How to MEX C/C++ programs: a simple example

Here is a simple example of how to bake in C code into Matlab by using mex. The aim of the program is to estimate mean and standard deviation of a sample from a normal distribution. It takes in two values: a double array ”par” that contains the true mean and standard deviation and an integer ”N” that is the sample size. In C the code for this program can look something like this:

```
/*Function mexEx1.c*/

#include <math.h>
#include <stdlib.h>
#include <gsl/gsl_rng.h>                      /*Random number generation*/
#include <gsl/gsl_randist.h>                 /*Statistical distributions*/
#include <gsl/gsl_statistics.h>    /*Functions like mean and variance*/

void mean_var_est(double *par, int N, double *val);

int main()
{
double par[2] = {1,2};         /*The true mean and standard deviation*/
int N = 10;                                          /*The sample size*/
double val[2];                 /*The estimated mean and std (now empty)*/

mean_var_est(par,N,val);          /*The call to mean_var_est function*/
return 0;
}

void mean_var_est(double *par, int N, double *val)
{
gsl_rng *r = gsl_rng_alloc(gsl_rng_mt19937); /*Define random number t*/
long seed = time(NULL)*getpid(); /*Define a seed for r*/
gsl_rng_set(r,seed); /*Initiate the random number generator with seed*/

double *v;
v = malloc(N*sizeof(double));
```

```
                    /*Allocates memory for a double array v of size N*/

int i;
for (i = 0; i < N; i++)
{v[i] = par[0]+gsl_ran_gaussian(r,par[1]);}
                            /*Generates N Normally distributed values*/

val[0] = gsl_stats_mean(v,1,N);        /*Calculates mean and variance*/
val[1] = sqrt(gsl_stats_variance(v,1,N));
printf("%f %f\n",val[0],val[1]);
                         /*Prints the calculated values on the screen*/
}
```

On Linux, you can compile it by typing "gcc -O2 -lgsl -lgslcblas mexEx1.c -o mexEx1" in the terminal and run by typing "./mexEx1".

The same program converted into something that Matlab can handle may look as follows:

```
/*Function mexEx2.c*/

#include "mex.h"                                    /*The mex library*/
#include <math.h>
#include <stdlib.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_statistics.h>
#include <gsl/gsl_cdf.h>

void mean_var_est(double *par, int N, double *val)
{
gsl_rng *r = gsl_rng_alloc(gsl_rng_mt19937);
long seed = time(NULL)*getpid();
gsl_rng_set(r,seed);

double *v;
v = malloc(N*sizeof(double));

int i;
for (i = 0; i < N; i++)
{v[i] = par[0]+gsl_ran_gaussian(r,par[1]);}

val[0] = gsl_stats_mean(v,1,N);
val[1] = sqrt(gsl_stats_variance(v,1,N));
                      /*No changes, except for the removal of "printf"*/
}


/* The gateway function that replaces the "main".
*plhs[] - the array of output values
*prhs[] - the array of input values
*/
void mexFunction( int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    double *par, *val;                 /*Variable declarations as in C*/
    int N;
    par = mxGetPr(prhs[0]);
                     /*Sets "par" to be the first of the input values*/
    N = mxGetScalar(prhs[1]);
                      /*Sets "N" to be the second of the input values*/
    plhs[0] = mxCreateDoubleMatrix(1,2,mxREAL); /*Creates a 1x2 matrix*/
    val = mxGetPr(plhs[0]);
                    /*Sets "val" to be the first of the output values*/
    /* call the computational routine */
    mean_var_est(par,N,val);
}
```

You can compile it by typing ”mex -lgsl -lgslcblas mexEx2.c” in the Matlab window and run the program as any other Matlab function, tex by typing ”a = mexEx2([1,2],100)”. Here, $\mu = 1, \sigma = 2, N = 100$ and ”a” is the vector that is going to contain the estimated variance and standard deviation.

In short, there are two necessary changes to be made to the C-code. Firstly, #include ”mex.h” has to be added to the header. Secondly, the ”main” function has to be replaced with the ”gateway” function that reads the input, passes it on to the C function and returns the output.

A bit of advice:

- Make sure that the type of input/output that you give at the Matlab prompt to the function is consistent with what the said function can handle. That is, you can’t for example write just ”mexEx2([1,2],100)” to run the code. Matlab will crash if you do. The error handling through ”mexErrMsgIdAndTxt” may come in handy.

- In the same spirit as the above. If you are not very good at C then you are almost bound to mess up array declaration/value assignment somewhere. This will lead to a segmentation error that will not be noticed when compiling but will, again, cause Matlab to crash. So do not start by writing a mex function directly, write the pure C version of it first and make sure it runs without errors.

Good places to look at:

http://www.cosy.sbg.ac.at/ uhl/C-matlab.pdf
http://www.mathworks.com/support/tech-notes/1600/1605.html