2D Traveltime Tomography With Anisotropy Using Analytic Fourier Kernels
Bill Menke, November 27, 2013

Fourier representation of slowness. I am using here a 2D Fourier series to represent slowness s(x,y) and all other relevant two-dimensional fields:

$$s(x,y) = \sum_i \sum_j$$

$$\left\{A_{ij}\cos(k_x x)\cos\left(k_y y\right) + B_{ij}\cos(k_x x)\sin\left(k_y y\right) + C\sin(k_x x)\cos\left(k_y y\right) + D_{ij}\sin(k_x x)\sin(k_y y)\right\}$$

Here k_x and k_y are wavenumbers. The motivation for using a Fourier basis is that smoothness constraints easily can be implemented by preferentially damping the higher wavenumber coefficients. I am using the sine and cosine basis, as contrasted to the complex exponential basis, because the latter requires complicated and tedious constraints on the symmetry of the complex coefficients.

Analytic computation of the Data Kernel. The data kernel for tomography just represents line integrals, say I, of the Fourier basis over rays that are presumed to be straight lines. For example:

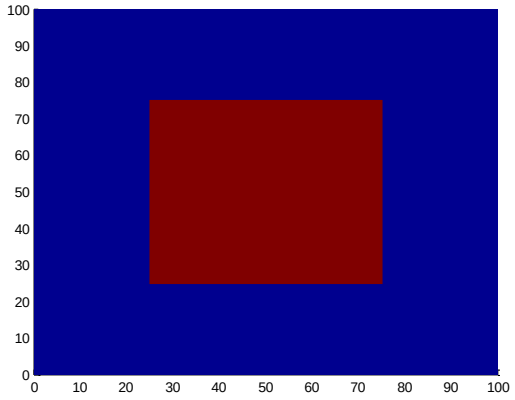
$$I = \int_{ray} \cos(k_x x)\cos\left(k_y y\right)\ ds$$

Here position [x(s),y(s)] varies with arclength s along the ray. Since x(s) and y(s) are linear functions of arclength, the line integral becomes an ordinary integral of two trigonometric functions, For example:

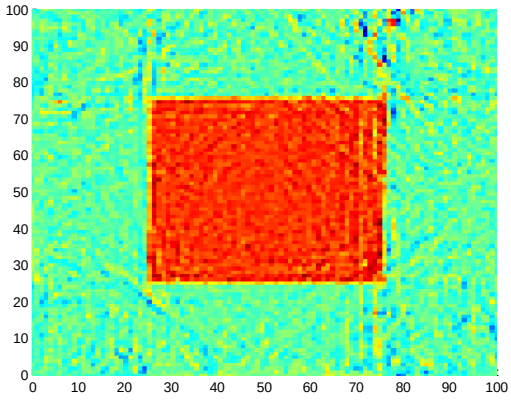
$$I = R\int_0^{s0} \cos(as)\cos(bs)\ ds + \ similar\ terms$$

Here a, b, R and s_0 are known functions of the ray endpoints. These integrals are known analytically (see, for, instance the Wikipedia article on integrals of trigonometric functions. However, these "textbook" formulas are singular in important special cases (such as when a=b), requiring the careful use of limits to derive formulas suitable for numerical evaluation.

Isotropic Tomography. Here's an example of isotropic tomography. The slowness model is defined on a 100x100 km rectangular grid. I created a dataset with 5000 random linear rays, each 2 km or greater length. Travel times are computed by numerical integration of the true slowness model (left) along the ray paths (rayforward.m). The model is represented as a 2D Fourier series with 40x40 wavenumbers. The data kernel **G** is computed by analytical integration of the Fourier basis along the linear rays. The inversion (rayinvert.m) uses damped least squares (epsilon=10^{-4}). The estimated model (right) predicts 99.97 percent of travel times.

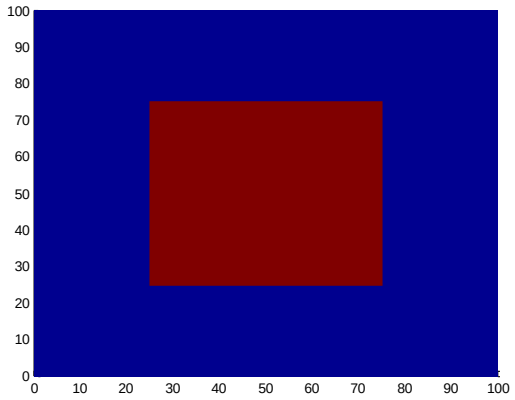


True Model

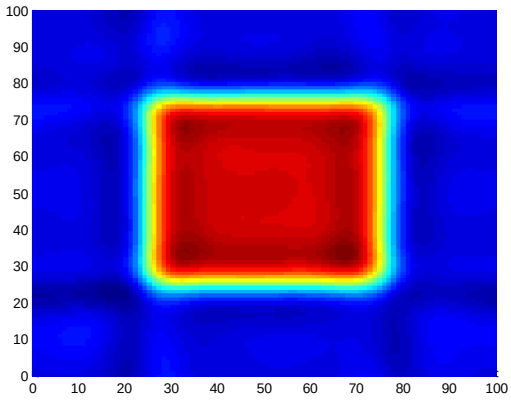


Estimated Model

Suppression of higher wavenumbers. I now add damping that scales with radial wavenumber, $k_r=(k_x^2+k_y^2)^{1/2}$. I use the prior constraint equation **Hm**=**h**, where $H=w(kr/k_0)^2$ and where $w=10^5$ and k_0 is the radial Nyquist frequency. The inversion (rayinvert2.m) uses generalized least squares. The estimated model (right) predicts 94.85 percent of travel times.



