Main.cpp

#include "matrix.h"

#include <cstdlib>

#include <iomanip>

#include <stdio.h>

#include <time.h>

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Nathan Redmond

\* COSC 320 Project 1

\* Created: 10 March 2019

\* Modified: 24 March 2019

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int main(int argc, char\*\* argv)

{

int num\_rows = 0;

int num\_cols = 1;

if( argc < 2 )

cout << "No file as argument available" << endl;

else

{

std::string fname = std::string(argv[1]);

ifstream inFile;

inFile.open(fname);

if (!inFile)

{

cout << "Can't open input file " << fname << endl;

return 1;

}

char temp[100];

while(!inFile.eof())

{

inFile.getline(temp, 100, '\n');

num\_rows++;

}

num\_rows = num\_rows - 1;

for(int i = 0; i < 100; i++)

{

if(temp[i] == ' ')

num\_cols++;

}

if( num\_cols < 2 )

{

cout << "One dimention input. Can not calculate OLS." << endl;

return 1;

}

inFile.close();

inFile.open(fname);

Matrix POINTS(num\_rows, num\_cols);

double point;

for(int x = 0; x < num\_rows; x++)

for(int y = 0; y < num\_cols; y++)

{

inFile >> point;

POINTS[x][y] = point;

}

Matrix t = POINTS.GetBetta();

Matrix OLS = POINTS.OLS(t);

cout << "LEAST SQUARES LINEAR REGRESSION: " << endl;

for(int t = OLS.getRows()-1; t >= 0; t--)

{

if( t == 0 )

cout << OLS[OLS.getRows()-1][0] << endl;

else if( t == 1 )

cout << OLS[OLS.getRows()-2][0] << "x + ";

else

cout << OLS[OLS.getRows()-1-t][0] << "x^" << t << " + ";

}

inFile.close();

}

srand (time(NULL));

int oper, rows, cols, rows2, cols2;

bool operation = true;

cout << "CREATE TWO MATRICIES FOR TESTING." << endl;

cout << "First Matrix number of rows: ";

cin >> rows;

cout << "First Matrix number of columns: ";

cin >> cols;

cout << "Second Matrix number of rows: ";

cin >> rows2;

cout << "Second Matrix number of columns: ";

cin >> cols2;

cout << endl;

Matrix a(rows, cols);

Matrix b(rows2, cols2);

a.randomizeMatrix();

b.randomizeMatrix();

Matrix c(a.getRows(), b.getCols());

cout << "Matrix a" << endl;

a.printMatrix();

cout << endl;

cout << "Matrix b" << endl;

b.printMatrix();

cout << endl;

cout << "ADD a+b" << endl;

c = a+b;

c.printMatrix();

cout << endl;

cout << "SUBTRACT a-b" << endl;

c = a-b;

c.printMatrix();

cout << endl;

cout << "MULTIPLY a\*b" << endl;

Matrix Mul(a.getRows(), b.getCols());

Mul = a\*b;

Mul.printMatrix();

cout << endl;

cout << "SCALAR of a" << endl;

Matrix Scal(a.getRows(), a.getCols());

Scal = a.scaleMatrix(3);

Scal.printMatrix();

cout << endl;

cout << "TRANSPOSE of a" << endl;

Matrix t(a.getCols(), a.getRows());

t = a.Transpose();

t.printMatrix();

cout << endl;

//----------------------------------------------------

cout << "BETTA of a" << endl;

Matrix temp = a.GetBetta();

temp.printMatrix();

cout << endl;

cout << "OLS of the Matrix of Points: " << endl;

Matrix lls = a.OLS(temp);

lls.printMatrix();

cout << endl;

}

Matrix.cpp

#include <iostream>

using namespace std;

#include <stdlib.h>

#include <math.h>

#include <cmath>

#include "matrix.h"

Matrix::Matrix(int r, int c)

{

rows = r;

cols = c;

m = new double\*[rows];

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

m[i] = new double[cols];

for(int x=0; x<rows; x++)

for(int y=0; y<cols; y++)

