```
%Priyanshi parauha
%Sampling and recovery of sparse signals using MLE


function main

rng('default');

experiment=1;   %1: 2 Diracs. 2: 6 Diracs. 3: 50 Diracs.
tau=1;          %size of the interval where the pulses lie. Must be positive
if experiment==1
    rng(0);     %for reproducibility of the results in the article
    K=2;        %number of Dirac pulses. Must be >=1
    N=11;       %number of measurements. Must be odd
    tk=[0.42;0.52];     %positions of the Dirac pulses. Must be in [0,tau[
    ak=[1;1];           %amplitudes of the Dirac pulses. Must be nonzero
    SNR=15;     %signal-to-noise ratio in dB
elseif experiment==2
    rng(10);
    K=6;        %use K=10 to see the effects of overfitting
    N=25;
    tk=[0.16;0.24;0.53;0.62;0.73;0.76];
    ak=[0.2;-0.3;0.1;0.7;-0.25;0.75];
    SNR=25;
elseif experiment==3
    rng(10);
    K=50;       %use K=30 to see the effects of underfitting
    N=1001;
    tk=rand(50,1);
    ak=rand(50,1)*2-1;
    SNR=35;
else
end

method=2;           % reconstruction method.
                    % 1:Cadzow denoising,
                    % 2:proposed method,
Nbnoiserea=1;       %number of noise realizations. The plot is different whether =1 ↙
or >1
displayresults=1;   %1 or 0: display or not the numerical results
displayfigure=1;    %1 or 0: display or not the figure


%we generate directly the Fourier coefficients of the signal
M=(N-1)/2;
vhatm0=exp(-i*2*pi/tau*(-M:M)'*tk')*ak;

if displayfigure
    figure
```

```
    vec=ak(1)*(1+2*sum(cos(2*pi*(1:M)'*((0:5000)/5000-tk(1)/tau)),1))/N;
    for k=2:length(tk), vec=vec+ak(k)*(1+2*sum(cos(2*pi*(1:M)'*((0:5000)/5000-tk(k) ↙
/tau)),1))/N; end
    plot((0:5000)/5000, vec,'color',[0.8,0.8,0.8]);
    hold on
    h1=stem(tk/tau,ak,'Color','k','MarkerEdgeColor','k','Markersize',7);
    xlim([0 1]);
    ymin=min(0,min(ak));
    ymax=max(0,max(ak));
    ylim([ymin-0.1 ymax+0.1]);
    set(get(h1,'BaseLine'),'LineStyle',':');
    set(gca,'FontSize',13);
    xlabel(['position [units of ' texlabel('tau') ' ]'],'fontsize',13);
    ylabel('amplitude','fontsize',13)
end
if Nbnoiserea>1
    thewaitbar = waitbar(0,'number of noise realizations simulated');
end

error1=0; error2=0;
tic
for noiserea=1:Nbnoiserea

    %we generate the noise and scale it so as to exactly have the desired SNR
    epshatm=fftshift(fft(randn(N,1)));
    vhatm=vhatm0+epshatm/norm(epshatm)*norm(vhatm0)/10^(SNR/20);
    vn=real(ifft(ifftshift(vhatm)));
    if displayfigure
        plot((0:N-1)/N,vn,'o','MarkerFaceColor',[0.8,0.8,0.8],'MarkerEdgeColor',↙
[0.8,0.8,0.8],'Markersize',4);
    end

    %%% Cadzow denoising method
    if method==1
        P=M;                %the matrices have size N-P x P+1. K<=P<=M required.
        Nbiter=50;          %number of iterations.
        Tdenoised=toeplitz(vhatm(P+1:N),vhatm(P+1:-1:1)); %the noisy matrix is the↙
initial estimate
        for iter=1:Nbiter
            [U S V]=svd(Tdenoised,0);
            Tdenoised=U(:,1:K)*S(1:K,1:K)*(V(:,1:K))';  %SVD truncation -> Tdenoised↙
has rank K
            Tdenoised=Toeplitzation(Tdenoised);         % -> Tdenoised is Toeplitz
        end
        %we reshape the Toeplitz matrix Tdenoised into a Toeplitz matrix with K+1 ↙
columns
        Tdenoised=toeplitz([Tdenoised(1,P-K+1:-1:1).';Tdenoised(2:N-P,1)],Tdenoised ↙
(1,P-K+1:P+1));
        [U S V]=svd(Tdenoised,0);
```

```
        estimtk=-angle(roots(V(:,K+1)))*tau/2/pi;    %estimated locations in [-tau/2,
tau/2[
        edgecolor='b';


    %%% Proposed method
    elseif method==2
        P=M;                %the matrices have size N-P x P+1. K<=P<=M required.
