Exploring Fourier series from computational perspective

Igor Krzywda

March 30, 2021

Contents

1 Introduction

One time when I was binging YouTube, I stumbled upon a video by Grant Sanderson of 3Blue1Brown on Fourier series which featured many visualizations, prime of whose were drawings on a complex plain which were generated using Fourier series. Having done some reading on Fourier series as well as Fourier transform, I decided that it would be a great idea to actually understand those concepts. Since for me the ultimate way of learning is solving problems and since I feel best at expressing ideas in code, I decided to set myself a challenge of programming a program that will redraw my sketches with Fourier series.

2 What is Fourier series?

The purpose of Fourier series is to approximate any periodic function with a sum of sines and cosines. The formula is given by (1)

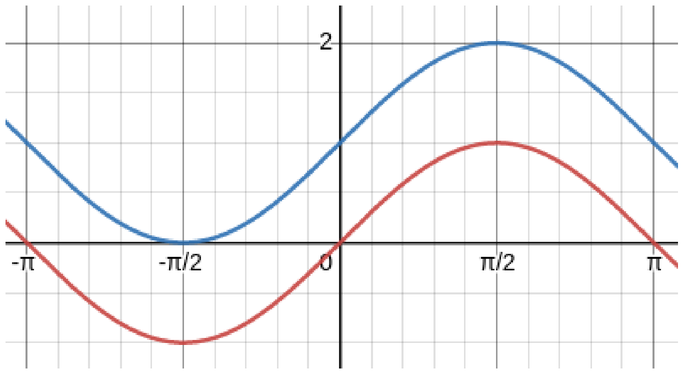
$$S_\infty(t) = a_0 + \sum_{n=1}^{\infty} a_n cos\frac{2\pi nt}{P} + b_n sin\frac{2\pi nt}{P} \tag{1}$$

From the formula we can observe two things, each sine and cosine in the sum has a weight assigned to it (a_n and b_n) and that n determines the frequency of the sinusoids. We can demistify the idea behind Fourier series by deriving formulas for all coefficients.

2.1 Deriving a_0

First coefficient we will find is a_0, which is the only coefficient that stands on its own. It is added before the actual series in order to compensate for the original function not oscillating around x axis on the plot. It basically is a vertical translation. We can see it if we add any constant to $sin(x)$, Figure 1.

Figure 1: red : $sin(x)$, blue : $2 + sin(x)$



The problem to solve is to find the axis against which the sum will oscillate The solution to this problem is the average value of the original function over one period. The formula given for finding a_0 goes as follows (2)

$$a_0 = \frac{1}{P}\int_{\frac{P}{2}}^{\frac{-P}{2}} f(t)dt \tag{2}$$

The average can be found through this formula because the integral gives us the area under the graph of function $f(t)$ from $-\frac{P}{2}$ to $\frac{P}{2}$. If we have an area, we can express it with any figure that has this area, since we want an offset from the x axis, we can draw a rectangle whose base has length of P. Since we want to find out what the height of that rectangle is, we need to divide its area by its width, which is exactly what we are doing when dividing the integral by P.

2.2 Finding a_n and b_n

Coefficients a_n and b_n determine the amplitudes of sine waves of frequency n. Since the goal is to approximate some function $f(t)$ with a sum of weighted sines and cosines, let that be the starting point. (3)

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n cos\frac{2\pi nt}{P} + b_n sin\frac{2\pi nt}{P} \tag{3}$$

In order to find the coefficients, we can exploit the properties of definite integrals of sine and cosine. To do that, we will need to expand the whole expression by either $cos\frac{2\pi nt}{P}$ when looking for a_n, or by $sin\frac{2\pi nt}{P}$ if we are looking for b_n.

2.2.1 Finding a_n

The expanded equation used to find a_n looks like this (4).

$$\begin{aligned}\int_{-\frac{P}{2}}^{\frac{P}{2}} f(t)cos\frac{2\pi nt}{P}dt = \int_{-\frac{P}{2}}^{\frac{P}{2}} a_0 cos\frac{2\pi nt}{P}dt &+ \int_{-\frac{P}{2}}^{\frac{P}{2}} a_1 cos\frac{2\pi t}{P}cos\frac{2\pi nt}{P}dt + \int_{-\frac{P}{2}}^{\frac{P}{2}} b_1 sin\frac{2\pi t}{P}cos\frac{2\pi nt}{P}dt \\ &+ \int_{-\frac{P}{2}}^{\frac{P}{2}} a_2 cos\frac{4\pi t}{P}cos\frac{2\pi nt}{P}dt + \int_{-\frac{P}{2}}^{\frac{P}{2}} b_2 sin\frac{4\pi t}{P}cos\frac{2\pi nt}{P}dt \\ &\vdots \\ &+ \int_{-\frac{P}{2}}^{\frac{P}{2}} a_n cos^2(\frac{2\pi nt}{P})dt + \int_{-\frac{P}{2}}^{\frac{P}{2}} b_n sin\frac{2\pi nt}{P}cos\frac{2\pi nt}{P}dt + ...\end{aligned} \tag{4}$$

From this rather long expansion we can distill and solve for five cases appearing on the right side of the equation:

1. $$\int_{-\frac{P}{2}}^{\frac{P}{2}} a_0 cos\frac{2\pi nt}{P}dt = a_0 \cdot \left[\frac{P}{2\pi n}sin\frac{2\pi nt}{P}\right]_{-\frac{P}{2}}^{\frac{P}{2}} = a_0 \cdot \frac{P}{2\pi n}(sin(\pi n) - sin(-\pi n)) = 0$$

