

Contents

Chapter 1 Numerical Methods, Number Systems and Error Analysis	2
1.1 Computational Problems and Algorithms	2
Motivation towards need of Numerical/Computational Methods	2
1.1.1 What advantage do we gain by employing numerical method of solution?	4
1.1.2 The Error Problem	6
1.2 Significant Digits	7
1.3 1.4 Number Representation and Storage in Computers	8
Representation of Integers	8
1.4.1 Decimal to Binary Integer	9
1.4.2 Conversion	9
Algorithm to convert Decimal Integer to Binary Integer	9
1.4.3 Binary to Decimal Integer	9
1.4.4 Conversion	10
Algorithm to convert Binary Integer to Decimal Integer	10
1.4.5 Representation of Fractions	10
1.4.6 Decimal to Binary Floating-Point	10
1.4.7 Conversion	10
Algorithm to convert Decimal Fraction to Binary Fraction	10
1.4.8 Binary to Decimal Integer	11
1.4.9 Conversion	11
1.4.1 Algorithm to convert Binary Fraction to Decimal Fraction	11
0 Numerical Errors	11
1.5 Round-Off Errors	11
Numerical Cancellations	12
1.5.2 Truncation Errors	13
1.5.3	13

	<u>1.6 Computational Methods for Error</u>	
	<u>Estimation</u>	<u>13</u>
	<u>Computational</u>	
<u>1.6.1</u>	<u>Methods</u>	<u>13</u>
	<u>Computational</u>	
<u>1.6.2</u>	<u>Efficiency</u>	<u>13</u>
	<u>True</u>	
<u>1.6.3</u>	<u>Error</u>	<u>14</u>
	<u>Relative True</u>	
<u>1.6.4</u>	<u>Error</u>	<u>15</u>
	<u>Relative Approximate</u>	
<u>1.6.5</u>	<u>Error</u>	<u>15</u>
<u>1.6.6</u>	<u>Tolerance</u>	<u>16</u>
	<u>Accuracy of</u>	
<u>1.6.7</u>	<u>Solution</u>	<u>16</u>
	<u>Error</u>	
<u>1.6.8</u>	<u>Estimation</u>	<u>17</u>

Chapter 1**Numerical Methods, Number Systems and Error Analysis****1.1 Computational Problems and Algorithms****1.1.1 Motivation towards need of Numerical/Computational Methods**

To solve many engineering and scientific problems we start the solution by, first, representing the problem at hand into an equivalent suitable (and usually simplified) mathematical model. These mathematical models are often derived from, either (a) engineering or science principles, or, otherwise (b) obtained from experimental data.

Once modelled mathematically the solutions generally result in need of using mathematical procedures that include but are not limited to:

- 1) nonlinear equations,
- 2) simultaneous linear equations,
- 3) differentiation,
- 4) integration,
- 5) curve fitting by interpolation or regression, and
- 6) differential equations

A mathematical model can be broadly defined as a formulation or equation that expresses the essential features of a physical system or process in mathematical terms. In a very general sense, it can be represented as a functional relationship of the form

$$y = f(x_1, x_2, \dots, x_n; p_1, p_2, \dots, p_m; F_1, F_2, \dots, F_k) \quad \text{Eq.1.1}$$

where the *dependent variable* is a characteristic that usually reflects the behavior or state of the system; the *independent variables* are usually dimensions, such as time and space, along which the system's behavior is being determined; the *parameters* are reflective of the system's properties or composition; and the *forcing functions* are external influences acting upon the system.

Example 1.1: We know that “Rate of Change of Momentum of a body with respect to time” is equal to the resultant Force on the body:

$$\frac{d}{dt}(mv) = F \quad \text{Eq.1.2}$$

where F = net force acting on the body (N, or $\text{kg} \cdot \text{m} \cdot \text{s}^{-2}$), m = mass of the object (kg), and a = its acceleration ($\text{m} \cdot \text{s}^{-2}$).

Rearranging the terms,

$$m \frac{dv}{dt} = F \quad \text{Eq.1.3}$$

Observe that Eq.1.3 is arranged in the form of Eq.1.1, where a is the dependent variable, m is a parameter of system (i.e. property or characteristic of the system), and F is a forcing function (In this simple model we did not account for any independent variable). Hence, through this equation we have mathematically modeled a system of moving body to account for its *rate of change of momentum*.

Example 1.2: Let us take another more realistic example. Let us model the system of a free falling parachutist.

Our aim, here, is to determine the velocity of parachutist at a given point of time.

During a free fall, the parachutist starts falling with a (downward) acceleration, a , attributed to the (force of) gravity, $= mg$, acting on his body mass, m (g being the *acceleration due to gravity*). This can be mathematically modeled just as done in Example 1.1:

$$m \frac{dv}{dt} = mg \quad \text{Eq.1.4}$$

Through this mathematical equation, we are saying that, the *rate of change in velocity*, $\frac{dv}{dt}$, of the parachutist is equal to the ratio of gravitational force (acting on him), mg , and (his body) mass, m .

