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

```
%%This class is a collection of functions related to the Brownian%%
%%motion in the context of Karhunen-Loeve Expansion%%
%%written by Tim Jaschek as a part of%%
%%-a presentation about the Levy construction of Brownian Motion%%
%%-a presentation about the idea of Karhunen-Loeve Expansion%%
%%-a part of his bachelor thesis%%

%%This Programm is used to generate FIGURE 3 in the thesis%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef BrMo
    properties (Constant)
        steps = 1000 ;
        time = linspace(1/BrMo.steps,1,BrMo.steps) ;
    end
    methods (Static)
        function [Z,lambda,psi] = compute_KLT_components(n)
            %this function computes the analytically solved eigenvalues and
            %eigenfunctions of a Brownian motion
            steps = BrMo.steps ;
            Z = randn(1,n) ;
            lambda = zeros(1,n) ;
            psi = zeros(n,steps) ;
            for j = 1:n
                lambda(j) = 1 / ((j-0.5) * pi);
                for k = 1:steps
                    psi(j,k) = sqrt(2)*sin((j-0.5)*pi*k/steps) ;
                end
            end
        end
        function BM = BrownianMotion(n,steps)
            %using the previous function, this function computes the n-th
            %partial sum of the Karhunen-Loeve Expansion of Brownian motion
            BM = zeros(1,steps) ;
            [Z,lambda,psi] = BrMo.compute_KLT_components(n);
            for j = 1:n
                for k = 1:steps
                    BM(k) = BM(k) + Z(j)*lambda(j)*psi(j,k);
                end
            end
```

```
            %OPPORTUNITY to plot:
            %figure
            %plot(BrMo.time,BM)
            %xlabel('time')
            %str=sprintf('Brownian Motion via Karhunen-Loeve Approx. for n = %d',n);
            %title(str)
        end
        function plotDevelop()
            %generates the first plot of FIGURE 3
            BM = zeros(1,BrMo.steps) ;
            [Z,lambda,psi] = BrMo.compute_KLT_components(80);
            figure
            subplot(1,2,1)
            title('Brownian Motion via Karhunen-Loeve Approx. for different n')
            hold on;
            for j = 1:80
                for k = 1:BrMo.steps
                    BM(k) = BM(k) + Z(j)*lambda(j)*psi(j,k);
                end
                if j==4 || j==16 || j==32 || j==64
                    plot(BrMo.time,BM)
                end
            end
            legend('n = 4','n = 16', 'n = 32','n = 64')
            hold off;
        end
        function plotMovie()
            %an exciting movie that shows the behaviour of the partial sums
            %of the Karhunen-Loeve Expansion is generated here
            BM = zeros(1,BrMo.steps) ;
            [Z,lambda,psi] = BrMo.compute_KLT_components(500);
            for j = 1:500
                for k = 1:BrMo.steps
                    BM(k) = BM(k) + Z(j)*lambda(j)*psi(j,k);
                end
                plot(BrMo.time,BM)
                title('Animation of KLT Series');
                pause(0.01)
            end
        end
        function LevyBase()
```

```
            %plots the Schauder base
            Bn = [0,1] ;
            List = [0,1] ;
            figure
            title('Schauder Base - Base from the Construction by L·vy');
            xlabel('time');
            hold on;
            plot(List,Bn);
            for n = 1:4
                if n==1
                    Bn = [Bn, 0];
                    List = linspace(0,1,3);
                else
                    Bn = [Bn, 1 , 0];
                    List = linspace(0,1,length(Bn));
                end
                plot(List,Bn);
            end
            hold off;
        end
        function KLTBase()
            %plots the Karhunen-Loeve Eigenfunction base
            [Z,lambda,psi] = BrMo.compute_KLT_components(6);
            figure
            title('Fourier Base - Base from Karhunen-Lo·ve Theorem');
            xlabel('time');
            hold on;
            for j = 1:6
                plot(BrMo.time,psi(j,:))
            end
            hold off;
        end
        function levy()
            %generates the second plot of FIGURE 3
            D_0 = linspace(0,1,2^0 + 1) ; %intervalls of dyadic points
            B_0 = [ 0 , randn]; %approximation of Brownian Motin in step 0
            subplot(1,2,2);
            title('Brownian Motion via Levy Construction for different n')
