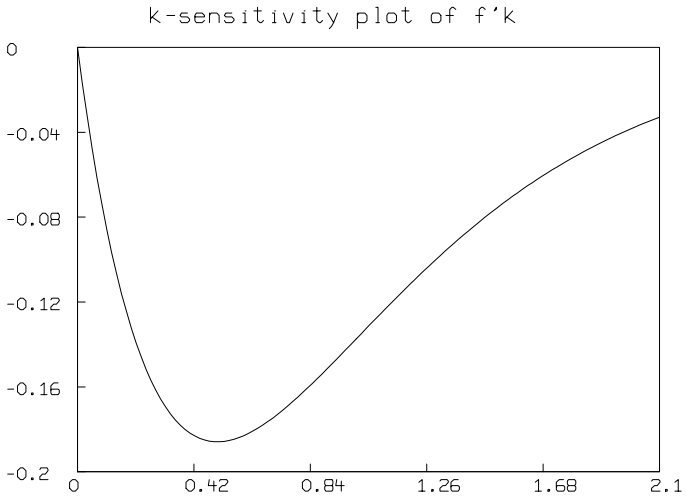
```
* function f(t)=exp(-k*t)
* constraints q={k>0}
* k=1;
* d = list(0,  1.0386,\
: .18, .717,  \
: .36, .5463, \
: .54, .33764,\
: .72, .27316,\
: .9,  .1692, \
: 1.1, .12954,\
: 1.2, .00815,\
: 1.4, .00707,\
: 1.6, .00555,\
: 1.8, .00291,\
: 2,   .003)
* d=shape(12,2,d)
* fit(k),f to d, constraints q
final parameter values
value             error               dependency          parameter
1.979230909       0.09787757296       -1.110223025e-16    K
4 iterations
CONVERGED
best weighted sum of squares = 1.878653e-02
weighted root mean square error = 4.132634e-02
weighted deviation fraction = 4.569453e-02
```

```
R squared = 9.849903e-01
no active constraints
* draw d pt circle lt none
* draw points(f,0:2.1!120) color red
* view
```



Then the k-sensitivity derivative curve can be computed and graphed as follows. Note this curve is a slice of a k-sensitivity surface defined by t and k, with k fixed at a value of interest.

```
* del w
* draw points(f'k,0:2.1!120)
* top title " k-sensitivity plot of f''k"
* view
```



If our model function is defined by a system of one or more differential equations, then computing the sensitivity derivative curves requires solving an augmented system of differential equations.

For example, if we have the simple kinetics differential equation model

$$\frac{dc(t)}{dt} = k_1(a_0 - c(t)) \cdot (b_0 - c(t)) - k_2 c(t) \quad \text{with } c(0) = 0,$$

with the parameters k_1 and k_2, then the function $\partial c(t)/\partial k_1$ can be computed numerically for a range of t-values by defining $y_1(t) = \partial c(t)/\partial k_1$, and introducing a differential equation for y_1, namely

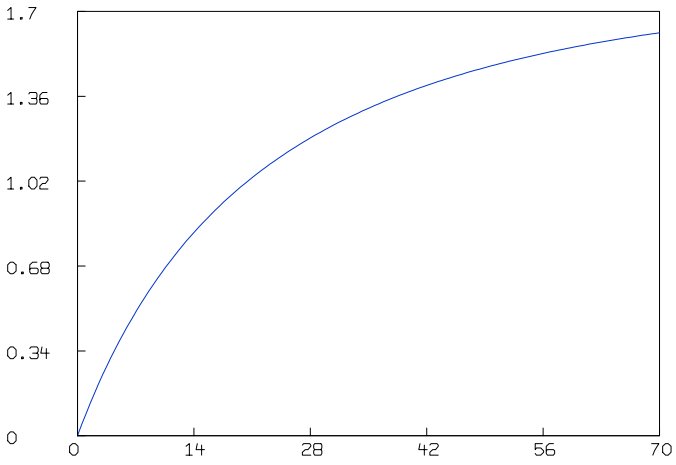
$$\frac{dy_1(t)}{dt} = \frac{\partial^2 c(t)}{\partial t \partial k_1}.$$

The function $\partial c(t)/\partial k_1$ can be similarly obtained by defining $y_2(t) = \partial c(t)/\partial k_2$, and introducing the differential equation

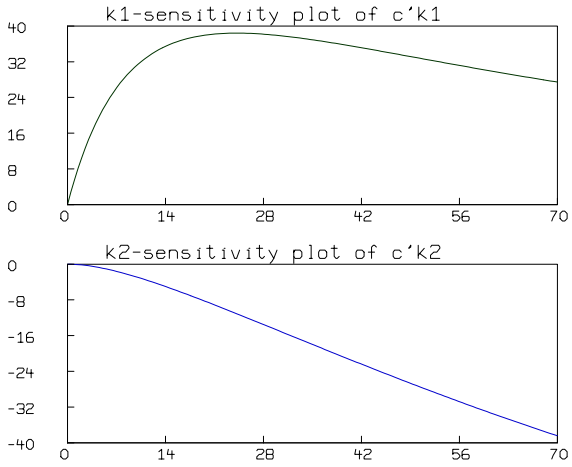
$$\frac{dy_2(t)}{dt} = \frac{\partial^2 c(t)}{\partial t \partial k_2}.$$

We show below how this example is handled in MLAB.

```
fct c't(t) = k1*(a0-c(t))*(b0-c(t))-k2*c(t)
initial c(0) = 0
k1 = .015;k2 = .002
a0 = 2;b0 = 3
draw points(c,0:70!140) color yellow
view
```



```
fct c'k1(t) = y1(t)
fct y1't(t)=c't'k1(t)
initial y1(0) = 0
fct c'k2(t) = y2(t)
fct y2't(t) = c't'k2(t)
initial y2(0) = 0
m=integrate(y1't,y2't,c't,0:70!140)
del w
draw m col (1,2) color red
frame 0 to 1, .5 to 1
top title "k1-sensitivity plot of c''k1"
w1=w

draw m col (1,4) color green
frame 0 to 1, 0 to .5
top title "k2-sensitivity plot of c''k2"
view
```



This picture is the same result that we obtain when we compute $\partial c/\partial k_1$ and $\partial c/\partial k_2$ by a centered finite difference approximation, as shown below.

```
k1v = k1; delta = .00015; k1=k1v+delta
m = points(c,0:70!140)
k1 = k1v-delta
m1 = points(c,0:70!140)
m col 2 = ((m col 2)-(m1 col 2))/(2*delta)
k1 = k1v
```

```
del w,w1
draw m color red
frame 0 to 1, .5 to 1
top title "numerical k1-sensitivity plot of c''k1"
w1 = w

k2v = k2; delta=.00002; k2=k2v+delta
m = points(c,0:70!140)
k2 = k2v-delta
m1 = points(c,0:70!140)
m col 2 = ((m col 2)-(m1 col 2))/(2*delta)
k2 = k2v
draw m color green
frame 0 to 1, 0 to .5
top title "numerical k2-sensitivity plot of c''k2"
```

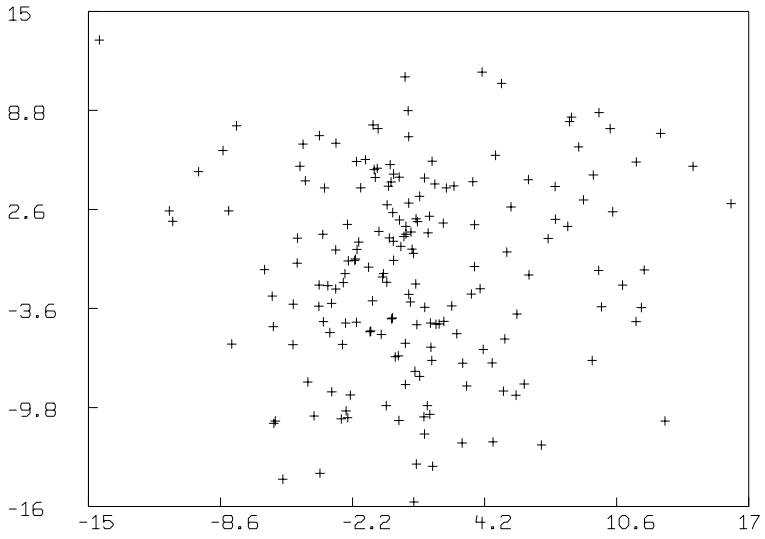
34 Cluster Analysis in *MLAB*

Situations often arise in which it is desirable to cluster a number of objects into smaller numbers of mutually exclusive groups, each having members that are as much alike as possible.

Such a clustering process depends on the 'distance' between the objects. Different definitions of 'distance' produce different clustering. *MLAB* has built-in functions for several well known clustering methods.

Below is an example of clustering using *MLAB*. The *kmeans* algorithm is used to cluster the input into 3 clusters.

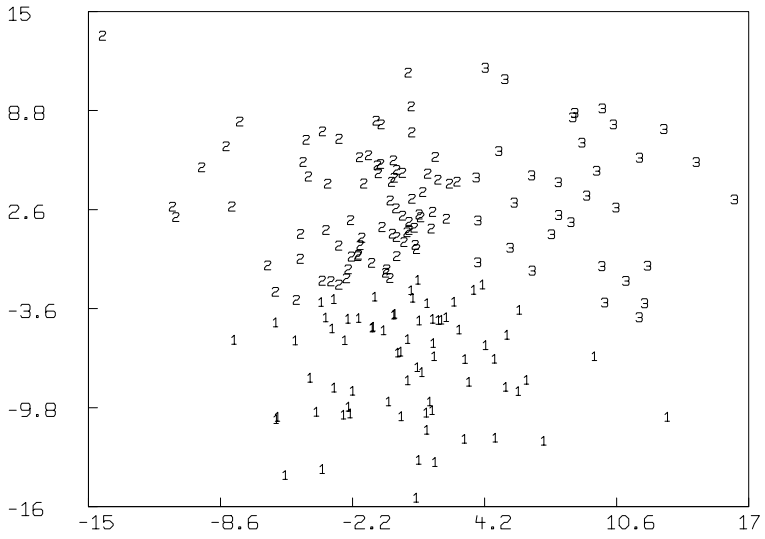
```
m = read(dataf, 500, 2);
draw m, pt crosspt lt none
view
```




```
k = kmeans(m,3)

 For 3 clusters: the initial error is 6157.127282
     after 1 iterations, the error is 4563.111736
     after 2 iterations, the error is 4364.649718
     after 3 iterations, the error is 4361.546152
     after 4 iterations, the error is 4361.546152
```

Now we will show the data points with each point labeled with its cluster number.

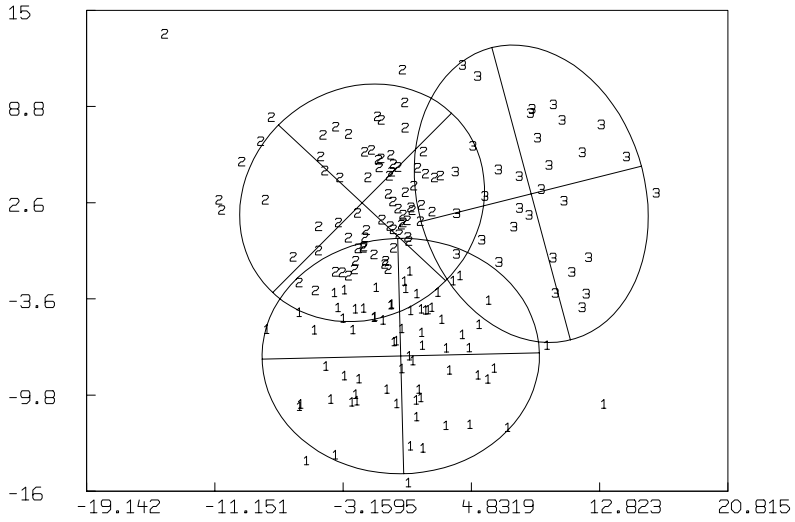
```
del w
draw m lt none pt none label k labelsize .01

view
```



We may draw the best-fitting bivariate normal ellipical contours of each cluster as follows. We will draw the ellipses which have .68 probability content. In general the ellipse with probability content p is $\{[x_1, x_2] \mid [x_1, x_2]V^{-1}[x_1, x_2]^T = CHISQI(p, 2)\}$.

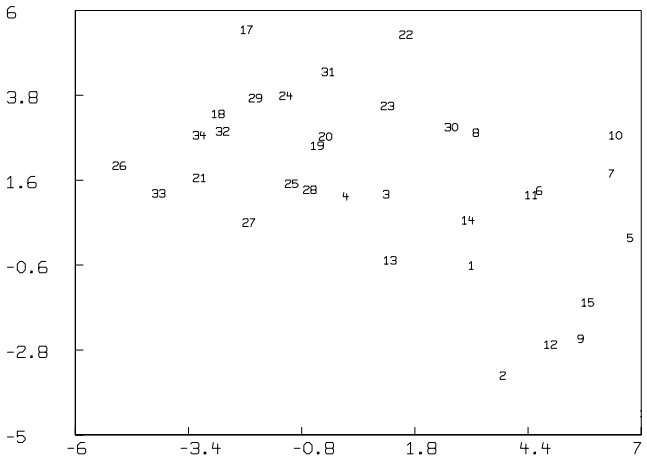
```
p = chisqi(0.68,2)
tv = 0:(2*pi)!80
fct x(t) = a*cos(t)*p
fct y(t) = b*sin(t)*p

for i = 1:3 do {
  q = compress((m&'(k=i)),3) col 1:2;
  mn = mean(q);
  c = cov(q);
  n = prcomp(c);
```

```
  a = sqrt(n[1,1]); b = sqrt(n[2,1]);

  z = (x on tv) &' (y on tv);
  e1 = n col 2:3 row 1;
  e2 = n col 2:3 row 2;
  r = e1 & e2;

  d1 = p*((-a*e1) & a*e1) + mn';
  d2 = p*((-b*e2) & b*e2) + mn';

  draw d1;
  draw d2;
  draw (z*r+mn');
}
window adjust wmatch
view
```

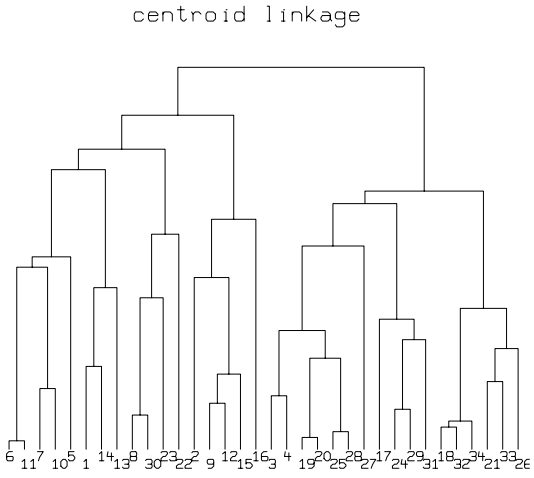


We can construct a so-called dendrogram in *MLAB* using various intercluster linking metrics. Below we will show the use of the simple centroid-based linking metric.

```
a = read(datac, 34, 2)
draw a lt none pt none label 1:34 labelsize .01
view
```




```
d = dists(a)
t = centroid(d)
n = dencurve(t)
draw n col 1:2 label n col 3:4 linetype marker labelsize .012 in w1
top title "centroid linkage" in w1
view
```



35 Non-Parametric Regression Modeling

If you have data for which a definite model, *i.e.* a set of mathematical functions, is known, then you can adjust the parameters in that model to fit the data using weighted least-square curve fitting; the resulting best-fit model can then be used for descriptive or predictive purposes.

If, on the other hand, a definite model is not known, as may occur for example with economic time series data or other phenomenological data, then some model which reasonably fits the data with any parametric form whatsoever may be useful; such a model is called a *non-parametric* model.

There are a variety of methods for obtaining a non-parametric model for given data. These methods include kernel estimation, polynomial fitting with moving-weights, smoothing splines, moving-means and moving-medians. Here we will concentrate on kernel estimation and moving-means; moving-weight polynomial fitting and smoothing splines have comparable performance properties, but they are far more mathematically-elaborate methods.

Let us focus on the 2-dimensional case where we have a collection of data points $(x_1, y_1), \ldots, (x_n, y_n)$ for which a model $f(x)$ is desired. In this context, kernel non-parametric regression estimation of f entails computing f as a weighted sum of the values $y_1, \ldots, y_n$ where the weights are normalized peaks defined by some kernel form given by a *kernel function* K with each peak centered at x_i so as to represent the contribution of the data point (x_i, y_i). Thus using kernel estimation defines f as

$$f(x) = \sum_{1 \le i \le n} y_i [K((x - x_i)/u) / \sum_{1 \le j \le n} K((x - x_j)/u)].$$

The kernel function K can be chosen in a variety of ways. One reasonable choice is as a truncated gaussian density function: $K(v) =$ if $|v| > 6$ then 0 else $g(v)$, where $g(v) = \frac{1}{2\pi} e^{-v^2}$. Actually, as you can see, the $1/(2\pi)$ factor is not necessary.

The parameter u is called the *kernel width*; it determines the spread of the kernel function. The larger u is, the more unimportant the differences in the individual y_i-values become. As u tends to ∞, $f(x)$ tends to a constant value independent of x. [What happens as $u \to 0$?] A reasonable choice for u is $[(\max_{1 \le i \le n} x_i - \min_{1 \le i \le n} x_i)/(2n/5)]^{1/2}$. A more elaborate approach might use dynamically-varying values of u depending upon the x-values.

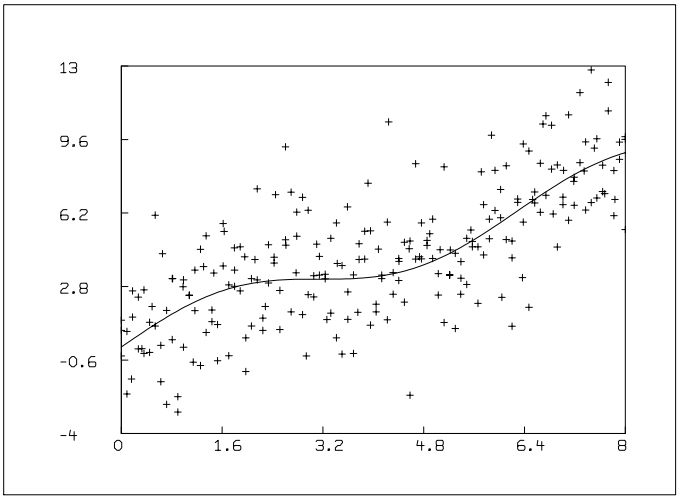
We may look at an example of non-parametric regression via kernel estimation using the mathematical and statistical modeling system *MLAB*. The *MLAB* computer program was originally developed at the National Institutes of Health and includes curve-fitting, differential equation-solving, statistics and graphics as some of its major capabilities. *MLAB* is a tool for researchers in science and engineering. *MLAB* is an ideal tool for solving simulation and parameter-estimation problems such as chemical kinetics, or neurophysiological models.

We'll actually use *MLAB* to generate our example data as well as to compute and display the kernel-based model function; this shows the ability of *MLAB* to do discrete simulation studies,

in addition to its central focus of doing continuous simulation via solving systems of differential equations. The text below shown in "typerwriter" font presents the *MLAB* commands used to compute and draw the exhibited results.

We will use the "trend" curve $sin(x) + x$ and add normal random noise to obtain our "data". We will arrange to generate some data points with duplicate x-values in order to subsequently demonstrate the handling of such data. The expression `a:b!k` denotes a column vector of `k` equally-spaced values starting with `a` and ending with `b`. The operator `&` denotes row-concatenation and the operator `&'` denotes column-concatenation. The expression `points(f,((0:8!90)r)&(0:8!50))` denotes the 230-row, 2-column matrix whose first column contains the values `0:8!90` repeated twice followed by the values `0:8!50` and whose second column contains the `f`-values corresponding to the x-values in the first column.

