Laboratorium 3

In [31]:
```
#tablice wielowymiarowe w Julii
Asmall=[[1.0 0.0 10]; [0.0 1.0 10]]
Bsmall=Asmall
Asmall
C = zeros(Float64, size(Asmall,1), size(Asmall,2))
```

Out[31]:
```
2×3 Array{Float64,2}:
 0.0  0.0  0.0
 0.0  0.0  0.0
```

In [26]:
```
size(Asmall)
```

Out[26]:
```
(2, 3)
```

In [14]:
```
# mnożenie macierzy - wersja naiwna, naiwna przez sprosob dostepu do
 pamieci
# najlepiej zeby dane byly blisko siebie w pamieci, przez sposob dzia
lania pamieci cache, ktora odczytuje dane
# blokami zwanymi liniami cache. Odczytywanie obok siebie jest duzo s
zybsze.
# Pytniem jest jak jest tablica przechowywana w pamieci, zazwyczaj wi
erszami (C) ale w julli sa przechowywane
# kolumnami
function naive_multiplication(A,B)
C=zeros(Float64,size(A,1),size(B,2))
  for i=1:size(A,1)
    for j=1:size(B,2)
        for k=1:size(A,2)
            C[i,j]=C[i,j]+A[i,k]*B[k,j]
        end
    end
end
C
end
```

Out[14]:
```
naive_multiplication (generic function with 1 method)
```

In [4]:
```
#kompilacja
naive_multiplication(Asmall,Bsmall)
```

Out[4]:
```
2×2 Array{Float64,2}:
 1.0  0.0
 0.0  1.0
```

In [5]:
```
#kompilacja funkcji BLASowej do mnożenia macierzy
#https://docs.julialang.org/en/stable/stdlib/linalg/#BLAS-Functions-1
#to tez jest mnozenie macierzy ale zoptymalizowanymi funkcjami BLAS
Asmall*Bsmall
```

Out[5]:
```
2×2 Array{Float64,2}:
 1.0  0.0
 0.0  1.0
```

In [6]:
```
A=rand(1000,1000); #tworzenie macierzy 1000 x 1000 z losowymi wartosc
iami
B=rand(1000,1000);
```

In [7]:
```
# Należy pamiętać o "column-major" dostępie do tablic -
# pierwszy indeks zmienia się szybciej
# tak jak Matlab, R, Fortran
# inaczej niz C, Python
A1 = [[1 2]; [3 4]]
vec(A1)
```

Out[7]:
```
4-element Array{Int64,1}:
 1
 3
 2
 4
```

In [15]:
```
# poprawiona funkcja korzytająca z powyższego oraz z faktu, że
#można zmieniać kolejność operacji dodawania (a co za tym idzie kolej
nosc petli).
# jest lepsza przez zamienienie kolejnosci petli i czesciej odczytuje
elementy bedace blizej siebie
function better_multiplication( A,B )
C=zeros(Float64,size(A,1),size(B,2))
  for j=1:size(B,2)
    for k=1:size(A,2)
        for i=1:size(A,1)
            C[i,j]=C[i,j]+A[i,k]*B[k,j]
        end
    end
end
C
end
```

Out[15]:
```
better_multiplication (generic function with 1 method)
```

In [9]:
```
better_multiplication(Asmall, Bsmall)
```

Out[9]:
```
2×2 Array{Float64,2}:
 1.0  0.0
 0.0  1.0
```

In [10]:
```
@elapsed naive_multiplication(A,B) #mierzenie czasu
```

Out[10]:
```
3.69591414
```

In [11]:
```
@elapsed better_multiplication(A,B)
```

Out[11]: 1.600793835

In [12]:
```
@elapsed A*B #blas
```

Out[12]: 0.016316159

In [59]:
```
# aproksymacja sredniokwadratowa wielomianem - tutaj przyklad dla wie
lomianu 3 stopnia
# pakiet Polynomials jest mozliwy do instalacji pod Juliabox
# https://github.com/JuliaMath/Polynomials.jl


using Polynomials
xs = 0:3; ys = [1,3,4,5]
fit1=polyfit(xs, ys,3)

# po prostu za xs podstawic to co mam za wielkosc macierzy a za ys po
dstawiac po kolei te zmienne dla których
# chce wyliczyc wielomian
```

Out[59]: 1.0 + 2.8333333333333335·x - 0.9999999999999999·x^2 + 0.16666666666666663·x^3

In [14]:
```
# obliczanie wartosci wielomianu
fit1(1)
```

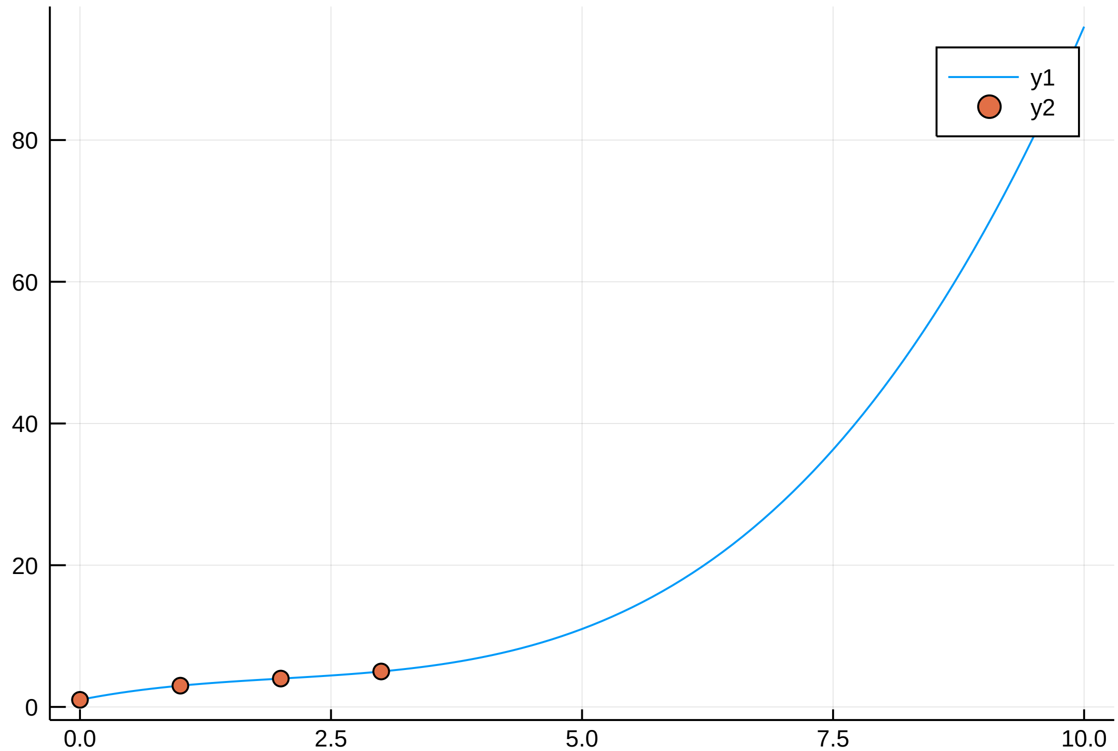
Out[14]: 836.4071935534389

In [15]:
```
# obliczanie wartosci wielomianu (drugi sposób)
polyval(fit1, 1)
```

Out[15]: 836.4071935534389

In [60]:
```
using Plots

# geste punkty do wyliczenia wartosci wielomianu aproksymujacego:
xd=0:0.1:10
# wykres wartosci wielomianu dla gestych punktow:
plot(xd,polyval(fit1, xd))

# ! -dodanie do tego samego wykresu punktów wg ktorych aproksymowalis
my
scatter!(xs,ys)
```

Out[60]:



Zadania

1.Uruchomić

- naive_multiplication(A,B),
- better_multiplication(A,B)
- mnożenie BLAS w Julii (A*B)

dla coraz większych macierzy i zmierzyć czasy. Narysować wykres zależyności czasu od rozmiaru macierzy wraz z słupkami błędów, tak jak na poprzednim laboratorium. Wszystkie trzy metody powinny być na jednym wykresie.

