Table of contents

* Analysis 3
  + Problem Area 3
  + interview 3
  + End users 3
  + List of objectives 4
  + Modelling
* Design 6
  + System Flowchart (general) 6
  + Structure tables (general desc of each file) 7
  + Matrix calculator 10
    - File breakdown
    - Flowcharts of methods
  + simultaneous equations solver 17
  + Simplex algorithm 20
    - File breakdown
    - Flowcharts of methods
* Technical solution 31
  + file structure 31
  + code 32
    - arrays.cpp 32
    - matrix.cpp 33
    - polynomial-solver.cpp 40
    - simplex-method.cpp 43
    - simultaneous-equations.cpp 51
    - GUI 54
      * addon.cpp 54
      * binding.gyp 56
      * dataparser.cpp 57
      * operation-handler.cpp 64
      * server.js 74
      * views 80
        + matrix.ejs 80
        + simplex.ejs 92
        + simul.ejs 103
        + menu.ejs 112
* Testing 113
  + Matrix calculator 113
    - multiplication 113
    - addition 114
    - subtraction 115
    - determinant 116
    - inverse 117
    - transpose 118
    - minor 119
    - multiply by constant 120
  + simultaneous equations 121
  + Simplex-method 124
  + dynamic UI 125
* Evaluation 130
  + Objectives 132
  + feedback 133
  + comments on feedback 133
  + improvements 133
* glossary 134

**Analysis**

**Problem area:**

The problem I aim to solve is the lack of an easily accessible tool that contains all the necessary tools that an A-level further mathematics student might need in order to aid him during his study. This includes a matrix calculator, a simplex algorithm solver, a simultaneous equations solver, a polynomial root finder and a graph plotter.

**Interview:**

interviewee: Giles Murphy, A level further mathematics student.

What do you use to check your answers when revising further maths topics such as matrices, the simplex algorithm?

I usually use the answer section in the textbook but the answers aren’t always correct and only provide the final solution.

How could this program make revising these topics easier?

Having a program that contains all the tools for further maths would be quite useful when going through my working and trying to see where I made an error. This would be especially useful when applying the simplex algorithm because it is very easy to make a mistake when solving one and very difficult to find where you made the mistake when you make one.

How form should the program take?

The program should be in the form of a website because that will make it easy to access from all devices without having to worry about updates to the software or platform compatibility.

What should the matrix calculator support?

Matrix addition, matrix subtraction, matrix multiplication, finding the determinant of matrices, finding the inverse of matrices and finding the minor of a matrix at any given index. There should also be a system to easily reuse the output of a matrix operation as an input to another matrix operation.

**End users:**

The end users are students who are taking further mathematics as an A-level or teachers who are teaching further mathematics A-level. They should be able to access the website from anywhere on any browser and be able to easily input the relevant data into the input fields and be able to easily read the output of the process they used.

**Objectives:**

1. General
   1. The program should be in the form of a website
   2. The website should have an easy-to-use interface
   3. The website should be compatible with all widely used browsers
2. Matrix calculator
   1. The matrix calculator should be able to handle matrices of any order
   2. The matrix calculator should be able to perform the following operations
      1. Matrix multiplication
      2. Matrix addition
      3. Matrix subtraction
      4. Finding the inverse of a matrix
      5. Finding the determinant of a matrix
      6. Transposing a matrix
      7. Finding the minor of a matrix from any element in that matrix
      8. multiply a matrix by a constant
   3. The matrix calculator should be return relevant error messages when the user enters an input that is invalid for the operation he wants to use
   4. The matrix calculator should have a button for each matrix operation
   5. The values in each input matrix should remain after an operation, successful or unsuccessful so that the user can reuse them
   6. The user should be able to copy the output of an operation to one of the input matrices and vice-versa
   7. The user should be able to swap the input matrices
3. Simplex algorithm
   1. The simplex solver should be able to accept inputs that are in standard form
   2. The website should be able to generate a table based on the number of variables, slack variables, artificial variables and constraints for the user to input his data
   3. The output of a simplex algorithm should be in text format and easy to read.
4. Polynomial root finder
   1. The user should be able to enter a polynomial in standard form
   2. The root finding algorithm should be able to find the roots to 10 significant figures
   3. The output should be displayed in text format

**Design**

**General data flow flowchart:**

Diagram

Description automatically generated

**Structure tables:**

**Library files:**

|  |  |  |
| --- | --- | --- |
| **Filename** | **Functions** | **description** |
| **matrix.cpp** | **inversematrix**  **transposematrix**  **finddeterminant**  **findminor**  **multiplymatrices**  **multiplybyconstant**  **addmatrices**  **printmatrix** | **This file is a library of all the core functions of the matrix calculator which can be called from another file.** |
| **simplex-method.cpp** | **apply\_simplex**  **convert\_to\_standard\_form**  **minimise\_objective\_function**  **maximise\_objective\_function**  **perform\_iteration**  **divide\_pivot\_row**  **get\_pivot\_row**  **get\_pivot\_column**  **printtableau**  **read\_simplex**  **printsolution** | **This file is a library of all the core functions of the simplex solver which can be called from another file.** |
| **simultaneous-equations-solver.cpp** | **SimulSolve**  **printresults**  **printtable** | **This file is a library of all the core functions of the simultaneous equations solver which can be called from another file.** |
| **Arrays.cpp** | **initializedouble2dpointerarray**  **copydouble2dpointerarray**  **initializeint2dpointerarray** | **Library file that contains c++ two-dimensional array initialization functions.** |
| **polynomial-solver.cpp** | **TBC** | **TBC** |

**Graphical user interface files:**

|  |  |  |
| --- | --- | --- |
| **Filename** | **Functions/ processes** | **description** |
| **server.js** | **app.listen**  **app.get(“/matrix”)**  **app.get(“/simul”)**  **app.get(“/simplex”)**  **app.post(“/matrix”)**  **app.post(“/simul”)**  **app.post(“/simplex”)**  **MatrixString** | **This file handles initializes the node js website and handles GET and POST requests to all pages on the website. It also sends POST data to addon.cpp and back to the .ejs files with the output of the called library functions.** |
| **matrix.ejs** | **fixed HTML structures**  **startup**  **SwapAandB**  **CopyPasteMatrix**  **GenerateMatrix**  **ConvertTo2dArray**  **DeleteColumn**  **DeleteRow**  **CreateColumn**  **CreateRow**  **CreateCell**  **UpdateSizeValues**  **ConvertStringToMat** | **This file contains the HTML structures and inline CSS that determine the layout of the matrix calculator page.**  **It also contains functions involved in generating tables and parsing data that is output from the server.js file as a result of a POST request being processed into an editable table.** |
| **simul.ejs** | **fixed HTML structures**  **startup**  **GenerateMatrix**  **ConvertTo2dArray**  **DeleteColumn**  **DeleteRow**  **CreateColumn**  **CreateRow**  **CreateCell**  **UpdateSizeValues**  **ConvertStringToMat** | **This file contains the HTML structures and inline CSS that determine the layout of the simultaneous equations solver page.**  **It also contains functions involved in generating tables and parsing data that is output from the server.js file as a result of a POST request being processed into a text area.** |
| **simplex.ejs** | **fixed HTML structures**  **startup**  **GenTable**  **AddHeaderElement**  **GenerateMatrix**  **ConvertTo2dArray**  **DeleteColumn**  **DeleteRow**  **CreateColumn**  **CreateRow**  **CreateCell**  **UpdateSizeValues**  **ConvertStringToMat** | **This file contains the HTML structures and inline CSS that determine the layout of the simplex solver page.**  **It also contains functions involved in generating tables and parsing data that is output from the server.js file as a result of a POST request being processed into a text area.** |

**intermediate input data parsing files:**

|  |  |  |
| --- | --- | --- |
| **Filename** | **Functions/processes** | **Description** |
| **addon.cpp** | **Main**  **Init** | **This file is the entrypoint to the c++ library files with all the functionality. It receives a string as input from server.js that is then passed to operation-handler.cpp and is decoded into relevant data structures that can be used to carry out operations. It also returns the string encoded with the result of the operations back to server.js.** |
| **operation-handler.cpp** | **DoOperation**  **DoSimplex**  **DoSimul**  **DoMatrixOperation** | **This file’s entrypoint is the DoOperation function where the string is partially decoded using dataparser.cpp. The decoded data is used to decide which operation will be called (e.g. simplex or matrixoperation or simul) where the string will be further decoded into the relevant data structures to carry be used by the c++ library files.** |
| **dataparser.cpp** | **parseTableau**  **ParseStringTo2dArray**  **parseSimulString**  **MakeMatrix**  **split**  **ConvertMatToString**  **IsDouble**  **IsInteger**  **SliceAlphabet** | **This file contains the functions that are used in decoding the string data into the relevant data structures e.g. matrix, simplex tableau, simultaneous equation matrix. This file is solely used by operation-handler.cpp to parse data.** |
| **binding.gyp** |  | **This file is used to compile the c++ files into executable files that can be used by server.js . This is where addon.cpp is designated as the entry-point for the c++ files.** |

**Matrix.cpp:**

**File breakdown:**

**note - The matrix inputs and outputs are in the form of an array of pointer arrays of doubles.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **inputs** | **outputs** | **description** |
| **inversematrix** | **matrix, matrix height, matrix width** | **inverse matrix** | **The inverse matrix is found by finding the matrix of cofactors of the input, transposing it, and multiplying it by 1/det where det is the determinant of the input matrix** |
| **matrixofcofactors** | **m, height, width** | **matrix of cofactors** | **The function iterates through each element of the matrix, finds the determinant of the minor and multiplies it by the -1 if appropriate for the element.** |
| **transposematrix** | **matrix, matrix height, matrix width** | **transposed matrix** | **The input matrix is reflected along the leading diagonal\* such that the index of an element in the matrix goes from being i,j to j,i.** |
| **finddeterminant** | **matrix, matrix height, matrix width** | **determinant of matrix as a double** | **If the order of the matrix is less than 3x3 then the function simply calculates the determinant using a formula. If the order of the matrix is >= 3x3 then the function finds the minor of the matrix for each element along it’s top row and recursively calls itself until each minor is reduced to a 2x2 matrix. Through this recursion, the function can find the determinant of a square matrix of any order.** |
| **findminor** | **matrix, matrix height, matrix width, index i, index j** | **minor matrix** | **This function returns the minor of a matrix at any given index that is in the matrix. It does this by overwriting the row and column that the chosen element is in with the rows and column in front and below it. The width and the height of the output matrix are just reduced by 1 each.** |
| **mutiplymatrices** | **matrix1, matrix2, m1height, m1width, m2height, m2width** | **matrix** | **This function will iterate through each row of m1, and for each row of m1 iterate through each column of m2 and multiply the respective elements of each row column pair and then add them together.** |
| **multiplybyconstant** | **matrix, height, width, constant** | **matrix** | **This function will iterate through each element of the input matrix and multiply them by the constant.** |
| **addmatrices** | **matrix1, matrix2, m1rows, m1cols, m2rows, m2cols** | **matrix** | **This function will iterate through each row and column and add each respective element of m1 and m2 together to form the output matrix.** |
| **printmatrix** | **matrix, height, width** | **void** | **This function is for testing purposes. It will iterate through each element in the input matrix and output it to the console with a space and an “endl” at the appropriate places to make the output be in the form a matrix in the console.** |