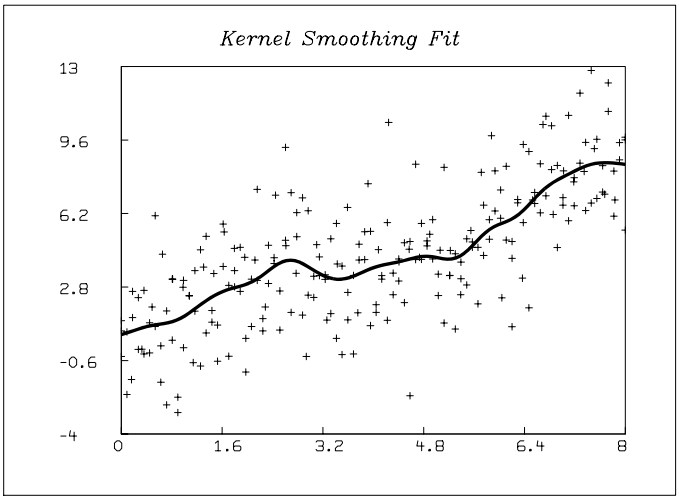
```
fct f(x)=g(x)+2*normran(0)
fct g(x)=sin(x)+x
m=sort(points(f,((0:8!90)^^2)&(0:8!50)))

draw m lt none pt crosspt ptsize .01
xv=0:8!120
draw trend = points(g,xv)
view
```




```
delete trend
```

Now we will compute the kernel-based non-parametric estimate for the data points in `m`. In order to be more efficient, we use embedded assignment operator (`_`) in the definition of the kernel estimator function sf. This allows us to accumulate the required denominator sum without repetitive computation.

```
n=nrows(m)
x= m col 1; y=m col 2;

fct sf(x)=(ks_0)+sum(j,1,n,[ks_ks+[kv_Kr((x-m[j,1])/u)]]*0+kv*m[j,2])/ks
fct Kr(x)=if abs(x)>6 then 0 else gaussd(x)

u = sqrt((maxv(x)-minv(x))/(2*n/5))
s=points(sf,xv)

draw kernel = s color orange lt (1,0,0,0,0,.005,0)
top title "Kernel Smoothing Fit" color red font 17
view
```




```
delete kernel
```

Note that the resulting curve is relatively smooth, but not as smooth as the underlying trend curve. The degree of "oscillation" is controlled by the kernel width parameter u. The greater u is, the smoother the kernel-estimator is and the less it tracks local features in the data.

Another common approach to non-parametric regression is to use a moving-average method. The basic idea is that the value at x_i should be the weighted average of the nearby data points. This has the effect of smoothing the variation in the data to yield less extreme and less oscillatory data points. Indeed all non-parametric regression schemes can be considered to be forms of data smoothing. Thus, for example, we may define our moving-average model as:

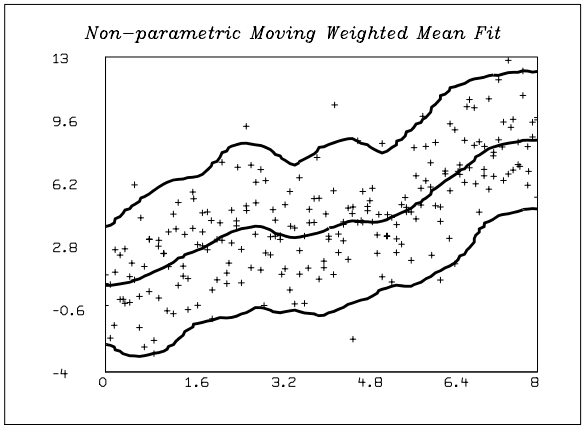
$$m(x_i) = \sum_{1 \le j \le k} w_j y_{i-1-\lfloor k/2 \rfloor + j}$$

Here k is the size of the moving data "window" and $w_1, \ldots, w_k$ are weights which sum to 1. Generally these weights should taper down to 0 at each end of the sequence; otherwise the moving-average model will be non-continuous. The moving-average method is a certain form of kernel estimation method which in its simplest form is restricted to estimating values only at the given ordinal locations $x_1, \ldots, x_n$. We can extend the moving-average method by using an interpolating function such as a cubic spline interpolating function of the points $(x_1, m(x_1)), \ldots, (x_n, m(x_n))$ produced by the moving-average computation.

MLAB contains a built-in moving-average operation together with various other weighted moving window computations. We can demonstrate the moving-average non-parametric model for the data exhibited above using *MLAB*.

Below we show the weighted moving-average non-parametric model curve and an associated 2σ standard-deviation error band. These curves are all computed using the corresponding builtin functions within *MLAB* in order to produce the non-parametric model curves and associated error bands for the data that we generated above. Note that the data has multiple points with the same x-values, which is easily handled by *MLAB*. In this case we will use a moving window of 45 points which are weighted with the weight-values specified in the vector `wm`.

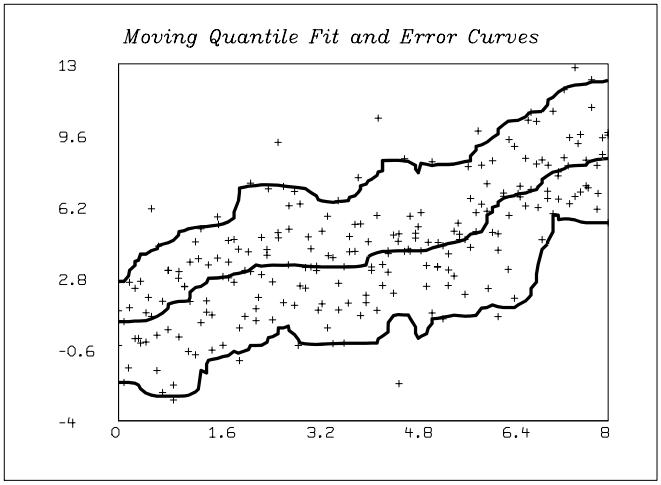
```
* wm=(0:1!15)&(1^^15)&(1:0!15)
* a=mmean(m col 2,45,0,4,wm)
* s=mstddev(m col 2,45,0,4,wm)

* draw m lt none pt crosspt ptsize .01
* za=(m col 1)&'a&'s; za=rdup(za); tx = za col 1;
* draw za col (1,2) color red lt (1,0,0,0,0,.005,0)
* draw tx &'((za col 2)+1.96*(za col 3)) color brown lt (1,0,0,0,0,.005,0)
* draw tx &'((za col 2)-1.96*(za col 3)) color brown lt (1,0,0,0,0,.005,0)
* top title "Non-parametric Moving Weighted Mean Fit" font 17
* view
```



Again the degree of "oscillation" is governed by the size of the moving window and the weights that are used.

When the data has large variations or outliers, a more robust method of estimating a central trend function is to use a moving-median non-parametric method. An example is shown below using the built-in weighted moving-quantile function `mquantile` found in *MLAB*.

```
* a=mquantile(m col 2,.5,45,0,4,wm)
* q1=mquantile(m col 2,.05,45,0,4,wm)
* q2=mquantile(m col 2,.95,45,0,4,wm)

* draw m lt none pt crosspt ptsize .01
* za=(m col 1)&'a&'q1&'q2; za=rdup(za);
* draw za col (1,2) color red lt (1,0,0,0,0,.005,0)
* draw za col (1,3) color brown lt (1,0,0,0,0,.005,0)
* draw za col (1,4) color brown lt (1,0,0,0,0,.005,0)
* top title "Moving Quantile Fit and Error Curves" font 17
* view
```



In this situation the estimated curve is noticablly "rougher", but less influenced by outlier points. One sometimes useful-strategum is to use .05 and .95 quantile curves to determine a data band region, discard those points outside the band, and then fit the remainning points with a moving mean curve.

36 A Missing Data Imputation Procedure

The general problem of handling missing data in the presence of observed covariates arises in many situations. For example, data on the incubation period of the HIV virus in *AIDS* patients are often censored. One procedure for handling this missing data has been proposed recently by *Gang Chen* and *Grace Yang* ["A conditional bootstrap procedure for reconstruction of incubation period of aids", *Mathematical Biosciences* Vol, 117 p253-269, Aug, 1993]. An example of this procedure is given here using the mathematical and statistical modeling software package *MLAB*.

Suppose we have $m + n$ HIV-positive individuals. For n of these individuals, we know both the amount of time that they have incubated the $AIDS$ virus and their current sero-index value ($T4$-cell count). Thus we have the data $(y_1, t_1), \ldots, (y_n, t_n)$ where t_i is the incubation time and y_i is the sero-index value for the ith individual. For the remaining m individuals, we know their current sero-index values $z_1, \ldots, z_m$, but we do not know the associated incubation times $s_1, \ldots, s_m$; these times are left-censored.

Our goal is to estimate the missing data values $s_1, \ldots, s_m$ associated with the covariate values $z_1, \ldots, z_m$. The procedure we use is based on resampling and has utility in a wide range of circumstances.

Let $t_1, \ldots, t_n$ and $s_1, \ldots, s_m$ be samples of the random variables $T_1, \ldots, T_n$ and $S_1, \ldots, S_m$. Let $y_1, \ldots, y_n$ and $z_1, \ldots, z_m$ be samples of the random variables $Y_1, \ldots, Y_n$ and $Z_1, \ldots, Z_m$. Note that, generally, T_i and Y_i are correlated, as are S_j and Z_j, in some unknown manner.

We want to estimate values $s_1, \ldots, s_m$ so that s_i will be a plausible and useful sample of S_i. (The notions of plausible and useful are somewhat dependent upon how the data is to be used.)

In this example, we will always choose s_i from $\{t_1, \ldots, t_n\}$, so no interpolation is involved. Also note we could use partial information on s_i, such as $s_i > h_i$ by merely taking $s_i = h_i$ whenever the generated value turns-out to be less than h_i.

The basic idea is: for any given z_i, we want to choose the corresponding value s_i from $\{t_1, \ldots, t_n\}$ according to the probability $P(S_i = t_j \mid z_i, (y, t))$. We will use the following formula for this conditional probability:

$$P(S_i = t_j \mid z_i, (y, t)) = \frac{K((z_i - y_j)/a_j)}{\sum_{k=1}^{n} K((z_i - y_k)/a_k)}$$

where K is a suitable kernel function and $a_1, a_2, \ldots, a_n$ are kernel-width parameters. Often K is the tent function, *i.e.* $K(x) =$ if $|x| > 1$ then 0 else $1 - abs(x)$. Another useful choice is the *Epanechnikov* kernel function $K(x) = 0.75 \cdot \max(1 - x^2, 0)$

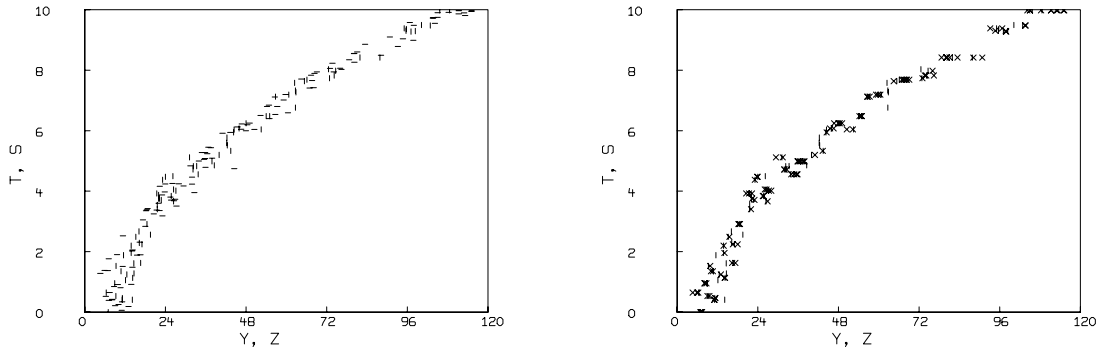
Note that $P(S_i = t_j \mid z_i, (y, t))$ only depends on the values $t_1, \ldots, t_n$ through the index j.

Let $p_{ij} := P(S_i = t_j \mid z_i, (y,t))$, and let $q_{ij} := \sum_{k=1}^{j} p_{ik}$, We can partition the unit interval $[0,1]$ into the intervals $[0, q_{i1}), [q_{i1}, q_{i2}), \ldots, [q_{i,n-1}, q_{in}]$ where $q_{in} = 1$. We now generate a uniform random number v in $(0,1)$ and see which subinterval, $[q_{i,j-1}, q_{ij})$, v falls into, and then choose s_i to be t_j.

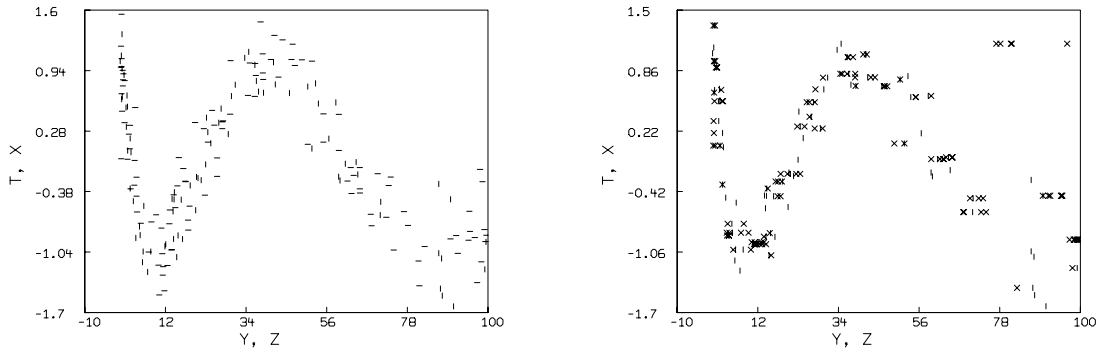
These choices for $s_1, \ldots, s_m$ "repair" the original data and thus solve our problem. Here is an example of this procedure given as an *MLAB do-file*. Note we use varying kernel-widths which are functions of the spacing of the y-observations.

```
/* read-in the Y, T, Z and S1 observed values */
y = read(yfile);
t = read(tfile);
z = read(zfile);
s1 = read(sfile); /* we will see how well we predict these values */
n = nrows(y);
m = nrows(z);

/* compute the varying kernel-width vector av */
v = sort(y &' t); t = v col 2; y = v col 1;
yd = y - rotate(y,1); yd[1] = yd[2];
av = 2*mmean(yd,floor(n/10)) *' mstddev(t,floor(n/8))

fct maxf(a,b) = if (a > b) then a else b
fct k(x) = if abs(x) > 1 then 0 else 1 - abs(x)

/* now, impute s[1:m] */
for i = m:1 do {\
        zv = z[i]; \
        mav = maxf on (av &' minv(abs on (zv-y))); \
        p = k on (zv-y)/'mav; /* generate conditional probabilities p */ \
        p = psum(p) &' (1:n); r = ran(0,0,p[n]); \
        s[i] = t[ceiling(lookup(p,r))]; /*select s[i] according to p*/ \
}
/* now s[1:m] holds the imputed values associated with z[1:m] */

draw y &' t lt none pt hbar ptsize .002 color green; /* fully specified data */
draw z &' s1 lt none pt vbar ptsize .01 color red; /* missing data */
view
delete w
draw y &' t lt none pt hbar ptsize .01 color green; /* fully specified data */
draw z &' s lt none pt xpt ptsize .01 color blue; /* imputed data */
view
```

Here are some examples showing the application of this procedure for imputing missing data.

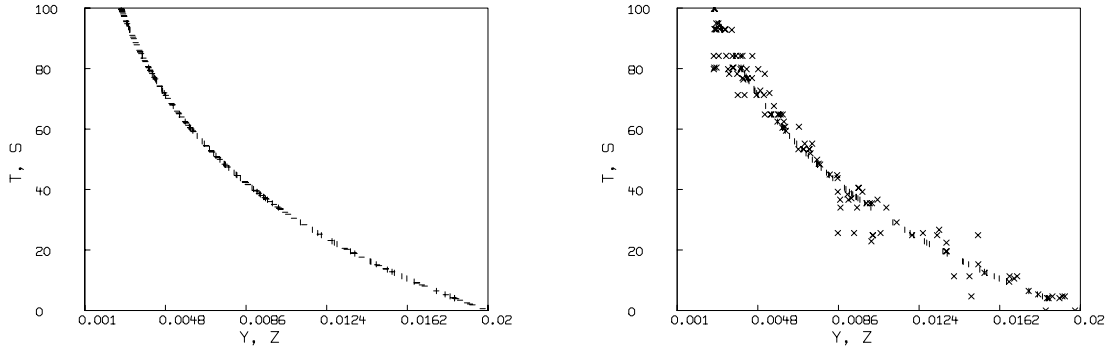


Model function $f(x) = sqrt(x - POISRAN(0,10))$. Left: vbar for original data, hbar for missing data. Right: vbar for original data, cross for imputed data.



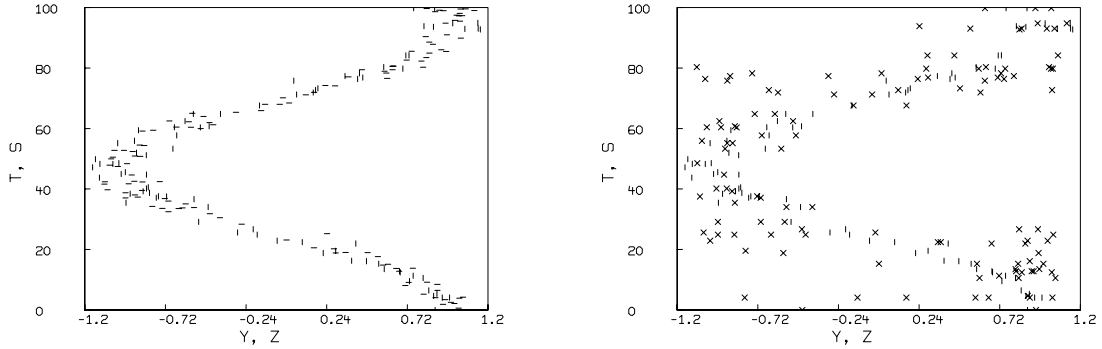
Model function $x(t) = t^2 + 0.1 \cdot NORMRAN(0), y(t) = cos\ \ (t) + 0.3 \cdot NORMRAN(0)$ Left: vbar for original data, hbar for missing data. Right: vbar for original data, cross for imputed data.

Here is an example of data without error.

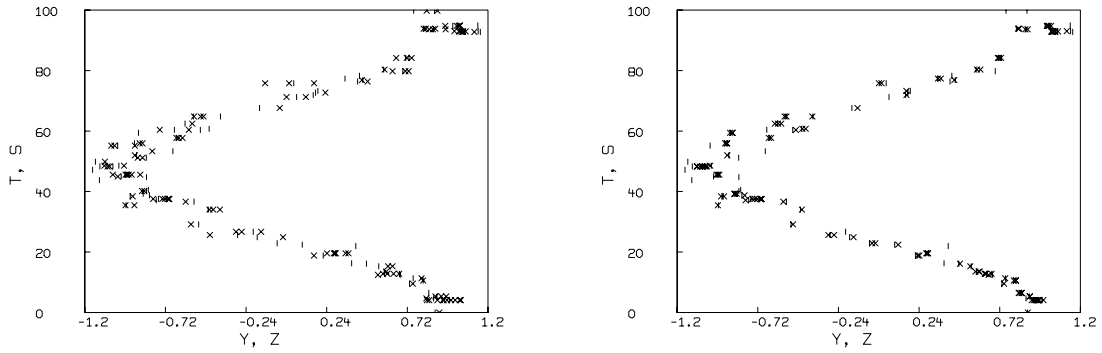


Model function $f(x) = -(\log x - \log b)/b$ *where* $b = 0.02$. *Left: vbar for original data, hbar for missing data. Right: vbar for original data, cross for imputed data.*

Here is an example based on a non-single-valued function where the conditional distribution of $S_i \mid \{z_i, (y,t)\}$ becomes increasingly bi-modal. In this case the kernel-width estimation method used above fails. The second pair of the following pictures show some results for modified choices of kernel-widths.



