**SEM II**

C Practicals Name: **Chandan Kumar Mahato**

Roll. No: **24237765063**

Group: **D**

**Practical 1 : WAP in C to Calculate the roots of a Quadratic Equation**

1. **x2-x-9=0**
2. **2x2-3x+9=0**

**Code:**

#include<stdio.h>

#include<math.h>

int main()

{

float a,b,c,d,x1,x2,x,xr,xim;

printf("Enter the cooefficients of ax^2+bx+c \n");

scanf("%f%f%f",&a,&b,&c);

d=b\*b-4\*a\*c;

float e = sqrt(d);

float f = sqrt(-d);

if(d>0)

{

printf("Real and distinct roots\n");

x1=(-b-e)/(2\*a);

x2=(-b+e)/(2\*a);

printf("Roots are x1=%f, x2=%f\n", x1,x2);

}

else if (d==0)

{

printf("Roots are equal\n");

x=-b/(2\*a);

printf("Root is x=%f\n", x);

}

else

{

printf("Roots are imaginary\n");

xr=-b/(2\*a);

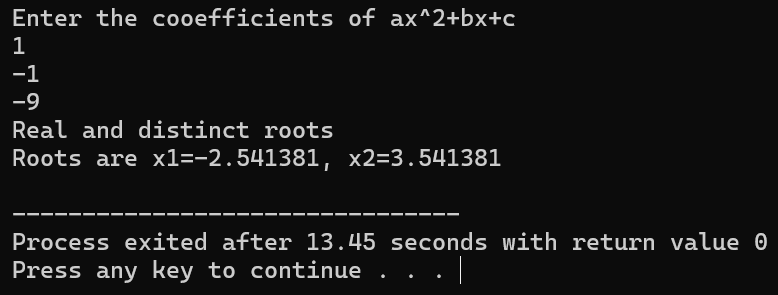
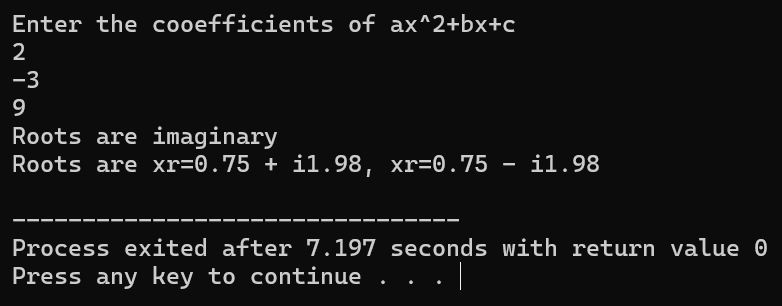
xim=f/(2\*a);

printf("Roots are xr=%0.2f + i%0.2f, xr=%0.2f - i%0.2f\n",xr, xim, xr, xim);

}

return 0;

}

**Output:**

**Practical 2 : WAP in C to find the real roots of the following equations using Newton Raphson Method**

1. **f(x)= x3+x2+x+10**
2. **f(x)= x4-x3+x-1**

**Code:**

**i)**

#include<stdio.h>

#include<math.h>

float f1(float x)

{

return x\*x\*x + x\*x + x + 10;

}

float f11(float x)

{

return 3\*x\*x + 2\*x + 1;

}

int main()

{

float xo,x1;

printf("Enter the initial guess, x0:\n");

scanf("%f",&xo);

do

{

x1=xo-f1(xo)/f11(xo);

if(fabs(x1-xo)<0.0000001)

{

break;

}

xo=x1;

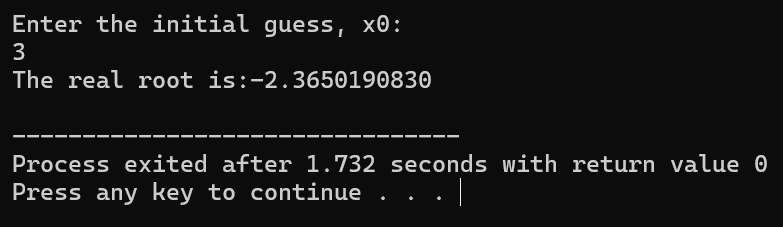
}

while(1);

printf("The real root is:%.10f\n",x1);

return 0;

}

**Output:**

**ii)**

#include<stdio.h>

#include<math.h>

float f1(float x)

{

return x\*x\*x\*x - x\*x\*x + x - 1;

}

float f11(float x)

{

return 4\*x\*x\*x - 3\*x\*x + 1;

}

int main()

{

float xo,x1;

printf("Enter the initial guess, x0:\n");

scanf("%f",&xo);

do

{

x1=xo-f1(xo)/f11(xo);

if(fabs(x1-xo)<0.0000001)

{

break;

}

xo=x1;

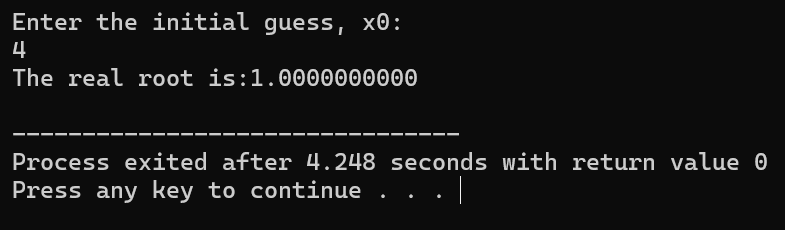
}

while(1);

printf("The real root is:%.10f\n",x1);

return 0;

}

**Output:**

**Practical 3 : WAP in C to find the 95% and 99% Confidence Interval for population mean when population standard deviation is known.**

**Code:**

#include<stdio.h>

#include<math.h>

int main()

{

int n,i;

float s\_mean=0,sd,z1,z2,x=0;

float lower\_limit1,upper\_limit1,lower\_limit2,upper\_limit2;

sd=3;

z1=1.96;

z2=2.58;

printf("Enter the sample size:\n");

scanf("%d",&n);

float arr[]={72,69,71,70,62,64,67,69,73,82,75,70,69,78,73,70,91,59,85,74};

for(i=0;i<n;i++)

{

x=x+arr[i];

