Self Assignment: Numerical Analysis

2024

Introduction

Numerical analysis as a whole is a branch of mathematics that deals with algorithms for obtaining numerical solutions to mathematical problems. It will play a crucial role in scientific computing. Precise solutions are not always feasible and hence the two fundamental concepts that we'll explore in this document are **Error Analysis** and **Numerical Stability**.

1 Error Analysis

Error analysis will help us understand how far an approximate value is from the true value. In numerical methods, there are typically two types of errors that we need to account for:

- **Round-off error:** This happens because computers can only store numbers up to a certain precision (using a limited number of decimal places). So, when a number can't be exactly represented, it gets rounded off, causing a small error.
- **Truncation error:** This occurs when a calculation that should go on indefinitely is cut off (or truncated). For example, when using a Taylor series (which adds an infinite number of terms), we often stop after a few terms, leaving some error.

1.1 Absolute and Relative Error

To quantify errors, we often use two terms: absolute error and relative error. These help us measure how far off the approximation is from the exact value.

- **Absolute error:** This is simply the difference between the exact value x_{exact} and the approximate value x_{approx}. The formula is:

$$E_{\text{abs}} = |x_{\text{exact}} - x_{\text{approx}}|$$

- **Relative error:** This gives us a sense of how significant the error is relative to the size of the exact value. It's the ratio of the absolute error to the exact value, calculated as:

$$E_{\text{rel}} = \frac{|x_{\text{exact}} - x_{\text{approx}}|}{|x_{\text{exact}}|}$$

1.2 Example: Computing the Square Root of 2

Let's take an example where we approximate the square root of 2. The exact value is $\sqrt{2} \approx 1.4142136$. Suppose a numerical method gives us an approximate value of $x_{\text{approx}} = 1.414$.

We can now calculate both the absolute error and the relative error.

- **Absolute error:**

$$E_{\text{abs}} = |x_{\text{exact}} - x_{\text{approx}}| = |1.4142136 - 1.414| = 0.0002136$$

So, the difference between the exact value and the approximation is 0.0002136.

- **Relative error:**

$$E_{\text{rel}} = \frac{|x_{\text{exact}} - x_{\text{approx}}|}{|x_{\text{exact}}|} = \frac{0.0002136}{1.4142136} \approx 0.000151$$

This means the error is about 0.0151% of the exact value, which is quite small.

1.3 Explanation for Beginners

- - **Absolute error** tells us how much the approximate value differs from the exact value in absolute terms.
- - **Relative error** helps us understand how big this error is compared to the exact value. For example, if the exact value is large, a small absolute error might not be a big deal, but if the exact value is small, that same absolute error might be more significant.

In this example, the absolute error is very small, and the relative error is even smaller, which means our approximation of $\sqrt{2}$ is quite accurate.

2 Conclusion

By using both absolute and relative error, we can better understand how good (or bad) our approximations are in numerical computations. Error analysis is important because it helps us figure out how much trust we can place in the results of numerical methods.

3 Numerical Stability

A numerical algorithm is said to be **numerically stable** if small changes in the input (such as rounding errors in the data) result in only small changes in the output. This is important because computers deal with approximations due to their limited precision, and we want our algorithms to give reliable results even when these small errors occur.

On the other hand, an algorithm is considered **unstable** if even small changes in the input can cause large, unpredictable changes in the output. When an algorithm is unstable, the results can become inaccurate very quickly.

3.1 Condition Number

One useful way to measure how sensitive a problem is to changes in the input is by using the **condition number**. The condition number tells us how much an error in the input can affect the output. A high condition number means the problem is sensitive to errors and could be unstable, while a low condition number means the problem is stable.

For a function $f(x)$, the condition number is defined as:

$$\kappa = \frac{|x|}{|f(x)|} \cdot \left| \frac{f'(x)}{f(x)} \right|$$

Here's what this means: - $|x|$ is the absolute value of the input. - $|f(x)|$ is the absolute value of the output. - $f'(x)$ is the derivative of the function $f(x)$, which tells us how fast $f(x)$ is changing with respect to x.

In simple terms, if the condition number κ is large, even small errors in the input (like rounding or small measurement errors) can cause large errors in the output. In this case, the problem is said to be **ill-conditioned**. If κ is small, the problem is **well-conditioned**, meaning the output won't change much if the input has small errors.

3.2 Example: Stability in Solving Linear Systems

To illustrate this concept, let's look at the problem of solving a system of linear equations, $A\mathbf{x} = \mathbf{b}$, where A is a matrix, $\mathbf{x}$ is the vector of unknowns, and $\mathbf{b}$ is the vector of known values.

We often use a method called **Gaussian elimination** to solve such systems. However, if the matrix A is close to being *singular* (which means that its determinant is very small or close to zero), the system becomes difficult to solve accurately. In this case, the condition number of A, denoted $\kappa(A)$, will be very large.

What does this mean? - If the condition number $\kappa(A)$ is large, the system of equations is *ill-conditioned*. This means that small errors in the data (for example, errors due to rounding or measurement inaccuracies) can lead to large errors in the computed solution. - If the condition number $\kappa(A)$ is small,

the system is *well-conditioned*, and small errors in the data will only lead to small errors in the solution.

3.3 Detailed Example: Large Condition Number

Suppose we are solving a system where the condition number of matrix A is $\kappa(A) = 10^6$. This means that any small error in the input can be magnified by a factor of 10^6.

Let's say we introduce an error of size 10^{-6} into $\mathbf{b}$ (the vector of known values). This small error could lead to an error in the computed solution $\mathbf{x}$ of size 1. In other words, even though the error in $\mathbf{b}$ was very tiny, the error in $\mathbf{x}$ (the solution) becomes very large, making the problem unstable.

3.4 Why is This Important?

Numerical stability is crucial because, in real-world problems, errors are unavoidable due to limitations in how computers represent numbers. If an algorithm is numerically unstable, these small errors can grow and make the final result very inaccurate or even completely wrong.

Therefore, when designing numerical algorithms, we aim to ensure that they are numerically stable, meaning that they can handle small errors without drastically affecting the result.