OWO Project
Epicycles

Avyay Irde, Vineet Tiwari

May 15, 2022

Abstract

In this report we discuss how we used epicycles to draw simple functions in python. The Discrete Fourier transform was implemented to calculate the Fourier coefficients for any given function and those calculated coefficients were used to draw epicycles for any given function. A further discussion was made onto changes that would be implemented if this project were to be taken up again.

Introduction

History of Epicycles

For centuries, the concept of epicycles—paths composed of circles moving along other circles—have been used to describe astronomical motions, ranging from the motions of our moon to the motion of other planets. In this paper we show how we can apply much of the same logic and methodology of the Fourier transform to epicycles. As a result, epicycles, while seemingly reasonable to describe astronomical paths for their time, are actually capable of describing any path, and do not actually provide much insight into the true motions of astronomical objects. While inconvenient for the ideas of classical astronomers, this same result means that we can use epicycles to create approximations of any shape, the main application of this paper. Even still, there remain improvements to be made and further exploration of this idea may be undertaken in the future.

Fourier Analysis

Fourier Series

A Fourier series is an expansion of a periodic function in terms of an infinite sum of sines and cosines. It make use of the orthogonality relationships of the sine and cosine functions. The computation and study of Fourier series is known as harmonic analysis and is extremely useful as a way to break up an arbitrary periodic function into a set of simple terms that can be plugged in, solved individually, and then recombined to obtain the solution to the original problem or an approximation to it to whatever accuracy is desired or practical.

In general, the Fourier series for any periodic function $f(x)$ of period $2L$ can be written as;

$$f(x) = \frac{a_o}{2} + \sum_{n=1}^{\infty} \Big(a_n cos(\frac{\pi n x}{L}) + b_n sin(\frac{\pi n x}{L}) \Big) \tag{1}$$

where a_o, a_n and b_n are known as Fourier coefficient that are calculated by using Fourier transform. Fourier transform basically applies the idea of Fourier series for a non-periodic function. A very "Math-English" kind of definition can be written as "The Fourier Transform takes a Time-based pattern, measures every possible cycle, and returns the overall 'cycle recipe' (the amplitude, offset, & rotation speed for every cycle that was found). Using Fourier transform, the Fourier coefficient can be calculated by the following formulas:

$$a_o = \frac{1}{L} \int_{-L}^{L} f(x) dx \tag{2}$$

$$a_n = \frac{1}{L} \int_{-L}^{L} f(x) cos(\frac{\pi n x}{L}) dx \tag{3}$$

$$b_n = \frac{1}{L} \int_{-L}^{L} f(x) sin(\frac{\pi n x}{L}) dx \tag{4}$$

The sinusoids are being scaled by using these Fourier coefficients a_o, a_n and b_n, where n can be interpreted as the integer multiple to the fundamental frequency of the function that gives us the frequency of the n_{th} sinusoidal wave in the sum.

Equation of Circle in Parametric form

In order to use the sinusoids to circles, we can either use the Euler's equation or we can use the equation of circles in parametric form. In this report, we will be using the later one. In parametric form, we represent our $(x,\ y)$ coordinates as $rcos(\theta)$ and $rsin(\theta)$ where (θ) is the angle between the line joining the $(x,\ y)$ point to the origin and the positive x-axis (going anti-clockwise) and r is the distance between the $(x,\ y)$ point and the origin.

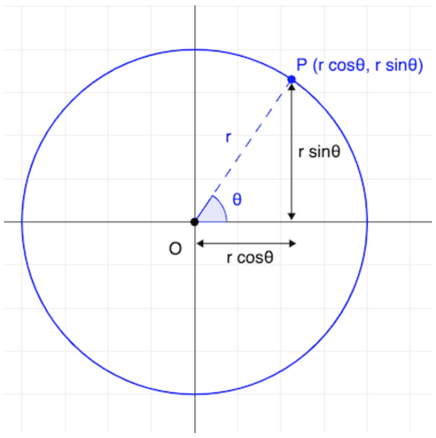

Figure 1: Representation of circle in Parametric form.

Thus, we can clearly see that by using the two parameters r and θ with our trigonometric functions, we can easily draw circles. Hence, transition from sinusoids to circle is not very difficult once we know what our r and θ are.

