DAA PROJECT

UE17CS251

# MATRIX LIBRARY

PRUTHVISH E – PES1201701629

HARISH PB – PES1201701435

# ABSTRACT

We work with many domains which need the implementation of matrices such as machine learning,AI,statistics,advanced mathematical problems,...etc.

Matrices play a vital role in the above mentioned fields it helps us to model a complex real life problem to a simple data structure such as a 2D array and solve the complex problem by applying suitable algorithms on the data, and getting suitable or desired results from it.

Matrices help us to transform a given complex data into a simple understandable form such as the 2D array.

We in our DAA project have implemented a matrix library in C to aid us in solving this complex problems.

# PROBLEM DEFINITION

Implementation of a matrix library in C language.

# DESIGN

//Creating a matrix of order = r\*c with r\*c elements

matrix \* create\_matrix(int c,int r,double \* elements);

//Display a Matrix

void displayMat(const matrix \*m);

//Check matrices are compatible for multiplication returns 1 if possible else false

int multiply\_compat(const matrix \*m1,const matrix\* m2);

//Multiply two compatible matrices returns NULL if the matrices are not compatible

matrix \* multiply\_mat(const matrix\* m1,const matrix\* m2);

//Delete a matrix

matrix\* destroy\_matrix(matrix \*m1);

//Add two matrices m1 and m2 returns null if incompatible

matrix\* add(const matrix \*m1,const matrix \*m2);

//Subtract two matrices m1 and m2 returns null if incompatible

matrix\* subtract(const matrix \*m1,const matrix \*m2);

//Create Identitiy Matrix of order n

matrix \*identity(int n);

//Create Zero Matrix of order r\*c

matrix \*zeroMatrix(int r, int c);

//Swap rows a and b of matrix m returns 1 on success

int row\_swap(matrix \*m, int a, int b);

//Swap columns a and b of matrix returns 1 on success

int col\_swap(matrix \*m, int a, int b);

//Multiply matrix by a Scalar returns 1 on success

int scalar\_multiply(matrix \*m, double f);

//Reduce row b by a factor\*a times returns 1 on success

int reduceRow(matrix \*m, int b, int a, double factor);

//Reduce column b by a factor\*a times returns 1 on success

int reduceCol(matrix \*m, int b, int a, double factor);

//Check for equality Of Two Matrices returns 1 if equal 0 otherwise

int equals(matrix \*m1, matrix \*m2);

//clone a matrix m

matrix \*clone(const matrix \*m);

//Transpose a matrix m

matrix \*transpose(const matrix \*m);

//Generate a random matrix

matrix \*rand\_matrix(int r, int c,float maxNo);

//Augment a matrix m1 with another matrix m2 returns NULL if incompatible

matrix \*aug\_matrix(const matrix \*m1,const matrix \* m2);

//To find Cofactor of a matrix

matrix \* cofactor(const matrix \* m1);

//To find adjoint of a matrix

matrix \* adjoint(const matrix \* m1);

//Find Determinant of a matrix m

double determinant(const matrix \*m,int n);

//Find Inverse of the matrix returns NULL if matrix is singular

matrix \*inverse(const matrix \*m1);

//minimum matrix Chain Multiplication for a system of compatible matrices

int minmatrixOrder(int \*orderDim,int n);

int recursiveMinOrder(int \*orderDim,int start,int end);

//Gaussian elimination to solve linear system equations of a augmented matrix returns NULL if no solution unique exists

double\* GaussianELimationSolutions(matrix \* m);

//forward Elimination of a augmented matrix m

int forwardElimation(matrix \*m);

//backward substitution of the matrix to find the solution returns NULL if no solution unique exists

double \* backSubstitution(matrix \*m);

//Find eigenvalues of a matrix m

double \*eigenvalues(matrix \*m);

//Finding minimum eigen value and corresponding eigen vector using Rayleighs power method

matrix \* eigenVectorPower(matrix \*m);

//LU factorization of a matrix m parameters are m and two matrices o same order of m to store L and U

void LUfactorize(matrix \*m,matrix \*L,matrix \* U);

//Strassen's matrix multiplication

matrix \* strassen\_mul(const matrix \*m1,const matrix\* m2);

# IMPLEMENTATION

## STRUCTURES USED

//struct node which stores the number of rows, columns and the matrix as a two dimensional array

typedef struct matrix{

int c;

int r;

double \*\*m;

}matrix;

## PSEUDO CODE

//Creating a matrix of order = r\*c with r\*c elements

matrix \* create\_matrix(int c,int r,double \* elements)

{

int n = c\*r;

matrix \* m = (matrix \*)malloc(sizeof(matrix));

m->c = c;

m->r = r;

m->m = (double \*\*)malloc(sizeof(double \*)\*r);

int k = 0;

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

{

m->m[i] = (double\*)malloc(sizeof(double)\*c);

for(int j = 0;j<c;j++)

{

m->m[i][j] = elements[k++];

}

}

return m;

}

//Display a Matrix

void displayMat(const matrix \*m)

{

int r = m->r;

int c = m->c;

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

{

for(int j = 0;j<c;j++)

{

printf("%lf ",m->m[i][j]);

}

printf("\n");

}

}

//Find Determinant of a matrix m using DECREASE AND CONQUER

double determinant(const matrix \*m,int n)

{

double D = 0;

if (n == 1)

return m->m[0][0];

matrix\* temp;

int sign = 1;

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

{

// Getting Cofactor of A[0][f]

temp = getCofactor(m, 0, f,n);

D += sign \* m->m[0][f] \* determinant(temp, n - 1);

// terms are to be added with alternate sign

sign = -sign;

}

return D;

}

//Transpose a matrix m

matrix \*transpose(const matrix \*m)

{

matrix \* Mclone = (matrix \*)malloc(sizeof(matrix));

Mclone->r = m->c;

Mclone->c = m->r;

Mclone->m = (double \*\*)malloc(sizeof(double \*)\*(m->c));

for(int i = 0;i<(m->c);i++)

{

Mclone->m[i] = (double \*)calloc(m->r,sizeof(double));

}

for (int i = 0; i < m->r; ++i)

{

for (int j = 0; j < m->c; ++j)

{

Mclone->m[j][i] = m->m[i][j];

}

}

return Mclone;

}

//Find Inverse of the Matrix returns NULL if matrix is singular

matrix \*inverse(const matrix \*m1)

{

int n = m1->r;

// Find determinant of A[][]

double det = determinant(m1, n);

if (det == 0)

{

return NULL;

}

matrix \* m = (matrix \*)malloc(sizeof(matrix));

m->r = n;

m->c = n;

m->m = (double \*\*)malloc(sizeof(double \*)\*n);

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

{

m->m[i] = (double \*)calloc(n,sizeof(double));

}

// Find adjoint

matrix \* adj = adjoint(m1);

// Find Inverse using formula "inverse(A) = adj(A)/det(A)"

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

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

m->m[i][j] = adj->m[i][j]/det;

return m;

}

//minimum matrix Chain Multiplication for a system of compatible matrices

int recursiveMinOrder(int \*orderDim,int start,int end)

{

if(start == end)

return 0;

int k;

int min = INT\_MAX;

int count;

for (k = start; k <end; k++)

{

count = recursiveMinOrder(orderDim, start, k) +

recursiveMinOrder(orderDim, k+1, end) +

orderDim[start-1]\*orderDim[k]\*orderDim[end];

if (count < min)

min = count;

}

// Return minimum count

return min;

}

int minmatrixOrder(int \*orderDim,int n)

{

return recursiveMinOrder(orderDim,1,n-1);

}

//Strassen's matrix multiplication returns null in case matrix is not 2\*2

matrix \* strassen\_mul(const matrix \*m1,const matrix\* m2)

{

if(m1->r != 2 || m1->c != 2 || m2->r != 2 || m2->c != 2)

{

return NULL;

}

matrix \* result = (matrix \*)malloc(sizeof(matrix));

result->r = m1->r;

result->c = m2->c;

result->m = (double \*\*)malloc(sizeof(double \*)\*m1->r);

for(int i=0; i<m1->r; i++)

{

result->m[i] = (double \*)malloc(sizeof(double \*)\*(m2->c));

}

double a= (m1->m[0][0] + m1->m[1][1]) \* (m2->m[0][0] + m2->m[1][1]);

double b= (m1->m[1][0] + m1->m[1][1]) \* m2->m[0][0];

double c= m1->m[0][0] \* (m2->m[0][1] - m2->m[1][1]);

double d= m1->m[1][1] \* (m2->m[1][0] - m2->m[0][0]);

double e= (m1->m[0][0] + m1->m[0][1]) \* m2->m[1][1];

double f= (m1->m[1][0] - m1->m[0][0]) \* (m2->m[0][0]+m2->m[0][1]);

double g= (m1->m[0][1] - m1->m[1][1]) \* (m2->m[1][0]+m2->m[1][1]);

result->m[0][0] = a + d- e + g;

result->m[0][1] = c + e;

result->m[1][0] = b + d;

result->m[1][1] = a - b + c + f;

return result;

}

# CONCLUSION

Hence,Matrices play a vital role in implementing advanced mathematical problems and complex real life problems ,the design of algorithms for which becomes really simple using matrix as a datastructure.

The set of inbuilt functions we implemented helps us to modify and change and apply some functions on the matrix,hence helps is using such matrices in complex problems without having to redefine certain functions.

It was a wonderful and learning experience for me while working on this project. This project took me through the various phases of project development and gave me the real insight into the world of software engineering.

It was due to this project I came to know how professional libraries are designed.

# REFERENCES

https://www.codewithc.com/c-program-for-power-method/

GeekforGeeks

Design and analysis of algorithms by Anany Levitin.

Introduction to algorithms by Cormen.

Linear algebra by Gilbert Strang