**In depth flowchart of high complexity functions:**

**inversematrix flowchart:**

A picture containing diagram

Description automatically generated

**matrixofcofactors flowchart:**

Diagram

Description automatically generated

**finddeterminant flowchart:**

Diagram

Description automatically generated

**findminor flowchart:**

Diagram

Description automatically generated

**simultaneous-equations-solver.cpp**

**File breakdown:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **inputs** | **outputs** | **description** |
| **SimulSolve** | **double matrix of coefficients, integer number of variables** | **list of solutions in order** | **The matrix of coefficients will first be split into a matrix of the right hand side of the equations and a matrix of the left hand side of the equations. Then the inverse of the coefficients of the LHS will be found and post-multiplied by the RHS, giving the solution to each variable.** |
| **printresults** | **list of variable names, list of results, number of variables** | **string displaying the results in a legible way.** | **The function iterates through each list and creates a string with each variable name followed by “ = “ followed by the respective result.** |
| **printable** | **list of variable names, matrix of coefficients, number of variables** | **string** | **This functions is for testing purposes. It iterates through the matrix of coefficients and unpacks it into the same format than that of the original equations that were entered in the UI.** |

**SimulSolve flowchart:**

Diagram

Description automatically generated

**simplex-method.cpp**

**File breakdown:**

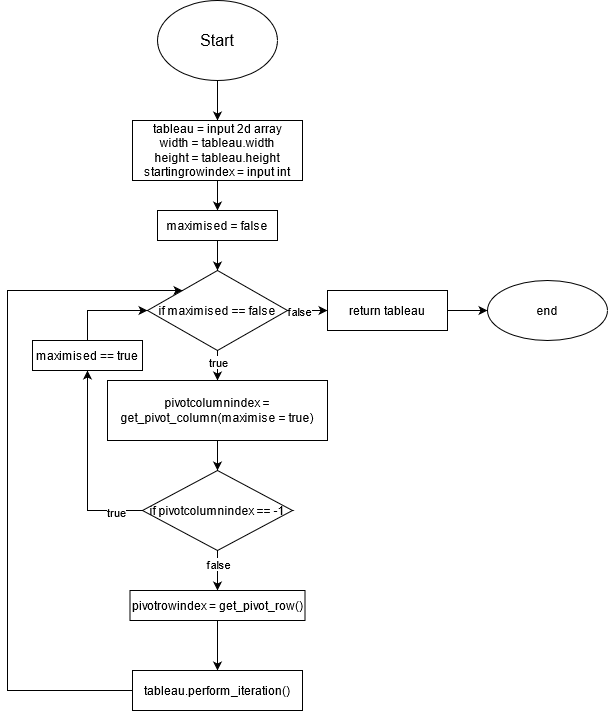
|  |  |  |  |
| --- | --- | --- | --- |
| **function** | **inputs** | **outputs** | **description** |
| **apply\_simplex** | **simplex tableau, numofvariables, numofconstraints, numofslackbariabels, numofartificialvariables, startingrowindex, variablenames, resultsstring** | **final tableau** | **This function detects whether a one-stage or two-stage simplex method is to be applied and then applies it using the maximization and minimization functions to produce the solved simplex tableau.** |
| **convert\_to\_standard\_form** | **simplex tableau, width, height, numofartificialvariables** | **standard form tableau** | **Deletes each artificial variable column and the artificial variable row.**  **This is used when going from a two-stage simplex to the one-stage phase.** |
| **minimise\_objective\_function** | **simplex tableau, width ,height, startingrowindex** | **minimized tableau** | **This function minimizes the objective function which is the first row in the tableau by performing simplex iterations until minimized.** |
| **maximise\_objective\_function** | **simplex tableau, width ,height, startingrowindex** | **maximized tableau** | **This function maximizes the objective function which is the first row in the tableau by performing simplex iterations until minimized.** |
| **perform\_iteration** | **simplex tableau ,pivot column, pivotrow, width ,height** | **tableau after one iteration** | **This function iterates through each constraint row and multiplies it by the relevant multiplier based on the pivot row.** |
| **divide\_pivot\_row** | **simplex tableau, pivot column, pivot row, width** | **tableau after pivot row is divided** | **This function divides the pivot row** |
| **get\_pivot\_row** | **simplex tableau, pivot column, height, right-hand-side column index, starting row index** | **index** | **this function finds the pivot row to apply the next simplex iteration to.** |
| **get\_pivot\_column** | **simplex tableau, width ,height** | **index** | **this function finds the pivot column to apply the next simplex iteration to.** |
| **printtableau** | **simplex tableau, width ,height** | **void** | **This is a function that prints a simplex tableau at any given point and is used for testing purposes** |
| **read\_simplex** | **simplex tableau, width ,height** | **double array containing solutions** | **This function reads a completed simplex tableau and outputs the values of each variable** |
| **printsolution** | **list of variable names, list of solutions, number of solutions, output string of solutions** | **void** | **This function formats the output of the read\_simplex function into text that is easily readable.** |

**apply\_simplex flowchart:**

Diagram

Description automatically generated with low confidence

**maximise\_objective\_function flowchart:**



**minimise\_objective function flowchart:**Diagram

Description automatically generated

**convert\_to\_standard\_form flowchart:**Diagram

Description automatically generated

**perform\_iteration flowchart:**

Diagram

Description automatically generated

**divide\_pivot\_row flowchart:**

Diagram

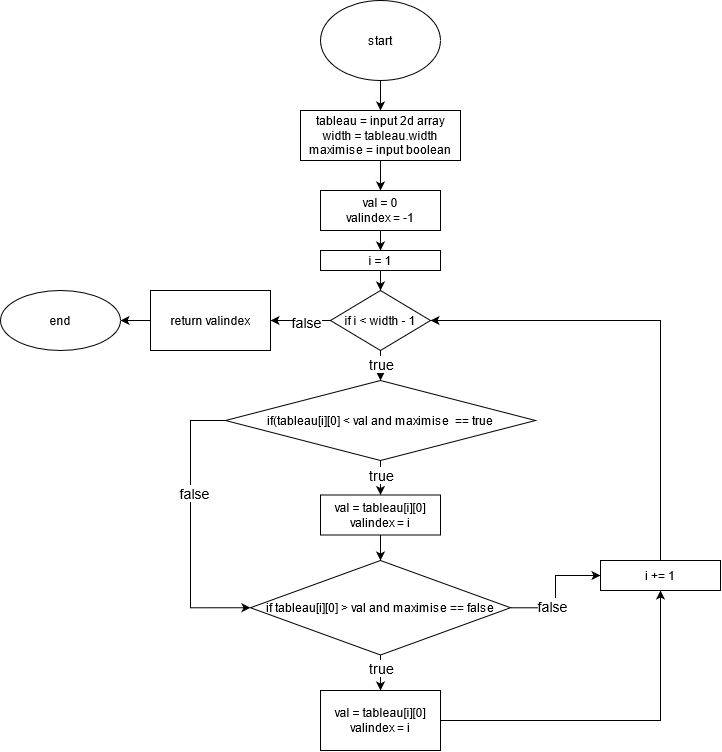
Description automatically generated

**get\_pivot\_row flowchart:**

Diagram

Description automatically generated

**get\_pivot\_column flowchart:**



**Technical solution:**

**file structure:**

**maths-toolkit:**

**│ arrays.cpp**

**│ matrix.cpp**

**│ polynomial-solver.cpp**

**│ simplex-method.cpp**

**│ simultaneous-equations-solver.cpp**

**│**

**└───GUI**

**│ addon.cpp**

**│ binding.gyp**

**│ dataparser.cpp**

**│ operation-handler.cpp**

**│ server.js**

**└───views**

**-- matrix.ejs**

**-- simplex.ejs**

**-- simul.ejs**

**-- menu.ejs**

Code:

arrays.cpp:

//start of code

double\*\* initializedouble2dpointerarray(int numarrays, int numelements) {

double\*\* arr = new double\*[numarrays];

for (int i = 0; i < numarrays; i++) {

arr[i] = new double[numelements];

}

return arr;

}

double\*\* copydouble2dpointerarray(double\*\* m, int numarrays, int numelements) {

double\*\* temp = initializedouble2dpointerarray(numarrays, numelements);

for(int x = 0; x < numarrays; x++) {

for (int y = 0; y < numelements; y++) {

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

}

}

return temp;

}

int\*\* initializeint2dpointerarray(int numarrays, int numelements) {

int\*\* arr = new int\*[numarrays];

for (int i = 0; i < numarrays; i++) {

arr[i] = new int[numelements];

}

return arr;

}

//end of code

matrix.cpp:

//start of code

//#include <iostream>

#include <stdexcept>

#include <cmath>

//#include "arrays.cpp"

using namespace std;

void printmatrix(double\*\* matrix, int height, int width) {

for (int j = 0; j < height; j++) {

cout << endl;

for (int i = 0; i < width; i++) {

cout << matrix[i][j] << " ";

}

}

cout << endl << endl << endl;

}

double\*\* addmatrices(double\*\* m1, double\*\* m2, int m1rows, int m1cols, int m2rows, int m2cols)

{

if (m1rows != m2rows || m1cols != m2cols)

{

throw invalid\_argument("MatrixAOrder != MatrixBOrder. Therefore matrix addition/subtraction for A and B is not defined");

}

double\*\* result = initializedouble2dpointerarray(m1rows, m1cols);

for (int i = 0; i < m1cols; i++)

{

for (int j = 0; j < m1rows; j++)

{

result[i][j] = m1[i][j] + m2[i][j];

}

}

return result;

}

double\*\* multiplybyconstant(double\*\* m, int height, int width, double constant) {

for (int i = 0; i < width; i++) {

for (int j = 0; j < height; j++) {

m[i][j] = m[i][j] \* constant;

}

}

return m;

}

double\*\* multiplymatrices(double\*\* m1, double\*\* m2, int m1height, int m1width, int m2height, int m2width) {

if (m1width != m2height) {

throw invalid\_argument("MatrixA width != MatrixB height. Therefore matrix multiplication for A and B is not defined");

}

double\*\* result = initializedouble2dpointerarray(m2width, m1height);

//check that m1width == m2height

for (int j = 0; j < m1height; j++) {

for (int i = 0; i < m2width; i++) {

double resultsum = 0;

for (int k = 0; k < m2height; k++) {

resultsum += m1[k][j] \* m2[i][k];

}

result[i][j] = resultsum;

}

}

return result;

}

double\*\* findminor(double\*\* m, int height, int width, int i, int j) { // i and j are indexes starting at 0, 0 for top left

// make copy so original doesn't get changed;

if (i >= width || j >= height)

{

throw invalid\_argument("The indexes of i and/or j are out of the range of the matrix");

}

if (width != height) {

throw invalid\_argument("non-square matrix does not have a minor");

}

if (height == 1) {

throw invalid\_argument("matrix with order 1x1 does not have a minor");

}

double\*\* result = copydouble2dpointerarray(m, width, height);

