Starkmann, Josef Michael 01.07.2018

**Softwareprojekt**

Institut für Mathematik der Universität Augsburg
Lehrstuhl für rechnerorientierte Statistik und Datenanalyse
Prof. Dr. Gernot Müller

Sommersemester 2018

---

# Maximum-Liklihood-Schätzer für Ornstein-Uhlbeck-Prozesse - Simulationsstudie

---

## 1. Ornstein-Uhlenbeck-Prozesse

### 1.1. Definition

**Definition 1.1.** Seien $a, \alpha \in \mathbb{R}$, $\kappa, \sigma > 0$ und $B(t)$ eine (standardisierte) brownsche Bewegung (Wiener Prozess), dann bezeichnen wir den Prozess $X_t$, welcher die stochastische Differentialgleichung

$$\begin{aligned} dX_t &= \kappa(\alpha - X_t)dt + \sigma dB(t) \\ X_0 &= a \end{aligned} \tag{1.1}$$

erfüllt, als Ornstein-Uhlenbeck-Prozess.

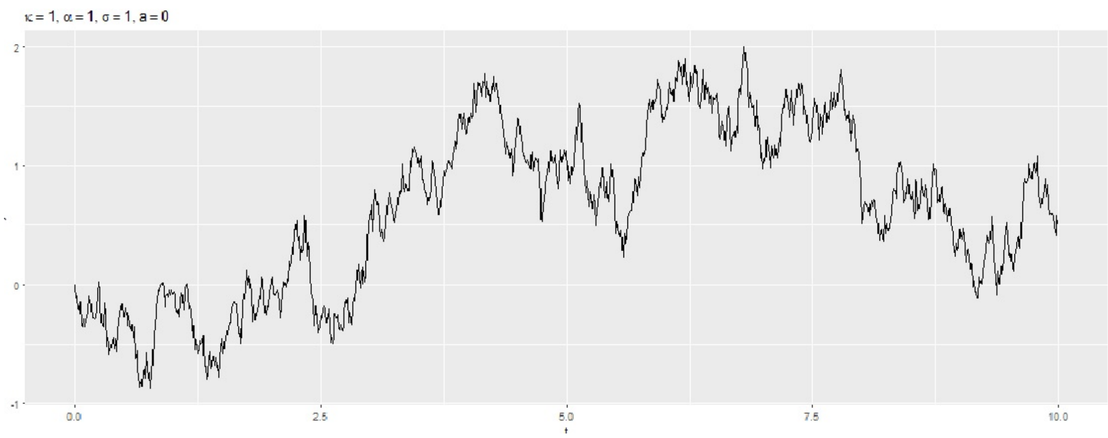


Abbildung 1.1: OU-Prozess mit Parametern $\kappa = \alpha = \sigma = 1, a = 0$

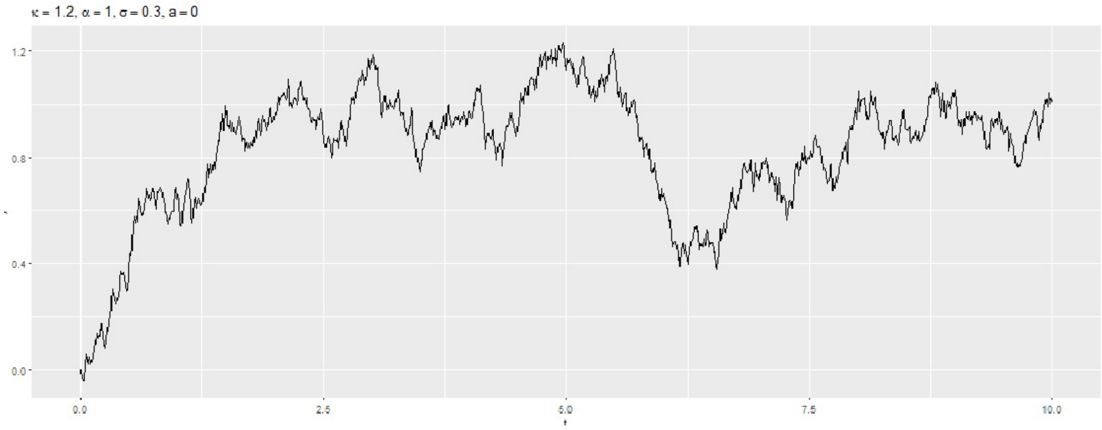


Abbildung 1.2: OU-Prozess mit Parametern $\kappa = 1.2, \alpha = 1, \sigma = 0.3, a = 0$

## 1.2. ML-Schätzer

Für die Berechnung der Maximum-Liklihood-Schätzer eines, wie in (1.1) beschriebenen, OU-Prozesses verwenden wir folgende Resultate aus [5].

**Satz 1.2.** *Sei $(X_i)_{i=0}^n$ die Beobachtung eines OU-Prozesses(vgl. (1.1)), $\delta$ die (konstante) Zeitdifferenz zwischen $X_i$ und $X_{i-1}$, dann berechnet man die ML-Schätzer $\hat{\kappa}, \hat{\alpha}, \hat{\sigma}$ durch*

$$\hat{\kappa} = -\delta^{-1} \log(\hat{\beta}_1) \tag{1.2}$$

$$\hat{\alpha} = \hat{\beta}_2, \tag{1.3}$$

$$\hat{\sigma}^2 = 2\hat{\kappa}\hat{\beta}_3(1 - \hat{\beta}_1)^{-1}, \tag{1.4}$$

$$\tag{1.5}$$

*wobei*

$$\hat{\beta}_1 = \frac{n^{-1}\sum_{i=1}^n X_i X_{i-1} - n^{-2}\sum_{i=1}^n X_i \sum_{i=1}^n X_{i-1}}{n^{-1}\sum_{i=1}^n X_{i-1}^2 - n^{-2}(\sum_{i=1}^n X_{i-1})^2}, \tag{1.6}$$

$$\hat{\beta}_2 = \frac{n^{-1}\sum_{i=1}^n (X_i - \hat{\beta}_1 X_{i-1})}{1 - \hat{\beta}_1}, \tag{1.7}$$

$$\hat{\beta}_3 = n^{-1}\sum_{i=1}^n \Big(X_i - \hat{\beta}_1 X_{i-1} - \hat{\beta}_2(1 - \hat{\beta}_1)\Big)^2. \tag{1.8}$$