2.Napisać w języku C:

```
- naiwną metodę mnożenia macierzy (wersja 1)
- ulepszoną za pomocą zamiany pętli metodę mnożenia macierzy (wersja 2), pam
iętając, że w C macierz przechowywana jest wierszami (row major order tzn A1
1,A12, ..., A1m, A21, A22,...,A2m, ..Anm), inaczej niż w Julii !
- skorzystać z  możliwości BLAS dostępnego w GSL(wersja 3).
```

Należy porównywać działanie tych trzech algorytmow bez włączonej opcji optymalizacji kompilatora. Przedstawić wyniki na jednym wykresie tak jak w p.1.(osobno niż p.1). (Dla chętnych) sprawdzić, co się dzieje, jak włączymy optymalizację kompilatora i dodać do wykresu.

3.Użyć funkcji polyfit z pakietu Polynomials do znalezienia odpowiednich wielomianow, ktore najlepiej pasują do zależności czasowych kazdego z algorytmow. Stopień wielomianu powinien zgadzać się z teoretyczną złożonoscią. Dodać wykresy uzyskanych wielomianow do wczesniejszych wykresów.

In [16]:

```
columns_and_rows = Int64[]
naive_time = Float64[]
better_time = Float64[]
blas_time = Float64[]

nb_of_tests = 10
i = 50
while(i <= 1000)
    for k=0:nb_of_tests
        A = rand(i,i)
        B = rand(i,i)
        push!(columns_and_rows, i)
        push!(naive_time,@elapsed naive_multiplication(A,B))
        push!(better_time,@elapsed better_multiplication(A,B))
        push!(blas_time,@elapsed A * B)
    end
    i += 50
end
columns_and_rows
```

Out[16]:

```
220-element Array{Int64,1}:
   50
   50
   50
   50
   50
   50
   50
   50
   50
   50
   50
  100
  100
    ⋮
  950
 1000
 1000
 1000
 1000
 1000
 1000
 1000
 1000
 1000
 1000
 1000
```

In [18]:
```
using DataFrames

df = DataFrame()
df[:columns_and_rows]= columns_and_rows
df[:naive_time] = naive_time
df[:better_time] = better_time
df[:blas_time] = blas_time
```

Out[18]:
```
220-element Array{Float64,1}:
 0.445678423
 2.42e-5
 1.8701e-5
 2.19e-5
 2.24e-5
 1.99e-5
 2.24e-5
 2.21e-5
 4.9201e-5
 2.08e-5
 2.22e-5
 0.000635004
 8.0501e-5
 ⋮
 0.011140474
 0.017180114
 0.015388102
 0.013482489
 0.021169041
 0.013324388
 0.013693191
 0.016768711
 0.013379489
 0.014136793
 0.01671251
 0.01357489
```

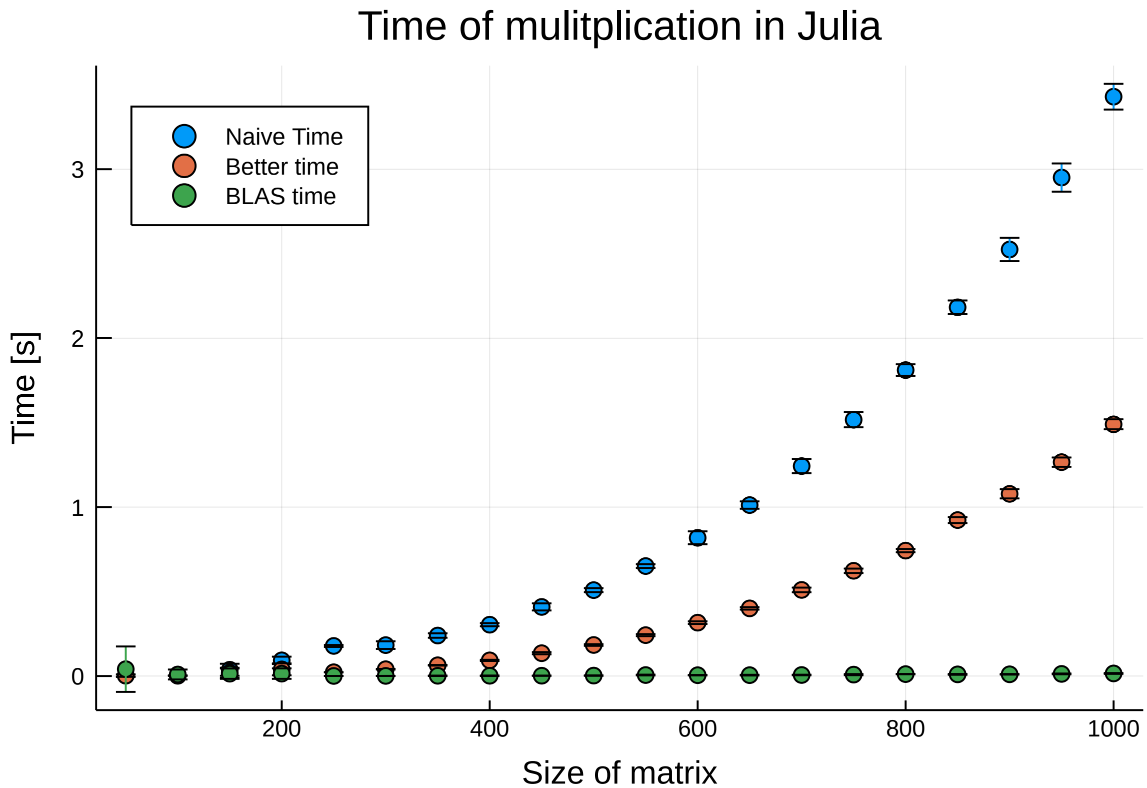
In [19]:
```
using Statistics
df2 = DataFrame(by(df, [:columns_and_rows],
    :naive_time => mean,
    :naive_time => std,
    :better_time => mean,
    :better_time => std,
    :blas_time => mean,
    :blas_time => std))
```

