Computer-based Numerical Methods
Board of Studies

Prof. H. N. Verma
Vice-Chancellor
Jaipur National University, Jaipur

Prof. M. K. Ghadoliya
Director,
School of Distance Education and Learning
Jaipur National University, Jaipur

Dr. Rajendra Takale
Prof. and Head Academics
SBPIM, Pune

Subject Expert Panel

Dr. Ramchandra G. Pawar
Director, SIBACA, Lonavala
Pune

Ashwini Pandit
Subject Matter Expert

Content Review Panel

Gaurav Modi
Subject Matter Expert

Shubhada Pawar
Subject Matter Expert

Copyright ©

This book contains the course content for Computer-based Numerical Methods.

First Edition 2013

Printed by
Universal Training Solutions Private Limited

Address
05th Floor, I-Space,
Bavdhan, Pune 411021.

All rights reserved. This book or any portion thereof may not, in any form or by any means including electronic or mechanical or photocopying or recording, be reproduced or distributed or transmitted or stored in a retrieval system or be broadcasted or transmitted.
Index

I. Content ........................................................................................................ II
II. List of Figures .............................................................................................. V
III. List of Tables ............................................................................................. VI
IV. Abbreviations ............................................................................................ VII
V. Case Study .................................................................................................. 113
VI. Bibliography .............................................................................................. 120
VII. Self Assessment Answers ........................................................................ 121

Book at a Glance
## Contents

**Chapter I**

- **Introduction to C Programming Language** .................................................. 1
  - Objective ........................................................................................................ 1
  - Learning outcome .......................................................................................... 1
  - 1.1 Introduction .................................................................................................. 2
  - 1.2 First Program in C ......................................................................................... 2
  - 1.3 Constants and Variables .............................................................................. 3
  - 1.4 Types of Variables ......................................................................................... 3
  - 1.5 Basic Data Types .......................................................................................... 4
  - 1.6 Loops ............................................................................................................. 5
  - 1.7 Control Structure .......................................................................................... 7
  - 1.8 Function ......................................................................................................... 11
  - 1.9 Pointer ........................................................................................................... 12
  - 1.10 Array ............................................................................................................ 13
  - 1.11 String .......................................................................................................... 15
  - 1.12 Structure and Union .................................................................................... 16

**Summary** ........................................................................................................... 19

**References** .......................................................................................................... 19

**Recommended Reading** ...................................................................................... 19

**Self Assessment** .................................................................................................. 20

**Chapter II**

- **Solution of Equations and Eigen Value Problems** ........................................ 22
  - Objective ........................................................................................................... 22
  - Learning outcome ............................................................................................. 22
  - 2.1 Definitions of Eigenvalues ........................................................................... 23
  - 2.2 Introduction to Eigenvalues .......................................................................... 23
  - 2.3 Solution to Algebraic Equations ................................................................... 25
  - 2.4 Root/Zero ....................................................................................................... 25
  - 2.5 Simple Root ................................................................................................... 26
  - 2.6 Multiple Root ................................................................................................. 26
  - 2.7 Direct Methods ............................................................................................. 27
  - 2.8 Iterative Methods .......................................................................................... 27
  - 2.9 Convergence of Iterative Methods ................................................................. 27
  - 2.10 Criterion to Terminate Iteration Procedure ................................................ 27
  - 2.11 Initial Approximation for an Iterative Procedure .......................................... 28
  - 2.12 Method of False Position ............................................................................ 28
  - 2.13 Newton-Raphson Method .......................................................................... 30
  - 2.14 General Iteration Method ........................................................................... 33
  - 2.15 Condition of Convergence .......................................................................... 34
  - 2.16 Convergence of the Iteration Methods ......................................................... 35
  - 2.17 Method of false position ............................................................................. 35
  - 2.18 Method of Successive Approximations or Fixed Point Iteration Method ... 36
  - 2.19 Newton-Raphson Method .......................................................................... 36

**Summary** ............................................................................................................... 39

**References** .......................................................................................................... 39

**Recommended Reading** ...................................................................................... 39

**Self Assessment** .................................................................................................. 40
Chapter III

Linear System of Algebraic Equations ................................................................. 42
Aim ....................................................................................................................... 42
Objectives ............................................................................................................ 42
Learning outcome .............................................................................................. 42
3.1 Introduction to Linear Algebraic Equation .................................................... 43
3.2 Method Linear Algebraic System .................................................................. 43
3.3 Elementary Row Transformations (Operations) ............................................ 44
3.4 Gauss Elimination Method .......................................................................... 45
3.5 Pivoting Procedures ...................................................................................... 47
3.6 Gauss-Jordan Method .................................................................................. 50
3.7 Inverse of a Matrix by Gauss-Jordan Method .............................................. 52
3.8 Iterative Methods .......................................................................................... 55
3.9 Gauss-Jacobi Iteration Method .................................................................... 56
3.10 Gauss-Seidel Iteration Method ................................................................. 60
3.11 Eigen Value Problem .................................................................................. 64
3.12 Power Method ............................................................................................ 64
Summary ............................................................................................................. 68
References .......................................................................................................... 68
Recommended Reading ..................................................................................... 68
Self Assessment .................................................................................................. 69

Chapter IV

Interpolation ......................................................................................................... 71
Aim ....................................................................................................................... 71
Objectives ............................................................................................................ 71
Learning outcome .............................................................................................. 71
4.1 Introduction to Interpolation ....................................................................... 72
4.2 Theorem: Weierstrass Approximation Theorem .......................................... 73
4.3 Interpolation and the Lagrange Polynomial ................................................ 75
4.4 Divided Differences ..................................................................................... 76
4.5 Newton’s Interpolatory Divided-Difference Formula .................................. 78
4.6 Theorem ....................................................................................................... 78
4.7 Inverse Interpolation ................................................................................... 80
4.8 Errors in Numerical Differentiation ............................................................ 81
4.9 Numerical Integration .................................................................................. 81
4.10 Newton-Cote’s Quadrature Formula .......................................................... 82
4.11 Trapezoidal Rule (N = 1) ........................................................................... 83
4.12 Simpson’s One-Third Rule (n = 2) ............................................................. 83
4.13 Simpson’s Three-Eighth Rule (N = 3) ......................................................... 84
4.14 Boole’s Rule ............................................................................................... 84
4.15 Gaussian Quadrature Formula .................................................................... 85
4.16 Numerical Evaluation Of Singular Integrals .............................................. 86
Summary ............................................................................................................. 88
References .......................................................................................................... 89
Recommended Reading ..................................................................................... 89
Self Assessment .................................................................................................. 90

Chapter V

Numerical Solution of Ordinary Differential Equations ................................. 92
Aim ....................................................................................................................... 92
Objectives ............................................................................................................ 92
Learning outcome .............................................................................................. 92
5.1 Introduction .................................................................................................. 93
5.2 Initial-Value and Boundary-Value Problems .............................................. 93
List of Figures

Fig. 1.1 C constants ................................................................................................................................. 4
Fig. 2.1 Root of f(x) =0 .......................................................................................................................... 26
Fig. 2.2 False position ............................................................................................................................ 29
Fig. 2.3 Network-Raphson method........................................................................................................ 31
Fig. 4.1 Graph of population................................................................................................................... 72
Fig. 4.2 Graph of polynomial .................................................................................................................. 73
Fig. 4.3 Graph of polynomial of equation (2).......................................................................................... 74
Fig. 4.4 Function of P(x)........................................................................................................................ 75
Fig. 4.5 Linear interpolation ................................................................................................................... 76
Fig. 4.6 Graph of a typical $L_{nk}$ ......................................................................................................... 82
Fig. 4.7 Numerical integration................................................................................................................ 102
Fig. 5.1 Euler’s method........................................................................................................................... 101
Fig. 5.2 Modified Euler’s method......................................................................................................... 102
Fig. 5.3 Graph for Y=1.0928 and x=0.1.............................................................................................. 103
List of Tables

Table 1.1 Keywords .......................................................................................................................... 3
Table 1.2 Basic data types .................................................................................................................. 4
Table 1.3 Data type – range of values ............................................................................................... 4
Table 1.4 Arrays and pointers .......................................................................................................... 13
Table 1.5 String functions ................................................................................................................ 16
Table 1.6 File operation ................................................................................................................... 18
Table 4.1 List of population .............................................................................................................. 72
Table 4.2 For approximate value “n” ............................................................................................... 74
Table 4.3 Generation of the divided differences .............................................................................. 77
Table 4.4 Polynomial in equation (30) ............................................................................................. 78
Table 4.5 Data in corresponding to f(x) ............................................................................................ 80
Table 4.6 Table for n=5 .................................................................................................................... 86
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>BCPL</td>
<td>Basic Combined Programming Language</td>
</tr>
<tr>
<td>BLAS</td>
<td>Basic Linear Algebra Subroutines</td>
</tr>
<tr>
<td>CPL</td>
<td>Combined Programming Language</td>
</tr>
<tr>
<td>DEPACK</td>
<td>Differential Equations Package</td>
</tr>
<tr>
<td>EISPACK</td>
<td>Matrix Eigensystem Routines</td>
</tr>
<tr>
<td>ELLPACK</td>
<td>Elliptic Partial Differential Equations Solver</td>
</tr>
<tr>
<td>IMSL</td>
<td>International Mathematical and Statistical Library</td>
</tr>
<tr>
<td>NAG</td>
<td>Numerical Algorithm Group</td>
</tr>
<tr>
<td>O.D.E</td>
<td>Ordinary Differential Equation</td>
</tr>
<tr>
<td>PDP</td>
<td>Programmed Data Processor</td>
</tr>
</tbody>
</table>
Chapter I
Introduction to C Programming Language

Aim
The aim of this chapter is to:

• introduce of C programming language
• discuss first program in C
• understand the printf syntax

Objectives
The objectives of this chapter are to:

• explain scanf() syntax
• elaborate C constants
• evaluate various keywords in C language

Learning outcome
At the end of this chapter, the students will be able to:

• enlist the types of variable
• draw basic data types
• understand the concept of loops
• explain the control structure
1.1 Introduction
C was developed by Dennis Ritchie at Bell Laboratories in 1972. Most of its principles and ideas were taken from
the earlier language B, BCPL and CPL. CPL was developed jointly between the Mathematical Laboratory at the
University of Cambridge and the University of London Computer Unit in 1960s. CPL (Combined Programming
Language) was developed with the purpose of creating a language that was capable of both machine independent
programming and would allow the programmer to control the behaviour of individual bits of information. But the
CPL was too large for use in many applications.

- In 1967, BCPL (Basic Combined Programming Language) was created as a scaled down version of CPL, while
  still retaining its basic features. This process was continued by Ken Thompson.
- He made B Language during working at Bell Labs.
- B Language was a scaled down version of BCPL. B Language was written for the systems programming.
- In 1972, a co-worker of Ken Thompson, Dennis Ritchie developed C Language by taking some of the generality
  found in BCPL to the B language.
- The original PDP-11 version of the Unix system was developed in assembly language.
- In 1973, C language had become powerful enough that most of the Unix kernel was rewritten in C. This was
  one of the first operating system kernels implemented in a language other than assembly.
- During the rest of the 1970’s, C spread throughout many colleges and universities because of its close ties to
  UNIX and the availability of C compilers.
- Soon, many different organisations began using their own versions of C Language. This was causing great
  compatibility problems.
- In 1983, the American National Standards Institute (ANSI) formed a committee to establish a standard definition
  of C Language, which is known as ANSI Standard C.
- Today, C is the most widely used System Programming Language.

1.2 First Program in C
Here is your first C program. Write carefully because C Language is a case sensitive language.

```c
#include <stdio.h>

void main()
{
  printf("Hello World\n");
}
```

Press ALT+F9 to compile your program. If you have any error in your program, you will get the message, remove
your errors and then execute your program you will get the output.

Hello World
printf()

The printf() function prints output to stdout, according to format and other arguments passed to printf(). The string
format consists of two types of items - characters that will be printed to the screen, and format commands that define
how the other arguments to printf() are displayed.

```c
printf( "Hello World’);
scanf()
```
The `scanf()` function reads input from stdin, according to the given format, and stores the data in the other arguments. It works a lot like `printf()`. The format string consists of control characters, whitespace characters, and non-whitespace characters.

```c
void main(void)
{
    int i;
    scanf("%d", &i);
    printf("%d", i);
}
```

1.3 Constants and Variables

Let us go through constants and variables used in C programming.

C Constants

The alphabets, numbers and special symbols when properly combined, form constants, variables and keywords. A constant is an entity that does not change.

Variables

A variable is an entity that may change its value. In any program, we typically do a lot of calculations. The results of these calculations are stored in computer memory locations. To make the retrieval and usage of these values, we give names to the memory locations. These names are called variables.

Keywords

A keyword is a word that is part of C Language itself. These words have predefined meanings and these words cannot be used as variable names.

<table>
<thead>
<tr>
<th>C Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
</tr>
<tr>
<td>auto</td>
</tr>
<tr>
<td>if</td>
</tr>
<tr>
<td>continue</td>
</tr>
<tr>
<td>typedef</td>
</tr>
<tr>
<td>float</td>
</tr>
<tr>
<td>volatile</td>
</tr>
<tr>
<td>while</td>
</tr>
</tbody>
</table>

Table 1.1 Keywords

1.4 Types of Variables

There are two main types of variables in C: numeric variables that hold only numbers or values, and string variables that hold text, from one to several characters long.
Following are the basic data types.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>32768 to +32767</td>
</tr>
<tr>
<td>Floating point</td>
<td>3.4e-38 to 3.4e+38</td>
</tr>
<tr>
<td>Double floating point</td>
<td>1.7e+308</td>
</tr>
<tr>
<td>Character</td>
<td>-128 to 127</td>
</tr>
</tbody>
</table>

Table 1.3 Data type – range of values
• **int**
  - used to define integer numbers
  - no decimal points are used
  - takes 16 bytes of memory to store the value on 16 bit machine
  - an integer can be positive or negative

• **float**
  - used to define floating point numbers
  - stores only floating point values
  - the values should be in decimal places and it takes 32 bytes of memory to store the value on 16 bit machine

• **double**
  - used to define big floating points
  - used to get the store on 16 bit machine, it requires 64 bytes of memory

• **char**
  - used to define characters
  - used to store single character or number at a time
  - used a 16 bit machine, 8 byte memory is required for character type of data

• **void**
  - We can specify a function using void data type.

### 1.6 Loops
Loops are used to repeat one statement or set statements more than one time. Most real programs contain some construct that loops within the program, performing repetitive actions on a stream of data or a region of memory. There are several ways to loop in C.

**For Loop**
- For loop is a counter loop.
- The for loop allows automatic initialisation of instrumentation of a counter variable.
- The general form is:

```c
for (initialisation; condition; increment/decrement) {
    statements block
}
```

- If the statement block is only one statement, the braces are not necessary.
- Although the for allows a number of variations, generally the initialisation is used to set a counter variable to its starting value.
- The condition is generally a relational statement that checks the counter variable against a termination value, and the increment increments (or decrements) the counter value.
- The loop repeats until the condition becomes false.
Example

```c
Main()
{
    int i;
    for(i = 0; i < count; i++)
    {
        printf("%d\n",i);
    }
}
```

**While Loop**
- The while loop repeats a statement until the test at the top proves false.
- The while loop has the general form:

```
while(condition)
{
    statement block
}
```

- The while tests its condition at the top of the loops.
- Therefore, if the condition is false to begin with, the loop will not execute at all.
- The condition may be any expression.
- An example of a while is as follows.
- It reads characters until end-of-file is encountered.

Example

```c
Main()
{
    int t = 0;
    while(t<=10)
    {
        printf("%d\n",t);
        t=t+1;
    }
}
```

do-while loop
- This is very similar to the while loop except that the test occurs at the end of the loop body.
- This guarantees that the loop is executed at least once before continuing.
- Such a setup is frequently used where data is to be read.
- The test then verifies the data, and loops back to read again if it was unacceptable.
1.7 Control Structure

C language possesses such decision making capabilities and supports the following statements known as control or decision-making statements:

- if statement
- switch statement
- Conditional operator statement
- goto statement

if statement
- The “if statement” is a powerful decision making statement and is used to control the flow of execution of statements.
- It is basically a two-way decision statement and is used in conjunction with an expression.

Syntax

```
if (conditional)
{
    block of statements executed if conditional is true;
}
else
{
    block of statements if condition false;
}
```

Example

```
main()
{
    int x=5
    if (x > 1)
    {
        x=x+10;
    }
    printf("%d", x);
}
```

if...else statement

The if...else statement is an extension of the simple if statement. The general form is:
If the condition is true, then the true-block statement(s), immediately following the if statement are executed; otherwise the false-block statement(s) are executed.

Example

```c
void main(void)
{
    int a, b;
    char ch;

    printf("Choice: \n");
    printf("(A) Add, (S) Subtract, (M) Multiply, or (D) Divide? \n");
    ch = getchar();
    printf("\n");

    printf("Enter a: ");
    scanf("%d", &a);
    printf("Enter b: ");
    scanf("%d", &b);

    if(ch=='A') printf("%d", a+b);
    else if(ch=='S') printf("%d", a-b);
    else if(ch=='M') printf("%d", a*b);
    else if(ch=='D' && b!=0) printf("%d", a/b);
}
```

if-else-if statement example

```c
void main(void)
{
    int numb;

    printf("Type any Number : ");
    scanf("%d", &numb);

    if(numb > 0)
    {
        printf("%d is the positive number", numb);
    }
    else if(numb < 0)
    {
        printf("%d is the Negative number", numb);
    }
    else printf("%d is zero",numb);
}
```
Switch statement
The switch case statements help control complex conditional and branching operations. The switch statement transfers control to a statement within its body.

Syntax
```
switch (expression)
{
    case item:
        statements;
        break;
    case item:
        statements;
        break;
    case item:
        statements;
        break;
    case item:
        statements;
        break;
    default:
        statement;
        break;
}
```

Example
```
#include

main()
{
    int numb;
    printf("Type any Number");
    scanf("%d", &numb);

    switch(numb % 2)
    {
        case 0:
            printf("the number %d is even \n", numb);
            break;

        case 1:
            printf("the number %d is odd \n", numb);
            break;
    }
}
```

Ternary condition
The ‘?’ (ternary condition) operator is a more efficient form for expressing simple if statements. It has the following form:

`expression1 ? expression2 : expression3`

Example
```
res = (a>b) ? a : b;
```

if a is greater than b than res has the value a else the res has value b.
**break statement**

break statement is used to exit from a loop or a switch, control passing to the first statement beyond the loop or a switch.

With loops, break can be used to force an early exit from the loop, or to implement a loop with a test to exit in the middle of the loop body. A break within a loop should always be protected within an if statement which provides the test to control the exit condition.

**Example**

```c
For(i=0;i<=10;i++)
{
    if(i==5){
        break;
    }
    printf("\n%d",i);
}
```

**Output**

```
0
1
2
3
4
```

**continue statement**

- continue is similar to the break statement but it only works within loops where its effect is to force an immediate jump to the loop control statement.
- Like a break, continue should be protected by an if statement.
Example

```c
For(i=0;i<10;i++)
{
    if(i==5){
        continue;
    }

    printf("\n%d",i);
}
```

Output:

0
1
2
3
4
6
7
8
9

The goto statement

- The goto is a unconditional branching statement used to transfer control of the program from one statement to another.
- One must ensure not to use too much of goto statement in their program because its functionality is limited.
- It is only recommended as a last resort if structured solutions are much more complicated.

1.8 Function

- Function is a block of statements which perform some specific task and always return single value to the calling function.
- Functions are used to minimise the repetition of code.
- Some languages distinguish between the functions which return variables and those which don’t.
- C assumes that every function will return a value.
- If the programmer wants a return value, this is achieved using the return statement.
- If no return value is required, none should be used when calling the function.
- There are two types of functions in c language.
Library functions
A function which is predefined in c language is called library function printf(), scanf(), getch() etc are library functions

User defined functions
A function written by a programmer is called user defined function.

Example
```c
#include
int add (int x, int y) {
    int z;
    z = x + y;
    return (z);
}
main ()
{
    int i, j, k;
    i = 15;
    j = 5;
    k = add(i, j);
    printf (“The value of k is %d\n”, k);
}
```

Output
The value of k is 30

Scope of function
Only a limited amount of information is available within the body of each function. Variables declared within the calling function can’t be accessed from the outside functions unless they are passed to the called function as arguments.

Global variables
A variable that is declared out side all functions is called Global variable. Global variables don’t die on return from a function. Their value is retained, and is available to any other function in whole program.

Local variables
A variable that is declared within a function is called Local variable. They are created each time the function is called, and destroyed on return from the function. The values passed to the functions (arguments) are also treated like local variables.

Static variables
Static variables are like local variables but they don’t die on return from the function. Instead their last value is retained, and it becomes available when the function is called again.

1.9 Pointer
A pointer is a variable suitable for keeping memory addresses of other variables; the values you assign to a pointer are memory addresses of other variables or other pointers.
C pointers are characterised by their value and data-type.

• The value is the address of the memory location the pointer points to, the type determines how the pointer will be incremented/decremented in pointer or subscript arithmetic.
• Pointers are used to manipulate arrays and they can be used to return more than one value from a function.
Pointers are declared by using the asterisk (*).

```c
int *p;
```

- Each variable has two attributes: address and value.
- The address is the location in memory.
- In that location, the value is stored.
- During the lifetime of the variable, the address is not changed but the value may change.

```c
#include

void main (void)
{
    int i;
    int * a;
    i = 10;
    a = &i;
    printf ("The address of i is %8u \n", a);
    printf ("The value at that location is %d\n", i);
    printf ("The value at that location is %d\n", *a);
}
```

Output:
The address of i is 631672
The value at that location is 10
The value at that location is 10

**Arrays and Pointers**

- An array is actually a pointer to the 0th element of the array.
- Dereferencing the array name will give the 0th element.
- This gives us a range of equivalent notations for array access.
- In the following examples, arr is an array.

<table>
<thead>
<tr>
<th>Array</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr[0]</td>
<td>*arr</td>
</tr>
<tr>
<td>arr[1]</td>
<td>*arr[arr+1]</td>
</tr>
<tr>
<td>arr[n]</td>
<td>*arr[arr+n]</td>
</tr>
</tbody>
</table>

Table 1.4 Arrays and pointers

**1.10 Array**

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier. An array is a data structure of multiple elements with the same data type. Array elements are accessed using subscript. The valid range of subscript is 0 to size -1.

**Declaration of array**

```c
int arr[10];
```
**Example**

```c
#include
void main(void)
{
    int a[5];
    int i;
    for(i = 0;i<5;i++)
    {
        a[i]=i;
    }
    for(i = 0;i<5;i++)
    {
        printf("%d value of I is = %d
",i,a[i]);
    }
}
```

**Output**
1 value of I is = 0
2 value of I is = 1
3 value of I is = 2
4 value of I is = 3
5 value of I is = 4

**Multidimensional arrays**

A multi-dimensional array of dimension n (i.e., an n-dimensional array or simply n-D array) is a collection of items which is accessed via n subscript expressions. Multidimensional arrays can be described as “arrays of arrays”.

**Example**

```c
#include
void main(void){
    int a[3][2];
    int i,j;
    for(i = 0;i<3;i++)
    {
        for(j=0;j<2 ;j++) {
            scanf("%d",&a[i][j]);       
        }
    }
    for(i = 0;i<3;i++)
    {
        for(j=0;j<2;j++) {
            printf("value in array %d
",a[i][j]);
        }
    }
}
```
1.11 String

Character type array is called string. All strings end with the NULL character. Use the %s placeholder in the printf() function to display string values.

Declaration

char name[50];

Example

```c
#include
void main ( void )
{
    char *st1 = “abcd”;
    char st2[] = “efgh”;
    printf( “%s
”, st1);
    printf( “%s
”, st2);
}

void main(void){
    printf(“%s
”,myname);
}
```

String input and output

- The gets function relieves the string from standard input device while put S outputs the string to the standard output device.
- The function gets accepts the name of the string as a parameter, and fills the string with characters that are input from the keyboard till newline character is encountered.
- The puts function displays the contents stored in its parameter on the standard screen.
- The syntax of the gets function is

```
gets (str_var);
```

The syntax of the puts character is

```
puts (str_var);
```

Example

```c
#include <stdio.h>
Void main ( )
{
    char myname [40];
    printf (“Type your Name :”);
    gets (myname);
    printf (“Your name is :”);
    puts(mynamer);
}```
### Some string functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcpy(string1, string2)</td>
<td>Copy string2 into string1</td>
</tr>
<tr>
<td>strcat(string1, string2)</td>
<td>Concatenate string2 into the end of string1</td>
</tr>
<tr>
<td>length= strlen(string)</td>
<td>Get the length of a string</td>
</tr>
<tr>
<td>strcmp(string1, string2)</td>
<td>Return 0 if string1 equals string2, otherwise nonzero</td>
</tr>
<tr>
<td>strchr(string1,chr);</td>
<td>Will find the first matching character in a string.</td>
</tr>
</tbody>
</table>

### 1.12 Structure and Union

A structure is a collection of variables under a single name. These variables can be of different types, and each has a name which is used to select it from the structure. A structure is a convenient way of grouping several pieces of related information together.

#### Declaring structures

```c
struct mystruct
{
    int numb;
    char ch;
}
```

Structure has name `mystruct` and it contains two variables: an integer named `numb` and a character named `ch`.

#### Declaring structure variable

```c
struct mystruct s1;
```

#### Accessing Member Variables

```c
s1.numb=12;
s1.ch='b';
```

```c
printf("s1.numb=%d",s1.numb);
printf("s1.ch=%c",s1.ch);
```

typedef can also be used with structures. The following creates a new type `sb` which is of type `struct chk` and can be initialised as usual:

```c
typedef struct chk
{
    char name[50];
    int magazinesize;
    float calibre;
}

sb;
    ab arnies={"adam",30,7};
```
Unions

- A union is an object that can hold any one of a set of named members.
- The members of the named set can be of any data type.
- Members are overlaid in storage.
- The storage allocated for a union is the storage required for the largest member of the union, plus any padding required for the union to end at a natural boundary of its strictest member.

```c
union {
    char n;
    int age;
    float weight;
} people;

people.n = 'g';
people.age = 26;
people.weight = 64;
```

- A file is a collection of bytes stored on a secondary storage device, which is generally a disk of some kind.
- The collection of bytes may be interpreted, for example, as characters, words, lines, paragraphs and pages from a textual document; fields and records belonging to a database; or pixels from a graphical image.
- There are two kinds of files that programmers deal with: text files and binary files.

Text files

- A text file can be a stream of characters that a computer can process sequentially.
- It is not only processed sequentially but only in forward direction.
- For this reason a text file is usually opened for only one kind of operation (reading, writing, or appending) at any given time.

Binary files

- A binary file is no different to a text file.
- It is a collection of bytes.
- In C Programming Language a byte and a character are equivalent.
- No special processing of the data occurs and each byte of data is transferred to or from the disk unprocessed.
- C Programming Language places no constructs on the file, and it may be read from, or written to, in any manner chosen by the programmer.

Opening a file

The general format of the function used for opening a file is

```c
FILE *fp;
fp=fopen("filename","mode");
```

- The first statement declares the variable fp as a pointer to the data type FILE.
- As stated earlier, File is a structure that is defined in the I/O Library.
- The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp. fopen() contain the file name and mode (the purpose of opening the file).
- r is used to open the file for read only.


- `w` is used to open the file for writing only.
- `a` is used to open the file for appending data to it.

### Closing a file

A file must be closed as soon as all operations on it have been completed. This would close the file associated with the file pointer. The input output library supports the function to close a file.

#### Syntax to close file

```c
fclose(filepointer);
```

#### Example

```c
#include
void main(void)
{
    FILE *myfile;
    char c;
    myfile = fopen(“firstfile.txt”, “r”);
    if (myfile == NULL) printf(“File doesn’t exist
”);
    else {
        do {
            c = getc(myfile);
            putchar(c);
        } while (c != EOF);
    }
fclose(myfile);
}
```

#### File operations function in C:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>fopen()</td>
<td>Create a new file. Opens an existing file</td>
</tr>
<tr>
<td>fclose</td>
<td>Close a file which has been opened for use</td>
</tr>
<tr>
<td>getc()</td>
<td>Reads a character from a file</td>
</tr>
<tr>
<td>putc()</td>
<td>Writes a character to a file</td>
</tr>
<tr>
<td>fprintf()</td>
<td>Writes a set of data values to a file</td>
</tr>
<tr>
<td>fscanf()</td>
<td>Reads a set of data values from a file</td>
</tr>
<tr>
<td>getw()</td>
<td>Reads a integer from a file</td>
</tr>
<tr>
<td>putw()</td>
<td>Writes an integer to the file</td>
</tr>
<tr>
<td>fseek()</td>
<td>Sets the position to a desired point in the file</td>
</tr>
<tr>
<td>ftell()</td>
<td>Give the current position in the file</td>
</tr>
<tr>
<td>rewind()</td>
<td>Sets the position to the beginnings of the file</td>
</tr>
</tbody>
</table>

**Table 1.6 File operation**
Summary

- C was developed by Dennis Ritchie at Bell Laboratories in 1972.
- Most of its principles and ideas were taken from the earlier language B, BCPL and CPL.
- CPL was developed jointly between the Mathematical Laboratory at the University of Cambridge and the University of London Computer Unit in 1960s.
- CPL (Combined Programming Language) was developed with the purpose of creating a language that was capable of both, machine independent programming and would allow the programmer to control the behaviour of individual bits of information. But the CPL was too large for use in many applications.
- Press ALT+F9 to compile your program. If you have any error in your program, you will get the message, remove your errors and then execute your program you will got the out put.
- The printf() function prints output to stdout, according to format and other arguments passed to printf().
- The scanf() function reads input from stdin, according to the given format, and stores the data in the other arguments. It works a lot like printf().
- The alphabets, numbers and special symbols when properly combined form constants, variables and keywords. A constant is an entity that does not change.
- A variable is an entity that may change it value. In any program we typically do lots of calculations.
- A keyword is a word that is part of C Language itself. These words have predefined meanings and these words cannot be used as variable names.
- There are two main types of variables in C: numeric variables that hold only numbers or values, and string variables that hold text, from one to several characters long.
- Integer used to define inter numbers, no decimal points are used, takes 16 bytes of memory to store the value on 16 bit machine an integer can be positive or negative.
- Double is used to define big floating points. It is used to get the store on 16 bit machine, it requires 64 bytes of memory.
- Do-while is very similar to the while loop except that the test occurs at the end of the loop body.
- C language possesses decision making capabilities and supports the control or decision-making statements.
- The if statement is a powerful decision making statement and is used to control the flow of execution of statements.
- The switch case statements help control complex conditional and branching operations.
- The switch statement transfers control to a statement within its body.

References


Recommended Reading

Self Assessment

1. _______ was developed with the purpose of creating a language that was capable of both machine
   independent programming.
   a. Combined Programming Language
   b. Collective Programming Language
   c. Communal Programming Language
   d. Coalesce Programming Language

2. ____ language was written for the systems programming.
   a. A
   b. B
   c. C
   d. D

3. The original PDP-11 version of the _______ was developed in assembly language.
   a. linux system
   b. cobol system
   c. solar system
   d. unix system

4. Press _______ to compile your program.
   a. ALT+F4
   b. ALT+F10
   c. ALT+F9
   d. ALT+F8

5. The printf() function prints output to stdout, according to format and other arguments passed to______.
   a. void()
   b. main()
   c. printf()
   d. scanf()

6. The scanf() function reads input from _______.
   a. stdi
   b. stdn
   c. stdop
   d. stdin

7. A constant is an entity that does not__________.
   a. adjust
   b. transform
   c. alteration
   d. change
8. A __________ is an entity that may change its value.
   a. variable
   b. constant
   c. uneven
   d. capricious

9. A keyword is a word that is part of _____ Language itself.
   a. C
   b. C++
   c. C#
   d. C

10. __________ are used to repeat one statement or set statements more than one time.
    a. Loops
    b. Rings
    c. Circles
    d. Rounds


Chapter II
Solution of Equations and Eigen Value Problems

Aim

The aim of this chapter is to:

- explain iterative methods
- draw out Newton-Raphson method
- discuss general iteration method

Objectives

The objectives of this chapter are to:

- state the concept of eigenvalues
- illustrate about simple root
- explain matrices used in eigenvalues calculation

Learning outcome

At the end of this chapter, the students will be able to:

- define eigen value
- discuss multiple root
- describe direct methods
2.1 Definitions of Eigenvalues

Definition 1:
A scalar associated with a given linear transformation of a vector space and having the property that there is some nonzero vector which when multiplied by the scalar, is equal to the vector obtained by letting the transformation operate on the vector; especially: a root of the characteristic equation of a matrix.

Definition 2:
In mathematical analysis, one of the sets of discrete values of a parameter, k, in an equation of the form Lx = kx. Such characteristic equations are particularly useful in solving differential equations, integral equations, and systems of equations. In the equation, L is a linear transformation such as a matrix or a differential operator, and x can be a vector or a function (called an eigenvector or eigenfunction). The totality of eigenvalues for a given characteristic equation is a set. In quantum mechanics, where L is an energy operator, the eigenvalues are energy values.

Definition 3:
The one of the scalars λ such that T(v) = λv, where T is a linear operator on a vector space, and v is an eigenvector. Also known as characteristic number; characteristic root; characteristic value; latent root; proper value.

2.2 Introduction to Eigenvalues

Eigenvalues are special sets of scalars associated with a linear system of equations (i.e., a matrix equation) that are sometimes also known as characteristic roots, characteristic values (Hoffman and Kunze 1971), proper values, or latent roots (Marcus and Minc 1988).

- The determination of the eigenvalues and eigenvectors of a system is extremely important in physics and engineering, where it is equivalent to matrix diagonalization and arises in such common applications as stability analysis, the physics of rotating bodies, and small oscillations of vibrating systems, to name only a few.
- Each Eigenvalue is paired with a corresponding so-called eigenvector (or, in general, a corresponding right eigenvector and a corresponding left eigenvector; there is no analogous distinction between left and right for eigenvalues).
- The decomposition of a square matrix ‘A’ into eigenvalues and eigenvectors is known in this work as Eigen decomposition, and the fact that this decomposition is always possible as long as the matrix consisting of the eigenvectors of ‘A’ is square is known as the Eigen decomposition theorem.
- The Lanczos algorithm is an algorithm for computing the eigenvalues and eigenvectors for large symmetric sparse matrices.
- Let ‘A’ be a linear transformation represented by a matrix ‘A’. If there is a vector $X \in \mathbb{R}^n$ such that

$$AX = \lambda X$$  \hspace{1cm} (1)

- For some scalar $\lambda$, then $\lambda$ is called the eigenvalues of $A$ with corresponding (right) eigenvector.
- Letting $A$ be a $k \times k$ square matrix

$$\begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1k} \\
  \vdots & \ddots & \vdots \\
  a_{k1} & a_{k2} & \cdots & a_{kk}
\end{bmatrix}
$$

(2)

- With eigenvalues $\lambda$, then the corresponding eigenvectors satisfy

$$\begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1k} \\
  \vdots & \ddots & \vdots \\
  a_{k1} & a_{k2} & \cdots & a_{kk}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_k
\end{bmatrix}
= \lambda
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_k
\end{bmatrix}
$$  \hspace{1cm} (3)

- This is equivalent to the homogeneous system
Equation (4) can be written compactly as

\[(A-\lambda I)X=0\]  

Where ‘I’ is an identity matrix. As shown in Cramer’s rule, a linear system of equations has nontrivial solutions iff (to understand concept of iff please refer Para given below) the determinant vanishes, so the solutions of equation (5) are given by

\[\text{def}(A-\lambda I)=0\]

Note on iff:
If and only if (i.e., necessary and sufficient). The terms “just if” or “exactly when” are sometimes used instead. A iff B is written symbolically as A \iff B, A \implies B, A \impliedby B, or A \equiv B. A iff B is also equivalent to A \implies B together with B \implies A, where the symbol “ \iff ” denotes “implies.” Other equivalent terms are “A is equivalent to B” (A \equiv B) and “A XNOR B”

This equation is known as the characteristic equation of A, and the left-hand side is known as the characteristic polynomial. For example, for a 2x2 matrix, the eigenvalues are

\[\lambda = \frac{1}{2} \left[ (a_{11} a_{22}) \pm \sqrt{4 a_{12} a_{21} + (a_{11} - a_{22})^2} \right]\]

\[x^2 - x(a_{11} + a_{22}) \pm (a_{11} a_{22} - a_{12} a_{21}) = 0\]

If all k eigenvalues are different, then plugging these back in gives k-1 independent equations for the k components of each corresponding eigenvector, and the system is said to be nondegenerate. If the eigenvalues are n-fold degenerate, then the system is said to be degenerate and the eigenvectors are not linearly independent.

In such cases, the additional constraint that the eigenvectors be orthogonal,

\[X_i \cdot X_j = \delta_{ij}\]

Where \(\delta_{ij}\) is the Kronecker delta, can be applied to yield “n” additional constraints, thus allowing solution for the eigenvectors.

Eigenvalues may be computed in Mathematic using Eigenvalues[\text{matrix}]. Eigenvectors and eigenvalues can be returned together using the command Eigensystem[\text{matrix}].

Assume we know the eigenvalues for

\[AX=\lambda X\]

Adding a constants times the identity matrix to A,

\[(A+cI)X=(\lambda+c)X=\lambda \cdot X\]

So the new eigenvalues equal the old plus c. Multiplying A by a constant c.
Now consider a similarity transformation of A. Let $|A|$ be the determinant of A, then

$$Z^{-1}AZ - \lambda I = Z^{-1}(A - \lambda I)Z$$

$$= |Z||Z - \lambda I||Z^{-1}|Z||Z - \lambda I||Z^{-1}|$$

$$= |A - \lambda I||A - \lambda I|$$

So, the eigenvalues are the same as for A.

2.3 Solution to Algebraic Equations

A problem of great importance in science and engineering is that of determining the roots/zeros of an equation of the form

$$f(x) = 0$$

A polynomial equation of the form

$$f(x) = P_n(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + ... + a_{n-1}x + a_n = 0$$

It is called an algebraic equation. An equation which contains polynomials, exponential functions, logarithmic functions, trigonometric functions etc. is called a transcendental equation.

For example,

$$3x^3 - 2x^2 - x - 5 = 0, x^4 - 3x^2 + 1 = 0, x^2 - 3x + 1 = 0,$$

Above are algebraic (polynomial) equations and

$$xe^{2x} - 1 = 0, \cos x - xe^x = 0, \tan x = x$$

Above mentioned are transcendental equations.

We assume that the function $f(x)$ is continuous in the required interval.

2.4 Root/Zero

A number $\alpha$, for which $f(\alpha) = 0$ is called a root of the equation $f(x) = 0$, or a zero of $f(x)$.

Geometrically, a root of an equation $f(x) = 0$ is the value of $x$ at which the graph of the equation $y = f(x)$ intersects the x-axis.

Refer the figure given below.
2.5 Simple Root

A number $\alpha$ is a simple root of $f(x) = 0$, if $f(\alpha) = 0$ and $f'(\alpha) \neq 0$. Then, we can write $f(x)$ as

$$f(x) = (x - \alpha) g(x), \ g(\alpha) \neq 0(4)$$

- For example, since $(x - 1)$ is a factor of $f(x) = x^3 + x - 2 = 0$, we can write

$$f(x) = (x - 1)(x^2 + x + 2) = (x - 1) g(x), \ g(1) \neq 0. \ (2)$$

- Alternately, we find $f(1) = 0$, $f'(x) = 3x^2 + 1$, $f'(1) = 4 \neq 0$. Hence, $x = 1$ is a simple root of

$$f(x) = x^3 + x - 2 = 0 \ (3)$$

2.6 Multiple Root

- A number $\alpha$ is a multiple root, of multiplicity $m$, of $f(x) = 0$, if

$$f(\alpha) = 0, \ f'(\alpha) = 0, ..., f^{(m-1)}(\alpha) = 0, \text{ and } f^{(m)}(\alpha) \neq 0 \ (1)$$

Then, we can write $f(x)$ as

$$f(x) = (x - \alpha)^m g(x), \ g(\alpha) \neq 0. \ (2)$$

- For example, consider the equation $f(x) = x^3 - 3x^2 + 4 = 0$. We find

$$f(2) = 8 - 12 + 4 = 0, \ f'(x) = 3x^2 - 6x, \ (3)$$

$$f'(2) = 12 - 12 = 0, \ (4)$$

$$f''(x) = 6x - 6, \ f''(2) = 6 \neq 0 \ (5)$$

Hence, $x = 2$ is a multiple root of multiplicity 2 (double root) of

$$f(x) = x^3 - 3x^2 + 4 = 0. \ (6)$$

- We can write

$$f(x) = (x - 2)^2 (x + 1) = (x - 2)^2 g(x), \ g(2) = 3 \neq 0. \ (7)$$

- In this chapter, we shall be considering the case of simple roots only.

Remark 1

- A polynomial equation of degree “n” has exactly $n$ roots, real or complex, simple or multiple, where as a transcendental equation may have one root, infinite number of roots or no root.

- We shall derive methods for finding only the real roots.

- The methods for finding the roots are classified as
  (i) Direct methods
  (ii) Iterative methods
2.7 Direct Methods

- These methods give the exact values of all the roots in a finite number of steps (disregarding the round-off errors).
- Therefore, for any direct method, we can give the total number of operations (additions, subtractions, divisions and multiplications).
- This number is called the operational count of the method.
- For example, the roots of the quadratic equation \( ax^2 + bx + c = 0, \ a \neq 0 \), can be obtained using the method.

\[
x = \frac{1}{2a} \left[ -b \pm \sqrt{b^2 - 4ac} \right] = \frac{1}{2a} \left[ -b \pm \sqrt{b^2 - 4ac} \right]
\]

- For this method, we can give the count of the total number of operations.
- There are direct methods for finding all the roots of cubic and fourth degree polynomials.
- However, these methods are difficult to use.
- Direct methods for finding the roots of polynomial equations of degree greater than 4 or transcendental equations are not available in literature.

2.8 Iterative Methods

- These methods are based on the idea of successive approximations.
- We start with one or two initial approximations to the root and obtain a sequence of approximations \( x_0, x_1, ..., x_k, \), which in the limit as \( k \to \infty \), converge to the exact root \( \alpha \).
- An iterative method for finding a root of the equation \( f(x) = 0 \) can be obtained as

\[
x_{k+1} = \phi(x_k), \ k = 0, 1, 2, ...
\]

- This method uses one initial approximation to the root \( x_0 \).
- The sequence of approximations is given by

\[
x_1 = \phi(x_0), \ x_2 = \phi(x_1), \ x_3 = \phi(x_2), ....
\]

- The function \( \phi \) is called an iteration function and \( x_0 \) is called an initial approximation.
- If a method uses two initial approximations \( x_0, x_1 \), to the root, then we can write the method as

\[
x_{k+1} = \phi(x_{k-1}, x_k), \ k = 1, 2, ....
\]

2.9 Convergence of Iterative Methods

- The sequence of iterates, \( \{x_k\} \), is said to converge to the exact root \( \alpha \), if

\[
\lim_{k \to \infty} x_k = \text{or } \lim_{k \to \infty} |x_k - \alpha| = 0.
\]

- The error of approximation at the kth iterate is defined as \( \varepsilon_k = x_k - \alpha \). Then, we can write above equation as

\[
\lim_{k \to \infty} |\text{error of approximation}| = \lim_{k \to \infty} |x_k - \alpha| = \lim_{k \to \infty} |\varepsilon_k| = 0, \quad (2)
\]

Remark 2
Given one or two initial approximations to the root, we require a suitable iteration function \( \phi \) for a given function \( f(x) \), such that the sequence of iterates, \( \{x_k\} \), converge to the exact root \( \alpha \). Further, we also require a suitable criterion to terminate the iteration.

2.10 Criterion to Terminate Iteration Procedure

- Since we cannot perform infinite number of iterations; we need a criterion to stop the iterations.
- We use one or both of the following criterion:
The equation \( f(x) = 0 \) is satisfied to a given accuracy or \( f(x_k) \) is bounded by an error tolerance \( \varepsilon \).
\[
| f(x_k) | \leq \varepsilon. \tag{1}
\]

The magnitude of the difference between two successive iterates is smaller than a given accuracy or an error bound \( \varepsilon \).
\[
| x_{k+1} - x_k | \leq \varepsilon \tag{2}
\]

- Generally, we use the second criterion.
- In some very special problems, we require to use both the criteria.
- For example, if we require two decimal place accuracy, then we iterate until \( | x_{k+1} - x_k | < 0.005 \).
- If we require three decimal place accuracy, then we iterate until \( | x_{k+1} - x_k | < 0.0005 \).
- As we have seen earlier, we require a suitable iteration function and suitable initial approximation (s) to start the iteration procedure.
- In the next section, we give a method to find initial approximation(s).

### 2.11 Initial Approximation for an Iterative Procedure

For polynomial equations, Descarte’s rule of signs gives the bound for the number of positive and negative real roots.

- We count the number of changes of signs in the coefficients of \( P_n(x) \) for the equation \( f(x) = P_n(x) = 0 \). The number of positive roots cannot exceed the number of changes of signs. For example, if there are four changes in signs, then the equation may have four positive roots or two positive roots or no positive root. If there are three changes in signs, then the equation may have three positive roots or definitely one positive root. (For polynomial equations with real coefficients, complex roots occur in conjugate pairs.)

- We write the equation \( f(-x) = P_n(-x) = 0 \), and count the number of changes of signs in the coefficients of \( P_n(-x) \). The number of negative roots cannot exceed the number of changes of signs. Again, if there are four changes in signs, then the equation may have four negative roots or two negative roots or no negative root. If there are three changes in signs, then the equation may have three negative roots or definitely one negative root.

We use the following theorem of calculus to determine an initial approximation. It is also called the intermediate value theorem.

- We use the following theorem of calculus to determine an initial approximation. It is also called the intermediate value theorem.

### 2.12 Method of False Position

The method is also called linear interpolation method or chord method or regula-falsi method.

- At the start of all iterations of the method, we require the interval in which the root lies.
- Let the root of the equation \( f(x) = 0 \), lie in the interval \( (x_{k-1}, x_k) \), that is, \( f_{k-1} = f_k < 0 \), where \( f(x_{k-1}) = f_{k-1} \), and \( f(x_k) = f_k \).

- Then, \( P(x_{k-1}, f_{k-1}), Q(x_k, f_k) \) are points on the curve \( f(x) = 0 \). Draw a straight line joining the points P and Q (Figs. given below).

The line PQ is taken as an approximation of the curve in the interval \( (x_{k-1}, x_k) \). The equation of the line PQ is given by
\[
\frac{x - x_k}{x_k - x_k} = \frac{f(x) - f_k}{f_{k-1} - f_k}.
\]

- The point of intersection of this line PQ with the x-axis is taken as the next approximation to the root.

Setting \( y = 0 \), and solving for \( x \), we get
\[
x = x_k \left( \frac{\frac{x_k - x_{k-1}}{f_{k-1} - f_k} f_k}{\frac{x_k - x_{k-1}}{f_{k-1} - f_k} f_k - f_{k-1}} \right) = \left( \frac{x_k - x_{k-1}}{f_{k-1} - f_k} \right) \frac{f_k}{f_{k-1} - f_k} - \left( \frac{\frac{x_k - x_{k-1}}{f_{k-1} - f_k}}{\frac{x_k - x_{k-1}}{f_{k-1} - f_k} f_k - f_{k-1}} \right) f_k
\]
The next approximation to the root is taken as
\[ x_{k+1} = x_k - \frac{f_k(x_k - x_{k-1})}{f_k - f_{k-1}} \]
(1)

Simplifying, we can also write the approximation as
\[ x_{k+1} = \frac{x_k f_k - f_{k-1} x_{k-1}}{f_k - f_{k-1}} \]
(2)

Therefore, starting with the initial interval \((x_0, x_1)\), in which the root lies, we compute
\[ x_2 = \frac{x_0 f_0 - x_1 f_{-1}}{f_1 - f_0} \]

Now, if \(f(x_0) f(x_2) < 0\), then the root lies in the interval \((x_0, x_2)\). Otherwise, the root lies in the interval \((x_2, x_1)\).

The iteration is continued using the interval in which the root lies, until the required accuracy criterion given in equation (1) and (2) is satisfied.

Alternate derivation of the method

Let the root of the equation \(f(x) = 0\), lie in the interval \((x_{k-1}, x_k)\). Then, \(P(x_{k-1}, f_{k-1})\), \(Q(x_k, f_k)\) are points on the curve \(f(x) = 0\).

Draw the chord joining the points \(P\) and \(Q\) (Refer the figures given below).

We approximate the curve in this interval by the chord, that is, \(f(x) \approx ax + b\).

The next approximation to the root is given by \(x = -b/a\).

Since the chord passes through the points \(P\) and \(Q\), we get
\[ f_{k-1} = ax_{k-1} + b \]
and
\[ f_k = ax_k + b \]

Subtracting the two equations, we get
\[ f_k - f_{k-1} = a(x_k - x_{k-1}) \]
or
\[ a = \frac{f_k - f_{k-1}}{x_k - x_{k-1}} \]

The second equation gives
\[ b = f_{k-1} - ax_{k-1} \]

Hence, the next approximation is given by
\[ x_{k+1} = \frac{b}{a} = \frac{f_k - ax_k}{f_k - f_{k-1}} \]

This is same as the method given in equation (1).

Fig. 2.2 False position

![Fig. 2.2 False position](image-url)
Remark 3
At the start of each iteration, the required root lies in an interval, whose length is decreasing. Hence, the method always converges.

Remark 4
The method of false position has a disadvantage. If the root lies initially in the interval \((x_0, x_1)\), then one of the end points is fixed for all iterations. For example, in Fig. 2.2a, the left end point \(x_0\) is fixed and the right end point moves towards the required root. Therefore, in actual computations, the method behaves

\[ X_{k+1} = \frac{x_0 f_k - x_k f_0}{f_k - f_0}, \quad k=1,2,3,.... \] (3)

In Fig. 2.2b, the right end point \(x_1\) is fixed and the left end point moves towards the required root. Therefore, in this case, in actual computations, the method behaves

\[ X_{k+1} = \frac{x_k f_1 - x_1 f_k}{f_1 - f_k}, \quad k=1,2,3,.... \] (4)

Remark 5
The computational cost of the method is one evaluation of the function \(f(x)\), for each iteration.

Remark 6
We would like to know why the method is also called a linear interpolation method. Graphically, a linear interpolation polynomial describes a straight line or a chord. The linear interpolation polynomial that fits the data \((x_{k-1}, f_{k-1}), (x_k, f_k)\) is given by

\[ f(x) = \frac{x-x_k}{x_{k-1}-x_k} f_{k-1} + \frac{x-x_{k-1}}{x_k-x_{k-1}} f_k \]

Setting \(f(x) = 0\), we get

\[ \frac{(x - x_{k-1}) f_k - (x - x_k) f_{k-1}}{x_k - x_{k-1}} = 0 \]

Or

\[ x(f_k - f_{k-1}) = x_{k-1} f_k - x_k f_{k-1} \]

Or

\[ x = x_k + 1 = \frac{x_{k-1} f_k - x_k f_{k-1}}{f_k - f_{k-1}} \]

This given the next approximation as given in equation (2)

2.13 Newton-Raphson Method

• This method is also called Newton’s method.
• This method is also a chord method in which we approximate the curve near a root, by a straight line.
• Let \(x_0\) be an initial approximation to the root of \(f(x) = 0\).
• Then, \(P(x_0, f_0)\), where \(f_0 = f(x_0)\), is a point on the curve.
• Draw the tangent to the curve at \(P\), (Fig. 2.3).
• We approximate the curve in the neighbourhood of the root by the tangent to the curve at the point \(P\).
• The point of intersection of the tangent with the x-axis is taken as the next approximation to the root.
• The process is repeated until the required accuracy is obtained.
• The equation of the tangent to the curve \( y = f(x) \) at the point \( P(x_0, f(x_0)) \) is given by

\[
y - f(x_0) = (x - x_0) f'(x_0)
\]

![Fig. 2.3 Network-Raphson method](image)

Where \( f'(x_0) \) is the slope of the tangent to the curve at \( P \).

• Setting \( y = 0 \) and solving for \( x \), we get

\[
x = x_0 - \frac{f(x_0)}{f'(x_0)} f'(x_0) \neq 0
\]

• The next approximation to the root is given by

\[
x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} f'(x_0) \neq 0
\]

• We repeat the procedure. The iteration method is defined as

\[
x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} f'(x_k) = 0
\]

This method is called the Newton-Raphson method or simply the Newton’s method. The method is also called the tangent method.

Alternate derivation of the method

• Let \( x_k \) be an approximation to the root of the equation \( f(x) = 0 \). Let \( \Delta x \) be an increment in \( x \) such that \( x_k + \Delta x \) is the exact root, that is \( f(x_k + \Delta x) \equiv 0 \).

Expanding in Taylor’s series about the point \( x_k \), we get

\[
f(x_k) + \Delta x f'(x_k) + \frac{(\Delta x)^2}{2!} f''(x_k) + ... = 0.
\]

Neglecting the second and higher powers of \( \Delta x \), we obtain.

\[
f(x_k) + \Delta x f'(x_k) + \frac{(\Delta x)^2}{2!} f''(x_k) + ... = 0
\]

Hence, we obtain the iteration method

\[
x_{k+1} = x_k + \Delta x = x_k - \frac{f(x_k)}{f'(x_k)} f'(x_k) \neq 0, \quad k = 0, 1, 2, 3, ...
\]
This is same as the method derived earlier.

**Remark 7**

- Convergence of the Newton’s method depends on the initial approximation to the root.
- If the approximation is far away from the exact root, the method diverges.
- However, if a root lies in a small interval \((a, b)\) and \(x_0 \in (a, b)\), then the method converges.

**Remark 8**

- We observe that the method may fail when \(f'(x)\) is close to zero in the neighbourhood of the root.
- Later, in this section, we shall give the condition for convergence of the method.

**Remark 9**

- The computational cost of the method is one evaluation of the function \(f(x)\) and one evaluation of the derivative \(f'(x)\), for each iteration.

**Example**

Derive the Newton’s method for finding \(1/N\), where \(N > 0\). Hence, find \(1/17\), using the initial approximation as (i) 0.05, (ii) 0.15. Do the iterations converge?

**Solution**

Let \(x = \frac{1}{N}\), or \(\frac{1}{N} = N\)

Define \(f(x) = \frac{1}{x} - N\)

Then,

\[ f'(x) = -\frac{1}{x^2} \]

Newton’s method gives

\[ x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} = x_k - \frac{(\frac{1}{x_k}) - N}{-\frac{1}{x_k^2}} = x_k + \left[ x_k - N x_k \right] = 2x_k - N x_k^2 \]

(i) With \(N = 17\), and \(x_0 = 0.05\), we obtain the sequence of approximations

\[ x_1 = 2x_0 - N x_0^2 = 2(0.05) - 17(0.05)^2 = 0.0575. \]
\[ x_2 = 2x_1 - N x_1^2 = 2(0.0575) - 17(0.0575)^2 = 0.058794. \]
\[ x_3 = 2x_2 - N x_2^2 = 2(0.058794) - 17(0.058794)^2 = 0.058823. \]
\[ x_4 = 2x_3 - N x_3^2 = 2(0.058823) - 17(0.058823)^2 = 0.058823. \]

Since, \(|x_4 - x_3| = 0\), the iterations converge to the root. The required root is 0.058823.

(ii) With \(N = 17\), and \(x_0 = 0.15\), we obtain the sequence of approximations

\[ x_1 = 2x_0 - N x_0^2 = 2(0.15) - 17(0.15)^2 = -0.0825. \]
\[ x_2 = 2x_1 - N x_1^2 = 2(-0.0825) - 17(-0.0825)^2 = -0.280706. \]
\[ x_3 = 2x^2 - Nx^2 = 2(-0.280706) - 17(-0.280706)^2 = -1.900942. \]

\[ x_4 = 2x^3 - Nx^3 = 2(-1.900942) - 17(-1.900942)^2 = -65.23275. \]

- We find that \( x_k \to -\infty \) as \( k \) increases. Therefore, the iterations diverge very fast. This shows the importance of choosing a proper initial approximation.

### 2.14 General Iteration Method

- The method is also called iteration method or method of successive approximations or fixed point iteration method.
