РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

дисциплина: Компьютерный практикум

по статистическому данных анализ

Студент: Доре Стевенсон Эдгар

Группа: НКН-бд-01-19

МОСКВА

2023 г.

Постановка задачи

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

1. Используя Jupyter Lab, повторите примеры
2. Выполните задания для самостоятельной работы.

Выполнение работы

1. Повторение примером


```
In [3]: #Примеры операций над кортежами:
        # длина кортежа x2:
        length(x2)
Out[3]: 3

In [4]: # обратиться к элементам кортежа x2:
        x2[1], x2[2], x2[3]
Out[4]: (1, 2.0, "tmp")

In [5]: # произвести какую-либо операцию (сложение)
        # с вторым и третьим элементами кортежа x1:
        c = x1[2] + x1[3]
Out[5]: 5

In [6]: # обращение к элементам именованного кортежа x3:
        x3.a, x3.b, x3[2]
Out[6]: (2, 3, 3)

In [7]: # проверка вхождения элементов tmp и 0 в кортеж x2
        # (два способа обращения к методу in()):
        in("tmp", x2), 0 in x2
Out[7]: (true, false)
```

In [8]:
```
# создать словарь с именем phonebook:
phonebook = Dict("Иванов И.И." => ("867-5309","333-5544"), "Бухгалтерия" => "555-2368")
# вывести ключи словаря:
keys(phonebook)
```

Out[8]:
```
Base.KeySet for a Dict{String,Any} with 2 entries. Keys:
  "Бухгалтерия"
  "Иванов И.И."
```

In [9]:
```
# вывести значения элементов словаря:
values(phonebook)
```

Out[9]:
```
Base.ValueIterator for a Dict{String,Any} with 2 entries. Values:
  "555-2368"
  ("867-5309", "333-5544")
```

In [10]:
```
# вывести заданные в словаре пары "ключ - значение":
pairs(phonebook)
```

Out[10]:
```
Dict{String,Any} with 2 entries:
  "Бухгалтерия" => "555-2368"
  "Иванов И.И." => ("867-5309", "333-5544")
```

In [11]:
```
# проверка вхождения ключа в словарь:
haskey(phonebook, "Иванов И.И.")
```

Out[11]:
```
true
```

In [12]:
```
# добавить элемент в словарь:
phonebook["Сидоров П.С."] = "555-3344"
```

Out[12]:
```
"555-3344"
```

In [14]:
```
# удалить ключ и связанные с ним значения из словаря
pop!(phonebook, "Иванов И.И.")
```

Out[14]:
```
("867-5309", "333-5544")
```

In [15]:
```
# 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)
```

Out[15]:
```
(Dict{String,Real}("bar" => 13.0,"baz" => 17,"foo" => 0.0), Dict{String,Real}("bar" => 42.0,"baz" => 17,"foo" => 0.0))
```

In [16]:
```
# создать множество из четырёх целочисленных значений:
A = Set([1, 3, 4, 5])
# создать множество из 11 символьных значений:
B = Set("abrakadabra")
# проверка эквивалентности двух множеств:
S1 = Set([1,2]);
S2 = Set([3,4]);
issetequal(S1,S2)
```

Out[16]:
```
false
```

In [17]:
```
S3 = Set([1,2,2,3,1,2,3,2,1]);
S4 = Set([2,3,1]);
issetequal(S3,S4)
```

Out[17]:
```
true
```

In [18]:
```
# объединение множеств:
C=union(S1,S2)
```

Out[18]:
```
Set{Int64} with 4 elements:
  4
  2
  3
  1
```

In [19]:
```
# пересечение множеств:
D = intersect(S1,S3)
```

Out[19]:
```
Set{Int64} with 2 elements:
  2
  1
```

In [20]:
```
# разность множеств:
E = setdiff(S3,S1)
```

Out[20]:
```
Set{Int64} with 1 element:
  3
```

In [21]:
```
# проверка вхождения элементов одного множества в другое:
issubset(S1,S4)
```

Out[21]:
```
true
```

In [22]:
```
# добавление элемента в множество:
push!(S4, 99)
# удаление последнего элемента множества:
pop!(S4)
```

Out[22]:
```
2
```

In [28]:
```
# создание пустого массива с абстрактным типом:
empty_array_1 = []
# вектор-столбец:
a = [1, 2, 3]
# вектор-строка:
b = [1 2 3]
# многомерные массивы (матрицы):
A = [[1, 2, 3] [4, 5, 6] [7, 8, 9]]
B = [[1 2 3]; [4 5 6]; [7 8 9]]
```