            hold on;
```

```
        for n=1:6
            D_next = linspace(0,1,2^n+1);
            B_next = zeros(1,length(D_next));
            j=1;
            for i=1:2^n+1
                if mod(i,2)== 0;
                    B_next(i)= sqrt(1/(2^(n+1)))*randn;
                else
                    B_next(i)= B_0(j);
                    j=j+1;
                end
            end
            for i=1:2^n+1
                if mod(i,2)== 0;
                    B_next(i)=B_next(i)+(B_0(i/2)+B_0(i/2+1))/2;

                end
            end
            D_0 = D_next;
            B_0 = B_next;
            if n==2 || n==4 || n==5 || n==6
                plot(D_0,B_0)
            end
        end
        legend('n = 4','n = 16', 'n = 32','n = 64')
        hold off;
    end
    function CoV()
        %plots the covariancefunction of Brownian motion
        [X,Y] = meshgrid(0:.05:1);
        Z = min(X,Y);
        surf(X,Y,Z)
        title('Kovarianzfunktion der Brownschen Bewegung')
        xlable('time')
    end
    function KLTlambda()
        %plots the Eigenvalues
        [Z,lambda,psi] = BrMo.compute_KLT_components(10);
        plot(linspace(1,10,10),lambda,'*')
        title('KLT Coefficients')
        xlabel('index')
        ylabel('lambda_i')
    end
    function CoVBB()
```

```
            %plots the covariancefunction of Brownian bridge
            [X,Y] = meshgrid(0:.05:1);
            Z = min(X,Y) - X.*Y
            surf(X,Y,Z)
            title('Kovarianzfunktion der Brownschen Bræcke')
        end
        function Mercer()
            %plots different approximations of the covariancefunction of
            %Brownian motion using Mercers theorem
            [Z,lambda,psi] = BrMb.compute_KLT_components(10);
            figure
            A = zeros(20,20);
            title('Mercers Theorem - covariance function of Brownian Motion');
            hold on;
            for n = 1:4
                for i = 1 : 20
                    for j = 1:20
                        A(i,j) = A(i,j) + lambda(n)*psi(n,50*i)*psi(n,50*j);
                    end
                end
            end
            [X,Y] = meshgrid(0.05:0.05:1);
            surf(X,Y,A)
        end
        function value = mother(t)
            %%%Haar mother function%%%%%
            if ((t<0.5) && (t >= 0))
                value = 1;
            elseif ((t>=0.5) && (t<1))
                value = -1;
            else
                value = 0;
            end
        end
        function psi = Haar(N)
            steps = linspace(1/N,1,N);
            %%Use the Haar Mother function%%%%
            psi(:,1) = ones(1,N);
            psi(N,1) = 0;
            n=0;
            k=0;
```

```
            count =2;
            while count <N+1
                max = 2^n;
                if k == max
                    n=n+1;
                    k=0;
                    continue
                end
                for t=1:N
                     psi(t,count) = 2^(n/2)*BrMb.mother(2^n*steps(t)-k);
                end
                count = count+1;
                k=k+1;
            end
        end
    end
end
```

```
%%This Programm generates a Plot eigenvalues and eigenfunctions of
%%the induced integral operators of different kernels
%%written by Tim Jaschek as a part of his bachelor thesis%%

%%This Programm is used to generate FIGURE 2 in the thesis%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%load the class Kernels
Kernels;

%Parameter for accuracy
N=50;

figure
for i=1:3;
    Mat = Kernels.KMat(i,N);
    [lambda,Phi] = Kernels.trapez_Sceme(Mat);
    subplot(3,3,[1+3*(i-1) 2+3*(i-1)]);
    for j=1:6
        hold on;
        plot(linspace(0,1,N+2),Phi(:,j));
    end
    if i ==1
        title('First 6 Eigenfunctions');
        ylabel('K(s,t)=min(s,t)');
    elseif i == 2
        ylabel('K(s,t)=min(s,t) - st');
    else
        ylabel('K(s,t)=exp(-|s-t|)');
    end
    hold off;
    subplot(3,3,3*i);
    plot(linspace(1,10,10),lambda(1:10),'o','color','red');
    if i ==1
        title('First 10 Eigenvalues');
    end
end
```

```
%%This class gives tools for image compressing via Principal%%
%%Component Analysis (KLT)%%
%%written by Tim Jaschek as a part of his bachelor thesis%%

%%Used to generate FIGURE 6 %%
%%...to generate it, type the following in your MATLAB command:
%%Image;
%%Image.program();
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef Image
    properties (Constant)
    end
    methods (Static)
