UE18MA251 - Linear Algebra - Coding Assignment

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - 'A'
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - 'A'

Assignment 1 -

Problem Statement - Using OpenMP for the Gaussian Elimination process and compare the code for parallelized and non parallelized versions.

Code -

```
// Gaussian elimination without pivoting.

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <sys/types.h>
#include <sys/times.h>
#include <sys/time.h>
#include <time.h>
#include <omp.h>

/* Program Parameters */
#define MAXN 2000  /* Max value of N */
int N;  /* Matrix size */

/* Matrices and vectors */
volatile float A[MAXN][MAXN], B[MAXN], X[MAXN];
/* A * X = B, solve for X */
```

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A

```
/* junk */
#define randm() 4|2[uid]&3

/* Prototype */
void gauss();  /* The function you will provide.
   * It is this routine that is timed.
   * It is called only on the parent.
   */

/* returns a seed for srand based on the time */
unsigned int time_seed() {
 struct timeval t;
 struct timezone tzdummy;

 gettimeofday(&t, &tzdummy);
 return (unsigned int)(t.tv_usec);
}

/* Set the program parameters from the command-line arguments */
void parameters(int argc, char **argv) {
 int seed = 0;  /* Random seed */
 char uid[32]; /*User name */

 /* Read command-line arguments */
 srand(time_seed());  /* Randomize */

 if (argc == 3) {
   seed = atoi(argv[2]);
   srand(seed);
```

```
  printf("Random seed = %i\n", seed);
 }
 if (argc >= 2) {
  N = atoi(argv[1]);
  if (N < 1 || N > MAXN) {
    printf("N = %i is out of range.\n", N);
    exit(0);
  }
 }
 else {
  printf("Usage: %s <matrix_dimension> [random seed]\n",
         argv[0]);
  exit(0);
 }

 /* Print parameters */
 printf("\nMatrix dimension N = %i.\n", N);
}

/* Initialize A and B (and X to 0.0s) */
void initialize_inputs() {
 int row, col;

 printf("\nInitializing...\n");
 for (col = 0; col < N; col++) {
  for (row = 0; row < N; row++) {
    A[row][col] = (float)rand() / 32768.0;
  }
  B[col] = (float)rand() / 32768.0;
  X[col] = 0.0;
```

```
 }


}

/* Print input matrices */
void print_inputs() {
 int row, col;

 if (N < 10) {
   printf("\nA =\n\t");
   for (row = 0; row < N; row++) {
     for (col = 0; col < N; col++) {
 printf("%5.2f%s", A[row][col], (col < N-1) ? ", " : ";\n\t");
     }
   }
   printf("\nB = [");
   for (col = 0; col < N; col++) {
     printf("%5.2f%s", B[col], (col < N-1) ? "; " : "]\n");
   }
 }
}

void print_X() {
 int row;

 if (N < 100) {
   printf("\nX = [");
   for (row = 0; row < N; row++) {
     printf("%5.2f%s", X[row], (row < N-1) ? "; " : "]\n");
   }
```

```
 }
}

int main(int argc, char **argv) {
 /* Timing variables */
 struct timeval etstart, etstop;  /* Elapsed times using gettimeofday() */
 struct timezone tzdummy;
 clock_t etstart2, etstop2;  /* Elapsed times using times() */
 unsigned long long usecstart, usecstop;
 struct tms cputstart, cputstop;  /* CPU times for my processes */

 /* Process program parameters */
 parameters(argc, argv);

 /* Initialize A and B */
 initialize_inputs();

 /* Print input matrices */
 print_inputs();

 /* Start Clock */
 printf("\nStarting clock.\n");
 gettimeofday(&etstart, &tzdummy);
 etstart2 = times(&cputstart);

 /* Gaussian Elimination */
 gauss();

 /* Stop Clock */
 gettimeofday(&etstop, &tzdummy);
```

```
etstop2 = times(&cputstop);
printf("Stopped clock.\n");
usecstart = (unsigned long long)etstart.tv_sec * 1000000 + etstart.tv_usec;
usecstop = (unsigned long long)etstop.tv_sec * 1000000 + etstop.tv_usec;

/* Display Output Screenshot  Screenshot  Screenshot  Screenshot  Screenshot  */
print_X();

/* Display timing results */
printf("\nElapsed time = %g ms.\n",
 (float)(usecstop - usecstart)/(float)1000);

printf("(CPU times are accurate to the nearest %g ms)\n",
 1.0/(float)CLOCKS_PER_SEC * 1000.0);
printf("My total CPU time for parent = %g ms.\n",
 (float)( (cputstop.tms_utime + cputstop.tms_stime) -
    (cputstart.tms_utime + cputstart.tms_stime) ) /
 (float)CLOCKS_PER_SEC * 1000);
printf("My system CPU time for parent = %g ms.\n",
 (float)(cputstop.tms_stime - cputstart.tms_stime) /
 (float)CLOCKS_PER_SEC * 1000);
printf("My total CPU time for child processes = %g ms.\n",
 (float)( (cputstop.tms_cutime + cputstop.tms_cstime) -
    (cputstart.tms_cutime + cputstart.tms_cstime) ) /
 (float)CLOCKS_PER_SEC * 1000);
    /* Contrary to the man pages, this appears not to include the parent */
printf("--------------------------------------------\n");

exit(0);
}
```

```
/* ------------------ Above Was Provided --------------------- */

/****** You will replace this routine with your own parallel version *******/
/* Provided global variables are MAXN, N, A[][], B[], and X[],
* defined in the beginning of this code.  X[] is initialized to zeros.
*/
void gauss() {
 int norm, row, col;  /* Normalization row, and zeroing
     * element row and col */
 float multiplier;

 printf("Computing Serially.\n");

 /* Gaussian elimination */

 for (norm = 0; norm < N - 1; norm++) {
   #pragma omp parallel for shared(A, B) private(multiplier,row,col)
   for (row = norm + 1; row < N; row++) {
     multiplier = A[row][norm] / A[norm][norm];
     for (col = norm; col < N; col++) {
          A[row][col] -= A[norm][col] * multiplier;
     }
     B[row] -= B[norm] * multiplier;
   }
 }
 /* (Diagonal elements are not normalized to 1.  This is treated in back
  * substitution.)
  */
```

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A

```
 /* Back substitution */
 for (row = N - 1; row >= 0; row--) {
   X[row] = B[row];
   for (col = N-1; col > row; col--) {
     X[row] -= A[row][col] * X[col];
   }
   X[row] /= A[row][row];
 }
}
```

Output Screenshot -

```
ITs-MacBook-Air% gcc openmp_gauss.c
ITs-MacBook-Air% ./a.out 3 5
Random seed = 5

Matrix dimension N = 3.

Initializing...

A =
         2.56, 43505.17, 25983.85;
        43102.30, 6212.55, 44136.17;
        50988.75, 15415.58, 60129.96;

B = [19218.47; 25842.39; 39130.36]

Starting clock.
Computing Serially.
Stopped clock.

X = [-0.96; -0.51;  1.60]

Elapsed time = 0.011 ms.
(CPU times are accurate to the nearest 0.001 ms)
My total CPU time for parent = 0 ms.
My system CPU time for parent = 0 ms.
My total CPU time for child processes = 0 ms.
```

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A

```
ITs-MacBook-Air% cd Desktop/LA/Assignment\
ITs-MacBook-Air% gcc openmp_gauss.c
ITs-MacBook-Air% ./a.out 100 40
Random seed = 40

Matrix dimension N = 100.

Initializing...

Starting clock.
Computing Serially.
Stopped clock.

Elapsed time = 1.388 ms.
(CPU times are accurate to the nearest 0.001 ms)
My total CPU time for parent = 0 ms.
My system CPU time for parent = 0 ms.
My total CPU time for child processes = 0 ms.
```

```
ITs-MacBook-Air% gcc openmp_gauss.c
ITs-MacBook-Air% ./a.out 1000 696
Random seed = 696

Matrix dimension N = 1000.

Initializing...

Starting clock.
Computing Serially.
Stopped clock.

Elapsed time = 1155.71 ms.
(CPU times are accurate to the nearest 0.001 ms)
My total CPU time for parent = 0.114 ms.
My system CPU time for parent = 0.001 ms.
My total CPU time for child processes = 0 ms.
```

Assignment 2 -

Problem Statement 1 - Using OpenMP find the LU decomposition of a matrix.

Code -

```
#include<stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
//#include <mpi.h>
```

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A

```
double **make2dmatrix(long n);
void free2dmatrix(double ** M, long n);
void printmatrix(double **A, long n);
long matrix_size,version;
char algo;

void decomposeOpenMP(double **A, long n)
{
   printf("\nDECOMPOSE OPENMP CALLED\n");
   long i,j,k,rows,mymin,mymax;
   int pid=0;
   int nprocs;
#pragma omp parallel shared(A,n,nprocs) private(i,j,k,pid,rows,mymin,mymax)
   {
#ifdef _OPENMP
       nprocs=omp_get_num_threads();
#endif
#ifdef _OPENMP
       pid=omp_get_thread_num();
#endif
    //  printf("1. I am proc no %d out of %d\n",pid,nprocs);
       rows=n/nprocs;
       mymin=pid * rows;
       mymax=mymin + rows - 1;
       if(pid==nprocs-1 && (n-(mymax+1))>0)
           mymax=n-1;
       for(k=0;k<n;k++){
           if(k>=mymin && k<=mymax){
               //#pragma omp for schedule(static)
```

```
                for(j=k+1;j<n;j++){
                    A[k][j] = A[k][j]/A[k][k];
                }
            }
#pragma omp barrier
            for(i=(((k+1) > mymin) ? (k+1) : mymin);i<=mymax;i++){
                //#pragma omp for schedule(static)
                for(j=k+1;j<n;j++){
                    A[i][j] = A[i][j] - A[i][k] * A[k][j];
                }
            }
        }
    }
}

int checkVersion1(double **A, long n)
{
    long i, j;
    for (i=0;i<n;i++)
    {
        for (j=0;j<n;j++){
            if(A[i][j]!=1){
                return 0;
            }
        }
    }
    return 1;
}
void initializeVersion1(double **A, long n)
{
```

```
    long i, j;
    for (i=0;i<n;i++){
        for (j=0;j<n;j++){
            if(i<=j )
                A[i][j]=i+1;
            else
                A[i][j]=j+1;
         }
    }
}

int checkVersion2(double **A, long n)
{
    long i,j;
    for(i=0;i<n;i++){
        if(A[i][i]!=1){
            return 0;
        }
        for(j=0;j<n;j++){
            if(i!=j && A[i][j]!=2){
                return 0;
            }
        }
    }
    return 1;
}
void initializeVersion2(double **A,long n){
    long i,j, k;
    for(i=0;i<n;i++){
        for(j=i;j<n;j++){
```

```
            if(i==j){
                k=i+1;
                A[i][j]=4*k-3;
            }
            else{
                A[i][j]=A[i][i]+1;
                A[j][i]=A[i][i]+1;
            }
        }
    }
}

double **getMatrix(long size,long version)
{
    double **m=make2dmatrix(size);
    switch(version){
    case 1:
        initializeVersion1(m,size);
        break;
    case 2:
        initializeVersion2(m,size);
        break;
    default:
        printf("INVALID VERSION NUMBER");
        exit(0);
    }
    return m;
}
int check(double **A, long size, long version){
    switch(version){
```

```
   case 1:
       return checkVersion1(A,size);
       break;
   case 2:
       return checkVersion2(A,size);
       break;
   default:
       printf("INVALID VERSION CHARACTER IN CHECK");
       exit(0);
   }
}

int main(int argc, char *argv[]){
       int choice;
       change:
       printf("Enter the size of matrix (N x N) where N = ");
       scanf("%lu",&matrix_size);
       version=1;
        //printmatrix(matrix,matrix_size);
   int wish=1;
   clock_t begin, end;
   double time_spent;
   double **matrix;
   int num_threads;
 while(wish!=0)
 {        printf("\n\nEnter your choice:\n1.Sequential processing\n2.Parallel
processing\n3.Change order of A\n0.Exit\n");
       scanf("%d",&wish);
       switch(wish)
       {
```

```
        case 1: /* Seq. LU factorization */
                num_threads=1;
                omp_set_num_threads(num_threads);
                matrix=getMatrix(matrix_size,version);
                begin = clock();
                decomposeOpenMP(matrix,matrix_size);
                end = clock();
                time_spent = ((double)(end - begin)) / CLOCKS_PER_SEC;
                printf("\n*********************************\n\n");
                printf("Processing Type:%s\n","Sequential");
                printf("Size of Matrix :%lu \n",matrix_size);
                printf("Version Number : %lu\n",version);
                //printf("Number of Procs : %lu\n",num_threads);
                printf("%s",check(matrix,matrix_size,version)==1? "FACTORIZATION
SUCCESSFULL\n":"DECOMPOSE FAIL\n");
                printf("DECOMPOSE TIME TAKEN : %f seconds\n",time_spent);
                printf("\n*********************************\n\n");
                free2dmatrix(matrix,matrix_size);
                break;

        case 2:/* Parallel LU Factorization*/
                printf("\nEnter the number of processes/threads:");
                scanf("%d",&num_threads);
                omp_set_num_threads(num_threads);
                matrix=getMatrix(matrix_size,version);
                begin = clock();
                decomposeOpenMP(matrix,matrix_size);
                end = clock();
                time_spent = ((double)(end - begin)) / CLOCKS_PER_SEC;
                printf("\n*********************************\n\n");
```

```
                printf("Processing Type:%s\n","Parallel");
                printf("Size of Matrix :%lu \n",matrix_size);
                printf("Version Number : %lu\n",version);
                printf("Number of Procs : %u\n",num_threads);
                printf("%s",check(matrix,matrix_size,version)==1? "FACTORIZATION
SUCCESSFULL\n":"DECOMPOSE FAIL\n");
                printf("DECOMPOSE TIME TAKEN : %f seconds\n",time_spent);
                printf("\n*********************************\n\n");
                free2dmatrix(matrix,matrix_size);
                break;
        case 3:goto change;
        }
            }
   return 0;
}
 double **make2dmatrix(long n)
{    long i;
   double **m;
   m = (double**)malloc(n*sizeof(double*));
   for (i=0;i<n;i++)
       m[i] = (double*)malloc(n*sizeof(double));
   return m;
}
// only works for dynamic arrays:
void printmatrix(double **A, long n)
{
   printf("\n *************** MATRIX ***************\n\n");
   long i, j;
   for (i=0;i<n;i++)
   {        for (j=0;j<n;j++)
```

```
            printf("%f ",A[i][j]);
        printf("\n");
    }
}

void free2dmatrix(double ** M, long n)
{
    long i;
    if (!M) return;
    for(i=0;i<n;i++)
        free(M[i]);
    free(M);
}
```

Output Screenshot -

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A

```
Enter the size of matrix (N x N) where N = 3


Enter your choice:
1.Sequential processing
2.Parallel processing
3.Change order of A
0.Exit
1

DECOMPOSE OPENMP CALLED

*********************************

Processing Type:Sequential
Size of Matrix :3
Version Number : 1
FACTORIZATION SUCCESSFULL
DECOMPOSE TIME TAKEN : 0.000078 seconds

*********************************



Enter your choice:
1.Sequential processing
2.Parallel processing
3.Change order of A
0.Exit
2

Enter the number of processes/threads:4

DECOMPOSE OPENMP CALLED

*********************************

Processing Type:Parallel
Size of Matrix :3
Version Number : 1
Number of Procs : 4
FACTORIZATION SUCCESSFULL
DECOMPOSE TIME TAKEN : 0.000407 seconds
```

Problem Statement 2 - Compute the four fundamental subspaces given an input matrix.

Code -

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


void back_substitution(int n,int m,float a[n][m]){
   for (int i=n-1 ; i>=0 ; i--)
   {
       if (a[i][i] != 0)
       {
```

```
            for (int k=i-1 ; k>=0 ; k--)
            {
                float l = a[k][i]/a[i][i];
                for(int j=0;j<m;j++){
                    //printf("%d %d %d\n",i,j,k);
                    a[k][j] -= l*a[i][j];
                }
            }
        }
    }
    printf("\n echelon  Matrix\n");
   for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            printf("%f ",a[i][j]);
        }
        printf("\n");
    }

}

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];
            }
        }
    }
  for(int i=0;i<n;i++){
      if (i<o && a[i][i]!=0) c++;
    }

    return c;
}



void echelon _form(int n,int m,float a[n][m]){

    int res = forward_elimination(n,m,a);
    printf("Column Matrix");
    back_substitution(n,m,a);
    printf("\nColumn Space:%d\n",res);
    for (int i=0 ; i<n ; i++)
    {
        for (int j=0 ; j<res ; j++)
```

```
        {
            printf("%f %f",a[j][i]);
        }
        printf("\n");
    }
    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("\n\n Transpose of a matrix (Row Matrix)");
    int res1 = forward_elimination(m,n,trans_a);
    back_substitution(m,n,trans_a);
    printf("\nRow Space:%d\n",res1);
    for (int i=0 ; i<n ; i++)
    {
        for (int j=0 ; j<res1 ; j++)
        {
            printf("%f %f",a[j][i]);
        }
        printf("\n");
    }
    printf("\nLeft Null Space:%d\n",n-res1);
    float left_null_space[n][n-res1];
    if (n-res1 != 0){
    for (int i=0 ; i<n ; i++)
    {
        for (int j=0 ; j<n-res1 ; j++)
        {
```

```
            left_null_space[i][j] = 0;
        }
    }
    for (int i=0 ; i<m ; i++)
    {
        if (a[i][i] != 0)
        {
            for (int j = i+1 ; j<n ; j++)
            {
                if (a[i][j] != 0)
                {
                    left_null_space[i][n-j-1] = -a[i][j]/a[i][i];
                    left_null_space[j][n-j-1] = 1;
                }
            }
        }
    }
    for (int i=0 ; i<n ; i++)
    {
        for (int j=0 ; j<n-res ; j++)
        {
            printf ("%f ",left_null_space[i][j]);
        }
        printf("\n");
    }
    }
    printf("\nNull Space:%d\n",m-res);
    float null_space[m][m-res];

    if (m-res !=0){
```

```
    for (int i=0 ; i<m ; i++)
    {
        for (int j=0 ; j<m-res ; j++)
        {
            null_space[i][j] = 0;
        }
    }

    for (int i=0 ; i<n ; i++)
    {
        if (a[i][i] != 0)
        {
            for (int j = i+1 ; j<m ; j++)
            {
                if (a[i][j] != 0)
                {
                    null_space[i][m-j-1] = -a[i][j]/a[i][i];
                    null_space[j][m-j-1] = 1;
                }
            }
        }
    }
    for (int i=0 ; i<m ; i++)
    {
        for (int j=0 ; j<m-res ; j++)
        {
            printf ("%f ",null_space[i][j]);
        }
        printf("\n");
    }
```

```
    }
}

int main(){

    int n,m;
    printf("Enter the dimensions:");
    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]);
        }
    }
     echelon _form(n,m,a);
    return 0;

}
```

Output Screenshot -

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A

```
ITs-MacBook-Air% ./a.out
Enter the dimensions:3 3
Enter the elements:
3 -1 2
4 6 7
1 0 2
Column Matrix
 Echleon Matrix
3.000000 0.000000 0.000000
0.000000 7.333333 0.000000
0.000000 0.000000 1.136364

Column Space:3
3.000000 0.0000000.000000 0.0000000.000000 0.000000
0.000000 0.0000007.333333 0.0000000.000000 0.000000
0.000000 0.0000000.000000 0.0000001.136364 0.000000


 Transpose of a matrix (Row Matrix)
 Echleon Matrix
3.000000 0.000000 0.000000
0.000000 7.333333 0.000000
0.000000 0.000000 1.136364

Row Space:3
3.000000 0.0000000.000000 0.0000000.000000 0.000000
0.000000 0.0000007.333333 0.0000000.000000 0.000000
0.000000 0.0000000.000000 0.0000001.136364 0.000000

Left Null Space:0

Null Space:0
```

Problem Statement 3 - Find the basis and dimension of a matrix and time taken for execution.

Code -

```c
#include <stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>


int R,C;
void swap(int mat[R][C], int row1, int row2,int col)
{
   for (int i = 0; i < col; i++)
   {
```

```
        int temp = mat[row1][i];
        mat[row1][i] = mat[row2][i];
        mat[row2][i] = temp;
    }
}

void display(int mat[R][C], int row, int col);

int rankOfMatrix(int mat[R][C],int *a)
{
    int rank = C;


    for (int row = 0; row < rank; row++)
    {
        if (mat[row][row])
    {
    //a[row]=1;
    for (int col = 0; col < R; col++)
    {
        if (col != row)
        {
            double mult = (double)mat[col][row] / mat[row][row];
            for (int i = 0; i < rank; i++)
            mat[col][i] -= mult * mat[row][i];

        }
    }
}
```

```
    else
    {
        int reduce = 1;
        for (int i = row + 1; i < R; i++)
        {
            if (mat[i][row])
            {
                swap(mat, row, i, rank);
                reduce = 0;
                break ;
            }
        }
        if (reduce)
        {
            a[row]=-1;
            rank--;
            for (int i = 0; i < R; i ++)
            mat[i][row] = mat[i][rank];

        }
        row--;
    }
}
return rank;
}

void display(int mat[R][C], int row, int col)
{

    for (int i = 0; i < row; i++)
```

```
    {
        for (int j = 0; j < col; j++)
        printf(" %d", mat[i][j]);
        printf("\n");
    }
}

void printbasis(int mat[R][C],int *a,int x)
{
    printf("Basis for the given matrix\n");
    for(int j=0;j<x;j++)
    {
        printf("( ");
        if(a[j]!=-1)
        {
            for(int i=0;i<R;i++)
            {
                printf("%d",mat[i][j]);
            }
        }
        printf(")");
        printf("\n");
    }
}

int main()
{
    clock_t start,end;
    printf("Enter the number of rows:\n");
    scanf("%d",&R);
```

```
printf("Enter the number of columns:\n");
scanf("%d",&C);
int mat[R][C];
int mat1[R][C];
for(int i=0;i<R;i++)
for(int j=0;j<C;j++)
{
    mat[i][j]=rand();
    mat1[i][j]=mat[i][j];
}

int x;



if(R<C)
{
  x=R;
}
else {
  x=C;
}

int a[x];
for(int i=0;i<x;i++)
a[i]=1;
start=clock();
int rank=rankOfMatrix(mat,a);
if(rank>x)
rank=x;
printf("\n");
```

```
   end=clock();
   printbasis(mat1,a,x);
   printf("%d is the dimension .\n",rank);
   printf("The time taken for this executionis
%lf\n",((double)(end-start))/CLOCKS_PER_SEC);
   return 0;
}
```

Output Screenshot -

```
ITs-MacBook-Air% gcc basis.c
ITs-MacBook-Air% ./a.out
Enter the number of rows:
6
Enter the number of columns:
5

Basis for the given matrix
( 16807470211272823564440113752250389654430311315709 33)
( 28247524910102754411154381651441282327147483316919749 3099)
( )
( 98494365814587779237424304282337884019980971578933518 16)
( 11441089302007237709114807987143542612181712956015057 95335)
2 is the dimension .
The time taken for this executionis 0.000033
```

Assignment 3 -

Problem Statement 1 - Compute Eigen Values and Eigen Vectors for a given square matrix

Code -

```
#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]);
    }
}
```

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A

Output Screenshot -

Problem Statement 2 - Compute the normal equation

Code -

Gram-Schmidt Method

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

int main()
{
   int n,m;
   printf("Enter the dimensions:");
   scanf("%d%d",&n,&m);

   float mat[n][m];
```

```
printf("Enter the elements\n");

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];
            }
            printf("%f\n",length);
            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 Screenshot -

```
ITs-MacBook-Air% gcc norm.c
ITs-MacBook-Air% ./a.out
Enter the dimensions:3 3
Enter the elements
3 4 -3
2 1 6
7 8 3
1.967742
11.946057
0.222311
0.381000 0.436926 -0.814821
0.254000 -0.896848 -0.362143
0.889001 0.068988 0.452679
```

Code (for the normal equation A'Ax_hat = A'b , where A' is A transpose) -

```
#include<stdio.h>
#include<math.h>
#include<omp.h>
#define MAXN 25


float determinant(float [][MAXN], float);
void cofactor(float [][MAXN], float,float [][MAXN],float [][MAXN],float);
void transpose(float [][MAXN], float [][MAXN], float,float [][MAXN],float
[][MAXN],float);


int findmultiply(float a[][MAXN],float b[][MAXN],float c[][MAXN],int m,int n,int p)
{
 int i,j,k;
#pragma omp parallel shared(a,b,c) private(i,j,k)
```

```
  {
#pragma omp for  schedule(static)
  for (i=0; i<m; ++i){
     for (j=0; j<n; ++j){
        a[i][j]=0.;
        for (k=0; k<p; ++k){
           a[i][j]=(a[i][j])+((b[i][k])*(c[k][j]));
        }
     }
  }
  }
  return 0;
}

int main()
{
 float a[MAXN][MAXN],b[MAXN][MAXN],c[MAXN][MAXN],e[MAXN][MAXN],m,n,d;
 int i, j;
 printf("Enter the number of the rows of the Matrix (that is m) : ");
 scanf("%f", &m);
 printf("Enter the number of the columns of the Matrix (that is n) : ");
 scanf("%f", &n);


 printf("Enter the elements of A %.0fX%.0f Matrix : \n", m, n);
 for (i = 0;i < m; i++)
   {
    for (j = 0;j < n; j++)
      {
       scanf("%f", &a[i][j]);
```

```
      c[j][i]=a[i][j];//this is the transpose;
      }
   }
 printf("Enter the elements of B %.0fX%.0f Matrix that is 'm' elements : \n", m,1.0);
 for (i = 0;i < m; i++)
   {
    scanf("%f",&b[i][0]);
   }
   //m=n;n=n;p=m
 findmultiply(e,c,a,n,n,m);

 d = determinant(e, n);
 if (d == 0)
  printf("\nInverse of Entered Matrix is not possible\n");
 else
  cofactor(e,n,c,b,m);
}

/*For calculating Determinant of the Matrix */
float determinant(float a[MAXN][MAXN], float k)
{
 float s = 1, det = 0, b[MAXN][MAXN];
 int i, j, m, n, c;
 if (k == 1)
   {
    return (a[0][0]);
   }
 else
   {
    det = 0;
```

```
    for (c = 0; c < k; c++)
    {
     m = 0;
     n = 0;
     for (i = 0;i < k; i++)
       {
         for (j = 0 ;j < k; j++)
           {
             b[i][j] = 0;
             if (i != 0 && j != c)
              {
                b[m][n] = a[i][j];
                if (n < (k - 2))
                 n++;
                else
                 {
                  n = 0;
                  m++;

                 }
                }
           }
        }
       det = det + s * (a[0][c] * determinant(b, k - 1));
       s = -1 * s;
       }

}
return (det);
```

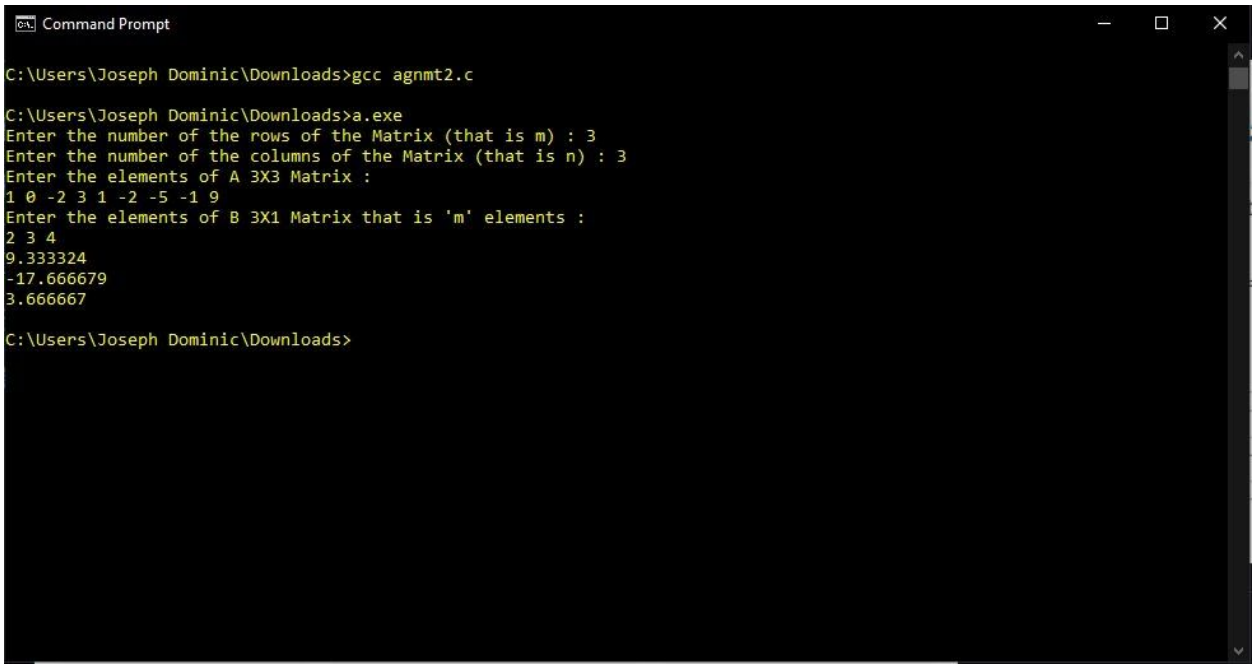
```
}
void cofactor(float num[MAXN][MAXN], float f,float c[MAXN][MAXN],float
b1[][MAXN],float border)
{
float b[MAXN][MAXN], fac[MAXN][MAXN];
int p, q, m, n, i, j;

for (q = 0;q < f; q++)
{
  for (p = 0;p < f; p++)
   {
    m = 0;
    n = 0;
    for (i = 0;i < f; i++)
    {
      for (j = 0;j < f; j++)
       {
         if (i != q && j != p)
         {
           b[m][n] = num[i][j];
           if (n < (f - 2))
            n++;
           else


            {
              n = 0;
              m++;
              }
           }
       }
```

```
    }

    fac[q][p] = pow(-1, q + p) * determinant(b, f - 1);
   }
 }
 transpose(num, fac, f,c,b1,border);

}
/*Finding transpose of matrix*/
void transpose(float num[MAXN][MAXN], float fac[MAXN][MAXN], float r,float
atrans[MAXN][MAXN],float b1[][MAXN],float border)
{
 int i, j;
 float b[MAXN][MAXN], inverse[MAXN][MAXN], d;
 for (i = 0;i < r; i++)
   {
    for (j = 0;j < r; j++)
      {
        b[i][j] = fac[j][i];
       }
   }
 d = determinant(num, r);
 for (i = 0;i < r; i++)
   {
    for (j = 0;j < r; j++)
      {
       inverse[i][j] = b[i][j] / d;
       }
   }
  float midmul[MAXN][MAXN];
```

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A

```
  findmultiply(midmul,inverse,atrans,r,border,r);
  float finalans[MAXN][MAXN];
  findmultiply(finalans,midmul,b1,r,1,border);
  for (int i=0;i<r;++i)
   printf("%f\n",finalans[i][0]);
}
```

Output Screenshot -



Problem Statement 3 - Compute the largest Eigen value and the corresponding eigen vector using power method

Code -

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

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A

```c
#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 Screenshot -

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A

PROBLEMS 6 OUTPUT TERMINAL DEBUG CONSOLE 2: zsh

```
ITs-MacBook-Air% gcc power_method.c
ITs-MacBook-Air% ./a.out
Enter Order of Matrix: 3
Enter Tolerable Error: 0.2
Enter Coefficient of Matrix:
1 4 5
6 7 8
-9 0 2
Enter Initial Guess Vector:
x[1]=3
x[2]=2
x[3]=4


STEP-1:
Eigen Value = 64.000000
Eigen Vector:
0.484375        1.000000        -0.296875

STEP-2:
Eigen Value = 7.531250
Eigen Vector:
0.398340        1.000000        -0.657676

STEP-3:
Eigen Value = 4.900415
Eigen Vector:
0.226503        0.842506        -1.000000

STEP-4:
Eigen Value = 4.038527
Eigen Vector:
-0.347521       -0.184086       -1.000000

STEP-5:
Eigen Value = 11.373729
Eigen Vector:
-0.534905       -1.000000       0.099148

STEP-6:
Eigen Value = 9.416245
Eigen Vector:
-0.428957       -1.000000       0.532319

STEP-7:
Eigen Value = 5.315193
Eigen Vector:
-0.332512       -1.000000       0.926636

STEP-8:
Eigen Value = 4.845878
Eigen Vector:
0.062046        -0.326459       1.000000

STEP-9:
Eigen Value = 6.087065
Eigen Vector:
0.617081        1.000000        0.236827

STEP-10:
```

Name - Tejas Srinivasan SRN - PES1201800110 Section - 4 - A
Name - Joseph Dominic SRN - PES1201800328 Section - 4 - A