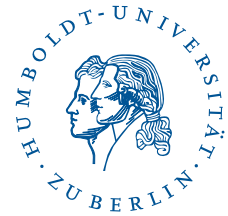


HUMBOLDT-UNIVERSITÄT ZU BERLIN



MATLAB-Programmcode zur Bachelorarbeit

# Die Karhunen-Loève-Zerlegung

Beispiele und Optimalität bezüglich des mittleren  
quadratischen Fehlers

eingereicht von

Tim Jaschek

Institut für Mathematik  
Humboldt Universität zu Berlin



```

1  %%This class is a collection of functions related to the ✓
Brownian%%
2  %%motion in the context of Karhunen-Loeve Expansion%%
3  %%written by Tim Jaschek as a part of%%
4  %% a presentation about the Levy construction of Brownian Motion%%
5  %% a presentation about the idea of Karhunen-Loeve Expansion%%
6  %% a part of his bachelor thesis%%
7
8  %%This Program is used to generate FIGURE 3 in the thesis%%
9  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10
11 classdef BrMb
12     properties (Constant)
13         steps = 1000 ;
14         time = linspace( 1/BrMb.steps, 1, BrMb.steps) ;
15     end
16     methods (Static)
17         function [Z,lambda,psi] = compute_KLT_components(n)
18             %%this function computes the analytically solved ✓
eigenvalues and
19             %%eigenfunctions of a Brownian motion
20             steps = BrMb.steps ;
21             Z = randn(1,n) ;
22             lambda = zeros(1,n) ;
23             psi = zeros(n,steps) ;
24             for j = 1:n
25                 lambda(j) = 1 / ((j-0.5) * pi);
26                 for k = 1:steps
27                     psi(j,k) = sqrt(2)*sin((j-0.5)*pi*k/steps) ;
28                 end
29             end
30         end
31         function BM = BrownianMotion(n, steps)
32             %%using the previous function, this function computes ✓
the n-th
33             %%partial sum of the Karhunen-Loeve Expansion of ✓
Brownian motion
34             BM = zeros(1,steps) ;
35             [Z,lambda,psi] = BrMb.compute_KLT_components(n) ;
36             for j = 1:n
37                 for k = 1:steps
38                     BM(k) = BM(k) + Z(j)*lambda(j)*psi(j,k) ;
39                 end
40             end

```

```

41         %OPPORTUNITY to plot:
42         %figure
43         %plot( BrMb. time, BM)
44         %xlabel(' time')
45         %str=sprintf(' Brownian Mbti on via Karhunen- Loeve✓
Approx. for n = %d', n);
46         %title(str)
47     end
48     function plotDevel op()
49         %generates the first plot of FIGURE 3
50         BM = zeros( 1, BrMb. steps) ;
51         [Z, lambda, psi] = BrMb. compute_KLT_components( 80);
52         figure
53         subplot( 1, 2, 1)
54         title(' Brownian Mbti on via Karhunen- Loeve Approx. for ✓
different n')
55         hold on;
56         for j = 1: 80
57             for k = 1: BrMb. steps
58                 BM( k) = BM( k) + Z( j) * lambda( j) * psi( j, k);
59             end
60             if j == 4 || j == 16 || j == 32 || j == 64
61                 plot( BrMb. time, BM)
62             end
63         end
64         legend(' n = 4', ' n = 16', ' n = 32', ' n = 64')
65         hold off;
66     end
67     function plotMbvi e()
68         %an exciting movie that shows the behaviour of the ✓
partial sums
69         %of the Karhunen- Loeve Expansion is generated here
70         BM = zeros( 1, BrMb. steps) ;
71         [Z, lambda, psi] = BrMb. compute_KLT_components( 500);
72         for j = 1: 500
73             for k = 1: BrMb. steps
74                 BM( k) = BM( k) + Z( j) * lambda( j) * psi( j, k);
75             end
76             plot( BrMb. time, BM)
77             title(' Animation of KLT Series');
78             pause( 0. 01)
79         end
80     end
81     function LevyBase()

```

```

82      %plots the Schauder base
83      Bn = [ 0, 1] ;
84      List = [ 0, 1] ;
85      figure
86      title('Schauder Base - Base from the Construction by
L·vy');
87      xlabel('time');
88      hold on;
89      plot(List, Bn);
90      for n = 1:4
91          if n==1
92              Bn = [ Bn, 0];
93              List = linspace(0, 1, 3);
94          else
95              Bn = [ Bn, 1 , 0];
96              List = linspace(0, 1, length(Bn));
97          end
98          plot(List, Bn);
99      end
100     hold off;
101 end
102 function KLTBase()
103     %plots the Karhunen-Loeve Eigenfunction base
104     [Z, lambda, psi] = Br Mb. compute_KLT_components(6);
105     figure
106     title('Fourier Base - Base from Karhunen-Lo·ve
Theorem');
107     xlabel('time');
108     hold on;
109     for j = 1:6
110         plot(Br Mb. time, psi(j, :))
111     end
112     hold off;
113 end
114 function Levy()
115     %generates the second plot of FIGURE 3
116     D_0 = linspace(0, 1, 2^0 + 1) ; %intervals of dyadic
points
117     B_0 = [ 0 , randn]; %approximation of Brownian Mbt in
step 0
118     subplot(1, 2, 2);
119     title('Brownian Mbtion via Levy Construction for
different n')
120     hold on;

```

```

121         for n=1:6
122             D_next = linspace(0, 1, 2^n+1);
123             B_next = zeros(1, length(D_next));
124             j=1;
125             for i=1:2^n+1
126                 if mod(i, 2) == 0;
127                     B_next(i) = sqrt(1/(2^(n+1))) * randn;
128                 else
129                     B_next(i) = B_0(j);
130                     j=j+1;
131                 end
132             end
133             for i=1:2^n+1
134                 if mod(i, 2) == 0;
135                     B_next(i) = B_next(i) + (B_0(i/2) + B_0(i/2+1)) /
136 /2;
137                 end
138             end
139             D_0 = D_next;
140             B_0 = B_next;
141             if n==2 || n==4 || n==5 || n==6
142                 plot(D_0, B_0)
143             end
144             legend('n = 4', 'n = 16', 'n = 32', 'n = 64')
145             hold off;
146         end
147     function CoV()
148         %plots the covariancefunction of Brownian motion
149         [X, Y] = meshgrid(0:.05:1);
150         Z = min(X, Y);
151         surf(X, Y, Z)
152         title('Kovarianzfunktion der Brownschen Bewegung')
153         xlabel('time')
154     end
155     function KLTlambdas()
156         %plots the Eigenvalues
157         [Z, lambda, psi] = BrMb.compute_KLT_components(10);
158         plot(linspace(1, 10, 10), lambda, '*')
159         title('KLT Coefficients')
160         xlabel('index')
161         ylabel('lambda_i')
162     end
163     function CoVBB()

```

```

164         %plots the covariancefunction of Brownian bridge
165         [X, Y] = meshgrid(0:.05:1);
166         Z = min(X, Y) - X.*Y
167         surf(X, Y, Z)
168         title('Kovarianzfunktion der Brownschen Bræcke')
169     end
170     function Mercer()
171         %plots different approximations of the✓
covariancefunction of
172         %Brownian motion using Mercers theorem
173         [Z, lambda, psi] = BrMb.compute_KLT_components(10);
174         figure
175         A = zeros(20, 20);
176         title('Mercers Theorem - covariance function of ✓
Brownian Mbtion');
177         hold on;
178         for n = 1:4
179             for i = 1 : 20
180                 for j = 1:20
181                     A(i, j) = A(i, j) + lambda(n)*psi(n, 50*i) ✓
*psi(n, 50*j);
182                 end
183             end
184         end
185         [X, Y] = meshgrid(0.05:0.05:1);
186         surf(X, Y, A)
187     end
188     function value = mother(t)
189         %%%Haar mother function%%
190         if ((t<0.5) && (t >= 0))
191             value = 1;
192         elseif ((t>=0.5) && (t<1))
193             value = -1;
194         else
195             value = 0;
196         end
197     end
198     function psi = Haar(N)
199         steps = linspace(1/N, 1, N);
200         %%%Use the Haar Mbtion%%
201         psi(:, 1) = ones(1, N);
202         psi(N, 1) = 0;
203         n=0;
204         k=0;

```

```
205         count =2;
206     while count <N+1
207         max = 2^n;
208         if k == max
209             n=n+1;
210             k=0;
211             continue
212         end
213         for t=1: N
214             psi ( t, count ) = 2^( n/ 2) *Br Mb. mother ( 2^n*steps ✓
(t) - k);
215         end
216         count = count+1;
217         k=k+1;
218     end
219 end
220 end
221 end
```



```

1 %%This Programm generates a Plot eigenvalues and eigenfunctions of
2 %%the induced integral operators of different kernels
3 %%written by Tim Jaschek as a part of his bachelor thesis%%
4
5 %%This Programm is used to generate FIGURE 2 in the thesis%%
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7
8 load the class Kernels
9 Kernels;
10
11 %Parameter for accuracy
12 N=50;
13
14 figure
15 for i=1:3;
16     Mat = Kernels.KMat(i, N);
17     [lambda, Phi] = Kernels.trapez_Scheme(Mat);
18     subplot(3,3,[1+3*(i-1) 2+3*(i-1)]);
19     for j=1:6
20         hold on;
21         plot(linspace(0,1,N+2),Phi(:,j));
22     end
23     if i ==1
24         title('First 6 Eigenfunctions');
25         ylabel('K(s,t)=min(s,t)');
26     elseif i == 2
27         ylabel('K(s,t)=min(s,t) - st');
28     else
29         ylabel('K(s,t)=exp(-|s-t|)');
30     end
31     hold off;
32     subplot(3,3,3*i);
33     plot(linspace(1,10,10),lambda(1:10),'o','color','red');
34     if i ==1
35         title('First 10 Eigenvalues');
36     end
37 end
38
39
40
41

