National University of Computer and Emerging Sciences
Islamabad Campus

Numerical Computing (CS-2008)

Date: 21 September 2024

Course Instructors

Dr.-Ing. Mukhtar Ullah, Dr. Ir. Imran Ashraf,
Dr. Mir Suleman Sarwar, Almas Khan

Sessional 1
Total Time (Hrs): 1
Total Marks: 60
Total Questions: 4

Roll No ______ Section ______ Student Signature ______

Attempt all the questions
Use answer sheet to answer all questions
Two bonus marks for answering questions in sequence, starting with Question 1 till the end
Properly indent your code and use meaningful names
DO NOT WRITE BELOW THIS LINE

CLO-3 Implement a numerical method in Python/Numpy/Scipy.

Question # 1 [Marks = 10+2]

- The Taylor's expansion of cos(.) centered at $x = 0$ is given by $\cos(x) = \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n}}{(2n)!}$ series. Implement this series by writing a Python function `my_cos` that accepts `x` as the first input and the number `N` of terms in the series as the second input.
- Provide a simple test code that calls `my_cos` to approximate cos(1.5) using 15 terms of the series.

Solution Part 1

```python
import math
def my_cos(x, N):
    res = 0.0
    for n in range(N):
        res += ((-1)**n) * (x**(2*n)) / math.factorial(2*n)

    return res
```

Solution Part 2

```python
x = 1.5
N = 15
cos_approx = my_cos(x, N)
print(f"Approximate cos({x}) with {N} terms: {cos_approx}")
```

CLO-2 Apply numerical methods for approximate solutions to mathematical problem.

Question # 2 [Marks = 4+4+4+4+4]

a) Consider the following code with different values of `n` with its corresponding output

1. Write the output you would expect by hand calculations next to each. If your expected output differs from the generated output, explain the reason for this difference. Identify the type of numerical error (in case of difference) and mention its source and reason (not exceeding three lines).

```
total = 0.0
for i in range(5):
    total += 0.1
for j in range(5):
    total -= 0.1
print(total)
```

Output1:

```
0
```

```
total = 0.0
for i in range(5):
    total += 0.1
for j in range(5):
    total -= 0.1
print(total)
```

Output2:

```
2.7755575615628914e-17
```

```
total = 0.0
for i in range(5):
    total += 0.1
for j in range(5):
    total -= 0.1
print(total)
```

Output3:

```
2.7755575615628914e-17
```

Ans: From code, and its results is clearly showing floating-point precision errors. Some arithmetic between floats number create issues. These round-off errors arise from the binary representation of floating-point numbers, leading to small inaccuracies like 2.7755575615628914e-17 instead of zero

2. Modify the code so that the output returns the expected result.

Ans :Consider the following

```
 total = 0.0
for i in range(5):
total += 0.1
for j in range(5):
total -= 0.1
print(total)
```

Output1:

```
0
```

```
 total = 0.0
for i in range(5):
total += 0.1
for j in range(5):
total -= 0.1
print(round(total,2))
```

Output2:

```
0.0
```

```
 total = 0.0
for i in range(5):
total += 0.1
for j in range(5):
total -= 0.1
print(round(total,2))
```

Output3:

```
0.0
```

b) Consider the following expansion for sin(**.**):

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \tag{1}$$

(a) If we truncate the series to the 3rd term (that involves the 5th power), identify the type of error in our approximation.

Ans: Discretization error will likely happens if we consider upto 3rd term .

(b) Compute the absolute error in the truncated value of sin(3.14) up to the 3rd term (involving 5th power).

Absolute Error:

The truncated approximation of sin(3.14) up to the 3rd term of the series is:

$$\sin(3.14) \approx 3.14 - \frac{3.14^3}{3!} + \frac{3.14^5}{5!} = 0.523849$$

The exact value of sin(3.14) is:

$$\sin(3.14) = 0.001593$$

Thus, the absolute error is:

$$\text{Absolute Error} = |\sin(3.14)_{\text{exact}} - \sin(3.14)_{\text{truncated}}| = |0.001593 - 0.523849| = 0.522256$$

(c) Quantify the relative error between the truncated value and the exact value.

Relative Error:
The relative error is given by:

$$\text{Relative Error} = \frac{|\text{Exact Value (Actual)} - \text{Truncated (Measured) Value}|}{|\text{Exact (Actual) Value}|}$$

Substituting the values:

$$\text{Relative Error} = \frac{0.522256}{0.001593} \approx 327.92$$

CLO-2 Apply numerical methods for approximate solutions to mathematical problem.