Model function $f(x) = 15 \cdot arccos(x - 0.1 \cdot NORMRAN(0))$. *Left: vbar for original data, hbar for missing data. Right: vbar for original data, cross for imputed data. Kernel-widths depend on moving standard deviations of the data.*



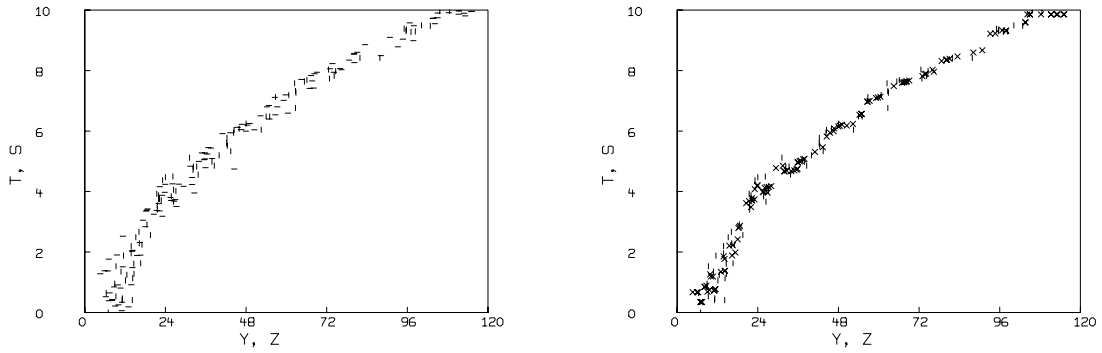
Model function $f(x) = 15 \cdot arccos(x - 0.1 \cdot NORMRAN(0))$. *Kernel-widths do not depend on the standard deviation of the data. Left: kernel-widths = 5 times moving means. Right: kernel-widths = moving means.*

Note that we can produce imputed data values for missing data in an indexed time-series by taking y and z to be the integers $1, 2, \ldots, m+n$, where z specifies those integer times at which data was not recorded.

The noise in the imputed data, viewed as a stochastic process, is dependent upon the widths used in the kernel function K and upon the known values $\{t_1, \ldots, t_n\}$ from which the imputed values are drawn. Often we would like the noise in the imputed data to be similar to the noise seen in the complete observed data. In the limiting case, where the data points are drawn without error from the graph of a smooth single-valued function, imputed values should be obtained by means of some suitable interpolation scheme, IS, such that $IS(z_i)$ produces a value for S_i by interpolating with a smooth function specified by the complete data points $(y_1, t_1), \ldots, (y_n, t_n)$. In general, we can define a smoothing interpolation function specified by the complete data points as the result of some weighted-average procedure. One such choice is to compute $IS(z_i) = E(S_i \mid z_i, (y, t))$, based on our kernel estimate of the conditional distribution for S_i.

In general, we could then choose an imputed data value for S_i as $\alpha C(z_i) + (1 - \alpha)\ IS(z_i)$ where $C(z_i)$ denotes the *Chen-Yang* procedure applied to select an imputed value associated with the covariate value z_i. The mixing parameter α can be chosen as a function of the noise percieved in the complete observed data; when no noise is present, $\alpha = 0$, and when no trend is apparent, $\alpha = 1$.

An example of this mixing computation for the first data set given above is shown below where $\alpha = .5$.



Model function $f(x) = -(log\ x - log\ b)/b$ *where* $b = 0.02$. *Left: vbar for original data, hbar for missing data. Right: vbar for original data, cross for imputed data.*

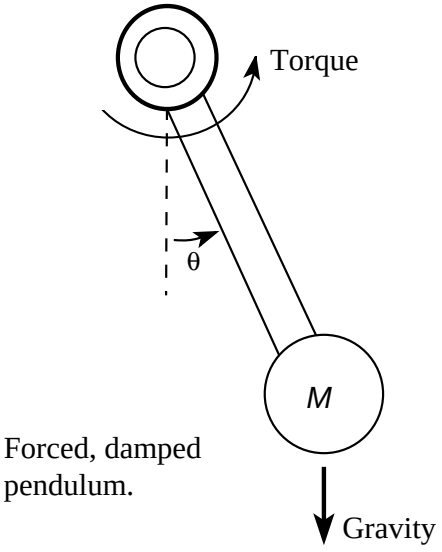
[1] *A conditional bootstrap procedure for reconstruction of incubation period of aids. Mathematical Biosciences* V117 p253-269

37 A Numerical study of the Forced Damped Pendulum

An ideal pendulum (*i.e.* with no friction) will swing back and forth (or loop in a full circle) forever if there is no outside force other than gravity acting upon it. Moreover, a pendulum with friction will come to rest if there is no other outside force besides gravity acting upon it. A more general forced damped pendulum with a periodic driving force pushing it shows more interesting asymptotic behavior than these two trivial cases. The angular position in radians as a function of time $\theta(t)$ of a forced damped pendulum is described by the following second order differential equation.

$$\frac{d^2\theta}{dt^2} + \nu\frac{d\theta}{dt} + sin\theta = \rho sin(2\pi ft), \;\; \theta(0) = \theta_0, \;\; \frac{d\theta}{dt}(0) = s, \tag{0.7}$$

where $\frac{d^2\theta}{dt^2}$ represents the inertia, $\nu\frac{d\theta}{dt}$ represents friction at the pivot, $sin(\theta)$ represents gravity, and $\rho sin(2\pi ft)$ represents a sinusoidal frequency f driving torque applied at the pivot. θ_0 is the initial angular position and s is the initial angular velocity of the pendulum.
Numerical solutions show that both chaotic and periodic solutions of the forced damped pendulum equation are possible depending on the particular choice of system parameters ν, ρ and f.

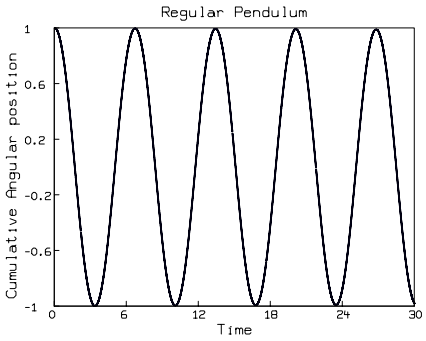


If we want to see how the pendulum is actually moving, we can solve the differential equation and plot the angle variable θ against time t. We can use the mathematical modeling system *MLAB* to solve this differential equation and display the solution. The following is an *MLAB* dofile that solves the forced damped pendulum equation.

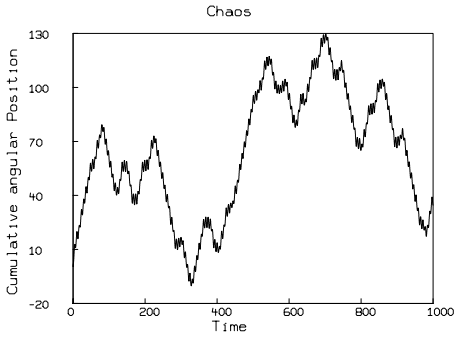
```
"pendulum.do — Solve the ODE for a forced pendulum"

"the differential equation for the forced damped pendulum"
fct theta"t(t) = rho*sin(t) - c1*theta't - sin(theta)
"rho*sin(t) is the driving force, c1*theta't is friction, sin(theta) is gravity"
```

```
"Read-in parameter values"
type "input the friction coefficient c1"; c1 = kread();
type "input the driven force amplitude" rho = kread();
type "input n: number of units of time to solve for"; n = kread();

"Read-in the initial conditions"
type "Specify the initial conditions"
type "input theta(0)"; theta0 = kread(); initial theta(0) = theta0
type "input theta'(0)"; thetap0 = kread(); initial theta't(0) = thetap0

"Specify the time values where we want the solution"
data = 0:n:.1; "get an array from 0 to n at every .05"

/* Now, integrate the differential equation. This is done implicitly
inside the operator 'points', the result matrix is put in the array m.
The first column of m is the data array generated above, the second
column is the theta value at the time corresponding to the time value in
the first column, the third column is the theta't value at the time
corresponding to the time value in the first column. */
m = points(theta, theta't,data)

"draw the trajectory (angular position vs. time)"
draw m col 1;2, color red
top title "Regular"
left title "Cumulative Angular Position";
bottom title "Time"
view; "view the picture"
```

The following pictures are generated by this *MLAB* dofile. (with modified graphics). They are plots of time against the angular position. The left picture is the trivial regular case with a periodic orbit where $c1 = 0$ and $rho = 0$. The right picture shows a chaotic orbit where $c1 = 0.3$ and $rho = 2.5$. In all cases, the initial point corresponds to the initial conditions denoted by *theta0* and *theta1* in the dofile.



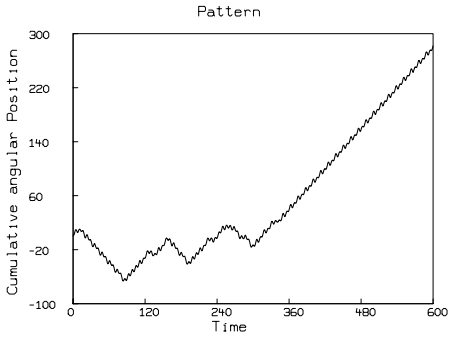
Pendulum with no friction, no exterior force. c1 = 0, rho = 0. initial point (1.0,0)



Chaotic orbit, no regular pattern observed, c1 = 0.3, rho = 2.5. initial point (0.5, 2.0)

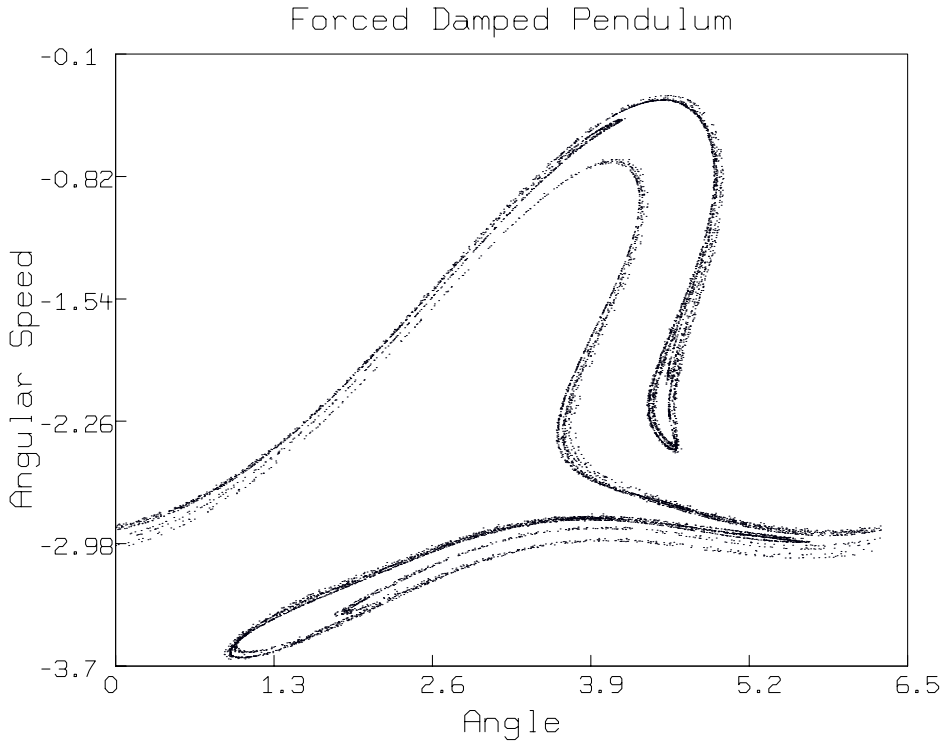
Here are another pair of pictures that are generated by the same dofile. The left picture shows a transient chaotic behavior which settles down to a periodic orbit after awhile. The parameter values are $c1 = 0.1$, $rho = 1.5$. The right picture shows another kind of behavior. It is not periodic, since the angle keeps increasing, it is not chaotic either since the the angle is increasing in a regular pattern, thus the Lyapunov exponent is not positive. The parameter values are $c1 = 0.2$, $rho = 2.5$.

Transient, started chaotic and settles down to periodic. c1 = 0.1, rho = 1.5, initial point (0.5, 2.0)



Not periodic, not chaotic, angle keeps increasing. c1 = 0.2, rho = 2.5, initial point (0.5, 0.9)

To see a trajectory of a point in the phase-space $(t, \theta(t), \theta'(t))$ for the forced damped pendulum, we will construct an object called the *Poincaré* map which is often used to reduce a continuous time system (or "flow") to a discrete time map with one less dimension. The basic idea of a *Poincaré* map is to choose some appropriate hypersurface in the phase space and observe the intersection of the orbit in the phase-space with the surface. Since the solution of the system is unique with a given initial point, when we neglect numerical round-off error, each intersection point will uniquely determine the successive point. Thus, a continuous flow is reduced to a discrete map with one less dimension.



Poincaré return map for the pendulum, the section plane is $t = 0$, $c1 = 0.3$, $rho = 2.5$. $n = 10000$. initial point (0.5, 2.0)

The picture above is the *Poincaré* map of the trajectory of the point $(0.5, 2.0)$ in the $(\theta(t), \theta'(t))$ reduced-dimension phase space for our forced damped pendulum system with the parameter $rho = 2.5$ and $c1 = 0.3$ which is the same as the chaos picture above. We used $n = 10000$. It is the *Poincaré* return map with the section surface $t = 0$, which is equivalent to $t = 2k\pi$ for any integer k since the variable t only appears in $sin(t)$, and for any integer value of k, $sin(2k\pi)$ will have the same value 0.

To compute the above *Poincaré* map, we used almost the same *MLAB dofile* with only a few minor changes. We only want the solution at every time when $t = 2k\pi$ (a multiple of 2π), Thus, we need to substitute the line for computing the time-list data in the above *MLAB* dofile with the following line:

```
data = 0:(pi2*n):pi2; "get an array from 0 to pi2*n at every pi2"
```

Also, we need to plot different columns of the solution matrix m, thus, we must substitute the draw statement lines with the following lines

```
/* The variable theta is the angle of the arm of the pendulum. The angle
```

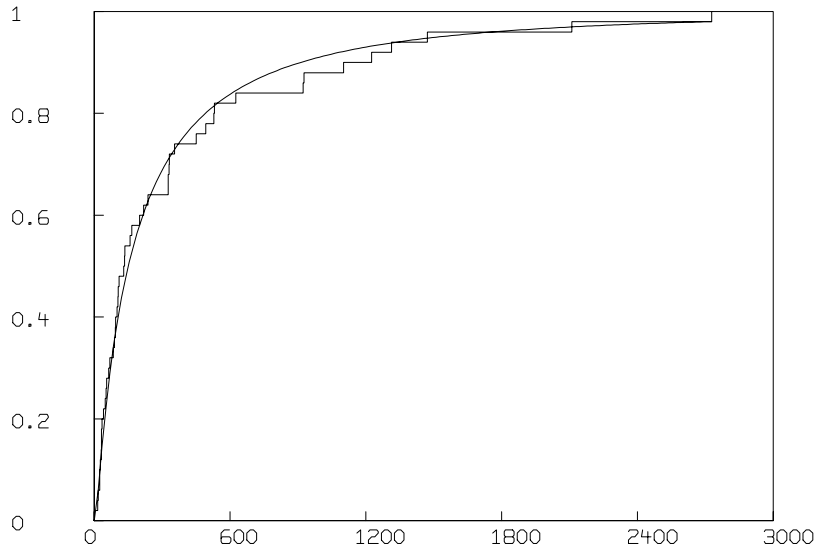
```
can range from 0 to pi2, any angle that is outside this region has a
corresponding angle in this region. We define a function to map all
the angles into 0:pi2 */
fct ft(theta) = mod(theta, pi2)
m col 2 = ft on (m col 2); "map col 2 into the region 0 to pi2"
"draw a dot at each phase plane at each point (theta,theta't)"
draw m col 2:3, color red, lt none, pt dotpt
top title "Forced Damped Pendulum"
left title "Angular Velocity"; bottom title "Angle"
view; "view the picture"
```

One can easily generate further pictures with other parameter values.

38 Maximum Likelihood Estimation

Given n points that are assumed to be samples of a lognormal distribution, we wish to estimate the parameters of the distribution via maximizing the likelihood of our sample, and then we wish to test whether the data is plausibly drawn from this best-fitting distribution.

```
x = read(dfile)
n = nrows(x)
fct ll(y,z) = sum(i,1,n,log(lnormd(x[i],y,z)))
y = 0; z = 1;
constraints q = {y > -5, y < 10, z > .5}
maximize(ll,y,z,q)

The function value is: -3.385912e+02
Argument(s): (5.011714e+00 1.978510e+00 )
Gradient: (-6.153095e-08 1.545697e-07 )
ecode = 0, Converged.
# of function evaluations: 33
# of gradient evaluations: 40
# of Quasi-Newton iterations: 18
There are no active constraints.
    = -338.591243

fct lf(x) = lnormf(x,y,z)
draw stepgraph(cdf(x))
draw points(lf,0:maxv(x)!160)
view
```




```
ks1t(x,lf) /* test if x is data from lf */

[K-S-test: are the samples in M plausibly drawn from the
distribution F?]
null hypothesis H0: M is drawn from the distribution F.
The scaled maximum deviation between cdf(M) and F is
distributed as the Kolmogorov-Smirnov K statistic.
The K-statistic value = 0.494806 at the point 135.087720

The probability P(K > 0.494806) = 0.967170
This means that a value of K larger than 0.494806 arises
about 96.717001 percent of the time, given H0.

    : a  5 by 1 matrix

  1: .494806371
  2: .494806371
  3: .489304956
  4: .967170007
  5: 135.08772
```

39 Charged Particles on a Disk

A classical electrostatics problem which drew some attention in the Scientific Correspondence column of *Nature* in 1985 (see *Nature*, `315` (1985) 104, 635; `316` (1985) 302; `317` (1985) 208) offers an interesting problem for the MLAB function minimization operator, `MINIMIZE`. The problem is: what will be the minimum potential energy configuration of a system of N equal point charges placed on a disk? (In what follows, the magnitude of a charge, the radius of the disk, and 4π times the capacitivity of the vacuum, ϵ, will be arbitrarily set equal to one.)

This paper will demonstrate how to use the `MINIMIZE` operator, differential equation solver, and graphics capabilities in the MLAB mathematical modeling system to find and visualize solutions to this electrostatics problem for some specific values of the number of charged particles. MLAB is a computer program whose name is an acronym for "Modeling LABoratory". It is an interactive interpreter with sophisticated instructions for performing function minimization, solving ordinary differential equations, curve fitting, and generating publication quality graphs. It was originally developed at the National Institutes of Health. It is available from Civilized Software for a variety of computer platforms including DOS- or Windows-based IBM-compatible personal computers, Macintoshes, and SUN, NeXT, IBM RS6000, DEC-Alpha and SGI Unix workstations.

Minimization with Explicit Constraints

To begin using MLAB, type `MLAB` at the operating system prompt of DOS or Unix systems, or double click on the MLAB icon of operating systems with a graphical user interface. MLAB then starts by typing some initialization information on the screen and an asterisk, `*`. The asterisk is the prompt for the user to type MLAB instructions. In what follows, MLAB instructions and responses will be typed in `BOLD FACE`.

The MLAB operator `MINIMIZE` finds a local minimum of a user defined function of several variables, optionally subject to linear constraints, given initial values of the variables. By default, a minimum or saddle point is defined as a point in parameter space where the magnitude of the components of the gradient of the objective function are less than 10^{-6}. We can use `MINIMIZE` to find the minimum potential energy configuration of a system of 2 equal point charges on a disk by expressing the potential energy of 2 charges in terms of the polar coordinates of each charge, defining a set of constraints that limit the polar coordinates to the unit disk, and calling the `MINIMIZE` operator. This simple problem will provide a basis for more complicated examples that arise later.

First, define the distance between two points on a plane given the polar coordinates of each point. If r_i and r_j are the distances of the i^{th} and j^{th} particles from the origin, and t_i and t_j are the corresponding angles measured in radians in a counter-clockwise direction from the positive x-axis, the distance between the 2 points is:

$$d(r_i, t_i, r_j, t_j) = \sqrt{(r_i \cos(t_i) - r_j \cos(t_j))^2 + (r_i \sin(t_i) - r_j \sin(t_j))^2}$$

This function can be expressed in MLAB with the `FUNCTION` statement (abbreviated `FCT`),

```
FCT D(RI,TI,RJ,TJ)=SQRT((RI*COS(TI)-RJ*COS(TJ))^2+(RI*SIN(TI)-RJ*SIN(TJ))^2)
```

For 2 charges, the potential energy of the 2 charge system is proportional to the reciprocal of this interparticle distance. We define a function for the potential energy of two particles in MLAB by typing,

```
FCT V2() = 1/D(R1,T1,R2,T2)
```

Although `V2` has no arguments, it is an implicit function of the radius and angle parameters `R1`, `R2`, `T1`, and `T2`.

