МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 1

Вариант 1 – 1, 2 – 5

ОТЧЕТ

о выполненном задании

студента 205 учебной группы факультета ВМК МГУ

Попова Алексея Павловича

Москва, 2018 г.

Подвариант 1

Цель работы

Изучить методы решения СЛАУ с использованием классической реализации алгоритма Гаусса и алгоритма Гаусса с поиском главного элемента. Также рассмотреть варианты применения этого алгоритма для решения конкретных систем линейных алгебраических уравнений, исследовать возможность применения этих алгоритмов для нахождения обратной матрицы, нахождения определителя матрицы и ее числа обусловленности.

Постановка задачи

Дана система уравнений: $A \cdot x = f$ порядка $n \times n$, где $\det A \neq 0$. Необходимо написать программу, которая решает заданную систему линейных алгебраических уравнений с пользовательским параметром (n) где (n - размерность матрицы) при помощи классического алгоритма Гаусса и алгоритма Гаусса с выбором главного элемента. Также параллельно необходимо реализовать методы нахождения определителя матрицы заданной системы ($\det A$), её обращения (A^{-1}) и нахождения её числа обусловленности ($cond(A)$).

Реализовать возможность задания матрицы системы в файл и вывода решения СЛАУ в файл, также предусмотреть возможность задания матрицы при помощи наперед заданной формулы.

Цели и задачи практической работы

Целью данной работы является изучение классической реализации алгоритма Гаусса и его реализации с поиском главного элемента.

Также были поставлены следующие задачи:

1. Решить СЛАУ методом Гаусса и методом Гаусса с поиском главного элемента;
2. Вычислить обратную матрицу;

3. Найти число обусловленности матрицы;

4. Вычислить обратную матрицу;

5. Исследовать вопрос вычислительной устойчивости метода Гаусса;

Решение поставленных задач реализовано программно (см. Приложение 1).

Алгоритмы решения задач

Решение СЛАУ методом Гаусса

Применяя равносильные матричные преобразования строк матрицы системы (Матрицы A, системы $A \cdot x = f$) она приводится к треугольному виду (в частности предложенный алгоритм приводит матрицу к треугольному виду с 1 на главной диагонали). В том числе необходимо делать соответствующие преобразования со столбом f (или, что проще, описать эти действия над матрицей $[A\,|\,f\,]$). Описанный выше этап алгоритма называется прямым ходом метода Гаусса. Теперь, для получения решений необходимо сделать так называемый, обратный ход метода Гаусса. Для этого, последовательно применяя равносильные матричные преобразования к матрице $\hat{A}$ (где матрица $\hat{A}$ - это матрица полученная после прямого хода метода Гаусса), и к столбцу $\hat{f}$, который получен из f применением соответственных преобразований для матрицы A (или в матричном виде, в терминах присоединенной матрицы $[\hat{A}\,|\,\hat{f}\,]$). Эти преобразования производятся начиная с последней строки матрицы $\hat{A}$, тем самым происходит непосредственное вычисление корней СЛАУ. После обратного хода метода Гаусса имеем: $[I\,|\,x]$, где x - вектор решения исходной СЛАУ.

Решение СЛАУ методом Гаусса с выобром главного элемента

Опишем данный метод, обратив внимание на отличия от классической реализации метода Гаусса. В данной алгоритме на каждом шаге, при преобразовании строк производится выбор главного элемента ($a = \max a_{ij} \forall j = 1...n$) и соответственно столбец, содержащий данный элемент (столбец с номером j) меняется местами со столбцом имеющим номер i. Данное преобразование записывается в “карту преобразований”. Далее производится итерация классического метода Гаусса. После окончания прямого хода, производится обратный ход метода Гаусса, и восстанавливается решение по “карте решений” (см. Приложение 1).

Вычисление обратной матрицы

Для решения данной задачи используется метод Гаусса с выбором главного элемента, но преобразования производятся с матрицей вида: $[A\,|\,I]$, следовательно, применяя к этой матрице метод Гаусса, мы получаем матриц вида: $[I\,|\,A^{-1}]$.

Вычисление числа обусловленности матрицы

Для решения этой задачи необходим алгоритм нахождения нормы матрицы. В силу того, что все матричные нормы эквивалентны, будем использовать бесконечную матричную норму. Также необходимо искать обратную матрицу, для этого воспользуется алгоритмом описанным выше. Имеем: $condA = |A|_{\inf} \cdot |A^{-1}|_{\inf}$. Вычисление бесконечной матричной нормы производится по схеме: вычисляем суммы модулей элементов в каждой строке, находим максимальную сумму из подсчитанных, это число является бесконечной матричной нормой.

Вычисление определителя матрицы

Для решения этой задачи, достаточно на каждом шаге алгоритма Гаусса с поиском главного члена, сохранять преобразования происходящие с определителем матрицы при применении матричных преобразований. В силу того, что в итоге мы получаем треугольную матрицу с 1 на диагонали (после прямого хода) произведение сохраненных изменений и будет $\det A$.

Описание программного решения данных задач

Программа содержит пакет ввода/вывода матриц из/в файл(-ла). Это позволяет существенно сократить время тестирования и отладки программы. Ввод подразумевает специальный формат, сначала необходимо ввести n - размер матрицы, и на следующих строка задать саму матрицу системы.

Также в программе реализован консольный интерфейс (см. Приложение 1) при помощи которого можно выбрать задачу, которую необходимо решить алгоритмически и формат входных значений для данного алгоритма (поддерживается ввод матрицы задаваемый формулой).

Вывод программы представляет из себя: численное решение поставленной задачи для конкретных входных данных.

Тестирование

В процессе тестирования реализованных алгоритмов собиралась статистическая информация, характеризующая работу алгоритма и производилось тестирование на заданных тестах. Все решение были проверены при помощи ресурса *https://www.wolframalpha.com/* (см. Приложение 2).

Выводы

В результате проделанной работы были реализованы методы решения СЛАУ при помощи алгоритма Гаусса и алгоритма Гаусса с выбором главного элемента. Также было замечено, что метод теряет устойчивость по мере возрастания размеров матриц систем, подаваемых для решения, в программной реализации это проблема была решена увеличением типа данных использованного для вычислений и всевозможном сокращении потерь точности за счет сокращения вычислений, также с этой проблей вполне удачно справляется метод Гаусса с выбором главного элемента, который условно устойчив на матрицах с плохим числом обусловленности ($cond(A)$). В ходе работы был реализован алгоритм поиска числа обусловленности матрицы и ее определителя. Также на основе построенных алгоритмов, как следствие был получен алгоритм обращения матрицы.

Подвариант 2

Цель работы

Изучить классические итерационные методы используемые для численного решения СЛАУ на примере метода верхней релаксации. Также изучить скорость сходимости этих методов в зависимости от выбора итерационного параметра w.

