

# ColabでJuliaを使うためのノートブック

このノートブックの内容については再配布・改変・部分的コピーその他すべてを自由に行って構いません。

このノートブックは[Google Colabで実行できる](https://colab.research.google.com/github/genkuroki/Statistics/blob/master/2022/07-4%20Julia%20notebook%20for%20Google%20Colab.ipynb) (<https://colab.research.google.com/github/genkuroki/Statistics/blob/master/2022/07-4%20Julia%20notebook%20for%20Google%20Colab.ipynb>).

**2025-05-13:** 以下のセルを `@_using` の行のコメントアウトを全部外してからGoogle Colabで実行すると5分から6分程度かかるようである。その待ち時間に耐え切れないと感じる人は自分のパソコン上にJuliaをJupyter上で実行する環境を作ればよい。コンピュータの取り扱いの初心者のうちはその作業は非常に難しいと感じるかもしれないが、適当に検索したり、AIに質問したりすればできるはずである。

```

In [1]: 1 # Google Colabと自分のパソコンの両方で使えるようにするための工夫
2
3 import Pkg
4
5 """すでにPkg.add済みのパッケージのリスト (高速化のために用意)"""
6 _packages_added = [info.name for (uuid, info) in Pkg.dependencies() if info.is_direct_dep]
7
8 """_packages_added内 ないパッケージをPkg.addする"""
9 add_pkg_if_not_added_yet(pkg) = if !(pkg in _packages_added)
10     println(stderr, "# $(pkg).jl is not added yet, so let's add it.")
11     Pkg.add(pkg)
12 end
13
14 """expr::Exprからusing内の`.`を含まないモジュール名を抽出"""
15 function find_using_pkgs(expr::Expr)
16     pkgs = String[]
17     function traverse(expr::Expr)
18         if expr.head == :using
19             for arg in expr.args
20                 if arg.head == :. && length(arg.args) == 1
21                     push!(pkgs, string(arg.args[1]))
22                 elseif arg.head == :(.) && length(arg.args[1].args) == 1
23                     push!(pkgs, string(arg.args[1].args[1]))
24                 end
25             end
26         else
27             for arg in expr.args arg isa Expr && traverse(arg) end
28         end
29     end
30     traverse(expr)
31     pkgs
32 end
33
34 """必要そうなPkg.addを追加するマクロ"""
35 macro autoadd(expr)
36     pkgs = find_using_pkgs(expr)
37     :(add_pkg_if_not_added_yet.($(pkgs))); $expr)
38 end
39
40 # 以下は黒木玄がよく使っているパッケージ達
41 # 例えばQuadGKパッケージ(数値積分のパッケージ)の使い方は
42 # QuadGK.jl をインターネットで検索すれば得られる.
43
44 ENV["LINES"], ENV["COLUMNS"] = 100, 100
45 using LinearAlgebra
46 using Printf
47 using Random
48 Random.seed!(4649373)
49
50 @autoadd begin
51     using Distributions
52     using StatsPlots
53     default(fmt=:png, legendfontsize=12)
54     #using BenchmarkTools
55     #using Optim
56     #using QuadGK
57     #using RDatasets
58     #using Roots
59     #using StatsBase
60     #using StatsFuns
61     #using SpecialFunctions
62     #using SymPy
63 end

```

Julia言語 (<https://julialang.org/>)については以下の検索で色々学べる.

- Julia言語のドキュメント: <https://docs.julialang.org/en/v1/> (<https://docs.julialang.org/en/v1/>).
- Julia言語について検索: <https://www.google.com/search?q=Julia%E8%A8%80%E8%AA%9E> (<https://www.google.com/search?q=Julia%E8%A8%80%E8%AA%9E>).
- Distributions.jlパッケージについて検索: <https://www.google.com/search?q=Distributions.jl> (<https://www.google.com/search?q=Distributions.jl>).
- Plots.jlパッケージについて検索: <https://www.google.com/search?q=Plots.jl> (<https://www.google.com/search?q=Plots.jl>).
- StatsPlots.jlパッケージについて検索: <https://www.google.com/search?q=StatsPlots.jl> (<https://www.google.com/search?q=StatsPlots.jl>).
- BenchmarkTools.jlパッケージについて検索: <https://www.google.com/search?q=BenchmarkTools.jl> (<https://www.google.com/search?q=BenchmarkTools.jl>).
- Optim.jlパッケージについて検索: <https://www.google.com/search?q=Optim.jl> (<https://www.google.com/search?q=Optim.jl>).