# 2. Applikation zur Simulationsstudie

## 2.1. Kurze Beschreibung der wichtigsten Funktionen

Ziel der Simulationsstudie war es, die Güte der Maximum-Liklihood-Schätzer (vgl. Satz 1.2) für Ornstein-Uhlenbeck Prozesse zu testen. Dazu werden in *"generate_ou_processes"* $D$ OU-Prozesse der Länge $n$ mit Zeitdifferenz *delta* mithilfe der Funktion *"sde.sim"* aus dem Paket *sde* (vgl. [6]) erzeugt und als $D \times (n+1)$ Array ausgegeben.

```
generate_ou_processes = function(n,D,x_0,kappa,alpha,sigma,delta){
X = matrix (nrow = D, ncol = n+1)
# PARAMETER NEED TO BE ADAPTED
theta = c(kappa*alpha,kappa,sigma)
# USE REPLICATE FUNCTION INSTEAD OF FOR QUEUE FOR PERFORMANCE REASONS
X = replicate(D,{sde.sim(t0 = 0, T=x_0+n*delta,X0 = x_0,N = n, delta =
    delta,theta = theta, model="OU")})
return(t(X))
}
```

Erzeugen der Daten

Mithilfe der Funktion "compute_ML_estimations", einem gegebenen Datensatz $X$ mit $D$ Zeitreihen der Länge $n$ und Zeitdifferenz *delta*, lassen sich nun, wie in 1.2 beschrieben, $\hat{\alpha}, \hat{\kappa}$ und $\hat{\sigma}$ $D-$mal berechnen.

```
# FUNCTION THAT COMPUTES THE ESTIMATIONS FOR A GIVEN DELTA
compute_ML_estimations = function (X,delta){
# NUMBER of points in one dataset
n = length(X[1,])-1
# number of datasets
D = length(X[,1])
alphas = 0
kappas = 0
sigmas = 0
betas1 = 0
betas2 = 0
betas3 = 0
# PARAMETER SCHaeTZEN vgl "Parameter Estimation and Bias Correction for
    Diffusion Process"
# ITERATE THROUGH ALL DATASETS
for (i in c(1:D)){
beta1 = (sum(X[i,-1]*X[i,-(n+1)])/n - 1/(n**2)*sum(X[i,-1])*sum(X[i,-(n+1)
    ]))/(sum(X[i,-(n+1)]**2)/n-1/(n**2)*sum(X[i,-(n+1)])**2)
beta2 = (sum(X[i,-1]-beta1*X[i,-(n+1)])/n)/(1-beta1)
beta3 = sum((X[i,-1] -beta1*X[i,-(n+1)]-beta2*(1-beta1))**2)/n

kappa_est = - 1/delta *log(beta1)
alpha_est = beta2
sigma_est = 2*kappa_est*beta3/(1-beta1**2)

```

```
betas1[i] = beta1
betas2[i] = beta2
betas3[i] = beta3
alphas[i] = alpha_est
kappas[i] = kappa_est
sigmas[i] = sigma_est
}
# RETURN RESULTS IN LIST
return(list(alphas=alphas,kappas=kappas,sigmas=sqrt(sigmas),betas1=betas1,
    betas2=betas2,betas3=betas3))
}
```

Berechnen der ML-Schätzer

## 2.2. Beschreibung der Anwendung

In der Anwendung besteht zunächst die Möglichkeit die Daten geeignet zu erzeugen. Sobald man diese öffnet, wird zunächst ein Datensatz mit den Defaultparametern (siehe Abb. 2.1) aus der Objektdatei "init.rds" gelesen. Danach besteht die Möglichkeit, sowohl die Anzahl der Zeitreihen $D$ (*"number of time series' D"*), Schritte pro Zeitreihe $n$ (*"steps per time series n"*), den Startwert $a$ ("starting value a"), die Zeitdifferenz $\delta$(*"time lag $\delta$"*), als auch die Parameter $(\kappa, \alpha, \sigma)$ anzupassen und mit dem Button *"Create Data and Compute Estimates"* neue Prozesse und Parameterschätzungen zu berechnen. Zusätzlich kann man aus Reproduzierbarkeitsgründen einen *"seed"* auswählen.

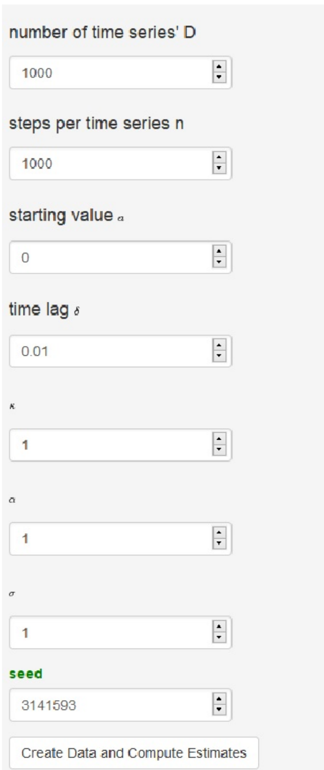


Abbildung 2.1: Eingabefeld App

Das erzeugte Layout besteht aus folgenden Elementen. Zunächst hat man im oberen Teil die Möglichkeit verschiedene generierte Zeitreihen zu betrachten, durch auswählen eines *"index of shown plot"* zwischen 1 und $D$ kann man jeweils eine der generierten Zeitreihen betrachten (vgl. Abb. 2.2).

