Chapter 1

Introduction

1.1 Introduction

Many real world problems, from computing the income tax to design of Nuclear Bomb, depend on simulation and prediction through complex mathematical models and a large amount of data. Mathematical modelling utilizes physical laws to develop equations representing the system behaviour. In some cases, a phenomenon may be very difficult or hazardous or expensive to model experimentally but may yield to *reasonably accurate* mathematical modeling. The mathematical model representing a physical system may be solved using experimental, analytical, and numerical methods or a combination of them. Rarely, however, does one obtain an “exact” solution because of the approximations introduced in the process at various stages. The mathematical model may, knowingly or inadvertently, involve some simplifying assumptions, the experimental procedure may have limitations in the measurement accuracy, the analytical solution may involve irrational numbers or infinite series, and the numerical solution invariably contains errors due to limited precision of computers. The objective, therefore, is to obtain a *reasonably accurate* solution with the *optimum use of resources*. The desired accuracy and efficiency would, of course, depend on the physical problem and must take into account both the nature of the problem and the intended use of the answer.

In the experimental methods, we reproduce the system using full-size or scaled models and obtain the solution by measuring (and, if needed, scaling) the relevant characteristics. There is generally an excellent representation of the physical system but the process is typically slow and often expensive. Scale effects are sometimes non-linear and complicated which need to be properly accounted for in the interpretation of results. Measurement accuracy, cost and scalability often restrict the application of such models.

Analytical solutions, if possible, are very useful for understanding the response of the system. However, these are often limited to very simple geometry and finding a solution can often be extremely difficult. Sometimes, the analytical solution may involve an infinite series, which may require summation of a large number of terms for desired accuracy and may be time consuming to evaluate.

More often than not, it is impossible to obtain a closed form solution of the complex mathematical models. Instead, one relies on large number of numerical calculations to obtain an approximation to the solution of the model. The branch of mathematics that deals with transformation of the mathematical problems to numerical calculations is often termed as *Numerical Methods*. Essentially, a numerical method transforms a mathematical problem in to a set of computations involving only addition, subtraction, multiplication and division. Mathematical analysis that helps to determine the proximity of the approximate solution, obtained using a numerical method, to the exact solution is termed *Numerical Analysis.*

Numerical methods use numbers to simulate mathematical processes and may be used to solve extremely complex systems. The solution is often quick and is easily adapted for parametric sensitivity studies. With the rapid advancement of computers, numerical methods have become an invaluable engineering tool. Some situations in which numerical methods would be the method of choice are: integration of a function for which either the integral cannot be expressed as analytical expressions or it is too cumbersome and time consuming to evaluate, solution of differential equations for complicated geometry and/or boundary conditions, large systems of equations, repeated solution of same systems under changing conditions, etc. Moreover, as an engineer, one is likely to use many software packages to perform numerical simulations, the use of which would probably be the most efficient means to solving engineering problems. One may be tempted to use these packages as *black-box*es but a meaningful use of these would require thorough understanding of underlying numerical methods. Sometimes, an engineer may have to develop his own software for numerical simulations due to the prohibitive cost, lack of flexibility, and poor computational efficiency of available packages. Even when the available packages are suitable for a particular problem, a working knowledge of numerical methods is valuable in case any difficulties or unreasonable results are encountered. We may also come across new mathematical models, which require a new software for solutions. The use of a numerical method is equally an art and a science (or shall we say, mathematics!). It is desirable to have the ability to select (and possibly modify) a numerical method for a specific problem.

We will try to illustrate the basic concepts with an example. Let's say *Joker* drops the wheel of *Bat-mobile* from the top of the *Vampire State Building* and *Batman* is looking out the window at the 13th floor, which is 666 m below the top. As the wheel passes by the window he reports to the police. How long after Joker drops the wheel the police are going to receive the call?

Transforming any practical problem into a mathematical model requires a series of assumptions such as

1. Assume the initial downward velocity to be zero (*reasonable assumption*).
2. Ignore all resistances due to air, wind, boundary layer around the building, etc. (*questionable*)
3. Assume that the acceleration due to gravity remains constant from the top of the building to the ground level (*reasonable*)
4. Assume the time taken between viewing the object and making the call to be zero. (*reasonable,* don't forget we are talking about Batman!)

5. We do not worry about what the Joker was doing at the top of the *Vampire State Building* with the wheel. (☺)

Using assumptions 2 and 3, the model for the problem can be formulated as a differential equation as follows:

$$\frac{d^2h}{dt^2} = g \tag{1.1}$$

where, *h* is the distance travelled in time *t* and *g* is the gravitational acceleration.

Integrating this equation and using assumptions 1 and 4, we obtain an algebraic equation as:

$$t = \sqrt{\frac{2h}{g}} \tag{1.2}$$

where t now represents the time when police receives the call. If *g* is taken as 9.81 m/sec^2, one can compute *t* (in sec) for $h=666$ m. Now, let us try to calculate this time, which is $\sqrt{2\times 666 / 9.81}$. After performing one multiplication and one division one arrives at approximately $\sqrt{135.77982}$. Since one cannot write or store (in a computer) infinitely many digits after the decimal, we need to *chop* or *round-off* this number.

