

Лабораторная работа №3

Управляющие структуры

Клюкин Михаил Александрович

Содержание

1	Цель работы	6
2	Задание	7
3	Выполнение лабораторной работы	8
3.1	Циклы while и for	8
3.2	Условные выражения	15
3.3	Функции	17
3.4	Сторонние библиотеки в Julia	25
4	Задания для самостоятельного выполнения	28
5	Выводы	54

Список иллюстраций

3.1	Пример использования цикла <code>while</code> для формирования элементов массива	9
3.2	Пример использования цикла <code>while</code> для работы со строковыми элементами массива	10
3.3	Пример использования цикла <code>for</code> для формирования элементов массива	11
3.4	Пример использования цикла <code>for</code> для работы со строковыми элементами массива	12
3.5	Пример использования цикла <code>for</code> для создания двумерного массива	13
3.6	Пример использования цикла <code>for</code> для создания двумерного массива	14
3.7	Пример использования цикла <code>for</code> для создания двумерного массива	14
3.8	Пример использования условного выражения	16
3.9	Пример использования тернарного оператора	17
3.10	Пример написания функций с помощью ключевых слов	18
3.11	Пример написания функций в одной строке	19
3.12	Пример изменения аргумента функции	20
3.13	Пример вызова функции <code>map</code>	21
3.14	Пример вызова функции <code>broadcast</code>	22
3.15	Пример возведения матрицы в квадрат	23
3.16	Пример вызова функции <code>broadcast</code>	24
3.17	Пример записи математического выражения через функцию <code>bbroadcast</code>	25
3.18	Пример использования пакетов	27
4.1	Вывод чисел и их квадратов с помощью цикла <code>while</code>	29
4.2	Вывод чисел и их квадратов с помощью цикла <code>for</code>	30
4.3	Создание словаря	31
4.4	Создание массива	31
4.5	Пример условного оператора	32
4.6	Пример тернарного оператора	32
4.7	Пример функции	33
4.8	Пример использования <code>map</code>	34
4.9	Пример возведения матрицы в степень	34
4.10	Пример замены столбца в матрице	35
4.11	Создание матрицы B	36
4.12	Вычисление матрицы $B^T B$	37
4.13	Создание матриц Z , E , $Z1$	38

4.14 Создание матриц Z_2	40
4.15 Создание матриц Z_3	42
4.16 Создание матриц Z_4	44
4.17 Создание функции <code>outer</code>	45
4.18 Создание матрицы A_1	45
4.19 Создание матрицы A_2	46
4.20 Создание матрицы A_3	47
4.21 Создание матрицы A_4	47
4.22 Создание матрицы A_5	48
4.23 Решение системы линейных уравнений	49
4.24 Создание матрицы M	50
4.25 Поиск числа элементов в каждой строке матрицы M	51
4.26 Поиск строк с двумя числами M	51
4.27 Поиск столбцов матрицы M	52
4.28 Вычленение выражений	53

Список таблиц

1 Цель работы

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

2 Задание

1. Используя Jupyter Lab, повторите примеры из раздела 3.2.
2. Выполните задания для самостоятельной работы (раздел 3.4).

3 Выполнение лабораторной работы

3.1 Циклы while и for

Для различных операций, связанных с перебором индексируемых элементов структур данных, традиционно используются циклы `while` и `for`.

Синтаксис `while`

```
while <условие>  
    <тело цикла>  
end
```

Пример использования цикла `while` для формирования элементов массива (рис. 3.1).


```
# пока n<10 прибавить к n единицу и распечатать значение:  
n = 0  
while n < 10  
    n += 1  
    println(n)  
end
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Рис. 3.1: Пример использования цикла while для формирования элементов массива

Другой пример демонстрирует использование while при работе со строковыми элементами массива, подставляя имя из массива в заданную строку приветствия и выводя получившуюся конструкцию на экран (рис. 3.2).

