Project: application of FFT to filtering the noised signal

Tomasz Chwiej

5th February 2025

1 Introduction

Among many applications of FFT the filtering of noised data is one of the simplest to conduct. Suppose we have gathered N samples of signal which contains contributions from the pure periodic function $f_{pure}(x)$ and from the noise

$$f(x) = f_{pure}(x) + r, \quad \text{r-noise} \tag{1}$$

$$f_{pure}(x) = f_{pure}(x+L), \quad \text{L-period in position space} \tag{2}$$

In order to get rid of the noise form the collected data we first calculate the FFT

$$f(k) = FFT\{f(x)\} \tag{3}$$

and then, assuming the periodic part dominates over the noise contribution in a transformed signal, find the maximum of power spectra in a reciprocal space

$$p_{max} = \max |f(k)|^2, \quad k = 0, 1, \ldots N-1 \tag{4}$$

and remove all data which contributions are less than a fraction ($\alpha < 1$) of this maximum

$$f(k) = \left\{ \begin{array}{ccc} f(k) & \Longleftrightarrow & |f(k)|^2 \geq \alpha \cdot p_{max} \\ 0 & \Longleftrightarrow & |f(k)|^2 < \alpha \cdot p_{max} \end{array} \right. \tag{5}$$

In the last step the data must be transformed back to real space which is done by calculating the inverse FFT. The output data shall be denoised. The aim of this project is to follow this route and to denoise the sampled data.

2 Practical part

Write the computer program which takes the following steps:

1. Generate the noised data.
   In calculations use following parameters: $N = 2^{10}$, $L = 4$ (period in position space), $\Delta_x = \frac{L}{N}$ - interval for real space nodes. The pure periodic signal is given by following expression

$$f_{pure}(x) = \sin(5\omega\, x) + \sin(15\omega\, x) + \sin(25\omega\, x) \tag{6}$$

   At each real space node

$$x_i = \Delta_x \cdot i, \qquad i = 0, 1, 2, \ldots, N-1 \tag{7}$$

   generate the noise using random number generator

$$r = 10 \cdot \frac{(double)rand()}{(double)RAND\_MAX} \tag{8}$$

function **rand()** and constant **RAND_MAX** are defined in standard library(stdlib.h in C and cstdlib in C++). Generate the sequence of noised data

$$f(x_i) = f_{pure}(x_i) + r, \quad i = 0, 1, 2, \ldots, N-1 \tag{9}$$

Write the noised data to a file and prepare the figure showing them.

2. Save the input data in 1D array and calculate the FFT using the routines **fftw_plan_dft_1d()** and **fftw_execute()** from FFTW numerical package (see Hints). Write the transformed data to file, draw the real and the imaginary parts of $f(k)$ in separate figures.

3. Denoise the data.
   Find the maximum of power spectra for the transformed data accordingly with Eq.4, remove the noise contribution (Eq. 5) and calculate the inverse FFT (see Hints). Write the output data to a file and draw the figure (only the real part) showig the denoised data.

At home prepare the raport. Repeat the noise filtering for $N = 2^{12}$ samples (other parameters leave unchanged). Compare both results, for $N = 2^{10}$ and $N = 2^{12}$, and try to assess which are the better, justify your answear.

3 Computational hints

In project we use two routines contained in FFTW3 numerical library (Fastest Fourier Transform in the West, version 3). The library must be linked during compilation, e.g. for C code

```
gcc *.c -lfftw3 -lm
```

or C++

```
g++ *.c -lfftw3 -lm
```

Although, FFTW defines its own complex number data types, we still may use these defined in C or C++, the compiler must only read the C/C++ header for the complex numbers first and after that read the header **fftw3.h**, for C code

```
#include<complex.h>
#include<fftw3.h>
```

and for C++ code

```
#include<complex>
#include<fftw3.h>
```

To calculate the FFT for our data three steps must be taken

- prepare the plan, in this phase the routine **fftw_plan_dft_1d(...)** is informed what type of transformation (forward or backward/inverse) must be conducted, it also decides which algorithm should be used
- perform FFT calculations with **fftw_execute(...)**
- deallocate internal memory with **fftw_destroy(...)**

Following piece of code show how to use it in **C**

```
#include<complex.h>
#include<fftw3.h>

int N;
double complex *in, *out;

in=(double complex *) malloc(N*sizeof(double complex));
out=(double complex *) malloc(N*sizeof(double complex));

fftw_plan plan = fftw_plan_dft_1d(N,in,out, FFTW_FORWARD, FFTW_ESTIMATE);
fftw_execute(plan);
fftw_destroy_plan(plan)

free(in);
free(out);
```

In **C++** code one may use C arrays allocated with malloc, or use more flexible vector type arrays (automatically deallocated) but then the pointer to the first element in vector must be passed

```
#include<complex>
#include<fftw3.h>
#include<vector>

int N;
vector<complex<double>> in(N);
vector<complex<double>> out(N);

fftw_plan plan = fftw_plan_dft_1d(N, (fftw_complex *)&in[0],
                                       (fftw_complex *)&out[0],
                                       FFTW_FORWARD, FFTW_ESTIMATE);
fftw_execute(plan);
fftw_destroy_plan(plan)
```

in both examples the routine takes the data from input array **in** and write the transformed output to an array **out**. If these arrays are different, then FFT is made out-of-place (input array is untouched), otherwise, for the same input and output array, the data in array are overwritten by FFT. Finally, after denoising the signal we want to calculate the inverse FFT, this is done by calling the routine **fftw_plan_dft_1d** with parameter **FFT_BACKWARD**.
And the final remark: the calculated FFT is not normalized, the user must multiply the forward FFT by $1/N$ itself.

4 Example results

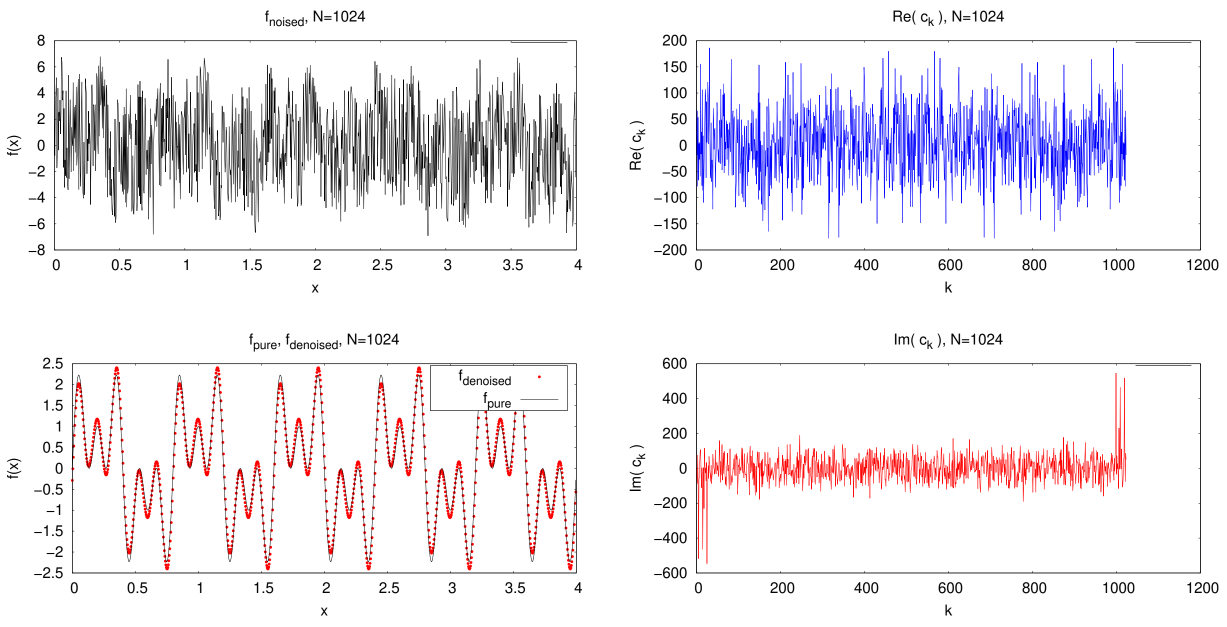

Figure 1: The noised signal (top-left) and signal after denoising (bottom-left). In the right column there are shown the real part (top-right) and the imaginary part (right-botom) of the Discrete Fourier Transform of noised signal (FFT).