Question # 3 [Marks = 4+4]

- Apply appropriate theorem to test the existence of a root for each of the following functions.
  (a) $f_1(x) = \ln(x) + \mathrm{e}^x$ in the interval $[-6, 6]$
  (b) $f_2(x) = \mathrm{e}^x + \cos(x) - x^3$ in the interval $[-4, 4]$

Solution:
We will apply the **Intermediate Value Theorem (IVT)**. Two conditions must be satisfied: 1. The function must be continuous on the given interval. 2. There must be a sign change between the function values at the endpoints of the interval.

(a) **For $f_1(x) = \ln(x) + e^x$ in the interval** $[-6, 6]$:
Continuity: - $\ln(x)$ is only defined for $x > 0$, so $f_1(x)$ is not continuous in the interval $[-6, 6]$ because it includes negative values where the logarithm is undefined.
Conclusion: - Since the function is not continuous in this interval, IVT cannot be applied, and we cannot test for a root in the interval $[-6, 6]$.

(b) **For $f_2(x) = e^x + \cos(x) - x^3$ in the interval** $[-4, 4]$:
Continuity: - $f_2(x)$ is the sum of continuous functions (e^x, $\cos(x)$, and x^3), so it is continuous over the entire interval $[-4, 4]$.
Sign Change: - At $x = -4$:

$$f_2(-4) = e^{-4} + \cos(-4) - (-4)^3 \approx 0.0183 - 0.6536 + 64 = 63.3647 > 0$$

- At $x = 4$:

$$f_2(4) = e^4 + \cos(4) - 4^3 \approx 54.5981 - 0.6536 - 64 = -9.2045 < 0$$

Conclusion: - Since $f_2(-4) > 0$ and $f_2(4) < 0$, there is a sign change. Therefore, by the IVT, there exists at least one root in the interval $(-4, 4)$.

CLO-3 Implement a numerical method in Python/Numpy/Scipy.

Question # 4 [Marks = 10+10]

Iteration	a	b	c	f(a)	f(b)	f(c)	Comments
1	0.4	0.6	0.5	-0.1316426517	0.3393068585	0.1018429767	update b
2	0.4	0.5	0.45	-0.1316426517	0.1018429767	-0.01548156824	update a
3	0.45	0.5	0.475	-0.01548156824	0.1018429767	0.04304572556	update b
4	0.45	0.475	0.4625	-0.01548156824	0.04304572556	0.01374702029	update b
5	0.45	0.4625	0.45625	-0.01548156824	0.01374702029	-0.0008762021942	tolerance level achieved

- Apply the bisection method to approximate the root of the equation

$$x + \sin(x) - \cos(x) = 0$$

  in the interval $[0.4, 0.6]$ with a tolerance of `1e-2`. Clearly show all the required steps.

- Implement a Python function named `bisection_method` that approximates the root of a given function f within a specified interval $[a, b]$. The function should take the following arguments:
  - `f`: A function representing the equation to solve.
  - `a`: The left endpoint of the interval.
  - `b`: The right endpoint of the interval.
  - `tol`: The desired tolerance for the root.
  - `max_iter`: The maximum number of iterations allowed.

  The function should return the approximate root or `None` if the root cannot be found within the specified tolerance or maximum iterations.

```python
import numpy as np
def f(x):
    return x + np.sin(x) - np.cos(x)

def bisection_method(a, b, tol, max_iter):
    if f(a) * f(b) > 0:
        print("Bisection method fails in the given interval.")
        return None

    iteration = 0
    while (b - a) / 2 > tol and iteration < max_iter:
        midpoint = (a + b) / 2
        if f(midpoint) == 0:
            return midpoint  # Exact root found
        elif f(a) * f(midpoint) < 0:
            b = midpoint
        else:
            a = midpoint
        iteration += 1

    if iteration == max_iter:
        print("Maximum iterations reached.")

    # Return the midpoint as the approximate root
    return (a + b) / 2

###### test usage
# interval [0.4, 0.6]
# tolerance of 1e-2
# maximum of 20 iterations
a = 0.4
b = 0.6
tolerance = 1e-2
max_iterations = 20
root = bisection_method(a, b, tolerance, max_iterations)
print('approximate value of root:', root)
```