Julia - A Modern Platform for Statistical Computing & Data Science

Date 14th Feb, 2026

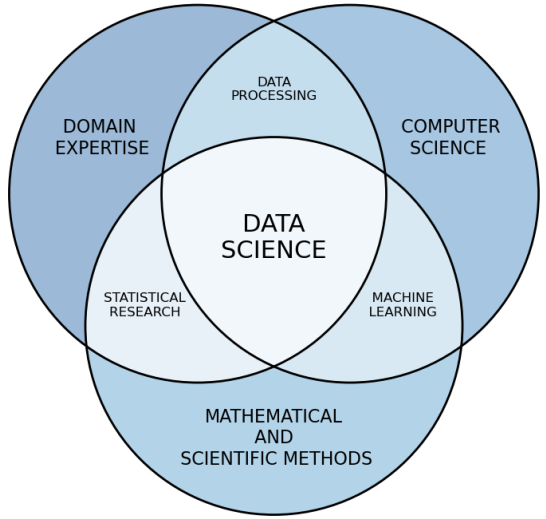
Sujit Sandipan Chaugule

B. Tech in Pharmaceutical Sciences & Technology

Personal Web Page: https://sites.google.com/view/statsphere

Foundations of Statistical Computing & Data Science

- **Statistical computing** and **data science** are foundational to modern scientific inquiry and evidence-based decision-making.
- Rapid growth in **data volume**, **variety**, and **complexity** across domains such as **ecology**, **biology**, **engineering**, **economics**, and **social sciences** has made computational approaches indispensable.
- **Statistical computing** provides the bridge between statistical theory and real-world applications through **efficient algorithms**, **numerical methods**, **simulation**, and **data visualisation**.
- **Data science integrates statistics**, **computer science**, and **domain knowledge** to transform raw data into meaningful and actionable insights.



Source: Data science - Data Cymru

Role of Statistical Computing in Modern Analysis.

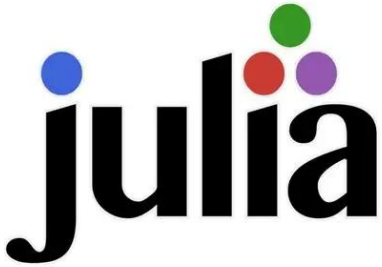
- Statistical computing enables the practical implementation of statistical models for large and complex datasets that cannot be handled manually.
- It combines statistical reasoning with algorithms, data structures, and efficient computing techniques from computer science.
- Key capabilities include simulation-based inference, probabilistic modelling, optimization, uncertainty quantification, and performance evaluation.
- It plays a central role in engineering systems, machine learning, and artificial intelligence by supporting predictive modelling and learning under uncertainty.

Impact of Data Science Across Domains

- In healthcare, data science supports disease prediction, personalized medicine, medical imaging analysis, and drug discovery.
- In engineering and manufacturing, data-driven approaches enable smart systems, predictive maintenance, digital twins, and real-time optimization.
- Financial, business, and environmental sectors rely on data science for risk management, forecasting, fraud detection, and climate modeling.
- Together, statistical computing and data science form a unifying framework that drives innovation, efficiency, and informed decision-making in modern science and engineering.

Limitations of Traditional Workflows in Statistical Computing & Data Science

- Statistical models use compact mathematical notation, but implementation in R, Python, or MATLAB is verbose and library-dependent, obscuring assumptions and increasing coding errors.
- Interpreted execution and reliance on external optimized libraries introduce overhead from language boundaries and memory copying, limiting scalability for large simulations and Monte Carlo methods.
- Exploratory analysis and production computation are often disconnected, requiring substantial code restructuring and reducing efficiency and reproducibility.
- These limitations motivate Julia as a unified platform combining mathematically expressive modeling with high-performance execution for modern statistical computing and data science.

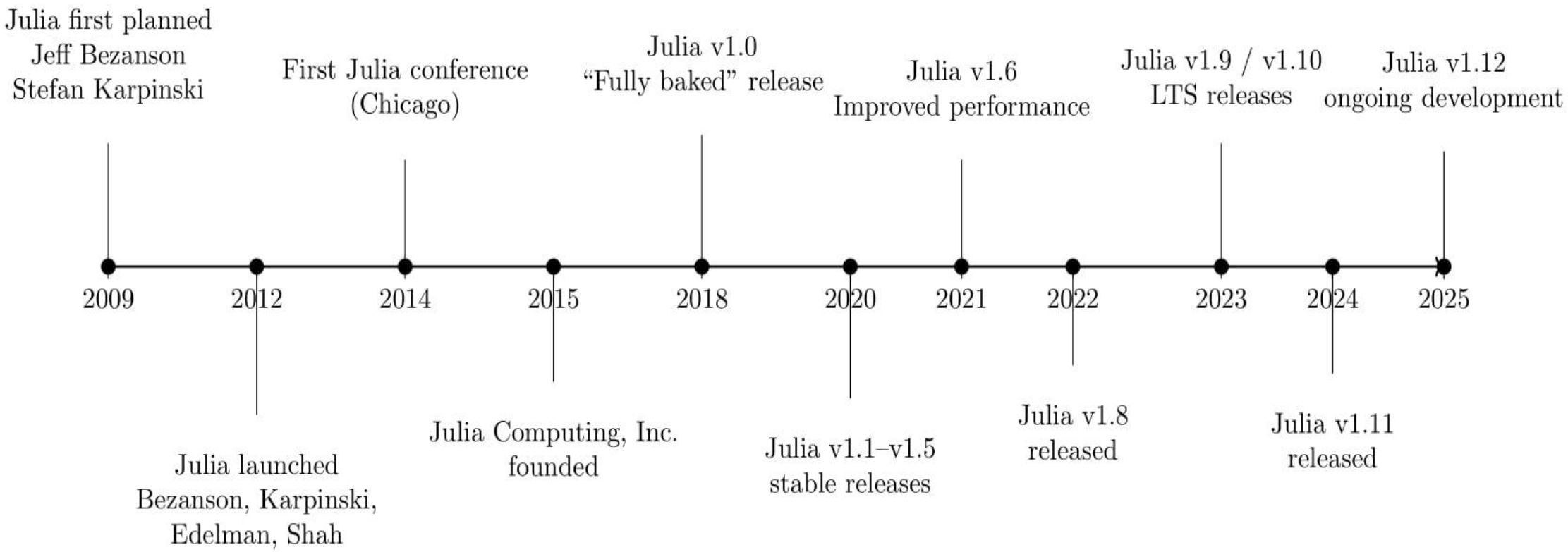


Julia: A New Platform

- Julia is an open-source, high-performance programming language introduced in 2012 by **Stefan Karpinski**, **Viral B. Shah**, **Jeff Bezanson** and **Alan Edelman**.
- It combines the **interactive syntax** of languages such as **Python**, **MATLAB**, and **R** with the speed of compiled languages like **C**, **C++**, and **Fortran**.
- Julia was designed for computationally intensive scientific and technical computing, balancing ease of use with execution efficiency.
- The language bridges **productivity-oriented** and performance-oriented programming paradigms, making it accessible to users from diverse technical computing backgrounds.

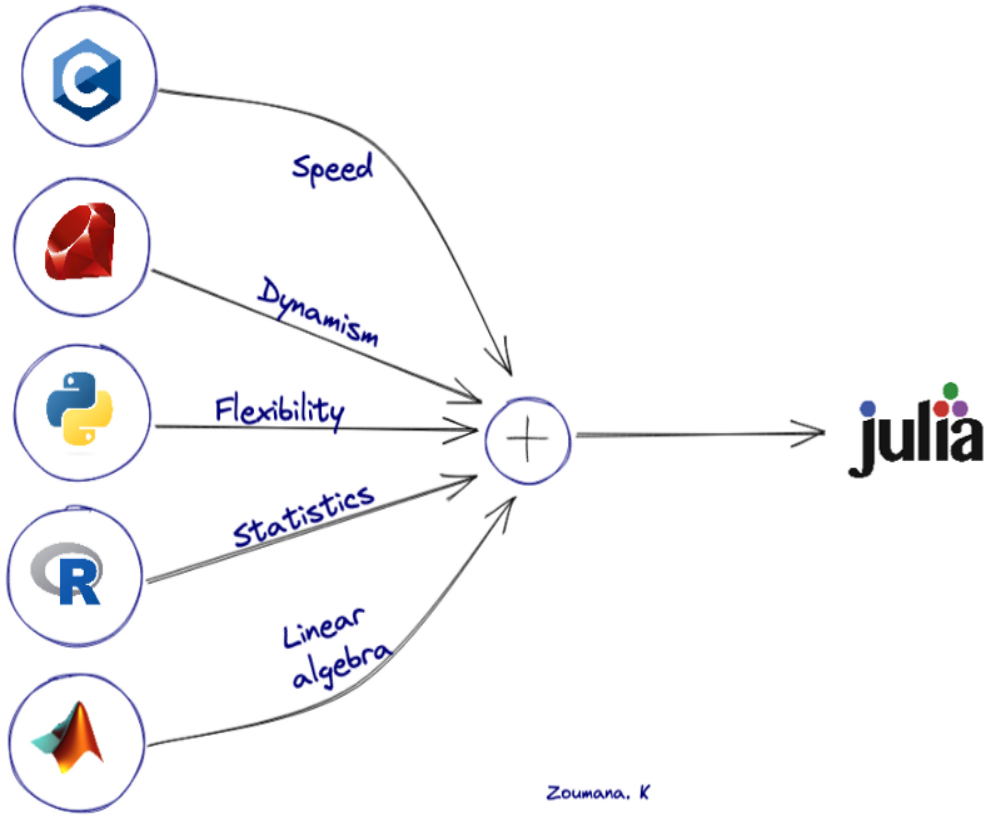
Julia: A New Platform

A timeline illustrating the origin and step-by-step evolution of the **Julia programming language**



Julia: A New Platform

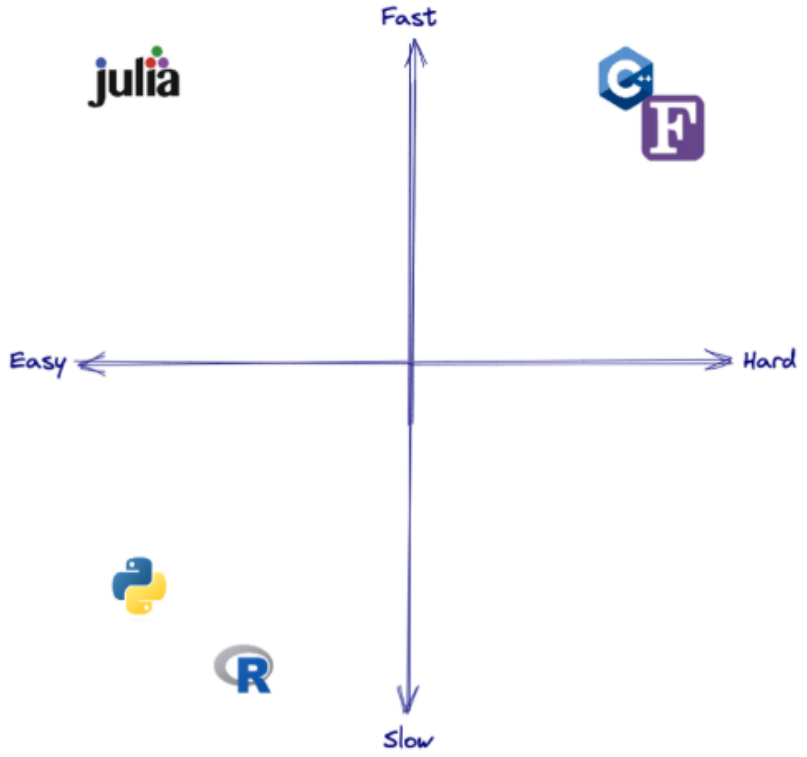
- High-performance dynamic language with a **just-in-time (JIT) compiler**, delivering speeds comparable to statically compiled languages like **C** and **Fortran**.
- Supports **distributed parallel execution**, precise numerical computations, and a comprehensive library of mathematical functions for scientific applications.
- Designed to overcome limitations of **R**, **Python**, and **MATLAB** in handling **large datasets** and **complex computations efficiently**.
- Extensive package ecosystem via **Integrated Package Management (IPM)**, suitable for **scientific computing**, **data analysis**, **machine learning**, and **scalable technical applications**.



Source: Introduction to Julia with a comparison to Python and R

Julia: A New Platform

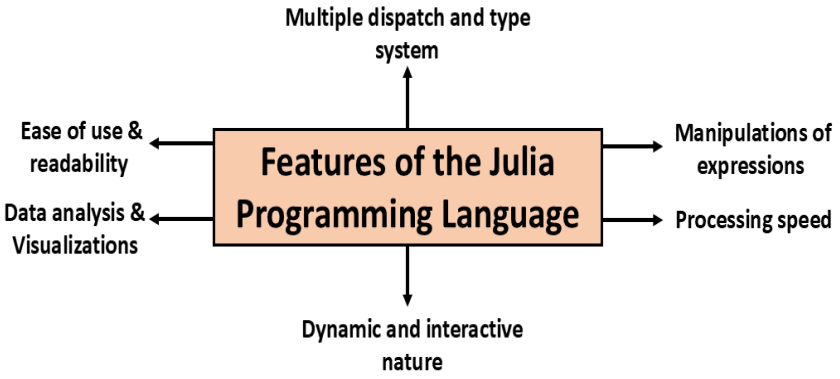
- Julia provides a straightforward and expressive syntax, making it easy to learn for users familiar with **MATLAB**, **R**, or **Python**.
- Multiple dispatch allows functions to be specialized based on argument types, enabling **high-performance generic programming** and easy extension of existing functions.
- Julia can directly call C, Fortran, and Python libraries, and package management via **IPM** simplifies access to community-developed tools.
- Julia combines the productivity of high-level dynamic languages with the execution speed of compiled languages, addressing the **two-language problem**.
- Built-in support for **parallel** and **distributed** computing allows efficient execution across multiple cores or machines, suitable for scientific computing and machine learning.



Source: Introduction to Julia with a comparison to Python and R

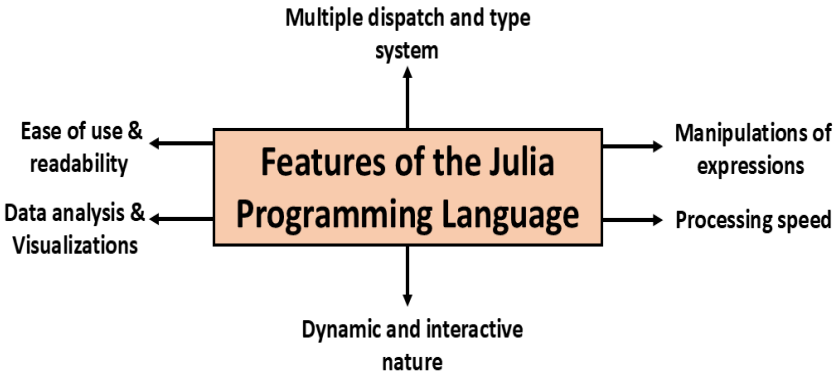
Some Important Features of Julia

- Julia provides a dynamic and interactive programming environment, allowing rapid experimentation and development without rewriting large portions of code.
- Its high execution speed approaches that of compiled languages like C/C++, making it ideal for large-scale data processing and complex computations.
- Julia supports parallel and distributed computing, reducing computation time on big datasets and improving scalability for statistical and machine learning workflows.
- Metaprogramming allows programs to construct and manipulate expressions as first-class values, addressing the two-language and expression problems in scientific computing.



Some Important Features of Julia

- Julia's simple syntax and high-level abstractions make model development, statistical analysis, and data workflows accessible to **Python**, **R**, or **MATLAB** users.
- Multiple dispatch and a robust type system enable expressive, modular, and reusable code while maintaining high performance across diverse implementations.
- Data analysis and visualization are highly efficient; MATLAB and Octave code can be reused via **MatlabCompat.jl**, while **RCall.jl** and **PyCall.jl** allow seamless **R/Python** integration.
- Julia unifies productivity and performance in a single language, eliminating the need to switch between high-level and low-level languages.



Mathematical Equations to R or Python Code: Lost in Translation

- **Loss of meaning in translation:** When mathematical equations are converted to R or Python code, the core idea can get obscured, making it hard to connect the code back to the theory.
- **Reduced transparency:** This disconnect lowers clarity within an organization or team, as not everyone can easily trace how the math is implemented in the code.
- **Communication gap:** Even fellow data scientists may struggle to fully understand the approach or logic behind the code if the mathematical intent isn't clear.
- **Lack of explainability in AI/ML:** This gap widens further in machine learning and AI, where models are already complex, leading to reduced interpretability.

Mathematical Equations to R or Python Code: Lost in Translation

Here consider the example of a bivariate normal PDF

$$f(x) = \frac{1}{2\pi|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu)^{\top}\Sigma^{-1}(x-\mu)\right\},$$

The bivariate normal distribution with mean vector and covariance matrix is:

$$\mu' = (\mu_1, \mu_2), \quad \Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{bmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Where $\sigma_{12} = \sqrt{\sigma_{11}}\sqrt{\sigma_{22}}\rho_{12}$, Simplify the expression of the density function and get the expression in terms of scalar quantities only.

R Code:

```r
library(mvtnorm)
mu = c(0, 0)
Sigma = matrix(data = c(0.5, 0, 0, 1),2, 2, byrow = TRUE)
x = rmvnorm(n = 10000, mean = mu, sigma = Sigma)
```

Julia Code:

```julia
using Distributions
μ = [0, 0]
Σ = [0.5 0; 0 1]
x = rand(MvNormal(μ, Σ), 10000)
```

Julia's deep learning, machine learning, and statistical computing ecosystem

An overview of Julia's ecosystem for deep learning, machine learning, and statistical computing is presented, highlighting community-driven packages for data manipulation, statistical modelling, scientific machine learning, and visualization. The following table provides a concise summary of major Julia communities, representative packages, and their primary application areas relevant to statistical computing and data science.

SR. NO.	COMMUNITY	REPRESENTATIVE PACKAGES	PRIMARY APPLICATIONS
1	JuliaStats	StatsBase.jl, MultivariateStats.jl, HypothesisTests.jl	Statistical modeling, inference, and machine learning
2	JuliaImages	ImageTransformations.jl, Images.jl, ImageSegmentation.jl	Image analysis and image processing
3	SciML	DifferentialEquations.jl, ModelingToolkit.jl	Scientific machine learning and mathematical modeling
4	JuliaData	DataFrames.jl, DataFramesMeta.jl	Data manipulation, storage, and input–output operations
5	FluxML	Flux.jl, MacroTools.jl, GeometricFlux.jl	Machine learning and deep learning
6	JuliaPlots	StatsPlots.jl, Plots.jl ,PlotlyJS.jl	Data visualization and graphical analysis

Julia's deep learning, machine learning, and statistical computing ecosystem

- **Statistical Computing:** Julia provides a unified, high-performance environment for data handling, statistical modelling, and visualization through the *JuliaStats* and *JuliaData* ecosystems. Core packages such as **`StatsBase.jl`**, **`HypothesisTests.jl`**, **`MultivariateStats.jl`**, **`DataFrames.jl`**, and **`StatsPlots.jl`** support descriptive statistics, inference, multivariate analysis, data wrangling, and exploratory visualization.
- **Machine Learning:** Julia's ML ecosystem is centred around **`MLJ`**, which offers a unified interface for model training, tuning, evaluation, and comparison. Leveraging multiple dispatch and type specialization, Julia enables flexible yet efficient ML workflows and supports cross-language integration via tools like **`JuliaConnectoR.`**
- **Deep Learning:** Native frameworks such as **`Flux.jl`** and **`Lux.jl`** enable expressive, high-performance neural network modelling with automatic differentiation and GPU support. Extensions like **`DiffEqFlux.jl`**, **`GraphNeuralNetworks.jl`**, and **`GeometricFlux.jl`** integrate deep learning with scientific computing, differential equations, and graph-structured data.

Notable Uses of Julia

Company

- Amazon
- Apple
- Capital One
- Google
- IBM
- Intel
- Pfizer
- JPMorgan
- Microsoft

Organizations

- NASA
- US Fed Reserve

Universities/ Institutes

- MIT
- Stanford
- TAMU
- ETH Zurich
- CMU
- CMI

IDEs/Environments for Julia

- VS Code + Julia extension
- Pluto.jl (reactive notebook)
- Jupyter (via IJulia)
- Google Colab

Installation of Packages in Julia

Julia provides a built-in package manager, **Pkg**, to install and manage external libraries that extend the language's capabilities. It allows users to easily install single or multiple packages, or even packages directly from GitHub repositories, ensuring flexibility and reproducibility in project environments.

- **Install a Single Package**

```
using Pkg
Pkg.add("Plots")
```

- **Install multiple packages at once**

```
using Pkg
Pkg.add(["Distributions", "Plots", "Statistics"])
```

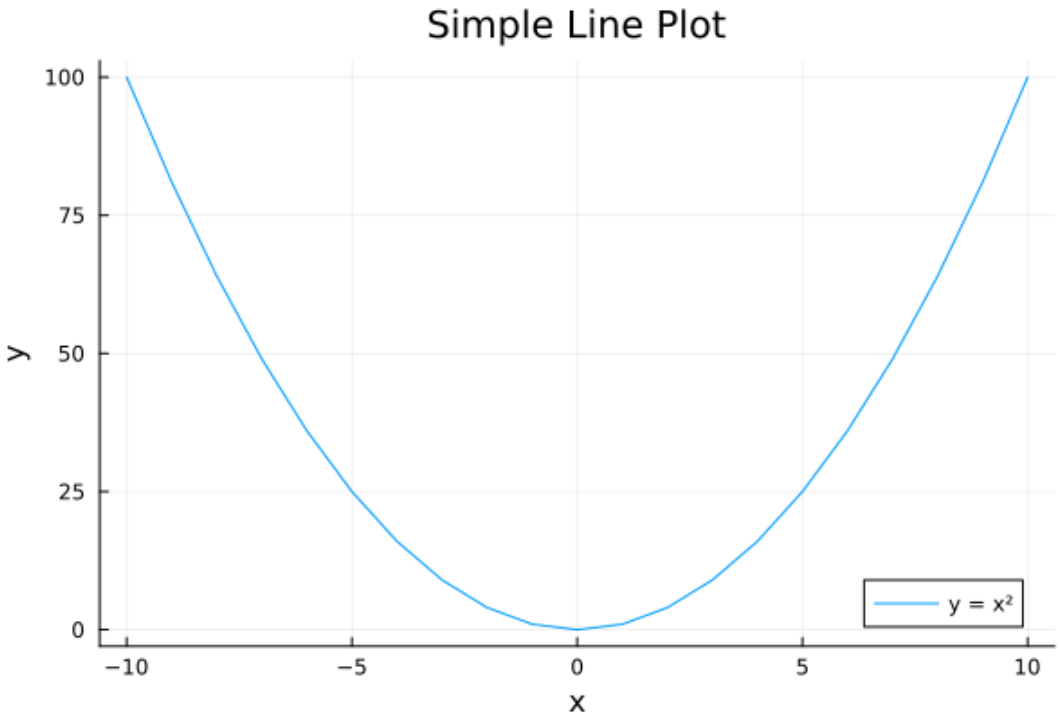
- **Install a package from a GitHub URL**

```
using Pkg
Pkg.add(url = "https://github.com/JuliaStats/GLM.jl")
```

