



ITMO UNIVERSITY



ИТМО ВТ

Отчет по Лабораторной работе №1
по курсу “Вычислительная математика”

Вариант №5 (Метод Гаусса)

Выполнил:
Студент группы Р32082
Панин Иван Михайлович

Преподаватель:
Машина Екатерина Алексеевна

Санкт-Петербург, 2023

Лабораторная работа 1. «Решение системы линейных алгебраических уравнений СЛАУ»

1. № варианта определяется как номер в списке группы согласно ИСУ.
2. В программе численный метод должен быть реализован в виде отдельной подпрограммы/метода/класса, в который исходные/выходные данные передаются в качестве параметров.
3. Размерность матрицы $n \leq 20$ (задается из файла или с клавиатуры - по выбору конечного пользователя).
4. Должна быть реализована возможность ввода коэффициентов матрицы, как с клавиатуры, так и из файла (по выбору конечного пользователя).

Для прямых методов должно быть реализовано:

- Вычисление определителя
- Вывод треугольной матрицы (включая преобразованный столбец В)
- Вывод вектора неизвестных: x_1, x_2, \dots, x_n
- Вывод вектора невязок: r_1, r, \dots, r_n

Для итерационных методов должно быть реализовано:

- Точность задается с клавиатуры/файла
- Проверка диагонального преобладания (в случае, если диагональное преобладание в исходной матрице отсутствует, сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто). В случае невозможности достижения диагонального преобладания - вывести соответствующее сообщение.
- Вывод вектора неизвестных: x_1, x_2, \dots, x_n
- Вывод количества итераций, за которое было найдено решение.
- Вывод вектора погрешностей: $|x_i^{(k)} - x_i^{(k-1)}|$

Содержание отчета:

- Цель работы,
- Описание метода, расчетные формулы,
- Листинг программы (по крайней мере, где реализован сам метод)
- Примеры и результаты работы программы,
- Выводы.
- Отчет предоставляется в электронном/бумажном виде.

1. Цель работы.

- Научиться программировать решения системы линейных алгебраических уравнений, обрабатывать ошибки, понимать разницу между прямыми и итерационными методами.

2. Описание метода, расчетные формулы.

