Mownit_Lab2

March 21, 2021

1 MOwNiT

1.1 Laboratorium 2

1.1.1 Analiza danych - DataFrames

- Zaawansowany pakiet do działania na tabelkach nxm danych
- Podobne do pandas DataFrames w Pythonie albo data.frame w R
- Specjalistyczne funkcje do statystyki
- W przypadku włąsnej instalacji należy zaistalować pakiet:

```
Pkg.add("DataFrames")
```

- DataFrame to rodzaj bazy danych in-memory
- Składa się z kolumn, do których odwołujemy się używając symboli
- Każda z kolumn może przechowywać inny typ (inaczej niż w dwuwymiarowych tablicach)
- od wersji 0.11 każda z kolumn jest typu Array{T,1} gdzie T jest okreslonym typem danych (np. Float64)
- strona pakietu: https://github.com/JuliaStats/DataFrames.jl
- dokumentacja najnowszej wersji: https://juliadata.github.io/DataFrames.jl/stable/index.html

[1]:
```
# za pierwszym razem instalujemy
# using Pkg
# Pkg.add("DataFrames")
```

[2]:
```
# Tworzenie DataFrame
using DataFrames
df1=DataFrame()
df1.MojaKolumna= 1:4
df1.x2= [4,pi,sqrt(2), 42]
df1.Col3= [true,false, true, false]
show(df1)
```

```
4×3 DataFrame
 Row   MojaKolumna  x2         Col3
       Int64        Float64    Bool

   1             1   4.0        true
   2             2   3.14159   false
```

```
  3             3   1.41421    true
  4             4   42.0      false
```

[3]:
```
# ! - nie tworzy kopii, : - tworzy kopię
typeof(df1[!,2])
```

[3]:
```
Array{Float64,1}
```

[4]:
```
typeof(df1.Col3)
```

[4]:
```
Array{Bool,1}
```

[5]:
```
# możemy też utworzyć DataFrame używając konstruktora
df2=DataFrame(MojaKolumna=1:10,
    Col2=[2,pi, sqrt(2), 3, 4,2,pi, sqrt(2), 3, 4],
    Col3=[true,true,false,true,false,true,true,false,true,false])
```

[5]:

	MojaKolumna	Col2	Col3
	Int64	Float64	Bool
1	1	2.0	1
2	2	3.14159	1
3	3	1.41421	0
4	4	3.0	1
5	5	4.0	0
6	6	2.0	1
7	7	3.14159	1
8	8	1.41421	0
9	9	3.0	1
10	10	4.0	0

[6]:
```
# ilosc wierszy
size(df2, 1)
```

[6]:
```
10
```

[7]:
```
#ilosc kolumn
size(df2, 2)
```

[7]:
```
3
```

[8]:
```
# można odwoływać się po indeksie
show(df2[:,2])
```

```
[2.0, 3.141592653589793, 1.4142135623730951, 3.0, 4.0, 2.0, 3.141592653589793,
1.4142135623730951, 3.0, 4.0]
```

[9]:
```
# albo po symbolu kolumny
show(df2[!,:Col2])
```

```
[2.0, 3.141592653589793, 1.4142135623730951, 3.0, 4.0, 2.0, 3.141592653589793,
1.4142135623730951, 3.0, 4.0]
```

[10]:
```
# wiersze lub podzbiory wierszy i kolumn uzyskujemy poprzez operator (:).␣
↪Wynikiem jest nowy DataFrame
show(df2[3,:])
```

```
DataFrameRow
 Row  MojaKolumna  Col2     Col3
      Int64        Float64  Bool

   3            3  1.41421  false
```

[11]:
```
# drugi i trzeci wiersz
df2[2:3, :]
```

[11]:

	MojaKolumna	Col2	Col3
	Int64	Float64	Bool
1	2	3.14159	1
2	3	1.41421	0

[12]:
```
# druga kolumna drugiego i trzeciego wiersza
df2[2:3, :Col2]
```

[12]:
```
2-element Array{Float64,1}:
 3.141592653589793
 1.4142135623730951
```

[13]:
```
# druga i trzecia kolumna drugiego i trzeciego wiersza
df2[2:3,[:Col2, :Col3]]
```

[13]:

	Col2	Col3
	Float64	Bool
1	3.14159	1
2	1.41421	0

[14]:
```
# pierwsze sześć wierszy
DataFrames.first(df2,6)
```

[14]:

	MojaKolumna	Col2	Col3
	Int64	Float64	Bool
1	1	2.0	1
2	2	3.14159	1
3	3	1.41421	0
4	4	3.0	1
5	5	4.0	0
6	6	2.0	1

[15]:
```
# ostatnie sześć wierszy
DataFrames.last(df2,6)
```

[15]:

	MojaKolumna	Col2	Col3
	Int64	Float64	Bool
1	5	4.0	0
2	6	2.0	1
3	7	3.14159	1
4	8	1.41421	0
5	9	3.0	1
6	10	4.0	0

[16]:
```
# nazwy kolumn
names(df2)
```

[16]:
```
3-element Array{String,1}:
 "MojaKolumna"
 "Col2"
 "Col3"
```

[17]:
```
# typy kolumn
eltype.(eachcol(df2))
```

[17]:
```
3-element Array{DataType,1}:
 Int64
 Float64
 Bool
```

[18]:
```
# podstawowe dane statystyczne o wartościach w kolumnie
describe(df2)
```

