

# ColabでJulia言語を使った統計学の勉強の仕方

- 黒木玄
- 2025-05-13

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/2022/07-3%20How%20to%20use%20Julia%20language%20in%20Google%20Colab%20for%20learning%20statistics.ipynb) (<https://colab.research.google.com/github/genkuroki/Statistics/blob/master/2022/07-3%20How%20to%20use%20Julia%20language%20in%20Google%20Colab%20for%20learning%20statistics.ipynb>).

## 目次

- ▼ [1. Google ColabでのJulia言語の使い方](#)
  - [1.1 ColabでのJuliaの実行](#)
  - [1.2 グラフの描き方](#)
  - [1.3 標準正規分布乱数のプロット](#)
  - [1.4 確率分布の扱い方](#)
  - [1.5 正規分布の確率密度関数のプロット](#)
- ▼ [2. Anscombeの例のプロット](#)
  - [2.1 RDatasets.jlパッケージのインストール](#)
  - [2.2 データのプロットの仕方](#)
- ▼ [3. Datasaurusの散布図のプロット](#)
  - [3.1 データの取得](#)
  - [3.2 散布図の作成](#)
- ▼ [4. 中心極限定理のプロット](#)
  - [4.1 素朴なワークフロー](#)
  - [4.2 Revise.jlを使うワークフロー](#)
  - [4.3 問題: 自分で関数を定義して実行してみよ.](#)

```
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  にないパッケージを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 @autoadd begin
41     using Distributions
42     using RDatasets
43     using StatsPlots
44     default(fmt = :png)
45 end
```

## 1 Google ColabでのJulia言語の使い方

### 1.1 ColabでのJuliaの実行

(1) ブラウザでGoogleアカウントのどれかにログインしておきます.

(2) [Google Colab \(https://colab.research.google.com/\)](https://colab.research.google.com/)にアクセスする.

(3) 「ノートブックを開く」の「GitHub」を選択する.

(4) GitHubにおいてある ipynb ファイルのURLを入力してEnterキーを押す. 例えば

- <https://github.com/genkuroki/Statistics/blob/master/2022/07-3%20How%20to%20use%20Julia%20language%20in%20Google%20Colab%20for%20learning%20statistics.ipynb>

というURLを入力する.

(5) 実際にその例のURLを入力してEnterキーを押すと, このファイルがGoogle Colabで開かれる.

(6) そのノートブックの全体をColabで実行し直したければ, 「ランタイム」→「すべてのセルを実行」を選択する.

(7) 適当にGoogle Colabの使い方を検索して調べればより詳しい使い方が分かる.

- 各セルの先頭に ? と入力した後に関数名などを入れるとヘルプを読むことができる.
- 各セルの先頭に ] と入力した後にパッケージ管理モードのコマンドを入力して実行できる.
- タブキーによる補完を使う.
- 各セルの最後に ; を付けて実行すると計算結果が表示されない.

**問題:** 以上を実際に行ってみよ.

```
In [2]: 1 1 + 1
```

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

```
In [3]: 1 sin(pi/6)
```

```
Out[3]: 0.49999999999999994
```

```
In [4]: 1 sinpi(1/6)
```

```
Out[4]: 0.5
```

## 1.2 グラフの描き方

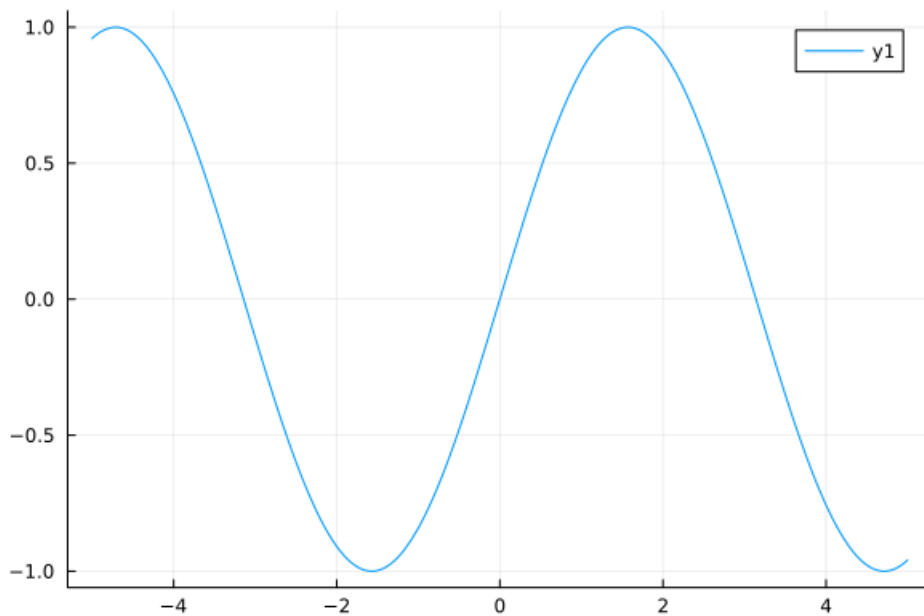
(7) Colabで統計学対応のグラフ作画パッケージを使うためには次を実行する:

```
import Pkg
Pkg.add("StatsPlots")
using StatsPlots
```

このノートブックでは最初のセルでこれと同等のことを実行できるようにしてあるので、最初のセルを実行しておけばよい。

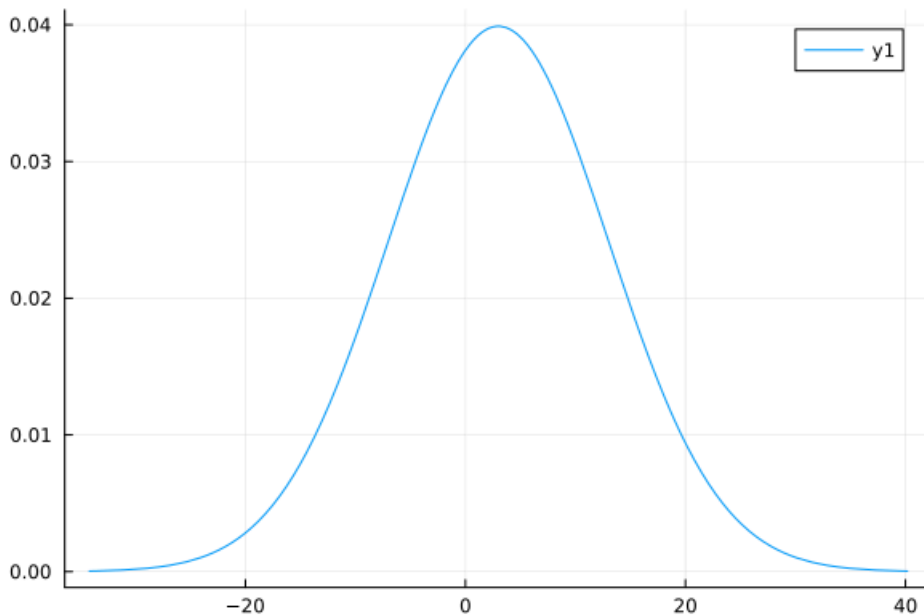
```
In [5]: 1 plot(sin)
```

```
Out[5]:
```



```
In [6]: 1 plot(Normal(3, 10))
```

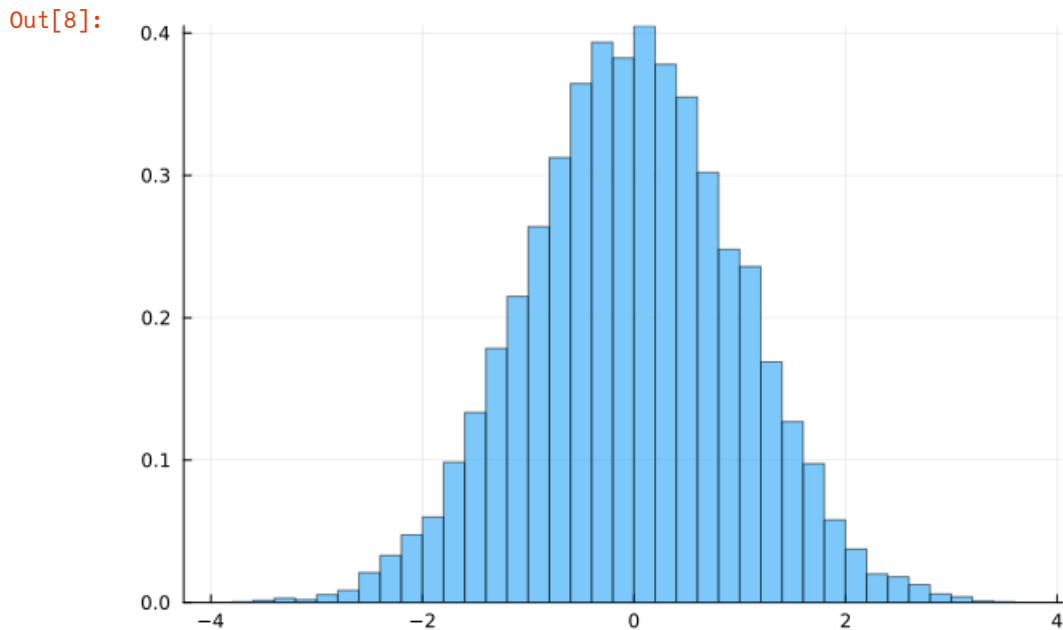
```
Out[6]:
```



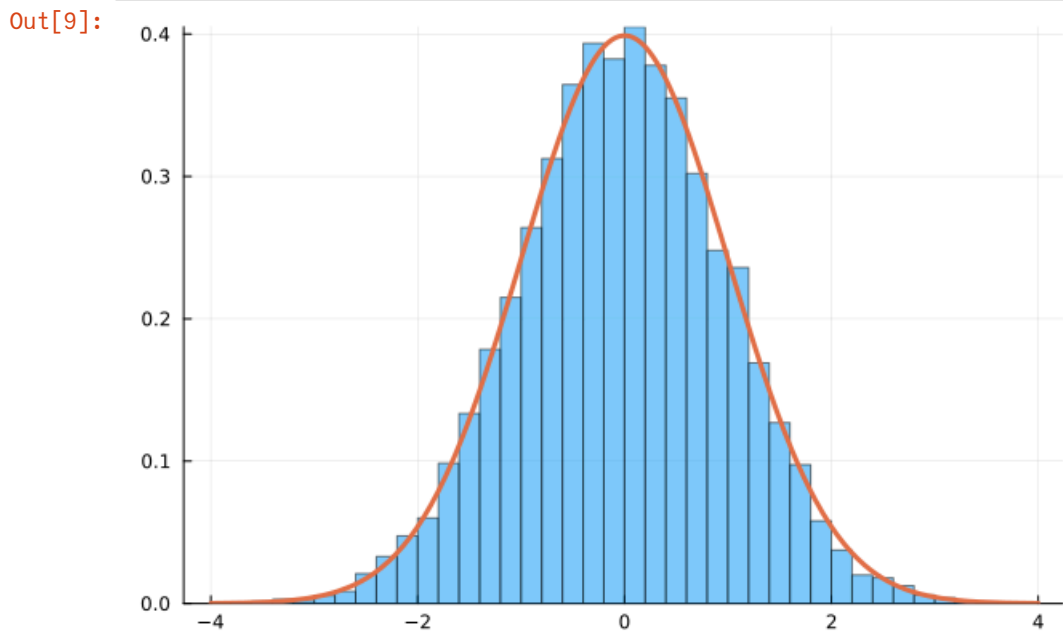
### 1.3 標準正規分布乱数のプロット

```
In [7]: 1 # 標準正規分布の乱数を10^4個生成  
2 Z = randn(10^4);
```

```
In [8]: 1 histogram(Z; norm=true, alpha=0.5, label="")
```



```
In [9]: 1 plot!(x → exp(-x^2/2)/sqrt(2pi), -4, 4; label="", lw=3)
```