- The first step in this method is to rewrite the given equation \( f(x) = 0 \) in an equivalent form as,
\[ x = \phi(x) \quad (1) \]
- There are many ways of rewriting \( f(x) = 0 \) in this form.
- For example, \( f(x) = x^3 - 5x + 1 = 0 \), can be rewritten in the following forms.

\[ x = \frac{x^3+1}{5}, \quad x = \left(5x + 1\right)^{1/3} \]
\[ x = \sqrt[3]{\frac{2x^3-1}{x}}, \text{etc} \quad (2) \]
- Now, finding a root of \( f(x) = 0 \) is same as finding a number \( \alpha \) such that \( \alpha = \phi(\alpha) \), that is,
- a fixed point of \( \phi(x) \).
- A fixed point of a function \( \phi \) is a point \( \alpha \) such that \( \alpha = \phi(\alpha) \). This result is also called the fixed point theorem.
- Using above equation, the iteration method is written as
\[ x_{k+1} = \phi(x_k), \quad k = 0, 1, 2, \ldots \quad (3) \]
- The function \( \phi(x) \) is called the iteration function.
- Starting with the initial approximation \( x_0 \), we compute the next approximations as
\[ x_1 = \phi(x_0), \quad x_2 = \phi(x_1), \quad x_3 = \phi(x_2), \ldots \]
- The stopping criterion is same as used earlier.
- Since there are many ways of writing \( f(x) = 0 \) as \( x = \phi(x) \), it is important to know whether all or at least one of these iteration methods converges.

**Remark 10**
Convergence of an iteration method \( x_{k+1} = \phi(x_k), \quad k = 0, 1, 2, \ldots \), depends on the choice of the iteration function \( \phi(x) \), and a suitable initial approximation \( x_0 \) to the root.

- Consider again, the iteration methods given in Eq. 2 for finding a root of the equation \( f(x) = x^3 - 5x + 1 = 0 \). The positive root lies in the interval \((0, 1)\).

\[ \text{i) } x_{k+1} = \frac{x_k^3+1}{5}, \quad k=0,1,2,3,4,5,\ldots \quad (4) \]
• With $x_0 = 1$, we get the sequence of approximations as
  
  $x_1 = 0.4, x_2 = 0.2128, x_3 = 0.20193, x_4 = 0.20165, x_5 = 0.20164.$

• The method converges and $x \approx x_5 = 0.20164$ is taken as the required approximation to the root.

ii) $x_{k+1} = (5x_k - 1)^{1/3}, k = 0, 1, 2, ... $ (5)

• With $x_0 = 1$, we get the sequence of approximations as
  
  $x_1 = 1.5874, x_2 = 1.9072, x_3 = 2.0437, x_4 = 2.0968,...$

• which does not converge to the root in $(0, 1)$.

iii) $x_{k+1} = \frac{2x_k}{x_k^2}, k = 0, 1, 2, 3, 4, ...$ (6)

• With $x_0 = 1$, we get the sequence of approximations as
  
  $x_1 = 2.0, x_2 = 2.1213, x_3 = 2.1280, x_4 = 2.1284,...$

• Which does not converge to the root in $(0, 1)$.

• Now, we derive the condition that the iteration function $\phi(x)$ should satisfy in order that the method converges.

### 2.15 Condition of Convergence

The iteration method for finding a root of $f(x) = 0$, is written as

$$x_{k+1} = \phi(x_k), k = 0, 1, 2,...$$ (1)

Let $\alpha$ be the exact root. That is,

$$\alpha = \phi(\alpha).$$ (2)

We define the error of approximation at the kth iterate as $e_k = x_k - \alpha, k = 0, 1, 2,...$

Subtracting (1) and (2), we obtain

$$x_k - \alpha = \phi(x_k) - \phi(\alpha) = (x_k - \alpha)\phi'(t_k) \quad \text{(using the mean value theorem)}$$ (3)

or

$$e_{k+1} = \phi'(t_k) e_k, x_k < t_k < \alpha.$$

Setting $k = k - 1$, we get $e_k = \phi'(t_k) e_{k-1}, x_{k-1} < t_{k-1} < \alpha$.

Hence,

$$e_{k+1} = \phi'(t_k) \phi'(t_{k-1}) e_{k-1}$$

Using (3) recursively, we get

$$e_{k+1} = \phi'(t_k) \phi'(t_{k-1}) \cdots \phi'(t_0) e_0.$$

• The initial error $e_0$ is known and is a constant. We have

$$|e_{k+1}| = |\phi'(t_k)| \cdot |\phi'(t_{k-1})| \cdots |\phi'(t_0)| \cdot |e_0|.$$

Let $|\phi'(t_k)| \leq c, k = 0, 1, 2,...$

Then, $|e_{k+1}| \leq c e_{k+1} \cdot |e_0|.$ (4)

• For convergence, we require that $|e_{k+1}| \to 0$ as $k \to \infty$. This result is possible, if and only if $c < 1$. Therefore, the iteration method (1) converges, if and only if.
\[ |\phi'(x_k)| \leq c < 1, \text{ for } x \text{ in the interval } (a, b). \]  

or

\[ |\phi'(x)| \leq c < 1, \text{ for all } x \text{ in the interval } (a, b). \]  

- We can test this condition using \( x_0 \), the initial approximation, before the computations are done.
- Let us now check whether the methods (1.19), (1.20), (1.21) converge to a root in \((0, 1)\) of the equation \( f(x) = x^3 - 5x + 1 = 0 \).

  i) We have \( \phi(x) = x^3 + \), \( \phi'(x) = 3x^2 \), and \( |\phi'(x)| = 3x^2 \leq 1 \) for all \( x \) in \( 0 < x < 1 \). Hence, the method converges to a root in \((0, 1)\).

  ii) We have \( \phi(x) = (5x-1)^{1/3} \), \( \phi'(x) = \frac{\frac{5}{3} (5x-1)^{-2/3}}{x^{2/3}} \). Now \( |\phi'(x)| < 1 \), when \( x \) is close to 1 and \( |\phi'(x)| > 1 \) in the other part of the interval. Convergence is not guaranteed.

  iii) We have \( \phi(x) = \sqrt[3]{\frac{5x-1}{x}} \), \( \phi'(x) = \frac{1}{2x^{2/3}/(5x-1)^{1/3}} \). Again, \( |\phi'(x)| < 1 \), when \( x \) is close to 1 and \( |\phi'(x)| > 1 \) in the other part of the interval. Convergence is not guaranteed.

**Remark 11**

- Sometimes, it may not be possible to find a suitable iteration function \( \phi(x) \) by manipulating the given function \( f(x) \).
- Then, we may use the following procedure.
- Write \( f(x) = 0 \) as \( x = x + \alpha f(x) = \phi(x) \), where \( \alpha \) is a constant to be determined.
- Let \( x_0 \) be an initial approximation contained in the interval in which the root lies.
- For convergence, we require
  \[ |\phi'(x_0)| = |1 + \alpha f'(x_0)| < 1. \]  

Simplifying, we find the interval in which \( \alpha \) lies.
- We choose a value for \( \alpha \) from this interval and compute the approximations.
- A judicious choice of a value in this interval may give faster convergence.

**2.16 Convergence of the Iteration Methods**

- We now study the rate at which the iteration methods converge to the exact root, if the initial approximation is sufficiently close to the desired root.
- Define the error of approximation at the \( k \)th iterate as \( \varepsilon_k = x_k - \alpha \), \( k = 0, 1, 2, \ldots \).

  **Definition**

  An iterative method is said to be of order \( p \) or has the rate of convergence \( p \), if \( p \) is the largest positive real number for which there exists a finite constant \( C \neq 0 \), such that

  \[ |\varepsilon_{k+1}| \leq C |\varepsilon_k|^p. \]  

The constant \( C \), which is independent of \( k \), is called the asymptotic error constant and it depends on the derivatives of \( f(x) \) at \( x = \alpha \).

Let us now obtain the orders of the methods that were derived earlier.

**2.17 Method of False Position**

- We have noted earlier (see Remark 4) that if the root lies initially in the interval \((x_o, x_i)\), then one of the end points is fixed for all iterations.
- If the left end point \( x_o \) is fixed and the right end point moves towards the required root, the method behaves like.
Substituting $x_k = \varepsilon_k + \alpha$, $x_{k+1} = \varepsilon_{k+1} + \alpha$, $x_0 = \varepsilon_0 + \alpha$, we expand each term in Taylor’s series and simplify using the fact that $f(\alpha) = 0$. We obtain the error equation as

$$\varepsilon_{k+1} = C\varepsilon_0\varepsilon_k,$$

where $C = \frac{f''(\alpha)}{2f'(\alpha)}$.

Since $\varepsilon_n$ is finite and fixed, the error equation becomes

$$|\varepsilon_k| = |C^*| |\varepsilon_k|,$$

where $C^* = C\varepsilon_0$.

Hence, the method of false position has order 1 or has linear rate of convergence.

### 2.18 Method of Successive Approximations or Fixed Point Iteration Method

- We have $x_{k+1} = \phi(x_k)$, and $\alpha = \phi(\alpha)$
- Subtracting, we get
  $$x_{k+1} - \alpha = \phi(x_k) - \phi(\alpha) = \phi(\alpha + x_k - \alpha) - \phi(\alpha)$$
  $$= [\phi(\alpha) + (x_k - \alpha) \phi'(\alpha) + ...] - \phi(\alpha)$$

or

$$\varepsilon_{k+1} = \varepsilon_k \phi'(\alpha) + O(\varepsilon_k^2).$$

Therefore, $|\varepsilon_{k+1}| = C |\varepsilon_k|$, $x_k < \alpha$, and $C = |\phi'(\alpha)|$. Hence, the fixed point iteration method has order 1 or has linear rate of convergence.

### 2.19 Newton-Raphson Method

The method is given by

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, f(x_k) \neq 0$$

Substituting $x_k = \varepsilon_k + \alpha$, $x_{k+1} = \varepsilon_{k+1} + \alpha$, we obtain

$$\varepsilon_{k+1} + \alpha = \varepsilon_k + \alpha - \frac{f(\varepsilon_k + \alpha)}{f'(\varepsilon_k + \alpha)}.$$

Expand the term in Taylor’s series, using the fact that $f(\alpha) = 0$, and cancelling $f'(\alpha)$, we obtain

$$\varepsilon_{k+1} = \varepsilon_k - \frac{\varepsilon_k f''(\alpha)}{2f'(\alpha)} - \frac{\varepsilon_k^2 f'''(\alpha)}{6f'(\alpha)} - ...$$

Neglecting the terms containing $\varepsilon_k^2$ and higher powers of $\varepsilon_k$, we get

$$\varepsilon_{k+1} = C\varepsilon_k,$$

where $C = \frac{f''(\alpha)}{2f'(\alpha)}$.

Therefore, Newton’s method is order 2 or has quadratic rate of convergence.

**Remark 12**

What is the importance of defining the order or rate of convergence of a method?
• Suppose that we are using Newton’s method for computing a root of \( f(x) = 0 \).
• Let us assume that at a particular stage of iteration, the error in magnitude in computing the root is \( 10^{-1} = 0.1 \).
• We observe from above equation 1, that in the next iteration, the error behaves like \( C (0.1)^2 = C (10^{-2}) \).
• That is, we may possibly get an accuracy of two decimal places.
• Because of the quadratic convergence of the method, we may possibly get an accuracy of four decimal places in the next iteration.
• However, it also depends on the value of \( C \).
• From this discussion, we conclude that both fixed point iteration and regula-falsi methods converge slowly as they have only linear rate of convergence.
• Further, Newton’s method converges at least twice as fast as the fixed point iteration and regula-falsi methods.

Remark 13
When does the Newton-Raphson method fail?

(i) The method may fail when the initial approximation \( x_0 \) is far away from the exact root \( \alpha \). However, if the root lies in a small interval \((a, b)\) and \( x_0 \in (a, b) \), then the method converges.
(ii) From Newton’s-Raphson Methods equation number (1), we note that if \( f'(\alpha) \approx 0 \), and \( f''(x) \) is finite then \( C \to \infty \) and the method may fail. That is, in this case, the graph of \( y = f(x) \) is almost parallel to x-axis at the root \( \alpha \).

Remark 14

• Let us have a re-look at the error equation.
• We have defined the error of approximation at the kth iterate as \( \varepsilon_k = x_k - \alpha \), \( k = 0, 1, 2, \ldots \) From \( x_{k+1} = \phi(x_k) \), \( k = 0, 1, 2, \ldots \) and \( \alpha = \phi(\alpha) \).

\[
x_{k+1} - \alpha = \phi(x_k) - \phi(\alpha) = \phi(\alpha + \varepsilon_k) - \phi(\alpha) \\
= \left[ \phi(\alpha) + \phi'(\alpha) \varepsilon_k + \frac{1}{2} \phi''(\alpha) \varepsilon_k^2 + \ldots \right] - \phi(\alpha) \\
or \\
\varepsilon_{k+1} = a_1 \varepsilon_k + a_2 \varepsilon_k^2 + \ldots \quad (1)
\]

where \( a_1 = \phi'(\alpha) \), \( a_2 = (1/2) \phi''(\alpha) \), etc.

• The exact root satisfies the equation \( \alpha = \phi(\alpha) \).
• If \( a_1 \neq 0 \) that is, \( \phi'(\alpha) \neq 0 \), then the method is of order 1 or has linear convergence. For the
• General iteration method, which is of first order, we have derived that the condition of convergence is \( | \phi'(x) | < 1 \) for all \( x \) in the interval \((a, b)\) in which the root lies.
• Note that in this method, \( | \phi'(x) | \neq 0 \) for all \( x \) in the neighbourhood of the root \( \alpha \).
• If \( a_1 = \phi'(\alpha) = 0 \), and \( a_2 = (1/2) \phi''(\alpha) \neq 0 \), then from above mentioned Eq. (1), the method is of order 2 or has quadratic convergence.
• Let us verify this result for the Newton-Raphson method. For the Newton-Raphson method.

\[
x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad \text{we have} \quad \phi(x) = x - \frac{f(x)}{f'(x)}.
\]

Then,

\[
\phi'(x) = 1 - \frac{[f'(x)]^2 - f(x) f''(x)}{[f'(x)]^2} = \frac{f(x) f''(x)}{[f'(x)]^2}
\]

and

\[
\phi''(\alpha) = \frac{f'(\alpha) f''(\alpha)}{[f'(\alpha)]^2} = 0
\]
Since \( f(\alpha) = 0 \) and \( f'(\alpha) \neq 0 \) (\( \alpha \) is a simple root).
When, \( x_k \to \alpha, f(x_k) \to 0 \), we have \( |\phi'(x_k)| < 1, \ k = 1, 2, \ldots \) and \( \to 0 \) as \( n \to \infty \).

Now,
\[
\phi''(x) = \frac{1}{[f'(x)]^3} \left[ f''(x) (f'(x))^2 + f(x) f'(x) - 2 f'(x) f''(x) \right]
\]
and
\[
\phi''(\alpha) = \frac{f''(\alpha)}{f'(\alpha)} \neq 0
\]

Therefore, \( a_2 \neq 0 \) and the second order convergence of the Newton’s method are verified.
Summary

- A scalar associated with a given linear transformation of a vector space and having the property that there is some nonzero vector which when multiplied by the scalar is equal to the vector obtained by letting the transformation operate on the vector; especially: a root of the characteristic equation of a matrix.
- Eigenvalues are a special set of scalars associated with a linear system of equations (i.e., a matrix equation) that are sometimes also known as characteristic roots, characteristic values (Hoffman and Kunze 1971), proper values, or latent roots (Marcus and Minc 1988, p. 144).
- The determination of the eigenvalues and eigenvectors of a system is extremely important in physics and engineering, where it is equivalent to matrix diagonalization and arises in such common applications as stability analysis, the physics of rotating bodies, and small oscillations of vibrating systems, to name only a few.
- If and only if (i.e., necessary and sufficient). The terms “just if” or “exactly when” are sometimes used instead. A \( \iff \) B is written symbolically as \( A \implies B, A \iff B, A \equiv B \), or \( A \equiv B \). A \( \iff \) B is also equivalent to \( A \implies B \) together with \( B \implies A \), where the symbol “\( \implies \)” denotes “implies.” Other equivalent terms are “A is equivalent to B” (\( A \equiv B \)) and “A XNOR B”
- If all \( k \) eigenvalues are different, then plugging these back in gives \( k-1 \) independent equations for the \( k \) components of each corresponding eigenvector, and the system is said to be nondegenerate. If the eigenvalues are \( n \)-fold degenerate, then the system is said to be degenerate and the eigenvectors are not linearly independent.
- A problem of great importance in science and engineering is that of determining the roots/zeros of an equation of the form \( f(x) = 0 \)
- A number \( \alpha \), for which \( f(\alpha) = 0 \) is called a root of the equation \( f(x) = 0 \), or a zero of \( f(x) \).
- Geometrically, a root of an equation \( f(x) = 0 \) is the value of \( x \) at which the graph of the equation \( y =f(x) \) intersects the \( x \)-axis.
- A number \( \alpha \) is a simple root of \( f(x) = 0 \), if \( f(\alpha) = 0 \) and \( f'(\alpha) \neq 0 \). Then, we can write \( f(x) \) as \( f(x) = (x - \alpha) g(x) \), \( g(\alpha) \neq 0 \)
- A polynomial equation of degree “\( n \)” has exactly \( n \) roots, real or complex, simple or multiple, where as a transcendental equation may have one root, infinite number of roots or no root.
- These methods are based on the idea of successive approximations.
- We start with one or two initial approximations to the root and obtain a sequence of approximations \( x_0, x_1, ..., x_k, ... \), which in the limit as \( k \to \infty \), converge to the exact root \( \alpha \).
- Given one or two initial approximations to the root, we require a suitable iteration function \( \varphi \) for a given function \( f(x) \), such that the sequence of iterates, \( \{x_k\} \), converge to the exact root \( \alpha \). Further, we also require a suitable criterion to terminate the iteration.

References


Recommended Reading

Self Assessment

1. Eigenvalues are a special ________ of scalars associated with a linear system of equations.
   a. system
   b. set
   c. graph
   d. values

2. The determination of the eigenvalues and eigenvectors of a system is extremely important in ____________.
   a. physics and engineering
   b. chemical and business
   c. phase and industrial
   d. potential and manufacturing

3. Each Eigenvalues is paired with a corresponding so-called __________
   a. eigenvelocity
   b. eigenvalue
   c. eigenvector
   d. eigenviscosity

4. The Lanczos algorithm is an algorithm for ____________ the eigenvalues and eigenvectors for large symmetric sparse matrices.
   a. computing
   b. totalling
   c. figuring
   d. calculating

5. If the eigenvalues are n-fold degenerate, then the system is said to be ________
   a. deteriorate
   b. be abridged to
   c. worsen
   d. degenerate

6. An equation which contains polynomials, exponential functions, logarithmic functions, trigonometric functions etc. is called a __________ equation.
   a. transcendental
   b. constant
   c. quadratic
   d. Polynomial

7. A __________ equation of degree “n” has exactly n roots, real or complex, simple or multiple.
   a. transcendental
   b. constant
   c. quadratic
   d. Polynomial
8. The ___________ method is also called chord method or regula-falsi method.
   a. straight interpolation
   b. regular interpolation
   c. false interpolation
   d. linear interpolation

9. __________ of the Newton’s method depends on the initial approximation to the root.
   a. Union
   b. Junction
   c. Convergence
   d. Meeting

10. The __________ method is also called the method of successive approximations.
    a. iteration
    b. recursive
    c. changing
    d. phasing
Chapter III
Linear System of Algebraic Equations

Aim
The aim of this chapter is to:

• discuss linear algebraic equation
• evaluate the method of linear algebraic system
• introduce direct method
• draw an elementary row transformations method

Objectives
The objectives of this chapter are to:

• state the gauss elimination method
• describe first stage elimination
• discuss remarks on each topic
• explain pivoting procedure

Learning outcome
At the end of this chapter, the students will be able to:

• solve equation and examples to understand concept
• elaborate on Gauss-Jordan method
• explain the inverse of a matrix by Gauss-Jordan method
• talk about the iterative methods
3.1 Introduction to Linear Algebraic Equation

- Consider a system of $n$ linear algebraic equations in $n$ unknowns
  \[ a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\
  a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\
  \vdots \quad \vdots \quad \vdots \quad \vdots \\
  a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \]

- Where $a_{ij}, i = 1, 2, \ldots, n, j = 1, 2, \ldots, n$, are the known coefficients, $b_i, i = 1, 2, \ldots, n$, are the known right hand side values and $x_i, i = 1, 2, \ldots, n$ are the unknowns to be determined. In matrix notation we write the system as
  \[ Ax = b \]  \( (1) \)

- The matrix $[A \mid b]$, obtained by appending the column $b$ to the matrix $A$ is called the augmented matrix. That is,
  \[ [A \mid b] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\
                      a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\
                      \vdots & \vdots & \cdots & \vdots & \vdots \\
                      a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{bmatrix} \]

3.2 Method Linear Algebraic System

We assume that the given system is consistent. The methods of solution of the linear algebraic system of equations (1) may be classified as direct and iterative methods.

Direct method

- Direct methods produce the exact solution after a finite number of steps (disregarding the round-off errors).
- In these methods, we can determine the total number of operations (additions, subtractions, divisions and multiplications).
- This number is called the operational count of the method.

Iterative method

- Iterative methods are based on the idea of successive approximations.
- We start with an initial approximation to the solution vector $x = x_0$, and obtain a sequence of approximate vectors $x_0, x_1, \ldots, x_k, \ldots$, which in the limit as $k \to \infty$, converge to the exact solution vector $x$.
- Now, we derive some direct methods.

Direct methods

- If the system of equations has some special forms, then the solution is obtained directly.
- We consider two such special forms.
  - Let $A$ be a diagonal matrix, $A = D$. That is, we consider the system of equations
\[ \begin{align*}
 Dx &= b \\
 a_{11}x_1 &= b_1 \\
 a_{22}x_2 &= b_2 \\
 \vdots & \quad \vdots & \quad \vdots \\
 a_{n-1,n-1}x_{n-1} &= b_{n-1} \\
 a_{nn}x_n &= b_n
\end{align*} \] (2)

This system is called a diagonal system of equations. Solving directly, we obtain
\[ x_i = \frac{b_i}{a_{ii}}, \quad a_{ii} \neq 0, \quad i = 1, 2, \ldots, n. \] (3)

Let \( A \) be an upper triangular matrix, \( A = U \). That is, we consider the system of equations \( Ux = b \) as
\[ \begin{align*}
 a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n &= b_i \\
 a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\
 \vdots & \quad \vdots & \quad \vdots \\
 a_{n-1,1}x_1 + a_{n-1,2}x_2 + \ldots + a_{n-1,n}x_n &= b_{n-1} \\
 a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n
\end{align*} \] (4)

This system is called an upper triangular system of equations. Solving for the unknowns in the order \( x_n, x_{n-1}, \ldots, x_1 \), we get
\[ \begin{align*}
 x_n &= \frac{b_n}{a_{nn}} \\
 x_{n-1} &= \left( b_{n-1} - a_{n-1,n}x_n \right) / a_{n-1,n-1} \\
 \vdots & \quad \vdots & \quad \vdots \\
 x_1 &= \left( b_1 - \sum_{j=2}^{n} a_{1,j}x_j \right) / a_{11}
\end{align*} \] (5)

• The unknowns are obtained by back substitution and this procedure is called the back substitution method.
• Therefore, when the given system of equations is one of the above two forms, the solution is obtained directly.
• Before we derive some direct methods, we define elementary row operations that can be performed on the rows of a matrix.

### 3.3 Elementary Row Transformations (Operations)

The following operations on the rows of a matrix \( A \) are called the elementary row transformations (operations).
• Interchange of any two rows. If we interchange the \( i \)th row with the \( j \)th row, then we usually denote the operation as \( R_i \leftrightarrow R_j \).
• Division/multiplication of any row by a non-zero number \( p \). If the \( i \)th row is multiplied by \( p \), then we usually denote this operation as \( pR_i \).
• Adding/subtracting a scalar multiple of any row to any other row.
• If all the elements of the \( j \)th row are multiplied by a scalar \( p \) and added to the corresponding elements of the \( i \)th row, then, we usually denote this operation as \( R_i \leftarrow R_i + pR_j \).
• Note the order in which the operation \( R_i \leftarrow R_i + pR_j \) is written.
• The elements of the \( j \)th row remain unchanged and the elements of the \( i \)th row get changed.
• These row operations change the form of “\( A \)”, but do not change the row-rank of “\( A \)”.
• The matrix “\( B \)” obtained after the elementary row operations is said to be row equivalent with “\( A \)”.

44
In the context of the solution of the system of algebraic equations, the solution of the new system is identical with the solution of the original system.

The above elementary operations performed on the columns of “A” (column C in place of row R) are called elementary column transformations (operations).

However, we shall be using only the elementary row operations.

In this section, we derive two direct methods for the solution of the given system of equations, namely, Gauss elimination method and Gauss-Jordan method.

3.4 Gauss Elimination Method

The method is based on the idea of reducing the given system of equations \( Ax = b \), to an upper triangular system of equations \( Ux = z \), using elementary row operations.

We know that these two systems are equivalent.

That is, the solutions of both the systems are identical.

This reduced system \( Ux = z \), is then solved by the back substitution method to obtain the solution vector \( x \).

We illustrate the method using the \( 3 \times 3 \) system.

\[
\begin{align*}
\begin{bmatrix}
    a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\
    a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\
    a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3
\end{bmatrix}
\end{align*}
\]  

We write the augmented matrix \( [A \mid b] \) and reduce it to the following form

\[
\begin{align*}
\begin{bmatrix}
    a_{11} & a_{12} & a_{13} \\
    a_{21} & a_{22} & a_{23} \\
    a_{31} & a_{32} & a_{33}
\end{bmatrix}
\end{align*} \xrightarrow{\text{Gauss elimination}}
\begin{align*}
\begin{bmatrix}
    b_1 \\
    b_2 \\
    b_3
\end{bmatrix}
\end{align*}
\]

The augmented matrix of the system (6) is

\[
\begin{bmatrix}
    a_{11} & a_{12} & a_{13} \\
    a_{21} & a_{22} & a_{23} \\
    a_{31} & a_{32} & a_{33}
\end{bmatrix}
\]

First stage of elimination

We assume \( a_{11} \neq 0 \). This element \( a_{i1} \) in the \( 1 \times 1 \) position is called the first pivot.

We use this pivot to reduce all the elements below this pivot in the first column as zeros.

Multiply the first row in (7) by \( a_{21}/a_{11} \) and \( a_{31}/a_{11} \), respectively and subtract from the second and third rows.

That is, we are performing the elementary row operations \( R_2 \leftarrow (a_{21}/a_{11})R_1 \) and \( R_3 \leftarrow (a_{31}/a_{11})R_1 \), respectively.

We obtain the new augmented matrix as

\[
\begin{bmatrix}
    a_{11} & a_{12} & a_{13} \\
    0 & a_{22}^{(1)} & a_{23}^{(1)} \\
    0 & a_{32}^{(1)} & a_{33}^{(1)}
\end{bmatrix}
\]

Where,

\[
\begin{align*}
    b_1^{(1)} &= b_1 \\
    b_2^{(1)} &= b_2 \\
    b_3^{(1)} &= b_3
\end{align*}
\]
Second stage of elimination

- We assume $a_{22}^{(1)} \neq 0$.
- This element $a_{22}^{(1)}$ in the $2 \times 2$ position is called the second pivot.
- We use this pivot to reduce the element below this pivot in the second column as zero.
- Multiply the second row in (8) by $a_{22}^{(1)}/a_{22}^{(1)}$ and subtract from the third row.
- That is, we are performing the elementary row operation $R_3 - (a_{32}^{(1)}/a_{22}^{(1)})R_2$.
- We obtain the new augmented matrix as

$$
\begin{bmatrix}
    a_{11} & a_{12} & a_{13} & b_1 \\
    0 & a_{22}^{(1)} & a_{23}^{(1)} & b_2^{(1)} \\
    0 & 0 & a_{33}^{(2)} & b_3^{(2)}
\end{bmatrix}
$$

Where,

$$a_{33}^{(2)} = a_{33}^{(1)} - \left(\frac{a_{32}^{(1)}}{a_{22}^{(1)}}\right) a_{23}^{(1)}, \quad b_3^{(2)} = b_3^{(1)} - \left(\frac{a_{32}^{(1)}}{a_{22}^{(1)}}\right) b_2^{(1)}$$

- The element $a_{33}^{(2)} \neq 0$ is called the third pivot.
- This system is in the required upper triangular form $[U|z]$.
- The solution vector $x$ is now obtained by back substitution.

From the third row, we get $x_3 = b_3^{(2)}/a_{33}^{(2)}$.

From the second row, we get $x_2 = (b_2^{(1)} - a_{23}^{(1)} x_3)/a_{22}^{(1)}$.

From the first row, we get $x_1 = (b_1 - a_{12} x_2 - a_{13} x_3)/a_{11}$.

- In general, using a pivot, all the elements below that pivot in that column are made zeros.
- Alternately, at each stage of elimination, we may also make the pivot as 1, by dividing that particular row by the pivot.

Remark 15

- When the above mentioned Gauss elimination method fail?
- It fails when any one of the pivots is zero or it is a very small number, as the elimination progresses.
- If a pivot is zero, then division by it gives over flow error, since division by zero is not defined.
- If a pivot is a very small number, then division by it introduces large round-off errors and the solution may contain large errors.
- For example, we may have the system.

$$
2x_2 + 5x_3 = 7 \\
7x_1 + x_2 - 2x_3 = 6 \\
2x_1 + 3x_2 + 8x_3 = 13
$$
in which the first pivot is zero.
3.5 Pivoting Procedures

- How do we avoid computational errors in Gauss elimination?
- To avoid computational errors, we follow the procedure of partial pivoting.
- In the first stage of elimination, the first column of the augmented matrix is searched for the largest element in magnitude and brought as the first pivot by interchanging the first row of the augmented matrix (first equation) with the row (equation) having the largest element in magnitude.
