Dokumentacja projektu nr. 2

Autor: Marcel Kaliński

Zajęcia: środa 14.15

# Wstęp

Zadanie polegało na napisaniu kalkulatora macierzowego. Dostępne operacje to mnożenia przez skalar, transponowanie, liczenie wyznacznika i odwracanie macierzy. Program miał opierać się o tablice.

# Specyfikacja

Najpierw zaprezentuję strukturę, w której przechowuje macierze. Zawiera ona 3 pola: liczba wierszy, liczba kolumn i tablicę 2D.

typedef struct mat {

unsigned int n\_rows;

unsigned int n\_cols;

float \*\*matrix;

}Matrix;

Następnie, aby móc odczytać macierz zadaną przez użytkownika należy w funkcji głównej podać ścieżkę do pliku tekstowego zawierającego naszą macierz. Powinna być ona w formacie:

1 2 3

2 4 1

5 2 1

Kolejne liczby muszą być oddzielone spacją. Macierz może być dowolnego kształtu, pod warunkiem, że maksymalna długość wierszy lub kolumn nie przekroczy 10.

Pierwszą operacją jest mnożenie przez skalar.

void scalar\_multiply(Matrix mat, float scalar)

{

for(int i=0; i<mat.n\_rows; i++)

{

for(int j=0; j<mat.n\_cols; j++)

{

mat.matrix[i][j] \*= scalar;

}

}

}

Następnie transponowanie.

Matrix transpose(Matrix mat)

{

int n\_rows = mat.n\_rows;

int n\_cols = mat.n\_cols;

Matrix new\_mat = create\_matrix(n\_cols, n\_rows);

float \*\*new\_matrix = new\_mat.matrix;

float \*\*matrix = mat.matrix;

for(int r=0; r<n\_rows; r++)

{

for(int c=0; c<n\_cols; c++)

{

new\_matrix[c][r] = matrix[r][c];

}

}

return new\_mat;

}

Trochę trudniejszym zadaniem jest wyznaczenie wyznacznika macierzy. Zaimplementowałem to korzystając z metody eliminacji Gaussa. Kiedy doprowadzimy macierz do postaci trójkątnej, czyli kiedy pod diagonalą będą same zera, to iloczyn diagonali da nam wyznacznik macierzy.

float det(const Matrix mat)

{

Matrix m = gaussian\_eliminate(mat);

float \*\*matrix = m.matrix;

if(m.n\_cols != m.n\_rows)

{

printf("n\_rows != n\_cols\n");

exit(EXIT\_FAILURE);

}

float val = 1;

for(int i=0; i < m.n\_rows; i++)

{

val \*=matrix[i][i];

}

return val;

}

Podobnym sposobem zaimplementowałem wyznaczanie macierzy odwrotnej. Również posłużyłem się eliminacją Gaussa, jednak w tym przypadku nie można skończyć algorytmu na macierzy trójkątnej, lecz trzeba doprowadzić macierz do takiej postaci, że na diagonali będą jedynki a poza nią zera. Wtedy macierz dołączana będzie naszą poszukiwaną macierzą odwrotną.

Matrix invert(const Matrix mat)

{

int n\_rows = mat.n\_rows;

int n\_cols = mat.n\_cols;

float \*\*matrix = mat.matrix;

if(n\_rows != n\_cols)

{

printf("macierz musi byc kwadratowa!!\n");

exit(EXIT\_FAILURE);

}

Matrix result\_mat = create\_matrix(n\_rows, n\_cols);

Matrix B = create\_matrix(n\_rows, n\_cols);

initialise\_zeros(B);

for(int i=0; i<B.n\_cols; i++)

{

B.matrix[i][i] = 1;

}

initialise\_like(mat, result\_mat);

float \*\*tmp = result\_mat.matrix;

float \*\*B\_tmp = B.matrix;

int c = 0;

// pierwsza iteracjia, z góry do dołu

for(; c<n\_cols; c++)

{

int r;

for(r=c+1; r<n\_rows; r++)

{

float L = tmp[r][c]/tmp[c][c];

for(int i=c; i<n\_cols; i++)

{

tmp[r][i] = tmp[r][i] - L\*tmp[c][i];

}

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

B\_tmp[r][i] = B\_tmp[r][i] - L\*B\_tmp[c][i];

}

}

// druga iteracja, od dołu do góry

for(c=n\_cols-1; c>0; c--)

{

int r;

for(r=c-1; r>=0; r--)

{

float L = tmp[r][c]/tmp[c][c];

for(int i=c; i>=0; i--)

{

tmp[r][i] = tmp[r][i] - L\*tmp[c][i];

}

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

B\_tmp[r][i] = B\_tmp[r][i] - L\*B\_tmp[c][i];

}

}

// mnożymy wiersze, aby po lewej były same "1" na diagonali

for(c=0; c<n\_cols; c++)

{

float L = 1 / tmp[c][c];

tmp[c][c] \*= L;

for(int r=0; r<n\_rows; r++)

B\_tmp[c][r] \*= L;

}

free\_matrix(result\_mat);

return B;

}