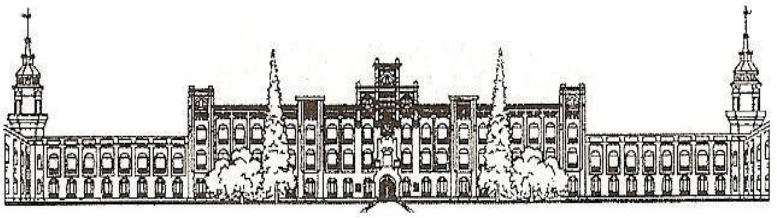

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут» ім. Ігоря Сікорського

Лабораторна робота №3

з дисципліни «Алгоритми та методи обчислень»

«Розв'язання систем лінійних алгебраїчних рівнянь»

Виконав студент групи: КВ-33

Козлов Сергій

Київ 2025

Мета роботи

Мета роботи – дослідити прямі та ітераційні методи розв’язання систем лінійних алгебраїчних рівнянь.

Завдання за варіантом

0 – схема єдиного поділу, метод ітерації Зейделя;

6	14	19	15	4	180
	17	33	5	10	208
	11	6	28	10	230
	6	19	3	13	149

Блок-схеми алгоритмів

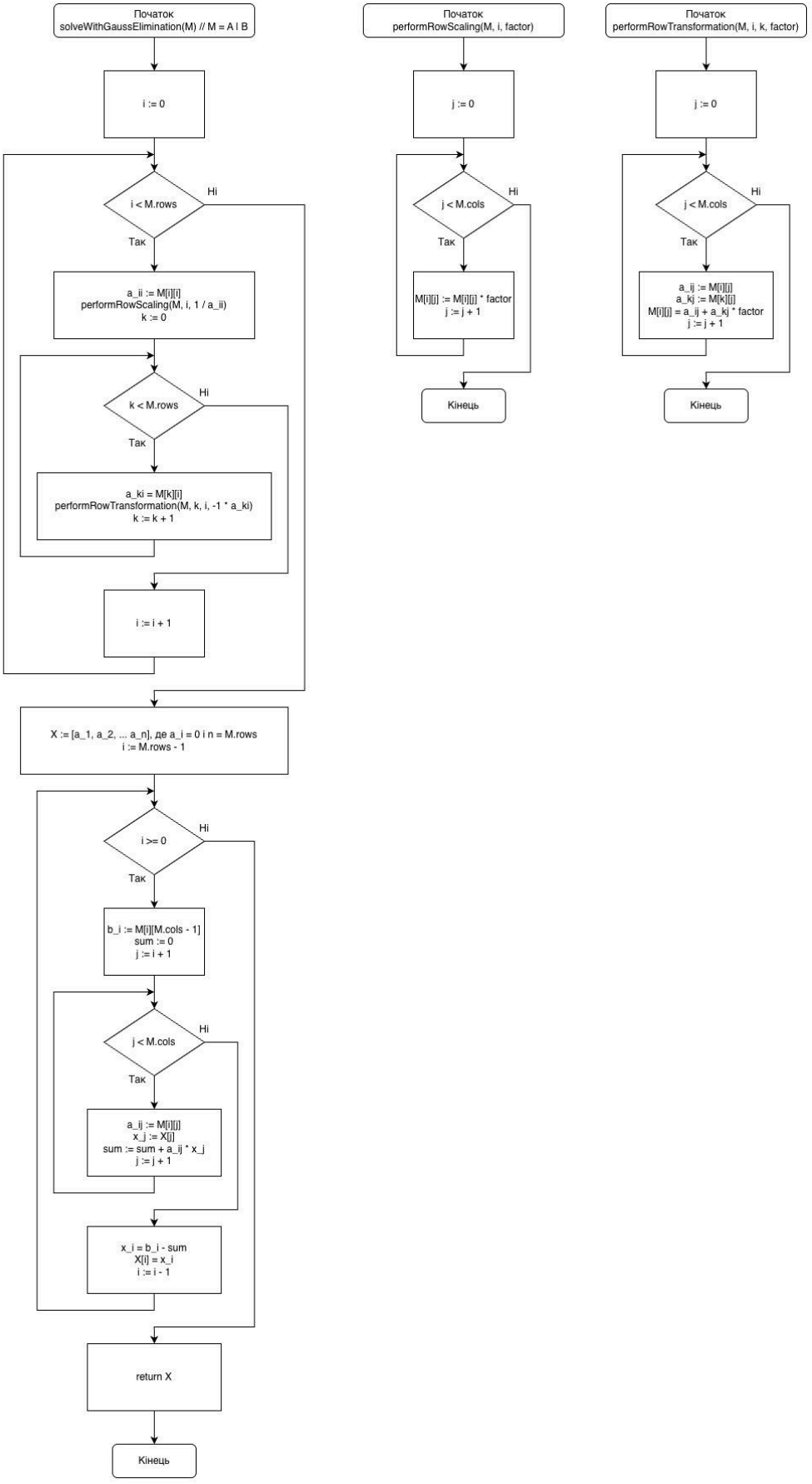

Рис. 1 - Блок-схема алгоритму єдиного поділу

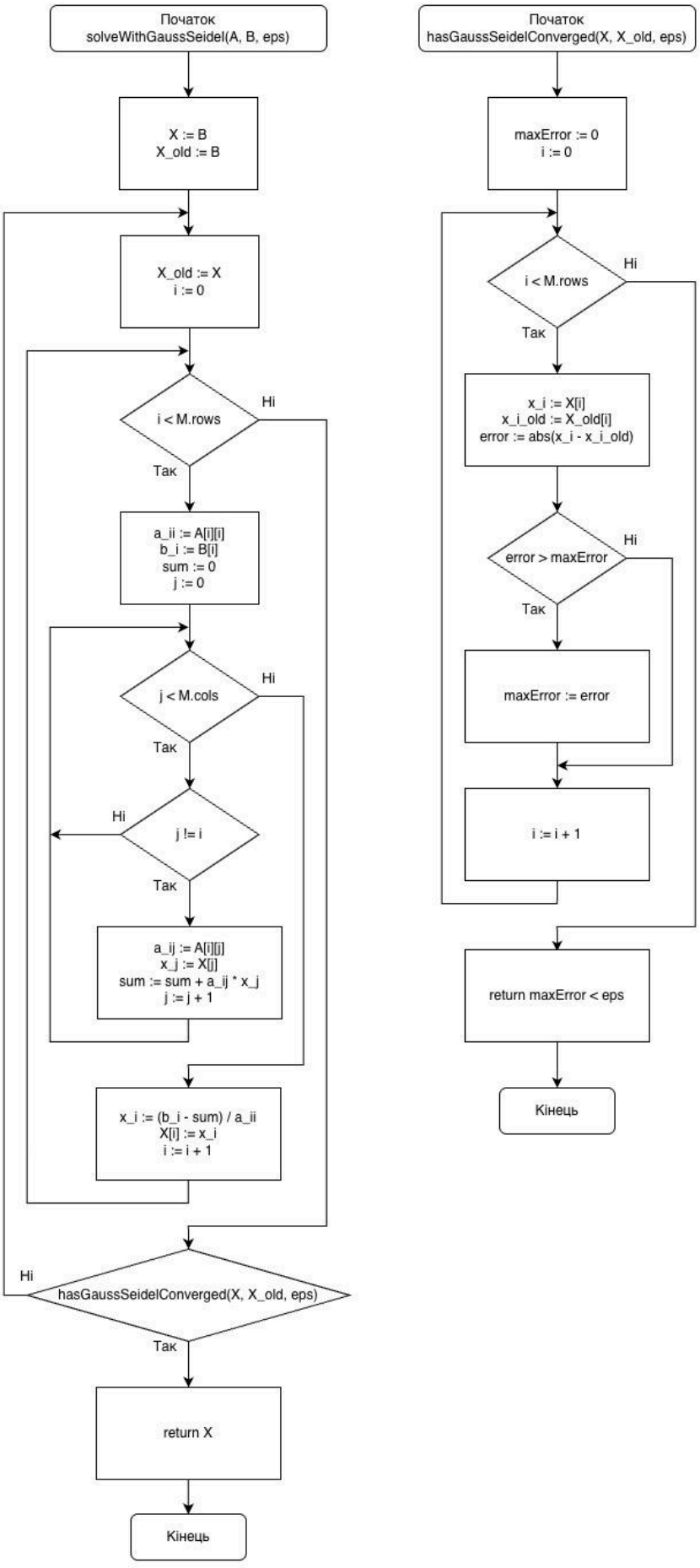

Рис. 2 - Блок-схема алгоритму Зейделя

Підготовчий етап

Оскільки визначник заданої матриці M ≠ 0, то СЛАР має єдиний розв'язок.

