TOMOGRAFIA SISMICA PARA OBTENER MODELOS DE VELOCIDAD DEL ESTRATO SOMERO APLICADA A DATOS SINTETICOS Y REALES

CARLOS ANDRES BECERRA BAYONA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICO MECANICAS
ESCUELA DE INGENIERIA CIVIL
BUCARAMANGA - COLOMBIA
2008

TOMOGRAFIA SISMICA PARA OBTENER MODELOS DE VELOCIDAD DEL ESTRATO SOMERO APLICADA A DATOS SINTETICOS Y REALES

AUTOR
CARLOS ANDRES BECERRA BAYONA

Proyecto de grado presentado como requisito para optar
al titulo de Ingeniero Civil
Modalidad: tesis de investigación

DIRECTORES
Ph.D. WILLIAM MAURICIO AGUDELO ZAMBRANO (ICP)
Msc. SAUL ERNESTO GUEVARA OCHOA (ICP)
Ph.D. ESPERANZA MALDONADO (UIS)

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICO MECANICAS
ESCUELA DE INGENIERIA CIVIL
BUCARAMANGA - COLOMBIA
2008

a mi familia

a mis padres Ana María y Ramón

a mis primos

a María T.

AGRADECIMIENTOS

El autor expresa su agradecimiento a:

WILLIAM AGUDELO Y SAUL GUEVARA por sus valiosas ideas y constante asesoría para llevar a cabo exitosamente esta investigación.

HERLING GONZALEZ Y JORGE MONSEGNI por las enseñanzas y recomendaciones sobre C++.

ANDRES CALLE por la asesoría sobre procesamiento sísmico.

PETROSISMICA - Grupo de Investigación, por abrir las puertas del mundo de la geofísica.

ECOPETROL - ICP por facilitar los medios y datos sísmicos usados en este caso de estudio.

SEG FOUNDATION por el reconocimiento y apoyo económico para realizar de esta investigación.

TABLA DE CONTENIDO

Pags.

INTRODUCCION 13
1. INFLUENCIA DEL ESTRATO SOMERO EN LA IMAGEN SISMICA . 15
1.1. EL ESTRATO SOMERO 16
1.1.1 METEORIZACION 16
1.1.1.1 METEORIZACIÓN QUIMICA 17
1.1.1.2 METEORIZACIÓN FISICA 18
1.1.2 TOPOGRAFIA 18
1.1.3 CASO DE ESTUDIO 18
1.2. SOLUCIONES AL PROBLEMA DEL ESTRATO SOMERO 20
1.2.1 CORRECCIONES ESTATICAS 21
1.2.2 MODELOS DE VELOCIDAD DEL ESTRATO SOMERO 22
1.2.2.1 MODELOS UNI-CAPA 22
1.2.2.2 MODELOS MULTI-CAPA 23
1.2.2.3 MODELOS PLUS/MINUS 24
1.2.2.4 MODELOS DE REFRACCION 25
1.2.2.5 MODELOS TOMOGRAFICOS 25
2. MODELOS SINTETICOS 26
2.1. GENERARACION DE DATOS SINTETICOS 27
2.1.1 SET BASE 27
2.1.1.1 CODIGO BASE 28
2.1.1.2 EXPLICACION DEL CODIGO BASE 30
2.1.2 SET SINTETICO CON OFFSET VARIABLE 39
3. TECNICAS DE INVERSION TOMOGRAFICA 46
3.1. TOMOGRAFIA SISMICA POR RAYOS 47
3.1.1. METODOS DE TRANSFORMACION 47
3.1.2. METODOS DE EXPANSION EN SERIES 54
3.1.2.1 PROBLEMA DIRECTO 54
3.1.2.2 DISCRETIZACION DEL MODELO 55
3.1.2.3 ALGORITMO DE KACZMARZ 57
3.1.2.4 INVERSION ART 63
3.1.2.5 INVERSION SIRT 64
3.1.2.6 INVERSION LSQR 65
4. TECNICAS DE ANALISIS DE INCERTIDUMBRE 67
4.1 JACKKNIFING 68
4.2 CHECKERBOARD TEST 68

4.3 MONTE CARLO 69
4.4 BOOTSTRAPPING 69
5. ANALISIS E INTERPRETACION DE RESULTADOS 71
5.1 PICADO DE PRIMEROS ARRIBOS 72
5.1.1 SET SINTETICO 73
5.1.2 SET REAL 74
5.2 INVERSION TOMOGRAFICA 75
5.2.1 CONTRUCCION DEL MODELO INICIAL 75
5.2.2 CONTRUCCION DE LA GRILLA 77
5.2.3 PROBLEMA DIRECTO 78
5.2.4 PROBLEMA INVERSO 79
5.3 ANALISIS JACKKNIFING 81
5.4 ANALISIS CHECKERBOARD 85
5.5 ANALISIS MONTECARLO 87
5.6 ANALISIS BOOTSTRAPPING 90
5.7 CORRECCION ESTATICA 94
6. CONCLUSIONES 96
BIBLIOGRAFIA 97
ANEXOS 101

LISTA DE FIGURAS

Pags.

Figura 1. (a) Ubicación general de la zona de estudio (b) ubicación de la linea Sísmica 1040 19
Figura 2. Trasecto idealizado (Tomado de Suelos de Colombia, Strakov, 1967, citado por Birkeland) 20
Figura 3. Diagrama esquemático para ilustrar los componentes de las correcciones estáticas (a) Modelo de estrato somero; (b) tiempo de corrección de la superficie a la base de la capa meteorizada; (c) tiempo de corrección de la base de la capa meteorizada a el datum de referencia (Tomado de Mike Cox, S*tatics Corretions for Seismic Refletions Survey,* 1999) 21
Figura 4. Ejemplo de un conjunto de disparo 2D con 3 capas interpretadas (superior) y correspondiente modelo multicapa, las capas definen la base la capa submeteorizada (inferior) 24
Figura 5. Diagrama de rayo reciproco, relación entre la localización de un conjunto de fuentes y el spread 24
Figura 6. (a) Modelo de estrato somero; (b) vista de fuente y algunos receptores para el primer disparo del set de datos 1; (c) trazado de rayos para el primer disparo del set de datos 1 28
Figura 7. Sismogramas sintéticos obtenidos con diferentes tipos de ondas (a) Primarias (b) Headwaves (c) Headwaves y primarias 35
Figura 8. Conjunto de trazas para un segmento de disparo, para el Set de Datos 1 37
Figura 9. Vista del disparo 1 en escala de Grises- Set de Datos Sintéticos 1 38
Figura 10. Representación esquemática de la geometría de la adquisición. Tomado del Reporte del Observador 40
Figura 11. Modelo - set de datos sintético con offset variable 42
Figura 12. Conjunto de disparos - Set de datos sintético con offset variable 43
Figura 13. Representación esquemática de un medio 48
Figura 14. Trazado de rayo sobre un medio (Ø=0˚) 49
Figura 15. Trazado de rayo sobre un medio (Ø=45˚) 50
Figura 16. Representación del teorema de la sección proyección. Tomado de Fitterman 1994 53
Figura 17. Diagrama de flujo del algoritmo de Kaczmar´z. Tomado de Fitterman 1994 58
Figura 18. Ejemplo de la aplicación del algoritmo de Kaczmar´z en un modelo de dos celdas con dos rayos que las atraviesan. Tomado de Fitterman 1994 60
Figura 19. Convergencia del algoritmo de Kaczmar´z en un modelo de dos celdas con dos rayos que las atraviesan. Tomado de Fitterman 1994 63
Figura 20. Convergencia del algoritmo de Kaczmar´z en un modelo de dos celdas con dos rayos que las atraviesan. Tomado de Fitterman 1994 65

Figura 21. Representación esquemática del método Bootstrapping 70
Figura 22. Picado de primeros arribos para el set de datos sintético (a) Vista del disparo 1 del set (b) Definición del rango para hacer el picado de primeros arribos (c) Vista de los disparos 1, 45, 110 y 156 del set de datos sintético 73
Figura 23. Vista del disparo 240 del set de datos real 74
Figura 24. Picado de primeros arribos para el set de datos real, Vista de algunos disparos picados del set de datos real 74
Figura 25. Primeros Arribos – Set de datos real 75
Figura 26. Gráfica de tiempo vs offset – Set de datos Reales 76
Figura 27. Curva de tiempo de viaje para ondas primarias, headwaves y reflejadas, Tomado de Thorne, 1995 76
Figura 28. Vista de algunos disparos sobre el modelo inicial, linea 1040-Uribante 78
Figura 29. Inversión del set de datos sintético (a) Modelo sintético (b) Modelo inicial (c) Modelo solución 80
Figura 30. Inversión del set de datos real (Linea1040 – Programa sísmico Uribante 2005) 81
Figura 31. Análisis Jacknifing – Set Sintético 83
Figura 32. Análisis Jacknifing – Linea 1040 83
Figura 33. Varias realizaciones Jacknifing con diferente número de datos eliminados (a) shift=10 (b) shift=16 (c) shift=20 (d) shift=50 84
Figura 34. Implementación Checkerboard test sobre modelo sintético (a) Patrón de anomalías (b) modelo perturbado (c) modelo perturbado invertido (d) diferencias entre modelos 86
Figura 35. Implementación Checkerboard test sobre modelo de la Linea 1040 (a) 20x20 (b) 100x100 (c) 150x150 (d) 200x200 87
Figura 36. Implementación Monte Carlo sobre modelo sintético (a) Modelo verdadero (b) Modelos Promedio Monte Carlo 88
Figura 37. Desviación estándar Implementación Monte Carlo - modelo sintético 88
Figura 38. Implementación Monte Carlo sobre modelo de la Linea 1040 (a) Modelo Solución sin perturbaciones en el modelo inicial (b) Modelo Promedio Monte Carlo (c) desviación estándar 90
Figura 39. Implementación Bootstrap sobre modelo sintético (a) Modelo Verdadero (b) Modelo promedio de la 50 realizaciones Bootstrap (b) Desviación estándar 91
Figura 40. Implementación Bootstrapping sobre modelo de la Linea 1040 (a) Solución original (b) Media de las 50 realizaciones Bootstrap (c) Desviación estándar 92
Figura 41. Resultados de técnicas de análisis de incertidumbre sobre modelo de la Linea 1040 (a) Jackknifing (b) Checkerboard (c) Monte Carlo (d) Bootstrapping 93
Figura 42**.** Correcciones estáticas calculadas para el modelo de la Linea 1040 95
Figura 43. Imágenes sísmicas corregidas- segmento de la Linea 1040 – Uribante 2005 (izquierda) Aplicando estáticas de refracción (derecha) Aplicando estáticas obtenidas del modelo tomografico 95

LISTA DE ANEXOS

Pags.

Anexo 1. Código para generar set de datos sintético 101
Anexo 2. Código para realizar análisis Jackknifing 106
Anexo 3. Código para realizar análisis Checkerboard 141
Anexo 4. Código para realizar análisis Monte Carlo 149
Anexo 5. Código para realizar análisis Bootstrapping 166
Anexo 6. Código para calcular correcciones estáticas 182

Title: SEISMIC TOMOGRAPHY FOR TO OBTAIN VELOCITY MODELS OF THE NEAR SURFACE APPLIED TO SYNTHETIC AND REAL DATA*

Author: BECERRA B. CARLOS A.**

Key Words: Subsurface, inversion, uncertainty, statics, seismic.

SUMMARY

The subsurface has importance in some science field; particularly for civil engineering is there where we puts builds, ways and bridges; also there the strong of the earthquakes are transmitted to our structures. This work uses the information of first arrival time for obtain velocity models using a seismic tomography inversion technique. After, the quality of results is evaluated performing uncertainty analysis methodologies such us: (1) Jackknife, in order to identify more sensibility zones to less data in the inversion (2) Checkerboard test, for identify well resolve zones (3) Monte Carlo, for estimate uncertainty associated with initial velocity model (4) Bootstrap, for estimate uncertainty associated with first arrive traveltime. I had applied these techniques in a real data set acquired in a structurally complex zone in the Catatumbo basin (Colombia). The final velocity model shows that the velocity of the weathering layer is between 0.8 and 1.5 Km/s, exhibits variations, the thickness can vary from near to 50 meters in zones with low slope to few meters in zones with high slope. I use the velocity model resultant for calculate the weathering and elevation correction, the qualify of seismic image for the segment of line studied is better when the correction has been applied.

---

*Graduate Thesis, Modality Research.

**Faculty of Physic - Mechanics Engineering, Civil Engineering Department. Director Engineer PhD. William Agudelo; Director Engineer MSc. Saul Guevara; Director Engineer PhD. Esperanza Maldonado.

Titulo: TOMOGRAFIA SISMICA PARA OBTENER MODELOS DE VELOCIDAD DEL ESTRATO SOMERO APLICADA A DATOS SINTETICOS Y REALES*

Autor: BECERRA B. CARLOS A.**

Palabras Claves: Subsuelo, Inversión, incertidumbre, estáticas, sísmica

RESUMEN

El subsuelo tiene importancia en muchos campos de la ciencia, en particular, para la ingeniería civil, es donde se cimentan edificaciones, vías y puentes; además es donde la fuerza de los sismos es transmitida a las estructuras. Por tanto, es fundamental caracterizar sus propiedades mecánicas. En este trabajo se usa información de tiempos de primeros arribos para obtener modelos de velocidad usando una técnica de inversión de tomografía sísmica. Después, la calidad del modelo de velocidad invertido es evaluado usando metodologías de análisis de incertidumbre tales como (1) Jackknife, para identificar zonas más sensibles del modelo a los datos faltantes en la inversión, (2) Prueba checkerboard, para identificar zonas mejor resueltas del modelo (3) Monte Carlo, para determinar la incertidumbre asociada a variaciones en el modelo inicial (4) Bootstrap, para estimar la incertidumbre asociada a los tiempos de viaje. Se han aplicado estas técnicas a un set de datos real adquirido en una zona compleja estructuralmente, en la cuenca del Catatumbo (Colombia). El modelo de velocidad final muestra que la velocidad de la capa meteorizada esta entre 0.8 Km/s a 1.5 Km/s, exhibe variaciones, el espesor puede variar de unos 50 metros en zonas con menor pendiente a unos pocos metros en zonas de alta pendiente. El modelo de velocidad resultante se usa para calcular corrección por capa meteorizada y por elevación, la calidad de la imagen imagen sísmica para el segmento de linea estudiado mejora al aplicar las correcciones calculadas.

---

*Proyecto de Grado, Modalidad Tesis de Investigación.

**Facultad de Ingenierías Físico – Mecánicas, Escuela de Ingeniería Civil. Director Ingeniero Ph.D. William Agudelo; Director Ingeniero MSc. Saul Guevara; Director Ingeniero Ph.D. Esperanza Maldonado.

INTRODUCCION

El estudio del subsuelo es útil en diferentes campos de la ciencia, en ingeniería civil conocer las características del subsuelo sobre el cual se cimentan los diferentes tipos de estructuras puede dar información importante para su adecuado diseño; en sismología es útil para conocer el medio por el cual se propagan las ondas de los sismos y determinar a que profundidad estos se han originado, en geofísica puede usarse para mejorar la imagen sísmica y reducir el riesgo exploratorio.

Este trabajo presenta el uso de la tomografía como una herramienta para construir modelos de velocidad del subsuelo a partir de datos de tiempos de viaje. Cada tipo de suelo tiene una velocidad de propagación de onda P característica (Cox, 1999), los tiempos de viaje se adquieren usando el método sísmico, que consiste en ubicar sobre la superficie del subsuelo una serie de receptores o geófonos que registran la señal que regresa a la superficie producto de generar artificialmente ondas sísmicas en otro punto del terreno, generalmente estas ondas artificiales se producen haciendo explotar cierta carga de dinamita a cierta profundidad bajo la superficie, esto en función del tipo de suelo y la profundidad del subsuelo que desea ser estudiada, el punto donde se genera la explosión se llama fuente (*source*). Los receptores suelen ubicarse sobre la superficie de terreno alineados y separados uno de otro entre 15 y 30 metros, la fuente se ubica en diversos puntos de la linea, el conjunto de señales registradas por todos los receptores cada vez que se generan ondas en la fuente es llamado disparo, la configuración geométrica de fuentes y receptores también determina la información que puede obtenerse del medio estudiado.

Para usar la técnica de tomografía se usa un set de datos sintético y un set de datos real, producto de una adquisición sísmica realizada en la cuenca del Catatumbo (Norte de Santander – Colombia) en el año 2005. Los datos sintéticos son generados (capítulo 2) teniendo en cuenta la geometría de la adquisición del set de datos real. Tomografía sísmica es una técnica de inversión iterativa, el desarrollo matemático es presentado en el capítulo 3 y la aplicación a los set de datos se discuten en el capítulo 5, el modelo solución resultante suele tener una incertidumbre asociada, para evaluar los posibles errores se usan técnicas de análisis de incertidumbre, estos se presentan en el capítulo 4, su aplicación y resultados se discuten en el capítulo 5. Los modelos de velocidad

resultantes pueden tener diferentes utilidades, en esta investigación se usan para mejorar la calidad de la imagen sísmica del subsuelo, a través del calculo de correcciones estáticas.

capítulo 1

INFLUENCIA DEL ESTRATO SOMERO EN LA IMAGEN SISMICA

Introducción

Se llama estrato somero a las primeras capas del suelo que suelen ser afectadas por diferentes agentes ambientales y se encuentran generalmente meteorizadas, por tanto la velocidad de propagación en estos estratos es baja, afectando el correcto posicionamiento de reflectores o refractores profundos en las imágenes sísmicas. Los efectos de meteorizaron son diferentes en cada sitio, en general los factores más importantes son: el clima, el tipo de suelo y la vegetación. En este capitulo se presentan los diferentes tipos de meteorización; así como la zona de estudio y las condiciones particulares a las que esta sometida; luego, se presenta el concepto de correcciones estáticas, una herramienta útil para mitigar el efecto del estrato somero en la imagen sísmica; por ultimo, se presenta una descripción de las metodologías disponibles para modelar el estrato somero.

1.1 EL ESTRATO SOMERO

El estrato somero tiene diferente significado según el enfoque sísmico o geológico: Sheriff (1991) desde el punto de vista sísmico lo define como: una capa de baja velocidad cercana a la superficie, donde usualmente el aire en vez del agua ocupa los espacios porosos de la roca o material no-consolidado. La capa meteorizada sísmica usualmente es diferente de la capa meteorizada geológica (el resultado de descomposición de las rocas). La diferencia entre ambos enfoques radica en que, desde el punto de vista de la sísmica el estrato somero es la capa hasta donde las velocidades son significativamente menores que las de las siguientes capas; desde el enfoque geológico el estrato somero es la capa hasta donde tiene lugar la descomposición de la roca. La capa meteorizada sísmica suele tener mayor espesor que la definida para los geólogos.

La velocidad de la capa meteorizada esta típicamente entre 500 m/s y 800 m/s (aunque puede ser 150 m/s para los primeros centímetros) que comparados con las velocidades de la capa submeteorizada que son 1500 m/s o mas muestran que se trata de una capa de baja velocidad o LVL (Low Velocity layer). Frecuentemente la base de la capa meteorizada sísmica es el nivel freático. Algunas veces la velocidad de la capa meteorizada tiene un gradiente, algunas veces esta conformada claramente por capas. La velocidad de la capa meteorizada puede calcularse usando datos de estudios pozos (uphole) y de primeros arribos de refracción.

Desde el enfoque geológico la capa meteorizada está constituída por los estratos donde se produce la descomposición o desintegración de las rocas. El proceso de meteorización puede ser físico, químico o ambos; el primero prepara a las rocas para que luego sean erosionadas y transportadas por el viento y agua. El proceso trabaja desde la superficie, donde tiene mayor efecto, esto es especialmente verdadero para sedimentos no consolidados tales como arena y arcillas. En la siguiente sección se presentan más detalles del concepto de meteorización.

1.1.1 METEORIZACION

La meteorización es el proceso que desintegra y descompone las rocas como consecuencia de procesos químicos y físicos. El efecto de este proceso sobre el suelo depende de las condiciones climáticas y del tipo de suelo. El proceso de meteorización requiere tiempo y tiene varias etapas: en general al inicio del proceso de

meteorización, los feldespatos se descomponen en caolinita, hierro hidratado y óxidos de aluminio, aunque las partículas de mica y cuarzo que son más resistentes permanecen casi inalteradas. El proceso de meteorización continúa y los contenidos de caolinitas disminuyen, alterándose los demás compuestos a otros. Mohr (1944) dividió el proceso en 5 etapas: (1) Etapa Inicial: la roca se encuentra inalterada, (2) Etapa Juvenil: es el inicio de la desintegración, pero la mayoría del material esta aun intacto, (3) Etapa Viril: la descomposición continua y aumenta el nivel de arcilla y suelo, (4) Etapa Senil: es la etapa donde tiene lugar la oxidación, (5) Etapa Final: el suelo se ha meteorizado por completo.

1.1.1.1 METEORIZACIÓN QUIMICA

La meteorización química o descomposición, involucra la acción del agua y ácidos débiles para la rotura de las rocas por medio de reacciones químicas. Unos pocos minerales son solubles en agua, tales como rocas salinas y otras aceptadoras de agua, tales como la anhidrita. Óxidos tales como el oxido de hierro, son formados por la acción del agua y el oxigeno, cuando estos son expuestos a la atmósfera se oxidan. Muchos minerales son solubles en ácidos débiles como el ácido carbónico, formados por agua y dióxido de carbono, que actúan sobre las rocas con un efecto de abrasión para producir superficies meteorizadas. La descomposición trabaja afectando los enlaces químicos a nivel de los minerales y eliminando los agentes que cementan la roca; esta acción altera la roca total o parcialmente, debilita la estructura o cambia el volumen, por un incremento de tamaño o por formación de cavidades.

La meteorización química está relacionada con el flujo de agua a través del suelo (lixiviación). Los procesos químicos involucrados en la alteración son principalmente: (1) La oxidación: que se produce cuando el oxigeno del aire entra en contacto con ciertos componentes químicos-mineralógicos de las rocas. (2) La hidrólisis: que se produce cuando el agua tiene contacto con minerales de silicato, formandose minerales arcillosos y residuos metálicos arenosos. (3) La hidratación: que es el proceso por el cual las moleculas de agua son aderidhas al mineral para formar un nuevo mineral. (4) La solución: que consiste en la disociación de las moléculas en iones por la acción de un agente disolvente como el agua, cuando el agente desaparece los materiales disueltos se precipitan (Mohr 1944).

La velocidad con que ocurren los procesos químicos de meteorización es función del clima (a mayor humedad mayor velocidad), de la composición mineralógica y del

tamaño de grano de la roca.

1.1.1.2 METEORIZACION FISICA

La meteorización física o mecánica incluye procesos como la expansión térmica, el crecimiento cristalino y la acción de organismos vivos. En el proceso, la composición de las rocas no se altera. Generalmente, la conducción de calor en el suelo es pobre, por tanto los cambios de temperatura producen expansiones y contracciones que causan granulación o laminación. Los procesos físicos involucrados principalmente son: (1) Expansión y contracción térmica: que destruye las rocas por la acción de ciclos que producen cambios volumétricos. (2) Acción por heladas: Ciclos de congelamiento y descongelamiento que produce tensiones que terminaran fragmentando la roca. (3) Exfoliación: Desprendimiento de la roca en láminas. (4) Acción de raíces: Las raíces que crecen en las grietas de las rocas generan esfuerzos de tracción. (5) Crecimiento cristalino: este es un aumento en el volumen que se producen en los cristales salinos.

En ambientes tropicales húmedos la meteorización química predomina, cuando existe vegetación la tasa de meteorización es mayor; en cambio, en ambientes tropicales secos la meteorización mecánica o física es la que predominan y suele ser un proceso más lento, al igual que en zonas montañosas con altas pendientes y poca vegetación; aunque generalmente los suelos son meteorizados por combinación de ambos fenómenos.

1.1.2 TOPOGRAFIA

Otro factor importante para tener en cuenta es la topografía de la superficie, que tiene efecto sobre la formación del estrato somero, los sistemas de drenaje que canalizan las aguas que se precipitan , los cambios de temperatura y clima con la altitud y las pendientes influyen en el contenido de materia orgánica del suelo. Desde el punto de vista del procesamiento de datos sísmicos la topografía juega un rol importante, debido a que siempre es necesario llevar todas las fuentes y receptores a un datum, que es simplemente un plano de referencia, detalles acerca de este procedimiento son presentados en la sección 1.2.1

1.1.3 CASO DE ESTUDIO

La zona de estudio, esta ubicada en la cuenca del Catatumbo, Norte de Santander. Para

aplicar las metodologías de los capítulos 3 y 4 se usó un segmento de la linea 1040 del programa sísmico Uribante 2005. La ubicación general de la zona y la ubicación de la linea se muestran el la figura 1.

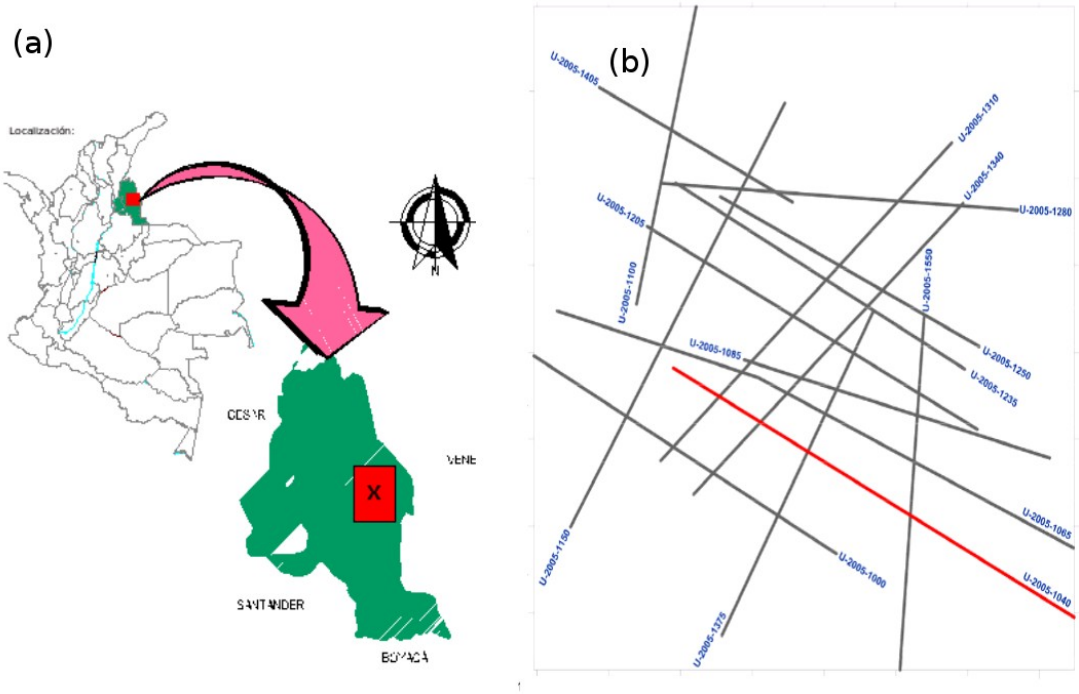


Figura1. (a) Ubicación general de la zona de estudio (b) ubicación de la linea Sísmica 1040.

A mayor escala, la zona de estudio se encuentra ubicada en la zona ecuatorial o zona tropical donde la energía solar es el factor mas influyente en el clima, temperaturas relativamente altas, regímenes de precipitación con grandes descargas y además abundante vegetación, hacen que el estrato somero tenga condiciones particulares, aquí la meteorización es más intensa, produciendo que el espesor de la capa meteorizada sea posiblemente mayor en comparación con el de otras zonas. En la figura 2 se muestra un trasecto idealizado que muestra las tendencias de precipitación, temperatura, tipo de suelo y vegetación en función de la ubicación geográfica. En cuanto a la topografía de la zona esta puede clasificarse como abrupta.

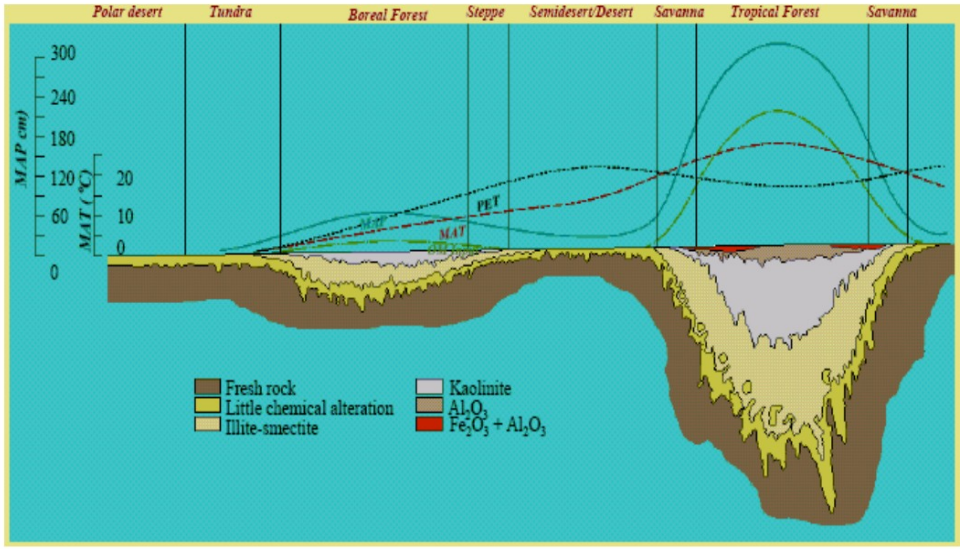


Figura 2. Trasecto idealizado (Tomado de Suelos de Colombia) - (Strakov, 1967, citado por Birkeland)

La información del estrato somero puede obtenerse por varios métodos: sísmica de refracción, sísmica de reflexión somera, estudios de uphole, métodos gravimétricos y magnéticos, métodos eléctricos y GPR (*Ground Penetrating Radar*). En este caso, se cuenta con datos de sísmica de reflección e información de 4 uphole para toda la zona, 1 de ellos ubicado sobre la linea.

1.2 SOLUCIONES PARA EL PROBLEMA DEL ESTRATO SOMERO

En la sección anterior se presentaron los factores más influyentes en la formación del estrato somero y su relación con el ciclo de vida mismo, en muchas campos de la ciencia como la biología, la agronomía, entre otros, el estrato somero es visto como objeto de estudio y fundamental; esta sección en cambio, presenta el estrato somero como "un problema", esto se debe, a que este tiene un efecto negativo en la calidad de las imágenes sísmicas; por esto, se considera un obstáculo para "mapear" el subsuelo. La calidad de la imagen sísmica podría mejorar si el "obstáculo" es eliminado (virtualmente–solo en los datos), así que un mayor conocimiento y mejor modelado del estrato somero puede resultar en una mejor imagen corregida. El estado de arte de metodologías disponibles para modelar la capa somera se presentan en la sección 1.2.2. Antes un concepto fundamental para el entendimiento de estas metodologías es presentado en la siguiente sección.

1.2.1 CORRECIONES ESTATICAS

Sheriff (1991) define correcciones estáticas como: correcciones aplicadas a datos sísmicos para compensar los efectos de variación en elevación, espesor y velocidad del estrato somero, o referencia a un datum. El objetivo es determinar los tiempos de reflexión que podrían haber sido observados si todas la medidas hubiesen sido realizadas sobre un plano con pendiente cero, sin presencia de material de baja velocidad (estrato somero). Estas correcciones se basan en datos de uphole, primeros arribos de refracción y/o eventos suavizados.

Las correcciones estáticas de datum (*datum statics correction*) según Mike Cox (1999) pueden dividirse en dos componentes: (1) corrección de la capa meteorizada (*weathering correction*) que se usa para remover el efecto de las capas someras y (2) corrección por elevación (*elevation correction*) que sirve para llevar las fuentes y receptores a un datum.

Para ilustrar cómo las correcciones estáticas son aplicadas en la práctica, la figura 3 muestra un diagrama esquemático tomado del libro de Mike Cox (*Statics Corrections for Seismic Reflections Survey*), con un modelo simple para el estrato somero que se muestra en la figura 3a. Los puntos A, B y C están sobre la superficie, que ha sido representada por una linea continua; los puntos A_b, B_b y C_b están sobre la base de la capa meteorizada, que es representada por la línea punteada; de igual manera los puntos A_d, B_d y C_d se ubican sobre en el datum.

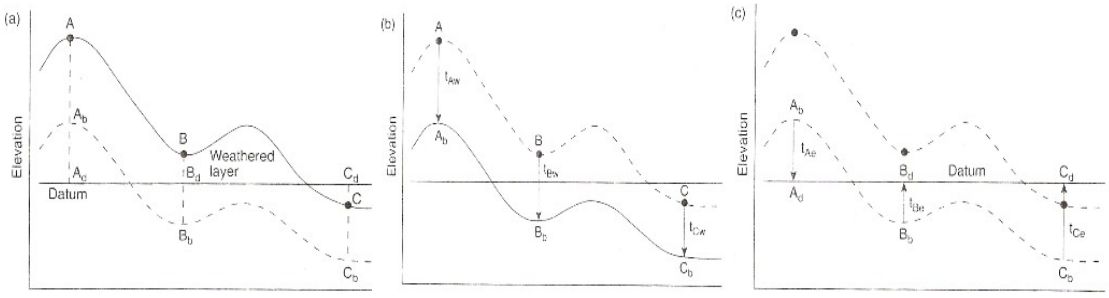


Figura 3. Diagrama esquemático para ilustrar los componentes de las correcciones estáticas (a) Modelo de estrato somero; (b) tiempo de corrección de la superficie a la base de la capa meteorizada; (c) tiempo de corrección de la base de la capa meteorizada a el datum de referencia (Tomado de Mike Cox, S*tatics Corretions for Seismic Refletions Survey,* 1999)

Hacer las correcciones estáticas sobre los puntos A, B y C, consiste en encontrar los tiempos de viaje que hubiesen sido registrados si estos puntos se encontraran ubicados sobre un plano con pendiente cero, es decir, en los puntos A_d B_d y C_d, según lo definido por Sheriff. El primer paso es eliminar el efecto de la capa meteorizada, tal que la base de la capa meteorizada sea ahora la nueva superficie de referencia, para esto se deben calcular los tiempos t_{Aw}, t_{Bw} y t_{Cw}, la nueva superficie de referencia se muestra en la figura 3b como una linea continua, esta corrección es frecuentemente llamada corrección de la capa meteorizada. El siguiente paso es ajustar los datos para que simulen haber sido registrados desde otra superficie de referencia, en este caso el datum de elevación. Ahora como se ve en la figura 3c, deben ser calculados los tiempos de viaje t_{Ae}, t_{Be} y t_{Ce}, la nueva superficie de referencia es la linea continua que representa el datum, esta corrección es frecuentemente llamada corrección por elevación.