        Nbiter=50;          %number of iterations.
        mu=0.1;             %parameter. Must be in ]0,2[
        gamma=0.51*mu;      %parameter. Must be in ]mu/2,1[
        %note: setting mu=0 and gamma=0 yields the Douglas-Rachford method
        Tnoisy=toeplitz(vhatm(P+1:N),vhatm(P+1:-1:1));
        W=ones(N-P,P+1)*(P+1);   %matrix of weights for the weighted Frobenius norm
        for indexcol=2:P+1
            for indexrow=1:P+2-indexcol
                W(indexrow,indexcol-1+indexrow)=P+2-indexcol;
                W(N-P-indexrow+1,P+3-indexcol-indexrow)=P+2-indexcol;
            end
        end
        Tdenoised=Tnoisy;        %the noisy matrix is the initial estimate
        mats=Tdenoised;          %auxiliary matrix
        for iter=1:Nbiter
            [U S V]=svd(mats+gamma*(Tdenoised-mats)+mu*(Tnoisy-Tdenoised)./W,0);
            Tdenoised=U(:,1:K)*S(1:K,1:K)*(V(:,1:K))';   %SVD truncation -> Tdenoised
has rank K
            mats=mats-Tdenoised+Toeplitzation(2*Tdenoised-mats);
        end
        %at this point, Tdenoised has rank K but is not exactly Toeplitz
        Tdenoised=Toeplitzation(Tdenoised);
        %we reshape the Toeplitz matrix Tdenoised into a Toeplitz matrix with K+1
columns
        Tdenoised=toeplitz([Tdenoised(1,P-K+1:-1:1).';Tdenoised(2:N-P,1)],Tdenoised
(1,P-K+1:P+1));
        [U S V]=svd(Tdenoised,0);
        estimtk=-angle(roots(V(:,K+1)))*tau/2/pi;    %estimated locations in [-tau/2,
tau/2[
        edgecolor='r';


    %%% procedure without denoising, a.k.a. total least squares method
    %%% equivalent to method=1 or method=2 with Nbiter=0.
    elseif method==3
        Tnoisy=toeplitz(vhatm(K+1:N),vhatm(K+1:-1:1));
        [U S V]=svd(Tnoisy,0);
        estimtk=-angle(roots(V(:,K+1)))*tau/2/pi;    %estimated locations in [-tau/2,
tau/2[
        edgecolor=[0.8 0 0.8];
```

```
    elseif method==4
        P=M;            %the matrices have size N-P x P+1. K<=P<=M required.
        if experiment==1, P=6; end  %better results than with P=5.
        Tnoisy=toeplitz(vhatm(P+1:N),vhatm(P+1:-1:1));
        vecnoisy=Tnoisy(:,1);
        Tnoisy=Tnoisy(:,2:end);
        vecnoisy=Tnoisy'*vecnoisy;
        Tnoisy=Tnoisy'*Tnoisy;
        [U,V]=eig(Tnoisy);
        g=zeros(P,1);
        for j=0:K-1
            g=g+U(:,end-j)'*vecnoisy/V(end-j,end-j)*U(:,end-j);
        end
        z=roots([-1;g]);
        [B,I]=sort(abs(z));
        estimtk=-angle(z(I(end-K+1:end)))*tau/2/pi;
        edgecolor=[0.6 0.3 0];

    elseif method==5
        P=M;            %the matrices have size N-P x P+1. K<=P<=M required.