```
myfriends = ["Ted", "Robyn", "Barney", "Lily", "Marshall"]
i = 1
while i <= length(myfriends)
    friend = myfriends[i]
    println("Hi $friend, it's great to see you!")
    i += 1
end
```

```
Hi Ted, it's great to see you!
Hi Robyn, it's great to see you!
Hi Barney, it's great to see you!
Hi Lily, it's great to see you!
Hi Marshall, it's great to see you!
```

Рис. 3.2: Пример использования цикла `while` для работы со строковыми элементами массива

Такие же результаты можно получить при использовании цикла `for`.

Синтаксис `for`:

```
for <переменная> in <диапазон>
    <тело цикла>
end
```

Рассмотренные выше примеры, но с использованием цикла `for` (рис. 3.3, 3.4).

```
for n in 1:2:10  
    println(n)  
end
```

1
3
5
7
9

```
for n in 1:1:10  
    println(n)  
end
```

1
2
3
4
5
6
7
8
9
10

Рис. 3.3: Пример использования цикла for для формирования элементов массива

```
myfriends = ["Ted", "Robyn", "Barney", "Lily", "Marshall"]  
for friend in myfriends  
    println("Hi $friend, it's great to see you!")  
end
```

```
Hi Ted, it's great to see you!  
Hi Robyn, it's great to see you!  
Hi Barney, it's great to see you!  
Hi Lily, it's great to see you!  
Hi Marshall, it's great to see you!
```

Рис. 3.4: Пример использования цикла `for` для работы со строковыми элементами массива

Пример использования цикла `for` для создания двумерного массива, в котором значение каждой записи является суммой индексов строки и столбца (рис. 3.5).

```

# инициализация массива m x n из нулей:
m, n = 5, 5
A = fill(0, (m, n))
# формирование массива, в котором значение каждой записи
# является суммой индексов строки и столбца:
for i in 1:m
    for j in 1:n
        A[i, j] = i + j
    end
end
A

```

5x5 Matrix{Int64}:

2	3	4	5	6
3	4	5	6	7
4	5	6	7	8
5	6	7	8	9
6	7	8	9	10

Рис. 3.5: Пример использования цикла for для создания двумерного массива

Создать двумерный массив также можно и другими способами (рис. 3.6, 3.7).

```
# инициализация массива m x n из нулей:
B = fill(0, (m, n))
for i in 1:m, j in 1:n
    B[i, j] = i + j
end
B
```

5×5 Matrix{Int64}:

2	3	4	5	6
3	4	5	6	7
4	5	6	7	8
5	6	7	8	9
6	7	8	9	10

Рис. 3.6: Пример использования цикла for для создания двумерного массива

```
C = [i + j for i in 1:m, j in 1:n]
C
```

5×5 Matrix{Int64}:

2	3	4	5	6
3	4	5	6	7
4	5	6	7	8
5	6	7	8	9
6	7	8	9	10

Рис. 3.7: Пример использования цикла for для создания двумерного массива

3.2 Условные выражения

Довольно часто при решении задач требуется проверить выполнение тех или иных условий. Для этого используют условные выражения.

Синтаксис условных выражений с ключевым словом:

```
if <условие 1>  
    <действие 1>  
elseif <условие 2>  
    <действие 2>  
else  
    <действие 3>  
end
```

Например, пусть для заданного числа N требуется вывести слово «Fizz», если N делится на 3, «Buzz», если N делится на 5, и «FizzBuzz», если N делится на 3 и 5 (рис. 3.8).

```

: # используем `&&` для реализации операции "AND"
  # операция % вычисляет остаток от деления
  N = 15
  if (N % 3 == 0) && (N % 5 == 0)
    println("FizzBuzz")
  elseif N % 3 == 0
    println("Fizz")
  elseif N % 5 == 0
    println("Buzz")
  else
    println(N)
  end
FizzBuzz

```

Рис. 3.8: Пример использования условного выражения

Синтаксис условных выражений с тернарными операторами:

$a ? b : c$

Такая запись эквивалентна записи условного выражения с ключевым словом:

```

if a
  b
else
  c
end

```

Пример использования тернарного оператора (рис. 3.9).


```
x = 5
y = 10
(x > y) ? x : y

10
```

Рис. 3.9: Пример использования тернарного оператора

3.3 Функции

Julia дает нам несколько разных способов написать функцию. Первый требует ключевых слов `function` и `end` (рис. 3.10).

```
function sayhi(name)
    println("Hi $name, it's great to see you!")
end
# функция возведения в квадрат:
function f(x)
    x^2
end
```

f (generic function with 1 method)

```
sayhi("C-3PO")
f(42)
```

Hi C-3PO, it's great to see you!

1764

Рис. 3.10: Пример написания функций с помощью ключевых слов

В качестве альтернативы, можно объявить любую из выше определённых функций в одной строке либо как анонимную (рис. 3.11).

```
sayhi2(name) = println("Hi $name, it's great to see you!")  
f2(x) = x^2
```

f2 (generic function with 1 method)

```
sayhi2("C-3P0")  
f2(42)
```

Hi C-3P0, it's great to see you!
1764

```
sayhi3 = name -> println("Hi $name, it's great to see you!")  
f3 = x -> x^2
```

#5 (generic function with 1 method)

```
sayhi3("C-3P0")  
f3(42)
```

Hi C-3P0, it's great to see you!
1764

Рис. 3.11: Пример написания функций в одной строке

По соглашению в Julia функции, сопровождаемые восклицательным знаком, изменяют свое содержимое, а функции без восклицательного знака не делают этого. Например, сравните результат применения `sort` и `sort!` (рис. 3.12).

```
# задаём массив v:  
v = [3, 5, 2]  
sort(v)  
v
```

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

```
sort!(v)  
v
```

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

```
v = [3, 5, 2]  
result = sort(v)  
result
```

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

Рис. 3.12: Пример изменения аргумента функции

В Julia функция `map` является функцией высшего порядка, которая принимает функцию в качестве одного из своих входных аргументов и применяет эту функцию к каждому элементу структуры данных, которая ей передаётся также в качестве аргумента (рис. 3.13).

```
map(f, [1, 2, 3])
```

```
3-element Vector{Int64}:
```

```
1
```

```
4
```

```
9
```

Рис. 3.13: Пример вызова функции `map`

Функция `broadcast` — ещё одна функция высшего порядка в Julia, представляющая собой обобщение функции `map`. Функция `broadcast()` будет пытаться привести все объекты к общему измерению, `map()` будет напрямую применять данную функцию поэлементно.

Синтаксис для вызова `broadcast` такой же, как и для вызова `map` (рис. 3.14).

```
f(x) = x^2  
broadcast(f, [1, 2, 3])
```

```
3-element Vector{Int64}:  
 1  
 4  
 9
```

Рис. 3.14: Пример вызова функции `broadcast`

Возведение матрицы в квадрат (рис. 3.15).

```
# Задаём матрицу A:  
A = [i + 3*j for j in 0:2, i in 1:3]  
A
```

```
3×3 Matrix{Int64}:
```

```
 1  2  3  
 4  5  6  
 7  8  9
```

```
f(A)
```

```
3×3 Matrix{Int64}:
```

```
30  36  42  
66  81  96  
102 126 150
```

Рис. 3.15: Пример возведения матрицы в квадрат

Функция `broadcast` применяет переданную функцию ко всем элементам матрицы (рис. 3.16).

```
broadcast(f, A)
```

```
3x3 Matrix{Int64}:
```

```
 1    4    9
16   25   36
49   64   81
```

Рис. 3.16: Пример вызова функции broadcast

Точечный синтаксис для `broadcast()` позволяет записать относительно сложные составные поэлементные выражения в форме, близкой к математической записи (рис. 3.17).