Las correcciones estáticas de datum son entonces la suma de la corrección por capa meteorizada y la corrección por elevación. Nótese en la figura 3c que en algunos puntos, los tiempos de viaje deben ser restados y en otros sumados; por tanto, es importante adoptar una convención de signos, para este caso será la definida por Cox: Una corrección estática negativa reduce los tiempos de viaje, es decir se usa cuando el datum esta por debajo de la topografía y por debajo de la capa meteorizada; una corrección estática positiva aumenta los tiempos de viaje y se usa cuando el datum esta sobre la topografía. Para el ejemplo presentado en la figura 3, la corrección estática total para los puntos A, B y C puede calcularse como:

$$T_A = -t_{Aw} - t_{Ae}; \qquad T_B = -t_{Bw} + t_{Be}; \qquad T_C = -t_{Cw} + t_{Ce}$$

El computo de correcciones estáticas requiere información de elevación de fuentes y receptores, esta puede ser obtenida de la geometría de la adquisición; así como el espesor y velocidades del estrato somero, estas son calculadas usando el procedimiento descrito en el capitulo 3 y son el principal objetivo de esta tesis. La generación de un modelo del estrato somero preciso es fundamental para obtener estáticas que realmente mejoren las imágenes sísmicas, en la siguiente sección se presenta las metodologías disponibles para obtener un modelo de velocidad del estrato somero.

1.2.2 MODELOS DE VELOCIDAD DEL ESTRATO SOMERO

1.2.2.1 MODELOS UNI-CAPA

Estos modelos usan solo información de upholes, que son pozos perforados aproximadamente 60 metros, luego puede obtenerse un modelo de velocidad 1D para capa pozo; el costo de la perforación de capa pozo es alto y como la separación entre los pozos es normalmente grande las velocidades entre ellos

deben interpolarse con gran imprecisión. Estos modelos de velocidad uni-capa resultan insuficientes cuando el muestreo espacial de la variación lateral del campo de velocidad no es adecuado. La estática resultante podría ser demasiado grande donde hay una velocidad del estrato somero menor, por ejemplo sobre un depósito de material no consolidado, y resultar demasiado pequeña sobre un afloramiento rocoso.

1.2.2.2 MODELOS MULTI-CAPA

Estos son modelos 2D que pueden ser construidos a partir de la interpretación de primeros arribos de conjuntos de disparos (*shot gathers*), aunque solo se utiliza un lado del spread (patrón geométrico de grupos de disparos relativo a la fuente sísmica). Esto se hace usando el método de refracción clásico (Cox 1999), de manera que los primeros tres horizontes pueden ser interpretados como lo muestra la figura 4 (superior) , en áreas con pendiente cero los arribos directos y refractados son picados, luego usando solo los disparos buenos, el espesor y las velocidades de varias capas pueden ser calculados de sus pendientes y tiempos interceptos (Bridle *et al*. 2003).

La figura 4 (inferior) es un modelo multi-capa del estrato somero creado usando el modelo de velocidad/profundidad derivado, este fue construido por un equipo de investigadores de Saudi Aramco (Bridle, Barsoukov, Al-Homaili, Ley, Al-Mustafa) usando lineas 2D, como parte de un proyecto de investigación sobre el estrato somero en Arabia Saudita en el año 2006. .

Luego usando el modelo multicapa derivado, debe aplicarse estáticas de datum. Aplicando este método generalmente se obtiene una mejor imagen que cuando se usa un modelo de 1 sola capa; sin embargo, este puede presentar problemas con estáticas de grandes amplitudes y longitud de onda corta, porque estas son bastante grandes y causan ciclos de saltos, adversamente afectando las estáticas residuales en sectores del modelo con topografia sin pendiente (Bridle Barsoukov, Al-Homaili, Ley and Al-Mustafa, 2006).

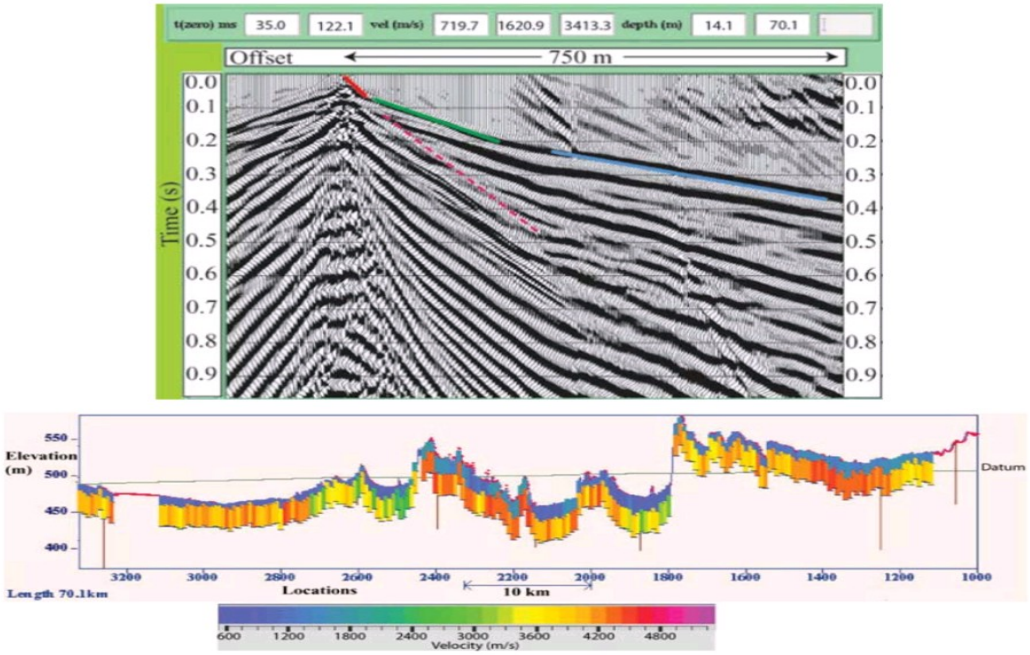


Figura 4. Ejemplo de un conjunto de disparo 2D con 3 capas interpretadas (superior) y correspondiente modelo multicapa, las capas definen la base la capa submeteorizada (inferior).

1.2.2.3 MODELOS PLUS/MINUS

Cuando existen cambios rápidos de velocidad a través de la linea, los modelos multicapa que solo usan solo un lado del spread, suelen no definir con precisión el modelo, pues en estas condiciones cada lado del spread tiene diferentes características. Plus/Minus usa entonces el método de camino de rayo reciproco, el primer reflector sobre la parte alta del spread es picado y los receptores con la misma localización son también picados en la parte baja de la localización de la segunda fuente. La figura 5 muestra un diagrama de caminos de rayos recíprocos. Usando una relación aritmética simple (Hagedoorn 1959; Cox 1999) la velocidad de refracción y el tiempo de retraso pueden ser calculados.

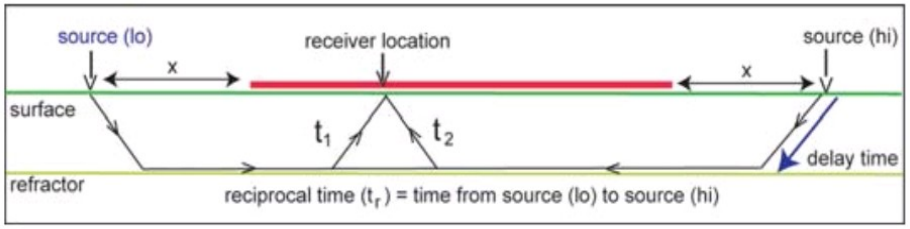


Figura 5. Diagrama de rayo reciproco, relación entre la localización de dos fuentes y el spread

Pares de conjuntos de fuente son picados para construir los mini-spread, cada localización de receptor puede tener muchos tiempos de retraso (*delay times*) y estos deben ser promediados. Los tiempos de retraso son convertidos a tiempos verticales y estos representan el tiempo en la capa más somera. Con los tiempos de retraso puede construirse un modelo de velocidad.

1.2.2.4 MODELOS DE REFRACCION

Existe una serie de métodos que se han desarrollado usando tiempos de primeros arribos refractados, el más conocido y usado es el método de Inversión Lineal Generalizado (*Generalized Linear Inversion method*) desarrollado por Hampson and Russell, 1984. La aplicación de este método a áreas con pendientes grandes y variables produce buenos resultados. Sin embargo, los métodos de refracción tradicionales tienen algunas limitaciones en la construcción del modelo de velocidad: por ejemplo las estructuras del estrato somero deben ser representadas como un modelo a capas y los tiempos de primeros arribos como refracciones en las interfaces del modelo. Como consecuencias de estas simplificaciones los métodos de refracción no pueden modelar gradientes de velocidad verticales en cada capa, y puede tener problemas cuando existen fuertes variaciones laterales.

1.2.2.5 MODELOS TOMOGRAFICOS

Este método usa la información de primeros arribos refractados, el trazado de rayos de refracción tomografíca genera correcciones estáticas para longitudes de onda larga y media. Una de las ventajas de esta técnica es que puede ser usada en áreas donde la velocidades del estrato somero tenga variaciones rápidas, otra ventaja es que el modelo de velocidades del estrato somero resultante esta en el dominio de la profundidad, además este puede usarse como entrada para hacer migración preapilado en profundidad. Una descripción detallada de este método puede encontrarse en el Capitulo 3.

capítulo 2

DATOS SISMICOS SINTÉTICOS

Introducción

El método sísmico busca obtener información de parámetros del subsuelo a través de mediciones realizadas desde la superficie, estas mediciones son la respuesta del medio. La geometría de las mediciones o geometría de la adquisición sísmica, puede restringir la información que se obtiene del medio; por tanto, es necesario construir un set de datos sintéticos que simule la geometría real de la adquisición . Como en los modelos sintéticos se conoce el medio (a diferencia del caso real) estos sirven para determinar la efectividad del método de inversión [Capitulo 3]. Luego de invertir un set de datos real, la exactitud de la solución no se conoce, por tanto, se implementan una serie de técnicas de análisis de incertidumbre [Capitulo 4] y los datos sintéticos generados sirven para evaluar el funcionamiento de estas metodologías.

El código creado, que usa algunas herramientas de *Seismic Unix*, permite generar datos sintéticos según las necesidades de esta investigación. Se inicia con un modelo sencillo (sección 2.1.1) que busca presentar el código (sección 2.1.1.1) y explicar su funcionamiento (sección 2.1.1.2), luego modificaciones simples en el cogido original (sección 2.1.2) permiten aproximarse a una mejor simulación del mundo real con modelos más completos, estas modificaciones serán comentadas en el desarrollo de este capitulo.

2.1 GENERACION DE DATOS SINTETICOS

Para generar información sísmica sintética simulando diversas geometrías particulares de adquisición sísmica y variación en propiedades físicas como espesor y velocidad, es necesario crear un código que permita variar fácilmente dichas características. Por tanto, se escribió un código (*shell script)* en Unix, que utiliza la herramienta cshot1.f para hacer el trazado de rayos sobre el modelo y cshot2.f para generar trazas sintéticas, ambos programas están escritos en el lenguaje Fortran, además se usa una serie de aplicaciones (comandos) que hacen parte de la herramienta de uso libre *Seismic Unix* creada por: "*The Center for Wave Phenomena Colorado School of Mines*".

2.1.1 SET BASE

Para empezar se supone un medio isotrópico compuesto por capas horizontales de velocidad constante, como se ilustra en la figura 6a. En la superficie del modelo que es también horizontal se ubican 600 receptores (geófonos-transductores) espaciados 15 metros uno de otro, a partir de la coordenada 30 (estación 1) ver figura 6b. La adquisición sintética se hizo de la siguiente forma: La primera fuente (*source*) se ubica en superficie en la coordenada 150 (estación 9), y todos los transductores registran los eventos de los primeros 4 segundos; las 600 trazas resultantes conforman el primer disparo (*shot*); antes de efectuarse el segundo disparo el geófono ubicado en la primera estación es "retirado" y otro es ubicado una estación más adelante de donde estaba el ultimo geófono en el disparo anterior; lo que indica que la red de geófonos avanza 15 metros (1 estación) en cada disparo. El segundo disparo se realiza a 150 metros (10 estaciones) respecto del primero y así sucesivamente; de esta manera, se cubre sobre la linea una distancia de 2 Km con 66 disparos.

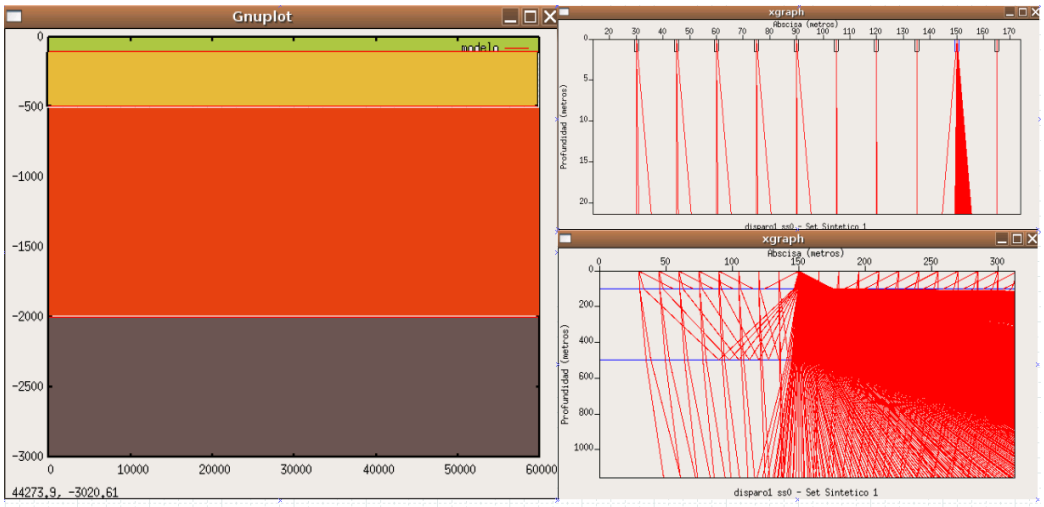


Figura 6. (a) Modelo de estrato somero ; (b) vista de fuente y algunos receptores para el primer disparo del set de datos 1; (c) trazado de rayos para el primer disparo del set de datos 1.

El conjunto completo de trazas generado por todos los disparos es un set de datos sintéticos; que en este caso, tiene número de trazas por disparo constante. Para generar el primer set de datos sísmicos sintéticos se usa el *shell,* que aparece a continuación en la sección 2.1.1.1 y que es explicado paso a paso en la sección 2.1.1.2.

2.1.1.1 CODIGO BASE

Los números de la izquierda, iniciando cada reglón, no son parte del código, se usan solo para describirlo:

```
1   #! /bin/sh
2   # File: eXe
3   # Generacion de Datos Sinteticos usando --> cshot
4   # Abril de 2007
5
6   # Tamaño del modelo
7   x1beg=0 x1end=3000 x2beg=0 x2end=60000
8   # Velocidades entre interfaces
9   vel1=600.0 vel2=2400.0 vel3=4000.0 vel4=5000.0 vel5=5200.0
10  # Numero de trazas por disparo
11  ntzs=600
12  # Numero de disparos
13  nums=67
14  nums1=`expr $nums - 2`
15
16  echo " ***-Empezando proceso-***"
17  k=0
```

```
18      while [ "$k" -ne "6" ]
19      do
20
21              echo " --Cambio de estacion de referencia $k"
22              ref=`expr 666 \* $k`
23              echo " --La estacion referencia es $ref"
24
25              i=1
26              while [ "$i" -ne "$nums" ]
27              do
28                      ro=`expr $i + 1 + $ref`
29                      g1=`expr $ro + 1`
30                      g2=`expr $g1 + 1`
31                      rf=`expr $i + $ntzs + $ref`
32                      s=`expr $i \* 10 + $ref`
33
34                      echo "0    0.\n15.     0.\n$ro $g1 $g2 $rf $s. 0." > geometry0
35
36                      echo    "multi1\n4\nplotcolors\nm\ndummywell\ns\ngeometry0\nsg\nrt\nfichero$i\n-90.
90.\n1.\n$vel1 $vel2 $vel3\n$vel4 $vel5\nn\n1 2 3 4\ny" > param1
37
38                      echo            "s\n1            1\n1            $ntzs\n10.          25.          35.
50.\n.150\n.004\n4.\nfichero${i}shot\nfichero${i}traces" > param2
39
40                      cshot1 |
41                      cshotplot >fichero${i}plot  outpar=fichero${i}par
42
43                      xgraph <fichero${i}plot par=fichero${i}par style=seismic \
44                      -geometry 600x300+650+650 \
45                      title="disparo$i ss$k - Set Sintetico 1" \
46                      label1="Profundidad (metros)"  label2="Abscisa (metros)" \
47                      x1beg=$x1beg x1end=$x1end x2beg=$x2beg x2end=$x2end &
48
49                      cshot2
50
51                      tiro=`expr $i + 66 \* $k`
52
53                      suaddhead <fichero${i}traces ftn=1 ns=1001 |
54                      sushw key=dt a=4000 | sushw key=tracr a=1 b=1 c=0 | sushw key=fldr a=$tiro b=0
c=0|sugain gagc=1 wagc=0.5 >sudata${i}trazas.su
55
56                      suximage title="Headwaves & Primarias disparo$i ss$k" \
57                      ybox=70 \
58                      label1="Time in Seconds"  label2="Trace" <sudata${i}trazas.su
59
60                      echo " **Disparo $i Listo!"
61
62              i=`expr $i + 1`
63              done
```

```
64
65      echo " ***Emsamblando Disparos***"
66      j=1
67      echo "cat \c" > temp0
68      echo " > dataseis${k}.su" > fin
69      while [ "$j" -ne "$nums" ]
70      do
71              echo "sudata${j}trazas.su \c" > temp
72              cat temp0 temp > aux
73              cp aux temp0
74              j=`expr $j + 1`
75      done
76
77      cat temp0 fin > cat.sh
78      chmod 777 cat.sh
79      ./cat.sh
80
81      echo " ***Archivos temporales eliminados***"
82      rm -f sudata*
83      rm -f temp*
84      rm -f fichero*
85      rm -f aux fin
86
87      k=`expr $k + 1`
88      done
89
90      echo " -***-Proceso terminado-***-"
```

2.1.1.2 EXPLICACION DEL CODIGO

líneas 1 a 4: Nótese que algunos de los reglones empiezan con el símbolo Numeral “#”, esto indica que este reglón es un comentario, en lo que sigue no se hará mas alusión a los reglones que empiecen por este símbolo, pues estos sirven para comentar el código; la línea 1 invoca el tipo de *Shell*, la línea 2, 3 y 4 hacen referencia al nombre del archivo, su uso y fecha de creación respectivamente.

líneas 6 a 7: En la línea 7 se asignan valores a las variables que determinan el tamaño del modelo. **x1beg** y **x1end** determinan las coordenadas mínimas y máximas respectivamente en el eje de la profundidad del modelo; de la misma forma, **x2beg** y **x2end** determinan las coordenadas mínimas y máximas respectivamente en el eje de las abscisas del modelo, como se ilustra en la figura 6.

líneas 8 a 9: En la línea 9 se asignan valores a las velocidades de las interfaces del

modelo, de esta forma **vel1** corresponde a la velocidad del estrato 1, que se extiende desde línea de la primera interfase (topografía horizontal en este caso) hasta la línea de la segunda interfase. La geometría de las interfaces se debe definir en un archivo creado con anticipación a la ejecución del programa, cuyo nombre podría ser cualquiera (**multi1** en este caso), siempre que se registre en **param1** (creado en la línea 36), **param1** es un archivo de entrada usado por **cshot1** llamado en la línea 40. Antes de seguir explicando el código, se muestra el formato de el archivo que contiene la geometría de las interfaces del modelo, llamado en este caso **multi1**:

0.	0.	Inicio de la capa de la Topografía
60000.	0.	...
1.	-99999.	Fin de la capa de Topografía
0.	100.	Interfase 1
60000.	100.	...
1.	-99999.	Fin Interfase 1
0.	500.	Interfase 2
60000.	500.	...
1.	-99999.	Fin de Interfase 2
0.	2000.	Interfase 3
6000.	2000.	...
1.	-99999.	Fin de Interfase 3
0.	3000.	Interfase 4
60000.	3000.	...
1.	-99999.	Fin de Interfase 4

En este archivo una fila representa un punto, a menos que el valor de la segunda columna sea -99999. La primera columna corresponde a la coordenada X (abscisas) de modelo, la segunda a la coordenada Y (profundidad). El valor -99999 es un estándar que indica donde termina cada interfaz.

líneas 10 a 11: En la línea 11 se especifica el número de trazas, en este caso constante por cada disparo, en una adquisición real pocas veces se da este caso.

línea 14: Aquí se define en la sintaxis propia del shell *Linux* para el calculo de variables enteras, el número de veces que se ejecutara el **bucle i**. que es declarado en la línea 25.

línea 16: Se usa el comando **echo**, que presenta una línea de texto en pantalla durante la ejecución del programa, en este caso anunciando el inicio de procesos en los bucles.

líneas 17 a 19: En este fragmento se define el inicio del bucle **k**, que tendrá otros bucles anidados (i, j), la definición de este bucle se hace usando la sintaxis propia de *Linux,* que es como sigue:

```
//...
    14: k=0
    15: while [ "$k" -ne "1" ]
    16: do
...
    62: k=`expr $k + 1`
    63: done
...//
```

en la línea 17 se asigna el valor en el cual empieza la variable $**k**, en la línea 18 : **while ["$k" -ne "6"]** se define la variable que cambia en cada iteración (**$k**), el símbolo **$,** indica que **k** es una variable y **-ne** es equivalente a "menor o igual que (<=)"; es decir, este proceso debe ejecutarse hasta que el valor de la variable **k** sea menor que **6**; en la línea 19 la sentencia **do** establece que en cada iteración debe ejecutarse lo declarado después de **do** hasta **done** en la línea 63; antes de cerrarse el ciclo debe especificarse como se actualizará la variable **k** para la siguiente iteración, que es el propósito de la línea 62: **k=`expr $k + 1`**, esta expresión es equivalente a **k=k+1**, sólo que está escrito en la sintaxis de *Linux* para el calculo de variables enteras.

líneas 21 a 23: Las líneas 21 y 23 se usan para tener un control del proceso que el programa está realizando durante la ejecución; la línea 22 calcula en valor de **ref,** esta es una variable que se usa para cambiar la red de receptores de sitio, cada vez que se actualiza el **bucle k.** Esto se hace sobre la misma línea con el objetivo de que el punto de disparo no quede fuera del arreglo de receptores también llamado *Split-Spread*. Se simula una situación que se presenta generalmente en cualquier adquisición sísmica terrestre, que es el cambio de posición de toda la red en cierto punto de la adquisición, pues deben atravesarse largas distancias y el numero de receptores (geófonos) es limitado. En este caso la línea tiene 10 Km, pero la red completamente tendida cubre solo 2 Km, por tanto debe mudarse la red de sitio 5 veces que es el número de veces que se ejecuta el **bucle k**). Es posible que esta operación pueda tener influencia sobre la imagen obtenida de esta área, pues en los extremos de la red el offset es máximo y solo se tiene información de lo que hay a un lado del punto de disparo, así que se esperaría que en el punto de cambio de la red la información que se tenga de estratos

más profundos es mejor.

líneas 25 a 27: Este es el inicio del bucle i, en los disparos, que hará una serie de operaciones para cada disparo, entre ellas: trazado de rayos sobre el modelo (problema directo), generación de trazas sintéticas y la asignación de los respectivas headers para cada traza; cada una de estas operaciones será tratada más adelante.

líneas 28 a 32: Se define cómo cambian en cada iteración las variables que definen la geometría de la adquisición. **ro** y **rf** representan la estación en la que se encuentra el primer y el ultimo receptor para un tiro respectivamente; **g1** y **g2** son variables que pueden ser usadas para introducir en el arreglo un segmento de línea sin receptores registrando señal, también llamado *gap* (hueco); en este caso se configuran para que la línea sísmica sea continua en cuanto a receptores.

línea 34: Cada disparo tiene su propia configuración geométrica en cuanto a ubicación de receptores y fuente, esta información debe almacenarse en un fichero llamado **geometry0,** por tanto se usa el comando **echo** para actualizarlo cada vez que ocurre una iteración en el **bucle i**, este fichero que sera leído por **cshot1** en la línea 40 para realizar el trazado de rayos sobre el modelo. **geometry0** es un fichero en formato ASCII que contiene la siguiente información:

1	0	0.				
2	15.	0.				
3	2	3	4	601	10.	0.

línea *1*: El primer valor es el numero de la estación que servirá de referencia para todos las demás estaciones que se especifiquen en este fichero y el segundo valor es la coordenada X donde dicha estación sera ubicada.
línea *2*: Se especifican dos flotantes para establecer la separación entre receptores y la profundidad a la que se ubican.
línea *3*: El primer valor es la estación donde esta ubicado el primer receptor para el disparo, el segundo y el tercero pueden usarse para que ciertos geófonos no registren señal durante la adquisición, el cuarto valor es la ubicación del ultimo geófono, por ultimo el quinto y sexto son la ubicación y profundidad de la fuente respectivamente. Todos los valores anteriores que son números de estaciones están referidas a la estación especificada en la línea 1.

línea 36: Usando el comando **echo** se actualiza el fichero **param1** en cada iteración,

este sera usado por **cshot1** para realizar el trazado de rayos sobre el modelo. **param1** contiene la siguiente información:

1 multi1
2 4
3 plotcolors
4 m
5 dummywell
6 s
7 geometry0
8 sg
9 rt
10 fichero1
11 -90. 90.
12 1.
13 600.0 2400.0 4000.0
14 5000.0 5200.0
15 n
16 1 2 3 4
17 y

línea ***1:*** Se especifica el nombre del fichero que contiene la geometría del adquisición.
línea ***2:*** Número de interfaces en el modelo.
línea ***3:*** Nombre del fichero que contiene información de los colores para la interfase gráfica.
línea ***4:*** Se especifica **m** si se desea dibujar el modelo
línea ***5:*** En caso de que se este simulando algún tipo de adquisición con fuentes o receptores en pozo esta línea se usa para especificar el nombre del fichero que contiene la información de las coordenadas del pozo.
línea ***6:*** Puede especificarse **s** si los disparos se hacen desde la superficie, o d si se hace desde el pozo.
línea ***7:*** Nombre del fichero que contiene la información de la geometría de fuente y receptores en cada disparo, llamado **geometry0** y creado en la línea 34.
línea ***8:*** Se especifica **sg** si se desea que en los disparos se grafique la posición de la fuentes y los receptores; o solo **s** para fuentes, o solo **g** para receptores.
línea ***9:*** Se escoge **rt** porque se desea graficar el trazado de rayos y además obtener el registro de tiempos, que sera usado después por el programa **cshot2**.
línea ***10:*** Se especifica el nombre del fichero de salida, que en este caso sera uno diferente por cada disparo, tantos como las veces que se ejecute el **blucle i**.
líneas ***11:*** En esta línea se determina el rango en grados en la que se hará el trazado de

rayos. en esta caso como el modelo sintético tiene topografía horizontal es escogido es el adecuado.

línea ***12:*** Se escoge el incremento en grados en la que se hara el trazado de rayo sobre el modelo.

líneas ***13*** y ***14:*** Se listan las velocidades de las interfaces primero las someras y luego las profundas.

línea ***15:*** Se escoge y/n (si/no) para trazar ondas directas sobre el modelo.

línea ***16:*** Se especifica el número de interfase sobre las que se propagaran las headwaves, debe escogerse preferiblemente las interfaces más rápidas.

líneas ***17:*** Se escoge y/n (si/no) para trazar ondas primarias sobre el modelo.

Para ilustrar el uso de los parámetros incluidos en las líneas 15, 16 y 17 del fichero **param1**, se presenta la figura 7 que muestra el resultado de realizar el trazado de rayos sobre el modelo usando diferentes tipo de onda.

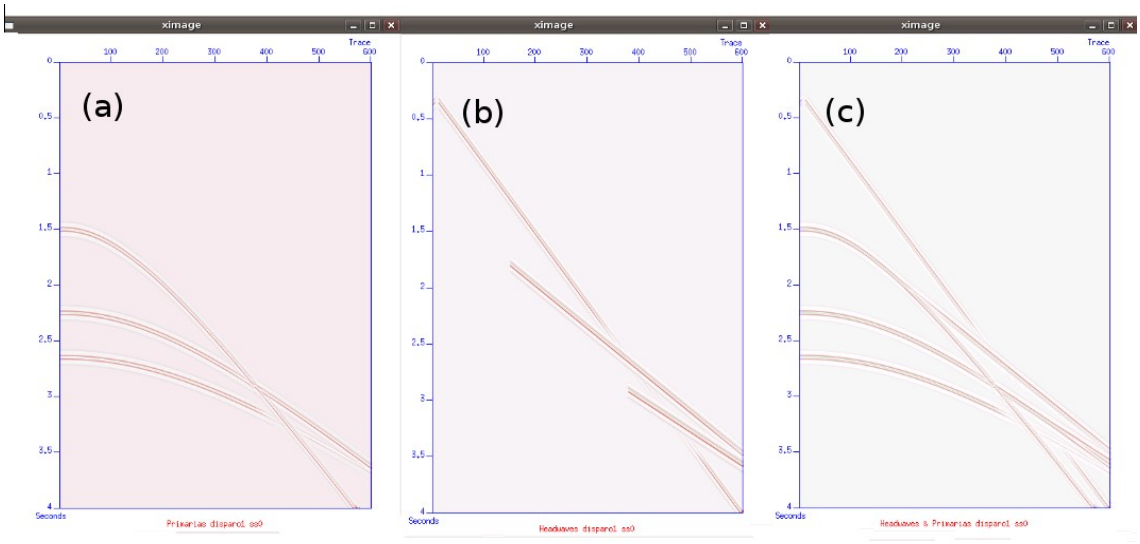


Figura 7. Sismogramas sintéticos obtenidos con diferentes tipos de ondas
(a) Primarias (b) Headwaves (c) Headwaves y primarias

línea 38: Se usa el comando **echo** para actualizar en cada iteración el fichero **param2**, que sera usado por **cshot2** para generar información sísmica sintética. A continuación se muestra el contenido de este fichero para el primer disparo:

1 s
2 1 1
3 1 600
4 10. 25. 35. 50.

5 **.**150
6 **.**004
7 4.
8 fichero1shot
9 fichero1traces

línea ***1:*** Se especifica **s** que indica la opción *shot* (disparo), esto quiere decir que se construirán registros de disparo, también existe la opción de construir registros de un receptor especifico escogiendo **r**.
línea ***2:*** Se define el registro de disparo a incluir en el shot.
línea ***3:*** Estos dos valores corresponden al primer y ultimo receptor respectivamente para este tiro.
línea ***4:*** Se define el espectro de frecuencia de la ondicula
línea ***5:*** Longitud de la ondícula en segundos.
línea ***6:*** Se establece la tasa de muestreo en segundos.
línea ***7:*** Se establece la longitud del registro en segundos.
línea ***8:*** Nombre del fichero de entrada, lleva el numero del disparo.
línea ***9:*** Nombre del fichero de salida, lleva el numero del disparo y contiene la información de las trazas sintéticas generadas por el programa cshor2.f para este tiro, es un fichero en formato binario.

línea 40: Se ejecuta el programa **cshot1**, este programa fue realizando por: "Center for Wave Phenomena (The Home of Seismic Un*x) Colorado School of Mines", y hace parte del paquete libre *Seismic Unix* disponible en *ftp.cwp.mines.edu*, el código fuente esta escrito en el lenguaje Fortran, este programa lee la información de los puntos que determinan la geometría de las interfaces y calcula los coeficientes interpolación spline-cubico, además lee la archivo **param1** que ya fue comentado. Producto de la ejecución se generan los siguientes archivos: **fichero*i*plot** y **fichero*i*par,** donde *i* es el numero de disparo.

líneas 43 a 47: Se usa el comando **xgraph** que sirve para generar un gráfico usando los ficheros de salida del schot1, en la línea 45 se añade titulo, en la línea 46 se asigna información de los ejes de la gráfica y en la línea 47 se especifican las variables que determina el tamaño del cuadro que contiene el gráfico.

línea 49: Se ejecuta el programa **cshot2**, este programa también fue realizando por: "Center for Wave Phenomena (The Home of Seismic Un*x) Colorado School of Mines", y hace parte del paquete libre *Seismic Unix* disponible en *ftp.cwp.mines.edu*, el

código fuente esta escrito en el lenguaje Fortran, este programa construye trazas usando los datos obtenidos del trazado de rayos por el **cshot1**. La información de las trazas obtenidas es guarda en un archivo denominado **ficheroitrazas.su**, uno por cada tiro. Una traza equivale a el registro obtenido por un transductor (geófono) durante una adquisición sísmica, la figura 8 muestra un conjunto de trazas producto de la ejecución de este programa.