Постановка задачи

Дана система уравнений: $A \cdot x = f$ порядка $n \times n$, где $\det A \neq 0$.
Необходимо написать программу, которая решает заданную систему линейных алгебраических уравнений с пользовательским параметром (n) где (n - размерность матрицы) при помощи метода верхней релаксации (в частности при $w = 1$ - метод верхней ралаксации превращается в классический метод Зейделя.

$$(D + A^{(-)}) \cdot (x^{k+1} - x^k) + A \cdot x^k = f,$$

где $D, A^{(-)}$ - диагональная и нижняя треугольная матрицы соответственно, k - номер текущей итерации алгоритма.
Эта формула путем несложных преобразований сводится к более понятному виду:

$$(D + w \cdot A^{(-)}) \cdot \frac{(x^{k+1} - x^k)}{w} + A \cdot x^k = f.$$

Реализовать возможность задания матрицы системы в файл и вывода решения СЛАУ в файл, также предусмотреть возможность задания матрицы при помощи наперед заданной формулы.

Цели и задачи практической работы

1. Используя итерационные методы решить СЛАУ (например: метод Зейделя или метод верхней/нижней релаксации);

2. Разработать критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы СЛАУ с наперед заданной точностью;

3. Изучить скорость сходимости итераций к точному решению задачи в зависимости от итерационного параметра w;

Решение поставленных задач реализовано программно (см. Приложение 3).

Алгоритмы решения

Решение СЛАУ методом верхенй релаксации

Данный метод позволяет при помощи последовательного приближения получить решение системы алгебраических уравнений вида $A \cdot x = f$ порядка $n \times n$, где $\det A \neq 0$ и $A = A^* > 0$. Данный метод верхней релаксации является представителем стационарных одношаговых итерационных методов линейной алгебры и записывается в виде:

$$(D + w \cdot A^{(-)}) \cdot \frac{(x^{k+1} - x^k)}{w} + A \cdot x^k = f$$

Но можно записать итерационную формулу в более понятном виде:

$$a_{ii} \cdot x_i^{k+1} = -w \cdot \sum_{j=1}^{i-1} a_{ij} \cdot x_j^{k+1} + (1 - w) \cdot a_{ii} \cdot x_i^k - w \cdot \sum_{j=i+1}^{n} a_{ij} \cdot x_j^k$$

Эта формула позволяет осуществить итерационный переход метода верхней релаксации. Посредствам последовательного приближения можно получить ответ с наперед заданной точностью.

Критерий остановки метода верхней релаксации

Как критерий остановки метода релаксации удобно использовать невязку: $|A \cdot x^k - f|_1 < \varepsilon$. Для доказательства этого факта достаточно показать, что справедливы следующие вложения:

$$\lim_{k \to \inf} |A \cdot x^k - f|_1 = 0 \Leftrightarrow \lim_{k \to \inf} |x^k - x|_1 = 0,$$

где: k - номер итерации, а x - точное решение.

Скорость сходимости в зависимости от w

Необходимое условие сходимости данного итерационного алгоритма:

$0 < w < 2$. Скорость сходимости метода верхней релаксации определяется параметром w, но подбор оптимального значения этого параметра нетривиальная задача, и поэтому мы не будем освещать ее в контексте данной задачи, а лишь напишем формулу, полученную эмпирическким путем:

$$w_{opt} = \frac{2}{1 + \sqrt{1 - \rho^2 \cdot (D^{-1} \cdot (A^{(+)} + A^{(-)}))}},$$

где $D, A^{(-)}, A^{(+)}$ - диагональная, нижняя треугольная, верхняя треугольная матрицы соответственно, а ρ - спектральный радиус матрицы A. Также в процессе решения задачи были получены статистические данные описывающие зависимость количества итераций алгоритма от значения w (см. Приложение 4).

Описание программного решения данной задачи

Программа содержит пакет ввода/вывода матриц из/в файл(-ла). Это позволяет существенно сократить время тестирования и отладки программы. Ввод подразумевает специальный формат, сначала необходимо ввести n - размер матрицы, и на следующих строка задать саму матрицу системы.

Также в программе реализован консольный интерфейс (см. Приложение 3) при помощи которого можно выбрать способ ввода информации и задать файлы ввода/вывода.

Вывод программы представляет из себя: численное решение поставленной задачи для конкретных входных данных.

Тестирование

В процессе тестирования реализованных алгоритмов собиралась статистическая информация, характеризующая работу алгоритма и производилось тестирование на заданных тестах. Производилась прогонка каждой тестовой матриц на наборах значений $0.05 \leq w \leq 1.95$ с шагом $step = 0.05$. Все решение были проверены при помощи ресурса *https://www.wolframalpha.com/* (см. Приложение 4).

Выводы

Метод верхней релаксации позволяет находить решение СЛАУ с наперед заданной точностью. Также одним из преимуществ данного метода является то, что нет необходимости производить преобразования матрицы, а работа производится только с векторами. В том числе, данный метод является итерационным, что позволяет варьировать число итераций для решения СЛАУ в зависимости от класса решаемой задачи. Немаловажно, что в процессе решения данной задачи была получена статистика по выбору начального параметра итерационного алгоритма и количество итераций с заданным параметром (см. Приложение 4), и непосредственно был разработан пакет функций (см. Приложение 3), который осуществляет решение СЛАУ методом верхней релаксации.

Приложение 1

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <math.h>

long double eps = 0.000000001;

void          simple_gaus_front( long double **A , long double *f , int n );
void          gaus_reverse( long double **A , long double *f , int n );
void          mainelem_gaus( long double **A , long double *f , int n );
long double **  rev_matrix_gaus( long double **A , int n , long double *det );
long double     cond_num( long double **A , long double **B , int n );
void          swap_line( long double **A , long double *f , int f_1 , int f_2 , int n );
void          swap_stb( long double **A , int f_1 , int f_2 , int n );
int          read_matrix_system( const char *file_name , long double ***MATRIX , long double
**STB );
int          read_matrix( const char *file_name , long double ***MATRIX );
void          write_matrix( const char *file_name , long double **A , long double *f , int n );
void          write_solution( const char *file_name , long double *f , int n );
long double **  copy_of_matrix( long double **res , long double **sen , int n );
void          make_matrix( const char *file_name , int mode );
void          free_mem( long double **A , long double *f , int n );

int
main( int argc , char **argv )
{
   //_______________________
   //its only for use this packet of meth
   if ( argc == 2 ) {
      printf( "arg1 : input file\n" );
      printf( "arg2 : output file\n" );
      printf( "arg3 : task [1 , 2 , 3 , 4 , 5 , 6]\n" );
      printf( "1 - solve by simple_gaus\n" );
      printf( "2 - solve by mainelem_gaus\n" );
      printf( "3 - find A^-1\n" );
      printf( "4 - find ObNum\n" );
      printf( "5 - find det\n" );
      printf( "6 - make matrix by formula and write it to file arg1 , and arg4 is task [1 - 5]" );
      return 0;
   }
   int task = 0;
   sscanf( argv[3] , "%d" , &task );
   if ( task == 6 ) {
      sscanf( argv[4] , "%d" , &task );
      if ( task >= 3 && task <= 5 ) {
```