```
A .+ 2 .* f.(A) ./ A
```

```
3×3 Matrix{Float64}:  
 3.0  6.0  9.0  
12.0 15.0 18.0  
21.0 24.0 27.0
```

```
broadcast(x -> x + 2 * f(x) / x, A)
```

```
3×3 Matrix{Float64}:  
 3.0  6.0  9.0  
12.0 15.0 18.0  
21.0 24.0 27.0
```

Рис. 3.17: Пример записи математического выражения через функцию `bbroadcast`

3.4 Сторонние библиотеки в Julia

Julia имеет более 2000 зарегистрированных пакетов, что делает их огромной частью экосистемы Julia. Есть вызовы функций первого класса для других языков, обеспечивающие интерфейсы сторонних функций. Можно вызывать функции из Python или R, например, с помощью PyCall или Rcall.

При первом использовании пакета в вашей текущей установке Julia вам необходимо использовать менеджер пакетов, чтобы явно его добавить.

При каждом новом использовании Julia (например, в начале нового сеанса в

REPL или открытии блокнота в первый раз) нужно загрузить пакет, используя ключевое слово `using` (рис. 3.18).


```
: import Pkg
Pkg.add("Example")

Updating registry at `~/.julia/registries/General.toml`
Resolving package versions...
Installed Example - v0.5.5
Updating `~/.julia/environments/v1.11/Project.toml`
[7876af07] + Example v0.5.5
Updating `~/.julia/environments/v1.11/Manifest.toml`
[7876af07] + Example v0.5.5
Precompiling project...
1217.3 ms ✓ ConcreteStructs
1214.5 ms ✓ IteratorInterfaceExtensions
2063.4 ms ✓ SimpleUnPack
1218.2 ms ✓ ExprTools
1223.8 ms ✓ PositiveFactorizations
1228.7 ms ✓ MuladdMacro
1222.0 ms ✓ LaTeXStrings
1226.0 ms ✓ AbstractFFTs
1258.1 ms ✓ FunctionWrappers
665.7 ms ✓ Example

: Pkg.add("Colors")
using Colors

Resolving package versions...
Installed Colors - v0.13.1
Updating `~/.julia/environments/v1.11/Project.toml`
[5ae59095] + Colors v0.13.1
Updating `~/.julia/environments/v1.11/Manifest.toml`
[5ae59095] + Colors v0.13.0 → v0.13.1
Precompiling project...
4989.9 ms ✓ Colors
900.1 ms ✓ SparseMatrixColorings → SparseMatrixColoringsColorsExt
3355.4 ms ✓ ColorSchemes
10230.0 ms ✓ PlotUtils
3303.1 ms ✓ PlotThemes
4595.5 ms ✓ RecipesPipeline
65153.0 ms ✓ Plots
4107.8 ms ✓ Plots → UnitfulExt
5634.3 ms ✓ Plots → IJuliaExt
9 dependencies successfully precompiled in 95 seconds. 458 already precompiled.

: palette = distinguishable_colors(100)

:


: rand(palette, 3, 3)


:

```

Рис. 3.18: Пример использования пакетов

4 Задания для самостоятельного выполнения

Используя циклы `while` и `for`, выведем на экран целые числа от 1 до 100 и напечатаем их квадраты (рис. 4.1, 4.2).

```
n = 1
while n <= 100
    println(n, ", ", n^2)
    n += 1
end
```

```
1, 1
2, 4
3, 9
4, 16
5, 25
6, 36
7, 49
8, 64
9, 81
10, 100
11, 121
12, 144
13, 169
14, 196
15, 225
16, 256
17, 289
18, 324
19, 361
20, 400
21, 441
22, 484
```

Рис. 4.1: Вывод чисел и их квадратов с помощью цикла while

```
for n in 1:100
    println(n, ", ", n^2)
end
```

```
1, 1
2, 4
3, 9
4, 16
5, 25
6, 36
7, 49
8, 64
9, 81
10, 100
11, 121
12, 144
13, 169
14, 196
15, 225
16, 256
17, 289
18, 324
19, 361
20, 400
21, 441
22, 484
23, 529
24, 576
```

Рис. 4.2: Вывод чисел и их квадратов с помощью цикла for

Используя циклы while и for, создадим словарь squares, который будет содержать целые числа в качестве ключей и квадраты в качестве их пар-значений (рис. 4.3).

