Response of an Elastic Half Space to an Arbitrary 3-D Vector Body Force
(Copyright 2002, Bridget R. Smith and David T. Sandwell)

We wish to calculate the displacement vector \( u(x,y,z) \) on the surface of the Earth due to a vector body force at depth. The approach will be used to simulate both volcanic inflation events with arbitrary source shape and motion on curved and discontinuous faults. For simplicity, we ignore the effects of sphericity. We assume a Poisson material, and maintain constant moduli with depth. The method follows:

i.) Develop the three differential equations relating 3-D vector displacement to a 3-D vector body force.
ii.) Take the 3-D Fourier transform to reduce the partial differential equation to a set of linear algebraic equations.
iii.) Solve the linear system using the symbolic capabilities in Matlab.
iv.) Perform the inverse fourier transform in the \( z \)-direction (depth) by repeated application of the Cauchy Residue Theorem.
v.) Check the analytic solution using the symbolic capabilities in Matlab.
vi.) Solve the Boussinesq Problem to correct the non-zero normal traction on the half-space
vii.) Construct screw dislocation and test with analytic line-source solution.
nii.) Integrate the point-source Green's function to simulate a vertical fault and check with the analytic fault-plane solution.
ix.) 2-D analytic test from 3-D solution
x.) Develop an equivalent body force for a general fault model.
xi.) Force couples on a regular grid
xii.) Fortran source code for elastic half-space model
xiii.) Modify the solution to have a layered half-space (see viscoelastic derivation)
xiv.) Modify the solution for viscoelastic rheology (see viscoelastic derivation)
xv.) Modify the solution to account for surface topography
i) *Develop differential equations*

The following equations relate stress to body forces, stress to strain, and strain to displacement:

\[
\tau_{ij,j} = -\rho_i \tag{1}
\]

\[
\varepsilon_{ij} = \frac{1}{2} (u_{i,j} + u_{j,i}) \tag{2}
\]

\[
\tau_{ij} = \delta_{ij} \lambda \varepsilon_{kk} + 2 \mu \varepsilon_{ij} \tag{3}
\]

Substitute (2) into (3):

\[
\tau_{ij} = \delta_{ij} \lambda u_{k,k} + \mu (u_{i,j} + u_{j,i}) \tag{4}
\]

Substitute (4) into (1):

\[
\delta_{ij} \lambda u_{k,k} + \mu (u_{i,j} + u_{j,i}) = -\rho_i \tag{5}
\]

Or

\[
(\lambda + \mu) u_{k,k} + \mu u_{i,k} = -\rho_i \tag{6}
\]

Expand (6) into three equations for \(u, v, w\):

\[
\mu \nabla^2 u + (\lambda + \mu) \left[ \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 w}{\partial z^2} \right] = -\rho_x \tag{7}
\]

\[
\mu \nabla^2 v + (\lambda + \mu) \left[ \frac{\partial^2 u}{\partial x \partial y} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 w}{\partial z \partial y} \right] = -\rho_y \tag{7}
\]

\[
\mu \nabla^2 w + (\lambda + \mu) \left[ \frac{\partial^2 u}{\partial x \partial z} + \frac{\partial^2 v}{\partial y \partial z} + \frac{\partial^2 w}{\partial z^2} \right] = -\rho_z \tag{7}
\]
ii) 3-D Fourier transform

We assume a point vector body force applied at \( x = 0, y = 0, \) and \( z = a \). In order to simulate a half-space, we also include the effects of an image source at \( x = 0, y = 0, \) and \( z = -a \) [Weertman, 1964]. Equation 8 describes a point body force at both source and image locations, where \( \mathbf{F} \) is a vector force with units of force.

\[
\rho(x,y,z) = \mathbf{F} \delta(x) \delta(y) \delta(z-a) + \mathbf{F} \delta(x) \delta(y) \delta(z+a)
\]  

We then take the 3-D Fourier transform of equation 7 in order to simplify the set of differential equations into algebraic equations. The following identities are used:

\[
\mathfrak{F} \left[ \frac{\partial u(x)}{\partial x} \right] = i2\pi k_x U(k)
\]

\[
\mathfrak{F} [\delta(x)] = 1
\]

\[
\mathfrak{F} [\delta(z-a) + \delta(z+a)] = e^{-2\pi i k_z a} + e^{2\pi i k_z a}
\]

After Fourier transforming, the linear system of equations then becomes

\[
\begin{bmatrix}
  k_x^2 + \frac{\mu k_z^2}{(\lambda + \mu)} & k_x k_y & k_x k_z \\
  k_y k_x & k_y^2 + \frac{\mu k_z^2}{(\lambda + \mu)} & k_y k_z \\
  k_z k_x & k_z k_y & k_z^2 + \frac{\mu k_z^2}{(\lambda + \mu)}
\end{bmatrix}
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix} =
\frac{e^{-i2\pi k_z a}}{4\pi^2}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\]

where \( \mathbf{k} = (k_x, k_y, k_z) \) and \( \mathbf{k}^2 = \mathbf{k} \cdot \mathbf{k} \).
iii) Solve the linear system using Matlab

The displacement solution to equation 10 is found using the symbolic library in Matlab or Mathematica:

\[
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix} = \frac{(\lambda + \mu)}{k\mu(\lambda + 2\mu)} \left[ \begin{array}{ccc}
(k_x^2 + k_y^2) + \frac{\mu k_z^2}{(\lambda + \mu)} & -k_x k_y & -k_x k_z \\
-k_x k_y & (k_y^2 + k_z^2) + \frac{\mu k_x^2}{(\lambda + \mu)} & -k_y k_z \\
-k_x k_z & -k_y k_z & (k_z^2 + k_x^2) + \frac{\mu k_y^2}{(\lambda + \mu)}
\end{array} \right] \frac{1}{4\pi^2} \begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\]

The following Matlab symbolic code verifies that this inversion is correct.

```matlab
% % Matlab routine to check the solution in the kx, ky, kz domain
% %
% % wavenumbers
% pi = sym('pi');
% kx=sym('kx');
% ky=sym('ky');
% kz=sym('kz');
% % elastic constants
% la=sym('la');
% mu=sym('mu');
% lam=la+mu;
% % combinations of wavenumbers
% c=sym('c');
% c=(kx*kx+ky*ky+kz*kz);
% % forward matrix
% A=[kx*kx+mu*c/lam, ky*kx, kz*kx; kx*ky, ky*ky+mu*c/lam, kz*ky; kx*kz, ky*kz, kz*kz+mu*c/lam];
% % solution in Fourier domain, inverse matrix
% B=[c*mu/lam+ky*kx+kz*lam, -kx*ky, -kx*kz; -kx*ky, c*mu/lam+kx*ky+kz*lam, -kz*ky; -kx*kz, -ky*kz, c*mu/lam+kx*ky+kz*lam];
% % normalize
% A=A*lam;
% B=B*lam/(mu*(la+2*mu)*c*c);
% % multiply to get the identity matrix
% C=B*A;
% simplify(C)
ans =  
[ 1, 0, 0]  
[ 0, 1, 0]  
[ 0, 0, 1]
```

- http://topex.ucsd.edu/body_force/
iv) **Perform the inverse Fourier transform along the k_z axis**

We apply the Cauchy Residue Theorem to perform the inverse Fourier transform of each term, for example $V_z(k)$. Our method involves finding the poles of $V_z$, and using calculus of residues to integrate with respect to $k_z$. The first step is to separate the denominator into its poles:

$$ V(k,z) = -F_x \frac{k_x k_y}{4 \pi^2} \frac{(\lambda + \mu)}{\mu(\lambda + 2 \mu)} \int_{-\infty}^{\infty} e^{i2\pi k_z(z-a)} \frac{d k_z}{(k_z + i|k|)^2(k_z - i|k|)^2} $$

(12)

where $k = (k_x, k_y)$ and $|k| = (k_x^2 + k_y^2)^{1/2}$. Here the denominator has four poles total, two at $k_z = -i|k|$, and two at $k_z = i|k|$. We will consider two cases $z > a$, and $z < a$. First consider the case for $z > a$. We have factored the denominator, and have recognized that the integrand will vanish for large, positive $z$ if we close the contour in the upper hemisphere:

The Cauchy Integral Theorem states that, for any analytic function, the following holds for a counter-clockwise path surrounding the pole:

$$ \oint \frac{f(z)}{z-z_0} dz = i2\pi \text{Residue}[f(z), z_0] $$

(13)

In the special case where there are poles of order greater than one, we compute the residue by the following:

(14)
\[
\text{Res}\left[f(z), z_0\right] = \frac{1}{m-1} \lim_{z \to z_0} \frac{d^q}{dz^q} \left[(z-z_0)^m f(z)\right]
\]

where \(z_0 = \text{pole} = \pm i|k|\)

\[m = \text{order} = 2\]

\[q = m - 1 \quad (\text{the } q^{th} \text{ derivative})\]

To satisfy the boundary condition of \(z > a \quad (\text{Im } k_z > 0)\), we must integrate around the positive pole \((k_z = i|k|)\).

\[
V(k, z) = -F_x \frac{k_z k_y}{4\pi^2} \frac{(\lambda + \mu)}{\mu(\lambda + 2\mu)} 2\pi i \text{Res} \left[f(k_z), i|k| \right]
\]

where \(f(k_z) = \frac{e^{i2\pi k_z(z-a)}}{(k_z + i|k|)^2(k_z - i|k|)^2}\)

\[
\text{Res} \left[f(k_z), i|k| \right] = \frac{1}{2 - 1} \lim_{k_z \to i|k|} \frac{d^{2-1}}{dk_z^{2-1}} \left[\frac{(k_z - i|k|)^2 e^{i2\pi k_z(z-a)}}{(k_z + i|k|)^2(k_z - i|k|)^2}\right]
\]

\[
= -2e^{-2\pi|k||z-a|} + 2\pi i(z-a)e^{-2\pi|k||z-a|}
\]

\[
V(k, z) = -F_x \frac{k_z k_y}{2\pi} \frac{(\lambda + \mu)}{\mu(\lambda + 2\mu)} \left(\frac{1 + 2\pi|k|(z-a)}{4|k|^2}\right) e^{-2\pi|k||z-a|}
\]

(15)

Next consider \(z < a\). For this case, we must close the integration path in the lower hemisphere to satisfy the boundary conditions, whereby the only contribution from this integration will be from the \(-i|k|\) pole. This result will be

\[
V(k, z) = -F_x \frac{k_z k_y}{2\pi} \frac{(\lambda + \mu)}{\mu(\lambda + 2\mu)} \left(\frac{1 + 2\pi|k|(z-a)}{4|k|^2}\right) e^{-2\pi|k||z-a|}.
\]

(16)

After completing the integration over the poles in the complex plane for all components of equation 11, the body-force solutions for the half-space containing the source are

\[\text{http://topex.ucsd.edu/body_force/}\]
When we set

we then add both body-force solutions for the source-space and the image-space to obtain the displacement. The result is

\[
\begin{pmatrix}
U(k, z) \\
V(k, z) \\
W(k, z)
\end{pmatrix}_{\text{source}} = BC \begin{pmatrix}
D + \frac{k^2}{|k|^2} - 2\pi|k|(|z| - a) \frac{k^2}{|k|^2} & \frac{-k_y k_z}{|k|^2} \{1 + 2\pi|k|(|z| - a)} & -i2\pi k_y (z - a) \\
-\frac{k_y k_z}{|k|^2} \{1 + 2\pi|k|(|z| - a)} & D + \frac{k^2}{|k|^2} - 2\pi|k|(|z| - a) \frac{k^2}{|k|^2} & -i2\pi k_y (z - a) \\
i2\pi k_y (z - a) & -i2\pi k_y (z - a) & D + 2\pi|k|(|z| - a)
\end{pmatrix} \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix}.
\]

The solutions for the half-space containing the image are

\[
\begin{pmatrix}
U(k, z) \\
V(k, z) \\
W(k, z)
\end{pmatrix}_{\text{image}} = BC \begin{pmatrix}
D + \frac{k^2}{|k|^2} - 2\pi|k|(|z| - a) \frac{k^2}{|k|^2} & \frac{-k_y k_z}{|k|^2} \{1 + 2\pi|k|(|z| - a)} & -i2\pi k_y (-z - a) \\
-\frac{k_y k_z}{|k|^2} \{1 + 2\pi|k|(|z| - a)} & D + \frac{k^2}{|k|^2} - 2\pi|k|(|z| - a) \frac{k^2}{|k|^2} & -i2\pi k_y (-z - a) \\
i2\pi k_y (-z - a) & -i2\pi k_y (-z - a) & D + 2\pi|k|(|z| - a)
\end{pmatrix} \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix},
\]

where

\[
B_i = \frac{e^{-2\pi i |k|(|z| - a)}}{2\pi|k|}, \quad B_i = \frac{e^{2\pi i |k|(-z - a)}}{2\pi|k|}, \quad C = \frac{(\lambda + \mu)}{4\mu(\lambda + 2\mu)}, \quad D = \frac{\lambda + 3\mu}{\lambda + \mu}.
\]

We emphasize the negative sign in front of the image \(F_i\) components, due to the nature of the image source vector, which has the opposite vertical component sign in the source space.

