Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра Информатики

Дисциплина «Методы численного анализа»

**ОТЧЕТ**

к лабораторной работе №2

на тему:

**«ЧИСЛЕННОЕ РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ПРОСТЫХ ИТЕРАЦИЙ И МЕТОДОМ ЗЕЙДЕЛЯ»**

БГУИР 6-05-0612-02 005

|  |
| --- |
| Выполнила студент группы 353504  АНТОНОВА Лидия Сергеевна |
|  |
| (дата, подпись студента) |
| Проверил доцент каф. Информатики  АНИСИМОВ Владимир Яковлевич |
|  |
| (дата, подпись преподавателя) |

Минск 2024

# 1 ЦЕЛЬ ВЫПОЛНЕНИЯ ЗАДАНИЯ

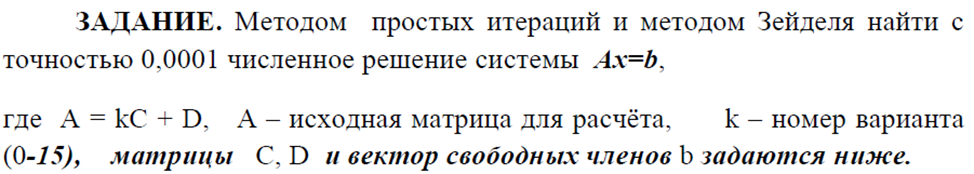
1 Изучить итерационные методы решения СЛАУ (метод простых итераций, метод Зейделя);

2 Составить алгоритм решения СЛАУ указанными методами, применимый для организации вычислений на ЭВМ;

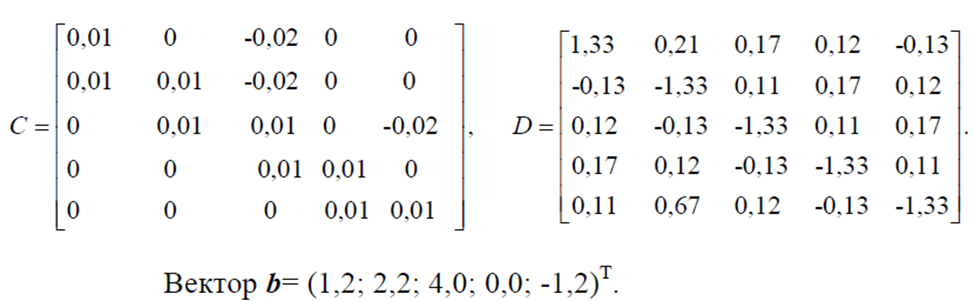
3 Составить программу решения СЛАУ по разработанному алгоритму;

4 Численно решить тестовые примеры и проверить правильность работы программы. Сравнить трудоемкость решения методом простых итераций и методом Зейделя.   
  
 **2 задание**

Вариант 1.

  
 Также необходимо найти абсолютную погрешность каждого из методов вычисления СЛАУ.

Исходные данные:

****

# 3 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

*Итерационные методы* основаны на построении сходящейся к точному решению х рекуррентной последовательности.

Для решения СЛАУ **методом простых итераций** преобразуем систему от первоначальной формы **Ax = b** или

к виду

x = Bx + C.

(2.2)

Здесь ***B*** - квадратная матрица с элементами bij (*i, j* = 1, 2. ..., *n*), ***с*** – вектор столбец с элементами сi (*i =* 1,2,..., *n*).

В развернутой форме записи система (2.2) имеет следующий вид:

Изображение выглядит как текст, Шрифт, снимок экрана, рукописный текст

Автоматически созданное описание

Вообще говоря, операция *приведения системы* к виду, *удобному* *для итераций*, не является простой и требует специальных знаний, а также существенного использования специфики системы.

Можно, например, преобразовать систему (2.1) следующим образом

Изображение выглядит как текст, Шрифт, рукописный текст, белый

Автоматически созданное описание

если диагональные элементы матрицы ***А*** отличны от нуля.

Можно преобразовать систему (2.1) в эквивалентную ей систему

x = (E – A)x + b.

