XLMATH.XLL

A Dynamic Link Library for Microsoft Excel 4.0
Version 2.1

XLMATH is a standalone dynamic link library (XLL) for Microsoft Excel 4.0 and contains custom functions and commands for diagonalizing a real symmetric matrix, curve fitting functions including nth order polynomial fitting, cubic spline functions fitting and data smoothing via Savitsky Golay or by weighted averages. The functions and commands are described and illustrated in the workbook XLMATH.XLW.

Files

The files included in the archive are:

Executable files:

README.1ST	Instructions for loading XLMATH and associated files
XLMATH.XLL	The executable XLL
XLMATH.HLP	The help file - must be in Excel subdirectory
XLMATH.XLW	Demonstration workbook: custom functions and dialog commands
SOURCE.ZIP	Selected source files

Excel Usage

XLMATH contains both custom command and function versions most of its tools. The command versions can be invoked from the Xlmath menu and are easily understood and used. Most of the custom functions added by XLMATH.XLL return arrays of data to Excel. If you are not familiar with Excel's array formula usage, please read the section on array formulae in the Excel User's Guide . After opening XLMATH.XLL, the custom functions can be found under the category "XLMath Add-In". Always open the XLL first using the Excel File Open... command.

Commands or Functions?

Curve fitting and diagonalizing can be performed either by invoking a command and entering the approriate ranges and values into dialog boxes, or by entering custom function definitions found under the category Xlmath-Add-In. Commands are easier to use but if you have frequently changing input data, you may prefer the custom function versions.

Column or Row Ranges?

With the exception of Diagonalize, the custom functions in XLMATH all require the input of one dimensional data vectors. These data vectors can be entered into the sheet as either column vectors (N x 1) or row vectors (1 x N) and XLMATH will recognize their shape. With the exception of diagonalize and CubicSplines, the array formulae should be entered into the same type of vector. For example, in the smoothing routines, if the data is a column vector (Nx1), then the array formula should also be entered into a column vector (Nx1). If the data is a row vector (1xN), then the array formulae should be

entered in a row vector (1xN). The curvefitting routine PolyCurvefit will identify the input as either a column or row vector. If the input is in the form of a column vector (Nx1), then the array formulae must be entered in an (N x 3) array. If the input is in the form of a row vector, then the array formulae must be entered into a (3 x N) array. For CubicSplines, the input can be in the form of column or row vectors, but the array formulae must always be entered into a rectangular array of dimension (N x 4) where N is the number of data points. For CubicSplines, it is more convenient to enter the variables into column vectors (Nx1) and then enter the array function into an N x 4 array.

Custom Command and Function Descriptions

The following is a brief description of the custom functions added by XLMATH.XLL The user should consult the worksheet XLMATH.XLS for detailed usage of these functions.

Diagonalize(SymMat)

This function returns the eigenvectors and eigenvalues of a real symmetric matrix. The input must be a real symmetric matrix (square) but only the top half is used and hence needs to be defined. The values are returned in output which is an (N+1) x N array where the last row contains the eigenvalues. In the dialog boxes, *Input* refers to the input real symmetric matrix range and *Output* referes to the (N+1)xN output range.

PolyCurveFit(Xvar, Yvar, Order)

Polynomial curve fitting results in a single polynomial equation of order m which is the least squares approximation of the observed data.

$$y = C_0 + C_1 \times X + C_2 \times X^2 + C_3 \times X^3 \ldots + C_m \times X^m$$