Next, define a set of constraints for the radii and angles. `R1` and `R2` must be less than or equal to 1 but greater than or equal to -1 in order for the charges to be confined to the unit disk. Since the potential energy of a configuration is the same if it is rotated by an arbitrary angle, `T1` is fixed to zero. These constraints are defined with the MLAB command,

```
CONSTRAINTS Q2 = {R1<=1,R1>=-1,R2<=1,R2>=-1,T1=0}
```

To find the minimum potential energy configuration, we make initial estimates for the parameters `R1`,`T1`,`R2`, and `T2` consistent with the constraints, and then call the minimization function.

```
R1 = .5; T1 = 0; R2 = .25; T2 = .3;
MINIMIZE(V2,R1,T1,R2,T2,Q2)
```

In this use of the `MINIMIZE` operator, the first argument is the name of the function to be minimized, the next 4 arguments are the names of the parameters to be varied during the minimization, and the last argument is the name of the set of linear constraints.

MLAB responds to the last command with

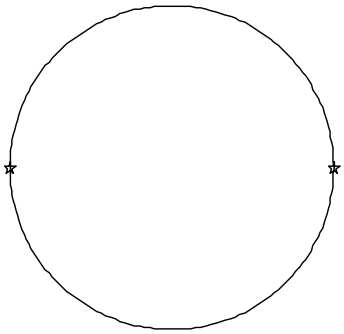
```
The function value is: 5.000000e-01
Argument(s): (1.000000e+00 0.000000e+00 -1.000000e+00 4.855033e-09 )
Gradient: (-2.500000e-01 -6.068791e-10 2.500000e-01 6.068791e-10 )
ecode = 0, Converged
# of function evaluations: 14
# of gradient evaluations: 28
# of Quasi-Newton iterations: 13
The following constraints are active: 5 1 4
    = 0.5
```

The minimum value of `V2` was thus determined to be .5. In this calculation MLAB determined the symbolic gradient of the objective function and applied a Quasi-Newton method to obtain the minimum. Upon completion of the minimization, `MINIMIZE` reports the value of the function at the minimum, the arguments at the minimum, and the components of the gradient at the minimum. It also reports the number of times the objective function was evaluated, the number of times the gradient of the function was evaluated, and the constraints which are active at the stopping point. The value reported for `ecode` indicates the exit condition; 0 signifies that a minimum satisfying the exit criterion has been found.

We can visualize the results graphically by determining 200 points on a unit circle, drawing straight-lines connecting those points, and placing stars at the positions of the charges found by the minimizer. This is done as follows:

```
TL = 0:(2*PI)!200
C = (COS ON TL) &' (SIN ON TL)
NO IMAGEBOX IN W
DRAW C
WINDOW ADJUST WMATCH
DRAW (R1*COS(T1)) &' (R1*SIN(T1)) PT STAR
DRAW (R2*COS(T2)) &' (R2*SIN(T2)) PT STAR
VIEW
```

The first statement generates a row vector called `TL` which contains 200 equally space numbers between 0 and 2π, inclusive. The second statement generates a 2 column matrix named `C` which holds the x and y coordinates of the 200 points forming a circle. The third statement creates a window structure called `W` for drawing. The fourth statement draws the points of the circle in `W`. The fifth statement ensures that the aspect ratio of the window is 1 so that the curve appears circular and not elliptical. The remaining `DRAW` statements place stars at the positions of the charges found by the minimization process. The `VIEW` command then makes the following figure appear on the display.



We find, not surprisingly, that the charges occupy opposite ends of a diameter. This solution will be found for any initial estimates of the parameters R1,T1,R2, and T2 which are consistent with the constraints.

To clear the graph from the screen, type:

```
UNVIEW
DELETE W
```

Next we find the minimum potential energy configuration for 3 particles. The function to be minimized, expressed in terms of radius and angle variables R1,T1,R2,T2,R3, and T3, is defined in MLAB by the command

```
FCT V3() = 1/D(R1,T1,R2,T2)+1/D(R1,T1,R3,T3)+1/D(R2,T2,R3,T3)
```

The constraints limiting the 3 charges to the unit disk without an arbitrary rotation are defined by typing

```
CONSTRAINTS Q3 = {R1<=1,R1>=-1,R2<=1,R2>=-1,R3<=1,R3>=-1,T1=0}
```

An initial guess is made and the minimization is performed by typing:

```
R1 = 1; T1 = 0; R2 = 0; T2 = .5; R3 = 1.; T3 = PI;
MINIMIZE(V3,R1,T1,R2,T2,R3,T3,Q3)
```

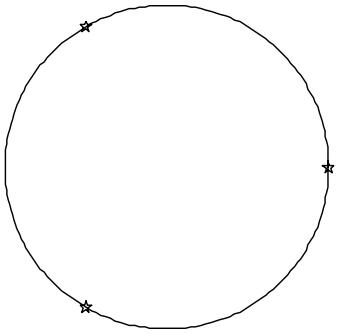
To this, MLAB responds,

```
The function value is: 1.732051e+00
Argument(s): (1.000000e+00 0.000000e+00 1.000000e+00 2.094396e+00
              1.000000e+00 4.188790e+00 )
Gradient: (-5.773502e-01 -8.001080e-08 -5.773503e-01 2.971898e-07
           -5.773503e-01 -2.171790e-07 )
ecode = 0, Converged.
# of function evaluations: 15
# of gradient evaluations: 34
# of Quasi-Newton iterations: 13
The following constraints are active: 7 1 5 3
    = 1.73205081
```

The final configuration can be visualized by typing the following commands:

```
NO IMAGEBOX IN W
DRAW C
WINDOW ADJUST WMATCH
DRAW (R1*COS(T1)) &' (R1*SIN(T1)) PT STAR
DRAW (R2*COS(T2)) &' (R2*SIN(T2)) PT STAR
DRAW (R3*COS(T3)) &' (R3*SIN(T3)) PT STAR
VIEW
```

The resulting figure reveals that the minimum potential energy configuration consists of the 3 charges at the vertices of an equilateral triangle.



The graph is cleared from the screen by typing

```
UNVIEW
DELETE W
```

The 4 particle problem is solved by typing the following statements in MLAB:

```
FCT V4() = 1/D(R1,T1,R2,T2)+1/D(R1,T1,R3,T3)+1/D(R1,T1,R4,T4)+\
           1/D(R2,T2,R3,T3)+1/D(R2,T2,R4,T4)+1/D(R3,T3,R4,T4)
CONSTRAINTS Q4 = {R1<=1,R1>=-1,R2<=1,R2>=-1,R3<=1,R3>=-1,\
           R4<=1,R4>=-1,T1=0}
R1 = 1; T1 = 0; R2 = .75; T2 = .5; R3 = .5; T3 = PI; R4 = 0; T4 = 2*PI/3;
MINIMIZE(V4,R1,T1,R2,T2,R3,T3,R4,T4,Q4)
```

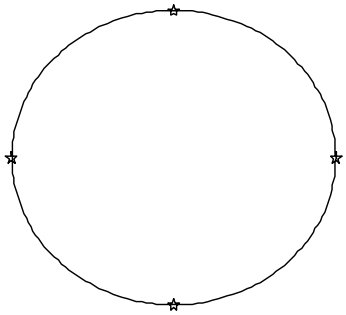
Note that the backslash character is typed to continue an MLAB command onto a second line. To this, MLAB responds

```
The function value is: 3.828427e+00
Argument(s): (1.000000e+00 0.000000e+00 1.000000e+00 3.141590e+00
              1.000000e+00 4.712386e+00 -1.000000e+00 1.099557e+01 )
Gradient: (-9.571079e-01 6.660381e-06 -9.571057e-01 2.899465e-06
           -9.571063e-01 -7.277632e-07 9.571073e-01 -8.832083e-06 )
ecode = 0, Converged.
# of function evaluations: 20
# of gradient evaluations: 56
# of Quasi-Newton iterations: 16
The following constraints are active: 9 1 3 5 8
    = 3.82842712
```

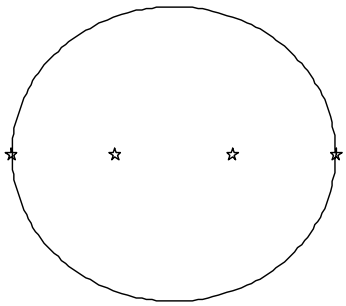
Issuing the drawing commands,

```
NO IMAGEBOX IN W
DRAW C
WINDOW ADJUST WMATCH
DRAW (R1*COS(T1)) &' (R1*SIN(T1)) PT STAR
DRAW (R2*COS(T2)) &' (R2*SIN(T2)) PT STAR
DRAW (R3*COS(T3)) &' (R3*SIN(T3)) PT STAR
DRAW (R4*COS(T4)) &' (R4*SIN(T4)) PT STAR
VIEW
```

shows the final configuration on the unit circle with the point charges occupying the vertices of a square.



The 4 particle problem demonstrates a difficulty that can arise in any greater-than-3 particle problem: the final configuration found by `MINIMIZE` may be a *local* minimum, or (rarely) a saddle point, not a *global* minimum; often this happens when we start with a local minimum or saddle point initially. For example, if the initial configuration for the 4 particle problem consists of all 4 particles equally spaced along one diameter of the disk, minimization of `V4` using `MINIMIZE` leads to the following configuration with a potential energy of 6.484435:



Comparison of the potential energies for the two runs, 3.828427 to 6.484435, reveals that the configuration found in the former run is the global minimum.

The Penalty Function Method

There are N*(N-1)/2 terms in the expression for the potential energy of N equally charged particles and 2*N+1 constraints. For large N, the potential energy function and the constraint set can become cumbersome to enumerate. To simplify matters, the so-called *penalty function* method can be employed. This method consists of adding a term to the general expression for the potential

energy of N point charges which causes the potential energy function to evaluate to a very large value in a continuous manner when the norm of any one of the radii is greater than 1.

The general expression for the potential energy of N point charges with 1 charge fixed at the intersection of the x-axis and the unit circle is

$$v_N(r,t) = \sum_{i=1}^{N-1} \left(\frac{1}{d(r_i, t_i, 1, 0)} + \sum_{j=i+1}^{N-1} \frac{1}{d(r_i, t_i, r_j, t_j)} \right)$$

where r is an array of N-1 numbers corresponding to the radii and t is an array of N-1 numbers corresponding to the angles. To this expression we add a term called a penalty term, which grows exponentially if any particle's radius should grow larger than 1. An example of the potential energy function with a penalty term which effectively confines the radii to the unit disk can be expressed in MLAB by

```
FCT VN(R,T) = SUM(I,1,N-1,1/D(R[I],T[I],1,0)+\
              SUM(J,I+1,N-1,1/D(R[I],T[I],R[J],T[J]))+\
              (IF (R[I]^2 <= 1) THEN 0 \
              ELSE 1000*(EXP(R[I]^2-1)-R[I]^2)))
```

By defining `VN` in this manner, the constraints are implicitly enforced when the function is evaluated and the potential energy function is determined for an arbitrary number of particles, N. One need only specify the value of N and initial guesses for the elements of the arrays `R` and `T` before calling the `MINIMIZE` operator. For example, the twelve particle problem may be solved by typing:

```
N = 12
T = 2*PI*(RAN ON 0^^(N-1))
R = RAN ON 0^^(N-1)
MINIMIZE(VN,R,T)
```

Here we have used the row-wise replication operator, which is denoted by a double caret, to create an eleven element array of zeroes. The MLAB `RAN` operator acting on the array of 11 zeroes returns 11 random numbers between 0 and 1. Thus, we are starting with a random initial configuration. MLAB's response to the `MINIMIZE` command is,

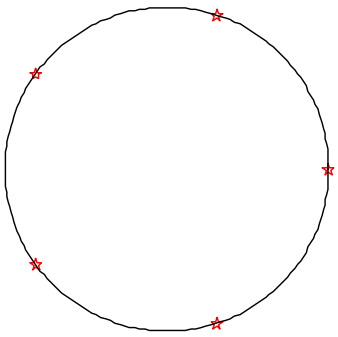
```
The function value is: 5.957297e+01
Argument(s): (1.000140e+00 1.000133e+00 1.000131e+00 1.000131e+00
              1.000141e+00 1.000153e+00 1.000137e+00 1.325854e-02
              1.000140e+00 1.000131e+00 1.000141e+00 3.421776e+00
              1.137801e+00 4.561728e+00 5.134079e+00 1.709461e+00
              5.706133e+00 2.280650e+00 1.395782e+00 2.851709e+00
```

```
              5.675664e-01 3.991057e+00 )
Gradient: (2.070053e-01 -1.246539e-01 -1.329891e-01 -1.460210e-01
           2.102167e-01 7.433105e-01 4.211969e-02 7.718492e-02
           1.650843e-01 -2.124406e-01 2.450464e-01 -2.610247e-03
          -1.696237e-02 -3.086713e-02 -1.094956e-03 8.888392e-04
          -4.157805e-02 -5.216834e-03 1.297788e-04 2.982172e-03
           -1.234044e-02 -3.144967e-02 )
ecode = 1, Exit due to slow rate of convergence.
Try a smaller tolerance value or a larger maxslow value.
# of function evaluations: 287
# of gradient evaluations: 192
# of Quasi-Newton iterations: 149
    = 59.5729652
```

In spite of the `ecode = 1` termination state, the result is the global minimum. The final configuration is visualized by typing the following MLAB commands:

```
DRAW 1&'0 PT STAR
FOR I = 1:(N-1) DO {DRAW (R[I]*COS(T[I]))&'(R[I]*SIN(T[I])) PT STAR}
WINDOW ADJUST WMATCH
VIEW
```

This results in the following figure:



From the results of 2,3, and 4 charges, you might have concluded that the minimum potential energy configuration for N charges confined to the unit disk is an N-gon inscribed on the unit circle. This conclusion is true for eleven or fewer particles, but as Berezin reported in *Nature* 315

(1985) 104, the configuration with 11 charges at the vertices of an 11-gon and one charge in the center is the minimum potential energy configuration for twelve charges. This calculation confirms Berezin's result.

The reader can use MLAB to explore minimum potential energy configurations for other values of N. We mention in passing three results reported by Queen in *Nature* 317 (1985) 208: for N=17, the lowest energy configuration consists of two charges at radius 0.31 and the other 15 charges at radius 1; for N=29, the lowest energy configuration consists of 6 charges at radius .49 and the remaining 23 at radius 1; and for N=30, the lowest energy configuration has one charge at the center, 6 at radius .55 and 23 at radius 1. These findings suggest that the charges tend to spread over the disk as N approaches infinity.

The Dynamical Problem

It is of interest to approach the problem of N charges on a unit disk from another viewpoint: what is the *dynamical motion* of N charges on a unit disk, given initial positions and velocities for each particle. In this section we will show how MLAB can be used to solve the equations of motion for five particles on the unit disk and visualize the resulting trajectory.

Let X1, Y1, X2, Y2, X3, Y3, X4, Y4, X5, and Y5 be the Cartesian coordinates, and PX1, PY1, PX2, PY2, PX3, PY3, PX4, PY4, PX5, and PY5 be the components of velocity for particles 1 through 5. We begin by defining a function for the polar radius of a particle in terms of the Cartesian coordinates of the particle by typing:

```
FCT RV(X,Y) = SQRT(X*X+Y*Y)
```

Ignoring contributions to forces on particles arising from magnetic fields (i.e. forces computed as the cross product of a particle's velocity and the magnetic field, $\vec{v} \times \vec{B}$), we assume the force on the i^{th} charge due to the j^{th} charge is given by Coulomb's law,

$$\vec{F}_{i,j} = \frac{\vec{r}_i - \vec{r}_j}{\sqrt{(\vec{r}_i - \vec{r}_j) \cdot (\vec{r}_i - \vec{r}_j)}^{\,3}}$$

where $\vec{r}_i$ is the vector from the origin to particle i and $\vec{r}_j$ is the vector from the origin to particle j. From Newton's Law, we can write an equation for the time derivative of each component of velocity for each particle:

```
FCT F(UI,VI,UJ,VJ) = (UI-UJ)/(SQRT((UI-UJ)^2+(VI-VJ)^2))^3
FCT PX1'T(T) = F(X1,Y1,X2,Y2)+F(X1,Y1,X3,Y3)+F(X1,Y1,X4,Y4)+F(X1,Y1,X5,Y5)
FCT PY1'T(T) = F(Y1,X1,Y2,X2)+F(Y1,X1,Y3,X3)+F(Y1,X1,Y4,X4)+F(Y1,X1,Y5,X5)
FCT PX2'T(T) = -F(X1,Y1,X2,Y2)+F(X2,Y2,X3,Y3)+F(X2,Y2,X4,Y4)+F(X2,Y2,X5,Y5)
FCT PY2'T(T) = -F(Y1,X1,Y2,X2)+F(Y2,X2,Y3,X3)+F(Y2,X2,Y4,X4)+F(Y2,X2,Y5,X5)
```

```
FCT PX3'T(T) = -F(X1,Y1,X3,Y3)-F(X2,Y2,X3,Y3)+F(X3,Y3,X4,Y4)+F(X3,Y3,X5,Y5)
FCT PY3'T(T) = -F(Y1,X1,Y3,X3)-F(Y2,X2,Y3,X3)+F(Y3,X3,Y4,X4)+F(Y3,X3,Y5,X5)
FCT PX4'T(T) = -F(X1,Y1,X4,Y4)-F(X2,Y2,X4,Y4)-F(X3,Y3,X4,Y4)+F(X4,Y4,X5,Y5)
FCT PY4'T(T) = -F(Y1,X1,Y4,X4)-F(Y2,X2,Y4,X4)-F(Y3,X3,Y4,X4)+F(Y4,X4,Y5,X5)
FCT PX5'T(T) = -F(X1,Y1,X5,Y5)-F(X2,Y2,X5,Y5)-F(X3,Y3,X5,Y5)-F(X4,Y4,X5,Y5)
FCT PY5'T(T) = -F(Y1,X1,Y5,X5)-F(Y2,X2,Y5,X5)-F(Y3,X3,Y5,X5)-F(Y4,X4,Y5,X5)
```

The time derivative of each particle's Cartesian coordinate is set equal to the corresponding component of the velocity so long as the polar radius of the particle is less than one. If the polar radius of the particle becomes greater than one, we constrain the particle to the unit disk by setting the radial component of the velocity equal to zero, preserving only the angular component of the velocity. The function `AN` defined below performs the necessary projection of the velocity vector of each particle. This is expressed in MLAB by typing:

```
FCT G(X,Y,PX,PY) = X*PX+Y*PY
FCT AN(X,Y,PX,PY,R) = PX-G(X/R,Y/R,PX,PY)*X/R
FCT K(X,Y,R,VX,VY,V) = IF (R < 1) THEN V ELSE AN(X,Y,VX,VY,R)
FCT H(X,Y,VX,VY,V) = K(X,Y,RV(X,Y),VX,VY,V)
FCT X1'T(T) = H(X1,Y1,PX1,PY1,PX1)
FCT Y1'T(T) = H(Y1,X1,PY1,PX1,PY1)
FCT X2'T(T) = H(X2,Y2,PX2,PY2,PX2)
FCT Y2'T(T) = H(Y2,X2,PY2,PX2,PY2)
FCT X3'T(T) = H(X3,Y3,PX3,PY3,PX3)
FCT Y3'T(T) = H(Y3,X3,PY3,PX3,PY3)
FCT X4'T(T) = H(X4,Y4,PX4,PY4,PX4)
FCT Y4'T(T) = H(Y4,X4,PY4,PX4,PY4)
FCT X5'T(T) = H(X5,Y5,PX5,PY5,PX5)
FCT Y5'T(T) = H(Y5,X5,PY5,PX5,PY5)
```

