Assignment 1 - Machine Learning

Andreas Timürtas

April 2021

Task 1

To find the weights w_i that minimizes equation 1 we will first differentiate, put the result equal to zero and then solve for the weights.

$$\frac{1}{2}||\mathbf{r}_i - \mathbf{x}_i w_i||_2^2 + \lambda |w_i| \tag{1}$$

$$\begin{aligned}
\frac{\partial}{\partial w_i}\frac{1}{2}||\mathbf{r}_i - \mathbf{x}_i w_i||_2^2 + \lambda|w_i| = \frac{\partial}{\partial w_i}\frac{1}{2}(\mathbf{r}_i^T - \mathbf{x}_i^T w_i)(\mathbf{r}_i - \mathbf{x}_i w_i) + \lambda|w_i| = \\
= \frac{\partial}{\partial w_i}\frac{1}{2}(\mathbf{r}_i^T\mathbf{r}_i - \mathbf{r}_i^T\mathbf{x}_i w_i - \mathbf{x}_i^T\mathbf{r}_i w_i + \mathbf{x}_i^T\mathbf{x}_i w_i^2) + \lambda|w_i| = \\
= \frac{1}{2}(2\mathbf{x}_i^T\mathbf{x}_i w_i - 2\mathbf{x}_i^T\mathbf{r}) + \lambda\frac{w_i}{|w_i|} = \mathbf{x}_i^T\mathbf{x}_i w_i - \mathbf{x}_i^T\mathbf{r} + \lambda\frac{w_i}{|w_i|} = 0 \Longrightarrow \\
\Longrightarrow w_i = \frac{\mathbf{x}_i^T\mathbf{r} - \lambda\frac{w_i}{|w_i|}}{\mathbf{x}_i^T\mathbf{x}_i} = \frac{\mathbf{x}_i^T\mathbf{r} - \lambda\,\mathrm{sgn}(w_i)}{\mathbf{x}_i^T\mathbf{x}_i} = \\
= \frac{\mathbf{x}_i^T\mathbf{r}|\mathbf{x}_i^T\mathbf{r}|}{\mathbf{x}_i^T\mathbf{x}_i|\mathbf{x}_i^T\mathbf{r}|} - \frac{\lambda\,\mathrm{sgn}(w_i)\,|\mathbf{x}_i^T\mathbf{r}|\,\mathbf{x}_i^T\mathbf{r}}{\mathbf{x}_i^T\mathbf{x}_i\,\mathbf{x}_i^T\mathbf{r}\,|\mathbf{x}_i^T\mathbf{r}|} = \frac{\mathbf{x}_i^T\mathbf{r}|\mathbf{x}_i^T\mathbf{r}|}{\mathbf{x}_i^T\mathbf{x}_i|\mathbf{x}_i^T\mathbf{r}|} - \frac{\lambda\,\mathrm{sgn}(w_i)\,\mathrm{sgn}(\mathbf{x}_i^T\mathbf{r})\,\mathbf{x}_i^T\mathbf{r}}{\mathbf{x}_i^T\mathbf{x}_i\,|\mathbf{x}_i^T\mathbf{r}|} = \\
= \frac{\mathbf{x}_i^T\mathbf{r}}{\mathbf{x}_i^T\mathbf{x}_i|\mathbf{x}_i^T\mathbf{r}|}(|\mathbf{x}_i^T\mathbf{r}| - \lambda\,\mathrm{sgn}(w_i)\,\mathrm{sgn}(\mathbf{x}_i^T\mathbf{r}))
\end{aligned} \tag{2}$$

We now have to show that $\mathrm{sgn}(w_i) = \mathrm{sgn}(\mathbf{x}_i^T\mathbf{r})$. To to that we use the fact that $\mathbf{x}_i^T\mathbf{x}_i \geq 0$ and $\lambda > 0$ and return to the differentiated equation.

$$\mathbf{x}_i^T\mathbf{x}_i w_i - \mathbf{x}_i^T\mathbf{r} + \lambda\frac{w_i}{|w_i|} = 0 \iff \mathbf{x}_i^T\mathbf{x}_i w_i + \lambda\frac{w_i}{|w_i|} = \mathbf{x}_i^T\mathbf{r} \tag{3}$$

The sign of the left side only depends on the sign of w_i because $\mathbf{x}_i^T\mathbf{x}_i$ and λ are non-negative. This means that if the equality must hold the sign of $\mathbf{x}_i^T\mathbf{r}$ must be equal to the sign of w_i. Therefore $\mathrm{sgn}(w_i) = \mathrm{sgn}(\mathbf{x}_i^T\mathbf{r})$ holds and the final expression in equation 2 can be written as

$$w_i = \frac{\mathbf{x}_i^T\mathbf{r}}{\mathbf{x}_i^T\mathbf{x}_i|\mathbf{x}_i^T\mathbf{r}|}(|\mathbf{x}_i^T\mathbf{r}| - \lambda) \tag{4}$$

which is what we wanted to show.

