



**Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ

ИУ «Информатика и системы управления»

КАФЕДРА

ИУ1 «Системы автоматического управления»

ОТЧЕТ

по лабораторной работе №1

«Методы решения СЛАУ»

по дисциплине

«Методы вычислений»

Выполнила: Санникова А.П.

Группа: ИУ1 – 32Б

Проверил: Бобков А.В.

Работа выполнена:

Отчет сдан:

Оценка:

Москва 2022

Оглавление

1. Метод Крамера	3
2. Метод Жордана – Гаусса	4
3. Метод решения с помощью разложения Холецкого	6
4. Метод Якоби	9
5. Метод Гаусса – Зейделя	11
6. Метод релаксаций	13
7. Метод градиентного спуска	14
Сравнение производительности методов	16

1. Метод Крамера

Как правило, данный метод применяется только для тех систем, где по количеству неизвестных столько же, сколько и уравнений. Чтобы получилось решить уравнение, главный определитель матрицы не должен равняться нулю.

Метод Крамера является универсальным и максимально точным, так как в нём присутствует малое количество операций деления.

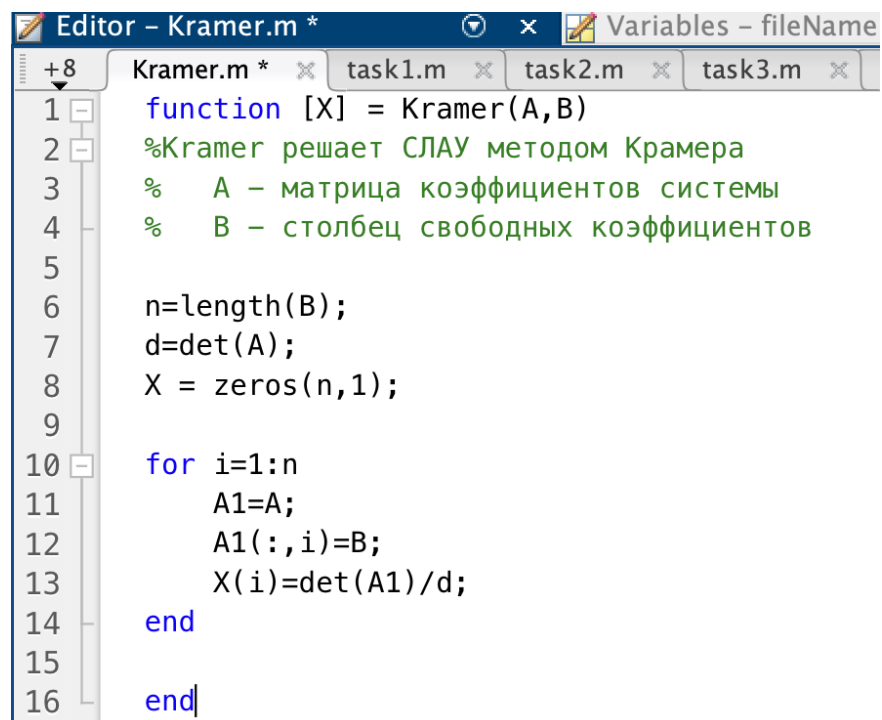
Вычислительная сложность: $T_{\text{опр}} = O(N^3)$ и $T_{\text{общ}} = O(N^4)$ - не может работать в реальном времени.

Алгоритм:

- вычислить главный определитель матрицы коэффициентов (не должен равняться 0);

- вычислить определители матриц, получающихся подстановкой столбца свободных членов B на место i-го столбца в матрице коэффициентов A;

- вычислить корни уравнения $x_i = \frac{\Delta_i}{\Delta}$



```
Editor - Kramer.m *
Variables - fileName
Kramer.m * task1.m task2.m task3.m
1 function [X] = Kramer(A,B)
2 %Kramer решает СЛАУ методом Крамера
3 % A - матрица коэффициентов системы
4 % B - столбец свободных коэффициентов
5
6 n=length(B);
7 d=det(A);
8 X = zeros(n,1);
9
10 for i=1:n
11     A1=A;
12     A1(:,i)=B;
13     X(i)=det(A1)/d;
14 end
15
16 end
```