Задав произвольным образом столбец начальных приближений x0 = (x10, x20, ... . xn0 )Т . подставим их в правые части системы (2.2) и вычислим новые приближения x1 = (x11, x21, ... . xn1), которые опять подставим в систему (2.2) и т.д. Таким образом, организуется итерационный процесс xk = Bxk-1 + c, k = 1,2 ..., . Известно, что система (2.1) имеет единственное решение x\* и последовательность {xk} сходится к этому решению со скоростью геометрической прогрессии, если || В || < 1 в любой матричной норме.

1. ) < 1;
2. ) < 1;

**Метод Зейделя.** Метод Зейделя является модификацией метода простых итераций. Суть его состоит в том, что при вычислении следующего хik : 2 в формуле xk = Bxk-1 + c , k =1,2, ... используются вместо х1k-1 ,…, уже вычисленные ранее хik,..., т.e.

Изображение выглядит как Шрифт, число, линия, текст

Автоматически созданное описание

(2.3)

Такое усовершенствование позволяет ускорить сходимость итераций почти в два раза. Кроме того, данный метод может быть реализован на ЭВМ без привлечения дополнительного массива, т.к. полученное новое сразу засыпается на место старого.

Схема алгоритма аналогична схеме метода простых итераций.

Самый простой способ приведения системы к виду, удобному для итераций, состоит в следующем. Из первого уравнения системы выразим неизвестное х1:

Изображение выглядит как текст, Шрифт, снимок экрана, белый

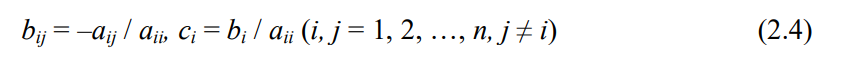
Автоматически созданное описание

Изображение выглядит как текст, Шрифт, белый, снимок экрана

Автоматически созданное описание

в которой на главной диагонали матрицы ***B*** находятся нулевые элементы.

Остальные элементы выражаются по формулам



Конечно, для возможности выполнения указанного преобразования необходимо, чтобы диагональные элементы матрицы ***А*** были ненулевыми.

Введем нижнюю ***В1*** (получается из ***В*** заменой нулями элементов стоявших на главной диагонали и выше ее) и верхнюю ***В2*** (получается из ***В*** заменой нулями элементов стоявших на главной диагонали и ниже ее) треугольные матрицы.

Заметим, что ***B = B1 + B2*** и поэтому решение ***х*** исходной системы удовлетворяет равенству

x = B1x + B2x + c

(2.5)

Выберем начальное приближение x(0) = [x1(0), x2(0), … , xn(0)]T. Подставляя его в правую часть равенства при верхней треугольной матрице ***B2*** и вычисляя полученное выражение, находим первое приближение

x(1) = B1x(0) + B2x(1)

(2.6)

Подставляя приближение **x(**1), получим

x(2) = B1x(1) + B2x(2)

(2.7)

Продолжая этот процесс далее, получим последовательность x(0) ,x(1), … , x(n) , … приближений к вычисляемых по формуле

x(k+1) = B1x(k) + B2x(k+1) + c

(2.8)

или в развернутой форме записи



Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

Объединив приведение системы к виду, удобному для итераций и метод Зейделя в одну формулу, получим



Тогда достаточным условием сходимости метода Зейделя будет условие *доминирования диагональных элементов в строках или столбцах матрицы*

***A, т.e.***

Изображение выглядит как текст, Шрифт, белый, рукописный текст

Автоматически созданное описание

Методы простой итерации и Зейделя сходятся примерно так же, как геометрическая прогрессия со знаменателем || ***В*** || .

# 4 Выполнение работы

При написании задания данной лабораторной работы использовалась система компьютерной алгебры Maple 2021. Для решения СЛАУ необходимо было использовать метод простых итераций и метод Зейделя.

Для нахождения матрицы A была реализована функция:

findA := proc(c::Matrix, d::Matrix, k)

return k\*c + d;

end proc;

Эта функция принимает две матрицы и коэффициент k, и возвращает результирующую матрицу, полученную как линейную комбинацию.