### 1.4 確率分布の扱い方

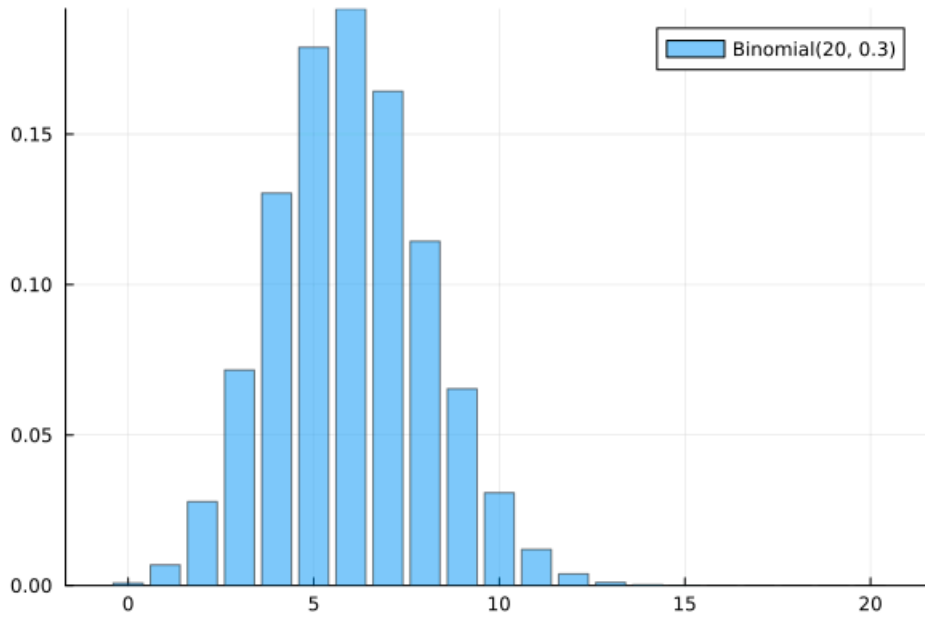
(7) 確率分布を扱うためのパッケージを使うためには次を実行する:

```
import Pkg  
Pkg.add("StatsPlots")  
using StatsPlots
```

このノートブックでは最初のセルでこれと同等のことを実行できるようにしてあるので、最初のセルを実行しておけばよい。

```
In [10]: 1 dist = Binomial(20, 0.3)
2 bar(dist; alpha=0.5, label="Binomial(20, 0.3)")
```

Out[10]:

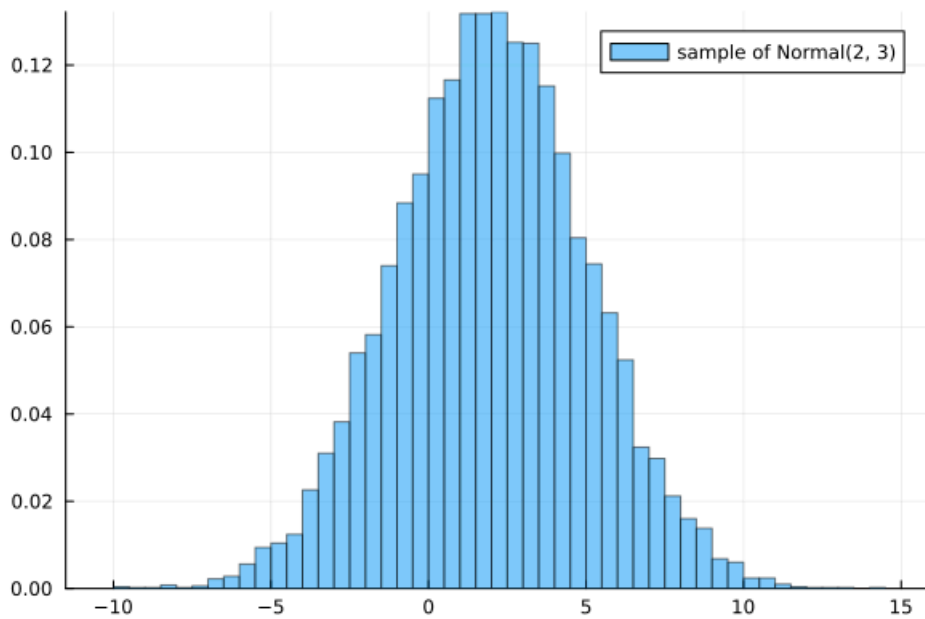


## 1.5 正規分布の確率密度関数のプロット

```
In [11]: 1 X = rand(Normal(2, 3), 10^4);
```

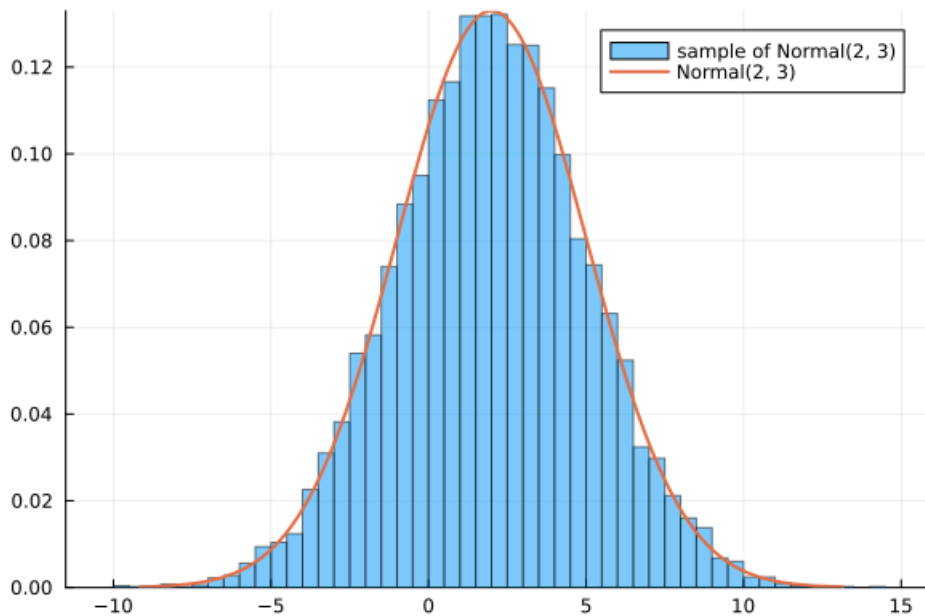
```
In [12]: 1 histogram(X; norm=true, alpha=0.5, label="sample of Normal(2, 3)")
```

Out[12]:



```
In [13]: 1 plot!(Normal(2, 3); label="Normal(2, 3)", lw=2)
```

Out[13]:



## 2 Anscombeの例のプロット

### 2.1 RDatasets.jlパッケージのインストール

確率分布を扱うためのパッケージを入れるためには次を実行する:

```
import Pkg
Pkg.add("RDatasets")
using RDatasets
```

このノートブックでは最初のセルでこれと同等のことを実行できるようにしてあるので、最初のセルを実行しておけばよい。

```
In [14]: 1 anscombe = dataset("datasets", "anscombe")
```

Out[14]: 11×8 DataFrame

Row	X1	X2	X3	X4	Y1	Y2	Y3	Y4
	Int64	Int64	Int64	Int64	Float64	Float64	Float64	Float64
1	10	10	10	8	8.04	9.14	7.46	6.58
2	8	8	8	8	6.95	8.14	6.77	5.76
3	13	13	13	8	7.58	8.74	12.74	7.71
4	9	9	9	8	8.81	8.77	7.11	8.84
5	11	11	11	8	8.33	9.26	7.81	8.47
6	14	14	14	8	9.96	8.1	8.84	7.04
7	6	6	6	8	7.24	6.13	6.08	5.25
8	4	4	4	19	4.26	3.1	5.39	12.5
9	12	12	12	8	10.84	9.13	8.15	5.56
10	7	7	7	8	4.82	7.26	6.42	7.91
11	5	5	5	8	5.68	4.74	5.73	6.89