Order is the order m of the fitting, i.e. 1 for a linear fit, 2 for a quadratic fit and etc. The order must be one less than the number of variables.
Xvar is a column or row vector of *N* independent variables (X).
Yvar is a column or row vector of *N* dependent variables (Y).
Xvar and *Yvar* must be both either row or column vectors. Using one as a row vector and the other as a column vector is not supported. It is recommended that both variables be entered as column vectors.
The output or the function PolyCurveFit must be entered into an N x 3 array if the input is in the form of a column vector or a 3 x N array if the input is in the form of a row vector. Assuming that both Xvar and Yvar are column vectors and the array formulae have been entered into a N x 3 array, then the first column of the return array contains the estimated Y values, the second row contains the residuals (differences between calculated and estimated y-values). The third column contains in the first (order + 1) rows, the polynomial coefficients. If fitted to 2nd order, the first three rows contain a0, a1, & a2. The following values are returned directly below the coefficients,
coefsig - a vector of dimension (order+1). Coefsig are the standard errors of coefficient estimates. The values are stored in the same order as the polynomial coefficients.
see - the standard error of the estimate
rsqrval - the r squared value - the sample correleation coefficient

cferror - returns 1 if the curve fit is singular, otherwise 0.

CubicSplines(Xvar, Yvar)

The CubicSplines() function fits a discrete set of cubic polynomial equations to a discrete set of data. Y-values may be interpolated for points between the original data points by applying the calculated cubic equations. The arguments to the function are;

Xvar - the N independent variables (X).

Yvar - the N dependent variables (Y).

Both variables can be entered as row or column vectors. However, the function returns and N x 4 array of coefficients and hence it is recommended that the variables be entered as column vectors.

CalcSpline(Xvar, Coef, X)

The function CaclSpline returns an interpolated y-value for the argument *X*. The argument *Xvar* is the column array of independent variables passed to CubicSplines() above. The argument *Coef* is the N x 4 array of cubic spline coefficients returned by CubicSplines() above. There is no dialog box version of this custom function.

SmoothSG(Data, SmoothNum, DerivNam)

This function performs a Savitsky - Golay smoothing and differentiation of data (see Savitsky, A. and Golay, J., Analytical Chemistry 36 (1964), p. 1627). The arguments are as follows;

Data - a column or row vector of the data to be smoothed. If the data is a column vector, then the function must be entered into a column vector and *vice versa* if the data is a row vector.

SmoothNum - holds the integer degree of smoothing

1 = 5 point smooth

2 = 7 point smooth

3 = 9 point smooth

4 = 11 point smooth

5 = 13 point smooth

DerivNum - holds the integer derivative degree

0 = smooth data only

1 = first derivative

2 = second derivative

The function returns a column or row vector of smoothed data.

SmoothWT(Data, Weights, Divisor)

This function is used to reduce the noise in a sample. The technique uses convolution where each data point is recalculated as a weighted average of its original value and surrounding data points. The arguments are as follows;

Data - the data points to be smoothed. The data may be entered into a column or row vector.

Weights - the weights used in the convolution process

The function returns a column or row vector of smoothed data points.

MODensity

The charge and bond order matrix in simple Huckel calculations is defined as a matrix

multiplication of *C*(T) x *Occ* x *C* where *C* is the matrix of coefficients and *Occ* is a matrix of occupancies. Although a multiplication of this form can be done in several steps with Excel 4.0, it is easier to use the custom function and calculate all of the charges and bond orders via MODensity. While it would be a simple task to create a command for MODensity, the author left it undone in the desire to force students to use the array method of function entry.

CustomFit

In data fitting, one typically has m data values $y_1, y_2, \ldots y_m$ which have been sampled for values $x_1, x_2, \ldots x_m$ of some independent variable x. It is then desired to fit a function f(x,p) which has n adjustable parameters, to be chosen so that the function best fits the data. The residuals are given by

$r_i(p) = f(x_i,p)-y_i \qquad i = 1,2,...m$

and a least squares solution is sought by minimizing S(p) which is the sum of the squares of the residuals . CustomFit uses the Levenberg-Marquardt method to solve for the solution to S(p) by stepping towards the solution via a sequence of corrections based on the solution to the equation

$(\mathbf{G} + a\mathbf{I})\mathbf{s} = -\mathbf{g}$

where **G** is a matrix of second derivatives, **g** is a matrix of first derivatives, **I** is a unit matrix and a is a damping factor. The Levenburg-Marquardt method is also characterized by the scaling of the matrix **G** by replacing it with a scaled matrix **C** where

$Cij = Gij/((Gii)^{1/2}.(Gjj)^{1/2})$ or $\mathbf{C} = \mathbf{D}^{-1}\mathbf{G}\mathbf{D}^{-1}$.

