

Лабораторная работа №2

Компьютерный практикум по статистическому анализу данных

Зиязетдинов Алмаз

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Кортежи	6
2.2	Словари	10
2.3	Множества	12
2.4	Массивы	15
2.5	Самостоятельная работа	27
3	Вывод	37
4	Список литературы. Библиография	38

Список иллюстраций

2.1	Примеры кортежей	7
2.2	Примеры операций над кортежами	9
2.3	Примеры словарей и операций над ними	11
2.4	Примеры множеств и операций над ними	13
2.5	Примеры множеств и операций над ними	14
2.6	Примеры массивов	16
2.7	Примеры массивов	17
2.8	Примеры массивов, заданных некоторыми функциями через включение	19
2.9	Некоторые операции для работы с массивами	21
2.10	Некоторые операции для работы с массивами	22
2.11	Некоторые операции для работы с массивами	23
2.12	Некоторые операции для работы с массивами	24
2.13	Некоторые операции для работы с массивами	25
2.14	Некоторые операции для работы с массивами	26
2.15	Решение заданий №1 и №2	28
2.16	Выполнение подпунктов задания №3	30
2.17	Выполнение подпунктов задания №3	31
2.18	Выполнение подпунктов задания №3	32
2.19	Выполнение подпунктов задания №3	33
2.20	Выполнение подпунктов задания №3	34
2.21	Решение заданий №4, №5 и №6	36

Список таблиц

1 Цель работы

Основная цель работы — изучить несколько структур данных, реализованных в Julia, научиться применять их и операции над ними для решения задач.

2 Выполнение лабораторной работы

2.1 Кортежи

Кортеж (Tuple) — структура данных (контейнер) в виде неизменяемой индексируемой последовательности элементов какого-либо типа (элементы индексируются с единицы).

Синтаксис определения кортежа: (element1, element2, ...).

Примеры кортежей (рис. 2.1):

Примеры кортежей:

```
[1]: # пустой кортеж:  
()
```

```
[1]: ()
```

```
[2]: # кортеж из элементов типа String:  
favoritelang = ("Python", "Julia", "R")
```

```
[2]: ("Python", "Julia", "R")
```

```
[3]: # кортеж из целых чисел:  
x1 = (1, 2, 3)
```

```
[3]: (1, 2, 3)
```

```
[4]: # кортеж из элементов разных типов:  
x2 = (1, 2.0, "tmp")
```

```
[4]: (1, 2.0, "tmp")
```

```
[5]: # именованный кортеж:  
x3 = (a=2, b=1+2)
```

```
[5]: (a = 2, b = 3)
```

Рис. 2.1: Примеры кортежей

Примеры операций над кортежами (рис. 2.2):

▼ Примеры операций над кортежами:

```
[6]: # длина кортежа x2:  
length(x2)
```

```
[6]: 3
```

```
[7]: # обратиться к элементам кортежа x2:  
x2[1], x2[2], x2[3]
```

```
[7]: (1, 2.0, "tmp")
```

```
[8]: # произвести какую-либо операцию (сложение)  
# с вторым и третьим элементами кортежа x1:  
c = x1[2] + x1[3]
```

```
[8]: 5
```

```
[9]: # обращение к элементам именованного кортежа x3:  
x3.a, x3.b, x3[2]
```

```
[9]: (2, 3, 3)
```

```
[10]: # проверка вхождения элементов tmp и 0 в кортеж x2  
# (два способа обращения к методу in()):  
in("tmp", x2), 0 in x2
```

```
[10]: (true, false)
```

Рис. 2.2: Примеры операций над кортежами

2.2 Словари

Словарь — неупорядоченный набор связанных между собой по ключу данных.

Синтаксис определения словаря: `Dict(key1 => value1, key2 => value2, ...)`.