Также была реализована функция Solves, которая проверяет существование и единственность решения системы. Если количество строк матрицы A не совпадает с количеством элементов вектора B, функция генерирует ошибку с сообщением:

Solves := proc(A::Matrix, B::Vector)

local rank\_A, rank\_AB, row\_dim;

rank\_A := Rank(A);

rank\_AB := Rank(<A | B>);

row\_dim := RowDimension(A);

if row\_dim < rank\_AB then

error "No solution: the system is inconsistent.";

elif rank\_A < row\_dim and rank\_AB = rank\_A then

error "Infinitely many solutions: the system is underdetermined.";

elif rank\_AB = row\_dim then

print("Unique solution: the system has a single solution.");

end if;

end proc;

Для решения СЛАУ был реализован метод простых итераций. Метод включает в себя проверку на диагональное доминирование, что является необходимым условием для сходимости. В случае, если диагональный элемент равен нулю, строки матрицы могут быть переставлены.

simpleIterations := proc(A::Matrix, B::Vector, tolerance::float, max\_iterations::integer)

local size, prev\_X, current\_X, stoper, i, j, k, max\_error, error\_, iterations, swapped, temp, sum;

size := RowDimension(A);

prev\_X := Vector(1 .. size, 0.);

current\_X := Vector(1 .. size);

stoper := false;

iterations := 0;

for i to size do

if A[i, i] = 0 then

swapped := false;

for k from i + 1 to size do

if A[k, i] <> 0 then

temp := Row(A, i);

A[i, 1 .. size] := Row(A, k);

A[k, 1 .. size] := temp;

B[i] := B[k];

B[k] := B[i];

swapped := true;

break;

end if;

end do;

if not swapped then

print("Warning: Unable to swap rows to avoid zero on the diagonal at row ", i);

return NULL;

end if;

end if;

end do;

for i to size do

sum := 0;

for j to size do

if j <> i then

sum := sum + abs(A[i, j]);

end if;

end do;

if abs(A[i, i]) < sum then

print("Warning: The matrix is not diagonally dominant at row ", i);

return NULL;

end if;

end do;

while not stoper and iterations < max\_iterations do

iterations := iterations + 1;

for i to size do

current\_X[i] := B[i];

for j to size do

if j <> i then

current\_X[i] := -prev\_X[j]\*A[i, j] + current\_X[i];

end if;

end do;

if A[i, i] <> 0 then

current\_X[i] := current\_X[i]/A[i, i];

else

print("Error: Division by zero at row ", i);

return NULL;

end if;

end do;

max\_error := 0.;

for i to size do

error\_ := abs(((A[i]) . current\_X) - B[i]);

max\_error := max(max\_error, error\_);

end do;

if max\_error < tolerance then

stoper := true;

end if;

prev\_X := current\_X;

end do;

if iterations = max\_iterations then

print("Maximum iterations reached without convergence.");

end if;

print("Maximum absolute errors in SimpleIterationsMethod = ", max\_error);

return current\_X;

end proc;

Реализуем метод Гаусса с выбором главного элемента по столбцу. Он практически идентичен классического методу Гаусса, отличием является нахождение строки с большим по модулю главным элементом и заменой первой строки на эту.

MyGaussianEliminationWithPivoting := proc(A::Matrix, B::Vector)

local n, i, j, k, q, x, maxIndex, maxValue, temp;

n := RowDimension(A);

for k to n - 1 do

maxValue := abs(A[k, k]);

maxIndex := k;

for i from k + 1 to n do

if maxValue < abs(A[i, k]) then

maxValue := abs(A[i, k]);

maxIndex := i;

end if;

end do;

if maxValue = 0 then

error "Невозможно продолжать: все элементы в столбце равны нулю.";

end if;

if maxIndex <> k then

temp := Row(A, k);

Row(A, k) := Row(A, maxIndex);

Row(A, maxIndex) := temp;

temp := B[k];

B[k] := B[maxIndex];

