Управляющие структуры

Гань Чжаолун

29 ноября, 2024, Москва, Россия

Российский Университет Дружбы Народов

Цели и задачи работы

Цель лабораторной работы

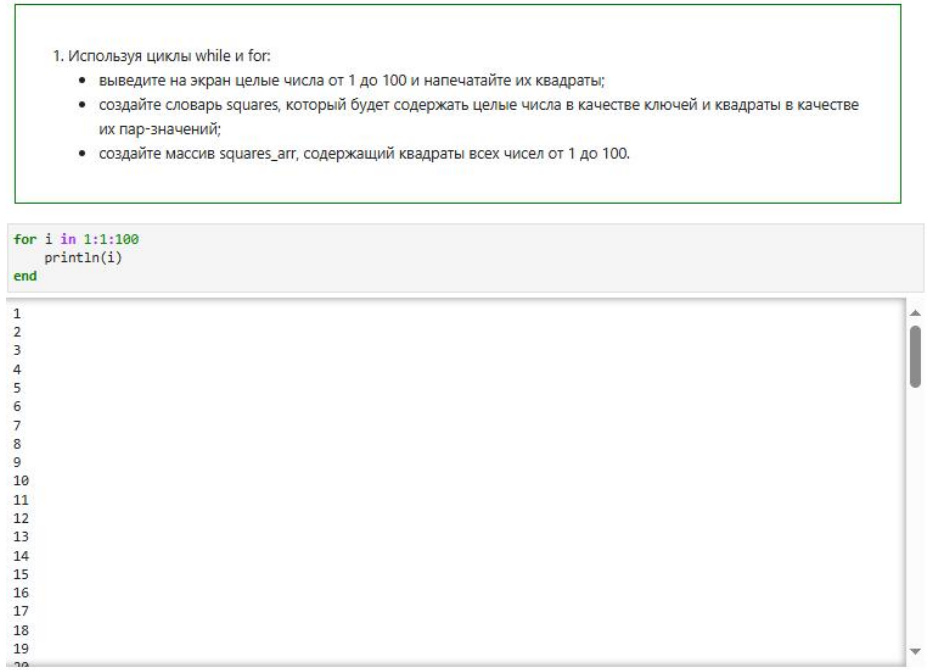
Основная цель работы — освоить применение циклов, функций и сторонних для Julia пакетов для решения задач линейной алгебры и работы с матрицами.

Процесс выполнения лабораторной работы

Используя Jupyter Lab, повторите примеры из раздела 3.2

Я повторю все задание 3.2 целиком

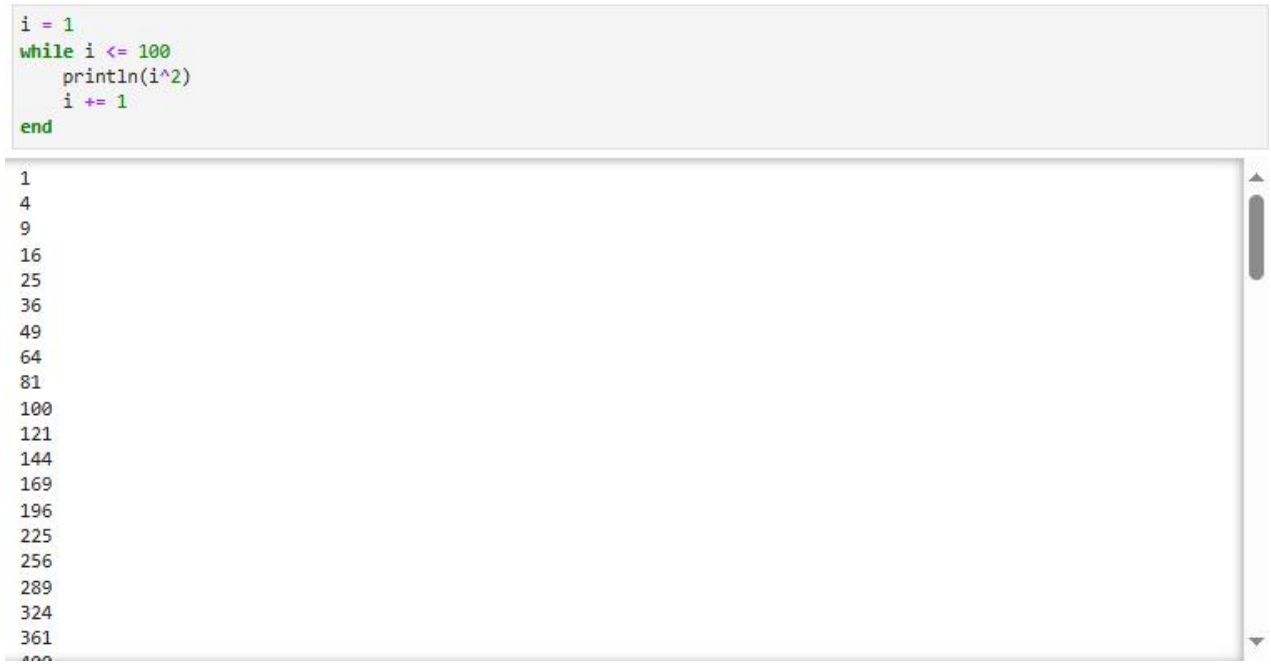
Выполните задания для самостоятельной работы