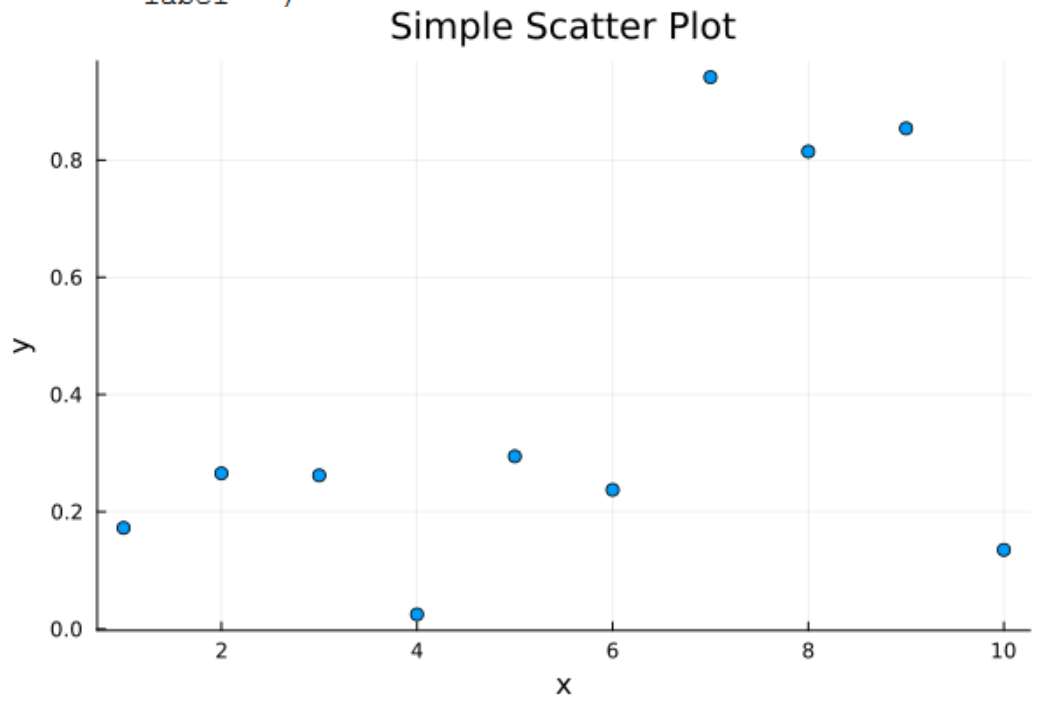
Basic Plotting in Julia

Julia provides powerful and flexible tools for data visualization through packages such as **`Plots.jl`** and **`Makie.jl`**. With simple and intuitive syntax, users can quickly create line plots, scatter plots, histograms, and more. Basic plotting helps in understanding data patterns, trends, and distributions effectively.

```julia
using Plots
x = -10:10
y = x.^2
plot(x, y, xlabel="x", ylabel="y", title="Simple Line Plot",
        label="y = x²")
```




```julia
using Plots
x = 1:10
y = rand(10)    # random values
scatter(x, y, xlabel="x", ylabel="y", title="Simple Scatter Plot",
        label="")
```



Probability Distributions in Julia

Binomial Distribution

The binomial distribution describes the probability of k successes in n independent Bernoulli trials with success probability p. The PMF is:

$$P(X = k \mid p) = \binom{n}{k} p^k (1-p)^{n-k}, \quad k = 0,1,\ldots,n; \ 0 \le p \le 1$$

For the example of an ecological study on a rare plant species, $n = 10$ and $p = 0.3$, representing the probability that a rare plant is found at a given location. The probability of observing exactly k successes can be calculated as:

```
using Distributions
```

```
pdf(Binomial(10, 0.3), 3)  # Probability of observing 3 successes
```

```
0.2668279320000006
```

To compute the cumulative probability $\mathbf{P}(X \le k)$, use:

```
cdf(Binomial(10, 0.3), 3) # Probability of at most 3 successes
```

```
0.6496107184000002
```

To find the number of successes corresponding to a cumulative probability, use

```
quantile(Binomial(10, 0.3), 0.7)  # Smallest k with P(X ≤ k) ≥ 0.7
```

```
4
```

Simulate outcomes for trials using:

```
rand(Binomial(10, 0.3), 5)
```

```
5-element Vector{Int64}:
 3
 5
 1
 1
 4
```

Probability Distributions in Julia

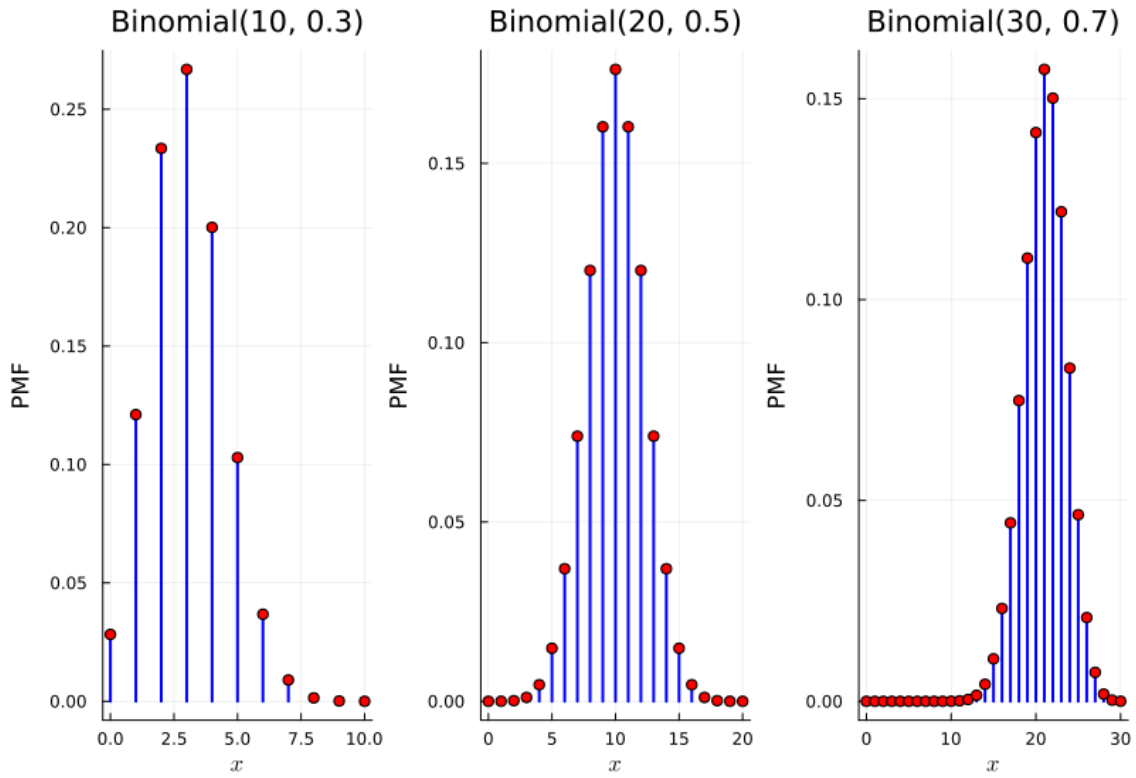
Binomial Distribution

To visualize how the binomial distribution behaves under different values of n and p, we generate a plot and its PMF for three parameter combinations

- $n = 10, p = 0.3,$
- $n = 20, p = 0.5,$
- $n = 30, p = 0.7.$

```
using Plots, Distributions, LaTeXStrings
```

```
n_vals = [10, 20, 30]
p_vals = [0.3, 0.5, 0.7]
plt = plot(layout = (1, 3), size = (800, 550))
for i in eachindex(n_vals)
    n = n_vals[i]
    p = p_vals[i]
    x = 0:n
    y = pdf.(Binomial(n, p), x)
    plot!(plt[i], x, y, color = "blue",lw = 2,seriestype = :stem,
        label = "", title = "Binomial($n, $p)", xlabel = L"x", ylabel = "PMF")
    scatter!(plt[i], x, y, marker = (:circle, 4), color = "red",label = "")
end
display(plt)
```



Probability Distributions in Julia

Poisson Distribution

The Poisson distribution is defined by a single parameter, λ- the average rate of events per unit time or space. The

PMF is: $\mathrm{P}(X = k|\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}; k = 0,1,\cdots;\ 0 < \lambda < \infty,$

where k is the number of events (sightings in our example) we are interested in To compute the probability of observing exactly bird sightings year:

```
using Distributions
```

```
pdf(Poisson(3), 2)  # Probability of observing 2 sightings
```

```
0.22404180765538775
```

```
pdf(Poisson(3), 5)  # # Probability of observing 5 sightings
```

```
0.10081881344492458
```

To compute the cumulative probability $\mathrm{P}(X \leq k)$,representing the probability of observing up to k sightings:

```
cdf(Poisson(3), 2) # Probability of observing at most 2 sightings
```

```
0.42319008112684353
```

```
cdf(Poisson(3), 5) # Probability of observing at most 5 sighti
```

```
0.9160820579686966
```

To determine the number of sightings corresponding to a given cumulative probability:

```
quantile(Poisson(3), 0.8)  # Smallest k with P(X ≤ k) ≥ 0.8
```

```
4
```

To simulate bird sightings over multiple years:

```
rand(Poisson(3), 10)  # Simulates the number of sightings for 10 years
```

```
10-element Vector{Int64}:
 2
 0
 3
 4
 3
 4
 5
 4
 2
 4
```

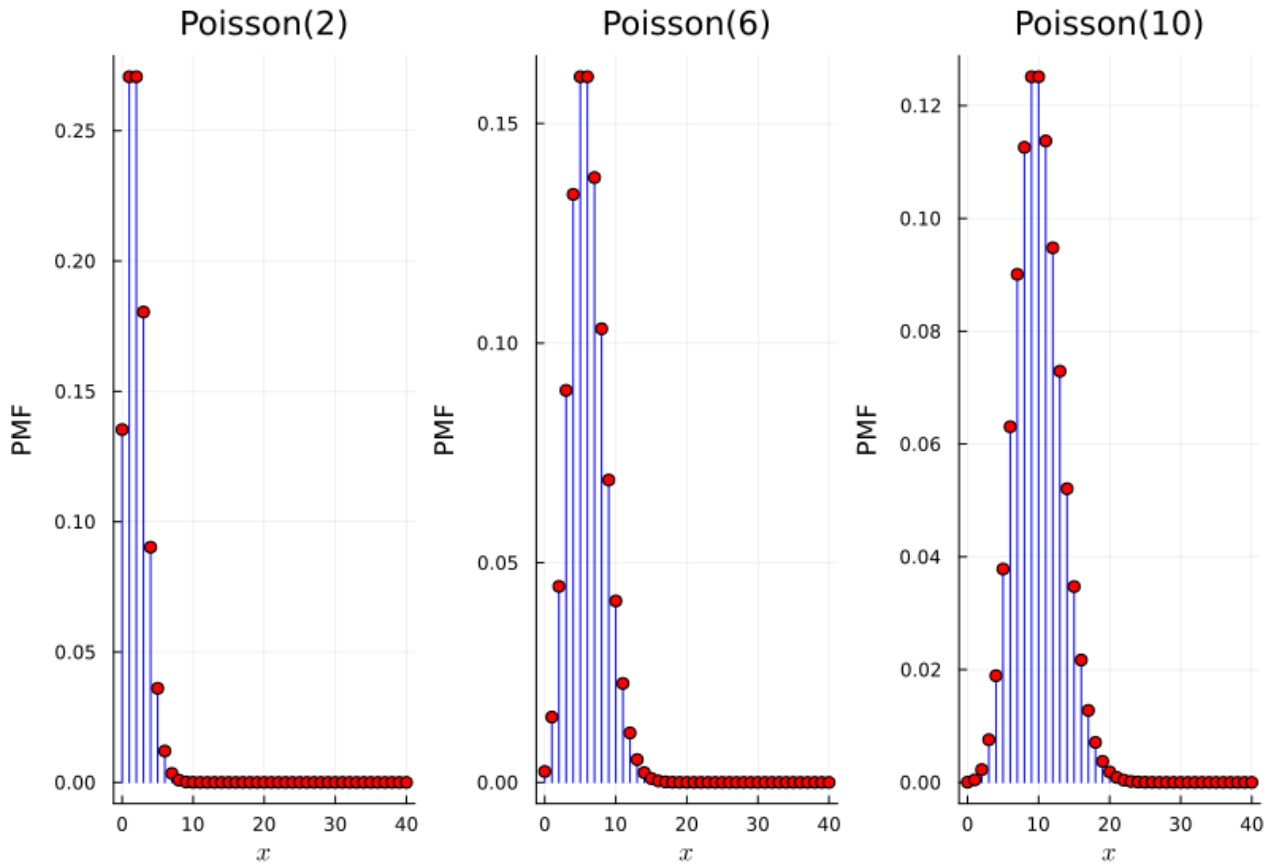
Probability Distributions in Julia

Poisson Distribution

The probability mass functions for various values of λ, highlighting the distribution's tendency to become more symmetric as λ increases.

```
using Plots, Distributions, LaTeXStrings
```

```
λ_vals = [2, 6, 10]
n = 40
plt = plot(layout = (1, 3), size = (800, 550))
for i in eachindex(λ_vals)
    λ = λ_vals[i]
    x = 0:n
    y = pdf.(Poisson(λ), x)
    plot!(plt[i], x, y, color = "blue",seriestype = :sticks,
        title = "Poisson($λ)", xlabel = L"x", ylabel = "PMF",label = "")
    scatter!(plt[i], x, y, marker = (:circle, 4), color = "red",label = "")
end
display(plt)
```



Probability Distributions in Julia

Normal Distribution

The normal distribution, also known as the Gaussian distribution, It models continuous data that tend to cluster around a central mean, with a symmetrical bell-shaped curve. If the random variable X is following the normal distribution with mean μ and variance σ^2 then it can be mathematically written as $X \sim \mathcal{N}(\mu, \sigma^2)$ .The PDF is:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad -\infty < x < \infty,\ \mu \in \mathbb{R},\ \sigma > 0,$$

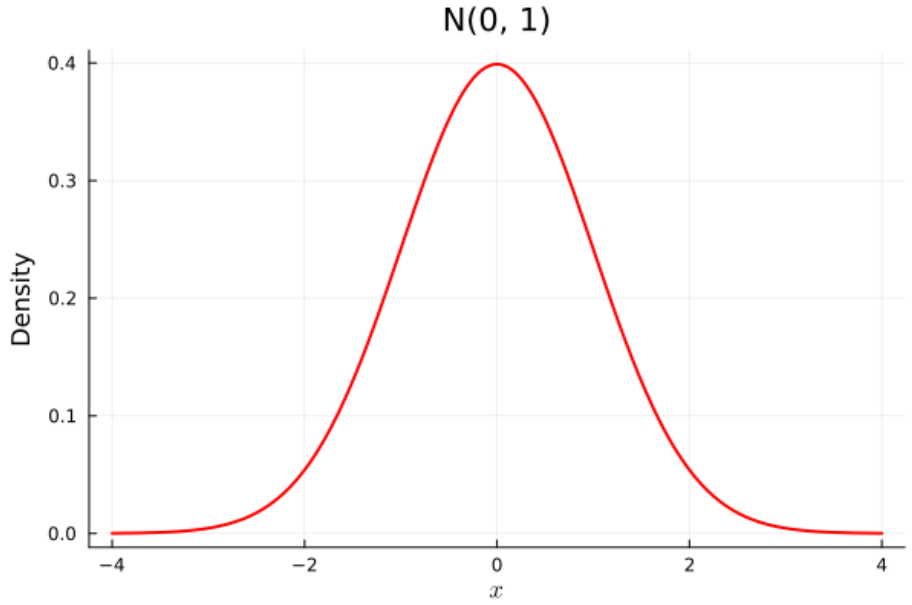
where μ is mean and σ is the standard deviation (spread).

```
using Plots, LaTeXStrings
```

```
μ = 0  # mean
σ = 1  # sd

function f(x, μ, σ)
    (1 / (sqrt(2 * π) * σ)) * exp(-((x - μ)^2) / (2 * σ^2))
end
# alternate way to write function
f(x, μ, σ) = (1 / (sqrt(2 * π) * σ)) * exp(-((x - μ)^2) / (2 * σ^2))

plot(x->f(x, μ, σ), -4, 4, color = "red", lw = 2, xlabel = L"x",
    ylabel = "Density", label = "", title = "N(0, 1)")
```



Probability Distributions in Julia

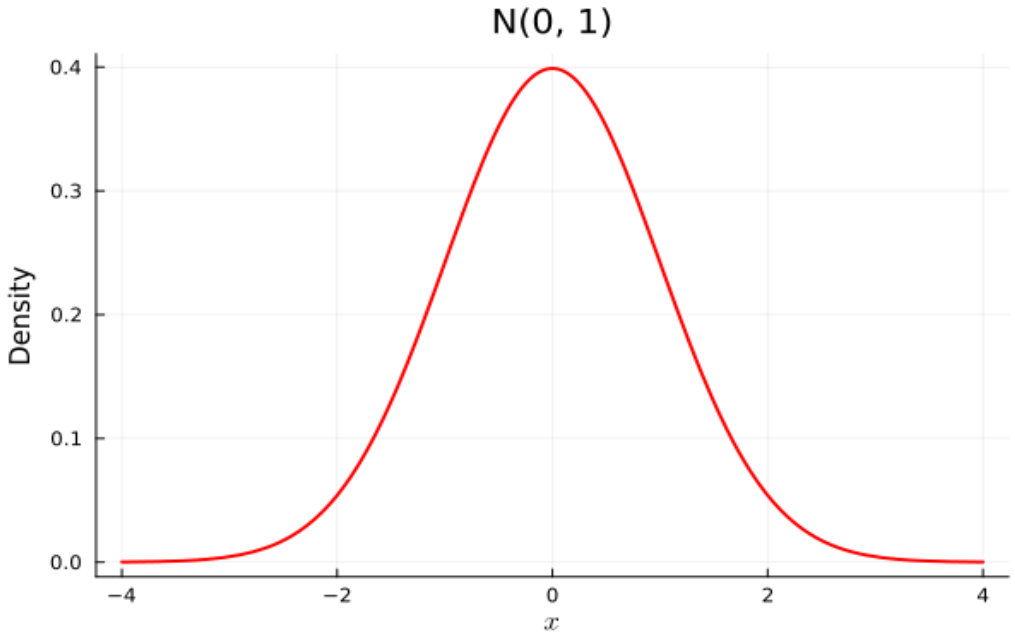
Normal Distribution

Also, we can use in-built function

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad -\infty < x < \infty,\ \mu \in \mathbb{R},\ \sigma > 0,$$

```
using Plots, LaTeXStrings, Distributions
```

```
μ = 0 # mean
σ = 1 # sd
plot(x->pdf(Normal(μ, σ), x), -4, 4,color = "red", lw = 2, xlabel = L"x",
    ylabel = "Density", label = "", title = "N(0, 1)")
```



Probability Distributions in Julia

Normal Distribution

Example: Suppose the diameters of bolts manufactured by a machine follow a normal distribution with a mean of 5 mm and a standard deviation of 0.2 mm. A quality control analyst wants to calculate the probability that a randomly selected bolt has a diameter between 4.8 mm and 5.2 mm. Since the variation in diameters is due to random factors and most values are expected to be close to the mean, the normal distribution is appropriate for modelling this scenario.

```
μ = 5  # mean is 5 mm
σ = 0.2 # standard deviation is 0.2 mm
```

```
0.2
```

To calculate the PDF value at $x = 5$:

```
pdf.(Normal(μ, σ), 5)
```

```
1.9947114020071635
```

Probability that a bolt has a diameter less than 5.2 mm:

```
cdf.(Normal(μ, σ), 5.2)
```

```
0.8413447460685433
```

Probability that the diameter is between 4.8 mm and 5.2 mm:

```
cdf.(Normal(μ, σ), 5.2) - cdf.(Normal(μ, σ), 4.8)
```

```
0.6826894921370865
```

The 90^{th} percentile of bolt diameters:

```
quantile.(Normal(μ, σ), 0.9)
```

```
5.25631031310892
```

Generate 10 random diameter values from this normal distribution:

```
rand(Normal(μ, σ), 10)
```

```
10-element Vector{Float64}:
 4.76986521274494
 4.873160591762204
 5.215170808748775
 4.852024608258627
 4.8903038439454445
 4.873004318486938
 5.499321425842579
 4.491204734074525
 4.9583861465095405
 5.006790193941598
```

Probability Distributions in Julia

Exponential Distribution

The exponential distribution describes the time between events that happen randomly but at a steady average rate. It's often used in situations like waiting times. For example, the time you wait for the next bus, phone call, or customer to arrive. The PDF is:

$$f(x \mid \lambda) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}; \lambda > 0$$

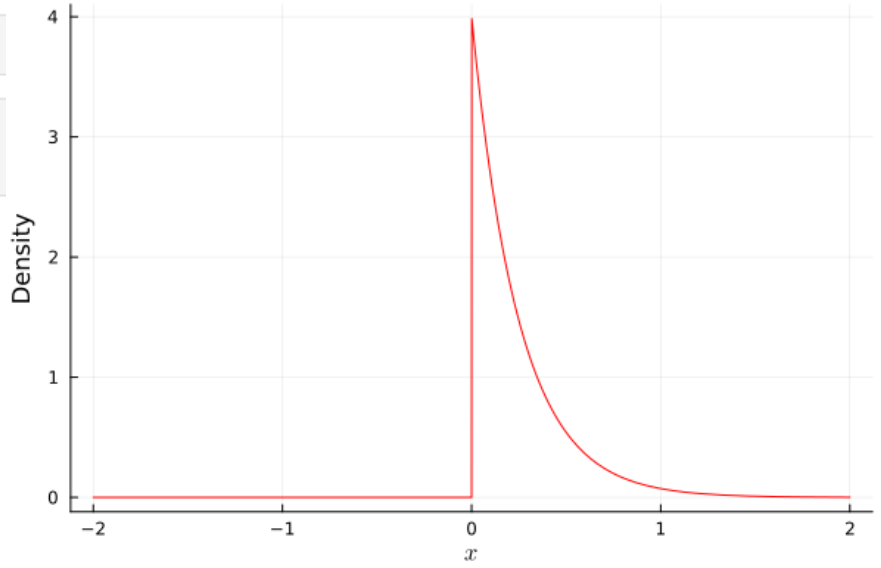
where $\lambda > 0$ is the rate parameter (the reciprocal of the mean), and x represents the time until the next event.

```
using Distributions, Plots, LaTeXStrings
```

```
λ = 4
plot(x->pdf(Exponential(1/λ), x), -2, 2, color = "red", xlabel = L"x", ylabel = "Density", label = "")
```

Rate parameter

Represents the time until the next event.



Probability Distributions in Julia

Exponential Distribution

Suppose that in a certain desert region, it rains once every 10 days on average. We want to analyse how long it will take until the next rain.

```
using Distributions
```

```
λ = 1 / 10  # Since the average time between rains is 10 days
```

```
0.1
```

The probability density that it rains exactly after 5 days:

```
pdf(Exponential(1/λ), 5)
```

```
0.06065306597126335
```

The probability that it rains within the next 7 days:

```
cdf(Exponential(1/λ), 7)
```

```
0.5034146962085905
```

After how many days is there an 80% chance that it has rained?

```
quantile(Exponential(1/λ), 0.8)
```

```
16.094379124341003
```

Simulate waiting times (in days) for the next 5 rainfalls:

```
rand(Exponential(1/λ), 5)
```

```
5-element Vector{Float64}:
  6.131999312828087
  5.821836897765856
 15.417475096239821
 20.013299626399043
  5.777582874532662
```

Probability Distributions in Julia

Gamma Distribution

The Gamma distribution is often used to model the amount of time or quantity needed for several events to occur, especially when these events happen at a steady average rate (in a Poisson process). It is useful when we want to understand how long it takes for something to accumulate like rainfall, sunlight, or nutrient intake, and is especially helpful in environmental and ecological studies where such quantities are naturally skewed (not symmetric). The PDF is:

$$f(x \mid \alpha, \beta) = \begin{cases} \dfrac{\beta^{\alpha} x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)}, & x \geq 0,\ \alpha > 0,\ \beta > 0 \\ 0, & \text{otherwise} \end{cases}$$

where

- $\alpha > 0$ is the **shape parameter**,
- $\beta > 0$ is the **rate parameter**,
- $\Gamma(\alpha)$ is the **Gamma function**.

Probability Distributions in Julia

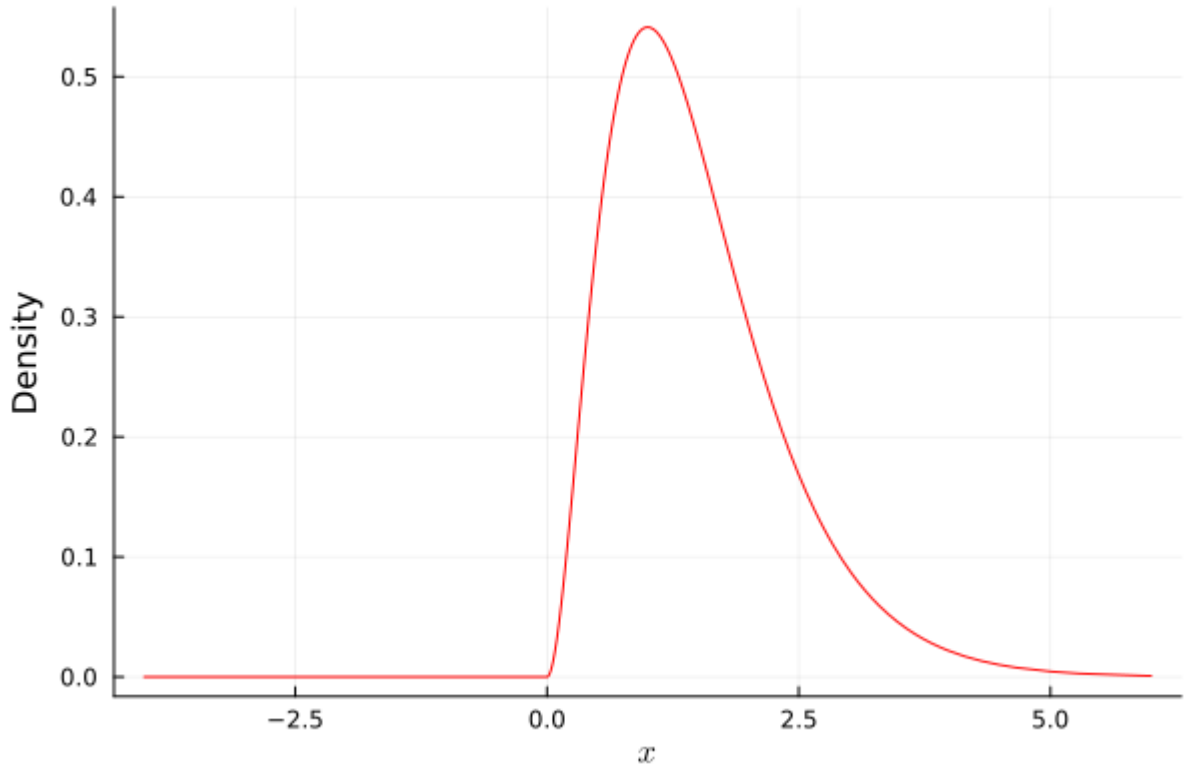
Gamma Distribution

```
using Distributions, Plots, LaTeXStrings
```

```
α = 3
β = 2
plot(x->pdf(Gamma(α, 1/β), x), -4, 6, color = "red", xlabel = L"x", ylabel = "Density", label="")
```

Shape parameter

Rate parameter



Probability Distributions in Julia

Gamma Distribution

Suppose an ecologist is studying a plant species that requires a cumulative rainfall of at least 30 mm before sprouting. Based on past data, the ecologist assumes that rainfall per storm follows a Gamma distribution with shape parameter $\alpha = 6$ and rate parameter $\beta = 0.2$ (mean rainfall per storm is $\frac{\alpha}{\beta} = 30\ mm$

```
using Distributions, Plots
```

```
α = 6
β = 0.2
```

```
0.2
```

Probability Density at a Specific Rainfall, say at 20mm

```
pdf(Gamma(α, 1/β), 20)
```

```
0.03125869037010633
```

Cumulative Probability for Rainfall Up to 30 mm:

```
cdf(Gamma(α, 1/β), 30)
```

```
0.5543203586353888
```

To find the rainfall value corresponding to the 90^{th} percentile:

```
quantile(Gamma(α, 1/β), 0.9)
```

```
46.373369466758106
```

To simulate five random rainfall values based on the Gamma distribution:

```
rand(Gamma(α, 1/β), 5)
```

```
5-element Vector{Float64}:
 44.178439761084256
 31.044173993892404
 41.089224438533755
 35.873682614430415
 24.453171610728592
```

Probability Distributions in Julia

Gamma Distribution

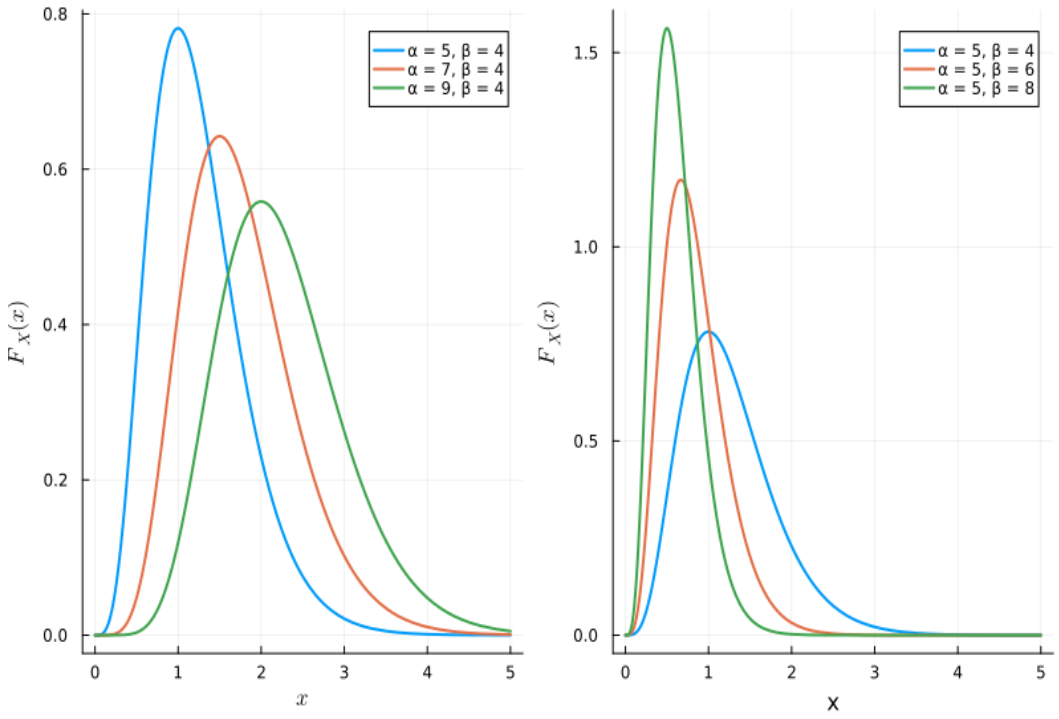
The Gamma distribution exhibits notable changes in shape depending on the values of the shape parameter α and the rate parameter β. As α increases, the distribution becomes more symmetric and its peak shifts to the right. In contrast, increasing β leads to a steeper decline and greater concentration near zero, due to its inverse relationship with the scale.

```
using Plots, LaTeXStrings, Distributions
```

```
α_vals = [5, 7, 9] # vary the α values
β= 4
p1 = plot(xlabel=L"x", ylabel=L"F_X(x)")
for α in α_vals
    plot!(x -> pdf(Gamma(α, 1/β), x),0, 5, lw = 2, label = "α = $α, β = $β")
end

α = 5
β_vals = [4, 6, 8] # varying θ values
p2 = plot(xlabel="x", ylabel=L"F_X(x)")
for β in β_vals
    plot!(x -> pdf(Gamma(α, 1/β), x),0, 5, lw = 2, label = "α = $α, β = $β")
end

plot(p1, p2, layout = (1, 2), size =(800, 500))
```



Comparison of Execution Time in R and Julia

To illustrate this, we consider an example of multiple linear regression.

R Code

```r
library(microbenchmark)
data = mtcars
attach(data)
microbenchmark(lm(mpg~cyl + hp + wt + drat))
```

Julia Code

```julia
using RDatasets, CRRao, BenchmarkTools, StatsModels
df = dataset("datasets", "mtcars")
@benchmark fit(@formula(MPG ~ HP + WT + Cyl + DRat), df, LinearRegression())
```

Language	Pkg/Function	Mean time taken
R	Stats/lm	478.348 μs
Julia	CRRao/fit	119.965 μs

Symbolic Computation in Julia

Julia provides a powerful symbolic computation environment for performing algebraic manipulations and analytical calculations. This functionality is supported through the **Symbolics.jl** package, where symbolic variables are defined using @variables and differentiation can be carried out using the derivatives function. In addition to Symbolics.jl, other packages such as **SymPy.jl** and **Reduce.jl** are also available in Julia for symbolic matematics and computer algebra operations.. With this approach, we can compute exact symbolic derivatives of functions such as $f_s(x) = \frac{2x}{3+x}$ and $g_s(x) = \frac{2x^2}{1+x^2}$

```
using Symbolics
@variables x
f_s = 2*x/(3+x)
```

$$\frac{2x}{3+x}$$

```
df_s = Symbolics.derivative(f_s, x) # derivative w.r.t x
```

$$\frac{-2x}{(3+x)^2} + \frac{2}{3+x}$$

```
using Symbolics
@variables x
g_s =  ((2*x^2)/(1+x^2))
```

$$\frac{2x^2}{1+x^2}$$

```
dg_s = Symbolics.derivative(g_s, x) # derivative w.r.t x
```

$$\frac{4x}{1+x^2} - 2x\frac{2x^2}{(1+x^2)^2}$$

Simple Linear Regression

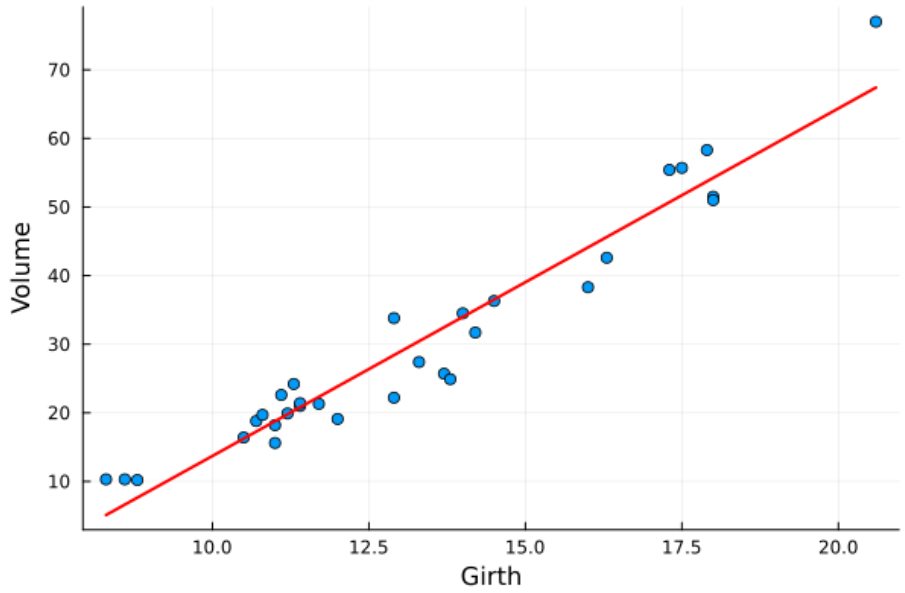
To demonstrate simple linear regression, we consider the **trees** dataset provided with **R**. In Julia, the same dataset c be imported using the **RDatasets.jl** package, enabling users to perform regression analysis in Julia using a dataset familiar from R. This dataset provides measurements of the diameter, height, and volume of timber in 31 felled black cherry trees. Note that the diameter (in inches) is erroneously labeled Girth in the data. Suppose we aim to predict the Volume of the timber using the tree diameter. Therefore, we consider the following statistical model

$$Volume = \beta_0 + \beta_1 Girth \ + \epsilon$$

In the above expression, Volume is the response variable, and Girth is the predictor variable. β_0 and β_1 are the intercept and slope parameters of the population regression line, respectively. The ϵ represents the random component that cannot be explained through the linear relationship between the response and predictor. We also assume that $\epsilon \sim N(0.\sigma^2)$

Simple Linear Regression

```
using Plots, Statistics, StatsBase
using LaTeXStrings, StatsModels, StatsPlots
using RDatasets, DataFrames, GLM
data = dataset("datasets", "trees")
first(data, 6) # first 6 observations
describe(data)  # summary of the data
size(data)  # dimensions of the data
completecases(data)  # to check whether is there any missing value
last(data, 6) # last 6 observations
names(data)  # variable names
lm_fit = lm(@formula(Volume~Girth), data)
predicted_vals = GLM.predict(lm_fit)
p1 = Plots.scatter(data.Girth, data.Volume, xlabel = "Girth",
    ylabel = "Volume", label = "")
Plots.plot!(data.Girth, predicted_vals, color = "red", lw = 2, label = "")
```




```
Volume ~ 1 + Girth

Coefficients:
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	-36.9435	3.36514	-10.98	<1e-11	-43.826	-30.061
Girth	5.06586	0.247377	20.48	<1e-18	4.55991	5.5718

Conclusion

- Julia has emerged as a **powerful scientific computing** language, effectively combining high-level expressiveness with performance comparable to low-level programming languages.
- Its unified programming environment allows researchers to develop, test, and deploy computational models efficiently without switching between multiple languages.
- Support for abstraction, metaprogramming, and a growing **scientific ecosystem** enables applications in simulation, optimization, machine learning, and data-intensive research.
- Interoperability with Python, R, C, and Fortran ensures seamless integration with existing workflows and facilitates adoption within established scientific communities.
- While **standardized documentation** remains limited, ongoing development and strong community support position Julia as a key platform for **modern scientific computing, statistical computing & data science**.

Acknowledgement

I express my sincere gratitude to **Dr. Amiya Ranjan Bhowmick (Department of Mathematics, ICT Mumbai)** for his invaluable mentorship and guidance. I consider myself truly fortunate to have been receiving his guidance for the past three years.

Special thanks to **Dr. Ajit Kumar (Department of Mathematics, ICT Mumbai)**, **Dr. Md Aktar UL Karim (Symbiosis Statistical Institute, Pune)**, **Dr. Dipali Vasudev Mestry (Department of Mathematics, ICT Mumbai)**, and **Dr. Dipak D. Pukale (University of Akron, USA)** for their encouragement and motivation.

References:

- Nagar, S., Nagar, & Anglin. (2017). *Beginning Julia Programming*. Springer Science+ Business Media.
- Dash, S. K. (2021). *Hands-On Julia Programming: An Authoritative Guide to the Production-Ready Systems in Julia (English Edition)*. BPB Publications.
- McNicholas, P. D., & Tait, P. (2019). *Data science with Julia*. Chapman and Hall/CRC.
- Phillips, L. (2023). *Practical Julia: A Hands-on Introduction for Scientific Minds*. No Starch Press
- Chaugule, S. S., & Bhowmick, A. R. (2025). *A gentle introduction to statistical computing using Julia* (Version 2025). GitHub. https://github.com/sujit016/A-Gentle-Introduction-to-Statistical-Computing-Using-Julia
- Verzani, J. (2025, August 29). *Calculus with Julia*. https://jverzani.github.io/CalculusWithJuliaNotes.jl/
- Nazarathy, Y., & Klok, H. (2021). *Statistics with Julia: Fundamentals for data science, machine learning and artificial intelligence*. Springer Nature

Thank You