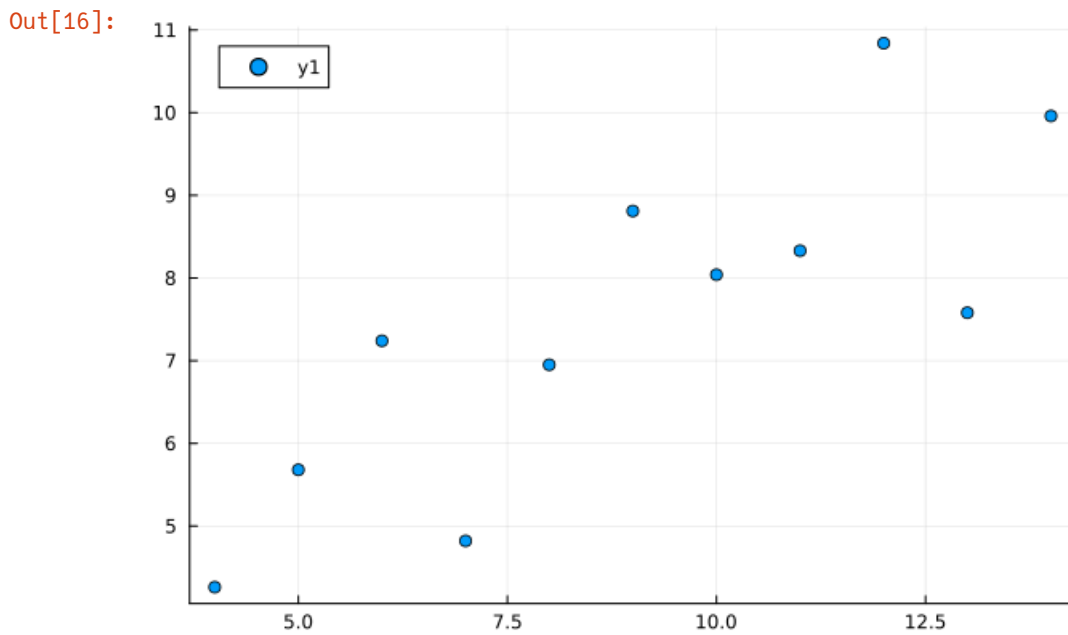
### 2.2 データのプロットの仕方

以下ではデータ1の場合のプロットの仕方を説明しよう。

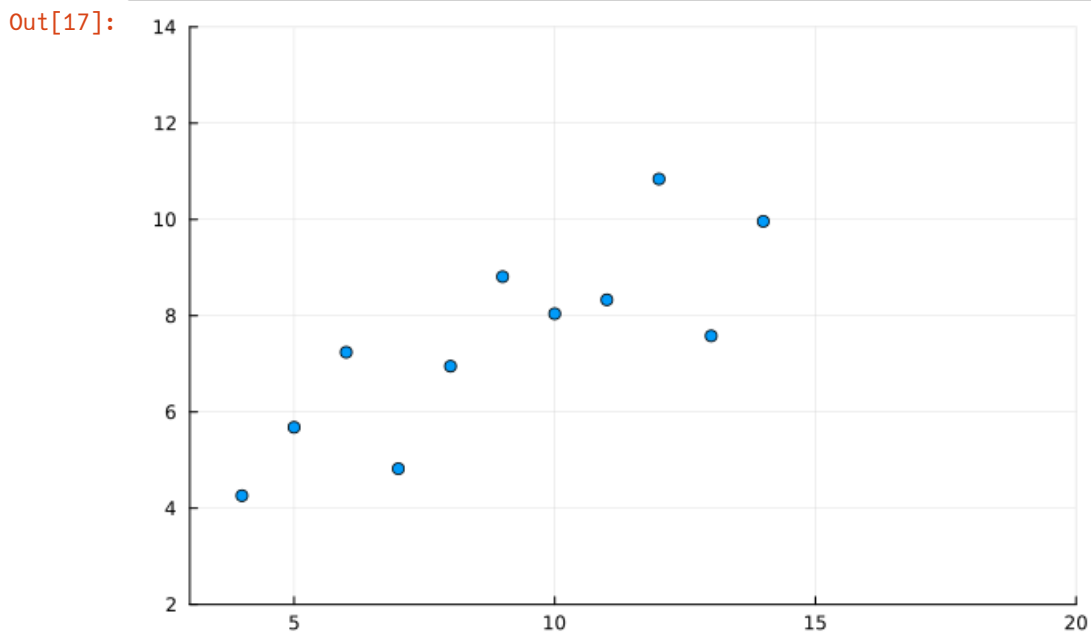
```
In [15]: 1 # x, y にデータを入れる
2 x, y = anscombe.X1, anscombe.Y1
```

Out[15]: ([10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5], [8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68])

```
In [16]: 1 # 散布図を描いてみる
2 using StatsPlots
3 scatter(x, y)
```



```
In [17]: 1 # xlim, ylimなどを追加
2 scatter(x, y; label="", xlim=(3, 20), ylim=(2, 14))
```



```
In [18]: 1 # データの標本平均や不偏分散・不偏共分散を計算
2 xbar = mean(x)
```

Out[18]: 9.0

```
In [19]: 1 ybar = mean(y)
```

Out[19]: 7.500909090909093

```
In [20]: 1 sx2 = var(x)
```

Out[20]: 11.0

```
In [21]: 1 sy2 = var(y)
```

Out[21]: 4.127269090909091

```
In [22]: 1 sxy = cov(x, y)
```

Out[22]: 5.501

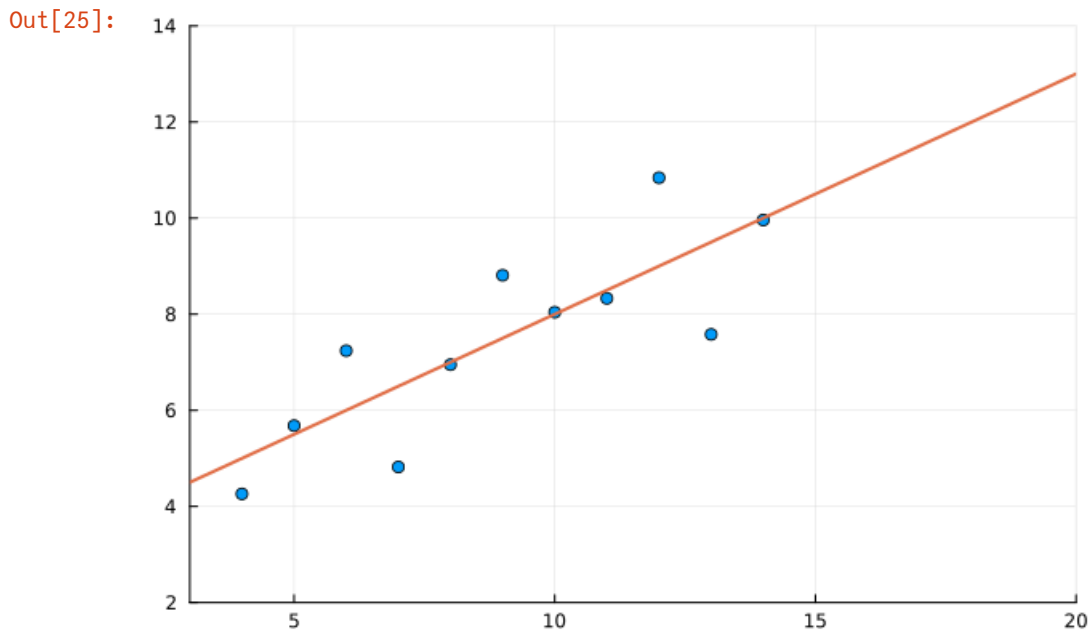
```
In [23]: 1 betahat = sxy/sx2
```

```
Out[23]: 0.5000909090909091
```

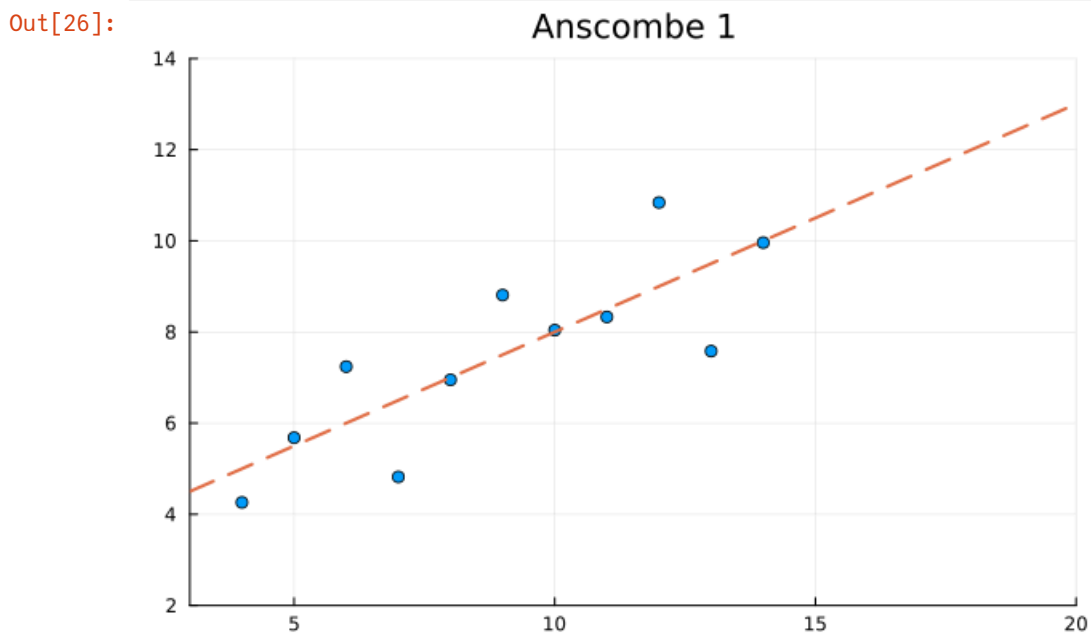
```
In [24]: 1 alphahat = ybar - betahat*xbar
```

```
Out[24]: 3.0000909090909103
```

```
In [25]: 1 scatter(x, y; label="", xlim=(3, 20), ylim=(2, 14))  
2 plot!(x → alphahat + betahat*x, 3, 20; label="", lw=2)
```



```
In [26]: 1 scatter(x, y; label="", xlim=(3, 20), ylim=(2, 14), title="Anscombe 1")  
2 plot!(x → alphahat + betahat*x, 3, 20; label="", lw=2, ls=:dash)
```



```
In [27]: 1 # design matrix  
2 X = x .^ (0:1)'
```

```
Out[27]: 11x2 Matrix{Int64}:
```

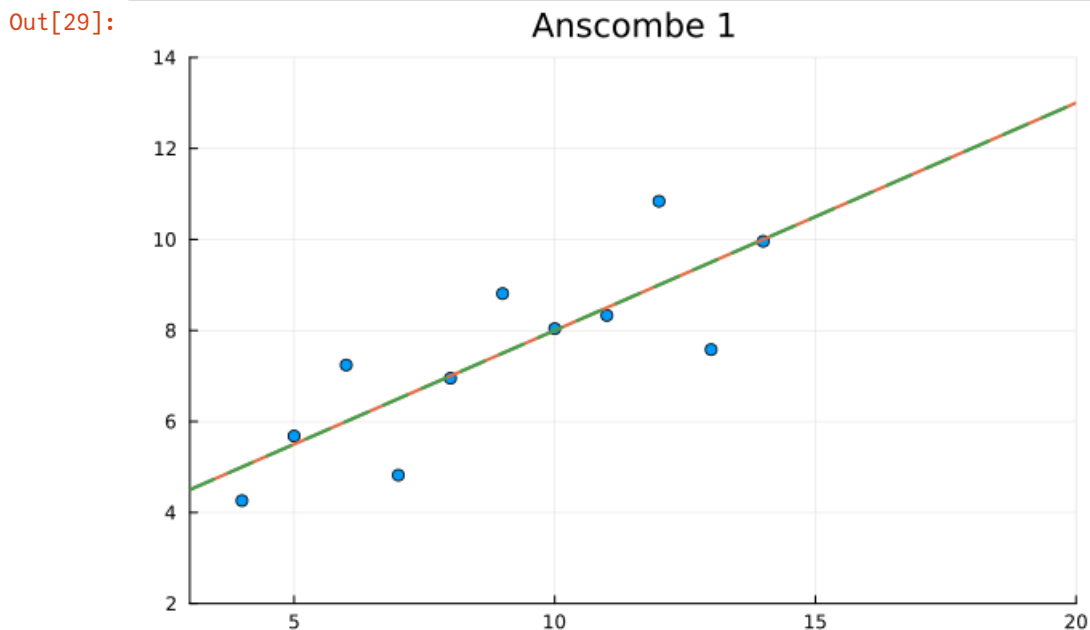
```
1 10  
1 8  
1 13  
1 9  
1 11  
1 14  
1 6  
1 4  
1 12  
1 7  
1 5
```



```
In [28]: 1 # 最小二乗法を一発実現 (計画行列の一般逆行列をyにかける)
        2 alphahat2, betahat2 = X \ y
```

```
Out[28]: 2-element Vector{Float64}:
         3.000090909090909
         0.500090909090909
```

```
In [29]: 1 # 2つの直線はぴったり重なり合う。
        2 scatter(x, y; label="", xlim=(3, 20), ylim=(2, 14), title="Anscombe 1")
        3 plot!(x → alphahat + betahat*x, 3, 20; label="", lw=2)
        4 plot!(x → alphahat2 + betahat2*x, 3, 20; label="", lw=2, ls=:dash)
```



問題: 他のアンスコムのデータについて同様のグラフを作成せよ.

### 3 Datasaurusの散布図のプロット

以下のデータは「[条件付き確率分布, 尤度, 推定, 記述統計](#)

<https://nbviewer.org/github/genkuroki/Statistics/blob/master/2022/06%20Conditional%20distribution%2C%20likelihood%2C%20estima>からのコピー＆ペースト.