In addition to the displacements, we will also need the vertical derivative of the vertical displacement for the \(F_i\) body force only. The horizontal derivatives are simply \(i2\pi k\) times the displacement. The result is

\[
\frac{\partial W}{\partial z} = -C \left[ e^{-2\pi i |k|(|z| - a)} \left( D + 2\pi |k| (z - a) - 1 \right) + e^{-2\pi i |k|(-z - a)} \left( D + 2\pi |k| (-z - a) - 1 \right) \right].
\]

We then add both body-force solutions for the source-space and the image-space to obtain our final source-image solution:

\[
\begin{pmatrix}
U(k, z) \\
V(k, z) \\
W(k, z)
\end{pmatrix} = \begin{pmatrix}
U(k, z) \\
V(k, z) \\
W(k, z)
\end{pmatrix}_{\text{source}} + \begin{pmatrix}
U(k, z) \\
V(k, z) \\
W(k, z)
\end{pmatrix}_{\text{image}}.
\]

When we set \(z=0\) then the solution is

http://topex.ucsd.edu/body_force/
\[
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}
= 2BC
\begin{bmatrix}
D + \frac{k^2}{|k|^2} + 2\pi |k| |\alpha| \frac{k^2_x}{|k|^2} & \frac{-k_x k_y}{|k|^2} \{1 - 2\pi |k| |\alpha|\} & i2\pi k_x a \\
\frac{-k_x k_y}{|k|^2} \{1 - 2\pi |k| |\alpha|\} & D + \frac{k^2}{|k|^2} + 2\pi |k| |\alpha| |k|^2 y & i2\pi k_y a \\
0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\] (19)

where 
\[
B = \frac{e^{2\pi k \cdot \nu}}{2\pi |k|}, \quad C = \frac{(\lambda + \mu)}{4\mu(\lambda + 2\mu)}, \quad D = \frac{\lambda + 3\mu}{\lambda + \mu}.
\]
v) Check the analytic solution using the symbolic capabilities in Matlab

Again we can use the symbolic capabilities in Matlab to check these answers. In this case we insert this solution into the original differential equation where the equations have been Fourier transformed in the x- and y- directions. The vertical derivatives are carried out symbolically!

```matlab
% matlab routine to check the body-force solution
% pi = sym('pi');
pi2= 4*pi*pi;
% wavenumbers and observation depth z
kx=sym('kx');
ky=sym('ky');
kh2=kx*kx+ky*ky;
kh=sqrt(kh2);
% source depth and elastic constants
a=sym('a');
z =sym('z');
z =a;
la=sym('la');
mu=sym('mu');
lam=la+mu;
% solution in Fourier domain after z-integration
B=exp(-2*pi*kh*(z-a))/(2*pi*kh);
C=lam/(4*mu*(la+2*mu));
D=(la+3*mu)/lam;
% x-body force
Ux = B*C*(D + (ky*ky)/kh2 - 2*pi*kh*(z-a)*kx/kh2); 
Vx = -B*C*kx*ky*(1+2*pi*kh*(z-a))/kh2; 
Wx = -B*C*i*2*pi*kx*(z-a); 
% y-body force
Uy = -B*C*kx*ky*(1+2*pi*kh*(z-a))/kh2;
Vy = B*C*(D + (kx*kx)/kh2 - 2*pi*kh*(z-a)*ky/kh2);
Wy = -B*C*i*2*pi*ky*(z-a); 
% z-body force
Uz = -B*C*i*2*pi*kx*(z-a); 
Vz = -B*C*i*2*pi*ky*(z-a); 
Wz = B*C*(D + 2*pi*kh*(z-a));
% add the solutions
U = Ux + Uy + Uz;
V = Vx + Vy + Vz;
W = Wx + Wy + Wz;
% compute rhx
RHX = mu*(-pi2*kx*kx*U-pi2*ky*ky*U+diff(U,'z',Z));
RHX = RHX + lam*(-pi2*kx*ky*U-pi2*ky*ky*V+i*2*pi*ky*diff(W,'z',1));
simplify(RHX)
ans = 0
% compute rhy
RHY = mu*(-pi2*kx*ky*V-pi2*ky*ky*V+diff(V,'z',Z));
RHY = RHY + lam*(-pi2*ky*ky*U-pi2*ky*ky*V+i*2*pi*ky*diff(W,'z',1));
simplify(RHY)
ans = 0
% compute rhz
```
\[
\begin{align*}
\text{RHZ} &= \mu(-\pi^2 k_x k_x W - \pi^2 k_y k_y W + \text{diff}(W, 'z', Z)) ; \\
\text{RHZ} &= \text{RHZ} + \lambda\text{m}((i^2\pi^2 k_x \text{diff}(U, 'z', 1) + i^2\pi^2 k_y \text{diff}(V, 'z', 1) \text{diff}(W, 'z', Z)) ; \\
\text{simplify(RHZ)} \\
\text{ans} &= 0
\end{align*}
\]
vi) **Determine the surface stress distribution on an elastic half-space to satisfy the boundary condition - The Boussinesq Problem**

We must satisfy the surface boundary condition, that being that the shear stress must be equal to zero. We have included an image source to partially satisfy this boundary condition, but by taking the derivative of the full solution (source and image) with respect to \( z \) and evaluating at \( z=0 \) we find that while the shear tractions disappears, the normal tractions are actually doubled!

\[
\begin{bmatrix}
\tau_{xz}(k) \\
\tau_{yz}(k) \\
\tau_{zz}(k)
\end{bmatrix} = e^{\beta a} \begin{bmatrix} 0 & 0 & 0 \\
-i \frac{k_x}{|k|} \left( \alpha \beta a + \frac{\mu}{(\lambda + 2\mu)} \right) & 0 & (\alpha \beta a - 1) \\
-i \frac{k_y}{|k|} \left( \alpha \beta a + \frac{\mu}{(\lambda + 2\mu)} \right) & 0 & 0
\end{bmatrix} \begin{bmatrix} F_s \\
F_y \\
F_z
\end{bmatrix}
\]

where \( \alpha = \frac{\lambda + \mu}{\lambda + 2\mu} \) \quad \beta = 2\pi|k| \tag{20} 

The following Matlab code verifies this computation of the stress tensor:

```matlab
% matlab routine to compute stress tensor (specifically stresses Txz Tyz Tzz) before and after we incorporate the image

% this computation produces a doubled vertical stress component that requires a Boussinesq correction
pi = sym('pi');
pi2 = 4*pi*pi;

% wavenumbers and observation depth z
kx = sym('kx');
ky = sym('ky');
kh2 = kx*kx + ky*ky;
kx = sqrt(kh2);

% source depth and elastic constants
a = sym('a');
z = sym('z');
la = sym('la');
mu = sym('mu');
lam = la + mu;

% solution in Fourier domain after z-integration
Bs = exp(-2*pi*kh*(z-a))/(2*pi*kh);
Bi = exp(-2*pi*kh*(-z-a))/(2*pi*kh);
C = lam/(4*mu*(la + 2*mu));
D = (la + mu)/lam;

% x-component
Uxs = Bs*la*D + (ky*ky)/kh2 - 2*pi*kh*(z-a)*kx/kh2);
Uxi = Bi*la*D + (ky*ky)/kh2 - 2*pi*kh*(-z-a)*kx/kh2);
Ux = Uxs + Uxi;

Uys = -Bs*kx*ky*(1+2*pi*kh*(z-a))/kh2;
Uyi = -Bi*kx*ky*(1+2*pi*kh*(-z-a))/kh2;
```

http://topex.ucsd.edu/body_force/
Uy = Uys + Uyi;
Uzs = -Bs*C*i*2*pi*kx*(z-a);
Uzi = -(Bi*C*i*2*pi*kx*(-z-a));
Uz = Uzs + Uzi;

% y-component
Vx = Uy;
Vxs = Bs*C*(D + (kx*kx)/kh2 - 2*pi*kh*(z-a)*ky*ky/kh2);
Vyi = Bi*C*(D + (kx*kx)/kh2 - 2*pi*kh*(-z-a)*ky*ky/kh2);
Vy = Vys + Vyi;
Vzs = -Bs*C*i*2*pi*ky*(z-a);
Vzi = -(Bi*C*i*2*pi*ky*(-z-a));
Vz = Vzs + Vzi;

% z-component
Wxs = -Bs*C*i*2*pi*kx*(z-a);
Wxi = Bi*C*i*2*pi*kx*(-z-a);
Wx = Wxs + Wxi;
Wys = -Bs*C*i*2*pi*ky*(z-a);
Wyi = Bi*C*i*2*pi*ky*(-z-a);
Wy = Wys + Wyi;
Wzs = Bs*C*(D + 2*pi*kh*(z-a));
Wzi = -(Bi*C*(D + 2*pi*kh*(-z-a)));
Wz = Wzs + Wzi;

% source
% now compute the derivative of the displacements with respect to z
sUxz = diff(Uxs,'z');
sUyz = diff(Uys,'z');
sUzz = diff(Uzs,'z');
sVxz = diff(Uys,'z');
sVy = diff(Vys,'z');
sVzz = diff(Vzs,'z');
sWxz = diff(Uzs,'z');
sWy = diff(Vzs,'z');
sWzz = diff(Wzs,'z');

% source + image
% now compute the derivative of the displacements with respect to z
Uxz = diff(Ux,'z');
Uyz = diff(Uy,'z');
Uzz = diff(Uz,'z');
Vxz = diff(Vx,'z');
Vyz = diff(Vy,'z');
Vzz = diff(Vz,'z');
Wxz = diff(Wx,'z');
Wyz = diff(Wy,'z');
Wzz = diff(Wz,'z');

% source
% now compute the stresses, tau-xz tau-yz tau-zz
sT11=mu*(limit(sUxz+i*2*pi*kx*Uzs,'z',0));
sT12=mu*(limit(sUyz+i*2*pi*kx*Vzs,'z',0));
sT13=mu*(limit(sUzz+i*2*pi*kx*Wzs,'z',0));
sT21=mu*(limit(sVxz+i*2*pi*ky*Uzs,'z',0));
sT22=mu*(limit(sVyz+i*2*pi*ky*Vzs,'z',0));
sT23=mu*(limit(sVzz+i*2*pi*ky*Wzs,'z',0));
sT31=(la + 2*mu)*limit(sWxz,'z',0)+la*limit(i*2*pi*kx*Uxs+i*2*pi*ky*Uys,'z',0);
sT32=(la + 2*mu)*limit(sWyz,'z',0)+la*limit(i*2*pi*kx*Uys+i*2*pi*ky*Vys,'z',0);
sT33=(la + 2*mu)*limit(sWzz,'z',0)+la*limit(i*2*pi*kx*Uzs+i*2*pi*ky*Vzs,'z',0);
We must impose a negative surface traction to match the boundary condition. We will solve the Boussinesq Problem in order to satisfy the surface boundary condition, for which an arbitrary traction is applied to an elastic half-space.

From the above full space solution, including both source and image at \( a < 0, a > 0 \) respectively, we found that the zero-traction surface boundary condition is only partially satisfied. The remaining surface traction is \( \tau_z(k) \).

We must impose a negative surface traction to match the boundary condition. To do this we solve the Boussinesq problem where an arbitrary traction is applied to an elastic half-space. Our approach is to find the Galerkin vector \( \Gamma_i \), such that we may define complimentary solutions satisfying the boundary condition of \( \tau_{i,j}(x,y,0) = -\tau_3 \) for both displacements and stresses:

\[
\begin{align*}
\mathbf{u}_i &= \Gamma_{i,\mathbf{k}} - \alpha \Gamma_{\mathbf{k},i} \\
\mathbf{\tau}_g &= \lambda (1-\alpha) \delta_{ij} \mathbf{\Gamma}_{i,\mathbf{k}} + \mu \left( \mathbf{\Gamma}_{i,\mathbf{k}j} + \mathbf{\Gamma}_{j,\mathbf{k}i} \right) - 2 \mu \alpha \mathbf{\Gamma}_{\mathbf{k}ij} \\
\text{where } \alpha &= \frac{\lambda + \mu}{\lambda + 2\mu}.
\end{align*}
\]  

(21)

We note that \( \mathbf{\tau}_{i,j} = 0 \) becomes the biharmonic equation for \( \mathbf{\Gamma} \), \( \Gamma_{i,jkk} = 0 \), and we assume \( \Gamma_1 = \Gamma_2 = 0 \) so only \( \Gamma_3 \) will be used, which we will now call \( \Gamma \).

