RobustSP toolbox: A Matlab Toolbox for Robust Signal Processing

October 15, 2018

Contents

Summary

This document contains the list of functions that are currently available in the RobustSP toolbox: a Matlab toolbox for robust signal processing. The toolbox can be freely used for non-commercial use only.

Please make appropriate references to our book:

Zoubir, A. M., Koivunen, V., Ollila, E., and Muma, M.
Robust Statistics for Signal Processing
Cambridge University Press, 2018.

1 Location and Scale Estimation

Huber's and Tukey's location M-estimates for real and complex valued data, or joint Huber's M-estimates of location and scale can be conveniently computed with the functions `Mloc`, and `Mlocscale`. Huber's and Tukey's scale M-estimates for real data are implemented in `MscaleHUB`, and `MscaleTUK`. The functions `MlocHUB`, `MlocTUK`, are included for educational purposes, and are limited to real valued data.

1.1 `MlocHUB` computes Huber's M-estimate of location with previously computed scale.

`MlocHUB` computes Huber's M-estimator, i.e., solves

$$\sum_{i=1}^{N} \psi\left(\frac{y_i - \hat{\mu}}{\hat{\sigma}}\right) = 0 \tag{1}$$

where

$$\psi(x) = \begin{cases} x & |x| \leq c \\ c\ \mathsf{sign}(x) & |x| > c \end{cases} \tag{2}$$

and $\hat{\sigma}$ is a previously computed robust scale estimate, for example, the normalized median absolute deviation

$$\mathsf{madn}(\mathbf{y}) = 1.4826 \cdot \mathsf{med}(|\mathbf{y} - \mathsf{med}(\mathbf{y})|). \tag{3}$$

The estimate is obtained via an iterative re-weighting algorithm, as summarized in Algorithm 1, where $\xi \in \mathbb{R}$ is a small positive constant. Alternatively, `MlocHUB` can also be terminated after a fixed number of iterations consistent with the stopping criterion being $n < N_{\text{iter}}$. For location estimation, the weights are determined by

$$W(x) = \begin{cases} \psi(x)/x, & \text{if } x \neq 0, \\ \left.\frac{d\psi(x)}{dx}\right|_{x=0}, & \text{if } x = 0. \end{cases} \tag{4}$$

1.1.1 Syntax

```
% Mloc_HUB computes Huber's M-estimate of
% location, i.e.,
%
% mu_hat = arg min_mu SUM_i rho_HUB(y_i - mu)
%
%
%   INPUTS:
%           y: real valued data vector of size N x 1
%           c: tuning constant c≥0
%
%   OUTPUTS:
%           mu_hat: Huber's M-estimate of location
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

1.1.2 Dependencies

`madn,whub`

Algorithm 1: `MlocHUB`: computes Huber's M-estimate of location with previously computed scale.

input : observations $\mathbf{y} \in \mathbb{R}^N$, tuning parameter $c \in \mathbb{R}$, scale estimate $\hat{\sigma}$
output : $\hat{\mu}$ that solves (1)
initialize: initial location estimate $\hat{\mu}^{(0)}$, iteration index $n = 1$, tolerance level $\xi \in \mathbb{R}$

while $\frac{|\hat{\mu}^{(n+1)} - \hat{\mu}^{(n)}|}{\hat{\sigma}} > \xi$ **do**

compute weights

$$w_i^{(n)} = W\left(\frac{y_i - \hat{\mu}^{(n)}}{\hat{\sigma}}\right)$$

compute location estimates

$$\hat{\mu}^{(n+1)} = \frac{\sum_{i=1}^N w_i^{(n)} y_i}{\sum_{i=1}^N w_i^{(n)}}$$

increment iteration index

$$n \leftarrow n + 1$$

return $\hat{\mu} \leftarrow \hat{\mu}^{(n+1)}$

1.1.3 Example

```
rng(0);
N = 100;
y = randn(N,1);
mu_hat = MlocHUB(y);
```

1.2 `MlocTUK` computes Tukey's M-estimate of location with previously computed scale.

`MlocTUK` computes Tukey's M-estimator, i.e., it solves Eq. (1) with

$$\psi(x) = \begin{cases} x\left(1 - \frac{x^2}{c^2}\right)^2 & |x| \le c \\ 0 & |x| > c. \end{cases} \tag{5}$$

and $\hat{\sigma}$ is a previously computed robust scale estimate, for example, the normalized median absolute deviation, see Eq. (3).

The estimate is obtained via an iterative re-weighting algorithm, as summarized in Algorithm 1, where $\xi \in \mathbb{R}$ is a small positive constant. Alternatively, MlocTUK can also be terminated after a fixed number of iterations consistent with the stopping criterion being $n < N_{\text{iter}}$. For location estimation, the weights are determined by Eq. (4).

Tukey's location M-estimates for real and complex valued data can be conveniently computed with the function Mloc.

1.2.1 Syntax

```
% Mloc_TUK computes Tukey's M-estimate of
% location, i.e.,
%
% mu_hat = arg min_mu SUM_i rho_TUK(y_i - mu)
%
```

```
%
%   INPUTS:
%           y: real valued data vector of size N x 1
%           c: tuning constant c≥0
%
%   OUTPUTS:
%           mu_hat: Tukey's M-estimate of location
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

1.2.2 Dependencies

madn,wtuk

1.2.3 Example

```
rng(0);
N = 100;
y = randn(N,1);
mu_hat = MlocTUK(y);
```

1.3 `Mloc` computes an M-estimate of location with previously computed scale for real and complex valued data.

1.3.1 Syntax

```
function [mu_hat] = Mloc(y,lossfun)

% Mloc computes the M-estimates of location using an auxiliary scale
% estimate. It uses the iterative reweighted least squares (IRWLS) algorithm
%
% INPUTS:
%       y : (numeric) data vector of size N x 1
% lossfun : (string) either 'huber' or 'tukey' to identify the desired
%           loss function
% OUTPUTS:
%           mu_hat: M-estimate of location
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

1.3.2 Dependencies

whub,wtuk,Mreg

1.3.3 Example

```
rng(0);
N = 100;
y = randn(N,1);
% Example 1: Huber's M-estimator
mu_hat = Mloc(y,'huber');
% Example 2: Tukey's M-estimator
mu_hat = Mloc(y,'tukey');
```

1.4 `MscaleHUB` computes Huber's M-estimate of scale with previously computed location.

M-estimates of scale are defined by

$$\frac{1}{N}\sum_{i=1}^{N}\psi\left(\frac{y_i-\hat{\mu}}{\hat{\sigma}}\right)\cdot\left(\frac{y_i-\hat{\mu}}{\hat{\sigma}}\right)=b. \tag{6}$$

Here, b is a positive constant that must satisfy $0<b<\rho(\infty)$. If $f(y_i|\mu,\sigma)$ is symmetric, then $\rho(\mu,\sigma|\mathbf{y})$ is even, and hence, $\psi(\mu,\sigma|\mathbf{y})$ is odd. M-estimates of scale can be represented as a weighted root-mean-square estimate. Let $W(x)$ be defined as in (4). Then, the equation for the M-estimation of scale (6) becomes

$$\begin{aligned}\frac{1}{b}\sum_{i=1}^{N}W\left(\frac{y_i-\hat{\mu}}{\hat{\sigma}}\right)\cdot\left(\frac{y_i-\hat{\mu}}{\hat{\sigma}}\right)^2&=N\\ \Leftrightarrow\hat{\sigma}=\sqrt{\frac{1}{Nb}\sum_{i=1}^{N}W\left(\frac{y_i-\hat{\mu}}{\hat{\sigma}}\right)\cdot(y_i-\hat{\mu})^2}.&\end{aligned} \tag{7}$$

Based on (7), Huber's M-estimates of scale, with previously computed robust location estimate $\hat{\mu}$, are computed through use of Algorithm 2 with $W(x)$ defined by Eq. (2) and Eq. (4). The default location estimate is the median, and for the initial scale estimate, we use the normalized median absolute deviation as defined in Eq. (3). In Algorithm 2, $\xi\in\mathbb{R}$ is a small positive

Algorithm 2: `MscaleHUB`: computes Huber's M-estimate of scale with previously computed location.

input : observations $\mathbf{y}\in\mathbb{R}^N$, tuning parameter $c\in\mathbb{R}$, location estimate $\hat{\mu}$
output : $\hat{\sigma}$ that solves (7)
initialize: initial scale estimate $\hat{\sigma}^{(0)}$, iteration index $n=1$, tolerance level $\xi\in\mathbb{R}$

while $|\hat{\sigma}^{(n+1)}/\hat{\sigma}^{(n)}-1|>\xi$ **do**

compute weights

$$w_i^{(n)}=W\left(\frac{y_i-\hat{\mu}}{\hat{\sigma}^{(n)}}\right)$$

compute scale estimates

$$\hat{\sigma}^{(n+1)}=\sqrt{\frac{1}{Nb}\sum_{i=1}^{N}w_i^{(n)}(y_i-\hat{\mu})^2}$$

increment iteration index

$$n\leftarrow n+1$$

return $\hat{\sigma}\leftarrow\hat{\sigma}^{(n+1)}$

constant, and Mscale can also be terminated after a fixed number of iterations. The positive constant b must satisfy $0<b<\rho(\infty)$ and is chosen as $\mathsf{E}[\rho(u)]$, where u is a standard normal random variable to achieve consistency with the Gaussian distribution.

1.4.1 Syntax

```
function sigma_hat = MscaleHUB(y,c)
% Mscale_HUB computes Huber's M-estimate of
% scale.
%
%
%   INPUTS:
%           y: real valued data vector of size N x 1
%           c: tuning constant c≥0
%
%   OUTPUTS:
%           sigma_hat: Huber's M-estimate of scale
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

1.4.2 Dependencies

madn,whub

1.4.3 Example

```
rng(0);
N = 100;
y = randn(N,1);
[mu_hat,sigma_hat] = MscaleHUB(y);
```

1.5 `MscaleTUK` computes Tukey's M-estimate of scale with previously computed location.

Based on (7), Tukey's M-estimates of scale, with previously computed robust location estimate $\hat{\mu}$, are computed through use of Algorithm 2 with $W(x)$ defined by Eq. (5) and Eq. (4). The default location estimate is the median, and for the initial scale estimate, we use the normalized median absolute deviation as defined in Eq. (3).

1.5.1 Syntax

```
function sigma_hat = MscaleTUK(y,c)
% Mscale_TUK computes Tukey's M-estimate of
% scale.
%
%
%   INPUTS:
%           y: real valued data vector of size N x 1
%           c: tuning constant c≥0
%
%   OUTPUTS:
%           sigma_hat: Tukey's M-estimate of scale
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

1.5.2 Dependencies

madn,wtuk

1.5.3 Example

```
rng(0);
N = 100;
y = randn(N,1);
[mu_hat,sigma_hat] = MscaleTUK(y);
```

1.6 `MlocscaleHUB` computes computes Huber's M-estimates of location and scale for real and complex valued data.

1.6.1 Syntax

```
function [mu_hat,sigma_hat] = MlocscaleHUB(y,c)

% Mlocscale computes Huber's joint M-estimates of location and scale.
%
%   INPUTS:
%           y: real valued data vector of size N x 1
%           c: tuning constant c≥0
%
%   OUTPUTS:
%           mu_hat: Huber's M-estimate of location
%           sigma_hat: Huber's M-estimate of scale
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

1.6.2 Dependencies

whub,hubreg

1.6.3 Example

```
rng(0);
N = 100;
y = randn(N,1);
[mu_hat,sigma_hat] = MlocscaleHUB(y);
```

1.7 Example: Sensitivity Curves for Location and Scale Estimation

Running the code below you obtain Figure 1 and Figure 2. This example was given in Section 1.3.1 and in Figure 1.5 and Figure 1.6 of the book.

```
% fix seed of random number generator for reproducibility
rng(2);
% number of measurements
N = 100;
% DC voltage in AWGN
x_N_minus1 = randn(N-1,1)+5;
% outlier values
Δ_x = linspace(0,10,1000);

% sensitivity curve for mean
SC_mean = zeros(size(Δ_x));
mu_hat = mean(x_N_minus1);
for ii = 1:length(Δ_x)
    SC_mean(ii) = N*(mean([x_N_minus1; Δ_x(ii)])-mu_hat);
end

% sensitivity curve for median
SC_med = zeros(size(Δ_x));
mu_hat = median(x_N_minus1);
for ii = 1:length(Δ_x)
    SC_med(ii) = N*(median([x_N_minus1; Δ_x(ii)])-mu_hat);
end

% sensitivity curve for Huber's location estimator
```

```
c =  1.3415;
SC_hub = zeros(size(Δ_x));
mu_hat = MlocHUB(x_N_minus1,c);
for ii = 1:length(Δ_x)
    SC_hub(ii) = N*(MlocHUB([x_N_minus1; Δ_x(ii)],c)-mu_hat);
end

% sensitivity curve for Tukey's location estimator
c = 4.68;
SC_tuk = zeros(size(Δ_x));
mu_hat = MlocTUK(x_N_minus1,c);
for ii = 1:length(Δ_x)
    SC_tuk(ii) = N*(MlocTUK([x_N_minus1; Δ_x(ii)],c)-mu_hat);
end

figure,
set(gca,'FontSize',18)
hold on
plot(Δ_x,SC_mean-mean(SC_mean),'linewidth',2)
plot(Δ_x,SC_med-mean(SC_med),'linewidth',2)
plot(Δ_x,SC_hub-mean(SC_hub),'linewidth',2)
plot(Δ_x,SC_tuk-mean(SC_tuk),'linewidth',2)
grid on
xlabel('Outlier value');
ylabel('Sensitivity curve');
legend('mean', 'median', 'Huber M', 'Tukey M');
```

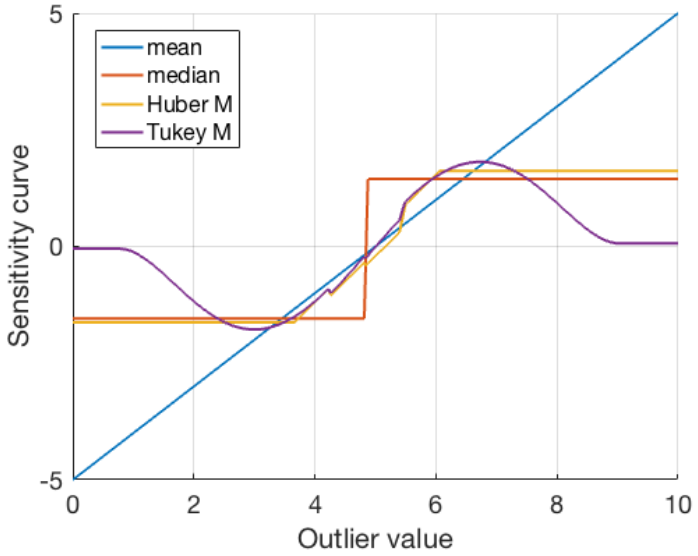


Figure 1: Sensitivity curves for location estimation.

```
% fix seed of random number generator for reproducibility
rng(2);
N = 100;
x_N_minus1 = randn(N-1,1)+5;
Δ_x = linspace(0,10,1000);
```

```
% Sensitivity Curve for standard deviation
SC_std = zeros(size(Δ_x));
std_hat = std(x_N_minus1);
for ii = 1:length(Δ_x)
    SC_std(ii) = N*(std([x_N_minus1; Δ_x(ii)])-std_hat);
end


% Sensitivity Curve for median absolute deviation
% that does not coverge to IF
SC_mad = zeros(size(Δ_x));
std_hat = madn(x_N_minus1);
for ii = 1:length(Δ_x)
    SC_mad(ii) = N*(madn([x_N_minus1; Δ_x(ii)])-std_hat);
end

% Sensitivity Curve for mean absolute deviation
% around the median
SC_mead = zeros(size(Δ_x));
std_hat = mean(abs(x_N_minus1-median(x_N_minus1)));
for ii = 1:length(Δ_x)
    SC_mead(ii) = N*(mean(abs([x_N_minus1; Δ_x(ii)]-median(x_N_minus1)))-std_hat);
end

% Sensitivity Curve for Huber's scale estimate
c =  1.3415;
SC_hub = zeros(size(Δ_x));
std_hat = MscaleHUB(x_N_minus1,c);
for ii = 1:length(Δ_x)
SC_hub(ii) = N*(MscaleHUB([x_N_minus1; Δ_x(ii)],c)-std_hat);
end

% Sensitivity Curve for Tukey's scale estimate
c = 4.68;
SC_tuk = zeros(size(Δ_x));
std_hat = MscaleTUK(x_N_minus1,c);
for ii = 1:length(Δ_x)
SC_tuk(ii) = N*(MscaleTUK([x_N_minus1; Δ_x(ii)],c)-std_hat);
end

figure,
set(gca,'FontSize',18)
hold on
plot(Δ_x,SC_std-min(SC_std),'linewidth',2)
plot(Δ_x,SC_mead-min(SC_mead),'linewidth',2)
plot(Δ_x,SC_hub-min(SC_hub),'linewidth',2)
plot(Δ_x,SC_tuk-min(SC_tuk),'linewidth',2)
grid on
xlabel('Outlier value');
ylabel('Sensitivity curve');
legend('Standard deviation', 'madn', 'Huber M', 'Tukey M');
```

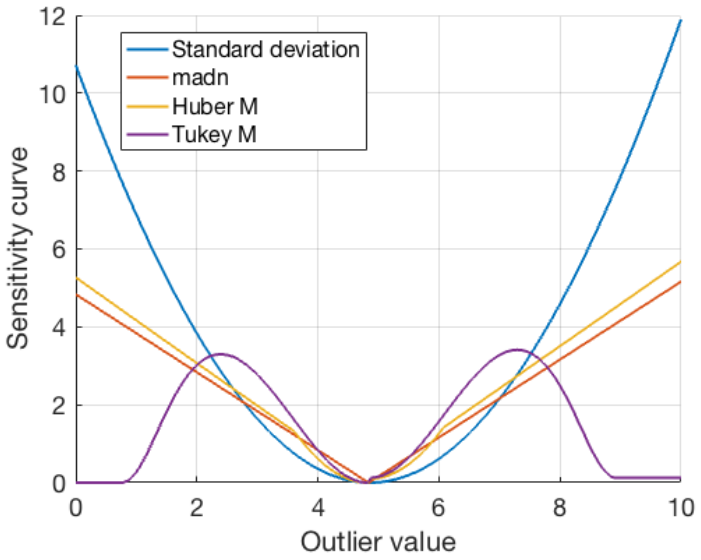


Figure 2: Sensitivity curves for scale estimation.

2 Robust and Penalized Estimation in the Linear Regression Model

2.1 `wmed` computes the weighted (univariate) median

`wmed` computes the weighted median $\hat{\beta}$ based on the data $y_i \in \mathbb{F}$ (= $\mathbb{R}$, or, $\mathbb{C}$) and weights $w_i \in [0, \infty)$, $i = 1, \ldots, N$, where

$$\underset{\beta \in \mathbb{F}}{\text{minimize}} \sum_{i=1}^{N} |y_i - \beta| \, w_i. \tag{8}$$

The function `wmed` is used for solving the LAD criterion in the simple linear regression case.

2.1.1 Syntax

```
function [beta, iter, converged] = wmed(y,w,verbose)
% wmed computes the weighted median for data y and weights w, i.e.
%
% beta = arg min_b  SUM_i | y_i - b | * w_i
%
% INPUTS:
%       y : (numeric) data given (real or complex)
%       w : (nubmer) positive real-valued weights. Inputs need to be of
%           same length
%       verbose: (logical) true of false (default). Set as true if you wish
%       to see convergence as iterations evolve
% OUTPUTS:
%       beta: (numeric) weighted median
%       converged: (logical) flag of convergence (in the complex-valued
%       data case)
%       iter: (numeric) the number of iterations (complex-valued case)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.1.2 Example

```
rng(0);
% Example 1: real data
y = 10*randn(1,100);  % data from N(0,10^2);
w = rand(1,100);      % weights from Unif(0,1);
wmed(y,w)

% Example 2: complex-valued data:
y = 10*(randn(1,100) + 1i*randn(1,100)); % data from complex Gaussian N(0,10^2);
wmed(y,w,true)
```

2.2 `elemfits` computes the elemental fits of simple linear regression.

`elemfits` computes the elemental fits in the simple linear regression, i.e., the intercept and slope,

$$b_{0,ij} = \frac{x_j y_i - x_i y_j}{x_j - x_i} \qquad \text{and} \qquad b_{1,ij} = \frac{y_i - y_j}{x_i - x_j},$$

of a line that passes through a data point $(x_i, y_i) \in \mathbb{R}^2$ and $(x_i, y_j) \in \mathbb{R}^2$, $1 \leq i < j \leq N$. There are $\binom{N}{2} = N(N-1)/2$ such elemental fits. The algorithm also computes the respective weight $w_{ij} = |x_i - x_j|$ of each elemental fit. The data $\{(y_i, x_i)\}_{i=1}^{N}$ is assumed to be real-valued.

2.2.1 Syntax

```
function [beta, w] = elemfits(y,x)
% [beta, w] = elemfits(y,x)
% elemfits compute the Nx(N-1)/2 elemental fits, i.e., intercepts b_{0,ij}
% and slopes b_{1,ij}, that define a line y = b_0+b_1 x that passes through
% the data points (x_i,y_i) and (x_j,y_j), i<j, where i, j in {1, ..., N}
% and the respective weights | x_i - x_j |
% INPUTS:
%    y : (numeric) N x 1 vector of real-valued outputs (response vector)
%    x : (numeric) N x 1 vector of inputs (feature vector)
% OUTPUTS:
%  beta: (numeric) N*(N-1)/2 matrix of elemental fits
%     w: (numeric) N*(N-1)/2 matrix of weights
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.3 `ladreg` computes the LAD estimate of regression

`ladreg` computes the LAD estimate of regression, $\hat{\boldsymbol{\beta}}_{\text{LAD}}$, which solves the optimization problem

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \, \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_1 \tag{9}$$

where $\mathbf{y}$ and $\mathbf{X}$ are given response (output) and feature (input) matrix which can be real- or complex-valued. `ladreg` uses the IRWLS algorithm when number of features is more than one and a weighted median in the simple linear regression case.

