Lab Manual for Numerical Analysis

Lab No. 2

MEASURING ERRORS, TYPES OF ERRORS, PROPAGATION OF ERRORS, ROUND OFF AND TRUNCATION ERRORS

Software Engineering Department
Bahria University (Karachi Campus)

BAHRIA UNIVERSITY KARACHI CAMPUS
Department of Software Engineering

NUMERICAL ANALYSIS

LAB EXPERIMENT # 2

Measuring errors, types of errors, propagation of errors, Round Off and Truncation Errors

OBJECTIVE:

The lab's objective is to comprehensively explore the concept of uncertainty, and teach students about different types of errors and provide a deep understanding of error measurement strategies. Hands-on exercises will enhance their understanding of error analysis in scientific and engineering contexts.

Introduction

In the ever-expanding landscape of science, engineering, and data-driven decision-making, **numerical analysis** stands as a cornerstone discipline. It empowers us to harness the power of mathematics and computation to solve **complex problems**, simulate intricate systems, and make informed choices based on numerical data. At the core of this discipline, a fundamental concept prevails, influencing every aspect of its application: **errors.**

Errors, in the context of computing and data analysis, refer to **discrepancies** or **deviations** between **observed, calculated,** or **measured** values and their **expected** or **true values.** They can arise from various sources, including limitations in measurement instruments, approximation methods, data quality, or computational precision.

Errors play a fundamental role in evaluating precision and reliability across scientific, engineering, and data-driven domains. Proficiency in comprehending and managing errors is indispensable for making informed choices and refining numerical accuracy. Errors extend beyond mere numerical values; they carry concrete consequences in real-world scenarios, significantly influencing outcomes, from the success of space missions to the performance of financial investments. In these contexts, they can often be the deciding factor between triumph and adversity.

In the upcoming sections, we'll delve into **error types including round off errors, truncation errors and inherent errors** (error propagation), examining how input inaccuracies impact numerical precision.

Measuring Errors

Measurement constitutes a fundamental component of scientific computations, yet absolute precision in measurements remains exceedingly rare. In the process of quantifying various parameters, it is common to encounter minor discrepancies, commonly referred to as errors. These errors encompass diverse categories, each of which can be mathematically expressed. Acquiring a comprehensive understanding of these errors is pivotal, as it not only facilitates accurate calculations but also affords opportunities for error rectification.

Broadly, measuring errors can be categorized into two primary types: **absolute** and **relative errors.**

a. Absolute Errors

Absolute error is a measurement of the magnitude or size of the difference between an observed or calculated value and the true or expected value. It quantifies how much a measurement or calculation deviates from the accurate or desired result, regardless of the direction of the deviation.

The absolute error is calculated by the subtraction of the actual value and the measured value of a quantity. If the actual value is x_0 and the measured value is x, the absolute error is expressed as,

$$\Delta x = x_0 - x$$

Here, Δx is the absolute error.

Example:

Let's consider an example where we measure the temperature of a substance, and we have the true temperature value as a reference. We will calculate the absolute error in Python:

```
# True temperature value (ground truth)
true_temperature = 25.0  # in degrees Celsius

# Measured temperature
measured_temperature = 24.5  # in degrees Celsius

# Calculate the absolute error
absolute_error = abs(measured_temperature - true_temperature)

# Display the result
print("True Temperature:", true_temperature, "°C")
print("Measured Temperature:", measured_temperature, "°C")
print("Absolute Error:", absolute_error, "°C")
```

In this Python script, we calculate the absolute error by finding the absolute difference between the measured temperature and the true temperature. This helps us quantify how much our measured value deviates from the actual temperature.

b. Relative Errors

Relative error is a dimensionless measurement that quantifies the proportion of the absolute error in a measurement or calculation relative to the true or expected value. It expresses the error as a percentage of the true value, providing a way to gauge the magnitude of the error in relation to the scale of the measurement.

The relative error is calculated by the ratio of absolute error and the actual value of the quantity. If the absolute error of the measurement is Δx, the actual value is x0, the measured value is x, the relative error is expressed as,

$$x_r = ((x_0 - x)/ x_0) * 100$$
$$= (\Delta x/x_0) * 100$$

Here, xr is the relative error.