{

m[x][y] = 0;

}

}

Matrix::Matrix()

{

rows = 10;

cols = 10;

m = new double\*[rows];

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

m[i] = new double[cols];

for(int x=0; x<rows; x++)

for(int y=0; y<cols; y++)

{

m[x][y] = 0;

}

}

Matrix::Matrix(const Matrix &temp)

{

rows = temp.rows;

cols = temp.cols;

m = new double\*[rows];

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

{

m[i] = new double[cols];

}

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

{

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

{

m[i][j] = temp.m[i][j];

}

}

}

Matrix::~Matrix()

{

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

delete [] m[i];

delete [] m;

}

double Matrix::getRows()

{

return rows;

}

double Matrix::getCols()

{

return cols;

}

Matrix& Matrix::operator =(const Matrix &b)

{

if (this != &b)

{

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

delete [] m[i];

delete [] m;

rows = b.rows;

cols = b.cols;

m = new double\*[rows];

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

{

m[i] = new double[cols];

}

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

{

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

{

m[i][j] = b.m[i][j];

}

}

}

return (\*this);

}

Matrix Matrix::operator +( Matrix & a )

{

if (a.getRows()!= this->getRows()|| a.getCols() != this->getCols() )

{

std:: cout << "error: Sizes are different" << std:: endl;

return \*this;

}

Matrix b(rows, cols);

for ( int i = 0 ; i < a.getRows(); i++ )

{

for (int j = 0; j < a.getCols(); j++ )

{

b.m[i][j] = double(this->m[i][j]+a.m[i][j]);

}

}

return b;

}

Matrix Matrix::operator -( Matrix& a )

{

if (a.getRows()!= this->getRows()|| a.getRows() != this->getCols() )

{

std:: cout << "error: Sizes are different" << std:: endl;

return \*this;

}

Matrix b(getRows(),getCols());

for ( int i = 0 ; i < a.getRows(); ++i )

{

for (int j = 0; j < a.getCols(); ++j )

{

b.m[i][j] = double(this->m[i][j] - a.m[i][j]);

}

}

return b;

}

//template <typename T>

Matrix Matrix::operator \*(Matrix& a)

{

if( this->getCols() == a.getRows() )

{

Matrix c(this->getRows(), a.getCols());

for( int i = 0; i < this->rows; ++i)

for(int j = 0; j < a.cols; ++j)

for( int k = 0; k < a.rows; ++k)

c.m[i][j] += double(this->m[i][k] \* a.m[k][j]);

return c;

}

else

{

cout << "error: Unmatching matrix sizes." << endl;

return \*this;

}

}

double\* & Matrix::operator [](const int &index) const // overloading operator []

{

return m[index];

}

Matrix Matrix::Transpose() const

{

//int Size=Columns\*Rows;

Matrix Temp(cols, rows);

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

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

Temp[j][i] = m[i][j];

return Temp;

}

void Matrix::resize(int newRow, int newCol)

{

rows = newRow;

cols = newCol;

}

double Matrix::getEntry( int i, int j) {

if (i >= getRows() || j >= getCols() || i < 0 || j < 0)

{

std::cout << "error: Wrong index, matrix size is not matching" << std::endl;

return 00;

}

else

return m[i][j];

}

void Matrix::setEntry( int i, int j, double val)

{

if (i >= getRows() || j >= getCols() || i < 0 || j < 0)

{

std::cout << "error: Wrong index, matrix size is not matching" << std::endl;

return;

}

else

m[i][j] = val;

}

void Matrix::randomizeMatrix()

{

for(int i = 0; i < this->rows; i++)

for(int j = 0; j < this->cols; j++)

{

this->m[i][j] = rand()%10;

}

}

Matrix Matrix::scaleMatrix(double num)

{

Matrix temp(rows, cols);

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

{

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

{

temp[i][j] = double(num \* m[i][j]);

}

}

return temp;

}

Matrix Matrix::Split( int TLRow, int TLCol, int BRRow, int BRCol )

