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?hl=ja)](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?hl=ja).

目次 ¶

In [1]:
```julia
# 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

import Pkg

"""すでにPkg.add済みのパッケージのリスト"""
packages_added = [info.name for (uuid, info) in Pkg.dependencies() if info.is_direct_dep]

"""必要ならPkg.assした後にusingしてくれる関数"""
function _using(pkg::AbstractString)
    if pkg in packages_added
        println("# $(pkg).jl is already added.")
    else
        println("# $(pkg).jl is not added yet, so let's add it.")
        Pkg.add(pkg)
    end
    println("> using $(pkg)")
    @eval using $(Symbol(pkg))
end

"""必要ならPkg.addした後にusingしてくれるマクロ"""
macro _using(pkg) :(_using($(string(pkg)))) end

@_using Distributions
@_using RDatasets
@_using StatsPlots
default(fmt = :png)
```

```
# Distributions.jl is already added.
> using Distributions
# RDatasets.jl is already added.
> using RDatasets
# StatsPlots.jl is already added.
> using StatsPlots
```

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える.
- 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)
```
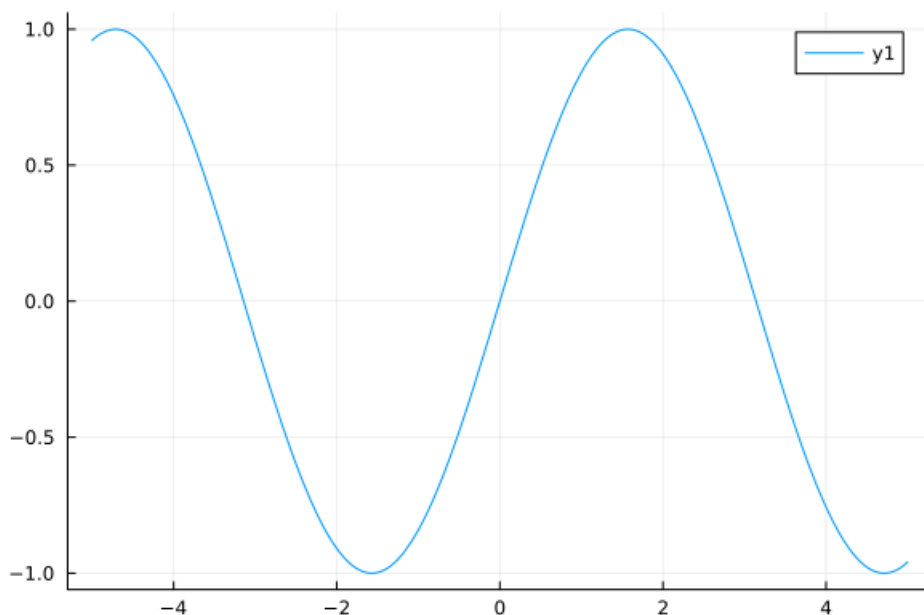Out[4]: 0.5

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
```

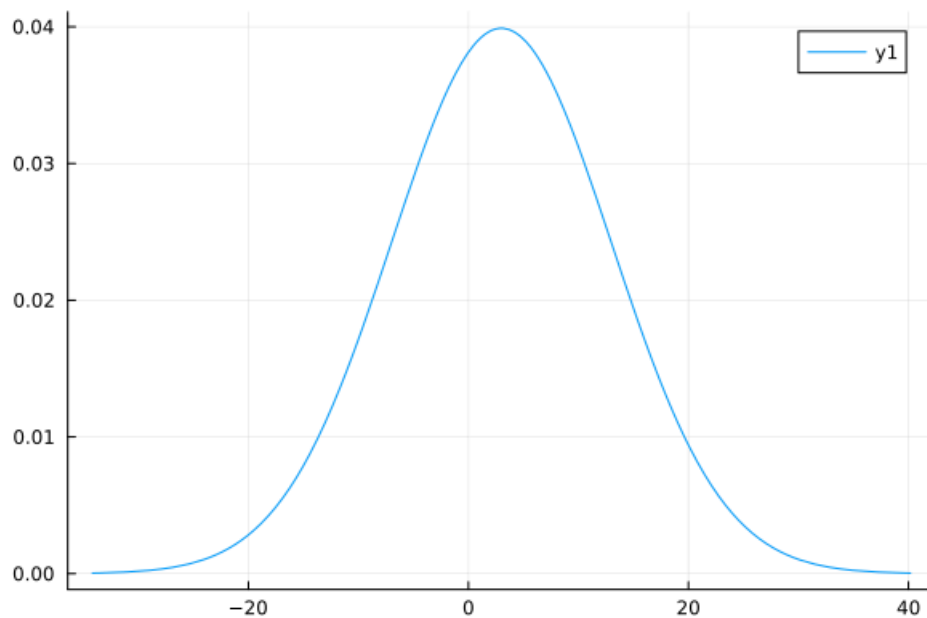このノートブックでは最初のセルでこれと同等のことを実行できるようにしてあるので, 最初のセルを実行しておけばよい.

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

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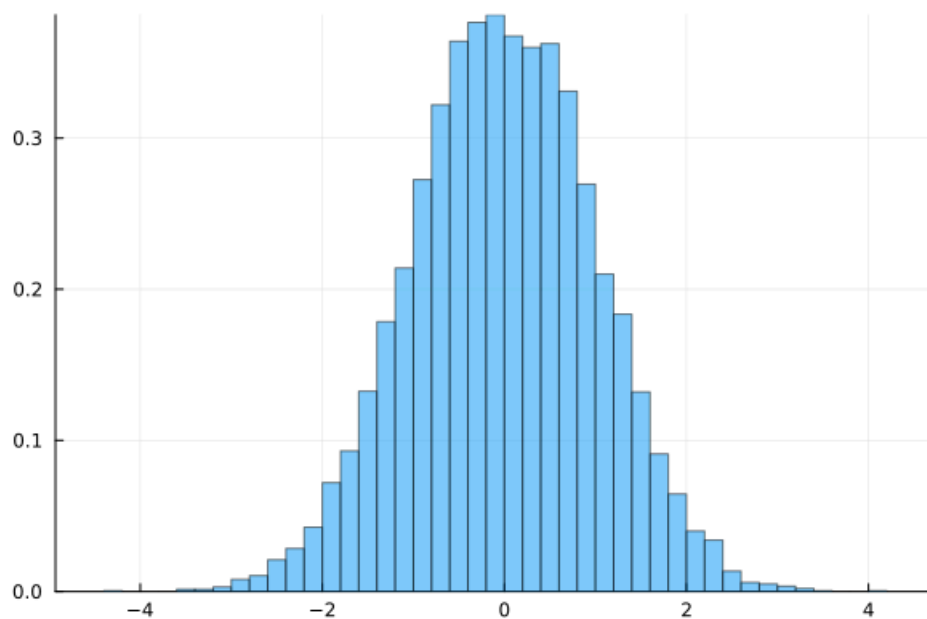
Out[6]:

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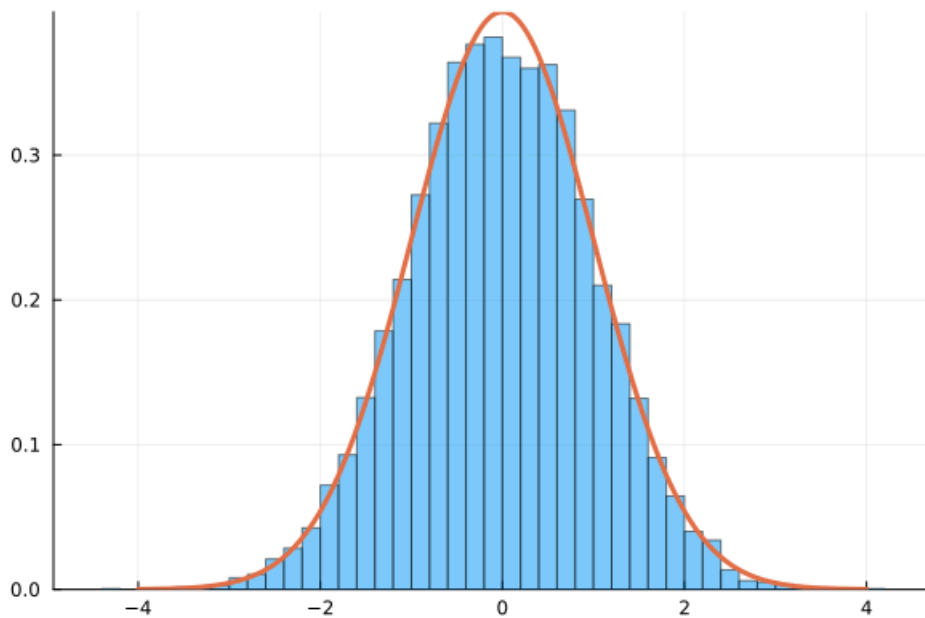
```
In [8]:  ▶  1  histogram(Z; norm=true, alpha=0.5, label="")
```

Out[8]:

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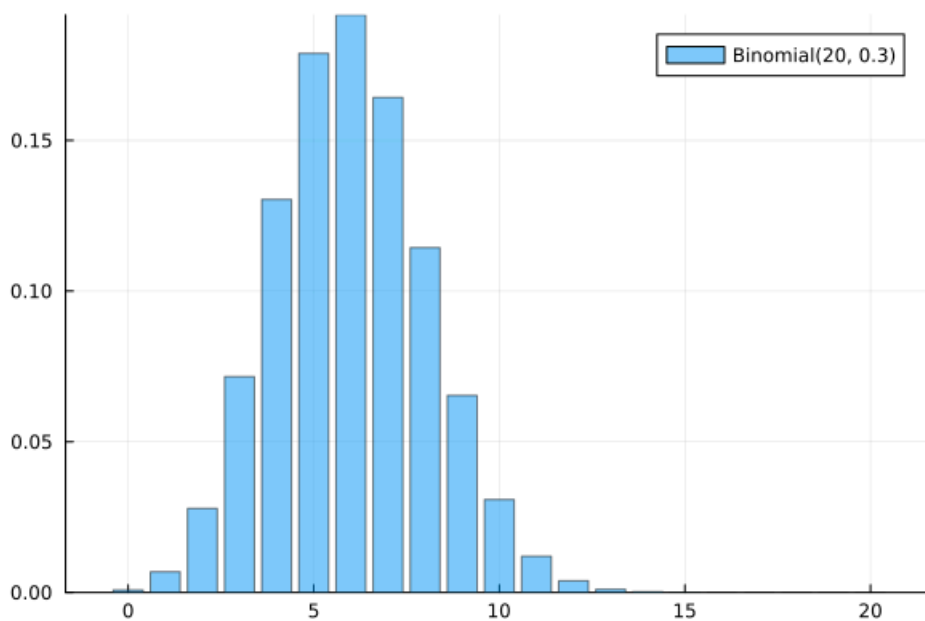
Out[9]:

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函数のプロット

```
In [11]:  ▶  1  X = rand(Normal(2, 3), 10^4);
```
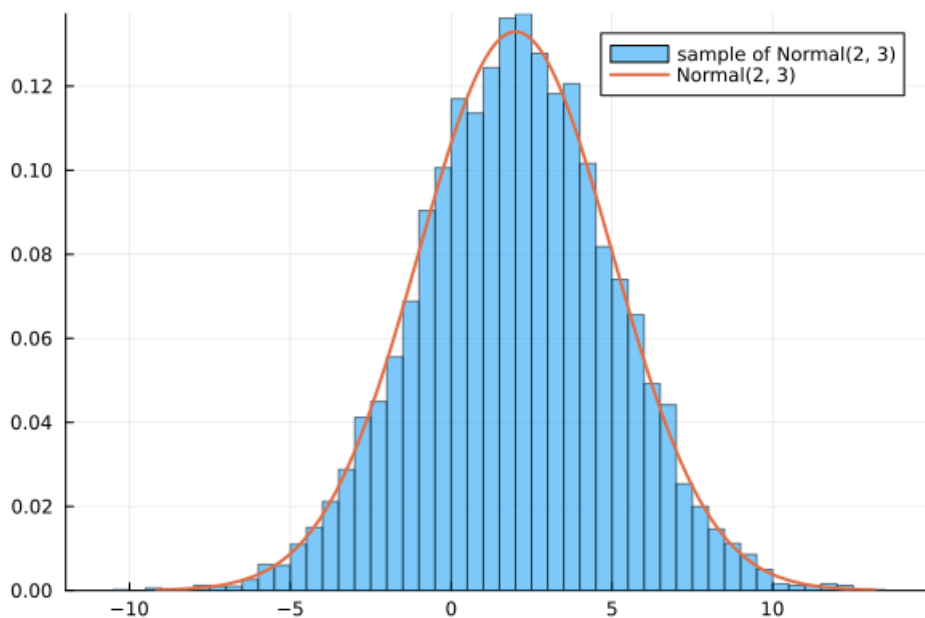
```
In [12]:  ▶  1  histogram(X; norm=true, alpha=0.5, label="sample of Normal(2, 3)")
```

Out[12]:
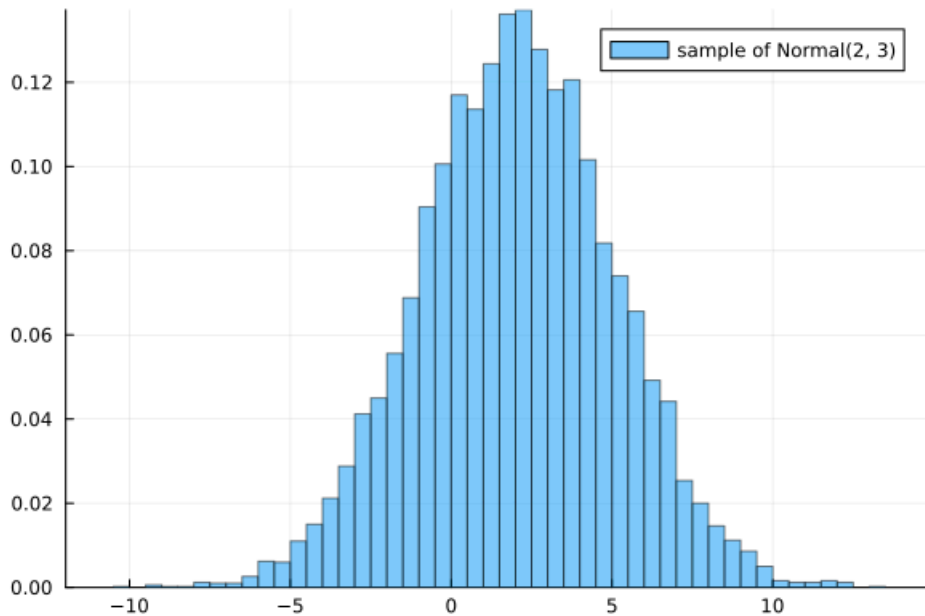

```
In [13]:  ▶  1  plot!(Normal(2, 3); label="Normal(2, 3)", lw=2)
```

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

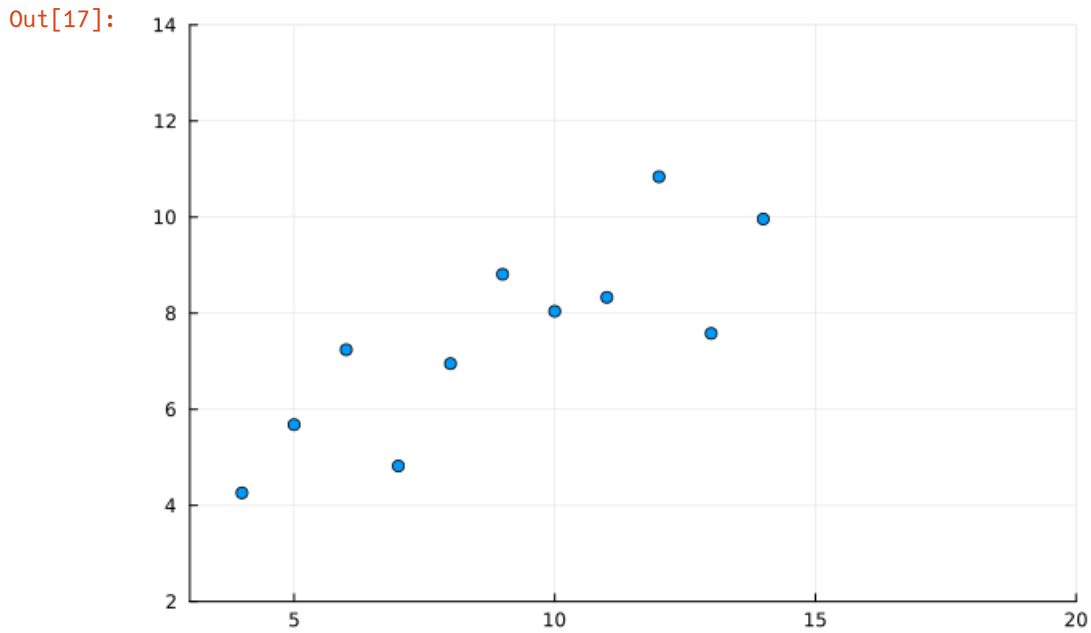以下ではデータ1の場合のプロットの仕方を説明しよう.

```
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)
```