}

s\_mean=x/n;

printf("Sample Mean is:%f\n",s\_mean);

lower\_limit1 = s\_mean - z1 \* (sd / sqrt(n));

upper\_limit1 = s\_mean + z1 \* (sd / sqrt(n));

lower\_limit2 = s\_mean - z2 \* (sd / sqrt(n));

upper\_limit2 = s\_mean + z2 \* (sd / sqrt(n));

printf("The 95%% Confidence Interval for mean is: (%f, %f)\n", lower\_limit1, upper\_limit1);

printf("The 99%% Confidence Interval for mean is: (%f, %f)\n", lower\_limit2, upper\_limit2);

printf("For 95% CI:\n");

if(lower\_limit1<75 && upper\_limit1>75)

{

printf("It is resonable to conclude that the mean exam score is 75\n");

}

else

{

printf("It is not resonable to conclude that the mean exam score is 75\n");

}

printf("For 99% CI:\n");

if(lower\_limit2<75 && upper\_limit2>75)

{

printf("It is resonable to conclude that the mean exam score is 75\n");

}

else

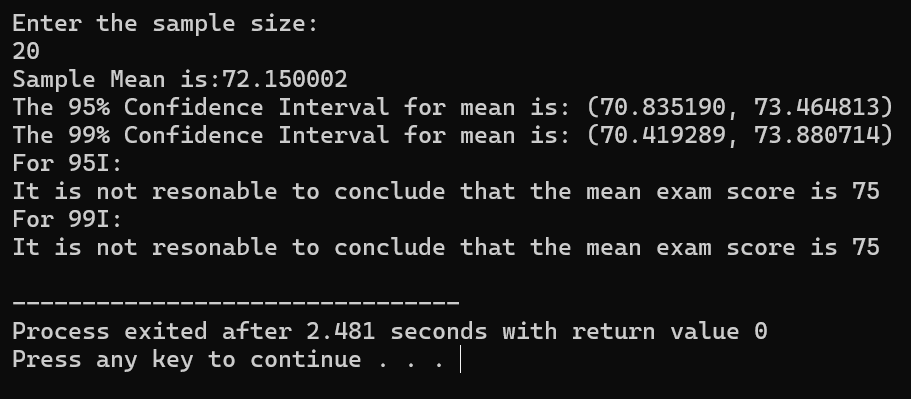
{

printf("It is not resonable to conclude that the mean exam score is 75\n");

}

}

**Output:**



**Practical 4 : WAP in C to find the 95% and 99% Confidence Interval for population mean when population standard deviation is unknown.**

**Code:**

#include<stdio.h>

#include<math.h>

int main()

{

int n,i;

float x=0,sum=0,sumsq=0,sample\_mean,sample\_var,s,t1,t2;

float lower\_limit1,upper\_limit1,lower\_limit2,upper\_limit2;

t1=2.16;

t2=3.012;

printf("Enter the number of samples:\n");

scanf("%d",&n);

float arr[]={16,18,20,34,26,22,28,32,21,20,14,30,35,25};

for(i=0;i<n;i++)

{

sum=sum+arr[i];

sumsq=sumsq+arr[i]\*arr[i];

}

sample\_mean=sum/n;

sample\_var=(sumsq/n)-(sample\_mean\*sample\_mean);

s=sqrt(sample\_var);

lower\_limit1=sample\_mean-(t1\*s/sqrt(n-1));

upper\_limit1=sample\_mean+(t1\*s/sqrt(n-1));

lower\_limit2=sample\_mean-(t2\*s/sqrt(n-1));

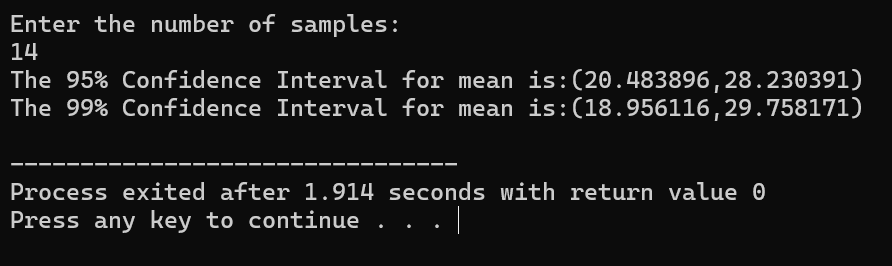
upper\_limit2=sample\_mean+(t2\*s/sqrt(n-1));

printf("The 95%% Confidence Interval for mean is:(%f,%f)\n",lower\_limit1,upper\_limit1);

printf("The 99%% Confidence Interval for mean is:(%f,%f)\n",lower\_limit2,upper\_limit2);

}

**Output:**



**Practical 5 : WAP in C to find the 95% Confidence Interval for population mean and Variance of Normal Population and coverage probability to Confidence Interval.**

**Code:**

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

int main()

{

int i,j,k;

float mu,sigma2,z,mean,var,x,countmu=0,countsigma2=0;

float lower\_limit\_mu[500],upper\_limit\_mu[500],lower\_limit\_sigma2[500],upper\_limit\_sigma2[500];

float sum\_lower\_limit\_mu=0, sum\_upper\_limit\_mu=0, sum\_lower\_limit\_sigma2=0, sum\_upper\_limit\_sigma2=0;

printf("Enter the value of populatin mean:\n");

scanf("%f",&mu);

printf("Enter the value of populatin variance:\n");

scanf("%f",&sigma2);

for(k=0;k<200;k++)

{

float sum=0,sum\_sq=0;

for(j=0;j<25;j++)

{

float u=0;

for(i=0;i<500;i++)

{

u=u+rand()/(1.0+RAND\_MAX);

}

z=(u-(500.0/2))/sqrt(500.0/12);

x=mu+z\*sqrt(sigma2);

sum=sum+x;

sum\_sq=sum\_sq+x\*x;

}

mean=sum/25;

var=sum\_sq/25-mean\*mean;

lower\_limit\_mu[k]=mean-1.96\*sqrt(var/25);

upper\_limit\_mu[k]=mean+1.96\*sqrt(var/25);

lower\_limit\_sigma2[k]=25\*var/39.364;

upper\_limit\_sigma2[k]=25\*var/12.401;

sum\_lower\_limit\_mu+=lower\_limit\_mu[k];

sum\_upper\_limit\_mu+=upper\_limit\_mu[k];

sum\_lower\_limit\_sigma2+=lower\_limit\_sigma2[k];

sum\_upper\_limit\_sigma2+=upper\_limit\_sigma2[k];

if(lower\_limit\_mu[k] < mu && upper\_limit\_mu[k] > mu)

