# CS450 Assignment 1 Jiadong Hong

## Question 1

### (a) What three properties characterize a well-posed problem?

Existence; Uniqueness; Stability.

### (b) List three sources of error in scientific computation

computational error;  propagated data error; Truncation Error;  Rounding Error

### (c) Explain the distinction between truncation (or discretization) and rounding.

Truncation (or discretization) and rounding are both sources of errors in scientific computation, but they involve different aspects of approximation. Truncation or discretization error arises when we approximate a mathematical process or function by converting it into a finite or discrete form. This means that we are simplifying continuous or infinite information into a limited set of values, which can introduce errors as the approximation may not perfectly capture the true nature of the process or function.

## Question 2

### What is an inverse problem?

Inverse problems, also known as inverse theory or inverse modeling, refer to a class of problems in various scientific and engineering fields where you aim to determine the causes or parameters of a system from a set of observations or measurements. In simpler terms, instead of predicting the outcome from known inputs, as in direct or forward problems, you're trying to work backward, figuring out the inputs or parameters that could produce the observed outcomes.

### How are the conditioning of a problem and its inverse related?

1. **Conditioning of a Problem**:
   - The conditioning of a problem assesses its sensitivity to small changes in input data. A well-conditioned problem is not highly sensitive to such changes, while an ill-conditioned problem is very sensitive.
2. **Inverse Problems**:
   - Inverse problems are those where you aim to determine input parameters from observed data. These can also be well-conditioned or ill-conditioned.
   - The conditioning of an inverse problem relates to how sensitive the estimated parameters are to small errors or uncertainties in the observed data.

So according to the lecture notes, we can imply that conditioning number of the inverse of a problem is just the reciprocal / multiplicative inverse of the original problem.

## Question 3

### Which of the following two mathematically equivalent expressions The number e can be defined by

$$e = \sum_{n=0}^{\infty} (\frac{1}{n!})$$

### where n! = n(n − 1)· · · 2 · 1 for n ̸= 0 and 0! = 1. Compute the absolute error and relative error in the following approximations of e

$$\sum_{n=0}^{5} \frac{1}{n!}$$

Absolute Error:

$$\Delta y = |\hat{y} - y| = |\sum_{n=0}^{5} \frac{1}{n!} - e| = 0.0016151617923787498$$

Relative Error:

$$\delta = \frac{|\hat{y} - y|}{y} = 0.0016151617923787498/e = 0.0005941848175817597$$

$$\sum_{n=0}^{10} \frac{1}{n!}$$

Absolute Error:

$$\Delta y = |\hat{y} - y| = |\sum_{n=0}^{10} \frac{1}{n!} - e| = 2.7312660577649694e - 08$$

Relative Error:

$$\delta = \frac{|\hat{y} - y|}{y} = 2.7312660577649694e - 08/e = 1.0047766310211053e - 08$$

## Question 4

Perform the following computations (i) exactly, (ii) using three digit chopping arithmetic, and (iii) using three-digit rounding arithmetic. (iv) Compute the relative errors in parts (ii) and (iii)

### (a)

$$\frac{4}{5} + \frac{1}{3}$$

(i)

$$\frac{4}{5} + \frac{1}{3} = \frac{17}{15}$$

(ii) Chopping: 1.13

(iii) Rounding: 1.13

(iv) relative error: both 0.0029411764705883033

## (b)

$$\frac{4}{5} \cdot \frac{1}{3}$$

(i)

$$\frac{4}{5} \cdot \frac{1}{3} = \frac{4}{15}$$

(ii) Chopping: 0.266

(iii) Rounding: 0.267

(iv) Chopping: 0.002499999999999933; Rounding: 0.0012500000000000705

## (c)

$$(\frac{1}{3} - \frac{3}{11}) + \frac{3}{20}$$

(i)

$$(\frac{1}{3} - \frac{3}{11}) + \frac{3}{20} = \frac{139}{660}$$

(ii) Chopping: 0.210

(iii) Rounding: 0.211

(iv) chopping: 0.0028776978417266374; rounding: 0.001870503597122288

## (d)

$$(\frac{1}{3} + \frac{3}{11}) - \frac{3}{20}$$

(i)

$$(\frac{1}{3} + \frac{3}{11}) - \frac{3}{20} = \frac{301}{660}$$

(ii) Chopping: 0.456

(iii) Rounding: 0.456

(iv) both 0.00013289036544845715

# Question 5

## (a) find

$$\lim_{x \to 0} \frac{e^x - e^{-x}}{x}$$

$$\lim_{x \to 0} \frac{e^x - e^{-x}}{x} = \lim_{x \to 0} \frac{e^x + e^{-x}}{1} = e^0 + e^0 = 1 + 1 = 2$$

