Lab 4: April 27-28, 2023 (Fourier series)

Naga Kandasamy

Drexel University
Department of Electrical and Computer Engineering

Spring 2023
ECE 105: Programming for Engineers II

Outline

Lab 4: Fourier series — goals

Goals for Lab 4

- Practice using `numpy` and `matplotlib`
- Practice vectorization
- Practice Riemann sum to approximate an integral
- Practice Fourier series representations of periodic signals

Lab 4: Fourier series — overview

- Challenge: Given a periodic function $f(x)$, periodic over the interval $[-\pi, +\pi]$, approximate f using only sines and cosines
- Basis functions: a family of simple functions from which more complex functions may be represented or approximated: here, basis functions are $\sin(nx)$ and $\cos(nx)$ for $n \in \mathbb{N}$ and $x \in [-\pi, +\pi]$
- Solution: the N-term Fourier series approximation of $f(x)$ is:

$$f(x) \approx \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(nx) + b_n \sin(nx)$$

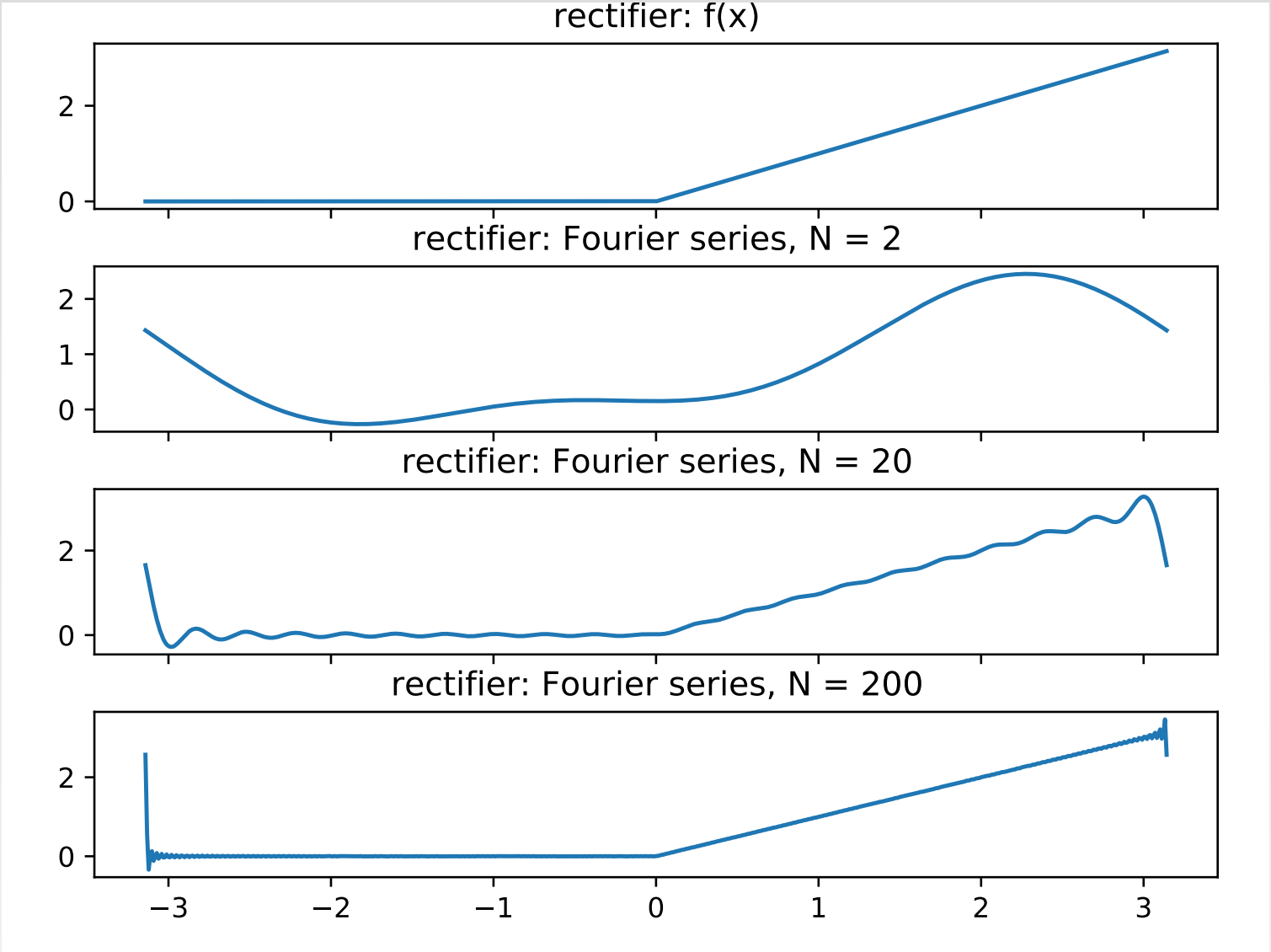
where a_n, b_n are the Fourier series coefficients given by:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{+\pi} f(x) \cos(nx) \mathrm{d}x$$

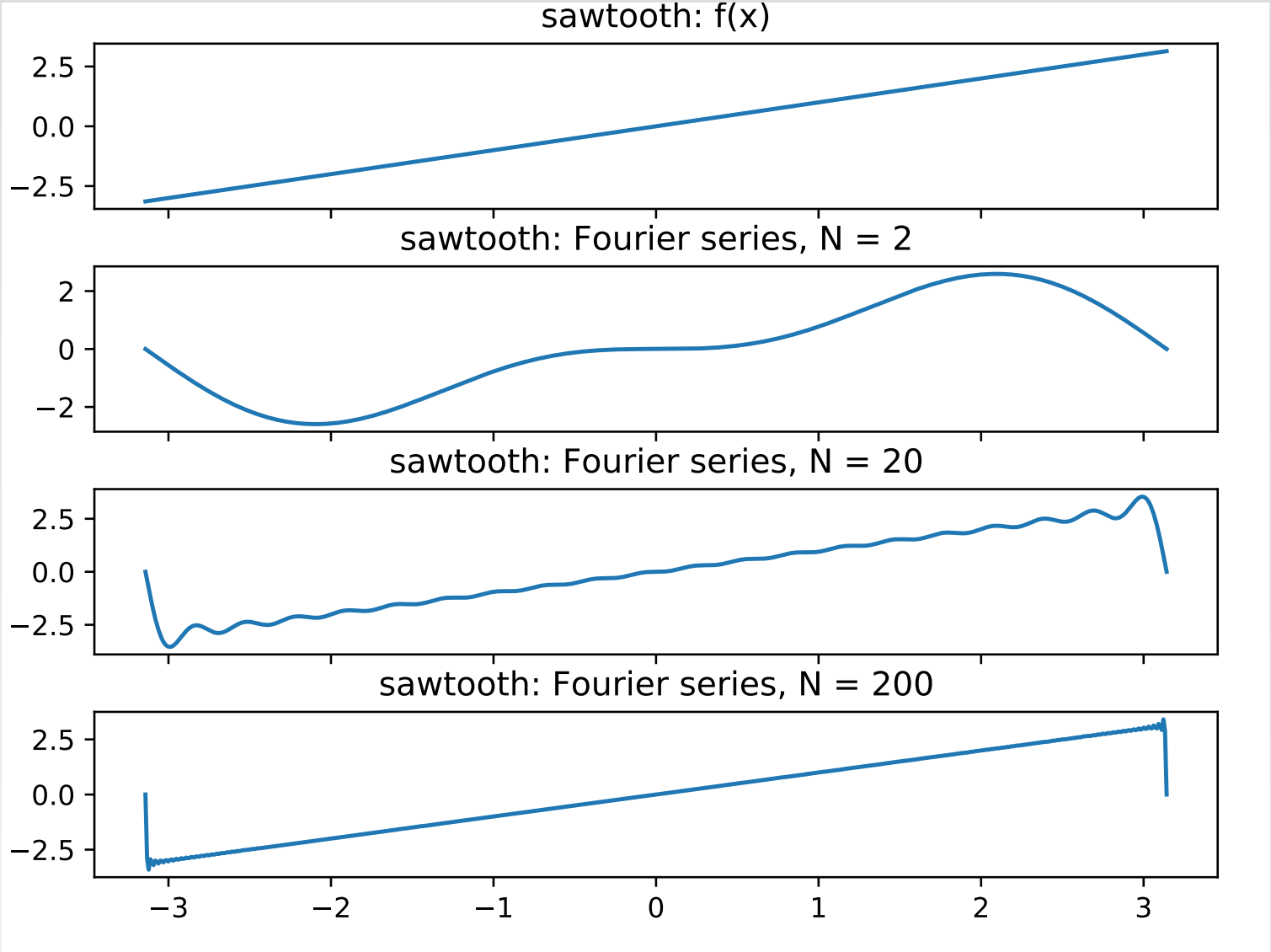
$$b_n = \frac{1}{\pi} \int_{-\pi}^{+\pi} f(x) \sin(nx) \mathrm{d}x$$

- Intuition: (a_n, b_n) are the correct weights on the coefficients of $\cos(nx)$ and $\sin(nx)$ to best represent the function f; approx. improves in N

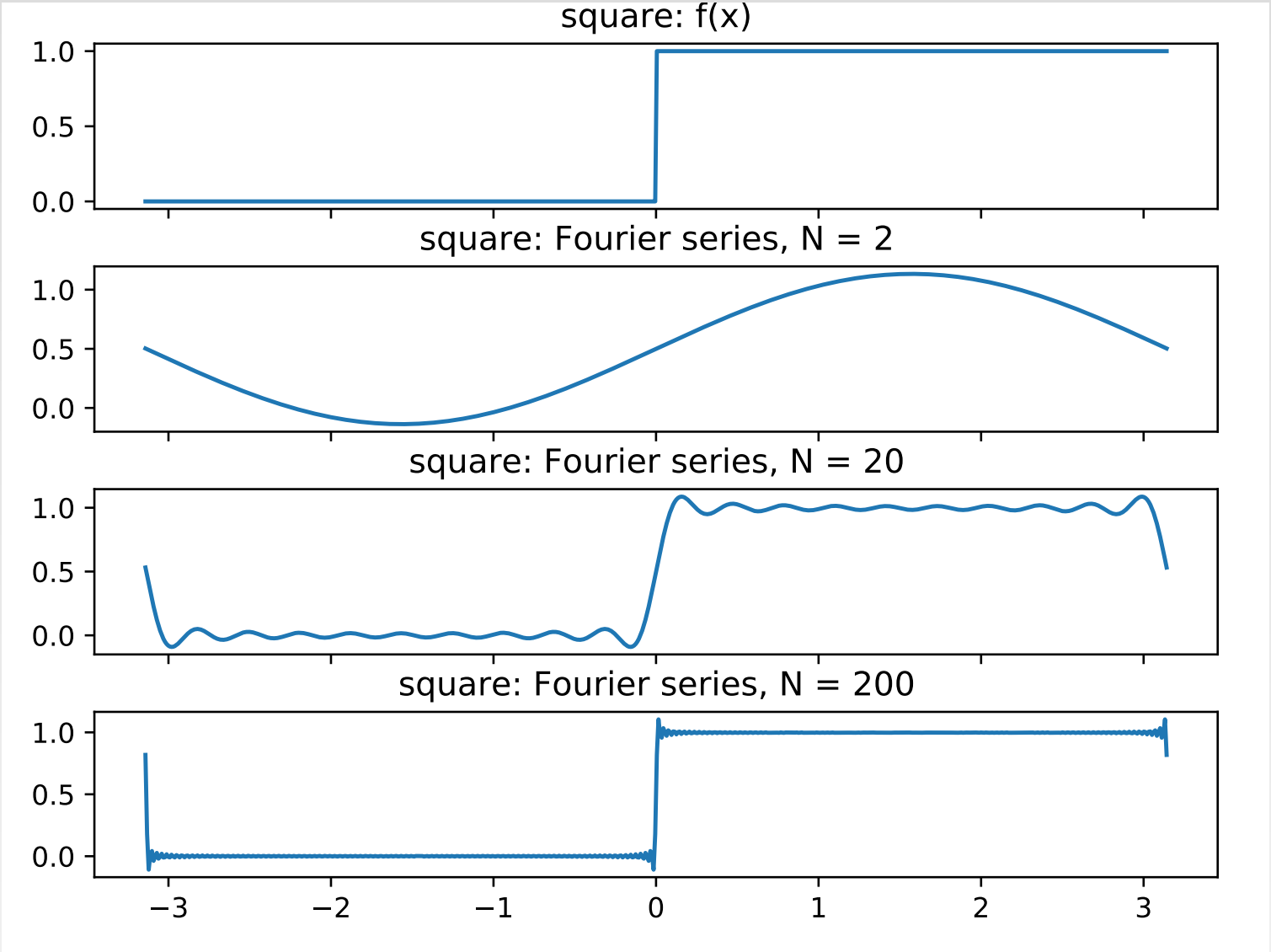
Lab 4: Fourier series — example #1: rectifier



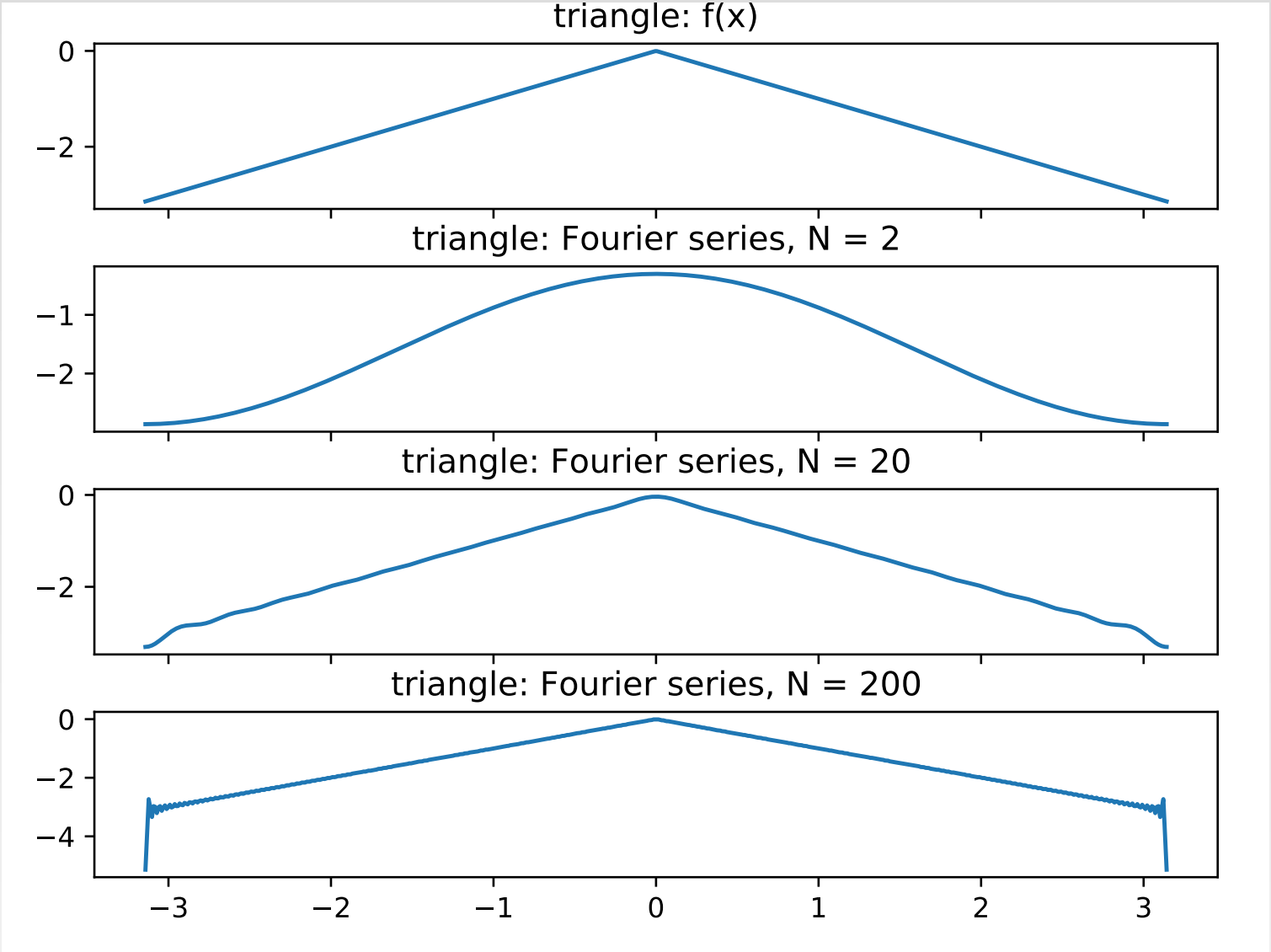
Lab 4: Fourier series — example #2: sawtooth



Lab 4: Fourier series — example #3: square



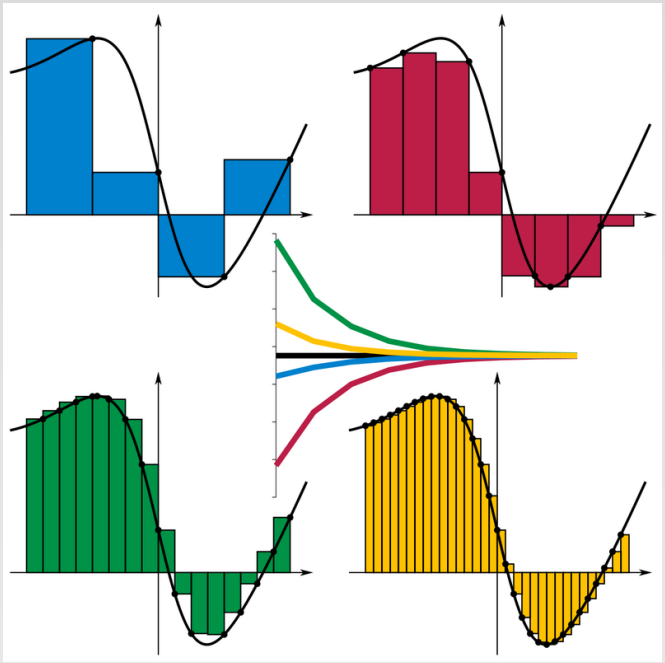
Lab 4: Fourier series — example #4: triangle



Lab 4: Fourier series — Gibbs phenomenon

- Consider the $N = 2, 20, 200$ plots for each of the four periodic functions (rectifier, sawtooth, square, triangle)
- Surprising phenomenon: for each of the four functions, increasing N has two contrasting effects:
  - increases the accuracy of the Fourier series approximation for all values x in $(-\pi, +\pi)$, but
  - decreases the accuracy of the Fourier series approximation for points x at or near $\pm\pi$
  - the decreas in accuracy is also present in the square wave at its point of discontinuity at $x = 0$
- The series approximation errors at the points of discontinuity is known as the Gibbs phenomenon, named after Josiah Gibbs.

Lab 4: Riemann integration



https://en.wikipedia.org/wiki/Riemann_integral

In calculus, the Riemann sum is an approximation of an integral as a sum, dividing up the domain of x into m small intervals, and adding up the areas of the corresponding rectangles, with widths $\mathrm{d}x$ and height $f(x_k)$, for $k \in [m]$:

$$\int_x f(x)\mathrm{d}x \approx \sum_{k=1}^{m} f(x_k)\mathrm{d}x$$

Outline

Lab 4 starter code: functions

1. `fsc(n, g, f, x, dx)`: Fourier series coefficient n for f using basis function g
2. `fsc_all(N_max, f, x, dx)`: compute Fourier series coefficients: a_0, a, b
3. `fs(x, a0, a, b, N)`: Fourier series representation at x using N terms
4. `plot_fsr(f, N1, N2, N3, x, dx, name, filename)`: 4 subplots: $f(x)$ and Fourier series using $N1, N2, N3$ terms
5. `f_NAME(x)`: various "simple" functions $f(x)$, taken as periodic over $[-\pi, +\pi]$, including `square`, `triangle`, `sawtooth`, `rectifier`

Lab 4 starter code: main function

```
# main function
if __name__ == "__main__":
    # specify dx: the distance between x-values over [-pi,+pi]
    dx = 1/100
    # number of points in [-pi, +pi] using width dx
    m = int(2 * np.pi / dx)
    # list of m x-axis values in [-pi, +pi]
    x = np.linspace(-np.pi, np.pi, m)

    # specify the three values of N (# terms in series)
    N1, N2, N3 = 2, 20, 200
    # call plot_fsr with four different signals defined above
    plot_fsr(f_square, N1, N2, N3, x, dx, 'square','Lab4-FSR-
                                        Square.pdf')
    plot_fsr(f_triangle, N1, N2, N3, x, dx, 'triangle', 'Lab4-
                                        FSR-Triangle.pdf')
    plot_fsr(f_sawtooth, N1, N2, N3, x, dx, 'sawtooth', 'Lab4-
                                        FSR-Sawtooth.pdf')
    plot_fsr(f_rectifier, N1, N2, N3, x, dx, 'rectifier', 'Lab4
                                        -FSR-Rectifier.pdf')
```

Lab 4 starter code: first function to be completed: `fsc`

```
# Fourier series coefficient n for f using basis function g
def fsc(n, g, f, x, dx):
    # 1. integral: (1/pi) f(xe) g(n xe) dx over all xe in x
    pass
```

- `fsc` computes the Fourier series coefficient (fsc) with index n, using basis function g for periodic function f using points x from interval $[-\pi, +\pi]$ with interval width dx
- Return the Riemann approximation of the coefficient:

$$c_n = \frac{1}{\pi}\int_{-\pi}^{+\pi} f(x)g(nx)\mathrm{d}x \approx \frac{1}{\pi}\sum_{k=1}^{m} f(x_k)g(nx_k)\mathrm{d}x$$

- Note, if basis function g is cos (sin) this is coefficient a_n (b_n)
- Recall `x` is a `numpy` array of length $m = \lfloor 2\pi/dx \rfloor$

Lab 4 starter code: second function to be completed: `fs`

```
# Fourier series representation at x using N terms
def fs(x, a0, a, b, N):
    # 1. sum the a_n coefficients from 1 to N
    pass
    # 2. sum the b_n coefficients from 1 to N
    pass
    # 3. return the Fourier series approximation
    pass
```

- `fs` takes arguments x (value at which f is to be approximated), a_0 (DC term), a (coefficients a_n), b (coefficients b_n), and N (# of terms in series)
- `fs` should be vectorized: return vector if x is in fact a vector, one for each scalar value in x
- First, compute the sum of $a_n \cos(nx)$ over $n \in [N]$
- Second, compute the sum of $b_n \sin(nx)$ over $n \in [N]$
- Third, return $a_0/2$ plus the two sums above
- We will explain on subsequent slides how these sums may be computed using the `numpy` command `np.sum` with the `axis` argument

Lab 4 starter code: second function to be completed: `fs`

- Recall that the goal of function `fs` is to compute the Fourier series approximation $\hat{f}(x_e)^{(N)}$ to $f(x_e)$ at each value, say x_e, in the length-m `numpy` array `x`:

$$f(x_e) \approx \hat{f}^{(N)}(x_e) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(nx_e) + \sum_{n=1}^{N} b_n \sin(nx_e).$$

- So, `fs` should return a length-m list: $[\hat{f}^{(N)}(x_1), \ldots, \hat{f}^{(N)}(x_m)]$, where $m = \lfloor 2\pi/\mathrm{d}x \rfloor$.
- Consider the list version of the first sum (one list element for each value x_e)

$$\left[\sum_{n=1}^{N} a_n \cos(nx_1), \ldots, \sum_{n=1}^{N} a_n \cos(nx_m) \right]$$

computed, using vectorization, by summing the Python expression: `[[a[n-1] * np.cos(n * x) for n in range(1,N+1)]]`

Lab 4 starter code: second function to be completed: `fs`

- The expression to be summed:
  `[[a[n-1] * np.cos(n * x) for n in range(1,N+1)]]` is a $N \times m$ two-dimensional array
- The rows of this array are indexed by n over $1, \ldots, N$, while the columns of this array are indexed by x_e over $x_1, \ldots, x_m$
- What is needed is to sum this array over all rows in each column, i.e., sum from $n = 1$ to $n = N$.
- This goal can be achieved by using the `axis=0` option in the `numpy` command `np.sum()`, as illustrated on the next slide

Lab 4 starter code: the axis option of np.sum

```
import numpy as np
x = np.array(range(3))
y = [x * n for n in range(4)]
print("y = ")
for ye in y: print(ye)
# use numpy sum on y
print("np.sum(y) = {}".format(np.sum(y)))
# sum columns
print("np.sum(y,axis=0) = {}".format(np.sum(y,axis=0)))
# sum rows
print("np.sum(y,axis=1) = {}".format(np.sum(y,axis=1)))
```

```
y =
[0 0 0]
[0 1 2]
[0 2 4]
[0 3 6]
np.sum(y) = 18
np.sum(y,axis=0) = [ 0  6 12]
np.sum(y,axis=1) = [0 3 6 9]
```

Outline

Lab 3 mechanics

- Form groups of 2 or 3 students
- TA will move from group to group to offer help and check on progress
- You earn 100 points for this laboratory if either:
  - You demonstrate your code works to the TA (and are then free to leave), OR
  - You continue working on the assignment until the end of the lab and the TA checks off your name and gives you partial credit based on the work completed during the lab; you may submit the completed assignment by end of day via BBLearn to earn full credit
- You will NOT earn points if you leave without checking in with TA