Out[16]:
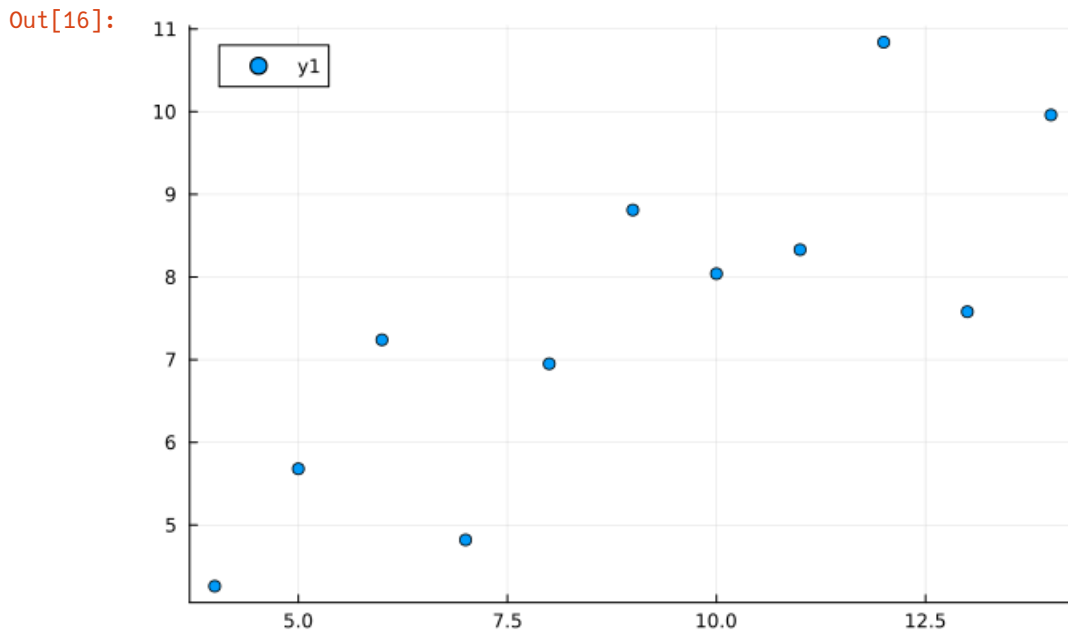
```
In [17]:  ▶   1  # xlim, ylimなどを追加
              2  scatter(x, y; label="", xlim=(3, 20), ylim=(2, 14))
```

Out[17]:


```
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)
```
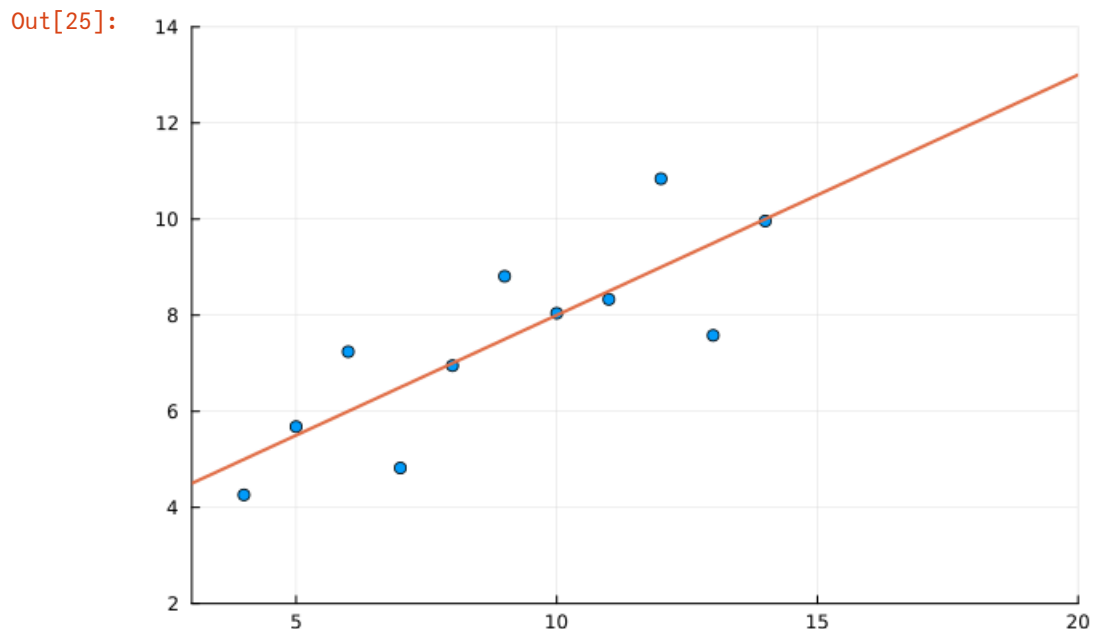
Out[22]: 5.501

```
In [23]:  ▶   1  betahat = sxy/sx2
```
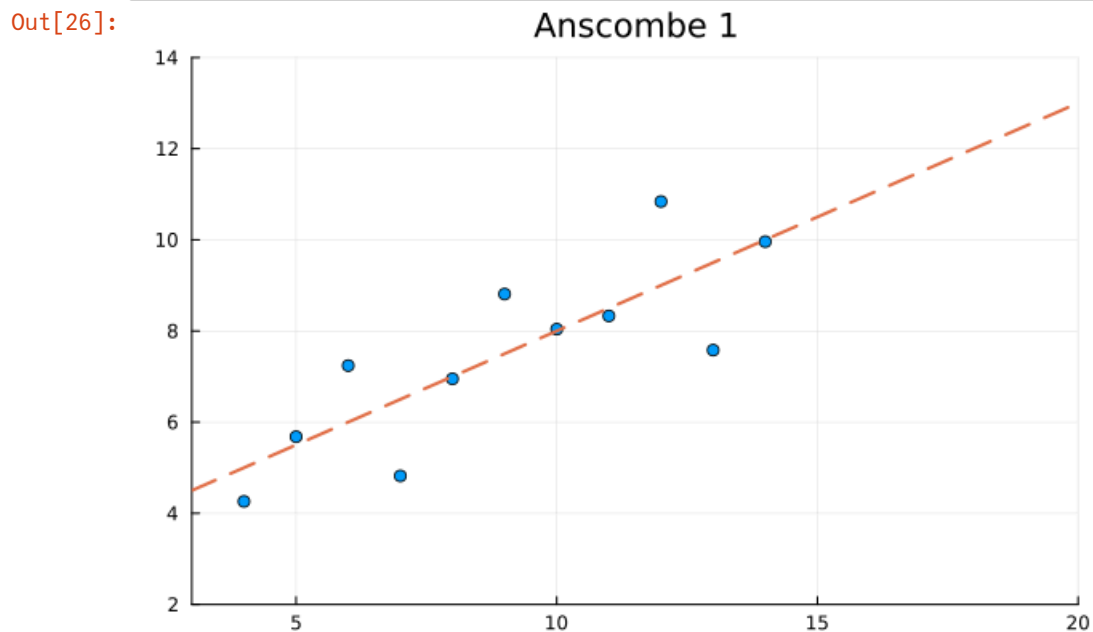
Out[23]: 0.5000909090909091

```
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)
```

Out[25]:

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)
```

Out[26]:

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

Out[27]:
```
11×2 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
```
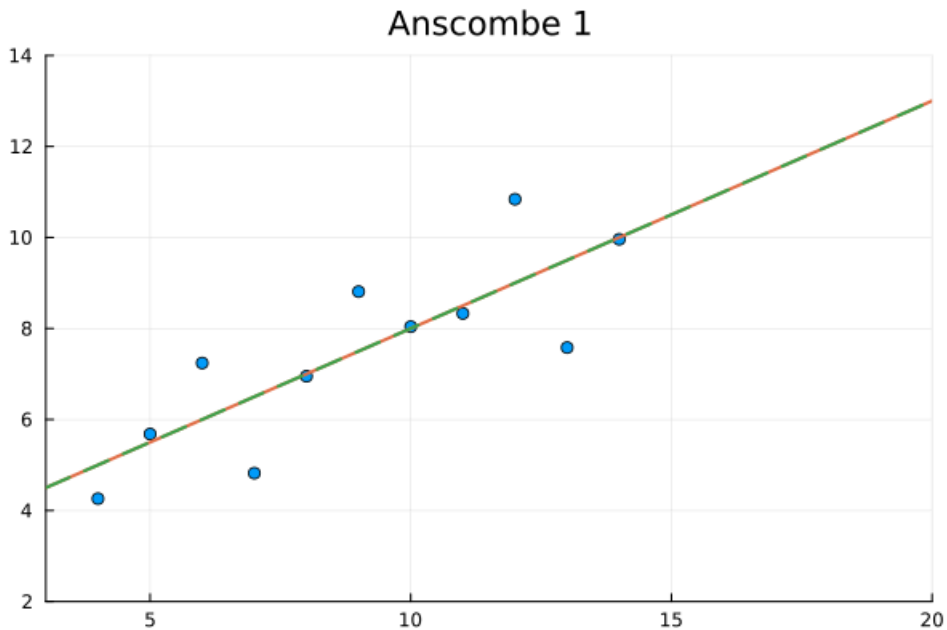
Out[28]:
```
2-element Vector{Float64}:
 3.000090909090909
 0.500090909090909
```