Eq.1.4 would hold true for the ideal case of a free falling body in vacuum. But, a moving body in a fluid (which is “air”, in this example) experiences resistance to movement, and this (force of) resistance, F_r , increases proportional with the velocity, v , of the body:

$$F_r = -cv \quad \text{Eq.1.5}$$

Here, c is the known as *coefficient of drag* (which is a parameter of this system).

As the parachutist keeps falling there are two (opposing) forces acting on his body; the downward force of gravity, mg , and the opposing (upward) force of resistance, $-cv$. So, the net force acting on parachutist's body is $mg - cv$.

Initially, the body gains velocity (downwards) because of small value of (remember, *air resistance*, F_r , is proportional to the velocity, and, as such, resistance will be low, initially, corresponding to small initial velocity), but as the body attains more and more velocity (downwards) the *air resistance* also increases proportionally, which ultimately leads to being equal to mg , at which, acceleration ceases and body continues to fall at a constant velocity, known as *terminal velocity*.

Hence, after employing the contributing factor of air resistance, at any point of time, the *rate of change of velocity* of the parachutist can be modeled, mathematically, as:

$$m \frac{dv}{dt} = mg - cv \quad \text{Eq.1.6}$$

which is a differential equation. To find the velocity of parachutist, at any point, we can integrate Eq.1.6 to obtain:

$$v = \frac{mg}{c} (1 - e^{-\frac{c}{m}t}) \quad \text{Eq.1.7}$$

Using Eq.1.7 we can tabulate velocity of parachutist of mass 68.1 kg at various points of time, with a *coefficient of air drag* as $12.5 \text{ kg} \cdot \text{s}^{-1}$:

Table 1.1: Velocity of parachutist (analytical)

Time, t (in seconds)	Velocity, v (in $\text{m} \cdot \text{s}^{-1}$)
0	0.00
2	16.40
4	27.77
6	35.64
8	41.10
10	44.87
12	47.49
∞	53.39

Hence, we were able to obtain our solution analytically.

Example 1.3: Let us solve the same problem of Example 1.2 with a slightly different approach, numerical method, which would also help us to appreciate the advantages of numerical methods over analytical approaches.

The *rate of change in velocity* can be approximated as:

$$\frac{dv}{dt} = \lim_{\Delta t \rightarrow 0} \frac{\Delta v}{\Delta t} \quad \text{Eq.1.8}$$

which results in:

$$\frac{dv}{dt} = \frac{v(t) - v(t-\Delta t)}{\Delta t} \quad \text{Eq.1.9}$$

Substituting Eq.1.8 and Eq.1.9 in Eq.1.6 yields:

$$\frac{v(t) - v(t-\Delta t)}{\Delta t} = -g + \frac{v(t-\Delta t)}{\tau} \Rightarrow v(t) = v(t-\Delta t) + \Delta t \left(-g + \frac{v(t-\Delta t)}{\tau} \right) \quad \text{Eq.1.10}$$

Eq.1.10 is the alternative solution or substitute for Eq.1.7 of Example 1.2. This is a ***numerical method*** of solution for the given problem.

1.1.2 What advantage do we gain by employing numerical method of solution?

- 1) Observe that Eq.1.10 is a recursive formula, which allows us to find the velocity of parachutist at any given point of time using his velocity at previous point of time.
- 2) Eq.1.10 does not involve any complicated calculations, but rather, very simple arithmetic calculations. This is usually the case with numerical methods or numerical solutions.
- 3) This was a very simplified model of the system, whereas, more realistic precise modeling may involve many other factors such as the current shape (or posture) of the parachutist, material of clothing, the spin and turns of parachutist, the change in density of air as he falls, the (micro) change in gravity (to be absolutely precise), etc. (Even the air resistance is not truly linearly proportional to the velocity, as assumed, but quadratically proportional). When all the affecting external factors are employed the

analytical solution becomes extremely complicated, and even the derivation of analytical solution itself may be complicated. Whereas, the equivalent numerical model involves much simplified and easier mathematical (usually confined to basic arithmetic) operations.

- 4) Observe that implementing the numerical solution on computers (using programming or software tools) is highly feasible (because performing millions of simple arithmetic calculations on modern computers in shortest span of time is inherently very much possible), whereas this may not be the case with analytical solutions (because even the fastest computers cannot perform any more complicated operations than simple arithmetic, although such functionality as logarithms, exponentials, integrals, and similar operations, may be provided which internally involves a complicated series of equivalent simple arithmetic).
- 5) Scientific and engineering problems sometimes do not reveal any well-defined relationship between the dependent variable and the external forcing factors. In such cases, analytical solution cannot be derived. But, using repeated experimental data, it is possible to develop close to accurate relationships (between the variables of system and external factors) using numerical methods.
- 6) Analytical methods always give us exact solutions (with in the domain of assumptions) to our problems. But, many a times, it suffices us if we can find a solution up to a given precision. In such situations the numerical models are usually much simpler compared to the equivalent analytical solutions. This may become clearer once we calculate the velocity of parachutist at various points of time using Eq.1.10 and

compare them to the values given in **Table 1.1**:

Table 1.2: Velocity of parachutist (numerical)

Time, t (in seconds)	Velocity, v (in m/s)
0	0.00
2	19.60
4	32.00
6	39.85
8	44.82
10	47.97
12	49.96
∞	53.39

Figure 1.1 compares the values in **Table 1.1** and **Table 1.2**.