        function X = load_space()
            %load an image
            X = imread('bluemarbel.jpg');
            %show the image
            figure
            image(X)
        end
        function B = blurBW(X)
            %function to blur an B/W image
            [m,n] = size(X);
            X = double(X);
            %%to 25%
            mm=m/2;
            nn = n/2;
            B=zeros(mm,nn);
            for i=1:mm
                for j=1:nn
                    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));
                end
            end
            B = uint8(B);
        end
        function Z = blur(X)
            %function to blur an R/G/B image
            XR = X(:,:,1);
            XG = X(:,:,2);
            XB = X(:,:,3);
            RN = Image.blurBW(XR);
            GN = Image.blurBW(XG);
```

```
            BN = Image.blurBW(XB);
            Z = cat(3, RN, GN, BN);
        end
        function X_trans = PCA(X,k)
            %function for image compression via KLT of a B/W image
            [m,n]=size(X);
            X = double(X);
            X_hat = X;
            mean = zeros(1,n);
            K = zeros(n,n);
            %%correct mean%%%
            for i=1:n
                mean(i) = sum(X(:,i))/m;
                X_hat(:,i) = X_hat(:,i)-mean(i);
            end
            %%covariance matrix%%%
            for i=1:n
                for j=1:n
                    K(i,j) = (1/(n-1))*dot((X_hat(:,i)),(X_hat(:,j)));
                end
            end
            %%Eigenvalues and Eigenvectors%%%
            [V,D]=eig(K);
            [lambda,ind] = sort(diag(D),'descend');
            Phi = V(:,ind);
            Phi = Phi(:,1:k);
            PhiT = Phi.';
            %%%Transform X %%%
            Y = PhiT*(X_hat);
            X_trans = Phi*Y;
            for i=1:n
                X_trans(:,i) = X_trans(:,i)+mean(i);
            end
            X_trans = uint8(X_trans);
        end
        function Z = PCA_RGB(X,k);
            %function for image compressing via KLT of an R/G/B image
            XR = X(:,:,1);
            XG = X(:,:,2);
            XB = X(:,:,3);
            RN = Image.PCA(XR,k);
            GN = Image.PCA(XG,k);
```

```
                BN = Image.PCA(XB,k);
                Z = cat(3, RN, GN, BN);
            end
            function program()
                %generates FIGURE 6
                disp('loading image')
                X = Image.load_space();
                figure
                set(gca, 'XTickLabel', [],'XTick',[])
                h = subplot(2,2,1)
                imshow(X)
                disp('data compressing...')
                Z = Image.PCA_RGB(X,400);
                hh = subplot(2,2,2)
                imshow(Z)
                Z = Image.PCA_RGB(X,200);
                hhh = subplot(2,2,3)
                imshow(Z)
                Z = Image.PCA_RGB(X,100);
                hhhh = subplot(2,2,4)
                imshow(Z)
                p = get(h, 'pos');
                pp = get(hh, 'pos');
                ppp = get(hhh, 'pos');
                pppp = get(hhhh, 'pos');
                p([3,4]) = p([3,4]) + [0.1 0.1];
                set(h, 'pos', p);
                pp([3,4]) = pp([3,4]) + [0.1 0.1];
                set(hh, 'pos', pp);
                ppp([3,4]) = ppp([3,4]) + [0.1 0.1];
                set(hhh, 'pos', ppp);
                pppp([3,4]) = pppp([3,4]) + [0.1 0.1];
                set(hhhh, 'pos', pppp);
            end
        end
    end
```

```matlab
%%This class is a collection of different Kernels and integration methods%%
%%to solve the Fredholm integral equation. It also provides a%%
%%Merverapproximation for the kernels. %%
%%written by Tim Jaschek as a part of his bachelor thesis%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
classdef Kernels
    properties (Constant)
    end
    methods (Static)
        function K_st = Kernel(i,s,t)
            %this is a collection of covariance functions%
            if i == 1
                %Brownian Motion%
                K_st = min(s,t);
            elseif i==2
                %Brownian Bridge%
                K_st = min(s,t) - s*t;
            elseif i==3
                %exponential kernel%
                K_st = exp(-abs(t-s));
            else
                K_st = 0;
            end
        end
        function Mat = KMat(i,N)
             %for given N and i, this function will generate an NxN matrix
             %for the i-th kernel.