Out[19]: 20 rows × 7 columns

	columns_and_rows	naive_time_mean	naive_time_std	better_time_mean	better_time_std	b
	Int64	Float64	Float64	Float64	Float64	
1	50	0.00258037	0.00713601	0.00233655	0.00696521	
2	100	0.00333974	9.22051e-5	0.00157779	6.40788e-5	
3	150	0.0371351	0.0355213	0.0191812	0.0298498	
4	200	0.0928441	0.0216986	0.0392809	0.0352772	
5	250	0.178135	0.00578942	0.0223672	0.000658748	
6	300	0.18299	0.0225436	0.0402549	0.00198967	
7	350	0.239635	0.0131353	0.0637023	0.00196339	
8	400	0.30382	0.00926658	0.0926878	0.00341562	
9	450	0.408957	0.0209303	0.135173	0.00649957	
10	500	0.508545	0.0118891	0.183753	0.00489786	
11	550	0.65091	0.0111029	0.242095	0.00608552	
12	600	0.818139	0.0380825	0.315904	0.00763555	
13	650	1.01197	0.0216512	0.400414	0.00730541	
14	700	1.24273	0.0426229	0.509718	0.0136065	
15	750	1.517	0.0448517	0.622955	0.0129383	
16	800	1.81098	0.0342295	0.742372	0.0100373	
17	850	2.18262	0.0405781	0.923156	0.0174981	
18	900	2.52477	0.0691341	1.07829	0.0271135	
19	950	2.95079	0.0834888	1.2661	0.0274116	
20	1000	3.4293	0.076056	1.48992	0.0295836	

In [20]:
```
using Plots

ydata = scatter(df2[:columns_and_rows],
    [df2[:naive_time_mean] df2[:better_time_mean] df2[:blas_time_mean
]],
    yerr = [df2[:naive_time_std] df2[:better_time_std] df2[:blas_time
_std]],
    labels = ["Naive Time" "Better time" "BLAS time"],
    title = "Time of mulitplication in Julia",
    legend=:topleft,
    xlabel = "Size of matrix",
    ylabel = "Time [s]",)
```

Out[20]:



In [21]:

```
using CSV
input0="result00.csv"
mydata0=CSV.read(input0, delim=";")
input1="result01.csv"
mydata1=CSV.read(input1, delim=";")
input2="result02.csv"
mydata2=CSV.read(input2, delim=";")
input3="result03.csv"
mydata3=CSV.read(input3, delim=";")
```

Out[21]: 200 rows × 4 columns

	columns_and_rows	naive_time	better_time	blas_time
	Int64⍰	Float64⍰	Float64⍰	Float64⍰
1	50	9.8e-5	6.7e-5	9.7e-5
2	50	9.4e-5	6.3e-5	0.000103
3	50	9.3e-5	7.3e-5	9.5e-5
4	50	0.000104	6.2e-5	9.2e-5
5	50	9.3e-5	6.3e-5	0.000129
6	50	9.3e-5	6.2e-5	9.2e-5
7	50	9.3e-5	6.1e-5	9.1e-5
8	50	9.3e-5	6.4e-5	0.000112
9	50	9.3e-5	6.2e-5	0.000124
10	50	0.000154	7.6e-5	0.000114
11	100	0.000944	0.00047	0.000702
12	100	0.000847	0.000632	0.00102
13	100	0.001298	0.000695	0.001074
14	100	0.000966	0.000449	0.000664
15	100	0.000819	0.000472	0.000645
16	100	0.000816	0.000448	0.000669
17	100	0.000814	0.000437	0.00071
18	100	0.000896	0.000475	0.000754
19	100	0.000805	0.000504	0.000652
20	100	0.000813	0.000437	0.00067
21	150	0.002834	0.001525	0.002231
22	150	0.002914	0.001524	0.002104
23	150	0.002866	0.001468	0.002066
24	150	0.002745	0.00143	0.002067
25	150	0.00279	0.001467	0.002049
26	150	0.002752	0.001416	0.002069
27	150	0.002769	0.001419	0.002067
28	150	0.002757	0.001409	0.002065
29	150	0.002768	0.001419	0.002056
30	150	0.002798	0.001433	0.002046
⋮	⋮	⋮	⋮	⋮

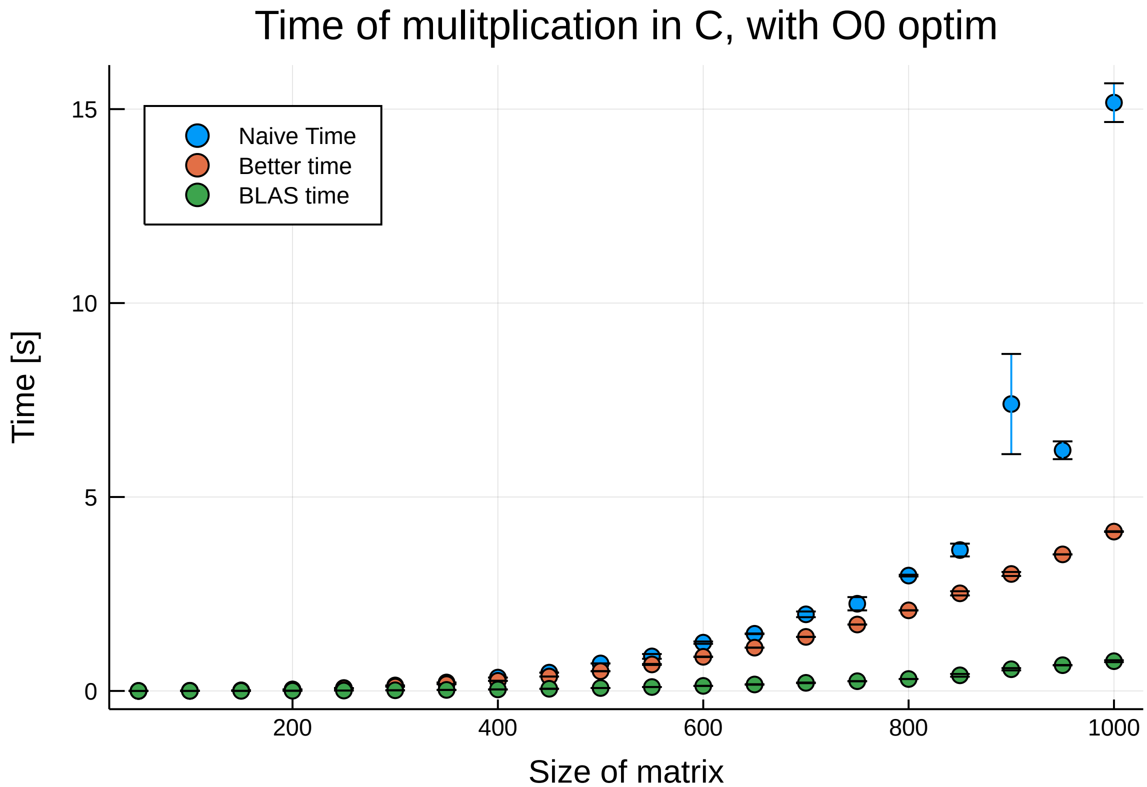
In [23]:
```
using Statistics, DataFrames, Plots
data0 = DataFrame(by(mydata0, [:columns_and_rows],
    :naive_time => mean,
    :naive_time => std,
    :better_time => mean,
    :better_time => std,
    :blas_time => mean,
    :blas_time => std))
data1 = DataFrame(by(mydata1, [:columns_and_rows],
    :naive_time => mean,
    :naive_time => std,
    :better_time => mean,
    :better_time => std,
    :blas_time => mean,
    :blas_time => std))
data2 = DataFrame(by(mydata2, [:columns_and_rows],
    :naive_time => mean,
    :naive_time => std,
    :better_time => mean,
    :better_time => std,
    :blas_time => mean,
    :blas_time => std))
data3 = DataFrame(by(mydata3, [:columns_and_rows],
    :naive_time => mean,
    :naive_time => std,
    :better_time => mean,
    :better_time => std,
    :blas_time => mean,
    :blas_time => std))
```