Task 2

We want to show that $\hat{w}_i^{(2)} - \hat{w}_i^{(1)} = 0$ when the regression matrix $\mathbf{X}$ is an orthonormal basis. We will first simplify $\mathbf{x}_i^T \mathbf{r}_i^{(j-1)}$.

$$\begin{aligned}
\mathbf{x}_i^T \mathbf{r}_i^{(j-1)} &= \mathbf{x}_i^T (\mathbf{t} - \sum_{l<i} \mathbf{x}_l \hat{w}_l^{(j)} - \sum_{l>i} \mathbf{x}_l \hat{w}_l^{(j-1)}) \\
&= \mathbf{x}_i^T \mathbf{t} - \sum_{l<i} \mathbf{x}_i^T \mathbf{x}_l \hat{w}_l^{(j)} - \sum_{l>i} \mathbf{x}_i^T \mathbf{x}_l \hat{w}_l^{(j-1)} \\
&= \{i \neq l \implies \mathbf{x}_i^T \mathbf{x}_l = 0\} = \mathbf{x}_i^T \mathbf{t}
\end{aligned}$$

The relation in the curly brackets is because the regression matrix is an orthonormal basis. We now continue by inserting the result above in the solution in equation 4.

$$\hat{w}_i^{(2)} - \hat{w}_i^{(1)} = \frac{\mathbf{x}_i^T \mathbf{t}}{|\mathbf{x}_i^T \mathbf{t}|}(|\mathbf{x}_i^T \mathbf{t}| - \lambda) - \frac{\mathbf{x}_i^T \mathbf{t}}{|\mathbf{x}_i^T \mathbf{t}|}(|\mathbf{x}_i^T \mathbf{t}| - \lambda) = 0 \tag{5}$$

This show that the coordinate descent solver in equation 4 converges in at most 1 full pass.

Task 3

If the data $\mathbf{t}$ is generated through

$$\mathbf{t} = \mathbf{X}\mathbf{w}^* + \mathbf{e}, \quad \mathbf{e} \backsim \mathcal{N}(\mathbf{0}_N, \sigma \mathbf{I}_N) \tag{6}$$

where $\mathbf{w}^*$ is the weights used to generate the the data and $\mathcal{N}(\cdot)$ is a Guassian distribution then a bias will be introduced when using LASSO to estimate $\hat{\mathbf{w}}$. We want to show that that bias

$$\lim_{\sigma \to 0} E(\hat{w}_i^{(1)} - w_i^*) = \begin{cases} -\lambda, & w_i^* > \lambda \\ -w_i^*, & |w_i^*| \leq \lambda \qquad \forall i \\ \lambda, & w_i^* < -\lambda \end{cases} . \tag{7}$$

We start by examining equation 4 with the simplification we used in task 2.

$$\begin{aligned}
\hat{w}_i^{(1)} &= \frac{\mathbf{x}_i^T \mathbf{t}}{|\mathbf{x}_i^T \mathbf{t}|}(|\mathbf{x}_i^T \mathbf{t}| - \lambda) \\
&= \frac{\mathbf{x}_i^T (\mathbf{X}\mathbf{w}^* + \mathbf{e})}{|\mathbf{x}_i^T (\mathbf{X}\mathbf{w}^* + \mathbf{e})|}(|\mathbf{x}_i^T (\mathbf{X}\mathbf{w}^* + \mathbf{e})| - \lambda) \\
&= \frac{\mathbf{x}_i^T \mathbf{X}\mathbf{w}^* + \mathbf{x}_i^T \mathbf{e}}{|\mathbf{x}_i^T \mathbf{X}\mathbf{w}^* + \mathbf{x}_i^T \mathbf{e}|}(|\mathbf{x}_i^T \mathbf{X}\mathbf{w}^* + \mathbf{x}_i^T \mathbf{e}| - \lambda)
\end{aligned}$$