Отже, метод єдиного поділу (метод Гаусса) можна використовувати без попередньої підготовки.

Умовою збіжності ітераційного алгоритму Зейделя є знаходження матриці коефіцієнтів в діагонально домінантній формі. Перевіримо цю умову:

$$\left|\begin{array}{cccc|c} 14 & 19 & 15 & 4 & 180 \\ 17 & 33 & 5 & 10 & 208 \\ 11 & 6 & 28 & 10 & 230 \\ 6 & 19 & 3 & 13 & 149 \end{array}\right|$$

19 + 15 + 4 = 38 > 14 - НЕ виконується

17 + 5 + 10 = 32 < 33 - виконується

11 + 6 + 10 = 27 < 28 - виконується

6 + 19 + 3 = 28 > 13 - НЕ виконується

Отже, для рядків (2) і (3) умова діагональної домінантності виконується. Приведемо СЛАР до потрібної форми:

$$\left|\begin{array}{cccc|c} 14 & 19 & 15 & 4 & 180 \\ 17 & 33 & 5 & 10 & 208 \\ 11 & 6 & 28 & 10 & 230 \\ 6 & 19 & 3 & 13 & 149 \end{array}\right|$$

(4) ⇒ (4) - 3(1) + 1(2) + 2(3)

$$\left|\begin{array}{cccc|c} 14 & 19 & 15 & 4 & 180 \\ 17 & 33 & 5 & 10 & 208 \\ 11 & 6 & 28 & 10 & 230 \\ 3 & 7 & 19 & 31 & 277 \end{array}\right|$$

(1) ⇒ 100(1) - 55(2) - 60(3) + 24(4)

$$\left|\begin{array}{cccc|c} -123 & -107 & 1 & -6 & -592 \\ 17 & 33 & 5 & 10 & 208 \\ 11 & 6 & 28 & 10 & 230 \\ 3 & 7 & 19 & 31 & 277 \end{array}\right|$$

Матриця приведена до діагонально домінантного вигляду.

Обчислення коренів СЛАР

```
Testing Gaussian Elimination:
Original Matrix A augmented with B:
[  14.0000,   19.0000,   15.0000,    4.0000,  180.0000]
[  17.0000,   33.0000,    5.0000,   10.0000,  208.0000]
[  11.0000,    6.0000,   28.0000,   10.0000,  230.0000]
[   6.0000,   19.0000,    3.0000,   13.0000,  149.0000]

Solution with Gaussian Elimination:
[   2.0000,    3.0000,    5.0000,    5.0000]
```

