1. I/O and Signal Flow

Write a program that reads a parameter file, processes a signal "frame-by-frame", and outputs the processed signal to a new file. The command line arguments should be:

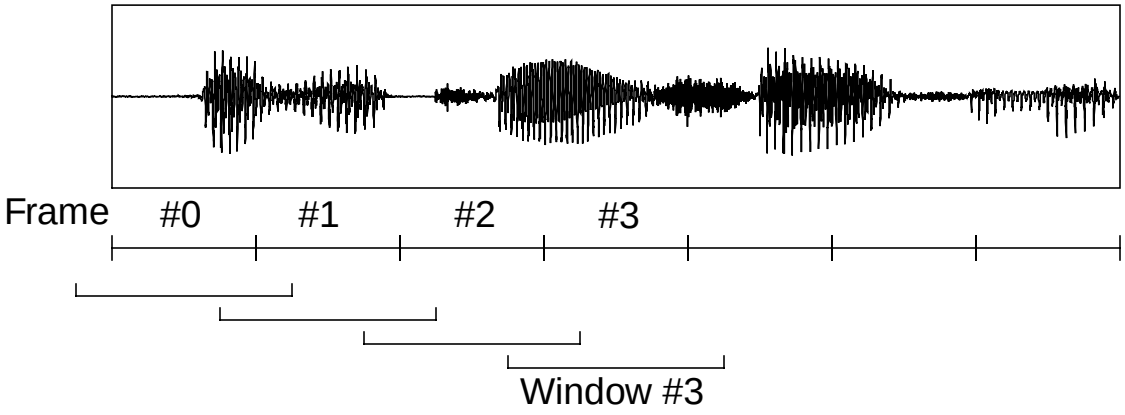
```
process_data    —parameter_file param.text \
                —output_file signal_out.binary \
                signal_in.binary
```

The parameter file should be an ASCII file containing three lines with this syntax:

```
sample_frequency    =  8000 Hz
frame_duration      =  0.02 secs
window_duration     =  0.03 secs
```

The input data will be 16-bit linear sampled data stored as a sequence of short ints (16 bits). The output data should be the same format.

The signal should be processed by keeping a buffer of length window_duration*sample_frequency and sliding it frame_duration*sample_frequency samples each frame:



The window is moved frame_duration*sample_frequency samples each frame. For this example, the output signal and the input signal should be the same.

2. SIgnal Resampling

Write a program that resamples a signal from one sample frequency to another. The command line arguments should be the same as problem no. 1. The parameter syntax should be the same as problem no. 1.

The parameter file should have the following parameters:

sample_frequency	=	8000 Hz
frame_duration	=	0.02 secs
interpolation_window_duration	=	0.100 secs
output_sample_frequency	=	16000.0 Hz

Perform the following operations:

(a) Upsample from 8000 Hz to 16000 Hz.
(b) Downsample the output of (a) from 16000 Hz to 8000 Hz.
(c) Plot the difference of the two signals.
(d) Downsample from 8000 Hz to 4000 Hz.
(e) Upsample from 4000 Hz to 8000 Hz.

Use the file test_speech_data.binary.

Listen to these files using an audio system. Explain the results in steps (c) and (e).

3. Filtering

Write a program that reads a filter from a file and filters a signal. The command line arguments should be the same as problem no. 1. The parameter syntax should be the same as problem no. 1.

The parameter file should have the following parameters:

```
filter:
    moving average:
        delay = 0          weight = 0.75
        delay = -1         weight = 0.25
        ...
    autoregressive:
        delay = -1;        weight = 0.5;
        delay = -2;        weight = 0.125;
        ...
    end
```

This file describes the following filter:

$$y(n) = 0.5y(n-1) + 0.125y(n-2) + 0.75x(n) + 0.25x(n-1)$$

Perform the following operations:

(a) Filter the test signal.
(b) Compute the filter impulse response.
(c) Compute the filter unit step response.
(d) Compute the filter frequency response by generating a signal of the following form:

$$x(n) = A\sin\left(\frac{2\pi 0.4n^2}{f_s}\right)$$

where $f_s = 8000.0$ Hz, and $0 \le n < 10000$. Explain this signal!