The initial positions and velocities are defined using the `INITIAL` statement which is abbreviated `INIT`. As a first example, we initialize four particles to the minimum potential energy configuration found for 4 particles; each particle with zero velocity. The fifth particle is placed slightly to the right of center with an initial velocity directed toward the charge lying on the negative y-axis.

```
INIT X1(0) = 1.; INIT Y1(0) = 0.; INIT PX1(0) = 0.; INIT PY1(0) = 0.;
INIT X2(0) = 0.; INIT Y2(0) = 1.; INIT PX2(0) = 0; INIT PY2(0) = 0;
INIT X3(0) = -1.; INIT Y3(0) = 0.; INIT PX3(0) = 0; INIT PY3(0) = 0;
INIT X4(0) = 0.; INIT Y4(0) = -1; INIT PX4(0) = 0.; INIT PY4(0) = 0;
INIT X5(0) = .30; INIT Y5(0) = .0; INIT PX5(0) = -.1; INIT PY5(0) = -.5;
```

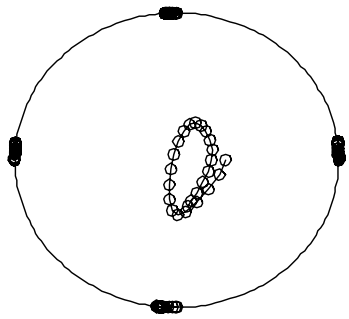
Finally we define the time range and time steps for which the motion is desired and some particulars regarding the differential equation solving method by typing:

```
JACSW = 0
T = 0:5!100
MANDSW = 1
METHOD = ADAMS
M = POINTS(X1,Y1,PX1,PY1,X2,Y2,PX2,PY2,X3,Y3,PX3,PY3,X4,Y4,PX4,PY4,\
           X5,Y5,PX5,PY5,T)
```

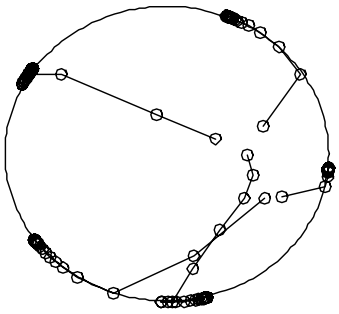
The POINTS operator integrates the equations of motion from time 0 to time 5 in 100 steps using the Adams method. It returns a 21 column by 100 row array with time in column 1 and the coordinates and components of each particle's velocity in the remaining columns. To visualize the trajectory we type:

```
DELETE M ROW 2:NROWS(M):2; DELETE M ROW 2:NROWS(M):2;
NO IMAGEBOX IN W
DRAW M COL (2,3) PT CIRCLE IN W
DRAW M COL (6,7) PT CIRCLE IN W
DRAW M COL (10,11) PT CIRCLE IN W
DRAW M COL (14,15) PT CIRCLE IN W
DRAW M COL (18,19) PT CIRCLE IN W
DRAW (COS ON 0:(2*PI)!200) &' (SIN ON 0:(2*PI)!200) IN W
WINDOW ADJUST WMATCH IN W
```

This results in the following figure:



When we choose more arbitrary initial positions and initial velocities for the charges, we obtain the following result:



Note how some charges move to the boundary, begin to travel in a direction determined by their velocity vector, and then reverse their directions under the influence of the other particles.

Such studies raise some interesting questions concerning dynamical systems. In the first example, how long will the particle in the center oscillate about the center? Will it ever escape to the boundary? Given enough time, will the particles in both examples converge to the minimum potential energy configuration for five charges? Note that this model dissipates energy due to the truncation of radial velocity components at the boundary. However the kinetic energy does not necessarily become 0; if the total angular momentum of the initial configuration is non-zero, the particles can enter a limit cycle in which they rotate forever.

Conclusion This paper has demonstrated how MLAB can be used to determine minimum potential energy configurations and the dynamical motion of N charged particles on the unit disk.

There are related problems which might be studied using MLAB. For example, we have seen that as N increases from 11 to 12, the minimum $\frac{1}{r}$-potential energy configuration expelled a particle to the center of the disk. At what point would the minimum potential energy configuration expel a charge to the center if the inter-particle potential were of another form?

Another interesting problem one might study with MLAB is Thomson's problem: what is the minimum potential energy configuration for N charges constrained to the surface of a unit sphere?

40 Local Maxima and Minima in Time-Series

MLAB has many unusual operators not commonly found in most mathematical and statistical software packages. Two such operators are the threshhold local minima and threshhold local maxima functions.

The threshhold local minima function, LMIN(V[,t]), computes a matrix consisting of those rows of the input matrix V for which the entries in last column of V hold the local minima with threshold t. The optional threshhold value t defaults to zero if it is omitted.

Given a list of values $v_1, v_2, ..., v_n$, the value v_i is a threshold-t local minimum if there exists a neighborhood $v_j, v_{j+1}, \ldots, v_i, v_{i+1}, \ldots, v_h$ of v_i with $j < i < h$ such that

(1) $v_i = \min(v_j, \ldots, v_h)$

(2) $v_i < v_k$ for $k = j, \ldots, i-1$

(3) $v_k - v_i \leq t$ for $k = j+1, \ldots, h-1$

(4) $v_j - v_i > t$ and $v_h - v_i > t$

The threshhold local maxima function, LMAX(V[,t]), computes a matrix consisting of those rows of the input matrix V for which the entries in last column of V hold the local maxima with threshold t. The optional threshhold value t defaults to zero if it is omitted.

Given a list of values $v_1, v_2, \ldots, v_n$, the value v_i is a threshold-t local maximum if there exists a neighborhood $v_j, v_{j+1}, \ldots, v_i, v_{i+1}, \ldots, v_h$ of v_i with $j < i < h$ such that

(1) $v_i = \max(v_j, \ldots, v_h)$

(2) $v_i > v_k$ for $k = j, \ldots, i-1$

(3) $v_i - v_k \leq t$ for $k = j+1, \ldots, h-1$

(4) $v_i - v_j > t$ and $v_i - v_h > t$

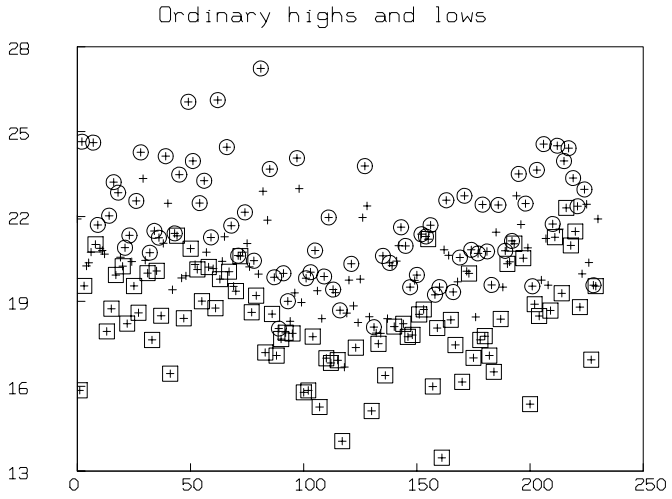
Suppose we wish to determine the successive points at which a stock price hits a new high or a new low. this can be done by finding all the local maxima and local minima with the LMAX and LMIN operators.

Suppose however, we want to define a new high or new low as a price which is more extreme than its ranges of neighboring prices by a certain percentage, say 5%. This isn't quite a precise concept, because exactly what are we assessing 5% of? Moreover, the LMAX and LMIN operators deal with absolute, not relative thresholds. We may resolve these problems by normalizing our prices relative to their moving average, *i.e.* by scaling a price $p(t)$ by the value $1/m(t)$ where $m(t)$ is

a suitable moving average price at time t. We may then apply LMAX and LMIN to this scaled time-series with the desired threshhold value.

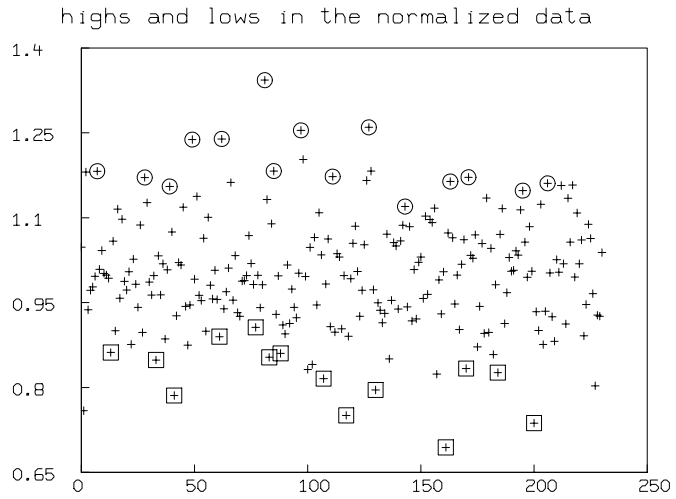
Here is an example of this device for a particular time-series of 230 prices at successive time-points.

```
p=read(stock,230,2)  /* read in data from file stock.dat */

draw p pt crosspt ptsize .01 lt none
hi=lmax(p)
lo=lmin(p)
draw hi pt circle lt none color red
draw lo pt square lt none color green
top title "Ordinary highs and lows"
view
```

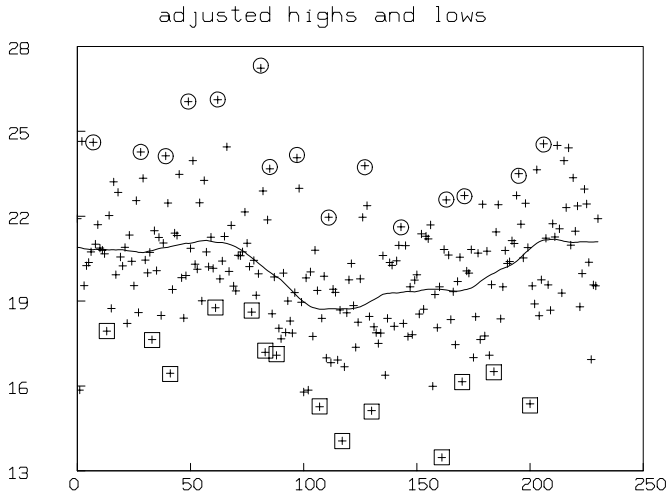



```
del w

wm=(0:1!15)&(1^^15)&(1:0!15)
v=mmean(p col 2,45,0,4,wm)
m=(p col 1)&'v
s=(p col 1)&'((p col 2)/'v)
draw s pt crosspt ptsize .01 lt none

hi=lmax(s,.3)
```

```
lo=lmin(s,.3)
draw hi pt circle lt none color red
draw lo pt square lt none color green
top title "highs and lows in the normalized data"
view
```




```
del w

draw p pt crosspt ptsize .01 lt none
draw m color yellow
hi col 2 = (hi col 2)*'(v row (hi col 1))
lo col 2 = (lo col 2)*'(v row (lo col 1))
draw hi pt circle lt none color red
draw lo pt square lt none color green
top title "adjusted highs and lows"
view
```


adjusted highs and lows
28
25
22
19
16
13
0
50
100
150
200
250

41 Surface Drawing with Sweeping

(Zhiping You)

A surface Q in 3-dimensional space may be represented in two parameter form as a function $Q(s,t) = (x(s,t), y(s,t), z(s,t))$. A special case of this is that the surface is the surface of a function. Then the parameter form of the surface is just $(x, y, z(x,y))$. In this paper, we will discuss a special case of surfaces in 3 dimensions. *i.e.* tube-like-surfaces that are generated by sweeping a space curve along another central space curve.

Given a space curve $r(t) = (x(t), y(t), z(t))$, at each point, there are three directions associated with it, the *tangent*, *normal* and *bi-normal* directions. The unit *tangent* vector is denoted by T, *i.e.* $T(t) = r'(t)/\|r'(t)\|$, the unit *normal* vector is denoted by N, *i.e.* $N(t) = T'(t)/\|T'(t)\|$, the unit *bi-normal* vector is denoted by B, *i.e.* $B(t) = T(t) \times N(t)$ (cross product). With $r(t), T(t), N(t)$ and $B(t)$, a tube-like surface can be expressed as follows:

$$Q(s,t) = r(t) + d \cdot (cos(s)N(t) - sin(s)B(t))$$

where d is a parameter corresponding to the radius of the rotation (In general d can be a function of t). For fixed t, when s runs from 0 to 2π, we have a circle around the point $r(t)$ in the N, B plane. As we change t, this circle moves along the space curve r, and we will generate a tube-like surface along r.

A simple example of this is a torus, where r is a circle. $r(t) = (cos(t), sin(t), 0)$. In this case, $T(t) = (-sin(t), cos(t), 0)$, $N(t) = (-cos(t), -sin(t), 0)$ and $B(t) = (0, 0, 1)$. Thus, the parametric form of the torus is:

$$\begin{aligned} x(s,t) &= cos(t) - d \cdot cos(s)cos(t) \\ y(s,t) &= sin(t) - d \cdot cos(s)sin(t) \\ z(s,t) &= d \end{aligned}$$

We can easily construct and draw such a torus in *MLAB* by merely computing the x, y and z functions on a regular grid of (s,t)-points. Using the functions x, y and z, this can be done as follows:

```
grid = cross((0:(2*pi)!30, 0:(2*pi)!30)
d = 0.5
m = (x on grid) &' (y on grid) &' (z on grid)
draw m lt hide
```

Note that the *MLAB* draw statement is able to correctly discover the underlying grid structure in the matrix *m* automatically!

We can also build a general *do-file* which draws a tube-like-surface given a central curve and a radius function. Here is such an *MLAB do-file*.

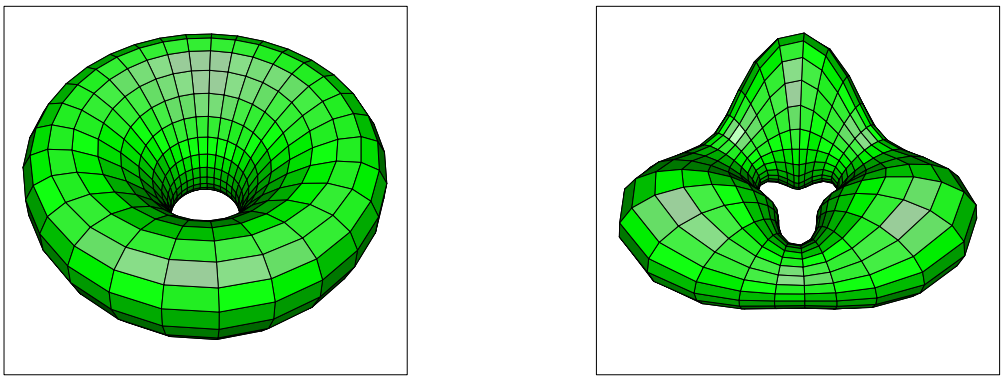
```
"tube.do"
"generate a tube of radius d(t) around the curve: r(t) = (x(t),y(t),z(t))"
"The central space curve component functions x(t), y(t) and z(t) must be"
"predefined. The radius function d(t) must also be predefined, and"
"finally the (s,t) grid matrix called grid must also be predefined."

"Construct unit tangent vector T(t)"
"nd(t) is the norm of the derivative"
fct nd(t) = ((x't(t))^2+(y't(t))^2+(z't(t))^2)^0.5
"a(t), b(t) and c(t) are the component of T(t)"
fct a(t) = x't(t)/nd(t)
fct b(t) = y't(t)/nd(t)
fct c(t) = z't(t)/nd(t)

"Construct unit normal vector N(t)"
"ndut(t): norm of the derivative of the unit tangent vector"
fct ndut(t) = ((a't(t))^2+(b't(t))^2+(c't(t))^2)^0.5
"an(t), bn(t) and cn(t) are the component of N(t)"
fct an(t) = a't(t)/ndut(t)
fct bn(t) = b't(t)/ndut(t)
fct cn(t) = c't(t)/ndut(t)

"Construct the unit binormal vector B(t)"
"ab(t), bb(t) and cb(t) are the component of B(t)"
fct ab(t) = b(t)*cn(t)-c(t)*bn(t)
fct bb(t) = c(t)*an(t)-a(t)*cn(t)
fct cb(t) = a(t)*bn(t)-b(t)*an(t)

"Now, construct the space tube"
"m(s,t) = r(t) + d*(cos(s)*N(t) + sin(s)*B(t))"
"d is the radius of the tube along the central curve"
"mx(s,t), my(s,t), mz(s,t) is the component of m(s,t)"
fct mx(s,t) = x(t) + d(t)*(cos(s)*an(t)+sin(s)*ab(t))
fct my(s,t) = y(t) + d(t)*(cos(s)*bn(t)+sin(s)*bb(t))
fct mz(s,t) = z(t) + d(t)*(cos(s)*cn(t)+sin(s)*cb(t))
```

```
"now generate the surface"
mm = (mx on grid) &' (my on grid) &' (mz on grid)
draw mm, linetype net
view
"end of tube.do"
```

Here is an example showing the use of tube.do to draw a torus.

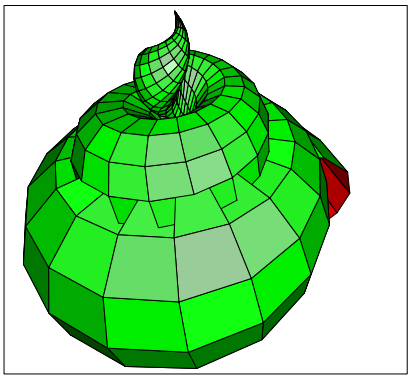
```
"the central line curve is (cos t, sin t, 0)"
"define the center space curve r(t) = (x(t), y(t), z(t))"
fct x(t) = cos(t)
fct y(t) = sin(t)
fct z(t) = 0
grid = cross((0:(2*pi)!30), (0:(2*pi)!30))
fct d(t) = 1
do tube.do
```

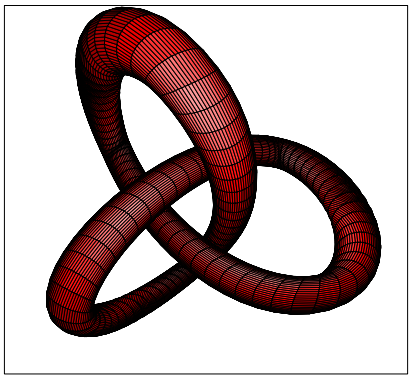
Here is some output from the above *do-file* tube.do.



Central curve r(t) = (cos(t), sin(t), 0). Left: a fixed radius. Right: a variable radius.

One can generate more interesting surfaces with other central curves. Here are two examples with two different central curves.





Left: Descending central curve and ascending radius. Right: self crossing central curve and constant radius.

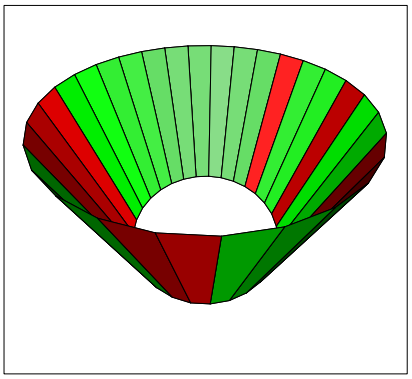
Generally speaking, our sweeping curve does not need to be a circle, it can be any 2-dimensional curve $c(s,t) = (p(s,t), q(s,t))$. Thus, instead of having

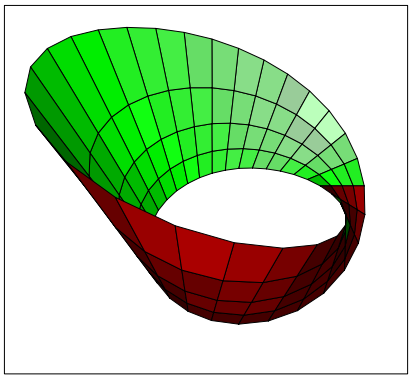
$$Q(s,t) = r(t) + d \cdot (cos(s)N(t) - sin(s)B(t))$$

as the sweeping surface, we have:

$$Q(s,t) = r(t) + p(s,t) \cdot N(t) - q(s,t) \cdot B(t)$$

In particular, when the curve c is a straight line segment, the sweeping surface becomes a band. Here are two such bands: one with a fixed line segment, and one with a rotating line segment which is the famous *Mobius* band.





