Introduction to CMex
E177
April 25, 2006
http://jagger.me.berkeley.edu/~pack/e177

Matlab variable types: primitives
The three main object classes in Matlab are
– double
  • multidimensional array of double precision floating point numbers
– char
  • multidimensional array of ascii characters
– logical
  • Multidimensional array of 1-bit (0/1) numbers
Eight other built-in primitive variable types are
– uint8, uint16, uint32, uint64
  • multidimensional array of 8, 16, 32 or 64-bit, unsigned integers
– int8, int16, int32, int64
  • multidimensional array of 8, 16, 32 or 64-bit, signed integers

Two things to remember
Every variable in Matlab is an array
Arrays are at least 2-dimensional (no notion of 1-d array)

Matlab variable types: derived primitives
Two other important object classes in Matlab are
– cell
  • multidimensional array of "containers"
– struct
  • multidimensional array of a structure with fields (common across array)
In a cell, the contents of a container may be
– double, char, intXX, uintYY
– Another cell array
– A struct array
– An object of other classes (listed on next slide)
In a struct, the value of a field may be
– Same list as above

The mxArray object
Inside the C-programs we write, all of these types of objects will be variables of the type mxArray.
We will never actually have an mxArray variable, but will always have pointers to them. All of the mxArray routines have input arguments that are addresses of mxArray variables.

Typical example within a CMEX file

The mexFunction gateway
Suppose we have a file somefunction.c

#include "mex.h"
void mexFunction(int nLHS, mxArray *pLHS[],
                 int nRHS, const mxArray *pRHS[])
{
    /* Code here */
}

What happens if in Matlab (ie, at command line, in script, or in function) a command of the form
[Y,Z] = somefunction(A,B,C);

The mexFunction gateway (cont’d)

#include "mex.h"
void mexFunction(int nLHS,
                 mxArray *pLHS[],
                 int nRHS,
                 const mxArray *pRHS[])
{
    /* Code here */
}

The function definition says
–mexFunction returns nothing (void)
–mexFunction accepts 4 arguments
  • 1st argument is an integer
  • 2nd argument is an array of (pointers to mxArray)
  • 3rd argument is an integer
  • 4th argument is an array of (pointers to mxArray), moreover, the value of this variable will not be changed (const qualifier)
The *mexFunction* gateway

Before executing

\[ [Y,Z] = \text{somefunction}(A,B,C); \]

we first need to compile the function. At the *Matlab* prompt,
execute the command

\[ \gg \text{mex somefunction.c} \]

This will create an executable file in the same folder that can be called directly from *Matlab*.

You may have to (once) set up the MEX infrastructure if you haven’t already. In that case, type

\[ \gg \text{mex –setup} \]

and simply follow the instructions.

---

**Where is raw data stored in DOUBLEs**

If `pMAT` is a pointer to an `mxArray`, and `mxIsDouble(pMat)`
is TRUE, then

- `mxGetPr` returns the address of the first element of the real part (stored in linear order)
- `mxGetPi` returns the address of the first element of the imaginary part

```c
double *pReal, *pImag;
mxArray *pMat;
/* code to put a legal address in pMat */
pReal = mxGetPr(pMat);
If (mxIsComplex(pMat)) {
    pImag = mxGetPi(pMat);
}
```

---

**Where are the contents of Cells**

If `pCell` is a pointer to an `mxArray`, and `mxIsCell(pCell)`
is TRUE, then with integer \( J \geq 0 \),

- `mxGetCell(pCell,J)` returns the address of the \((J+1)\)th `mxArray`
in the cell (using linear ordering)

```c
int idx;
mxArray *pMat, *pCell;
...  
idx = 0; /* the first element */
pMat = mxGetCell(pCell,idx);
idx = 1;
pMat = mxGetCell(pCell,idx); /* 2nd elmnt */
```

Let’s use this to study addresses in cells, in `pdisplay.c`