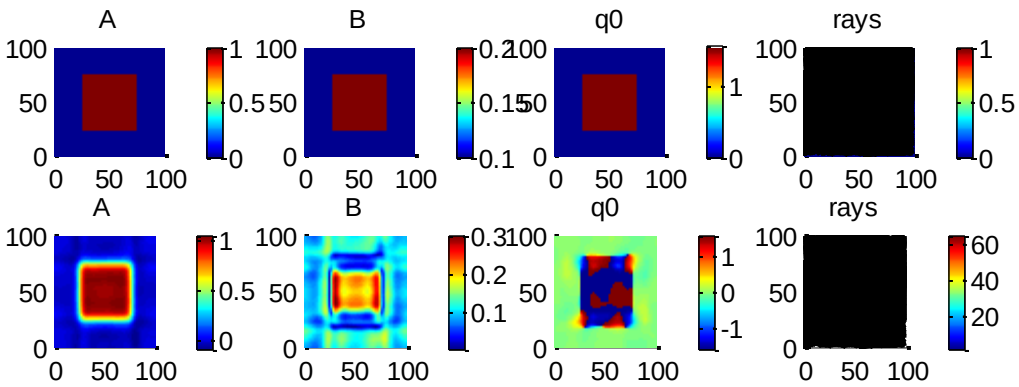
True Model



Estimated Model

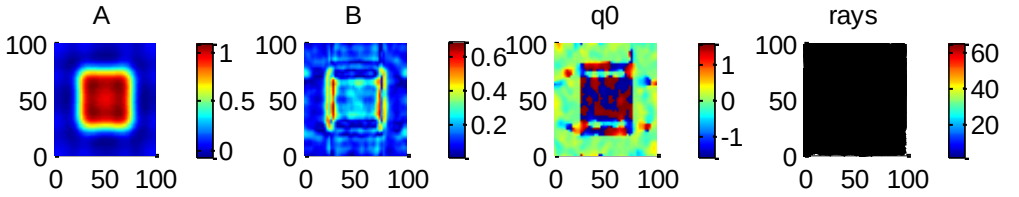
Anisotropic Tomography. The model is presumed to possess azimuthal anisotropy with the form slowness = A + B cos[2(q-q_0)], where q is the ray direction and q_0 is the anisotropic slow direction. A, B and q_0 are presumed to be functions of position. The model is parameterized with three Fourier series, for A(x,y), $b_c(x,y)=B\cos(2q_0)$ and $b_s(x,y)=B\sin(2q_0)$. The kernel functions are substantially the same as in the isotropic case, except the rows associated with b_c and b_s are multiplied by cos(2q) and sin(2q), respectively. The same radial damping is applied as in the isotropic case investigated above, except that provisions are made to damp A differently than b_c and b_s. The test run shown below uses the same damping, $w_I=w_A=10^{-5}$, as in the isotropic case, and predicts 95.1% of travel times. The isotropic part of

the solution achieves only 86.4%, so the inclusion of anisotropy leads to a significant improvement of the fit.

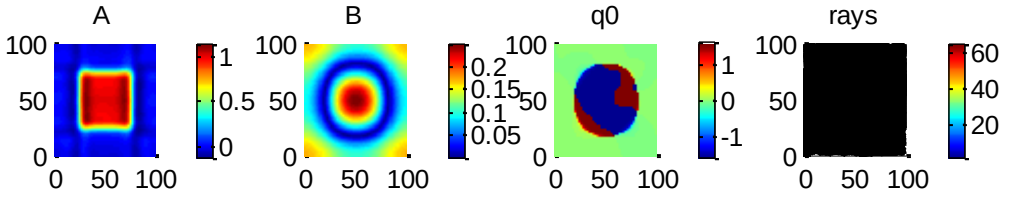


True model (top row) and estimated model (bottom row)

I have played around with the damping a bit. Here's an inversion with the anisotropic damping parameter reduced to $w_A=10^4$. It predicts 96.8% of travel times. This is only a slight improvement over the previous case at comes with the price of significantly increasing the heterogeneity of the anisotropy.



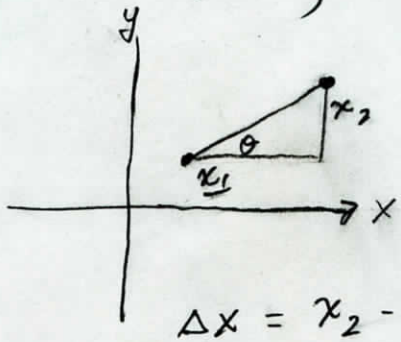
Another inversion I've tried has the anisotropic damping parameter *increased* to $w_A=10^6$. It predicts 94.4% of travel times. Note the strong radial symmetry of the estimated B and q0. They arise because the now-strong damping severely suppresses the higher wavenumber components..



w = slowness
k = wavenumber

$$\delta w(x,y) = \sum_j \sum_i \{ A_{ij} \cos(k_{xi} x) \cos(k_{yj} y) + B_{ij} \sin(k_{xi} x) \cos(k_{yj} y) + C_{ij} \cos(k_{xi} x) \sin(k_{yj} y) + D_{ij} \sin(k_{xi} x) \sin(k_{yj} y) \}$$

$$\delta T = \int_{line} \delta w \, ds$$



$$\Delta x = x_2 - x_1$$
$$\Delta y = y_2 - y_1$$
$$\theta = \tan^{-1}(\Delta y / \Delta x)$$
$$x = x_1 + s\cos\theta = a + bs$$
$$y = y_1 + s\sin\theta = c + ds$$
$$\underline{x} = x_1 \Rightarrow s = s_1 = 0$$
$$\underline{x} = x_2 \Rightarrow s = s_2 = [(\Delta x)^2 + (\Delta y)^2]^{1/2}$$

Term 1

$$T_1 = A_{ij} \cos(k_{xi} x) \cos(k_{yj} y)$$
$$= A_{ij} \cos(k_{xi} a + k_{xi} b s) \cos(k_{yj} c + k_{yj} d s)$$
$$= A_{ij} [\cos(k_{xi} a) \cos(k_{xi} b s) - \sin(k_{xi} a) \sin(k_{xi} b s)]$$
$$[\cos(k_{yj} c) \cos(k_{yj} d s) - \sin(k_{yj} c) \sin(k_{yj} d s)]$$
$$= A_{ij} [\cos(k_{xi} a) \cos(k_{yj} c) \cos(k_{xi} b s) \cos(k_{yj} d s)$$
$$- \cos(k_{xi} a) \sin(k_{yj} c) \cos(k_{xi} b s) \sin(k_{yj} d s)$$
$$- \sin(k_{xi} a) \cos(k_{yj} c) \sin(k_{xi} b s) \cos(k_{yj} d s)$$
$$+ \sin(k_{xi} a) \sin(k_{yj} c) \sin(k_{xi} b s) \sin(k_{yj} d s)$$

$\int_0^{s_2} ds$: I_{ss} : $\int \sin b_1 x \sin b_2 x \, dx = \frac{\sin((b_2-b_1)x)}{2(b_2-b_1)} - \frac{\sin((b_2+b_1)x)}{2(b_2+b_1)}$

From Wikipedia

I_{cc} : $\int \cos a_1 x \cos a_2 x \, dx = \frac{\sin((a_2-a_1)x)}{2(a_2-a_1)} + \frac{\sin((a_2+a_1)x)}{2(a_2+a_1)}$

I_{sc} : $\int \sin a_1 x \cos a_2 x \, dx$
(note rewrote 1st term) $= \frac{\cos((a_2-a_1)x)}{2(a_2-a_1)} - \frac{\cos((a_2+a_1)x)}{2(a_2+a_1)}$