```
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)
```

Out[29]:

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 Datasarus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%20estimat
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, 28.9
744, 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, 71.53
85, 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, 7
3.8462, 75.1282, 76.6667, 77.6923, 79.7436, 81.7949, 83.3333, 85.1282, 86.4103, 87.9487, 89.487
2, 93.3333, 95.3846, 98.2051, 56.6667, 59.2308, 60.7692, 63.0769, 64.1026, 64.359, 74.359, 71.2
821, 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, 57.43
59, 60.0, 64.1026, 66.9231, 71.2821, 74.359, 78.2051, 67.9487, 68.4615, 68.2051, 37.6923, 39.48
72, 91.2821, 50.0, 47.9487, 44.1026]

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

Out[32]:
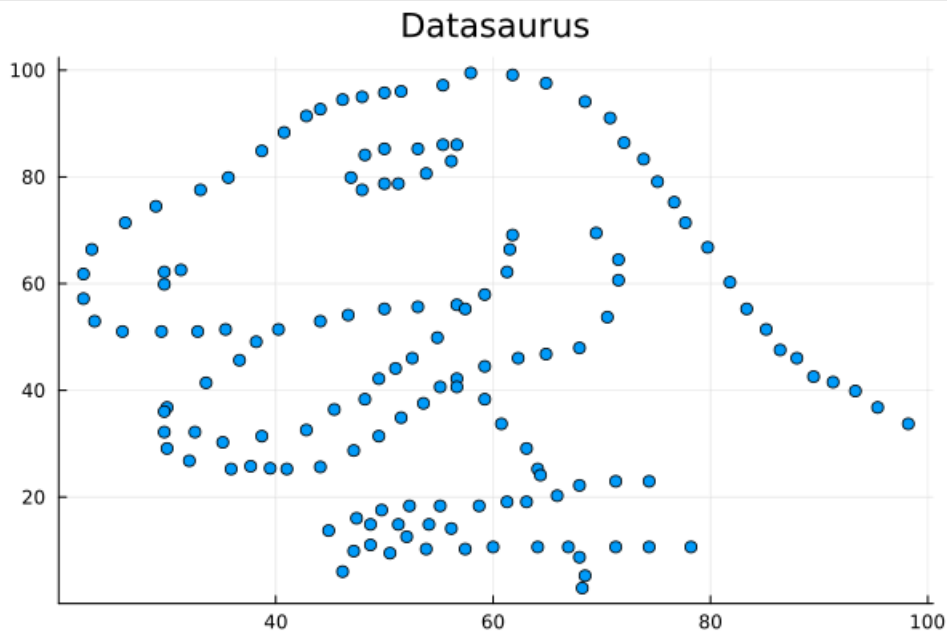
問題: Datasaurusについて検索 (https://www.google.com/search?q=Datasaurus)して見つけた解説を読め.

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 = 10^4
             4  Xs = [rand(dist, n) for _ in 1:L]
```

Out[34]: 10000-element Vector{Vector{Float64}}:
 [12.602711227697185, 7.146112565907991, 9.1271231141078, 3.091612798801891, 6.988897108177396, 3.352685388317398, 5.374660600466008, 12.367330372878694, 4.1990606118724365, 6.144653074287047]
 [6.0810291984432485, 10.731475972720006, 2.869801624490905, 8.71564530541806, 8.024767078660135, 2.4140060205429017, 13.366225713882118, 12.315760649705254, 4.454018568806769, 1.6801473452499321]
 [3.5989572007953, 1.236324775927729, 5.729728369208039, 2.0558473514881332, 11.48758314932965, 5.703479304603708, 10.08991875772663, 1.815090207811051, 6.260131446401576, 8.759209264219134]
 [1.2165606449915964, 8.547179370335039, 2.4686466316306617, 13.393562019992615, 5.4159171344489545, 7.828882536056421, 3.935081665128013, 2.1284090302273735, 5.975921430024259, 4.0069799111028095]
 [4.132120076304036, 0.8184070204224182, 8.186708731671903, 3.214765035463298, 10.195825620373604, 2.96232948993103, 3.2452072955548514, 9.348969667967658, 4.781753549043367, 5.391682040975515]
 [9.436487682122555, 5.66860505945952, 1.1840669014175675, 7.570727274997283, 6.289820196559887, 6.106719429207291, 3.98163803900153, 2.2828840859354305, 4.209773088635767, 4.44823728247811]
 [5.366656341359559, 1.8443117457220648, 3.5715849796442427, 5.451909723052879, 8.96687285038639, 7.856655534620271, 2.495147950812529, 4.563084107751079, 3.036316661897325, 3.412256692871023]
 [5.1565168135739805, 1.7278023929343254, 9.947028705300669, 8.766956664781008, 3.3100959210307597, 20.803066321434947, 7.587892623865767, 10.447698419309997, 2.6768056861319374, 4.792684848280302]
 [2.7548410689758276, 1.6862356954071103, 7.3995597107440965, 4.242379358604053, 9.918394874408447, 4.86040489909051, 1.6783694471163666, 2.1021625449106263, 2.6461854316526807, 2.1226085031211155]
 [7.1061473765575105, 17.725422178904736, 1.1363460630173858, 13.485091267540117, 12.438176397193725, 7.641213233739684, 5.718681360832315, 5.270716136297121, 2.540459782901874, 22.651207648298296]
 [0.414822734413966, 0.4842649944368213, 9.388314247477057, 4.473771976366229, 6.014680545779385, 4.626277089192574, 5.556435263423252, 3.824949171891835, 23.187012923516626, 5.022331745492875]
 [7.1940727972627725, 10.112841227094238, 0.6262731236539947, 2.0218670369576315, 1.0262577099298669, 0.9601151284917709, 5.42212553791611, 4.987558885917323, 13.934174897175192, 4.007985843678283]
 [6.499113291136594, 3.660847700174346, 10.879617691804198, 7.087408500862205, 13.341744635220783, 5.716656246141269, 1.6092745637245873, 7.796023189691585, 4.2605706654553455, 2.665376183957621]
 ⋮
 [1.2228922124075599, 5.996260017555414, 5.380102313292133, 5.170666170581756, 2.2592615720055407, 4.898378540840235, 2.278247940279995, 2.9997575373502516, 6.164744353625599, 1.7750131857888256]
 [5.107079172890312, 1.5638048295918994, 7.827812150301177, 6.676741866306035, 11.344454252966296, 1.4845707196339326, 12.330100550790743, 7.396646992484744, 1.8221685529320415, 3.9456074402979646]
 [26.082672042901063, 2.5725682118462685, 7.939653293991121, 6.0273220547882165, 3.931446865840964, 8.60924062026492, 3.2373901098365647, 13.170410379049166, 4.356373021686507, 9.067415283080791]
 [5.996941272692072, 3.1392730462950573, 8.361325173677141, 3.713018580650785, 3.745389881079382, 4.639292998480688, 1.7045530400358397, 1.7993835269019962, 7.228334609477067, 1.3986383112013543]
 [3.595710140844552, 4.830462103120278, 1.2408312391150258, 5.284470500140458, 7.303437230569725, 3.8408444291293433, 13.227776921250012, 3.3117172821916254, 6.696966024676705, 3.4596380516788287]
 [13.121046182987188, 10.304781325223782, 6.682782005031292, 5.263294365867491, 3.8642420991027286, 5.552028887073756, 5.570624955236236, 4.343702608024147, 8.669101686410277, 10.417632813492455]
 [4.968352191935824, 3.5605066775855043, 15.857800711387029, 8.723697285434127, 1.0656916913514123, 1.9783851692612495, 2.90734947240884, 13.116377597062343, 9.425487933526004, 2.4599524035174385]
 [5.920112165587787, 0.9139892593681438, 3.7472895532404453, 1.6196864664240695, 7.020246575322485, 12.535316025832724, 9.713555069043585, 2.5236811939978487, 4.704766666109526, 13.329929156668076]
 [8.478521319515092, 0.67798983091211, 10.534079343169285, 3.251690353063197, 9.592081196489215, 12.252090287122373, 6.564307467458624, 5.191381463352717, 3.0605840811603526, 1.8182504635475705]
 [1.659635093038043, 3.611675435538645, 21.68710599287342, 0.964015901777645, 4.08001706298832, 13.119526446068159, 6.243117536786096, 0.7583581373881281, 6.058069146856844, 15.369587515614526]
 [1.058145426326134, 9.881857647753446, 5.384739213745467, 10.904438483874442, 4.680398329241913, 6.182428487723981, 15.97942718842679, 7.486368798706872, 9.383181413639202, 2.8934742920492456]
 [11.590395720251879, 8.315547229959142, 9.010087743063378, 1.898505170156119, 2.6040703086065538, 2.687136483277755, 9.861348168184282, 0.06073856985257482, 7.356950590659732, 9.599480618804343]

```
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}:
          7.0394846862513845
          7.065287747791871
          5.673626982751095
          5.491714037393774
          5.227776852770768
          5.117895903981493
          4.656479658811734
          7.52165483966437
          3.941114153403084
          9.571346144528277
          6.299286069199062
          5.029327218807717
          6.351663266816854
          ⋮
          3.8145323843727312
          5.949898652819515
          8.499449188328558
          4.172615044049138
          5.279185392271602
          7.378923692844934
          6.406360113346976
          6.20285721315947
          6.142097580579054
          7.355110826892982
          7.38344592814875
          6.298426060281576
```

```
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}(α=2.0, θ=3.0), n=10
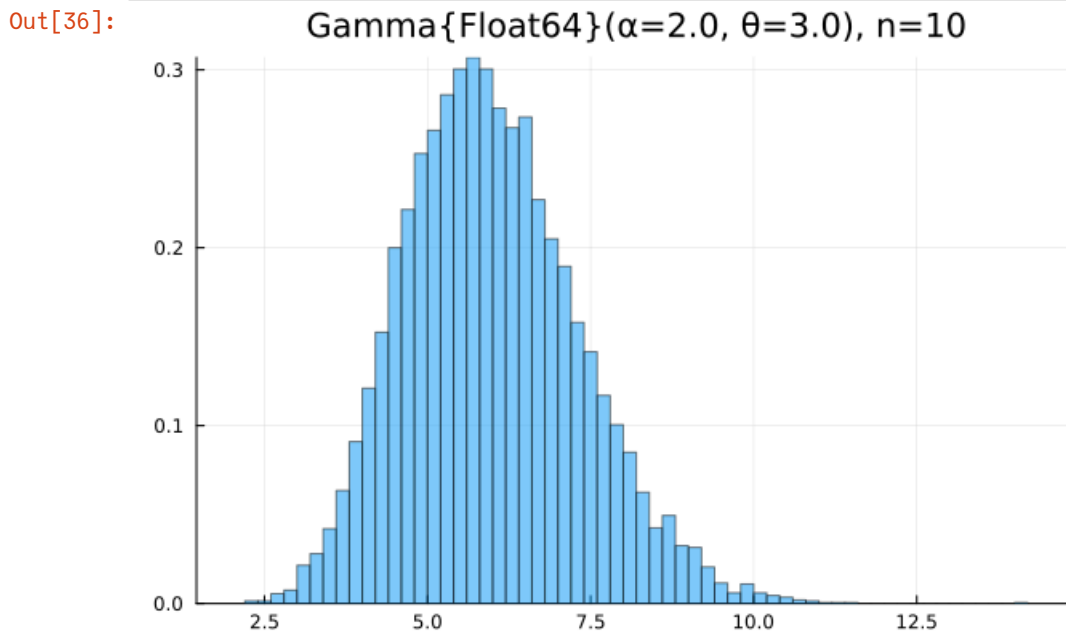
```
In [37]:  ▶  1  # 中心極限定理による正規分布近似を設定
             2  mu = mean(dist)
             3  sigma = std(dist)
             4  normal_approx = Normal(mu, sigma/sqrt(n))
```

Out[37]: Normal{Float64}(μ=6.0, σ=1.3416407864998736)