// delete row

for (int x = 0; x < width; x++) {

for (int y = j; y < height - 1; y++) {

result[x][y] = result[x][y+1];

}

}

// delete column

for (int y = 0; y < height; y++) {

for(int x = i; x < width - 1; x++) {

result[x][y] = result[x+1][y];

}

}

height -= 1;

width -= 1;

return result;

}

double finddeterminant(double\*\* m, int height, int width) {

if (width != height) {

throw invalid\_argument("non-square matrix does not have a determinant");

}

if (height == 2) {

return m[0][0] \* m[1][1] - m[0][1] \* m[1][0];

}

if (height == 1) {

return m[0][0];

}

double det = 0;

for (int i = 0; i < width; i++) {

double\*\* minor = findminor(m, height, width, i, 0);

det += pow(-1, i) \* m[i][0] \* finddeterminant(minor, height-1, width-1);

}

return det;

}

double\*\* transposematrix(double\*\* m, int height, int width) {

if (width != height) {

throw invalid\_argument("non-square matrix cannot be transposed");

}

double\*\* result = initializedouble2dpointerarray(width, height);

for (int i = 0; i < width; i++) {

for (int j = 0; j < i+1; j++) {

result[j][i] = m[i][j];

result[i][j] = m[j][i];

}

}

return result;

}

double\*\* matrixofcofactors(double\*\* m, int height, int width)

{

double\*\* result = initializedouble2dpointerarray(width, height);

for (int i = 0; i < width; i++) {

for (int j = 0; j < height; j++) {

double\*\* minor = findminor(m, height, width, i, j);

//cout << "minor " << i << ", " << j << ":" << endl;

//printmatrix(minor, height-1, width-1);

result[i][j] = pow(-1, i+j) \* finddeterminant(minor, height-1, width-1);

}

}

return result;

}

double\*\* inversematrix(double\*\* m, int height, int width) {

if (width != height) {

throw invalid\_argument("non-square matrix has no inverse");

}

double\*\* result = initializedouble2dpointerarray(width, height);

//find determinant

double det = finddeterminant(m, height, width);

double constant = (double)1 / det;

//cout << "determinant = " << det << endl;

if (det == 0) {

throw invalid\_argument("matrix with determinant = 0 does not have an inverse");

}

if (height == 1) {

result[0][0] = 1/m[0][0];

return result;

}

/\*if (height == 2) {

result[0][0] = m[1][1];

result[1][1] = m[0][0];

result[0][1] = m[0][1] \* -1;

result[1][0] = m[1][0] \* -1;

result = multiplybyconstant(result, 2, 2, constant);

return result;

}\*/

// find matrix of cofactors

/\*

for (int i = 0; i < width; i++) {

for (int j = 0; j < height; j++) {

double\*\* minor = findminor(m, height, width, i, j);

//cout << "minor " << i << ", " << j << ":" << endl;

//printmatrix(minor, height-1, width-1);

result[i][j] = pow(-1, i+j) \* finddeterminant(minor, height-1, width-1);

}

}

\*/

result = matrixofcofactors(m, height, width);

//transpose matrix

//cout << "pretranspose: " << endl;

//printmatrix(result, height, width);

result = transposematrix(result, height, width);

result = multiplybyconstant(result, height, width, constant);

//printmatrix(result, height, width);

return result;

}

//end of code

polynomial-solver.cpp:

//start of code

#include <cmath>

double GetFuncOutput(double input, int orderOfEqn, int\* coefficients) // coeff in form an \* xn an-1 \* xn-1 ... a0 x0

{

double output;

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

{

output += coefficients[i] \* pow(input, i);

}

return output;

}

bool CheckChangeOfSign(orderOfEqn, int\* coefficients, double[2] bounds)

{

lowerBoundOutput = GetFuncOutput(bound \* -1, orderOfEqn, coefficients);

upperBoundOutput = GetFuncOutput(bound, orderOfEqn, coefficients);

if (GetFuncOutput(bound[0], orderOfEqn, coefficients) \* GetFuncOutput(bound[1], orderOfEqn, coefficients) <= 0)

{

return true;

}

else

{

return false;

}

}

double[2] FindInitialBounds(int orderOfEqn, int\* coefficients)

{

double bound = 0.1;

double increment = bound;

bool found = false;

//double\* previousOutput

while(!found)

{

lowerBoundOutput = GetFuncOutput(bound \* -1, orderOfEqn, coefficients);

upperBoundOutput = GetFuncOutput(bound, orderOfEqn, coefficients);

if (lowerBoundOutput \* upperBoundOutput <= 0)

{

found = true;

}

else

{

bound += increment;

}

}

double[2] bounds = {-1 \* bound, bound};

return bounds;

}

double Bisection(int orderOfEqn, int\* coefficients, int dp) //dp = decimal places

{

//find initial bounds

double[2] bounds = FindInitialBounds(orderOfEqn, coefficients);

//half bounds each time

double diff;

bool found = false;

while (!found)

{

double midpoint = 0.5(bounds[0] + bounds[1]);

CheckChangeOfSign({})

}

//stop when change in result is lower than required significant figures

}

// end of code

simplex-method.cpp:

//start of code

using namespace std;

void printsolution(string\* varnames, double\* solutions, int nsolutions, stringstream &buffer) {

buffer << endl;

for (int i = 0; i < nsolutions; i++) {

buffer << varnames[i] << " = " << solutions[i] << endl;

}

}

double\* read\_simplex(double\*\* tableau, int width, int height) {

double\* solutions = new double[width - 1];

int test = 0;

for(int i = 0; i < width - 1; i++) {

bool foundnonzeroval = false;

bool nonbasic = false;

int basicvalindex;

for (int j = 0; j < height; j++) {

if (tableau[i][j] != 0) {

if (tableau[i][j] == 1 && foundnonzeroval == false) {

foundnonzeroval = true;

basicvalindex = j;

}

else {

solutions[i] = 0;

nonbasic = true;

break;

}

}

}

if (nonbasic == false)

{

solutions[i] = tableau[width - 1][basicvalindex];

}

}

return solutions;

}

void printtableau(double\*\* tableau, int width, int height) {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

cout << tableau[j][i] << " ";

}

cout << endl;

}

cout << endl << endl << endl;

}

int get\_pivot\_column(double\*\* tableau, int width, bool maximise) {

int val = 0;

int valindex = -1;

for (int i = 1; i < width - 1; i++) {

if (tableau[i][0] < val && maximise == true) {

val = tableau[i][0];

valindex = i;

}

if (tableau[i][0] > val && maximise == false) {

val = tableau[i][0];

valindex = i;

}

}

return valindex;

}

int get\_pivot\_row(double\*\* tableau, int pivotcolumnindex, int height, int rhscolumnindex, int startingrowindex) {

//second row down must be constraints

//only first row is objectiev function, therefore strip second stage when applicable

int minratioindex;

double minratio;

bool minratiodefined = false;

for (int i = startingrowindex; i < height; i++) {

if (tableau[pivotcolumnindex][i] == 0) {

continue;

}

double ratio = tableau[rhscolumnindex][i] / tableau[pivotcolumnindex][i];

if (ratio < 0) {

continue;

}

if (minratiodefined == false) {

minratio = ratio;

minratioindex = i;

minratiodefined = true;

}

if (ratio < minratio) {

minratio = ratio;

minratioindex = i;

}

}

return minratioindex; //minratioindex == minpivotrowindex is true

}

double\*\* divide\_pivot\_row(double\*\* tableau, int pivotcolumnindex, int pivotrowindex, int width) {

double multiplier = tableau[pivotcolumnindex][pivotrowindex];

for (int i = 0; i < width; i++) {

tableau[i][pivotrowindex] /= multiplier;

}

return tableau;

}

double\*\* perform\_iteration(double\*\* tableau, int pivotcolumnindex, int pivotrowindex, int width, int height) {

// i represents the i th row and j represents the j th column

tableau = divide\_pivot\_row(tableau, pivotcolumnindex, pivotrowindex, width);

for (int i = 0; i < height; i++) {

if (i == pivotrowindex) {

continue;

}

double multiplier = -1 \* tableau[pivotcolumnindex][i] / tableau[pivotcolumnindex][pivotrowindex];

for (int j = 0; j < width; j++) {

tableau[j][i] += multiplier \* tableau[j][pivotrowindex];

}

}

return tableau;

}

double\*\* maximise\_objective\_function(double\*\* tableau, int width, int height, int startingrowindex) {

cout << "maximising:" << endl;

bool maximised = false;

while(maximised == false) {

printtableau(tableau, width, height);

int pivotcolumnindex = get\_pivot\_column(tableau, width, true);

cout << "pivot column index: " << pivotcolumnindex << endl;

if (pivotcolumnindex == -1) {

maximised = true;

continue;

//maximised

}

int pivotrowindex = get\_pivot\_row(tableau, pivotcolumnindex, height, width - 1, startingrowindex);

cout << "pivot row index: " << pivotrowindex << endl;

tableau = perform\_iteration(tableau, pivotcolumnindex, pivotrowindex, width, height);

}

return tableau;

}

double\*\* minimise\_objective\_function(double\*\* tableau, int width, int height, int startingrowindex) {

cout << "minimising:" << endl;

bool minimised = false;

while(minimised == false) {

printtableau(tableau, width, height);

int pivotcolumnindex = get\_pivot\_column(tableau, width, false);

cout << "pivot column index: " << pivotcolumnindex << endl;

if (pivotcolumnindex == -1) {

minimised = true;

continue;

//maximised

}

int pivotrowindex = get\_pivot\_row(tableau, pivotcolumnindex, height, width - 1, startingrowindex);

cout << "pivot row index: " << pivotrowindex << endl;

tableau = perform\_iteration(tableau, pivotcolumnindex, pivotrowindex, width, height);

}

return tableau;

}

double\*\* convert\_to\_standard\_form(double\*\* tableau,int &width, int &height, int nartificialvar) {

cout << "width: " << width << endl;

for (int i = 0; i < width; i++) {

//delete &tableau[i][0];

//cout << "works" << endl;

for (int j = 0; j < height - 1; j++) {

//cout << j << endl;

//cout << tableau[i][j] << " " << tableau[i][j + 1] << endl;

tableau[i][j] = tableau[i][j + 1];

}

}

//cout << "works" << endl;

height -= 1;

for (int i = 0; i < height; i++) { // move RHS column to first art variable then change width index to effectively remove all art var

tableau[width - 1 - nartificialvar][i] = tableau[width - 1][i];

}

width -= nartificialvar;

for (int j = 0; j < height; j++) // remove art var "A" at beginning

{

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

{

tableau[i][j] = tableau[i+1][j]; //need to shift all var back by 1 along row (horizontally)

}

}

width -= 1; // move index

return tableau;

}

double\*\* apply\_simplex(double\*\* tableau, int nvar, int nconstraints, int nslackvar, int nartificialvar, int startingrowindex, string\* varnames, stringstream &buffer) {

int height;

int width = nvar + nslackvar + nartificialvar + 2;

if (nartificialvar != 0) {

width++;

height = nconstraints + 2;

tableau = minimise\_objective\_function(tableau, width, height, startingrowindex);

