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.

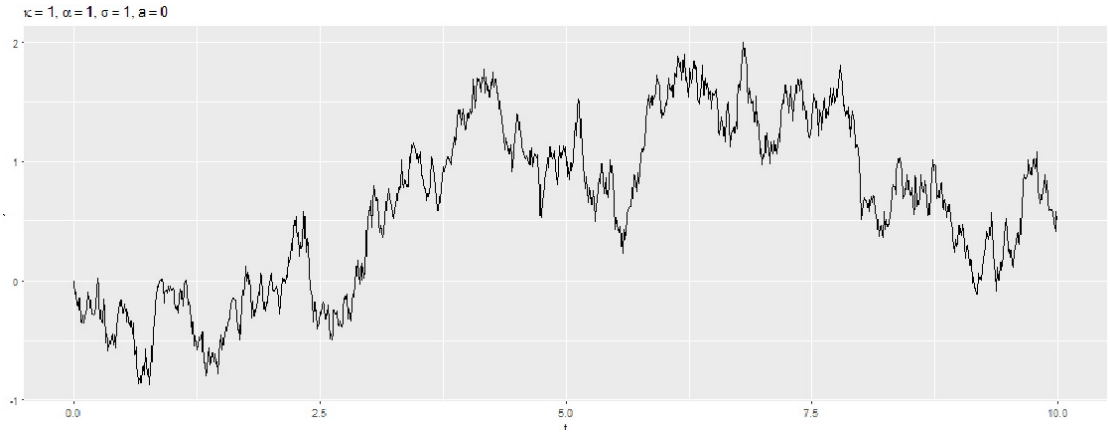

Abbildung 1.1: OU-Prozess mit Parametern $\kappa = \alpha = \sigma = 1, a = 0$

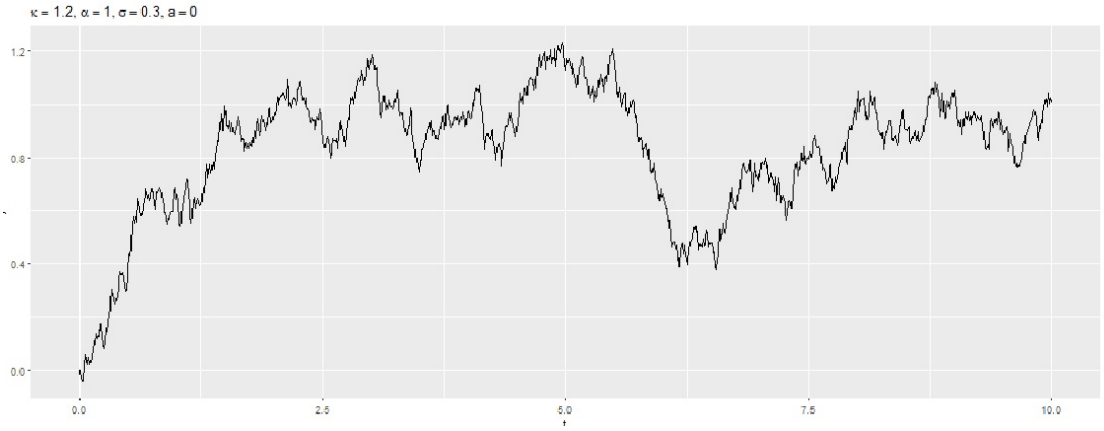

Abbildung 1.2: OU-Prozess mit Parametern $\kappa = 1.2, \alpha = 1, \sigma = 0.3, a = 0$

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)), δ 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)

```

```
24  betas1[i] = beta1
25  betas2[i] = beta2
26  betas3[i] = beta3
27  alphas[i] = alpha_est
28  kappas[i] = kappa_est
29  sigmas[i] = sigma_est
30  }
31  # RETURN RESULTS IN LIST
32  return(list(alphas=alphas,kappas=kappas,sigmas=sqrt(sigmas),betas1=betas1,
        betas2=betas2,betas3=betas3))
33  }
```

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 δ(*"time lag* δ"), als auch die Parameter (κ, α, σ) anzupassen und mit dem Button *"Create Data and Compute Estimates"* neue Prozesse und Parameterschätzungen zu berechnen. Zusätzlich kann man aus Reproduzierbarkeitsgründen einen *"seed"* auswählen.

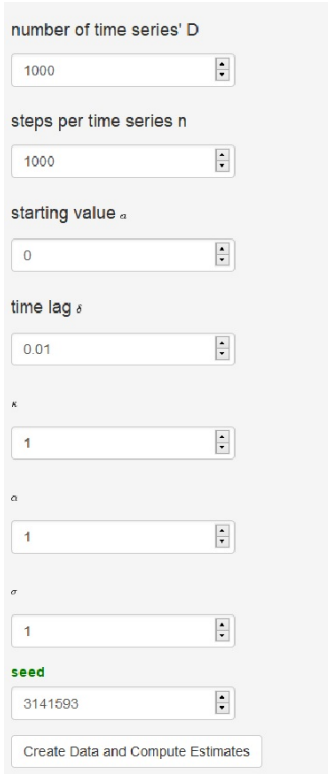

Abbildung 2.1: Eingabefeld App

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).

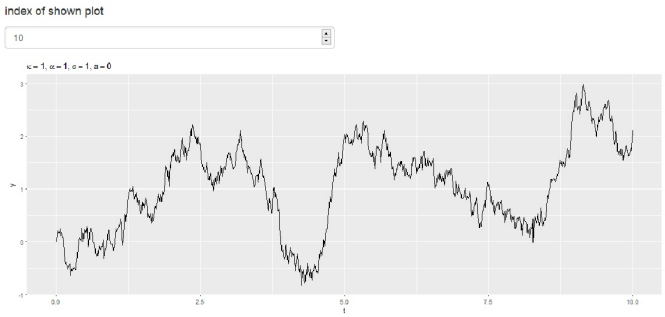

Abbildung 2.2: Der *"index of shown plot"*-te Prozess

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).

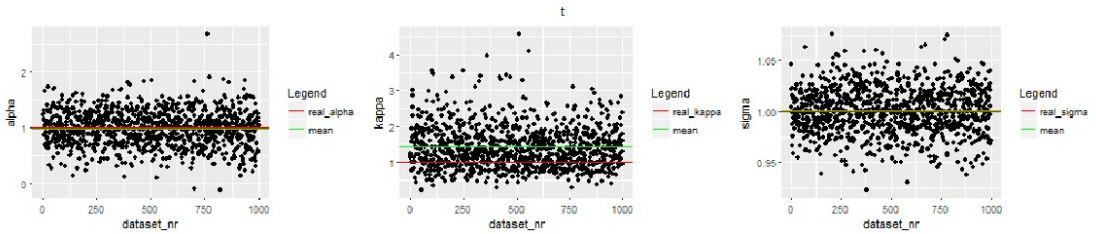

Abbildung 2.3: Schätzungen für α, κ, σ

Im folgenden wird in Abhängigkeit von δ der veränderte RMSE, mittlere Fehler und die veränderte Varianz aufgezeigt. Das heißt, zum Beispiel für ein ursprüngliches $\delta = 0.01$ (δ 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).

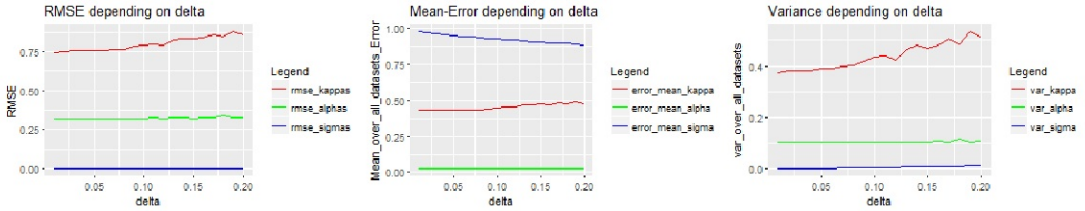

Abbildung 2.4: Veränderung RMSE, mittlerer Fehler, Varianz der Schätzungen mit steigendem δ

Zusätzlich zur Abb. 2.4, kann man sich im folgenden Teil der Anwendungen, die oben beschriebenen Schätzungen für ein größeres δ 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 δ 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.

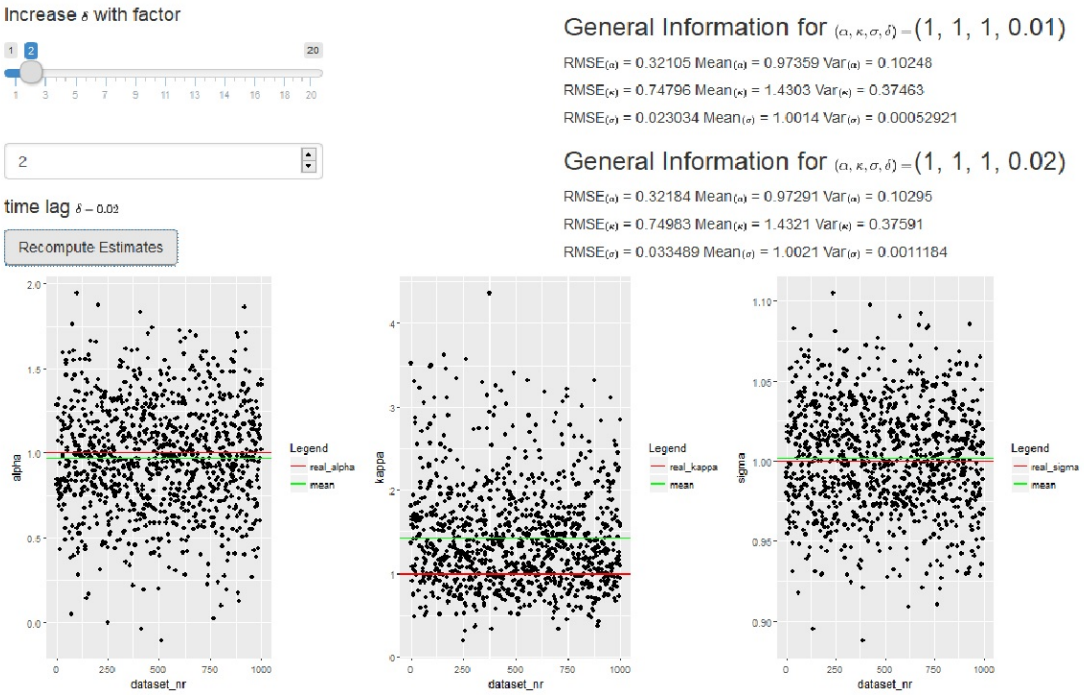

Abbildung 2.5: Neu berechnete Schätzer, mit neuem um Faktor *"Increase δ with factor"* vergrößertem δ

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

Anmerkung:
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

```
1
2  # SOFTWAREPROJEKT OU-Prozesse
3  # Josef Starkmann
4  # dX_t = \kappa(\alpha-X_t)dt + \sigma dB(t)
5  # X_0 = a
6  #
7  # This is the user-interface definition of a Shiny web application. You can
8  # run the application by clicking 'Run App' above.
9  #
10 # Find out more about building applications with Shiny here:
11 #
12 #    http://shiny.rstudio.com/
13 #
14
15 # IMPORT SHINY LIBRARY
16 library(shiny)
17 X=# Define UI for application that draws a histogram
18 shinyUI(fluidPage(
19
20   # Application title
21   # TITLE
22   withMathJax(titlePanel("Computation of $$dX_t = \\kappa(\\alpha-X_t)dt +
         \\sigma dB(t)$$ $$ X_0 = a$$")),
23   # SHORT HELPTEXT WITH GENERAL INFORMATION
24   helpText("Softwareprojekt OU-Process - Josef Starkmann"),
25
26   # CREATE A SIDEBARLAYOUT WITH SIDEBAR PANEL
27   sidebarLayout(
28     sidebarPanel(
29       ###### Parameter ###############
30       # ANZAHL Realisierungen pro Datensatz n
31
32       # ANZAHL DATENSAETZE  D
33
34       # X_0 = a
35
36       # \kappa
37
38       # \alpha
39
40       # \sigma
41
42       # \delta, dt
43       ################################
44        # CREATE ALL IMPORTANT INPUT FIELDS
```

```
45          numericInput("D",h4('number of time series\' D'),value = 1000, step
                =1,width = '50%'),#'
46          numericInput("n",h4('steps per time series n'),value = 1000, step
                =1,width = '50%'),
47          numericInput("a",h4(withMathJax("starting value \\( a \\)")),value =
                 0, step =0.01,width = '50%'),
48          numericInput("delta",h4(withMathJax("time lag \\( \\delta \\)")),
                value = 1e-2, step =1e-5,width = '50%'),
49          numericInput("kappa",h4(withMathJax("\\( \\kappa \\)")),value = 1,
                step =0.01,width = '50%'),
50          numericInput("alpha",h4(withMathJax("\\( \\alpha \\)")),value = 1,
                step =0.01,width = '50%'),
51          numericInput("sigma",h4(withMathJax("\\( \\sigma \\)")),value = 1,
                step =0.01,width = '50%'),
52          numericInput("seed",div('seed',style = "color:green"),value =
                3141593,step = 1, width = '50%'),
53          actionButton("create_data_button","Create Data and Compute Estimates
                ")
54
55      ),
56
57      # CREATE MAINPANEL WITH PLOTS AND INFORMATION
58      mainPanel(
59         #numericInput with Number of the Plot to show(can be one of the D
               Datasets)
60          numericInput("plt",h4('index of shown plot'),value = 10,min = 1,max
               = 20, step =1,width = '50%'),
61
62          # Plot of the choosen process, can be changed dynamically
63          plotOutput("process_plot"),
64
65          # Plot of the estimatetes of alpha, kappa, sigma +mean_error_
               depending on delta and variance depending on delta
66          plotOutput("distPlot"),
67
68          # FLUID ROW WITH TWO COLUMNS OF WIDTH 6 EACH ONE ROW HAS WIDTH 12
69          fluidRow(
70            # FIRST LEFT ROW WITH THREE ELEMENTS A SLDER FOR DELTA, a
                  correlating numeric input with the same value and a button
71           column(6,
72                  sliderInput("delta_new_slide",h4(withMathJax("Increase \\(
                       \\delta \\) with factor")),min = 1, max = 100, value
                       =1),
73                  numericInput("delta_new_in", label = "",value = 1),
74                  uiOutput('text_delta'),
75                  actionButton("recompute","Recompute Estimates")
76                  ),
77            # SECOND OUTPUT IN THE COLUMN WITH THE GENERAL INFOS
78           column(6,uiOutput('infos'),uiOutput('infos2'))
79
```

```
80
81          ),
82          # FURTHER PLOTS THAT OCCUR AFTER WE CLICK RECOMPUTE ESTIMATES
83          plotOutput("further_plots")
84        )
85    )
86 ))
```

OUProcess_Estimation/UI.R

A.2. Server.R

```
1  # SOFTWAREPROJEKT OU-Prozesse
2  # Josef Starkmann
3  # dX_t = \kappa(\alpha-X_t)dt + \sigma dB(t)
4  # X_0 = a
5  #
6  # This is the server logic of a Shiny web application. You can run the
7  # application by clicking 'Run App' above.
8  #
9  # Find out more about building applications with Shiny here:
10 #
11 #    http://shiny.rstudio.com/
12 #
13
14
15
16
17
18
19 # IMPORT NECESSARY LIBRARIES
20 library(shiny)
21 library(ggplot2)
22 library(sde)
23 library(grid)
24
25 #VARIABLE TO CHECK IF INITIALIZATION OR BUTTON CLICKED
26 is.init <<- TRUE
27
28 print_output = function(b_in,input_in,delta_first_in){
29   #check if first initial call or second "recompute" call
30   if(substitute(b_in) == 'b()') del = delta_first_in else del = delta_first
       _in*input_in$delta_new_in #else substitute(b_in) =b_new()
31   str_headline = paste("General Information for ","\\( (\\alpha,\\kappa,\\
       sigma,\\delta) =\\)(",toString(input_in$alpha),", ",toString(input_in
       $kappa),", ",toString(input_in$sigma),", ",toString(del),") ",sep =""
       )
32   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="
      ")
33  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="
      ")
34  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="
      ")
35  # count nas
36  number_nas=sum(is.na(b_in$kappas))
37  if(number_nas == 0){
38  list(
39    h3(withMathJax(str_headline)),
40    h5(withMathJax(str_alpha)),
41    h5(withMathJax(str_kappa)),
42    h5(withMathJax(str_sigma)))
43 }
44 else{# if nas occur
45  str_star= paste("*",toString(number_nas)," NAs occured! The marked means
      and variances are computed without these NAs!")
46  list(
47  h3(withMathJax(str_headline)),
48  h5(withMathJax(str_alpha)),
49  div(h5(withMathJax(str_kappa)),style = "color:red"),
50  div(h5(withMathJax(str_sigma)),style = "color:red"),
51  div(str_star,style = "color:red"))
52  }
53 }
54
55
56
57 # FUNCTION THAT CREATES THE OU PROCESSES
58 # dX_t = \kappa(\alpha-X_t)dt + \sigma dB(t)
59 # X_0 = a
60
61 # ERZEUGEN DER DATENSAETZE
62 # i-ter Datensatz = X[i,]
63 # generate data
64
65 ######  Parameter  ################
66 # ANZAHL Realisierungen pro Datensatz n
67
68 # ANZAHL DATENSAETZE  D
69
70 # X_0 = a
```

```
71
72 # \kappa
73
74 # \alpha
75
76 # \sigma
77
78 # \delta , dt
79 ################################
80
81 generate_ou_processes = function(n,D,x_0,kappa,alpha,sigma,delta){
82   X_ = matrix (nrow = D, ncol = n+1)
83   # PARAMETER NEED TO BE ADAPTED
84   theta = c(kappa*alpha,kappa,sigma)
85   # USE REPLICATE FUNCTION INSTEAD OF FOR QUEUE FOR PERFORMANCE REASONS
86   X_ = replicate(D,{sde.sim(t0 = 0, T=x_0+n*delta,X0 = x_0,N = n, delta =
         delta,theta = theta, model="OU")})
87   return(t(X_))
88 }
89
90 # FUNCTION THAT COMPUTES THE ESTIMATIONS FOR A GIVEN DELTA
91 compute_ML_estimations = function(X, delta){
92   # NUMBER of points in one dataset
93   n = length(X[1,])-1
94   # number of datasets
95   D = length(X[,1])
96   alphas = 0
97   kappas = 0
98   sigmas = 0
99   betas1 = 0
100  betas2 = 0
101  betas3 = 0
102  # PARAMETER SCHaeTZEN vgl "Parameter Estimation and Bias Correction for
         Diffusion Process"
103  # ITERATE THROUGH ALL DATASETS
104  for (i in c(1:D)){
105    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)
106    beta2 = (sum(X[i,-1]-beta1*X[i,-(n+1)])/n)/(1-beta1)
107    beta3 = sum((X[i,-1] -beta1*X[i,-(n+1)]-beta2*(1-beta1))**2)/n
108
109    kappa_est = - 1/delta *log(beta1)
110    alpha_est =  beta2
111    sigma_est = 2*kappa_est*beta3/(1-beta1**2)
112
113    betas1[i] = beta1
114    betas2[i] = beta2
115    betas3[i] = beta3
116    alphas[i] = alpha_est
117    kappas[i] = kappa_est
```

```
118     sigmas[i] = sigma_est
119   }
120   # RETURN RESULTS IN LIST
121   return(list(alphas=alphas,kappas=kappas,sigmas=sqrt(sigmas),betas1=betas1
          ,betas2=betas2,betas3=betas3))
122 }
123
124 # RETURNS VARIANCE, MEAN, RMSE PLOT DEPENDING ON TIME DIFFERENCE
125 # input: GENERATED OU PROCESSES and first delta
126 # output: vector with all deltas (whole number products of delta_first) and
        means and variances of the alphas,kappas, sigmas
127 compute_mean_variance_time_plot = function(X,delta_first,input){
128   n = length(X[1,])
129   rmse_kappas = 0
130   mean_kappas = 0
131   var_kappas  = 0
132   rmse_alphas = 0
133   mean_alphas = 0
134   var_alphas  = 0
135   rmse_sigmas = 0
136   mean_sigmas = 0
137   var_sigmas = 0
138   deltas = 0
139   #
140   # # iterate through dataset take every i-th point (at least 50)
141   for (i in c(1:(n%/%50))){# atleast 50 points
142     # new delta
143     deltas[i] = delta_first*i
144     # new estimator with every i-th Point
145     est = compute_ML_estimations(X[,seq(1,n,by = i)],delta = deltas[i])
146     # compute all means and variances of the estimated parameters possible
          nas removed
147
148     mean_kappas[i] = mean(est$kappas,na.rm=TRUE)
149     #COMPUTE RMSE
150     rmse_kappas[i] = sqrt(mean((est$kappas-input$kappa)**2,na.rm=TRUE))
151     var_kappas[i]  = var(est$kappas,na.rm=TRUE)
152
153     mean_alphas[i] = mean(est$alphas,na.rm=TRUE)
154     #COMPUTE RMSE
155     rmse_alphas[i] = sqrt(mean((est$alphas-input$alpha)**2,na.rm=TRUE))
156     var_alphas[i]  = var(est$alphas,na.rm=TRUE)
157
158     mean_sigmas[i] = mean(est$sigmas,na.rm=TRUE)
159     #COMPUTE RMSE
160     mean_sigmas[i] = sqrt(mean((est$sigmas-input$sigma)**2,na.rm=TRUE))
161     var_sigmas[i] = var(est$sigmas,na.rm=TRUE)
162   }
163   # RETURN RESULTS IN LIST
```

```
164   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))
165 }
166
167
168 # Define server logic
169 shinyServer(function(input, output,session){
170
171   #CREATE REACTIVE VARIABLE GETS VALUE IF AND ONLY IF input$create_data_
          button is clicked
172   delta_first <- eventReactive(input$create_data_button,{
173     isolate({
174       print("IN DELTA FIRST")
175       set.seed(input$seed)
176       input$delta
177     })
178   },ignoreInit = FALSE,ignoreNULL = FALSE)
179
180   #CREATE REACTIVE VARIABLE GETS VALUE IF AND ONLY IF input$create_data_
          button is clicked
181   #loads file "init.rds" on initialization (FASTER!) and computes file when
           button is clicked
182   X <- eventReactive(input$create_data_button,{
183       isolate({
184       #LOADS DATASET AT INITALIZATION
185       if(is.init){
186         is.init <<- FALSE
187         fil = file("init.rds")
188         temp = readRDS(fil)
189         close(fil)
190         temp
191       }
192       else{
193       #CREATES DATASET ON input$create_data_button_clicked
194         generate_ou_processes(input$n,input$D,input$a,input$kappa,input$alpha,
              input$sigma,input$delta)
195       }
196       })},ignoreInit = FALSE,ignoreNULL = FALSE)
197
198   #CREATE REACTIVE VARIABLE GETS VALUE IF AND ONLY IF X changes
199   #b <- eventReactive(input$create_data_button,{isolate({
200   b <- eventReactive(X(),{isolate({
201     compute_ML_estimations(X()[,seq(0,input$n,1)],delta = input$delta)
202   })},ignoreInit = FALSE,ignoreNULL = FALSE)
203
204
205   #CREATE REACTIVE VARIABLE GETS VALUE IF AND ONLY IF X has changed
206   b_new <- eventReactive(list(input$recompute, X()) ,{isolate({
```

```
207       print("IN COMPUTE ML2")
208       compute_ML_estimations(X()[,seq(0,input$n,input$delta_new_in)],delta =
              input$delta_new_in*delta_first())
209       })},ignoreInit = FALSE,ignoreNULL = FALSE)
210
211
212 #SLIDER AND NUMERIC INPUT WITH SAME VALUE
213   observe({
214       updateSliderInput(session, "delta_new_slide", value = input$delta_new_
              in)
215     },priority = 10)
216
217     observe({
218       updateNumericInput(session, "delta_new_in", value = input$delta_new_
              slide)
219     },priority = 10)
220
221
222       # FIRST PLOT OF ONE TIME SERIES, POSSIBILITY TO SEE FURTHER TIME SERIES
              OF THE DATASET WITH THE input$plt numericInput
223       output$process_plot <- renderPlot({
224         # dependency necessary in order to replot if plot is changed or new
                data created
225         input$plt
226         X()
227         #print(is.null(X()))
228         #print(X())
229         isolate({
230           t =seq(0,input$n*delta_first(),input$delta)
231           y = X()[input$plt,]
232
233           # Create plot with title
234           title = bquote(list(kappa==.(input$kappa),alpha==.(input$alpha),
                  sigma==.(input$sigma),a==.(input$a)))
235           p_1 = qplot(x=t,y=y,geom = 'line')+ggtitle(title)
236
237           # Create GRIDLayout and add plot
238           grid.newpage()
239           pushViewport(viewport(layout = grid.layout(1,1)))
240           vplayout = function(x,y) viewport(layout.pos.row =x,layout.pos.col=
                  y)
241           print(p_1,vp = vplayout(1,1))
242         })
243
244       })
245       output$distPlot <- renderPlot({
246
247        # dependecy to X()
248         X()
249        # isolate the rest
```

```
250     isolate({
251     # UPDATE MAX VALUE OF SLIDER
252     min_points = 50 #atleast 50 points
253     updateNumericInput(session,"delta_new_in", max = input$n%/%min_points)
254     updateSliderInput(session,"delta_new_slide", max = input$n%/%min_
            points)
255
256     # CREATE FIRST PLOT WITH ALPHAS AND MEAN AND REAL ALPHA (THE ONE THE
            PROCESSES WERE CREATED)
257     p_2 = qplot(y=b()$alphas, x = c(1:input$D), geom = 'point',xlab="
            dataset_nr",ylab='alpha')
258     p_2 = p_2 + geom_hline(aes(yintercept = input$alpha,color = 'real_
            alpha'))
259     p_2 = p_2 +geom_hline(aes(yintercept = mean(b()$alphas),color = 'mean'
            ))
260     # ADD LEGEND AND COLORS
261     p_2 =p_2 + scale_color_manual("Legend",values = c('real_alpha'='red','
            mean'='green'),breaks = c('real_alpha','mean'))
262
263     # CREATE SECOND PLOT WITH KAPPAS AND MEAN AND REAL KAPPA(THE ONE THE
            PROCESSES WERE CREATED)
264     p_3 =qplot(y=b()$kappas, x = c(1:input$D),geom = 'point',xlab="dataset
            _nr",ylab='kappa')
265     p_3 =p_3 + geom_hline(aes(yintercept = input$kappa,color="real_kappa")
            )
266     p_3 =p_3 + geom_hline(aes(yintercept = mean(b()$kappas),color = 'mean'
            ))
267     # ADD LEGEND AND COLORS
268     p_3 =p_3 + scale_color_manual("Legend",values = c('real_kappa'='red','
            mean'='green'),breaks = c('real_kappa','mean'))
269
270     # CREATE THIRD PLOT WITH SIGMAS AND MEAN AND REAL SIGMA(THE ONE THE
            PROCESSES WERE CREATED)
271     p_4 = qplot(y=b()$sigmas, x = c(1:input$D),geom = 'point',xlab="
            dataset_nr",ylab='sigma' )
272     p_4 = p_4 + geom_hline(aes(yintercept = input$sigma,color="real_sigma"
            ))
273     p_4 = p_4 + geom_hline(aes(yintercept = mean(b()$sigmas),color = 'mean
            '))
274     # ADD LEGEND
275     p_4 =p_4 + scale_color_manual("Legend",values = c('real_sigma'='red','
            mean'='green'),breaks = c('real_sigma','mean'))
276
277     # COMPUTE ESTIMATIONS BY TAKING ONLY EVERY 2nd, 3thd,...n%/%50th POINT
278     results = compute_mean_variance_time_plot(X(),delta_first(),input)
279
280     #CREATE PLOTS WITH RMSE DEPENDING ON DELTA (delta=delta_first,....
            delta=n%/%50*delta_first)
281     dfm = data.frame(delta=results$deltas,rmse_kappas=results$rmse_kappas,
            rmse_alphas = results$rmse_alphas,rmse_sigmas = results$rmse_
```

```
        sigmas)
282     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'))
283     # ADD LEGEND/COLORS
284     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'))
285     # ADD LABELS
286     p_5 = p_5 +ylab("RMSE")+ggtitle('RMSE depending on delta')
287
288
289     # CREATE PLOT WITH MEANS AND VARIANCES DEPENDING ON DELTA (delta=delta
            _first,.... delta=n%/%50*delta_first)
290     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))
291     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'))
292     # ADD LEGEND/COLORS
293     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
            '))
294     # ADD LABELS
295     p_6 = p_6 +ylab("Mean_over_all_datasets_Error")+ggtitle('Mean-Error
            depending on delta')
296
297     # VARIANCE PLOT
298     dfm = data.frame(delta=results$deltas,var_kappas=results$var_kappas,
            var_alphas = results$var_alphas,var_sigmas = results$var_sigmas)
299     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'))
300     # ADD LEGEND/COLORS
301     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'))
302     # ADD LABELS
303     p_7 =p_7 + ylab("var_over_all_datasets")+ggtitle('Variance depending
            on delta')#, y="Var_over_all_datasets",title ="Variance by delta")
304
305     # ADD GRID LAYOUT AND PLOTS AT RIGHT POSITION
306     grid.newpage()
307     pushViewport(viewport(layout = grid.layout(2,3)))
308     vplayout = function(x,y) viewport(layout.pos.row =x,layout.pos.col=y)
309     print(p_2,vp = vplayout(1,1))
310     print(p_3,vp = vplayout(1,2))
311     print(p_4,vp = vplayout(1,3))
```

```
312       print(p_5,vp = vplayout(2,1))
313       print(p_6,vp = vplayout(2,2))
314       print(p_7,vp = vplayout(2,3))
315
316       }) #isolation end
317 })
318
319
320
321   # DEFINE PLOTS FOR A BIGGER TIME DIFFERENCE DELTA OF THE GIVEN DATA SETS
322   # e.g. delta_first = 0.01 input$delta_new_in = 2 -> delta_new = 0.2
323   output$further_plots <- renderPlot({
324
325     # PLOTS ARE CHANGED IF AND ONLY IF ONE OF THOSE BUTTONS PUSHED
326     input$create_data_button
327     input$recompute
328     isolate({
329     # CREATE FIRST PLOT WITH ALPHAS AND MEAN AND REAL ALPHA OF THE NEW
          ESTIMATION
330     p_2 = qplot(y=b_new()$alphas, x = c(1:length(b_new()$alphas)), geom = '
          point',xlab="dataset_nr",ylab='alpha')
331     p_2 = p_2 + geom_hline(aes(yintercept = input$alpha,color = 'real_alpha
          '))
332     p_2 = p_2 + geom_hline(aes(yintercept = mean(b_new()$alphas), color = '
          mean'))
333     p_2 = p_2 + scale_color_manual("Legend",values = c('real_alpha'='red','
          mean'='green'),breaks = c('real_alpha','mean'))
334
335     # CREATE FIRST PLOT WITH KAPPAS AND MEAN AND REAL KAPPAS OF THE NEW
          ESTIMATION
336     p_3 = qplot(y=b_new()$kappas, x = c(1:length(b_new()$alphas)),geom = '
          point',xlab="dataset_nr",ylab='kappa')
337     p_3 = p_3 + geom_hline(aes(yintercept = input$kappa,color = 'real_kappa
          '))
338     p_3 = p_3 + geom_hline(aes(yintercept = mean(b_new()$kappas,na.rm=TRUE)
          ,color = 'mean'))
339     # ADD LEGEND/COLORS
340     p_3 = p_3 + scale_color_manual("Legend",values = c('real_kappa'='red','
          mean'='green'),breaks = c('real_kappa','mean'))
341
342     # CREATE FIRST PLOT WITH SIGMAS AND MEAN AND REAL SIGMAS OF THE NEW
          ESTIMATION
343     p_4 = qplot(y=b_new()$sigmas, x = c(1:length(b_new()$alphas)),geom = '
          point' ,xlab="dataset_nr",ylab='sigma')
344     p_4 = p_4 + geom_hline(aes(yintercept = input$sigma,color = 'real_sigma
          '))
345     p_4 = p_4 + geom_hline(aes(yintercept = mean(b_new()$sigmas,na.rm=TRUE)
          ,color = 'mean'))
346     # ADD LEGEND/COLORS
```

```
347     p_4 = p_4 + scale_color_manual("Legend",values = c('real_sigma'='red','
            mean'='green'),breaks = c('real_sigma','mean'))
348
349     # ADD GRID LAYOUT AND PLOTS AT RIGHT POSITION
350     grid.newpage()
351     pushViewport(viewport(layout = grid.layout(1,3)))
352     vplayout = function(x,y) viewport(layout.pos.row =x,layout.pos.col=y)
353     print(p_2,vp = vplayout(1,1))
354     print(p_3,vp = vplayout(1,2))
355     print(p_4,vp = vplayout(1,3))
356     })
357   })
358
359   # PRINT UPDATED DELTA VALUE
360   output$text_delta <- renderUI({
361     str = paste("time lag \\( \\delta =",toString(input$delta_new_in*delta_
            first()), " \\) ",sep="")
362     h4(withMathJax(str))
363   })
364
365   # PRINT NEW GENERAL INFORMATION (first part)FIELD WHEN  create_data_
        button is clicked
366   output$infos <- renderUI({
367       input$create_data_button
368       isolate({
369         print_output(b(),input,delta_first())
370       })
371     }
372   )
373   # PRINT NEW General INFORMATION FIELD (second part/scaled delta) WHEN
        Recompute  is clicked
374   output$infos2 <- renderUI({
375       input$recompute
376       input$create_data_button
377       isolate({
378         print_output(b_new(),input,delta_first())
379       })
380     }
381   )
382 })
```

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*