3.8. Design and use of IIR filters in Matlab

(If you do not have access to Matlab, this section can be skipped without upsetting the flow of the course.)

The Matlab signal processing tool kit provides a number of built-in functions, sparing us the effort of writing programs to do it in C. Matlab provides a number of functions for designing digital filters: in particular, for selecting appropriate *a*'s and *b*'s for filters with given frequency characteristics. By and large, I will only be using one of the kinds of digital filter that Matlab includes — Butterworth filters — which is a good general-purpose kind. In the Matlab manual there are the details about other filter designs. They are conceptually very similar, and are just as easy to use, but I am only going to use one type of filter as an example.

Listing 3.4. Example of filtering in Matlab

```
>> fid = fopen('joe.dat','r');
>> In = fread(fid,6000,'short');
>> [b,a] = butter(5,600/4000);
>> Loband = filter(b,a,In);
>> [bb,aa] = butter(4,3000/4000,'high');
>> Hiband = filter(bb,aa,In);
>> fid2 = fopen('joe_lp.dat','w');
>> fwrite(fid2,Loband,'integer*2');
>> status = fclose(fid2);
>> fid3 = fopen('joe_hp.dat','w');
>> fwrite(fid3,Hiband,'integer*2');
>> status = fclose(fid3);
```

Listing 3.4 gives an example of how you can design and use digital filters in Matlab. Rather than being a program, it records a sequence of commands typed by the user during an interactive session with Matlab. The double "greater than" at the beginning of each line is the Matlab prompt. Matlab isn't compiled, it is just an interpreter, so you can type each line in after the prompt and it does each operation as you tell it to. You have to start Matlab by typing "`matlab`" to begin with.

<u>Student: Right, so you get a Unix prompt and then you just type Matlab.</u>

Yes, and then this >> prompt comes up.

<u>Student: So these double arrows are the Matlab prompt, OK, and then each line is an instruction.</u>

Yes, it is an instruction to Matlab. Now I will briefly go through them, line-by-line, because it is fairly typical of the kind of interaction that you might have when using Matlab tools. The first two lines open the file `joe.dat` and reads some of it ("Joe took fath—") into the vector variable `In`. In then next four lines, we design some filters, we apply the two filters to the original signal to create two new signals, and in the remaining six lines we write each of those new signals to a file.

So, in the first line, we open the file `joe.dat` for reading. `fid` is a file identification number, the number that Matlab assigned to `joe.dat`. Line 2 reads in 16000 samples: the `fread` statement is short for "file read": "from `fid` read 16000 shorts and put them in `In`", which is a variable name for a Matlab array, a column vector in fact. OK so we put the signal there and forget about it. Now the thid step is something different; here we design the first filter. It is a Butterworth filter, so the function name is `butter`. The number 5 means that we want a fifth order filter, which means that we want a filter that refers to the present value of the input and the five previous ones, six in all, in fact. 600/4000 specifies the cut-off frequency of the low pass filter that we want to design, 600 Hz: the 4000 in the denominator is half of the sampling rate. (We could have put 0.15 instead of 600/4000, but expressing the cut-off frequency as a fraction enables us to adapt this listing to other cut-off frequencies and other sample rates more easily.) Remember from chapter 1 that the sampling theorem says that the highest frequency that you can measure is half the sampling rate; that is where the number 4000 in this expression comes from. (It is known as the <u>Nyquist frequency</u>: half the sampling rate is the highest frequency you can analyse; see section 2.5.) Once again we express frequency as a ratio of the sampling frequency, just as we did in table 3.3. But unlike table 3.3, here we are relating the desired frequency to *half* the sampling frequency. Matlab requires it this way; but the idea of expressing frequency as a ratio is basically the same, OK?

Now when you enter line 3, that generates a sequence of *b*'s and a sequence of *a*'s, which are the desired coefficients in the general equation for the IIR filter. They will be whatever numbers they are. We don't care what they are: we are primarily interested in are the acoustic characteristics of the filter. We need to get the *a*'s and *b*'s somehow, and that is what this function does. That is why on the left hand side of line 3 in square brackets it says `[b,a]`: it means vector `b` (a sequence of *b*'s) and vector `a`, a sequence of *a*'s. Right?

<u>Student: Yes.</u>

In line 4 we apply the filter to the input wave `In`, using the given *b*'s and *a*'s. That is what "`filter(b,a,In)`" means. We put the result of that in a variable called `Loband`. So `Loband` is the output of applying the filter to the input, a filter with stated *b* and *a* coefficients. OK, did you understand that all now?

<u>Student: Yes: `Loband` is the result of applying the filter to the input.</u>

Yes, that is right, that is exactly right.

OK, now, in line 5 we design a new filter, and I am going to call the vectors that store the *b*'s and *a*'s of this filter `bb` and `aa`. Because we are designing another filter, with a new set of coefficients, we can use new variable names. You can call them what you like; you can call them `anew` and `bnew` if you like; it doesn't make any difference.

<u>Student: They are new multipliers.</u>

You can call them `x` and `y`, you can call them anything you like. It doesn't matter. In line 3 I only called them `b` and `a` out of convention, because it looks like the examples in the textbooks, but they are just variables so you can name them pretty much whatever you want.

Lines 5 and 6 define and apply a high pass filter to the same input, `In`. Line 7 is like the first line, except that it mentions a different file name, and we are opening it to write, not to read. That is why it doesn't say `'r'` at the end, it says `'w'`, meaning we are opening a file to write to. Here we are only opening it though, it isn't yet written. Matlab creates a new file called `joe_lp.dat`. Line 8 is where we actually do the writing: `fwrite` means "file write", `fid2` refers to `joe_lp.dat`, as defined in line 7, and `Loband` is the variable containing the low-pass filtered signal. We also have to specify what kind of objects to write. Curiously, to write shorts we have to state `'integer*2'`, meaning two-byte integers. MATLAB can write out numbers in various ways. It could write them out as ASCII text, it could write them out as floating point numbers, or in other formats. So we are going to write them out as two-byte integers.

After we have written the file we have to close it. That is what `fclose` does on line 9. The next 3 lines open, write and close the file `joe_hp.sd` using the signal stored in the variable `Hiband`. Easy peasy, yes?

<u>Student: That is great, it really works doesn't it!</u>

It really works, yes, and it is very simple.