2.3.1 Syntax

```
function [b1, iter] = ladreg(y,X,intcpt,b0,printitn)
% [b1, iter] = ladreg(y,X,intcpt,...)
% ladreg computes the LAD regression estimate
% INPUTS:
%         y: numeric response N x 1 vector (real/complex)
%         X: numeric feature  N x p matrix (real/complex)
%    intcpt: (logical) flag to indicate if intercept is in the model
%        b0: numeric optional initial start of the regression vector for
%            IRWLS algorithm. If not given, we use LSE (when p>1).
%  printitn: print iteration number (default = 0, no printing) and
%            other details
% OUTPUTS:
%        b1: (numberic) the regression coefficient vector
%      iter: (numeric) # of iterations (given when IRWLS algorithm is used)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.3.2 Dependencies

`ladlasso`

2.3.3 Example

```
rng(0);
N = 100;
X = (1/sqrt(2))*(randn(100,1) + 1i*randn(100,1));   % feature vector N(0,1)
e = (1/sqrt(2*4))*(randn(100,1) + 1i*randn(100,1)); % N(0,1/4) noise
y = ones(N,1)*(1+1i) + X*(1+1i) + e;  %response
ladreg(y,X,true,[],1) % compute the LAD estimate for model with intercept
```

2.4 `rladreg` computes the rank LAD estimate of regression

`rladreg` computes the rank LAD estimate of regression, $\hat{\boldsymbol{\beta}}_{\mathrm{R}}$, which solves the optimization problem

$$\underset{\boldsymbol{\beta}\in\mathbb{F}^p}{\text{minimize}} \sum_{i<j} |(y_i - y_j) - (\mathbf{x}_{[i]} - \mathbf{x}_{[j]})^\top \boldsymbol{\beta}|,$$

where $\mathbf{y}$ and $\mathbf{X}$ are given response (output) and feature (input) matrix which can be real- or complex-valued. `rladreg` uses the IRWLS algorithm when the number of features is more than one and a weighted median in the simple linear regression case.

2.4.1 Syntax

```
function [b1,iter] = rladreg(y,X,b0,printitn)
% [b1, iter] = rladreg(y,X,...)
% ladreg computes the LAD regression estimate
% INPUTS:
%         y: numeric response N x 1 vector (real/complex)
%         X: numeric feature  N x p matrix (real/complex)
%        b0: numeric optional initial start of the regression vector for
%            IRWLS algorithm. If not given, we use LSE (when p>1).
%  printitn: print iteration number (default = 0, no printing) and
%            other details
% OUTPUTS:
%        b1: numeric the regression coefficient vector
%      iter: (numeric) # of iterations (given when IRWLS algorithm is used)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.4.2 Example

```
rng(0);
N = 100;
X = (1/sqrt(2))*(randn(100,1) + 1i*randn(100,1));   % feature vector N(0,1)
e = (1/sqrt(2*4))*(randn(100,1) + 1i*randn(100,1)); % N(0,1/4) noise
y = ones(N,1)*(1+1i) + X*(1+1i) + e;  %response
rladreg(y,X)
```

2.5 `Mreg` computes an M-estimate of regression based on an auxiliary scale.

`Mreg` computes an M-estimate of regression $\hat{\boldsymbol{\beta}}$ defined as a solution to an estimating equation

$$\sum_{i=1}^{N} \psi\left(\frac{y_i - \mathbf{x}_{[i]}^\top \hat{\boldsymbol{\beta}}}{\hat{\sigma}}\right)\mathbf{x}_{[i]}^* = \mathbf{0},$$

where $\psi : \mathbb{F} \to \mathbb{R}_0^+$ is a score function corresponding to either Tukey's of Huber's loss function and $\hat{\sigma}$ is an auxiliary scale estimate. If $\hat{\sigma}$ is not given as input, `Mreg` uses a properly scaled Median Absolute Deviation (MAD) scale estimate of the residuals based on a LAD regression fit. `Mreg` uses the IRWLS algorithm to compute the solution.

2.5.1 Syntax

```
function [b1, sig] = Mreg(y,X,lossfun,b0,verbose)
% [b1, sig] = Mreg(y,X,lossfun,b0)
% Mreg computes the M-estimates of regression using an auxiliary scale
% estimate. It uses the iterative reweighted least squares (IRWLS) algorithm
%
```

```
% INPUTS:
%       y : (numeric) data vector of size N x 1 (output, response vector)
%       X : (numeric) data matrix of size N x p (input, feature matrix)
%           If the model has intercept, then first column of X should be a
%           vector of ones.
% lossfun : (string) either 'huber' or 'tukey' to identify the desired
%           loss function
%      b0 : (numeric) Optional robust initial start (regression vector) of
%           iterations. If not given, we use the LAD regression estimate
%  verbose: (logical) true of false (default). Set as true if you wish
%           to see convergence as iterations evolve.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.5.2 Dependencies

`ladlasso`, `wmed`

2.5.3 Dependencies

`ladreg`, `wtuk`, `whub`

2.5.4 Example

```
rng(0);
N = 100;
X = (1/sqrt(2))*(randn(100,1) + 1i*randn(100,1));   % feature vector N(0,1)
e = (1/sqrt(2*4))*(randn(100,1) + 1i*randn(100,1)); % N(0,1/4) noise
y = ones(N,1)*(1+1i) + X*(1+1i) + e;  %response
Mreg(y,[ones(N,1) X]) % uses Huber's loss as default
Mreg(y,[ones(N,1) X],'tukey',[],true) % use Tukey's loss function and print ...
    iteration convergence
```

2.6 `hubreg` computes the joint M-estimates of regression and scale solving Huber's criterion

`hubreg` computes the joint M-estimates of regression and scale, $(\hat{\boldsymbol{\beta}}, \hat{\sigma})$, that minimize Huber's criterion,

$$\underset{\boldsymbol{\beta},\sigma>0}{\text{minimize}} \left\{ \frac{2N}{\gamma}(\alpha\sigma) + \sum_{i=1}^{N} \rho_{\text{H},c}\left(\frac{y_i - \mathbf{x}_{[i]}^{\top}\boldsymbol{\beta}}{\sigma} \right) \sigma \right\},$$

where $\rho_{\text{H},c}$ is the Huber's loss function, γ is a constant defined as

$$\gamma = \begin{cases} 1, & \text{complex-valued case, } \mathbb{F} = \mathbb{C} \\ 2, & \text{real-valued case, } \mathbb{F} = \mathbb{R}. \end{cases}$$

and α is a consistency factor computed as

$$\alpha = \frac{1}{2} \times \begin{cases} c^2(1 - F_{\chi_2^2}(2c^2)) + F_{\chi_4^2}(2c^2), & \text{complex-valued case, } \mathbb{F} = \mathbb{C} \\ c^2(1 - F_{\chi_1^2}(c^2)) + F_{\chi_3^2}(c^2), & \text{real-valued case, } \mathbb{F} = \mathbb{R}, \end{cases}$$

where $F_{\chi_k^2}$ denotes the cdf of the χ_k^2-distribution and c is the tuning (threshold) constant of Huber's function. `hubreg` computes the solution $(\hat{\boldsymbol{\beta}}, \hat{\sigma})$ using the minimization majorization algorithm detailed in the book.

2.6.1 Syntax

```
function [b1, sig1, iter] = hubreg(y,X,c,sig0,b0,printitn)
% [b1, sig1, iter] = hubreg(y,X,...)
% hubreg computes the joint M-estimates of regression and scale using
% Huber's criterion. Function worsk for both real- and complex-valued data.
%
% INPUTS:
%        y: Numeric data vector of size N x 1 (output, respones)
%        X: Numeric data matrix of size N x p. Each row represents one
%           observation, and each column represents one predictor (feature).
%           If the model has an intercept, then first column needs to be a
%           vector of ones.
%         c: numeric threshold constant of Huber's function
%      sig0: (numeric) initial estimator of scale [default: SQRT(1/(n-p)*RSS)]
%        b0: initial estimator of regression (default: LSE)
%  printitn: print iteration number (default = 0, no printing)
% OUTPUTS:
%       b1: the regression coefficient vector estimate
%     sig1: the estimate of scale
%     iter: the # of iterations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.6.2 Dependencies

`psihub`

2.6.3 Example

```
rng(0);
N = 100;
X = (1/sqrt(2))*(randn(100,1) + 1i*randn(100,1));   % feature vector N(0,1)
e = (1/sqrt(2*4))*(randn(100,1) + 1i*randn(100,1)); % N(0,1/4) noise
y = ones(N,1)*(1+1i) + X*(1+1i) + e;  %response
hubreg(y,[ones(N,1) X])
hubreg(y,[ones(N,1) X],[],[],[],true) % same thing but prints the convergence to screen
```

2.7 `enet` computes the (Lasso and) elastic net

`enet` solves the elastic net (EN) optimization problem for the regression model without intercept,

$$\underset{\boldsymbol{\beta}}{\text{minimize}}\ \frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2+\lambda\Big\{\frac{1}{2}(1-\alpha)\|\boldsymbol{\beta}\|_2^2+\alpha\|\boldsymbol{\beta}\|_1\Big\}, \tag{10}$$

where $\lambda \geq 0$ is the positive tuning or penalty parameter, $\alpha \in [0,1]$ is a user chosen EN tuning parameter and predictors (feature vectors, i.e., the columns of $\mathbf{X}$) are assumed to be standardized to have unit norms. Note that Lasso and ridge regression are special cases when $\alpha = 1$ and $\alpha = 0$ respectively. `enet` uses the cyclic co-ordinate descent (CCD) algorithm as its computational engine. `enet` is an utility function of `enetpath` which computes the solution (regularization) path, i.e., a set of solutions for an array of penalty parameter values. We recommend to use the `enetpath` as the main function for computing the EN.

2.7.1 Syntax

```
function [beta,iter] = enet(y,X,beta,lambda,alpha,verbose)
% [beta,iter] = enet(y,X,beta,lambda,...)
% enet computes the elastic net estimator using the cyclic co-ordinate
% descent (CCD) algorithm.
%
```

```
% INPUTS:
%       y : (numeric) data vector of size N x 1 (output, respones)
%         if the intercept is in the model, then y needs to be centered.
%       X : (numeric) data matrix of size N x p (input, features)
%           Columns are assumed to be standardized (i.e., norm(X(:,j))=1)
%           as well as centered (if intercept is in the model).
%    beta : (numeric) regression vector for initial start for CCD algorithm
%  lambda : (numeric) a postive penalty parameter value
%  alpha  : (numeric) elastic net tuning parameter in the range [0,1]. If
%           not given then use alpha = 1 (Lasso)
% OUTPUTS:
%   b1    : (numberic) the regression coefficient vector
%   iter  : (numeric) # of iterations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.8 `enetpath` computes the Lasso/EN regularization path

`enetpath` computes the elastic net (EN) or Lasso estimator, $\hat{\boldsymbol{\beta}}_{\text{LS}}(\lambda, \alpha)$, which solves

$$\underset{(\beta_0, \boldsymbol{\beta}) \in \mathbb{F} \times \mathbb{F}^p}{\text{minimize}} \ \frac{1}{2}\big\|\mathbf{y} - \beta_0 \mathbf{1} - \mathbf{X}\boldsymbol{\beta}\big\|_2^2 + \lambda\Big\{\frac{1}{2}(1-\alpha)\|\boldsymbol{\beta}\|_2^2 + \alpha\|\boldsymbol{\beta}\|_1\Big\}, \tag{11}$$

over $L+1$ grid points of penalty values,

$$[\lambda] = \{\lambda_0, \ldots, \lambda_L\}, \quad \lambda_0 > \lambda_1 > \ldots > \lambda_L, \tag{12}$$

where the sequence $\{\lambda_i\}$ is monotonically decreasing from λ_0 to $\lambda_L \approx 0$ on a log-scale. Above $\alpha \in [0,1]$ is a user chosen (fixed) EN tuning parameter By default, we use $\lambda_L = \epsilon\lambda_0$, so $\lambda_j = \epsilon^{j/L}\lambda_0 = \epsilon^{1/L}\lambda_{j-1}$ We set $\lambda_0 = \lambda_{\max}(\alpha)$, where $\lambda_{\max}(\alpha)$ denotes the smallest penalty parameter λ for which we obtain all zero solution, i.e., $\hat{\boldsymbol{\beta}}_{\text{LS}}(\lambda_0, \alpha) = \mathbf{0}$, but $\hat{\boldsymbol{\beta}}_{\text{LS}}(\lambda, \alpha) \neq 0$ for $\lambda < \lambda_0$. The Lasso estimator is obtained for $\alpha = 1$.

2.8.1 Syntax

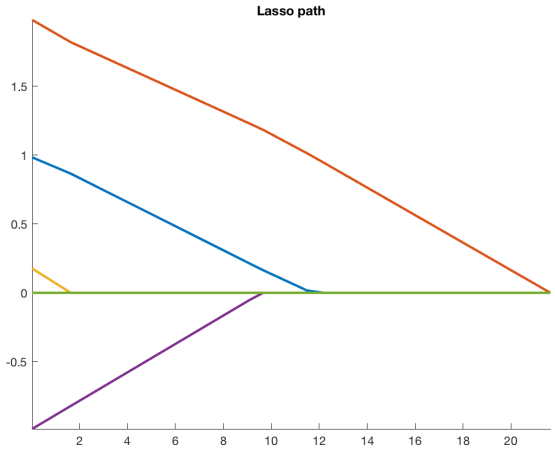
```
function [B, stats] = enetpath(y, X, alpha, L, eps, intcpt,printitn)
% [B, lamgrid, BIC, MSE] = enetpath(y, X,...)
% enethpath computes the elastic net (EN) regularization path (over grid
% of penalty parameter values). Uses pathwise CCD algorithm.
% INPUTS:
%       y : Numeric data vector of size N x 1 (output, respones)
%       X : Nnumeric data matrix of size N x p. Each row represents one
%           observation, and each column represents one predictor (feature).
%   intcpt: Logical flag to indicate if intercept is in the model
%  alpha  : Numeric scalar, elastic net tuning parameter in the range [0,1].
%           If not given then use alpha = 1 (Lasso)
%      eps: Positive scalar, the ratio of the smallest to the
%           largest Lambda value in the grid. Default is eps = 10^-4.
%       L : Positive integer, the number of lambda values EN/Lasso uses.
%           Default is L=100.
% printitn: print iteration number (default = 0, no printing)
% OUTPUTS:
%   B    : Fitted EN/Lasso regression coefficients, a p-by-(L+1) matrix,
%          where p is the number of predictors (columns) in X, and L is
%          the  number of Lambda values. If intercept is in the model, then
%          B is (p+1)-by-(L+1) matrix, with first element the intercept.
%  stats  : structure with following fields:
%           Lambda = lambda parameters in ascending order
%           MSE = Mean squared error (MSE)
%           BIC = Bayesian information criterion values for each lambda
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.8.2 Dependencies

enet

2.8.3 Example

```
rng default % For reproducibility
X = randn(100,5); % design (feature) matrix
% beta has 1 zero coefficient, 1 close to zero, and 3 non-zeros coeff's
beta = [1;2;0.2;-1;0];
beta0 = 1; % intercept
y = beta0*ones(100,1)+X*beta + randn(100,1)*.2; %  response vector
[B, stats] = enetpath(y,X); % computes the Lasso regularization path
figure;  hold on; title('Lasso path');
for ii =1:size(X,2)
    plot(stats.Lambda,B(ii+1,:),'LineWidth',2.0)
end
axis tight
```



2.9 `ladlasso` computes the LAD-Lasso estimate of regression

`ladlasso` computes the LAD-Lasso estimate, $\hat{\boldsymbol{\beta}}_{\text{LAD}}(\lambda)$, which solves the penalized optimization problem

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \, \|\mathbf{y} - \beta_o \mathbf{1} - \mathbf{X}\boldsymbol{\beta}\|_1 + \lambda \|\boldsymbol{\beta}\|_1.$$

`ladlasso` uses the IRWLS algorithm when the number of features is larger than one and a weighted median or elemetal fits approach in the simple linear regression case. It is possible to specify if the intercept is in the model ($\beta_0 \neq 0$) as well as to provide an initial value of regression coefficient vector to start the iterations.

2.9.1 Syntax

```
function [b1, iter] = ladlasso(y,X,lambda,intcpt,b0,printitn)
% [b1, iter] = ladlasso(y,X,lambda,b0,intcpt)
% ladlasso computes the LAD-Lasso regression estimates for given complex-
% or real-valued data.  If number of predictors, p, is larger than one,
% then IRWLS algorithm is used, otherwise a weighted median algorithm
% (N > 200) or elemental fits (N<200).
% INPUTS:
%   y      : (numeric) response N x 1 vector (real/complex)
```

```
%   X      : (numeric) feature  N x p matrix (real/complex)
%   lambda : penalty parameter (≥ 0)
%   b0     : (numeric) Optional initial start (regression vector) of
%            iterations. If not given, we use LSE (when p>1).
%   intcpt : (logical) flag to indicate if intercept is in the model
% printitn : print iteration number (default = 0, no printing)
% OUTPUTS:
%   b1     : (numberic) the regression coefficient vector
%   iter   : (numeric) # of iterations (only when IRWLS algorithm is used)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.9.2 Dependencies

`wmed`, `elemfits`

2.10 `ladlassopath` computes the LAD-Lasso solution path

`ladlassopath` computes the LAD-Lasso solution (regularization) path, that is, it computes $\hat{\boldsymbol{\beta}}_{\text{LAD}}(\lambda)$ over $L+1$ grid points of penalty values:

$$[\lambda] = \{\lambda_0, \ldots, \lambda_L\}, \quad \lambda_0 > \lambda_1 > \ldots > \lambda_L,$$

where the sequence $\{\lambda_i\}$ is monotonically decreasing from λ_0 to $\lambda_L \approx 0$ on a log-scale. By default, we use $\lambda_L = \epsilon\lambda_0$, so $\lambda_j = \epsilon^{j/L}\lambda_0 = \epsilon^{1/L}\lambda_{j-1}$ We set $\lambda_0 = \lambda_{\max}$, where $\lambda_{\max}$ denotes the smallest penalty parameter λ for which we obtain all zero solution, i.e., $\hat{\boldsymbol{\beta}}_{\text{LAD}}(\lambda_0) = \mathbf{0}$. As an approximation of $\lambda_{\max}$, we use

$$\lambda_{\max} \approx \|\mathbf{X}^{\mathsf{H}}\mathsf{sign}(\mathbf{y})\|_\infty.$$

($\|\mathbf{a}\|_\infty = \max_i |a_i|$ denotes the ℓ_∞-norm and $\big[\mathsf{sign}(\mathbf{y})\big]_i = \mathsf{sign}(y_i)$).

2.10.1 Syntax

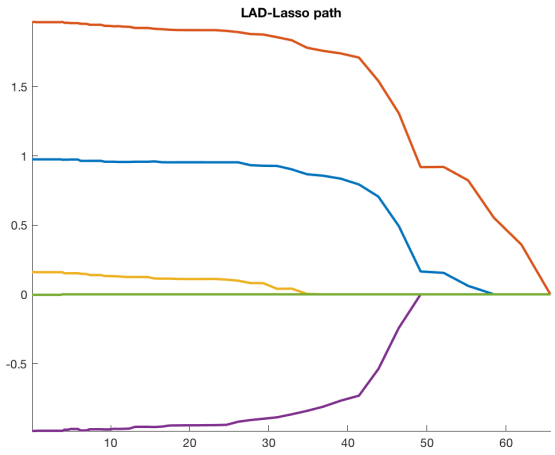
```
function [B, stats] = ladlassopath(y,X,L,eps,intcpt,reltol,printitn)
% [B, stats] = ladlassopath(y, X,...)
% ladlassopath computes the LAD-Lasso regularization path (over grid
% of penalty parameter values). Uses IRWLS algorithm.
% INPUTS:
%       y : Numeric data vector of size N x 1 (output, respones)
%       X : Numeric data matrix of size N x p. Each row represents one
%           observation, and each column represents one predictor (feature).
%   intcpt: Logical (true/false) flag to indicate if intercept is in the
%           regression model
%      eps: Positive scalar, the ratio of the smallest to the
%           largest Lambda value in the grid. Default is eps = 10^-3.
%       L : Positive integer, the number of lambda values EN/Lasso uses.
%           Default is L=120.
%   reltol : Convergence threshold for IRWLS. Terminate when successive
%           estimates differ in L2 norm by a rel. amount less than reltol.
% printitn: print iteration number (default = 0, no printing)
% OUTPUTS:
%   B    : Fitted LAD-Lasso regression coefficients, a p-by-(L+1) matrix,
%          where p is the number of predictors (columns) in X, and L is
%          the  number of Lambda values. If intercept is in the model, then
%          B is (p+1)-by-(L+1) matrix, with first element the intercept.
%  stats  : structure with following fields:
%           Lambda = lambda parameters in ascending order
%           MeAD = Mean Absolute Deviation (MeAD) of the residuals
%           gBIC = generalized Bayesian information criterion (gBIC) value
%                  for each lambda parameter on the grid.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.10.2 Dependencies