Out[28]:
```
3×3 Array{Int64,2}:
 1  2  3
 4  5  6
 7  8  9
```

In [29]:
```
# одномерный массив из 8 элементов (массив $1 \times 8$)
# со значениями, случайно распределёнными на интервале [0, 1):
c = rand(1,8)
```

Out[29]:
```
1×8 Array{Float64,2}:
 0.944048  0.601037  0.954965  0.359763  …  0.585821  0.36182  0.0915807
```

In [33]:
```
# многомерный массив $2 \times 3$ (2 строки, 3 столбца) элементов
# со значениями, случайно распределёнными на интервале [0, 1):
C = rand(2,3)
```

Out[33]:
```
2×3 Array{Float64,2}:
 0.826433  0.461196  0.899106
 0.154855  0.346251  0.295687
```

In [32]:
```
# трёхмерный массив:
D = rand(4, 3, 2)
```

Out[32]:
```
4×3×2 Array{Float64,3}:
[:, :, 1] =
 0.429819   0.301406  0.00514119
 0.0793302  0.627647  0.878831
 0.630654   0.165663  0.433177
 0.932676   0.883069  0.325384

[:, :, 2] =
 0.679334  0.0930755  0.687508
 0.859722  0.46839    0.47003
 0.781023  0.0968937  0.759207
 0.432809  0.600891   0.166625
```

In [34]:
```
#Примеры массивов, заданных некоторыми функциями через включение:
# массив из квадратных корней всех целых чисел от 1 до 10:
roots = [sqrt(i) for i in 1:10]
# 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]
```

Out[34]:
```
5-element Array{Int64,1}:
   3
  27
  75
 147
 243
```

In [35]:
```
#Примеры массивов, заданных некоторыми функциями через включение:
# массив из квадратных корней всех целых чисел от 1 до 10:
roots = [sqrt(i) for i in 1:10]
```

Out[35]:
```
10-element Array{Float64,1}:
  1.0
  1.4142135623730951
  1.7320508075688772
  2.0
  2.23606797749979
  2.449489742783178
  2.6457513110645907
  2.8284271247461903
  3.0
  3.1622776601683795
```

In [36]:
```
# 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]
```

Out[36]:
```
5-element Array{Int64,1}:
   3
  27
  75
 147
 243
```

In [37]:
```
# 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)]
```

Out[37]:
```
4-element Array{Int64,1}:
  1
  9
 49
 81
```

In [38]:
```
# одномерный массив из пяти единиц:
ones(5)
```

Out[38]:
```
5-element Array{Float64,1}:
 1.0
 1.0
 1.0
 1.0
 1.0
```

In [39]:
```
# двумерный массив 2x3 из единиц:
ones(2,3)
```

Out[39]:
```
2×3 Array{Float64,2}:
 1.0  1.0  1.0
 1.0  1.0  1.0
```

In [40]:
```
# одномерный массив из 4 нулей:
zeros(4)
```

Out[40]:
```
4-element Array{Float64,1}:
 0.0
 0.0
 0.0
 0.0
```

In [41]:
```
# заполнить массив 3x2 цифрами 3.5
fill(3.5,(3,2))
```

Out[41]:
```
3×2 Array{Float64,2}:
 3.5  3.5
 3.5  3.5
 3.5  3.5
```

In [42]:
```
# заполнение массива посредством функции repeat():
repeat([1,2],3,3)
repeat([1 2],3,3)
```

Out[42]:
```
3×6 Array{Int64,2}:
 1  2  1  2  1  2
 1  2  1  2  1  2
 1  2  1  2  1  2
```

In [43]:
```
# преобразование одномерного массива из целых чисел от 1 до 12
# в двумерный массив 2x6
a = collect(1:12)
b = reshape(a,(2,6))
```

Out[43]:
```
2×6 Array{Int64,2}:
 1  3  5  7   9  11
 2  4  6  8  10  12
```

In [44]:
```
# транспонирование
b'
# транспонирование
c = transpose(b)
```

Out[44]:
```
6×2 LinearAlgebra.Transpose{Int64,Array{Int64,2}}:
  1   2
  3   4
  5   6
  7   8
  9  10
 11  12
```

In [45]:
```
# массив 10x5 целых чисел в диапазоне [10, 20]:
ar = rand(10:20, 10, 5)
```

