```
%%This class is a collection of different Kernels and integration methods%%
%%to solve the Fredholm integralequation. 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
```