Non Linear Regression Models

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

Introduction

In this tutorial, we shall understand how to fit nonlinear regression models using R for some give data set. We shall consider only a single input variable age and out variable size. We shall first simulate the data set artificially with some fixed parameter choices and then apply nonlinear least squares on the data set. This will help us to understand the accuracy of the nonlinear least squares method using Julia.

Simulation of growth data

For demonstration, we consider the following nonlinear function:

$$\text{size} = a \times \text{age}^b + e$$

where a and b are fixed parameters, and the error component e follows a normal distribution with mean 0 and variance σ^2.We use the function Random.seed!() to ensure that the simulation studies are reproducible. For the simulation study, we fixed the parameter values as $a = 1$, $b = 0.4$, and $\sigma = 0.2$.

In [1]:
```
using Statistics, StatsBase, StatsModels
using Plots, Distributions, LaTeXStrings, Random
using DataFrames, LinearAlgebra
```

In [2]:
```
Random.seed!(123)
age = collect(1:0.1:10)  # age variable
print(age)
```

```
[1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6,
2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.3,
4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0,
6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 7.0, 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7,
7.8, 7.9, 8.0, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9.0, 9.1, 9.2, 9.3, 9.4,
9.5, 9.6, 9.7, 9.8, 9.9, 10.0]
```

In [3]:
```
length(age)
```

Out[3]:
```
91
```

In [4]:
```
a = 1      # true value of a
b = 0.4    # true value of b

size_0 = a .* age .^ b     # here we yse size_0 imstead of size
println(first(size_0, 6))
```

```
[1.0, 1.0388601182540846, 1.0756537569325701, 1.1106503068343223, 1.144066355858723,
1.1760790225246736]
```

In [5]:
```
plot(age, size_0, color = "red", xlabel = "age", lw = 2,
    ylabel = "size", label = "")
```

Out[5]:

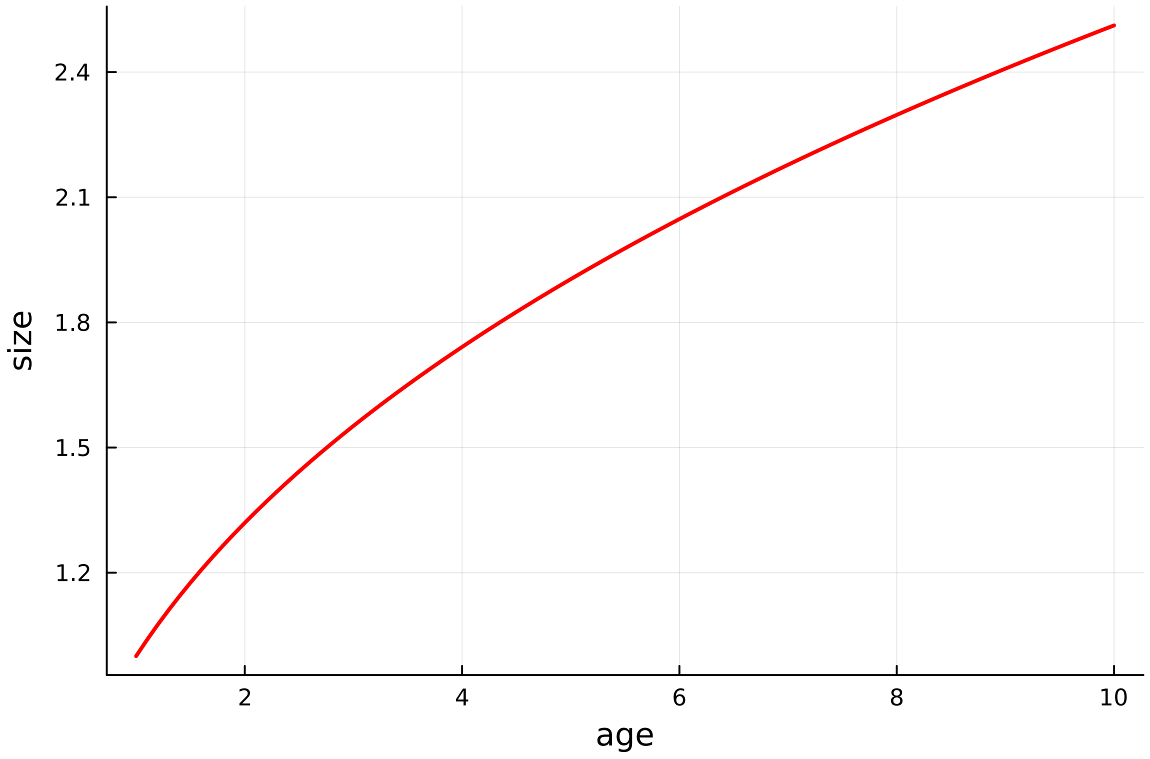

Figure 1: The mean growth profile. The parameters are fixed at $a = 1$ and $b = 0.4$. The plot should be considered as a conditional expectation of the response variable (size) at different values of age variable. $\mathbb{E}(\text{size}|\text{age})$ as a function of age. If we fix age = 4 (say), then expected size is $a \times 4^b$.

In [6]:
```
Random.seed!(123)
size_0 = a .* age .^ b .+ rand(Normal(0, 0.2), length(age))

# print(size_0)

scatter(age, size_0, color = "darkgrey", xlabel = "age",
    ylabel = "size", label = "")
plot!(age, a .* age .^ b , color = "red", lw = 2,label = "")
```

Out[6]:

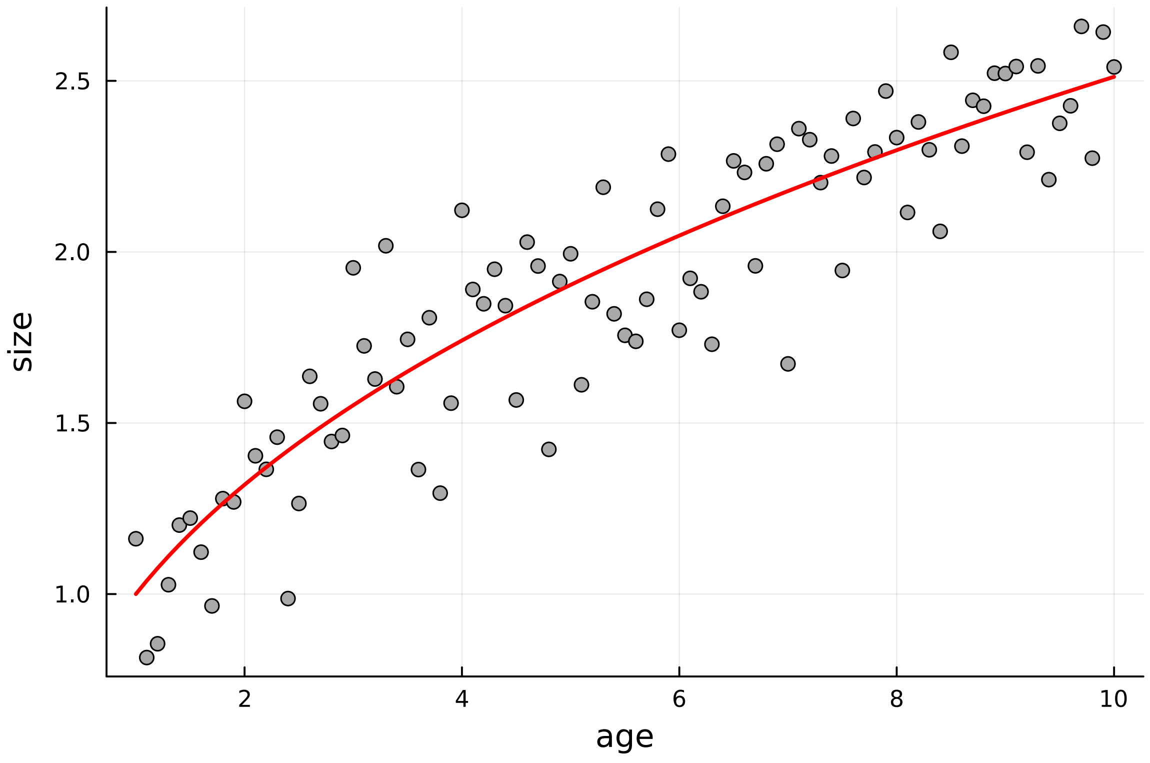

Figure 2: The simulated data set. The function Random.seed!() is set at 123 to ensure that the simulation study is reproducible. The parameters $a = 1$, $b = 0.4$, and $\sigma = 0.2$. The sample size is $n = 91$. The true conditional mean value of the response given age is added in red colour.

In [7]:
```
data = DataFrame(age = age, size_0 = size_0)
first(data, 6)
```

Out[7]: 6×2 DataFrame

Row	age	size_0
	Float64	Float64
1	1.0	1.16166
2	1.1	0.814446
3	1.2	0.854727
4	1.3	1.02725
5	1.4	1.20158
6	1.5	1.22204

To estimate the parameters based on the simulated data, we need to minimize the error sum of squares, which is given by:

$$\mathrm{ESS} = \sum_{i=1}^{n} \left(\mathrm{size}_i - a \times \mathrm{age}_i^b\right)^2$$

with respect to a and b. In the following, we try to visualize the ESS as a function of a and b. n is the sample size or the number of rows in the data set.

In [8]:
```
function ESS(a, b)
    sum((size_0 .- a .* age .^b).^2)
end

a_vals = collect(0:0.1:2)
b_vals = collect(0:0.1:2)

ESS_vals = Matrix{Float64}(undef, length(a_vals), length(b_vals))

for i in 1:length(a_vals)
    for j in 1:length(b_vals)
        ESS_vals[i,j] = ESS(a_vals[i], b_vals[j])
    end
end
```

In [9]:
```
println("The matrix containing the error sum of squares values:")
```

```
The matrix containing the error sum of squares values:
```

In [10]:
```
surface(a_vals, b_vals, ESS_vals, xlabel="a_vals", ylabel="b_vals",
 zlabel="ESS_vals",color = :viridis, camera=(50,30))
```

Out[10]:

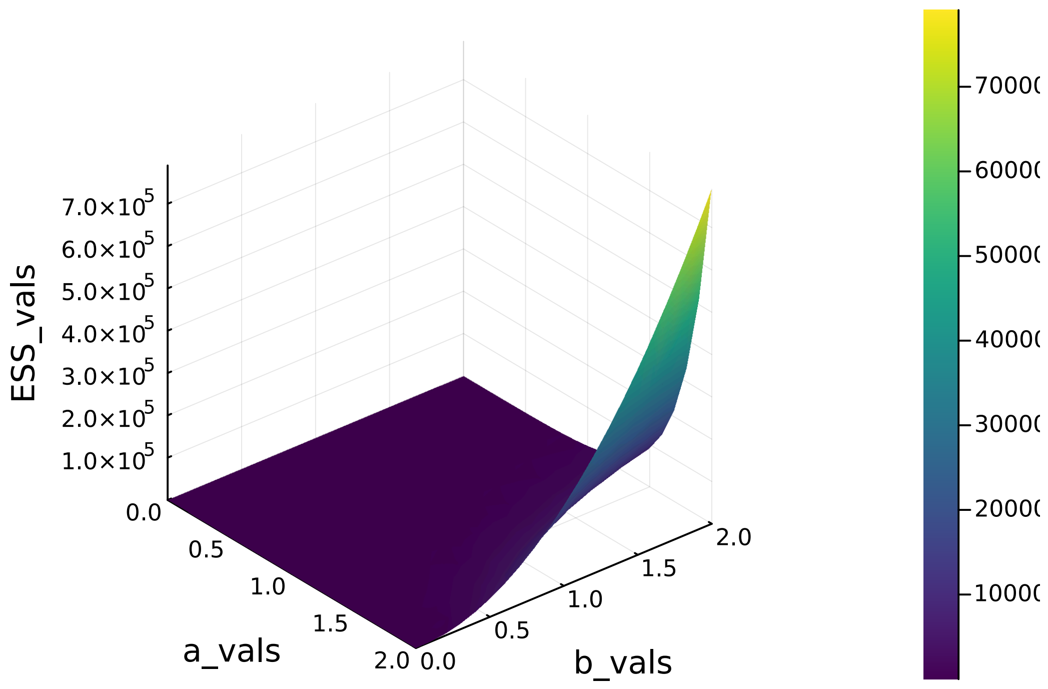

Figure 3: The variation of the error sum of squares as a function of a and b. We need to choose the values of the parameters at which the surface is minimum.

nonlinear least sqaures using Julia

In [11]:
```
using LsqFit  # required package for fitting for non linear model
```

