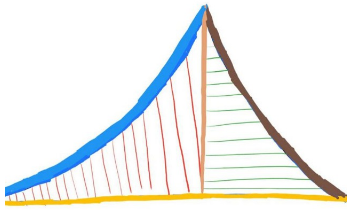
COMPUTER PROGRAMMING
Through High School Mathematics

G. V. V. Sharma



Contents

Introduction

This book introduces computer programming through high school mathematics

Chapter 1

Installation

1.1. Software Installation

1. On your android device, install fdroid apk from

   ```
   https://www.f-droid.org/
   ```

2. Install Termux from apkpure

3. Install basic packages on termux

   ```
   #Give termux access to your user directory in android
   termux-setup-storage


   #Upgrade packages
   apt update && apt upgrade
   apt install build-essential openssh


   #Mandatory packages
   apt install curl git wget subversion proot proot-distro python nmap neovim ranger
   #-------------------End Install Termux
   ```

```
——————————————————————————————
```

4. Install Ubuntu on termux

```
proot−distro install ubuntu
proot−distro login ubuntu
```

5. Install Packages

```
apt update && apt upgrade
apt install apt−utils build−essential cmake neovim
apt install git wget subversion imagemagick nano
apt install avra avrdude gcc−avr avr−libc
#———————————————————End Installing ubuntu on termux
    ——————————————————————————————

#——————————————————— Installing python3 on termuxubuntu
    ——————————————————————————————
apt install python3−pip python3−numpy python3−scipy python3−matplotlib
    python3−mpmath python3−sympy python3−cvxopt
#——————————————————— End installing python3 on termuxubuntu
    ——————————————————————————————
```

Chapter 2

The First Program

T:his manual shows how to generate data in a file using a C program and importing it in Python.

1. Graphically show that the function

$$f(x) = \begin{cases} -x & x < 1 \\ a + \cos^{-1}(x+b) & 1 \le x \le 2 \end{cases} \tag{2.1}$$

is continuous at $x = 1$ for $b = -1, a - b = -\frac{\pi}{2}$.

Solution: The following python code yields Fig. 2.1 verifying the above result.

```
import numpy as np
import matplotlib.pyplot as plt
#Computation
b = −1
x2 = np.linspace(−1,1,100)
x3 = np.linspace(1,2,100)
a = −1 − np.pi/2.0
y = −x2
z = a + np.arccos(b+(x3))
```

```
#Plotting
plt.plot(x3,z, label = '$f(x) = -x$')
plt.plot(x2,y, label = '$f(x) = a + \cos^{-1}(x+b)$')

sol = np.zeros((2,1))
sol[0] = 1
sol[1] = -1

#Display solution
A = sol[0]
B = sol[1]

plt.plot(A,B,'o')
for xy in zip(A,B):
    plt.annotate('(%s, %s)' % xy, xy=xy, xytext=(30,0), textcoords='offset 
        points')

plt.grid()
plt.legend(loc='best',prop={'size':11})
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
#Comment the following line
#plt.savefig('../figs/ee16b1005.eps')
plt.show()
```

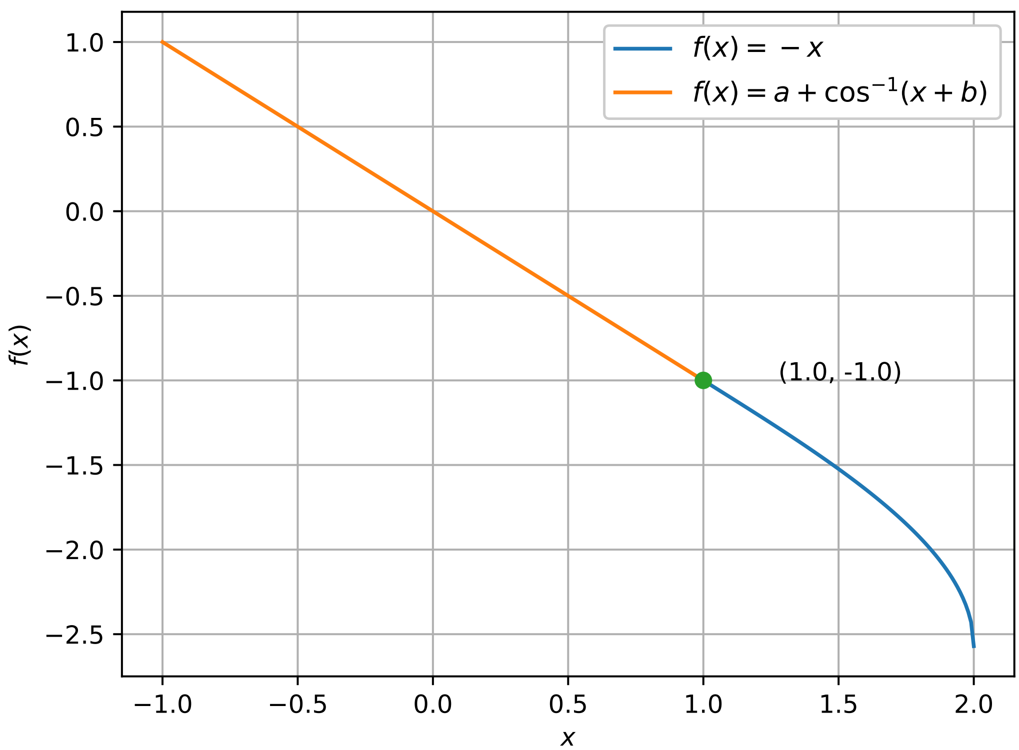


Figure 2.1: Substituting the values of a and b in $f(x)$, the graph is smooth at $x = 1$. So $f(x)$ is continuous as well as differentiable $x = 1$.

2. Write a C program to generate an arithmetic progression with first term $a = -1$, last term $l = 1$ and number of terms $n = 100$ and print the numbers on the screen.

   Solution:

```
#include <stdio.h>


int main(void)
{
```

```
float a = −1.0, l = 1.0, d;
int n = 100, i;

//Common difference
d = (l−a)/(n−1);

for(i = 0; i < 100; i++)
{
printf("%f\n",a+i*d);
}

return 0;
}
```

3. Repeat the above exercise by using functions for finding the common difference and the $n$term given a, l and n.

Solution:

```
#include <stdio.h>

float a_n(float,float,int);
float c_d(float,float,int);

int main(void)
{
float a = −1.0, l = 1.0, d;
```

```
int n = 100, i;
d = c_d(a,l,n);
for(i = 0; i < 100; i++)
{
printf("%f\n",a_n(a,d,i));
}

return 0;
}
//nth term of AP
float a_n(float a,float d,int i)
{
return a+i*d;
}

//Common difference
float c_d(float a,float l,int n)
{
float d;
d = (l-a)/(n-1);
return d;
}
```

4. Repeat the above exercise by printing the numbers in a file called test.dat

Solution:

```
#include <stdio.h>

int main(void)
{
FILE *fp;
float a = -1.0, l = 1.0, d;
int n = 100, i;

//Common difference
d = (l-a)/(n-1);

//Open file for writing
fp = fopen("test.dat", "w");

for(i = 0; i < 100; i++)
{
fprintf(fp, "%f\n", a+i*d);
}
fclose(fp);
return 0;
}
```

5. Now run the following program. Comment.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
#Computation
b = -1
x2 = np.loadtxt('test.dat',dtype='float')
#x2 = np.linspace(-1,1,100)
x3 = np.linspace(1,2,100)
a = -1 - np.pi/2.0
y = -x2
z = a + np.arccos(b+(x3))

#Plotting
plt.plot(x3,z, label = '$f(x) = -x$')
plt.plot(x2,y, label = '$f(x) = a + \cos^{-1}(x+b)$')

sol = np.zeros((2,1))
sol[0] = 1
sol[1] = -1

#Display solution
A = sol[0]
B = sol[1]

plt.plot(A,B,'o')
for xy in zip(A,B):
        plt.annotate('(%s, %s)' % xy, xy=xy, xytext=(30,0), textcoords='offset points')
```

```
plt.grid()
plt.legend(loc='best',prop={'size':11})
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
plt.show()
```

6. Compute $f(x)$ in (2.1) through a C program

   Solution:

```
#include <stdio.h>
#include <math.h>

float f(float);



int main(void)
{

printf("%f\n",f(1.99));

return 0;
}

//Common difference
float f(float x)
```

```
{
float b = -1.0, a;

a = b-M_PI_2;

if(x < 1)
{
        return -x;

}
else if(x >= 1 && x <= 2)
{
return a+acos(x+b);
}
else

return 0;

}
```

7. Do all the computations in Problem 1 in C and verify your results by plotting in python.

Chapter 3

Data Structures

T:his manual shows how to use pointers for arrays as well as linked lists. Programming lists and trees is taught through polynomial algebra and matrix operations.

1. Write a C program to generate an arithmetic progression (AP) with first term $a = -1$, last term $l = 1$ and number of terms $n = 100$. Store these numbers in a pointer array.

   Solution:

```
#include <stdio.h>
#include <stdlib.h>

//Main function
int main(void)
{
//Variable declarations
double a = -1.0, l = 1.0, d, *ap;
int n = 100, i;

//Creating memory for ap
ap = (double *)malloc(n * sizeof(double));

//Common difference
```

```
d = (l−a)/(n−1);


//Generating the AP array
for(i = 0; i < 100; i++)
{
ap[i] = a+i*d;
}


//Printing values
for(i = 0; i < n; i++)
        printf("%lf\n",ap[i]);


free(ap);
return 0;
}
```

2. Modify the above program to create a function for generating the AP pointer array.

Solution:

```
#include <stdio.h>
#include <stdlib.h>


double *linspace_pointer(double, double, int );
int main(void)
{
double a = −1.0, l = 1.0, *ap;
```

```
int n = 100, i;

//Assigning pointer to a
ap = linspace_pointer(a,l,n);

for(i = 0; i < n; i++)
        printf("%lf\n",ap[i]);

//Common difference

return 0;
}

double *linspace_pointer(double a, double l, int n)
{
//Variable declarations
double d, *ap;
int i;

//Creating memory for ap
ap = (double *)malloc(n * sizeof(double));

//Common difference
d = (l-a)/(n-1);
```

```
//Generating the AP
for(i = 0; i < 100; i++)
{
ap[i] = a+i*d;
}
//Returning the address of the first memory block
return ap;


}
```

3. Repeat the above exercise through a list.

   Solution:

```
#include <stdio.h>
#include <stdlib.h>


typedef struct list
{
double data;
struct list *next;
}node;


node *linspace_pointer(double, double, int );


int main(void)
{
```

```
node *ap;
double a = -1.0, l = 1.0;
int n = 100;

//Getting the head of the AP list
ap = linspace_pointer(a,l,n);

//Printing the AP
while(ap->next != NULL)
{
        printf("%lf\n", ap->data);
        ap = ap->next;
}

return 0;
}

node *linspace_pointer(double a, double l, int n)
{
//Variable declarations
node *ap, *head;
double d;
int i;

//Common difference
```

```
d = (l-a)/(n-1);


ap = (node *)malloc(sizeof(node));
head = ap;
//Generating the AP
for(i = 0; i < 100; i++)
{


ap->data = a+i*d;
//Creating memory for next node
ap->next = (node *)malloc(sizeof(node));
//Initializing next node
ap->next->next = NULL;
//node increment
ap = ap->next;
}
//Returning the address of the first memory block
return head;


}
```

Consider the polynomials

$$p(x) = x + 1 \tag{3.1}$$

$$q(x) = x^2 + 2x + 3 \tag{3.2}$$

4. Polynomial Addition: Evaluate $p(x) + q(x)$ using pointer arrays.

5. Repeat the above exercise using a list.

6. Polynomial Multiplication: Using convolution, find $p(x)q(x)$ using pointer arrays

7. Repeat the above exercise using a list.

8. Generalize the above polynomial operations for any degree using both pointer arrays and lists.

9. Matrix Operations: Create a matrix using pointer arrays

   Solution:

```
#include <stdio.h>
#include <stdlib.h>

//This program shows how to use pointers as 2-D arrays

//Function declaration
double **createMat(int m,int n);
void readMat(int m,int n,double **p);
void print(int m,int n,double **p);
//End function declaration

int main() //main function begins
{

//Defining the variables
```

```
int m,n;//integers
double **a;

printf("Enter the size of the matrix m  n \n");
scanf("%d %d", &m,&n);



printf("Enter the values of the matrix\n");
a = createMat(m,n);//creating the matrix a
readMat(m,n,a);//reading values into the matrix a
print(m,n,a);//printing the matrix a

return 0;
}



//Defining the function for matrix creation
double **createMat(int m,int n)
{
 int i;
 double **a;

 //Allocate memory to the pointer
a = (double **)malloc(m * sizeof( *a));
    for (i=0; i<m; i++)
```

```
        a[i] = (double *)malloc(n * sizeof( *a[i]));

 return a;
}
//End function for matrix creation



//Defining the function for reading matrix
void readMat(int m,int n,double **p)
{
 int i,j;
 for(i=0;i<m;i++)
 {
  for(j=0;j<n;j++)
  {
   scanf("%lf",&p[i][j]);
  }
 }
}
//End function for reading matrix




//Defining the function for printing
void print(int m,int n,double **p)
```

```
{
 int i,j;

 for(i=0;i<m;i++)
 {
  for(j=0;j<n;j++)
  printf("%lf␣",p[i][j]);
 printf("\n");
 }
}
```

10. Let

$$A = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \tag{3.3}$$

Use pointer arrays for the following.

(a) Generate A^t which is the transpose of A.

(b) Obtain $A + A^t$.

(c) Obtain $A - A^t$.

(d) Obtain AA^t.

(e) Obtain A^{-1}.

11. Repeat the above exercise using a two dimensional list.

12. Binary Search Tree:

(a) Enter the list of numbers

$$S = \{3, 6, 2, 1, 5, 9, 4, 7, 0, 8\} \tag{3.4}$$

into a binary tree in such a fashion that the smaller number goes to the left brach.

(b) Access this tree in such a manner as to print the numbers in ascending order.

(c) Repeat the exercise to print the numbers in descending order.

(d) Try to do both the above exercises using recursion.

13. Polynomial Division: Let

$$q(x) = p(x)g(x) + r(x), \tag{3.5}$$

where $g(x)$ is the quotient polynomial and $r(x)$ is the remainder polynomial. Obtain the coefficient list for $g(x)$ and $r(x)$.

Chapter 4

Shared Libraries

T:his manual shows how to build a calculator using Python and shared C libraries. Through this, even beginners can learn how to build some simple software applications with graphical user interfaces (GUIs).

4.1. Python Calculator

1. Execute

```
calculator//EE1083/calculator/software/codes/solution/pythonprogs/tkcalc.py
```

4.2. Shared Libraries in GCC

1. Write a C function to multiply two given numbers. Save it in the file titled as **mul.c**

Solution:

```
//function to multiply two numbers

float mul(float num1, float num2)
  {
    return num1*num2; //function returns multiplication of num1 and num2
```

```
    }

//Run the following commnad for generating the .so file
//cc −fPIC −shared −o mul.so mul.c
```

2. Open the Terminal and go to the directory where the **mul.c** file is saved.

3. Type the following command in the Terminal.

   Solution:

```
cc −fPIC −shared −o mul.so mul.c
```

   Note that you will have to use the **-lm** switch for **math.h** functions.

4. Type the following program in **main.c**

   Solution:

```
#include <stdio.h>

float mul(float,float);

int main(void)
{
printf("%f\n",mul(4,5));
return 0;
}
//gcc main.c mul.so −Wl,−rpath=$(pwd)
```

5. Run the above program

   Solution:

   ```
   gcc main.c mul.so -Wl,-rpath=$(pwd)
   ./a.out
   ```

   The advantange of using **mul.so** is that the multiplication function needs to be compiled only once. It can then be used in any C program.

6. Repeat the above exercises for adding two numbers.

7. Write all the required C routines for the calculator in Problem 1 and generate the shared libraries. Test all the routines.

4.3. Shared libraries in Python

1. Write a Python script to multiply two numbers using C function.

   Solution:

   ```
   #Calling C function in Python
   from ctypes import *

   #load the shared object file
   multip = CDLL('./mul.so')

   a=2.0
   b=8.0

   ```

```
#Find multiplication of floats


mul = multip.mul
mul.restype = c_float


print (a,"x",b,"=", mul(c_float(a), c_float(b)))
```

2. Call the function written above in the Python GUI calculator to perform multipication.

   Solution: Save

   ```
   EE1083/calculator/software/codes/pythonprogs/calc_mul_root.py
   ```

   in directory where **mul.c** is saved. Execute **calc_mul_root.py**.

3. Use C routines in **calc_mul_root.py** for all arithmetic operations in the calculator.

4.4. Integer Triangles

1. Given the perimeter of a triangle (it should be an integer) write a C program to find all the possible triangles with integer sides. You just have to print the lengths of the sides of each such triangle.

2. Create a GUI application in Python for the previous problem.