B[maxIndex] := temp;

end if;

for i from k + 1 to n do

q := A[i, k]/A[k, k];

A[i, k .. n] := A[i, k .. n] - q\*A[k, k .. n];

B[i] := B[i] - q\*B[k];

end do;

end do;

x := Vector(n);

for k from n by -1 to 1 do

if A[k, k] = 0 then

if B[k] = 0 then

error "Ошибка при обратном ходе: система имеет бесконечное множество решений.";

else

error "Ошибка при обратном ходе: система не имеет решений.";

end if;

end if;

x[k] := (B[k] - add(A[k, j]\*x[j], j = k + 1 .. n))/A[k, k];

end do;

return x;

end proc:

Также был реализован метод Зейделя, который является итерационным методом решения СЛАУ. В отличие от метода простых итераций, он использует текущее значение переменной сразу после его обновления.

seidelMethod := proc(A::Matrix, B::Vector, tolerance::float, max\_iterations::integer)

local size, prev\_X, current\_X, stoper, i, j, k, max\_error, error\_, iterations, swapped, temp, sum;

size := RowDimension(A);

prev\_X := Vector(1 .. size, 0.);

current\_X := Vector(1 .. size);

stoper := false;

iterations := 0;

for i to size do

if A[i, i] = 0 then

swapped := false;

for k from i + 1 to size do

if A[k, i] <> 0 then

temp := Row(A, i);

A[i, 1 .. size] := Row(A, k);

A[k, 1 .. size] := temp;

B[i] := B[k];

B[k] := B[i];

swapped := true;

break;

end if;

end do;

if not swapped then

print("Warning: Unable to swap rows to avoid zero on the diagonal at row ", i);

return NULL;

end if;

end if;

end do;

for i to size do

sum := 0;

for j to size do

if j <> i then

sum := sum + abs(A[i, j]);

end if;

end do;

if abs(A[i, i]) < sum then

print("Warning: The matrix is not diagonally dominant at row ", i);

return NULL;

end if;

end do;

while not stoper and iterations < max\_iterations do

iterations := iterations + 1;

for i to size do

current\_X[i] := B[i];

for j to size do

if j < i then

current\_X[i] := -current\_X[j]\*A[i, j] + current\_X[i];

elif i < j then

current\_X[i] := -prev\_X[j]\*A[i, j] + current\_X[i];

end if;

end do;

current\_X[i] := current\_X[i]/A[i, i];

end do;

max\_error := 0.;

for i to size do

error\_ := abs(((A[i]) . current\_X) - B[i]);

max\_error := max(max\_error, error\_);

end do;

if max\_error < tolerance then

stoper := true;

end if;

prev\_X := current\_X;

end do;

if iterations = max\_iterations then

print("Maximum iterations reached without convergence.");

end if;

print("Maximum absolute errors in SeidelMethod = ", max\_error);

return current\_X;

end proc;

Также для сверки значений будет использована встроенная функция, с помощью которой будем находить значение корней. Листинг представлен ниже.

X := Vector[column](5, [x1, x2, x3, x4, x5]);

MX := A . X;

Maple\*solution;

Vector([seq(rhs(fsolve({MX(1) = B(1), MX(2) = B(2), MX(3) = B(3), MX(4) = B(4), MX(5) = B(5)}, {x1, x2, x3, x4, x5})[i]), i = 1 .. 5)]);

Root := Vector([seq(rhs(fsolve({MX(1) = B(1), MX(2) = B(2), MX(3) = B(3), MX(4) = B(4), MX(5) = B(5)}, {x1, x2, x3, x4, x5})[i]), i = 1 .. 5)]);

В качестве дополнительного задания оба методы были также реализованы с использованием нормализации векторов для повышения точности.

Обычные методы, как метод простой итерации, основываются на разложении системы уравнений и итеративном обновлении значений переменных. В этом подходе обновления проводятся на основе предыдущих значений без дополнительной нормализации. Это может привести к медленной сходимости, особенно для плохо обусловленных систем, где значения переменных могут значительно варьироваться.