Here we can use that we have an orthonormal regression matrix $\mathbf{X}$ with the property that $\mathbf{X}^T\mathbf{X} = \mathbf{I}$ which gives us the simplification $\mathbf{x}_i^T \mathbf{X}\mathbf{w}^* = w_i^*$.

$$\hat{w}_i^{(1)} = \frac{w_i^* + \mathbf{x}_i^T \mathbf{e}}{|w_i^* + \mathbf{x}_i^T \mathbf{e}|}(|w_i^* + \mathbf{x}_i^T \mathbf{e}| - \lambda) \tag{8}$$

This expression holds for (simplified like we did above)

$$|\mathbf{x}_i^T \mathbf{r}^{(j-1)}| = |w_i^*| > \lambda \tag{9}$$

and gives us two cases, $w_i^* > \lambda$ and $w_i^* < -\lambda$. We insert the expression of w_i^* into the right side of equation 7 and then use that $\lim_{\sigma \to 0} \mathbf{e} = 0$.

$$
\begin{aligned}
\lim_{\sigma\to 0} E(\hat{w}_i^{(1)} - w_i^*) &= \lim_{\sigma\to 0} E\left(\frac{w_i^* + \mathbf{x}_i^T\mathbf{e}}{|w_i^* + \mathbf{x}_i^T\mathbf{e}|}(|w_i^* + \mathbf{x}_i^T\mathbf{e}| - \lambda) - w_i^*\right) \\
&= E\left(\frac{w_i^*}{|w_i^*|}(|w_i^*| - \lambda) - w_i^*\right) \\
&= E\left(w_i^* - \frac{w_i^*}{|w_i^*|}\lambda - w_i^*\right) = E\left(-\frac{w_i^*}{|w_i^*|}\lambda\right) = \begin{cases} -\lambda, & w_i^* > \lambda \\ \lambda. & w_i^* < -\lambda \end{cases}
\end{aligned}
$$

We have now shown the first and third case in equation 7 and will continue by showing the second case. Assume that $|\mathbf{x}_i^T\mathbf{r}^{(j-1)}| \leq \lambda$ then $\hat{w}_i^{(j)} = 0$, which we insert into the right side of equation 7.

$$
\lim_{\sigma\to 0} E(\hat{w}_i^{(1)} - w_i^*) = \lim_{\sigma\to 0} E(-w_i^*) = -w_i^*, \quad |\mathbf{x}_i^T\mathbf{r}^{(j-1)}| \leq \lambda
$$

We showed before that $\mathbf{x}_i^T\mathbf{r}^{(j-1)} = w_i^*$ which gives us the three cases in equation 7.

Task 4

The figures 1, 2 and 3 illustrates how different values of the hyperparameter λ affect the estimated model. When λ is set to a small value, as in figure 1, the estimation is fitted to the noise in the data and curves to nearly perfectly follow the points.

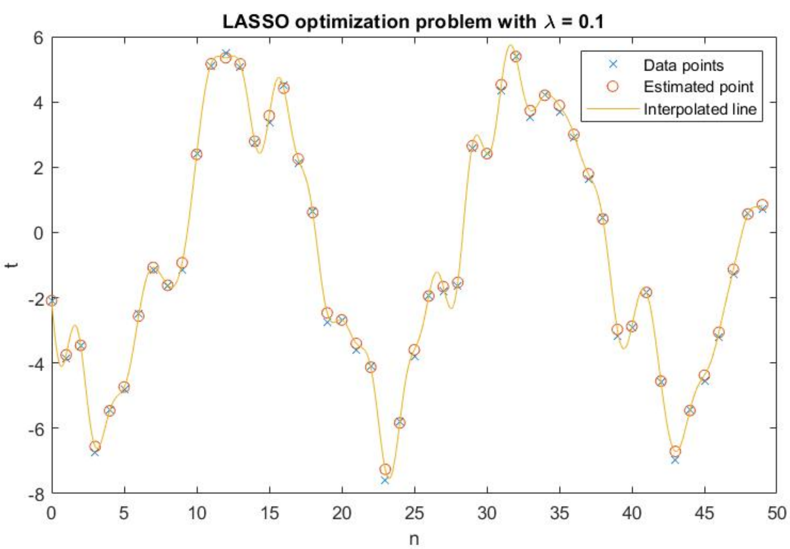

Figure 1: LASSO estimation using $\lambda = 0.1$

When instead λ is larger as in figure 2 the estimation does not have the same capability to curve to fit the data points. This results in an estimation that is underfitted.