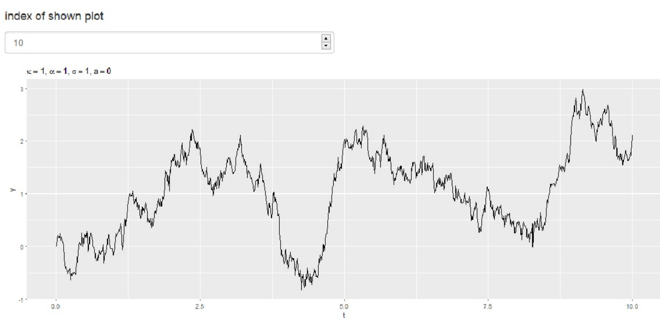


Abbildung 2.2: Der *"index of shown plot"*-te Prozess

Unterhalb dieses Plots werden Grafiken erzeugt, welche die in 1.2 beschriebenen ML-Schätzungen $\hat{\alpha}, \hat{\kappa}, \hat{\sigma}$ pro Zeitreihe bestimmen, zusätzlich ist der tatsächliche Parameter (rot) und der Mittelwert (grün) angegeben (siehe Abb. 2.3).

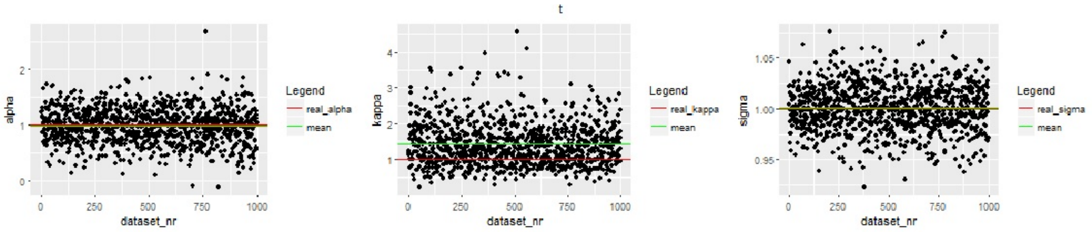


Abbildung 2.3: Schätzungen für $\alpha, \kappa, \sigma$

Im folgenden wird in Abhängigkeit von $\delta$ der veränderte RMSE, mittlere Fehler und die veränderte Varianz aufgezeigt. Das heißt, zum Beispiel für ein ursprüngliches $\delta = 0.01$ ($\delta$ mit dem die Daten erzeugt wurden), besagen diese Plots an der Stelle $\delta = 0.05$ die veränderten Eigenschaften der ML-Schätzer, wenn wir nur noch jeden fünften Punkt der Zeitreihe in der Schätzung berücksichtigen (vgl. Abb. 2.4).

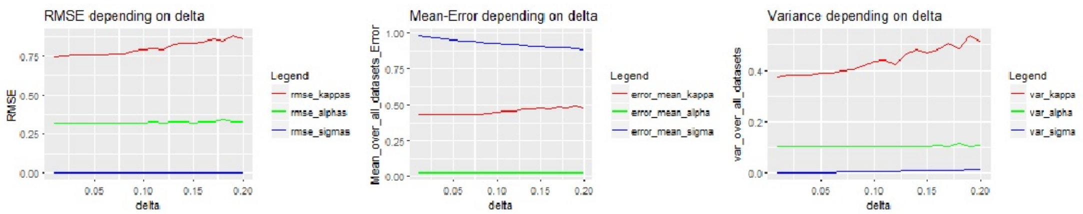


Abbildung 2.4: Veränderung RMSE, mittlerer Fehler, Varianz der Schätzungen mit steigendem $\delta$

Zusätzlich zur Abb. 2.4, kann man sich im folgenden Teil der Anwendungen, die oben beschriebenen Schätzungen für ein größeres $\delta$ noch genauer ansehen (vgl. Abb. 2.5). Im *"General Information"*-Feld werden dafür RMSE, Mean und Varianz der Schätzungen für ein festes *delta* aufgezählt, dabei bezieht sich der obere Teil auf die mit allen Zeitpunkte geschätzten Parameter. Der untere auf ein in *"Increase $\delta$ with factor"* ausgewähltes und mit *"Recompute Estimates"* bestätigtes Vielfaches des anfänglichen *delta*. Zusätzlich werden analog zur Abb. 2.3 die $\hat{\alpha}, \hat{\kappa}$ und $\hat{\sigma}$ für jede der $D$ Zeitreihen dargestellt.