#### 3.1 データの取得

- <https://www.dropbox.com/sh/xaxpz3pm5r5awes/AADUbGVagF9i4RmM9JkPtvIEa?dl=0>  
(<https://www.dropbox.com/sh/xaxpz3pm5r5awes/AADUbGVagF9i4RmM9JkPtvIEa?dl=0>)
- <https://visualizing.jp/the-datasaurus-dozen/> (<https://visualizing.jp/the-datasaurus-dozen/>)
- <https://www.openintro.org/data/index.php?data=datasaurus> (<https://www.openintro.org/data/index.php?data=datasaurus>)



In [30]:



```
1 datasaurus = [  
2     55.3846 97.1795  
3     51.5385 96.0256  
4     46.1538 94.4872  
5     42.8205 91.4103  
6     40.7692 88.3333  
7     38.7179 84.8718  
8     35.6410 79.8718  
9     33.0769 77.5641  
10    28.9744 74.4872  
11    26.1538 71.4103  
12    23.0769 66.4103  
13    22.3077 61.7949  
14    22.3077 57.1795  
15    23.3333 52.9487  
16    25.8974 51.0256  
17    29.4872 51.0256  
18    32.8205 51.0256  
19    35.3846 51.4103  
20    40.2564 51.4103  
21    44.1026 52.9487  
22    46.6667 54.1026  
23    50.0000 55.2564  
24    53.0769 55.6410  
25    56.6667 56.0256  
26    59.2308 57.9487  
27    61.2821 62.1795  
28    61.5385 66.4103  
29    61.7949 69.1026  
30    57.4359 55.2564  
31    54.8718 49.8718  
32    52.5641 46.0256  
33    48.2051 38.3333  
34    49.4872 42.1795  
35    51.0256 44.1026  
36    45.3846 36.4103  
37    42.8205 32.5641  
38    38.7179 31.4103  
39    35.1282 30.2564  
40    32.5641 32.1795  
41    30.0000 36.7949  
42    33.5897 41.4103  
43    36.6667 45.6410  
44    38.2051 49.1026  
45    29.7436 36.0256  
46    29.7436 32.1795  
47    30.0000 29.1026  
48    32.0513 26.7949  
49    35.8974 25.2564  
50    41.0256 25.2564  
51    44.1026 25.6410  
52    47.1795 28.7180  
53    49.4872 31.4103  
54    51.5385 34.8718  
55    53.5897 37.5641  
56    55.1282 40.6410  
57    56.6667 42.1795  
58    59.2308 44.4872  
59    62.3077 46.0256  
60    64.8718 46.7949  
61    67.9487 47.9487  
62    70.5128 53.7180  
63    71.5385 60.6410  
64    71.5385 64.4872  
65    69.4872 69.4872  
66    46.9231 79.8718  
67    48.2051 84.1026  
68    50.0000 85.2564  
69    53.0769 85.2564  
70    55.3846 86.0256  
71    56.6667 86.0256  
72    56.1538 82.9487  
73    53.8462 80.6410  
74    51.2821 78.7180  
75    50.0000 78.7180  
76    47.9487 77.5641  
77    29.7436 59.8718  
78    29.7436 62.1795  
79    31.2821 62.5641  
80    57.9487 99.4872
```

81	61.7949	99.1026
82	64.8718	97.5641
83	68.4615	94.1026
84	70.7692	91.0256
85	72.0513	86.4103
86	73.8462	83.3333
87	75.1282	79.1026
88	76.6667	75.2564
89	77.6923	71.4103
90	79.7436	66.7949
91	81.7949	60.2564
92	83.3333	55.2564
93	85.1282	51.4103
94	86.4103	47.5641
95	87.9487	46.0256
96	89.4872	42.5641
97	93.3333	39.8718
98	95.3846	36.7949
99	98.2051	33.7180
100	56.6667	40.6410
101	59.2308	38.3333
102	60.7692	33.7180
103	63.0769	29.1026
104	64.1026	25.2564
105	64.3590	24.1026
106	74.3590	22.9487
107	71.2821	22.9487
108	67.9487	22.1795
109	65.8974	20.2564
110	63.0769	19.1026
111	61.2821	19.1026
112	58.7179	18.3333
113	55.1282	18.3333
114	52.3077	18.3333
115	49.7436	17.5641
116	47.4359	16.0256
117	44.8718	13.7180
118	48.7179	14.8718
119	51.2821	14.8718
120	54.1026	14.8718
121	56.1538	14.1026
122	52.0513	12.5641
123	48.7179	11.0256
124	47.1795	9.8718
125	46.1538	6.0256
126	50.5128	9.4872
127	53.8462	10.2564
128	57.4359	10.2564
129	60.0000	10.6410
130	64.1026	10.6410
131	66.9231	10.6410
132	71.2821	10.6410
133	74.3590	10.6410
134	78.2051	10.6410
135	67.9487	8.7180
136	68.4615	5.2564
137	68.2051	2.9487
138	37.6923	25.7692
139	39.4872	25.3846
140	91.2821	41.5385
141	50.0000	95.7692
142	47.9487	95.0000
143	44.1026	92.6923
144	];	

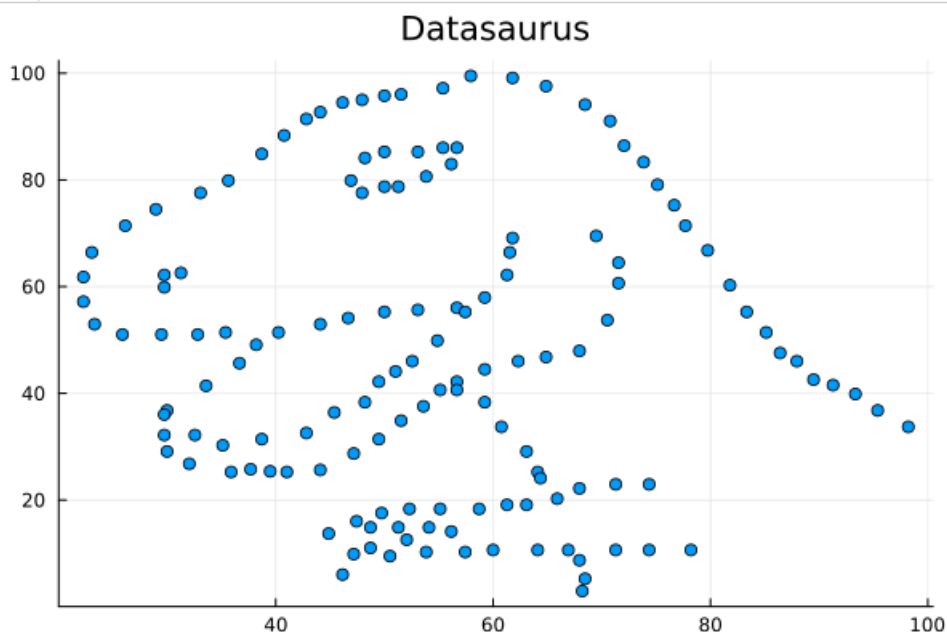
## 3.2 散布図の作成

```
In [31]: 1 # 行列Aの第j列はA[:,j]
2 @show datasaurus[:,1];
```

```
datasaurus[:, 1] = [55.3846, 51.5385, 46.1538, 42.8205, 40.7692, 38.7179, 35.641, 33.0769, 28.9744, 26.1538, 23.0769, 22.3077, 22.3077, 23.3333, 25.8974, 29.4872, 32.8205, 35.3846, 40.2564, 44.1026, 46.6667, 50.0, 53.0769, 56.6667, 59.2308, 61.2821, 61.5385, 61.7949, 57.4359, 54.8718, 52.5641, 48.2051, 49.4872, 51.0256, 45.3846, 42.8205, 38.7179, 35.1282, 32.5641, 30.0, 33.5897, 36.6667, 38.2051, 29.7436, 29.7436, 30.0, 32.0513, 35.8974, 41.0256, 44.1026, 47.1795, 49.4872, 51.5385, 53.5897, 55.1282, 56.6667, 59.2308, 62.3077, 64.8718, 67.9487, 70.5128, 71.5385, 71.5385, 69.4872, 46.9231, 48.2051, 50.0, 53.0769, 55.3846, 56.6667, 56.1538, 53.8462, 51.2821, 50.0, 47.9487, 29.7436, 29.7436, 31.2821, 57.9487, 61.7949, 64.8718, 68.4615, 70.7692, 72.0513, 73.8462, 75.1282, 76.6667, 77.6923, 79.7436, 81.7949, 83.3333, 85.1282, 86.4103, 87.9487, 89.4872, 93.3333, 95.3846, 98.2051, 56.6667, 59.2308, 60.7692, 63.0769, 64.1026, 64.359, 74.359, 71.2821, 67.9487, 65.8974, 63.0769, 61.2821, 58.7179, 55.1282, 52.3077, 49.7436, 47.4359, 44.8718, 48.7179, 51.2821, 54.1026, 56.1538, 52.0513, 48.7179, 47.1795, 46.1538, 50.5128, 53.8462, 57.4359, 60.0, 64.1026, 66.9231, 71.2821, 74.359, 78.2051, 67.9487, 68.4615, 68.2051, 37.6923, 39.4872, 91.2821, 50.0, 47.9487, 44.1026]
```

```
In [32]: 1 scatter(datasaurus[:,1], datasaurus[:,2]; label="", title="Datasaurus")
```

Out[32]:



問題: [Datasaurusについて検索 \(https://www.google.com/search?q=Datasaurus\)](https://www.google.com/search?q=Datasaurus) して見つけた解説を読め。

## 4 中心極限定理のプロット

### 4.1 素朴なワークフロー

以下のセルの内容を julia の `julia>` プロンプトに順番に入力すれば(コピー & ペーストすれば)同じ結果が得られる。各行の最後にセミコロン `;` を追加すれば計算結果の出力を抑制できる。

```
In [33]: 1 # 確率分布を dist と書く。
2 dist = Gamma(2, 3)
```