`ladlasso`

2.10.3 Example

```
% LAD-LASSO
rng default % For reproducibility
X = randn(100,5); % design (feature) matrix
% beta has 1 zero coefficient, 1 close to zero, and 3 non-zeros coeff's
beta = [1;2;0.2;-1;0];
beta0 = 1; % intercept
y = beta0*ones(100,1)+X*beta + randn(100,1)*.2; %  response vector
[Blad, statslad] = ladlassopath(y, X,[],[],[],1.0e-9,true);
figure;  hold on; title('LAD-Lasso path');
for ii =1:size(X,2) % plotting the solution paths
    plot(statslad.Lambda,Blad(ii+1,:),'LineWidth',2.0)
end
axis tight
```



2.11 `ranklasso` computes the Rank (LAD-)Lasso estimate of regression

`ranklasso` computes the rank (LAD-)Lasso estimate, $\hat{\boldsymbol{\beta}}_{\mathrm{R}}(\lambda)$, which solves the penalized optimization problem

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \sum_{i<j} |(y_i - y_j) - (\mathbf{x}_{[i]} - \mathbf{x}_{[j]})^\top \boldsymbol{\beta}| + \lambda \|\boldsymbol{\beta}\|_1.$$

`ranklasso` uses the IRWLS algorithm when the number of features is larger than one and a weighted median or elemetal fits approach in the simple linear regression case. If the intercept is in the model, so $\beta_0 \neq 0$ (`intcpt` is set as `true`), then $\hat{\beta}_0(\lambda)$ is computed as estimate is

$$\hat{\beta}_{0,\mathrm{R}}(\lambda) = \underset{\beta_0 \in \mathbb{F}}{\arg\min} \sum_{i<j} \left| \frac{\hat{r}_i(\lambda) + \hat{r}_j(\lambda)}{2} - \beta_0 \right|, \tag{13}$$

where $\hat{r}_i(\lambda) = y_i - \mathbf{x}_{[i]}^\top \hat{\boldsymbol{\beta}}_{\mathrm{R}}(\lambda)$ are the obtained residuals of the rank-LAD fit for given penalty value λ. In the real-valued case, (13) is the Hodges–Lehmann (HL) median of residuals but in the complex-valued case, the solution to (13) is best described as the spatial HL median of the residuals and computed using the function `spatmed`. It is also possible to give initial value of regression coefficient vector to start the iterations.

2.11.1 Syntax

```
function [b1,iter] = ranklasso(y,X,lambda,b0,printitn)
% [b1, iter] = ranklasso(y,X,lambda,...)
% ladreg computes the rank (LAD-)regression estimate
% INPUTS:
%       y  : numeric data vector of size N x 1 (output, respones)
%       X  : numeric data matrix of size N x p (input, features)
%   lambda : penalty parameter (≥ 0)
%      b0  : numeric optional initial start (regression vector) of
%            iterations. If not given, we use LSE.
% printitn : print iteration number (default = 0, no printing) and
%            other details
% OUTPUTS:
%   b1     : numeric the regression coefficient vector
%   iter   : (numeric) # of iterations (given when IRWLS algorithm is used)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.12 `ranklassopath` computes the Rank-Lasso regularization path.

`ranklassopath` computes the rank LAD-Lasso regularization path in $L+1$ grid points of penalty values

$$[\lambda] = \{\lambda_0, \ldots, \lambda_L\}, \quad \lambda_0 > \lambda_1 > \ldots > \lambda_L, \tag{14}$$

where the sequence $\{\lambda_i\}$ is monotonically decreasing from λ_0 to $\lambda_L \approx 0$ on a log-scale. By default, we use $\lambda_L = \epsilon\lambda_0$, so $\lambda_j = \epsilon^{j/L}\lambda_0 = \epsilon^{1/L}\lambda_{j-1}$ We set $\lambda_0 = \lambda_{\max}$, where $\lambda_{\max}$ denotes the smallest penalty parameter λ for which we obtain approximately all zero solution, i.e., $\hat{\boldsymbol{\beta}}_{\mathrm{R}}(\lambda_0) = \mathbf{0}$.

2.12.1 Syntax

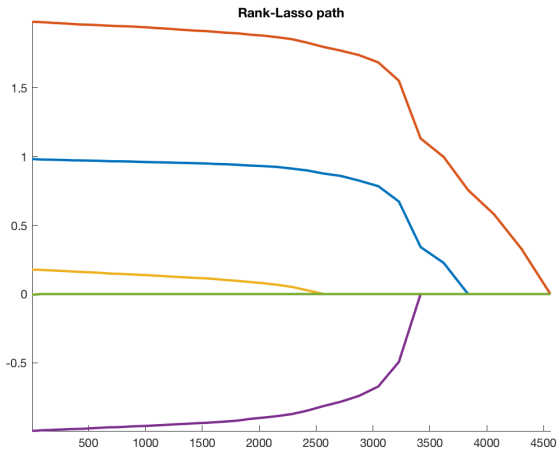
```
function [B, B0, stats] = ranklassopath(y, X, L, eps,reltol,printitn)
% [B, B0, stats] = ranklassopath(y, X,...)
% ranklassopath computes the rank LAD-Lasso regularization path (over grid
% of penalty parameter values). Uses IRWLS algorithm.
% INPUTS:
%       y : Numeric data vector of size N x 1 (output, respones)
%       X : Numeric data matrix of size N x p. Each row represents one
%           observation, and each column represents one predictor (feature).
%       L : Positive integer, the number of lambda values on the grid to be
%           used. The default is L=120.
%     eps: Positive scalar, the ratio of the smallest to the
%           largest Lambda value in the grid. Default is eps = 10^-3.
%   reltol : Convergence threshold for IRWLS. Terminate when successive
%           estimates differ in L2 norm by a rel. amount less than reltol.
% printitn: print iteration number (default = 0, no printing)
% OUTPUTS:
%   B    : Fitted RLAD-Lasso regression coefficients, a p-by-(L+1) matrix,
%          where p is the number of predictors (columns) in X, and L is
%          the  number of Lambda values.
%     B0 : estimates values of intercepts
%  stats : structure with following fields:
%          Lambda = lambda parameters in ascending order
%          GMeAD = Mean Absolute Deviation (MeAD) of the residuals
%          gBIC = generalized Bayesian information criterion (gBIC) value
%                 for each lambda parameter on the grid.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.12.2 Dependencies

`ladlasso`

2.12.3 Example

```
rng default % For reproducibility
X = randn(100,5); % design (feature) matrix
% beta has 1 zero coefficient, 1 close to zero, and 3 non-zeros coeff's
beta = [1;2;0.2;-1;0];
beta0 = 1; % intercept
y = beta0*ones(100,1)+X*beta + randn(100,1)*.2; %  response vector
[Brlad, B0rlad, statsrlad] = ranklassopath(y,X); %
figure;  hold on; title('Rank-Lasso path');
for ii =1:size(X,2)
    plot(statsrlad.Lambda,Brlad(ii,:),'LineWidth',2.0)
end
axis tight;
```



2.13 `rankflasso` computes the Fused Rank-Lasso estimate

`rankflasso` computes the rank fused Lasso estimate, which solves the penalized optimization problem and solved

$$\underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\text{minimize}} \quad \sum_{i<j} |(y_i - y_j) - (\mathbf{x}_{[i]} - \mathbf{x}_{[j]})^\top \boldsymbol{\beta}| + \lambda_1 \|\boldsymbol{\beta}\|_1 + \lambda_2 \sum_{j=2}^{p} |\beta_j - \beta_{j-1}|, \tag{15}$$

where $\lambda_1, \lambda_2 \geq 0$ form a pair of fixed regularization parameters, chosen by the user,

2.13.1 Syntax

```
function [b,iter] = rankflasso(y,X,lambda1,lambda2,b0,printitn)
% b = rankflasso(y,X,lambda1,lambda2,...)
% Computes the rank fused-Lasso regression estimates for given fused
% penalty value lambda_2 and for a range of lambda_1 values
%
% INPUTS:
%   y       : numeric response N x 1 vector (real/complex)
%   X       : numeric feature  N x p matrix (real/complex)
%   lambda1 : positive penalty parameter for the Lasso penalty term
%   lambda2 : positive penalty parameter for the fused Lasso penalty term
%   b0      : numeric optional initial start (regression vector) of
%           iterations. If not given, we use LSE (when p>1).
% printitn : print iteration number (default = 0, no printing)
% OUTPUTS:
%   b      : numeric regression coefficient vector
```

```
%   iter   : positive integer, the number of iterations of IRWLS algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.13.2 Dependencies

`ladlasso`

2.14 `hublasso` computes the M-Lasso estimate.

`hublasso` computes the solutions $(\hat{\boldsymbol{\beta}}, \hat{\sigma})$ to M-lasso estimating equations for Huber's loss function:

$$\langle \mathbf{x}_j, \hat{\mathbf{r}}_\psi \rangle = \lambda \hat{s}_j \quad \text{for } j = 1, \ldots, p \tag{16}$$

$$\hat{\sigma}^2 = \frac{1}{N(2\alpha)} \left\| \hat{\mathbf{r}}_\psi \right\|_2^2 \tag{17}$$

where the $\hat{\mathbf{r}}_\psi$ denotes the pseudo-residual based on Huber's loss function at the solution, where $\mathbf{r}_\psi$ is defined as

$$\mathbf{r}_\psi \equiv \mathbf{r}_\psi(\boldsymbol{\beta}, \sigma) = \psi_{\mathrm{H},c}\left(\frac{\mathbf{y} - \mathbf{X}\boldsymbol{\beta}}{\sigma} \right) \sigma$$

and $\psi_{\mathrm{H},c}$ is the associated score function of Huber's loss function. Above the notation is such that score function $\psi_{\mathrm{H},c}(\cdot)$ acts coordinate-wise to vector $\mathbf{r}/\sigma$, i.e., $[\psi_{\mathrm{H},c}(\mathbf{r}/\sigma)]_i = \psi_{\mathrm{H},c}(r_i/\sigma)$.

2.14.1 Syntax

```
function  [b0, sig0, psires] = hublasso(y,X,c,lambda,b0,sig0,reltol,printitn)
% [B, stats] = hublasso(y, X,c,lambda,b0,sig0,...)
% hublasso computes the M-Lasso estimate for a given penalty parameter
% using Huber's loss function
% INPUTS:
%        y: Numeric data vector of size N x 1 (output, respones)
%        X: Numeric data matrix of size N x p. Each row represents one
%           observation, and each column represents one predictor (feature)
%        c: Threshold constant of Huber's loss function
%   lambda: positive penalty parameter value
%       b0: numeric initial start of the regression vector
%     sig0: numeric positive scalar, initial scale estimate.
%   reltol: Convergence threshold. Terminate when successive
%           estimates differ in L2 norm by a rel. amount less than reltol.
%           Default is 1.0e-5
% printitn: print iteration number (default = 0, no printing)
% OUTPUTS:
%       b0: regression coefficient vector estimate
%     sig0: estimate of the scale
%   psires: pseudoresiduals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.15 `hublassopath` computes the M-Lasso solution path.

`hublassopath` function computes the whole M-Lasso regularization path in $L+1$ grid points of penalty values

$$[\lambda] = \{\lambda_0, \ldots, \lambda_L\}, \quad \lambda_0 > \lambda_1 > \ldots > \lambda_L, \tag{18}$$

where the sequence $\{\lambda_i\}$ is monotonically decreasing from λ_0 to $\lambda_L \approx 0$ on a log-scale. By default, we use $\lambda_L = \epsilon\lambda_0$, so $\lambda_j = \epsilon^{j/L}\lambda_0 = \epsilon^{1/L}\lambda_{j-1}$ We set $\lambda_0 = \lambda_{\max}$, where $\lambda_{\max}$ denotes the smallest penalty parameter λ for which we obtain approximately all zero solution, i.e., $\hat{\boldsymbol{\beta}}(\lambda_0) = \mathbf{0}$.

hublassopath function is the main function to compute the M-Lasso estimates and it has more flexibility than hublasso function: for instance, the feature vectors need not be standardized, the intercept can be in the model, and you do not need to specify the initial starts for regression vector and the scale. In general it is recommended to use hublassopath function as hublasso is primarily an auxiliary function that is used hublassopath.

2.15.1 Syntax

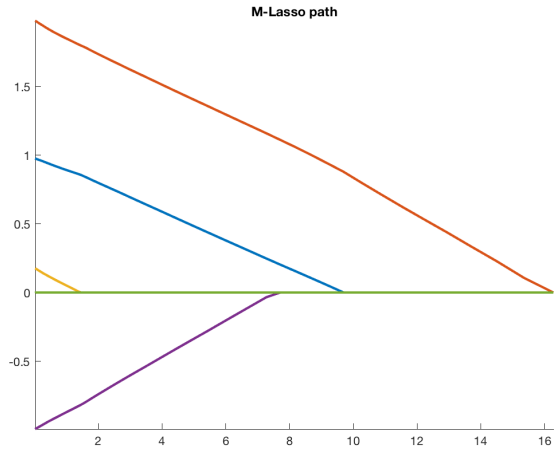
```
function  [B, B0, stats] = hublassopath(y,X,c,L,eps,intcpt,reltol,printitn)
% [B, stats] = hublassopath(y, X,...)
% hublassopath computes the M-Lasso regularization path (over grid
% of penalty parameter values) using Huber's loss function
% INPUTS:
%        y: Numeric data vector of size N x 1 (output, respones)
%        X: Numeric data matrix of size N x p. Each row represents one
%           observation, and each column represents one predictor (feature)
%           columns are standardized to unit length.
%        c: Threshold constant of Huber's loss function (optional;
%            otherwise use defaul falue)
%   intcpt: Logical (true/false) flag to indicate if intercept is in the
%           regression mode. Default is true.
%      eps: Positive scalar, the ratio of the smallest to the
%           largest Lambda value in the grid. Default is eps = 10^-3.
%       L : Positive integer, the number of lambda values EN/Lasso uses.
%           Default is L=120.
%   reltol : Convergence threshold for IRWLS. Terminate when successive
%           estimates differ in L2 norm by a rel. amount less than reltol.
% printitn: print iteration number (default = 0, no printing)
% OUTPUTS:
%   B     : Fitted M-Lasso regression coefficients, a p-by-(L+1) matrix,
%           where p is the number of predictors (columns) in X, and L is
%           the  number of Lambda values. If intercept is in the model, then
%           B is (p+1)-by-(L+1) matrix, with first element the intercept.
%  stats  : structure with following fields:
%           Lambda = lambda parameters in ascending order
%           sigma = estimates of the scale (a (L+1) x 1 vector)
%           gBIC = generalized Bayesian information criterion (gBIC) value
%                 for each lambda parameter on the grid.
% NOTE: this functions assumes model with no intercept. It is recommonded
%       to use hublassopath for computation of the M-Lasso solution. This function ...
    is auxiliary function for the hublassopath function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.15.2 Dependencies

hublasso, hubreg

2.15.3 Example

```
rng default % For reproducibility
X = randn(100,5); % design (feature) matrix
% beta has 1 zero coefficient, 1 close to zero, and 3 non-zeros coeff's
beta = [1;2;0.2;-1;0];
beta0 = 1; % intercept
y = beta0*ones(100,1)+X*beta + randn(100,1)*.2; %  response vector
[Bhub, B0hub, statshub] = hublassopath(y,X); % computes the Lasso regularization path
figure;  hold on; title('M-Lasso path');
for ii =1:size(X,2)
    plot(statshub.Lambda,Bhub(ii,:),'LineWidth',2.0)
end
axis tight
```



2.16 Image denoising example

Running the script `image_example.m`, also listed below, yields Figure 3 and Figure 4. This image denoising example was given in Section 3.4.3 and in Figure 3.5 of the book.

```
clear; clc;
addpath('toolbox');
addpath('toolbox/data/');

%- Read image of sqares (20 x 20 pixels)
load images.mat % contains the vectors y20 (clean data) and y20n (noisy data)
n = numel(y20)
scaledata = @(x) 3*(x - min(x))./(max(x)-min(x));

%-- Plot the the image
figure(1);clf;
subplot(2,2,1)
imagesc(reshape(y20,sqrt(n),sqrt(n)));
colormap gray;
axis square off;
title('original image');

%-- Plot the signal
figure(2); clf;
subplot(2,2,1);
plot(1:n,y20,'k*','MarkerSize',14);
axis tight;
xlabel('$i$','Interpreter','Latex')
ylabel('$\beta_i$','FontSize',20,'Interpreter','Latex')
set(gca,'YTick',[0 1 2 3],'XTick',[1 100 200 300 400]);
set(gca','LineWidth',3,'FontSize',20,'FontName','TimesNewRoman');
title('original signal')

%-- Plot the image + noise
figure(1)
subplot(2,2,2)
imagesc(reshape(y20n,sqrt(n),sqrt(n)));
axis square off;
colormap gray;
title('image + noise');

%-- Plot the noisy (measured) signal
figure(2);
```

```
subplot(2,2,2);
plot(1:n,scaledata(y20n),'k*','MarkerSize',14);
xlabel('$i$','FontSize',16,'Interpreter','Latex')
ylabel('$y_{i}$','FontSize',20,'Interpreter','Latex')
set(gca,'YTick',[0 1 2 3],'XTick',[1 100 200 300 400]);
set(gca,'LineWidth',3,'FontSize',20,'FontName','TimesNewRoman');
title('measured signal')


%-- Compute the Lasso solution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = 20; % grid size
[Blas20n,stats] = enetpath(y20n,eye(n),1,L,10^-3,false);
Blas20n = Blas20n(:,2:end); % get rid of all zeros (first column)
% Choose the best Lasso solution
ero = scaledata(Blas20n) - repmat(y20,1,size(Blas20n,2));
[MSElasso, indx] = min(sum(ero.^2)); MSElasso
Blas = Blas20n(:,indx); % the best Lasso solution
lam_las = stats.Lambda(1+indx); % the best lambda value

figure(1);
subplot(2,2,3);
imagesc(reshape(Blas,20,20));
axis square off;
colormap gray;
title(['Lasso: lambda = ', num2str(lam_las)]);

figure(2);
subplot(2,2,3);
plot(1:n,scaledata(Blas(:)),'k*','MarkerSize',14);
axis tight;
xlabel('$i$','FontSize',16,'Interpreter','Latex')
ylabel('$\hat \beta_{i}$','FontSize',20,'Interpreter','Latex')
set(gca,'YTick',[0 1 2 3],'XTick',[1 100 200 300 400]);
set(gca,'LineWidth',3,'FontSize',20,'FontName','TimesNewRoman');
title('Lasso');

%-- Compute the Rank-FLasso solution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% start with some initial values of lambda1 and lambda2
lambda2 = 340;
lambda1 = 124;
B1 = rankflasso(y20n,eye(n),lambda1,lambda2,Blas,1);
MSE_rank1 = sum((scaledata(B1)- y20).^2)

%-- adjust the parameters
lambda2 = 420;
lambda1 = 35;
B2= rankflasso(double(y20n),eye(20*20),lambda1,lambda2,B1,1);
MSE_rank2 = sum((scaledata(B2)- y20).^2)

figure(1);
subplot(2,2,4)
imagesc(reshape(B2,20,20));
axis off square;
title(['Rank-FLasso: lambda1= ', num2str(lambda1), ' lambda2 = ', num2str(lambda2)]);
colormap gray;

figure(2);
subplot(2,2,4);
plot(1:n,scaledata(B2),'k*','MarkerSize',14);
axis tight;
xlabel('$i$','FontSize',16,'Interpreter','Latex')
ylabel('$\hat \beta_{i}$','FontSize',20,'Interpreter','Latex')
set(gca,'YTick',[0 1 2 3],'XTick',[1 100 200 300 400]);
set(gca,'LineWidth',3,'FontSize',20,'FontName','TimesNewRoman');
```

```
title('Rank-FLasso');
```

2.17 Application Example: Prostate Cancer

Running the code below you obtain Figure 5 and Figure 6. This example was given in Section 3.7 and in Figure 3.6 and Figure 3.7 of the book.