//delete first row and artificial variable columns

tableau = convert\_to\_standard\_form(tableau, width, height, nartificialvar);

tableau = maximise\_objective\_function(tableau, width, height, 1);

}

else {

height = nconstraints + 1;

tableau = maximise\_objective\_function(tableau, width, height, startingrowindex);

}

double\* solutions = read\_simplex(tableau, width, height);

cout << endl << "solutions[2]: " << solutions[2] << endl;

printsolution(varnames, solutions, width-1, buffer);

return tableau;

}

// end of code

simultaneous-equations-solver.cpp:

//start of code

//#include "matrix.cpp"

using namespace std;

string printtable(string\* varnames, double\*\* coefficients, int numofvar) {

stringstream buffer;

cout << endl << "simultaneous equations: " << endl << endl;

for (int j = 0; j < numofvar; j++) {

for (int i = 0; i < numofvar; i++) {

string sign = "+";

if (numofvar - i == 1) {

buffer << coefficients[i][j] << varnames[i] << " = " << coefficients[i+1][j] << endl;

}

else

{

buffer << coefficients[i][j] << varnames[i] << " " << sign << " ";

}

}

}

return buffer.str();

}

string printresults(string\* varnames, double\* results, int numofvar) {

stringstream buffer;

buffer << endl;

for (int i = 0; i < numofvar; i++) {

buffer << varnames[i] << " = " << results[i] << endl;

}

return buffer.str();

}

double\* SimulSolve(double\*\* coefficients, int numofvariables) { //equations in coefficient form line by line, where there is no coefficient the entry is 0

//the right most coefficient is the RHS (right hand side of the equation) value and all other values are the coefficients of the variables in order on the left hand side of the equation

//create square matrix from variables (nom of variables = num of equations) and matrix of order (numofvar x 1) from the RHS

double\*\* rhs = initializedouble2dpointerarray(1, numofvariables);

double\*\* inverse;

for (int j = 0; j < numofvariables; j++) {

rhs[0][j] = coefficients[numofvariables][j];

}

try

{

inverse = inversematrix(coefficients, numofvariables, numofvariables);

}

catch (invalid\_argument &exc)

{

throw invalid\_argument("this system of simultaneous equations does not have a unique solution");

}

//cout << "rhs val" << endl;

//printmatrix(rhs, numofvariables, 1);

//cout << "inverse val" << endl;

//printmatrix(inverse, numofvariables, numofvariables);

//cout << "inverse 1, 0 = " << inverse[1][0] << endl;

double\*\* results = multiplymatrices(inverse, rhs, numofvariables, numofvariables, numofvariables, 1); // yields an order (numofvar x 1) matrix

//cout << "result multiplication: " << endl;

//printmatrix(results, numofvariables, 1);

return results[0];

}

// end of code

GUI/addon.cpp:

//start of code

// addon.cc

#include <node.h>

#include "operation-handler.cpp"

using namespace std;

using v8::Exception;

using v8::FunctionCallbackInfo;

using v8::Isolate;

using v8::Local;

using v8::Number;

using v8::Object;

using v8::String;

using v8::Value;

void Main(const FunctionCallbackInfo<Value>& args) {

Isolate\* isolate = args.GetIsolate();

String::Utf8Value str(isolate, args[0]);

string cppStr(\*str);

string returnstring;

try

{

returnstring = DoOperation(cppStr);

}

catch(exception &e)

{

stringstream ss;

ss << "Error: " << e.what();

returnstring = ss.str();

//returnstring = "Error: " + e.what();

}

//cout << returnstring;

const char \* returnchararray = returnstring.c\_str();

args.GetReturnValue().Set(String::NewFromUtf8(

isolate, returnchararray).ToLocalChecked());

// Perform the operation

//double value = args[0].As<Number>()->Value() + args[1].As<Number>()->Value();

//Local<Number> num = Number::New(isolate, value);

//Local<String> v8String = String::NewFromUtf8(isolate, cppStr.c\_str(), String::kNormalString);

// Set the return value (using the passed in

// FunctionCallbackInfo<Value>&)

//args.GetReturnValue().Set(v8String);

}

void Init(Local<Object> exports) {

NODE\_SET\_METHOD(exports, "main", Main);

}

NODE\_MODULE(NODE\_GYP\_MODULE\_NAME, Init)

// end of code

GUI/binding.gyp:

//start of code

{

"targets": [

{

"target\_name": "addon",

"sources": [ "addon.cpp" ]

}

]

}

// end of code

GUI/dataparser.cpp:

//start of code

#include <sstream>

#include <iostream>

#include <vector>

#include "../arrays.cpp"

using namespace std;

string\* SliceAlphabet(string alphabet[24], int numOfElements)

{

string\* outputstring = new string[numOfElements];

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

{

outputstring[i] = alphabet[i];

}

return outputstring;

}

bool IsInteger(const std::string &s)

{

if(s.empty() || ((!isdigit(s[0])) && (s[0] != '-') && (s[0] != '+'))) return false;

char \* p;

strtol(s.c\_str(), &p, 10);

return (\*p == 0);

}

bool IsDouble(string str)

{

const char\* c = str.c\_str();

char\* endptr = 0;

strtod(c, &endptr);

if(\*endptr != '\0' || endptr == c)

return false;

return true;

}

string ConvertMatToString(double\*\* mat, int\* order)

{

string outputstring = "";

outputstring += to\_string(order[0]) + " " + to\_string(order[1]) + " ";

for (int i = 0; i < order[1]; i++)

{

for (int j = 0; j < order[0]; j++)

{

outputstring += to\_string(mat[i][j]) + " ";

}

}

return outputstring;

}

string\* split(const string &s, char seperator)

{

string\* output = new string[s.length()];

string::size\_type prev\_pos = 0, pos = 0;

int x = 0;

while((pos = s.find(seperator, pos)) != string::npos)

{

string substring( s.substr(prev\_pos, pos-prev\_pos) );

output[x] = substring;

prev\_pos = ++pos;

x++;

}

output[x] = s.substr(prev\_pos, pos-prev\_pos); // Last word

return output;

}

void MakeMatrix(double\*\* &mat, int &matrows, int &matcols, string\* array, int &index)

{

matrows = stoi(array[index]);

index++;

matcols = stoi(array[index]);

index++;

mat = initializedouble2dpointerarray(matcols, matrows);

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

{

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

{

try {

mat[i][j] = stod(array[index]);

}

catch (invalid\_argument &e) {

throw invalid\_argument("Matrix contains invalid characters");

}

index++;

}

}

}

void parseSimulString(string &optype, double\*\* &coeff, int &width, int &height, string\* inputarray, int &index)

{

optype = inputarray[index];

index++;

try

{

MakeMatrix(coeff, height, width, inputarray, index);

}

catch (invalid\_argument &e) {

throw invalid\_argument("simultaneous equations contain invalid characters");

}

}

void ParseStringTo2dArray(string &optype, int &mat1rows, int &mat1cols, int &mat2rows, int &mat2cols, double\*\* &mat1, double\*\* &mat2, string\* inputarray, int index, string &constant, string &i, string &j)

{

//inputarray format: mat1rows, mat1cols, elements of mat1, mat2rows, mat2cols, elements of mat 2

optype = inputarray[index];

index++;

MakeMatrix(mat1, mat1rows, mat1cols, inputarray, index);

MakeMatrix(mat2, mat2rows, mat2cols, inputarray, index);

constant = inputarray[index];

index++;

i = inputarray[index];

index++;

j = inputarray[index];

index++;

}

void parseTableau(double\*\* &tableau, int &numOfVar, int &numOfConstraints, int &numOfSlackVar, int &numOfArtificialVar, int &startingRowIndex, string &opType, string\* &varnames, string\* array, int index)

{

string variablenames[8] = {"P", "X", "Y", "Z", "W", "V", "U", "T"};

opType = array[index];

index++;

numOfVar = stoi(array[index]);

index++;

numOfSlackVar = stoi(array[index]);

index++;

numOfArtificialVar = stoi(array[index]);

index++;

if (numOfArtificialVar == 0)

{

startingRowIndex = 1;

}

else

{

startingRowIndex = 2;

}

int matrows, matcols;

MakeMatrix(tableau, matrows, matcols, array, index);

varnames = new string[matcols - startingRowIndex];

int temp = 0;

for (int i = 0; i < numOfVar+1; i++)

{

varnames[i] = variablenames[i];

temp++;

}

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

{

varnames[temp] = "s" + to\_string(i+1);

temp++;

}

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

{

varnames[temp] = "a" + to\_string(i+1);

temp++;

}

numOfConstraints = matrows - startingRowIndex;

}

// end of code

GUI/operation-handler.cpp:

//start of code

#include "dataparser.cpp"

#include "../matrix.cpp"

#include "../simultaneous-equations-solver.cpp"

#include "../simplex-method.cpp"

using namespace std;

string DoMatrixOperation(string\* inputarray, int index)

{

bool resultismatrix = true;

string outputstring;

string optype;

int matArows, matAcols, matBrows, matBcols;

double\*\* matA;

double\*\* matB;

int\* resultorder = new int[2];

double\*\* result;

string constant;

string i, j; //index of minor

ParseStringTo2dArray(optype, matArows, matAcols, matBrows, matBcols, matA, matB, inputarray, index, constant, i, j);

if (optype == "Add")

{

result = addmatrices(matA, matB, matArows, matAcols, matBrows, matBcols);

resultorder[0] = matArows;

resultorder[1] = matAcols;

}

if (optype == "Sub")

{

matB = multiplybyconstant(matB, matBrows, matBcols, -1);

result = addmatrices(matA, matB, matArows, matAcols, matBrows, matBcols);

resultorder[0] = matArows;

resultorder[1] = matAcols;

}

if (optype == "Mult")

{

result = multiplymatrices(matA, matB, matArows, matAcols, matBrows, matBcols);

resultorder[0] = matArows;

resultorder[1] = matBcols;

//printmatrix(result, 2, 2);

}

if (optype == "DetA")

{

outputstring = "Det: " + to\_string(finddeterminant(matA, matArows, matAcols));

resultismatrix = false;

}

if (optype == "DetB")

{

outputstring = "Det: " + to\_string(finddeterminant(matB, matBrows, matBcols));

resultismatrix = false;

}

if (optype == "InvA")

{

result = inversematrix(matA, matArows, matAcols);

resultorder[0] = matArows;

resultorder[1] = matAcols;

}

if (optype == "InvB")

{

result = inversematrix(matB, matBrows, matBcols);

resultorder[0] = matBrows;

resultorder[1] = matBcols;

}

if (optype == "SquareA")

{

result = multiplymatrices(matA, matA, matArows, matAcols, matArows, matAcols);

resultorder[0] = matArows;

resultorder[1] = matAcols;

}

if (optype == "SquareB")

{

result = multiplymatrices(matB, matB, matBrows, matBcols, matBrows, matBcols);

resultorder[0] = matBrows;

resultorder[1] = matBcols;

}

if (optype == "TransA")

{

result = transposematrix(matA, matArows, matAcols);

resultorder[0] = matArows;

resultorder[1] = matAcols;

}

if (optype == "TransB")

{

result = transposematrix(matB, matBrows, matBcols);

resultorder[0] = matBrows;

resultorder[1] = matBcols;