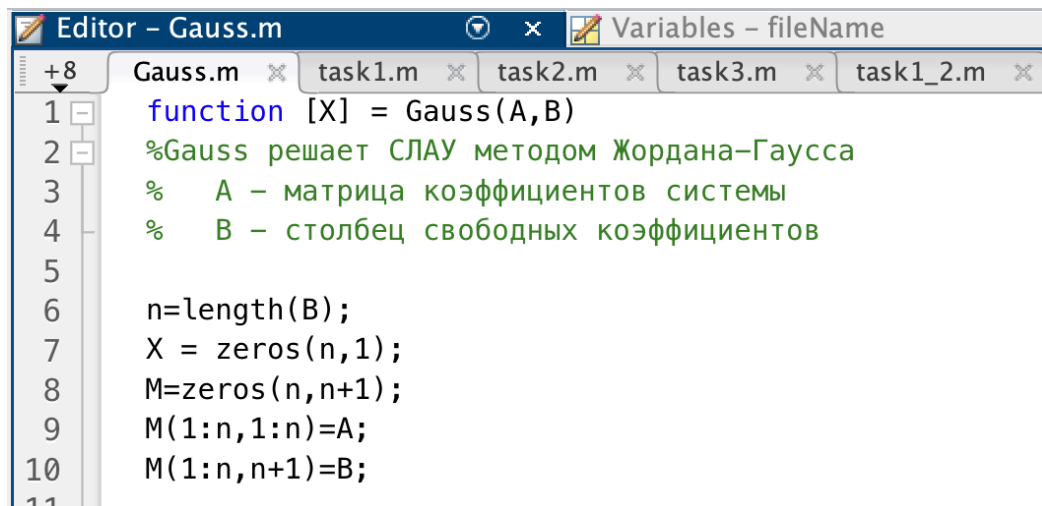
2. Метод Жордана – Гаусса

Этот метод является модификацией метода Гаусса — в отличие от исходного (метода Гаусса) метод Жордана-Гаусса позволяет решить СЛАУ в один этап (без использования прямого и обратного ходов). Он является также универсальным, но менее точным, чем другие методы из-за деления.

Вычислительная сложность: $T = O(N^3)$ - может работать в реальном времени при маленьком N .

Алгоритм:

- составить расширенную матрицу из матрицы коэффициентов и столбца свободных членов и привести ее левую часть к диагональному виду;
- правая часть расширенной матрицы и есть решение системы.



```
Editor - Gauss.m
Variables - fileName
Gauss.m task1.m task2.m task3.m task1_2.m
1 function [X] = Gauss(A,B)
2 %Gauss решает СЛАУ методом Жордана-Гаусса
3 % A - матрица коэффициентов системы
4 % B - столбец свободных коэффициентов
5
6 n=length(B);
7 X = zeros(n,1);
8 M=zeros(n,n+1);
9 M(1:n,1:n)=A;
10 M(1:n,n+1)=B;
11
```

```

12 for i=1:n
13     if M(i,i) == 0
14         for j =i+1:n
15             if M(j,i)~=0
16                 M(i,1:n+1)=M(i,1:n+1)+M(i,n+1);
17                 break
18             end
19             if M(i,j) == 0
20                 return
21             end
22         end
23     end
24     M(i,1:n)=M(i,1:n)/M(i,i);
25     for j=1:n
26         if j~=i
27             M(j,1:n+1)=M(j,:)-M(j,i)*M(i,:);
28         end
29     end
30     X=M(1:n,n+1);
31 end

```

3. Метод решения с помощью разложения Холецкого

Разложение Холецкого (метод квадратного корня) — представление симметричной положительно определённой матрицы A в виде $A=LL^T$, где L — нижняя треугольная матрица со строго положительными элементами на диагонали. Разложение Холецкого всегда существует и единственно для любой симметричной положительно определённой матрицы.