```
clear; clc;
addpath('/examples');

load data/prostate
[n, p] = size(X);
Xone = [ones(n,1) X];
LSE = Xone \ y % Least squares estimate
GRlen = 120;

%-- LASSO
%%%%%%%%%%
[B,stats] = enetpath(y,X,1);
[¬,k] = min(stats.BIC);
blas = B(:,k); % LASSO-BIC solution
bmaxlas = sum(abs(B(2:end,end))); % largest value of || \beta ||_1
%- PLOT
h=figure(1); clf;
subplot(2,2,1);
locs = B(2:end,end);
locs(3) = locs(3)-0.025; % 'age' is too close, so put it down
locs(7) = locs(7)+0.01;  % 'gleason' is too close, so put it up
loc_x = sum(abs(blas(2:end)))/bmaxlas;
prostate_plot_setup(sum(abs(B(2:end,:)))/bmaxlas,B(2:end,:),locs,loc_x,names)

%-- LAD-LASSO
%%%%%%%%%%%%%
reltol = 1.0e-7;
tic;[Blad, statslad] = ladlassopath(y, X,[],[],[],reltol);
[¬,ladind] = min(statslad.gBIC);
blad = Blad(:,ladind); % LAD-Lasso BIC solution
bmaxlad = max(sum(abs(Blad(2:end,:)))); % largest solution || \beta ||_1

%- PLOT
figure(1); subplot(2,2,3);
locs = Blad(2:end,end);
locs(2) = locs(2)+0.02; % lweight up
locs(7) = locs(7)+0.02; % gleason up
locs(3) = locs(3)-0.02; % age down
loc_x = sum(abs(blad(2:end)))/bmaxlad;
prostate_plot_setup(sum(abs(Blad(2:end,:)))/bmaxlad,Blad(2:end,:),locs,loc_x,names)

%-- Rank-LASSO
%%%%%%%%%%%%%%
[Brlad,¬,statsrlad] = ranklassopath(y, X);
[¬,rladind] = min(statsrlad.gBIC);
brlad = Brlad(:,rladind);
bmaxrlad = max(sum(abs(Brlad)));

%- PLOT
figure(1);
subplot(2,2,4)
locs = Brlad(:,end);
locs(7) = locs(7)+0.01; % gleason up
```

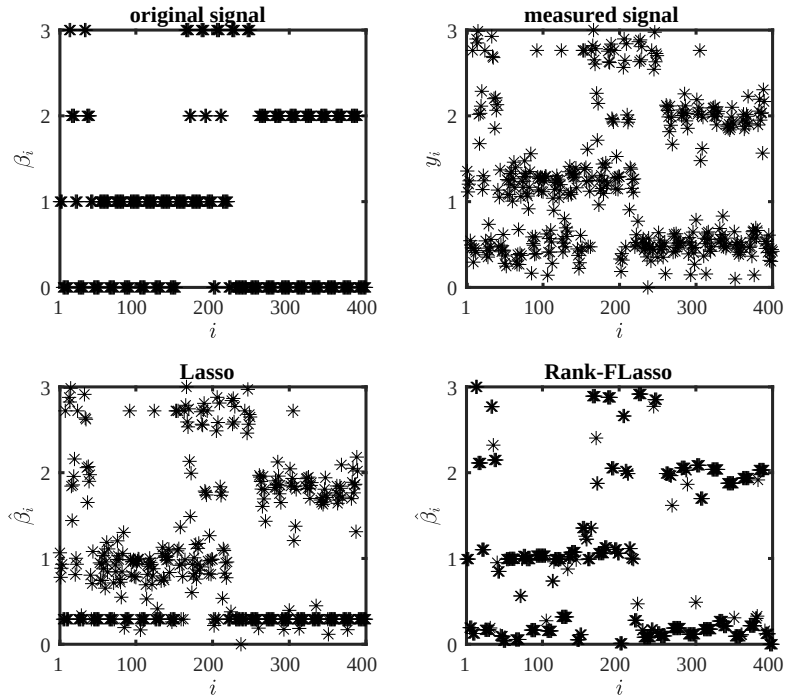


Figure 3: Image denoising example: original, measured noisy, and estimated signals

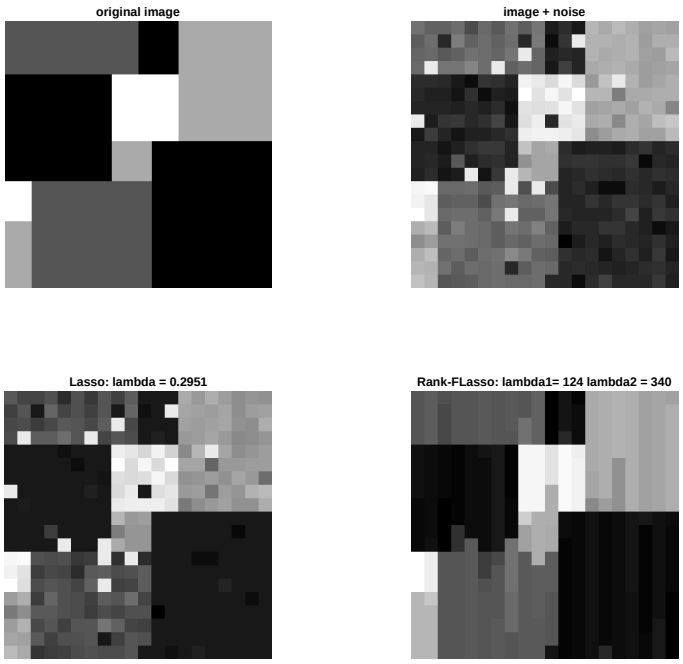


Figure 4: Image denoising example: original, measured noisy, and estimated images

```
locs(3) = locs(3)-0.03; % age down
loc_x = sum(abs(brlad))/bmaxrlad;
prostate_plot_setup(sum(abs(Brlad))/bmaxrlad,Brlad,locs,loc_x,names)

%-- M-LASSO
%%%%%%%%%%%%
[Bhub,¬,statshub] = hublassopath(y,X);
[¬,hubind] = min(statshub.gBIC);
bhub = Bhub(:,hubind);
bmaxhub = max(sum(abs(Bhub)));

%- PLOT
figure(1); subplot(2,2,2);
locs = Bhub(:,end);
loc_x = sum(abs(bhub))/bmaxhub;
locs(3) = locs(3)-0.03; % age down
locs(7) = locs(7)+0.02; % gleason up
prostate_plot_setup(sum(abs(Bhub))/bmaxhub,Bhub,locs,loc_x,names)

%-- OUTLIER --
%%%%%%%%%%%%%%

yout = y;
yout(1) = 55;

%-- LASSO
%%%%%%%%%%%
[Bout,stats2] = enetpath(yout,X,1);
[¬,k] = min(stats2.BIC);
blas_out = Bout(:,k); % LASSO-BIC solution
bmaxlas_out = sum(abs(Bout(2:end,end))); % largest value of || \beta ||_1

%- PLOT
figure(2); clf;
subplot(2,2,1);
locs = Bout(2:end,end);
locs(1) = locs(1)-0.04; % lcacvol down
locs(3) = locs(3)-0.08; % age down
locs(4) = locs(4)+0.06; % lbph up
locs(6) = locs(6)+0.02; % lcp up
locs(8) = locs(8)+0.07; % pgg45 up
loc_x = sum(abs(blas_out(2:end)))/bmaxlas_out;
prostate_plot_setup(sum(abs(Bout(2:end,:)))/bmaxlas_out,Bout(2:end,:),locs,loc_x,names,[])

%-- LAD-LASSO
%%%%%%%%%%%%%%
reltol = 1.0e-7;
tic;[Blad2, statslad2] = ladlassopath(yout,X,[],[],[],reltol);
[¬,ladind2] = min(statslad2.gBIC);
blad2 = Blad2(:,ladind2); % LAD-Lasso BIC solution
bmaxlad2 = max(sum(abs(Blad2(2:end,:)))); % largest solution || \beta ||_1

%- PLOT
figure(2); subplot(2,2,3);
locs = Blad2(2:end,end);
locs(6) = locs(6)-0.04; % lcp
locs(8) = locs(8)+0.02; % pgg45 up
locs(7) = locs(7)+0.02; % gleason up
locs(3) = locs(3)-0.02; % age down
loc_x = sum(abs(blad2(2:end)))/bmaxlad2;
prostate_plot_setup(sum(abs(Blad2(2:end,:)))/bmaxlad2,Blad2(2:end,:),locs,loc_x,names)

%-- Rank-LASSO
%%%%%%%%%%%%%%%
[Brlad2,¬,statsrlad2] = ranklassopath(yout,X);
[¬,rladind2] = min(statsrlad2.gBIC);
```

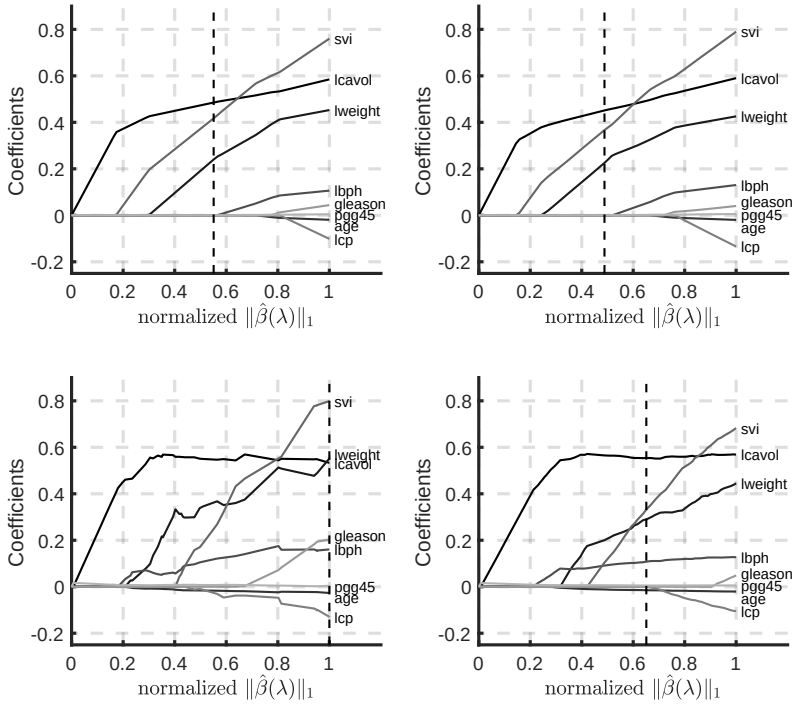


Figure 5: Prostate cancer data: solutions paths (no outlier)

```
brlad2 = Brlad2(:,rladind2);
bmaxrlad2 = max(sum(abs(Brlad2)));

%- PLOT
figure(2);
subplot(2,2,4)
locs = Brlad2(:,end);
locs(7) = locs(7)+0.04; % gleason up
locs(3) = locs(3)-0.025; % age down
locs(6) = locs(6)-0.015; % lcp
loc_x = sum(abs(brlad2))/bmaxrlad2;
prostate_plot_setup(sum(abs(Brlad2))/bmaxrlad2,Brlad2,locs,loc_x,names)

%-- M-LASSO
%%%%%%%%%%%%
[Bhub2,¬,statshub2] = hublassopath(yout,X);
[¬,hubind2] = min(statshub2.gBIC);
bhub2 = Bhub2(:,hubind2);
bmaxhub2 = max(sum(abs(Bhub2)));

%- PLOT
figure(2); subplot(2,2,2);
loc_x = sum(abs(bhub2))/bmaxhub2;
locs = Bhub2(:,end);
locs(3) = locs(3)-0.03; % age down
locs(7) = locs(7)+0.02; % gleason up
prostate_plot_setup(sum(abs(Bhub2))/bmaxhub2,Bhub2,locs,loc_x,names)
```

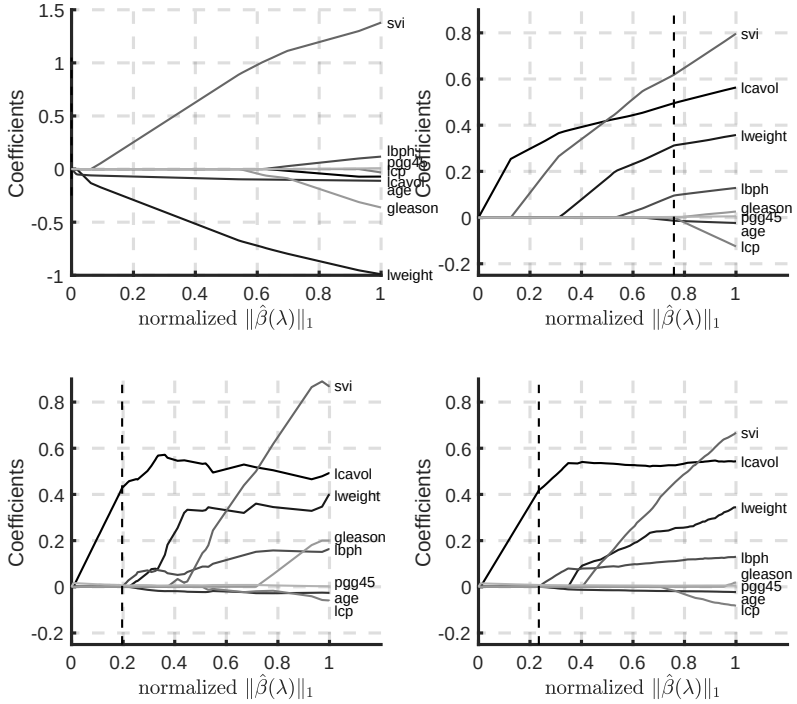


Figure 6: Image denoising example: original, measured noisy, and estimated images

3 Robust Estimation of Location and Scatter (Covariance) Matrix

3.1 `spatmed` computes the spatial median location estimator

`spatmed` computes the spatial median location estimator $\hat{\boldsymbol{\mu}}$ given the data matrix $\mathbf{X}$, that is, it solves the optimization problem

$$\underset{\boldsymbol{\mu}}{\text{minimize}} \sum_{i=1}^{N} \left\|\mathbf{x}_{[i]} - \boldsymbol{\mu}\right\|_2.$$

The data matrix can be real- or complex-valued.

3.1.1 Syntax

```
function  smed = spatmed(X,printitn)
%  Computes the spatial median based on (real or complex) data matrix X.
%  INPUTS:
%         X: Numeric data matrix of size N x p. Each row represents one
%           observation, and each column represents one variable
% printitn : print iteration number (default = 0, no printing)
%
% OUTPUTS:
%      smed: Spatial median estimate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3.1.2 Example

```
X = randn(100,3) + 1i*randn(100,3);
smed0 = spatmed(X)
```

3.2 `signcm` computes the sample spatial sign covariance matrix

`signcm` computes the spatial sign covariance matrix given the data matrix $\mathbf{X}$, defined as

$$\mathbf{C} = \frac{1}{N} \sum_{i=1}^{N} \frac{\mathbf{x}_{[i]} \mathbf{x}_{[i]}^{\mathsf{H}}}{\|\mathbf{x}_{[i]}\|_2^2}$$

or its centered version,

$$\mathbf{C} = \frac{1}{N} \sum_{i=1}^{N} \frac{(\mathbf{x}_{[i]} - \hat{\boldsymbol{\mu}})(\mathbf{x}_{[i]} - \hat{\boldsymbol{\mu}})^{\mathsf{H}}}{\|\mathbf{x}_{[i]} - \hat{\boldsymbol{\mu}}\|_2^2}$$

where $\hat{\boldsymbol{\mu}}$ is the spatial median location estimator. The data matrix can be real- or complex-valued.

3.2.1 Syntax

```
function [C,smed0] = signcm(X,center);
% C = scm(X,center);
% calculates the spatial sign covariance matrix (SCM).
%
% INPUTS:
%         X: Numeric data matrix of size N x p. Each row represents one
%           observation, and each column represents one variable.
```

```
%    center: logical (true/false). If true, then center the data using
%            spatial median. Default is false
% OUTPUTS:
%         C: spatial sign covariance matrix
%     smed0: spatial median (computed only if center = true)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3.2.2 Example

```
X = randn(100,3) + 1i*randn(100,3);
smed0 = signcm(X) % should be close to (1/3)*I
```

3.3 `Mscat` computes the M-estimator of scatter

`Mscat` computes the M-estimator of scatter (covariance) matrix, i.e., solves the optimization problem

$$\hat{\mathbf{\Sigma}} = \arg\min_{\mathbf{\Sigma}\succ 0} \left\{ L(\mathbf{\Sigma}) = \sum_{i=1}^{N} \rho(\mathbf{x}_i^{\mathsf{H}} \mathbf{\Sigma}^{-1} \mathbf{x}_i) - \frac{N}{\gamma} \ln |\mathbf{\Sigma}^{-1}| \right\} \tag{19}$$

where $\rho(t) : \mathbb{R}_0^+ \to \mathbb{R}^+$, referred to as *loss function*, verifies conditions stated in the book. Currently, Huber's, Tyler's and t_ν-loss functions are implemented. Note that `Mscat` can take both real- or complex-valued data.

3.3.1 Syntax

```
function [C,invC,iter,flag] = Mscat(X,loss,losspar,invC,printitn)
% [C,invC,iter,flag] = Mscat(X,loss,...)
% computes M-estimator of scatter matrix for the n x p data matrix X
% using the loss function 'Huber' or 't-loss' and for a given parameter of
% the loss function (i.e., q for Huber's or degrees of freedom v for
% the t-distribution).
%
% Data is assumed to be centered (or the symmetry center parameter = 0)
%
% INPUTS:
%        X: the data matrix with n rows (observations) and p columns.
%     loss: either 'Huber' or 't-loss' or 'Tyler'
%  losspar: parameter of the loss function: q in [0,1) for Huber and
%           d.o.f. v ≥ 0 for t-loss. For Tyler you do not need to specify
%           this value. Parameter q determines the treshold
%           c^2 as the qth quantile of chi-squared distribution with p
%           degrees of freedom distribution (Default q = 0.8). Parameter v
%           is the def.freedom of t-distribution (Default v = 3)
%           if v = 0, then one computes Tyler's M-estimator
%     invC: initial estimate is the inverse scatter matrix (default =
%           inverse of the sample covariance matrix)
% printitn: print iteration number (default = 0, no printing)
% OUTPUTS:
%        C: the M-estimate of scatter using Huber's weights
%     invC: the inverse of C
%     iter: nr of iterations
%     flag: flag (true/false) for convergence
%
% Note: Requires Statistics and Machine Learning Toolbox (calls functions
% chi2cdf and chi2inv in case of Huber's loss function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3.3.2 Example

```
n=100; % sample size
theta=pi/4;
```

```
E = [cos(theta) -sin(theta); sin(theta)  cos(theta)];
L = diag([5 1]);
Sig = E*L*E';
X = randn(n,2)*sqrtm(Sig); % ¬ N_3(0,Sig)
hatSig = Mscat(X,'t-loss',3); % compute t_3-M estimator
```