[18]:

	variable	mean	min	median	max	nmissing	eltype
	Symbol	Float64	Real	Float64	Real	Int64	DataType
1	MojaKolumna	5.5	1	5.5	10	0	Int64
2	Col2	2.71116	1.41421	3.0	4.0	0	Float64
3	Col3	0.6	0	1.0	1	0	Bool

[19]:
```
using Statistics
mean(df2.Col2)
```

[19]:
```
2.711161243192578
```

[20]:
```
var(df2.Col2)
```

[20]:
```
0.9150284373648316
```

[21]:

```julia
# Pkg.add("CSV")
using CSV
input="winequality.csv"
mydata=CSV.read(input, delim=";",DataFrame)
```

[21]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	
	Float64	Float64	Float64	Float64	Float64	Float64	
1	7.4	0.7	0.0	1.9	0.076	11.0	...
2	7.8	0.88	0.0	2.6	0.098	25.0	...
3	7.8	0.76	0.04	2.3	0.092	15.0	...
4	11.2	0.28	0.56	1.9	0.075	17.0	...
5	7.4	0.7	0.0	1.9	0.076	11.0	...
6	7.4	0.66	0.0	1.8	0.075	13.0	...
7	7.9	0.6	0.06	1.6	0.069	15.0	...
8	7.3	0.65	0.0	1.2	0.065	15.0	...
9	7.8	0.58	0.02	2.0	0.073	9.0	...
10	7.5	0.5	0.36	6.1	0.071	17.0	...
11	6.7	0.58	0.08	1.8	0.097	15.0	...
12	7.5	0.5	0.36	6.1	0.071	17.0	...
13	5.6	0.615	0.0	1.6	0.089	16.0	...
14	7.8	0.61	0.29	1.6	0.114	9.0	...
15	8.9	0.62	0.18	3.8	0.176	52.0	...
16	8.9	0.62	0.19	3.9	0.17	51.0	...
17	8.5	0.28	0.56	1.8	0.092	35.0	...
18	8.1	0.56	0.28	1.7	0.368	16.0	...
19	7.4	0.59	0.08	4.4	0.086	6.0	...
20	7.9	0.32	0.51	1.8	0.341	17.0	...
21	8.9	0.22	0.48	1.8	0.077	29.0	...
22	7.6	0.39	0.31	2.3	0.082	23.0	...
23	7.9	0.43	0.21	1.6	0.106	10.0	...
24	8.5	0.49	0.11	2.3	0.084	9.0	...
25	6.9	0.4	0.14	2.4	0.085	21.0	...
26	6.3	0.39	0.16	1.4	0.08	11.0	...
27	7.6	0.41	0.24	1.8	0.08	4.0	...
28	7.9	0.43	0.21	1.6	0.106	10.0	...
29	7.1	0.71	0.0	1.9	0.08	14.0	...
30	7.8	0.645	0.0	2.0	0.082	8.0	...
...	...	...	...	...	...	...	

[22]:

```julia
describe(mydata)
```

[22]:

	variable	mean	min	median	max	nmissing	eltype
	Symbol	Float64	Real	Float64	Real	Int64	DataType
1	fixed acidity	8.31964	4.6	7.9	15.9	0	Float64
2	volatile acidity	0.527821	0.12	0.52	1.58	0	Float64
3	citric acid	0.270976	0.0	0.26	1.0	0	Float64
4	residual sugar	2.53881	0.9	2.2	15.5	0	Float64
5	chlorides	0.0874665	0.012	0.079	0.611	0	Float64
6	free sulfur dioxide	15.8749	1.0	14.0	72.0	0	Float64
7	total sulfur dioxide	46.4678	6.0	38.0	289.0	0	Float64
8	density	0.996747	0.99007	0.99675	1.00369	0	Float64
9	pH	3.31111	2.74	3.31	4.01	0	Float64
10	sulphates	0.658149	0.33	0.62	2.0	0	Float64
11	alcohol	10.423	8.4	10.2	14.9	0	Float64
12	quality	5.63602	3	6.0	8	0	Int64

[23]:
```
typeof(mydata)
```

[23]:
```
DataFrame
```

[24]:
```
size(mydata)
```

[24]:
```
(1599, 12)
```

[25]:
```
# Dzielenie DataFrame na podgrupy i działania na nich

# Split -Apply - Combine

#https://dataframes.juliadata.org/stable/man/split_apply_combine/

# rozdzielanie na podgrupy po jakości wina (quality)
# Split

wine_grouped=groupby(mydata, :quality)
```

[25]:

GroupedDataFrame with 6 groups based on key: quality

First Group (681 rows): quality = 5

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	
	Float64	Float64	Float64	Float64	Float64	Float64	
1	7.4	0.7	0.0	1.9	0.076	11.0	...
2	7.8	0.88	0.0	2.6	0.098	25.0	...
3	7.8	0.76	0.04	2.3	0.092	15.0	...
4	7.4	0.7	0.0	1.9	0.076	11.0	...
5	7.4	0.66	0.0	1.8	0.075	13.0	...
6	7.9	0.6	0.06	1.6	0.069	15.0	...
7	7.5	0.5	0.36	6.1	0.071	17.0	...
8	6.7	0.58	0.08	1.8	0.097	15.0	...
9	7.5	0.5	0.36	6.1	0.071	17.0	...
10	5.6	0.615	0.0	1.6	0.089	16.0	...
11	7.8	0.61	0.29	1.6	0.114	9.0	...
12	8.9	0.62	0.18	3.8	0.176	52.0	...
13	8.9	0.62	0.19	3.9	0.17	51.0	...
14	8.1	0.56	0.28	1.7	0.368	16.0	...
15	7.6	0.39	0.31	2.3	0.082	23.0	...
16	7.9	0.43	0.21	1.6	0.106	10.0	...
17	8.5	0.49	0.11	2.3	0.084	9.0	...
18	6.3	0.39	0.16	1.4	0.08	11.0	...
19	7.6	0.41	0.24	1.8	0.08	4.0	...
20	7.9	0.43	0.21	1.6	0.106	10.0	...
21	7.1	0.71	0.0	1.9	0.08	14.0	...
22	6.7	0.675	0.07	2.4	0.089	17.0	...
23	8.3	0.655	0.12	2.3	0.083	15.0	...
24	5.2	0.32	0.25	1.8	0.103	13.0	...
25	7.3	0.45	0.36	5.9	0.074	12.0	...
26	7.3	0.45	0.36	5.9	0.074	12.0	...
27	8.1	0.66	0.22	2.2	0.069	9.0	...
28	6.8	0.67	0.02	1.8	0.05	5.0	...
29	7.7	0.935	0.43	2.2	0.114	22.0	...
30	8.7	0.29	0.52	1.6	0.113	12.0	...
...	...	...	...	...	...	...	

...

Last Group (10 rows): quality = 3

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	
	Float64	Float64	Float64	Float64	Float64	Float64	
1	11.6	0.58	0.66	2.2	0.074	10.0	...
2	10.4	0.61	0.49	2.1	0.2	5.0	...
3	7.4	1.185	0.0	4.25	0.097	5.0	...
4	10.4	0.44	0.42	1.5	0.145	34.0	...
5	8.3	1.02	0.02	3.4	0.084	6.0	...
6	7.6	1.58	0.0	2.1	0.137	5.0	...
7	6.8	0.815	0.0	1.2	0.267	16.0	...
8	7.3	0.98	0.05	2.1	0.061	20.0	...
9	7.1	0.875	0.05	5.7	0.082	3.0	...
10	6.7	0.76	0.02	1.8	0.078	6.0	...

[26]:
```julia
# podsumowanie ile jest win w każdej grupie
combine(wine_grouped, nrow)
```

[26]:

	quality	nrow
	Int64	Int64
1	5	681
2	6	638
3	7	199
4	4	53
5	8	18
6	3	10

[27]:
```julia
combine(wine_grouped, "fixed acidity" => mean)
```

[27]:

	quality	fixed acidity_mean
	Int64	Float64
1	5	8.16725
2	6	8.34718
3	7	8.87236
4	4	7.77925
5	8	8.56667
6	3	8.36

[28]:
```julia
# zliczenie liczby win o danej jakości i zawartości alkoholu, posortowane
wine_grouped2=sort(combine(groupby(mydata, [:quality,:alcohol]),nrow=> :
↪liczba), [:quality,:alcohol])
```

[28]:

	quality	alcohol	liczba
	Int64	Float64	Int64
1	3	8.4	1
2	3	9.0	1
3	3	9.7	1
4	3	9.8	1
5	3	9.9	1
6	3	9.95	1
7	3	10.2	1
8	3	10.7	1
9	3	10.9	1
10	3	11.0	1
11	4	9.0	2
12	4	9.05	1
13	4	9.1	2
14	4	9.2	3
15	4	9.3	2
16	4	9.4	2
17	4	9.6	6
18	4	9.7	2
19	4	9.8	3
20	4	9.9	1
21	4	10.0	4
22	4	10.1	1
23	4	10.3	1
24	4	10.4	3
25	4	10.5	1
26	4	10.9	3
27	4	11.0	4
28	4	11.1	1
29	4	11.2	3
30	4	11.3	1
...	...	...	...

[29]:
```
# zapis do pliku
CSV.write("dataframe1.csv", wine_grouped2)
```

[29]: "dataframe1.csv"

1.1.2 Graficzna reprezentacja DataFrames

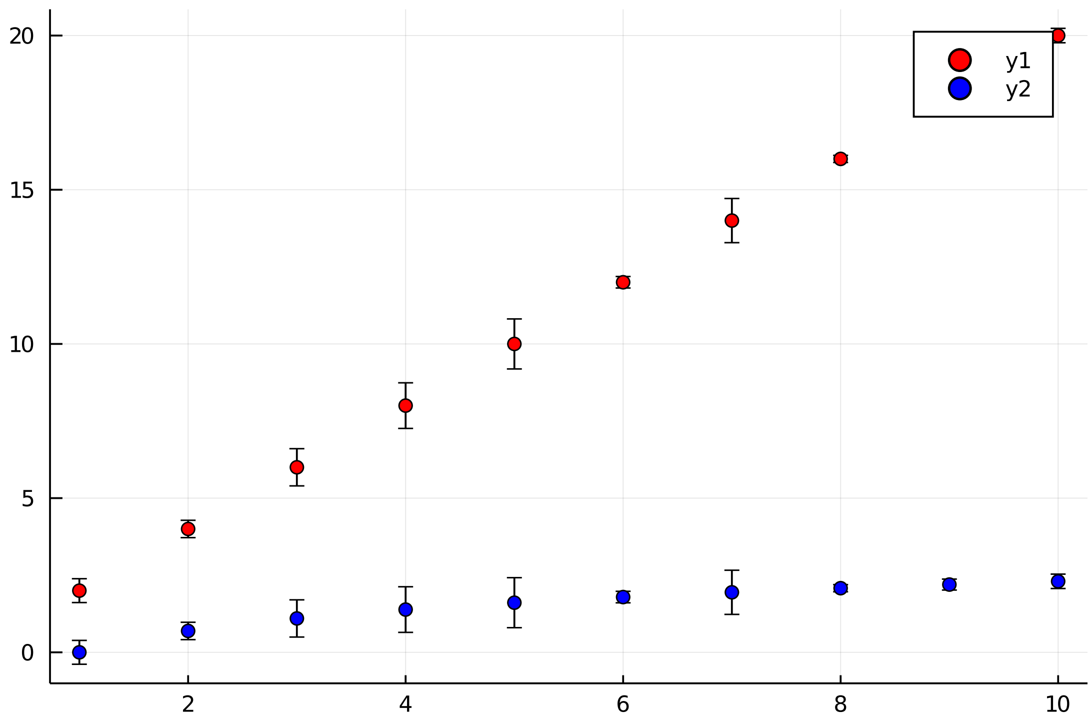
[30]:
```
using DataFrames
df = DataFrame(a = 1:10, b = map(x->2x,(1:10)), c = map(x->log(x),(1:10)),
 ↪d=rand(10), e=map(x->x%2,(1:10)))
```