```
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)
```

Gamma{Float64}(α=2.0, θ=3.0), n=10
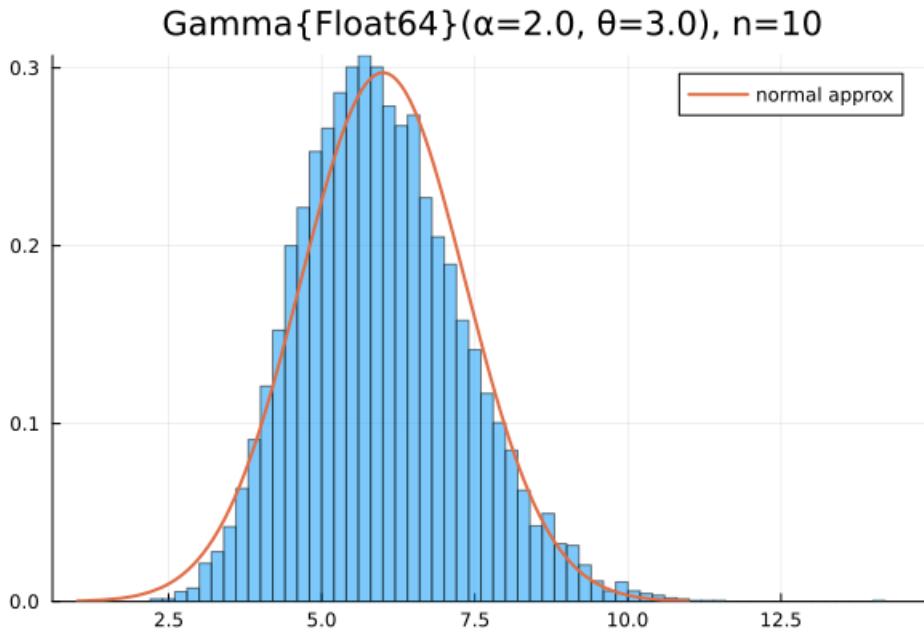
$n = 10$ が小さすぎてずれが大きい.

```
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)
```

Gamma{Float64}(α=2.0, θ=3.0), n=100
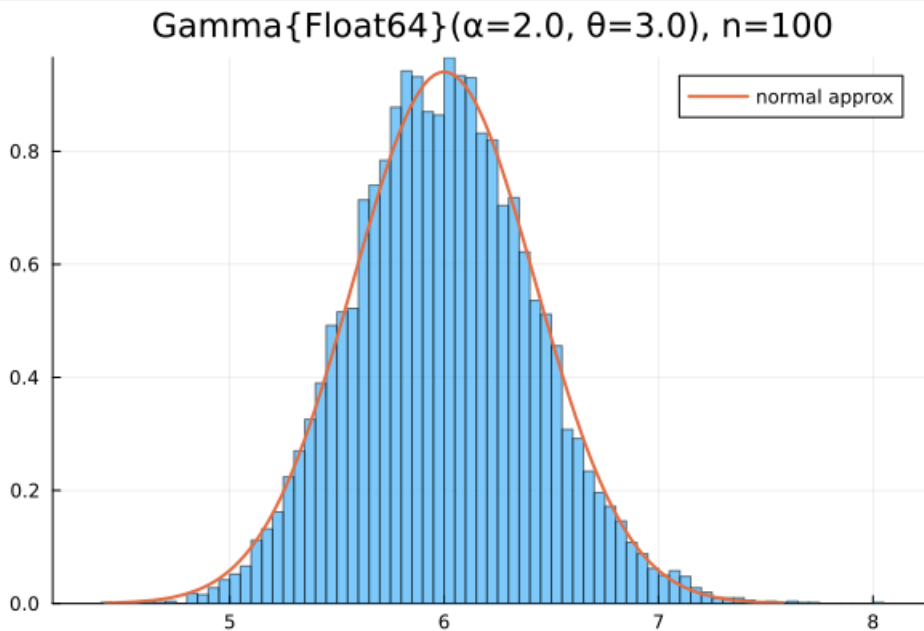
$n = 100$ にしたら, 正規分布とよく一致するようになった.

```
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]:

Gamma{Float64}(α=2.0, θ=3.0), n=100
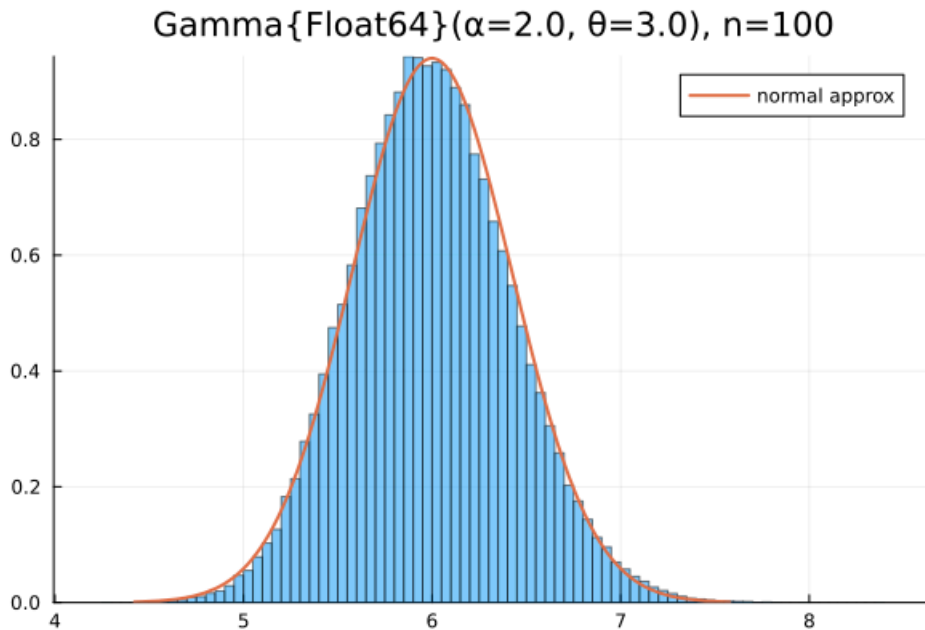
In [41]:

```
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]:

Gamma{Float64}(α=2.0, θ=3.0), n=1000
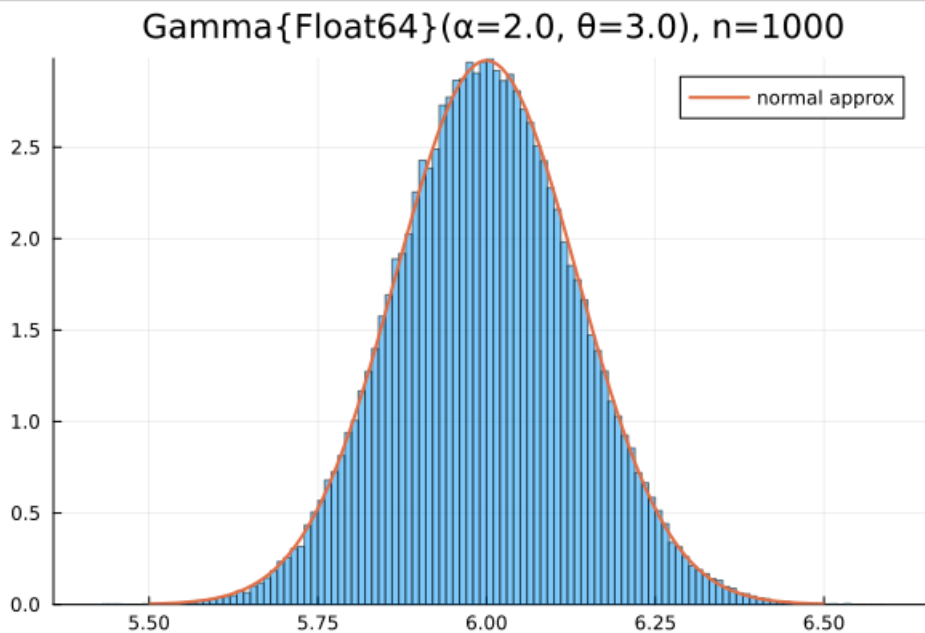
4.2 Revise.jlを使うワークフロー

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.

似た仕事は函数化して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, 函数の定義を julia> プロンプトに直接入力すると, 試行錯誤で函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/timholy/Revise.jl)](https://github.com/timholy/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]:
```julia
using StatsPlots
using Distributions
default(size=(400, 250), titlefontsize=10)