We intend to apply a *Numerical Method* and use a computer to compute the square root of the rounded off number. Recall, a numerical method as well as a computer only does simple algebraic operations. How does one compute square root of a number (rational or irrational) by addition, subtraction, multiplication and division? We invoke our mathematical knowledge to transform the square root problem into a language that computer can understand, namely, $+$, $-$, $\times$ and $\div$.

It is possible to generate a sequence that converges to the square root of a number. One can then get arbitrarily close to the required value by progressing further and further in the sequence. For example, to compute the square root of a number *a* (*a*>0), one can write two different iterative sequences as follows:

Sequence 1: $$x_{n+1} = \frac{a}{x_n} \tag{1.3}$$

Sequence 2: $$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \tag{1.4}$$

It is easy to see, that *if the sequences converge*, their limits (*l*) will be equal to $\sqrt{a}$[1]. However, the primary concern is whether they converge? One is often interested to know that before one does a few iterations and discovers that the sequence is not converging. For example, for any initial guess $x_0 > 0$, *Sequence 1* oscillates between x_0 and a/x_0 and does not converge to $\sqrt{a}$. On the other hand, for *Sequence 2*:

[1] Put $x_{n+1} = x_n = l$.

For any initial guess $x_0 > 0$,

$2x_{n+1}x_n = x_n^2 + a \geq 2x_n\sqrt{a}$ since, $\left(x_n - \sqrt{a}\right)^2 \geq 0$. This means $x_{n+1} \geq \sqrt{a}$. So, the sequence is bounded.

Also, for n ≥ 2,

$x_n - x_{n+1} = \frac{1}{2}\left(\frac{x_n^2 - a}{x_n}\right) \geq 0$ since, $x_{n+1} \geq a$, $\forall n$ and $x_o > 0$. So, the sequence is decreasing.

This analysis shows that *Sequence 2* is a monotonically decreasing sequence that is bounded below (*Cauchy* sequence) and thus convergent. For example, to find the square root of 135.77982, we start with an initial guess of the square root as x_0=12 (we know that the square of 12 is 144, therefore the desired root should be close to 12). Subsequent iterations (with an eight digit accuracy) using Eq. (1.4) result in the values $x_1 = 11.657493, x_2 = 11.652461, x_3 = 11.652460, x_4 = 11.652460$. Thus the square root of 135.77982 is obtained in only 4 iterations as 11.652460. Assuming that we have no idea about the approximate answer, we may start from a very different value, say, x_0=1. It would take us 10 iterations to arrive at the root. Similarly, starting from an initial guess of 100, we would again converge to the correct answer in 10 iterations.

We have now seen two computational schemes (Eq. 1.3 and 1.4) for finding the square root of a number. Subsequent mathematical analysis showed that one scheme (Eq. 1.4) converges to the true solution whereas the other one (Eq. 1.3) does not. The computational scheme (Eq. 1.4) that converges to the true solution can be termed as a *Numerical Method* for computing the square root of a number. The mathematical analysis conducted to determine whether the computational scheme converges or not is called *Numerical Analysis.*

Using the numerical method described above, one can solve the algebraic form of the model for the problem of *Joker* dropping the wheel (Eq. 1.2). However, *Batman* being a very precise person wants to know the precision of your calculation. So, we need to know where the errors are and how to estimate them.

1.2 Errors

In any problem solving exercise one encounters errors in many forms and shapes. Some errors may add-up or some may cancel each other to give a net error in the final result. We will classify different forms of errors as follows:

1. Error in the Model or Model Error.
2. Error in the Data or Data Error.
3. Truncation Error.
4. Round-off Error.

Let's try to understand different forms of errors from the example cited in the previous section.

Mathematical model of a physical process is more often than not a *Spherical Cow*. One can minimize assumptions and make it a *Cylindrical Horse* at best but rarely, if ever, they represent the true physical process. This is because, the physical processes are often too complex or some of the processes cannot be characterized. For example, the assumptions 1 and 2 may not be valid for the problem of *Joker* dropping the wheel. Errors imparted in the final result due to these assumptions or approximations in the model formulation are termed as *Model Errors*.

The value of gravitational acceleration (g) was taken as 9.81 m/sec^2. The value of g at the site may be 9.80897653879 m/sec^2, which was approximated as 9.81 m/sec^2. Similarly, the distance between the window and rooftop was taken as 666 m. If one measures more accurately, it may be 665.99 m or 666.04 m. These errors in the data also lead to some error in the final result, which is known as *Data Error*.