Рис. 3 - результат обчислень методом єдиного поділу

```
Testing Gauss-Seidel Method:
Original Matrix A augmented with B:
[  14.0000,   19.0000,   15.0000,    4.0000,  180.0000]
[  17.0000,   33.0000,    5.0000,   10.0000,  208.0000]
[  11.0000,    6.0000,   28.0000,   10.0000,  230.0000]
[   6.0000,   19.0000,    3.0000,   13.0000,  149.0000]

Diagonally dominant Matrix A augmented with B:
[-123.0000, -107.0000,    1.0000,   -6.0000, -592.0000]
[  17.0000,   33.0000,    5.0000,   10.0000,  208.0000]
[  11.0000,    6.0000,   28.0000,   10.0000,  230.0000]
[   3.0000,    7.0000,   19.0000,   31.0000,  277.0000]

Is matrix A diagonally dominant? Yes

Solution with Gauss-Seidel:
[   2.0000,    3.0000,    5.0000,    5.0000]
```

Рис. 4 - результат обчислень методом Зейделя з точністю $1*10^{-5}$

Обидва методи, дають однаковий результат. Виконаємо перевірку:

14 * 2 + 19 * 3 + 15 * 5 + 4 * 5 = 28 + 57 + 75 + 20 = 180

Отже, обидва методи працюють правильно.

Висновок

Під час виконання лабораторної роботи були досліджені прямі і ітераційні методи розв’язання СЛАР – конкретно, метод єдиного поділу (метод Гаусса) і ітераційний метод Зейделя (Гаусса-Зейделя).

Для дослідження і перевірки роботи методів була розроблена програма мовою C. Обидва методи повернули результат з бажаною точністю, однак вони відміння в сценаріях оптимального застосування.

Метод єдиного поділу простий для розуміння, але має часову складність $O(N^3)$, де N це кількість невідомих. Це робить його недоречним для розв’язання великих СЛАР.

Метод Зейделя має часову складність $O(N^2)$. Однак, за умовою сходження, матриця коефіцієнтів повинна бути діагонально домінантною. Приведення СЛАР до такого вигляду може бути нетривіальним навіть для простих систем, подібних до розглянутої. Отже, цей метод має місце при розрахунку СЛАР, які образу мають правильну форму, або можуть бути легко до неї приведені.

В інших випадках варто звернутись до інших ітераційних методів – наприклад алгоритму SOR (Successive Over-Relaxation).

Вихідний код програми

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

#include "gauss-seidel/gauss-seidel.h"
#include "gauss/gauss.h"
#include "matrix/matrix.h"
#include "vector/vector.h"

void testGaussElimination() {
 // clang-format off
 double a_b[] = {
   14, 19, 15,  4, 180,
   17, 33,  5, 10, 208,
   11,  6, 28, 10, 230,
    6, 19,  3, 13, 149,
 };
 // clang-format on

 printf("Testing Gaussian Elimination:\n");

 Matrix A_B = createMatrix(4, 5, a_b);
 char *A_B_str = matrixToString(A_B);
 printf("Original Matrix A augmented with B:\n%s\n", A_B_str);
 free(A_B_str);

 Vector X = solveWithGaussianElimination(A_B);
 char *X_str = vectorToString(X);
 printf("Solution with Gaussian Elimination:\n%s\n", X_str);
 free(X_str);

 destroyMatrix(A_B);
 destroyVector(X);
}

void testGaussSeidel() {
 // clang-format off
 double a_b[] = {
   14, 19, 15,  4, 180, // 14 < 19 + 15 + 4 = 38 (not diagonally dominant)
   17, 33,  5, 10, 208, // 33 > 17 + 5 + 10 = 32
   11,  6, 28, 10, 230, // 28 > 11 + 6 + 10 = 27
    6, 19,  3, 13, 149, // 13 < 6 + 19 + 3 = 28 (not diagonally dominant)
 };
 // clang-format on

 printf("Testing Gauss-Seidel Method:\n");

 Matrix A_B = createMatrix(4, 5, a_b);

 char *A_B_str = matrixToString(A_B);
 printf("Original Matrix A augmented with B:\n%s\n", A_B_str);
 free(A_B_str);