             Mat = zeros(N+2,N+2);
             for j = 1:N+2
                 for k = 1:N+2
                     %use symmetry to save operations
                     if k<j
                         Mat(j,k) = Mat(k,j);
                     else
                         Mat(j,k) = Kernels.Kernel(i,(j-1)/(N+1),(k-1)/(N+1));
                     end
                 end
             end
        end
        function [lambda,Phi] = uniform_Sceme(K)
```

```
            %UNIFORM SCEME
            N = length(K)-2;
            sqW = sqrt(1/(N+2))* eye(N+2);
            Mat = sqW*K*sqW;
            [V,D]=eig(Mat);
            [lambda,ind] = sort(diag(D),'descend');
            E_vectors = V(:,ind);
            Phi = sqrt(N+2)*E_vectors;
            %%bring the EV in the right direction%%
            for i = 1:N+2
                if Phi(2,i)<0
                    Phi(:,i)=-Phi(:,i);
                end
            end
        end
        function [lambda,Phi] = trapez_Sceme(K)
            %TRAPEZ SCEME
            N = length(K)-2;
            sqW = sqrt(1/(N+1))* eye(N+2);
            sqW(1,1) = sqrt(1/(2*(N+1)));
            sqW(N+2,N+2) = sqW(1,1);
            qW = sqrt(N+2)*eye(N+2);
            qW(1,1) = sqrt(2*(N+1));
            qW(N+2,N+2) = qW(1,1);
            Mat = sqW*K*sqW;
            [V,D]=eig(Mat);
            [lambda,ind] = sort(diag(D),'descend');
            E_vectors = V(:,ind);
            Phi = qW*E_vectors;
            %%bring the EV in the right direction%%
            for i = 1:N+2
                if Phi(2,i)<0
                    Phi(:,i)=-Phi(:,i);
                end
            end
        end
        function [lambda,Phi] = simpson_Sceme(K)
            %SIMPSON SCEME
            N = length(K)-2;
            sqW = sqrt(1/(3*(N+1)))* eye(N+2);
            qW = sqrt(3*(N+1))*eye(N+2);
            for i = 2:N+1
                sqW(i,i)=sqW(i,i)*sqrt(2);
                qW(i,i)=qW(i,i)/sqrt(2);
```

```
           end
           for i = 2:(N+1)/2+1
               j=2*(i-1);
               sqW(j,j)=sqW(j,j)*sqrt(2);
               qW(j,j)=qW(j,j)/sqrt(2);
           end
           Mat = sqW*K*sqW;
           [V,D]=eig(Mat);
           [lambda,ind] = sort(diag(D),'descend');
           E_vectors = V(:,ind);
           Phi = qW*E_vectors;
           %%bring the EV in the right direction%%
           for i = 1:N+2
                if Phi(2,i)<0
                    Phi(:,i)=-Phi(:,i);
                end
           end
       end
       function K = MercerApprox(lambda,Phi,n)
           %INPUT: lambda - Eigenvalues,
           %       Phi - Eigenfunctions,
           %       n - summations

           %OUTPUT: K as approximation of covariance matrix
           N = length(lambda)-2;
           K=zeros(N+2,N+2);
           for s=1:N+2
               for t=1:N+2
                    if t<s
                        K(s,t) = K(t,s);
                    else
                       for i=1:n
                           K(s,t) = K(s,t) + lambda(i)*Phi(s,i)*Phi(t,i);
                       end
                    end
               end
           end
       end

   end
end
```

```matlab
%%This Programm generates a Plot for different Kernels and partial%%
%%sums of Mercer's series for the covariance function%%
%%written by Tim Jaschek as a part of his bachelor thesis%%

%%This Programm is used to generate FIGURE 1 in the thesis%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%load the class Kernels
Kernels;

%Parameter for accuracy
N=16;

figure
for i=1:3;
    Mat = Kernels.KMat(i,N);
    [lambda,Phi] = Kernels.trapez_Sceme(Mat);
    for j=1:3;
        K=Kernels.MercerApprox(lambda,Phi,j+(j-1)^2);
        subplot(4,3,i+3*(j-1));
        surfc(linspace(0,1,N+2),linspace(0,1,N+2),K,'edgealpha','1');
        if j == 1
            if i ==1
                title('K(s,t)=min(s,t)');
                zlabel('n=1');
            elseif i ==2
                title('K(s,t)=min(s,t) - st');
            else
                title('K(s,t)=exp(-|s-t|)');
            end
        elseif j== 2
            if i == 1
                zlabel('n=3');
            end
        else
            if i == 1
                zlabel('n=7');
            end
        end
    end
    subplot(4,3,i+9);
    surfc(linspace(0,1,N+2),linspace(0,1,N+2),Mat,'edgealpha','1');
```

```
    if i == 1
        zlabel('analytic');
    end
end
```

```matlab
%%This class is a collection of functions to expand a Brownian motion wrt.
