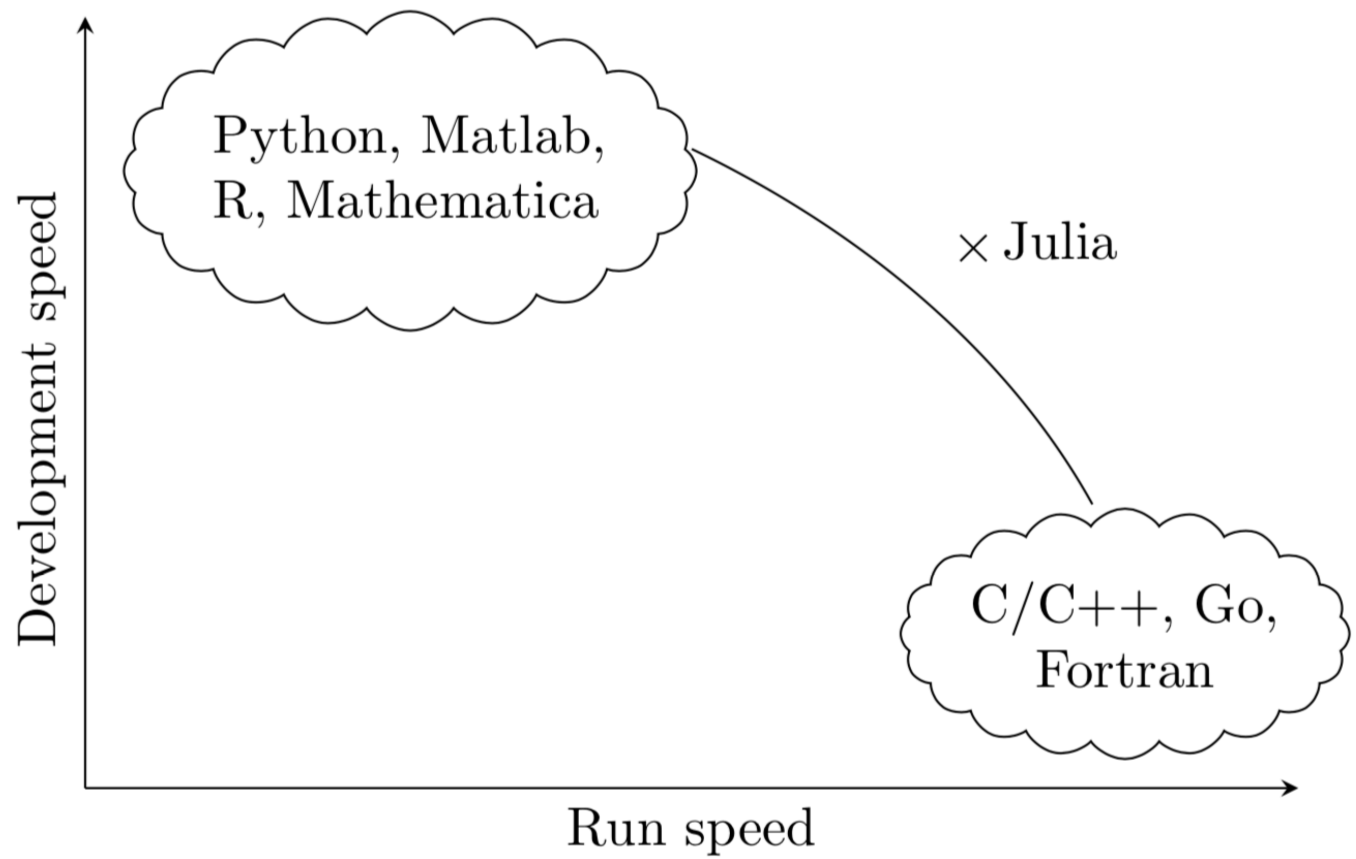
Exploring julia : A Statistical Perspective

Yoni Nazarathy,
The University of Queensland,
joint work with Hayden Klok.


Exploring Julia:
A Statistical Primer.
DRAFT
Hayden Klok, Yoni Nazarathy
June 4, 2018
Springer
Analysis of Engineering & Scientific Data (STAT2201)
Course level
Undergradua
Faculty
Science
School
Probability, Statistics and Scientific Computing (CIVL2530)
Course level
Unde
Facu
Engin
Techn
Current course offerings
Dr Dorival Pedroso
Senior Lecturer
School of Civil Engineering
Faculty of Engineering, Architecture and Information Technology
d.pedroso@uq.edu.au
THE AUSTRALIAN
BUSINESS REVIE
NEWS OPINION BUSINESS REVIEW NATIONAL AFFAIRS SPORT LIFE TECH ARTS TRAVE
TECHNOLOGY
Roames is game to boost STEM interest
KRISHAN SHARMA
The Australian 12:00AM March 8, 2016
Save
Queensland geospatial company, Roames, is looking to tap into gamification as a possible solution to the STEM skills shortage as it looks to ramp up operations in the energy sector.
The Brisbane-based company began life as a business unit of state government-
julia
FOR DATA SCIENCE
Zacharias Voulgaris, PhD
Julia for
Data Science
Packt
EE103/CME103: Introduction to Matrix Methods
Professors Stephen Boyd and David Tse, Stanford University
This is the website for EE103/CME103, Autumn quarter 2017–18. EE103/CME103 will next be taught in Sprir
Osgood.
Software
Julia
Julia files
DID YOU KNOW?
Swift is a general-purpose, multi-pa
language developed by Apple In
Google
The Go Programming Language
GO

A book about "basic statistics" with Julia


Python, Matlab,
R, Mathematica
× Julia
C/C++, Go,
Fortran
Development speed
Run speed

History [edit]

Work on Julia was started in 2009 by Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman who set out to create a language that was both high-level and fast. On 14 February 2012 the team launched[23] a website with a blog post explaining the language's mission. Since then, the Julia community has grown, with over 1,800,000 downloads as of January 2018.[24] It has attracted some high-profile clients, from investment manager BlackRock, which uses it for time-series analytics, to the British insurer Aviva, which uses it for risk calculations. In 2015, the Federal Reserve Bank of New York used Julia to make models of the US economy, noting that the language made model estimation "about 10 times faster" than before (previously used MATLAB). Julia's co-founders established Julia Computing in 2015 to provide paid support, training, and consulting services to clients, though Julia itself remains free to use. At the 2017 JuliaCon[25] conference, Jeff Reiger, Keno Fischer and others announced[26] that the Celeste project[27] used Julia to achieve "peak performance of 1.54 petaFLOPS using 1.3 million threads"[28] on 9300 Knights Landing (KNL) nodes of the Cori supercomputer (the 5th fastest in the world at the time; 8th fastest as of November 2017). Julia thus joins C, C++, and Fortran as high-level languages in which petaFLOPS computations have been written.

The JuliaCon[29] academic conference for Julia users and developers has been held annually since 2014.

JuliaLang / julia

Watch 646 | Star 11,162 | Fork 2,369

Code | Issues 2,057 | Pull requests 516 | Insights

0.7

New issue

⚠ **Past due by 6 months** **99%** complete

An intermediate release feature-equivalent to 1.0 but with deprecations.

2 Open ✓ 502 Closed

WIP: RFC: Create type SecureString ✕ security strings — 39
#24738 opened on 24 Nov 2017 by omus

bump LLVM BB version and use assertion builds on CI ✕ ci — 24
#27182 opened 15 days ago by vchuravy • Approved

Chapter 1:
Introducing Julia

Listing 1.1: Hello world and perfect squares

```
println("There is more than one way to say hello:")

#This is an array consisting of three strings
helloArray = ["Hello","G'day","Shalom"]

for i in 1:3
    println("\t", helloArray[i], " World!")
end

println("\nThese squares are just perfect:")

#This construct is called a 'comprehension'
squares = [i^2 for i in 0:10]

#You can loop on elements of arrays without having to use indexing
for s in squares
    print("  ",s)
end

#The last line of every code snippet is also evaluated as output (in addition to
#        any figures and printing output generated previously).
sqrt(squares)
```

```
There is more than one way to say hello:
        Hello World!
        G'day World!
        Shalom World!

These squares are just perfect:
  0  1  4  9  16  25  36  49  64  81  100
11-element Array{Float64,1}:
  0.0
  1.0
  2.0
  3.0
  4.0
  5.0
  6.0
  7.0
  8.0
  9.0
 10.0
```

When exploring statistics and other forms of numerical computation, it is often useful to use a *comprehension* as a basic programming construct. As explained above, a typical form of a comprehension is,

```
[f(x) for x in aaa]
```

Here aaa is some array (or more generally, a collection of objects). Such a comprehension creates an array of elements, where each element x of aaa is transformed via f(x). Comprehensions are ubiquitous in the code examples we present in this book. We often use them due to their expressiveness and simplicity. We now present a simple additional example:

Listing 1.2: Using a comprehension

```
1  array1 = [(2n+1)^2 for n in 1:5]
2  array2 = [sqrt(i) for i in array1]
3  println(typeof(1:5), "  ", typeof(array1), "  ", typeof(array2))
4  1:5, array1, array2
```

```
UnitRange{Int64}  Array{Int64,1}  Array{Float64,1}
(1:5, [9, 25, 49, 81, 121], [3.0, 5.0, 7.0, 9.0, 11.0])
```

- Line 1 creates an array, named array1, containing the elements of the mathematical set,

  $$\{(2n+1)^2 \ : \ n \in \{1,\dots,5\}\},$$

  in order. However, while mathematical sets are not ordered, arrays generated by Julia comprehensions are ordered. Observe also the literal 2 in the multiplication 2n, without explicit use of the * symbol.
- In line 2, array2 is created.
- In line 3, we print the typeof() three types of expressions. The type of 1:5 (used to create array1) is a UnitRange of Int64. It is a special type of object which encodes "the integers 1,...,5" without explicitly allocating memory for this. Then the type of both array1 and array2 is Array, with Int64 and Float64 as element type, respectively.
- Line 4 creates a tuple by using the comma between array1 and array2. As it is the last line of the of the code it is printed as output. Note in the output that the values of the first array are printed as integers (no decimal point) and the values in the second array are printed as floating point numbers (hence the decimal point).

Julia has a powerful type system which allows for *user defined types*. One can check the type of a variable using the `typeof()` function, while the functions `subtype()` and `supertype()` return the *subtype* and *supertype* of a particular type respectively. As an example `Bool` is a subtype of `Integer`, while `Real` is the supertype of Integer. This is illustrated in figure 1.4, which shows the type hierachy of numbers in Julia.