}

if (optype == "MinA")

{

if (!IsInteger(i))

{

throw invalid\_argument("minor index i must be an integer");

}

if (!IsInteger(j))

{

throw invalid\_argument("minor index j must be an integer");

}

result = findminor(matA, matArows, matAcols, stoi(i), stoi(j));

resultorder[0] = matArows - 1;

resultorder[1] = matAcols - 1;

}

if (optype == "MinB")

{

if (!IsInteger(i))

{

throw invalid\_argument("minor index i must be an integer");

}

if (!IsInteger(j))

{

throw invalid\_argument("minor index j must be an integer");

}

result = findminor(matB, matBrows, matBcols, stoi(i), stoi(j));

resultorder[0] = matBrows - 1;

resultorder[1] = matBcols - 1;

}

if (optype == "MultconstA")

{

if (!IsDouble(constant))

{

throw invalid\_argument("constant must be a real number");

}

result = multiplybyconstant(matA, matArows, matAcols, stod(constant));

resultorder[0] = matArows;

resultorder[1] = matAcols;

}

if (optype == "MultconstB")

{

if (!IsDouble(constant))

{

throw invalid\_argument("constant must be a real number");

}

result = multiplybyconstant(matB, matBrows, matBcols, stod(constant));

resultorder[0] = matBrows;

resultorder[1] = matBcols;

}

if (resultismatrix)

{

outputstring = ConvertMatToString(result, resultorder);

}

//cout << outputstring;

return outputstring;

}

string DoSimul(string\* array, int index)

{

string alphabet[] = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"};

double\* result;

double\*\* coefficients;

int numOfVariables;

string optype;

int width, height;

parseSimulString(optype, coefficients, width, height, array, index);

//cout << "width: " << width << endl << "height: " << height << endl;

if (width != height + 1)

{

throw invalid\_argument("number of equations must be equal to number of variables");

}

numOfVariables = height;

if (numOfVariables > 24) {

throw invalid\_argument("too many variables entered");

}

result = SimulSolve(coefficients, numOfVariables);

string\* variables = SliceAlphabet(alphabet, numOfVariables);

return printresults(variables, result, numOfVariables);

}

string DoSimplex(string\* array, int index)

{

stringstream buffer;

string\* varnames;

double\*\* result;

double\*\* tableau;

int numOfVar, numOfConstraints, numOfSlackVar, numOfArtificialVar, startingRowIndex;

string opType;

parseTableau(tableau, numOfVar, numOfConstraints, numOfSlackVar, numOfArtificialVar, startingRowIndex, opType, varnames, array, index);

cout << "numOfVar: " << numOfVar << endl << "nconstraints: " << numOfConstraints << endl << "numOfSlackVar: " << numOfSlackVar << endl << "numOfArtificialVar: " << numOfArtificialVar << endl << "starting row index: " << startingRowIndex << endl;

int width = numOfVar + numOfSlackVar + numOfArtificialVar + 1 + startingRowIndex;

int height = numOfConstraints + startingRowIndex;

printtableau(tableau, width, height);

result = apply\_simplex(tableau, numOfVar, numOfConstraints, numOfSlackVar, numOfArtificialVar, startingRowIndex, varnames, buffer);

return buffer.str();

}

string DoOperation(string inputstring)

{

int index = 0;

string\* array = split(inputstring, ' ');

string opsource = array[index];

index++;

if (opsource == "Matrix")

{

return DoMatrixOperation(array, index);

}

if (opsource == "Simul")

{

return DoSimul(array, index);

}

else if (opsource == "Simplex")

{

try{

return DoSimplex(array, index);

}

catch(exception &e)

{

throw invalid\_argument("invalid input");

}

}

}

// end of code

GUI/server.js:

//start of code

const express = require('express'),

path = require('path')

const addon = require('./build/Release/addon');

const bodyParser = require('body-parser');

const app = express();

const port = 3000

/\*function ConvertStringToMat(chararray)

{

var index = 0;

var matrows = chararray[index];

index++;

var matcols = chararray[index];

index++;

var mat = new Array(matcols);

for (var i = 0; i < matcols; i++)

{

mat[i] = new Array(matrows);

for (var j = 0; j < matrows; j++)

{

mat[i][j] = parseFloat(chararray[index]);

index++;

}

}

return mat;

} \*/

function MatrixString(dict, matLetter)

{

//console.log("test log: " + dict["Matrix" + matLetter + "Size"]);

var matSize = dict["Matrix" + matLetter + "Size"].split("x");

var str = "";

str += matSize[0] + " " + matSize[1] + " "

for (var i = 0; i < matSize[1]; i++)

{

for (var j = 0; j < matSize[0]; j++)

{

str += dict[matLetter + "\_" + i + "\_" + j] + " ";

}

}

return str

}

app.use(bodyParser.urlencoded({ extended: true }));

app.set('view engine', 'ejs');

app.set('views', path.resolve( \_\_dirname, 'views'));

app.get('/simul', function(req, res) {

res.render('simul', {matrixAString: "", message: "", result: ""});

})

app.get('/', function(req, res) {

res.render('menu', {});

})

app.post('/simul', function(req, res){

console.log(req.body);

var matrixAString = MatrixString(req.body, "A");

var inputstring = "Simul " + req.body["type"] + " " + matrixAString;

console.log("inputstring: " + inputstring);

var outputstring = addon.main(inputstring);

//console.log("outputstring: " + outputstring);

//var outputmatrix = ConvertStringToMat(outputstring.split(" "));

if (outputstring.split(" ")[0] == "Error:")

{

console.log(outputstring);

res.render('simul', {matrixAString: matrixAString, message: outputstring, result: ""});

}

else {

res.render('simul', {matrixAString: matrixAString, message:"", result: outputstring});

}

})

app.get('/simplex', function(req, res) {

res.render('simplex', {matrixAString: "", message: "", result: ""});

})

app.post('/simplex', function(req, res){

console.log(req.body);

var matrixAString = MatrixString(req.body, "A");

var inputstring = "Simplex " + req.body["type"] + " " + req.body["numofvar"] + " " + req.body["numofslackvar"] + " " + req.body["numofartificialvar"] + " " + matrixAString;

console.log("inputstring: " + inputstring);

var outputstring = addon.main(inputstring);

//console.log("outputstring: " + outputstring);

//var outputmatrix = ConvertStringToMat(outputstring.split(" "));

if (outputstring.split(" ")[0] == "Error:")

{

console.log(outputstring);

res.render('simplex', {matrixAString: matrixAString, message: outputstring, result: ""});

}

else {

res.render('simplex', {matrixAString: matrixAString, message:"", result: outputstring});

}

})

app.get('/matrix', function(req, res) {

res.render('matrix', {matrixAString: "", matrixBString: "", matrixCString: "", message: ""});

})

app.post('/matrix', function(req, res){

console.log(req.body);

var matrixAString = MatrixString(req.body, "A");

var matrixBString = MatrixString(req.body, "B");

var inputstring = "Matrix " + req.body["type"] + " " + matrixAString + matrixBString + req.body["cVal"] + " " + req.body["i"] + " " + req.body["j"];

console.log("inputstring: " + inputstring);

var outputstring = addon.main(inputstring);

//console.log("outputstring: " + outputstring);

//var outputmatrix = ConvertStringToMat(outputstring.split(" "));

if (outputstring.split(" ")[0] == "Error:" || outputstring.split(" ")[0] == "Det:")

{

console.log(outputstring);

res.render('matrix', {matrixAString: matrixAString, matrixBString: matrixBString, matrixCString: "", message: outputstring});

}

else {

res.render('matrix', {matrixAString: matrixAString, matrixBString: matrixBString, matrixCString: outputstring, message:""});

}

})

app.listen(port, () => {

console.log(`Example app listening at http://localhost:${port}`)

})

process.on('uncaughtException', function (err) {

console.error((new Date).toUTCString() + ' uncaughtException:', err.message)

console.error(err.stack)

process.exit(1)

})

// end of code

GUI/views/matrix.ejs:

//start of code

<body><div id="content" role="main"><!-- #BeginEditable "Body" -->

<h1 align="center">Matrix Calculator</h1>

<div style="position:relative; border-radius: 10px; display:block; background-color: hsla(240,100%,96%,1); ">

<form name="MatrixInputs" method="POST" action="/matrix">

<div style="display: inline-block; min-height:200px; min-width:300px; vertical-align: top; position: relative; background-color: hsla(240,100%,94%,1); border-radius: 10px; margin:2px; ">

<div id="MatrixATitle" style="margin-left:20px; font: bold 22px Arial; text-align: left; color: darkblue; ">1x0</div>

<input type="hidden" id="MatrixASize" name="MatrixASize" value="1x0">

<button id="upBtn" type="button" style="position:absolute; left:172px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="DeleteRow('MatrixA')">▲</button>

<button id="dnBtn" type="button" style="position:absolute; left:202px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="CreateRow('MatrixA')">▼</button>

<button id="ltBtn" type="button" style="position:absolute; left:232px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="DeleteColumn('MatrixA')">◀</button>

<button id="rtBtn" type="button" style="position:absolute; left:262px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="CreateColumn('MatrixA')">▶</button>

<table id="MatrixA" style="margin: 25px 5px 5px 30px; text-align: left;">

<tbody></tbody>

</table></div>

<div style="display: inline-block; min-height:200px; min-width:300px; vertical-align: top; position: relative; background-color: hsla(240,100%,94%,1); border-radius: 10px; margin:2px; ">

<div type="SUBMIT" id="MatrixBTitle" name="MatrixSize" style="margin-left:20px; font: bold 22px Arial; text-align: left; color: darkblue;">1x0</div>

<input type="hidden" id="MatrixBSize" name="MatrixBSize" value="1x0">

<button id="upBtn" type="button" style="position:absolute; left:172px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="DeleteRow('MatrixB')">▲</button>

<button id="dnBtn" type="button" style="position:absolute; left:202px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="CreateRow('MatrixB')">▼</button>

<button id="ltBtn" type="button" style="position:absolute; left:232px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="DeleteColumn('MatrixB')">◀</button>

<button id="rtBtn" type="button" style="position:absolute; left:262px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="CreateColumn('MatrixB')">▶</button>

<table style="margin: 25px 5px 5px 30px; text-align: left;" id="MatrixB">

<tbody></tbody>

</table></div>

<br>

<div style="padding:2px; margin-bottom:5px; text-align: center; background-color: lightskyblue;border-radius: 10px; ">

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="Add">A+B</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="Sub">A−B</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="Mult">AB</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="DetA">det(A)</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="DetB">det(B)</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="InvA">inv(A)</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="InvB">inv(B)</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="button" value="Swap" onclick="SwapAandB(MatrixA, MatrixB)">A↔B</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="SquareA">A<sup>2</sup></button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="SquareB">B<sup>2</sup></button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="TransA">A<sup>T</sup></button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="TransB">B<sup>T</sup></button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="MinA">minor(A)</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="MinB">minor(B)</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="MultconstA">cA</button>

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="MultconstB">cB</button>

<input type="text" name="cVal" style="color: #0000ff; background-color: #eeffee; text-align:center; font: 20px Arial; width:35px; border-radius: 10px; " value="constant">