```
        make_matrix( argv[1] , 1 );
    } else {
        make_matrix( argv[1] , 0 );
    }
}
unlink( argv[2] );
if ( task == 1 ) {
    long double **A = NULL , *f = NULL;
    int size = read_matrix_system( argv[1] , &A , &f );
    simple_gaus_front( A , f , size );
    gaus_reverse( A , f , size );
    write_solution( argv[2] , f , size );
    free_mem( A , f , size );
    return 0;
}
if ( task == 2 ) {
    long double **A = NULL , *f = NULL;
    int size = read_matrix_system( argv[1] , &A , &f );
    mainelem_gaus( A , f , size );
    write_solution( argv[2] , f , size );
    free_mem( A , f , size );
    return 0;
}
if ( task == 3 ) {
    long double **A = NULL;
    long double det;
    int size = read_matrix( argv[1] , &A );
    long double **B = rev_matrix_gaus( A , size , &det );
    write_matrix( argv[2] , B , NULL , size );
    free_mem( A , NULL , size );
    free_mem( B , NULL , size );
    return 0;
}
if ( task == 4 ) {
    long double **A = NULL;
    long double det;
    int size = read_matrix( argv[1] , &A );
    long double **B = rev_matrix_gaus( A , size , &det );
    long double ob = cond_num( A , B , size );
    FILE *out = fopen( argv[2] , "a" );
    fprintf( out , "ObNum: %Lf\n" , ob );
    fclose( out );
    free_mem( A , NULL , size );
    free_mem( B , NULL , size );
    return 0;
}
if ( task == 5 ) {
    long double **A = NULL;
```

```
        long double det;
        int size = read_matrix( argv[1] , &A );
        long double **B = rev_matrix_gaus( A , size , &det );
        FILE *out = fopen( argv[2] , "a" );
        fprintf( out , "Det: %Lf\n" , det );
        fclose( out );
        free_mem( A , NULL , size );
        free_mem( B , NULL , size );
        return 0;
    }
    //======================
    return 0;
}

void
simple_gaus_front( long double **A , long double *f , int n )
{
    printf( "GAUS_FRONT WAS STARTED\n" );
    //______________________
    //the list to the top go
    for ( int i = 0 ; i < n ; i++ ) { //string iterator
        //______________________
        //find non zero string and mode matrix
        int num_non_zero = i;
        for ( ; num_non_zero < n ; num_non_zero++ ) { //find not zero elem in stb
            if ( fabsl( A[num_non_zero][i] ) > eps ) {
                break;
            }
        }
        if ( i != num_non_zero ) { //if not zero elem in set position
            swap_line( A , f , i , num_non_zero , n );
        }
        //======================
        //______________________
        //devide string
        for ( int j = i + 1 ; j < n ; j++ ) { //stb iterator
            A[i][j] /= A[i][i]; //take main num
        }
        f[i] /= A[i][i];
        A[i][i] = 1;
        //======================
        //______________________
        //reduse matrix with round result
        for ( int k = i + 1 ; k < n ; k++ ) { //dec of the string to zero start number
            for ( int p = i + 1 ; p < n ; p++ ) {
                A[k][p] -= A[i][p] * A[k][i];
            }
            f[k] -= f[i] * A[k][i];
```

```
            A[k][i] = 0;
        }
        //=======================
    }
    //=======================
    printf( "GAUS_FRONT WAS FINISHED\n" );
}

void
gaus_reverse( long double **A , long double *f , int n )
{
    printf( "GAUS_REVERSE WAS STARTED\n" );
    //_______________________
    //the list to go the end
    for ( int i = n - 1 ; i >= 0 ; i-- ) { //set in the last point in matrix
        for ( int j = i - 1 ; j >= 0 ; j-- ) {
            f[j] -= A[j][i] * f[i];
            A[j][i] = 0;
        }
    }
    //=======================
    printf( "GAUS_REVERSE WAS FINISHED\n" );
}

void
mainelem_gaus( long double **A , long double *f , int n )
{
    printf( "MAINELEM_GAUS WAS STARTED\n" );
    //_______________________
    //map of stb location for allocate start position
    int *map_of_stb_location = (int *) malloc( n * sizeof( *map_of_stb_location ) );
    for ( int i = 0 ; i < n ; i++ ) {
        map_of_stb_location[i] = i;
    }
    //=======================
    //_______________________
    //only gaus
    for ( int i = 0 ; i < n ; i++ ) { //string iterator
        //_______________________
        //find max element
        int num_of_max = i; //num of max stb
        long double max = fabsl( A[i][i] );
        for ( int j = i + 1 ; j < n ; j++ ) { //find max elem in string
            if ( fabsl( A[i][j] ) > max ) {
                max = fabsl( A[i][j] );
                num_of_max = j;
            }
        }
```

```
        if ( i != num_of_max ) { //if not zero elem in set position
            swap_stb( A , i , num_of_max , n );
            int dop = map_of_stb_location[i];
            map_of_stb_location[i] = map_of_stb_location[num_of_max];
            map_of_stb_location[num_of_max] = dop;
        }
        //======================
        //______________________
        //devide string
        for ( int j = i + 1 ; j < n ; j++ ) { //stb iterator
            A[i][j] /= A[i][i]; //take main num
        }
        f[i] /= A[i][i];
        A[i][i] = 1;
        //======================
        //______________________
        //reduse matrix with round result
        for ( int k = i + 1 ; k < n ; k++ ) { //dec of the string to zero start number
            for ( int p = i + 1 ; p < n ; p++ ) {
                A[k][p] -= A[i][p] * A[k][i];
            }
            f[k] -= f[i] * A[k][i];
            A[k][i] = 0;
        }
        //======================
    }
    //======================
    //______________________
    //make finish deal
    gaus_reverse( A , f , n ); //start reverse gaus to make a solution
    long double *ask = (long double *) malloc( n * sizeof( *ask ) );
    for ( int i = 0 ; i < n ; i++ ) { //get start allocate base on start map
        ask[map_of_stb_location[i]] = f[i];
    }
    for ( int i = 0 ; i < n ; i++ ) { //copy the solution
        f[i] = ask[i];
    }
    free( ask ); //free mem
    free( map_of_stb_location ); //free mem
    //======================
    printf( "MAINELEM_GAUS WAS FINISHED\n" );
}

long double **
rev_matrix_gaus( long double **A , int n , long double *det )
{
    printf( "REV_MATRIX WAS STARTED\n" );
    *det = 1; //its only fo calc determ
```

```
long double **copy_of_A = copy_of_matrix( A , NULL , n );
//______________________
//map of stb location for allocate start position
int *map_of_stb_location = (int *) malloc( n * sizeof( *map_of_stb_location ) );
for ( int i = 0 ; i < n ; i++ ) {
    map_of_stb_location[i] = i;
}
long double **B = (long double **) calloc( n , sizeof( *B ) );
for( int i = 0 ; i < n ; i++ ) {
    B[i] = (long double *) calloc( n , sizeof( **B ) );
    B[i][i] = 1;
}
//======================
//______________________
//only gaus
for ( int i = 0 ; i < n ; i++ ) { //string iterator
    //______________________
    //find max element
    int num_of_max = i; //num of max stb
    long double max = fabsl( A[i][i] );
    for ( int j = i + 1 ; j < n ; j++ ) { //find max elem in string
        if ( fabsl( A[i][j] ) > max ) {
            max = fabsl( A[i][j] );
            num_of_max = j;
        }
    }
    if ( i != num_of_max ) { //if not zero elem in set position
        *det *= -1; //its only fo calc determ
        swap_stb( A , i , num_of_max , n );
        int dop = map_of_stb_location[i];
        map_of_stb_location[i] = map_of_stb_location[num_of_max];
        map_of_stb_location[num_of_max] = dop;
    }
    //======================
    //______________________
    //devide string
    *det *= A[i][i]; //its only fo calc determ
    for ( int j = i + 1 ; j < n ; j++ ) { //stb iterator
        A[i][j] /= A[i][i]; //take main num
    }
    for ( int j = 0 ; j < n ; j++ ) {
        B[i][j] /= A[i][i];
    }
    A[i][i] = 1;
    //======================
    //______________________
    //reduse matrix with round result
    for ( int k = i + 1 ; k < n ; k++ ) { //dec of the string to zero start number
```