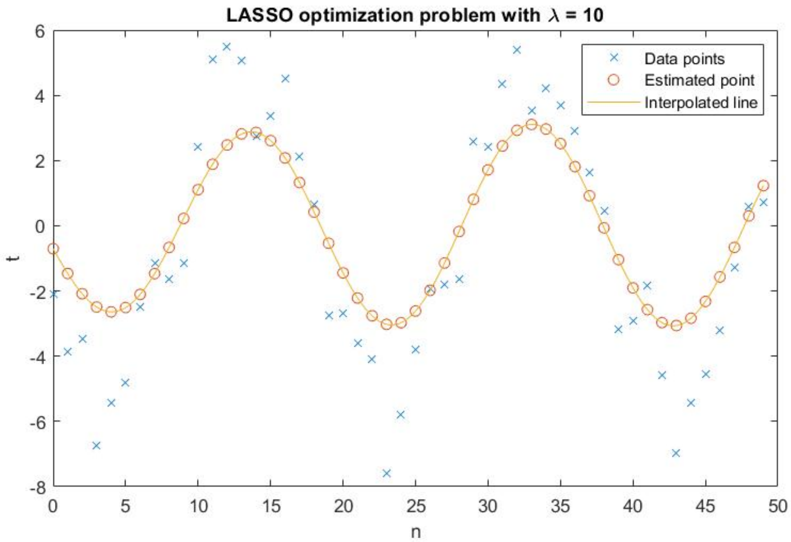

Figure 2: LASSO estimation using $\lambda = 10$

After some testing I found that a λ equal to around 4 gave an estimation that followed the data good and at the same time did not follow the noise to closely. This result can be seen in figure 3.

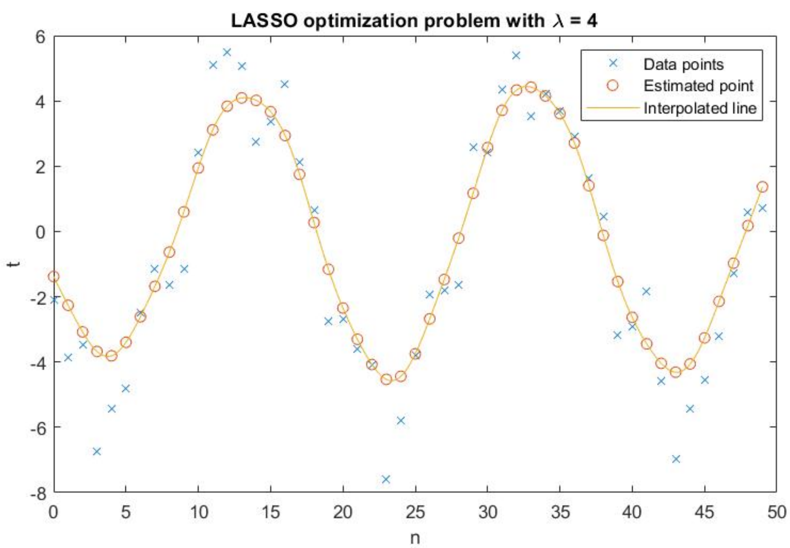

Figure 3: LASSO estimation using $\lambda = 4$

The number of coordinates used in the three estimates above was 207, 6 and 11 for figure 1, 2 respectively 3. If we compare to the number of coordinates needed to generate the data that is 4 we can see that all our estimations need more coordinates.

Task 5

The plot in figure 4 show how the root-mean-square error changes when the regularization hyperparameter λ increases. The red line is the error for the estimation data, the data the model trained on while the blue line is the error for the validation data, the data the model had not seen during training. As we can see the estimation RMSE increases with a larger λ because the model no longer have the same capability to follow the data and overfit. The validation RMSE in the other hand first decreases when the model can not follow the points in the estimation data as closely. After λ increases further the model will underfit for both the data sets and therefore both RMSE will increase. The optimal λ is chosen to be the value that has the smallest error for the validation data. This λ was found to be equal to 1.8738.