Out[33]: `Gamma{Float64}(α=2.0, θ=3.0)`

```
In [34]: ▶ 1 # 確率分布 dist のサイズ n のサンプルを L 個生成
2 n = 10
3 L = 104
4 Xs = [rand(dist, n) for _ in 1:L]
```

```
Out[34]: 10000-element Vector{Vector{Float64}}:
 [6.435028031967606, 5.5202236420304445, 18.93665254889968, 14.545107674245921, 1.1686763514845
 737, 9.19839791258536, 2.002566713142896, 2.8507939646481035, 3.9778556316198994, 2.06649427836
 57345]
 [11.05431340499406, 4.7506637699969705, 3.324798065733279, 15.13381275191475, 6.88570907355971
 9, 7.16502896513472, 7.926514968910601, 9.358763599830395, 4.398868291277946, 1.002035905960071
 2]
 [12.728386849546037, 0.7098808009318759, 6.375648089502405, 0.2650046815408918, 6.932624328782
 614, 1.7365457197454557, 5.319797607147697, 11.106451857008427, 6.2883445444778445, 11.52728129
 0283145]
 [9.597758506139794, 10.111763694160347, 4.34596267075415, 6.7551893318791, 8.221499426849327,
 20.186100830224213, 3.219647103113612, 11.087439960974013, 11.21953991191235, 4.94552890217622
 7]
 [13.779720831022594, 12.270469129312314, 8.356971290818686, 10.253847246367565, 0.860032618248
 8083, 5.247103833347676, 12.225913458080516, 3.0642555547283488, 14.07853686989805, 9.468749926
 659632]
 [1.84164321114844, 0.14895906927210062, 11.273187555112445, 3.5324425012199723, 5.614561058631
 877, 4.397167838333208, 10.75372689027117, 3.70522236740811, 1.6160084237158914, 7.99782720268
 1133]
 [4.734962895993206, 2.112526674958939, 1.020944578422427, 6.826446345685925, 3.913362573300072
 5, 3.6234012071388926, 5.264410025760969, 20.256271198009546, 3.6412086963950427, 7.44437421019
 93635]
 [6.513031896711741, 14.907215517160381, 2.815993484873153, 4.709052452916216, 5.99034833066616
 4, 2.544578343095543, 2.764849998139055, 3.807861000807573, 5.314977059056595, 2.27587194253250
 2]
 [0.7052283410012068, 7.128983327183754, 2.851218224450956, 9.772516525468477, 2.22238545702122
 36, 7.345640176140877, 12.589370554015822, 4.299607835393241, 1.4404281261277978, 1.64793978933
 54482]
 [0.7042070722320246, 4.628858856557481, 0.9807484124996912, 9.692353178951402, 4.4027214646743
 74, 12.195576035171317, 3.470286094381709, 5.2028863179439195, 7.864447668275659, 1.21505941172
 38853]
 [2.5523409230127054, 0.8859134651433248, 10.331488828337632, 3.0265782484859614, 5.20857291779
 6248, 2.7459633383671496, 5.952640326300393, 9.789784517999927, 3.0451937296694247, 1.778680491
 3485428]
 [7.362344669499334, 0.8052563528412954, 6.801722543272977, 7.2771628500835455, 8.4326679428888
 59, 1.2149537543720725, 2.347785528856806, 14.372796769945527, 4.406370219553867, 1.82438883028
 53835]
 [8.733575132320805, 7.833892780405475, 7.732828502170568, 5.239981630968478, 6.73769304733766
 6, 5.883889724785667, 4.89096206294479, 3.5710587323880176, 10.820681606128142, 8.1087195157080
 39]
 ⋮
 [10.587443369361633, 5.071771578858746, 18.344707098754903, 3.1592544375860503, 11.21180208193
 5964, 7.239075088689536, 11.769042848303018, 7.693132146734982, 9.383396127436399, 7.5488081320
 35125]
 [4.435890445662884, 8.917786994572033, 15.008961773999413, 1.483368086795855, 11.5860094589391
 03, 3.342577132008422, 1.94506127337108, 7.987775044461232, 4.44998534608557, 3.31675557189129
 9]
 [2.883208879190245, 4.134987422969188, 4.70073940461959, 3.011728818348626, 7.9436298207405, 1
 0.68184962010822, 5.028942057901908, 4.440396029600737, 3.909270783043222, 3.0388153382682574]
 [3.316764331135285, 1.7256467603350991, 3.8924410760110506, 4.232796447263006, 3.2729034695568
 116, 27.076326065394593, 15.047412981080395, 6.5605996666348965, 4.917761301272014, 5.106725651
 136027]
 [7.4482270459531446, 3.618815953153567, 2.2816284846998953, 0.17760356943692446, 1.93304380657
 14164, 8.660050635930695, 8.935144235830869, 2.9458645972723825, 1.4154614598935402, 16.5909984
 5857407]
 [3.369136456221207, 7.169381979333802, 9.292648890347586, 10.341615610561458, 9.41277655808944
 8, 2.8303052674440634, 4.211294643511569, 4.2666973766735135, 5.6740951404814, 2.63320449709405
 54]
 [3.335807379575362, 6.16488544045245, 4.644665868590204, 2.1363885512284826, 0.144601106706437
 5, 5.3338436463309495, 3.577057266831382, 6.12156745118289, 5.46026915203866, 4.79599710549636
 3]
 [10.216119428222594, 2.0926206138180463, 0.6715248750819816, 2.9596474908447163, 5.56011364450
 8776, 5.284353967490761, 1.4354815801421266, 8.40843612426726, 4.728444802540432, 7.94178268066
 6996]
 [13.595295582532819, 8.833423770337824, 5.271881516075578, 11.4165386042134, 9.42376538229325,
 9.805734219947777, 5.171956405707512, 10.90183364661856, 4.931521255604856, 11.011535941420671]
 [2.0289942296697543, 2.46170789873583, 11.97217714473473, 2.3272405884179688, 7.9832343145789
 5, 4.36777749535647, 13.512083959287207, 2.771578552065134, 12.45709230091004, 12.2107859107782
 56]
 [3.844028805819601, 10.808676556597486, 10.875178091474197, 8.028321936027108, 5.8294225300686
 22, 5.445278168829813, 6.840999281095747, 4.6821585546333, 23.16895727395986, 9.33047949567519
 7]
 [4.094799765842113, 1.7758452151576316, 2.2501210243648857, 3.9653020238286585, 4.145406451521
 063, 5.647052635883619, 1.949711143248154, 4.4717466069420615, 1.070804607710305, 8.39104704743
 9754]
```

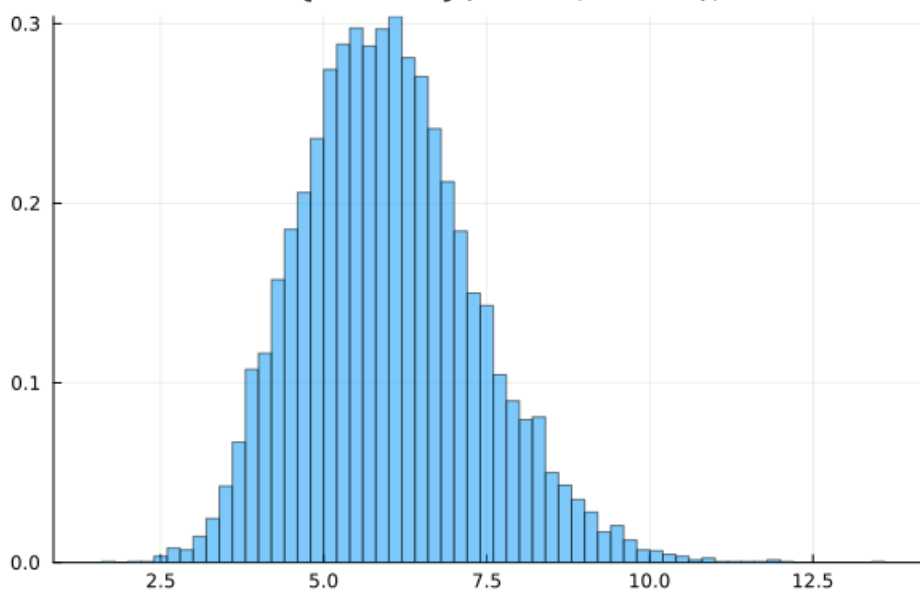
```
In [35]: 1 # L個のサイズnのサンプルの各々の標本平均を計算
        2 Xbars = mean.(Xs)
```

Out[35]: 10000-element Vector{Float64}:

```
6.670179674899022
7.100050879731251
6.29899657689664
8.969043033818313
8.960560075848417
5.088074598712704
5.883790840586438
5.1643780025958925
5.0003318356138795
5.035714451241146
4.5317156786461315
5.484544946159967
6.9553282735157635
⋮
9.200843290969635
6.247417112778689
4.977356817479049
7.514937774981918
5.40068382473165
5.92011564197581
4.171508296843319
4.929852520758368
9.036348632475228
7.209267239453434
8.885350069418092
3.7761836521938243
```

```
In [36]: 1 # Xbarのヒストグラムを表示
        2 histogram(Xbars; norm=true, alpha=0.5, label="", title="$dist, n=$n")
```

Out[36]: Gamma{Float64}( $\alpha=2.0$ ,  $\theta=3.0$ ), n=10



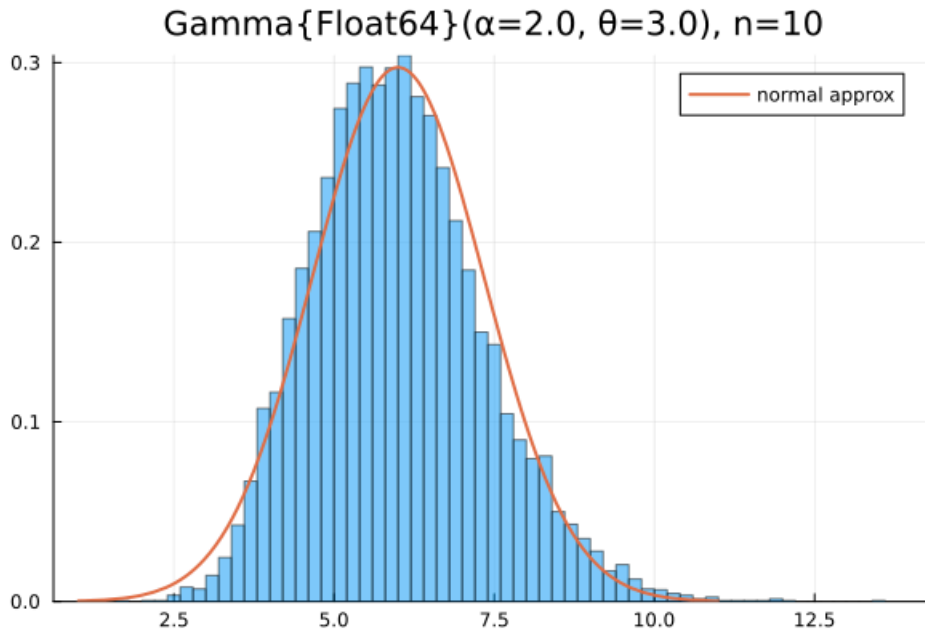
```
In [37]: 1 # 中心極限定理による正規分布近似を設定
        2 mu = mean(dist)
        3 sigma = std(dist)
        4 normal_approx = Normal(mu, sigma/sqrt(n))
```

Out[37]: Normal{Float64}( $\mu=6.0$ ,  $\sigma=1.3416407864998736$ )



```
In [38]: 1 # 上のグラフに重ねて正規分布をプロット
2 plot!(normal_approx; label="normal approx", lw=2)
```

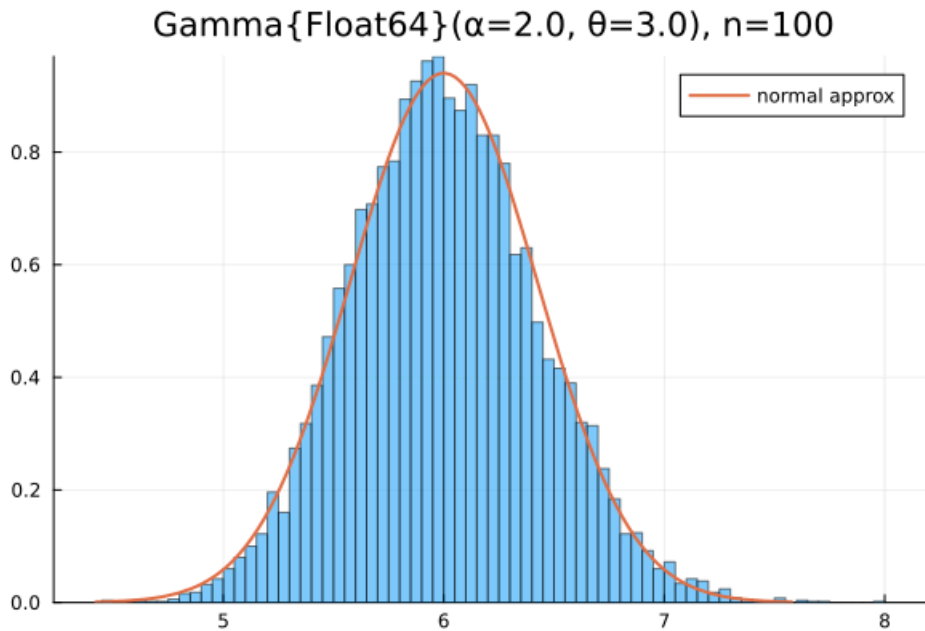
Out[38]:



$n = 10$  が小さすぎてずれが大きい.

```
In [39]: 1 # nを大きくしてやり直してみる.
2 n = 100
3 L = 10^4
4 Xs = [rand(dist, n) for _ in 1:L]
5 Xbars = mean.(Xs)
6 histogram(Xbars; norm=true, alpha=0.5, label="", title="$dist, n=$n")
7 mu = mean(dist)
8 sigma = std(dist)
9 normal_approx = Normal(mu, sigma/sqrt(n))
10 plot!(normal_approx; label="normal approx", lw=2)
```

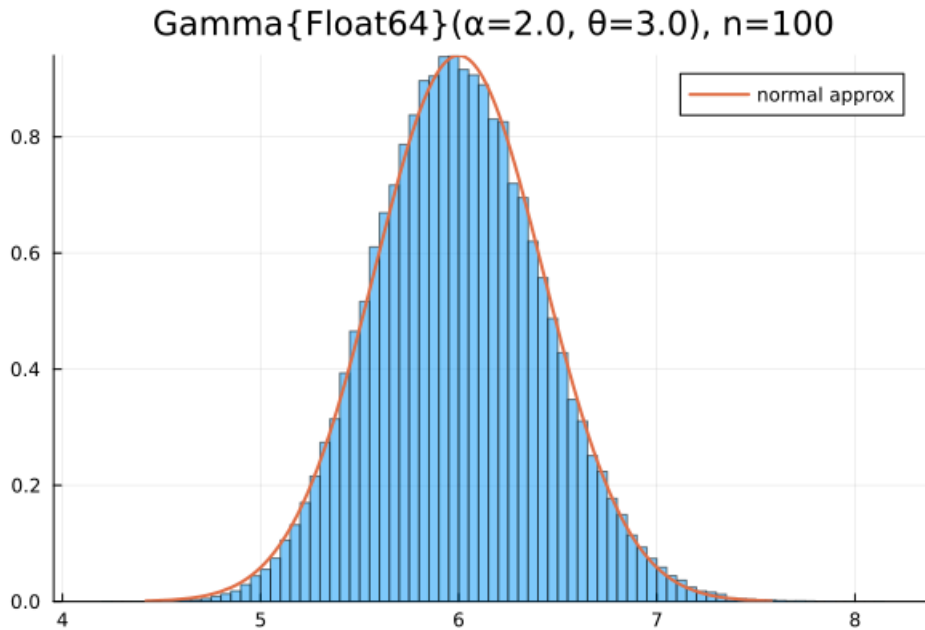
Out[39]:



$n = 100$  にしたら, 正規分布とよく一致するようになった.

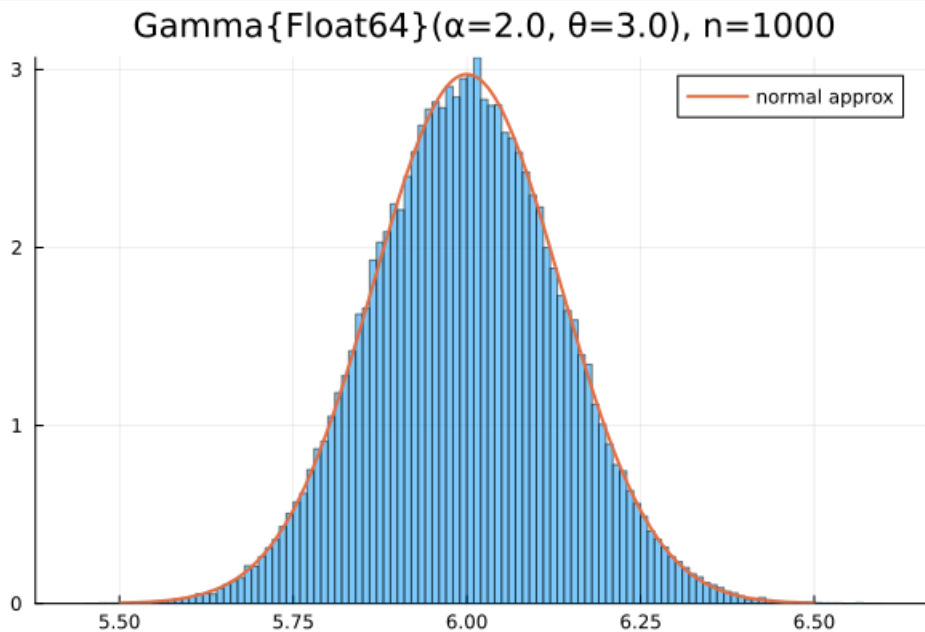
```
In [40]: 1 # Lも大きくしてやり直してみる.
2 n = 100
3 L = 10^5
4 Xs = [rand(dist, n) for _ in 1:L]
5 Xbars = mean.(Xs)
6 histogram(Xbars; norm=true, alpha=0.5, label="", title="$dist, n=$n")
7 mu = mean(dist)
8 sigma = std(dist)
9 normal_approx = Normal(mu, sigma/sqrt(n))
10 plot!(normal_approx; label="normal approx", lw=2)
```

Out[40]:



```
In [41]: 1 # Lも大きくしてやり直してみる.
2 n = 1000
3 L = 10^5
4 Xs = [rand(dist, n) for _ in 1:L]
5 Xbars = mean.(Xs)
6 histogram(Xbars; norm=true, alpha=0.5, label="", title="$dist, n=$n")
7 mu = mean(dist)
8 sigma = std(dist)
9 normal_approx = Normal(mu, sigma/sqrt(n))
10 plot!(normal_approx; label="normal approx", lw=2)
```

Out[41]:



## 4.2 Revise.jlを使うワークフロー

上のように素朴に毎回コードを入力することは非常に面倒である。

似た仕事は関数化して1行の入力で実行できるようにしておく方がよい。

しかし、関数の定義を `julia> プロンプトに直接入力すると、試行錯誤で関数の定義を何度も変える作業が非常に面倒になる。`

もしも、関数の定義をファイルに書いておき、ファイル内の関数の定義を書き換えると、自動的に `julia>` プロンプトの側に関数の定義の変更が反映されるようにできれば非常に便利である。それを実現するのが [Revise.jl](https://github.com/timholly/Revise.jl) (<https://github.com/timholly/Revise.jl>) パッケージである。Revise.jlパッケージは

```
pkg> add Revise
```

でインストールできる。

### 4.3 問題: 自分で関数を定義して実行してみよ.

以下のセルのように関数を定義しておくと、同じような仕事を何度も楽に実行できるようになる。

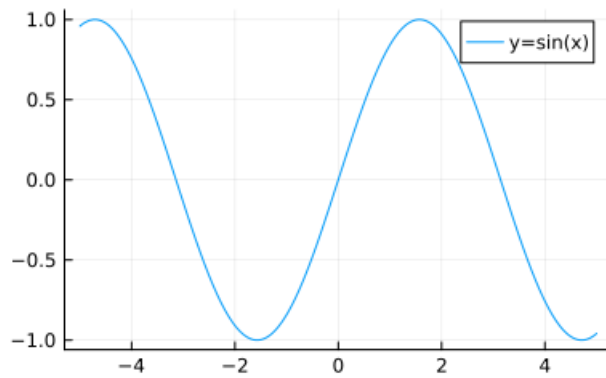
```
In [42]: 1 using StatsPlots
2 using Distributions
3 default(size=(400, 250), titlefontsize=10)
4
5 function hello_sine()
6     println("Hello, Sine!")
7     plot(sin; label="y=sin(x)")
8 end
9
10 function plot_central_limit_theorem(dist, n; L=10^4, bin=:auto)
11     distname = mydistname(dist)
12     mu = mean(dist)
13     sigma = std(dist)
14     Xs = [rand(dist, n) for _ in 1:L]
15     Xbars = mean.(Xs)
16     normal_approx = Normal(mu, sigma/sqrt(n))
17
18     if dist isa DiscreteUnivariateDistribution
19         mu = mean(dist)
20         sigma = std(dist)
21         a = round(n*mu - 4.5sqrt(n)*sigma)
22         b = round(n*mu + 4.5sqrt(n)*sigma)
23         ran = a-0.5:b+0.5
24         bin = ran / n
25     end
26
27     histogram(Xbars; bin, norm=true, alpha=0.5, label="Xbars")
28     plot!(normal_approx; lw=2, label="normal approx")
29     title!("$distname, n=$n")
30 end
31
32 mypdf(dist, x) = pdf(dist, x)
33 mypdf(dist::DiscreteUnivariateDistribution, x) = pdf(dist, round{Int}(x))
34 mydistname(dist) = replace(string(dist), r"{\[\^}\]*}" => "")
35
36 function plot_dist(dist; xlim0=nothing)
37     distname = mydistname(dist)
38     if isnothing(xlim0)
39         mu = mean(dist)
40         sigma = std(dist)
41         a = max(minimum(dist), mu - 4.5sigma)
42         b = min(maximum(dist), mu + 4.5sigma)
43         if dist isa DiscreteUnivariateDistribution
44             a, b = a-1, b+1
45         else
46             a, b = a-0.025(b-a), b+0.025(b-a)
47         end
48         xlim0 = (a, b)
49     end
50     plot(x -> mypdf(dist, x), xlim0...; label="", title="$distname")
51 end
52
53 function plot_dist_clt(dist, n; L=10^4, xlim0=nothing)
54     P0 = plot_dist(dist; xlim0)
55     P1 = plot_central_limit_theorem(dist, n; L)
56     plot(P0, P1; size=(800, 250), layout=(1, 2))
57 end
```

```
Out[42]: plot_dist_clt (generic function with 1 method)
```

```
In [43]: 1 hello_sine()
```

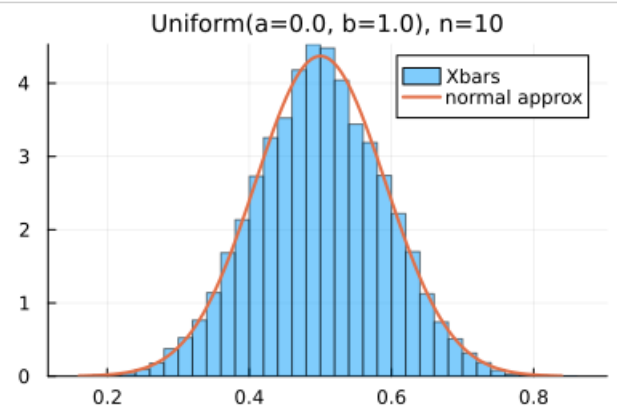
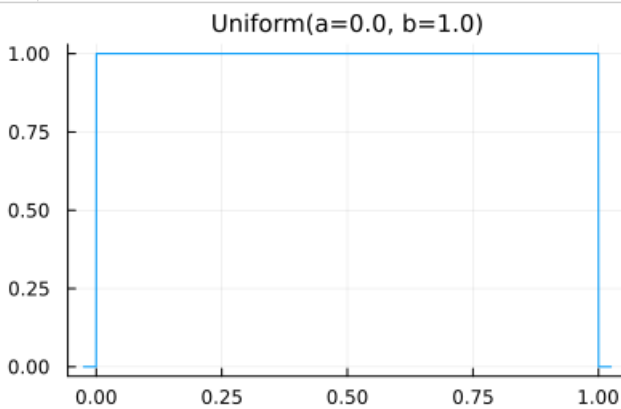
Hello, Sine!