I_{cs} : $\int \cos a_1 x \sin a_1 x \, ds$
(transp. of 3rd integral) $= -\frac{\cos((a_2-a_1)x)}{2(a_2-a_1)} - \frac{\cos((a_2+a_1)x)}{2(a_2+a_1)}$

$$\int_0^{s_2} T_1 \, ds = A_{ij} \times$$

$$\left\{ \begin{array}{l} \cos(kx_i a)\cos(ky_j c) \times \left[I_{cc}(kx_i b, ky_j d, s_2) - I_{cc}(\cdot\cdot 0) \right] \\ -\cos(kx_i a)\sin(ky_j c) \times \left[I_{cs}(kx_i b, ky_j d, s_2) - I_{cs}(\cdot\cdot 0) \right] \\ -\sin(kx_i a)\cos(ky_j c) \times \left[I_{sc}(kx_i b, ky_j d, s_2) - I_{sc}(\cdot\cdot 0) \right] \\ +\sin(kx_i a)\sin(ky_j c) \times \left[I_{ss}(kx_i b, ky_j d, s_2) - I_{ss}(\cdot\cdot 0) \right] \end{array} \right.$$

$I_{ss}(b_1, b_2, 0)$: 0

$I_{cc}(a_1, a_2, 0)$: 0

$I_{sc}(a_1, a_2, 0)$: $\frac{a_1}{a_2^2 - a_1^2}$

$I_{cs}(a_1, a_2, 0)$: $\frac{-a_2}{a_2^2 - a_1^2}$

$I_{sc}(a_1, a_2, 0)$: $\frac{1}{2(a_2-a_1)} - \frac{1}{2(a_2+a_1)}$

$= \frac{1}{2}\left(\frac{a_2+a_1}{(a_2^2-a_1^2)} - \frac{a_2-a_1}{(a_2^2-a_1^2)} \right)$

$= \frac{a_1}{a_2^2 - a_1}$

$I_{cs}(a_1, a_2, 0)$: $-\frac{1}{2(a_2-a_1)} - \frac{1}{2(a_2+a_1)}$

$= \frac{1}{2}\left(\frac{-a_2-a_1}{a_2^2-a_1^2} - \frac{a_2-a_1}{a_2^2-a_1^2} \right)$

$= \frac{-a_2}{a_2^2 - a_1^2}$

$$T_4 = D_{ij} \sin(k_{xi} x) \sin(k_{yj} y)$$

$$= D_{ij} \left[\sin(k_{xi} a + k_{xi} bs) \sin(k_{yj} c + k_{yj} ds) \right]$$

$$= D_{ij} \left[\sin(k_{xi} a)\cos(k_{xi} bs) + \cos(k_{xi} a)\sin(k_{xi} bs) \right]$$
$$\left[\sin(k_{yj} c)\cos(k_{yj} ds) + \cos(k_{yj} c)\sin(k_{yj} ds) \right]$$

$$= D_{ij} \Big[\sin(k_{xi} a)\sin(k_{yj} c) \boxed{\cos(k_{xi} bs)\cos(k_{yj} ds)} \; I_{cc}$$
$$+ \sin(k_{xi} a)\cos(k_{yj} c) \boxed{\cos(k_{xi} bs)\sin(k_{yj} ds)} \; I_{cs}$$
$$+ \cos(k_{xi} a)\sin(k_{yj} c) \boxed{\sin(k_{xi} bs)\cos(k_{yj} ds)} \; I_{sc}$$
$$+ \cos(k_{xi} a)\cos(k_{yj} c) \boxed{\sin(k_{xi} bs)\sin(k_{yj} ds)} \Big] \; I_{ss}$$

$$T_2 = B_{ij} \sin(k_{xi} x)\cos(k_{yj} y)$$

$$= B_{ij} \sin(k_{xi} a + k_{xi} bs)\cos(k_{yj} c + k_{yj} ds)$$

$$= B_{ij} \left[\sin(k_{xi} a)\cos(k_{xi} bs) + \cos(k_{xi} a)\sin(k_{xi} bs) \right]$$
$$\left[\cos(k_{yj} c)\cos(k_{yj} ds) - \sin(k_{yj} c)\sin(k_{yj} ds) \right]$$

$$= B_{ij} \left[S\,C\,I_{cc} - S\,S\,I_{cs} + C\,C\,I_{sc} - C\,S\,I_{ss} \right]$$

$$T_3 = C_{ij} \cos(k_{xi} x)\sin(k_{yj} y)$$

$$= C_{ij} \cos(k_{xi} a + k_{xi} bs)\sin(k_{yj} c + k_{yj} ds)$$

$$= C_{ij} \left[\cos(k_{xi} a)\cos(k_{xi} bs) - \sin(k_{xi} a)\sin(k_{xi} bs) \right]$$
$$\left[\sin(k_{yj} c)\cos(k_{yj} ds) + \cos(k_{yj} c)\sin(k_{yj} ds) \right]$$

$$= C_{ij} \left[C\,S\,I_{cc} + C\,C\,I_{cs} - S\,S\,I_{sc} - S\,C\,I_{ss} \right]$$

proper limit of I_{cs} when $b_1 \approx b_2$

From wikipedia

integration const to enforce $I_{cs} \to 0$ as $x \to 0$

$$I_{cs} = -\frac{\cos((b_2-b_1)x)}{2(b_2-b_1)} - \frac{\cos((b_2+b_1)x)}{2(b_2+b_1)} + \frac{b_2}{b_2^2-b_1^2}$$

Taylor series

$$I_{cs} = -\frac{\left[1-\frac{1}{2}(b_2-b_1)^2x^2\right]}{2(b_2-b_1)} - \frac{\left[\cos((b_2+b_1)x)-1\right]}{2(b_2+b_1)} - \frac{1}{2(b_2+b_1)} + \frac{b_2}{b_2^2-b_1^2}$$

$$= \underbrace{-\frac{1}{2(b_2-b_1)} - \frac{1}{2(b_2+b_1)} + \frac{2b_2}{2(b_2^2-b_1^2)}}_{} \underbrace{-\frac{\left[\cos((b_2+b_1)x)-1\right]}{2(b_2+b_1)} + \frac{1}{4}(b_2-b_1)x^2}_{\text{These terms ok}}$$

$$= -\frac{(b_2+b_1)}{2(b_2^2-b_1^2)} - \frac{(b_2-b_1)}{2(b_2^2-b_1^2)} + \frac{2b_2}{2(b_2^2-b_1^2)}$$

$$= \frac{-b_2-b_2+2b_2-b_1+b_1}{2(b_2^2-b_1^2)}$$

$$= \boxed{0 - \frac{\left[\cos((b_2+b_1)x)-1\right]}{2(b_2+b_1)} + \frac{1}{4}(b_2-b_1)x^2}$$

Limit for I_{sc}

$$I_{sc} = \frac{\cos((b_2-b_1)x)}{2(b_2-b_1)} - \frac{\cos((b_2+b_1)x)}{2(b_2+b_1)} - \frac{b_2}{b_2^2-b_1^2}$$

$$\boxed{= -\frac{\left[\cos((b_2+b_1)x)-1\right]}{2(b_2+b_1)} - \frac{1}{4}(b_2-b_1)x^2}$$

$$\to 1 - \frac{\frac{1}{2}(b_2-b_1)^2x^2}{2(b_2-b_1)}$$

Iss limit

$$I_{cc} = \frac{\sin(b_2-b_1)x}{2(b_2-b_1)} - \frac{\sin((b_2+b_1)x)}{2(b_2+b_1)}$$