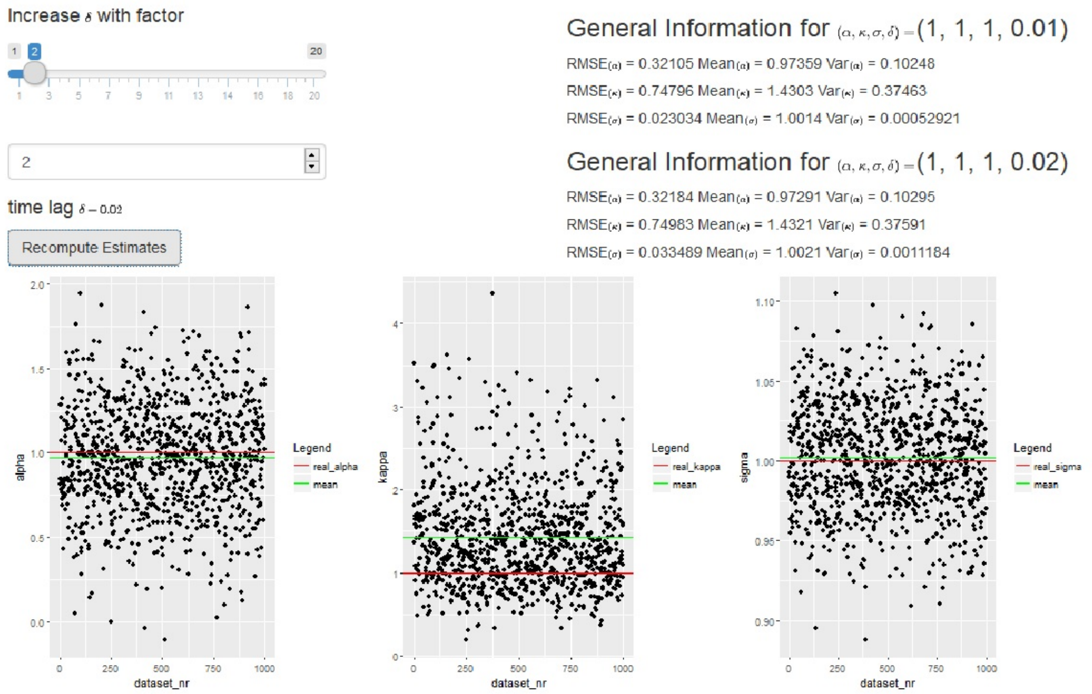


Abbildung 2.5: Neu berechnete Schätzer, mit neuem um Faktor *"Increase δ with factor"* vergrößertem $\delta$

# 3. Mögliche Erweiterungen und weitere Informationen

Mögliche Verbesserungen beziehungsweise Erweiterungen:

- Grafiken speichern
- Erzeugen der Daten versuchen durch Parallelisierung zu beschleunigen (z.B mit *parallel*-Package, vgl. [3])
- Erzeugte Daten importieren und exportieren

<u>**Anmerkung**</u>:
Es gibt die Möglichkeit, die initialisierten Zeitreihen zu ändern in dem man mit folgendem Code eine neue "init.rds" Objektdatei generiert und sie im Ordner der Anwendung speichert. **Vorsicht:** Um danach korrekte Berechnungen der Schätzungen bzw. Anwendung zu gewährleisten, muss man dann auch die Defaultparameter der Inputfelder anpassen.

```
###NEW PARAMETERS####
#ANZAHL DATENSAETZE  D
D = 1000

#ANZAHL Realisierungen pro Datensatz n
n = 10000
######Parameter###############
#X_0 = a
a = 0
#\kappa
kappa = 1
#\alpha
alpha = 1.2
#\sigma
sigma = 0.3
#\delta, dt
delta = 1e-5
##############################
X = generate_ou_processes(n, D, a, kappa,alpha,sigma, delta)
file<-file("path\\to\\app\\OUProcess_Estimation\\init.rds")
## save a single object to file
saveRDS(X, fil)
#close filestream
close(fil)
```

neuer Datensatz in init.rds speichern

## 3.1. Weitere Informationen

- Möglichkeit die App online auszuführen:
  https://ou-process.shinyapps.io/OUProcess_Estimation/
- Git-Repository der Anwendung:
  https://github.com/JosefStarkm/Softwareprojekt_OU_Processes
- Weitere Informationen, Dokumentationen, Beispiele und Tutorials zu Shiny (R Package mit dem die App erstellt wurde) in [4].
- Package mit dem die Plots erstellt wurden: ggplot2 (vgl. [1])

# A. Code

## A.1. UI.R

```

# SOFTWAREPROJEKT OU-Prozesse
# Josef Starkmann
# dX_t = \kappa(\alpha-X_t)dt + \sigma dB(t)
# X_0 = a
#
# This is the user-interface definition of a Shiny web application. You can
# run the application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

# IMPORT SHINY LIBRARY
library(shiny)
X=# Define UI for application that draws a histogram
shinyUI(fluidPage(

  # Application title
  # TITLE
  withMathJax(titlePanel("Computation of $$dX_t = \\kappa(\\alpha-X_t)dt +
      \\sigma dB(t)$$ $$ X_0 = a$$")),
  # SHORT HELPTEXT WITH GENERAL INFORMATION
  helpText("Softwareprojekt OU-Process - Josef Starkmann"),

  # CREATE A SIDEBARLAYOUT WITH SIDEBAR PANEL
  sidebarLayout(
    sidebarPanel(
      ###### Parameter ###############
      # ANZAHL Realisierungen pro Datensatz n

      # ANZAHL DATENSAETZE  D

      # X_0 = a

      # \kappa

      # \alpha

      # \sigma

      # \delta, dt
      ################################
       # CREATE ALL IMPORTANT INPUT FIELDS
```

```
        numericInput("D",h4('number of time series\' D'),value = 1000, step
            =1,width = '50%'),#'
        numericInput("n",h4('steps per time series n'),value = 1000, step
            =1,width = '50%'),
        numericInput("a",h4(withMathJax("starting value \\( a \\)")),value =
             0, step =0.01,width = '50%'),
        numericInput("delta",h4(withMathJax("time lag \\( \\delta \\)")),
            value = 1e-2, step =1e-5,width = '50%'),
        numericInput("kappa",h4(withMathJax("\\( \\kappa \\)")),value = 1,
            step =0.01,width = '50%'),
        numericInput("alpha",h4(withMathJax("\\( \\alpha \\)")),value = 1,
            step =0.01,width = '50%'),
        numericInput("sigma",h4(withMathJax("\\( \\sigma \\)")),value = 1,
            step =0.01,width = '50%'),
        numericInput("seed",div('seed',style = "color:green"),value =
            3141593,step = 1, width = '50%'),
        actionButton("create_data_button","Create Data and Compute Estimates
            ")

    ),

    # CREATE MAINPANEL WITH PLOTS AND INFORMATION
    mainPanel(
      #numericInput with Number of the Plot to show(can be one of the D
          Datasets)
      numericInput("plt",h4('index of shown plot'),value = 10,min = 1,max
          = 20, step =1,width = '50%'),

      # Plot of the choosen process, can be changed dynamically
      plotOutput("process_plot"),

      # Plot of the estimatetes of alpha, kappa, sigma +mean_error_
          depending on delta and variance depending on delta
      plotOutput("distPlot"),

      # FLUID ROW WITH TWO COLUMNS OF WIDTH 6 EACH ONE ROW HAS WIDTH 12
      fluidRow(
        # FIRST LEFT ROW WITH THREE ELEMENTS A SLDER FOR DELTA, a
            correlating numeric input with the same value and a button
        column(6,
               sliderInput("delta_new_slide",h4(withMathJax("Increase \\(
                   \\delta \\) with factor")),min = 1, max = 100, value
                   =1),
               numericInput("delta_new_in", label = "",value = 1),
               uiOutput('text_delta'),
               actionButton("recompute","Recompute Estimates")
               ),
        # SECOND OUTPUT IN THE COLUMN WITH THE GENERAL INFOS
        column(6,uiOutput('infos'),uiOutput('infos2'))