 // Make row (4) diagonally dominant
 // (4) = (4) - 3*(1) + (2) + 2*(3)
 performRowTransformation(A_B, 3, 0, -3.0);
 performRowTransformation(A_B, 3, 1, 1.0);
 performRowTransformation(A_B, 3, 2, 2.0);

 // Make row (1) diagonally dominant
 // (1) = 100*(1) - 55*(2) - 60*(3) - 24*(4)
```

```
 performRowScaling(A_B, 0, 100);
 performRowTransformation(A_B, 0, 1, -55.0);
 performRowTransformation(A_B, 0, 2, -60.0);
 performRowTransformation(A_B, 0, 3, 24.0);

 char *A_B_diagonal_str = matrixToString(A_B);
 printf("Diagonally dominant Matrix A augmented with B:\n%s\n",
A_B_diagonal_str);
 free(A_B_diagonal_str);

 Matrix A = getMatrixSubmatrix(A_B, 0, 0, 4, 4);
 Vector B = getMatrixColumn(A_B, 4);
 printf("Is matrix A diagonally dominant? %s\n\n",
isMatrixDiagonallyDominant(A) ? "Yes" : "No");

 Vector X = solveWithGaussSeidel(A, B, 0.00001);  // 10^-5
 char *X_str = vectorToString(X);
 printf("Solution with Gauss-Seidel:\n%s\n", X_str);
 free(X_str);

 destroyMatrix(A_B);
 destroyMatrix(A);
 destroyVector(B);
 destroyVector(X);
}

int main() {
 printf("\n\n");

 testGaussElimination();

 printf("\n-----------------------\n\n");

 testGaussSeidel();

 printf("\n\n");

 return 0;
}
```

```c
// vector.h
#ifndef VECTOR_H
#define VECTOR_H

typedef struct {
 int length;
 double *data;
} Vector;

Vector createVector(int length, double *data);
Vector cloneVector(Vector vector);
void destroyVector(Vector vector);

double getVectorElement(Vector vector, int index);
void setVectorElement(Vector vector, int index, double value);

char *vectorToString(Vector vector);

#endif
```

```c
// vector.c
#include "vector.h"

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

Vector createVector(int length, double *data) {
 Vector vector;
 vector.length = length;
 vector.data = (double *) malloc(length * sizeof(double));

 if (data == NULL) {
   for (int i = 0; i < length; i++) {
     vector.data[i] = 0.0;
   }
   return vector;
 }

 for (int i = 0; i < length; i++) {
   vector.data[i] = data[i];
 }
 return vector;
}

Vector cloneVector(Vector vector) {
 Vector newVector = createVector(vector.length, NULL);

 for (int i = 0; i < vector.length; i++) {
   newVector.data[i] = vector.data[i];
 }

 return newVector;
}

void destroyVector(Vector vector) {
 free(vector.data);
}

double getVectorElement(Vector vector, int index) {
 return vector.data[index];
}

void setVectorElement(Vector vector, int index, double value) {
 vector.data[index] = value;
}

char *vectorToString(Vector vector) {
 char *buffer = (char *) malloc(256 * sizeof(char));
 int offset = 0;

 offset += sprintf(buffer + offset, "[");

 for (int i = 0; i < vector.length; i++) {
   offset += sprintf(buffer + offset, "% 9.4f", vector.data[i]);

   if (i < vector.length - 1) {
     offset += sprintf(buffer + offset, ", ");
   }
 }

 offset += sprintf(buffer + offset, "]");

 return buffer;
}
```

```c
// matrix.h
#ifndef MATRIX_H
#define MATRIX_H

#include "../vector/vector.h"

typedef struct {
 int rows;
 int cols;
 double *data;
} Matrix;

Matrix createMatrix(int rows, int cols, double *data);
Matrix cloneMatrix(Matrix matrix);
void destroyMatrix(Matrix matrix);

double getMatrixElement(Matrix matrix, int i, int j);
void setMatrixElement(Matrix matrix, int i, int j, double value);