%%to different orthonormal bases and compute the Total Mean Squared Error
%%for the approximation with the n-th partial sum
%%written by Tim Jaschek as a part of his bachelor thesis%%

%%Used to generate FIGURE 3 and the data for TABULAR 5.1 in this thesis %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef MSE
    properties (Constant)
        %Parameter for the partial sum
        n = 20;

        %Parameter for the accuracy of the Kernels
        N = 1000;
    end
    methods (Static)
        function approximation()
            %load the classes BrMo and Kernels
            BrMo;
            Kernels;
            %%%generate a Brownian Motion with N steps%%%%
            p = 10000; %parameter for the accuracy of the BM
            X = BrMo.BrownianMotion(p,MSE.N);
            %%%get ORTHONORMAL BASES
            %%%the analytic KLT Eigenfunctions
            %%%the HaarWavelets
            %%%Eigenfunctions of Brownian Bridge
            %%%Eigenfunctions of Exponential Kernel
            [Z,lambda,psi] = BrMo.compute_KLT_components(MSE.N);
            psi = psi.';
            Haar = BrMo.Haar(1000);
            [lambda,Bridge] = Kernels.trapez_Sceme(Kernels.KMat(2,MSE.N));
            [lambda,Exp] = Kernels.trapez_Sceme(Kernels.KMat(3,MSE.N));
            %%%Compute the Approximations%%%%
            X_hat = MSE.Approx(X,psi(:,1:MSE.n));
            X_tilde = MSE.Approx(X,Haar(:,1:MSE.n));
            X_bridge = MSE.Approx(X,Bridge(:,1:MSE.n));
            X_exp = MSE.Approx(X,Exp(:,1:MSE.n));
```

```
        %%%plot both%%%
        figure
        hold on;
        plot(linspace(1/MSE.N,1,MSE.N),X)
        plot(linspace(1/MSE.N,1,MSE.N),X_hat)
        plot(linspace(1/MSE.N,1,MSE.N),X_tilde)
        plot(linspace(1/MSE.N,1,MSE.N),X_bridge)
        %plot(linspace(1/MSE.N,1,MSE.N),X_exp)
        hold off;
    end
    function meansquare()
        %load the classes BrMo and Kernels
        BrMo;
        Kernels;

        disp('Compute TMSE of Karhunen-Loeve-Base...')
        [Z,lambda,psi] = BrMo.compute_KLT_components(MSE.N);
        psi = psi.';
        Mse = zeros(1,MSE.N);
        for i=1:30
            X = BrMo.BrownianMotion(5000,MSE.N);
            X_hat = MSE.Approx(X,psi(:,1:MSE.n));
            for j=1:MSE.N
                Mse(j) = Mse(j) + (X(j)-X_hat(j))^2;
            end
        end
        Mse = Mse/30;
        TMse = 0;
        h=1/MSE.N;
        for j=1:MSE.N-1
                TMse = TMse + h/2 * (Mse(j) + Mse(j+1));
        end
        TMse
        %%%%%%%%%%Haar%%%%%%
        disp('Compute TMSE of Haar-Base...')
        Haar = BrMo.Haar(1000);
        Mse = zeros(1,MSE.N);
        for i=1:30
            X = BrMo.BrownianMotion(5000,MSE.N);
            X_hat = MSE.Approx(X,Haar(:,1:MSE.n));
            for j=1:MSE.N
                Mse(j) = Mse(j) + (X(j)-X_hat(j))^2;
            end
        end
```

```matlab
            Mse = Mse/30;
            TMse = 0;
            h=1/MSE.N;
            for j=1:MSE.N-1
                    TMse = TMse + h/2 * (Mse(j) + Mse(j+1));
            end
            TMse
            %%%%%%%%%Bridge%%%%%%
            disp('Compute TMSE of Brownian-Bridge-Base...')