{

countmu+=1;

}

if(lower\_limit\_sigma2[k] < sigma2 && upper\_limit\_sigma2[k] > sigma2)

{

countsigma2+=1;

}

}

printf("The 95%% CI for population mean is: (%f,%f)\n", sum\_lower\_limit\_mu/200, sum\_upper\_limit\_mu/200);

printf("The 95%% CI for population sigma2 is: (%f,%f)\n", sum\_lower\_limit\_sigma2/200, sum\_upper\_limit\_sigma2/200);

float coveragemu=countmu/200;

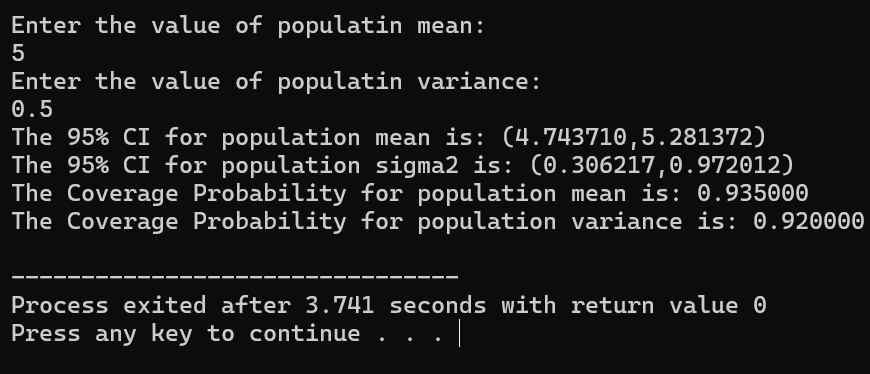
float coveragesigma2=countsigma2/200;

printf("The Coverage Probability for population mean is: %f\n", coveragemu);

printf("The Coverage Probability for population variance is: %f\n", coveragesigma2);

}

**Output:**

****

**Practical 6 : WAP in C to find the derivative of a given functions:**

1. **f(x)=5x + x2, at x=2**
2. **g(x) = 6x4 - 2x3 + x - 1, at x=3**
3. **h(x) = xex2**

**Code:**

#include<stdio.h>

#include<math.h>

float fx(float x1)

{

return(5\*x1 + pow(x1,2));

}

float gx(float x2)

{

return(6\*pow(x2,4) - 2\*pow(x2,3) + x2 -1);

}

float hx(float x3)

{

return(x3\*exp(x3\*x3));

}

int main()

{

float x1,x2,x3,d1,d2,d3,h;

h=0.00001;

printf("Enter the value of x1:\n");

scanf("%f",&x1);

d1=(fx(x1+h)-fx(x1))/h;

printf("f'(x)=%f\n",d1);

printf("Enter the value of x2:\n");

scanf("%f",&x2);

d2=(gx(x2+h)-gx(x2))/h;

printf("g'(x)=%f\n",d2);

printf("Enter the value of x3:\n");

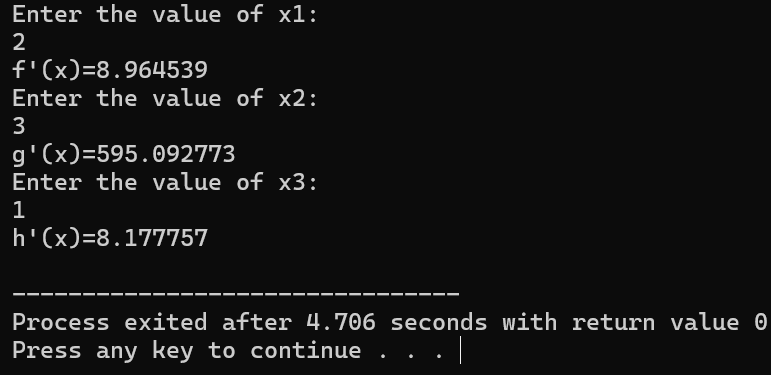
scanf("%f",&x3);

d3=(hx(x3+h)-hx(x3))/h;

printf("h'(x)=%f\n",d3);

}

**Output:**



**Practical 7 : WAP in C to evaluate the appropriate integral using Trapezoidal Rule.**

1. **, (ii) , (iii)**

**Code:**

#include<stdio.h>

#include<math.h>

float f1(float x)

{

return(exp(-(x\*x)/2));

}

float f2(float x)

{

return(sqrt(1+x\*x));

}

float f3(float x)

{

return(pow(cos(x),2)\*sqrt(1+x\*x\*x));

}

int main()

{

int n,i,a1,b1,a2,b2,a3,b3;

float h1,h2,h3,I1,I2,I3;

float sum1=0,sum2=0,sum3=0;

printf("Enter the number of subintervals:\n");

scanf("%d", &n);

a1=0,b1=2,a2=1,b2=6,a3=-1,b3=2;

h1=(float)(b1-a1)/n;

h2=(float)(b2-a2)/n;

h3=(float)(b3-a3)/n;

for(i=1;i<n;i++)

{

sum1+=2\*f1(a1+i\*h1);

sum2+=2\*f2(a2+i\*h2);

sum3+=2\*f3(a3+i\*h3);

}

I1=(h1/2)\*(f1(a1)+f1(b1)+sum1);

printf("The value of the first integral is:%f\n",I1);

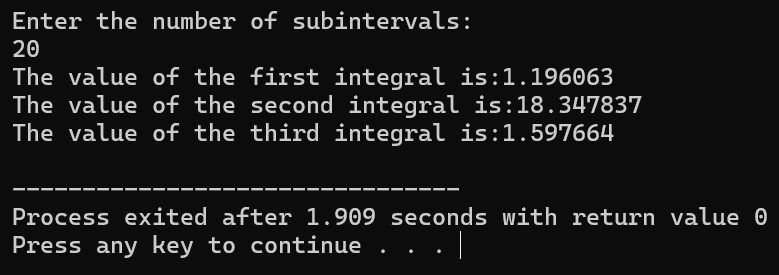
I2=(h2/2)\*(f2(a2)+f2(b2)+sum2);

printf("The value of the second integral is:%f\n",I2);