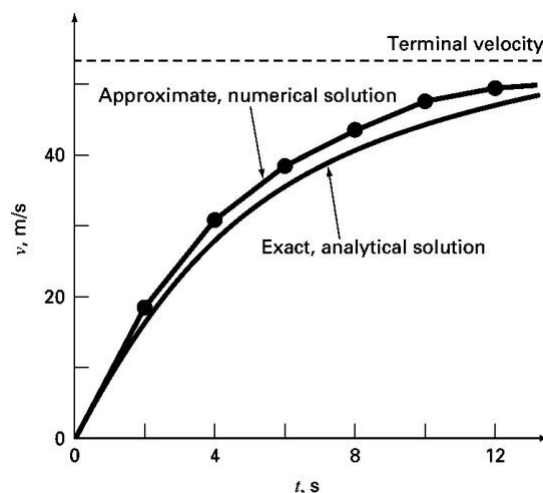


Figure 1.1: Comparison - Analytical and Numerical Solutions of Examples 1.2 and 1.3

Observe that accuracy of numerical results can be increased by reducing the time interval. For example, **Table 1.3** and **Table 1.4** calculates the velocity of parachutist with a time interval of 0.2 seconds and 0.05 seconds, respectively:

Table 1.3: Velocity of parachutist (interval=0.2s) **Table 1.4: Velocity of parachutist (interval=0.05s)**

Time, t (in seconds)	Velocity, v (in m s^{-1})
0	0.00
0.20	1.96
0.40	3.85
0.60	5.67
0.80	7.43
1.00	9.12
1.20	10.74
1.40	12.31
1.60	13.82
1.80	15.28
2	16.68
2.20	18.03
2.40	19.33
2.60	20.58
2.80	21.79
3.00	22.95
3.20	24.07
3.40	25.15
3.60	26.18
3.80	27.19
4	28.15
4.20	29.08
4.40	29.97
4.60	30.83
4.80	31.66
5.00	32.46
5.20	33.23
5.40	33.98
5.60	34.69
5.80	35.38
6	36.04
6.20	36.68
6.40	37.30
6.60	37.89
6.80	38.46
7.00	39.01
7.20	39.54
7.40	40.05
7.60	40.54
7.80	41.02
8	41.47

Time, t (in seconds)	Velocity, v (in m s^{-1})
0	0.00
0.05	0.49
0.10	0.98
0.15	1.46
0.20	1.94
0.25	2.41
0.30	2.88
0.35	3.34
0.40	3.80
0.45	4.26
0.50	4.71
0.55	5.15
0.60	5.60
0.65	6.04
0.70	6.47
0.75	6.90
0.80	7.33
0.85	7.75
0.90	8.17
0.95	8.59
1.00	9.00
1.05	9.41
1.10	9.81
1.15	10.21
1.20	10.61
1.25	11.00
1.30	11.39
1.35	11.78
1.40	12.16
1.45	12.54
1.50	12.91
1.55	13.29
1.60	13.66
1.65	14.02
1.70	14.38
1.75	14.74
1.80	15.10
1.85	15.45
1.90	15.80
1.95	16.14
2	16.48

1.2 The Error Problem

Whenever a system is modeled as a mathematical formulation, errors are bound to appear in it's solution. These errors usually come from one or both of the two possibilities:

- 1) Error in model approximation
- 2) Error in mathematical computation (Numerical Errors)

The *former error* (i.e. *modeling error*) falls outside the domain of numerical methods, and should be controlled by a careful analysis of the system, and modeling it as accurate and precise as is required. For example, in the previous example of freely falling parachutist, many finer (but less significant) external forces were ignored [such as the shape and curvature of parachutist, his current posture, material of clothing, the spins and turns of parachutist, the change in density of air as he falls, the (micro) change in gravity (to be absolutely precise), considering the quadratic relationship of air resistance to the velocity instead of linearly proportional relationship, etc.], and only forces considered were the *weight* of the parachutist and the *air drag*. This may have sufficed the current requirements for the solution of the problem, but there may arise other situations where this model may fail to give the desired finer precision and higher accuracy, and hence demands a more elaborate and accurate model which would consider even those external factors which were ignored here.

The *latter error* (i.e. *numerical error*) arise due to one or both of the two reasons: either (a) calculations were stopped before the desired precision has reached (remember, numerical methods usually involves recursive formulae, and hence computations should be repeated till a certain predefined precision has reached), or (b) a value may have lost due to use of binary representation of values in the computer.

These numerical errors can be controlled as per the requirement. To control the numerical errors it is important to understand the sources of these errors and then develop strategies to overcome or minimize these errors to the desired accuracy. Numerical errors and their remedies will be studied in the following sections.