2. For $m \in N$ and $m \neq n$

$$\begin{aligned}\int_{-\frac{P}{2}}^{\frac{P}{2}} a_m cos\frac{2\pi mt}{P}cos\frac{2\pi nt}{P}dt &= a_m \cdot \int_{-\frac{P}{2}}^{\frac{P}{2}} \frac{1}{2}\left(cos\frac{(m-n)2\pi t}{P} + cos\frac{(m+n)2\pi t}{P}\right)dt \\ &= \frac{a_m}{2} \cdot \left[\frac{P}{(m-n)2\pi n}sin\frac{(m-n)2\pi t}{P} + \frac{P}{(m+n)2\pi n}sin\frac{(m+n)2\pi t}{P}\right]_{-\frac{P}{2}}^{\frac{P}{2}} \\ &= \frac{Pa_m}{4\pi}\left(\frac{1}{m-n}sin((m-n)\pi) + \frac{1}{m+n}sin((m+n)\pi)\right. \\ &\left. -\frac{1}{m-n}sin(-(m-n)\pi) - \frac{1}{m+n}sin(-(m+n)\pi)\right) = 0\end{aligned}$$

3. For $m \in N$ and $m \neq n$

$$\begin{aligned}\int_{-\frac{P}{2}}^{\frac{P}{2}} b_m sin\frac{2\pi mt}{P}cos\frac{2\pi nt}{P}dt &= b_m \cdot \int_{-\frac{P}{2}}^{\frac{P}{2}} \frac{1}{2}\left(sin\frac{(m+n)2\pi t}{P} + sin\frac{(m-n)2\pi t}{P}\right)dt \\ &= \frac{b_m}{2} \cdot \left[\frac{-P}{2\pi(m+n)}cos\frac{(m+n)2\pi t}{P} + \frac{-P}{2\pi(m-n)}cos\frac{(m-n)2\pi t}{P}\right]_{-\frac{P}{2}}^{\frac{P}{2}} \\ &= \frac{Pb_m}{4\pi}\left(\frac{-1}{m+n}cos((m+n)\pi) + \frac{-1}{m-n}cos((m-n)\pi)\right. \\ &\left. -\frac{-1}{m+n}cos(-(m+n)\pi) - \frac{-1}{m-n}cos(-(m-n)\pi)\right) = 0\end{aligned}$$

4.

$$\int_{-\frac{P}{2}}^{\frac{P}{2}} b_n sin\frac{2\pi nt}{P}cos\frac{2\pi nt}{P}dt = b_n\int_{-\frac{P}{2}}^{\frac{P}{2}}\frac{1}{2}\left. sin\frac{4\pi nt}{P}+sin(0)\right)dt$$
$$= \frac{b_n}{2}\left[\frac{-P}{4\pi n}cos\frac{4\pi nt}{P}-cos(0)\right]_{-\frac{P}{2}}^{\frac{P}{2}}$$
$$= \frac{b_n}{2}\left(\frac{-P}{4\pi n}cos(2\pi n)-1-\frac{-P}{4\pi n}cos(-2\pi n)+1\right)=0$$

5.

$$\int_{-\frac{P}{2}}^{\frac{P}{2}} cos^2(\frac{2\pi nt}{P})dt = \frac{1}{2}\int_{-\frac{P}{2}}^{\frac{P}{2}} 1+cos\frac{4\pi nt}{P}dt = \frac{1}{2}\cdot\left[t+\frac{P}{4\pi n}sin\frac{4\pi nt}{P}\right]_{-\frac{P}{2}}^{\frac{P}{2}}$$
$$= \frac{1}{2}\left(\frac{P}{2}-\frac{4\pi n}{P}sin(2\pi n)+\frac{P}{2}+\frac{4\pi n}{P}sin(-2\pi n)\right)=\frac{P}{2}$$

We can see that the only term after expansion and integration that does not amount to zero is $\int_{-\frac{P}{2}}^{\frac{P}{2}} cos^2(\frac{2\pi nt}{P})dt$, which means that our equation will simplify to the following form (5)

$$\int_{-\frac{P}{2}}^{\frac{P}{2}} f(t)cos\frac{2\pi nt}{P}dt = a_n\frac{P}{2} \tag{5}$$

So in order to find a_n, we will get the following(6)

$$a_n = \frac{2}{P}\int_{-\frac{P}{2}}^{\frac{P}{2}} f(t)cos\frac{2\pi nt}{P}dt \tag{6}$$

Formula for b_n is very similar (7) and stems from the same logic as one of a_n, see Appendix A for calculations.

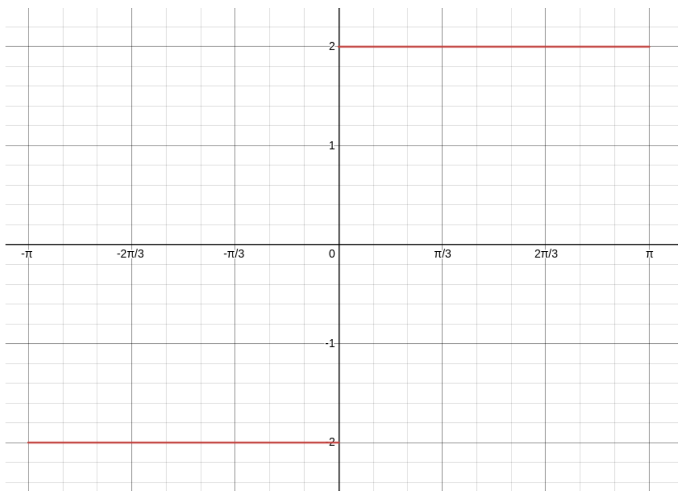
$$b_n = \frac{2}{P}\int_{-\frac{P}{2}}^{\frac{P}{2}} f(t)sin\frac{2\pi nt}{P}dt \tag{7}$$

2.3 Approximating a step function

Before diving straight to other concepts and the drawings, let us take a second and manually approximate a step function defined in (8) and plotted in Figure 2

$$\begin{aligned} f(t) &= \begin{cases} -2 \text{ if } -\pi \le t < 0 \\ 2 \text{ if } 0 \le t \le \pi \end{cases} \\ f(t) &= f(t+2\pi) \end{aligned} \tag{8}$$

Figure 2: Step function f(t)



If we look on the graph, we can already establish that the definite integral of the function over the whole period is zero, hence $a_0 = 0$.