In [12]:
```
# Define the nonlinear model function
model(age, p) = p[1] .* age .^ p[2]

# Initial parameter estimates
p = [0.5, 1.0]
```

```
# Fit the model
fit_0 = curve_fit(model, age, size_0, p)
a, b = coef(fit_0)
println("Estimated parameters: a = $a, b = $b")
```

```
Estimated parameters: a = 0.9821147835826877, b = 0.40608793931634557
```

In [13]:
```
a_hat = coef(fit_0)[1]
```

Out[13]:
```
0.9821147835826877
```

In [14]:
```
b_hat = coef(fit_0)[2]
```

Out[14]:
```
0.40608793931634557
```

Let us now plot the fitted curve to the data

In [15]:
```
typeof(fit_0)
```

Out[15]:
```
LsqFit.LsqFitResult{Vector{Float64}, Vector{Float64}, Matrix{Float64}, Vector{Float
64}, Vector{LsqFit.LMState{LsqFit.LevenbergMarquardt}}}
```

In [16]:
```
# ectract the fitted parameters
p_fit_0 = fit_0.param

# fitted values
size_hat = model(age, p_fit_0)

# residual values
error_hat = fit_0.resid

scatter(age, size_0, color = "darkgrey", markersize = 6, label = "")
plot!(age, size_hat, color = "blue", linestyle = :dash, lw = 2,
label = "")
```

Out[16]:

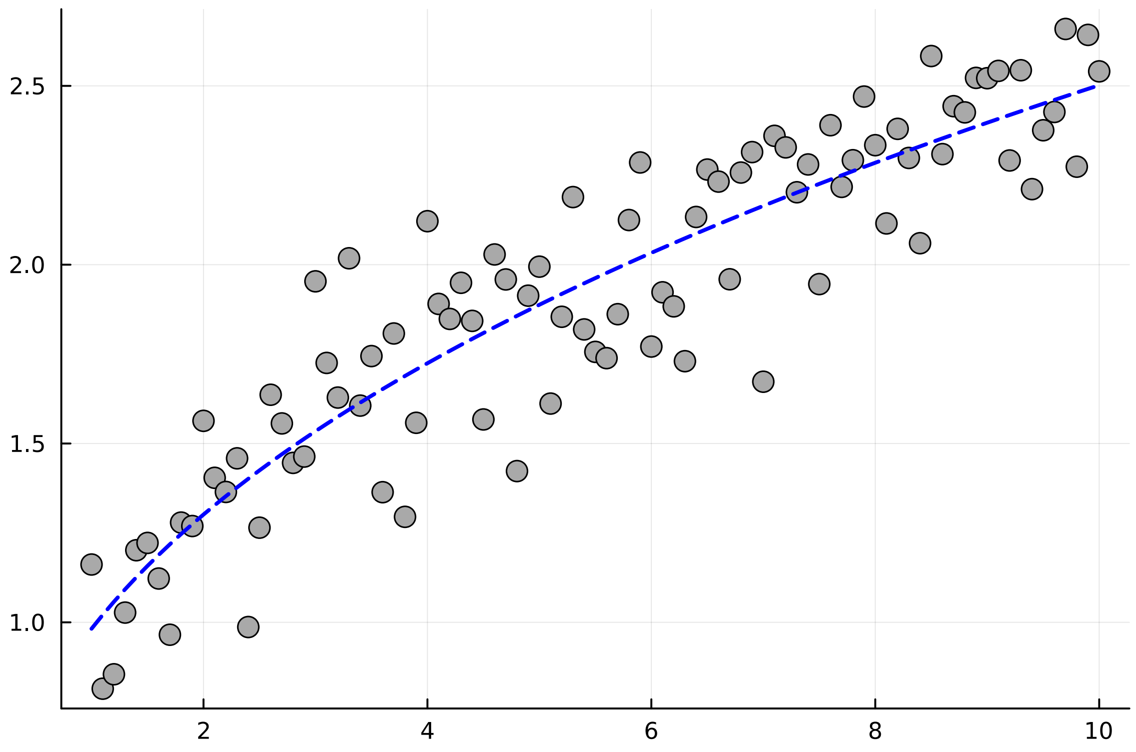

Figure 4: The fitted allometric growth equation using the LsqFit routine in Julia.

In [17]:
```julia
p1 = histogram(error_hat, normalize = true, xlabel = L"\widehat{e}",
ylabel = "density", label = "")

p2 = scatter(age, error_hat, color = "red", markersize = 6,
    xlabel = "age", ylabel = L"\widehat{e}", label = "")
hline!([0], color = "blue", linestyle = :dash, lw = 2,label = "")

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

Out[17]:

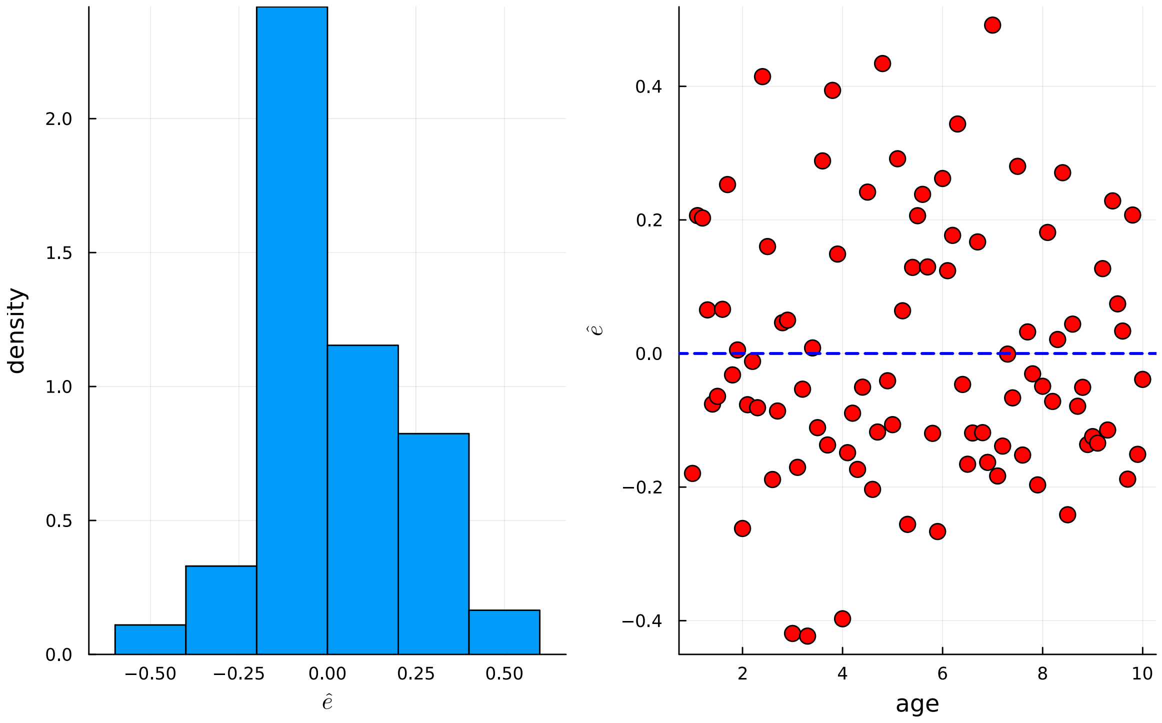

Figure 5: The behaviour of the residuals play the most critical role in nonlinear regression models

Understanding the summary of nls

In [18]:
```julia
# Extract fitted parameters
p_fit = fit_0.param

# compute the standard error
cov_matrix = estimate_covar(fit_0)

std_errors = sqrt.(diag(cov_matrix))  # Standard errors of parameters

# Compute confidence intervals (assuming normality)
alpha_0 = 0.05  # 95% confidence level
z_value = 1.96  # Approximate for 95% CI

conf_int = [(p_fit[i] - z_value * std_errors[i], p_fit[i] + z_value * std_errors[i])
    for i in eachindex(p_fit)]

# Print summary
println("Parameter Estimates: ", p_fit)
println("Standard Errors: ", std_errors)
println("95% Confidence Intervals: ", conf_int)
```

```
Parameter Estimates: [0.9821147835826877, 0.40608793931634557]
Standard Errors: [0.03996812178127027, 0.02198832982576361]
95% Confidence Intervals: [(0.903777264891398, 1.0604523022739774), (0.36299081285784
89, 0.44918506577484224)]
```

Understanding the uncertainty of nls estimates

If we want to understand the uncertainty associated with the parameter estimates, we need to repeat the following process M times (say) and obtain M many estimates of a and b. The histograms of these estimated values will give us an idea how the estimates will vary if we repeatedly sample data from the same model population. Let us repeat the above task here $M = 1000$ times and obtain the estimates $\hat{a}$ and $\hat{b}$ in each repetition.

In [1]:
```
using Statistics, StatsBase, StatsModels
using Plots, Distributions, LaTeXStrings, Random
using DataFrames, LinearAlgebra
using LsqFit
```

In [2]:
```
M = 1000
age = collect(1:0.1:10)   # age variable
a = 1  # true value of a
b = 0.4 # true value of b

a_hat = zeros(M)
b_hat = zeros(M)

for i in 1:M
    Random.seed!(i)
    size_0 = a .* age .^ b .+ rand(Normal(0, 0.2), length(age))
    data = DataFrame(age = age, size_0 = size_0)
    # Define the nonlinear model function
    model(age, p) = p[1] .* age .^ p[2]
    # Initial parameter estimates
    p = [0.5, 1.0]
    # Fit the model
    fit_0 = curve_fit(model, age, size_0, p)
    a_hat[i] = coef(fit_0)[1]
    b_hat[i] = coef(fit_0)[2]
end
```

In [3]:
```
p1 = histogram(a_hat, normalize = true, xlabel = L"\widehat{a}",
ylabel = "density", bins = 30, label = "")
scatter!([a],[0], color = "red", markersize = 12, label = "")

p2 = histogram(b_hat, normalize = true, xlabel = L"\widehat{a}",
ylabel = "density", bins = 30, label = "")
scatter!([b],[0], color = "red", markersize = 12, label = "")

plot(p1, p2, layout = (1,2), size = (800, 400))
```

Out[3]:

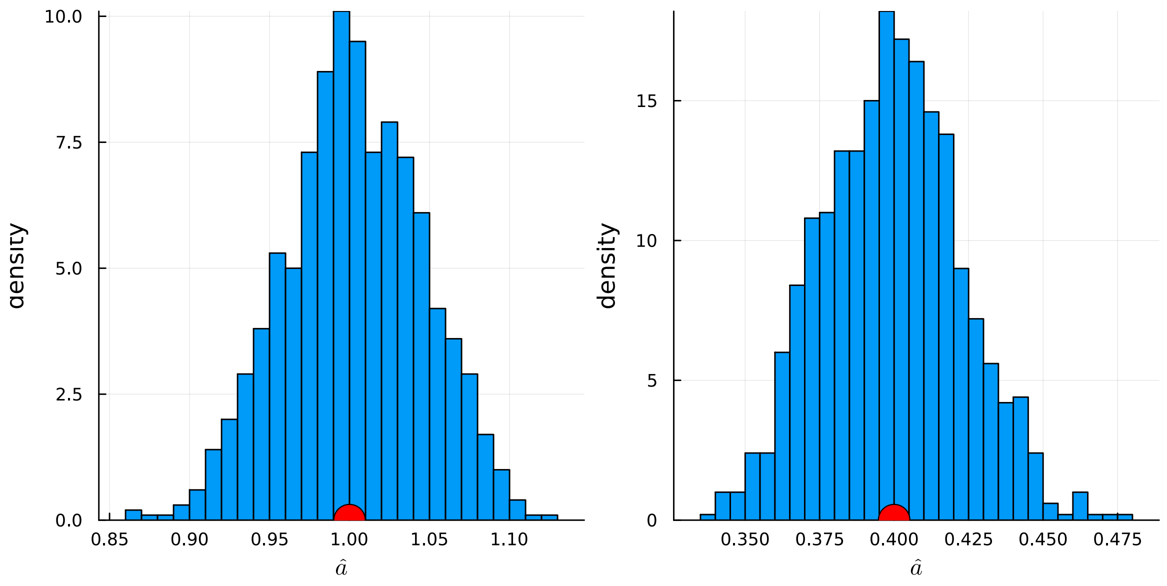

Figure 6: The approximate sampling distribution of the nonlinear least squares estimates of a and b. The sampling distribution is quiet well approxiamted by the normal distribution. In addition, note that the estimated values are centered about the true values of a and b using which the articial data sets have been simulated

Homoschedasticity versus heteroschedasticity

The statistical model to perform the nonlinear regression model is of the following form:

$$\text{size}_i = a \times \text{age}_i^b + e_i, \quad i \in \{1, 2, \ldots, n\}.$$

In the previous simulations, we have considered that $Var(e_i) = \sigma^2$ for all $i \in \{1, 2, \ldots, n\}$, that means, at different ages, the variability in the size variable remains the same about the true mean function. To simulate a scenario, we consider that $Var(e) = \sigma^2 \times \text{age}^2$. It means that as the age increases, the variability in size also increases.

In [4]:

```julia
a = 1
b = 0.4
age = collect(1:0.1:20)
size_0 = a .* age .^ b .+ rand.(Normal.(0, 0.2 .* sqrt.(age)))
data = DataFrame(age = age, size_0 = size_0)

# Define the nonlinear model function
model(age, p) = p[1] .* age .^ p[2]

# Initial parameter estimates
p = [0.5, 1.0]

# Fit the model
fit_0 = curve_fit(model, age, size_0, p)

# ectract the fitted parameters
p_fit_0 = fit_0.param

# fitted values
size_hat = model(age, p_fit_0)

# residual values
error_hat = fit_0.resid

p1 = scatter(age, size_0, color = "darkgrey", xlabel = "age",
    ylabel = "size" ,label = "")
plot!(age, size_hat, color = "blue", linestyle = :dash,
lw = 2, label = "")

p2 = scatter(age, error_hat, color = "red", xlabel = "age",
    ylabel = L"\widehat{e}" ,label = "")
hline!([0], color = "blue", linestyle = :dash, lw = 2, label = "")

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

Out[4]:

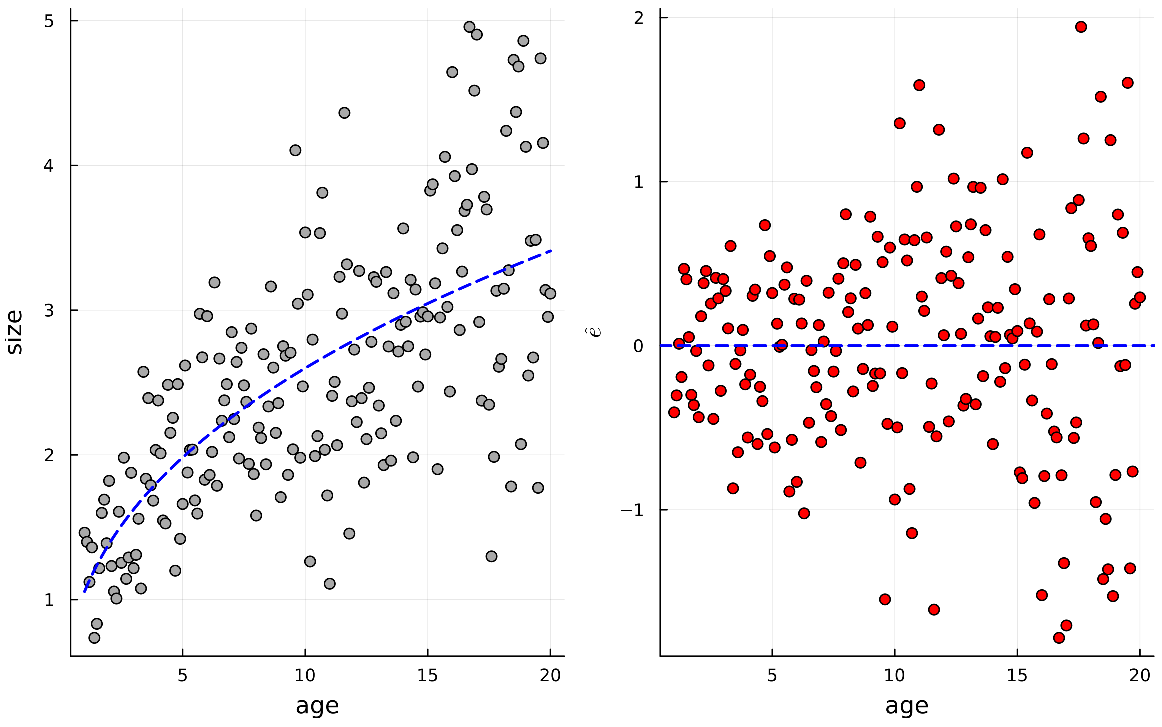

Figure 7: The simulated data gives clear indication of the presence of heteroschedasticity. The plot of residuals against the age gives whether the errors do not have a constantvariance.

Confidence Interval and Prediction Interval

In [5]:

```
a = 1
b = 0.4
age = collect(1:0.1:20)
size_0 = a .* age .^ b .+ rand(Normal(0, 0.2), length(age))
data = DataFrame(age = age, size_0 = size_0)

# Define the nonlinear model function
model(age, p) = p[1] .* age .^ p[2]

# Initial parameter estimates
p = [0.5, 1.0]

# Fit the model
fit_0 = curve_fit(model, age, size_0, p)

# The nonlinear least squares estimates of a and b are obtained as
coef(fit_0)
```

Out[5]:

```
2-element Vector{Float64}:
 0.999062782743309
 0.39762391497849175
```

The Variance Covariance matrix of $\widehat{a}$ and $\hat{b}$ is given by

In [6]:
```
vcov(fit_0)
```

Out[6]:
```
2×2 Matrix{Float64}:
  0.000703039  -0.000274405
 -0.000274405   0.000111923
```

To obtain the confidence interval at an unseen value of age, say age*, we need to compute the variance of

$$\widehat{\text{size}}^{*} = \hat{a} \times (\text{age}^{*})^{\hat{b}}$$

We observe that $\widehat{\text{size}}^{*}$ is a nonlinear function of $\hat{a}$ and $\hat{b}$ and we call it as $\psi\left(\hat{a}, \hat{b}\right)$, and we need to compute the variance of this quantity.
To approximate the variance of $\psi\left(\hat{a}, \hat{b}\right)$, we consider the first-order Taylor's approximation about the true values of a and b as given below (neglecting the higher-order terms):

$$\psi\left(\hat{a}, \hat{b}\right) = \psi(a, b) + (\hat{a} - a)\frac{\partial \psi}{\partial a} + (\hat{b} - b)\frac{\partial \psi}{\partial b}$$

Taking expectation on both sides, we obtain

$$E\left(\psi\left(\hat{a}, \hat{b}\right)\right) \approx \psi(a, b)$$

and the approximate variance of $\psi\left(\hat{a}, \hat{b}\right)$ can be obtained as

$$\begin{aligned} E\Big(\psi\left(\hat{a}, \hat{b}\right) - \psi(a, b)\Big)^2 &\approx E\left[(\hat{a} - a)\frac{\partial \psi}{\partial a} + (\hat{b} - b)\frac{\partial \psi}{\partial b}\right]^2 \\ &= Var(\hat{a})(\psi_a)^2 + Var(\hat{b})(\psi_b)^2 + 2 \times Cov(\hat{a}, \hat{b})\psi_a\psi_b \\ &= \begin{bmatrix} \psi_a & \psi_b \end{bmatrix} \begin{bmatrix} Var(\hat{a}) & Cov(\hat{a}, \hat{b}) \\ Cov(\hat{a}, \hat{b}) & Var(\hat{b}) \end{bmatrix} \begin{bmatrix} \psi_a \\ \psi_b \end{bmatrix} \end{aligned}$$

In the following code, we compute the variance of $Var\left(\psi\left(\hat{a}, \hat{b}\right)\right)$. The estimate of the variance $\widehat{Var}\left(\psi\left(\hat{a}, \hat{b}\right)\right)$ is obtained by evaluating $\frac{\partial \psi}{\partial a}$ and $\frac{\partial \psi}{\partial b}$ at $\hat{a}$ and $\hat{b}$.

In [7]:
```
a_hat = coef(fit_0)[1]
b_hat = coef(fit_0)[2]

psi_a = age .^ b_hat
psi_b = a_hat .* age .^ b_hat .* log.(age)

size_var = zeros(length(age))

for i in 1:length(age)
     size_var[i] = (psi_a[i]) .^2 .*vcov(fit_0)[1,1] .+
     (psi_b[i]) .^2 .* vcov(fit_0)[2,2] .+
     2 .* psi_a[i] .* psi_b[i] .* vcov(fit_0)[1,2]
end
```

We now construct an approximate 95% confidence interval which is given by

$$\left(\widehat{\text{size}} - 1.96 \times \sqrt{\widehat{Var}(\text{size})}, \widehat{\text{size}} + 1.96 \times \sqrt{\widehat{Var}(\text{size})}\right).$$

In the construction of the confidence interval, the irreducible error did not contribute anything. To compute the prediction interval, we need to consider the variation in the prediction due to sampling variation in the estimate of the parameters and also the random error component. Therefore, the variance of the $\widehat{\text{size}}$ is given by

$$Var(\widehat{\text{size}}) = Var\left(\psi\left(\widehat{a}, \hat{b}\right)\right) + Var(\hat{e}) = Var\left(\psi\left(\widehat{a}, \hat{b}\right)\right) + \widehat{\sigma}^2.$$

Therefore, the approximately 95% prediction interval is given by

$$\left(\widehat{\text{size}} \mp 1.96 \times \sqrt{\widehat{Var}\left(\psi\left(\widehat{a}, \hat{b}\right)\right) + \widehat{\sigma}^2}\right)$$