Примеры словарей и операций над ними (рис. 2.3):

Примеры словарей и операций над ними:

```
[11]: # создать словарь с именем phonebook:
phonebook = Dict("Иванов И.И." => ("867-5309", "333-5544"), "Бухгалтерия" => "555-2368")

[11]: Dict{String, Any} with 2 entries:
      "Бухгалтерия" => "555-2368"
      "Иванов И.И." => ("867-5309", "333-5544")

[12]: # вывести ключи словаря:
keys(phonebook)

[12]: KeySet for a Dict{String, Any} with 2 entries. Keys:
      "Бухгалтерия"
      "Иванов И.И."

[13]: # вывести значения элементов словаря:
values(phonebook)

[13]: ValueIterator for a Dict{String, Any} with 2 entries. Values:
      "555-2368"
      ("867-5309", "333-5544")

[14]: # вывести заданные в словаре пары "ключ - значение":
pairs(phonebook)

[14]: Dict{String, Any} with 2 entries:
      "Бухгалтерия" => "555-2368"
      "Иванов И.И." => ("867-5309", "333-5544")

[15]: # проверка вхождения ключа в словарь:
haskey(phonebook, "Иванов И.И.")

[15]: true

[16]: # добавить элемент в словарь:
phonebook["Сидоров П.С."] = "555-3344"

[16]: "555-3344"

[18]: # удалить ключ и связанные с ним значения из словаря
pop!(phonebook, "Иванов И.И.")

[18]: ("867-5309", "333-5544")

[19]: # Объединение словарей (функция merge()):
a = Dict{"foo" => 0.0, "bar" => 42.0};
b = Dict{"baz" => 17, "bar" => 13.0};
merge(a, b), merge(b,a)

[19]: (Dict{String, Real}{ "bar" => 13.0, "baz" => 17, "foo" => 0.0}, Dict{String, Real}{ "bar" => 42.0, "baz" => 17, "foo" => 0.0})
```

Рис. 2.3: Примеры словарей и операций над ними

2.3 Множества

Множество, как структура данных в Julia, соответствует множеству, как математическому объекту, то есть является неупорядоченной совокупностью элементов какого-либо типа. Возможные операции над множествами: объединение, пересечение, разность; принадлежность элемента множеству.

Синтаксис определения множества: `Set(itr)` где `itr` — набор значений, сгенерированных данным итерируемым объектом или пустое множество.

Примеры множеств и операций над ними (рис. 2.4 - рис. 2.5):

Примеры множеств и операций над ними:

```
[20]: # создать множество из четырёх целочисленных значений:  
A = Set([1, 3, 4, 5])
```

```
[20]: Set{Int64} with 4 elements:  
5  
4  
3  
1
```

```
[21]: # создать множество из 11 символьных значений:  
B = Set("abrakadabra")
```

```
[21]: Set{Char} with 5 elements:  
'a'  
'd'  
'r'  
'k'  
'b'
```

```
[22]: # проверка эквивалентности двух множеств:  
S1 = Set([1,2]);  
S2 = Set([3,4]);  
issetequal(S1,S2)
```

```
[22]: false
```

```
[23]: S3 = Set([1,2,2,3,1,2,3,2,1]);  
S4 = Set([2,3,1]);  
issetequal(S3,S4)
```

```
[23]: true
```

Рис. 2.4: Примеры множеств и операций над ними

```
[25]: # объединение множеств:  
C = union(S1, S2)
```

```
[25]: Set{Int64} with 4 elements:  
4  
2  
3  
1
```

```
[26]: # пересечение множеств:  
D = intersect(S1, S3)
```

```
[26]: Set{Int64} with 2 elements:  
2  
1
```

```
[27]: # разность множеств:  
E = setdiff(S3, S1)
```

```
[27]: Set{Int64} with 1 element:  
3
```

```
[28]: # проверка вхождения элементов одного множества в другое:  
issubset(S1, S4)
```

```
[28]: true
```

```
[29]: # добавление элемента в множество:  
push!(S4, 99)
```

```
[29]: Set{Int64} with 4 elements:  
2  
99  
3  
1
```

```
[30]: # удаление последнего элемента множества:  
pop!(S4)
```

```
[30]: 2
```

Рис. 2.5: Примеры множеств и операций над ними

2.4 Массивы

Массив — коллекция упорядоченных элементов, размещённая в многомерной сетке. Векторы и матрицы являются частными случаями массивов.