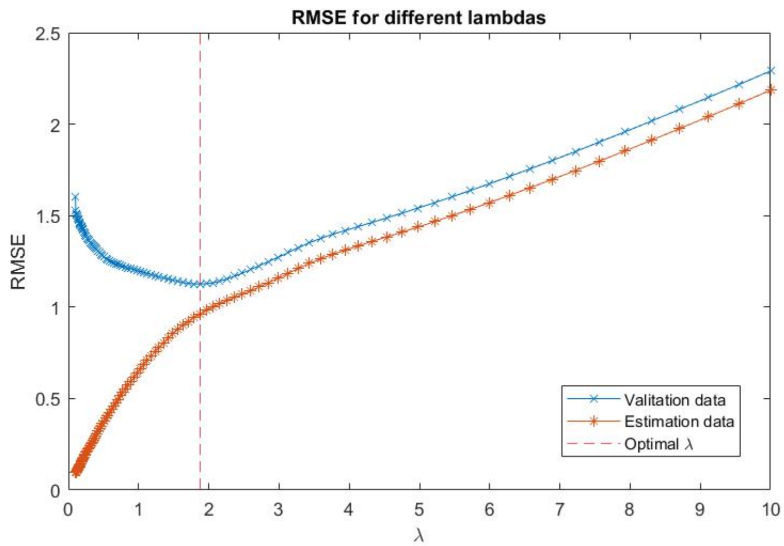

Figure 4: RMSE against different λ for both estimation and validation data

Figure 5 illustrates the reconstructed estimation with the optimal λ given by the k-fold cross validation process.

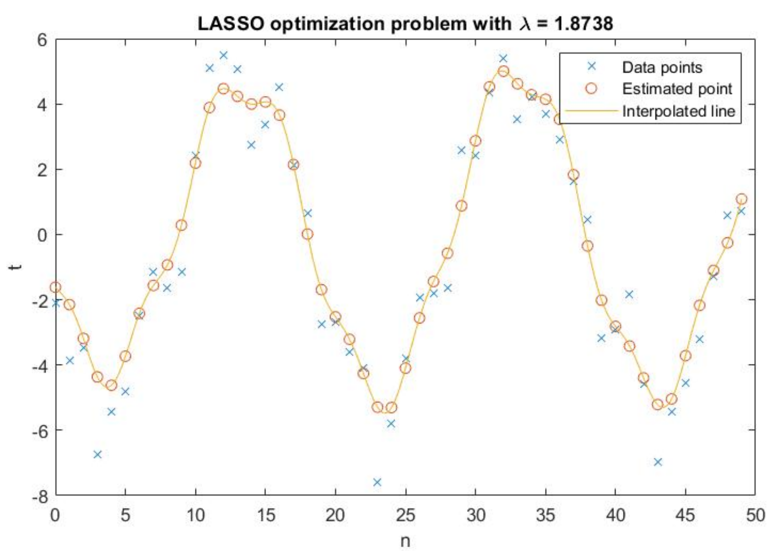

Figure 5: Reconstruction plot for the optimal λ

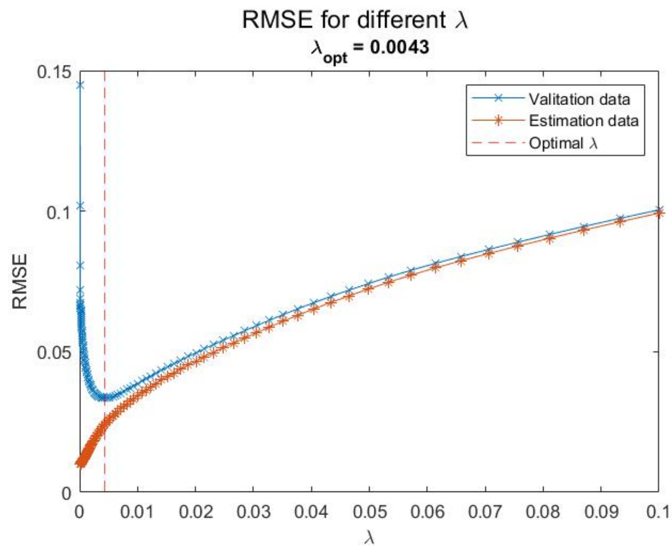

Figure 6: RMSE against different λ for both estimation and validation data

Task 6

Similarly to figure 4, figure 6 shows the RMSE error for both the estimation and validation data sets against different values of λ. Again we can see the behaviour where the estimation RMSE increases for larger λ while validation RMSE first decreases and the increases. The λ that gives the minimal RMSE for the validation data was chosen as the optimal λ and was found to be $\lambda_{opt} = 0.0043$.