[30]:

	a	b	c	d	e
	Int64	Int64	Float64	Float64	Int64
1	1	2	0.0	0.386491	1
2	2	4	0.693147	0.281443	0
3	3	6	1.09861	0.603057	1
4	4	8	1.38629	0.73946	0
5	5	10	1.60944	0.810854	1
6	6	12	1.79176	0.186859	0
7	7	14	1.94591	0.716171	1
8	8	16	2.07944	0.116305	0
9	9	18	2.19722	0.177662	1
10	10	20	2.30259	0.233535	0

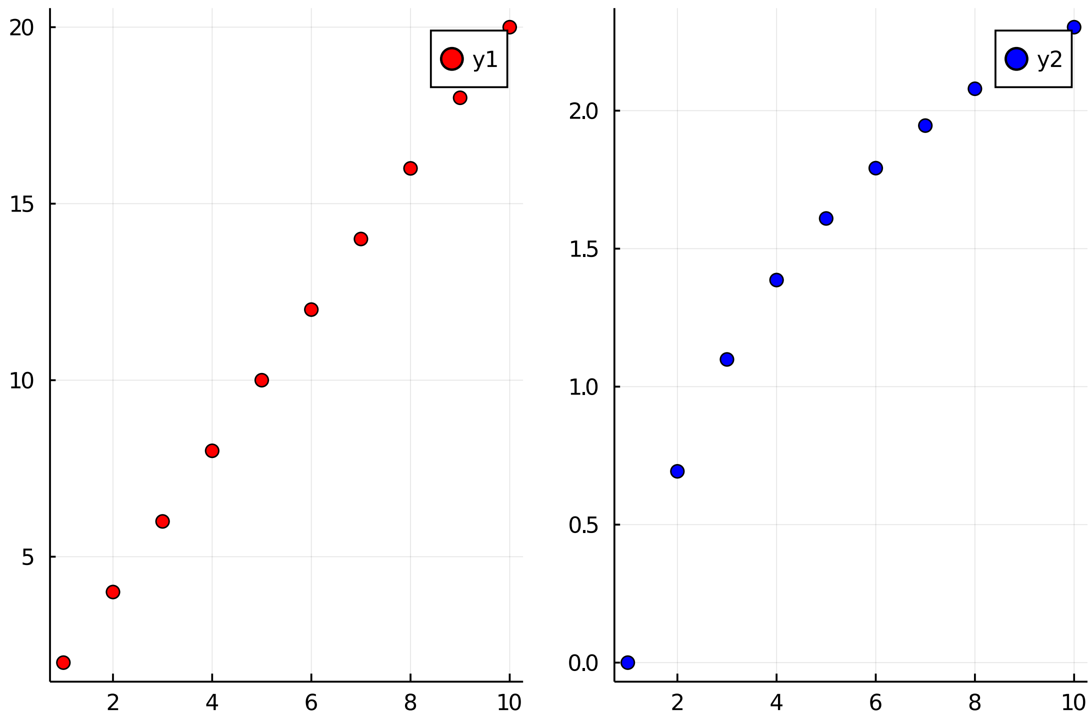
[31]:
```julia
# Mozna odwolywac sie bezposrednio do kolumn w poniższy sposób (gdyz sa typu␣
↪jednowymiarowych tablic
# Array{T,1})

using Plots
scatter(df.a, [df.b, df.c], colour = [:red :blue], yerr=df.d)
```

[31]:



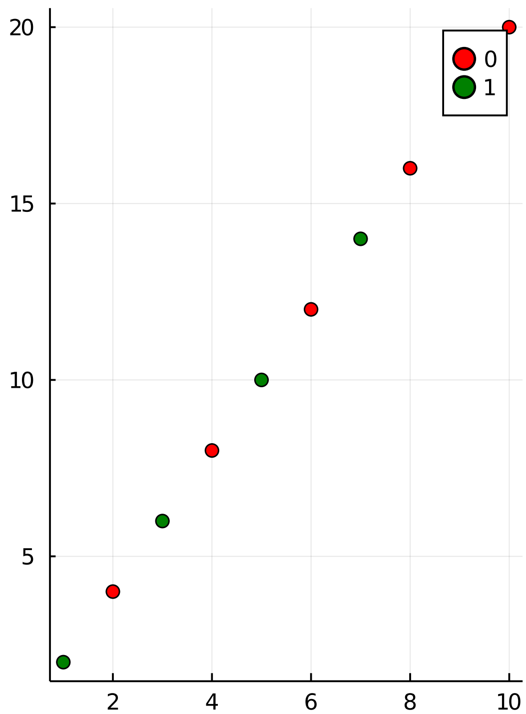
[32]:
```julia
# użycie opcji layout
scatter(df.a, [df.b, df.c],  colour = [:red :blue ], layout=2)
```