Take the 2-D Fourier transform of the biharmonic equation:

\[
\Gamma_{i,jkk} = \frac{\partial^2}{\partial x^2} \left( \nabla^2 \Gamma \right) + \frac{\partial^2}{\partial y^2} \left( \nabla^2 \Gamma \right) + \frac{\partial^2}{\partial z^2} \left( \nabla^2 \Gamma \right) = 0
\]  

(22)

A general solution to this problem is:

\[
\Gamma(k,z) = (A + B(2\pi |k|z))e^{2\pi k_+ z} + (C + D(2\pi |k|z))e^{-2\pi k_+ z}
\]  

(23)

Our solution will be valid for \( z < 0 \) so we need \( C = D = 0 \) in order to suppress the two solutions that diverge as \( z \to -\infty \). We will use the traction \( -\tau_3(k) \) to balance the stresses arising from the forces \( F_x, F_y \), and \( F_z \).

http://topex.ucsd.edu/body_force/
Boussinesq Solution \[ \tau_{xz} \big|_0 = \tau_{yz} \big|_0 = 0 \quad \tau_{zz} \big|_0 = -\tau_3 \]

\[
U_B = -\alpha \frac{\partial^2 \Gamma}{\partial x \partial z} \quad V_B = -\alpha \frac{\partial^2 \Gamma}{\partial y \partial z} \quad W_B = -\alpha \frac{\partial^2 \Gamma}{\partial z^2} + \nabla^2 \Gamma
\]

\[
\tau_{xz} = \mu \left( \frac{\partial U}{\partial z} + \frac{\partial U}{\partial x} \right) = \mu \frac{\partial}{\partial x} \left[ \nabla^2 \Gamma - 2\alpha \frac{\partial^2 \Gamma}{\partial z^2} \right]
\]

\[
\tau_{yz} = \mu \frac{\partial}{\partial y} \left[ \nabla^2 \Gamma - 2\alpha \frac{\partial^2 \Gamma}{\partial z^2} \right]
\]

Note that if the stress quantity in brackets is zero, then the stress terms will be zero.

We then have \[ \Gamma(k, z) = (A + B\beta z)e^{ik} \quad \Gamma(k, 0) = A \quad \frac{\partial^2 \Gamma}{\partial z^2} \big|_0 = \beta^2(2B + A) \]

where \( \beta = 2\pi |k| \) which we will substitute into \( \tau_{xz} \),

\[
\beta \left[ -A + (1 - 2\alpha)(2B + A) \right] = 0 \tag{25}
\]

\[
A = -B \left( 2 - \frac{1}{\alpha} \right)
\]

We then make a substitution for \( A \), which reduces the Galerkin vector and its derivatives to

\[
\Gamma(k, z) = B \left( \beta z + \frac{1}{\alpha} - 2 \right)e^{ik} \quad \frac{\partial^2 \Gamma}{\partial z^2} = \beta^2 B \left( \beta z + \frac{1}{\alpha} \right)e^{ik} \tag{26}
\]

We pause here briefly to summarize our solutions thus far,

\[
\Gamma(k, z) = -B \left( 2 - \frac{1}{\alpha} - \beta z \right)e^{ik}
\]

\[
U_B = -i\alpha(2\pi)k_x\beta B \left( 1 - \frac{1}{\alpha} - \beta z \right)e^{ik}
\]

\[
V_B = -i\alpha(2\pi)k_y\beta B \left( 1 - \frac{1}{\alpha} - \beta z \right)e^{ik}
\]

\[
W_B = -\alpha\beta^2 B \left( \frac{1}{\alpha} - \beta z \right)e^{ik}
\]

\[
\tau_{xx} = 2\beta B \left\{ \mu \alpha (2\pi k^2) + \lambda(1 - \alpha)(2\pi k^2)^2 + \mu \alpha (2\pi k^2) \beta z \right\}e^{ik}
\]

\[
\tau_{yy} = 2\beta B \left\{ \mu \alpha (2\pi k^2) + \lambda(1 - \alpha)(2\pi k^2)^2 + \mu \alpha (2\pi k^2) \beta z \right\}e^{ik}
\]

\[
\tau_{zz} = -2\alpha\mu(\beta)^3 B(1 - \beta z)e^{ik}
\]
Our solutions differ from those in the Stekete paper because we are solving for a solution at \( z < 0 \).

Now use \( \tau_{zz} \) to solve for \( B \):

\[
\tau_{zz} = -2\alpha \mu B^3 [1 - \beta z] e^{\beta z}
\]

\[
\tau_{zz}(k,0) = -2\alpha \mu B^3 = -\tau_3 \quad (28)
\]

Solving for \( B \),

\[
B(k) = \frac{\tau_3(k)}{\mu B^3}
\]

We then make the appropriate substitutions for \( B \) and find our final Boussinesq solutions where \( dW_B / dz \) is needed for the stress computation.

\[
U_B = -i2\pi k \cdot \frac{1}{2\mu} \frac{\tau_3(k)}{B^2} \left[ 1 - \frac{1}{\alpha} - \beta z \right] e^{\beta z}
\]

\[
V_B = -i2\pi k \cdot \frac{1}{2\mu} \frac{\tau_3(k)}{B^2} \left[ 1 - \frac{1}{\alpha} - \beta z \right] e^{\beta z}
\]

\[
W_B = -\frac{1}{2\mu} \frac{\tau_3(k)}{\beta} \left[ 1 - \frac{1}{\alpha} - \beta z \right] e^{\beta z} \quad (29)
\]

\[
\frac{dW_B}{dz} = -\frac{\tau_3(k)}{2\mu} \left[ 1 - 1 - \beta z \right] e^{\beta z}
\]

\[
\begin{bmatrix}
-i \frac{k}{|k|} \left( \alpha \beta a + \frac{\mu}{\lambda + 2\mu} \right) F_x \\
-i \frac{k}{|k|} \left( \alpha \beta a + \frac{\mu}{\lambda + 2\mu} \right) F_y \\
(\alpha \beta a - 1) F_z
\end{bmatrix}
\]

where \( \tau(k) = e^{\beta a} \)

\[
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix} = \begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{source}} + \begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{image}} + \begin{bmatrix}
U_B(k) \\
V_B(k) \\
W_B(k)
\end{bmatrix} \quad (30)
\]

To re-cap, we now have three solutions that comprise our total solution:
The following Matlab code verifies Boussinesq solution:

```matlab
% matlab routine to check the algebra for the Boussinesq solution from
% Steketee, J. A., On Volterra's dislocations in a semi-infinite elastic
% pi = sym('pi');
pi2= 4*pi*pi;
% wavenumbers and observation depth z
kx=sym('kx');
ky=sym('ky');
kh2=kx*kx+ky*ky;
kh=sqrt(kh2);
lam=sym('lam');
mu=sym('mu');
al=(lam+mu)/(lam+2*mu);
syms Gam x y z;
ex=exp(2*pi*kh*z);
% check Tau_xz/yz for correct algebra
% define U,V,W as function of tau
laplace=diff(Gam,x,2)+ diff(Gam,y,2)+ diff(Gam,z,2);
U=-al*diff((diff(Gam,x)),z);
V=-al*diff((diff(Gam,y)),z);
W=-al*diff(Gam,z,2)+ laplace;
% define tau_xz as in notes
tauxz=(mu)*(diff(U,z)+ diff(W,x));
% define tauzz as function of Galerkin vector
simplify_tauxz=mu*diff((laplace - 2*al*diff(Gam,z,2)),x);
% take difference to find mistakes
tauzero=simplify(tauxz-simplify_tauxz)
ans:  tauzero = 0
% check bracket term for correct algebra
% unknown constants
A=sym('A');
B=sym('B');
beta=2*pi*kh;
G=ex*(A+B*beta*z);
% take the derivatives of the galerkin vector
Gz = diff(G,'z');
Gzz= diff(Gz,'z');
% evaluate at z=0
G0   = limit(G,'z',0);
Gz0  = limit(Gz,'z',0);
Gzz0 = limit(Gzz,'z',0);
% term in bracket
Bracket=((2*pi*i*kx)*G0 + ((2*pi*i*ky)*G0 + Gzz0) - 2*al*Gzz0;
% solve Bracket for 'A'
A=solve(Bracket,A);
% define A as in notes
myA=-B*(2-1/al);
% take difference to find mistakes
Azero=simplify(myA-A)
ans:  Azero = 0
```

http://topex.ucsd.edu/body_force/
% check new solutions with A substitution
% evaluate Galerkin vector for solution of A
G=limit( (ex*(A+B*beta*z)),'A',myA);
% take the derivatives of the Galerkin vector
Gz = diff(G,'z');
Gzz = diff(Gz,'z');
% check Galerkin, U,V,W,txx,tyy,tzz for mistakes
Galerkin=-B*(2-1/al-beta*z)*ex;
zeroGalerkin=simplify(G-Galerkin)
ans: zeroGalerkin = 0

U=-al*2*pi*i*kx*Gz;
newU=i*al*2*pi*kx*beta*B*(1-1/al-beta*z)*ex;
zeroU=simplify(U-newU)
ans: zeroU = 0

V=-al*2*pi*i*ky*Gz;
newV=i*al*2*pi*ky*beta*B*(1-1/al-beta*z)*ex;
zeroV=simplify(V-newV)
ans: zeroV = 0

W=(1-al)*Gzz + (2*pi*i*kx)^2*G + (2*pi*i*ky)^2*G;
newW=al*beta^2*B*(1/al-beta*z)*ex;
zeroW=simplify(W-newW)
ans: zeroW = 0

txx= lam*( (2*pi*i*kx)*U + (2*pi*i*ky)*V + diff(W,z) ) + 2*mu*(2*pi*i*kx*U);
newtxx=2*beta*B*(mu*al*(2*pi*kx)^2 + lam*(1-al)*(2*pi*ky)^2 + mu*al*(2*pi*ky)^2*beta*z)*ex;
zerotxx=simplify(txx-newtxx)
ans: zerotxx = 0

tyx= lam*( (2*pi*i*kx)*U + (2*pi*i*ky)*V + diff(W,z) ) + 2*mu*(2*pi*i*ky*V);
newtyx=2*beta*B*(mu*al*(2*pi*ky)^2 + lam*(1-al)*(2*pi*kx)^2 + mu*al*(2*pi*ky)^2*beta*z)*ex;
zerotyy=simplify(tyx-newtyy)
ans: zerotyy = 0

tzx= lam*( (2*pi*i*ky)*U + (2*pi*i*ky)*V + diff(W,z) ) + 2*mu*(2*pi*i*ky*V);
newtzx=2*al*mu*beta^3*B*(1-beta*z)*ex;
zerotzz=simplify(tzx-newtzx)
ans: zerotzz = 0

% sum of derivatives of txz, tyz, tzz should be zero if solutions satisfy the differential equation
txz = mu*(diff(newU,'z') + i*2*pi*kx*newW);
tyz = mu*(diff(newV,'z') + i*2*pi*ky*newW);
Oursum= simplify((i*2*pi*kx*txx + i*2*pi*ky*tyz + diff(tzz,'z'))
ans: Oursum = 0
% check the Stekete Solution
Sex=exp(-2*pi*kh*z)/(2*pi*kh);% check Stekete Solution
SU=Sex*i*2*pi*kx*(2*pi*kh*z-1/al+1);% SU=Sex*i*2*pi*kx*(2*pi*kh*z-1/al+1);
SV=Sex*i*2*pi*ky*(2*pi*kh*z-1/al+1);% SV=Sex*i*2*pi*ky*(2*pi*kh*z-1/al+1);
SW=Sex*2*pi*kh*(2*pi*kh*z+1/al);% SW=Sex*2*pi*kh*(2*pi*kh*z+1/al);
STxz=mu*(diff(SU,'z')+i*2*pi*kx*SW);% STxz=mu*(diff(SU,'z')+i*2*pi*kx*SW);
STyz=mu*(diff(SV,'z')+i*2*pi*ky*SW);% STyz=mu*(diff(SV,'z')+i*2*pi*ky*SW);
STzz=(lam+2*mu)*diff(SW,'z')+lam*[i*2*pi*ky*SU+i*2*pi*ky*SV];% STzz=(lam+2*mu)*diff(SW,'z')+lam*[i*2*pi*ky*SU+i*2*pi*ky*SV];
Steketeesum=simplify(i*2*pi*kx*STxz+i*2*pi*ky*STyz+diff(STzz,'z'))
ans: Steketeesum = 0

% check final Boussinesq solutions
syms t3
newB = t3/(2*al*mu*beta^3);
% evaluate U,V,W at new solution for B, take difference in notes to find any mistakes
UB1=limit(U,'B',newB);
myUB1 = -i*2*pi*txz/(2*mu)*t3/(beta^2)*(1-1/al-beta*z)*ex;
zeroUB1=simplify(UB1-myUB1)
ans: zeroUB1 = 0
VB1=limit(V,'B',newB);
myVB1 = -i*2*pi*t3/(2*mu)*t3/(beta^2)*(1-1/al-beta*z)*ex;
zeroVB1=simplify(VB1-myVB1)
ans: zeroVB1 = 0
WB1=limit(W,'B',newB);
myWB1 = -1/(2*mu)*t3/beta*(1/al-beta*z)*ex;
zeroWB1=simplify(WB1-myWB1)
ans: zeroWB1 = 0

The following Matlab code verifies that the original differential equation is satisfied with the Boussinesq contribution:

pi = sym('pi');
piz = 4*pi*pi;piz = 4*pi*pi;

% wavenumbers and observation depth z
kx=sym('kx');
ky=sym('ky');
kh2=kx*kx+ky*ky;kx=sym('kx');
kh=sqrt(kh2);
% source depth and elastic constants
a = sym('a');
z = sym('z');
lam = sym('lam');
mu = sym('mu');

% solution in Fourier domain after z-integration
Bs = exp(-2*pi*kh*(z-a))/(2*pi*kh);
Bi = exp(-2*pi*kh*(-z-a))/(2*pi*kh);
C = (lam+mu)/(4*mu*(lam+2*mu));
D = (lam+3*mu)/(lam+mu);

% constants
alpha = (lam + mu)/(lam + 2*mu);
beta = 2*pi*kh;
ab1 = (1/alpha + beta.*z);
ab2 = (1/alpha + beta.*z + 1);
ez = exp(beta.*z);
A = exp(beta.*a);
B = alpha*beta.*a;
MUU = mu/(lam+2*mu);

% generate components of stress tensor

% x-component
Uxs = Bs*C*(D + (ky*ky)/kh2 - 2*pi*kh*(z-a)*kx/kx/kh2);
Uxi = Bi*C*(D + (ky*ky)/kh2 - 2*pi*kh*(-z-a)*kx/kx/kh2);
Ubx = -2*pi*i/(2*mu)*kx.*t3x./(beta.*beta).*(1-ab1).*ez;
Ux = Uxs + Uxi + Ubx;
Uys = -Bs*C*kx*ky*(1+2*pi*kh*(z-a))/kh2;
Uyi = -Bi*C*kx*ky*(1+2*pi*kh*(-z-a))/kh2;
Uby = -2*pi*i/(2*mu)*kx.*t3y./(beta.*beta).*(1-ab1).*ez;
Uy = Uys + Uyi + Ubz;
Uz = -Bs*C*2*pi*kx*(z-a);
Uzi = -(Bi*C*2*pi*kx*(-z-a));
% make z image negative
Ubz = -2*pi*i/(2*mu)*kx.*t3z./(beta.*beta).*(1-ab1).*ez;

% y-component
Vbx = -2*pi*i/(2*mu)*ky.*t3x./(beta.*beta).*(1-ab1).*ez;
Vx = Uys + Uyi + Vbx;
Vys = Bs*C*(D + (kx*kx)/kh2 - 2*pi*kh*(z-a)*ky*ky/kh2);
Vyi = Bi*C*(D + (kx*kx)/kh2 - 2*pi*kh*(-z-a)*ky*ky/kh2);
Vby = -2*pi*i/(2*mu)*ky.*t3y./(beta.*beta).*(1-ab1).*ez;
Vy = Vys + Vyi + Vby;
Vz = -Bs*C*i*2*pi*ky*(z-a);
Vzi = -(Bi*C*i*2*pi*ky*(-z-a));
% make z image negative
Vbz = -2*pi*i/(2*mu)*ky.*t3z./(beta.*beta).*(1-ab1).*ez;
Vz = Vz + Vzi + Vbz;

% z-component
Wxs = -Bs*C*i*2*pi*kx*(z-a);
Wxi = Bi*C*i*2*pi*kx*(-z-a);
Wbx = -1/(2*mu)*t3x./(beta).*((1/alpha - beta.*z).*ez;
Wx = Wxs + Wxi + Wbx;
Wys = -Bs*C*i*2*pi*ky*(z-a);
Wy = Bi*C*i*2*pi*ky*(-z-a);
Wby = -1/(2*mu)*t3y./(beta).*((1/alpha - beta.*z).*ez;
Wy = Wys + Wyi + Wby;
Wzs = Bsc*(D + 2*pi*kh*(z-a));
Wzi = -(Bi*C*(D + 2*pi*kh*(-z-a)));
Wbz = -1/(2*mu)*t3z./(beta).*ez;
Wz = Wzs + Wzi + Wbz;

% add together to check differential equation
U = Ux + Uy + Uz;
V = Vx + Vy + Vz;
W = Wx + Wy + Wz;

% compute rhx
RHX = mu*(-pi2*kx*kx*U-pi2*ky*ky*U+diff(U,'z',2));
RHX = simplify(RHX + (lam+mu)*(-pi2*kx*kx*U-pi2*kx*ky*V+i*2*pi*kx*diff(W,'z',1)));

% compute rhy
RHY = mu*(-pi2*kx*kx*V-pi2*ky*ky*V+diff(V,'z',2));
RHY = simplify(RHY + (lam+mu)*(-pi2*kx*ky*U-pi2*ky*ky*V+i*2*pi*ky*diff(W,'z',1)));

% compute rhz
RHZ = mu*(-pi2*kx*kx*W-pi2*ky*ky*W+diff(W,'z',2));
RHZ = simplify(RHZ + (lam+mu)*(i*2*pi*kx*Ux+i*2*pi*ky*Vx+i*2*pi*ky*Vz+i*2*pi*ky*Wz+i*2*pi*ky*Wz+i*2*pi*ky*Wz+i*2*pi*ky*Wz+i*2*pi*ky*Wz+i*2*pi*ky*Wz+i*2*pi*ky*Wz+i*2*pi*ky*Wz));

ans:  
RHX = 0
RHY = 0
RHZ = 0

% now compute the derivative of the displacements with respect to z
Uxz = diff(Ux,'z');
Uyz = diff(Uy,'z');
Uzz = diff(Uz,'z');
Vxz = diff(Vx,'z');
Vyz = diff(Vy,'z');
Vzz = diff(Vz,'z');
Wxz = diff(Wx,'z');
Wyz = diff(Wy,'z');
Wzz = diff(Wz,'z');

% now compute the stresses, tau-xz tau-yz tau-ztau-xz
T11=mu*(limit(Uxz+i*2*pi*kx*Wx,'z',0));
T12=mu*(limit(Uyz+i*2*pi*kx*Wy,'z',0));
T13=mu*(limit(Uzz+i*2*pi*kx*Wz,'z',0));
T21=mu*(limit(Vxz+i*2*pi*ky*Wx,'z',0));
T22=mu*(limit(Vyz+i*2*pi*ky*Wy,'z',0));
T23=mu*(limit(Vzz+i*2*pi*ky*Wz,'z',0));
T31=(lam + 2*mu)*limit(Wxz,'z',0)+lam*limit(i*2*pi*kx*Ux+i*2*pi*ky*Vx,'z',0);
T32=(lam + 2*mu)*limit(Wyz,'z',0)+lam*limit(i*2*pi*kx*Uy+i*2*pi*ky*Vy,'z',0);
T33=(lam + 2*mu)*limit(Wzz,'z',0)+lam*limit(i*2*pi*kx*Uz+i*2*pi*ky*Vz,'z',0);

% new stress tensor
Tauz=simplify([T11, T12, T13;
T21, T22, T23;
T31, T32, T33]);

newTXZ = simplify([T11 T12 T13])
newTYZ = simplify([T21 T22 T23])
newTZZ = simplify([T31 T32 T33])

ans: 
newTXZ = [ 0, 0, 0]
newTYZ = [ 0, 0, 0]
newTZZ = [ 0, 0, 0]
vii) Screw dislocation and numerical test with line source

We will now construct a screw dislocation source by taking the derivative of the point source in the direction normal to the fault plane. This corresponds to multiplication in the Fourier transform domain.

\[
E - W \text{ trending fault: } \int \frac{df(x,y)}{dy} = i2\pi k_y F(k_x, k_y)
\]  \hspace{1cm} (32)

\[
N - S \text{ trending fault: } \int \frac{df(x,y)}{dx} = i2\pi k_x F(k_x, k_y)
\]  \hspace{1cm} (33)

In practice, the body forces due to the stress discontinuity across a fault plane are approximated by the derivative of a Gaussian function, effectively producing a model fault with a finite thickness. Curved faults are constructed with short line segments having cosine tapered ends and are typically 6-10 km long. The fault trace is imbedded in a two-dimensional grid which is Fourier transformed, multiplied by the transfer functions above, and inverse Fourier transformed. The horizontal complexity of the fault system has no effect on the speed of the computation. However, variations in locking depth along the fault system require computing the model for each different locking depth and the outputs summed to form the full solution.

In a previous derivation (http://topex.ucsd.edu/geodynamics/13fault_disp.pdf) we showed that the y-displacement \( v \) due to a line source is given by

\[
v(x) = \frac{-A}{\pi\mu} \frac{x}{(x^2 + a^2)}
\]  \hspace{1cm} (33)

where \( A \) is the line force per unit length. For comparison with this solution we must divide the fourier solution by \( \Delta x \) which corresponds to the cell-size in the \( x \)-direction. The comparison between the two solutions is shown in Figure 1. For a source depth of 10 km, the \( x \)-length of the fourier grid must be greater than 8192 to achieve better than \( 10^{-3} \) numerical accuracy. A smaller dimension can be used if the fault has a finite length in the \( y \)-direction.
The following Matlab code was used to compute and display the above analytical comparison.

```matlab
% matlab program to find displacement field due to line source
%
% set up line source geometry
depth=-10; % depth (km)
z=0; % observation depth
%
% lame parameters
lam=1;
mu=1;
alpha = (lam + mu)/(lam+2*mu);
%
% generate wave numbers in x direction
Lx=512; % length
nx=512; % number of pixels in x
dx=Lx/nx; % spacing in x
nx2=nx/2;
kx= -nx2: (nx2-1); % set up x wave number vector
kx= kx./Lx; % wave numbers in x
%
% generate wave numbers in y direction
Ly=512; % length
ny=512; % number of pixels in y
dy=Ly/ny; % spacing in y
ny2=ny/2;
ky= -ny2: (ny2-1); % set up y wave number vector
ky= ky'./Ly; % wave numbers in y
%
% set up fault azimuth
theta=90.;
rad=pi/180.;
thetas=theta*rad;
theta=thetas+pi/2.;
%
% set up pt y-source force in center of grid
yy=(1:ny)';
xx=nx/2*ones(length(yy),1);
force=ones(length(yy),1);
```

http://topex.ucsd.edu/body_force/
% set up force option
px=zeros(ny,nx);
py=zeros(ny,nx);
pz=zeros(ny,nx);
po=zeros(ny,nx);
for t=1:length(xx)
    po(yy(t),xx(t))=force(t);
end
py=po.*sin(thetas);
px=po.*cos(thetas);
pz=po;
% fourier transform pt force grid into k domain
pkx=fftshift(fft2(px));
pky=fftshift(fft2(py));
pkz=fftshift(fft2(pz));
% generate the 2-D wavenumber arrays
[kxx,kyy]=karray(kx,ky);
% generate model for source and image [sU,sV,sW], [iU,iV,iW]
[sux,svx,sux,suy,suv,svy,suz,suz,suz]=sourceU(kxx,kyy,depth,lam,mu,z);
[iux,ivx,iux,ivy,ivy,ivy,ivy,ivy,ivy]=imageU(kxx,kyy,depth,lam,mu,z);
% add the source-solutions
sU = sux + suy + suz;
sV = svx + svy + svz;
sW = swx + swy + swz;
% add the image-solutions
iU = iux + ivx + iux;
iV = ivx + ivy + ivz;
iW = iw + iw + iw;
% add both source and image solutions together
usi = sU + iU;
vs = sV + iV;
ws = sW + iW;
% generate tau solutions (stress in z)
tauz(kxx,kyy,depth,lam,mu,z);
% generate the boussinesq solution:
Ub = boussinesq(kxx,kyy,lam,mu,t3x,t3y,t3z,z);
% add the boussinesq solutions to the source-image solution:
U = Usi + Ub;
V = Vsi + Vb;
W = Wsi + Wb;
% generate force couple
k = cos(thetan).*kxx + sin(thetan).*kyy ;
Vcouple = -2*pi*i.*k.*pky ;
% inverse fft for integrated source
v = real(ifft2(fftshift(Vcouple)));
% set up analytic solution (Weertman)
xp=(nx2-1):1:nx2;
xp=xp*dx;
vanalytical=-1*(dx.*xp./((xp.*xp+(depth.*depth))/(pi*mu)));
% set up x and y axis for image
xd= (0:dx:Lx)';
xd= -@:dy:Ly)';
% display
subplot(2,1,1),plot(axisx,vanalytical,'b',axisx,vfourier,:')
legend('analytic solution', 'fourier solution')
title('Displacement profile of a double couple LINE source')
xlabel('x-distance from the fault (km)')
ylabel('displacement amplitude ')
vdiff = vfourier - vanalytical;
subplot(2,1,2),plot(axisx,vdiff)
title('Amplitude difference between analytic and fourier solution')
xlabel('x-distance from the fault (km)')
ylabel('displacement amplitude difference')
viii) Vertical integration of point source to create fault plane and numerical test

We integrate the point source Green’s function between depths $d_1$ and $d_2$ to simulate a fault plane. For a complex dipping fault, this integration could be done numerically. However, if the faults are assumed to be vertical, the integration can be performed analytically. The body force is applied between the deep depth $d_1$ (e.g., minus infinity) and the shallow depth $d_2$. The displacement or stress (derivatives are computed analytically) can be evaluated at any depth $z$ above $d_2$. Note the displacement is the sum of three terms: a source, an image, and a Boussinesq correction.