```

```

1 %%This class gives tools for image compressing via Principal%%
2 %%Component Analysis (KLT)%%
3 %%written by TimJaschek as a part of his bachelor thesis%%
4
5 %%Used to generate FIGURE 6 %%
6 %%..to generate it, type the following in your MATLAB command:
7 %%Image;
8 %%Image.program();
9 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10
11 classdef Image
12     properties (Constant)
13     end
14     methods (Static)
15         function X = load_space()
16             %load an image
17             X = imread('bluemarbel.jpg');
18             %show the image
19             figure
20             image(X)
21         end
22         function B = blurBW(X)
23             %function to blur an B/W image
24             [m n] = size(X);
25             X = double(X);
26             %to 25%
27             mm=m/2;
28             nn = n/2;
29             B=zeros(mm,nn);
30             for i=1:mm
31                 for j=1:nn
32                     B(i,j) = 1/4 *(X(2*i-1,2*j-1) + X(2*i,2*j-1) + X(2*i-1,2*j) + X(2*i,2*j));
33                 end
34             end
35             B = uint8(B);
36         end
37         function Z = blur(X)
38             %function to blur an R/G/B image
39             XR = X(:,:,1);
40             XG = X(:,:,2);
41             XB = X(:,:,3);
42             RN = Image.blurBW(XR);
43             GN = Image.blurBW(XG);

```

```

44         BN = Image.blurBW(XB);
45         Z = cat(3, RN, GN, BN);
46     end
47     function X_trans = PCA(X, k)
48         %function for image compression via KLT of a B/W image
49         [m n] = size(X);
50         X = double(X);
51         X_hat = X;
52         mean = zeros(1, n);
53         K = zeros(n, n);
54         %%correct mean%%
55         for i=1:n
56             mean(i) = sum(X(:, i))/m;
57             X_hat(:, i) = X_hat(:, i) - mean(i);
58         end
59         %%covariance matrix%%
60         for i=1:n
61             for j=1:n
62                 K(i, j) = (1/(n-1))*dot((X_hat(:, i)), (X_hat(:, j)));
63             end
64         end
65         %%Eigenvalues and Eigenvectors%%
66         [V, D] = eig(K);
67         [lambda, ind] = sort(diag(D), 'descend');
68         Phi = V(:, ind);
69         Phi = Phi(:, 1:k);
70         Phi_T = Phi.';
71         %%%Transform X %%%
72         Y = Phi_T*(X_hat);
73         X_trans = Phi*Y;
74         for i=1:n
75             X_trans(:, i) = X_trans(:, i) + mean(i);
76         end
77         X_trans = uint8(X_trans);
78     end
79     function Z = PCA_RGB(X, k);
80         %function for image compressing via KLT of an R/G/B
image
81         XR = X(:, :, 1);
82         XG = X(:, :, 2);
83         XB = X(:, :, 3);
84         RN = Image.PCA(XR, k);
85         GN = Image.PCA(XG, k);

```

```

86         BN = Image.PCA(XB, k);
87         Z = cat(3, RN, GN, BN);
88     end
89     function program()
90         %generates FIGURE 6
91         disp('loading image')
92         X = Image.load_space();
93         figure
94         set(gca, 'XTickLabel', [], 'XTick', [])
95         h = subplot(2, 2, 1)
96         imshow(X)
97         disp('data compressing...')
98         Z = Image.PCA_RGB(X, 400);
99         hh = subplot(2, 2, 2)
100        imshow(Z)
101        Z = Image.PCA_RGB(X, 200);
102        hhh = subplot(2, 2, 3)
103        imshow(Z)
104        Z = Image.PCA_RGB(X, 100);
105        hhhh = subplot(2, 2, 4)
106        imshow(Z)
107        p = get(h, 'pos');
108        pp = get(hh, 'pos');
109        ppp = get(hhh, 'pos');
110        pppp = get(hhhh, 'pos');
111        p([3, 4]) = p([3, 4]) + [0.1 0.1];
112        set(h, 'pos', p);
113        pp([3, 4]) = pp([3, 4]) + [0.1 0.1];
114        set(hh, 'pos', pp);
115        ppp([3, 4]) = ppp([3, 4]) + [0.1 0.1];
116        set(hhh, 'pos', ppp);
117        pppp([3, 4]) = pppp([3, 4]) + [0.1 0.1];
118        set(hhhh, 'pos', pppp);
119    end
120 end
121 end
122

```

```

1 %%This class is a collection of different Kernels and integration✓
methods%%
2 %%to solve the Fredholm integralequation. It also provides a%%
3 %%Merverapproximation for the kernels. %%
4 %%written by TimJaschek as a part of his bachelor thesis%%
5
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7 classdef Kernels
8     properties (Constant)
9     end
10    methods (Static)
11        function K_st = Kernel(i,s,t)
12            %this is a collection of covariance functions%
13            if i == 1
14                %Brownian Motion%
15                K_st = min(s,t);
16            elseif i==2
17                %Brownian Bridge%
18                K_st = min(s,t) - s*t;
19            elseif i==3
20                %exponential kernel%
21                K_st = exp(-abs(t-s));
22            else
23                K_st = 0;
24            end
25        end
26        function Mat = KMat(i,N)
27            %for given N and i, this function will generate an NxN✓
matrix
28            %for the i-th kernel.
29            Mat = zeros(N+2,N+2);
30            for j = 1:N+2
31                for k = 1:N+2
32                    %use symmetry to save operations
33                    if k<j
34                        Mat(j,k) = Mat(k,j);
35                    else
36                        Mat(j,k) = Kernels.Kernel(i,(j-1)/(N+1), ✓
(k-1)/(N+1));
37                    end
38                end
39            end
40        end
41        function [lambda,Phi] = uniform_Scene(K)

```

```

42      %UNIFORM SCENE
43      N = length( K) - 2;
44      sqW = sqrt( 1/( N+2)) * eye( N+2);
45      Mat = sqW*K*sqW
46      [ V, D]=ei g( Mat);
47      [ lambda, ind] = sort( di ag( D), ' descend' );
48      E_vectors = V(:, ind);
49      Phi = sqrt( N+2)*E_vectors;
50      %%bring the EV in the right direction%%
51      for i = 1:N+2
52          if Phi( 2, i)<0
53              Phi(:, i)=- Phi(:, i);
54          end
55      end
56  end
57  function [ lambda, Phi] = trapez_Scene( K)
58      %TRAPEZ SCENE
59      N = length( K) - 2;
60      sqW = sqrt( 1/( N+1)) * eye( N+2);
61      sqW( 1, 1) = sqrt( 1/( 2*( N+1)));
62      sqW( N+2, N+2) = sqW( 1, 1);
63      qW = sqrt( N+2)*eye( N+2);
64      qW( 1, 1) = sqrt( 2*( N+1));
65      qW( N+2, N+2) = qW( 1, 1);
66      Mat = sqW*K*sqW
67      [ V, D]=ei g( Mat);
68      [ lambda, ind] = sort( di ag( D), ' descend' );
69      E_vectors = V(:, ind);
70      Phi = qW*E_vectors;
71      %%bring the EV in the right direction%%
72      for i = 1:N+2
73          if Phi( 2, i)<0
74              Phi(:, i)=- Phi(:, i);
75          end
76      end
77  end
78  function [ lambda, Phi] = simpson_Scene( K)
79      %SIMPSON SCENE
80      N = length( K) - 2;
81      sqW = sqrt( 1/( 3*( N+1))) * eye( N+2);
82      qW = sqrt( 3*( N+1)) *eye( N+2);
83      for i = 2:N+1
84          sqW( i, i)=sqW( i, i)*sqrt( 2);
85          qW( i, i)=qW( i, i)/sqrt( 2);

```

```

86         end
87         for i = 2:(N+1)/2+1
88             j=2*(i-1);
89             sqW(j,j)=sqW(j,j)*sqrt(2);
90             qW(j,j)=qW(j,j)/sqrt(2);
91         end
92         Mat = sqW*K*sqW
93         [V,D]=eig(Mat);
94         [lambda,ind] = sort(diag(D),'descend');
95         E_vectors = V(:,ind);
96         Phi = qW*E_vectors;
97         %%bring the EV in the right direction%%
98         for i = 1:N+2
99             if Phi(2,i)<0
100                 Phi(:,i)=-Phi(:,i);
101             end
102         end
103     end
104     function K = MercerApprox(lambda,Phi,n)
105         %INPUT: lambda - Eigenvalues,
106         %        Phi - Eigenfunctions,
107         %        n - summations
108
109         %OUTPUT: K as approximation of covariance matrix
110         N = length(lambda)-2;
111         K=zeros(N+2,N+2);
112         for s=1:N+2
113             for t=1:N+2
114                 if t<s
115                     K(s,t) = K(t,s);
116                 else
117                     for i=1:n
118                         K(s,t) = K(s,t) + lambda(i)*Phi(s,i)
119 *Phi(t,i);
120                     end
121                 end
122             end
123         end
124     end
125 end
126 end
127

```

```

1 %%This Programm generates a Plot for different Kernels and ✓
partial %%
2 %%sums of Mercer's series for the covariance function%%
3 %%written by Tim Jäschek as a part of his bachelor thesis%%
4
5 %%This Programm is used to generate FIGURE 1 in the thesis%%
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7
8 %load the class Kernels
9 Kernels;
10
11 %Parameter for accuracy
12 N=16;
13
14 figure
15 for i=1:3;
16     Mat = Kernels.KMat(i, N);
17     [lambda, Phi] = Kernels.trapez_Scheme(Mat);
18     for j=1:3;
19         K=Kernels.MercerApprox(lambda, Phi, j+(j-1)^2);
20         subplot(4,3,i+3*(j-1));
21         surfc(linspace(0,1,N+2),linspace(0,1,N+2), ✓
K, 'edgeal pha', '1');
22         if j == 1
23             if i ==1
24                 title('K(s,t)=min(s,t)');
25                 zlabel('n=1');
26             elseif i ==2
27                 title('K(s,t)=min(s,t) - st');
28             else
29                 title('K(s,t)=exp(-|s-t|)');
30             end
31         elseif j == 2
32             if i == 1
33                 zlabel('n=3');
34             end
35         else
36             if i == 1
37                 zlabel('n=7');
38             end
39         end
40     end
41     subplot(4,3,i+9);
42     surfc(linspace(0,1,N+2),linspace(0,1,N+2), Mat, 'edgeal pha', '1');

```



```
43     if i == 1
44         zlabel('analytic');
45     end
46 end
47
48
49
50
```

```

1 %%This class is a collection of functions to expand a Brownian✓
motion wrt.
2 %%to different orthonormal bases and compute the Total Mean✓
Squared Error
3 %%for the approximation with the n-th partial sum
4 %%written by Tim Jaschek as a part of his bachelor thesis%%
5
6 %%Used to generate FIGURE 3 and the data for TABULAR 5.1 in this✓
thesis %%
7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8
9 classdef MSE
10     properties (Constant)
11         %Parameter for the partial sum
12         n = 20;
13
14         %Parameter for the accuracy of the Kernels
15         N = 1000;
16     end
17     methods (Static)
18         function approximation()
19             %load the classes BrMb and Kernels
20             BrMb;
21             Kernels;
22             %%%generate a Brownian Motion with N steps%%
23             p = 10000; %parameter for the accuracy of the BM
24             X = BrMb.BrownianMotion(p, MSE.N);
25             %%%get ORTHONORMAL BASES
26             %%%the analytic KLT Eigenfunctions
27             %%%the Haar Wavelets
28             %%%Eigenfunctions of Brownian Bridge
29             %%%Eigenfunctions of Exponential Kernel
30             [Z, lambda, psi] = BrMb.compute_KLT_components(MSE.N);
31             psi = psi.';
32             Haar = BrMb.Haar(1000);
33             [lambda, Bridge] = Kernels.trapez_Scene(Kernels.KMat(2, ✓
MSE.N));
34             [lambda, Exp] = Kernels.trapez_Scene(Kernels.KMat(3, MSE. ✓
N));
35             %%%Compute the Approximations%%
36             X_hat = MSE.Approx(X, psi(:, 1:MSE.n));
37             X_tilde = MSE.Approx(X, Haar(:, 1:MSE.n));
38             X_bridge = MSE.Approx(X, Bridge(:, 1:MSE.n));
39             X_exp = MSE.Approx(X, Exp(:, 1:MSE.n));

```

```

40     %%%plot both%%
41     figure
42     hold on;
43     plot(linspace(1/MBE.N, 1, MBE.N), X)
44     plot(linspace(1/MBE.N, 1, MBE.N), X_hat)
45     plot(linspace(1/MBE.N, 1, MBE.N), X_tilde)
46     plot(linspace(1/MBE.N, 1, MBE.N), X_bridge)
47     %plot(linspace(1/MBE.N, 1, MBE.N), X_exp)
48     hold off;
49 end
50 function meansquare()
51     %load the classes BrMb and Kernels
52     BrMb;
53     Kernels;
54
55     disp('Compute TMBE of Karhunen-Loeve-Base...')
56     [Z, lambda, psi] = BrMb.compute_KLT_components(MBE.N);
57     psi = psi.';
58     Me = zeros(1, MBE.N);
59     for i=1:30
60         X = BrMb.BrownianMotion(5000, MBE.N);
61         X_hat = MBE.Approx(X, psi(:, 1:MBE.n));
62         for j=1:MBE.N
63             Me(j) = Me(j) + (X(j) - X_hat(j))^2;
64         end
65     end
66     Me = Me/30;
67     TMe = 0;
68     h=1/MBE.N;
69     for j=1:MBE.N-1
70         TMe = TMe + h/2 * (Me(j) + Me(j+1));
71     end
72     TMe
73     %%%%%%%%%%Haar%%%%%%%%%
74     disp('Compute TMBE of Haar-Base...')
75     Haar = BrMb.Haar(1000);
76     Me = zeros(1, MBE.N);
77     for i=1:30
78         X = BrMb.BrownianMotion(5000, MBE.N);
79         X_hat = MBE.Approx(X, Haar(:, 1:MBE.n));
80         for j=1:MBE.N
81             Me(j) = Me(j) + (X(j) - X_hat(j))^2;
82         end
83     end

```

```

84     Me = Me/30;
85     TMe = 0;
86     h=1/MSE.N;
87     for j=1:MSE.N-1
88         TMe = TMe + h/2 * (Me(j) + Me(j+1));
89     end
90     TMe
91     %%%%%%%%%Bridge%%%%%%%%
92     disp('Compute TME of Brownian-Bridge-Base...')
93     [lambda, Bridge] = KernelS.trapez_Scene(KernelS.KMat(2, ✓
MSE.N));
94     Me = zeros(1, MSE.N);
95     for i=1:30
96         X = BrMb.BrownianMbtion(5000, MSE.N);
97         X_hat = MSE.Approx(X, Bridge(:, 1:MSE.n));
98         for j=1:MSE.N
99             Me(j) = Me(j) + (X(j)-X_hat(j))^2;
100        end
101    end
102    Me = Me/30;
103    TMe = 0;
104    h=1/MSE.N;
105    for j=1:MSE.N-1
106        TMe = TMe + h/2 * (Me(j) + Me(j+1));
107    end
108    TMe
109    %%%%%%%%%Exponential%%%%%%%%
110    disp('Compute TME of Exponential-Base...')
111    [lambda, Exp] = KernelS.trapez_Scene(KernelS.KMat(3, MSE. ✓
N));
112    Me = zeros(1, MSE.N);
113    for i=1:30
114        X = BrMb.BrownianMbtion(5000, MSE.N);
115        X_hat = MSE.Approx(X, Exp(:, 1:MSE.n));
116        for j=1:MSE.N
117            Me(j) = Me(j) + (X(j)-X_hat(j))^2;
118        end
119    end
120    Me = Me/30;
121    TMe = 0;
122    h=1/MSE.N;
123    for j=1:MSE.N-1
124        TMe = TMe + h/2 * (Me(j) + Me(j+1));
125    end

```

```

126         TMe
127     end
128     function X_hat = Approx(X, Phi);
129         %%assume X has N steps on [0,1] and we have at least n✓
130         Eigenvectors with N steps each%%
131         X_hat = zeros(1, MSE.N);
132         A = zeros(1, MSE.n);
133         h = 1/MSE.N;
134         for i=1:MSE.n
135             %%compute the integrals via trapez scene%%
136             for j=1:MSE.N-1
137                 A(i) = A(i) + h/2 * (X(j)*Phi(j,i) + X(j+1)✓
138                 *Phi(j+1,i));
139             end
140         end
141     for i=1:MSE.N
142         X_hat(i) = dot(A, Phi(i, 1:MSE.n));
143     end
144 end
145
146
147
148
149

```

```

1 %%This programm compares uniform trapez and Simpson-Sceme for %%
2 %%approximation of solutions to the Fredholm integral equation. %%
3 %%written by Tim Jaschek as a part of his bachelor thesis%%
4
5 %%Used to generate data for Tabular 6.1 and 6.2 %%
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7
8 %Import the class Kernels which contains some Kernels and
9 %integration scenes.
10 Kernels;
11
12 %Parameter for the Number of approximation steps
13 N = 45;
14
15 %Generation of different Kernels
16 BrownianMotion = Kernels.KMat(1, N);
17 BrownianBridge = Kernels.KMat(2, N);
18 ExponentialKer = Kernels.KMat(3, N);
19
20 %BROWNIAN MOTION
21 %Solve Fredholm integral equality with different Scenes
22 [lambda1, Phi1] = Kernels.uniform_Sceme(BrownianMotion);
23 [lambda2, Phi2] = Kernels.trapez_Sceme(BrownianMotion);
24 [lambda3, Phi3] = Kernels.simpson_Sceme(BrownianMotion);
25 %Compute analytic solutions for first Eigenvalues
26 lambda = [lambda1(1) lambda2(1) lambda3(1)];
27 Phi = [Phi1(:, 1) Phi2(:, 1) Phi3(:, 1)];
28 la = (2/pi)^2;
29 ph = zeros(N+2, 1);
30 for i=1:N+2
31     ph(i) = sqrt(2) * sin(0.5*pi*((i-1)/(N+2)));
32 end
33 plot(linspace(0, 1, N+2), ph, linspace(0, 1, N+2), Phi(:, 1))
34 %Compute the error terms
35 absolute_error_lambda = abs(la(1)-lambda)
36 relative_error_lambda = abs(la(1)-lambda)/la(1)*100
37 absolute_error_phi = zeros(1, 3);
38 relative_error_phi = zeros(1, 3);
39 for i=1:3
40     absolute_error_phi(i) = max(abs(ph-Phi(:, i)));
41     relative_error_phi(i) = absolute_error_phi(i)/max(abs(Phi(:, i),
42 i)));
43 end
43 absolute_error_phi

```

```

44 relative_error_phi *100
45
46 %BROWNIAN BRIDGE
47 %Solve Fredholm integral equality with different Scenes
48 [lambda1, Phi 1] = Kernel s. uniform_Scene( BrownianBridge);
49 [lambda2, Phi 2] = Kernel s. trapez_Scene( BrownianBridge);
50 [lambda3, Phi 3] = Kernel s. simpson_Scene( BrownianBridge);
51 %Compute analytic solutions for first Eigenvalues
52 lambda = [lambda1(1) lambda2(1) lambda3(1)];
53 Phi = [Phi 1(:, 1) Phi 2(:, 1) Phi 3(:, 1)];
54 la = (1/pi)^2;
55 ph = zeros(N+2, 1);
56 for i=1:N+2
57     ph(i) = sqrt(2) * sin(pi*((i-1)/(N+2)));
58 end
59 plot(linspace(0, 1, N+2), ph, linspace(0, 1, N+2), Phi(:, 1))
60 %Compute the error terms
61 absolute_error_lambda = abs(la(1)-lambda)
62 relative_error_lambda = abs(la(1)-lambda)/la(1)*100
63 absolute_error_phi = zeros(1, 3);
64 relative_error_phi = zeros(1, 3);
65 for i=1:3
66     absolute_error_phi(i) = max(abs(ph-Phi(:, i)));
67     relative_error_phi(i) = absolute_error_phi(i)/max(abs(Phi(:, i)));
68 end
69 absolute_error_phi
70 relative_error_phi *100
71
72
73
74

```

```

1 %%This class generates a plot of a Brownian sheet%%
2 %%written by Tim Jäschke as a part of his bachelor thesis%%
3
4 %%Used to generate FIGURE 7 %%
5 %%..to generate it, type the following in your MATLAB command:
6 %%Sheet;
7 %%Sheet.plotit();
8 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9
10 classdef Sheet
11     properties (Constant)
12         N = 500;
13         n = 200;
14     end
15     methods (Static)
16         function lam = sqlambda(i,j)
17             lam = 4 / ((2*j - 1) * (2*i - 1) * pi ^2);
18         end
19         function ph = phi(i,j)
20             ph = zeros(Sheet.N, Sheet.N);
21             for k=1:Sheet.N
22                 for l=1:Sheet.N
23                     ph(k,l) = 2*sin((i - 0.5)*pi*k/Sheet.N)*sin((j - 0.5)*pi*l/Sheet.N);
24                 end
25             end
26         end
27         function plotit()
28             BS = zeros(Sheet.N, Sheet.N);
29             BS2 = zeros(Sheet.N, Sheet.N);
30             xi = randn(1, Sheet.n^2);
31             figure
32             for i=1:5
33                 for j=1:5
34                     lam = Sheet.sqlambda(i,j);
35                     phi = Sheet.phi(i,j);
36                     BS2 = BS2 + lam*phi*xi(Sheet.n*(i-1)+j);
37                 end
38             end
39             subplot(3,1,1);
40             surf(linspace(1/Sheet.N, 1, Sheet.N), linspace(1/Sheet.N, 1, Sheet.N), BS2, 'edgealpha', '0');
41             colormap jet

```



```
43         tic;
44         for i=1:Sheet.n
45             for j=1:Sheet.n
46                 lam = Sheet.sqlambda(i,j);
47                 phi = Sheet.phi(i,j);
48                 BS = BS + lam*phi*xi(Sheet.n*(i-1)+j);
49             end
50         i
51     end
52     toc
53     subplot(3,1,[2,3])
54     surf(linspace(1/Sheet.N,1,Sheet.N),linspace(1/Sheet.N,1,Sheet.N),BS,'edgealpha','0');
55     colormap jet
56     end
57 end
58 end
59
```



```

41         for j=1:N
42             for k=1:N
43                 K(j, k) = AK(abs(j - k)+1);
44             end
45         end
46     end
47     function K = AutoCo2(data)
48         [M N] = size(data);
49         K = zeros(N, N);
50         for j = 1:N
51             for k = 1:N
52                 %use symmetry to save operations
53                 if k<j
54                     K(j, k) = K(k, j);
55                 else
56                     for l=1:M
57                         K(j, k) = K(j, k) + data(l, k)*data(l, j);
58                     end
59                     K(j, k)=K(j, k)/M
60                 end
61             end
62         end
63     end
64     function coeff = KLT(K, E)
65         Kernel s;
66         [lambda, Phi] = Kernel s.trapez_Sceme(K);
67         Phi(:, 1)=sqrt(lambda(1))*Phi(:, 1);
68         %for i = 2:5
69         %     Phi(:, 1)=Phi(:, 1)+sqrt(lambda(i))*Phi(:, i);
70         %end
71         N = length(E);
72         A = zeros(1, N);
73         for j=1:N
74             A(j) = Phi(j, 1)+E(j);
75         end
76         coeff = fft(A);
77     end
78     function compare()
79         %build measure values
80         N = 1400;
81         M = 40000;
82         figure
83         %different factor for the noise amplitude
84         for i = 1:4

```

```

85         data = zeros(M,N);
86         if i ==1
87             z=2;
88         elseif i ==2
89             z=4;
90         elseif i == 3
91             z=10;
92         else
93             z=100;
94         end
95         for j = 1:M
96             %generate M times tone + noise
97             tone = Signal.SinTone(300,N);
98             noise = z*randn(1,N);
99             data(j,:) = tone + noise;
100        end
101        %first line is B
102        B = data(1,:);
103        %take the time
104        tic;
105        %build covariance matrix
106        K = Signal.AutoCo(data);
107        %KARHUNEN-LOEVE TRANSFORMATION
108        %KLT returns first Eigenfunction in Fourier base
109        coeff = Signal.KLT(K,zeros(1,N));
110        spectrum_KLT = Signal.Spectrum(coeff);
111        toc
112        tic;
113        %FAST FOURIER TRANSFORM
114        spectrum_FFT = Signal.Spectrum(fft(B));
115        toc
116        subplot(4,2,1+2*(i-1));
117        plot(spectrum_FFT);
118        if i == 1
119            title('SNR=0.5 - FFT');
120        elseif i == 2
121            title('SNR=0.25 - FFT');
122        elseif i == 3
123            title('SNR=0.1 - FFT');
124        elseif i == 4
125            title('SNR=0.01 - FFT');
126        end
127        xlabel('Frequenz in Hz');
128        ylabel('Magnitude');

```

```

129         subplot(4,2,2+2*(i-1));
130         plot(spectrum_KLT);
131         if i == 1
132             title('SNR=0.5 - KLT');
133         elseif i == 2
134             title('SNR=0.25 - KLT');
135         elseif i == 3
136             title('SNR=0.1 - KLT');
137         elseif i == 4
138             title('SNR=0.01 - KLT');
139         end
140         xlabel('Frequenz in Hz');
141         ylabel('Magnitude');
142     end
143 end
144 function compare2()
145     %build measure values
146     N = 1400;
147     M = 300;
148     figure
149     %different factor for the noise amplitude
150     for i = 1:4
151         data = zeros(M,N);
152         if i ==1
153             z=2;
154         elseif i ==2
155             z=4;
156         elseif i == 3
157             z=10;
158         else
159             z=50;
160         end
161         for j = 1:M
162             %generate M times tone + noise
163             tone = Signal.Si nTone(300,N);
164             noise = z*randn(1,N);
165             data(j,:) = tone + noise;
166         end
167         %compute Expectation
168         E = zeros(1,N);
169         for j = 1:N
170             E(j) = sum(data(:,j))/M
171         end
172         for j=1:M

```

```

173         data(j,:) = data(j,:) - E;
174     end
175     %first line is B
176     B = data(1,:);
177     %take the time
178     tic;
179     %build covariance matrix
180     K = Signal.AutoCo2(data);
181     %KARHUNEN-LOEVE TRANSFORMATION
182     %KLT returns first Eigenfunction in Fourier base
183     coeff = Signal.KLT(K, E);
184     spectrum_KLT = Signal.Spectrum(coeff);
185     toc
186     tic;
187     %FAST FOURIER TRANSFORM
188     spectrum_FFT = Signal.Spectrum(fft(B+E));
189     toc
190     subplot(4, 2, 1+2*(i-1));
191     plot(spectrum_FFT);
192     if i == 1
193         title('SNR=0.5 - FFT');
194     elseif i == 2
195         title('SNR=0.25 - FFT');
196     elseif i == 3
197         title('SNR=0.1 - FFT');
198     elseif i == 4
199         title('SNR=0.02 - FFT');
200     end
201     xlabel('Frequenz in Hz');
202     ylabel('Magnitude');
203     subplot(4, 2, 2+2*(i-1));
204     plot(spectrum_KLT);
205     if i == 1
206         title('SNR=0.5 - KLT');
207     elseif i == 2
208         title('SNR=0.25 - KLT');
209     elseif i == 3
210         title('SNR=0.1 - KLT');
211     elseif i == 4
212         title('SNR=0.02 - KLT');
213     end
214     xlabel('Frequenz in Hz');
215     ylabel('Magnitude');
216 end

```

217 end

218 end

219 end

220