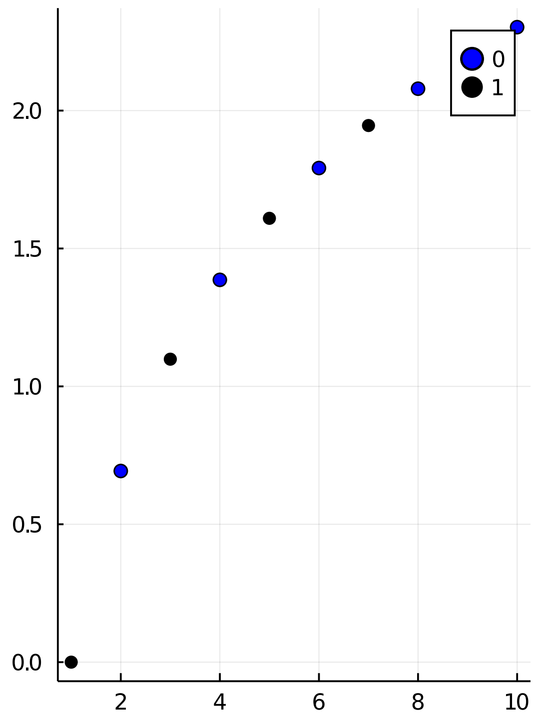
[32]:



[33]:
```
# użycie opcji layout i grup
scatter(df.a, [df.b, df.c], group=df.e,  colour = [:red :blue :green :black],␣
↪layout=2)
```

[33]:





1.1.3 Zadanie

- Napisać program w języku Julia do obliczania iloczynu skalarnego wektorów (LinearAlgebra.dot) i mnożenia macierzy kwadratowej przez wektor z użyciem operatora *.
- Uruchomić i zmierzyć czasy działania obydwu funkcji - każdej dla różnych rozmiarów wektorów. Dokonać 10 pomiarów dla każdego rozmiaru wektora.
- Czasy działania powinny być zapisywane do jednego pliku CSV. Należy zaplanować odpowiednią strukturę kolumn takiego pliku.
- Wczytać dane z w/w pliku do jednego DataFrame w języku Julia.
- Korzystając z mechanizmów DataFrame w języku Julia obliczyć średnią i odchylenie standardowe, w taki sposób, aby narysować wykresy średnich czasów obliczenia operacji w zależności od rozmiaru wektora. Dodać słupki błędów do obydwu wykresów uzyskanych z obliczenia odchylenia standardowego.
- Proszę poukładac wykresy w tabelkę za pomocą opcji layouts: https://docs.juliaplots.org/latest/layouts/#Simple-Layouts-1.
- Należy zadbać o staranne podpisanie osi i wykresów.

1.1.4 Rozwiązanie

Funkcje:

[34]:
```
using LinearAlgebra
```

[35]:
```julia
function dot_product(x,y)
    return dot(x,y)
end
```

[35]: dot_product (generic function with 1 method)

[36]:
```julia
x = [1,2,3]
y = [3,4,5]
dot_product(x,y)
```

[36]: 26

[37]:
```julia
function scalar_times_matrix(scalar, m)
    return scalar * m
end
```

[37]: scalar_times_matrix (generic function with 1 method)

[38]:
```julia
m1 = [1 5 ; 3 2]
scalar1 = 6
scalar_times_matrix(scalar1, m1)
```

[38]:
```
2×2 Array{Int64,2}:
  6  30
 18  12
```

time tests:

[39]:
```julia
@time dot_product(x,y)
@time scalar_times_matrix(scalar1, m1)
```

```
  0.000003 seconds
  0.000003 seconds (1 allocation: 112 bytes)
```

[39]:
```
2×2 Array{Int64,2}:
  6  30
 18  12
```

[40]:
```julia
rand(10)
```

[40]:
```
10-element Array{Float64,1}:
 0.8558386026774496
 0.704399624415722
 0.3280155865558867
 0.9379351938664204
 0.9773428947566583
 0.7842185646424709
 0.7018560624340788
```

```
 0.02403169856818277
 0.855607147160194
 0.04489338134714993
```

[44]:
```julia
function count_dot_product_timetest(range)
    dot_p_time = []
    for i=range
    #     println(i)
        x1 = rand(i)
        y1 = rand(i)
        push!(dot_p_time, @elapsed dot_product(x1,y1))
    end
    return dot_p_time
end
```

[44]:
```
count_dot_product_timetest (generic function with 1 method)
```

[45]:
```julia
count_dot_product_timetest(1000:1000:10000)
```

[45]:
```
10-element Array{Any,1}:
 1.67e-6
 3.6e-7
 6.1e-7
 8.5e-7
 9.1e-7
 1.04e-6
 1.17e-6
 1.31e-6
 1.44e-6
 1.62e-6
```

[46]:
```julia
function count_scalar_times_matrix_timetest(range)
    times = []
    for i=range
        m = [ rand(i) for j=1:i ]
#         println(m)
        push!( times, @elapsed scalar_times_matrix(rand(), m))
    end
    return times
end
```

[46]:
```
count_scalar_times_matrix_timetest (generic function with 1 method)
```