Left: Regular band with fixed radius and direction. Right: Mobius band.

One can easily see that if we adjust the radius, direction, number of points on each circle etc, we can generate many interesting surfaces.

42 Contour Maps

MLAB has a very powerful contour map algorithm. It computes a matrix holding the points on the individual level curves, so that their paths can be used in further computations, as well as graphed when desired. (Many contour algorithms produce plots, but do not produce explicit numbers, so nothing can be done with such algorithms, except look at the produced plot.) The paths generated with the *MLAB CONTOUR* operator are guaranteed to be non-self-intersecting. The most complex pattern that can arise is a four-leaf-clover shape.

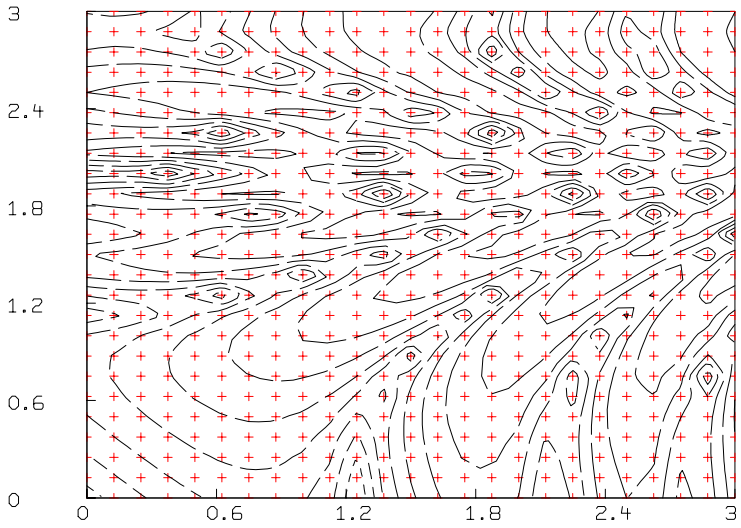
The *MLAB CONTOUR* operator does require that the data points of the surface to be "contoured" are given corresponding to a regular grid. If only irregularly-positioned points are known, the surface interpolation function can be used to obtain data points on a regular grid by nearest-neighbor average triangle interpolation.

Some examples of contour maps produced in *MLAB* are given below.

Example 1: Plot a contour map of the function

$$f(x,y) = sqrt(abs(cos(x^2 - y^3))) + log(abs(x+y)+1)$$

in the rectangle $[0,3] \times [0,3]$.



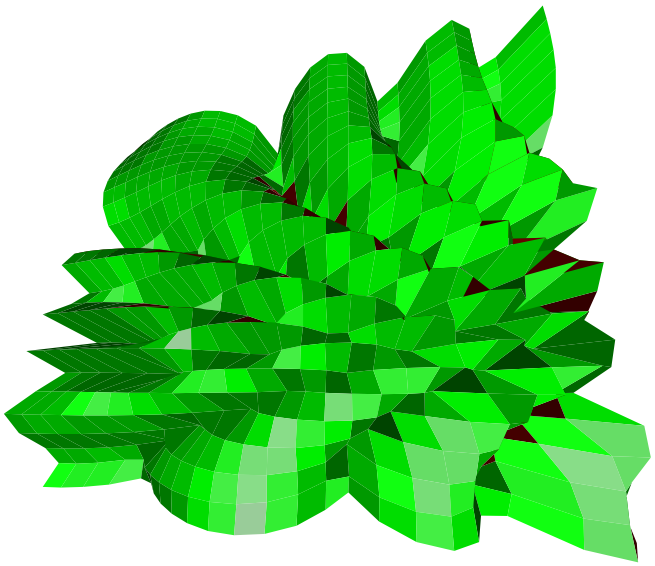
contour map for the function $f(x,y) = sqrt(abs(cos(x^2 - y^3))) + log(abs(x+y)+1)$ *in* $[0,3] \times [0,3]$.

The *MLAB* commands to do this are given below.

```
* FUNCTION F(x,y) = SQRT(ABS(COS(X^2-Y^3))) + LOG(1+ABS(X+Y))
* M = CROSS(0:3!25, 0:3!25)
* M COL 3 = F ON M
* DRAW CONTOUR(M) LINETYPE SVMARKER
* DRAW M COL 1:2, LINETYPE NONE, POINTYPE CROSSPT, PTSIZE .01 COLOR YELLOW
* VIEW
```

We can also look at our surface using the *MLAB* 3*D* graphics facilities as follows.

```
* DELETE W
* DRAW M LINETYPE HIDDEN
* VIEW
```

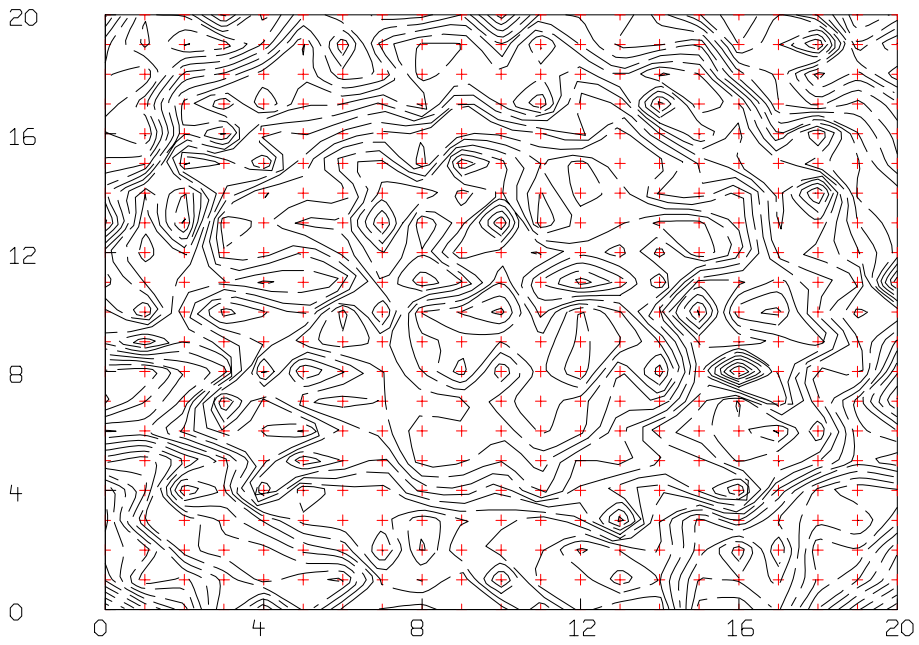


3D picture corresponding to the previous contour map

Example 2: Plot a contour map of the 441 data values stored in the file CDATA.DAT which are the elevations of a surface, stored by rows, corresponding to the grid $\{0, 1, \ldots, 20\} \times \{0, 1, \ldots, 20\}$.

```
* D = READ(CDATA,21,21)
```

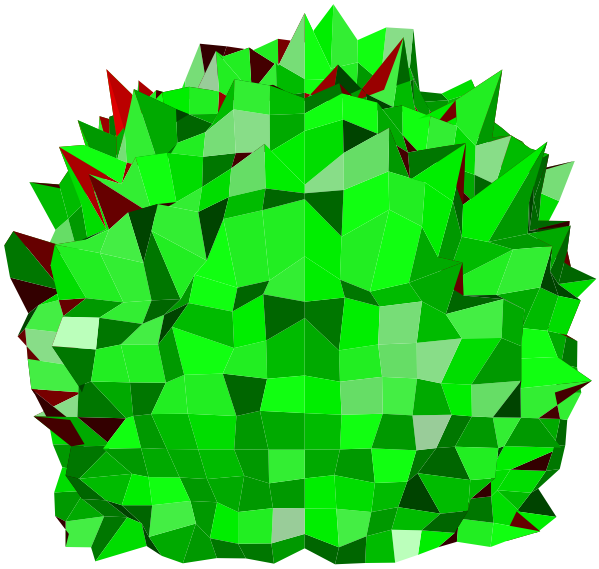
```
* D = CROSS(0:20,0:20) &' LIST(D)
* C = CONTOUR(D,30)
* DRAW C LINETYPE SVMARKER
* DRAW D COL 1:2, LT NONE, PT CROSSPT, PTSIZE .01, COLOR YELLOW
* VIEW
```



contour map for the surface described by the data points in CDATA.DAT.

We can also look at our surface using the *MLAB* 3*D* graphics facilities as follows.

```
* DELETE W
* DRAW M LINETYPE HIDDEN
* VIEW
```



3D picture corresponds to the previous contour map

Note we have not described how hard-copy plots are made, but this document shows plotting can be easily done.

43 Spatial Statistics(Voronoi Diagrams)

Our problem is to decide if two bivariate random vectors have the same bivariate distribution functions, except for possibly different isotropic scalings, given two sets of samples: $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\} \subset \mathcal{R}^2$.

One approach to this problem is based on the so-called Voronoi diagram (a Voronoi diagram is also known as a Dirichlet tessallation) of each point set. The Voronoi diagram of a point set $\{x_1, \ldots, x_n\}$ in the plane is the collection of closed regions $V_1, \ldots, V_n$ which are the "natural" neighborhoods of the data points $x_1, \ldots, x_n$ in the plane; the Voronoi regions $V_1, \ldots, V_n$ satisfy the following conditions.

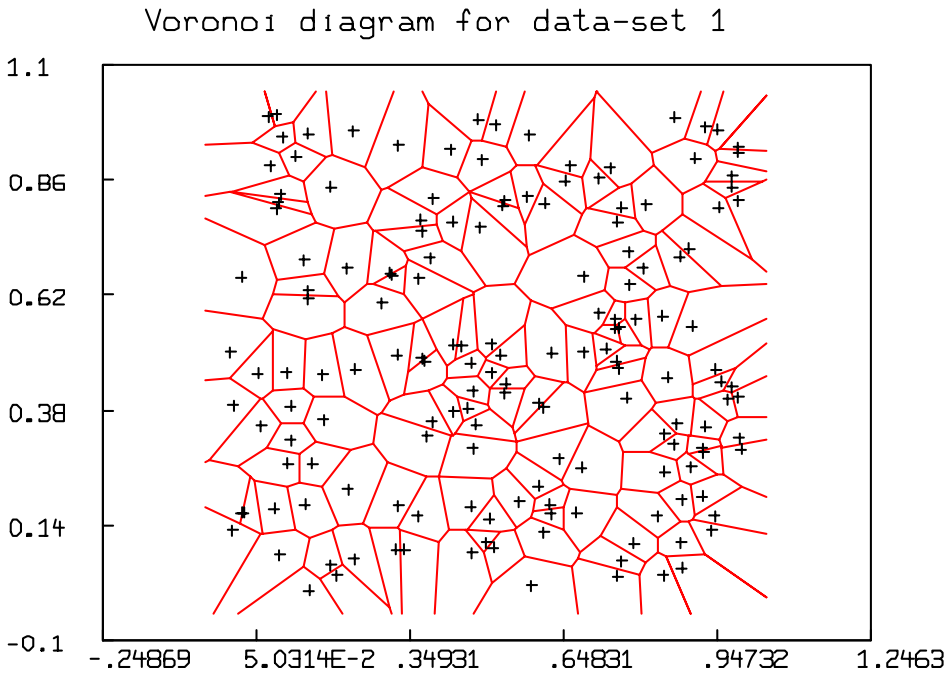
1. $x_i \in V_i$ for $i = 1, \ldots, n$
2. $V_i \cap V_j = \partial V_i \cap \partial V_j$ for $i \neq j$. When $V_i \cap V_j$ is not empty, it is a segment on the bisector line of the line-segment between x_i and x_j.
3. $V_1 \cup V_2 \cup \cdots \cup V_n = \mathcal{R}^2$.
4. $V_i = \{\, x \mid |x - x_i|_2 \leq |x - x_j|_2 \text{ for } \quad j = 1, \ldots, n \,\}$.

If the two spatial samples come from the same bivariate population (with possible isotropic scale differences), then the suitably-scaled areas of the bounded Voronoi regions will also have the same univariate distribution. Thus we may compute the two empirical cumulative distribution functions of the two sets of values corresponding to the scaled areas of the bounded Voronoi regions of the two spatial samples.

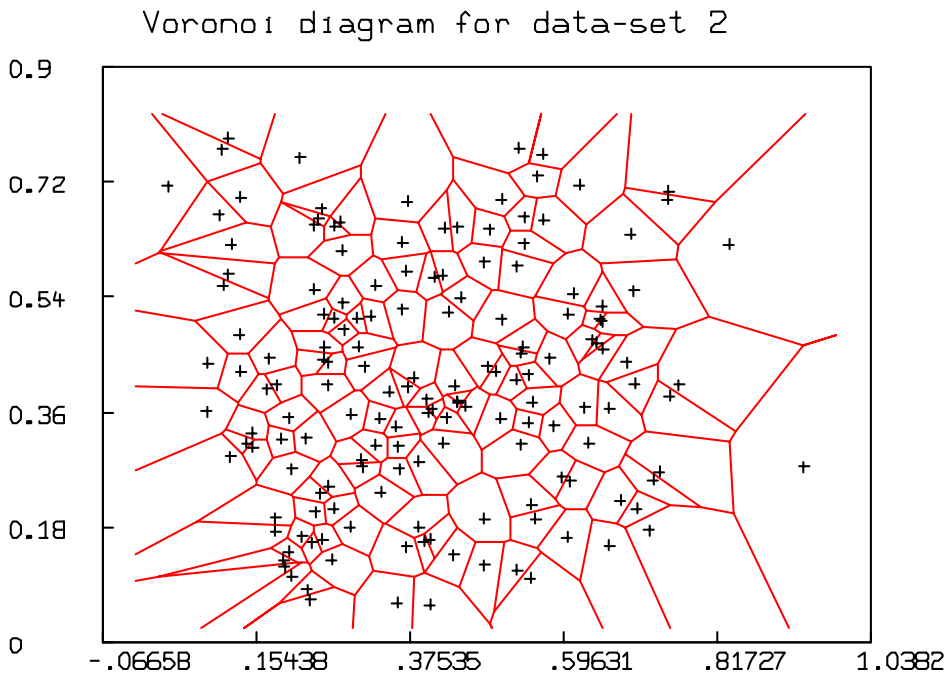
We may then use the Kolomogorov-Smirnov test to decide if these two cumulative distribution functions are plausibly based on the same underlying population.

This process is carried-out in MLAB as follows.

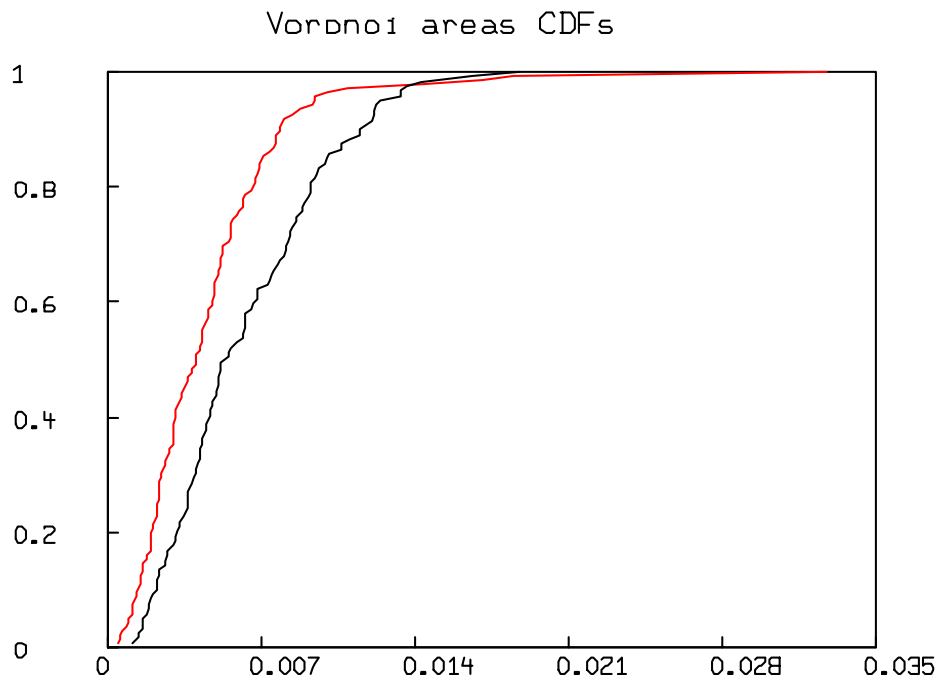
```
* d1 = read(d1,200,2)
* draw d1 pt crosspt ptsize .01 lt none
* draw vorcurve(d1) lt marker color red
* window adjust wmatch
* top title "Voronoi diagram for data-set 1"
* view
```




```
* d2 = read(d2,200,2)
* draw d2 pt crosspt ptsize .01 lt none
* draw vorcurve(d2) lt marker color red
* window adjust wmatch
* top title "Voronoi diagram for data-set 2"
* view
```




```
* v1 = vorstat(d1) col 2
* v1 = compress(v1)
* xd = maxv(d1 col 1) - minv(d1 col 1)
* yd = maxv(d1 col 2) - minv(d1 col 2)
* az1 = xd*yd
* v1 = v1/az1
* z1 = cdf(v1)
*
* v2 = vorstat(d2) col 2
* v2 = compress(v2)
* xd = maxv(d2 col 1) - minv(d2 col 1)
* yd = maxv(d2 col 2) - minv(d2 col 2)
* az2 = xd*yd
* v2 = v2/az2
* z2 = cdf(v2)
*
* draw z1
* draw z2 color red
* top title "Voronoi areas CDFs"
* view
```

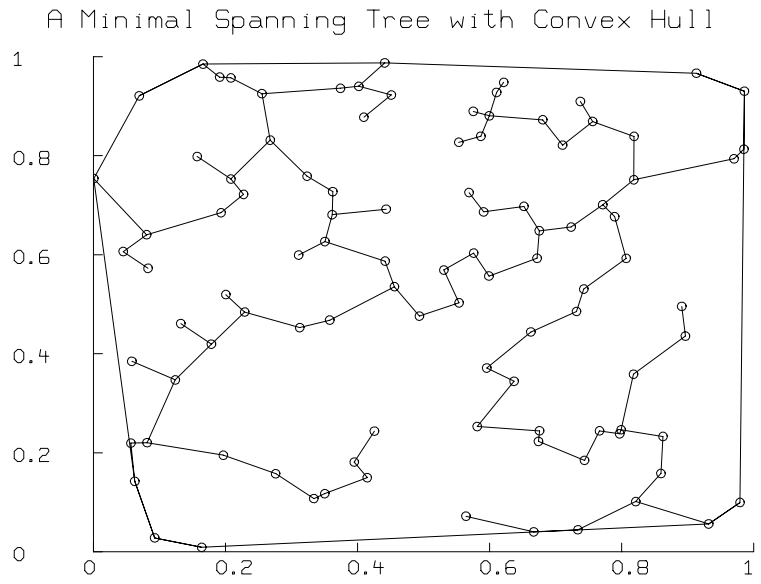



```
* ksf2t(z1,z2)
[K-S-test: are M1 and M2 empirical cdf estimates of the same distribution?]
null hypothesis H0: M1 and M2 are cdf estimates of the same distribution.
Then the scaled maximum deviation between the cdf(M1) and the cdf(M2) is
distributed according to the Kolmogorov-Smirnov K statistic.
The K-statistic value = 1.991649

The probability P(K > 1.991649) = 0.000717
This means that a value of K larger than 1.991649 arises
about 0.071718 percent of the time, given H0.

    : a  5 by 1 matrix

  1: 1.99164923
  2: 0.25
  3: 0.25
  4: 7.17178243E-4
  5: 6.19572536E-3
```

44 Computing A Minimal Spanning Tree

Given a set of nodes and a metric defining the distances between the nodes, people often want to find a minimal spanning tree for these nodes. MLAB has a built-in function, `MST`, to construct such a tree. Given a distance matrix D for a set of points given in the rows of a 2-column matrix M, `MST(D)` generates a 3-column output matrix defining the edges and edge-length of a minimal spanning tree of the points in M. Here is an example of MLAB computing a minimal spanning tree. We also computeand display the convex hull of these points.

```
m = shape(100,2, ran on 0^^200) /* make random data */
d = dists(m) /* find the distances between the points. */
e = mst(d) /* compute the minimal spanning tree */

v = mesh(m row (e col 1), m row (e col 2))

draw m pt circle ptsize .01 lt none
draw v lt alternate color red
top title = "A Minimal Spanning Tree with Convex Hull"
q = chull(m) /* compute the convex hull of m */
q = q & (q row 1) /* make it a closed curve */
draw q
view
```



45 Great Circle Routes on a Mercator Projection of the World

The shortest route between two points on the globe lies on an arc of the great circle which contains the two points. The great circle containing two points is unique so long as the two points are not on opposite sides of the globe.