```
            for ( int p = i + 1 ; p < n ; p++ ) {
                A[k][p] -= A[i][p] * A[k][i];
            }
            for ( int p = 0 ; p < n ; p++ ) {
                B[k][p] -= B[i][p] * A[k][i];
            }
            A[k][i] = 0;
        }
        //=======================
    }
    //=======================
    //________________________
    //the list to go the end reverse go
    for ( int i = n - 1 ; i >= 0 ; i-- ) { //set in the last point in matrix
        for ( int j = i - 1 ; j >= 0 ; j-- ) {
            for ( int p = 0 ; p < n ; p++ ) {
                B[j][p] -= A[j][i] * B[i][p];
            }
            A[j][i] = 0;
        }
    }
    //=======================
    //________________________
    //make finish deal
    long double **ask = (long double **) malloc( n * sizeof( *ask ) ); //swap string to male a
result
    for ( int i = 0 ; i < n ; i++ ) { //get start allocate base on start map
        ask[map_of_stb_location[i]] = B[i];
    }
    for ( int i = 0 ; i < n ; i++ ) { //copy the solution
        B[i] = ask[i];
    }
    free( ask ); //free mem
    free( map_of_stb_location ); //free mem
    //=======================
    A = copy_of_matrix( copy_of_A , A , n );
    free_mem( copy_of_A , NULL , n );
    printf( "REV_MATRIX WAS FINISHED\n" );
    return B;
}

long double
cond_num( long double **A , long double **B , int n )
{
    long double ob = 0;
    long double *mas_A = (long double *) calloc( n , sizeof( *mas_A ) );//mass of string sum in
matrix A
```

```
    long double *mas_B = (long double *) calloc( n , sizeof( *mas_B ) );//mass of string sum in
matrix B = A^-1
    //______________________
    //only find string sum of both matrix
    for( int i = 0 ; i < n ; i++ ) {
        for( int j = 0 ; j < n ; j++ ) {
            mas_A[i] += fabsl( A[i][j] );
            mas_B[i] += fabsl( B[i][j] );
        }
    }
    //======================
    //______________________
    //only find max sum of string of both matrix
    long double norma_A = mas_A[0]; //inf norma
    long double norma_B = mas_B[0]; //inf norma
    for ( int i = 1 ; i < n ; i++ ) {
        if ( mas_A[i] > norma_A ) {
            norma_A = mas_A[i];
        }
        if ( mas_B[i] > norma_B ) {
            norma_B = mas_B[i];
        }
    }
    //======================
    ob = norma_A * norma_B;
    free( mas_A );
    free( mas_B );
    return ob;
}

void
swap_line( long double **A , long double *f , int f_1 , int f_2 , int n ) //swap system line
{
    long double dop;
    for ( int p = f_1 ; p < n ; p++ ) { //swap string of matrix
        dop = A[f_1][p];
        A[f_1][p] = A[f_2][p];
        A[f_2][p] = dop;
    }
    if ( f ) {
        dop = f[f_1]; //swap string of f stb
        f[f_1] = f[f_2];
        f[f_2] = dop;
    }
}

void
swap_stb( long double **A , int f_1 , int f_2 , int n ) //swap A stb
```

```
{
   long double dop;
   for ( int p = 0 ; p < n ; p++ ) { //swap stb of matrix
      dop = A[p][f_1];
      A[p][f_1] = A[p][f_2];
      A[p][f_2] = dop;
   }
}

int
read_matrix_system( const char *file_name , long double ***MATRIX , long double **STB )
{
   //______________________
   //allocate size of matrix
   long double **A = *MATRIX;
   long double *f = *STB;
   FILE *in = fopen( file_name , "r" );
   int n;
   fscanf( in , "%d" , &n );
   printf( "SIZE :: %d\n" , n );
   //======================
   //______________________
   //get memory
   A = (long double **) malloc( n * sizeof( *A ) );
   for ( int i = 0 ; i < n ; i++ ) {
      A[i] = (long double *) malloc ( n * sizeof( **A ) );
   }
   f = (long double *) malloc ( n * sizeof( *f ) );
   //======================
   //______________________
   //read matrix from file with file_name
   for ( int i = 0 ; i < n ; i++ ) {
      for ( int j = 0 ; j < n ; j++ ) {
         fscanf( in , "%Lf" , &A[i][j] );
      }
      fscanf( in , "%Lf" , &f[i] );
   }
   fclose( in );
   //======================
   printf( "MATRIX OF SYSTEM WAS READED\n" );
   *MATRIX = A;
   *STB = f;
   return n;
}

int
read_matrix( const char *file_name , long double ***MATRIX )
{
```

```
    //_______________________
    //allocate size of matrix
    long double **A = *MATRIX;
    FILE *in = fopen( file_name , "r" );
    int n;
    fscanf( in , "%d" , &n );
    printf( "SIZE :: %d\n" , n );
    //=======================
    //_______________________
    //get memory
    A = (long double **) malloc( n * sizeof( *A ) );
    for ( int i = 0 ; i < n ; i++ ) {
        A[i] = (long double *) malloc ( n * sizeof( **A ) );
    }
    //=======================
    //_______________________
    //read matrix from file with file_name
    for ( int i = 0 ; i < n ; i++ ) {
        for ( int j = 0 ; j < n ; j++ ) {
            fscanf( in , "%Lf" , &A[i][j] );
        }
    }
    fclose( in );
    //=======================
    printf( "MATRIX WAS READED\n" );
    *MATRIX = A;
    return n;
}

void
write_matrix( const char *file_name , long double **A , long double *f , int n )
{
    //_______________________
    //write matrix in file with file_name
    FILE *out = fopen( file_name , "a" );
    fprintf( out , "NEW MATRIX\n" );
    for ( int i = 0 ; i < n ; i++ ) {
        for ( int j = 0 ; j < n ; j++ ) {
            fprintf( out , "%Lf " , A[i][j] );
        }
        if ( f ) {
            fprintf( out , "| %Lf\n" , f[i] );
        } else {
            fprintf( out , "\n" );
        }
    }
    fprintf( out , "END MATRIX\n" );
    fclose( out );
```