3.4 Signal Detection Application

Running the script below you obtain Figure 7. This example was given in Section 4.6 and is equivalent to Figure 4.3 in the book.

```
clear;
NR_ITER = 10000;
p = 8;  % dimension
s = exp(1i*(1:p)*pi).'; %  pulse of || p ||^2 = m
alpha = [0.001:0.002:0.04 0.04]; % set of PFA's

lam =  1- alpha.^(1/(p-1));  % = betainv(1-alpha,1,m-1);
nu = 0.5;
nlist = [4*p 8*p 12*p];  % number of secondary dta
ntest = 10; % number of primary data

Pfa1 = zeros(numel(nlist),numel(alpha));
Pfa2 = zeros(numel(nlist),numel(alpha));

Lambda1 = zeros(NR_ITER,ntest);
Lambda2 = zeros(NR_ITER,ntest);

rng default % For reproducibility

for kk = 1:numel(nlist)

    n = nlist(kk);

    for it = 1:NR_ITER

        % Generate covariance matrix
        %l = rand(1,p);
        l = unifrnd(0.1,1,[1 p]);
        P = orth(rand(p,p) + 1i*rand(p,p));
        sig  = p*P*diag(l./sum(l))*P';
        sig(1:(p+1):p^2)=real(sig(1:(p+1):p^2));
        sqrsig = sqrtm(sig);

        % Generate the secondary data and compute the covariance
        x0 = sqrsig*sqrt(1/2)*(randn(p,n) + 1i*randn(p,n)); % ¬ N_p(0,I)
        x = repmat(sqrt(gamrnd(nu,1/nu,1,n)),p,1).*x0 ; % ¬ K_p,v(0,I)
        hsig1 = Mscat(x','t-loss',0);   % compute Tyler's M-estimator
        hsig2 = Mscat(x','Huber',0.8);  % Huber's M-estimator
        B1 = sqrtm(inv(hsig1)); %
        B2 = sqrtm(inv(hsig2)); %

        % Generate primary data from C K_v(0,sig);
        z0 = sqrsig*sqrt(1/2)*(randn(p,ntest)+ 1i*randn(p,ntest));
        z = repmat(sqrt(gamrnd(nu,1/nu,1,ntest)),p,1).*z0 ;% primary data

        % Compute the ADAPTIVE DETECTOR
        v1 = B1*z; q1 = B1*s; q1 = q1/norm(q1);
        v2 = B2*z; q2 = B2*s; q2 = q2/norm(q2);
        v1 = bsxfun(@rdivide, v1, sqrt(sum(v1.*conj(v1))));
        v2 = bsxfun(@rdivide, v2, sqrt(sum(v2.*conj(v2))));
        Lambda1(it,:) = abs(v1'*q1).^2; Lambda2(it,:) = abs(v2'*q2).^2;

        if mod(it,250) ==0, fprintf('. '); end
```

```
    end
    fprintf('\n');
    for al = 1:numel(alpha)
       Pfa1(kk,al) = sum(Lambda1(:) > lam(al))/(NR_ITER*ntest);
       Pfa2(kk,al) = sum(Lambda2(:) > lam(al))/(NR_ITER*ntest);
    end
end

beep;

%-- Plot  Threshold  vs PFA
figure(2); clf
subplot(2,2,1);
plot(lam,alpha,'LineStyle','-','Color','k','LineWidth',2)
axis([min(lam) max(lam) 0 0.08]);  hold on;
plot(lam,Pfa1(3,:),'--',lam,Pfa1(2,:),':',lam,Pfa1(1,:),'-.','LineWidth',2)
set(gca,'YTick',[0 0.01 0.03 0.05 0.07])
ylabel('$P_{\mathrm{FA}}$','Interpreter','Latex')
xlabel('Detection threshold $(\lambda)$','Interpreter','Latex')
hleg=legend('$n=\infty$','$n=96$','$n=64$','$n=32$');
set(hleg,'Interpreter','Latex','FontSize',20)
grid on
set(gca,'LineWidth',3,'FontSize',18);

subplot(2,2,2);
plot(lam,alpha,'LineStyle','-','Color','k','LineWidth',2)
axis([min(lam) max(lam) 0 0.08]);  hold on;
plot(lam,Pfa2(3,:),'--',lam,Pfa2(2,:),':',lam,Pfa2(1,:),'-.','LineWidth',2)
set(gca,'YTick',[0 0.01 0.03 0.05 0.07])
ylabel('$P_{\mathrm{FA}}$','Interpreter','Latex')
xlabel('Detection treshold $(\lambda)$','Interpreter','Latex')
hleg=legend('$n=\infty$','$n=96$','$n=64$','$n=32$');
set(hleg,'Interpreter','Latex','FontSize',20)
grid on
set(gca,'LineWidth',3,'FontSize',18);

%-- Plot the histogram
subplot(2,2,3);
histogram(Lambda1,30,'normalization','pdf')
ylabel('Frequency','FontSize',20,'FontName','Helvetica');
xlabel('$\hat \Lambda$','FontSize',20,'FontName','Helvetica','Interpreter','Latex');
hold on
xval = 0:0.01:1;
plot(xval,betapdf(xval,1,p-1),'k-','LineWidth',2)
axis([0 1 0 max(betapdf(xval,1,p-1))])
set(gca,'LineWidth',3,'FontSize',18);
grid on

%--  Q-Q plot for Tyler's M-estimator for n = 96
lams = sort(Lambda1(:));  % samples
len = numel(lams);
pvals = ((1:len) - (1/2))/len;
qvals = betainv(pvals,1, p-1); % quantiles of exp with mu=1
q99 = betainv(0.99,1,p-1);
q999 = betainv(0.999,1,p-1);
e99 = lams(len-1000); e999 = lams(len-100);

subplot(2,2,4);
plot(qvals,lams,'.','Color',repmat(0.2,1,3),'LineWidth',1.4);
hold on;
plot([0 1],[0 1],'k--','LineWidth',1.4) % straight line
axis([0 0.85 0 0.85])
set(gca,'YTick',0:0.1:0.8,'Xtick',0:0.1:0.8)

ylabel('Empirical quantile','FontSize',20,'FontName','Helvetica');
xlabel('Theorerical quantile','FontSize',20,'FontName','Helvetica');
```

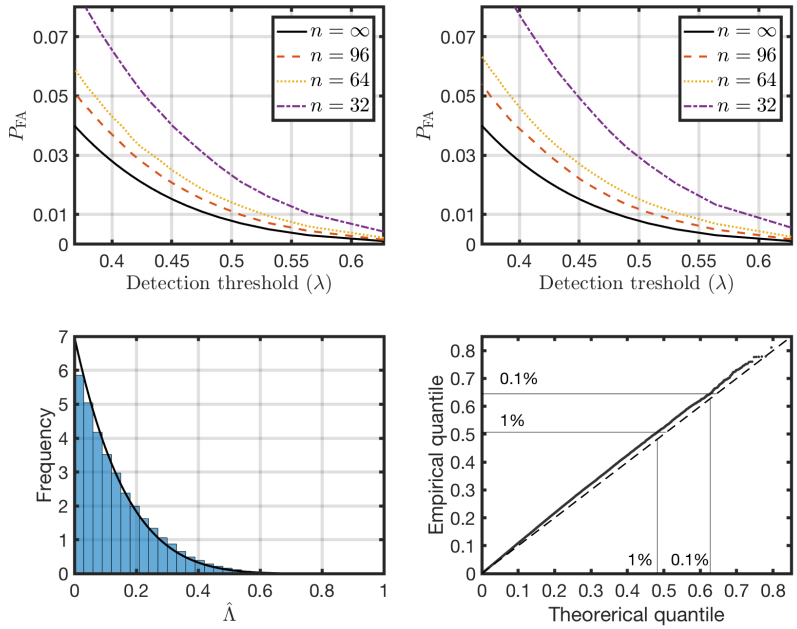


Figure 7: Output of running the code of Section 3.4. Equivalent to Figure 4.3 in the book.

```
plot([q99 q99], [0 q99],'k-'); plot([0 e99], [e99 e99],'k-')
plot([q999 q999], [0 q999],'k-'); plot([0 e999], [e999 e999],'k-')
text(0.05,0.55,'1%','FontSize',16); text(0.05,0.7,'0.1%','FontSize',16);
text(0.4,0.05,'1%','FontSize',16); text(0.52,0.05,'0.1%','FontSize',16);
set(gca,'LineWidth',3,'FontSize',18);
```

4 Robust Filtering

4.1 Extended Kalman Filter (EKF) based Tracking of Mobile User Equipment based on TOA Measurements

Let

$$\mathbf{x}_t = (x_t, y_t, dx_t/dt, dy_t/dt)^\top \tag{20}$$

be the unknown state vector that describes the positions and velocities of an agent in a two-dimensional plane; the agent being consistent with mobile user equipment. Furthermore, let the movement of the agent be described by the following nonlinear discrete-time statistical state equation

$$\mathbf{x}_t = \mathbf{F}\mathbf{x}_{t-1} + \mathbf{G}\mathbf{v}_{t-1} \tag{21}$$

and measurement equation

$$\mathbf{y}_t = \mathbf{h}(\mathbf{x}_t) + \mathbf{w}_t. \tag{22}$$

Here,

$$\mathbf{F} = \begin{pmatrix} \mathbf{I}_2 & \Delta t \mathbf{I}_2 \\ \mathbf{0}_2 & \mathbf{I}_2 \end{pmatrix}$$

models the (linear) state transitions, which in this example are constant over time, namely $\mathbf{F}_t \equiv \mathbf{F}$. The matrix

$$\mathbf{G} = \begin{pmatrix} \Delta t^2/2 \cdot \mathbf{I}_2 \\ \Delta t \cdot \mathbf{I}_2 \end{pmatrix}$$

with Δt denoting the sampling period, maps the accelerations to the position changes of the agent. The uncertainty in the motion model defined by (21) is modeled by the zero-mean white Gaussian random process $\mathbf{v}_t = (v_1(t), v_2(t))^\top \sim \mathcal{N}(\mathbf{0}_2, \mathbf{Q}_t)$.

Eq. (22) relates the observations at the fixed terminals (base stations, anchors) to the state vector $\mathbf{x}_t$ describing the positions and velocities of the agent via the nonlinear function $\mathbf{h}(\cdot) \in \mathbb{R}^{q\times 1}$, where q in this case is the number of anchors. Herein, $\mathbf{h}(\cdot)$ depends on the measured signal parameter. For example, for time-of-arrival (TOA) based tracking, at each anchor $m = 1, \ldots, q$

$$h_m(\boldsymbol{\beta}_t) = \sqrt{(x_{\mathrm{UE},t} - x_{\mathrm{BS},m})^2 + (y_{\mathrm{UE},t} - y_{\mathrm{BS},m})^2} \tag{23}$$

with $\boldsymbol{\beta}_t = (x_{\mathrm{UE},t}, y_{\mathrm{UE},t})^\top$ denoting the unknown positions of the agent at time instant t and $(x_{\mathrm{BS},m}, y_{\mathrm{BS},m})^\top$ being the fixed (and known) position of the mth anchor. In this example, we assume that all measurements are time-of-arrival measurements that are consistent with (23) so that $\mathbf{h}(\cdot) \equiv \mathbf{h}_m(\cdot)$.

4.2 `ekf_toa` Computes the Extended Kalman Filter (EKF) for the Tracking of Mobile User Equipment based on TOA Measurements

The extended Kalman filter algorithm is summarized in Algorithm 3.

The EKF linearizes $\mathbf{h}_t(\mathbf{x}) = (h_1(\mathbf{x}), \ldots, h_q(\mathbf{x}))^\top$ around the predicted state estimate:

$$\mathbf{h}_t(\mathbf{x}_t) \approx \mathbf{h}_t(\mathbf{x}_{t|t-1}) + \mathbf{J}_\mathbf{h}(\mathbf{x}_{t|t-1})(\mathbf{x}_t - \mathbf{x}_{t|t-1}), \tag{29}$$

where

$$\mathbf{J}_\mathbf{h}(\mathbf{x}_{t|t-1}) = \left.\frac{\partial \mathbf{h}_t(\mathbf{x}_t)}{\partial \mathbf{x}_t}\right|_{\mathbf{x}_t = \mathbf{x}_{t|t-1}} \tag{30}$$

Algorithm 3: This algorithm computes the standard extended Kalman filter algorithm

input : $\mathbf{f}_t(\cdot) \in \mathbb{R}^{p\times p}$, $\mathbf{h}_t(\cdot) \in \mathbb{R}^{q\times p}$, $\mathbf{y}_t \in \mathbb{R}^{q\times 1}$, $\mathbf{Q}_t \in \mathbb{R}^{p\times p}$, $\mathbf{R}_t \in \mathbb{R}^{q\times q}$

1 **initialization**: $\mathbf{x}_{0|0}$, $\mathbf{\Sigma}_{0|0}$

2 **prediction**:

$$\mathbf{x}_{t|t-1} \approx \mathbf{f}_t(\mathbf{x}_{t-1|t-1}) \tag{24}$$

$$\mathbf{\Sigma}_{t|t-1} = \mathbf{J}_{\mathbf{f}}(\mathbf{x}_{t-1|t-1})\mathbf{\Sigma}_{t-1|t-1}\mathbf{J}_{\mathbf{f}}^{\top}(\mathbf{x}_{t-1|t-1}) + \mathbf{Q}_t \tag{25}$$

3 **correction**:

$$\mathbf{x}_{t|t} \approx \mathbf{x}_{t|t-1} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{h}(\mathbf{x}_{t|t-1})) \tag{26}$$

$$\mathbf{\Sigma}_{t|t} = \left(\mathbf{I}_p - \mathbf{K}_t\mathbf{J}_{\mathbf{h}}(\mathbf{x}_{t|t-1})\right)\mathbf{\Sigma}_{t|t-1} \tag{27}$$

where

$$\mathbf{K}_t = \mathbf{\Sigma}_{t|t-1}\mathbf{J}_{\mathbf{h}}^{\top}(\mathbf{x}_{t|t-1})\left(\mathbf{J}_{\mathbf{h}}(\mathbf{x}_{t|t-1})\mathbf{\Sigma}_{t|t-1}\mathbf{J}_{\mathbf{h}}^{\top}(\mathbf{x}_{t|t-1}) + \mathbf{R}_t\right)^{-1} \tag{28}$$

is the extended Kalman filter gain and $\mathbf{\Sigma}_{t|t-1}, \mathbf{\Sigma}_{t|t}$ are the state prediction error and correction error covariance matrices, respectively.

is the Jacobian matrix of partial derivatives

$$\mathbf{J}_{\mathbf{h}}(\mathbf{x}) = \begin{pmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_q}{\partial x_1} & \frac{\partial h_q}{\partial x_2} & \cdots & \frac{\partial h_q}{\partial x_p} \end{pmatrix} \in \mathbb{R}^{q\times p} \tag{31}$$

evaluated at the predicted state estimate $\mathbf{x}_{t|t-1}$.

Similarly, to linearize $\mathbf{f}_t(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_p(\mathbf{x}))^{\top}$, the extended Kalman filter uses a first-order Taylor series expansion around $\mathbf{x}_{t-1|t-1}$

$$\mathbf{f}_t(\mathbf{x}_{t-1}) \approx \mathbf{f}_t(\mathbf{x}_{t-1|t-1}) + \mathbf{J}_{\mathbf{f}}(\mathbf{x}_{t-1|t-1}), \tag{32}$$

where $\mathbf{J}_{\mathbf{f}}(\mathbf{x})$ is the Jakobian matrix of partial derivatives

$$\mathbf{J}_{\mathbf{f}}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_p}{\partial x_1} & \frac{\partial f_p}{\partial x_2} & \cdots & \frac{\partial f_p}{\partial x_p} \end{pmatrix} \in \mathbb{R}^{p\times p}. \tag{33}$$

4.2.1 Syntax

```
function [th_hat, P_min, P, parameter] = ekf_toa(r_ges,theta_init,BS,parameter)
% EKF for tracking with time-of-arrival (ToA) estimates.
%
%% INPUTS:
% r_ges:    measured distances (M x N)
% theta_init:  initial state estimate
% BS:   base station positions
%
%% OUTPUTS:
% th_hat:             state estimates
% P_min:              apriori covariance
% P:                  aposteriori covariance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4.2.2 Dependencies

4.2.3 Example

This implements one Monte Carlo run for the EKF with the parameters given in the example `EKF_ToA_Tracking.m`

```
% set the parameters for data generation and for trackers
set_parameters_book; parameter.numbermc = 1;
% generate measurements
parameter = create_environment_book(parameter,parameter.start,parameter.sigma_v);
% generate random starting point
randnvector = [ parameter.initial_sigma(1)*randn parameter.initial_sigma(2)*randn ...
    parameter.initial_sigma(3)*randn parameter.initial_sigma(4)*randn]';
theta_init = parameter.start + randnvector;
% estimate positions using extended Kalman filter
[ekf_th(ii,:,:)]    = ekf_toa(parameter.measureddistances,theta_init,parameter.BS,ekf);
```

4.3 `ekf_toa_robust` computes the M-estimation-based extended Kalman filter algorithm for the tracking of a mobile agent.

`ekf_toa_robust` implements a robust extended Kalman filter for mixed LOS/NLOS conditions, where a batch-mode linear regression estimation technique is applied. Starting from the linear state equation (21), and linearizing the nonlinear vector function $\mathbf{h}(\mathbf{x}_t)$ in the measurement equation (22) around the predicted state estimate $\mathbf{x}_{t|t-1}$ based on (29), yields

$$\mathbf{y}_t \approx \mathbf{h}(\mathbf{x}_{t|t-1}) + \mathbf{J}_\mathbf{h}(\mathbf{x}_{t|t-1})(\mathbf{x}_t - \mathbf{x}_{t|t-1}) + \mathbf{w}_t, \tag{34}$$

where $\mathbf{J}_\mathbf{h}(\mathbf{x})$ is the Jacobian matrix, which, from (31) and (20), takes the form

$$\mathbf{J}_\mathbf{h}(\mathbf{x}_{t|t-1}) = \begin{pmatrix} \frac{dh_1(\mathbf{x}_t)}{dx_t}\Big|_{\mathbf{x}_t=\mathbf{x}_{t|t-1}} & \frac{dh_1(\mathbf{x}_t)}{dy_t}\Big|_{\mathbf{x}_t=\mathbf{x}_{t|t-1}} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \frac{dh_q(\mathbf{x}_t)}{dx_t}\Big|_{\mathbf{x}_t=\mathbf{x}_{t|t-1}} & \frac{dh_q(\mathbf{x}_t)}{dy_t}\Big|_{\mathbf{x}_t=\mathbf{x}_{t|t-1}} & 0 & 0 \end{pmatrix}.$$

The linear state-space model defined by (21) and (22) can be rewritten as a linear regression as follows

$$\begin{pmatrix} \mathbf{F}\mathbf{x}_{t-1|t-1} \\ \mathbf{y}_t - \mathbf{h}(\mathbf{x}_{t|t-1}) + \mathbf{J}_\mathbf{h}(\mathbf{x}_{t|t-1})\mathbf{x}_{t|t-1} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_p \\ \mathbf{J}_\mathbf{h}(\mathbf{x}_{t|t-1}) \end{pmatrix} \mathbf{x}_t + \mathbf{e_t} \tag{35}$$

where

$$\mathbf{e_t} = \begin{pmatrix} -\mathbf{F}(\mathbf{x}_{t-1} - \mathbf{x}_{t-1|t-1}) + \mathbf{G}\mathbf{v}_{t-1} \\ \mathbf{w}_t \end{pmatrix}$$

with

$$\mathsf{E}\left[\mathbf{e}_t\mathbf{e}_t^\top\right] = \begin{pmatrix} \mathbf{\Sigma}_{t|t-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_t \end{pmatrix} = \mathbf{L}_t\mathbf{L}_t^\top$$

and

$$\mathbf{\Sigma}_{t|t-1} = \mathbf{F}\mathbf{\Sigma}_{t-1|t-1}\mathbf{F}^\top + \mathbf{G}\mathbf{Q}_t\mathbf{G}^\top.$$

Multiplying the inverse of the Choleski decomposition $\mathbf{L}_t$ to the left of (35) results in a linear regression system

$$\tilde{\mathbf{y}}_t = \tilde{\mathbf{X}}_t\boldsymbol{\beta}_t + \tilde{\mathbf{v}}_t, \tag{36}$$

where

$$\tilde{\mathbf{y}}_t = \mathbf{L}_t^{-1} \begin{pmatrix} \mathbf{x}_{t|t-1} \\ \mathbf{y}_t - \mathbf{h}(\mathbf{x}_{t|t-1}) + \mathbf{J}_{\mathbf{h}}(\mathbf{x}_{t|t-1})\mathbf{x}_{t|t-1} \end{pmatrix},$$
$$\tilde{\mathbf{X}}_t = \mathbf{L}_t^{-1} \begin{pmatrix} \mathbf{I}_p \\ \mathbf{J}_{\mathbf{h}}(\mathbf{x}_{t|t-1}) \end{pmatrix}, \tag{37}$$
$$\boldsymbol{\beta}_t = \mathbf{x}_t,$$

and

$$\tilde{\mathbf{w}}_t = \mathbf{L}_t^{-1}\mathbf{e}_t.$$

By recognizing the equivalence of (36) and the linear regression model, a robust linear regression estimator can be used to estimate $\mathbf{x}_t$, i.e., to track the agent given the TOA measurements. The function `ekf_toa_robust` uses regression M-estimates with a smoothed assymetric tanh score function, because the NLOS outliers introduce a positive bias, i.e., an assymetric error distribution. Algorithm 4 summarizes the M-estimation-based extended Kalman filter algorithm.

Algorithm 4: This algorithm computes the M-estimation-based extended Kalman filter algorithm for the tracking of a mobile agent.

input : $\mathbf{F} \in \mathbb{R}^{p\times p}$, $\mathbf{G} \in \mathbb{R}^{p\times 2}$, $\mathbf{h}_t(\cdot) \in \mathbb{R}^{q\times p}$, $\mathbf{y}_t \in \mathbb{R}^{q\times 1}$, $\mathbf{Q}_t \in \mathbb{R}^{p\times p}$, $\mathbf{R}_t \in \mathbb{R}^{q\times q}$
output : $\mathbf{x}_{t|t}$
initialize: $\mathbf{x}_{0|0}$, $\boldsymbol{\Sigma}_{0|0}$

1 **prediction**:

$$\mathbf{x}_{t|t-1} \approx \mathbf{F}(\mathbf{x}_{t-1|t-1}) \tag{38}$$
$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}\boldsymbol{\Sigma}_{t-1|t-1}\mathbf{F}^\top(\mathbf{x}_{t-1|t-1}) + \mathbf{G}\mathbf{Q}_t\mathbf{G}^\top \tag{39}$$

2 **correction**:
3 Using `m_param_est`, compute the M-estimate with assymetric tanh score function based on the equivalent linear regression model (36) to obtain $\mathbf{x}_{t|t}$.
4 Approximate the correction error covariance matrix by

$$\boldsymbol{\Sigma}_{t|t} \approx (\tilde{\mathbf{X}}_t^\top \tilde{\mathbf{X}}_t)^{-1}, \tag{40}$$

where $\tilde{\mathbf{X}}_t$ is given in (37).

4.3.1 Syntax

```
function [th_hat, P_min, P, numberit, parameter] = ...
    ekf_toa_robust(r_ges,theta_init,BS,parameter)
% The function computes the robust EKF with ToA estimates based on M-estimation.
%
%% INPUTS:
% r_ges:    measured distances (M x N)
% theta_init:  initial state estimate
% BS:   base station positions
%
%% OUTPUTS:
% th_hat:             state estimates
% P_min:              apriori covariance
% P:                  aposteriori covariance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4.3.2 Dependencies

m_param_est

4.3.3 Example

This implements one Monte Carlo run for the EKF with the parameters given in the example EKF_ToA_Tracking.m

```
% set the parameters for data generation and for trackers
set_parameters_book; parameter.numbermc = 1;
% generate measurements
parameter = create_environment_book(parameter,parameter.start,parameter.sigma_v);
% generate random starting point
randnvector = [ parameter.initial_sigma(1)*randn parameter.initial_sigma(2)*randn ...
    parameter.initial_sigma(3)*randn parameter.initial_sigma(4)*randn]';
theta_init = parameter.start + randnvector;
% estimate positions using robust extended Kalman filter
    [ekf_Hc(ii,:,:)]   = ...
        ekf_toa_robust(parameter.measureddistances,theta_init,parameter.BS,rekf);
```

4.4 TOA-based Tracking of a Mobile Agent in a mixed LOS/NLOS Environment.

This example compares the performance of the M-estimation-based extended Kalman filter (REKF), as implemented in ekf_toa_robust to that of the standard extended Kalman filter (EKF) as implemented in ekf_toa, for a tracking problem in a mixed LOS/NLOS environment.

