XLMATH.DLL

A Dynamic Link Library for Microsoft Excel
Version 1.0

XLMATH is a DLL for Excel and contains custom functions for diagonalizing a real symmetric matrix, creating a frequency distribution, and curve fitting functions including nth order polynomial fitting, cubic spline functions fitting and data smoothing via Savitsky Golay or by weighted averages. While XLMATH.DLL may be used as it is, the archive file includes the source programs necessary to extend the DLL with additional functions or to create a new DLL for Excel.

Files

The files included in the archive are:

Executable files:

README	Instructions for executing the demo worksheet called XLMATH.XLS
XLMATH.DLL	The executable DLL
OPTIMEM.DLL	A memory management DLL (c) Applegate Software
XLMATH.XLS	A demonstration worksheet
XLMATH.XLM	Macros required to register the custom functions
POLYDE.XLC	A demo chart for polynomial fitting
SPLINEDE.XLC	A demo chart for spline function fitting
SMOOTHDE.XLC	A demo chart for data smoothing
XLMATH.XLW	A workspace file to load XLMATH.XLS and the associated macro and chart files

Source Code header files:

XLMATH.H, XLMINIT.H, XLMUTIL.H, and XLMCURVE.H
The file OPTIMEM.H is copyrighted by Applegate Software and is not included in the archive.

Source Code:

XLMINIT.C	Task and memory initialization routines
XLMUTIL.C	Memory allocation routines
XLMATH.C	Main DLL interface routines
XLMATH.DEF	Windows .DEF file
XLMATH.RC	Windows resource file

The file XLMCURVE.C is not included in the archive because the routines in the file are the copyright of Quinn-Curtis and have been extracted from their Science and Engineering Tools library.

Excel Usage

All of the custom functions added by XLMATH.DLL 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 (p279ff). The custom functions added by the DLL must be registered with Excel prior to usage in a worksheet. This means that the user must load the XLMATH.XLM macro sheet prior to attempting to use any of the custom functions. The macro sheet contains an auto open macro which initializes the DLL and registers the custom functions with Excel. The macro sheet also contains an auto close macro which unregisters the function. If the user closes the macro sheet prior to closing the worksheet, the custom functions added by the DLL will no longer be available. Always open the macro sheet first, and close the macro sheet last.

Custom Function Descriptions

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

InitXLMath()

This function is used to initialize XLMATH.DLL and must always be called in the auto open macro prior to using any of the other custom functions. For sample usage see XLMATH.XLM. This function should only be used in the auto open macro and should not be made visible to the worksheet user.

ExitXLMath()

This function is used to exit XLMATH.DLL and must always be called in the auto close macro. Failure to do so will cause error messages from both XLMATH and OPTIMEM. As with InitXLMath(), this function should not be visible to the worksheet user.

Frequency(Values, Intervals,)

The Frequency function returns a frequency distribution table. The function behaves in a manner identical to the found in Quattro Pro. *Values* must be a column of values for which the user wishes to calculate a frequency table. The column range, *Intervals*, defines the intervals of the values. The resulting frequency distribution table can be found by entering the *Frequency()* array formula in a column range which has one row more than the number of interval values. The first frequency value contains the number of values which are less than or equal to the first interval. The second and subsequent frequency

values contain the number of values greater than the lower interval and less than or equal to the upper interval. The final frequency value indicates the number of values that exceed the last interval value.

Diagonalize(SymMat)
This function returns the eigenvectors and eigenvalues of a real symmetric matrix. The values are returned in an (N+1) x N array where the last row conatins the eigenvalues.

PolyCurveFit(IndVar, DepVar, Order)
Polynomial curve fitting results in a single polynomial equation of order m which is the least squares approximation of the observed data.
$y = C0 + C1 \times X + C2 \times X^2 + C3 \times X^3 \ldots + Cm \times X^m$
In this function, *IndVar* is a column range of independent variables (X), *DepVar* is a column range of dependent variables (Y) and Order is the order of polynomial (m) fittted to the dependent vaiables. The function PolyCurveFit must be entered into an N x 3 array where N is the number of independent variables. 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 columns 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). It returns the standard errors of coefficient estimates. The values are stored in the same order as the polynoliam 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.

CubicSplines(IndVar, DepVar)
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;
IndVar - the N independent variables (X)
DepVar - the N dependent variables (Y)
The function returns and N x 4 array of coefficients.

CalcSpline(IndVar, Coef, X)
The function CaclSpline returns an interpolated y-value for the argument X. The argument IndVar 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.

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 range of the data to be smoothed
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 range 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 argumants are as follows;
Data - the data points to be smoothed
Weights - the weights used in the convolution process
Divisor - the normalization factor for the weights
The function returns a column range of smoothed data points.

Development Notes

XLMATH can serve as a both black-box DLL and as a template for writing a custom DLL for Excel. The creation of a DLL for Excel is not a simple task if one has to begin from scratch but by using the routines in XLMINIT and XLMUTIL, the creation of a new DLL should be greatly simplified. The following is a brief discussion of the salient points regarding the creation of DLL's.