(expand in taylor series) (term ok)

$$= \frac{(b_2-b_1)x}{2(b_2-b_1)} - \quad \| \quad \frac{(b_2-b_1)x}{2(b_2-b_1)} - \frac{(b_2-b_1)^2 x^3}{12}$$

$$= \left(\frac{x}{2}\right) - \frac{\sin((b_2+b_1)x)}{2(b_2+b_1)}$$

Icc limit

$$I_{cc} = \frac{\sin(b_2-b_1)x}{2(b_2-b_1)} + \frac{\sin((b_2+b_1)x)}{2(b_2+b_1)}$$

taylor series

$$= \frac{(b_2-b_1)x}{2(b_2-b_1)} + \frac{\sin(b_2+b_1)x}{2(b_2+b_1)}$$

$$= \left(\frac{x}{2}\right) + \frac{\sin(b_2+b_1)x}{2(b_2+b_1)} \qquad \frac{x}{2} - \frac{(b_2-b_1)^2 x^3}{12}$$

limits $b_1 \approx -b_2$

$$I_{ss} = \frac{\sin((b_2-b_1)x)}{2(b_2-b_1)} - \frac{\sin((b_2+b_1)x)}{2(b_2+b_1)}$$

$$= \frac{\sin((b_2-b_1)x)}{2(b_2-b_1)} - \left(\frac{x}{2} - \frac{(b_2+b_1)^3 x^3}{12(b_2+b_1)} \right)$$

$$= \quad " \quad - \frac{x}{2} + \frac{(b_2+b_1)^2 x^3}{12}$$

limit $b_1 \to 0$ and $b_2 \neq 0$

$$= -\frac{(b_2-b_1)^2 x^3}{12} + \frac{(b_2+b_1)^2 x^3}{12}$$

$$I_{cc} = \frac{\sin((b_2-b_1)x)}{2(b_2-b_1)} + \frac{\sin((b_2+b_1)x)}{2(b_2+b_1)}$$

$$\frac{\sin((b_2-b_1)x)}{2(b_1-b_2)} + \frac{x}{2} - \frac{(b_2+b_1)^2 x^3}{12}$$

Anisotropy. — azimuthal

θ = ray angle

θ_0 = slow axis

$$A + B\cos(2(\theta - \theta_0))$$

$$B\cos(2\theta - 2\theta_0) = B\cos(2\theta)\cos(2\theta_0) + B\sin(2\theta)\sin(2\theta_0)$$

$$\underbrace{[B\cos(2\theta_0)]}_{b_c(x,y)}\cos 2\theta + \underbrace{[B\sin(2\theta_0)]}_{b_s(x,y)}\sin 2\theta$$

$$b_c = B\cos 2\theta_0 \qquad b_s = B\sin 2\theta_0$$

$$(b_c^2 + b_s^2)^{1/2} = B \qquad \theta_0 = \tfrac{1}{2}\text{atan2}(b_s, b_c)$$

