L314
MPHIL IN ADVANCED COMPUTER SCIENCE
COMPUTER SCIENCE TRIPOS Part III

Monday 28 October 2024 12:00 to Monday 4 November 2024 12:00

Module L314 – Digital Signal Processing – Assignment 1

This assignment involves programming. The recommended programming language is Julia *and library functions referred to in the problems may be found in the Julia packages* `DSP.jl`, `FFTW.jl` *and* `Plots.jl`. *[Implementations in other suitable languages, using equivalent library functions, such as* MATLAB*'s* Signal Processing Toolbox, *or the Python packages* `matplotlib` *and* `scipy.signal`, *are also acceptable.]*

Prepare the solutions and answers to all parts as a single PDF file and include all source code written, along with any required outputs produced by the programs. A `Pluto.jl` *notebook provides a convenient way to combine answer text, Julia source code and outputs into a single PDF. [Alternatives include a* Jupyter *notebook for either a Julia or Python solution, or* MATLAB*'s* `publish` *function or* Live Editor.*]*

Submit your work via

https://www.vle.cam.ac.uk/course/view.php?id=254472

no later than 12:00 on Monday 4 November 2024.

Students will be required to sign an undertaking that work submitted will be entirely their own; no collaboration is permitted.

1 (*a*) Julia commands (similar to)

```
using DSP, Plots
plot_filter(x, y; kwargs...) =
  plot([x y], linecolor=[:blue :red], markercolor=[:blue :red],
       markershape=[:x :circle], markerstrokewidth=3,
       size = (800, 250), xticks=[], yticks=[0],
       label=["x" "y"], ylim=[-7,7]; kwargs...);
x = [ 0,  0,  0, -4,  0, 0,  0,  0,  0, 0, 2,  2, 2, 2,
      2,  0, -3, -3, -3, 0,  0,  0,  0, 0, 1, -4, 0, 4,
      3, -1,  2, -3, -1, 0,  2, -4, -2, 1, 0,  0, 0, 3,
     -3,  3, -3,  3, -3, 3, -3,  3, -3, 0, 0,  0, 0, 0, 0 ];
y = filt([1, 1, 1, 1]/4, [1], x);
plot_filter(x, y; title="4-point moving average system")
```

produced the plot on slide 19 to illustrate the 4-point moving average system. The `DSP.jl` library function `filt(b, a, x)` applies to the finite sequence x the discrete system defined by the constant-coefficient difference equation with coefficient vectors b and a (see slide 25).

Change in this program the `filt` parameters to implement instead the

(*i*) exponential averaging system (slide 20)

(*ii*) accumulator system (slide 21)

(*iii*) backward difference system (slide 22)

and provide the coefficient vectors b and a for each of these systems.

[*Note:* A function equivalent to `filt` is in MATLAB called `filter` and in Python `scipy.signal.lfilter`.]

(*b*) (*i*) Simulate the reconstruction a sampled frequency-limited signal, following these steps:

- Generate a one second long Gaussian noise sequence r with a sampling rate of 300 Hz, where each sample is independent and identically distributed and drawn from a normal distribution, using the `randn` function.

- Taper the noise sequence r by setting its first and last 15 samples to zero.

- Use the `DSP.jl` function call

  ```
  digitalfilter(Lowpass(45, fs=300),
                FIRWindow(Windows.hamming(51)))
  ```

  to design a finite impulse response low-pass filter with a cut-off frequency of 45 Hz. This function will return a vector b for use in a digital filter of the type shown on slide 25. What vector a is required in addition?

  [*Note:* In MATLAB an equivalent function call is `fir1(50, 45/150)` and in Python `scipy.signal.firwin(51, 45/150)`.]

  Use the `filtfilt` function in order to apply that filter to the generated noise signal, resulting in the filtered noise signal x. (This function applies the filter twice, once in forward and once in backward direction.)

- Then sample x at 100 Hz by setting all but every third sample value to zero, resulting in the (equally long) sequence y.

- Implement sinc interpolation with a suitably scaled sinc function (and any required loops) to reconstruct the zeroed samples of y, resulting in a reconstructed sequence z.

- Generate another low-pass filter with `digitalfilter`, with a cut-off frequency of 50 Hz and apply it with `filtfilt` to y, resulting in interpolated sequence u. Multiply the result by three, to compensate the energy lost during sampling.

- Plot x, y, z and u, all on top of each other (superimposed) in one figure, and compare x with z and u.

- Lastly, turn the calls to `digitalfilter` and `Windows.hamming` into comments and replace them with your own implementations.

