Assignment 3: Numerical solution of an ordinary differential equation

Valentina Watanabe
21367661

October 2024

1 Direction plot

```
    #!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 26 12:10:13 2024

@author: vwitch
"""

import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0,5,25)
x = np.linspace(-3,3,25)
X,T = np.meshgrid(x,t)

def f(x, t):
    f = (1 + t) * x + 1 - 3 * t + t**2
    return f

dx_dt = f(X, T)
dt = np.ones_like(T)

#Normalize
magnitude = np.sqrt(dt**2 + dx_dt**2)
dt_normalized = dt / magnitude
dx_dt_normalized = dx_dt / magnitude

#Plot
fig, ax = plt.subplots(figsize=(8, 6))
ax.quiver(T, X, dt_normalized, dx_dt_normalized, angles='xy')
ax.set_xlabel('t')
ax.set_ylabel('x')
ax.set_title('Direction Field for dx/dt = (1 + t)x + 1 - 3t + t^2')
ax.set_xlim(0, 5)
ax.set_ylim(-3, 3)
plt.grid()
plt.show()
```

2 Euler's Method

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 26 14:26:26 2024

@author: vwitch
"""

import numpy as np
import matplotlib.pyplot as plt

def f(x, t):
    return (1 + t) * x + 1 - 3 * t + t ** 2

t_values = np.linspace(0, 5, 25)
x_values = np.linspace(0, 100000, 25)
T, X = np.meshgrid(t_values, x_values)
dXdt = f(X, T)

U = np.ones_like(T)
V = dXdt

fig, ax = plt.subplots(figsize=(10, 6))
ax.quiver(T, X, U, V, angles='xy', scale_units='xy', scale=50, color='b')
ax.set_xlim(0, 5)
ax.set_ylim(0, 100000)
ax.set_xlabel("t")
ax.set_ylabel("x")
ax.set_title("Direction Field for dx/dt = (1 + t)x + 1 - 3t + t^2")


t_start, t_end, h = 0, 5, 0.04
num_steps = int((t_end - t_start) / h)
t_points = np.linspace(t_start, t_end, num_steps + 1)
```

```
x_points = np.zeros(num_steps + 1)
x_points[0] = 0.0655

for i in range(num_steps):
    x_points[i + 1] = x_points[i] + h * f(x_points[i], t_points[i])

ax.plot(t_points, x_points, 'r-', label="Euler Method Solution")
ax.legend()
plt.show()
```

3 Improved Euler's

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 28 18:06:36 2024

@author: vwitch
"""


import numpy as np
import matplotlib.pyplot as plt

def f(x, t):
    return (1 + t) * x + 1 - 3 * t + t ** 2

t_values = np.linspace(0, 5, 25)
x_values = np.linspace(0, 1500, 25)
T, X = np.meshgrid(t_values, x_values)
dXdt = f(X, T)

U = np.ones_like(T)
V = dXdt

fig, ax = plt.subplots(figsize=(10, 6))
ax.quiver(T, X, U, V, angles='xy', scale_units='xy', scale=50, color='b')
ax.set_xlim(0, 5)
ax.set_ylim(0, 1500)
ax.set_xlabel("t")
ax.set_ylabel("x")
ax.set_title("Direction Field for dx/dt = (1 + t)x + 1 - 3t + t^2")
```

```
t_start, t_end, h = 0, 5, 0.04
num_steps = int((t_end - t_start) / h)
t_points = np.linspace(t_start, t_end, num_steps + 1)
x_points = np.zeros(num_steps + 1)
x_points[0] = 0.0655


for i in range(num_steps):
    k1 = f(x_points[i], t_points[i])
    x_temp = x_points[i] + h * k1
    k2 = f(x_temp, t_points[i] + h)
    x_points[i + 1] = x_points[i] + (h / 2) * (k1 + k2)


ax.plot(t_points, x_points, 'r-', label="Improved Euler Method Solution")
ax.legend()
plt.show()
```