<input type="text" name="i" style="color: #0000ff; background-color: #eeffee; text-align:center; font: 20px Arial; width:35px; border-radius: 10px; " value="minorindex">

<input type="text" name="j" style="color: #0000ff; background-color: #eeffee; text-align:center; font: 20px Arial; width:35px; border-radius: 10px; " value="minorindex">

<div id="msg" style="height:22px; margin:2px; font: 20px Arial; text-align: center; color: red; "><%=message%></div></div>

</form>

<div style="border: none;">

<div style="float:left; text-align: center;">

<div id="outLbl"></div>

<div id="dirty"></div>

<br>

<button id="btn16" style="color: #000aae; font: 18px Arial;" class="togglebtn" onclick="CopyPasteMatrix('MatrixC', 'MatrixA')">C to A</button>

<br><button id="btn17" style="color: #000aae; font: 18px Arial;" class="togglebtn" onclick="CopyPasteMatrix('MatrixC', 'MatrixB')">C to B</button>

<br><button id="btn18" style="color: #000aae; font: 18px Arial;" class="togglebtn" onclick="CopyPasteMatrix('MatrixA', 'MatrixC')">A to C</button>

<br><button id="btn19" style="color: #000aae; font: 18px Arial;" class="togglebtn" onclick="CopyPasteMatrix('MatrixB', 'MatrixC')">B to C</button>

<br></div>

<div style="overflow: hidden; display: block;">

<div style="display: inline-block; min-height:200px; min-width:300px; vertical-align: top; position: relative; background-color: hsla(240,100%,94%,1); border-radius: 10px; margin:2px; ">

<div id="MatrixCTitle" style="margin-left:20px; font: bold 22px Arial; text-align: left; color: darkblue; ">1x0</div>

<input type="hidden" id="MatrixCSize" name="MatrixCSize" value="1x0">

<button id="upBtn" type="button" style="position:absolute; left:172px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="DeleteRow('MatrixC')">▲</button>

<button id="dnBtn" type="button" style="position:absolute; left:202px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="CreateRow('MatrixC')">▼</button>

<button id="ltBtn" type="button" style="position:absolute; left:232px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="DeleteColumn('MatrixC')">◀</button>

<button id="rtBtn" type="button" style="position:absolute; left:262px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="CreateColumn('MatrixC')">▶</button>

<table id="MatrixC" style="margin: 25px 5px 5px 30px; text-align: left;">

<tbody></tbody>

</table></div>

</div>

</div>

</div>

<div style="font: 11px arial; color: #6600cc;"></div>

</div>

<script>

function ConvertStringToMat(chararray)

{

var index = 0;

var matrows = chararray[index];

index++;

var matcols = chararray[index];

index++;

var mat = new Array(matcols);

for (var i = 0; i < matcols; i++)

{

mat[i] = new Array(matrows);

for (var j = 0; j < matrows; j++)

{

mat[i][j] = parseFloat(chararray[index]);

index++;

}

}

return mat;

}

function UpdateSizeValues(tablename)

{

var table = document.getElementById(tablename);

var title = document.getElementById(tablename + "Title");

var titleinput = document.getElementById(tablename + "Size");

title.innerHTML = table.id.slice(-1) + ": " + String(table.rows.length) + "x" + String(table.rows[0].cells.length)

titleinput.value = String(table.rows.length) + "x" + String(table.rows[0].cells.length)

}

function CreateCell(columnindex, rowindex, table, textvalue) {

var cell = table.rows[rowindex].insertCell(columnindex);

cell.innerHTML = '';

cell.style = "width: fit-content;";

var input = document.createElement('input');

//var text = "NEW CELL" + String(columnindex);

input.type = "text";

input.value = textvalue;

input.name = table.id.slice(-1) + "\_" + String(columnindex) + "\_" + String(rowindex); //first num is left to right, second num is from top to bottom.

input.style = "color: #0000ff; background-color: #eeffee; text-align:center; font: 15px Arial; width:40px; border-radius: 10px;";

cell.appendChild(input);

}

function CreateRow(tablename) {

var table = document.getElementById(tablename);

var row = table.insertRow(table.rows.length);

for (var i = 0; i < table.rows[0].cells.length; i++) { //to add cells to the previous rows for square matrix

/\*if (i < table.rows.length - 1) {

CreateCell(i, table.rows.length - 1, table, 0);

}\*/

CreateCell(i, table.rows.length - 1, table, 0);

}

UpdateSizeValues(tablename);

}

function CreateColumn(tablename)

{

var table = document.getElementById(tablename);

var len = table.rows[0].cells.length;

for (var j = 0; j < table.rows.length; j++)

{

for (var i = table.rows[j].cells.length; i < len+1; i++)

CreateCell(i, j, table, 0);

}

UpdateSizeValues(tablename);

}

function DeleteRow(tablename) {

var table = document.getElementById(tablename);

if (table.rows.length > 1)

{

table.deleteRow(table.rows.length - 1);

UpdateSizeValues(tablename);

}

}

function DeleteColumn(tablename)

{

var table = document.getElementById(tablename);

if (table.rows[0].cells.length > 1)

{

for (var i = 0; i < table.rows.length; i++) {

table.rows[i].deleteCell(table.rows[i].cells.length - 1);

}

UpdateSizeValues(tablename);

}

}

function ConvertTo2dArray(tablename)

{

var table = document.getElementById(tablename);

var matrix = new Array(table.rows[0].cells.length);

for (var i = 0; i < table.rows[0].cells.length; i++)

{

matrix[i] = new Array(table.rows.length);

for (var j = 0; j < table.rows.length; j++)

{

matrix[i][j] = table.rows[j].cells[i].firstChild.value;

}

}

return matrix;

}

function GenerateMatrix(tablename, matrix)

{

var table = document.getElementById(tablename);

// delete table

var len = table.rows.length

for (var i = 0; i < len; i++)

{

table.deleteRow(0);

}

for (var j = 0; j < matrix[0].length; j++)

{

table.insertRow(j);

for (var i = 0; i < matrix.length; i++)

{

CreateCell(i, j, table, matrix[i][j]);

}

}

UpdateSizeValues(tablename);

}

function CopyPasteMatrix(tablename1, tablename2)

{

var matrix = ConvertTo2dArray(tablename1)

GenerateMatrix(tablename2, matrix);

}

function SwapAandB()

{

var matrixA = ConvertTo2dArray("MatrixA");

var matrixB = ConvertTo2dArray("MatrixB");

GenerateMatrix("MatrixA", matrixB);

GenerateMatrix("MatrixB", matrixA);

}

</script>

<script type="text/javascript">

function startup() {

var predefined = [false, false, false];

var matrixLetters = ["A", "B", "C"];

var matrices = ["<%=matrixAString%>", "<%=matrixBString%>", "<%=matrixCString%>"];

for (var i = 0; i < matrices.length; i++)

{

if (matrices[i] != "") {

predefined[i] = true;

//console.log("string for matrix" + matrixLetters[i] + ": " + matrices[i]);

var matrix = ConvertStringToMat(matrices[i].split(" "));

GenerateMatrix("Matrix" + matrixLetters[i], matrix);

}

}

for (var i = 0; i < matrices.length; i++)

{

if (!predefined[i])

{

var nrows = 2

for (var j = 0; j < nrows; j++)

{

//console.log("adding rows and cols for Matrix" + matrixLetters[i]);

CreateRow("Matrix" + matrixLetters[i]);

CreateColumn("Matrix" + matrixLetters[i]);

}

}

}

}

window.onload = startup;

</script>

</div>

</body>

// end of code

GUI/views/simplex.ejs:

//start of code

<body><div id="content" role="main"><!-- #BeginEditable "Body" -->

<h1 align="center">Simplex Solver</h1>

<div style="position:relative; border-radius: 10px; display:block; background-color: hsla(240,100%,96%,1); ">

<form name="MatrixInputs" method="POST" action="/simplex">

<div style="display: inline-block; min-height:200px; min-width:300px; vertical-align: top; position: relative; background-color: hsla(240,100%,94%,1); border-radius: 10px; margin:2px; ">

<div id="MatrixATitle" style="margin-left:20px; font: bold 22px Arial; text-align: left; color: darkblue; ">A: 1x1</div>

<input type="hidden" id="MatrixASize" name="MatrixASize" value="1x1">

<table id="MatrixA" style="margin: 25px 5px 5px 30px; text-align: left;">

<tbody>

</tbody>

</table>

</div>

<table id="MatrixA" style="margin: 25px 5px 5px 30px; text-align: left;">

<tbody>

<tr>

<th style="color: rgb(0, 0, 255); background-color: rgb(238, 255, 238); text-align: center; font: 15px Arial; width: 80px; border-radius: 20px;">num of var </th>

<th style="color: rgb(0, 0, 255); background-color: rgb(238, 255, 238); text-align: center; font: 15px Arial; width: 80px; border-radius: 20px;">num of slackvar</th>

<th style="color: rgb(0, 0, 255); background-color: rgb(238, 255, 238); text-align: center; font: 15px Arial; width: 80px; border-radius: 20px;">num of constraints</th>

<th style="color: rgb(0, 0, 255); background-color: rgb(238, 255, 238); text-align: center; font: 15px Arial; width: 80px; border-radius: 20px;">num of artificial var</th>

</tr>

<tr>

<td>

<input type="text" id="numofvar" name="numofvar" placeholder="3" style="color: rgb(0, 0, 255); background-color: rgb(238, 255, 238); text-align: center; font: 15px Arial; width: 40px; border-radius: 10px;">

</td>

<td>

<input type="text" id="numofslackvar" name="numofslackvar" placeholder="3" style="color: rgb(0, 0, 255); background-color: rgb(238, 255, 238); text-align: center; font: 15px Arial; width: 40px; border-radius: 10px;">

</td>

<td>

<input type="text" id="numofconstraints" name="numofconstraints" placeholder="3" style="color: rgb(0, 0, 255); background-color: rgb(238, 255, 238); text-align: center; font: 15px Arial; width: 40px; border-radius: 10px;">

</td>

<td>

<input type="text" id="numofartificialvar" name="numofartificialvar" placeholder="0" style="color: rgb(0, 0, 255); background-color: rgb(238, 255, 238); text-align: center; font: 15px Arial; width: 40px; border-radius: 10px;">

</td>

</tr>

</tbody>

</table>

<button id="gentable" type="button" style="font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="GenTable('MatrixA')">generatetable</button>

<br>

<div style="padding:2px; margin-bottom:5px; text-align: center; background-color: lightskyblue;border-radius: 10px; ">

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="Solve">Solve</button>

<div id="msg" style="height:22px; margin:2px; font: 20px Arial; text-align: center; color: red; "><%=message%></div></div>

</form></div>

<div style="font: 11px arial; color: #6600cc;">

<textarea id="w3review" name="w3review" rows="4" cols="50">

<%=result%>

</textarea></div>

</div>

<script>

function ConvertStringToMat(chararray)

{

var index = 0;

var matrows = chararray[index];

index++;

var matcols = chararray[index];

index++;

var mat = new Array(matcols);

for (var i = 0; i < matcols; i++)

{

mat[i] = new Array(matrows);

for (var j = 0; j < matrows; j++)