Models and Methods

Calculating Fourier coefficient of any function

As we have seen in the previous section, generally the Fourier coefficient of any given function can be calculated by the given formulas (2, 3 and 4). But, while working on a computer, we rather use the Discrete Fourier transform method to calculate the Fourier coefficients.
First, we took a triangular wave function of period 200 units as shown in the following figure;
The equation of the above triangular wave is given by,

$$f(x) = \begin{cases} x; \quad 0 \leq x \leq 100 \\ (100 - x); \quad 100 < x \leq 200 \end{cases}$$

Now, in order to determine the Fourier coefficient of the above triangular function, we use the following formula of Discrete Fourier Transform;

$$a_0 = \frac{1}{L}\sum_{i=0}^{200} f(i)dx$$

$$a_n = \frac{1}{L}\sum_{i=0}^{200} f(i)cos\Big(\frac{2n\pi X[i]}{L}\Big)dx$$

$$b_n = \frac{1}{L}\sum_{i=0}^{200} f(i)sin\Big(\frac{2n\pi X[i]}{L}\Big)dx$$

where L is the length of the period, which is 200 in our case, and dx is defined as the difference in length between $X[i]$ and $X[i+1]$. Also, i ranges from 0 t0 200 implies that we are adding 200 terms in our sum to get

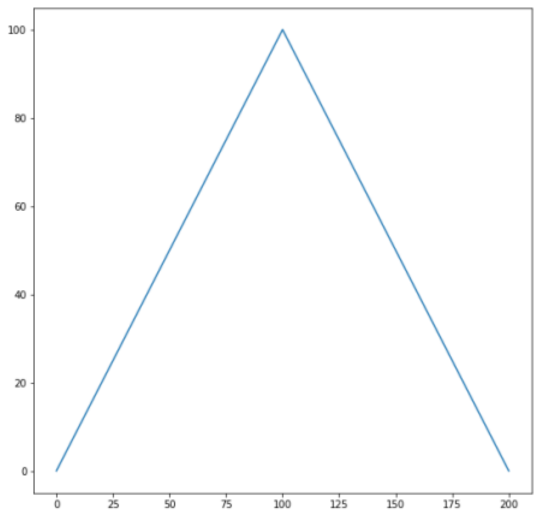

Figure 2: A simple triangular wave.

the value of the respective coefficient. As discussed earlier, on increasing the number of terms in the sum, we'll get more precise value of the coefficients. However, here we are keeping it to 200 only because we are getting decent results with this much value. Now, just by changing the values of n in the Fourier coefficients a_n and b_n, we'll get our various Fourier coefficient $a_1,\ a_2,\ a_3, ...$ and $b_1,\ b_2,\ b_3, ...$

Making Circles from Fourier coefficients

Now, as we have got these bunch of Fourier coefficients, how will me make circles out of these? It's much simpler than it looks like. All we have to know is that if we would have deal with these coefficients in a complex plane, we must have gotten a complex number as our Fourier coefficient, i.e. in the form of $(a_n + ib_n)$. Here, a_n and b_n are same as we got in our case. We can use either way to determine these coefficients (thanks to Euler), but all that we need to remember is that the norm of this complex number gives us the radius of the circle. Yes, it's that simple! Thus, the radius of the required circles can be written as,

$$r_n = \sqrt{a_n^2 + b_n^2} \tag{5}$$

But wait, we just discussed that we need at least two parameters to track a circle and those were the radius r and the angle θ. Here, the importance of θ is much more than just being an angle, rather it also determines the frequency of rotation of the radius vector as it start rotating, which we'll see later. As for now, let's just go with the fact that our θ can be defined as,

$$\theta = n \times \omega \times t \tag{6}$$

where ω is the natural frequency of our function, which is $\frac{2\pi}{L}$, n is the integral multiple to the natural frequency that gives us the frequency of rotation of n_{th} circle and t is the time. By using this relationship, we can draw various circles for every value of n. Each successive circle will have its centre on the circumference of the previous circle. However, the centre of the first circle is chosen to be the origin. Here is the demonstration of how it actually looks like for n=30;

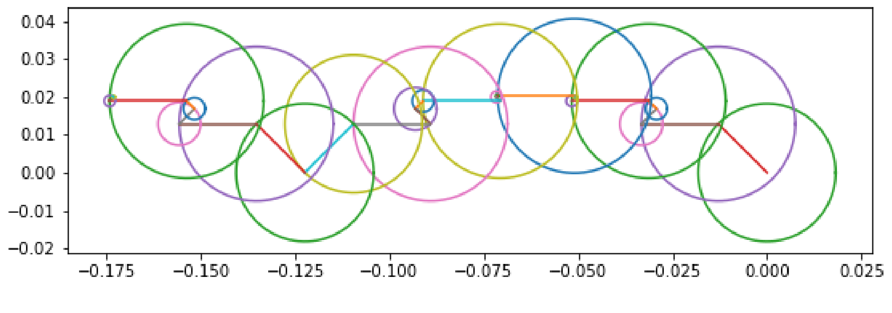

Figure 3: Demonstrating various Circles in an epicycle for n=30.

