INTRO TO ODES

BMES 678: Programming Assignment

```python
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import numpy as np

from typing import Callable
```

Phase Diagram

Consider the following Lotka-Volterra predator-prey model:

- $x'(t) = Ax(t) - Bx(t)y(t)$
- $y'(t) = Cx(t)y(t) - Dy(t)$

Let $A = D = 2, B = C = 1.8$

Draw the phase diagram. You do not need to show any integral curves.

What are the predator/prey equilibrium populations (i.e., population sizes do not change over time)?

```python
# constants
A = D = 2
B = C = 1.8

# meshgrid
x = np.arange(-3, 3, 0.2)
y = np.arange(-3, 3, 0.2)
X, Y = np.meshgrid(x, y)

# derivatives
dx = A*X - B*X*Y
dy = C*X*Y - D*Y

# normalize
norm = np.sqrt(dx**2 + dy**2)
dx /= norm
dy /= norm

# define the system of equations
def lotka_volterra_equilibrium(z):
    x, y = z
    dxdt = A*x - B*x*y
    dydt = C*x*y - D*y
    return [dxdt, dydt]

# initial guess for the equilibrium point
z_guess = [1.5, 1.0]
```

```
# solve for the equilibrium point
equilibrium = fsolve(lotka_volterra_equilibrium, z_guess)

# print equilibrium point
print(f"Equilibrium: x = {equilibrium[0]:.2f}, y = {equilibrium[1]:.2f}")

# plot
fig, ax = plt.subplots(figsize=(5, 5))
ax.quiver(X, Y, dx, dy, color="blue", alpha=0.5, label="Phase Diagram")
ax.plot(equilibrium[0], equilibrium[1], 'ro', label='Equilibrium')
ax.spines[["top", "right"]].set_visible(False)
ax.set_xlabel("Species x", fontsize=10)
ax.set_ylabel("Species y", fontsize=10)
ax.legend(loc="upper center", bbox_to_anchor=(0.5, -0.1), ncol=2, frameon=False)
ax.set_title("Phase-Diagram for the Lotka-Volterra Predator-Prey Model", fontsize=11);
```

```
Equilibrium: x = 1.11, y = 1.11
```

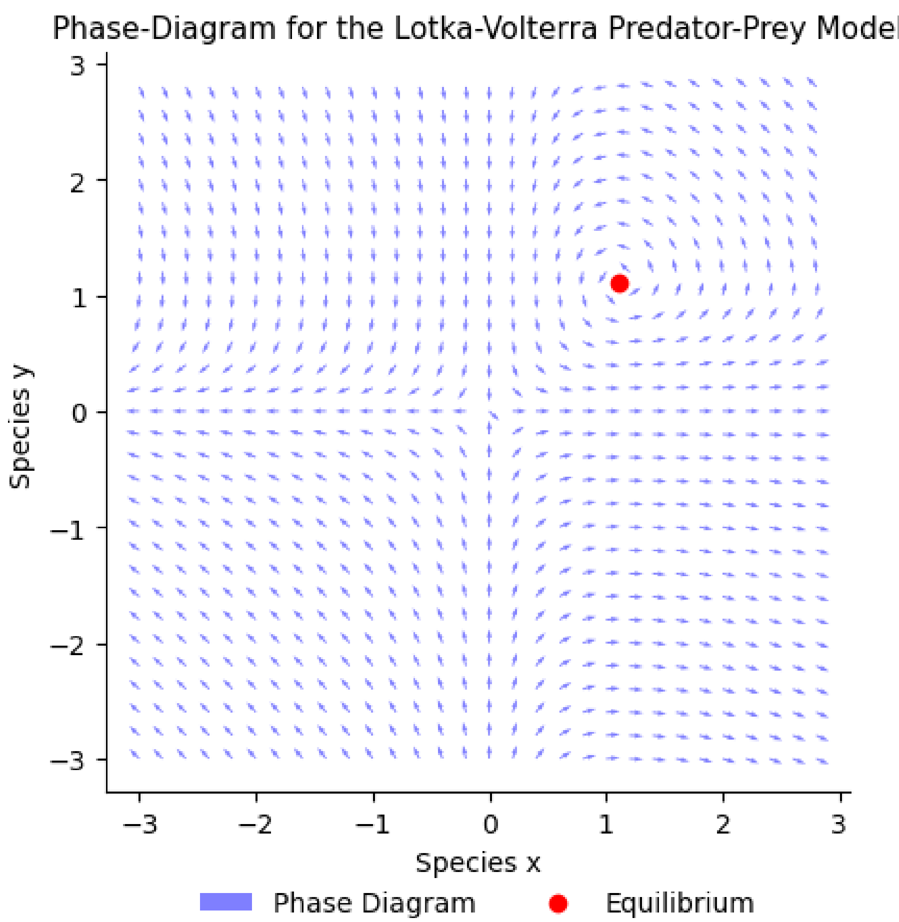

Figure 1: Phase diagram for the Lotka-Volterra predator-prey model

Euler's Method

Write a function *[X,Y]=eulermethod(fprime, timespan, y0, h=0.1)* where

- `fprime`: function handle representing derivative of f for a given value of x.
- `timespan=[x0 xend]`: starting and ending values of x
- `y0`: f(x0), value of function at x=x0.
- `h`: step size (default h=0.1)

