Fourier Transforms

Ben Glick and Arden Rasmussen

September 2018

1 Introduction

The Fourier transform decomposes a function of time into the frequencies that make it up, in a way similar to how a musical chord can be expressed using the frequencies of its constituent notes. A Fourier series is a representation of a potentially complex function as a linear combination of several sines and cosines. The Fourier transform of a function has several useful properties, among which are the fact that the Fourier transform of a function will always be continuous and invertible.

This report contains write-ups of each book problem solved. They are all taken from chapter seven of Mark Newman's *Computational Physics*.

2 The Algorithms

There are a number of algorithms for computing fourier transforms of functions. The ones we are interested in are all discrete fourier transforms (DFTs). The DFT transforms discrete samples of a function into discrete samples of its fourier transform. Because DFTs do not deal with continuous functions, they can be implemented on computers to specified amounts of precision. The most important algorithm we use is a Fast Fourier Transform (FFT). The FFT is required because the naive implementation of the DFT is too slow to use on sufficiently large data sets. The specific FFT algorithm we used in this project is called the Cooley-Tukey FFT algorithm. This algorithm recursively breaks down a DFT into many smaller DFTs and operations required to stitch the smaller results back together into the intended DFT. This algorithm is particularly flexible because of its recursive nature.

Because it breaks down DFTs into smaller DFTs, any other DFT algorithm can be used in the base case to ultimately solve the problem.

3 Problem 7.2

Problem 7.2 concerns historical data which tracks the number of sunspots observed each month since January of 1749.

3.1 Part (a)

Part (a) asks for a plot of the number of sunspots as a function of time. This plot is included in Fig. 1.; We can see that the number of sunspots has fluctuated regularly since the beginning of the recorded data. We estimate the length of this cycle is approximately 133.2 months.

3.2 Part (b)

Part (b) asks us to compute the fourier transform of the sunspot data, calculate the magnitude squared ($|c_k|^2$) of each Fourier coefficient, and make a plot of these calculations as a function of k. This plot is also known as the power spectrum of the sunspot function. In this plot, peaks of the function represent a frequency in the fourier series which has a higher amplitude than others. This plot is included in Fig. 2. We used the FFT algorithm implemented by numpy to calculate this fourier series. This peak corresponds to the frequency of the periodic wave we can see in the original data.

3.3 Part (c)

Part (c) asks for the approximate value of k to which the peak corresponds, and what is the period of the sine wave with this k value. We find that this peak occurs roughly at $k = 24$. This corresponds to a sine wave with period 130 months, which is close to our predicted 133.1 months.

This kind of Fourier analysis is a sensitive method for detecting periodicity in signals. Even in cases where it is not clear to the eye that there is a periodic component to a signal, it may still be possible to find one using a Fourier transform.

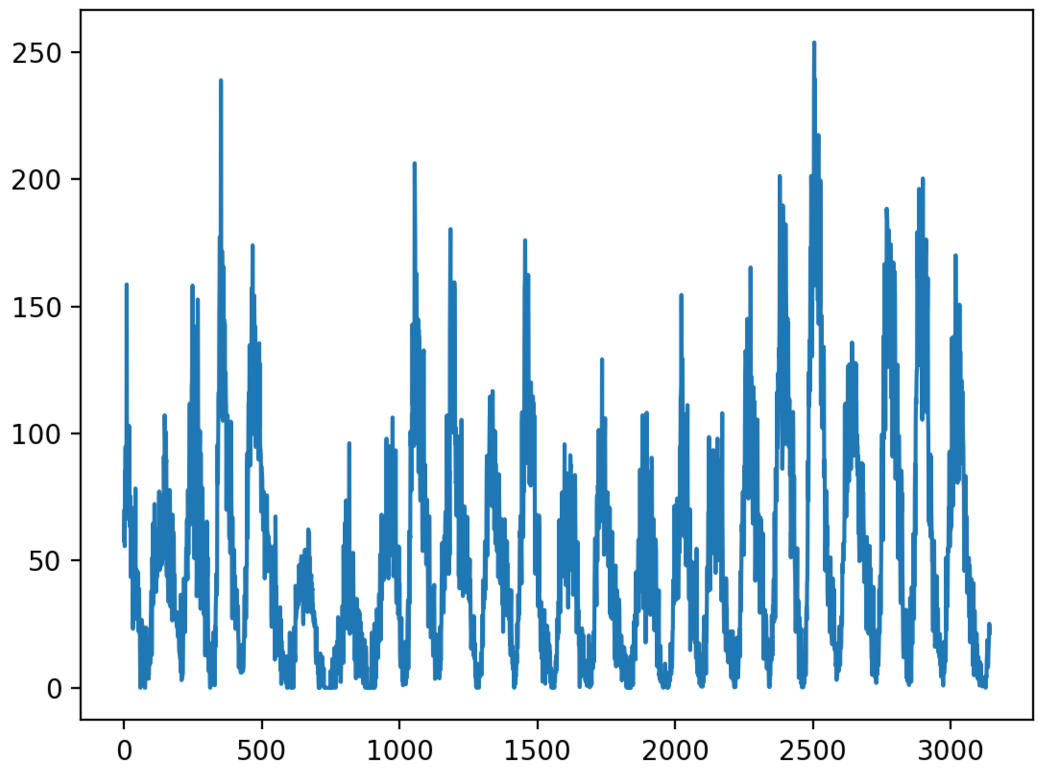

Figure 1: Plot of number of sunspots observed in a given month over time.

