LA Assignment

C with openMP

Rahul Raman , Danish

PES1201800146  PES

**1.Gaussian Elimination**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include<omp.h>

double time\_elapsed(struct timespec start, struct timespec end) {

    double t;

    t = (end.tv\_sec - start.tv\_sec);

    t += (end.tv\_nsec - start.tv\_nsec) \* 0.000000001;

    return t;

}

int forward\_elimination(int n,float a[n][n+1]){

    for(int i=0;i<n-1;i++){

        //#pragma omp parallel for shared(a) private(l,i,k)

        if(a[i][i]==0){

            for(int m=i+1;m<n;m++){

                if(a[m][i]!=0){

                    for(int b=0;b<n+1;b++){

                        float temp= a[i][b];

                        a[i][b] = a[m][b];

                        a[m][b] = temp;

                    }

                    break;

                }

            }

            if(a[i][i]==0) return 0;

        }

        for(int k=1;k<n-i;k++){

            float l = a[i+k][i]/a[i][i];

            for(int j=0;j<n+1;j++){

                a[i+k][j] -= l\*a[i][j];

            }

        }

    }

    /\*

    printf("\nFinal Matrix\n");

   for(int i=0;i<n;i++){

        for(int j=0;j<n+1;j++){

            printf("%f ",a[i][j]);

        }

        printf("\n");

    } \*/

    return 1;

}

int forward\_elimination\_p(int n,float a[n][n+1]){

    for(int i=0;i<n-1;i++){

        #pragma omp parallel for shared(a) private(l,i,k)

        if(a[i][i]==0){

            for(int m=i+1;m<n;m++){

                if(a[m][i]!=0){

                    for(int b=0;b<n+1;b++){

                        float temp= a[i][b];

                        a[i][b] = a[m][b];

                        a[m][b] = temp;

                    }

                    break;

                }

            }

            if(a[i][i]==0) return 0;

        }

        for(int k=1;k<n-i;k++){

            float l = a[i+k][i]/a[i][i];

            for(int j=0;j<n+1;j++){

                a[i+k][j] -= l\*a[i][j];

            }

        }

    }

    /\*

    printf("\nFinal Matrix\n");

   for(int i=0;i<n;i++){

        for(int j=0;j<n+1;j++){

            printf("%f ",a[i][j]);

        }

        printf("\n");

    } \*/

    return 1;

}

void back\_substitution(int n,float a[n][n+1]){

    float sol[n];

    for (int i = n-1; i >= 0; i--)

    {

        sol[i] = a[i][n];

        for (int j=i+1; j<n; j++)

            sol[i] -= a[i][j]\*sol[j];

        sol[i] = sol[i]/a[i][i];

    }

    printf("\nSolution for the system:\n");

    for (int i=0; i<n; i++)

        printf("%f\n", sol[i]);

}

void gauss\_elimination(int n,float a[n][n+1]){

    int res = forward\_elimination(n,a);

    if(res == 0) printf("\nSingular");

    else{

        int lhs=0;

        int rhs = 1 && a[n-1][n];

        for(int z=0;z<n;z++){

            lhs = lhs || (int)a[n-1][z];

        }

        if(lhs==0 && rhs==0) printf("\nSingular and Infinitely many Solution");

        else if(lhs==0 && rhs!=0) printf("\nSingular and No Solution");

        else {

            back\_substitution(n,a);

            printf("\nNon Singular and unique solution");

        }

    }

}

void gauss\_elimination\_p(int n,float a[n][n+1]){

    int res = forward\_elimination\_p(n,a);

    if(res == 0) printf("\nSingular");

    else{

        int lhs=0;

        int rhs = 1 && a[n-1][n];

        for(int z=0;z<n;z++){

            lhs = lhs || (int)a[n-1][z];

        }

        if(lhs==0 && rhs==0) printf("\nSingular and Infinitely many Solution");

        else if(lhs==0 && rhs!=0) printf("\nSingular and No Solution");

        else {

            back\_substitution(n,a);

            printf("\nNon Singular and unique solution");

        }

    }

}

int main(){

    int n;

    printf("Enter the number of unknown variables:");

    scanf("%d",&n);

    float a[n][n+1];

    printf("Enter the elements:\n");

    for(int i=0;i<n;i++){

        for(int j=0;j<n+1;j++){

            scanf("%f",&a[i][j]);