In this MLAB example we are concerned with the great circle route between two points as it appears on a *Mercator* projection. The Mercator projection is a common and familiar map that is obtained by 1) projecting coastlines of land masses on the surface of the globe from the center of the globe onto a cylinder that surrounds the globe and is tangent to the globe along the equator; 2) cutting the cylinder along the projection of a meridian–in this case we choose the 180 degree meridian; and 3) unrolling the cylinder. Owing to the finite size of a map, the cylinder is truncated at the top and the bottom so that the north and south poles are not represented. In the Mercator projection of the globe, lines of constant latitude and lines of constant longitude (meridians) appear as straight line segments.

The great circle route between two points on a Mercator projection is particularly simple if the two points both have zero latitude or if the two points both have equal longitudes. In the former case it is a horizontal line segment along the equator, and in the latter case it is a vertical line segment along a meridian. But for the general case of two points with different latitudes and longitudes, the great circle route between the two points on the Mercator projection is not a straight line segment. The following MLAB script computes and draws the great circle route between two arbitrary points on a Mercator projection of the world.

```
"file: gcr.do"
"What is the great circle route on a Mercator projection for two"
"arbitrary points?  Parameterize the great circle route as:"

"  (1-t)*R1+t*R2/(norm(t*R1+(1-t)*R2))"

"where R1 is the unit vector in 3 space pointing to (lat1,lon1),"
"      R2 is the unit vector in 3 space pointing to (lat2,lon2),"
"and   t is a scalar ranging from 0 to 1."

"the default points are Atlantic City, NJ and Gibraltar, Spain."
clearscreen()

if (dtype(pass1) = UNKNOWN) then \
   {pass1 = 1;

    "draw the Mercator projection first time through"
    "map.dat contains a digitized rendition of the world land"
```

```
    "masses' coastlines"
    n = read("map.dat",2000,2)
    m col 1 = n col 1
    m col 2 = tand on n col 2
    draw m lt marker
    window -180 to 180, tand(-65) to tand(65)
    yaxis label list(-65,-52,-23,23,52,65)

    "define functions for converting between ll=(lat,lon) and c=(x,y,z) where"
    "(x,y,z) are Cartesian coordinates in Euclidean 3 space with origin at"
    "the center of the earth, x-axis directed from the origin to"
    "(lat=0,lon=0), y-axis directed from the origin to (lat=0,lon=90),"
    "and z-axis directed from the origin to the north pole"
    fct ll2x(lat,lon) = cosd(lon)*sind(90-lat)
    fct ll2y(lat,lon) = sind(lon)*sind(90-lat)
    fct ll2z(lat,lon) = cosd(90-lat)
    fct c2lon(x,y,z) = atan2(y,x)*180/pi
    fct c2lat(x,y,z) = 90-acos(z)*180/pi

    "define functions for generating the great circle arc in Euclidean"
    "3 space. alpha[1] will be defined as the angle between r1 and r2"
    fct nn(t) = sqrt(1+2*t*(1-t)*(alpha[1]-1))
    fct x(t) = ((1-t)*r1[1]+t*r2[1])/nn(t)
    fct y(t) = ((1-t)*r1[2]+t*r2[2])/nn(t)
    fct z(t) = ((1-t)*r1[3]+t*r2[3])/nn(t)
    tl = 0:1!360
   }

"get longitude and latitude of starting point"
ttl = "Enter a STARTING point: "
pr[1] = "Latitude (-90<South<0,0<North<90)  "
pr[2] = "Longitude (-180<West<0,0<East<180) "
df[1] = "39"
df[2] = "-74"
str1 = GetStrings(ttl,pr,df)

"convert string responses to numbers and check for valid input"
st = "lat1 ="+str1[1]; do st;
st = "lon1 ="+str1[2]; do st;
if ((abs(lat1) > 90) OR (abs(lon1) > 180)) then break;

"get longitude and latitude of ending point"
```

```
ttl = "Enter an ENDING point:   "
df[1] = "36"
df[2] = "-5"
str1 = GetStrings(ttl,pr,df)

"convert string responses to numbers and check for valid input"
st = "lat2 ="+str1[1]; do st;
st = "lon2 ="+str1[2]; do st;
if ((abs(lat2) > 90) OR (abs(lon2) > 180)) then break;

"if a point is on a pole, move it off the pole by .01 degree.  this"
"avoids singularities in the Mercator projection since poles under"
"that projection map to infinity."
if (lat1 = 90) then lat1 = 89.99;
if (lat1 = -90) then lat1 = -89.99;
if (lat2 = 90) then lat2 = 89.99;
if (lat2 = -90) then lat2 = -89.99;

"order r1 = (lat1,lon1) and r2 = (lat2,lon2) so that r1 is west of r2"
if (lon1 > lon2) then \
   {"swap r1 and r2"
    lat = lat2; lat2 = lat1; lat1 = lat;
    lon = lon2; lon2 = lon1; lon1 = lon;
   }

"draw cases where great circle is simply a point or straight line segment on"
"a meridian or the equator. These cases do not require calculation."
if ((lat1 = lat2) AND (lon1 = lon2)) then \
   {"start and end at same point"
    draw lon1 &' tand(lat1) lt none pt dotpt color red;
    view;
    break;
   }
if (lon1 = lon2) then \
   {"the great circle route follows a meridian"
    p = (lon1 &' lat1) & (lon2 &' lat2);
    p col 2 = tand on p col 2;
    draw p color red;
    view;
    break;
   }
if (lon2-lon1 = 180) then \
```

```
   {"The great circle route follows a meridian through a pole."
    "Set i to 1 if its the north pole, -1 if its the south pole."
    "This catches the case where (lat1,lon1) and (lat2,lon2) are on"
    "opposite sides of the globe and there are many great circle arcs."
    "In that case we generate the great circle arc passing through the"
    "north pole."
    if (lat2 >= -lat1) then i = 1 else i = -1;
    p = (lon1 &' lat1) & (lon1 &' i*89.9);
    p col 2 = tand on p col 2;
    draw p color red;
    p = (lon2 &' lat2) & (lon2 &' i*89.9);
    p col 2 = tand on p col 2;
    draw p color red;
    view;
    break;
   }

"here we have (lat1,lon1) and (lat2,lon2) which are distinct, with"
"(lat1,lon1) to the west of (lat2,lon2); the great circle route not does"
"lie on a meridian."

"convert (lat1,lon1) and (lat2,lon2) to vectors r1 and r2"
r1[1] = ll2x(lat1,lon1); r1[2] = ll2y(lat1,lon1); r1[3] = ll2z(lat1,lon1);
r2[1] = ll2x(lat2,lon2); r2[2] = ll2y(lat2,lon2); r2[3] = ll2z(lat2,lon2);

"find the angle between the two vectors"
alpha = r1'*r2

"generate 360 Cartesian points between r1 and r2"
p = (x on tl) &' (y on tl) &' (z on tl)

"convert Cartesian points (x,y,z) in p back to (lon,lat)"
p = (c2lon on p) &' (c2lat on p)

"apply transformation to get Mercator projection"
p col 2 = tand on p col 2

"draw the great circle arc"
draw p lt none pt dotpt color red
view
"end of file: gcr.do"
```

Here is the log-file which results from running gcr.do with the default starting and ending points:

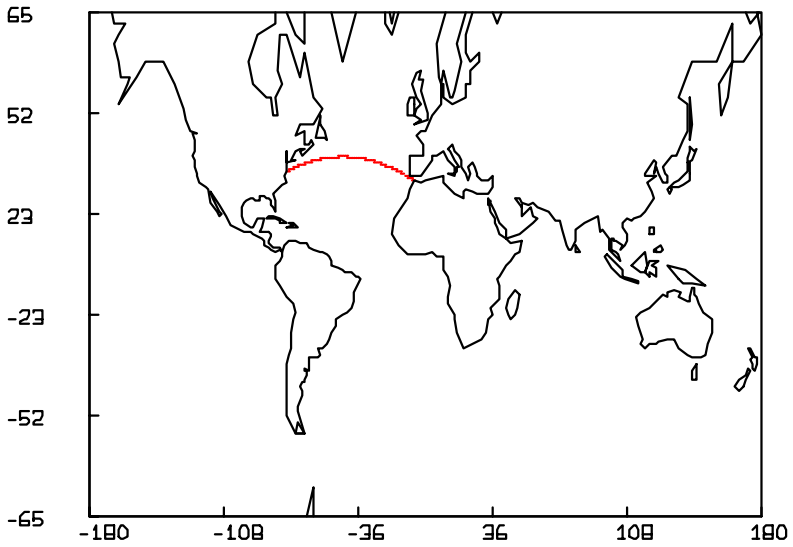
```
    MLAB: Mathematical Modeling System, Revision: October 1, 1994
    Copyright: Civilized Software, Inc. (301)652-4714

  Thu Aug 24 14:41:28 1995
  '* ' is the command prompt

  * do gcr
```

Enter a STARTING point:	
Latitude (-90<South<0,0<North<90)	39
Longitude (-180<West<0,0<East<180)	-74

Enter an ENDING point:	
Latitude (-90<South<0,0<North<90)	36
Longitude (-180<West<0,0<East<180)	-5

```
  * exit
  end of MLAB.LOG
```

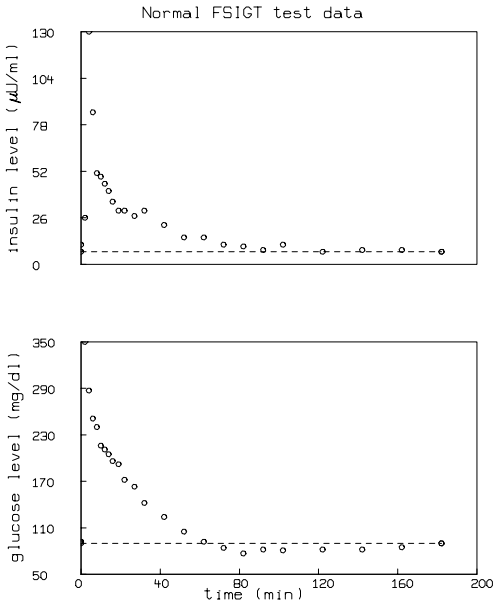
The final view command in the script results in the following figure:



The figure shows the great circle route from Atlantic City, NJ (latitude = 39, longitude = -74) to Gibraltar, Spain (latitude = 36, longitude = -5). Note that although Gibraltar is southeast of Atlantic City, the initial heading along the great circle route from Atlantic City to Gibraltar is *north*easterly.

46 The Minimal Models for Glucose and Insulin Kinetics

Minimal models of glucose and insulin plasma levels in dogs and humans during frequently-sampled intravenous glucose tolerance (FSIGT) tests have been developed and used by Dr. Richard N. Bergman and co-workers since the 1970's. (See references 1-5.) In a typical FSIGT test, blood samples are taken from a fasting subject at regular intervals of time, following a single intravenous injection of glucose. The blood samples are then analyzed for glucose and insulin content. The figure below shows a typical response from a normal subject.



Qualitatively, the glucose level in plasma starts at a peak due to the injection, drops to a minimum which is below the basal (pre-injection) glucose level, and then gradually rises to the basal level. The insulin level in plasma rapidly rises to a peak immediately after the injection, drops to a minimum which is above the basal insulin level, rises again to a lesser peak, and then gradually drops to the basal level. Depending on the state of the subject, there can be wide variations from this response; for example, the glucose level may not drop below basal level, the first peak in insulin level may have different amplitude, there may be no secondary peak in insulin level, or there may be more than two peaks in insulin level.

The glucose and insulin minimal models provide a quantitative and parsimonious description of glucose and insulin concentrations in the blood samples following the glucose injection. The glucose minimal model involves two physiologic compartments: a plasma compartment and an interstitial tissue compartment; the insulin minimal model involves only a single plasma compartment. The glucose and insulin minimal models allow us to characterize the FSIGT test data in terms of four

metabolic indices:

- S_I = insulin sensitivity: the dependence of fractional glucose disappearance on plasma insulin,
- S_G = glucose effectiveness: the fractional ability of glucose to lower its own concentration in plasma independent of increased insulin,
- ϕ_1 = first phase pancreatic responsivity: a measure of the size of the first peak in plasma insulin due to the glucose injection, and
- ϕ_2 = second phase pancreatic responsivity: a measure of the size of the second peak of plasma insulin which follows the first peak and the refractory period.

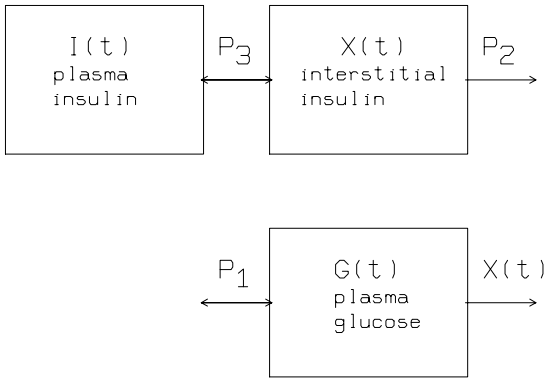
This paper will demonstrate the use of the mathematical modeling computer program MLAB to simulate insulin and glucose plasma levels during an FSIGT test and determine values of the metabolic indices from a data set via curve-fitting.

Reference 3 provides the following FSIGT test data (also shown in the graphs above) from a normal individual:

time (minutes)	glucose level (mg/dl)	insulin level (μU/ml)
0	92	11
2	350	26
4	287	130
6	251	85
8	240	51
10	216	49
12	211	45
14	205	41
16	196	35
19	192	30
22	172	30
27	163	27
32	142	30
42	124	22
52	105	15
62	92	15
72	84	11
82	77	10
92	82	8
102	81	11
122	82	7
142	82	8
162	85	8
182	90	7

Using a spreadsheet program, word processor, or the MLAB file editor, these numbers can be stored in an ASCII text file named `TEST.DAT`.

The following diagram summarizes the minimal model for glucose kinetics:



Insulin leaves or enters the interstitial tissue compartment at a rate proportional to the difference between the plasma insulin level, $I(t)$, and the basal level, I_b; if the plasma insulin level falls below the basal level, insulin leaves the interstitial tissue compartment, and if the plasma insulin level rises above the basal level, insulin enters the interstitial tissue compartment. Insulin also disappears from the interstitial tissue compartment via a second pathway at a rate proportional to the amount of insulin in the interstitial tissue compartment.

Glucose leaves or enters the plasma compartment at a rate proportional to the difference between the plasma glucose level, $G(t)$, and the basal level, G_b; if the plasma glucose level falls below the basal level, glucose enters the plasma compartment, and if the glucose level rises above the basal level, glucose leaves the plasma compartment. Glucose also disappears from the plasma compartment via a second pathway at a rate proportional to the amount of insulin in the interstitial tissue.

The differential equations corresponding to the glucose minimal model are:

$$\begin{aligned} \frac{dX(t)}{dt} &= -p_2 \cdot X(t) + p_3 \cdot (I(t) - I_b), \text{ and} \\ \frac{dG(t)}{dt} &= -X(t) \cdot G(t) + p_1 \cdot (G_b - G(t)) \end{aligned}$$

with $G(0) = G_0$ and $X(0) = 0$. In these equations, t is time, $G(t)$ is the plasma glucose concentration at time t, $I(t)$ is the plasma insulin concentration at time t, and $X(t)$ is the interstitial insulin at time t. G_b is the basal plasma glucose concentration and I_b is the basal plasma insulin concentration. Basal plasma concentrations of glucose and insulin are typically measured either before, or 180 minutes after, administration of glucose. There are four unknown parameters in this model: p_1, p_2, p_3, and G_0.

Note that in this model, glucose is utilized at the constant rate p_1, when we neglect *feedback* effects due to interstitial insulin as represented by the term $-X(t) \cdot G(t)$. An additional amount of plasma insulin will cause the amount of interstitial insulin to change, which in turn, will cause the rate of glucose utilization to change. The *insulin sensitivity* is defined as $S_I = p_3/p_2$ and the *glucose effectiveness* is defined as $S_G = p_1$.

The following MLAB commands in the do-file `g.do` estimate values for the parameters p_1, p_2, p_3, and G_0 given the time course of plasma glucose and insulin. The values of the parameters found minimize, in the least squares sense, the weighted difference between the measured time course of plasma glucose and the parameter-dependent solution to the glucose minimal model differential equations. The plasma insulin concentration function is obtained by linear interpolation of the time-insulin values listed in `TEST.DAT`. This is done by employing the MLAB function `LOOKUP`.

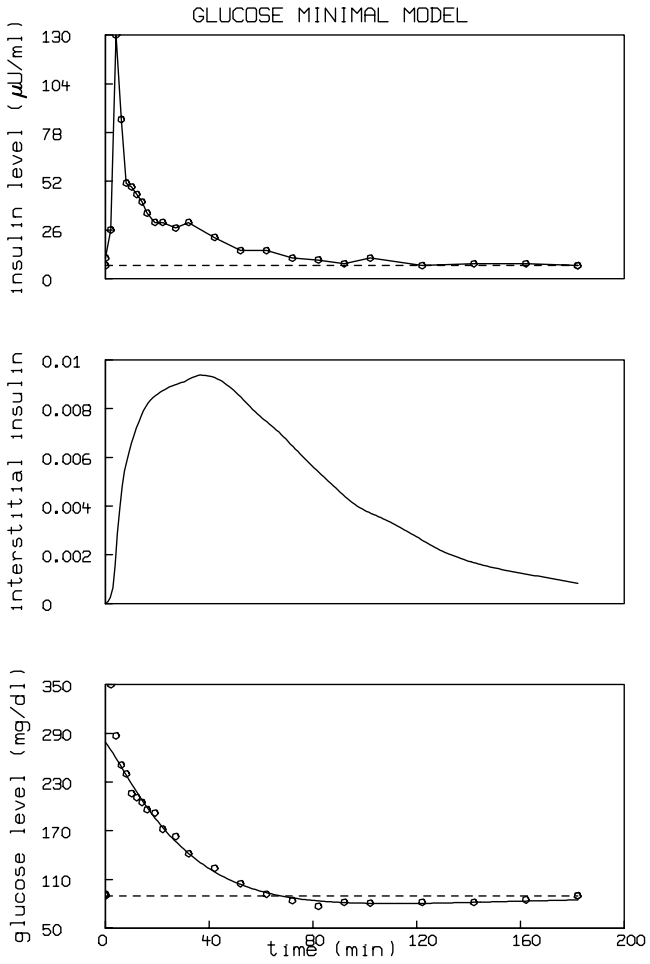
```
"file: g.do = glucose minimal model"
"-------------------"
"get data consisting of time values in column 1, plasma glucose"
"levels in column 2, and plasma insulin levels in column 3. Set"
"n to the number of time values. Set gdat to the (time,glucose level)"
"ordered pairs. Set idat to the (time,insulin level) ordered pairs."
data = read(test,50,3);
m = nrows(data);
gdat = data col (1,2);
idat = data col (1,3);

"define the glucose minimal model:"
" t is time"
" g(t) is plasma glucose level"
" i(t) is plasma insulin level, empirically-defined by interpolation in idat"
```

```
" x(t) is interstitial insulin"
" gb is basal (180 min) plasma glucose level"
" ib is basal (180 min) plasma insulin level"
fct g't(t) = -(p1+x(t))*g(t)+p1*gb
fct x't(t) = -p2*x(t)+p3*(i(t)-ib)
fct i(t) = lookup(idat,t)
init g(0) = g0
init x(0) = 0.0
gb = gdat(m,2)
ib = idat(m,2)

"define weights for glucose level data, censoring data up to time t = 8"
fct wf(i) = if gdat(i,1) < 8 then 0 else if gdat(i,1) = 8 then 10 else 1
wts = wf on 1:m

"define constraints for p1, p2, p3, and g0"
constraints q = {p1>0,p2>0,p3>0,g0>0}

"give initial estimates for parameters p1,p2,p3, and g0"
p1 = .03082; p2 = .02093; p3 = .00001062; g0 = 287;

"fit the model to the weighted data with defined constraints"
fit (p1,p2,p3,g0), g to gdat with weight wts constraints q

type "glucose effectiveness",p1
type "insulin sensitivity",p3/p2

"draw 3 graphs:1-plasma insulin level function and data vs time"
"              2-interstitial insulin vs time"
"              3-fitted glucose level function and data vs time"
"horizontal dashed lines show basal levels"
top title "GLUCOSE MINIMAL MODEL"
draw idat pt circle ptsize .01
draw shape(2,2,list(idat(1,1),ib,idat(m,1),ib)) lt dashed pt circle ptsize .01
left title "     insulin level ('15Tm'RU/ml)"
delete w.xaxis
frame .25 to .75, .667 to 1
w1 = w

draw points(x,gdat(1,1):gdat(m,1)!200)
left title "interstitial insulin"
delete w.xaxis
```

```
frame .25 to .75, .334 to .666
w2 = w

draw points(g,gdat(1,1):gdat(m,1)!200)
draw gdat lt none pt circle ptsize .01
draw shape(2,2,list(gdat(1,1),gb,gdat(m,1),gb)) lt dashed pt circle ptsize .01
left title "glucose level (mg/dl)"
bottom title "time v(min)"
frame .25 to .75, 0 to .333
view

save idat,gdat,ib,gb,g0,m in tmp.sav
```