function hello_sine()
    println("Hello, Sine!")
    plot(sin; label="y=sin(x)")
end

function plot_central_limit_theorem(dist, n; L=10^4, bin=:auto)
    distname = mydistname(dist)
    mu = mean(dist)
    sigma = std(dist)
    Xs = [rand(dist, n) for _ in 1:L]
    Xbars = mean.(Xs)
    normal_approx = Normal(mu, sigma/sqrt(n))

    if dist isa DiscreteUnivariateDistribution
        mu = mean(dist)
        sigma = std(dist)
        a = round(n*mu - 4.5sqrt(n)*sigma)
        b = round(n*mu + 4.5sqrt(n)*sigma)
        ran = a-0.5:b+0.5
        bin = ran / n
    end

    histogram(Xbars; bin, norm=true, alpha=0.5, label="Xbars")
    plot!(normal_approx; lw=2, label="normal approx")
    title!("$distname, n=$n")
end

mypdf(dist, x) = pdf(dist, x)
mypdf(dist::DiscreteUnivariateDistribution, x) = pdf(dist, round(Int, x))
mydistname(dist) = replace(string(dist), r"{[^}]*}"⇒"")

function plot_dist(dist; xlim0=nothing)
    distname = mydistname(dist)
    if isnothing(xlim0)
        mu = mean(dist)
        sigma = std(dist)
        a = max(minimum(dist), mu - 4.5sigma)
        b = min(maximum(dist), mu + 4.5sigma)
        if dist isa DiscreteUnivariateDistribution
            a, b = a-1, b+1
        else
            a, b = a-0.025(b-a), b+0.025(b-a)
        end
        xlim0 = (a, b)
    end
    plot(x → mypdf(dist, x), xlim0...; label="", title="$distname")
end

function plot_dist_clt(dist, n; L=10^4, xlim0=nothing)
    P0 = plot_dist(dist; xlim0)
    P1 = plot_central_limit_theorem(dist, n; L)
    plot(P0, P1; size=(800, 250), layout=(1, 2))
end
```

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

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

Hello, Sine!

Out[43]:
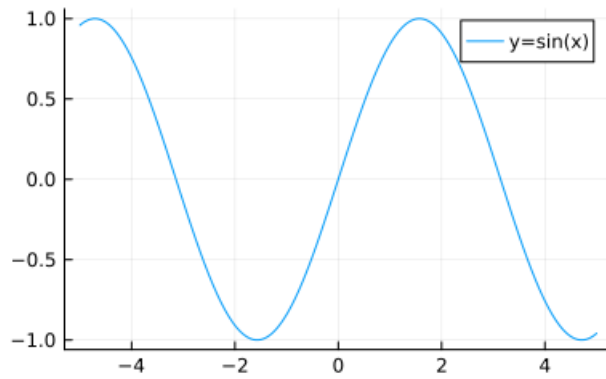

```
In [44]:    ▶    1  plot_dist_clt(Uniform(), 10)
```

Out[44]:
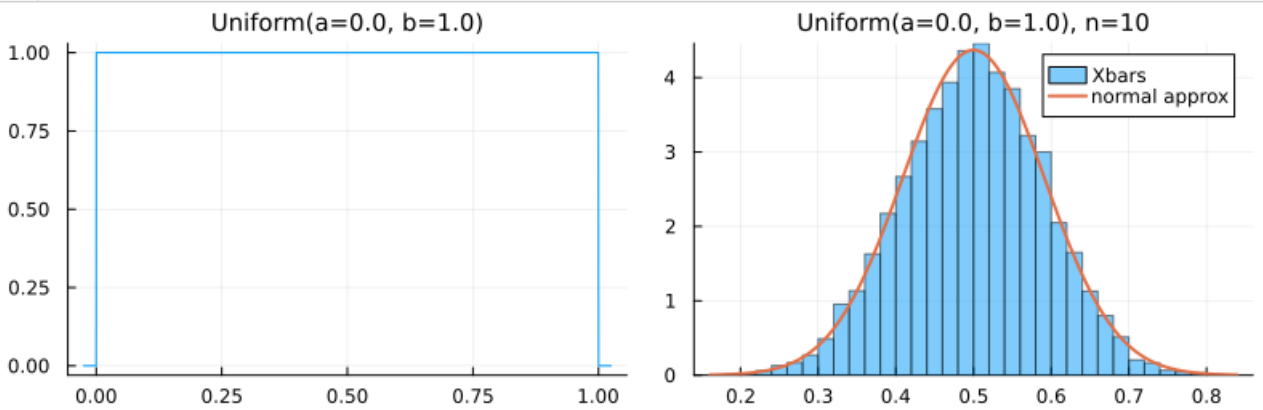

```
In [45]:    ▶    1  plot_dist_clt(Laplace(), 10)
```

Out[45]:
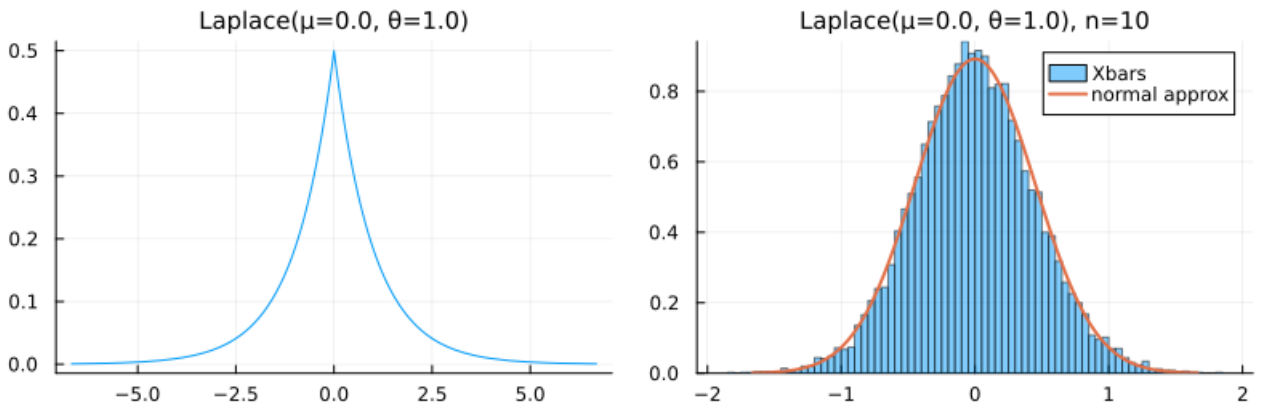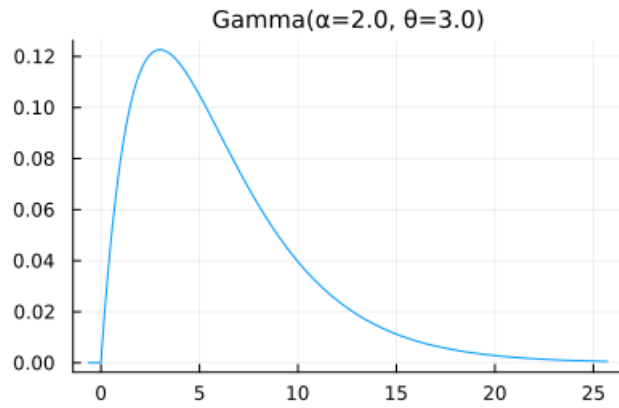