Сначала я вывел числа от 1 до 100, применив цикл `for`. Это позволило последовательно пройти все значения от 1 до 100 и напечатать их, что является простым и эффективным способом организации итераций по фиксированному диапазону.

Рисунок 1. Код и результат Задания 1-1

Выполните задания для самостоятельной работы



```
i = 1
while i <= 100
    println(i^2)
    i += 1
end
```

```
1
4
9
16
25
36
49
64
81
100
121
144
169
196
225
256
289
324
361
```

Сначала я вывел числа от 1 до 100, применив цикл `for`. Это позволило последовательно пройти все значения от 1 до 100 и напечатать их, что является простым и эффективным способом организации итераций по фиксированному диапазону.

Рисунок 2. Код и результат Задания 1-2

Выполните задания для самостоятельной работы

```julia
# инициализируем словарь
squares = Dict{Int64, Int64}()
# занесем значения в словарь
for i in 1:1:100
    push!(squares, i => i^2)
end
pairs(squares)
```

```
32 => 1024
6  => 36
67 => 4489
45 => 2025
73 => 5329
64 => 4096
90 => 8100
4  => 16
13 => 169
54 => 2916
63 => 3969
86 => 7396
91 => 8281
62 => 3844
58 => 3364
52 => 2704
12 => 144
28 => 784
75 => 5625
⋮  => ⋮
```

Я создал словарь `squares`, используя цикл `for`, чтобы добавить в словарь числа в качестве ключей и их квадраты как значения. В итоге словарь `squares` содержал все числа от 1 до 100 и соответствующие им квадраты, что позволило иметь прямую связь между числом и его квадратом для дальнейшего использования.

Рисунок 3. Код и результат Задания 1-3

Выполните задания для самостоятельной работы

```
# создадим массив из целых чисел от 1 до 100
numbers = []
for i in 1:1:100
    append!(numbers, i)
end

# возведем в квадрат кажое число из массива numbers
squares_arr = []
i = 1
while i <= length(numbers)
    append!(squares_arr, numbers[i]^2)
    i += 1
end
squares_arr
```

```
100-element Vector{Any}:
     1
     4
     9
    16
    25
    36
    49
    64
    81
   100
   121
   144
   169
     ⋮
  7921
  8100
  8281
  8464
```

Наконец, я создал массив `squares_arr`, содержащий квадраты чисел от 1 до 100. Сначала я инициализировал массив чисел от 1 до 100, а затем с помощью цикла `while` возводил каждое число в квадрат и добавлял в новый массив `squares_arr`. В результате массив `squares_arr` включал все квадраты чисел, что упрощает доступ к ним при необходимости.

Рисунок 4. Код и результат Задания 1-4

Выполните задания для самостоятельной работы

2. Напишите условный оператор, который печатает число, если число чётное, и строку «нечётное», если число нечётное. Перепишите код, используя тернарный оператор.

```
N = 5
if N % 2 == 0
    println("Четное число")
else
    println("Нечетное число")
end
```

```
Нечетное число
```

```
N = 4
(N % 2 == 0) ? println("Четное число") : println("Нечетное число")
```

```
Четное число
```

В первом шаге я написал условный оператор `if-else` для проверки четности числа. Здесь я проверяю, делится ли число `N` на 2 без остатка. Если остаток равен нулю (`N % 2 == 0`), то программа выводит `"Четное число"`, иначе — `"Нечетное число"`.
На втором шаге я переписал тот же условный оператор, используя тернарный оператор, Тернарный оператор в Julia имеет следующую структуру: `(условие) ? значение_если_истинно : значение_если_ложно`. Здесь я проверяю, является ли число `N` четным, и в зависимости от результата либо вывожу `"Четное число"`, либо `"Нечетное число"`.

Рисунок 5. Код и результат Задания 2

Выполните задания для самостоятельной работы

3. Напишите функцию add_one, которая добавляет 1 к своему входу.

```
function add_one(X)
    X + 1
end
add_one(5)
```

```
6
```

- Я определил функцию `add_one`, которая принимает один аргумент `X`.

- В теле функции я добавил `1` к значению `X` и возвращаю результат. Поскольку Julia автоматически возвращает последнее вычисленное выражение в функции, явное использование `return` не требуется.

- Я вызвал функцию `add_one(5)`, чтобы протестировать ее работу. В данном случае значение `5` передается в функцию как аргумент `X`.

- Вызов функции `add_one(5)` возвращает значение `6`, так как к аргументу `5` добавляется `1`.

Рисунок 6. Код и результат Задания 3

Выполните задания для самостоятельной работы

4. Используйте map() или broadcast() для задания матрицы A, каждый элемент которой увеличивается на единицу по сравнению с предыдущим.

```
X = fill(1, 4 * 4)
Y = collect(0:(length(X) - 1))
X = reshape(map(+, X, Y), (4, 4))
```

```
4×4 Matrix{Int64}:
 1  5   9  13
 2  6  10  14
 3  7  11  15
 4  8  12  16
```

- `X = fill(1, 4 * 4)`: Я создал одномерный массив из 16 элементов, каждый из которых равен `1`.

- `Y = collect(0:(length(X) - 1))`: Я создал массив `Y`, который содержит значения от `0` до `15`. Эти значения будут добавляться к каждому элементу матрицы `X`, чтобы получить последовательность, увеличивающуюся на единицу с каждым шагом.

- `X = reshape(..., (4, 4))`: Я преобразовал полученный одномерный массив в матрицу размером `4x4`. В итоге получена матрица размером `4x4`, в которой каждый элемент увеличивается на единицу по сравнению с предыдущим.

Рисунок 7. Код и результат Задания 4

Выполните задания для самостоятельной работы

5. Задайте матрицу A

```
A = [1 1 3; 5 2 6; -2 -1 -3]
```

```
3×3 Matrix{Int64}:
  1   1   3
  5   2   6
 -2  -1  -3
```

```
# возведение в степень 3
A^3
```

```
3×3 Matrix{Int64}:
 0  0  0
 0  0  0
 0  0  0
```

```
# заменим 3 столбец матрицы A на сумму второго и третьего столбца
for i in 7:1:9
    A[i] += A[i-3]
end
A
```

```
3×3 Matrix{Int64}:
  1   1   4
  5   2   8
 -2  -1  -4
```