Элементы матрицы L можно вычислить, начиная с верхнего левого угла матрицы, по формулам

$$\begin{aligned}l_{11} &= \sqrt{a_{11}}, \\l_{j1} &= \frac{a_{j1}}{l_{11}}, \quad j \in [2, n], \\l_{ii} &= \sqrt{a_{ii} - \sum_{p=1}^{i-1} l_{ip}^2}, \quad i \in [2, n], \\l_{ji} &= \frac{1}{l_{ii}} \left(a_{ji} - \sum_{p=1}^{i-1} l_{ip} l_{jp} \right), \quad i \in [2, n-1], j \in [i+1, n].\end{aligned}$$

Тогда СЛАУ $AX = B$ можно заменить системой

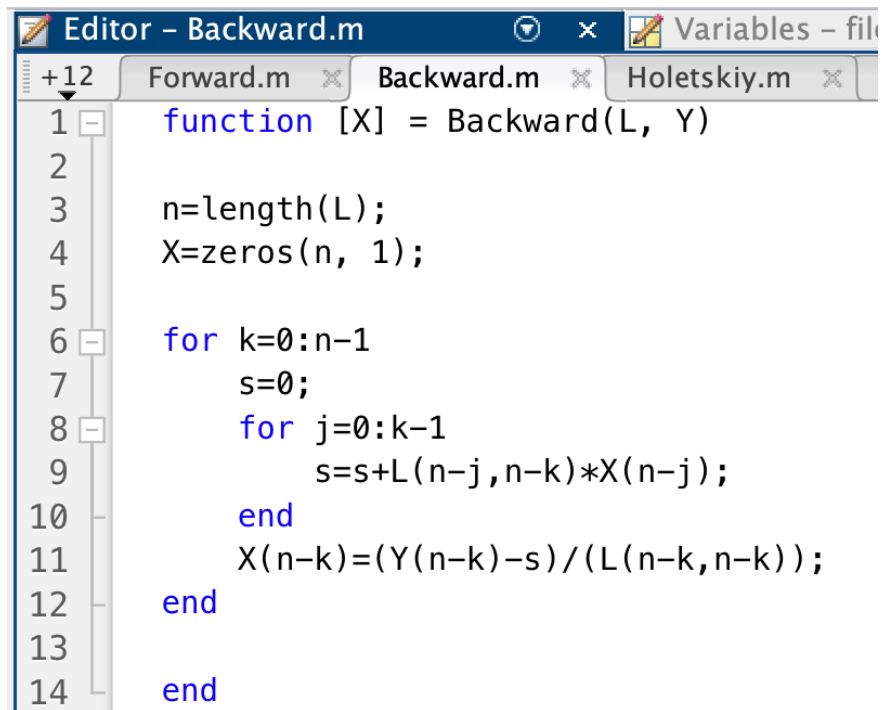
$$\{LY = B L^T X = Y\}$$

Метод квадратного корня является еще менее точным по своему определению и неуниверсальным.

Вычислительная сложность: $T_Y = O(N^2)$ и $T_{\text{общ}} = O(N^3)$.

```
Editor - Holetskiy.m
Variables - fileName
Mechanics Explorers
Forward.m Backward.m Holetskiy.m Gauss.m task1.m task2.m task3.m
1 function [L] = Holetskiy(A)
2 %Holetskiy разделяет симметричную и положительно определенную матрицу A на
3 %две треугольных матрицы
4 %A - матрица коэффициентов системы
5
6 n=length(A);
7 L=zeros(n,n);
8
9 for i=1:n
10     %Поддиагональные
11     for j=1:i-1
12         s=0;
13         for k=1:j-1
14             s=s+L(i,k)*L(j,k);
15         end
16         L(i,j)=(A(i,j)-s)/L(j,j);
17     end
18     %Диагональные
19     s=0;
20     for j=1:i-1
21         s=s+L(i,j)^2;
22     end
23     L(i,i)=sqrt(A(i,i)-s);
24 end
25
26 end
```

```
Editor - Forward.m *
Forward.m * Backward.m Holetski
1 function [Y] = Forward(L,B)
2
3 n=length(L);
4 Y=zeros(n,1);
5
6 for i=1:n
7     s=0;
8     for j=1:i-1
9         s=s+L(i,j)*Y(j);
10    end
11    Y(i)=(B(i)-s)/L(i,i);
12 end
13
14 end
```



The image shows a MATLAB Editor window titled "Editor - Backward.m". The window contains a script for a function named "Backward". The code is as follows:

```
1 function [X] = Backward(L, Y)
2
3     n=length(L);
4     X=zeros(n, 1);
5
6     for k=0:n-1
7         s=0;
8         for j=0:k-1
9             s=s+L(n-j, n-k)*X(n-j);
10        end
11        X(n-k)=(Y(n-k)-s)/(L(n-k, n-k));
12    end
13
14 end
```


4. Метод Якоби

Метод Якоби – это итерационный и численный метод решения СЛАУ. Суть его заключается в расщеплении матрицы коэффициентов A на диагональную D и остаток R .

Тогда новое решение системы X_{t+1} можно найти, пользуясь менее точным решением X_t :

$$AX=B \Rightarrow X_{t+1} = D^{-1}[B - RX_t]$$

Такой процесс должен быть остановлен при достижении минимальной ошибки вычислений. Ошибку можно косвенно оценить по приращению

$$\varepsilon = \sum_{i=1}^n (x_i^* - x_i) < \varepsilon_0$$

Метод Якоби не универсален, т.к. для его сходимости обязательно условие диагонального доминирования в матрице коэффициентов.

Вычислительная сложность: $T = O(N^2)$ - быстрый. Хорошо распараллеливается.

```
Editor - Yacoby.m  Variables - fileName
lab1_kramer.m  Yacoby.m  Forward.m  Backw...
1  function X = Yacoby(A,B,e0,T)
2
3  %Yacoby решает СЛАУ методом Якоби
4
5  n=length(A);
6  X=zeros(n,1);
7  X1=X;
8  e=e0+1;
9
10 while e>e0
11     for i=1:n
12         s=0;
13         for j=1:n
14             if i~=j
15                 s=s+A(i,j)*X(j);
16             end
17         end
18         X1(i)=(B(i)-s)/A(i,i);
19     end
20     e=sum(abs(X1-X));
21     X=X1;
22     T=T-1;
23     if T<0
24         break;
25     end
26 end
```

5. Метод Гаусса – Зейделя

Метод Гаусса-Зейделя – это итерационный и численный метод решения СЛАУ. Суть его заключается в расщеплении матрицы коэффициентов A на диагональную D , верхнюю треугольную R и нижнюю треугольную L .

Тогда новое решение системы X_{t+1} можно найти, пользуясь решением предыдущей итерации X_t :

$$AX=B \Rightarrow X_{t+1} = (L + D)^{-1}[B - RX_t]$$

Такой процесс должен быть остановлен при достижении минимальной ошибки вычислений. Ошибку можно косвенно оценить по приращению

$$\varepsilon = \sum_{i=1}^n (x_i^* - x_i) < \varepsilon_0$$

Метод Гаусса-Зейделя также не универсален, т.к. для его сходимости обязательно условие диагонального доминирования в матрице коэффициентов.

Вычислительная сложность: $T = O(N^2)$ - быстрый (в 2-3 раза быстрее Якоби). Достижима любая точность. Не распараллеливается.