Общий синтаксис одномерных массивов: `array_name_1 = [element1, element2, ...]`, `array_name_2 = [element1 element2 ...]`

Примеры массивов (рис. 2.6 - рис. 2.7):

Примеры массивов:

```
[31]: # создание пустого массива с абстрактным типом:  
empty_array_1 = []
```

```
[31]: Any[]
```

```
[32]: # создание пустого массива с конкретным типом:  
empty_array_2 = (Int64[])  
empty_array_3 = (Float64[])
```

```
[32]: Float64[]
```

```
[33]: # вектор-столбец:  
a = [1, 2, 3]
```

```
[33]: 3-element Vector{Int64}:  
 1  
 2  
 3
```

```
[34]: # вектор-строка:  
b = [1 2 3]
```

```
[34]: 1x3 Matrix{Int64}:  
 1  2  3
```

```
[35]: # многомерные массивы (матрицы):  
A = [[1, 2, 3] [4, 5, 6] [7, 8, 9]]  
B = [[1 2 3]; [4 5 6]; [7 8 9]]
```

```
[35]: 3x3 Matrix{Int64}:  
 1  2  3  
 4  5  6  
 7  8  9
```

Рис. 2.6: Примеры массивов


```
[36]: # одномерный массив из 8 элементов (массив $1 \times 8$)
# со значениями, случайно распределёнными на интервале [0, 1):
c = rand(1,8)
```

```
[36]: 1x8 Matrix{Float64}:
 0.557104  0.208502  0.388682  0.276108  ...  0.886156  0.497785  0.302989
```

```
[39]: # многомерный массив $2 \times 3$ (2 строки, 3 столбца) элементов
# со значениями, случайно распределёнными на интервале [0, 1):
C = rand(2,3)
```

```
[39]: 2x3 Matrix{Float64}:
 0.665453  0.68888  0.198584
 0.980142  0.952256  0.197731
```

```
[38]: # трёхмерный массив:
D = rand(4, 3, 2)
```

```
[38]: 4x3x2 Array{Float64, 3}:
[:, :, 1] =
 0.950623  0.42717  0.556466
 0.11687  0.827311  0.702365
 0.761554  0.762658  0.490271
 0.994326  0.740285  0.928052

[:, :, 2] =
 0.0408368  0.295546  0.699044
 0.36721  0.885788  0.631421
 0.712159  0.363554  0.572507
 0.0582373  0.323775  0.926727
```

Рис. 2.7: Примеры массивов

Примеры массивов, заданных некоторыми функциями через включение (рис. 2.8):

Примеры массивов, заданных некоторыми функциями через включение:

```
[40]: # массив из квадратных корней всех целых чисел от 1 до 10:  
roots = [sqrt(i) for i in 1:10]
```

```
[40]: 10-element Vector{Float64}:  
 1.0  
 1.4142135623730951  
 1.7320508075688772  
 2.0  
 2.23606797749979  
 2.449489742783178  
 2.6457513110645907  
 2.8284271247461903  
 3.0  
 3.1622776601683795
```

```
[42]: # массив с элементами вида 3*x^2,  
# где x - нечётное число от 1 до 9 (включительно)  
ar_1 = [3*i^2 for i in 1:2:9]
```

```
[42]: 5-element Vector{Int64}:  
 3  
 27  
 75  
 147  
 243
```

```
[43]: # массив квадратов элементов, если квадрат не делится на 5 или 4:  
ar_2=[i^2 for i=1:10 if (i^2%5!=0 && i^2%4!=0)]
```

```
[43]: 4-element Vector{Int64}:  
 1  
 9  
 49  
 81
```

Рис. 2.8: Примеры массивов, заданных некоторыми функциями через включение

Некоторые операции для работы с массивами: (рис. 2.9 - рис. 2.14):

Некоторые операции для работы с массивами:

```
[44]: # одномерный массив из пяти единиц:  
ones(5)
```

```
[44]: 5-element Vector{Float64}:  
 1.0  
 1.0  
 1.0  
 1.0  
 1.0
```

```
[45]: # двумерный массив 2x3 из единиц:  
ones(2,3)
```

```
[45]: 2x3 Matrix{Float64}:  
 1.0  1.0  1.0  
 1.0  1.0  1.0
```

```
[46]: # одномерный массив из 4 нулей:  
zeros(4)
```

```
[46]: 4-element Vector{Float64}:  
 0.0  
 0.0  
 0.0  
 0.0
```

```
[47]: # заполнить массив 3x2 цифрами 3.5  
fill(3.5,(3,2))
```

```
[47]: 3x2 Matrix{Float64}:  
 3.5  3.5  
 3.5  3.5  
 3.5  3.5
```

Рис. 2.9: Некоторые операции для работы с массивами

```
[48]: # заполнение массива посредством функции repeat():  
repeat([1,2],3,3)  
repeat([1 2],3,3)
```

```
[48]: 3x6 Matrix{Int64}:  
 1 2 1 2 1 2  
 1 2 1 2 1 2  
 1 2 1 2 1 2
```

```
[49]: # преобразование одномерного массива из целых чисел от 1 до 12  
# в двумерный массив 2x6  
a = collect(1:12)  
b = reshape(a,(2,6))
```

```
[49]: 2x6 Matrix{Int64}:  
 1 3 5 7 9 11  
 2 4 6 8 10 12
```

```
[50]: # транспонирование  
b'
```

```
[50]: 6x2 adjoint(::Matrix{Int64}) with eltype Int64:  
 1 2  
 3 4  
 5 6  
 7 8  
 9 10  
 11 12
```

```
[51]: # транспонирование  
c = transpose(b)
```

Рис. 2.10: Некоторые операции для работы с массивами

```
[51]: # транспонирование  
c = transpose(b)
```

```
[51]: 6x2 transpose(::Matrix{Int64}) with eltype Int64:  
 1  2  
 3  4  
 5  6  
 7  8  
 9 10  
11 12
```

```
[52]: # массив 10x5 целых чисел в диапазоне [10, 20]:  
ar = rand(10:20, 10, 5)
```

```
[52]: 10x5 Matrix{Int64}:  
18 19 11 20 12  
16 17 19 17 13  
20 18 14 20 19  
19 16 20 13 10  
13 17 20 13 14  
16 19 19 13 20  
18 14 19 15 11  
13 11 16 16 18  
19 15 15 18 19  
11 10 13 14 20
```

```
[53]: # выбор всех значений строки в столбце 2:  
ar[:, 2]
```

```
[53]: 10-element Vector{Int64}:  
19  
17  
18  
16  
17  
19  
14  
11  
15  
10
```

Рис. 2.11: Некоторые операции для работы с массивами

```
[54]: # выбор всех значений в столбцах 2 и 5:  
ar[:, [2, 5]]
```

```
[54]: 10x2 Matrix{Int64}:  
 19 12  
 17 13  
 18 19  
 16 10  
 17 14  
 19 20  
 14 11  
 11 18  
 15 19  
 10 20
```

```
[55]: # все значения строк в столбцах 2, 3 и 4:  
ar[:, 2:4]
```

```
[55]: 10x3 Matrix{Int64}:  
 19 11 20  
 17 19 17  
 18 14 20  
 16 20 13  
 17 20 13  
 19 19 13  
 14 19 15  
 11 16 16  
 15 15 18  
 10 13 14
```

```
[57]: # значения в строках 2, 4, 6 и 8 столбцах 1 и 5:  
ar[[2, 4, 6], [1, 5]]
```

```
[57]: 3x2 Matrix{Int64}:  
 16 13  
 19 10  
 16 20
```

Рис. 2.12: Некоторые операции для работы с массивами


```
[58]: # значения в строке 1 от столбца 3 до последнего столбца:  
ar[1, 3:end]
```

```
[58]: 3-element Vector{Int64}:  
      11  
      20  
      12
```

```
[59]: # сортировка по столбцам:  
sort(ar,dims=1)
```

```
[59]: 10x5 Matrix{Int64}:  
      11  10  11  13  10  
      13  11  13  13  11  
      13  14  14  13  12  
      16  15  15  14  13  
      16  16  16  15  14  
      18  17  19  16  18  
      18  17  19  17  19  
      19  18  19  18  19  
      19  19  20  20  20  
      20  19  20  20  20
```

```
[60]: # сортировка по строкам:  
sort(ar,dims=2)
```

```
[60]: 10x5 Matrix{Int64}:  
      11  12  18  19  20  
      13  16  17  17  19  
      14  18  19  20  20  
      10  13  16  19  20  
      13  13  14  17  20  
      13  16  19  19  20  
      11  14  15  18  19  
      11  13  16  16  18  
      15  15  18  19  19  
      10  11  13  14  20
```