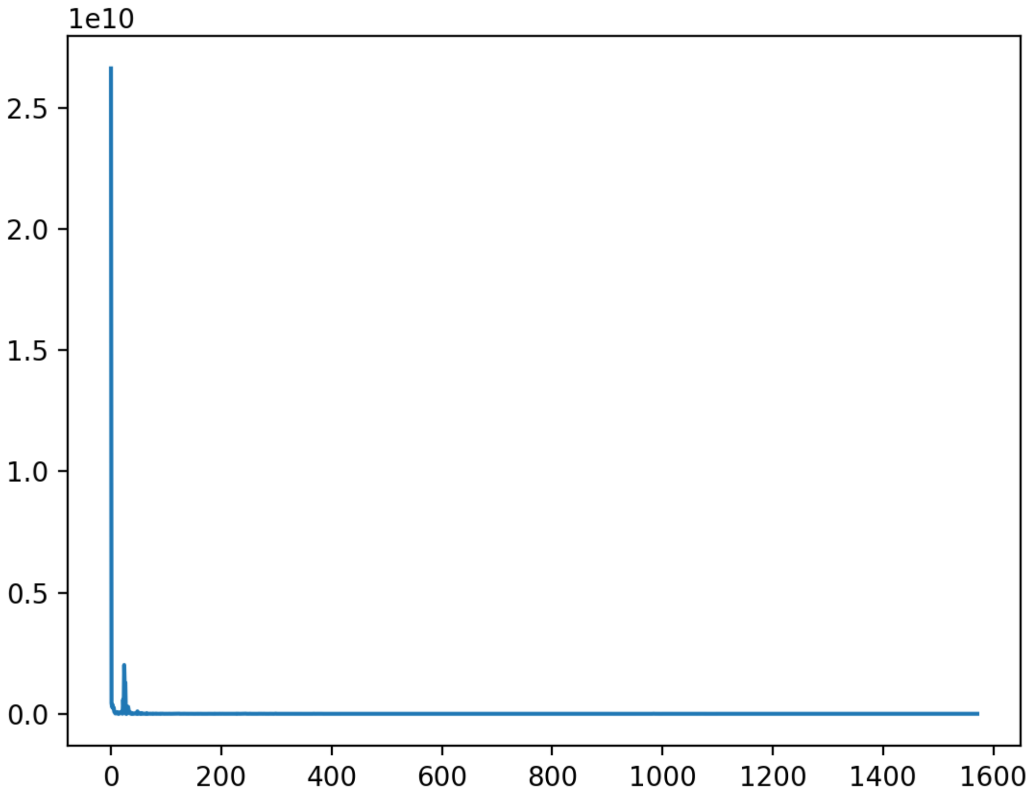

Figure 2: Fourier Transform of the Sunspot Function from Fig. 1

4 Problem 7.3

Problem 7.3 concerns data representing the waveform of a single note, played on, respectively, a piano and a trumpet.

4.1 Part (a)

Part (a) asks for a program that loads a waveform from one of these files, plots it, then calculates its discrete Fourier transform and plots the magnitudes of the first 10,000 coefficients. We had to use a fast Fourier transform for the calculation. We applied our program to the piano and trumpet waveforms. These are included in Fig. 3 and Fig. 4.

From observing the fourier transforms of these two instruments, we can see that the piano has one dominant frequency at a given note, while the trumpet has multiple which all sound together to make a single note. The trumpet's waveform appears to have several constituent frequencies which are represented by relatively evenly spaced fourier coefficients in the fourier transform of its waveform..

4.2 Part (b)

Part (b) asks us to use our Fourier transform results calculate what note the instruments were playing. We know that both waveforms were recorded at the industry-standard rate of 44,100 samples per second and both instruments were playing the same musical note when the recordings were made. In the piano recording, the peak occurs at $k \approx 1200$. Using the fact that 44,100 samples are made per second, we can say that the period of this note takes 1200 samples, or 0.0272 seconds. This leads to a frequency of 36.7 Hz, which corresponds to the note D_1.

5 Problem 7.4

Problem 7.4 concerns the daily closing value for each business day from late 2006 until the end of 2010 of the Dow Jones Industrial Average, which is a measure of average prices on the US stock market.

It asks us to create a program which does a number of things:

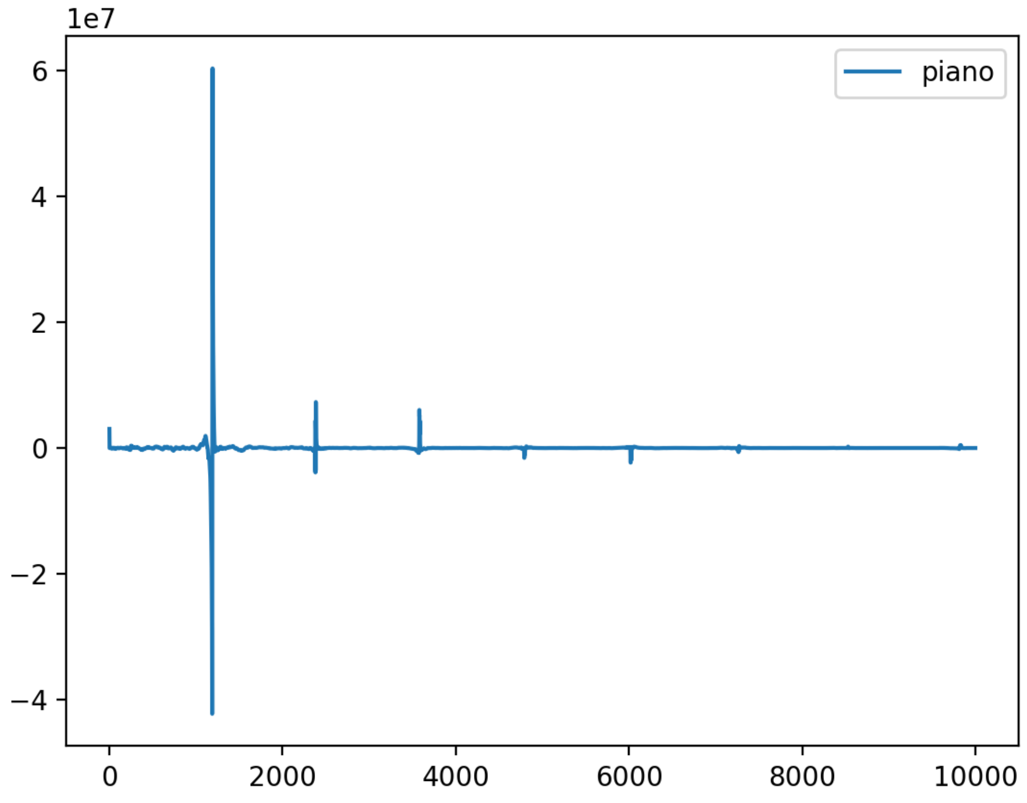

Figure 3: Fourier transform of the piano playing a single note

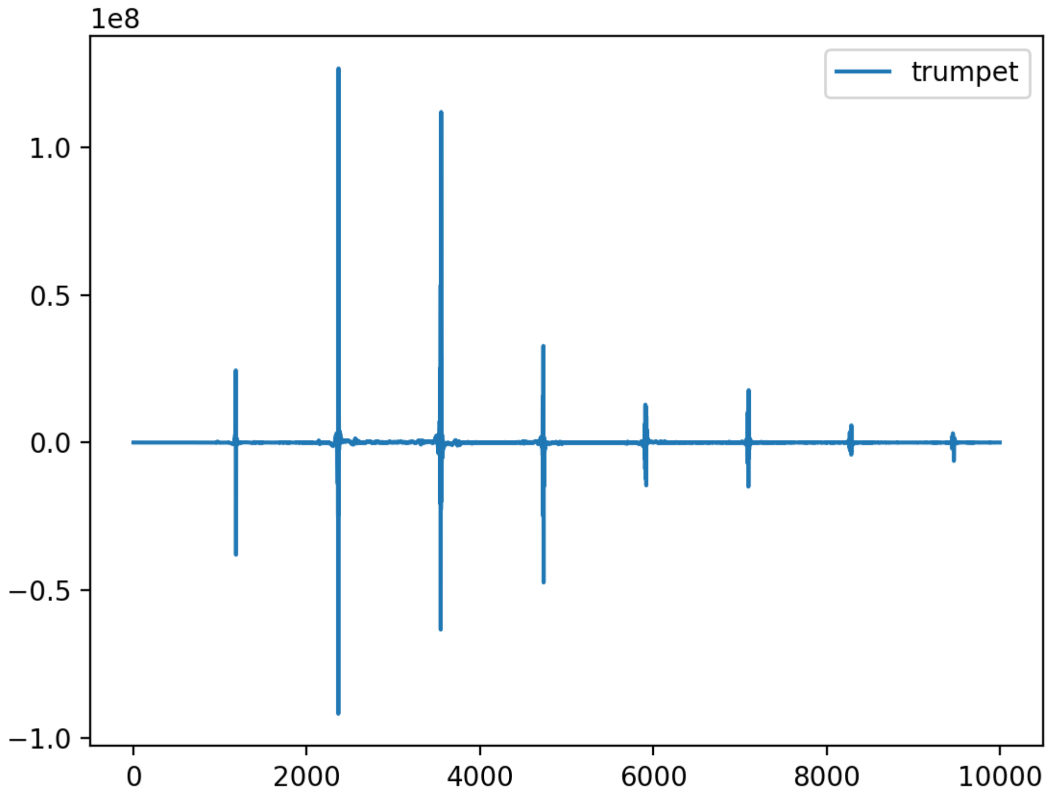

Figure 4: Fourier transform of the trumpet playing a single note

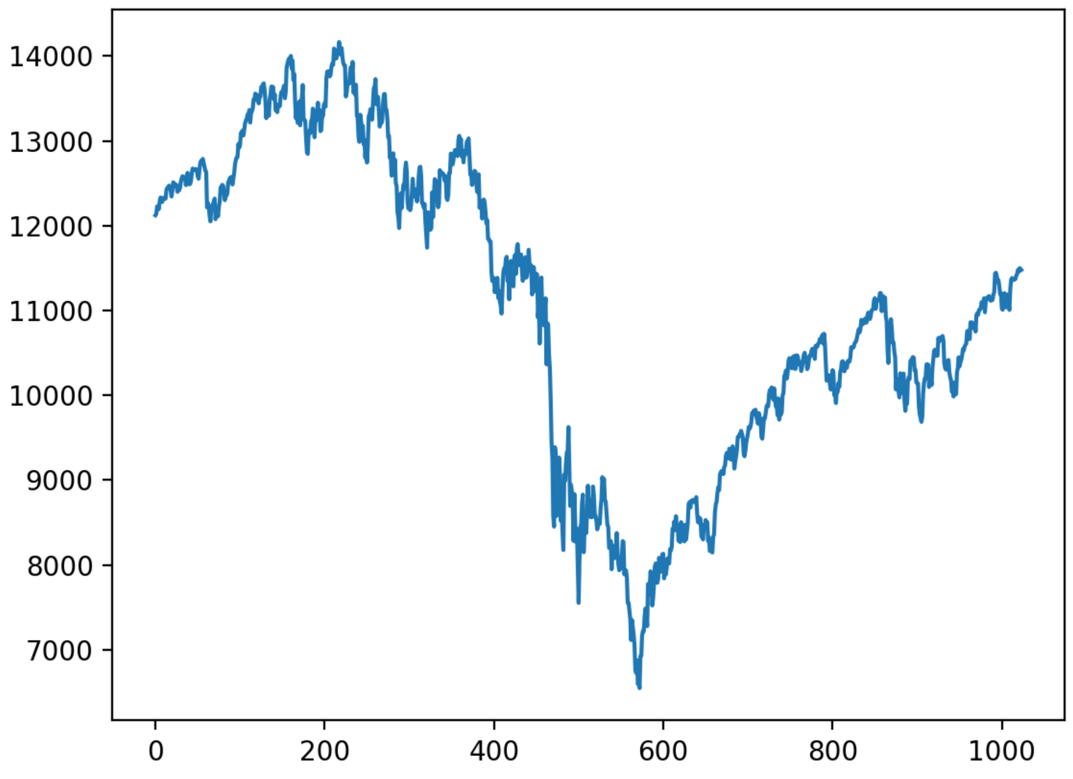

Figure 5: Daily closing value of the Dow Jones index from 2006 to 2010

5.1 Part (a)

Part (a) asks us to read in the data and plot them on a graph. This graph is in Fig. 5.