One of the main differences between dynamic link libraries (DLL's) and static libraries is
that the function code in a DLL is loaded dynamically during run time. For Microsoft
Excel, the DLL's can be written in a language such as C or FORTRAN, compiled and
linked into a DLL, and then registered within an Excel macro and subsequently used as a

custom function within an accompanying Excel worksheet. Two major concerns in writing a DLL are memory management and the interface between Excel and the DLL library.

Memory Management

Memory management in a Windows program is particularly difficult because much of the literature on memory management is relevant to real mode windows which will no longer be supported in Windows 3.1. In real mode, the application had to be prepared to run out of memory any time that the application allocated or locked a memory block. This meant extensive operation checking and error processing for out of memory errors. Since memory was at a premium, the real mode application couldn't leave memory blocks locked for longer than absolutely necessary. In standard or enhanced modes, Windows uses hardware memory mapping which allows the application to allocate and lock memory blocks at initialization and unlock and free them only when no longer required. It is quite obvious that new applications will be developed only for standard and enhanced modes.

Although memory allocations in standard and enhanced mode applications can be written more easily and straightforwardly, small or medium model programs are still recommended. DLL's in particular should be written in the small model since they are faster and easier to maintain. This then necessiates the use of the _far keyword to access data on the global heap. The need to use the _far keyword is particularly important in a DLL . For example, in an Excel DLL, the transfer of array data between Excel and the DLL must be via _far pointers. Also, arrays created in a DLL must be dynamically allocated and due to the 64k limit on local memory, they must invariably be allocated in global memory. Finally, because the DLL does not have its own stack but instead uses the stack of the calling program, automatic variables, when passed by reference, must be passed as _far pointers.

Although the use of the _far keyword is at times bothersome, the main difficulty in using the global heap is the Windows system-wide limit of 8192 selectors for global memory. Each global allocation uses one selector which means that the allocation of smaller global memory arrays is both inefficient and easily leads to a depletion of selectors before a depletion of memory. Since this limit is inherent in the design of the Intel 80x86 chip, it is unlikely to change in the near future. The only practical solution to the limited number of selectors is to use an internal memory manager which allocates global memory within a single global segment, thus using a single selector. Although memory allocation routines are described in a number of standard programming texts (see Advanced C by Paul & Gail Anderson), creating such an allocator for Windows is a daunting task. In the author's opinion, the only reasonable solution to the problem of memory management is to purchase a commercial memory management library such as

OPTIMEM. This library manages the allocation of memory blocks within a segment allowing for efficient allocation of small blocks of memory. This library also allows the user to use shared memory which further reduces the memory demand upon Windows. The greatest benefit of such a memory management library is that it lets the user concentrate on the task at hand which is to write a useful custom function in the minimum amount of time.

If you really don't want to purchase OPTIMEM, then there is a work around although it is not recommended. When compiling XLMATH, define the flag SUB_OPTIMEM by adding the term /DSUB_OPTIMEM to the compiler flags. This will delete all references to the OPTIMEM library and include substitute routines. All memory will be allocated in the global heap and kept locked for the duration of its use. If you want to try your hand at building a memory suballocation module, then an article by Paul Yao on subsegment allocation in the *Microsoft Sytems Journall* 6, January 1991, p75, is required reading. In addition, the BOOK/DISK SET of Martin Heller's text (7) contains a sub-allocator called **smalloc.**

Interface

Excel is able to interface with a DLL by passing to the DLL and returning from the DLL a number of different data types. A very useful data type is K which is a floating-point array, passed by reference. The C data structure describing this data type is commonly defined as

```
typedef struct fp
{
        WORD wRows;
        WORD wCols;
        double Data[1];
} FP;
```

The first word of the structure contains the number of rows. The second word contains the number of columns. This is followed by rows x columns floating point numbers. Since the Excel array exists in a global memory segment, reference to the array is made with a far pointer defined by

```
typedef FP FAR     *LPFP
```

Accessing a one dimensional array of type K is relatively straightforward and is done by
defining a far pointer to a FP structure. For example, the Frequency function is
defined as

```
LPFP PASCAL FAR  Frequency(LPFP lpValues, LPFP lpIntervals)
```

In this function, both far pointers lpValues and lpIntervals refer to type K arrays passed to the function from Excel. Individual elements of the array can be accessed via statements such as lpValues->Data[wIndex] where wIndex is an array index defined in a DO loop. DLL computed values are passed back to Excel in an internally allocated type K array. The allocation and initialization of the array is performed in the function InitRetBuffer(). By convention within XLMATH DLL, the array returned to Excel is always called XLKInt (for internal).

Accessing a dynamically allocated two dimensional array in C with standard array
subscripts such as a[i][j] is not a simple process. However, the method of accessing these
arrays is well described in a number of texts on the C language . The method involves
the creation of pointers to each row of the matrix Data and then subsequently accessing the array with the array of pointers. Since the Excel allocated array is in global memory, the access must be via far pointers to type double and the type LPLPREAL is defined as
typedef double FAR * FAR *LPLPREAL

Prior to using the DLL in an Excel worksheet, the DLL functions must be registered in an Excel macro sheet. This is usually done in the Excel auto_open macro. Prior to closing, the DLL functions must be unregistered in an Excel macro sheet. This is normally done in the Excel auto_close macro. The details of this registration and unregistration procedure are illustrated in the macro sheet XLMATH.XLM.