```
In [46]:    ▶    1  plot_dist_clt(Gamma(2, 3), 10)
```
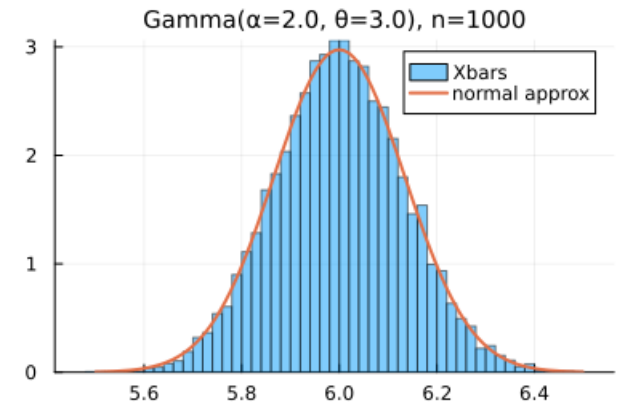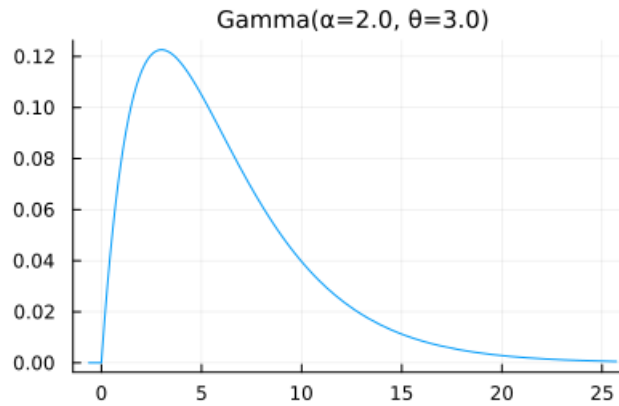
Out[46]:
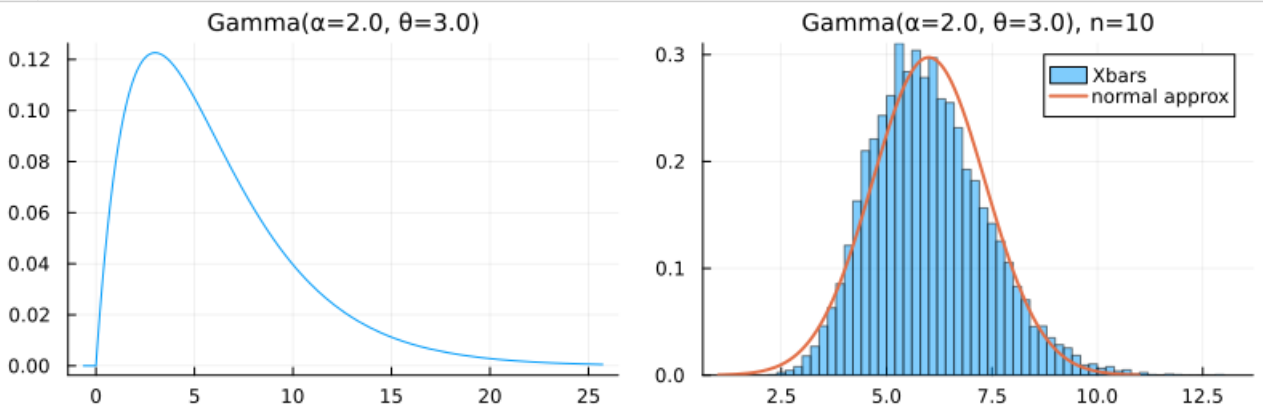
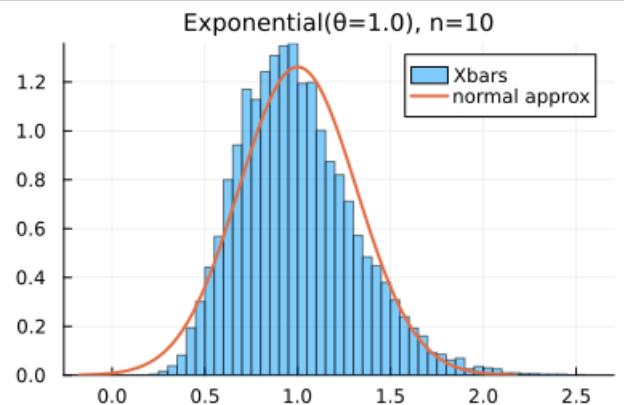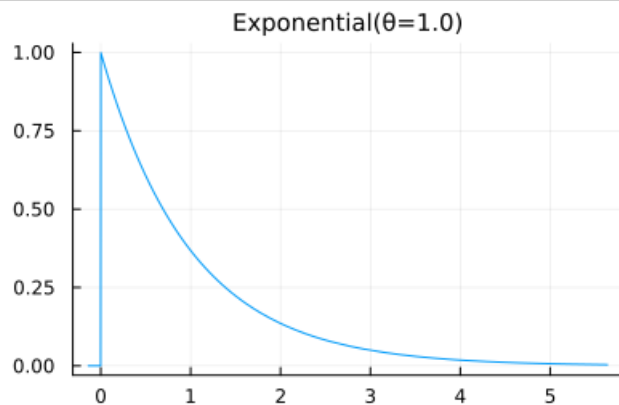▶| `1 plot_dist_clt(Gamma(2, 3), 100)`

Out[47]:
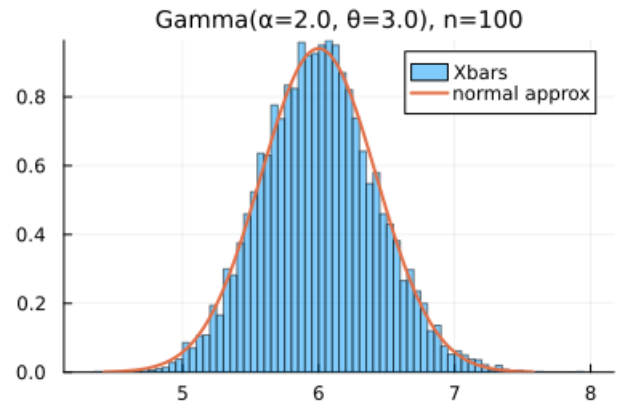
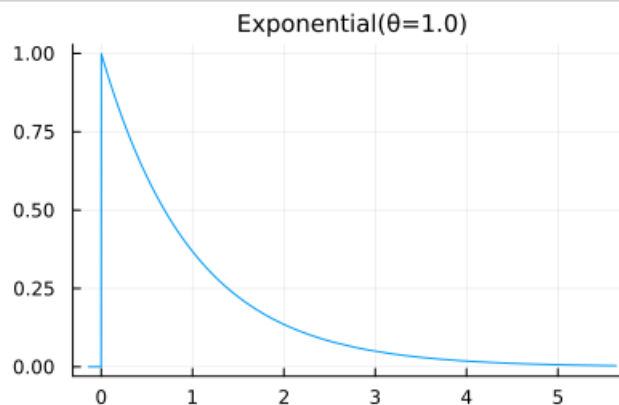In [48]: ▶| `1 plot_dist_clt(Gamma(2, 3), 1000)`

Out[48]:

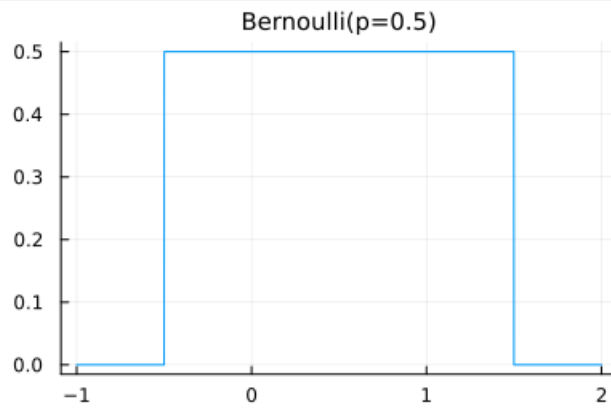
In [49]: ▶| `1 plot_dist_clt(Exponential(), 10)`

Out[49]:

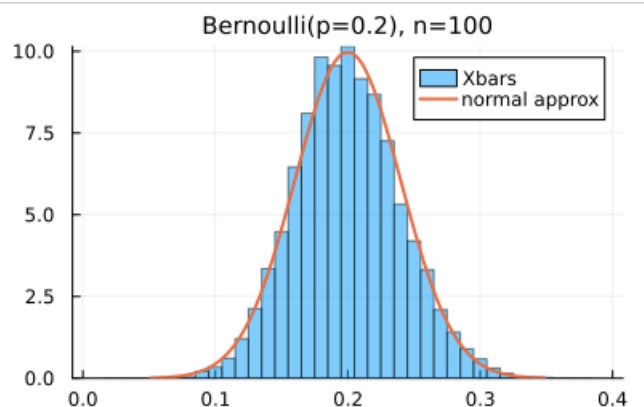In [50]: ▶| `1 plot_dist_clt(Exponential(), 100)`

Out[50]:
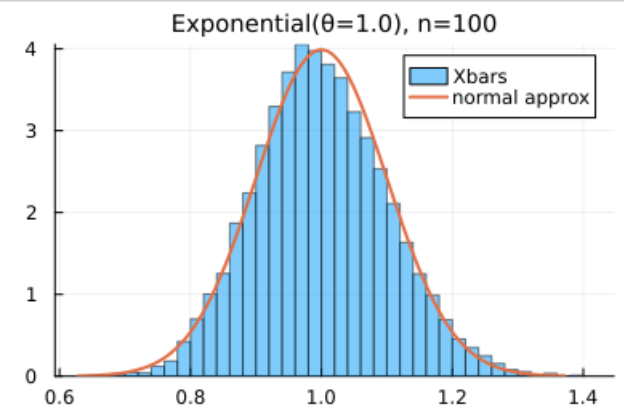
```
In [51]:  ▶  1  plot_dist_clt(Bernoulli(), 10)
```

Out[51]:
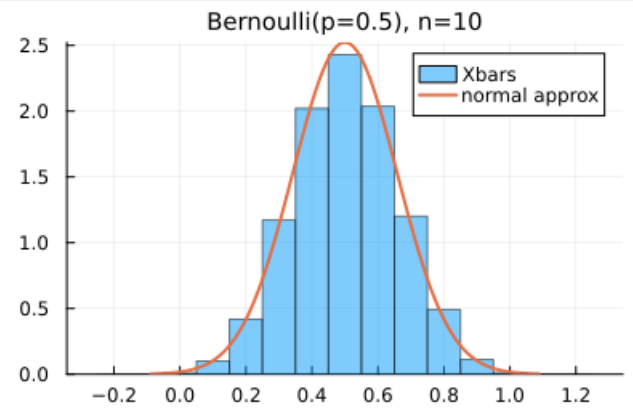