```
squares = Dict()
for n in 1:100
    squares[n] = n^2
end
println(squares)
```

```
Dict{Any, Any}(5 => 25, 56 => 3136, 35 => 1225, 55 => 3025, 60 => 3600, 30 => 900, 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, 23 => 529, 92 => 8464, 41 => 1681, 43 => 1849, 11 => 121, 36 => 1296, 68 => 4624, 69 => 4761, 98 => 9604, 82 => 6724, 85 => 7225, 39 => 1521, 84 => 7056, 77 => 5929, 7 => 49, 25 => 625, 95 => 9025, 71 => 5041, 66 => 4356, 76 => 5776, 34 => 1156, 50 => 2500, 59 => 3481, 93 => 8649, 2 => 4, 10 => 100, 18 => 324, 26 => 676, 27 => 729, 42 => 1764, 87 => 7569, 100 => 10000, 79 => 6241, 16 => 256, 20 => 400, 81 => 6561, 19 => 361, 49 => 2401, 44 => 1936, 9 => 81, 31 => 961, 74 => 5476, 61 => 3721, 29 => 841, 94 => 8836, 46 => 2116, 57 => 3249, 70 => 4900, 21 => 441, 38 => 1444, 88 => 7744, 78 => 6084, 72 => 5184, 24 => 576, 8 => 64, 17 => 289, 37 => 1369, 1 => 1, 53 => 2809, 22 => 484, 47 => 2209, 83 => 6889, 99 => 9801, 89 => 7921, 14 => 196, 3 => 9, 80 => 6400, 96 => 9216, 51 => 2601, 33 => 1089, 40 => 1600, 48 => 2304, 15 => 225, 65 => 4225, 97 => 9409)
```

Рис. 4.3: Создание словаря

Используя циклы `while` и `for`, создадим массив `squares_arr`, содержащий квадраты всех чисел от 1 до 100 (рис. 4.4).

```
squares_arr = [n^2 for n in 1:100]
println(squares_arr)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000]
```

Рис. 4.4: Создание массива

Напишем условный оператор, который печатает число, если число чётное, и строку «нечётное», если число нечётное (рис. 4.5).

```
n = 3
if n % 2 == 0
    println(n)
else
    println("Нечетное")
end
```

Нечетное

Рис. 4.5: Пример условного оператора

Перепишем код, используя тернарный оператор (рис. 4.6).

```
n = 4
println(n % 2 == 0 ? n : "Нечетное")
```

4

Рис. 4.6: Пример тернарного оператора

Напишем функцию `add_one`, которая добавляет 1 к своему входу (рис. 4.7).


```
function add_one(x)
    return x + 1
end
println(add_one(1))
```

2

Рис. 4.7: Пример функции

Используем `map()` для задания матрицы *A*, каждый элемент которой увеличивается на единицу по сравнению с предыдущим (рис. 4.8).

```
A = [1 2 3; 1 2 3; 1 2 3]
B = map(x -> x + 1, A)
B
```

```
3x3 Matrix{Int64}:
 2  3  4
 2  3  4
 2  3  4
```

Рис. 4.8: Пример использования map

Зададим матрицу A и найдем ее куб (рис. 4.9).

```
A = [1 1 3; 5 2 6; -2 -1 -3]
println(map(x -> x ^ 3, A))

[1 1 27; 125 8 216; -8 -1 -27]
```

Рис. 4.9: Пример возведения матрицы в степень

Заменяем третий столбец матрицы A на сумму второго и третьего столбцов (рис. 4.10).

