n = la normal a la superficie.

Así, el campo de onda $u_i(\mathbf{x})$ puede ser calculado en cualquier parte dentro del volumen V, una vez que la fuerza perturbadora $f_n(\mathbf{x}')$ en el interior del volumen, el desplazamiento $u_n(\mathbf{x}')$ y la tracción asociada $n_j c_{njkl} \partial'_k u_l(\mathbf{x}')$ en la superficie envolvente son conocidas.

ANEXO A2: DEARROLLO ALGEBRÁICO DE TAN β (Figura 11)

De la trigonometría:

$Sen\beta = \frac{sen\beta}{cos\beta} cos\beta$,

$Sen\beta = tan\beta \frac{1}{sec\beta}$

$Sen\beta = \frac{tan\beta}{sec\beta}$

Como

$1 + \ tan^2 \propto = sec^2 \propto$,

Entonces:

$Sen\beta = \frac{tan\beta}{\sqrt{1+tan^2\beta}}$,

de la ley de Snell

$\frac{Sen\beta}{Sen\propto} = \frac{V_S}{V_p}$,

se obtiene que:

$Sen\beta = \frac{V_S}{V_p} Sen \propto$,

$\frac{tan\beta}{\sqrt{1+tan^2\beta}} = \frac{V_S}{V_p} Sen \propto$,

$tan\beta \ = \frac{V_S}{V_p}(1 + tan^2\beta)^{1/2} Sen \propto$

$$= \frac{V_S}{V_p}\left(\frac{1}{cos^2\beta}\right)^{1/2} sen \propto$$

$$= \frac{V_s}{V_p}\frac{1}{(1-sen^2\beta)^{1/2}}sen \propto$$

$$= \frac{V_s}{V_p}\frac{sen\propto}{(1-sen^2\beta)^{1/2}}$$

$$= \frac{V_s}{V_p}\frac{sen\propto}{\left[1-\left(\frac{V_s}{V_p}sen\propto\right)^2\right]^{1/2}}$$

$$= \frac{V_s}{V_p}\frac{sen\propto}{\left[1-\left(\frac{V_s}{V_p}\right)^2 sen^2\propto\right]^{1/2}}$$

$$= \frac{V_s}{V_p}\frac{sen\propto}{\left[sen^2\propto+ cos^2\propto - \left(\frac{V_s}{V_p}\right)^2 sen^2\propto\right]^{1/2}}$$

$$= \frac{V_s}{V_p}\frac{sen \propto}{cos^2 \propto^{1/2}\left[\frac{sen^2 \propto^2}{cos^2 \propto} + 1 - \left(\frac{V_s}{V_p}\right)^2 \frac{sen^2 \propto}{cos^2 \propto}\right]^{1/2}}$$

$$tan\beta = \frac{V_s}{V_p}\frac{sen\propto/cos\propto}{\left[tan^2\propto+ 1 - \left(\frac{V_s}{V_p}\right)^2 tan^2\propto\right]^{1/2}} = \frac{V_s}{V_p}\frac{tan\propto}{\left[1+ \left(\frac{V_s}{V_p}\right)^2 tan^2\propto\right]^{1/2}}$$

$$= \frac{tan \propto}{\left(\frac{V_p}{V_s}\right)\left\{1+\left[1 - \left(\frac{V_s}{V_p}\right)^2 tan^2 \propto\right]\right\}^{1/2}}$$

$$= \frac{tan \propto}{\left(\frac{V_p}{V_s}\right)^2\left\{1+\left[1 - \left(\frac{V_s}{V_p}\right)^2\right] tan^2 \propto\right\}^{1/2}}$$

$$tan\beta = \frac{tan\propto}{\left\{\left(\frac{V_p}{V_s}\right)^2+\left(\left(\frac{V_p}{V_s}\right)^2\left(\frac{V_s}{V_p}\right)^2-1\right)tan^2\propto\right\}^{1/2}})$$

ANEXO A3, DESARROLLO ALGEBRÁICO DE D (Figura 11)

Elevando al cuadrado ambos lados, se tiene:

$$\left(\frac{2D-x}{2D+x}\right)^2 = \frac{1}{\left\{\left(\frac{V_p}{V_s}\right)^2 + \left[\left(\frac{V_p}{V_s}\right)^2 - 1\right]\frac{\left(D+\frac{x}{2}\right)^2}{z^2}\right\}},$$

$$\frac{(2D-x)^2}{(2D+x)^2}\left\{\left(\frac{V_p}{V_s}\right)^2 + \left[\left(\frac{V_p}{V_s}\right)^2 - 1\right]\frac{\left(D+\frac{x}{2}\right)^2}{z^2}\right\} = 1,$$

$$\frac{(2D-x)^2}{(2D+x)^2}\left\{\left(\frac{V_p}{V_s}\right)^2 + \frac{\left(\frac{V_p}{V_s}\right)^2}{4z^2}(2D+x)^2 - \frac{(2D+x)^2}{4z^2}\right\} = 1,$$

$$\frac{(2D-x)^2}{(2D+x)^2}\left(\frac{V_p}{V_s}\right)^2 + \frac{\left(\frac{V_p}{V_s}\right)^2}{4z^2}(2D+x)^2\frac{(2D-x)^2}{(2D+x)^2} - \frac{(2D-x)^2(2D+x)^2}{4z^2(2D+x)^2} = 1,$$

$$\frac{(2D-x)^2}{(2D+x)^2}\left(\frac{V_p}{V_s}\right)^2 + \frac{\left(\frac{V_p}{V_s}\right)^2}{4z^2}(2D+x)^2 - \frac{(2D-x)^2}{4z^2} = 1,$$

$$\frac{(2D-x)^2\left(V_p/V_s\right)^2 4z^2 + \left(V_p/V_s\right)^2(2D+x)^2(2D-x)^2 - (2D-x)^2(2D+x)^2}{4z^2(2D+x)^2} = 1,$$

$$(2D-x)^2\left(V_p/V_s\right)^2 4z^2 + \left(V_p/V_s\right)^2(2D+x)^2(2D-x)^2 - (2D-x)^2(2D+x)^2 = 4z(2D+x)^2$$

$$(2D-x)^2\left(V_p/V_s\right)^2 4z^2 - 4z^2(2D+x)^2 + (4D^2-x^2)^2\left(\left(V_p/V_s\right)^2 - 1\right) = 0,$$

$$(4D-4Dx+x^2)\left(V_p/V_s\right)^2 4z^2 - 4z^2(4D-4Dx+x^2) + (16D^4 - 8D^2x^2 + x^4)\left[\left(V_p/V_s\right)^2 - 1\right] = 0,$$

$$16D^4\left[\left(V_p/V_s\right)^2-1\right]+D^2\left\{4\,.\,4z^2\left(V_p/V_s\right)^2-\;4\,.\,4\,z^2-8x^2\left[\left(V_p/V_s\right)^2-1\right]\right\}+D\left\{-D4X\,.\,4z^2\left(V_p/V_s\right)^2-4z^2\,.\,4x\right\}\;+\;\left\{x^2\left(V_p/V_s\right)^2 4z^2-4z^2x^2+\;x^4\left[\left(V_p/V_s\right)^2-\;1\right]\right\}=\;0\,,$$

$$16D^4\left[\left(V_p/V_s\right)^2-1\right]+D^2\left\{16z^2\left[\left(V_p/V_s\right)^2-1\right]-8x^2\left[\left(V_p/V_s\right)^2-1\right]\right\}-D\left\{16z^2x\left[\left(V_p/V_s\right)^2+1\right]\right\}+\left\{4z^2x^2\left[\left(V_p/V_s\right)^2-\;1\right]+x^4\left[\left(V_p/V_s\right)^2-1\right]\right\}=\;0,$$

$$16D^4\left[\left(V_p/V_s\right)^2-1\right]+D^2\left\{\left[16z^2-8x^2\right]\left[\left(V_p/V_s\right)^2-1\right]\right\}-D\left\{16z^2x\left[\left(V_p/V_s\right)^2+1\right]\right\}+\left\{4x^2z^2+x^4\left[\left(V_p/V_s\right)^2-1\right]\right\}=\;0\,,$$

$$D^4+D^2\frac{\left[16z^2-8x^2\right]\left[\left(V_p/V_s\right)^2-1\right]}{16\left[\left(V_p/V_s\right)^2-1\right]}-\frac{D\;16z^2x\left[\left(V_p/V_s\right)^2+1\right]}{16\left[\left(V_p/V_s\right)^2-1\right]}+\frac{4x^2z^2+x^4\left[\left(V_p/V_s\right)^2-1\right]}{16\left[\left(V_p/V_s\right)^2-1\right]}\;=\;0,$$

$$D^4+D^2\frac{\left[16z^2-8x^2\right]}{16\left[\left(V_p/V_s\right)^2-1\right]}-\frac{D\;z^2x\left[\left(V_p/V_s\right)^2+1\right]}{\left[\left(V_p/V_s\right)^2-1\right]}+\frac{4x^2z^2+x^4}{16}\;=\;0,$$

ANEXO A4, PUNTO DE CONVERSIÓN Vs PROFUNDIDAD

Parámetros	
Vp/Vs	1.7

	CMP (P) = 2000	CMP (P) = 1000	CMP (P) = 250
Profundidad	X= 4000	X= 2000	X= 500
100	3927.30	1927.41	430.09
200	3854.82	1855.80	378.24
300	3782.82	1786.34	350.20
400	3711.59	1720.37	336.38
500	3641.44	1659.20	329.11
600	3572.68	1603.89	324.92
700	3505.67	1555.10	322.32
800	3440.74	1512.96	320.60
900	3378.22	1477.14	319.41
1000	3318.39	1447.02	318.54
1100	3261.52	1421.83	317.90
1200	3207.78	1400.81	317.42
1300	3157.33	1383.23	317.03
1400	3110.20	1368.49	316.73
1500	3066.42	1356.06	316.48
1600	3025.91	1345.52	316.28

1700	2988.58	1336.54	316.12
1800	2954.28	1328.84	315.98
1900	2922.82	1322.20	315.86
2000	2894.04	1316.43	315.76
2100	2867.72	1311.41	315.67
2200	2843.67	1307.00	315.59
2300	2821.70	1303.12	315.53
2400	2801.62	1299.69	315.47
2500	2783.26	1296.64	315.42
2600	2766.47	1293.91	315.37
2700	2751.08	1291.47	315.33
2800	2736.97	1289.28	315.30
2900	2724.02	1287.30	315.26
3000	2712.11	1285.50	315.23

ANEXO A5, PUNTO DE CONVERSIÓN Y A LA RELACIÓN Vp/Vs.

Z = 2000;			
Vp/Vs	X= 500	X= 1000	X= 2000
1.5	300.7485	605.9536	1246.399
1.6	308.5436	622.142	1283.001
1.7	315.7564	637.0894	1316.434
1.8	322.4495	650.9288	1347.054
1.9	328.6767	663.7758	1375.175
2	334.4846	675.7307	1401.069
2.1	339.914	686.8812	1424.975
2.2	345.0005	697.3041	1447.103
2.3	349.7754	707.067	1467.636
2.4	354.2663	716.2295	1486.733
2.5	358.4977	724.8445	1504.534
2.6	362.4914	732.9589	1521.164
2.7	366.2667	740.6143	1536.732
2.8	369.8411	747.8482	1551.333
2.9	373.2301	754.6941	1565.053
3	376.4477	761.1818	1577.968

ANEXO A6. SECUENCIA DE PROCESO PARA USUARIOS PROMAX.

Pre-proceso:

1. CONVERSIÓN FORMATO SEG-Y A FORMATO ProMAX.

   Conversión de los datos sísmicos del formato demultiplexado SEG-Y al formato de datos del software de procesamiento.

2. ASIGNACIÓN DE GEOMETRIA.

   Asignación de geometría: coordenadas, elevación, de pozos y receptores, ubicación de CMP, distribución de cubrimiento, *offsets*, tiempo de pozo, número de canales, geometría de registro y CDP, a los encabezados de las trazas sísmicas.

3. REVISION DE GEOMETRIA

   Verificación de la correcta asignación de las coordenadas en los encabezados y la correspondencia con las trazas sísmicas, usando los arribos de trazas cercanas al punto de tiro. Se empleó la visualización de los registros acompañados con primeros arribos teóricos, lo que permite detectar errores que se pudiesen presentar tanto en la posición de los pozos como en la elevación de las estaciones receptoras y por ende determinar errores en el tendido. Revisión de FFID'S, número de canales, trazas ruidosas, trazas con polaridad invertida, trazas muertas, longitud de registro.

4. ENMUDECIMIENTO CAPA DE REFRACCION

   Eliminación en las trazas sísmicas de las amplitudes asociadas a las ondas refractadas.

5. ECUALIZACIÓN DE TRAZA.

   Balanceo de amplitudes debido a la pérdida de energía en el acople carga-suelo.

6. RECUPERACIÓN DE AMPLITUDES : TAR

   Balanceo de amplitudes debido a la pérdida de energía en el acople carga-suelo.

   Corrección constante de amplitud : -4 dB/s

   DECONVOLUCION

   Compresión en el tiempo de las ondículas que componen las trazas sísmicas.

   Deconvolución Tipo : Spiking, Ricker fase mínima

   Longitud de Operador : 160 ms

7. BALANCEO ESPECTRAL

Compensación del contenido frecuencial debido a la dispersión del paquete de onda en las reflexiones.

Valores de Frecuencia : 10-16-60-90 Hz

8. CONTROL AUTOMATICO DE GANANCIA (AGC)

Balanceo de las amplitudes en toda la traza sísmica.

Longitud de Operador : 500 ms

Proceso:

9. CALCULO DE LA ESTATICA POR ELEVACION

De la información de geometría, se compensan las variaciones en tiempo de viaje causado por las elevaciones de la fuente y el receptor.

Transforma las trazas como si se hubiesen registrado en una superficie horizontal común o datum.

Datum : 1000 msnm.

Velocidad de Reemplazamiento : 2400 m/s

10. ANALISIS DE VELOCIDAD

Se determina la velocidad rms que permita un óptimo apilamiento de las trazas en el dominio CDP.

Tipo : Constante

Intervalo de análisis : 0.5 km.

Datum : Floating

11. CORRECCIÓN POR SOBRE TIEMPO NORMAL (NMO)

Remueve los efectos en las reflexiones de la separación fuente receptor. Transforma las trazas como si fuesen reflexiones de incidencia normal.

12. APILADO CON ESTÁTICA POR ELEVACIÓN

Se suman las trazas en las familias CDP para mejorar la relación señal a ruido.

13. EDITADO DE PRIMEROS ARRIBOS

Editado de tiempos de primer arribo, se realiza traza por traza.

Offset :2400m.

14. CÁLCULO Y APLICACIÓN DE ESTÁTICA POR REFRACCIÓN

Se construye un modelo de velocidad y profundidad de la capa somera del subsuelo. Se compensa el efecto de la capa de baja velocidad.

Algoritmo : Gauss Seidel.

Datum : 1000 msnm

Velocidad de Reemplazamiento : 2400 m/s

15. ANÁLISIS DE VELOCIDAD

Tipo : porcentaje

Intervalo de análisis : 0.25 km.

Datum : Floating

16. CORRECCIÓN POR SOBRETIEMPO NORMAL

17. APILADO CON ESTÁTICA POR REFRACCIÓN

18. CÁLCULO Y APLICACIÓN DE ESTÁTICA RESIDUAL

Se remueven las pequeñas variaciones remanentes en los tiempos de viaje, causados por la inexacta estática de refracción o el modelo de velocidad de apilado.

Ventana : 500 ms

Trazas de referencia (Smash) : 3

19. CORRECCIÓN POR SOBRETIEMPO NORMAL

20. APILADO CON ESTÁTICA RESIDUAL

Post-Apilado

21. PASO DE VELOCIDADES RMS A DATUM

22. SUAVIZADO DE VELOCIDADES

23. MIGRACIÓN EN TIEMPO

Tipo :Kirchhoff

Porcentaje :100%

Máximo Buzamiento :65 grados

Para visualización

24. DECONVOLUCIÓN F-X

25. FILTRO PASABANDA TIPO : Ormsby bandpass

Dominio : Frecuencia

Valores de Frecuencia : 10-20-60-80 Hz

26. CONTROL AUTOMÁTICO DE GANANCIA (AGC)

Longitud de Operador : 500 ms

ANEXO A7. SECUENCIA DE PROCESO onda PS

1. CONVERSIÓN FORMATO SEG-Y A FORMATO ProMAX.

2. ASIGNACIÓN DE GEOMETRIA.

3. REVISION DE GEOMETRIA.

4. CORRECCIÓN DE POLARIDAD.

   Se invierte la polaridad de las trazas, deflexión primer arribo hacia la izquierda.

5. ENMUDECIMIENTO CAPA DE REFRACCIÓN

6. ECUALIZACIÓN DE TRAZA.

7. RECUPERACIÓN DE AMPLITUDES : TAR

   Corrección constante de amplitud : -4 dB/s.

8. DECONVOLUCIÓN

   Deconvolución Tipo : predictiva

   Longitud de Operador : 160 ms

9. BALANCEO ESPECTRAL

   Valores de Frecuencia : 5-10-50-70 Hz

10. CONTROL AUTOMATICO DE GANANCIA (AGC)

11. APLICACIÓN DE ESTÁTICAS POR ELEVACIÓN

12. CÁLCULO DE LA CORRECCIÓN POR OFFSSET A CADA TRAZA

    Se calcula la corrección de la ubicación del punto de conversión usando las ecuaciones planteadas, asumiendo un valor *Vp/Vs*. Se repite para varios valores *Vp/Vs* en el rango (1.1 a 2.7).

13. APLICACIÓN CORRECCIÓN COORDENADAS DE CDP A CADA TRAZA.

    Se modifican las coordenadas de BIN-CDP asociada a cada traza basado en la corrección CCP.

14. REASIGNACIÓN DE BIN-CDP A CADA TRAZA

Se reenumera el valor asociado de CDP en los encabezados de cada traza.

15. REAGRUPAMIENTO DE LAS TRAZAS POR BIN-CDP

    Se agrupan por familias de los nuevos valores de CDP.

16. CONSTRUCCIÓN DEL CAMPO DE VELOCIDAD INICIAL P a S

    Se toma el campo de velocidad de apilado de onda P y se calcula un campo de velocidad de arranque para la onda convertida.

17. ANÁLISIS DE VELOCIDAD

18. CORRECCIÓN POR SOBRETIEMPO NORMAL

19. APILADO CON ESTÁTICA POR ELEVACIÓN

20. EDITADO DE PRIMEROS ARRIBOS : Offset 2400 m

21. CÁLCULO Y APLICACIÓN DE ESTÁTICA POR REFRACCIÓN

22. GENERACION NUEVA ESTÁTICA PARA RECEPTOR

    Se modifican los valores de la estática de receptor, calculados a partir del modelo generado para la capa somera.

23. CORRECCIÓN POR SOBRETIEMPO NORMAL

24. APILADO CON ESTÁTICA POR REFRACCIÓN AJUSTADA

25. ANÁLISIS DE VELOCIDAD

26. CÁLCULO Y APLICACIÓN DE ESTÁTICA RESIDUAL

    Ventana : 500 ms

    Trazas de referencia (Smash) : 3

27. CORRECCIÓN POR SOBRETIEMPO NORMAL

28. APILADO CON ESTÁTICA RESIDUAL

Proceso Post-Apilado

29. PASO DE VELOCIDADES RMS A DATUM

30. SUAVIZADO DE VELOCIDADES

31. MIGRACIÓN EN TIEMPO

Tipo :Kirchhoff

Porcentaje :100%

Máximo Buzamiento :65

Para visualización

32. DECONVOLUCIÓN F-X

33. FILTRO PASABANDA TIPO : Ormsby bandpass

Dominio : Frecuencia

Valores de Frecuencia : 5-10-16-50-70 Hz

34. CONTROL AUTOMATICO DE GANANCIA (AGC)

Longitud de Operador : 500 ms

ANEXO B1, PUNTO DE CONVERSIÓN Vs PROFUNDIDAD

```
% La omisión de tildes se debe al manejo de lenguaje c.
% se grafica la curva distancia de conversión vs profundidad
%Solucion de la ecuacion 6.27, polinomio de cuarto orden.
% D^4 + (Z^2 - X/2)D^2 - (X K Z^2)D + (X^4 +4X^2 Z^2)/16 = 0
% Xp distancia Fuente al Punto de Conversion  (en metros)
% D distancia Punto medio al Punto de Conversion (en metros)
% X distancia Fuente-Receptor, offset (en metros)
% Z Profundidad del reflector de interes (en metros)
% Vps relacion Vp/Vs
% K = (Vps^2 + 1)/(Vps^2 - 1)
% C4 = 1.0
% C3 = 0
% C2 = (Z^2  - X^2 /2)
% C1 = - (X K Z^2)
% C0 = (X^4 + 4X^2 Z^2)/16
 clear
%Calculo y graficado de la curva CCP vs Depth para Vp/Vs = 1.7, 2.0 y 2.3
%para un offset X = 4000 m.
%
 % Parametros Iniciales
Zmax = 4000;
N = round(Zmax/100);
Vps = 1.7;
X = 4000;
for j= 1:N
 Z(j)= j*100;
    K = ((Vps^2) + 1)/((Vps^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z(j)^2) - (X^2)/2;
    C1 = -1*(X*K*(Z(j)^2));
    C0 = ((X^4) + 4*(X^2)*(Z(j)^2))/16;
    % Calculo de las Raices del Polinomio, es decir D
    P = [C4 C3 C2 C1 C0];
    D = roots(P);
    % Calculo de los posibles Puntos de Conversion, CCP.
     for i = 1:4
      Xp(i) = D(i) + X/2;
     end
    %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
    if (Xp(1)>= X/2) && (Xp(1) <= X)
     XXp(j) = Xp(1);
    elseif (Xp(2)>= X/2) && (Xp(2) <= X)
     XXp(j) = Xp(2);
    elseif (Xp(3)>= X/2) && (Xp(3) <= X)
     XXp(j) = Xp(3);
    else (Xp(4)>= X/2) && (Xp(4) <= X)
     XXp(j) = Xp(4);
    end
  XXp(j);
  end
plot(XXp,Z,'r')
```

```
hold on;
 % Parametros Iniciales
Zmax = 4000;
N = round(Zmax/100);
Vps = 2.0;
X = 4000;
for j= 1:N
 Z(j)= j*100;
    K = ((Vps^2) + 1)/((Vps^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z(j)^2) - (X^2)/2;
    C1 = -1*(X*K*(Z(j)^2));
    C0 = ((X^4) + 4*(X^2)*(Z(j)^2))/16;
    % Calculo de las Raices del Polinomio, es decir D
    P = [C4 C3 C2 C1 C0];
    D = roots(P);
    % Calculo de los posibles Puntos de Conversion, CCP.
     for i = 1:4
      Xp(i) = D(i) + X/2;
     end
    %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
    if (Xp(1)>= X/2) && (Xp(1) <= X)
     XXp(j) = Xp(1);
    elseif (Xp(2)>= X/2) && (Xp(2) <= X)
     XXp(j) = Xp(2);
    elseif (Xp(3)>= X/2) && (Xp(3) <= X)
     XXp(j) = Xp(3);
    else (Xp(4)>= X/2) && (Xp(4) <= X)
     XXp(j) = Xp(4);
    end
  XXp(j);
end

plot(XXp,Z,'b')
hold on;

% Parametros Iniciales
Zmax = 4000;
N = round(Zmax/100);
Vps = 2.3;
X = 4000;
for j= 1:N
 Z(j)= j*100;
    K = ((Vps^2) + 1)/((Vps^2) - 1);
    C4 = 1.0;
    C3 = 0;

    % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z(j)^2) - (X^2)/2;
    C1 = -1*(X*K*(Z(j)^2));
    C0 = ((X^4) + 4*(X^2)*(Z(j)^2))/16;
```

```
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
    XXp(j) = Xp(1);
   elseif (Xp(2)>= X/2) && (Xp(2) <= X)
    XXp(j) = Xp(2);
   elseif (Xp(3)>= X/2) && (Xp(3) <= X)
    XXp(j) = Xp(3);
   else (Xp(4)>= X/2) && (Xp(4) <= X)
    XXp(j) = Xp(4);
   end
  XXp(j);
  end
plot(XXp,Z,'g')
 %Calculo y graficado de la curva CCP vs Depth para Vp/Vs = 1.7, 2.0 y 2.3
%para un offset X = 2000 m.
%
 % Parametros Iniciales
Zmax = 4000;
N = round(Zmax/100);
Vps = 1.7;
X = 2000;
for j= 1:N
 Z(j)= j*100;
   K = ((Vps^2) + 1)/((Vps^2) - 1);
   C4 = 1.0;
   C3 = 0;
    % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
   C4 = 1.0;
   C3 = 0;
   C2 = (Z(j)^2) - (X^2)/2;
   C1 = -1*(X*K*(Z(j)^2));
   C0 = ((X^4) + 4*(X^2)*(Z(j)^2))/16;
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
    XXp(j) = Xp(1);
   elseif (Xp(2)>= X/2) && (Xp(2) <= X)
    XXp(j) = Xp(2);
   elseif (Xp(3)>= X/2) && (Xp(3) <= X)
    XXp(j) = Xp(3);
   else (Xp(4)>= X/2) && (Xp(4) <= X)
    XXp(j) = Xp(4);
   end
  XXp(j);
```

```
  end
plot(XXp,Z,'r')
hold on;
 % Parametros Iniciales
Zmax = 4000;
N = round(Zmax/100);
Vps = 2.0;
X = 2000;
for j= 1:N
 Z(j)= j*100;
    K = ((Vps^2) + 1)/((Vps^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z(j)^2) - (X^2)/2;
    C1 = -1*(X*K*(Z(j)^2));
    C0 = ((X^4) + 4*(X^2)*(Z(j)^2))/16;
    % Calculo de las Raices del Polinomio, es decir D
    P = [C4 C3 C2 C1 C0];
    D = roots(P);
    % Calculo de los posibles Puntos de Conversion, CCP.
     for i = 1:4
      Xp(i) = D(i) + X/2;
     end
    %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
    if (Xp(1)>= X/2) && (Xp(1) <= X)
     XXp(j) = Xp(1);
    elseif (Xp(2)>= X/2) && (Xp(2) <= X)
     XXp(j) = Xp(2);
    elseif (Xp(3)>= X/2) && (Xp(3) <= X)
     XXp(j) = Xp(3);
    else (Xp(4)>= X/2) && (Xp(4) <= X)
     XXp(j) = Xp(4);
    end
  XXp(j);

end

plot(XXp,Z,'b')
hold on;

% Parametros Iniciales
Zmax = 4000;
N = round(Zmax/100);
Vps = 2.3;
X = 2000;
for j= 1:N
 Z(j)= j*100;
    K = ((Vps^2) + 1)/((Vps^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z(j)^2) - (X^2)/2;
```

```
   C1 = -1*(X*K*(Z(j)^2));
   C0 = ((X^4) + 4*(X^2)*(Z(j)^2))/16;
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
    XXp(j) = Xp(1);
   elseif (Xp(2)>= X/2) && (Xp(2) <= X)
    XXp(j) = Xp(2);
   elseif (Xp(3)>= X/2) && (Xp(3) <= X)
    XXp(j) = Xp(3);
   else (Xp(4)>= X/2) && (Xp(4) <= X)
    XXp(j) = Xp(4);
   end
  XXp(j);
  end
plot(XXp,Z,'g')
 %Calculo y graficado de la curva CCP vs Depth para Vp/Vs = 1.7, 2.0 y 2.3
%para un offset X = 500 m.
%
 % Parametros Iniciales
Zmax = 4000;
N = round(Zmax/100);
Vps = 1.7;
X = 500;
for j= 1:N
 Z(j)= j*100;
   K = ((Vps^2) + 1)/((Vps^2) - 1);
   C4 = 1.0;
   C3 = 0;
    % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
   C4 = 1.0;
   C3 = 0;
   C2 = (Z(j)^2) - (X^2)/2;
   C1 = -1*(X*K*(Z(j)^2));
   C0 = ((X^4) + 4*(X^2)*(Z(j)^2))/16;
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
    XXp(j) = Xp(1);
   elseif (Xp(2)>= X/2) && (Xp(2) <= X)
    XXp(j) = Xp(2);
   elseif (Xp(3)>= X/2) && (Xp(3) <= X)
    XXp(j) = Xp(3);
   else (Xp(4)>= X/2) && (Xp(4) <= X)
    XXp(j) = Xp(4);
```

```
    end
  XXp(j);
  end
plot(XXp,Z,'r')
hold on;
 % Parametros Iniciales
Zmax = 4000;
N = round(Zmax/100);
Vps = 2.0;
X = 500;
for j= 1:N
 Z(j)= j*100;
    K = ((Vps^2) + 1)/((Vps^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z(j)^2) - (X^2)/2;
    C1 = -1*(X*K*(Z(j)^2));
    C0 = ((X^4) + 4*(X^2)*(Z(j)^2))/16;
    % Calculo de las Raices del Polinomio, es decir D
    P = [C4 C3 C2 C1 C0];
    D = roots(P);
    % Calculo de los posibles Puntos de Conversion, CCP.
     for i = 1:4
      Xp(i) = D(i) + X/2;
     end
    %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
    if (Xp(1)>= X/2) && (Xp(1) <= X)
     XXp(j) = Xp(1);
    elseif (Xp(2)>= X/2) && (Xp(2) <= X)
     XXp(j) = Xp(2);
    elseif (Xp(3)>= X/2) && (Xp(3) <= X)
     XXp(j) = Xp(3);
    else (Xp(4)>= X/2) && (Xp(4) <= X)
     XXp(j) = Xp(4);
    end
  XXp(j);
  end
 plot(XXp,Z,'b')
hold on;
 % Parametros Iniciales
Zmax = 4000;
N = round(Zmax/100);
Vps = 2.3;
X = 500;
for j= 1:N
 Z(j)= j*100;
    K = ((Vps^2) + 1)/((Vps^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z(j)^2) - (X^2)/2;
    C1 = -1*(X*K*(Z(j)^2));
```

```
   C0 = ((X^4) + 4*(X^2)*(Z(j)^2))/16;
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
    XXp(j) = Xp(1);
   elseif (Xp(2)>= X/2) && (Xp(2) <= X)
    XXp(j) = Xp(2);
   elseif (Xp(3)>= X/2) && (Xp(3) <= X)
    XXp(j) = Xp(3);
   else (Xp(4)>= X/2) && (Xp(4) <= X)
    XXp(j) = Xp(4);
   end
  XXp(j);

end
plot(XXp,Z,'g')
title('Distancia Horizontal Punto de Conversion (m)')
ylabel('Profundidad Reflector (m)')
legend('Vp/Vs =1.7','Vp/Vs =2.0','Vp/Vs =2.3')
 grid;xlabel('Offset (m)');ylabel('Profundidad (m)');
set(gca,'YDir','reverse');
```

ANEXO B2, PUNTO DE CONVERIÓN vs RELACIÓN *Vp/Vs*

% Se grafica la curva distancia de conversion normalizada vs variación en el valor % *Vp/Vs*

```
%Solucion de la ecuacion 6.27, polinomio de cuarto orden.
% D^4 + (Z^2 - X/2)D^2 - (X K Z^2)D + (X^4 +4X^2 Z^2)/16 = 0
% Xp distancia Fuente al Punto de Conversion  (en metros)
% D distancia Punto medio al Punto de Conversion (en metros)
% X distancia Fuente-Receptor, offset (en metros)
% Z Profundidad del reflector de interes (en metros)
% Vps relacion Vp/Vs
% K = (Vps^2 + 1)/(Vps^2 - 1)
% C4 = 1.0
% C3 = 0
% C2 = (Z^2  - X^2 /2)
% C1 = - (X K Z^2)
% C0 = (X^4 + 4X^2 Z^2)/16

%Calculamos y Graficamos la Distancia del Punto de Conversion vs el cociente Vp/Vs
% para una profundidad Z = 1000m, 2000m, 3000m y 4000m y offsets X = 500m, 2000m y 4000m
 clear
 %**************************************
% Profundidad Z = 1000m
% Parametros Iniciales
 Z = 1000;
 Vpsmax = 3.0;
 X = 500;
    N = round((Vpsmax-1.5)*10) + 1;
 for j= 1:N
  Vps(j)= 1.4 + 0.1*j;
        K = ((Vps(j)^2) + 1)/((Vps(j)^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z^2) - (X^2)/2;
    C1 = -1*(X*K*(Z^2));
    C0 = ((X^4) + 4*(X^2)*(Z^2))/16;
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
     XXp(j) = Xp(1);
    elseif (Xp(2)>= X/2) && (Xp(2) <= X)
    XXp(j) = Xp(2);
    elseif (Xp(3)>= X/2) && (Xp(3) <= X)
    XXp(j) = Xp(3);
    else (Xp(4)>= X/2) && (Xp(4) <= X)
    XXp(j) = Xp(4);
  end
 XXp(j);
  end
 plot(Vps,XXp/X,'-r')
```

```
 hold on;
    % Parametros Iniciales
Z = 1000;
Vpsmax = 3.0;
X = 2000;
    N = round((Vpsmax-1.5)*10) + 1;
 for j= 1:N
  Vps(j)= 1.4 + 0.1*j;
         K = ((Vps(j)^2) + 1)/((Vps(j)^2) - 1);
     C4 = 1.0;
     C3 = 0;
      % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
     C4 = 1.0;
     C3 = 0;
     C2 = (Z^2) - (X^2)/2;
     C1 = -1*(X*K*(Z^2));
     C0 = ((X^4) + 4*(X^2)*(Z^2))/16;
    % Calculo de las Raices del Polinomio, es decir D
    P = [C4 C3 C2 C1 C0];
    D = roots(P);
    % Calculo de los posibles Puntos de Conversion, CCP.
     for i = 1:4
      Xp(i) = D(i) + X/2;
     end
    %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
    if (Xp(1)>= X/2) && (Xp(1) <= X)
      XXp(j) = Xp(1);
      elseif (Xp(2)>= X/2) && (Xp(2) <= X)
      XXp(j) = Xp(2);
      elseif (Xp(3)>= X/2) && (Xp(3) <= X)
      XXp(j) = Xp(3);
      else (Xp(4)>= X/2) && (Xp(4) <= X)
      XXp(j) = Xp(4);
    end
  XXp(j);
   end
  plot(Vps,XXp/X,'xr')
 hold on;
 % Parametros Iniciales
Z = 1000;
Vpsmax = 3.0;
X = 4000;
    N = round((Vpsmax-1.5)*10) + 1;
 for j= 1:N
  Vps(j)= 1.4 + 0.1*j;
        K = ((Vps(j)^2) + 1)/((Vps(j)^2) - 1);
     C4 = 1.0;
     C3 = 0;
      % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
     C4 = 1.0;
     C3 = 0;
     C2 = (Z^2) - (X^2)/2;
     C1 = -1*(X*K*(Z^2));
     C0 = ((X^4) + 4*(X^2)*(Z^2))/16;
    % Calculo de las Raices del Polinomio, es decir D
    P = [C4 C3 C2 C1 C0];
    D = roots(P);
```

```
    % Calculo de los posibles Puntos de Conversion, CCP.
     for i = 1:4
      Xp(i) = D(i) + X/2;
     end
    %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
    if (Xp(1)>= X/2) && (Xp(1) <= X)
      XXp(j) = Xp(1);
      elseif (Xp(2)>= X/2) && (Xp(2) <= X)
      XXp(j) = Xp(2);
      elseif (Xp(3)>= X/2) && (Xp(3) <= X)
      XXp(j) = Xp(3);
      else (Xp(4)>= X/2) && (Xp(4) <= X)
      XXp(j) = Xp(4);
    end
   XXp(j);
    end
   plot(Vps,XXp/X,'+r')
  hold on;
 % ***********************************************
% Profundidad Z = 2000m
% Parametros Iniciales
 Z = 2000;
 Vpsmax = 3.0;
 X = 500;
    N = round((Vpsmax-1.5)*10) + 1;
  for j= 1:N
   Vps(j)= 1.4 + 0.1*j;
         K = ((Vps(j)^2) + 1)/((Vps(j)^2) - 1);
     C4 = 1.0;
     C3 = 0;
    % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
     C4 = 1.0;
     C3 = 0;
     C2 = (Z^2) - (X^2)/2;
     C1 = -1*(X*K*(Z^2));
     C0 = ((X^4) + 4*(X^2)*(Z^2))/16;
    % Calculo de las Raices del Polinomio, es decir D
    P = [C4 C3 C2 C1 C0];
    D = roots(P);
    % Calculo de los posibles Puntos de Conversion, CCP.
     for i = 1:4
      Xp(i) = D(i) + X/2;
     end
    %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
    if (Xp(1)>= X/2) && (Xp(1) <= X)
      XXp(j) = Xp(1);
      elseif (Xp(2)>= X/2) && (Xp(2) <= X)
      XXp(j) = Xp(2);
      elseif (Xp(3)>= X/2) && (Xp(3) <= X)
      XXp(j) = Xp(3);
      else (Xp(4)>= X/2) && (Xp(4) <= X)
      XXp(j) = Xp(4);
    end
   XXp(j);
    end
   plot(Vps,XXp/X,'-g')
  hold on;
```

```
 % Parametros Iniciales
Z = 2000;
Vpsmax = 3.0;
X = 2000;
   N = round((Vpsmax-1.5)*10) + 1;
 for j= 1:N
  Vps(j)= 1.4 + 0.1*j;
         K = ((Vps(j)^2) + 1)/((Vps(j)^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z^2) - (X^2)/2;
    C1 = -1*(X*K*(Z^2));
    C0 = ((X^4) + 4*(X^2)*(Z^2))/16;
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
     XXp(j) = Xp(1);
     elseif (Xp(2)>= X/2) && (Xp(2) <= X)
     XXp(j) = Xp(2);
     elseif (Xp(3)>= X/2) && (Xp(3) <= X)
     XXp(j) = Xp(3);
     else (Xp(4)>= X/2) && (Xp(4) <= X)
     XXp(j) = Xp(4);
   end
  XXp(j);
   end

 plot(Vps,XXp/X,'xg')
 hold on;
%************************************************
% ************************************************
% Profundidad Z = 3000m
% Parametros Iniciales
Z = 3000;
Vpsmax = 3.0;
X = 500;
   N = round((Vpsmax-1.5)*10) + 1;
 for j= 1:N
  Vps(j)= 1.4 + 0.1*j;
         K = ((Vps(j)^2) + 1)/((Vps(j)^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z^2) - (X^2)/2;
    C1 = -1*(X*K*(Z^2));
    C0 = ((X^4) + 4*(X^2)*(Z^2))/16;
```

```
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
     XXp(j) = Xp(1);
     elseif (Xp(2)>= X/2) && (Xp(2) <= X)
     XXp(j) = Xp(2);
     elseif (Xp(3)>= X/2) && (Xp(3) <= X)
     XXp(j) = Xp(3);
     else (Xp(4)>= X/2) && (Xp(4) <= X)
     XXp(j) = Xp(4);
   end
  XXp(j);
   end
  plot(Vps,XXp/X,'xb')
 hold on;
 % Parametros Iniciales
Z = 3000;
Vpsmax = 3.0;
X = 2000;
   N = round((Vpsmax-1.5)*10) + 1;
 for j= 1:N
  Vps(j)= 1.4 + 0.1*j;
        K = ((Vps(j)^2) + 1)/((Vps(j)^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z^2) - (X^2)/2;
    C1 = -1*(X*K*(Z^2));
    C0 = ((X^4) + 4*(X^2)*(Z^2))/16;
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
     XXp(j) = Xp(1);
     elseif (Xp(2)>= X/2) && (Xp(2) <= X)
     XXp(j) = Xp(2);
     elseif (Xp(3)>= X/2) && (Xp(3) <= X)
     XXp(j) = Xp(3);
     else (Xp(4)>= X/2) && (Xp(4) <= X)
     XXp(j) = Xp(4);
   end
  XXp(j);

 end
```

```
  plot(Vps,XXp/X,'+b')
 hold on;
% Parametros Iniciales
Z = 3000;
Vpsmax = 3.0;
X = 4000;
  N = round((Vpsmax-1.5)*10) + 1;
 for j= 1:N
  Vps(j)= 1.4 + 0.1*j;
       K = ((Vps(j)^2) + 1)/((Vps(j)^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z^2) - (X^2)/2;
    C1 = -1*(X*K*(Z^2));
    C0 = ((X^4) + 4*(X^2)*(Z^2))/16;
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
     XXp(j) = Xp(1);
     elseif (Xp(2)>= X/2) && (Xp(2) <= X)
     XXp(j) = Xp(2);
     elseif (Xp(3)>= X/2) && (Xp(3) <= X)
     XXp(j) = Xp(3);
     else (Xp(4)>= X/2) && (Xp(4) <= X)
     XXp(j) = Xp(4);
   end
  XXp(j);
   end
%  plot(Vps,XXp/X,'vb')
 hold on;
%************************************************
% ***********************************************
% Profundidad Z = 4000m
% Parametros Iniciales
 Z = 4000;
 Vpsmax = 3.0;
 X = 500;
  N = round((Vpsmax-1.5)*10) + 1;
 for j= 1:N
  Vps(j)= 1.4 + 0.1*j;
       K = ((Vps(j)^2) + 1)/((Vps(j)^2) - 1);
    C4 = 1.0;
    C3 = 0;
     % Calculo de constantes del polinomio C4*D^4 + C3*D^3 + C2*D^2 + C1*D + C0 = 0
    C4 = 1.0;
    C3 = 0;
    C2 = (Z^2) - (X^2)/2;
    C1 = -1*(X*K*(Z^2));
```

```
    C0 = ((X^4) + 4*(X^2)*(Z^2))/16;
   % Calculo de las Raices del Polinomio, es decir D
   P = [C4 C3 C2 C1 C0];
   D = roots(P);
   % Calculo de los posibles Puntos de Conversion, CCP.
    for i = 1:4
     Xp(i) = D(i) + X/2;
    end
   %Escogencia del Punto de Conversion que cumpla X/2 < Xp < D + X/2
   if (Xp(1)>= X/2) && (Xp(1) <= X)
     XXp(j) = Xp(1);
     elseif (Xp(2)>= X/2) && (Xp(2) <= X)
     XXp(j) = Xp(2);
     elseif (Xp(3)>= X/2) && (Xp(3) <= X)
     XXp(j) = Xp(3);
     else (Xp(4)>= X/2) && (Xp(4) <= X)
     XXp(j) = Xp(4);
    end
  XXp(j);

 end
  plot(Vps,XXp/X,'-k')
title('Distancia Horizontal Punto de Conversion vs Vp/Vs')
%ylabel('Punto de Conversion Normalizado (m)')
%xlabel('Vp/Vs')
legend('z/x =8.0','z/x =6.0','z/x =4.0','z/x =2.0','z/x =1.5','z/x =1.0','z/x =0.5','z/x =0.25')
grid;xlabel('Vp/Vs');ylabel('Punto de Conversion Normalizado');
```

ANEXO B3, TRAYECTORIA DE RAYOS P Y PS

```
zp=0:10:4000;vp=1800+.6*zp;vs=.5*vp;zs=zp;     %velocity model

zsrc=100;zrec=500;zd=3000;            % source receiver and reflector depths

xoff=1000:100:3000;caprad=10;itermax=4;      %offsets, cap radius, and max inter

pfan=-1;optflag=1;pflag=1;dflag=2;         % default ray fan, and various flags
  figure;subplot(2,1,1);
[t,p]=traceray_pp(vp,zp,zsrc,zrec,zd,xoff,caprad,pfan,itermax,optflag,pflag,dflag);
title(['Simulación de modo de propagación P-P zsrc=' num2str(zsrc), ' zrec=' num2str(zrec)])
%line(xoff,zrec*ones(size(xoff)),'color','b','linestyle','none','marker','v')
line(0,zsrc,'color','r','linestyle','none','marker','*')
grid;xlabel('Offset (m)');ylabel('Profundidad (m)');
%plot(xoff,t);grid;
set(gca,'YDir','reverse');
%xlabel('meters');ylabel('seconds');
 %PS reflection
subplot(2,1,2);
[ts,p]=traceray_ps(vp,zp,vs,zs,zsrc,zrec,zd,xoff,caprad,pfan,itermax,optflag,pflag,dflag);
title(['Simulación de modo de propagación PS zsrc=' num2str(zsrc), ' zrec=' num2str(zrec)])
%line(xoff,zrec*ones(size(xoff)),'color','b','linestyle','none','marker','v')
line(0,zsrc,'color','r','linestyle','none','marker','*')
grid;xlabel('Offset (m)');ylabel('Profundidad (m)');
%plot(xoff,ts);grid;
set(gca,'YDir','reverse');
axis([0 1800 0 3000]);
%xlabel('meters');ylabel('seconds');
function [t,p,L]=traceray_pp(vp,zp,  zs,   zr,zd,x,xcap,pfan,itermax,optflag,pflag,dflag)
 % TRACERAY_PP: traces a P-P (or S-S) reflection for v(z)
%
% [t,p,L]=traceray_pp(vp,zp,zs,zr,zd,x,xcap,pfan,itermax,optflag,pflag,dflag)
%
% TRACERAY_PP uses the modified bisection algorithm
% to trace a p-p reflection
% in a stratified medium.
%
% vp,zp  = velocity and depth vectors giving the p wave model. Depths are
%      considered to be the tops of homogeneous layers.
%    Can also be used for an s-s reflection
% zs   = depth of the source (scalar)
% zr     = depth of receivers (scalar)
% zd   = depth of the reflection (scalar)
% x    = vector of desired source-receiver offsets (horizontal)
%      MUST be non-negative numbers
% xcap  = (scalar) capture radius
% pfan  = vector of ray parameters to use for the initial fan of rays
%    By default, the routine generates the fan using straight rays.
%    Setting pfan to -1 or omitting it activates default behavior.
%      Setting pfan to -2 causes the routine to use the pfan used in the
%      last call to this program. (pfan of -1 and -2 are identical on the
%    first call.)
% itermax = maximum number of iterations allowed for ray capture
% ============================ Default = 4 ===================================
% optflag = if nonzero then refine captured rays by linear interpolation
% ============================ Default = 1 ===================================
% pflag  = if nonzero, then print information about all failed rays
```

```
% ============================ Default = 0 ================================
% dflag  = 0 no action
%      = 1 then a new figure window will be opened and the raypaths plotted
%      = 2 raypaths will be plotted in the current window
% ============================ Default = 0 ================================
% NOTE: All z variables must be specified relative to a common datum
%
% t    = vector of traveltimes for each x
% p    = vector of ray parameters for each x
% L    = vector of geometrical spreading factors for each x
%   (if L is not asked for, it will not be calculated)
%
% NOTE: failed rays are flagged with inf (infinity) for traveltime and
%   nan for ray parameter.
%
if(nargin<12)
  dflag=0;
end
if(nargin<11)
  pflag=0;
end
if(nargin<10)
  optflag=1;
end
 if(nargin<9)
  itermax=4;
end
if( nargin<8)
  pfan=-1;
end
 if(~prod(double(size(vp)==size(zp))))
  error('vp and zp must be the same size');
end
 if(length(zs )~=1 | length(zr)~=1 | length(zd)~=1 | length(xcap)~=1 )
  error(' zs,zr,zd and xcap must be scalars ')
end
ind=find(x<0);
if(~isempty(ind))
  error('offsets must be nonnegative');
end
 %make sure zs < Zd and zr<zd
if(zs > zd )
  error(' zs must be less than zd');
end
if(zr > zd )
  error(' zr must be less than zd');
end
 %adjust zs,zr,and zd and z2 so that they won't be exactly on layer boundaries
  zs=zs+100000*eps;
  zr=zr+100000*eps;
  zd=zd-100000*eps;
  if( zs < zp(1) )
  error(' source depth outside model range');
 elseif(zr < zp(1))
  error('receiver depth outside model range');
 end
  %determine the layers we propagate through
```

```
%down leg
ind=find(zp>zs);
if(isempty(ind))
 ibegd=length(zp);
else
 ibegd=ind(1)-1;
end
ind=find(zp>zd);
if(isempty(ind))
 iendd=length(zp);
else
 iendd=ind(1)-1;
end
 %test for sensible layer indices
if(iendd<1 | iendd > length(zp) | ibegd <1 | iendd > length(zp))
  %somethings wrong. return nans
  t=inf*ones(size(x));
  p=nan*ones(size(x));
  return;
end
 %create v and z arrays for down leg
vp=vp(:);zp=zp(:);
vpd=[vp(ibegd:iendd);vp(iendd)];%last v is irrelevent.
zpd=[zs;zp(ibegd+1:iendd);zd];
%up leg
ind=find(zp>zr);
if(isempty(ind))
 ibegu=length(zp);
else
 ibegu=ind(1)-1;
end
ind=find(zp>zd);
if(isempty(ind))
 iendu=length(zp);
else
 iendu=ind(1)-1;
end
 %test for sensible layer indices
if(iendu<1 | iendu > length(zp) | ibegu <1 | iendu > length(zp))
  %somethings wrong. return nans
  t=inf*ones(size(x));
  p=nan*ones(size(x));
  return;
end
 %create v and z arrays for up leg
 vpu=[vp(ibegu:iendu);vp(iendu)];%last v is irrelevent.
zpu=[zr;zp(ibegu+1:iendu);zd];
 % combine into one model. We shoot oneway rays through an
% equivalent model consiting of the down-leg model followed by
% the upleg model (upside down).
vp1=[vpd(1:end-1);flipud(vpu(1:end-1))];
%vp1=[vpd;flipud(vpu(1:end-1))];
tmp=flipud(zpu);
zp1=[zpd;cumsum(abs(diff(tmp)))+zpd(end)];
%zp1=[zpd;cumsum(abs(diff(tmp(1:end-1))))+zpd(end)];
 z1=zs;%start depth
z2=max(zp1)+100000*eps;%end depth
```

```
 %determine pmax
vmax=max([vp1]);
vmin=min([vp1]);
nearlyone=1-10*eps;
%pmax=nearlyone/vmax;
pmax=nearlyone/vmin;
 %the meaning of pmax is that it is the maximum ray parameter which will not
%be critically refracted anywhere in vp or vs.
if(pfan==-2)
 global PFAN
 if(isempty(PFAN))
  pfan=-1;
 else
  pfan=PFAN;
  PFAN=-2;
 end
else
 PFAN=0;
end
 if(pfan==-1)
 %shoot a fan of rays Iterate once if the fan does not span the xrange
 %guess angle range
 nrays=max([3*length(x),10]);
 pfan=linspace(0,1/max(vp1),nrays);
else
 %make sure its a row vector
 ind=find(isnan(pfan));
 if(~isempty(ind))
   pfan(ind)=[];
 end
 pfan=pfan(:)';
 pfan=sort(pfan);
 ind=find(pfan==0);
 if(isempty(ind))
   pfan=[0 pfan];
 end
 ind=find(pfan>=pmax);
 if(~isempty(ind))
   pfan(ind)=[];
 end
 pfan=[pfan pmax];
 nrays=length(pfan);
end
 %shoot first fan
%
[xx,tt]=shootray(vp1,zp1,pfan);
xmax=max(xx);
xmin=min(xx);
 %see if we have spanned the x range and revise if needed
imin=surround(xx,min(x));
imax=surround(xx,max(x));
newfan=0;
if(isempty(imax))
 if(max(pfan)<pmax)
   pm= .5*(max(pfan)+pmax);
     pfan=[pfan pm];
   imax=length(pfan)-1;
```

```
   newfan=1;
  end
end
if(~isempty(imax))
  pfan=pfan(imin:imax+1);
end

%shoot new fan (if p changed)
if(length(pfan)~=nrays | newfan)
  % first p
  [xx,tt]=shootray(vp1,zp1,pfan);
  % invoke symmetry to double these
  xmax=max(xx);
  xmin=min(xx);
end
 %
% loop over x
%  -1 find xx which brackets x(k)
%  -2 shoot a finer fan of rays
%  -3 find the rays which bracket x(k)
%  -4 repeat 2 and 3 until a ray is captured at x(k)
%
 t=inf*ones(size(x));
p=nan*ones(size(x));
 for k=1:length(x)
   ind=surround(xx,x(k));
  if(isempty(ind))
    if(x(k)>=xmax) %test for off high end
      i1=length(xx);
      i2=[];
    else
      i1=[]; %off low end
      i2=1;
    end
  else
    if(length(ind)>1) %use first arrival
      it=find(tt(ind)==min(tt(ind)));
      ind=ind(it);
    end
    i1=ind;
    i2=ind+1;
  end
    %i1 and i2 are indices into pfan of the rays that bracket x(k)
    captured=0;
  hopeless=0;
  iter=0;
  x1=xx(i1); x2=xx(i2);
  t1=tt(i1); t2=tt(i2);
  p1=pfan(i1); p2=pfan(i2);
  if( isempty(i2) & max(pfan)==pmax)
    hopeless=1;
  end
  while(~captured & iter<itermax & ~hopeless)
      iter=iter+1;
      %test for capture, x1 and x2 are the offsets for rays pfan(i1) and pfan(i2)
    xtst1=abs(x(k)-x1);
    xtst2=abs(x(k)-x2);
```

```
if( xtst1< xcap & xtst1<=xtst2)
  captured=1;
  p(k)=p1;
  t(k)=t1;
  %final adjustment
  if(optflag)
    %linear interpolation
    t(k)= t1 + (x(k)-x1)*(t2-t1)/(x2-x1);
    p(k)= p1 + (x(k)-x1)*(p2-p1)/(x2-x1);
  end
          %disp([' capture on iter= ' int2str(iter)])
 elseif( xtst2 < xcap & xtst2<=xtst1)
  captured=1;
  p(k)=p2;
  t(k)=t2;
     %final adjustment
  if(optflag)
    %linear interpolation
    t(k)= t2 + (x(k)-x2)*(t1-t2)/(x1-x2);
    p(k)= p2 + (x(k)-x2)*(p1-p2)/(x1-x2);
  end
       %disp([' capture on iter= ' int2str(iter)])
end
    %shoot a new fan
if(~captured)
  %end cases first
  if(isempty(i1))
    p2=p1;
    p1=0;
  elseif(isempty(i2))
    %p2=.5*(p2+pmax);
    p1=p2;
    p2=pmax;
  end
  dp= (x(k)-x1)*(p2-p1)/(x2-x1);
  p0= p1+ dp;
  %p2= min([p1+2*dp, pmax]);
  %p2=min([p1+2*dp p2]);
  %p1= p1+.1*dp;

  %p0=.5*(p1+p2);

  %pnew now contains only 1 new ray. The code below espects this
  pnew=[p1 p0 p2];
  %nnew=3;
  %pnew=linspace(min([p1 p2]),max([p1 p2]),nnew);

  if(isempty(pnew))
    hopeless=1;
  else
    %shoot and check for nans
    %[xxnew,ttnew]=shootray(vp1,zp1,pnew);
    [xtmp,ttmp]=shootray(vp1,zp1,pnew(2));
    xxnew=[x1 xtmp x2];
    ttnew=[t1 ttmp t2];

    xmx=max(xxnew);
```

```
      xmn=min(xxnew);

      %analyze the fan. see if we have bracketed our target
      ind=surround(xxnew,x(k));
      if(isempty(ind))
       if(~isempty(xmx))
        if(x(k)>=xmx)
          n2=2*(x(k)-xxnew(2))/(xxnew(3)-xxnew(2));
          p2=min([p1+n2*dp, pmax]);
          pnew(3)=p2;
        end
        [xxnew,ttnew]=shootray(vp1,zp1,pnew);

        xmx=max(xxnew);
        xmn=min(xxnew);
        ind=surround(xxnew,x(k));
        if(isempty(ind));
          i1=3;
          i2=[];
        else
          i1=ind;
          i2=ind+1;
        end
       else
        i1=[];
        i2=1;
       end

      else
       if(length(ind)>1) %use first arrival
        it=find(ttnew(ind)==min(ttnew(ind)));
        ind=ind(it);
       end
       i1=ind;
       i2=ind+1;
      end
      if( isempty(i2) & max(pnew)==pmax)
       hopeless=1;
      else
       x1=xxnew(i1); x2=xxnew(i2);
       t1=ttnew(i1); t2=ttnew(i2);
       p1=pnew(i1); p2=pnew(i2);
      end
     end
    end

  end
  if(hopeless)
   disp(['hopeless after ' int2str(iter) ' iterations'])
  end
  %end the while loop
  if(~captured & pflag)
    disp([' after ' int2str(iter) ' iterations, capture failed for z1,z2= ' num2str(z1) ',' num2str(z2) ' and x= '
num2str(x(k))]);
  end
    %end the for loop
end
```

```
 if(PFAN)
  PFAN=pfan;
end
 if(dflag)
  if(dflag==1)
    figure;
  end
  [hray,xray]=drawray(vpd,zpd,zs,zd,0,p,'k');
  [hray2,xray2]=drawray(vpu,zpu,zd,zr,xray,p,'k');
    if(dflag==1)
    %#function flipy
    flipy;xlabel('offset');ylabel('depth');
    if(length(x)>1)
      dx=x(2)-x(1);
    else
      dx=100;
    end
    axis([min([x(:);0])-dx max([x(:);0])+dx min(zp) max([zp;zs;zr;zd])])
  end
end
 if(nargout>2)
  L=sphdiv(vp1,zp1,p);
end
```

```
function [t,p,L]=traceray_ps(vp,zp,vs,zs,zsrc,zr,zd,x,xcap,pfan,itermax,optflag,pflag,dflag)

% TRACERAY_PS: traces a PS (or S-P) reflection for v(z)
%
% [t,p,L]=traceray_ps(vp,zp,vs,zs,zsrc,zr,zd,x,xcap,pfan,itermax,optflag,pflag,dflag)
%
% TRACERAY_PS uses the modified bisection algorithm
% to trace a PS reflection
% in a stratified medium.
%
% vp,zp  = velocity and depth vectors giving the p wave model. Depths are
%      considered to be the tops of homogeneous layers.
% vs,zs  = velocity and depth vectors giving the s wave model.
% for both models, depths are considered to be the tops of homogeneous layers. Thus,
% the first velocity applies from the first depth to the second. A constant velocity
% would be specified like vp=1500;zp=0; etc.
% zsrc    = depth of the source (scalar)
% zr     = depth of receiver (scalar)
% zd   = depth of the reflection (scalar)
% x    = vector of desired source-receiver offsets (horizontal)
%      MUST be non-negative numbers
% xcap  = (scalar) capture radius
% pfan  = vector of ray parameters to use for the initial fan of rays
%    By default, the routine generates the fan using straight rays.
%    Setting pfan to -1 or omitting it activates default behavior.
%      Setting pfan to -2 causes the routine to use the pfan used in the
%      last call to this program. (pfan of -1 and -2 are identical on the
%    first call.)
% itermax = maximum number of iterations allowed for ray capture
% ============================ Default = 4 ================================
% optflag = if nonzero then refine captured rays by linear interpolation
% ============================ Default = 1 ================================
% pflag  = if nonzero, then print information about all failed rays
% ============================ Default = 0 ================================
% dflag  = 0 no action
%      = 1 then a new figure window will be opened and the raypaths plotted
%      = 2 raypaths will be plotted in the current window
% ============================ Default = 0 ================================
% NOTE: All z variables must be specified relative to a common datum
%
% t    = vector of traveltimes for each x
% p    = vector of ray parameters for each x
% L   = vector of geometrical spreading factors for each x
%   (if L is not asked for, it will not be calculated)
%
% G.F. Margrave, CREWES Project, June 1995
%
% NOTE: It is illegal for you to use this software for a purpose other
% than non-profit education or research UNLESS you are employed by a CREWES
% Project sponsor. By using this software, you are agreeing to the terms
% detailed in this software's Matlab source file.

% BEGIN TERMS OF USE LICENSE
%
% This SOFTWARE is maintained by the CREWES Project at the Department
% of Geology and Geophysics of the University of Calgary, Calgary,
% Alberta, Canada. The copyright and ownership is jointly held by
```

```
% its author (identified above) and the CREWES Project. The CREWES
% project may be contacted via email at: crewesinfo@crewes.org %
% The term 'SOFTWARE' refers to the Matlab source code, translations to
% any other computer language, or object code %
% Terms of use of this SOFTWARE %
% 1) Use of this SOFTWARE by any for-profit commercial organization is
%  expressly forbidden unless said organization is a CREWES Project
%  Sponsor. %
% 2) A CREWES Project sponsor may use this SOFTWARE under the terms of the
%  CREWES Project Sponsorship agreement. %
% 3) A student or employee of a non-profit educational institution may
%  use this SOFTWARE subject to the following terms and conditions:
%  - this SOFTWARE is for teaching or research purposes only.
%  - this SOFTWARE may be distributed to other students or researchers
%   provided that these license terms are included.
%  - reselling the SOFTWARE, or including it or any portion of it, in any
%   software that will be resold is expressly forbidden.
%  - transfering the SOFTWARE in any form to a commercial firm or any
%   other for-profit organization is expressly forbidden.
%
% END TERMS OF USE LICENSE
 if(nargin<14)
  dflag=0;
end
if(nargin<13)
  pflag=0;
end
if(nargin<12)
  optflag=1;
end

if(nargin<11)
  itermax=4;
end
if( nargin<10)
  pfan=-1;
end

if(~prod(double(size(vp)==size(zp))))
  error('vp and zp must be the same size');
end

if(~prod(double(size(vs)==size(zs))))
  error('vp and zp must be the same size');
end

if(length(zsrc)~=1 | length(zr)~=1 | length(zd)~=1 | length(xcap)~=1 )
  error(' zsrc,zr,zd and xcap must be scalars ')
end
ind=find(x<0);
if(~isempty(ind))
  error('offsets must be nonnegative');
end
 %make sure zsrc < Zd and zr zd )
if (zsrc>zd)
  error(' zsrc must be less than zd');
end if(zr > zd )
```

```
 error(' zr must be less than zd');
end
 %adjust zsrc,zr,and zd and z2 so that they won't be exactly on layer boundaries
 zsrc=zsrc+100000*eps;
 zr=zr+100000*eps;
 zd=zd-100000*eps;
 if( zsrc < zp(1) | zsrc < zs(1))
 error(' source depth outside model range');
elseif(zr < zp(1) | zr < zs(1) )
 error('receiver depth outside model range');
end
 %determine the layers we propagate through
%down leg
ind=find(zp>zsrc);
if(isempty(ind))
 ibegd=length(zp);
else
 ibegd=ind(1)-1;
end
ind=find(zp>zd);
if(isempty(ind))
 iendd=length(zp);
else
 iendd=ind(1)-1;
end
 %test for sensible layer indices
if(iendd<1 | iendd > length(zp) | ibegd <1 | iendd > length(zp))
  %somethings wrong. return nans
  t=inf*ones(size(x));
  p=nan*ones(size(x));
  return;
end
 %create v and z arrays for down leg
vp=vp(:);zp=zp(:);
vpd=[vp(ibegd:iendd);vp(iendd)];%last v is irrelevent.
zpd=[zsrc;zp(ibegd+1:iendd);zd];
%up leg
ind=find(zs>zr);
if(isempty(ind))
 ibegu=length(zs);
else
 ibegu=ind(1)-1;
end
ind=find(zs>zd);
if(isempty(ind))
 iendu=length(zs);
else
 iendu=ind(1)-1;
end
 %test for sensible layer indices
if(iendu<1 | iendu > length(zs) | ibegu <1 | iendu > length(zs))
  %somethings wrong. return nans
  t=inf*ones(size(x));
  p=nan*ones(size(x));
  return;
end
 %create v and z arrays for up leg
```

```
vs=vs(:);zs=zs(:);
vsu=[vs(ibegu:iendu);vs(iendu)];%last v is irrelevent.
zsu=[zr;zs(ibegu+1:iendu);zd];
 % combine into one model. We shoot oneway rays through an
% equivalent model consiting of the down-leg model followed by
% the upleg model (upside down).
vmod=[vpd(1:end-1);flipud(vsu(1:end-1))];
 tmp=flipud(zsu);
zmod=[zpd;cumsum(abs(diff(tmp)))+zpd(end)];

z1=zsrc;%start depth
z2=max(zmod)+100000*eps;%end depth
 %determine pmax
vmax=max([vmod]);
vmin=min([vmod]);
nearlyone=.999;
 pmax=nearlyone/vmax;
 %the meaning of pmax is that it is the maximum ray parameter which will not
%be critically refracted anywhere in vp or vs.
if(pfan==-2)
 global PFAN
 if(isempty(PFAN))
  pfan=-1;
 else
  pfan=PFAN;
  PFAN=-2;
 end
else
 PFAN=0;
end
 if(pfan==-1)
 %shoot a fan of rays Iterate once if the fan does not span the xrange
 %guess angle range
 nrays=max([3*length(x),10]);
 pfan=linspace(0,1/max(vmod),nrays);

else
 %make sure its a row vector
 ind=find(isnan(pfan));
 if(~isempty(ind))
   pfan(ind)=[];
 end
 pfan=pfan(:)';
 pfan=sort(pfan);
 ind=find(pfan==0);
 if(isempty(ind))
   pfan=[0 pfan];
 end
 ind=find(pfan>=pmax);
 if(~isempty(ind))
   pfan(ind)=[];
 end
 pfan=[pfan pmax];
 nrays=length(pfan);
end
 %shoot first fan
%
```

```
[xx,tt]=shootray(vmod,zmod,pfan);
 xmax=max(xx);
xmin=min(xx);
 %see if we have spanned the x range and revise if needed
imin=surround(xx,min(x));
imax=surround(xx,max(x));
newfan=0;
if(isempty(imax))
  if(max(pfan)==xmax) %test for off high end %!!!!!!!!!!!!!!!!!!!!!!!!
    pm= .5*(max(pfan)+pmax);
     pfan=[pfan pm];
    imax=length(pfan)-1;
    newfan=1;
  end
end
if(~isempty(imax))
  pfan=pfan(imin:imax+1);
end

%shoot new fan (if p changed)
if(length(pfan)~=nrays | newfan)
  % first p
  [xx,tt]=shootray(vmod,zmod,pfan);
  % invoke symmetry to double these
  xmax=max(xx);
  xmin=min(xx);
end

%
% loop over x
%  -1 find xx which brackets x(k)
%  -2 shoot a finer fan of rays
%  -3 find the rays which bracket x(k)
%  -4 repeat 2 and 3 until a ray is captured at x(k)
%
 t=inf*ones(size(x));
p=nan*ones(size(x));
 for k=1:length(x)
   ind=surround(xx,x(k));
  if(isempty(ind))
    if(x(k)>=xmax) %test for off high end
      i1=length(xx); %!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      i2=[];
    else
      i1=[]; %off low end
      i2=1;
    end
  else %PROBLEMA ELSE!!!!!!!
    if(length(ind)>1) %use first arrival
      it=find(tt(ind)==min(tt(ind)));
      ind=ind(it);
    end
    i1=ind;
    i2=ind+1;
  end

  %i1 and i2 are indices into pfan of the rays that bracket x(k)
```

```
   captured=0;
 hopeless=0;
 iter=0;
 x1=xx(i1); x2=xx(i2);
 t1=tt(i1); t2=tt(i2);
 p1=pfan(i1); p2=pfan(i2);
 if( isempty(i2) & max(pfan)==pmax)
  hopeless=1;
 end
   while(~captured & iter<itermax & ~hopeless)%  while(~captured & iter<xcap & xtst1<=xtst2)
     iter=iter+1;
   %test for capture, x1 and x2 are the offsets for rays pfan(i1) and pfan(i2)
  xtst1=abs(x(k)-x1);
  xtst2=abs(x(k)-x2);
  if( xtst1< xcap & xtst1<=xtst2)
   captured=1;
   p(k)=p1;
   t(k)=t1;
   %final adjustment
   if(optflag)
    %linear interpolation
    t(k)= t1 + (x(k)-x1)*(t2-t1)/(x2-x1);
    p(k)= p1 + (x(k)-x1)*(p2-p1)/(x2-x1);
   end
           %disp([' capture on iter= ' int2str(iter)])
  elseif( xtst2 < xcap & xtst2<=xtst1)
   captured=1;
   p(k)=p2;
   t(k)=t2;
       %final adjustment
   if(optflag)
    %linear interpolation
    t(k)= t2 + (x(k)-x2)*(t1-t2)/(x1-x2);
    p(k)= p2 + (x(k)-x2)*(p1-p2)/(x1-x2);
   end
        %disp([' capture on iter= ' int2str(iter)])
 end
  %shoot a new fan
 if(~captured)
  %end cases first
  if(isempty(i1))
    p2=p1;
    p1=0;
  elseif(isempty(i2))
    %p2=.5*(p2+pmax);
    p1=p2;
    p2=pmax;
  end
  dp= (x(k)-x1)*(p2-p1)/(x2-x1);
  p0= p1+ dp;
  %p2= min([p1+2*dp, pmax]);
  %p2=min([p1+2*dp p2]);
  %p1= p1+.5*dp;
   %p0=.5*(p1+p2);
       %pnew now contains only 1 new ray. The code below espects this
  pnew=[p1 p0 p2];
  %nnew=10;
```

```
  %pnew=linspace(min([p1 p2]),max([p1 p2]),nnew);
       if(isempty(pnew))
   hopeless=1;
  else
   %shoot and check for nans
   %[xxnew,ttnew]=shootray(vmod,zmod,pnew);
   [xtmp,ttmp]=shootray(vmod,zmod,pnew(2));
   xxnew=[x1 xtmp x2];
   ttnew=[t1 ttmp t2];
     xmx=max(xxnew);
   xmn=min(xxnew);
         %analyze the fan. see if we have bracketed our target
   ind=surround(xxnew,x(k));
   if(isempty(ind))
    if(~isempty(xmx))
     if(x(k)>=xmx)
      n2=2*(x(k)-xxnew(2))/(xxnew(3)-xxnew(2));
      p2=min([p1+n2*dp, pmax]);
      pnew(3)=p2;
     end
     [xxnew,ttnew]=shootray(vmod,zmod,pnew);

     xmx=max(xxnew);
     xmn=min(xxnew);
     ind=surround(xxnew,x(k));
     if(isempty(ind));
      i1=3;
      i2=[];
     else
      i1=ind;
      i2=ind+1;
     end
    else
     i1=[];
     i2=1;
    end

   else
    if(length(ind)>1) %use first arrival
     it=find(ttnew(ind)==min(ttnew(ind)));
     ind=ind(it);
    end
    i1=ind;
    i2=ind+1;
   end
   if( isempty(i2) & max(pnew)==pmax)
    hopeless=1;
   else
    x1=xxnew(i1); x2=xxnew(i2);
    t1=ttnew(i1); t2=ttnew(i2);
    p1=pnew(i1); p2=pnew(i2);
   end
  end
 end

end
if(hopeless)
```

```
   disp(['hopeless after ' int2str(iter) ' iterations'])
  end
  %end the while loop
  if(~captured & pflag)
    disp([' capture failed for zsrc,zr,zd= ' num2str(zsrc) ',' num2str(zr) ',' num2str(zd)       ' and x= ' num2str(x(k))]);
  end

  %end the for loop
end

if(PFAN)
  PFAN=pfan;
end

if(dflag)
  if(dflag==1)
    figure;
  end
  [hray,xray]=drawray(vpd,zpd,zsrc,zd,0,p,'k');
  [hray2,xray2]=drawray(vsu,zsu,zd,zr,xray,p,'r');

  if(dflag==1)
    %#function flipy
    flipy;xlabel('offset');ylabel('depth');
    if(length(x)>1)
      dx=x(2)-x(1);
    else
      dx=100;
    end
    axis([min([x(:);0])-dx max([x(:);0])+dx min(zp) max([zp;zsrc;zr;zd])])
  end
end

if(nargout>2)
  L=sphdiv(vmod,zmod,p);
end
```

```
function [x,t]=shootray(v,z,p)
%
% [x,t]=shootray(v,z,p)
%
% SHOOTRAY shoots a single ray or fan of rays through a stratified
% velocity model.
% It is assumed that v is a vector of interval velocities and that
% the ray goes from z(1) to z(length(z)). This routine does no error
% checking for efficiency. Be sure that v and z are both the same
% length and that p is a row vector. It returns inf if a critical
% refration occurs. This routine will get the same results as rayfan
% but is 25% faster
%
% v... COLUMN vector of interval velocities
% z... COLUMN vector of depths to the tops of the intervals
% NOTE: v and z must be at least length 2 or an abort will occur
% p... scalar or ROW vector ray parameters (Must be a row vector.)
% x ... scalar or vector of horizontal distances
% t ... scalar or vector of traveltimes
%

%check for critical angle
iprop=1:length(z)-1;
sn = v(iprop)*p;
 %
 % sn is an n by m matrix where n is the length of iprop (the number of
 %  layers propagated through) and m is the length of p (the number of
 %  unique ray parameters to use). Each column of sn corresponds to a
 %  single ray parameter and contains the sin of the vertical angle;
 %
[ichk,pchk]=find(sn>1);
 %compute x and t
cs=sqrt(1-sn.*sn)+eps;
vprop=v(iprop)*ones(1,length(p));
thk=abs(diff(z))*ones(1,length(p));
if(size(sn,1)>1)
 x=sum( (thk.*sn)./cs);
 t=sum(thk./(vprop.*cs));
else
 x=(thk.*sn)./cs;
 t=thk./(vprop.*cs);
end
%assign infs
if(~isempty(ichk))
 x(pchk)=inf*ones(size(pchk));
 t(pchk)=inf*ones(size(pchk));
end
function [hray,xray]=drawray(v,z,z1,z2,x1,p,kol,lns,lw,d3)
% DRAWRAY: draws rays given their ray parameters
%
% [hray,xray]=drawray(v,z,z1,z2,x1,p,kol,lns,lw,d3)
%
% DRAWRAY draws the rays whose rayparameters are found in the vector
% p in the stratified medium defined by v & z. Rays are drawn in
% the current axes and their handles are returned.
%
% v,z ... vectors describing the stratified medium, (Interval
```

```
%  velocity versus depth)
% z1 ... depth the ray starts at
% z2 ... depth the ray ends at.
% x1 ... vector of x coordinates the rays start at
% p ... vector of ray parameters
% kol ... color to draw the ray in
% ******* default 'g' *********
% lns ... linestyle to darw with
% ******* default '-' *********
% lw ... linewidth to draw the ray
% ******* default .5 *********
% d3 ... if 1, then the ray will be drawn in front of the current figure
% ******* default 0 *********
% hray ... vector of handles of the rays (one per element of p)
% xray ... vector of x coordinates that the rays end at
%
% G.F. Margrave, CREWES Project, Feb 2000
%
% NOTE: It is illegal for you to use this software for a purpose other
% than non-profit education or research UNLESS you are employed by a CREWES
% Project sponsor. By using this software, you are agreeing to the terms
% detailed in this software's Matlab source file.
 % BEGIN TERMS OF USE LICENSE
%
% This SOFTWARE is maintained by the CREWES Project at the Department
% of Geology and Geophysics of the University of Calgary, Calgary,
% Alberta, Canada. The copyright and ownership is jointly held by
% its author (identified above) and the CREWES Project. The CREWES
% project may be contacted via email at: crewesinfo@crewes.org
%
% The term 'SOFTWARE' refers to the Matlab source code, translations to
% any other computer language, or object code
%
% Terms of use of this SOFTWARE
%
% 1) Use of this SOFTWARE by any for-profit commercial organization is
%  expressly forbidden unless said organization is a CREWES Project
%  Sponsor.
%
% 2) A CREWES Project sponsor may use this SOFTWARE under the terms of the
%  CREWES Project Sponsorship agreement.
%
% 3) A student or employee of a non-profit educational institution may
%  use this SOFTWARE subject to the following terms and conditions:
%  - this SOFTWARE is for teaching or research purposes only.
%  - this SOFTWARE may be distributed to other students or researchers
%   provided that these license terms are included.
%  - reselling the SOFTWARE, or including it or any portion of it, in any
%   software that will be resold is expressly forbidden.
%  - transfering the SOFTWARE in any form to a commercial firm or any
%   other for-profit organization is expressly forbidden.
%
% END TERMS OF USE LICENSE

if(nargin<10) d3=0; end
if(nargin<9) lw=.5; end
if(nargin<8) lns='-';end
```

```
if(nargin<7) kol='g'; end
if(length(x1)==1)
  x1=x1*ones(size(p));
end

%preliminaries
 z=z(:);
 v=v(:);
 p=p(:)'; %make p a row vector
 nz=length(z);
 %adjust z1 and z2 so that they won't be exactly on layer boundaries
 zt=min([z1 z2]);
 zt
 z1
 if(zt==z1)
  z1=z1+100000*eps;
  z2=z2-100000*eps;
  inverted=0;
 else
  z1=z1-100000*eps;
  z2=z2+100000*eps;
  inverted=1;
 end
 if( z1< z(1) | z2< z(1) )
  error(' start or end depth outside model range');
 end

%determine layers propagated through
iprop=0;
inverted=0;
if(~inverted)
   ind=find(z>z1);%should never be ==
   if(isempty(ind))
    ibeg=nz;
   else
    ibeg=ind(1);
   end

   ind=find(z>z2);%should never be ==
   if(isempty(ind))
    iend=nz;
   else
    iend=ind(1);
   end
   if(ibeg~=iend)
     % these are the layers propagated thru
    iprop=ibeg:-1:iend;
   else
    iprop=ibeg;
   end
  v1=v(ibeg);
  v2=v(iend-1);
 end
 %algo=iprop;
 zprop=[z1;z(iprop);z2];%propagation depths
 %the reason for the following difference in the way velocity is assigned
 %is because a particular v applies to the model BELOW its corresponding
```

```
%depth
if(~inverted)
 vprop=[v1;v(iprop)];%last point (v2) is irrelevant
else
 vprop=[v1;v(iprop-1)];%last point (v2) is irrelevant
end
 %
% sn is an n by m matrix where n is the length of iprop (the number of
%  layers propagated through) and m is the length of p (the number of
%  unique ray parameters to use). Each column of sn corresponds to a
%  single ray parameter and contains the sin of the vertical angle;
%
sn = vprop*p;
ind=find(sn>1);
sn(ind)=nan*ones (size(ind));
cs=sqrt(1-sn.*sn);
vprop=vprop*ones(1,length(p));%make a matrix
thk=abs(diff(zprop))*ones(1,length(p));
%x=x1*ones(length(zprop),length(p));
x=ones(length(zprop),1)*(x1(:)');
x(2:length(zprop),:)=(thk.*sn)./cs;
x=cumsum(x);
zr=zprop*ones(1,length(p));

if(d3==1)
 zz=ones(size(zr));
else
 zz=zeros(size(zr));
end
hray=line(x,zr,zz,'color',kol,'linestyle',lns,'linewidth',lw);
xray=x(end,:);
```

//

```
function [i2]=surround(xx,x)
i=length(xx);
for k=1:i,v(k)=xx(k),end
v(i+1)=x;
MAX=min(v);
i1=round(MAX);
i2=abs(i1)+1;
```