```
    printf( "MATRIX WRITE IN FILE : %s\n" , file_name );
    //=======================
}

void
write_solution( const char *file_name , long double *f , int n )
{
    //_______________________
    //write solution only in file
    FILE *out = fopen( file_name , "a" );
    fprintf( out , "SOLUTION\n");
    for( int i = 0 ; i < n ; i++ ) {
        fprintf( out , "x_%d = %0.15Lf\n" , i + 1 , f[i] );
    }
    fclose( out );
    //=======================
}

long double **
copy_of_matrix( long double **A , long double **B , int n ) //its only copy matrix for save from
algo
{
    //_______________________
    //copy
    if ( !B ) {
        B = (long double **) calloc( n , sizeof( *B ) );
        for( int i = 0 ; i < n ; i++ ) {
            B[i] = (long double *) calloc( n , sizeof( **B ) );
        }
    }
    for ( int i = 0 ; i < n ; i++ ) {
        for ( int j = 0 ; j < n ; j++ ) {
            B[i][j] = A[i][j];
        }
    }
    //=======================
    return B;
}

void
make_matrix( const char *file_name , int mode ) //gen matrix: mode 1 - only matrix ; 0 - full
system
{
    //_______________________
    //gen matrix
    int n = 30 , m = 20;
    FILE *out = fopen( file_name , "w" );
    fprintf( out , "%d\n" , n );
```

```
    for ( int i = 1 ; i <= n ; i++ ) {
        for ( int j = 1 ; j <= n ; j++ ) {
            if ( i != j ) {
                fprintf( out , "%Lf " , (long double) (i + j) / (long double) (n + m) );
            } else {
                fprintf( out , "%Lf " , (long double) n + ( m * m ) + ( (long double) j / m ) + ( (long
double) i / n ) );
            }
        }
        if ( mode == 0 ) {
            fprintf( out , "%Lf\n" , (long double) m * i + n );
        } else {
            fprintf( out , "\n" );
        }
    }
    fclose( out );
    //=======================
}

void
free_mem( long double **A , long double *f , int n )
{
    //_______________________
    //free memory after use matrix
    for( int i = 0 ; i < n ; i++ ) {
        free( A[i] );
    }
    free( A );
    free( f );
    //=======================
}
```

Test 1

4
2 2 -1 1 4
4 3 -1 2 6
8 5 -3 4 12
3 3 -2 4 6

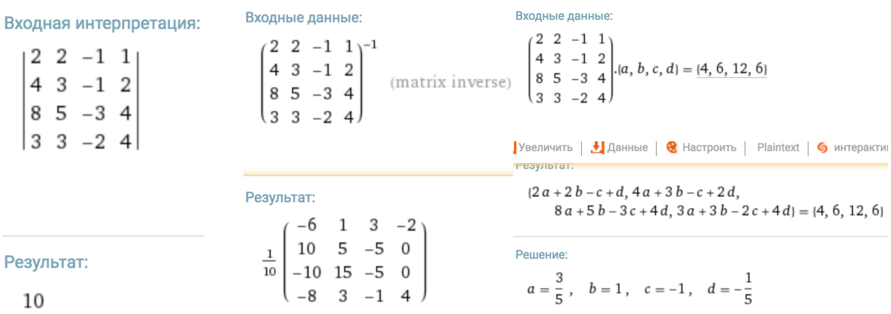
SOLUTION 1
x_1 = 0.600000000000000
x_2 = 1.000000000000000
x_3 = -1.000000000000000
x_4 = -0.200000000000000

SOLUTION 2
x_1 = 0.600000000000000
x_2 = 1.000000000000000
x_3 = -1.000000000000000
x_4 = -0.200000000000000

NEW MATRIX
-0.600000 0.100000 0.300000 -0.200000
1.000000 0.500000 -0.500000 0.000000
-1.000000 1.500000 -0.500000 0.000000
-0.800000 0.300000 -0.100000 0.400000
END MATRIX

Cond: 60.000000
Det: 10.000000

*{{ 2 , 2 , -1 , 1 },{ 4 , 3 , -1 , 2 },{ 8 , 5 , -3 , 4 },{ 3 , 3 , -2 , 4 }} * { a , b , c , d } = { 4 , 6 , 12 , 6 }*



Test 2

4
2 5 -8 3 8
4 3 -9 1 9
2 3 -5 -6 7
1 8 -7 0 12

SOLUTION 1
x_1 = 3.000000000000000
x_2 = 2.000000000000000
x_3 = 1.000000000000000
x_4 = 0.000000000000000

SOLUTION 2
x_1 = 3.000000000000000
x_2 = 2.000000000000000
x_3 = 1.000000000000000
x_4 = 0.000000000000000

NEW MATRIX
-1.719577 1.222222 -0.656085 0.862434
-0.650794 0.333333 -0.269841 0.507937
-0.989418 0.555556 -0.402116 0.560847
-0.074074 0.111111 -0.185185 0.074074
END MATRIX

Cond: 80.285714
Det: -189.000000

*{{ 2 , 5 , -8 , 3 },{ 4 , 3 , -9 , 1 },{ 2 , 3 , -5 , -6 },{ 1 , 8 , -7 , 0 }} * { a , b , c , d } = { 8 , 9 , 7 , 12 }*

Входная интерпретация:

$$\begin{vmatrix} 2 & 5 & -8 & 3 \\ 4 & 3 & -9 & 1 \\ 2 & 3 & -5 & -6 \\ 1 & 8 & -7 & 0 \end{vmatrix}$$

Увеличить | Данные |

Результат:

-189

$$\begin{pmatrix} 2 & 5 & -8 & 3 \\ 4 & 3 & -9 & 1 \\ 2 & 3 & -5 & -6 \\ 1 & 8 & -7 & 0 \end{pmatrix}^{-1} \text{(matrix inverse)}$$

Результат:

$$\frac{1}{189}\begin{pmatrix} -325 & 231 & -124 & 163 \\ -123 & 63 & -51 & 96 \\ -187 & 105 & -76 & 106 \\ -14 & 21 & -35 & 14 \end{pmatrix}$$

Входные данные:

$$\begin{pmatrix} 2 & 5 & -8 & 3 \\ 4 & 3 & -9 & 1 \\ 2 & 3 & -5 & -6 \\ 1 & 8 & -7 & 0 \end{pmatrix} . \{a, b, c, d\} = \{8, 9, 7, 12\}$$

Результат:

$$\{2a + 5b - 8c + 3d,\ 4a + 3b - 9c + d,\ 2a + 3b - 5c - 6d,\ a + 8b - 7c\} = \{8, 9, 7, 12\}$$

Решение:

$a = 3,\quad b = 2,\quad c = 1,\quad d = 0$

Test 3

3
2 -1 0 3
-1 2 -1 4
0 -1 2 5

SOLUTION 1
x_1 = 5.500000000000000
x_2 = 8.000000000000000
x_3 = 6.500000000000000

SOLUTION 2
x_1 = 5.500000000000000
x_2 = 8.000000000000000
x_3 = 6.500000000000000

NEW MATRIX
0.750000 0.500000 0.250000
0.500000 1.000000 0.500000
0.250000 0.500000 0.750000
END MATRIX

Cond: 8.000000
Det: 4.000000

*{{ 2 , -1 , 0 },{ -1 , 2 , -1 },{ 0 , -1 , 2 }} * { a , b , c } = { 3 , 4 , 5 }*

Входная интерпретация:

$$\begin{vmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{vmatrix}$$

Результат:

4

Входные данные:

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}^{-1} \text{(matrix inverse)}$$

Результат:

$$\frac{1}{4}\begin{pmatrix} 3 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 3 \end{pmatrix}$$

Входные данные:

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}.\{a, b, c\} = \{3, 4, 5\}$$

Увеличить | Данные | Настроить | Pla

Результат:

$$\{2a - b, -a + 2b - c, 2c - b\} = \{3, 4, 5\}$$

Решение:

$$a = \frac{11}{2}, \quad b = 8, \quad c = \frac{13}{2}$$

Test 4

$$A_{ij} = \frac{i+j}{m+n}, i \neq j$$

$$A_{ij} = n + m^2 + \frac{j}{m} + \frac{i}{n}, i = j \; \forall i, j = 1...n$$

$$b_i = 20 \cdot i + 30$$

SOLUTION 1
x_1 = 0.094220856385308
x_2 = 0.139664640363886
x_3 = 0.185099268187903
x_4 = 0.230524742405332
x_5 = 0.275941065140727
x_6 = 0.321348240336259
x_7 = 0.366746270219767
x_8 = 0.412135156596181
x_9 = 0.457514904037024
x_10 = 0.502885514450388
x_11 = 0.548246989321968
x_12 = 0.593599333851879
x_13 = 0.638942549628852
x_14 = 0.684276637819734
x_15 = 0.729601604252472
x_16 = 0.774917450196826
x_17 = 0.820224176501179
x_18 = 0.865521789620556
x_19 = 0.910810290506128
x_20 = 0.956089679688203
x_21 = 1.001359964248124
x_22 = 1.046621144818862
x_23 = 1.091873221613033
x_24 = 1.137116202337544
x_25 = 1.182350087307552
x_26 = 1.227574876418363
x_27 = 1.272790578001696
x_28 = 1.317997192055276
x_29 = 1.363194718157486
x_30 = 1.408383165264102

SOLUTION 2
x_1 = 0.094220856385308
x_2 = 0.139664640363886
x_3 = 0.185099268187903
x_4 = 0.230524742405332
x_5 = 0.275941065140727

x_6 = 0.321348240336259
x_7 = 0.366746270219767
x_8 = 0.412135156596181
x_9 = 0.457514904037024
x_10 = 0.502885514450388
x_11 = 0.548246989321968
x_12 = 0.593599333851879
x_13 = 0.638942549628852
x_14 = 0.684276637819734
x_15 = 0.729601604252472
x_16 = 0.774917450196826
x_17 = 0.820224176501179
x_18 = 0.865521789620556
x_19 = 0.910810290506128
x_20 = 0.956089679688203
x_21 = 1.001359964248124
x_22 = 1.046621144818862
x_23 = 1.091873221613033
x_24 = 1.137116202337544
x_25 = 1.182350087307552
x_26 = 1.227574876418363
x_27 = 1.272790578001696
x_28 = 1.317997192055276
x_29 = 1.363194718157486
x_30 = 1.408383165264102

Cond: 1.122125
Det:
11032591757245778721867315865928395682697796809573674337936498887558459459371008.000000

Приложение 3

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <math.h>

long double w = 0.8;

void        relax_method( long double **A , long double *f , long double eps , int n );
void        one_iteraton( long double **A , long double *f , long double *x_k , long double
*x_k_1 , int n );
long double    accuracy_of_solve_KOSHI( long double *x_k , long double *x_k_1 , int n );
long double    accuracy_of_solve_MATRIX( long double **A , long double *f , long double *x_k
, int n );
int         read_matrix_system( const char *file_name , long double ***MATRIX , long double
**STB );
void        write_matrix( const char *file_name , long double **A , long double *f , int n );
void        write_solution( const char *file_name , long double *f , int n );
void        make_matrix( const char *file_name , int mode );
void        free_mem( long double **A , long double *f , int n );

int
main( int argc , char **argv )
{
    //________________________
    //its only for use this packet of meth
    if ( argc == 2 ) {
        printf( "arg1 : input file\n" );
        printf( "arg2 : output file\n" );
        printf( "arg3 : accuracy\n" );
        printf( "arg4 : matrix in file <1> , matrix by fomula <2>\n" );
        return 0;
    }
    long double eps;
    sscanf( argv[3] , "%Lf" , &eps );
    int task;
    sscanf( argv[4] , "%d" , &task );
    unlink( argv[2] );
    if ( task == 1 ) {
        long double **A = NULL , *f = NULL;
        int size = read_matrix_system( argv[1] , &A , &f );
        relax_method( A , f , eps , size );
        write_solution( argv[2] , f , size );
        free_mem( A , f , size );
        return 0;
    }
```

```
    if ( task == 2 ) {
        long double **A = NULL , *f = NULL;
        make_matrix( argv[1] , 0 );
        int size = read_matrix_system( argv[1] , &A , &f );
        relax_method( A , f , eps , size );
        write_solution( argv[2] , f , size );
        free_mem( A , f , size );
        return 0;
    }
    if ( task == 3 ) {
        long double **A = NULL , *f = NULL;
        w = 0.05;
        for ( ; w < 1.97 ; w += 0.05 ) {
            int size = read_matrix_system( argv[1] , &A , &f );
            printf( "W = %Lf " , w );
            relax_method( A , f , eps , size );
            free_mem( A , f , size );
            A = NULL ; f = NULL;
        }
    }
    return 0;
}

void
relax_method( long double **A , long double *f , long double eps , int n )
{
    long double *x_k = (long double *) malloc ( n * sizeof( *x_k ) );
    long double *x_k_1 = (long double *) malloc ( n * sizeof( *x_k_1 ) );
    for ( int i = 0 ; i < n ; i++ ) { //initialized start value
        x_k[i] = 1;
    }
    //______________________
    //make next iteration of algo
    int k = 0;
    while ( 1 ) {
        if ( accuracy_of_solve_MATRIX( A , f , x_k , n ) < eps ) { // accuracy_of_solve_KOSHI( x_k
, x_k_1 , n ) < eps
            break;
        }
        one_iteraton( A , f , x_k , x_k_1 , n );
        k++;
    }
    //======================
    //______________________
    //copy of solution in f
    for ( int i = 0 ; i < n ; i++ ) {
        f[i] = x_k[i];
    }
```

```
    //=======================
    printf( "COUNT_IT :: %d\n" , k );
    free( x_k );
    free( x_k_1 );
}

void
one_iteraton( long double **A , long double *f , long double *x_k , long double *x_k_1 , int n )
{
    //_______________________
    //iteration
    long double q = 1 - ( 1 / w ); //its koef of k_i
    for ( int i = 0 ; i < n ; i++ ) {
        x_k_1[i] = f[i];
        for ( int j = 0 ; j < i ; j++ ) {
            x_k_1[i] -= A[i][j] * x_k_1[j];
        }
        x_k_1[i] -= q * A[i][i] * x_k[i];
        for ( int j = i + 1 ; j < n ; j++ ) {
            x_k_1[i] -= A[i][j] * x_k[j];
        }
        x_k_1[i] *= w;
        x_k_1[i] /= A[i][i];
    }
    for ( int i = 0 ; i < n ; i++ ) {
        long double dop;
        dop = x_k[i];
        x_k[i] = x_k_1[i];
        x_k_1[i] = dop;
    }
    //really now i+i is x_k and i is x_k_1
    //=======================
}

long double
accuracy_of_solve_KOSHI( long double *x_k , long double *x_k_1 , int n )
{
    //_______________________
    //calc norma of solve
    long double norma = 0;
    for ( int i = 0 ; i < n ; i++ ) {
        norma += fabsl( x_k[i] - x_k_1[i] );
    }
    //=======================
    return norma;
}

long double
```

```
accuracy_of_solve_MATRIX( long double **A , long double *f , long double *x_k , int n )
{
   //______________________
   //calc norma of solve
   long double *Ax = (long double *) calloc( n , sizeof( *Ax ) );
   for ( int i = 0 ; i < n ; i++ ) {
      for ( int j = 0 ; j < n ; j++ ) {
         Ax[i] += A[i][j] * x_k[j];
      }
   }
   long double norma = 0;
   for ( int i = 0 ; i < n ; i++ ) {
      norma += fabsl( f[i] - Ax[i] );
   }
   free( Ax );
   //======================
   return norma;
}

int
read_matrix_system( const char *file_name , long double ***MATRIX , long double **STB )
{
   //______________________
   //allocate size of matrix
   long double **A = *MATRIX;
   long double *f = *STB;
   FILE *in = fopen( file_name , "r" );
   int n;
   fscanf( in , "%d" , &n );
   // printf( "SIZE :: %d\n" , n );
   //======================
   //______________________
   //get memory
   A = (long double **) malloc( n * sizeof( *A ) );
   for ( int i = 0 ; i < n ; i++ ) {
      A[i] = (long double *) malloc ( n * sizeof( **A ) );
   }
   f = (long double *) malloc ( n * sizeof( *f ) );
   //======================
   //______________________
   //read matrix from file with file_name
   for ( int i = 0 ; i < n ; i++ ) {
      for ( int j = 0 ; j < n ; j++ ) {
         fscanf( in , "%Lf" , &A[i][j] );
      }
      fscanf( in , "%Lf" , &f[i] );
   }
   fclose( in );
```

```
    //=======================
    // printf( "MATRIX OF SYSTEM WAS READED\n" );
    *MATRIX = A;
    *STB = f;
    return n;
}

void
write_matrix( const char *file_name , long double **A , long double *f , int n )
{
    //_______________________
    //write matrix in file with file_name
    FILE *out = fopen( file_name , "a" );
    fprintf( out , "NEW MATRIX\n" );
    for ( int i = 0 ; i < n ; i++ ) {
        for ( int j = 0 ; j < n ; j++ ) {
            fprintf( out , "%Lf " , A[i][j] );
        }
        if ( f ) {
            fprintf( out , "| %Lf\n" , f[i] );
        } else {
            fprintf( out , "\n" );
        }
    }
    fprintf( out , "END MATRIX\n" );
    fclose( out );
    printf( "MATRIX WRITE IN FILE : %s\n" , file_name );
    //=======================
}

void
write_solution( const char *file_name , long double *f , int n )
{
    //_______________________
    //write solution only in file
    FILE *out = fopen( file_name , "a" );
    fprintf( out , "SOLUTION\n");
    for( int i = 0 ; i < n ; i++ ) {
        fprintf( out , "x_%d = %0.15Lf\n" , i + 1 , f[i] );
    }
    fclose( out );
    //=======================
}

void
make_matrix( const char *file_name , int mode ) //gen matrix: mode 1 - only matrix ; 0 - full
system
{
```

```
    //________________________
    //gen matrix
    int n = 30 , m = 20;
    FILE *out = fopen( file_name , "w" );
    fprintf( out , "%d\n" , n );
    for ( int i = 1 ; i <= n ; i++ ) {
        for ( int j = 1 ; j <= n ; j++ ) {
            if ( i != j ) {
                fprintf( out , "%Lf " , (long double) (i + j) / (long double) (n + m) );
            } else {
                fprintf( out , "%Lf " , (long double) n + ( m * m ) + ( (long double) j / m ) + ( (long
double) i / n ) );
            }
        }
        if ( mode == 0 ) {
            fprintf( out , "%Lf\n" , (long double) m * i + n );
        } else {
            fprintf( out , "\n" );
        }
    }
    fclose( out );
    //======================
}

void
free_mem( long double **A , long double *f , int n )
{
    //________________________
    //free memory after use matrix
    for( int i = 0 ; i < n ; i++ ) {
        free( A[i] );
    }
    free( A );
    free( f );
    //======================
}
```

К сожалению матрицы из варианта не подходят для тестирования данного алгоритма, поэтому были сгенерированы другие матрицы.

Test 1

3
2 -1 0 3
-1 2 -1 4
0 -1 2 5

SOLUTION
x_1 = 5.499999999958773
x_2 = 7.999999999952658
x_3 = 6.499999999972818

EPS = 0.0000000001

W = 0.050000 COUNT_IT :: 1686
W = 0.100000 COUNT_IT :: 822
W = 0.150000 COUNT_IT :: 533
W = 0.200000 COUNT_IT :: 389
W = 0.250000 COUNT_IT :: 302
W = 0.300000 COUNT_IT :: 245
W = 0.350000 COUNT_IT :: 203
W = 0.400000 COUNT_IT :: 172
W = 0.450000 COUNT_IT :: 148
W = 0.500000 COUNT_IT :: 128
W = 0.550000 COUNT_IT :: 112
W = 0.600000 COUNT_IT :: 99
W = 0.650000 COUNT_IT :: 88
W = 0.700000 COUNT_IT :: 78
W = 0.750000 COUNT_IT :: 69
W = 0.800000 COUNT_IT :: 62
W = 0.850000 COUNT_IT :: 55
W = 0.900000 COUNT_IT :: 49
W = 0.950000 COUNT_IT :: 43
W = 1.000000 COUNT_IT :: 37
W = 1.050000 COUNT_IT :: 32
W = 1.100000 COUNT_IT :: 27
W = 1.150000 COUNT_IT :: 22
W = 1.200000 COUNT_IT :: 17
W = 1.250000 COUNT_IT :: 20
W = 1.300000 COUNT_IT :: 22
W = 1.350000 COUNT_IT :: 25

W = 1.400000 COUNT_IT :: 28
W = 1.450000 COUNT_IT :: 33
W = 1.500000 COUNT_IT :: 37
W = 1.550000 COUNT_IT :: 44
W = 1.600000 COUNT_IT :: 50
W = 1.650000 COUNT_IT :: 61
W = 1.700000 COUNT_IT :: 74
W = 1.750000 COUNT_IT :: 90
W = 1.800000 COUNT_IT :: 115
W = 1.850000 COUNT_IT :: 159
W = 1.900000 COUNT_IT :: 243
W = 1.950000 COUNT_IT :: 496

*{{ 2 , -1 , 0 },{ -1 , 2 , -1 },{ 0 , -1 , 2 }} * { a , b , c } = { 3 , 4 , 5 }*

Test 2

3
21 41 50 1
41 546 346 2
50 346 349 3

SOLUTION
x_1 = 0.039263343491183
x_2 = -0.003141863684024
x_3 = 0.006085723954520

EPS = 0.0000000001

W = 0.050000 COUNT_IT :: 2731
W = 0.100000 COUNT_IT :: 1331
W = 0.150000 COUNT_IT :: 905
W = 0.200000 COUNT_IT :: 675
W = 0.250000 COUNT_IT :: 532
W = 0.300000 COUNT_IT :: 435
W = 0.350000 COUNT_IT :: 365
W = 0.400000 COUNT_IT :: 312
W = 0.450000 COUNT_IT :: 270
W = 0.500000 COUNT_IT :: 236
W = 0.550000 COUNT_IT :: 208
W = 0.600000 COUNT_IT :: 184
W = 0.650000 COUNT_IT :: 164
W = 0.700000 COUNT_IT :: 146
W = 0.750000 COUNT_IT :: 131
W = 0.800000 COUNT_IT :: 117
W = 0.850000 COUNT_IT :: 105

