Polynomial smoothing of time series with additive step discontinuities: MATLAB Toolbox

Ivan Selesnick

August 2012

Contents

1 Introduction

This document is supplementary material for the paper:

Paper: Polynomial smoothing of time series with additive step discontinuities

Journal: IEEE Transactions on Signal Processing.

Authors: I. W. Selesnick, S. Arnold, and V. R. Dantham
Polytechnic Institute of New York University
Electrical and Computer Engineering
New York, USA

The paper addresses the problem of estimating simultaneously a local polynomial signal and an approximately piecewise constant signal from a noisy additive mixture. The supplementary material includes a MATLAB software package. The software package includes programs to reproduce the examples in the paper. The input-output parameters of the main functions are listed in Section 2. Examples are given in detail in the subsequent sections. Section 6 gives a listing of the main functions.

Some functions in this software packages require the use the MATLAB Signal Processing Toolbox (specifically, the `buffer` function). The package makes extensive use of sparse matrices and solvers for sparse banded matrices in MATLAB.

For additional information or questions, contact Ivan Selesnick (e-mail: selesi@poly.edu).

This work is supported by NSF grants CCF-1018020 and CBET-0933531.

2 Input-Output Parameters

```
[x, p, cost] = patv(y, d, lambda, Nit, mu0, mu1)
PATV: Simultaneous polynomial approximation and total variation filtering

INPUT
  y - noisy data
  d - order of polynomial
  lambda - regularization parameter
  Nit - number of iterations
  mu0, mu1 - augmented Lagrangian parameters

OUTPUT
  x - TV component
  p - polynomial component
  cost - cost function history
```

```
[x, p, cost] = lopatv(y, L, P, deg, lambda, Nit, mu0, mu)
LoPATV: Simultaneous local polynomial approximation and total variation filtering
(sliding window with overlapping)

INPUT
  y - noisy data
  L - block length
  P - overlapping (number of samples common to adjacent blocks)
  deg - polynomial degree
  lambda - regularization parameter
  Nit - number of iterations
  mu0, mu - augmented Lagrangian parameters

OUTPUT
  x - step function (TV component)
  p - local polynomial component
  cost - cost function history

Number of blocks = (length(y)-L)/(L-P)+1
If this is not an integer, then input signal y will be truncated.
```

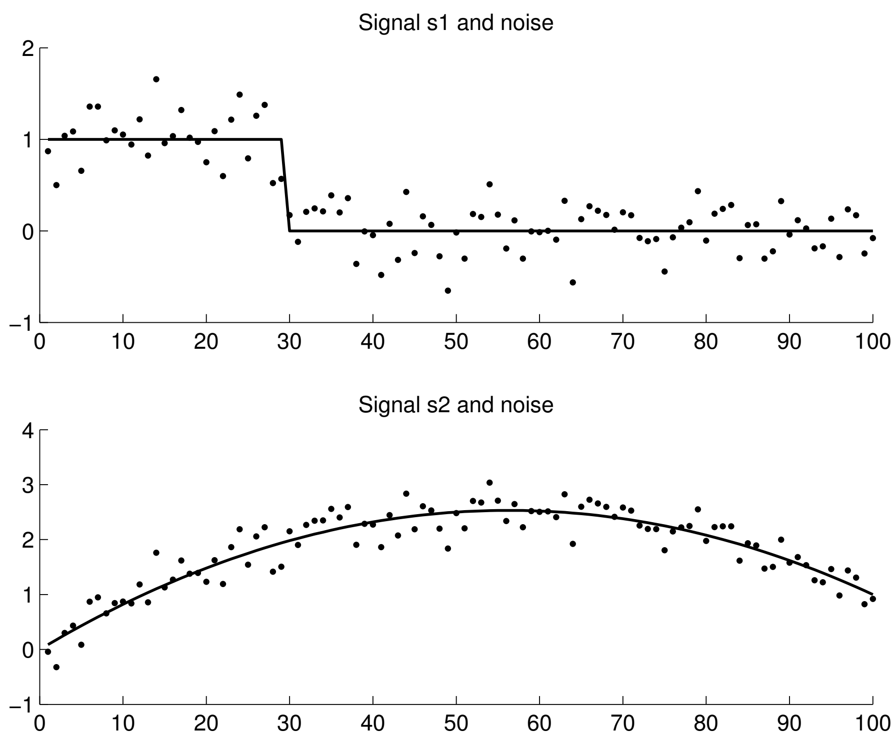
```
[x, p, cost, constr] = cpatv(y, d, r, Nit, mu0, mu1)
C-PATV: Simultaneous polynomial approximation and total variation filtering,
constrained formulation: ||H(y-x)||_2 <= r

INPUT
  y - noisy data
  d - order of polynomial
  r - constraint parameter
  Nit - number of iterations
  mu0, mu1 - augmented Lagrangian parameters

OUTPUT
  x - TV component
  p - polynomial component
  cost - cost function history
  constr - constraint function history
```

```
[x,p,cost] = patv_Lp(y, d, lambda, p, E, Nit, mu0, mu1)
Enhanced PATV: Simultaneous polynomial approximation and total variation
filtering
  Regularization : lambda * sum((abs(diff(x)) + E).^p);

INPUT
  y - noisy data
  d - order of polynomial
  lambda - regularization parameter
  p, E - Lp norm
  Nit - number of iterations
  mu0, mu1 - Augmented Lagrangian parameters

OUTPUT
  x - TV component
  p - polynomial component
  cost - cost function history
```

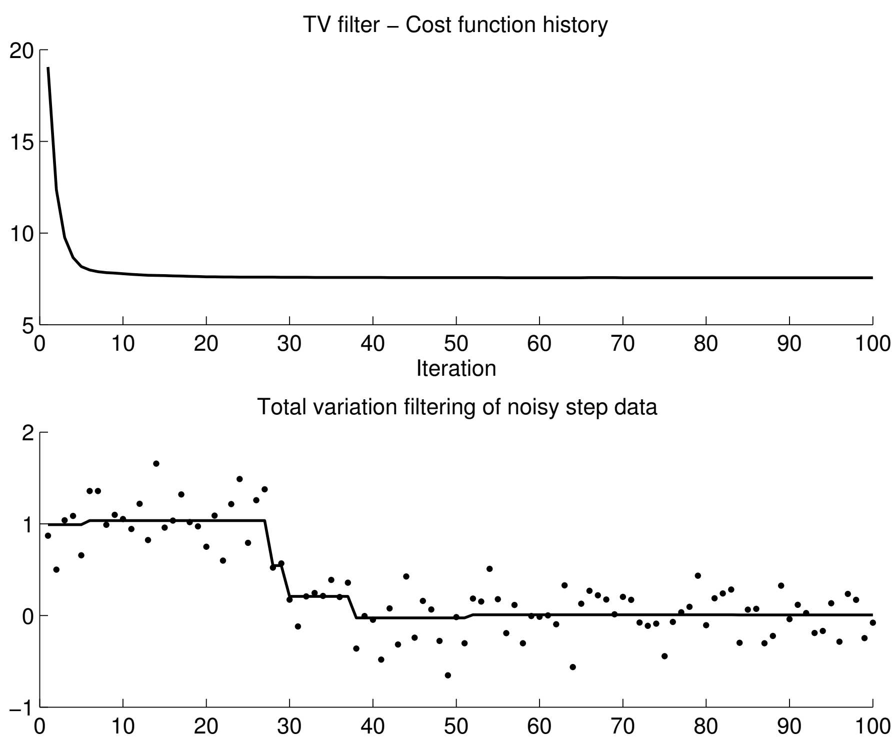
```
[x, p, cost] = lopatv_Lp(y, L, P, deg, lambda, Nit, mu0, mu, pow, E)
LoPATV_Lp: Enhanced local polynomial approximation and total variation filtering
(sliding window with overlapping) with Lp norm

INPUT
  y - noisy data
  L - block length
  P - overlapping (number of samples common to adjacent blocks)
  deg - polynomial degree (1, 2, 3)
  lambda - regularization parameter
  Nit - number of iterations
  mu - augmented Lagrangian parameters
  pow - power (Lp norm)
  E - small number

OUTPUT
  x - TV component
  p - local polynomial component
  cost - cost function history

Number of blocks = (length(y)-L)/(L-P)+1
If this is not an integer, then input signal y will be truncated.
```

3 Example 1

```
% Create test signals

N = 100;                              % N : length of data
n = (1:N)';
s1 = n < 0.3*N;                       % s1 : step function
s2 = 2-2*((n-N/2)/(N/2)).^2;
s2 = s2 + n/N;                        % s2 : polynomial function

randn('state',0);                     % Initialize randn so that example can be exactly reproduced
noise = randn(N,1);
sigma = 0.26;
noise = noise / sqrt((1/N)*sum(noise.^2)) * sigma;      % Gaussian (normal) noise

figure(1)
clf
subplot(2,1,1)
plot(n, s1, n, s1+noise, '.')
title('Signal s1 and noise')
box off

subplot(2,1,2)
plot(n, s2, n, s2+noise, '.')
title('Signal s2 and noise')
box off
```

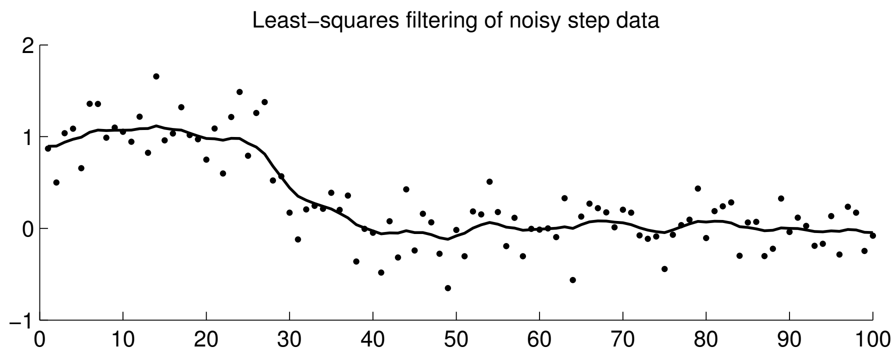


```
% Example of Total Variation filtering
% TV filtering of noisy step data

lambda = 1.6;                           % lambda : regularization parameter for TV filter
Nit = 100;                              % Nit : number of iterations for TV filter algorithm
[x_tv, cost] = TVfilt(s1+noise, lambda, Nit, 1.5, 10);

figure(1)
subplot(2,1,1)
plot(cost)
title('TV filter - Cost function history')
xlabel('Iteration')
box off

% Display output of TV filter

subplot(2,1,2)
plot(n, s1 + noise, '.k', n, x_tv);
title('Total variation filtering of noisy step data');
box off
```