{

mat[i][j] = parseFloat(chararray[index]);

index++;

}

}

return mat;

}

function UpdateSizeValues(tablename)

{

var table = document.getElementById(tablename);

var title = document.getElementById(tablename + "Title");

var titleinput = document.getElementById(tablename + "Size");

title.innerHTML = table.id.slice(-1) + ": " + String(table.rows.length - 1) + "x" + String(table.rows[0].cells.length)

titleinput.value = String(table.rows.length - 1) + "x" + String(table.rows[0].cells.length)

}

function CreateCell(columnindex, rowindex, table, textvalue) {

var cell = table.rows[rowindex].insertCell(columnindex);

cell.innerHTML = '';

var input = document.createElement('input');

//var text = "NEW CELL" + String(columnindex);

input.type = "text";

input.value = textvalue;

input.name = table.id.slice(-1) + "\_" + String(columnindex) + "\_" + String(rowindex - 1); //first num is left to right, second num is from top to bottom.

input.style = "color: #0000ff; background-color: #eeffee; text-align:center; font: 15px Arial; width:40px; border-radius: 10px; ";

cell.appendChild(input);

}

function CreateRow(tablename) {

var table = document.getElementById(tablename);

var row = table.insertRow(table.rows.length);

for (var i = 0; i < table.rows[0].cells.length; i++) { //to add cells to the previous rows for square matrix

/\*if (i < table.rows.length - 1) {

CreateCell(i, table.rows.length - 1, table, 0);

}\*/

CreateCell(i, table.rows.length - 1, table, 0);

}

UpdateSizeValues(tablename);

}

function CreateColumn(tablename)

{

var table = document.getElementById(tablename);

var len = table.rows[0].cells.length;

for (var j = 1; j < table.rows.length; j++)

{

CreateCell(len, j, table, 0);

}

UpdateHeaderRow(tablename);

UpdateSizeValues(tablename);

}

function ConvertTo2dArray(tablename)

{

var table = document.getElementById(tablename);

var matrix = new Array(table.rows[0].cells.length);

for (var i = 0; i < table.rows[0].cells.length; i++)

{

matrix[i] = new Array(table.rows.length);

for (var j = 0; j < table.rows.length; j++)

{

matrix[i][j] = table.rows[j].cells[i].firstChild.value;

}

}

return matrix;

}

function GenerateMatrix(tablename, matrix)

{

var table = document.getElementById(tablename);

// delete table

var len = table.rows.length

for (var i = 0; i < len; i++)

{

table.deleteRow(0);

}

table.insertRow(0);

for (var j = 0; j < matrix[0].length; j++)

{

table.insertRow(j+1);

for (var i = 0; i < matrix.length; i++)

{

CreateCell(i, j+1, table, matrix[i][j]);

}

}

UpdateHeaderRow(tablename);

UpdateSizeValues(tablename);

}

function AddHeaderElement(row, value)

{

var headerCell = document.createElement("TH");

headerCell.innerHTML = value;

row.appendChild(headerCell);

}

function GenTable(tablename)

{

var table = document.getElementById(tablename);

var numofvar = parseInt(document.getElementById("numofvar").value);

var numofconstraints = parseInt(document.getElementById("numofconstraints").value);

var numofslackvar = parseInt(document.getElementById("numofslackvar").value);

var numofartificialvar = parseInt(document.getElementById("numofartificialvar").value);

var varnames = ["P", "X", "Y", "Z", "W", "V", "U", "T"];

var len = table.rows.length;

for (var i = 0; i < len; i++)

{

table.deleteRow(0);

}

var hasartificialvar = false;

var row = table.insertRow(0);

if (numofartificialvar != 0)

{

AddHeaderElement(row, "A");

hasartificialvar = true;

}

for (var i = 0; i < numofvar+1; i++)

{

AddHeaderElement(row, varnames[i]);

}

for (var i = 0; i < numofslackvar; i++)

{

AddHeaderElement(row, "s" + String(i+1));

}

for (var i = 0; i < numofartificialvar; i++)

{

AddHeaderElement(row, "a" + String(i+1));

}

AddHeaderElement(row, "RHS");

var numofcols = table.rows[0].cells.length;

if (hasartificialvar)

{

CreateRow(tablename);

}

for (var i = 0; i < numofconstraints+1; i++)

{

CreateRow(tablename);

}

}

</script>

<script type="text/javascript">

function startup() {

var predefined = [false];

var matrixLetters = ["A"];

var matrices = ["<%=matrixAString%>"];

for (var i = 0; i < matrices.length; i++)

{

if (matrices[i] != "") {

predefined[i] = true;

//console.log("string for matrix" + matrixLetters[i] + ": " + matrices[i]);

var matrix = ConvertStringToMat(matrices[i].split(" "));

GenerateMatrix("Matrix" + matrixLetters[i], matrix);

}

}

for (var i = 0; i < matrices.length; i++)

{

if (!predefined[i])

{

var nrows = 2

CreateRow("Matrix" + matrixLetters[i]);

for (var j = 0; j < nrows; j++)

{

//console.log("adding rows and cols for Matrix" + matrixLetters[i]);

CreateRow("Matrix" + matrixLetters[i]);

CreateColumn("Matrix" + matrixLetters[i]);

}

}

}

}

window.onload = startup;

</script>

</body>

// end of code

GUI/views/simul.ejs:

//start of code

<body><div id="content" role="main"><!-- #BeginEditable "Body" -->

<h1 align="center">Simultaneous equations Solver</h1>

<div style="position:relative; border-radius: 10px; display:block; background-color: hsla(240,100%,96%,1); ">

<form name="MatrixInputs" method="POST" action="/simul">

<div style="display: inline-block; min-height:200px; min-width:300px; vertical-align: top; position: relative; background-color: hsla(240,100%,94%,1); border-radius: 10px; margin:2px; ">

<div id="MatrixATitle" style="margin-left:20px; font: bold 22px Arial; text-align: left; color: darkblue; ">A: 1x1</div>

<input type="hidden" id="MatrixASize" name="MatrixASize" value="1x1">

<button id="upBtn" type="button" style="position:absolute; left:172px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="DeleteRow('MatrixA')">▲</button>

<button id="dnBtn" type="button" style="position:absolute; left:202px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="CreateRow('MatrixA')">▼</button>

<button id="ltBtn" type="button" style="position:absolute; left:232px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="DeleteColumn('MatrixA')">◀</button>

<button id="rtBtn" type="button" style="position:absolute; left:262px; top:20px; height:23px; font: 14px/16px Arial; color: #000aae; " class="togglebtn" onclick="CreateColumn('MatrixA')">▶</button>

<table id="MatrixA" style="margin: 25px 5px 5px 30px; text-align: left;">

<tbody>

</tbody>

</table></div>

<br>

<div style="padding:2px; margin-bottom:5px; text-align: center; background-color: lightskyblue;border-radius: 10px; ">

<button name="type" style="color: #000aae; font: 18px Arial;" class="togglebtn" type="SUBMIT" value="Solve">Solve</button>

<div id="msg" style="height:22px; margin:2px; font: 20px Arial; text-align: center; color: red; "><%=message%></div></div>

</form></div>

<div style="font: 11px arial; color: #6600cc;">

<textarea id="w3review" name="w3review" rows="4" cols="50">

<%=result%>

</textarea></div>

</div>

<script>

function ConvertStringToMat(chararray)

{

var index = 0;

var matrows = chararray[index];

index++;

var matcols = chararray[index];

index++;

var mat = new Array(matcols);

for (var i = 0; i < matcols; i++)

{

mat[i] = new Array(matrows);

for (var j = 0; j < matrows; j++)

{

mat[i][j] = parseFloat(chararray[index]);

index++;

}

}

return mat;

}

function UpdateHeaderRow(tablename)

{

var array = ["a", "b", "c", "d", "e", "f", "g", "h"]

table = document.getElementById(tablename)

table.deleteRow(0);

var row = table.insertRow(0);

for (var i = 0; i < table.rows[1].cells.length; i++)

{

var headerCell = document.createElement("TH");

headerCell.innerHTML = array[i];

row.appendChild(headerCell);

}

}

function UpdateSizeValues(tablename)

{

var table = document.getElementById(tablename);

var title = document.getElementById(tablename + "Title");

var titleinput = document.getElementById(tablename + "Size");

title.innerHTML = table.id.slice(-1) + ": " + String(table.rows.length - 1) + "x" + String(table.rows[0].cells.length)

titleinput.value = String(table.rows.length - 1) + "x" + String(table.rows[0].cells.length)

}

function CreateCell(columnindex, rowindex, table, textvalue) {

var cell = table.rows[rowindex].insertCell(columnindex);

cell.innerHTML = '';

var input = document.createElement('input');

//var text = "NEW CELL" + String(columnindex);

input.type = "text";

input.value = textvalue;

input.name = table.id.slice(-1) + "\_" + String(columnindex) + "\_" + String(rowindex - 1); //first num is left to right, second num is from top to bottom.

input.style = "color: #0000ff; background-color: #eeffee; text-align:center; font: 15px Arial; width:40px; border-radius: 10px; ";

cell.appendChild(input);

}

function CreateRow(tablename) {

var table = document.getElementById(tablename);

var row = table.insertRow(table.rows.length);

for (var i = 0; i < table.rows[0].cells.length; i++) { //to add cells to the previous rows for square matrix

/\*if (i < table.rows.length - 1) {

CreateCell(i, table.rows.length - 1, table, 0);

}\*/

CreateCell(i, table.rows.length - 1, table, 0);

}

UpdateSizeValues(tablename);

}

function CreateColumn(tablename)

{

var table = document.getElementById(tablename);

var len = table.rows[0].cells.length;

for (var j = 1; j < table.rows.length; j++)

{

CreateCell(len, j, table, 0);

}

UpdateHeaderRow(tablename);

UpdateSizeValues(tablename);

}

function DeleteRow(tablename) {

var table = document.getElementById(tablename);

if (table.rows.length > 0)

{

table.deleteRow(table.rows.length - 1);

UpdateSizeValues(tablename);

}

}

function DeleteColumn(tablename)

{

var table = document.getElementById(tablename);

if (table.rows[0].cells.length > 0)

{

for (var i = 0; i < table.rows.length; i++) {

table.rows[i].deleteCell(table.rows[i].cells.length - 1);

}

UpdateSizeValues(tablename);

}

}

function ConvertTo2dArray(tablename)

{

var table = document.getElementById(tablename);

var matrix = new Array(table.rows[0].cells.length);

for (var i = 0; i < table.rows[0].cells.length; i++)

{

matrix[i] = new Array(table.rows.length);

for (var j = 0; j < table.rows.length; j++)

{

matrix[i][j] = table.rows[j].cells[i].firstChild.value;

}

}

return matrix;

}

function GenerateMatrix(tablename, matrix)

{

var table = document.getElementById(tablename);

// delete table

var len = table.rows.length

for (var i = 0; i < len; i++)

{

table.deleteRow(0);

}

table.insertRow(0);

for (var j = 0; j < matrix[0].length; j++)

{

table.insertRow(j+1);

for (var i = 0; i < matrix.length; i++)

{

CreateCell(i, j+1, table, matrix[i][j]);