Метод Зейделя, являющийся модификацией метода простой итерации, улучшает сходимость за счет поэтапного обновления переменных. Здесь используются текущие значения сразу после их вычисления, что может ускорить процесс. Однако, как и в случае с методом простой итерации, он не всегда обеспечивает необходимую точность, особенно в сложных случаях.

В отличие от этих методов, использование нормализации векторов в обоих подходах позволяет значительно улучшить их эффективность. Нормализация помогает поддерживать величину векторов в разумных пределах, что снижает ошибки округления и повышает общую стабильность расчетов. В результате нормализованные методы простой итерации и Зейделя достигают более точных и устойчивых решений. Это делает их более подходящими для решения сложных систем линейных уравнений, обеспечивая лучшую сходимость и меньшую погрешность.

normalize\_vector := proc(v)

local norm\_v;

norm\_v := Norm(v, 2);

if norm\_v = 0 then

return v;

else

return v / norm\_v;

end if;

end proc:

simpleIterationsByNormalize := proc(A::Matrix, B::Vector, tolerance::float, max\_iterations::integer, omega::float)

local size, prev\_X, current\_X, stoper, iterations, max\_error, error\_, i, j, sum;

size := RowDimension(A);

prev\_X := Vector(1 .. size, 0.);

current\_X := Vector(1 .. size);

stoper := false;

iterations := 0;

for i to size do

sum := 0;

for j to size do

if j <> i then sum := sum + abs(A[i, j]); end if;

end do;

if abs(A[i, i]) < sum then

print("Warning: The matrix is not diagonally dominant at row ", i);

return NULL;

end if;

end do;

while not stoper and iterations < max\_iterations do

iterations := iterations + 1;

for i to size do

current\_X[i] := B[i];

for j to size do

if j <> i then

current\_X[i] := current\_X[i] - A[i, j] \* prev\_X[j];

end if;

end do;

current\_X[i] := current\_X[i] / A[i, i];

current\_X[i] := prev\_X[i] + omega \* (current\_X[i] - prev\_X[i]);

end do;

max\_error := 0.;

for i to size do

error\_ := abs(((A[i]) . current\_X) - B[i]);

max\_error := max(max\_error, error\_);

end do;

if max\_error < tolerance then

stoper := true;

end if;

prev\_X := current\_X;

end do;

if iterations = max\_iterations then

print("Maximum iterations reached without convergence.");

end if;

return current\_X;

end proc:

seidelMethodByNormalize := proc(A::Matrix, B::Vector, tolerance::float, max\_iterations::integer, omega::float)

local size, prev\_X, current\_X, stoper, iterations, max\_error, error\_, i, j, sum;

size := RowDimension(A);

prev\_X := Vector(1 .. size, 0.);

current\_X := Vector(1 .. size);

stoper := false;

iterations := 0;

for i to size do

sum := 0;

for j to size do

if j <> i then sum := sum + abs(A[i, j]); end if;

end do;

if abs(A[i, i]) < sum then

print("Warning: The matrix is not diagonally dominant at row ", i);

return NULL;

end if;

end do;

while not stoper and iterations < max\_iterations do

iterations := iterations + 1;

for i to size do

current\_X[i] := B[i];

for j to size do

if j < i then

current\_X[i] := current\_X[i] - A[i, j] \* current\_X[j];

elif j > i then

current\_X[i] := current\_X[i] - A[i, j] \* prev\_X[j];

end if;

end do;

current\_X[i] := current\_X[i] / A[i, i];

current\_X[i] := prev\_X[i] + omega \* (current\_X[i] - prev\_X[i]);

end do;

max\_error := 0.;

for i to size do

error\_ := abs(((A[i]) . current\_X) - B[i]);

max\_error := max(max\_error, error\_);

end do;

if max\_error < tolerance then

stoper := true;

end if;

prev\_X := current\_X;

end do;

if iterations = max\_iterations then

print("Maximum iterations reached without convergence.");

end if;

return current\_X;

end proc:

# 5 АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Первый тестовый пример – условие задания. На рисунке 1 представлен результат вычисления матрицы A.