In [8]:
```
# ectract the fitted parameters
p_fit_0 = fit_0.param

# fitted values
size_hat = model(age, p_fit_0)

# residual values
error_hat = fit_0.resid

for i in 1:length(age)
     size_var[i] = (psi_a[i]) .^2 .*vcov(fit_0)[1,1] .+
     (psi_b[i]) .^2 .* vcov(fit_0)[2,2] .+
      2 .* psi_a[i] .* psi_b[i] .* vcov(fit_0)[1,2]
end

p1 = scatter(age, size_0 , color = "darkgrey", markersize = 6,
xlabel = "age", ylabel = "size", title = "confidenece interval",
         label = "")
plot!(age, size_hat .+ 1.96 .* sqrt.(size_var), color = "blue",
linestyle = :dash, lw = 2, label = "")

plot!(age, size_hat .- 1.96 .* sqrt.(size_var), color = "blue",
linestyle = :dash, lw = 2, label = "")

p2 = scatter(age, size_0, marker=:circle, color="darkgrey",
        title="Prediction Interval", xlabel="age", ylabel="size",
label = "")

for i in 1:length(age)
    size_var[i] = (psi_a[i])^2 * vcov(fit_0)[1,1] +
                  (psi_b[i])^2 * vcov(fit_0)[2,2] +
                  2 * psi_a[i] * psi_b[i] * vcov(fit_0)[1,2] +
                  var(error_hat)
end

plot!(age, size_hat .+ 1.96 .* sqrt.(size_var), color="red", lw=2,
    linestyle=:dash, label = "")
plot!(age, size_hat .- 1.96 .* sqrt.(size_var), color="red", lw=2,
    linestyle=:dash, label = "")

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

Out[8]:

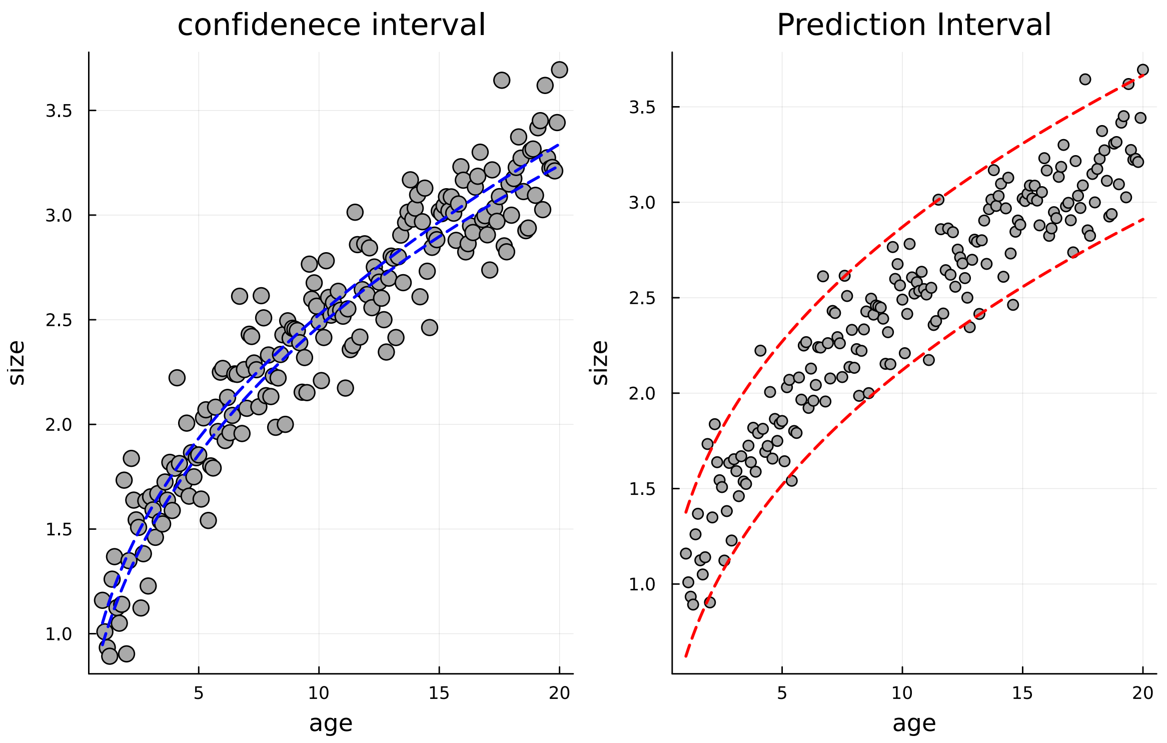

Figure 8: The demonstration of the confidence interval and prediction interval in the context of nonlinear regression model. It can be observed that the prediction interval is wider than the confidence interval. The prediction interval takes the variance of the error component also in account, where the confidence interval considers only theuncertainty associated with the estimated parameters

Taylor's approximation for one variable

In computing the confidence and prediction interval, we have chosen the normal distribution. It is important to check how accurate these approximations are. To understand this concept, we consider a simple example. Suppose that $X_1, X_2, \ldots, X_n$ be a random sample from the Exponential distribution with rate parameter λ. We are interested to estimate the parameter λ. We know the following from the Central Limit Theorem that for large n

$$\overline{X}_n \approx N\left(\frac{1}{\lambda}, \frac{1}{\lambda^2 n}\right).$$

By the method of moments, we see that the Method of Moment Estimator for λ is $\frac{1}{\overline{X}_n} = \psi(\overline{X}_n)$ (say).

By the first order Taylor's approximation of $\psi(\overline{X}_n)$ about $\frac{1}{\lambda}$, we obtain

$$\psi(\overline{X}_n) \approx \psi\left(\frac{1}{\lambda}\right) + \left(\overline{X}_n - \frac{1}{\lambda}\right)\psi'\left(\frac{1}{\lambda}\right).$$

Now taking expectation on both sides

$$\mathbb{E}\left[\psi(\overline{X}_n)\right] \approx \psi\left(\frac{1}{\lambda}\right) = \lambda.$$

For computing the variance, we see that

$$\mathbb{E}\left[\psi(\overline{X}_n) - \lambda\right]^2 \approx \mathbb{E}\left[\left(\overline{X}_n - \frac{1}{\lambda}\right)\psi'\left(\frac{1}{\lambda}\right)\right]^2 = \mathrm{Var}_\lambda(\overline{X}_n)\lambda^4 = \frac{\lambda^2}{n},$$

provided $\psi'\left(\frac{1}{\lambda}\right) \neq 0$. Therefore, the approximately

$$\operatorname{Var}\left(\frac{1}{\overline{X}_n}\right) \approx \frac{\lambda^2}{n}.$$

Let us verify the same using computer simulation. In addition, we overlay a normal density function with mean λ and variance $\frac{\lambda^2}{n}$ on the histograms for different values of n. The histograms are simulated using 1000 replications.

In [1]:
```julia
using Statistics, StatsBase, StatsModels
using Plots, Distributions, LaTeXStrings, Random
using DataFrames, LinearAlgebra
```

In [2]:
```julia
lambda = 3
n_vals = [3, 5, 10, 25, 50, 100]
rep = 1000
plt = plot(layout=(2, 3), size=(800, 600))

for (idx, n) in enumerate(n_vals)
    psi_xbar = zeros(rep)
    for i in 1:rep
        x = rand(Exponential(1/lambda), n)
        psi_xbar[i] = 1/mean(x)
    end
    histogram!(psi_xbar, bins=:auto, normalize=true,
        xlabel=L"\psi(\bar{X_n})", xlims=extrema(psi_xbar),
        ylabel="Density", title="n = $n", subplot=idx , legend=false)

    plot!(x -> pdf(Normal(lambda, sqrt(lambda^2/n)), x), color="red",
        lw=2, subplot=idx)
end

display(plt)
```

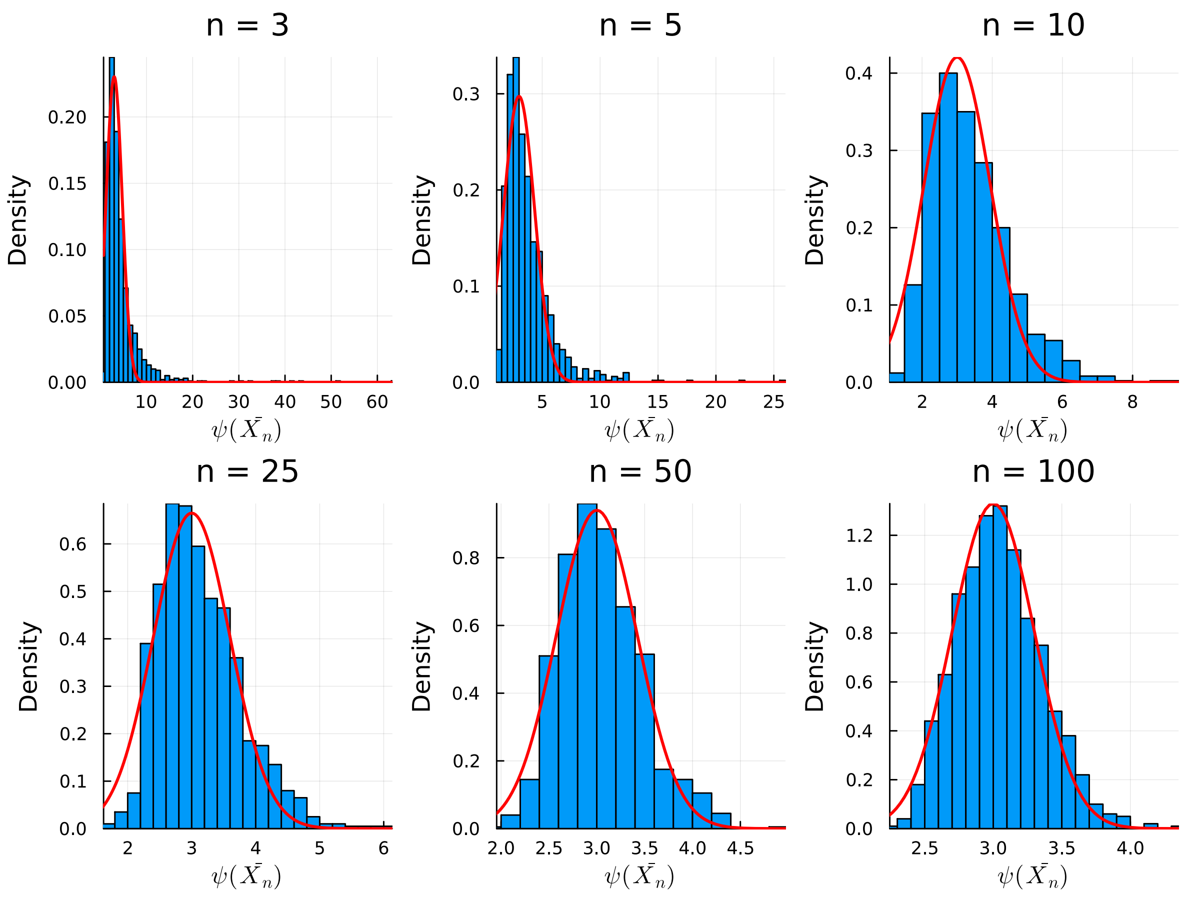

Figure 9: As the sample size increases,the sampling distribution of the function of the sample mean is well approximated by a normal distribution. This approximation is known as the Delta method. The histograms are obtained by repeatedly sampling 1000 times from the exponential distribution with rate parameter $\lambda = 3$.

Bootstrapping regression model

In the following code, we investigate how we can compute the standard of the estimate using the non parametric bootstrap procedure

In [3]:
```
a = 1
b = 0.4
age = collect(1:0.1:20)
Random.seed!(123)
size_0 = a .* age .^ b .+ rand(Normal(0, 0.2), length(age))
scatter(age, size_0, color = "darkgrey", markersize = 6,
xlabel = "age", ylabel = "size", label = "")
```

Out[3]:

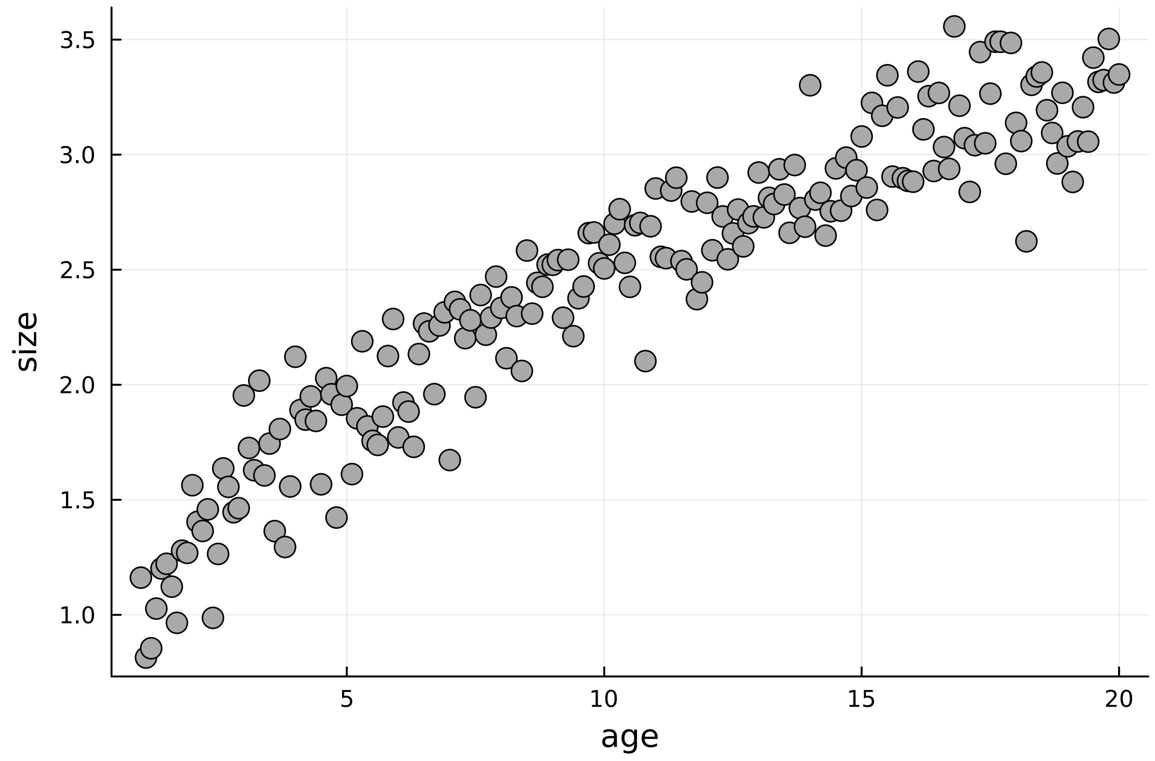

Figure 10: We fixed the Random.seed!(123) and simulate the data sets from the population nonlinear regression model. The parameters are fixed as $a = 1$ and $b = 0.4$ and the errors are assumed to be normally distributed. This data set will be resampled to obtain the bootrstrap estimate of the parmaeters and also compute the bootstrap standard error of the estimates.

In [5]:
```
using LsqFit
data = DataFrame(age = age, size_0 = size_0)
B = 1000  # Number of bootstrap samples
a_hat = zeros(B)
b_hat = zeros(B)

for i in 1:B
    ind = StatsBase.sample(1:nrow(data), nrow(data), replace=true)
    boot_data = data[ind, :]

    # Define the nonlinear model function
    model(age, p) = p[1] .* age .^ p[2]

    # Initial parameter estimates
    p0 = [0.5, 1.0]

    # Fit the model using bootstrapped data
    boot_fit = curve_fit(model, boot_data.age, boot_data.size_0, p0)

    # Store estimated parameters
    a_hat[i] = coef(boot_fit)[1]
    b_hat[i] = coef(boot_fit)[2]
end
```

In [6]:
```
mean(a_hat)   # bootstrap mean
```

Out[6]:
```
0.9867796819706952
```

In [7]:
```
std(a_hat)   # bootstrap standard error of a_hat
```

```
Out[7]:  0.02623223645603711
```

```
In [8]:  mean(b_hat)
```

```
Out[8]:  0.40473507899038036
```

```
In [9]:  std(b_hat)
```

```
Out[9]:  0.010733616600425952
```

We nowv isualize the sampling distribution of $\widehat{a}$ and $\hat{b}$ based on B = 1000$ bootstrap samples.

```
In [10]: p1 = histogram(a_hat, normalize = true, xlabel = L"\widehat{a}",
         ylabel = "density", title = "B = 1000", label = "")
         println("95% bootstrap CI for a based on normal distribution is given as \n")
```

```
95% bootstrap CI for a based on normal distribution is given as
```

```
In [11]: mean(a_hat)- 1.96*std(a_hat), mean(a_hat) + 1.96*std(a_hat)
```

```
Out[11]:  (0.9353644985168624, 1.0381948654245279)
```

```
In [12]: println("nonparametric 95% CI for a is")
```

```
nonparametric 95% CI for a is
```

```
In [13]: quantile(a_hat, [2.5, 97.5] ./100)
```

```
Out[13]:  2-element Vector{Float64}:
           0.9345860167313804
           1.037376553479089
```

```
In [14]: p2 = histogram(b_hat, normalize = true, xlabel = L"\widehat{b}",
         ylabel = "density", title = "B = 1000", label = "")
         println("95% bootstrap CI for a based on normal distribution is given as \n")
```

```
95% bootstrap CI for a based on normal distribution is given as
```

```
In [15]: mean(b_hat)- 1.96*std(b_hat), mean(b_hat) + 1.96*std(b_hat)
```

```
Out[15]:  (0.3836971904535455, 0.4257729675272152)
```

```
In [16]: println("nonparametric 95% CI for b is")
```

```
nonparametric 95% CI for b is
```

```
In [17]: quantile(b_hat, [2.5 , 97.5] ./100)
```

```
Out[17]:  2-element Vector{Float64}:
           0.38407540358654213
           0.4276865458917835
```

```
In [18]: plot(p1, p2, layout = (1,2), size = (800, 500))
```

Out[18]:

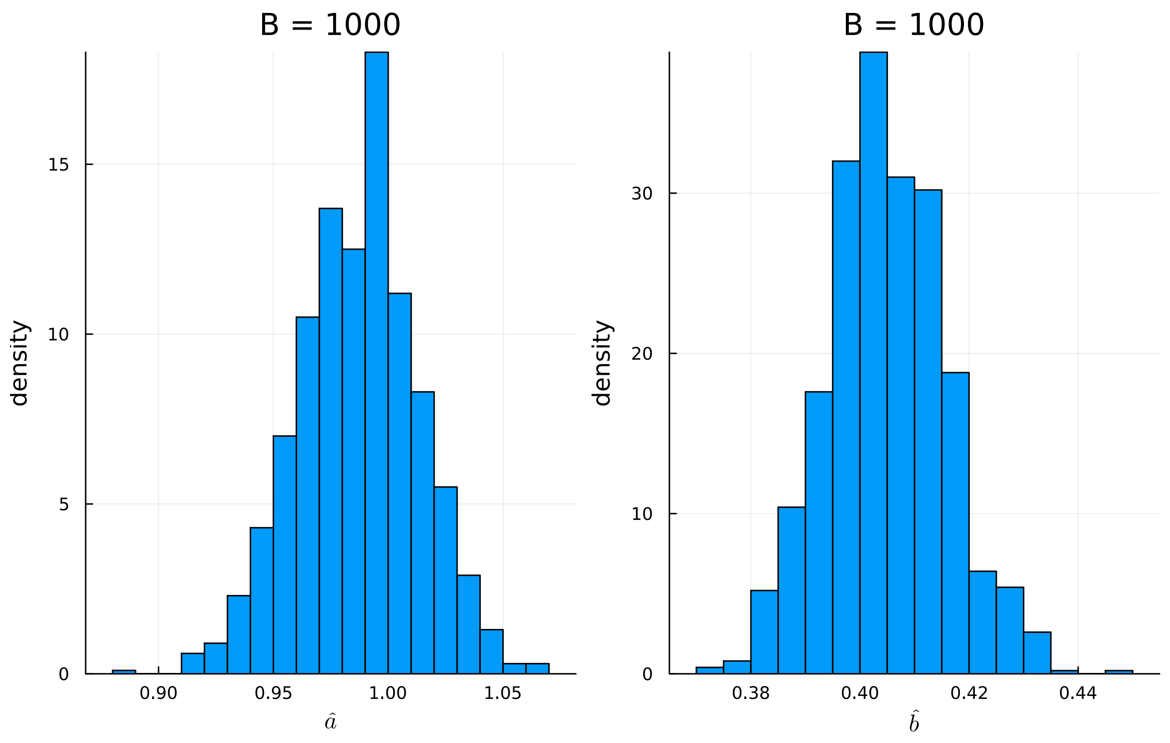

Figure 11: The bootstrap sampling distribution of $\hat{a}$ and $\hat{b}$ based on 1000 bootstrap samples.It may be noted that the bootstarp estimates are centered about the nls estimates of the parmaeters a and b based on the complete data sets, not centered around the true values of a and b using which the data has been simulated.

Case study using synthetic data generation

Suppose that we have the population regression function $f(x|\mathbf{b})$ parameterized by $\mathbf{b} = (b_0, b_1)'$ and the statistical model for the data is given by

$$y_i = f(x|\mathbf{b}) + \epsilon_i = \frac{b_0 x}{b_1 + x} + \epsilon_i, \quad i \in \{1, 2, \ldots, n\}.$$

We assume that the errors ϵ_is are normally distributed with mean 0 and variance σ^2 and they are also independent. Statisticians formally call it IID (Independent and Identically Distributed). In the following code, we first simulate the synthetic data by fixing the population parameters b_0, b_1, σ^2.

In [1]:
```
using Statistics, StatsBase, StatsModels
using Plots, Distributions, LaTeXStrings, Random
using DataFrames, LinearAlgebra
using LsqFit
```

In [2]:
```
# plot the mean function
Random.seed!(1234)
b0 = 1
b1 = 0.5

f(x) =  b0*x/(b1+x)  # define the function

p1 = plot(f, 0, 2.5, color = "red", lw = 2 , xlabel = "x",
ylabel = "f(x)", label = "")
sigma2 = 0.01  # variance
x = collect(0:0.1:2)
println(x)
```

```
[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6,
1.7, 1.8, 1.9, 2.0]
```

In [3]:
```
n = length(x)  # Lenngth of data
y = b0 .*x ./ (b1 .+ x) + rand(Normal(0, sqrt(sigma2)),n)
println(y)
```

```
[0.09706563288552145, 0.0687448255131467, 0.37590037407369514, 0.371719687075536, 0.3
8436522210888835, 0.3554822884713767, 0.8161969396261235, 0.7357781196768929, 0.69136
5017385362, 0.5547134522374947, 0.7372659800449276, 0.796655528205341, 0.793032138229
2673, 0.7307913331228599, 0.8328500204119117, 0.8407836998936501, 0.7413417243019591,
0.8497975902363936, 0.6612146785351385, 0.9008165248735148, 0.9220043796712128]
```

In [4]:
```
p2 = scatter(x, y, color = "grey", markersize = 8,xlabel = "x", ylabel = "y",
label = "")
data = Matrix{Float64}(undef, 5, length(x))

# Simulate data
for i in 1:size(data, 1)
    data[i, :] .= b0 .* x ./ (b1 .+ x) .+ randn(n) .* sqrt(sigma2)
end

# Plot data
p3 = scatter(x, data', marker=:circle, label="", xlabel="x", ylabel="y")

plot(p1, p2, p3, layout = (1,3), size = (800, 400))
```

Out[4]:

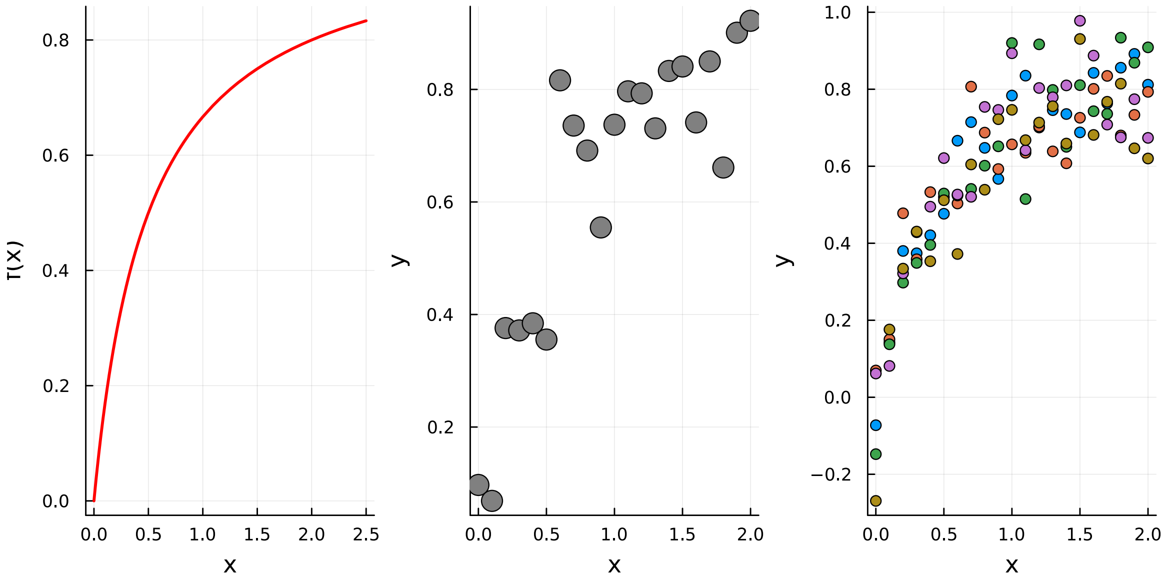

Figure 12: The leftmost panel is the mean population regression function, and the right panel contains the simulated data using the seed value 1234. The seed value is used to ensure the reproducibility of the plots. For simulation, the parameter choices are set as $b_0 = 1$, $b_1 = 0.5$, $\sigma^2 = 0.01$. In the rightmost panel, some more simulation has been carried out to demonstrate the randomness across different simulated data sets.

We consider the minimization of the error sum of squares as the first approach to estimate the parameters. We minimize the following function with respect to b_0 and b_1.

$$\mathrm{ESS}(\mathbf{b}) = \sum_{i=1}^{n} \left(y_i - \frac{b_0 x_i}{b_1 + x_i} \right)^2.$$

We first plot the surface of the $\mathrm{ESS}(\mathbf{b})$ with different choices of b_0 and b_1. For the user, show the plots using the surface() function from the Plots.jl package and also by using the PlotlyJS.jl package.

In [5]:
```julia
function fun_ESS(b)
     b0 = b[1]
     b1 = b[2]
     sum((y- b0 .*x ./ (b1 .+ x)) .^2)
end

b0_vals = collect(0.01:0.05:2)
b1_vals = collect(0.01:0.05:2)

ESS_vals = Matrix{Float64}(undef, length(b0_vals), length(b1_vals))

for i in 1:length(b0_vals)
    for j in 1:length(b1_vals)
        ESS_vals[i, j] = fun_ESS([b0_vals[i], b1_vals[j]])
    end
end
```

In [6]:
```julia
surface(b0_vals, b1_vals, ESS_vals',xlabel="b0", ylabel="b1",
    zlabel="ESS",title="Error Sum of Squares (ESS) Surface",
    camera=(50, 40))
```

Out[6]:

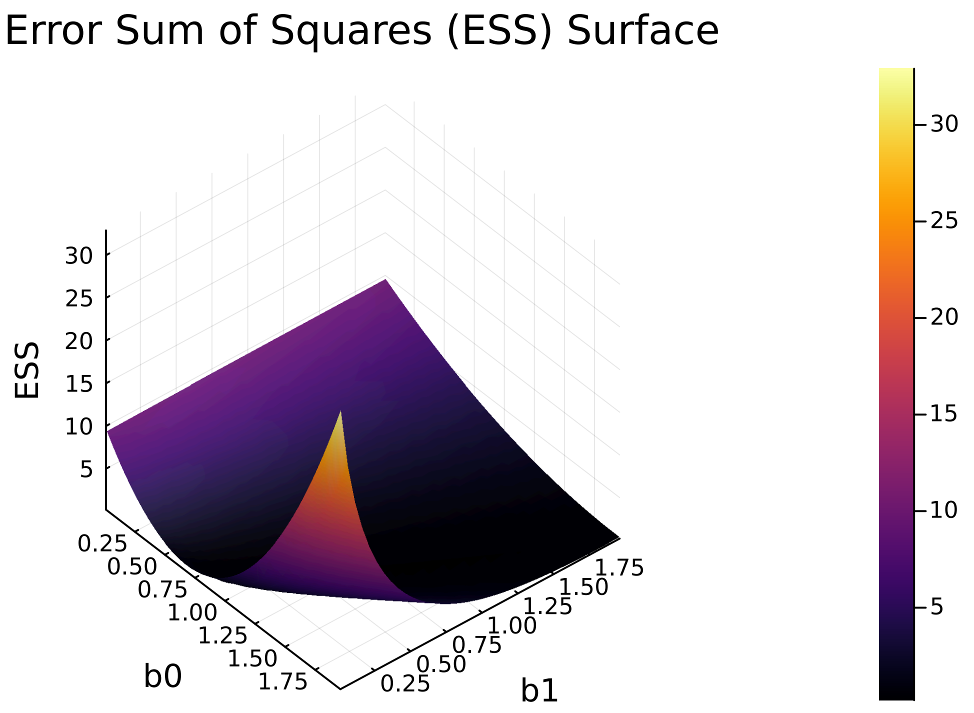

Figure13: The surface plot is generated using the Plots.surface function.

In the following code,we use the function `Optim.optimize()` to minimize the error sum squares function with respect to b_0 and b_1.

In [7]:
```
using Optim
using ForwardDiff  # for computing Hessian matrix
using LinearAlgebra
```

In [8]:
```
out = optimize(fun_ESS, [1.5, 1])
```

Out[8]:
```
 * Status: success

 * Candidate solution
    Final objective value:     2.079164e-01

 * Found with
    Algorithm:     Nelder-Mead

 * Convergence measures
    √(Σ(yᵢ-ȳ)²)/n ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    31
    f(x) calls:    62
```

In [9]:
```
# Extract the estimated parameters
b_hat = Optim.minimizer(out)

# Print the estimates
println(b_hat)
```

```
[1.090271115059162, 0.5270048257244626]
```

In the following, we obtain the estimates of the parameter by using the method of maximum likelihood. The likelihood function is given by

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} f(y_i|x_i; b_0, b_1, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}\left(y_i - \frac{b_0 x_i}{b_1 + x_i}\right)^2\right)$$

which simplifies to

$$\mathcal{L}(b_0, b_1, \sigma^2) = \left(\frac{1}{\sigma^2 2\pi}\right)^{n/2} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i - \frac{b_0 x_i}{b_1 + x_i}\right)^2\right).$$

We write down the log-likelihood function as

$$l(b_0, b_1, \sigma^2) = -\frac{n}{2}\log(\sigma^2) - \frac{n}{2}\log(2\pi) - \sum_{i=1}^{n}\left(y_i - \frac{b_0 x_i}{b_1 + x_i}\right)^2.$$

Instead of maximizing the log-likelihood function, we can minimize the negative log-likelihood function $-l(b_0, b_1, \sigma^2)$.

In [10]:
```
# Maximum Likelihood Estimation
n = length(x)

# Log-likelihood function
function fun_logLik(b)
```

```
    b0 = b[1]
    b1 = b[2]
    sigma2 = exp(b[3])  # Ensure sigma2 is always positive
    -(n/2) * log(sigma2) - (n/2) * log(2*pi) -
    (1/(2*sigma2)) * sum((y .- (b0 .* x) ./ (b1 .+ x)).^2)
end

# Negative log-likelihood function
function fun_neglogLik(b)
    b0 = b[1]
    b1 = b[2]
    sigma2 = exp(b[3])  # Ensure sigma2 is always positive
    (n/2) * log(sigma2) + (n/2) * log(2*pi) +
    (1/(2*sigma2)) * sum((y .- (b0 .* x) ./ (b1 .+ x)).^2)
end

# Optimize using Nelder-Mead
out = optimize(fun_neglogLik, [1, 1, log(0.1)], NelderMead())

# Extract parameter estimates
b_hat = Optim.minimizer(out)
b0_hat = b_hat[1]
b1_hat = b_hat[2]
sigma2_hat = exp(b_hat[3])  # Convert back from log-scale

# Print estimates
println("Estimated parameters: b0 = ", b0_hat, ", b1 = ",
    b1_hat, ", sigma2 = ", sigma2_hat)
```

```
Estimated parameters: b0 = 1.090314309777411, b1 = 0.5270640202033637, sigma2 = 0.009
900537229017589
```

The estimate that we have obtained by employing the method of maximum likelihood is subject to uncertainty due to the random nature of the data set. Therefore, we need to report the uncertainty or the standard error of the estimate. We omit the following result without proof which states that for large sample size the variance-covariance matrix of $\hat{b}_0$, $\hat{b}_1$, and $\hat{\sigma}^2$ is well approximated by the inverse of the Hessian matrix evaluated at the MLE with a negative sign.

$$-H = -\begin{bmatrix} \frac{\partial^2 l}{\partial b_0^2} & \frac{\partial^2 l}{\partial b_0 \partial b_1} & \frac{\partial^2 l}{\partial b_0 \partial \sigma^2} \\ \frac{\partial^2 l}{\partial b_1 \partial b_0} & \frac{\partial^2 l}{\partial b_1^2} & \frac{\partial^2 l}{\partial b_1 \partial \sigma^2} \\ \frac{\partial^2 l}{\partial \sigma^2 \partial b_0} & \frac{\partial^2 l}{\partial \sigma^2 \partial b_1} & \frac{\partial^2 l}{\partial (\sigma^2)^2} \end{bmatrix} \Bigg|_{(\hat{b}_0, \hat{b}_1, \hat{\sigma}^2)}$$

In the `optimize` function from the `Optim.jl` package, we can use the argument `autodiff = :forward` to obtain the estimated Hessian matrix at the MLE.

In [11]:
```
out = optimize(fun_neglogLik, [1, 1, log(0.1)], NelderMead(),
autodiff = :forward)
b_hat = out.minimizer
```

Out[11]:
```
3-element Vector{Float64}:
  1.090314309777411
  0.5270640202033637
 -4.615166257756583
```

In [12]:
```
H = ForwardDiff.hessian(fun_neglogLik, b_hat)
```

Out[12]:
```
3×3 Matrix{Float64}:
  804.214        -523.323         0.000472918
 -523.323         385.648        -0.000263371
    0.000472918    -0.000263371  10.5003
```

In [13]:
```
println("The Hessian matrix evaluate at the MLE is given by ")
```

```
The Hessian matrix evaluate at the MLE is given by
```

In [14]:
```
H
```

Out[14]:
```
3×3 Matrix{Float64}:
  804.214        -523.323         0.000472918
 -523.323         385.648        -0.000263371
    0.000472918    -0.000263371  10.5003
```

In [15]:
```
println("The inverse of the Hessian matrix with negative sign is given by")
```

```
The inverse of the Hessian matrix with negative sign is given by
```

In [16]:
```
inv(H)
```

Out[16]:
```
3×3 Matrix{Float64}:
  0.0106308    0.014426    -1.1696e-7
  0.014426     0.022169    -9.36755e-8
 -1.1696e-7  -9.36755e-8   0.0952358
```

Therefore, the square root of the diagonal entries of the matrix $(-H)^{-1}$ will give the standard error of the MLE. A natural question arises: how good these approximations are? To see this, we can visualize the sampling distribution of

$$W_0 = \frac{\hat{b}_0 - b_0}{\widehat{SE}(\hat{b}_0)}$$

and

$$W_1 = \frac{\hat{b}_1 - b_1}{\widehat{SE}(\hat{b}_1)}$$

by computer simulation based on 1000 replications.

In [12]:
```
using Optim, ForwardDiff, LinearAlgebra
using Distributions, Statistics, Plots
using LaTeXStrings, Distributions
```

In [18]:
```
rep = 1000
w_0 = zeros(rep)
w_1 = zeros(rep)


for i in 1:rep
    x = collect(0:0.05:2)
    n = length(x)
    y = (b0 .* x ./ (b1 .+ x)) .+ rand(Normal(0, sigma2_hat), n)

    out = optimize(fun_neglogLik, [1, 1, log(0.1)], NelderMead(),
          autodiff = :forward)
    b_hat = out.minimizer
    H = ForwardDiff.hessian(fun_negloglik, b_hat)
    H_inv = inv(H)

    w_0[i] = (b_hat[1] - b0) / sqrt(H_inv[1,1])
    w_1[i] = (b_hat[2] - b1) / sqrt(H_inv[2,2])
end


p1 = Plots.histogram(w_0, normalize=true, xlabel=L"W_0", ylabel="Density",
                title="n = $n", legend=false, bins=30)
plot!(x -> pdf(Normal(0,1), x), color="red", lw=2)


p2 = Plots.histogram(w_1, normalize=true, xlabel=L"W_1", ylabel="Density",
                title="n = $n", legend=false, bins=30)
plot!(x -> pdf(Normal(0,1), x), color="red", lw=2)

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

Out[18]:

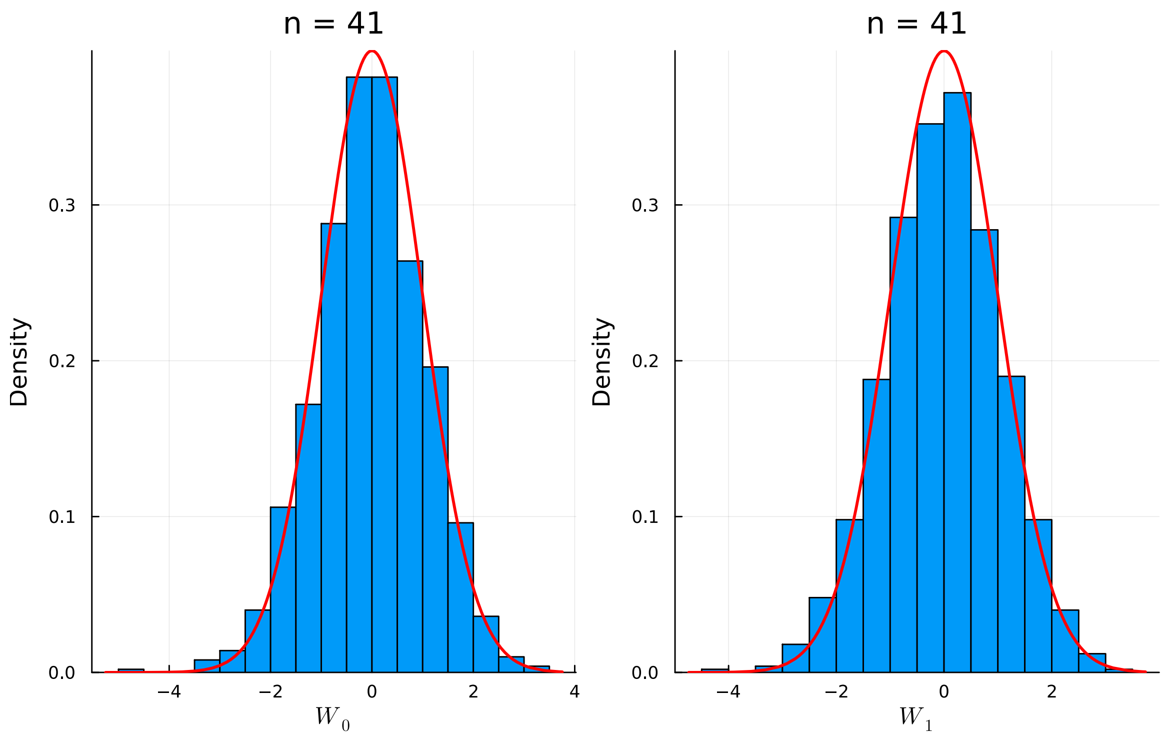

Figure 14: The sampling distribution of the estimator centered at the true value and scaled by the estimated standard error. An approximation with the standard normal distribution is shown for reference.

The simulations suggest that for large sample size n,

$$W_0 = \frac{\hat{b}_0 - b_0}{\widehat{SE}(\hat{b}_0)} \sim \mathcal{N}(0,1),$$

therefore,

$$\hat{b}_0 \sim \mathcal{N}\left(b_0, \widehat{SE}(\hat{b}_0)^2\right),$$

for large sample size n. Therefore, for large n, a $(1-\alpha)\%$ confidence interval for b_0 can be obtained as

$$\left(\hat{b}_0 - z_{\alpha/2}\widehat{SE}(\hat{b}_0), \hat{b}_0 + z_{\alpha/2}\widehat{SE}(\hat{b}_0)\right),$$

where $P(Z > z_{\alpha/2}) = \frac{\alpha}{2}$ and $Z \sim \mathcal{N}(0,1)$.

Case Study: Local Maximization

In a classroom setting, a nonlinear function was randomly suggested to be used as the systematic component of a nonlinear regression model for data simulation. The proposed function is:

$$f(x) = \sin(ax) + bx^2, \quad x \in [0,2]$$

Thus, the statistical modeling framework is given by:

$$y_i = f(x_i) + \epsilon_i, \quad i \in \{1, 2, \ldots, n\}$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. Data is simulated from this regression model, and the model parameters a, b, σ^2 are estimated using the method of maximum likelihood.
In the first step, a sample of size n is simulated by fixing the parameters at $a = 3$, $b = 1$, and $\sigma^2 = 0.05$. The parameter vector is denoted as:

$$\theta = (a, b, \sigma^2)'$$

In [1]:
```
using Statistics, Plots, Distributions
using Random, StatsBase, LaTeXStrings
using Optim, ForwardDiff, LinearAlgebra
```

In [2]:
```
# the mean regression function
a = 3
b = 1

# define the function
f(x) = sin(a .*x) .+ b .*x .^2  # population non linear regression function

n = 30  # sample size
sigma2 = 0.1 # population standard  deviation
x = range(0,2, length = n)
y = zeros(n)

for i in 1:n
    y[i] = rand(Normal(f(x[i]), sqrt(sigma2)))
end
```

In [3]:
```
print(y)
```

```
[0.018812102153547946, -0.09569179210896156, 0.6174695533670822, 0.7865425092488416,
0.9374414825092556, 1.308193998511752, 1.6166235662988986, 1.037118242364362, 1.05871
5121530936, 1.2583780048928568, 0.7279984720481907, 1.5277696829855154, 1.14884298742
45383, 1.079823290597151, 1.3061404026954961, 1.3319947118712756, 1.4653180835536153,
0.6314509729567517, 0.5234421947580528, 0.7766756783001698, 0.7273504489343715, 1.082
2699632387807, 1.4324199260663595, 1.4583354330487905, 1.2681319499166381, 1.66940656
50676733, 2.5075193449472417, 2.8423147016021417, 3.039516700477734, 3.65273428017273
1]
```

In [4]:
```
scatter(x,y, color = "darkgrey", markersize = 6, xlabel = "x",
ylabel = "y", legend = false)
plot!(f, color = "red", lw = 2)
```

Out[4]:

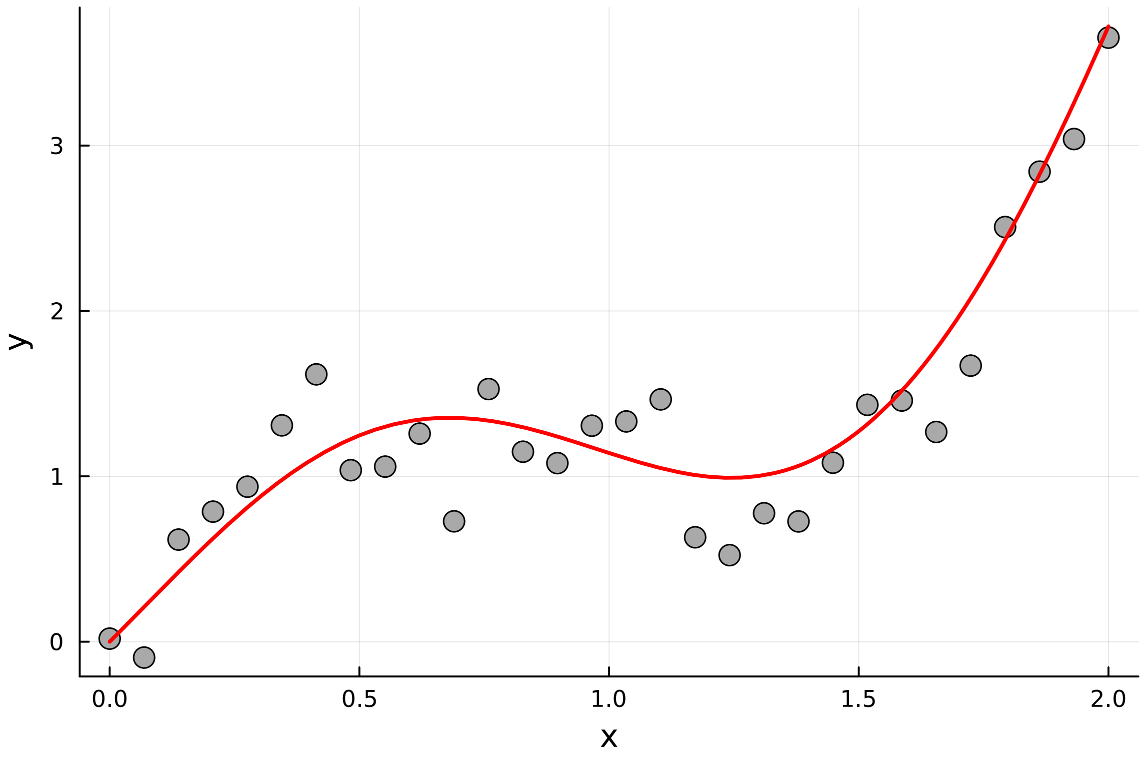

Figure 15: The simulated data set and the population regression mean function is overlaid for reference.

In the following Julia codes, we minimize the negative of the log-likelihood function. The reader is encouraged to write down the expression of the likelihood function explicitly and plot the surface of the likelihood function for different choices of a and b.

In [5]:
```
function fun_neglogLik(theta)
    a = theta[1]
    b = theta[2]
    sigma2 = theta[3]
    (n / 2) * log(sigma2) + (n / 2) * log(2 * pi) +
    (1 / (2 * sigma2)) * sum((y .- sin.(a .* x) .- b .* x .^ 2) .^ 2)
end

out = optimize(fun_neglogLik, [2.8,0.9,0.04], NelderMead(),
          autodiff = :forward)
```

Out[5]:
```
 * Status: success

 * Candidate solution
    Final objective value:     2.401039e+00

 * Found with
    Algorithm:     Nelder-Mead

 * Convergence measures
    √(Σ(yᵢ-ȳ)²)/n ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    73
    f(x) calls:    139
```

the population nonlinear regression function and the estimated (sample) nonlinear regression function for reference. The estimated regression function is given by

$$f(x) = sin(\hat{a}x) + bx^2$$

```
In [6]: param = out.minimizer
        a_hat = param[1]
        b_hat = param[2]
        sigma2_hat = param[3]

        println("a_hat: $a_hat")
        println("b_hat: $b_hat")
        println("sigma2_hat: $sigma2_hat")
```

```
a_hat: 3.049082245019311
b_hat: 0.9407317906362939
sigma2_hat: 0.06871395839938724
```

```
In [7]: scatter(x,y, color = "darkgrey", markersize = 6, xlabel = "x",
        ylabel = "y", label = "")
        plot!(f, color = "red", lw = 2, label = L"f(x)")
        plot!(x->sin(a_hat*x) + b_hat*x^2, color = "blue", lw = 2,
            label = L"\widehat{f(x)}")
```

Out[7]:

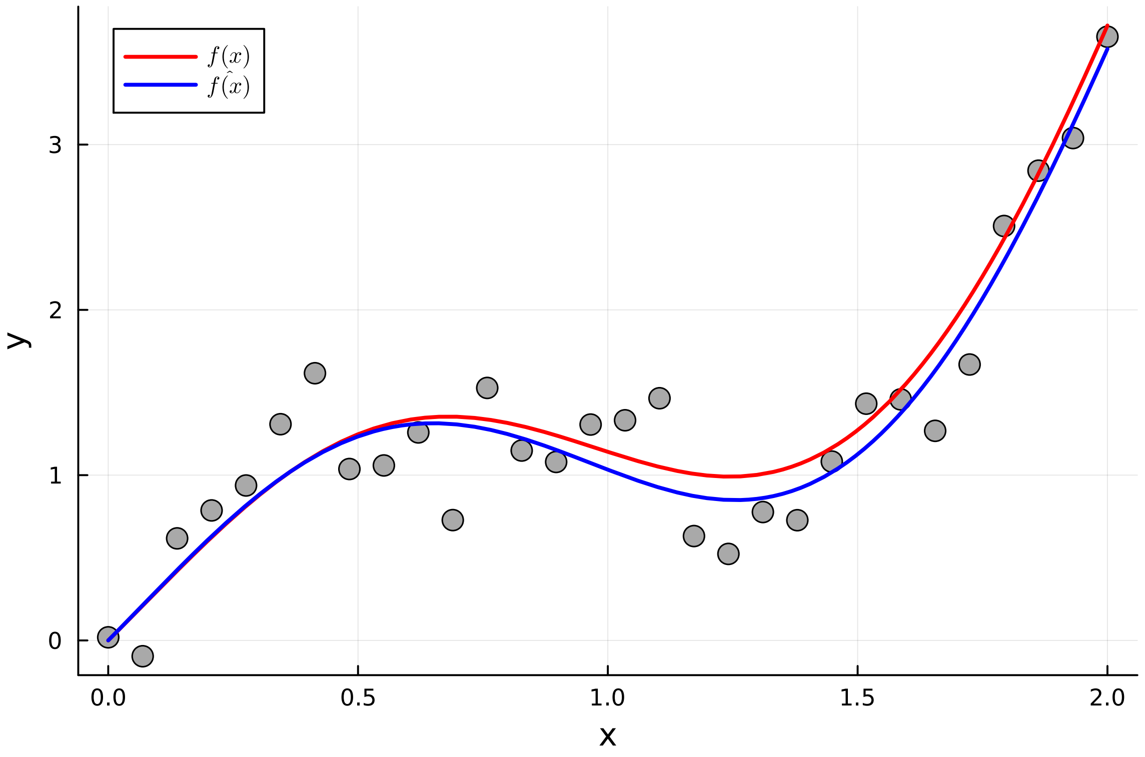

Figure 16: The estimated regression function and the population regression function is shown using different colour and they are in good

Certainly, the above estimates are subject to uncertainty. Basically, if we simulate another set of data and compute the MLE, these estimates will certainly differ. However, we want to understand how much they can vary. To obtain this information, the Hessian matrix (H) will be useful. The estimate of the variance of the MLE of $\hat{a}$ is given by H^{-1}_{11}, evaluated at $\hat{\theta}$.

```
In [8]: H = ForwardDiff.hessian(fun_negloglik, param)
```

```
Out[8]: 3×3 Matrix{Float64}:
          303.876       241.137        -0.0177521
          241.137      1469.63         -0.0306426
           -0.0177521    -0.0306426   3176.85
```

In [9]:
```
inv(H)
```

Out[9]:
```
3×3 Matrix{Float64}:
  0.00378344   -0.000620788  1.51538e-8
 -0.000620788   0.000782304  4.07684e-9
  1.51538e-8    4.07684e-9   0.000314777
```

To understand the uncertainty associated with these estimates, first, we simulate the sampling distribution of $\hat{\theta}$ by repeating the simulation experiment M times. We simulate the sampling distribution of $\hat{a}$, $\hat{b}$, and $\hat{\sigma}^2$.
We also compute the following three quantities:

$$W_a = \frac{\hat{a} - a}{\mathrm{SE}(\hat{a})}, \quad W_b = \frac{\hat{b} - b}{\mathrm{SE}(\hat{b})}, \quad W_{\sigma^2} = \frac{(n-2)\hat{\sigma}^2}{\sigma^2}.$$

In [10]:
```
M = 1000
a_hat = zeros(M)
b_hat = zeros(M)
sigma2_hat = zeros(M)
se_a_hat = zeros(M)
se_b_hat = zeros(M)
se_sigma2_hat = zeros(M)

for j in 1:M
    x = range(0, stop=2, length=n)
    y = zeros(n)

    for i in 1:n
        y[i] = rand(Normal(f(x[i]), sqrt(sigma2)))
    end

    out = optimize(fun_neglogLik, [2.8, 0.9, 0.04], NelderMead())

    a_hat[j] = out.minimizer[1]
    b_hat[j] = out.minimizer[2]
    sigma2_hat[j] = out.minimizer[3]

    H = ForwardDiff.hessian(fun_neglogLik, out.minimizer)

    se_a_hat[j] = sqrt(inv(H)[1, 1])
    se_b_hat[j] = sqrt(inv(H)[2, 2])
    se_sigma2_hat[j] = sqrt(inv(H)[3, 3])
end

gr()
plot(layout=(2,3))

p1 = histogram(a_hat, normalize = true, xlabel=L"\hat{a}", title="n = $n",
legend = false)
scatter!([a], [0], markersize=5, color=:red)

p2 = histogram(b_hat, normalize = true, xlabel=L"\hat{b}", title="n = $n",
legend = false)
scatter!([b], [0], markersize=5, color=:red)

p3 = histogram(sigma2_hat, normalize=true, xlabel=L"\hat{\sigma}^2",
    title="n = $n", legend = false)
scatter!([sigma2], [0], markersize=5, color=:red)
```

```
p4 = histogram((a_hat .- a) ./ se_a_hat, normalize=true,
    xlabel=L"\frac{\hat{a} - a}{SE(\hat{a})}", title="n = $n",
    legend = false)
plot!(x -> pdf(Normal(0,1), x), lw=2, color=:red)

p5 = histogram((b_hat .- b) ./ se_b_hat, normed=true,
    xlabel=L"\frac{\hat{b} - b}{SE(\hat{b})}", title="n = $n",
    legend = false)
plot!(x -> pdf(Normal(0,1), x), lw=2, color=:red)

p6 = histogram((n-2) * sigma2_hat ./ sigma2, normed=true,
     xlabel=L"\frac{(n-2) \hat{\sigma}^2}{\sigma^2}", title="n = $n",
     legend = false)
plot!(x -> pdf(Chisq(n-2), x), lw=2, color=:red)

plot(p1,p2,p3,p4,p5,p6, layout = (2,3), size = (900, 600))
```

Out[10]:

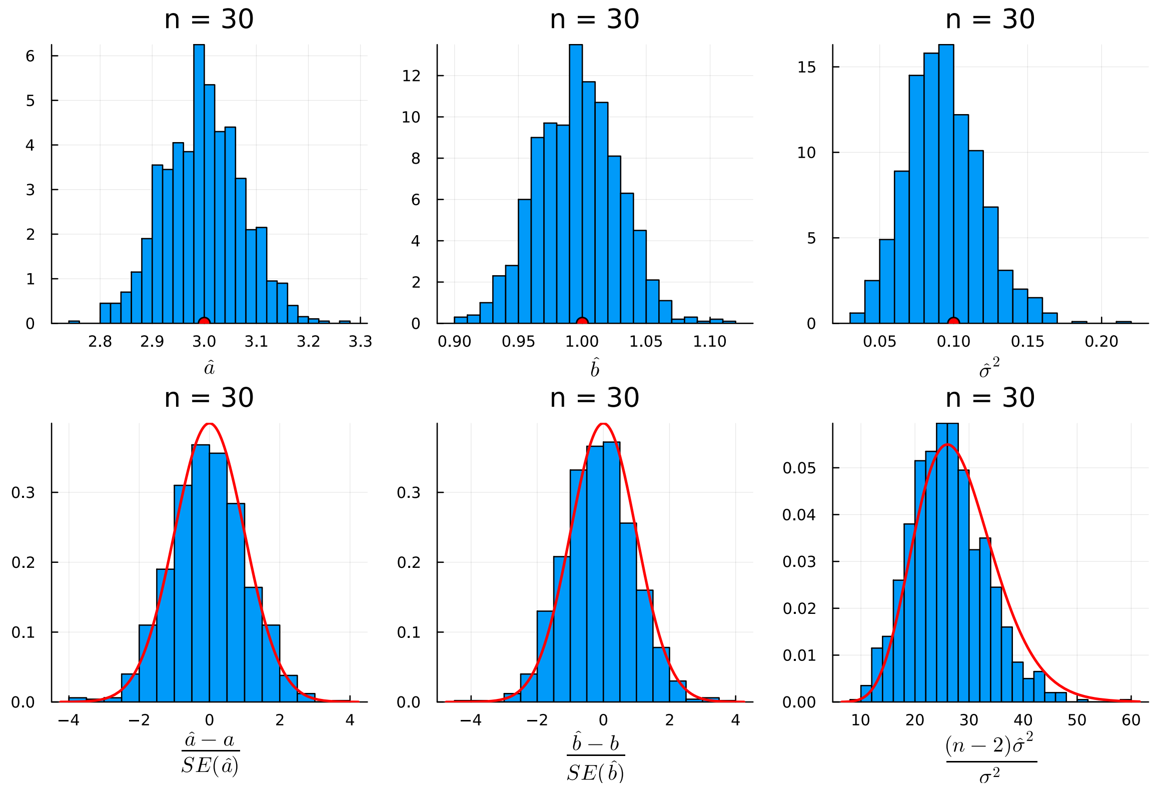

Figure 17: The top panel represents the simulated sampling distribution of the MLEs based on $M = 1000$ replications. The over red dot indicates the true value of the parameters using which the training data sets have been simulated. It is interesting to see that MLEs are centered about the true value, ensuring unbiasedness of the MLE. In the lower panel, the simulated distributions of W_a, W_b, and W_{σ^2} are drawn. The standard normal distribution is overlaid on the first two histograms (bottom panel), and the last figure is overlaid with a χ^2 distribution with $(n-2)$ degrees of freedom.

In [11]:

```
using DataFrames
theta_hat = hcat(a_hat, b_hat, sigma2_hat)
theta_hat = DataFrame(theta_hat, [:a_hat, :b_hat, :sigma2_hat])
first(theta_hat, 6)
```

Out[11]: 6×3 DataFrame

Row	a_hat	b_hat	sigma2_hat
	Float64	Float64	Float64
1	3.05724	0.994951	0.137901
2	2.97046	0.964989	0.0800547
3	3.00901	0.96112	0.0980362
4	2.92913	0.981682	0.0910456
5	2.88379	1.00479	0.121578
6	3.08076	1.03918	0.0671051

In [12]:
```
using StatsPlots, PairPlots, CairoMakie
pairplot(theta_hat, fullgrid=true)
```

Out[12]:

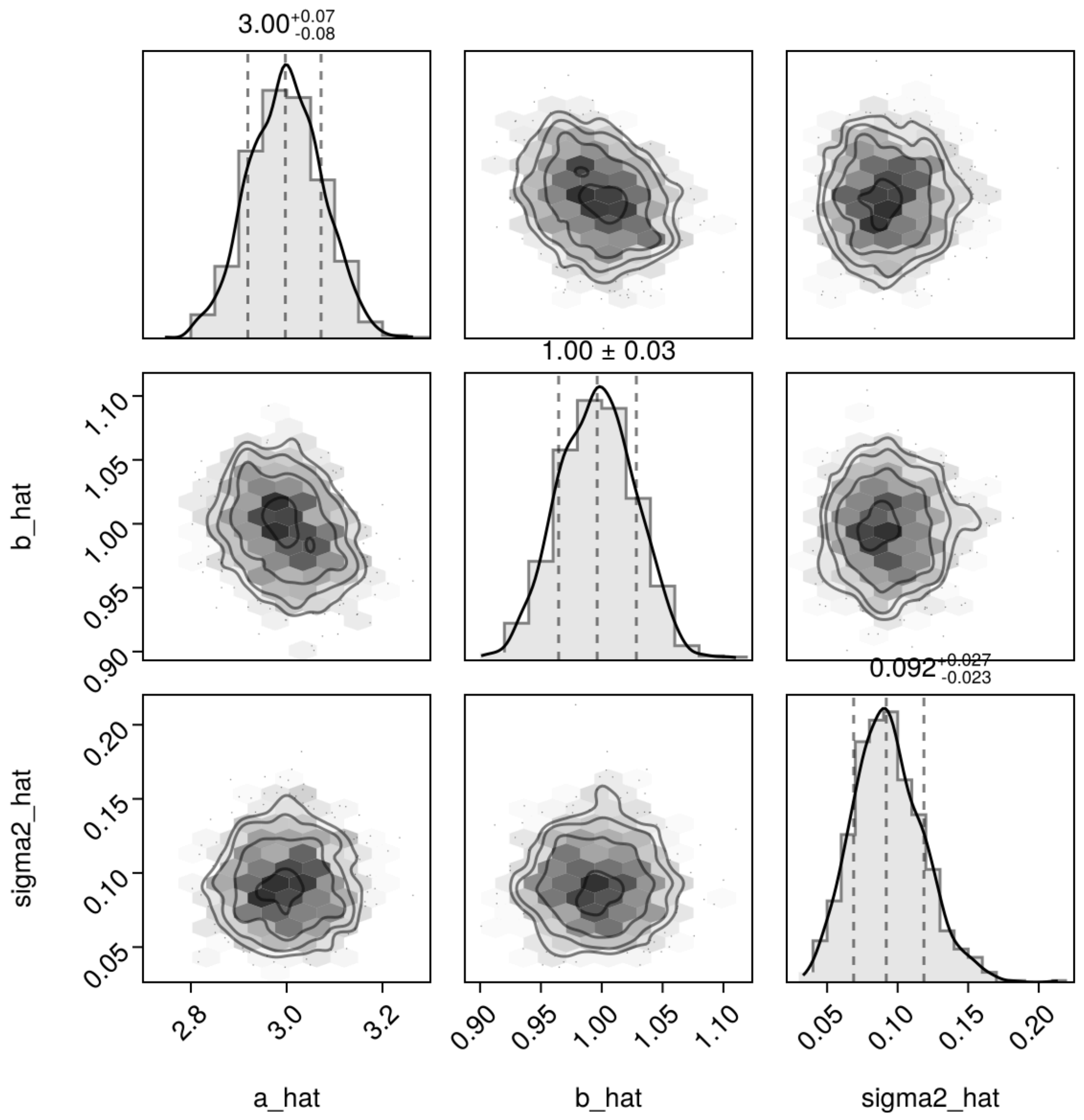

Figure 18: The pairs plot indicates a small negative correlations between $\widehat{a}$ and $\hat{b}$, where as,these estimators are independent with $\widehat{\sigma}^2$

Sensitivity to the Initial Conditions

In the following, we visualize the error sum of squares as a function a and b. The surface clearly demonstrates the existence of multiple local minimum and depending upon the initial conditions different minimum will be achieved

In [13]:
```
# Define the function
fun_ESS = function(a, b)
    sum((y .- sin.(a .* x) .- b .* x.^2) .^ 2)
end

# Define a and b values
a_vals = 0:0.01:8
b_vals = 0:0.01:2

# Initialize the matrix
ESS_vals = Matrix{Float64}(undef, length(a_vals), length(b_vals))

# Compute ESS values
for i in 1:length(a_vals)
    for j in 1:length(b_vals)
        ESS_vals[i, j] = fun_ESS(a_vals[i], b_vals[j])
    end
end

# 3D Surface Plot
plot(a_vals, b_vals, ESS_vals', st=:surface, xlabel="a", ylabel="b",
     zlabel="ESS", camera=(60, 10))
```

Out[13]:

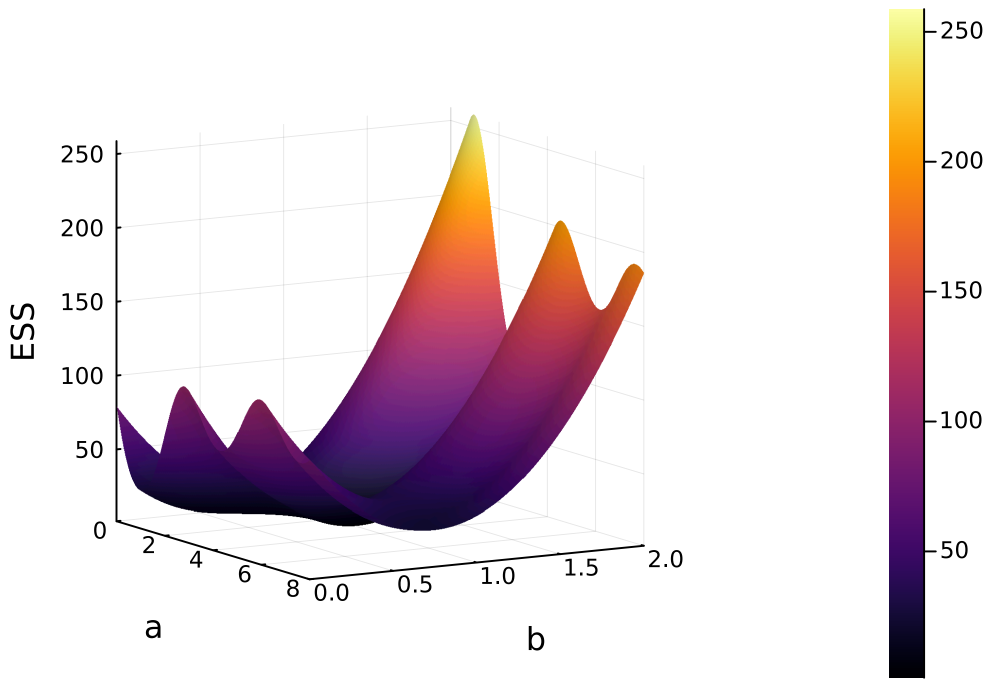

Figure 19: The surface of the negative log-likelihood function for a specific choice of σ^2. Here we plot the error sum of squares, however, one must note that under the normality assumption, the surface is proportional to the negative log-likelihood function.