Other thing we want to observe is that the function $f(t)$ is odd, meaning that $f(t) = -f(-t)$ and the function that is odd in our series is the sine, we know that all coefficients a_n will be zero because otherwise they would distort our approximation.

$$\begin{aligned} b_n &= \frac{2}{\pi}\int_{-\pi}^{\pi} f(t)sin(nt)dt = \frac{2}{\pi}\cdot\left[f(t)\cdot t\cdot\frac{1}{-nt}cos(nt)\right]_{-\pi}^{\pi} \\ &= \frac{2}{\pi}\cdot\left[\frac{2}{n}\left(cos(n\pi)+cos(n\pi)-(-cos(-(n\pi)+cos(-n\pi))\right)\right] \\ &= \frac{4}{n\pi}\left(2cos(n\pi)+2cos(-n\pi)\right) = \begin{cases} \frac{8}{n\pi} \text{ if n is odd} \\ 0 \text{ if n is even} \end{cases} \end{aligned} \tag{9}$$

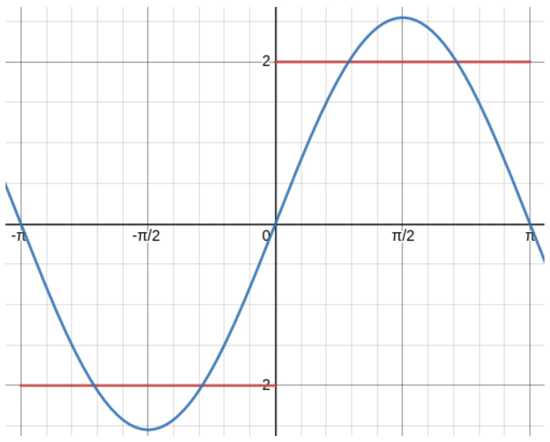
The series for our step function looks like this (10)

$$S_\infty(t) = \sum_{n=1}^{\infty} \frac{8}{(2n-1)\pi} sin((2n-1)t) \tag{10}$$

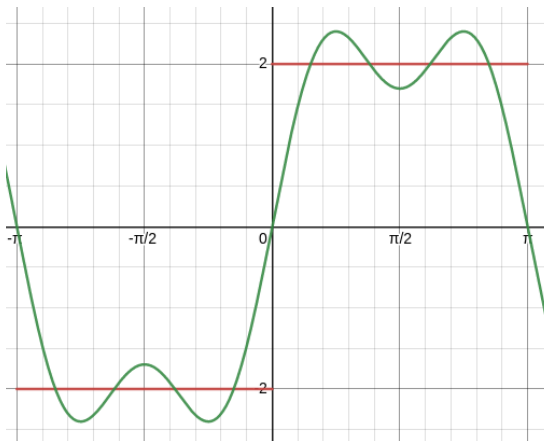
Now we can make a few plots of the series with increasing number of terms and see it converge to $f(t)$, Figure 11

Figure 3

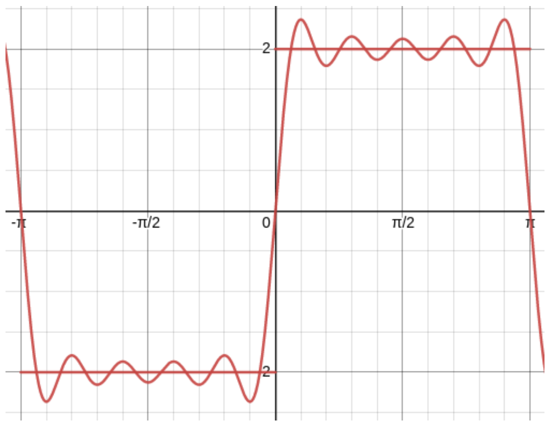
(a) n = 0



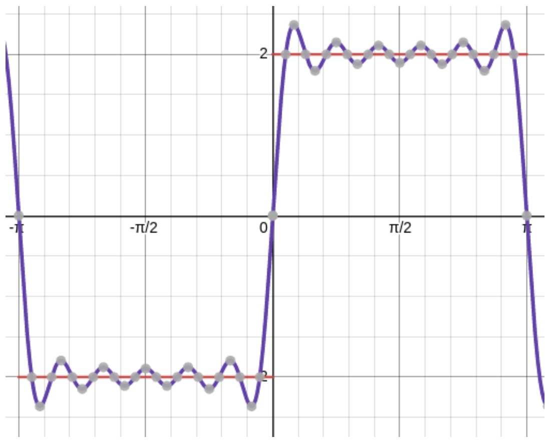
(b) n = 2



(c) n = 5



(d) n = 7



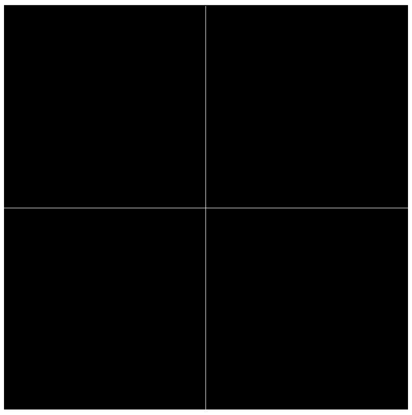
We now know what the pure form of Fourier series is, but now we need to use it to draw images, so from now on, we will keep on walking through all the concepts, but in the context of the program itself.

3 Overview of the program

3.1 Taking input

Input is taken from the user in form of an array of (X, Y) coordinates on a window prompted using SFML (Simple and Fast Multimedia Library) for C++ programming language (Figure 4). If the mouse is left-clicked and moved, the coordinates are recorded and saved in the array, as well as printed on the window.