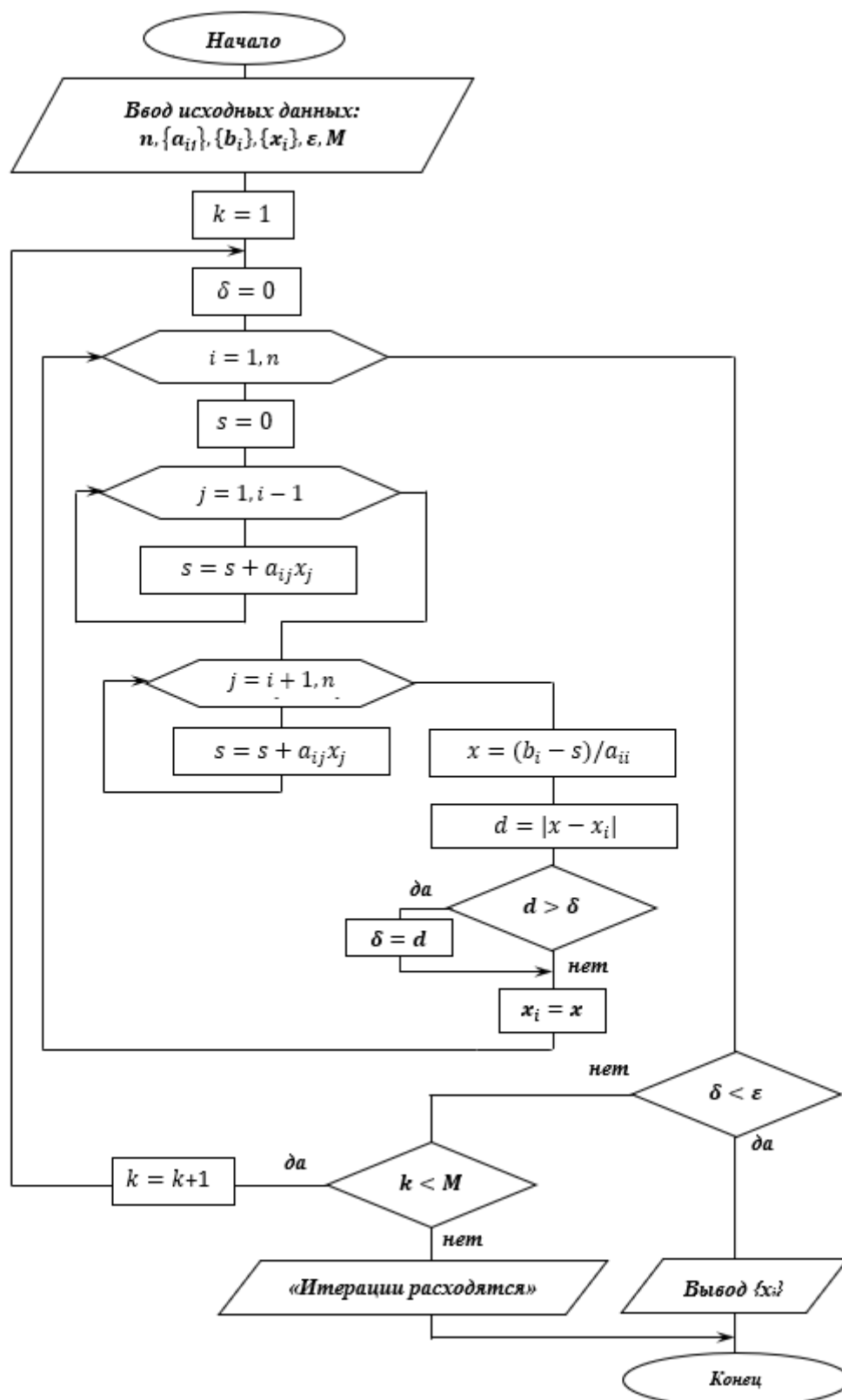
$$x_{n-1} = b_{n-1} / a_{n-1,n-1},$$

$$x_i = (b_i - \sum_{j=i+1}^{n-1} a_{ij}x_j) / a_{ii}, \quad i = n-2, n-3, \dots, 0$$

Для начала постулируем, что пользователем на вход была дана совместная матрица.

- 1. С помощью функции `add_all` избавляемся от нулей в элементах a_{ii} путём прибавления к рассматриваемой строке строк, таких, что в них тот же по порядку элемент $\neq 0$.
- 2. С помощью функции `gauss()` приводим матрицу вначале к нижнетреугольному виду (прямым ходом), а потом — к верхнетреугольному (обратным ходом). Алгоритм Гаусса подразумевает последовательное вычитание из последующих строк, строки, рассматриваемой на данной итерации алгоритма, умноженной на такой коэффициент, чтобы элемент последующей строки, соответствующий элементу a_{ii} рассматриваемой строки, стал равен нулю, в результате вычитания. Вначале идём „сверху вниз“, тем самым приводя матрицу к нижнетреугольному виду, а потом идём „снизу вверх“, тем самым приводя матрицу к верхнетреугольному виду.
- 3. С помощью функции `one_diagonal` приводим все диагональные элементы матрицы к единице. Таким образом, на диагонали матрицы будут единичные элементы и найти вектор с корнями не составит труда.
- 4. С помощью функции `extranw()`, собственно, находим вектор с корнями.
- 5. С помощью функции `find_nevyaz()`, находим вектор невязок, подставив значения из вектора с корнями в исходную матрицу и вычтя полученные значения из свободного члена.
- 6. С помощью функции `determinant()` или функции `determinant_fpu()`, приводим прямым ходом алгоритма Гаусса исходную матрицу к нижнетреугольному виду, и далее, находим произведение диагональных элементов, что и будет значением определителя.

Блок-схема:



3. Листинг программы.

```

align 8
add_all: ; rsi - ptr to equ, rdi - height, rdx - width
mov r8, rdx

```

```

xor r9, r9
._checkloop:
    pxor xmm1, xmm1
    mov rax, r8
    xor rdx, rdx
    mul r9
    add rax, r9
    movss xmm0, [rsi+rax*4]
    cmpss xmm1, xmm0, 0
    movd edx, xmm1
    and edx, edx
    jnz ._zero_found
    ._zero_fixed:
    inc r9
    cmp r9, rdi
jnz ._checkloop

```

```

xor rax, rax

```

```

ret

```

```

._zero_found:
    xor r10, r10
    ._find_nzero_loop:
        pxor xmm1, xmm1
        mov rax, r8
        xor rdx, rdx
        mul r10
        add rax, r9
        movss xmm0, [rsi+rax*4]
        cmpss xmm1, xmm0, 0
        movd edx, xmm1
        and edx, edx
        jz ._add_to_zero
        inc r10
        cmp r10, rdi
    jnz ._find_nzero_loop

```

```

mov rax, 1

```

```

ret

```

```

    ._add_to_zero: ; r9 - row with zero, r10 - row to add

```

```

        mov rcx, r8
        shr rcx, 3

```

```

        mov rax, r8
        xor rdx, rdx
        mul r9
        mov r11, rax

```

```

        mov rax, r8
        xor rdx, rdx
        mul r10

```

```

;dec rcx
shl rcx, 3
add rax, rcx
add r11, rcx
shr rcx, 3
;inc rcx

._addloop:
    sub rax, 8
    sub r11, 8
    vmovups ymm0, [rsi+r11*4] ; row with zero
    vmovups ymm1, [rsi+rax*4]
    vaddps ymm0, ymm1
    vmovups [rsi+r11*4], ymm0
    dec rcx
    jnz ._addloop
jmp ._zero_fixed

```

align 8

one_diagonal: ; rdi - height, rsi - equ, rdx - width

```
mov r8, rdx
```

```
xor r9, r9
```

._loop:

```

mov rax, r8
xor rdx, rdx
mul r9
lea rax, [rsi+rax*4]

```

```

mov edx, 0x3F800000
movd xmm1, edx
movss xmm2, [rax+r9*4]
divss xmm1, xmm2
vbroadcastss ymm3, xmm1

```

```

mov rcx, r8
shr rcx, 3
;dec rcx
shl rcx, 5
add rax, rcx
shr rcx, 5
;inc rcx

```

._sloop:

```
sub rax, 32
```

```

vmovups ymm0, [rax]
vmulps ymm0, ymm3, ymm0
vmovups [rax], ymm0

```

```

                                dec rcx
                                jnz ._sloop

                                shr rcx, 5
                                inc rcx

                                inc r9
                                cmp r9, rdi
                                jnz ._loop

ret

```

align 8

```

gauss: ; rdi - answ, rsi - equ, rdx - width, rcx - height
        mov r8, rdx ; preserve width, since rdx is used by mul
        xor r9, r9 ; height counter
        push r12
        ;push rdi

        ._loop_straight:
            mov rax, r8
            xor rdx, rdx
            mul r9
            mov r11, rax
            lea r11, [rsi+r11*4]
            lea r10, [r9+1]
            cmp r10, rcx
            jz ._backw
            ;vmovups ymm0, [r11]
            ._loop_straight_mul:
                mov rax, r8
                xor rdx, rdx
                mul r10
                lea rax, [rsi+rax*4]

                movss xmm1, [r11+r9*4]
                movss xmm2, [rax+r9*4]
                pxor xmm4, xmm4
                cmpss xmm4, xmm2, 0
                movd edx, xmm4
                and edx, edx
                jnz ._loop_straight_mul_zeroed

                vdivss xmm1, xmm2, xmm1

                ;movss [rax+r9*4], xmm1
                ;jmp ._loop_straight_1ymm_mul_zeroed

                vbroadcastss ymm4, xmm1

```

```

mov rdi, r8
shr rdi, 3
;dec rcx
shl rdi, 5
add rax, rdi
add r11, rdi
shr rdi, 5
;inc rcx

._sm_loop:
    sub rax, 32
    sub r11, 32
    vmovups ymm0, [r11]
    vmulps ymm3, ymm0, ymm4
    vmovups ymm1, [rax]
    vsubps ymm1, ymm1, ymm3
    vmovups [rax], ymm1

    dec rdi
jnz ._sm_loop

._loop_straight_mul_zeroed:

inc r10
cmp r10, rcx
jnz ._loop_straight_mul

    lea r10, [r9+1]
    mov rax, r8
    xor rdx, rdx
    mul r10
    lea r11, [rsi+rax*4]
    pxor xmm4, xmm4
    movss xmm0, [r11+r10*4]
    cmpss xmm4, xmm0, 0
    movd edx, xmm4
    and edx, edx
    jnz ._diag_next_zero_straight
    ._fixed_kinda:

inc r9
cmp r9, rcx
jnz ._loop_straight

._backw:
lea r9, [rcx-1]

;mov rax, 228
;ret

```



```

    ._loop_backward:
        mov rax, r8
        xor rdx, rdx
        mul r9
        mov r11, rax
        lea r11, [rsi+r11*4]
        lea r10, [r9-1]
        cmp r10, 0
        jl .end
        ;vmovups ymm0, [r11]
    ._loop_backward_mul:
        mov rax, r8
        xor rdx, rdx
        mul r10
        lea rax, [rsi+rax*4]

        movss xmm1, [r11+r9*4]
        movss xmm2, [rax+r9*4]
        pxor xmm4, xmm4
        cmpss xmm4, xmm2, 0
        movd edx, xmm4
        and edx, edx
        jnz ._loop_backward_mul_zeroed

        vdivss xmm1, xmm2, xmm1

        ;movss [rax+r9*4], xmm1
        ;jmp ._loop_straight_1ymm_mul_zeroed

        vbroadcastss ymm4, xmm1

        mov rdi, r8
        shr rdi, 3
        ;dec rcx
        shl rdi, 5
        add rax, rdi
        add r11, rdi
        shr rdi, 5
        ;inc rcx

    ._sm_loop_bckw:
        sub rax, 32
        sub r11, 32
        vmovups ymm0, [r11]
        vmulps ymm3, ymm0, ymm4
        vmovups ymm1, [rax]
        vsubps ymm1, ymm1, ymm3
        vmovups [rax], ymm1

        dec rdi

```

```
jnz  ._sm_loop_bckw
```

```
._loop_backward_mul_zeroed:
```

```
dec  r10  
cmp  r10, 0  
jge  ._loop_backward_mul
```

```
dec  r9  
jnz  ._loop_backward
```

```
.end:
```

```
pop  r12  
xor  rax, rax  
ret
```

```
._diag_next_zero_straight: ; r10 - row number with zero  
lea  r12, [r10+1]
```

```
._diag_next_zero_straight_loop:
```

```
mov  rax, r8  
xor  rdx, rdx  
mul  r12  
lea  rax, [rsi+rax*4]
```

```
pxor  xmm4, xmm4  
movss xmm0, [rax+r10*4]  
cmpss xmm4, xmm0, 0  
movd  edx, xmm4  
and  edx, edx  
jz  ._diag_next_zero_straight_loop_add  
inc  r12  
cmp  r12, rcx
```

```
jnz  ._diag_next_zero_straight_loop  
jmp  ._fixed_kinda
```

```
._diag_next_zero_straight_loop_add:
```

```
mov  rdx, r8
```

```
shl  rdx, 2  
add  rax, rdx  
add  r11, rdx  
shr  rdx, 5
```

```
._diag_next_zero_straight_loop_add_sm_loop:
```

```
sub  rax, 32  
sub  r11, 32  
vmovups ymm0, [r11]
```

```

        vmovups ymm1, [rax]
        vaddps ymm0, ymm0, ymm1
        vmovups [r11], ymm0
        dec rdx
        jnz ._diag_next_zero_straight_loop_add
jmp     ._fixed_kinda

```

align 8

extranws: ; rsi - equ, rdi - ansdbuf, rcx - height, rdx - width

```

mov r8, rdx
xor r9, r9
.anws_loop:
    mov rax, r8
    xor rdx, rdx
    mul r9
    add rax, rcx

    mov eax, dword [rsi+rax*4]
    stosd

    inc r9
    cmp r9, rcx
jnz .anws_loop
ret

```

align 8

find_nevyaz: ; rdi - ptr to nevyaz, rsi - ptr to equ copy, rdx - ptr to answ,
rcx - height, r8 - width

```

push r12
mov r12, rdx

lea rdx, [r8*4]
sub rsp, rdx
xor r9, r9
._parsebloop:
    mov rax, r8
    mul r9
    lea rax, [rsi+rax*4]
    mov edx, dword [rax+rcx*4]
    mov dword [rsp+r9*4], edx
    mov dword [rax+rcx*4], 0

    inc r9
    cmp r9, rcx
jnz ._parsebloop

xor r9, r9 ; height counter
._loop_mul:

```

```

    mov rax, r8
    mul r9
    lea r11, [rsi+rax*4] ; addr of the current row

    mov rax, r12

    mov r10, r8
    shl r10, 2
    add rax, r10
    add r11, r10
    shr r10, 5

    __sm_loop:
        sub rax, 32
        sub r11, 32
        vmovups ymm0, [rax]
        vmovups ymm1, [r11]
        vmulps ymm1, ymm0, ymm1
        vmovups [r11], ymm1

        dec r10
    jnz __sm_loop

    inc r9
    cmp r9, rcx
jnz __loop_mul

xor r9, r9
__hadd_loop:
    mov rax, r8
    mul r9
    lea rax, [rsi+rax*4] ; cur row BAR

    movss xmm0, [rax]
    mov r10, 1
    __hadd_loop_2:
        addss xmm0, [rax+r10*4]
        inc r10
        cmp r10, rcx
    jnz __hadd_loop_2
    movss xmm1, [rsp+r9*4] ; get cur b
    subss xmm1, xmm0
    movss [rdi+r9*4], xmm1

    inc r9
    cmp r9, rcx
jnz __hadd_loop

    lea rsp, [rsp+r8*4]

```

```
pop r12
```

```
xor rax, rax  
ret
```

```
determinant: ; rdi - det ptr, rsi - ptr to equ copy, rdx - width, rcx - height  
mov r8, rdx ; preserve width, since rdx is used by mul  
xor r9, r9 ; height counter  
push rdi  
;push rdi
```

```
._loop_straight:  
    mov rax, r8  
    xor rdx, rdx  
    mul r9  
    mov r11, rax  
    lea r11, [rsi+r11*4]  
    lea r10, [r9+1]  
    cmp r10, rcx  
    jz ._mul_diag  
    ;vmovups ymm0, [r11]  
    ._loop_straight_mul:  
        mov rax, r8  
        xor rdx, rdx  
        mul r10  
        lea rax, [rsi+rax*4]  
  
        movss xmm1, [r11+r9*4]  
        movss xmm2, [rax+r9*4]  
        pxor xmm4, xmm4  
        cmpss xmm4, xmm2, 0  
        movd edx, xmm4  
        and edx, edx  
        jnz ._loop_straight_mul_zeroed  
  
        vdivss xmm1, xmm2, xmm1  
  
        ;movss [rax+r9*4], xmm1  
        ;jmp ._loop_straight_1ymm_mul_zeroed  
  
        vbroadcastss ymm4, xmm1  
  
        mov rdi, r8  
        shr rdi, 3  
        ;dec rcx  
        shl rdi, 5  
        add rax, rdi  
        add r11, rdi  
        shr rdi, 5  
        ;inc rcx
```

```

        ._sm_loop:
            sub rax, 32
            sub r11, 32
            vmovups ymm0, [r11]
            vmulps ymm3, ymm0, ymm4
            vmovups ymm1, [rax]
            vsubps ymm1, ymm1, ymm3
            vmovups [rax], ymm1

            dec rdi
        jnz ._sm_loop

```

```

._loop_straight_mul_zeroed:

    inc r10
    cmp r10, rcx
    jnz ._loop_straight_mul

```

```

    inc r9
    cmp r9, rcx
    jnz ._loop_straight

```

```

._mul_diag:
    pop rdi
    mov r9, 1
    movss xmm0, [rsi]
._mul_diag_loop:
        mov rax, r8
        mul r9
        lea rax, [rsi+rax*4]
        mulss xmm0, [rax+r9*4]
        inc r9
        cmp r9, rcx
    jnz ._mul_diag_loop
    movss [rdi], xmm0
    xor rax, rax

```

```
ret
```

determinant_fpu: ; rdi - det ptr, rsi - ptr to equ copy, rdx - width, rcx - height

```

    mov r8, rdx ; preserve width, since rdx is used by mul
    xor r9, r9 ; height counter
    push rdi
    ;push rdi

```

```

._loop_straight:
    mov rax, r8

```

```

xor rdx, rdx
mul r9
mov r11, rax
lea r11, [rsi+r11*4]
lea r10, [r9+1]
cmp r10, rcx
jz ._mul_diag
;vmovups ymm0, [r11]
._loop_straight_mul:
    mov rax, r8
    xor rdx, rdx
    mul r10
    lea rax, [rsi+rax*4]

    movss xmm1, [r11+r9*4]
    movss xmm2, [rax+r9*4]
    pxor xmm4, xmm4
    cmpss xmm4, xmm2, 0
    movd edx, xmm4
    and edx, edx
    jnz ._loop_straight_mul_zeroed

    vdivss xmm1, xmm2, xmm1

    ;movss [rax+r9*4], xmm1
    ;jmp ._loop_straight_1ymm_mul_zeroed

    vbroadcastss ymm4, xmm1

    mov rdi, r8
    shr rdi, 3
    ;dec rcx
    shl rdi, 5
    add rax, rdi
    add r11, rdi
    shr rdi, 5
    ;inc rcx

    ._sm_loop:
        sub rax, 32
        sub r11, 32
        vmovups ymm0, [r11]
        vmulps ymm3, ymm0, ymm4
        vmovups ymm1, [rax]
        vsubps ymm1, ymm1, ymm3
        vmovups [rax], ymm1

        dec rdi
    jnz ._sm_loop

```

._loop_straight_mul_zeroed:

```
inc r10
cmp r10, rcx
jnz ._loop_straight_mul

    lea r10, [r9+1]
    mov rax, r8
    xor rdx, rdx
    mul r10
    lea r11, [rsi+rax*4]
    pxor xmm4, xmm4
    movss xmm0, [r11+r10*4]
    cmpss xmm4, xmm0, 0
    movd edx, xmm4
    and edx, edx
    jnz ._diag_next_zero_straight
    ._fixed_kinda:
```

```
inc r9
cmp r9, rcx
jnz ._loop_straight
```

._mul_diag:

```
pop rdi
mov r9, 1
fld dword [rsi] ; st0 - a11
._mul_diag_loop:
    mov rax, r8
    mul r9
    lea rax, [rsi+rax*4]
    fld dword [rax+r9*4] ; st0 - a11, st1 - a11
    fmulp st1, st0
    inc r9
    cmp r9, rcx
    jnz ._mul_diag_loop
```

dw 0x3fdb ; NASM is kinda dumb. Should be the same as fstp

oword [rdi]

```
xor rax, rax
```

```
ret
```

._diag_next_zero_straight: ; r10 - row number with zero

```
lea r12, [r10+1]
```

._diag_next_zero_straight_loop:

```
mov rax, r8
xor rdx, rdx
mul r12
```



```

        lea rax, [rsi+rax*4]

        pxor xmm4, xmm4
        movss xmm0, [rax+r10*4]
        cmpss xmm4, xmm0, 0
        movd edx, xmm4
        and edx, edx
        jz  ._diag_next_zero_straight_loop_add
        inc r12
        cmp r12, rcx
jnz  ._diag_next_zero_straight_loop
jmp  ._fixed_kinda

._diag_next_zero_straight_loop_add:
        mov rdx, r8

        shl rdx, 2
        add rax, rdx
        add r11, rdx
        shr rdx, 5

        ._diag_next_zero_straight_loop_add_sm_loop:
                sub rax, 32
                sub r11, 32
                vmovups ymm0, [r11]
                vmovups ymm1, [rax]
                vaddps ymm0, ymm0, ymm1
                vmovups [r11], ymm0
                dec rdx
        jnz  ._diag_next_zero_straight_loop_add
jmp  ._fixed_kinda

```

Также прикладываю ссылку на облачный репозиторий с исходным кодом программы: https://github.com/adivanced/Gauss_algorithm_in_C_and_assembly

4. Примеры работы.

```
advanced@advanced-Nitro-ANS15-55:~/Desktop/itmo/4s/vichmat/1$ ./main
Select the input type(f/t(f for file, t for terminal)):f

Input the file name:equ.txt

Initial matrix
0.000000 228.337006 123.211998 229.220001 3.000000 0.000000 0.000000 0.000000
0.000000 123.209999 12.200000 12.300000 4.000000 0.000000 0.000000 0.000000
22.200001 1234.123047 12345.120117 11.110000 5.000000 0.000000 0.000000 0.000000
124.230003 234.123001 124.230003 0.000000 6.000000 0.000000 0.000000 0.000000

Matrix after zero-elimination
22.200001 1462.460083 12468.332031 240.330002 8.000000 0.000000 0.000000 0.000000
0.000000 123.209999 12.200000 12.300000 4.000000 0.000000 0.000000 0.000000
22.200001 1234.123047 12345.120117 11.110000 5.000000 0.000000 0.000000 0.000000
146.430008 1696.583130 12592.562500 240.330002 14.000000 0.000000 0.000000 0.000000


Matrix after Gauss
1.000000 0.000000 0.000000 0.000000 -0.014169 0.000000 0.000000 0.000000
0.000000 1.000000 0.000000 0.000000 0.034751 0.000000 0.000000 0.000000
-0.000000 -0.000000 1.000000 -0.000000 -0.003026 -0.000000 -0.000000 -0.000000
0.000000 0.000000 0.000000 1.000000 -0.019903 0.000000 0.000000 0.000000

x1=-0.014169
x2=0.034751
x3=-0.003026
x4=-0.019903

r1=0.000000
r2=0.000000
r3=0.000003
r4=0.000004

determinant: -38728934402.398518
```

Проверка:

 **Определитель (детерминант) матрицы**




Матрица

0.0 228.337 123.212 229.22
0.00 123.21 12.2 12.3
22.2 1234.123 12345.12 11.11
124.23 234.123 124.23 0

Точность вычисления
Знаков после запятой: 5

РАССЧИТАТЬ

Определитель (детерминант) матрицы
-38728932041.86129

 ССЫЛКА  СОХРАНИТЬ  ВИДЖЕТ



Решение системы линейных уравнений методом Гаусса

СЛАУ в матричном виде

```
0.0 228.337 123.212 229.22 3
0.00 123.21 12.2 12.3 4
22.2 1234.123 12345.12 11.11 5
124.23 234.123 124.23 0 6
```

Точность вычисления

Знаков после запятой: 6

РАССЧИТАТЬ

Количество решений

1

Вектор решения системы уравнений

| -0.014169 0.034751 -0.003026 -0.019903 |

Выводы:

При выполнении данной лабораторной работы, я научился реализовывать алгоритм Гаусса с помощью программирования. Обычно метод Гаусса является наиболее оптимальным для использования. Так как итерационные методы являются трудоёмкими. Метод Гаусса с выбором главного элемента также будет достаточно трудоёмким ввиду необходимости перестановки строк. Моя конкретная реализация алгоритма Гаусса использует потоковое расширение процессоров intel, AVX, что ускоряет вычисления. Также для вычисления дискриминанта используется x87 FPU, оперирующий 80-битными числами с плавающей точкой, что даёт дополнительную точность при вычислении дискриминанта.