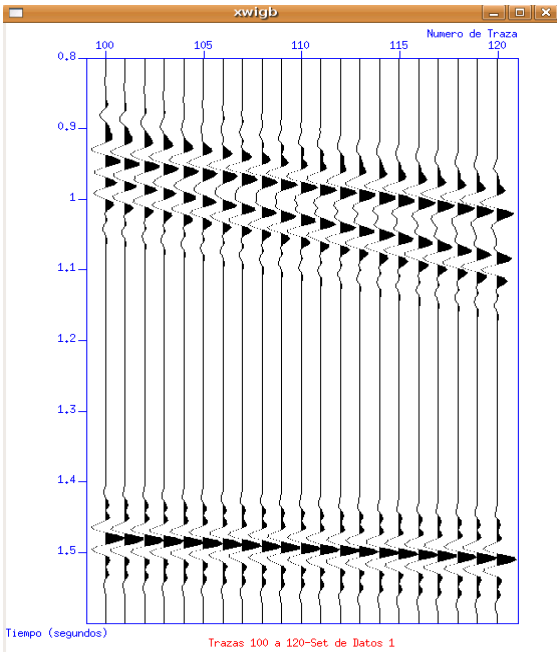


Figura 8. Conjunto de trazas para un segmento de disparo, para el juego de datos 1

Para el juego de datos generado por este programa la longitud de los registros es 4 segundos, en la figura 8 se muestra solo 1 segundo de registro el segmento entre 0.6 y 1.6 segundos.

línea 51: La variable **tiro** calculada en esta línea es básicamente un contador, que sirve para saber el número de disparo o *shot* que esta realizando el programa. Nótese que se tiene en cuenta el cambio de posición de la red, por esta razón es función de **k**. La variable tiro sera usada en la línea 54.

líneas 53 a 54: Al final de la línea 53 esta el símbolo " | ", llamado *the pipe*, sirve para conectar dos procesos que se ejecutan uno después de otro; de esta forma la línea 53 y 54 ejecutan los siguientes procesos: en la línea 53 se usa el comando **suaddhead,** como su nombre los indica pertenece a Seismic Unix (SU) y sirve para adicionar headers a un

juego de datos, que en este caso es llamado **fichero*i*traces** obtenido de la ejecución del cshot2. Nótese que entre ellos esta el operador director " **<** " este indica que el fichero especificado debe pasarse a través de algún comando, luego se declara el valor de ftn y la tasa de muestreo. En la línea 54 se usa el comando **sushw** para adicionar varios headers a cada traza en todos los disparos dentro del juego de datos, usando la expresión **key=tracr a=1 b=1 c=0** se declara el nombre del headers a adicionar y se utilizan otros parámetros que determinan el valor del header citado, en cada traza; el significado de cada parámetro es como sigue: **a** es el valor del header en la primera traza, **b** el incremento dentro del grupo, **c** el incremento del grupo y **j** numero de elementos dentro del grupo. Este comando se usa de forma diferente en un segmento de la línea 56 del juego de datos 2, donde se usa para adicionar el headers offset.

líneas 56 a 58: Se usa el comando **suximage** para graficar un disparo, se especifica el titulo, el tamaño de la ventana y el rotulo de cada eje, la figura 9 muestra el resultado de la ejecución de estas líneas para un solo disparo:

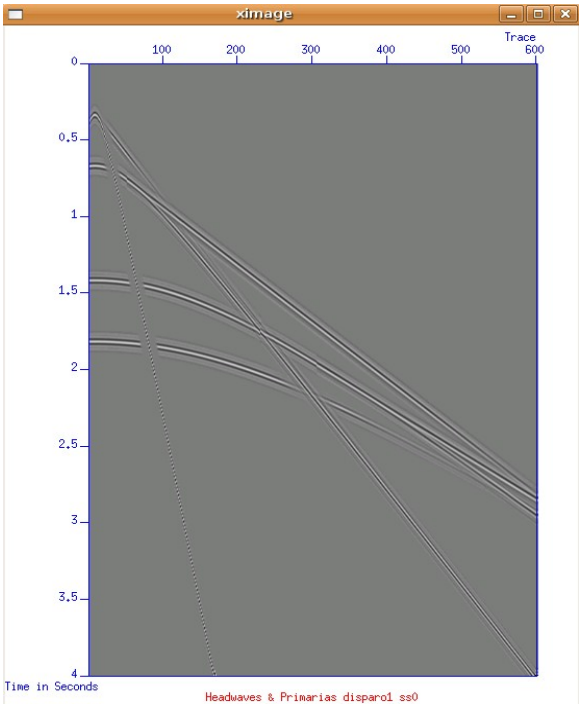


Figura 9. Vista del disparo 1 en escala de Grises- juego de datos Sintéticos 1

líneas 60 a 63: La instrucción de la línea 60 imprime en pantalla un aviso durante la ejecución, cada vez que la información sísmica se ha completado para un disparo; La líneas 62 y 63 sirven para actualizar la variable i antes de iniciar la siguiente iteración.

líneas 65 a 79: Cuando finalizan las iteraciones del **bucle i** se obtienen un conjunto de ficheros que contiene cada uno la información sísmica de un disparo, el paso a seguir es ensamblar tales ficheros. Para realizar esta tarea se utiliza el **bucle j** y el *shell* **cat.sh**. El **bucle j** inicia en la línea 66 y termina en la línea 75, en las líneas 67, 68, 71 y 72 se crean los archivos temporales **temp0**, **fin**, **temp** y **aux** respectivamente, que se usaran para crear el fichero **cat.sh** en la línea 77, este pequeño *shell* debe ejecutarse para que la información de los disparos sea ensamblada y guardada en un fichero llamado **dataseis$*i*.su,** uno por cada cambio de posición de la red. La función del **bucle j** es imprimir en el fichero **temp** el nombre del fichero que contiene la información sísmica de cada disparo, esto permite que **cat.sh** haga su tarea usando el comando **cat**, que sirve para concatenar archivos; en la línea 78 se crea el ejecutable de tal fichero con los permisos adecuados y el línea 79 se ejecuta.

líneas 82 a 85: Se hace uso del comando **rm** para eliminar archivos temporales que han sido usados durante la ejecución del programa pero que ya no se necesitan.

líneas 87 y 88: Se especifica como cambia la variable **k** en cada iteración y termina el **bucle k**.

línea 90: Se anuncia en pantalla la terminación del proceso.

2.2.1 SET SINTETICO CON OFFSET VARIABLE

En esta sección se presentan las modificaciones realizadas a el *shell* base presentado en la sección 2.1.1.1, el propósito es simular la configuración geometríca usada en la adquisición sísmica de la línea 1040, del programa Uribante 2005.

El **bucle k** presentado en la sección anterior, permitía extender la misma configuración geométrica de la adquisición sobre una línea sísmica, cambiando de posición en el momento justo, todo el arreglo de receptores; dado que para nuestro propósito, un tamaño del modelo igual a 10 km resulta suficiente, dicho bucle no sera usado en lo que sigue, así que no aparecerá en los códigos anexos para este sección. Otra modificación importante y que puede considerarse un acercamiento a los procedimientos que generalmente se realizan en el trabajo de campo, es simular una adquisición sísmica con número de trazas variante en cada disparo, para hacer esto posible se ha incluido en

el *shell* el siguiente fragmento de código:

```
27          if [ $s -ge 371 ]; then
28                  if [ $ro -le 371 ]; then
29                  ro=371
30                  fi
31                          if [ $rf -ge 1038 ]; then
32                          rf=1038
33                          fi
```

Los números de la izquierda indican el número de línea del *shell* [anexo 1] al que se hace referencia, Nótese que se han definido una serie de condiciones que cambian los valores de los términos que definen la geometría de la adquisición en cada iteración. Estos son **ro** y **rf** que representan la estación en la que se encuentra el primer y el ultimo receptor para un tiro respectivamente; para explicar lo que hace este segmento de código, se presenta en la figura 10 un esquema de la geometría de la adquisición:

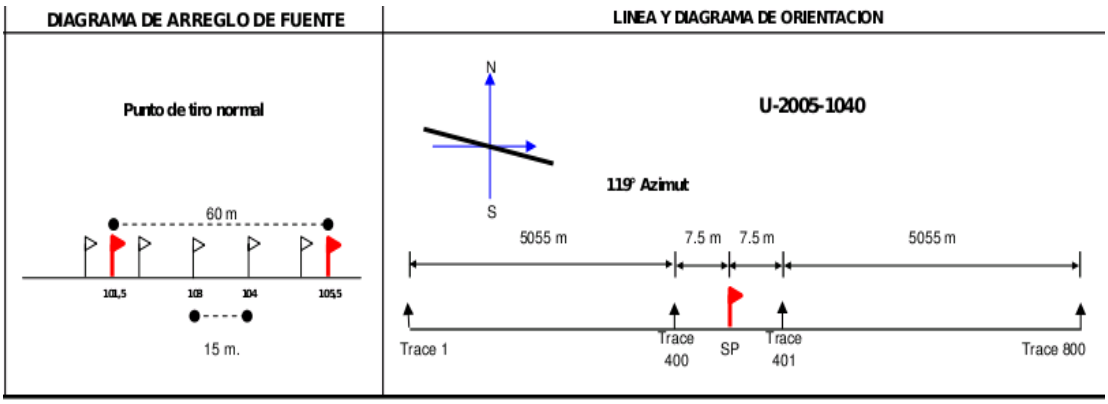


Figura 10. Representación esquemática de la geometría de la adquisición. Tomado del Reporte del Observador

El número de receptores aumenta por cada disparo al inicio de la línea, luego permanece constante durante un segmento de la línea, lo que también se conoce como *full-spread,* y disminuye al final del tendido sísmico.

Además existe la necesidad de incluir otros segmentos de código para complementar la información de los encabezados o *header* en los datos sísmicos, la información adicionada sera requerida para realizar el proceso de picado de primarios arribos; tales segmentos de código son la línea 15 y las líneas 55 a 56, que se presentan a continuación:

```
15    ./masHw.exe

55    suaddhead <fichero${i}traces ftn=1 ns=1001 |
56    sushw key=dt a=4000 | sushw key=tracl a=1 b=1 c=0 j=$deltag | sushw key=ep a=$s b=0
      c=0 | sushw key=sx a=$sx b=0 c=0 | sushw key=offset infile=offsetHw$i | sushw key=ntr
      a=$deltag b=0 c=0 | sushw key=fldr a=$i b=0 c=0 | sugain gagc=1 wagc=0.5
      >sudata${i}trazas.su
```

La línea 15 es muy importante porque ejecuta un programa creado en c++ [Anexo 1], que genera tantos ficheros en formato binario como número de disparos contenga el set. Los ficheros contienen los headers que deben aficionarse a las datos sísmicos. En este caso, cada uno contiene información del valor del offset para cada traza dentro de cada disparo. El header es incluido en la línea 56 de este mismo programa mediante la siguiente instrucción: **sushw key=offset infile=offsetHw$i**. Nótese que se usa el comando **sushw,** este es un comando de Seismic Unix que puede ser usado para configurar uno o más headers especificando algunos parámetros; como en el caso anterior: **key=offset,** donde la palabra **key** es la abreviatura de keyword, que es la palabra reservada por Seismic Unix para referirse a un Headers. Luego **infile=offsetHw$**i hace referencia al nombre del fichero que contiene la información del header a incluir en el juego de datos Sísmicos. Recuérdese que esta operación se realiza dentro del **bucle i**; lo que indica que debe existir un archivo binario por cada tiro; también se hubiera podido incluir esta información a el juego de datos completo; aunque al final resulta lo mismo y no existe diferencia en tiempo de computo.

Como se aprecia en la línea 56 el comando **sushw** también permite calcular valores para otros headers usando una serie de patrones, el mecanismo es simple, primero se especifica el nombre del headers por ejemplo key=tracl, luego se utiliza el siguiente parámetro para establecer el valor del header en la primera traza a=1, además puede usarse un segundo parámetro que indica el incremento del valor del header dentro del grupo b=1, por ultimo c=0 indica el incremento de grupo en caso de que existan varios.

Por ultimo se han adicionado al *shell* las líneas 92 a 95, para cambiar el formato, de los datos sísmicos sintéticos a el formato estándar para intercambio de datos sísmicos SEG-Y.

```
92    echo " ***Generando archivo seg-y***"
93    segyhdrs <dataseis.su
94    segywrite <dataseis.su tape=dataseis.sgy endian=1 hfile=header bfile=binary
```

```
95    rm -f dataseis.su
96    echo " -***-Proceso terminado-***-"
```

Este paso es necesario porque los set's de datos generados, se exportan a PROMAX un programa comercial de procesamiento sísmico, para realizar el picado de primeros arribos (sección 5.1). Luego serán invertidos usando tomografía sísmica (sección 5.2). En la figura 11 se muestran las velocidades usadas para generar el set sintético y la figura 12 un conjunto de disparos (*shot gather*) del set generado.

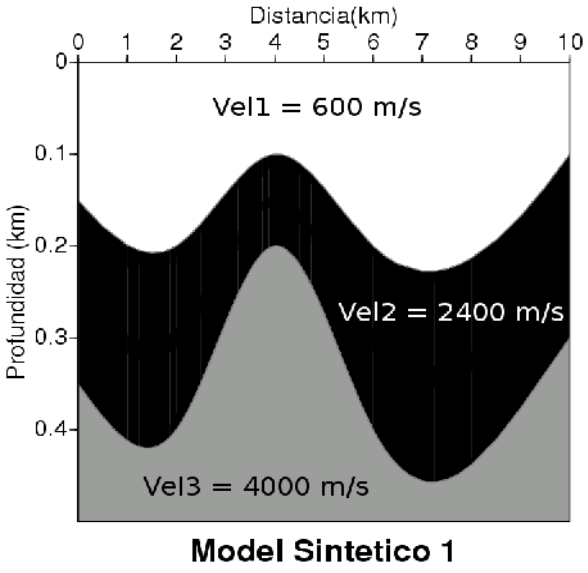
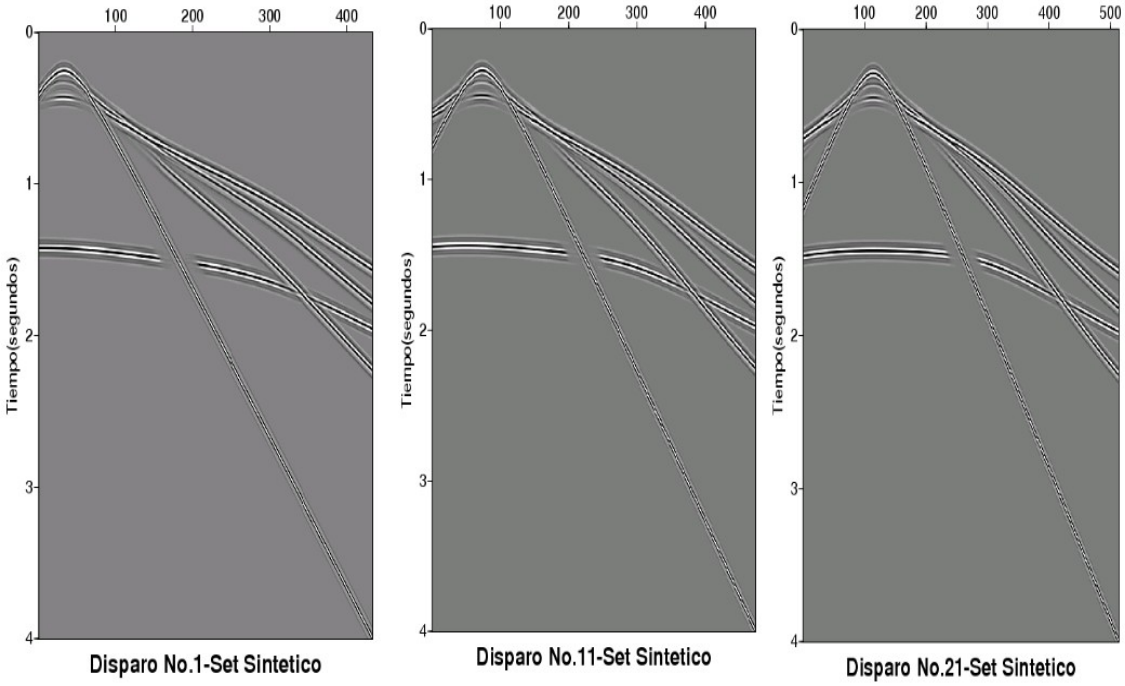
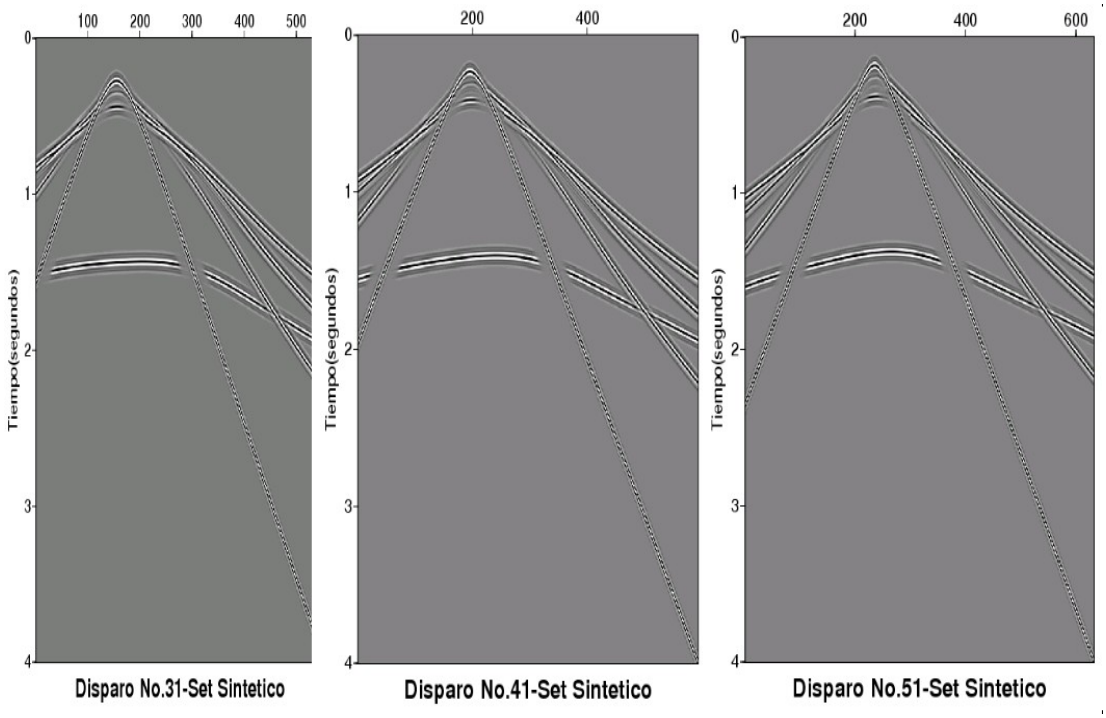
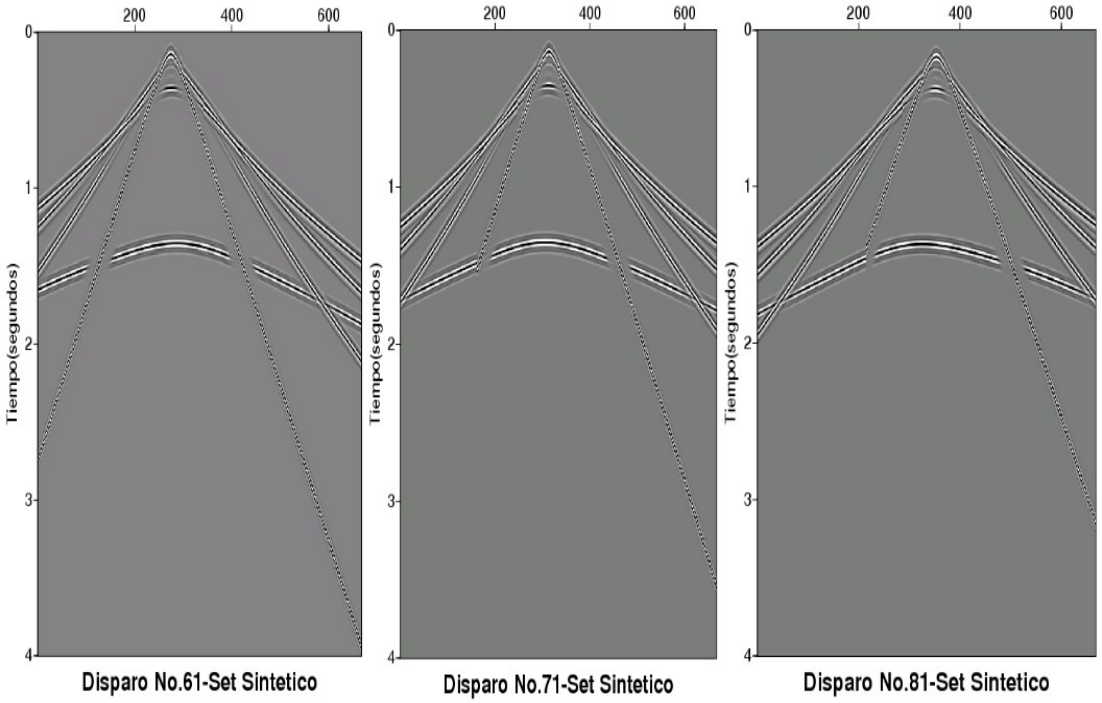
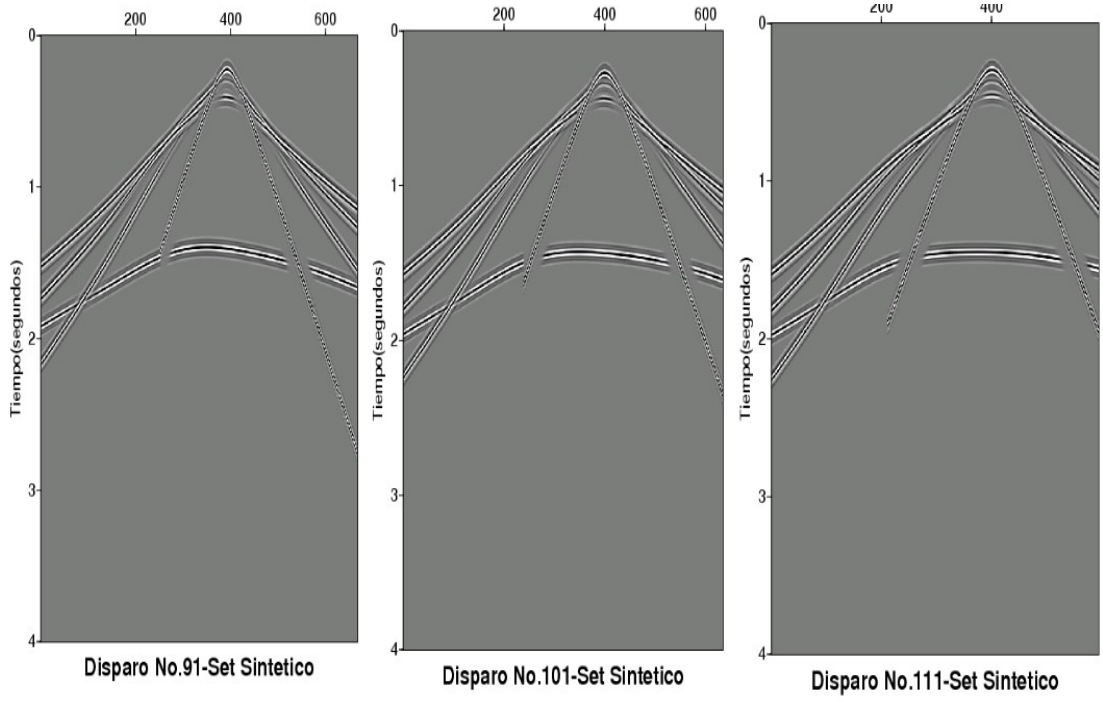


Figura 11. Modelo - set de datos sintético con offset variable.


100 200 300 400
0 1 2 3 4
Tiempo(segundos)
Disparo No.1-Set Sintetico
100 200 300 400
0 1 2 3 4
Tiempo(segundos)
Disparo No.11-Set Sintetico
100 200 300 400 500
0 1 2 3 4
Tiempo(segundos)
Disparo No.21-Set Sintetico


100 200 300 400 500
0 1 2 3 4
Tiempo(segundos)
Disparo No.31-Set Sintetico
200 400
0 1 2 3 4
Tiempo(segundos)
Disparo No.41-Set Sintetico
200 400 600
0 1 2 3 4
Tiempo(segundos)
Disparo No.51-Set Sintetico


200
400
600
0
1
2
3
4
Tiempo(segundos)
Disparo No.61-Set Sintetico
Disparo No.71-Set Sintetico
Disparo No.81-Set Sintetico


200
400
600
0
1
2
3
4
Tiempo(segundos)
Disparo No.91-Set Sintetico
Disparo No.101-Set Sintetico
Disparo No.111-Set Sintetico

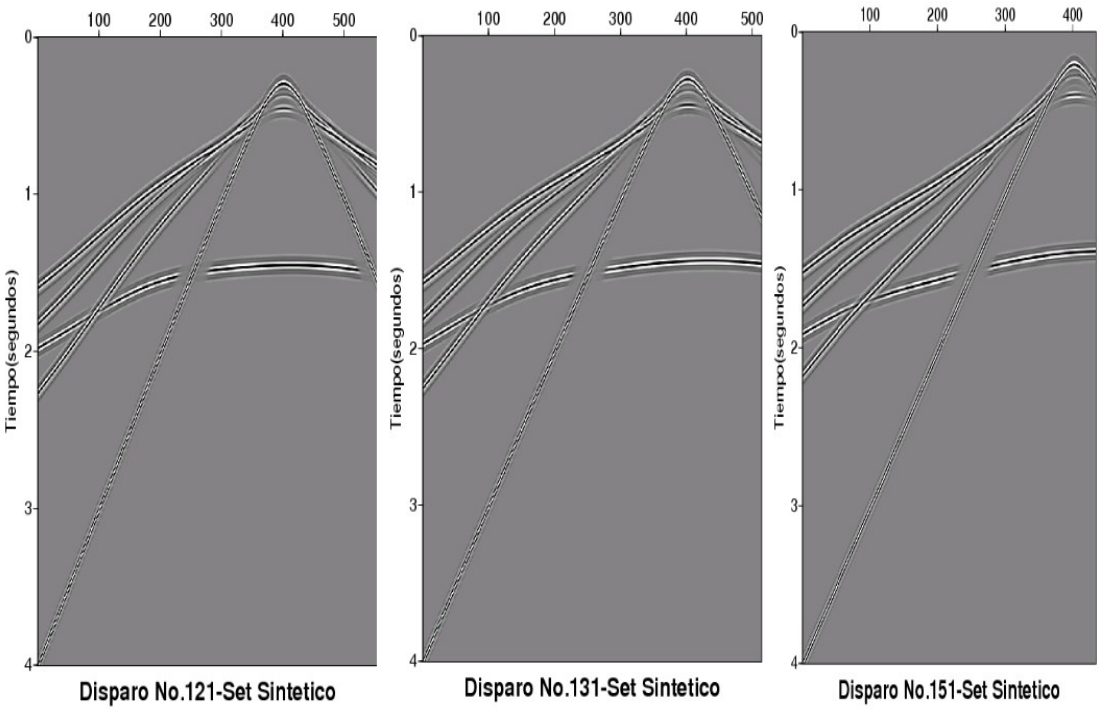


Figura 12. Conjunto de disparos - Set de datos sintético con offset variable.

capítulo 3

TECNICAS DE INVERSION TOMOGRAFICAS

Introducción

Las técnicas de tomografía sísmica ha sido muy usadas en varios campos de la ciencia, a mediados del siglo pasado nacieron en la medicina, usadas con gran éxito para detectar tumores en el cerebro. En sismología, se empezaron a usar alrededor de los años 1970s para construir modelos de velocidad que permitieran calcular con más exactitud la localización (profundidades) del foco de sismos, desde entonces se ha dado amplio uso en localización de sismos a partir de tiempos de viaje [Menke, 2000], también se usan para estudiar el espesor y velocidades del interior de la tierra a escala crustal [Sallarès and Charvis, 2003]. En geofísica las aplicaciones de la tomografía sísmica cubren un amplio rango de usos; por ejemplo: para hacer correcciones estáticas, para hacer modelos de velocidad y a partir de estos ayudar a la interpretación geoestadística de datos de registro de pozo o para predecir tensión y presión de poro en las rocas, además puede ser vista como una herramienta que complementa los métodos sísmicos convencionales y proporcionar información del subsuelo previamente no disponible.

Se trata de utilizar fuentes externas y a partir de las respuestas del medio generar imágenes para investigar áreas de interés geológico; esto es similar a los rayos X utilizados en medicina, que son usados para crear una imagen de la estructura interna del cuerpo de un paciente.

Implementar la técnica de tomografía sísmica requiere previamente hacer un modelo que represente a las ondas sísmicas viajando a través del subsuelo, existen dos modelos teóricos que pueden usarse para describir este fenómeno; teoría de rayos (*seismic ray tomography*) o teoría de difracción (*diffraction tomography*); la primera se usa cuando el tamaño de las heterogeneidades encontradas en el medio son mucho mayores que la longitud de onda sísmica producida por la fuente, y difracción cuando estas magnitudes son semejantes. En este capítulo solo se discuten los métodos de tomografía basados en trazado de rayos, para los que se discutirán sus limitaciones y aplicaciones especificas.

3.1 TOMOGRAFIA SISMICA POR RAYOS

Existen varios métodos basados en la teoría de trazado de rayo para hacer reconstrucción de imagen, estos pueden ser clasificados en: métodos de transformación (sección 3.1.1) o en métodos de expansión en series (sección 3.1.2); los primeros son más usados a aplicaciones médicas pero su marco conceptual sirve para introducir el uso de la tomografía en aplicaciones de sísmica.

3.1.1 METODOS DE TRANSFORMACION

Existen dos métodos de transformación: el primero es el método de transformación directa (*direct-transform ray tomography*) y el segundo que es una aplicación parecida pero más eficiente computacionalmente, conocido como método de retro-proyección (*backprojection ray tomography)*; para poder entender estos métodos se hace necesario comprender el teorema del cual se derivan, llamado teorema de la sección proyección (*proyection slice theorem*).

Supongamos que vamos a hacer un sondeo utilizando rayos X, el propósito es conocer la distribución en coordenadas espaciales, de la variación de un parámetro físico de interés; esta distribución se notara de aquí en adelante $M(x,z)$ haciendo alusión a que es una función escalar de variable vectorial en el medio (*the model function)*. Este

parámetro puede darnos por ejemplo información acerca de cómo está distribuida la velocidad (en tomografía es más útil el inverso de la velocidad) dentro de un objeto y por lo tanto, cómo es afectada una onda sísmica propagándose en este medio, el parámetro medido (que se le llama también parámetro observado) puede ser la atenuación de la intensidad del rayo X o como se estudiara en la sección 3.1.2, el tiempo de propagación. En nuestro caso el objeto podría ser un cubo que contuviese dentro de si, una cuña perfectamente acoplada; ambos con materiales de diferente densidad (la cuña más densa que el cubo) [Figura 13].

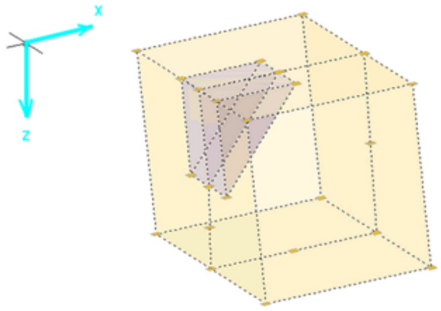


Figura 13. Representación esquemática de un medio.

En el modelo sintético utilizaremos sólo el plano x-z que se encuentra en la mitad del modelo. En una aplicación real seria necesario hacer varios sondeos en diferentes planos x-z a través de todo el eje paralelo al plano x-z, para poder tener una idea de la forma real en 3D del objeto. Para hablar del teorema del cual se derivan los métodos de transformación se usará sólo un plano x-z. Los rayos X son emitidos por transmisores y viajan a través del medio paralelos al plano x-z y son medidos por receptores ubicados al costado extremo, como se muestra en la figura 14; el eje por el cual viajan los rayos será llamado el eje v, y hará parte de un sistema de coordenadas u-v ubicado sobre el mismo plano pero rotado un ángulo θ con respecto al sistema de coordenadas x-z. En la figura 14 $\theta = 0$ y en la figura 15 $\theta = 45^0$.

La relación que existe entre los dos sistemas de coordenadas cuando uno esta rotado un ángulo θ con respecto al otro esta expresado en la siguiente ecuación:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta \\ \operatorname{sen}\theta & \cos\theta \end{bmatrix} * \begin{bmatrix} u \\ v \end{bmatrix} \qquad (3\text{-}1)$$

El sistema de coordenadas u-v es importante porque permite describir todas las posibles orientaciones de los transmisores y los detectores a través del objeto. El eje u

es el eje sobre el cual se ubican los receptores y es perpendicular a la dirección de propagación de los rayos

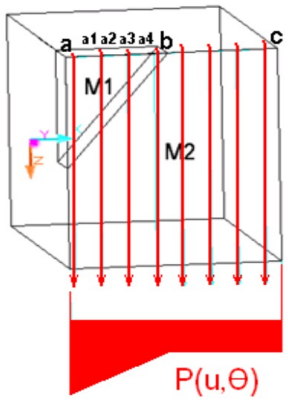


Figura 14. Trazado de rayo sobre un medio (Ø=0˚)

Para un rayo dado en la figura 14 se puede definir una relación entre la intensidad inicial del rayo I_0, es decir, la que emitió del transmisor y la intensidad medida por el detector $I(u,\Theta)$ en (u,Θ); esta relación esta representada por la siguiente expresión y se denomina una proyección o "*data function*":

$$P(u,\theta) = \frac{I_0 - I(u,\theta)}{I_0} \qquad (3\text{-}2)$$

Suponiendo arbitrariamente que la intensidad del rayo emitido es igual a 1; que la cuña esta constituida por un material que tiene la capacidad de disminuir la intensidad de los rayos en un 50% y el cubo por un material que tiene la capacidad de absorber la intensidad en un 20%; ambos en función de su geometría, se puede construir una proyección (*data function*) para un ángulo dado. En la tabla [1] se muestra el calculo para la proyección $\theta = 0$; estas proyecciones son las que aparecen sobre el eje u en las figuras 12 y 13 cada una con un ángulo diferente, 0º y 45º respectivamente.

punto	X	M1	M2	M1	M2	M1+M2	
a	0	0,5	0,5	0,25	0,1	0,35	0,65
a1	0,1	0,4	0,6	0,2	0,12	0,32	0,68
a2	0,2	0,3	0,7	0,15	0,14	0,29	0,71
a3	0,3	0,2	0,8	0,1	0,16	0,26	0,74
a4	0,4	0,1	0,9	0,05	0,18	0,23	0,77
b	0,5	0	1	0	0,2	0,2	0,8
c	1	0	1	0	0,2	0,2	0,8

tabla 1

El teorema de proyección permite encontrar una relación entre $M(x,z)$ y $P(u,\theta)$, es decir entre los parámetros del medio y las observaciones, para lo cual debe cumplirse la condición de que la atenuación de los rayos en el medio sea pequeña (los valores altos de atenuación usados en el cálculo de las proyecciones fueron escogidos arbitrariamente solo para construirlas).

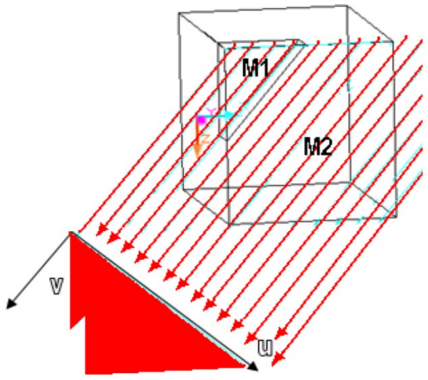


Figura 15. Trazado de rayo sobre un medio ($\emptyset=45^\circ$)

Cumpliéndose esta condición dicha relación puede ser expresada matemáticamente así:

$$P(u,\theta)=\int_{ray} M(x,z)dv \qquad (3\text{-}3)$$

En un sondeo sólo se conoce la proyección $P(u,\theta)$ y se desea conocer $M(x,z)$; observando la relación anterior se nóta que cada observación puede ser vista como una solución particular de esta ecuación; como la primera suposición también se cumple para el medio fuera del objeto, la expresión anterior se puede expresar en términos de los limites infinitos:

$$P(u,\theta)=\int_{-\infty}^{\infty} M(x,z)dv \qquad (3\text{-}4)$$

Para obtener una relación mas útil es necesario transformar $M(x,z)$ y $P(u,\theta)$ al dominio de las frecuencias, utilizando este criterio se llega a la trasformada de Fourier 2-D para $M(x,z)$:

$$\tilde{M}(K_x,K_z)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} M(x,z)e^{-j(k_x x+K_z z)}dxdz \qquad (3\text{-}5)$$

En esta ecuación k_x y k_z son frecuencias espaciales a lo largo de los ejes x y z,

estas frecuencias son función de λ que es la longitud de onda:

$$k = \frac{2\pi}{\lambda} \qquad (3\text{-}6)$$

Es importante mencionar que si se conoce la transformada de Fourier $\widetilde{M}(x,z)$, es posible determinar $M(x,z)$; como se aprecia en la siguiente expresión, donde esta planteada la transformada inversa de Fourier:

$$M(x,z) = \frac{1}{4\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \tilde{M}(K_x K_z) e^{j(K_x x + K_z z)} \partial K_x \partial K_z \qquad (3\text{-}7)$$

Para la proyección que también debe pasarse al dominio de las frecuencias, $\widetilde{P}(\Omega,\theta)$ representa la transformada de Fourier 1-D de $P(\Omega,\theta)$:

$$\tilde{P}(\Omega,\theta) = \int_{-\infty}^{\infty} P(u,\theta) e^{-j\Omega u} \partial u \qquad (3\text{-}8)$$

En esta ecuación Ω es una frecuencia espacial a lo largo del eje u, sustituyendo

$$P(u,\theta) = \int_{ray} M(x,z) dv \qquad (3\text{-}9)$$

en la ecuación anterior se tiene:

$$\tilde{P}(\Omega,\theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} M(x,z) e^{-j\Omega u} \partial v \partial u \qquad (3\text{-}10)$$

Pero se usa la siguiente expresión para dejar la ecuación anterior en términos de x y z:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta \\ \operatorname{sen}\theta & \cos\theta \end{bmatrix} * \begin{bmatrix} u \\ v \end{bmatrix} \qquad (3\text{-}11)$$

El resultado de este proceso es:

$$\tilde{P}(\Omega,\theta) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} M(x,z) e^{-j\Omega(x\cos\theta + z\sin\theta)} dx\, dz \qquad \text{ò}$$

$$\tilde{P}(\Omega,\theta)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} M(x,z)e^{-j(\Omega(\cos\theta)x+\Omega(\sin\theta)z)}dx\,dz \qquad (3\text{-}12)$$

Comparando la ecuación anterior con:

$$\tilde{M}(K_X,K_Z)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} M(x,z)e^{-j(k_x x+K_z z)}dx\,dz \qquad (3\text{-}13)$$

Se ve que $\widetilde{P}(\Omega,\theta)$ es simplemente la transformada de Fourier de $M(x,z)$ donde Kx y Kz esta limitado al eje Ω por:

$$\begin{aligned} k_x &= \Omega\cos\theta \\ k_z &= \Omega\sin\theta \end{aligned} \qquad (3\text{-}14)$$

Sustituyendo esta expresión en:

$$\tilde{P}(\Omega,\theta)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} M(x,z)e^{-j(k_x x+k_z z)}dx\,dz \qquad (3\text{-}15)$$

Esta es la función de los datos *(the data function)* en el dominio de la frecuencia y ahora recordemos la expresión de la función del modelo también en frecuencias:

$$\tilde{M}(K_x,K_z)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} M(x,z)e^{-j(k_x x+K_z z)}dx\,dz \qquad (3\text{-}16)$$

Mirando las dos ecuaciones anteriores se puede llegar a la siguiente relación que es el mencionado teorema de la sección proyección (*the projection slice theorem*): $\widetilde{P}(\Omega,\theta)=\widetilde{M}(Kx,Kz)$ En palabras este teorema dice que $\widetilde{P}(\Omega,\theta)$ es igual a una sola sección de $\widetilde{M}(x,z)$; esto es más claro en la Figura 16:

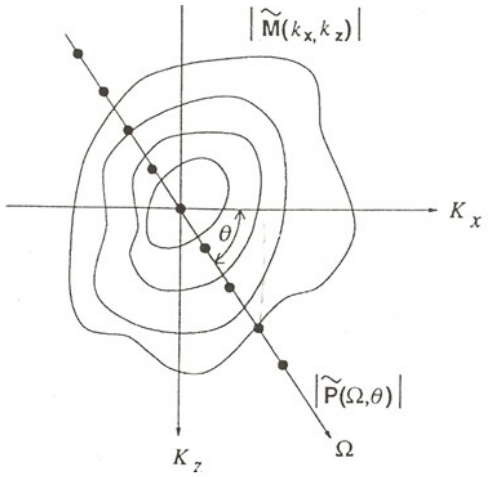


Figura 16. Representación del teorema de la sección proyección. Tomado de Fitterman 1994

Este teorema es muy importante porque nos dice que debemos hacer sondeos variando el ángulo θ de 0º a 180º para tener una representación fiel del medio, y establece una relación entre los datos medidos (*data function*, $P(\Omega,\theta)$) y el Medio (*model function*, $M(x,z)$).

A partir de este teorema se derivan dos métodos: el primero es el directo, del cual no se hablara mucho debido a su limitación, para poder emplearlo se requiere la interpolación de datos de una grilla polar a una grilla cartesiana, este proceso es tortuoso e inestable, y puede fácilmente producir errores que no permiten reproducir fielmente las características reales de el medio por el cual se propaga la energía.

El método de retro-proyección no presenta las limitaciones del método directo; su algoritmo se puede resumir en tres pasos básicos:

Paso1: adquisición de datos: es la medición del parámetro desconocido; que puede ser atenuación de la intensidad o tiempos de viaje.

$$P(u,\theta)=\int_{-\infty}^{\infty} M(x,z)\,dv \qquad (3\text{-}17)$$

Paso 2: tomar la transformada de Fourier 1-D de cada proyección *(data function)* a lo largo de el eje u:

$$\tilde{P}(u,\theta)=\int_{-\infty}^{\infty} P(u,\theta)*e^{-j\Omega u}\,du \qquad (3\text{-}18)$$

Paso 3: usar la siguiente ecuación para calcular $M(x,z)$ desconocido:

$$M(x,z)=\frac{1}{4\pi^2}\int\int \tilde{P}(\Omega,\theta)*e^{j\Omega(x\cos\theta+z\sin\theta)}|\Omega|d\Omega d\theta \qquad (3\text{-}19)$$

Este es un eficiente algoritmo para reconstrucción de imagen, porque es fácilmente computable. Pero tiene la limitación que solo puede ser aplicado en áreas pequeñas, debido a que el ensamblaje donde están ubicados las fuentes y los receptores debe rotar alrededor del área de interés, imposible en sísmica. Por este motivo son presentados a continuación otras metodologías.

3.1.2 MÉTODOS DE EXPANSIÓN EN SERIES (*series expansion methods*)

Los métodos de transformación determinan $M(x,z)$ a partir de la medida de la atenuación de la energía en el medio, pero el camino que toman los rayos es líneal, por esta razón es que son útiles en medicina, pero su aplicación a la sísmica es pobre; por esto la importancia de los métodos de expansión en series, porque comprenden un grupo de algoritmos que permiten trayectorias curvas de los rayos y a través de los cuales se puede determinar $M(x,z)$ *(the model function)*.

3.1.2.1 PROBLEMA DIRECTO (*forward Modeling Problem*)

El método de la expansión en series determina $M(x,z)$, a partir de un $M(est)$, el proceso llevado a cabo es iterativo y se hace por comparación entre la función de los datos observados (P^{obs}) con una función de datos predichos (P^{pred}).

Como se mencionó en la sección anterior, existe una relación entre $M(x,z)$ y P, que es más evidente en el dominio de la frecuencia, en el caso de la expansión en series esta relación se puede escribir así:

$$P^{obs}(u,\theta)=\int_{ray} M^{true}(x,z)\,dv \qquad (3\text{-}20)$$

Pero como el camino de los rayos no es líneal sino curvo, es necesario hacer un cambio en la notación de la ecuación anterior, y dejarla en función de un vector posición r.

$$P^{obs} = \int_{ray} M^{true}(r)\,dr \qquad (3\text{-}21)$$

En esta ecuación P^{obs} representa una proyección medida dada (*observed tomography data*) y $M^{true}(r)$ la distribución espacial desconocida (*Model function*) que se desea determinar. Partiendo de la ecuación 3-21 se puede formular el problema directo, que es una herramienta importante de los métodos de inversión y esta expresado en la siguiente ecuación:

$$P = \int_{ray} M(r)\,dr \qquad (3\text{-}22)$$

donde, P es P^{pred} (datos predichos) y $M(r)$ es M^{est} (la función del modelo estimado).

3.1.2.2 DISCRETIZACION DEL MODELO

Para permitir el cálculo del método es necesario discretizar el medio por donde se propagan las ondas, esto se hace dividiendo esta área en muchas celdas pequeñas, a mayor número de celdas más refinado será el modelamiento. A cada celda se le asigna un valor promedio del parámetro físico estudiado que puede ser la atenuación o la lentitud (slowness). Entonces M *(model function)* es dividido en J celdas y cada celda estará notada por M_j que representa un valor promedio de $M(r)$.

Como el modelo es discretizado se requiere hacer algunos cambios en la ecuación 3-22, esta puede escribirse así:

$$P = \sum_{j=1}^{J} M_j S_j \qquad (3\text{-}23)$$

En esta ecuación M_j representa un parámetro de interés promedio del medio, S_j es la longitud del rayo en la celda y J es el número de celdas en las que ha sido discretizado el modelo. Si se tiene sólo una fuente y un receptor, solo se tendrá el registro de un rayo atravesando el medio, dicho rayo no atraviesa todas las celdas, así

que para que la ecuación anterior sea consistente se requiere hacer $S_j = 0$ para todas las celdas que no son atravesadas por el rayo, para modelar adecuadamente el área de interés se requiere que exista no sólo un rayo atravesando el medio; así que deben introducirse más fuentes, mas receptores y tener el registro de muchos rayos atravesando el medio; entonces se hace necesario modificar la ecuación anterior para que exista por cada rayo un valor de proyección, la ecuación anterior se puede expresar así:

$$P_i = \sum_{j=1}^{J} M_j S_{ij} \qquad \text{Para i = 1, 2,3,……., I,} \qquad (3\text{-}24)$$

En esta ecuación I representa el número total de rayos, S_{ij} es la longitud del camino del i-esimo rayo a través de la j-esima celda, y M_j es la estimación discretizada del modelo *(model function)* para la j-esima celda, siendo J el número total de celdas. Esta ecuación puede modelar el proceso de adquisición de datos si se miden tantas proyecciones como número de rayos haya; el parámetro físico medido puede ser los tiempos de viaje o el porcentaje de la variación de la intensidad de los rayos X; y una estimación de un valor promedio de un parámetro físico del modelo verdadero para cada celda, que puede ser la lentitud (slowness) o la atenuación de los rayos X. matemáticamente esto se puede expresar así:

$$P_i^{obs} = \sum_{j=1}^{J} M_j^{\,true} S_{ij} \quad \text{Para i = 1, 2,3,……., I,} \qquad (3\text{-}25)$$

Una de las técnicas para poder resolver esta ecuación es el método de Kaczmarz` que es el tema de la siguiente sección. Este algoritmo proporciona la estructura teórica para resolver indirectamente la ecuación 3-25, antes se escriben las ecuaciones anteriores en forma matricial, para hacer más sencillo el proceso:

$$P = SM \qquad (3\text{-}26)$$

donde P es un vector que contiene los datos de la predicción de las proyecciones, M es un vector que contiene los valores estimados para el medio discretizado y S es una matriz que contiene la longitud del camino de los I rayos y las J celdas:

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ . \\ . \\ . \\ P_I \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} & S_{13} & . & . & . & S_{1J} \\ S_{21} & S_{22} & S_{23} & . & . & . & S_{2J} \\ S_{31} & S_{32} & S_{33} & . & . & . & S_{3J} \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ S_{I1} & S_{I2} & S_{I3} & . & . & . & S_{IJ} \end{bmatrix} * \begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ . \\ . \\ . \\ M_J \end{bmatrix} \qquad (3\text{-}27)$$

En esta relación S puede pensarse como un operador líneal que actúa en el vector M, produciendo el vector P *(predicted data)*. De manera similar se puede expresar matricialmente la misma relación en términos de P^{obs} y M^{true} así:

$$P^{obs} = S.M^{true} \qquad (3\text{-}28)$$

Recordemos que aquí el objetivo es encontrar M^{true} pero la ecuación anterior no puede ser resuelta directamente, las observaciones (P^{obs}) son conocidas, así que el problema es: encontrar un operador S^{-g} (*generalized inverse operador*) y aplicarlo a ambos lados de la ecuación anterior para poder determinar M^{true} :

$$S^{-g}.P^{obs} = S^{-g}S.M^{true} \qquad (3\text{-}29)$$

$$S^{-g}.P^{obs} = M^{true} \qquad (3\text{-}30)$$

Esta ecuación presenta algunos problemas para su cómputo, porque la matriz S es normalmente muy grande y además está mal condicionada; esto hace el cómputo inestable.

3.1.2.3 ALGORITMO DE KACZMAR'Z

Es un algoritmo que sirve para encontrar una solución aproximada de la ecuación 3-30; el método es iterativo y puede explicarse brevemente a través del siguiente diagrama de flujo, que puede tomarse como un procedimiento de toda técnica tomográfica:

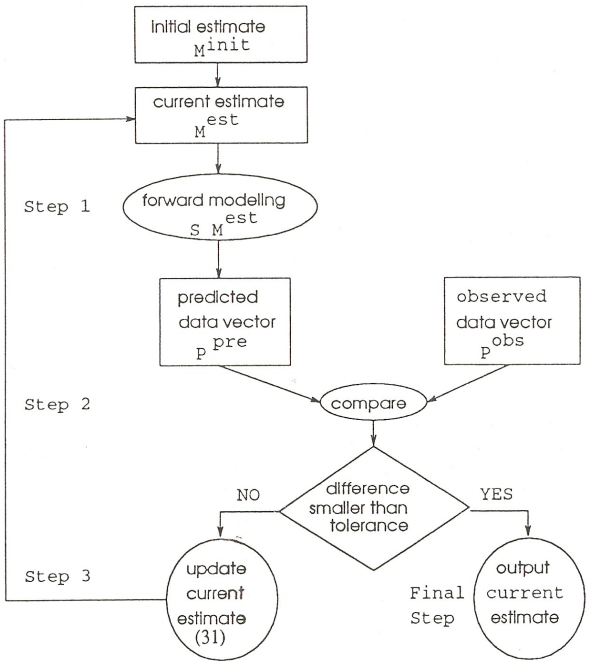


Figura 17. Diagrama de flujo del algoritmo de Kaczmar´z. Tomado de Fitterman 1994

Para empezar hay que suponer un modelo inicial (vector Modelo); es decir: M^{init}, este vector entra al bucle iterativo de el algoritmo y sirve como la primera estimación: M^{est}.

Después se ejecuta el primer paso que consiste en usar la ecuación 3-31 en forma matricial, para encontrar un vector de datos predichos P^{pred}:

$$P = SM \qquad (3\text{-}31)$$

En este paso la matriz S se determina por medio de un método de trazado de rayos. El segundo paso consiste en hacer una comparación entre P^{pred} y P^{obs} y tomar la diferencia absoluta entre las dos, si esta diferencia es menor que la tolerancia especificada el algoritmo llega a su fin; debido a que el modelo predicho explica los datos, o como lo expresa la ecuación 3-32:

$$M^{est} \approx M^{true} \qquad (3\text{-}32)$$

En caso de que la diferencia no sea menor que la tolerancia, se hace uso de la información de la diferencia para actualizar el M^{est} ; esto se puede expresar mediante

la ecuación 3-33:

$$M^{(new)est} = M^{est} + \Delta^i M \qquad (3\text{-}33)$$

Donde $\Delta^i M$ representa el incremento de la actualización de M^{est}.

La forma de computar el incremento es lo esencial de este método y puede representarse matemáticamente según la siguiente expresión:

$$\Delta^i M = \begin{bmatrix} \Delta^i M_1 \\ \Delta^i M_2 \\ . \\ . \\ \Delta^I M_J \end{bmatrix} \qquad (3\text{-}34)$$

Donde:

$$\Delta^i M_J = S_{ij} \frac{P_i^{obs} - P_i^{pre}}{\sum_{j'=1}^{J} (S_{ij})^2} \qquad (3\text{-}35)$$

También puede ser escrito

$$\Delta^i M_J = S_{ij} \frac{P_i - \sum_{j'=1}^{J} S_{ij'} M_{j'}^{est}}{\sum_{j'=1}^{J} (S_{ij})^2} \qquad (3\text{-}36)$$

Las ecuaciones anteriores pueden ser derivadas a partir de la geometría, el siguiente ejemplo tomado de Fitterman (1994), muestra la derivación de la anterior fórmula y además muestra como converge el método a la solución.

Supongamos que se tienen sólo dos rayos (I=2), que atraviesan sólo dos celdas (J=2) las proyecciones observadas a través del medio se pueden expresar matemáticamente así:

$$P_1^{obs} = S_{11}M_1 + S_{12}M_2 \qquad (3\text{-}37)$$

$$P_2^{obs} = S_{21}M_1 + S_{22}M_2 \qquad (3\text{-}38)$$

Para el rayo 1 y 2 respectivamente. En las ecuaciones anteriores las proyecciones son conocidas, porque son datos observados, M_1 y M_2 son desconocidos.

En la figura 18 se muestra un grafico 2D con ejes M_1 y M_2,

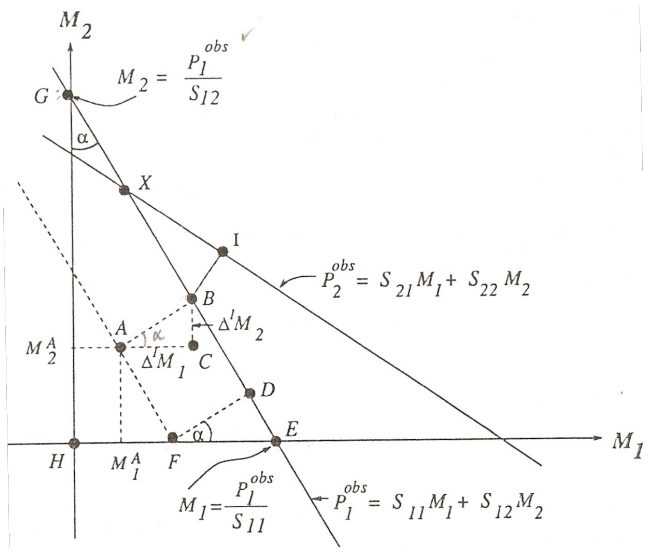


Figura 18. Ejemplo de la aplicación del algoritmo de Kaczmar´z en un modelo de dos celdas con dos rayos que las atraviesan. Tomado de Fitterman (1994)

La solución a la que queremos llegar es el punto donde se interceptan los hiperplanos de las dos proyecciones, como ya se indicó en la explicación del algoritmo lo que sigue es asumir un modelo inicial representado en el plano por el punto A; el siguiente paso es converger al punto B que es la proyección perpendicular paralela a uno de los hiperplanos esto puede representarse matemáticamente por las siguientes expresiones:

$$M_1^B = M_1^A + \Delta^1 M_1 \qquad (3\text{-}39)$$

$$M_2^B = M_2^A + \Delta^2 M_2 \qquad (3\text{-}40)$$

En estas ecuaciones los deltas representan los incrementos; usando las relaciones geométricas entre los triángulos ABC, FED y GEH (figura 2) se pueden escribir las

siguientes relaciones:

$$\Delta' M_1 = \overline{A}\overline{B}Cos\alpha = \overline{D}\overline{F}Cos\alpha = \overline{E}\overline{F}\cos^2\alpha = \overline{E}\overline{F}\frac{GH^2}{GE^2} \qquad (3\text{-}41)$$

$$\Delta' M_2 = \bar{A}\bar{B}Sen\alpha = \bar{D}\bar{F}Sen\alpha = \bar{E}\bar{F}\cos\alpha * \mathrm{Sen}\alpha = \bar{E}\bar{F}\frac{GH}{GE}\frac{EH}{GE} \qquad (3\text{-}42)$$

Ahora los segmentos EF, GH, GE, EH, GE (figura 2) deben estar en términos de P y S.

Para poder determinar el segmento FE es necesario tener en cuenta la ubicación del punto E; que esta determinada por el punto de intersección de un hiperplano con el eje M_1. El punto F es la intersección con el eje M_1 de una línea que es paralela a un hiperplano que contiene el punto A; usando la siguiente relación se pueden encontrar las coordenadas del punto F.

$$S_1 M_1^{\,A} + S_{12} M_2^{\,A} = S_{11} M_1 + S_{12} M \qquad (3\text{-}43)$$

Entonces las coordenadas del punto F a lo largo del eje M_1, esta dado por

$$M_1 = \left(\left(S_1 M_1^{\,A} + S_{12} M_2^{\,A} \right) / S_{11}; M_2 = 0 \right) \qquad (3\text{-}44)$$

De esta manera la longitud del segmento de línea EF esta dado por:

$$\overline{E}\overline{F} = \frac{1}{S_{11}}\left[P_1^{obs} - \left(S_{11} M_1^{\,A} + S_{12} M_2^{\,A} \right) \right] \qquad (3\text{-}45)$$

La relación entre las distancias GH y GE, se pueden determinar geométricamente del mismo plano

$$\frac{\overline{G}\overline{H}}{\overline{G}\overline{E}} = \frac{P_1^{obs}/S_{12}}{\sqrt{\left(P_1^{\,obs}/S_{11} \right)^2 + \left(P_1^{obs}/S_{12} \right)^2}} \qquad (3\text{-}46)$$

$$\frac{\overline{G}\overline{H}}{\overline{G}\overline{E}} = \frac{S_{11}}{\sqrt{S_{11}^{\,2} + S_{12}^{\,2}}} \qquad (3\text{-}47)$$

De igual manera para las distancias EH y GE

$$\frac{\overline{EH}}{\overline{GE}} = \frac{S_{12}}{\sqrt{S_{11}^{\ 2} + S_{12}^{\ 2}}} \qquad (3\text{-}48)$$

Sustituyendo las ecuaciones anteriores en las siguientes expresiones

$$\Delta' M_1 = \overline{EF}\frac{GH^2}{GE^2} \qquad (3\text{-}49)$$

$$\Delta' M_2 = \overline{EF}\frac{\overline{GH}}{GE}\frac{\overline{EH}}{GE} \qquad (3\text{-}50)$$

Se llega a:

$$\Delta' M_1 = S_{11}\frac{P_1^{obs} - P_1^{pre}}{S_{11}^{\ 2} + S_{12}^{\ 2}} \qquad (3\text{-}51)$$

$$\Delta' M_2 = S_{12}\frac{P_1^{obs} - P_1^{pre}}{S_{11}^{\ 2} + S_{12}^{\ 2}} \qquad (3\text{-}52)$$

Donde:

$$P_1^{pre} = S_{11}M_1^A + S_{12}M_2^A \qquad (3\text{-}53)$$

Estos son los resultados de las correcciones sobre el modelo para el primer rayo. Estas expresiones representan simplemente la proyección de un modelo estimado a uno de los hiperplanos, hasta acá lo que se ha hecho es movernos del punto A (punto de coordenadas iniciales arbitrarias) al punto B que es un punto de proyección a el primer hiperplano (figura 18), lo que sigue es moverse hasta el punto I, que es la proyección de el punto B hasta el hiperplano 2, esto se puede hacer usando los datos que se tiene para el segundo rayo.

Si se alternan las proyecciones del modelo estimado entre los dos hiperplanos entonces las actualizaciones del modelo estimado pueden converger al punto X, como se muestra en la siguiente figura 19.

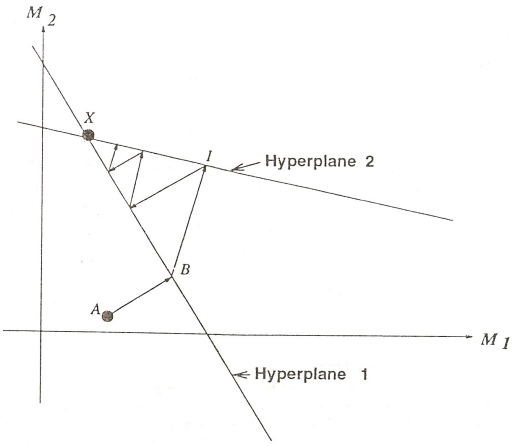


Figura 19. Convergencia del algoritmo de Kaczmar´z en un modelo de dos celdas con dos rayos que las atraviesan. Tomado de Fitterman 1994

3.1.2.4 INVERSION ART

ART es un implementación derivada directamente del algoritmo de Kaczmarz, La Técnica de Reconstrucción Algebraica (*Algebraic Reconstruction Technique*) es uno de los algoritmos más comunes en tomografía sísmica con trazado de rayo, este puede ser usado para resolver la ecuación 3-28. Como lo indica el diagrama de flujo de la figura 17, el primer paso es configurar un modelo inicial, este es un vector que contiene un valor del parámetro físico de interés para cada celda del modelo. Luego se entra al bucle iterativo, donde se realiza el modelamiento directo, el modelo inicial es usado para hacer el trazado de rayo, usando la ecuación 3-26 se calculan los tiempos de viaje predichos:

$$P_{i pred} = \sum_{j=1}^{J} M_{j est} S_{ij} \qquad (3\text{-}54)$$

Una vez calculado el vector de tiempos de viaje predichos, este es comparada con el vector de tiempos de viaje observado, la tolerancia es usada para saber si es necesario hacer más iteraciones, en caso de que así sea, la ecuación 3-35 se usa para calcular la actualización del parámetro de interés para cada celda:

$$\Delta^i M_J = S_{ij} \frac{P_i^{obs} - P_i^{pre}}{\sum_{j'=1}^{j} \left(S_{ij}\right)^2} \qquad (3\text{-}55)$$

Luego, las actualizaciones deben aplicarse al modelo, cada vez el modelo cambia y se acerca a el modelo real, la ecuación 3-33 puede ser usada:

$$M^{(new)est} = M^{est} + \Delta^i M \qquad (3\text{-}56)$$

3.1.2.5 INVERSION SIRT

SIRT es otro algoritmo útil para resolver la ecuación 3-28. La Técnica de Reconstrucción Iterativa Simultanea (*Simultaneous Iterative Reconstruction Technique*) es muy similar a ART, algoritmo presentado en la sección anterior, la diferencia radica en que todos los I rayos son trazados a través del modelo, tal que todas las correcciones $\Delta^i M_J$ determinadas para los I hyperplanos son conocidas. Entonces un promedio de los $\Delta^i M_J$ con respecto al índice i es tomado para cada celda del modelo, para obtener un nuevo modelo estimado. Como en el algoritmo ART, el modelo estimado se actualiza, los datos de tiempos de viaje son calculados y se comparan con los tiempos de viaje observados.

Al igual que en ART lo primero es encontrar un modelo inicial, una vez el modelo inicial este disponible se inicia el bucle iterativo. Se usa el modelo inicial par hacer el trazado de rayo y calcular los tiempos de viaje predichos para cada uno de los i rayos:

$$P_{i^{pred}} = \sum_{j=1}^{J} M_{j^{est}} S_{ij} \qquad (3\text{-}57)$$

Luego, debe encontrarse la corrección para cada celda, solo que como se ve en la siguiente ecuación para la j-esima celda la corrección se promedia para cada rayo:

$$\Delta^i M_J = \frac{1}{W_j} \sum_{i=1}^{I} \Delta^i M_j \qquad (3\text{-}58)$$

$$\Delta^i M_J = \frac{1}{W_j} \sum_{i=1}^{I} S_{ij} \frac{P_i^{obs} - \sum_{j'=1}^{J} S_{ij'} M_{j'}^{est}}{\sum_{j'=1}^{J} S_{ij}^2} \qquad (3\text{-}59)$$

Donde W_j es el numero de rayos que se interceptan en la j-esima celda o la densidad de

rayos usado para obtener un promedio de la corrección ΔM_J . Por ultimo se determina el nuevo modelo estimado de las correcciones del modelo promedio ΔM_J , o:

$$M_j^{(new)est} = M^{est} + \Delta M_j \quad j = 1, \ldots\ldots, J \qquad (3\text{-}60)$$

La figura 18 ilustra la diferencia en la convergencia de los algoritmos ART y SIRT, nuevamente para el ejemplo mencionado en la sección 3.1.2.3 con 2 celdas y 2 rayos atravesándolas.

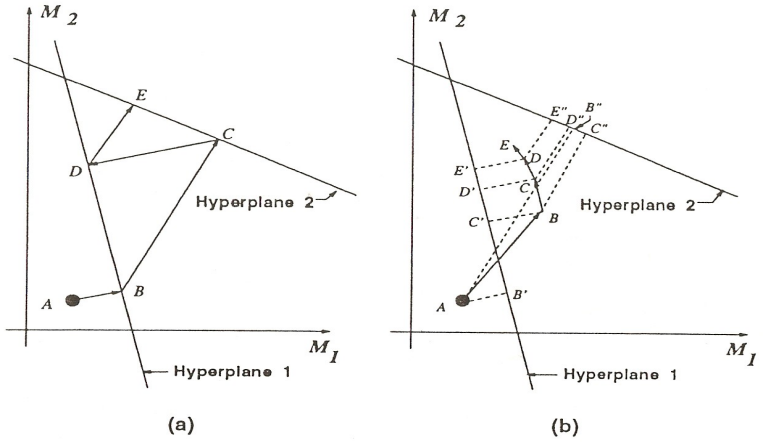


Figura 20. Convergencia del algoritmo de Kaczmar´z en un modelo de dos celdas con dos rayos que las atraviesan, Tomado de Fitterman 1994.

3.1.2.6 INVERSION LSQR

Este es otro algoritmo útil para resolver la ecuación 3-28, es un tipo de gradiente conjugado, converge rápidamente y es ideal para el aplicaciones tomograficas, pues es particularmente efectivo para matrices con valores cero en la mayoría de sus elementos; además, comparado con los algoritmos ART y SIRT es más eficiente [Gloaguen and Marcotte, 2005]. La correlación de una iteración a otra y la derivada de la suma de el residuo son usadas como criterio de convergencia.

Siguiendo el algoritmo de Kaczmar'z, debe partirse de un modelo inicial, este modelo es llamado M_{est}, en el paso 1 del algoritmo se usa el M_{est} para calcular los tiempos de viaje predichos (modelamiento directo), en el paso 2 los tiempos observados y los tiempos de viaje predichos deben ser comparados, la diferencia es llamada error:

$$e = P_{obs} - P_{pred} \qquad (3\text{-}61)$$

La idea es encontrar el mínimo de:

$$e^T e=(P_{obs}-P_{pred})^T(P_{obs}-P_{pred}) \qquad (3\text{-}62)$$

donde A^T es la matriz transpuesta de A, cuando se evalúan en el mínimo, las derivadas parciales de la ecuación anterior, con respecto a la lentitud son cero.

$$\frac{(\partial(e^T e))}{(\partial s_j)}=2(L^T(P_{obs}-P_{pred}))_j=0 \quad j=1,\ldots,n \qquad (3\text{-}63)$$

Esto implica,

$$L^T P_{pred}=L^T P_{obs} \qquad (3\text{-}64)$$

Esta es la ecuación normal de mínimos cuadrados estándar.

Este es un tipo de problema que se conoce como mal condicionado, en particular según la cobertura de rayos se tienen zonas subdeterminadas (celdas en las cuales hay pocos rayos), sobredeterminadas (donde hay muchos rayos que la atraviesan) o indeterminadas (que no tienen ningún rayo). En estos casos el método de solución puede ser bastante inestable. Para esto se amplia la matriz y se añaden dos matrices adicionales: la matriz $\lambda\mathbf{C}_v$, donde $\mathbf{C}_v$ es la matriz de covarianza vertical y $\lambda\mathbf{C}_h$, donde $\mathbf{C}_h$ es la matriz de covarianza horizontal. Estas matrices tienen como tamaño el número de parámetros del modelo y lo suavizan de forma que un rayo no afecta solamente las celdas por las que pasa, sino también las celdas cercanas. El coeficiente λ determina el ancho de la zona de suavizamiento.

Adicionalmente puede incluirse una matriz de amortiguamiento $\varepsilon\mathbf{G}$, que es una matriz diagonal cuyo tamaño es el número de parámetros del modelo. El coeficiente ε controla la amplitud de las perturbaciones en la solución. El sistema final tendría la forma:

$$\begin{pmatrix} \frac{1}{\sigma_{\mathbf{d}}}\frac{\partial \mathbf{t}}{\partial \mathbf{u}} \\ \lambda_v \mathbf{C}_v \\ \lambda_h \mathbf{C}_h \\ \epsilon \mathbf{G} \end{pmatrix} \Delta\mathbf{m} = \begin{pmatrix} \frac{\Delta \mathbf{t}}{\sigma_{\mathbf{d}}} \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad (3\text{-}65)$$

capítulo 4

ANALISIS DE INCERTIDUMBRE

Introducción

Un modelo de velocidad producto de una inversión, usando alguno de los métodos presentados en el capítulo 3, queda incompleto sin una medida del error asociado. Para saber qué tan confiable es el modelo solución, en este capítulo se presentan metodologías de análisis de incertidumbre. Debido a la naturaleza del problema inverso, los parámetros de interés no tienen solución única; así, que se usan 4 técnicas estadísticas: (1) Jackknifing (sección 4.1) que sirve para estimar las zonas del modelo más sensibles a la falta de datos en la inversión; (2) Checkerboard (sección 4.2) que sirve para estimar las zonas mejor resueltas en el modelo solución; (3) Monte Carlo (sección 4.3) que sirve para evaluar la sensibilidad de la solución obtenida a variaciones en el modelo inicial; (4) Bootstrapping (sección 4.4) que sirve para evaluar la incertidumbre en los tiempos de viaje. Los códigos creados para aplicar estas técnicas se presentan en los Anexos: 2, 3, 4 y 5.

4.1 JACKKNIFE

Jackknifing fue inventado por Quenouille en 1956 y desarrollado por Tukey en 1957, y es un procedimiento para estimar la incertidumbre de todos los parámetros del modelo [Lees & Crosson 1989; White & Clowes 1990]. La idea es invertir un determinado número de veces, subconjuntos incompletos del juego de datos y medir la influencia en el modelo solución de los datos faltantes.

Por tanto los datos de tiempo de viaje deben ser aleatoriamente divididos en un número de subconjuntos de igual tamaño, de forma que cada tiempo de viaje aparezca sólo en 1 subconjunto.