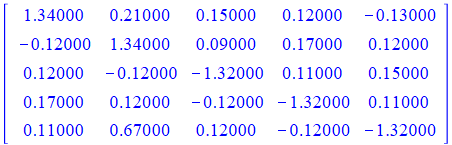


Рисунок 1 – Результат вычисления матрицы A

На рисунке 2 представлены решения, найденные с помощью описанных выше методов.

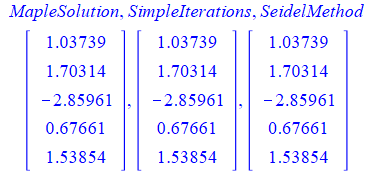


Рисунок 2 – Результат выполнения

Основываясь на рассчитанных решениях, можно сделать вывод, что все решения, использованные в данной работе, дают одинаковые решение с точностью 0,00001.

Для оценки точности решения системы линейных алгебраических уравнений (СЛАУ) используется максимальная абсолютная погрешность каждого из методов, которая вычисляется следующим образом:

max\_error := 0.;

for i to size do

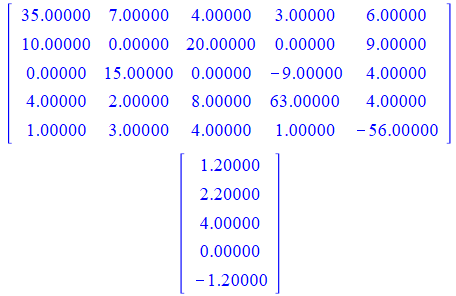
error\_ := abs(((A[i]) . current\_X) - B[i]);

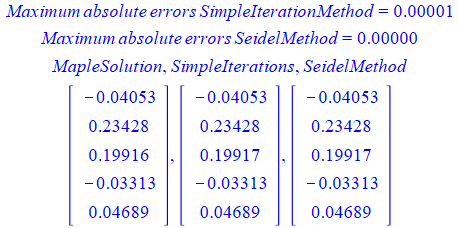
max\_error := max(max\_error, error\_);

end do;

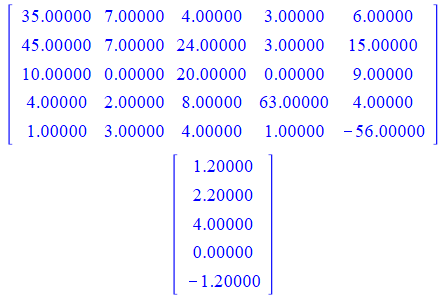
Для следующих тестовых примеров будут выбраны произвольные значения элементов матрицы A.

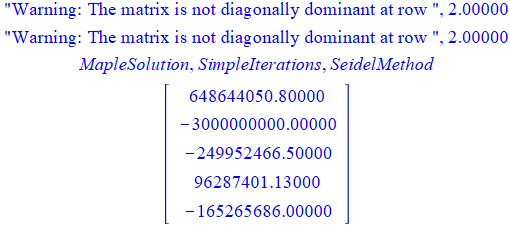
В данном примере попробуем в матрице А несколько элементов главной диагонали заполнить нулями. Начальные условия и результат выполнения представлены ниже.





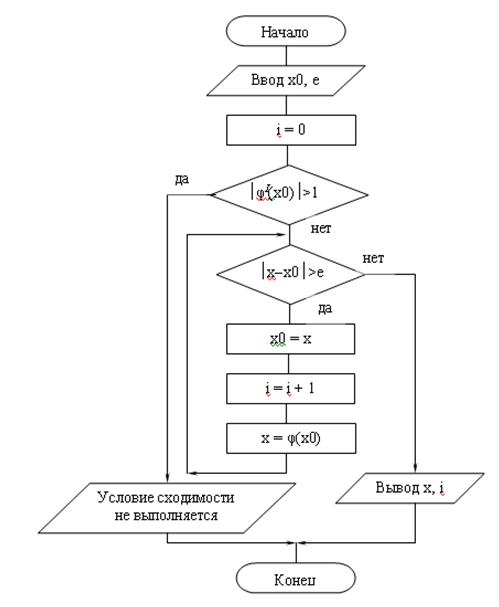
В следующем примере пусть одна из строк будет равна сумме двух других строк матрицы.