Many of the processes of mathematics, such as differentiation, integration, and the evaluation of series, imply the use of a limit which is an infinite process. The machine has finite speed and can only do a finite number of operations in a finite length of time. This leads to a *Truncation Error* in the process. To illustrate truncation error, let us consider the Taylor's series expansion of a function $h(t)$ at $(t + \Delta t)$,

$$h(t+\Delta t) = h(t) + \Delta t\frac{dh}{dt} + \frac{\Delta t^2}{2!}\frac{d^2h}{dt^2} + \frac{\Delta t^3}{3!}\frac{d^3h}{dt^3} + \frac{\Delta t^4}{4!}\frac{d^4h}{dt^4} + \cdots\cdots \qquad (1.5)$$

Using this one can express $h(t)$ at $(t + 2\Delta t)$ and at $(t - \Delta t)$ as follows:

$$h(t+2\Delta t) = h(t) + (2\Delta t)\frac{dh}{dt} + \frac{(2\Delta t)^2}{2!}\frac{d^2h}{dt^2} + \frac{(2\Delta t)^3}{3!}\frac{d^3h}{dt^3} + \frac{(2\Delta t)^4}{4!}\frac{d^4h}{dt^4} + \cdots\cdots \qquad (1.6)$$

$$h(t-\Delta t) = h(t) - \Delta t\frac{dh}{dt} + \frac{\Delta t^2}{2!}\frac{d^2h}{dt^2} - \frac{\Delta t^3}{3!}\frac{d^3h}{dt^3} + \frac{\Delta t^4}{4!}\frac{d^4h}{dt^4} - \cdots\cdots \qquad (1.7)$$

Using various combinations of equations (1.5), (1.6) and (1.7) one can approximate the second derivative in the differential form of the model problem (Eq. 1.1) in different ways:

$$\frac{d^2h}{dt^2} = \frac{h(t+\Delta t) - 2h(t) + h(t-\Delta t)}{\Delta t^2} - \frac{\Delta t^2}{12}\frac{d^4h}{dt^4} - \cdots\cdots \qquad (1.8)$$

$$\frac{d^2h}{dt^2} = \frac{h(t+2\Delta t) - 2h(t+\Delta t) + h(t)}{\Delta t^2} - \Delta t\frac{d^3h}{dt^3} - \frac{7\Delta t^2}{12}\frac{d^4h}{dt^4}\cdots\cdots \qquad (1.9)$$

If we consider the first term on the right hand side as the approximation of the second derivative, we make some error. This error is due to approximation of an infinite series by a finite number of terms or in other words, due to truncation of the series. Error encountered in this way is termed as truncation error. As seen from these equations, the truncation error is proportional to $(\Delta t)^2$ in the first form (Eq. 1.8) and to Δt in the second (Eq. 1.9). For small Δt , therefore, the first form will have a much smaller error. Further discussion of the truncation error and how to solve the differential form of the model problem using these approximations of the second derivative will be discussed in a latter chapter. Here we illustrate the truncation error by using the example of the well-known infinite series for the sine function.

Example 1.1: The sine of an angle *x* is evaluated using an infinite series given by

$$\sin x = \sum_{n=1}^{\infty} \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!}$$

Show a plot of the truncation error over the interval (0,π) with the series truncated after 1, 2, 3, 5, 10, and 100 terms.

Solution:

The following plot shows the computed values obtained using 1, 2, 3, and 5 terms of the series (beyond 5 terms the curves are not shown since they plot on the same line):

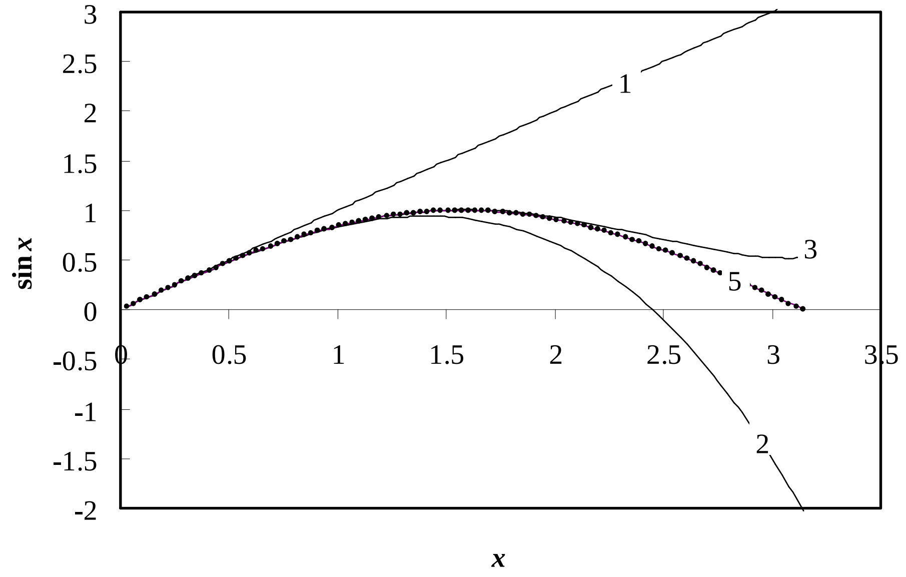

Figure 1.1 The sine series truncated after a few terms. Symbols show the exact values and the lines show the computed values using the number of terms mentioned on the curves.

The following plot shows the error in the computed values obtained using 1, 2, 3, 5, and 10 terms of the series (for 100 terms the error was negligible). The error, e_r, is the error relative to the true value of sin *x* (naturally, the points x=0 and π have been excluded).

