

MatLab API in C++

Christopher Dabney

Purpose

- MatLab ...
 - is an interpreted scripting language
 - conversion to object code is at runtime; computation time can be a little slow
 - has excellent prototyping and plotting functionality
 - contains convenient and very robust matrix operation packages

Purpose (cont)

- C++
 - is a programming language, optimal and with high speed floating point computation
 - is non-trivial to produce visual effects in
 - eg. plotting, GUIs, nice tables
 - can be difficult to secure a robust vector algebra package for

Purpose (cont)

- Solution: C++ invoking MatLab commands
 - number crunching in C++
 - matrix operations in MatLab
 - plotting, graphing, tables in MatLab
- For programmers with a robust & complex C++ program intending to plot results they are already obtaining
- For MatLab programming with scripts which frequently lag

Interacting with MatLab

- Not restricted to C++; Java, Perl, Fortran, and other languages can do it also
- Not restricted to the MS Visual Studio environment the demos are written in
- Also possible to invoke a C++ routine from MatLab's environment
- Three ways to interact
 - Send Data from C++ to MatLab
 - Call a MatLab function from C++
 - Generate a Dynamic Link Library (dll) file from a .m file
- We will use the first one

Demo Requirements

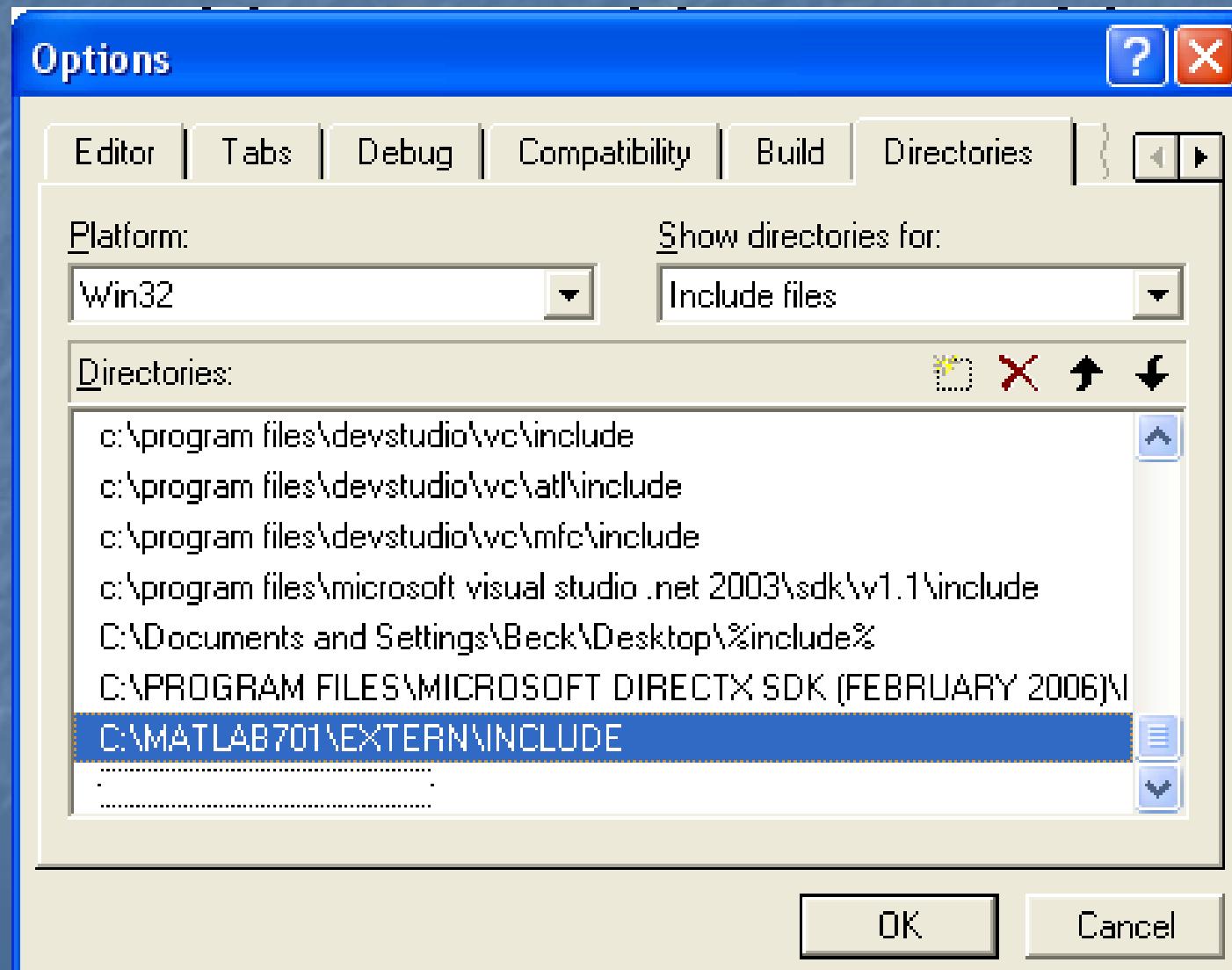
- A C++ Compiler
 - the demo uses MS Visual C++ v.6.0
- MatLab
 - the demo uses MatLab v.7.0.1
- Assume's prior programming experience in both MatLab and C++
 - no explanations are given for the commands
- Following slides explain setting up the demonstration program environment