}

}

UpdateHeaderRow(tablename);

UpdateSizeValues(tablename);

}

</script>

<script type="text/javascript">

function startup() {

var predefined = [false];

var matrixLetters = ["A"];

var matrices = ["<%=matrixAString%>"];

for (var i = 0; i < matrices.length; i++)

{

if (matrices[i] != "") {

predefined[i] = true;

//console.log("string for matrix" + matrixLetters[i] + ": " + matrices[i]);

var matrix = ConvertStringToMat(matrices[i].split(" "));

GenerateMatrix("Matrix" + matrixLetters[i], matrix);

}

}

for (var i = 0; i < matrices.length; i++)

{

if (!predefined[i])

{

var nrows = 2

CreateRow("Matrix" + matrixLetters[i]);

for (var j = 0; j < nrows; j++)

{

//console.log("adding rows and cols for Matrix" + matrixLetters[i]);

CreateRow("Matrix" + matrixLetters[i]);

CreateColumn("Matrix" + matrixLetters[i]);

}

}

}

}

window.onload = startup;

</script>

</body>

// end of code

GUI/views/menu.ejs:

//start of code

<ul>

<li><a href="matrix">matrix calculator</a></li>

<li><a href="simplex">simplex solver</a></li>

<li><a href="simul">simultaneous equations solver</a></li>

</ul>

//end of code

**Testing**

**Matrix calculator**

**matrix multiplication**

|  |  |  |  |
| --- | --- | --- | --- |
| **type** | **input** | **output** | **iscorrect** |
| **square matrix** |  |  | **Y** |
| **square matrix** | A screenshot of a phone  Description automatically generated with medium confidence | A screen shot of a phone  Description automatically generated with low confidence | **Y** |
| **non-square matrix** | Graphical user interface, application  Description automatically generated | Graphical user interface, text, application  Description automatically generated | **Y** |
| **1x1** | Graphical user interface, application  Description automatically generated | A picture containing graphical user interface  Description automatically generated | **Y** |
| **invalid input** | Graphical user interface, application  Description automatically generated | **“Error: Matrix contains invalid characters”** | **Y** |
| **invalid order of matrices** | Graphical user interface, application  Description automatically generated | “Error: MatrixA width != MatrixB height. Therefore matrix multiplication for A and B is not defined” | **Y** |

**matrix addition**

|  |  |  |  |
| --- | --- | --- | --- |
| **type** | **input** | **output** | **iscorrect** |
| **square matrix** |  |  | **Y** |
| **large square matrix** |  |  | **Y** |
| **non-square matrix** |  |  | **Y** |
| **invalid input** |  | **“** Error: Matrix contains invalid characters” | **Y** |
| **invalid order** |  | **“** Error: MatrixAOrder != MatrixBOrder. Therefore matrix addition/subtraction for A and B is not defined” | **Y** |

**matrix subtraction**

|  |  |  |  |
| --- | --- | --- | --- |
| **type** | **input** | **output** | **iscorrect** |
| **square matrix** |  |  | **Y** |
| **large square matrix** |  |  | **Y** |
| **non-square matrix** |  |  | **Y** |
| **invalid input** |  | **“** Error: Matrix contains invalid characters” | **Y** |
| **invalid order** |  | **“** Error: MatrixAOrder != MatrixBOrder. Therefore matrix addition/subtraction for A and B is not defined” | **Y** |

**finding determinant of a matrix**

|  |  |  |  |
| --- | --- | --- | --- |
| **type** | **input** | **output** | **iscorrect** |
| **square matrix** |  |  | **Y** |
| **large square matrix** |  |  | **Y** |
| **invalid input** |  | **“** Error: Matrix contains invalid characters” | **Y** |
| **invalid order** |  | **“**Error: non-square matrix does not have a determinant**”** | **Y** |
| **1x1 matrix** |  |  | **Y** |

**finding inverse of a matrix**

|  |  |  |  |
| --- | --- | --- | --- |
| **type** | **input** | **output** | **iscorrect** |
| **square matrix** |  |  | **Y** |
| **large square matrix** |  |  | **Y** |
| **invalid input** |  | **“** Error: Matrix contains invalid characters” | **Y** |
| **invalid order** |  | **“**Error: non-square matrix has no inverse**”** | **Y** |
| **1x1 matrix** |  |  | **Y** |

**transposing a matrix**

|  |  |  |  |
| --- | --- | --- | --- |
| **type** | **input** | **output** | **iscorrect** |
| **square matrix** |  |  | **Y** |
| **large square matrix** |  |  | **Y** |
| **invalid input** |  | **“** Error: Matrix contains invalid characters” | **Y** |
| **invalid order** |  | “Error: non-square matrix cannot be transposed” | **Y** |
| **1x1 matrix** |  |  | **Y** |

**finding the minor of a matrix at a given element**

|  |  |  |  |
| --- | --- | --- | --- |
| **type** | **input** | **output** | **iscorrect** |
| **square matrix** |  |  | **Y** |
| **large square matrix** |  |  | **Y** |
| **invalid input** |  | **“** Error: Matrix contains invalid characters” | **Y** |
| **no index i specified** |  | **“**Error: minor index i must be an integer**”** | **Y** |
| **no index j specified** |  | **“**Error: minor index j must be an integer” | **Y** |
| **invalid order** |  | **“**Error: non-square matrix does not have a minor**”** | **Y** |
| **1x1 matrix** |  | “Error: matrix with order 1x1 does not have a minor” | **Y** |

**multiplication of matrix by constant**

|  |  |  |  |
| --- | --- | --- | --- |
| **type** | **input** | **output** | **iscorrect** |
| **square matrix** |  |  | **Y** |
| **large square matrix** |  |  | **Y** |
| **non-square matrix** |  |  | **Y** |
| **invalid input** |  | **“** Error: Matrix contains invalid characters” | **Y** |
| **no constant specified** |  | **“**Error: constant must be a real number**”** | **Y** |

**Simplex solver:**

|  |  |  |  |
| --- | --- | --- | --- |
| **type** | **input** | **output** | **iscorrect** |
| **no artificial variables** |  |  | **Y** |
| **no artificial variables more constraints** |  |  | **Y** |
| **with artificial variables** |  |  | **Y** |
| **invalid input** |  |  | Y |

**simultaneous equations solver:**

|  |  |  |  |
| --- | --- | --- | --- |
| **type** | **input** | **output** | **iscorrect** |
| **two variables** |  |  | **Y** |
| **three variables** |  |  | **Y** |
| **invalid input** |  | **“**Error: simultaneous equations contain invalid characters” | **Y** |
| **no solution** |  | **“**Error: this system of simultaneous equations does not have a unique solution” | **Y** |

**User interface:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **input** | **output** | **iscorrect?** |
| **output matrices** | **matrices a and b** | **matrix c** | **yes (as shown from matrix test section)** |
| **swap matrices** |  |  | **Y** |
| **render matrix calculator web page** | **GET request to /matrix** |  | **Y** |
| **render simultaneous equations web page** | **GET request to /simul** |  | **Y** |
| **render simplex solver web page** | **GET request to /simplex** |  | **Y** |
| **render error messages** | **input erroneous input** | **e.g.** | **Yes as shown from other test sections.** |
| **generate simplex tableau** |  |  | **Y** |

**Evaluation:**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **objective** | **met?** | **comment** |
| **1a** | The program should be in the form of a website | **yes** |  |
| **1b** | The website should have an easy-to-use interface | **yes** | **The client feedback suggests that the interface is easy to use.** |
| **1c** | The website should be compatible with all widely used browsers | **yes** |  |
| **2a** | The matrix calculator should be able to handle matrices of any order | **yes** |  |
| **2bi** | Matrix multiplication | **yes** |  |
| **2bii** | Matrix addition | **yes** |  |
| **2biii** | Matrix subtraction | **yes** |  |
| **2biv** | Finding the inverse of a matrix | **yes** |  |
| **2bv** | Finding the determinant of a matrix | **yes** |  |
| **2bvi** | Transposing a matrix | **yes** |  |
| **2bvii** | Finding the minor of a matrix from any element in that matrix | **yes** |  |
| **2bviii** | multiply a matrix by a constant | **yes** |  |
| **2c** | The matrix calculator should be return relevant error messages when the user enters an input that is invalid for the operation he wants to use | **yes** |  |
| **2d** | The matrix calculator should have a button for each matrix operation | **yes** |  |
| **2e** | the values in each input matrix should remain after an operation, successful or unsuccessful so that the user can reuse them | **yes** |  |
| **2f** | The user should be able to copy the output of an operation to one of the input matrices and vice-versa | **yes** |  |
| **2g** | The user should be able to swap the input matrices | **yes** |  |
| **3a** | The simplex solver should be able to accept inputs that are in standard form | **yes** |  |
| **3b** | The website should be able to generate a table based on the number of variables, slack variables, artificial variables and constraints for the user to input his data | **yes** |  |
| **3c** | The output of a simplex algorithm should be in text format and easy to read. | **yes** |  |
| **4a** | The user should be able to enter a polynomial in standard form | **no** |  |
| **4b** | The root finding algorithm should be able to find the roots to 10 significant figures | **no** |  |
| **4c** | The output should be displayed in text format | **no** |  |

**feedback from user:**

The website interface was functional and easy to use, however the elements on the web pages could have been placed and styled better, especially the main menu.

The matrix calculator makes it easy to perform calculations quickly because you can reuse the answer to the previous calculation in a new calculation and your inputs don’t disappear when a calculation is performed. The fact that it can perform calculations on matrices of any order is also useful. The system for adding rows and removing rows to the matrix calculator and choosing an operation to perform on one or two of the matrices are intuitive. However, the matrix calculator could be improved by enabling the cells in each table to adjust their width based on their content, because to see a long number, you need to scroll through the cell which is unhelpful. This also makes it difficult to see which input fields are for the constant and which are for the indexes of the element used when finding a minor.

The simplex calculator is straight forward to use; however, it would be better if you could input constraints and the objective function in the form of inequalities and equations instead of having to convert them to linear programming form before inputting them into the table.

The simultaneous equations solver is also well designed but suffers the same problem than the matrix calculator in that the cells don’t adjust to the size of the content within them.

My biggest criticism is that the polynomial solver was not implemented.

**comments on feedback:**

The feedback from my client suggests that my application is fully functional apart from the polynomial-solver for which I didn’t have time to implement the user interface but I have written the c++ library file for. It suggests that there are improvements to be made including the lack of resizing of the cells in the tables when displaying large content and that the simplex solver only accepts input that is in linear programming form.

**improvements:**

To improve the webpage layout I should have placed the table input fields more centrally and used CSS to make the elements on the web pages look more tidy.

To make the cells resize when they contain a large input, I should have added a javascript function to dynamically change the width of the cells to fit the input.

To make the application more complete, I should have implemented a user interface for the input of polynomial functions so that the user could use that part of my library.

**glossary:**

**matrix:** A 2d array of data where the position of each element can identified using [i, j] where i represents the horizontal position and j represents the vertical position and [0,0] is the top left element.

**leading diagonal:** collection of entries {i,j} where i = j.

**2d pointer array:** An array of pointers where each point to the address in memory of the first element in an array.