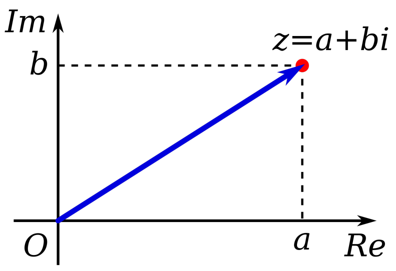
Figure 4



3.2 Implementation of complex numbers

The drawing is a set of points with x and y coordinates, however, all calculations done in the next stages need to be done on complex numbers. Akin to points that are described in terms of coordinates, complex numbers are also described in terms of two dimensions. One of them everyone knows really well, because this is the the real component, which, as the name suggests, consists of real numbers. The component that allows us to move vertically does not belong on the real number line because this number is $\sqrt{-1}$, usually denoted as i. So any complex number can be denoted as following: $a + bi$ and can be depicted like in figure 5 [1]

Figure 5



The implementation of complex numbers in code is a template structure containing two fields - real component (`re`) and imaginary (`im`).

```
template <class T>
struct Complex
{
    T re = 0,
      im = 0;
}
```

The most important thing, however, is overloading operators for this structure, which will make later code much more readable. Below is an example of overloading multiplication operator that will be used a lot in later code. The implementation stems from equation (11)

$$(a + bi) \cdot (c + di) = ac - bd + adi + cbi \tag{11}$$

```
    Complex operator*(Complex n)
    {
        Complex t;

        t.re = this->re * n.re - this->im * n.im;
```

[1]Graphic taken from `https://en.wikipedia.org/wiki/Complex_number`

```
        t.im = this->re * n.im + this->im * n.re;

        *this = t;

        return *this;
    };
```

3.3 Discrete Fourier transform (DFT)

Next step in the program is to find the coefficients for the Fourier series. However, the coefficients will be a bit different, since we are working with a set of complex numbers rather than a function. This means that the series will have a different form, but first we need to find the weights, which can be found using DFT, which is defined by formula (12).

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \left[cos\left(\frac{2\pi}{N}kn\right) - isin\left(\frac{2\pi}{N}kn\right)\right] \tag{12}$$

Now let us walk through what each variable means:

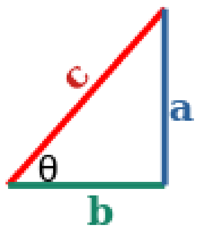
- X_k - output complex number of DFT
- N - size of input set
- k - frequency
- x_n - n^{th} complex number in the input set $\{x_0, x_1, ..., x_{N-1}\}$

The first thing to go through is why we are multiplying our input complex number by a complex sum?

3.3.1 Expressing a point on a unit circle using complex numbers

Let us take a right triangle with hypotenuse of length 1 (Figure 6) and angle θ between that hypotenuse (c) and the base (b) of triangle.

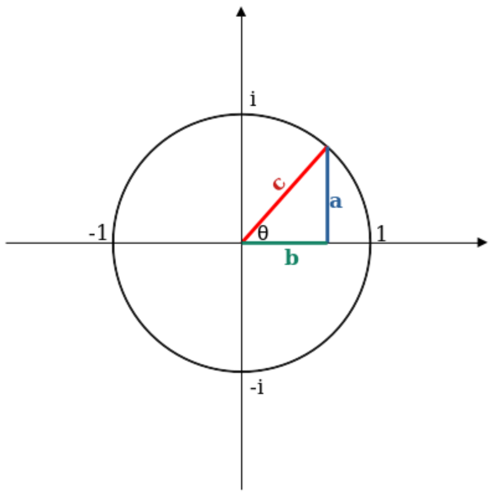
Figure 6



If we wanted to find the length of the base (b), it would be equal to the $cos\theta$ since the hypotenuse has the length 1. Similarly, if we wanted to find the height of the triangle (a), it would be the $sin\theta$.

Now we can inscribe such right triangle into a unit circle on a complex plane where the hypotenuse of the triangle would simultaneusly be the radius of the triangle (Figure 7) with some angle θ between radius and the horizontal axis.

Figure 7



We can find the coordinate of the point where the radius touches the circumference using the inscribed triangle, so $cos\theta$ would be the horizontal coordinate and $i \cdot sin\theta$ would be the vertical component since we are on complex plane. So the whole position can be described as $cos\theta + isin\theta$. Conversly, if $cos\theta + isin\theta$ are plotted for every θ between 0 and 2π, we get the same unit circle - Figure **??**

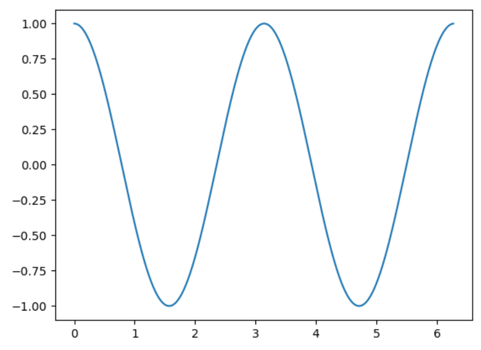
3.3.2 Applying Discrete Fourier Transform

The formula for DFT is a sum, however in order to visualize what is happening when applying DFT to a set of values, we will plot all values that are being summed, as well as the average of this sum (an average is taken normally when applying Fourier series, however, it can also be done during DFT, which will aid the readibility of the plots). Let us apply Discrete Fourier Transform to an easy-to- visualize example. We will plot all results from expression (13)

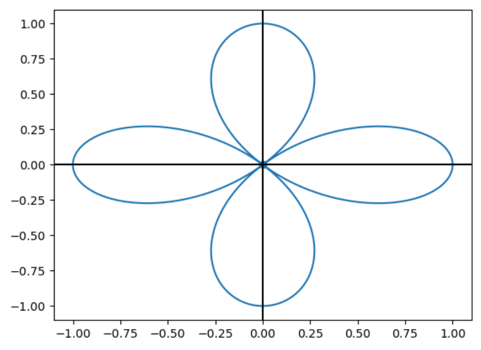
$$x_n \cdot \left[cos\left(\frac{2\pi}{N}kn\right) - isin\left(\frac{2\pi}{N}kn\right)\right] \tag{13}$$

where x_n are the elements of set, which is a discrete form of $cos(2\theta)$ from 0 to 2π, and k (or frequency) is 1, which means that all values will be mapped within one revolution around the unit circle (Figure 8, where subfigure 8a is the plot of set x and subfigure 8b is the plot of results) of the expression (13)

Figure 8



(a) $cos(2t)$

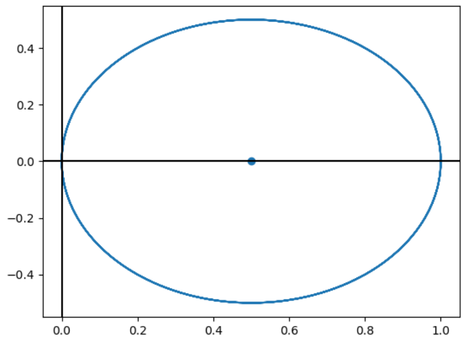


(b) $cos(2t)$ - wrapped

In plot 8b we can see that the values have aligned in a shape that is symmetric about real (horizontal) and imaginary (vertical) axes, which means for every element in the set, there is another one, which has an inverse value, which, after summing all values gives zero.