Your function should calculate using Euler's method and store in Y, the successive values of f(x) for X=x0..xend. It is okay if the last entry in X does not reach exactly xend.

Use the eulermethod() function you wrote to approximate the value of sin(x), for x=0..10. Compare the values of sin(x) and the approximated values by showing them on the same plot. What is the average absolute error in your approximated values (average for entire x=0..10, not just for x=10)?

```python
def eulermethod(
    fprime: Callable,
    timespan: tuple[float, float],
    y0: float,
    h: float = 0.1,
) -> tuple[list[float], list[float]]:
    """Euler's method for solving ODEs"""

    x0, xend = timespan
    X = [x0 + i*h for i in range(int(np.ceil((xend - x0) / h)))]
    if X[-1] < xend: # make sure the last entry in X reaches exactly xend
        X[-1] = xend
    Y = [y0]
    for i in range(1, len(X)):
        Y.append(Y[-1] + h * fprime(X[i-1], Y[-1]))

    return X, Y


# use eulermethod to approximate the value of sin(x), for x=0..10.
# d(sin(x))/dx = cos(x)
X, Y_hat = eulermethod(lambda x, y: np.cos(x), (0, 10), 0)

# compute actual values
Y = np.sin(X)

# MAE
mae = np.mean(np.abs(Y_hat - Y))
print(f"MAE: {mae:.4f}")
```

```
MAE: 0.0530
```

```python
fig, ax = plt.subplots(figsize=(6,3))

ax.plot(X, Y_hat, label="Euler[sin(x)]", color="blue")
ax.plot(X, Y, label="sin(x)", color="red")
ax.fill_between(X, Y_hat, Y, color="red", alpha=0.2, label="Error")
ax.minorticks_on()
ax.legend(fancybox=False, framealpha=1)
ax.spines[["top", "right"]].set_visible(False)
ax.grid(which="major", color="gray", linewidth=0.5)
ax.grid(which="minor", color="gray", linewidth=0.2, linestyle="--")
ax.set_xlabel("x", fontsize=11)
ax.set_ylabel("sin(x)", fontsize=11)
ax.set_title("Euler's Method for Approximating sin(x)", fontsize=12);
```

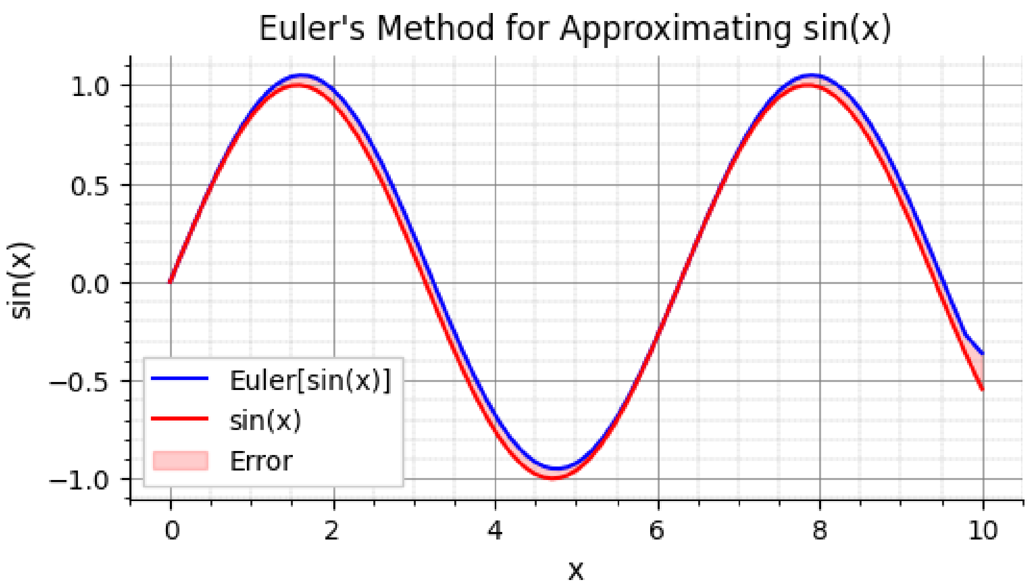

Figure 2: Euler's method for approximating $sin(x)$