Out[23]: 20 rows × 7 columns

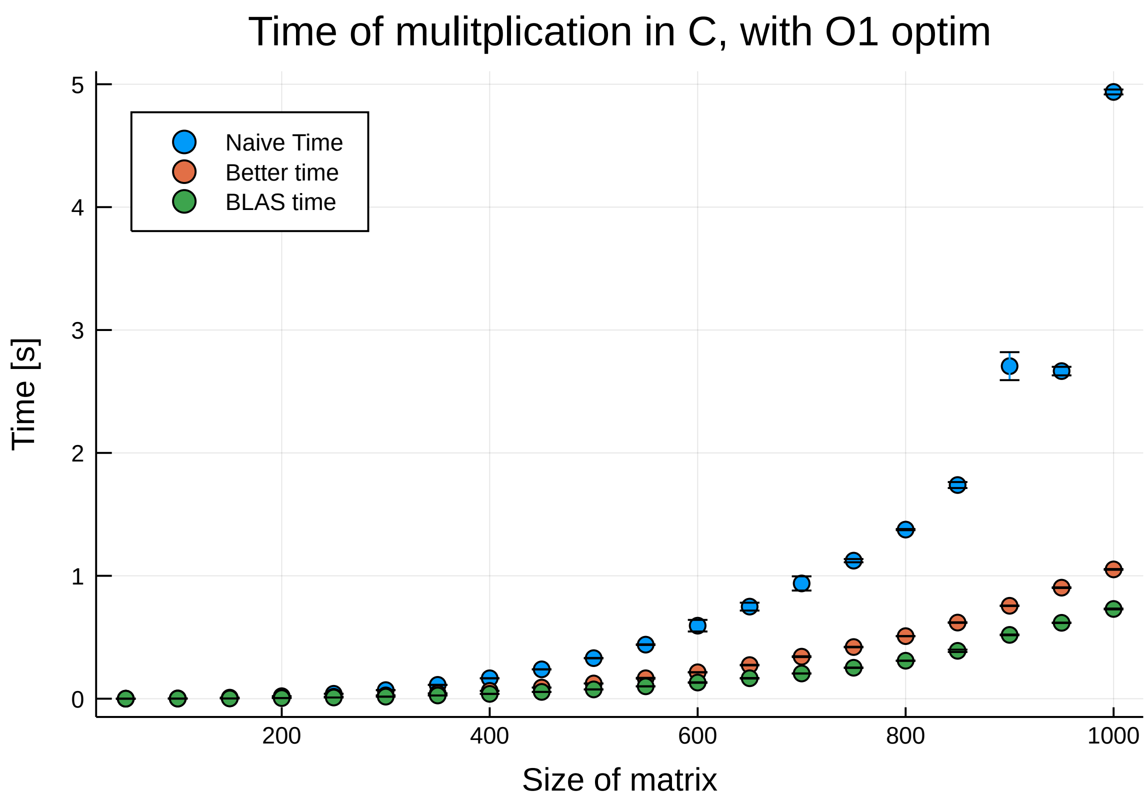
	columns_and_rows	naive_time_mean	naive_time_std	better_time_mean	better_time_std	b
	Int64⍰	Float64	Float64	Float64	Float64	
1	50	0.0001008	1.9031e-5	6.53e-5	5.16505e-6	
2	100	0.0009018	0.000150958	0.0005019	8.88075e-5	
3	150	0.0027993	5.55479e-5	0.001451	4.36552e-5	
4	200	0.0070292	6.93282e-5	0.0033728	1.62535e-5	
5	250	0.0138767	8.9614e-5	0.0065372	6.45838e-5	
6	300	0.0242617	0.0001352	0.0111669	8.02641e-5	
7	350	0.0434019	0.000519811	0.0176253	0.000202312	
8	400	0.059222	0.000271867	0.0258232	0.000176843	
9	450	0.0897198	0.00298483	0.0365432	0.000374439	
10	500	0.132688	0.000173886	0.050158	0.000104083	
11	550	0.189049	0.0218715	0.0663637	0.000542822	
12	600	0.239029	0.0207561	0.086083	0.00118062	
13	650	0.296271	0.00397298	0.109232	0.000735391	
14	700	0.407073	0.0560403	0.141404	0.00580082	
15	750	0.463916	0.0712514	0.173345	0.00794257	
16	800	0.562829	0.00434685	0.207849	0.000872308	
17	850	0.738729	0.0305803	0.266075	0.00505022	
18	900	1.54044	0.0614094	0.351571	0.00353188	
19	950	1.44931	0.11367	0.441002	0.00383281	
20	1000	3.43011	0.0996768	0.513976	0.00112116	

In [24]:
```
plotO0 = scatter(data0[:columns_and_rows],
    [data0[:naive_time_mean] data0[:better_time_mean] data0[:blas_tim
e_mean]],
    yerr = [data0[:naive_time_std] data0[:better_time_std] data0[:bla
s_time_std]],
    labels = ["Naive Time" "Better time" "BLAS time"],
    title = "Time of mulitplication in C, with O0 optim",
    legend=:topleft,
    xlabel = "Size of matrix",
    ylabel = "Time [s]")
plotO1 = scatter(data1[:columns_and_rows],
    [data1[:naive_time_mean] data1[:better_time_mean] data1[:blas_tim
e_mean]],
    yerr = [data1[:naive_time_std] data1[:better_time_std] data1[:bla
s_time_std]],
    labels = ["Naive Time" "Better time" "BLAS time"],
    title = "Time of mulitplication in C, with O1 optim",
    legend=:topleft,
    xlabel = "Size of matrix",
    ylabel = "Time [s]")
plotO2 = scatter(data2[:columns_and_rows],
    [data2[:naive_time_mean] data2[:better_time_mean] data2[:blas_tim
e_mean]],
    yerr = [data2[:naive_time_std] data2[:better_time_std] data2[:bla
s_time_std]],
    labels = ["Naive Time" "Better time" "BLAS time"],
    title = "Time of mulitplication in C, with O2 optim",
    legend=:topleft,
    xlabel = "Size of matrix",
    ylabel = "Time [s]")

plotO3 = scatter(data3[:columns_and_rows],
    [data3[:naive_time_mean] data3[:better_time_mean] data3[:blas_tim
e_mean]],
    yerr = [data3[:naive_time_std] data0[:better_time_std] data3[:bla
s_time_std]],
    labels = ["Naive Time" "Better time" "BLAS time"],
    title = "Time of mulitplication in C, with O3 optim",
    legend=:topleft,
    xlabel = "Size of matrix",
    ylabel = "Time [s]")


naive_compare = scatter([data0[:columns_and_rows] data1[:columns_and_
rows] data2[:columns_and_rows]],
    [data0[:naive_time_mean]
     data1[:naive_time_mean]
     data2[:naive_time_mean] ],
    yerr =
    [data0[:naive_time_std]
     data1[:naive_time_std]
     data2[:naive_time_std] ],
    labels = ["Naive Time O0" "Naive Time O1" "Naive Time O2"],
    title = "naive in optimalization dependency",
    legend=:topleft,
    xlabel = "Size of matrix",
```

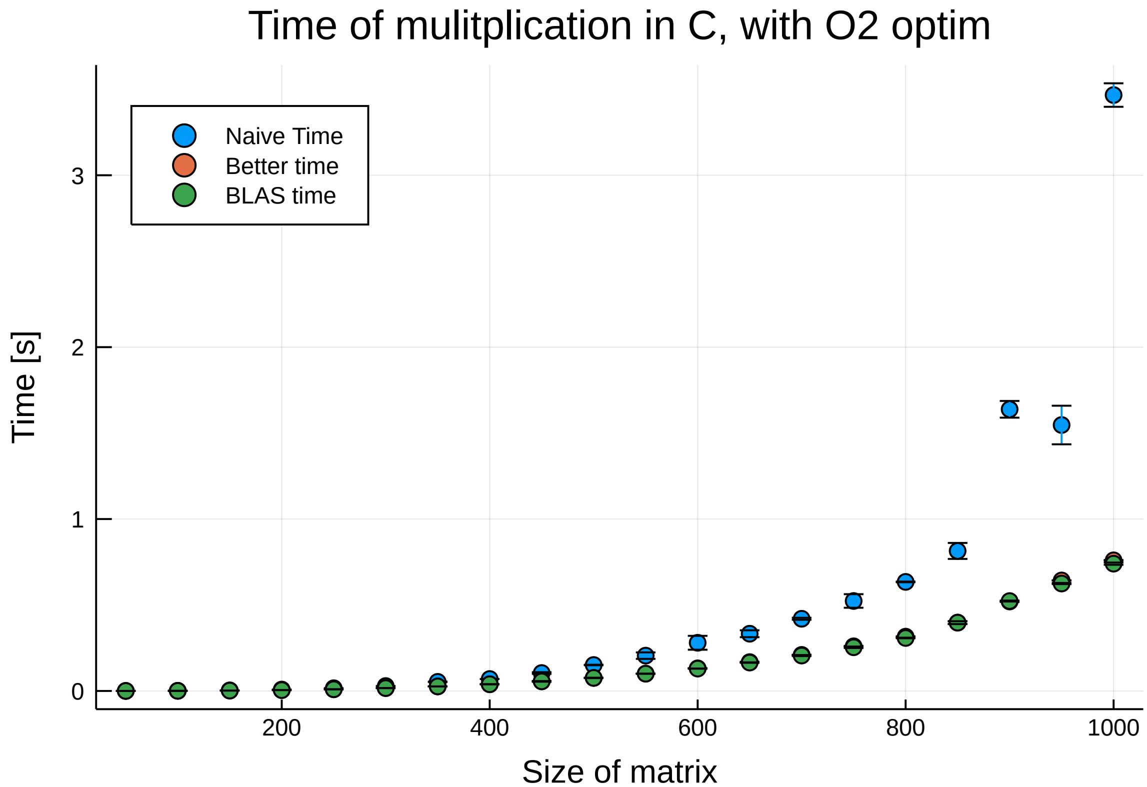
```
    ylabel = "Time [s]",)


display(plot00)
display(plot01)
display(plot02)
display(plot03)

display(naive_compare)
```

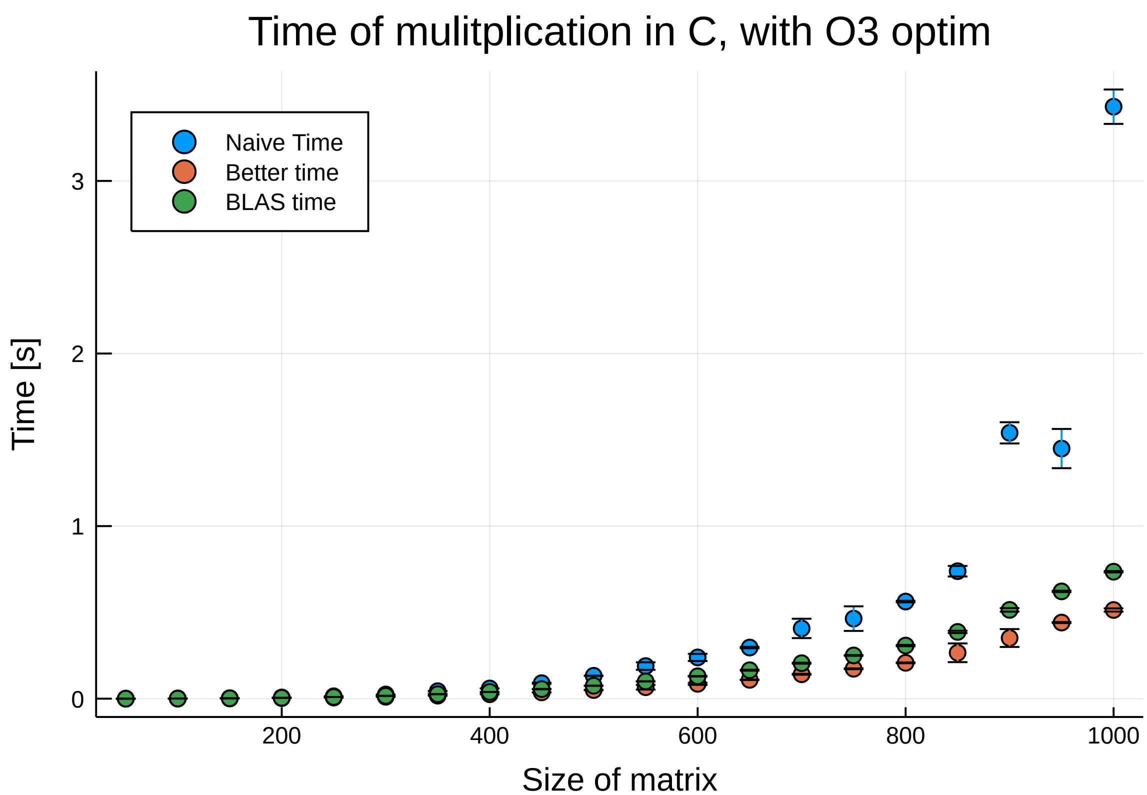


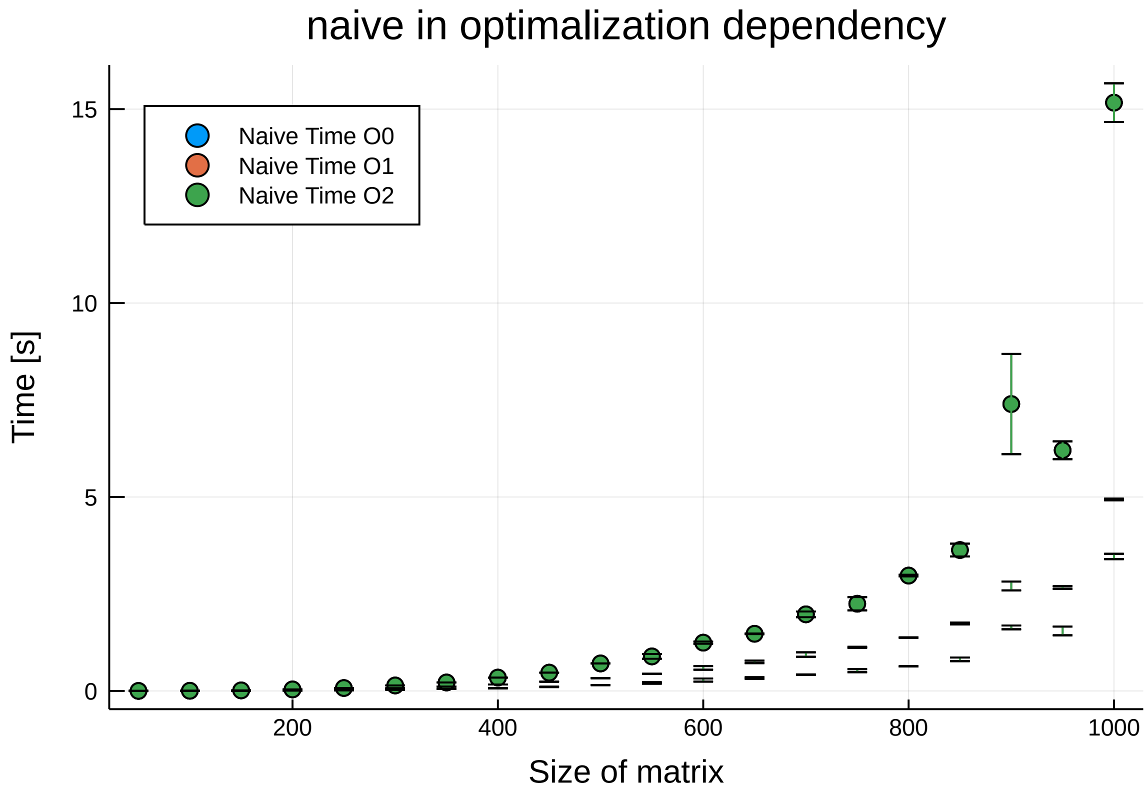
Time of mulitplication in C, with O0 optim
Naive Time
Better time
BLAS time
Time [s]
Size of matrix
0
5
10
15
200
400
600
800
1000



Time of mulitplication in C, with O1 optim
Naive Time
Better time
BLAS time
Time [s]
Size of matrix
0
1
2
3
4
5
200
400
600
800
1000

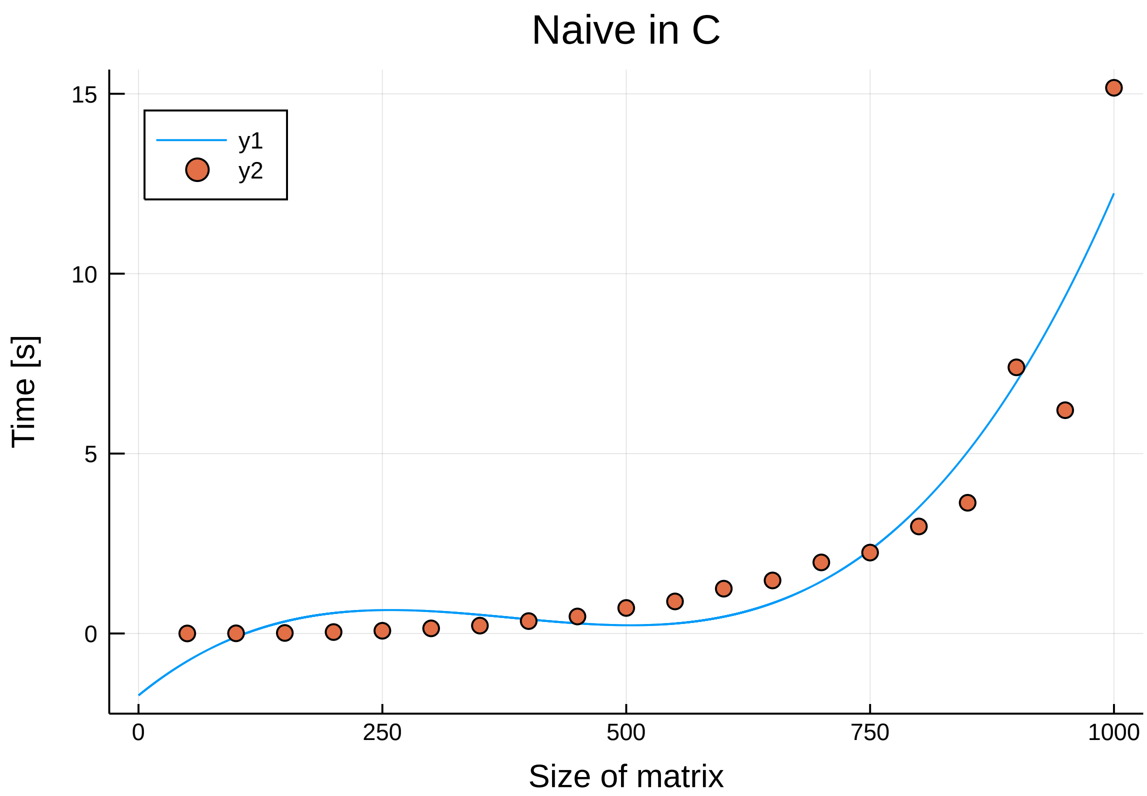

Time of mulitplication in C, with O2 optim
Naive Time
Better time
BLAS time
Time [s]
Size of matrix
0
1
2
3
200
400
600
800
1000


Time of mulitplication in C, with O3 optim
Naive Time
Better time
BLAS time
Time [s]
Size of matrix
0
1
2
3
200
400
600
800
1000


naive in optimalization dependency
Naive Time O0
Naive Time O1
Naive Time O2
Time [s]
0
5
10
15
200
400
600
800
1000
Size of matrix

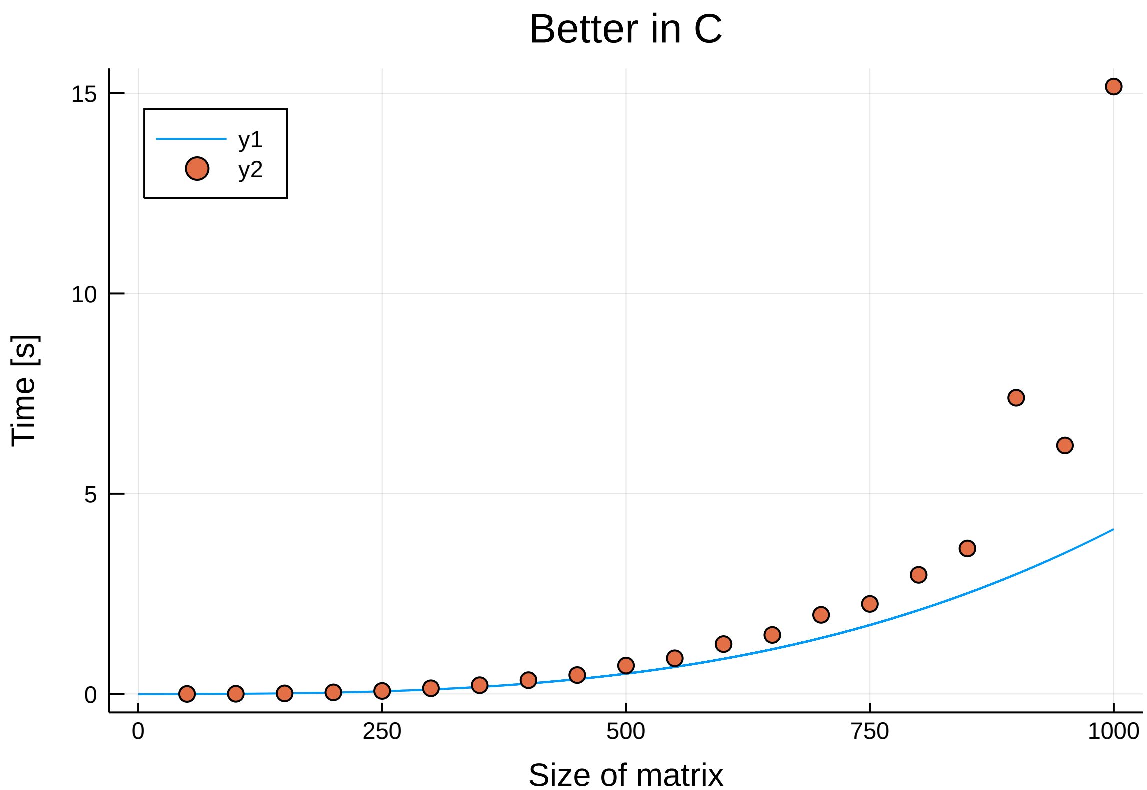
In [75]:
```
using Polynomials, Plots

fit_data0_naive=polyfit(data0[:columns_and_rows],data0[:naive_time_me
an],3)
xd=0:0.01:1000
plot(xd,polyval(fit_data0_naive, xd))
scatter!(data0[:columns_and_rows],data0[:naive_time_mean])
scatter!(labels = ["Naive Time 00" "Polynomial approximation"],
legend=:topleft,xlabel = "Size of matrix", ylabel = "Time [s]", title
= "Naive in C")
#print(fit_data0_naive)
```

Out[75]:



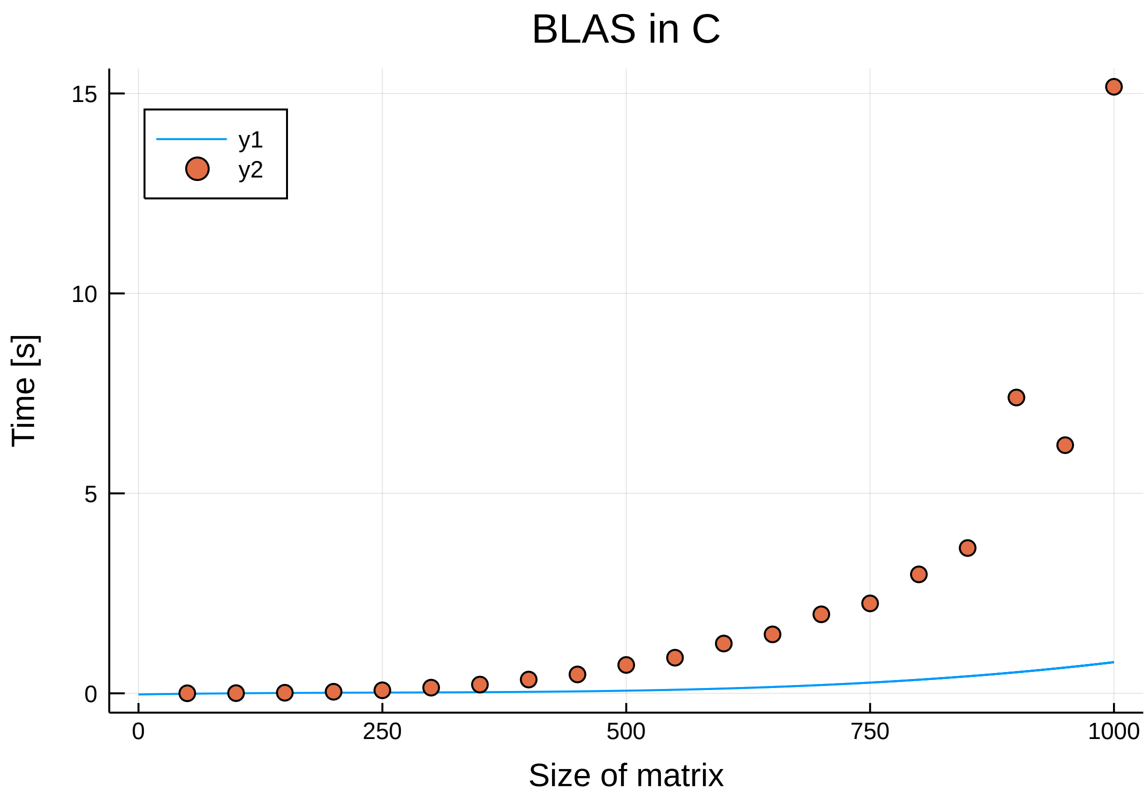
In [81]:
```
fit_data0_better=polyfit(data0[:columns_and_rows],data0[:better_time_
mean],3)
xd=0:0.01:1000
plot(xd,polyval(fit_data0_better, xd))
scatter!(data0[:columns_and_rows],data0[:naive_time_mean])
scatter!(labels = ["Naive Time O0" "Polynomial approximation"],
legend=:topleft,xlabel = "Size of matrix", ylabel = "Time [s]", title
= "Better in C")
#print(fit_data0_better)
```

Out[81]:



In [76]:
```
fit_data0_blas=polyfit(data0[:columns_and_rows],data0[:blas_time_mean
],3)
xd=0:0.01:1000
plot(xd,polyval(fit_data0_blas, xd))
scatter!(data0[:columns_and_rows],data0[:naive_time_mean])
scatter!(labels = ["Naive Time O0" "Polynomial approximation"],
legend=:topleft,xlabel = "Size of matrix", ylabel = "Time [s]", title
= "BLAS in C")
#print(fit_data0_blas)
```

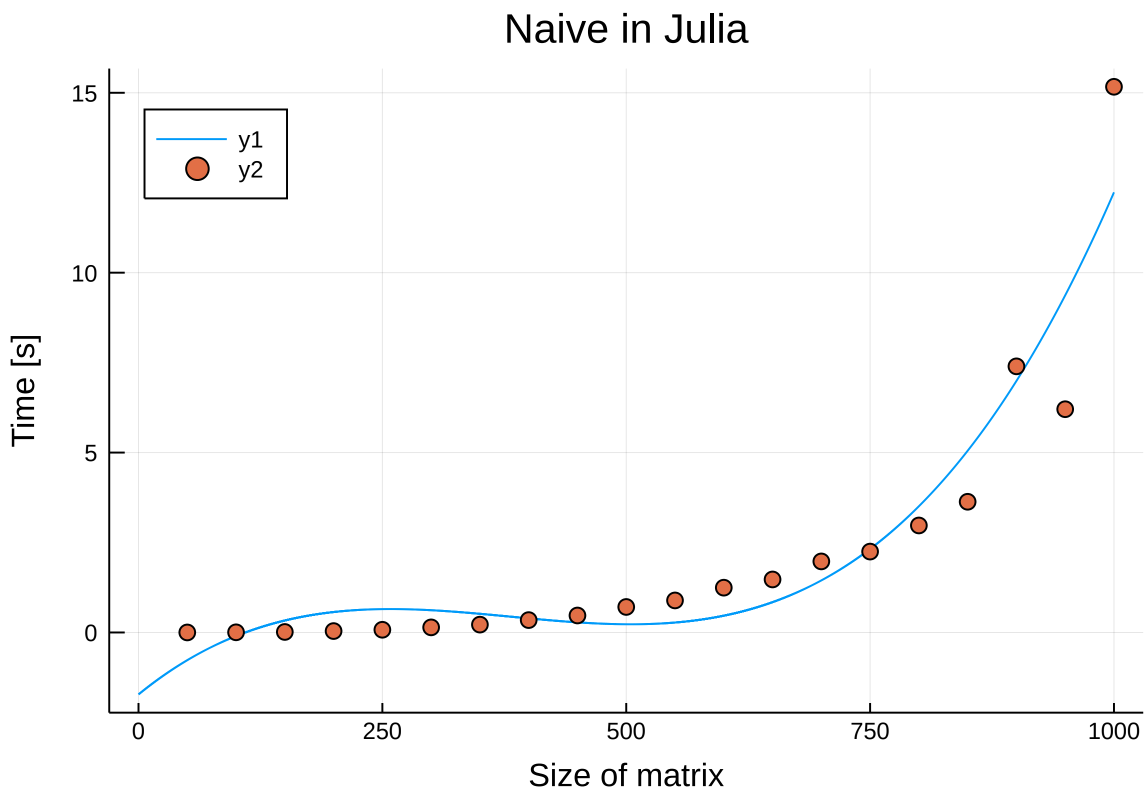
Out[76]:



In [77]:

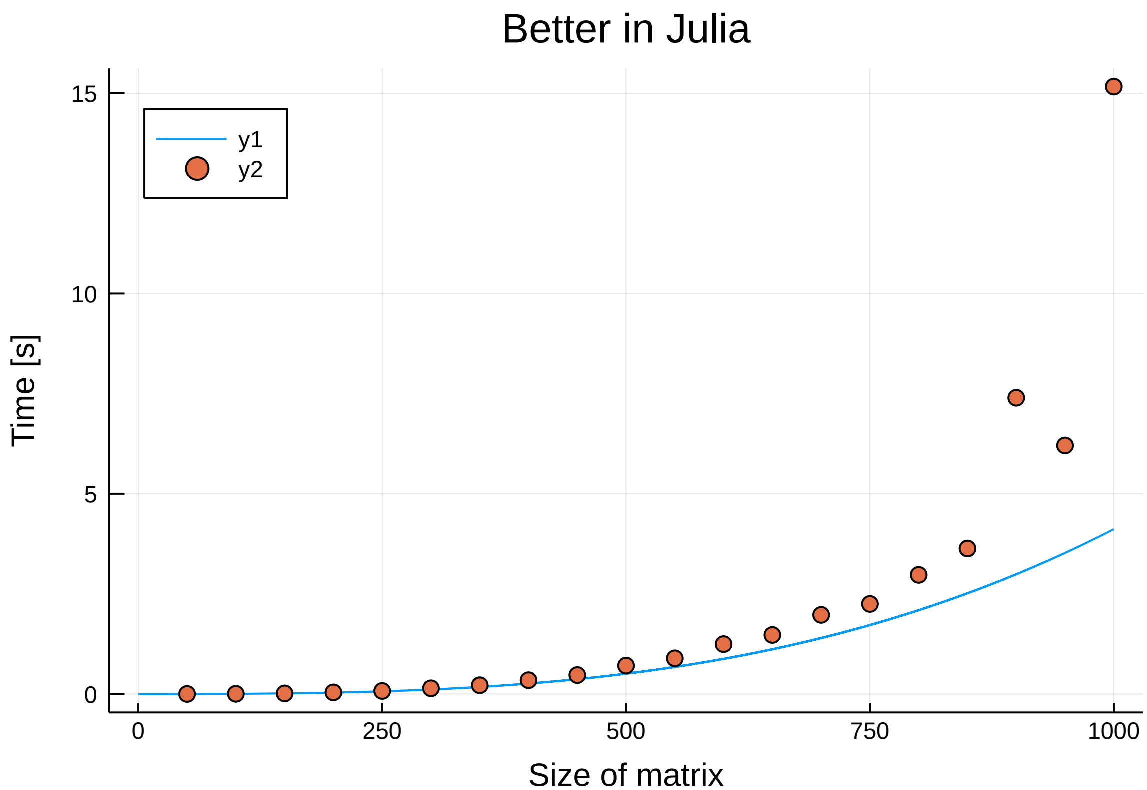
```
fit_julia_naive=polyfit(df2[:columns_and_rows],df2[:naive_time_mean],
3)
xd=0:0.01:1000
plot(xd,polyval(fit_data0_naive, xd))
scatter!(data0[:columns_and_rows],data0[:naive_time_mean])
scatter!(labels = ["Naive Time 00" "Polynomial approximation"],
legend=:topleft,xlabel = "Size of matrix", ylabel = "Time [s]", title
= "Naive in Julia")
#print(fit_julia_naive)
```

Out[77]:



In [78]:
```
fit_julia_better=polyfit(df2[:columns_and_rows],df2[:better_time_mean
],3)
xd=0:0.01:1000
plot(xd,polyval(fit_data0_better, xd))
scatter!(data0[:columns_and_rows],data0[:naive_time_mean])
scatter!(labels = ["Naive Time 00" "Polynomial approximation"],
legend=:topleft,xlabel = "Size of matrix", ylabel = "Time [s]", title
= "Better in Julia")
#print(fit_julia_better)
```

Out[78]:



In [79]:

```
fit_julia_blas=polyfit(df2[:columns_and_rows],df2[:blas_time_mean],3)
xd=0:0.01:1000
plot(xd,polyval(fit_data0_blas, xd))
scatter!(data0[:columns_and_rows],data0[:naive_time_mean])
scatter!(labels = ["Naive Time O0" "Polynomial approximation"],
legend=:topleft,xlabel = "Size of matrix", ylabel = "Time [s]", title
= "BLAS in Julia")
#print(fit_julia_blas)
```

Out[79]: