Exercise 1: Probabilistic models and likelihood functions

Eawag Summer School in Environmental Systems Analysis

Objectives:

- Being able to derive and implement the likelihood function of a model.
- Performing maximum likelihood estimations.
- Understanding the difference between evaluating a likelihood function and sampling from a likelihood function.

Toy Models:

Some exercises will use the toy models “Monod”, “Growth”, or “Survival”, which were introduced in the previous lectures. You can see their definitions for instance in the slides “Mathematical Representation and Construction of Models”.

1. Likelihood for a linear model ★

Analytical expression

Construct the likelihood function for a simple one-dimensional linear model and additive, independent identically distributed (i.i.d.) normal errors. That is, the deterministic part of the model is given by

$$y_{det}(x, \beta, \gamma) = \beta x + \gamma \,,$$

and the probabilistic model is obtained by adding a Gaussian noise term.

Hint: Have a look at the slides “Formulation of model likelihood functions” if you need inspiration.

Likelihood evaluation

For the linear model, implement a function that returns the logarithm of the likelihood, for given model parameters and measurement data. Read the data provided in the file `model_linear.csv` and use them as “observation data” to compute the log-likelihood for two parameter sets: $\{\beta = 2,\ \gamma = 1,\ \sigma = 1\}$ and $\{\beta = 3,\ \gamma = 2,\ \sigma = 1\}$. Which parameters are more likely to have generated this data?

Hints

R **Julia**

Read the data in to a `DataFrame`

```
using DataFrames
import CSV

dat = DataFrame(CSV.File("data/model_linear.csv", delim=' '))
dat.x
dat.y
```

Your loglikelihood function should look like this

```
using Distributions

function loglikelihood_linear(theta, x, y)
    y_det = ... # first evaluate the deterministic model output,
        # using the parameters par = c(beta, gamma, sigma)

        ## Calculate loglikelihood. use `logpdf(Normal(...))`

        end
```

Likelihood optimization

Use an optimizer to find the parameter values that maximise the likelihood (the so called maximum likelihood estimator, MLE). Plot the resulting linear model together with the data. Does the result look reasonable?

Hints

R **Julia**

Use the function `minimizer` from the package `Optim.jl`. As such, you will have to create a negative-loglikelihood function to minimize. This could look like this:

```
using Optim
param = Optim.minimizer(optimize(???, ???));
best_param = ???
```

Linear regression

Use the standard linear regression function to estimate the parameters, and compare them to the ones you found through likelihood maximisation in Exercise 1.3.

Hints

R **Julia**

Use the function `lm` from the package `GLM.jl` (Generalized Linear Model) and the macro `@formula`.

Solutions

Analytical expression

For a one-dimensional linear model with an additive error, a model output Y is given by the deterministic term y_{det} plus the noise term. For the noise term we choose here identical independent normal errors, so that for a single observation i (in a set of n observations) the full model is given by

$$Y_i = y_{i,det} + Z_i, \qquad Z_i \sim \mathcal{N}(0, \sigma), \qquad i \in \{1, \ldots, n\} \ .$$

For the deterministic part, we have an input x and two model parameters $\{\beta, \gamma\}$, from which y_{det} is given by

$$y_{det}(x, \beta, \gamma) = \beta x + \gamma \ .$$

We therefore have three model parameters in total: β and γ from the deterministic term, and σ from the noise term. For convenience, we collect these into the set of parameters $\boldsymbol{\theta} = (\{\beta, \gamma, \sigma\})^{T}$.

So for a single observation, the model is given by

$$Y_i = \beta x_i + \gamma + Z_i \ ,$$

which is equivalent to

$$Y_i \sim \mathcal{N}\left(y_{i,det}(x_i, \beta, \gamma), \sigma\right) \ .$$

That is, the model output Y_i is normally distributed, where the mean is the deterministic model output $y_{i,det}$ and the standard deviation σ is the standard deviation of the error term Z_i. Note that adding a constant μ to a standard normally distributed random variable Z results in a new normally distributed random variable, the mean of which is simply shifted: $Y = \mu + Z, Z \sim \mathcal{N}(0, \sigma) \implies Y \sim \mathcal{N}(\mu, \sigma)$.

So, the likelihood for a single observation y_i, given input x_i and parameters $\boldsymbol{\theta}$, is given by

$$p(y_i|x_i, \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - y_{i,det})^2}{2\sigma^2}\right) .$$

Under the assumption of independence, the likelihood of all observations $\mathbf{y} = \{y_1, y_2, \ldots, y_n\}$, given all inputs $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, is then given by

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = p(y_1|x_1, \boldsymbol{\theta})\, p(y_2|x_2, \boldsymbol{\theta})\, p(y_3|x_3, \boldsymbol{\theta}) \;\ldots\; p(y_n|x_n, \boldsymbol{\theta}) = \prod_{i=1}^{n} p(y_i|x_i, \boldsymbol{\theta}) \;,$$

which is then given in full by

$$\begin{aligned} L(\boldsymbol{\theta}) = p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - y_{i,det})^2}{2\sigma^2}\right) \\ &= \frac{1}{(2\pi)^{n/2}\sigma^n} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \beta x_i - \gamma)^2\right) . \end{aligned}$$

Note: If the observations were not independent, the likelihood would instead be given by

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = p(y_1|\mathbf{x}, \boldsymbol{\theta})\, p(y_2|y_1, \mathbf{x}, \boldsymbol{\theta})\, p(y_3|y_1, y_2, \mathbf{x}, \boldsymbol{\theta}) \;\ldots\; p(y_n|y_1, y_2, \ldots, y_{n-1}, \mathbf{x}, \boldsymbol{\theta}) \;.$$

Only because the observations are independent, this reduces to the simple product given above.

Likelihood evaluation

R **Julia**

The likelihood can be implemented like this

```julia
using Distributions

function loglikelihood_linear(beta, gamma, sigma, x, y)
    y_det = beta .* x .+ gamma
    return sum(logpdf.(Normal.(y_det, sigma), y))
end
```

Then read you data so that you can manipulate it

```julia
using DataFrames
import CSV

linear_data = CSV.read(joinpath("..", "..", "data", "model_linear.csv"),
                       DataFrame)
#show(linear_data, allrows=true) if you want to see all your data

x = linear_data.x
y = linear_data.y
```

Compute

```julia
#Log likelihood with parameters β = 2, γ = 1, σ = 1
loglikelihood_linear(2,1,1, x, y)
```

```
## -27.551462490914655
```

```julia
#Log likelihood with parameters β = 3, γ = 2, σ = 1
loglikelihood_linear(3,2,1, x, y)
```

```
## -166.72710705284968
```

```julia
#parameters  β = 2, γ = 1, σ = 1 are more likely to have produced to data
```

Likelihood optimisation

R Julia

Implementing the negative loglikelihood that we will minimize. Note, we optimize for the `log_sigma` to avoid any negative values for `sigma` .

```julia
function neg_loglikelihood_linear(θ::Vector)
    beta, gamma, log_sigma = θ
    # minus sign to be able to minimize after
    - loglikelihood_linear(beta, gamma, exp(log_sigma), x, y)
end
```

Minimizing

```julia
using Optim
θinit = [2.0, 1, log(0.5)]
```

```
## 3-element Vector{Float64}:
##   2.0
##   1.0
##  -0.6931471805599453
```

```
param = Optim.minimizer(optimize(neg_loglikelihood_linear, θinit));

param[3] = exp(param[3])        # back-transform to sigma
```

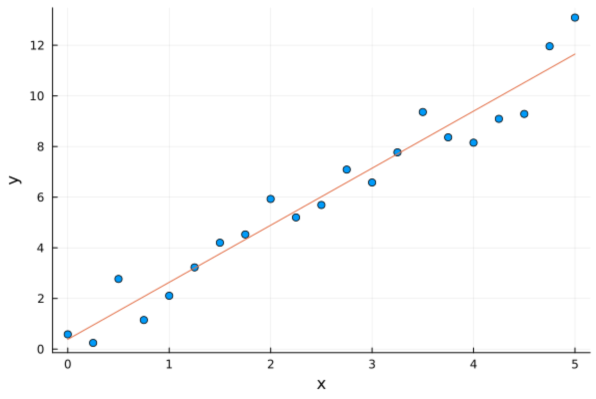
```
## 0.7991010329384892
```

```
param
```

```
## 3-element Vector{Float64}:
##  2.25342685673029
##  0.3843207189590402
##  0.7991010329384892
```

Plotting

```
using Plots

scatter(x, y,
    labels=false,
    xlabel = "x",
    ylabel = "y");
plot!(x -> param[1]*x + param[2],
      labels = false)
```



Linear regression

The linear regression functions works differently (it uses a method called “QR decomposition”), but should give very similar results to our likelihood optimisation.

R **Julia**

```
using GLM
linear_mod = GLM.lm(@formula(y ~ x), linear_data)
```

```
## StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Vector{Float64}}, GL
M.DensePredChol{Float64, CholeskyPivoted{Float64, Matrix{Float64}, Vector{Int6
4}}}}, Matrix{Float64}}
##
## y ~ 1 + x
##
## Coefficients:
## ────────────────────────────────────────────────────────────────────────
##                 Coef.  Std. Error      t  Pr(>|t|)  Lower 95%  Upper 95%
## ────────────────────────────────────────────────────────────────────────
## (Intercept)  0.384349    0.353936   1.09    0.2911  -0.356447    1.12514
## x            2.25342     0.121103  18.61    <1e-12   1.99995     2.50689
## ────────────────────────────────────────────────────────────────────────
```

Note, `(intercept)` corresponds to γ and `x` to β.

2. Likelihood for model "Monod" ★

Analytical expression

Construct the likelihood function for the model "Monod". Use the deterministic part of the model given in the introduction to the exercises and assume i.i.d. normal errors on the deterministic model output. The result will look very similar to the solution of Exercise 1.1!

Likelihood evaluation

Implement a function that returns the logarithm of the likelihood, for given model parameters and measurement data. Read the data provided in the file `model_monod_stoch.dat` and use them as "observation data" to compute the log-likelihood for the parameter sets $\{r_{max} = 5,\ K = 3,\ \sigma = 0.2\}$ and $\{r_{max} = 10,\ K = 4,\ \sigma = 0.2\}$. Which parameters are more likely to have generated this data?

Solutions

Analytical expression

For the model "Monod" with additive normal i.i.d. noise, constructing the likelihood follows exactly the same procedure as in the linear case (Exercise 1.1), only with a simple change in the deterministic part of the model. So again, our complete probabilistic model is given by the deterministic part plus a Gaussian noise term,

$$Y_i = y_{i,det} + Z_i, \qquad Z_i \sim \mathcal{N}(0, \sigma)\ ,$$

but now the deterministic part is given by

$$y_{det}(x, \boldsymbol{\theta}_{monod}) = r(C, r_{max}, K) = \frac{r_{max} C}{K + C} \ .$$

We therefore again have three model parameters in total: $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{monod}, \sigma\} = \{r_{max}, K, \sigma\}$.

The expression for the likelihood is then similar to that of the linear model:

$$L(\boldsymbol{\theta}) = p(\mathbf{y}|\mathbf{C}, r_{max}, K, \sigma) = \frac{1}{(2\pi)^{n/2}\sigma^n} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - r(C_i, r_{max}, K))^2\right) \ .$$

Likelihood evaluation

R Julia

Implementing the loglikelihood

```
using ComponentArrays
using Distributions
include("../../models/models.jl") # load the deterministic model

function loglikelihood_monod(yobs, C, par)
    ## deterministic part:
    ydet = model_monod(C, par)

    ## Calculate loglikelihood:
    return sum(logpdf.(Normal.(ydet, par.sigma), yobs))
end
```

Read the data

```
monod_data = CSV.read(joinpath("..", "..", "data", "model_monod_stoch.csv"),
             DataFrame)

C = monod_data.C
yobs = monod_data.C
```

We can compute

```
## compare the two parameter vectors
par1 = ComponentVector(r_max = 5, K = 3, sigma=0.2)
```

```
## ComponentVector{Float64}(r_max = 5.0, K = 3.0, sigma = 0.2)
```

```
par2 = ComponentVector(r_max = 10, K = 4, sigma=0.2)
```

```
## ComponentVector{Float64}(r_max = 10.0, K = 4.0, sigma = 0.2)
```

```
loglikelihood_monod(yobs, C, par1)
```

```
## -2456.3270173582187
```

```
loglikelihood_monod(yobs, C, par2)
```

```
## -438.3824168930038
```

Plotting

```
# use the better parameters to compute the deterministic predictions
y_det = model_monod(C, par1)

# plots
scatter(C, yobs,
        labels = false,
        xlabel = "C",
        ylabel = "r");
plot!(C, y_det,
      labels = false)
```

3. Forward model simulation ★

Deterministic model simulation

Produce deterministic model outputs. Do this for the two example models:

- Model "Monod" over a concentration (C) range from 0 to 10 with the default parameter values $r_{\max} = 5$ and $K = 3$.
- Model "Growth" over a time interval from 0 to 2 with default parameter values $\mu = 4$, $K = 10$, $b = 1$, $Y = 0.6$, $C_{\mathrm{M,ini}} = 10$, and $C_{\mathrm{S,ini}} = 50$.

Plot and interpret the results.

Hints

R **Julia**

Both deterministic models are already implemented as `model_monod` and `model_growth` in the file `models.jl`.

Probabilistic model simulation

For the model "Monod", write a function that produces model outputs (i.e. samples from the probabilistic model). In other words, we want to simulate new observation data including observation noise.

- For what could that be useful?

Make the assumptions that the noise is i.i.d normal with standard deviation $\sigma = 0.2$. For the deterministic model use $r_{\max} = 5$ and $K = 3$.

Use your function to simulate several thousand probabilistic model realisations (hypothetical data sets), for fixed model parameters, and plot the 10% and 90% quantiles as continuous prediction bands.

Hints

R **Julia**

For the computation of quantiles, you can use the function `quantile` .

Solutions

Deterministic model simulation

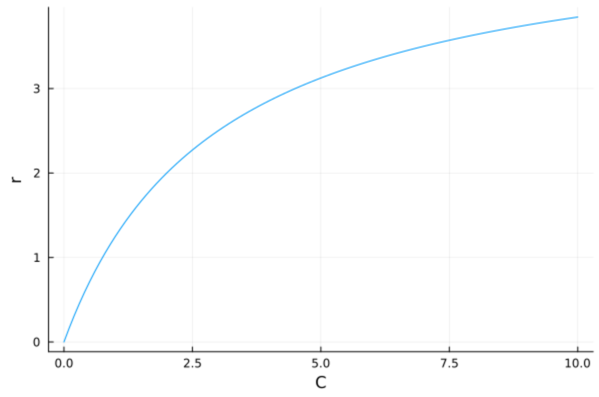
R **Julia**

Monod model

```
using ComponentArrays
include("../../models/models.jl") # load the model

#set parameters
par = ComponentVector(r_max = 5, K = 3)
C_monod = 0:0.1:10

# run deterministic model monod
Y_monod = model_monod(C_monod, par)
```

Plotting

```
plot(C_monod, Y_monod,
    labels = false,
    xlabel = "C",
    ylabel = "r")
```

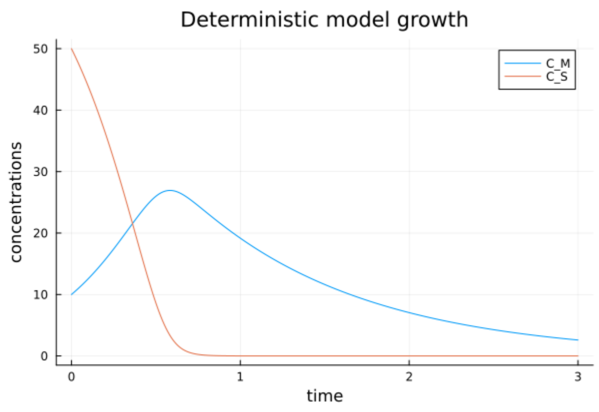


Growth model

```julia
using ComponentArrays
include("../../models/models.jl") # load the model

# set parameters
par = ComponentVector(mu=4, K=10, b=1, Y=0.6,
                      inits = (C_M=10.0, C_S=50.0))
times = 0:0.01:3

# run model
res = model_growth(times, par)
```

Plotting

```julia
plot(res.time, res.C_M,
     label = "C_M",
     xlabel = "time",
     ylabel = "concentrations",
     title = "Deterministic model growth");
plot!(res.time, res.C_S, label="C_S")
```



Stochastic model simulation

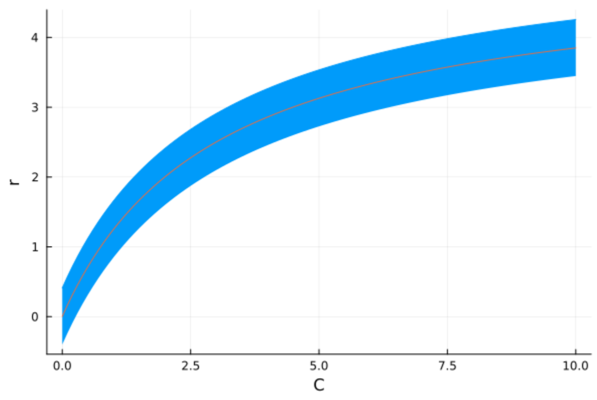
The growth model takes the input variables $\mathbf{C} = (C_1, \dots, C_n)$ and returns the deterministic growth rate $\mathbf{r}_{\text{det}} = \mathbf{r}(\mathbf{C}, r_{\max}, K) = \{r(C_1, r_{\max}, K), \dots, r(C_n, r_{\max}, K)\}$. We simulate the measurement errors by generating n random normal values with mean 0 and standard deviation σ and add them to the deterministic model output.

R Julia

```julia
using ComponentArrays
include("../../models/models.jl") # load the deterministic model

# function to simulate stochastic realisations
function  simulate_monod_stoch(C, par)
    Ydet = model_monod(C, par)
    z = rand(Normal(0, par.sigma)) # adding noise
    Ydet .+ z
end

# parameters
par = ComponentVector(r_max = 5, K = 3, sigma=0.2)
C_monod = 0:0.1:10
n_sample = 1000

monod_stoch = hcat((simulate_monod_stoch(C_monod, par) for _ in 1:n_sample)...)

#compute quantile
low_quantile = [quantile(monod_stoch[i,:], 0.025) for i in 1:length(C_monod)]
med_quantile = [quantile(monod_stoch[i,:], 0.5) for i in 1:length(C_monod)]
upper_quantile = [quantile(monod_stoch[i,:], 0.975) for i in 1:length(C_monod)]
```

Plotting

```julia
plot(C_monod, upper_quantile,
    fillrange = low_quantile,
    labels = false,
    xlabel = "C",
    ylabel = "r");
plot!(C_monod, med_quantile,
    labels = false)
```



4. Likelihood and forward simulation for

the model “Survival”

Analytical expression for the likelihood

Construct the likelihood for the model “Survival”. The mortality rate λ, upon multiplication with an infinitesimal time interval Δt, denotes the probability to die within this time interval, given that one is still alive at the beginning of it. If $S(t)$ denotes the probability of an individual to be still alive at time point t, this reads as

$$\frac{S(t+\Delta t) - S(t)}{S(t)} = -\lambda \Delta t \,.$$

If we let $\Delta t \to 0$ this equation turns into the differential equation

$$\dot{S}(t) = -\lambda S(t)\,.$$

Solve this equation to find the time-dependence of $S(t)$ (Hint: try an exponential ansatz or wolframalpha.com (https://www.wolframalpha.com/)). From this solution, derive the probability for an individual to die within time-interval $[t_{i-1}, t_i]$. Now, consider N independent individuals, each with the same mortality rate λ. Derive the likelihood function, for a vector of death counts $\mathbf{y} = (y_1, \dots, y_n)$, where y_i denotes the number of deaths occurring within time interval $[t_{i-1}, t_i]$, for $0 = t_0 < t_1 < \cdots < t_n$.

Hint: Look up the definition of the multinomial distribution!

Forward simulation

Write a function that simulates output from this probabilistic model, for given parameter values. Use this function to simulate several thousand model realisations, for fixed model parameters (use $N = 30$ individuals with mortality rate $\lambda = 0.2d^{-1}$ and 5 subsequent observation windows of one day each). Use a boxplot to visualize the results.

Likelihood evaluation

Implement a function that returns the logarithm of the likelihood, for given parameter values and measurement data. Check that your log-likelihood is implemented correctly by generating model outputs and computing the likelihood for several parameters, including the one that you generated the data with.

Solutions

Analytical expression for the likelihood

The probability for an individual with mortality rate λ to survive until time t is given by the exponential

$$S(t, \lambda) = e^{-\lambda t}\,.$$

Thus, the probability of the individual to die within time interval $[t_{i-1}, t_i]$ is given by

$$p_{i,\lambda} = S(t_{i-1},\lambda) - S(t_i,\lambda)\,.$$

Assume that we have N individuals, each with the same mortality rate. We want to calculate the probability that y_1 individuals die in time interval $[t_0, t_1]$, y_2 individuals die in time interval $[t_2, t_3]$ etc. Those individuals who survive the whole experiment die in the time interval $[t_n, \infty]$, and this happens with probability $p_{n+1} = S(t_n)$. For the time being, assume that we distinguish the individuals and want to calculate the probability that individual 1 dies in, say, time interval $[t_3, t_4]$, individual 2 in time interval, say, $[t_1, t_2]$ etc. in such a way that the counts $\mathbf{y}$ are assumed. Due to the independence of the individuals, the probability, for such an event, is then simply given by the product

$$p_{1,\lambda}^{y_1} \cdot p_{2,\lambda}^{y_2} \cdot \ldots \cdot p_{n,\lambda}^{y_n}\,.$$

Since we do not distinguish individuals, we have to multiply this product by a so-called multinomial coefficient, which counts the number of ways to put N individuals into $n+1$ buckets, with y_1 ending up in the first bucket etc. Thus, the likelihood function, for model parameter λ, given an output $\mathbf{y}$ of our survival model is described by the so-called multinomial distribution

$$L_{\text{survival}}(\lambda, \mathbf{y}) = \frac{N!}{y_1!\cdots y_{n+1}!} p_{1,\lambda}^{y_1} \cdot p_{2,\lambda}^{y_2} \cdot \ldots \cdot p_{n+1,\lambda}^{y_{n+1}}\,.$$

Forward simulation

R **Julia**

Create a function survival, whose arguments are:

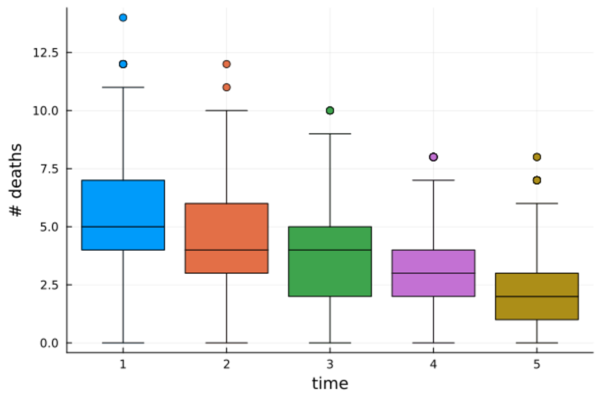
- `N,` the number of individuals,
- `t` , a vector of the form 1:k
- `lambda` , the mortality rate.

Returns a random vector `y` of number of deaths per bin.

```julia
function survival(N::Int64, t, lambda)
    # S is a vector whose first component is 1 and last is 0,
    # because the probability to be alive at time t=0 is 1,
    # and the probability to be alive at time t=infinity is 0
    S = exp.(-lambda .* t)
    S = [1; S; 0]
    p = -diff(S)

    y = rand(Multinomial(N, p))[1:(end-1)]
    return y
end

# setting parameter
N = 30
t = 1:5
lambda = 0.2
n_sample = 1000

pred_survival = Matrix{Int}(undef, n_sample, length(t))
for i in 1:n_sample
    pred_survival[i,:] = survival(N, t, lambda)
end
```

Plotting

```julia
using StatsPlots

boxplot(pred_survival,
        labels = false, xlab="time", ylab="# deaths")
```



Likelihood evaluation

R Julia

```
function loglikelihood_survival(N::Int64, t, y::Vector, lambda)
    S = exp.(-lambda .* t)
    S = [1; S; 0]
    p = -diff(S)
    ## Add number of survivors at the end of the experiment:
    push!(y, N-sum(y))

    logpdf(Multinomial(N, p), y)
end
```

```
# Generating and checking
N = 30
```

```
## 30
```

```
t = 1:5
```

```
## 1:5
```

```
lambda = 0.2
```

```
## 0.2
```

```
y = survival(N, t, lambda)
```

```
## 5-element Vector{Int64}:
##  7
##  4
##  5
##  2
##  2
```

```


loglikelihood_survival(N, t, y, lambda)
```

```
## -8.078613538971037
```

5. Sensitivity analysis (NEEDS TO BE REWRITTEN!)

For the models “Monod” and “Growth” try and compare different sensitivity analysis approaches.

Manual sensitivity analysis

Increase the model parameters by 10% and by 50%, redo the forward simulations, and try to understand the effect of the parameter change on the model results.

Local sensitivity analysis

A simple metric that measures the sensitivity of model parameters θ to model output Y is defined as

$$s_{loc} = \frac{\Delta Y}{\Delta \theta}$$

Often a relative metric is easier to interpret:

$$s_{loc} = \frac{\theta}{Y} \frac{\Delta Y}{\Delta \theta}$$

Set the ranges $\Delta\theta$ to 10% and by 50% of the parameter values and interpret the results.

Variance-based sensitivity

Conduct a variance-based regional (global) sensitivity analyses with lognormal parameter distributions based on default mean values and standard deviations of 10% and 50% of the mean and interpret the results.

Hints

R **Julia**

Use the package `GlobalSensitivity.jl` (https://docs.sciml.ai/GlobalSensitivity/stable/) and the method `eFAST` .

Comparing different sensitivity measures

Compare the results of the different sensitivity approaches. Do the rankings agree? Which one would you trust?

Solutions !!TODO!!

Local sensitivity analysis

Variance-based sensitivity

R **Julia**

```julia
using GlobalSensitivity

res1 = gsa(model.growth,
           eFAST(),
           [(0,1),
            (2, 5)
            ],
           samples=1500)
```

Or

```julia
using GlobalSensitivityAnalysis
using Distributions
using DataStructures


# define the data
data = SobolData(
    params = OrderedDict(:x1 => Uniform(-3.14159265359, 3.14159265359),
        :x2 => Uniform(-3.14159265359, 3.14159265359),
        :x3 => Uniform(-3.14159265359, 3.14159265359)),
    N = 1000
)

# generate samples using Sobol sequence
samples = sample(data)

# run model (example)
Y = model.(samples)

# perform Sobol Analysis
analyze(data, Y)
```