        Y0=zeros(N-P,P);
        Y1=zeros(N-P,P);
        for k=0:N-P-1
            Y0(k+1,:)=vhatm((1:P)+k);
            Y1(k+1,:)=vhatm((2:P+1)+k);
        end
        [U,S,V]=svd(Y1,0);
        Sp=S(1:K,1:K); Up=U(:,1:K); Vp=V(:,1:K);
        [U,S,V]=svd(Y0,0);
        Y0=U(:,1:K)*S(1:K,1:K)*V(:,1:K)';
        Zl=Vp*inv(Sp)*Up'*Y0;
        z=eig(Zl);
        [B,I]=sort(abs(z));
        estimtk=angle(z(I(end-K+1:end)))*tau/2/pi;
        edgecolor=[0 0.6 0.3];

    elseif method==6
        estimtk=-rootmusic(vhatm,K)*tau/2/pi;
        edgecolor=[1 0.7 0];
    end

    ind=find(estimtk<0);
    estimtk(ind)=estimtk(ind)+tau;          %estimated locations in [0,tau[
    estimak=real(pinv(exp(-i*2*pi/tau*(-M:M)'*estimtk'))*vhatm);     %estimated↙
amplitudes
    if displayfigure
        ymin=min(ymin,min(estimak));
        ymax=max(ymax,max(estimak));
        if Nbnoiserea==1
```

```
            stem(estimtk/tau,estimak,'--','Color',↙
edgecolor,'MarkerFaceColor','none','MarkerEdgeColor',edgecolor,'Markersize',7);
        else
            plot(estimtk/tau,estimak,'+','MarkerFaceColor',↙
edgecolor,'MarkerEdgeColor',edgecolor,'Markersize',2);
        end
    end
    if Nbnoiserea>1, waitbar(noiserea/Nbnoiserea); end;
    estimvhatm=exp(-i*2*pi/tau*(-M:M)'*estimtk')*estimak;
    error1=error1+(norm(estimvhatm-vhatm0)^2/N-error1)/noiserea; %robust computation↙
of the mean
    if experiment==1, error2=error2+(mspe(estimtk(1),estimtk(2),tk(1),tk(2),tau)-↙
error2)/noiserea; end;
end     %of loop over noise realizations
toc
if displayfigure, ylim([ymin-0.1 ymax+0.1]); end
if Nbnoiserea>1, close(thewaitbar); end
if displayresults
    disp('Estimated locations:');
    disp(estimtk');
    disp('Estimated amplitudes:');
    disp(estimak');
    fprintf('Lowpass MSE: %g. MSPE: %g\n',error1,error2);
end
end     %of main program


function Matres=Toeplitzation(Mat)
%this function returns a Toeplitz matrix, closest to Mat for the Frobenius norm.
%this is done by simply averaging along the diagonals.
    [height,width]=size(Mat);  %height>=width required
    Matres=Mat;
    for indexcol=2:width
        valdiag=0;
        valdiag2=0;
        for indexrow=1:width-indexcol+1
            valdiag=valdiag+Mat(indexrow,indexcol-1+indexrow);
            valdiag2=valdiag2+Mat(height-indexrow+1,width-indexcol+2-indexrow);
        end
        valdiag=valdiag/(width-indexcol+1);
        valdiag2=valdiag2/(width-indexcol+1);
        for indexrow=1:width-indexcol+1
            Matres(indexrow,indexcol-1+indexrow)=valdiag;
            Matres(height-indexrow+1,width-indexcol+2-indexrow)=valdiag2;
        end
    end
    for indexcol=1:height-width+1
        valdiag=0;
        for indexrow=1:width
```

```
            valdiag=valdiag+Mat(indexcol+indexrow-1,indexrow);
        end
        valdiag=valdiag/width;
        for indexrow=1:width
            Matres(indexcol+indexrow-1,indexrow)=valdiag;
        end
    end
end     %of the function


%this more compact form is slower.
function Matres=Toeplitzation2(Mat)
    [height,width]=size(Mat);
    vec=zeros(height+width-1,1);
    for index=-height+1:width-1
        vec(index+height)=mean(diag(Mat,index));
    end
    Matres=toeplitz(vec(height:-1:1),vec(height:height+width-1));
end


%computes the MSE between the pairs (a,b) and (c,d), for tau-periodic values.
function res=mspe(a,b,c,d,tau)
    aux=0.5*tau;
    e1=(mod(a-c+aux,tau)-aux).^2+(mod(b-d+aux,tau)-aux).^2;
    e2=(mod(a-d+aux,tau)-aux).^2+(mod(b-c+aux,tau)-aux).^2;
    res=min(e1,e2)/2;
end
```