Model now has 3 Fourier series $A(x,y)$ $b_c(x,y)$ $b_s(x,y)$

Kernel for b_c just has rows multiplied by $\cos 2\theta$

b_s just has rows multiplied by $\sin 2\theta$

(θ = ray angle)

```
% Integral cos(b1 x) cos(b2 x) dx
% analytic formula from Wikipedia
% checked by numerical integration of a few test cases

function [y] = Icc( b1, b2, x, uselimit )

eps = 0.01;
tm = (abs(b2-b1)<eps);
tp = (abs(b1+b2)<eps);

if( (uselimit==1) && (tm||tp) )
    % Limit by expanding sin in Taylor Series and keeping 2 terms
    if( tm && (~tp) )
        y = (x/2) -(((b2-b1)^2)*(x^3))/12 +(sin((b2+b1)*x)/(2*(b2+b1)));
    elseif ( (~tm) && tp )
        y = (sin((b2-b1)*x)/(2*(b2-b1))) +(x/2) -(((b2+b1)^2)*(x^3)/12);
    else
        y = (x/2) -(((b2-b1)^2)*(x^3))/12 +(x/2) -(((b2+b1)^2)*(x^3)/12);
    end
else
% Note the integration constant is chosen to enforce Ics(b1,b2,x=0)=0
%       <----------------- Wikipedia ------------------------------>   <-
zero integration constant ->
%
    y = (sin((b2-b1)*x)/(2*(b2-b1))) + (sin((b2+b1)*x)/(2*(b2+b1)));
end
end


-----------------------------------------------------------


% Integral cos(b1 x) sin(b2 x) dx
% analytic formula from Wikipedia
% checked by numerical integration of a few test cases

function [y] = Ics( b1, b2, x, uselimit )
eps = 0.01;
tm = (abs(b2-b1)<eps);
tp = (abs(b1+b2)<eps);

if( (uselimit==1) && (tm||tp) )
    if( tm && (~tp) )
    % Limit by expanding one cos in Taylor Series and keeping the first
    % term and by writing the other cosine as (cos()-1) + 1
    y = ((b2-b1)*x*x/4) -((cos((b2+b1)*x)-1)/(2*(b2+b1)));
    elseif( (~tm) && tp )
    y = (-(cos((b2-b1)*x)-1)/(2*(b2-b1))) + (b2+b1)*x*x/4;
    else
    y = (b2-b1)*x*x/4 + (b2+b1)*x*x/4;
    end
else
% Note the integration constant is chosen to enforce Ics(b1,b2,x=0)=0
```

```
%       <----------------- Wikipedia ----------------------------->   <-
integration constant ->
%
    y = (-cos((b2-b1)*x)/(2*(b2-b1))) -(cos((b2+b1)*x)/(2*(b2+b1))) +
(b2/(b2*b2-b1*b1));
end
end


-------------------------------------------------------


% Integral sin(b1 x) cos(b2 x) dx
% analytic formula from Wikipedia
% checked by numerical integration of a few test cases

function [y] = Isc( b1, b2, x, uselimit )

eps = 0.01;
tm = (abs(b2-b1)<eps);
tp = (abs(b1+b2)<eps);

if( (uselimit==1) && (tm||tp) )
    if( tm && (~tp) )
    % Limit by expanding one cosine in a Taylor series and keeping
    % two terms and by writing other cosine as (cos()-1) + 1
    y = -(b2-b1)*x*x/4 -((cos((b2+b1)*x)-1)/(2*(b2+b1)));
    elseif( (~tm) && tp )
    y = ((cos((b2-b1)*x)-1)/(2*(b2-b1))) +(b2+b1)*x*x/4;
    else
    y = -(b2-b1)*x*x/4 +(b2+b1)*x*x/4;
    end
else
    % Note the integration constant is chosen to enforce Ics(b1,b2,x=0)=0
%       <----------------- Wikipedia ----------------------------->   <-
integration constant ->
%
    y = (cos((b2-b1)*x)/(2*(b2-b1))) - (cos((b2+b1)*x)/(2*(b2+b1))) -
(b1/(b2*b2-b1*b1));
end
end


-------------------------------------------------------


% Integral sin(b1 x) sin(b2 x) dx
% analytic formula from Wikipedia, supplemented by limits derived by me
% checked by numerical integration of a few test cases

function [y] = Iss( b1, b2, x, uselimit )

eps = 0.01;
tm = (abs(b2-b1)<eps);
```

```
tp = (abs(b1+b2)<eps);

if( (uselimit==1) && (tm||tp) )
    if( tm && (~tp) )
       % Obtain limit by expanding sin((b2-b1)*x)
       % in a Taylor series and keeping the first two terms
       y = (x/2) -(((b2-b1)^2)*(x^3)/12) -(sin((b2+b1)*x)/(2*(b2+b1)));
    elseif( (~tm) && (tp ) )
       % Obtain limit by expanding sin((b2-b1)*x)
       % in a Taylor series and keeping the first term
       y = (sin((b2-b1)*x)/(2*(b2-b1))) - (x/2) + ((b2+b1)^2)*(x^3)/12;
    else
       y = -(((b2-b1)^2)*(x^3)/12) + ((b2+b1)^2)*(x^3)/12;
    end
else
        % Note the integration constant is chosen to enforce
Ics(b1,b2,x=0)=0
%       <----------------- Wikipedia ----------------------------->   <-
zero integration constant ->
%
    y = (sin((b2-b1)*x)/(2*(b2-b1))) - (sin((b2+b1)*x)/(2*(b2+b1)));
end
end


-----------------------------------------------------------


clear all

% x axis
xmin = 0;
xmax = 100;
Nx = 1000;
Dx = (xmax-xmin)/Nx;
x = Dx*[0:Nx-1]';

% y axis
ymin = 0;
ymax = 100;
Ny = 1000;
Dy = (ymax-ymin)/Ny;
y = Dy*[0:Ny-1]';

% anisotropic slowness, s(y,x)=w+wB(cos(2(q-q0)) (note y is first index)
% where w is issotropic baclground, wB is magnitude of anisotropy,
% q is ray direction and q0 is anisotropic fast direction
w = zeros(Ny,Nx);
wB = 0.1*ones(Ny,Nx);
wq0 = zeros(Ny,Nx);

w(250:750,250:750)=1;
wB(250:750,250:750)=0.2;
wq0(250:750,250:750)=pi/2;
```

```
% plot model
figure(1);
clf;
subplot(1,4,1)
hold on;
axis xy;
axis( [xmin, xmax, ymin, ymax] );
imagesc( [ymin, ymax], [xmin, xmax], w);
colorbar;
title('A');
subplot(1,4,2)
hold on;
axis xy;
axis( [xmin, xmax, ymin, ymax] );
imagesc( [ymin, ymax], [xmin, xmax], wB);
colorbar;
title('B');
subplot(1,4,3)
hold on;
axis xy;
axis( [xmin, xmax, ymin, ymax] );
imagesc( [ymin, ymax], [xmin, xmax], wq0);
colorbar;
title('q0');

% data are traveltimes along rays thru model
Nrays = 5000;
D = zeros(Nrays,5); % data: x1, x2, y1, y2, T

subplot(1,4,4)
hold on;
axis xy;
axis( [xmin, xmax, ymin, ymax] );
imagesc( [ymin, ymax], [xmin, xmax], w);
colorbar;
title('rays');

% compute traveltime for eacg ray
for iray = [1:Nrays]

    % randomly pick ray endpoints
    ix1 = random('unid',Nx,1,1);
    ix2 = random('unid',Nx,1,1);
    iy1 = random('unid',Ny,1,1);
    iy2 = random('unid',Ny,1,1);
    x1 = x(ix1);  y1 = y(iy1);
    x2 = y(ix2);  y2 = y(iy2);
    q = atan2( (y2-y1), (x2-x1) );
    plot( [x1, x2], [y1, y2], 'k-', 'LineWidth', 2 );

    % sample rays
    s1 = 0;