If we do the same operation, but now we change k to match the frequency of the input, which is 2, as the function $cos(2t)$, that the input set is a discrete form of, completes two cycles in one period. If we plot all of those values along with the average of the sum, we get what is shown in Figure 9.

Figure 9

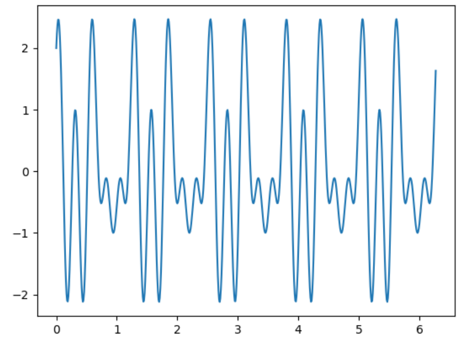


The shape that the values formed in this case is different, as it is only symmetric horizontally, which means that the imaginary components will cancel out, whereas the real components will give a non-zero value, which, after taking the average is in the center of this shape.

The results of DFT tell how much present a component that can be described as $cos\left(\frac{2\pi}{N}kn\right) - isin\left(\frac{2\pi}{N}kn\right)$ is in the set. The concrete values are used later on in phase where Fourier series is used.

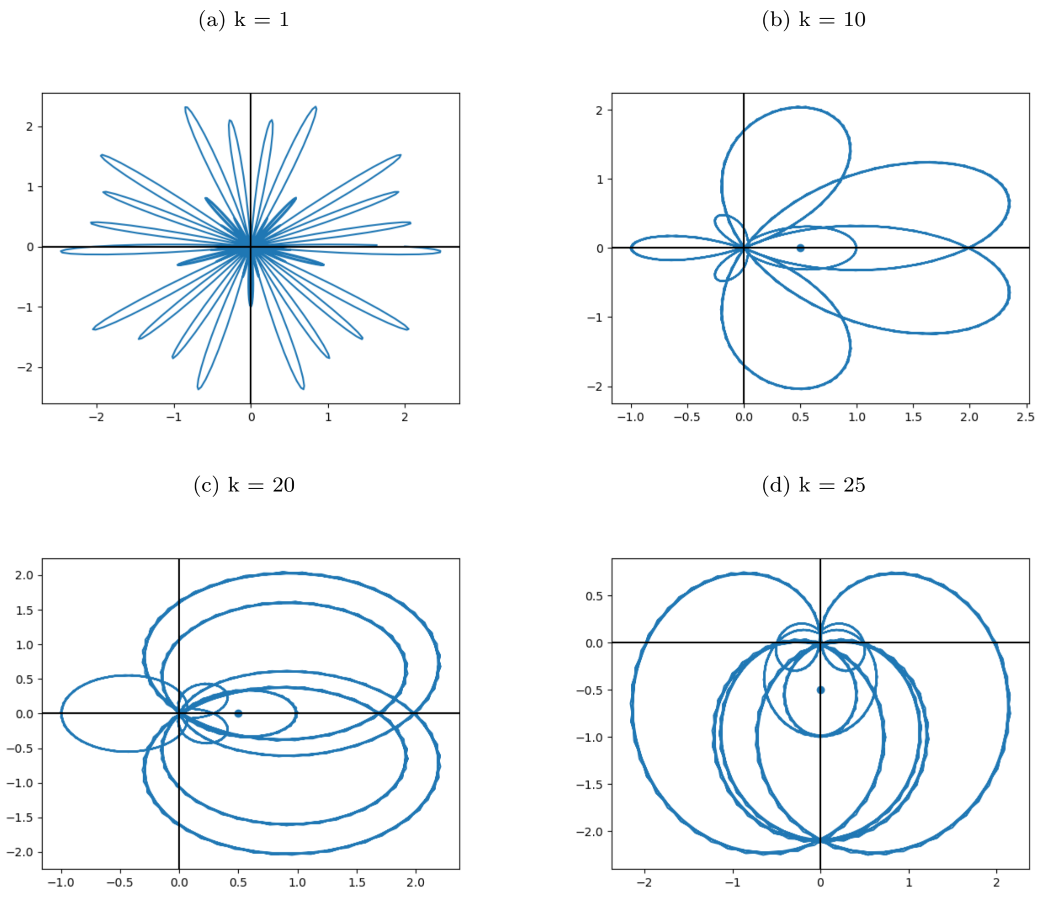
We can, however, take a look how the values will align when we will be applying DFT to sets that are a discrete form of sum of sinusoids. This is one of the most important uses of DFT, as sound signals are composed of many sine waves with different frequencies and DFT allows for finding those frequencies. As an example we shall take a function $f(t) = cos(10t) + cos(20t) + sin(25t)$, whose plot is shown in Figure 10. We can expect non-zero values for $k = 10, 20, 25$.

Figure 10



Below, in Figure**??** we can see the plots of all elements of input set multiplied by $cos\left(\frac{2\pi}{N}kn\right) - isin\left(\frac{2\pi}{N}kn\right)$. k in the captions shows the value of k used in the expression and the dots in the plots are the averages of the results of DFT. $\frac{1}{N}$

Figure 11



We can see that for values of k that correspond to the frequencies of the sinusoids making up the original function, the plots bias towards one side, which causes the sums of all those values to be bigger than zero.