Example:

Let's consider an example where we are calculating the relative error when measuring the weight of an object, comparing it to the true weight:

```
# True weight of the object (ground truth)
true_weight = 500.0  # in grams

# Measured weight
measured_weight = 495.0  # in grams

# Calculate the relative error as a percentage
relative_error = (abs(measured_weight - true_weight) / true_weight) * 100

# Display the result
print("True Weight:", true_weight, "grams")
print("Measured Weight:", measured_weight, "grams")
print("Relative Error:", relative_error, "%")
```

In this Python script, we calculate the relative error by finding the absolute difference between the measured weight and the true weight, dividing it by the true weight, and expressing the result as a percentage. This provides a relative measure of the error in relation to the true weight of the object.

Propagation of Errors

Propagation of errors, also known as inherent error or propagation or error analysis, refers to the process of quantifying how uncertainties or errors in the input variables of a mathematical or computational model affect the uncertainty or error in the output or result of that model. It involves determining how variations or inaccuracies in the initial data or measurements contribute to the overall uncertainty in the final calculated value.

The general formula for propagating uncertainties or errors in a function involving one or more variables is as follows:

For Addition and Subtraction

When you perform addition (R=A+B) or subtraction (R=A−B) of values A and B with associated uncertainties ΔA and ΔB, the propagated uncertainty (ΔR) is calculated as:

$$\Delta R = \sqrt{(\Delta A)^2 + (\Delta B)^2}$$

For Multiplication and Division:

When you perform multiplication (R=A·B) or division (R=A/B) of values A and B with associated uncertainties ΔA and ΔB, the propagated relative uncertainty Type equation here. is calculated as (for Multiplication (R=A·B) while for Division (R=A/B) :

$$\Delta R = \mathrm{R\ x}\ \sqrt{(\frac{\Delta A}{A})^2 + (\frac{\Delta B}{B})^2}$$

Example:

Consider the following simple example of propagation of errors (addition):

```
# Define variables and uncertainties
x = 5.0              # Measured value of x
delta_x = 0.2        # Uncertainty in x
y = 3.0              # Measured value of y
delta_y = 0.1        # Uncertainty in y

# Perform addition
result = x + y

# Calculate the propagated uncertainty
delta_result = math.sqrt((delta_x)**2 + (delta_y)**2)

# Display the result and propagated uncertainty
print("Result (x + y) =", result)
print("Propagated Uncertainty =", delta_result)
```

In this python example, we have measured values for x and y with associated uncertainties (Δx and Δy). Firstly addition is performed as (x+y) to obtain result, and then the propagated uncertainty (Δresult) for addition is simply calculated as sum of the uncertainties in x and y.

Example:

Let's illustrate the propagation of errors using a Python example. We'll calculate the volume of a cuboid based on measurements of its length (L), width (W), and height (H), all of which have associated uncertainties (ΔL, ΔW, and ΔH).

The formula for the volume of a cuboid is $V=L \cdot W \cdot H$, and we'll use the formula for propagation of errors to estimate the uncertainty (ΔV) in the calculated volume:

```
import math

# Measurements and their uncertainties
length = 10.0  # cm
width = 5.0    # cm
height = 3.0   # cm

delta_length = 0.2  # cm
delta_width = 0.1   # cm
delta_height = 0.05  # cm

# Calculate the volume of the cuboid
volume = length * width * height

# Calculate the propagated uncertainty in the volume
delta_volume = volume * math.sqrt(( delta_length/length)**2 +
                (delta_width/width)**2 +
                (delta_height/height)**2)

# Display the results
print("Length:", length, "cm")
print("Width:", width, "cm")
print("Height:", height, "cm")
print("Volume:", volume, "cubic cm")
```

In this Python example, we calculate the volume of a cuboid using measurements of length, width, and height, along with their associated uncertainties. We then use the formula for propagation of errors to estimate the propagated uncertainty in the calculated volume. This demonstrates how uncertainties in input measurements affect the uncertainty in the final result, providing a more accurate representation of the precision of the volume calculation

Round Off Errors

Round-off errors arise from the finite precision of computers in representing numerical values. For instance, consider π; it possesses an infinite number of digits, but in Python, due to precision constraints, only a finite number of decimal places can be stored. While these round-off errors may appear inconsequential, they can amass significance in scenarios involving iterative processes dependent on each other, potentially resulting in substantial deviations from the expected outcome. Additionally, when performing arithmetic operations involving both large and small numbers, the precision of the smaller number may be compromised through rounding. As a best practice, it is advisable to prioritize summations of numbers with comparable magnitudes to preserve precision when dealing with smaller values.