            [lambda,Bridge] = Kernels.trapez_Sceme(Kernels.KMat(2,MSE.N));
            Mse = zeros(1,MSE.N);
            for i=1:30
                X = BrMo.BrownianMotion(5000,MSE.N);
                X_hat = MSE.Approx(X,Bridge(:,1:MSE.n));
                for j=1:MSE.N
                    Mse(j) = Mse(j) + (X(j)-X_hat(j))^2;
                end
            end
            Mse = Mse/30;
            TMse = 0;
            h=1/MSE.N;
            for j=1:MSE.N-1
                    TMse = TMse + h/2 * (Mse(j) + Mse(j+1));
            end
            TMse
            %%%%%%%%%Exponential%%%%%%
            disp('Compute TMSE of Exponential-Base...')
            [lambda,Exp] = Kernels.trapez_Sceme(Kernels.KMat(3,MSE.N));
            Mse = zeros(1,MSE.N);
            for i=1:30
                X = BrMo.BrownianMotion(5000,MSE.N);
                X_hat = MSE.Approx(X,Exp(:,1:MSE.n));
                for j=1:MSE.N
                    Mse(j) = Mse(j) + (X(j)-X_hat(j))^2;
                end
            end
            Mse = Mse/30;
            TMse = 0;
            h=1/MSE.N;
            for j=1:MSE.N-1
                    TMse = TMse + h/2 * (Mse(j) + Mse(j+1));
            end
```

```
            TMse
        end
        function X_hat = Approx(X,Phi);
            %%assume X has N steps on [0,1] and we have at least n Eigenvectors with N steps each%%%
            X_hat = zeros(1,MSE.N);
            A = zeros(1,MSE.n);
            h = 1/MSE.N;
            for i=1:MSE.n
                %%%compute the integrals via trapez sceme%%%
                for j=1:MSE.N-1
                    A(i) = A(i) + h/2 * (X(j)*Phi(j,i) + X(j+1)*Phi(j+1,i));
                end
            end
            for i=1:MSE.N
                X_hat(i) = dot(A,Phi(i,1:MSE.n));
            end
        end
    end
end
```

```
%%This programm compares uniform, trapez and Simpson-Sceme for %%
%%approximation of solutions to the Fredholm integral equation.%%
%%written by Tim Jaschek as a part of his bachelor thesis%%

%%Used to generate data for Tabular 6.1 and 6.2 %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Import the class Kernels which contains some Kernels and
%integration scemes.
Kernels;

%Parameter for the Number of approximation steps
N = 45;

%Generation of different Kernels
BrownianMotion = Kernels.KMat(1,N);
BrownianBridge = Kernels.KMat(2,N);
ExponentialKer = Kernels.KMat(3,N);

%BROWNIAN MOTION
%Solve Fredhol integralequality with different Scemes
[lambda1,Phi1] = Kernels.uniform_Sceme(BrownianMotion);
[lambda2,Phi2] = Kernels.trapez_Sceme(BrownianMotion);
[lambda3,Phi3] = Kernels.simpson_Sceme(BrownianMotion);
%Compute analytic solutions for first Eigenvalues
lambda = [lambda1(1) lambda2(1) lambda3(1)];
Phi  = [Phi1(:,1) Phi2(:,1) Phi3(:,1)];
la = (2/pi)^2;
ph = zeros(N+2,1);
for i=1:N+2
   ph(i) = sqrt(2)* sin(0.5*pi*((i-1)/(N+2)));
end
%plot(linspace(0,1,N+2),ph,linspace(0,1,N+2),Phi(:,1))
%Compute the error terms
absolute_error_lambda = abs(la(1)-lambda)
relative_error_lambda = abs(la(1)-lambda)/la(1)*100
absolute_error_phi  = zeros(1,3);
relative_error_phi  = zeros(1,3);
for i=1:3
     absolute_error_phi(i) = max(abs(ph-Phi(:,i)));
     relative_error_phi(i) = absolute_error_phi(i)/max(abs(Phi(:,i)));
end
absolute_error_phi
```

```
relative_error_phi*100

%BROWNIAN BRIDGE
%Solve Fredhol integralequality with different Scemes
[lambda1,Phi1] = Kernels.uniform_Sceme(BrownianBridge);
[lambda2,Phi2] = Kernels.trapez_Sceme(BrownianBridge);
[lambda3,Phi3] = Kernels.simpson_Sceme(BrownianBridge);
%Compute analytic solutions for first Eigenvalues
lambda = [lambda1(1) lambda2(1) lambda3(1)];
Phi = [Phi1(:,1) Phi2(:,1) Phi3(:,1)];
la = (1/pi)^2;
ph = zeros(N+2,1);
for i=1:N+2
    ph(i) = sqrt(2)* sin(pi*((i-1)/(N+2)));
end
plot(linspace(0,1,N+2),ph,linspace(0,1,N+2),Phi(:,1))
%Compute the error terms
absolute_error_lambda = abs(la(1)-lambda)
relative_error_lambda = abs(la(1)-lambda)/la(1)*100
absolute_error_phi = zeros(1,3);
relative_error_phi = zeros(1,3);
for i=1:3
      absolute_error_phi(i) = max(abs(ph-Phi(:,i)));
      relative_error_phi(i) = absolute_error_phi(i)/max(abs(Phi(:,i)));
end
absolute_error_phi
relative_error_phi*100
```

```matlab
%%This class generates a plot of a Brownian sheet%%
%%written by Tim Jaschek as a part of his bachelor thesis%%

%%Used to generate FIGURE 7 %%
%%...to generate it, type the following in your MATLAB command:
%%Sheet;
%%Sheet.plotit();
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef Sheet
    properties (Constant)
        N = 500;
        n = 200;
    end
    methods (Static)
        function lam = sqlambda(i,j)
            lam = 4 / ((2*j-1)*(2*i-1) * pi^2);
        end
        function ph = phi(i,j)
            ph = zeros(Sheet.N,Sheet.N);
            for k=1:Sheet.N
                for l=1:Sheet.N
                    ph(k,l) = 2*sin((i-0.5)*pi*k/Sheet.N)*sin((j-0.5)*pi*l/Sheet.N);
                end
            end
        end
        function plotit()
            BS = zeros(Sheet.N,Sheet.N);
            BS2 = zeros(Sheet.N,Sheet.N);
            xi = randn(1,Sheet.n^2);
            figure
            for i=1:5
                for j=1:5
                    lam = Sheet.sqlambda(i,j);
                    phi = Sheet.phi(i,j);
                    BS2 = BS2 + lam*phi*xi(Sheet.n*(i-1)+j);
                end
                i
            end
            subplot(3,1,1);
            surf(linspace(1/Sheet.N,1,Sheet.N),linspace(1/Sheet.N,1,Sheet.N),BS2,'edgealpha','0');
            colormap jet
```

```
            tic;
            for i=1:Sheet.n
                for j=1:Sheet.n
                    lam = Sheet.sqlambda(i,j);
                    phi  = Sheet.phi(i,j);
                    BS = BS + lam*phi*xi(Sheet.n*(i-1)+j);
                end
                i
            end
            toc
            subplot(3,1,[2,3])
            surf(linspace(1/Sheet.N,1,Sheet.N),linspace(1/Sheet.N,1,Sheet.N),BS,'edgealpha','0');
            colormap jet
        end
    end
end
```

```matlab
%%This class is a collection of functions related to signal detection via
%%Karhunen Loeve Transformation
%%written by Tim Jaschek as a part of his bachelor thesis%%

%%Used to generate FIGURE 5 %%
%%...to generate it, type the following in your MATLAB command:
%%Signal;
%%Signal.compare2();
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef Signal
    properties (Constant)
    end
    methods (Static)
        function tone = SinTone(toneFreq,sampleFreq)
            %build sine tone
            t = (1:sampleFreq) / sampleFreq;            % build time steps of length 1 second
            tone = sin(2 * pi * toneFreq * t);          % sinusoidal modulation
        end
        function playTone(tone,sampleFreq)
            %play tone
            sound(tone, sampleFreq);     % sound function from Matlab
            pause(1.5);                  % wait
        end
        function spect = Spectrum(coeff)
            spect = abs(coeff);
            %know spectrum is two sided. Make it one sided:
            spect = spect(1:length(coeff)/2+1);
            spect(2:end-1) = 2*spect(2:end-1);
        end
        function K = AutoCo(data)
            [M,N] = size(data);
            K = zeros(N,N);
            AK = zeros(N);
            for j=1:N
                for l=1:M
                    AK(j) =  AK(j) + data(l,1)*data(l,j);
                end
                AK(j)=AK(j)/M;
            end
```

```
        for j=1:N
            for k=1:N
                K(j,k) = AK(abs(j-k)+1);
            end
        end
    end
    function K = AutoCo2(data)
       [M,N] = size(data);
       K = zeros(N,N);
       for j = 1:N
            for k = 1:N
                %use symmetry to save operations
                if k<j
                    K(j,k) = K(k,j);
                else
                    for l=1:M
                        K(j,k)= K(j,k) + data(l,k)*data(l,j);
                    end
                    K(j,k)=K(j,k)/M;
                end
            end
       end
    end
    function coeff = KLT(K,E)
        Kernels;