See Appendix B for source code of script used to generate all plots above.

3.3.3 Why naive implementation of DFT is inefficient

Below is implementation of DFT that is inside the source code of the program.

```
C_set *DFT(C_set input)
{
    C_set *out = new C_set;
    C temp, X_k;
    unsigned N = input.size();
    double a = (2 * M_PI) / N, x;

    for (unsigned k = 0; k < N; ++k)
    {
        for (unsigned n = 0; n < N; ++n)
        {
            x = a * n * k;
            temp = C(cos(x), -sin(x));

            X_k += (temp * input[n]);
        }
        out->push_back(X_k);
        X_k = C(0,0);
    }

    return out;
}
```

Inside the function there are two for loops. The first one iterates through frequencies (k) as we want more coefficients than for one frequency. The inner loop is the exact implementation of the formula for DFT, where all terms from the input set are multiplied by $cos\left(\frac{2\pi}{N}kn\right) - isin\left(\frac{2\pi}{N}kn\right)$ and added together to compute the coefficient for frequency k. This is also the source of the inefficiency of this algorithm. Because there are two nested loops that go from 0 to $N-1$, the time complexity of this algorithm is $\mathcal{O}(n^2)$, which means that given an input of size n, the computer will make n^2 operations. This is very inefficient and slow for real-life applications of DFT, as the sizes of inputs for those applications can get very big. For example, a WAV file containing 1 second of audio signal has $44'100$ samples. Processing a short song would take a lot of computing resources. This is why in most actual implementations of DFT, an algorithm called Fast Fourier Transform (FFT) is used, whose time complexity is $\mathcal{O}(n\log(n))$.

3.4 Fourier series and making drawings

The last stage of the program is to recreate the original drawing using the inverse discrete Fourier transform (IDFT), which is the same as discrete Fourier series[2]. The formula is given in equation (14).

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot \left[cos\left(\frac{2\pi}{N}kn\right) + isin\left(\frac{2\pi}{N}kn\right)\right] \tag{14}$$

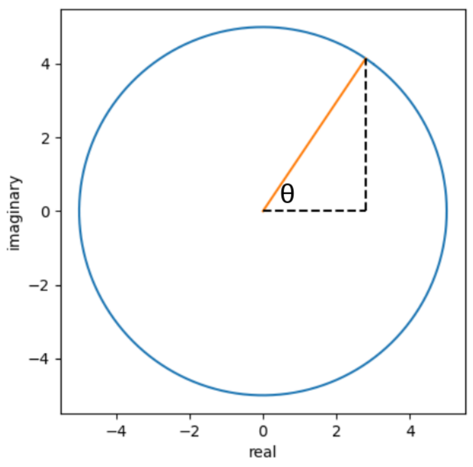
If we compare formula for IDFT (14) to the formula for Fourier series from section 2 (15), we can see the same concept in different executions.

$$S_\infty(t) = a_0 + \sum_{n=1}^{\infty} a_n cos\frac{2\pi nt}{P} + b_n sin\frac{2\pi nt}{P} \tag{15}$$

Inside both series a form of sinusoid is being multiplied by some coefficient. In the case of real form of Fourier series, an amplitude of a sine wave with a certain frequency is being regulated. In case of IDFT, where the coefficient is a complex number, the weight not only regulates the amplitude of the complex sinusoid $cos\left(\frac{2\pi}{N}kn\right) + isin\left(\frac{2\pi}{N}kn\right)$, but also its phase shift, which can be better explained pictorially. In Figure 12 we can see a term from a series whose coefficient is $\frac{1}{N}X_k = 3 + 4i$. In the plot, the circle is the path that this term would have made over the whole period, whereas the line from the center to a point on a circumference of the circle is the starting point. The radius of this circle is the hypotenuse of the triangle formed by the real and imaginary component of the term. The phase shift is angle θ.

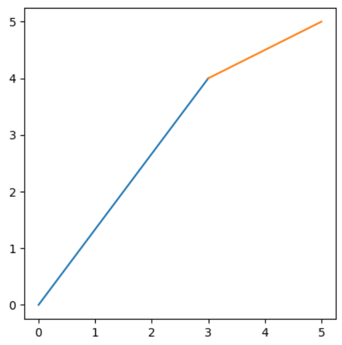
[2]terms IDFT and discrete Fourier series will be used interchangeably in this section

Figure 12



Because complex numbers are being added in the series, a good way to visualize this operation is by seeing complex numbers as lines that are "glued" together by their ends like in Figure 13

Figure 13



3.4.1 Implementation of IDFT in code

Below is an implementation of IDFT in the source code of the program.

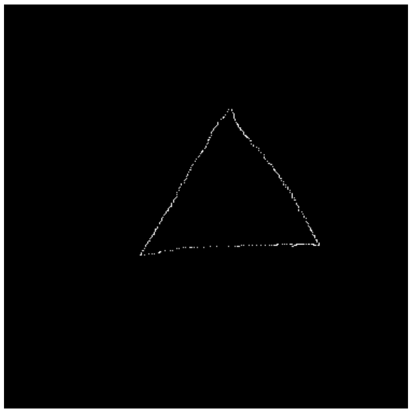
```
void draw_IDFT(C_set &x_k, VertexArray &drawing, unsigned n)
{
    C temp, x_n;
    unsigned N = x_k.size();
    double step = (2 * M_PI) / N, theta;

    for (unsigned t = 0; t < N; ++t)
    {
        for (unsigned k = 0; k < n; ++k)
        {
            theta = step * t * k;
            temp = C(cos(theta), sin(theta));

            x_n += (temp * x_k[k]) / N;
        }
        drawing[t] = Vector2f(x_n.re + 400, -x_n.im + 400);
        x_n = C(0,0);
    }
}
```

The function takes a set of coefficients computed by DFT function (`x_k`), an array of coordinates to be put on the screen (`drawing`) and a variable `n`, which allows for controlling how many terms are being added in the series. The upper for loop iterates over one period in $N - 1$ steps in order to get coordinates for a full period. The inner loop iterates from 0 to n so that the user can regulate the number of arguments being added in the series and inside this loop is a direct implementation of IDFT. At the bottom, the complex number `x_n` is converted to coordinates that work with SFML and is saved to the `drawing` array.