The solution for the step **s** is then given by

$s = -\mathbf{D}^{-1}(\mathbf{G} + a\mathbf{I})^{-1}\mathbf{D}^{-1}\mathbf{g}$

where $\mathbf{D}^{-1}$ is a diagonal matrix and $(\mathbf{G} + a\mathbf{I})^{-1}$ is determined from the eigenvalue decomposition as $\mathbf{C} = \mathbf{A}.\mathbf{L}.\mathbf{A}^T$ where L is a diagonal matrix of eigenvalues and A is a matrix of eigenvectors. If none of the eigenvalues are zero, then the inverse of **C** exists and is defined as $\mathbf{C}^{-1} = A.\mathbf{L}^{-1}\mathbf{A}^T$.

Pragmatically, iterations must also be halted when no significant change is obtained in two successive choices of the parameters. This condition will be encountered when the model nonlinear equation has a parameter redundancy. For reference see Marquardt, D.M. "An algorithm for Least-Squares Estimation of Nonlinear Parameters", J. Soc. Indust. Appl. Math., 11, 431-441, 1963

As a general rule, "best-fitting" is obtained only when

1. termination occurs before the maximum number of iterations (50)
2. S is minimal
3. there is no parameter redundancy indicated by the eigenvalues near zero
4. scale vectors are finite

To use CustomFit, the user must supply a user written macro as illustrated below for the function y = ax/(b+x).

```
                    USERMACRO
                    =RESULT(1)
                    =ARGUMENT("kIndex",1)
                    =ARGUMENT("rgP",64)
                    =ARGUMENT("kX",1)
                    =SET.VALUE(D20:E20,rgP)
Ret2                =D20*nX/(E20+kX)
                    =RETURN(Ret2)
```

The macro takes three arguments, kIndex - the index of the k'th data point; rgP - the array of parameters; kX - the value of the k'th independent variable. The macro must return a value for the calculated y (dependent variable). In addition, the name "CustomFitMacro" must be defined as "USERMACRO" in the worksheet. In this example, the user would select the define names command and define the name "CustomFitMacro" as "=MACRO1.XLM!USERMACRO". The name USERMACRO must also be a defined name in the macro sheet. The index k is not normally required but may be used if direct indexing of the dependent variable is required or preferred.

An Alternative

Most of what is accomplished by using CustomFit can also be done more simply by using the native Excel Solver command. The net effect of CustomFit is to minimize the sum of the squares of the deviations between experimental and calculated data points. Solver is quite capable of performing this minimization if you provide it with the array of adjustable parameters and the location of the sum of squares. This alternative is explored fully in the worksheet XLMCFIT.XLS and the user is referred to that worksheet for further details.

Development Notes

Xlmath v1.0

Xlmath v1.0 is described by the author in an article published in the Journal of Chemical Education (2nd quarter of 1993). The intent of the author in this article and in Xlmath was to make persons aware of the ease with which DLL's could be written for Excel and to convince educators and others to attempt to write their own DLL's and to abandon stand-alone programs. Since the writing of the paper and the development of v1.0, Microsoft published the Excel API (Microsoft Press, ISBN# 1-55615-521-2). The publishing of the API made it even easier to write standalone DLL's and made it possible to interface the custom functions and commands without the need for any macro language. Since the publication of the API made v1.0 obsolete, the author decided to revise v1.0 and re-write v2.0 to conform to the API.

Differences between v2.1 and v2.0

1. The Xlmath menu did not appear if you were using a a language variant of Excel that did not have the command "Help". This has been corrected and the Xlmath menu will appear regardless of language. Hpoefully, it will also run on all language variants of Excel.