1.3 Significant Digits

Significant digits are important in showing the truth, one has, in a reported number. The significant digits of a number are those that can be used with confidence.

Example 1.4: Suppose there is a function in the university where all the Second, Third and Fourth Year students were invited. The manager of food arrangement may ask “How many meals should I order to cook?” In response to his question the answer may come out “Arrange for 4000 meals”.

On the other hand, if the university has decided to grant a special scholarship of ETB 5300 to each of the Second, Third and Fourth year students, and the question posed is “How many ETBs should be deposited in the bank?” then it is inappropriate to use the same count of 4000 students as used during the function, but instead the count should be (something like) 3976.

In the first case (of students’ function) the number of significant digits used to count the students were 1, because the count was taken to reflect the accuracy only in the thousands digit, and the digits in hundreds, tens and ones places were insignificant. But, in the second case (of scholarship grant) the number of significant digits were 4, because the count was considered accurately from thousands up to the ones place.

Although it is usually a straightforward procedure to ascertain the significant digits of a number, some cases can lead to confusion. For example, zeros are not always significant digits because they may be necessary just to locate a decimal point. The numbers 0.00001845, 0.0001845, and 0.001845 all have four

significant digits. Similarly, when trailing zeros are used in large numbers, it is not clear how many, if any, of the zeros are significant. For example, at face value the number 45,300 may have three, four, or five significant digits, depending on whether the zeros are known with confidence.

Such uncertainty can be resolved by using scientific notation, where 4.53×10^4 , 4.530×10^4 , 4.5300×10^4 designate that the number is known to three, four, and five significant digits, respectively.

1.4 Number Representation and Storage in Computers

1.4.1 Representation of Integers

Due to several reasons (mostly technical), computers perform all their calculations in binary number system. This number system consists of only two digits: 0 and 1. As such, integer values are counted in Binary Number System as follows:

Table 1.5: Decimal and Binary representation of values

Value	Decimal Representation	Binary Representation
Zero	0	0
One	1	1
Two	2	10
Three	3	11
Four	4	100
Five	5	101
Six	6	110
Seven	7	111
Eight	8	1000
Nine	9	1001
Ten	10	1010
Eleven	11	1011
Twelve		
e	12	1100

Each place (Ones, Tens, Hundreds, Thousands, etc.) in Decimal Number System can hold one of the ten possible Decimal Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Similarly, each place (Ones, Twos, Fours, Eights, Sixteens, etc.) in Binary Number System can hold one of the two possible Binary Digits: 0, and 1. **Table 1.6** shows place values of Decimal and Binary number systems:

Table 1.6: Place values in Decimal and Binary number systems

	...	10^6	10^5	10^4	10^3	10^2	10^1	10^0
Decimal Number System	...	Millions	Hundred-Thousands	Ten-Thousands	Thousands	Hundreds	Tens	Ones
Binary Number System	...	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	...	Sixty-Fours	Thirty-Twos	Sixteens	Eights	Fours	Twos	Ones

1.4.2 Decimal to Binary Integer Conversion

When we write an integer, such as 237, our mind treats this integer as the decimal representation of the value $200 + 30 + 7 = 237_{10}$, or, more precisely, $2 \times 10^2 + 3 \times 10^1 + 7 \times 10^0 = 237_{10}$. Similarly, the same value, 237, can be binary represented as: $128 + 64 + 32 + 8 + 4 + 1 = 237_2$, or, more precisely, $1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 11101101_2$. Hence, any value can be represented in Binary Number System by dissecting it into equivalent Binary place values (Observe that there is one and only one such possible dissection).

A more practical method to convert any decimal integer to its equivalent binary representation is “Collecting remainders in repeated integer division by 2” as shown below:

Table 1.7: Decimal to Binary integer conversion using repeated division

Divisor	Dividend/Quotient	Remainder
2	237	
2	118	1
2	59	0
2	29	1
2	14	1
2	7	0
2	3	1
2	1	1
2	0	1

Now, read the remainders bottom-up: $237_{10} = 11101101_2$.

1.4.3 Algorithm to convert Decimal Integer to Binary Integer Step # 1.

Initialize the divisor: $\leftarrow 2$

Step # 2.

integer value N :

Initialize the quotient to decimal

Step # 3. Initialize the binary integer (as empty value): $\leftarrow \emptyset$ Step # 4. Repeat steps # 5 and # 6 till Q remains positive

Step # 5. Integer divide Q by d , to calculate new Q and remainder r : $Q = \lfloor Q/d \rfloor$, $r = Q \% d$

Step # 6. Update the binary integer (by attaching r to left of B): $B = r \text{ concatenated with } B$ Step # 7. B contains the binary integer value

1.4.4 Binary to Decimal Integer Conversion

Converting a Binary Integer into equivalent Decimal Integer is as easy as collecting (i.e. adding) all the place values which carry a 1 in the Binary representation. Hence, the decimal equivalent of a Binary Number would be:

110010110011_2

Place-Values Binary Digits	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Decimal Number
Value Contributed by Bit	1	1	0	0	1	0	1	1	0	0	1	1	
$\rightarrow$ (Sum up)	2048	1024	0	0	128	0	32	16	0	0	2	1	$= 3251_{10}$

Bit is an acronym for Binary Digit

1.4.5 Algorithm to convert Binary Integer to Decimal Integer

Step # 1. Initialize m as maximum place value used in binary number B : $\leftarrow$ ()

Step #

2. Initialize the decimal integer value N : $\leftarrow 0$

Step #

3. Initialize p (the power of 2): $\leftarrow 0$

Step # 4. Repeat steps # 5 to # 7 till p remains less than or equal to m

Step #

5. Calculate value contributed by binary digit t at place value 2^p : $\leftarrow t \times 2^p$

Step #

6. Update the decimal integer (by adding v to N): $\leftarrow N + v$

Step #

7. Increase the value of p : $\leftarrow p + 1$

Step #

8. N contains the decimal integer value

1.4.6 Representation of Fractions

To represent fractions (or floating-point numbers) in Binary Number System, the place values continue to the right of decimal-point, just as they do in Decimal Number System:

Table 1.8: Place values for fractions in Decimal and Binary number systems

Decimal	...	10^6	10^5	10^4	10^3	10^2	10^1	10^0	.	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	...
Binary	...	2^6	2^5	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	...

1.4.7 Decimal to Binary Floating-Point Conversion

When we write a decimal fraction, such as 0.6875, our mind treats this integer as the decimal representation of the value $0.6 + 0.08 + 0.007 + 0.0005 = 0.6875_{10}$, or, more precisely, $6 \times 10^{-1} + 8 \times 10^{-2} + 7 \times 10^{-3} + 5 \times 10^{-4} = 0.6875_{10}$. Similarly, the same value, 0.6875, can be binary represented as: $0.5 + 0.125 + 0.0625 = 0.6875_{10}$, or, more precisely, $1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} = 0.1011_2$. Hence, any value can be represented in Binary Number System by dissecting it into equivalent Binary place values (Observe that there is one and only one such possible dissection).

A more practical method to convert any decimal integer to its equivalent binary representation is “Collecting integers in repeated multiplication by 2” as shown below:

Table 1.9: Decimal to Binary floating-point conversion using repeated multiplication

Multiplier	Multiplicand/ Fraction	Product	Integer Part
2	0.6875	1.3750	1
2	0.375	0.750	0
2	0.75	1.50	1
2	0.5	1.0	1

Now, read the integers top-down: $0.6875_{10} = 0.1011_2$.

1.4.8 Algorithm to convert Decimal Fraction to Binary Fraction

Step # 1. Initialize the multiplier: $\leftarrow 2$

Step # 2. Initialize the multiplicand to decimal fraction value N : $\leftarrow$ 0.

Step # 3. Initialize the binary fraction (with a zero followed by point): $\leftarrow$ 0.

Step # 4. Repeat steps # 5 to # 8 till F remains more than zero

Step #

5. Multiply F by m , as P : $\leftarrow \times$

Step #

6. Put fractional part of P in r : $\leftarrow$ ()

Step #

7. Put integer part of P in F : $\leftarrow$ ()

Step #

8. Update the binary fraction (by attaching r to right of B): $\leftarrow$ (,)

Step #

9. B contains the binary integer value

1.4.9 Binary to Decimal Integer Conversion

Converting a Binary Fraction into equivalent Decimal Fraction is as easy as collecting (i.e. adding) all the place values which carry a 1 in the Binary representation. Hence, the decimal equivalent of a Binary Fraction

0.1100101₂ would be:

Place-Values	2 ⁻¹	2 ⁻²	2 ⁻³	2 ⁻⁴	2 ⁻⁵	2 ⁻⁶	2 ⁻⁷	Decimal
Binary Digits	1	1	0	0	1	0	1	Number
Value Contributed by Bit (Sum up)	0.5	0.25	0	0	0.03125	0	0.0078125	= 0.7890625 ₁₀

1.4.10 Algorithm to convert Binary Fraction to Decimal Fraction

Step # 1. Initialize m as absolute minimum place value used in binary fraction B :

Step #

2. Initialize the decimal fraction value N : $\leftarrow$ 0

Step #

3. Initialize p (the power of 2): $\leftarrow$ 0

Step # 4. Repeat steps # 5 to # 7 till p remains less than or equal to m

Step #

5. Calculate value contributed by binary digit t at place value 2^p : $\leftarrow \times 2^p$

Step #

6. Update the decimal fraction (by adding v to N): $\leftarrow +$

Step #

7. Increase the value of p : $\leftarrow + 1$

Step # N contains the decimal integer value

8.

1.5 Numerical Errors

1.5.1 Round-Off Errors

The first category of Numerical Errors is Round-Off Errors. **Round-Off Errors** originate from the fact that computers retain only a limited number of significant digits during calculations, and the remaining value is discarded and lost. If the calculated value falls within the range of significant digits then the calculations will proceed with precise results.