Running the code below you obtain Figure 8, Figure 9, and Figure 10. This example was given in Section 7.4 of the book.

```
%%  created by Michael Muma
%   version 30 May 2018
%   when using code, cite our work:
%
%   "Robust Statistics for Signal Processing"
%   Zoubir, A.M. and Koivunen, V. and Ollila, E. and Muma, M.
%   Cambridge University Press, 2018.
%
%   and
%
%  "Robust Tracking and Geolocation for Wireless Networks in NLOS Environments."
%   Hammes, U., Wolsztynski, E., and Zoubir, A.M.
%   IEEE Journal on Selected Topics in Signal Processing, 3(5), 889-901, 2009.
%
%
%   The example uses functions that were written by Ulrich Hammes, Signal ...
    Processing Group, TU Darmstadt, February 2009
%
%

clear all;

% set the parameters for data generation and for trackers
set_parameters_book

tic
for ii=1:parameter.mc
```

```
    parameter.numbermc = ii;

    % generate measurements
    parameter = create_environment_book(parameter,parameter.start,parameter.sigma_v);

    % generate random starting point
    randnvector = [ parameter.initial_sigma(1)*randn ...
        parameter.initial_sigma(2)*randn parameter.initial_sigma(3)*randn ...
        parameter.initial_sigma(4)*randn]';
    theta_init = parameter.start + randnvector;

    % estimate positions using (robust) extended Kalman filter
    [ekf_th(ii,:,:)]   = ...
        ekf_toa(parameter.measureddistances,theta_init,parameter.BS,ekf);
    [ekf_Hc(ii,:,:)]   = ...
        ekf_toa_robust(parameter.measureddistances,theta_init,parameter.BS,rekf);
end

toc

show_results_book
```

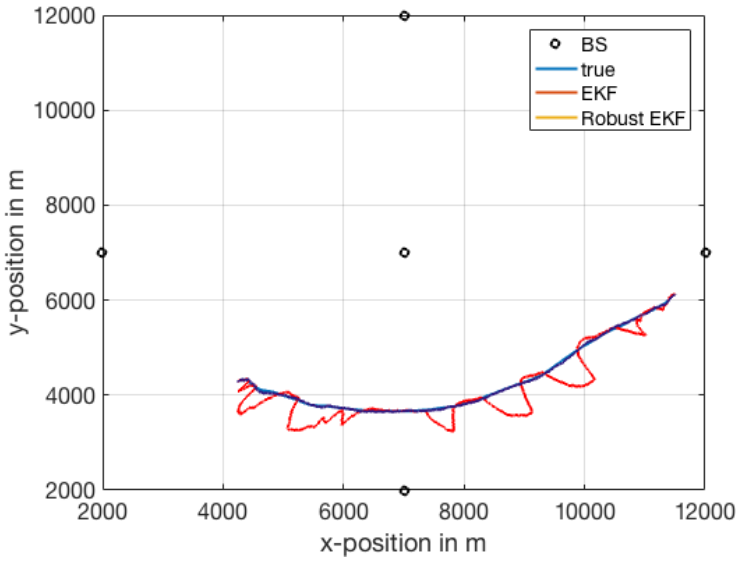


Figure 8: Trajectrory and EKF-based estimates as implemented in `ekf_toa` and `ekf_toa_robust`.

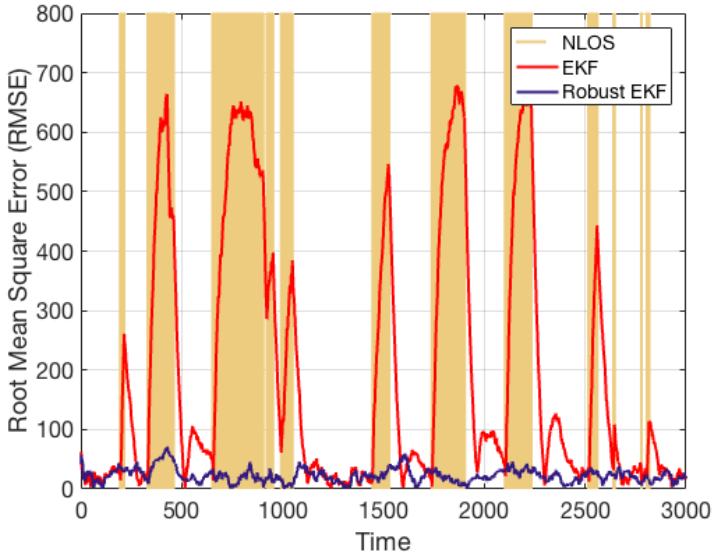


Figure 9: RMSE of EKF and robust EKF.

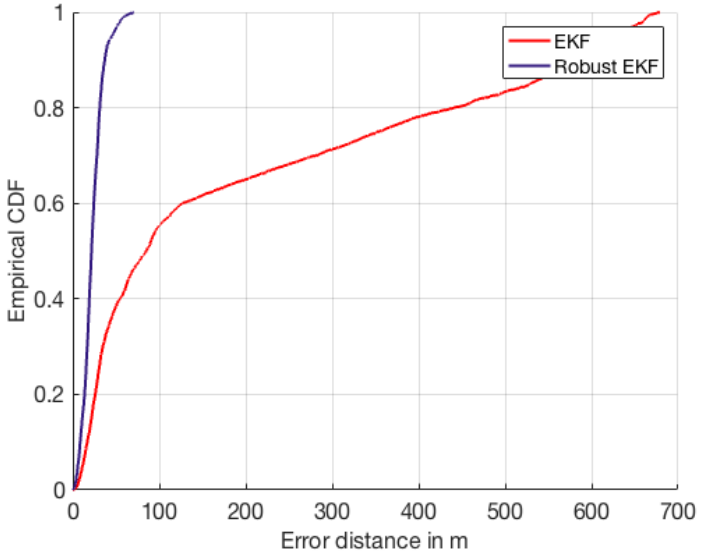


Figure 10: Cumulative distribution function (cdf) for EKF and robust EKF.

5 Robust Methods for Dependent Data

Autoregressive moving-average (ARMA) models are among the most popular models for characterizing dependent data and they have long been used in numerous real-world applications. This Section therefore focuses on robust parameter estimation for ARMA models associated with random processes for which the majority of the samples are appropriately modeled by a stationary and invertible ARMA model and a minority consists of outliers with respect to the ARMA model.

The methods in this Section are based on the the bounded influence propagation (BIP)-ARMA model, which is defined as

$$y_t = \mu + \sum_{i=1}^{p} \phi_i (y_{t-i} - \mu) + a_t - \sum_{i=1}^{r} \left(\phi_i a_{t-i} + (\theta_i - \phi_i) \sigma \eta \left(\frac{a_{t-i}}{\sigma} \right) \right). \tag{41}$$

Here, $r = \max(p, q)$, if $r > p$, $\phi_{p+1} = \ldots = \phi_r = 0$, and if $r > q$, $\theta_{q+1} = \ldots = \theta_r = 0$. ARMA models are included by setting $\eta(x) = x$, while the propagation of outliers is prevented by choosing $\eta(x)$ to be one of the well-known robust score functions, e.g., Tukey's (5). For such a choice, all innovations that lie within some region around μ are left untouched and, on the other hand, the effect of a single AO or RO is bounded to a single corrupted innovation.

In (41), σ is a robust M-scale of a_t, i.e., it solves

$$\mathsf{E}\left[\psi \left(\frac{a_t}{\sigma} \right) \cdot \left(\frac{a_t}{\sigma} \right) \right] = b, \tag{42}$$

where the positive constant b must satisfy $0 < b < \rho(\infty)$ and is chosen as $\mathsf{E}[\rho(u)]$, where u is a standard normal random variable to achieve consistency with the Gaussian distribution.

From (41), the innovations sequence can be recursively obtained, for $t \geq p + 1$, according to

$$a_t^b(\boldsymbol{\beta}, \sigma) = y_t - \mu - \sum_{i=1}^{p} \phi_i (y_{t-i} - \mu) + \sum_{i=1}^{r} \left(\phi_i a_{t-i}^b(\boldsymbol{\beta}, \sigma) + (\theta_i - \phi_i) \sigma \eta \left(\frac{a_{t-i}^b(\boldsymbol{\beta}, \sigma)}{\sigma} \right) \right). \tag{43}$$

5.1 `ar_est_bip_tau` computes the BIP-τ estimate of the AR(p) model parameters via a robust Levinson–Durbin procedure.

`ar_est_bip_tau` computes the BIP-AR(p) τ-estimates based on a robust Levinson–Durbin procedure. The objective is to minimize the τ-scale of the BIP-AR(p) innovations, i.e.,

$$\hat{\sigma}_\tau(\boldsymbol{a}_N^b(\boldsymbol{\phi})) = \hat{\sigma}(\boldsymbol{a}_N^b(\boldsymbol{\phi})) \sqrt{\frac{1}{N-p} \sum_{t=p+1}^{N} \rho_2 \left(\frac{a_t^b(\boldsymbol{\phi})}{\hat{\sigma}(\boldsymbol{a}_N^b(\boldsymbol{\phi}))} \right)}, \tag{44}$$

where

$$a_t^b(\boldsymbol{\phi}, \sigma) = y_t - \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{i=1}^{p} \left(\phi_i a_{t-i}^b(\boldsymbol{\phi}, \sigma) - \phi_i \sigma \eta \left(\frac{a_{t-i}^b(\boldsymbol{\phi}, \sigma)}{\sigma} \right) \right), \tag{45}$$

The procedure is summarized in Algorithm 47. Figure 11 details results for the robust Levinson–Durbin procedure that allow the computation of the BIP-τ estimate of ϕ_1, i.e., the AR(1) model parameter. The top graph depicts the results for $y_t^{\varepsilon} = y_t$ (no contamination/outliers) with $\phi_1 = -0.5$ for $\sigma = 1$, $N = 1000$. The bottom graph displays an illustrative additive outlier (AO) example with 10 % equally spaced AOs with amplitudes of 10.

Algorithm 5: `ar_est_bip_tau`: computes the BIP-τ estimate of the AR(p) model parameters via a robust Levinson–Durbin procedure.

input : $\mathbf{y} \in \mathbb{R}^N$, p, grid spacing Δ_{ζ^0}
output : $\hat{\boldsymbol{\phi}}_\tau = (\hat{\phi}_1, \ldots, \hat{\phi}_p)^\top$, innovations τ-scale $\hat{\sigma}_\tau$
initialize: no initialization required

if $p > 1$ **then**
  for $\zeta^0 = -0.99 : \Delta_{\zeta^0} : 0.99$ **do**
    Compute AR(1) innovations $\boldsymbol{a}_N(\zeta^0)$ and BIP-AR(1) innovations $\boldsymbol{a}_N^b(\zeta^0, \hat{\sigma}(\zeta^0))$.
    Compute τ-scales resulting in $\hat{\sigma}_\tau(\boldsymbol{a}_N(\zeta^0))$ and $\hat{\sigma}_\tau(\boldsymbol{a}_N^b(\zeta^0, \hat{\sigma}(\zeta^0)))$.
  Fit polynomial to $(\zeta^0, \hat{\sigma}_\tau(\boldsymbol{a}_N(\zeta^0)))$, and $(\zeta^0, \hat{\sigma}_\tau(\boldsymbol{a}_N^b(\zeta^0, \hat{\sigma}(\zeta^0))))$ at $\zeta^0 = -0.99 : \Delta_{\zeta^0} : 0.99$.
  Compute the values ζ_1, ζ_2, for which the objective functions that are approximated by the polynomials, is minimized. Use the estimates based on the model that provides the smaller minimum, i.e.,

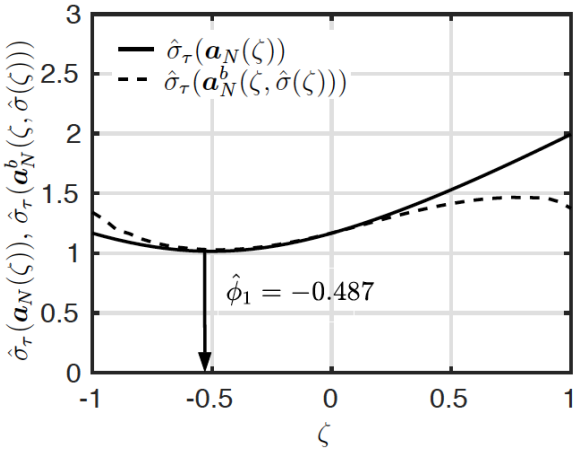
$$\hat{\phi}_1 = \underset{\zeta_1, \zeta_2}{\arg\min} \ \left\{ \hat{\sigma}_{\tau,\min}(\boldsymbol{a}_N(\zeta_1)), \hat{\sigma}_{\tau,\min}(\boldsymbol{a}_N^b(\zeta_2, \hat{\sigma}(\zeta_2))) \right\}. \tag{46}$$

if $p > 1$ **then**
  for $m = 2, \ldots, p$ **do**
    for $\zeta^0 = -0.99 : \Delta_{\zeta^0} : 0.99$ **do**
      Compute AR(m) innovations $\boldsymbol{a}_N(\zeta^0)$ and BIP-AR(m) innovations $\boldsymbol{a}_N^b(\zeta^0, \hat{\sigma}(\zeta^0))$.
      Compute τ-scales resulting in $\hat{\sigma}_\tau(\boldsymbol{a}_N(\zeta^0))$ and $\hat{\sigma}_\tau(\boldsymbol{a}_N^b(\zeta^0, \hat{\sigma}(\zeta^0)))$.
    Fit polynomial to $(\zeta^0, \hat{\sigma}_\tau(\boldsymbol{a}_N(\zeta^0)))$, and $(\zeta^0, \hat{\sigma}_\tau(\boldsymbol{a}_N^b(\zeta^0, \hat{\sigma}(\zeta^0))))$ at $\zeta^0 = -0.99 : \Delta_{\zeta^0} : 0.99$.
    Compute the values ζ_1, ζ_2, for which the objective functions that are approximated by the polynomials, is minimized. Use the estimates based on the model that provides the smaller minimum, i.e.,

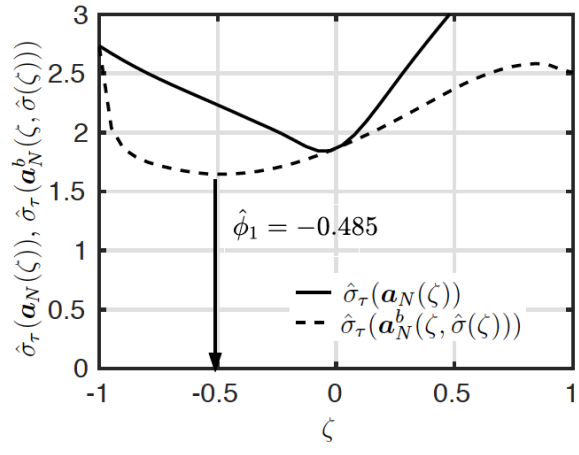
$$\hat{\phi}_{m,m} = \underset{\zeta_1, \zeta_2}{\arg\min} \ \left\{ \hat{\sigma}_{\tau,\min}(\boldsymbol{a}_N(\zeta)), \hat{\sigma}_{\tau,\min}(\boldsymbol{a}_N^b(\zeta, \hat{\sigma}(\zeta))) \right\} \tag{47}$$

    where the estimates from lower orders are incorporated by applying the Levinson–Durbin recursion

$$\hat{\phi}_{m,m} = \begin{cases} \zeta & \text{if} \quad i = m \\ \hat{\phi}_{m-1,i} - \zeta \hat{\phi}_{m-1,m-i} & \text{if} \quad 1 \le i \le m-1 \end{cases} \tag{48}$$



(a)



(b)

Figure 11: Example of finding $-1 < \zeta < 1$ which minimizes $\hat{\sigma}_\tau(\boldsymbol{a}_N(\zeta))$ and $\hat{\sigma}_\tau(\boldsymbol{a}_N^b(\zeta, \hat{\sigma}(\zeta)))$ for an AR(1) process with $\phi_1 = -0.5$ and $\sigma = 1$. (a) $y_t^\varepsilon = y_t$ clean data example. (b) 10 % equally spaced AOs with amplitudes of 10.

5.1.1 Syntax

```
function [phi_hat, x_filt, a_scale_final]=ar_est_bip_tau(x,P)
%   The function ar_est_bip_tau(x,P) comuptes BIP tau-estimates of the
%   AR model parameters. It also computes an outlier cleaned signal using
%   BIP-AR(P) predictions, and the tau-scale of the estimated innovations
%   series.
%
%% INPUTS:
% x: data (observations/measurements/signal)
% P: autoregressive order

%% OUTPUTS:
% phi_hat: a vector of bip tau estimates for each order up to P
% x_filt: cleaned version of x using robust BIP predictions
% a_scale_final: minimal tau scale of the innovations of BIP AR or AR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5.1.2 Dependencies

`madn`, `tau_scale`, `m_scale`, `ma_infinity`, `eta`

5.1.3 Example

```
rng(0);
%% Example: AR(1) with 30 percent isolated outliers

% AR(1) observations
N = 300; a = randn(N,1); x = filter(1,[1 -0.8],a); p = 1; q = 0;

% Isolated outliers
cont_prob = 0.3;     % outler contamination probability
outlier_ind = find(sign(rand(N,1)-(cont_prob))<0); % outlier index
outlier = 100*randn(N,1); % contaminating process
v = zeros(N,1); % additive outlier signal
v(outlier_ind) = outlier(outlier_ind);
v(1) = 0; % first sample should not be an outlier

% 30 percent of isolated additive outliers
x_ao = x+v;

% BIP Tau estimation
[phi_hat, x_filt, a_scale_final]=ar_est_bip_tau(x_ao,p);
```

5.2 `ar_est_bip_s` computes the BIP-S estimate of the AR(p) model parameters via a robust Levinson–Durbin procedure.

For S-estimators, i.e. for `ar_est_bip_s` the algorithm is nearly identical to the `ar_est_bip_tau`. The only difference is that the τ-scale of the BIP-AR(p) innovations in Eq. (44) is replaced by an M-scale, $\hat{\sigma}(\boldsymbol{a}_N^b(\boldsymbol{\phi}))$, of the BIP-AR($p$) innovations.

5.2.1 Syntax

```
function [phi_hat, x_filt, a_scale_final]=ar_est_bip_s(x,P)
%   The function ar_est_bip_s(x,P) comuptes BIP S-estimates of the
%   AR model parameters. It also computes an outlier cleaned signal using
%   BIP-AR(P) predictions, and the M-scale of the estimated innovations
%   series.
%
%% INPUTS:
```

```
% x: data (observations/measurements/signal)
% P: autoregressive order

%% OUTPUTS:
% phi_hat: a vector of bip tau estimates for each order up to P
% x_filt: cleaned version of x using robust BIP predictions
% a_scale_final: minimal tau scale of the innovations of BIP AR or AR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5.2.2 Dependencies

`madn`, `m_scale`, `ma_infinity`, `eta`

5.2.3 Example

```
rng(0);
%% Example: AR(1) with 30 percent isolated outliers

% AR(1) observations
N = 300; a = randn(N,1); x = filter(1,[1 -0.8],a); p = 1; q = 0;

% Isolated outliers
cont_prob = 0.3;    % outler contamination probability
outlier_ind = find(sign(rand(N,1)-(cont_prob))<0); % outlier index
outlier = 100*randn(N,1); % contaminating process
v = zeros(N,1); % additive outlier signal
v(outlier_ind) = outlier(outlier_ind);
v(1) = 0; % first sample should not be an outlier

% 30 percent of isolated additive outliers
x_ao = x+v;

% BIP S estimation
[phi_hat, x_filt, a_scale_final]=ar_est_bip_s(x_ao,p);
```

5.3 `robust_starting_point` computes the BIP-τ estimate-based starting point for robust ARMA parameter estimation.

