Homework 3 Computational Physics

Giacomo Recine[*]

21 October 2025

Contents

[*]gr2640@nyu.edu

1 Introduction

In this homework, we practice using the Fourier transform. First, in Sec. 2, we implement the discrete Fourier transform by hand using a dataset that contains the observed number of sunspots on the Sun. Then, in Sec. 3, we work with the Fast Fourier Transform (FFT), which is particularly useful because it is much faster than the classical Fourier transform. In this section, we also study how the inverse FFT behaves when the input data is reduced, exploring the effects of truncating the Fourier coefficients. Finally, in Sec. 4, we apply the so-called convolution theorem to a practical example: restoring a blurred photograph using a point spread function, which we model as a Gaussian.

2 Exercise 7.2 from Newman

In this exercise, we will learn how to implement a discrete Fourier transform by hand. First of all, we will begin with a brief review of the theory behind this very useful technique.

2.1 Discrete Fourier Transform (DFT)

It is well known that a periodic function $f(x)$ defined on a finite interval $0 \leq x < L$ can be written as a Fourier series can be expressed in terms of its Fourier series, meaning that we can write

$$f(x) = \sum_{k=-\infty}^{\infty} \gamma_k \exp\Big(i\frac{2\pi kx}{L}\Big), \tag{1}$$

where γ_k are, in general, complex coefficients. These coefficients can be determined as

$$\gamma_k = \frac{1}{L}\int_0^L f(x) \exp\Big(-i\frac{2\pi kx}{L}\Big)\, dx. \tag{2}$$

Thus, given the function $f(x)$, we can compute the Fourier coefficients γ_k; conversely, given the coefficients γ_k, we can reconstruct $f(x)$.

Anyway, for some functions $f(x)$, the integral in Eq. (2) can be evaluated analytically, allowing the Fourier coefficients γ_k to be computed exactly. However, in many cases this is not possible: the integral may be too difficult to solve, or $f(x)$ may not even be known in an analytical form, it might be a signal measured in a laboratory experiment or the output of a computer program. In such situations, we can instead compute the Fourier coefficients numerically.

Using the trapezoidal method, with N slices of width $h = L/N$, which we have learned in the past to evaluate integrals numerically, we can write

$$\gamma_k = \frac{1}{L}\cdot\frac{L}{N}\bigg[\frac{1}{2}f(0) + \frac{1}{2}f(L) + \sum_{n=1}^{N-1} f(x_n) \exp\Big(-i\frac{2\pi kx_n}{L}\Big)\bigg], \tag{3}$$

where the positions x_n of the sample points for the integral are

$$x_n = \frac{n}{N}L, \quad n = 0, 1, \ldots, N. \tag{4}$$

Since $f(x)$ is, by hypothesis, periodic, we have $f(L) = f(0)$, thus the Eq. (3) simplifies to

$$\gamma_k = \frac{1}{N}\sum_{n=0}^{N-1} f(x_n) \exp\Big(-i\frac{2\pi kn}{N}\Big), \tag{5}$$

and we can use this formula to evaluate the Fourier coefficients γ_k on a computer.

A simpler way to write Eq. (5) in such situations is to define $y_n = f(x_n)$ as the values of the N samples to write

$$\gamma_k = \frac{1}{N}\sum_{n=0}^{N-1} y_n \exp\Big(-i\frac{2\pi kn}{N}\Big). \tag{6}$$

In this form, the equation does not require us to know the positions x_n of the sample points or even the width L of the interval, since neither enters the formula. All we need to know are the sample values y_n and the total number of samples N. The sum above is a standard quantity known as the discrete Fourier transform (DFT) of the samples y_n, which we denote as c_k

$$c_k = \sum_{n=0}^{N-1} y_n \exp\Big(-i\frac{2\pi kn}{N}\Big), \tag{7}$$

which differs from γ_k only by a factor of $1/N$. Computing c_k allows us to obtain the exact Fourier transform of the sampled data. Moreover, it can be shown that from the coefficients c_k, it is possible to recover the original sample values y_n. In particular:

$$y_n = \frac{1}{N}\sum_{k=0}^{N-1} c_k \exp\Big(i\frac{2\pi kn}{N}\Big). \tag{8}$$

This result is called the inverse discrete Fourier transform (IDFT) and it is the counterpart of the forward transform.

2.2 Solution of the exercise

We now apply the theory briefly introduced to the first practical exercise.

2.2.1 Point a

We consider the file `sunspots.txt`, which contains the observed number of sunspots on the Sun for each month since January 1749. The file consists of two columns: the first represents the month index, and the second gives the corresponding sunspot number. Plotting the data, we obtain

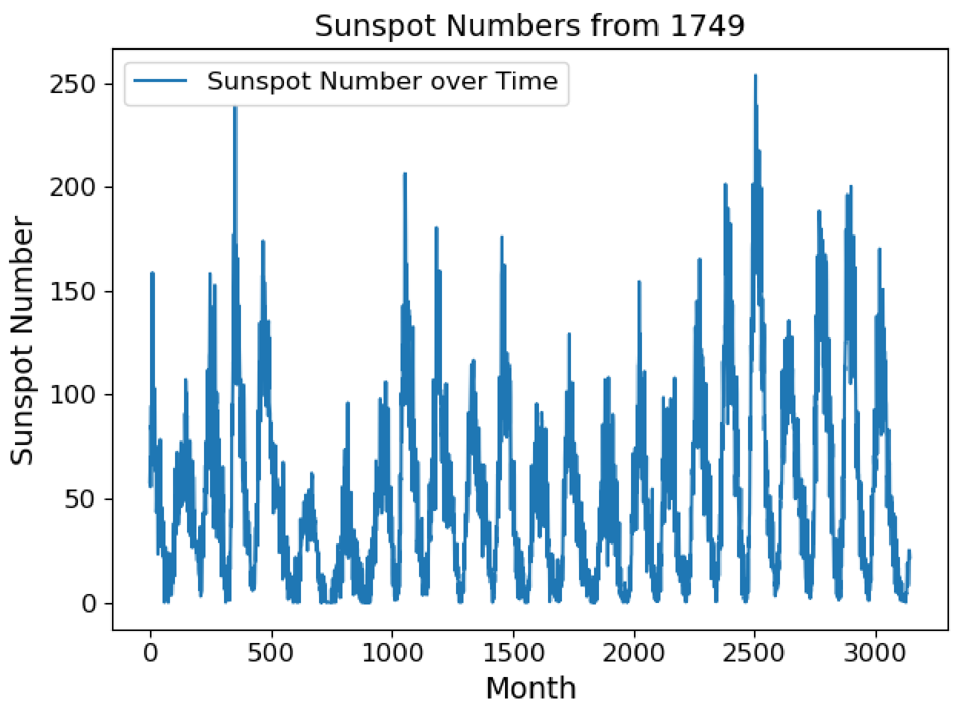

Figure 1: Plot of the data contained in the file.

We immediately observe that the number of sunspots fluctuates in a regular cycle over the entire observation period. A first rough estimate of the cycle length can be obtained by measuring the distance between two consecutive minima, for example. By zooming in on the previous graph for the months in the range $[990, 1190]$, we obtain

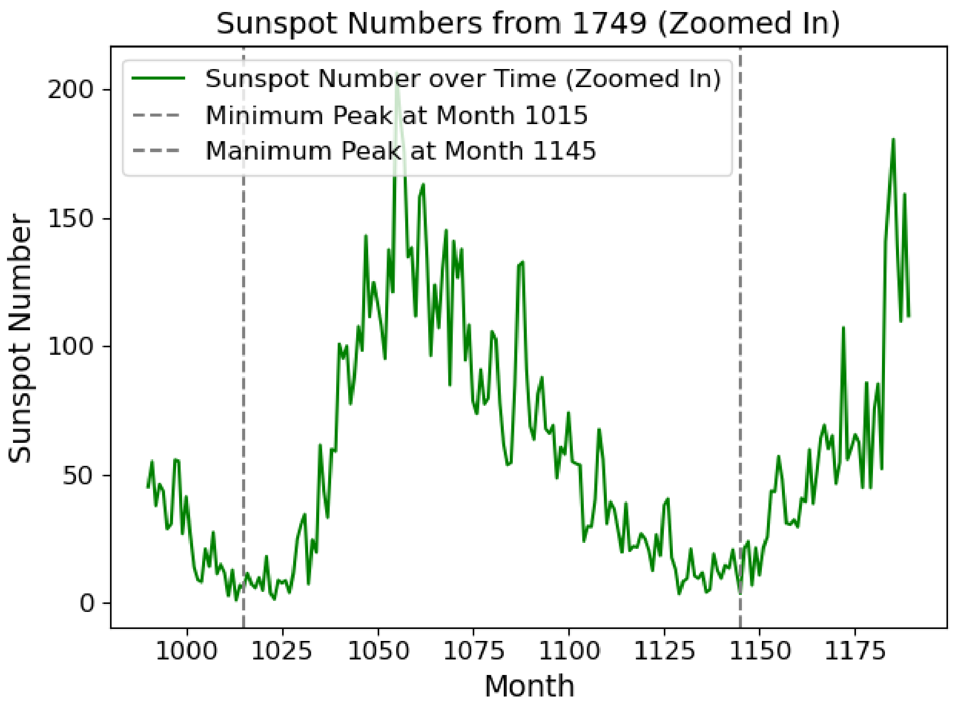

Figure 2: Zoom of the data to estimate the cycle length.

Then, a rough estimate of the cycle length is

$$T = (1145 - 1015)\,\text{months} = 130\,\text{months}. \tag{9}$$

We will check whether this estimate is compatible with the value obtained from the Fourier coefficients.

2.2.2 Point b

We have computed the Fourier transform by calculating c_k as given by Eq. (7). However, it is more informative to consider $|c_k|^2$, the squared magnitude of the Fourier coefficients, as a function of k. This quantity is also known as the power spectrum of the sunspot signal, and plotting this in function of k we have

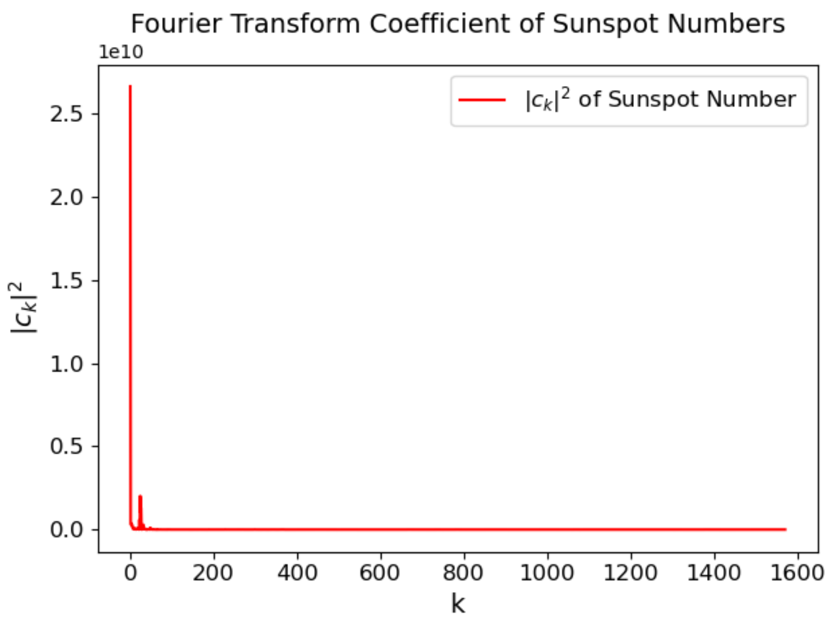

Figure 3: Fourier coefficients $|c_k|^2$ for the sunspot data.

We notice two notable peaks in the Fourier spectrum: one at $k = 0$ and another at a non-zero k. The more important one is the non-zero peak, which indicates that there is a specific frequency in the Fourier series with a significantly higher amplitude than the others. This means that a large sinusoidal term with this frequency dominates the signal, corresponding to the periodic oscillations visible in the original sunspot data. To estimate the value of this frequency, we can zoom into the region of the spectrum where this peak occurs (as seen in the plot) and determine graphically the value of k at which the amplitude $|c_k|^2$ reaches its maximum. Graphically, we have

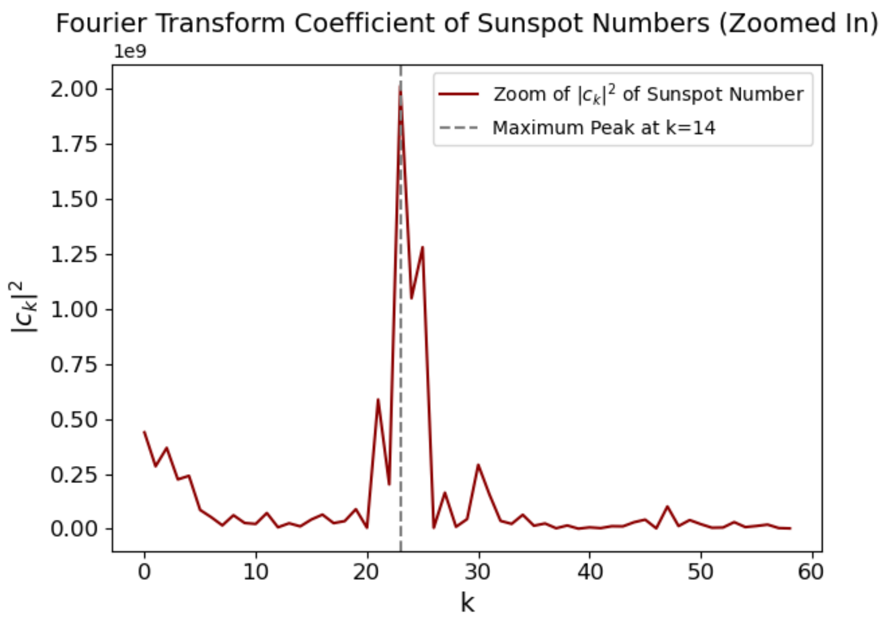

Figure 4: Zoom of Fourier coefficients $|c_k|^2$ plot in order to estimate the value of $k \neq 0$ for which we have the maximum.

The value of k corresponding to the peak is

$$k = 23, \tag{10}$$

and, in particular, from the definition of the frequency $f = k/N$, we can obtain the period (or cycle length) as

$$P = \frac{1}{f} = \frac{N}{k} = 136.65\,\text{months}. \tag{11}$$

This value is comparable with the rough estimate T obtained in the solution of point (a), as expected.

3 Exercise 7.4 from Newman

Through this exercise, we will learn how to implement the Fast Fourier Transform (FFT) using Python. Before doing so, let us first review the theory behind the Fast Fourier Transform.

3.1 Fast Fourier Transform

We saw in the previous chapter how to compute the Discrete Fourier Transform, in particular it is given by the Eq. (7). Now, there is a clever trick for calculating the DFT much faster than is possible than by directly evaluating the Eq. (7). This trick is called the fast Fourier transform (FFT). It was discovered by Carl Friedrich Gauss in 1805, when he was 28 years old. This new method is simpler when the number of samples is a power of two, so let us consider the case where $N = 2^m$, with m integer.

Consider the sum in the DFT equation, Eq. (7), and let us divide the terms into two equally sized groups, which we can always do when N is a power of two. Let the first group consist of the terms with n even and the second the terms with n odd. We consider the even terms first, i.e., the terms where $n = 2r$ with integer $r = 0, \ldots, \frac{N}{2} - 1$. The sum of these even terms is

$$E_k = \sum_{r=0}^{N/2-1} y_{2r} \exp\left(-i\frac{2\pi k(2r)}{N}\right) = \sum_{r=0}^{N/2-1} y_{2r} \exp\left(-i\frac{2\pi kr}{N/2}\right). \tag{12}$$

But this is simply another Fourier transform, just like Eq. (7), but with $N/2$ samples instead of N. Similarly, the odd terms, meaning those with $n = 2r + 1$, sum to

$$\sum_{r=0}^{N/2-1} y_{2r+1} \exp\left(-i\frac{2\pi k(2r+1)}{N}\right) = \exp\left(-i\frac{2\pi k}{N}\right) \sum_{r=0}^{N/2-1} y_{2r+1} \exp\left(-i\frac{2\pi kr}{N/2}\right) = \exp\left(-i\frac{2\pi k}{N}\right) O_k. \tag{13}$$

where in this latter case O_k is another Fourier transform with $N/2$ samples. Hence, by these definitions, the Fourier coefficients will be the following sum

$$c_k = E_k + e^{-i2\pi k/N} O_k, \tag{14}$$

We have found that the Fourier coefficients are the sum of two terms, E_k and O_k, each of which is itself a discrete Fourier transform of the same function $f(x)$, but with half as many points spaced twice as far apart. The second term includes an additional factor $e^{-i2\pi k/N}$, called a twiddle factor, which is straightforward to compute. Therefore, if we can compute the two smaller Fourier transforms, we can easily obtain the coefficients c_k. How do we compute these smaller Fourier transforms? We apply the same procedure recursively: each transform is divided into its even and odd components, which can then be combined as the sum of two smaller transforms, with a twiddle factor applied in between. Because N is a power of two, this halving process can be repeated until we reach transforms that consist of only a single data point. For a transform containing just one sample, there is only a single Fourier coefficient, c_0. So, by taking $k = 0$ and $N = 1$ in Eq. (7), we get

$$c_0 = \sum_{n=0}^{0} y_n e^0 = y_0. \tag{7.40}$$

In other words, at the level of a single sample, the Fourier transform is trivial: it is equal to the value of the sample itself. Thus, no further computation is necessary, and we already have all the coefficients we need.

Without going into the details, the main advantage of this method is its speed compared to the direct computation of the discrete Fourier transform. The total number of coefficients that need to be calculated is equal to the number of levels, $\log_2 N$, times N coefficients per level, giving a total of $N \log_2 N$ coefficients. This is significantly more efficient than the approximately N^2 terms required for a brute-force computation of the DFT. Another useful observation is that the inverse discrete Fourier transform, which converts the Fourier coefficients back into the original samples, has essentially the same form as the forward transform; the only difference is the absence of a minus sign in the exponential. This implies that the inverse transform can also be computed efficiently using the same techniques as for the forward transform. The resulting procedure is naturally called the inverse fast Fourier transform, or inverse FFT.

Finally, in these developments we have assumed that the number of samples is a power of two, but it is possible to implement the fast Fourier transform for cases where the number of samples is not a power of two, but the algebra becomes more cumbersome, so we will omit the details here.

3.2 Solution of the exercise

3.2.1 Point a

We are now ready to tackle the exercise. We use the data contained in the file `dow.txt`, which includes the daily closing values for each business day from late 2006 until the end of 2010 of the Dow Jones Industrial Average, a measure of average stock prices on the US market. We can then read the data and plot it, obtaining the following

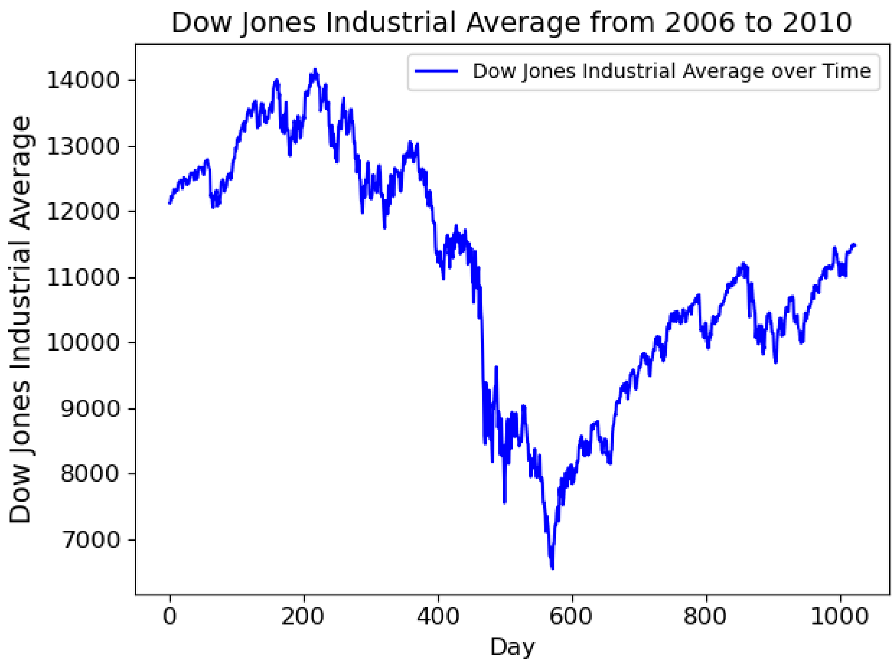

Figure 5: Plot of the data contained in the file under consideration.

3.2.2 Point b, c, d and e

We now compute the Fast Fourier Transform of the data using the Python function `rfft` from `numpy.fft`, which returns an array of $N/2 + 1$ complex numbers. In our case, this produces an array of length 513, as expected, since $N = 1024$.

At this point, we can perform an interesting operation. Suppose we create a new array by setting all but the first 10% of the elements of the Fourier array to zero (i.e., we keep only the first 10% and set the remaining 90% to zero). We can then compute the inverse Fourier transform of this "filtered" array using the Python function `irfft`, which is also contained in `numpy.fft`. This function returns an array with the same dimensions as the original data array. By plotting this array, obtained from the inverse Fourier transform of the "filtered" array, we obtain the following

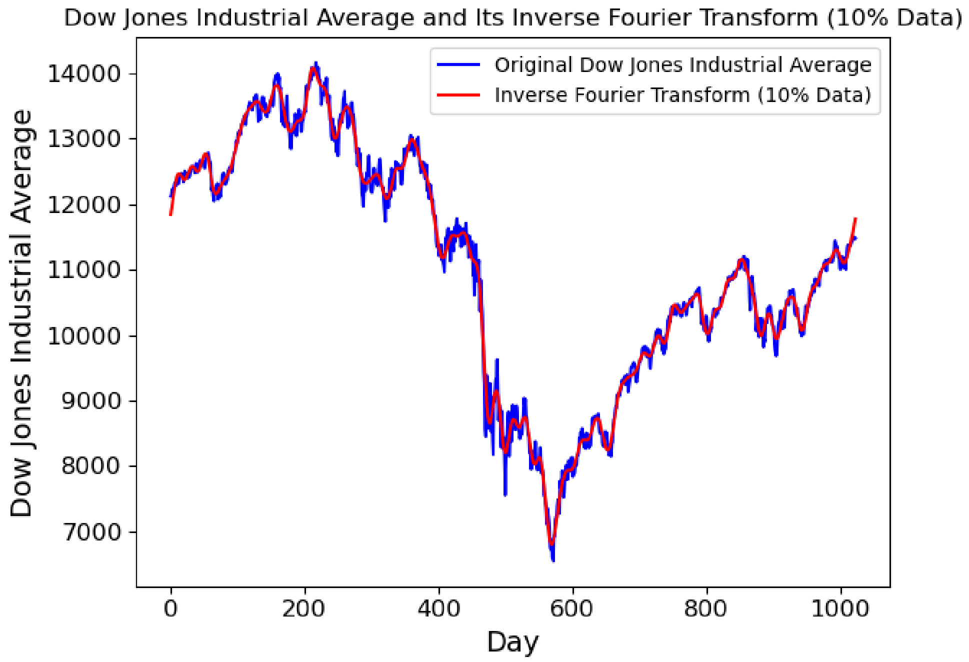

Figure 6: Plot of the original data contained in the file under consideration, together with the result of the inverse Fourier transform of the "filtered" spectrum obtained from the original Fourier transform array by setting the last 90% of its coefficients to zero.

We can clearly see that the inverse Fourier transform does not accurately reproduce the original behavior, since we are using only 10% of the total Fourier coefficients. This effect becomes even more pronounced if we create a new array from the original one, keeping only the first 2% of the coefficients. In this case, by repeating the same procedure as before, we obtain

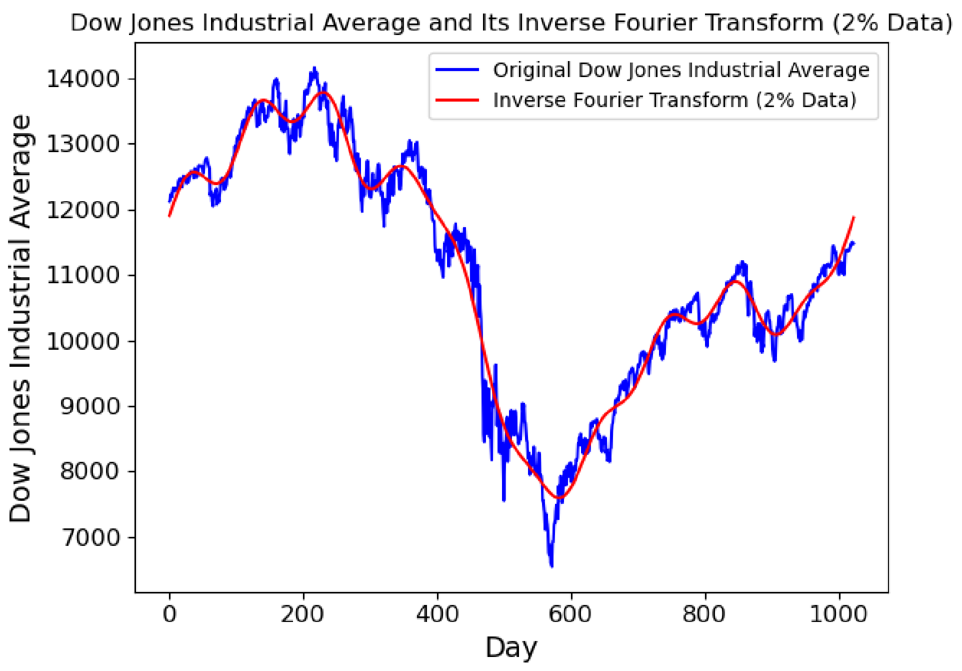

Figure 7: Plot of the original data contained in the file under consideration, together with the result of the inverse Fourier transform of the "filtered" spectrum obtained from the original Fourier transform array by setting the last 98% of its coefficients to zero.

As expected, the result is even less accurate, since we are now using only a small fraction of the total Fourier transform array.

4 Exercise 7.9 from Newman

4.1 A brief introduction

In this exercise, we will explore how to recover a sharp image starting from a blurred one. When a photograph becomes blurred, each point in the image is spread out according to a specific distribution known as the point spread function (or PSF). This function describes how a single point of light is dispersed across the image, causing the overall blurring effect. To formalize this, let us consider a grayscale image, which can be represented by a function $a(x, y)$ describing the brightness at each position (x, y). We denote the point spread function by $f(x, y)$. Hence, a single bright point located at the origin will appear as $f(x, y)$ in the blurred image. If $f(x, y)$ is broad, the image will appear strongly blurred, whereas a narrow $f(x, y)$ corresponds to a sharper image. In general, the brightness $b(x, y)$ of the blurred image at a given point (x, y) can be expressed as

$$b(x, y) = \int_0^K \int_0^L a(x', y')\, f(x - x', y - y')\, dx'\, dy', \tag{15}$$

where $K \times L$ denotes the dimensions of the image. This relationship is known as the convolution of the original image with the point spread function.

Without going into detail, by applying the definition of the Fourier transform and performing some algebraic manipulation, it can be shown that the Fourier transform of a blurred image is equal to the product of the Fourier transforms of the original (unblurred) image and of the point spread function (this result is formally known as the convolution theorem). For an image of dimensions $K \times L$, the corresponding two-dimensional Fourier transforms are related by

$$b_{k,l} = KL\, a_{k,l}\, f_{k,l}. \tag{16}$$

In the digital realm of computers, images are not represented as continuous functions $a(x, y)$, but rather as discrete grids of sampled values. Consequently, the corresponding Fourier transforms are discrete rather than continuous. Nevertheless, the underlying mathematics remains essentially the same. The main practical difficulty in image deblurring is that the point spread function is usually unknown. In most cases, one must experiment with different forms until a suitable one is found. For many cameras, it is often a reasonable approximation to assume that the point spread function has a Gaussian shape

$$f(x, y) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right), \tag{17}$$

where σ represents the width of the Gaussian. Even with this assumption, however, the precise value of σ is typically unknown, and it is often necessary to experiment with different values to determine the one that produces the best results.

In this exercise, we assume that the value of σ is known, and in particular we take $\sigma = 25$.

4.2 Solution of the exercise

4.2.1 Point a

First, we take the file `blur.txt`, which contains a grid of values representing the brightness of a black-and-white photograph that has been deliberately blurred using a Gaussian point spread function with width $\sigma = 25$. By storing the data in a two-dimensional array of real numbers, we can visualize it on the computer screen as a density plot, obtaining

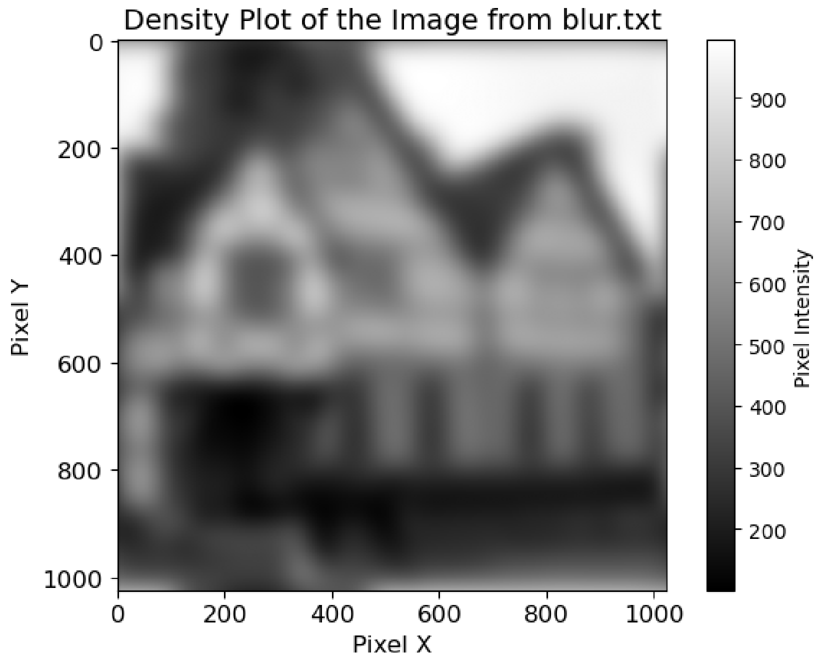

Figure 8: Density plot of the data contained in the file under consideration. This is the blurred image that will be restored to a sharp version.

We need to restore the image that we have just displayed.

4.2.2 Point b

Now we create an array, of the same size as the image, containing a grid of samples drawn from the Gaussian $f(x, y)$ defined in Eq. (17) with $\sigma = 25$. We then visualize these values as a density plot on the screen, providing a representation of the point spread function. An important point to remember is that the point spread function is periodic along both axes. This means that values corresponding to negative x and y are repeated at the edges of the array. Since the Gaussian is centered at the origin, bright patches will appear in each of the four corners of the image. To correctly position the Gaussian, we use the Python function `roll` from `numpy`. The resulting density plot is shown in Figure Fig. 9

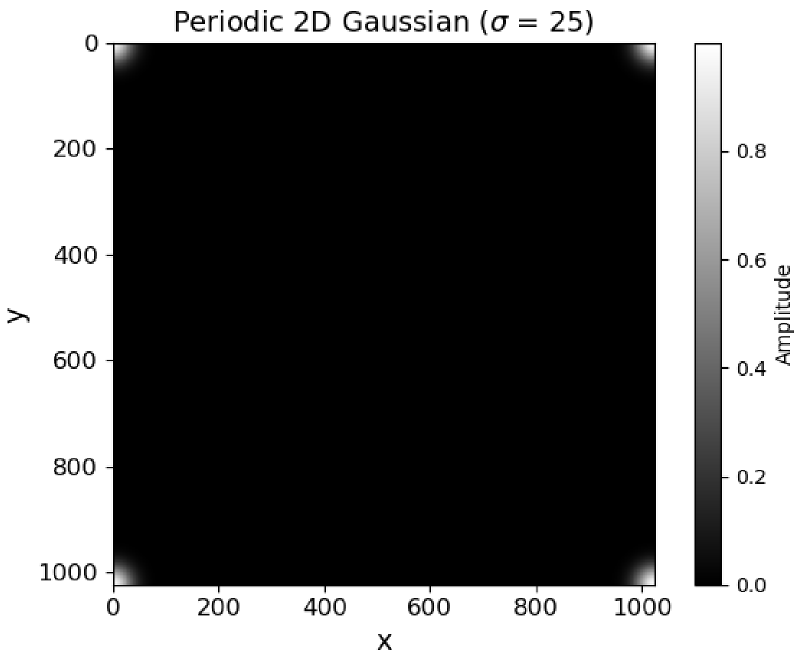

Figure 9: Density plot of the Gaussian point spread function.

We will use this to restore the blurred image.

4.2.3 Point c

In this point we combine the two programs done in the previous steps and include Fourier transforms using the functions `rfft2` and `irfft2` from `numpy.fft` to create a program that performs the following steps

1. read in the blurred photo;
2. calculate the point spread function;
3. compute the Fourier transforms of both the image and the point spread function;
4. divide the Fourier transform of the image by the Fourier transform of the point spread function;
5. perform an inverse Fourier transform to obtain the unblurred photo;
6. display the unblurred photo on the screen.

There is, however, a subtle point that deserves consideration. When dividing the Fourier transform of the image by the Fourier transform of the point spread function, it is possible that some values of the point spread function's Fourier transform are zero or very close to zero. In such cases, division would either result in an error (since division by zero is undefined) or produce extremely large values (due to division by a very small number). A practical workaround is to set a threshold ϵ: if a value in the Fourier transform of the point spread function is smaller than ϵ, we leave that coefficient unchanged instead of dividing by it. The choice of ϵ is not critical, but it should be reasonably small; in our case, we use $\epsilon = 10^{-5}$. By implementing these steps, we obtain the deblurred image, shown in Fig. 10

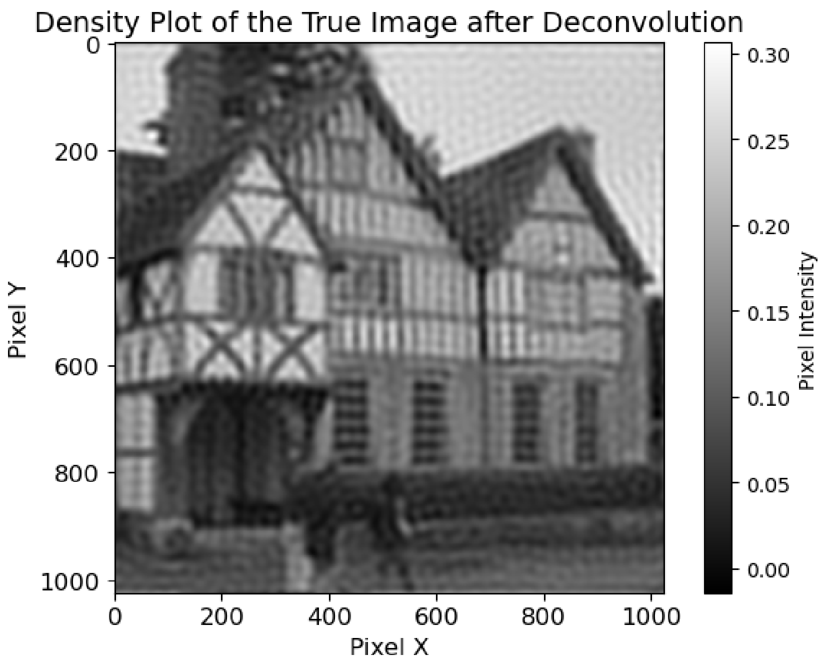

Figure 10: Density plot of the deblurred image.

Although the image is not perfectly sharp, it is significantly clearer compared to the original blurred image shown in Figure 8.

Note that it is impossible to perfectly deblur any photo and make it completely sharp. This is due to two main reasons: permanent information loss at the frequencies where the Fourier transform of the Point Spread Function (PSF) is exactly zero and noise amplification, which occurs when attempting to divide the blurred image's Fourier transform by the very small values of the PSF's Fourier transform. Even tiny amounts of noise in the original image are multiplied by extremely large factors, resulting in severe artifacts and making the final image not perfectly deblur.

5 Conclusion

In this homework, we learned how to implement the discrete Fourier transform (see Sec. 2), the Fast Fourier Transform in Sec. 3, and, finally, how to restore a blurred photograph using the convolution theorem with a Gaussian as the point spread function (see Sec. 4). We also explored how the inverse Fast Fourier Transform works and how the result changes when the input data is reduced. From the last exercise, we learned a useful technique that has many applications in physics and imaging. For example, it is used in astronomy to enhance images taken by telescopes. A famous application was with the Hubble Space Telescope: after it was discovered that the telescope's main mirror had a serious manufacturing flaw and produced blurry images, scientists were able to partially correct the blurring using Fourier transform techniques.

6 Code

The code containing all the results can be found at the following Repository GitHub.