Drawing with Fourier Series

Brock Taylor[1]

[1]*Department of Mechanical Engineering, Columbia University, New York, NY 10027*

This paper examines the practical implementation of Fourier Series to approximate an arbitrary complex function of time. The author wrote three programs in Python [1] to explore the use of Fourier Series for this purpose. The first program implements Fourier Series in one dimension for a well-defined function of x. The second program expands this principle by allowing the user to input a drawing of an arbitrary function of x. The third program generalizes to functions of t in the complex plane.

INTRODUCTION

A Fourier Series takes a function and deconstructs it into the superposition of an infinite number of sinusoidal waves. This paper examines the use of Fourier Series to take an arbitrary function of time (a drawing) and reconstruct it using a Fourier Series. In section , we examine the general principle of a Fourier Series. The author of this paper wrote a couple programs to help visualize the use of Fourier Series, which is also explained in detail in this section. Section generalizes the approach, looking at how an arbitrary function of time in the complex plane can be deconstructed using the same principle. The author of this paper also wrote a program implementing this principle, which is demonstrated in this section. Inspiration for this project was taken from Grant Sanderson, creator of 3Blue1Brown, an online educational platform. His website and work can be found here [2]. Alternatively, his YouTube channel can be found here.

FOURIER ANALYSIS

The term Fourier Series describes the decomposition of a function into the superposition of a series of sinusoidal waves (sine and cosine functions). The base equations are as follows, from page 34 of Stein et al's Fourier Analysis. [3]

$$f(x) = \sum_{n=-\infty}^{n=\infty} a_n e^{(n*2\pi i x)/L} \tag{1}$$

$$a_n = \frac{1}{L}\int_0^L f(x) e^{(-n*2\pi i x)/L} dx \tag{2}$$

The function f(x) can be decomposed into an infinite sum of terms comprised of some coefficient a_n and some complex exponential. Expanding the summation and manipulating some yields the equations below, which are perhaps more easily understood in the context of the superposition of sinusoids.

$$f(x) = \sum_{n=0}^{n=\infty} A_n \cos\frac{n*2\pi x}{L} + B_n \sin\frac{n*2\pi x}{L} \tag{3}$$

$$A_n = \frac{2}{L}\int_{-L/2}^{L/2} f(x)\cos\frac{n*2\pi x}{L} dx \tag{4}$$

$$B_n = \frac{2}{L}\int_{-L/2}^{L/2} f(x)\sin\frac{n*2\pi x}{L} dx \tag{5}$$

Looking at only the cosine function in equation 3, given a function f(x) that is L-periodic (a requirement for it to be deconstructed into a Fourier Series), the $\frac{2\pi}{L}$ term translates the cosine function to also be L-periodic in order to match the function. The inclusion of n means that the cosine function will be some integer multiple of L-periodic, and allows for higher frequency cosine functions to be present for higher levels of n (for more accurate approximation). The very same analysis describes the behavior of the sine function. The coefficient A_n represents a sort of average value for the function f(x) over L normalized by an L-periodic cosine function. The same is true for B_n, except it is normalized over L by an L-periodic sine function.

To better understand how to draw with Fourier Analysis, a program was written in Python [1] with the purpose of constructing a Fourier Series given a set of data points. Firstly, a simpler program was developed to deconstruct an arbitrary function into a superposition of sine and cosine functions. The program can be found here. An arbitrary one dimensional function can be input in the program (by default, it takes the polynomial function $y = x^4 + 3x^2$) and a Fourier Series is constructed for a set number of terms based off of desired accuracy (increasing the number of terms in the series increases the accuracy of the approximation). The following image demonstrates the use of the program on the above polynomial function over the interval 0 to 10 using 10 terms in the series.

The program works by numerically calculating the integrals required to find coefficients A_n and B_n to build the series based off of equations 3, 4, and 5. It uses

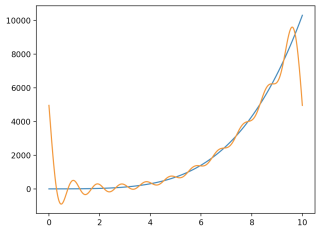

FIG. 1. Polynomial Fourier Series

Numpy [4], and Scipy [5] to compute the calculations, then uses matplotlib [6] to graph the resulting Fourier Series approximation.

Building upon this principle, the program that can be found here first requests the user to input a drawing of an arbitrary one dimensional function of x (behavior is undefined if the user inputs more than one y-value for a single x-value). The program reads the drawing as an array of RGB values for each pixel and translates the it into a set of x-y pairs. It then conducts the same numerical Fourier analysis explained above on these coordinates. This program uses Numpy [4], Scipy [5], and matplotlib [6] to do the computations to construct a Fourier Series and plot the result. It also uses Python Imaging Library (PIL) [7] and tkinter [8] to collect user input. The process of collecting user input in this manner is based off a discussion on the online Q&A platform stackoverflow, in which it was described by user Mohammed Janati Idrissi [9]. Below is an example of user input being interpreted by the program.

FIG. 2. Arbitrary Function Input

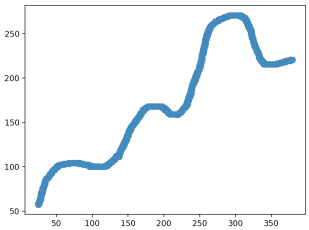

FIG. 3. Arbitrary Function Plot

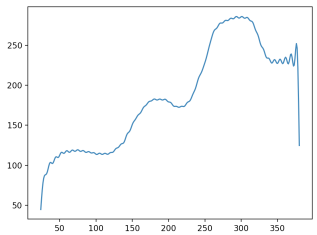

FIG. 4. Arbitrary Function Fourier Series

Figure 2 is an example of a drawing of an arbitrary function that the user inputs into the program. Figure 3 demonstrates the plot of the x-y pairs that the program records for the drawing and the Fourier Series of these x-y pairs interpreted as an arbitrary function of x. The integrals required for Fourier analysis are done numerically as a trapezoidal summation. Using numerical calculations of the integrals required for Fourier Series allows for the decomposition of arbitrary functions of x over a set interval $x_1 \leq x \leq x_2$

Difficulty sometimes arises in processing the image provided by the user. The library used only paints pixels at a set rate, the user drawing too quickly will result in gaps between pixels. This causes undefined behavior when deconstructing the function into a Fourier Series. This can be seen in Figures 4 and 5 below.

FIG. 5. Arbitrary Function Drawing With Incorrect Input

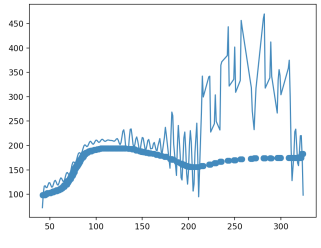

FIG. 6. Arbitrary Function Drawing With Incorrect Input Series

DRAWING WITH FOURIER

Expanding the previous analysis to the vector space of the two-dimensional complex plane allows more complex images to be created using Fourier Series. This analysis takes advantage of the ability of the complex exponential to represent circular motion in the complex plane. This allows one to decompose an arbitrary line drawing into a superposition of an infinite number of circular motions. Given an arbitrary function of time f(t) which represents a line drawing in the complex plane, decomposition into a Fourier Series is as follows:

$$f(t) = \sum_{n=-\infty}^{n=\infty} c_n e^{n*2\pi it} \tag{6}$$

Note that this is the same series as equation 1 generalized as a vector function of t. Generalizing the process to arbitrary functions of time allows for a more nuanced explanation of the summation. Examining the inclusion of $-\infty$ as the lower bound of summation, think of the process of the summation of complex exponentials rotating in the complex plane. Including the bound $-\infty$ allows clockwise-rotating vectors which, in the context of drawing, is necessary for accuracy. Below is the analysis for the first term of the series.

$$c_0 e^{0*2\pi it} \qquad c_0 = \int_0^1 f(t)dt \tag{7}$$

The 0 term of the series, shown above, represents the midpoint of the drawing (consistent with the 0 term of a Fourier series representing the average value). Note the 0 in the exponent as well. This term will not be rotating, it will be a constant vector in the complex plane, onto which all subsequent vectors will be added. The equation to the right allows for one to solve for the coefficient. Consistent with the first term representing a constant average position of the drawing, this equation provides the average value for f(t) (leaving out 1/1 for simplicity). Since all elements of the series complete a whole number of cycles in t = 1 seconds by construction, the integral need only be taken between bounds 0 and 1 to complete the whole image. Following the same logic, a similar formula can be derived for an arbitrary constant c_n

$$c_n e^{n*2\pi it} \qquad c_n = \int_0^1 f(t)e^{-n*2\pi it}dt \tag{8}$$

The presence of -n serves to zero out the exponent of the term corresponding to the desired coefficient. This causes the desired term to no longer rotate, and the same average value problem as seen above can be solved to find the coefficient. Meanwhile, any other term in the series will rotate a whole number of times, resulting in the integral equaling zero.

If f(t) is a complex vector function of t, these integrals may be well-defined. However, given an arbitrary drawing, it is better to numerically compute the integrals. By choosing a number of elements corresponding to desired accuracy and solving for the coefficients corresponding to these terms, one may construct a Fourier Series which approximates an arbitrary vector function in the complex plane.

$$f(t) \approx \sum_{n=-k}^{n=k} c_n e^{n*2\pi it} \tag{9}$$

The variable k in the above equation determines the number of elements in the series ($N = 2k+1$) and therefore the level of accuracy with which the series approximates the input function. The program found here implements the process described above using Python [1]. Below are sample images produced by the program for various values of k in equation 9. Various levels of accuracy are presented to demonstrate how the series approximates the image.

FIG. 7. Input Image

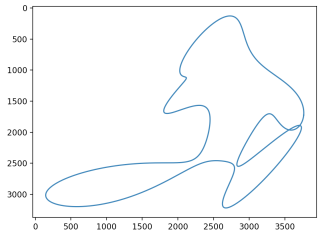

FIG. 8. Fourier Approximation using n = 10

Unlike the previously discussed program which takes an arbitrary function of x as its input, this program takes an SVG file as input. From the SVG input, it uses svgpathtools [10] to interpret the file as a set of polynomial paths. The full image is parameterized by finding the time over which the function should trace each path. This is done by dividing the path length by the total length of all paths in the image. Since the full image should be traced in a time of one second, this percentage is the time

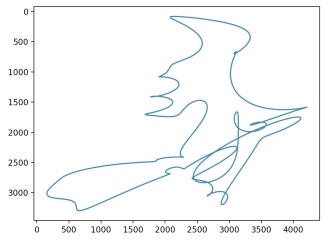

FIG. 9. Fourier Approximation using n = 25

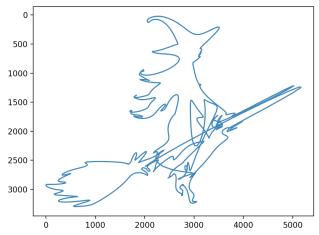

FIG. 10. Fourier Approximation using n = 100

over which the function should trace a given path. By doing this for each path, the image can be deconstructed into a piecewise function of time where each piece is a complex polynomial.

The coefficients, c_n from equation 8 are computed numerically by evaluating f for 1000 values of time between 0 and 1. The real and imaginary components of the integral are then computed using SciPy [5] and summed. The series itself is computed numerically as well, and the real and imaginary parts can be plotted in the complex plane using matplotlib [6]. Numpy [4] and Pandas [11] [12] for ease of computation and managing variables

CONCLUSIONS

We presented an explanation of Fourier Series and a few applications in both the real and complex planes. All programs discussed can be found here.

Fourier_Func.py implements the first program discussed in this paper. It takes a one-dimensional function of x as input and computes the Fourier Series approximation between 0 and 10. It then plots the original function alongside the Fourier Series.

The program main.py adds user input, allowing the user to draw an arbitrary function of x in one dimension. It then does the same calculations to create a Fourier Series approximation of this function and plots the results.

Drawing.py is the final program discussed. It takes an SVG file as input. It then interprets the image as a piecewise function of time over the interval $0 \leq t \leq 1$ in which each piece is a complex polynomial. Computing the required integrals numerically allows for the construction of the Fourier Series, which is a set of complex numbers. Plotting these yields the resultant images: Figures 7, 8 and 9.

Fourier Series are a powerful method of approximating an arbitrary function and, as discussed in section and implemented in Drawing.py, can be generalized to vector functions including the complex plane.

ACKNOWLEDGMENTS

The author is grateful to Professor Marka for inspiring and teaching young researchers. The author is grateful to Grant Sanderson of 3Blue1Brown for inspiring this project. The author would also like to thank the teams of all Python libraries used and the Python team themselves for providing the tools to demonstrate the principles discussed in this paper.

[1] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual* (CreateSpace, Scotts Valley, CA, 2009).
[2] G. Sanderson, "3Blue1Brown online educational resource," (2021).
[3] E. M. Stein and R. Shakarchi, *Princeton Lectures in analysis: Fourier Analysis* (Princeton Univ. Press, 2003).
[4] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, Nature **585**, 357 (2020).
[5] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, Nature Methods **17**, 261 (2020).
[6] J. D. Hunter, Computing in Science & Engineering **9**, 90 (2007).
[7] P. Umesh, CSI Communications **23** (2012).
[8] F. Lundh, URL: www. pythonware. com/library/tkinter/introduction/index. htm (1999).
[9] M. J. Idrissi, "Jupyter notebook: let a user inputs a drawing," (2019).
[10] A. Port,.
[11] T. pandas development team, "pandas-dev/pandas: Pandas," (2020).
[12] Wes McKinney, in *Proceedings of the 9th Python in Science Conference*, edited by Stéfan van der Walt and Jarrod Millman (2010) pp. 56 – 61.