Lastly, in order to draw the function out of these circles, we use the equations of circles in parametric form that gives us the x and y coordinates of the rotating circle with changing time. Thus, by using equation-5 and 6 in the parametric form equation of a circle, we got our x and y values for each point as;

$$x_n = R_n cos(n \times \omega \times t) \tag{7}$$

$$y_n = R_n sin(n \times \omega \times t) \tag{8}$$

where we keep moving the time t to observe the motion of the tip of the radius vector at different instance of time. Now, as we get these x and y coordinates of all the n circles, we will add all the x and y positions of these circles to get the final position of the tip of n_{th} circle. Then, finally we will plot the x and y values of that final tip with change in time. This will give us our required function.

Turning Mathematics into Codes

We started out on pen and paper developing an algorithm to calculate and find the Fourier coefficients a given set of points. We initially started by using a model function that looked like the triangle shown in Figure 2:

```
Y=[]
X = np.arange(0,201,1)
for i in range(0,201):
    if(X[i] <= 100):
        Y.append(X[i])
    elif(X[i] > 100):
        Y.append(200-X[i])
```

This bit of code now gives a function $f(x)$ to work off of. The plot in Figure 2 was made by simply plotting $X[\,]$ & $Y[\,]$
Given this function now we had to find the Fourier coefficients for the given function. We know the equation and algorithm for finding these as defined in the previous part. In python this was done in the following way,

```
def a_n (y = Y, n=10):
    an = 0
    for i in range(0,len(y)):
        an += y[i] * np.cos(2*n*np.pi*(X[i]/L)) * dx
    return an/(L/2)

def b_n (y = Y, n=10):
    bn = 0
    for i in range(0,len(y)):
        bn += y[i] * np.sin(2*n*np.pi*(X[i]/L)) * dx
    return bn/(L/2)
```

Here we see we define 2 functions a_n & b_n which take 2 parameters, the function in question and n which is the nth coefficient to be found. This is also known as the Discrete Fourier Transform (DFT) and is a very general way of finding the Fourier coefficients of any given function. There are more advanced and efficient ways of calculating the Fourier coefficients, but for our purposes the DFT used works well enough.
At this point we have the ability to calculate the coefficients for any given function and up to any value of n. So, we now create 2 arrays of the coefficients to prevent having to call the function every time we need to use the coefficient.

```
n = 75
An = []
Bn = []

for i in range(0,n):
    An.append(a_n(Y,i))
    Bn.append(b_n(Y,i))

An = np.array(An)
Bn = np.array(Bn)
```

We here have found the coefficients up to $n = 75$. Now we can get to the epicycles bit of our project. We use the coefficients calculated to now calculate the the radius of each circle in our epicycle. This is also put into another array to simplify using the radii of all these circles.

```
R = []

for i in range(0,n):
    R.append(np.sqrt(An[i]**2+Bn[i]**2))
R = np.array(R)
```

Now that we have the radii of all the circles we can move ahead and start using the using the circles to redraw the original function. This was done by creating a time function using which we periodically rotate around the the circle. Each consecutive circles center lies on this rotating point of the previous circle. So for every given time we have to calculate the position of the point on the nth circle. This was implimented in the following way.

```
u = np.zeros((len(time),n))
v = np.zeros((len(time),n))
xn = np.zeros_like(time)
yn = np.zeros_like(time)
x0 = 0.0
y0 = 0.0

for i in range(0,len(time)):
    for j in range(n):
        x0 += R[j] * np.cos(j*w*time[i])
        y0 += R[j] * np.sin(j*w*time[i])
        u[i][j] = x0
        v[i][j] = y0
    xn[i] = x0
    yn[i] = y0
```

plotting xn & yn gives us,

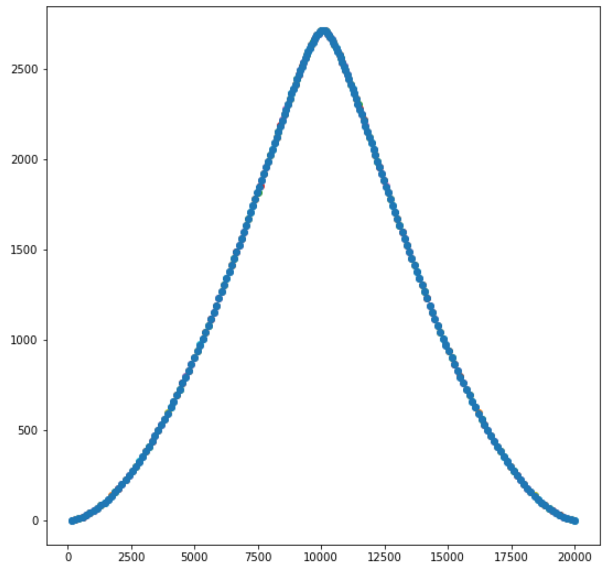

Figure 4: Triangular function plotted using circles.

And an animation of this process is also attached with this report.

The code was again run for another function which looked as follows,

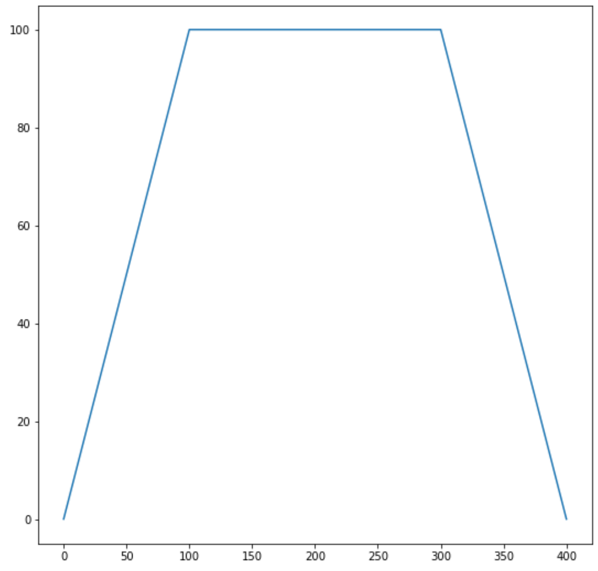

Figure 5: New test function.

And running our code on this function we get,

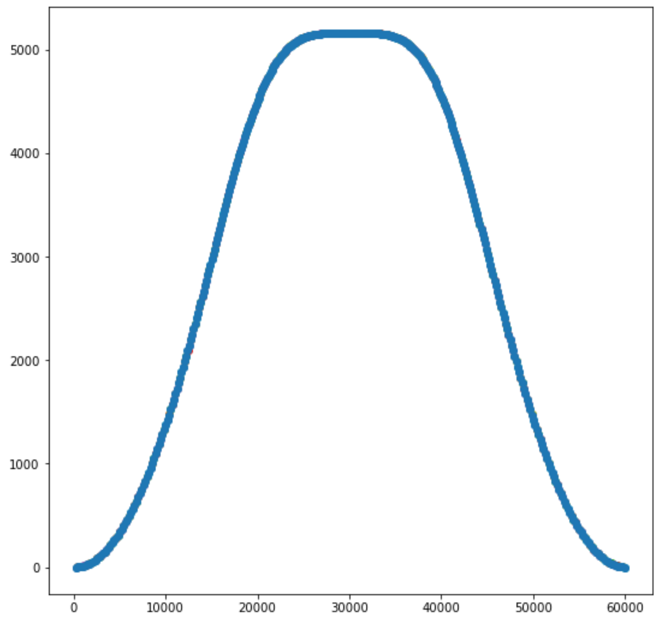

Figure 6: New function plotted using circles.

Results and Discussion

Epicycles were first developed in the the late third century BC and have been used for centuries until Newton and Leibniz developed calculus did they go out of fashion. The accuracy to which epicycles were able to predict the motion of planets was fascinating. The main thing that their usage here tells us is that the motion of planets is periodic. But, today apart from being a cool coding exercise and a great way to understand and appreciate the Fourier transform epicycles have very limited use today.

Further exploration

If we were to take up this project again there would be several things we would like to add and change in the execution.

- Add the animation for the circles that re-draw the function. This will obviously help in understanding what is actually happening in the included gif.
- Add a way to upload any image into the program and have it create an outline of the given figure and then generate epicycles for said image.
- Generate the Fourier coefficients using the complex numbers method, which is more efficient than the method used.
- Like many other implementations of such a program, have some way for the user to draw some figure and then have the program generate epicycles for the same.

Conclusion

As the screenshots of our example results show, our method does a fairly reasonable job of approximating the shape of input shapes, especially given the computational limitations and compromises that we made in the interest of saving time. However, it would have been even better if we would have made some even more complex functions by using epicycles. Our key motivation behind learning this way of drawing things is to draw the orbits of the various planets of our solar system and to compare it with the known paths. Such a comparison could shed light on how accurate these epicycles could have been.

References

1. https://www.amritaryal.com.np/posts/categories/mathematics-and-engineering/2.html
2. https://en.wikipedia.org/wiki/Deferentandepicycle
3. https://thecodingtrain.com/CodingChallenges/130.2-fourier-transform-drawing.html
4. https://www.myfourierepicycles.com/