I3=(h3/2)\*(f3(a3)+f3(b3)+sum3);

printf("The value of the third integral is:%f\n",I3);

}

**Output:**

**Practical 8 : WAP in C to calculate the Sum of two matrices.**

**Code:**

#include<stdio.h>

#include<math.h>

int main()

{

int i,j,m,n,p,q;

int A[10][10], B[10][10],C[10][10];

printf("Enter the number of rows in 1st matrix:\n");

scanf("%d",&m);

printf("Enter the number of columns in 1st matrix:\n");

scanf("%d",&n);

printf("Enter the number of rows in 2nd matrix:\n");

scanf("%d",&p);

printf("Enter the number of columns in 2nd matrix:\n");

scanf("%d",&q);

if(m!=p && n!=q)

{

printf("Matrix addition is not possible.");

}

else

{

printf("Enter the elements of 1st matrix:\n");

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

scanf("%d",&A[i][j]);

}

}

printf("Enter the elements of 2nd matrix:\n");

for(i=0;i<p;i++)

{

for(j=0;j<q;j++)

{

scanf("%d",&B[i][j]);

}

}

printf("The resultant matrix is:\n");

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

C[i][j]=A[i][j]+B[i][j];

printf("%d\t",C[i][j]);

}

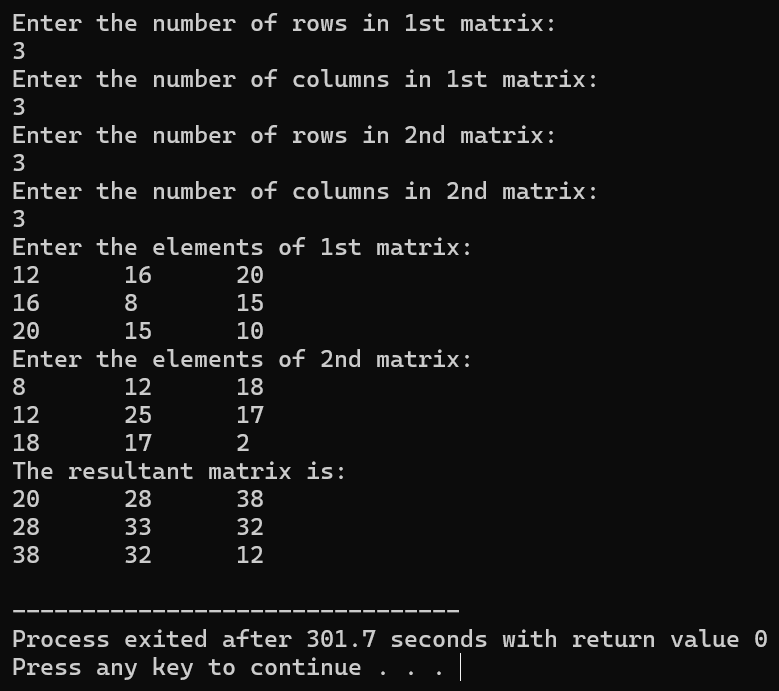
printf("\n");

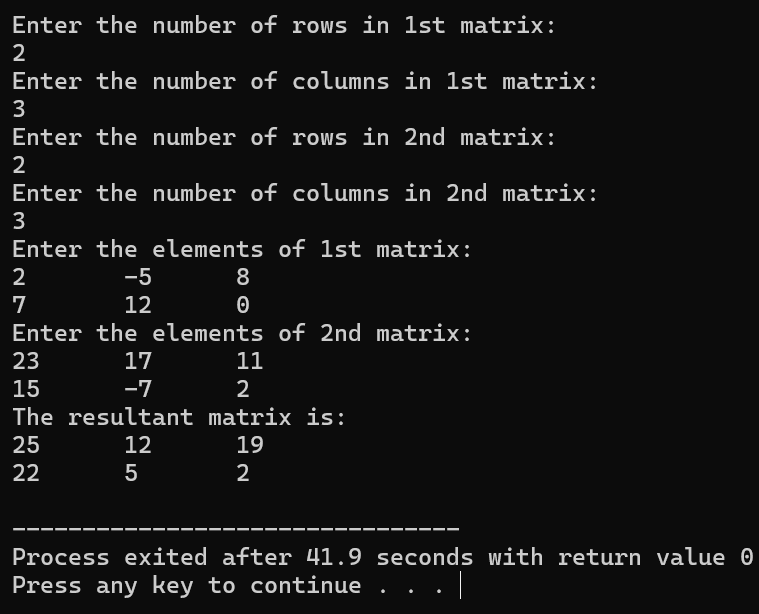
}

}

}

**Output:**





**Practical 9 : WAP in C to calculate the Product of two matrices.**

**Code:**

#include<stdio.h>

#include<math.h>

int main()

{

int i,j,k,m,n,p,q,s=0;

int A[10][10], B[10][10],C[10][10];

printf("Enter the number of rows in 1st matrix:\n");

scanf("%d",&m);

printf("Enter the number of columns in 1st matrix:\n");

scanf("%d",&n);

printf("Enter the number of rows in 2nd matrix:\n");

scanf("%d",&p);

printf("Enter the number of columns in 2nd matrix:\n");

scanf("%d",&q);

if(n!=p)

{

printf("Matrix Multiplication is not possible.");

}

else

{

printf("Enter the elements of 1st matrix:\n");

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

scanf("%d",&A[i][j]);

}

}

printf("Enter the elements of 2nd matrix:\n");

for(i=0;i<p;i++)

{

for(j=0;j<q;j++)

{

scanf("%d",&B[i][j]);

}

}

for(i=0;i<m;i++)

{

for(j=0;j<q;j++)

{

s=0;

for(k=0;k<p;k++)

{

s+=A[i][k]\*B[k][j];

}

C[i][j]=s;

}

}

printf("The resultant Matrix is:\n");

for(i=0;i<m;i++)

{

for(j=0;j<q;j++)

{

printf("%d\t",C[i][j]);