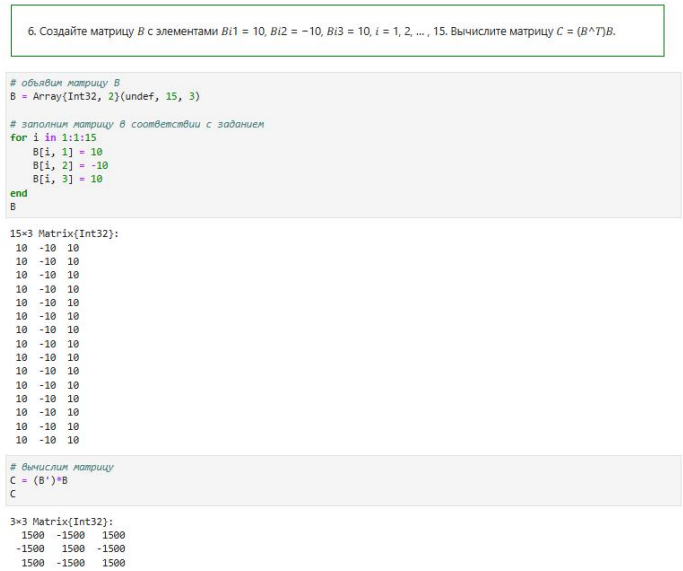
- создал матрицу `A` размером `3x3`, используя синтаксис в Julia: `A = [1 1 3; 5 2 6; -2 -1 -3]`.

- применил оператор `^` для возведения матрицы в степень 3 (`A^3`), что означает умножение матрицы на саму себя трижды.

- заменил третий столбец матрицы `A` на сумму второго и третьего столбцов с использованием цикла `for`:
  - В цикле `for i in 7:1:9` я перебираю индексы элементов третьего столбца, начиная с 7-го элемента и заканчивая 9-м.
  - Каждому элементу третьего столбца (`A[i]`) я добавляю соответствующий элемент второго столбца (`A[i-3]`), чтобы получить нужный результат.

Рисунок 8. Код и результат Задания 5

Выполните задания для самостоятельной работы



6. Создайте матрицу B с элементами $Bi1 = 10$, $Bi2 = -10$, $Bi3 = 10$, $i = 1, 2, \ldots, 15$. Вычислите матрицу $C = (B^T)B$.

```
# объявим матрицу B
B = Array{Int32, 2}(undef, 15, 3)

# заполним матрицу в соответствии с заданием
for i in 1:1:15
    B[i, 1] = 10
    B[i, 2] = -10
    B[i, 3] = 10
end
B
```

```
15×3 Matrix{Int32}:
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
 10  -10  10
```

```
# вычислим матрицу
C = (B')*B
C
```

```
3×3 Matrix{Int32}:
  1500  -1500   1500
 -1500   1500  -1500
  1500  -1500   1500
```

- Я объявил матрицу `B` размером `15x3` с использованием `Array{Int32, 2}(undef, 15, 3)`, что означает создание двумерного массива типа `Int32` с размерами `15` строк и `3` столбца.
Использование `undef` позволяет сначала выделить память для массива, а затем заполнить его необходимыми значениями.

- Я использовал цикл `for i in 1:1:15` для перебора всех строк матрицы.
Заполнил все элементы в каждой строке значениями: `10`, `-10` и `10` для первого, второго и третьего столбцов соответственно.

- Я вычислил матрицу `C` как произведение транспонированной матрицы `B` и самой матрицы `B`: `C = (B') * B`.
Здесь `B'` обозначает транспонирование матрицы `B`, а затем выполняется матричное умножение.

Рисунок 9. Код и результат Задания 6

Выполните задания для самостоятельной работы

7. Создайте матрицу Z размерности 6 × 6, все элементы которой равны нулю, и матрицу E, все элементы которой равны 1. Используя цикл while или for и закономерности расположения элементов, создайте следующие матрицы размерности 6 × 6.

```
z = zeros(Int64, 6, 6)
```

```
6×6 Matrix{Int64}:
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
```

```
E = ones(Int64, 6, 6)
```

```
6×6 Matrix{Int64}:
 1  1  1  1  1  1
 1  1  1  1  1  1
 1  1  1  1  1  1
 1  1  1  1  1  1
 1  1  1  1  1  1
 1  1  1  1  1  1
```

Рисунок 10. Код и результат Задания 7-1

Выполните задания для самостоятельной работы

```
z1 = zeros(Int64, 6, 6)
for i in 1:1:6
    if i != 1
        z1[i, i - 1] = E[i, i - 1]
    end
    if i != 6
        z1[i, i + 1] = E[i, i + 1]
    end
end
z1
```

```
6×6 Matrix{Int64}:
 0  1  0  0  0  0
 1  0  1  0  0  0
 0  1  0  1  0  0
 0  0  1  0  1  0
 0  0  0  1  0  1
 0  0  0  0  1  0
```

```
z2 = zeros(Int64, 6, 6)
for i in 1:1:6
    z2[i,i] = 1
    if(i+2 <= 6) z2[i,i + 2] = E[i, i + 2] end
    if(i-2 >= 1) z2[i, i - 2] = E[i, i - 2] end
end
z2
```

```
6×6 Matrix{Int64}:
 1  0  1  0  0  0
 0  1  0  1  0  0
 1  0  1  0  1  0
 0  1  0  1  0  1
 0  0  1  0  1  0
 0  0  0  1  0  1
```

- z1: Создана с элементами равными `1`, расположенными по соседству с главной диагональю. Цикл добавляет единицы в элементы, которые находятся рядом с главной диагональю.

- z2: Создана с элементами главной диагонали равными `1`, а также с добавлением единиц через две позиции от главной диагонали.

Рисунок 11. Код и результат Задания 7-2

Выполните задания для самостоятельной работы

```
z3 = zeros(Int64, 6, 6)
for i in 1:1:6
    z3[i,7-i] = 1
    if((7-i+2) <= 6) z3[i,9 - i] = E[i,9 - i] end
    if((7-i-2) >= 1) z3[i, 5 - i] = E[i, 5 - i] end
end
z3
```

```
6×6 Matrix{Int64}:
 0  0  0  1  0  1
 0  0  1  0  1  0
 0  1  0  1  0  1
 1  0  1  0  1  0
 0  1  0  1  0  0
 1  0  1  0  0  0
```

```
z4 = zeros(Int64, 6, 6)
for i in 1:1:6
    z4[i,i] = 1
    if(i+2 <= 6) z4[i, i + 2] = E[i, i + 2] end
    if(i-2 >= 1) z4[i, i - 2] = E[i, i - 2] end
    if(i+4 <= 6) z4[i, i + 4] = E[i, i + 4] end
    if(i-4 >= 1) z4[i, i - 4] = E[i, i - 4] end
end
z4
```

```
6×6 Matrix{Int64}:
 1  0  1  0  1  0
 0  1  0  1  0  1
 1  0  1  0  1  0
 0  1  0  1  0  1
 1  0  1  0  1  0
 0  1  0  1  0  1
```

- z3: Создана с зеркальным отражением элементов относительно главной диагонали и с дополнительными значениями, отстоящими на две позиции.

- z4: Создана с элементами главной диагонали, а также с добавлением единиц через две и четыре позиции от диагонали. Это обеспечивает равномерное распределение `1` вдоль диагонали и по сторонам.

Рисунок 12. Код и результат Задания 7-3

Выполните задания для самостоятельной работы

8. Напишите свою функцию, аналогичную функции outer() языка R. Функция должна иметь следующий интерфейс: outer(x,y,operation).

```
function outer(x, y, operation)
    if (ndims(x) == 1) x = reshape(x, (size(x, 1), size(x, 2))) end
    if (ndims(y) == 1) y = reshape(y, (size(y, 1), size(y, 2))) end
    c = zeros(size(x)[1], size(y)[2])
    for i in 1:size(x)[1], j in 1:size(y)[2], k in 1:size(x)[2]
        c[i, j] += operation(x[i, k], y[k, j])
    end
    return c
end
```

```
outer (generic function with 1 method)
```

```
# проверим на данной матрице работу функции
X = [[1 1 3]; [5 2 6]; [-2 -1 -3]]
```

```
3×3 Matrix{Int64}:
  1   1   3
  5   2   6
 -2  -1  -3
```

```
X * X
```

```
3×3 Matrix{Int64}:
  0   0   0
  3   3   9
 -1  -1  -3
```

```
outer(X, X, *)
```

```
3×3 Matrix{Float64}:
  0.0   0.0   0.0
  3.0   3.0   9.0
 -1.0  -1.0  -3.0
```

1. Создание функции `outer`:
   - Функция `outer` принимает три аргумента: `x`, `y`, и `operation`.
   - Сначала проверяется, являются ли входные данные векторами (`ndims(x) == 1`). Если это так, то вектора преобразуются в матрицы с использованием функции `reshape`.
   - Создается пустая результирующая матрица `c` размером `(size(x)[1], size(y)[2])`, которая будет содержать результаты.
   - Далее используется вложенный цикл `for`, который проходит по всем элементам и выполняет операцию, переданную в аргументе `operation`.

2. Проверка работы функции:
   - В качестве проверки, я использовал функцию `outer` для произведения матрицы `X` самой на себя и убедился, что результат эквивалентен обычному матричному умножению (`X * X`).

Рисунок 13. Код и результат Задания 8-1

Выполните задания для самостоятельной работы

```
# Построим первую матрицу
A1 = outer(collect(0:4), collect(0:4)', +)
```

```
5×5 Matrix{Float64}:
 0.0  1.0  2.0  3.0  4.0
 1.0  2.0  3.0  4.0  5.0
 2.0  3.0  4.0  5.0  6.0
 3.0  4.0  5.0  6.0  7.0
 4.0  5.0  6.0  7.0  8.0
```

```
# построим вторую матрицу
A2 = outer(collect(0:4), collect(1:5)', ^)
```

```
5×5 Matrix{Float64}:
 0.0   0.0   0.0    0.0     0.0
 1.0   1.0   1.0    1.0     1.0
 2.0   4.0   8.0   16.0    32.0
 3.0   9.0  27.0   81.0   243.0
 4.0  16.0  64.0  256.0  1024.0
```

```
# построим третью матрицу
# возьмем остаток от деления на 5
A3 = outer(collect(0:4), collect(0:4)', +).%5
```

```
5×5 Matrix{Float64}:
 0.0  1.0  2.0  3.0  4.0
 1.0  2.0  3.0  4.0  0.0
 2.0  3.0  4.0  0.0  1.0
 3.0  4.0  0.0  1.0  2.0
 4.0  0.0  1.0  2.0  3.0
```

- A1: Матрица, созданная на основе векторов `0:4` и операции сложения. Эта операция фактически создает таблицу сложения для чисел от 0 до 4.
- A2: Матрица, созданная с использованием возведения в степень для элементов векторов `0:4` и `1:5`.
- A3: Сложение элементов с последующим взятием остатка от деления на `5`. Это создает матрицу, где значения периодически повторяются с шагом `5`.

Рисунок 14. Код и результат Задания 8-2

Выполните задания для самостоятельной работы

```
# построим четвертую матрицу
# возьмем остаток от деления на 10
A4 = outer(collect(0:9), collect(0: 9)', +).%10
```

```
10×10 Matrix{Float64}:
 0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0
 1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0  0.0
 2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0  0.0  1.0
 3.0  4.0  5.0  6.0  7.0  8.0  9.0  0.0  1.0  2.0
 4.0  5.0  6.0  7.0  8.0  9.0  0.0  1.0  2.0  3.0
 5.0  6.0  7.0  8.0  9.0  0.0  1.0  2.0  3.0  4.0
 6.0  7.0  8.0  9.0  0.0  1.0  2.0  3.0  4.0  5.0
 7.0  8.0  9.0  0.0  1.0  2.0  3.0  4.0  5.0  6.0
 8.0  9.0  0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0
 9.0  0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0
```

```
# построим пятую матрицу
# возьмем остаток от деления на 9
A5 = outer(collect(0:8), collect(-9:-1)', -).%9
```

```
9×9 Matrix{Float64}:
 0.0  8.0  7.0  6.0  5.0  4.0  3.0  2.0  1.0
 1.0  0.0  8.0  7.0  6.0  5.0  4.0  3.0  2.0
 2.0  1.0  0.0  8.0  7.0  6.0  5.0  4.0  3.0
 3.0  2.0  1.0  0.0  8.0  7.0  6.0  5.0  4.0
 4.0  3.0  2.0  1.0  0.0  8.0  7.0  6.0  5.0
 5.0  4.0  3.0  2.0  1.0  0.0  8.0  7.0  6.0
 6.0  5.0  4.0  3.0  2.0  1.0  0.0  8.0  7.0
 7.0  6.0  5.0  4.0  3.0  2.0  1.0  0.0  8.0
 8.0  7.0  6.0  5.0  4.0  3.0  2.0  1.0  0.0
```