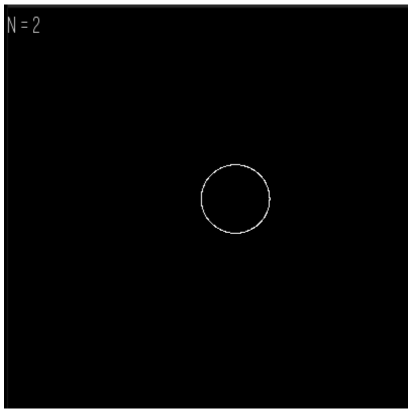
3.4.2 Inductive example of IDFT using the program

Figure 14

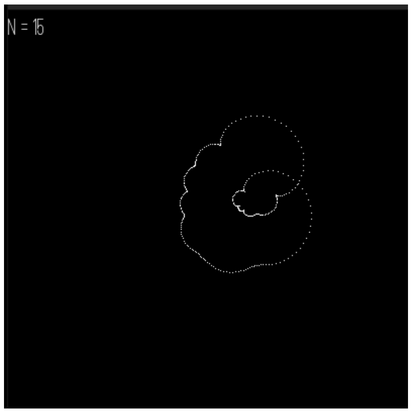
(a) original input



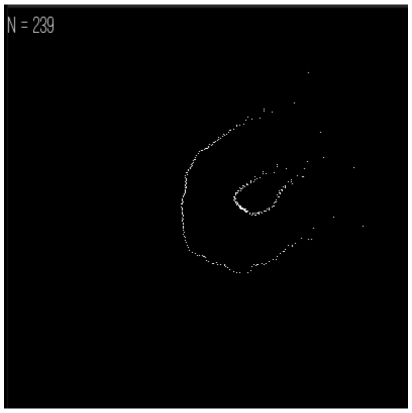
(b) n = 2



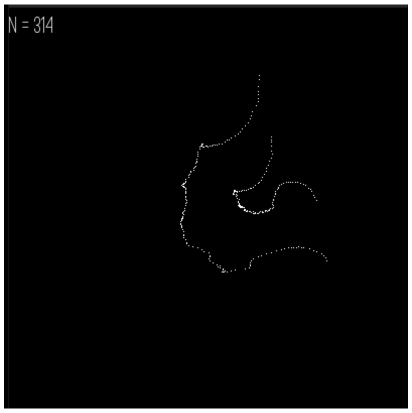
(c) n = 15



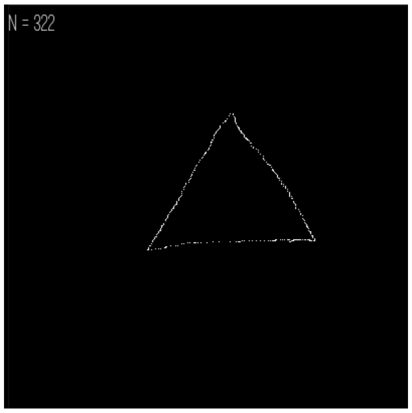
(d) n = 239



(e) n = 314



(f) n = 322



4 Drawings

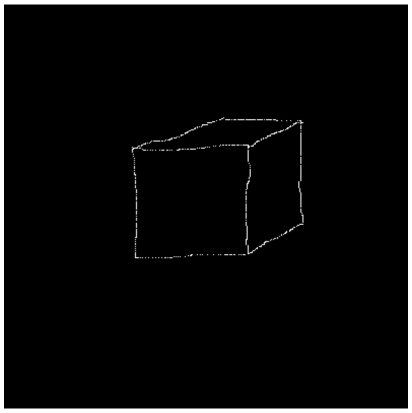
Below are sample examples of the program working. A few frames from the program have been picked in order to best depict the working of both program and discrete Fourier series.