}

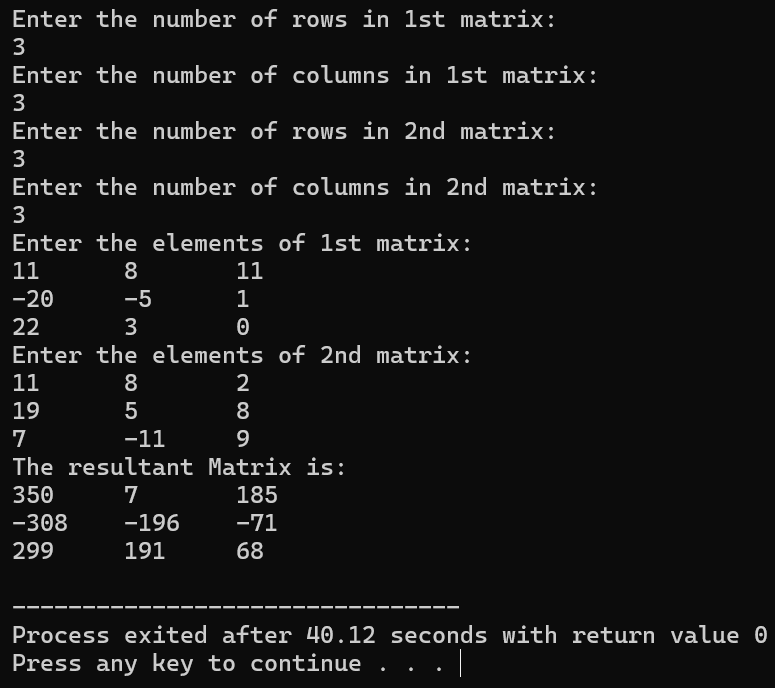
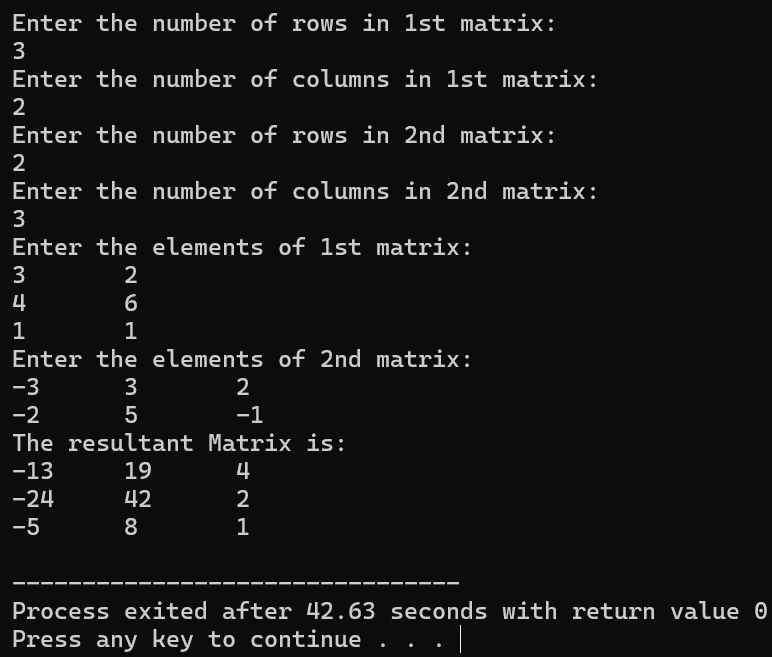
printf("\n");

}

}

}

**Output:**



**Practical 10 : WAP in C to find the Determinant of a Square Matrix.**

**Code:**

#include<stdio.h>

#include<math.h>

int main()

{

int n,i,j,k,s=0,count;

float A[10][10],det,f,temp;

printf("Enter the one dimension of the square matrix:\n");

scanf("%d",&n);

printf("Enter the elements of A are:\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%f",&A[i][j]);

}

}

det=1;

count=0;

for(j=0;j<n;j++)

{

if(A[j][j] == 0)

{

s=j+1;

while(s<n && A[s][j]==0)

{

s=s+1;

}

if(s==n)

{

printf("A is Singular Matrix and |A| = 0");

return 0;

}

for(k=0;k<n;k++)

{

temp=A[j][k];

A[j][k]=A[s][k];

A[s][k]=temp;

}

count+=1;

}

for(i=j+1;i<n;i++)

{

f=A[i][j]/A[j][j];

for(k=0;k<n;k++)

{

A[i][k]=A[i][k]-f\*A[j][k];

}

}

}

for(i=0;i<n;i++)

{

det\*=A[i][i];

}

if(count%2!=0)

{

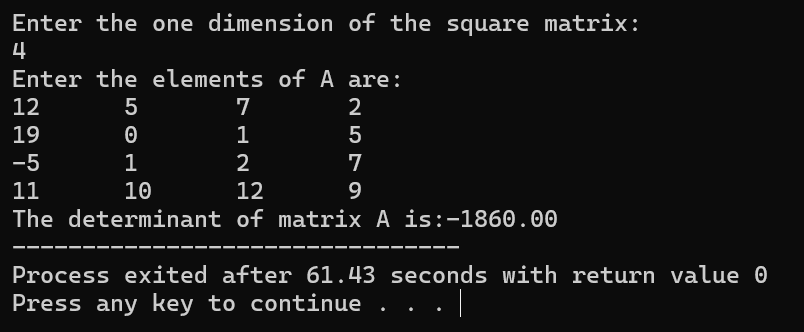
det=-det;

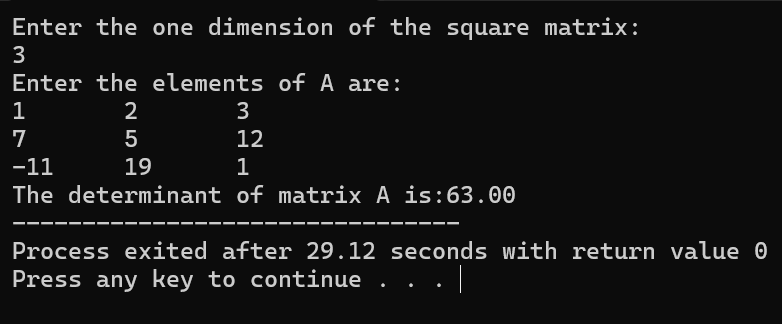
}

printf("The determinant of matrix A is:%0.2f",det);