- A4: Матрица размером `10x10`, элементы которой — это сумма чисел с последующим взятием остатка от деления на `10`.
- A5: Используется операция вычитания, затем берется остаток от деления на `9` между значениями из диапазонов `0:8` и `-9:-1`.

Рисунок 15. Код и результат Задания 8-3

Выполните задания для самостоятельной работы

9. Решите следующую систему линейных уравнений с 5 неизвестными.

```
# зададим коэффициенты из системы уравнений
array_system = Array{Int64, 2}(undef, 5, 5)
m = 5
n = 5

for i in 1:1:m
    for j in 1:1:n
        array_system[i, j] = 1 + abs(i - j)
    end
end
println(round.(Int32, array_system))


# обозначим ответы
answers = [7; -1; -3; 5; 17]
```

```
Int32[1 2 3 4 5; 2 1 2 3 4; 3 2 1 2 3; 4 3 2 1 2; 5 4 3 2 1]
```

```
5-element Vector{Int64}:
  7
 -1
 -3
  5
 17
```

```
# с помощью обратной матрицы получим решение данной матрицы
array_system_inv = inv(array_system)
round.(Int, array_system_inv)
x_n = round.(Int, array_system_inv * answers)
```

```
5-element Vector{Int64}:
 -2
  3
  5
  2
 -4
```

1. Задание коэффициентов матрицы `A`:
   - Я задал матрицу `array_system` размером `5x5`, используя цикл `for`. Каждый элемент `A[i, j]` был вычислен как `1 + abs(i - j)`, что создает матрицу со специфической симметричной структурой.
   - Использование цикла позволяет легко обобщить решение на матрицы большей размерности (например, `nxn`), сохраняя ту же закономерность при формировании элементов.

2. Задание вектора ответов:
   - Вектор ответов (`answers`) представляет собой значения правой части системы уравнений: `[7, -1, -3, 5, 17]`.

3. Решение системы с помощью обратной матрицы:
   - Я вычислил обратную матрицу `array_system_inv` с помощью функции `inv()`.
   - Далее для нахождения решения `x` я умножил обратную матрицу на вектор ответов (`x_n = array_system_inv * answers`).
   - Значения были округлены с использованием функции `round.(Int, ...)`, чтобы получить целые числа в результате.

Рисунок 16. Код и результат Задания 9

Выполните задания для самостоятельной работы

10. Создайте матрицу M размерности 6 × 10, элементами которой являются целые числа, выбранные случайным образом с повторениями из совокупности 1, 2, ... , 10.

```
M = rand(1:10, 6, 10)
```

```
6×10 Matrix{Int64}:
 4   1  10  7  4   7  2  3  3  8
 2  10   3  3  3   8  3  3  3  6
 4   3   4  4  7   2  6  2  1  7
 6   6   3  2  9   5  1  3  7  2
 3   2   3  7  6   8  4  8  1  5
 5   7   5  5  9  10  9  4  7  4
```

- Найдите число элементов в каждой строке матрицы M, которые больше числа N (например, N = 4).

```
N = 6
for i in 1:1:6
    count = 0
    for j in 1:1:10
        if M[i, j] > N
            count += 1
        end
    end
    println("Количество элементов больше ", N, " в строке ", i, " ровно ", count)
end
```

```
Количество элементов больше 6 в строке 1 ровно 4
Количество элементов больше 6 в строке 2 ровно 2
Количество элементов больше 6 в строке 3 ровно 2
Количество элементов больше 6 в строке 4 ровно 2
Количество элементов больше 6 в строке 5 ровно 3
Количество элементов больше 6 в строке 6 ровно 5
```

1. Создание матрицы `M`:
   - Я создал матрицу `M` размером `6x10`, заполненную случайными целыми числами от `1` до `10`, используя функцию `rand(1:10, 6, 10)`.

