Simultaneously estimating differential delay and differential attenuation

Bill Menke, March 2017

Suppose that a displacement pulse $u_2(t)$ is delayed and attenuated with respect to displacement pulse $u_1(t)$. The attenuation is assumed to follow a standard Azimi model, parameterized by t^*, and the delay is parameterized by a delay time t_0. The Azimi model is chosen because it is self-consistent and because it approximately corresponds to a constant quality factor. Becase it is causal, the Azimi model has its own built-in delay; t_0 represents an additional delay.

We build an filter $f(t, t^*, t_0)$ that combines the Azimi attenuation operator with a delay operator. We then use a grid search to find the (t^*, t_0) that best solves:

$$u_2^{obs}(t) = u_2^{pre}(t) \quad \text{with} \quad u_2^{pre}(t) = f(t, t^*, t_0) * u_1^{obs}(t)$$

where $*$ is convolution. The error is defined as $E(t^*, t_0) = \left\| u_2^{obs}(t) - u_2^{pre}(t) \right\|_2^2$. Tests (see below) indicate that the this procedure is very effective in recovering (t^*, t_0).

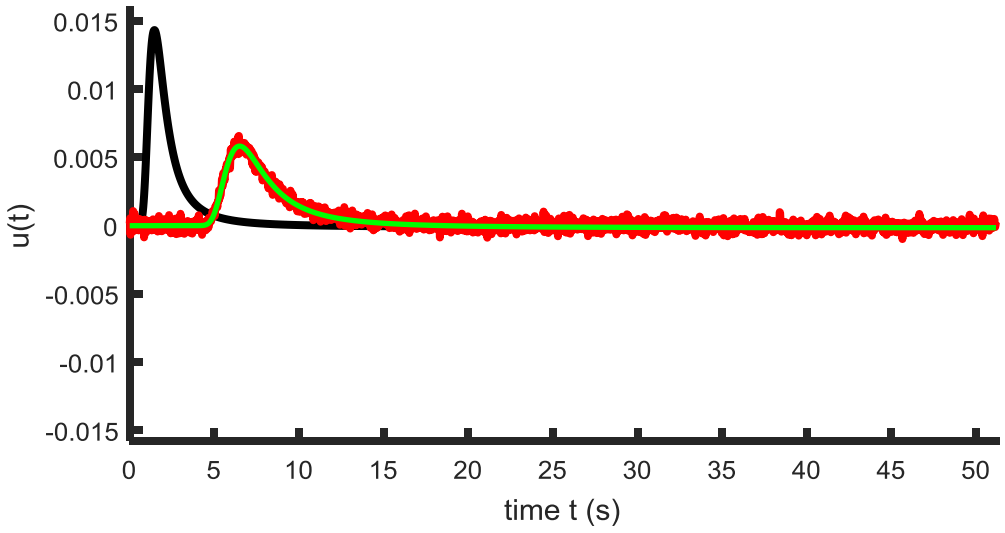

Figure 1. The less attenuated pulse $u_1(t)$ (black) and the delayed and more attenuated pulse $u_2(t)$ (green). Random noise has added to $u_2(t)$ (red) to simulate observational error.

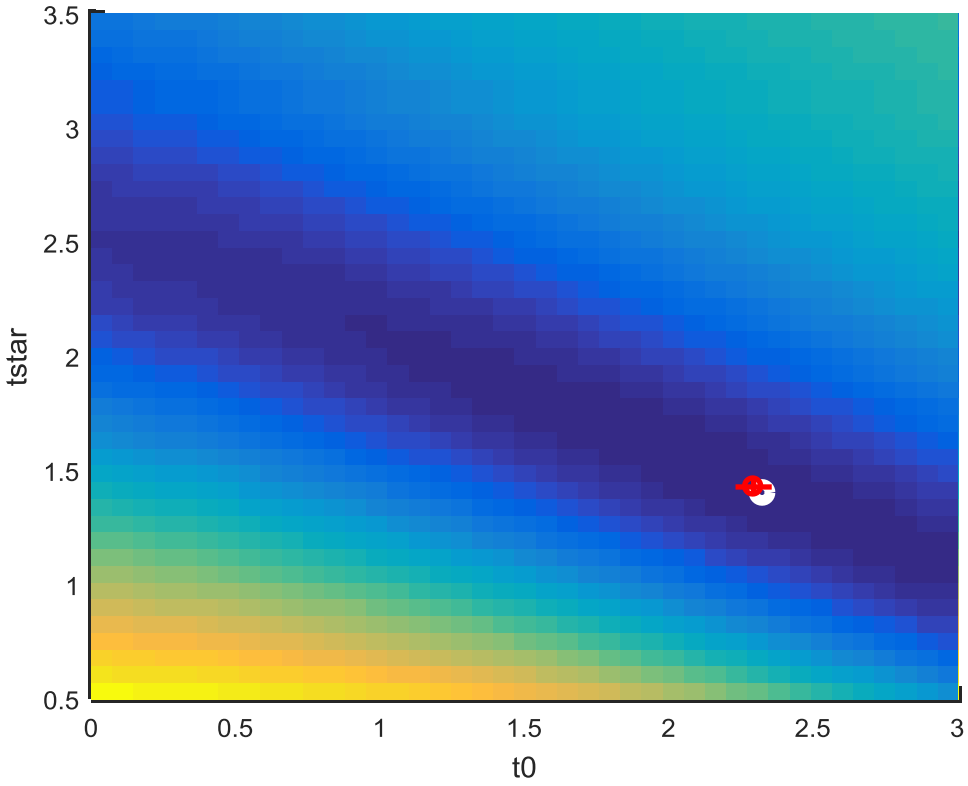

Figure 1. Error surface $E(t^*, t_0)$ (colors) with the true solution $(t^*, t_0)^{true} = (1.2, 2.4)$ (white circle), Estimated solution (red circle) and error bars (red bars).

```
clear all;

N = 2048; % number of samples
N2 = 1024;
Dt = 0.025; % sampling interval
f0 = 50; % Azimi corner frequency

% construct the less-attenuated data
tstar1 = 1.00;
t01 = 0.0;
[ t, pulse0, pulse1, f, Qf1, cw1 ] = azimi( N, Dt, tstar1, 1, 1, f0 );
pulse1 = delay(pulse1,Dt,t01-N2*Dt);
pulse1 = real(pulse1);
pulse1 = chebyshevfilt(pulse1, Dt, 0.001, 1.0);

% construct the more-attenuated data
tstar2 = 2.40;
t02 = 2.4;
[ t, pulse0, pulse2, f, Qf2, cw2 ] = azimi( N, Dt, tstar2, 1, 1, f0 );
pulse2 = delay(pulse2,Dt,t02-N2*Dt);
pulse2 = real(pulse2);
```

```
pulse2 = chebyshevfilt(pulse2, Dt, 0.001, 1.0);
sigmad = 0.05*max(abs(pulse2));
pulse2 = pulse2+random('Normal',0,sigmad,N,1);

t0true = t02-t01;
tstartrue = tstar2-tstar1;

figure(1);
clf;
set(gca,'LineWidth',3);
hold on;
amp = max(abs(pulse1));
axis( [t(1), t(end), -1.1*amp, 1.1*amp ] );
plot( t, pulse1, 'k-', 'LineWidth', 3 );
plot( t, pulse2, 'r-', 'LineWidth', 3 );
xlabel('time t (s)');
ylabel('u(t)');

Ntstar = 41;
tstarmin = 0.5;
tstarmax = 3.5;
Dtstar = (tstarmax-tstarmin)/(Ntstar-1);
tstar = tstarmin + Dtstar*[0:Ntstar-1]';

Nt0 = 41;
t0min = 0;
t0max = 3;
Dt0 = (t0max-t0min)/(Nt0-1);
t0 = t0min + Dt0*[0:Nt0-1]';

E = zeros( Ntstar, Nt0 );

for i = [1:Ntstar]
    [ t, pulse0, A0, f, Qf2, cw2 ] = azimi( N, Dt, tstar(i), 1, 1, f0 );
    for j = [1:Nt0]
         A = delay(A0,Dt,t0(j)-N2*Dt);
         A = real(A);
         A = chebyshevfilt(A, Dt, 0.001, 1.0);
         f = conv( A, pulse1 );
         f = f(1:N);
         e = pulse2 - f;
         E(i,j) = e'*e;
end
end

figure(2);
clf;
hold on;
set(gca,'LineWidth',3);
axis( [t0min, t0max, tstarmin, tstarmax] );

[E1, k] = min(E);
[Emin, ic] = min(E1);
ir = k(ic);
```

```
imagesc([t0min,t0max],[tstarmin,tstarmax],E);

t0best = t0min+Dt0*(ic-1);
tstarbest = tstarmin+Dtstar*(ir-1);
plot( t0best, tstarbest, 'wo', 'LineWidth', 4 );
% plot( t02-t01, tstar2-tstar1, 'go', 'LineWidth', 2 );
xlabel('t0');
ylabel('tstar');

[ t0est, tstarest, E0, covm, status ] = Esurface( t0, tstar, E, sigmad^2 );
figure(2);
plot( t0est, tstarest, 'ro', 'LineWidth', 2 );

sigma_t0 = sqrt(covm(1,1));
sigma_tstar = sqrt(covm(2,2));
fprintf('Delta t0    %f +/- %f (95%%)\n', t0est, 2*sigma_t0 );
fprintf('Delta tstar %f +/- %f (95%%)\n', tstarest, 2*sigma_tstar );
plot( [t0est-2*sigma_t0, t0est+2*sigma_t0]', [tstarest, tstarest], 'r-',
'LineWidth', 2);
plot( [t0est, t0est]', [tstarest-2*sigma_tstar, tstarest+2*sigma_tstar]', 'r-
', 'LineWidth', 2);

[ t, pulse0, A0, f, Qf2, cw2 ] = azimi( N, Dt, tstarest, 1, 1, f0 );
A = delay(A0,Dt,t0est-N2*Dt);
A = real(A);
A = chebyshevfilt(A, Dt, 0.001, 1.0);
f = conv( A, pulse1 );
f = f(1:N);
figure(1);
plot( t, f, 'g-', 'LineWidth', 2 );


function [ t, pulse0, pulse, f, Qw, cw ] = azimi( N, Dt, x, c0, Q, f0 )

% input parameters:
% f0 corner frequency of Azimi Q model, in hz (e.g. 50)
% c0 base velocity in km/s (e.g. 4.5);
% x  propagation distance in km (e.g. 100)
% Q  low frequency quality factor (e.g. 10)
% N  number of samples in pulse (e.g. 1024);
% Dt sampling interbal (e.g. 0.1)

% returned values
% t time array
% pulse0 input pulse, a unit spike at time N/2
% pulse attentated pulse
% f frequencies in Hz
% Qw frequency dependent quality factors
% cw frequency dependent phase velocities


% time series
t = Dt*[0:N-1]';
pulse0 = zeros(N,1);
pulse0(N/2)=1;
```

```
% standard fft setup
fny = 1/(2*Dt);
N2 = N/2+1;
df = fny / (N/2);
f = df*[0:N2-1]';
w = 2*pi*f;
w0 = 2*pi*f0;

% attenuation factor
% exp( -a(w) x ) = exp( - wx / 2Qc )
%
% propagation law with velocity c=w/k and slowness s=1/c=k/w
% exp{ i(kx - wt) } = exp{ iw(sx - t) }
% propagation law
% exp( iwsx )

% Azimi's second law en.wikipedia.org/wiki/Azimi_Q_models
%
% a(w) = a2 |w| / [ 1 + a3 |w| ]
% note that for w<<w0 a(w) =
%
% s(w) = s0 + 2 a2 ln( a3 w ) / [ pi (1 - a3^2 w^2 ) ]

% now set a3 = 1/w0 where w0 is a reference frequency
% and set a2 = 1 / (2Qc0) where c0 is a reference velocity
% so that
% a(w) = (1/2Qc0) |w| / [ 1 +  |w/w0| ]
% so for w/w0 << 1
% a(w) = w/(2Qc0)  and Q(w) = w/(2 a c0)

a2 = 1 / (2*Q*c0);
a3 = 1 / w0;
a = a2*w ./ ( 1 + a3.*w );
Qw = w ./ (2.*a.*c0);
Qw(1) = Q;
ds = -2*a2*log(a3*w) ./ (pi*(1-(a3^2).*(w.^2 )));
ds(1)=0;

cw = 1./( (1/c0) + ds );

dt = fft(pulse0);
dp = dt(1:N2);
dp = dp .* exp(-a*x) .* exp(-complex(0,1)*w.*ds.*x);
dtnew = [dp(1:N2);conj(dp(N2-1:-1:2))]; % fold out negative frequencies
pulse = ifft(dtnew);

end


% chebyshevfilt
% chebyshev IIR bandpass filter
% din - input array of data
% Dt - sampling interval
% flow - low pass frequency, Hz
```

```
% fhigh - high pass frequency, Hz
% dout - output array of data
% u - the numerator filter
% v - the denominator filter
% these filters can be used again using dout=filter(u,v,din);

function [dout, u, v] = chebyshevfilt(din, Dt, flow, fhigh)

% sampling rate
rate=1/Dt;

% ripple parameter, set to ten percent
ripple=0.1;

% normalise frequency
fl=2.0*flow/rate;
fh=2.0*fhigh/rate;

% center frequency
cf = 4 * tan( (fl*pi/2) ) * tan( (fh*pi/2) );

% bandwidth
bw = 2 * ( tan( (fh*pi/2) ) - tan( (fl*pi/2) ) );

% ripple parameter factor
rpf = (sqrt((1.0+1.0/(ripple*ripple))) + 1.0/ripple) ^ (0.5);
a = 0.5*(rpf-1.0/rpf);
b = 0.5*(rpf+1.0/rpf);

u=zeros(5,1);
v=zeros(5,1);
theta = 3*pi/4;
sr = a * cos(theta);
si = b * sin(theta);
es = sqrt(sr*sr+si*si);
tmp= 16.0 - 16.0*bw*sr + 8.0*cf + 4.0*es*es*bw*bw - 4.0*bw*cf*sr + cf*cf;
v(1) = 1.0;
v(2) = 4.0*(-16.0 + 8.0*bw*sr - 2.0*bw*cf*sr + cf*cf)/tmp;
v(3) = (96.0 - 16.0*cf - 8.0*es*es*bw*bw + 6.0*cf*cf)/tmp;
v(4) = (-64.0 - 32.0*bw*sr + 8.0*bw*cf*sr + 4.0*cf*cf)/tmp;
v(5) = (16.0 + 16.0*bw*sr + 8.0*cf + 4.0*es*es*bw*bw + 4.0*bw*cf*sr +
cf*cf)/tmp;
tmp = 4.0*es*es*bw*bw/tmp;
u(1) = tmp;
u(2) = 0.0;
u(3) = -2.0*tmp;
u(4) = 0.0;
u(5) = tmp;

[dout]=filter(u,v,din);

return
```

```
function [ u ] = delay( u, Dt, t0 )

N = length(u);
fmax=1/(2.0*Dt);
df=fmax/(N/2);
Nf=N/2+1;
dw=2*pi*df;
w=dw*[0:N/2,-N/2+1:-1]';
u=ifft(exp(complex(0,-1)*w*t0).*fft(u));

end


function [ x0, y0, E0, covxy, status ] = Esurface( xc, yr, E, sigmad2 )
% Estd: analysis of 2D error surface
% input:
%     E: matrix of errors xc increases with columns, yr with rows
%     xc, yr: corresponding vectors of (xc, yr) values; must be evenly-spaced
%     sigmad2: variance of data that went into E=e'*e with e-dobs-dpre
% output:
%     x0, y0: location of minimum in E
%     E0: value of E at minimum
%     covxy: covariance matrix
%     status: 1 on success, 0 on fail
% method: quadratic approximation

% default return values
x0=0;
y0=0;
E0=0;
covxy=zeros(2,2);
status=0;
Nx = length(xc);
Ny = length(yr);
Dx = xc(2)-xc(1);
Dy = yr(2)-yr(1);

[E1, k] = min(E);
[Emin, ic] = min(E1);
ir = k(ic);

if( (ir<2) || (ir>(Ny-1)) || (ic<2) || (ic>(Nx-1)) )
    % minimum not in interior
    return;
end

% quadratic model with 9 data
% E = m1 + m2*x + m3*y + m4*x*x/2 + m5*x*y + m6*y*y/2
% dE/dx = m2 + m4*x + m5*y
% dE/dy = m3 + m5*x + m6*y
% at minimum
% 0 = d1 + D2 [x,y]'  so [x,y] = -inv(D2)*d1;
d = zeros(9,1);
x = Dx*[-1, 0, 1, -1, 0, 1, -1, 0, 1]';
y = Dy*[-1, -1, -1, 0, 0, 0, 1, 1, 1]';
```

```
d = [ E(ir-1,ic-1), E(ir-1,ic), E(ir-1,ic+1), E(ir,ic-1), E(ir,ic),
E(ir,ic+1), E(ir+1,ic-1), E(ir+1,ic), E(ir+1,ic+1)]';
G = [ones(9,1), x, y, 0.5*x.*x, x.*y, 0.5*y.*y];
m = (G'*G)\(G'*d);
E0 = m(1);
d1 = [m(2), m(3)]';
D2 = [m(4), m(5); m(5), m(6) ];
invD2 = inv(D2);
v = -invD2*d1;
x0 = xc(ic)+v(1);
y0 = yr(ir)+v(2);

covxy = (sigmad2)*2*invD2;

status=1;
end
```