- In the second stage of elimination, the second column is searched for the largest element in magnitude among the n – 1 elements leaving the first element, and this element is brought as the second pivot by interchanging the second row of the augmented matrix with the later row having the largest element in magnitude.
- This procedure is continued until the upper triangular system is obtained.
- Therefore, partial pivoting is done after every stage of elimination.
- There is another procedure called complete pivoting.
- In this procedure, we search the entire matrix A in the augmented matrix for the largest element in magnitude and bring it as the first pivot.
- This requires not only an interchange of the rows, but also an interchange of the positions of the variables.
- It is possible that the position of a variable is changed a number of times during this pivoting.
- We need to keep track of the positions of all the variables. Hence, the procedure is computationally expensive and is not used in any software.

Remark 16

- Gauss elimination method is a direct method.
- Therefore, it is possible to count the total number of operations, that is, additions, subtractions, divisions and multiplications.
- Without going into details, we mention that the total number of divisions and multiplications (division and multiplication take the same amount of computer time) is \( n (n^2 + 3n – 1)/3 \).
- The total number of additions and subtractions (addition and subtraction take the same amount of computer time) is \( n (n – 1)(2n + 5)/6 \).

Remark 17

- When the system of algebraic equations is large, how do we conclude that it is consistent or not, using the Gauss elimination method?
- A way of determining the consistency is from the form of the reduced system (9).
- We know that if the system is inconsistent, then rank (A) ≠ rank [A|b].
- By checking the elements of the last rows, conclusion can be drawn about the consistency or inconsistency.
- Suppose that in (9), \( a_{33}^{(2)} ≠ 0 \) and \( a_{33}^{(2)} a_{33}^{(2)} ≠ 0 \).
- Then, rank (A) = rank [A|b] = 3.
- The system is consistent and has a unique solution.
- Suppose that we obtain the reduced system as

\[
\begin{bmatrix}
 a_{11} & a_{12} & a_{13} & b_1 \\
 0 & a_{22}^{(1)} & a_{23}^{(1)} & b_2^{(1)} \\
 0 & 0 & a_{33}^{(2)} & b_3^{(2)} \\
\end{bmatrix}
\]

- Then, rank (A) = 2, rank [A|b] = 3 and rank (A) ≠ rank [A|b]. Therefore, the system is inconsistent and has no solution.
- Suppose that we obtain the reduced system as
Then, \( \text{rank} \ (A) = \text{rank} \ [A|b] = 2 < 3 \). Therefore, the system has \( 3 - 2 = 1 \) parameter family of infinite number of solutions.

**Example**

Solve the system of equations.

\[
\begin{align*}
x_1 + 10x_2 - x_3 &= 3 \\
2x_1 + 3x_2 + 20x_3 &= 7 \\
10x_1 - x_2 + 2x_3 &= 4
\end{align*}
\]

Using the Gauss elimination with partial pivoting.

\[
\begin{bmatrix}
1 & 10 & -1 & 3 \\
2 & 3 & 20 & 7 \\
10 & -1 & 2 & 4
\end{bmatrix}
\]

We perform the following elementary row transformations and do the eliminations.

**Solution**

We perform the following elementary row transformations and do the eliminations.

\[
\begin{align*}
R_1 &\leftrightarrow R_3 ; 
\begin{bmatrix}
10 & -1 & 2 & 4 \\
2 & 3 & 20 & 7 \\
10 & -1 & 2 & 4
\end{bmatrix}
R_2 - (R_1/5), R_3 - (R_1/10) : \\
\begin{bmatrix}
10 & -1 & 2 & 4 \\
0 & 3.2 & 19.6 & 6.2 \\
0 & 101 & -12 & 26
\end{bmatrix} \\
R_2 &\leftrightarrow R_5 ; 
\begin{bmatrix}
10 & -1 & 2 & 4 \\
0 & 3.2 & 19.6 & 6.2 \\
10 & -1 & 2 & 4
\end{bmatrix}
\]

Back substitution gives the solution.

**Example**

Solve the system of equations

\[
\begin{align*}
2x_1 + x_2 + x_3 - 2x_4 &= -10 \\
4x_1 + 2x_2 + x_3 &= 8 \\
3x_1 + 2x_2 + 2x_3 &= 7 \\
x_1 + 3x_2 + 2x_3 - x_4 &= -5
\end{align*}
\]

Using the Gauss elimination with partial pivoting.

**Solution**

The augmented matrix is given by
We perform the following elementary row transformations and do the eliminations.

\[
\begin{bmatrix}
2 & 1 & 1 & -2 & -10 \\
4 & 0 & 2 & 1 & 8 \\
3 & 2 & 2 & 0 & 7 \\
1 & 3 & 2 & -1 & -5
\end{bmatrix}
\]

Example
Solve the system of equations

Using the Gauss elimination method.

Solution
Let us solve this problem by making the pivots as 1. The augmented matrix is given by

\[
\begin{bmatrix}
3 & 3 & 4 & 20 \\
2 & 1 & 3 & 13 \\
1 & 1 & 3 & 6
\end{bmatrix}
\]

Using back substitution, we obtain

\[
x_4 = \begin{pmatrix} -\frac{52}{3} \\ -\frac{6}{13} \end{pmatrix} = 8, x_3 = -2 \begin{pmatrix} \frac{17}{3} - \frac{1}{12} x_3 \\ \frac{17}{3} - \frac{1}{12} (8) \end{pmatrix} = -2 \begin{pmatrix} \frac{17}{3} - \frac{1}{12} (8) \end{pmatrix} = -10, \\
x_2 = \begin{pmatrix} 3 \\ -7 \end{pmatrix} x_3 + \begin{pmatrix} 5 \\ 4 \end{pmatrix} x_4 = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \end{pmatrix} \left[ -7 \begin{pmatrix} 3 \\ 2 \end{pmatrix} x_3 + \begin{pmatrix} 5 \\ 4 \end{pmatrix} x_4 \right] = \begin{pmatrix} \frac{1}{3} \end{pmatrix} \left[ -7 \begin{pmatrix} 3 \\ 2 \end{pmatrix} (-10) + \begin{pmatrix} 5 \\ 4 \end{pmatrix} (8) \right] = 6,
\]

\[
x_1 = \begin{pmatrix} 1 \\ 4 \end{pmatrix} [8 - 2x_3 - x_4] = \begin{pmatrix} 1 \\ 4 \end{pmatrix} [8 - 2(-10) - 8] = 5.
\]
Back substitution gives the solution as
\[ x_3 = -\frac{20}{3}, \quad x_2 = \frac{20}{3} - 4 \left( -\frac{2}{5} \right) = \frac{35}{5} = 7. \]

Example
Test the consistency of the following system of equations.
\[
\begin{align*}
  x_1 + 10x_2 - x_3 &= 3 \\
  2x_1 + 3x_2 + 20x_3 &= 7 \\
  9x_1 + 22x_2 + 79x_3 &= 45
\end{align*}
\]

Using the Gauss elimination method.

Solution
We have the augmented matrix as
\[
\begin{bmatrix}
  1 & 10 & -1 & 3 \\
  2 & 3 & 20 & 7 \\
  9 & 22 & 79 & 45
\end{bmatrix}
\]

We perform the following elementary row transformations and do the eliminations.
\[
R_2 - 2R_1, \quad R_3 - 9R_1:
\begin{bmatrix}
  1 & 10 & -1 & 3 \\
  0 & -17 & 22 & 1 \\
  0 & -68 & 88 & 18
\end{bmatrix},
R_3 - 4R_2:
\begin{bmatrix}
  1 & 10 & -1 & 3 \\
  0 & -17 & 22 & 1 \\
  0 & 0 & 0 & 14
\end{bmatrix}
\]

Now, \( \text{rank } [A] = 2 \), and \( \text{rank } [A|b] = 3 \). Therefore, the system is inconsistent and has no solution.

3.6 Gauss-Jordan Method

- The method is based on the idea of reducing the given system of equations \( Ax = b \), to a diagonal system of equations \( Ix = d \), where \( I \) is the identity matrix, using elementary row operations.
- We know that the solutions of both the systems are identical.
- This reduced system gives the solution vector \( x \).
- This reduction is equivalent to finding the solution as \( x = A^{-1}b \).

\[
[A|b] \xrightarrow{\text{Gauss-Jordan method}} [I | X]
\]

- In this case, after the eliminations are completed, we obtain the augmented matrix for a 3 × 3 system as
\[
\begin{bmatrix}
  1 & 0 & 0 & d_1 \\
  0 & 1 & 0 & d_2 \\
  0 & 0 & 1 & d_3
\end{bmatrix}
\]

and the solution is \( x_i = d_i, \ i = 1, 2, 3 \).

Elimination procedure
- The first step is same as in Gauss elimination method, that is, we make the elements below the first pivot as zeros, using the elementary row transformations.
• From the second step onwards, we make the elements below and above the pivots as zeros using the elementary row transformations.

• Lastly, we divide each row by its pivot so that the final augmented matrix is of the form (10). Partial pivoting can also be used in the solution.

• We may also make the pivots as 1 before performing the elimination.

• Let us illustrate the method.

Example
Solve the following system of equations
\[ x_1 + x_2 + x_3 = 1 \]
\[ 4x_1 + 3x_2 - x_3 = 6 \]
\[ 3x_1 + 5x_2 + 3x_3 = 4 \]
Using the Gauss-Jordan method (i) without partial pivoting, (ii) with partial pivoting.

Solution
We have the augmented matrix as
\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
4 & 3 & -1 & 6 \\
3 & 5 & 3 & 4
\end{bmatrix}
\]

• We perform the following elementary row transformations and do the eliminations.

\[
R_2 - 4R_1, R_3 - 3R_1 : \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & -5 & 2 \\ 0 & 2 & 0 & 1 \end{bmatrix}
\]

\[
R_1 + R_2, R_3 + 2R_2 : \begin{bmatrix} 1 & 0 & -4 & 3 \\ 0 & -1 & -5 & 2 \\ 0 & 0 & -10 & 5 \end{bmatrix}
\]

\[
R_1 - (4/10)R_2, R_3 - (5/10)R_2 : \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & 0 & -1/2 \\ 0 & 0 & -10 & 5 \end{bmatrix}
\]

Now, making the pivots as 1, ((– R_2), (R_3/(– 10))) we get

\[
\begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1/2 \\
0 & 0 & 1 & -1/2
\end{bmatrix}
\]

Therefore, the solution of the system is \( x_1 = 1, x_2 = 1/2, x_3 = -1/2 \).

• We perform the following elementary row transformations and do the eliminations.

\[
R_2 - 4R_3, R_3 - 3R_1 : \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & -5 & 2 \\ 0 & 2 & 0 & 1 \end{bmatrix}
\]

\[
R_1 + R_2, R_3 + 2R_2 : \begin{bmatrix} 1 & 0 & -4 & 3 \\ 0 & -1 & -5 & 2 \\ 0 & 0 & -10 & 5 \end{bmatrix}
\]

\[
R_1 - (4/10)R_2, R_3 - (5/10)R_2 : \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & 0 & -1/2 \\ 0 & 0 & -10 & 5 \end{bmatrix}
\]

Now, making the pivots as 1, ((– R_2), (R_3/(– 10))) we get
Therefore, the solution of the system is \( x_1 = 1, x_2 = 1/2, x_3 = -1/2 \).

We perform the following elementary row transformations and do the elimination.

\[
R_1 \leftrightarrow R_2 : \begin{bmatrix} 4 & 3 & -1 & | & 6 \\ 1 & 1 & 1 & | & 1 \\ 3 & 5 & 3 & | & 4 \end{bmatrix}, \quad R_1/4 : \begin{bmatrix} 1 & 3/4 & -1/4 & | & 3/2 \\ 1 & 1 & 1 & | & 1 \\ 3 & 5 & 3 & | & 4 \end{bmatrix}
\]

\[
R_2 - R_1, R_3 - 3R_1 : \begin{bmatrix} 1 & 3/4 & -1/4 & | & 3/2 \\ 0 & 1/4 & 5/4 & | & -1/2 \\ 0 & 1/4 & 15/4 & | & -1/2 \end{bmatrix}
\]

\[
R_2 \leftrightarrow R_3 : \begin{bmatrix} 1 & 3/4 & -1/4 & | & 3/2 \\ 0 & 1/4 & 15/4 & | & -1/2 \\ 0 & 1/4 & 5/4 & | & -1/2 \end{bmatrix}, \quad R_3/(1/4) : \begin{bmatrix} 1 & 3/4 & -1/4 & | & 3/2 \\ 0 & 1 & 15/11 & | & -2/11 \\ 0 & 1/4 & 5/4 & | & -1/2 \end{bmatrix}
\]

\[
R_1 - (3/4) R_2, R_3 - (1/4)R_2 : \begin{bmatrix} 1 & 0 & -14/11 & | & 18/11 \\ 0 & 1 & 15/11 & | & -2/11 \\ 0 & 0 & 10/11 & | & -5/11 \end{bmatrix}
\]

\[
R_3/(10/11) : \begin{bmatrix} 1 & 0 & -14/11 & | & 18/11 \\ 0 & 1 & 15/11 & | & -2/11 \\ 0 & 0 & 1 & | & -1/2 \end{bmatrix}
\]

\[
R_1 + (14/11) R_3, R_2 - (15/11)R_3 : \begin{bmatrix} 1 & 0 & 0 & | & 1 \\ 0 & 1 & 0 & | & 1/2 \\ 0 & 0 & 1 & | & -1/2 \end{bmatrix}
\]

Therefore, the solution of the system is \( x_1 = 1, x_2 = 1/2, x_3 = -1/2 \).

**Remark 18**

- The Gauss-Jordan method looks very elegant, as the solution is obtained directly.
- However, it is computationally more expensive than Gauss elimination.
- For large \( n \), the total number of divisions and multiplications for Gauss-Jordan method is almost 1.5 times the total number of divisions and multiplications required for Gauss elimination.
- Hence, we do not normally use this method for the solution of the system of equations.
- The most important application of this method is to find the inverse of a non-singular matrix.
- We present this method in the following section.

### 3.7 Inverse of a Matrix by Gauss-Jordan Method

- As given in Remark 18, the important application of the Gauss-Jordan method is to find the inverse of a non-singular matrix \( A \).
- We start with the augmented matrix of \( A \) with the identity matrix \( I \) of the same order.
- When the Gauss-Jordan procedure is completed, we obtain

\[
\text{Gauss-Jordan method} \quad \begin{bmatrix} A & | & I \end{bmatrix} \rightarrow \begin{bmatrix} I & | & A^{-1} \end{bmatrix}
\]

Since, \( AA^{-1} = I \).
Remark 19

- Partial pivoting can also be done using the augmented matrix \([A|I]\).
- However, we cannot first interchange the rows of \(A\) and then find the inverse.
- Then, we would be finding the inverse of a different matrix.

Example

Find the inverse of the matrix

\[
\begin{bmatrix}
1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3
\end{bmatrix}
\]

Using the Gauss-Jordan method (i) without partial pivoting, and (ii) with partial pivoting.

Solution

Consider the augmented matrix

\[
\begin{bmatrix}
1 & 1 & 1 & | & 1 & 0 & 0 \\ 4 & 3 & -1 & | & 0 & 1 & 0 \\ 3 & 5 & 3 & | & 0 & 0 & 1
\end{bmatrix}
\]

\[
R_2 - 4R_1, \quad R_3 - 3R_1: \quad \begin{bmatrix}
1 & 1 & 1 & | & 1 & 0 & 0 \\ 0 & -1 & -5 & | & -4 & 1 & 0 \\ 0 & 2 & 0 & | & -3 & 0 & 1
\end{bmatrix}
\]

\[-R_2: \quad \begin{bmatrix}
1 & 1 & 1 & | & 1 & 0 & 0 \\ 0 & 1 & 5 & | & 4 & -1 & 0 \\ 0 & 2 & 0 & | & -3 & 0 & 1
\end{bmatrix}
\]

\[R_1 - R_2, \quad R_3 - 2R_2: \quad \begin{bmatrix}
1 & 0 & -4 & | & -3 & 1 & 0 \\ 0 & 1 & 5 & | & 4 & -1 & 0 \\ 0 & 0 & -10 & | & -11 & 2 & 1
\end{bmatrix}
\]

\[R_3/(−10): \quad \begin{bmatrix}
1 & 0 & -4 & | & -3 & 1 & 0 \\ 0 & 1 & 5 & | & 4 & -1 & 0 \\ 0 & 0 & 1 & | & 11/10 & -2/10 & -1/10
\end{bmatrix}
\]

\[R_1 + 4R_3, \quad R_3 - 5R_3: \quad \begin{bmatrix}
1 & 0 & 0 & | & 14/10 & 2/10 & -4/10 \\ 0 & 1 & 0 & | & -15/10 & 0 & 5/10 \\ 0 & 0 & 1 & | & 11/10 & -2/10 & -1/10
\end{bmatrix}
\]

- We perform the following elementary row transformations and do the eliminations.

Therefore, the inverse of the given matrix is given by

\[
\begin{bmatrix}
7/5 & 1/5 & -2/5 \\ -3/2 & 0 & 1/2 \\ 11/10 & -1/5 & -1/10
\end{bmatrix}
\]

- We perform the following elementary row transformations and do the eliminations.
Therefore, the inverse of the matrix is given by

\[
\begin{bmatrix}
7/5 & 1/5 & -2/5 \\
-3/2 & 0 & 1/2 \\
11/10 & -1/5 & -1/10
\end{bmatrix}
\]

**Example**

Using the Gauss-Jordan method, find the inverse of

\[
\begin{bmatrix}
2 & 2 & 3 \\
2 & 1 & 1 \\
1 & 3 & 5
\end{bmatrix}
\]

**Solution**

We have the following augmented matrix.

\[
\begin{bmatrix}
2 & 2 & 3 & 1 & 0 & 0 \\
2 & 1 & 1 & 0 & 1 & 0 \\
1 & 3 & 5 & 0 & 0 & 1
\end{bmatrix}
\]

We perform the following elementary row transformations and do the eliminations.
Therefore, the inverse of the given matrix is given by

\[ \begin{bmatrix} 2 & -1 & -1 \\ -9 & 7 & 4 \\ 5 & -4 & -2 \end{bmatrix} \]

### 3.8 Iterative Methods

- As discussed earlier, iterative methods are based on the idea of successive approximations.
- We start with an initial approximation to the solution vector \( x = x_0 \), to solve the system of equations \( Ax = b \), and obtain a sequence of approximate vectors \( x_0, x_1, ..., x_k, ... \), which in the limit as \( k \to \infty \), converges to the exact solution vector \( x = A^{-1}b \).

A general linear iterative method for the solution of the system of equations \( Ax = b \), can be written in matrix form as

\[ x^{(k+1)} = Hx^{(k)} + c, \quad k = 0, 1, 2, ... \]  

Where \( x^{(k+1)} \) and \( x^{(k)} \) are the approximations for \( x \) at the \( (k+1) \)th and \( k \)th iterations respectively.

\( H \) is called the iteration matrix, which depends on \( A \) and \( c \) is a column vector, which depends on \( A \) and \( b \).

- When to stop the iteration? We stop the iteration procedure when the magnitudes of the differences between the two successive iterates of all the variables are smaller than a given accuracy or error tolerance or an error bound \( \varepsilon \), that is,

\[ |x_i^{(k+1)} - x_i^{(k)}| \leq \varepsilon, \quad \text{for all } i. \]  

For example, if we require two decimal places of accuracy, then we iterate until

\[ |x_i^{(k+1)} - x_i^{(k)}| < 0.005, \]

...... for all \( i \),

- If we require three decimal places of accuracy, then we iterate until

\[ |x_i^{(k+1)} - x_i^{(k)}| < 0.0005, \]

.........for all \( i \),

Convergence property of an iterative method depends on the iteration matrix \( H \).
Now, we derive two iterative methods for the solution of the system of algebraic equations.

\[
\begin{align*}
    a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\
    a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\
    a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3
\end{align*}
\] (12)

### 3.9 Gauss-Jacobi Iteration Method

- Sometimes, the method is called Jacobi method. We assume that the pivots \( a_{ii} \neq 0 \), for all \( i \). Write the equations as
  
  \[
  \begin{align*}
  x_1^{(k+1)} &= \frac{1}{a_{11}} \left[ b_1 - (a_{12}x_2^{(k)} + a_{13}x_3^{(k)}) \right] \\
  x_2^{(k+1)} &= \frac{1}{a_{22}} \left[ b_2 - (a_{21}x_1^{(k)} + a_{23}x_3^{(k)}) \right] \\
  x_3^{(k+1)} &= \frac{1}{a_{33}} \left[ b_3 - (a_{31}x_1^{(k)} + a_{32}x_2^{(k)}) \right], \quad k = 0, 1, 2, \ldots
  \end{align*}
  \] (13)

- Since we replace the complete vector \( x^{(k)} \) in the right hand side of (13) at the end of each iteration, this method is also called the method of simultaneous displacement.

**Remark 20**

- A sufficient condition for convergence of the Jacobi method is that the system of equations is diagonally dominant, that is, the coefficient matrix \( A \) is diagonally dominant. We can verify that
  \[
  \left| \alpha_{ii} \right| \approx \sum_{j=1, i \neq j}^n \left| \alpha_{ij} \right|
  \]
  This implies that convergence may be obtained even if the system is not diagonally dominant.
  - If the system is not diagonally dominant, we may exchange the equations, if possible, such that the new system is diagonally dominant and convergence is guaranteed.
  - However, such manual verification or exchange of equations may not be possible for large systems that we obtain in application problems.
  - The necessary and sufficient condition for convergence is that the spectral radius of the iteration matrix \( H \) is less than one unit, that is, \( \rho(H) < 1 \), where \( \rho(H) \) is the largestEigen value in magnitude of \( H \).
  - Testing of this condition is beyond the scope of the syllabus.

**Remark 21**

How do we find the initial approximations to start the iteration?

- If the system is diagonally dominant, then the iteration converges for any initial solution vector.
- If no suitable approximation is available, we can choose \( x = 0 \), that is \( x_i = 0 \) for all \( i \).
- Then, the initial approximation becomes \( x_i = b_i / a_{ii} \), for all \( i \).

**Example**

Solve the system of equations
\[ 4x_1 + x_2 + x_3 = 2 \]
\[ x_1 + 5x_2 + 2x_3 = -6 \]
\[ x_1 + 2x_2 + 3x_3 = -4 \]

Using the Jacobi iteration method. Use the initial approximations as

\( (i) x_i = 0, i = 1, 2, 3, \quad (ii) x_i = 0.5, x_2 = -0.5, x_3 = 0.5. \)

Perform five iterations in each case.

**Solution**

Note that the given system is diagonally dominant. Jacobi method gives the iterations as

\[ x_1^{(k+1)} = 0.25 \left[ 2 - (x_2^{(k)} + x_3^{(k)}) \right] \]
\[ x_2^{(k+1)} = 0.2 \left[ -6 - (x_1^{(k)} + 2x_3^{(k)}) \right] \]
\[ x_3^{(k+1)} = 0.33333 \left[ 4 - (x_1^{(k)} + 2x_2^{(k)}) \right], \quad k = 0, 1, \ldots \]

We have the following results.

**First iteration**

\[ x_1^{(1)} = 0.25 \left[ 2 - (x_2^{(0)} + x_3^{(0)}) \right] = 0.5, \]
\[ x_2^{(1)} = 0.2 \left[ -6 - (x_1^{(0)} + 2x_3^{(0)}) \right] = -1.2, \]
\[ x_3^{(1)} = 0.33333 \left[ 4 - (x_1^{(0)} + 2x_2^{(0)}) \right] = -1.33333. \]

**Second iteration**

\[ x_1^{(2)} = 0.25 \left[ 2 - (x_2^{(1)} + x_3^{(1)}) \right] = 0.25 \left[ 2 - (-1.2 - 1.33333) \right] = 1.13333, \]
\[ x_2^{(2)} = 0.2 \left[ -6 - (x_1^{(1)} + 2x_3^{(1)}) \right] = 0.2 \left[ -6 - (0.5 + 2(-1.33333)) \right] = -0.76668, \]
\[ x_3^{(2)} = 0.33333 \left[ 4 - (x_1^{(1)} + 2x_2^{(1)}) \right] = 0.33333 \left[ 4 - (0.5 + 2(-1.2)) \right] = -0.7. \]

**Third iteration**

\[ x_1^{(3)} = 0.25 \left[ 2 - (x_2^{(2)} + x_3^{(2)}) \right] = 0.25 \left[ 2 - (-0.76668 - 0.7) \right] = 0.86667, \]
\[ x_2^{(3)} = 0.2 \left[ -6 - (x_1^{(2)} + 2x_3^{(2)}) \right] = 0.2 \left[ -6 - (1.13333 + 2(-0.7)) \right] = -1.14667, \]
\[ x_3^{(3)} = 0.33333 \left[ 4 - (x_1^{(2)} + 2x_2^{(2)}) \right] = 0.33333 \left[ 4 - (1.13333 + 2(-0.76668)) \right] = -1.19998. \]

**Fourth iteration**

\[ x_1^{(4)} = 0.25 \left[ 2 - (x_2^{(3)} + x_3^{(3)}) \right] = 0.25 \left[ 2 - (-1.14667 - 1.19999) \right] = 1.08666, \]
\[ x_2^{(4)} = 0.2 \left[ -6 - (x_1^{(3)} + 2x_3^{(3)}) \right] = 0.2 \left[ -6 - (0.86667 + 2(-1.19998)) \right] = -0.89334, \]
\[ x_3^{(4)} = 0.33333 \left[ 4 - (x_1^{(3)} + 2x_2^{(3)}) \right] = 0.33333 \left[ 4 - (0.86667 + 2(-1.14667)) \right] = -0.85777. \]

**Fifth iteration**

\[ x_1^{(5)} = 0.25 \left[ 2 - (x_2^{(4)} + x_3^{(4)}) \right] = 0.25 \left[ 2 - (-0.89334 - 0.85777) \right] = 0.93778, \]
\[ x_2^{(5)} = 0.2 \left[ -6 - (x_1^{(4)} + 2x_3^{(4)}) \right] = 0.2 \left[ -6 - (1.08666 + 2(-0.85777)) \right] = -1.07422, \]
\[ x_3^{(5)} = 0.33333 \left[ 4 - (x_1^{(4)} + 2x_2^{(4)}) \right] = 0.33333 \left[ 4 - (1.08666 + 2(-0.89334)) \right] = -1.09998. \]

It is interesting to note that the iterations oscillate and converge to the exact solution

\( x_1 = 1.0, x_2 = -1, x_3 = -1.0. \)

\( (ii) x_1^{(0)} = 0.5, x_2^{(0)} = -0.5, x_3^{(0)} = 0.5. \)
First iteration
\[ x_1^{(1)} = 0.25 \left[ 2 - (x_2^{(0)} + x_3^{(0)}) \right] = 0.25 \left[ 2 - (0.5 - 0.5) \right] = 0.75, \]
\[ x_2^{(1)} = 0.2 \left[ -6 - (x_1^{(0)} + 2x_3^{(0)}) \right] = 0.2 \left[ -6 - (0.5 + 2(0.5)) \right] = -1.1, \]
\[ x_3^{(1)} = 0.3333 \left[ -4 - (x_1^{(0)} + 2x_2^{(0)}) \right] = 0.3333 \left[ -4 - (0.5 + 2(0.5)) \right] = -1.16667. \]

Second iteration
\[ x_1^{(2)} = 0.25 \left[ 2 - (x_2^{(1)} + x_3^{(1)}) \right] = 0.25 \left[ 2 - (1.06667 - 1.16667) \right] = 1.06667, \]
\[ x_2^{(2)} = 0.2 \left[ -6 - (x_1^{(1)} + 2x_3^{(1)}) \right] = 0.2 \left[ -6 - (0.75 + 2(-1.16667)) \right] = -0.88333, \]
\[ x_3^{(2)} = 0.3333 \left[ -4 - (x_1^{(1)} + 2x_2^{(1)}) \right] = 0.3333 \left[ -4 - (0.75 + 2(-1.1)) \right] = -0.84999. \]

Third iteration
\[ x_1^{(3)} = 0.25 \left[ 2 - (x_2^{(2)} + x_3^{(2)}) \right] = 0.25 \left[ 2 - (-0.88333 - 0.84999) \right] = 0.93333, \]
\[ x_2^{(3)} = 0.2 \left[ -6 - (x_1^{(2)} + 2x_3^{(2)}) \right] = 0.2 \left[ -6 - (1.06667 + 2(-0.84999)) \right] = -1.07334, \]
\[ x_3^{(3)} = 0.3333 \left[ -4 - (x_1^{(2)} + 2x_2^{(2)}) \right] = 0.3333 \left[ -4 - (1.06667 + 2(-0.84999)) \right] = -1.09999. \]

Fourth iteration
\[ x_1^{(4)} = 0.25 \left[ 2 - (x_2^{(3)} + x_3^{(3)}) \right] = 0.25 \left[ 2 - (-1.07334 - 1.09999) \right] = 1.04333, \]
\[ x_2^{(4)} = 0.2 \left[ -6 - (x_1^{(3)} + 2x_3^{(3)}) \right] = 0.2 \left[ -6 - (0.93333 + 2(-1.09999)) \right] = -0.94667, \]
\[ x_3^{(4)} = 0.3333 \left[ -4 - (x_1^{(3)} + 2x_2^{(3)}) \right] = 0.3333 \left[ -4 - (0.93333 + 2(-1.09999)) \right] = -0.92887. \]

Fifth iteration
\[ x_1^{(5)} = 0.25 \left[ 2 - (x_2^{(4)} + x_3^{(4)}) \right] = 0.25 \left[ 2 - (-0.94667 - 0.92887) \right] = 0.96889, \]
\[ x_2^{(5)} = 0.2 \left[ -6 - (x_1^{(4)} + 2x_3^{(4)}) \right] = 0.2 \left[ -6 - (1.04333 + 2(-0.92887)) \right] = -1.03712, \]
\[ x_3^{(5)} = 0.3333 \left[ -4 - (x_1^{(4)} + 2x_2^{(4)}) \right] = 0.3333 \left[ -4 - (1.04333 + 2(-0.94667)) \right] = -1.04999. \]

Example
Solve the system of equations.
\[ 26x_1 + 2x_2 + 2x_3 = 12.6 \]
\[ 3x_1 + 27x_2 + x_3 = -14.3 \]
\[ 2x_1 + 3x_2 + 17x_3 = 6.0 \]

Using the Jacobi iteration method. Obtain the result correct to three decimal places.

Solution

The given system of equations is strongly diagonally dominant. Hence, we can expect faster convergence. Jacobi method gives the iterations as

\[ x_1^{(k+1)} = \frac{12.6 - (2x_2^{(k)} + 2x_3^{(k)})}{26}, \]
\[ x_2^{(k+1)} = \frac{-14.3 - (3x_1^{(k)} + x_3^{(k)})}{27}, \]
\[ x_3^{(k+1)} = \frac{6.0 - (2x_1^{(k)} + 3x_2^{(k)})}{17}, \]

Choose the initial approximation as \( x_1^{(0)} = 0, x_2^{(0)} = 0, x_3^{(0)} = 0. \) We obtain the following results.
First iteration

\[ x_1^{(1)} = \frac{1}{26} \left[ 12.6 - (2x_2^{(0)} + 2x_3^{(0)}) \right] = \frac{1}{26} [12.6] = 0.48462, \]

\[ x_2^{(1)} = \frac{1}{27} \left[ -14.3 - (3x_1^{(0)} + x_3^{(0)}) \right] = \frac{1}{27} [-14.3] = -0.52963, \]

\[ x_3^{(1)} = \frac{1}{17} \left[ 6.0 - (2x_1^{(0)} + 3x_2^{(0)}) \right] = \frac{1}{17} [6.0] = 0.35294. \]

Second iteration

\[ x_1^{(2)} = \frac{1}{26} \left[ 12.6 - (2x_2^{(1)} + 2x_3^{(1)}) \right] = \frac{1}{26} [12.6 - 2(-0.52963 + 0.35294)] = 0.49821, \]

\[ x_2^{(2)} = \frac{1}{27} \left[ -14.3 - (3x_1^{(1)} + x_3^{(1)}) \right] = \frac{1}{27} [-14.3 - (3(0.48462) + 0.35294)] = -0.59655, \]

\[ x_3^{(2)} = \frac{1}{17} \left[ 6.0 - (2x_1^{(1)} + 3x_2^{(1)}) \right] = \frac{1}{17} [6.0 - (2(0.48462) + 3(-0.52963))] = 0.38939. \]

Third iteration

\[ x_1^{(3)} = \frac{1}{26} \left[ 12.6 - (2x_2^{(2)} + 2x_3^{(2)}) \right] = \frac{1}{26} [12.6 - 2(-0.59655 + 0.38939)] = 0.50006, \]

\[ x_2^{(3)} = \frac{1}{27} \left[ -14.3 - (3x_1^{(2)} + x_3^{(2)}) \right] = \frac{1}{27} [-14.3 - (3(0.49821) + 0.38939)] = -0.59941, \]

\[ x_3^{(3)} = \frac{1}{17} \left[ 6.0 - (2x_1^{(2)} + 3x_2^{(2)}) \right] = \frac{1}{17} [6.0 - (2(0.49821) + 3(-0.59655))] = 0.39960. \]

Fourth iteration

\[ x_1^{(4)} = \frac{1}{26} \left[ 12.6 - (2x_2^{(3)} + 2x_3^{(3)}) \right] = \frac{1}{26} [12.6 - 2(-0.59941 + 0.39960)] = 0.50000, \]

\[ x_2^{(4)} = \frac{1}{27} \left[ -14.3 - (3x_1^{(3)} + x_3^{(3)}) \right] = \frac{1}{27} [-14.3 - (3(0.50006) + 0.39960)] = -0.59999, \]

\[ x_3^{(4)} = \frac{1}{17} \left[ 6.0 - (2x_1^{(3)} + 3x_2^{(3)}) \right] = \frac{1}{17} [6.0 - (2(0.50006) + 3(-0.59941))] = 0.39989. \]

\[ |x_1^{(4)} - x_1^{(3)}| = |0.5 - 0.50006| = 0.00006, \]

\[ |x_2^{(4)} - x_2^{(3)}| = | -0.59999 + 0.59941 | = 0.00058, \]

\[ |x_3^{(4)} - x_3^{(3)}| = |0.39989 - 0.39960| = 0.00029. \]

Three decimal places of accuracy have not been obtained at this iteration.

Fifth iteration

\[ x_1^{(5)} = \frac{1}{26} \left[ 12.6 - (2x_2^{(4)} + 2x_3^{(4)}) \right] = \frac{1}{26} [12.6 - 2(-0.59999 + 0.39989)] = 0.50001, \]

\[ x_2^{(5)} = \frac{1}{27} \left[ -14.3 - (3x_1^{(4)} + x_3^{(4)}) \right] = \frac{1}{27} [-14.3 - (3(0.50000) + 0.39989)] = -0.60000, \]

\[ x_3^{(5)} = \frac{1}{17} \left[ 6.0 - (2x_1^{(4)} + 3x_2^{(4)}) \right] = \frac{1}{17} [6.0 - (2(0.50000) + 3(-0.59999))] = 0.40000. \]

We find

\[ |x_1^{(4)} - x_1^{(3)}| = |0.50001 - 0.5| = 0.00001, \]

\[ |x_2^{(4)} - x_2^{(3)}| = |-0.6 + 0.59999| = 0.00001, \]

\[ |x_3^{(4)} - x_3^{(3)}| = |0.4 - 0.39989| = 0.00011. \]
Since, all the errors in magnitude are less than 0.0005, the required solution is

\[ x_1 = 0.5, \; x_2 = -0.6, \; x_3 = 0.4. \]

**Remark 22**

What is the disadvantage of the Gauss-Jacobi method?

- At any iteration step, the value of the first variable \( x_1 \) is obtained using the values of the previous iteration.
- The value of the second variable \( x_2 \) is also obtained using the values of the previous iteration, even though the updated value of \( x_1 \) is available.
- In general, at every stage in the iteration, values of the previous iteration are used even though the updated values of the previous variables are available.
- If we use the updated values of \( x_1, x_2, \ldots, x_{i-1} \) in computing the value of the variable \( x_i \), then we obtain a new method called Gauss-Seidel iteration method.

### 3.10 Gauss-Seidel Iteration Method

As pointed out in Remark 22, we use the updated values of \( x_1, x_2, \ldots, x_{i-1} \) in computing the value of the variable \( x_i \). We assume that the pivots \( a_{ii} \neq 0 \), for all \( i \). We write the equations as

\[
\begin{align*}
a_{11}x_1 &= b_1 - (a_{12}x_2 + a_{13}x_3) \\
a_{22}x_2 &= b_2 - (a_{21}x_1 + a_{23}x_3) \\
a_{33}x_3 &= b_3 - (a_{31}x_1 + a_{32}x_2)
\end{align*}
\]

The Gauss-Seidel iteration method is defined as

\[
\begin{align*}
x_1^{(k+1)} &= \frac{1}{a_{11}} [b_1 - (a_{12}x_2^{(k)} + a_{13}x_3^{(k)})] \\
x_2^{(k+1)} &= \frac{1}{a_{22}} [b_2 - (a_{21}x_1^{(k+1)} + a_{23}x_3^{(k)})] \\
x_3^{(k+1)} &= \frac{1}{a_{33}} [b_3 - (a_{31}x_1^{(k+1)} + a_{32}x_2^{(k+1)})]
\end{align*}
\]

\( k = 0, 1, 2, \ldots \)

(14)

This method is also called the method of successive displacement.

We observe that (14) is same as writing the given system as

\[
\begin{align*}
a_{12}x_2^{(k+1)} &= - (a_{12}x_2^{(k)} + a_{13}x_3^{(k)}) \\
a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} &= b_2 - a_{23}x_3^{(k)} \\
a_{31}x_1^{(k+1)} + a_{32}x_2^{(k+1)} + a_{33}x_3^{(k+1)} &= b_3
\end{align*}
\]

(15)

**Remark 23**

- A sufficient condition for convergence of the Gauss-Seidel method is that the system of equations is diagonally dominant, that is, the coefficient matrix \( A \) is diagonally dominant.
- This implies that convergence may be obtained even if the system is not diagonally dominant.
- If the system is not diagonally dominant, we may exchange the equations, if possible, such that the new system is diagonally dominant and convergence is guaranteed.
- The necessary and sufficient condition for convergence is that the spectral radius of the iteration matrix \( H \) is less than one unit, that is, \( \rho(H) < 1 \), where \( \rho(H) \) is the largest eigen value in magnitude of \( H \).
• Testing of this condition is beyond the scope of the syllabus.
• If both the Gauss-Jacobi and Gauss-Seidel methods converge, then Gauss-Seidel method converges at least two times faster than the Gauss-Jacobi method.

Example
Find the solution of the system of equations
\[ 45x_1 + 2x_2 + 3x_3 = 58 \]
\[ -3x_1 + 22x_2 + 2x_3 = 47 \]
\[ 5x_1 + x_2 + 20x_3 = 67 \]
Correct to three decimal places, using the Gauss-Seidel iteration method.

Solution
The given system of equations is strongly diagonally dominant. Hence, we can expect fast convergence. Gauss-Seidel method gives the iteration.

First iteration
\[ x_1^{(1)} = \frac{1}{45} (58 - 2x_2^{(1)} - 3x_3^{(1)}) \]
\[ x_2^{(1)} = \frac{1}{22} (47 + 3x_1^{(1)} - 2x_3^{(1)}) \]
\[ x_3^{(1)} = \frac{1}{20} (67 - 5x_2^{(1)} - x_1^{(1)}) \]

Second iteration
\[ x_1^{(2)} = \frac{1}{45} (58 - 2x_2^{(2)} - 3x_3^{(2)}) = \frac{1}{45} (58 - 2(2.31212) - 3(2.91217)) = 0.99198, \]
\[ x_2^{(2)} = \frac{1}{22} (47 + 3x_1^{(2)} - 2x_3^{(2)}) = \frac{1}{22} (47 + 3(0.99198) - 2(2.91217)) = 2.00689, \]
\[ x_3^{(2)} = \frac{1}{20} (67 - 5x_2^{(2)} - x_1^{(2)}) = \frac{1}{20} (67 - 5(0.99198) - (2.00689)) = 3.00166 \]

Third iteration
\[ x_1^{(3)} = \frac{1}{45} (58 - 2x_2^{(3)} - 3x_3^{(3)}) = \frac{1}{45} (58 - 2(2.00689) - 3(3.00166)) = 0.99958, \]
\[ x_2^{(3)} = \frac{1}{22} (47 + 3x_1^{(3)} - 2x_3^{(3)}) = \frac{1}{22} (47 + 3(0.99958) - 2(3.00166)) = 1.99979, \]
\[ x_3^{(3)} = \frac{1}{20} (67 - 5x_2^{(3)} - x_1^{(3)}) = \frac{1}{20} (67 - 5(0.99958) - (1.99979)) = 3.00012. \]
Fourth iteration
\[ x_1^{(4)} = \frac{1}{45} (58 - 2x_2^{(3)} - 3x_3^{(3)}) = \frac{1}{45} (58 - 2(1.99979) - 3(3.00012)) = 1.00000, \]
\[ x_2^{(4)} = \frac{1}{22} (47 + 3x_1^{(4)} - 2x_3^{(3)}) = \frac{1}{22} (47 + 3(1.00000) - 2(3.00012)) = 1.99999, \]
\[ x_3^{(4)} = \frac{1}{20} (67 - 5x_1^{(4)} - x_2^{(4)}) = \frac{1}{20} (67 - 5(1.00000) - (1.99999)) = 3.00000. \]

We find
\[ |x_1^{(4)} - x_1^{(3)}| = |1.00000 - 0.99958| = 0.00042 \]
\[ |x_2^{(4)} - x_2^{(3)}| = |1.99999 - 1.99979| = 0.00020 \]
\[ |x_3^{(4)} - x_3^{(3)}| = |3.00000 - 3.00012| = 0.00012 \]

Since all the errors in magnitude are less than 0.0005, the required solution is
\[ x_1 = 1.0, x_2 = 1.99999, x_3 = 3.0. \]
Rounding to three decimal places, we get \[ x_1 = 1.0, x_2 = 2.0, x_3 = 3.0. \]

Example
Computationally show that Gauss-Seidel method applied to the system of equations
\[ 8x_1 - 6x_2 + 2x_3 = 23 \]
\[ -4x_1 + x_2 - x_3 = -8 \]
\[ x_1 - 3x_2 + 7x_3 = 17 \]
diverges.

- Take the initial approximations as \[ x_1 = 0.9, x_2 = -3.1, x_3 = 0.9. \]
- Interchange the first and second equations and solve the resulting system by the Gauss-Seidel method.
- Again take the initial approximations as \[ x_1 = 0.9, x_2 = -3.1, x_3 = 0.9, \] and obtain the result correct to two decimal places.
- The exact solution is \[ x_1 = 1.0, x_2 = -3.0, x_3 = 1.0. \]

Solution
Note that the system of equations is not diagonally dominant. Gauss-Seidel method gives the iteration
\[ x_1^{(k+1)} = \frac{[23 + 6x_2^{(k)} - 2x_3^{(k)}]}{3} \]
\[ x_2^{(k+1)} = [-8 + 4x_1^{(k+1)} + x_3^{(k+1)}] \]
\[ x_3^{(k+1)} = [17 - x_1^{(k+1)} + 3x_2^{(k+1)}]/7 \]

Starting with the initial approximations \[ x_1 = 0.9, x_2 = -3.1, x_3 = 0.9, \] we obtain the following results.

First iteration
\[ x_1^{(1)} = \frac{1}{3} [23 + 6(-3.1) - 2(0.9)] = \frac{1}{3} [23 + 6(-3.1) - 2(0.9)] = 0.8667, \]
\[ x_2^{(1)} = [-8 + 4(0.8667) + 0.9] = -3.6332, \]
\[ x_3^{(1)} = \frac{1}{7} [17 - 0.8667 + 3(-3.6332)] = 0.7477 \]
Second iteration

\[ x_1^{(2)} = \frac{1}{3} \left[ 23 + 6x_2^{(1)} - 2x_3^{(1)} \right] = \frac{1}{3} \left[ 23 + 6 \cdot (-3.6332) - 2(0.7477) \right] = -0.0982 \]

\[ x_2^{(2)} = -8 + 4x_1^{(2)} + x_3^{(1)} = [-8 + 4(-0.0982) + 0.7477] = -7.6451, \]

\[ x_3^{(2)} = \frac{1}{7} \left[ 17 - x_1^{(2)} + 3x_2^{(2)} \right] = \frac{1}{7} \left[ 17 + 0.0982 + 3(-7.6451) \right] = -0.8339. \]

Third iteration

\[ x_1^{(3)} = \frac{1}{3} \left[ 23 + 6x_2^{(2)} - 2x_3^{(2)} \right] = \frac{1}{3} \left[ 23 + 6 \cdot (-7.6451) - 2(-0.8339) \right] = -7.0676 \]

\[ x_2^{(3)} = [-8 + 4x_1^{(3)} + x_3^{(2)}] = [-8 + 4(-7.0676) - 0.8339] = -37.1043, \]

\[ x_3^{(3)} = \frac{1}{7} \left[ 17 - x_1^{(3)} + 3x_2^{(3)} \right] = \frac{1}{7} \left[ 17 + 7.0676 + 3(-37.1043) \right] = -12.4636. \]

It can be observed that the iterations are diverging very fast.

Now, we exchange the first and second equations to obtain the system.

\[-4x_1 + x_2 - x_3 = -8 \]

\[3x_1 - 6x_2 + 2x_3 = 23 \]

\[x_1 - 3x_2 + 7x_3 = 17.\]

The system of equations is now diagonally dominant. Gauss-Seidel method gives iteration

\[ x_1^{(k+1)} = \frac{8 + x_2^{(k)} - x_3^{(k)}}{4} \]

\[ x_2^{(k+1)} = 23 - 3x_1^{(k+1)} - 2x_3^{(k)} \]

\[ x_3^{(k+1)} = \frac{1}{7} \left[ 17 - x_1^{(k+1)} + 3x_2^{(k+1)} \right]. \]

Starting with the initial approximations \( x_1 = 0.9, x_2 = -3.1, x_3 = 0.9 \), we obtain the following results.

First iteration

\[ x_1^{(1)} = \frac{1}{4} \left[ 8 + x_2^{(0)} - x_3^{(0)} \right] = \frac{1}{4} \left[ 8 - 3.1 - 0.9 \right] = 1.0 \]

\[ x_2^{(1)} = \frac{1}{6} \left[ 23 - 3x_1^{(1)} - 2x_3^{(0)} \right] = \frac{1}{6} \left[ 23 - 3(1.0) - 2(0.9) \right] = -3.0333 \]

\[ x_3^{(1)} = \frac{1}{7} \left[ 17 - x_1^{(1)} + 3x_2^{(1)} \right] = \frac{1}{7} \left[ 17 - 1.0 + 3(-3.0333) \right] = 0.9857. \]

Second iteration

\[ x_1^{(2)} = \frac{1}{4} \left[ 8 + x_2^{(1)} - x_3^{(1)} \right] = \frac{1}{4} \left[ 8 - 3.0333 - 0.9857 \right] = 0.9953, \]

\[ x_2^{(2)} = \frac{1}{6} \left[ 23 - 3x_1^{(2)} - 2x_3^{(1)} \right] = \frac{1}{6} \left[ 23 - 3(0.9953) - 2(0.9857) \right] = -3.0071, \]

\[ x_3^{(2)} = \frac{1}{7} \left[ 17 - x_1^{(2)} + 3x_2^{(2)} \right] = \frac{1}{7} \left[ 17 - 0.9953 + 3(-3.0071) \right] = 0.9976. \]

Third iteration

\[ x_1^{(3)} = \frac{1}{4} \left[ 8 + x_2^{(2)} - x_3^{(2)} \right] = \frac{1}{4} \left[ 8 - 3.0071 - 0.9976 \right] = 0.9988, \]

\[ x_2^{(3)} = \frac{1}{6} \left[ 23 - 3x_1^{(3)} - 2x_3^{(2)} \right] = \frac{1}{6} \left[ 23 - 3(0.9988) - 2(0.9976) \right] = -3.0014 \]

\[ x_3^{(3)} = \frac{1}{7} \left[ 17 - x_1^{(3)} + 3x_2^{(3)} \right] = \frac{1}{7} \left[ 17 - 0.9988 + 3(-3.0014) \right] = 0.9996. \]
Fourth iteration
\[ x_1^{(4)} = \frac{1}{4} [8 + x_2^{(3)}] - x_8^{(3)} = \frac{1}{4} [8 - 3.0014 - 0.9996] = 0.9998, \]
\[ x_2^{(4)} = \frac{1}{6} [23 - 3x_1^{(4)} - 2x_8^{(3)}] = \frac{1}{6} [23 - 3(0.9998) - 2(0.9996)] = -3.0002 \]
\[ x_3^{(4)} = \frac{1}{7} [17 - x_1^{(4)} + 3x_2^{(4)}] = \frac{1}{7} [17 - 0.9998 + 3(-3.0002)] = 0.9999. \]

We find
\[
\begin{align*}
|x_1^{(4)} - x_1^{(3)}| &= |0.9998 - 0.9988| = 0.0010, \\
|x_2^{(4)} - x_2^{(3)}| &= |-3.0002 + 3.0014| = 0.0012 \\
|x_3^{(4)} - x_3^{(3)}| &= |0.9999 - 0.9996| = 0.0003.
\end{align*}
\]

Since all the errors in magnitude are less than 0.005, the required solution is
\[ x_1 = 0.9998, \quad x_2 = -3.0002, \quad x_3 = 0.9999 \]

Rounding to two decimal places, we get \( x_1 = 1.0, \quad x_2 = -3.0, \quad x_3 = 1.0 \)

### 3.11 Eigen Value Problem
- The concept of eigen values and finding eigen values and eigen vectors of a given matrix, is very important for engineers and scientists.
- Consider the eigen value problem
  \[ Ax = \lambda x. \]  
  \[ (15) \]
- The eigen values of a matrix A are given by the roots of the characteristic equation
  \[ |A - \lambda I| = 0 \]  
  \[ (16) \]
- If the matrix A is of order n, then expanding the determinant, we obtain the characteristic equation as
  \[ p(\lambda) = (-1)^n \lambda^n + a_1 \lambda^{n-1} + \ldots + a_{n-1} \lambda + a_n = 0 \]  
  \[ (17) \]
- For any given matrix we write the characteristic equation (17), expand it and find the roots \( \lambda_1, \lambda_2, \ldots, \lambda_n \), which are the eigen values.
- The roots may be real, repeated or complex. Let \( x_i \) be the solution of the system of the homogeneous equations (15), corresponding to the eigen value \( \lambda_i \).
- These vectors \( x_i, i = 1, 2, \ldots, n \) are called the eigen vectors of the system.
- There are several methods for finding the eigen values of a general matrix or a symmetric matrix.

### 3.12 Power Method
- The method for finding the largest eigen value in magnitude and the corresponding eigen vector of the eigen value problem \( Ax = \lambda x \), is called the power method.
- What is the importance of this method? Let us re-look at the Remarks 20 and 23.
- The necessary and sufficient condition for convergence of the Gauss-Jacobi and Gauss-Seidel iteration methods is that the spectral radius of the iteration matrix H is less than one unit, that is, \( \rho(H) < 1 \), where \( \rho(H) \) is the largest eigen value in magnitude of H.
- If we write the matrix formulations of the methods, then we know H.
- We can now find the largest eigen value in magnitude of H, which determines whether the methods converge or not.
- We assume that \( \lambda_1 > \lambda_2 > \ldots > \lambda_n \) are distinct eigen values such that
  \[ |\lambda_1| > |\lambda_2| > \ldots > |\lambda_n| \]  
  \[ (18) \]
Let \( v_1, v_2, \ldots, v_n \) be the eigen vectors corresponding to the eigen values \( \lambda_1, \lambda_2, \ldots, \lambda_n \), respectively.

The method is applicable if a complete system of \( n \) linearly independent eigen vectors exist, even though some of the eigen values \( \lambda_2, \lambda_3, \ldots, \lambda_n \), may not be distinct.

The \( n \) linearly independent eigen vectors form an \( n \)-dimensional vector space.

Any vector \( v \) in this space of eigen vectors \( v_1, v_2, \ldots, v_n \) can be written as a linear combination of these vectors. That is,

\[
 v = c_1 v_1 + c_2 v_2 + \ldots + c_n v_n
\]

(19)

Premultiplying by \( A \) and substituting \( Av_1 = \lambda_1 v_1, Av_2 = \lambda_2 v_2, \ldots, Av_n = \lambda_n v_n \), we get

\[
 Av = c_1 \lambda_1 v_1 + c_2 \lambda_2 v_2 + \ldots + c_n \lambda_n v_n
\]

\[
 = \lambda_1 \left[ c_1 v_1 + c_2 \left( \frac{\lambda_2}{\lambda_1} \right) v_2 + \ldots + c_n \left( \frac{\lambda_n}{\lambda_1} \right) v_n \right]
\]

Premultiplying repeatedly by \( A \) and simplifying, we get

\[
 A^2 v = \lambda_1^2 \left[ c_1 v_1 + c_2 \left( \frac{\lambda_2}{\lambda_1} \right)^2 v_2 + \ldots + c_n \left( \frac{\lambda_n}{\lambda_1} \right)^2 v_n \right]
\]

\[
 A^3 v = \lambda_1^3 \left[ c_1 v_1 + c_2 \left( \frac{\lambda_2}{\lambda_1} \right)^3 v_2 + \ldots + c_n \left( \frac{\lambda_n}{\lambda_1} \right)^3 v_n \right]
\]

\[
 A^k v = \lambda_1^k \left[ c_1 v_1 + c_2 \left( \frac{\lambda_2}{\lambda_1} \right)^k v_2 + \ldots + c_n \left( \frac{\lambda_n}{\lambda_1} \right)^k v_n \right]
\]

(20)

As \( k \to \infty \), the right hand sides of (20) and (21) tend to \( \lambda_1^k c_1 v_1 \), and \( \lambda_1^k \sum_{i=2}^{n} c_i v_i \), since

\[
 | \lambda_i / \lambda_1 | < 1, \ i = 2, 3, \ldots, n
\]

Both the right hand side vectors in (21), (22)

\[
 [c_1 v_1 + c_2 (\lambda_2 / \lambda_1)^k v_2 + \ldots + c_n (\lambda_n / \lambda_1)^k v_n]
\]

and

\[
 [c_1 v_1 + c_2 (\lambda_2 / \lambda_i)^{k+1} v_2 + \ldots + c_n (\lambda_n / \lambda_i)^{k+1} v_n]
\]

tend to \( c_1 v_1 \), which is the eigen vector corresponding to \( \lambda_1 \).

The eigen value \( \lambda_i \) is obtained as the ratio of the corresponding components of \( A^{k+1} v \) and \( A^k v \). That is,

\[
 \lambda_i = \lim_{k \to \infty} \frac{(A^{k+1} v)_r}{(A^k v)_r}, \ r = 1, 2, 3, \ldots, n
\]

Where the suffix \( r \) denotes the \( r \)th component of the vector. Therefore, we obtain \( n \) ratios, all of them tending to the same value, which is the largest eigen value in magnitude, \( |\lambda_1| \)

**When do we stop the iteration?**
The iterations are stopped when all the magnitudes of the differences of the ratios are less than the given error tolerance.

**Remark 24**

- The choice of the initial approximation vector \( v_0 \) is important.
- If no suitable approximation is available, we can choose \( v_0 \) with all its components as one unit, that is, \( v_0 = [1, 1, 1, \ldots, 1]^T \).
- However, this initial approximation to the vector should be non-orthogonal to \( v_i \).
Remark 25

- Faster convergence is obtained when \(|\lambda_2| \ll |\lambda_1|\).
- As \(k \to \infty\), premultiplication each time by \(A\), may introduce round-off errors.
- In order to keep the round-off errors under control, we normalize the vector before premultiplying by \(A\).
- The normalization that we use is to make the largest element in magnitude as unity.
- If we use this normalization, a simple algorithm for the power method can be written as follows.
  \[
  y_{k+1} = Ay_k, \\
  v_{k+1} = y_{k+1}/m_{k+1}
  \]
  (22)(1.57)  (23)(1.58)
- Where \(m_{k+1}\) is the largest element in magnitude of \(y_{k+1}\).
- Now, the largest element in magnitude of \(v_{k+1}\) is one unit. Then (21) can be written as
  \[
  \lambda_1 = \lim_{k \to \infty} \frac{(y_{k+1})_r}{(v_k)_r}, \quad r = 1, 2, 3, ..., n
  \]
  (24)(1.59)
  and \(v_{k+1}\) is the required eigen vector.

Remark 26

It may be noted that as \(k \to \infty\), \(m_{k+1}\) also gives \(|\lambda_1|\)

Remark 27

- Power method gives the largest eigen value in magnitude.
- If the sign of the eigen value is required, then we substitute this value in the determinant \(|A - \lambda I|\) and find its value.
- If this value is approximately zero, then the eigen value is of positive sign.
- Otherwise, it is of negative sign.

Example

Determine the dominant eigen value of

\[
A = \begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\]

by power method.

Solution

- Let the initial approximation to the eigen vector be \(v_0\). Then, the power method is given by
  \[
  y_{k+1} = Ay_k, \\
  v_{k+1} = y_{k+1}/m_{k+1}
  \]
  Where \(m_{k+1}\) is the largest element in magnitude of \(y_{k+1}\). The dominant eigen value in magnitude is given by
  \[
  \lambda_1 = \lim_{k \to \infty} \frac{(y_{k+1})_r}{(v_k)_r}, \quad r = 1, 2, 3, ..., n
  \]
  and \(v_{k+1}\) is the required eigen vector.

Let \(v_0 = [1 \ 1]^T\). We have the following results.
\[ y_1 = A v_0 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 7 \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \end{bmatrix}, \quad m_1 = 7, \quad v_1 = \frac{y_1}{m_1} = \frac{1}{7} \begin{bmatrix} 3 \\ 7 \end{bmatrix} = \begin{bmatrix} 0.42857 \\ 1 \end{bmatrix}. \]

\[ y_2 = A v_1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0.42857 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.42857 \\ 5.28571 \end{bmatrix}, \quad m_2 = 5.28571, \]

\[ v_2 = \frac{y_2}{m_2} = \frac{1}{5.28571} \begin{bmatrix} 2.42857 \\ 5.28571 \end{bmatrix} = \begin{bmatrix} 0.45946 \\ 1 \end{bmatrix}. \]

\[ y_3 = A v_2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0.45946 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.45946 \\ 5.37838 \end{bmatrix}, \quad m_3 = 5.37838, \]

\[ v_3 = \frac{y_3}{m_3} = \frac{1}{5.37838} \begin{bmatrix} 2.45946 \\ 5.37838 \end{bmatrix} = \begin{bmatrix} 0.45729 \\ 1 \end{bmatrix}. \]

\[ y_4 = A v_3 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0.45729 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.45729 \\ 5.37187 \end{bmatrix}, \quad m_4 = 5.37187, \]

\[ v_4 = \frac{y_4}{m_4} = \frac{1}{5.37187} \begin{bmatrix} 2.45729 \\ 5.37187 \end{bmatrix} = \begin{bmatrix} 0.45744 \\ 1 \end{bmatrix}. \]

\[ y_5 = A v_4 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0.45744 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.45744 \\ 5.37232 \end{bmatrix}, \quad m_5 = 5.37232, \]

\[ y_6 = A v_5 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0.45743 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.45743 \\ 5.37229 \end{bmatrix}. \]

Now, we find the ratios
\[ \lambda_i = \lim_{k \to \infty} \frac{(y_{i+1})_r}{(v_k)_r}, \quad r = 1, 2 \]

We obtain the ratios as
\[ \frac{2.45743}{0.45743} = 5.37225, \quad 5.37229. \]

The magnitude of the error between the ratios is \[ 5.37225 - 5.37229 \] = 0.00004 < 0.00005. Hence, the dominant eigen value, correct to four decimal places is 5.3722.
Summary

- Direct methods produce the exact solution after a finite number of steps (disregarding the round-off errors).
- In these methods, we can determine the total number of operations (additions, subtractions, divisions and multiplications).
- This number is called the operational count of the method.
- Iterative methods are based on the idea of successive approximations.
- We start with an initial approximation to the solution vector $x = x_0$, and obtain a sequence of approximate vectors $x_0, x_1, ..., x_k, ...$, which in the limit as $k \to \infty$, converge to the exact solution vector $x$.
- Now, we derive some direct methods.
- The following operations on the rows of a matrix $A$ are called the elementary row transformations (operations).
- Interchange of any two rows. If we interchange the $i$th row with the $j$th row, then we usually denote the operation as $R_i \leftrightarrow R_j$.
- Division/multiplication of any row by a non-zero number $p$. If the $i$th row is multiplied by $p$, then we usually denote this operation as $pR_i$.
- The method is based on the idea of reducing the given system of equations $Ax = b$, to an upper triangular system of equations $Ux = z$, using elementary row operations.
- We know that these two systems are equivalent.
- That is, the solutions of both the systems are identical.
- This reduced system $Ux = z$, is then solved by the back substitution method to obtain the solution vector $x$.
- To avoid computational errors, we follow the procedure of partial pivoting.
- In the first stage of elimination, the first column of the augmented matrix is searched for the largest element in magnitude and brought as the first pivot by interchanging the first row of the augmented matrix (first equation) with the row (equation) having the largest element in magnitude.
- The method is based on the idea of reducing the given system of equations $Ax = b$, to a diagonal system of equations $Ix = d$, where $I$ is the identity matrix, using elementary row operations.
- We know that the solutions of both the systems are identical.
- This reduced system gives the solution vector $x$.
- The first step is same as in Gauss elimination method, that is, we change the elements below the first pivot as zeros, using the elementary row transformations.
- From the second step onwards, we change the elements below and above the pivots as zeros, using the elementary row transformations.

References


Recommended Reading

Self Assessment

1. _______ methods produce the exact solution after a finite number of steps.
   a. Direct
   b. Straight
   c. Shortest
   d. Express

2. Iterative methods are based on the idea of __________ approximations.
   a. consecutive
   b. successive
   c. succeeding
   d. straight

3. If the system of equations has some special forms, then the solution is obtained__________.
   a. straight
   b. openly
   c. honestly
   d. directly

4. The unknowns are obtained by back substitution and this procedure is called the back __________ method.
   a. swap
   b. substitution
   c. exchange
   d. switch

5. A pivot, all the elements below that pivot in that column are made _______.
   a. negative
   b. zeros
   c. one
   d. negative

6. If a pivot is a very small number, then ________ by it introduces large round-off errors and the solution may
   contain large errors.
   a. division
   b. multiplication
   c. addition
   d. summation

7. To avoid computational errors, we follow the procedure of _______ pivoting.
   a. partial
   b. prejudiced
   c. one-sided
   d. unfair
8. Gauss elimination method is a ______ method.
   a. shortest
   b. through
   c. direct
   d. express

9. The Gauss-Jordan method looks very elegant as the solution is obtained ______.
   a. straight
   b. openly
   c. freely
   d. directly

10. The most important application of this method is to find the ______ of a non-singular matrix.
    a. opposite
    b. contrary
    c. inverse
    d. converse
Chapter IV
Interpolation

Aim
The aim of this chapter is to:

• introduce the concept of interpolation
• explain Weierstrass Approximation theorem
• describe Lagrange polynomial
• discuss the divided differences

Objectives
The objectives of this chapter are to:

• explain the Newton’s Interpolatory divided-difference formula
• elaborate Newton forward-difference formula
• explain the inverse interpolation
• enlist the errors in numerical differentiation

Learning outcome
At the end of this chapter, the students will be able to:

• discuss numerical integration
• clarify Newton-Cote’s quadrature formula
• state the concept of Trapezoidal rule
• discuss the Simpson’s one-third rule
4.1 Introduction to Interpolation

- A census of the population of the United States is taken every 10 years.
- The following table lists the population, in thousands of people, from 1940 to 1990.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Population (in thousands)</td>
<td>132,165</td>
<td>151,326</td>
<td>179,323</td>
<td>203,302</td>
<td>226,542</td>
<td>249,633</td>
</tr>
</tbody>
</table>

Table 4.1 List of population

In reviewing these data, we might ask whether they could be used to provide a reasonable estimate of the population, say, in 1965 or even in the year 2010.

Predictions of this type can be obtained by using a function that fits the given data. This process is called interpolation.

One of the most useful and well-known classes of functions mapping the set of real numbers into itself is the class of algebraic polynomials, the set of functions of the form.

\[ P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0, \quad (1) \]

Where \( n \) is a nonnegative integer and \( a_0, \ldots, a_n \) are real constants.

One reason for their importance is that they uniformly approximate continuous functions.

Given any function, defined and continuous on a closed and bounded interval, there exists a polynomial that is as “close” to the given function as desired.

This result is expressed precisely in the following theorem.
4.2 Theorem: Weierstrass Approximation Theorem

- Suppose that $f$ is defined and continuous on $[a, b]$. For each $\epsilon > 0$, there exists a polynomial $P(x)$, with the property that $|f(x) - P(x)| < \epsilon$, for all $x$ in $[a, b]$ \hspace{1cm} (2)

- The proof of this theorem can be found in most elementary texts on real analysis.
- Another important reason for considering the class of polynomials in the approximation of functions is that the derivative and indefinite integral of a polynomial are easy to determine and are also polynomials.
- For these reasons, polynomials are often used for approximating continuous functions.
- The Taylor polynomials were introduced in the first section of the chapter, where they were described as one of the fundamental building blocks of numerical analysis.
- Given this prominence, you might assume that polynomial interpolation would make heavy use of these functions. However, this is not the case.
- The Taylor polynomials agree as closely as possible with a given function at a specific point, but they concentrate their accuracy near that point.
- A good interpolation polynomial needs to provide a relatively accurate approximation over an entire interval, and Taylor polynomials do not generally do this.
- For example, suppose we calculate the first six Taylor polynomials about $X_0 = 0$ for $f(x) = e^x$.
- Since the derivatives of $f(x)$ are all $e^x$, which evaluated at $X_0 = 0$ gives 1, the Taylor polynomials are

$$
P_0(x) = 1, \quad P_1(x) = 1 + x, \quad P_2(x) = 1 + x + \frac{x^2}{2}, \quad P_3(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6},
$$

$$
P_4(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}, \quad \text{and} \quad P_5(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120}.
$$

(3)
Although better approximations are obtained for \( f(x) = e^x \) if higher-degree Taylor polynomials are used, this is not true for all functions.

Consider, as an extreme example, using Taylor polynomials of various degrees for \( f(x) = \frac{1}{x} \) expanded about \( x_0 = 1 \) to approximate \( f(3) = \frac{1}{3} \). Since

\[
 f(x) = x^{-1}, \quad f'(x) = -x^{-2}, \quad f''(x) = (-1)^2 2 \cdot x^{-3},
\]

and, in general,

\[
 f^{(k)}(x) = (-1)^k k! x^{-k-1}, \tag{4}
\]

the Taylor polynomials are

\[
 P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(1)}{k!} (x - 1)^k = \sum_{k=0}^{n} (-1)^k (x - 1)^k \tag{5}
\]

To approximate \( f(3) = \frac{1}{3} \) by \( P_n(3) \) for increasing values of \( n \), we obtain the values in Table given below, rather a dramatic failure!

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_n(3) )</td>
<td>1</td>
<td>-1</td>
<td>3</td>
<td>-5</td>
<td>11</td>
<td>-21</td>
<td>43</td>
<td>-85</td>
</tr>
</tbody>
</table>

Table 4.2 For approximate value “n”

Since the Taylor polynomials have the property that all the information used in the approximation is concentrated at the single point \( x_0 \).

The type of difficulty that occurs here is quite common and limits Taylor polynomial approximation to the situation in which approximations are needed only at points close to \( x_0 \).

For ordinary computational purposes it is more efficient to use methods that include information at various points, which we consider in the remainder of this chapter.

The primary use of Taylor polynomials in numerical analysis is not for approximation purposes but for the derivation of numerical techniques and error estimation.
4.3 Interpolation and the Lagrange Polynomial

- Since the Taylor polynomials are not appropriate for interpolation, alternative methods are needed.
- In this section we find approximating polynomials that are determined simply by specifying certain points on the plane through which they must pass.
- The problem of determining a polynomial of degree one that passes through the distinct points \((x_0, y_0)\) and \((x_1, y_1)\) is the same as approximating a function \(f\) for which \(f(x_0) = y_0\) and \(f(x_1) = y_1\) by means of a first-degree polynomial interpolating, or agreeing with, the values of \(f\) at the given points. We shall first define the functions:

\[
L_0(x) = \frac{x - x_1}{x_0 - x_1} \quad \text{and} \quad L_1(x) = \frac{x - x_0}{x_1 - x_0},
\]

and then define

\[
P(x) = L_0(x)f(x_0) + L_1(x)f(x_1)
\]

(7)

Since

\[
L_0(x_0) = 1, \quad L_0(x_1) = 0, \quad L_1(x_0) = 0, \quad \text{and} \quad L_1(x_1) = 1,
\]

(8)

We have,

\[
P(x_0) = 1 \cdot f(x_0) + 0 \cdot f(x_1) = f(x_0) = y_0
\]

(9)

(10)

So \(P\) is the unique linear function passing through \((x_0, y_0)\) and \((x_1, y_1)\).

To generalize the concept of linear interpolation, consider the construction of a polynomial of degree at most \(n\) that passes through the \(n + 1\) points.

\[
(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n)).
\]

(11)

In this case, we need to construct for each \(k = 0, 1, \ldots, n\), a function \(L_{n,k}(X)\) with the property that \(L_{n,k}(X_i) = 0\) when \(i \neq k\) and \(L_{n,k}(X_k) = 1\).

To satisfy \(L_{n,k}(X_i) = 0\) for each \(i \neq k\) requires that the numerator of \(L_{n,k}(x)\) contains the term

\[
(x - x_0)(x - x_1) \ldots (x - x_k)(x - x_{k+1}) \ldots (x - x_n).
\]

(12)

To satisfy \(L_{n,k}(X_k) = 1\), the denominator of \(L_{n,k}(x)\) must be equal to this term evaluated at \(x = x_k\). Thus,
The interpolating polynomial is easily described once the form of $L_{n,k}$ is known. This polynomial, called the $n$th Lagrange interpolating polynomial, is defined in the following theorem.

If $x_0, x_1, \ldots, x_n$ are $n + 1$ distinct numbers and $f$ is a function whose values are given at these numbers, then a unique polynomial $P(x)$ of degree at most $n$ exists with

$$f(x_k) = P(x_k), \text{ for each } k = 0, 1, \ldots, n$$

This polynomial is given by

$$P(x) = f(x_0)L_{n,0}(x) + \cdots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x).$$

Where, for each $k = 0, 1, \ldots, n$,

$$L_{n,k}(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)}$$

$$= \prod_{i=0}^{n} \frac{x-x_i}{x_k-x_i}.$$

### 4.4 Divided Differences

- Iterated interpolation was used in the previous section to generate successively higher degree polynomial approximations at a specific point.
- Divided-difference methods introduced in this section are used to successively generate the polynomials themselves.
- Our treatment of divided-difference methods will be brief since the results in this section will not be used extensively in subsequent material.
- Older texts on numerical analysis have extensive treatments of divided-difference methods.
- Suppose that $P_n(x)$ is the $n$th Lagrange polynomial that agrees with the function $f$ at the distinct numbers.
- The divided differences of $f$ with respect to $x_0, x_1, \ldots, x_n$ are used to express $P_n(x)$ in the form

$$P_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots + a_n(x-x_0)(x-x_1)\cdots(x-x_{n-1}).$$

For appropriate constants $a_0, a_1, \ldots, a_n$.

- To determine the first of these constants, $a_0$, note that if $P_n(x)$ is written in the form of Eq. (16), then evaluating $P_n(x)$ at $x_0$ leaves only the constant term $a_0$; that is,

$$a_0 = P_n(x_0) = f(x_0)$$

- Similarly, when $P(x)$ is evaluated at $x_i$, the only nonzero terms in the evaluation of $P_n(x_i)$ are the constant and linear terms,

$$f(x_0) + a_1(x_1-x_0) = P_n(x_1) = f(x_1);$$

so,

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1-x_0}$$
• The zeroth divided difference of the function $f$ with respect to $X_i$, denoted $f[X_i]$, is simply the value of $f$ at $X_i$: 
\[ f[X_i] = f(x_i) \]  
(20)

• The remaining divided differences are defined inductively; the first divided difference of $f$ with respect to $x_i$ and $x_{i+1}$ is denoted $f[x_i, x_{i+1}]$ and is defined as 
\[ f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i} \]  
(21)

• The second divided difference, $f[x_i, x_{i+1}, x_{i+2}]$ is defined as 
\[ f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+2}, x_{i+1}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i} \]  
(22)

• Similarly, after the $(k - 1)$st divided differences, 
\[ f[x_i, x_{i+1}, x_{i+2}, \ldots, x_{i+k-1}] \quad \text{and} \quad f[x_{i+1}, x_{i+2}, \ldots, x_{i+k-1}, x_{i+k}] \]  
have been determined, the $k$th divided difference relative to $x_i, x_{i+1}, x_{i+2}, \ldots, x_{i+k}$ is given by 
\[ f[x_i, x_{i+1}, \ldots, x_{i+k-1}, x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i} \]  
(23)

• With this notation, Eq. 20 can be reexpressed as $a_0 = f[x_0, x_0]$, and the interpolating polynomial in Eq. 19 is 
\[ \begin{align*}
P_n(x) &= f[x_0] + f[x_0, x_1](x - x_0) + a_2(x - x_0)(x - x_1) \\
&\quad + \cdots + a_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}).
\end{align*} \]  
(25)

• As might be expected from the evaluation of $a_0$ and $a_1$, the required constants are 
\[ a_k = f[x_0, x_1, x_2, \ldots, x_k], \quad k = 1, 2, \ldots, n. \]  
(26)

for each $k = 0, 1, \ldots, n$. So $P_n(x)$ can be rewritten as 
\[ P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](x - x_0) \cdots (x - x_{k-1}) \]  
(27)

• The value of $f[x_0, x_1, \ldots, x_n]$ is independent of the order of the numbers $x_0, x_1, \ldots, x_n$.

• This equation is known as Newton’s interpolatory divided difference formula.

• The generation of the divided differences is outlined in the Table given below.

• Two fourth and one fifth difference could also be determined from these data.

### Table 4.3 Generation of the divided differences

<table>
<thead>
<tr>
<th>$x$</th>
<th>$f(x)$</th>
<th>First divided differences</th>
<th>Second divided differences</th>
<th>Third divided differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>$f[x_0]$</td>
<td>$f[x_0, x_1]$ = $\frac{f[x_1] - f[x_0]}{x_1 - x_0}$</td>
<td>$f[x_1, x_2]$ = $\frac{f[x_2] - f[x_1]}{x_2 - x_1}$</td>
<td>$f[x_2, x_3]$ = $\frac{f[x_3] - f[x_2]}{x_3 - x_2}$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>$f[x_1]$</td>
<td>$f[x_0, x_1, x_2]$ = $\frac{f[x_2] - f[x_0]}{x_2 - x_0}$</td>
<td>$f[x_1, x_2, x_3]$ = $\frac{f[x_3] - f[x_1]}{x_3 - x_1}$</td>
<td>$f[x_2, x_3, x_4]$ = $\frac{f[x_4] - f[x_2]}{x_4 - x_2}$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$f[x_2]$</td>
<td>$f[x_1, x_2, x_3]$ = $\frac{f[x_3] - f[x_1]}{x_3 - x_1}$</td>
<td>$f[x_2, x_3, x_4]$ = $\frac{f[x_4] - f[x_2]}{x_4 - x_2}$</td>
<td>$f[x_3, x_4, x_5]$ = $\frac{f[x_5] - f[x_3]}{x_5 - x_3}$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$f[x_3]$</td>
<td>$f[x_2, x_3, x_4]$ = $\frac{f[x_4] - f[x_2]}{x_4 - x_2}$</td>
<td>$f[x_3, x_4, x_5]$ = $\frac{f[x_5] - f[x_3]}{x_5 - x_3}$</td>
<td>$f[x_4, x_5]$ = $\frac{f[x_5] - f[x_4]}{x_5 - x_4}$</td>
</tr>
</tbody>
</table>

77
4.5 Newton’s Interpolatory Divided-Difference Formula

- To obtain the divided-difference coefficients of the interpolatory polynomial \( P \) on the \((n+1)\) distinct numbers \( x_0, x_1, \ldots, x_n \) for the function \( f \):
- INPUT numbers \( x_0, x_1, \ldots, x_n \); values \( f(x_0), f(x_1), \ldots, f(x_n) \) as \( F_{0,0}, \ldots, F_{n,0} \).
- OUTPUT the numbers \( F_{1,0}, \ldots, F_{n,n} \), where

\[
P(x) = \sum_{i=0}^{n} F_{i,0} \prod_{j=0}^{i-1} (x - x_j)
\]  

(28)

Step 1 For \( i = 1, 2, \ldots, n \)
- For \( j=1,2,\ldots,i \)
- set \( F_{i,j} = \frac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}} \)  

(29)

Step 2 OUTPUT \((F_{1,0}, \ldots, F_{n,n});\) \((F_{i,i} \text{ is } f[x_0, x_1, \ldots, x_i].)\)

STOP.

Example
Solve given table using coefficients of the Newton forward divided-difference form of the interpolatory polynomial are along the diagonal in the table.

The polynomial is:

\[
P_4(x) = 0.7651977 - 0.4837057(x - 1.0) - 0.1087339(x - 1.0)(x - 1.3) + 0.0658784(x - 1.0)(x - 1.3)(x - 1.6) + 0.0018251(x - 1.0)(x - 1.3)(x - 1.6)(x - 1.9).
\]  

(30)

Table 4.4 Polynomial in equation (30)

<table>
<thead>
<tr>
<th>( i )</th>
<th>( x_i )</th>
<th>( f[x_i] )</th>
<th>( f[x_{i-1}, x_i] )</th>
<th>( f[x_{i-2}, x_{i-1}, x_i] )</th>
<th>( f[x_{i-3}, \ldots, x_i] )</th>
<th>( f[x_{i-4}, \ldots, x_i] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
<td>0.7651977</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.3</td>
<td>0.6200860</td>
<td>-0.4837057</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.6</td>
<td>0.4554022</td>
<td>-0.5489460</td>
<td>-0.0494433</td>
<td>0.0658784</td>
<td>0.0018251</td>
</tr>
<tr>
<td>3</td>
<td>1.9</td>
<td>0.2818186</td>
<td>-0.5786120</td>
<td>0.0118183</td>
<td>0.0680685</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2.2</td>
<td>0.1103623</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.6 Theorem

Suppose that \( f \in C^n[a, b] \) and \( x_0, x_1, \ldots, x_n \) are distinct numbers in \([a, b]\). Then a number \( \xi \) exists in \((a, b)\) with

\[
f[x_0, x_1, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}
\]  

(32)

Proof

...
Let
\[ g(x) = f(x) - P_n(x) \]  
(33)

- Since \( f(x_i) = P_n(x_i) \), for each \( i = 0, 1, \ldots, n \), the function \( g \) has \( n + 1 \) distinct zeros in \([a,b] \).
- The Generalized Rolle’s Theorem implies that a number \( \xi \) in \((a, b) \) exists with \( g(n)(\xi) = 0 \), so
\[ 0 = f^{(n)}(\xi) - P_n^{(n)}(\xi) \]  
(34)

Since \( P_n(x) \) is a polynomial of degree \( n \) whose leading coefficient is \( f[x_0, x_1, \ldots, x_n] \),
\[ P_n^{(n)}(x) = n! f[x_0, x_1, \ldots, x_n], \]  
(35)

- For all values of \( x \). As a consequence,
\[ f[x_0, x_1, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!} \]  
(34)

- Newton’s interpolatory divided-difference formula can be expressed in a simplified form when \( x_0, x_1, \ldots, x_n \) are arranged consecutively with equal spacing.
- In this case, we introduce the notation \( h = x_{i+1} - x_i \) for each \( i = 0, 1, \ldots, n - 1 \) and let \( x = x_0 + sh \).
- Then the difference \( x - x_i \) can be written as \( x - x_i = (s - i)h \).
- So equation (27) becomes
\[ P_n(x) = P_n(x_0 + sh) = f[x_0] + shf[x_0, x_1] + (s - 1)h^2 f[x_0, x_1, x_2] + \cdots + s(s - 1)(s - n + 1)h^n f[x_0, x_1, \ldots, x_n] \]
\[ = \sum_{k=0}^{n} s(s - 1) \cdots (s - k + 1)h^k f[x_0, x_1, \ldots, x_k]. \]  
(35)

- Using binomial-coefficient notation,
\[ \binom{s}{k} = \frac{s(s - 1) \cdots (s - k + 1)}{k!} . \]  
(36)

- We can express \( P_n(x) \) compactly as
\[ P_n(x) = P_n(x_0 + sh) = f[x_0] + \sum_{k=1}^{n} \binom{s}{k} k! h^k f[x_0, x_1, \ldots, x_k] \]  
(37)

- This formula is called the Newton forward divided-difference formula.
- Another form, called the Newton forward-difference formula, is constructed by making use of the forward difference notation \( \Delta \Delta \) introduced in Aitken’s \( \Delta^2 \Delta^2 \) method. With this notation,
\[ f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{1}{h} \Delta f(x_0) \]
\[ f[x_0, x_1, x_2] = \frac{1}{2h} \left( \frac{\Delta f(x_1) - \Delta f(x_0)}{h} \right) = \frac{1}{2h^2} \Delta^2 f(x_0), \]

and, in general,
\[ f[x_0, x_1, \ldots, x_k] = \frac{1}{k! h^k} \Delta^k f(x_0) \]  
(38)

Then Eq. (37) has the following formula.
**Newton Forward-Difference Formula**

\[ P_n(x) = f[x_0] + \sum_{k=1}^{n} \binom{s}{k} \Delta^k f(x_0) \]  

- If the interpolating nodes are reordered as \( x_n, x_{n-1}, \ldots, x_0 \), a formula similar to Eq. (27) results:

\[ P_n(x) = f[x_n] + f[x_n, x_{n-1}](x - x_n) + f[x_n, x_{n-1}, x_{n-2}](x - x_n)(x - x_{n-1}) + \cdots + f[x_n, \ldots, x_0](x - x_n)(x - x_{n-1}) \cdots (x - x_1). \]  

- If the nodes are equally spaced with \( x = x_n + sh \) and \( x = x_i + (s + n-i)h \), then

\[ P_n(x) = f[x_n] + shf[x_n, x_{n-1}] + s(s + 1)h^2 f[x_n, x_{n-1}, x_{n-2}] + \cdots + s(s + 1) \cdots (s + n - 1)h^n f[x_n, \ldots, x_0]. \]

This form is called the Newton backward divided difference formula.

- It is used to derive a more commonly applied formula known as the Newton backward difference formula.

**Newton Backward-Difference Formula**

\[ P_n(x) = f[x_n] + \sum_{k=1}^{n} (-1)^k \binom{-s}{k} \nabla^k f(x_n) \]

### 4.7 Inverse Interpolation

- Interpolation is the process of determining the value of the dependent variable \( f \) corresponding to a particular value of the independent variable \( x \) when the function \( f(x) \) is described by a set of tabular data.

- Inverse interpolation is the process of determining the value of the independent variable \( x \) corresponding to a particular value of the dependent variable \( f \). In other words, inverse interpolation is evaluation of the inverse function \( x(f) \).

- Inverse interpolation can be accomplished by:
  - Fitting a polynomial to the inverse function \( x(f) \)
  - Solving a direct polynomial \( f(x) \) iteratively for \( x(f) \)

- Fitting a polynomial to the inverse function \( x(f) \) appears to be the obvious approach.

- However, some problems may occur. The inverse function \( x(f) \) may not resemble a polynomial.

- The values of \( f \) most certainly is not equally spaced. In such cases, a direct fit polynomial \( f(x) \) may be preferred, even though it must be solved iteratively for \( x(f) \), for example, by Newton’s method.

- Consider the following set of tabular data, which corresponds to the function \( f(x) = 1/x \):

<table>
<thead>
<tr>
<th>( x )</th>
<th>( f(x) )</th>
<th>( \Delta f(x) )</th>
<th>( \Delta^2 f(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>0.303030</td>
<td>-0.0008912</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>0.294118</td>
<td>-0.008404</td>
<td>0.000508</td>
</tr>
<tr>
<td>3.5</td>
<td>0.285714</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.5 Data in corresponding to \( f(x) \)**

- Let us find the value of \( x \) for which \( f(x) = 0.30 \). The exact solution is \( x = 1/0.3 \), \( f(x) = 1/0.3 = 3.333333 \ldots \).

- Let us evaluate the quadratic Lagrange polynomial, \( x = x(f) \), for \( f = 0.30 \). Thus,
\[ x = \frac{(0.30 - 0.294118)(0.30 - 0.285714)}{(0.303030 - 0.294118)(0.303030 - 0.285714)} \]  
\[ + \frac{(0.30 - 0.294118)(0.294118 - 0.285714)}{(0.303030)(0.294118 - 0.285714)} \]  
\[ + \frac{(0.30 - 0.294118)(0.30 - 0.294118)}{(0.285714 - 0.303030)(0.285714 - 0.294118)} \] 
(3.3)

(3.4)

(3.5)

(43)

• Which yields \( x = 3.333301 \). The error is Error = 3.333301 - 3.333333 = -0.000032.

• Alternatively, let us evaluate the quadratic Newton forward-difference polynomial for \( f = f(x) \). Thus,

\[ f(x) = f_0 + s \Delta f_0 + \frac{s(s - 1)}{2} \Delta^2 f_0 \]  
(44)

Substituting the values from the table into Eq. (44) yields

\[ 0.30 = 0.303030 + s(-0.008912) + \frac{s(s - 1)}{2}(0.000508) \]  
(45)

Simplifying Eq. (45) gives

\[ 0.000508s^2 - 0.018332s + 0.006060 = 0 \]  
(46)

Solving Eq. (46) by the quadratic formula yields two solutions, \( S = 0.333654 \) and 35.752960. The second root is obviously extraneous. Solving for \( x \) from the first root yields

\[ x = x_0 + sh = 3.3 + (0.333654)(0.1) = 3.333365 \]  
(47)

• The error is Error = 3.333365 - 3.333333 = 0.000032.

### 4.8 Errors in Numerical Differentiation

• In numerical differentiation, the error in the higher order derivatives occurs due to the fact that, although the tabulated function and its approximating polynomial would agree at the set of data points, their slopes at these points may vary considerably.

• Numerical differentiation is, therefore, an unsatisfactory process and should be used only in rare cases.

• The numerical computation of derivatives involves two types of errors:
  - Truncation errors
  - Rounding errors

• The truncation error is caused by replacing \( \frac{1}{6h} \left| \frac{\Delta^2 y_{-2} + \Delta^2 y_{-1}}{2} \right| \) by means of an interpolating polynomial.

• The truncation error in the first derivative = \( \frac{1}{12h^2} |\Delta^4 y_{-2}| \)

• The rounding error is proportional to 1/h in the case of the first derivatives, while it is proportional to 1/h^2 in the case of the second derivatives, and so on.

• The maximum rounding error in the first derivative = \( \frac{3 \varepsilon}{2h} \)

• The maximum rounding error in the second derivative = \( \frac{4 \varepsilon}{h^2} \)

• Where \( \varepsilon \) is the maximum error in the value of \( y_i \).

### 4.9 Numerical Integration

• Given a set of tabulated values of the integrand \( f(x) \), determining the value of \( \int_{\alpha}^{\beta} f(x) \, dx \) is called numerical integration.
• The given interval of integration is subdivided into a large number of subintervals of equal width \( h \) and the function tabulated at the points of subdivision is replaced by any one of the interpolating polynomials like Newton-Gregory’s, Stirling’s, Bessel’s over each of the subintervals and the integral is evaluated.

• There are several formulae for numerical integration which we shall derive in the sequel.

4.10 Newton-Cote’s Quadrature Formula

Let

\[ I = \int_{a}^{b} y \, dx \]

Where \( y \) takes the values \( y_0, y_1, y_2, \ldots, y_n \) for \( x = x_0, x_1, x_2, \ldots, x_n \).

Let the interval of integration \( (a, b) \) be divided into \( n \) equal sub-intervals, each of width

\[ h = \frac{b - a}{n} \]

So that,

\[ x_0 = a, x_1 = x_0 + h, x_2 = x_0 + 2h, \ldots, x_n = x_0 + nh = b \]

\[ \therefore \quad I = \int_{x_0}^{x_0 + nh} f(x) \, dx \]

Since any \( x \) is given by \( x = x_0 + r \, h \) and \( dx = h \, dr \)

\[ \therefore \quad I = h \int_{0}^{n} f(x_0 + rh) \, dr \]

\[ = h \sum_{0}^{n} \left[ y_0 + r \Delta y_0 + \frac{r(r-1)}{2!} \Delta^2 y_0 + \frac{r(r-1)(r-2)}{3!} \Delta^3 y_0 + \ldots \right] \, dr \]

by Newton’s forward interpolation formula,

\[ = nh \left[ y_0 + \frac{n}{2} \Delta y_0 + \frac{n(n-3)}{12} \Delta^2 y_0 + \frac{n(n-2)^2}{24} \Delta^3 y_0 + \ldots \right] \quad (48) \]
This is a general quadrature formula and is known as Newton-Cote’s quadrature formula. A number of important deductions viz. Trapezoidal rule, Simpson’s one-third and three-eighth rules, Weddle’s rule can be immediately deduced by putting \( n = 1, 2, 3, \) and \( 6, \) respectively, in formula (49)

### 4.11 Trapezoidal Rule (\( N = 1 \))

- Putting \( n = 1 \) in formula (48) and taking the curve through \( (x_0, y_0) \) and \( (x_1, y_1) \) as a polynomial of degree one so that differences of an order higher than one vanish, we get

\[
\int_{x_0}^{x_0 + h} f(x) \, dx = h \left( \frac{1}{2} \Delta y_0 \right) = \frac{h}{2} (2y_0 + (x_1 - y_0)) = \frac{h}{2} (y_0 + y_1)
\]

- Similarly, for the next sub-interval \( (x_0 + h, x_0 + 2h) \), we get

\[
\int_{x_0 + h}^{x_0 + 2h} f(x) \, dx = \frac{h}{2} (y_1 + y_2), \ldots, \int_{x_0 + (n - 1)h}^{x_0 + nh} f(x) \, dx = \frac{h}{2} (y_{n-1} + y_n)
\]

- Adding the above integrals, we get

\[
\int_{x_0}^{x_0 + nh} f(x) \, dx = \frac{h}{2} \left[ (y_0 + y_n) + 2(y_1 + y_2 + \ldots + y_{n-1}) \right]
\]

Which is known as Trapezoidal rule.

- By increasing the number of subintervals, thereby making \( h \) very small, we can improve the accuracy of the value of the given integral.

### 4.12 Simpson’s One-Third Rule (\( n = 2 \))

- Putting \( n = 2 \) in formula (48) and taking the curve through \( (x_0, y_0), (x_1, y_1) \) and \( (x_2, y_2) \) as a polynomial of degree two so that differences of order higher than two vanish, we get

\[
\int_{x_0}^{x_0 + 2h} f(x) \, dx = 2h \left[ y_0 + \Delta y_0 + \frac{1}{6} \Delta^2 y_0 \right]
\]

\[
= \frac{2h}{6} \left[ 6y_0 + 6(y_1 - y_0) + (y_2 - 2y_1 + y_0) \right]
\]

\[
= \frac{h}{3} (y_0 + 4y_1 + y_2)
\]

- Similarly,

\[
\int_{x_0}^{x_0 + 4h} f(x) \, dx = \frac{h}{3} (y_2 + 4y_3 + y_4), \ldots, \int_{x_0 + (n - 2)h}^{x_0 + nh} f(x) \, dx = \frac{h}{3} (y_{n-2} + 4y_{n-1} + y_n)
\]

- Adding the above integrals, we get

\[
\int_{x_0}^{x_0 + nh} f(x) \, dx = \frac{h}{3} \left[ (y_0 + y_n) + 4(y_1 + y_3 + \ldots + y_{n-1}) + 2(y_2 + y_4 + \ldots + y_{n-2}) \right]
\]

\[
(50)
\]

- This is known as Simpson’s one-third rule.

- While using this formula, the given interval of integration must be divided into an even number of sub-intervals, since we find the area over two sub-intervals at a time.
4.13 Simpson's Three-Eighth Rule \((N = 3)\)

- Putting \(n = 3\) in formula (48) and taking the curve through \(x_0, y_0\), \(x_1, y_1\), \(x_2, y_2\), and \(x_3, y_3\) as a polynomial of degree three so that differences of order higher than three vanish, we get

\[
\int_{x_3}^{x_0 + 3h} f(x) \, dx = \frac{3h}{8} \left[ \frac{3}{2} y_0 + \frac{3}{4} \Delta^2 y_0 + \frac{1}{8} \Delta^3 y_0 \right]
\]

- Similarly,

\[
\int_{x_0 + 6h}^{x_0} f(x) \, dx = \frac{3h}{8} \left[ 3y_0 + 3y_4 + 3y_5 + y_6 \right]. \ldots \]

\[
\int_{x_0 + (n - 3)h}^{x_0 + 6h} f(x) \, dx = \frac{3h}{8} \left[ y_{n-3} + 3y_{n-2} + 3y_{n-1} + y_n \right]
\]

- Adding the above integrals, we get

\[
\int_{x_0}^{x_0 + nh} f(x) \, dx = \frac{3h}{8} \left[ (y_0 + y_n) + 3(y_1 + y_2 + y_4 + y_5) + \ldots + (y_{n-2} + y_{n-1}) + 2(y_3 + y_6 + \ldots + y_{n-3}) \right]
\]

which is known as Simpson's three-eighth rule.

- While using this formula, the given interval of integration must be divided into sub-intervals whose number \(n\) is a multiple of 3.

4.14 Boole's Rule

- Putting \(n = 4\) in formula (48) and neglecting all differences of order higher than four, we get

\[
\int_{x_0}^{x_0 + 4h} f(x) \, dx = h \int_0^4 \left[ y_0 + r \Delta y_0 + \frac{r(r-1)}{2!} \Delta^2 y_0 + \frac{r(r-1)(r-2)}{3!} \Delta^3 y_0 + \ldots \right] \, dr
\]

By Newton's forward interpolation formula

\[
= 4h \left[ y_0 + \frac{n}{2} \Delta y_0 + \frac{n(2n-3)}{12} \Delta^2 y_0 + \frac{n(n-2)^2}{24} \Delta^3 y_0 \right.
\]

\[
+ \left. \left( \frac{n^4}{5} - \frac{3n^3}{2} + \frac{11n^2}{3} - 3n \right) \Delta^4 y_0 \right]_0^7
\]

\[
= 4h \left[ y_0 + 2\Delta y_0 + \frac{5}{3} \Delta^2 y_0 + \frac{3}{2} \Delta^3 y_0 + \frac{7}{90} \Delta^4 y_0 \right]
\]

\[
= \frac{2h}{45} (7y_0 + 32y_1 + 12y_2 + 32y_3 + 7y_4)
\]

Similarly,

\[
\int_{x_0 + 4h}^{x_0 + 8h} f(x) \, dx = \frac{2h}{45} (7y_4 + 32y_5 + 12y_6 + 32y_7 + 7y_8)
\]

- Adding all these integrals from \(x_0\) to \(x_0 + nh\), where \(n\) is a multiple of 4, we get
\[
\int_{x_0}^{x_0 + nh} f(x) \, dx = \frac{2h}{45} \left[ 7y_0 + 32y_1 + 12y_2 + 32y_3 + 14y_4 + 32y_5 + 12y_6 + 32y_7 + 14y_8 + \ldots \right]
\]

(52)

- This is known as Boole’s rule.
- While applying Boole’s rule, the number of sub-intervals should be taken as a multiple of 4.

### 4.15 Gaussian Quadrature Formula

- Consider the numerical evaluation of the integral
  \[
  \int_a^b f(x) \, dx
  \]
  (53)

- So far, we studied some integration formulae which require values of the function at equally spaced points of the interval.
- Gauss derived a formula which uses the same number of function values but with different spacing and gives better accuracy.
- Gauss’s formula is expressed in the form
  \[
  \int_a^b F(u) \, du = W_1 F(u_1) + W_2 F(u_2) + \ldots + W_n F(u_n)
  \]
  (54)

  - Where Wi and ui are called the weights and abscissae respectively.
  - The formula has an advantage that the abscissae and weights are symmetrical with respect to the middle point of the interval.
  - In equation (54), there are altogether 2n arbitrary parameters and therefore the weights and abscissae can be determined so that the formula is exact when \( F(u) \) is a polynomial of degree not exceeding \( 2n - 1 \).
- Hence, we start with
  \[
  F(u) = C_0 + C_1 u + C_2 u^2 + C_3 u^3 + \ldots + C_{2n-1} u^{2n-1}
  \]
  (55)

- Then from (54),
  \[
  \int_{-1}^{1} F(u) \, du = \int_{-1}^{1} (C_0 + C_1 u + C_2 u^2 + C_3 u^3 + \ldots + C_{2n-1} u^{2n-1}) \, du
  \]
  \[
  = 2 C_0 + \frac{2}{3} C_2 + \frac{2}{5} C_4 + \ldots
  \]
  (56)

- Set \( u = u_i \) in (55), we get
  \[
  F(u_i) = C_0 + C_1 u_i + C_2 u_i^2 + C_3 u_i^3 + \ldots + C_{2n-1} u_i^{2n-1}
  \]

- From (54),
  \[
  \int_{-1}^{1} F(u) \, du = W_1 (C_0 + C_1 u_1 + C_2 u_1^2 + \ldots + C_{2n-1} u_1^{2n-1}) + \ldots
  \]
  \[
  + W_2 (C_0 + C_1 u_2 + C_2 u_2^2 + \ldots + C_{2n-1} u_2^{2n-1}) + \ldots
  \]
  \[
  + W_n (C_0 + C_1 u_n + C_2 u_n^2 + \ldots + C_{2n-1} u_n^{2n-1}) + \ldots
  \]
  (57)

- Which can be written as
  \[
  \int_{-1}^{1} F(u) \, du = C_0 (W_1 + W_2 + \ldots + W_n) + C_1 (W_1 u_1 + W_2 u_2 + \ldots + W_n u_n) + C_2 (W_1 u_1^2 + W_2 u_2^2 + \ldots + W_n u_n^2) + \ldots
  \]
  \[
  + C_{2n-1} (W_1 u_1^{2n-1} + W_2 u_2^{2n-1} + \ldots + W_n u_n^{2n-1}) + \ldots
  \]
  (58)

- Now equations (57) and (58) are identical for all values of \( C_i \) and hence comparing the coefficients of \( C_i \), we obtain 2n equations.
In 2n unknowns $W_i$ and $u_i$ $(i = 1, 2, \ldots, n)$.

The abscissae $u_i$ and the weights $W_i$ are extensively tabulated for different values of $n$.

The table up to $n = 5$ is given below:

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\pm u_i$</th>
<th>$W_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.57735, 02692</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.88888 88889</td>
</tr>
<tr>
<td>3</td>
<td>0.77459 66692</td>
<td>0.55555 55556</td>
</tr>
<tr>
<td></td>
<td>0.33998 10436</td>
<td>0.65214 51549</td>
</tr>
<tr>
<td></td>
<td>0.86113 63116</td>
<td>0.34785 48451</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.56888 88889</td>
</tr>
<tr>
<td>4</td>
<td>0.77459 66692</td>
<td>0.55555 55556</td>
</tr>
<tr>
<td></td>
<td>0.33998 10436</td>
<td>0.65214 51549</td>
</tr>
<tr>
<td></td>
<td>0.86113 63116</td>
<td>0.34785 48451</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.56888 88889</td>
</tr>
<tr>
<td>5</td>
<td>0.53846 93101</td>
<td>0.47862 86705</td>
</tr>
<tr>
<td></td>
<td>0.90617 98459</td>
<td>0.23692 68851</td>
</tr>
</tbody>
</table>

Table 4.6 Table for n=5

In general case, the limits of integral in (56) have to be changed to those in (54) by transformation.

\[ x = \frac{1}{2} u (b - a) + \frac{1}{2} (a + b) \]  

(60)

### 4.16 Numerical Evaluation Of Singular Integrals

- The various numerical integration formulae we have discussed so far are valid, if integrand $f(x)$ can be expanded by a polynomial or, alternatively can be expanded in a Taylor’s series in the interval $[a, b]$.

- In a case where function has a singularity, the preceding formulae cannot be applied and special methods will have to be adopted.

- Consider,

\[ I(f) = \int_{a}^{b} \frac{f(x)}{x - t} \, dx \]  

Which is singular at $t = x$

- Its Principal value,

\[ P(I) = \lim_{\varepsilon \to 0} \left[ \int_{a}^{t-\varepsilon} \frac{f(x)}{x - t} \, dx + \int_{t+\varepsilon}^{b} \frac{f(x)}{x - t} \, dx \right] ; a < t < b \]  

\[ = I(f) \text{ (for } t < a \text{ or } t > b) \]  

(62)

- Set $x = a + uh$ and $t = a + kh$ in (1), we get

\[ P(I) = \sum_{u} f(a + hu) \frac{f(a + hu)}{u - k} \, du \]  

(63)

- Replacing $f(a + hu)$ by Newton’s forward difference formula at $x = a$ and simplifying, we get

\[ I(f) = \sum_{j=a}^{j} \frac{\Delta^j f(a)}{j!} C_j \]  

(64)
Where the constants $C_j$ are given by
\[ C_j = P \int_0^p \frac{(u)_j}{u - k} \, du \tag{65} \]

In (65), $(u)_0 = 1$, $(u)_1 = u$, $(u)_2 = u (u - 1)$ etc.

Various approximate formulae can be obtained by truncating the series on R.H.S. of (64).

Eqn. (64) may be written as
\[ I_n(f) = \sum_{j=0}^{n} \frac{\Delta^j f(a)}{j!} C_j \tag{66} \]

We obtain rules of orders 1, 2, 3, .... etc. by setting $n = 1, 2, 3, \ldots$ respectively.

Two point rule ($n = 1$):
\[ I_1(f) = \sum_{j=0}^{1} \frac{\Delta^j f(a)}{j!} C_j \tag{67} \]
\[ = C_0 f(a) + C_1 \Delta f(a) \]
\[ = (C_0 - C_1) f(a) + C_1 f(a + h) \tag{68} \]

Three-point rule ($n = 2$):
\[ I_2(f) = \sum_{j=0}^{2} \frac{\Delta^j f(a)}{j!} C_j = C_0 f(a) + C_1 \Delta f(a) + C_2 \Delta^2 f(a) \]
\[ = \left( C_0 - C_1 + \frac{1}{2} C_2 \right) f(a) + (C_1 - C_2) f(a + h) \]
\[ + \frac{1}{2} C_2 f(a + 2h) \tag{69} \]

In above relations (68) and (69), values of $C_j$ are given by,
\[ C_0 = \log_e \left| \frac{p - k}{k} \right| \]
\[ C_1 = p + C_0 k \]
\[ C_2 = \frac{1}{2} p^2 + p (k - 1) + C_0 k (k - 1) \tag{70} \]
A census of the population of the United States is taken every 10 years.

The following table lists the population, in thousands of people, from 1940 to 1990.

One of the most useful and well-known classes of functions mapping the set of real numbers into itself is the class of algebraic polynomials, the set of functions of the form

\[ P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0, \]

Suppose that \( f \) is defined and continuous on \([a, b]\). For each \( \varepsilon > 0 \), there exists a polynomial \( P(x) \), with the property that

\[ |f(x) - P(x)| < \varepsilon, \text{ for all } x \in [a, b] \]

The proof of this theorem can be found in most elementary texts on real analysis.

Another important reason for considering the class of polynomials in the approximation of functions is that the derivative and indefinite integral of a polynomial are easy to determine and are also polynomials.

In reviewing these data, we might ask whether they could be used to provide a reasonable estimate of the population, say, in 1965 or even in the year 2010.

Predictions of this type can be obtained by using a function that fits the given data. This process is called interpolation.

Another important reason for considering the class of polynomials in the approximation of functions is that the derivative and indefinite integral of a polynomial are easy to determine and are also polynomials.

For these reasons, polynomials are often used for approximating continuous functions.

The Taylor polynomials were introduced in the first section of the book, where they were described as one of the fundamental building blocks of numerical analysis.

For ordinary computational purposes it is more efficient to use methods that include information at various points, which we consider in the remainder of this chapter.

The primary use of Taylor polynomials in numerical analysis is not for approximation purposes but for the derivation of numerical techniques and error estimation.

Since the Taylor polynomials are not appropriate for interpolation, alternative methods are needed.

In this section we find approximating polynomials that are determined simply by specifying certain points on the plane through which they must pass.

Our treatment of divided-difference methods will be brief since the results in this section will not be used extensively in subsequent material.

Older texts on numerical analysis have extensive treatments of divided-difference methods.

Interpolation is the process of determining the value of the dependent variable \( f \) corresponding to a particular value of the independent variable \( x \) when the function \( f(x) \) is described by a set of tabular data.

Inverse interpolation is the process of determining the value of the independent variable \( x \) corresponding to a particular value of the dependent variable \( f \). In other words, inverse interpolation is evaluation of the inverse function \( x(f) \).

Inverse interpolation can be accomplished by:

- Fitting a polynomial to the inverse function \( x(f) \)
- Solving a direct polynomial \( f(x) \) iteratively for \( x(f) \)
References


Recommended Reading

Self Assessment

1. The _________ polynomials were introduced in the first section of the book, where they were described as one of the fundamental building blocks of numerical analysis.
   a. Taylor
   b. Shimon
   c. Euler
   d. Simon

2. A good interpolation polynomial needs to provide a relatively ______ approximation over an entire interval.
   a. accurate
   b. perfect
   c. correct
   d. precise

3. Taylor polynomials have the property that all the _________ used in the approximation is concentrated at the single point $X_0$.
   a. sequence
   b. data
   c. information
   d. values

4. The Taylor polynomials are not __________ for interpolation.
   a. inappropriate
   b. appropriate
   c. precise
   d. wrong

5. Numerical differentiation is, therefore, an __________ process and should be used only in rare cases.
   a. unacceptable
   b. substandard
   c. unsatisfactory
   d. disappointing

6. The numerical computation of __________ involves two types of errors
   a. derivatives
   b. unoriginals
   c. imitatives
   d. copies
7. Gauss derived a formula which uses the same number of function values but with different spacing and gives
__________
   a. correction
   b. better accuracy.
   c. system
   d. truth.

8. The Gaussian quadrature formula has an advantage that the abscissae and weights are symmetrical with respect
to the ____________ point of the interval.
   a. middle
   b. last
   c. first
   d. intermediate

9. In Gaussian’s formulae Wi and ui are called the ____________ respectively.
   a. weights and abscissa
   b. mass and abscissa
   c. quantity and abscissa
   d. height and abscissa

10. The truncation error is caused by ________ the tabulated function by means of an interpolating polynomial.
    a. restoring
    b. returning
    c. replacing
    d. swapping
Chapter V
Numerical Solution of Ordinary Differential Equations

Aim
The aim of this chapter is to:

• state the concept of numerical solution of differential equations
• explain initial value and boundary-value problem
• discuss single step and multi-step methods

Objectives
The objectives of this chapter are to:

• compare single-step and multi-step methods
• discuss the Picard’s method of successive approximations
• solve equation on Picard’s methods
• explain Euler’s method

Learning outcome
At the end of this chapter, the students will be able to:

• elaborate Euler’s graph
• explain Runge-Kutta Methods
• explain the order Runge-Kutta method
• discuss second order Runge-Kutta method
5.1 Introduction
A physical situation concerned with the rate of change of one quantity with respect to another gives rise to a differential equation.

- Consider the first order ordinary differential equation.
  \[ \frac{dy}{dx} = f(x, y) \]  
  With the initial condition,
  \[ y(x_0) = y_0 \]  
- Many analytical techniques exist for solving such equations, but these methods can be applied to solve only a selected class of differential equations.
- However, a majority of differential equations appearing in physical problems cannot be solved analytically.
- Thus, it becomes imperative to discuss their solution by numerical methods.
- In numerical methods, we do not proceed in the hope of finding a relation between variables, but we find the numerical values of the dependent variable for certain values of independent variable.
- It must be noted that even the differential equations which are solvable by analytical methods can be solved numerically as well.

5.2 Initial-Value and Boundary-Value Problems
Problems in which all the conditions are specified at the initial point only are called initial-value problems. For example, the problem given by eqns. (1) and (2) is an initial value problem. Problems involving second and higher order differential equations, in which the conditions at two or more points are specified, are called boundary-value problems. To obtain a unique solution of nth order ordinary differential equation, it is necessary to specify n values of the dependent variable and/or its derivative at specific values of independent variable.

5.3 Single Step and Multi-Step Methods
- The numerical solutions are obtained step-by-step through a series of equal intervals in the independent variable so that as soon as the solution y has been obtained at \( x = x_i \), the next step consists of evaluating \( y_{i+1} \) at \( x = x_{i+1} \).
- The methods which require only the numerical value \( y_i \) in order to compute the next value \( y_{i+1} \) for solving eqn. (1) given above are termed as single step methods.
- The methods which require not only the numerical value \( y_i \) but also at least one of the past values \( y_{i-1}, y_{i-2}, \ldots \) are termed as multi-step methods.

5.4 Comparison of Single-Step And Multi-Step Methods
- The single step method has obvious advantages over the multi-step methods that use several past values \( y_{i}, y_{i-1}, \ldots, y_{i-p} \) and that require initial values \( y_1, y_2, \ldots, y_n \) that have to be calculated by another method.
- The major disadvantage of single-step methods is that they use many more evaluations of the derivative to attain the same degree of accuracy compared with the multi-step methods.

5.5 Numerical Methods of Solution of O.D.E.
In this chapter, we will discuss various numerical methods of solving ordinary differential equations. We know that these methods will yield the solution in one of the two forms:
  a) A series for y in terms of powers of x from which the value of y can be obtained by direct substitution.
  b) A set of tabulated values of x and y.
- Picard’s method and Taylor’s method belong to class (a) while those of Euler’s, Runge-Kutta, Adams-Bashforth, Milne’s, etc. belong to class (b).
- Methods which belong to class (b) are called step-by-step methods or marching methods because the values of y are computed by short steps ahead for equal intervals of the independent variable.
- In Euler’s and Runge-Kutta methods, the interval range \( h \) should be kept small, hence they can be applied for tabulating y only over a limited range.
To get functional values over a wider range, the Adams-Bashforth, Milne, Adams-Moulton, etc. methods may be used since they use finite differences and require starting values, usually obtained by Taylor’s series or Runge-Kutta methods.

### 5.6 Picard’s Method Of Successive Approximations

Picard was a distinguished Professor of Mathematics at the University of Paris, France.

- He was famous for his research on the Theory of Functions. Consider the differential equation:
  \[
  \frac{dy}{dx} = f(x, y); \quad y(x_0) = y_0
  \]  \hspace{1cm} (3)

- Integrating eqn. (3) between the limits \(x_0\) and \(x\) and the corresponding limits \(y_0\) and \(y\), we get:
  \[
  \int_{x_0}^{x} dy = \int_{x_0}^{x} f(x, y) \, dx
  \]

\[
  y - y_0 = \int_{x_0}^{x} f(x, y) \, dx
  \]

\[
  y = y_0 + \int_{x_0}^{x} f(x, y) \, dx
  \]  \hspace{1cm} (4)

- In equation (4), the unknown function \(y\) appears under the integral sign.

- This type of equation is called integral equation.

- This equation can be solved by the method of successive approximations or iterations.

- To obtain the first approximation, we replace \(y\) by \(y_0\) in the R.H.S. of eqn. (4).

- Now, the first approximation is
  \[
  y^{(1)} = y_0 + \int_{x_0}^{x} f(x, y_0) \, dx
  \]

- The integrand is a function of \(x\) alone and can be integrated.

- For a second approximation, replace \(y_0\) by \(y^{(1)}\) in \(f(x, y_0)\) which gives
  \[
  y^{(2)} = y_0 + \int_{x_0}^{x} f(x, y^{(1)}) \, dx
  \]

- Proceeding in this way, we obtain \(y^{(3)}\), \(y^{(4)}\), ........, \(y^{(n-1)}\) and \(y^{(n)}\) where
  \[
  y^{(n)} = y_0 + \int_{x_0}^{x} f(x, y^{(n-1)}) \, dx \quad \text{with} \quad y(x_0) = y_0
  \]

- As a matter of fact, the process is stopped when the two values of \(y\) viz. \(y^{(n-1)}\) and \(y^{(n)}\) are the same to the desired degree of accuracy.

- Picard’s method is of considerable theoretical value.

- Practically, it is unsatisfactory because of the difficulties which arise in performing the necessary integrations.

- However, each step gives a better approximation of the required solution than the preceding one.

#### Example

Given the differential eqn.

\[
\frac{dy}{dx} = \frac{x^2}{y^2 + 1}
\]

with the initial condition \(y = 0\) when \(x = 0\). Use Picard’s method to obtain \(y\) for \(x = 0.25, 0.5\) and \(1.0\) correct to three decimal places.

#### Solution

The given initial value problem is
\[ \frac{dy}{dx} = f(x, y) = \frac{x^2}{y^2 + 1} \]

Where \( y = y_0 = 0 \) at \( x = x_0 = 0 \)

We have first approximation,

\[ y^{(1)} = y_0 + \int_{x_0}^{x} f(x, y_0) \, dx \]

\[ = 0 + \int_{0}^{x} \frac{x^2}{0 + 1} \, dx = \frac{1}{3} x^3 \]

\[ \text{(5)} \]

Second approximation,

\[ y^{(2)} = y_0 + \int_{x_0}^{x} f(x, y^{(1)}) \, dx \]

\[ = 0 + \int_{0}^{x} \frac{x^2}{\left( \frac{x^3}{3} \right)^2 + 1} \, dx \]

\[ = \left[ \tan^{-1} \frac{x^3}{3} \right]_0^x = \tan^{-1} \frac{x^3}{3} \]

\[ = \frac{1}{3} x^3 - \frac{1}{3} \left( \frac{1}{3} \frac{1}{3} x^3 \right)^3 + \ldots \]

\[ = \frac{1}{3} x^3 - \frac{1}{81} x^9 + \ldots \]

\[ \text{(6)} \]

From (5) and (6), we see that \( y^{(1)} \) and \( y^{(2)} \) agree to the first term \( x^3 \). To find the range of values of \( x \) so that the series with the term \( \frac{1}{3} x^3 \) alone will give the result correct to three decimal places, we put

\[ \frac{1}{81} x^9 \leq .0005 \]

\[ \text{(7)} \]

Which gives:

\[ x^9 \leq .0405 \quad \text{or} \quad x \leq 0.7 \]

Hence,

\[ y(.25) = \frac{1}{3} (.25)^3 = .005 \]

and

\[ y(0.5) = \frac{1}{3} (0.5)^3 = .042 \]

To find \( y(1.0) \), we make use of eqn. (6) which gives,
Example
Use Picard’s method to obtain $y$ for $x = 0.2$. Given:
\[ \frac{dy}{dx} = x - y \]
with initial condition $y = 1$ when $x = 0$.

Solution
Here $f(x, y) = x - y$, $x_0 = 0$, $y_0 = 1$
We have first approximation,
\[ y^{(1)} = y_0 + \int_0^x f(x, y_0) \, dx = 1 + \int_0^x (x - y_0) \, dx = 1 - x + \frac{x^2}{2} \]

Second approximation,
\[ y^{(2)} = y_0 + \int_0^x f(x, y^{(1)}) \, dx = 1 + \int_0^x (x - y^{(1)}) \, dx \\
= 1 + \int_0^x \left( x - 1 + x - \frac{x^2}{2} \right) \, dx = 1 - x + x^2 - \frac{x^3}{6} \]

Third approximation,
\[ y^{(3)} = y_0 + \int_0^x f(x, y^{(2)}) \, dx = 1 + \int_0^x (x - y^{(2)}) \, dx \\
= 1 + \int_0^x \left( x - 1 + x - x^2 + \frac{x^3}{6} \right) \, dx \\
= 1 - x + x^2 - \frac{x^3}{3} + \frac{x^4}{24} \]

Fourth approximation,
\[ y^{(4)} = y_0 + \int_0^x f(x, y^{(3)}) \, dx = 1 + \int_0^x (x - y^{(3)}) \, dx \\
= 1 + \int_0^x \left( x - 1 + x - x^2 + \frac{x^3}{3} - \frac{x^4}{24} \right) \, dx \\
= 1 - x + x^2 - \frac{x^3}{3} + \frac{x^4}{12} - \frac{x^5}{120} \]

Fifth approximation,
\[ y^{(5)} = y_0 + \int_0^x f(x, y^{(4)}) \, dx = 1 + \int_0^x (x - y^{(4)}) \, dx \\
= 1 + \int_0^x \left( x - 1 + x - x^2 + \frac{x^3}{3} - \frac{x^4}{12} + \frac{x^5}{120} \right) \, dx \\
= 1 - x + x^2 - \frac{x^3}{3} + \frac{x^4}{12} - \frac{x^5}{60} + \frac{x^6}{720} \]

When $x = 0.2$, we get
Thus, \( y = 0.837 \) when \( x = 0.2 \).

**Example**

Use Picard’s method to obtain \( y \) for \( x = 0.1 \). Given that:

\[
\frac{dy}{dx} = 3x + y^2; \ y = 1 \ at \ x = 0.
\]

**Solution**

Here,

\[
f(x, y) = 3x + y^2, \ x_0 = 0, \ y_0 = 1
\]

First approximation,

\[
y^{(1)} = y_0 + \int_0^x f(x, y_0) \, dx
\]

\[
= 1 + \int_0^x (3x + 1) \, dx
\]

\[
= 1 + x + \frac{3}{2} x^2
\]

Second approximation,

\[
y^{(2)} = 1 + x + \frac{5}{2} x^2 + \frac{4}{8} x^3 + \frac{3}{4} x^4 + \frac{9}{20} x^5
\]

Third approximation,

\[
y^{(3)} = 1 + x + \frac{5}{2} x^2 + 2x^3 + \frac{23}{12} x^4 + \frac{25}{12} x^5
\]

\[
\quad + \frac{68}{45} x^6 + \frac{1157}{1260} x^7 + \frac{17}{32} x^8 + \frac{47}{240} x^9
\]

\[
\quad + \frac{27}{400} x^{10} + \frac{81}{4400} x^{11}
\]

When \( x = 0.1 \), we have

\[
y^{(1)} = 1.115, \quad y^{(2)} = 1.1264, \quad y^{(3)} = 1.12721
\]

Thus,

\[
y = 1.127
\]

When \( x = 0.1 \).
5.7 Picard’s Method for Simultaneous First Order Differential Equations

Let \( \frac{dy}{dx} = \phi(x, y, z) \) and \( \frac{dz}{dx} = f(x, y, z) \)

Be the simultaneous differential eqns. with initial conditions \( y(x_0) = y_0; z(x_0) = z_0 \).

Picard’s method gives

\[
\begin{align*}
  y^{(1)} &= y_0 + \int_{x_0}^{x} \phi(x, y, z) \, dx; \\
  z^{(1)} &= z_0 + \int_{x_0}^{x} f(x, y, z) \, dx \\
  y^{(2)} &= y_0 + \int_{x_0}^{x} \phi(x, y^{(1)}, z^{(1)}) \, dx; \\
  z^{(2)} &= z_0 + \int_{x_0}^{x} f(x, y^{(1)}, z^{(1)}) \, dx
\end{align*}
\]

And so on as successive approximations.

Example

Approximate \( y \) and \( z \) by using Picard’s method for the particular solution of \( \frac{dy}{dx} = x + z, \quad \frac{dz}{dx} = x - y^2 \) given that \( y = 2, \ z = 1 \) when \( x = 0 \).

Solution

Let

\[
\begin{align*}
  \frac{dy}{dx} &= \phi(x, y, z) \Rightarrow y = y_0 + \int_{x_0}^{x} \phi(x, y, z) \, dx \\
  \frac{dz}{dx} &= f(x, y, z) \Rightarrow z = z_0 + \int_{x_0}^{x} f(x, y, z) \, dx
\end{align*}
\]

Here, \( x_0 = 0, \ y_0 = 2, \ z_0 = 1 \)

We have,

\[
\begin{align*}
  \frac{dy}{dx} &= \phi(x, y, z) \Rightarrow y = y_0 + \int_{x_0}^{x} \phi(x, y, z) \, dx \\
  \frac{dz}{dx} &= f(x, y, z) \Rightarrow z = z_0 + \int_{x_0}^{x} f(x, y, z) \, dx
\end{align*}
\]

Also,

\[
\begin{align*}
  \frac{dy}{dx} &= \phi(x, y, z) \Rightarrow y = y_0 + \int_{x_0}^{x} \phi(x, y, z) \, dx \\
  \frac{dz}{dx} &= f(x, y, z) \Rightarrow z = z_0 + \int_{x_0}^{x} f(x, y, z) \, dx
\end{align*}
\]

First approximation,

\[
\begin{align*}
  y^{(1)} &= y_0 + \int_{x_0}^{x} \phi(x, y, z) \, dx = 2 + \int_{0}^{x} (x + z_0) \, dx \\
  &= 2 + \int_{0}^{x} (x + 1) \, dx = 2 + x + \frac{x^2}{2}
\end{align*}
\]

And

\[
\begin{align*}
  z^{(1)} &= z_0 + \int_{x_0}^{x} f(x, y, z) \, dx = 1 + \int_{0}^{x} (x - y_0^2) \, dx \\
  &= 1 + \int_{0}^{x} (x - 4) \, dx = 1 - 4x + \frac{x^2}{2}
\end{align*}
\]
Second approximation,

\[ y^{(2)} = y_0 + \int_{x_0}^{x} \phi(x, y^{(1)}, z^{(1)}) \, dx \]

\[ = 2 + \int_{x_0}^{x} \left( x + z^{(1)} \right) \, dx \]

\[ = 2 + \int_{0}^{x} \left( x + 1 - 4x + \frac{x^2}{2} \right) \, dx \]

\[ = 2 + x - \frac{3}{2} x^2 + \frac{x^3}{6} \]

\[ z^{(2)} = z_0 + \int_{x_0}^{x} f(x, y^{(1)}, z^{(1)}) \, dx \]

\[ = 1 + \int_{0}^{x} \left[ x - \left( 2 + x + \frac{x^2}{2} \right) \right] \, dx \]

\[ = 1 - 4x - \frac{3}{2} x^2 - x^3 - \frac{x^4}{4} - \frac{x^5}{20}. \]

**Example**

Solve by Picard’s method, the differential equations

\[ \frac{dy}{dx} = z, \quad \frac{dz}{dx} = x^3 (y + z) \]

Where

\[ y = 1, \quad z = \frac{t}{2} \alpha t \quad x = 0 \]

Obtain the values of y and z from III approximation when x = 0.2 and x = 0.5.

**Solution**

Let

\[ \phi(x, y, z) = z, \quad f(x, y, z) = x^3 (y + z) \]

Here,

\[ x_0 = 0, \quad y_0 = 1, \quad z_0 = \frac{1}{2} \]

First approximation,

\[ y^{(1)} = y_0 + \int_{x_0}^{x} \phi(x, y_0, z_0) \, dx = 1 + \int_{0}^{x} z_0 \, dx \]

\[ = 1 + \frac{1}{2} x \]

\[ z^{(1)} = z_0 + \int_{x_0}^{x} f(x, y_0, z_0) \, dx = \frac{1}{2} + \int_{0}^{x} x^3 (y_0 + z_0) \, dx \]

\[ = \frac{1}{2} + \frac{3}{2} \frac{x^4}{4}. \]
Second approximation,
\[ y^{(2)} = 1 + \int_0^x z^{(1)} \, dx = 1 + \int_0^x \left( \frac{1}{2} + \frac{x^3}{40} \right) \, dx 
= 1 + \frac{x}{2} + \frac{3}{40} x^6 \]
\[ z^{(2)} = \frac{1}{2} + \int_0^x x^3 (y^{(1)} + z^{(1)}) \, dx \]
\[ = \frac{1}{2} + \int_0^x x^3 \left( \frac{1}{2} + \frac{3}{8} x^4 + \frac{x^6}{10} + \frac{3}{64} x^3 \right) \, dx \]
\[ = \frac{1}{2} + \frac{3}{8} x^4 + \frac{x^6}{10} + \frac{3}{64} x^3 \]

Third approximation,
\[ y^{(3)} = 1 + \int_0^x z^{(2)} \, dx = 1 + \int_0^x \left( \frac{1}{2} + \frac{x^3}{40} + \frac{x^6}{60} + \frac{3x^9}{576} \right) \, dx \]
\[ = 1 + \frac{x}{2} + \frac{3}{40} x^6 + \frac{x^6}{60} + \frac{3x^9}{576} \]
\[ z^{(3)} = \frac{1}{2} + \int_0^x x^3 (y^{(2)} + z^{(2)}) \, dx \]
\[ = \frac{1}{2} + \int_0^x x^3 \left( \frac{1}{2} + \frac{3}{8} x^4 + \frac{x^6}{10} + \frac{3}{64} x^3 + \frac{7}{40} x^5 + \frac{3}{64} x^3 \right) \, dx \]
\[ = \frac{1}{2} + \frac{3}{8} x^4 + \frac{x^6}{10} + \frac{3}{64} x^3 + \frac{7}{360} x^5 + \frac{3}{768} x^{12} \]

When \( x = 0.2 \)
\[ y^{(0)} = 1 + 0.1 + \frac{3}{40} (0.2)^5 + \frac{(0.2)^5}{60} + \frac{3}{576} (0.2)^9 \]
\[ = 1.100024 \text{ (leaving higher terms)} \]
\[ z^{(0)} = \frac{1}{2} + \frac{3}{8} (0.2)^4 + \frac{(0.2)^5}{10} + \frac{3}{64} (0.2)^9 + \frac{7}{360} (0.2)^5 + \frac{3}{768} (0.2)^{12} \]
\[ = 0.500632 \text{ (leaving higher terms)} \]

When \( x = 0.5 \)
\[ y^{(0)} = 1 + \frac{5}{2} + \frac{8}{40} (.5)^6 + \frac{(.5)^6}{60} + \frac{3}{576} (.5)^{10} \]
\[ = 1.25234375 \]
\[ z^{(0)} = \frac{1}{2} + \frac{3}{8} (.5)^4 + \frac{(.5)^5}{10} + \frac{3}{64} (.5)^9 + \frac{7}{360} (.5)^5 + \frac{3}{768} (.5)^{12} \]
\[ = 0.5234375. \]

### 5.8 Euler’s Method

Euler’s method is the simplest one-step method and has a limited application because of its low accuracy. This method yields solution of an ordinary differential equation in the form of a set of tabulated values.

- In this method, we determine the change \( \Delta y \) is \( y \) corresponding to small increase in the argument \( x \).
- Consider the differential equation.
\[ \frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0 \]

- Let \( y = g(x) \) be the solution of (7). Let \( x_0, x_1, x_2, \ldots \) be equidistant values of \( x \).
- In this method, we use the property that in a small interval, a curve is nearly a straight line.
- Thus, at the point \((x_0, y_0)\), we approximate the curve by the tangent at the point \((x_0, y_0)\).

**Fig. 5.1 Euler’s method**

- The eqn. of the tangent at \( P_0(x_0, y_0) \) is
  \[
  y - y_0 = \left( \frac{dy}{dx} \right)_{P_0} (x - x_0) = f(x_0, y_0) (x - x_0)
  \]
  \[
  y = y_0 + (x - x_0) f(x_0, y_0)
  \]

  (8)

  This gives the \( y \)-coordinate of any point on the tangent.
- Since the curve is approximated by the tangent in the interval \((x_0, x_1)\), the value of \( y \) on the curve corresponding to \( x = x_1 \) is given by the above value of \( y \) in eqn. (8) approximately.
- Putting \( x = x_1 = x_0 + h \) in eqn. (8), we get
  \[
  y_1 = y_0 + hf(x_0, y_0)
  \]
  Thus \( Q_1 \) is \((x_1, y_1)\)
- Similarly, approximating the curve in the next interval \((x_1, x_2)\) by a line through \( Q_1(x_1, y_1) \) with slope \( f(x_1, y_1) \), we get
  \[
  y_2 = y_1 + hf(x_1, y_1)
  \]
  In general, it can be shown that,
  \[
  y_{n+1} = y_n + hf(x_n, y_n)
  \]
  This is called Euler’s Formula.
- A great disadvantage of this method lies in the fact that if \( \frac{dy}{dx} \) changes rapidly over an interval, its value at the beginning of the interval may give a poor approximation as compared to its average value over the interval and thus the value of \( y \) calculated from Euler’s method may be in much error from its true value.
- These errors accumulate in the succeeding intervals and the value of \( y \) becomes erroneous.
- In Euler’s method, the curve of the actual solution \( y = g(x) \) is approximated by a sequence of short lines.
  - The process is very slow.
  - If \( h \) is not properly chosen, the curve \( P_0Q_1Q_2 \ldots \) of short lines representing numerical solution deviate significantly from the curve of actual solution.
To avoid this error, Euler’s modified method is preferred because in this, we consider the curvature of the actual curve inplace of approximating the curve by sequence of short lines.

### 5.9 Modified Euler’s Method

The modified Euler’s method gives greater improvement in accuracy over the original Euler’s method.

- Here, the core idea is that we use a line through \((x_0, y_0)\) whose slope is the average of the slopes at \((x_0, y_0)\) and \((x_1, y_1^{(1)})\) where \(y_1^{(1)} = y_0 + hf(x_0, y_0)\).
- This line approximates the curve in the interval \((x_0, x_1)\).
- Geometrically, if \(L_1\) is the tangent at \((x_0, y_0)\), \(L_2\) is a line through \((x_1, y_1^{(1)})\) of slope \(f(x_1, y_1^{(1)})\) and \(\overline{\overline{L}}\) is the line through \((x_1, y_1^{(1)})\) but with a slope equal to the average of \(f(x_0, y_0)\) and \(f(x_1, y_1^{(1)})\) then the line \(L\) through \((x_0, y_0)\) and parallel to \(\overline{\overline{L}}\) is used to approximate the curve in the interval \((x_0, x_1)\).
- Thus the ordinate of the point B will give the value of \(y_1\). Now, the eqn. of the line \(AL\) is given by

\[
y_1 = y_0 + h \left[ \frac{f(x_0, y_0) + f(x_1, y_1^{(1)})}{2} \right]
\]

![Fig. 5.2 Modified Euler’s method](image)

- A generalised form of Euler’s modified formula is

\[
y_1^{(n+1)} = y_0 + \frac{h}{2} \left[ f(x_0, y_0) + f(x_1, y_1^{(n)}) \right] \quad n = 0, 1, 2, \ldots
\]

- Where \(y_1^{(n)}\) is the nth approximation to \(y_1\). The above iteration formula can be started by choosing \(y_1^{(1)}\) from Euler’s formula

\[
y_1^{(1)} = y_0 + hf(x_0, y_0)
\]

- Since this formula attempts to correct the values of \(y_{n+1}\) using the predicted value of \(y_{n+1}\) (by Euler’s method), it is classified as a one-step predictor-corrector method.

**Example**

Given

\[
\frac{dy}{dx} = \frac{y-x}{y+x} \quad \text{with } y = 1 \text{ for } x = 0.
\]

Find \(y\) approximately for \(x = 0.1\) by Euler’s method.
Solution

We have
\[
\frac{dy}{dx} = f(x, y) = \frac{y - x}{y + x}; \quad x_0 = 0, \ y_0 = 1, \ h = 0.1
\]

Hence the approximate value of \( y \) at \( x = 0.1 \) is given by

\[
y_1 = y_0 + hf(x_0, y_0) \\
= 1 + .02 \left( \frac{1 - 0}{1 + 0} \right) = 1.02
\]

At \( x_1 = 0.04 \), \( y_2 = y_1 + hf(x_1, y_1) \)
\[
= 1.02 + .02 \left( \frac{1.02 - .02}{1.02 + .02} \right) = 1.0392
\]

At \( x_2 = 0.06 \), \( y_3 = 1.0392 + (.02) \left( \frac{1.0392 - .04}{1.0392 + .04} \right) = 1.0577
\]

At \( x_3 = 0.08 \), \( y_4 = 1.0877 + (.02) \left( \frac{1.0877 - .06}{1.0877 + .06} \right) = 1.0756
\]

At \( x_4 = .1 \), \( y_5 = 1.0756 + (.02) \left( \frac{1.0756 - .08}{1.0756 + .08} \right) = 1.0928
\]

\[
y_1 = y_0 + hf(x_0, y_0) \quad \text{using } y_{n+1} = y_n + hf(x_n, y_n)
\]
\[
= 1 + (.1) + \left( \frac{1 - 0}{1 + 0} \right) = 1.1
\]

Much better accuracy is obtained by breaking up the interval 0 to 0.1 into five steps. The approximate value of \( y \) at \( x_5 = .02 \) is given by,

Hence, \( y = 1.0928 \) when \( x = 0.1 \)

Fig. 5.3 Graph for \( Y=1.0928 \) and \( x=0.1 \)
Consider the differential equation
With the condition

\[ y(x) = y_0 + (x - x_0)y'_0 + \frac{(x - x_0)^2}{2!} y''_0 + \frac{(x - x_0)^3}{3!} y'''_0 + \ldots \]  
\[ \text{(9)} \]

If \( y(x) \) is the exact solution of (9) then \( y(x) \) can be expanded into a Taylor’s series about the point \( x = x_0 \) as

\[ y(x) = y_0 + (x - x_0)y'_0 + \frac{(x - x_0)^2}{2!} y''_0 + \frac{(x - x_0)^3}{3!} y'''_0 + \ldots \]
\[ \text{(10)} \]

Where dashes denote differentiation with respect to \( x \).

Differentiating (9) successively with respect to \( x \), we get

\[ y'' = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \frac{dy}{dx} = \frac{\partial f}{\partial x} + f \frac{\partial f}{\partial y} \]
\[ \text{(11)} \]
\[ y''' = \frac{d}{dx} (y'') = \left( \frac{\partial f}{\partial x} + f \frac{\partial f}{\partial y} \right) \left( \frac{\partial f}{\partial x} + f \frac{\partial f}{\partial y} \right) \]
\[ \text{(12)} \]
and so on.

Putting \( x = x_0 \) and \( y = y_0 \) in the expressions for \( y' \), \( y'' \), \( y''' \) ... and substituting them in eqn.(10), we get a power series for \( y(x) \) in powers of \( x - x_0 \).

\[ y(x) = y_0 + (x - x_0)y'_0 + \frac{(x - x_0)^2}{2!} y''_0 + \frac{(x - x_0)^3}{3!} y'''_0 + \ldots \]
\[ \text{(13)} \]

Putting \( x = x_1 (= x_0 + h) \) in (13), we get

\[ y_1 = y(x_1) = y_0 + hy'_0 + \frac{h^2}{2!} y''_0 + \frac{h^3}{3!} y'''_0 + \ldots \]
\[ \text{(14)} \]

Here \( y'_0 \), \( y''_0 \), \( y'''_0 \) ... can be found by using (9) and its successive differentiations (11) and (12) at \( x = x_0 \). The series (14) can be truncated at any stage if \( h \) is small.

After obtaining \( y_1 \), we can calculate \( y'_1 \), \( y''_1 \), \( y'''_1 \), ...... from (9) at \( x_1 = x_0 + h \).

Now, expanding \( y(x) \) by Taylor’s series about \( x = x_1 \), we get

\[ y_2 = y_1 + hy'_1 + \frac{h^2}{2!} y''_1 + \frac{h^3}{3!} y'''_1 + \ldots \]

Proceeding, we get

\[ y_n = y_{n-1} + hy'_{n-1} + \frac{h^2}{2!} y''_{n-1} + \frac{h^3}{3!} y'''_{n-1} + \ldots \]

Practically, this method is not of much importance because of its need of partial derivatives.

Moreover if we are interested in a better approximation with a small truncation error, the evaluation of higher order derivatives is needed which are complicated in evaluation.

Besides its impracticability, it is useful in judging the degree of accuracy of the approximations given by other methods.
We can determine the extent to which any other formula agrees with the Taylor’s series expansion.

Taylor’s method is one of those methods which yield the solution of a differential equation in the form of a power series. This method suffers from a serious disadvantage that \( h \) should be small enough so that successive terms in the series diminish quite rapidly.

5.11 Taylor’s Method For Simultaneous 1 Order Differential Equations

Simultaneous differential equations of the type
\[
\frac{dy}{dx} = f(x, y, z)
\]
and
\[
\frac{dz}{dx} = g(x, y, z)
\]
With initial conditions \( y(x_0) = y_0 \) and \( z(x_0) = z_0 \) can be solved by Taylor’s method.

If \( h \) is the step-size then
\[ y_1 = y(x_0 + h) \text{ and } z_1 = z(x_0 + h) \]

Taylor’s algorithm for (15) and (16) gives
\[
y_1 = y_0 + hy_0' + \frac{h^2}{2!} y_0'' + \frac{h^3}{3!} y_0''' + \ldots
\]
and
\[
z_1 = z_0 + hz_0' + \frac{h^2}{2!} z_0'' + \frac{h^3}{3!} z_0''' + \ldots
\]

Differentiating (15) and (16) successively, we get \( y'' \), \( y''' \), \ldots, \( z'' \), \( z''' \), \ldots etc. So the values \( y_0'' \), \( y_0''' \), \ldots and \( z_0'' \), \( z_0''' \), \ldots can be obtained. Substituting them in (17) and (18), we get \( y_1 \), \( z_1 \) for the next step.

\[
y_2 = y_1 + hy_1' + \frac{h^2}{2!} y_1'' + \frac{h^3}{3!} y_1''' + \ldots
\]
and
\[
z_2 = z_1 + hz_1' + \frac{h^2}{2!} z_1'' + \frac{h^3}{3!} z_1''' + \ldots
\]

Since \( y_1 \) and \( z_1 \) are known, \( y_1' \), \( y_1'' \), \( y_1''' \), \ldots, \( z_1' \), \( z_1'' \), \( z_1''' \), \ldots can be calculated. Hence \( y_2 \) and \( z_2 \) can be obtained. Proceeding in this manner, we get other values of \( y \), step-by-step.

Example

Use Taylor’s series method to solve
\[
\frac{dy}{dx} = x + y; \quad y(1) = 0
\]
Numerically up to \( x = 1.2 \) with \( h = 0.1 \). Compare the final result with the value of explicit solution.

Solution

Here,
By Taylor’s series, we have
\[ y_1 = y_0 + h y_0' + \frac{h^2}{2!} y_0'' + \frac{h^3}{3!} y_0''' + \frac{h^4}{4!} y_0^{(iv)} + \ldots \]
\[ y(1+h) = 0 + (0.1) 1 + \frac{(0.1)^2}{2!} 2 + \frac{(0.1)^3}{3!} 2 + \frac{(0.1)^4}{4!} 2 + \ldots \]
\[ y(1.1) = 0.1103081 = 0.110 \text{ (app.)} \]

Also,
\[ x_1 = x_0 + h = 1.1 \]
\[ y_1' = x_1 + y_1 = 1.1 + 0.11 = 1.21 \]
\[ y_1'' = 1 + y_1' = 1 + 1.21 = 2.21 \]
\[ y_1''' = y_1'' = 2.21 \]
\[ y_1^{(iv)} = 2.21 \]
\[ y_1^{(v)} = 2.21 \]

Now,
\[ y(1.1+h) = y_1 + h y_1' + \frac{h^2}{2!} y_1'' + \frac{h^3}{3!} y_1''' + \ldots \]
\[ = 0.11 + (0.1)(1.21) + \frac{(0.1)^2}{2} (2.21) + \ldots \]