Out[45]:
```
10×5 Array{Int64,2}:
 19  18  20  19  13
 15  10  16  16  10
 19  18  10  14  15
 19  11  12  14  11
 19  10  16  20  14
 16  11  20  11  12
 11  18  15  16  14
 16  10  13  10  12
 17  18  20  14  20
 13  11  20  19  15
```

In [46]:
```
# выбор всех значений строки в столбце 2:
ar[:, 2]
```

Out[46]:
```
10-element Array{Int64,1}:
 18
 10
 18
 11
 10
 11
 18
 10
 18
 11
```

In [47]:
```
# выбор всех значений в столбцах 2 и 5:
ar[:, [2, 5]]
```

Out[47]:
```
10×2 Array{Int64,2}:
 18  13
 10  10
 18  15
 11  11
 10  14
 11  12
 18  14
 10  12
 18  20
 11  15
```

In [48]:
```
# все значения строк в столбцах 2, 3 и 4:
ar[:, 2:4]
```

Out[48]:
```
10×3 Array{Int64,2}:
 18  20  19
 10  16  16
 18  10  14
 11  12  14
 10  16  20
 11  20  11
 18  15  16
 10  13  10
 18  20  14
 11  20  19
```

In [49]:
```
# значения в строках 2, 4, 6 и в столбцах 1 и 5:
ar[[2, 4, 6], [1, 5]]
```

Out[49]:
```
3×2 Array{Int64,2}:
 15  10
 19  11
 16  12
```

In [50]:
```
# значения в строке 1 от столбца 3 до последнего столбца:
ar[1, 3:end]
```

Out[50]:
```
3-element Array{Int64,1}:
 20
 19
 13
```

In [51]:
```
# сортировка по строкам:
sort(ar,dims=1)
```

Out[51]:
```
10×5 Array{Int64,2}:
 11  10  10  10  10
 13  10  12  11  11
 15  10  13  14  12
 16  11  15  14  12
 16  11  16  14  13
 17  11  16  16  14
 19  18  20  16  14
 19  18  20  19  15
 19  18  20  19  15
 19  18  20  20  20
```

In [52]:
```
# сортировка по столбцам:
sort(ar,dims=2)
```

Out[52]:
```
10×5 Array{Int64,2}:
 13  18  19  19  20
 10  10  15  16  16
 10  14  15  18  19
 11  11  12  14  19
 10  14  16  19  20
 11  11  12  16  20
 11  14  15  16  18
 10  10  12  13  16
 14  17  18  20  20
 11  13  15  19  20
```

In [53]:
```
# поэлементное сравнение с числом
# (результат - массив логических значений):
ar .> 14
```

Out[53]:
```
10×5 BitArray{2}:
 1  1  1  1  0
 1  0  1  1  0
 1  1  0  0  1
 1  0  0  0  0
 1  0  1  1  0
 1  0  1  0  0
 0  1  1  1  0
 1  0  0  0  0
 1  1  1  0  1
 0  0  1  1  1
```

In [54]:
```
# возврат индексов элементов массива, удовлетворяющих условию:
findall(ar .> 14)
```

Out[54]:
```
27-element Array{CartesianIndex{2},1}:
 CartesianIndex(1, 1)
 CartesianIndex(2, 1)
 CartesianIndex(3, 1)
 CartesianIndex(4, 1)
 CartesianIndex(5, 1)
 CartesianIndex(6, 1)
 CartesianIndex(8, 1)
 CartesianIndex(9, 1)
 CartesianIndex(1, 2)
 CartesianIndex(3, 2)
 CartesianIndex(7, 2)
 CartesianIndex(9, 2)
 CartesianIndex(1, 3)
 ⋮
 CartesianIndex(6, 3)
 CartesianIndex(7, 3)
 CartesianIndex(9, 3)
 CartesianIndex(10, 3)
 CartesianIndex(1, 4)
 CartesianIndex(2, 4)
 CartesianIndex(5, 4)
 CartesianIndex(7, 4)
 CartesianIndex(10, 4)
 CartesianIndex(3, 5)
 CartesianIndex(9, 5)
 CartesianIndex(10, 5)
```