    s2 = sqrt( (x2-x1)^2 + (y2-y1)^2 );
```

```
    if( s2<(Nx/50) ) % ignore really short rays
        continue;
    end
    Ns = 1000;
    Ds = (s2-s1)/Ns;
    s = Ds*[0:Ns-1]';

    % bookkeeping for sampling slowness along ray
    ixs = 1+floor( Nx*(x1 + s*cos(q))/(xmax-xmin) );
    jys = 1+floor( Ny*(y1 + s*sin(q))/(ymax-ymin) );
    xs = x(ixs);
    ys = y(jys);

    % numerical integration along ray
    I1 = 0;
    for i=[1:Ns]
        wxy = w(jys(i),ixs(i)) + wB(jys(i),ixs(i))*cos(2*(q-
wq0(jys(i),ixs(i))));
        I1 = I1 + Ds*wxy;
    end

    % save data
    D(iray,1)=x1; D(iray,2)=x2; D(iray,3)=y1; D(iray,4)=y2; D(iray,5)=I1;

end

% write data to a file
dlmwrite('anidata.txt',D,'\t');

------------------------------------------------------------

clear all
global F;

% slowness is 2D Fourier series
% I use sines and cosine representation rather than complex exponentials
% so that I don't have to deal with the real/imaginary part symmetry
% constraints

% w(x,y) = Si Sj (Aij cos(kxi x) cos(kyj y) + Bij sin(kxi x) cos(kyj y)
%              +  Cij cos(kxi x) sin(kyj y) + Dij sin(kxi x) sin(kyj y)
% where kxi and kyj are wavenumbers (k = 2PI/wavelength)

% the kernel, containing the ray integrals, is computed analytically
% using horrible trigonometric functions that I spent rather too long
% deriving and verifying

% Load data
% adjustable parameter: data file
datafile = 'anidata.txt';
D = load(datafile); % data: ray endpoints x1, x2, y1, y2, Traveltime
[N, ncols] = size(D);
dobs = D(:,5);
```

```
% adjustable parameters: boundaries of study region
xmin = 0;
xmax = 100;
ymin = 0;
ymax = 100;

% adjustable parameters: number of grid points in x and y
% this choice controls both the Nyquist Frequency and the
% sampling interval of the final gridded slowness
Nx = 100;
Ny = 100;

% plot raypaths
figure(2);
clf;
subplot(1,4,4)
hold on;
axis xy;
axis( [xmin, xmax, ymin, ymax] );
title('rays');
for iray = [1:N] % loop over rays
    x1 = D(iray,1);  y1 = D(iray,3);
    x2 = D(iray,2);  y2 = D(iray,4);
    plot( [x1, x2], [y1, y2], 'k-', 'LineWidth', 2 );
end
colorbar;

% Representation of azimuthal anisotropy:
% slowness = A + B cos[2(q-q0)] with q=ray_angle, q0=fast_axis_angle
% and w=A, wc=Bcos(q0), ws=B(sin2q0) define three unknown 2D fields
% such that A=w, B = sqrt( wc^2 + ws^2) and q0 = atan2(bx,bc)
% Note y index is first in all 2D fields, e.g. w(x,y)
west = zeros(Ny,Nx);
wcest = zeros(Ny,Nx);
wsest = zeros(Ny,Nx);
Best = zeros(Ny,Nx);
q0est = zeros(Ny,Nx);

% x wavenumbers
Dx = (xmax-xmin)/Nx;
x = Dx*[0:Nx-1]';
kxny = 2*pi/(2*Dx); % Nyquist wavenumber
Nkxmax = (Nx/2)+1;
Dkx = kxny/(Nkxmax-1);
kx = Dkx*[0:Nkxmax-1]';

% adjustable parameter: number of x wavenumbers used isotropic field
Nkx_iso = 40;

% adjustable parameter: number of x wavenumbers used anisotropic field
Nkx_ani = 40;

% y wavenumbers
Dy = (ymax-ymin)/Ny;
```

```
y = Dy*[0:Ny-1]';
kyny = 2*pi/(2*Dy); % Nyquist wavenumber
Nkymax = (Ny/2)+1;
Dky = kxny/(Nkymax-1);
ky = Dky*[0:Nkymax-1]';

% adjustable parameter: number of y wavenumbers used isotropic field
Nky_iso = 10;

% adjustable parameter: number of y wavenumbers used anisotropic field
Nky_ani = 10;

% data kernel definition
% data, dobs, observed traveltimes along rays
% N number of data, equals number of rays
% M is the total number of model parameters, m
% The model parameters contain three groups of Fourier coefficients
% for w, wc and ws. All zero columns associated with sines of zero
% bwavenumber are omitted. The backpointers are the only easy way
% to find a specific model parameter within m, as the orderin is
% complicated

M = 4*Nkx_iso*Nky_iso + 2*4*Nkx_ani*Nky_ani; % note wc, ws have same
number of wavenumbers
G = zeros(N, M);

% build data kernel
% backpointers
    ikxindex = zeros(M,1); % x wavenumber index of a given model
parameter
    jkyindex = zeros(M,1); % y wavenumber index of a given model
parameter
    kxindex = zeros(M,1);  % x wavenumber of a given model parameter
    kyindex = zeros(M,1);  % y wavenumber of a given model parameter
    cindex = zeros(M,1);   % coefficient type (Aij, Bij, Cij, Dij) = (1,
2, 3, 4)
    vindex = zeros(M,1);   % variable type (w, wc, ws) = (1, 2, 3)

for iray = [1:N] % loop over rays

    % ray info
    x1 = D(iray,1);  y1 = D(iray,3);
    x2 = D(iray,2);  y2 = D(iray,4);
    s1=0; s2=sqrt( (x2-x1)^2 + (y2-y1)^2 );
    q = atan2( (y2-y1), (x2-x1) );
    a=x1; b=cos(q); b2=cos(2*q);
    c=y1; d=sin(q); d2=sin(2*q);
    ind = 0; % model parameter counter (handles omitted zero columns)

    % fourier coefficients of w
    for ikx = [1:Nkx_iso] % loop over x wavenumbers
        kxi = kx(ikx);
        for jky = [1:Nky_iso] % loop over y wavenumbers
            kyj = ky(jky);
```

```
% A-type cos()cos() model parameter

ind = ind+1; % current model parameter and backpointers
ikxindex(ind)=ikx;
jkyindex(ind)=jky;
kxindex(ind)=kxi;
kyindex(ind)=kyj;
cindex(ind)=1; % A-type coefficient
vindex(ind)=1; % w-type series

% some definitions
kxia=kxi*a; ckxia=cos(kxia); skxia=sin(kxia);
kxib=kxi*b; ckxib=cos(kxib); skxib=sin(kxib);
kyjc=kyj*c; ckyjc=cos(kyjc); skyjc=sin(kyjc);
kyjd=kyj*d; ckyjd=cos(kyjd); skyjd=sin(kyjd);

% T1 (integral) cos(kxi*x) cos(kyj*y) ds
% checked against numerical calculations
T1 = 0;
T1 = T1 + ckxia*ckyjc*Icc(kxib,kyjd,s2,1);
T1 = T1 - ckxia*skyjc*Ics(kxib,kyjd,s2,1);
T1 = T1 - skxia*ckyjc*Isc(kxib,kyjd,s2,1);
T1 = T1 + skxia*skyjc*Iss(kxib,kyjd,s2,1);
G(iray,ind) = T1;

% B-type sin()cos() model parameter
if( ikx~=1  ) % ignore sin(kx1 x) cos(ky y) terms

ind = ind+1; % current model parameter and backpointers
ikxindex(ind)=ikx;
jkyindex(ind)=jky;
kxindex(ind)=kxi;
kyindex(ind)=kyj;
cindex(ind)=2; % B-type coefficient
vindex(ind)=1; % w-type series

% T2 (integral) sin(kxi*x) cos(kyj*y) ds
% checked against numerical calculations
T2 = 0;
T2 = T2 + skxia*ckyjc*Icc(kxib,kyjd,s2,1);
T2 = T2 - skxia*skyjc*Ics(kxib,kyjd,s2,1);
T2 = T2 + ckxia*ckyjc*Isc(kxib,kyjd,s2,1);
T2 = T2 - ckxia*skyjc*Iss(kxib,kyjd,s2,1);
G(iray,ind) = T2;

end

% C-type cos()sin() model parameter
if( jky~=1  ) % ignore cos(kx x) sin(ky1 y) terms
ind = ind+1; % current model parameter and backpointers
ikxindex(ind)=ikx;
jkyindex(ind)=jky;
kxindex(ind)=kxi;
```

```
            kyindex(ind)=kyj;
            cindex(ind)=3; % C-type coefficient
            vindex(ind)=1; % w-type series

            % T3 (integral) cos(kxi*x) sin(kyj*y) ds
            % checked against numerical calculations
            T3 = 0;
            T3 = T3 + ckxia*skyjc*Icc(kxib,kyjd,s2,1);
            T3 = T3 + ckxia*ckyjc*Ics(kxib,kyjd,s2,1);
            T3 = T3 - skxia*skyjc*Isc(kxib,kyjd,s2,1);
            T3 = T3 - skxia*ckyjc*Iss(kxib,kyjd,s2,1);
            G(iray,ind) = T3;
            end

            % D-type sin()sin() model parameter
            if( ~((ikx==1) || (jky==1)) ) % ignore sin(kx1 x) sin(ky1 y)
terms

            ind = ind+1; % current model parameter and backpointers
            ikxindex(ind)=ikx;
            jkyindex(ind)=jky;
            kxindex(ind)=kxi;
            kyindex(ind)=kyj;
            cindex(ind)=4; % D-type coefficient
            vindex(ind)=1; % w-type series

            % T4  (integral) sin(kxi*x) sin(kyj*y) ds
            % checked against numerical calculations
            T4 = 0;
            T4 = T4 + skxia*skyjc*Icc(kxib,kyjd,s2,1);
            T4 = T4 + skxia*ckyjc*Ics(kxib,kyjd,s2,1);
            T4 = T4 + ckxia*skyjc*Isc(kxib,kyjd,s2,1);
            T4 = T4 + ckxia*ckyjc*Iss(kxib,kyjd,s2,1);
            G(iray,ind) = T4;

            end

        end % iky
    end % ikx

    % fourier coefficients of wc and ws
    for ikx = [1:Nkx_ani] % loop over x wavenumbers
        kxi = kx(ikx);
        for jky = [1:Nky_ani] % loop over y wavenumbers
            kyj = ky(jky);

            % A-type cos()cos() model parameter

            % some definitions
            kxia=kxi*a; ckxia=cos(kxia); skxia=sin(kxia);
            kxib=kxi*b; ckxib=cos(kxib); skxib=sin(kxib);
            kyjc=kyj*c; ckyjc=cos(kyjc); skyjc=sin(kyjc);
            kyjd=kyj*d; ckyjd=cos(kyjd); skyjd=sin(kyjd);
```

```
% T1 (integral) cos(kxi*x) cos(kyj*y) ds
% checked against numerical calculations
T1 = 0;
T1 = T1 + ckxia*ckyjc*Icc(kxib,kyjd,s2,1);
T1 = T1 - ckxia*skyjc*Ics(kxib,kyjd,s2,1);
T1 = T1 - skxia*ckyjc*Isc(kxib,kyjd,s2,1);
T1 = T1 + skxia*skyjc*Iss(kxib,kyjd,s2,1);

ind = ind+1;
ikxindex(ind)=ikx;
jkyindex(ind)=jky;
kxindex(ind)=kxi;
kyindex(ind)=kyj;
cindex(ind)=1; % A-type coefficient
vindex(ind)=2; % wc-type series
G(iray,ind) = b2*T1;

ind = ind+1;
ikxindex(ind)=ikx;
jkyindex(ind)=jky;
kxindex(ind)=kxi;
kyindex(ind)=kyj;
cindex(ind)=1; % A-type coefficient
vindex(ind)=3; % ws-type series
G(iray,ind) = d2*T1;

% B-type sin()cos() model parameter
if( ikx~=1  ) % ignore sin(kx1 x) cos(ky y) terms

% T2 (integral) sin(kxi*x) cos(kyj*y) ds
% checked against numerical calculations
T2 = 0;
T2 = T2 + skxia*ckyjc*Icc(kxib,kyjd,s2,1);
T2 = T2 - skxia*skyjc*Ics(kxib,kyjd,s2,1);
T2 = T2 + ckxia*ckyjc*Isc(kxib,kyjd,s2,1);
T2 = T2 - ckxia*skyjc*Iss(kxib,kyjd,s2,1);

ind = ind+1; % current model parameter and backpointers
ikxindex(ind)=ikx;
jkyindex(ind)=jky;
kxindex(ind)=kxi;
kyindex(ind)=kyj;
cindex(ind)=2; % B-type coefficient
vindex(ind)=2; % wc-type series
G(iray,ind) = b2*T2;

ind = ind+1; % current model parameter and backpointers
ikxindex(ind)=ikx;
jkyindex(ind)=jky;
kxindex(ind)=kxi;
kyindex(ind)=kyj;
cindex(ind)=2; % B-type coefficient
vindex(ind)=3; % ws-type series
G(iray,ind) = d2*T2;
```

```
            end

            % C-type cos()sin() model parameter
            if( jky~=1  ) % ignore cos(kx x) sin(ky1 y) terms

            % T3 (integral) cos(kxi*x) sin(kyj*y) ds
            % checked against numerical calculations
            T3 = 0;
            T3 = T3 + ckxia*skyjc*Icc(kxib,kyjd,s2,1);
            T3 = T3 + ckxia*ckyjc*Ics(kxib,kyjd,s2,1);
            T3 = T3 - skxia*skyjc*Isc(kxib,kyjd,s2,1);
            T3 = T3 - skxia*ckyjc*Iss(kxib,kyjd,s2,1);

            ind = ind+1; % current model parameter and backpointers
            ikxindex(ind)=ikx;
            jkyindex(ind)=jky;
            kxindex(ind)=kxi;
            kyindex(ind)=kyj;
            cindex(ind)=3; % C-type coefficient
            vindex(ind)=2; % wc-type series
            G(iray,ind) = b2*T3;

            ind = ind+1; % current model parameter and backpointers
            ikxindex(ind)=ikx;
            jkyindex(ind)=jky;
            kxindex(ind)=kxi;
            kyindex(ind)=kyj;
            cindex(ind)=3; % C-type coefficient
            vindex(ind)=3; % ws-type series
            G(iray,ind) = d2*T3;

            end

            % D-type sin()sin() model parameter
            if( ~((ikx==1) || (jky==1)) ) % ignore sin(kx1 x) sin(ky1 y)
terms

            % T4  (integral) sin(kxi*x) sin(kyj*y) ds
            % checked against numerical calculations
            T4 = 0;
            T4 = T4 + skxia*skyjc*Icc(kxib,kyjd,s2,1);
            T4 = T4 + skxia*ckyjc*Ics(kxib,kyjd,s2,1);
            T4 = T4 + ckxia*skyjc*Isc(kxib,kyjd,s2,1);
            T4 = T4 + ckxia*ckyjc*Iss(kxib,kyjd,s2,1);

            ind = ind+1; % current model parameter and backpointers
            ikxindex(ind)=ikx;
            jkyindex(ind)=jky;
            kxindex(ind)=kxi;
            kyindex(ind)=kyj;
            cindex(ind)=4; % D-type coefficient
            vindex(ind)=2; % wc-type series
            G(iray,ind) = b2*T4;
```

```