5.2 Part (b)

Part (b) asks us to calculate the coefficients of the discrete Fourier transform of the data using the function rfft from numpy.fft, which produces an array of $\frac{1}{2}N+1$ complex numbers. These are plotted in Fig. 6

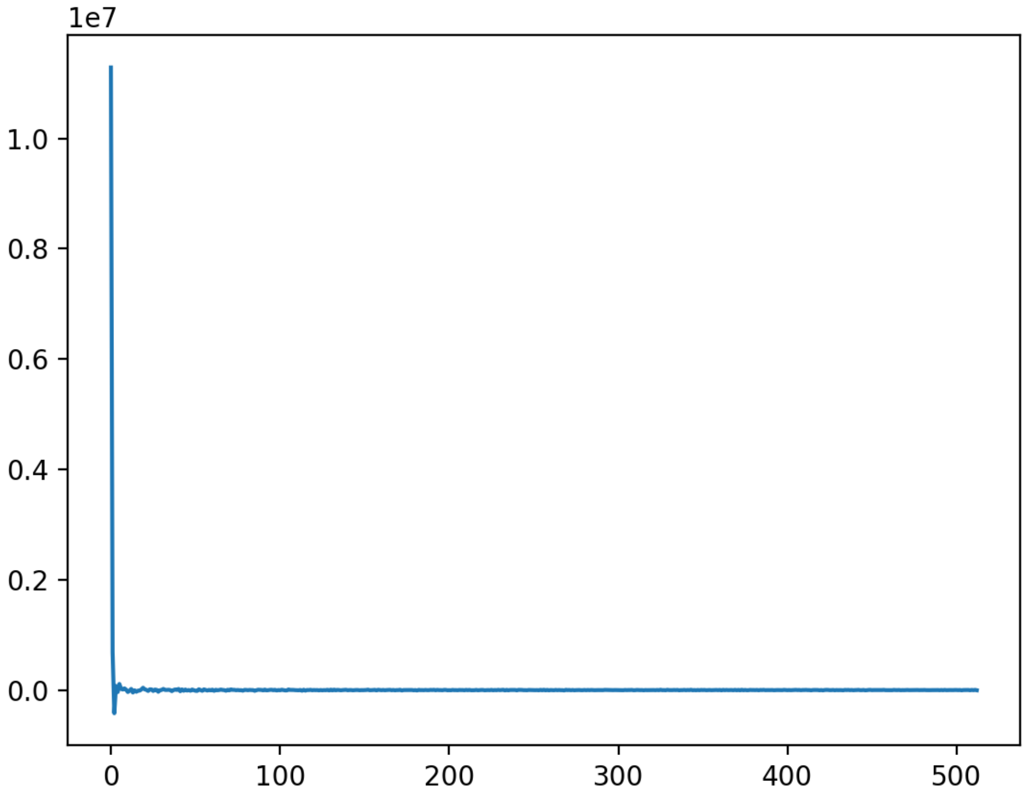

Figure 6: Fourier transform of daily closing value of the Dow Jones index from 2006 to 2010

5.3 Part (c)

Part (c) asks us to set all but the first 10% of the elements of this array to zero. The plot this produces is essentially the same as the one generated in part (b), so it is omitted here.

5.4 Part (d)

Part (d) instructs us to calculate the inverse Fourier transform of the resulting array, zeros and all, using the function irfft, and plot it on the same graph as the original data. This plot is in 7

We can see that the inverse Fourier transform mostly follows the value of the function we want it to, but it looks a bit smoother. This is because when we set the values of the Fourier coefficients with relatively low amplitudes (those last 90%), we de-noise the function. This occurs because we are disregarding the constituent frequencies in the function which are insignificant when compared to the general trend of the function, but take all of the data into account.

5.5 Part (e)

Part (e) instructs us to repeat part (d), but disregarding the last 98% of the Fourier coefficients, not just the last 90%. This plot is in Fig. 8

We can see that something similar has occurred to that from section (d). In this case, it has gone even further. The function has been further smoothed, and we see an even more de-noised version of the function that follows the general trend of the initial data, but loses some of the more fine-grained detail from the real-life data. Smoothed graphs like this and the one from section (d) can be useful to help understand the general trend of a function without the noise involved with real world data.

6 Problem 7.5

This problem also related to de-noising of a function, but it concerns a known function, the square wave with amplitude 1 and frequency 1 Hz. A plot of the original function and the smoothed signal is in Fig. 9

As expected, we find that the signal is not simply smoothed—there are artifacts, wiggles, in the results. These artifacts come from the fact that

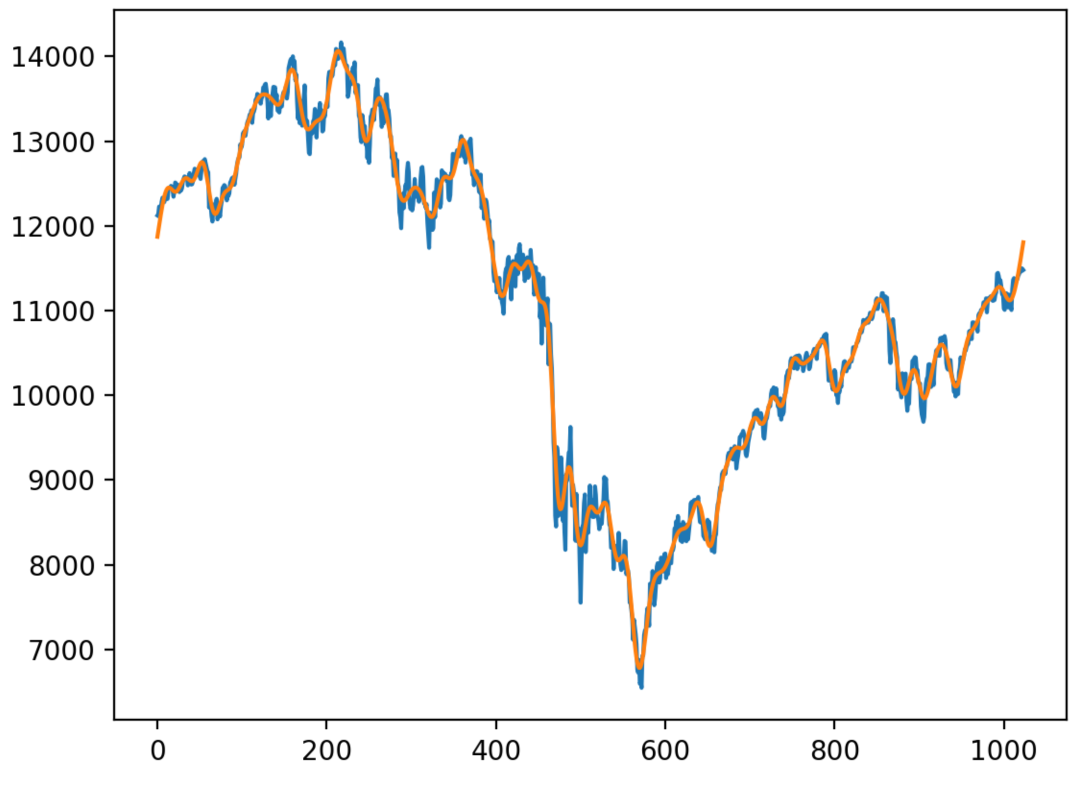

Figure 7: Inverse Fourier transform of daily closing value of the Dow Jones index from 2006 to 2010, plotted over the real samples.

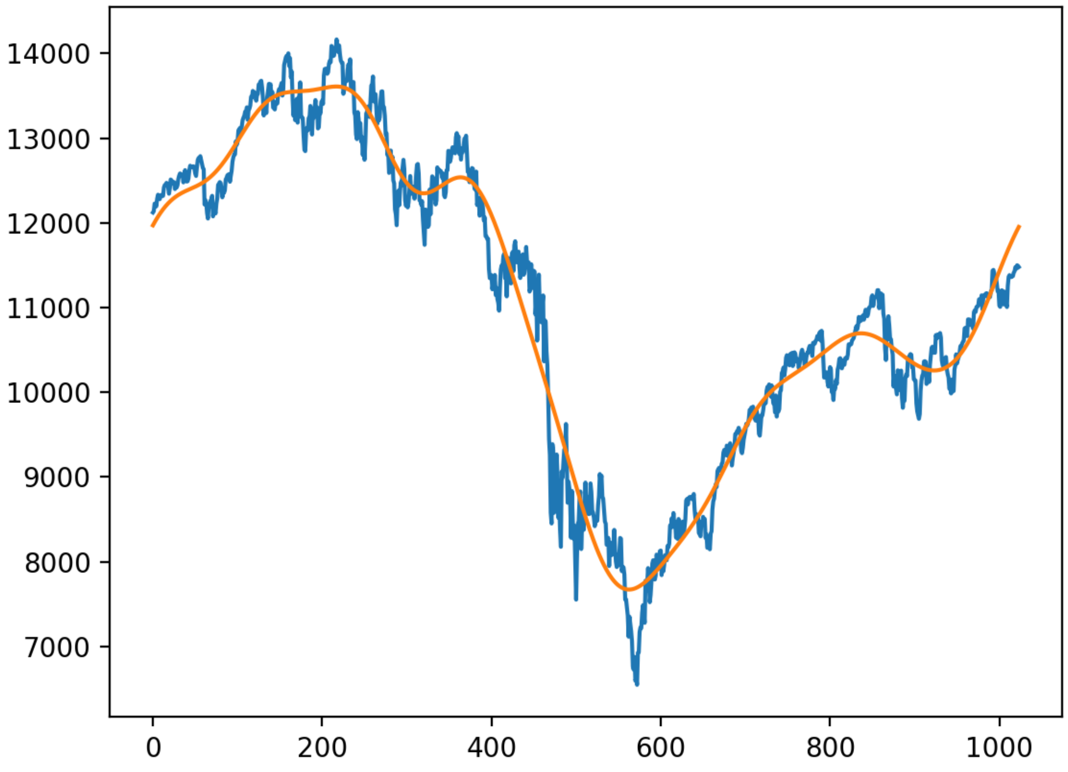

Figure 8: Inverse Fourier transform of daily closing value of the Dow Jones index from 2006 to 2010, plotted over the real samples.

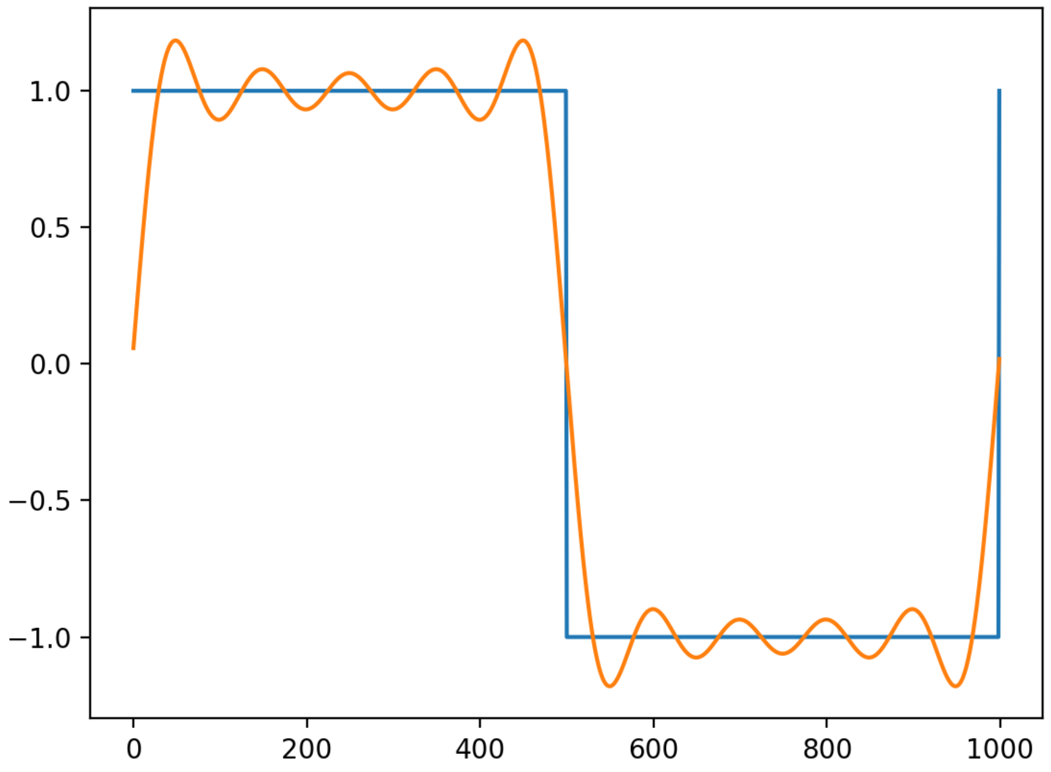

Figure 9: Inverse Fourier transform of the square wave function, with the real function.

there are jump discontinuities in this square wave function. The function is not well-defined where it jumps from the value of 1 to -1, and this causes problems for the Fourier transform. These artifacts and wiggles are there because of the difficulty inherent in approximating a discontinuous function by a finite series of continuous sine and cosine waves, as we are doing when calculating the Fourier transform.

Artifacts similar to these arise when Fourier coefficients are discarded in audio and visual compression schemes and are the primary source of imperfections in digitally compressed sound and images.

7 Problem 7.6

Like exercise 7.4, this problem concerns the function which represents the value of the Dow Jones index over time, but this problem concerns a different period of time. The time period studied in that exercise was special in one sense: the value of the Dow at the end of the period was almost the same as at the start, so the function was, roughly speaking, periodic.

This exercise concerns the value of the Dow Jones index from 2004 until 2008. Over this period the value changed considerably from a starting level around 9000 to a final level around 14000.

8 Part (a)

Part (a) asks us to write a program similar to the one for Exercise 7.4, part (e), in which you read the data and plot it on a graph. Then smooth the data by calculating its Fourier transform, setting all but the first 2% of the coefficients to zero, and inverting the trans- form again, plotting the result on the same graph as the original data. This plot is in Fig. 10

At the beginning and end of the plot we see large deviations away from the true smoothed function. These occur because the function is required to be periodic, so it needs to deviate substantially from the correct value to make the two ends of the function meet. In some situations (including this one) this behavior is unsatisfactory. If we want to use the Fourier transform for smoothing, we would certainly prefer that it not introduce artifacts of this kind.

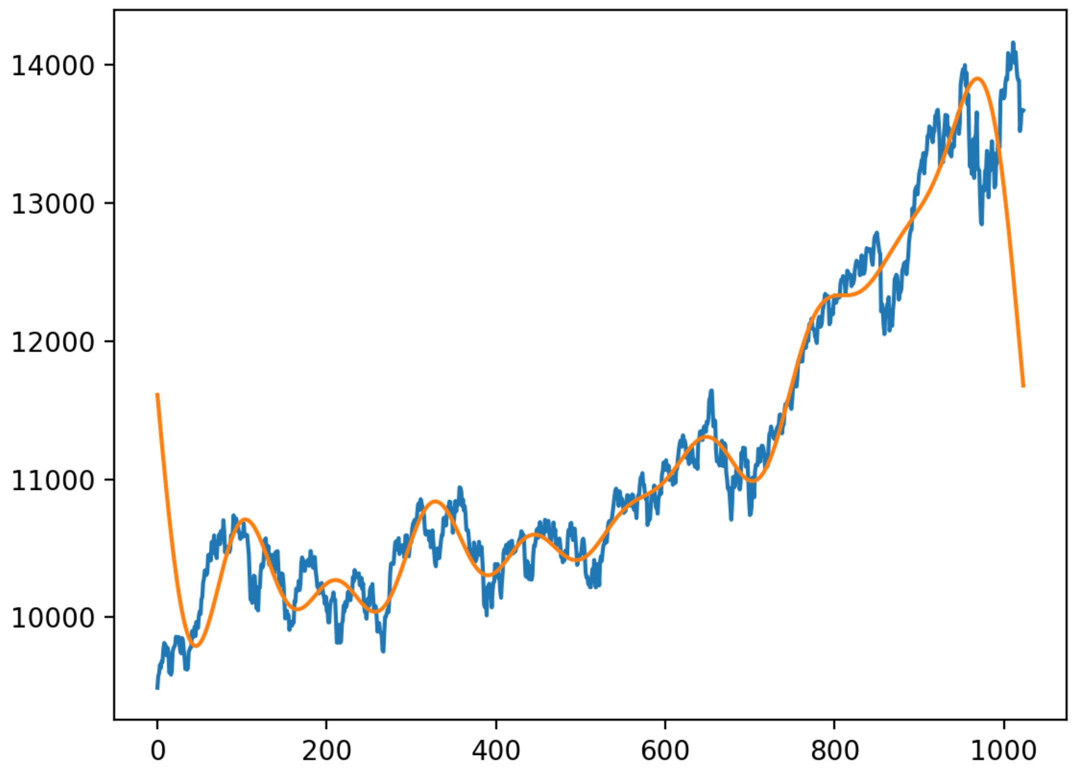

Figure 10: Inverse Fourier transform of daily closing value of the Dow Jones index from 2004 to 2008, plotted over the real samples.

9 Part (b)

Part (b) asks us to modify your program to repeat the same analysis using discrete cosine transforms. As expected, we see a significant improvement, with less distortion of the function at the ends of the interval. This occurs because the cosine transform does not force the value of the function to be the same at both ends. This plot is included in Fig. 11

It is because of the artifacts introduced by the strict periodicity of the DFT that the cosine transform is favored for many technological applications, such as audio compression. The artifacts can degrade the sound quality of compressed audio and the cosine transform generally gives better results. The cosine transform is not wholly free of artifacts itself however. It's true it does not force the function to be periodic, but it does force the gradient to be zero at the ends of the interval (which the ordinary Fourier transform does not).

10 Problem 7.9

Problem 7.9 concerns applying the fourier transform to deblur images. This is possible because the Fourier transform of the blurred photo is the product of the Fourier transforms of the unblurred photo and the point spread function.

10.1 Part (a)

Part (a) asks us to show a grid of values representing brightness on a black-and-white photo—a badly out-of-focus one that has been deliberately blurred using a Gaussian point spread function of width $\sigma = 25$. This image is in Fig. 12

10.2 Part (b)

Part (b) asks us to write another program that creates an array, of the same size as the photo, containing a grid of samples drawn from the Gaussian f (x, y) above with $\sigma = 25$ and make a density plot of these values on the screen. This is in Fig. 13.

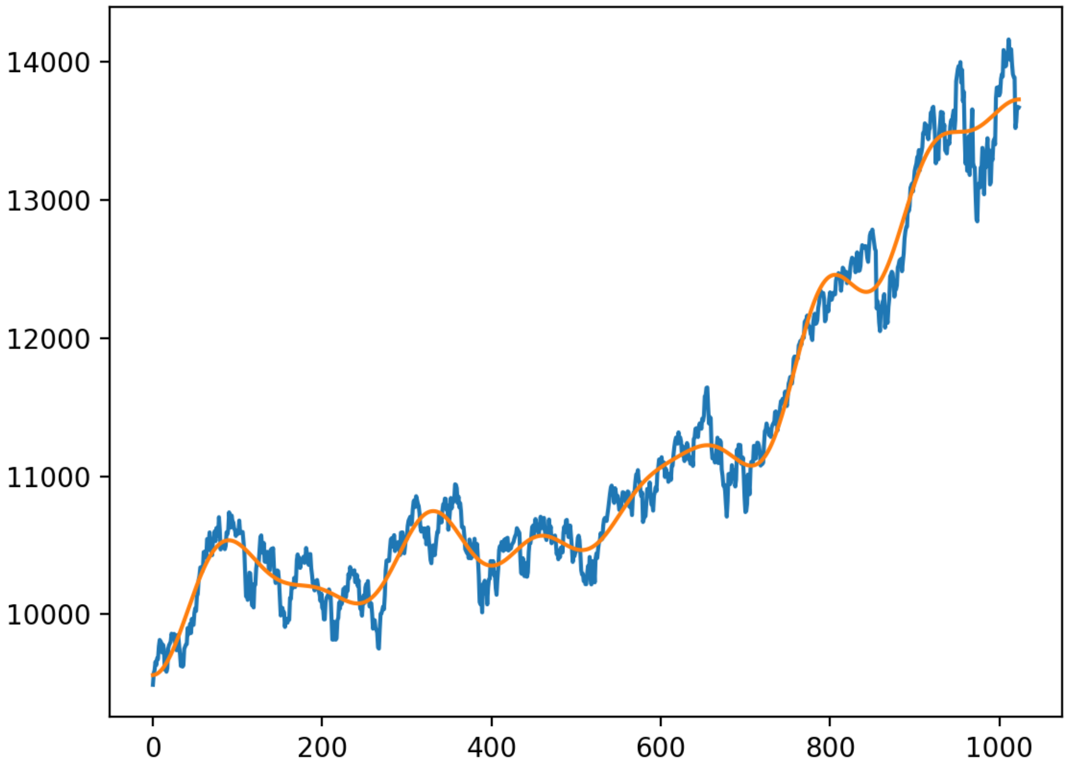

Figure 11: Inverse cosine transform of daily closing value of the Dow Jones index from 2004 to 2008, plotted over the real samples.

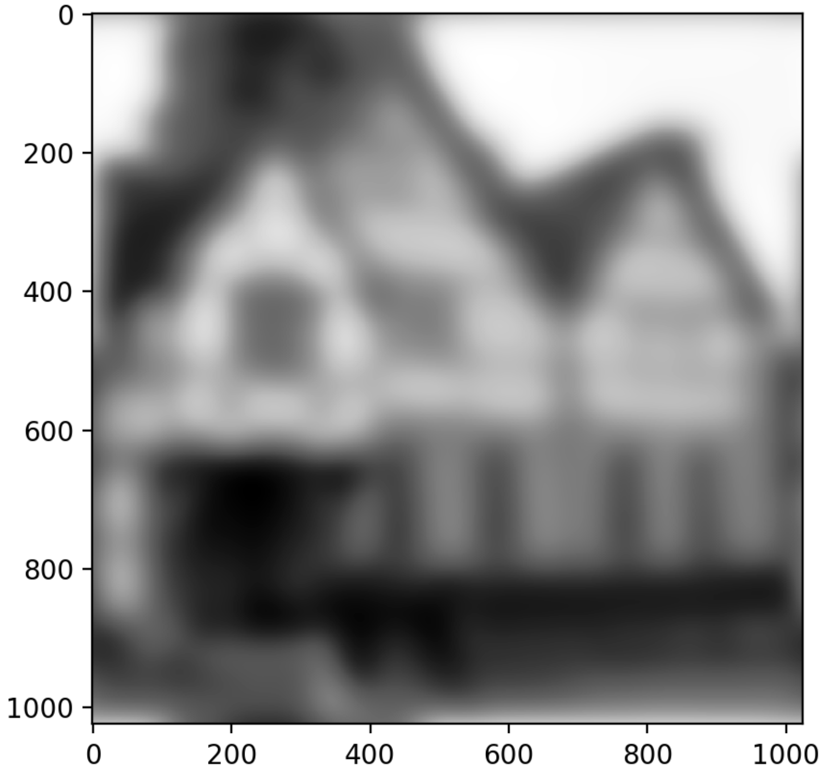

Figure 12: Blurred picture from 7.9 (a)