Task 7

The original audio have some clear noise in the background and after denoising with our optimal λ the resulting audio had less noise in it. At the same time the volume seems to have decreased and there still remain some sounds that probably should not be in the audio. When increasing λ to 0.1 and denoising the audio most of the frequencies disappear and the sounds gets really low, almost as we are underwater. When we instead increases λ the audio resembles the original audio more and more and the noise starts comes back. One could try to find a more suitable λ to denoise the audio but if it is essential that the quality of the resulting audio is the best possible one then there is probably some more robust methods to use.

1 MATLAB code

1.1 Main code

```
%% Task 4
lambda = 10;
what = skeleton_lasso_ccd(t, X, lambda);

plot(n, t, 'x')
hold on
plot(n, X*what, 'o')
plot(ninterp, Xinterp*what)
title('LASSO optimization problem with {\lambda} = 10')
xlabel('n')
ylabel('t')
legend('Data points', 'Estimated point', 'Interpolated line')
disp(['Number of weights in what: ' num2str(length(find(what))) ' lambda = ' num2str(lambda)])
%% Task 5
lambdavec = exp(linspace(log(0.1), log(10), 100));
[wopt,lambdaopt,RMSEval,RMSEest] = skeleton_lasso_cv(t, X, lambdavec, 5);

figure(1)
plot(lambdavec, RMSEval, '-x')
hold on
plot(lambdavec, RMSEest, '-*')
xline(lambdaopt, '--r');
xlabel('{\lambda}')
ylabel('RMSE')
title('RMSE for different lambdas')
legend('Valitation data', 'Estimation data', 'Optimal {\lambda}')

what = skeleton_lasso_ccd(t, X, lambdaopt);
%% Task 5
figure(2)
plot(n, t, 'x')
hold on
plot(n, X*what, 'o')
plot(ninterp, Xinterp*what)
title('LASSO optimization problem with {\lambda} = 1.8738')
xlabel('n')
ylabel('t')
legend('Data points', 'Estimated point', 'Interpolated line')

%% Task 6
lambdavec = exp(linspace(log(0.0001), log(0.1), 100));
[Wopt,lambdaopt,RMSEval,RMSEest] = skeleton_multiframe_lasso_cv(Ttrain,Xaudio,lambdavec,5);
%% Task 6
plot(lambdavec, RMSEval, '-x')
hold on
plot(lambdavec, RMSEest, '-*')
xline(lambdaopt, '--r');
xlabel('{\lambda}')
ylabel('RMSE')
```

```matlab
title('{\lambda_{opt}} = 0.0043')
suptitle('RMSE for different {\lambda}')
legend('Valitation data', 'Estimation data', 'Optimal {\lambda}')
%% Task 7
save('task5','Wopt','lambdaopt','RMSEval','RMSEest')
%% Testing for task 7
Tq = fft(Ttest);
f = (0:length(Tq)-1)*fs/length(Tq);
Yq = fft(Yclean);
subplot(211)
plot(f,abs(Tq))
subplot(212)
plot(f,abs(Yq))
%% Play audio
soundsc(Ttest, fs)
pause(5)
soundsc(Yclean, fs)
%% Denoise audio
[Yclean] = lasso_denoise(Ttest,X,0.1);
save('denoised_audio','Yclean','fs')
```