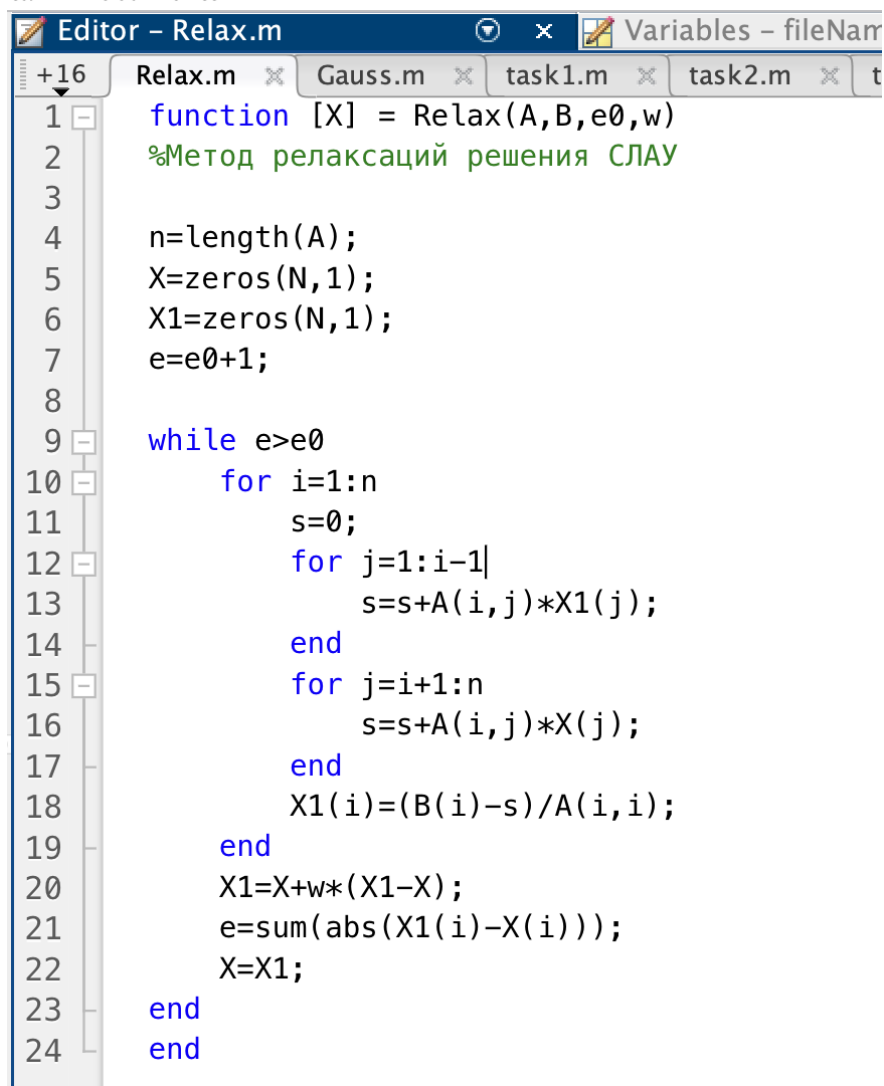
```
Editor - Saidel.m
+15
Saidel.m x Forward.m x Backward.m x Hol
1 function X = Saidel(A,B,e0,T)
2
3     n=length(A);
4     X=zeros(n,1);
5     X1=X;
6     e=e0+1;
7
8     while e>e0
9         for i=1:n
10            s=0;
11            for j=1:i-1
12                s=s+A(i,j)*X(j);
13            end
14            for j=i+1:n
15                s=s+A(i,j)*X(j);
16            end
17            X1(i)=(B(i)-s)/A(i,i);
18        end
19        e=sum(abs(X1-X));
20        T=T-1;
21        X=X1;
22        if T<0
23            break
24        end
25    end
26    end
```

6. Метод релаксаций

Метод верхних релаксаций – итерационный метод решения СЛАУ. Его суть заключается в следующем: после вычисления очередного приближения решения СЛАУ X_{t+1} , например, по Гауссу – Зейделю, эту компоненту дополнительно смещают на некоторую величину ω , что должно предоставить возможность как можно быстрее найти наиболее точное решение.

$$X' = X_t + \omega [X_{t+1} - X_t]$$

Вычислительная сложность: $T = O(N^2)$ - быстрый. Достижима любая точность. Не распараллеливается.