2. There was a memory leak in the routines performing SG and WT smoothing. If you used these routines repeatedly, in previous versions, each use would increase the amount of memory used by Xlmath. You likely have not noticed this unless you inspected the memory usage with Heapwalker.
3. The behaviour of the memory allocation schemes (malloc() etc) in Microsoft C/C++ version 7.0 has become more compatible with Windows 3.1. Hence the memory allocation program SMRTHEAP.DLL has been eliminated and _fmalloc() and _ffree() substituted where required (see MS Developers Network: Allocating Memory the Old-Fashioned Way: _fmalloc and Applications for Windows[TM], 1992, Dale Rogerson, Microsoft Corporation). Xlmath is now a large model DLL. It is still easier to debug your program with SMARTHEAP and hence I have left the SMARTHEAP statements in the code but they are now invoked only when specified in the makefile.

Differences between v2.0 and v1.0

There are two fundamental difference between v1.0 and v2.0. XLMATH v2.0 includes both custom functions and commands. The commands are invoked by selecting the menu *Xlmath* and completing the dialog box prompts. XLMATH v2.0 also uses the Excel API to both register the custom functions and commands and to run the dialog box routines. The Excel API eliminates the need for a macro sheet. Since Excel v4.0 has its own version of Frequency(), the XLMATH version has been deleted.

Source Code

The source code included in XLMATH v2.1 includes the source required to write your own XLL. It does **not** include all of the source code required to re-compile Xlmath 2.1. For personal reasons, the author will not release the source code for the actual operational functions. You don't need it since you have a fully executable version of these functions. In addition, the source code for the curve fitting and data smoothing routines in v2.1 are a modified form of the Science & Engineering Tools routines sold by Quinn-Curtis. This source is copyrighted by and belongs to Quinn-Curtis but may be purchased from them at 35 Highland Circle, Needham, MA 02194 USA.

Memory Management

For those of you who have read my article in J.Chem.Ed., a thousand apologies. Memory management has been a most difficult aspect of Windows(TM) and all of the difficulty arises from the "real" mode requirements that pre-date Windows 3.0. Finally, I have got it straight and the straight answer is that memory management is very simple. Just do it as you did before. The following is a quote from an article in the Microsoft Devoper's Network CD by Dale Rogerson (The C/C++ Compiler Learns New Tricks)

1. Use the LARGE model
2. Use malloc().

Now what could be more simple? The source code in Xlmath contains a lot of casts to ensure that the pointers are FAR pointers. You don't need these casts because the large model compiler automatically casts all data to FAR. It was just too much work to remove

them so as the saying goes "do as I say and not as I did". Be particularly careful not to use the Windows definitions of near pointers such as NPSTR. These definitions use the keyword "__near" in their definition and the compiler cannot convert this to a __far pointer. Good luck with your own DLL's.

Freeware

Xlmath is freeware. This means that you can freely copy it, use it, modify it, and give copies to all your friends (as long you give them all of the *unmodified* files that you received). However, if you wish to modify and/or use the source code included, please add a note indicating that portions of your program are copyrighted by Roy Kari. The best place to add this note is in your About Box.

THE SOFTWARE AUTHOR (ROY KARI) DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE PRODUCT. SHOULD THE PROGRAM PROVE DEFECTIVE, THE USER ASSUMES THE RISK OF PAYING THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION AND ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES. IN NO EVENT WILL THE AUTHOR BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING WITHOUT LIMITATION DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION AND THE LIKE) ARISING OUT OF THE USE OR THE INABILITY TO USE THIS PRODUCT EVEN IF THE AUTHOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

If you do encounter problems with Xlmath, or if you think of a way to improve it feel free to contact me.

Although I don't want cash for Xlmath, I am interested in hearing from people who use it. To this end, please send a note via EMAIL or a fax to:

Roy Kari
Department of Chemistry & Biochemistry
Laurentian University
Sudbury, Ont.
Canada
P3E 2C6

voice: (705) 675-1151
fax: (705) 675-4844
Internet: "ROY@NICKEL.LAURENTIAN.CA"