\[ y(1.2) = 0.232 \text{ (app.)} \]

The analytical solution of the given differential equation is
\[ y = -x - 1 + 2e^{x-1} \]

When \( x = 1.2 \), we get
\[ y = -1.2 - 1 + 2e^{1.2} = 0.242. \]

### 5.12 Runge-Kutta Methods

- More efficient methods in terms of accuracy were developed by two German Mathematicians Carl Runge (1856-1927) and Wilhelm Kutta (1867-1944).
- These methods are well-known as Runge-Kutta methods.
- They are distinguished by their orders in the sense that they agree with Taylor’s series solution up to terms of \( h^r \) where \( r \) is the order of the method.
These methods do not demand prior computation of higher derivatives of \( y(x) \) as in Taylor’s method.

In place of these derivatives, extra values of the given function \( f(x, y) \) are used.

The fourth order Runge-Kutta method is used widely for finding the numerical solutions of linear or non-linear ordinary differential equations.

Runge-Kutta methods are referred to as single-step methods.

The major disadvantage of Runge-Kutta methods is that they use many more evaluations of the derivative \( f(x, y) \) to obtain the same accuracy compared with multi-step methods.

A class of methods known as Runge-Kutta methods combines the advantage of high order accuracy with the property of being one step.

### 5.13 First Order Runge-Kutta Method

Consider the differential equation:

\[
\frac{dy}{dx} = f(x, y); \quad y(x_0) = y_0
\]  

(19)

Euler’s method gives

\[
y_1 = y_0 + hf(x_0, y_0) = y_0 + h f(x_0, y_0)
\]

(20)

Expanding by Taylor’s series, we get

\[
y_1 = y(x_0 + h) = y_0 + h f(x_0, y_0) + \frac{h^2}{2!} y_0'' + \ldots
\]

(21)

Comparing (20) and (21), it follows that Euler’s method agrees with Taylor’s series solution up to the term in \( h \). Hence Euler’s method is the first order Runge-Kutta method.

### 5.14 Second Order Runge-Kutta Method

- Consider the differential equation
  
  \[ y' = f(x, y) \text{ with the initial condition } y(x_0) = y_0 \]

- Let \( h \) be the interval between equidistant values of \( x \) then in II order Runge-Kutta method, the first increment in \( y \) is computed from the formulae
  
  \[
  k_1 = hf(x_0, y_0) \\
  k_2 = hf(x_0 + h, y_0 + k_1)
  \]

  \[ \Delta y = \frac{1}{2} (k_1 + k_2) \]

- Taken in the given order.
  
  \[
  x_1 = x_0 + h \\
  y_1 = y_0 + \Delta y = y_0 + \frac{1}{2} (k_1 + k_2)
  \]

- In a similar manner, the increment in \( y \) for the second interval is computed by means of the formulae,
  
  \[
  k_1 = hf(x_1, y_1) \\
  k_2 = hf(x_1 + h, y_1 + k_1)
  \]

  \[ \Delta y = \frac{1}{2} (k_1 + k_2) \]

  and similarly for the next intervals.
The inherent error in the second order Runge-Kutta method is of order $h^3$.

### 5.15 Third Order Runge-Kutta Method

This method gives the approximate solution of the initial value problem

\[
\frac{dy}{dx} = f(x, y); \quad y(x_0) = y_0 \quad \text{as}
\]

\[
y_1 = y_0 + \delta y
\]

\[
\delta y = \frac{h}{6}(k_1 + 4k_2 + k_3)
\]

Here,

\[
k_1 = f(x_0, y_0)
\]

\[
k_2 = f \left( x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2} \right)
\]

\[
k_3 = f(x_0 + h, y_0 + k')
\]

Formula (22) can be generalized for successive approximations. Expression in (22) agrees with Taylor’s series expansion for $y_1$ up to and including terms in $h^3$. This method is also known as Runge’s method.

### 5.16 Fourth Order Runge-Kutta Method

- The fourth order Runge-Kutta Method is one of the most widely used methods and is particularly suitable in cases when the computation of higher derivatives is complicated.
- Consider the differential equation $y' = f(x, y)$ with the initial condition $y(x_0) = y_0$.
- Let $h$ be the interval between equidistant values of $x$, then the first increment in $y$ is computed from the formulae.

\[
k_1 = hf'(x_0, y_0)
\]

\[
k_2 = hf' \left( x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2} \right)
\]

\[
k_3 = hf' \left( x_0 + \frac{h}{2}, y_0 + \frac{k_2}{2} \right)
\]

\[
k_4 = hf' \left( x_0 + h, y_0 + k_3 \right)
\]

\[
\Delta y = \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)
\]

_taken in the given order.

- Then, $x_i = x_0 + h$ and $y_i = y_0 + \Delta y$
- In a similar manner, the increment in $y$ for the II interval is computed by means of the formulae
\[ k_1 = hf(x_1, y_1) \]
\[ k_2 = hf\left(x_1 + \frac{h}{2}, y_1 + \frac{k_1}{2} \right) \]
\[ k_3 = hf\left(x_1 + \frac{h}{2}, y_1 + \frac{k_2}{2} \right) \]
\[ k_4 = hf(x_1 + h, y_1 + k_3) \]
\[ \Delta y = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \]

and similarly for the next intervals.

- \( y' = f(x, y) \), \( y(x_0) = y_0 \) as Runge-Kutta’s method.
- It is to be noted that the calculations for the first increment are exactly the same as for any other increment. The change in the formula for the different intervals is only in the values of \( x \) and \( y \) to be substituted. Hence, to obtain \( \Delta y \) for the \( n \)th interval, we substitute \( x_{n-1}, y_{n-1} \), in the expressions for \( k_1, k_2 \), etc.
- The inherent error in the fourth order Runge-Kutta method is of the order \( h^5 \).

### 5.17 Predictor-Corrector Methods

- In Runge-Kutta Methods, we need only the information at \( (x_i, y_i) \) to calculate the value of \( y_{i+1} \) and no attention is paid to the nature of the solution at the earlier points.
- To overcome this defect, Predictor-Corrector Methods are useful.
- The technique of refining an initially crude predicted estimate of \( y_i \) by means of a more accurate corrector formula is called, Predictor-Corrector Method.
- The modified Euler’s Method of solving the initial value problem, can be stated as

\[ y_1^p = y_0 + hf(x_0, y_0) \]  
(24)

\[ y_1^c = y_0 + \frac{h}{2} \left[ f(x_0, y_0) + f(x_1, y_1^p) \right] \]  
(25)

Here we predict the value of \( y_1 \) by Euler’s Method and use it in (26) to get a corrected or improved value. This is a typical case of Predictor-Corrector Method.

- In this section, we will obtain two important Predictor-Corrector Methods, namely, Milne’s Simpson Method and Adams-Moulton (or Adams-Bash Fourth) Method.
- Both of these methods are of IV order and the error is of order \( h^5 \).
- These methods make use of four starting values of \( y \), namely, \( y_0, y_1, y_2, \) and \( y_3 \). Hence, these methods are also called as Multi-Step Methods.
Summary

- A physical situation concerned with the rate of change of one quantity with respect to another gives rise to a differential equation.
- Many analytical techniques exist for solving such equations, but these methods can be applied to solve only a selected class of differential equations.
- However, a majority of differential equations appearing in physical problems cannot be solved analytically.
- Thus it becomes imperative to discuss their solution by numerical methods.
- In numerical methods, we do not proceed in the hope of finding a relation between variables but we find the numerical values of the dependent variable for certain values of independent variable.
- Problems in which all the conditions are specified at the initial point only are called initial-value problems.
- The numerical solutions are obtained step-by-step through a series of equal intervals in the independent variable so that as soon as the solution $y$ has been obtained at $x = x_i$, the next step consists of evaluating $y_{i+1}$ at $x = x_{i+1}$.
- The single step method has obvious advantages over the multi-step methods that use several past values ($y_{n}$, $y_{n-1}$, $y_{n-2}$, ...., $y_{n-p}$) and that require initial values ($y_{1}$, $y_{2}$, ...., $y_{n}$) that have to be calculated by another method.
- Picard was a distinguished Professor of Mathematics at the university of Paris, France.
- Euler’s method is the simplest one-step method and has a limited application because of its low accuracy. This method yields solution of an ordinary differential equation in the form of a set of tabulated values.
- The modified Euler’s method gives greater improvement in accuracy over the original Euler’s method.
- More efficient methods in terms of accuracy were developed by two German Mathematicians Carl Runge (1856-1927) and Wilhelm Kutta (1867-1944). These methods are well-known as Runge-Kutta methods.

References


Recommended Reading

Self Assessment

1. A physical situation concerned with the rate of change of one quantity with respect to another gives rise to a differential __________.
   a. equation
   b. solution
   c. resolution
   d. evolution

2. Problems in which all the conditions are specified at the initial point only are called __________.
   a. final-value problems
   b. initial-value problems
   c. last-value problems
   d. null-value problems

3. Who was a distinguished Professor of Mathematics at the University of Paris, France?
   a. Picard
   b. Dr. Hung. T. Nguyen
   c. Euler
   d. Gaussian

4. Which statement is true?
   a. Euler’s method is the simplest one-step method and has no limited application because of its low accuracy.
   b. Euler’s method is the simplest two-step method and has no limited application because of its low accuracy.
   c. Euler’s method is the simplest one-step method and has a limited application because of its low accuracy.
   d. Euler’s method is the simplest two-step method and has a limited application because of its low accuracy.

5. The modified Euler’s method gives greater improvement in _______ over the original Euler’s method.
   a. looseness
   b. quality
   c. accuracy
   d. temperance

6. More efficient methods in terms of accuracy were developed by two ________ mathematicians.
   a. Indian
   b. German
   c. Canadian
   d. European

7. Runge-Kutta methods are referred to as ________ step methods.
   a. triple
   b. double
   c. single
   d. multiple
8. Which order of the Runge-Kutta method is used widely for finding the numerical solutions of linear or non-linear ordinary differential equations?
   a. first
   b. second
   c. third
   d. fourth

9. __________ order of the Runge-Kutta method gives the approximate solution of the initial value problem.
   a. First
   b. Second
   c. Third
   d. Fourth

10. Which statement is true?
    a. Taylor’s method is one of those methods which yield the solution of a differential equation in the form of a power series.
    b. Taylor’s method is one of those methods which yield the solution of a differential equation in the form of an energy series.
    c. Taylor’s method is one of those methods which yield the solution of a differential equation in the form of an efficiency series.
    d. Taylor’s method is one of those methods which yield the solution of a differential equation in the form of a control series.
Case Study I

Interpolation

In this chapter, we considered approximating integrals of functions of one, two, or three variables and approximating the derivatives of a function of a single real variable. The Midpoint rule, Trapezoidal rule, and Simpson’s rule are introduced to elucidate the techniques and error analysis of quadrature methods.

Composite Simpson’s rule is easy to use and produces accurate approximations unless the function oscillates in a subinterval of the interval of integration. Adaptive quadrature can be used if the function is suspected of oscillatory behaviour.

To minimise the number of nodes while maintaining accuracy, we studied Gaussian quadrature. Romberg integration was introduced to take advantage of the easily applied Composite Trapezoidal rule and extrapolation. Most software for integrating a function of a single real variable is based on the adaptive approach or extremely accurate Gaussian formulas.

Cautious Romberg integration is an adaptive technique that includes a check to make sure that the integrand is smoothly behaved over subintervals of the integral of integration. This method has been successfully used in software libraries.

Multiple integrals are generally approximated by extending good adaptive methods to higher dimensions. Gaussian-type quadrature is also recommended to decrease the number of function evaluations.

The main routines in both the IMSL and NAG Libraries are based on QUADPACK: A Subroutine Package for Automatic Integration by R. Piessens, E. de Doncker-Kapenga, C. W. Uberhuber, and D. K. Kahaner published by Springer-Verlag in 1983 [PDUK].

The IMSL Library contains the function QDAG, which is an adaptive integration scheme based on the 21-point Gaussian-Kronrod rule using the l0-point Gaussian rule for error estimation.

The Gaussian rule uses the ten points \( x_1, \ldots, x_{10} \) and weights \( w_1, \ldots, w_{10} \) to give the quadrature formula

\[
\sum_{i=1}^{10} w_i f(x_i)
\]

to approximate \( \int_a^b f(x) \, dx \). The additional points \( x_{11}, \ldots, x_{21} \), and the new weights \( v_1, \ldots, v_{21} \), are then used in the Kronrod formula

\[
\sum_{i=1}^{21} v_i f(x_i)
\]

The results of the two formulas are compared to eliminate error. The advantage in using \( x_1, \ldots, x_{10} \) in each formula is that \( f \) needs to be evaluated only at 21 points. If independent 10-point and 21-point Gaussian rules were used, 31 function evaluations would be needed. This procedure permits endpoint singularities in the integrand.

Other IMSL subroutines are QDAGS, which allows for end point singularities; QDAGP, which allows user-specified singularities; QDAGI, which allows infinite intervals of integration; and QDNG. This is a nonadaptive procedure for smooth functions. The subroutine TWODQ uses the Gauss-Kronrod rules to integrate a function of two variables. There is also a subroutine QAND to use Gaussian quadrature to integrate a function of \( n \) variables over \( n \) intervals of the form \([a_i, b_i]\).

The NAG Library includes the subroutine D01AJF to compute the integral of \( f \) over the interval \([a, b]\) using an adaptive method based on Gaussian Quadrature using Gauss 10-point and Kronrod 21-point rules. The subroutine D01AHF is used to approximate \( \int_a^b f(x) \, dx \) using a family of Gaussian-type formula based on 1, 3, 5, 7, 15, 31, 63, 127, and 255 nodes. These interfacing high-precision rules are due to Patterson [Pat] and are used in an adaptive manner. The subroutine D01GBF is for multiple integrals and DOIGAF approximates an integral given only data points instead of the function. NAG includes many other subroutines for approximating integrals.
The Maple function call
\[ \texttt{int(f,a..b);} \]

Computes the definite integral \( \int_{a}^{b} f(x) \, dx \). The numerical method applies singularity handling routines and then uses Clenshaw-Curtis quadrature. If this fails, due to singularities in or near the interval, then an adaptive double-exponential quadrature method is applied. The adaptive Newton-Cotes formula can be applied by specifying the option \_NCrule in the Maple function call.

\[ \texttt{int(f,a..b, digits,\_NCrule);} \]

The method attempts to achieve a relative error tolerance \( 0.5 \times 10^{-(\text{Digits})} \), where Digits is a variable in Maple that specifies the number of digits of rounding Maple uses for numerical calculation.

The default value for Digits is 10, but it can be changed to any positive integer \( n \) by the command \( \text{Digits} = n; \)

The MATLAB command QUAD approximates the definite integral \( \int_{a}^{b} f(x) \, dx \) using an adaptive Simpson’s rule, and QUAD8 approximates the definite integral using an adaptive eight-panel Newton-Cotes rule.

Although numerical differentiation is unstable, derivative approximation formula are needed for solving differential equations. The NAG Library includes the subroutine D04AAF for the numerical differentiation of a function of one real variable with differentiation to the fourteenth derivative being possible.

The IMSL function DERIV uses an adaptive change in step size for finite differences to approximate the first, second, or third, derivative of \( f \) at \( x \) to within a given tolerance. IMSL also includes the subroutine QDDER to compute the derivatives of a function defined on a set of points using quadratic interpolation. Both packages allow the differentiation and integration of interpolatory cubic splines constructed by the subroutines.

For further reading on numerical integration we recommend the books by Engels [E] and by Davis and Rabinowitz [DR]. For more information on Gaussian quadrature see Stroud and Secrest [StS]. Books on multiple integrals include those by Stroud [Stro] and the recent book by Sloan and Joe [SJ].

Questions:

1. Which methods are used to describe error analysis of quadrature?
   **Answer:**
   We considered approximating integrals of functions of one, two, or three variables and approximating the derivatives of a function of a single real variable. The Midpoint rule, Trapezoidal rule, and Simpson’s rule were studied to introduce the techniques and error analysis of quadrature methods.

2. Explain Composite Simpson’s rule in short.
   **Answer:**
   Composite Simpson’s rule is easy to use and produces accurate approximations unless the function oscillates in a subinterval of the interval of integration. Adaptive quadrature can be used if the function is suspected of oscillatory behaviour.

3. What do we do to minimise the number of nodes while maintaining accuracy?
   **Answer:**
   To minimise the number of nodes while maintaining accuracy, we studied Gaussian quadrature. Romberg integration was introduced to take advantage of the easily applied Composite Trapezoidal rule and extrapolation.
4. What is Cautious Romberg integration?
   **Answer:**
   Cautious Romberg integration is an adaptive technique that includes a check to make sure that the integrand is smoothly behaved over subintervals of the integral of integration. This method has been successfully used in software libraries.

5. Explain libraries used main routine.
   **Answer:**
   The main routines in both the IMSL and NAG Libraries are based on QUADPACK: A Subroutine Package for Automatic Integration by R. Piessens, E. de Doncker-Kapenga, C. W. Uberhuber, and D. K. Kahaner published by Springer-Verlag in 1983 [PDUK].
Case Study II

Case Study on Initial-Value

In this subject we have considered methods to approximate the solutions to initial-value problems for ordinary differential equations.

We began with a discussion of the most elementary numerical technique, Euler’s method. This procedure is not sufficiently accurate to be of use in applications, but it illustrates the general behaviour of the more powerful techniques, without the accompanying algebraic difficulties.

The Taylor methods were then considered as generalizations of Euler’s method. They were found to be accurate but cumbersome because of the need to determine extensive partial derivatives of the defining function of the differential equation.

The Runge-Kutta formulas simplified the Taylor methods, while not significantly increasing the error.

To this point we had considered only one-step methods, techniques that use only data at the most recently computed point. Multistep methods are discussed in some section, where explicit methods of Adams Bashforth type and implicit methods of Adams-Moulton type were considered. These culminate in predictor-corrector methods, which use an explicit method, such as an Adams Bashforth, to predict the solution and then apply a corresponding implicit method, like an Adams-Moulton, to correct the approximation.

Methods of the Runge-Kutta-Fehlberg type are generally sufficient for nonstiff problems when moderate accuracy is required. The extrapolation procedures are recommended for nonstiff problems where high accuracy is required. Extensions of the implicit Trapezoidal method to variable-order and variable step-size implicit Adams-type methods are used for stiff initial-value problems.

The IMSL Library includes two subroutines for approximating the solutions of initial value problems. Each of the methods solves a system of m first-order equations in m variables. The equations are of the form

\[
\frac{du_i}{dt} = f_i(t, u_1, u_2, \ldots, u_m), \quad \text{for } i = 1, 2, \ldots, m,
\]

Where \(u_i(t_0)\) is given for each \(i\). The variable step-size subroutine IVPRK is based on the Runge-Kutta-Verner fifth-order and sixth-order methods described in subject. A subroutine of Adams type to be used for stiff equations is due to C. William Gear and is given by IVPAG. This method uses implicit multistep methods of order up to 12 and backward differentiation formulas of order up to 5.

The Runge-Kutta-type procedures contained in the NAG Library are called D02BGF, D02BHF, D02PCF, and D02PDE. D02BGF and D02BHF are based on the Merson form of the Runge-Kutta method. A variable-order and variable step-size Adams method is contained in the procedure D02CJE. Variable step-size backward-difference methods for stiff systems are contained in the procedure D02EJF. Other routines incorporate the same methods but iterate until a component of the solution attains a given value or until a function of the solution is zero.

The netlib Library includes several subroutines for approximating the solutions of initial-value problems in the package ODE. The subroutine dverk.f is based on the Runge-Kutta-Verner fifth- and sixth-order methods. The subroutine rkf45.f is based on the Runge-Kutta-Fehlberg fourth-order and fifth-order methods. For stiff ordinary differential equation initial value problems, the subroutine epsode.f based on variable coefficient backward differentiation formula can be used.
Questions:
1. Explain Taylor’s method.
2. Explain Euler’s method.
3. What is an IMSL Library?
4. Discuss NAG library in detail.
5. What is netlib Library?
Case Study III

Case Study on Eigen Value

In the subject discussed the approximation of eigenvalues and eigenvectors. The Gerschgorin circles give a crude approximation to the location of the eigenvalues of a matrix. The Power method can be used to find the dominant eigenvalues and an associated eigenvector for an arbitrary matrix A. If A is symmetric, the Symmetric Power method gives faster convergence to the dominant eigenvalue and an associated eigenvector.

The Inverse Power method will find the eigenvalue closest to a given value and an associated eigenvector. This method is often used to refine an approximate eigenvalue and to compute an eigenvector once an eigenvalue has been found by some other technique. Deflation methods, such as Wielandt deflation, obtain other eigenvalues once the dominant eigenvalue is known. These methods are used if only a few eigenvalues are required since they are susceptible to roundoff error.

The Inverse Power method should be used to improve the accuracy of approximate eigenvalues obtained from a deflation technique. Methods based on similarity transformations, such as Householder’s method, are used to convert a symmetric matrix into a similar matrix that is tridiagonal (or upper Hessenberg if the matrix is not symmetric).

Techniques such as the QR method can then be applied to the tridiagonal (or upper Hessenberg) matrix to obtain approximations to all the eigenvalues. The associated eigenvectors can be found by using an iterative method, such as the Inverse Power method, or by modifying the QR method to include the approximation of eigenvectors.

We restricted our study to symmetric matrices and presented the QR method only to compute eigenvalues for the symmetric case. The subroutines in the IMSL and NAG libraries are based on those contained in EISPACK and LAPACK.

In general, the subroutines transform a matrix into the appropriate form for the QR method or one of its modifications, such as the QL method. The subroutines approximate all the eigenvalues and can approximate an associated eigenvector for each eigenvalue. There are special routines that find all the eigenvalues within an interval or region or that find only the largest or smallest eigenvalue.

Subroutines are also available to determine the accuracy of the eigenvalues approximation and the sensitivity of the process to roundoff error. The LAPACK routine SGEBAL prepares a real nonsymmetric matrix A for further processing.

It tries to use permutation matrices to transform A to a similar block upper triangular form. Similarity transformations are used to balance the rows and columns in form. The routine SGEHRD can then be used to convert A to a similar upper Hessenberg matrix H.

The matrix H is then reduced via SHSEQR to Schur form ST S’, where S is orthogonal and the diagonal of T holds the eigenvalues of A. STREVC can then be used to obtain the corresponding eigenvectors.

The LAPACK routine SSYTRD is used to reduce a real symmetric matrix A to a similar tridiagonal matrix via Householder’s method. The routine SSTEQR uses an implicitly shifted QR algorithm to obtain all the eigenvalues and eigenvectors of A.

The IMSL subroutine EVLRG produces all eigenvalues of A in increasing order of magnitude. This subroutine first balances the matrix A using a version of the EISPACK routine BALANC, so that the sums of the magnitudes of the entries in each row and in each column are approximately the same. This leads to greater stability in the ensuing computations.
EVLRG next performs orthogonal similarity transformations, such as in Householder’s method, to reduce A to a similar upper Hessenberg matrix. This portion is similar to the EISPACK subroutine ORTHES.

Finally, the shifted QR algorithm is performed to obtain all the eigenvalues. This part is similar to the subroutine HQR in EISPACK. The IMSL subroutine EVCRG is the same as EVRLG, except that corresponding eigenvectors are computed. The subroutine EVLSF computes the eigenvalues of the real symmetric matrix A. The matrix A is first reduced to tridiagonal form using a modification of the EISPACK routine TRED2. Then the eigenvalues are computed using a modification of the EISPACK routine IMTQL2, which is a variation of the QR method called the implicit QL method.

The subroutine EVCSF is the same as EVLSF except that the eigenvectors are also calculated. Finally, EVLRH and EVCRH compute all eigenvalues of the upper Hessenberg matrix A and, additionally, EVCRH computes the eigenvectors. These subroutines are based on the subroutines HQR and HQR2, respectively, in EISPACK. The NAG library has similar subroutines based on the EISPACK routines. The subroutine F02EBF computes the eigenvalues of a real matrix and, optionally, the eigenvectors. The matrix is first balanced and then is reduced to upper-Hessenberg form for the QR method. If only the eigenvalues are required, then the algorithm uses a Hessenberg QR method to compute the eigenvalues; if the eigenvectors are also required, then a Schur factorization is used.

The subroutine F02FAF is used on a real symmetric matrix to compute the eigenvalues in increasing order of magnitude and optionally the eigenvectors. The subroutine first reduces the matrix to tridiagonal form using Householder’s method.

The eigenvalues are then computed using a variant of the symmetric tridiagonal QR algorithm. The subroutine F08FEF implements Householder’s algorithm directly for symmetric matrices to produce a similar tridiagonal symmetric matrix.

Routines are also available in the NAG library for directly balancing real matrices, recovering eigenvectors if a matrix was first balanced, and performing other operations on special types of matrices.

Questions:
1. What do you mean by Eigen value and Eigen vector?
2. Explain inverse power method.
3. What is use of QR method?
4. Write about NAG and IMSL libraries.
5. Write in short about LAPACK routine.
Bibliography

References

- Oscar Williams, O., *Commercial Arithmetic, Profit & Loss*. Biblio Bazar. pp3-85
- Stone, M. H, 1980.*The borderline syndromes: constitution, personality, and adaptation.* Psychology.

Recommended Reading

Self Assessment Answers

Chapter I
1. a
2. b
3. d
4. c
5. c
6. d
7. d
8. a
9. a
10. a

Chapter II
1. b
2. a
3. c
4. a
5. d
6. a
7. d
8. d
9. c
10. a

Chapter III
1. a
2. b
3. d
4. b
5. b
6. a
7. a
8. c
9. d
10. c

Chapter IV
1. a
2. a
3. c
4. b
5. c
6. a
7. b
8. a
9. a
10. c
Chapter V
1. a
2. b
3. a
4. c
5. c
6. b
7. c
8. d
9. c
10. a