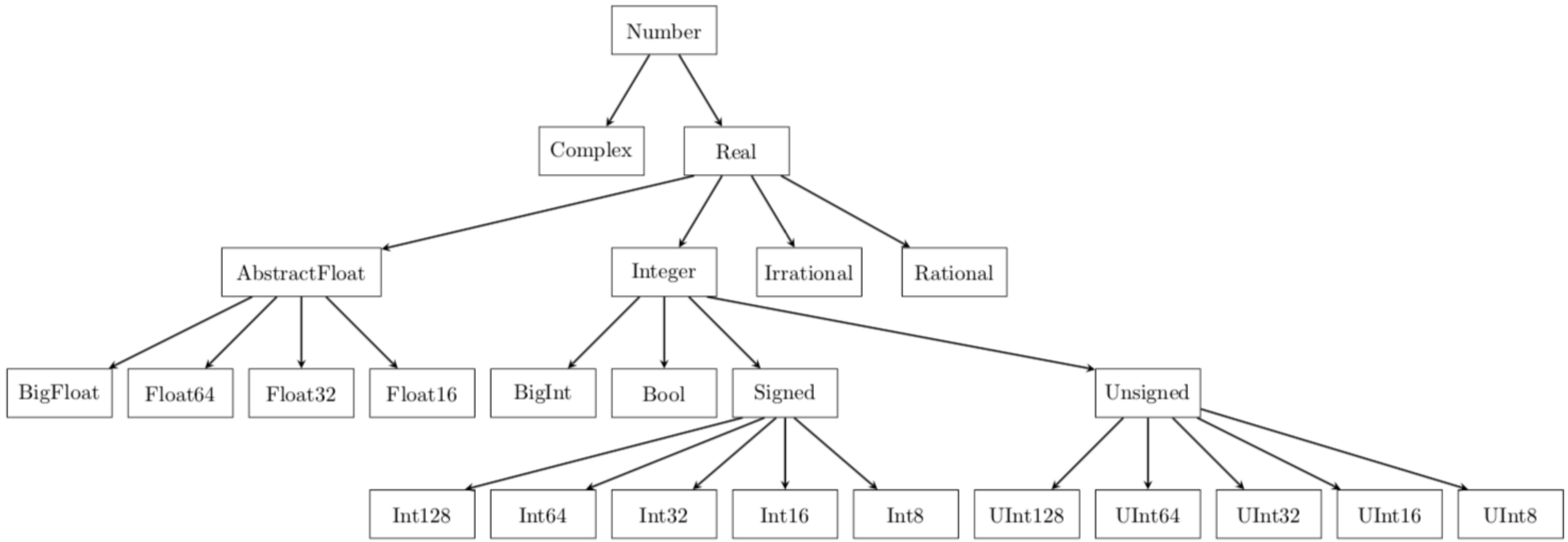


Figure 1.4: Type Hierarchy for Julia Numbers.

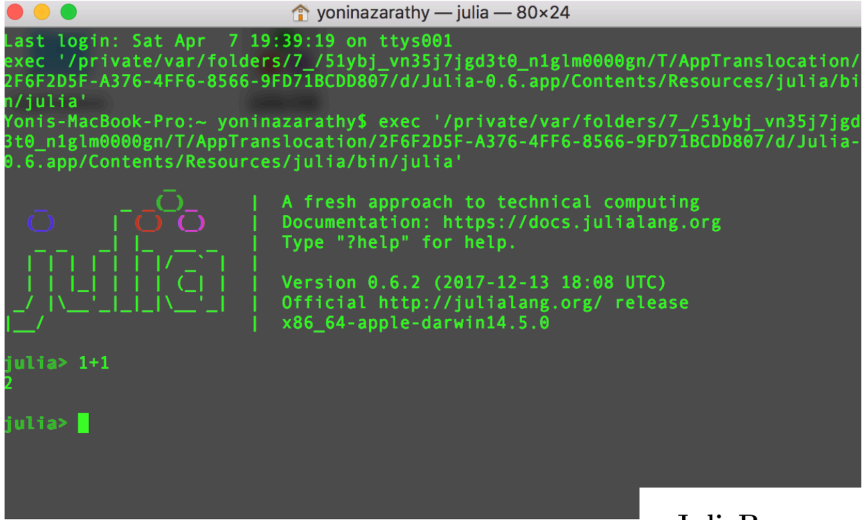
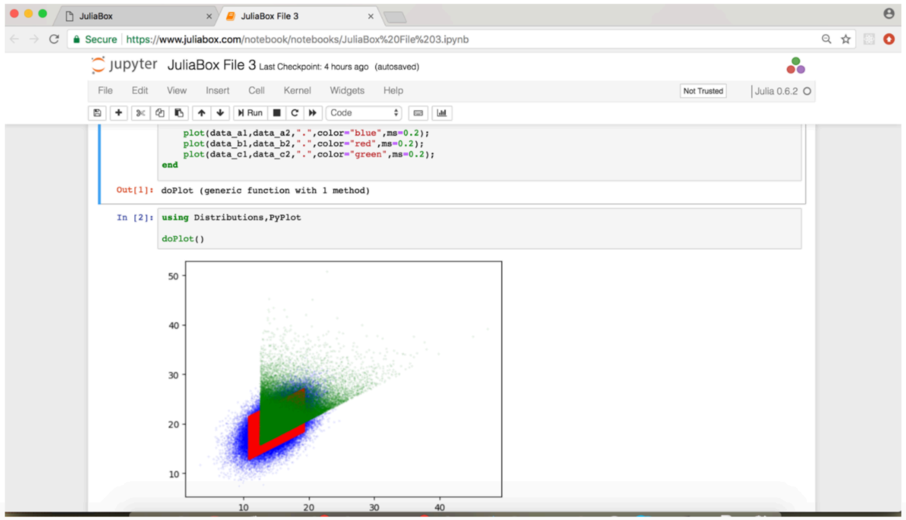


Figure 1.6: The download page for the Julia Kernel a

JuliaBox

An alternative to using the REPL is to use *JuliaBox*, an on-line (currently free) product supplied by Julia computing. At The University of Queensland, we have been using JuliaBox for a large engineering statistics course serving 500 students per semester. It has worked reliability and efficiently. JuliaBox uses *Jupyter notebooks* - this is a an easy to use web-interface that often serves other languages such as Python and R. See Figure 1.7. Juliabox is available at https://juliabox.com/. To use it, one must first sign in using either a LinkedIn, GitHub, or Google account.



Listing 1.7: Steady state of a Markov chain in several ways

```
# Transition probability matrix
P = [0.5 0.4 0.1;
     0.3 0.2 0.5;
     0.5 0.3 0.2]

# First way
P^100
piProb1 = (P^100)[1,:]

# Second way
A = vcat((P' - eye(3))[1:2,:],ones(3)')
b = [0 0 1]'
piProb2 = A\b

# Third way
eigVecs = eigvecs(P')
highestVec = eigVecs[:,findmax(abs(eigvals(P)))[2]]
piProb3 = Array{Float64}(highestVec)/norm(highestVec,1);

# Fourth way
using StatsBase
numInState = zeros(3)
state = 1
N = 10^6
for t in 1:N
    numInState[state] += 1
    state = sample(1:3,weights(P[state,:]))
end
piProb4 = numInState/N


[piProb1 piProb2 piProb3 piProb4]
```

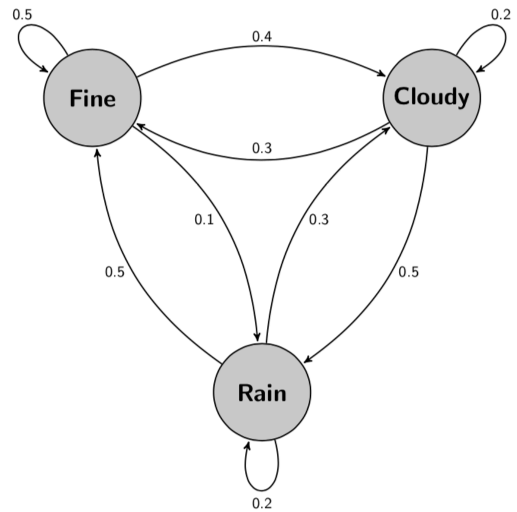


Figure 1.10: Three state Markov chain of the weather. Notice the sum of the arrows leaving each state is 1.

Listing 1.8: An example of a JSON file

```
{
  "words": [ "heaven","hell","man","woman","boy","girl","king","queen",
        "prince","sir","love","hate","knife","english","england","god"],
  "numToShow": 5
}
```

Listing 1.9: Web interface JSON and string parsing

```
using HTTP, JSON

data = HTTP.request("GET", "https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/
                files/t8.shakespeare.txt");

shakespeare = convert(String, data.body)
shakespeareWords = split(shakespeare)

jsonWords = HTTP.request("GET", "https://raw.githubusercontent.com/h-Klok/
                StatsWithJuliaBook/master/1_chapter/jsonCode.json");
parsedJsonDict = JSON.parse( convert(String, jsonWords.body))

keywords = Array{String}(parsedJsonDict["words"])
numberToShow = parsedJsonDict["numToShow"]
wordCount = Dict([(x,count(w -> lowercase(w) == lowercase(x), shakespeareWords))
                  for x in keywords])

sortedWordCount = sort(collect(wordCount),by=last,rev=true)
sortedWordCount[1:numberToShow]
```

```
5-element Array{Pair{String,Int64},1}:
"king"=>1698
"love"=>1279
"man"=>1033
"sir"=>721
"god"=>555
```

Listing 1.11: Histogram of hailstone sequence lengths

```
using PyPlot

function hailLength(n::Int)
    x = 0
    while n != 1
        if n % 2 == 0
            n = Int(n/2)
        else
            n = 3n +1
        end
        x += 1
    end
    return x
end

lengths = [hailLength(n) for n in 2:10^7]

plt[:hist](lengths, 1000, normed="true")
xlabel("Length")
ylabel("Frequency")
```

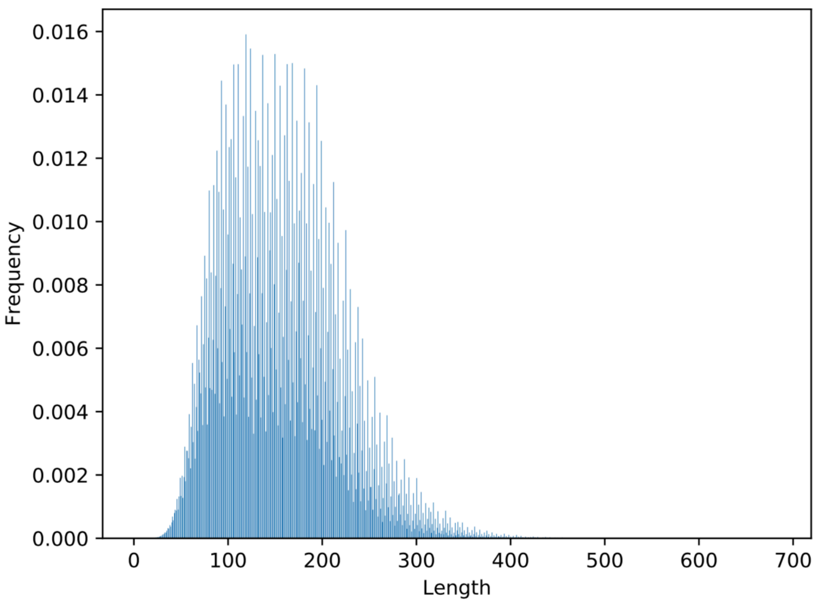


Figure 1.12: Histogram of hailstone sequence lengths.

Listing 1.15: Estimating pi

```
using PyPlot, PyCall
@pyimport matplotlib.patches as patch

srand(1)
N = 10^5
data = [[rand(),rand()] for _ in 1:N]
indata = filter((x)-> (norm(x) <= 1), data)
outdata = filter((x)-> (norm(x) > 1), data)
piApprox = 4*length(indata)/N
println("Pi Estimate: ", piApprox)

fig = figure("Primitives",figsize=(5,5))
plot(first.(indata),last.(indata),".",ms=0.2);
plot(first.(outdata),last.(outdata),".",ms=0.2);
ax = fig[:add_subplot](1,1,1)
ax[:set_aspect]("equal")
r1 = patch.Wedge([0,0],1,0, 90,fc="none",ec="red")
ax[:add_artist](r1)
```

```
Pi Estimate: 3.14348
```

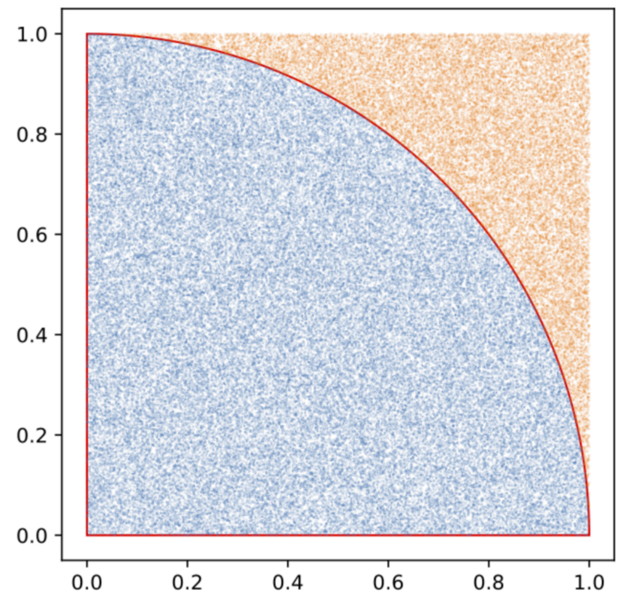


Figure 1.15: Estimating π via Monte-Carlo.