Out[43]:



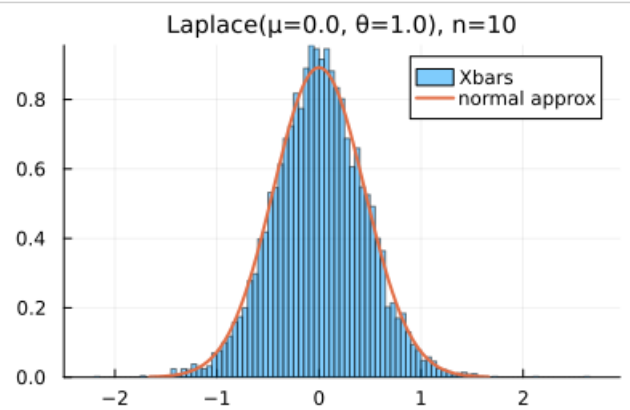
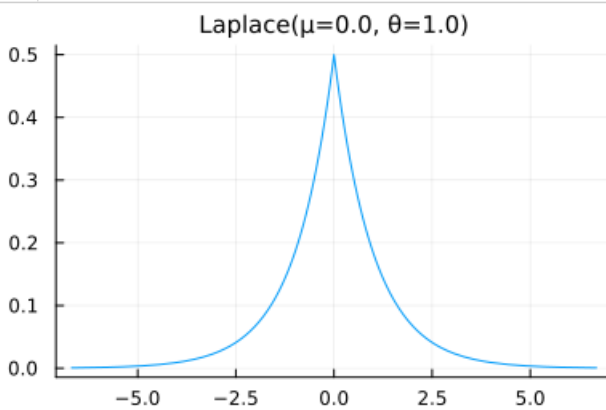
```
In [44]: 1 plot_dist_clt(Uniform(), 10)
```

Out[44]:



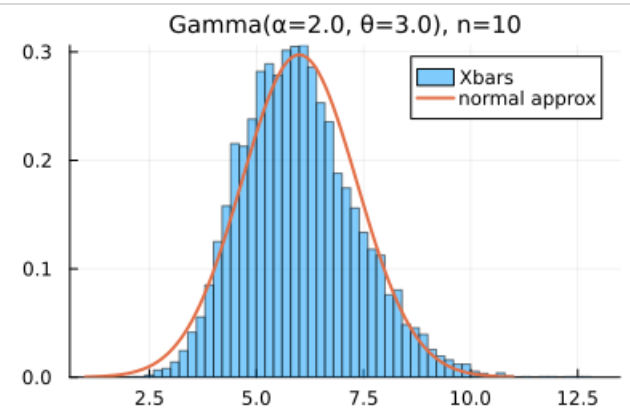
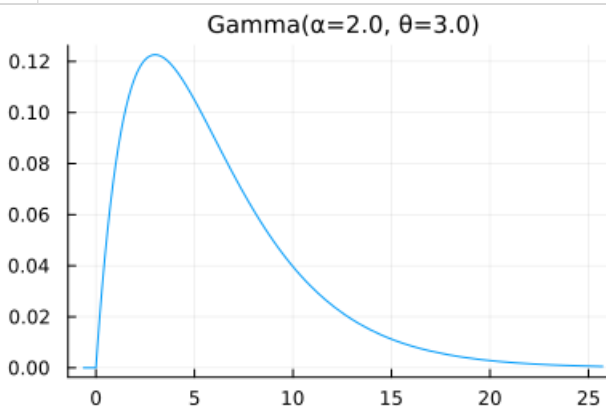
```
In [45]: 1 plot_dist_clt(Laplace(), 10)
```

Out[45]:



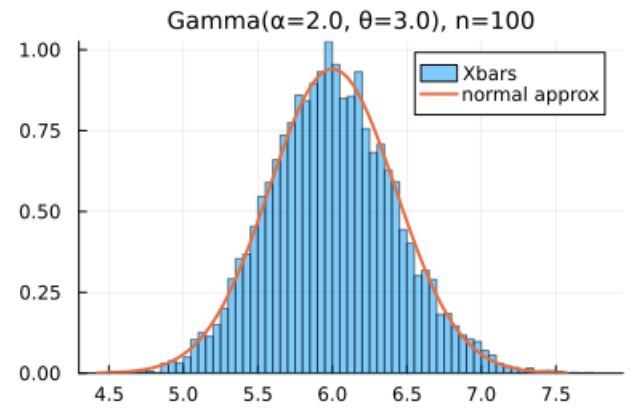
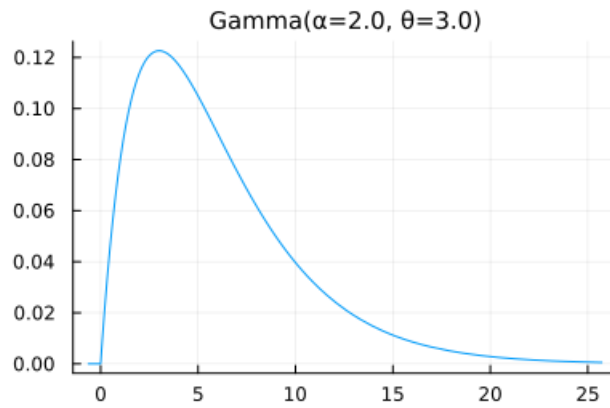
```
In [46]: 1 plot_dist_clt(Gamma(2, 3), 10)
```

Out[46]:



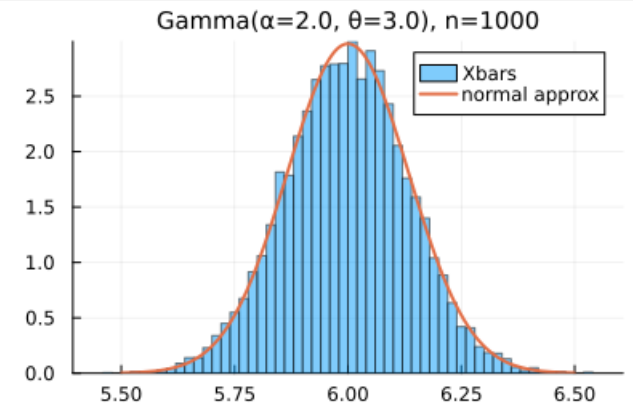
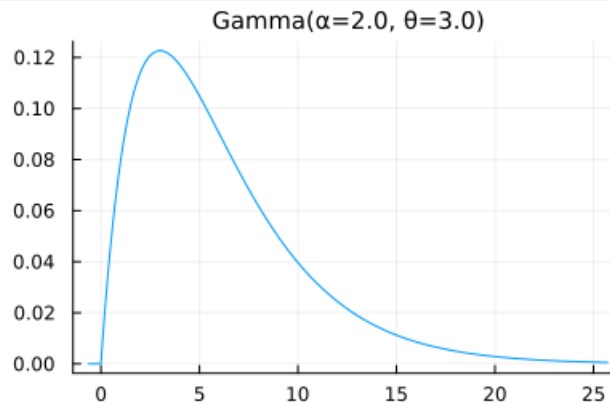
In [47]: 1 plot\_dist\_clt(Gamma(2, 3), 100)

Out[47]:



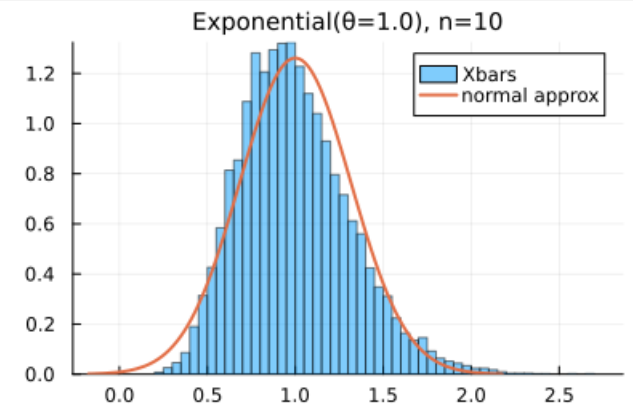
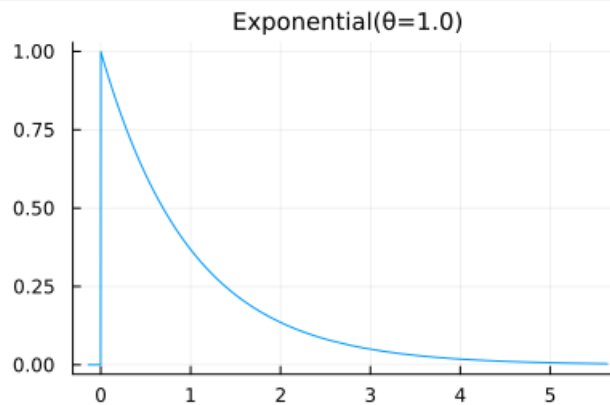
In [48]: 1 plot\_dist\_clt(Gamma(2, 3), 1000)

Out[48]:



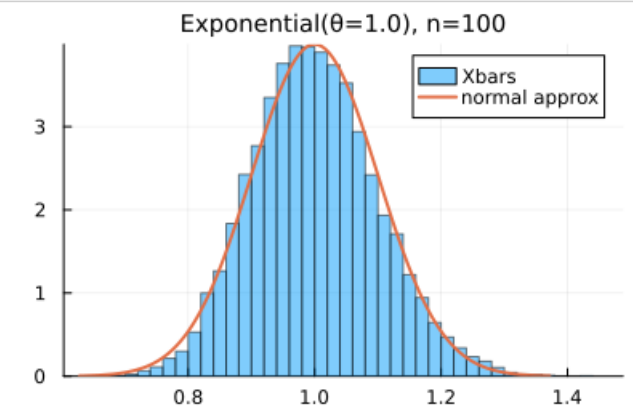
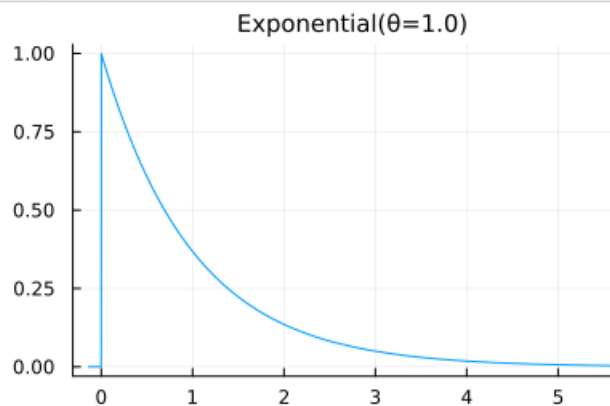
In [49]: 1 plot\_dist\_clt(Exponential(), 10)

