Lab session 2: Signals and Systems

The problems 1, 2, and 3 are worth 10 points each. Problem 4, and 5 are worth 15 points each. Problem 6 is worth 25 points, and problem 7 is worth 5 points. You get 10 points for free, totalling 100 points.

From Themis, you can download a small C program called `demo.c` that shows how to read/write signals and allocate memory dynamically. You may use this code in your own implementations.

Problem 1: Linear filter

The input for this problem are a *kernel* $h[\,]$ of a FIR filter, and a discrete input signal $x[\,]$. Your output should be the signal $y[\,]$ that is obtained by feeding the FIR filter the input $x[\,]$. Your program should only show the samples of y where $x[\,]$ and $h[\,]$ have 'overlap' in the sliding window process (because you cannot print an infinite signal).

Example 1:

input:
```
2: [1,-1]
10: [0,0,0,1,1,1,1,0,0,0]
```
output:
```
11: [0,0,0,1,0,0,0,-1,0,0,0]
```

Example 2:

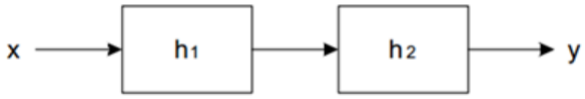
input:
```
10: [0,0,0,1,1,1,1,0,0,0]
2: [1,-1]
```
output:
```
11: [0,0,0,1,0,0,0,-1,0,0,0]
```

Example 3:

input:
```
3: [0,1,0]
3: [1,2,3]
```
output:
```
5: [0,1,2,3,0]
```

Problem 2: Cascade of two FIR filters

The input for this problem are the kernels $h_1[\,]$ and $h_2[\,]$ of two FIR filters, and a discrete input signal $x[\,]$. Your output should be the signal $y[\,]$ that is obtained by applying the filters as shown in the following diagram.



Example 1:

input:
```
2: [1,-1]
2: [1,-1]
10: [0,0,0,1,1,1,1,0,0,0]
```
output:
```
12: [0,0,0,1,-1,0,0,-1,1,0,0,0]
```

Example 2:

input:
```
2: [0,1]
3: [0,0,1]
4: [1,2,3,2]
```
output:
```
7: [0,0,0,1,2,3,2]
```

Example 3:

input:
```
1: [2]
2: [0,3]
4: [1,2,3,2]
```
output:
```
5: [0,6,12,18,12]
```

Problem 3: Series of FIR filters

This problem is similar to the previous problem, except that we place n FIR filters in a cascade (the previous exercise is the case $n = 2$). The first line of the input contains n. Next follow the n filter kernels h_1, h_2, .., h_n. The last input line is the input signal $x[\,]$. The output should be the signal $y[\,]$.

Example 1:

input:
```
2
2: [1,-1]
2: [1,-1]
10: [0,0,0,1,1,1,1,0,0,0]
```
output:
```
12: [0,0,0,1,-1,0,0,-1,1,0,0,0]
```

Example 2:

input:
```
5
2: [1,-1]
2: [1,-1]
2: [1,-1]
2: [1,-1]
2: [1,-1]
10: [0,0,0,1,1,1,1,0,0,0]
```
output:
```
15: [0,0,0,1,-4,6,-4,0,4,-6,4,-1,0,0,0]
```

Problem 4: FIR filter?

The input of this problem consists of a discrete input signal $x[\,]$ followed by the output $y[\,]$ that is obtained by feeding $x[\,]$ to some unknown system. Your program should determine if the system might be a FIR system. If it is not, the output should be `NO FIR`. If it is, your program should output the impulse reponse of the system.

Example 1:

input:
```
2: [1,-1]
5: [4,-2,2,-2,-2]
```
output:
```
4: [4,2,4,2]
```

Example 2:

input:
```
2: [1,-1]
5: [4,-2,1,-2,-2]
```
output:
```
NO FIR
```

Example 3:

input:
```
2: [1,-1]
4: [3,-1,-1,-1]
```
output:
```
3: [3,2,1]
```

Problem 5: 1D correlator

The input for this problem are a *template* $h[\,]$, and a discrete input signal $x[\,]$. Your output should be the *steady state* of the signal $y[\,]$ that is obtained by *correlating* the signal $x[\,]$ with the template $h[\,]$. We assume that the number of samples of the template h (notation $|h|$) is less than the number of samples of the input x. Recall that the *steady state* of the correlation is defined as:

$$y[d] = \sum_{i=0}^{|h|-1} x[i+d]h[i] \quad \text{for } 0 \leq d < 1 + |x| - |h|$$

[Note: you are not expected to implement the correlation using the convolution theorem (i.e. using FFTs). That will be the main topic of lab session 3.]

Example 1:

input:
```
3: [2,3,5]
10: [1,2,3,5,4,420,1,12,13,15]
```
output:
```
8: [23,38,41,2122,1273,903,103,138]
```

Example 2:

input:
```
3: [17,2,19]
10: [1,6,12,11,19,4,21,12,3,9]
```
output:
```
8: [257,335,587,301,730,338,438,381]
```

Example 3:

input:
```
2: [1,42]
10: [1,42,1,42,1,42,10,52,10,52]
```
output:
```
9: [1765,84,1765,84,1765,462,2194,472,2194]
```

Problem 6: 1D Pearson Correlator

As can be seen from the outputs in problem 5, direct correlation is often not very useful. We can see that clearly in the first example input. The pattern `[2,3,5]` is present in the signal at index location 1, but the correlation value at index 1 is only 38 (which is not the maximum of the correlation series). Moreover, the pattern is also present at the end of the input signal in the sequence `[12,13,15]` but a constant value (10) is added to it. A good template matcher should be insensitive for this, and therefore we rarely use direct correlation and apply *Pearson correlation* instead. The Pearson correlation has values between +1 and -1, where 1 means a total positive correlation (perfect match), 0 means no correlation at all, and -1 means total negative correlation (perfect correlation, but minima are maxima and vice versa). The Pearson correlation is defined as (see `https://en.wikipedia.org/wiki/Pearson_correlation_coefficient`):

$$y[d] = \frac{\sum_{i=0}^{|h|-1}(x[i+d] - \bar{x})(h[i] - \bar{h})}{\sqrt{\sum_{i=0}^{|h|-1}(x[i+d] - \bar{x})^2}\sqrt{\sum_{i=0}^{|h|-1}(h[i] - \bar{h})^2}}$$

Again, we assume x to be the signal, and h to be a (smaller) template. In the above formula, the sums therefore range from 0 to $|h| - 1$. Moreover, the notation $\bar{h}$ denotes the mean value of h (which is a constant for any d since we compute only the steady state), and $\bar{x}$ is the mean value of the samples from x that overlap with h given a value for d (and thus is not constant!).

Extend the code for problem 5 with a function `pearsonCorrelator1D`, which computes first a correlation (using `correlator1D` and next corrects the result such that we get the steady state of a Pearson Correlation. This involves some arithmetic (see for example the above wikipedia link). Note that the output of the direct correlation is an array

of integers, so you need to copy this result into an array of `doubles` before you start computing this correction. The input/output behaviour of your program must be the same as in problem 5, except that the output must be Pearson correlation values, rounded to 5 digits after the decimal dot.

Example 1:

input:

```
3: [2,3,5]
10: [1,2,3,5,4,420,1,12,13,15]
```

output:

```
8: [0.98198,1.00000,0.32733,0.94423,-0.19509,-0.74065,0.80296,1.00000]
```

Example 2:

input:

```
3: [17,2,19]
10: [1,6,12,11,19,4,21,12,3,9]
```

output:

```
8: [0.15959,-0.54127,0.67900,-0.92966,1.00000,-0.82675,-0.10762,0.90419]
```

Example 3:

input:

```
2: [1,42]
10: [1,42,1,42,1,42,10,52,10,52]
```

output:

```
9: [1.00000,-1.00000,1.00000,-1.00000,1.00000,-1.00000,1.00000,-1.00000,1.00000]
```

Problem 7: 1D template matching

The input for this problem has almost the same format as in the previous two exercises. The difference lies in the first line, which contains a floating point number T (threshold) which has a value from the range $-1 \leq T \leq 1$. The remaining two lines contains a template h and a signal x (both with positive numbers). The output of this problem should be a list of index locations where the Pearson Correlation is greater or equal to T, and the correlation value itself. For each location you should print one line of output.

Example 1:

input:

```
1.0
3: [2,3,5]
10: [1,2,3,5,4,420,1,12,13,15]
```

output:

```
1 1.00000
7 1.00000
```

Example 2:

input:

```
0.9
3: [17,2,19]
10: [1,6,12,11,19,4,21,12,3,9]
```

output:

```
4 1.00000
7 0.90419
```

Example 3:

input:

```
0.9
2: [1,42]
10: [1,42,1,42,1,42,10,52,10,52]
```

output:

```
0 1.00000
2 1.00000
4 1.00000
6 1.00000
8 1.00000
```