Determining a robust estimate for the ARMA parameters, in general, involves minimizing a non-convex problem. The crucial point is to find a starting point that is sufficiently close to the true ARMA parameter vector $\boldsymbol{\beta}$. Due to the computational complexity, except for some very simple cases (e.g., $p+q \leq 2$), it is not possible to perform an exhaustive grid search.

`robust_starting_point` implements a simple procedure to find a starting point $\hat{\boldsymbol{\beta}}_0$, based on the BIP-AR model. For the BIP-AR model, the one-step prediction of y_t can be computed recursively for $t \geq p+1$ via:

$$\hat{y}_t = \sum_{i=1}^{p} \phi_i \left(y_{t-i} - a_{t-i}^b(\hat{\boldsymbol{\beta}}, \hat{\sigma}) + \hat{\sigma}\eta\left(\frac{a_{t-i}^b(\hat{\boldsymbol{\beta}}, \hat{\sigma})}{\hat{\sigma}} \right) \right) \tag{49}$$

and outlier-cleaned observations are obtained for $t \geq p+1$ by computing

$$y_t^* = y_t - a_t^b(\hat{\boldsymbol{\beta}}, \hat{\sigma}) + \hat{\sigma}\eta\left(\frac{a_t^b(\hat{\boldsymbol{\beta}}, \hat{\sigma})}{\hat{\sigma}} \right). \tag{50}$$

Here, $a_t^b(\hat{\boldsymbol{\beta}}, \hat{\sigma})$, $t = p+1, \ldots, N$ denote the BIP-AR innovations, and η is the score function that bounds outlier propagation.

To find a starting point for the ARMA parameter estimation, the data is first cleaned from outliers using an AR(p) approximation, which can be computed with `ar_bip_tau` or `ar_bip_s`. The starting point $\hat{\boldsymbol{\beta}}_0$ for robust ARMA parameter estimation is computed, based on y_t^*, and by using any classical ARMA parameter estimator. In this implementation, the function `robust_starting_point` calls `armax` from the MATLAB System Identification Toolbox to compute classical ARMA parameter estimate based on cleaned data. We suggest to replace the highlighted code by a different (nonrobust) ARMA parameter estimator if you do not have the MATLAB System Identification Toolbox. Note that any subsequent ARMA parameter estimation is performed on the observed data and the AR-approximation–based outlier cleaning is only used within the starting point algorithm to find $\hat{\boldsymbol{\beta}}_0$.

5.3.1 Syntax

```
function [beta_initial, x_filt] = robust_starting_point(x,p,q)
%   The function  robust_starting_point(x,p,q) provides a robust initial estimate ...
    for robust ARMA parameter estimation based on BIP-AR(p_long) approximation. It ...
    also computes an outlier cleaned signal using BIP-AR(p_long) predictions
%
%% INPUTS:
% x: data (observations/measurements/signal)
% p: autoregressive order
% q: moving-average order
%
%% OUTPUTS:
% beta_initial: robust starting point for AR(p)and MA(q) parameters based on ...
    BIP-AR(p_long) approximation
% x_filt: outlier cleaned signal using BIP-AR(p_long) predictions
%
%   The function "robust_starting_point" calls "armax" from Matlab System ...
    Identification
%   Toolbox to compute classical ARMA parameter estimate based on cleaned
%   data. Replace highlighted code by a different (nonrobust) ARMA parameter ...
    estimator if you
%   do not have the toolbox.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5.3.2 Dependencies

`ar_est_bip_s` or `ar_est_bip_tau`, `armax` (MATLAB System Identification Toolbox)

5.3.3 Example

```
rng(0);
%% Example: ARMA(1,1) with 10 percent patchy outliers

% Generate ARMA(1,1) observations
N = 1000; a = randn(N,1); x = filter([1 0.2],[1 -.8],a); p = 1; q = 1;

% Generate a patch of outliers of length 101 samples
v = 1000*randn(101,1);

% 10 percent of patchy additive outliers
x_ao = x; x_ao(100:200) = x(100:200)+v;

% robust starting point
[beta_initial, x_filt] = robust_starting_point(x_ao,p,q);
```

5.4 `arma_est_bip_tau` computes the BIP-τ estimate of the ARMA(p, q) model parameters.

As discussed in Section 8.6 of the book, there exists a nonlinear LS formulation for the τ-estimators of the ARMA(p, q) parameters. In this implementation, `arma_est_bip_tau` uses a Levenberg-Marquardt algorithm to solve the nonlinear LS problem, where the ARMA(p, q) innovations are replaced by the BIP-ARMA(p, q) innovations to bound the propagation of outliers. The function `robust_starting_point` provides a robust initial estimate.

5.4.1 Syntax

```
function result = arma_est_bip_tau(x,p,q)
%   The function arma_est_bip_tau(x,p,q) comuptes BIP tau-estimates of the
%   ARMA model parameters. It also computes an outlier cleaned signal using ...
    BIP-ARMA(p,q) predictions
%
%% INPUTS:
% x: data (observations/measurements/signal)
% p: autoregressive order
% q: moving-average order
%
%% OUTPUTS:
% result.ar_coeffs: vector of BIP-AR(p) tau-estimates
% result.ma_coeffs: vector of BIP-MA(q) tau-estimates
% result.inno_scale: BIP S-estimate of the innovations scale
% result.cleaned signal: outlier cleaned signal using BIP-ARMA(p,q) predictions
% result.ar_coeffs_init: robust starting point for BIP-AR(p) tau-estimates
% result.ma_coeffs_init: robust starting point for BIP-MA(q) tau-estimates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5.4.2 Dependencies

`robust_starting_point`, `ar_est_bip_tau`, `bip_tau_resid_sc`, `arma_tau_resid_sc`

5.4.3 Example

```
%% Example 3: MA(2) with 20 percent isolated outliers

% Generate MA(2) observations
N = 500; a = randn(N,1); x = filter([1 -0.7 0.5],1,a); p = 0; q = 2;

% Generate isolated outliers
cont_prob = 0.2;
outlier_ind = find(sign(rand(N,1)-(cont_prob))<0);
outlier = 100*randn(N,1);
v = zeros(N,1);
v(outlier_ind) = outlier(outlier_ind);
v(1:2) = 0;

% 20 percent of isolated additive outliers
x_ao = x+v;

% BIP Tau estimation
result = arma_est_bip_tau(x_ao,p,q);
```

Figure 12 shows the original, contaminated, and BIP-MA(2) τ-estimated signals for an MA(2) process with parameters (-0.8,0.5), and 20 percent additive outliers. For this example, the estimated MA parameters are (-0.7450, 0.4880). For further ARMA(p,q) examples, simply run the

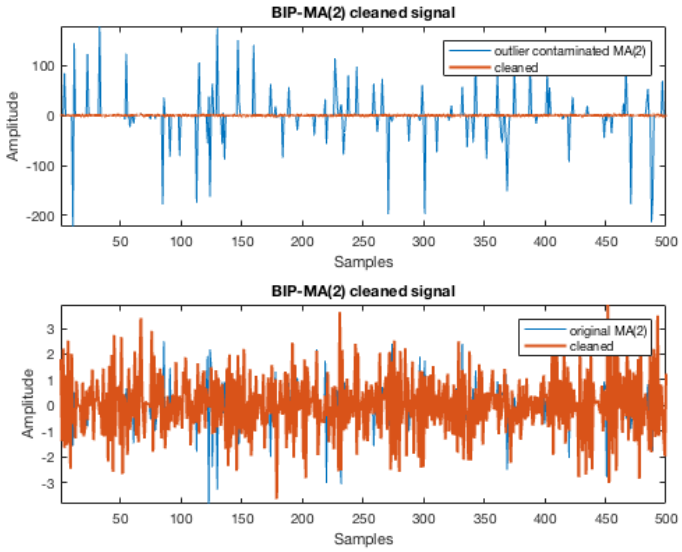


Figure 12: Original, contaminated, and BIP-MA(2) τ-estimated signals for an MA(2) process with parameters (-0.8,0.5), and 20 percent additive outliers. The estimated MA parameters are (-0.7450, 0.4880).

script `BIP_ARMA_Tau_demo.m`.

5.5 `arma_est_bip_s` computes the BIP-S estimate of the ARMA(p, q) model parameters.

Section 8.6 of the book provides the nonlinear LS formulation for the S-estimators of the ARMA(p, q) parameters. In this implementation, `arma_est_bip_s` uses a Levenberg-Marquardt algorithm to solve the nonlinear LS problem, where the ARMA(p, q) innovations are replaced by the BIP-ARMA(p, q) innovations to bound the propagation of outliers. The function `robust_starting_point` provides a robust initial estimate.

5.5.1 Syntax

```
function result = arma_est_bip_s(x,p,q)
%   The function arma_est_bip_s(x,p,q) comuptes BIP S-estimates of the
%   ARMA model parameters. It also computes an outlier cleaned signal using ...
    BIP-ARMA(p,q) predictions
%
%% INPUTS:
% x: data (observations/measurements/signal)
% p: autoregressive order
% q: moving-average order
%
%% OUTPUTS:
% result.ar_coeffs: vector of BIP-AR(p) S-estimates
% result.ma_coeffs: vector of BIP-MA(q) S-estimates
% result.inno_scale: BIP S-estimate of the innovations scale
% result.cleaned signal: outlier cleaned signal using BIP-ARMA(p,q) predictions
% result.ar_coeffs_init: robust starting point for BIP-AR(p) S-estimates
% result.ma_coeffs_init: robust starting point for BIP-MA(q) S-estimates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5.5.2 Dependencies

`robust_starting_point`, `ar_est_bip_s`, `bip_s_resid_sc`, `arma_s_resid_sc`

5.5.3 Example

```
%% Example 3: MA(2) with 20 percent isolated outliers

% Generate MA(2) observations
N = 500; a = randn(N,1); x = filter([1 -0.7 0.5],1,a); p = 0; q = 2;

% Generate isolated outliers
cont_prob = 0.2;
outlier_ind = find(sign(rand(N,1)-(cont_prob))<0);
outlier = 100*randn(N,1);
v = zeros(N,1);
v(outlier_ind) = outlier(outlier_ind);
v(1:2) = 0;

% 20 percent of isolated additive outliers
x_ao = x+v;

% BIP S-estimation
result = arma_est_bip_s(x_ao,p,q);
```

5.6 `arma_est_bip_mm` computes the BIP-MM estimate of the ARMA(p, q) model parameters.

A BIP-MM estimator of the ARMA(p, q) model parameters consists of two steps. First, a BIP-MM estimate is computed with `arma_est_bip_s`. This is followed by a BIP-M estimator using the initial parameters and scale from `arma_est_bip_s`.

As discussed in Section 8.6 of the book, there exists a nonlinear LS formulation for the M-estimators of the ARMA(p, q) parameters. In this implementation, `arma_est_bip_mm` uses a Levenberg-Marquardt algorithm to solve the nonlinear LS problem, where the ARMA(p, q) innovations are replaced by the BIP-ARMA(p, q) innovations to bound the propagation of outliers. The function `robust_starting_point` provides a robust initial estimate.

5.6.1 Syntax

```
function result = arma_est_bip_mm(x,p,q)
%   The function  arma_est_bip_mm(x,p,q) comuptes BIP MM-estimates of the
%   ARMA model parameters. It also computes an outlier cleaned signal using ...
    BIP-ARMA(p,q) predictions
%
%% INPUTS:
% x: data (observations/measurements/signal)
% p: autoregressive order
% q: moving-average order
%
%% OUTPUTS:
% result.ar_coeffs: vector of BIP-AR(p) MM-estimates
% result.ma_coeffs: vector of BIP-MA(q) MM-estimates
% result.inno_scale: BIP s-estimate of the innovations scale
% result.cleaned signal: outlier cleaned signal using BIP-ARMA(p,q) predictions
% result.ar_coeffs_init: robust starting point for BIP-AR(p) MM-estimates
% result.ma_coeffs_init: robust starting point for BIP-MA(q) MM-estimates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5.6.2 Dependencies

`arma_est_bip_s`, `robust_starting_point`, `ar_est_bip_s`, `bip_s_resid_sc`, `arma_s_resid_sc`

5.6.3 Example

```
%% Example 3: MA(2) with 20 percent isolated outliers

% Generate MA(2) observations
N = 500; a = randn(N,1); x = filter([1 -0.7 0.5],1,a); p = 0; q = 2;

% Generate isolated outliers
cont_prob = 0.2;
outlier_ind = find(sign(rand(N,1)-(cont_prob))<0);
outlier = 100*randn(N,1);
v = zeros(N,1);
v(outlier_ind) = outlier(outlier_ind);
v(1:2) = 0;

% 20 percent of isolated additive outliers
x_ao = x+v;

% BIP MM-estimation
result = arma_est_bip_mm(x_ao,p,q);
```

5.7 Application Example: Robust Data Cleaning for PPG-Based Pulse-Rate Variability Analysis

This example considers the estimation of the pulse-rate variability from photoplethysmographic (PPG) signals. Running the code below you obtain Figure 13, Figure 14, and Figure 15. This example was given in Section 11.5 of the book.

```
load('ecg_ppg.mat')


%% Plot PPG and ECG (referece) Signals
t_ppg = linspace(0,length(ppg_signal)/1000,length(ppg_signal));
t_ecg = linspace(0,length(ecg_signal)/1000,length(ecg_signal));

figure,
hold on
subplot(2,1,1); plot(t_ppg,ppg_signal,'linewidth',2);xlabel('Time ...
    (s)');ylabel('Amplitude (mV)');title('Photoplethysmogram (PPG)');axis tight;
subplot(2,1,2); plot(t_ecg,ecg_signal,'linewidth',2);xlabel('Time ...
    (s)');ylabel('Amplitude (mV)');title('Electrocardiogram (ECG)');axis tight;


%% Plot inter-beat-intervals
ibi_ppg = diff(ppg_pos);
ibi_ecg = diff(ecg_pos);

figure,
subplot(2,1,1); ...
    plot(ibi_ppg,'linewidth',2);xlabel('Samples');ylabel('Inter-beat-interval ...
    (ms)');title('Inter-beat-intervals for Photoplethysmogram (PPG)');axis tight;
subplot(2,1,2); ...
    plot(ibi_ecg,'linewidth',2);xlabel('Samples');ylabel('Inter-beat-interval ...
    (ms)');title('Inter-beat-intervals for Electrocardiogram (ECG)');axis tight;
```

```
%% BIP-Tau Data Cleaning of PPG inter-beat-intervals


% Order Selection was computed offline based on robust information criteria
% for pp = 0:10
%     for qq = 0:10
%         result = arma_est_bip_tau(ibi_ppg-median(ibi_ppg),pp,qq);
%         SIC(pp+1,qq+1) = ...
    log(result.inno_scale^2)+(pp+qq)/length(ibi_ppg)*log(length(ibi_ppg));
%         AIC(pp+1,qq+1) = log(result.inno_scale^2)+2*(pp+qq)/length(ibi_ppg);
%         HQC(pp+1,qq+1) = ...
    log(result.inno_scale^2)+2*(pp+qq)/length(ibi_ppg)*log(log(length(ibi_ppg)));
%     end
% end

% Selected ARMA model
p = 0;
q = 11;

% BIP-ARMA parameter estimation and data cleaning
result = arma_est_bip_tau(fliplr(ibi_ppg)-median(ibi_ppg),p,q); % data is flipped ...
    since outliers appear in the first few samples
ibi_ppg_cl = fliplr(result.cleaned_signal') + median(ibi_ppg);

disp('Selected ARMA model: MA(11)')
disp('estimated coefficients:')
ma_coeffs = result.ma_coeffs
figure,
subplot(2,1,1); plot(ibi_ppg); hold on; ...
    plot(ibi_ppg_cl,'linewidth',2);xlabel('Samples');ylabel('Inter-beat-interval ...
    (ms)');title('Cleaned Inter-beat-intervals for Photoplethysmogram ...
    (PPG)');legend('original','cleaned');axis tight;
subplot(2,1,2); plot(ibi_ecg); hold on; ...
    plot(ibi_ppg_cl,'linewidth',2);xlabel('Samples');ylabel('Inter-beat-interval ...
    (ms)');title('Cleaned Inter-beat-intervals for PPG and ECG ...
    reference');legend('ECG reference','cleaned');axis tight;
```

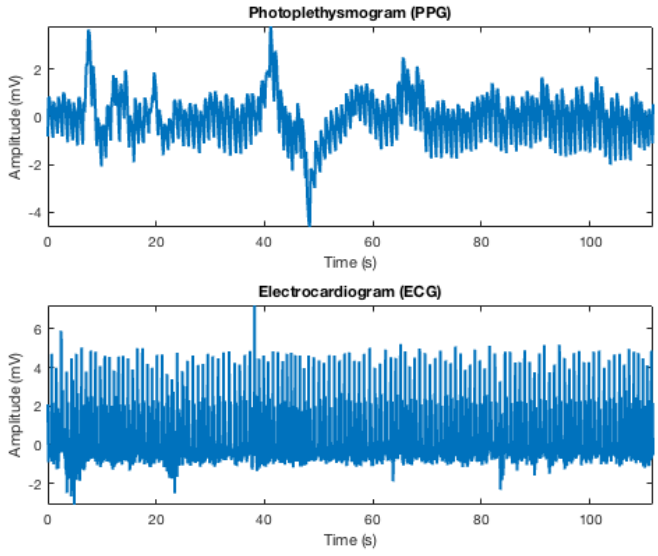


Figure 13: Measured electrocardiogram (ECG) and PPG signals.

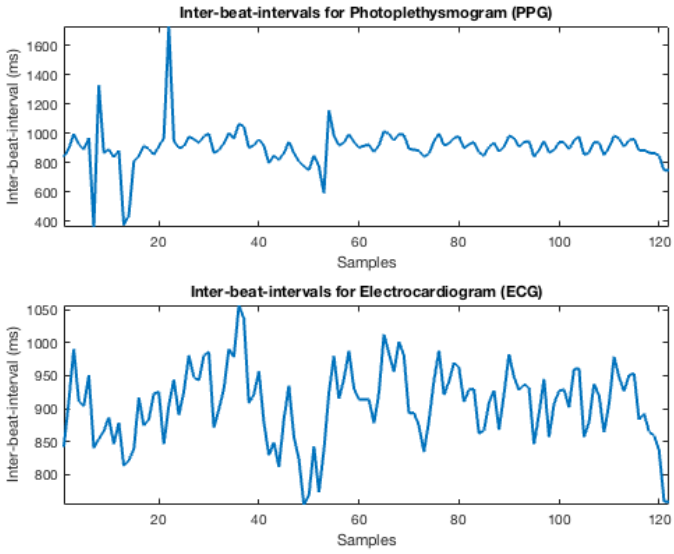


Figure 14: Measured inter-beat intervals for ECG and PPG signals.

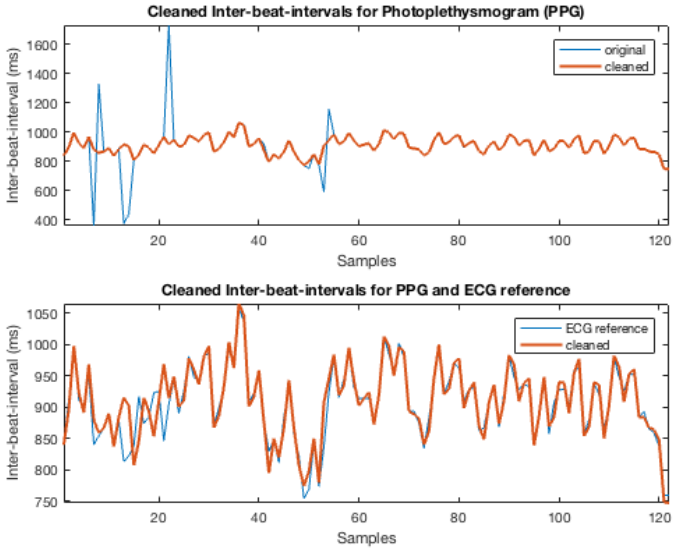


Figure 15: Cleaned inter-beat interval for PPG based on `arma_est_bip_tau`.

6 Robust Spectral Estimation

6.1 `repeated_median_filter` computes the repeated median estimate of the Fourier coefficients.

This approach uses robust trigonometric regression to fit sine and cosine coefficients at each Fourier frequency. An available sample $\mathbf{y}_N = (y_1, \ldots, \ldots y_N)^\top$, can be represented as the sum of N cosines and sines at the Fourier frequencies $\omega_k = 2\pi k/N$, for $k = 0, \ldots, \lfloor N/2 \rfloor$, according to

$$y_t = a_0 + \sum_{0<k<N/2} (a_k \cos(\omega_k t) + b_k \sin(\omega_k t)) + (-1)^t a_{N/2}, \quad t = 1, \ldots, N \tag{51}$$

where

$$a_k = \frac{2}{N} \sum_{t=1}^{N} y_t \cos(\omega_k t),$$

$$b_k = \frac{2}{N} \sum_{t=1}^{N} y_t \sin(\omega_k t),$$

$$a_0 = \frac{1}{N} \sum_{t=1}^{N} y_t,$$

and the last term

$$a_{N/2} = \frac{1}{N} \sum_{t=1}^{N} y_t (-1)^t$$

is included only if N is even. The function `repeated_median_filter` is our implementation of the method described in
High breakdown methods of time series analysis.
Tatum, L.G., and Hurvich, C. M.
Journal of the Royal Statistical Society. Series B (Methodological), pp. 881-896, 1993

Algorithm 6 details the pseudo-code for computing the initial repeated median estimates of $a_k, b_k, k = 1, \ldots, (N-1)/2$, for the case of N being prime. In this case, the estimator has a breakdown point of 50%. The full algorithm for non-prime N, which also computes a cleaned version of $\mathbf{y}_N$, by using the inverse Fourier transform of the robustly estimated Fourier coefficients, is implemented in `repeated_median_filter`. Unfortunately, Algorithm 6 is computationally complex.

6.1.1 Syntax

```
function [xFRM, ARM, BRM] = repeated_median_filter(x)
%   The function repeated_median_filter(x) is our implementation of the
%   method described in
%
%   "High breakdown methods of time series analysis.
%   Tatum, L.G., and Hurvich, C. M.
%   Journal of the Royal Statistical Society. Series B (Methodological),
%   pp. 881-896, 1993.
%
%   The code is based on an implementation by Falco Strasser, Signal Processing
%   Group, TU Darmstadt, October 2010.
%
%% inputs
```

Algorithm 6: `repeated_median_filter`: computes the repeated median estimate of the Fourier coefficients.

input : $\mathbf{y}_N \in \mathbb{R}$ sample of length $N > 2$ and N prime, M
output : $\hat{a}_k, \hat{b}_k, k = 1, \ldots, (N-1)/2$
initialize: $\hat{a}_k, \hat{b}_k = 0, k = 1, \ldots, (N-1)/2$

$$a_0 = \mathsf{med}(\mathbf{y}_N)$$

$$\mathbf{y}_N \leftarrow \mathbf{y}_N - a_0$$

for $m = 1, \ldots, M$ **do**
  for *kth strongest frequency,* $k \in \{1, \ldots, (N-1)/2\}$ **do**
    for $u = 1, \ldots, N$ **do**
      for $v = 1, \ldots, N$ *with* $v \neq u$ **do**

$$a_{kuv} \leftarrow \frac{y_u \sin(\omega_k v) - y_v \sin(\omega_k u)}{\sin(\omega_k (v-u))}$$

$$b_{kuv} \leftarrow \frac{y_v \sin(\omega_k u) - y_u \sin(\omega_k v)}{\sin(\omega_k (v-u))}$$

$$\tilde{a}_k \leftarrow \mathsf{med}_v(\mathsf{med}_u(a_{kuv}))$$
$$\tilde{b}_k \leftarrow \mathsf{med}_v(\mathsf{med}_u(a_{kuv}))$$

$$\mathbf{y}_N \leftarrow \mathbf{y}_N - \tilde{a}_k \cos(\omega_k t) - \tilde{b}_k \sin(\omega_k t)$$

$$\hat{a}_k \leftarrow \hat{a}_k + \tilde{a}_k$$
$$\hat{b}_k \leftarrow \hat{b}_k + \tilde{b}_k$$

```
% x: data (observations/measurements/signal), real-valued vector
%
%
%% outputs
% xFRM: repeated median filtered (outlier cleaned) signal
% ARM:  Fourier coefficients for cosine
% BRM:  Fourier coefficients for sine
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

6.1.2 Dependencies

6.1.3 Example

```
rng(0);

% ARMA(4,3) example by Moses et al.
a = [1 -1.316 1.4401 -1.0919 0.83527];
b_0 = 0.13137;
b = [0.13137 0.023543 0.10775 0.03516]/b_0;
p = 4;
q = 3;
N = 512;
w = linspace(0,pi,N/2);
s = exp(1i*w);
H = polyval(b,s) ./ polyval(a,s);  % true spectrum

z = randn(N,1);
y = filter(b,a,z);

 for jj = 50:51:N
 % 2 percent additive outliers
  y(jj) = y(jj)+1000*randn(1,1);
  end

% Repeated Median PSD Estimator
[xFRM, ARM, BRM] = repeated_median_filter(y);
```

6.2 `biweight_filter` computes the biweight estimate of the Fourier coefficients.

This approach, again, uses robust trigonometric regression to fit sine and cosine coefficients at each Fourier frequency.

Sine and cosine coefficients, $a_k, b_k, k = 1, \ldots, \lfloor N/2 \rfloor$, for each discrete frequency ω_k, be obtained from

$$\arg\min_{a_k, b_k} \sum_{t=1}^{N} \rho\left(\frac{r_t^{(k)}}{\hat{\sigma}_k}\right), \tag{52}$$

where $\hat{\sigma}_k$ is a robust scale estimate of $\mathbf{r}_k = (r_1^{(k)}, \ldots, r_N^{(k)})^\top$ with $r_t^{(k)} = y_t - a_k \cos(\omega_k t) - b_k \sin(\omega_k t)$ and $\rho(x)$ is Tukey's biweight function. Since the optimization problem stated in (52) is non-convex, Tatum and Hurvich proposed using the repeated median estimator as an initializer.

6.2.1 Syntax

```
function  [xFB, ABi, BBi] = biweight_filter(x)
%   The biweight_filter(x) is our implementation of the
%   method described in
%
%   "High breakdown methods of time series analysis.
%   Tatum, L.G., and Hurvich, C. M.
```

```
%   Journal of the Royal Statistical Society. Series B (Methodological),
%   pp. 881-896, 1993.
%
%   The code is based on an implementation by Falco Strasser, Signal Processing
%   Group, TU Darmstadt, October 2010.
%
%% INPUTS
% x: data (observations/measurements/signal), real-valued vector
%
%
%% OUTPUTS
% xFBi: Biweight filtered (outlier cleaned) signal
% ABi:  Fourier coefficients for cosine
% BBi:  Fourier coefficients for sine
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

6.2.2 Dependencies

`repeated_median_filter`, `fit` (MATLAB Curve Fitting Toolbox)

6.2.3 Example

```
rng(0);

% ARMA(4,3) example by Moses et al.
a = [1 -1.316 1.4401 -1.0919 0.83527];
b_0 = 0.13137;
b = [0.13137 0.023543 0.10775 0.03516]/b_0;
p = 4;
q = 3;
N = 512;
w = linspace(0,pi,N/2);
s = exp(1i*w);
H = polyval(b,s) ./ polyval(a,s);  % true spectrum

z = randn(N,1);
y = filter(b,a,z);

 for jj = 50:51:N
 % 2 percent additive outliers
  y(jj) = y(jj)+1000*randn(1,1);
  end

% Repeated Median PSD Estimator
[xFRM, ARM, BRM] = repeated_median_filter(y);
```

6.3 `spec_arma_est_bip_tau` computes the robust ARMA power spectral density estimate based on the bounded innovation propagation τ-estimator.

The Weierstrass theorem states that any continuous power spectral density (PSD) can be approximated arbitrarily closely by a rational PSD, provided that the degrees of the polynomials are chosen sufficiently large. Robust ARMA parameter estimators that were discussed in Section 8.3 of the book, lead to robust PSD estimators if the ARMA parameter estimates are inserted into the following equation

$$\hat{S}_{yy}(e^{\jmath\omega}) = \hat{\sigma}^2 \left| \frac{\hat{\theta}(e^{\jmath\omega})}{\hat{\phi}(e^{\jmath\omega})} \right|^2 , \tag{53}$$

with

$$\hat{\phi}(e^{\jmath\omega}) = 1 - \sum_{k=1}^{p} \hat{\phi}_k e^{-\jmath\omega k}, \tag{54}$$

$$\hat{\theta}(e^{\jmath\omega}) = 1 - \sum_{l=1}^{q} \hat{\theta}_l e^{-\jmath\omega l}. \tag{55}$$

and $\hat{\sigma}^2$ denoting the innovations variance.

The function `spec_arma_est_bip_tau` estimates the PSD based on (53), and estimates the parameters by calling the function `arma_est_bip_tau`.

6.3.1 Syntax

```
function [PxxdB, Pxx, w, sigma_hat] = spec_arma_est_bip_tau(x,p,q)
%   The function spec_arma_est_bip_tau(x,p,q) comuptes spectral estimates using the ...
    BIP tau-estimates of the
%   ARMA model parameters.
%
%% INPUTS
% x: data (observations/measurements/signal)
% p: autoregressive order
% q: moving-average order
%
%% OUTPUTS
% PxxdB: spectral estimate in dB
% Pxx: spectral estimate
% w: frequency (0,pi)
% sigma_hat: BIP tau-scale estimate of the innovations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

6.3.2 Dependencies

`arma_est_bip_tau`, `armax` (MATLAB System Identification Toolbox)

6.3.3 Example

```
rng(0);

% ARMA(4,3) example by Moses et al.
a = [1 -1.316 1.4401 -1.0919 0.83527];
b_0 = 0.13137;
b = [0.13137 0.023543 0.10775 0.03516]/b_0;
p = 4;
q = 3;
N = 512;
w = linspace(0,pi,N/2);
s = exp(1i*w);
H = polyval(b,s) ./ polyval(a,s);  % true spectrum

z = randn(N,1);
y = filter(b,a,z);

 for jj = 50:51:N
 % 2 percent additive outliers
  y(jj) = y(jj)+1000*randn(1,1);
  end

[PyydB, Pyy, w, sigma_hat] = spec_arma_est_bip_tau(y,p,q);

figure,
```

```
hold on
plot(w/(2*pi),10*log10(1/(2*pi)*abs(H).^2)) % true spectrum
plot(w/(2*pi), PxxdB) % BIP tau-estimate
```

6.4 `spec_arma_est_bip_s` computes the robust ARMA power spectral density estimate based on the bounded innovation propagation S-estimator.

The function `spec_arma_est_bip_s` estimates the PSD based on (53), and estimates the parameters by calling the function `arma_est_bip_s`.

6.4.1 Syntax

```
function [PxxdB, Pxx, w, sigma_hat] = spec_arma_est_bip_s(x,p,q)
%   The function spec_arma_est_bip_s(x,p,q) comuptes spectral estimates using the ...
    BIP S-estimates of the
%   ARMA model parameters.
%
%% INPUTS
% x: data (observations/measurements/signal)
% p: autoregressive order
% q: moving-average order
%
%% OUTPUTS
% PxxdB: spectral estimate in dB
% Pxx: spectral estimate
% w: frequency (0,pi)
% sigma_hat: BIP M-scale estimate of the innovations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

6.4.2 Dependencies

`arma_est_bip_s`, `armax` (MATLAB System Identification Toolbox)

6.4.3 Example

```
rng(0);

% ARMA(4,3) example by Moses et al.
a = [1 -1.316 1.4401 -1.0919 0.83527];
b_0 = 0.13137;
b = [0.13137 0.023543 0.10775 0.03516]/b_0;
p = 4;
q = 3;
N = 512;
w = linspace(0,pi,N/2);
s = exp(1i*w);
H = polyval(b,s) ./ polyval(a,s);  % true spectrum

z = randn(N,1);
y = filter(b,a,z);

 for jj = 50:51:N
 % 2 percent additive outliers
  y(jj) = y(jj)+1000*randn(1,1);
  end

[PyydB, Pyy, w, sigma_hat] = spec_arma_est_bip_s(y,p,q);

figure,
```

```
hold on
plot(w/(2*pi),10*log10(1/(2*pi)*abs(H).^2)) % true spectrum
plot(w/(2*pi), PxxdB) % BIP S-estimate
```

6.5 `spec_arma_est_bip_mm` computes the robust ARMA power spectral density estimate based on the bounded innovation propagation MM-estimator.

The function `spec_arma_est_bip_mm` estimates the PSD based on (53), and estimates the parameters by calling the function `arma_est_bip_mm`.

6.5.1 Syntax

```
function [PxxdB, Pxx, w, sigma_hat] = spec_arma_est_bip_mm(x,p,q)
%   The function spec_arma_est_bip_s(x,p,q) comuptes spectral estimates using the ...
    BIP MM-estimates of the
%   ARMA model parameters.
%
%% INPUTS
% x: data (observations/measurements/signal)
% p: autoregressive order
% q: moving-average order
%
%% OUTPUTS
% PxxdB: spectral estimate in dB
% Pxx: spectral estimate
% w: frequency (0,pi)
% sigma_hat: BIP M-scale estimate of the innovations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

6.5.2 Dependencies

`arma_est_bip_mm`, `armax` (MATLAB System Identification Toolbox)

6.5.3 Example

```
rng(0);

% ARMA(4,3) example by Moses et al.
a = [1 -1.316 1.4401 -1.0919 0.83527];
b_0 = 0.13137;
b = [0.13137 0.023543 0.10775 0.03516]/b_0;
p = 4;
q = 3;
N = 512;
w = linspace(0,pi,N/2);
s = exp(1i*w);
H = polyval(b,s) ./ polyval(a,s);  % true spectrum

z = randn(N,1);
y = filter(b,a,z);

 for jj = 50:51:N
 % 2 percent additive outliers
  y(jj) = y(jj)+1000*randn(1,1);
  end

[PyydB, Pyy, w, sigma_hat] = spec_arma_est_bip_mm(y,p,q);

figure,
```

```
hold on
plot(w/(2*pi),10*log10(1/(2*pi)*abs(H).^2)) % true spectrum
plot(w/(2*pi), PxxdB) % BIP MM-estimate
```

6.6 Spectral Estimation Example: ARMA(4,3) With Additive Outliers

This example considers the estimation an ARMA(4,3) spectrum. Running the code below you obtain Figure 16, Figure 17, and Figure 18. This example was given in Section 9.3 of the book.

```
% ARMA(4,3) by Moses et al with 2 percent additive outliers

rng(0);

%% Generate Data
N = 512;

% example from Moses
a = [1 -1.316 1.4401 -1.0919 0.83527];

b_0 = 0.13137;
b = [0.13137 0.023543 0.10775 0.03516]/b_0;

p = 4;
q = 3;

w = linspace(0,pi,N/2);
s = exp(1i*w);
H = polyval(b,s) ./ polyval(a,s);

z = randn(N,1);
y = filter(b,a,z);
x = y;  % clean data case

 for jj = 50:51:N
 % 2 percent additive outliers
  y(jj) = y(jj)+1000*randn(1,1);
  end

%% Compute PSD estimates

% periodogram estimate
Pxx = abs(fft(y)).^2;

% BIP Tau ARMA estimate
[PxxdB_tau, Pxx_tau, w, sigma_hat] = spec_arma_est_bip_tau(y,p,q);

% Repeated Median PSD Estimator
[xFRM, ARM, BRM] = repeated_median_filter(y);
Pxx_RM = abs(fft(xFRM)).^2;


%% Plot results
figure,
set(gca,'FontSize',18)
hold on
plot(w/(2*pi),10*log10(1/(2*pi)*abs(H).^2), 'linewidth',2) % true spectrum
plot(w/(2*pi), 10*log10(1/N/(2*pi)*Pxx(1:N/2)), 'linewidth',2) % periodogram
xlabel('Normalized frequency')
ylabel('PSD [dB]')
legend('true PSD', 'Periodogram')


%% plot results
figure,
```

```
set(gca,'FontSize',18)
hold on
plot(w/(2*pi),10*log10(1/(2*pi)*abs(H).^2), 'linewidth',2) % true spectrum
plot(w/(2*pi),PxxdB, 'linewidth',2) % BIP-ARMA tau PSD estimate
xlabel('Normalized frequency')
ylabel('PSD [dB]')
legend('true PSD', 'BIP tau-estimate')

%% plot results
figure,
set(gca,'FontSize',18)
hold on
plot(w/(2*pi),10*log10(1/(2*pi)*abs(H).^2), 'linewidth',2) % true spectrum
plot(w/(2*pi), 10*log10(1/N/(2*pi)*Pxx_RM(1:N/2)), 'linewidth',2) % repeated median ...
    PSD estimate
xlabel('Normalized frequency')
ylabel('PSD [dB]')
legend('true PSD', 'Repeated median estimate')
```

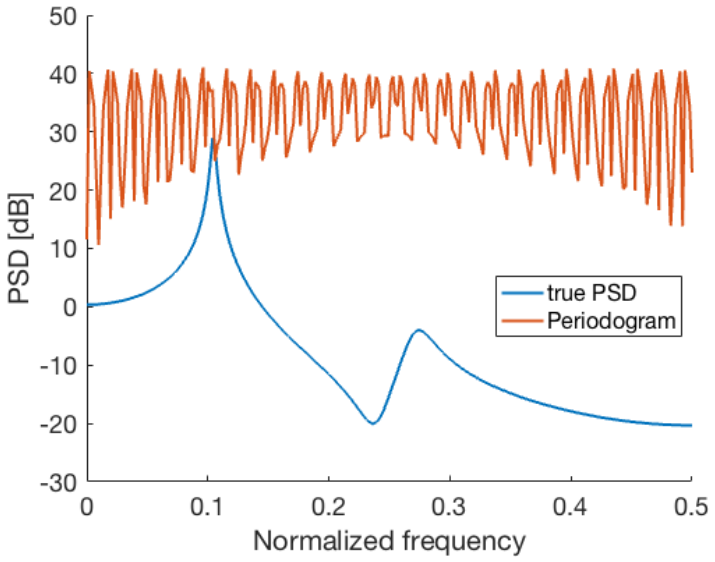


Figure 16: True PSD and periodogram estimate for an ARMA(4,3) with 2 percent additive outliers.

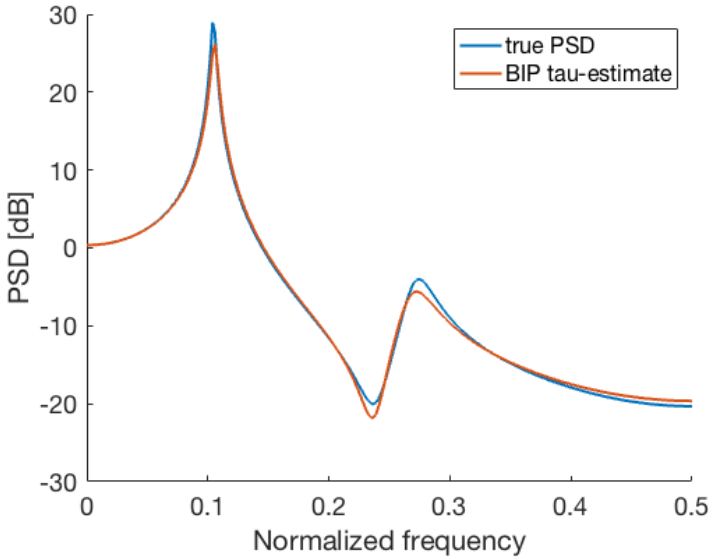


Figure 17: True PSD and BIP-τ estimate for an ARMA(4,3) with 2 percent additive outliers.

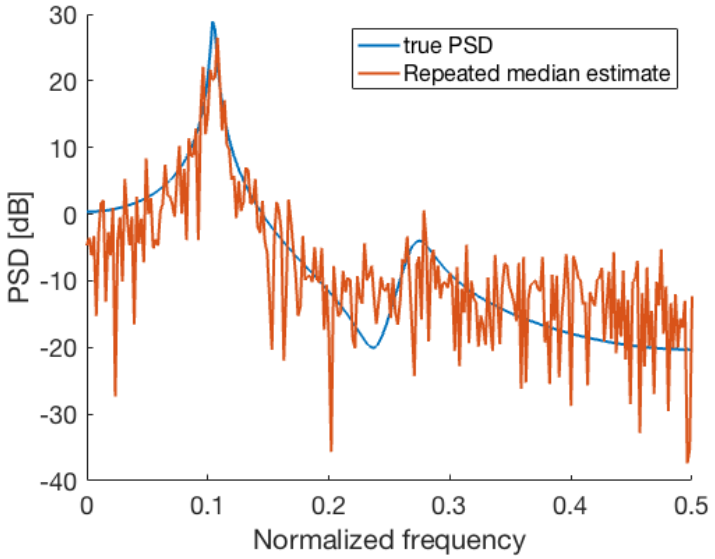


Figure 18: True PSD and repeated median filter periodogram estimate for an ARMA(4,3) with 2 percent additive outliers.

7 Auxiliary Functions

7.1 `whub`**: computes Huber's weights.**

7.2 `psihub`**: computes Huber's $\psi(x)$.**

7.3 `rhohub`**: computes Huber's $\rho(x)$.**

7.4 `wtuk`**: computes Tukey's weights.**

7.5 `psituk`**: computes Tukey's $\psi(x)$.**

7.6 `rhotuk`**: computes Tukey's $\rho(x)$.**

7.7 `SoftThres`**: computes soft thresholding function.**

7.8 `madn`**: computes the normalized median absolute deviations scale estimate.**

8 Data Sets from the Book

- `ecg_ppg.mat`
- `images.mat`
- `prostate.mat`

9 History and Major Updates

This is version 1.0 of the toolbox, which has been finalized on 10 October 2018. It has been tested using MATLAB R2016b running on macOS Sierra version 10.12.6 (16G29).