```

```

        ),
        # FURTHER PLOTS THAT OCCUR AFTER WE CLICK RECOMPUTE ESTIMATES
        plotOutput("further_plots")
      )
   )
))
```

OUProcess_Estimation/UI.R

## A.2. Server.R

```
# SOFTWAREPROJEKT OU-Prozesse
# Josef Starkmann
# dX_t = \kappa(\alpha-X_t)dt + \sigma dB(t)
# X_0 = a
#
# This is the server logic of a Shiny web application. You can run the
# application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#






# IMPORT NECESSARY LIBRARIES
library(shiny)
library(ggplot2)
library(sde)
library(grid)

#VARIABLE TO CHECK IF INITIALIZATION OR BUTTON CLICKED
is.init <<- TRUE

print_output = function(b_in,input_in,delta_first_in){
  #check if first initial call or second "recompute" call
  if(substitute(b_in) == 'b()') del = delta_first_in else del = delta_first_in*input_in$delta_new_in #else substitute(b_in) =b_new()
  str_headline = paste("General Information for ","\\( (\\alpha,\\kappa,\\sigma,\\delta) =\\)(",toString(input_in$alpha),", ",toString(input_in$kappa),", ",toString(input_in$sigma),", ",toString(del),") ",sep ="")
  str_alpha =paste("RMSE\\( (\\alpha )\\) = ",format(sqrt(mean((b_in$alphas-input_in$alpha)**2,na.rm=TRUE)),digits = 5),"    Mean\\( (\\alpha )
```

```
\\) = ",format(mean(b_in$alphas,na.rm=TRUE),digits =5),"      Var\\( (\\alpha )\\) = ",format(var(b_in$alphas,na.rm=TRUE),digits =5),sep="")
  str_kappa =paste("RMSE\\( (\\kappa )\\) = ",format(sqrt(mean((b_in$kappas-input_in$kappa)**2,na.rm=TRUE)),digits =5),"      Mean\\( (\\kappa )\\) = ",format(mean(b_in$kappas,na.rm=TRUE),digits =5),"      Var\\( (\\kappa )\\) = ",format(var(b_in$kappas,na.rm=TRUE),digits =5),sep="")
  str_sigma =paste("RMSE\\( (\\sigma )\\) = ",format(sqrt(mean((b_in$sigmas-input_in$sigma)**2,na.rm=TRUE)),digits =5),"      Mean\\( (\\sigma )\\) = ",format(mean(b_in$sigmas,na.rm=TRUE),digits =5),"      Var\\( (\\sigma )\\) = ",format(var(b_in$sigmas,na.rm=TRUE),digits =5),sep="")
  # count nas
  number_nas=sum(is.na(b_in$kappas))
  if(number_nas == 0){
  list(
    h3(withMathJax(str_headline)),
    h5(withMathJax(str_alpha)),
    h5(withMathJax(str_kappa)),
    h5(withMathJax(str_sigma)))
}
else{# if nas occur
  str_star= paste("*",toString(number_nas)," NAs occured! The marked means and variances are computed without these NAs!")
  list(
  h3(withMathJax(str_headline)),
  h5(withMathJax(str_alpha)),
  div(h5(withMathJax(str_kappa)),style = "color:red"),
  div(h5(withMathJax(str_sigma)),style = "color:red"),
  div(str_star,style = "color:red"))
  }
}



# FUNCTION THAT CREATES THE OU PROCESSES
# dX_t = \kappa(\alpha-X_t)dt + \sigma dB(t)
# X_0 = a

# ERZEUGEN DER DATENSAETZE
# i-ter Datensatz = X[i,]
# generate data

###### Parameter ###############
# ANZAHL Realisierungen pro Datensatz n

# ANZAHL DATENSAETZE D

# X_0 = a
```

```
# \kappa

# \alpha

# \sigma

# \delta , dt
##################################

generate_ou_processes = function(n,D,x_0,kappa,alpha,sigma,delta){
  X_ = matrix (nrow = D, ncol = n+1)
  # PARAMETER NEED TO BE ADAPTED
  theta = c(kappa*alpha,kappa,sigma)
  # USE REPLICATE FUNCTION INSTEAD OF FOR QUEUE FOR PERFORMANCE REASONS
  X_ = replicate(D,{sde.sim(t0 = 0, T=x_0+n*delta,X0 = x_0,N = n, delta =
      delta,theta = theta, model="OU")})
  return(t(X_))
}

# FUNCTION THAT COMPUTES THE ESTIMATIONS FOR A GIVEN DELTA
compute_ML_estimations = function(X,delta){
  # NUMBER of points in one dataset
  n = length(X[1,])-1
  # number of datasets
  D = length(X[,1])
  alphas = 0
  kappas = 0
  sigmas = 0
  betas1 = 0
  betas2 = 0
  betas3 = 0
  # PARAMETER SCHaeTZEN vgl "Parameter Estimation and Bias Correction for
      Diffusion Process"
  # ITERATE THROUGH ALL DATASETS
  for (i in c(1:D)){
    beta1 = (sum(X[i,-1]*X[i,-(n+1)])/n - 1/(n**2)*sum(X[i,-1])*sum(X[i,-(n
        +1)]))/(sum(X[i,-(n+1)]**2)/n-1/(n**2)*sum(X[i,-(n+1)])**2)
    beta2 = (sum(X[i,-1]-beta1*X[i,-(n+1)])/n)/(1-beta1)
    beta3 = sum((X[i,-1] -beta1*X[i,-(n+1)]-beta2*(1-beta1))**2)/n

    kappa_est = - 1/delta *log(beta1)
    alpha_est =  beta2
    sigma_est = 2*kappa_est*beta3/(1-beta1**2)

    betas1[i] = beta1
    betas2[i] = beta2
    betas3[i] = beta3
    alphas[i] = alpha_est
    kappas[i] = kappa_est
```

```
    sigmas[i] = sigma_est
  }
  # RETURN RESULTS IN LIST
  return(list(alphas=alphas,kappas=kappas,sigmas=sqrt(sigmas),betas1=betas1
      ,betas2=betas2,betas3=betas3))
}

# RETURNS VARIANCE, MEAN, RMSE PLOT DEPENDING ON TIME DIFFERENCE
# input: GENERATED OU PROCESSES and first delta
# output: vector with all deltas (whole number products of delta_first) and
     means and variances of the alphas,kappas, sigmas
compute_mean_variance_time_plot = function(X,delta_first,input){
  n = length(X[1,])
  rmse_kappas = 0
  mean_kappas = 0
  var_kappas  = 0
  rmse_alphas = 0
  mean_alphas = 0
  var_alphas  = 0
  rmse_sigmas = 0
  mean_sigmas = 0
  var_sigmas = 0
  deltas = 0
  #
  # # iterate through dataset take every i-th point (at least 50)
  for (i in c(1:(n%/%50))){# atleast 50 points
    # new delta
    deltas[i] = delta_first*i
    # new estimator with every i-th Point
    est = compute_ML_estimations(X[,seq(1,n,by = i)],delta = deltas[i])
    # compute all means and variances of the estimated parameters possible
        nas removed

    mean_kappas[i] = mean(est$kappas,na.rm=TRUE)
    #COMPUTE RMSE
    rmse_kappas[i] = sqrt(mean((est$kappas-input$kappa)**2,na.rm=TRUE))
    var_kappas[i]  = var(est$kappas,na.rm=TRUE)

    mean_alphas[i] = mean(est$alphas,na.rm=TRUE)
    #COMPUTE RMSE
    rmse_alphas[i] = sqrt(mean((est$alphas-input$alpha)**2,na.rm=TRUE))
    var_alphas[i]  = var(est$alphas,na.rm=TRUE)

    mean_sigmas[i] = mean(est$sigmas,na.rm=TRUE)
    #COMPUTE RMSE
    mean_sigmas[i] = sqrt(mean((est$sigmas-input$sigma)**2,na.rm=TRUE))
    var_sigmas[i] = var(est$sigmas,na.rm=TRUE)
  }
  # RETURN RESULTS IN LIST
```

```
  return(list(deltas = deltas,rmse_kappas = rmse_kappas,rmse_alphas = rmse_alphas, rmse_sigmas = rmse_sigmas, mean_kappas = mean_kappas, mean_alphas = mean_alphas, mean_sigmas = mean_sigmas,var_kappas=var_kappas,var_alphas = var_alphas,var_sigmas=var_sigmas))
}


# Define server logic
shinyServer(function(input, output,session){

  #CREATE REACTIVE VARIABLE GETS VALUE IF AND ONLY IF input$create_data_button is clicked
  delta_first <- eventReactive(input$create_data_button,{
    isolate({
      print("IN DELTA FIRST")
      set.seed(input$seed)
      input$delta
    })
  },ignoreInit = FALSE,ignoreNULL = FALSE)

  #CREATE REACTIVE VARIABLE GETS VALUE IF AND ONLY IF input$create_data_button is clicked
  #loads file "init.rds" on initialization (FASTER!) and computes file when button is clicked
  X <- eventReactive(input$create_data_button,{
     isolate({
     #LOADS DATASET AT INITALIZATION
     if(is.init){
       is.init <<- FALSE
       fil = file("init.rds")
       temp = readRDS(fil)
       close(fil)
       temp
     }
    else{
    #CREATES DATASET ON input$create_data_button_clicked
     generate_ou_processes(input$n,input$D,input$a,input$kappa,input$alpha,input$sigma,input$delta)
    }
     })},ignoreInit = FALSE,ignoreNULL = FALSE)

  #CREATE REACTIVE VARIABLE GETS VALUE IF AND ONLY IF X changes
  #b <- eventReactive(input$create_data_button,{isolate({
  b <- eventReactive(X,{isolate({
    compute_ML_estimations(X()[,seq(0,input$n,1)],delta = input$delta)
  })},ignoreInit = FALSE,ignoreNULL = FALSE)


  #CREATE REACTIVE VARIABLE GETS VALUE IF AND ONLY IF X has changed
  b_new <- eventReactive(list(input$recompute, X) ,{isolate({
```

```
    print("IN COMPUTE ML2")
    compute_ML_estimations(X()[,seq(0,input$n,input$delta_new_in)],delta =
        input$delta_new_in*delta_first())
    })},ignoreInit = FALSE,ignoreNULL = FALSE)


#SLIDER AND NUMERIC INPUT WITH SAME VALUE
 observe({
    updateSliderInput(session, "delta_new_slide", value = input$delta_new_
        in)
  },priority = 10)

  observe({
    updateNumericInput(session, "delta_new_in", value = input$delta_new_
        slide)
  },priority = 10)


    # FIRST PLOT OF ONE TIME SERIES, POSSIBILITY TO SEE FURTHER TIME SERIES
        OF THE DATASET WITH THE input$plt numericInput
    output$process_plot <- renderPlot({
      # dependency necessary in order to replot if plot is changed or new
          data created
      input$plt
      input$create_data_button
      #print(is.null(X()))
      #print(X())
      isolate({
        t =seq(0,input$n*delta_first(),input$delta)
        y = X()[input$plt,]

        # Create plot with title
        title = bquote(list(kappa==.(input$kappa),alpha==.(input$alpha),
            sigma==.(input$sigma),a==.(input$a)))
        p_1 = qplot(x=t,y=y,geom = 'line')+ggtitle(title)

        # Create GRIDLayout and add plot
        grid.newpage()
        pushViewport(viewport(layout = grid.layout(1,1)))
        vplayout = function(x,y) viewport(layout.pos.row =x,layout.pos.col=
            y)
        print(p_1,vp = vplayout(1,1))
      })

    })
    output$distPlot <- renderPlot({

     # dependecy to create data button and plot
     input$create_data_button
     # isolate the rest
```

```
    isolate({
    # UPDATE MAX VALUE OF SLIDER
    min_points = 50 #atleast 50 points
    updateNumericInput(session,"delta_new_in", max = input$n%/%min_points)
    updateSliderInput(session,"delta_new_slide", max = input$n%/%min_points)

    # CREATE FIRST PLOT WITH ALPHAS AND MEAN AND REAL ALPHA (THE ONE THE PROCESSES WERE CREATED)
    p_2 = qplot(y=b()$alphas, x = c(1:input$D), geom = 'point',xlab="dataset_nr",ylab='alpha')
    p_2 = p_2 + geom_hline(aes(yintercept = input$alpha,color = 'real_alpha'))
    p_2 = p_2 +geom_hline(aes(yintercept = mean(b()$alphas),color = 'mean'))
    # ADD LEGEND AND COLORS
    p_2 =p_2 + scale_color_manual("Legend",values = c('real_alpha'='red','mean'='green'),breaks = c('real_alpha','mean'))

    # CREATE SECOND PLOT WITH KAPPAS AND MEAN AND REAL KAPPA(THE ONE THE PROCESSES WERE CREATED)
    p_3 =qplot(y=b()$kappas, x = c(1:input$D),geom = 'point',xlab="dataset_nr",ylab='kappa')
    p_3 =p_3 + geom_hline(aes(yintercept = input$kappa,color="real_kappa"))
    p_3 =p_3 + geom_hline(aes(yintercept = mean(b()$kappas),color = 'mean'))
    # ADD LEGEND AND COLORS
    p_3 =p_3 + scale_color_manual("Legend",values = c('real_kappa'='red','mean'='green'),breaks = c('real_kappa','mean'))

    # CREATE THIRD PLOT WITH SIGMAS AND MEAN AND REAL SIGMA(THE ONE THE PROCESSES WERE CREATED)
    p_4 = qplot(y=b()$sigmas, x = c(1:input$D),geom = 'point',xlab="dataset_nr",ylab='sigma' )
    p_4 = p_4 + geom_hline(aes(yintercept = input$sigma,color="real_sigma"))
    p_4 = p_4 + geom_hline(aes(yintercept = mean(b()$sigmas),color = 'mean'))
    # ADD LEGEND
    p_4 =p_4 + scale_color_manual("Legend",values = c('real_sigma'='red','mean'='green'),breaks = c('real_sigma','mean'))

    # COMPUTE ESTIMATIONS BY TAKING ONLY EVERY 2nd, 3thd,...n%/%50th POINT
    results = compute_mean_variance_time_plot(X(),delta_first(),input)

    #CREATE PLOTS WITH RMSE DEPENDING ON DELTA (delta=delta_first,....delta=n%/%50*delta_first)
    dfm = data.frame(delta=results$deltas,rmse_kappas=results$rmse_kappas,rmse_alphas = results$rmse_alphas,rmse_sigmas = results$rmse_
```

```
sigmas)
    p_5 = ggplot(data=dfm)+geom_line(aes(x=delta,y=rmse_kappas,color ='kappa'))+ geom_line(aes(x=delta,y= rmse_alphas,color='alpha')) + geom_line(aes(x=delta,y=rmse_sigmas,color='sigma'))
    # ADD LEGEND/COLORS
    p_5 = p_5 + scale_color_manual("Legend",values = c('kappa'='red','alpha'='green','sigma'='blue'), breaks=c('kappa','alpha','sigma'), labels = c('rmse_kappas','rmse_alphas','rmse_sigmas'))
    # ADD LABELS
    p_5 = p_5 +ylab("RMSE")+ggtitle('RMSE depending on delta')


    # CREATE PLOT WITH MEANS AND VARIANCES DEPENDING ON DELTA (delta=delta_first,.... delta=n%/%50*delta_first)
    dfm = data.frame(delta=results$deltas,error_kappa=abs(results$mean_kappas-input$kappa),error_alpha = abs(results$mean_alphas-input$alpha),error_sigma = abs(results$mean_sigmas-input$sigma))
    p_6 = ggplot(data=dfm)+geom_line(aes(x=delta,y=error_kappa,color ='kappa'))+ geom_line(aes(x=delta,y= error_alpha,color='alpha')) + geom_line(aes(x=delta,y=error_sigma,color='sigma'))
    # ADD LEGEND/COLORS
    p_6 = p_6 + scale_color_manual("Legend",values = c('kappa'='red','alpha'='green','sigma'='blue'), breaks=c('kappa','alpha','sigma'), labels = c('error_mean_kappa','error_mean_alpha','error_mean_sigma'))
    # ADD LABELS
    p_6 = p_6 +ylab("Mean_over_all_datasets_Error")+ggtitle('Mean-Error depending on delta')

    # VARIANCE PLOT
    dfm = data.frame(delta=results$deltas,var_kappas=results$var_kappas, var_alphas = results$var_alphas,var_sigmas = results$var_sigmas)
    p_7 = ggplot(data=dfm)+geom_line(aes(x=delta,y=var_kappas,color ='kappa'))+ geom_line(aes(x=delta,y= var_alphas,color='alpha')) + geom_line(aes(x=delta,y=var_sigmas,color='sigma'))
    # ADD LEGEND/COLORS
    p_7 =p_7 +scale_color_manual("Legend",values = c('kappa'='red','alpha'='green','sigma'='blue'),breaks = c('kappa','alpha','sigma'), labels=c('var_kappa','var_alpha','var_sigma'))
    # ADD LABELS
    p_7 =p_7 + ylab("var_over_all_datasets")+ggtitle('Variance depending on delta')#, y="Var_over_all_datasets",title ="Variance by delta")

    # ADD GRID LAYOUT AND PLOTS AT RIGHT POSITION
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(2,3)))
    vplayout = function(x,y) viewport(layout.pos.row =x,layout.pos.col=y)
    print(p_2,vp = vplayout(1,1))
    print(p_3,vp = vplayout(1,2))
    print(p_4,vp = vplayout(1,3))
```

```
      print(p_5,vp = vplayout(2,1))
      print(p_6,vp = vplayout(2,2))
      print(p_7,vp = vplayout(2,3))

      }) #isolation end
})



  # DEFINE PLOTS FOR A BIGGER TIME DIFFERENCE DELTA OF THE GIVEN DATA SETS
  # e.g. delta_first = 0.01 input$delta_new_in = 2 -> delta_new = 0.2
  output$further_plots <- renderPlot({

    # PLOTS ARE CHANGED IF AND ONLY IF ONE OF THOSE BUTTONS PUSHED
    input$create_data_button
    input$recompute
    isolate({
    # CREATE FIRST PLOT WITH ALPHAS AND MEAN AND REAL ALPHA OF THE NEW ESTIMATION
    p_2 = qplot(y=b_new()$alphas, x = c(1:length(b_new()$alphas)), geom = 'point',xlab="dataset_nr",ylab='alpha')
    p_2 = p_2 + geom_hline(aes(yintercept = input$alpha,color = 'real_alpha'))
    p_2 = p_2 + geom_hline(aes(yintercept = mean(b_new()$alphas), color = 'mean'))
    p_2 = p_2 + scale_color_manual("Legend",values = c('real_alpha'='red','mean'='green'),breaks = c('real_alpha','mean'))

    # CREATE FIRST PLOT WITH KAPPAS AND MEAN AND REAL KAPPAS OF THE NEW ESTIMATION
    p_3 = qplot(y=b_new()$kappas, x = c(1:length(b_new()$alphas)),geom = 'point',xlab="dataset_nr",ylab='kappa')
    p_3 = p_3 + geom_hline(aes(yintercept = input$kappa,color = 'real_kappa'))
    p_3 = p_3 + geom_hline(aes(yintercept = mean(b_new()$kappas,na.rm=TRUE),color = 'mean'))
    # ADD LEGEND/COLORS
    p_3 = p_3 + scale_color_manual("Legend",values = c('real_kappa'='red','mean'='green'),breaks = c('real_kappa','mean'))

    # CREATE FIRST PLOT WITH SIGMAS AND MEAN AND REAL SIGMAS OF THE NEW ESTIMATION
    p_4 = qplot(y=b_new()$sigmas, x = c(1:length(b_new()$alphas)),geom = 'point' ,xlab="dataset_nr",ylab='sigma')
    p_4 = p_4 + geom_hline(aes(yintercept = input$sigma,color = 'real_sigma'))
    p_4 = p_4 + geom_hline(aes(yintercept = mean(b_new()$sigmas,na.rm=TRUE),color = 'mean'))
    # ADD LEGEND/COLORS
```

```
    p_4 = p_4 + scale_color_manual("Legend",values = c('real_sigma'='red','mean'='green'),breaks = c('real_sigma','mean'))

    # ADD GRID LAYOUT AND PLOTS AT RIGHT POSITION
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(1,3)))
    vplayout = function(x,y) viewport(layout.pos.row =x,layout.pos.col=y)
    print(p_2,vp = vplayout(1,1))
    print(p_3,vp = vplayout(1,2))
    print(p_4,vp = vplayout(1,3))
    })
  })

  # PRINT UPDATED DELTA VALUE
  output$text_delta <- renderUI({
    str = paste("time lag \\( \\delta =",toString(input$delta_new_in*delta_first()), " \\) ",sep="")
    h4(withMathJax(str))
  })

  # PRINT NEW GENERAL INFORMATION (first part)FIELD WHEN create_data_button is clicked
  output$infos <- renderUI({
      input$create_data_button
      isolate({
        print_output(b(),input,delta_first())
      })
    }
  )
  # PRINT NEW General INFORMATION FIELD (second part/scaled delta) WHEN Recompute is clicked
  output$infos2 <- renderUI({
      input$recompute
      input$create_data_button
      isolate({
        print_output(b_new(),input,delta_first())
      })
    }
  )
})
```

OUProcess_Estimation/Server.R

## References


[1] *ggplot2*. https://ggplot2.tidyverse.org/

[2] *Installation Shiny*. https://www.r-project.org/nosvn/pandoc/shiny.html

[3] *Package 'parallel'*. https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf

[4] *Shiny*. https://shiny.rstudio.com

[5] Chen, S. X. ; Tang, C. Y.: Parameter estiation and bias correction for diffusion processes. *Journal of Econometrics* (2006), S. 65–82

[6] Iacus, S. M.: *Package 'sde' - Simlation and Interference for Stochastic Differential Equations*