Рис. 2.13: Некоторые операции для работы с массивами

```
[62]: # возврат индексов элементов массива, удовлетворяющих условию:  
findall(ar .> 14)
```

```
[62]: 32-element Vector{CartesianIndex{2}}:  
 CartesianIndex{1, 1}  
 CartesianIndex{2, 1}  
 CartesianIndex{3, 1}  
 CartesianIndex{4, 1}  
 CartesianIndex{6, 1}  
 CartesianIndex{7, 1}  
 CartesianIndex{9, 1}  
 CartesianIndex{1, 2}  
 CartesianIndex{2, 2}  
 CartesianIndex{3, 2}  
 CartesianIndex{4, 2}  
 CartesianIndex{5, 2}  
 CartesianIndex{6, 2}  
 ⋮  
 CartesianIndex{9, 3}  
 CartesianIndex{1, 4}  
 CartesianIndex{2, 4}  
 CartesianIndex{3, 4}  
 CartesianIndex{7, 4}  
 CartesianIndex{8, 4}  
 CartesianIndex{9, 4}  
 CartesianIndex{3, 5}  
 CartesianIndex{6, 5}  
 CartesianIndex{8, 5}  
 CartesianIndex{9, 5}  
 CartesianIndex{10, 5}
```

Рис. 2.14: Некоторые операции для работы с массивами

2.5 Самостоятельная работа

Выполнение заданий №1 и №2 (рис. 2.15):

```
[63]: A = Set([0,3,4,9])
      B = Set([1,3,4,7])
      C = Set([0,1,2,4,7,8,9])
      P = union(intersect(A,B), intersect(A,C), intersect(B,C))
      println(P)
```

```
Set([0, 4, 7, 9, 3, 1])
```

```
[64]: set1 = Set(["белый", "красный", "синий"])
      set2 = Set(["желтый", "синий", "голубой"])
```

```
[64]: Set{String} with 3 elements:
      "голубой"
      "желтый"
      "синий"
```

```
[66]: set1 = Set(["белый", "красный", "синий"])
      set2 = Set(["желтый", "синий", "голубой"])
      intersection = intersect(set1, set2)
      println(intersection)
```

```
Set(["синий"])
```

```
[67]: set1 = Set([1000, 2000, 20])
      set2 = Set([200, 1000, 4000])
      intersection = intersect(set1, set2)
      println(intersection)
```

```
Set([1000])
```

Рис. 2.15: Решение заданий №1 и №2

Выполнение задания №3 (всех подпунктов) (рис. 2.16 - рис. 2.20):

▼ 3.1) массив $(1, 2, 3, \dots, N-1, N)$, N выберите больше 20

```
[73]: N = 25  
array1 = collect(1:N)  
println(array1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
```

3.2) массив $(N, N-1, \dots, 2, 1)$, N выберите больше 20

```
[75]: array2 = collect(N:-1:1)  
println(array2)
```

```
[25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

3.3) массив $(1, 2, 3, \dots, N-1, N, N-1, \dots, 2, 1)$, N выберите больше 20

```
[76]: array3 = vcat(collect(1:N), collect(N:-1:1))  
println(array3)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5,  
4, 3, 2, 1]
```

3.4) массив с именем tmp вида $(4, 6, 3)$

```
[77]: tmp = [4, 6, 3]  
println(tmp)
```

```
[4, 6, 3]
```

Рис. 2.16: Выполнение подпунктов задания №3

3.5) массив, в котором первый элемент массива tmp повторяется 10 раз

```
[78]: array4 = fill(tmp[1], 10)
      println(array4)
```

 $[4, 4, 4, 4, 4, 4, 4, 4, 4, 4]$

3.6) массив, в котором все элементы массива tmp повторяются 10 раз

```
[80]: array5 = repeat(tmp, 10)
println(array5)
```

[4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3, 4, 6, 3]

3.7) массив, в котором первый элемент массива `tmp` встречается 11 раз, второй элемент — 10 раз, третий элемент — 10 раз

```
[81]: array6 = vcat(fill(tmp[1], 11), fill(tmp[2], 10), fill(tmp[3], 10))  
      println(array6)
```

[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]

3.8) массив, в котором первый элемент массива `tmp` встречается 10 раз подряд, второй элемент — 20 раз подряд, третий элемент — 30 раз подряд

```
[82]: array7 = vcat(fill(tmp[1], 10), fill(tmp[2], 20), fill(tmp[3], 30))  
println(array7)
```

[illegible]

Рис. 2.17: Выполнение подпунктов задания №3

3.9) массив из элементов вида $2^{tmp[i]}$, $i = 1, 2, 3$, где элемент $2^{tmp[3]}$ встречается 4 раза; посчитайте в полученном векторе, сколько раз встречается цифра 6, и выведите это значение на экран

```
[100]: # Задаем массив tmp
tmp = [4, 6, 3]