{

int size\_rows = BRRow - TLRow;

int size\_cols = BRCol - TLCol;

Matrix temp(this->rows/2, this->cols/2);

for(int x = TLRow; x <= BRRow; x++)

for(int y = TLCol; y <= BRCol; y++)

{

temp.m[x - TLRow][y - TLCol] = m[x][y];

}

return temp;

}

Matrix Matrix::PadMatrix(int power)

{

if(ceil(log2(rows)) == floor(log2(rows)))

if ( ceil(log2(cols)) == floor(log2(cols)) )

return (\*this);

Matrix temp(pow(2, power), pow(2, power));

for(int x = 0; x < temp.getRows(); x++)

for(int y = 0; y < temp.getCols(); y++)

{

if(x == y)

temp.m[x][y] = 1;

else

temp.m[x][y] = 0;

}

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

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

{

temp.m[i][j] = m[i][j];

}

return temp;

}

Matrix Matrix::Inverse()

{

if( this->getRows() == 1 )

{

Matrix inv(this->getRows(), this->getCols());

inv[0][0] = 1 / this->m[0][0];

return inv;

}

int power = 0;

while(this->getRows() > pow(2, power))

{

power++;

}

Matrix TEMP = this->PadMatrix(power);

Matrix B = TEMP.Split(0, 0, (TEMP.rows/2)-1, (TEMP.cols/2)-1);

Matrix C = TEMP.Split(TEMP.rows/2, 0, TEMP.rows-1, (TEMP.cols/2)-1);

Matrix CT = TEMP.Split(0, TEMP.cols/2, (TEMP.rows/2)-1, TEMP.cols-1);

//Matrix CT = C.Transpose();

Matrix D = TEMP.Split(TEMP.rows/2, TEMP.cols/2, TEMP.rows-1, TEMP.cols-1);

Matrix BInv = B.Inverse();

Matrix W = C\*BInv;

Matrix WT = BInv\*CT;

Matrix X = W\*CT;

Matrix S = D-X;

Matrix V = S.Inverse();

Matrix Y = V\*W;

Matrix YT = Y.Transpose();

Matrix T = YT.scaleMatrix(-1);

Matrix U = Y.scaleMatrix(-1);

Matrix Z = WT\*Y;

Matrix R = BInv + Z;

Matrix Temp\_Inv(TEMP.rows, TEMP.cols);

for(int a = 0; a < TEMP.rows/2; a++)

for(int b = 0; b < TEMP.cols/2; b++)

{

Temp\_Inv[a][b] = R[a][b];

}

for(int c = TEMP.rows/2; c < TEMP.rows; c++)

for(int d = 0; d < TEMP.cols/2; d++)

{

Temp\_Inv[c][d] = U[c-TEMP.cols/2][d];

}

for(int e = 0; e < TEMP.rows/2; e++)

for(int f = TEMP.cols/2; f < TEMP.cols; f++)

{

Temp\_Inv[e][f] = T[e][f-TEMP.cols/2];

}

for(int g = TEMP.rows/2; g < TEMP.rows; g++)

for(int h = TEMP.cols/2; h < TEMP.cols; h++)

{

Temp\_Inv[g][h] = V[g-TEMP.rows/2][h-TEMP.cols/2];

}

Matrix resized(this->getRows(), this->getCols());

for(int i = 0; i < this->getRows(); i++)

for(int j = 0; j < this->getCols(); j++)

{

resized[i][j] = Temp\_Inv[i][j];

}

return resized;

}

Matrix Matrix::GetBetta()

{

int colNumb = this->cols - 1;

Matrix B(this->rows, 1);

for( int i = 0; i < this->rows; i++ )

{

B.m[i][0] = this->m[i][colNumb];

}

return B;

}

Matrix Matrix::OLS( Matrix& B )

{

Matrix A(this->rows, this->cols);

for( int i = 0; i < this->rows; i++ )

for( int j = 0; j < this->cols; j++ )

{

if( j == this->cols - 1 )