- QuadGK.jlパッケージについて検索: <https://www.google.com/search?q=QuadGK.jl> (<https://www.google.com/search?q=QuadGK.jl>)
- RDatasets.jlパッケージについて検索: <https://www.google.com/search?q=RDatasets.jl> (<https://www.google.com/search?q=RDatasets.jl>)
- Roots.jlパッケージについて検索: <https://www.google.com/search?q=Roots.jl> (<https://www.google.com/search?q=Roots.jl>)
- StatsBase.jlパッケージについて検索: <https://www.google.com/search?q=StatsBase.jl> (<https://www.google.com/search?q=StatsBase.jl>)
- StatsFuns.jlパッケージについて検索: <https://www.google.com/search?q=StatsFuns.jl> (<https://www.google.com/search?q=StatsFuns.jl>)
- SpecialFunctions.jlパッケージについて検索: <https://www.google.com/search?q=SpecialFunctions.jl> (<https://www.google.com/search?q=SpecialFunctions.jl>)
- SymPy.jlパッケージについて検索: <https://www.google.com/search?q=SymPy.jl> (<https://www.google.com/search?q=SymPy.jl>)

## @autoadd マクロの使い方

例えば、パッケージA.jlやB.jlをインストール前(Pkg.add前)であるとき、

```
using A
using B: b1, b2
```

を実行しようとするエラーになってしまう。しかし、

```
@autoadd using A
@autoadd using B: b1, b2
```

または

```
@autoadd begin
using A
using B: b1, b2
end
```

を実行すれば、自動的にパッケージA.jlやB.jlがインストールされてから、using達が実行される。

以下のように @macroexpand を使えば具体的に何が実行されるかを確認できる。

```
In [2]: 1 (@macroexpand @autoadd using A) ▶ Base.remove_linenum!
```

```
Out[2]: quote
    Main.add_pkg_if_not_added_yet.(["A"])
    using A
end
```

```
In [3]: 1 (@macroexpand @autoadd using A, B, C) ▶ Base.remove_linenum!
```

```
Out[3]: quote
    Main.add_pkg_if_not_added_yet.(["A", "B", "C"])
    using A, B, C
end
```

```
In [4]: 1 (@macroexpand @autoadd using A: a1, a2, @a3) ▶ Base.remove_linenum!
```

```
Out[4]: quote
    Main.add_pkg_if_not_added_yet.(["A"])
    using A: a1, a2, @a3
end
```

```
In [5]: ► 1 (@macroexpand @autoadd begin
2 using A: a1
3 using A.B
4 using A.C: c1, c2
5 #using D
6 using E, A.F, G
7 using H: h1, h2
8 using I
9 end) ► Base.remove_linenums!
```

```
Out[5]: quote
Main.add_pkg_if_not_added_yet(["A", "E", "G", "H", "I"])
begin
  using A: a1
  using A.B
  using A.C: c1, c2
  using E, A.F, G
  using H: h1, h2
  using I
end
end
```