2.Выполните задания для самостоятельной работы

In [55]:
```
#Даны множества: A = {0, 3, 4, 9}, B = {1, 3, 4, 7}, C = {0, 1, 2, 4, 7, 8, 9}. Найти P = A ∩ B ∪ A ∩ B ∪ A ∩ C ∪ B ∩ C.
A=Set([0, 3, 4, 9])
B=Set([1, 3, 4, 7])
C=Set([0, 1, 2, 4, 7, 8, 9])
P=intersect(A,B)
P=union(P,A)
P=intersect(P,B)
P=union(P,A)
P=intersect(P,C)
P=union(P,B)
P=intersect(P,C)
A
```

Out[55]:
```
Set{Int64} with 4 elements:
  0
  4
  9
  3
```

In [56]:
```
#Приведите свои примеры с выполнением операций над множествами элементов разных типов.
push!(A, 5)
pop!(A)
P1=setdiff(A, P)
a=fill(0.0,10)
a1=rand(3,5)
a2=collect(1:1:10.0)
```

Out[56]:
```
10-element Array{Float64,1}:
  1.0
  2.0
  3.0
  4.0
  5.0
  6.0
  7.0
  8.0
  9.0
 10.0
```

In [60]:
```
N=30
#массив (1, 2, 3, … N - 1, N )
A1 = [ i for i in 1:N]
```

Out[60]:
```
30-element Array{Int64,1}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
  ⋮
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
```

In [61]:
```
#массив (N, N - 1 … , 2, 1)
A2 = [N-i for i in 0:N-1 ]
```

Out[61]:
```
30-element Array{Int64,1}:
 30
 29
 28
 27
 26
 25
 24
 23
 22
 21
 20
 19
 18
  ⋮
 12
 11
 10
  9
  8
  7
  6
  5
  4
  3
  2
  1
```

In [62]:
```
#массив (1, 2, 3, … , N - 1, N, N - 1, … , 2, 1)
A3 = vcat(A1,A2)
```

Out[62]:
```
60-element Array{Int64,1}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
  ⋮
 12
 11
 10
  9
  8
  7
  6
  5
  4
  3
  2
  1
```

In [63]:
```
#массив с именем tmp вида (4, 6, 3)
tmp = [4, 6 ,3]
```

Out[63]:
```
3-element Array{Int64,1}:
 4
 6
 3
```

In [64]:
```
#массив, в котором первый элемент массива tmp повторяется 10 раз;
tmp1 = repeat([tmp[1]], 10)
```

Out[64]:
```
10-element Array{Int64,1}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
```

In [65]:
```
#массив, в котором все элементы массива tmp повторяются 10 раз;
tmp2 =repeat(tmp, 10)
```

Out[65]:
```
30-element Array{Int64,1}:
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 ⋮
 4
 6
 3
 4
 6
 3
 4
 6
 3
 4
 6
 3
```

In [66]:
```
#массив, в котором первый элемент массива tmp встречается 11 раз, второй элемент – 10 раз, третий элемент – 10 раз;
tmp3 = vcat(repeat([tmp[1]], 11), repeat([tmp[2]], 10), repeat([tmp[3]], 10))
```

Out[66]:
```
31-element Array{Int64,1}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 6
 6
 ⋮
 6
 6
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
```

In [67]:
```
#массив, в котором первый элемент массива tmp встречается 10 раз подряд, второй элемент – 20 раз подряд, третий элемент – 30 раз
tmp4 = vcat(repeat([tmp[1]], 10), repeat([tmp[2]], 20), repeat([tmp[3]], 30))
```

Out[67]:
```
60-element Array{Int64,1}:
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 4
 6
 6
 6
 ⋮
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
 3
```

In [68]:
```
#массив из элементов вида 2^tmp[i]
tmp5 = repeat([2^tmp[i] for i in 1:3], 4)
```

Out[68]:
```
12-element Array{Int64,1}:
 16
 64
  8
 16
 64
  8
 16
 64
  8
 16
 64
  8
```

In [69]:
```
#вектор значений y = e^x * cos(x)
tmp6=[y=exp(x)*cos(x) for x = 3.0:0.1:6]
using Statistics
mean(tmp6)
```

Out[69]:
```
53.11374594642971
```

In [70]:
```
#вектор вида (x^i, y^j), x = 0.1, i = 3, 6, 9, … , 36, y = 0.2, j = 1, 4, 7, … , 34;
A4=[ [0.1^i, 0.2^j]  for i = 3:3:36 for j=1:3:34]
```

Out[70]:
```
144-element Array{Array{Float64,1},1}:
 [0.001000000000000002, 0.2]
 [0.001000000000000002, 0.0016000000000000007]
 [0.001000000000000002, 1.2800000000000005e-5]
 [0.001000000000000002, 1.0240000000000006e-7]
 [0.001000000000000002, 8.192000000000006e-10]
 [0.001000000000000002, 6.553600000000055e-12]
 [0.001000000000000002, 5.2428800000000056e-14]
 [0.001000000000000002, 4.194304000000005e-16]
 [0.001000000000000002, 3.3554432000000048e-18]
 [0.001000000000000002, 2.68435456000004e-20]
 [0.001000000000000002, 2.1474836480000035e-22]
 [0.001000000000000002, 1.717986918400003e-24]
 [1.0000000000000004e-6, 0.2]
 ⋮
 [1.000000000000002e-36, 0.2]
 [1.000000000000002e-36, 0.0016000000000000007]
 [1.000000000000002e-36, 1.2800000000000005e-5]
 [1.000000000000002e-36, 1.0240000000000006e-7]
 [1.000000000000002e-36, 8.192000000000006e-10]
 [1.000000000000002e-36, 6.553600000000055e-12]
 [1.000000000000002e-36, 5.2428800000000056e-14]
 [1.000000000000002e-36, 4.194304000000005e-16]
 [1.000000000000002e-36, 3.3554432000000048e-18]
 [1.000000000000002e-36, 2.68435456000004e-20]
 [1.000000000000002e-36, 2.1474836480000035e-22]
 [1.000000000000002e-36, 1.717986918400003e-24]
```

In [71]:
```
#Вектор с элементами 2^i/i, i = 1, 2, … , M, M = 25;
A5 = [ (2^i)/i for i = 1:1:25 ]
```

Out[71]:
```
25-element Array{Float64,1}:
       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
```

In [72]:
```
#Вектор вида ("fn1", "fn2", …, "fnN")
num = collect(1:30)
fn= repeat("fn",30)
map(string, fn, num)
```

Out[72]:
```
30-element Array{String,1}:
 "f1"
 "n2"
 "f3"
 "n4"
 "f5"
 "n6"
 "f7"
 "n8"
 "f9"
 "n10"
 "f11"
 "n12"
 "f13"
 ⋮
 "f19"
 "n20"
 "f21"
 "n22"
 "f23"
 "n24"
 "f25"
 "n26"
 "f27"
 "n28"
 "f29"
 "n30"
```

In [73]:
```
# Векторы x = (x1, x2, … , xn) и y = (y1, y2, … , yn) целочисленного типа длины n = 250 как случайные выборки из совокупности 0,
x=rand(0:999,250)
y=rand(0:999,250)
# сформируйте вектор (y2 – x1, … , yn – xn-1);
y[2:250]-x[1:249]
```

Out[73]:
```
249-element Array{Int64,1}:
 -390
 -248
 -337
  212
 -752
  517
   36
 -377
  240
  169
  211
 -658
 -293
    ⋮
  156
  466
 -194
 -408
  385
  202
 -278
 -167
 -235
 -531
  820
  387
```

In [74]:
```
# - сформируйте вектор (x1 + 2x2 - x3, x2 + 2x3 - x4, … , xn-2 + 2xn-1 - xn);
x[1:248]+2*x[2:249]-x[3:250]
```

Out[74]:
```
248-element Array{Int64,1}:
 1585
 1705
  891
 2125
  710
 -208
 2024
 1040
  685
 1124
 1905
 1878
 1643
    ⋮
 1666
 1472
  534
 1015
 1543
 -184
 1584
  716
 1093
 1630
  381
  509
```

In [75]:
```
#сформируйте вектор (sin(y1)/cos(x2),sin(y2)/cos(x3), … ,sin(yn-1)/cos(xn));
[ sin(y[i])/cos(x[i+1]) for i in 1:249 ]
```

Out[75]:
```
249-element Array{Float64,1}:
 -1.389148885291015
  0.35325593651065845
 -0.8636717671297872
  0.9758953456144296
  0.8549252104465984
  1.1841219408292523
 -1.23056774278581
  2.0954853190293514
  2.332859424192246
 -0.7374725463714232
 -0.6689576251075006
  1.481856158977029
 -1.5893143903249205
    ⋮
 -4.522276674019646
  4.535676908851898
  2.3786971216942217
 -1.7985369712984942
 -1.7760604826378195
 -6.438421192585966
 -0.5271768111119243
  0.10417332095632917
  0.8557245298202915
 -1.010202253655946
  0.7004218235209354
 -2.200574181565245
```

