October 20, 2006

MAT-INF 1100: Compulsory Assignment 2

Deadline: November 3, 2006, 14:30

Information

The written answers are to be handed in at the office of the Maths Department on the 6th floor (7. etasje in Norwegian) by 14:30 on Friday, November 3. For Javaprograms and output from programs, printouts should be handed in, but for every problem you should also hand in a description with comments on your results. This should be written by yourself (by hand or computer). In your home directory there should be an unprotected and executable version of your programs, and the address of your home directory and the names of the programs should be included in your written answers.

Students who become ill or for other reasons wish to apply for postponement or exemption from this assignment should contact Elisabeth Seland (room B726 in Niels Henrik Abels building, telephone 22 85 59 07, email: elisabh@math.uio.no) in good time before the deadline.

Students are encouraged to work together on this assignment, and the teaching assistants will answer general questions, but cannot provide complete solutions. *The final answers that you hand in must be produced by yourself, and you must be able to explain the contents of your answers if you are called in for an oral examination (may happen if there is suspicion of copying).*

Note that there are some tips for problem 3 at the end of this text. Making use of the advice given there will simplify your programming job.

Remember that both of the two compulsory assignments in MAT-INF 1100 must be passed before you are allowed to sit the final course exam. *To pass this assignment all the problems must be reasonably well answered .*

Problem 1. Filtering of sound

In this problem you are to program some simple filter operations on sounds. To aid you in the programming you may use the example program found

in the file `Filters.java`. This program will show you how to use the class `MatInf1100Sound` (you will have copies of both programs if you have executed the `wget`-command on the home page of this assignment.) The program `Filters.java` is very simple and should give you enough information to solve the problems below.

The class `MatInf1100Sound` contains a good deal of code for manipulating sound, but there is not much you need to know about this class. What you do need to know is that the constructor of `MatInf1100Sound` has the signature

```
MatInf1100Sound(short [] samples, double sampleRate)
```

In other words, it has two parameters `samples` (which contains the sample values of the sound), and `sampleRate` which contains the sample rate. These two quantities are also available via the dot-notation, that is if `f` is an object of type `MatInf1100Sound` thenl `f.samples` will give the samples of `f` and `f.sampleRate` will give you the sample rate of `f`. As you will gather from `Filters.java` the class also contains the methods `getSoundFromFile(string)` and `getSampleSound()` which you may find useful for reading a sound from a file or generating the string sound. You should be familiar with this from a problem earlier in the course.

Please note that the sound-format in the `MatInf1100Sound`-class uses samples of type `short` (which is what is usually used on CDs), and not floating point numbers in the interval $[-1, 1]$ as assumed in chapter 4 of the lecture notes. You therefore have to make sure that your data are converted to `short` before playing the sound and that the noise you add is scaled so that it has an effect. (The values of a `short` may vary in the interval $[-32768, 32767]$.) Some sound-files that you may use for testing come with the `wget`-command.

a) Write a filter that adds noise to a given sound as suggested on page 59 in the lecture notes. Adjust the factor 0.2 so that the noise is distinctly audible. For information about generating random numbers you may want to take a look at the program `testrandom.java` which you can download using the `wget`-command. Describe briefly the audible effect that this filter has on the sound.

b) Write a low-pass filter that dampens the high frequencies as suggested on page 60–61 in the lecture notes by replacing a sample by a weighted average of the sample itself and its four neighbours on either side. As coefficients you should use the numbers in row 8 of Pascal's triangle

divided by the sum of the numbers, in other words a natural generalisation of the last two filters on page 61. Be careful near the ends of the signal. Briefly describe the audible effect the filter has on the sound.

c) Write a high-pass filter that dampens the low frequencies as suggested on page 62 in the lecture notes, but replace the filter by a filter of the same length as in problem 1b). Use coefficients with same value as in 1b), but let their sign alternate between positive and negative. Describe briefly the audible effect the filter has on the sound.

d) (Not compulsory!) Write a program that plays the 8 tones of the A-minor scale. Use a sample rate of 8000, and let each tone last for 0.4s with a pause of 0.1s between each tone.

Problem 2. Manual multiresolution analysis

In this problem we are going to manually perform a multiresolution analysis. This will be useful background knowledge for the programming in problem 3. A good start is to read Chapter 10 in the lecture notes.

a) We have a data set $(x_i, c_i)_{i=0}^{8}$ given by

$$x_i = i - 4 \quad \text{for } i = 0, \ldots, 8$$

and

$$c_i = x_i^2 \quad \text{for } i = 0, \ldots, 8.$$

In other words we have 9 points taken from the graph of the parabola $y = x^2$ on the interval $[-4, 4]$. We use the notation of Section 10.2.1 in the lecture notes and let $f(x) = L_1(c_0, \ldots, c_8)$ denote the piecewise linear function that satisfies the conditions $f(x_i) = c_i$ for $i = 0, \ldots, 8$.