}

**Output:**





**Practical 11 : WAP in C to find the Determinant of a Square Matrix.**

**Code:**

#include<stdio.h>

#include<math.h>

int main()

{

int n,i,j,k;

float A[100][100],I[100][100],var,f;

printf("Enter the dimension of matrix A:\n");

scanf("%d",&n);

printf("Enter the elements of A:\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%f",&A[i][j]);

if(i==j)

{

I[i][j]=1;

}

else

{

I[i][j]=0;

}

}

}

for(i=0;i<n;i++)

{

var=A[i][i];

for(j=0;j<n;j++)

{

A[i][j]=A[i][j]/var;

I[i][j]=I[i][j]/var;

}

for(k=0;k<n;k++)

{

if(k!=i)

{

f=A[k][i];

for(j=0;j<n;j++)

{

A[k][j]=A[k][j]-f\*A[i][j];

I[k][j]=I[k][j]-f\*I[i][j];

}

}

}

}

printf("The Inverse of Matrix A is:\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

printf("%0.3f\t", I[i][j]);

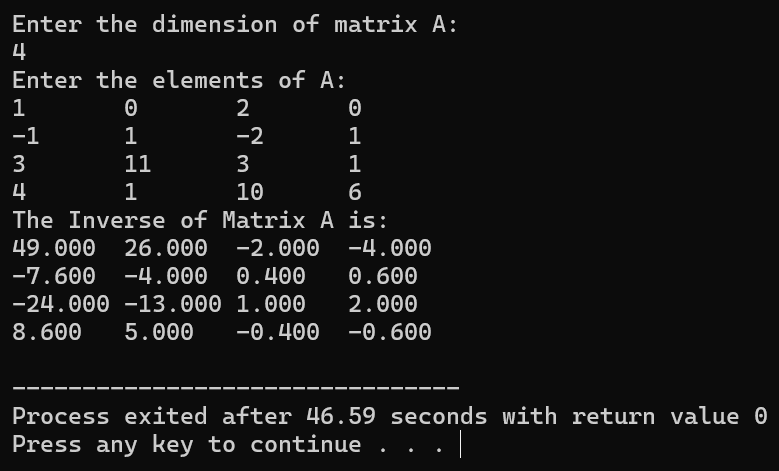
}

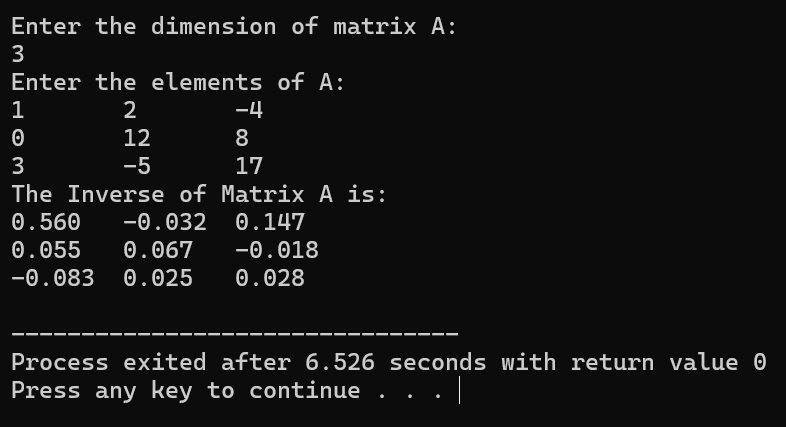
printf("\n");

}

}

**Output:**





**Practical 12 : WAP in C to verify the Cayley Hamilton theorem of the matrix.**

**Code:**

#include<stdio.h>

#include<math.h>

void matrix\_multiplication(float A[100][100],int m,int n,float B[100][100],int p,int q,float C[100][100])

{

int i,j,k;

if(n!=p)

{

printf("Matrix Multiplication is not possible");

}

else

{

for(i=0;i<m;i++)

{

for(j=0;j<q;j++)

{

C[i][j]=0;

for(k=0;k<p;k++)

{

C[i][j]+=A[i][k]\*B[k][j];

}

}

}

}

}

int main()

{

int n,i,j,k,count=0;

float A[100][100],A2[100][100],A3[100][100],A4[100][100],I[100][100],sol[100][100];

printf("Enter the order of the matrix A:\n");

scanf("%d",&n);

printf("Enter the elements of Matrix A:\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%f",&A[i][j]);

if(i==j)

{

I[i][j]=1;

}

else

{

I[i][j]=0;

}

}

}

matrix\_multiplication(A,n,n,A,n,n,A2);

matrix\_multiplication(A,n,n,A2,n,n,A3);

matrix\_multiplication(A2,n,n,A2,n,n,A4);

printf("\nMatrix (A^4-8A^3+21A^2-20A+5I)\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

sol[i][j]=A4[i][j]-8\*A3[i][j]+21\*A2[i][j]-20\*A[i][j]+5\*I[i][j];

if(sol[i][j]==0)

{

count+=1;

printf("%f\t",sol[i][j]);

}

}

printf("\n");

}

if(count==n\*n)

{

printf("Cayley Hamilton Theorem has been verified");

}

else

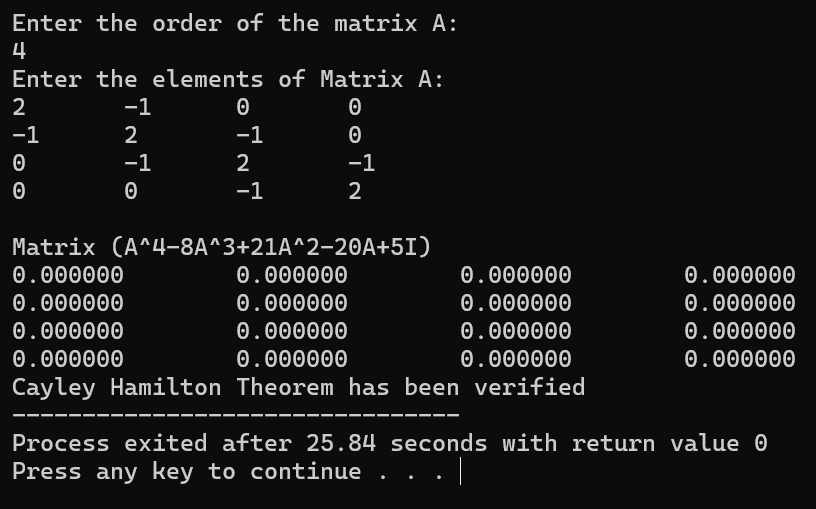
{

printf("Cayley Hamilton Theorem has not been verified");