In [76]:
```
#Вычислите сумму
sum([ exp(-x[i+1]) / (x[i]+10) for i in 1:249])
```

Out[76]:
```
7.065791692716306e-9
```

In [77]:
```
#Выберите элементы вектора y, значения которых больше 600, и выведите на экран; определите индексы этих элементов;
yf=findall(y->y>600,y)
yr=[]
for i in 1:length(yf)
    push!(yr,y[yf[i]])
end
```

In [78]:
```
#определите значения вектора x, соответствующие значениям вектора y, значения которых больше 600
xr=[]
for i in 1:length(yf)
    push!(xr,x[yf[i]])
end
```

In [79]:
```
#сформируйте вектор (|x1 - x|^1/2, |x2 - x|^1/2, … , |xn - x|^1/2 )

av_x=mean(x)
[ abs(x[i]-av_x)^0.5 for i in 1:250 ]
```

Out[79]:
```
250-element Array{Float64,1}:
  6.938587752561756
 17.092220452591874
 12.0475723695689
  4.985579204064458
 17.382289837647974
 21.162608534866393
 14.347682739731876
 21.122121105608688
 15.487285107467997
  4.91365444450462
 14.870911202747463
 20.97960914793219
 17.150626810702867
  ⋮
  6.917803119488151
 15.161002605368814
  7.4929300010076165
  5.903897018072046
 21.420924349803396
 18.142326201454985
 16.936823787239447
 11.71084967028439
  2.802855686616775
 20.269583123488257
 10.623370463275768
  6.917803119488151
```

In [80]:
```
#определите, сколько элементов вектора y отстоят от максимального значения не более, чем на 200;
ymax=maximum(y)
ym200=findall(y->(abs(y-ymax)<=200),y)
length(ym200)
```

Out[80]:
```
54
```

In [81]:
```
#определите, сколько чётных и нечётных элементов вектора x;
xchet=findall(x->(x%2==0),x)
println(length(xchet))

xnechet=findall(x->(x%2!=0),x)
println(length(xnechet))
```

```
126
124
```

In [82]:
```
#определите, сколько элементов вектора x кратны 7;
x7=findall(x->(x%7==0),x)
length(x7)
```

Out[82]:
```
31
```

In [83]:
```
#отсортируйте элементы вектора x в порядке возрастания элементов вектора y;
mp=sortperm(y)
x_sort=[]
for i in mp
    push!(x_sort,x[i])
end
```

In [84]:
```
#выведите элементы вектора x, которые входят в десятку наибольших
x_top=sort(x, rev=true)
for i in 1:10
    println(x_top[i])
end
```

```
995
994
993
991
989
984
978
977
966
962
```

In [85]:
```
#сформируйте вектор, содержащий только уникальные (неповторяющиеся) элементы вектора x.
unique(x)
```

Out[85]:
```
216-element Array{Int64,1}:
 597
 841
 694
 524
 851
 101
 343
 995
 309
 573
 770
 989
 843
   ⋮
 801
 207
```

In [86]:
```
#Создайте массив squares, в котором будут храниться квадраты всех целых чисел от 1 до 100.
squares= [i^2 for i in 1:100]
```

Out[86]:
```
100-element Array{Int64,1}:
     1
     4
     9
    16
    25
    36
    49
    64
    81
   100
   121
   144
   169
     ⋮
  7921
  8100
  8281
  8464
  8649
  8836
  9025
  9216
  9409
  9604
  9801
 10000
```

In [87]:
```
#Подключите пакет Primes (функции для вычисления простых чисел). Сгенерируйте
#массив myprimes, в котором будут храниться первые 168 простых чисел. Определите
#89-е наименьшее простое число. Получите срез массива с 89-го до 99-го элемента
#включительно, содержащий наименьшие простые числа.
using Primes
myprimes=primes(999)
println(myprimes[89])
println(myprimes[89:99])
```

```
461
[461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523]
```

In [88]:
```
#Вычислите суммы
s1=[i^3 + 4*i^2 for i in 10:100]
sum(s1)
```

Out[88]:
```
26852735
```

In [89]:
```
s2=[(2^i)/i + (3^i)/(i*i) for i in 1:25]
sum(s2)
```

Out[89]:
```
2.1291704368143802e9
```

In [90]:
```
x=collect(2:2:38)
sum(cumprod(x./(x.+1)))
```

Out[90]:
```
5.97634613789762
```

In [ ]:
```

```

Выводы

Получены навыки работы со структурами данных в Julia