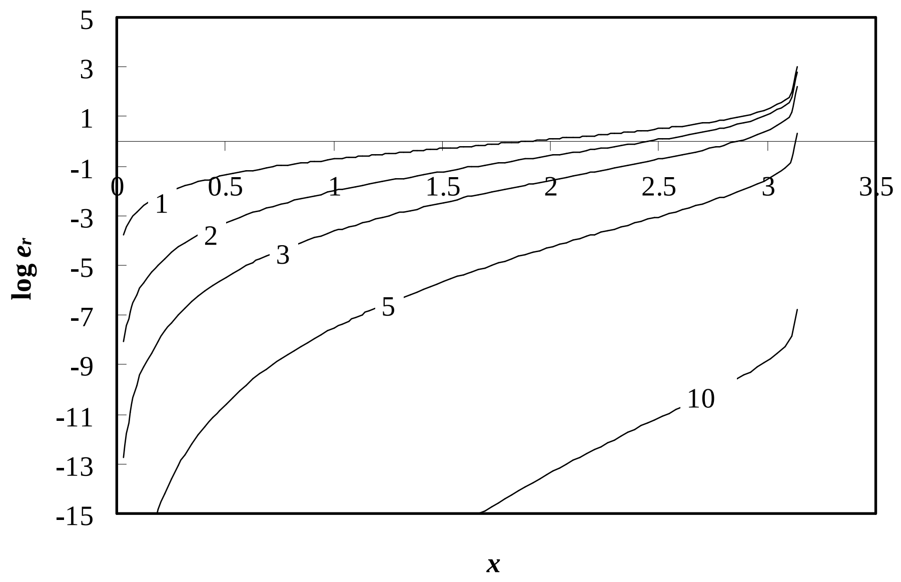

Figure 1.2 The truncation error in the sine series computed with a few terms. The lines show the log of the relative error.

It is apparent that the error is very small for small x but increases significantly as x approaches π. However, even with only 10 terms, the error near x=π is quite small (about 10^{-7}).

Since all computing machines have finite word length, the numbers are always rounded off after a few significant digits following decimal. The number of significant digits is often dependent on the machine and precision one uses to perform the computations. For example, if one uses a precision of four significant digits, time for the phone call in the model problem will be $\sqrt{2\times 666 / 9.81} = \sqrt{135.8}$. But the same problem with double precision calculation will result in $\sqrt{135.77982}$. Now, if one uses the *Sequence 2* to compute the square root, the number will be rounded off in each iteration of the sequence. The error in the final result due to these rounding off is termed as Round-off Error.

A floating point number is generally represented in a computer as $m \cdot b^p$ where, *b* is the base of the number system, e.g. 2 (binary), 10 (decimal), 16 (hexadecimal) etc., m is the mantissa given by $1/b \le m < 1$, and *p* is the power which is an integer. Let's illustrate the round-off error by using decimal system[1]:

[1] Computers use the binary system but humans understand the decimal system much better. Therefore, the examples used here assume the numbers to be stored in the decimal system.

A real number (a) in a decimal system is represented as $a = m_a \times 10^p$ where, $0.1 \le m_a < 1$. If the number is irrational and we try to store this number in a computer, m_a will be rounded[1] after some significant digits following decimal. Let's assume that our machine can store a maximum of t places after decimal *i.e.*, the number is rounded off after t decimal places[2]. We denote the rounded-off mantissa as $\tilde{m}_a$ and the corresponding approximate real number as $\tilde{a} = \tilde{m}_a \times 10^p$.

The absolute error in the mantissa due to round off can be written as,

$$|m_a - \tilde{m}_a| \le 0.5 \times 10^{-t}$$

The relative error in real number a due to round-off is given by (note that $0.1 \le m_a < 1$)

$$\left|\frac{a - \tilde{a}}{a}\right| \le \frac{0.5 \times 10^{-t} \times 10^p}{m_a \times 10^p} \le 0.5 \times 10^{1-t} = u \text{ (say)} \qquad (1.10)$$

Since, t will vary depending on the machine one uses and the precision of calculation (single *vs.* double), u is call the *machine precision unit* or simply *round-off unit*. For a few isolated computations, round-off error would be very small. However, most practical computations involve a series of steps with each step using the computed values from the previous one. This may cause an unacceptable accumulation of errors in the final result. To analyze this behaviour, we look at the propagation of error and the related concept of condition of problems and stability of algorithms.

1.3 Error Propagation and Condition Number of a problem

In the previous section we saw that the errors in variables are introduced by several means, e.g. data error, round-off error etc. One would then be interested in estimating the error introduced in the final computation of the functional value as a result of these errors in the variables. In other words, how the errors in variables propagate into the function, whether they tend to grow or decay as the computations proceed. For example, in surveying, if one has measured two sides of a triangle and the included angle, and computes the area of the triangle, what is the likely error in the area due to measurement errors in the sides and the angle? In this section, we will derive an approximate expression that will enable us to calculate the error in the functional value if one has the knowledge of the function and the error in the independent variables.