}

}

**Output:**



**Practical 13 : WAP in C to solve the System of Linear Equation using Gauss Elimination Method.**

**Code:**

#include<stdio.h>

#include<math.h>

int main()

{

int n,i,j,k,s=0,count;

float A[10][10],x[10],det,f,temp;

printf("Enter the one dimension of the square matrix:\n");

scanf("%d",&n);

printf("Enter the elements of augmented matrix A|b:\n");

for(i=0;i<n;i++)

{

for(j=0;j<=n;j++)

{

scanf("%f",&A[i][j]);

}

}

det=1;

count=0;

for(j=0;j<n;j++)

{

if(fabs(A[j][j]) < 1e-6)

{

s=j+1;

while(s<n && fabs(A[s][j]) < 1e-6)

{

s=s+1;

}

if(s==n)

{

printf("A is Singular Matrix and |A| = 0");

return 0;

}

for(k=0;k<n;k++)

{

temp=A[j][k];

A[j][k]=A[s][k];

A[s][k]=temp;

}

count+=1;

}

for(i=j+1;i<n;i++)

{

f=A[i][j]/A[j][j];

for(k=0;k<=n;k++)

{

A[i][k]=A[i][k]-f\*A[j][k];

}

}

}

// Back Substitution

for(i=n-1; i>=0; i--)

{

x[i]=A[i][n];

for(j=i+1; j<n; j++)

{

x[i]=x[i]-A[i][j]\*x[j];

}

x[i]=x[i]/A[i][i];

}

printf("\nThe solution of the system is:\n");

for(i=0; i<n; i++)

{

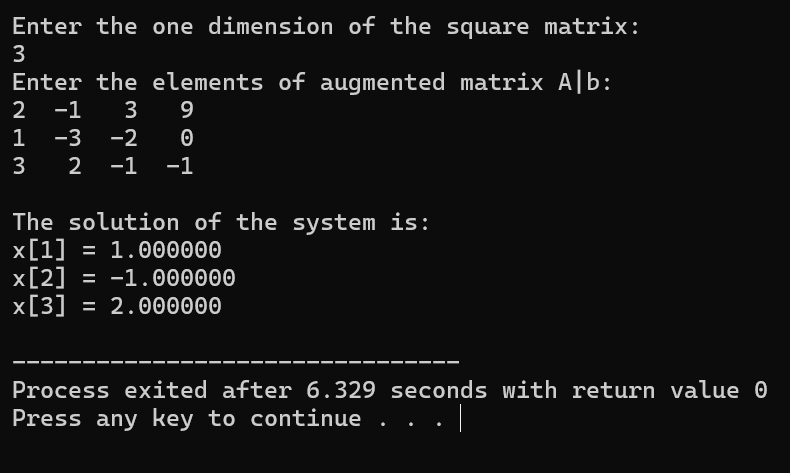
printf("x[%d] = %f\n",i+1,x[i]);

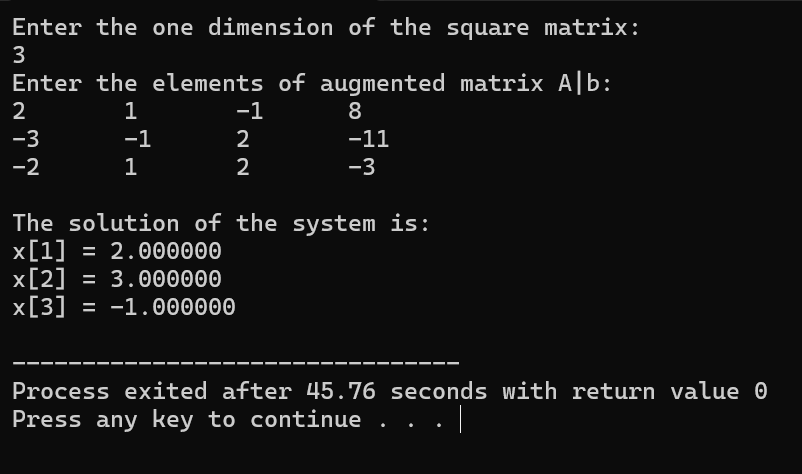
}

return 0;

}

**Output:**





**Practical 14 : WAP in C to calculate the Transition Probability Matrix and determine the number of students doing Mathematics and English work over subsequent periods using Multiplication.**

**Code:**

#include<stdio.h>

#include<math.h>

void matrix\_multiplication(float A[1][2],int m,int n,float B[2][2],int p,int q,float C[1][2])

{

int i,j,k;

for(i=0;i<m;i++)

{

for(j=0;j<q;j++)

{

C[i][j]=0;

for(k=0;k<p;k++)

{

C[i][j]+=A[i][k]\*B[k][j];

}

}

}

}

int main()

{

int i,j;

float TPM[2][2],ISV[1][2],S\_after[1][2],S\_temp[1][2];

printf("Enter the elements of the Transition Probability Matrix:\n");

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

scanf("%f",&TPM[i][j]);

}

}

printf("Enter the Initial State Vector:\n");

for(i=0;i<1;i++)

{

for(j=0;j<2;j++)

{

scanf("%f",&ISV[i][j]);

}

}

matrix\_multiplication(ISV,1,2,TPM,2,2,S\_temp);

matrix\_multiplication(S\_temp,1,2,TPM,2,2,S\_after);

printf("The Transition Probability Matrix is:\n");

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

printf("%f\t",TPM[i][j]);

}

printf("\n");