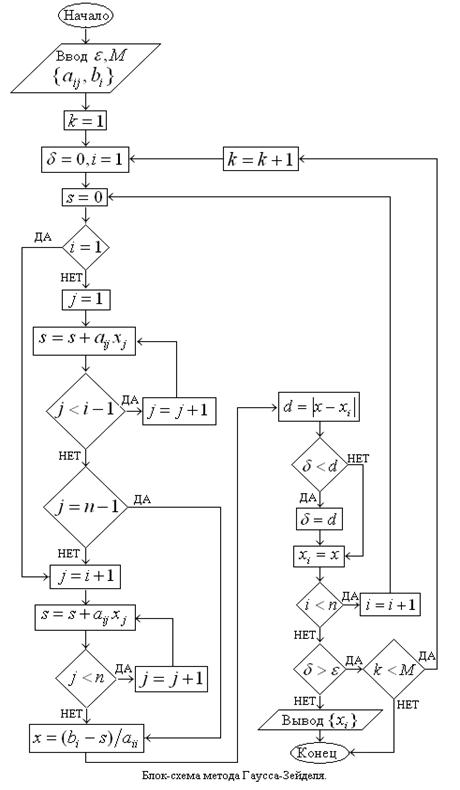


****

Если в матрице присутствует линейная зависимость, а количество строк больше трех, система уравнений, как правило, не будет сходиться к решению с использованием методов простых итераций или Зейделя. Это связано с тем, что такие методы требуют, чтобы количество независимых уравнений соответствовало количеству переменных для достижения сходимости. В противном случае система может оказаться вырожденной, что приводит к отсутствию единого решения.

# 6 Блок-схемы алгоритмов





# Вывод

В ходе выполнения данной лабораторной работы были изучены и применены итерационные методы решения СЛАУ, в частности метод простых итераций и метод Зейделя для решения системы линейных уравнений; были созданы соответствующие алгоритмы, применимые для организации вычислений на ЭВМ, и реализации программ на языке С++ для решения поставленной задачи; для проверки правильности работы программы были выполнены тестовые примеры и проведена оценка.

Сравнение трудоемкости метода простых итераций и метода Зейделя с точки зрения используемой памяти и процессорного времени может зависеть от ряда факторов, таких как размер системы уравнений, структура матрицы A, точность вычислений и специфика реализации алгоритмов на конкретном программном и аппаратном обеспечении. В целом, можно выделить следующие общие соображения:

1. **Память**:
   * **Метод простых итераций**: Требует дополнительной памяти для хранения временных переменных x^{(k+1)} и x^{(k)}. Это означает, что потребление памяти прямо пропорционально размеру вектора неизвестных x. Если система имеет большой размер, это может потребовать значительного объема дополнительной памяти.
   * **Метод Зейделя**: Обновление значений неизвестных происходит "на месте" внутри одного вектора, и поэтому требуется меньше дополнительной памяти, чем в методе простых итераций. В этом методе также потребление памяти прямо пропорционально размеру вектора неизвестных, но он может быть более эффективным с точки зрения использования памяти.
2. **Процессорное время**:
   * **Метод простых итераций**: Этот метод обычно требует большего числа итераций для сходимости, особенно если матрица A плохо обусловлена. Каждая итерация включает в себя умножение матрицы A на вектор x, что является вычислительно затратной операцией. Процессорное время может быть высоким, особенно для больших систем.
   * **Метод Зейделя**: Этот метод часто сходится быстрее, так как обновление значений неизвестных происходит более локально. Кроме того, в методе Зейделя можно использовать данные из самых последних итераций, что может ускорить сходимость. Это может привести к меньшему процессорному времени, особенно в случае систем с хорошей условной численностью.

В итоге, метод Зейделя обычно имеет небольшое преимущество с точки зрения использования памяти и может быть более эффективным с точки зрения процессорного времени, особенно для больших и плохо обусловленных систем уравнений.