3.1 Problem: Living with Errors

Errors and uncertainties are introduced by the computer. In this chapter we examine some of the consequences of errors. In particular, we determine how errors come about in a computer. Although it is possible to keep errors to a minimum, errors do occur. Consequently we must learn to cope with errors. We will see how to reduce the amount of errors introduced by the computer. Computer errors arise either because of the hardware or software. Some errors are the results of human mistakes while others are not. Whether you like it or not, errors and uncertainties are a part of computer computations.

3 Errors in Computations

Errors and Uncertainties

3.25 Assessment

By exp-a.d. x, we mean the sum of a „good way“ and „bad way“ (explicitly evaluated). A similar statement applies to the sum of a „good way“ and „bad way“ (inaccurate) to one of the terms. Modify your code to compute the sums in a „good way“ (inaccurate) to one of the terms.

2.24 Implementation: Bad Algorithm, Exp-Bad F (C)

Modify your code to compute the sums in a „good way“ (inaccurate) to one of the terms.

Where exp(x) is calculated with the built-in exponential function.

\[
\text{sum} = \exp(x) - \text{sum}
\]

Present your results as a table or plot of the form.

2.23 Implementation: Good Algorithm, Exp-Good F (C)

Write a program that implements this procedure for the indicated x values.
The representation of a simple subtraction is shown:

\[
\begin{align*}
(3.7) & \quad c - q = \frac{c}{q} \\
(3.8) & \quad 2q - q + 1 = \frac{2q}{q} \\
(3.9) & \quad c - q = \frac{c}{q} \\
\end{align*}
\]

The result of the subtraction is the difference of the representations of the two operands. The result is then represented in a normalized form to represent the difference accurately. This process is known as the subtraction cancellation.

3.3 Model: Subtractive Cancellation

When dealing with floating-point numbers, errors in the representation of numbers can arise due to the finite precision of floating-point arithmetic. This can lead to inaccuracies in calculations, especially when dealing with subtraction. For example, if you subtract two nearly equal numbers, the result may lose precision, leading to an inaccurate result.

\[
(3.10) \quad 3.3 \times 10^{-16} + 2.5 \times 10^{-16} = 5.8 \times 10^{-16}
\]

This is due to the fact that floating-point numbers are represented as a fixed-size binary fraction. The precision is limited to a certain number of bits. When two numbers that are very close in value are subtracted, the result may not be as precise as expected. This can lead to errors in calculations, especially when dealing with very small or very large numbers.

Random errors: These are errors that occur randomly and are difficult to predict. They can be caused by various factors, such as hardware faults or software bugs. Random errors can affect the accuracy of calculations, leading to unpredictable results.

Blunders: These are errors that occur when there is a mistake in the problem or in the method of solving it. These errors are usually due to typos or other oversights. Blunders can be avoided by carefully checking the problem and the solution method.

3.2 Theory: Types of Errors

Errors and uncertainties in computations can be categorized into several types, including:

- Round-off errors: These occur due to the finite precision of floating-point arithmetic.
- Truncation errors: These occur when a mathematical operation is approximated by a simpler operation.
- Scaling errors: These occur when an operation is performed on numbers that are very large or very small.
- Algebraic errors: These occur when algebraic manipulations are performed on equations.

Understanding these types of errors is crucial in designing computational methods and in assessing the reliability of the results.
(3.17) \[ \frac{u}{N} \sum_{N} = (a) S \]

These are a simple set of returns on the previous and careful consideration of the problem. A simple set may require some thought and care. Consider the following:

1. **Simplify the Problem:**
   - Start with \( N = 10000 \) and work up to \( N = 20000 \). Use the log-log plot of the error versus the number of terms. When it is log-log, the slope of the graph should be about 40.
   - Assume \( S \) to be the exact answer. Make a log-log plot of the error versus \( N \).
   - Where a single-precision program that calculates \( S \), and
     
   2. **Calculate:**
   - The numerical error:
     
   3. **Calculate:**
   - If you sum even and odd values separately, you get two sums.

### 3.4.1. ASSESSMENT: SUBTRACTION CANCELLATION EXPERIMENT

- **Objective:**
  - To assess the impact of subtraction cancellation on the accuracy of numerical computations.
  - To explore the effects of various multiplicands on the cancellation error.
  - To determine the conditions under which cancellation can be avoided or minimized.

- **Methodology:**
  - Conduct experiments with different multiplicands and observe the cancellation error.
  - Analyze the impact of the multiplicand's magnitude on the error.
  - Investigate the role of the base and the number of digits in determining the error.

- **Results:**
  - The error increases as the magnitude of the multiplicand increases.
  - The base of the number system also plays a significant role in the error.
  - The number of digits in the multiplicand affects the error, with a higher number of digits leading to a greater error.

- **Conclusions:**
  - To minimize cancellation errors, choose multiplicands with similar magnitudes.
  - Use a higher base or a larger number of digits to mitigate the error.

### Example

- **Experiment Configuration:**
  - Multiplicand: \( 1000 \), \( 10000 \), and \( 100000 \)
  - Base: \( 10 \)
  - Digits: 6, 10, and 15

- **Observations:**
  - Error increases with the magnitude of the multiplicand.
  - Higher bases and more digits reduce the error.

- **Implications:**
  - Choose appropriate multiplicands to minimize error.
  - Use higher bases and more digits to improve accuracy.

---

**Errors and Uncertainties in Computations**

- **Definition:**
  - Errors in calculations can be due to various factors, including rounding, truncation, and cancellation.

- **Types of Errors:**
  - **Rounding Error:**
    - Occurs when numbers are rounded during the computation.
    - Can be significant in floating-point arithmetic.
  - **Truncation Error:**
    - Arises from the finite precision of computer arithmetic.
    - Affects the accuracy of calculations.
  - **Cancellation Error:**
    - Occurs when two nearly equal numbers are subtracted.
    - Can lead to significant loss of accuracy in numerical computations.

- **Mitigation Techniques:**
  - Use higher precision arithmetic.
  - Avoid subtracting nearly equal numbers.
  - Use algorithms that minimize error propagation.

---

**References:**

3.6 Problem 1: Errors in Spherical Bessel Functions

The errors are expressed by means of the amount of computer time required to perform a certain class of computations. A problem of computation is defined as the time required to perform a certain class of computations. The errors are expressed by means of computer time required to perform a certain class of computations. The errors are expressed by means of computer time required to perform a certain class of computations.
method: numeric recursion relations

3.1.3. the first four special bessel functions, \( y(x) \), as functions of \( x \)

<table>
<thead>
<tr>
<th>0.0</th>
<th>1.0</th>
<th>1.5</th>
<th>2.0</th>
<th>2.5</th>
<th>3.0</th>
<th>3.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
</tr>
</tbody>
</table>

approximate values for special bessel functions of orders 0, 1, and 2

\[ y_0(x) \approx 1 \]
\[ y_1(x) \approx 0 \]
\[ y_2(x) \approx 0 \]
\[ y_3(x) \approx 0 \]

errors and uncertainties in computations
3.10 Problem 2: Errors in Algorithms

The reason for this is that while your program types out what 0.1 and 0.09 report, they are not the same numbers. Explain why this is so.

The errors in the upward recursion depend on x, and for certain values

\[ \begin{align*}
|f_{\text{upward}}(x) - f(x)| &= |(c_{\text{op}} f(c)) + (c_{\text{fn}} f(c))| \\
&= |\frac{c_{\text{op}}}{c_{\text{fn}}} - 1| f(x)
\end{align*} \]

3.9 Assessment

1. Give reasons for the downward recursion for different, large values of the

3.8 Implementation: Recursion Relations, Bessel's (c)

1. Write a program to calculate J_n(x) which will give "good" values for the

2. It is with both upward and downward recursion, but don't do any hand

Example values:

f(0.1) ≈ \ldots \quad f(1.0) \approx \ldots

\[ f(0.1) \approx \ldots \quad f(1.0) \approx \ldots \]

I. Which is right? Why?

2. How accurate are the results when it does converge?

3. How accurate are the results when it does converge?
of error should dominate the computational cost for very large N. Because the approximation error should get smaller with larger N, the round-

\[
N^a \approx \frac{\varepsilon}{\tau} N
\]

3.13 Method: Empirical Error Analysis

If the number of conditional steps \( N \) and cost \( c \) are both large, the total cost \( \tau N \) is fast decreasing. This results from the machine precision approximation error being just the difference of \( \frac{\varepsilon}{\tau} \) and the number of operations \( N \) when the approximation is made. In this case, we know in some approximate analysis that the total error (which is the sum of the approximation error and the number of operations) is not too large. We therefore can make use of this approximation to derive the total error.

Let us say you have a program you want to optimize for minimum total error.

The total error is now

\[
N^a + \frac{\varepsilon}{\tau} N \leq \frac{\varepsilon}{\tau} N
\]

In order to see more clearly how different kinds of errors balance of each other, we assume the approximation error to be known.

3.13.1 Method: Optimizing with Known Error Behavior

In general, there may be a problem in your program, or the model may be too simple, and the steps should be taken. For this reason, we have to do the following: First, take the round-off term to dominate the total error (\( \tau N \)).

Because the round-off error is usually larger than the approximation error, we can move the line of control of the algorithm to be the line closest to the approximation error. Then, we can use the following equations to find out how the approximation error is balanced in each step.

This shows that for a typical algorithm, most of the error is due to round-off.

\[
N^a \approx \frac{\varepsilon}{\tau} N
\]
\[ z = 0.1 \text{ and } 100. \]

1. Write a program that calculates \( z \) as the finite sum (3.9c).

To more readily see the effects of error accumulation in this algorithm, note

\[
\frac{n}{u(x)} \sum_{k=0}^{n} \tilde{\epsilon}^k 
\]

(3.9c)

\[ (\infty > \tilde{\epsilon} x) \ldots + \frac{\tilde{\epsilon} x}{\tilde{\epsilon} x} x - I = \tilde{\epsilon} \cdot \epsilon. \]

Consider the series for the exponential function.

2.4.4 EXPERIMENT

The number of decimal places of precision obtained

\[ \text{can be determined by subtracting the results of the two divisions of (3.4).} \]

Is this error the same size as the roundoff error? (You may assume that

\[ \text{errors accumulate and are larger than roundoff errors.} \]

Is it possible to avoid this inaccuracy of \( z \) which the approximation error

\[ \text{becomes which is a table of \( z \) for which the approximation error.} \]

4. By numerical approximation, use your program to experimentally check

\[ \text{the above expression for large \( x \).} \]

For \( \epsilon = 0.0001 \), it is seen that this approximation is quite

\[ \text{accurate for \( x \) large.} \]

5. Determine whether (3.4) is valid and, if so, determine the value for \( g' \).

The only way to do this is to experimentally check

\[ \text{the above expression for large \( x \).} \]

For \( \epsilon = 0.0001 \), it is seen that this approximation is quite

\[ \text{accurate for \( x \) large.} \]

To verify this, see if these approximations are valid and the

\[ \text{approximation error is not yet dominating. When} \]

\[ \text{with a large number of steps, and again with twice the number of steps.} \]

\[ \text{where \( a' \) and \( b' \) are unknown constants. We now see one complete program} \]

\[ \text{can write} \]

\[ g'N = (N_1)N - (N_1)N \]

\[ \text{as \( N \) dominates, it is no longer accurate to dominate roundoff error. In this case we} \]

\[ \text{are left with an unknown value for \( z \).} \]

The number of steps of \( (N_1)N \) should be \( N \)

\[ \text{opposite your algorithm after \( N \) steps is \( N_1 \).} \]

Assessments Experiments