## ランダムウォーク

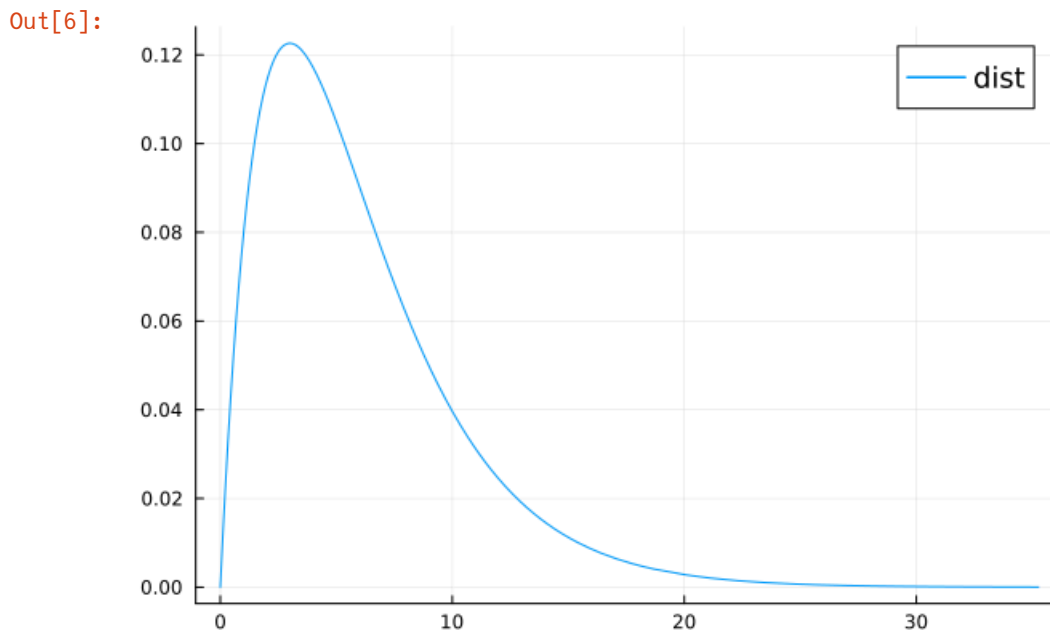
期待値が  $\mu$  で標準偏差が  $\sigma$  の確率分布の独立同分布確率変数列  $X_1, X_2, X_3, \dots$  について,

$$W_n = (X_1 - \mu) + (X_2 - \mu) + \dots + (X_n - \mu), \quad n = 1, 2, 3, \dots$$

の様子がどうなるかを見てみよう.

```
In [6]: ► 1 dist = Gamma(2, 3)
2 @show mu, sigma = mean(dist), std(dist)
3 plot(dist; label="dist")
```

```
(mu, sigma) = (mean(dist), std(dist)) = (6.0, 4.242640687119285)
```



```
In [7]: ► 1 X_minus_mu = rand(dist - mu, 10) # X_1 - mu, X_2 - mu, ..., X_10 - mu を生成
```

```
Out[7]: 10-element Vector{Float64}:
 3.0311267478769928
-1.4409032773374868
-4.693642987661233
-1.7502325815689757
10.650497896417825
-2.563141099889714
-3.189505348153571
-1.8701849265462256
-3.0119934886403863
-4.283202704007239
```

```
In [8]: 1 cumsum(X_minus_mu) # W_1, W_2, ..., W_10 を作成
```

```
Out[8]: 10-element Vector{Float64}:  
 3.0311267478769928  
 1.590223470539506  
-3.1034195171217274  
-4.853652098690703  
 5.796845797727122  
 3.2337046978374078  
 0.044199349683836875  
-1.8259855768623883  
-4.837979065502775  
-9.121181769510013
```

In [9]: 1 ?cumsum

search: cumsum cumsum! sum

```
Out[9]: cumsum(A; dims::Integer)
```

Cumulative sum along the dimension `dims`. See also [cumsum! \(@ref\)](#) to use a preallocated output array, both for performance and to control the precision of the output (e.g. to avoid overflow).

## Examples

```
julia> a = [1 2 3; 4 5 6]
2×3 Matrix{Int64}:
 1  2  3
 4  5  6
```

```
julia> cumsum(a, dims=1)
2×3 Matrix{Int64}:
 1  2  3
 5  7  9
```

```
julia> cumsum(a, dims=2)
2×3 Matrix{Int64}:
 1  3  6
 4  9 15
```

!!! note The return array's `eltype` is `Int` for signed integers of less than system word size and `UInt` for unsigned integers of less than system word size. To preserve `eltype` of arrays with small signed or unsigned integer `accumulate(+, A)` should be used.

```
```\doctest
julia> cumsum(Int8[100, 28])
2-element Vector{Int64}:
 100
 128

julia> accumulate(+, Int8[100, 28])
2-element Vector{Int8}:
 100
-128
```\
```

In the former case, the integers are widened to system word size and therefore the result is `Int64[100, 128]`. In the latter case, no such widening happens and integer overflow results in `Int8[100, -128]`.

---

```
cumsum(itr)
```

Cumulative sum of an iterator.

See also [accumulate \(@ref\)](#) to apply functions other than `+`.

!!! compat "Julia 1.5" `cumsum` on a non-array iterator requires at least Julia 1.5.

## Examples

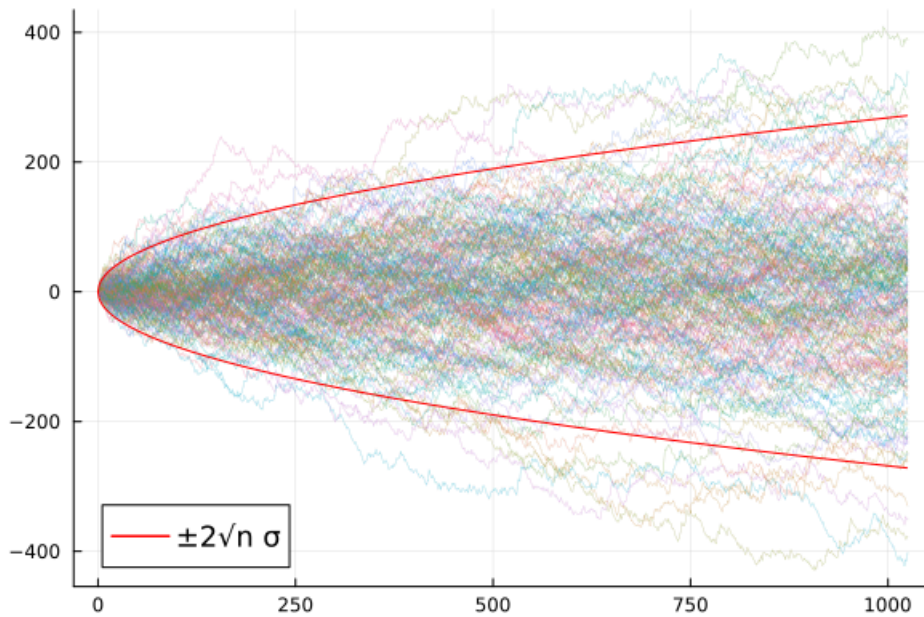
```
julia> cumsum(1:3)
3-element Vector{Int64}:
 1
 3
 6
```

```
julia> cumsum((true, false, true, false, true))
(1, 1, 2, 2, 3)
```

```
julia> cumsum(fill(1, 2) for i in 1:3)
3-element Vector{Vector{Int64}}:
 [1, 1]
 [2, 2]
 [3, 3]
```

```
In [10]: 1 nmax = 2^10 # maximum sample size
2 niters = 200 # number of iterations
3 Ws = [cumsum(rand(dist - mu, nmax)) for _ in 1:niters] # [W_1, W_2, ..., W_nmax] をniters個生成
4
5 plot()
6 for W in Ws
7     plot!([0; W]; label="", lw=0.3, alpha=0.5)
8 end
9 plot!(n -> +2sqrt(n)*sigma, 0, nmax; label="±2√n σ", c=:red)
10 plot!(n -> -2sqrt(n)*sigma, 0, nmax; label="", c=:red)
```

Out[10]:



期待値が 0 のギャンブルを  $n$  回繰り返すと、トータルでの勝ち負けの金額はおおよそ  $\pm 2\sqrt{n}\sigma$  の範囲におさまる(ランダムウォークの偏差).

## 大数の法則

期待値が 0 で標準偏差が  $\sigma$  の確率分布の独立同分布確率変数列  $X_1, X_2, X_3, \dots$  について、サイズ  $n$  の標本平均

$$\bar{X}_n = \frac{X_1 + X_2 + \dots + X_n}{n}$$

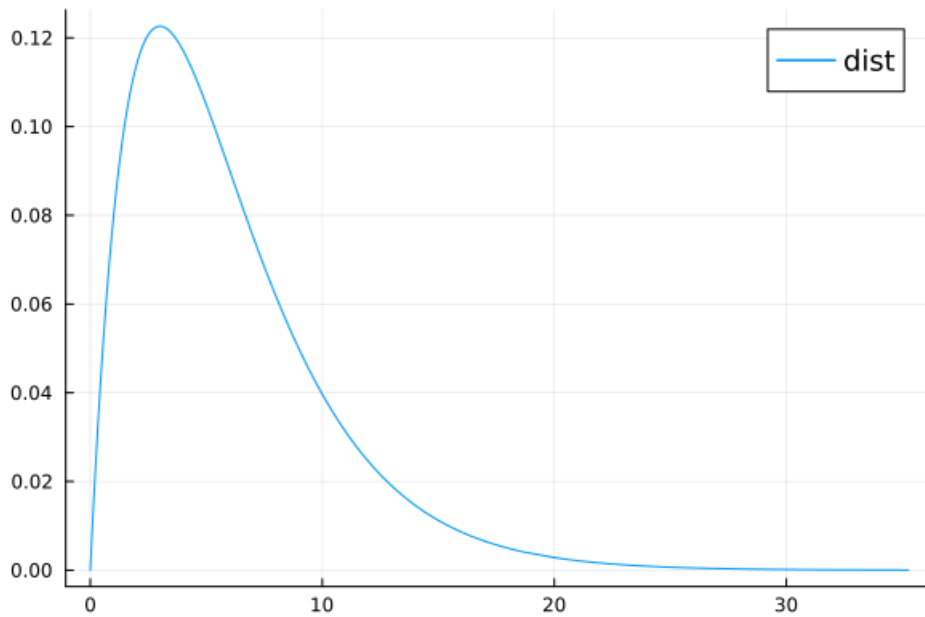
の様子がどうなるかを見てみよう.



```
In [11]: 1 dist = Gamma(2, 3)
2 @show mu, sigma = mean(dist), std(dist)
3 plot(dist; label="dist")
```

(mu, sigma) = (mean(dist), std(dist)) = (6.0, 4.242640687119285)

Out[11]:



```
In [12]: 1 X = rand(dist, 10) # X_1, X_2, ..., X_10 を生成
```

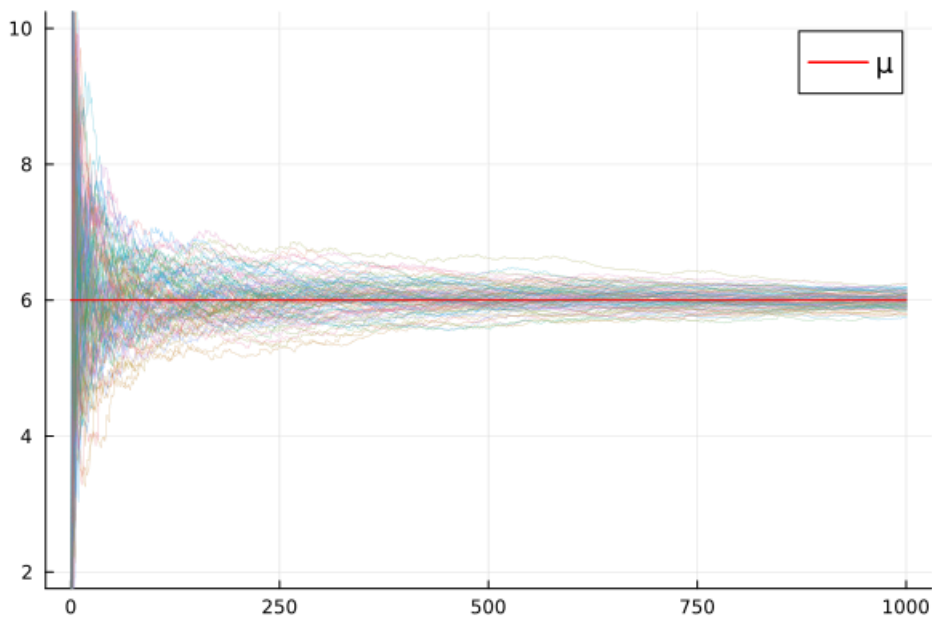
Out[12]: 10-element Vector{Float64}:  
0.40805136957750254  
11.35246443572607  
1.7750298759291916  
2.871320442475634  
5.377988900748953  
1.6866822858975898  
4.050010771853904  
4.8982949336627355  
1.492397480850824  
1.1322589811898953

```
In [13]: 1 Xbar = cumsum(X) ./ (1:10) # Xbar_1, Xbar_2, ..., Xbar_10 を作成
```

Out[13]: 10-element Vector{Float64}:  
0.40805136957750254  
5.8802579026517865  
4.511848560410922  
4.101716530927099  
4.356971004891471  
3.911922885059157  
3.9316497260298355  
4.052480376983947  
3.7680267218580448  
3.5044499477912296

```
In [14]: 1 nmax = 1000 # maximum sample size
2 niters = 100 # number of iterations
3 Xbars = [cumsum(rand(dist, nmax)) ./ (1:nmax) for _ in 1:niters] # [Xbar_1, ..., Xbar_nmax]
4
5 plot()
6 for Xbar in Xbars
7     plot!([0; Xbar]; label="", lw=0.3, alpha=0.5)
8 end
9 plot!(x → mu, 0, nmax; label="μ", c=:red)
10 plot!(ylim=(mu-sigma, mu+sigma))
```

Out[14]:



期待値が 0 のギャンブルを  $n$  回繰り返すと、1回ごとの勝ち負けの平均値は  $\mu$  に近付く(大数の法則).

ランダムウォーク(トータルでの勝ち負けの金額の話)と大数の法則(トータルの勝ち負けの金額を繰り返した回数の  $n$  で割って得られる1回ごとの平均値の話)を混同するとひどい目にあうだろう!

## 中心極限定理の素朴な確認の仕方

期待値が  $\mu$  で標準偏差が  $\sigma$  の確率分布の独立同分布確率変数列  $X_1, X_2, X_3, \dots$  について、標本平均  $\bar{X}_n = (X_1 + \dots + X_n)/n$  が従う分布は  $n$  が大きくなると、期待値  $\mu$  と標準偏差  $\sigma/\sqrt{n}$  を持つ正規分布で近似される. すなわち、

$$Y_n = \sqrt{n} (\bar{X} - \mu) = \frac{(X_1 - \mu) + \dots + (X_n - \mu)}{\sqrt{n}}$$

が従う分布は、 $n$  が大きいとき、期待値 0 と標準偏差  $\sigma$  を持つ正規分布で近似され、

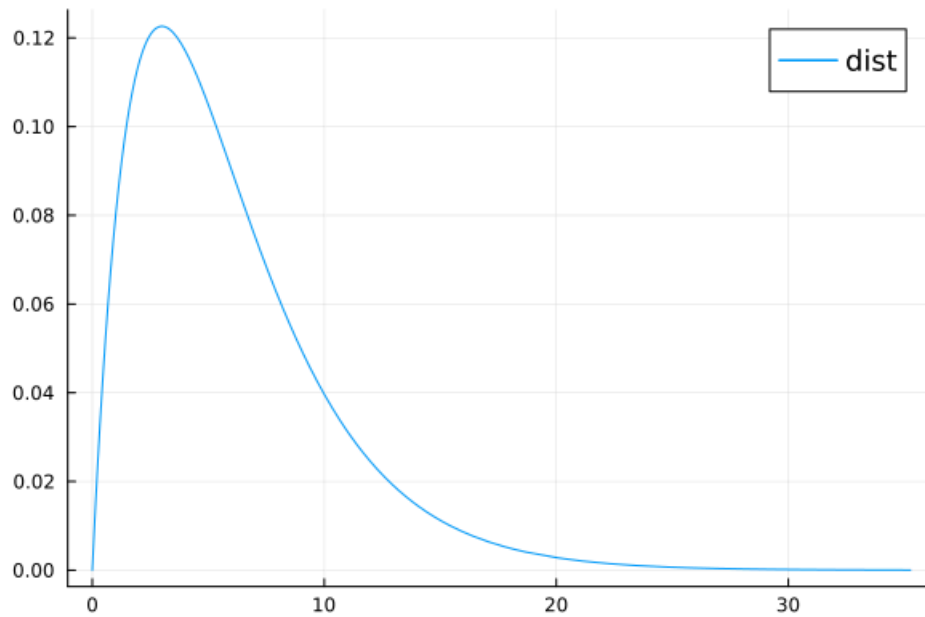
$$Z_n = \frac{\sqrt{n} (\bar{X} - \mu)}{\sigma} = \frac{(X_1 - \mu) + \dots + (X_n - \mu)}{\sqrt{n} \sigma}$$