Setting up the Environment

■ Creating paths to MatLab:

- Tools -> Options
- Directories Tab
- Select "Include files" from the drop-down list
- Add "C:\MATLAB701\EXTERN\INCLUDE" to the list
- Select "Library Files" from the drop-down list
- Add "C:\MATLAB701\EXTERN\LIB\WIN32\MICROSOFT\MSVC60"
- Library paths vary depending on the language and compiler

Screenshot



Setting up the Environment (cont)

- Two ways to include the appropriate library files:

Method 1) - Project -> Settings

- Links Tab

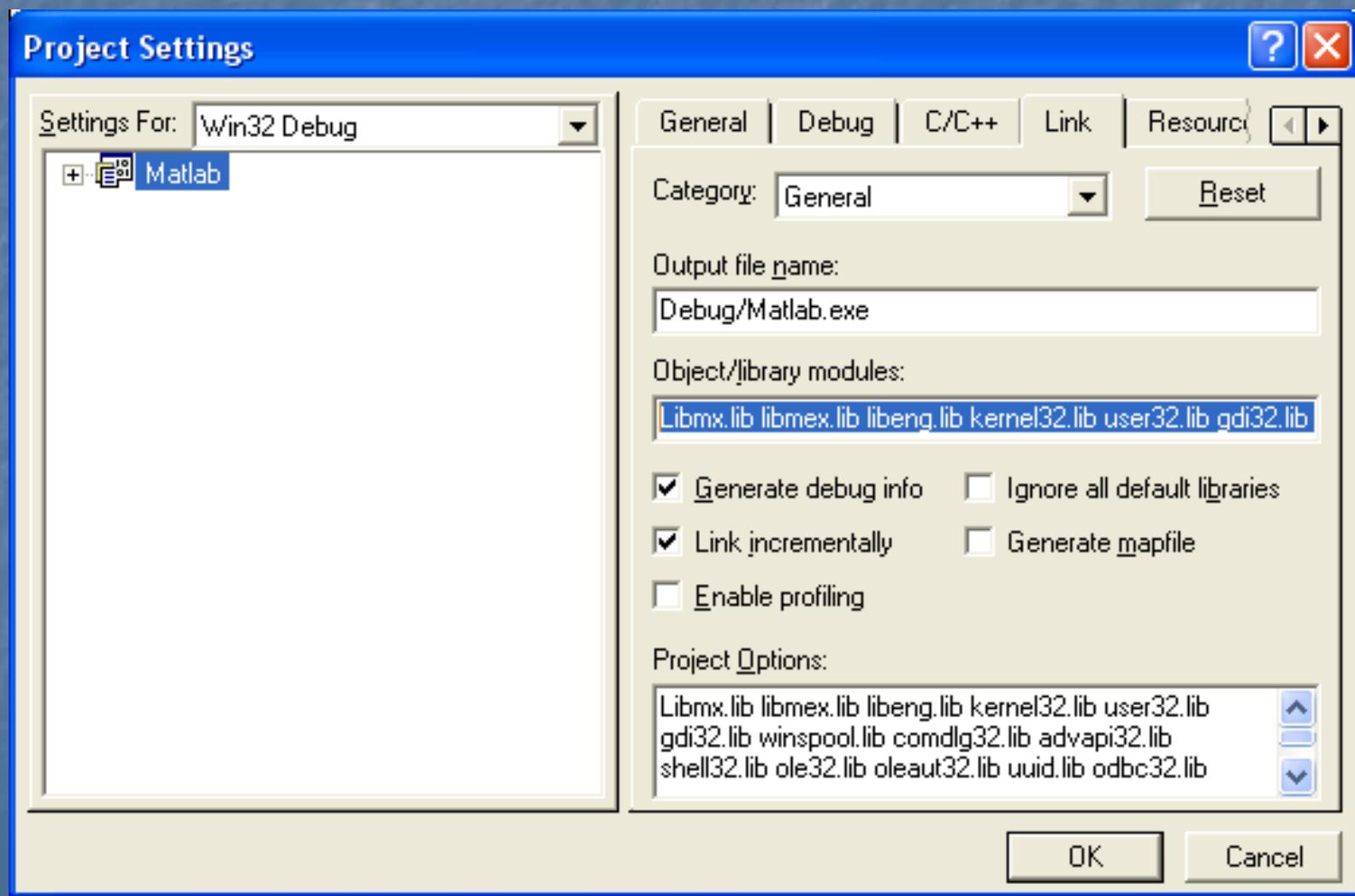
Under "Object/Library Module:", add three file names:

- Libmx.lib
- libmex.lib
- libeng.lib

Method 2) Add the following lines of code just under the includes in the source code:

```
#pragma comment( lib, "Libmx.lib" )  
#pragma comment( lib, "libmex.lib" )  
#pragma comment( lib, "libeng.lib" )
```

Screenshot



Connecting to MatLab Engine

- Header: #include <engine.h>, MatLab's Engine
- In the program, create a pointer to it:

```
Engine *m_pEngine;  
m_pEngine = engOpen(NULL) ;  
if (m_pEngine == NULL)  
{  
    cout << "Error: Not Found"  
    << endl;  
    exit(1);  
}
```

Invoking Commands

- `engEvalString(Engine* ptr, string cmd);`

```
engEvalString(m_pEngine, "x = 1:1:10);";  
engEvalString(m_pEngine, "y = x.^2;");  
engEvalString(m_pEngine, "plot(x,y);");
```

- Enables programmer to invoke any command in MatLab from the C++ platform (convenient)
- Not special by itself - might as well work straight from MatLab

IO between C++ and MatLab

- Use the “matrix array” *mxArray* data type
- Every variable in MatLab is a matrix – for C++ to exchange data with it, a data type that both C++ and MatLab recognize is needed
- *mxArray* can “bundle” C++ variables as matrices, so both platforms recognize this data type

Inserting Values (input)

- To pass a variable, eg. `x[0]`, into MatLab, create an *mxArray* for it
 - Allocate the space (`mxCreateDoubleMatrix`)
 - Copy the value (`memcpy`)
 - Name the variable in the Engine (`engPutVariable`)

```
double x[0];
mxArray *m_X;
m_X=mxCreateDoubleMatrix(1, 1, mxREAL);
memcpy((void *)mxGetPr(m_X), (void *)x,
sizeof(double)*1);
engPutVariable(m_pEngine, "x", m_X);
```

- Pointer *m_pEngine* is used to specify the engine
- Variable `x` in the MatLab Engine gets the value of `x[0]` in C++

Extracting Values (output)

- To extract the MatLab Engine's variable z, pull it into C++ as a *mxArray* matrix, then extract the bundled value out of that *mxArray* and into a C++ variable

```
double *cresult;  
mxArray *mresult;  
mresult = engGetVariable(m_pEngine, "z");  
cresult = mxGetPr(mresult);  
cout << cresult[0];
```

- mxGetPr* returns a pointer to a copy of the *double* value z stored in the MatLab Engine
- Warning: Unpredictable fatal errors occur if the data type in MatLab doesn't closely resemble the data type in C++ eg. Copying a 3x1 vector into a scalar

Passing Arrays & Matrices

- To pass arrays (vector), adjust the size parameters of the memory allocation and copy routine to match the dimensions of the array being passed
- Pass a vector a of dimensions: $1 \times \text{SIZE}$

```
double a[SIZE];
mxArray *A;
// assume a gets initialized, all values
A=mxCreateDoubleMatrix(1, SIZE, mxREAL);
memcpy((void *)mxGetPr(A), (void *)a,
sizeof(double)*SIZE);
engPutVariable(m_pEngine, "a", A);
```

- Copies entire C++ array a into the MatLab Engine's a

Passing Arrays & Matrices (cont)

- Pass a matrix of dimensions: SIZE x SIZE

```
double c[SIZE][SIZE];
// assume c gets initialized, all of it
mxArray *mxc;
mxc = mxCreateDoubleMatrix(SIZE, SIZE, mxREAL);
memcpy((void *) mxGetPr(mxc), (void *)c,
sizeof(double)*SIZE*SIZE);
engPutVariable(m_pEngine, "c", mxc);
engEvalString(m_pEngine, "c = c'");
```

- Note: C++ has row-major storage, and MatLab has column-major storage. Thus a matrix being exchanged must be transposed, either before or after the exchange, to maintain the matrix's semantics

See the Demos Program

- Demonstrates all of these features so far in Microsoft Visual Development Studio C++ v.6.0
- Demonstrates graph plots, invoking commands, input, output, and passing vectors and matrices.

Internet References

- Welcome to Zhenwang's Homepage (Q&A)
 - <http://www.sfu.ca/~zyao/teaching/ensc488faq.htm>
- A Tutorial to Call MATLAB Functions from Within A C/C++ Program
 - http://prism.mem.drexel.edu/Shah/public_html/c2matlab.htm
- Microsoft Visual Studio C++ & Interfacing Matlab with C/C++, Java
 - <http://www.qcf.gatech.edu/academic/LabDataAccess/C++IntroductionMatLab.Interfaces.doc>
- MatLab – The Language of Technical Computing, External Interfaces
 - http://www.mathworks.com.au/access/helpdesk/help/pdf_doc/matlab/apiext.pdf

Questions? ...

Thank You

No MatLab matrices were harmed in the making of this presentation.