```
A[:, 3] = A[:, 2] + A[:, 3]
A
```

```
3x3 Matrix{Int64}:
```

```
 1    1    4
 5    2    8
-2   -1   -4
```

Рис. 4.10: Пример замены столбца в матрице

Создадим матрицу B с элементами $B_{i1} = 1, B_{i2} = -10, B_{i3} = 10, i = 1, 2, \dots, 15$ (рис. 4.11).

```
B = repeat([10 -10 10], 15, 1)
```

15x3 Matrix{Int64}:

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

Рис. 4.11: Создание матрицы B

Вычислим матрицу $C = B^T B$ (рис. 4.12).

$$C = B^T * B$$

C

3x3 Matrix{Int64}:

1500	-1500	1500
-1500	1500	-1500
1500	-1500	1500

Рис. 4.12: Вычисление матрицы $B^T B$

Создайте матрицу Z размерности 6x6, все элементы которой равны нулю, и матрицу E, все элементы которой равны 1. Используя цикл for, создадим матрицу Z_1 (рис. 4.13).

```

Z = zeros(Int, 6, 6)
E = ones(Int, 6, 6)

n = 6
Z1 = copy(Z)
for i in 1:n
    for j in 1:n
        if abs(i - j) == 1
            Z1[i, j] = 1
        end
    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

Рис. 4.13: Создание матриц Z, E, Z1

Используя цикл `for`, создадим матрицу Z_2 (рис. 4.14).

```

Z2 = copy(Z)
for i in 1:n
    if i+2 < n
        Z2[i, i] = 1
        Z2[i, i+2] = 1
    elseif (i == 3) || (i == 4)
        Z2[i, i] = 1
        Z2[i, i+2] = 1
        Z2[i, i-2] = 1
    else
        Z2[i, i] = 1
        Z2[i, i-2] = 1
    end
end
Z2

```

6×6 Matrix{Int64}:

1	0	1	0	0	0
0	1	0	1	0	0
0	0	1	0	1	0
0	1	0	1	0	1
0	0	1	0	1	0
0	0	0	1	0	1

Рис. 4.14: Создание матриц Z2

Используя цикл `for`, создадим матрицу Z_3 (рис. 4.15).

```

Z3 = copy(Z)
for i in 1:n
    for j in 1:n
        if (i+j) % 2 != 0
            Z3[i, j] = 1
        end
    end
end
Z3

```

6×6 Matrix{Int64}:

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
1	0	1	0	1	0

Рис. 4.15: Создание матриц Z3

Используя цикл `for`, создадим матрицу Z_4 (рис. 4.16).

```

Z4 = copy(Z)
for i in 1:n
    for j in 1:n
        if (i+j) % 2 == 0
            Z4[i, j] = 1
        end
    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

Рис. 4.16: Создание матриц Z4

Напишем свою функцию, аналогичную функции `outer()` языка R. Функция должна иметь следующий интерфейс: `outer(x,y,operation)` (рис. 4.17).

```
function outer(x, y, operation)
  return [operation(i, j) for i in x, j in y]
end
```

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

Рис. 4.17: Создание функции `outer`

Используя написанную функцию `outer()`, создадим матрицу A_1 (рис. 4.18).

```
A1 = outer(0:4, 0:4, +)
```

5x5 Matrix{Int64}:

0	1	2	3	4
1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8

Рис. 4.18: Создание матрицы A_1

Используя написанную функцию `outer()`, создадим матрицу A_2 (рис. 4.19).

```
A2 = outer(0:4, 1:5, ^)
```

```
A2
```

```
5x5 Matrix{Int64}:
```

0	0	0	0	0
1	1	1	1	1
2	4	8	16	32
3	9	27	81	243
4	16	64	256	1024

Рис. 4.19: Создание матрицы A_2

Используя написанную функцию `outer()`, создадим матрицу A_3 (рис. 4.20).

```
A3 = outer(0:4, 0:4, (x, y) -> mod(x+y, 5))  
A3
```

5×5 Matrix{Int64}:

0	1	2	3	4
1	2	3	4	0
2	3	4	0	1
3	4	0	1	2
4	0	1	2	3

Рис. 4.20: Создание матрицы A_3

Используя написанную функцию `outer()`, создадим матрицу A_4 (рис. 4.21).