For example, if the precision of a computer is to retain only the first 15 decimal places (these 15 decimal places would be part of total number of significant digits of the computer, remaining significant digits would contribute to integer part of rational numbers), then the fraction $\frac{1}{2}$ would be retained as

0.5000000000000000. Hence, if a calculation such as $\frac{1}{2} \times 31.76$ is made then the calculations proceed with accurate results: $\frac{1}{2} \times 31.76 = 0.5000000000000000 \times 31.76 = 15.880000000000000$. This case happened because the value of fraction $\frac{1}{2}$ was retained well within the range of significant digits provided by the computer.

On the contrary, if the fraction happened to be $\frac{1}{3}$ instead of $\frac{1}{2}$ then the calculation would proceed as $\frac{1}{3} \times 31.76 = 0.3333333333333333 \times 31.76 = 10.586666666666666$ which is definitely not accurate but only an approximation (because some part of true value is lost). This is a case of Round-Off Error. Similar cases happen when we want to perform calculations which involve π , or $\sqrt{2}$.

Another reason for Round-Off Errors is the use of Binary Number System in computers. Some values, which can be accurately represented in Decimal Number System, can never be represented accurately in Binary Number System.

For example, fraction $\frac{2}{5}$ can be represented accurately as 0.4000000000000000 in Decimal Number System, whereas, if we try to represent the same fraction in Binary Number System then value would be stored as 0.011001100110011 which is only an approximation (because some value is lost).

These errors which result because of lack of precise and accurate representation of values in computers are known as **Round-Off Errors**.

1.5.2 Numerical Cancellations

Numerical Cancellations are a result or consequence of *Round-Off Errors*. Numbers are stored in computers in the scientific notation format: $m \times 10^e$, where m is called the mantissa, b is the base (most of the time it is taken as 10), and e is known as the exponent. While storing the values, b is not stored but presumed, and only the values of m and e are stored.

Furthermore, to increase the accuracy of stored values, many computers “standardize” the values by effectively shifting the values to fit in available significant digits. For example, if a computer is storing only first

7 decimal digits, when a calculation like 0.0000339 is performed, and the value is stored normally, then the value stored would be 0.0000339 . But the computer would shift the number and use scientific notation to store the value as 0.3397087×10^{-4} , which is definitely more accurate. But, this shifting of digits, in some cases, results in loss of accuracy.

For example, if two numbers, 0.2671507×10^1 and 0.3844981×10^{-2} , are to be added, then computer, before proceeding for addition, shifts the digits of mantissa for smaller number (here, the second number), so as to equate the exponents of the two numbers, and converts the second number to 0.0003844×10^1 , then performs the actual addition:

$$\begin{array}{r} 0.2671507 \times 10^1 \\ + 0.0003844 \times 10^1 \\ \hline 0.2675351 \times 10^1 \end{array}$$

Similar cases occur during other arithmetic operations also. And the effect cumulates as the number of computations increase.

1.5.3 Truncation Errors

The second category of Numerical Errors is Truncation Errors. **Truncation Errors** are those that result from using an approximation in place of an exact mathematical procedure.

For example, we know that the rate of change of velocity (which is known as acceleration) of a body is equal to the Force acting on unit mass of the body:

$$\frac{dv}{dt} = \frac{\Delta v}{\Delta t} = \frac{(v_{i+1} - v_i)}{\Delta t} = \frac{v_{i+1} - v_i}{t_{i+1} - t_i}$$

This is the “exact” relationship between the terms involved. But, for the purpose of numerical calculations we approximate this relationship by writing:

Because this is only an approximation of true value we introduced an error in our calculation.

Such errors which result due to the substitution of approximate mathematical models in place of true analytical solutions are known as **Truncation Errors**.

1.6 Computational Methods for Error Estimation

1.6.1 Computational Methods

Computational Methods (a.k.a **Numerical Methods**) are procedures that allow for efficient solution of a mathematically formulated problem in a finite number of steps, within an arbitrary precision. Although scientific calculators can handle simple problems, computers are needed in most other cases.

An **algorithm** is a well defined step-by-step procedure, consisting a finite number of steps, to solve a given problem, in a finite amount of time. Numerical Methods are those algorithms, which are translated into a form most suitable for computers, and consists of a set of guidelines to perform predetermined mathematical (i.e. algebraic and logical) operations leading to an approximate solution of a specific problem.

1.6.2 Computational Efficiency

To effectively describe an algorithm, we will use code. A code consists of a set of inputs, the required operations, and a list of outputs.

An algorithm is stable if a small change in the initial data will correspond to a small change in the final output. Otherwise, the algorithm is unstable.

An **Iterative Method** (a.k.a. **Iterative Procedure** or simply **Procedure**) is a process that starts with an initial guess and computes successive approximations of the solution of a problem until a '*reasonably accurate approximation*' is obtained.

An Iterative Method is a representative of an algorithm, and as such, it must stop after a finite number of iterations and/or time. There are *two ways to stop a procedure*: (1) when a terminating condition is satisfied, or (2) when the maximum number of iterations have exceeded. Technically, terminating condition is called **Stopping Criteria**.