            ind = ind+1; % current model parameter and backpointers
            ikxindex(ind)=ikx;
            jkyindex(ind)=jky;
            kxindex(ind)=kxi;
            kyindex(ind)=kyj;
            cindex(ind)=4; % D-type coefficient
            vindex(ind)=3; % ws-type series
            G(iray,ind) = d2*T4;

            end

        end % iky
    end % ikx

end % iray

G = G(:,1:ind); % truncate to remove extra model paremeters associated
with zero columns
M = ind;

% now add damping that drives the coefficients of the higher wavenumbers
% toward zero.  The equation Hm=h will have a diagonal H and h=0;  The
% larger the size of Hii, the more wavenumber i will be supressed.  The
% isotropic and anisotropic parts of the model can be damped differenly

H = zeros(M,M);
h = zeros(M,1);
kscale = sqrt(kxny*kxny + kyny*kyny); % wavenumber scaling based on
nyquist

% adjustable parameters: coefficients of weight fundctions
wt_iso = 1e5; % overall weight for the length damping
power_iso = 2;
wt_ani = 1e4; % overall weight for the length damping
power_ani = 0;

% create diagonal matrix H of weights
for ind=[1:M]
    ikx=ikxindex(ind); % now backpointers come in handy
    jky=jkyindex(ind);
    kxi=kxindex(ind);
    kyj=kyindex(ind);
    ic=cindex(ind);
    iv=vindex(ind);
    kr = sqrt( kxi*kxi + kyj*kyj );
    if( iv==1 ) % isotropic part
        H(ind,ind) = wt_iso*((kr/kscale)^power_iso);
    else % anisotropic part
        H(ind,ind) = wt_ani*((kr/kscale)^power_ani);
    end
end

% estimate model parameters via generalized least squares
```

```
% At some point, I suppose that I should code so as to create F
% directly, so as to save space, rather than to concatenate
F = [G; H];
f = [dobs; h];
mest = bicg( @weightedleastsquaresfcn, F'*f, 1e-5, 3*M );

% traveltime predition
dpre = G*mest;
rmstt = sqrt(dobs'*dobs/N);
rmserror = sqrt((dobs-dpre)'*(dobs-dpre)/N);
fprintf('RMS traveltime error (Gm ): %f percent improvement %f\n',
rmserror, 100-100*rmserror/rmstt );

% plot traveltime errors
figure(3);
clf;
subplot(1,2,1);
hold on;
plot( dobs, dpre, 'ko' );
xlabel('Tobs');
ylabel('Tpre');
title('from G*mest');

% Brute-force summation of fourier coefficients to compute west, wcest
and wsest.
% It would be much faster to rearrange the cos-sin representation into a
complex
% exponential representation and then use a fast fourier transform
for ix = [1:Nx]
    x0 = x(ix);
    for iy = [1:Ny]
        y0 = y(iy);
        for ind = [1:M]
            ikx=ikxindex(ind); % now backpointers come in handy
            jky=jkyindex(ind);
            kxi=kxindex(ind);
            kyj=kyindex(ind);
            ic=cindex(ind);
            iv=vindex(ind);
            kxix = kxi*x0; kyjy = kyj*y0;
            ckxix=cos(kxix); skxix=sin(kxix);
            ckyjy=cos(kyjy); skyjy=sin(kyjy);
            v=0;
            if( ic == 1 )
                v = mest(ind) * ckxix*ckyjy;
            elseif ( ic == 2 )
                v = mest(ind) * skxix*ckyjy;
            elseif ( ic == 3 )
                v = mest(ind) * ckxix*skyjy;
            elseif ( ic == 4 )
                v = mest(ind) * skxix*skyjy;
            end
            if( iv==1 )
                west(iy,ix) = west(iy,ix) + v;
```

```
            elseif( iv==2 )
                wcest(iy,ix) = wcest(iy,ix) + v;
            elseif( iv==3 )
                wsest(iy,ix) = wsest(iy,ix) + v;
            end
        end
    end
end

% plot isotropic part of model
figure(2)
subplot(1,4,1)
hold on;
axis xy;
axis( [xmin, xmax, ymin, ymax] );
imagesc( [ymin, ymax], [xmin, xmax], west);
colorbar;
title('A');

% compute B abd q0
pib2 = pi/2;
for ix=[1:Nx]
for iy=[1:Ny]
    Best(iy,ix) = sqrt( (wcest(iy,ix)^2) + (wsest(iy,ix)^2) );
    myq0 = atan2( wsest(iy,ix), wcest(iy,ix) )/2;
    if( myq0 < (-pib2) )
        myq0 = myq0 + pi;
    elseif( myq0 > pib2 )
        myq0 = myq0 - pi;
    end
    q0est(iy,ix) = myq0;
end
end

% plot anisotropic part of model
subplot(1,4,2);
hold on;
axis xy;
axis( [xmin, xmax, ymin, ymax] );
imagesc( [ymin, ymax], [xmin, xmax], Best);
colorbar;
title('B');
subplot(1,4,3)
hold on;
axis xy;
axis( [xmin, xmax, ymin, ymax] );
imagesc( [ymin, ymax], [xmin, xmax], q0est);
colorbar;
title('q0');

% check the integrity of the data kernel by
% computing the traveltime for each ray using
% the same method as was done for the forward
% problem
```

```
Tpre_iso = zeros(N,1);
Tpre = zeros(N,1);
for iray = [1:N] % loop over rays

    % ray info
    x1 = D(iray,1);  y1 = D(iray,3);
    x2 = D(iray,2);  y2 = D(iray,4);
    s1=0; s2=sqrt( (x2-x1)^2 + (y2-y1)^2 );
    q = atan2( (y2-y1), (x2-x1) );

    % sample rays
    s1 = 0;
    s2 = sqrt( (x2-x1)^2 + (y2-y1)^2 );

    % arclength sampling along ray
    Ns = 1000;
    Ds = (s2-s1)/Ns;
    s = Ds*[0:Ns-1]';

    % bookkeeping for sampling slowness along ray
    ixs = 1+floor( Nx*(x1 + s*cos(q))/(xmax-xmin) );
    jys = 1+floor( Ny*(y1 + s*sin(q))/(ymax-ymin) );
    xs = x(ixs);
    ys = y(jys);

    % numerical integration along ray
    I1 = 0;
    I2 = 0;
    for i=[1:Ns]
        wxy = west(jys(i),ixs(i));
        Dwxy = Best(jys(i),ixs(i))*cos(2*(q-q0est(jys(i),ixs(i))));
        I1 = I1 + Ds*wxy;
        I2 = I2 + Ds*Dwxy;
    end

    % save data
    Tpre_iso(iray)=I1;
    Tpre(iray)=I1+I2;

end

figure(3);
subplot(1,2,2);
hold on;
plot( dobs, Tpre, 'ko' );
xlabel('Tobs');
ylabel('Tpre');
title('from integration');

rmserror2 = sqrt((dobs-Tpre)'*(dobs-Tpre)/N);
fprintf('RMS traveltime error (ray): %f percent improvement %f\n',
rmserror2, 100-100*rmserror2/rmstt );

rmserror3 = sqrt((dobs-Tpre_iso)'*(dobs-Tpre_iso)/N);
```

```
fprintf('RMS traveltime error (iso): %f percent improvement %f\n',
rmserror3, 100-100*rmserror3/rmstt );
```