**ECE 4522/5514: DIgital Signal Processing**

# Computer Assignment (CA) No. 3: Real-Time Filtering

The goal of this assignment is to introduce you to efficient implementation of digital filters. You will use your buffering program for this implementation:

cat x.raw | my\_buffer\_program | play -e signed -c 1 -b 16 -r 8000 -t raw -

There is only one task :)

Recall the general expression for a linear constant coefficient difference equation:



Configure your buffer program so that you read N=160 points from the input stream and send M=160 points to the output stream. You should loop over the input until you reach the end of the stream, and write the same number of output points that you read from the input.

Demonstrate that everything is working properly by showing that when you apply an input signal, you produce EXACTLY the same output signal. Configure your program so that it reads data and passes the data directly to the output. Show that the output file is identical to the input file:

cat x.raw | my\_buffer\_program > x\_new.raw

diff x.raw x\_new.raw

Next, implement a function that filters a buffer of data using the above difference equation. The function definition, or user interface, should be something like this:

bool filter(short int\* sig\_o, short int\* sig\_i, long N, float\* a, long p, float\* b, long q);

where:

short int\* sig\_o: filtered signal (output)

short int\* sig\_i: signal to be filtered (input)

long N: number of input (and output) points

float\* a: autoregressive (AR) filter coefficients (input)

long p: number of AR coefficients (input)

float\* b: moving average (MA) filter coefficients (input)

long q: number of MA coefficients (input)

Verify your result by processing the same data stream through MATLAB and demonstrating that you get exactly the same result. Try this for several different types of filters (e.g., MA only, AR only, ARMA).

Note that your filter must retain memory between function calls – it cannot assume zero initial conditions for each call. It must initialize itself with the first call. Also, your filter implementation should not use shift registers. Instead, use pointer manipulations to implement the delay lines needed for this filter to operate properly.

This general structure can be used to implement many DSP algorithms. We will continue to use this structure throughout the course.