In principle, the terminating condition should check to see whether an approximation calculated in a step is within a prescribed tolerance of the true value. This is known as **true error**. In practice, however, the true value (of solution) is not available. As a result, one practical form of a terminating condition is whether the difference between two successively generated quantities by the iterative method is within a prescribed tolerance. This difference is known as **approximate error**. The ideal scenario is when an algorithm meets the terminating condition, at a reasonably fast rate. If it does not, then the total number of iterations performed should not exceed a prescribed maximum number of iterations.

Computational Efficiency is a relative measure of speed of an algorithm, compared to similar other approaches, to solve the same problem. A good estimate of *computational efficiency* can be given in the form of **Rate of Convergence** (many people simply call it **Convergence**) of the algorithm. To understand rate of convergence, consider a sequence that converges to . Then, the error at the nth iteration is then defined as:

If there exists a positive number ϵ and a positive constant C such that

$$\lim_{n \rightarrow \infty} \frac{|e_n|}{|e_{n-1}|} = C, \quad n = 0, 1, 2, \dots$$

then we say that is the *rate of convergence* of the sequence. For some sequences, $C = 1$ satisfies the above equation, in which case we say that *convergence of the sequence is linear*. For other sequences, $C = 2$ satisfies this condition, to say *convergence is quadratic*. There are yet other sequences whose rate of convergence is found to be *fractional*! ($1 < C < 2$).

1.6.3 True Error

Numerical errors are result of approximations done while calculating solutions. In general, analytical solutions give exact value or **True Value**, whereas, numerical methods provide only **Approximate Value**. The relationship between True Value and Approximate Value is always:

$$\text{True Error} = \text{True Value} - \text{Approximate Value}$$

or, equivalently:

$$\text{True Error} = \text{True Value} - \text{Approximate Value} \quad \text{Eq.1.11}$$

Example 1.5 : You were asked to measure the height of a building and a car, and your instruments measured their heights as 999 cm and 24 cm, respectively. But, the actual heights are known to be 1000 cm and 25 cm, respectively. How much are the errors?

For the building,

$$= 1000 - 999 = 1$$

and, for the car

$$= 25 - 24 = 1$$

As can be observed, True Error does not reveal the “severity” of error. In this example, it looks as if errors in both measurements are similar and equal.

1.6.4 Relative True Error

The “severity” of error can be estimated by comparing (i.e. calculating the ratio of) the *True Error* with the *True Value*. This is known as **Relative Percentage True Error** or simply **Relative True Error**:

$$= \frac{\text{True Error}}{\text{True Value}} \times 100 \quad \text{Eq.1.12}$$

Example 1.6 : For the problem given in Example 1.5, calculate the *Relative True Errors*.

For the building,

$$= \frac{1}{999} \times 100 \approx 0.1\%$$

and, for the car

$$= \frac{1}{25} \times 100 = 4\%$$

As can be seen, the error in calculation of height of car was much more severe compared to error in calculation of height of building. Hence, almost always, we use *Relative True Error* while performing numerical calculations.

1.6.5 Relative Approximate Error

As mentioned earlier, *Numerical Methods* always produce approximate solutions, and, as such, it is not possible to get the *True Value* of solution to calculate the *Relative True Error*. Due to this situation an alternate approach to “estimate” the error is to calculate **Relative Percentage Approximate Error** or simply **Relative Approximate Error**:

$$= \frac{\text{Approximate Error}}{\text{Approximate Value}} \times 100$$

i.e. = $\frac{\text{Approximate Error}}{\text{Approximate Value}} \times 100$

or, equivalently:

$$= \frac{\text{Absolute Error}}{\text{Magnitude of the Quantity}} \times 100\% \quad \text{Eq.1.13}$$

1.6.6 Tolerance

As would be learnt further, Numerical Methods often involve repeated calculations (reason being most of these methods are either recursive or iterative), and there should be some criteria to stop those repetitions. This Stopping Criteria is mostly mentioned in the form of a tolerance.

Tolerance, denoted by ϵ , is the maximum Relative Approximate Error allowed, and this would be usually mentioned in the form of precision required for a given number of significant digits. Hence, if precision is required up to n significant digits then $\epsilon = 10^{-n}$. For example, if precision is desired up to 5 significant digits then tolerance is $\epsilon = 10^{-5} = 0.00001$.

Because the number of significant digits will always be positive the stopping criteria most often used in Numerical Methods would be:

$$| \epsilon | \leq \epsilon$$

Eq.1.14

i.e. absolute relative approximate error should not be more than a predefined tolerance. As such, most of the time we calculate absolute errors rather than actual errors (which may be positive or negative).

1.6.7 Accuracy of Solution

It is worth mention that if the absolute relative approximate error falls below the following value:

$$| \epsilon | \leq (0.5 \times 10^{-n})\%$$

Eq.1.15

then, we can be pretty sure that our solution is accurate up to at least n significant digits (it is a proven lemma).