# Формируем массив: элемент 2^tmp[3] повторяется 4 раза
result_array = [2^tmp[i] for i in 1:3]
result_array = vcat(result_array, repeat([2^tmp[3]], 4))

# Преобразуем массив в строку для поиска цифры '6'
result_string = join(result_array, "")
count_6 = count(x -> x == '6', result_string)

# Выводим массив и количество цифр 6
println("Результирующий массив: ", result_array)
println("Количество цифры 6: ", count_6)
```

Результирующий массив: [16, 64, 8, 8, 8, 8]

Количество цифры 6: 2

3.10) вектор значений $y = e^x \cos(x)$ в точках $x = 3, 3.1, 3.2, \dots, 6$, найдите среднее значение y

```
[98]: using Statistics

x = 3:0.1:6
y = [exp(x) * cos(x) for x in x]

println("Среднее значение y: ", mean(y))
```

Среднее значение y: 53.11374594642971

Рис. 2.18: Выполнение подпунктов задания №3

3.11) вектор вида (x^i, y^j) , $x = 0.1, i = 3, 6, 9, \dots, 36, y = 0.2, j = 1, 4, 7, \dots, 34$

[101]:

```
xi = collect(3:3:36) * 0.1
yj = collect(1:3:34) * 0.2
vector11 = [(x, y) for x in xi, y in yj]
println(vector11)
```

```
[(0.30000000000000004, 0.2) (0.30000000000000004, 0.8) (0.30000000000000004, 1.4000000000000001) (0.30000000000000004, 2.0) (0.30000000000000004, 2.6) (0.30000000000000004, 3.2) (0.30000000000000004, 3.8000000000000003) (0.30000000000000004, 4.4) (0.30000000000000004, 5.0) (0.30000000000000004, 5.6000000000000005) (0.30000000000000004, 6.2) (0.30000000000000004, 6.800000000000001) (0.6000000000000001, 0.2) (0.6000000000000001, 0.8) (0.6000000000000001, 1.4000000000000001) (0.6000000000000001, 2.0) (0.6000000000000001, 2.6) (0.6000000000000001, 3.2) (0.6000000000000001, 3.8000000000000003) (0.6000000000000001, 4.4) (0.6000000000000001, 5.0) (0.6000000000000001, 5.6000000000000005) (0.6000000000000001, 6.2) (0.6000000000000001, 6.800000000000001) (0.9, 0.2) (0.9, 0.8) (0.9, 1.4000000000000001) (0.9, 2.0) (0.9, 2.6) (0.9, 3.2) (0.9, 3.8000000000000003) (0.9, 4.4) (0.9, 5.0) (0.9, 5.6000000000000005) (0.9, 6.2) (0.9, 6.800000000000001) (1.2000000000000002, 0.2) (1.2000000000000002, 0.8) (1.2000000000000002, 1.4000000000000001) (1.2000000000000002, 2.0) (1.2000000000000002, 2.6) (1.2000000000000002, 3.2) (1.2000000000000002, 3.8000000000000003) (1.2000000000000002, 4.4) (1.2000000000000002, 5.0) (1.2000000000000002, 5.6000000000000005) (1.2000000000000002, 6.2) (1.2000000000000002, 6.800000000000001) (1.5, 0.2) (1.5, 0.8) (1.5, 1.4000000000000001) (1.5, 2.0) (1.5, 2.6) (1.5, 3.2) (1.5, 3.8000000000000003) (1.5, 4.4) (1.5, 5.0) (1.5, 5.6000000000000005) (1.5, 6.2) (1.5, 6.800000000000001) (1.8, 0.2) (1.8, 0.8) (1.8, 1.4000000000000001) (1.8, 2.0) (1.8, 2.6) (1.8, 3.2) (1.8, 3.8000000000000003) (1.8, 4.4) (1.8, 5.0) (1.8, 5.6000000000000005) (1.8, 6.2) (1.8, 6.800000000000001) (2.1, 0.2) (2.1, 0.8) (2.1, 1.4000000000000001) (2.1, 2.0) (2.1, 2.6) (2.1, 3.2) (2.1, 3.8000000000000003) (2.1, 4.4) (2.1, 5.0) (2.1, 5.6000000000000005) (2.1, 6.2) (2.1, 6.800000000000001) (2.4000000000000004, 0.2) (2.4000000000000004, 0.8) (2.4000000000000004, 1.4000000000000001) (2.4000000000000004, 2.0) (2.4000000000000004, 2.6) (2.4000000000000004, 3.2) (2.4000000000000004, 3.8000000000000003) (2.4000000000000004, 4.4) (2.4000000000000004, 5.0) (2.4000000000000004, 5.6000000000000005) (2.4000000000000004, 6.2) (2.4000000000000004, 6.800000000000001) (2.7, 0.2) (2.7, 0.8) (2.7, 1.4000000000000001) (2.7, 2.0) (2.7, 2.6) (2.7, 3.2) (2.7, 3.8000000000000003) (2.7, 4.4) (2.7, 5.0) (2.7, 5.6000000000000005) (2.7, 6.2) (2.7, 6.800000000000001) (3.0, 0.2) (3.0, 0.8) (3.0, 1.4000000000000001) (3.0, 2.0) (3.0, 2.6) (3.0, 3.2) (3.0, 3.8000000000000003) (3.0, 4.4) (3.0, 5.0) (3.0, 5.6000000000000005) (3.0, 6.2) (3.0, 6.800000000000001) (3.3000000000000003, 0.2) (3.3000000000000003, 0.8) (3.3000000000000003, 1.4000000000000001) (3.3000000000000003, 2.0) (3.3000000000000003, 2.6) (3.3000000000000003, 3.2) (3.3000000000000003, 3.8000000000000003) (3.3000000000000003, 4.4) (3.3000000000000003, 5.0) (3.3000000000000003, 5.6000000000000005) (3.3000000000000003, 6.2) (3.3000000000000003, 6.800000000000001) (3.6, 0.2) (3.6, 0.8) (3.6, 1.4000000000000001) (3.6, 2.0) (3.6, 2.6) (3.6, 3.2) (3.6, 3.8000000000000003) (3.6, 4.4) (3.6, 5.0) (3.6, 5.6000000000000005) (3.6, 6.2) (3.6, 6.800000000000001)]
```

3.12) вектор с элементами $(2^i)/i, i = 1, 2, \dots, M, M = 25$

[102]:

```
M = 25
vector12 = [2^i / i for i in 1:M]
println(vector12)
```

```
[2.0, 2.0, 2.6666666666666665, 4.0, 6.4, 10.666666666666666, 18.285714285714285, 32.0, 56.888888888888886, 102.4, 186.1818181818182, 341.3333333333333, 630.1538461538462, 1170.2857142857142, 2184.5333333333333, 4096.0, 7710.117647058823, 14563.555555555555, 27594.105263157893, 52428.8, 99864.38095238095, 190650.18181818182, 364722.0869565217, 699050.6666666666, 1.34217728e6]
```

Рис. 2.19: Выполнение подпунктов задания №3

3.13) вектор вида ("fn1", "fn2" ..., "fnN"), N = 30

[104]:

```
N = 30
vector13 = ["fn%i" for i in 1:N]
println(vector13)
```

```
["fn1", "fn2", "fn3", "fn4", "fn5", "fn6", "fn7", "fn8", "fn9", "fn10", "fn11", "fn12", "fn13", "fn14", "fn15", "fn16", "fn17", "fn18", "fn19", "fn20", "fn21", "fn22", "fn23", "fn24", "fn25", "fn26", "fn27", "fn28", "fn29", "fn30"]
```

3.14) векторы $x = (x_1, x_2, \dots, x_n)$ и $y = (y_1, y_2, \dots, y_n)$ целочисленного типа длины $n = 250$ как случайные выборки из совокупности 0, 1, ..., 999; на его основе:

[107]:

```
using Random

x = rand{0:999, 250}
y = rand{0:999, 250}

# Выбор y > 600
filtered_y = y[y .> 600]
println("Элементы y > 600: ", filtered_y)

# Соответствующие элементы x
corresponding_x = x[findall(y .> 600)]
println("Соответствующие x: ", corresponding_x)

# Различные операции
sum_exp = sum(exp.(-x[2:end] + x[1:end-1]) .+ 10)
println("Сумма: ", sum_exp)

# Уникальные элементы x
unique_x = unique(x)
println("Уникальные элементы x: ", unique_x)
```

Элементы y > 600: [800, 665, 970, 985, 732, 883, 739, 698, 823, 910, 884, 862, 622, 916, 757, 903, 631, 987, 997, 945, 946, 873, 786, 668, 751, 668, 898, 702, 952, 757, 618, 884, 927, 941, 742, 980, 810, 638, 668, 740, 996, 970, 758, 973, 682, 939, 705, 874, 806, 711, 955, 614, 689, 724, 844, 672, 999, 778, 876, 819, 627, 795, 624, 940, 975, 756, 880, 893, 802, 708, 978, 737, 808, 824, 786, 728, 868, 933, 847, 888, 843, 730, 658, 992, 999, 918, 833, 876, 734, 969, 904, 845, 869, 896, 645, 839, 729, 936]

Соответствующие x: [842, 798, 395, 108, 32, 394, 46, 518, 125, 630, 100, 960, 940, 515, 846, 996, 705, 177, 582, 338, 703, 955, 354, 325, 521, 108, 794, 829, 459, 339, 843, 684, 208, 999, 947, 447, 895, 763, 834, 485, 675, 190, 921, 144, 633, 92, 436, 101, 621, 387, 659, 647, 379, 223, 15, 361, 495, 177, 404, 961, 3, 72, 676, 371, 817, 506, 207, 701, 795, 829, 934, 180, 393, 604, 732, 554, 233, 838, 923, 242, 404, 790, 589, 245, 115, 110, 436, 887, 459, 238, 994, 849, 592, 317, 392, 373, 780, 39]

Сумма: Inf

Уникальные элементы x: [966, 463, 425, 842, 531, 12, 798, 214, 411, 450, 395, 108, 32, 394, 46, 236, 508, 638, 518, 53, 781, 647, 124, 774, 125, 630, 100, 353, 539, 960, 940, 515, 846, 57, 683, 488, 688, 996, 705, 986, 345, 741, 347, 177, 28, 582, 338, 516, 479, 264, 759, 95, 404, 570, 400, 369, 300, 653, 943, 703, 955, 718, 401, 354, 325, 521, 636, 519, 772, 749, 794, 238, 669, 829, 8, 656, 459, 339, 438, 715, 756, 876, 110, 412, 220, 843, 75, 684, 810, 643, 860, 158, 351, 877, 766, 314, 208, 965, 780, 233, 679, 999, 682, 947, 447, 454, 421, 895, 329, 763, 834, 833, 243, 802, 485, 540, 675, 190, 921, 237, 144, 633, 362, 951, 206, 175, 583, 474, 155, 92, 970, 134, 436, 101, 621, 387, 701, 659, 379, 861, 223, 704, 453, 782, 15, 361, 334, 822, 495, 611, 500, 169, 961, 81, 3, 72, 103, 676, 750, 371, 817, 506, 69, 224, 207, 577, 795, 984, 742, 316, 197, 722, 933, 934, 180, 393, 604, 616, 946, 298, 711, 732, 302, 185, 554, 332, 403, 136, 168, 246, 838, 368, 923, 242, 422, 937, 391, 40, 790, 589, 245, 115, 760, 182, 553, 263, 597, 887, 994, 849, 962, 592, 317, 816, 941, 392, 373, 733, 607, 39, 38, 788]

Рис. 2.20: Выполнение подпунктов задания №3

Выполнение заданий №4, №5 и №6 (рис. 2.21):

```
[63]: A = Set([0,3,4,9])
      B = Set([1,3,4,7])
      C = Set([0,1,2,4,7,8,9])
      P = union(intersect(A,B), intersect(A,C), intersect(B,C))
      println(P)
```

```
Set([0, 4, 7, 9, 3, 1])
```

```
[64]: set1 = Set(["белый", "красный", "синий"])
      set2 = Set(["желтый", "синий", "голубой"])
```

```
[64]: Set{String} with 3 elements:
      "голубой"
      "желтый"
      "синий"
```

```
[66]: set1 = Set(["белый", "красный", "синий"])
      set2 = Set(["желтый", "синий", "голубой"])
      intersection = intersect(set1, set2)
      println(intersection)
```

```
Set(["синий"])
```

```
[67]: set1 = Set([1000, 2000, 20])
      set2 = Set([200, 1000, 4000])
      intersection = intersect(set1, set2)
      println(intersection)
```

```
Set([1000])
```

Рис. 2.21: Решение заданий №4, №5 и №6

3 Вывод

В ходе выполнения лабораторной работы были изучены несколько структур данных, реализованных в Julia, а также научились применять их и операции над ними для решения задач.

4 Список литературы. Библиография

[1] Julia Documentation: <https://docs.julialang.org/en/v1/>