### (b) Use three-digit rounding arithmetic to evaluate f(0.1).

$$f(0.1) \approx 2.003335000396882$$

For rounding:

$$f(0.1) \approx 2.00$$

### (c) Replace each exponential function with its third Maclaurin polynomial, and repeat part (b).

$$\begin{aligned}
e^x &= 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + O(x^4) \\
e^{-x} &= 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + O(x^4) \\
\frac{e^x - e^{-x}}{x} &= \frac{\left(1 + x + \frac{x^2}{2} + \frac{x^3}{6} + O(x^4)\right) - \left(1 - x + \frac{x^2}{2} - \frac{x^3}{6} + O(x^4)\right)}{x} \\
&= \frac{1 + x + \frac{x^2}{2} + \frac{x^3}{6} - 1 + x - \frac{x^2}{2} + \frac{x^3}{6} + O(x^4)}{x} \\
&= \frac{2x + \frac{2x^3}{3} + O(x^4)}{x} \\
&= 2 + \frac{2x^2}{3} + O(x^3)
\end{aligned}$$

$$f(x) \approx 2 + \frac{2x^2}{3} + O(x^3)$$
$$\Rightarrow f(0.1) \approx 2.006666666666667$$

For Rounding:

$$f(0.1) \approx 2.01$$

### (d) The actual value is f(0.1) = 2.003335000. Find the relative error for the values obtained in parts (b) and (c).

For (b): 0.0016647240726088577

For (c): 0.0033269523070279913

## Question 6

Use the 64-bit long real format to find the decimal equivalent of the following floating-point machine numbers.

### (a) 0 10000001010 1001001100000000000000000000000000000000000000000000

$$(-1)^0 \times (1 + \frac{1}{2} + \frac{1}{16} + \frac{1}{128} + \frac{1}{256}) \times 2^{11} = 3224$$

### (b) 1100000010101001001100000000000000000000000000000000000000000000

$$(-1)^1 \times (1 + \frac{1}{2} + \frac{1}{16} + \frac{1}{128} + \frac{1}{256}) \times 2^{11} = -3224$$

## Question 7

The sine function is given by the infinite series

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

### (a) What are the forward and backward errors if we approximate the sine function by using only the first term in the series, i.e., sin(x) ≈ x, for x = 0.1, 0.5, and 1.0?

All *BE*=|*x*−*x*|=0

Let's calculate the forward error for the given values of x:

- For x = 0.1: FE = |sin(0.1) - 0.1| ≈ 0.00016
- For x = 0.5: FE = |sin(0.5) - 0.5| ≈ 0.0207
- For x = 1.0: FE = |sin(1.0) - 1.0| ≈ 0.1585

### (b) What are the forward and backward errors if we approximate the sine function by using the first two terms in the series?

- For x = 0.1:
  - FE = |sin(0.1) - (0.1 - (0.1^3)/3!)| ≈ 0.00000161
  - BE = |(0.1^3)/(3!)| = 0.00000161
- For x = 0.5:
  - FE = |sin(0.5) - (0.5 - (0.5^3)/3!)| ≈ 0.00271
  - BE = |(0.5^3)/(3!)| = 0.00271
- For x = 1.0:
  - FE = |sin(1.0) - (1.0 - (1.0^3)/3!)| ≈ 0.0134
  - BE = |(1.0^3)/(3!)| = 0.0134

## Question 8

### (a)

Which of the two mathematically equivalent expressions

$$x^2 - y^2 \text{and} (x - y)(x + y)$$

can be evaluated more accurately in floating-point arithmetic? Why?

In floating-point arithmetic, the expression (*x*−*y*)(*x*+*y*) is generally evaluated more accurately than x^2−y^2. This difference in accuracy arises from how floating-point arithmetic works and the potential for catastrophic cancellation in the subtraction operation.

This means that if *x* and *y* are very close in value, subtracting them directly to compute *x*^2−*y*^2 can lead to a substantial loss of significant digits, which reduces the accuracy of the result.