[47]:
```julia
count_scalar_times_matrix_timetest(5:5:5)
```

[47]:
```
1-element Array{Any,1}:
 0.071297963
```

[48]:
```
count_scalar_times_matrix_timetest(1000:1000:10000)
```

[48]:
```
10-element Array{Any,1}:
 0.016134337
 0.013399577
 0.018280244
 0.063499102
 0.174219639
 0.213000473
 0.187807184
 0.125771638
 0.186799449
 0.169232097
```

Building a data frame

[52]:
```
interval  = 1000 #start and interval
range = interval:interval:10*interval
timetest_data = DataFrame(length = range,
    dot = count_dot_product_timetest(range),
    scalar = count_scalar_times_matrix_timetest(range))
```

[52]:

	length	dot	scalar
	Int64	Any	Any
1	1000	3.64e-6	0.00225909
2	2000	1.52e-6	0.00506711
3	3000	2.21e-6	0.0122022
4	4000	2.56e-6	0.0263606
5	5000	3.42e-6	0.029506
6	6000	3.78e-6	0.0401959
7	7000	4.49e-6	0.0834425
8	8000	4.87e-6	0.193702
9	9000	5.75e-6	0.208038
10	10000	6.3e-6	0.182438

[53]:
```
for i=1:9
    tmp = DataFrame(length = range,
    dot = count_dot_product_timetest(range),
    scalar = count_scalar_times_matrix_timetest(range))
    append!(timetest_data,tmp)
end
```

[54]:
```
timetest_data
```

[54]:

	length	dot	scalar
	Int64	Any	Any
1	1000	3.64e-6	0.00225909
2	2000	1.52e-6	0.00506711
3	3000	2.21e-6	0.0122022
4	4000	2.56e-6	0.0263606
5	5000	3.42e-6	0.029506
6	6000	3.78e-6	0.0401959
7	7000	4.49e-6	0.0834425
8	8000	4.87e-6	0.193702
9	9000	5.75e-6	0.208038
10	10000	6.3e-6	0.182438
11	1000	1.48e-6	0.0015613
12	2000	3.7e-7	0.00495143
13	3000	6.1e-7	0.0106986
14	4000	7.4e-7	0.0181781
15	5000	9.2e-7	0.0384263
16	6000	1.03e-6	0.316284
17	7000	1.22e-6	0.181065
18	8000	1.33e-6	0.238194
19	9000	1.49e-6	0.150924
20	10000	1.59e-6	0.199682
21	1000	2.03e-6	0.00228895
22	2000	3.6e-7	0.00511193
23	3000	5.9e-7	0.0109547
24	4000	6.3e-7	0.0184792
25	5000	8.7e-7	0.0287421
26	6000	1.001e-6	0.0526848
27	7000	1.13e-6	0.060837
28	8000	1.28e-6	0.0792995
29	9000	1.45e-6	0.0934194
30	10000	1.59e-6	0.19632
…	…	…	…

Save to file

[55]:
```
CSV.write("timetests.csv", timetest_data)
```

[55]: "timetests.csv"

Read from file

[56]:
```
my_timetest_data = CSV.read("timetests.csv", delim=",",DataFrame)
```

[56]:

	length	dot	scalar
	Int64	Float64	Float64
1	1000	3.64e-6	0.00225909
2	2000	1.52e-6	0.00506711
3	3000	2.21e-6	0.0122022
4	4000	2.56e-6	0.0263606
5	5000	3.42e-6	0.029506
6	6000	3.78e-6	0.0401959
7	7000	4.49e-6	0.0834425
8	8000	4.87e-6	0.193702
9	9000	5.75e-6	0.208038
10	10000	6.3e-6	0.182438
11	1000	1.48e-6	0.0015613
12	2000	3.7e-7	0.00495143
13	3000	6.1e-7	0.0106986
14	4000	7.4e-7	0.0181781
15	5000	9.2e-7	0.0384263
16	6000	1.03e-6	0.316284
17	7000	1.22e-6	0.181065
18	8000	1.33e-6	0.238194
19	9000	1.49e-6	0.150924
20	10000	1.59e-6	0.199682
21	1000	2.03e-6	0.00228895
22	2000	3.6e-7	0.00511193
23	3000	5.9e-7	0.0109547
24	4000	6.3e-7	0.0184792
25	5000	8.7e-7	0.0287421
26	6000	1.001e-6	0.0526848
27	7000	1.13e-6	0.060837
28	8000	1.28e-6	0.0792995
29	9000	1.45e-6	0.0934194
30	10000	1.59e-6	0.19632
...	...	...	...

[57]:
```
names(my_timetest_data)
```

[57]:
```
3-element Array{String,1}:
 "length"
 "dot"
 "scalar"
```

[58]:
```
describe(my_timetest_data)
```

[58]:

	variable	mean	min	median	max	nmissing	eltype
	Symbol	Float64	Real	Float64	Real	Int64	DataType
1	length	5500.0	1000	5500.0	10000	0	Int64
2	dot	1.39462e-6	3.5e-7	1.195e-6	6.3e-6	0	Float64
3	scalar	0.0805247	0.0015613	0.0407524	0.316284	0	Float64