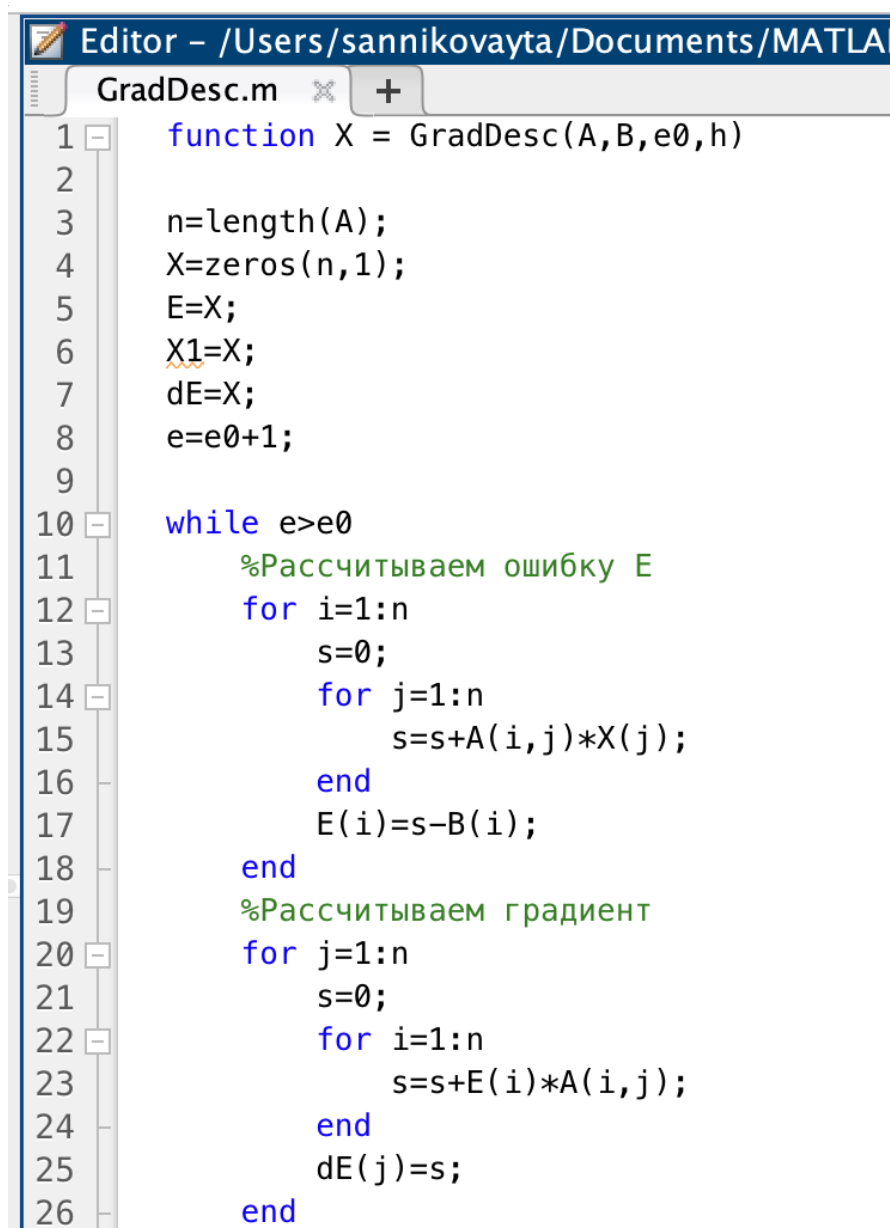
```
Editor - Relax.m
+16
1 function [X] = Relax(A,B,e0,w)
2 %Метод релаксаций решения СЛАУ
3
4 n=length(A);
5 X=zeros(N,1);
6 X1=zeros(N,1);
7 e=e0+1;
8
9 while e>e0
10     for i=1:n
11         s=0;
12         for j=1:i-1
13             s=s+A(i,j)*X1(j);
14         end
15         for j=i+1:n
16             s=s+A(i,j)*X(j);
17         end
18         X1(i)=(B(i)-s)/A(i,i);
19     end
20     X1=X+w*(X1-X);
21     e=sum(abs(X1(i)-X(i)));
22     X=X1;
23 end
24 end
```

7. Метод градиентного спуска

Метод градиентного спуска позволяет прийти к более точному решению, зная необходимое направление (градиент ошибки). Шаг (скорость) оптимизации h в таком случае может быть константой, дробным (уменьшаться) или вычисленным наискорейшим спуском. Данный метод также нуждается в принудительной остановке.

$$X^* = X - h * \text{grad}E$$

Этот метод универсален. Его скорость медленнее остальных итерационных методов, но быстрее, чем прямые.