[1] Instead of rounding (i.e., incrementing the last digit by 1 if the next digit is 5 or larger, keeping it same otherwise), one could also use chopping (not changing the last digit irrespective of the next digit). However, rounding is more accurate, though it requires storing a few additional digits.

[2] The intermediate computations are generally carried out using a *few* (typically 2 or 3) more significant digits than t and the final result is rounded off to t digits.

Let y be a function of n independent variables, $x_1, x_2, \ldots, x_n$, which are written compactly as a vector X. To find the error in y as a result of errors in the data, ΔX, we write $y = f(X)$ and the approximate vector, $\tilde{X} = X - \Delta X$, in which $\Delta x_i = x_i - \tilde{x}_i$.

Due to this error in the variables, approximate functional value is given by $\tilde{y} = f(\tilde{X})$ while the true functional value is $y = f(X)$. So, the error in the functional value is expressed as:

$$\Delta y = y - \tilde{y} = f(X) - f(\tilde{X}) \tag{1.11}$$

Since $\Delta X = X - \tilde{X}$ i.e., $X = \tilde{X} + \Delta X$, we use the multivariate Taylor's series and write

$$f(X) = f(\tilde{X} + \Delta X) = f(\tilde{X}) + \sum_{i=1}^{n} \Delta x_i \left.\frac{\partial f}{\partial x_i}\right|_{\tilde{x}_i} + \cdots\cdots \text{(Higher Order Terms)} \tag{1.12}$$

The errors are likely to be small and the *Higher Order Terms* consisting of errors raised to the power 2 or more can, therefore, be neglected. Then, from (1.11) and (1.12), we have

$$\Delta y = \sum_{i=1}^{n} \Delta x_i \left.\frac{\partial f}{\partial x_i}\right|_{\tilde{x}_i} \tag{1.13}$$

Following example illustrates the use of equation (1.13).

Example 1.2: The gravitational acceleration is computed by observing the time, t, it takes for an object to fall a certain height, h. The height measurement is accurate to a cm and the time measurement to a 100th of a second. What error is expected in the computed value of g, if the measured height is 666.00 m and the time 11.65 s?

Solution:

Accounting for the error, the actual values of h and t may be written as 666.00±0.01 m, and 11.65±0.01 s, respectively. Writing $g = f(h,t) = \frac{2h}{t^2}$, we have $\frac{\partial f}{\partial h} = \frac{2}{t^2}$ and $\frac{\partial f}{\partial t} = -\frac{4h}{t^3}$. From Eq. (1.13), therefore,

$$\Delta g = \frac{2}{\tilde{t}^2}\Delta h - \frac{4\tilde{h}}{\tilde{t}^3}\Delta t = 0.014736\Delta h - 1.6848\Delta t$$

Clearly, the errors due to error in h and t may offset each other if the errors in both h and t are of the same sign (e.g., if we have made positive errors and underestimated both the distance and time). However, we would consider the worst case scenario of making the maximum error (0.01 m and 0.01 s) in both h and t and with opposite signs and obtain an upper bound on the magnitude of the error in g as $|\Delta g|$=0.016995 m/s^2. In other words, we will put the modulus sign on all the terms in Eq. (1.13).

The error is generally[1] written as

True Error (e) = True Value – Approximate Value

Note that this definition of error is contingent on the “true value” being known. In most cases where we apply the numerical method, the true value is not known and we seek an approximate value which is *close* to the true value. Typically it is done by generating a sequence (see Eq. 1.4) which approaches the true value. Therefore, it is customary to define an approximate error as

Approximate Error (ε) = Current Approximation – Previous Approximation

In Example 1.2, both *h* and *t* had a maximum measurement error of 0.01 (m and s, respectively). However, since the magnitude of *h* was very large compared to *t*, the error in *t* is much more detrimental to the accuracy of the computed *g*. The error in *g* was obtained as about 0.017 m/s^2. Whether this is acceptable or not will depend on the expected value of g. For example, if g is close to 10 m/s^2, the error may be acceptable. But if g is, say, about 0.1 m/s^2, the error is not acceptable. Therefore, instead of the *absolute error* in the functional value and the variables, it may be better to define a *relative error* as

True Relative Error (e_r) = (True Value – Approximate Value) / True Value

and

Approximate Relative Error (ε_r) = (Current Approximation – Previous Approximation) / Current Approximation

One could then ask whether the *relative* error in the variables will be magnified or reduced in the functional value following computations. In other words, if we make a small relative error in the variable, how much relative error in the function would it cause? This sensitivity of the function to small errors in the variables is defined through a *Condition Number* (C_P) of a problem as shown below.

To describe the relative error in *f*(*x*) for a small relative error in *x*, we define the condition number as the ratio of the relative change in the functional, *f(x)*, to the relative change in the variable for a small perturbation Δ*x* in the variable *x*. Therefore

$$C_P = \frac{\left|\dfrac{f(x+\Delta x)-f(x)}{f(x)}\right|}{\left|\dfrac{\Delta x}{x}\right|} \qquad (1.14)$$

For arbitrarily small Δ*x* or as $\Delta x \to 0$, the above expression reduces to,

[1] Sometimes the error is written as (approximate value – true value). We will, however, follow the more common representation.

$$C_P = \left|\frac{xf'(x)}{f(x)}\right| \tag{1.15}$$

It is easy to see from the definition that, if $C_P = 1$, the relative error in the functional value remains same as that in the variable. When $C_P > 1$, the error is magnified in the function and the problem is called *ill conditioned*. When $C_P < 1$, the error is attenuated and the problem is *well conditioned* (depending on the type of problem, sometimes ill-conditioning is assumed to occur when C_P is larger than a specified number greater than 1. For example, we may accept relative errors of, say, 5%, in the function for a 1% error in the variable). The condition number of a problem is illustrated in the following example.

Example 1.3: The equation for a hyperbola is given as $xy-y+1=0$. The abscissa is measured at some locations and the ordinate is computed as $y=1/(1-x)$. It is known that the measurement of x involves some error. Find the condition number of the problem (computing y for a measured x). For what values of x would it be well-conditioned?

Solution:

Using Eq. (1.15), $C_P = \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x/(1-x)^2}{1/(1-x)}\right| = \left|\frac{x}{1-x}\right|$. For the problem to be well conditioned, therefore, x should be less than 0.5. At x=0.5, y=2, and if we change x by 1% to 0.505, we get y=2.0202, a relative change of about 1%. This indicates a condition number of 1 (which is also obtained from the expression above). Similarly, for x=0.98, y=50, and if we change x by 1% to 0.9898, we get y=98.04, a relative change of about 96% (the condition number from the equation comes out to be 49 at x=0.98 and 97 at x=0.9898).

If the problem is ill-conditioned, there is not much we can do about it except to use higher precision in the computations. On the other hand, a well-conditioned problem can often be solved using more than one *Numerical Method* or, in any *Numerical Method*, by more than one algorithm. Each algorithm will have a specific set of floating point operations that will lead to round-off errors. For the same problem, one can have different round-off errors by using different algorithms. So, even for a well-conditioned problem, the errors may be very different for different algorithms, as shown below.

Example 1.4: A well-conditioned problem is to find the positive root of the equation $y^2+2y-x=0$ for x close to zero. The evaluation of $y=\sqrt{1+x}-1$ has a condition number equal to 1 for x approaching zero. Let the computation be done by an algorithm which uses the following three steps: (a) compute 1+x (b) perform the square root operation on the result of (a), and (c) subtract 1 from the result of (b). Find the condition number of each step.

Solution:

The condition number of step (a) is $\left|\dfrac{x\times 1}{1+x}\right|_{x\to 0}=0$ and the result (x_a) is close to 1 as x approaches zero. The condition number of step (b) is $\left|\dfrac{x_a\times\dfrac{1}{2\sqrt{x_a}}}{\sqrt{x_a}}\right|_{x_a\to 1}=\dfrac{1}{2}$ and the result (x_b) is close to 1 as x_a approaches 1. The condition number of step (c) is $\left|\dfrac{x_b\times 1}{x_b-1}\right|_{x_b\to 1}=\infty$ and its result (y) is close to zero as x_b approaches 1. Due to this ill-conditioned step, the algorithm will result in large relative error in y computed with small errors in x, and the above algorithm is *unstable*. Since the problem is well-conditioned, we should be able to devise an algorithm which would be stable, i.e., all steps would be well-conditioned. The reader should verify that the equivalent algorithm given by $y=\dfrac{x}{\sqrt{1+x}+1}$ is stable.

1.4 Condition Number of an algorithm

One can also define a condition number for an algorithm, which takes into account propagation of errors through a set of numerical computation in an algorithm. This is done through a *Backward Error Analysis,* which determines the change necessary in the input variable (or data) to explain the final error in the output (or result) obtained through errors introduced at various stages of calculation. A measure of this change in the input data required to explain the errors in the output result is termed the condition number of the algorithm (C_A). Denoting the *exact* input data by x and the corresponding function value by y, let us assume that an algorithm A operating on x produces the result y_A on a machine with precision unit u (see Eq. 1.10). We look for the input data, x_A, which would result in the exact function value, y, if computations are done with infinite precision. The condition number of the algorithm is then defined through

$$\frac{|x-x_A|}{|x|}\le C_A u \qquad (1.16)$$

For example, a multiplication between two real numbers x = 0.12 and y = 0.1554 would yield 0.018648. We assume that x is not subject to round-off errors and is exactly known. If we want to carry out a floating point operation ($x\times y$) with precision of four decimal places (u = 0.5×10^{-3}), the result, generally written as, $fl(x\times y)$, is 0.1865×10^{-1}. However, we know that the true result $(x\times y)_T$ is 0.18648×10^{-1}. We can obtain the relative error in this computation as follows:

$$\frac{|fl(x\times y)-(x\times y)_T|}{|(x\times y)_T|}=0.10725\times10^{-3}\le u \qquad (1.17)$$

As pointed out in the previous equation, u is an upper bound for the relative error. If we did a multiplication between $x = 0.12$ and $y = 0.1554167$ with infinite precision, we would obtain the same result (0.01865) as we did using floating point operation. If we denote the new value of y required to explain the final result as $y(1+\delta)$, then one can write,

$$fl(x\times y) = x\times y(1+\delta) \tag{1.18}$$

So, the final error in the result is equivalent to introducing an error in y. In this case, $\delta = 0.10725\times10^{-3}$. By combining equations (1.17) and (1.18), it is easily seen that u is an upper bound for δ as well, i.e., $|\delta| \le u$.