Luego el juego de datos completo menos cada subconjunto es invertido, el resultado de cada inversión da un sentido de la importancia de los datos que faltan en la solución y un estimado de la incertidumbre de cada parámetro puede ser calculado.

4.2 CHECKERBOARD TEST

Checkerboard al igual que Jackknifing (sección 4.1), hace parte de las técnicas de evaluación de modelos indirectas, pues se refiere a medidas que están relacionadas a parámetros tales como la resolución o la incertidumbre del modelo, pero sin la derivación de modelos adicionales que satisfagan los datos reales [Zelt, 1999], es una de las técnicas comúnmente aplicadas a inversión de tiempos de viaje y a estudios de tomografía.

Checkerboard estima la resolución espacial del modelo completo [Zelt, 1999], y consiste en sobreponer un patrón de altas y bajas anomalías sobre el modelo final, que debe usarse luego como modelo inicial, este modelo se invierte y la solución debe ser muy similar al original en regiones donde la solución es única.

Una medida de qué tan bien una región del modelo es resuelta está relacionada a que tan bien el patrón de anomalías conocido es recobrado en la inversión [Zelt, 1999]. El modelo solución final proporciona la cobertura de rayos y las anomalías la heterogeneidad que se quiere invertir.

4.3 ANALISIS MONTE CARLO

Los métodos Monte Carlo son importantes en el análisis de problemas inversos, donde expresiones analíticas no puede ser usadas directamente para relacionar los datos con los parámetros de modelo. Estos métodos pueden dividirse en dos categorías: métodos de muestreo y métodos de optimización [Mosegaard and Sambridge, 2002]. Los métodos de muestreo son útiles cuando se está interesado en conocer medidas de la resolución y de la incertidumbre de la solución. Los métodos de optimización son útiles en la búsqueda de la optimización global de la solución.

Dado el interés de esta investigación se usa solo el enfoque de muestreo Monte Carlo, estos métodos están basados en el muestreo aleatorio del espacio del modelo [Mosegaard, 1998]. La implementación consiste en muestrear el espacio de las posibles velocidades que puede tener del modelo inicial, entonces se conforma cierto número de modelos iniciales que deben ser luego invertidos cada uno, usando el mismo conjunto de tiempos de primeros arribos. Un estimado de la sensibilidad de la solución a variaciones en el modelo inicial puede ser calculada.

4.4 BOOTSTRAPPING

Bootstrapping fue desarrollado por Efron en 1979 para el computo de errores estándar. Es una técnica de computación intensiva, puede considerarse como un tipo especial de simulación, porque simula a partir de una población, basado en los datos. Puede usarse para asignar medidas de precisión a un estadístico estimado y para asignar intervalos de confianza [Efron and Tishirani, 1993]. Además para evaluar la incertidumbre de la velocidades en imágenes tomograficas [Lanz and Maurer, 1998].

Para ilustrar el método se incluye una representación esquemática, la figura 21 y el procedimiento se resume en los siguientes 4 pasos (Sacchi, 1998):

1) Se parte de N muestras de una variable genérica: $x = (x_1, x_2, x_3, \ldots, x_N)$ en nuestro caso: x_i, representa el valor de tiempo de viaje para un par fuente-receptor.

2) Se calculan las B muestras Bootstrap: donde una muestra Bootstrap se establece

así: $x_i = x_k$ donde i = 1, 2, 3 , . . . , N y k es una variable aleatoria uniforme; es decir, una muestra Bootstrap es un conjunto de tiempos de viajes perturbados aleatoriamente del un conjunto inicial.

3) Evaluar el estadístico de interés: cada muestra Bootstrap deber ser evaluada en el estadístico de interés, esto es: $\phi_i' = tomo(t_i')$ donde i = 1, 2, 3 , . . . , B esto puede ser, una ecuación analítica o una serie de procesos con diversas ecuaciones matemáticas, en nuestro caso, todo el proceso, es la inversión (tomo), nuevamente por cualquiera de las metodologías antes presentadas. Entonces ϕ_i' es el nuevo valor de velocidad que toma el nodo i-esimo después del proceso de inversión.

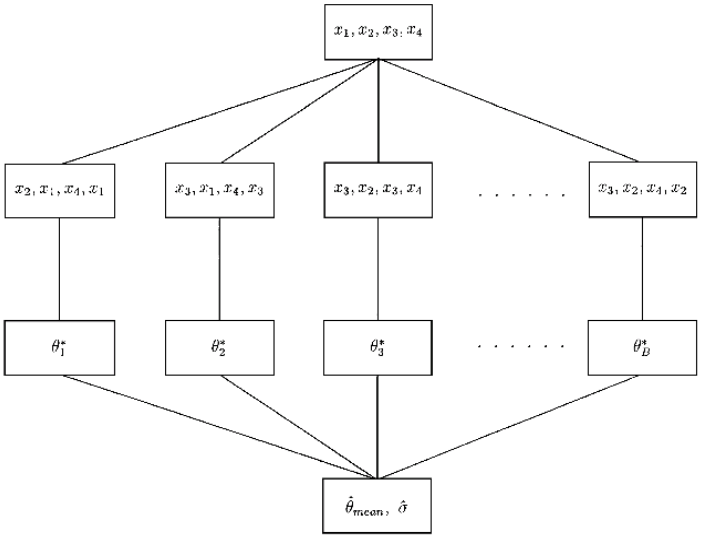


Figura 21. Representación esquemática del método Bootstrapping (Tomado de Sacchi, 1998)

4) Estimar la media y desviación estándar: después de invertir las B muestras bootstrap. Puede construirse un tomograma con los valores de velocidad promedio en cada nodo; además puede calcularse de desviación estándar, con la siguiente ecuación:

$$\partial' = \sqrt{\frac{1}{B-1}\sum_{i=1}^{B}(\phi' - \langle \phi' \rangle)^2}$$

$$\text{donde } \langle \phi' \rangle = \sum_{j=1}^{B} \phi_j / B$$

capítulo 5

ANALISIS E INTERPRETACION DE RESULTADOS

Introducción

En este capítulo se presenta la inversión del juego de datos sintético del capítulo 2 y del juego de datos real de la línea 1040, del programa sísmico Uribante 2005. Luego, cada una de las metodologías presentadas en el capítulo 4 es aplicada a ambos modelos de velocidad solución. En las secciones 5.3 a 5.6 se hace una descripción de como cada metodología es implementada con el uso de los programas de los anexos 2 a 5 y se discuten los resultados obtenidos de cada una de ellas.

5.1 PICADO DE PRIMEROS ARRIBOS

Como se mencionó en el capítulo 3, el método de tomografía construye un modelo de velocidad a partir de información de primeros arribos, así que el primer evento registrado en cada traza debe "picarse", es decir, digitalizar el tiempo que le tomó a la onda en viajar desde la fuente hasta cada receptor, para cada uno de los disparos del set. Para los picados se utilizó el software Promax, que organiza la información en un fichero ASCII, donde la información se organiza por paquetes de receptores para cada disparo, como sigue:

```
SHOT        1601   1175189   1388580   115.55    12.80    10.51
TRACE        1556   1174599   1388927   155.41     275.85
TRACE        1557   1174613   1388921   155.19     270.18
TRACE        1559   1174639   1388907   155.57     255.50
   .
   .
   .
TRACE        1999   1180512   1385896   595.54    1683.15
TRACE        2000   1180525   1385889   590.69    1686.64
TRACE        2001   1180538   1385882   585.93    1691.73
SHOT        1605   1175246   1388562   114.77    12.80    10.56
TRACE        1556   1174599   1388927   155.41     296.41
TRACE        1557   1174613   1388921   155.19     290.60
TRACE        1559   1174639   1388907   155.57     278.01
```

Las líneas que inician con: "SHOT" contienen información del punto donde se realizo en disparo, estos datos se organizan así:

a) "SHOT" en las columnas1 a 4
b) ID del disparo
c) Coordenada X de la estación de disparo
d) Coordenada Y de la estación de disparo
e) Coordenada Z de la estación de disparo
f) Profundidad del disparo (en ft)
g) Tiempo de Uphole de el disparo en milisegundos

Las líneas de receptores contiene la siguiente información en el orden dado:

a) "TRACE" en las columnas1 a 5

b) ID del receptor
c) Coordenada X de la estación del receptor
d) Coordenada Y de la estación del receptor
e) Coordenada Z de la estación del receptor
f) Tiempo de primer Arribo en milisegundos

En las secciones 5.1.1 y 5.1.2 se muestra el proceso para el set sintético y el real respectivamente.

5.1.1 SET SINTETICO

En la figura 22a se muestra un disparo del set sintético, el picado se hizo automáticamente, una banda en los disparos, donde el programa busca los primeros arribos, debe definirse como se muestra en la figura 22b. La figura 22c muestra algunos disparos picados.

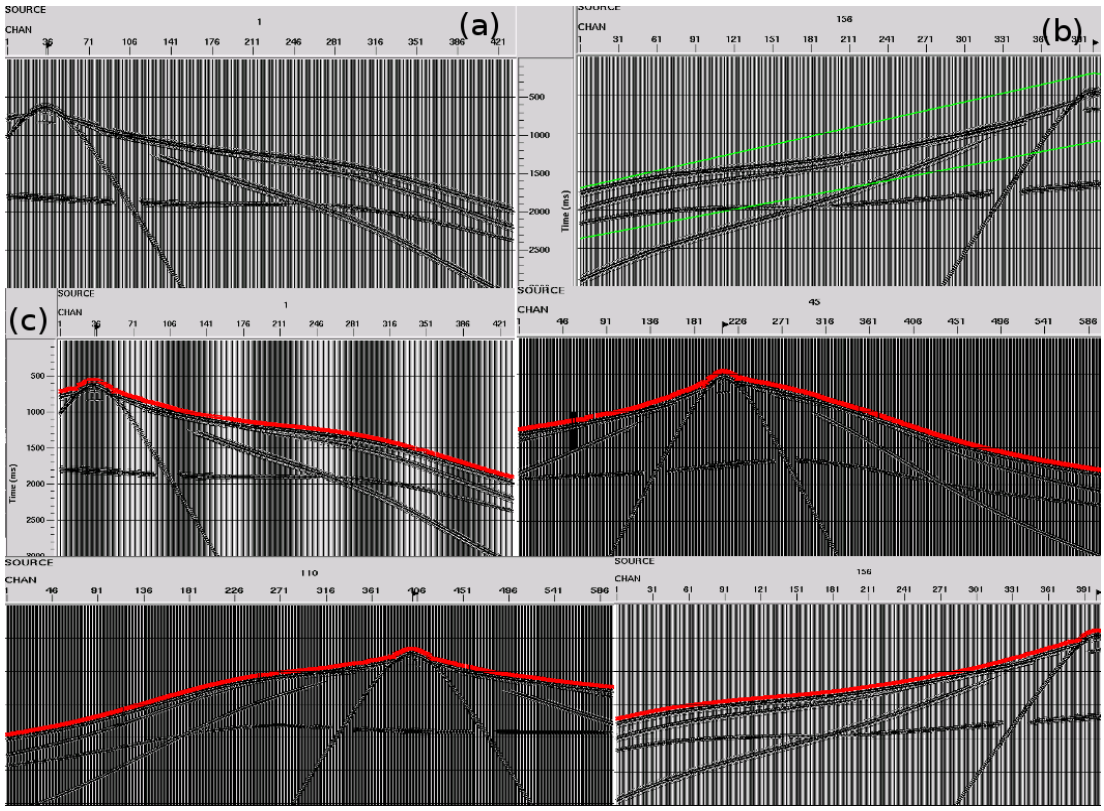


Figura 22. Picado de primeros arribos para el juego de datos sintético (a) Vista del disparo 1 del juego de datos (b) Definición de la banda para hacer el picado de primeros arribos (c) Vista de los disparos 1, 45, 110 y 156 del juego de datos sintético con los picados superpuestos.

5.1.2 SET REAL

El mismo procedimiento se aplicó al juego de datos real de la línea 1040, Uribante 2005. La figura 23 muestra un disparos del juego de datos y en la figura 24 algunos disparos picados.

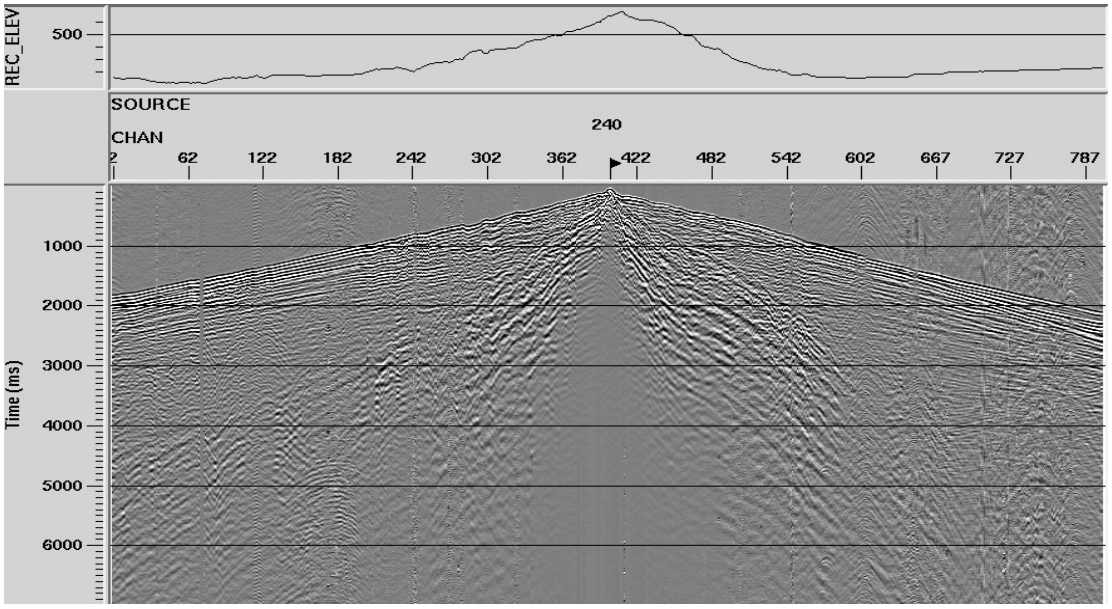


Figura 23. Vista del disparo 240 del juego de datos real.

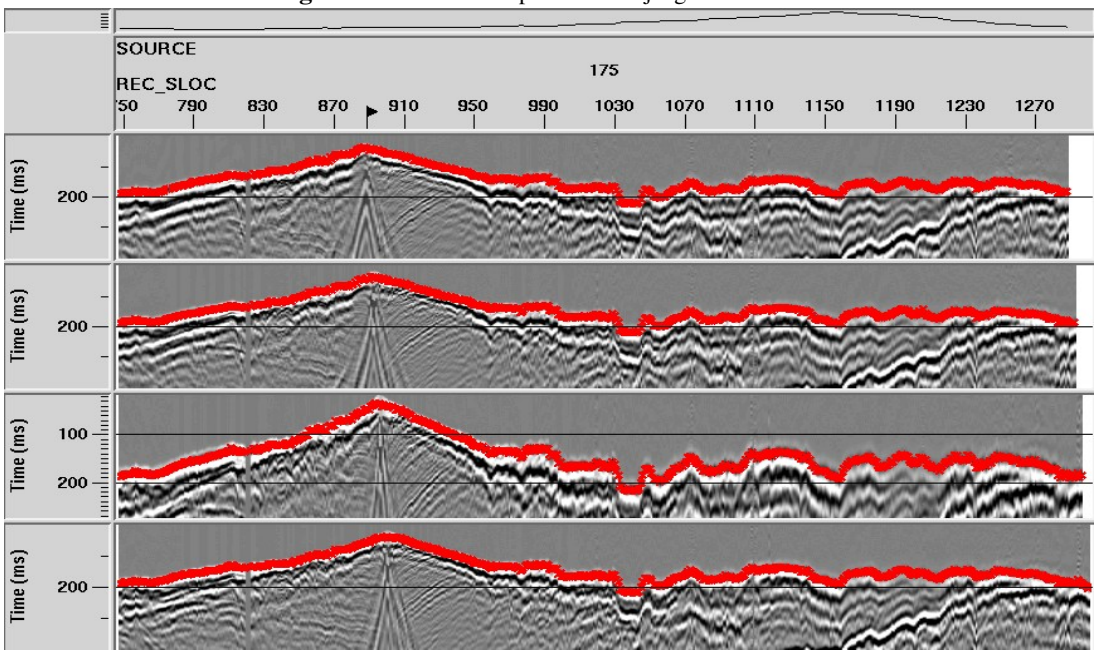


Figura 24. Picado de primeros arribos para el juego de datos real, Vista de algunos disparos picados del juego de datos real.

Finalmente, la figura 25 muestra el resultado del picado de primeros arribos para el juego de datos real, el eje horizontal representa el eje de las coordenadas X del modelo y el eje vertical los tiempos de primeros arribos en segundos.

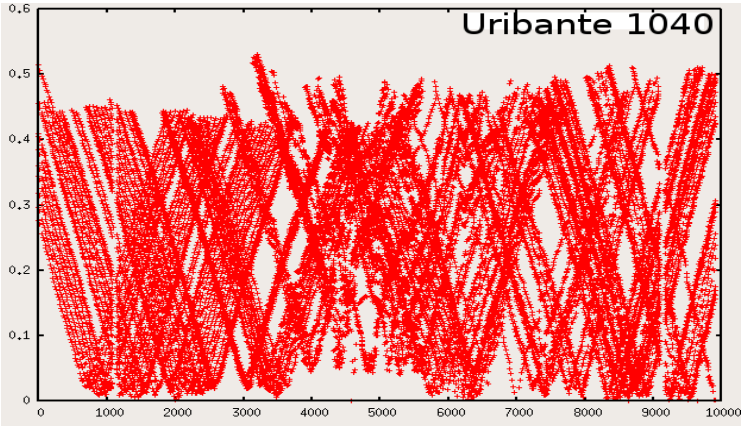


Figura 25. Primeros Arribos – juego de datos real.

5.2 INVERSION TOMOGRAFICA

El algoritmo de inversión usado fue desarrollado por el profesor Jun Korenaga, de la Universidad Yale, USA. El código fuente esta escrito en lenguaje c y es de uso libre, para más información ver [Korenaga et. al, 2000]

El código fue desarrollado para aplicaciones sismológicas, así que para hacer practico el uso del cogido a aplicaciones con datos sísmicos, se crearon los métodos: *"leerEntrada"* y *"genInputGrilla"* de la clase *"analisis"*, estos están escritos en lenguaje c++ [Anexo 2], la utilidad de este código es, que solo necesita el fichero de salida producto del picado de primeros arribos (sección 5.1) . Normalmente los software de procesamiento como Promax, permiten leer la información contenida y generar ficheros que contienen información necesaria para para realizar la inversión, como la geometría de la adquisición, la topografía de la línea, la grilla de velocidades con el modelo inicial, y los tiempos de viaje de primeros arribos. Ver Anexo 2.

5.2.1 CONSTRUCCION DEL MODELO INICIAL

El algoritmo de tomografía usado para realizar las inversiones es líneal; por tanto, si el modelo inicial está muy lejos de la solución, este puede volverse inestable y generar soluciones sin sentido físico. Así que para generar un buen modelo inicial también se

usan los datos de tiempos de viaje, se genera una gráfica como la mostrada en la Figura 26:

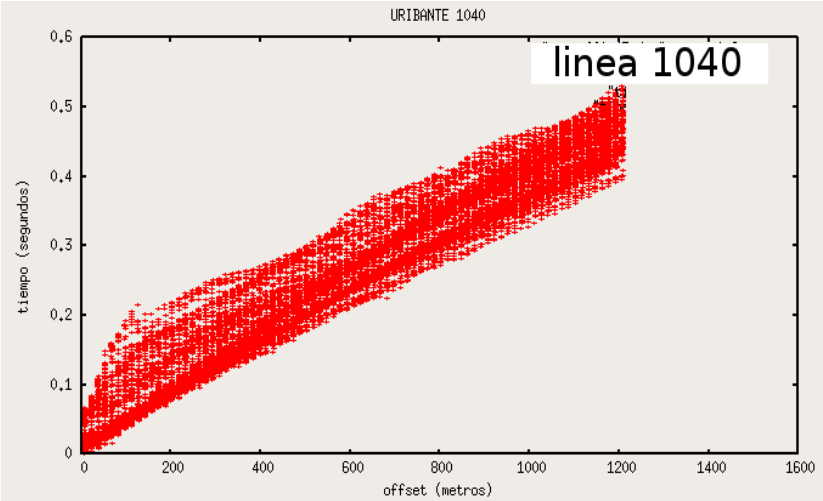


Figura 26. Gráfica de tiempo vs offset absoluto – Juego de datos Reales

La gráfica de tiempo versus offset da la información necesaria para calcular espesores y velocidades aproximadas, usando la expresión:

$$X_c = 2z\sqrt{\frac{(a_2+a_1)}{(a_2-a_1)}}$$

Donde z es el espesor de la capa refractante, a1 la velocidad de la capa superior, a2 de la capa inferior y Xc el offset de corte de las dos curvas para cada capa. Esta ecuación es obtenida de la relación geométrica mostrada en la figura 27 [Thorne, 1995]:

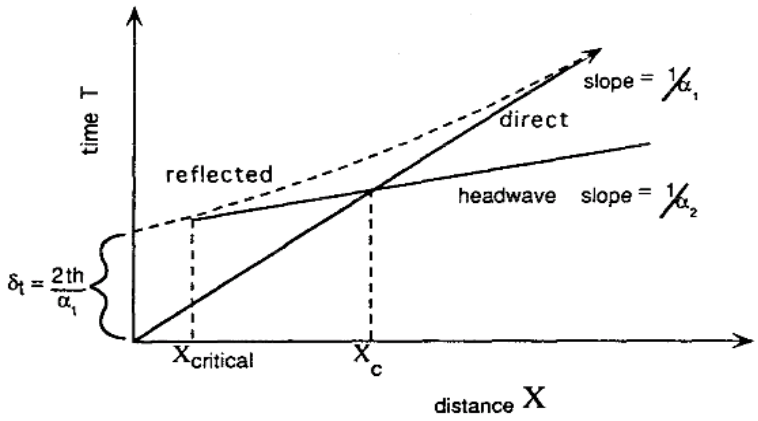


Figura 27. Curva de tiempo de viaje para ondas primarias, headwaves y reflejadas, Tomado de Thorne, 1995.

El método "*genInputGrilla*" usa la información obtenida para construir un modelo inicial, solo que cada capa es un gradiente de velocidad, detalles de la construcción puede verse en el código adjunto en el Anexo2.

5.2.2 CONSTRUCCION DE LA GRILLA

El tamaño de la grilla es importante y debe ser asignado en función de las necesidades del usuario. Depende que el detalle de la solución requerido y del muestreo de la adquisición, y debe tenerse en cuenta que una separación entre nodos demasiado pequeña produce tiempo de computo grande.

La separación entre nodos puede ser asignado especificando los valores nx y nz, estos son los argumentos primero y tercero respectivamente del método "*genInputGrilla*" de la clase "*analisis*".

El tamaño del modelo en la longitud horizontal es asignado automáticamente, esto se hace leyendo el archivo de entrada que contiene las coordenadas 3-D (xyz) y pasándolas al plano xz. La profundidad del modelo puede ser especificada usando segundo argumento del método "*genInputGrilla*" llamado: profundidad, que se entiende como la distancia bajo la menor cota topográfica.

Finalmente, el modelo inicial construido debe ponerse sobre la grilla, para esto el método "*genInputGrilla*" genera el fichero "*mymodelinicial.dat*", este se genera automáticamente usando la información leída del fichero de entrada y la suministrada por el usuario. Toda esta información se ordena de la siguiente manera dentro del fichero:

línea 1:
nx nz Vagua Vaire -Numero de nodos en x, z, Velocidades
línea 2:
X1 X2 X3 ... Xnx -Coordenadas en los nodos x
línea 3:
C1 C2 C3 ... Cnx -Cota Topográfica (negativa)
línea 4:
Z1 Z2 Z3 ... Znx -Coordenadas en los nodos Z
línea 5 a línea (4+nx):
Vi1 Xi2 Xi3 ... Xi,nx -Velocidad en el nodo(i,j)

La línea 4 que también es generada automáticamente, indica el valor de la topografía

para cada nodo de la grilla, como esta información no estaba disponible, se encuentra interpolando la información de fichero de entrada.

5.2.3 PROBLEMA DIRECTO

El software de tomografía de Korenaga usa un esquema de trazado de rayos híbrido, basado en el método gráfico [*Moser*, 1991] con local ray-bending refinado [*Moser et al.*, 1992]. El trazado de rayos sobre el modelo inicial sirve para calcular los tiempos de viaje predichos (ecuación 3-31), la figura 28 muestra un modelo inicial construido (sección 5.2.1), sobre este la grilla creada (sección 5.2.2) y se muestra como ilustración el trazado de rayos de cinco de los 153 disparos.

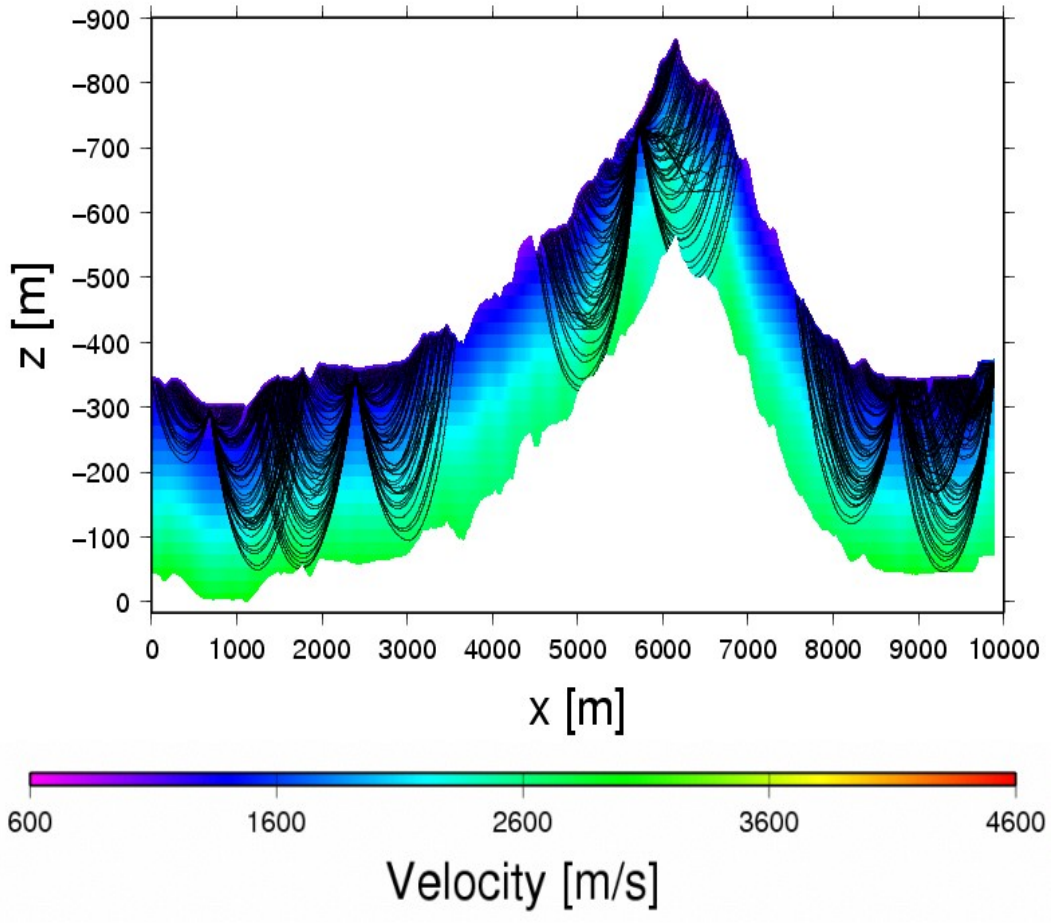
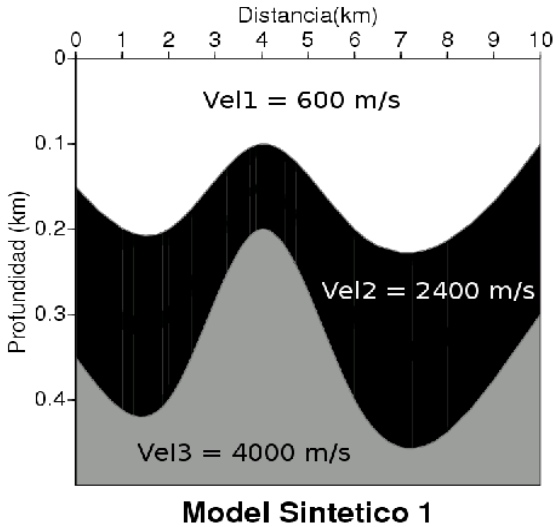


Figura 28. Vista de algunos disparos sobre el modelo inicial, línea 1040-Uribante

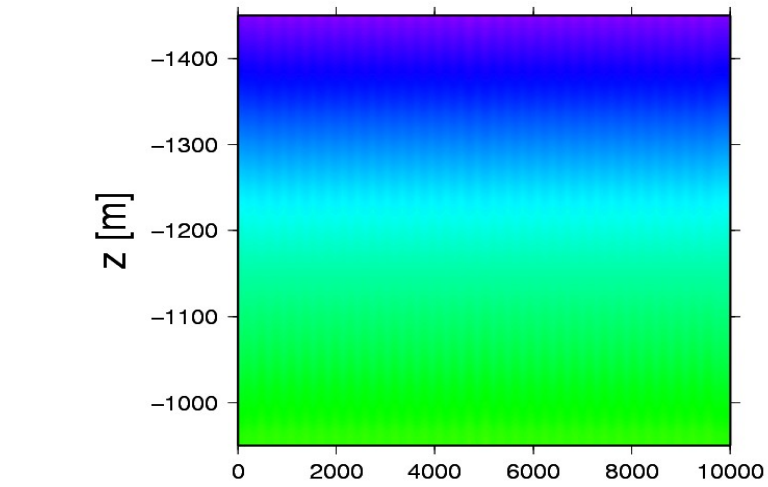
5.2.4 PROBLEMA INVERSO

En orden a estabilizar la inversión se requiere restringir las perturbaciones en velocidad y espesor, para esto en el método "*genInputGrilla*" de la clase "*analisis*" se crea el fichero "*correfile.dat*" que contiene un valor de longitud de correlación horizontal y vertical para cada nodo de la grilla. En el código se define una función de longitud de correlación horizontal que varia linealmente desde diez veces la separación horizontal entre nodos (nx), en la topografía; hasta veinte veces la separación horizontal entre nodos en la parte más profunda. La función de longitud de correlación vertical también varia linealmente desde una vez la separación vertical entre nodos (nz), en la topografía; hasta cuarenta veces la separación vertical entre nodos en la parte más profunda.

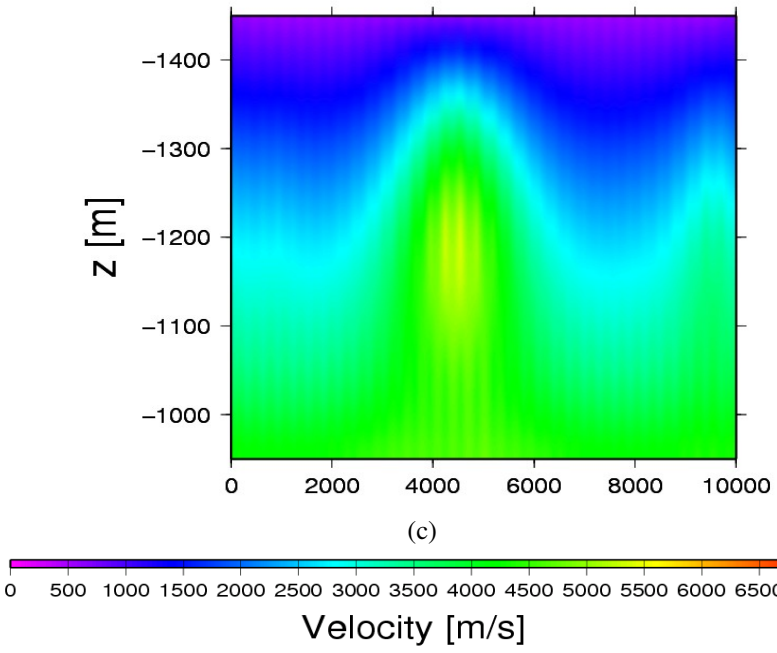
El proceso de inversión minimiza la diferencia entre tiempos de viaje predichos (sección 5.2.3) y tiempos de viaje observados (primeros arribos), para esto debe especificarse la tolerancia, que para este caso es 0.001. La figura 29 muestra el resultado de la inversión para el juego de datos sintético con offset variable. Finalmente el resultado de la inversión del set datos real se muestra en la figura 30.



(a)



(b)



(c)

Figura 29. Inversión del juego de datos sintético (a) Modelo sintético (b) Modelo inicial (c) Modelo solución

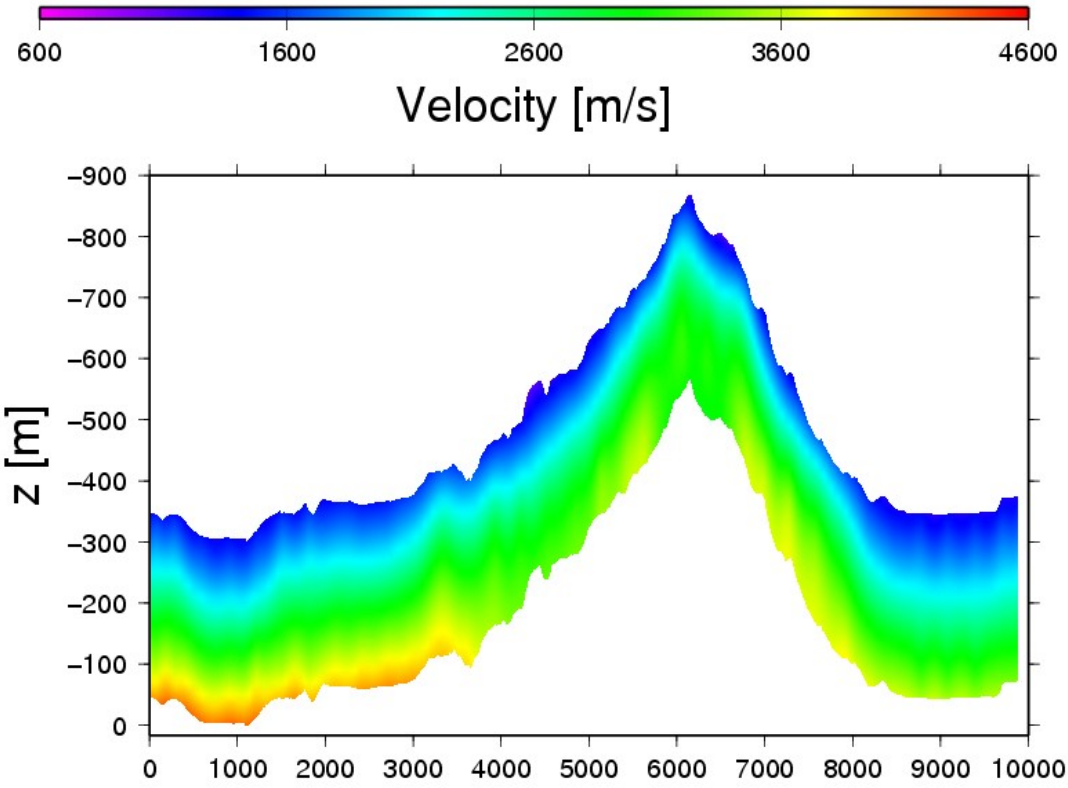


Figura 30. Inversión del set real (línea1040 – Programa sísmico Uribante 2005)

El modelo solución se obtiene usando 23088 registros de tiempo de primeros arribos, organizados en 153 disparos con offset variable, 10 metros de separación horizontal y vertical entre nodos. En general, el modelo de velocidad resultante muestra mayor espesor en la capa de baja velocidad, para sectores del modelo con menor pendiente y menor espesor para la capa de baja velocidad en zonas con mayor pendiente; lo cual tiene sentido, debido a que zonas con mayor pendiente suelen ser erosionadas y las de menor pendiente zonas de depositación; pequeños cambios en estas zonas pueden deberse a diferente tipo de suelo o condiciones diferentes de vegetación. En las siguientes secciones (5.3 a 5.6) se describe como aplicar las técnicas de análisis de incertidumbre presentadas en el capítulo 4 sobre el modelo invertido y se discuten los resultados obtenidos.

5.3 ANALISIS JACKKNIFFING

Jackknife consiste en invertir una serie de juego de datos incompletos para ver la influencia de datos faltantes en la solución. Para generar los n subconjuntos se creo el

método *"jacknifing"* de la clase *"analisis"*, la implementación consiste en eliminar cierto número de tiempos de arribos del set completo; para conformar cada subset se generan por cada disparo, un conjunto de números enteros aleatorios con distribución uniforme [0,nt-ne] que indican las trazas (primeros arribos) que se van a eliminar; nt es el número de receptores del disparo dado, menos ne, el número de picados para eliminar. La cantidad de primeros arribos que se eliminan de cada disparo (en), está determinado por el parámetro de entrada *shift* del programa *"jacknifing"* (Ver Anexo 2), siendo constante para cada disparo. Se requieren n realizaciones Jackknifing; n es un parámetro de entrada de la clase análisis llamado: *numol* (número de modelos a generar) y puede ser especificado. El método *"jacknifing"* sirve para generar n número de ficheros especificados por el usuario, este los crea y los nombra asi: ("*out**0**.dat", "out**1**.dat", ... , "out**n-1**.dat"*),.

Usando el programa *jacknifing.sh* del Anexo 2, que esta escrito en shell o interprete de comandos, el juego de datos completo es invertido y luego se invierten los n subset, el modelo de velocidad producto de la inversión para el set completo se almacena en un fichero llamado "*invtomoCom.dat"*, de igual forma los modelos de velocidad producto de la inversión de los n subset se almacenan en ficheros independientes y se nombran como sigue: ("*invtomo**0**.dat", "invtomo**1**.dat", ... , "invtomo**n-1**.dat"*). Esta información se usa para calcular la media de los residuales o diferencia entre el modelo solución del set completo y cada subset. Este *shell* usa los programas *jacknifingTest.cpp, jacknifing.h, jacknifing.cpp* escritos en c++ para calcular las diferencias entre los modelos invertidos [Anexos 2], y los shell *figrayos.sh, figJacknifing.sh* [Anexos 2] que usan GMT para graficar los resultados obtenidos. GMT es una colección de herramientas Unix de dominio publico para generación de gráficas científicas.

En la práctica, primero se aplicó jackknifing al juego de datos sintético [cuya generación se explicá en el capítulo 2], de cada subset se eliminan 10 trazas por disparo, 1560 en total, se hicieron 50 realizaciones de jackknifing, la figura 31 muestra la media de los residuales obtenida, los colores oscuros indican regiones donde la solución es más dependiente de los datos faltantes, que para este caso son los bordes del modelo, esto de debe a la menor cobertura de rayo en estos nodos; también en la zona central, donde se debe a que existe en esta zona mayor velocidad en el modelo (figura 29a).

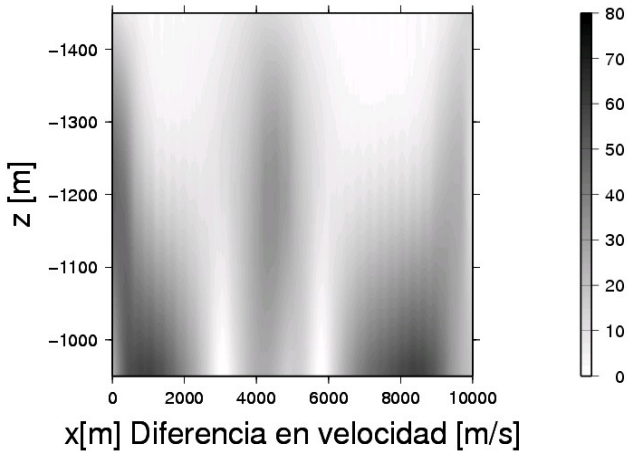


Figura 31. Análisis Jacknifing – Set Sintético

El mismo procedimiento se aplico al juego de datos real, en donde el número de trazas por disparo varia de 99 a 161 y se tienen 149 disparos. Así que se eliminan 10 trazas por disparo en cada subset, 1490 trazas en total para el set en cada realización. La figura 32 muestra la media de las diferencias entre el set completo y cada subset, para los 50 subset jackknifing.

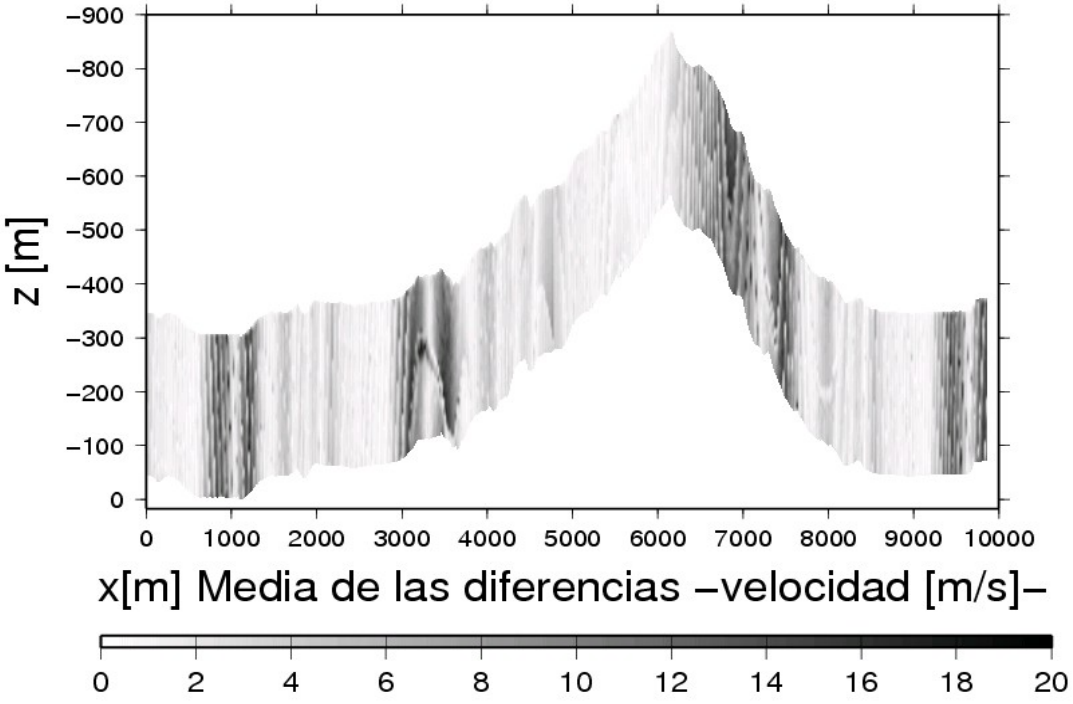


Figura 32. Análisis Jackknifing – línea 1040

En la figura 32 las zonas oscuras del modelo son las más dependientes de los datos eliminados, para comprobar que estas zonas no dependen de número de trazas eliminadas, se usaron los programas del anexo 2 para realizar más análisis con diferente número de trazas eliminadas (ne), como se aprecia en la figura 33 las zonas resultantes siempre son las mismas; pero, la magnitud de la media de las diferencias cambia, en las figuras 33 a, b, c, d se muestra los resultado de eliminar 10, 16, 20 y 50 trazas por disparo. Al eliminar 10 las velocidades encontradas pueden variar hasta ±20 m/s, al eliminar 16 hasta ±35 m/s, eliminando 20 hasta ±50 m/s y eliminando 50 que es para algunos disparos el 50% de las trazas, puede variar hasta ±140 m/s. Como los cambios en las magnitudes de los valores de velocidad son pequeños, comparados con los valores de velocidad del medio (entre 600-3600 m/s) figura 30; puede decirse, para este caso que el número de datos por disparo y el número de disparos son suficientes para obtener un modelo de velocidad confiable, suficientemente muestreado. Además se constata que esta prueba no indica la magnitud del error dada su variabilidad.

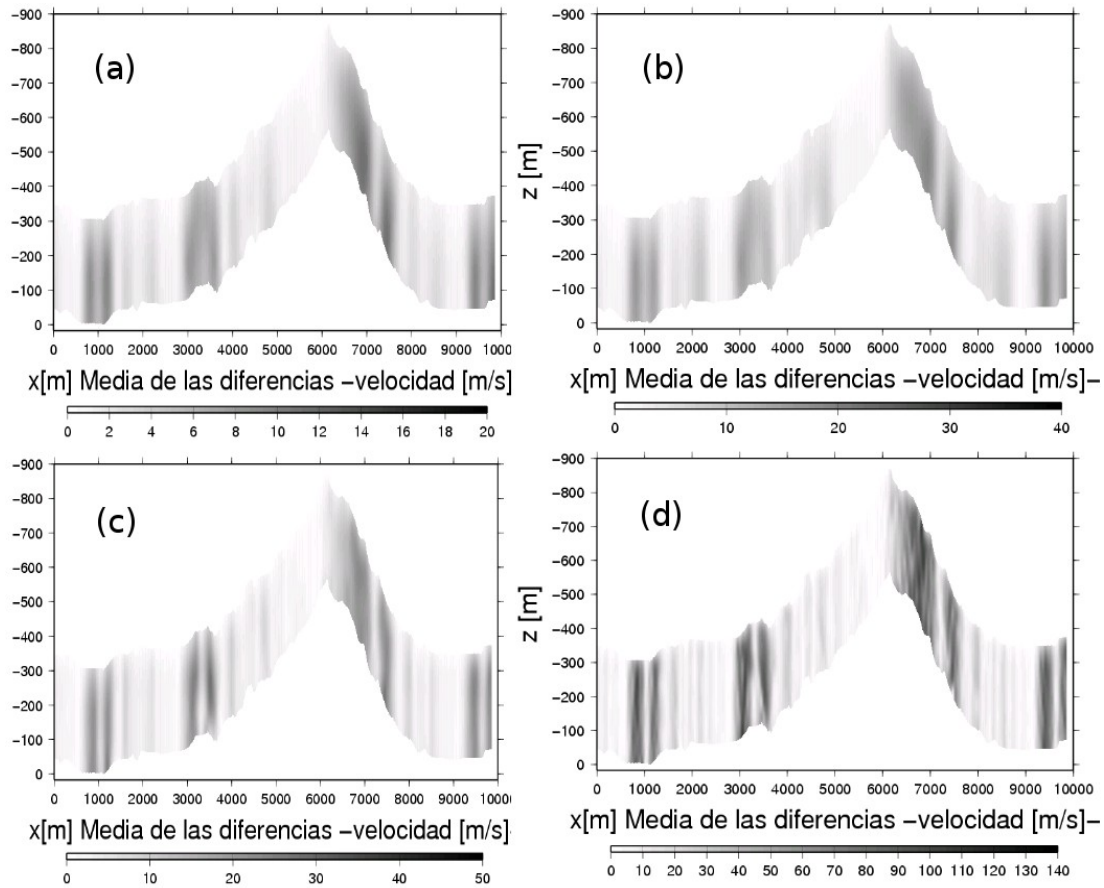


Figura 33. Varias realizaciones Jacknifing con diferente número de datos eliminados
(a) shift=10 (b) shift=16 (c) shift=20 (d) shift=50

5.4 ANALISIS CHECKERBOARD

Esta metodología consiste en perturbar el modelo de velocidad solución con anomalías positivas y negativas organizadas en forma de tablero de ajedrez, luego el modelo perturbado es nuevamente invertido, para realizar este análisis se usa el programa "*checkerboard.sh*" [Anexo 3]. Las zonas del modelo invertido que reconstruyan las anomalías, se dice que están bien resueltas; para distinguir estas zonas se usa la clase *"checkerboard"* (programas *checkerboard.cpp*, *checkerboardTest* y *checkerboard.h* del Anexo 3) que calcula la diferencia entre el modelo con anomalías checkerboard y el invertido. Las zonas con diferencias pequeñas están mejor resueltas, que las que presentan mayores diferencias.

Para ilustrar el procedimiento se usa el set sintético, el modelo de velocidad solución (*inv_sln.sintetico*) se perturba usando el programa *edit_smesh*, este genera un nuevo modelo (*modelcheckerboard.dat*) que contiene las anomalías, el patrón de anomalías se muestra en la figura 34a, como las velocidades del modelo solución varían en ambas direcciones, el resultado de aplicar las anomalías sobre el modelo se muestra en la figura 34b, luego este modelo es invertido, la solución se muestra en la figura 34c, por ultimo se grafica la diferencia entre ambos modelo (figura 34c), donde pueden apreciarse la zonas mejor y peor resueltas, para el caso del set sintético las zonas con mayores diferencias indican mayor incertidumbre en la solución.

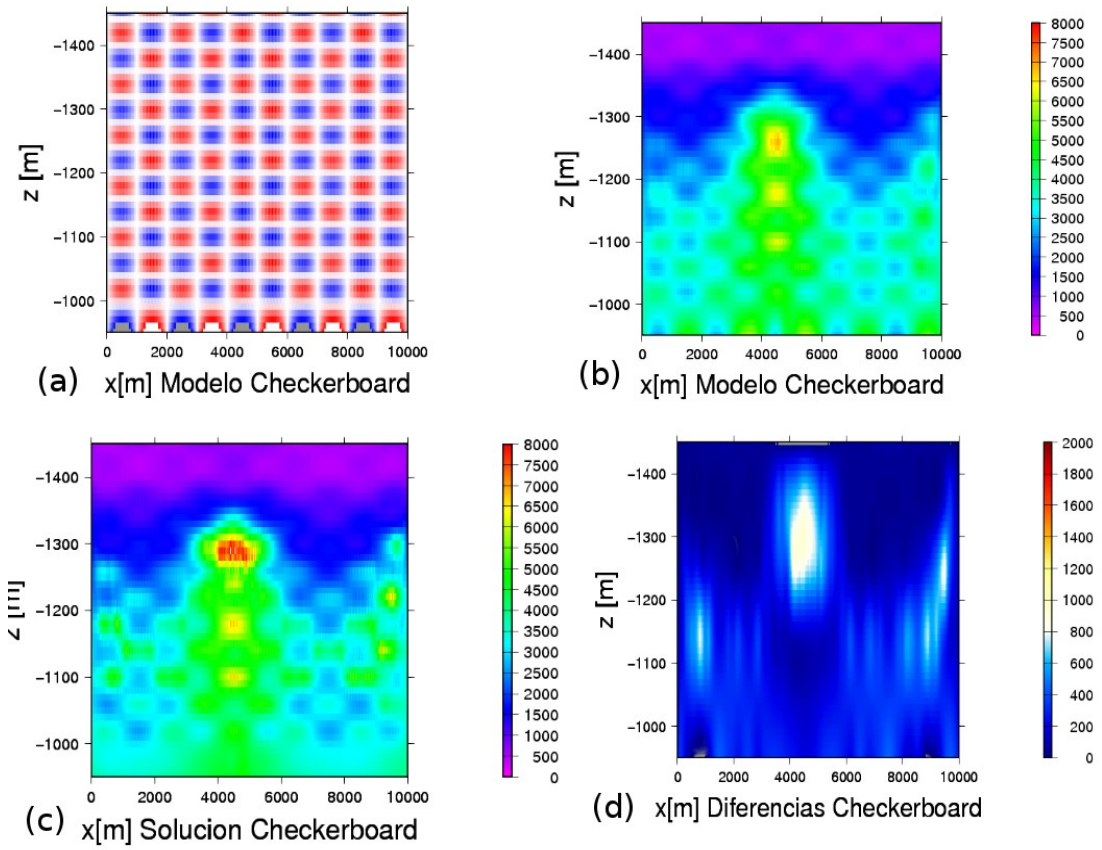


Figura 34. Implementación Checkerboard test sobre modelo sintético (a) Patrón de anomalías (b) modelo perturbado (c) modelo perturbado invertido (d) diferencias entre modelos

Checkerboard es útil para estimar la resolución del modelo, es decir, puede obtenerse una medida de la unicidad de la solución en ciertas zonas [Mosegaard, 1998]. Se realizan varias pruebas sobre el juego de datos real, donde el modelo de velocidad solución (Figura 30) es perturbado con una serie de patrones checkerboard (tablero de ajedrez) de diferente tamaño, la figura 35 muestra las diferencias entre el modelo invertido y el perturbado de cada prueba. (a) muestra el resultado usando un tamaño de anomalía de 20x20; (b) 100x100; (c) 150x150 y (d) 200x200. Las zonas con pequeñas diferencias son las que están mejor resueltas o donde la solución es única, cuando el tamaño de las anomalías es menor (caso a y b) las zonas mejor resueltas están más claramente definidas.

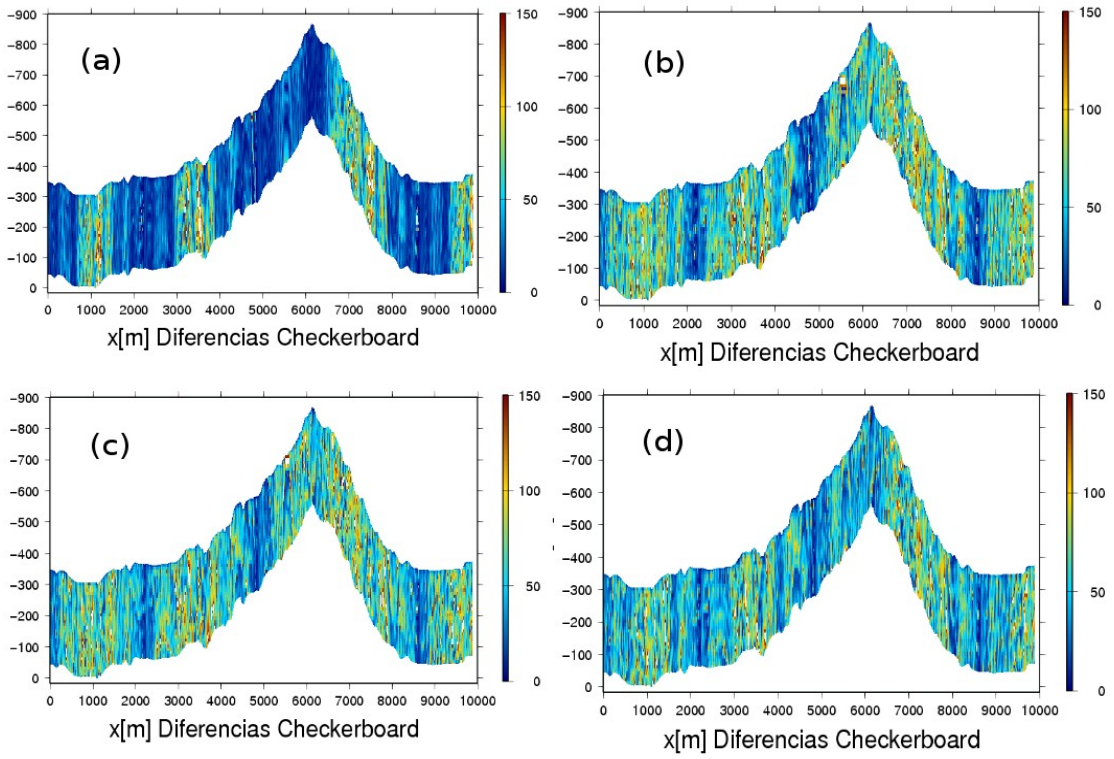


Figura 35. Implementación Checkerboard test sobre modelo de la línea 1040 (a) 20x20 (b) 100x100 (c) 150x150 (d) 200x200

5.5 ANALISIS MONTE CARLO

Esta metodología consiste en perturbar aleatoriamente el modelo inicial, para realizar este análisis se usa el programa *"montecarlo.sh"* del Anexo 4, este usa la clase *"montecarlo"* que esta definida en "*montecarlogen.cpp"*, *"montecarlogenTest.cpp"* y *"montecarlogen.h"* para perturbar los valores de velocidad del modelo inicial, el valor máximo a ser perturbado puede especificarse usando *"Ampl"* que es un parámetro de entrada de la clase; luego cada modelo perturbado debe ser invertido, el programa *"figmontecarlo.sh"* se usa para mostrar el resultado de la inversión de cada modelo perturbado; y el programa *"figmontecarloFIN.sh"* para graficar el promedio de todos los modelos pertubados invertidos y su respectiva desviación estándar, Ver Anexo 4.

Primero se realiza el análisis Monte Carlo sobre el juego de datos sintético, usando 5% como porcentaje de perturbación y un total de 55 realizaciones. El modelo inicial usado (figura 29 c) es diferente del verdadero (figura 29a) para poder comprobar los resultados obtenidos del análisis Monte Carlo. La figura 36a muestra el modelo verdadero, la figura 36b muestra el modelo solución promedio de las 55 realizaciones;

gracias a que se conoce el modelo verdadero pueden identificarse zonas donde la solución no corresponde con las velocidades reales (figura 37b). El objetivo del método Monte Carlo es identificar estas zonas y cuantificar los posibles errores.

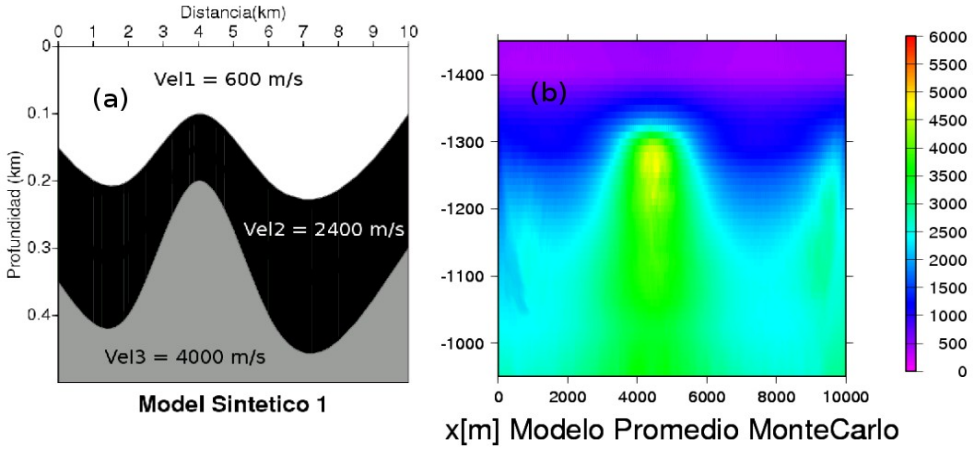


Figura 36. Implementación Monte Carlo sobre modelo sintético (a) Modelo verdadero (b) Modelos Promedio Monte Carlo

La figura 37b muestra la desviación estándar, pueden verse claramente identificadas zonas donde las velocidades no son coincidentes con la del modelo verdadero, pero no se identifican estas zonas en la parte baja del modelo. Esto es debido, a que en esta parte del modelo hay menor cobertura de rayo.

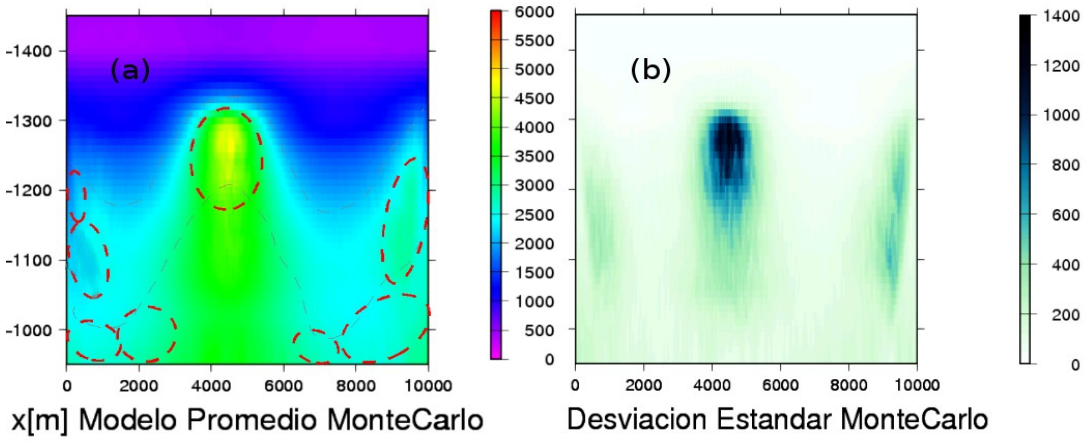
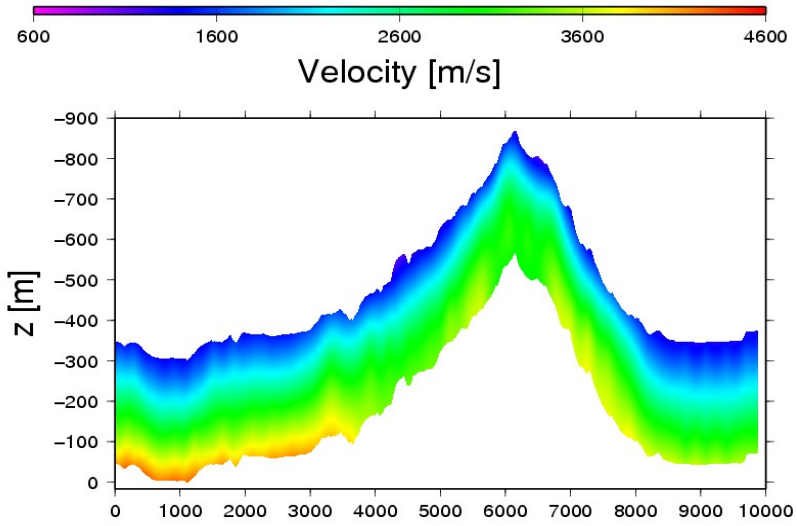


Figura 37. Desviación estándar Implementación Monte Carlo - modelo sintético

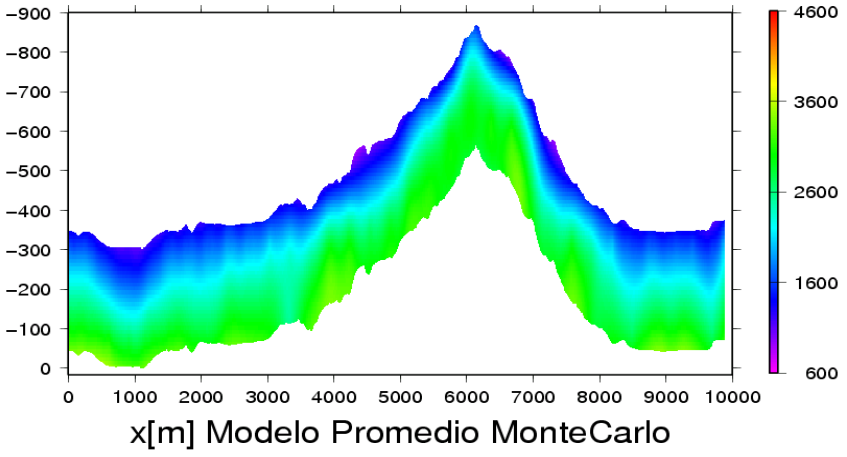
En las zonas identificadas el error se ha cuantificado; por ejemplo, en la parte derecha de la figura 37b puede observarse una de estas zonas, el error aproximado es 500m/s, restando este a las velocidad encontrada en la correspondiente zona de la figura 37a,

2900m/s aproximadamente, puede encontrarse la velocidad verdadera para esta zona 2400m/s (figura 36a). Puede decirse entonces, que a través del análisis Monte Carlo pueden identificarse y cuantificarse posibles errores en zonas del modelo que son más sensibles a variaciones en el modelo inicial.

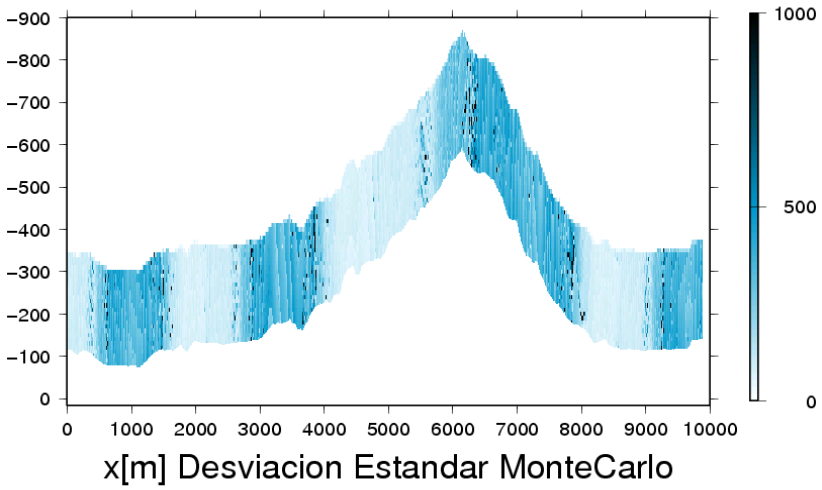
Para realizar el análisis Monte Carlo sobre el modelo inicial del caso real (línea 1040-Uribante) se uso un porcentaje de perturbación de 5%, se hicieron 70 realizaciones, usando los programas del anexo 4 se obtuvieron los resultados mostrado en la figura 38. La figura 38b muestra el modelo promedio de todas las realizaciones, como se esperaba es similar al modelo solución sin perturbaciones en el modelo inicial (figura 38a) obtenido en la sección sección 5.2.4. La figura 37c muestra la desviación estándar obtenida.



(a)



(b)



(c)

Figura 38. Implementación Monte Carlo sobre modelo de la línea 1040 (a) Modelo Solución sin perturbaciones en el modelo inicial (b) Modelo Promedio Monte Carlo (c) desviación estándar

El resultado del análisis Monte Carlo muestra que variaciones aleatorias en el modelo de velocidad inicial de hasta el 5% del valor original, generan variaciones en las velocidades del modelo solución de hasta ±1000m/s pero solo en pocos nodos de la grilla (puntos oscuros figura 38c), realmente se evidencian dos grandes grupos de zonas: (1) un grupo de zonas (color celeste en figura 38c) con variaciones de ±300m/s en promedio y (2) otro grupo de zonas (color más claro) con variaciones de ±100 m/s en promedio. Las zonas con mayores variaciones son coincidentes con las zonas más sensibles a la falta de datos en el análisis jackknifing (figura 32) y con las zonas peor resueltas o donde la solución es no única en el análisis checkerboard (figura 35a).

5.6 ANÁLISIS BOOTSTRAPPING

Es un método de remuestreo, evalúa la incertidumbre en las velocidades en el tomograma solución debida a variaciones en los tiempos de viaje, el error puede ser cuantificado. Para realizar este análisis se usa el programa *"bootstrapping.sh"* del Anexo 5, este usa la clase "*bootstrap"* definida en *"bootstrapgen.cpp"*, *"bootstrapgenTest.cpp"* y *"bootstrapgen.h"* para perturbar el conjunto de datos de tiempos de primeros arribos, un parámetro de entrada es *"Ampl"* que define el porcentaje máximo del valor original a ser perturbado aleatoriamente en cada dato, los programas *"figBoostrap.sh"*, *"figBootstrapDiff.sh"* y *"figBootstrapFIN.sh"* del anexo 5 son usados para graficar los resultados nuevamente usando GMT (Generic Mapping Tools).

El análisis se realiza sobre el juego de datos sintético del capítulo 2, las velocidades fueron perturbadas hasta 10%, 50 muestra bootstrap fueron generadas, luego cada una fue invertida y el modelo promedio de estas inversiones se muestra en la figura 39b; este es similar al modelo verdadero (figura 39a), la figura 39c muestra las zonas de mayor variación y los respectivos errores para cada una.

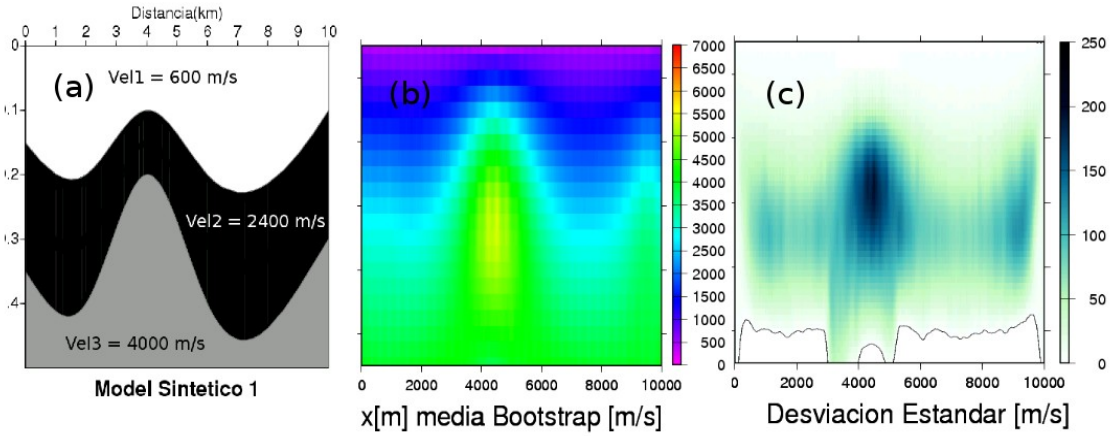
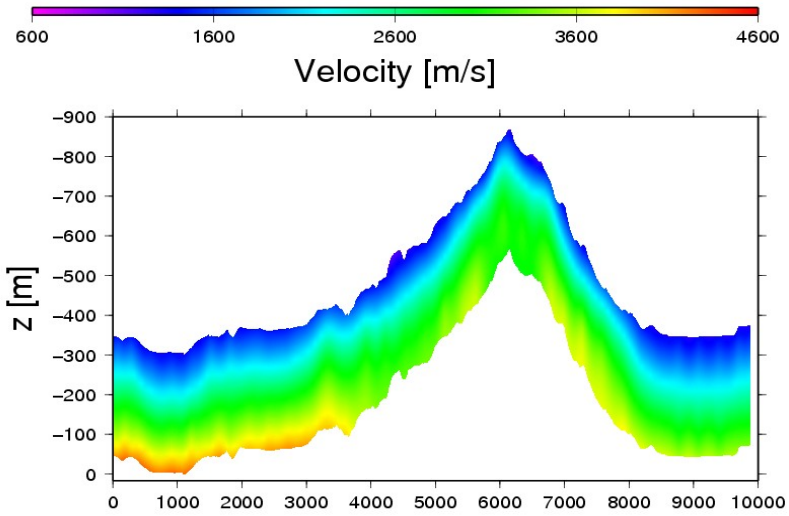
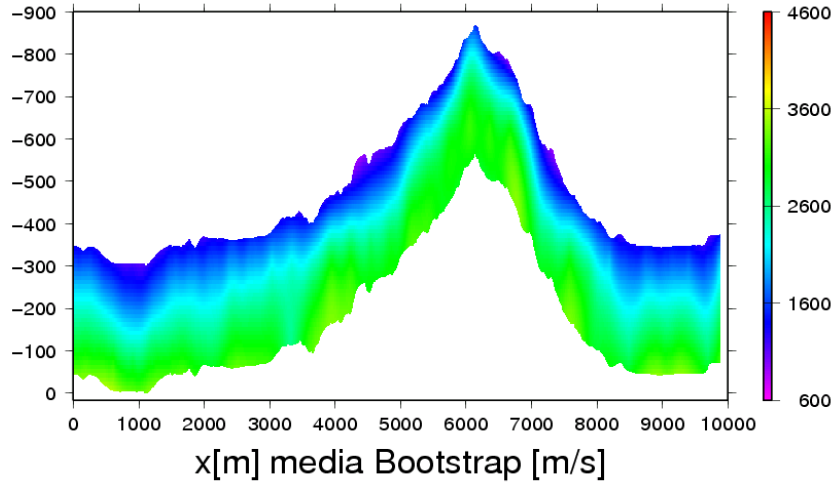


Figura 39. Implementación Bootstrap sobre modelo sintético (a) Modelo Verdadero (b) Modelo promedio de la 50 realizaciones Bootstrap (b) Desviación estándar

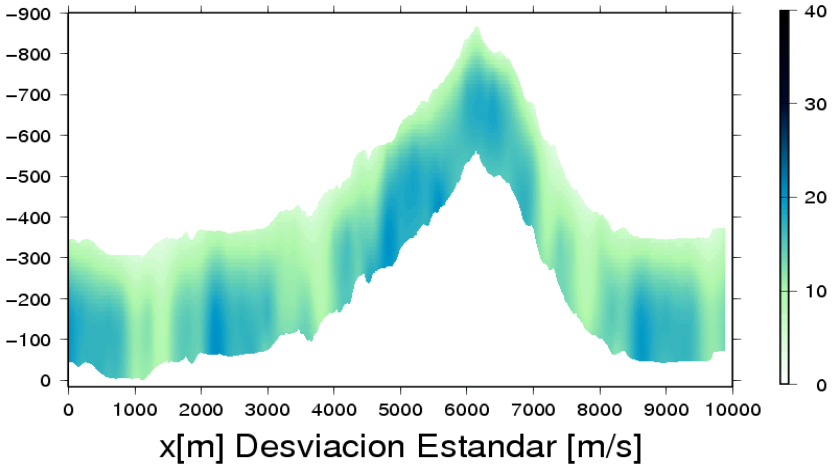
Para encontrar el error asociado en el modelo de velocidad solución del juego de datos real, se realizo el análisis usando 50 perturbaciones bootstrap y 10% como el porcentaje de perturbación máximo



(a)



(b)



(c)

Figura 40. Implementación Bootstrapping sobre modelo de la línea 1040 (a) Solución original (b) Media de las 50 realizaciones Bootstrap (c) Desviación estándar

El resultado (figura 40c) muestra que variaciones aleatorias en los tiempos de viaje de hasta el 10% del valor original, generan variaciones en las velocidades del modelo solución de hasta ±40m/s, ±12m/s en promedio, comparando la magnitud de las variaciones encontradas ±12m/s con las velocidades del medio entre 600m/s y 4000m/s, puede decirse que la incertidumbre asociada a los tiempos de viaje de primeros arribos usados en el proceso de inversión es despreciable. Como se esperaba el modelo solución no perturbado (sección 5.2.4) es similar al modelo solución promedio de todas las muestras bootstrap, figuras 40a y 40c respectivamente.

Cada técnica de análisis implementada evalúa la incertidumbre asociada al modelo solución invertido de forma diferente, jackknifing mide las zonas del modelo más

dependientes de los datos usados en la inversión; checkerboard las zonas del modelo mejor y peor resueltas o zonas donde la solución no es única; Monte Carlo la incertidumbre asociada al modelo inicial y bootstrapping la incertidumbre asociada a los tiempos de primeros arribos.

El análisis realizado sobre el caso real, los datos de la línea 1040 (cuenca del Catatumbo, Colombia), muestran que las zonas donde la solución no es única (zonas con mayores diferencias en la figura 41b) también son las zonas que suelen tener mayores variaciones por cambios de velocidades en el modelo inicial (zonas con mayor desviación estándar en la figura 41c) y son las más sensibles a la falta de datos en la inversión (zonas con mayores diferencias en la figura 41a).

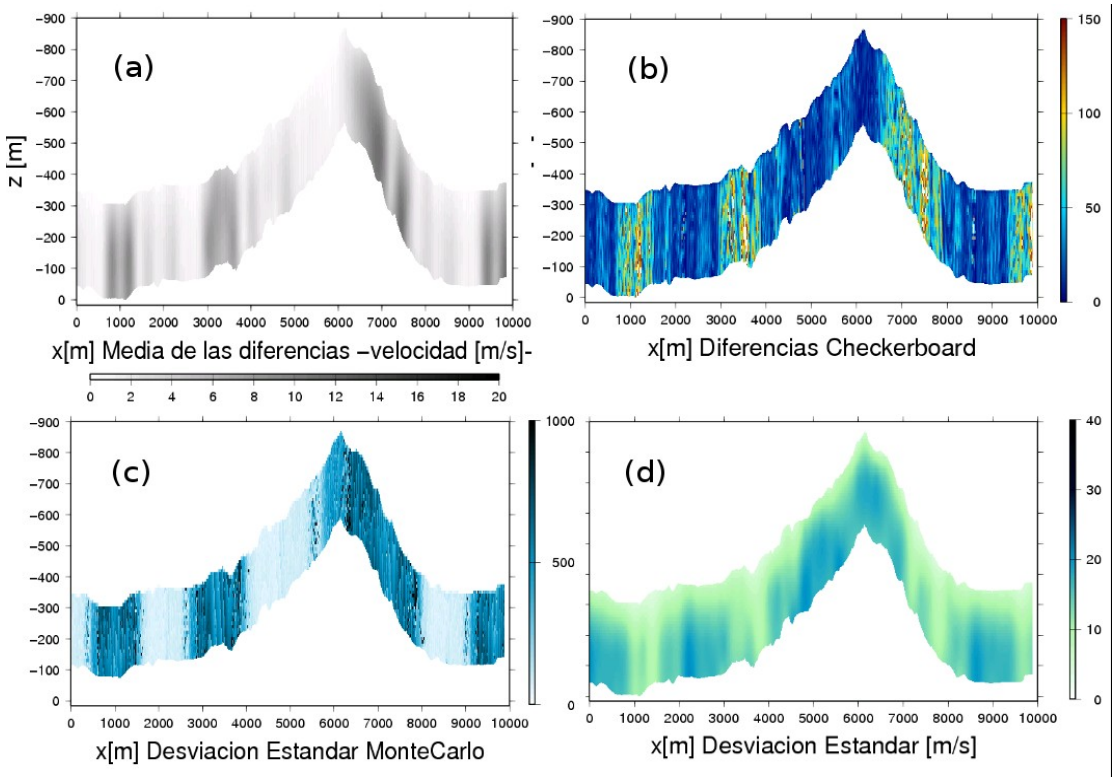


Figura 41. Resultados de técnicas de análisis de incertidumbre sobre modelo de la línea 1040
(a) Jackknifing (b) Checkerboard (c) Monte Carlo (d) Bootstrapping

Por medio del uso de las técnicas jackknifing, checkerboard, Monte Carlo se han identificado un conjunto de zonas donde la incertidumbre en la solución es mayor, para obtener valores de velocidad más confiables, debe usarse en el inversión información adicional, como: información de métodos no sísmicos, información de registro de pozo o información geológica de la zona. El análisis bootstrapping da una idea de la

magnitud de las variaciones que puede tener el modelo solución asociada a la incertidumbre en los datos de tiempos de primeros arribos, si estas variaciones son pequeñas comparándolas con las velocidades encontradas del medio, puede decirse que el modelo solución invertido es confiable.

5.7 CORRECCIONES ESTATICAS

El modelo de velocidad solución puede usarse para corregir imágenes sísmicas, los conceptos relacionados presentados en la sección 1.2.1 se usaron para crear el programa "correccion.sh", este programa calcula el tiempo que debe restarse o adicionarse de las estaciones del modelo para obtener el correcto posicionamiento en los reflectores. Las coordenadas y cotas de cada estación deben estar disponibles para realizar el calculo de las correcciones, para esto se implemento el programa "analisis" (analisis.cpp, analisisTest.cpp, analisis.h) del Anexo 6, este programa lee el fichero llamado "clientes2.geo" que contiene la información de cada receptor organizada por disparos y genera los programas "OUTSOU.dat" y "OUTREC.dat" que contienen el numero, la coordenada X y la cota de cada estación para fuentes y receptores respectivamente. Luego, estos ficheros y el que contiene el modelo de velocidades son leídos por la clase "correccion" definida en los ficheros: "delay.cpp", "delayTest.cpp" y "delay.cpp" (Anexo 6). Para calcular la corrección por estrato somero y la corrección por elevación la clase "correcion" usa los siguientes parámetros de entrada: (1) "lim" este valor define la velocidad limite hasta donde se considera el estrato somero, el programa realiza una interpolación doble sobre la grilla para encontrar el tiempo de propagación desde la superficie hasta donde se finaliza el estrato somero, (2) y (3) "dx" y "dz" se especifica la separación entre los nodos horizontales y verticales respectivamente usados en la inversión tomografica, (4) y (5) "Cotamax", "Cotamin" especifica los valores de cotas máximas y mínimas del modelo, (6) "datum" define el nivel de referencia hasta donde se calcula el la corrección por elevación, desde la base de la capa meteorizada, (7) "VelRemp" define la velocidad de reemplazamiento, que es la usada para calcular la corrección desde la base de la capa meteorizada hasta el datum o también llamada corrección por elevación. Finalmente, la corrección por capa meteorizada y por elevación calculas se usan o restan usando la notación especificada en la sección 1.2.1 el valor resultante es llamado corrección estática, es negativa cuando el datum esta bajo el estrato somero y positivo cuando esta por encima (figura 42). El programa genera los ficheros de salida "outSOUcorrection.dat" y "outRECcorrection.dat" que contiene el numero de estación y la corrección estática calculada, para las fuente y los receptores respectivamente.

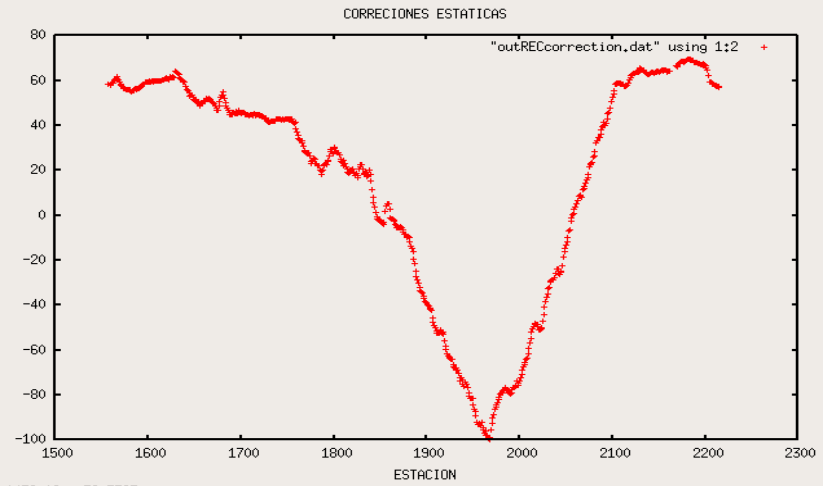


Figura 42. Correcciones estáticas calculadas para el modelo de la Linea 1040

Usando el modelo de velocidad obtenido del análisis Monte Carlo y los programas del anexo 6 se calcularon las correcciones estáticas para el segmento de la linea 1040 usado en esta investigación, estas correcciones se usan para obtener una imagen apilada (figura 43-derecha) y se compara con otra que fue obtenida usando correcciones estáticas de refracción (figura 43-izquierda). Puede verse en la imagen resultante de la corrección tomografica, mejor definición de reflectores alrededor de los 1500 ms

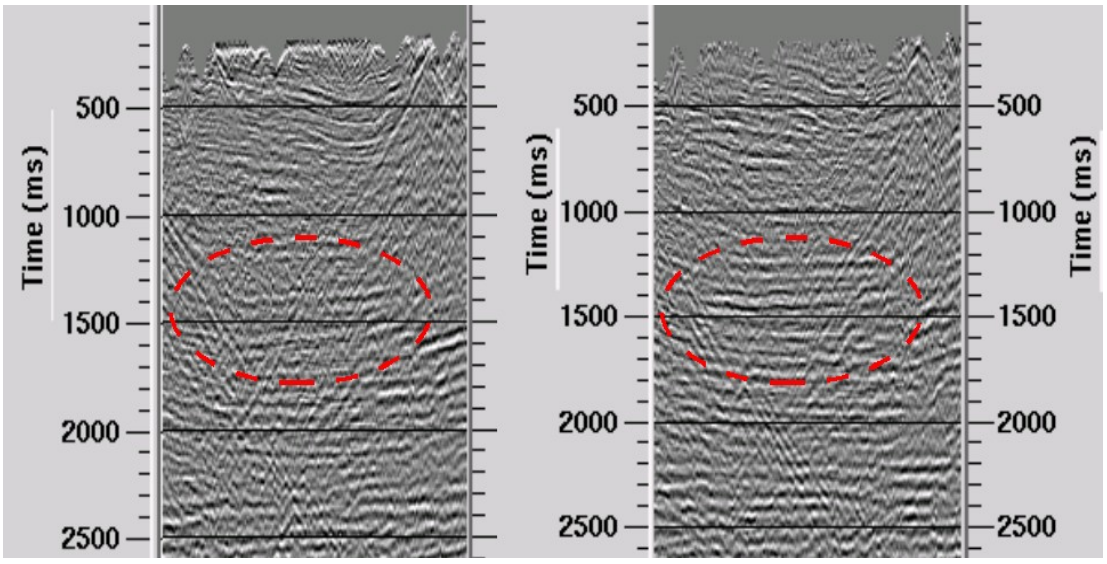


Figura 43. Imágenes sísmicas corregidas- segmento de la Linea 1040 – Uribante 2005
(izquierda) Aplicando estáticas de refracción (derecha) Aplicando estáticas obtenidas del modelo tomografico

6. **CONCLUSIONES**

- Tomografía sísmica es una herramienta útil para obtener modelos de velocidad del estrato somero a partir de información sísmica, el uso de técnicas de análisis de incertidumbre sugiere zonas donde el resultado de la inversión es menos confiable.

- El análisis Jackknifing identifica zonas del modelo más sensibles a la falta de datos en la inversión, si el número de datos eliminados por disparo cambia, estas zonas aun pueden identificarse, pero la magnitud de las diferencias varia, a mayor cantidad de datos faltantes mayor es la magnitud de las diferencias encontradas.

- El análisis Checkerboard identifica zonas del modelo solución mejor resueltas, a menor tamaño en el patrón de anomalías usado mejor pueden identificarse estas zonas.

- El análisis Monte Carlo identifica la variación en el modelo solución debida a variaciones en las velocidades del modelo inicial, los posibles errores pueden ser cuantificados calculando la desviación estándar de los modelos perturbados en todas las realizaciones Monte Carlo.

- El análisis Bootstrapping evalúa la incertidumbre asociada a los tiempos de primeros arribos usados en la inversión, puede cuantificarse y encontrarse la distribución espacial en el modelo, del error en las velocidades del modelo solución, calculando la desviación estándar de los modelos perturbados invertidos de todas las realizaciones Bootstrap.

- El uso de técnicas de análisis de incertidumbre sirve para identificar zonas del modelo con problemas de unicidad en la solución, esto sugiere zonas donde el uso de información adicional es necesario para mejorar la solución.

- El modelo de velocidad promedio producto del análisis Monte Carlo puede usarse con éxito para realizar correcciones por estrato somero y elevación, resultando en una mejor imagen sísmica profunda.

BIBLIOGRAFIA

BRIDLE R., LEY II. R,. AL-HOMAILI M., MADDINSON B., and JANSSEN K. 2003. Practical application of implementing a layer model with control from first breaks Saudi Arabia. **73rd** *SEG Meeting, Dallas, USA, Expanded Abstracts*, 1996-99.

BRIDLE Ralph., BARSOUKOV Nikolai., AL-HOMAILI Mohammad., LEY Robert., and AL-MUSTAFA Amemrah., Comparison state of the art near-surface models of a seismic test line from Saudi Arabia. *Geophysical Prospecting*, **54**, Pages 667-680, 2006.

COX M., 1999, Static Corrections for Seismic Reflection Surveys, *Geophysics References Series*, Number **9**, Society of Exploration Geophysics, Published 1999, Tulsa - Oklahoma, United States of America.

EFRON, B., 1979, Bootstrap methods: Another look at the Jackknife: *Ann. Statist*, **7**, Pages 1-26

EFRON B., and TIBSHIRANI, R., 1993, An introduction to the bootstrap method: Chapman and Hall.

FITTERMAN V. David., LINES R. Larry., 1994, Fundamentals of Seismic Tomography, *Geophysical Monograph Series*, Number **6**, Pages 1-43, Published 1994, United States of America.

GLOAGUEN E., MARCOTTE D., CHOUTEAN M., PERROUD H., Borehole radar velocity inversion using cokriging and cosimulation. *Journal of Applied Geophysics*, Volume **57**, Issue 4, July 2005, Pages 242 – 259.

HAMPSON, D., and RUSSELL, B., 1984, First-break interpretation using generalized linear inversion, J. Can. Soc. Expl. *Geophysics*, **25**, 40-50.

KORENAGA J., HOLBROOK W. S., KENT G. M., KELEMEN P. B., DETRICK R. S., LARCEN H. C., HOPPER J. R., DALH Jensen T., 2000, Crustal structure of the southeast Greenland margin from joint refraction and reflection seismic tomography, *Journal of Geophysical Research*, Vol **105**, NO. B9, Pages 21591 – 21614, September 10, 2000.

LANZ E., MAURER H., and GREEN A., 1998. Refraction tomography over a buried disposal site, *Geophysics*, Vol **63**, NO 4 (July- August 1998); Pages 1414-1433.

LEES, J.M. & CROSSON, R.S., 1989. Tomographic inversion for three-dimensional velocity structure at Mount St. Helens using earthquake data, *J. geophys. Res.,* **94**, 5716-5728.

McGAUGUEY J. W, and YOUNG P. R. Comparison of ART, SIRT, Least-Squares, and SVD Two-Dimensional Tomographic Inversions of Field Data. Pages. 74-77. Noranda Technology Centre, Queen's University, Canada.

MENKE, 2000, Case Studies of Seismic Tomography and Earthquake Location in a Regional Context -- Lamont-Doherty Earth Observatory of Columbia University.

MOHR, R. C. J, 1944, Soils of equatorial regions, with special references to the Netherlands East indies, J. W. Edwards, *Ann Arbor*, U.S.A., **766** pp

MOSEGAARD K., and SAMBRIDGE M., 2002, Monte Carlo analysis of inverse problems., Inverse Problems **18** R29-R54., *IOPscience*., IP Address: 200.119.3.209

MOSEGAARD K., 1998 Resolution analysis of general inverse problems through inverse Monte Carlo sampling., Inverse Problems **14** 405-425., *IOPscience*., IP Address: 200.119.3.209

MOSER, T. J., Shortest path calculation of seismic rays, *Geophysics*, **56**, Pages 59- 67, 1991

MOSER, T. J., NOLET G., SNEIDER R., Ray bending revisited, Bull, *Seismol. Soc. Am*., **82**, Pages 259-288, 1992

PHILLIPS S. and FEHLER M. "Traveltime tomography: A comparison of popular methods". *Geophysics*, Vol **56**, No 10 (October); P.1639-1649. Society of Exploration Geophysicists 1991.

QUENOUILLE, M. H. 19576, Notes on bias in estimation, *Biometrika*, **43**, Pages 353-360

SACCHI M. "A bootstrap procedure for high-resolution velocity analysis" *Geophysics*, Vol **63**, No 5 (September); P.1716-1725. Society of Exploration Geophysicists 1998.

SCALES J. Tomography Inversion via the conjugate gradient method. *Geophysics*, Vol **52**, No 2 (February); Pages 179-185. Society of Exploration Geophysicists 1988.

SALLARÈS V. and CHARVIS P., 2003, *Journal of Geophysical Research*, Vol. **108**, NO. B12, 2564, doi:10.1029/2003JB002431.

SHERIFF R. E, 199, Encyclopedic dictionary of exploration geophysics, Third edition, *Society of Exploration Geophysics*.

THORNE L., WALLACE. T., 1995, Modern Global Seismology., International Geophysics Series, Vol **58**, United States of America.

TUKEY, J. W. 1957, On the comparative anatomy of transformations, Annuals of mathematical Statistics, 28, 602-603

WHITE, D.J. & CLOWES, R.M., 1990. Shallow crustal structure beneath the Juan de Fuca Ridge from 2-D seismic refraction tomography, *Geophys. J. Int.,* **100**, 349–367.

ZHANG, J., TEN Brink, U.S. & TOKSOZ, M.N., 1998. Nonlinear refraction and reflection traveltime tomography, *Journal of Geophysical Research.,* **103**, 29 743–29 75

ZELT A. Colin., Modelling strategies and model assessment for wide-angle seismic traveltime data, *Geophysics*, Vol **139**; P.183-204. Society of Exploration Geophysicists 1999.

ZELT A. Colin., Lateral velocity resolution for three-dimensional seismic refraction

data, *Geophys, J. Int.,* **135**, Pages 1101-1112

ZHU T., CHEADLE S., PETRELLA A., GRAY S.., 2001, First-Arrival Tomography for Near-Surface Model Building ; Veritas DGC Inc., Suite 2200 - 5 Avenue, SW, Calgary, Alberta, Canada, T2P 5A2., EAGE 63rd Conference & Technical Exhibition - Amsterdam, The Netherlands, 11 - 15 June 2001.

ANEXOS

ANEXO 1

Código para generar set de datos sintético

NOMBRE DEL FICHERO: (masHw.cpp)

```
// masHw.cpp-programa que utiliza la clase geometria para Incluir headers a el set de datos
#include<iostream>
#include "geometria.h"

using namespace std;

int main()
{
        Geometria set1(157, 15, 60);
        set1.generarHw();
        set1.ficheroPromax();
}
```

NOMBRE DEL FICHERO: (geometria.h)

```
//definicion de la clase Geometria
#if !defined( _GEOMETRIA_H_ )
#define _GEOMETRIA_H_

class   Geometria
{
        //Atributos-miembros publicos
        private:
                int Nshot;
                int geoX;
                int souX;
        public:
        int s; //Numero de Estacion de la fuente
        int ro;
        int egeo;
        float gx, gy;
```

```
        int rf;
        int ntr;
        float offset; //offset
        float maxOffset; //Maximo offset
        Geometria() {}
        Geometria( int Ndisparos, int Destaciones, int Dfuentes);
        int generarHw(); //Generar Informacion del Header offset
        int ficheroPromax(); //Generar fichero ASCCI para exportar a Promax
};
#endif // _GEOMETRIA_H
```

NOMBRE DEL FICHERO: (geometria.cpp)

```
//geometria.cpp
#include<iostream>
#include "geometria.h"
#include "segy.h"

using namespace std;

Geometria::Geometria(int Ndisparos, int Destaciones, int Dfuentes)
{
        Nshot= Ndisparos; geoX= Destaciones; souX= Dfuentes;
}

int Geometria::generarHw()
{
        FILE *out;
        FILE *out1;
        char namefile[15];
        char namefile1[15];
        for(int i=1; i<Nshot; i++)
        {
           sprintf(namefile,"offsetHw%d",i);
           sprintf(namefile1,"gxHw%d",i);
           out= fopen(namefile,"wb");
           out1= fopen(namefile1,"wb");
                int ro= i * 4;
                int rf= ro + 800;
                int s= ro + 400;

                if (s >= 371 && s <= 1038)
                {
                        if (ro < 371)
                        {
                                ro=371;
                                        if(rf > 1038)
                                        {
```

```
                                                    rf=1038;
                                                }
                                        }
                                }
                                ntr= rf-ro;
                                for(int j=0; j<ntr; j++)
                                {
                                        egeo= ro + j;
                                        offset= (egeo - s) * geoX;
                                                if(maxOffset < offset)
                                                {
                                                        maxOffset=offset;
                                                }
                                        gx= egeo * geoX;
                                        fwrite(&offset, sizeof(float),1,out);
                                        fwrite(&gx, sizeof(float),1,out1);
                                }
        fclose (out);
        fclose (out1);
        }
        printf("%f\n", maxOffset);//PrintMaxOffset
        return (0);
}

int Geometria::ficheroPromax()
{
        FILE *out2;
        out2= fopen("zroPromax.txt","w");
        FILE *out3;
        out3= fopen("zOtros.txt","w");
        for(int i=1; i<Nshot; i++)
        {
                int ro= i * 4;
                int rf= ro + 800;
                int s= ro + 400;

                if (s >= 371 && s <= 1038)
                {
                        if (ro < 371)
                        {
                                ro=371;
                        }
                                        if (rf > 1038)
                                        {
                                                rf=1038;
                                        }
                }
                ntr= rf-ro;
        fprintf(out2, "%i\n", ro);
```

```
        fprintf(out3, "%i %i %i %i %i\n", i, ro, s, rf, ntr);
        }
        fclose (out2);
        fclose (out3);
        return (0);
}
```

NOMBRE DEL FICHERO: (Xcshot)

```
#! /bin/sh
# File: Xcshot
# Generación de Datos Sintético usando --> cshot
# Mayo de 2007

# Tamaño del modelo
x1beg=0 x1end=4000 x2beg=0 x2end=16000
# Velocidades entre interfaces
vel1=1500.0 vel2=3000.0 vel3=4000.0 vel4=5000.0 vel5=5200.0
# Numero de disparos
nums=157

echo " ***-Empezando proceso-***"

./masHw.exe

        i=1
        while [ "$i" -ne "$nums" ]
        do

                ro=`expr $i \* 4`
                g1=`expr $ro + 100`
                g2=`expr $g1 + 1`
                rf=`expr $ro + 800`
                s=`expr $ro + 400`

        if [ $s -ge 371 ]; then
                if [ $ro -le 371 ]; then
                ro=371
                fi
                        if [ $rf -ge 1038 ]; then
                        rf=1038
                        fi

                deltag=`expr $rf - $ro`
                sx=`expr $s \* 15`

                echo "0    0.\n15.     0.\n$ro $g1 $g2 $rf $s. 0." > geometry0
```

```
                echo "multi1\n4\nplotcolors\nm\ndummywell\ns\ngeometry0\nsg\nrt\nfichero$i\n-90.
90.\n1.\n$vel1 $vel2 $vel3\n$vel4 $vel5\nn\n1 2 3 4\ny" > param1

                echo "s\n1 1\n1 $deltag\n10. 25. 35. 50.\n.150\n.004\n4.\nfichero${i}shot\nfichero${i}traces" >
param2

                cshot1 |
                cshotplot >fichero${i}plot  outpar=fichero${i}par

                xgraph <fichero${i}plot par=fichero${i}par style=seismic \
                -geometry 600x300+650+650 \
                title="Sintetico - disparo$i" \
                label1="Profundidad (metros)"  label2="Abscisa (metros)" \
                x1beg=$x1beg x1end=$x1end x2beg=$x2beg x2end=$x2end &

                cshot2

                suaddhead <fichero${i}traces ftn=1 ns=1001 |
                sushw key=dt a=4000 | sushw key=tracl a=1 b=1 c=0 j=$deltag | sushw key=ep a=$s b=0 c=0
| sushw key=sx a=$sx b=0 c=0 | sushw key=sy a=10000.0 b=0 c=0 | sushw key=gy a=10000.0 b=0 c=0 | sushw
key=gelev a=1000.0 b=0 c=0 | sushw key=selev a=1000.0 b=0 c=0 | sushw key=offset infile=offsetHw$i | sushw
key=gx infile=gxHw$i | sushw key=ntr a=$deltag b=0 c=0 | sushw key=fldr a=$i b=0 c=0 | sugain gagc=1
wagc=0.5 >sudata${i}trazas.su


                #suximage title="Headwaves & Primarias disparo$i" \
                #ybox=70 \
                #label1="Time in Seconds"  label2="Trace" <sudata${i}trazas.su

                echo " **Disparo $i Listo!"
        fi
        i=`expr $i + 1`
        done

echo " ***Emsamblando Disparos***"
j=1
echo "cat \c" > temp0
echo " > dataseis.su" > fin
while [ "$j" -ne "$nums" ]
do
        echo "sudata${j}trazas.su \c" > temp
        cat temp0 temp > aux
        cp aux temp0
        j=`expr $j + 1`
done

cat temp0 fin > cat.sh
chmod 777 cat.sh
./cat.sh
```

```
echo " ***Archivos temporales eliminados***"
#rm -f sudata*
rm -f temp*
rm -f fichero*
rm -f aux fin
rm -f offsetHw*

echo " ***Generando archivo seg-y***"
segyhdrs <dataseis.su
segywrite <dataseis.su tape=dataseis.sgy endian=0 hfile=header bfile=binary
#rm -f dataseis.su
echo " -***-Proceso terminado-***-"
```

Anexo 2

Código para realizar análisis Jackknifing

NOMBRE DEL FICHERO: (analisisTest.cpp)

```
// analisisTest.cpp-programa que utiliza la clase analisis
#include<iostream>
#include "analisis.h"

using namespace std;

int main()
{
        analisis a(50); //NUmero de modelos q va a generar
        a.leerEntrada();
        a.genInputGrilla(20, 300, 5, 4000.00, 1000.00);//size cell en x, profundidad del modelo, size cell(en z),
Velocidad maxima, Velocidad minima.
        a.jackknifing(10, 300);//Numero de filas a eliminar de cada disparo , profundidad
}
```

NOMBRE DEL FICHERO: (analisis.h)

```
//definicion de la clase Analisis
#if !defined( _ANALISIS_H_ )
#define _ANALISIS_H_

class analisis
```

```
{
        //atributos
        private:
                int Numol; //NUmero de Modelos
    protected:
                void mensage1();
        //miembros publicos
        public:
        analisis() {}
        analisis(int numol);
        double leerEntrada();
        double genInputGrilla(int dx, int profundidad, int dz, float Vmax, float Vmin);
        double genttfile();
        double jackknifing(int shift, int); //shift: Numero de trazas que se desean sacar de cada disparo
        int shot;
        int cont[800]; //Contador de: Numero de receptores por disparo
        float coorXmin;
        float coorXref;
        float coorYref;
        float Cotamax, Cotamin, Cotadiff;
};
#endif // _ANALISIS_H_
```