1.2 LASSO cyclic coordinate descent

```matlab
function what = lasso_ccd(t,X,lambda,wold)
% what = lasso_ccd(t,X,lambda,wold)
% Solves the LASSO optimization problem using cyclic coordinate descent.
%
%   Output:
%    what    - Mx1 LASSO estimate using cyclic coordinate descent algorithm
%
%   inputs:
%   t        - Nx1 data column vector
%   X        - NxM regression matrix
%   lambda   - 1x1 hyperparameter value
%   (optional)
%   wold     - Mx1 lasso estimate used for warm-starting the solution.

% Check for match between t and X
[N,M] = size(X);
if size(t,1) ~= N
    disp('Sizes in t and X do not match!')
    what = [];
    return
end

if nargin < 4
    wold = zeros(M,1); % set wold to zeros if warm-start is unavailable
end

% Optimization variables and preallocation
Niter = 50; % number of iterations
updatecycle = 5; % at which intensity all variables should be updated.
zero_tol = lambda; % what is to be considered equal to zero in support.
w = wold; % set intial w to wold from previous lasso estimate, if available
```

```
wsup = double(abs(w)>zero_tol); % defines the non-zero indices of w

r = t - X*w; % calculate residual and use it instead of y-Xw with proper indexing.

for kiter = 1:Niter

    % Snippet below is a common way of speeding up the estimation process. Use it
    % if you like. Basically, only the non-zero estimates are updated
    % at every iteration. The zero estimates are only updated every
    % updatecycle number of iterations. Use to your liking. Otherwise use
    % contents of else statement.
    if rem(kiter,updatecycle) && kiter>2
        kind_nonzero = find(wsup);
        randind = randperm(length(kind_nonzero));
        kindvec_random = kind_nonzero(randind);
    else
        kindvec = 1:M;
        kindvec_random = kindvec(randperm(length(kindvec)));
    end

    % sweep over coordinates, in randomized order defined by kInd_random
    for ksweep = 1:length(kindvec_random)
        kind = kindvec_random(ksweep); % Pick out current coordinate to modify.

        x = X(:,kind); % ... select current regression vector
        r = r + x*w(kind); % ... put impact of old w(kind) back into the residual.
        if abs(x'*r) > lambda
            w(kind) = (x'*r)/(x'*x*abs(x'*r))*(abs(x'*r)-lambda);
        else
            w(kind) = 0;
        end % ... update the lasso estimate at coordinate kind
        r = r - x*w(kind); % ... remove impact of newly estimated w(kind) from residual.

        wsup(kind) = double(abs(w(kind))>zero_tol); % update whether w(kind) is zero or not.

    end
end

what = w; % assign function output.
end
```

1.3 LASSO cross validation

```
function [wopt,lambdaopt,RMSEval,RMSEest] = lasso_cv(t,X,lambdavec,K)
% [wopt,lambdaopt,VMSE,EMSE] = lasso_cv(t,X,lambdavec)
% Calculates the LASSO solution problem and trains the hyperparameter using
% cross-validation.
%
%   Output:
%   wopt        - mx1 LASSO estimate for optimal lambda
%   lambdaopt   - optimal lambda value
%   MSEval      - vector of validation MSE values for lambdas in grid
%   MSEest      - vector of estimation MSE values for lambdas in grid
%
```

```
%   inputs:
%   y            - nx1 data column vector
%   X            - nxm regression matrix
%   lambdavec    - vector grid of possible hyperparameters
%   K            - number of folds

[N,M] = size(X);
Nlam = length(lambdavec);

% Preallocate
SEval = zeros(K,Nlam);
SEest = zeros(K,Nlam);


% cross-validation indexing
index = 1:N;
randomind = index(randperm(length(index))); % Select random indices for validation and estimation
location = 1; % Index start when moving through the folds
Nval = floor(N/K); % How many samples per fold
hop = Nval; % How many samples to skip when moving to the next fold.


for kfold = 1:K

    if kfold == K
        valind = randomind(location+1:end); % Select validation indices
        estind = randomind(1: location); % Select estimation indices
    else
        valind = randomind(location+1:location+hop); % Select validation indices
        estind = randomind([1: location location+hop+1:end]); % Select estimation indices
    end

    assert(isempty(intersect(valind,estind)), "There are overlapping indices in valind and estind!"); % assert empty intersection between valind and estind
    wold = zeros(M,1); % Initialize estimate for warm-starting.

    for klam = 1:Nlam

        what = skeleton_lasso_ccd(t(estind), X(estind,:), lambdavec(klam), wold); % Calculate LASSO estimate on estimation indices for the current lambda-value.

        SEval(kfold,klam) = (1/length(valind)) * norm(t(valind)-X(valind,:)*what)^2; % Calculate validation error for this estimate
        SEest(kfold,klam) = (1/length(estind)) * norm(t(estind)-X(estind,:)*what)^2; % Calculate estimation error for this estimate

        wold = what; % Set current estimate as old estimate for next lambda-value.
        disp(['Fold: ' num2str(kfold) ', lambda-index: ' num2str(klam)]) % Display current fold and lambda-index.

    end

    location = location+hop; % Hop to location for next fold.
```

```
end


MSEval = mean(SEval,1); % Calculate MSE_val as mean of validation error over the folds.
MSEest = mean(SEest,1); % Calculate MSE_est as mean of estimation error over the folds.
[~, lambdaind] = min(MSEval);
lambdaopt = lambdavec(lambdaind); % Select optimal lambda


RMSEval = sqrt(MSEval);
RMSEest = sqrt(MSEest);


wopt = skeleton_lasso_ccd(t, X, lambdaopt); % Calculate LASSO estimate for selected lambda using all data.

end
```

