

ECE-486 Laboratory 1, Spring 2009

Due Feb. 2, 11:00 am

Objectives

- Learn to write MATLAB functions and scripts that call MATLAB functions.
- Become familiar with using Linux to develop and debug C programs.

Assignment

TASK 1: Learn to use “structures” in Matlab. For this program, implement discrete-time convolution in which the input signal is provided *one sample at a time*. For each call to the function, your program should accept a single input sample, and provide the corresponding output sample for the convolution result (using the most recent input values). Use a structure to store the required information for the convolution from one call to the next.

To do this, first create a function “`init_conv`” with a call format given by

```
s = init_conv(h);
```

Here, “`h`” is a fixed length vector containing the desired impulse response of an FIR filter, and the returned structure “`s`” should be a structure containing all information needed to implement the filter in subsequent calls (a copy of `h`, any required storage arrays, etc.). Convolution output samples should be calculated *one sample at a time* by a companion routine “`calc_conv`” with a call format given by

```
[y,s] = calc_conv(x,s);
```

The input “`x`” should be a scalar input sample, and the calculated result “`y`” should be a single sample of the output signal. The structure “`s`” is used as both an input and output argument, and may be modified by the function. However, the amount of storage required by the structure `s` should *not increase* as `calc_conv` is repeatedly called.

Your solution should not use the MATLAB `filter()` or `conv()` functions.

TASK 2: Implement the problem assigned in TASK 1 in C under Linux. Use the `gcc` compiler to produce executable code. To debug your program, you can use the `ddd` debugger (but you must specify the “`-g`” option to the `gcc` compiler).

You should become familiar with the use of pointers to manage memory and pass parameters in C. In particular, you’ll need to be comfortable with the

`malloc()`, `calloc()`, and `free()` functions. You'll also need to understand the use of C structures so that ALL information needed to maintain the convolution calculation can be encapsulated into a single data structure. You should avoid the use of global variables to pass information between routines. In future assignments, your collection of subroutines may be used to maintain several different convolutions at the same time (by using separate structures `s1`, `s2`, `s3`, etc.).

In writing your routines, separate all code required to implement the convolution algorithm from the code required to test your routines. The goal is to create a collection of subroutines to perform convolution which could be linked to *any* main program. Place subroutines/functions in a separate file from the main program which you use to test your routines. For example, "`file1.c`" might contain your `main()` function, and "`conv_subs.c`" might contain a collection of functions called by `main()` (or by a different main program) to do the calculations. By convention, a third file `conv_subs.h` should also be created and included (using `#include` statements) in *both* `file1.c` and `conv_subs.c`. Use the following as a guideline as to what should (or should not) be included in each of the files:

`conv_subs.h`: Should contain structure declarations and declarations describing the function interface.

This file should *NOT* include and variable definitions or executable code. It should not declare or allocate any memory.

`conv_subs.c`: Should contain the actual function definitions needed to implement the convolution. Usually, at least three functions are defined: One initialization routine to create the convolution data structure, allocate any required memory, and initialize variables; A second routine which performs the actual calculations (analogous to the Matlab routine `calc_conv()` in Task 1); and a third routine which is called at program termination to cleanly deallocate any memory which was previously reserved.

Generally, this file should avoid performing any I/O. (Use print statements only for debugging purposes, or for issuing an error message if execution is being terminated.)

`file1.c` The file containing your main program definition. For this lab, this file should exercise your convolution routines to demonstrate the function interface and to verify that the routines produce the expected results.

To compile the program, use

```
gcc -g file1.c file2.c
```

This command should produce an executable file named "`a.out`". Run the program by typing "`a.out`", or start the debugger using

```
ddd a.out
```

What should be handed in:

1. Hand in a *short* (English) description of the programs you are submitting, and how they are used. Provide documentation showing how *you* tested your routine and give your test results showing that it works.
2. Hand in commented MATLAB and C source code listings.
Program listings and documentation will be collected at the beginning of class on the due date.
3. Create a TAR archive containing *only* your MATLAB and C source code. The TAR file should be named lab1.tar and uploaded on the ECE 486 website (use the “Submissions” tab for your group on the course web site). The TAR file must be uploaded before class (11:00 AM) on the due date.
4. Demonstrate your working routine during your scheduled group meeting time in the week of the due date