A.m[i][j] = 1;

else

A.m[i][j] = this->m[i][j];

}

Matrix I = (A.Transpose() \* A);

Matrix IIN = I.Inverse();

Matrix AT = A.Transpose();

Matrix T = IIN\*AT;

Matrix TB = T \* B;

return TB;

}

void Matrix::printMatrix()

{

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

{

for(int x=0; x<getCols(); x++)

{

cout << m[i][x] << " ";

}

cout << endl;

}

}

Matrix.h

#ifndef MATRIX\_H

#define MATRIX\_H

class Matrix{

private:

double \*\*m;

int rows;

int cols;

public:

Matrix();

Matrix(int n\_rows, int n\_cols);

Matrix(const Matrix &temp); //copy constructor

~Matrix();

double getRows();

double getCols();

double getEntry(int i, int j);

void setEntry(int i, int j, double val);

void printMatrix();

void resize(int newRow, int newCol);

void randomizeMatrix();

Matrix Transpose() const;

double\* & operator[](const int &index) const;

Matrix& operator =(const Matrix &b);

Matrix operator +(Matrix& b);

Matrix operator -(Matrix& b);

Matrix operator \*(Matrix& b);

Matrix Split(int TLRow, int TLCol, int BRRow, int BRCol);

Matrix OLS(Matrix& B);

Matrix Inverse();

Matrix PadMatrix(int power);

Matrix scaleMatrix(double num);

Matrix GetBetta();

};

#endif

OUTPUT:

Nates-MacBook-Pro:Project1Redmond nateredmond$ make

g++ -c main.cpp

g++ -c matrix.cpp

g++ -std=c++11 main.o matrix.o -o Project1

Nates-MacBook-Pro:Project1Redmond nateredmond$ ./Project1 points100.txt

LEAST SQUARES LINEAR REGRESSION:

1.97942x + 4.78614

CREATE TWO MATRICIES FOR TESTING.

First Matrix number of rows: 4

First Matrix number of columns: 4

Second Matrix number of rows: 4

Second Matrix number of columns: 4

Matrix a

4 9 4 8

4 8 2 6

0 7 6 5

3 9 2 1

Matrix b

9 3 1 2

9 1 2 8

2 6 4 2

0 9 7 0

ADD a+b

13 12 5 10

13 9 4 14

2 13 10 7

3 18 9 1

SUBTRACT a-b

-5 6 3 6

-5 7 0 -2

-2 1 2 3

3 0 -5 1

MULTIPLY a\*b

125 117 94 88

112 86 70 76

75 88 73 68

112 39 36 82

SCALAR of a

12 27 12 24

12 24 6 18

0 21 18 15

9 27 6 3

TRANSPOSE of a

4 4 0 3

9 8 7 9

4 2 6 2

8 6 5 1

BETTA of a

8

6

5

1

OLS of the Matrix of Points:

2.85714

-2.14286

2.07143

7.57143

Nates-MacBook-Pro:Project1Redmond nateredmond$

Nates-MacBook-Pro:Project1Redmond nateredmond$ ./Project1 points100-3d.txt

LEAST SQUARES LINEAR REGRESSION:

0.00755109x^2 + 1.00559x + -0.0127631

CREATE TWO MATRICIES FOR TESTING.

First Matrix number of rows: 3

First Matrix number of columns: 3

Second Matrix number of rows: 3

Second Matrix number of columns: 3

Matrix a

7 5 0

0 8 8

3 0 9

Matrix b

1 2 1

4 9 3

2 1 7

ADD a+b

8 7 1

4 17 11

5 1 16

SUBTRACT a-b

6 3 -1

-4 -1 5

1 -1 2

MULTIPLY a\*b

27 59 22

48 80 80

21 15 66

SCALAR of a

21 15 0

0 24 24

9 0 27

TRANSPOSE of a

7 0 3

5 8 0

0 8 9

BETTA of a

0

8

9

OLS of the Matrix of Points:

-1.42553

-0.659574

13.2766

Nates-MacBook-Pro:Project1Redmond nateredmond$