C/C++ MEX Functions

Note: If you are not a CS-major student, it is possible that you have never programmed in C/C++. This topic is optional for you then, and an alternative task will be provided to you for the lab session of this topic.

What are MEX Functions?

- Compiled dynamically linked subroutines to be used in MATLAB environments.
- In MATLAB, use them like regular MATLAB functions.
- When to use:
  - Better efficiency
  - Simpler implementation in C/C++
  - Use of features and tools in C/C++ (such as STL)
  - Wrap existing codes/algorithms in C/C++ so they can be used in MATLAB
  - …
- When MEX and **m** files of the same name exist in the same directory, the MEX file is used for execution.
  - The **m** file (optional) is used to contain the help text.

Writing MEX Functions

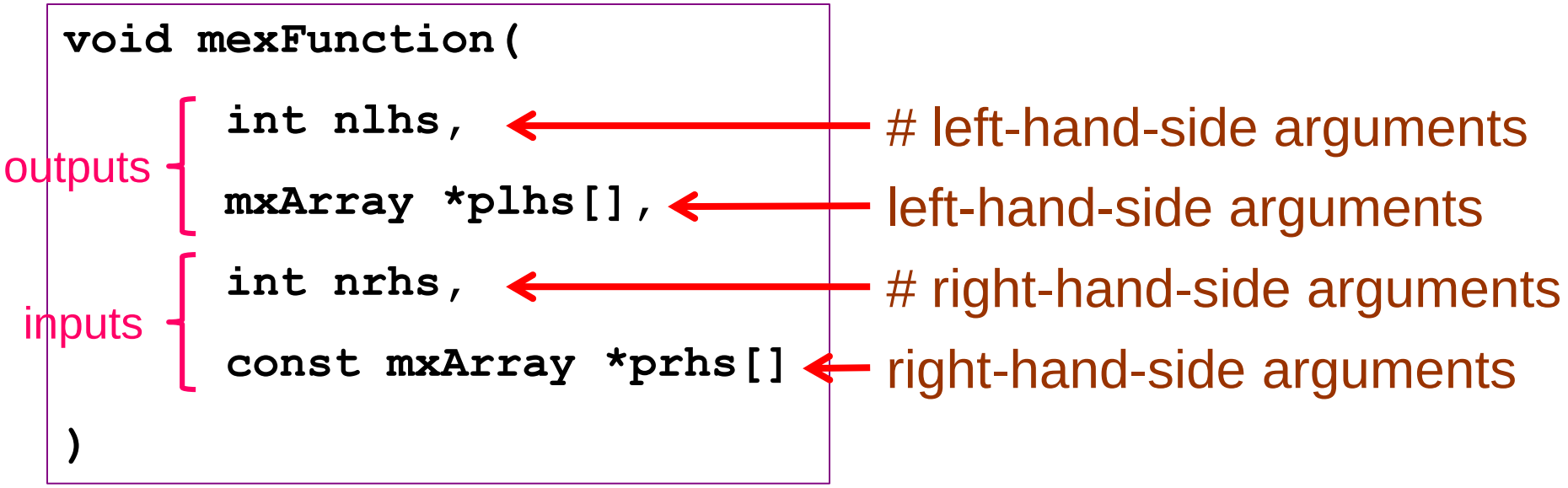
- Since we are actually writing in C/C++, it is useful to write the code in a C/C++ IDE to take advantage of its features (such as formatting and syntax checking).
- You can also edit the C/C++ code with the MATLAB editor. The correct syntax formatting will be used if you save the file with the .c or .cpp extension.
- Use `mex -setup` to see the compiler used.

Using MEX Functions

- At MATLAB command window, run **`mex`** from the command window to compile a C/C++ source file
  - Syntax: **`mex file_name.cpp`**
- The extension of the resulting **MEX** file depends on the platform. (You should see **.mexw64** on a Windows system.)
- Put the file in a directory where it can be found.
- When sharing **MEX** files, it is necessary to ensure that the environment where they are used is the target environment of compilation and have all the necessary library files (including the correct C runtime) installed. (For example, 32-bit mex files can not be used with 64-bit MATLAB, and vice versa.)

Components of MEX Source Codes

- **`mexFunction`**: The gateway routine (entrance point)



- **`mexAtExit`**: (optional) Usually for clean-up, such as to close open files.
- The actual processing code (one or multiple functions) for the processing.

Code Features

- Required include files:
  - **`mex.h`**: For **`mex*`** functions (used to access MATLAB programming features).
  - **`matrix.h`**: For **`mx*`** functions and classes (used to handle MATLAB arrays).
- The class **`mxArray`**: This corresponds to the arrays in MATLAB, so all the input and output data are stored in objects of this class. It can be made to store structures, cell arrays, etc.
- There are many functions available.