```
In [52]:  ▶  1  plot_dist_clt(Bernoulli(), 100)
```
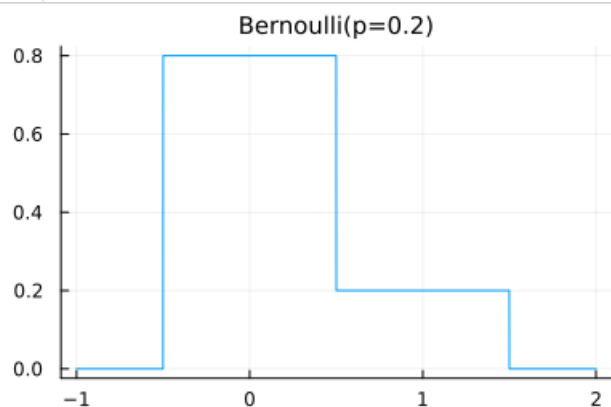
Out[52]:
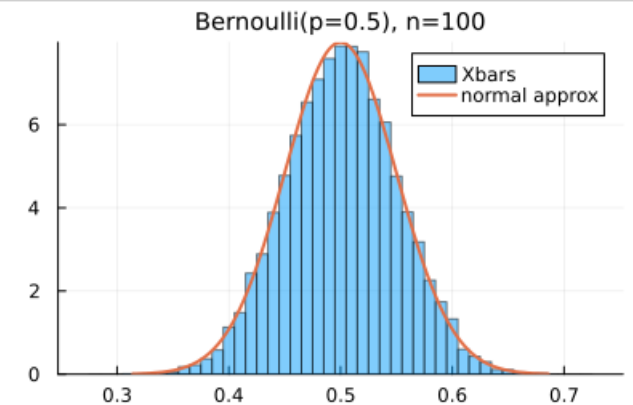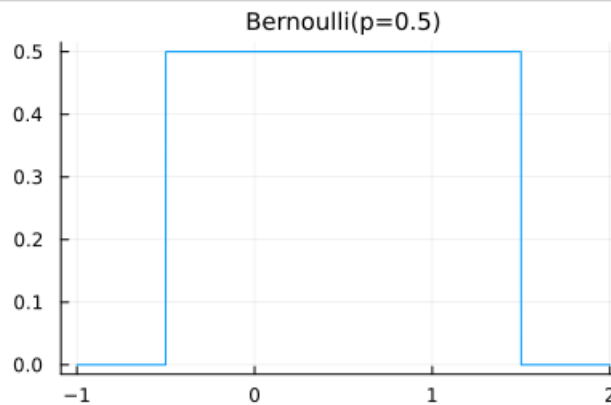

```
In [53]:  ▶  1  plot_dist_clt(Bernoulli(0.2), 10)
```

Out[53]:
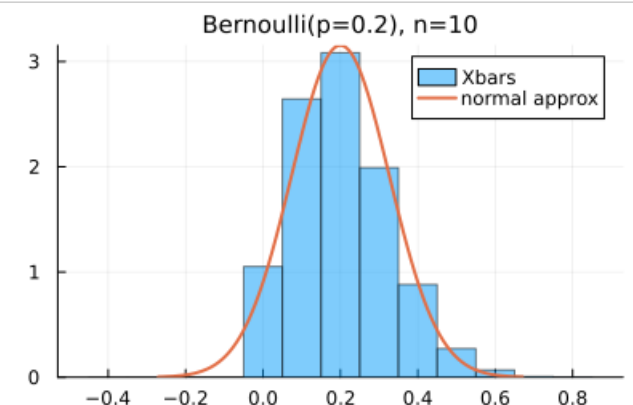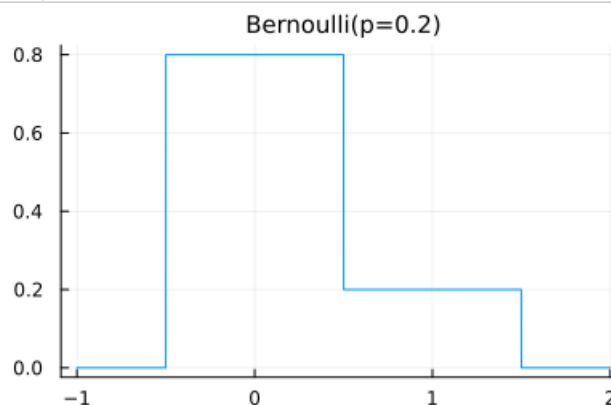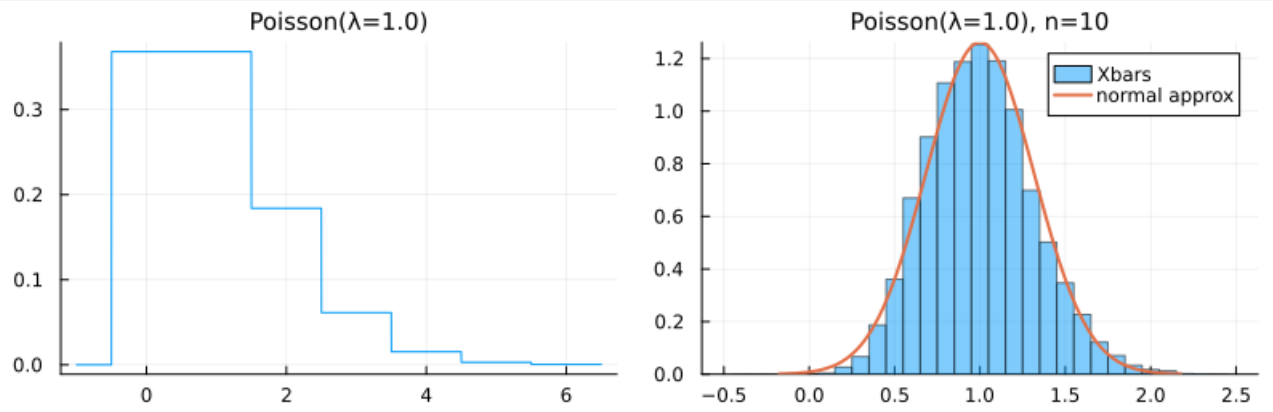


```
In [54]:  ▶  1  plot_dist_clt(Bernoulli(0.2), 100)
```

Out[54]:

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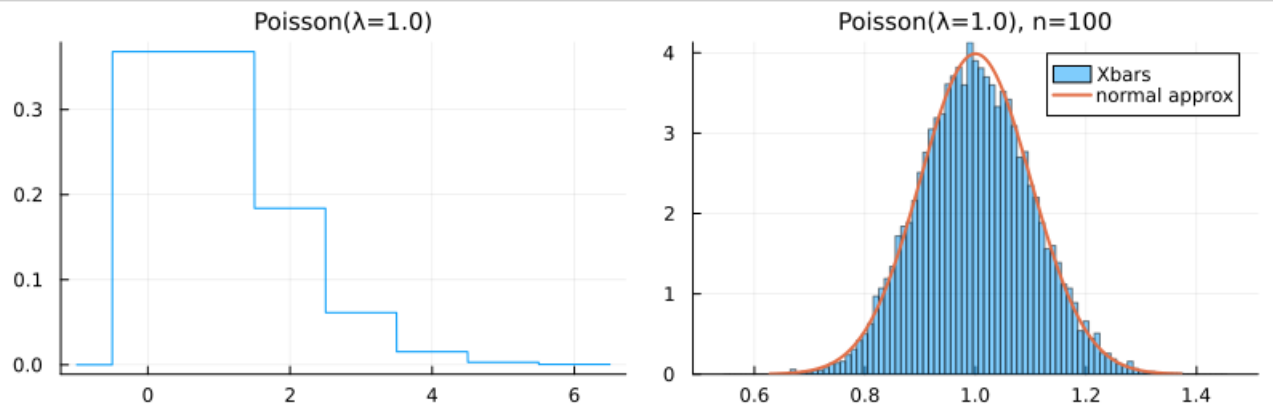
Out[55]:


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

Out[56]:


```
In [ ]:  ▶  1
```