        }

    }

   /\*for(int i=0;i<n;i++){

        for(int j=0;j<n+1;j++){

            printf("%f ",a[i][j]);

        }

        printf("\n");

    }

    \*/

   struct timespec start, end;

   clock\_gettime(CLOCK\_REALTIME, &start);

    gauss\_elimination(n,a);

    clock\_gettime(CLOCK\_REALTIME, &end);

    printf("\nTime spent on gaussian\_elimination for %d variables sequentially: %lf\n",n,

            time\_elapsed(start, end));

    clock\_gettime(CLOCK\_REALTIME, &start);

    gauss\_elimination\_p(n,a);

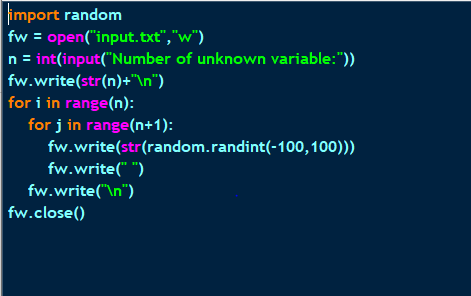
    printf("\nTime spent on gaussian\_elimination for %d variables in parallel: %lf\n",n,

            time\_elapsed(start, end));

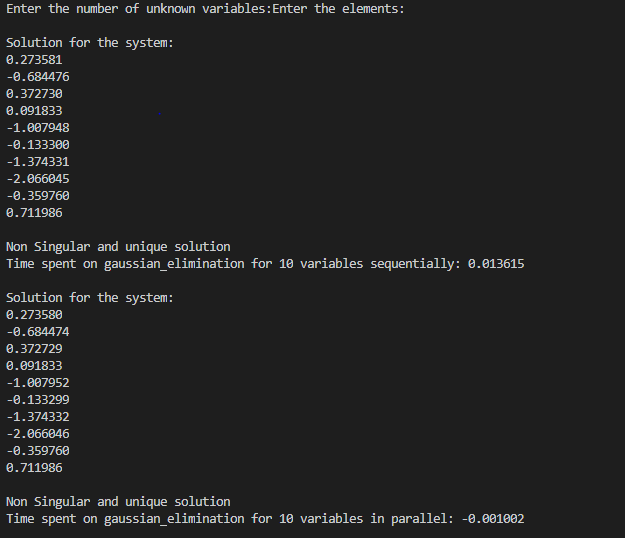
    return 0;

}

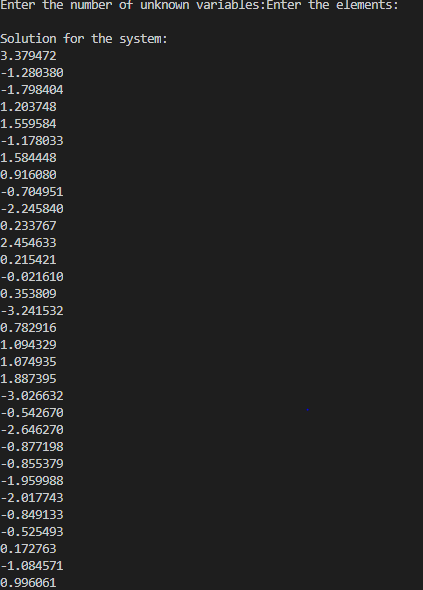
**Generating test case:**

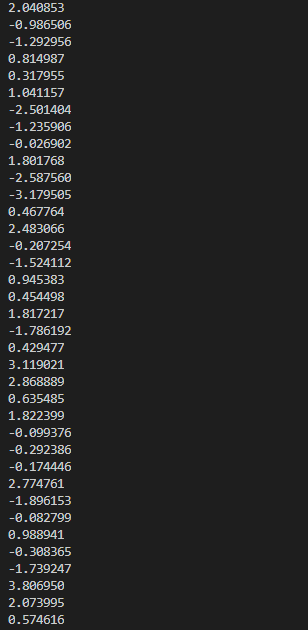


**Output : For 10 variables**



**For 100 variables :**









**2. The LU Decomposition**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

double time\_elapsed(struct timespec start, struct timespec end) {

    double t;

    t = (end.tv\_sec - start.tv\_sec);

    t += (end.tv\_nsec - start.tv\_nsec) \* 0.000000001;

    return t;

}

int lu(int n,float a[n][n+1]){

    float L[n][n];

    for (int i=0;i<n;i++){

        for(int j=0;j<n;j++){

            if(i==j) L[i][j]=1.0f;

            else L[i][j]=0.0f;

        }

    }

    for(int i=0;i<n-1;i++){

        if(a[i][i]==0){

            for(int m=i+1;m<n;m++){

                if(a[m][i]!=0){

                    for(int b=0;b<n+1;b++){

                        float temp= a[i][b];

                        a[i][b] = a[m][b];

                        a[m][b] = temp;

                    }

                    break;

                }

            }

            if(a[i][i]==0) return 0;

        }

        for(int k=1;k<n-i;k++){

            float l = a[i+k][i]/a[i][i];

            L[i+k][i] = l;

            for(int j=0;j<n+1;j++){

                a[i+k][j] -= l\*a[i][j];

            }

        }

    }

    printf("\n L :=\n");

    for(int i=0;i<n;i++){

        for(int j=0;j<n;j++){

            printf("%f ",L[i][j]);

        }

        printf("\n");

    }

    printf("\n U :=\n");

   for(int i=0;i<n;i++){

        for(int j=0;j<n;j++){

            printf("%f ",a[i][j]);

        }

        printf("\n");

    }

    return 1;

}

/\*void back\_substitution(int n,float a[n][n+1]){

    float sol[n];

    for (int i = n-1; i >= 0; i--)

    {

        sol[i] = a[i][n];

        for (int j=i+1; j<n; j++)

            sol[i] -= a[i][j]\*sol[j];

        sol[i] = sol[i]/a[i][i];

    }

    printf("\nSolution for the system:\n");

    for (int i=0; i<n; i++)

        printf("%f\n", sol[i]);

}

void gauss\_elimination(int n,float a[n][n+1]){

    int res = forward\_elimination(n,a);

    if(res == 0) printf("\nSingular");

    else{

        int lhs=0;

        int rhs = 1 && a[n-1][n];

        for(int z=0;z<n;z++){

            lhs = lhs || (int)a[n-1][z];

        }

        if(lhs==0 && rhs==0) printf("\nSingular and Infinitely many Solution");

        else if(lhs==0 && rhs!=0) printf("\nSingular and No Solution");

        else {

            back\_substitution(n,a);

            printf("\nNon Singular and unique solution");

        }

    }

}

\*/

int main(){

    int n;

    printf("Enter the number of unknown variables:");

    scanf("%d",&n);

    float a[n][n+1];

    printf("Enter the elements:\n");

    for(int i=0;i<n;i++){

        for(int j=0;j<n+1;j++){

            scanf("%f",&a[i][j]);

        }

    }

   /\*for(int i=0;i<n;i++){

        for(int j=0;j<n+1;j++){

            printf("%f ",a[i][j]);

        }

        printf("\n");

    }

    \*/

   struct timespec start, end;

   clock\_gettime(CLOCK\_REALTIME, &start);

    lu(n,a);

    clock\_gettime(CLOCK\_REALTIME, &end);

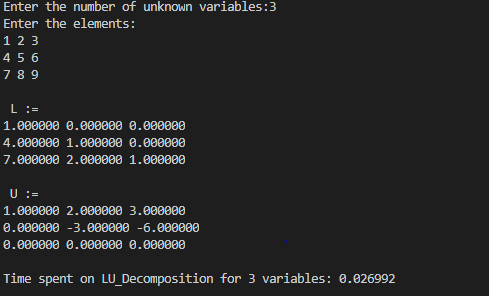
    printf("\nTime spent on LU\_Decomposition for %d variables: %lf\n",n,

            time\_elapsed(start, end));

    return 0;

}

**Output:**



**3. Inverse of a matrix using Gaussian Elimination**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int swap\_rows(int n,float a[n][2\*n],int pos)

{

    int temp[2\*n+1];

    for (int i=pos+1;i<=n;i++){

        if (a[i][i]!=0){

            for (int j=0;j<=2\*n;j++){

                temp[j] = a[i][j];

                a[i][j] = a[pos][j];

                a[pos][j] = temp[j];

            }

            return 1;

        }

    }

    return 0;

}

int main()

{

         int i,j,k,n;

         printf("Enter order of matrix: ");

         scanf("%d", &n);

         float a[n+1][2\*n+1],ratio;

         printf("Enter coefficients of Matrix:\n");

         for(i=1;i<=n;i++)

         {

              for(j=1;j<=n;j++)

              {

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

              }

         }

         for(i=1;i<=n;i++)

         {

              for(j=1;j<=n;j++)

              {

                   if(i==j)

                   {

                        a[i][j+n] = 1;

                   }

                   else

                   {

                        a[i][j+n] = 0;

                   }

              }

         }

         for(i=1;i<=n;i++)

         {

              if(a[i][i] == 0.0)

              {

                  if (swap\_rows(n,a,i))  continue;

                  printf("Mathematical Error!");

                  exit(0);

              }

              for(j=1;j<=n;j++)

              {

                   if(i!=j)

                   {

                        ratio = a[j][i]/a[i][i];

                        for(k=1;k<=2\*n;k++)

                        {

                            a[j][k] = a[j][k] - ratio\*a[i][k];

                        }

                   }

              }

         }

         for(i=1;i<=n;i++)

         {

              for(j=n+1;j<=2\*n;j++)

              {

                a[i][j] = a[i][j]/a[i][i];

              }

         }

         printf("\nInverse Matrix is:\n");

         for(i=1;i<=n;i++)

         {

              for(j=n+1;j<=2\*n;j++)

              {

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

              }

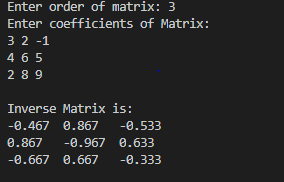
              printf("\n");

         }

         return(0);

}

**Output:**



**4. The Span of Column Space of a Matrix**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

double time\_elapsed(struct timespec start, struct timespec end) {

    double t;

    t = (end.tv\_sec - start.tv\_sec);

    t += (end.tv\_nsec - start.tv\_nsec) \* 0.000000001;

    return t;

}

int forward\_elimination(int n,int o,float a[n][o]){

    int c=0;

    for(int i=0;i<n-1;i++){

        if(a[i][i]==0){

            for(int m=i+1;m<n;m++){

                if(a[m][i]!=0){

                    for(int b=0;b<o;b++){

                        float temp= a[i][b];

                        a[i][b] = a[m][b];

                        a[m][b] = temp;

                    }

                    break;

                }

            }

            if(a[i][i]==0) return 0;

        }

        for(int k=1;k<n-i;k++){

            float l = a[i+k][i]/a[i][i];

            for(int j=0;j<o;j++){

                a[i+k][j] -= l\*a[i][j];

            }

        }

    }

    printf("\nFinal Matrix\n");

   for(int i=0;i<n;i++){

       if (a[i][i]!=0) c++;

        for(int j=0;j<o;j++){

            printf("%f ",a[i][j]);

        }

        printf("\n");

    }

    return c;

}

void gauss\_elimination(int n,int m,float a[n][m]){

    int res = forward\_elimination(n,m,a);

    printf("Pivots:%d",res);

}

int main(){

    int n,m;

    printf("n m :");

    scanf("%d",&n);

    scanf("%d",&m);

    float a[n][m];

    printf("Enter the elements:\n");

    for(int i=0;i<n;i++){

        for(int j=0;j<m;j++){

            scanf("%f",&a[i][j]);

        }

    }

   /\*for(int i=0;i<n;i++){

        for(int j=0;j<n+1;j++){

            printf("%f ",a[i][j]);

        }

        printf("\n");

    }

    \*/

   struct timespec start, end;

   clock\_gettime(CLOCK\_REALTIME, &start);

    gauss\_elimination(n,m,a);

    clock\_gettime(CLOCK\_REALTIME, &end);

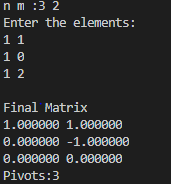
    printf("\nTime spent : %lf\n",n,

            time\_elapsed(start, end));

    return 0;

}

**Output:**



**5. The Four Fundamental Subspaces**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

double time\_elapsed(struct timespec start, struct timespec end) {

    double t;

    t = (end.tv\_sec - start.tv\_sec);

    t += (end.tv\_nsec - start.tv\_nsec) \* 0.000000001;

    return t;

}

int forward\_elimination(int n,int o,float a[n][o]){

    int c=0;

    for(int i=0;i<n-1;i++){

        if(a[i][i]==0){

            for(int m=i+1;m<n;m++){

                if(a[m][i]!=0){

                    for(int b=0;b<o;b++){

                        float temp= a[i][b];

                        a[i][b] = a[m][b];

                        a[m][b] = temp;

                    }

                    break;

                }

            }

            if(a[i][i]==0) return 0;

        }

        for(int k=1;k<n-i;k++){

            float l = a[i+k][i]/a[i][i];

            for(int j=0;j<o;j++){

                a[i+k][j] -= l\*a[i][j];

            }

        }

    }

    printf("\nFinal Matrix\n");

   for(int i=0;i<n;i++){

       if (i<o && a[i][i]!=0) c++;

        for(int j=0;j<o;j++){

            printf("%f ",a[i][j]);

        }

        printf("\n");

    }

    return c;

}

void gauss\_elimination(int n,int m,float a[n][m]){

    int res = forward\_elimination(n,m,a);

    printf("Column Space:%d\n",res);

    printf("Null Space:%d\n",n-res);

    float trans\_a[m][n];

    for (int i=0 ; i<m ; i++){

        for (int j=0 ; j<n ; j++){

            trans\_a[i][j] = a[j][i];

        }

    }

    printf("Transpose ");

    int res1 = forward\_elimination(m,n,trans\_a);

    printf("Row Space:%d\n",res1);

    printf("Left Null Space:%d\n",m-res1);

}

int main(){

    int n,m;

    printf("n m :");

    scanf("%d",&n);

    scanf("%d",&m);

    float a[n][m];

    printf("Enter the elements:\n");

    for(int i=0;i<n;i++){

        for(int j=0;j<m;j++){

            scanf("%f",&a[i][j]);

        }

    }

   /\*for(int i=0;i<n;i++){

        for(int j=0;j<n+1;j++){

            printf("%f ",a[i][j]);

        }

        printf("\n");

    }

    \*/

   struct timespec start, end;

   clock\_gettime(CLOCK\_REALTIME, &start);

    gauss\_elimination(n,m,a);

    clock\_gettime(CLOCK\_REALTIME, &end);

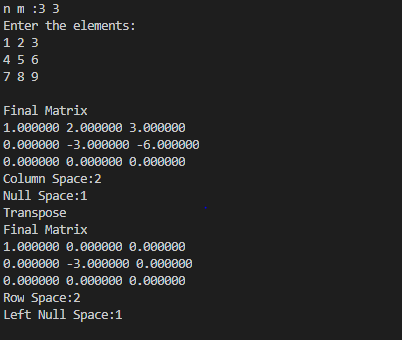
    printf("\nTime spent : %lf\n",n,

            time\_elapsed(start, end));

    return 0;

}

**Output:**



**6. Projections by Least Squares**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int swap\_rows(int n,float a[n][2\*n],int pos)

{

    int temp[2\*n];

    for (int i=pos+1;i<=n;i++){

        if (a[i][i]!=0){

            for (int j=0;j<=2\*n;j++){

                temp[j] = a[i][j];

                a[i][j] = a[pos][j];

                a[pos][j] = temp[j];

            }

            return 1;

        }

    }

    return 0;

}

void inverse(int n,float a[n+1][2\*n+1],float b[n+1][n+1])

{

    int i,j,k;

    float ratio;

    for(i=1;i<=n;i++)

    {

        for(j=1;j<=n;j++)

        {

            if(i==j)

            {

                a[i][j+n] = 1;

            }

            else

            {

                a[i][j+n] = 0;

            }

        }

    }

    for(i=1;i<=n;i++)

    {

        if(a[i][i] == 0.0)

        {

            if (swap\_rows(n,a,i))  continue;

            printf("Mathematical Error!");

            exit(0);

        }

        for(j=1;j<=n;j++)

        {

            if(i!=j)

            {

                ratio = a[j][i]/a[i][i];

                for(k=1;k<=2\*n;k++)

                {

                    a[j][k] = a[j][k] - ratio\*a[i][k];

                }

            }

        }

    }

    for(i=1;i<=n;i++)

    {

        for(j=n+1;j<=2\*n;j++)

        {

            a[i][j] = a[i][j]/a[i][i];

        }

    }

    for(i=1;i<=n;i++)

    {

        for(j=n+1;j<=2\*n;j++)

        {

            b[i][j-n] = a[i][j];

        }

    }

}

void multiplyMatrices(int r1, int c1, int r2, int c2,float first[r1+1][c1+1], float second[r2+1][c2+1], float mult[r1+1][c2+1] )

{

    for (int i = 1; i <= r1; ++i) {

        for (int j = 1; j <= c2; ++j) {

            mult[i][j] = 0;

        }

    }

    for (int i = 1; i <= r1; ++i) {

        for (int j = 1; j <= c2; ++j) {

            for (int k = 1; k <= c1; ++k) {

                mult[i][j] += first[i][k] \* second[k][j];

            }

        }

    }

    for (int i=1 ; i<=r1 ; i++){

        for(int j=1 ; j<=c2 ; j++){

            printf("%0.3f ",mult[i][j]);

        }

        printf("\n");

    }

}

int main()

{

         int i,j,k,n,m,n1,m1;

         printf("Enter order of matrix A ( nxm ): ");

         scanf("%d", &n);

         scanf("%d", &m);

         float input[n+1][m+1],trans[m+1][n+1],a[m+1][2\*m+1],b[n+1][m1+1],inv[m+1][m+1],mul[m+1][m+1],res[n+1][m1+1],final\_res[m+1][m1+1];

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

         for(i=1;i<=n;i++)

         {

              for(j=1;j<=m;j++)

              {

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

                trans[j][i] = input[i][j];

              }

         }

        printf("Enter order of matrix B ( n ): ");

        scanf("%d", &m1);

        printf("Enter coefficients of Matrix B:\n");

        for(i=1;i<=n;i++)

         {

              for(j=1;j<=m1;j++)

              {

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

              }

         }

        multiplyMatrices(m,n,n,m,trans,input,mul);

        for (int i=1 ; i<=m ; i++){

            for (int j=1 ; j<=m ; j++){

                a[i][j] = mul[i][j];

            }

        }

        inverse(m,a,inv);

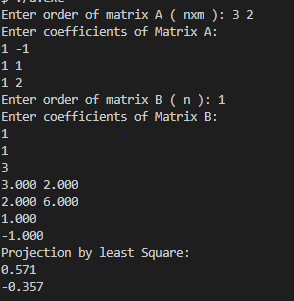
        multiplyMatrices(m,n,n,m1,trans,b,res);

        printf("Projection by least Square:\n");

        multiplyMatrices(m,m,m,m1,inv,res,final\_res);

}

**Output:**



**7. The Gram-Schmidt Orthogonalization**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

void main()

{

    int n,m;

    printf("nxm :");

    scanf("%d",&n);

    scanf("%d",&m);

    float mat[n][m];

    for(int i=0 ; i<n ; i++)

    {

        for (int j=0 ; j<m ; j++)

        {

            scanf("%f",&mat[i][j]);

        }

    }

    float ratio = 0;

    float length = 0;

    for (int i=1 ; i<m ; i++)

    {

        for(int k=i-1 ; k>=0 ; k--)

        {

            for (int l=0 ; l<n ; l++)

            {

                ratio += mat[l][i]\*mat[l][k];

            }

            for (int p=0 ; p<n ; p++)

            {

                length += mat[p][k]\*mat[p][k];

            }

            ratio = ratio/length;

            for (int l=0 ; l<n ; l++)

            {

                mat[l][i] -= ratio\*mat[l][k];

            }

            ratio = 0;

            length = 0;

            for (int u=0 ; u<n ; u++)

            {

                length += mat[u][i]\*mat[u][i];

            }

            for (int z=0 ; z<n ; z++)

            {

                mat[z][i] = mat[z][i]/sqrt(length);

            }

            length = 0;

        }

    }

    length = 0;

    for (int i=0 ; i<n ; i++)

    {

        length += mat[i][0] \* mat[i][0];

    }

    for (int i=0 ; i<n ; i++)

    {

        mat[i][0] = mat[i][0]/sqrt(length);

    }

    for (int i=0 ; i<n ; i++)

    {

        for (int j=0 ; j<m ; j++)

        {

            printf("%f ",mat[i][j]);

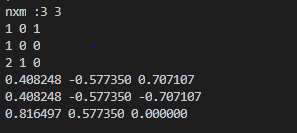
        }

        printf("\n");

    }

}

**Output:**



**8. Eigen values and Eigen Vectors of a Matrix**

#include<stdio.h>

#include<math.h>

void main()

{

    int i,j,n;

    float A[40][40],x[40],z[40],e[40],zmax,emax;

    printf("\nEnter the order of matrix:");

    scanf("%d",&n);

    printf("\nEnter matrix elements row-wise\n");

    for(i=1; i<=n; i++)

    {

        for(j=1; j<=n; j++)

        {

            scanf("%f",&A[i][j]);

        }

    }

    printf("\nEnter the column vector\n");

    for(i=1; i<=n; i++)

    {

        printf("X[%d]=",i);

        scanf("%f",&x[i]);

    }

    do

    {

        for(i=1; i<=n; i++)

        {

            z[i]=0;

            for(j=1; j<=n; j++)

            {

                z[i]=z[i]+A[i][j]\*x[j];

            }

        }

        zmax=fabs(z[1]);

        for(i=2; i<=n; i++)

        {

            if((fabs(z[i]))>zmax)

                zmax=fabs(z[i]);

        }

        for(i=1; i<=n; i++)

        {

            z[i]=z[i]/zmax;

        }

        for(i=1; i<=n; i++)

        {

            e[i]=0;

            e[i]=fabs((fabs(z[i]))-(fabs(x[i])));

        }

        emax=e[1];

        for(i=2; i<=n; i++)

        {

            if(e[i]>emax)

                emax=e[i];

        }

        for(i=1; i<=n; i++)

        {

            x[i]=z[i];

        }

    }

    while(emax>0.001);

    printf("\n The required eigen value is %f",zmax);

    printf("\n\nThe required eigen vector is :\n");

    for(i=1; i<=n; i++)

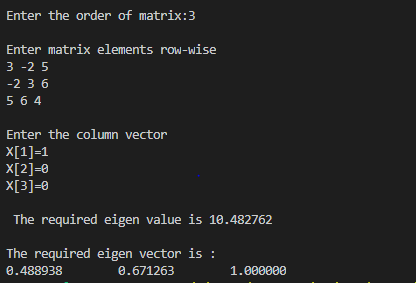
    {

        printf("%f\t",z[i]);

    }

}

**Output**:



**9. The Largest Eigen Value of a Matrix by the Power Method**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define SIZE 10

int main()

{

     float a[SIZE][SIZE], x[SIZE],x\_new[SIZE];

     float temp, lambda\_new, lambda\_old, error;

     int i,j,n, step=1;

     /\* Inputs \*/

     printf("Enter Order of Matrix: ");

     scanf("%d", &n);

     printf("Enter Tolerable Error: ");

     scanf("%f", &error);

     /\* Reading Matrix \*/

     printf("Enter Coefficient of Matrix:\n");

     for(i=1;i<=n;i++)

     {

          for(j=1;j<=n;j++)

          {

               scanf("%f", &a[i][j]);

          }

     }

     /\* Reading Intial Guess Vector \*/

     printf("Enter Initial Guess Vector:\n");

     for(i=1;i<=n;i++)

     {

          printf("x[%d]=",i);

          scanf("%f", &x[i]);

     }

     /\* Initializing Lambda\_Old \*/

     lambda\_old = 1;

     /\* Multiplication \*/

     up:

     for(i=1;i<=n;i++)

     {

          temp = 0.0;

          for(j=1;j<=n;j++)

          {

            temp = temp + a[i][j]\*x[j];

          }

          x\_new[i] = temp;

     }

     /\* Replacing \*/

     for(i=1;i<=n;i++)

     {

        x[i] = x\_new[i];

     }

     /\* Finding Largest \*/

     lambda\_new = fabs(x[1]);

     for(i=2;i<=n;i++)

     {

          if(fabs(x[i])>lambda\_new)

          {

            lambda\_new = fabs(x[i]);

          }

     }

     /\* Normalization \*/

     for(i=1;i<=n;i++)

     {

        x[i] = x[i]/lambda\_new;

     }

     /\* Display \*/

     printf("\n\nSTEP-%d:\n", step);

     printf("Eigen Value = %f\n", lambda\_new);

     printf("Eigen Vector:\n");

     for(i=1;i<=n;i++)

     {

        printf("%f\t", x[i]);

     }

     /\* Checking Accuracy \*/

     if(fabs(lambda\_new-lambda\_old)>error)

     {

          lambda\_old=lambda\_new;

          step++;

          goto up;

     }

     return(0);

}

**Output:**