(*ii*) Why should the first filter have a lower cut-off frequency than the second?

(*c*) (*i*) Simulate the reconstruction of a sampled band-pass signal, with these steps:

- Generate a 1 s noise sequence r, as in part $(b)(i)$, but this time use a sampling frequency of 3 kHz. Set the first and last 500 samples to zero.

- Apply to that with `filtfilt` a band-pass filter that attenuates frequencies outside the interval 31–44 Hz, resulting in filtered sequence x. The following `DSP.jl` function call will design such a filter for you:

```
digitalfilter(Bandpass(31, 44, fs=3000),
              Chebyshev2(3, 30))
```

  This function returns a filter object that can be passed directly to `filt` or `filtfilt`. [*Hint:* To obtain a representation as coefficient vectors b and a, you can convert that filter object to type `PolynomialRatio` and then use functions `coefa` and `coefb` on that.]

  [*Note:* In MATLAB an equivalent function call is `cheby2(3, 30, [31 44]/1500)`, which returns *two* vectors b and a. In Python: `scipy.signal.cheby2(3,30, [31/1500,44/1500], "bandpass")`.]

- Then sample the resulting signal at 30 Hz by setting all but every 100-th sample value to zero, resulting in y.

- Implement sinc interpolation with a suitably scaled *and modulated* sinc function (and any required loops) to reconstruct an approximation of x from y, resulting in a reconstructed sequence z.

- Generate with

```
digitalfilter(Bandpass(30, 45, fs=3000),
              Chebyshev2(3, 20))
```

  another band-pass filter for the interval 30–45 Hz and apply it to y, to reconstruct the original signal as sequence u. (What factor do you have to multiply it by, to compensate the energy lost during sampling?)

- Plot x, y, z, and u, all on top of each other in one figure, and compare the original band-pass signal x and the two reconstructed versions z and u after being sampled at 30 Hz.

(*ii*) Why does the reconstructed waveform differ much more from the original if you reduce all the cut-off frequencies of all band-pass filters by 5 Hz?

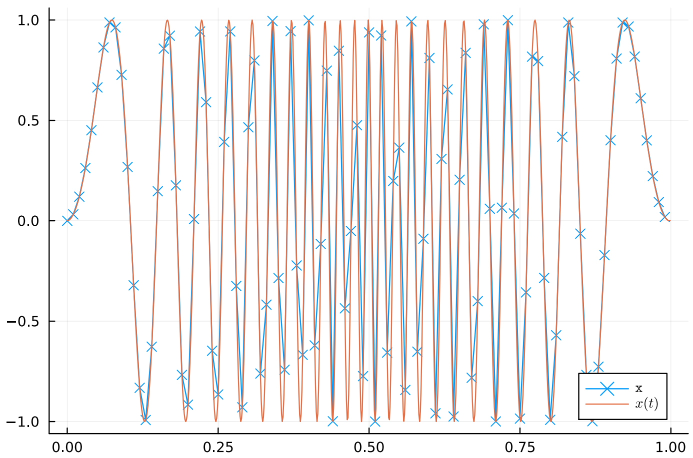
(*d*) Write a function `fft_interp(x, f)` that receives a vector `x` of real-valued floating-point numbers and an integer `f`. You can assume that the vector `x` contains the result of sampling a continuous function $x(t)$ that contains only frequencies less than half the sampling frequency used to obtain `x`. The function should return a vector that is `f` times longer than `x` and represents approximately the same continuous signal $x(t)$ as `x`, but sampled with a factor `f` higher sampling frequency.

In your function, use the Fast Fourier Transform to efficiently approximate sinc interpolation. After applying any necessary padding to `x`, use the `fft` function to convert that finite discrete sequence into a frequency-domain representation `X`. Then modify `X` such that it resembles the frequency-domain representation of the same signal $x(t)$ if it had been sampled at an `f`-times higher sampling frequency. Finally apply the `ifft` function to return to the time domain, remove any padding and return the result.

Generate two test signals `x` to demonstrate this signal, each 100 samples long:

(*i*) a unit impulse at the 10th sample, all other samples being zero;

(*ii*) a sine-wave of variable frequency that starts with value 0 and with a normalized frequency of 0 rad/sample, then increases during the first half of the signal (over 50 samples) in frequency linearly until it reaches a normalized frequency of 0.9π rad/sample, and then decreases again linearly down to 0 rad/sample during the second half of the signal.



For each of these test signals, prepare a plot that shows both `x` and `fft_interp(x, 8)` plotted on top of each other (superimposed) as lines, such that the corresponding samples are aligned with each other both horizontally and vertically.

END OF PAPER