Because Windows allows the execution of multiple copies of Excel, with each copy given a unique task handle, the DLL application must choose to either support only one task or multiple tasks. If the DLL is written to support multiple tasks, then the DLL must pass back to the task (Excel) a pointer to a unique return buffer. The management of multiple tasks in XLMATH is accomplished by using a data structure which contains the task handle and the return buffer pointer for that task. The data structure used is;

```
typedef struct td
{
        HANDLE      hTask;
        LPVOID      lpReturnBuffer;
} TD;
```

Task management is performed by several routines in the module XLMUTIL.C.

After writting the above descrition on task management, the author engaged in a converstaion with several experts on Excel. The following is an edited version of the converstaion:

QUESTION:

```
> If you maintain only a single "return" array, then what happens
> if someone starts a second copy of excel and they both
> call the dll. I would have thought that you have to maintain
> a single "return" array for each task that calls your DLL.
> What's the real story?

An interesting observation, but I don't think that win3x's
"multitasking" gives rise to such worries. Since the multitasking
in windows is cooperative, so long as the DLL has control and doesn't
have provisions for giving it up, it's the only thing running, right? So
the question is whether Excel, after it has called a DLL, but before it
has copied the new value to its destination, allows another program
(e.g. instantiation of itself) to run (cooperates?). I doubt it does,
since that would be a really bad time to do so. Besides which, Excel
doesn't seem that "cooperative" in the first place. But I'm curious...
Any Excel hackers out there care to comment?

I do know that it has worked so far, even with multiple copies open.
However, that is no guarantee of correctness.
Clay
------------------------------------------------------------------------
Excel copies out the array before using it. On Windows with
DOS you can be sure that the second copy of Excel will not
get any processor time until this happens, so you're probably
ok.

If you are worried about which instance of Excel is talking to you,
you can find out the instance handle of Excel as follows:
```

```
HANDLE GetExcelHWND()
{
HANDLE hTask;
HWND hwnd;

hTask = GetCurrentTask();
hwnd = GetActiveWindow();
while (hwnd)
{
if (GetWindowTask(hwnd) == hTask)
break;
hwnd = GetNextWindow(hwnd, GW_HWNDNEXT);
}

return hwnd;
}

HANDLE GetExcelInst()
{
return (HANDLE) GetWindowWord (GetExcelHWND(), GWW_HINSTANCE);
}

_Joel Spolsky
joelsp@microsoft.com
------------------------------------------------------------------------
```

QUESTION:

```
>Joel's short routine makes it clear how to find the Excel instance
>but it still leaves me a little confused about the need for
>finding the instance handle of Excel. If the need is to find
>a unique identifier for the copy of Excel that called the
>DLL, would it not be sufficient to simply use the task handle?


Yep; you can use the task handle for uniqueness. The instance handle is
still useful for other things like MakeProcInstance.
_Joel
------------------------------------------------------------------------
```

SUMMARY:

```
I think we've pretty much established that in Excel/Win/DOS it is OK to
pass back a pointer to memory local to the DLL, and without worrying
about multiple instances of Excel calling the DLL (see Joel's post on
the subject). In NT (Not There?), this may not work, since real
multitasking/threading is implemented; but for the moment, that's
academic. Whether you *want* to do this is subject to your taste. It is
certainly more efficient (and to an extent, more elegant) than worrying
about passing space to the DLL. We do it *a lot*.

Clay
```

AUTHOR'S COMMENTS

It appears that task management is not an issue in Excel. As Clay says, it is a matter of taste. The task management routines in XLMATH can be omitted.

Excel v4.0 Notes

Excel v4.0 was **released** after XLMATH was written. No changes are required in XLMATH but the function Frequency() is now internally available in v4.0. In addition, v4.0 contains two new macros called Auto_Activate and Auto_Deactivate that can be used to run the macros required to Register and Unregister the custom functions in XLMATH.

References:

1. *Microsoft QuickC for Windows - C for Windows*; Microsoft Corporation, 1991; p540

2. P. Wilken; D. Honekamp, *Windows System programming*; Abacus, 1991; p657

3. *Microsoft Excel Function Reference*, Microsoft Corporation, 1991; Appendix A

4. P. Anderson; G. Anderson, *Advanced C*; Hayden Books, 1988; p 182

5. *OptiMem*, Applegate Software, 1991

6. *Micosoft Developer's Kit*, Microsoft Corporation; Microsoft Corporation, 1991

7. Martin Heller, *Advanced Windows Programming,* Wiley, 1992

Author:

Roy Kari
Department of Chemistry & Biochemistry
Laurentian University
Sudbury, Ont.
Canada
P3E 2C6
(705) 675-1151

The author would appreciate any and all comments on XLMATH. The author may be reached via Internet EMAIL at "ROY@NICKEL.LAURENTIAN.CA"
The author would appreciate receiving comments and any corrections or additions that anyone makes to XLMATH.