Vector getMatrixColumn(Matrix matrix, int j);
Matrix getMatrixSubmatrix(Matrix matrix, int i_start, int j_start, int rows,
int cols);

void performRowScaling(Matrix matrix, int i, double scalar);
void performRowTransformation(Matrix matrix, int i, int k, double scalar);

char *matrixToString(Matrix matrix);

#endif
```

```c
// matrix.c
#include "matrix.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "../vector/vector.h"

Matrix createMatrix(int rows, int cols, double *data) {
 Matrix matrix;
 matrix.rows = rows;
 matrix.cols = cols;
 matrix.data = (double *) malloc(rows * cols * sizeof(double));

 if (data == NULL) {
   for (int i = 0; i < rows; i++) {
     for (int j = 0; j < cols; j++) {
       matrix.data[i * cols + j] = 0.0;
     }
   }

   return matrix;
 }

 for (int i = 0; i < rows; i++) {
   for (int j = 0; j < cols; j++) {
     matrix.data[i * cols + j] = data[i * cols + j];
   }
 }

 return matrix;
}

Matrix cloneMatrix(Matrix matrix) {
 return createMatrix(matrix.rows, matrix.cols, matrix.data);
}

void destroyMatrix(Matrix matrix) {
 free(matrix.data);
}

double getMatrixElement(Matrix matrix, int i, int j) {
 return matrix.data[i * matrix.cols + j];
}

void setMatrixElement(Matrix matrix, int i, int j, double value) {
 matrix.data[i * matrix.cols + j] = value;
}

Vector getMatrixColumn(Matrix matrix, int j) {
 Vector column = createVector(matrix.rows, NULL);

 for (int i = 0; i < matrix.rows; i++) {
   double value = getMatrixElement(matrix, i, j);

   setVectorElement(column, i, value);
 }

 return column;
}

Matrix getMatrixSubmatrix(Matrix matrix, int i_start, int j_start, int rows,
int cols) {
```

```
 Matrix submatrix = createMatrix(rows, cols, NULL);

 for (int i = 0; i < rows; i++) {
   for (int j = 0; j < cols; j++) {
     double value = getMatrixElement(matrix, i_start + i, j_start + j);

     setMatrixElement(submatrix, i, j, value);
   }
 }

 return submatrix;
}

void performRowScaling(Matrix matrix, int i, double scalar) {
 for (int j = 0; j < matrix.cols; j++) {
   double value = getMatrixElement(matrix, i, j);

   setMatrixElement(matrix, i, j, value * scalar);
 }
}

void performRowTransformation(Matrix matrix, int i, int k, double scalar) {
 for (int j = 0; j < matrix.cols; j++) {
   double a_ij = getMatrixElement(matrix, i, j);
   double a_kj = getMatrixElement(matrix, k, j);

   setMatrixElement(matrix, i, j, a_ij + scalar * a_kj);
 }
}

char *matrixToString(Matrix matrix) {
 char *buffer = (char *) malloc(matrix.rows * 256);
 buffer[0] = '\0';

 char rowBuffer[256];

 for (int i = 0; i < matrix.rows; i++) {
   int offset = 0;

   offset += sprintf(rowBuffer + offset, "[");

   for (int j = 0; j < matrix.cols; j++) {
     float a_ij = getMatrixElement(matrix, i, j);

     offset += sprintf(rowBuffer + offset, "% 9.4f", a_ij);

     if (j < matrix.cols - 1) {
       offset += sprintf(rowBuffer + offset, ", ");
     }
   }

   offset += sprintf(rowBuffer + offset, "]\n");
   strcat(buffer, rowBuffer);
 }

 return buffer;
}
```

```h
// gauss.h
#ifndef GAUSS_H
#define GAUSS_H

#include "../matrix/matrix.h"
#include "../vector/vector.h"