Chapter 2:
Basic Probability

Lattice Paths

We now consider a 5×5 square grid on which an ant walks from the south west corner to the north east corner, taking either a step north or a step east at each grid intersection. From figure 2.3, it is clear there are many possible paths the ant could take. Let us set the sample space to be,

$$\Omega = \text{All possible lattice paths},$$

where the term *Lattice Path* corresponds to a trajectory of the ant going from the south west point, $(0,0)$ to the north east point, (n,n). Since Ω is finite, we can consider the number of elements in it, denoted $|\Omega|$. One question we may ask is what is this number? The answer for a general $n \times n$ grid is,

$$|\Omega| = \binom{2n}{n} = \frac{(2n)!}{(n!)^2}.$$

For example if $n = 5$ then $|\Omega| = 252$. The use of the *binomial coefficient* here is because out of the the $2n$ steps that the ant needs to take (going from $(0,0)$ to (n,n)), n steps need to be north and n need to be east.

Within this context of lattice paths there are a variety of questions. One common question has to do with the event (or set):

$$A = \text{Lattice paths that stay above the diagonal the whole way from}$$

The question of the size of A, namely $|A|$, has interested many people in combinat

$$|A| = \frac{\binom{2n}{n}}{n+1}.$$

Model I - As in the previous examples, assume a symmetric probability space, i.e. each lattice path is equally likely. For this model, obtaining probabilities is a question of counting and the result just follows the combinatorial expressions above:

$$\mathbb{P}_{\text{I}}(A) = \frac{|A|}{|\Omega|} = \frac{1}{n+1}. \tag{2.2}$$

Model II - We assume that at each grid intersection where the ant has an option of where to go ('east' or 'north'), it chooses either east or north, both with equal probabiltiy 1/2. In the case where there is no option for the ant (i.e. it hits the east or north border) then it simply continues along the border to the final destination (n,n). For this model, it is as simple to obtain an expression for $\mathbb{P}(A)$. One way to do it is by considering a *recurrence relation* for the probabilities (sometimes known as *first step analysis*). We omit the details and present the result:

$$\mathbb{P}_{\text{II}}(A) = \frac{|A|}{|\Omega|} = \frac{\binom{2n-1}{n}}{2^{2n-1}}.$$

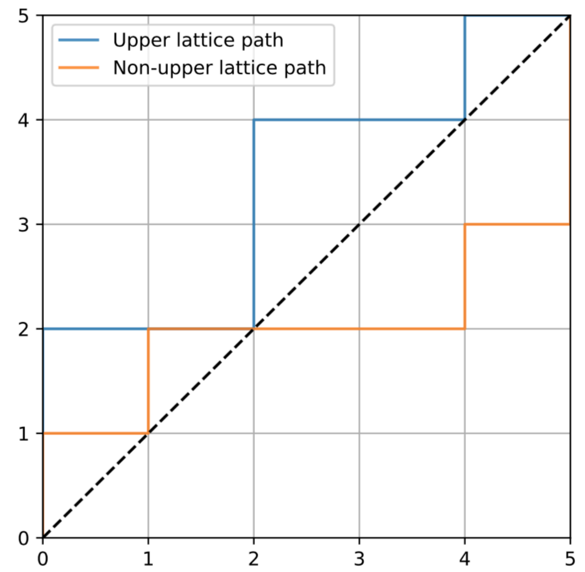


Figure 2.3: Example of two different Catalan Paths.

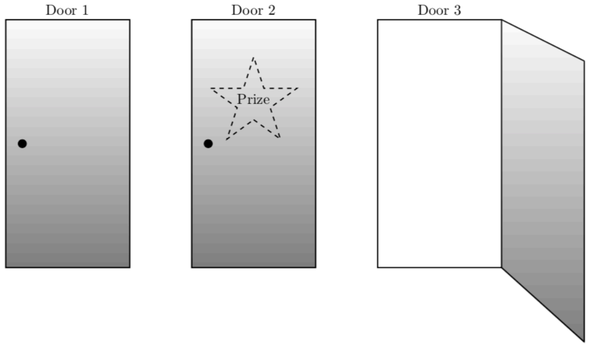


Figure 2.5: If the prize is behind Door 2 and Door 1 is chosen, the GSH must reveal door 3.

$$\mathbb{P}(A_1 \mid B_2) = \frac{\mathbb{P}(B_2 \mid A_1)\mathbb{P}(A_1)}{\mathbb{P}(B_2)} = \frac{\frac{1}{2} \times \frac{1}{3}}{\frac{1}{2}} = \frac{1}{3}, \qquad \text{(Policy I)}$$

$$\mathbb{P}(A_3 \mid B_2) = \frac{\mathbb{P}(B_2 \mid A_3)\mathbb{P}(A_3)}{\mathbb{P}(B_2)} = \frac{1 \times \frac{1}{3}}{\frac{1}{2}} = \frac{2}{3}. \qquad \text{(Policy II)}$$

Listing 2.13: The Monty Hall problem

```
function montyHall(switchPolicy)
    prize = rand(1:3)
    choice = rand(1:3)

    if prize == choice
        revealed = rand(setdiff(1:3,choice))
    else
        revealed = rand(setdiff(1:3,[prize,choice]))
    end

    if switchPolicy
        choice = setdiff(1:3,[revealed,choice])[1]
    end

    return choice == prize
end

N = 10^6
sum([montyHall(true) for _ in 1:N])/N,
sum([montyHall(false) for _ in 1:N])/N
```

```
(0.666778, 0.33387)
```

Chapter 3:
Probability Distributions

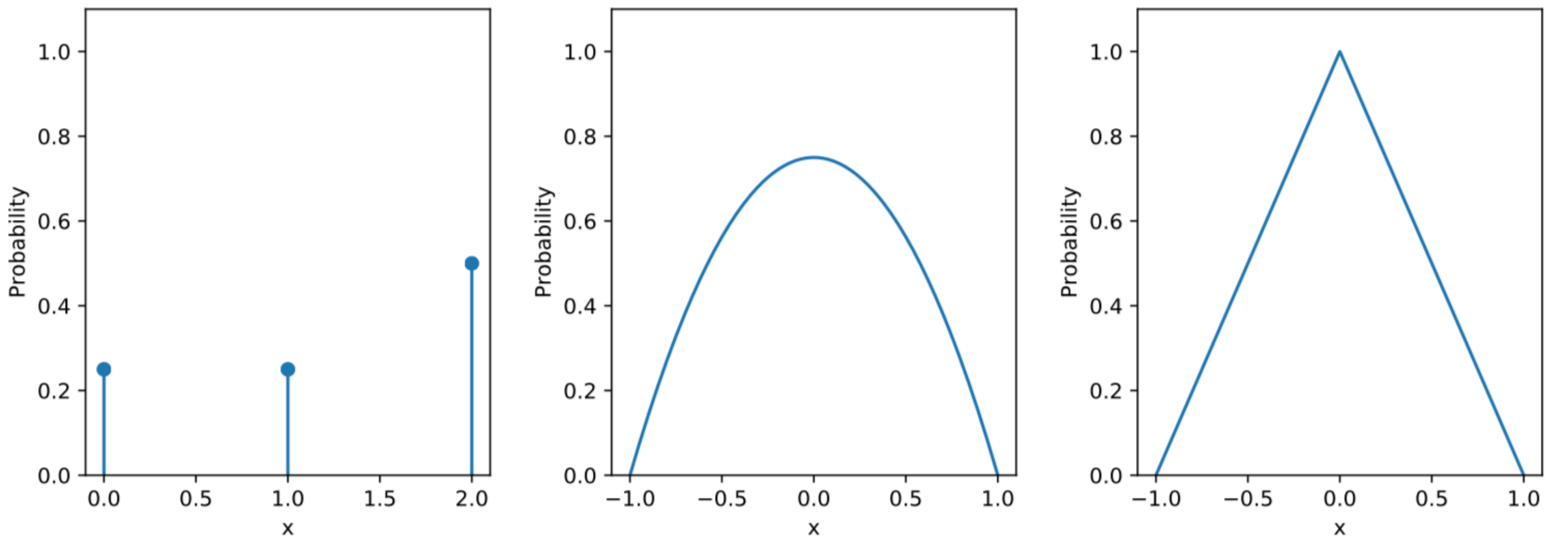


Figure 3.2: Three different examples of probability distributions.

Listing 3.3: Expectation via numerical integration

```
using QuadGK

sup = (-1,1)
f1(x) = 3/4*(1-x^2)
f2(x) = x < 0 ? x+1 : 1-x

expect(f,support) = quadgk((x) -> x*f(x),support[1],support[2])[1]

expect(f1,sup),expect(f2,sup)
```

```
(0.0, -2.0816681711721685e-17)
```

Listing 3.11: Using `rand` with `Distributions`

```
using Distributions, StatsBase

dist1 = TriangularDist(0,10,5)
dist2 = DiscreteUniform(1,5)
theorMean1, theorMean2 = mean(dist1), mean(dist2)

N=10^6
data1 = rand(dist1,N)
data2 = rand(dist2,N)
estMean1, estMean2 = mean(data1), mean(data2)

println("Symmetric Triangular Distiribution on [0,10] has mean $theorMean1
        (estimated: $estMean1)")
println("Discrete Uniform Distiribution on {1,2,3,4,5} has mean $theorMean2
        (estimated: $estMean2)")
```

```
Symmetric Triangular Distiribution on [0,10] has mean 5.0 (estimated: 4.998652531225146)
Discrete Uniform Distiribution on {1,2,3,4,5} has mean 3.0 (estimated: 2.998199)
```

Listing 3.12: Inverse transform sampling

```
using Distributions, PyPlot

triangDist = TriangularDist(0,2,1)
xGrid = 0:0.1:2
N = 10^6
inverseSampledData = quantile.(triangDist,rand(N))

plt[:hist](inverseSampledData,50, normed = true,ec="k",
        label="Inverse transform\n sampled data")
plot(xGrid,pdf(triangDist,xGrid),"r",label="PDF")
legend(loc="upper right")
```

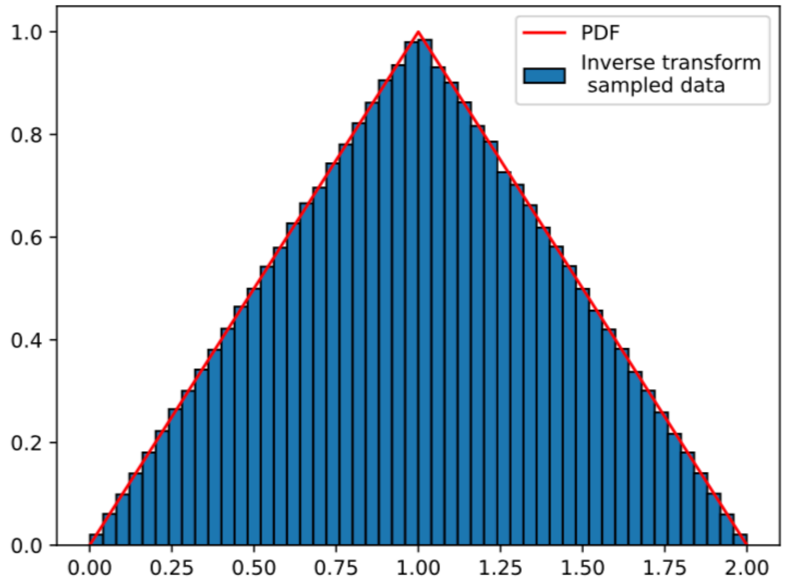


Figure 3.8: Histogram of data generated using inverse transform sampling.