4 Runge-Kutta

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 28 18:21:37 2024

@author: vwitch
"""

import numpy as np
import matplotlib.pyplot as plt

def f(x, t):
    return (1 + t) * x + 1 - 3 * t + t ** 2

t_values = np.linspace(0, 5, 25)
x_values = np.linspace(0, -17500, 25)
T, X = np.meshgrid(t_values, x_values)
dXdt = f(X, T)

U = np.ones_like(T)
V = dXdt

fig, ax = plt.subplots(figsize=(10, 6))
ax.quiver(T, X, U, V, angles='xy', scale_units='xy', scale=50, color='b')
ax.set_xlim(0, 5)
ax.set_ylim(-17500,10)
ax.set_xlabel("t")
ax.set_ylabel("x")
ax.set_title("Direction Field for dx/dt = (1 + t)x + 1 - 3t + t^2")


t_start, t_end, h = 0, 5, 0.04
num_steps = int((t_end - t_start) / h)
t_points = np.linspace(t_start, t_end, num_steps + 1)
x_points = np.zeros(num_steps + 1)
x_points[0] = 0.0655


for i in range(num_steps):
    k1 = f(x_points[i], t_points[i])
    k2 = f(x_points[i] + h/2 * k1, t_points[i] + h/2)
    k3 = f(x_points[i] + h/2 * k2, t_points[i] + h/2)
    k4 = f(x_points[i] + h * k3, t_points[i] + h)
    x_points[i + 1] = x_points[i] + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)


ax.plot(t_points, x_points, 'r-', label="Runge-Kutta (4th Order) Solution")
ax.legend()
plt.show()
```

5 Comparison with smaller step size

```
    #!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 26 14:26:26 2024

@author: vwitch
"""

import numpy as np
import matplotlib.pyplot as plt

def f(x, t):
    return (1 + t) * x + 1 - 3 * t + t ** 2

t_values = np.linspace(0, 5, 25)
x_values = np.linspace(-25000, 25000, 25)
```

```
T, X = np.meshgrid(t_values, x_values)
dXdt = f(X, T)

U = np.ones_like(T)
V = dXdt

fig, ax = plt.subplots(figsize=(10, 6))
ax.quiver(T, X, U, V, angles='xy', scale_units='xy', scale=50, color='b')
ax.set_xlim(0, 5)
ax.set_ylim(-25000,20000)
ax.set_xlabel("t")
ax.set_ylabel("x")
ax.set_title("Numerical solutions for dx/dt = (1 + t)x + 1 - 3t + t^2")

t_start, t_end, h = 0, 5, 0.02
num_steps = int((t_end - t_start) / h)
t_points = np.linspace(t_start, t_end, num_steps + 1)
x_points = np.zeros(num_steps + 1)
x_points[0] = 0.0655

t_points1 = np.linspace(t_start, t_end, num_steps + 1)
x_points1 = np.zeros(num_steps + 1)

t_points2 = np.linspace(t_start, t_end, num_steps + 1)
x_points2 = np.zeros(num_steps + 1)


for i in range(num_steps):
    x_points[i + 1] = x_points[i] + h * f(x_points[i], t_points[i])

for i in range(num_steps):
    k1 = f(x_points1[i], t_points1[i])
    x_temp = x_points1[i] + h * k1
    k2 = f(x_temp, t_points1[i] + h)
    x_points1[i + 1] = x_points1[i] + (h / 2) * (k1 + k2)

for i in range(num_steps):
    k1 = f(x_points2[i], t_points2[i])
    k2 = f(x_points2[i] + h/2 * k1, t_points2[i] + h/2)
    k3 = f(x_points2[i] + h/2 * k2, t_points2[i] + h/2)
    k4 = f(x_points2[i] + h * k3, t_points2[i] + h)
    x_points2[i + 1] = x_points2[i] + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)

ax.plot(t_points, x_points, 'r-', label="Euler Method Solution")
ax.plot(t_points1, x_points1, 'b-', label="Improved Euler Method Solution")
ax.plot(t_points2, x_points2, 'c-', linestyle = 'dashed',label="Runge-Kutta (4th
Order) Solution")
ax.legend()
plt.show()
ax.legend()
plt.show()
```