```
A4 = outer(0:9, 0:9, (x, y) -> mod(x+y, 10))  
A4
```

10×10 Matrix{Int64}:

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

Рис. 4.21: Создание матрицы A_4

Используя написанную функцию `outer()`, создадим матрицу A_5 (рис. 4.22).

```
A5 = outer(0:8, 0:8, (x, y) -> mod(x-y, 9))  
A5
```

9×9 Matrix{Int64}:

0	8	7	6	5	4	3	2	1
1	0	8	7	6	5	4	3	2
2	1	0	8	7	6	5	4	3
3	2	1	0	8	7	6	5	4
4	3	2	1	0	8	7	6	5
5	4	3	2	1	0	8	7	6
6	5	4	3	2	1	0	8	7
7	6	5	4	3	2	1	0	8
8	7	6	5	4	3	2	1	0

Рис. 4.22: Создание матрицы A_5

Решим следующую систему линейных уравнений с 5 неизвестными (рис. 4.23).


```

A = [
    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;
]
b = [7; -1; -3; 5; 17]

x = A \ b
x

```

```

5-element Vector{Float64}:
-2.000000000000000000036
 3.000000000000000000058
 4.99999999999999999998
 1.99999999999999999991
-3.9999999999999999999

```

Рис. 4.23: Решение системы линейных уравнений

Создайте матрицу M размерности 6×10 , элементами которой являются целые числа выбранные случайным образом с повторениями из совокупности $1, 2, \dots, 10$ (рис. 4.24).

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

```
M
```

```
6×10 Matrix{Int64}:
```

8	9	5	1	5	4	2	10	9	2
8	8	9	5	1	8	10	3	5	1
2	9	8	1	7	10	8	9	9	9
5	9	10	2	1	6	4	3	8	4
9	8	9	6	7	1	7	1	7	4
7	6	8	8	8	1	7	7	4	8

Рис. 4.24: Создание матрицы M

Найдите число элементов в каждой строке матрицы M , которые больше числа N (например, $N = 4$) (рис. 4.25).

```

N = 4
greater_than_N = sum(M .> N)
greater_than_N

```

41

Рис. 4.25: Поиск числа элементов в каждой строке матрицы M

Определим, в каких строках матрицы M число M (например, $M = 7$) встречается ровно 2 раза (рис. 4.26).

```

twice = [i for i=1:6 if sum(M[i, :].==7)==2]
twice

```

Int64[]

Рис. 4.26: Поиск строк с двумя числами M

Определим все пары столбцов матрицы M , сумма элементов которых больше K (например, $K = 75$) (рис. 4.27).

```

K = 75
pairs = []
for i in 1:size(M, 2)-1
    for j in i + 1:size(M, 2)
        if sum(M[:, i] .+ M[:, j]) > K
            push!(pairs, (i, j))
        end
    end
end
pairs

```

18-element Vector{Any}:

```

(1, 2)
(1, 3)
(1, 7)
(1, 9)
(2, 3)
(2, 5)
(2, 6)
(2, 7)
(2, 8)
(2, 9)
(2, 10)
(3, 5)
(3, 6)
(3, 7)
(3, 8)
(3, 9)
(3, 10)
(7, 9)

```

Рис. 4.27: Поиск столбцов матрицы M

Вычислим выражения (рис. 4.28).

```
sum1 = sum(i^4 / (3 + j) for i in 1:20 for j in 1:5)  
sum1
```

639215.2833333334

```
sum2 = sum(i^4 / (3 + i * j) for i in 1:20 for j in 1:5)
```

89912.02146097136

Рис. 4.28: Вычисление выражений

5 Выводы

В результате выполнения данной лабораторной работы мы освоили применение циклов функций и сторонних для Julia пакетов для решения задач линейной алгебры и работы с матрицами.