The MLAB log-file and picture generated by executing the do-file `g.do` follow:

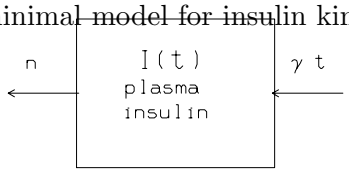
```
* do "g.do"
final parameter values
      value               error                dependency    parameter
  0.02649256302       0.01367779755          0.9976038298   P1
  0.02543609572         0.029223424          0.9932825935   P2
 1.281692067e-05      1.516217536e-05          0.9978008452   P3
    279.1123014         15.38803832          0.9901777503   G0
5 iterations
CONVERGED
best weighted sum of squares = 4.237885e+02
weighted root mean square error = 4.603197e+00
weighted deviation fraction = 1.469631e-02
R squared = 6.905055e-01
no active constraints

     glucose effectiveness
  P1 = .026492563

     insulin sensitivity
    = 5.03887106E-4

creating save file: TMP.SAV
```


GLUCOSE MINIMAL MODEL
insulin level (μU/ml)
130
104
78
52
26
0
interstitial insulin
0.01
0.008
0.006
0.004
0.002
0
glucose level (mg/dl)
350
290
230
170
110
50
0
40
80
120
160
200
time (min)

The insulin sensitivity, S_I, for this data set is estimated as 5.039×10^{-4} $\text{min}^{-1} \cdot (\mu\text{U/ml})^{-1}$ which is within the normal range reported in reference 2: 2.1 to 18.2 $\times 10^{-4}$ $\text{min}^{-1} \cdot (\mu\text{U/ml})^{-1}$. The glucose utilization, S_G, for this data set is estimated as 0.0265 min^{-1}, which is also within the normal range reported in reference 2: 0.0026 to 0.039 min^{-1}.

Next we examine the minimal model for insulin kinetics.

The following diagram summarizes the minimal model for insulin kinetics:



The course of plasma glucose in time, $G(t)$, is given by linear interpolation of the time-glucose values listed in `TEST.DAT`. Insulin enters the plasma insulin compartment at a rate proportional to the product of time and the concentration of glucose above a threshold amount. Here, time is the interval, in minutes, from the glucose injection. If the plasma glucose level drops below the threshold amount, the rate of insulin entering the plasma compartment is zero. Insulin is cleared from the plasma compartment at a rate proportional to the amount of insulin in the plasma compartment.

The minimal model for insulin kinetics is given by the equation:

$$\frac{dI(t)}{dt} = \begin{cases} -n \cdot I(t) + \gamma \cdot (G(t) - h) \cdot t, & \text{if } G(t) > h \\ -n \cdot I(t), & \text{otherwise.} \end{cases}$$

with $I(0) = I_0$. n is the insulin clearance fraction, h is roughly the basal glucose plasma level, and γ is a measure of the secondary pancreatic response to glucose. The *first phase pancreatic responsivity* is defined as $\phi_1 = \frac{I_{max} - I_b}{n \cdot (G_0 - G_b)}$ where I_{max} is the maximum insulin response. The *second phase pancreatic responsivity* is defined as $\phi_2 = \gamma \times 10^4$.

The following MLAB commands in the do-file `gi.do` estimate values for the parameters n, γ, h, and I_0 given the time course of plasma glucose. The values of the parameters found minimize (in the weighted least squares sense) the difference between the measured time course of plasma insulin and the parameter-dependent solution to the insulin minimal model differential equations.

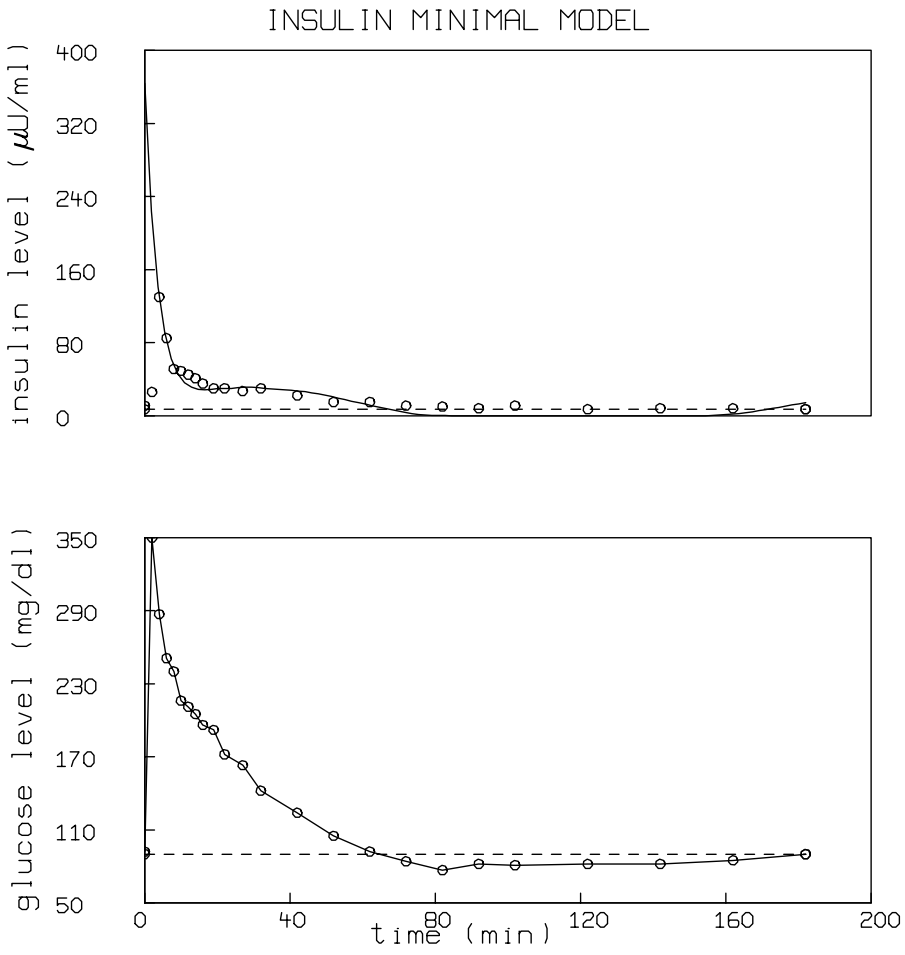
```
"file: gi.do = insulin minimal model"
reset
```

```
"read insulin, glucose levels from temporary save file"
use tmp.sav

"define the insulin minimal model"
fct i't(t) = -n*i+gama*(if g(t) < h then 0 else (g(t)-h))*t
fct g(t) = lookup(gdat,t)
init i(0) = i0

"determine weights for insulin data"
fct wf(i) = if idat(i,1) < 3 then 0 else if idat(i,1) <= 8 then 10 else 1
wts = wf on 1:m

"define a constraint set for all the parameters"
constraints q1 = {n > 0, gama > 0, h > 0, i0 > 0}

"provide initial guesses for the paramters"
n = .3; gama = .003349; h = 89.50; i0 = 410.4

"fit the insulin minimal model to the insulin data"
DISASTERSW = -1
fit (n,gama,h,i0), i to idat with weight wts constraints q1

type "phase 1 pancreas responsivity",(maxv(idat)-ib)/(n*(g0-gb))
type "phase 2 pancreas responsivity",10000*gama

"draw 2 graphs:1-fitted plasma insulin level and data vs time"
"              2-glucose plasma level data vs time"
top title "INSULIN MINIMAL MODEL"
draw idat lt none pt circle ptsize .01
draw points(i,idat(1,1):idat(m,1)!100)
draw shape(2,2,list(idat(1,1),ib,idat(m,1),ib)) lt dashed pt circle ptsize .01
left title "     insulin level ('15Tm'RU/ml)"
delete w.xaxis
frame .15 to .85, .5 to 1
w1 = w

draw gdat pt circle ptsize .01
draw shape(2,2,list(gdat(1,1),gb,gdat(m,1),gb)) lt dashed pt circle ptsize .01
left title "glucose level (mg/dl)"
bottom title "time v(min)"
frame .15 to .85, 0 to .5
```

```
view
```

The log-file and picture generated by executing the do-file `gi.do` follow:

```
* do "gi.do"
Using: M,G0,GB,IB,GDAT,IDAT
final parameter values
      value               error                 dependency    parameter
   0.2673230345       0.01603033225          0.9611152205   N
 0.004074459794      0.0006378976592          0.7211259427   GAMA
    83.74403736         2.329398193          0.2073681454   H
     363.666326          26.5919455          0.9471979611   I0
4 iterations
CONVERGED
best weighted sum of squares = 1.292680e+03
weighted root mean square error = 8.039529e+00
weighted deviation fraction = 2.772192e-02
no active constraints

     phase 1 pancreas responsivity
    = 3.46163989

     phase 2 pancreas responsivity
    = 40.7445979
```


INSULIN MINIMAL MODEL
insulin level (μU/ml)
400
320
240
160
80
0
glucose level (mg/dl)
350
290
230
170
110
50
0
40
80
120
160
200
time (min)

The phase 1 pancreas responsivity, ϕ_1, is estimated as 3.462 $\min \cdot (\mu\text{U/ml})(\text{mg/dl})^{-1}$ for this data set. This is within the normal range for ϕ_1 reported in reference 3 as 2.0 to 4.0. The phase 2 pancreas responsivity, ϕ_2, is estimated as 40.745 $\min^{-2} \cdot (\mu\text{U/ml})(\text{mg/dl})^{-1}$. This is slightly higher than the normal range for ϕ_2 reported in reference 3 as 20 to 35.

Note that when performing the least squares minimization between the solution of the insulin minimal model equation and the measured plasma insulin levels, relative weights of 0, 10, and 1, were assigned to the plasma insulin levels so that less reliable values would not adversely effect the estimation of parameters and more reliable values would be more heavily weighted. Using appropriate weights is generally important in fitting forms of the minimal model. While there are many ways of weighing data, this particular method was suggested by Walton (reference 6).

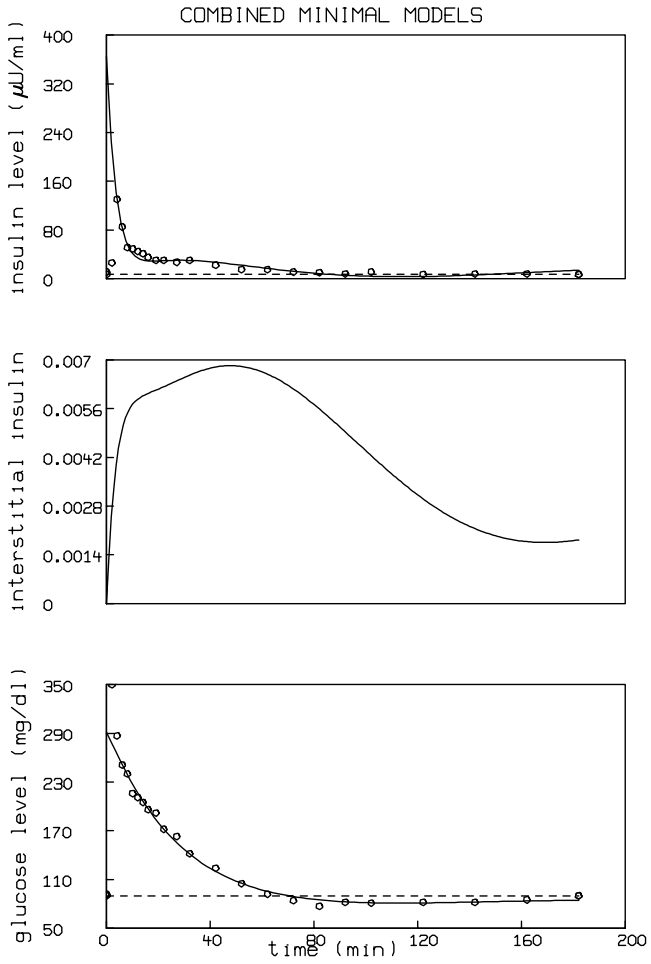
The glucose minimal model provides differential equations for the plasma glucose and interstitial insulin levels. The insulin minimal model provides a differential equation for the plasma insulin level. It is possible to combine all three differential equations into one model. This is demonstrated in the following do-file, `ggi.do`:

```
"file: ggi.do - combined glucose and insulin minimal models"
"read insulin, glucose levels from temporary save file"
use tmp.sav

"define the combined glucose-insulin minimal model"
fct i't(t) = -n*i+gama*(if g < h then 0 else (g-h))*t
fct g't(t) = -(p1+x)*g+p1*gb
fct x't(t) = -p2*x+p3*(i-ib)
init i(0) = i0
init g(0) = g0
init x(0) = 0

"determine weights for glucose level data, censoring data up to time t = 8"
fct wg(i) = if gdat(i,1) < 8 then 0 else if gdat(i,1) = 8 then 10 else 1
wgs = wg on 1:m

"determine weights for insulin level data"
fct wi(i) = if idat(i,1) < 3 then 0 else if idat(i,1) <= 8 then 10 else 1
wis = wi on 1:m

"define a constraint set for all of the parameters"
constraints q1 = {n>0,gama>0,h>0,p1>0,p2>0,p3>0,i0>0,g0>0}

"provide initial guesses for the parameters"
n = .3; gama = .003349; h = 89.50; p1 = .03082; p2 = .02093
p3 = .00001062; i0 = 403.4; g0 = 287
```

```
"fit both of the models to the data"
disastersw = -1
fit (n,gama,h,p1,p2,p3,i0,g0), i to idat with weight wis, \
   g to gdat with weight wgs, constraints q1

type "glucose effectiveness",p1
type "insulin sensitivity",p3/p2
type "phase 1 pancreas responsivity",(maxv(idat)-ib)/(n*(g0-gb))
type "phase 2 pancreas responsivity",10000*gama
```

Here is the resulting MLAB log-file and picture:

```
final parameter values
      value               error                dependency    parameter
   0.2658844452       0.01153178897          0.9621356273   N
 0.003911687955      0.0004693022146          0.7457248671   GAMA
    79.03532257         2.421480638          0.4908060854   H
  0.03168360775      0.004806175221          0.9712465296   P1
   0.0123362991      0.006558350386           0.888606309   P2
 4.891692162e-06      1.923309135e-06          0.9627481277   P3
    364.8353065         18.88965714          0.9496151759   I0
    291.2242018          5.82052435          0.8836723395   G0
5 iterations
CONVERGED
best weighted sum of squares = 1.278572e+03
weighted root mean square error = 5.653697e+00
weighted deviation fraction = 1.545963e-02
R squared = 3.943754e-01
no active constraints

     glucose effectiveness
  P1 = 3.16836078E-2

     insulin sensitivity
    = 3.96528337E-4

     phase 1 pancreas responsivity
    = 3.27088221

     phase 2 pancreas responsivity
    = 39.1168796
```


COMBINED MINIMAL MODELS
insulin level (μU/ml)
400
320
240
160
80
0
interstitial insulin
0.007
0.0056
0.0042
0.0028
0.0014
0
glucose level (mg/dl)
350
290
230
170
110
50
0
40
80
120
160
200
time (min)

These results show that, as is common, there is not a unique set of parameters that characterize a FSIGT test data set. The combined minimal model is seen to generate slightly lower values for glucose effectiveness and insulin sensitivity than the glucose minimal model, and slightly higher values for phase 1 and 2 pancreas responsivity than the insulin minimal model.

There are various devices that could be explored in order to improve the family of models studied here. First, these models employed the MLAB operator `LOOKUP` to linearly interpolate glucose and insulin plasma time course data. Alternatively, the MLAB operator `SMOOTHSPLINE` could be used to provide glucose and insulin plasma time course curves that are not only continuous, but also have continous first and second derivatives.

Second, several authors have augmented the insulin minimum model to account for plasma levels of C-peptide (see references 7-9). It is a straightforward exercise to implement the C-Peptide minimal model using MLAB.

This paper has shown how MLAB can be used to calculate diagnostically important metabolic indices which arise in the glucose and insulin minimal models from frequently-sampled intravenous glucose tolerance test data. The MLAB program is generally an ideal tool for the study of compartmental models. You can contact Civilized Software for further examples in neurophysiology and pharmacology.

References:

1. Saad, M.F., Anderson, R.L., Laws, A., Watanabe, R.M., Kades, W.W., Chen, Y.-D.I., Sands, R.E., Pei, D., Savage, P.J., Bergman, R.N., *A Comparison Between the Minimal Model and the Glucose Clamp in the Assessment of Insulin Sensitivity Across the Spectrum of Glucose Tolerance* **Diabetes** 43 (1994) pp. 1114-21.

2. Steil, G.M., Volund, A., Kahn, S.E., Bergman, R.N.: *Reduced Sample Number for Calculation of Insulin Sensitivity and Glucose Effectiveness From the Minimal Model*, **Diabetes** 42 (1993) pp. 250-256.

3. Pacini, G., Bergman, R.N.: *MINMOD: A Computer Program to Calculate Insulin Sensitivity and Pancreatic Responsivity from the Frequently Sampled Intravenous Glucose Tolerance Test*, **Computer Methods and Programs in Biomedicine** 23 (1986) pp. 113-22.

4. Toffolo, G., Bergman, R.N., Finegood, D.T., Bowden, C.R., Cobelli, C.: *Quantitative Estimation of Beta Cell Sensitivity to Glucose in the Intact Organism–A Minimal Model of Insulin Kinetics in the Dog*, **Diabetes** 29 (1980) pp. 979-990.

5. Bergman, R.N., Ider, Y.Z., Bowden, C.R., Cobelli, C.: *Quantitative Estimation of Insulin Sensitivity*, **American Journal of Physiology** 236 (1979) pp. E667-77.

6. Walton, C.W., Wynn Department of Metabolic Medicine, Imperial College of Science, Technology, and Medicine, *private communication*, April 1998.

7. Watanabe, R.M., Volund, A., Roy, S., Bergman, R.N.: *Prehepatic B-Cell Secretion During the Intravenous Glucose Tolerance Test in Humans: Application of a Combined Model of Insulin and C-Peptide Kinetics*, **Journal of Clinical Endocrinology and Metabolism** 69 (1988) pp. 790-797.

8. Cobelli, C., Pacini, G.: *Insulin Secretion and Hepatic Extraction in Humans by Minimal Modeling of C-Peptide and Insulin Kinetics*, **Diabetes** 37 (1988) 223-231.

9. Volund, A., Polonsky, K.S., Bergman, R.N.: *Calculated Pattern of Intraportal Insulin Appearance Without Independent Assessment of C-Peptide Kinetics*, **Diabetes** 36 (1987) 1195-1202.

Index