1.4 LASSO multiframe cross validation

```
function [Wopt,lambdaopt,RMSEval,RMSEest] = multiframe_lasso_cv(T,X,lambdavec,K)
% [wopt,lambdaopt,VMSE,EMSE] = multiframe_lasso_cv(T,X,lambdavec,n)
% Calculates the LASSO solution for all frames and trains the
% hyperparameter using cross-validation.
%
%   Output:
%   Wopt        - mxnframes LASSO estimate for optimal lambda
%   lambdaopt   - optimal lambda value
%   VMSE        - vector of validation MSE values for lambdas in grid
%   EMSE        - vector of estimation MSE values for lambdas in grid
%
%   inputs:
%   T           - NNx1 data column vector
%   X           - NxM regression matrix
%   lambdavec   - vector grid of possible hyperparameters
%   K           - number of folds

% Define some sizes
NN = length(T);
[N,M] = size(X);
Nlam = length(lambdavec);

% Set indexing parameters for moving through the frames.
framehop = N;
idx = (1:N)';
framelocation = 0;
Nframes = 0;
while framelocation + N <= NN
    Nframes = Nframes + 1;
    framelocation = framelocation + framehop;
end % Calculate number of frames.

% Preallocate
Wopt = zeros(M,Nframes);
```

```
SEval = zeros(K,Nlam);
SEest = zeros(K,Nlam);

% Set indexing parameter for the cross-validation indexing
Nval = floor(N/K);
cvhop = Nval;
randomind = idx(randperm(length(idx)));% Select random indices for picking out validation and
    estimation indices.

framelocation = 0;
for kframe = 1:Nframes % First loop over frames

    cvlocation = 0;

    for kfold = 1:K % Then loop over the folds

        if kfold == K
            valind = randomind(cvlocation+1:end); % Select validation indices
            estind = randomind(1: cvlocation); % Select estimation indices
        else
            valind = randomind(cvlocation+1:cvlocation+cvhop); % Select validation indices
            estind = randomind([1: cvlocation cvlocation+cvhop+1:end]); % Select estimation
                indices
        end
        assert(isempty(intersect(valind,estind)), "There are overlapping indices in valind
            and estind!"); % assert empty intersection between valind and estind


        t = T(framelocation + idx); % Set data in this frame
        wold = zeros(M,1);  % Initialize old weights for warm-starting.

        for klam = 1:Nlam  % Finally loop over the lambda grid

            what = skeleton_lasso_ccd(t(estind), X(estind,:), lambdavec(klam), wold);%
                Calculate LASSO estimate at current frame, fold, and lambda

            SEval(kfold,klam) = SEval(kfold,klam) + (1/length(valind))*norm(t(valind)-X(
                valind,:)*what)^2; % Add validation error at current frame, fold and lambda
                to the validation error for this fold and lambda, summing tthe error over the
                 frames
            SEest(kfold,klam) = SEest(kfold,klam) + (1/length(estind))*norm(t(estind)-X(
                estind,:)*what)^2; % Add estimation error at current frame, fold and lambda
                to the estimation error for this fold and lambda, summing tthe error over the
                 frames

            wold = what; % Set current LASSO estimate as estimate for warm-starting.
            disp(['Frame: ' num2str(kframe) ', Fold: ' num2str(kfold) ', Hyperparam: '
                num2str(klam)]) % Display progress through frames, folds and lambda-indices.
        end

        cvlocation = cvlocation+cvhop; % Hop to location for next fold.
    end

    framelocation = framelocation + framehop; % Hop to location for next frame.
```

```
end



MSEval = mean(SEval,1); % Average validation error across folds
MSEest = mean(SEest,1); % Average estimation error across folds
[~, lambdaind] = min(MSEval);
lambdaopt = lambdavec(lambdaind); % Select optimal lambda

% Move through frames and calculate LASSO estimates using both estimation
% and validation data, store in Wopt.
framelocation = 0;
for kframe = 1:Nframes
    t = T(framelocation + idx);
    Wopt(:,kframe) = skeleton_lasso_ccd(t, X, lambdaopt, wold);
    framelocation = framelocation + framehop;
end

RMSEval = sqrt(MSEval);
RMSEest = sqrt(MSEest);

end
```