For this exercise, we'll assume $z = 0$. The integration will be over the source depth, $a$.

Let $z' = a$. An examination of the solution above shows that there are only two basic integrals to perform.

\begin{align*}
(1) &= \left(2\pi|k|\right)^{-1} \int_{d_1}^{d_2} e^{2\pi|k|z'} dz' \\
(2) &= \int_{d_1}^{d_2} \frac{e^{2\pi|k|z'}}{z'} dz'
\end{align*}

The first integral is

\begin{equation}
(1) = \left(2\pi|k|\right)^{-1} \left(e^{2\pi|k|d_2} - e^{2\pi|k|d_1}\right) \tag{34}
\end{equation}

The second integral is performed using integration by parts

\begin{align*}
(2) &= \left(2\pi|k|\right)^{-1} \left(e^{2\pi|k|d_2}(2\pi|k|d_2 - 1) - e^{2\pi|k|d_1}(2\pi|k|d_1 - 1)\right) \\
\end{align*}

After a little algebra, the solution matrix for both source and image is

\begin{align*}
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{source}} &=
\begin{bmatrix}
\phi_{x} & \phi_{y} & \phi_{z} \\
U_{xy} & U_{yx} & U_{xz} \\
V_{yz} & V_{zy} & V_{zz}
\end{bmatrix}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\quad
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{image}} &=
\begin{bmatrix}
\phi_{x} & \phi_{y} & \phi_{z} \\
U_{xi} & U_{yi} & U_{zi} \\
V_{zi} & V_{yi} & V_{zi}
\end{bmatrix}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\end{align*}

\begin{align*}
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{source}} &=
\begin{bmatrix}
\phi_{x} & \phi_{y} & \phi_{z} \\
U_{xy} & U_{yx} & U_{xz} \\
V_{yz} & V_{zy} & V_{zz}
\end{bmatrix}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\quad
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{image}} &=
\begin{bmatrix}
\phi_{x} & \phi_{y} & \phi_{z} \\
U_{xi} & U_{yi} & U_{zi} \\
V_{zi} & V_{yi} & V_{zi}
\end{bmatrix}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\end{align*}

\begin{align*}
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{source}} &=
\begin{bmatrix}
\phi_{x} & \phi_{y} & \phi_{z} \\
U_{xy} & U_{yx} & U_{xz} \\
V_{yz} & V_{zy} & V_{zz}
\end{bmatrix}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\quad
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{image}} &=
\begin{bmatrix}
\phi_{x} & \phi_{y} & \phi_{z} \\
U_{xi} & U_{yi} & U_{zi} \\
V_{zi} & V_{yi} & V_{zi}
\end{bmatrix}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\end{align*}

\begin{align*}
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{source}} &=
\begin{bmatrix}
\phi_{x} & \phi_{y} & \phi_{z} \\
U_{xy} & U_{yx} & U_{xz} \\
V_{yz} & V_{zy} & V_{zz}
\end{bmatrix}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\quad
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{image}} &=
\begin{bmatrix}
\phi_{x} & \phi_{y} & \phi_{z} \\
U_{xi} & U_{yi} & U_{zi} \\
V_{zi} & V_{yi} & V_{zi}
\end{bmatrix}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\end{align*}
where the source solutions are:

\[
U_{x\alpha}(k) = \frac{C}{\beta^2} \left\{ e^{-\beta(z+d_2)} \left[ D + \frac{k_y^2}{|k|^2} - \frac{k_z^2}{|k|^2} (1 + \beta(z - d_2)) \right] - e^{-\beta(z + d_1)} \left[ D + \frac{k_y^2}{|k|^2} - \frac{k_z^2}{|k|^2} (1 + \beta(z - d_1)) \right] \right\}
\]

\[
U_{y\alpha}(k) = -\frac{C}{\beta^2} \frac{k_x k_y}{|k|} \left\{ e^{-\beta(z - d_2)} (2 + \beta(z - d_2)) - e^{-\beta(z - d_1)} (2 + \beta(z - d_1)) \right\}
\]

\[
U_{z\alpha}(k) = -i \frac{C}{\beta^2} \frac{k_y}{|k|} \left\{ e^{-\beta(z-d_2)} (1 + \beta(z - d_2)) - e^{-\beta(z-d_1)} (1 + \beta(z - d_1)) \right\}
\]

\[
V_{y\alpha}(k) = \frac{C}{\beta^2} \left\{ e^{-\beta(z-d_2)} \left[ D + \frac{k_x^2}{|k|^2} - \frac{k_z^2}{|k|^2} (1 + \beta(z - d_2)) \right] - e^{-\beta(z-d_1)} \left[ D + \frac{k_x^2}{|k|^2} - \frac{k_z^2}{|k|^2} (1 + \beta(z - d_1)) \right] \right\}
\]

\[
V_{z\alpha}(k) = -i \frac{C}{\beta^2} \frac{k_x}{|k|} \left\{ e^{-\beta(z-d_2)} (1 + \beta(z - d_2)) - e^{-\beta(z-d_1)} (1 + \beta(z - d_1)) \right\}
\]

\[
W_{z\alpha}(k) = \frac{C}{\beta^2} \left\{ e^{-\beta(z-d_2)} [D + 1 + \beta(z - d_2)] - e^{-\beta(z-d_1)} [D + 1 + \beta(z - d_1)] \right\}
\]

(37)

and the image solutions are:

\[
U_{x\alpha}(k) = \frac{C}{\beta^2} \left\{ e^{\beta(z + d_2)} \left[ D + \frac{k_y^2}{|k|^2} - \frac{k_z^2}{|k|^2} (1 - \beta(z + d_2)) \right] - e^{\beta(z + d_1)} \left[ D + \frac{k_y^2}{|k|^2} - \frac{k_z^2}{|k|^2} (1 - \beta(z + d_1)) \right] \right\}
\]

\[
U_{y\alpha}(k) = -\frac{C}{\beta^2} \frac{k_y k_x}{|k|} \left\{ e^{\beta(z + d_2)} (2 - \beta(z + d_2)) - e^{\beta(z + d_1)} (2 - \beta(z + d_1)) \right\}
\]

\[
U_{z\alpha}(k) = i \frac{C}{\beta^2} \frac{k_x}{|k|} \left\{ e^{\beta(z + d_2)} (1 - \beta(z + d_2)) - e^{\beta(z + d_1)} (1 - \beta(z + d_1)) \right\}
\]

\[
V_{y\alpha}(k) = \frac{C}{\beta^2} \left\{ e^{\beta(z + d_2)} \left[ D + \frac{k_x^2}{|k|^2} - \frac{k_z^2}{|k|^2} (1 - \beta(z + d_2)) \right] - e^{\beta(z + d_1)} \left[ D + \frac{k_x^2}{|k|^2} - \frac{k_z^2}{|k|^2} (1 - \beta(z + d_1)) \right] \right\}
\]

\[
V_{z\alpha}(k) = i \frac{C}{\beta^2} \frac{k_x}{|k|} \left\{ e^{\beta(z + d_2)} (1 - \beta(z + d_2)) - e^{\beta(z + d_1)} (1 - \beta(z + d_1)) \right\}
\]

\[
W_{z\alpha}(k) = \frac{C}{\beta^2} \left\{ e^{\beta(z + d_2)} [D + 1 - \beta(z + d_2)] - e^{\beta(z + d_1)} [D + 1 - \beta(z + d_1)] \right\}
\]

(38)

where \( C = \frac{(\lambda + \mu)}{4\mu(\lambda + 2\mu)} \), \( D = \frac{\lambda + 3\mu}{\lambda + \mu} \), \( \alpha = \frac{\lambda + \mu}{\lambda + 2\mu} \), \( |k| = (k_x^2 + k_y^2)^{1/2} \), \( \beta = 2\pi|k| \)

* denotes components that are multiplied by a negative sign when computing.
Note that now $F$ has units of force/length since we have integrated over depth.

We can rewrite these expressions to isolate the factors containing $e^{-\beta z}$ and $-\beta Ze^{-\beta z}$. This will help to simplify the computer code as well as to prepare us for the derivation of the layered models. The solution really contains only 6 independent functions because the tensor is symmetric.

$$U'(k, Z) = \begin{bmatrix} U_x & U_y & U_z \\ U_y & V_y & V_z \\ U_z & V_z & W_z \end{bmatrix} \quad \text{and} \quad U^i(k, Z) = \begin{bmatrix} U_x & U_y & U_z \\ U_y & V_y & V_z \\ -U_z & -V_z & -W_z \end{bmatrix}$$ (39)

The 6 independent functions are:

$$\left[ \begin{array}{c} U_x \\ U_y \\ U_z \\ V_y \\ V_z \\ W_z \end{array} \right] = \frac{C}{\beta^2} \begin{bmatrix} D + \frac{k_x^2}{|k|^2} & -\frac{k_x^2}{|k|^2} & \frac{k_x^2}{|k|^2} \\ -2 \frac{k_x k_y}{|k|^2} & -\frac{k_y^2}{|k|^2} \\ -i \frac{k_x}{|k|} & i \frac{k_x}{|k|} \\ \frac{k_y^2}{|k|^2} & D + \frac{k_y^2}{|k|^2} \\ -i \frac{k_y}{|k|} & i \frac{k_y}{|k|} \\ D + 1 & \end{bmatrix} \begin{bmatrix} e^{-\beta|z|} \\ \beta |Z| e^{-\beta|z|} \end{bmatrix}$$ (40)

The source and image terms can be written as

$$\begin{bmatrix} U(k) \\ V(k) \\ W(k) \end{bmatrix} = \begin{bmatrix} U'(k, z - d_z) - U'(k, z - d_i) \\ F_x \\ F_y \\ F_z \end{bmatrix} + \begin{bmatrix} U'(k, z + d_z) - U'(k, z + d_i) \\ F_x \\ F_y \\ F_z \end{bmatrix}$$ (41)

Again we can use the symbolic capabilities in Matlab to check these answers. As in the previous case, we insert this solution into the original differential equation where the equations have been Fourier transformed in the $x$- and $y$- directions. The vertical derivatives are carried out symbolically as before.
% matlab routine to check the depth-integrated solution for source space

\[ pi = \text{sym}'pi'\;
\]  
\[ pi2=4*pi*\text{pi}; \]

% wavenumbers and observation depth z  
\[ kx=\text{sym}'kx'; \]
\[ ky=\text{sym}'ky'; \]
\[ kh2=kx*kx+ky*ky; \]
\[ kh=\text{sqrt}(kh2); \]

% source depth and elastic constants  
\[ d1=\text{sym}'d1'; \]
\[ d2=\text{sym}'d2'; \]
\[ z =\text{sym}'z'; \]
\[ d1=z-d1; \]
\[ d2=z-d2; \]
\[ la=\text{sym}'la'; \]
\[ mu=\text{sym}'mu'; \]
\[ lam=la+mu; \]

% solution in Fourier domain after z-integration  
\[ C = lam/(4*mu*(la+2*mu)); \]
\[ D = (la+3*mu)/lam; \]
\[ B = 2*pi*kh; \]
\[ D1 = B*d1; \]
\[ D2 = B*d2; \]
\[ E1 = \text{exp}(-D1); \]
\[ E2 = \text{exp}(-D2); \]

% compute the x displacements  
\[ t2 = E2*(C + ky*ky/kh2 - kx*kx/kh2*(1+D2)); \]
\[ t1 = E1*(C + ky*ky/kh2 - kx*kx/kh2*(1+D1)); \]
\[ Ux = C/(B*B)*(t2-t1); \]
\[ Uy = -i*C/(B*B)*ky/kh*(E2*(1 + D2) - E1*(1 + D1)); \]
\[ Uz = -i*C/(B*B)*kx/kh*(E2*(1 + D2) - E1*(1 + D1)); \]

% compute the y-displacements  
\[ Vx = Uy; \]
\[ t2 = E2*(C + kx*kx/kh2 - ky*ky/kh2*(1+D2)); \]
\[ t1 = E1*(C + kx*kx/kh2 - ky*ky/kh2*(1+D2)); \]
\[ Vy = C/(B*B)*(t2-t1); \]
\[ Vz = -i*C/(B*B)*kx/kh*(E2*(1 + D2) - E1*(1 + D1)); \]

% compute the z-displacements  
\[ Wx = Uz; \]
\[ Wy = Vz; \]
\[ Wz = C/(B*B)*(E2*(D + 1 + D2) - E1*(D + 1 + D1)); \]

% add the solutions  
\[ U = Ux + Uy + Uz; \]
\[ V = Vx + Vy + Vz; \]
\[ W = Wx + Wy + Wz; \]

% compute rhox  
\[ \text{RHX} = \text{mu}*(-pi2*kx*kx*U-pi2*ky*ky*U+\text{diff}(U,'z',2)); \]
\[ \text{RHX} = \text{RHX} + \text{lam}*(-pi2*kx*kx*U-pi2*ky*ky*V+i*2*pi*ky*\text{diff}(W,'z',1)); \]
\[ \text{simplify(\text{RHX})} \]
\[ \text{ans: RHX} = 0 \]

% compute rhy  
\[ \text{RHY} = \text{mu}*(-pi2*kx*kx*V-pi2*ky*ky*V+\text{diff}(V,'z',2)); \]
\[ \text{RHY} = \text{RHY} + \text{lam}*(-pi2*kx*ky*U-pi2*ky*ky*V+i*2*pi*ky*\text{diff}(W,'z',1)); \]
\[ \text{simplify(\text{RHY})} \]
As with the case of the point source, we also find non-zero terms in the stress tensor corresponding to the normal components:

\[
\begin{bmatrix}
\tau_{x2} \\
\tau_{y2} \\
\tau_{z2}
\end{bmatrix}
= \begin{bmatrix}
\tau_1 \\
\tau_2 \\
\tau_3
\end{bmatrix}
= \frac{1}{\beta}
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
\tau_{3x} & \tau_{3y} & \tau_{3z}
\end{bmatrix}
\begin{bmatrix}
F_x \\
F_y \\
F_z
\end{bmatrix}
\]

(42)

where

\[
\tau_{3x} = -i k_x \left\{ e^{\beta z} \left( \alpha \beta d_2 - \frac{\lambda}{(\lambda + 2\mu)} \right) - e^{\beta i} \left( \alpha \beta d_1 - \frac{\lambda}{(\lambda + 2\mu)} \right) \right\}
\]

\[
\tau_{3y} = -i k_y \left\{ e^{\beta z} \left( \alpha \beta d_2 - \frac{\lambda}{(\lambda + 2\mu)} \right) - e^{\beta i} \left( \alpha \beta d_1 - \frac{\lambda}{(\lambda + 2\mu)} \right) \right\}
\]

\[
\tau_{3z} = \left\{ e^{\beta z} \left( \alpha \beta d_2 - \frac{\mu}{(\lambda + 2\mu)} - 2\alpha \right) - e^{\beta i} \left( \alpha \beta d_1 - \frac{\mu}{(\lambda + 2\mu)} - 2\alpha \right) \right\}
\]

The Boussinesq solutions remain the same because they are independent of depth. Therefore, we use the same set of solutions, but simply make a substitution for \(\tau_3(k)\) corresponding to our new \(\tau_3(k)\) solution. Again we would like to isolate the factors containing \(e^{\beta \xi}\) and \(-\beta \xi e^{\beta \xi}\).

\[
\begin{bmatrix}
\tau_{3x} \\
\tau_{3y} \\
\tau_{3z}
\end{bmatrix}
= T(k, d_2) - T(k, d_1)
\]

\[
\begin{bmatrix}
T_x \\
T_y \\
T_z
\end{bmatrix}
= \begin{bmatrix}
i \frac{\lambda}{(\lambda + 2\mu)} k_x \cdot \frac{1}{k} \cdot \left\{ e^{\beta \xi} \right\} + i \alpha k_x \cdot \frac{1}{k} \cdot \left\{ e^{-\beta \xi} \right\} \\
i \frac{\lambda}{(\lambda + 2\mu)} k_y \cdot \frac{1}{k} \cdot \left\{ e^{\beta \xi} \right\} + i \alpha k_y \cdot \frac{1}{k} \cdot \left\{ e^{-\beta \xi} \right\} \\
i \frac{\lambda}{(\lambda + 2\mu)} k_z \cdot \frac{1}{k} \cdot \left\{ e^{\beta \xi} \right\} - \alpha \cdot \frac{1}{k} \cdot \left\{ e^{-\beta \xi} \right\}
\end{bmatrix}
\]

(44)

Our final integrated solution is then:

\[
\begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}
= \begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{source}} + \begin{bmatrix}
U(k) \\
V(k) \\
W(k)
\end{bmatrix}^{\text{image}} + \begin{bmatrix}
U^B(k) \\
V^B(k) \\
W^B(k)
\end{bmatrix}
\]

(45)

http://topex.ucsd.edu/body_force/
The following Matlab code verifies this integrated solution, checking both the boundary conditions and differential equation:

```
% matlab routine to compute stresses (tau(z)) from the body-force solution for both source and the image and Boussinesq correction
% pi = sym('pi');
pi2= 4*pi*pi;
% wavenumbers and observation depth z
kxx = sym('kxx');
kyy = sym('kyy');
kh2 = kxx*kxx+kyy*kyy;
kh = sqrt(kh2);
% source depth and elastic constants
syms d1 d2
z = sym('z');
lam = sym('lam');
mu = sym('mu');
alpha = (lam + mu)/(lam + 2*mu);
B = 2*pi*kh;
beta = B;
C = (lam + mu)/(4*mu*(lam+2*mu));
D = (lam+3*mu)/(lam + mu);
% set constants for computing tau solution
ed1 = exp(beta.*d1);
Bd1 = beta.*d1;
ed2 = exp(beta.*d2);
Bd2 = beta.*d2;
M2U = mu/(lam+2*mu);
L2U = lam/(lam+2*mu);
A1 = -ed1./beta ;
A2 = -ed2./beta ;
% generate components of stress tensor
t1x = 0*A1;
t1y = 0*A1;
t1z = 0*A1;
t2x = 0*A2;
t2y = 0*A2;
t2z = 0*A2;
t3x = A2.*(i*kxx./kh.*(Bd2*alpha-L2U))-A1.*(i*kxx./kh.*(Bd1.*alpha-L2U));
t3y = A2.*(i*kyy./kh.*(Bd2*alpha-L2U))-A1.*(i*kyy./kh.*(Bd1.*alpha-L2U));
t3z = A2.*(2*alpha+M2U-alpha*Bd2)-A1.*(2*alpha+M2U-alpha*Bd1);
% set constants for UVW source and image solutions for upper and lower boundary (2-1)
es1 = exp(-(z-d1)*B);
e11 = exp((z+d1)*B);
Ms1 = es1./(B.*B);
Mi1 = e11./(B.*B);
es2 = exp(-(z-d2)*B);
e12 = exp((z+d2)*B);
Ms2 = es2./(B.*B);
Mi2 = e12./(B.*B);
```

http://topex.ucsd.edu/body_force/
% compute U - displacement
sUx1 = Ms1.*C.*((0 + kyy.*kyy./kh2 - kxx.*kxx./kh2.)*(1 + (z-d1)*B));
sWy1 = -Ms1.*C.*sx =Ms1.*C.*kyy./kh2.)*2 + (z-d1)*B);
sUz1 = -i*Ms1.*C.*kxx./kh.*(1 + (z-d1)*B);
iUx1 = Mi1.*C.*((0 + kyy.*kyy./kh2 - kxx.*kxx./kh2.)*(1 + (z-d1)*B));
iWy1 = -Mi1.*C.*kxx./kh2.)*2 + (z-d1)*B));
iUz1 = -(i*Mi1.*C.*kxx./kh.*(1 - (z+d1)*B)); %negative
Ux1 = sUx1 + iUx1;
Uy1 = sWy1 + iWy1;
Uz1 = sUz1 + iUz1;
U1 = Ux1 + Uy1 + Uz1;

sUx2 = Ms2.*C.*((0 + kyy.*kyy./kh2 - kxx.*kxx./kh2.)*(1 + (z-d2)*B));
sWy2 = -Ms2.*C.*sx =Ms2.*C.*kyy./kh2.)*2 + (z-d2)*B);
sUz2 = -i*Ms2.*C.*kxx./kh.*(1 + (z-d2)*B);
iUx2 = Mi2.*C.*((0 + kyy.*kyy./kh2 - kxx.*kxx./kh2.)*(1 - (z+d2)*B));
iWy2 = -Mi2.*C.*kxx./kh2.)*2 + (z-d2)*B));
iUz2 = -(i*Mi2.*C.*kxx./kh.*(1 - (z+d2)*B)); %negative
Ux2 = sUx2 + iUx2;
Uy2 = sWy2 + iWy2;
Uz2 = sUz2 + iUz2;
U2 = Ux2 + Uy2 + Uz2;

% compute V - displacement
sVx1 = Ms1.*C.*((0 + kyy.*kyy./kh2 - kyy.*kyy./kh2.)*(1 + (z-d1)*B));
sVz1 = -i*Ms1.*C.*kyy./kh.*(1 + (z-d1)*B);
iVx1 = Mi1.*C.*((0 + kyy.*kyy./kh2 - kyy.*kyy./kh2.)*(1 - (z+d1)*B));
iVz1 = -(i*Mi1.*C.*kyy./kh.*(1 - (z+d1)*B)); %negative
Vx1 = sVx1 + iVx1;
Vy1 = sVy1 + iVy1;
Vz1 = sVz1 + iVz1;
V1 = Vx1 + Vy1 + Vz1;

sVx2 = Ms2.*C.*((0 + kyy.*kyy./kh2 - kyy.*kyy./kh2.)*(1 + (z-d2)*B));
sVz2 = -i*Ms2.*C.*kyy./kh.*(1 + (z-d2)*B);
iVx2 = Mi2.*C.*((0 + kyy.*kyy./kh2 - kyy.*kyy./kh2.)*(1 - (z+d2)*B));
iVz2 = -(i*Mi2.*C.*kyy./kh.*(1 - (z+d2)*B)); %negative
Vx2 = sVx2 + iVx2;
Vy2 = sVy2 + iVy2;
Vz2 = sVz2 + iVz2;
V2 = Vx2 + Vy2 + Vz2;

% compute W - displacement
sWx1 = -i*Ms1.*C.*kxx./kh.*(1 + (z-d1)*B);
sWy1 = -i*Ms1.*C.*kxy./kh.*(1 + (z-d1)*B);
sWz1 = Ms1.*C.*( D + 1 + (z-d1)*B);
iWx1 = (i*Mi1.*C.*kxx./kh.*(1 - (z+d1)*B));
iWy1 = (i*Mi1.*C.*kxy./kh.*(1 - (z+d1)*B));
iWz1 = -(Mi1.*C.*( D + 1 - (z+d1)*B)); %negative
Wx1 = sWx1 + iWx1;
Wy1 = sWy1 + iWy1;
Wz1 = sWz1 + iWz1;
W1 = Wx1 + Wy1 + Wz1;

sWx2 = -i*Ms2.*C.*kxx./kh.*(1 + (z-d2)*B);
sWy2 = -i*Ms2.*C.*kxy./kh.*(1 + (z-d2)*B);
sWz2 = Ms2.*C.*( D + 1 + (z-d2)*B);
iWx2 = (i*Mi2.*C.*kxx./kh.*(1 - (z+d2)*B));
iWy2 = (i*Mi2.*C.*kxy./kh.*(1 - (z+d2)*B));
iWz2 = -(Mi2.*C.*( D + 1 - (z+d2)*B)); %negative
Wx2 = sWx2 + iWx2;
Wy2 = sWy2 + iWy2;
Wz2 = sWz2 + iWz2;
W2 = Wx2 + Wy2 + Wz2;
% subtract lower boundary from upper boundary (d2-d1)
Uint = U2 - U1;
Vint = V2 - V1;
Wint = W2 - W1;