Use Lemma 10.2 and decompose f into an approximation g and an error function e according to the formulas (10.1)–(10.4). Sketch the three functions f, g and e.

b) Set $g_1 = g$ and $e_1 = e$, and decompose g_1 in the same way as $g_1 = g_2 + e_2$. Then decompose g_2 as $g_2 = g_3 + e_3$. We then have $f = g_3 + e_3 + e_2 + e_1$. Sketch the functions g_3, e_3 and e_2.

c) For compression we set to zero all coefficients in the error function that are smaller than a given tolerance ϵ, as described in sections 10.4 and 10.5 in the notes. Try first with the tolerance $\epsilon = 0.1$ and then with

the tolerance $\epsilon = 1$. In each case you will end up with slightly perturbed error functions that we call $\tilde{e}_i$ for $i = 1, 2, 3$. Compute the two corresponding approximations to f given by $\tilde{f} = g_3 + \tilde{e}_3 + \tilde{e}_2 + \tilde{e}_1$, sketch these and compare with f. How much compression did you get in each of the two cases?

Problem 3. Sound compression

In this problem you are to program a version of lossy compression of a sound signal as suggested in chapter 10 of the lecture notes. You should therefore first read this chapter.

The problem consists of programming the decomposition as suggested by the formulas (10.3) and (10.4) in chapter 10, and setting those of the ws that have their absolute value smaller than a given tolerance (that you choose yourself) to zero. You should also program the reconstruction given by formulas (10.5) and (10.6).

Your programs should first be tested on the simple data set from problem 2. When this works you can try a test sound by decomposing the test sound and saving the result to a file, using the program `gzip` for encoding the file efficiently and comparing the size of the encoded file with the size of the original test sound. You should also reconstruct the changed sound, play it, and give a subjective opinion of the quality. You should have at least one such test example where the size of the compressed file should be considerably less than the size with the test sound, but at the same time it should not be difficult to recognise the test sound from the compressed sound.

a) Program a simplified decomposition algorithm where you only decompose once, such that f is split into $f = g_1 + e_1$. You should also program the reconstruction algorithm by implementing the formulas (10.5) and (10.6). It is probably wise to first program the case where the number of samples is an odd number, and then extend your program to handle the case of even numbers afterwards. You may either read the tolerance from the terminal or leave it as a constant in your program.

Test your decomposition and reconstruction routines on the parabola data with the help of the program `test_parabola.java`, with the tolerance equal to 0, 3 and 16. Check that the results agree with what you computed by hand in problem 2 and report your results. Then test your routines with a sound, using the program `CompressionTester.java` and give your opinion on the quality.

b) (Not compulsory.) Write a complete decomposition program which decomposes your signal k times (you must find a suitable value for k yourself) and sets all w_i which are less than the tolerance in absolute value to zero. You should also program the corresponding reconstruction algorithm. Remember that the straightforward implementation of the algorithms in chapter 10 only work for very special lengths of the data set. Section 10.3.5 discusses how to handle this.

Some tips

By making use of code that has already been written, your work with this problem will be reduced to a fraction of what it would otherwise be. If you have executed the `wget`-command shown on the home page of this assignment, you have obtained a collection of Java-programs. For solving this problem you may find the files

```
AbstractCompressedSound.java
Compressed1100Sound.java
CompressionTester.java
MatInf1100Sound.java
forslaga.java
forslagb.java
test_parabola.java
```

useful. The file `Compressed1100Sound.java` contains a very coarse decomposition method, but will work with `CompressionTester.java`. The files `forslaga.java` and `forslagb.java` contain skeletons that may prove useful for solving problems 3a and 3b. Note that your final solution must have the name `Compressed1100Sound.java` in order to work with `CompressionTester.java`. For this reason it is best to place the solutions for 3a and 3b in different file directories.

For problem 3a, the class should contain a contructor that decomposes a sound signal once, a reconstruction method and the four variables

```
short d[]; //Samplene til g
short w[]; // Samplene til feilfunksjonene
double sampleRate;
short eps; //Toleranse
```

The constructor requires a parameter of type `MatInf1100Sound` and should decompose the sound as suggested by the algorithm given by formulas (10.3)

and (10.4) in the lecture notes. The file `forslaga.java` contains more information.

To solve problem 3b, your class should contain a constructor that decomposes the signal several times, a method that reconstructs a signal, and the four variables

```
short d[]; //Samplene til
short w[][]; // Feilfunksjonen(e)
double sampleRate;
short eps; //Toleranse
```

In other words, the only difference from problem 3a is that we now have an array of samples of error functions. The file `forslagb.java` contains more information.

It is simplest to just have one version of `Compressed1100Sound.java` in the directory where you are working. It may therefore be easiest to have one directory for each of the two problems.

For testing you may use the program `CompressionTester.java`. This program will generate a short test sound (or read a sound from file) in `MatInf1100Sound`-format and write this sound to the file `original.obj`. It will then call the constructor in `Compressed1100Sound` and thereby perform the decomposition algorithm written by you, write the decomposed sound to the file `Compressed1100Sound.obj`, perform the reconstruction and play the reconstructed signal. You should then use the program `gzip` and do a lossless compression of the file that was generated, as described in the instructions in the file `CompressionTester.java`.

Note that if you try to use a test sound that last more than a few seconds you have to give Java more memory when you run the program. Information on how this is done can be found at the beginning of the file `CompressionTester.java`.

Thank you Pål!

Finally: Thank you to Pål Hermun Johansen (Ph.D-student in Mathematics) who has written the code that you will be using here and that hopefully will prove useful to you.

Good luck!!