が従う分布は、 $n$  が大きいとき、標準正規分布で近似される.

```
In [15]: 1 dist = Gamma(2, 3)
2 @show mu, sigma = mean(dist), std(dist)
3 plot(dist; label="dist")
```

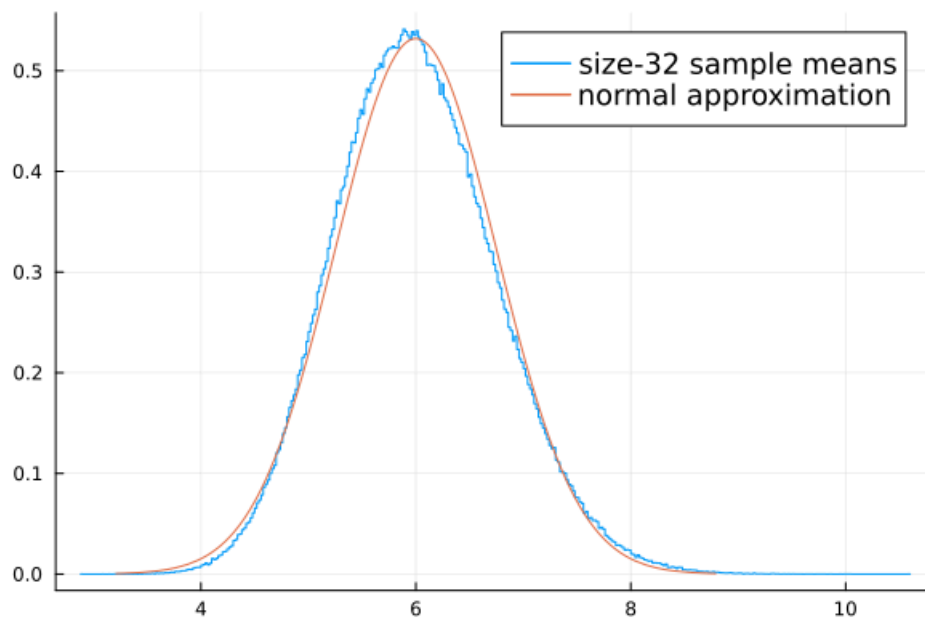
(mu, sigma) = (mean(dist), std(dist)) = (6.0, 4.242640687119285)

Out[15]:



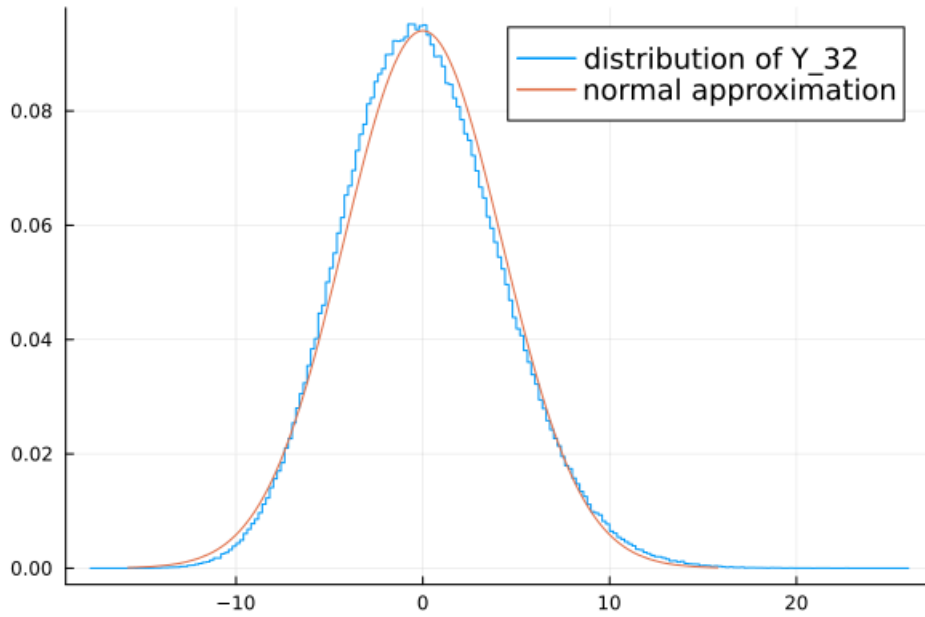
```
In [16]: 1 n = 2^5 # sample size
2 niters = 10^6 # number of iterations
3 Xbars = [mean(rand(dist, n)) for _ in 1:niters] # niters個の標本平均を計算
4
5 stephist(Xbars; norm=true, label="size-$n sample means")
6 plot!(Normal(mu, sigma/sqrt(n)); label="normal approximation")
```

Out[16]:



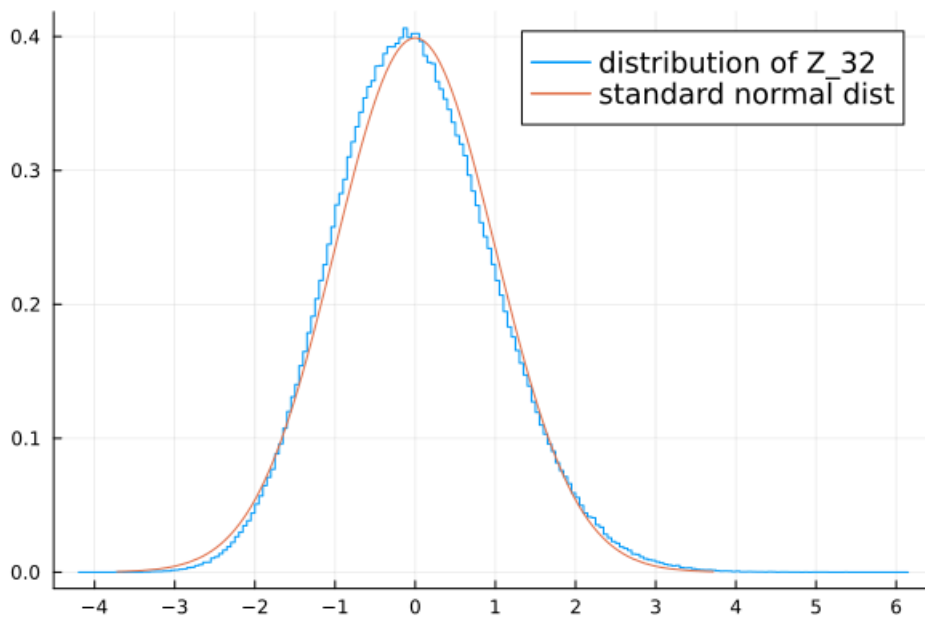
```
In [17]: 1 n = 2^5 # sample size
2 Yns = [sqrt(n) * (Xbar - mu) for Xbar in Xbars] # Z_nを繰り返し計算
3
4 stephist(Yns; norm=true, label="distribution of Y_$n")
5 plot!(Normal(0, sigma); label="normal approximation")
```

Out[17]:



```
In [18]: 1 n = 2^5 # sample size
2 Zns = [sqrt(n) * (Xbar - mu) / sigma for Xbar in Xbars] # Z_nを繰り返し計算
3
4 stephist(Zns; norm=true, label="distribution of Z_$n")
5 plot!(Normal(); label="standard normal dist")
6 plot!(xtick=-10:10)
```

Out[18]:



以下は自由に使って下さい

In [ ]: 1

In [ ]: 1

In [ ]: 1

In [ ]: 1

