Process for Forward Fast Fourier Transform (FFT) Numerical Computation

Overview

Forward FFT takes time domain data points and converts them to frequency domain coefficients. First it solves for complex numbers (C_k), then converts to a, b coefficients for use in partial sums trigonometric interpolation equation.

Prerequisites

- N = Number of data points. N must be a power of 2, can zero-pad otherwise. Data points must be evenly spaced on $[-\pi, \pi]$.
- m = Degree. Higher solves for higher frequencies in the approximation. $m \leq N/2$

Step 1) Generate Data Points

For data points (x_j, y_j) on $[-\pi, \pi]$:

$$x_j = -\pi + \frac{2\pi j}{N}, \quad j = 0, 1, \ldots, N-1$$

$$y_j = f(x_j), \quad j = 0, 1, \ldots, N-1$$

Step 2) Pre-compute setup values:

a) Pre-computed "Roots of Unity":

$$W_k = e^{-2\pi i k/N} = \cos(2\pi k/N) - i\sin(2\pi k/N), \quad k = 0, 1, \ldots, (N/2)-1$$

Note: Only need to compute each W_k once.
Note: W values "loop" around a unit circle for higher values.

ex: $N = 8$, with $\log_2(8) = 3$:

$$W_0 = e^{-2\pi i(0)/8} = 1.0000 + 0.0000i \quad (0°)$$
$$W_1 = e^{-2\pi i(1)/8} = 0.7071 - 0.7071i \quad (45°)$$
$$W_2 = e^{-2\pi i(2)/8} = 0.0000 - 1.0000i \quad (90°)$$
$$W_3 = e^{-2\pi i(3)/8} = -0.7071 - 0.7071i \quad (135°)$$

$$W_4 = -W_0 = -1.0000 + 0.0000i \quad (180°)$$
$$W_5 = -W_1 = -0.7071 + 0.7071i \quad (225°)$$
...

Loop rules:

- $W_{k+4} = -W_k$ (opposite side of circle)
- $W_{k+8} = W_k$ (full rotation)

b) Initial complex values array:

i) Set initial complex values to y values:

$$C_j = y_j, \quad j = 0, 1, \ldots, N-1$$

ii) Bit-reverse the complex values index positions so butterfly pairing is correct. For array size N, each index i (for 0 to $N-1$) is found by:

(a) Convert to binary in $\log_2(N)$ bits
(b) Reverse the bits
(c) Convert back to decimal
(d) Re-order complex pairs with new index

Ex: $N = 8$, with $\log_2(8) = 3$ bits:

i	binary	reversed	new i
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

$$C_0 \to C_0, C_1 \to C_4, C_2 \to C_2, C_3 \to C_6, \text{ etc.}$$

Note: Only need to compute bit-reversal index table once.

Step 3) Butterfly Pairs

Combine complex values in pairs over stages, updating values one step at a time, until the final complex values are fully computed. Pairs of C_k are found by increasing the distance when grouping the arrays:

- Stage Number: $s = 1, 2, \ldots, \log_2(N)$
- Distance between pairs in stage: $d = 2^{s-1}$

For example, with $N = 8$ and $\log_2(8) = 3$ stages:

Stage 1 (distance = 1): $[0], [1], [2], [3], [4], [5], [6], [7] \rightarrow \{(0,1), (2,3), (4,5), (6,7)\}$
Stage 2 (distance = 2): $[0,1], [2,3], [4,5], [6,7] \rightarrow \{(0,2), (1,3), (4,6), (5,7)\}$
Stage 3 (distance = 4): $[0,1,2,3], [4,5,6,7] \rightarrow \{(0,4), (1,5), (2,6), (3,7)\}$

For each pair, perform the following (N = total # of points, d = distance between pairs):

$$\begin{aligned} \eta &= C_{k+d} \times W_{(k \times N/2d)} \\ C_k &= C_k + \eta \\ C_{k+d} &= C_k - \eta \end{aligned}$$

Step 4) Coefficient Extraction

Convert complex values to a_k and b_k values:

$$a_0 = \frac{1}{N} Re(C_0)$$

$$-a_k = \frac{2}{N} Re(C_k) \qquad b_k = \frac{2}{N} Im(C_k) \quad \text{, for odd } k = 1, 3, \ldots, (N/2) - 1$$

$$a_k = \frac{2}{N} Re(C_k) \qquad -b_k = \frac{2}{N} Im(C_k) \quad \text{, for even } k = 2, 4, \ldots, (N/2) - 1$$

$$a_{N/2} = \frac{1}{N} Re(C_{N/2}), \quad \text{for } k = N/2$$

Step 5) Construct Fourier series approximation

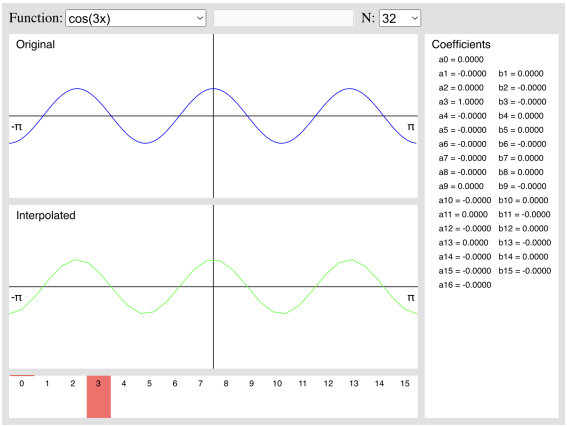
Interpolating Polynomial Equation for FFT:

$$S_m(x) = \frac{a_0 + a_m \cos(mx)}{2} + \sum_{k=1}^{m-1} (a_k \cos(kx) + b_k \sin(kx))$$

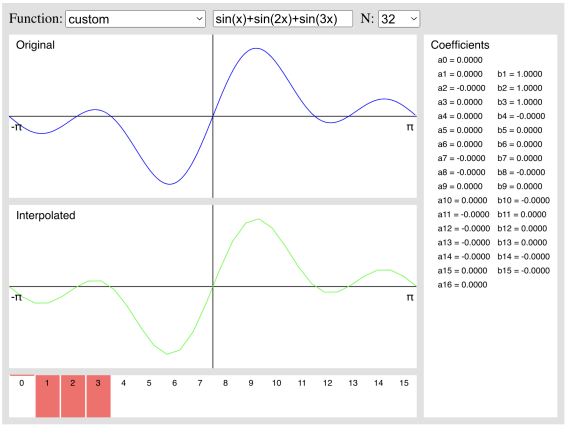
Numerical Computation Implementation

To validate the FFT process above was correct we implemented it from scratch in p5.js, creating an interactive web interface on top that allows users to analyze arbitrary functions. User selected functions are sampled at N points between $-\pi$ and π, our FFT computes the complex coefficients and derives the a_k and b_k coefficients. Those coefficients are used to construct the Fourier series partial sum approximation, which is then graphed below the original function for comparison. The magnitude of each coefficient calculated and displayed on the bottom. The user can either select from predefined functions or input custom functions, and adjust the N sampling resolution.

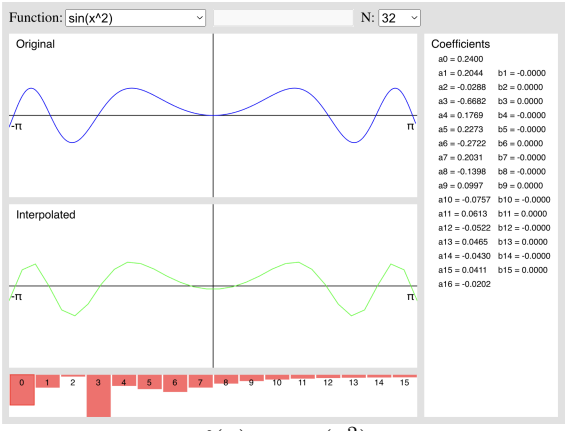
Click Play icon in top left to run the app:
`https://editor.p5js.org/travisformayor/sketches/by5UTP__f`



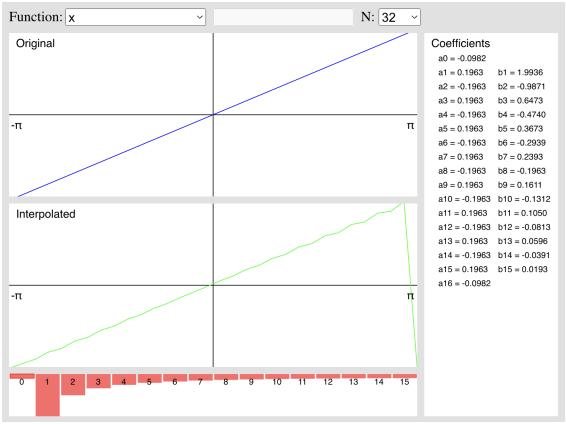
$f(x) = cos(3x)$
Notice 3rd spectrum.



$f(x) = sin(x) + sin(2x) + sin(3x)$
Notice 1, 2, and 3 spectrums.



$f(x) = sin(x^2)$



$f(x) = x$
Notice the break on the end. See below for more information on the Gibbs Phenomenon.

Gibbs Phenomenon occurs near points in a Fourier transformation involving a complete revolution of the unit circle. Since FFT trigonometric interpolation is constructed using a combination of sine and cosine functions to create a best fit, it utilizes the periodicity of the unit circle (2π to make a complete revolution) and, therefore, has large jumps at points of discontinuity. Take, for

example, the function $f(x) = x$ (pictured above). Since x is not periodic and is instead monotonically increasing, once the interpolation of the function nears points that are a multiple of $\pm\pi$, the natural periodicity of the sine and cosine functions being used to make the best fit of the function have a huge jump as they attempt to "reset" to their negative side of the unit circle. This does not happen to functions in the program that are periodic, such as $\sin(x)$, $\cos(x)$, x^2, or $|x|$, but other examples where it does occur would be any constant times x ($2x$, $3x$, ...), e^x, or x to an odd degree (x^3, x^5, ...).