W = 0.900000 COUNT_IT :: 94
W = 0.950000 COUNT_IT :: 83
W = 1.000000 COUNT_IT :: 74
W = 1.050000 COUNT_IT :: 65
W = 1.100000 COUNT_IT :: 56
W = 1.150000 COUNT_IT :: 47
W = 1.200000 COUNT_IT :: 34
W = 1.250000 COUNT_IT :: 35
W = 1.300000 COUNT_IT :: 38
W = 1.350000 COUNT_IT :: 41
W = 1.400000 COUNT_IT :: 46
W = 1.450000 COUNT_IT :: 50
W = 1.500000 COUNT_IT :: 58
W = 1.550000 COUNT_IT :: 66
W = 1.600000 COUNT_IT :: 78
W = 1.650000 COUNT_IT :: 92
W = 1.700000 COUNT_IT :: 110
W = 1.750000 COUNT_IT :: 134
W = 1.800000 COUNT_IT :: 173
W = 1.850000 COUNT_IT :: 234
W = 1.900000 COUNT_IT :: 363
W = 1.950000 COUNT_IT :: 744

*{{ 21 , 41 , 50 },{ 41 , 546 , 346 },{ 50 , 346 , 349 }} * { a , b , c } = { 1 , 2 , 3 }*

Test 3

3
15442 4971 218 4
4971 3325 -249 44
218 -249 83 7
(8)
SOLUTION
x_1 = -0.060171272229531
x_2 = 0.156502484984923
x_3 = 0.711885013340776

EPS = 0.0000000001

W = 0.050000 COUNT_IT :: 8913
W = 0.100000 COUNT_IT :: 4335
W = 0.150000 COUNT_IT :: 2809
W = 0.200000 COUNT_IT :: 2045
W = 0.250000 COUNT_IT :: 1587
W = 0.300000 COUNT_IT :: 1281
W = 0.350000 COUNT_IT :: 1061

W = 0.400000 COUNT_IT :: 896
W = 0.450000 COUNT_IT :: 767
W = 0.500000 COUNT_IT :: 663
W = 0.550000 COUNT_IT :: 576
W = 0.600000 COUNT_IT :: 500
W = 0.650000 COUNT_IT :: 426
W = 0.700000 COUNT_IT :: 386
W = 0.750000 COUNT_IT :: 360
W = 0.800000 COUNT_IT :: 330
W = 0.850000 COUNT_IT :: 301
W = 0.900000 COUNT_IT :: 274
W = 0.950000 COUNT_IT :: 249
W = 1.000000 COUNT_IT :: 226
W = 1.050000 COUNT_IT :: 205
W = 1.100000 COUNT_IT :: 185
W = 1.150000 COUNT_IT :: 167
W = 1.200000 COUNT_IT :: 150
W = 1.250000 COUNT_IT :: 133
W = 1.300000 COUNT_IT :: 118
W = 1.350000 COUNT_IT :: 102
W = 1.400000 COUNT_IT :: 87
W = 1.450000 COUNT_IT :: 71
W = 1.500000 COUNT_IT :: 56
W = 1.550000 COUNT_IT :: 65
W = 1.600000 COUNT_IT :: 75
W = 1.650000 COUNT_IT :: 89
W = 1.700000 COUNT_IT :: 107
W = 1.750000 COUNT_IT :: 133
W = 1.800000 COUNT_IT :: 171
W = 1.850000 COUNT_IT :: 235
W = 1.900000 COUNT_IT :: 362
W = 1.950000 COUNT_IT :: 743

*{{ 15442 , 4971 , 218 },{ 4971 , 3325 , -249 },{ 218 , -249 , 83 }} * { a , b , c } = { 4 , 44 , 7 }*

Test 4

$$A_{ij} = \frac{i+j}{m+n}, i \neq j$$
$$A_{ij} = n + m^2 + \frac{j}{m} + \frac{i}{n}, i = j \, \forall i, j = 1...n$$
$$b_i = 20 \cdot i + 30$$

SOLUTION
x_1 = 0.094220856385316

x_2 = 0.139664640363894
x_3 = 0.185099268187910
x_4 = 0.230524742405339
x_5 = 0.275941065140733
x_6 = 0.321348240336265
x_7 = 0.366746270219772
x_8 = 0.412135156596186
x_9 = 0.457514904037029
x_10 = 0.502885514450393
x_11 = 0.548246989321973
x_12 = 0.593599333851883
x_13 = 0.638942549628855
x_14 = 0.684276637819737
x_15 = 0.729601604252475
x_16 = 0.774917450196829
x_17 = 0.820224176501182
x_18 = 0.865521789620558
x_19 = 0.910810290506130
x_20 = 0.956089679688205
x_21 = 1.001359964248125
x_22 = 1.046621144818863
x_23 = 1.091873221613034
x_24 = 1.137116202337544
x_25 = 1.182350087307551
x_26 = 1.227574876418362
x_27 = 1.272790578001693
x_28 = 1.317997192055272
x_29 = 1.363194718157481
x_30 = 1.408383165264096

EPS = 0.0000000001

W = 0.050000 COUNT_IT :: 617
W = 0.100000 COUNT_IT :: 301
W = 0.150000 COUNT_IT :: 195
W = 0.200000 COUNT_IT :: 142
W = 0.250000 COUNT_IT :: 110
W = 0.300000 COUNT_IT :: 89
W = 0.350000 COUNT_IT :: 74
W = 0.400000 COUNT_IT :: 62
W = 0.450000 COUNT_IT :: 53
W = 0.500000 COUNT_IT :: 46
W = 0.550000 COUNT_IT :: 40
W = 0.600000 COUNT_IT :: 35
W = 0.650000 COUNT_IT :: 31

W = 0.700000 COUNT_IT :: 27
W = 0.750000 COUNT_IT :: 23
W = 0.800000 COUNT_IT :: 20
W = 0.850000 COUNT_IT :: 17
W = 0.900000 COUNT_IT :: 14
W = 0.950000 COUNT_IT :: 11
W = 1.000000 COUNT_IT :: 7
W = 1.050000 COUNT_IT :: 12
W = 1.100000 COUNT_IT :: 15
W = 1.150000 COUNT_IT :: 18
W = 1.200000 COUNT_IT :: 21
W = 1.250000 COUNT_IT :: 24
W = 1.300000 COUNT_IT :: 27
W = 1.350000 COUNT_IT :: 31
W = 1.400000 COUNT_IT :: 35
W = 1.450000 COUNT_IT :: 41
W = 1.500000 COUNT_IT :: 47
W = 1.550000 COUNT_IT :: 54
W = 1.600000 COUNT_IT :: 63
W = 1.650000 COUNT_IT :: 74
W = 1.700000 COUNT_IT :: 90
W = 1.750000 COUNT_IT :: 111
W = 1.800000 COUNT_IT :: 144
W = 1.850000 COUNT_IT :: 197
W = 1.900000 COUNT_IT :: 305
W = 1.950000 COUNT_IT :: 626