}

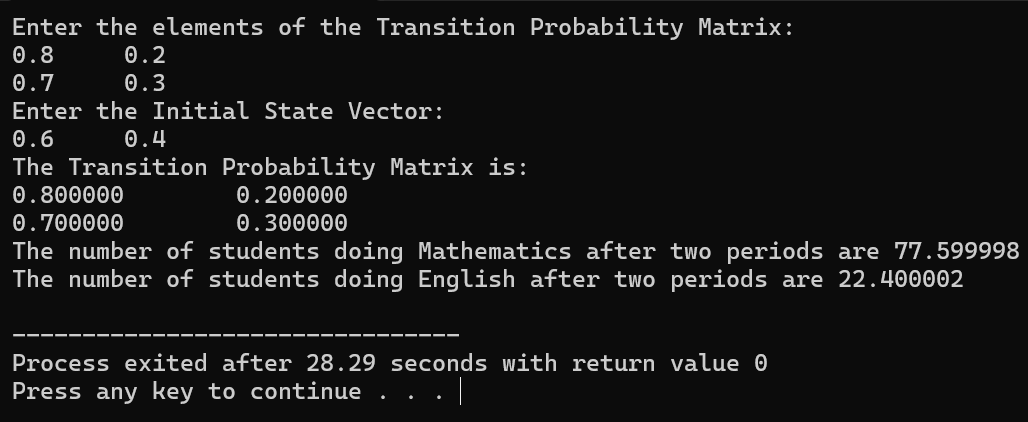
printf("The number of students doing Mathematics after two periods are %f\n", 100\*S\_after[0][0]);

printf("The number of students doing English after two periods are %f\n", 100\*S\_after[0][1]);

return 0;

}

**Output:**

****

**Practical 15 : WAP in C to calculate the Dominant Eigen Value using Power Method.**

**Code:**

#include<stdio.h>

#include<math.h>

int n;

void mat\_vec\_mult(float mat[100][100],float vec[100],float res[100])

{

int i,j;

for(i=0;i<n;i++)

{

res[i]=0;

for(j=0;j<n;j++)

{

res[i]+=mat[i][j]\*vec[j];

}

}

}

float vec\_mag(float vec[100])

{

int i;

float sum=0;

for(i=0;i<n;i++)

{

sum+=vec[i]\*vec[i];

}

return(sqrt(sum));

}

void norm\_vec(float vec[100])

{

float mag=vec\_mag(vec);

for(int i=0;i<n;i++)

{

vec[i]=vec[i]/mag;

}

}

float dom\_eig\_val(float mat[100][100],float ini\_vec[100],float eig\_vec[100])

{

float new\_vec[100];

float eig\_val=0;

for(int i=0;i<n;i++)

{

new\_vec[i]=ini\_vec[i];

}

for(int iter=0;iter<1000;iter++)

{

mat\_vec\_mult(mat,new\_vec,eig\_vec);

eig\_val=vec\_mag(eig\_vec);

norm\_vec(eig\_vec);

float error=0;

for(int i=0;i<n;i++)

{

error+=fabs(new\_vec[i]-eig\_vec[i]);

}

if(error<0.0000001)

{

break;

}

for(int i=0;i<n;i++)

{

new\_vec[i]=eig\_vec[i];

}

}

return eig\_val;

}

int main()

{

int i,j;

float mat[100][100],ini\_vec[100],eig\_vec[100],eig\_val;

printf("Enter the size of the Matrix:\n");

scanf("%d",&n);

printf("Enter the elements of the Matrix:\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%f",&mat[i][j]);

}

}

printf("Enter the elements of the Initial Vector:\n");

for(i=0;i<n;i++)

{

scanf("%f",&ini\_vec[i]);

eig\_vec[i]=0;

}

eig\_val=dom\_eig\_val(mat,ini\_vec,eig\_vec);

printf("The Dominant Eigen Value is:%f\n",eig\_val);

printf("The corresponding Eigen Vector is:\n");

for(i=0;i<n;i++)

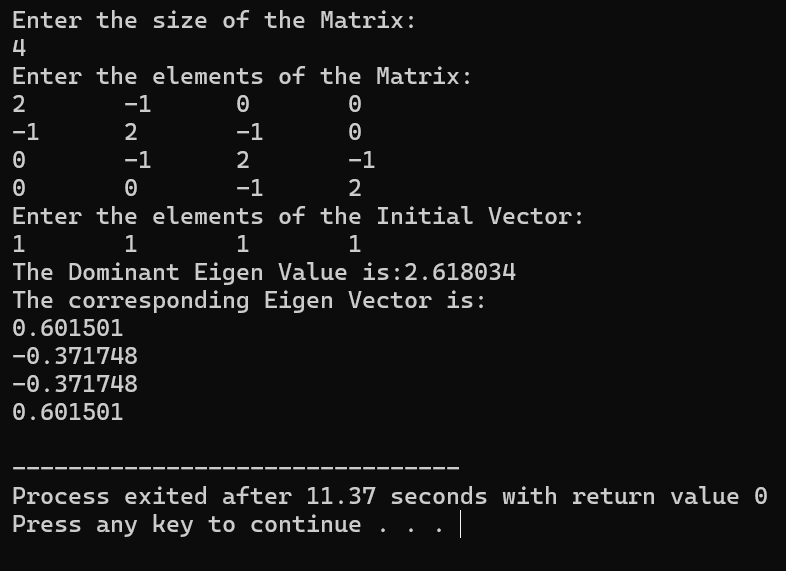
{

printf("%f\n",eig\_vec[i]);

}

return 0;

}

**Output:**

**Practical 16 : WAP in C to find the Generalised Inverse of a matrix.**

**Code:**

#include<stdio.h>

#include<math.h>

int main()

{

int m,n,i,j;

float A[100][100],B[2][2],B\_det,B\_inv[2][2],G[100][100];

printf("Enter the number of rows of Matrix A:\n");

scanf("%d",&m);

printf("Enter the number of columns of Matrix A:\n");

scanf("%d",&n);

printf("Enter the elements of Matrix A:\n");

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

scanf("%f",&A[i][j]);

}

}

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

B[i][j]=A[i][j];

}

}

B\_det=B[0][0]\*B[1][1]-B[1][0]\*B[0][1];

B\_inv[0][0]=(B[1][1]/B\_det);

B\_inv[1][0]=-(B[1][0]/B\_det);

B\_inv[0][1]=-(B[0][1]/B\_det);

B\_inv[1][1]=(B[0][0]/B\_det);

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

G[i][j]=0.0;

}

}

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

G[i][j]=B\_inv[i][j];

}

}

printf("The Generalised Inverse of A is:\n");

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

printf("%0.3f\t",G[i][j]);

}

printf("\n");

}

return 0;

}

**Output:**

