Introduction to Computational Quantum Mechanics: Application-based Learning with Python

Braden M. Weight
Department of Physics and Astronomy
University of Rochester

May 8, 2023

Contents

Chapter 1

Python and Environments

No assignment here.

Chapter 2

Types of Variables in Python

1. Write a program that to evaluate e^x with $x = -5.5$ using a Taylor series,

$$e^x = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \mathcal{O}(x^4), \tag{2.1}$$

for orders 0, 1, 2, 3, 4, 5, 10, and 100. Note that $e^x \approx 1 + x$ is the first-order expansion. How quickly does the result converge to the exact result ? At what order, can you achieve an accuracy of 1% or better (% Error = $(x_{\text{Approx}} - x_{\text{exact}})/x_{\text{exact}})$) ? Make a plot of the result with respect to the expansion order. For plotting, see example below.

2. The $N \times N$ discrete Fourier transform matrix is defined by,

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \gamma & \gamma^2 & \gamma^3 & \cdots & \gamma^{N-1} \\ 1 & \gamma^2 & \gamma^4 & \gamma^6 & \cdots & \gamma^{2(N-1)} \\ 1 & \gamma^3 & \gamma^6 & \gamma^9 & \cdots & \gamma^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \gamma^{N-1} & \gamma^{2(N-1)} & \gamma^{3(N-1)} & \cdots & \gamma^{(N-1)(N-1)} \end{bmatrix}, \tag{2.2}$$

where $\gamma = e^{-2\pi i/N}$.

(a) Use Numpy to construct the 4×4 matrix and save to a file using the 'np.savetxt()' function.

(b) Write a function to accept an integer N and output the corresponding $N \times N$ array for the discrete Fourier transform.

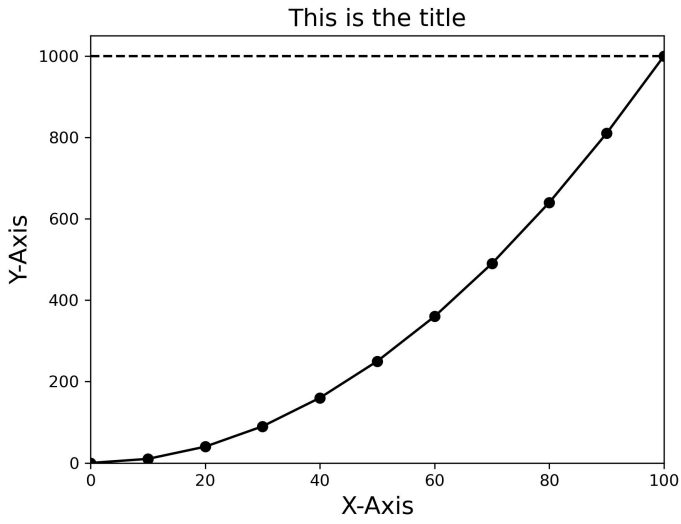

Figure 2.1: Generated from the code “example_plot.py”.

```
# This is an example for plotting a quadratic function
import numpy as np
import matplotlib
matplotlib.use('Agg') # This is required for the NDSU cluster...not sure why
from matplotlib import pyplot as plt

X = np.arange(0,100+10,10)
Y = 0.1 * X**2
plt.plot(X,Y,"-o",c="black",label="Y(x) = $\frac{X^2}{4}$")
plt.plot(X,np.ones(len(X))*Y[-1],"--",c="black",label="Y(x) = $\frac{X^2}{4}$")
plt.xlim(X[0],X[-1])
plt.ylim(0)
plt.xlabel("X-Axis",fontsize=15)
plt.ylabel("Y-Axis",fontsize=15)
plt.title("This is the title",fontsize=15)
plt.savefig("example_plot.jpg", dpi=300)
```

Chapter 3

Numerical Calculus

3.1 Differentiation

1. Using the function,
$$f(x) = x^{\sin(x)}, \quad x \in (1, 10), \tag{3.1}$$
plot the function $f(x)$ and its first derivative $f'(x)$. Use WolframAlpha [https://www.wolframalpha.com] (or Mathematica) to find the analytic result. The syntax is "D[f(x), x]" in both WolframAlpha and Mathematica. Plot the error between the numerical and analytical results for the first derivative using the forward difference and central difference formulas.

2. The central difference formula can be extended to two dimensions and can be written as,
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+\Delta x, y) - f(x-\Delta x, y)}{2\Delta x} + \mathcal{O}(\Delta x^2) \tag{3.2}$$
$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x, y+\Delta y) - f(x, y-\Delta y)}{2\Delta y} + \mathcal{O}(\Delta y^2) \tag{3.3}$$
$$\frac{\partial f(x,y)}{\partial x^2} \approx \frac{f(x+\Delta x, y) - 2f(x,y) + f(x-\Delta x, y)}{\Delta x^2} + \mathcal{O}(\Delta x^2) \tag{3.4}$$
$$\frac{\partial f(x,y)}{\partial y^2} \approx \frac{f(x, y+\Delta y) - 2f(x,y) + f(x, y-\Delta y)}{\Delta y^2} + \mathcal{O}(\Delta y^2) \tag{3.5}$$
$$\left.\frac{\partial f(x,y)}{\partial x \partial y}\right|_{jk} \approx \frac{f_{j+1,k+1} - f_{j+1,k} - f_{j,k+1} + 2f_{j,k} - f_{j-1,k} - f_{j,k-1} - f_{j-1,k-1}}{2\Delta x \Delta y} + \mathcal{O}(\Delta x^2 \Delta y^2) \tag{3.6}$$
for the first and second derivatives. Note that I switched from $x + \Delta x$ notation to $j + 1$ notation to allow the final equation to sit on one line. Make sure you convince yourself that they mean the same thing. In fact, the final notation is more common when discussing numerical evaluation of functions and their derivatives.

Consider the following function of two variables x and y,

$$f(x, y) = x^2 + y^2 \tag{3.7}$$

Plot the following set of 1D functions calculated numerically alongside the analytic result.

$$g_1(x) = f(x, y = 1), \tag{3.8}$$
$$g_2(y) = f(x = 1, y), \tag{3.9}$$
$$g_3(x) = \frac{\partial f(x, y)}{\partial x}|_{y=1}, \tag{3.10}$$
$$g_4(y) = \frac{\partial f(x, y)}{\partial y}|_{x=1}, \tag{3.11}$$
$$g_5(x) = \frac{\partial f(x, y)}{\partial x \partial y}|_{y=1} \tag{3.12}$$

Note: the solution to this problem will be five, separate plots, each with two curves (numerical and analytic results).

3.2 Root-finding Algorithms

1. Use the Newton-Raphson Secant method to find the non-trivial roots (*i.e.*, non-zero roots) of
$$f(x) = x^2 - 150x + 5000. \tag{3.13}$$
Plot the convergence ($|f(x_n) - f(x_{\rm EXACT})|$) as a function of the number of iteration steps n for each of the two roots.

3.3 Numerical Integration

1. In quantum mechanics, particles, such as the electron, are described by wavefunctions $\psi(x)$ whose square magnitudes $|\psi(x)|^2$ are interpreted as probability distributions $\mathcal{P}(x) = |\psi(x)|^2$. There are two main properties of all probability distributions.

   - $\mathcal{P}(x)$ must be normalized,
   $$1 = \int_{-\infty}^{\infty} dx\ \mathcal{P}(x), \tag{3.14}$$
   - $\mathcal{P}(x)$ must be real- and positive-valued.

   The square magnitude of a wavefunction (which is in general a complex function) is obtained by,
   $$\mathcal{P}(x) = \psi^*(x)\psi(x) = |\psi(x)|^2, \tag{3.15}$$
   where $\psi^*(x)$ indicates the complex conjugate of $\psi(x)$. If $\psi(x)$ is purely real-valued, then,
   $$\psi^*(x) = \psi(x). \tag{3.16}$$

   Consider the following non-normalized wavefunctions,

   (a)
   $$\psi(x) = \mathcal{N}\sin(\pi x), \qquad x =\in (0, 1) \tag{3.17}$$

   (b)
   $$\psi(x) = \mathcal{N}\mathrm{e}^{-\frac{x^2}{2}}, \qquad x \in (-\infty, \infty) \tag{3.18}$$

   (c)
   $$\psi(x) = \mathcal{N}x, \qquad x \in (0, 10). \tag{3.19}$$

   For each of the three wavefunctions, **by hand** (pen and paper), find the exact normalization factor $\mathcal{N}$ for $\psi(x)$ that satisfies,
   $$1 = \int_a^b dx\ \mathcal{P}(x) = \mathcal{N}^2 \int_a^b dx\ |\psi(x)|^2 \tag{3.20}$$
   and compare to Riemann and/or trapezoidal numerical integration techniques for 5, 10, 20, and 50 grid points (*i.e.*, the number of rectangles/trapezoids).

   Note: Feel free to use Mathematica or Wolfram Alpha (https://www.wolframalpha.com/) to perform analytic integrations. The syntax is something like "Integrate[f(x), a, b]".

2. In quantum mechanics, wavefunctions $|\psi\rangle$ are the shape of mathematical vectors, which have a single dimension of some (usually infinite) length. In the position representation $\psi(x) \equiv \langle x|\psi\rangle$, there are infinite values of x you could have, so the vector of $|\psi\rangle$ is infinite in that sense.

   In the previous problem, we took these wavefunctions $\psi(x)$ and enforced normalization by making use Eq. 3.20. This is a statement of the conservation of probability, but

we can alternatively show that two wavefunctions that are solutions to some Hamiltonian (*e.g.*, the quantum harmonic oscillator) are orthogonal to one another such that $\langle\psi_n|\psi_m\rangle = \delta_{nm}$, where δ_{nm} is the Kronecker-delta function defined as $\delta_{nm} = 0$ for $n \neq m$ and $\delta_{nm} = 1$ for $n = m$. In the same representation as the previous problem (*i.e.*, position representation),

$$\delta_{nm} = \langle\psi_n|\psi_m\rangle = \int_a^b dx \langle\psi_n|x\rangle\langle x|\psi_m\rangle = \int_a^b dx\ \psi_n^*(x)\psi_m(x). \tag{3.21}$$

For the solutions to the quantum harmonic oscillator,

$$\psi_n(x) \sim \mathrm{e}^{-\frac{\gamma x^2}{2}} H_n(\sqrt{\gamma}x), \qquad n = 0, 1, 2, ... \tag{3.22}$$

where $\gamma = \frac{m\omega}{\hbar}$ (which we can just call $\gamma = 1$ for our purposes), H_n is the n^{th} Hermite polynomial (which can be obtained in python using scipy.special.hermite, see documentation at SciPy Hermite Documentation).

Using the definition provided above, normalize the lowest two wavefunctions (n = 0 and n = 1), and verify that they are now orthogonal to one another by constructing the so-called overlap matrix defined as,

$$S_{nm} = \int dx\ \psi_n^*(x)\psi_m(x) = \delta_{nm} \doteq \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.23}$$

and printing to the screen. Generalize it so that one can produce an arbitrarily sized overlap matrix $N \times N$. You will need a for-loop to loop over n, m and a numpy array to store the $N \times N$ overlap matrix.

3. We can generalize numerical integration to two dimensions $f(x) \to f(x, y)$ whose total integration can be written as,

$$I = \int_{x_0}^{x_1} dx \int_{y_0}^{y_1} dy\ f(x, y) \approx \sum_{x_i}^{N_x} \Delta x \sum_{y_i}^{N_y} \Delta y\ f(x_i, y_i). \quad \text{(Riemann Sum)} \tag{3.24}$$

Here, the approximation is the Riemann sum approximation. In one dimension, these were rectangles of width Δx and height $f(x)$, and in two dimensions these are rectangular prisms of width Δx, depth Δy, and height $f(x, y)$.

Integrate the following function:

$$f(x, y) = \sin^2(x^3y^2), \qquad x \in (-0.5, 1), \quad y \in (-1, 1). \tag{3.25}$$

Choose $\Delta x = \Delta y = 0.005$. The exact solution (up to 7 digits) is $I = 0.151586$.