ENGR 285 — Homework 3

Vincent Edwards

March 24, 2025

Euler Method

$$\frac{dy}{dx} = f(x, y)$$

$$y(x_{i+1}) \approx y(x_i) + h \cdot f(x_i, y_i)$$

euler_method.py

```python
import numpy as np

def calculate(x_0, x_f, y_0, h, diff):
    delta_x = x_f - x_0
    N = np.round(delta_x / h).astype(int) + 1

    x, h = np.linspace(x_0, x_f, N, retstep=True)
    y = np.array([y_0 for _ in range(N)], dtype=float)

    for i in range(N - 1):
        y[i + 1] = y[i] + h * diff(x[i], y[i])

    return x, y

def relative_error(x, x_true):
    return np.abs(x - x_true) / x_true

def maximize_h_within_error(x_0, x_f, y_0, diff, y_true, error_threshold,
h_min=1e-9):
    N = 1
    while (h := (x_f - x_0) / N) > h_min:
        x, y = calculate(x_0, x_f, y_0, h, diff)
        y_estimate = y[-1]
        error = relative_error(y_estimate, y_true)
        if error < error_threshold:
            return h, y_estimate, error
        N += 1
    return None, None, None
```

Problem 1

Write a program that utilizes a while-loop to return the square root of an arbitrary positive number to the nearest whole number; i.e. find the largest integer who's square is less than the given number, and compare to the square of the next integer. Do not use any non-elementary math operations (i.e. the only math operations you can use are addition, subtraction, multiplication, and division). Test your code with several numbers of various sizes to check that it is working properly. Write out or copy/paste the code.

```
def int_sqrt(x):
    """
    Return the square root of an arbitrary positive number to the nearest whole
number
    """
    n = 1
    while n * n < x:
        n += 1
    previous = n - 1
    previous_squared = previous * previous
    if (n * n) - x <= x - previous_squared:
        return n
    else:
        return n - 1

test_input = [2, 3, 6, 7, 12, 13, 20, 21, 30, 31, 42]
print("Input | Output")
print("----- | ------")
for x in test_input:
    print(f"{x:5} | {int_sqrt(x):6}")
```

Output:

```
Input | Output
----- | ------
    2 |      1
    3 |      2
    6 |      2
    7 |      3
   12 |      3
   13 |      4
   20 |      4
   21 |      5
   30 |      5
   31 |      6
   42 |      6
```

Problem 2

One of the simplest ways to numerically calculate definite integrals is with Monte Carlo integration. Write a program that will approximately integrate the function $f(x) = e^{-\frac{1}{2}x^2}$ from $0 < x < 1$ by choosing 100 points (x, y) in the square $0 < x < 1$ and $0 < y < 1$ with random coordinates, counting how many of them lie under the curve, and dividing by the total number of points. Write out or copy/paste the code. Run your program 6 times and record the outcomes.

```
import numpy as np
rng = np.random.default_rng(seed=285)

def f(x):
    return np.exp(-x**2 / 2)

N = 100
count = 0
trials = 6
for i in range(trials):
    x_values = rng.random(N)
    y_values = rng.random(N)
    below = (y_values < f(x_values)).sum()
    integral = below / N
    print(f"Trial {i + 1}: {integral}")
```

Output:

```
Trial 1: 0.79
Trial 2: 0.89
Trial 3: 0.79
Trial 4: 0.84
Trial 5: 0.86
Trial 6: 0.83
```

Problem 3

For each of the following initial value problems, solve using the Euler Method and either sketch or copy/paste a graph of the result over the given interval. Use a step size small enough that continuing to lower the step size further yields no noticeable change in the appearance of the graph.

Part a

$y' = (x - y)^2$ subject to $y(0) = 0.5$ for $0 < x < 2$

```
import euler_method
import matplotlib.pyplot as plt

x, y = euler_method.calculate(0, 2, 0.5, 0.01, lambda x, y: (x - y)**2)
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set(xlabel="$x$", ylabel="$y$")
fig.savefig("media/03_a.svg")
```