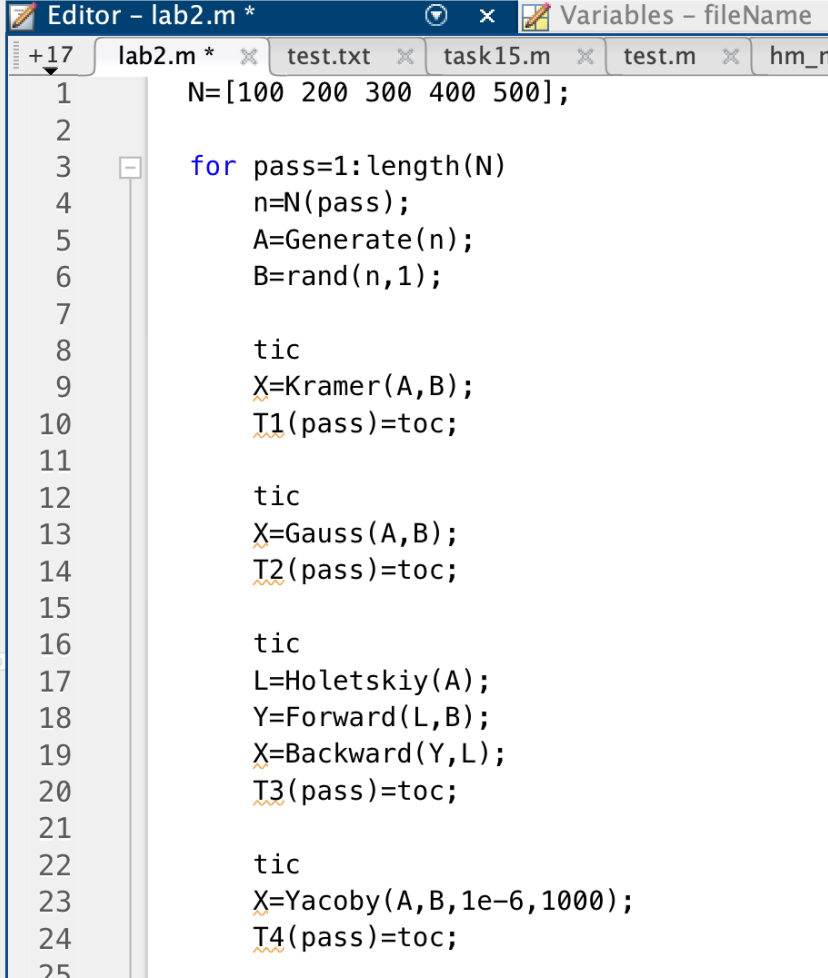


```
Editor - /Users/sannikovayta/Documents/MATLAB
GradDesc.m x +
1 function X = GradDesc(A,B,e0,h)
2
3     n=length(A);
4     X=zeros(n,1);
5     E=X;
6     X1=X;
7     dE=X;
8     e=e0+1;
9
10    while e>e0
11        %Рассчитываем ошибку E
12        for i=1:n
13            s=0;
14            for j=1:n
15                s=s+A(i,j)*X(j);
16            end
17            E(i)=s-B(i);
18        end
19        %Рассчитываем градиент
20        for j=1:n
21            s=0;
22            for i=1:n
23                s=s+E(i)*A(i,j);
24            end
25            dE(j)=s;
26        end
```

```
27      %Рассчитываем X
28      X1=X-h*dE;
29      e=sum(abs(e));
30      X=X1;
31  end
32  end
33
```

Сравнение производительности методов

Сравнить производительность методов можно, построив графики зависимости времени поиска решения от размерности матрицы коэффициентов.



```
Editor - lab2.m *
lab2.m * test.txt task15.m test.m hm_n
+17
1 N=[100 200 300 400 500];
2
3 for pass=1:length(N)
4     n=N(pass);
5     A=Generate(n);
6     B=rand(n,1);
7
8     tic
9     X=Kramer(A,B);
10    T1(pass)=toc;
11
12    tic
13    X=Gauss(A,B);
14    T2(pass)=toc;
15
16    tic
17    L=Holetskiy(A);
18    Y=Forward(L,B);
19    X=Backward(Y,L);
20    T3(pass)=toc;
21
22    tic
23    X=Yacoby(A,B,1e-6,1000);
24    T4(pass)=toc;
25
```



```

26         tic
27         X=Saidel(A,B,1e-6,1000);
28         T5(pass)=toc;
29
30         tic
31         X=Relax(A,B,1e-6,0.01);
32         T6(pass)=toc;
33
34         tic
35         X=GradDesc(A,B,1e-6,0.001);
36         T7(pass)=toc;
37
38     end
39
40     figure,plot(N,T1,N,T2,N,T3,N,T4,N,T5,N,T6,N,T7);
41     legend('Kramer','Gauss','Holetsy','Jacoby','Zeidel','Relax','Gradient');
42     grid on
43
44     figure,plot(N,T4,N,T5,N,T6,N,T7);
45     legend('Jacoby','Zeidel','Relax','Gradient');
46     grid on

```

