Computer Simulation Assignment I Code

Valentina Watanabe
21367661

September 2024

1 Section A

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 19 16:51:54 2024

@author: vwitch
"""

#Heron's root finding method

import matplotlib.pyplot as plt
import numpy as np

a = 2

x = a

n = [0]
xn = [0.5]

for i in range(1,10):
    x = (x + (a/x))/2
    xn.append(x)
    n.append(i)

print(x)
plt.figure()
plt.scatter(n, xn, color='m')
plt.plot(n, xn, color='m')
plt.xlabel("n values")
plt.ylabel("xn")
```

```
plt.title("Heron's method, Xn vs n values")
plt.grid("True")
plt.show()

actual_value = np.sqrt(2)
xn_error = []
for i in xn:
    error = np.abs((actual_value - i)) / actual_value
    xn_error.append(error)
plt.figure()
plt.loglog(n, xn_error, color='c')
plt.scatter(n, xn_error, color='c')
plt.plot(n, np.log(xn_error), color='c')
plt.xlabel("n values")
plt.ylabel("xn error")
plt.title("Heron's method, Xn error vs n values")
plt.grid("True")
plt.show()
print("Valentina Watanabe, 21367661")
```

2 Section B

2.1 1.3A

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 21 14:04:28 2024

@author: vwitch
"""
#Findind over and underflow
under = 1.0

over = 1.0

N = 2000

for i in range(1, N + 1):
    over *= 2
    # Detect overflow
    if over == float('inf'):
        print(f"Overflow occurred at loop {i}")
        break
```

```
for i in range(1, N + 1):
    under /= 2
    # Detect underflow
    if under == 0.0:
        print(f"Underflow occurred at loop {i}")
        break


print("Valentina Watanabe, 21367661")
```

2.2 1.3B

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 21 15:13:12 2024

@author: vwitch
"""
eps = 1.0

N=2000

for i in range(N):
    eps = eps/2
    one = 1.0 + eps
    if one == 1.0:
        break

eps = eps*2

print(f"Machine epsilon: {eps}")

eps1 = 1.0 + 0j

N1=2000

for i in range(N1):
    eps1 = eps1 / 2
    one1 = 1.0 + eps1
    if one1.real == 1.0 and one.imag == 0:
        break

eps1 = eps1*2
```

```
print(f"Machine epsilon: {eps1}")
print("Valentina Watanabe, 21367661")
```

2.3 1.4

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 21 14:18:42 2024

@author: vwitch
"""

import numpy as np
import matplotlib.pyplot as plt

# Define functions and their exact derivatives
def func_exp(t):
    return np.exp(t)

def deriv_exp(t):
    return np.exp(t)

def func_cos(t):
    return np.cos(t)

def deriv_cos(t):
    return -np.sin(t)

# Forward difference
def forward_diff(f, t, h):
    return (f(t + h) - f(t)) / h

# Central difference
def central_diff(f, t, h):
    return (f(t + h) - f(t - h)) / (2 * h)

# Relative error
def relative_error(approx, exact):
    return np.abs(approx - exact) / np.abs(exact)

# Function to compute derivatives and relative errors for different step sizes
def compute_derivatives_and_errors(f, df, t_values, h_values):
    forward_results = {}
    central_results = {}
```

```
    for t in t_values:
        forward_derivatives = []
        central_derivatives = []
        forward_errors = []
        central_errors = []

        for h in h_values:
            # Compute forward and central derivatives
            forward_approx = forward_diff(f, t, h)
            central_approx = central_diff(f, t, h)

            # Compute exact derivative
            exact = df(t)

            # Compute relative errors
            forward_err = relative_error(forward_approx, exact)
            central_err = relative_error(central_approx, exact)

            forward_derivatives.append(forward_approx)
            central_derivatives.append(central_approx)
            forward_errors.append(forward_err)
            central_errors.append(central_err)

        forward_results[t] = {'derivatives': forward_derivatives, 'errors':
        forward_errors}
        central_results[t] = {'derivatives': central_derivatives, 'errors':
        central_errors}

    return forward_results, central_results

# Values of t and step sizes h
t_values = [0.1, 1.0, 100.0]
h_values = np.logspace(0, -16, num=17)  # Step sizes ranging from 1 to machine
precision

# Compute for exp(t)
forward_exp, central_exp = compute_derivatives_and_errors(func_exp, deriv_exp,
t_values, h_values)

# Compute for cos(t)
forward_cos, central_cos = compute_derivatives_and_errors(func_cos, deriv_cos,
t_values, h_values)

# Plotting the results
fig, axes = plt.subplots(3, 2, figsize=(14, 12))
```

```
for i, t in enumerate(t_values):
    # Forward difference plot for exp(t)
    axes[i, 0].loglog(h_values, forward_exp[t]['errors'], label='Forward',
    color='red', marker='o')
    axes[i, 0].loglog(h_values, central_exp[t]['errors'], label='Central',
    color='blue', marker='o')
    axes[i, 0].set_title(f"Relative Error for exp(t) at t={t}")
    axes[i, 0].set_xlabel('h')
    axes[i, 0].set_ylabel('Relative Error')
    axes[i, 0].legend()

    # Forward difference plot for cos(t)
    axes[i, 1].loglog(h_values, forward_cos[t]['errors'], label='Forward',
    color='red', marker='o')
    axes[i, 1].loglog(h_values, central_cos[t]['errors'], label='Central',
    color='blue', marker='o')
    axes[i, 1].set_title(f"Relative Error for cos(t) at t={t}")
    axes[i, 1].set_xlabel('h')
    axes[i, 1].set_ylabel('Relative Error')
    axes[i, 1].legend()

plt.tight_layout()
plt.show()
```