### (b)

For what values of x and y, relative to each other, is there a substantial difference in the accuracy of the two expressions?

when *x* and *y* are very close in value and x and y have the same sign.

## Question 9

```
import math

# Function to calculate Stirling's approximation
def stirling_approximation(n):
    return math.sqrt(2 * math.pi * n) * (n / math.e) ** n

# Function to calculate the factorial of n
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Function to calculate absolute and relative errors
def calculate_errors(n):
    true_value = factorial(n)
    approx_value = stirling_approximation(n)

    absolute_error = abs(approx_value - true_value)
    relative_error = absolute_error / true_value

    return absolute_error, relative_error

# Calculate errors for n = 1 to 10
for n in range(1, 11):
    abs_error, rel_error = calculate_errors(n)
    print(f'n = {n}:')
    print(f'Stirling\'s Approximation: {stirling_approximation(n)}')
    print(f'True Factorial: {factorial(n)}')
    print(f'Absolute Error: {abs_error}')
    print(f'Relative Error: {rel_error:.4%}')
    print('-----------------------------------')
```

Here is the log:

```
n = 1:
Stirling's Approximation: 0.9221370088957891
```

```
True Factorial: 1
Absolute Error: 0.07786299110421091
Relative Error: 7.7863%
----------------------------------
n = 2:
Stirling's Approximation: 1.9190043514889832
True Factorial: 2
Absolute Error: 0.08099564851101682
Relative Error: 4.0498%
----------------------------------
n = 3:
Stirling's Approximation: 5.836209591345864
True Factorial: 6
Absolute Error: 0.16379040865413597
Relative Error: 2.7298%
----------------------------------
n = 4:
Stirling's Approximation: 23.506175132893294
True Factorial: 24
Absolute Error: 0.4938248671067065
Relative Error: 2.0576%
----------------------------------
n = 5:
Stirling's Approximation: 118.0191679575901
True Factorial: 120
Absolute Error: 1.9808320424099009
Relative Error: 1.6507%
----------------------------------
n = 6:
Stirling's Approximation: 710.078184642185
True Factorial: 720
Absolute Error: 9.921815357815035
Relative Error: 1.3780%
----------------------------------
n = 7:
Stirling's Approximation: 4980.395831612462
True Factorial: 5040
Absolute Error: 59.604168387538266
Relative Error: 1.1826%
----------------------------------
n = 8:
Stirling's Approximation: 39902.39545265671
True Factorial: 40320
Absolute Error: 417.6045473432896
Relative Error: 1.0357%
----------------------------------
n = 9:
Stirling's Approximation: 359536.87284194835
True Factorial: 362880
Absolute Error: 3343.1271580516477
Relative Error: 0.9213%
----------------------------------
n = 10:
Stirling's Approximation: 3598695.6187410373
True Factorial: 3628800
Absolute Error: 30104.381258962676
```

```
Relative Error: 0.8296%
----------------------------------
```

As we can see the absolute error is growing while the relative error is decreasing.

# Question 10

## (a)

The expression |3 * (4/3 - 1) - 1| calculates ε by exploiting the property that in many binary floating-point systems, the value 4/3 cannot be represented exactly, so it introduces a small error. Subtracting 1 and then multiplying by 3 magnifies this error, and finally subtracting 1 again helps to isolate the error term. This error term approximates the unit roundoff ε for the specific floating-point representation.

## (b)

```python
import numpy as np

# Single precision (32-bit)
epsilon_single = abs(np.float32(3) * (np.float32(4)/np.float32(3) -
 np.float32(1)) - np.float32(1))
print(f"Single Precision: {epsilon_single}")

# Double precision (64-bit)
epsilon_double = abs(np.float64(3) * (np.float64(4)/np.float64(3) -
np.float64(1)) - np.float64(1))
print(f"Double Precision: {epsilon_double}")
```

here is log:

```
Single Precision: 1.1920928955078125e-07
Double Precision: 2.220446049250313e-16
```

It did work.

## (c)

The trick might not work as expected in a floating-point system with base β = 3. The reason is that the representation of numbers and the properties of ε depend on the base of the system. In a base-3 system, numbers like 4/3 might have exact representations, and the error magnification process used in the trick may not produce an accurate approximation of ε. This is because the fundamental properties of ε in a base-3 system can be different from those in the more common binary systems. To estimate ε in a base-3 system, you would need to understand its specific characteristics and representation details.