This process of estimating perturbation (δ) required in the input variable (y) to explain the final result is called *Backward Error Analysis*. An obvious advantage of using the backward error analysis is that we do not need to know the true value of the result, which was needed in a forward error analysis and which is generally not known. The illustration above was for one operation only. It is easy to imagine that, in an algorithm, a series of floating point operation will be required to obtain the final result. In the process, the round-off error will accumulate or may sometimes get cancelled. In the worst case of accumulating round-off error, a larger perturbation will be required in the input variable to explain the final error. This required perturbation in the input data is expressed as $C_A u$, where, C_A is the condition number of the algorithm. For the multiplication problem, therefore, the condition number of the algorithm is equal to 1, as u is the upper bound for the relative change in the input variable. On the other hand, if we consider the algorithm $f(x) = \sqrt{1+x^2} - 1$, and evaluate the function using four significant digits (without using additional buffer digits) for x=0.4980×10^{-1}, we get a value of 0.1000×10^{-2}. This function value corresponds to an exact computation with x_A=0.447325×10^{-1}, a relative error of 0.10176. With a u value of 0.5×10^{-3}, the condition number C_A is obtained as 204, indicating the instability of the algorithm.

One should note that we have used the modulus in Eqs. (1.14 – 1.16) indicating that there is a single input variable, x. If there are several input variables subjected to round-off errors, we would replace the modulus with some indicator of the size of the vector, X. Although we assume reader familiarity with vector algebra, we briefly mention here various vector norms, which could be used to represent the *magnitude* of a vector. These would also be useful in Chapter 3 when we discuss the matrix norms.

The L_p norm of an n-dimensional vector, X, *i.e.*, $(x_1, x_2, \ldots, x_n)$, is given by:

$$\|X\|_p = \left(|x_1|^p + |x_2|^p + |x_3|^p + \ldots + |x_n|^p\right)^{1/p} \qquad p \ge 1 \tag{1.19}$$

The *Euclidean norm*, representing the *length* of the vector in the n-dimensional space is obtained with p=2 and is, therefore, called the L_2 norm. The L_∞ norm or the maximum *norm*, representing the maximum *distance* travelled along any direction (i.e, $\max_i |x_i|$) is obtained as p$\rightarrow \infty$. And the L$_1$ norm represents the total distance travelled *along the individual directions*, i.e., $\sum_{i=1}^{n}|x_i|$. Some properties of the vector norms[1] are (α is a scalar):

$$\begin{aligned}
&\|X\| = 0 \text{ only if } X \text{ is a null vector; otherwise } \|X\| > 0\\
&\|\alpha X\| = |\alpha|\|X\| \qquad\qquad (1.20)\\
&\|X_1 + X_2\| \le \|X_1\| + \|X_2\|
\end{aligned}$$

The above discussion of condition number of a problem and the condition number of an algorithm, combined with the norm of a vector, allows us to write the error in the *actual computation* of a function as follows:

Let *X* and *Y* represent the exact input and output vectors, respectively, of the function *f(X)*. Let $\tilde{X}$ represent the actual input and let Y_A be the result of an algorithm *A* to evaluate the function, *f(X)*, applied to this input on a machine with precision unit *u*. Also, let X_A be an input which would result in the same output, Y_A, if computations are performed with infinite precision. The error in the output is then $Y–Y_A$, and the relative error could be written as

$$\begin{aligned}
\frac{\|Y - Y_A\|}{\|Y\|} = \frac{\|f(X) - f_A(\tilde{X})\|}{\|f(X)\|} &\le \frac{\|f(X) - f(\tilde{X})\|}{\|f(X)\|} + \frac{\|f(\tilde{X}) - f_A(\tilde{X})\|}{\|f(X)\|}\\
&\square \frac{\|f(X) - f(\tilde{X})\|}{\|f(X)\|} + \frac{\|f(\tilde{X}) - f_A(\tilde{X})\|}{\|f(\tilde{X})\|}\\
&= \frac{\|f(X) - f(\tilde{X})\|}{\|f(X)\|} + \frac{\|f(\tilde{X}) - f(X_A)\|}{\|f(\tilde{X})\|}\\
&\le C_P \frac{\|X - \tilde{X}\|}{\|X\|} + \tilde{C}_P \frac{\|\tilde{X} - X_A\|}{\|\tilde{X}\|}\\
&\le C_P\left(r + C_A u\right) \qquad\qquad (1.21)
\end{aligned}$$

where, *r* is the relative error in the input and $\tilde{C}_P$ represents the condition number of the problem of evaluating *f(X)* at $\tilde{X}$, which is approximately equal to C_P at *X*. Note that we have used the properties of the norm, the definition of the condition number of a problem, and the definition of the condition number of an algorithm to arrive at the final result for the upper bound of the error in an actual computation.

Example 1.5: The computation of $y = f(x) = 2x$ is to be performed for x=0.045676 on a computer using 4 significant digits. What would be an upper bound on the relative error?

Solution:

[1] True for all L_p norms.

Since we use only 4 significant digits, the exact value of x would be rounded off to $\tilde{x} = 0.4568 \times 10^{-1}$, a relative error in the input, r, of 8.7573×10^{-5}. The condition number of the problem is 1, using Eq. (1.15). Condition number of the algorithm for floating point multiplication was obtained earlier as 1. Hence the upper bound on relative error would be $(r+u) = 5.8757 \times 10^{-4}$.

While the description of different errors and their analysis is interesting and provides us an overview of possible errors in computations and their propagation, from a practical viewpoint, we are more interested in the errors which are in our control while developing or applying a numerical method. The round-off errors could often be minimized by writing the algorithm in a slightly different form and the truncation errors could be reduced by using more steps in the computation. However, generally the reduction of error comes at the cost of more computation time and proper application of a numerical method boils down to an optimum balance between the error and effort. The discussion of various topics in the subsequent chapters of this book, therefore, stresses these two aspects. For example, when acceptable errors in the result are specified, we would look to minimize the computational effort to achieve the desired accuracy. And when different methods with similar computational efforts are compared, we would look to minimize the errors.

Exercise 1.0

1. An instrument gives result y in terms of the independent variable x according to $y(x) = x + \frac{a^3}{x^2}$ where, a = 6.8704294974 is an instrument constant. We need to compute the difference $\Delta y = y(x_1) - y(x_2)$ for two subsequent measurements with x_1 = 8.5834541395 and x_2 = 8.7282534483. Compute Δy from the definition of y using floating point operations rounding mantissa to 5 decimals (5-digit precision). Use double precision calculation to obtain a more accurate estimate and treat this as true solution. Estimate the relative error.

2. In problem 1, express Δy in terms of products and ratios involving a, x_1 and x_2. Recompute using the same 5-digit precision and estimate the relative error. Comment on the differences observed.

3. It is required to compute the roots of the quadratic equation $x^2 - x - 2\varepsilon^2 = 0$ for ε = 0.001. Perform the computation with 5-digit precision and estimate the corresponding relative error. performing operations by rounding all mantissas to six decimals. Use double precision calculation to obtain the true solution that is needed to calculate the relative errors.

4. Express the roots by employing a Taylor series expansion. Now compute taking only 2 terms of the Taylor series and estimate the corresponding relative error.

5. On a triangular plot of land, two sides were measured as a = 100.0 ± 0.1 m and b = 101.0 ± 0.1 m. The included angle between the two measured sides was estimated using a

Theodolite as $C = 58.00^o \pm 0.01^o$. How accurately is it possible to estimate the third side c ? What is the range of error in the estimation of the area of the plot ?

6. What is meant by the condition number of a problem? The following set of equations is solved to get the value of x for a given ε. For what values of ε will this problem be well-conditioned?

$$x + y = 2$$
$$x + (1-\varepsilon)y = 1$$

1.5 Outline of the book

In the next chapter, we look at the methods of solving a set of linear equations because these are quite frequently encountered during application of several numerical methods. We look at the conditions under which a solution exists, and discuss various techniques of solution, broadly classified as direct or iterative methods, depending on whether we get the solution in a definite number of operations or whether we keep on going till the desired accuracy is achieved. The aspects of ill-conditioned system of equations are discussed which lead to a description of eigenvalues and the methods to find these. Chapter 3 starts with the solution of nonlinear equations in a single variable. These are also called root-finding problems since the given nonlinear equation could be written as f(x)=0 and the solution of the equation amounts to finding the zeroes of *f(x)*. Various methods, their errors, and the rate of convergence are described. The methods have been broadly classified in bracketing and open methods depending on whether, to start the process, we need to obtain a closed interval containing the root or not. The extension of these methods to nonlinear equations involving multiple variables is also discussed. The next two chapters (4 and 5) describe the approximations of functions (given either as a continuous function or in discrete form as a set of values), their derivatives, and integrals. Chapter 4 describes various techniques of interpolation (when the approximate function passes through all the given data points) and regression (where the approximation captures the general trend of the function or data), and the errors associated with these. Numerical differentiation and integration is discussed in Chapter 5 for both continuous and discrete cases. Chapters 6 and 7 discuss various techniques of numerically solving differential equations, starting with the easier problem of ordinary differential equations (ODE) in Chapter 6, and moving on to partial differential equations (PDE) in Chapter 7. Both explicit and implicit methods have been described for solution of ODE's. The classification of PDE's into elliptic, parabolic, and hyperbolic, and the solution techniques for each have been discussed. Error and stability analysis for the differential equations is emphasized. Finally, in Chapter 8, some advanced topics are briefly covered to introduce the reader to techniques used for solving differential equations, which are generally not mentioned in an introductory course on numerical analysis. The treatment of these topics is, by choice, rather superficial and the interested reader is directed to appropriate resources for further details.