2. Определение количества элементов больше `N` в каждой строке:
   - Использовал цикл `for`, чтобы пройти по каждой строке матрицы и посчитать количество элементов, которые больше заданного числа `N` (в данном случае `N = 6`).
   - Результат выводится для каждой строки, показывая количество элементов, которые превышают `N`.

Рисунок 17. Код и результат Задания 10-1

Выполните задания для самостоятельной работы

- Определите, в каких строках матрицы M число M (например, M = 7) встречается ровно 2 раза?

```
M_ = 7

for i in 1:1:6
    count = 0
    for j in 1:1:10
        if M[i, j] == M_
            count += 1
        end
    end
    if count == 2
        println("Число ", M_, " встречается в строке ", i, " ровно 2 раза")
    end
end
```

```
Число 7 встречается в строке 1 ровно 2 раза
Число 7 встречается в строке 3 ровно 2 раза
Число 7 встречается в строке 6 ровно 2 раза
```

- Определите все пары столбцов матрицы M, сумма элементов которых больше K (например, K = 75).

```
function sum_75(matrix, K)
    matrix_new = rand(10)
    for i in 1:1:10
        sum = 0
        for j in 1:1:6
            sum += matrix[j, i]
        end
        matrix_new[i] = sum
    end
    # теперь просуммируем все значения matrix_new попарно
    for i in 1:1:10
        for j in i+1:1:10
            sum = matrix_new[i] + matrix_new[j]
            if sum > K
                println("Столбцы - ", i, " и ", j)
            end
        end
    end
end
```

```
sum_75 (generic function with 1 method)
```

```
sum_75(M, 75)
```

```
Столбцы - 5 и 6
```

3. Определение строк, где число `M_` встречается ровно два раза:
   - В следующем цикле я посчитал, сколько раз число `M_` (в данном случае `M_ = 7`) встречается в каждой строке матрицы `M`.
   - Если число `7` встречается ровно два раза, выводится номер этой строки.

4. Определение пар столбцов, сумма элементов которых больше `K`:
   - Я создал функцию `sum_75(matrix, K)`, которая сначала вычисляет сумму элементов каждого столбца матрицы.
   - Затем эти суммы попарно суммируются, и проверяется, превышает ли эта сумма заданное значение `K` (в данном случае `K = 75`).
   - Выводятся все пары столбцов, сумма которых больше `K`.

Рисунок 18. Код и результат Задания 10-2

Выполните задания для самостоятельной работы

11. Вычислите:

$$\sum_{i=1}^{20} \sum_{j=1}^{5} \frac{i^4}{3+j}$$

$$\sum_{i=1}^{20} \sum_{j=1}^{5} \frac{i^4}{3+ij}$$

```
first_sum = 0
for i in 1:1:20
    for j in 1:1:5
        first_sum += (i^4)/(3+j)
    end
end
println(first_sum)
```

```
639215.2833333334
```

```
second_sum = 0
for i in 1:1:20
    for j in 1:1:5
        second_sum += (i^4)/(3+i*j)
    end
end
println(second_sum)
```

```
89912.02146097136
```

1. Первая сумма:
   - Я использовал два вложенных цикла для вычисления этой суммы.
   - Внешний цикл проходит по значениям `i` от `1` до `20`, а внутренний — по значениям `j` от `1` до `5`.
   - Для каждой комбинации `i` и `j` вычисляется дробь `(i^4) / (3 + j)` и добавляется к переменной `first_sum`.

2. Вторая сумма:
   - Аналогично первой сумме, я использовал вложенные циклы для вычисления.
   - Для каждой комбинации `i` и `j` я вычисляю дробь `(i^4) / (3 + i * j)` и добавляю к переменной `second_sum`.

Рисунок 19. Код и результат Задания 11

Выводы по проделанной работе

Вывод

Освоил применение циклов, функций и сторонних для Julia пакетов для решения задач линейной алгебры и работы с матрицами.