Example 1.7: Find $\sqrt{0.5}$ correct to 3 decimal places.

Let us start by calculating desired Percent Tolerance:

$$\epsilon = 0.5 \times 10^{-2-} = 0.5 \times 10^{-2-3} = 0.05\%$$

Hence, we proceed till the error falls below 0.05%. The Maclaurin series expansion of e^x is:

$$= 1 + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots + \frac{x^n}{n!}$$

In this series we keep on adding new terms till the error falls below 0.05%. We start to calculate our first approximation by substituting $x = 0$, to get:

$$0.5 = e^{0.50} = 1 + \frac{0.50}{1} + \frac{0.50^2}{2!} + \dots$$

At this point it is not possible to calculate the error (because there is no previous approximation available to which we can compare the current approximation). So, we proceed by adding next term:

$$0.5 = \frac{0.5^0}{1} + \frac{0.5^1}{1} = 1.5$$

Now, the *Percentage Relative Approximate Error* is:

$$= \left| \frac{1.5 - 0.5}{1.5} \right| \times 100 = 66.67\%$$

Definitely, $66.67\% > 0.05\%$. Hence, we proceed by adding more terms. These calculations are shown tabulated below:

Table 1.10 Calculate correct to 3 decimals

<i>n</i>		(%)
0	1	
1	1.5	33.33
2	1.625	7.69
	1.6458333	
3	33	1.27
	1.6484375	0.15
4	00	8
	1.6486979	0.015
5	17	8

As **Table 1.10** shows, the error falls below ($= 0.05\%$) after adding 6 terms of the series (i.e. at $n = 5$), and 0.5 can be calculated correct to 3 decimal places as 1.648 (the actual value is $0.5 = 1.648721 \dots$)

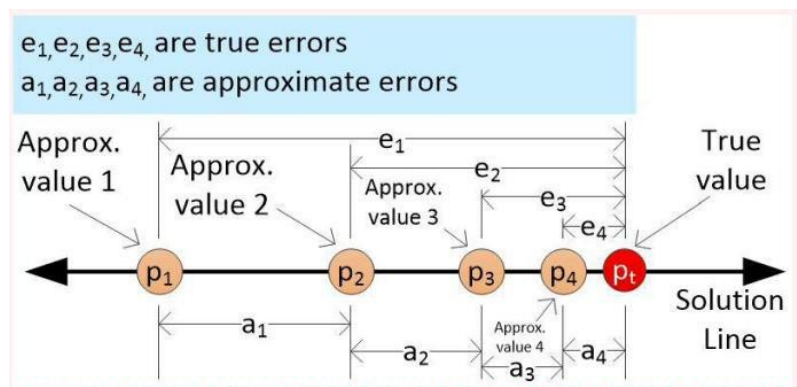
1.6.8 Error Estimation

As mentioned earlier, one form of *stopping criteria* of an algorithm is “*successively diminishing errors in any two consecutive iterations should fall below a prescribed tolerance*”. As such, the *error* should be calculated in every *iteration* of the *algorithm*.

But it is also explained earlier, that calculation of *true error* is not possible, because we do not know the *true value* of computation, and, as such, we seek to find the *approximate error* in calculation.

Using *approximate error* as *stopping criteria* of an iterative algorithm is as good as using the *true error*. This can be understood with **Figure 1.2**.

In the figure, for some given problem, the actual value of solution is given by the point represented by right-most point of the solution line, (representing point of true solution). Suppose our *iterative algorithm* found the *first approximate solution*, shown by the left-most



point on the solution line, p_1 . In second

iteration, our *iterative algorithm* produces a

Figure 1.2 Concept of True and Approximate Errors

second approximate solution, y_2 . Similarly, the

third, fourth, fifth, and remaining approximate solutions would be found, and each new approximate solution would move closer and closer to the actual solution value, .

When the iterative algorithm found the first approximate solution, x_1 , the *true error* between the *true value* and *first approximate value* x_1 is e_1 (*first true error*). Similarly, other successive true errors e_2, e_3, e_4, e_5 , and so on (for demonstration purpose, only few of these points have been shown.)

But, the problem we face while calculating *true errors* e_1, e_2, e_3, e_4, e_5 , and so on, is that, we do not know the value of *actual solution* or *true value*, . To overcome this problem, we seek to calculate the *approximate errors* $\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4, \epsilon_5$, and so on, instead of calculating true errors e_1, e_2, e_3, e_4, e_5 , and so on.

A very important point to be observed is that $\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4, \epsilon_5$, and so on, are decreasing very "*proportionally*" to e_1, e_2, e_3, e_4, e_5 , and so on, respectively. This relationship between the *approximate error* and *true error*, combined with the possibility to calculate *approximate errors* $\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4, \epsilon_5$, and so on, provides us to employ the *stopping criteria* needed for our *iterative algorithm*.

If we generalize the approximate error found in the k^{th} iteration as ϵ_k , then the stopping criteria would be $\epsilon_k < \epsilon_{\text{tol}}$, where ϵ_{tol} is the *tolerance* of error.