% set up boussinesq solutions
ab = (1/alpha + beta.*z);
e = exp(beta.*z);

% generate components of U,V,W
Ub = -2*pi*i/(2*mu)*kxx.*t3x./(beta.*beta).*(1-ab).*e;
Uy = -2*pi*i/(2*mu)*kyy.*t3y./(beta.*beta).*(1-ab).*e;
Ub = -2*pi*i/(2*mu)*kzy.*t3z./(beta.*beta).*(1-ab).*e;
Vb = -2*pi*i/(2*mu)*kyy.*t3y./(beta.*beta).*(1-ab).*e;
Vb = -2*pi*i/(2*mu)*kzy.*t3z./(beta.*beta).*(1-ab).*e;
Wb = -1/(2*mu)*t3x./(beta).*(1/alpha - beta.*z).*e;
Wb = -1/(2*mu)*t3y./(beta).*(1/alpha - beta.*z).*e;
Wb = -1/(2*mu)*t3z./(beta).*(1/alpha - beta.*z).*e;

% add components:
Ub = Ubx + Uby + Ubz;
Vb = Vbx + Vby + Vbz;
Wb = Wbx + Wby + Wbz;

% add boussinesq solution to source-image solution to check diff. equation
U = Uint + Ub;
V = Vint + Vb;
W = Wint + Wb;

% check differential equation

% compute rnx
RNX = mu*(-pi2*kxx*kxx*U-pi2*kyy*kyy*U+diff(U,'z',2));
RNX = simplify(RNX + (lam+mu)*(-pi2*kxx*kxx*U-pi2*kyy*kyy*V+i*2*pi*kxx*diff(W,'z',1)))
ans: RNX = 0

% compute rny
RNY = mu*(-pi2*kxx*kxx*V-pi2*kyy*kyy*V+diff(V,'z',2));
RNY = simplify(RNY + (lam+mu)*(-pi2*kxx*kxx*U-pi2*kyy*kyy*V+i*2*pi*kyy*diff(W,'z',1)))
ans: RNY = 0

% compute rnz
RHZ = mu*(-pi2*kxx*kxx*W-pi2*kyy*kyy*W+diff(W,'z',2));
RHZ = simplify(RHZ + (lam+mu)*(-pi2*kxx*kxx*U-pi2*kyy*kyy*V+i*2*pi*kyy*diff(W,'z',1) + diff(W,'z',2)))
ans: RHZ = 0

% add boussinesq components to source-image components (d2-d1)
Ux = Ux2 - Ux1 + Ubx;
Uy = Uy2 - Uy1 + Uby;
Uz = Uz2 - Uz1 + Ubz;
Vx = Vx2 - Vx1 + Vbx;
Vy = Vy2 - Vy1 + Vby;
Vz = Vz2 - Vz1 + Vbz;
Wx = Wx2 - Wx1 + Wbx;
Wy = Wy2 - Wy1 + Wby;
Wz = Wz2 - Wz1 + Wbz;

% now compute the derivative of the displacements with respect to z
Uxz = diff(Ux,'z');
Uyz = diff(Uy,'z');
Uzz = diff(Uz,'z');
Vxz = diff(Vx,'z');
Vyz = diff(Vy,'z');
Vzz = diff(Vz,'z');
Wxz = diff(Wx,'z');
Wyz = diff(Wy,'z');
Wzz = diff(Wz,'z');
We also check this case against an analytic solution. The surface displacement due to a two-dimensional fault that is freely slipping between depths of \(d_1\) and \(d_2\) [Weertman, 1964] is:

\[
v(x) = \frac{A}{\pi \mu} \left( \tan^{-1} \frac{x}{d_2} - \tan^{-1} \frac{x}{d_1} \right)
\]

where \(A\) is slip magnitude. For comparison with this solution we must divide the Fourier solution by \(\Delta x\) which corresponds to the cell-size in the x-direction. In Figure 2., we have imbedded an infinitely long fault in the y-dimension in a 1-km spaced grid (\(\Delta x = 1\)). We have assigned an upper locking depth of 5 km (\(d_3\)) to the fault plane and have extended the lower depth to infinity (\(d_1\)). The comparison between the analytic solution (above) and an fault-perpendicular profile of our Fourier model is shown in Figure 3. Numerical accuracy between the analytic and Fourier solutions on the order of \(10^{-3}\) is shown in Figure 4.
(x) **Develop an equivalent body force for a general fault model**

Suppose we have a fault plane of finite extent. Assume for now that the fault is vertical and extends from $d_1$ to $d_2$ as in the previous model. Let $\mathbf{F}(\mathbf{x})$ be a 3-dimensional body force defined in the fault plane and $\mathbf{n}$ is the normal vector to the plane. Following the displacement formulation of Okada [1985], the three components of $\mathbf{F}$ are:

- $F_1$ - strike-slip displacement (horizontal)
- $F_2$ - dip-slip displacement (vertical)
- $F_3$ - opening (horizontal, dike)

Now the three components of the body force are:

$$
F_x = F_1 \cos(\theta) - F_3 \sin(\theta) \\
F_y = F_1 \sin(\theta) + F_3 \cos(\theta) \\
F_z = F_2
$$

where $\theta$ is the angle between the fault trace and the $x$-axis. To calculate the equivalent body force vector, we need to take the derivative of the body force vector in the direction that is normal $\mathbf{n}$ to the fault.

$$
\rho(\mathbf{x}) = \nabla \mathbf{F}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x})
$$

Note that $\nabla \mathbf{F}$ is a stress tensor and now has units of force/area. The total displacement vector is

$$
\mathbf{V}(\mathbf{k}) = \mathbf{A}(\mathbf{k}) \Im \left[ \nabla \mathbf{F}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) \right]
$$

The $\mathbf{A}$-tensor is the solution provided above relating the vector displacement to the vector body force distribution. As in the case of the Okada models [Feigl et al., 1999], one defines a series of fault segments having an orientation, length, and three components of slip. Because our model is a stress formulation rather than a displacement formulation, the three components of slip are replaced by three body forces. The following formula provides a mapping in the case of an infinite fault plane of uniform slip.

$$
\mathbf{F} = \mu \Delta \mathbf{V}
$$

When the lower edge of the fault is extended to infinite depth as in the case of the SAF system model, a Fourier cosine transform is used in the across-fault direction to maintain the far-field velocity $\mathbf{V}$ step across the plate boundary. To avoid Fourier artifacts where the fault enters the bottom of the grid and leaves the top of the grid, the fault is extended beyond the top of the model and angled to match the intersection point at the bottom.
The formulation above computes a numerical derivative of the force vector to create a stress tensor using a grid spacing of $\Delta x$. This results in the following, more familiar relationship.

$$\tau = \frac{F}{\Delta x} = \mu \frac{\Delta V}{\Delta x}$$

(51)
xi) **Force couples on a regular grid**

A dislocation on a fault plane is commonly represented by body force couples. In the case of a horizontal strike-slip fault, a double-couple should be used to ensure local balance moment in the horizontal plane [Burridge and Knopoff, 1964]. Here we describe the algorithm for generating single- and double-couple body forces for a segmented fault trace mapped onto a regular grid.

Consider a grid with cell spacing $\Delta x$. The final displacement and stress model cannot resolve features smaller than the cell spacing, thus we approximate a fault segment with a finite length $L$ and a finite thickness $\sigma \geq \Delta x$ as follows

$$ f(x, y) = g(x) h(y) $$  \hspace{1cm} (52)

where the across-fault function $h(y)$ is a Gaussian function

$$ h(y) = \frac{1}{\sigma \sqrt{2\pi}} \exp \left( \frac{y^2}{2\sigma^2} \right). $$  \hspace{1cm} (53)

We represent a curved fault trace as a large number of straight overlapping segments of the form

$$ g(x) = \begin{cases} 
1 - \cos \frac{\pi (x + 2\Delta x)}{2\Delta x} & -2\Delta x < x < 2\Delta x \\
1 & 2\Delta x < x < L - 2\Delta x \\
1 - \cos \frac{\pi (L + 2\Delta x - x)}{2\Delta x} & L - 2\Delta x < x < L + 2\Delta x 
\end{cases} $$  \hspace{1cm} (54)

where $x$ is the distance from the start of the segment and $L$ is the segment length. The segments are arranged end-to-end so the sum of the overlapping cosine functions equals one. The spatial variations in the force-couple are constructed by taking the derivatives of the fault function. The primary couple is parallel to the fault ($x$-direction) and corresponds to the fault-normal derivative

$$ f_i(x, y) = g(x) \frac{\partial h}{\partial y} $$  \hspace{1cm} (55)

where

$$ \frac{\partial h}{\partial y} = \frac{-y}{\sigma^{3/2} \sqrt{2\pi}} \exp \left( \frac{y^2}{2\sigma^2} \right). $$  \hspace{1cm} (56)
The secondary force couple is perpendicular to the fault (y-direction) and corresponds to the fault-parallel derivative

\[ f_2(x, y) = \frac{\partial g}{\partial x} h(y) \]  

(57)

where

\[
\frac{\partial g}{\partial x} = \begin{cases} 
\frac{\pi}{8\Delta x} \sin \frac{\pi(x + 2\Delta x)}{4\Delta x} & -2\Delta x < x < 2\Delta x \\
1 & 2\Delta x < x < L - 2\Delta x \\
\frac{-\pi}{8\Delta x} \sin \frac{\pi(L + 2\Delta x - x)}{4\Delta x} & L - 2\Delta x < x < L + 2\Delta x
\end{cases}.
\]

(58)

A rotation matrix is used to rotate the force couple functions to the proper orientation

\[
\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\]

(59)

where \( \theta \) is the angle between the x-axis and the fault trace (\( \theta > 0 \) represents counterclockwise rotation).

Three modes of displacement can be applied on each fault segment. \( F_1 \) is strike-slip, \( F_2 \) is dip-slip, and \( F_3 \) is opening of the fault. Once the primary and secondary force couple functions are computed and rotated into the fault direction, they are multiplied by the strength of the dislocation to form three grids corresponding to the \( F_1 \), \( F_2 \), and \( F_3 \) modes. These three force components must then be rotated into the Cartesian frame \( f_x, f_y \), and \( f_z \) using the following formulas.

\[
\begin{pmatrix} f_{1}(x, y) \\ f_{2}(x, y) \\ f_{3}(x, y) \end{pmatrix} = f_1(x, y) \begin{pmatrix} -\cos \theta & \sin \theta \\ \sin \theta & \cos \theta \\ 1 \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \\ F_3 \end{pmatrix}. 
\]

(60)

Balancing of the moment due to the horizontal strike-slip force couple \( F_1 \) requires a second force component given by

\[
\begin{pmatrix} f_{2}^2(x, y) \\ f_{3}^2(x, y) \end{pmatrix} = f_2(x, y) \begin{pmatrix} F_1 \sin \theta \\ F_1 \cos \theta \end{pmatrix}. 
\]

(61)
Note that this force couple only applies to the end of each fault segment and the forces largely cancel where fault segments abut as described in Burridge and Knopoff [1964]. The moment generated by the vertical dip-slip $F_2$ and the opening $F_3$ of the fault will produce topography that will balance the moment under the restoring force of gravity.
xii) Fortran source code for elastic half-space model

The following Fortran code fftfault.f (and subroutines element.f, fault.f, boussinesq.f) are used to compute displacement and stress from a set of input fault parameters and write these solutions to appropriate grd files. The user sets the fault dimensions, location, and grid spacing from the command line:

```
%  fftfault D1 D2 Zobs elements.xyF xd.grd yd.grd zd.grd
%
D1 = lower bound on fault depth (negative)
D2 = upper bound on fault depth (negative)
Zobs = observation plane depth (negative)
elements.xyF = position of fault elements and force magnitudes (x1 y1 x2 y2 F1 F2 F3)
xd.grd = name of x-displacement grd file (or txx-stress grd file)
yd.grd = name of y-displacement grd file (or tyy-stress grd file)
zd.grd = name of z-displacement grd file (or txy-stress grd file)
```

```
c program fftfault
  c program to calculate the surface displacement or stress
c due to planar vertical fault element(s) between depth D1 and D2.
c the size of the area is changed by modifying ni and nj in the
c parameter statement below.
c
***main program ****************************
  implicit complex (c)
  real kx,ky
  real*8 rln0,rlt0,ddx,ddy,rland,rdum
  parameter(ni=2048,nj=2048,nwork=32768,nj2=nj/2+1,ni2=ni/2+1)
  character*80 felement,fxdisp,fydisp,fzdisp
  character*80 cd1,cd2,cz,cstr
  c common/elastic/rlam,rmu,rc,rd,alph,pi
  c dimension fx(nj,ni),fy(nj,ni),fz(nj,ni)
  dimension u(nj,ni),v(nj,ni),w(nj,ni)
  complex fkx(nj2,ni),fky(nj2,ni),fkz(nj2,ni)
  complex uk(nj2,ni),vk(nj2,ni),wk(nj2,ni),Tk
  dimension n(2)
  complex work(nwork)
  equivalence (fx(1,1),fkx(1,1))
  equivalence (fy(1,1),fky(1,1))
  equivalence (fz(1,1),fkz(1,1))
  equivalence (u(1,1),uk(1,1))
  equivalence (v(1,1),vk(1,1))
  equivalence (w(1,1),wk(1,1))
  c pi=acos(-1.)
```

http://topex.ucsd.edu/body_force/
c zero the arrays fxdisp,fydisp,fzdisp
c
do 30 i=1,ni
do 30 j=1,nj
fx(j,i)=0.
fy(j,i)=0.
fz(j,i)=0.
30 continue
c set the dimensions for fourt
c
n(1)=nj
n(2)=ni
30 continue
c get values from command line
c
narg = iargc()
if(narg.lt.8) then
write(*,'(a)')'  
write(*,'(a)')'  
& 'Usage: fftfault D1 D2 Zobs ele.dat istr xd.grd yd.grd zd.grd'
write(*,'(a)')'  
write(*,'(a)')'  
& ' D1, D2 - depth to bottom and top of fault (neg) '  
write(*,'(a)')'  
& ' Zobs - depth of observation plane '  
write(*,'(a)')'  
& ' ele.dat - file of: x1,x2,y1,y2,F1,F2,F3'  
write(*,'(a)')'  
& '  istress - (0)-disp. U,V,W or (1)-stress Txx,Tyy,Txy'  
write(*,'(a)')'  
& '  x,y,z.grd - output files of disp. or stress '  
write(*,'(a)')'  
stop
else
  call getarg(1,cd1)
  call getarg(2,cd2)
  call getarg(3,cz)
  call getarg(5,cstr)
  read(cd1,*)d1
  read(cd2,*)d2
  read(cz,*)zobs
  read(cstr,*)istr
  if(d2.gt.0..or.d1.ge.d2.or.zobs.lt.d2.or.zobs.gt.0) then
   write(*,'(a)')' depths should be negative and d1 < d2 < zobs'
   stop
  endif
  call getarg(4,felement)
  call getarg(6,fxdisp)
  nc=index(fxdisp,')'
fdisp(nc:nc)='\0'
call getarg(7,fydisp)
  nc=index(fydisp,')'
fdisp(nc:nc)='\0'
call getarg(8,fzdisp)
  nc=index(fzdisp,')'
fzdisp(nc:nc)='\0'
endif
c
open the input file and load the force arrays
open(unit=5,file=felement,status='old')
read(5,*)cd1
read the source constants
read(5,*) dx,sig,rlam,rmu,ideep
width=dx*nj
height=dx*ni
alph=(rlam+rmu)/(rlam+2*rmu)
rc=alph/(4.*rmu)
rd=(rlam+3*rmu)/(rlam+rmu)
sig=sig/dx
if(sig.lt.1) sig=1
read the elements
50 read(5,*,end=999)x1,x2,y1,y2,F1,F2,F3
x1=x1/dx
x2=x2/dx
invert the y-positions with i-index
y1=ni-y1/dx
y2=ni-y2/dx
figure out the part of the array to add the new forces
ipd=3*sig+2
jx0=min(x1,x2)-ipd
jxf=max(x1,x2)+.5+ipd
iy0=min(y1,y2)-ipd
iyf=max(y1,y2)+.5+ipd
do 100 iy=i0,iyf
do 200 jx=jx0,jxf
if(jx.le.0.or.jx.gt.nj) go to 200
if(iy.le.0.or.iy.gt.ni) go to 200
x=jx
y=iy
call element(x,y,x1,y1,x2,y2,dx,sig,F1,F2,F3,fxx,fyy,fzz)
fx(jx,iy)=fx(jx,iy)+fxx
fy(jx,iy)=fy(jx,iy)+fyy
fz(jx,iy)=fz(jx,iy)+fzz
remove the x-forces if the fault goes deep
if(ideep.eq.1)fx(jx,iy)=0.
contine
200 continue
100 continue
999 close(unit=5)
if this is a deep source then mirror the source in the x-direction
this will make the zero-slope but finite displacement BC's match
one could also use a cosine transform
if(ideep.eq.1) then
  do 150 iy=1,ni
  do 150 jx=1,nj2
    fx(nj+1-jx,iy)=-fx(jx,iy)
    fy(nj+1-jx,iy)= fy(jx,iy)
    fz(nj+1-jx,iy)= fz(jx,iy)
  continue
  endif

c  take the fourier transform of the forces
  call fourt(fx,n,2,-1,0,work,nwork)
  call fourt(fy,n,2,-1,0,work,nwork)
  call fourt(fz,n,2,-1,0,work,nwork)

c  construct the fault solution

do 255 i=1,ni
  ky=-(i-1)/height
  if(i.ge.ni2) ky= (ni-i+1)/height
  do 255 j=1,nj2
    kx=(j-1)/width
    call fault(kx,ky,d1,d2,zobs,cux,cuy,cuz,
     + cvx,cvy,cvz,
     + cwx,cwy,cwz,
     + c3x,c3y,c3z)
    uk(j,i)=(fkx(j,i)*cux+fky(j,i)*cuy+fkz(j,i)*cuz)/(ni*nj)
    vk(j,i)=(fkx(j,i)*cvx+fky(j,i)*cvy+fkz(j,i)*cvz)/(ni*nj)
    wk(j,i)=(fkx(j,i)*cwx+fky(j,i)*cwy+fkz(j,i)*cwz)/(ni*nj)
    Tk  =(fkx(j,i)*c3x+fky(j,i)*c3y+fkz(j,i)*c3z)/(ni*nj)
  enddo

do the boussinesq problem

call boussinesq(kx,ky,zobs,cub,cvb,cwb)
c
now either output displacement or stress

uk(j,i)=rmu*(uk(j,i)+Tk*cub)
vk(j,i)=rmu*(vk(j,i)+Tk*cvb)
wk(j,i)=rmu*(wk(j,i)+Tk*cwb)

c  now compute the stress if requested.

if(istr.eq.1) then
  cux=cmplx(0.,2.*pi*kx)*uk(j,i)
  cvy=cmplx(0.,2.*pi*ky)*vk(j,i)
  cuv=cmplx(0.,2.*pi*ky)*uk(j,i)
  cvx=cmplx(0.,2.*pi*kx)*vk(j,i)
  cwz=rlam*(cux+cvy)/(rlam+2.*rmu)
  c
  assume units are km so divide by 1000
  units originally in m, take derivative wrt km, get m/km
  divide by 1000 in order to get m/m
  need to divide by another 1000 because answer in mm
  
  uk(j,i)=.001*.001*((rlam+2.*rmu)*cux+rlam*(cvy+cvwz))
  vk(j,i)=.001*.001*((rlam+2.*rmu)*cvy+rlam*(cux+cvwz))
  wk(j,i)=.001*.001*(rmu*(cuy+cvx))
endif
255  continue
c  do the inverse fft's

256
c  call fourt(u,n,2,1,-1,work,nwork)
call fourt(v,n,2,1,-1,work,nwork)
call fourt(w,n,2,1,-1,work,nwork)
c  write 3 grd files some parameters must be real*8
c
  rlt0=0.
ddx=dx
ddy=dx
  rland=9998.
  rdum=9999.
  if(istp.eq.1) then
    rland=9998.d0*rmu
    rdum=9999.d0*rmu
  endif
  call writegrd(u,nj,ni,rln0,rlt0,ddx,ddy,
+  rland,rdum,fxdisp,fxdisp)
call writegrd(v,nj,ni,rln0,rlt0,ddx,ddy,
+  rland,rdum,fydisp,fydisp)
call writegrd(w,nj,ni,rln0,rlt0,ddx,ddy,
+  rland,rdum,fzdisp,fzdisp)
  stop
end
subroutine element(x,y,x1,y1,x2,y2,dxx,sig,F1,F2,F3,Fx,Fy,Fz)

c create the force vector for this element

c input:

c x, y - computation point (km)

c x1,y1 - start position (km)

c x2,y2 - end position (km)

c dx - grid spacing (km)

c sig - source width in pixels (try 1)

c F1 - strike-slip

c F2 - dip-slip

c F3 - opening mode

c output:

c fxx - x-component of force couple

c fyy - y-component of force couple

c fzz - z-component of force couple

c set the x and y taper lengths

c sigx=sig

c sigy=2

c compute the length and orientation of the fault

c Dx=(x2-x1)

c Dy=(y2-y1)

c L=sqrt(Dx* Dx + Dy* Dy)

c theta=atan2(Dy,Dx)

c st=sin(theta)

c ct=cos(theta)

c rotate the vector into the model space

c x0=x-x1

c y0=y-y1

c xp=x0*ct+y0*st

c yp=-x0*st+y0*ct

c compute the h and dh/dx function

c h=exp(-0.5*(((yp/sigx)**2))/(sqrt(2*pi)*sigx)/(dxx*dxx))

c dh=-yp*exp(-0.5*(((yp/sigx)**2))/(sqrt(2*pi)*sigx**3)/(dxx*dxx))

c compute the g function

c if(xp .gt. -sigy .and. xp .lt. sigy) then

c g=0.5*(1-cos((pi*(xp+sigy))/(2*sigy))

c else if(xp .ge. sigy .and. xp .le. L-sigy) then

c g=1.

c else if(xp .gt. L-sigy .and. xp .lt. L+sigy) then

c g=0.5*(1-cos((pi*(L+sigy-xp))/(2*sigy))

c else

c g=0.

c endif
subroutine fault(kx,ky,d1,d2,z,cux,cuy,cuz,
+     cvx, cvy, cvz,
+     cwx, cwy, cwz,
+     c3x, c3y, c3z)

c  input
  kx - x wavenumber         (1/km)
  ky - y wavenumber         (1/km)
  d1 - bottom of fault d1 < d2 < 0. (km)
  d2 - top of fault         (km)
  z   - observation plane   (km)

c  output
  cux,cuy,cuz - elements of force/displacement matrix
  cvx, cvy, cvz             (km**2/Pa)
  cwx, cwy, cwz
  c3x, c3y, c3z - non-zero vertical stress at surface z=0

implicit complex (c)
implicit real*4 (k)

c routine to compute the greens function for a fault element

c  common/elastic/rlam,rmu,rc,rd,alph,pi

c if (kx.eq.0.and.ky.eq.0.) then
  cux=cmplx(0.,0.)
  cuy=cmplx(0.,0.)
  cuz=cmplx(0.,0.)
  cvx=cmplx(0.,0.)
  cvy=cmplx(0.,0.)
  cvz=cmplx(0.,0.)
  cwx=cmplx(0.,0.)
  cwy=cmplx(0.,0.)
  cwz=cmplx(0.,0.)
  c3x=cmplx(0.,0.)
  c3y=cmplx(0.,0.)
  c3z=cmplx(0.,0.)
else
  kh2=kx*kx+ky*ky
  kh=sqrt(kh2)
  B=2*pi*kh
  B2=B*B
  Bs1=B*(z-d1)
  Bs2=B*(z-d2)
  Bi1=B*(z+d1)
  Bi2=B*(z+d2)
  Fx=g*(dh*(F1*ct+F3*st))
  Fy=g*(dh*(F1*st-F3*ct))
  Fz=g*(dh*F2)
  return
end
subroutine boussinesq(kx, ky, z, cub, cvb, cwb)

c input
kx - x wavenumber (1/km)
y - y wavenumber (1/km)
z - observation plane (km)
c
output
b - x-displacement (km/Pa)
cv - y-displacement
cw - z-displacement
c
implicit complex (c)
implicit real*4 (k)
c
routine to compute the greens function for a fault element
c
common/elastic/rlam, rmu, rc, rd, alph, pi
c
if (kx.eq.0. and ky.eq.0.) then
   cub=complex(0., 0.)
cv=complex(0., 0.)
cw=complex(0., 0.)
else
   kh2=kx*kx+ky*ky
   kh=sqrt(kh2)
   B=2*pi*kh
   B2=B*B
   if(arg.lt.-50.) then
      eb=0.
   else
      eb=exp(arg)
   endif
   arg1=(1.-1./alph-B*z)/(2.*rmu)
   arg2=(1./alph-B*z)/(2.*rmu)
   cub=complex(0., -2.*pi*kx*arg1*eb/B)
   cv=complex(0., -2.*pi*ky*arg1*eb/B)
   cwb=complex(-arg2*eb/B)
endif
return
end