Vector solveWithGaussianElimination(Matrix augmentedMatrix);

void performForwardElimination(Matrix augmentedMatrix);

Vector performBackwardSubstitution(Matrix upperTriangularMatrix);

#endif
```

```c
// gauss.c
#include "gauss.h"

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

#include "../matrix/matrix.h"
#include "../vector/vector.h"

Vector solveWithGaussianElimination(Matrix A_B) {
 performForwardElimination(A_B);

 return performBackwardSubstitution(A_B);
}

void performForwardElimination(Matrix A_B) {
 for (int i = 0; i < A_B.rows; i++) {
   double a_ii = getMatrixElement(A_B, i, i);

   performRowScaling(A_B, i, 1.0 / a_ii);

   for (int k = i + 1; k < A_B.rows; k++) {
     double a_ki = getMatrixElement(A_B, k, i);

     performRowTransformation(A_B, k, i, -1.0 * a_ki);
   }
 }
}

Vector performBackwardSubstitution(Matrix matrix) {
 Vector X = createVector(matrix.rows, NULL);

 for (int i = matrix.rows - 1; i >= 0; i--) {
   double b_i = getMatrixElement(matrix, i, matrix.cols - 1);

   double sum = 0.0;

   for (int j = i + 1; j < matrix.cols - 1; j++) {
     double a_ij = getMatrixElement(matrix, i, j);
     double x_j = getVectorElement(X, j);

     sum += a_ij * x_j;
   }

   double x_i = b_i - sum;

   setVectorElement(X, i, x_i);
 }

 return X;
}
```

```c
// gauss-seidel.h
#ifndef GAUSS_SEIDEL_H
#define GAUSS_SEIDEL_H

#include "../matrix/matrix.h"
#include "../vector/vector.h"

typedef struct {
 Matrix L;
 Matrix U;
} GaussSeidelSplitResult;

Vector solveWithGaussSeidel(Matrix A, Vector B, double eps);

int hasGaussSeidelConverged(Vector X, Vector X_old, double eps);

int isMatrixDiagonallyDominant(Matrix A);

#endif
```

```c
// gauss-seidel.c
#include "gauss-seidel.h"

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

#include "../matrix/matrix.h"
#include "../vector/vector.h"

Vector solveWithGaussSeidel(Matrix A, Vector B, double eps) {
 Vector X = cloneVector(B);
 Vector X_old = cloneVector(B);

 do {
   destroyVector(X_old);
   X_old = cloneVector(X);

   for (int i = 0; i < A.rows; i++) {
     double a_ii = getMatrixElement(A, i, i);

     double b_i = getVectorElement(B, i);

     double sum = 0.0;

     for (int j = 0; j < A.cols; j++) {
       if (j == i) {
         continue;
       }

       double a_ij = getMatrixElement(A, i, j);
       double x_j = getVectorElement(X, j);

       sum += a_ij * x_j;
     }

     double x_i = (b_i - sum) / a_ii;

     setVectorElement(X, i, x_i);
   }
 } while (!hasGaussSeidelConverged(X, X_old, eps));

 destroyVector(X_old);

 return X;
}

int hasGaussSeidelConverged(Vector X, Vector X_old, double eps) {
 double maxError = 0.0;

 for (int i = 0; i < X.length; i++) {
   double x_i = getVectorElement(X, i);
   double x_i_old = getVectorElement(X_old, i);

   double error = fabs(x_i - x_i_old);

   if (error > maxError) {
     maxError = error;
   }
 }

 return maxError < eps;
}
```

```
int isMatrixDiagonallyDominant(Matrix A) {
 for (int i = 0; i < A.rows; i++) {
   double a_ii = fabs(getMatrixElement(A, i, i));
   double sum = 0.0;

   for (int j = 0; j < A.cols; j++) {
     if (i != j) {
       sum += fabs(getMatrixElement(A, i, j));
     }
   }

   if (a_ii < sum) {
     return 0;
   }
 }

 return 1;
}
```