What to do in the Gateway Routine

- Argument checking:
  - Numbers, types, and dimensions of input and output arguments.
  - Generate appropriate error messages. (Unlike m functions, it is not very convenient to debug MEX functions. So do all the necessary checking here.)
- Data conversion:
  - Convert the input data to C/C++ data types.
  - Create **`mxArray`** objects to hold the output data.
  - If any of the inputs or outputs are array indices, be careful about the difference of 1-based indices (in MATLAB) and 0-based indices (in C/C++).
- Call the function(s) for actual processing.

Some Useful `mx*` Functions

The following are mostly used for setting output arrays:

- **`mxCreateDoubleMatrix`**: Returns a pointer (**`mxArray*`**) to a 2-D array of type double.
- **`mxCreateDoubleScalar`**: Returns a pointer (**`mxArray*`**) to a scalar (1x1) array of type double.
- **`mxCreateNumericArray`**: Returns a pointer to an array of any number of dimensions and any numeric class.
- **`mxDuplicateArray`**: Make a **deep copy** of an array.
- **`mxCreate*`**: Create other kinds of arrays …

Some Useful `mx*` Functions

- **`mxGetPr`**: Returns a pointer (**`double*`**) to the data of a double **mxArray**. (This is the real part for a complex array.)
  - Use **`mxGetData`** (returns **`void*`**) for **mxArray** of other classes.
  - Use **`mxGetScalar`** to get the first element (real part).
- **`mxGetNumberOfDimensions`** and **`mxGetDimensions`**: Get information of array size.
- **`mxGetM`** and **`mxGetN`**: Numbers of rows and columns (most useful for 2-D arrays).
- **`mxGetNumberOfElements`**:
- …

Some Useful `mex*` Functions

These communicate with the MATLAB environment:

- **`mexPrintf`**: Formatted output (C **`printf`**).
- **`mexErrMsgTxt`**: Error message (MATLAB **`error`**).
- **`mexCallMATLAB`**: Call a MATLAB function, with inputs and output arguments.
- **`mexEvalString`**: MATLAB **`eval`**, no outputs.
- **`mexGet`** and **`mexSet`**: Get and set the properties of graphics objects via their handles.
- …

Example MEX Functions

- Example: (multiplying every element of an array by two)

```
#include "mex.h"
#include "matrix.h"
void ctimes2(const double *A, double *B, int n)
{
  for (int i = 0; i < n; i++) B[i] = A[i] * 2;
}
/* The gateway routine */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
  /* input argument checking */
  if (nrhs != 1 || nlhs != 1) {
    mexErrMsgTxt("(ctimes2) usage: B = ctimes2(A)");
  }
  /* input A */
  double *A = mxGetPr(prhs[0]); //* pointer to the array content of A
  int n = mxGetNumberOfElements(prhs[0]); // # elements
  mwSize ndim = mxGetNumberOfDimensions(prhs[0]); // # dimensions
  const mwSize *dims = mxGetDimensions(prhs[0]); // sizes of the dimensions
  /* output B */
  plhs[0] = mxCreateNumericArray(ndim, dims, mxDOUBLE_CLASS, mxREAL);
  double *B = mxGetPr(plhs[0]); //* pointer to the array content of B
  /* actual processing */
  ctimes2(A, B, n);
}
```

Example MEX Functions

- Example:

(pairwise squared distances between two sets of vectors)

```
#include <math.h>
#include "mex.h"
#include "matrix.h"

void get_pair_d2(int L, int N1, int N2, const double *V1, const double *V2, double *D2)
{
  int i, j, ki, qi, qj, k, iL;
  double dv;
  for (i = 0; i < N2; i++) {
    ki = i * N1;
    qi = i * L;
    for (j = 0; j < N1; j++) {
      qj = j * L;
      k = ki + j; // linear index in D2
      D2[k] = 0;
      for (iL = 0; iL < L; iL++) {
        dv = V2[qi+iL] - V1[qj+iL];
        D2[k] += dv * dv;
      }
      D2[k] = sqrt(D2[k]);
    }
  }
}
```

continued on next page

Example MEX Functions

- Example:

(pairwise squared distances between two sets of vectors)

```
/* The gateway routine */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
  /* input argument checking */
  if (nrhs != 2 || nlhs != 1) {
    mexErrMsgTxt("(pairwise squared distances) usage: D2=getPairD2(V1,V2)");
  }
  if (mxGetM(prhs[0]) != mxGetM(prhs[1])) {
    mexErrMsgTxt("V1 and V2 should have the same number of rows");
  }
  /* inputs V1 and V2 (2-D array) */
  double *V1 = mxGetPr(prhs[0]);
  double *V2 = mxGetPr(prhs[1]);
  int L = mxGetM(prhs[0]); // dimension of the points
  int N1 = mxGetN(prhs[0]); // # points in V1
  int N2 = mxGetN(prhs[1]); // # points in V2
  /* output D2 (2-D array) */
  plhs[0] = mxCreateDoubleMatrix(N1, N2, mxREAL);
  double *D2 = mxGetPr(plhs[0]);

  /* actual computation */
  get_pair_d2(L, N1, N2, V1, V2, D2);
}
```