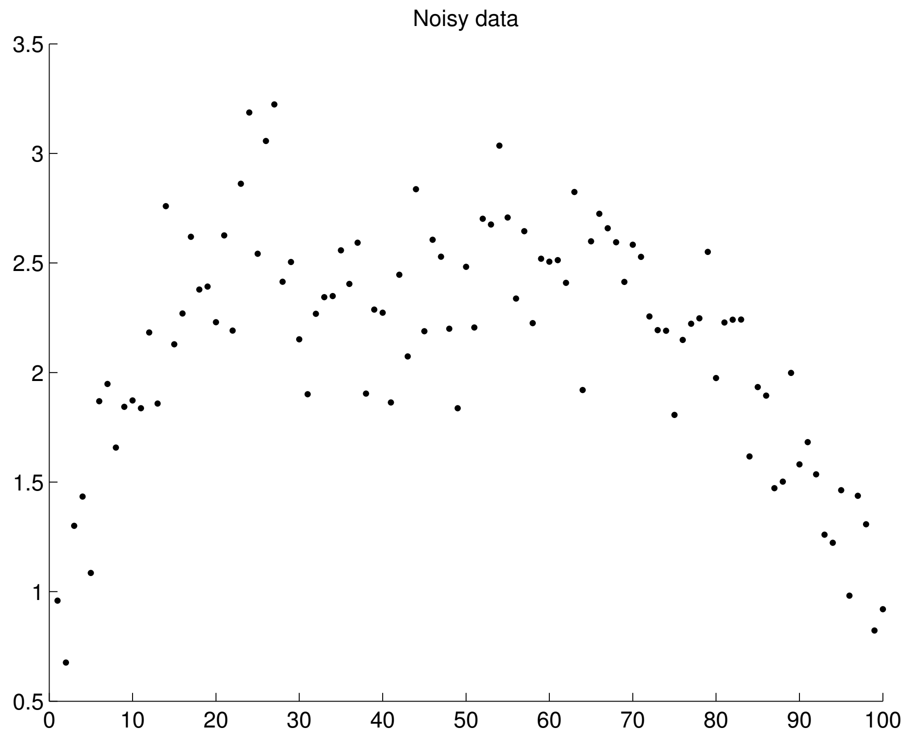
The function `TVfilt` is an ADMM-algorithm to perform total variation filtering. Recently an exact algorithm for TV filtering has been derived. See http://hal.archives-ouvertes.fr/hal-00675043/

```
% Least-square filtering of noisy step data

lambda = 10;                              % lambda : regularzation parameter for least squares filter
e = ones(N-1, 1);
D = spdiags([e -e], [0 1], N-1, N);       % D : first-order different matrix (sparse)
H = speye(N) + lambda*(D'*D);             % H : I + lambda D'*D (sparse)
x_l2 = H \ (s1+noise);

figure(1)
clf
subplot(2,1,1)
plot(n, s1 + noise, '.k', n, x_l2);
title('Least-squares filtering of noisy step data');
box off
```



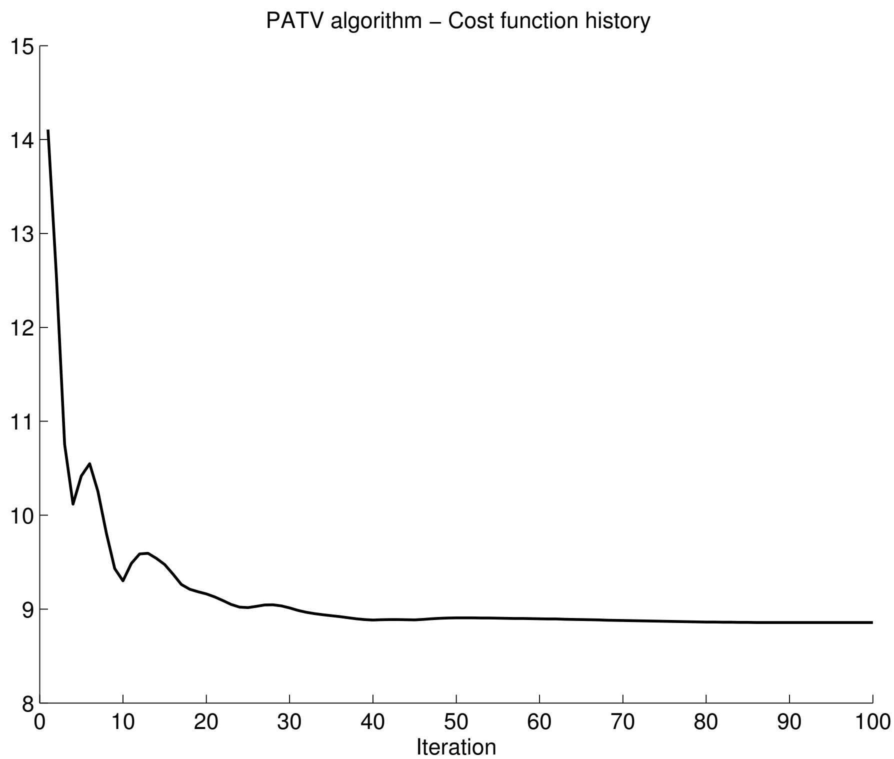
The least-square solution can be found by solving a sparse system of equations.

```
% Create polynomial signal with additive step discontinuity

s = s1 + s2;                 % s : polynomial signal with additive step discontinuity
y = s + noise;               % y : noisy signal

figure(1)
clf
plot(n, y, '.k')
title('Noisy data');
box off
```



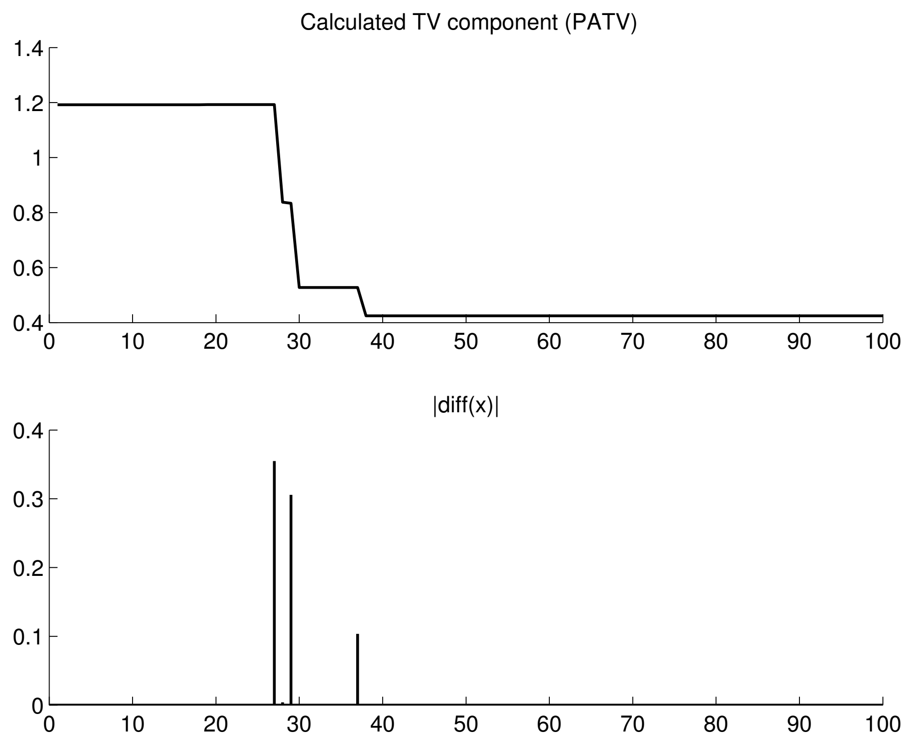
The data consists of a polynomial component, an additive step discontinuity, and noise.

```
% Run PATV filter algorithm (Polynomial approximation + total variation)

% parameters
d = 2;                          % d : degree of approximation polynomial
lambda = 3;                     % lambda : regularization parameter

Nit = 100;                      % Nit : number of iterations
mu0 = 20;                       % mu0 : ADMM parameter
mu1 = 0.2;                      % mu1 : ADMM parameter
[x, p, cost] = patv(y, d, lambda, Nit, mu0, mu1);

% Display cost function history

figure(1)
clf
plot(cost)
title('PATV algorithm - Cost function history');
xlabel('Iteration')
box off
```



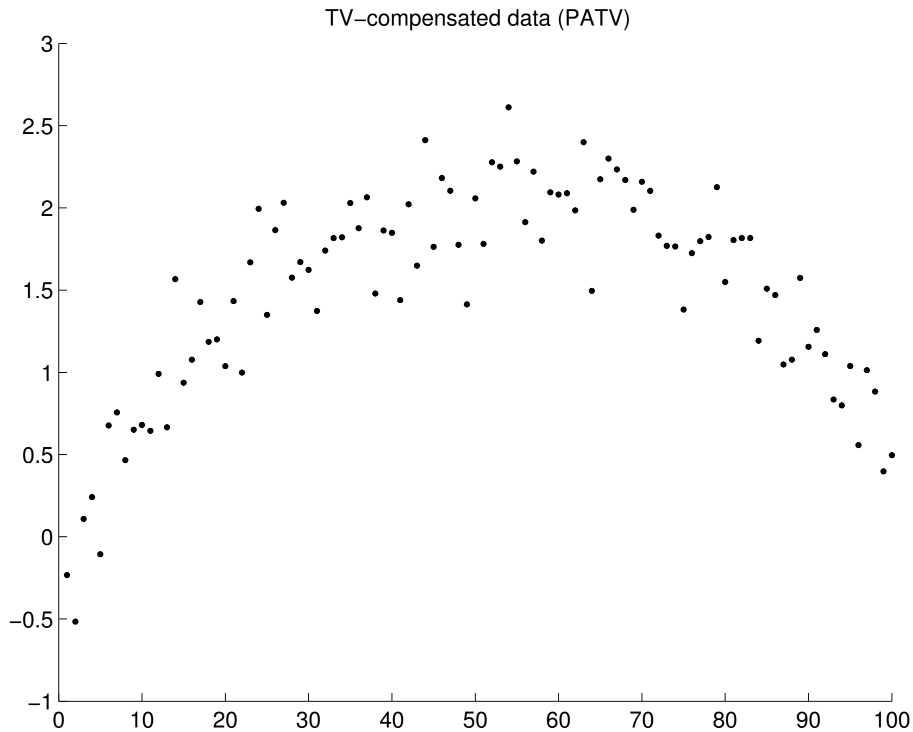
The PATV algorithm converges in about 50 iterations for this example. The parameters `mu0` and `mu1` affect the convergence rate.

```
% Plot calculated TV component

figure(1)
clf
subplot(2,1,1)
plot(n, x, 'k')
title('Calculated TV component (PATV)');
box off

% Display first-order difference of TV component

subplot(2,1,2)
stem(abs(diff(x)),'marker','none')
title('|diff(x)|');
box off
% There are 3 non-zero values
```



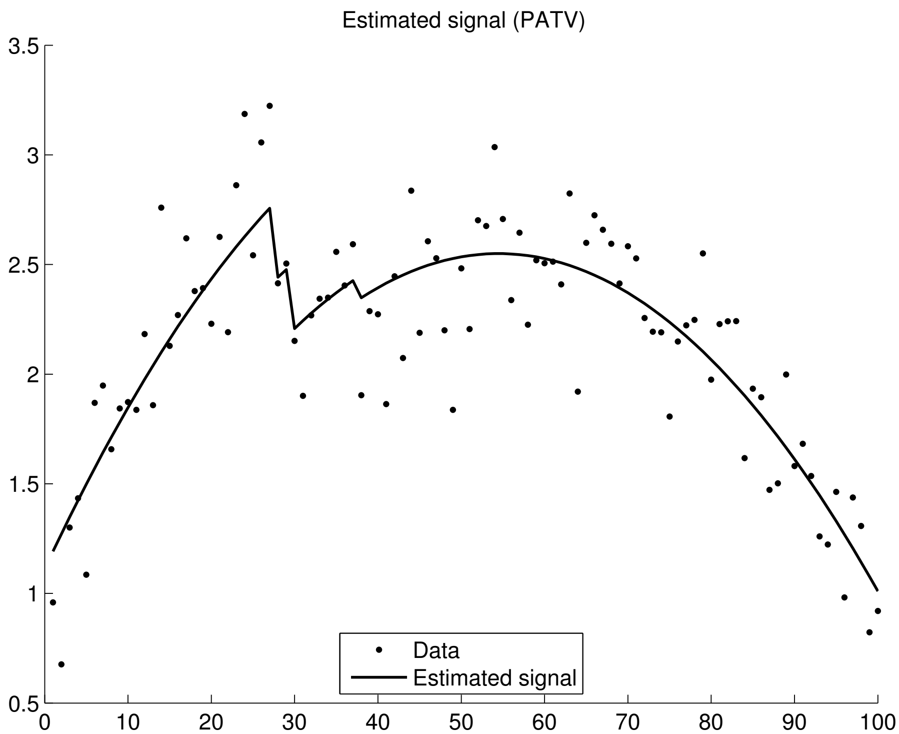
The calculated TV component has three discontinuities. Its first-order difference is sparse.

```
% Plot TV-compensated data

figure(1)
clf
plot(n, y - x, '.k')
title('TV-compensated data (PATV)');
box off
```



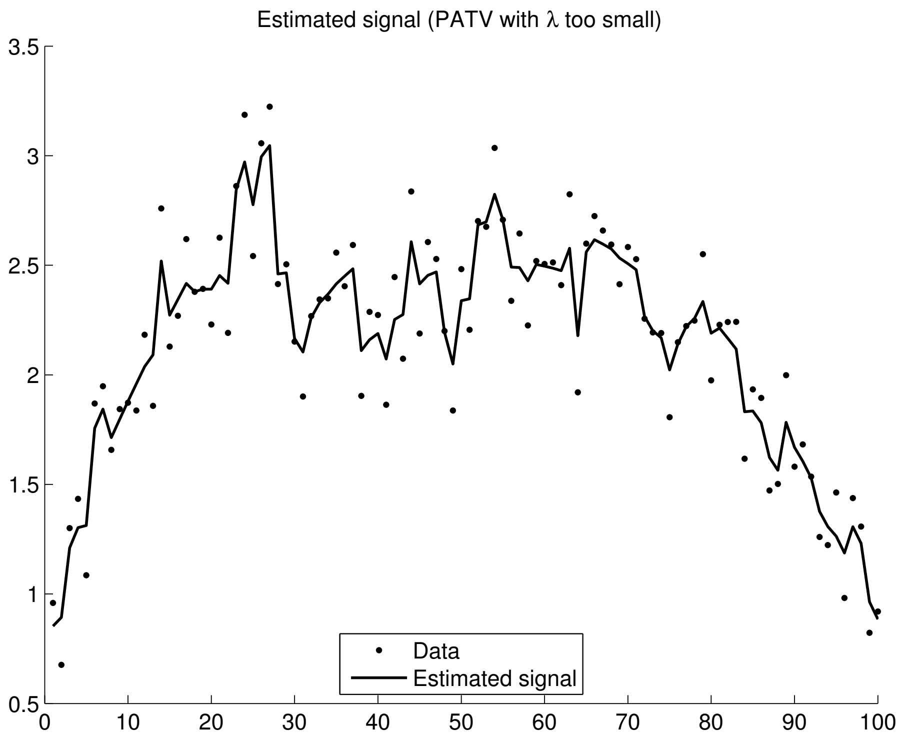
The TV-compensated data can be better approximated by a low-order polynomial than the original data.

```
% Display estimated signal

figure(1)
clf
plot(n, y,'.k', n, x+p, 'black')
title('Estimated signal (PATV)');
legend('Data','Estimated signal', 'Location','south')
box off
```



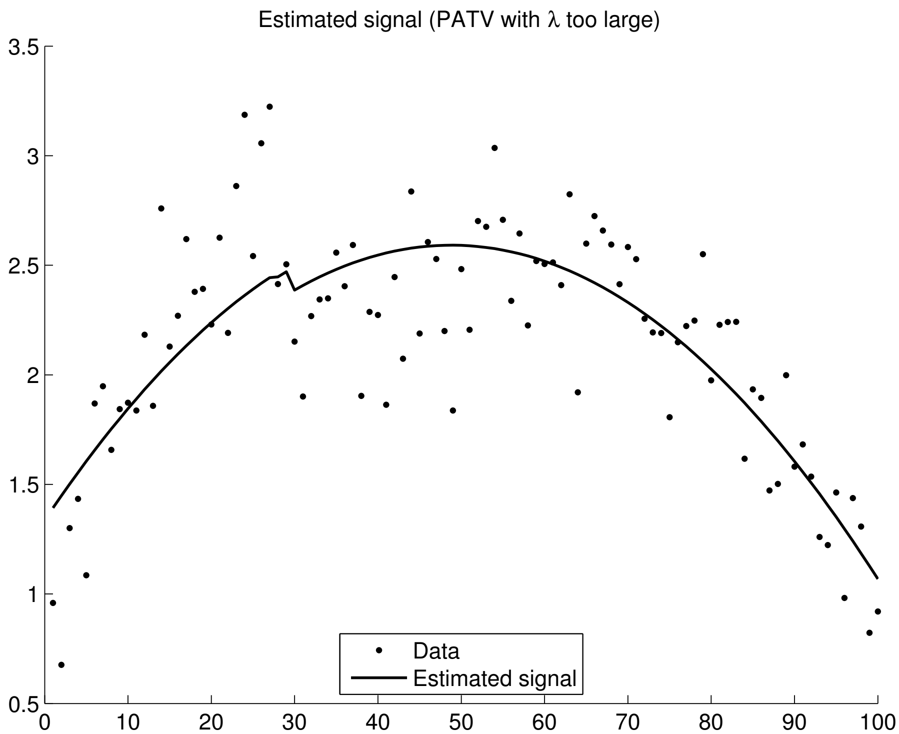
The output of the PATV algorithm is the sum of a low-order polynomial signal and the calculated TV component. It is smooth except for a small number of discontinuities.

```
% Lambda too small
% When lambda is too small, the noise is not fully reduced

lambda = 0.2;
[x, p, cost] = patv(y, d, lambda, Nit, mu0, mu1);

figure(1)
clf
plot(n, y,'.k', n, x+p, 'black')
title('Estimated signal (PATV with  \lambda too small)');
legend('Data','Estimated signal', 'Location','south')
box off
```



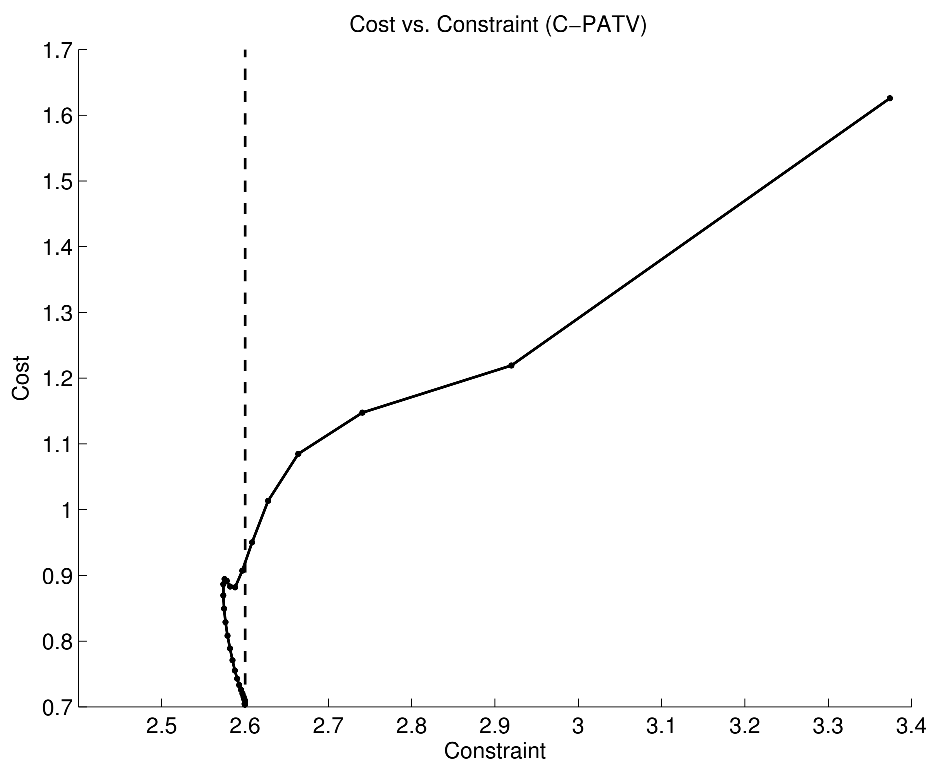
When λ is too small, the result of the PATV algorithm is noisy.

```
% Lambda too large
% When lambda is too large, the step discontinuity is under-estimated

lambda = 7;
[x, p, cost] = patv(y, d, lambda, Nit, mu0, mu1);

figure(1)
clf
plot(n, y,'.k', n, x+p, 'black')
legend('Data','Estimated signal', 'Location','south')
title('Estimated signal (PATV with  \lambda too large)');
box off
```



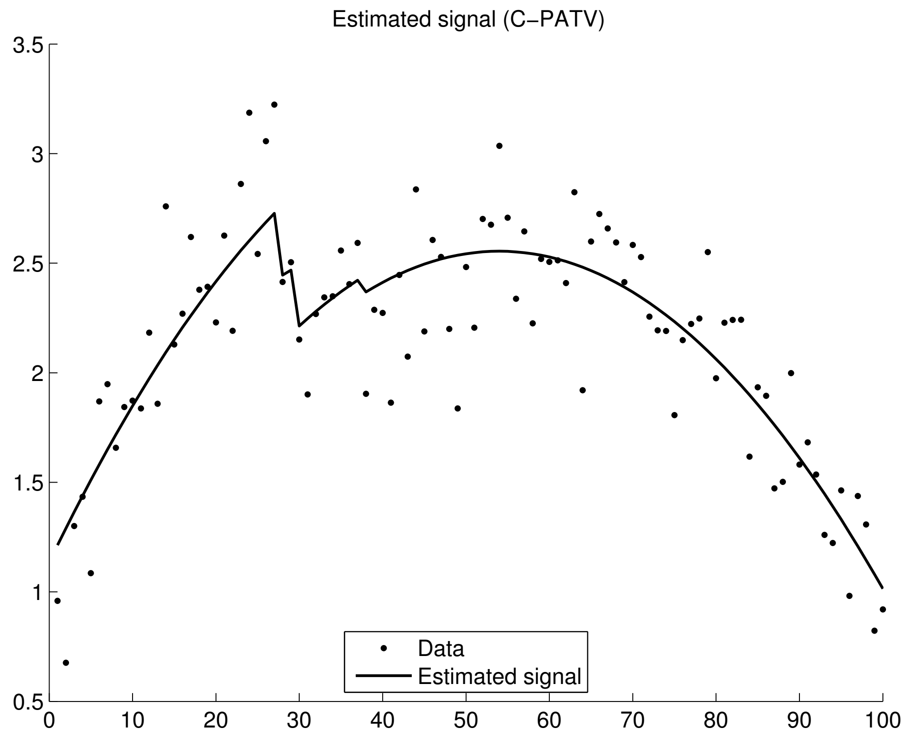
When λ is too large, the result of the PATV algorithm under-estimates the additive step discontinuity.

```
% Run C-PATV algorithm - Constrained formulation of PATV

r = sqrt(N) * sqrt(sum((1/N)*(noise.^2)));      % r : constraint parameter
Nit = 50;                            % Nit : number of iterations
mu0 = 3.5;                           % mu0 : ADMM parameter
mu1 = 0.5;                           % mu1 : ADMM parameter
[x_constr, p_constr, cost, constr] = cpatv(y, d, r, Nit, mu0, mu1);

% check constraint:
e  =  y - x_constr -  p_constr;
sqrt((1/N) * sum(e.^2))              % This value should be r/sqrt(N).

% Cost function versus constraint function histories
figure(1)
clf
plot(constr, cost,'.-');
xlabel('Constraint')
ylabel('Cost')
title('Cost vs. Constraint (C-PATV)')
ax3 = axis;
line([1 1]*r, ax3(3:4), 'linestyle', '--')
axis(ax3)
box off
```



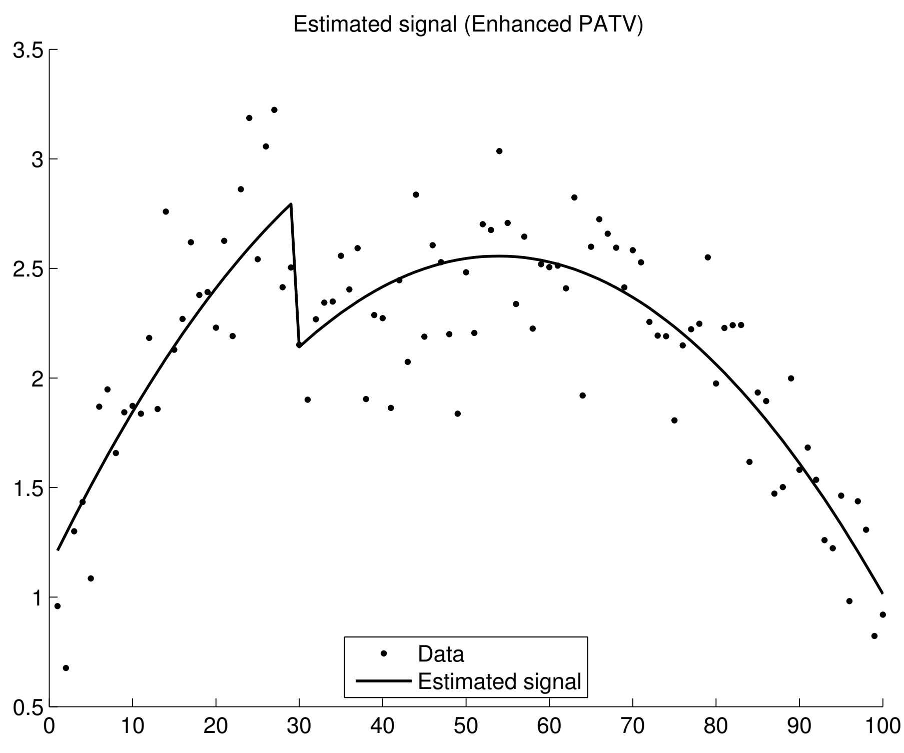
The C-PATV algorithm converges to a solution satisfying the constraint.

```
% Display estimated signal (C-PATV)

figure(1)
clf
plot(n, y,'.k', n, x_constr+p_constr, 'black') % , 'MarkerSize',MS)
title('Estimated signal (C-PATV)');
legend('Data','Estimated signal', 'Location','south')
box off
```



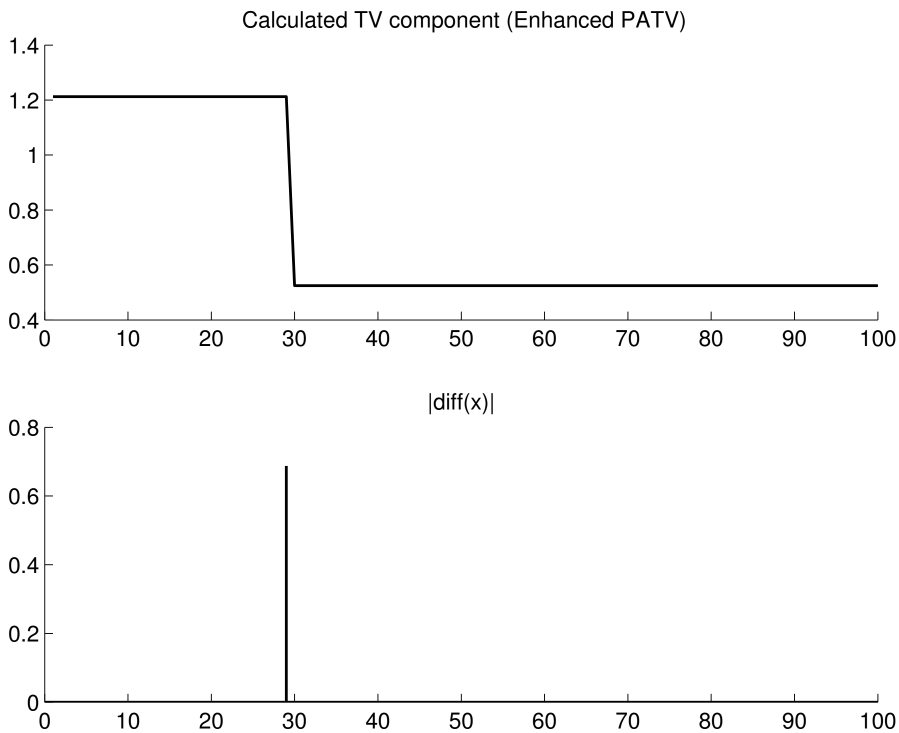
The result of C-PATV is like PATV, but the problem formulation is different.

```
% Enhanced PATV
% Lp quasi-norm minimization with p < 1 leads to fewer extraneous steps in the
% estimated signal

lambda = 3;                     % lambda : regularization parameter

Nit = 100;                      % Nit : number of iterations
mu0 = 20;                       % mu0 : ADMM parameter
mu1 = 0.2;                      % mu1 : ADMM parameter

p = 0.7;                        % p : power (Lp quasi-norm)
E = 0.02;                       % E : small number

[x, p, cost] = patv_Lp(y, d, lambda, p, E, Nit, mu0, mu1);

figure(1), clf
plot(n, y,'.k', n, x+p, 'black')
title('Estimated signal (Enhanced PATV)');
legend('Data','Estimated signal', 'Location','south')
box off
```



The enhanced PATV algorithm (using ℓ_p quasi-norm minimization) has fewer extraneous discontinuities.

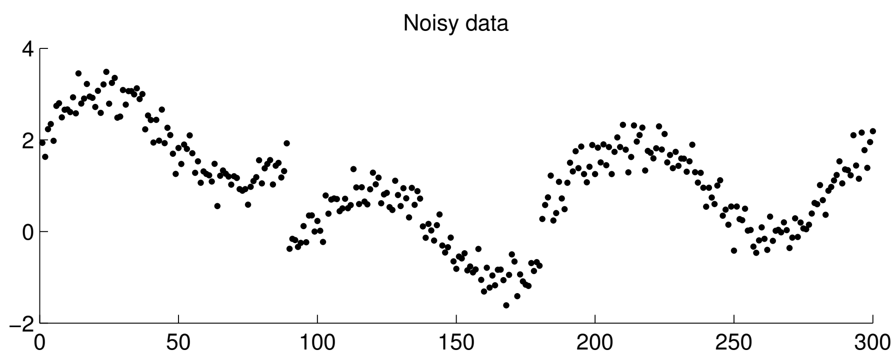
```
% Ehnahced PATV: There are fewer extraneous steps in the calculated TV component.

figure(1)
subplot(2,1,1)
plot(x)
title('Calculated TV component (Enhanced PATV)')
box off

subplot(2,1,2)
stem(abs(diff(x)), 'marker','none')
hold off
title('|diff(x)|')
box off
```



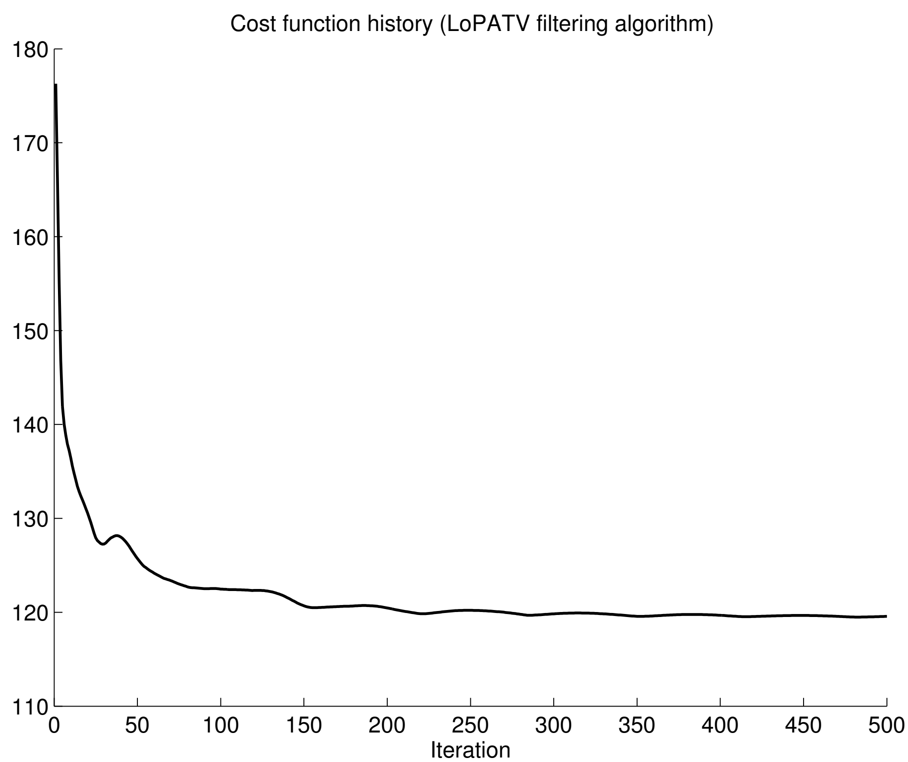
The first-order difference of the calculated TV component has one non-zero value here.

4 Example 2

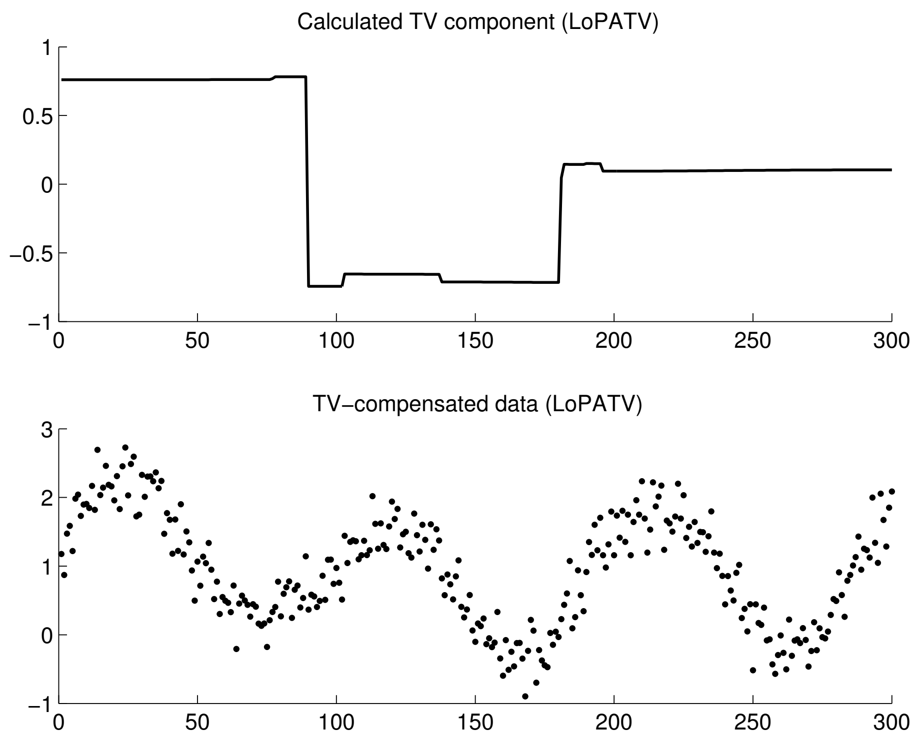
```
% Create simulated signal (smooth signal with additive step discontinuities)

N = 300;
n = (1:N)';

s1 = 2*(n < 0.3*N) + 1*(n > 0.6*N);          % step function
s2 = sin(0.021*pi*n);                         % smooth function
s = s1 + s2;                                  % total signal

randn('state',0);                             % Initialize randn so that example can be exactly reproduced
sigma = 0.3;
noise = sigma*randn(N,1);
y = s + noise;                                % noisy signal

figure(1)
clf
subplot(2,1,1)
plot(y,'.k')
title('Noisy data');
box off
```



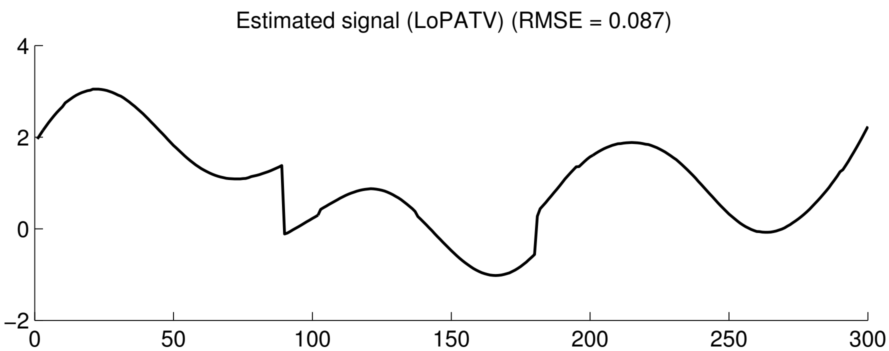
```
% Run LoPATV filtering algorithm (local polynomial approximation + TV filtering)

deg = 2;            % deg : degree of polynomial
P = 40;             % P : block overlap
L = 50;             % L : block length
lambda = 8;         % lambda : regularization parameter

(N-L)/(L-P)+1       % This is the number of blocks - it should be an
                    % integer, otherwise the data will be truncated

Nit = 500;          % Nit : number of iterations
mu0 = 10;           % mu0 : augmented Lagrangian parameter
mu = 1;             % mu : augmented Lagrangian parameter

[x, p, cost] = lopatv(y, L, P, deg, lambda, Nit, mu0, mu);    % Run the LoPATV algorithm

rmse_lopatv = sqrt(mean((s - x - p).^2));      % root-mean-square error

fprintf('LoPATV: RMSE = %f\n', rmse_lopatv)

% The cost function history flattens out as the algorithm converges.
figure(1)
clf
plot(cost)
title('Cost function history (LoPATV filtering algorithm)')
xlabel('Iteration')
box off
```

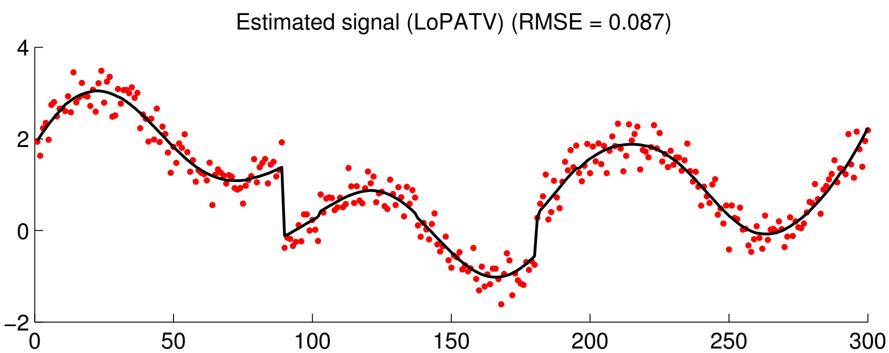


```
% Display calculated TV component

figure(1)
clf
subplot(2,1,1)
plot(x, 'black')
title('Calculated TV component (LoPATV)');
box off

% Display TV-compensated data

subplot(2,1,2)
plot(y - x, 'k.')
title('TV-compensated data (LoPATV)');
box off
```

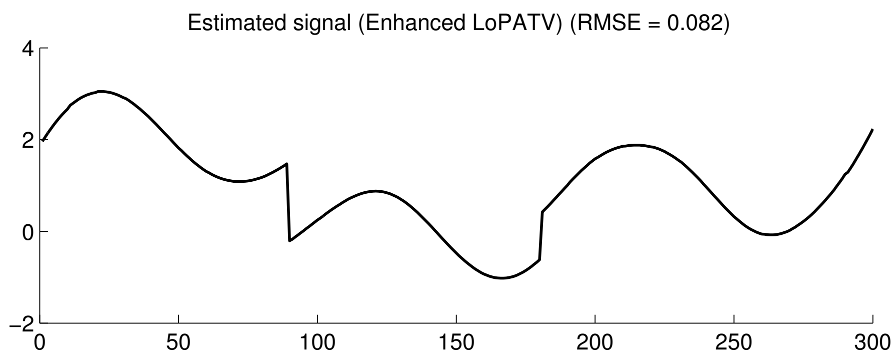


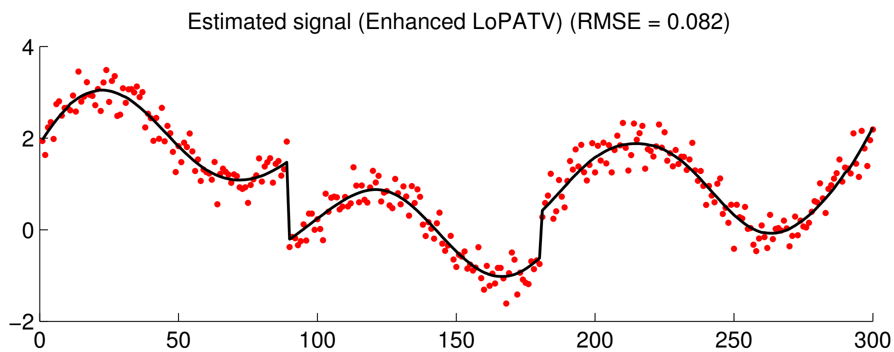
```
% Result of LoPATV filtering

txt = sprintf('Estimated signal (LoPATV) (RMSE = %.3f)', rmse_lopatv);

figure(1), clf
subplot(2,1,1)
plot(x + p, 'color','black')
title(txt)
box off

% Display with noisy data

subplot(2,1,2)
plot(n, y, '.r', n, x + p, 'black')
title(txt)
box off
```



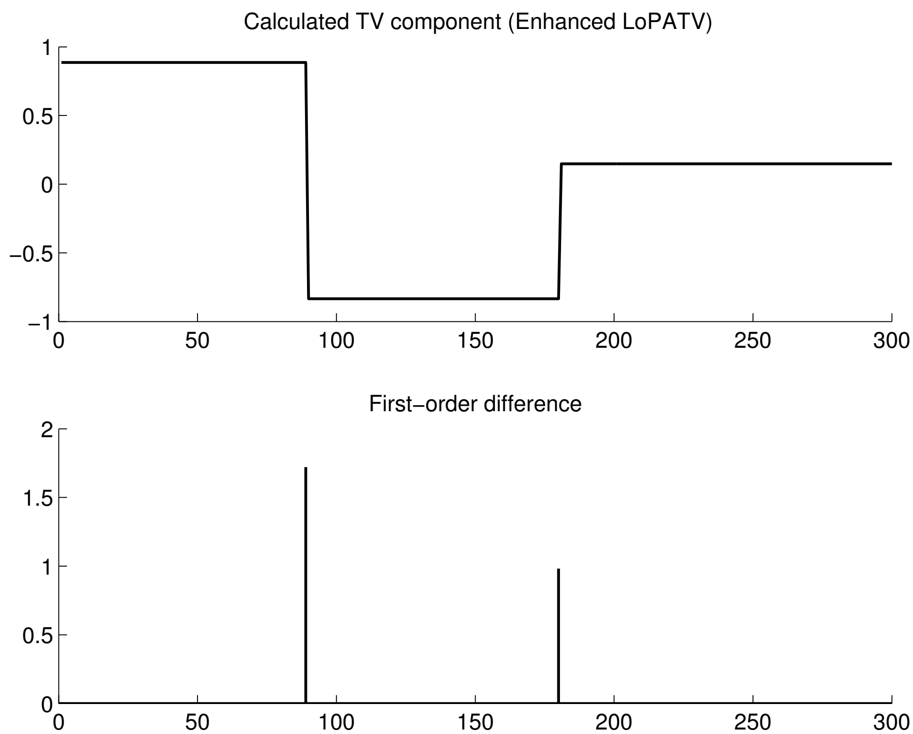


```
% Enhanced LoPATV - Lp quasi-norm minimization

mu0 = 50;
mu = .1;
Nit = 200;

pow = 0.7;
E = 0.05;

[x, p, cost] = lopatv_Lp(y, L, P, deg, lambda, Nit, mu0, mu, pow, E);

rmse_lopatv_Lp = sqrt(mean((s - x - p).^2));

fprintf('Enhanced LoPATV: RMSE = %.2e\n', rmse_lopatv_Lp)

% Display result
txt = sprintf('Estimated signal (Enhanced LoPATV) (RMSE = %.3f)', rmse_lopatv_Lp);
figure(1), clf
subplot(2,1,1)
plot(x + p, 'black')
title(txt)
box off

% Display result with noisy data
subplot(2,1,2)
plot(n, y,'.r', n, x + p, 'black')
title(txt)
box off
```





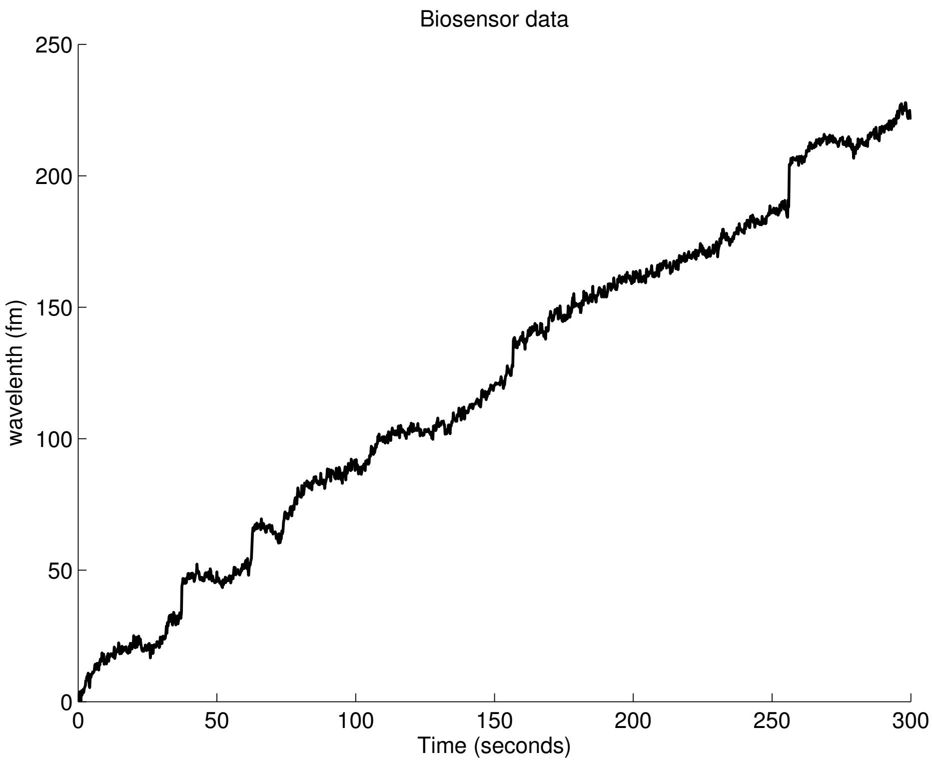
```
% Calculated TV component

figure(1)
clf
subplot(2,1,1)
plot(x, 'black')
title('Calculated TV component (Enhanced LoPATV)');
box off

% First-order difference

subplot(2,1,2)
stem(abs(diff(x)),'marker','none', 'color','black')
title('First-order difference');
box off
```



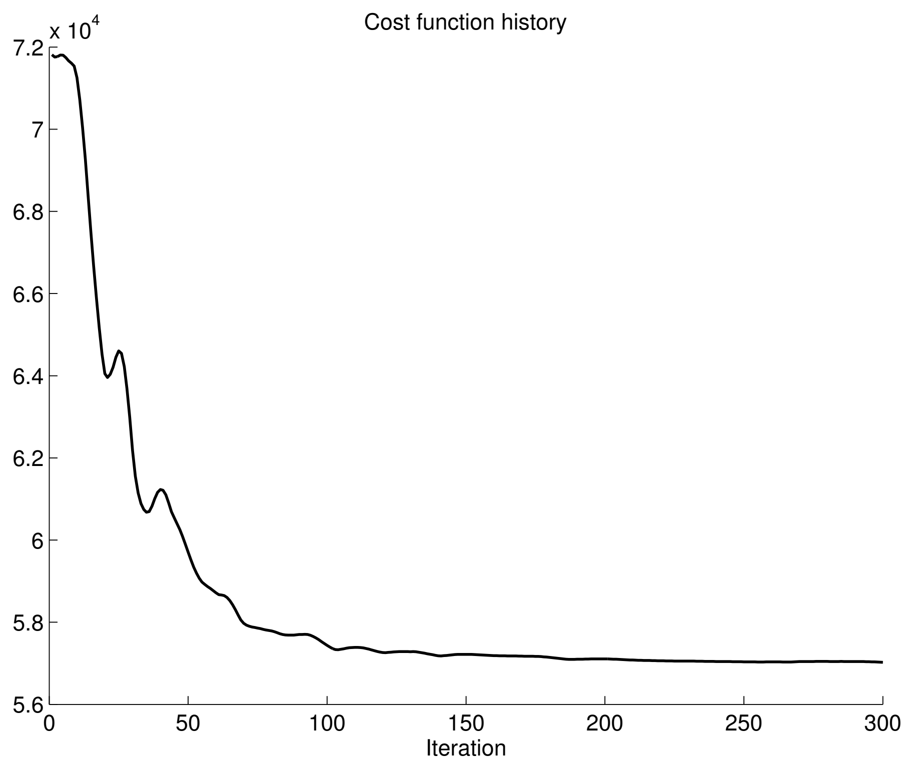
Using enhanced LoPATV, the calculated TV component has only two discontinuities.

5 Example 3

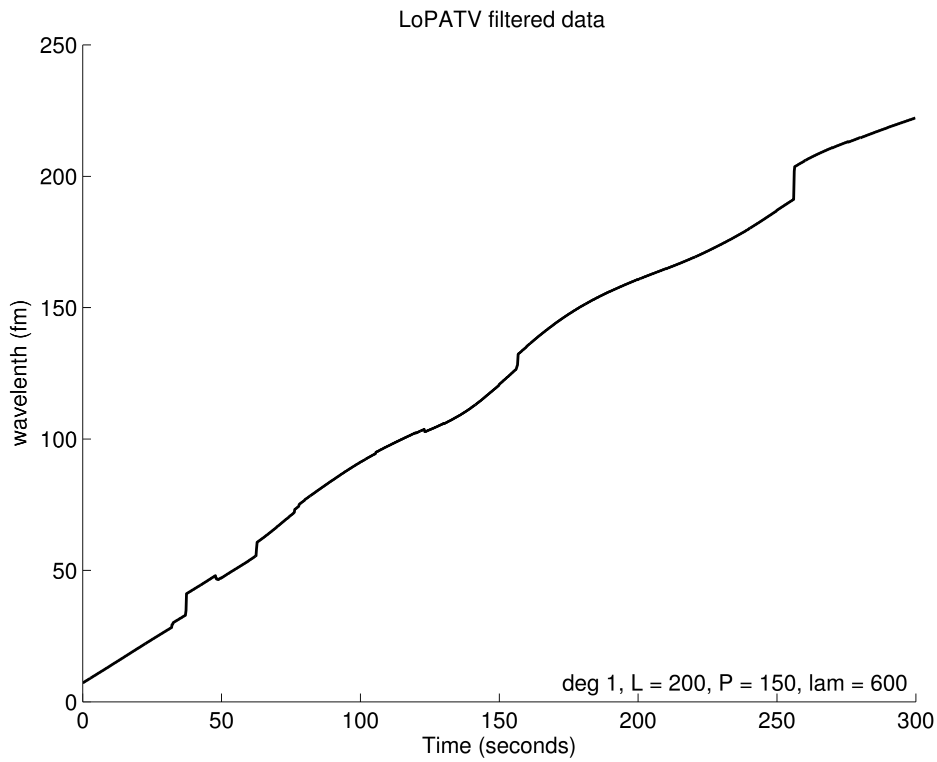
```
% Load data

load wgm_data.txt       % load WGM sensor data

y = wgm_data;
N = length(y);          % N : 1500
n = 1:N;
t = (0:N-1)/5;          % t : time axis (sampling rate is 5 samples/second)

figure(1)
clf
plot(t, y, 'black')
title('Biosensor data');
xlabel('Time (seconds)')
ylabel('wavelenth (fm)')
box off
```



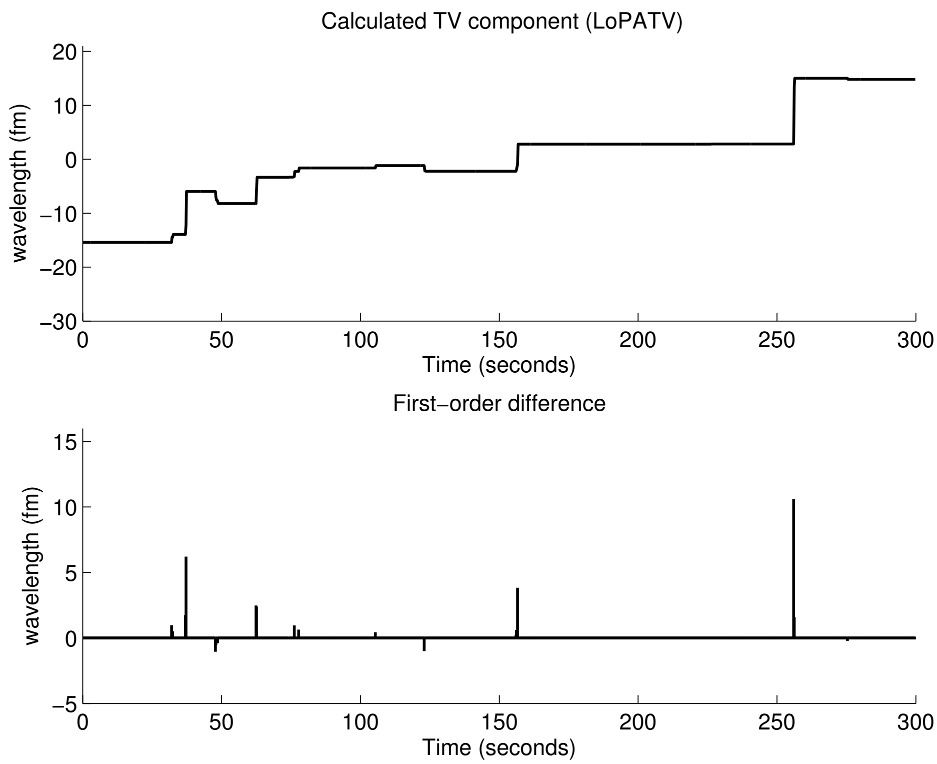
```
% LoPATV filtering
% Local polynomial approximation + total variation filter

lambda = 600;       % lambda : TV regularization parameter
L = 200;            % L : block length
P = 150;            % P : block overlap
deg = 1;            % deg : degree of polynomial

(N-L)/(L-P)+1       % This is the number of blocks - it should be an
                    % integer, otherwise the data will be truncated

mu0 = 500;
mu = .05;
Nit = 300;

[x, s, cost] = lopatv(y, L, P, deg, lambda, Nit, mu0, mu);
% x : TV component (approximate step signal)
% s : smooth low-pass signal
% cost : cost function history

figure(1)
plot(cost, 'black')
title('Cost function history')
xlabel('Iteration')
box off
```



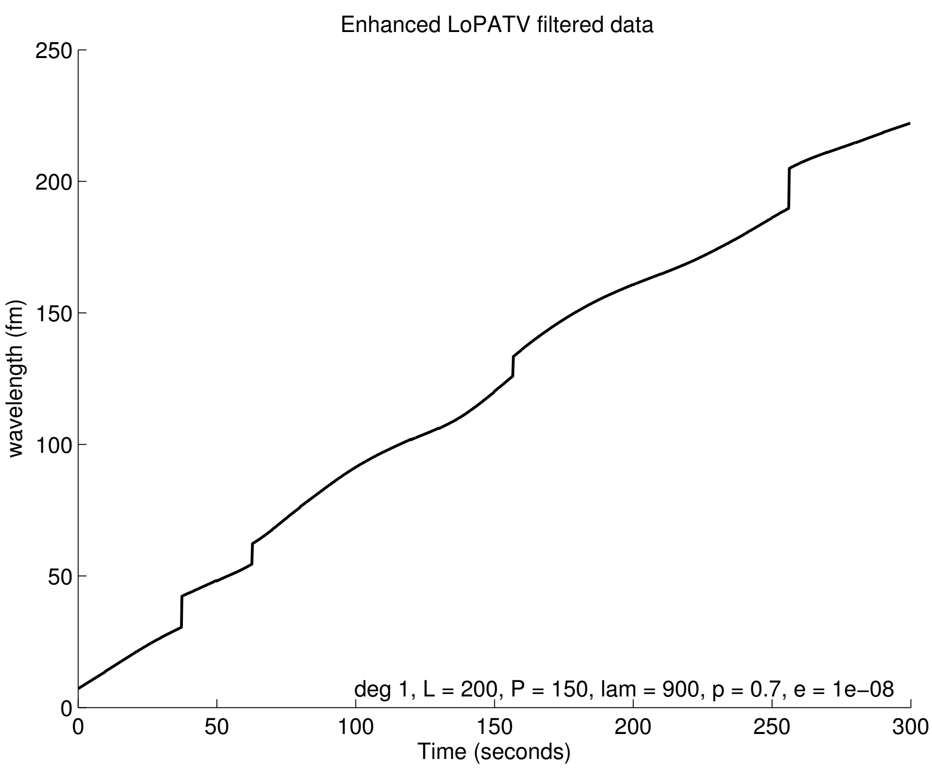
```
% Display filtered data

figure(1)
clf
plot(t, x+s, 'black')
xlabel('Time (seconds)')
ylabel('wavelenth (fm)')
title('LoPATV filtered data')
box off

txt = sprintf('deg %d, L = %d, P = %d, lam = %.f', deg, L, P, lambda);
text(0.99, 0.01, txt, 'units', 'normalized', ...
    'horizontalalignment', 'right', ...
    'verticalalignment', 'bottom', ...
    'fontsize', 12);
```



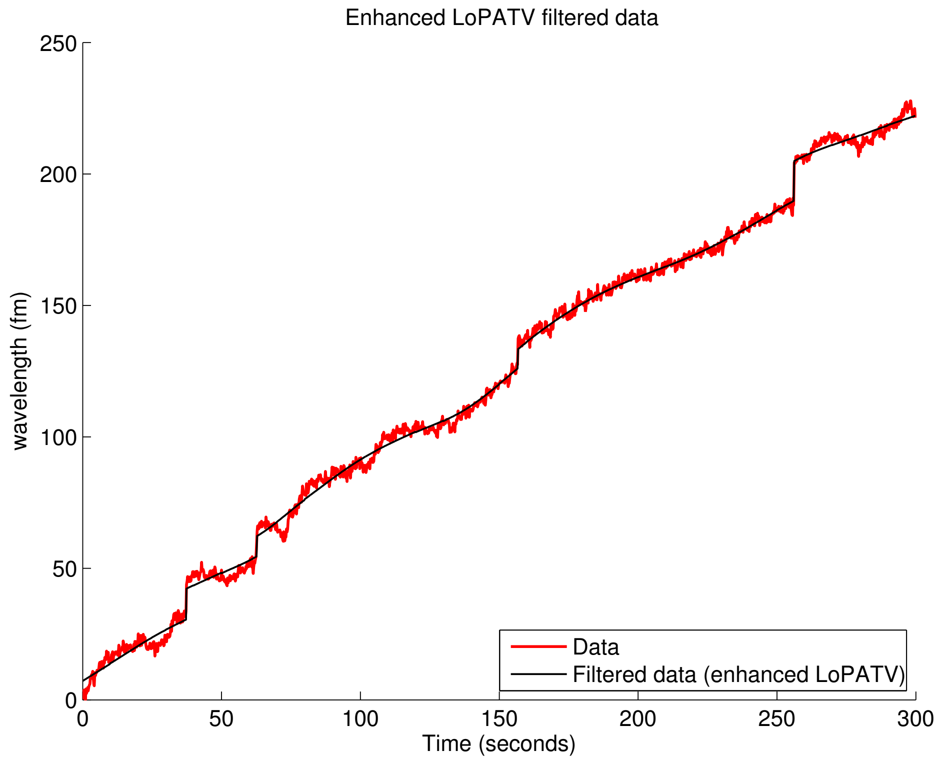
```
% Display step signal

figure(1)
clf
subplot(2,1,1)
plot(t, x, 'black')
title('Calculated TV component (LoPATV)');
ylim([-30 21])
ylabel('wavelength (fm)')
xlabel('Time (seconds)')
box off

subplot(2,1,2)
stem(t(1:end-1), diff(x), 'marker','none', 'color', 'black')
title('First-order difference');
ylim([-5 16])
ylabel('wavelength (fm)')
xlabel('Time (seconds)')
box off
```



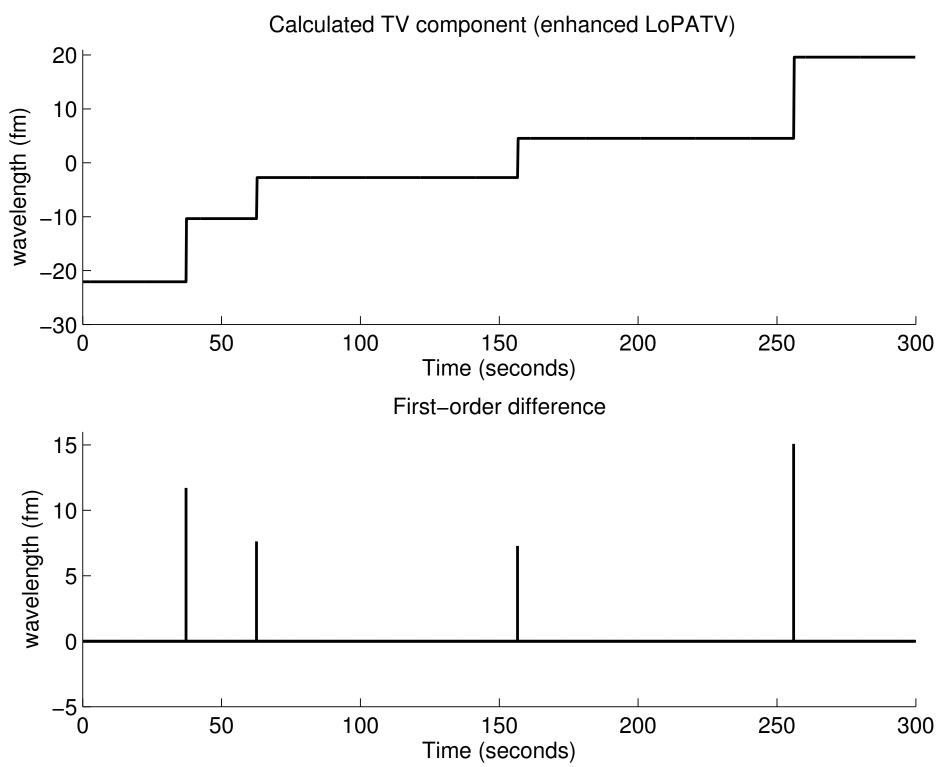
```
% Enhanced LoPATV filtering -- Lp quasi-norm minimization

p = 0.7;
E = 1e-8;
lambda = 900;

[x, s, cost] = lopatv_Lp(y, L, P, deg, lambda, Nit, mu0, mu, p, E);

figure(1)
clf
plot(t, s+x, 'black');
xlabel('Time (seconds)')
ylabel('wavelength (fm)')
title('Enhanced LoPATV filtered data')
box off

txt = sprintf('deg %d, L = %d, P = %d, lam = %.f, p = %.1f, e = %.2g', deg, L, P, lambda, p, E);
h = text(0.98, 0.01, txt, 'units', ...
    'normalized', 'horizontalalignment', ...
    'right', 'verticalalignment', ...
    'bottom', 'fontsize', 12);
```



```
% Display output of algorithm with data on same axis

figure(1)
clf
plot(t, y, 'color', 'red');
line(t, x+s,'linewidth',1,'color','black')
legend('Data','Filtered data (enhanced LoPATV)', 'location','southeast')
xlabel('Time (seconds)')
ylabel('wavelength (fm)')
title('Enhanced LoPATV filtered data')
box off
```



```
% Display step signal

figure(2)
clf
subplot(2,1,1)
plot(t, x, 'black')
title('x(t)');
ylim([-30 21])
ylabel('wavelength (fm)')
xlabel('Time (seconds)')
title('Calculated TV component (enhanced LoPATV)');
box off

subplot(2,1,2)
stem(t(1:end-1), diff(x),'marker','none', 'color', 'black')
title('First-order difference');
xlabel('Time (seconds)')
ylim([-5 16])
ylabel('wavelength (fm)')
box off
```



The enhanced LoPATV algorithm produces a result with fewer extraneous steps. Compare with the result of LoPATV on page 27.

6 Programs

```
function [x, p, cost] = patv(y, d, lambda, Nit, mu0, mu1)
% [x, p, cost] = patv(y, d, lambda, Nit, mu0, mu1)
% PATV: Simultaneous polynomial approximation and total variation
% filtering
%
% INPUT
%   y - noisy data
%   d - order of polynomial
%   lambda - regularization parameter
%   Nit - number of iterations
%   mu0, mu1 - augmented Lagrangian parameters
%
% OUTPUT
%   x - TV component
%   p - polynomial component
%   cost - cost function history

% Ivan Selesnick
% Polytechnic Institute of New York University
% December 2011
% Reference: Polynomial Smoothing of Time Series with Additive Step Discontinuities
% I. W. Selesnick, S. Arnold, and V. R. Dantham

y = y(:);                   % convert to column vector
cost = zeros(1,Nit);        % cost function history
N = length(y);

n = (0:N-1)';
G = zeros(N,d);
for k = 1:d, G(:,k) = n.^k;  end        % exclude dc term (included in TV component)
G = orth(G);                            % orthogonalize cols of G (so that G' G = I)

e = ones(N-1,1);
D = spdiags([-e e],[0 1],N-1,N);        % D : first-order difference (sparse matrix)
A = mu0*(D'*D) + mu1*speye(N);          % A : mu0 D'D + mu1 I (sparse matrix)
D = @(x) diff(x);                       % D (operator)
DT = @(x) [-x(1); -diff(x); x(end)];    % D' (operator)
H = @(x) x - G * (G'*x);

x = zeros(N,1);                         % initializations
d0 = zeros(N-1,1);
d1 = zeros(N,1);

for k = 1:Nit
    u0 = soft(D(x)+d0, 0.5*lambda/mu0);
    b = x + d1;
    u1 = ((y + mu1*b) + G*(G'*(b-y))) / (1+mu1);
    x = A \ (mu0*DT(u0-d0) + mu1*(u1-d1));         % sparse system solve
    d0 = d0 - (u0-D(x));
    d1 = d1 - (u1-x);
    cost(k) = sum(abs(lambda .* D(x))) + sum(abs(H(x-y)).^2);
```

```
end
p = G * (G' * (y - x));
```

```
function [x, p, cost, constr] = cpatv(y, d, r, Nit, mu0, mu1)
% [x, p, cost, constr] = cpatv(y, d, r, Nit, mu0, mu1)
% C-PATV: Simultaneous polynomial approximation and total variation filtering,
% constrained formulation: ||H(y-x)||_2 <= r
%
% INPUT
%   y - noisy data
%   d - order of polynomial
%   r - constraint parameter
%   Nit - number of iterations
%   mu0, mu1 - augmented Lagrangian parameters
%
% OUTPUT
%   x - TV component
%   p - polynomial component
%   cost - cost function history
%   constr - constraint function history

% Ivan Selesnick
% Polytechnic Institute of New York University
% December 2011
% Reference: Polynomial Smoothing of Time Series with Additive Step Discontinuities
% I. W. Selesnick, S. Arnold, and V. R. Dantham

y = y(:);                               % convert to column vector
cost = zeros(1,Nit);                    % cost function history
constr = zeros(1,Nit);                  % constraint function history
N = length(y);
n = (0:N-1)';
G = zeros(N,d);
for k = 1:d, G(:,k) = n.^k;  end        % exclude dc term (included in TV component)
G = orth(G);                            % orthogonalize cols of G (so that G' G = I)
e = ones(N-1,1);
D = spdiags([-e e],[0 1],N-1,N);        % D (sparse matrix)
A = mu0*(D'*D) + mu1*speye(N);          % mu0 D'D + mu1 I (sparse matrix)
D = @(x) diff(x);                       % D (operator)
DT = @(x) [-x(1); -diff(x); x(end)];    % D' (operator)
H = @(x) x - G * (G'*x);                % H = I - G'*G (operator)
F = (1/mu1)*eye(d) - (G' * (A \ G));    % F (sparse matrix solve)
FG = F\G';
x = zeros(N,1);                         % initializations
d0 = zeros(N-1,1);
d1 = zeros(N,1);
Hy = H(y);
for k = 1:Nit
    u0 = soft(D(x) + d0, 0.5/mu0);
    u1 = projball(H(x) + d1, Hy, r);
    b = A \ (mu0*DT(u0-d0) + mu1*H(u1-d1));     % sparse matrix solve
    x = b + A \ ( G * (FG * b));                % sparse matrix solve
    d0 = d0 - (u0 - D(x));
    d1 = d1 - (u1 - H(x));
```

```
    cost(k) = sum(abs(D(x)));
    constr(k) = sqrt(sum(abs(H(x-y)).^2));
end
p = G * (G' * (y - x));
```

```
function [x, p, cost] = lopatv(y, L, P, deg, lambda, Nit, mu0, mu)
% [x, p, cost] = lopatv(y, L, P, deg, lambda, Nit, mu0, mu)
% LoPATV: Simultaneous local polynomial approximation and total variation filtering
% (sliding window with overlapping)
%
% INPUT
%   y - noisy data
%   L - block length
%   P - overlapping (number of samples common to adjacent blocks)
%   deg - polynomial degree
%   lambda - regularization parameter
%   Nit - number of iterations
%   mu0, mu - augmented Lagrangian parameters
%
% OUTPUT
%   x - step function (TV component)
%   p - local polynomial component
%   cost - cost function history
%
% Number of blocks = (length(y)-L)/(L-P)+1
% If this is not an integer, then input signal y will be truncated.

% Ivan Selesnick
% Polytechnic Institute of New York University
% December 2011
% Reference: Polynomial Smoothing of Time Series with Additive Step Discontinuities
% I. W. Selesnick, S. Arnold, and V. R. Dantham

y = y(:);                               % convert to column vector
N = length(y);

M = (N-L)/(L-P);                        % M : number of blocks - 1
if M > floor(M)
    N = floor(M)*(L-P)+L;
    y = y(1:N);
    fprintf('Note in lopatv.m: The input signal will be truncated down to length %d\n',N)
    fprintf('so it is consistent with the block length %d and overlap %d.\n', L, P);
end

cost = zeros(1,Nit);                    % cost function history
G = zeros(L,deg+1);
for k = 0:deg, G(:,k+1) = (0:L-1)'.^k; end
G = orth(G);                            % orthogonalize columns of G (so that G' G = I)
H = @(x) x - G * (G'*x);
buff = @(x) buffer(x, L, P, 'nodelay');
s = invbuffer(buff(ones(1,N)), P);

e = ones(N-1,1);
D = spdiags([-e e],[0 1],N-1,N);                % D (sparse matrix)
```

```
A = mu0*(D'*D) + mu*spdiags(s,0,N,N);           % mu0 D'D + mu S (sparse matrix)
D = @(x) diff(x);                               % D (operator)
DT = @(x) [-x(1); -diff(x); x(end)];            % D' (operator)
x = zeros(size(y));                             % initialization
yb = buff(y);
xb = buff(x);
d0 = zeros(N-1,1);
d = zeros(size(xb));

for it = 1:Nit
    u0 = soft(D(x)+d0, 0.5*lambda/mu0);
    b = xb + d;
    u = ((yb + mu*b) + G*(G'*(b-yb))) / (1+mu);
    x = A \ (mu0*DT(u0-d0) + mu*invbuffer(u-d, P));    % sparse system solve
    xb = buff(x);
    d0 = d0 - (u0-D(x));
    d = d - (u-xb);
    cost(it) = sum(abs(lambda .* D(x))) + sum(sum(abs(H(buff(y-x))).^2));
end
p_blocks = G * (G' * buff(y - x));

w = hamming(L);                     % column vector
p = invbuffer(p_blocks, P, w);      % compute local polynomial estimate
```

```
function [x,p,cost] = patv_Lp(y, d, lambda, p, E, Nit, mu0, mu1)
% [x,p,cost] = patv_Lp(y, d, lambda, p, E, Nit, mu0, mu1)
% Enhanced PATV: Simultaneous polynomial approximation and total variation
% filtering
%   Regularization : lambda * sum((abs(diff(x)) + E).^p);
%
% INPUT
%   y - noisy data
%   d - order of polynomial
%   lambda - regularization parameter
%   p, E - Lp norm
%   Nit - number of iterations
%   mu0, mu1 - Augmented Lagrangian parameters
%
% OUTPUT
%   x - TV component
%   p - polynomial component
%   cost - cost function history

% Ivan Selesnick
% Polytechnic Institute of New York University
% December 2011
% Reference: Polynomial Smoothing of Time Series with Additive Step Discontinuities
% I. W. Selesnick, S. Arnold, and V. R. Dantham

y = y(:);                   % convert to column vector
N = length(y);

n = (0:N-1)';
G = zeros(N,d);
```

```
for k = 1:d, G(:,k) = n.^k;  end        % exclude dc term (included in TV component)
G = orth(G);                            % orthogonalize cols of G (so that G' G = I)

e = ones(N-1,1);
D = spdiags([-e e],[0 1],N-1,N);        % D (sparse matrix)
A = mu0*(D'*D) + mu1*speye(N);          % mu0 D'D + mu1 I (sparse matrix)
D = @(x) diff(x);                       % D (operator)
DT = @(x) [-x(1); -diff(x); x(end)];    % D' (operator)
H = @(x) x - G * (G'*x);

x = zeros(N,1);                         % initializations
d0 = zeros(N-1,1);
d1 = zeros(N,1);

M = 15; % M : number of outer iterations
cost = zeros(M,Nit);    % cost function history
b = lambda;             % initialize

for i = 1:M
    for k = 1:Nit
        u0 = soft(D(x)+d0, 0.5*b/mu0);
        tmp = x + d1;
        u1 = ((y + mu1*tmp) + G*(G'*(tmp-y))) / (1+mu1);
        x = A \ (mu0*DT(u0-d0) + mu1*(u1-d1));         % sparse system solve
        d0 = d0 - (u0-D(x));
        d1 = d1 - (u1-x);
        cost(i,k) = sum(abs(b .* D(x))) + sum(abs(H(x-y)).^2);
    end
    b = lambda * p * (abs(diff(x)) + E).^(p-1);
end
p = G * (G' * (y - x));
```

```
function [x, p, cost] = lopatv_Lp(y, L, P, deg, lambda, Nit, mu0, mu, pow, E)
% [x, p, cost] = lopatv_Lp(y, L, P, deg, lambda, Nit, mu0, mu, pow, E)
% LoPATV_Lp: Enhanced local polynomial approximation and total variation filtering
% (sliding window with overlapping) with Lp norm
%
% INPUT
%   y - noisy data
%   L - block length
%   P - overlapping (number of samples common to adjacent blocks)
%   deg - polynomial degree (1, 2, 3)
%   lambda - regularization parameter
%   Nit - number of iterations
%   mu - augmented Lagrangian parameters
%   pow - power (Lp norm)
%   E - small number
%
% OUTPUT
%   x - TV component
%   p - local polynomial component
%   cost - cost function history
%
% Number of blocks = (length(y)-L)/(L-P)+1
```

```
% If this is not an integer, then input signal y will be truncated.

% Ivan Selesnick
% Polytechnic Institute of New York University
% December 2011
% Reference: Polynomial Smoothing of Time Series with Additive Step Discontinuities
% I. W. Selesnick, S. Arnold, and V. R. Dantham

y = y(:);                               % convert to column vector
N = length(y);

M = (N-L)/(L-P);                        % M : number of blocks - 1
if M > floor(M)
    N = floor(M)*(L-P)+L;
    y = y(1:N);
    fprintf('Note in lopatv.m: The input signal will be truncated down to length %d\n',N)
    fprintf('so it is consistent with the block length %d and overlap %d.\n', L, P);
end

cost = zeros(1,Nit);                    % cost function history
G = zeros(L,deg+1);
for k = 0:deg, G(:,k+1) = (0:L-1)'.^k; end
G = orth(G);                            % orthogonalize columns of G (so that G' G = I)
H = @(x) x - G * (G'*x);
buff = @(x) buffer(x, L, P, 'nodelay');
s = invbuffer(buff(ones(1,N)), P);

e = ones(N-1,1);
D = spdiags([-e e],[0 1],N-1,N);                % D (sparse matrix)
A = mu0*(D'*D) + mu*spdiags(s,0,N,N);           % mu0 D'D + mu S (sparse matrix)
D = @(x) diff(x);                               % D (operator)
DT = @(x) [-x(1); -diff(x); x(end)];            % D' (operator)
x = zeros(size(y));                             % initialization
yb = buff(y);
xb = buff(x);
d0 = zeros(N-1,1);
d = zeros(size(xb));

M = 5; % M : number of outer iterations (increase, if nec.)
cost = zeros(M,Nit);    % cost function history
b = lambda;             % initialize

for i = 1:M
    for k = 1:Nit
        u0 = soft(D(x)+d0, 0.5*b/mu0);
        tmp = xb + d;
        u = ((yb + mu*tmp) + G*(G'*(tmp-yb))) / (1+mu);
        x = A \ (mu0*DT(u0-d0) + mu*invbuffer(u-d, P));    % sparse system solve
        xb = buff(x);
        d0 = d0 - (u0-D(x));
        d = d - (u-xb);
        cost(i,k) = sum(abs(b .* D(x))) + sum(sum(abs(H(buff(y-x))).^2));
    end
    b = lambda * pow * (abs(diff(x)) + E).^(pow-1);
    % plot(1./b,'k'),  drawnow
end
```

```
p_blocks = G * (G' * buff(y - x));

w = hamming(L);                     % column vector
p = invbuffer(p_blocks, P, w);      % compute local polynomial estimate
```

```
function y = soft(x, T)
% Soft-threshold function (real or complex x)
% y = soft(x, T)
% Input
%    x : data
%    T : threshold
%
% If x and T are both multidimensional, then they must be of the same size.

y = max(1 - T./abs(x), 0) .* x;
```

```
function v = projball(x, y, r);

R = sqrt( sum(abs(x(:) - y(:)).^2) );
if R <= r
    v = x;
else
    v = y + (r/R) * (x-y);
end
```

The MATLAB `buffer` function in the Signal Processing ToolBox is required for the LoPATV algorithms. In addition, we need an inverse for the `buffer` function. As an inverse function is not provided in MATLAB, we need the following function `invbuffer` which inverts the MATLAB `buffer` function.

```
function x = invbuffer(X, P, w)
% x = invbuffer(X, P)
% inverts the buffer function
%
% x = invbuffer(X, P, w)
% Multiplies each frame by window w
%
% Each column of X is one block of x
% P : overlap

% Ivan Selesnick
% Polytechnic Institute of New York University
% December 2011

[L, M] = size(X);
% L : length of block
% M : number of blocks

x = zeros(L+(M-1)*(L-P),1);

if nargin < 3
```

```
    for i = 1:M
        x((i-1)*(L-P)+(1:L)) = x((i-1)*(L-P)+(1:L)) + X(:,i);
    end
else
    s = x; % zeros
    w = w(:);
    for i = 1:M
        x((i-1)*(L-P)+(1:L)) = x((i-1)*(L-P)+(1:L)) + w .* X(:,i);
        s((i-1)*(L-P)+(1:L)) = s((i-1)*(L-P)+(1:L)) + w;
    end
    x = x./s;
end
```