4. Discrete Fourier Series Analysis and Synthesis

Write a program that analyzes a signal on a frame-basis, computes the coefficients of a discrete fourier series (DFS) each frame, outputs these to a binary file as floating point numbers, takes these coefficients and computes an inverse DFS, and outputs the new signal to a file. Follow the paradigm of problem no. 1. Support the following cases:

(a) Output the DFS coefficients for each frame from [0,N/2];

(b) Support an option "—average" which, when specified from the command line, outputs only one set of coefficients per file. These coefficients should be the average of the log of $\{c_k\}$. In other words, for each frame, compute the DFS, take the log of the magnitude of $\{c_k\}$, and add it to a running average. After the last frame in the file is processed, output the final average. This represents a technique to compute the long-term spectral characteristics of the signal.

(c) For the case where frame_duration = window_duration, show that the output signal computed from the inverse DFS is the same as the input signal.

(d) Repeat (c) for window_duration = 0.03 secs and frame_duration = 0.02 secs. What happened?

Any ideas how to improve the result?

5. Discrete Fourier Transform

Write a program that analyzes a single frame of data from a signal file and computes the discrete Fourier transform. The program should accept the following parameters from a parameter file:

center_time = 0.5 secs
window_duration = 0.128 secs
lower_frequency = 300 Hz
upper_frequency = 600 Hz
frequency_resolution = 10 Hz

Compute the discrete Fourier transform at frequency values that are increments of the frequency_resolution starting with the lower frequency and ending with the upper frequency.

Print the values of the spectrum in pairs (frequency, dB value) to stdout. This output can then be piped into the xgraph plotting program.

6. Frequency Response From Sampling The Z-Transform

Generate an ideal impulse train with a fundamental frequency of 100 Hz at a sample frequency of 10,000 Hz. Analyze this signal at a frame_duration of 20 msec and a window_duration of 40 msec.

Perform the following operations:

(a) Plot the frequency response of frame number 10 of the signal by plotting the magnitude of the discrete Fourier series coefficients for the frequency range [0,5000 Hz].

(b) Plot the frequency response of frame number 10 of the signal by sampling the discrete Fourier transform at increments of 100 Hz.

(c) Plot the frequency response of frame number 10 of the signal by sampling the z-transform at increments of 100 Hz.

(d) Filter the signal through the filter of problem no. 3. and repeat (a)-(c).

(e) Repeat (b) for a window_duration of 21 msec. Compare the result with (b) and explain.

(f) Consider the value of the frequency response in (a) as the true value. Define an error function as the difference in the measured magnitude minus the “true value” in (a). Evaluate the spectrum as in (b) at 500 Hz, and plot this error function as a function of the window_duration for the range [0.02 sec, 0.100 sec]. Explain.

(g) Repeat (f) for the spectrum evaluated at 513 Hz.

7. The SchÜr-Cohn Stability Test

Implement a program that tests the stability of any filter specified by the parameter file in problem no. 3. The program should work something like this:

```
test_filter_stability -parameter_file foo.text
    the filter specified in foo.text is STABLE.
```

The program should call a function:

```
int check_filter_stability_C(FILTER* filter);
```

the return value should be “0” if the filter is stable, “-1” if the filter is unstable.

The above function should call another function:

```
int convert_predictor_to_reflection_C(float* rc, float* a, int order);
```

This should implement the SchÜr-Cohn Stability test. Also implement the recursion in the reverse direction:

```
int convert_reflection_to_predictor_C(float* a, float* rc, int order);
```

Both routines should return “0” on success, and “-1” on failure to complete the recursion (instability).