Growth Curve Modelling using Julia

Sujit Sandipan Chaugule[1], Dr. Amiya Ranjan Bhowmick[2]

[1]Department of Pharmaceutical Sciences and Technology, Institute of Chemical Technology, Mumbai

[2]Department of Mathematics, Institute of Chemical Technology, Mumbai

1. Growth Curve Models

Scientific formalization of the notion of growth in living organisms is an age-old problem. For a long time, it has been a challenging issue for scientists to develop appropriate measures of growth or the rate of growth of living organisms. Such measures are receiving renewed importance in applied sciences, e.g., in zoology, botany, ecology, population dynamics, demography, cell dynamics, bacterial growth, finance, etc. The sigmoid functions, Gompertz, General Logistic, and General von Bertalanffy and their associated differential equations have applications to model self-limited population growth in diverse fields, e.g., sociology, fish growth, plant growth, and tumor growth. Particularly in the fisheries literature, much has been discussed on models using von Bertalanffy growth law, including criticisms, testing for parameter differences, bioenergetic applications, and re-parameterizations. There have also been theoretical approaches to defining a general framework to study growth models and a new family of sigmoid growth functions has been introduced, namely, Trans-General Logistic, Trans-General von Bertalanffy, and Trans-Gompertz. Growth curve models are increasingly used in several areas of interdisciplinary research. For example, growth models play an important role in modeling the density regulation in the abundance of natural populations. The growth models such as logistic, theta-logistic, Gompertz, etc. have potential applications in population dynamics that may be used for predictions and forecasting extinctions. The following references are very helpful

- **Tsoularis, A. and Wallace, J. 2002. Analysis of Logistic growth Models. Mathematical Biosciences, 179, 21-55**
- **Seber, G. A. F. and Wild, C. J. 2003. Nonlinear Regression. John Wiley and Sons, Inc**

In this session, some common growth curve models will be studied, which have immense applications across disciplines. Julia code will be included between the concepts of growth models. In this section, solving differential equations using Julia will also be covered.
- Growth models are usually represented as differential equations that describe the evolution of population over time (that is, temporal evolution). `Such differential equations describe the rate of change in population size.` These growth models are usually developed for modeling the growth of the population under certain assumptions (`usually specified by the domain experts to incorporate biological hypotheses`).
- Once the differential equation has been formed, it is of interest to obtain the size of the population at some time point `t` from the rate equations. `Often, it is not possible to solve the differential equations analytically, in this case, numerical methods play a critical role. Let it be taken care of by the mathematicians.`
- Due to the increasing availability of open-source efficient software, such equations can be solved numerically using them. In this session, Julia will be used as a tool to solve the differential equations describing the growth of populations.
- During the lectures of Dr. Sabyasachi Bhattacharya, various growth equations were seen, ranging from fundamental exponential growth to Allee growth equations. `In this session, the focus will be on learning how Julia can be used to solve those differential equations.`

Appropriate notes and comments will be used wherever required. The following are the growth models that will be discussed:

2. Exponential Growth Curve Model

Let us assume that a simple homogeneous population is growing under the assumption that all the changes in the population's size or density are regulated by births and deaths only. Let the per capita birth rate be b and the per capita death rate be d.
Then, the relative growth rate is given by:

$$\frac{1}{X}\frac{dX}{dt} = b - d$$

This gives an ordinary differential equation with an initial condition $X(0) = X_0$, say. Let:

$$r = b - d$$

r is known as the **intrinsic growth rate** and plays a very important role in population dynamics.

One can easily solve the equation to get the solution for $X(t)$ as a function of t, which is:

$$X(t) = X_0 \cdot \exp(rt)$$

It is easy to see that:

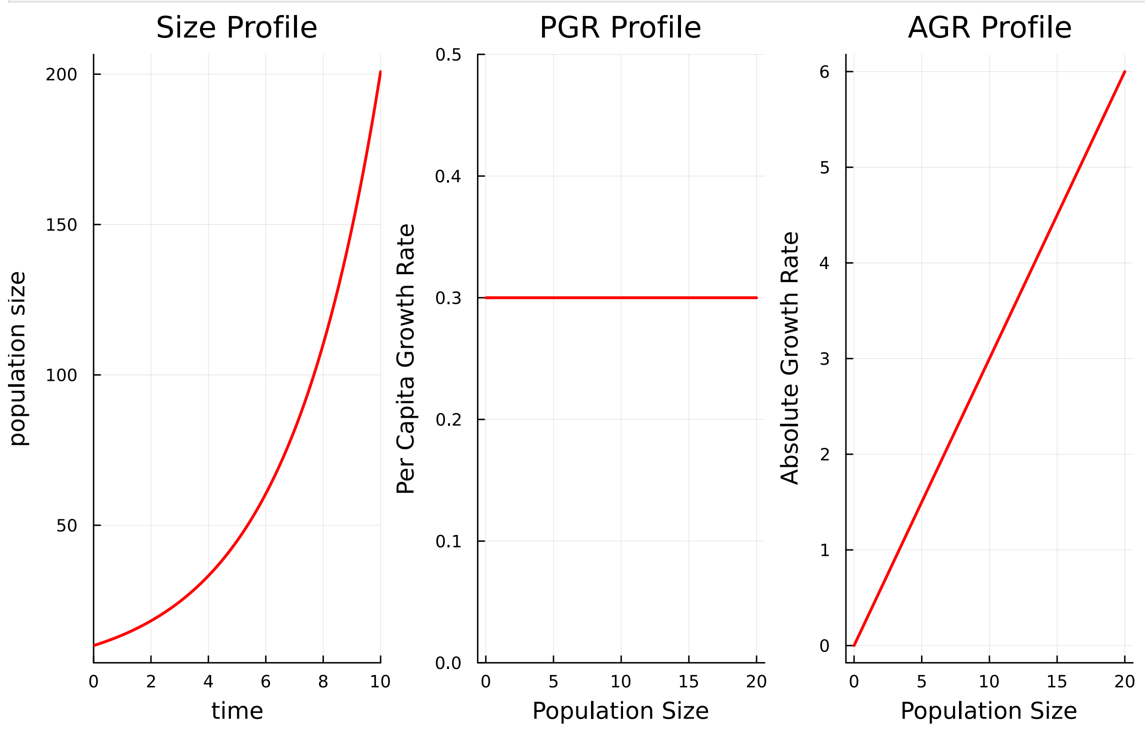
- For $r > 0$, the population explodes as time increases.
- For $r < 0$, the population dies out.
- For $r = 0$, the population size remains constant.

This is an appropriate occasion to introduce how to solve this differential equation `numerically`. The syntax will follow a standard format so that the same approach can be reused for other models.
Here is the code:

In [1]:
```julia
using Plots
using DifferentialEquations
using Statistics
using Distributions
using LaTeXStrings
```

In [2]:
```julia
X = 10.0  # Initial condition
r = 0.3   # Parameter
tspan = (0.0, 10.0)  # Time span

# ODE function
function expFun(dX, X, p, t)
    dX[1] = r * X[1]
end

# ODE Problem
prob = ODEProblem(expFun, [X], tspan)

# Solve using Tsit5   # Tsit5() is a Runge-Kutta method
sol = solve(prob, Tsit5(), saveat = 0.01)

p1 = plot(sol, color = "red", xlabel = "time", lw = 2,ylabel = "population size",
    title = "Size Profile", label = "" )

p2 = plot(x-> 0.3*x^0, 0, 20, color = "red", lw = 2,xlabel = "Population Size",
    ylabel  = "Per Capita Growth Rate",ylims = (0, 0.5) ,title = "PGR Profile", label = "")

p3 = plot(x-> 0.3*x, 0, 20, color = "red", lw = 2,xlabel = "Population Size",
    ylabel  = "Absolute Growth Rate" ,title = "AGR Profile", label = "")

plot(p1, p2, p3, layout = (1,3), size = (800, 500))
```

Out[2]:



3. Logistic Growth Curve Model

Note that, in the exponential model, the growth is governed by constant per capita birth and death rates that lead to unlimited growth. In real-life situations, this is unrealistic. In the natural environment, the populations are always limited by resources that lead to intra-specific competition when the population size becomes large. This concept leads to the density-dependent growth regulation.
Let us assume that the per capita growth rate linearly decreases with population size and reaches zero at the `carrying capacity`, commonly denoted by K. This leads to the logistic equation given by:

$$\frac{1}{X}\frac{dX}{dt} = r\left(1 - \frac{X}{K}\right)$$

As usual, this ordinary autonomous differential equation is supported by the initial condition:

$$X(0) = X_0$$

The exact solution is given by:

$$X_t = \frac{K}{1 + \left(\frac{K}{X_0} - 1\right)e^{-rt}}$$

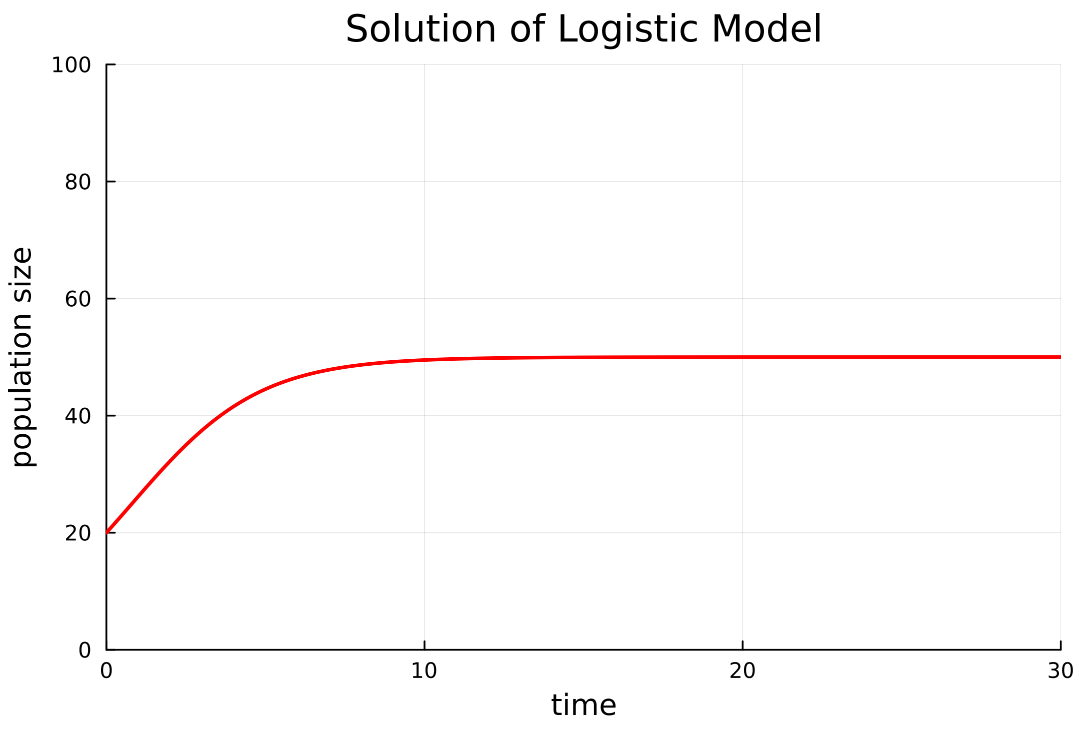
It can be easily seen that as $t \to \infty$, the population size stabilizes to the carrying capacity K.

In [3]:
```
using Plots
using DifferentialEquations
using Statistics
using Distributions
```

In [4]:
```
X = 20.0              # Initial population size
r = 0.5               # Growth rate
K = 50.0              # Carrying capacity
tspan = (0.0, 30.0)   # Time span
```

```
# Logistic growth function
function logisticFun(dX, X, p, t)
    dX[1] = r * X[1] * (1 - X[1]/K)
end

# Define the ODE problem
prob = ODEProblem(logisticFun, [X], tspan)

# Solve using Tsit5 method
sol = solve(prob, Tsit5(), saveat = 0.01)

plot(sol, color = "red", lw = 2,xlabel = "time", ylabel = "population size",
    title = "Solution of Logistic Model", ylims = (0, 100),label = "")
```

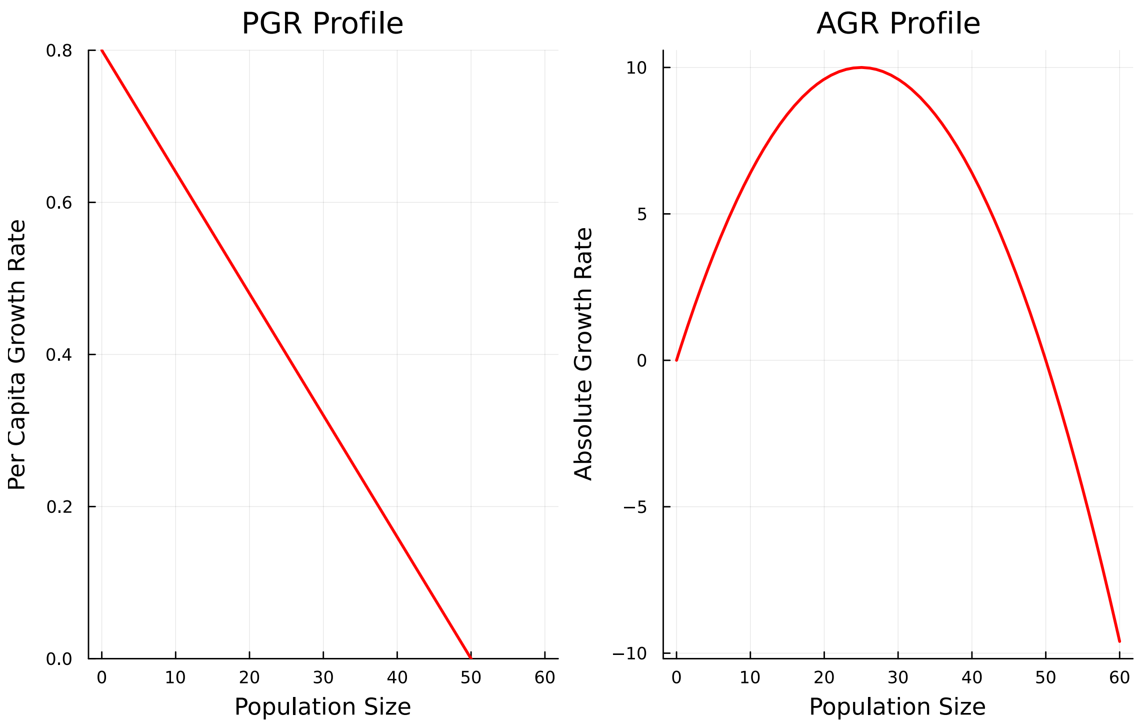
Out[4]:



In the following code, we generate multiple solutions of the logistic differential equation for different initial conditions. Run the code and note that all the solutions are converging to carrying capacity K =50, in the program

In [5]:
```
r = 0.8
K = 50

# Per Capita Growth Rate = r * (1 - x/K)
p1 = plot(x -> r * (1 - x/K), 0, 60, color = "red", lw = 2,xlabel = "Population Size",
    ylabel = "Per Capita Growth Rate", ylims = (0, r), title = "PGR Profile", label = "")

# Absolute Growth Rate = r * X * (1 - X/K)
p2 = plot(x -> r * x * (1 - x/K), 0, 60, color = "red", lw = 2 ,xlabel = "Population Size",
    ylabel = "Absolute Growth Rate", title = "AGR Profile", label = "")

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

Out[5]:



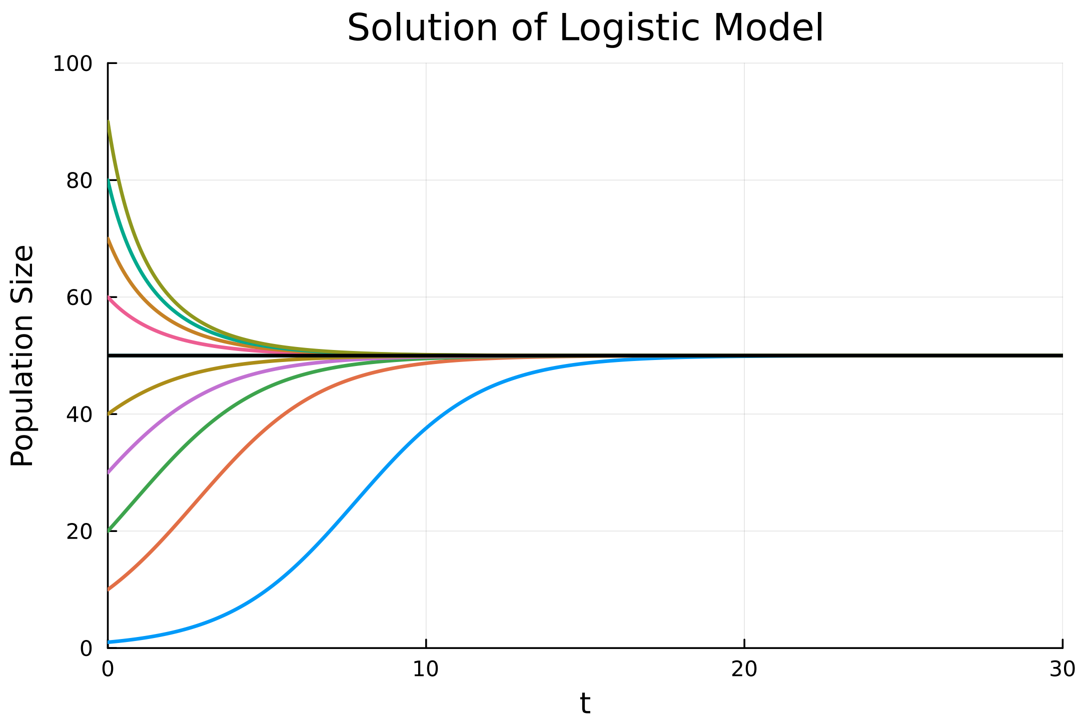
One of the very important concepts in growth curve analysis is the `stability of the equilibrium point`. An equilibrium point refers to the population size where there is no growth (mathematically, $\frac{dX_t}{dt} = 0$). For example, in the logistic growth equation, there are two equilibrium points: one at `zero` and the other at K (the carrying capacity). A technique known as `linear stability analysis` can be used to assess the stability of these equilibrium points.

In a logistic model, the population eventually converges to the carrying capacity of the environment. This can be illustrated through a computer simulation by solving the model for different initial conditions and plotting the solution curves to observe convergence behavior. In the following figure, all solution trajectories approach the same carrying capacity, K.

In [6]:
```
using Plots
using DifferentialEquations
using Statistics
using Distributions
```

In [7]:
```
init_pop = [1,10,20,30,40,50,60,70,80,90]
r = 0.5                # Growth rate
K = 50.0               # Carrying capacity
tspan = (0.0, 30.0)    # Time span

# Logistic growth function
function logisticFun(dX, X, p, t)
    dX[1] = r * X[1] * (1 - X[1]/K)
end

for i in 1:length(init_pop)
    X = init_pop[i]
    prob = ODEProblem(logisticFun, [X], tspan)
    sol = solve(prob, Tsit5(), saveat = 0.01)
    if i == 1
        p1 = plot(sol, color = i, lw = 2, ylims = (0, 100), xlabel = "Time",
                   ylabel = "Population Size", label = "",
                   title = "Solution of Logistic Model")
    else
        plot!(sol, color = i, lw = 2, label = "")
    end
end
```

```
hline!([50], lw = 2, color = "black", label = "")
display(p1)
```



4. Gompertz Growth Curve Model

The Gompertz model describes the dynamics of a population that grows with an intrinsic rate of growth that decays exponentially. The equation is commonly written in the non-autonomous differential equation form as:

$$\frac{dX}{dt} = r \cdot \exp(-\alpha t) X$$

The equation can also be written as an autonomous differential equation:

$$\frac{dX}{dt} = \alpha X \ln\left(\frac{K}{X}\right)$$

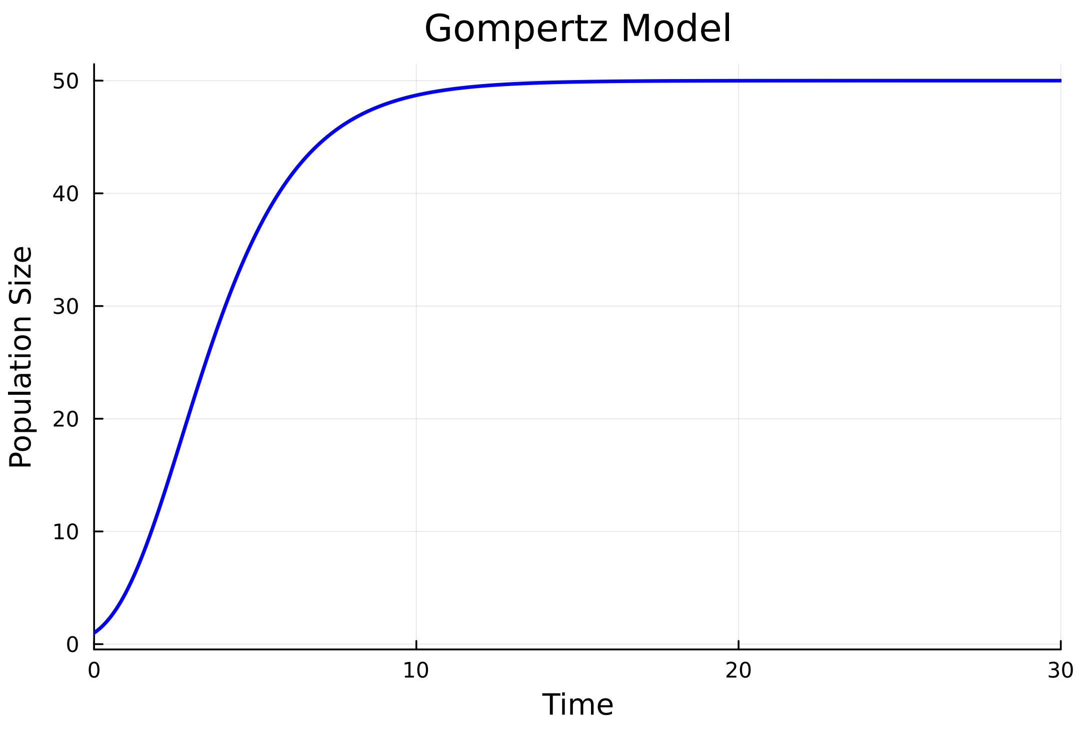
where K is the carrying capacity. Here is the code:

In [8]:
```
using Plots
using DifferentialEquations
using Statistics
using Distributions
```

In [9]:
```
using DifferentialEquations, Plots

X = 1.0         # Initial condition
α = 0.5         # Growth rate parameter
K = 50.0        # Carrying capacity
tspan = (0.0, 30.0)  # Time span

# Gompertz growth function
function GompertzFun(dX, X, p, t)
    dX[1] = α * X[1] * log(K / X[1])
end

# Define the ODE problem
prob = ODEProblem(GompertzFun, [X], tspan)
```

```
# Solve using Tsit5 method
sol = solve(prob, Tsit5(), saveat = 0.01)

# Plotting
plot(sol, xlabel = "Time", ylabel = "Population Size",
     title = "Gompertz Model", lw = 2, color = :blue, label = "")
```

Out[9]:

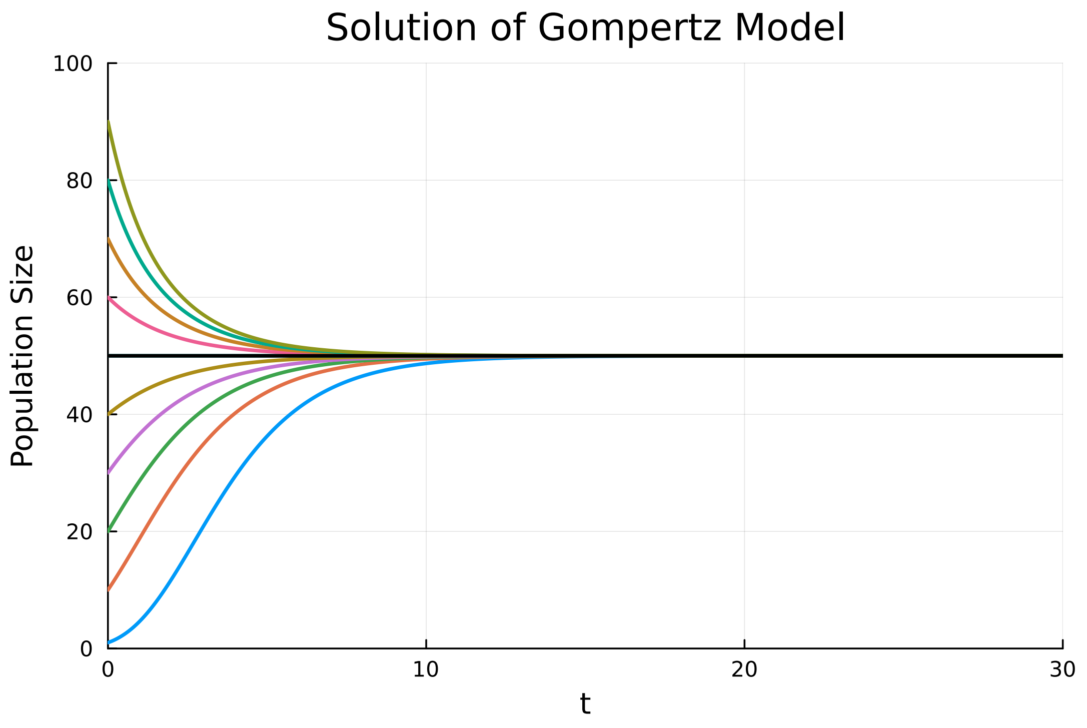


Code for multiple solutions for different initial popualtion size

In [10]:

```
init_pop = [1,10,20,30,40,50,60,70,80,90]

X = 1.0        # Initial condition
α = 0.5        # Growth rate parameter
K = 50.0       # Carrying capacity
tspan = (0.0, 30.0)  # Time span

# Gompertz growth function
function GompertzFun(dX, X, p, t)
    dX[1] = α * X[1] * log(K / X[1])
end

for i in 1:length(init_pop)
    X = init_pop[i]
    prob = ODEProblem(GompertzFun, [X], tspan)
    sol = solve(prob, Tsit5(), saveat = 0.01)
    if i == 1
        p1 = plot(sol, color = i, lw = 2, ylims = (0, 100), xlabel = "t",
                   ylabel = "Population Size", label = "",
                   title = "Solution of Gompertz Model")
    else
        plot!(sol, color = i, lw = 2, label = "")
    end
end

hline!([50], lw = 2, color = "black", label = "")
```

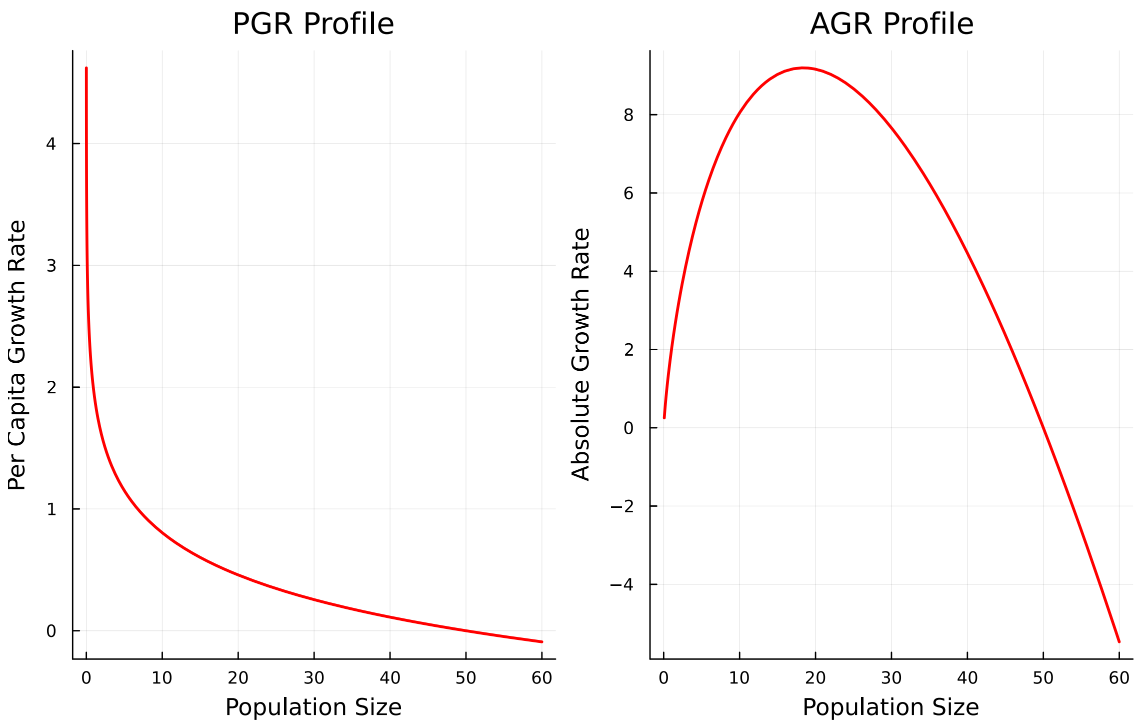
Out[10]:



In [11]:

```julia
# plots of PGR Profile and AGR Profile
α = 0.5
K = 50
# Per Capita Growth Rate =  α*log(K/x)
p1 = plot(x -> α*log(K/x), 0, 60, color = "red", lw = 2,xlabel = "Population Size",
    ylabel = "Per Capita Growth Rate", title = "PGR Profile", label = "")

# Absolute Growth Rate =  α*x*log(K/x)
p2 = plot(x ->α*x*log(K/x)  , 0, 60, color = "red", lw = 2 ,xlabel = "Population Size",
    ylabel = "Absolute Growth Rate", title = "AGR Profile", label = "")

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

Out[11]:



The evolution of Gompertz equation came from a very important biological assumption. Under the exponential growth law, the intrinsic growth rate of the population (r) is assumed to be constant over time. However, it might happen that the intrinsic growth rate is decaying over time at an exponential rate:

$$r(t) = r_0 \exp(-\alpha t)$$

where r_0 is a constant and α is the decaying rate.

A critical role of growth curve models: For example, the Gompertz growth curve model and the logistic growth curve model both look similar. However, they have very different intrinsic properties. For example, given any initial population size, the logistic growth equation will give the population to converge at carrying capacity only. However, if we change the initial population size, the Gompertz model may converge to a different capacity.
So, starting with multiple models and statistical selection of the best model may not always give rise to a biologically correct model. It is a tendency for many statisticians to compare models and give the best model to biologists.
`In such a scenario, the statistical model selection through ISRP (Interval Specific Rate Parameter) would give a higher chance of correct model identification (Bhowmick et al., 2014, Journal of Biological Physics). In fact, the probability of correct identification of model is better than the Fisher growth model (Pal et al., 2018, Journal of Theoretical Biology).`

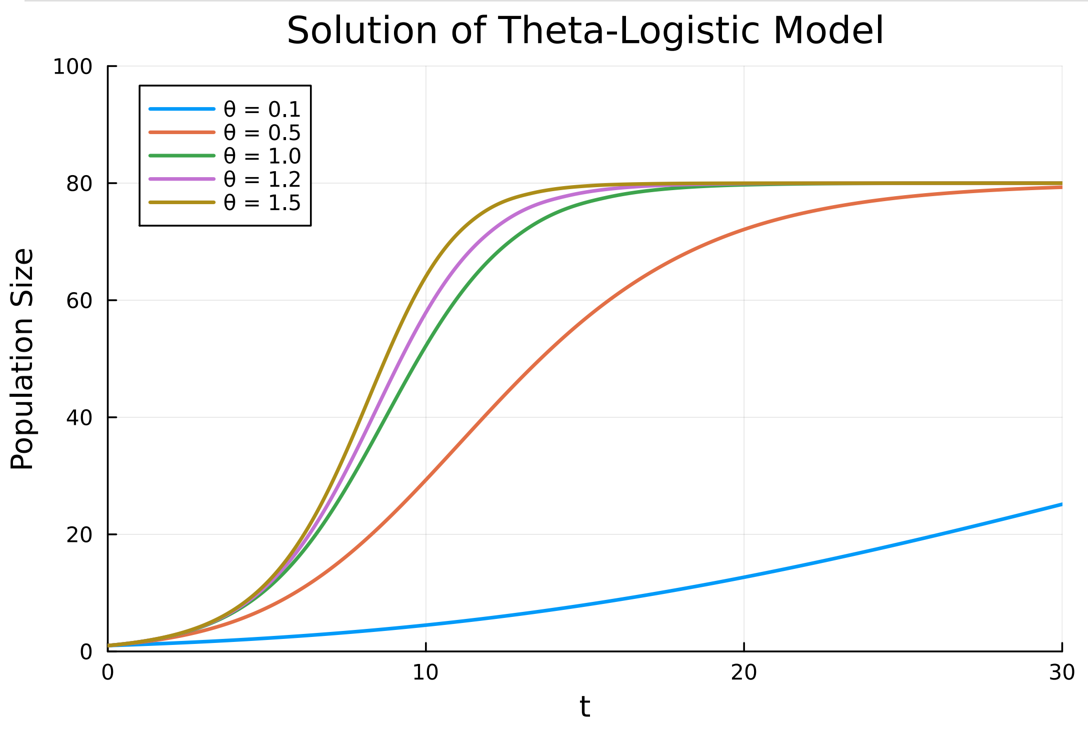
4. Theta-Logistic Growth Curve Model

The θ-logistic model (Gilpin and Ayala, 1973; Gilpin et al., 1976) defines a general class of models of density regulation as the function of only a single parameter θ that regulates the density-dependent mechanism. In this model framework, the population is assumed to have maximum fitness when they are small in numbers. As population size increases, per capita growth rate decreases monotonically due to intra-specific competition being the main regulatory mechanism (unlike the Allee model).
The θ-logistic model is given by:

$$\frac{dX_t}{dt} = rX_t\left(1 - \left(\frac{X_t}{K}\right)^\theta\right)$$

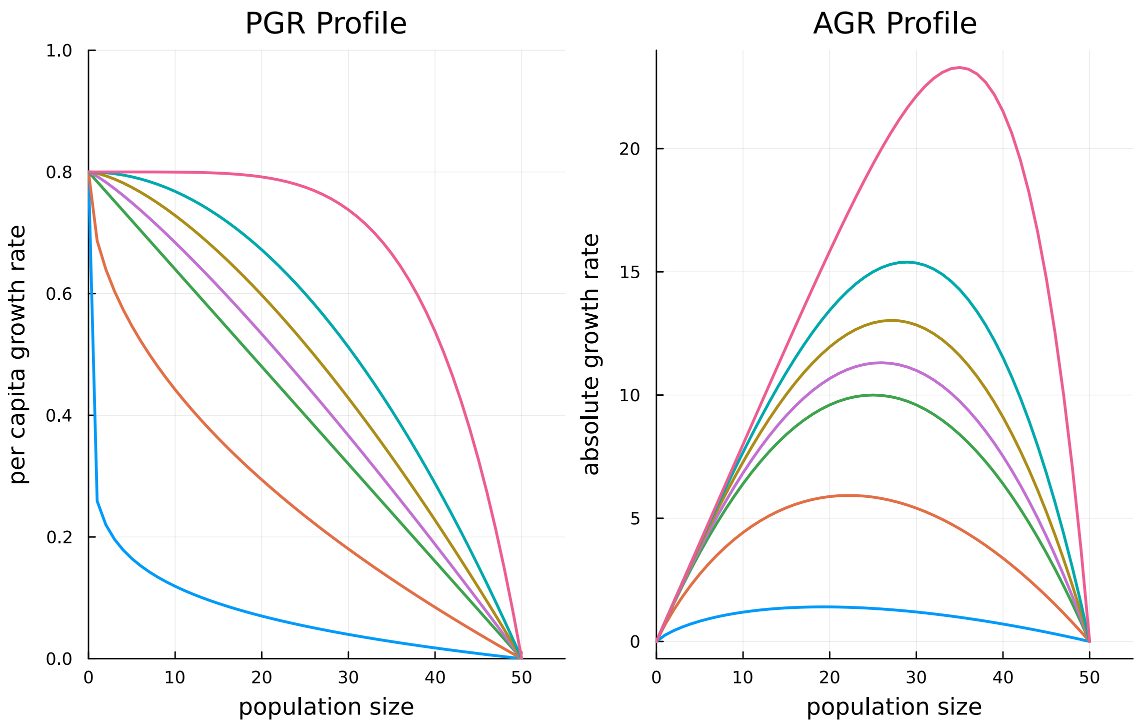
In [12]:
```julia
using Plots
using DifferentialEquations
using Statistics
using Distributions
```

In [13]:
```julia
θ_vals = [0.1, 0.5, 1, 1.2, 1.5]
X = 1  # initial condition
r = 0.5
K = 80.0
tspan = (0.0, 30.0)

function thetalogisticFun(dX, X, p, t)
    θ = p
    dX[1] = r * X[1] * (1 - (X[1] / K)^θ)
end

for i in 1:length(θ_vals)
    θ = θ_vals[i]
    prob = ODEProblem(thetalogisticFun, [X], tspan, θ)
    sol = solve(prob, Tsit5(), saveat = 0.01)
    if i == 1
        p1 = plot(sol, color = i, lw = 2, ylims = (0, 100), xlabel = "t",
                  ylabel = "Population Size", label = "θ = $θ",
                  title = "Solution of Theta-Logistic Model")
    else
        plot!(sol, color = i, lw = 2, label = "θ = $θ")
    end
end

display(p1)
```



In [14]:
```
# Per-capita growth profile for different values of theta
# Absoulute Growth Rate Profile for different values of theta
r = 0.8
K = 50
θ_vals = [0.1, 0.5, 1, 1.2, 1.5, 2, 5]
X = collect(0:1:50)
PGR = Matrix{Float64}(undef, length(θ_vals), length(X))
AGR = Matrix{Float64}(undef, length(θ_vals), length(X))

for i in 1:length(θ_vals)
    PGR[i,:] .=  r .*(1 .-(X ./K).^(θ_vals[i]))
end


for i in 1:length(θ_vals)
    AGR[i,:] .=  r*X .*(1 .-(X //K) .^(θ_vals[i]))
end

p1 = plot(X, PGR', lw = 2, xlims = (0, 55) ,ylims = (0,1), xlabel = "population size",
    ylabel = "per capita growth rate", title = "PGR Profile",label = "")

p2 = plot(X, AGR', lw = 2, xlims = (0, 55), xlabel = "population size",
    ylabel = "absolute growth rate", title = "AGR Profile",label = "")

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

Out[14]:



4.1. Few important notes from the Clark et al. (2010)

- **Note 1**: Concave r–N curves are typical of so-called 'r-selected' organisms where density dependence acts strongly at lower densities, whereas convex curves arise from 'K-selected' species where density dependence acts to reduce growth only at higher densities (although these terms have fallen out of favour).
- **Note 2**: The growth response is parameterized jointly by r_m and θ. For a population growing from low abundance, the parameter h reflects those aspects of a species' evolved life history (demographic rates) that determine how abruptly growth slows as abundance interacts with resource availability (Freckleton et al. 2003; Sibly et al. 2005) and type of competition (Johnston, Berryman & Lima 2008).
- **Note 3**: When concave ($\theta < 1$), the growth response (i.e., the 'r–N curve') ideally characterizes a population unable to recover quickly from extrinsic perturbations, whereas a convex growth response ($\theta > 1$) implies that density feedback occurs mainly above some (relatively large) threshold abundance (Fowler 1981; Owen-Smith 2006), typical, for instance, of the population dynamics in large mammals (McCullough 1999; Owen-Smith 2006).

5. Allee Growth Curve Equations

The Allee effect, named after ecologist W. C. Allee, corresponds to density-mediated drop in population fitness when they are small in numbers (Allee, 1931; Dennis, 1989; Fowler and Baker, 1991; Stephens and Sutherland, 1999). The harmful effects of inbreeding depression, mate limitation, predator satiation etc. reduce fitness as the population size decreases. For such dynamics, the maximum fitness is achieved by the species at an intermediate population size, unlike logistic or theta-logistic growth models. Such observations usually correspond to the mechanisms giving rise to an Allee effect. In recent decades, due to an increasing number of threatened and endangered species, the Allee effect has received much attention from conservation biologists. The related theoretical consequences and the empirical evidence have made the Allee effect an important component in both theoretical and applied ecology. In general, there are two types of Allee effects considered in the natural populations across a variety of taxonomic groups, viz. component and demographic Allee effect.

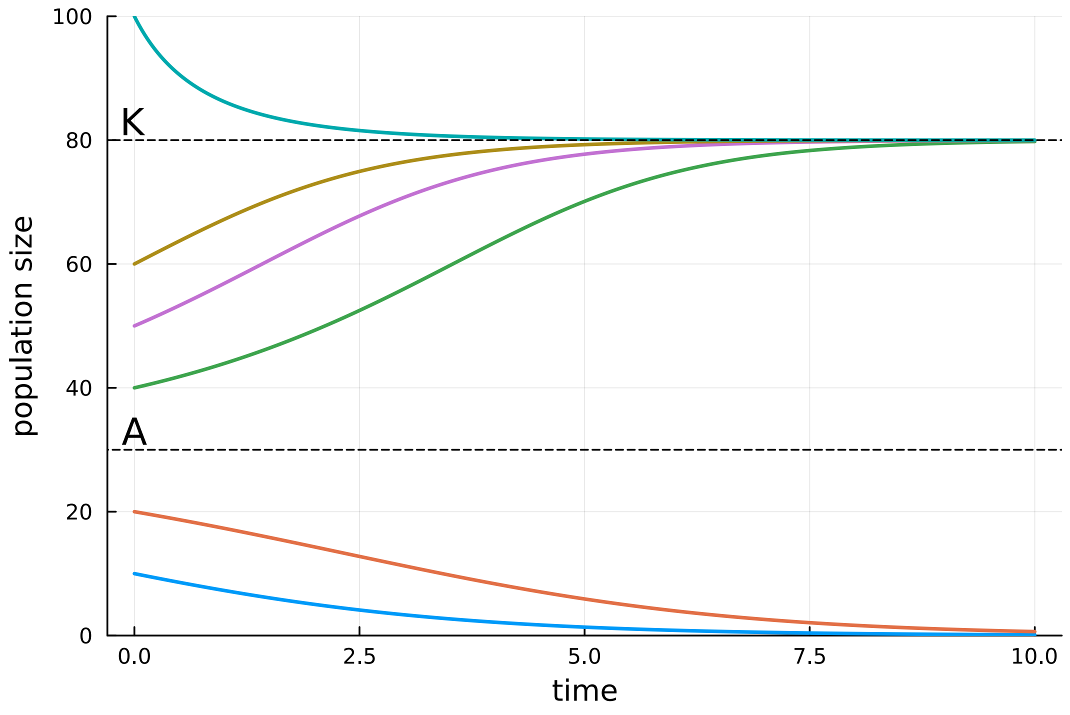
- The **component Allee effect** modifies some component of individual fitness with the changes in population sizes or density.
- The **demographic Allee effect**: If the per capita growth rate is low at small density but remains positive, it is called the *weak demographic Allee effect*.
- The *strong Allee effect* is characterized by a threshold density below which the per capita growth rate is negative, leading to deterministic extinction.

The critical density is called the **Allee threshold** and has significant importance in conservation biology (Hackney and McGraw, 2001), population management (Myers et al., 1995; Liermann and Hilborn, 1997) and invasion control (Johnson et al., 2006).
There are a large number of studies available in the literature on the mathematical modeling of the demographic Allee effect (see Table 3.1, Courchamp et al. (2008)).

In [15]:
```julia
using Plots
using DifferentialEquations
using Statistics
using Distributions
```

In [16]:
```julia
using DifferentialEquations, Plots

# Parameters
A = 30.0                             # Threshold population size
K = 80.0                             # Carrying capacity of the environment
init_pop = [10.0, 20.0, 40.0, 50.0, 60.0, 100.0]  # Different initial populations
r = 0.5

# Allee model function
function AlleeFun!(dX, X, p, t)
    dX[1] = r * X[1] * (X[1]/A - 1) * (1 - X[1]/K)
end

# Time span
times = 0.0:0.01:10.0
tspan = (0.0, 10.0)

for i in 1:length(init_pop)
    state = [init_pop[i]]
    prob = ODEProblem(AlleeFun!, state, tspan)
    sol = solve(prob, Tsit5(), saveat=times)

    if i == 1
        p1 = plot(sol.t, sol[1,:],
             main = "Solution of Allee growth equation",
             xlabel = "time", ylabel = "population size",
             ylim = (0, 100), lw = 2, color = i, label = "")
    else
        plot!(sol.t, sol[1,:], lw = 2, color = i, label = "")
    end
end

hline!([K], color = "black", linestyle = :dash, label = "")
hline!([A], color = "black", linestyle = :dash, label = "")
annotate!([0],[83], "K")
annotate!([0],[33], "A")
```

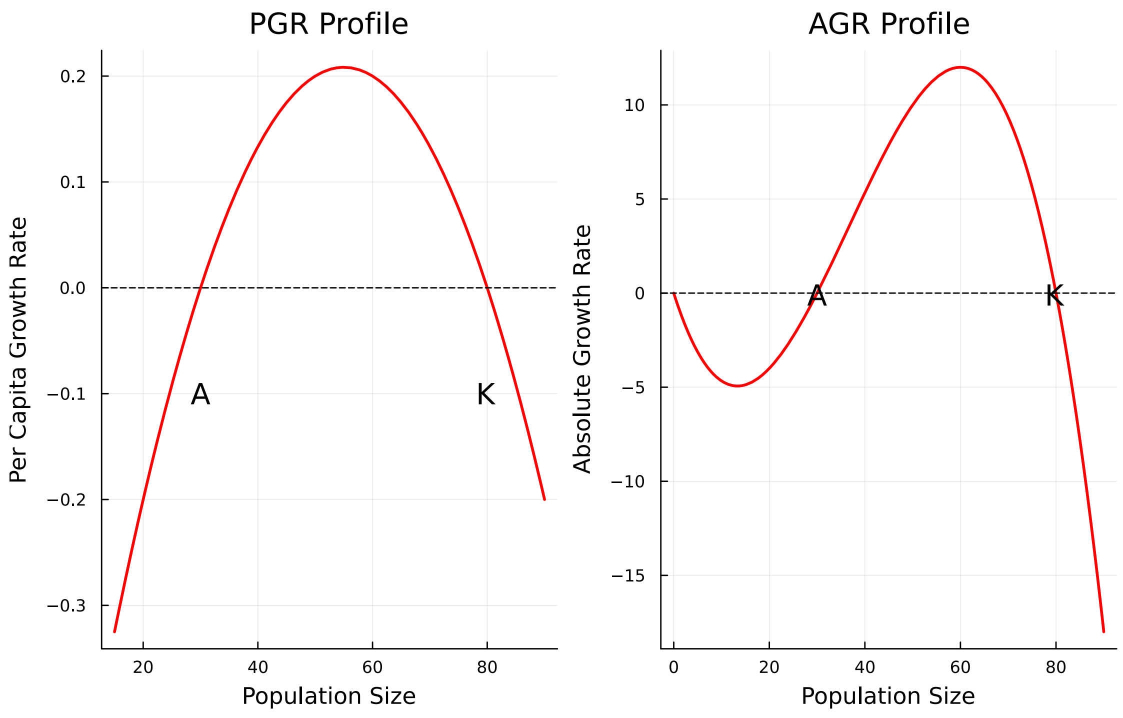
Out[16]:



In [17]:

```
r=0.8; A=30; K=80

# Per Capita Growth Rate
p1 = plot(x -> r*(x/A - 1)*(1 - x/K), 15, 90, color = "red", lw = 2,xlabel = "Population S
    ylabel = "Per Capita Growth Rate", title = "PGR Profile", label = "")
hline!([0], color = "black", linestyle = :dash, label = "")
annotate!([80],[-0.1], "K")
annotate!([30],[-0.1], "A")
# Absolute Growth Rate
p2 = plot(x -> r*x*(x/A - 1)*(1 - x/K) , 0, 90, color = "red", lw = 2 ,xlabel = "Populati
    ylabel = "Absolute Growth Rate", title = "AGR Profile", label = "")
hline!([0], color = "black", linestyle = :dash, label = "")
annotate!([80],[-0.1], "K")
annotate!([30],[-0.1], "A")

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

Out[17]:



6. Nonlinear Regression

We have already discussed some aspects of linear regression using `Julia`, particularly the use of the function `lm()` from the `GLM.jl` package. This function is particularly useful for fitting linear models or nonlinear models that can be converted to a linear model. However, there are many models that cannot be written as a linear function of the parameters. In such a case, `nonlinear regression modeling` is required.
In Julia, the `LsqFit.curve_fit` function from the `LsqFit.jl` package takes care of the estimation of parameters for `nonlinear functions`. Since we have already learned about different growth curve models, we shall use these models as nonlinear functions whose parameters are to be estimated from given data. To do this, we shall write functions for all the models.
The idea is that, for each of the growth models,

$$\mathbf{X}_t = f(t),$$

we shall simulate the data at different time points. Then we fit the model using the simulated data and obtain the parameter estimates using the `curve_fit` function. We shall compare the `true curve` and the `estimated curve` using plots. Note that we have already solved the growth equations in Exercise-16 to obtain $\mathbf{X}_t$ as a function of t. Run the following code, and the idea will become clear.

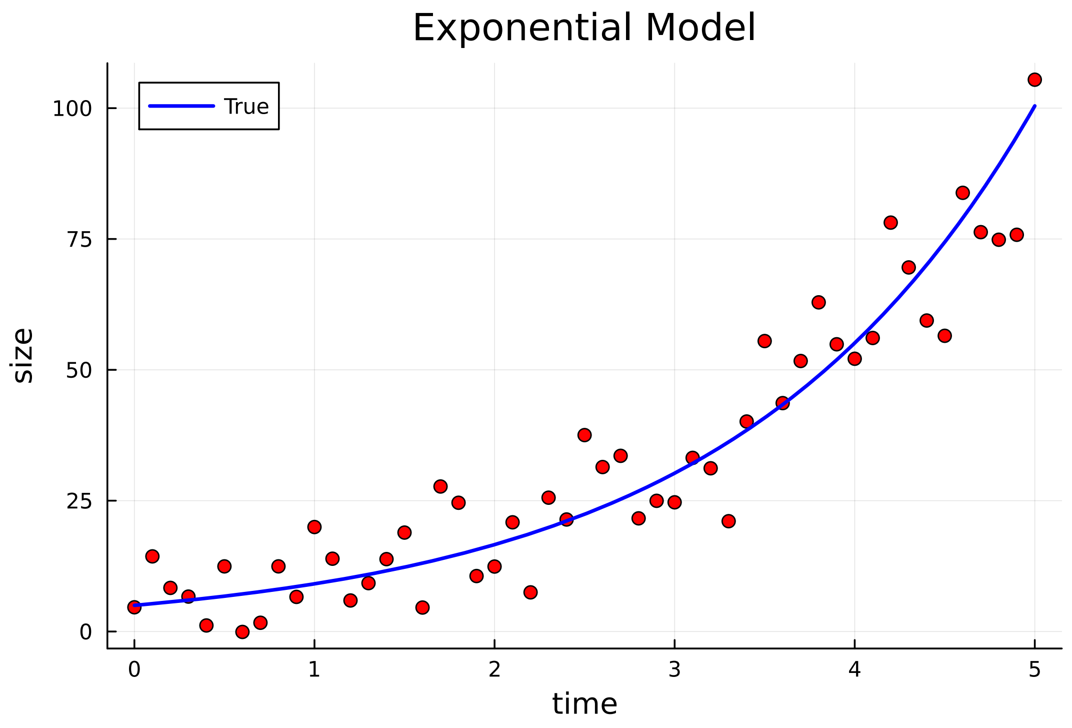
> **Note:**
> Before going into real data, we shall study a little bit about **statistical simulation**. The idea is very simple: to simulate data artificially. Every time, we may not have the data to fit a model. However, Julia will allow us to incorporate the randomness in the observations by **artificially simulating random numbers**.

6.1. Exponential Model

In [18]:
```julia
using Distributions
using Statistics
using LsqFit
using LaTeXStrings
using Plots
using DataFrames
```

In [19]:
```julia
# True parameters
x0 = 5
r = 0.6
# Time points
t = collect(0:0.1:5)
# Simulated data: Exponential model with Gaussian noise
x = x0 .* exp.(r .* t) .+ rand(Normal(0, 9), length(t))
data = DataFrame(t = t, x = x)
scatter(t, x, color = "red", xlabel = "time", ylabel = "size",
label = "", title = "Exponential Model")
plot!(t-> x0 .*exp.(r .*t), label = "True", color = :blue, lw = 2)
```

Out[19]:



In [20]:
```
ExpModel(t, p) = p[1] .* exp.(p[2] .* t)  # p[1] = x0, p[2] = r
```

Out[20]:
```
ExpModel (generic function with 1 method)
```

In [21]:
```
# Initial parameter guess:
p0 = [6.0, 0.7]

# Fit the model
fit_1 = curve_fit(ExpModel, t, x, p0)
x0_hat, r_hat = coef(fit_1)  # estimated parameters
println("Estimated parameters: x0 = $x0_hat, r = $r_hat")
```

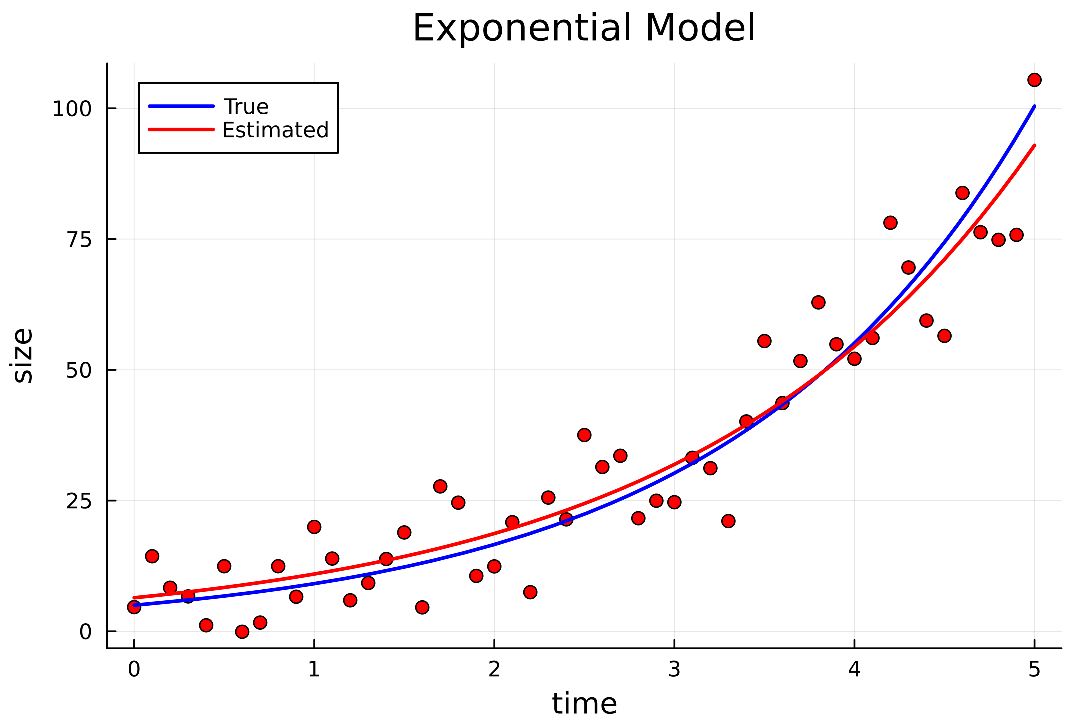
```
Estimated parameters: x0 = 6.42066336239715, r = 0.5344692490406453
```

In [22]:
```
# extract the fitted parameters
p_fit_1 = fit_1.param
```

Out[22]:
```
2-element Vector{Float64}:
 6.42066336239715
 0.5344692490406453
```

In [23]:
```
# fitted values
x_hat = ExpModel(t, p_fit_1)
plot!(t, x_hat, color = "red", lw = 2, label = "Estimated")
```

Out[23]:

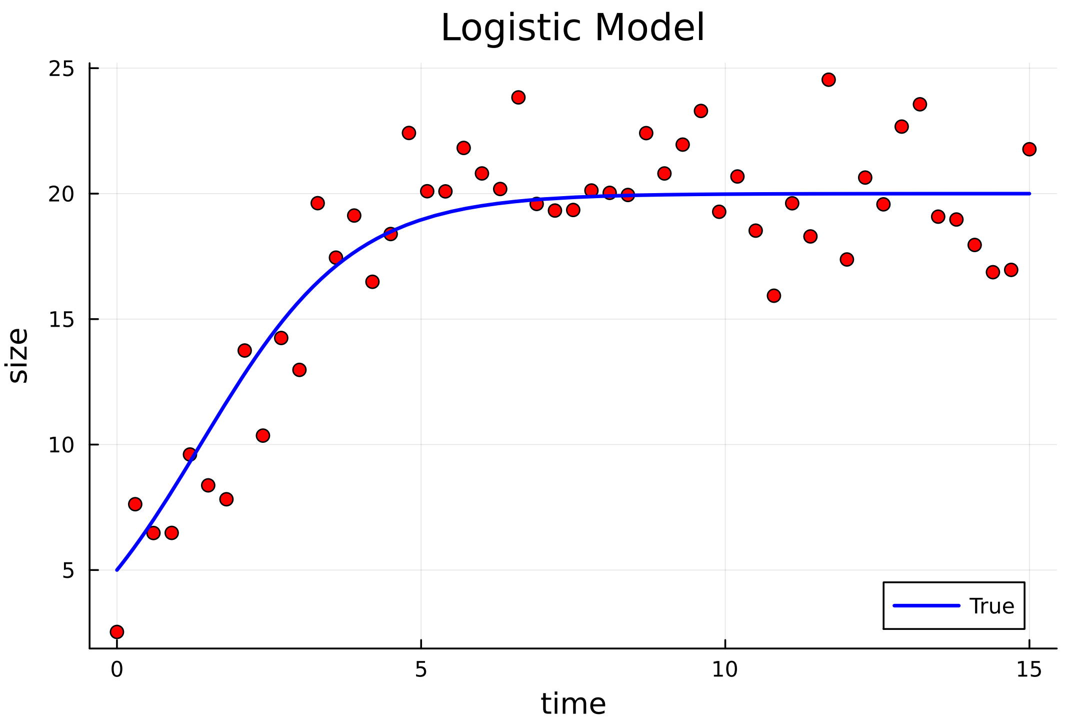


6.2. Logistic Model

In [24]:
```
using Distributions
using Statistics
using LsqFit
using LaTeXStrings
using Plots
using DataFrames
```

In [25]:
```
# true parameters
x0 = 5; r = 0.8; K = 20;

# function of logistic model

function logisticFun(t, x0, r, K)
     K/(1+(K/x0 - 1)*exp(-r*t))
end

t = collect(0:0.3:15)
x = logisticFun.(t, x0, r, K) .+ rand(Normal(0, 2), length(t))
data = DataFrame(t = t, x = x )

scatter(t, x, color = "red", xlabel = "time", ylabel = "size",
label = "", title = "Logistic Model")
plot!(t-> logisticFun(t, x0, r, K), label = "True", color = :blue, lw = 2)
```

Out[25]:

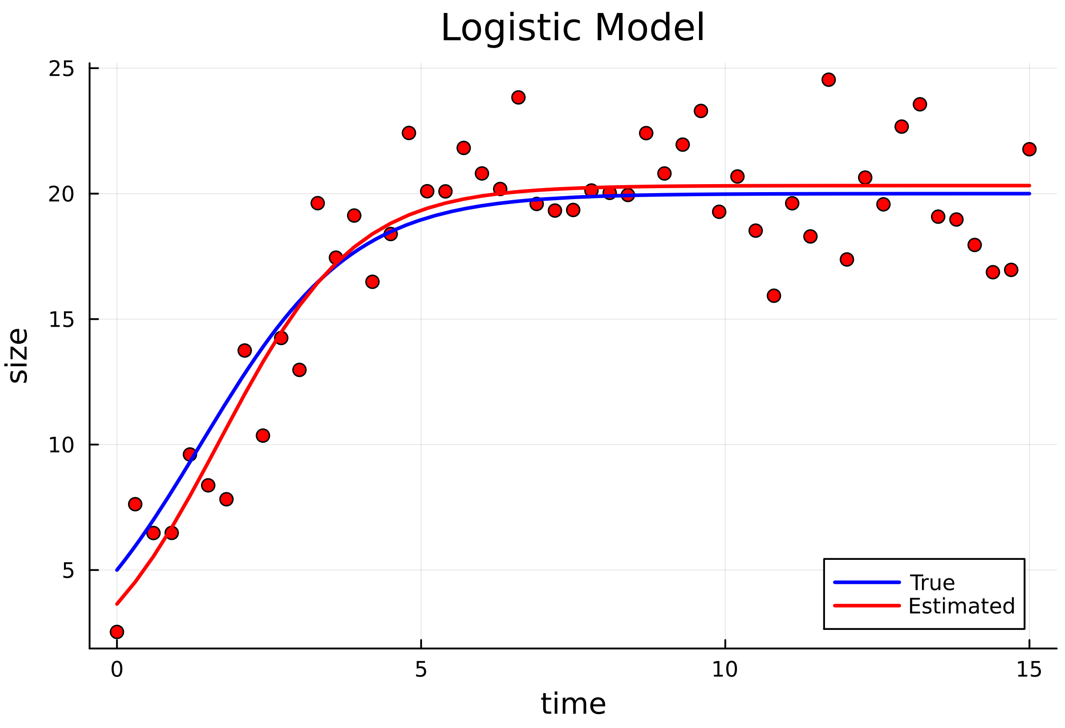


In [26]:
```
logisticModel(t, p) = p[3] ./ (1 .+ (p[3] ./ p[1] .- 1) .* exp.(-p[2] .* t))

# Initial guess
p0 = [6.0, 1.0, 20.0]

# Fit the model
fit_logisticModel = curve_fit(logisticModel, t, x, p0)

x0_hat, r_hat, K_hat = coef(fit_logisticModel)  # estimated parameters
println("Estimated parameters: x0 = $x0_hat, r = $r_hat, K = $K_hat")
```

```
Estimated parameters: x0 = 3.645711681119522, r = 0.8993251464335363, K = 20.3210877239354
56
```

In [27]:
```
# extract the fitted parameters
p_fit_logisticModel = fit_logisticModel.param
# fitted values
x_hat = logisticModel(t, p_fit_logisticModel)
plot!(t, x_hat, color = "red", lw = 2, label = "Estimated")
```

Out[27]:

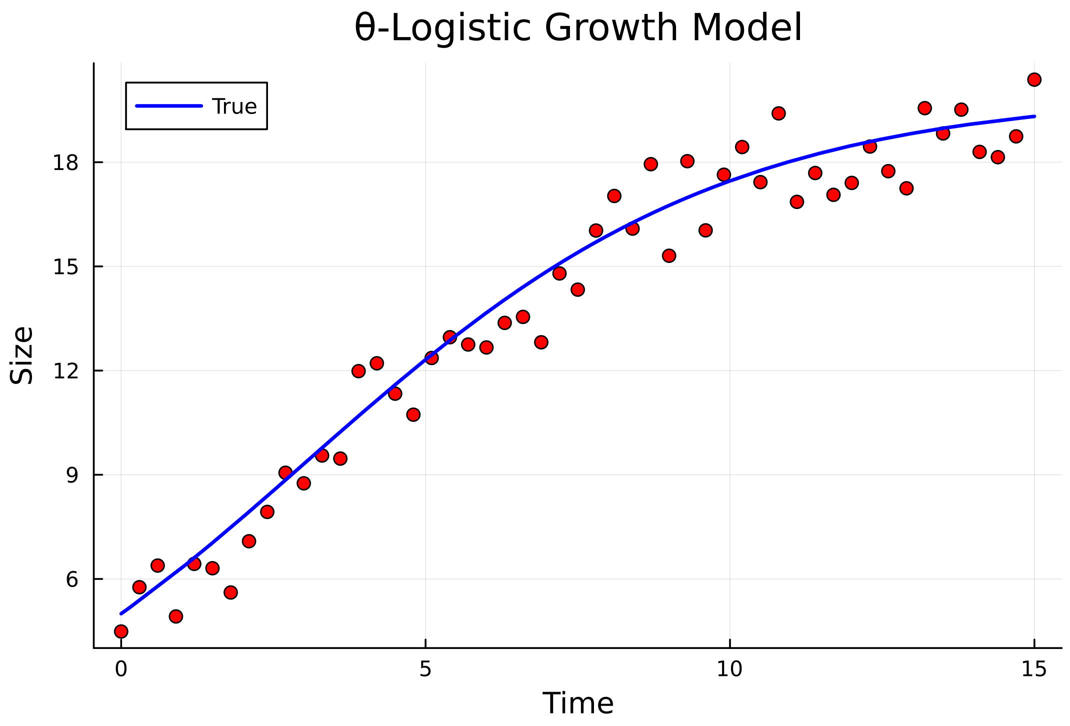


6.3. theta-Logistic Model

In [28]:
```
using Distributions
using Statistics
using LsqFit
using LaTeXStrings
using Plots
using DataFrames
```

In [29]:
```
# ϑ-Logistic growth function
function ThetaLogisticFun(t, x0, r, K, θ)
    K / (1 + exp(-θ * r * t) * ((K / x0)^θ - 1))^(1 / θ)
end

x0 = 5
r = 0.4
K = 20
θ = 0.7
t = collect(0:0.3:15)
x = ThetaLogisticFun.(t, x0, r, K, θ) .+ rand(Normal(0, 1), length(t))
data = DataFrame(t = t, x = x)
scatter(t, x, color = "red", xlabel = "Time", ylabel = "Size",
        label = "", title = "θ-Logistic Growth Model")
plot!(t -> ThetaLogisticFun(t, x0, r, K, θ), label = "True",
      color = :blue, lw = 2)
```

Out[29]:

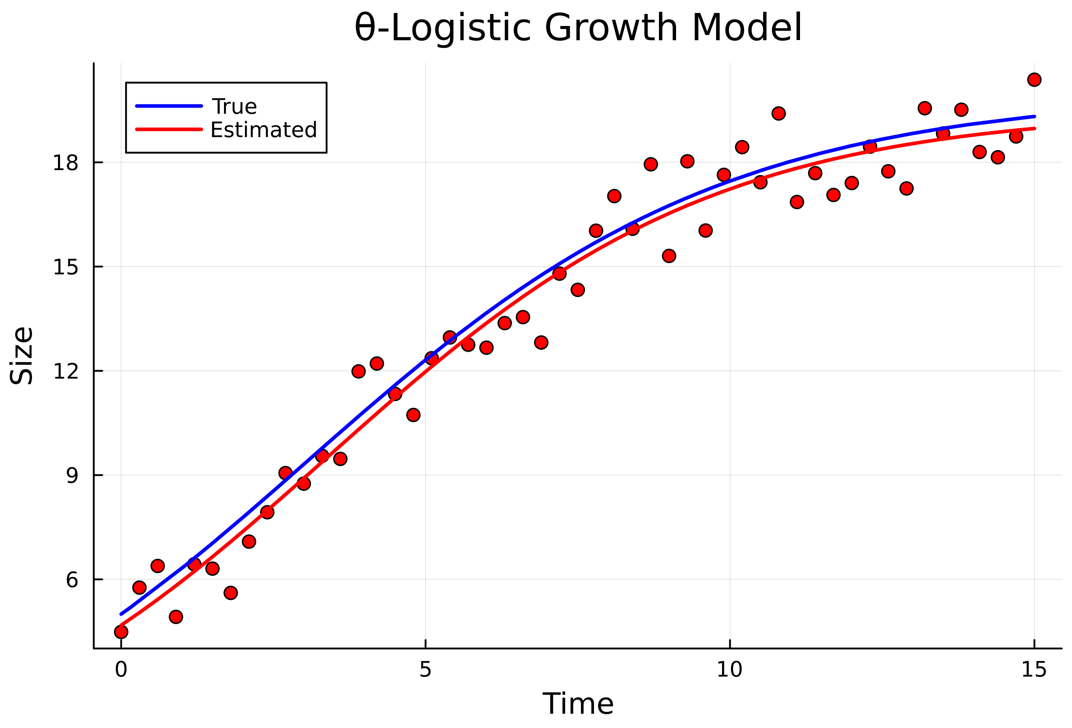


In [30]:
```
# ϑ-Logistic growth function to peform non linear model
function ThetaLogisticModel(t, p)
     p[3] ./ (1 .+ exp.(-p[4] .* p[2] .* t) .*
        ((p[3] ./ p[1]) .^ p[4] .- 1)) .^ (1 ./ p[4])
end

p0 = [5.0, 0.4, 20.0, 0.7]

fit_ThetaLogisticModel = curve_fit(ThetaLogisticModel, t, x, p0)

x0_hat, r_hat, K_hat, θ_hat = coef(fit_ThetaLogisticModel)  # estimated parameters
println("Estimated parameters: x0 = $x0_hat, r = $r_hat, K = $K_hat,
    θ = $θ_hat")
```

```
Estimated parameters: x0 = 4.6799945476969445, r = 0.35698403609554547, K = 19.51227146345
7367,
    θ = 0.8577278482028278
```

In [31]:
```
# extract the fitted parameters
p_fit_ThetaLogisticModel = fit_ThetaLogisticModel.param
# fitted values
x_hat = ThetaLogisticModel(t, p_fit_ThetaLogisticModel)
plot!(t, x_hat, color = "red", lw = 2, label = "Estimated")
```

Out[31]:



6.4. Gompertz Model

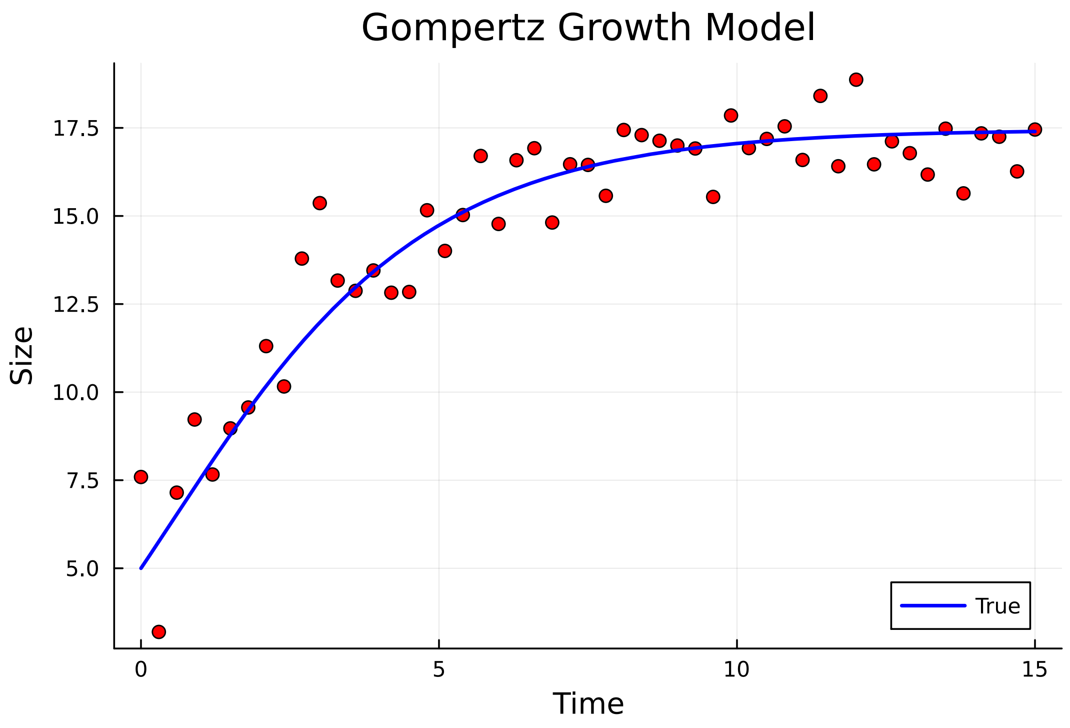
In [32]:

```
using Distributions
using Statistics
using LsqFit
using LaTeXStrings
using Plots
using DataFrames
```

In [33]:

```
function GompertzFun(t, x0, r, α)
     x0*exp(r*(1-exp(-α*t))/α)
end

x0 = 5; α = 0.4; r = 0.5;

t = collect(0:0.3:15)
x = GompertzFun.(t, x0, r, α) .+ rand(Normal(0, 1), length(t))

data = DataFrame(t = t, x = x)
scatter(t, x, color = "red", xlabel = "Time", ylabel = "Size",
        label = "", title = "Gompertz Growth Model")

plot!(t -> GompertzFun(t, x0, r, α) , label = "True",
      color = :blue, lw = 2)
```

Out[33]:

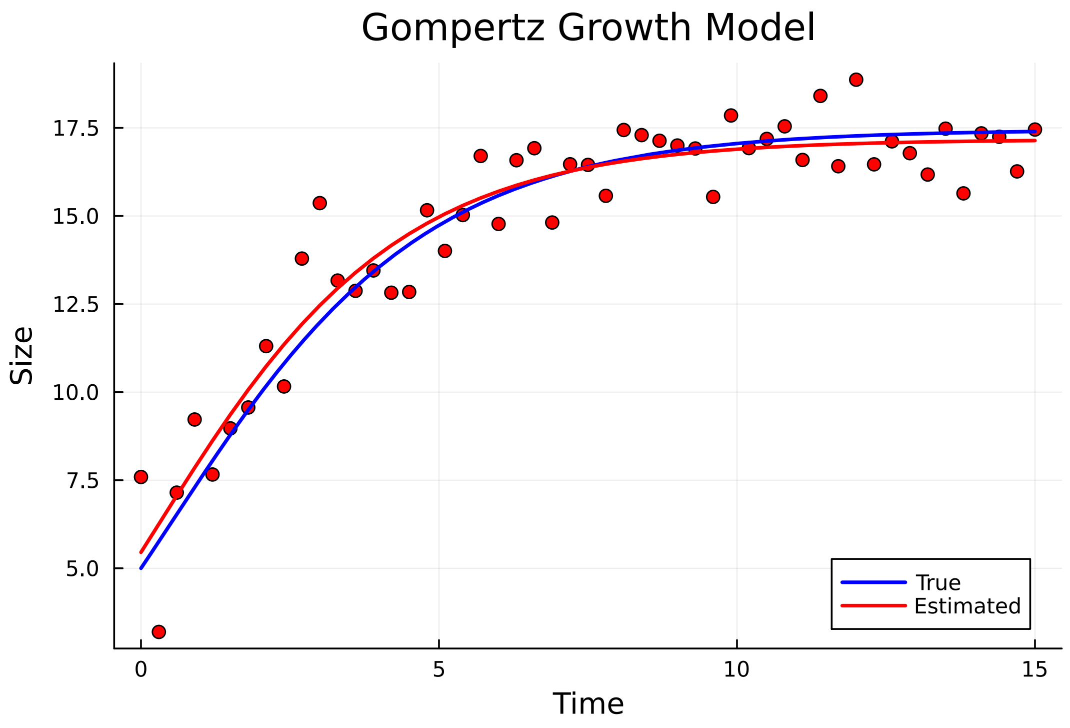


In [34]:
```
# define the model for to perform non -linear model
function GompertzModel(t, p)
    p[1] .* exp.((p[2] .* (1 .- exp.(-p[3] .* t))) ./ p[3])
end

p0 = [5.0, 0.5, 0.4]

fit_GompertzModel = curve_fit(GompertzModel, t, x, p0)

x0_hat, r_hat, α_hat = coef(fit_GompertzModel)  # estimated parameters
println("Estimated parameters: x0 = $x0_hat, r = $r_hat, α = $α_hat")
```

```
Estimated parameters: x0 = 5.452980248155089, r = 0.48732335884336203, α = 0.4247747872039
7666
```

In [35]:
```
# extract the fitted parameters
p_fit_GompertzModel = fit_GompertzModel.param
# fitted values
x_hat = GompertzModel(t, p_fit_GompertzModel)
plot!(t, x_hat, color = "red", lw = 2, label = "Estimated")
```

Out[35]:



6.5. Bootstrapping Non-linear Regression Model

Note that we have already computed the bootstrap estimates of the parameters of the linear regression model. The same approach can also be applied to nonlinear regression models. First, we will demonstrate this by explicitly writing Julia code, and then we shall explore the use of specialized Julia packages to obtain the bootstrap distribution of the parameters for nonlinear models. We take the logistic model as a case study, which can be easily adapted to other models.

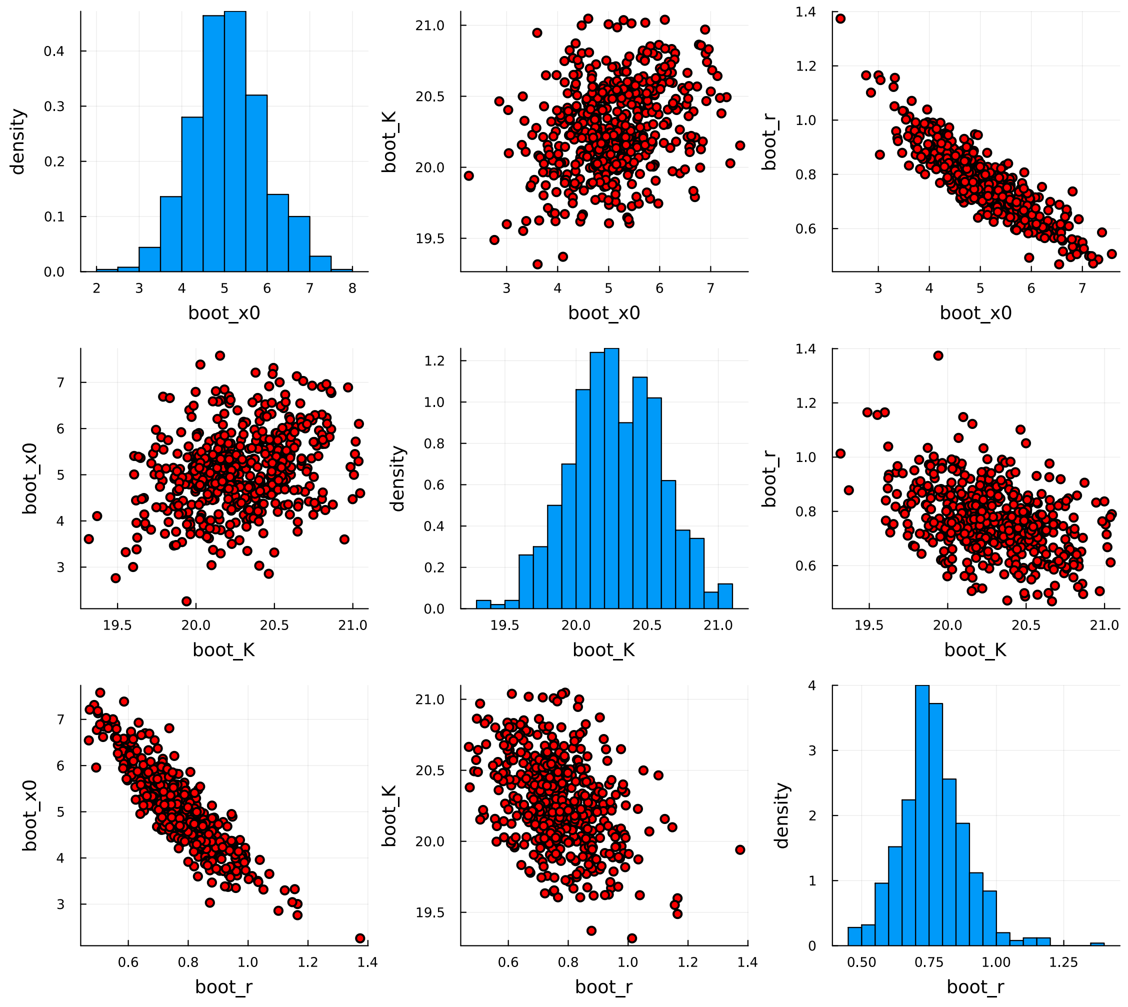
In [36]:

```julia
using Distributions
using Statistics
using LsqFit
using LaTeXStrings
using Plots
using DataFrames
using StatsBase
```

In [37]:

```julia
function logistic_model(t, p)
    p[2] ./ (1 .+ (p[2] ./ p[1] .- 1) .* exp.(-p[3] .* t))
end

x0 = 5
K = 20
r = 0.8
t = collect(0:0.3:15)
x = logistic_model(t, [x0, K, r]) .+ rand(Normal(0, 2), length(t))

data = DataFrame(t = t, x = x)

# Bootstrap step
B = 500
boot_x0 = zeros(B)
boot_K  = zeros(B)
boot_r  = zeros(B)
p0 = [5.0, 20.0, 0.8]  # initial guess

for i in 1:B
    brow = sample(1:nrow(data), nrow(data), replace = true)
    newdata = data[brow, :]
    t_boot = newdata.t
    x_boot = newdata.x

    fit = curve_fit(logistic_model, t_boot, x_boot, p0)
    boot_x0[i] = coef(fit)[1]
    boot_K[i]  = coef(fit)[2]
    boot_r[i]  = coef(fit)[3]
end

plot(layout = (3, 3), size = (1000, 900))

histogram!(boot_x0, normalize = true , xlabel = "boot_x0",ylabel = "density",
subplot = 1, label = "")

scatter!(boot_x0, boot_K, color = "red", xlabel = "boot_x0",
ylabel = "boot_K", subplot = 2, label = "")

scatter!(boot_x0, boot_r, color = "red", xlabel = "boot_x0",
ylabel = "boot_r", subplot = 3, label = "")

scatter!(boot_K, boot_x0, color = "red", xlabel = "boot_K",
ylabel = "boot_x0", subplot = 4, label = "")

histogram!(boot_K, normalize = true , xlabel = "boot_K",ylabel = "density",
subplot = 5, label = "")

scatter!(boot_K, boot_r, color = "red", xlabel = "boot_K",
ylabel = "boot_r", subplot = 6, label = "")


scatter!(boot_r, boot_x0, color = "red", xlabel = "boot_r",
ylabel = "boot_x0", subplot = 7, label = "")

scatter!(boot_r, boot_K, color = "red", xlabel = "boot_r",
ylabel = "boot_K", subplot = 8, label = "")

histogram!(boot_r, normalize = true , xlabel = "boot_r",ylabel = "density",
 subplot = 9, label = "")
```

Out[37]:



7. Real Data Analysis

Data were collected on length of fish, Cirrhinus mrigala, at 12 consecutive time points for each of the four equi-spaced directions to be referred to as A, B, C and D, emanating at 45◦ from the center of the lake to its four corners. At each time point, 12 measurements were available. Fishes were combined in the hoop nets placed at an equal radial distance from the center in each of these directions.The details of the data set can be found in `Bhowmick and Bhattacharya, 2014 (Mathematical Biosciences, Elsevier Publications)`

In [38]:
```julia
using Distributions
using Statistics
using LsqFit
using Plots
using LaTeXStrings
using CSV
using Distributions
using DataFrames
```

In [39]:
```julia
data_A = CSV.File("LocationA.txt"; delim = '\t',
    header = false, missingstring = "NA") |> DataFrame
t = 1:12  # time points
size(data_A)  # dimensions of data

# Convert DataFrame to matrix and transpose for plotting
mat = Matrix(data_A)
mat_trans = permutedims(mat)
p1 = scatter(t, mat_trans, xlabel = "Time (week)", ylabel = "Observed length (cm)",
        title = "Location A", color = "black", label = "")
```

```julia
x = mean.(eachcol(data_A))
data = DataFrame(t = t, x = x)
```

Out[39]: 12×2 DataFrame

Row	t	x
	Int64	Float64
1	1	3.6425
2	2	3.9475
3	3	4.24
4	4	4.73167
5	5	5.645
6	6	6.83083
7	7	7.89083
8	8	8.6825
9	9	9.00417
10	10	9.20083
11	11	9.57917
12	12	9.82

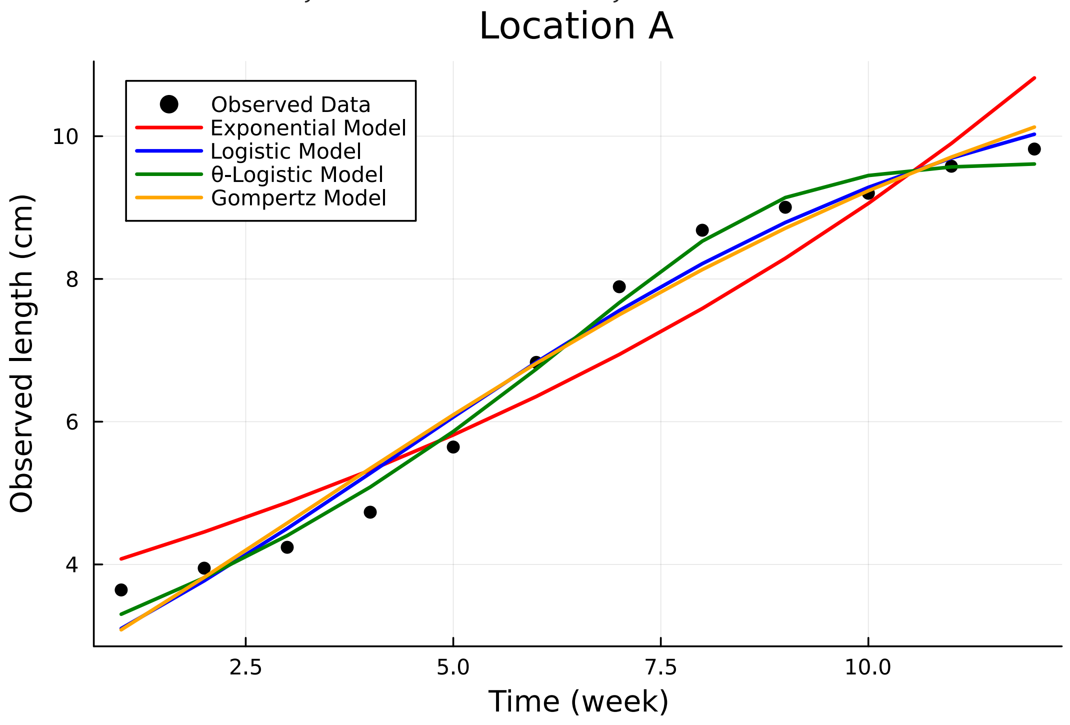
7.1. Fitting the Different Growth Models to the Real Data

In [40]:
```julia
using LsqFit, Plots

# --- Exponential Model ---
ExpModel(t, p) = p[1] .* exp.(p[2] .* t)  # p = [x0, r]
p0 = [x[1], 0.7]
fit_ExpModel = curve_fit(ExpModel, t, x, p0)
x0_hat, r_hat = coef(fit_ExpModel)
println("Estimated parameters of Exponential Model:\n  x0 = $x0_hat, r = $r_hat")
x_hat = ExpModel(t, fit_ExpModel.param)

# Initialize plot
plot(t, x, seriestype = :scatter, color = :black, label = "Observed Data",
     xlabel = "Time (week)", ylabel = "Observed length (cm)",
    title = "Location A")

plot!(t, x_hat, color = "red", lw = 2, label = "Exponential Model")


# --- Logistic Model ---
logisticModel(t, p) = p[3] ./
(1 .+ (p[3] ./ p[1] .- 1) .* exp.(-p[2] .* t))
p0 = [x[1], 1.0, 20.0]
fit_logisticModel = curve_fit(logisticModel, t, x, p0)
x0_hat, r_hat, K_hat = coef(fit_logisticModel)
println("Estimated parameters of Logistic Model:
    \n  x0 = $x0_hat, r = $r_hat, K = $K_hat")
x_hat = logisticModel(t, fit_logisticModel.param)
plot!(t, x_hat, color = "blue", lw = 2, label = "Logistic Model")


# --- ϑ-Logistic Model ---
ThetaLogisticModel(t, p) = p[3] ./
```

```
(1 .+ exp.(-p[4] .* p[2] .* t) .*
    ((p[3] ./ p[1]) .^ p[4] .- 1)) .^ (1 ./ p[4])
p0 = [x[1], 0.4, 20.0, 0.7]
fit_ThetaLogisticModel = curve_fit(ThetaLogisticModel, t, x, p0)
x0_hat, r_hat, K_hat, θ_hat = coef(fit_ThetaLogisticModel)
println("Estimated parameters of θ-Logistic Model:\n
    x0 = $x0_hat, r = $r_hat, K = $K_hat, θ = $θ_hat")
x_hat = ThetaLogisticModel(t, fit_ThetaLogisticModel.param)
plot!(t, x_hat, color = "green", lw = 2, label = "θ-Logistic Model")


# --- Gompertz Model ---
GompertzModel(t, p) = p[1] .* exp.((p[2] .* (1 .- exp.(-p[3] .* t)))
    ./ p[3])
p0 = [x[1], 0.5, 0.4]
fit_GompertzModel = curve_fit(GompertzModel, t, x, p0)
x0_hat, r_hat, α_hat = coef(fit_GompertzModel)
println("Estimated parameters of Gompertz Model:
    \n  x0 = $x0_hat, r = $r_hat, α = $α_hat")
x_hat = GompertzModel(t, fit_GompertzModel.param)
plot!(t, x_hat, color = :orange, lw = 2, label = "Gompertz Model")
```

```
Estimated parameters of Exponential Model:
  x0 = 3.731256008921676, r = 0.08869542730368626
Estimated parameters of Logistic Model:

  x0 = 2.5124147566579245, r = 0.2810406634337698, K = 11.219871961463506
Estimated parameters of θ-Logistic Model:

    x0 = 2.85937111441383, r = 0.1440655709013581, K = 9.630039939023035, θ = 7.9972074881
52421
Estimated parameters of Gompertz Model:

  x0 = 2.399547804690016, r = 0.2719611664125153, α = 0.16174666801872914
```

Out[40]:



7.2. Accuracy Measures for All the Model Fitting Result

To assess and compare the goodness-of-fit of the four nonlinear growth models :— `Exponential`, `Logistic`, `θ-Logistic`, and `Gompertz`,we compute the `Mean Squared Error (MSE)` for each model. The MSE is calculated as the mean of the squared residuals, which provides a measure

of how well the model predictions align with the observed data. A lower MSE value indicates a better fit.
The following MSE values correspond to each model:

In [41]:
```
# Compute Mean Squared Errors
mse_exp       = mean(residuals(fit_ExpModel).^2)
mse_logistic  = mean(residuals(fit_logisticModel).^2)
mse_theta     = mean(residuals(fit_ThetaLogisticModel).^2)
mse_gompertz  = mean(residuals(fit_GompertzModel).^2)

# Display as a vector
mse_values = [mse_exp, mse_logistic, mse_theta, mse_gompertz]
println("MSE for each model:")
println("Exponential:     ", round(mse_exp, digits=4))
println("Logistic:        ", round(mse_logistic, digits=4))
println("θ-Logistic:      ", round(mse_theta, digits=4))
println("Gompertz:        ", round(mse_gompertz, digits=4))
```

```
MSE for each model:
Exponential:     0.4312
Logistic:        0.1087
θ-Logistic:      0.0448
Gompertz:        0.1401
```

📊 Model Selection Using AIC in Julia

To compare and select the best nonlinear model, we use the **Akaike Information Criterion (AIC)**, which balances goodness-of-fit and model complexity.

Since the `LsqFit.jl` package in Julia does **not provide a built-in `AIC()` function**, we manually compute it using the log-likelihood-based formula:

$$\mathrm{AIC} = n \cdot \log(2\pi \cdot \frac{\mathrm{RSS}}{n}) + n + 2k$$

where:

- n is the number of observations,
- RSS is the residual sum of squares,
- k is the number of model parameters.

This formulation matches the output from R's `nls()` function and ensures **positive AIC values**.

A **lower AIC value** indicates a better model, considering both **fit accuracy** and **model simplicity**.

In [42]:
```
function aic(fit::LsqFit.LsqFitResult, n::Int)
    rss = sum(residuals(fit).^2)
    k = length(coef(fit))
    return n * log(2π * rss / n) + n + 2k
end

n = length(x)  # number of data points

aic_exp      = aic(fit_ExpModel, n)
aic_logistic = aic(fit_logisticModel, n)
aic_theta    = aic(fit_ThetaLogisticModel, n)
aic_gompertz = aic(fit_GompertzModel, n)

aic_vals = [aic_exp, aic_logistic, aic_theta, aic_gompertz]
model_names = ["Exponential", "Logistic", "θ-Logistic", "Gompertz"]

for (name, aic_val) in zip(model_names, aic_vals)
    println(rpad(name, 12), "AIC = ", round(aic_val, digits = 2))
end
```

```
Exponential AIC = 27.96
Logistic    AIC = 13.43
θ-Logistic  AIC = 4.8
Gompertz    AIC = 16.47
```

A model with a lower AIC is preferred over the model with a higher AIC. If AIC is considered as the measure of goodness of fit then the theta-logistic model is the best model describing the data. You may have to take caution and check the definition of AIC. The maximum value of AIC may also give an indication of better model or the minimum value of AIC. It depends on the way how the AIC computation has been implemented in the software.Another cautionary tale for AIC is that if all the candidates in the set of models are poor, still AIC returns the best model. (A critical discussion can be found in the Burnham and Anderson, 2002) (We may discuss in the class as well