Out[49]:



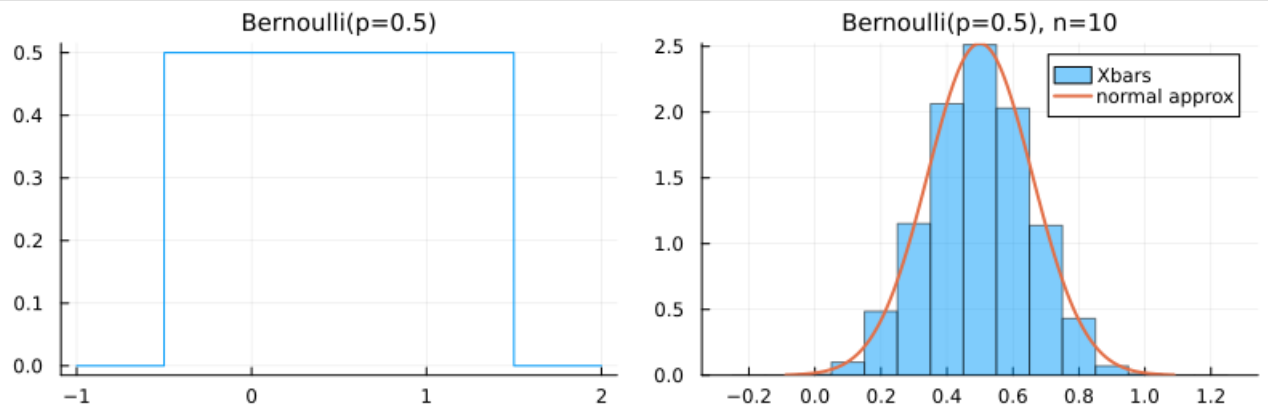
In [50]: 1 plot\_dist\_clt(Exponential(), 100)

Out[50]:



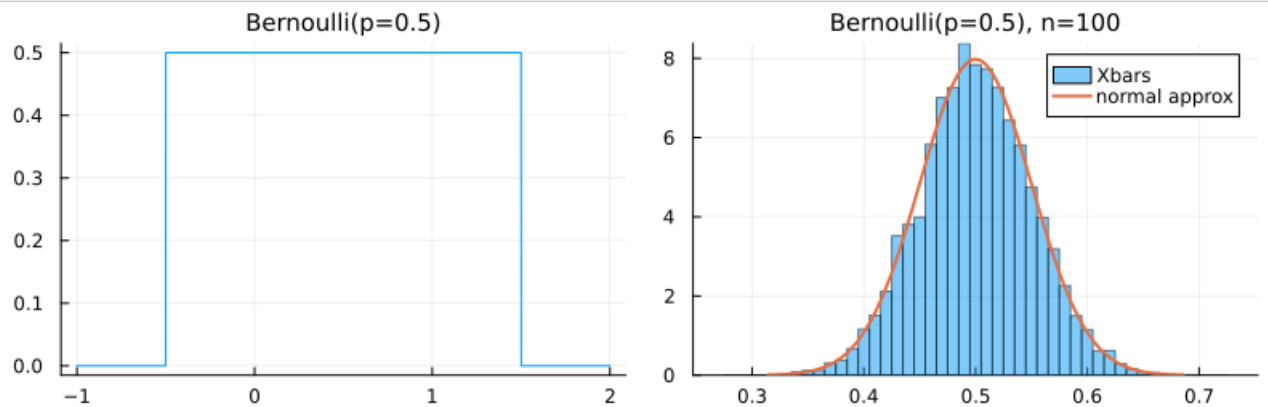
```
In [51]: 1 plot_dist_clt(Bernoulli(), 10)
```

Out[51]:



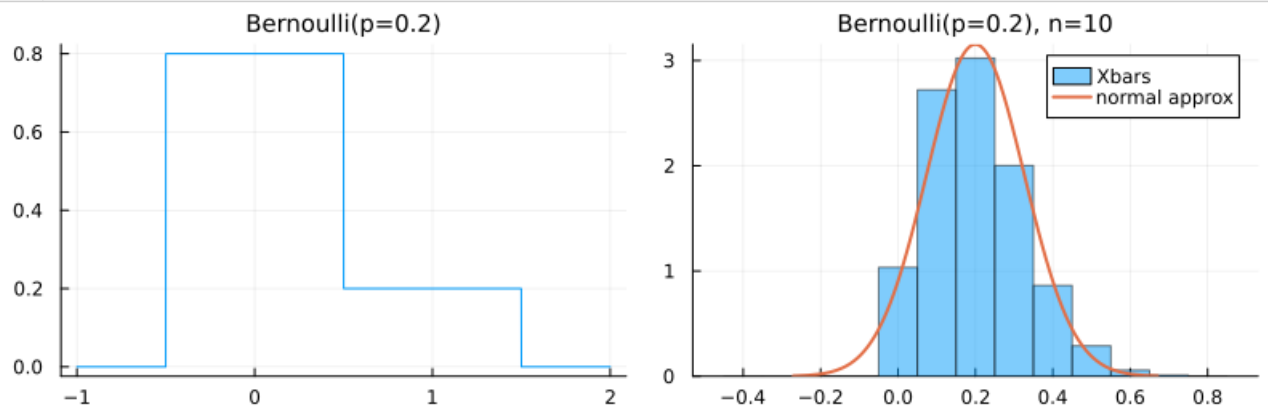
```
In [52]: 1 plot_dist_clt(Bernoulli(), 100)
```

Out[52]:



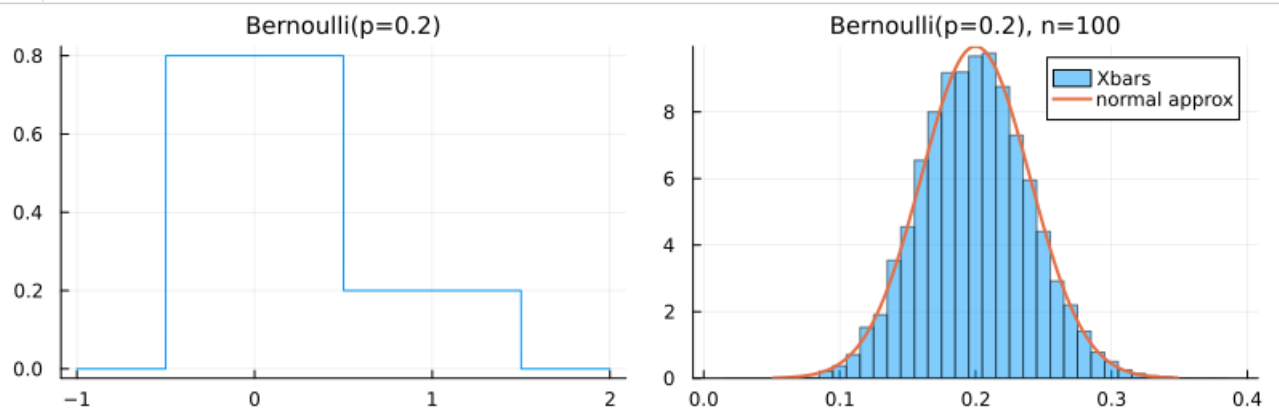
```
In [53]: 1 plot_dist_clt(Bernoulli(0.2), 10)
```

Out[53]:



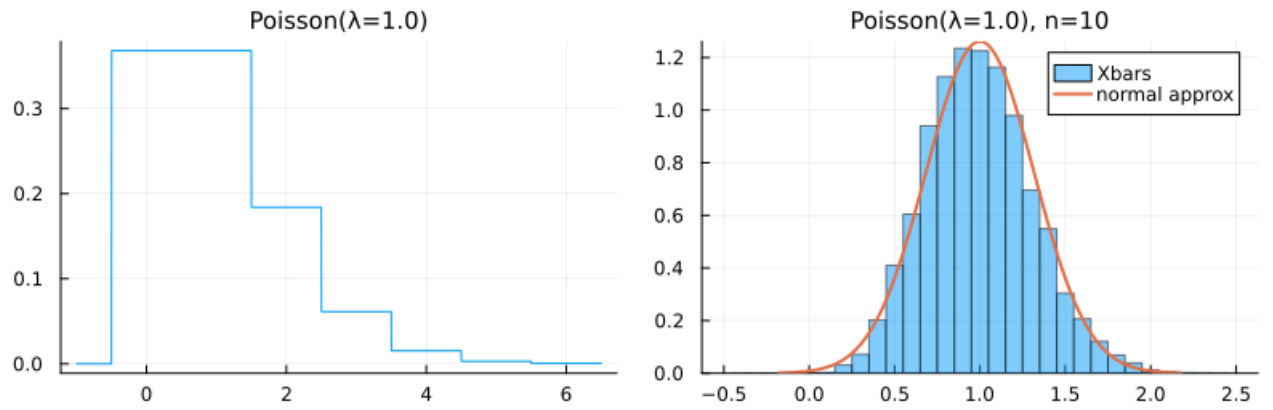
```
In [54]: 1 plot_dist_clt(Bernoulli(0.2), 100)
```

Out[54]:



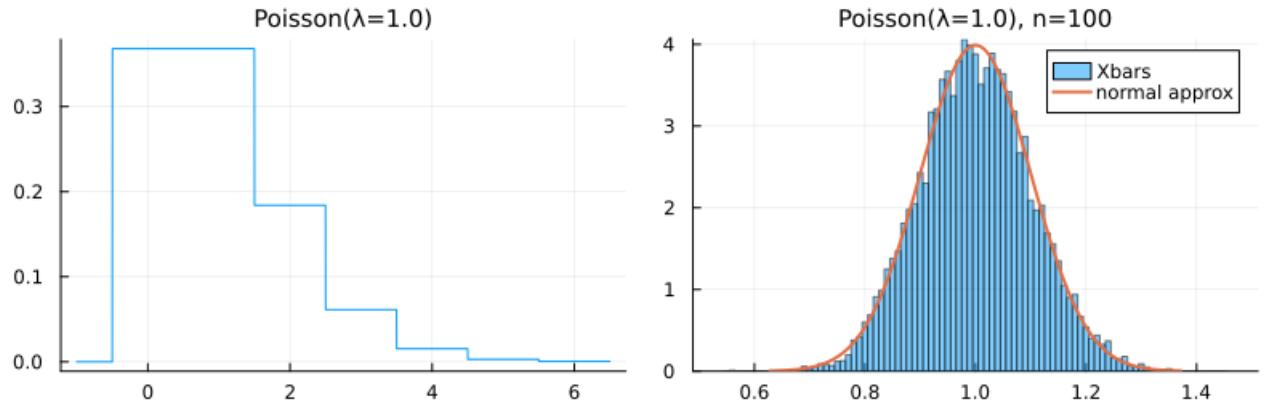
```
In [55]: 1 plot_dist_clt(Poisson(), 10)
```

Out[55]:



```
In [56]: 1 plot_dist_clt(Poisson(), 100)
```

Out[56]:



```
In [ ]: 1
```