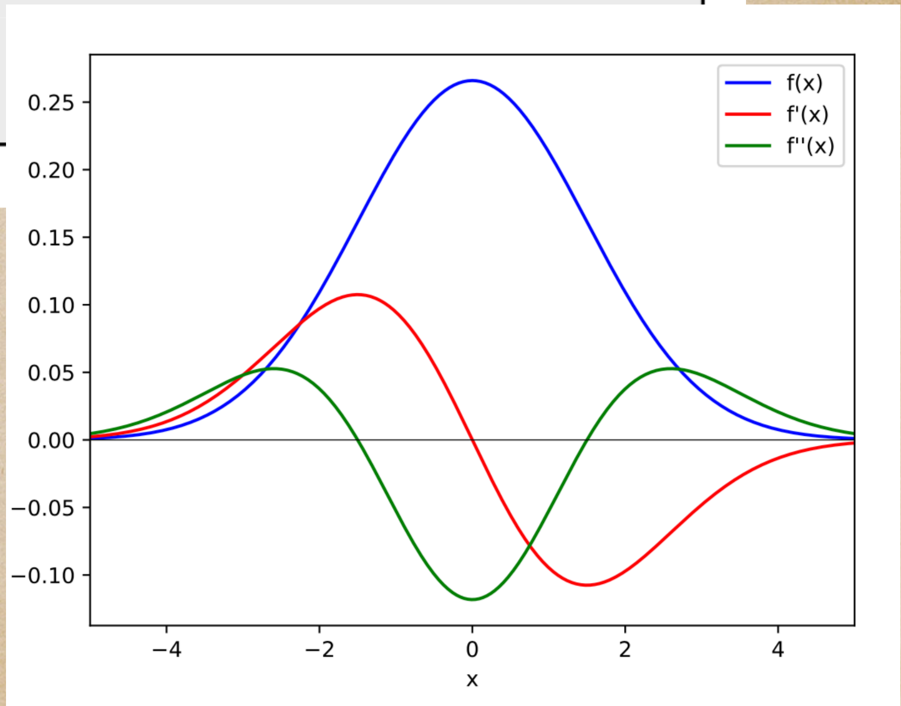
Listing 3.20: Families of continuous distributions

```julia
using Distributions
dists = [
    Uniform(10,20),
    Exponential(3.5),
    Gamma(0.5,7),
    Beta(10,0.5),
    Weibull(10,0.5),
    Normal(20,3.5),
    Rayleigh(2.4),
    Cauchy(20,3.5)];

println("Distribution \t\t\t\t\t\t Parameters \t Support")
reshape([dists ;  params.(dists) ;
                ((d)->(minimum(d),maximum(d))).(dists) ],
                length(dists),3)
```

```
Distribution                                         Parameters      Support
8  3 Array{Any,2}:
 Distributions.Uniform{Float64}(a=10.0, b=20.0)   (10.0, 20.0)  (10.0, 20.0)
 Distributions.Exponential{Float64}(  =3.5)        (3.5,)         (0.0, Inf)
 Distributions.Gamma{Float64}(  =0.5,   =7.0)       (0.5, 7.0)     (0.0, Inf)
 Distributions.Beta{Float64}(  =10.0,   =0.5)       (10.0, 0.5)    (0.0, 1.0)
 Distributions.Weibull{Float64}(  =10.0,   =0.5)    (10.0, 0.5)    (0.0, Inf)
 Distributions.Normal{Float64}(  =20.0,   =3.5)     (20.0, 3.5)    (-Inf, Inf)
 Distributions.Rayleigh{Float64}(  =2.4)           (2.4,)         (0.0, Inf)
 Distributions.Cauchy{Float64}(  =20.0,   =3.5)     (20.0, 3.5)    (-Inf, Inf)
```

Listing 3.27: Numerical derivatives of the normal density

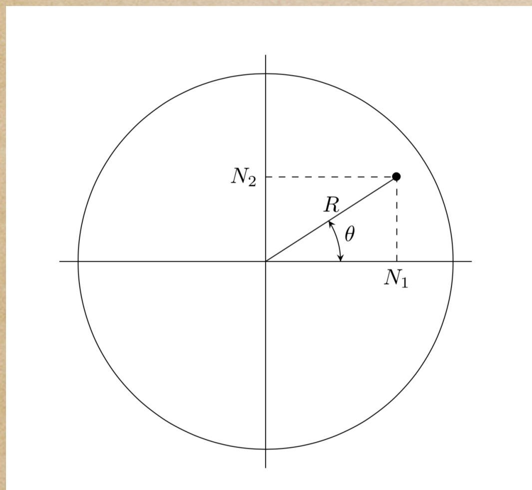
```
using Distributions, Calculus ,PyPlot

xGrid = -5:0.01:5
sig = 1.5
normalDensity(z) = pdf(Normal(0,sig),z)

d0 = normalDensity.(xGrid)
d1 = derivative.(normalDensity,xGrid)
d2 = second_derivative.(normalDensity, xGrid)

ax = gca()
plot(xGrid,d0,"b",label="f(x)")
plot(xGrid,d1,"r",label="f'(x)")
plot(xGrid,d2,"g",label="f''(x)")
plot([-5,5],[0,0],"k", lw=0.5)
xlabel("x")
xlim(-5,5)
legend(loc="upper right")
```



Listing 3.28: Alternative representations of Rayleigh random variables

```
using Distributions

N = 10^6
sig = 1.7

data1 = sqrt.(-(2* sig^2)*log.(rand(N)))

distG = Normal(0,sig)
data2 = sqrt.(rand(distG,N).^2 + rand(distG,N).^2)

distR = Rayleigh(sig)
data3 = rand(distR,N)

mean.([data1, data2, data3])
```

```
3-element Array{Float64,1}:
 2.12994
 2.12935
 2.13188
```



for $y \geq 0$,

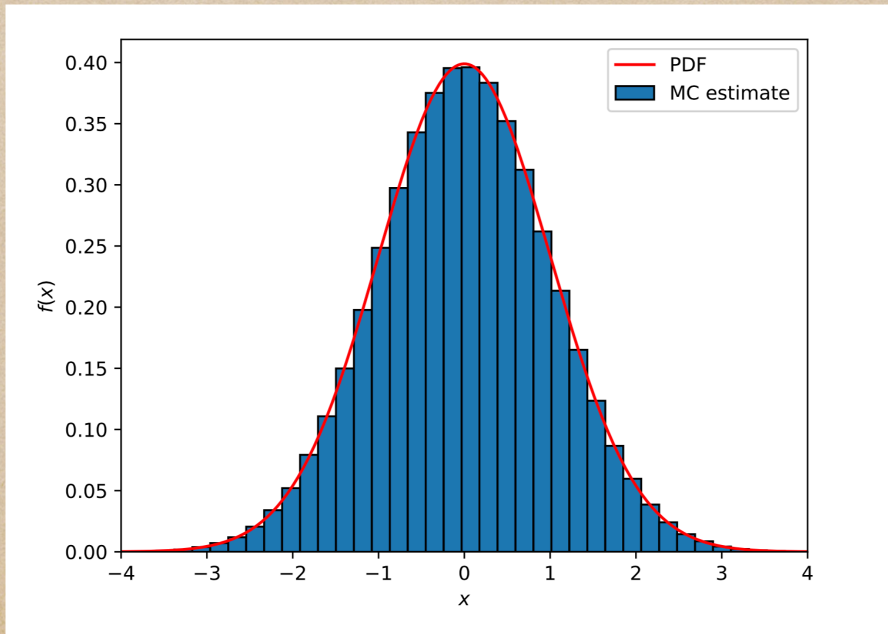
$$F_R(y) = \mathbb{P}(\sqrt{X} \leq y) = \mathbb{P}(X \leq y^2) = F_X(y^2) = 1 - \exp\Big(-\frac{y^2}{2\sigma^2}\Big),$$

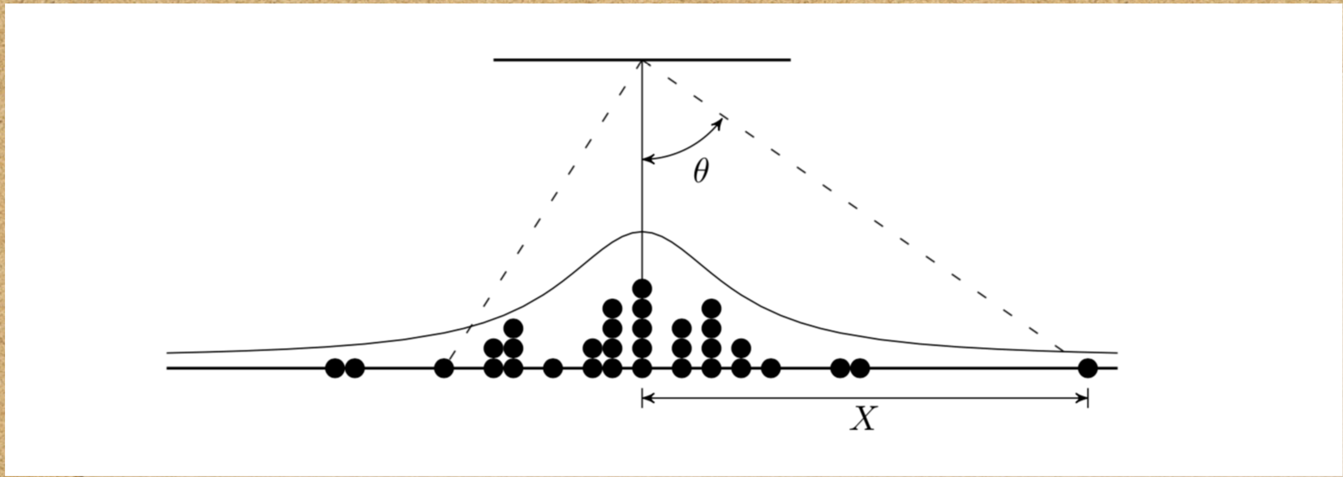
g, we get the density,

$$f_R(y) = \frac{y}{\sigma^2} \exp\Big(-\frac{y^2}{2\sigma^2}\Big).$$

Listing 3.29: The Box-Muller transform

```
using Distributions, PyPlot
srand(1)

X() = sqrt(-2*log(rand()))*cos(2*pi*rand())
xGrid = -4:0.01:4

plt[:hist]([X() for _ in 1:10^6],50,
    normed = true,ec="k",label="MC estimate")
plot(xGrid,pdf(Normal(),xGrid),"-r",label="PDF")
xlim(-4,4)
xlabel(L"$x$")
ylabel(L"f(x)")
legend(loc="upper right")
```





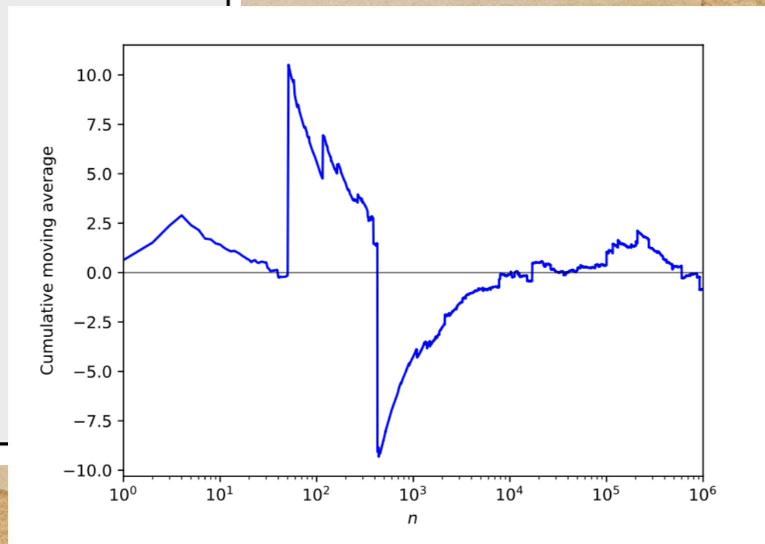
$$F_X(x) = \mathbb{P}(X \leqslant x) = \mathbb{P}\big(\tan(\theta) \leqslant x\big) = \mathbb{P}\big(\theta \leqslant \text{atan}(x)\big) = F_\theta\big(\text{atan}(x)\big) = \begin{cases} 0 & x < -\pi/2, \\ \frac{1}{\pi}\text{atan}(x) & x \in [-\pi/2, \pi/2], \\ 1 & \pi/2 < x. \end{cases}$$

The density is the obtained by taking the derivative, which evaluates to,

$$f(x) = \frac{1}{\pi(1+x^2)}.$$

Listing 3.30: The law of large numbers breaks down with very heavy tails

```
using PyPlot

srand(4)
n = 10^6

data = tan(rand(n)*pi - pi/2)
averages = accumulate(+,data)./collect(1:n)

plot(1:n,averages,"b")
plot([1,n],[0,0],"k",lw=0.5)
xscale("log")
xlim(0,n)
xlabel(L"$n$")
ylabel("Rolling \naverage",rotation=0,labelpad=20)
```



Listing 3.32: Generating random vectors with desired mean and covariance

```
using Distributions,PyPlot

SigY = [ 6 4 ; 4 9]
muY = [15 ; 20]
bruteCholFact(S) = Array(cholfact(S)[:L])
A = bruteCholFact(SigY)

N = 10^5

dist_a = Normal()
rvX_a() = [rand(dist_a) ; rand(dist_a)]
rvY_a() = A*rvX_a() + muY
data_a = [rvY_a() for _ in 1:N]
data_a1 = first.(data_a)
data_a2 = last.(data_a)

dist_b = Uniform(-sqrt(3),sqrt(3))
rvX_b() = [rand(dist_b) ; rand(dist_b)]
rvY_b() = A*rvX_b() + muY
data_b = [rvY_b() for _ in 1:N]
data_b1 = first.(data_b)
data_b2 = last.(data_b)

dist_c = Exponential()
rvX_c() = [rand(dist_c) - 1; rand(dist_c) - 1]
rvY_c() = A*rvX_c() + muY
data_c = [rvY_c() for _ in 1:N]
data_c1 = first.(data_c)
data_c2 = last.(data_c)

plot(data_a1,data_a2,".",color="blue",ms=0.2);
plot(data_b1,data_b2,".",color="red",ms=0.2);
plot(data_c1,data_c2,".",color="green",ms=0.2);
```

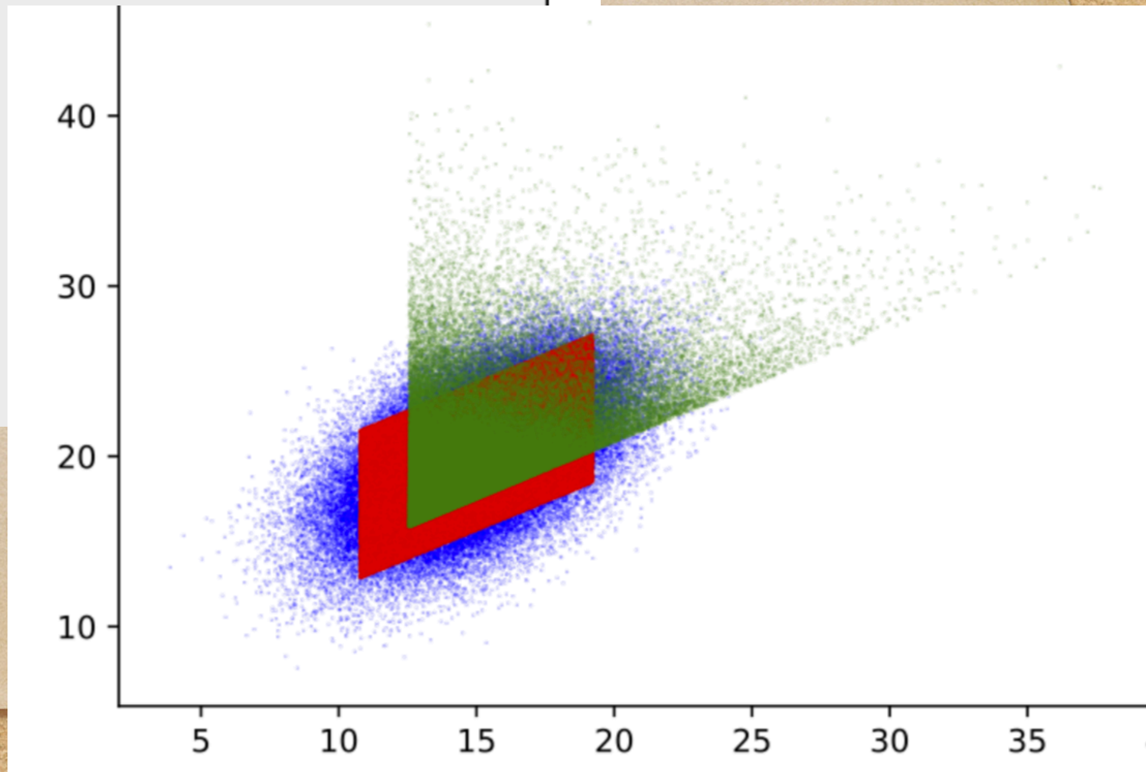
$$\Sigma_{\mathbf{Y}} = AA'.$$

as the desired $\mu_{\mathbf{Y}}$ and $\Sigma_{\mathbf{Y}}$.

find a matrix A that satisfies (3.19). For this the *Cholesky de*

assume we wish to generate a random vector $\mathbf{Y}$ with,

$$\mu_{\mathbf{Y}} = \begin{bmatrix} 15 \\ 20 \end{bmatrix} \quad \text{and} \quad \Sigma_{\mathbf{Y}} = \begin{bmatrix} 9 & 4 \\ 4 & 16 \end{bmatrix}.$$



$$f(\mathbf{x}) = (2\pi)^{-n/2} e^{-\frac{1}{2}\mathbf{x}'\mathbf{x}}.$$

The example below illustrates numerically that this is a valid pdf for $n = 2$ via numerical integration.

Listing 3.33: Multidimensional integration

```
using HCubature
M = 4
f(x) = (2*pi)^(-length(x)) * exp(-(1/2)*x'x)
hcubature(f,[-M,M],[-M,M])
```

Chapter 4:
Processing and Summarising Data

Listing 4.1: Creating a `DataFrame`

```
using DataFrames

purchaseData = readtable("purchaseData.csv");
```

Listing 4.2: Overview of a `DataFrame`

```
include("dataframeCreation.jl")
println(head(purchaseData))
println(showcols(purchaseData))
```

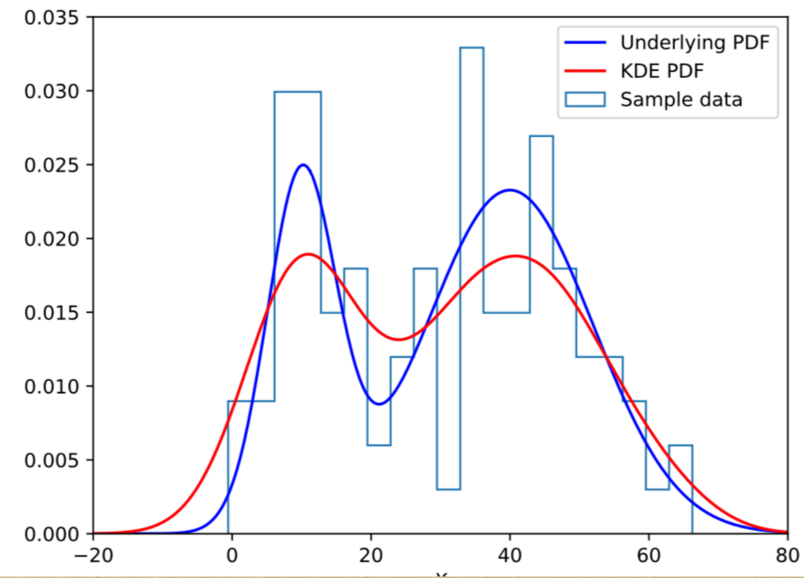
```
| Row | me       | Date       | Time     | Type    | Price   |
----------------------------------------------------------------
| 1   | MARYAN   | 14/09/2008 | 12:21 AM | E       | 8403    |
| 2   | REBECCA  | 11/03/2008 | 8:56 AM  | missing | 6712    |
| 3   | ASHELY   | 5/08/2008  | 9:12 PM  | E       | 7700    |
| 4   | KHADIJAH | 2/09/2008  | 10:35 AM | A       | missing |
| 5   | TANJA    | 1/12/2008  | 12:30 AM | B       | 19859   |
| 6   | JUDIE    | 17/05/2008 | 12:39 AM | E       | 8033    |

| Col # | Name  | Eltype                 | Missing | Values                 |
------------------------------------------------------------------------------
| 1     | Name  | Union{Missing, String} | 13      | MARYAN...RIVA          |
| 2     | Date  | Union{Missing, String} | 0       | 14/09/2008...30/12/2008 |
| 3     | Time  | Union{Missing, String} | 5       | 12:21 AM...5:48 AM     |
| 4     | Type  | Union{Missing, String} | 10      | E...B                  |
| 5     | Price | Union{Int64, Missing}  | 14      | 8403...15432           |
```

Listing 4.3: Referencing data in a `DataFrame`

```
include("dataframeCreation.jl")
println(purchaseData[13:17,[:Name]])
println(purchaseData[:Name][13:17])
purchaseData[ismissing.(purchaseData[:Time]), :]
```

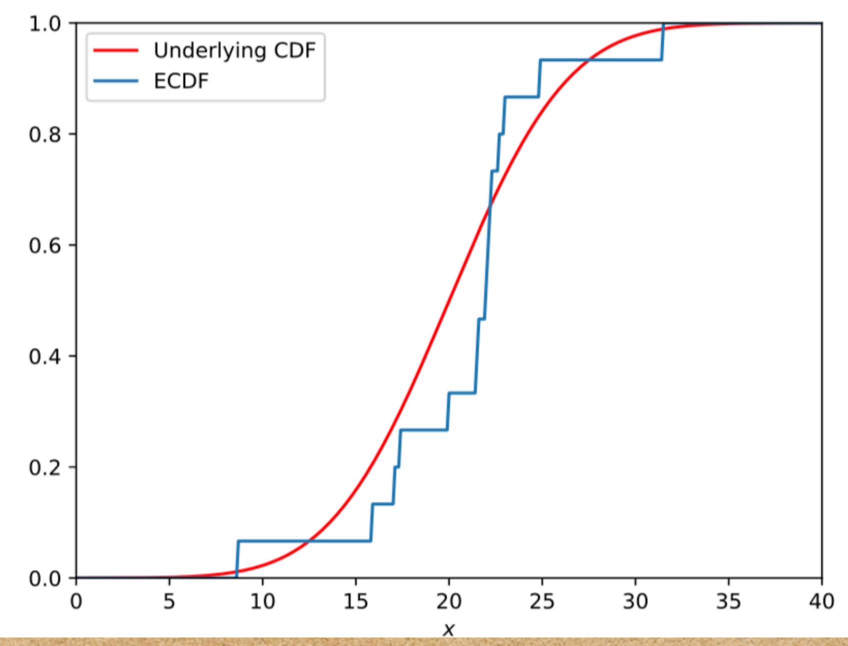
Listing 4.11: Kernel density estimation

```julia
using Distributions, KernelDensity, PyPlot
srand(1)

mu1, sigma1 = 10, 5
mu2, sigma2 = 40, 12

z1 = Normal(mu1,sigma1)
z2 = Normal(mu2,sigma2)

p = 0.3

function mixRv()
    (rand() <= p) ? rand(z1) : rand(z2)
end

function actualPDF(x)
    p*pdf(z1,x) + (1-p)*pdf(z2,x)
end

numSamples = 100
data = [mixRv() for _ in 1:numSamples]

xGrid = -20:0.1:80
pdfActual = actualPDF.(xGrid)
kdeDist = kde(data)
pdfKDE = pdf(kdeDist,xGrid)

plt[:hist](data,20, histtype = "step", normed=true, label="Sample data")
plot(xGrid,pdfActual,"-b",label="Underlying PDF")
plot(xGrid,pdfKDE, "-r",label="KDE PDF")
xlim(-20,80)
ylim(0,0.035)
xlabel(L"X")
legend(loc="upper right")
```



Listing 4.13: Empirical cumulative distribution function

```
using Distributions, StatsBase,PyPlot

srand(1)
underlyingDist = Normal(20,5)
data = rand(underlyingDist, 15)

empiricalDF = ecdf(data)

xGrid = 0:0.1:40
plot(xGrid,cdf(underlyingDist,xGrid),"-r",label="Underlying CDF")
plot(xGrid,empiricalDF(xGrid),label="ECDF")
xlim(0,40)
ylim(0,1)
xlabel(L"x")
legend(loc="upper left")
```



Chapter 5: Statistical Inference Ideas

Listing 5.3: Are the sample mean and variance independent?

```
using Distributions,PyPlot

function statPair(dist,n)
    sample = rand(dist,n)
    [mean(sample),var(sample)]
end

stdUni = Uniform(-sqrt(3),sqrt(3))

n, N = 2, 10^5
dataUni = [statPair(stdUni,n) for _ in 1:N]
dataUniInd = [[mean(rand(stdUni,n)),var(rand(stdUni,n))] for _ in 1:N]
dataNorm = [statPair(Normal(),n) for _ in 1:N]
dataNormInd = [[mean(rand(Normal(),n)),var(rand(Normal(),n))] for _ in 1:N]

figure("test", figsize=(10,5))
subplot(121)
plot(first.(dataUni),last.(dataUni),".b",ms="0.1",label="Same group")
plot(first.(dataUniInd),last.(dataUniInd),".r",ms="0.1", label="Separate group")
xlabel(L"$\overline{X}$")
ylabel(L"$S^2$")
legend(markerscale=60,loc="upper right")
ylim(0,10)

subplot(122)
plot(first.(dataNorm),last.(dataNorm),".b",ms=
plot(first.(dataNormInd),last.(dataNormInd),".
xlabel(L"$\overline{X}$")
ylabel(L"$S^2$")
legend(markerscale=60,loc="upper right")
ylim(0,10)
savefig("sampleStatInd.png")
```

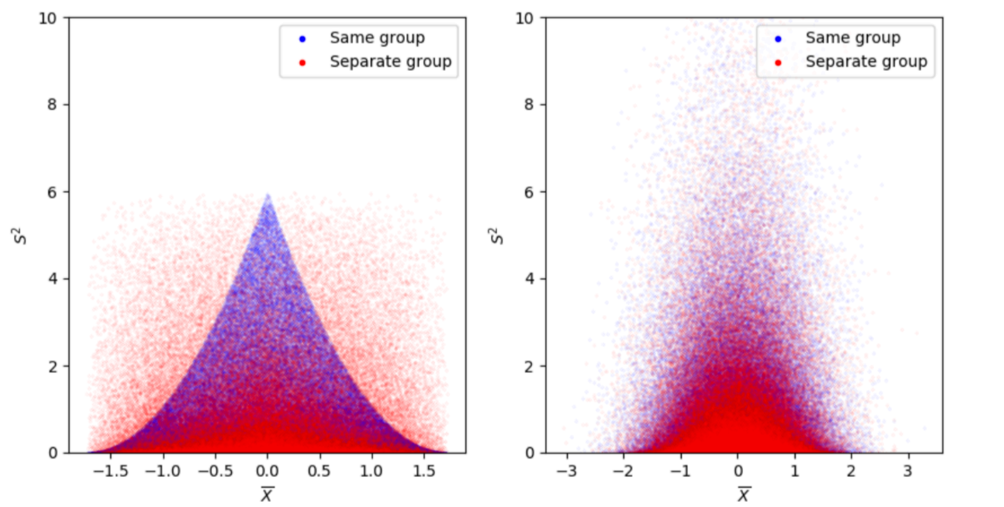


Figure 5.4: Pairs of $\overline{X}$ and S^2 for standard uniform (left) and standard normal (right). Blue points indicate the statistics were calculated from the same sample, red indicates the statistics were calcualted from separate sample groups.

Listing 5.8: Point estimation via method of moments using a numerical solver

```julia
using Distributions, NLsolve
srand(1)

a, b, c = 3, 5, 4
dist = TriangularDist(a,b,c)
n = 2000
samples = rand(dist,n)

m_k(k,data) = 1/n*sum(data.^k)
mHats = [m_k(i,samples) for i in 1:3]

function equations(F, x)
    F[1] = 1/3*( x[1] + x[2] + x[3] ) - mHats[1]
    F[2] = 1/6*( x[1]^2 + x[2]^2 + x[3]^2 + x[1]*x[2] + x[1]*x[3] +
                x[2]*x[3] ) - mHats[2]
    F[3] = 1/10*( x[1]^3 + x[2]^3 + x[3]^3 + x[1]^2*x[2] + x[1]^2*x[3] +
                x[2]^2*x[1] + x[2]^2*x[3] + x[3]^2*x[1] + x[3]^2*x[2] +
                x[1]*x[2]*x[3] ) - mHats[3]
end

nlOutput = nlsolve(equations, [ 0.1; 0.1; 0.1])
println("Found estimates for (a,b,c) = ", nlOutput.zero)
println(nlOutput)
```

$$\hat{m}_1 = \frac{1}{3}(a+b+c),$$
$$\hat{m}_2 = \frac{1}{6}(a^2+b^2+c^2+ab+ac+bc),$$
$$\hat{m}_3 = \frac{1}{10}(a^3+b^3+c^3+a^2b+a^2c+b^2a+b^2c+c^2a+c^2b+abc).$$

```
Found estimates for (a,b,c) = [3.98312, 3.01452, 5.00224]
Results of Nonlinear Solver Algorithm
 * Algorithm: Trust-region with dogleg and autoscaling
 * Starting Point: [0.1, 0.1, 0.1]
 * Zero: [3.98312, 3.01452, 5.00224]
 * Inf-norm of residuals: 0.000000
 * Iterations: 17
 * Convergence: true
   * |x - x'| < 0.0e+00: false
   * |f(x)| < 1.0e-08: true
 * Function Calls (f): 18
 * Jacobian Calls (df/dx): 14
```

$$f(x) = \frac{\lambda^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\lambda x}$$

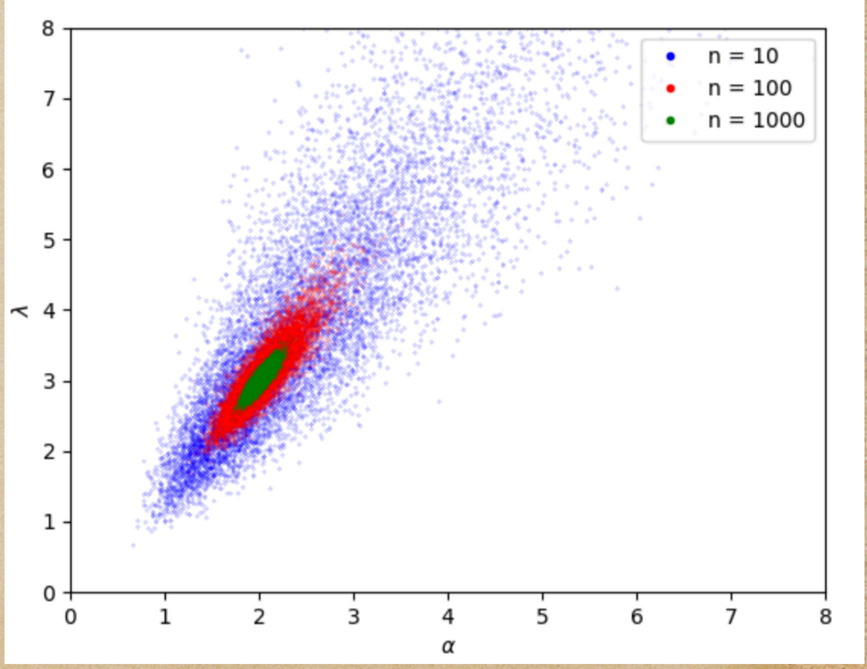
$$L(\theta\ ;\ x_1,\dots,x_n) = f_{X_1,\dots,X_n}(x_1,\dots,x_n\ ;\ \theta) = \prod_{i=1}^{n} f(x_i\ ;\ \theta).$$

where $\psi(z) := \frac{d}{dz}\log(\Gamma(z))$ is the well known *digamma function*. Hence we find that α^* must satisfy:

$$\log(\alpha) - \psi(\alpha) - \log(\overline{x}) + \overline{x}_\ell = 0. \tag{5.14}$$

Listing 5.10: MLE of a gamma distributions parameters

```
using SpecialFunctions, Distributions, PyPlot, Roots

eq(alpha, xb, xbl) = log.(alpha) - digamma.(alpha) - log(xb) + xbl

actualAlpha, actualLambda = 2, 3
gammaDist = Gamma(actualAlpha,1/actualLambda)

function mle(sample)
    alpha = fzero( (a)->eq(a,mean(sample),mean(log(sample))), 1)
    lambda = alpha/mean(sample)
    return [alpha,lambda]
end

N = 10^4

mles10 = [mle(rand(gammaDist),10)) for _ in 1:N]
mles100 = [mle(rand(gammaDist),100)) for _ in 1:N]
mles1000 = [mle(rand(gammaDist),1000)) for _ in 1:N]

plot(first.(mles10),last.(mles10),"b.",ms="0.3",label="n = 10")
plot(first.(mles100),last.(mles100),"r.",ms="0.3",label="n = 100")
plot(first.(mles1000),last.(mles1000),"g.",ms="0.3",label="n = 1000")
xlabel(L"$\alpha$")
ylabel(L"$\lambda$")
xlim(0,8)
ylim(0,8)
legend(markerscale=20,loc="upper right")
```



Listing 5.13: A simple CI in practice

```
using Distributions, PyPlot
srand(2)

mu=5.57
alpha = 0.05
L(obs) = obs - (1-sqrt(alpha))
U(obs) = obs + (1-sqrt(alpha))

tDist = TriangularDist(mu-1,mu+1,mu)
N = 100

for k in 1:N
    observation = rand(tDist)
    LL,UU = L(observation), U(observation)
    plt[:bar](k,bottom = LL,UU-LL,color= (LL < mu && mu < UU) ? "b" : "r")
end

plot([0,N+1],[mu,mu],c="k",label="Parameter value")
legend(loc="upper right")
ylabel("Value")
xticks([])
xlim(0,N+1)
ylim(3,8)
```

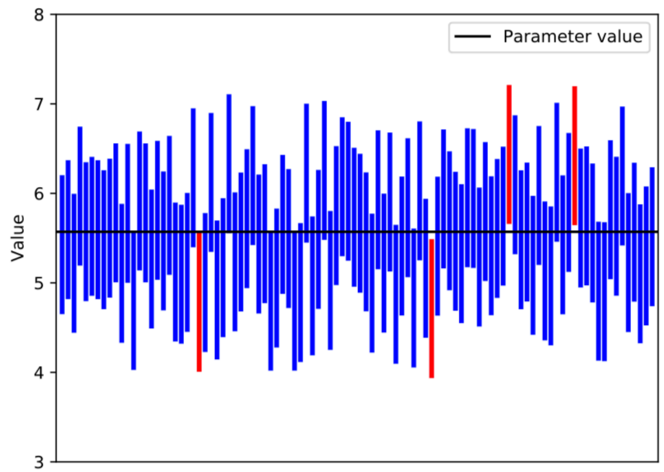
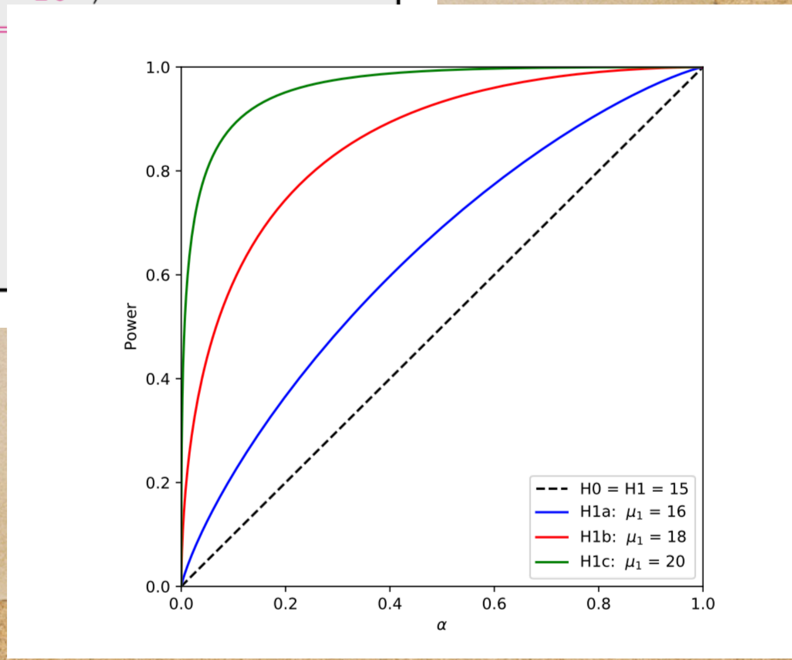


Figure 5.11: 100 confidence intervals. The blue confidence interval bars contain our unknown parameter, while the red ones do not.

Listing 5.16: Comparing receiver operating curves

```
using Distributions, StatsBase, PyPlot

mu0, mu1a, mu1b, mu1c, std = 15, 16, 18, 20, 2

dist0 = Normal(mu0,std)
dist1a = Normal(mu1a,std)
dist1b = Normal(mu1b,std)
dist1c = Normal(mu1c,std)

tauGrid = 5:0.1:25

falsePositive = ccdf(dist0,tauGrid)
truePositiveA = ccdf(dist1a,tauGrid)
truePositiveB = ccdf(dist1b,tauGrid)
truePositiveC = ccdf(dist1c,tauGrid)

figure("ROC",figsize=(6,6))
subplot(111)
plot([0,1],[0,1],"--k", label="H0 = H1 = 15")
plot(falsePositive, truePositiveA,"b", label=L"H1a: $\mu_1$ = 16")
plot(falsePositive, truePositiveB,"r", label=L"H1b: $\mu_1$ = 18")
plot(falsePositive, truePositiveC,"g", label=L"H1c: $\mu_1$ =

PyPlot.legend()
xlim(0,1)
ylim(0,1)
xlabel(L"\alpha")
ylabel("Power")
```



Chapter 6: Confidence Intervals

leviation s, then the probability statement (6.3) no longer holds. However, Section 5.2) we are able to correct the confidence interval to,

$$\overline{x} \pm t_{1-\alpha/2,n-1} \, \frac{s}{\sqrt{n}}.$$

is the $1-\alpha/2$ quantile of a t-distribution with $n-1$ degrees of freedom. as `quantile(TDist(n-1),1-alpha/2)`.

Listing 6.2: CI single sample population variance assumed unknown

```
using Distributions, HypothesisTests

data = readcsv("machine1.csv")[:,1]
xBar, n = mean(data), length(data)
s = std(data)
alpha = 0.1
t = quantile(TDist(n-1),1-alpha/2)

println("Calculating formula: ", (xBar - t*s/sqrt(n), xBar + t*s/sqrt(n)))
println("Using confint() function: ", confint(OneSampleTTest(xBar,s,n),alpha))
```

```
Calculating formula:       (52.49989385779555, 53.412518384764276)
Using confint() function: (52.49989385779555, 53.412518384764276)
```

$$T = \frac{\overline{X}_1 - \overline{X}_2 - (\mu_1 - \mu_2)}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}},$$

$$v = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(s_1^2/n_1\right)^2}{n_1 - 1} + \frac{\left(s_2^2/n_2\right)^2}{n_2 - 1}}.$$

$$T \underset{\text{approx}}{\sim} t(v).$$

$$\overline{x}_1 - \overline{x}_2 \pm t_{1-\alpha/2,v} \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}.$$

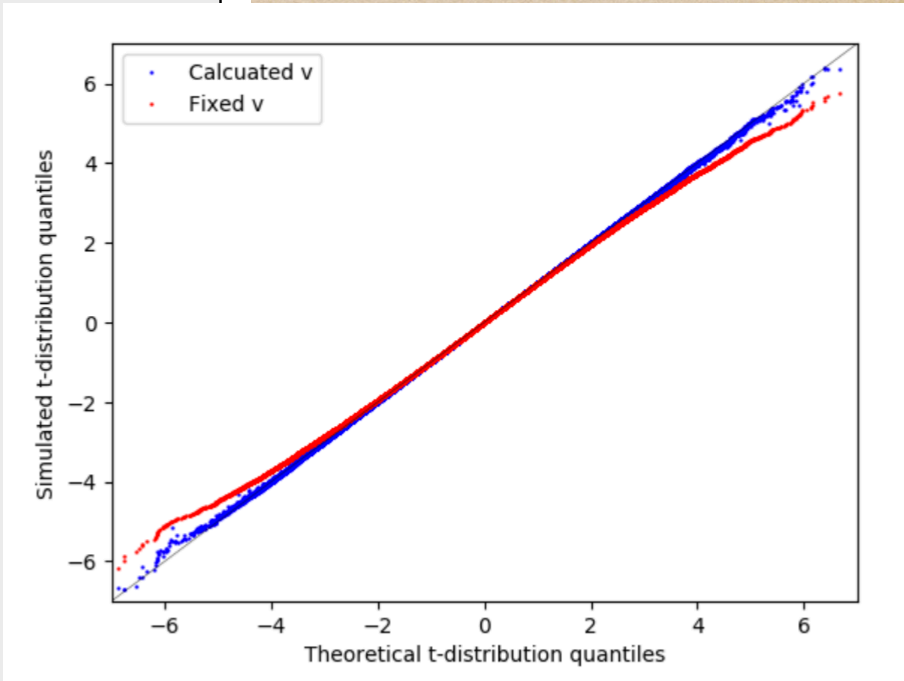
Listing 6.5: CI difference in population means variance unknown not assumed equal

```
using Distributions, HypothesisTests

data1 = readcsv("machine1.csv")[:,1]
data2 = readcsv("machine2.csv")[:,1]
xBar1, xBar2 = mean(data1), mean(data2)
s1, s2 = std(data1), std(data2)
n1, n2 = length(data1), length(data2)
alpha = 0.05

v = (s1^2/n1 + s2^2/n2)^2 / ( (s1^2/n1)^2 / (n1-1) + (s2^2/n2)^2 / (n2-1) )

t = quantile(TDist(v),1-alpha/2)

println("Calculating formula: ", (xBar1 - xBar2 - t*sqrt(s1^2/n1 + s2^2/n2),
                                   xBar1 - xBar2 + t*sqrt(s1^2/n1 + s2^2/n2)))
println("Using confint(): ",   confint(UnequalVarianceTTest(data1,data2),alpha))
```

```
Calculating formula: (1.0960161148824918, 2.9216026983153505)
Using confint():     (1.0960161148824918, 2.9216026983153505)
```

Listing 6.6: QQ plot of t-statistics for v calculated by Satterthwaite vs constant v

```
using Distributions, PyPlot

mu1, sig1, n1 = 0, 2, 8
mu2, sig2, n2 = 0, 30, 15
dist1 = Normal(mu1, sig1)
dist2 = Normal(mu2, sig2)

N = 10^6
tdArray = Array{Tuple{Float64,Float64}}(N)

df(s1,s2,n1,n2) =
    (s1^2/n1 + s2^2/n2)^2 / ( (s1^2/n1)^2/(n1-1) + (s2^2/n2)^2/(n2-1) )

for i in 1:N
    x1Data = rand(dist1, n1)
    x2Data = rand(dist2, n2)

    x1Bar,x2Bar = mean(x1Data),mean(x2Data)
    s1,s2 = std(x1Data),std(x2Data)

    tStat = (x1Bar - x2Bar) / sqrt(s1^2/n1 + s2^2/n2)

    tdArray[i] = (tStat , df(s1,s2,n1,n2))
end
tdArray = sort(tdArray,1)

invVal(data,i) = quantile(TDist(data),i/(N+1))

xCoords = Array{Float64}(N)
yCoords1 = Array{Float64}(N)
yCoords2 = Array{Float64}(N)

for i in 1:N
    xCoords[i] = first(tdArray[i])
    yCoords1[i] = invVal(last(tdArray[i]),i)
    yCoords2[i] = invVal(n1+n2-2,i)
end

plot(xCoords,yCoords1,label="Calcuated v","b.",ms="1.5")
plot(xCoords,yCoords2,label="Fixed v","r.",ms="1.5")
plot([-10,10],[-10,10],"k",lw="0.3")
legend(loc="upper left")
xlim(-7,7)
ylim(-7,7)
xlabel("Theoretical t-distribution quantiles")
ylabel("Simulated t-distribution quantiles")
savefig("vDOF_comparions.pdf")
```

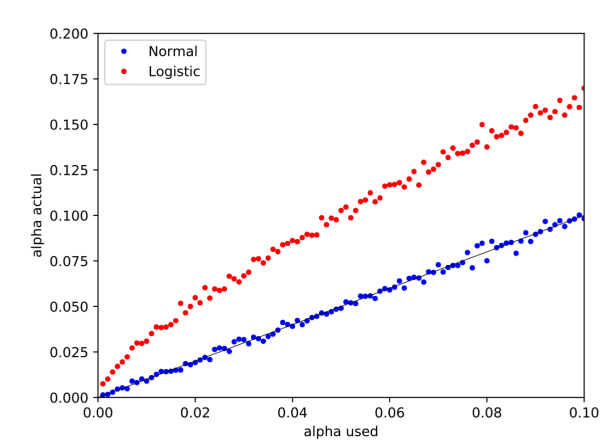


$$\frac{(n-1)s^2}{\chi^2_{n-1,1-\alpha/2}} < \sigma^2 < \frac{(n-1)s^2}{\chi^2_{n-1,\alpha/2}}.$$

$$f(x) = \frac{e^{-\frac{x-\mu}{s}}}{s\left(1+e^{-\frac{x-\mu}{s}}\right)^2}.$$

Listing 6.10: Actual alpha vs alpha used

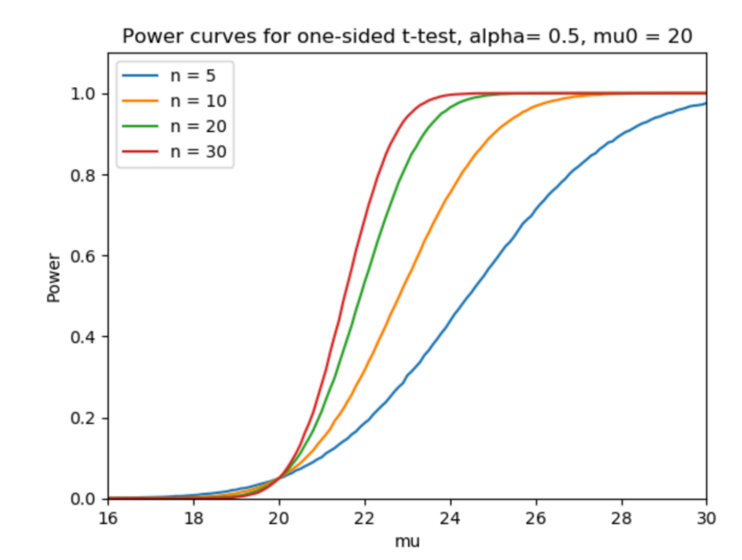
```
using Distributions, PyPlot

n, N = 15, 10^4
alphaUsed = 0.001:0.001:0.1
dNormal = Normal(2,sqrt(2))
dLogistic = Logistic(2,0.88)

function alphaSimulator(dist, n, alpha)
    popVar = var(dist)
    coverageCount = 0
    for i in 1:N
        sVar = var(rand(dist, n))
        L = (n - 1) * sVar / quantile(Chisq(n-1),1-alpha/2)
        U = (n - 1) * sVar / quantile(Chisq(n-1),alpha/2)
        coverageCount +=  L < popVar && popVar < U
    end
    1 - coverageCount/N
end

plot(alphaUsed, alphaSimulator.(dNormal,n,alphaUsed),".b",label="Normal")
plot(alphaUsed, alphaSimulator.(dLogistic, n, alphaUsed),".r",label="Logistic")
plot([0,0.1],[0,0.1],"k",lw=0.5)
xlabel("alpha used")
ylabel("alpha actual")
legend(loc="upper left")
xlim(0,0.1)
ylim(0,0.2)
```



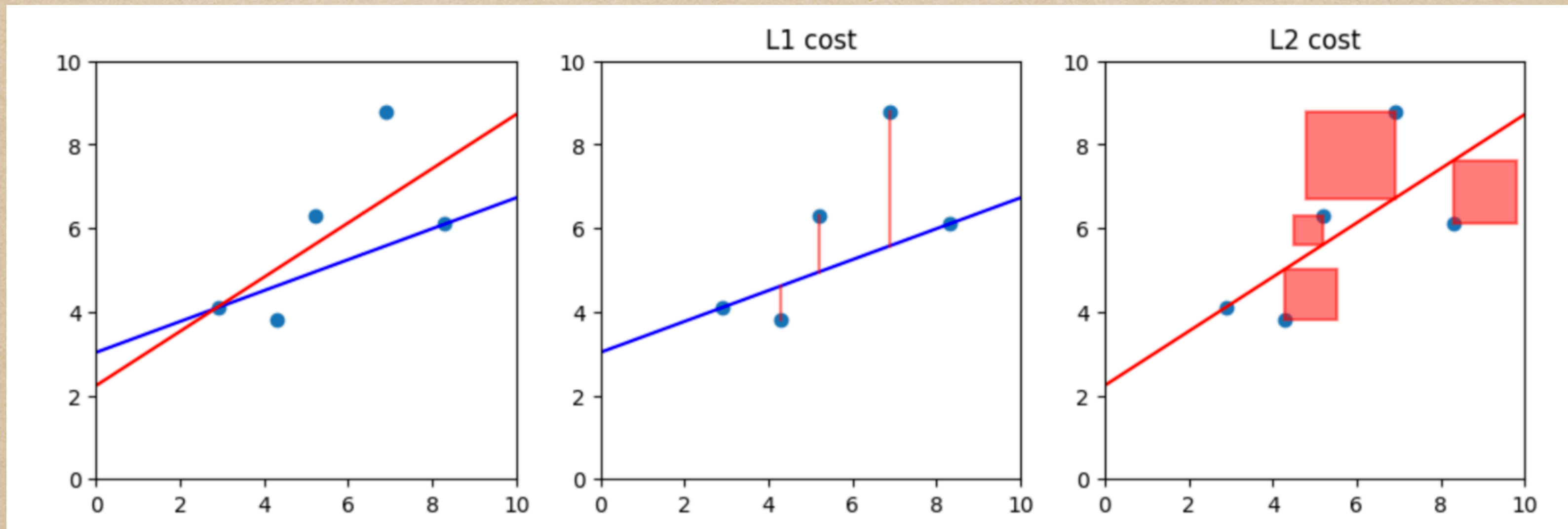
Chapter 7: Hypothesis Testing

Listing 7.12: powercurves4

```
function powerEstimate(mu0,mu1,sig,n,alpha,NN)
    sampleH1 = [tStat(mu0,mu1,sig,n) for _ in 1:NN]; #generate a whole lot of t-sta
    critVal = quantile(TDist(n-1),1-alpha)
    sum(sampleH1 .>critVal)/length(sampleH1)
end

rangeMu1 = 16:0.1:30
powersN05 = [powerEstimate(20,mu1,5,5,0.05,10^5) for mu1 in rangeMu1]
powersN10 = [powerEstimate(20,mu1,5,10,0.05,10^5) for mu1 in rangeMu1]
powersN20 = [powerEstimate(20,mu1,5,20,0.05,10^5) for mu1 in rangeMu1]
powersN30 = [powerEstimate(20,mu1,5,30,0.05,10^5) for mu1 in rangeMu1];

PyPlot.plot(rangeMu1,powersN05, label="n = 5")  # power curve for 5 samples
PyPlot.plot(rangeMu1,powersN10, label="n = 10")  # power curve for 10 samples
PyPlot.plot(rangeMu1,powersN20, label="n = 20"); # power curve for 20 samples
PyPlot.plot(rangeMu1,powersN30, label="n = 30")  # power curve for 30 samples
PyPlot.legend()
xlim(16,30)
ylim(0,1.1)
xlabel("mu")
ylabel("Power")
title("Power curves for one-sided t-test, alpha= 0.5, mu0 = 20");
savefig("powercurves2.png")
```



Chapter 8: Regression Models


L1 cost
L2 cost

Listing 8.5: Logistic Regression

```julia
using PyPlot, DataFrames, Distributions

x = [0.50,0.75,1.00,1.25,1.50,1.75,1.75,2.00,2.25,2.50,2.75,3.00,3.25,3.50,4.00,4.25,4
```



```julia
y = [0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1]
data = DataFrame(X = x, Y = y)

model = glm(Y ~ X, data, Binomial(), LogitLink())

# Now verify using
# Probability of passing exam = 1/(1+exp(-(-4.0777+1.5046* Hours)))

C = coef(model)

xm = linspace(0,maximum(data[1]),100)
ym = 1./(1+exp(-(C[1]+C[2].*xm)))
xlim = (0,maximum(x))
PyPlot.scatter(x,y)
PyPlot.plot(xm,ym,"r")
model
```

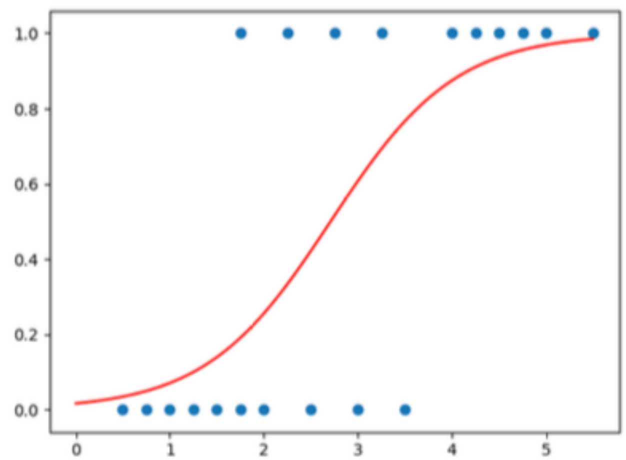
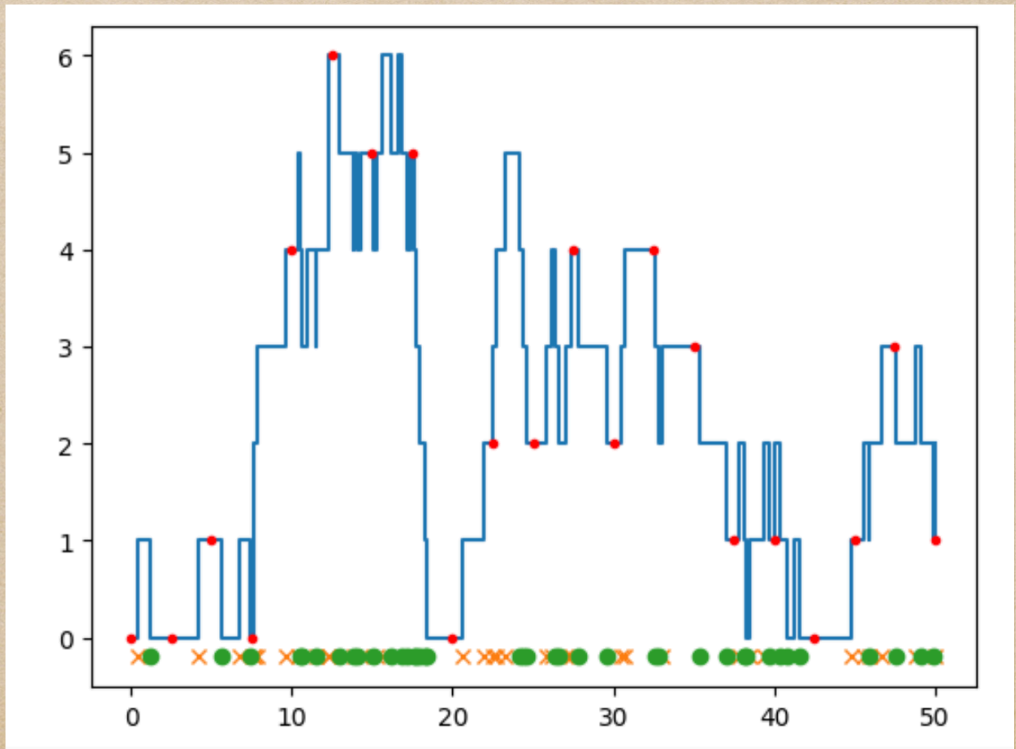


Figure 8.5

Chapter 9:
Simulation of Dynamic Models

```julia
using DataStructures,Distributions

function simulateMM1(lambda,mu,Q0,T)
    t, Q = 0.0 , Q0
    tValues = [0.0]
    qValues = [Q0]
    while t<T
        if Q == 0 #arrival to an empty system
            t += rand(Exponential(1/lambda))
            Q = 1
        else  #change of state when system is not empty
            t += rand(Exponential(1/(lambda +mu)))
            Q += 2(rand() < lambda/(lambda+mu)) -1
        end
        push!(tValues,t)
        push!(qValues,Q)
    end
    return[tValues, qValues]
end
```

```julia
using PyPlot
T = 50
Q0 = 0
queueTraj = simulateMM1(0.9,1.0,Q0,T);
times = queueTraj[1]
qValues = queueTraj[2]
temp = stichSteps(times,qValues)
timesForPlot = temp[1]
qForPlot = temp[2]

delta = 2.5
qSampled = sampleQ(times,qValues,delta)
plot(timesForPlot,qForPlot)
plot(0:delta:T,qSampled,".",color="r")

arrDep = findArrDep(times,qValues)
arrs = arrDep[1]
deps = arrDep[2]
plot(arrs,-0.2*ones(length(arrs)),"x")
plot(deps,-0.2*ones(length(deps)),"o");
```



Chapter 10:
A View Forward

Chapter 10

A View Forward - Skeleton Only

This chapter is currently under construction.

10.1 Additional Language Features

10.2 A Variety of Julia Packages

10.3 R DataSets and using R Packages

Listing 10.1: Using RDatasets

```
using RDatasets
hair = dataset("datasets","HairEyeColor")
```

10.4 Using and Calling Python Packages

10.5 Other Integrations

10.6 Further Reading

The End