        [lambda,Phi] = Kernels.trapez_Sceme(K);
        Phi(:,1)=sqrt(lambda(1))*Phi(:,1);
        %for i = 2:5
        %    Phi(:,1)=Phi(:,1)+sqrt(lambda(i))*Phi(:,i);
        %end
        N = length(E);
        A = zeros(1,N);
        for j=1:N
            A(j) = Phi(j,1)+E(j);
        end
        coeff = fft(A);
    end
    function compare()
        %build measure values
        N = 1400;
        M = 40000;
        figure
        %different factor for the noise amplitude
        for i = 1:4
```

```
data = zeros(M,N);
if i ==1
    z=2;
elseif i ==2
    z=4;
elseif i == 3
    z=10;
else
    z=100;
end
for j = 1:M
    %generate M times tone + noise
    tone = Signal.SinTone(300,N);
    noise = z*randn(1,N);
    data(j,:) = tone + noise;
end
%first line is B
B = data(1,:);
%take the time
tic;
%build covariance matrix
K = Signal.AutoCo(data);
%KARHUNEN-LOEVE TRANSFORMATION
%KLT returns first Eigenfunction in Fourier base
coeff = Signal.KLT(K,zeros(1,N));
spectrum_KLT = Signal.Spectrum(coeff);
toc
tic;
%FAST FOURIER TRANSFORM
spectrum_FFT = Signal.Spectrum(fft(B));
toc
subplot(4,2,1+2*(i-1));
plot(spectrum_FFT);
if i == 1
    title('SNR=0.5 - FFT');
elseif i == 2
    title('SNR=0.25 - FFT');
elseif i == 3
    title('SNR=0.1 - FFT');
elseif i == 4
    title('SNR=0.01 - FFT');
end
xlabel('Frequenz in Hz');
ylabel('Magnitude');
```

```
                subplot(4,2,2+2*(i-1));
                plot(spectrum_KLT);
                if i == 1
                    title('SNR=0.5 - KLT');
                elseif i == 2
                    title('SNR=0.25 - KLT');
                elseif i == 3
                    title('SNR=0.1 - KLT');
                elseif i == 4
                    title('SNR=0.01 - KLT');
                end
                xlabel('Frequenz in Hz');
                ylabel('Magnitude');
            end
        end
        function compare2()
            %build measure values
            N = 1400;
            M = 300;
            figure
            %different factor for the noise amplitude
            for i = 1:4
                data = zeros(M,N);
                if i ==1
                    z=2;
                elseif i ==2
                    z=4;
                elseif i == 3
                    z=10;
                else
                    z=50;
                end
                for j = 1:M
                    %generate M times tone + noise
                    tone = Signal.SinTone(300,N);
                    noise = z*randn(1,N);
                    data(j,:) = tone + noise;
                end
                %compute Expectation
                E = zeros(1,N);
                for j = 1:N
                    E(j) = sum(data(:,j))/M;
                end
                for j=1:M
```

```
            data(j,:)=data(j,:)-E;
        end
        %first line is B
        B = data(1,:);
        %take the time
        tic;
        %build covariance matrix
        K = Signal.AutoCo2(data);
        %KARHUNEN-LOEVE TRANSFORMATION
        %KLT returns first Eigenfunction in Fourier base
        coeff = Signal.KLT(K,E);
        spectrum_KLT = Signal.Spectrum(coeff);
        toc
        tic;
        %FAST FOURIER TRANSFORM
        spectrum_FFT = Signal.Spectrum(fft(B+E));
        toc
        subplot(4,2,1+2*(i-1));
        plot(spectrum_FFT);
        if i == 1
            title('SNR=0.5 - FFT');
        elseif i == 2
            title('SNR=0.25 - FFT');
        elseif i == 3
            title('SNR=0.1 - FFT');
        elseif i == 4
            title('SNR=0.02 - FFT');
        end
        xlabel('Frequenz in Hz');
        ylabel('Magnitude');
        subplot(4,2,2+2*(i-1));
        plot(spectrum_KLT);
        if i == 1
            title('SNR=0.5 - KLT');
        elseif i == 2
            title('SNR=0.25 - KLT');
        elseif i == 3
            title('SNR=0.1 - KLT');
        elseif i == 4
            title('SNR=0.02 - KLT');
        end
        xlabel('Frequenz in Hz');
        ylabel('Magnitude');
    end
```

```
            end
    end
end

```