4.1 Triangle

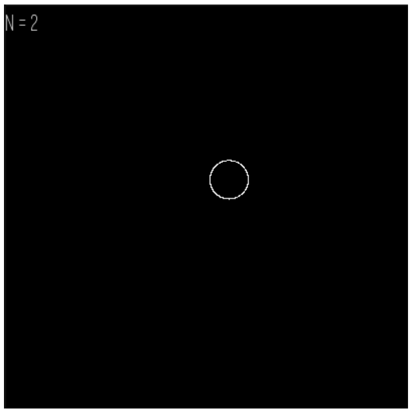
4.2 Cube

Figure 15

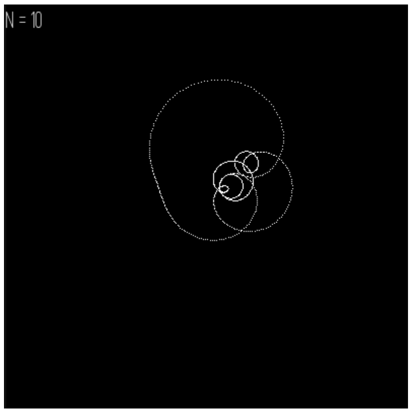
(a) original input



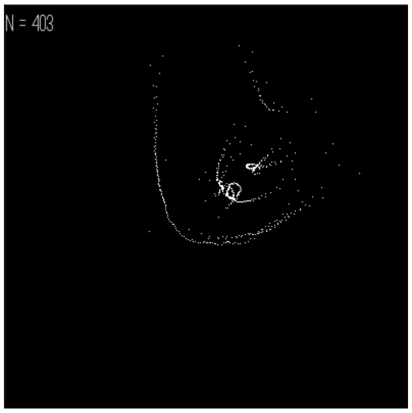
(b) n = 2



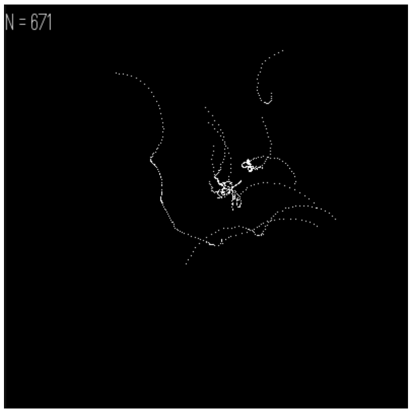
(c) n = 15



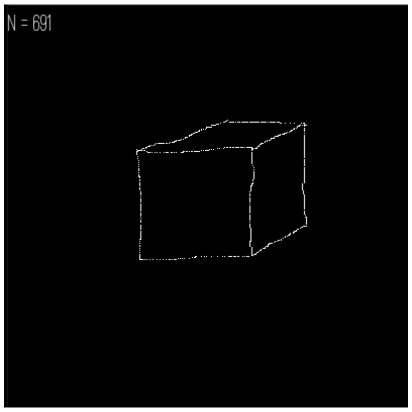
(d) n = 239



(e) n = 314



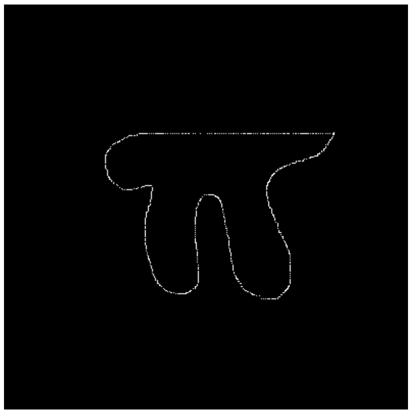
(f) n = 322



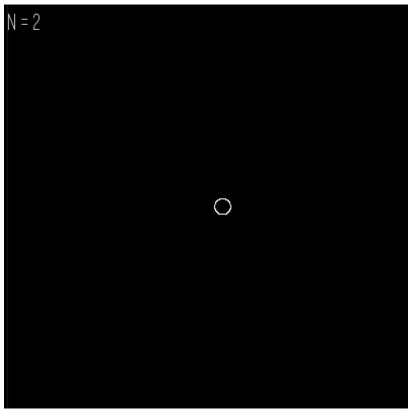
4.3 Pi

Figure 16

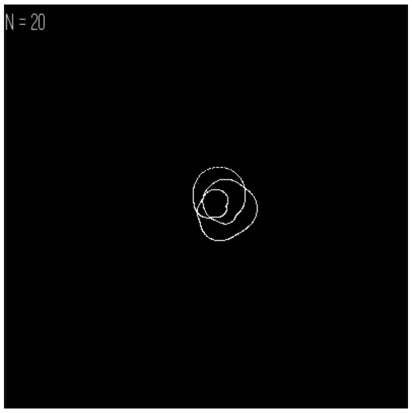
(a) original input



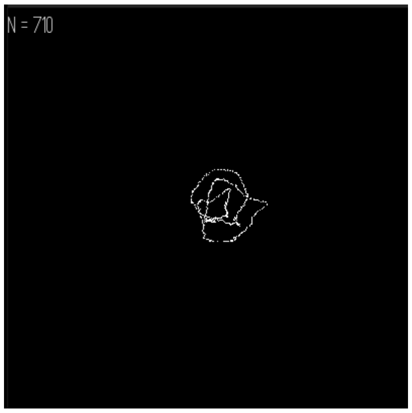
(b) n = 2



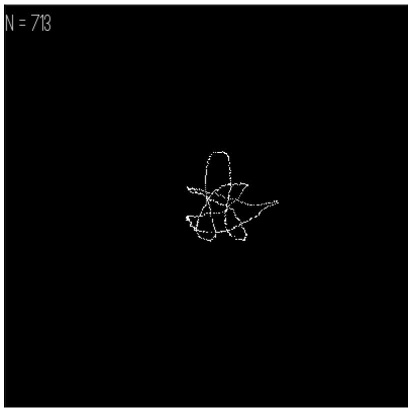
(c) n = 15



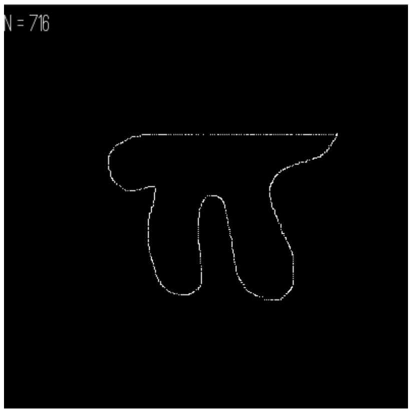
(d) n = 239



(e) n = 314



(f) n = 322



5 Appendices

5.1 Appendix A

5.2 Appendix B

Python script used to generate plots of functions mapped to unit circle.

```python
def wrap_function(func_in, freq):

    N = len(func_in)
    data = np.zeros(N, dtype = np.complex_)
    theta = (6.28 / N) * freq
    coeff = complex(0,0)

    for n in range(0, N):
        c = complex(m.cos(theta * n), -(m.sin(theta * n)))
        data[n] = func_in[n] * c
        coeff += data[n]

    coeff /= N

    return (data, coeff)
```

```
def plot_complex(data, name):

    plt.plot(data[0].real, data[0].imag)
    plt.scatter(data[1].real, data[1].imag)
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.savefig(sys.argv[1])

    plt.show()
```

Modules needed to run the script:

- numpy
- matplotlib
- math

5.3 Appendix C

Prerequisites

- gcc / Clang / LLVM compiler for C++11 and higher
- SFML media library

Compilation and use guide for Fourier drawer program for UNIX machines:

1. Clone git repository to your machine from the link: `https://github.com/ikrzywda/fourier_drawer`
2. change directory to `fourier_drawer/drawer`
3. use `make` to compile the program
4. type `./fourier_drawer` inside `drawer` directory to launch the program

Controls

- **left mouse button** to draw inside Fourier sketchpad window
- **Enter** to accept input given in Fourier sketchpad window and to terminate the program when in Fourier drawer window
- **K** to increase the number of arguments in Fourier drawer window
- **J** to decrease the number of arguments in Fourier drawer window