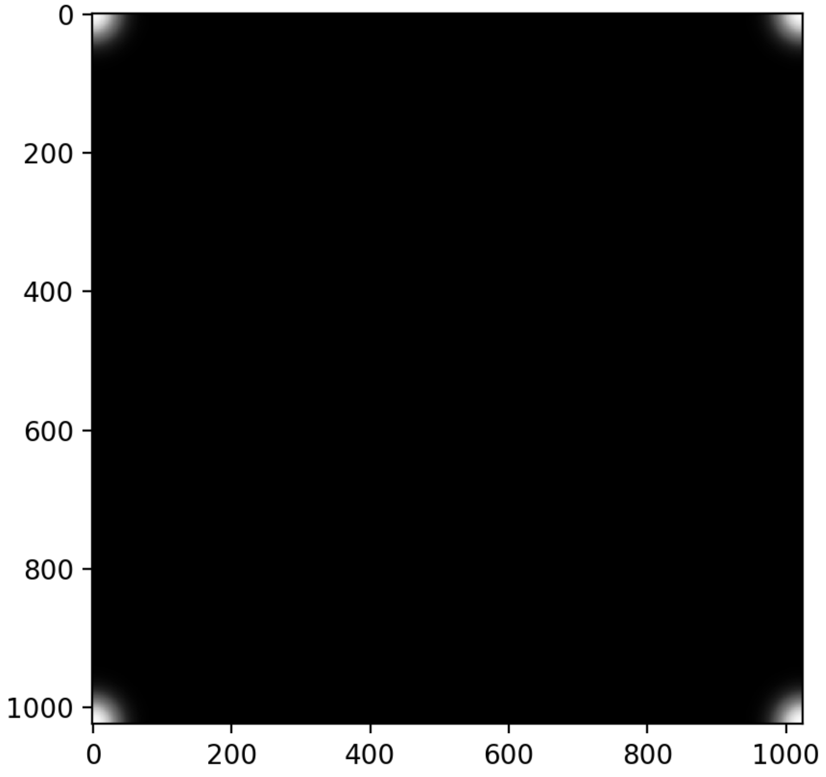

Figure 13: Point spread function from 7.9 (b)

10.3 Part (c)

Part (c) asks us to deblur the photo by carrying out the following process:

- Reads in the blurred photo
- Calculates the point spread function
- Fourier transforms both
- Divides one by the other
- Performs an inverse transform to get the unblurred photo
- Displays the unblurred photo on the screen

The deblurred photo is in Fig. 14

10.4 Part (d)

Part (d) asks us to discuss what is it that limits our ability to deblur a photo, and why can we not perfectly unblur any photo and make it completely sharp.

When we deblur photos, the inverse transform cannot be perfect. There are problems similar to those we found when we were analyzing the Dow Jones index and when we were analyzing the square wave function. These artifacts that come up in one-dimensional applications still occur in two-dimensionalk ones. There will always be points that we cannot deal with perfectly, and these lead to graininess and other imperfections in our deconvoluted picture.

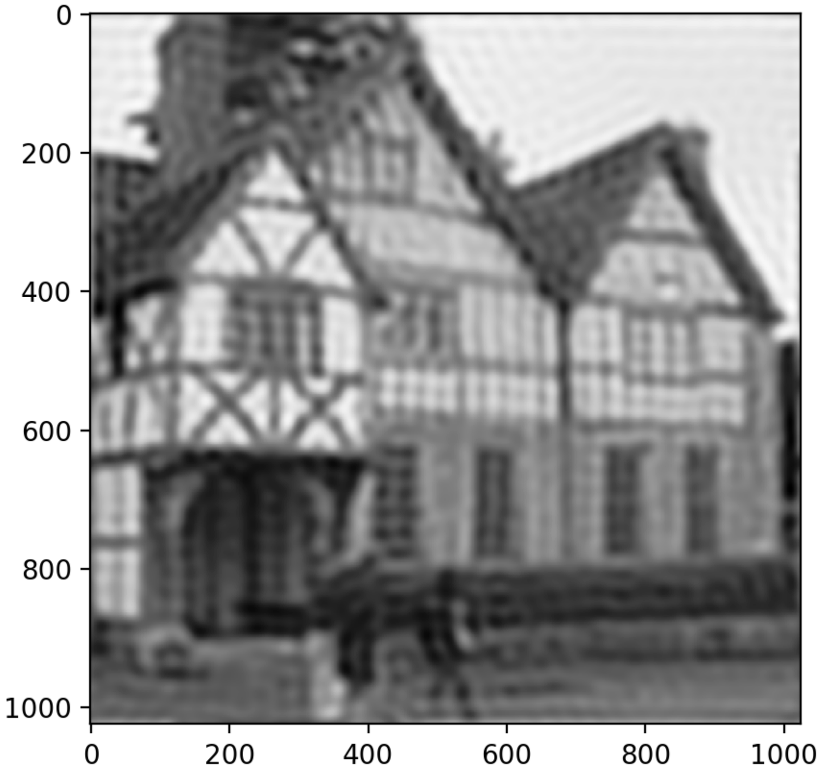

Figure 14: Deconvoluted image from 7.9 (c)