Example:

Consider the following example that illustrates round-off error in the context of iterative calculations, similar to the accumulation of errors in real-world scenarios:

```
def ln_approximation(x, n):
    result = 0.0
    for i in range(1, n + 1):
        term = ((-1) ** (i - 1)) * ((x - 1) ** i) / i
        result += term
    return result

x = 2 # The value for which you want to calculate ln(2)
n_terms = 100  # Number of terms (adjust for accuracy)

approximation = ln_approximation(x, n_terms)

# Calculate the true value (the natural logarithm of 2)
true_value = math.log(4)

# Calculate the round-off error
round_off_error = abs(true_value - approximation)

print("Approximated Value:", approximation)
print("True Value (ln(2)):", true_value)
print("Round-off Error:", round_off_error)
```

In this Python example, we use an alternating series to approximate the natural logarithm of 2 (**ln(2)**). We perform a large number of iterations (**num_iterations**) to approach the true value of **ln(2)**. However, due to round-off errors introduced in each iteration, the approximation may not be perfectly accurate. The round-off error quantifies the difference between the true value and the approximation, showcasing how round-off errors accumulate in complex computations.

Truncation Errors

Truncation error occurs when we simplify a mathematical process or function using a limited number of terms or operations. It emerges from the challenge of approximating infinite or continuous processes with finite calculations, leading to disparities between the approximation and the actual process. Similar to round-off errors, truncation errors are rooted in the inherent constraints of numerical computations.

From the standpoint of numerical analysis, truncation errors are a crucial consideration when approximating complex mathematical operations. They can accumulate as we reduce the complexity of a process by taking fewer terms or steps, potentially affecting the accuracy of our results. Truncation errors often become more pronounced when dealing with highly oscillatory or rapidly changing functions.

Example:

Consider approximating the value of the mathematical constant "e" using a truncated series expansion such as the Maclaurin series:

$$\mathbf{e} \approx 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$

If we stop after a finite number of terms, we obtain an approximation of "e" that contains a truncation error. The more terms we include, the closer our approximation gets to the actual value of "e."

```
import math

def calculate_e(n):
    e_approximation = 1.0  # Initialize with the first term of the series

    for i in range(1, n):
        term = 1.0 / math.factorial(i)  # Calculate the next term in the series
        e_approximation += term
    return e_approximation

n_terms = 10  # Number of terms in the series (adjust for accuracy)

# True value of e
true_e = math.exp(1)

# Approximate e using a truncated series with 5 terms
e_approximation = calculate_e(n_terms)

# Calculate the truncation error
truncation_error = abs(true_e - e_approximation)
print("True e:", true_e)
print("Approximated e:", e_approximation)
print("Truncation Error:", truncation_error)
```

In this Python example, we approximate the value of "e" using a truncated series with 5 terms, leading to a truncation error. The truncation error quantifies the difference between the actual value of "e" and our approximation.

Lab Tasks

1. Write a Python program that calculates the absolute as well as relative error present in the following measurements;

   Actual values: [11.0098, 167.902, 56.0567, 67.9860]
   Measured values: [12.0001, 166.802, 55.0001, 69.0000]

2. Write a Python program the find the propagated error in the area and perimeter found for the following measurements:
   - SQUARE (length : 45.09 cm ; uncertainty : 0.01 cm)
   - CIRCLE (radius : 34.90 cm ; uncertainty : 0.05 cm)
   - TRIANGLE (side1 : 70.9 m ; uncertainty : 0.23 m,
     side2 : 89.07cm ; uncertainty : 0.07 m,
     base : 76.07cm ; uncertainty : 0.04 m,
     height : 100.07cm ; uncertainty : 0.05 m)
   - TRAPEZIUM(side1 : 670.9 m ; uncertainty : 0.53 m (parallel one),
     side2 : 849.07cm ; uncertainty : 0.27 m (parallel one)
     side3 : 376.07cm ; uncertainty : 0.74 m,
     side4 : 716.07cm ; uncertainty : 0.14 m,
     height : 231.07cm ; uncertainty : 0.25 m)

3. Write a Python program that calculates the square root of following numbers using both the math.sqrt function (which uses floating-point arithmetic) and a custom square root function that uses integer arithmetic. Then, find and compare the results to observe the rounding error.

   (56.90, 100.45, 67.90, 25.67, 56.67)

4. Write a python program which find the value of PI (π) using Taylor series, and then find the truncating error occurred due to the use of finite number of terms.

   HINT: arctan (1) = π/4, and formula for finding Tylor series for arctan is:

$$\mathbf{arctan(x)} = \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{2n+1};\ |z| \leq 1,\ z \neq i, i$$