[59]:
```
sort(my_timetest_data)
```

[59]:

	length	dot	scalar
	Int64	Float64	Float64
1	1000	1.48e-6	0.0015613
2	1000	1.94e-6	0.00174932
3	1000	1.96e-6	0.00172581
4	1000	1.98e-6	0.00165277
5	1000	1.99e-6	0.00172213
6	1000	1.99e-6	0.00179711
7	1000	2.01e-6	0.0016456
8	1000	2.03e-6	0.00228895
9	1000	2.19e-6	0.00166728
10	1000	3.64e-6	0.00225909
11	2000	3.5e-7	0.00516651
12	2000	3.6e-7	0.00511193
13	2000	3.7e-7	0.00495143
14	2000	3.7e-7	0.00502901
15	2000	3.7e-7	0.00505884
16	2000	3.7e-7	0.00513876
17	2000	3.8e-7	0.00501253
18	2000	3.8e-7	0.005056
19	2000	3.8e-7	0.00511957
20	2000	1.52e-6	0.00506711
21	3000	5.9e-7	0.0109547
22	3000	6.0e-7	0.0108949
23	3000	6.1e-7	0.0106986
24	3000	6.1e-7	0.0109282
25	3000	6.1e-7	0.0109441
26	3000	6.1e-7	0.135208
27	3000	6.2e-7	0.0108007
28	3000	6.3e-7	0.0107364
29	3000	6.3e-7	0.0109288
30	3000	2.21e-6	0.0122022
…	…	…	…

Group by length

[60]:
```
my_data_grouped = groupby(my_timetest_data, :length)
```

[60]:

GroupedDataFrame with 10 groups based on key: length

First Group (10 rows): length = 1000

	length	dot	scalar
	Int64	Float64	Float64
1	1000	3.64e-6	0.00225909
2	1000	1.48e-6	0.0015613
3	1000	2.03e-6	0.00228895
4	1000	1.99e-6	0.00179711
5	1000	1.99e-6	0.00172213
6	1000	1.98e-6	0.00165277
7	1000	2.01e-6	0.0016456
8	1000	1.96e-6	0.00172581
9	1000	1.94e-6	0.00174932
10	1000	2.19e-6	0.00166728

…

Last Group (10 rows): length = 10000

	length	dot	scalar
	Int64	Float64	Float64
1	10000	6.3e-6	0.182438
2	10000	1.59e-6	0.199682
3	10000	1.59e-6	0.19632
4	10000	1.59e-6	0.220918
5	10000	1.58e-6	0.191838
6	10000	1.52e-6	0.275361
7	10000	1.52e-6	0.190851
8	10000	1.55e-6	0.20593
9	10000	1.53e-6	0.272087
10	10000	1.5e-6	0.191683

Add means and standard deviation columns

[74]:
```
data_to_plot = combine(my_data_grouped, "dot" => mean, "dot" => std,
        "scalar" => mean, "scalar" => std)
```

[74]:

	length	dot_mean	dot_std	scalar_mean	scalar_std
	Int64	Float64	Float64	Float64	Float64
1	1000	2.121e-6	5.63372e-7	0.00180694	0.000254644
2	2000	4.85e-7	3.63784e-7	0.00507117	6.46483e-5
3	3000	7.72e-7	5.05411e-7	0.0234296	0.0392771
4	4000	9.15e-7	5.79296e-7	0.0227744	0.0101443
5	5000	1.148e-6	7.99136e-7	0.0303074	0.00321438
6	6000	1.2931e-6	8.74784e-7	0.0759262	0.0848919
7	7000	1.521e-6	1.04349e-6	0.15205	0.0973346
8	8000	1.773e-6	1.09367e-6	0.136518	0.0631795
9	9000	1.8911e-6	1.35602e-6	0.144653	0.0580555
10	10000	2.027e-6	1.50176e-6	0.212711	0.0337879

[75]:
```
data_to_plot
```

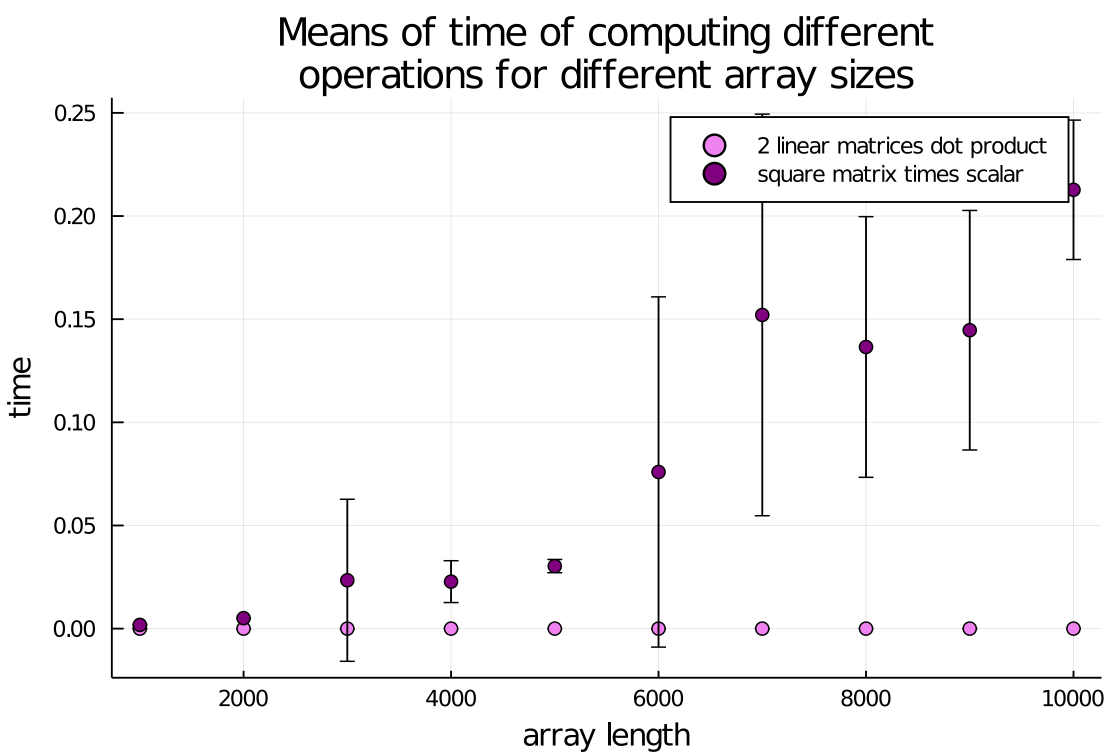
[75]:

	length	dot_mean	dot_std	scalar_mean	scalar_std
	Int64	Float64	Float64	Float64	Float64
1	1000	2.121e-6	5.63372e-7	0.00180694	0.000254644
2	2000	4.85e-7	3.63784e-7	0.00507117	6.46483e-5
3	3000	7.72e-7	5.05411e-7	0.0234296	0.0392771
4	4000	9.15e-7	5.79296e-7	0.0227744	0.0101443
5	5000	1.148e-6	7.99136e-7	0.0303074	0.00321438
6	6000	1.2931e-6	8.74784e-7	0.0759262	0.0848919
7	7000	1.521e-6	1.04349e-6	0.15205	0.0973346
8	8000	1.773e-6	1.09367e-6	0.136518	0.0631795
9	9000	1.8911e-6	1.35602e-6	0.144653	0.0580555
10	10000	2.027e-6	1.50176e-6	0.212711	0.0337879

[76]:

```
using Plots
scatter(data_to_plot.length,
    [data_to_plot.dot_mean data_to_plot.scalar_mean],
    colour = [:violet :purple ],
    yerr= [data_to_plot.dot_std data_to_plot.scalar_std],
    label = ["2 linear matrices dot product" "square matrix times scalar"],
    title = "Means of time of computing different
operations for different array sizes",
    xlab = "array length",
    ylab = "time"
)
```

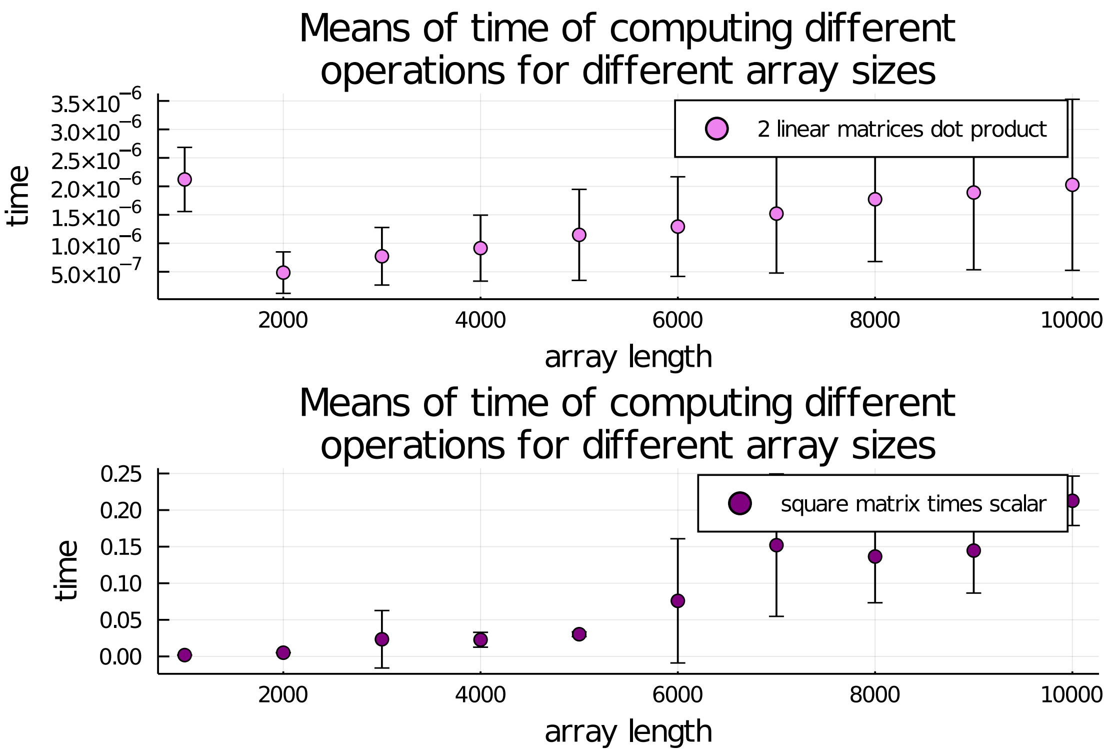
[76]:



[77]:

```
using Plots
scatter(data_to_plot.length,
    [data_to_plot.dot_mean data_to_plot.scalar_mean],
    colour = [:violet :purple ],
    yerr= [data_to_plot.dot_std data_to_plot.scalar_std],
    label = ["2 linear matrices dot product" "square matrix times scalar"],
    title = "Means of time of computing different
operations for different array sizes",
    xlab = "array length",
    ylab = "time",
    layout = (2,1)
)
```

[77]:



Wykresy potwierdziły moje przypuszczenia. Jak widać na powyższych wykresach, czas obliczania iloczynu skalarnego 2-ch macierzy rośnie liniowo względem długości macierzy, natomiast obliczanie iloczynu macierzy kwadratowej przez skalar rośnie kwadratowo względem długości boku macierzy.

[ ]:

[ ]:

[ ]: