Lab session 3: Signals and Systems

The 3rd (and last) lab session consists of 7 problems. The problems 1, 3, 4, and 5 are worth 15 points. The problems 2, 6, and 7 are worth 10 points. You get 10 points for free, totalling 100 points.

Problem 1: 1D Convolution using FFT

The input and output for this problem are the same as for exercise 1 of lab session 2. The input consists of a *kernel* $h[\,]$ of a FIR filter, and a discrete input signal $x[\,]$. Your output should be the discrete signal $y[\,]$, which is the convolution of $h[\,]$ and $x[\,]$. The key difference with exercise 1 from lab 2 is that you have to implement the convolution using the *convolution theorem* and the *recursive* FFT algorithm as explained in the lecture slides of the 5th lecture. The samples of $h[\,]$ and $x[\,]$ are integer valued, so the output samples will also be integer valued. However, it is important to use double precision values in your calculations, and round the output samples to the nearest integer. You are advised to make use of the data type `double complex` which is available after including the header file `complex.h` at the top of your program. This way, you do not have to write complex arithmetic operators (like addition and multiplication) yourself.

Note: If you submit the same code as you did for exercise 1 of lab session 2, then you will pass a few test cases in Themis. However. the TAs (Teaching Assistants) will check the submissions manually, and if your submission does not use the FFT algorithm, then the submission will be rejected and is actually considered as fraud!

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: 1D Steady State Correlator using FFT

This problem is the same problem as exercise 5 of lab 2. However, this time you need to use the FFT in your implementation.

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:

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

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 3: 1D Steady State Pearson Correlator using FFT
This problem is the same problem as exercise 6 of lab 2. However, this time you need to use the FFT in your implementation.

Recall that Pearson correlation is defined as:
(see `https://en.wikipedia.org/wiki/Pearson_correlation_coefficient`):

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

Here, we assume x to be a 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!).
Use your code from the previous exercise to compute a steady state correlation, followed by a step that corrects the result such that we get the steady state of a Pearson Correlation. The output must be `double` 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]
```

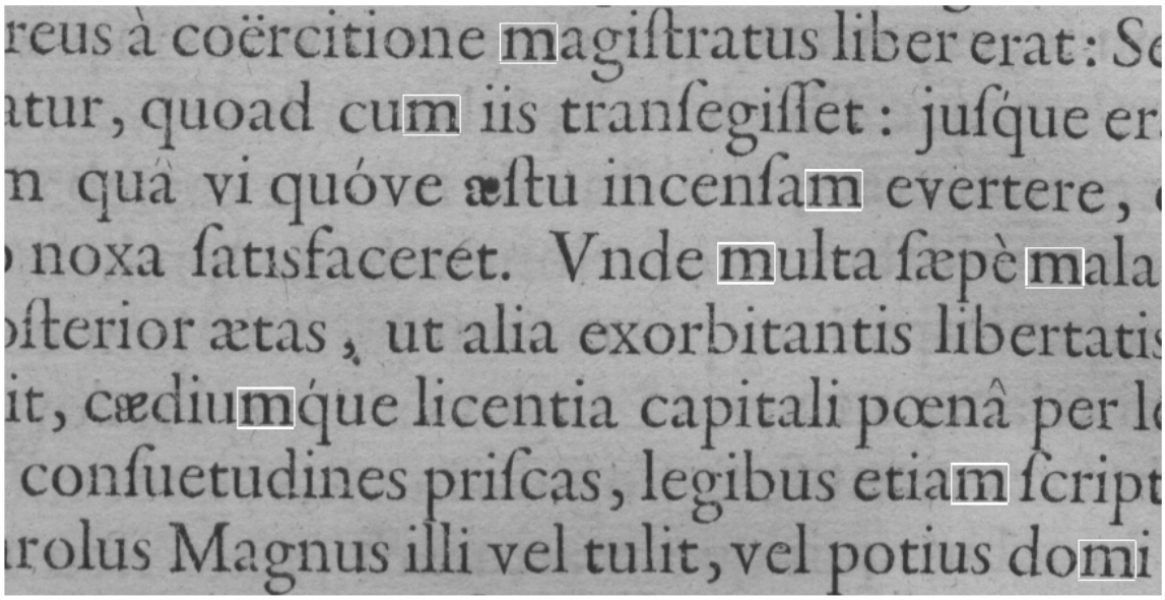
Example 3:
input:
```
2: [1,42]
10: [1,42,1,42,1,42,10,52,10,52]
```
output:
```
9: [1.00000,-1.00000,1.00000,-1.00000,1.00000,-1.00000,1.00000,-1.00000,1.00000]
```

Problem 4: mini OCR (Optical Character Recognition) using a 2D Pearson correlator
In this problem you will perform Pearson correlation on digital gray scale images. The input for this problem are a PGM (simple image format) image, and a small template image (also a PGM image). Most of the code has been written for you, and can be found in the file `pearson2D.c` on Themis. The input image that we will use is called `emmius.pgm`. It is taken from of a scan of a page from an old book (from 1614) written by the founder of the university of Groningen (Ubbo Emmius). The template is a letter (the letter 'm') from this page, and is called `M.pgm`. These two file names are hard-coded in the code of this exercise. Of course, you normally would not do that, but in this case it is necessary to pass the test in Themis. So, do not change these file names! You can display these images using the Linux command `display`. The objective of this exercise is to count (and of course, not by hand!) how many times the letter 'm' occurs in the scan. The number of occurrences must be printed on the output. Note that this problem has only one test case in Themis. It is considered cheating (we check this!) to submit a program that only prints a number without calculation. The file `pearson2D.c` is a completely functional program, except for the routine `fft2D`, which should compute the 2-dimensional FFT of an image. In the file you will see that the code of this routine looks like:

```
void fft2D(int direction, int width, int height, double **re, double **im) {
  /* width and height must both be powers of two! */
  /* Note that the parameters re and im denotes the REal part and
   * the IMaginary part of the 2D image.
   */
  printf("fft2D: YOU HAVE TO IMPLEMENT THE BODY OF THIS FUNCTION YOURSELF\n");
```

```
  printf("MAKE USE OF THE fft1D() FUNCTION\n");
  exit(0);
}
```

A nice property of the 2D FFT is that it can be implemented using only 1D FFTs. The 2D FFT can be computed by performing only 1D FFTs in two passes. In the first pass, we perform a 1D FFT on each row (real to complex) of the image. In the second pass, we perform a 1D FFT on each column resulting from the first pass (complex to complex). The same holds for the inverse 2D FFT. The one-dimensional FFTs have been implemented for you, and are already present in the file `pearson2D.c`. Note that this implementation of the 1D FFT is not recursive, but that is not important. Once you implemented the function `fft2D` correctly, you can run the program. The output is the number of occurrences of the letter 'm'. Moreover, as a side-product, the program produces the file `matches.pgm`. Use the program `display` to see the result. It should look like:



Problem 5: Forward Number Theoretic Transform

In the lecture about the FFT algorithm, it was discussed that it is also possible to compute FFT's in other fields than the complex numbers. In this problem, you will change the FFT code that you made for exercise 1 into a version that computes in the field of integers modulo a prime. This version is called an NTT (Number Theoretic Transform).

The result of the forward transform is not directly useful in the sense that it returns a spectrum. However, it is useful in combination with the convolution theorem (which we will use in problem 7).

In this problem, we will be using the prime 40961. Moreover, it is given that $\omega = 243$ is a *primitive 8192th root of unity* in the ring modulo 40961, i.e.

$$243^{8192} \bmod 40961 = 1 \quad \text{and} \quad 243^i \bmod 40961 \neq 243^j \bmod 40961 \text{ for all } 0 \leq i < j < 8192$$

There is a small demo program on Themis (file `demopowmod.c`) which contains a handy routine to compute efficiently $a^b \bmod c$. Once you implemented the transform, use the convolution theorem to implement the convolution algorithm which runs in $O(N \log N)$ time.

Note that the test sets in Themis are chosen such that overflow should not be an issue as long as you use `unsigned ints` (so not plain `ints`). The length of the inputs are powers of two, and do not exceed 8192 elements. Moreover, the test inputs do not contain any negative numbers.

Example 1:

input:
```
1: [42]
```
output:
```
1: [42]
```

Example 2:

input:
```
2: [1,2]
```
output:
```
2: [3,40960]
```

Example 3:

```
4: [1,2,3,4]
```
output:
```
4: [10,11877,40959,29080]
```

Problem 6: Inverse Number Theoretic Transform

Of course, we also want to have an inverse NTT. This exercise concerns making the inverse NTT under the same conditions as the previous exercise.

Example 1:

input:

```
1: [42]
```

output:

```
1: [42]
```

Example 2:

input:

```
2: [3,40960]
```

output:

```
2: [1,2]
```

Example 3:

```
4: [10,11877,40959,29080]
```

output:

```
4: [1,2,3,4]
```

Problem 7: 1D convolution via the Number Theoretic Transform

Use the convolution theorem, the forward NTT, and the inverse NTT to make a convolution algorithm that does not use complex numbers (only integers).

The test sets in Themis are chosen such that the length of the convolution does not exceed 8192. Moreover, the inputs do not contain any negative numbers, and the output samples do not exceed 40960.

Note: Of course, you are not allowed to submit your solution for problem 1 to Themis as a solution for this problem. If you do this anyway, this will be detected by the teaching assistants and reported as fraud!

Example 1:

input:

```
3: [1,0,1]
5: [1,1,0,1,1]
```

output:

```
7: [1,1,1,2,1,1,1]
```

Example 2:

input:

```
4: [1,0,1,1]
5: [1,1,0,1,1]
```

output:

```
8: [1,1,1,3,2,1,2,1]
```

Example 3:

```
5: [1,1,0,1,1]
5: [1,1,0,1,1]
```

output:

```
9: [1,2,1,2,4,2,1,2,1]
```