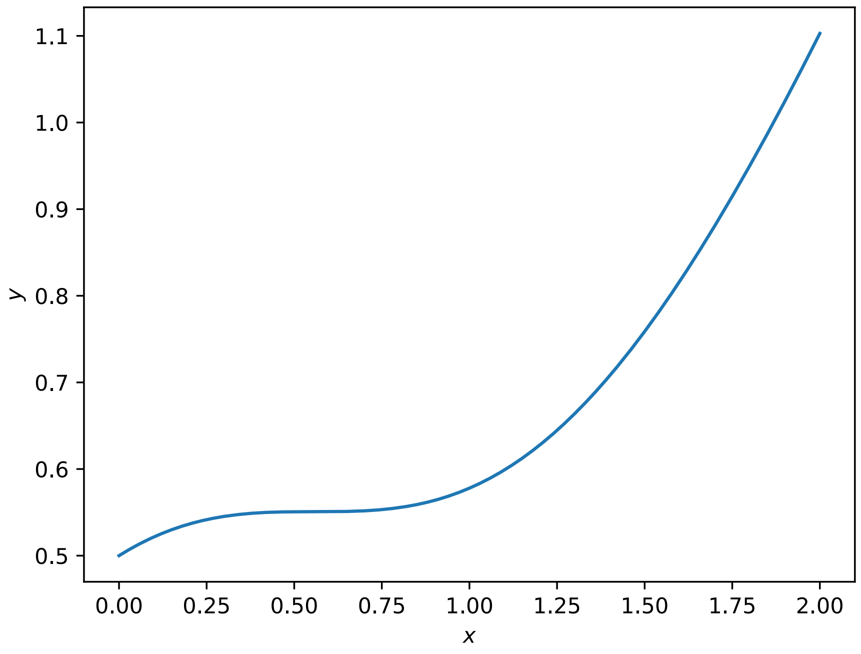

Figure 1: Graph of the solution to $y' = (x - y)^2$ subject to $y(0) = 0.5$

Part b

$y' = \sin(x + y^2)$ subject to $y(-\pi) = 0.1$ for $-\pi < x < 2\pi$

```
import euler_method
import numpy as np
import matplotlib.pyplot as plt

x, y = euler_method.calculate(-np.pi, 2*np.pi, 0.1, 0.01, lambda x, y: np.sin(x +
y**2))
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set(xlabel="$x$", ylabel="$y$")
fig.savefig("media/03_b.svg")
```

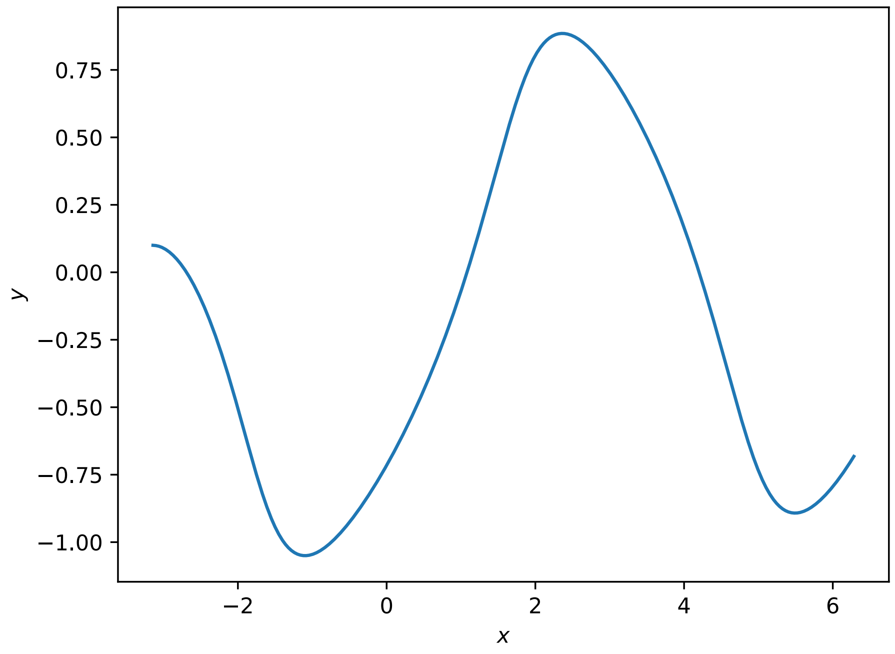

Figure 2: Graph of the solution to $y' = \sin(x + y^2)$ subject to $y(-\pi) = 0.1$

Problem 4

For each of the following initial value problems, first solve for $y(x)$ by separating the equation. Then use the Euler Method to determine an estimate for the maximum step size such that $y(4)$ is accurate to less than 1%. (i.e. run an Euler Method program repeatedly, decreasing step size each time, until the calculated value of $y(4)$ is less than 1% different from the exact answer)

Part a

$2y' = \tan y$ subject to $y(0) = 0.05$

$$\begin{aligned} 2y' &= \tan(y) \\ \frac{dy}{dx} &= \frac{1}{2}\tan(y) \\ \frac{dy}{\tan(y)} &= \frac{1}{2}dx \\ \int \cot(y)dy &= \int \frac{1}{2}dx \\ \ln(\sin(y)) &= \frac{1}{2}x + C \\ \sin(y) &= Ce^{\frac{1}{2}x} \\ y &= \sin^{-1}\left(Ce^{\frac{1}{2}x}\right) \end{aligned}$$

$\underline{y(0) = 0.05}$

$$\begin{aligned} 0.05 &= \sin^{-1}\left(Ce^{\frac{1}{2}(0)}\right) \\ C &= \sin(0.05) \end{aligned}$$

$$y = \sin^{-1}\left(\sin(0.05)e^{\frac{1}{2}x}\right)$$

$$y(4) = \sin^{-1}\left(\sin(0.05)e^{\frac{1}{2}(4)}\right)$$

$$y(4) = \sin^{-1}(\sin(0.05)e^{2}) \approx 0.3783$$

```python
import euler_method
import numpy as np

def f(x, y):
    return np.tan(y) / 2

x_0 = 0
x_f = 4
y_0 = 0.05

y_true = np.asin(np.sin(0.05) * np.exp(2))
threshold = 0.01

h, y_estimate, relative_error = euler_method.maximize_h_within_error(x_0, x_f, y_0,
f, y_true, threshold)

if h is None:
    print("Estimates do not get close enough to the true value as h decreases")
else:
    print(f"With h = {h:.3}, y(4) is estimated to be {y_estimate:.4}
({relative_error:.4%} from the true value)")
```

Output:

```
With h = 0.0186, y(4) is estimated to be 0.3745 (0.9998% from the true value)
```

Part b