NOMBRE DEL FICHERO: (analisis.cpp)

```
//analisis.cpp
//Write by: CarlosBecerra
//Date: Sep/Nov/2007
#include<iostream>
#include "analisis.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        char id[5];
        int estacion;
        int cooX;
        int cooY;
        float cota;
        float tpa; //Tiempo de Primer Arribo
        float tup; //Tiempo de uphole, solo para SHOT's
```

```
}lectura ;

void analisis::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }

analisis::analisis(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double analisis::leerEntrada() { //Lee un fichero secuencial
  FILE *fichero;
  lectura leer;
  int trace =0;
  shot =0; //Iniciando contador
  int sumpick =0; //Contador NUmero Total de picks
  coorXmin= 100000000000.00;

        if((fichero=fopen("clientes2.geo", "r"))==NULL)
        printf("\n\nCaution1: don't read file: Look input file name\n\n");
        else {

      while (!feof(fichero)) {

                fscanf(fichero,"%s",&leer.id);
                int resu = strncmp(leer.id, "SHOT", 4); //Compara 2 cadenas de caracteres
                fseek(fichero, -strlen(leer.id)*sizeof(char), SEEK_CUR); //devuelve el puntero al inicio del
fichero
                if (resu != 0) { //lee y imprime RECEIVER
                fscanf(fichero,"%s %d %d %d %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa);
                trace ++;

                        if(leer.cooX < coorXmin) { //Encontrar coordenadas de punto de referencia, En los
shot's
                        coorXmin = leer.cooX;
                        coorXref = coorXmin;
                        coorYref = leer.cooY;
                        }
                }
                if (resu == 0) { //lee y imprime SHOT
                fscanf(fichero,"%s %d %d %d %f %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa, &leer.tup);
                cont[shot] = trace;
                shot ++;
```

```
                    trace =0;

                                if(leer.cooX < coorXmin) { //Encontrar coordenadas de punto de referencia, En los
receptores's
                                coorXmin = leer.cooX;
                                coorXref = coorXmin;
                                coorYref = leer.cooY;
                                }
                    }
          }
          cont[shot]=trace;
              }
          fclose(fichero);

                    cout << "\n\n********************** info 1 **********************" << endl;

                    cout << "NUmero de receptores por disparo: " << endl;
                    for(int i=1; i<=shot; i++) { //Imprime Informacion leida del fichero
                    printf("%d ", cont[i]); //Imprime Numero de trazas por disparo.
                    sumpick = sumpick + cont[i];
                    }
                    cout << "\n\n********************** info 2 **********************" << endl;
                    cout << "El set de Datos leido contiene " << shot << " disparos, " << sumpick << " picks" <<
endl;
                    cout << "Las coordenadas (x,y) del punto de referencia son: ";
                    printf( "%10.2f %10.2f\n\n", coorXref ,coorYref);
}

double analisis::genInputGrilla(int dx, int profundidad, int dz, float Vmax, float Vmin) {//Genera modelo
inicial(Grilla); file de longitud de correlaccion
  FILE *fichero;
  lectura leer;
  float COORx, COORxoff, COORshot, tempx, tempz, tempO, vel, tpatmp, offset;
  int Vwater=1500;
  int Vair=333;
  float Xmax;
  int nodox, nodoz;
  float correlengH, correlengV, correlengAirH, correlengAirV;
  vector<float> coorx;//Defino vector, Coordenadas X: No Ordenadas, Repetidas
  vector<float> coorz;//Defino vector, Coordenadas Y: No Ordenadas, Repetidas
  vector<float> filex;//Coordenadas X: Ordenadas, No Repetidas
  vector<float> filetopo;//Coordenadas Y Topograficas: Ordenadas, No Repetidas
  vector<float> filez;//Coordenadas Y: Ordenadas, No Repetidas
  vector<float> OrderZ;
  vector<int> filenodox;//Contiene las Coordenadas X de los nodos de la Grilla
  vector<float> fileZgrilla;//Contiene las Cotas de los nodos de la Grilla
  vector<int> fileZ;//Contiene las Coordenadas de los nodos de la Grilla
  vector<float> offFile1;//Contiene offset absolutos
  vector<float> offFile2;//Contiene offset absolutos
```

```
vector<float> offFile3;//Contiene offset absolutos
vector<float> offFile4;//Contiene offset absolutos
vector<float> offFile5;//Contiene offset absolutos
vector<float> offtpaFile1;//Contiene tiempos de primeros arribos
vector<float> offtpaFile2;//Contiene tiempos de primeros arribos
vector<float> offtpaFile3;//Contiene tiempos de primeros arribos
vector<float> offtpaFile4;//Contiene tiempos de primeros arribos
vector<float> offtpaFile5;//Contiene tiempos de primeros arribos
vector<int> RecEst;//Contiene Estacion de Receptores
vector<int> SouEst;//Contiene Estacion de Fuentes
vector<float> RecCoordX;//Contiene Coordenada X de Receptores
vector<float> SouCoordX;//Contiene Coordenada X de Fuentes
//vector<float> numAleR;//Contiene un conjunto de numeros Aleatorios
//vector<float> filetopon;//Coordenadas Y Topograficas: No Ordenadas, No Repetidas

        if((fichero=fopen("clientes2.geo", "r"))==NULL)
        printf("\n\ncaution1: don't read file, Revisar el nombre del fichero de entrada!\n\n");
        else {
          while (!feof(fichero)) {

                  for(int i=1; i<=shot; i++) { //lee e imprime informacion de 1 disparo
                 fscanf(fichero,"%s %d %d %d %f %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa, &leer.tup);
                  SouEst.push_back(leer.estacion);
              COORx = sqrt(pow(coorXref-(leer.cooX),2)+pow(coorYref-(leer.cooY),2));
                  SouCoordX.push_back(COORx);
                  COORshot = COORx;
                  coorx.push_back(COORx);//adiciona este elemento al final del vector
                  coorz.push_back(leer.cota);

                   for(int j=1; j<=cont[i]; j++) { //lee e imprime informacion de los receptores
                    fscanf(fichero,"%s %d %d %d %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa);
                   RecEst.push_back(leer.estacion);
                COORxoff = sqrt(pow(coorXref-(leer.cooX),2)+pow(coorYref-(leer.cooY),2));
                   RecCoordX.push_back(COORxoff);
                   offset = abs(COORshot - COORxoff);//Calculo de offset
                   tpatmp = leer.tpa / 1000;//Calculo de tiempo de primer arribo

                    if (i<45) {//Encontrando variacion lateral de velocidad...
                    offFile1.push_back(offset);
                    offtpaFile1.push_back(tpatmp); }
                    else {
                            if (i<60) {
                      offFile2.push_back(offset);
                      offtpaFile2.push_back(tpatmp); }
                            else {
                                    if (i<75) {
                            offFile3.push_back(offset);
```

```
                                offftpaFile3.push_back(tpatmp); }
                                        else {
                                                if (i<90) {
                                        offFile4.push_back(offset);
                                        offtpaFile4.push_back(tpatmp); }
                                                else {
                                                offFile5.push_back(offset);
                                        offftpaFile5.push_back(tpatmp);
                                                }
                                        }
                                }
                        }

                COORx = sqrt(pow(coorXref-(leer.cooX),2)+pow(coorYref-(leer.cooY),2));
                        coorx.push_back(COORx);//adiciona este elemento al final del vector
                        coorz.push_back(leer.cota);
                }
                    }
              }
            }
  fclose(fichero);

//pruebA de informacion de recpetores y fuentes...
FILE *outtSOU;
outtSOU= fopen("OUTSOU.dat","w");
int hola1SOU = SouEst.size();
for (int k=0; k<hola1SOU; k++) {
fprintf(outtSOU, "%d %-10.2f\n", SouEst[k], SouCoordX[k]); }
fclose(outtSOU);
//...
FILE *outtSOU1;
outtSOU1= fopen("OUTREC.dat","w");
int hola1SOU1 = RecEst.size();
for (int k=0; k<hola1SOU1; k++) {
fprintf(outtSOU1, "%d %-10.2f\n", RecEst[k], RecCoordX[k]); }
fclose(outtSOU1);
//fin... pruebA de informacion de recpetores y fuentes...


            //Generar tiempo-offset --> fichero que contiene tiempo vs. offset
            //Imprime parte 1
            FILE *outtvsoff1;
            outtvsoff1= fopen("timeoffset1.dat","w");
            int hola1 = offFile1.size();
            for (int k=0; k<hola1; k++) {
            fprintf(outtvsoff1, "%10.2f %10.5f\n", offFile1[k], offftpaFile1[k]); }
            fclose(outtvsoff1);
            //Imprime parte 2
            FILE *outtvsoff2;
```

```
outtvsoff2= fopen("timeoffset2.dat","w");
int hola2 = offFile2.size();
for (int k=0; k<hola2; k++) {
fprintf(outtvsoff2, "%10.2f %10.5f\n", offFile2[k], offtpaFile2[k]); }
fclose(outtvsoff2);
//Imprime parte 3
FILE *outtvsoff3;
outtvsoff3= fopen("timeoffset3.dat","w");
int hola3 = offFile3.size();
for (int k=0; k<hola3; k++) {
fprintf(outtvsoff3, "%10.2f %10.5f\n", offFile3[k], offtpaFile3[k]); }
fclose(outtvsoff3);
//Imprime parte 4
FILE *outtvsoff4;
outtvsoff4= fopen("timeoffset4.dat","w");
int hola4 = offFile4.size();
for (int k=0; k<hola4; k++) {
fprintf(outtvsoff4, "%10.2f %10.5f\n", offFile4[k], offtpaFile4[k]); }
fclose(outtvsoff4);
//Imprime parte 5
FILE *outtvsoff5;
outtvsoff5= fopen("timeoffset5.dat","w");
int hola5 = offFile5.size();
for (int k=0; k<hola5; k++) {
fprintf(outtvsoff5, "%10.2f %10.5f\n", offFile5[k], offtpaFile5[k]); }
fclose(outtvsoff5);


int NumElem = coorx.size();//tamaño del vector
for(int i=0; i<(NumElem-1); i++) {
  for(int j=0; j<(NumElem-1); j++) {///Ordenar vector de Coordenadas X con correspondiente Cota
    if(coorx[j] > coorx[j+1]) {
    tempx=coorx[j];
    tempz=coorz[j];
    coorx[j]=coorx[j+1];
    coorz[j]=coorz[j+1];
    coorx[j+1]=tempx;
    coorz[j+1]=tempz;
    }
  }
}

//Vector de Cotas en orden Ascendente --> Para encontrar Cotas maxima & minima
OrderZ = coorz;//Copia de vectores
for(int i=0; i<(NumElem-1); i++) {
  for(int j=0; j<(NumElem-1); j++) {///Ordenar vector de Coordenadas X con correspondiente Cota
     if(OrderZ[j] > OrderZ[j+1]) {
     tempO=OrderZ[j];
     OrderZ[j]=OrderZ[j+1];
```

```
    OrderZ[j+1]=tempO;
   }
  }
 }
 Cotamax = OrderZ[NumElem-1];
 Cotamin = OrderZ[0];
 Cotadiff = Cotamax - Cotamin;

 cout << "\n********** info 3 **********" << endl;
 printf("La Cota maxima es: %8.2f y la minima es %8.2f la profundidad es %8d\n", Cotamax, Cotamin,
profundidad);
 cout << "**********  info  **********\n" << endl;

 //Crear Nuevos vectores que no contengan coordenadas_X repetidas
 int nwe=NumElem;
 for(int k=0; k<nwe; k++) {//No incluir Repetidos en vectores
   if(coorx[k] != coorx[k+1]) {
   filex.push_back(coorx[k]);
   filetopo.push_back(coorz[k]);
   }
 }

 //Cambio del plano de referencia de las Cotas
 int tam1 = filetopo.size();
 float filetopon[tam1];//Nuevo vector
 for(int i=0; i<tam1; i++) {
 filetopon[i] = filetopo[i] - Cotamin + profundidad;
 }

 //Generar archivo con informacion de coordenadas y Cotas
 FILE *outTOPO;
 outTOPO= fopen("myTOPOGRAFIA.dat","w");
 for(int i=0; i<tam1; i++) {
 fprintf(outTOPO, "%8.2f %8.2f %8.2f\n", filex[i], filetopo[i], filetopon[i]); }

 //Generar correfile --> fichero de longitud de Correlacion
 FILE *outCorrel;
 outCorrel= fopen("correfile.dat","w");
 //Generar Xfile, Zfile, Tfile --> gen_smesh
 FILE *outxf;
 outxf= fopen("mymodelinicial.dat","w");
 int NumElemxf = filex.size();//tamaño del vector
 int NumElemzft = NumElemxf + profundidad/dz;
 Xmax = filex[NumElemxf-1];


 float nx = ((Xmax / dx) + 1);
 int nxint = nx;
 float nz = ((profundidad / dz) + 1);
```

```
float nzadd = ((Cotamax - Cotamin) / dz) + 1;
int nzaddint = nzadd;
int nzint = nz;
//nzint = nz + nzaddint;//Borrar

//:::(1):::INFORMACION DEL NUMERO DE NODOS
fprintf(outxf, "%d %d %d %d\n", nxint, nzint, Vwater, Vair);
fprintf(outCorrel, "%d %d\n", nxint, nzint);

FILE *outxJackk;//Fichero que sera usado para el calculo de la media en el analisis Jackknifing
outxJackk= fopen("differenAcm.dat","w");
int limite = nxint * nzint;
float vlini = 0.00;
for(int c=0; c<limite; c++) {
fprintf(outxJackk, "%f", vlini);
}
fclose(outxJackk);

//:::(2):::COORDENADAS X//
for(int k=0; k< nxint; k++) {//Imprime coordenadas X de los nodos de la Grilla
  if (k==(nxint-1)) {
  nodox = k * dx;
  filenodox.push_back(nodox);//adiciona este elemento al final del vector
  fprintf(outxf, "%d \n", nodox);
  fprintf(outCorrel, "%d \n", nodox);
  }
     else {
     nodox = k * dx;
     filenodox.push_back(nodox);//adiciona este elemento al final del vector
     fprintf(outxf, "%d ", nodox);
     fprintf(outCorrel, "%d ", nodox);
     }
}

//:::(3):::TOPOGRAFIA//
//Cota Topografica para el nodo x=0
fileZgrilla.push_back(filetopon[0]);//adiciona este elemento al final del vector
//Cota Topografica para el nodo 0<x<nx
int NumElemtmp = filenodox.size();//tamaño del vector
int NumElemtmpx = filex.size();//tamaño del vector
int number;
float elevtgrilla, diference;
for(int i=1; i<nxint; i++) { //Busca una Cota para cada nodo de la Grilla
    for(int j=0; j<NumElemtmpx; j++) { //Busca el Numero
        if(filenodox[i] <= filex[j]) {
              number = j;
                diference = ((filenodox[i]-filex[number-1]) * (filetopon[number]-filetopon[number-1]))/
(filex[number]-filex[number-1]);
              elevtgrilla = filetopon[number-1] + diference;
```

```
                fileZgrilla.push_back(elevtgrilla);//adiciona este elemento al final del vector
                j= NumElemtmpx;
                }
            }
        }
        for(int k=0; k<NumElemtmp; k++) {///Imprime Coordenadas Z Topografica de los nodos de la Grilla
          if (k==(NumElemtmp-1)) {
          fileZgrilla[k] = (-1) * fileZgrilla[k]; // Poner la Topografia Negativa
          fprintf(outxf, "%-10.2f\n", fileZgrilla[k]);
          fprintf(outCorrel, "%-10.2f\n", fileZgrilla[k]);
          }
            else {
            fileZgrilla[k] = (-1) * fileZgrilla[k]; // Poner la Topografia Negativa
            fprintf(outxf, "%-10.2f", fileZgrilla[k]);
            fprintf(outCorrel, "%-10.2f", fileZgrilla[k]);
            }
        }

        //:::(4):::COORDENADAS Z//
        int NumElemzf = filez.size();
        for(int k=0; k<nzint; k++) {///Imprime coordenadas Z de los nodos de la Grilla
           if (k==(nzint-1)) {
           nodoz = (k * dz);
           fileZ.push_back(nodoz);//adiciona este elemento al final del vector
           fprintf(outxf, "%d\n", nodoz);
           fprintf(outCorrel, "%d\n", nodoz);
           }
              else {
              nodoz = (k * dz);
              fileZ.push_back(nodoz);//adiciona este elemento al final del vector
              fprintf(outxf, "%d ", nodoz);
              fprintf(outCorrel, "%d ", nodoz);
              }
        }

        //:::(5):::Genera Grilla de Velocidades - Cambiando el patron de la grilla cada DETERMINDADA
PROFUNDIDAD
        NumElemzf = filez.size();
        int limnodx = filenodox.size();//tamaño del vector
        int limnodz = fileZ.size();//tamaño del vector
        for(int k=0; k<limnodx; k++) {///Una iteracion para cada nodo X de la grilla
           for(int i=0; i<limnodz; i++) {///Una iteracion para cada nodo Z de la grilla
                   if(i < (limnodz-1)) {
                     float z1 = 0.00;
                     float z2 = 22.00;
                     Vmin = 2000.00; Vmax= 2500.00;
                     if(fileZ[i]<z2) {
                     vel = Vmin + (((fileZ[i] - z1) * (Vmax-Vmin)) / (z2-z1)); //Paralelo a la Topografia
                     //vel = 1000.00; //Velocidad Constante en la Grilla
```

```
            fprintf(outxf, "%-10.2f", vel); }
                  else {
                    z1 = 22.00;
                    z2 = 70.50;//Profundidad maxima
                    Vmin = 2500.00; Vmax= 2667.00;
                    if(fileZ[i]<z2) {
                    vel = Vmin + (((fileZ[i] - z1) * (Vmax-Vmin)) / (z2-z1)); //Paralelo a la Topografia
                    //vel = 3900; //Velocidad Constante en la Grilla
                 fprintf(outxf, "%-10.2f", vel); }
                      else {
                        z1 = 70.50;
                        z2 = 400.00;//Profundidad maxima
                        Vmin = 2667.00; Vmax= 4400.00;
                        vel = Vmin + (((fileZ[i] - z1) * (Vmax-Vmin)) / (z2-z1)); //Paralelo a la Topografia
                   fprintf(outxf, "%-10.2f", vel);
                     }
                   }
                  }
                   else {
                     float z1 = 0.00;
                     float z2 = 22.00;//Profundidad maxima
                     if(fileZ[i]<z2) {
                     vel = Vmin + (((fileZ[i] - z1) * (Vmax-Vmin)) / (z2-z1)); //Paralelo a la Topografia
                     //vel = 1000.00; //Velocidad Constante en la Grilla
            fprintf(outxf, "%-10.2f\n", vel);
                      }
                      else {
                        z1 = 22.00;
                        z2 = 70.50;//Profundidad maxima
                        Vmin = 2500.00; Vmax= 2667.00;
                        if(fileZ[i]<z2) {
                        vel = Vmin + (((fileZ[i] - z1) * (Vmax-Vmin)) / (z2-z1)); //Paralelo a la Topografia
                        //vel = 3900; //Velocidad Constante en la Grilla
                 fprintf(outxf, "%-10.2f\n", vel);
                      }
                      else {
                        z1 = 70.50;
                        z2 = 400.00;//Profundidad maxima
                        Vmin = 2667.00; Vmax= 4400.00;
                        vel = Vmin + (((fileZ[i] - z1) * (Vmax-Vmin)) / (z2-z1)); //Paralelo a la Topografia
                   fprintf(outxf, "%-10.2f\n", vel);
                      }
                   }
          }
        }
      }
      fclose(outxf);
      fclose(outTOPO);
```

```
//Informacion
cout << "\n\n********************** info 4 **********************" << endl;
cout << "nx: " << nxint << " nz: " << nzint << " vels: " << Vwater << " ," << Vair << endl;
cout << "Tamaño de las Celdas de la Grilla: (" << dx << " x " << dz << ")" << endl;

//:::(6-a):::Genera Valores de Longitud de Correlacion Horizontal - Para los nodos de la Grilla
correlengAirH = 0.00;
for(int j=0; j<limnodx; j++) {//Una iteracion para cada nodo X de la grilla
   for(int i=0; i<limnodz; i++) {//Una iteracion para cada nodo Z de la grilla
     if(i == (limnodz-1)) {
          correlengH = (dx*10.00) + (((abs(fileZgrilla[j] - fileZ[i])) * (dx*20.00 -
dx*10.00))/(profundidad)); //Tamaño de la Correlacion Horizontal
      fprintf(outCorrel, "%-10.2f\n", correlengH); }
     else {
          //if (fileZ[i] >= fileZgrilla[j]) {
          correlengH = (dx*10.00) + (((abs(fileZgrilla[j] - fileZ[i])) * (dx*20.00 -
dx*10.00))/(profundidad)); //Tamaño de la Correlacion Horizontal
          fprintf(outCorrel, "%-10.2f", correlengH);
          //}
          //else { fprintf(outCorrel, "%-10.2f", correlengAirH); }
     }
    }
}

//:::(6-b):::Genera Valores de Longitud de Correlacion Vertical - Para los nodos de la Grilla
correlengAirV = 0.00;
for(int j=0; j<limnodx; j++) {//Una iteracion para cada nodo X de la grilla
   for(int i=0; i<limnodz; i++) {//Una iteracion para cada nodo Z de la grilla
     if(i == (limnodz-1)) {
          correlengV = dz + (((abs(fileZgrilla[j] - fileZ[i])) * (dz*40.00 - dz))/(profundidad)); //Tamaño
de la Correlacion Vertical
      fprintf(outCorrel, "%-10.2f\n", correlengV); }
     else {
          //if (fileZ[i] >= fileZgrilla[j]) {
          correlengV = dz + (((abs(fileZgrilla[j] - fileZ[i])) * (dz*40.00 - dz))/(profundidad)); //Tamaño
de la Correlacion Vertical
          fprintf(outCorrel, "%-10.2f", correlengV);
          //}
          //else { fprintf(outCorrel, "%-10.2f", correlengAirV); }
     }
    }
}
fclose(outCorrel);


//:::(7):::Genera fichero con el rango de interes del modelo
FILE *range;
range= fopen("outPoligono.dat","w");
int Nrange = filenodox.size();
```

```
            int nrange = Nrange -1;
            for(int f=0; f<Nrange; f++) {///Capa de Topografia
            fprintf(range, "%d %-10.2f\n", filenodox[f], fileZgrilla[f]);
            }
            int XminRange = filenodox[0];
            //int XmaxRange = filenodox[nrange];
            float ZfirstRange = fileZgrilla[0];
            //float ZlastRange = fileZgrilla[nrange];
            //fprintf(range, "%d %-10.2f\n", XmaxRange, ZlastRange);
            float Ztmprange;
            for(int f=nrange; f>-1; f--) {///Capa de interes
            Ztmprange = fileZgrilla[f] + 300.00;//Espesor de la capa de Interes
            fprintf(range, "%d %-10.2f\n", filenodox[f], Ztmprange);
            }
            fprintf(range, "%d %-10.2f\n", XminRange, ZfirstRange);
            fclose(range);

}


double analisis::jackknifing(int shift, int profundidad) { //Lee y escribe un fichero secuencial
  FILE *fichero;
  FILE *out1;
  char namefile[15];
  lectura leer;
  int traceNumero, ntime;
  float COOR;
  char IDshot[] = "s";
  char IDtrace[] = "r";
  int code = 0; //INDICA: 0/refraccion; 1/refleccion
  float dt = 0.01;

            cout << "\n\n********************** info 5 **********************" << endl;
            cout << "***************** Analisis Jackknifing *****************\n" << endl;

 //:::(1):::GENERACION DE input para INVERTIR el set de datos Completo//

            FILE *outComple;
            outComple= fopen("outCompleto.dat","w");

            if((fichero=fopen("clientes2.geo", "r"))==NULL)
            printf("\n\ncaution1 don't read file\n\n");
            else {
              while (!feof(fichero)) {
              fprintf(outComple, "%d\n", shot);
                 for(int i=1; i<=shot; i++) { //lee e imprime informacion de 1 disparo
                      fscanf(fichero,"%s %d %d %d %f %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa, &leer.tup);
                 COOR = sqrt(pow(coorXref-(leer.cooX),2)+pow(coorYref-(leer.cooY),2));
```

```
                int nUmRec = cont[i];
                leer.cota = (leer.cota - Cotamin + profundidad - 5.00 ) * (-1);  //La fuente se entierra 5 metros.
                fprintf(outComple, "%s %10.2f %10.2f %10d\n", IDshot, COOR, leer.cota, nUmRec);
                traceNumero = cont[i];

                    //int jexe1 = traceNumero + 1; //Numero de veces q debe ejecutarse el bucle j
            for(int j=1; j<=nUmRec; j++) { //lee e imprime informacion de los receptores
                     fscanf(fichero,"%s %d %d %d %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa);
                  COOR = sqrt(pow(coorXref-(leer.cooX),2)+pow(coorYref-(leer.cooY),2));
                     leer.cota = (leer.cota - Cotamin + profundidad) * (-1);
                     leer.tpa = leer.tpa / 1000;//Cambiando Unidades de milisegundos a segundos
                   fprintf(outComple, "%s %10.2f %10.2f %5d %12.5f %10.2f\n", IDtrace, COOR, leer.cota, code,
leer.tpa, dt);
                  }
                 }
               }
             }
            fclose(fichero);
            fclose (outComple);

 //:::(2):::GENERACION DE n INVERTIBLES JACKKNIFING//
 long tm;
 time_t segundos;
 time(&segundos);
 printf("\n%s\n", ctime(&segundos)); //Informacion de tiempo en ejecucion
 srand((unsigned)time(0)); //cambio semilla

  for(int n=0; n<=Numol; n++) { //Cada Iteracion Genera 1 fichero que contine tiempos de primeros arribos a
invertir
    ntime = n + 1;
    time(&segundos);
    printf("\n%s", ctime(&segundos)); //Informacion de tiempo en ejecucion
    do //Esperar 1 segundo para generar otro NUmero Aleatorio
    tm = clock(); //Esperar 1 segundo para generar otro NUmero Aleatorio
    while (tm/CLOCKS_PER_SEC < ntime); //Esperar 1 segundo para generar otro NUmero Aleatorio

            sprintf(namefile, "out%d.dat", n); //Ajustando pararametros para generar ficheros de Salida
            out1= fopen(namefile,"w");

            if((fichero=fopen("clientes2.geo", "r"))==NULL)
            printf("\n\ncaution1 don't read file\n\n");
            else {
              while (!feof(fichero)) {
              fprintf(out1, "%d\n", shot);
                 for(int i=1; i<=shot; i++) { //lee e imprime informacion de 1 disparo
                     fscanf(fichero,"%s %d %d %d %f %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa, &leer.tup);
                 COOR = sqrt(pow(coorXref-(leer.cooX),2)+pow(coorYref-(leer.cooY),2));
```

```
int nUmRec = cont[i] - shift; // NUmero de receptores por cada disparo, restando el numero de picks especificado
leer.cota = (leer.cota - Cotamin + profundidad - 5.00 ) * (-1);  //La fuente se entierra 50 metros.....
fprintf(out1, "%s %10.2f %10.2f %10d\n", IDshot, COOR, leer.cota, nUmRec);
traceNumero = cont[i];

/*Generacion de un Conjunto Numeros de Aleatorios*/
int longbytes = 67;
int nmaximo= traceNumero - 1; //lo q cambia
vector<int> numAleR;//Contiene un conjunto de numeros Aleatorios
int mynum; //Variable del Numero aleatorio usado para eliminar un dato
int shiftR = shift + (shift/2);//Generar mas Numeros Aleatorios para reemplazar Aleatorios Repetidos
for(int r=0; r<shiftR; r++) {
mynum = rand()%nmaximo + 1;
numAleR.push_back(mynum);//adiciona este elemento al final del vector
//cout << mynum << " "; //Imprimir NUmero Aleatorio
}

//Ordenar el vector que contiene el conjunto de Numeros Aleatorios Repetidos
int NoElemAle = numAleR.size();//tamaño del vector
int tempAl;
for(int s=0; s<(NoElemAle-1); s++) {
 for(int t=0; t<(NoElemAle-1); t++) {
  if(numAleR[t] > numAleR[t+1]) {
  tempAl = numAleR[t];
  numAleR[t]=numAleR[t+1];
  numAleR[t+1]=tempAl;
  }
 }
}

//No incluir Repetidos
int nwe = shift;
int NoElemAleS;
vector<int> numAle;//Contiene un conjunto de numeros Aleatorios sin Repetidos
for(int k=0; k<NoElemAle; k++) {//No incluir Repetidos el vector
  if(numAleR[k] != numAleR[k+1]) {
  numAle.push_back(numAleR[k]);
  NoElemAleS = numAle.size();//tamaño del vector
        if(NoElemAleS == nwe) {//Condicion para finalizar iteraciones
        k= NoElemAle; }//El numero de Aleatorios deseado ha sido alcanzado
  }
}

//Imprimir Vector de Numeros Aleatorios
cout << "fichero " << n << " shot " << i << endl;
for(int d=0; d<NoElemAleS; d++) {
```

```
                    if(d == (NoElemAleS-1)) {
                printf("% d\n", numAle[d]); }//Imprimir NUmero Aleatorio
                    else {
                printf("% d", numAle[d]); }//Imprimir NUmero Aleatorio
                  }

                    int intoAl = 0;
                    mynum = numAle[intoAl];
                    int jexe = cont[i]; //Numero de veces q debe ejecutarse el bucle j
          for(int j=1; j<=jexe; j++) { //lee e imprime informacion de los receptores
                    if (mynum == j) {
                    fseek(fichero, longbytes, SEEK_CUR); //No bytes en cada fila
                            intoAl = intoAl + 1;
                       mynum = numAle[intoAl];
                    }
                    else {
                     fscanf(fichero,"%s %d %d %d %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa);
                COOR = sqrt(pow(coorXref-(leer.cooX),2)+pow(coorYref-(leer.cooY),2));
                    leer.cota = (leer.cota - Cotamin + profundidad) * (-1);
                    leer.tpa = leer.tpa / 1000;//Cambiando Unidades de milisegundos a segundos
                 fprintf(out1, "%s %10.2f %10.2f %5d %12.5f %10.2f\n", IDtrace, COOR, leer.cota, code, leer.tpa,
dt);
                                }
                }
               }
             }
          }
          fclose(fichero);
          fclose (out1);
 }
          cout << "\n\n****************** Fin Jackknifing ******************\n" << endl;
}
```

NOMBRE DEL FICHERO: (jacknifingTest.cpp)

```
// jacknifing.cpp-programa usado para realizar analisis de incertidumbre de un modelo de velocidad
/* Lee un fichero que contiene un modelo de velocidad solucion; se almacena en un vector y se computan las
diferencias entre la solucion del set Completo y las soluciones de los subset  */
#include<iostream>
#include "jacknifing.h"

using namespace std;


int main()
{
          jacknifing a(5); //NUmero de modelos q va a generar
```

```
        a.leerCompleto();
}
```

NOMBRE DEL FICHERO: (jacknifing.h)

```
//definicion de la clase jacknifing
#if !defined( _JACKNIFING_H_ )
#define _JACKNIFING_H_

class jacknifing
{
        //atributos
        private:
                int Numol; //NUmero de Modelos
    protected:
                void mensage1();
        //miembros publicos
        public:
        jacknifing() {}
        jacknifing(int numol);
        double leerCompleto();
};
#endif // _JACKNIFING_H_
```

NOMBRE DEL FICHERO: (jacknifing.cpp)

```
//jacknifing.cpp
//Write by: CarlosBecerra
//Date: Dec/2007
#include<iostream>
#include "jacknifing.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        int nx, nz, Vwater, Vair; //No. de nodos de la grilla y velocidades en el agua y en al aire
        int cxg, czg; //Coordenadas X y Z de la Grilla
        int coox;
        float ccxg; //Cota de la coordenada X de la grilla
        float velg;
        float cooz, veltmp;
```

```
}lectura ;

void jacknifing::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }

jacknifing::jacknifing(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double jacknifing::leerCompleto() { //Lee el fichero que contiene la solucion del set completo
  FILE *fichero;
  lectura leer;
  vector<int> Vcxg;//Vector que contiene Coordenadas X de la Grilla
  vector<int> Vcxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vczg;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> Vczg1;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> MvelCom;//Vector que contiene Coordenadas X de la Grilla
  int cxgi;
  float ccxgi;
  int czgi;
  float veli;

//:::::(1-l):::::Lee el fichero que contiene el modelo solucion del set de Datos Completo
        if((fichero=fopen("invtomoCom.dat", "r"))==NULL)
        printf("\n\nCaution1: don't read file: Look input file name\n\n");
        else {
                while (!feof(fichero)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
```

```
                    for(int z=0; z<leer.nz; z++) {
                      fscanf(fichero,"%d", &leer.czg);//Lee coordenadas X de la grilla
                      czgi = leer.czg;
                      Vczg.push_back(czgi);//adiciona este elemento al final del vector
                    }
                  //:::(5):::Lee informacion de la coordenadas x de la grilla
                    for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                            for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                            fscanf(fichero,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                       veli = leer.velg;
                       MvelCom.push_back(veli);//adiciona este elemento al final del vecto
                            }
                    }


                  }//fin_while
          }//fin_else
          fclose(fichero);

//:::::(1-p):::::Imprime un fichero que contiene el modelo solucion del set de Datos Completo
          FILE *outtomo;
          outtomo= fopen("mytomofinal.dat","w");
          //:::(1):::Imprime informacion de la grilla.
          fprintf(outtomo, "%d %d %d %d\n", leer.nx, leer.nz, leer.Vwater, leer.Vair);
          //:::(2):::Imprime informacion de la coordenadas x de la grilla
          for(int x=0; x<leer.nx; x++) {//Imprime coordenadas X de la grilla
            if (x < (leer.nx-1)) { //Ultimo elemto del arreglo
            fprintf(outtomo, "%d ", Vcxg[x]); }
            else { //Demas elementos del arreglo
            fprintf(outtomo, "%d\n", Vcxg[x]); }
          }
          //:::(3):::Imprime informacion de la Cota para cada coordenada x de la grilla
          for(int x=0; x<leer.nx; x++) {//Imprime cota para cada coordenadas X de la grilla
            if (x < (leer.nx-1)) { //Ultimo elemto del arreglo
            fprintf(outtomo, "%-10.2f ", Vccxg[x]); }
            else { //Demas elementos del arreglo
            fprintf(outtomo, "%-10.2f\n", Vccxg[x]); }
          }
          //:::(4):::Imprime informacion de la coordenadas x de la grilla
          for(int z=0; z<leer.nz; z++) {//Imprime coordenadas X de la grilla
            if (z < (leer.nz-1)) { //Ultimo elemto del arreglo
            fprintf(outtomo, "%d ", Vcxg[z]); }
            else { //Demas elementos del arreglo
            fprintf(outtomo, "%d\n", Vcxg[z]); }
          }
          fclose(outtomo);


//:::::(2-l):::::Lee el fichero que contiene el modelo solucion de 1 subset
```

```
//double jacknifing::leerSubset() { //Lee el fichero que contiene la solucion de un subset
FILE *fichero1;
vector<float> Mvel;//Vector que contiene Coordenadas X de la Grilla

        if((fichero1=fopen("invtomo1.dat", "r"))==NULL)
        printf("\n\nCaution1: don't read file: Look input file name\n\n");
        else {
                while (!feof(fichero1)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero1,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero1,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                          for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                          fscanf(fichero1,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                     veli = leer.velg;
                     Mvel.push_back(veli);//adiciona este elemento al final del vecto
                          }
                  }

                }//fin_while
        }//fin_else
        fclose(fichero1);

//:::::(2-p):::::Imprime un fichero que contiene el modelo solucion de 1 subset
        FILE *outtomo1;
        outtomo1= fopen("mytomofinal1.dat","w");

        //:::(1):::Imprime informacion de la grilla.
        fprintf(outtomo1, "%d %d %d %d\n", leer.nx, leer.nz, leer.Vwater, leer.Vair);
        //:::(2):::Imprime informacion de la coordenadas x de la grilla
        for(int x=0; x<leer.nx; x++) {//Imprime coordenadas X de la grilla
```

```
  if (x < (leer.nx-1)) { //Ultimo elemto del arreglo
  fprintf(outtomo1, "%d ", Vcxg[x]); }
  else { //Demas elementos del arreglo
  fprintf(outtomo1, "%d\n", Vcxg[x]); }
 }
 //:::(3):::Imprime informacion de la Cota para cada coordenada x de la grilla
 for(int x=0; x<leer.nx; x++) {//Imprime cota para cada coordenadas X de la grilla
  if (x < (leer.nx-1)) { //Ultimo elemto del arreglo
  fprintf(outtomo1, "%-10.2f ", Vccxg[x]); }
  else { //Demas elementos del arreglo
  fprintf(outtomo1, "%-10.2f\n", Vccxg[x]); }
 }
 //:::(4):::Imprime informacion de la coordenadas x de la grilla
 for(int z=0; z<leer.nz; z++) {//Imprime coordenadas X de la grilla
  if (z < (leer.nz-1)) { //Ultimo elemto del arreglo
  fprintf(outtomo1, "%d ", Vcxg[z]); }
  else { //Demas elementos del arreglo
  fprintf(outtomo1, "%d\n", Vcxg[z]); }
 }
 fclose(outtomo1);


//:::::(2-r):::::Restar vectores, Calcula la diferencia entre la solucion del modelo completo y un subset
 int Nober = Mvel.size();
 vector<float> diffVel;//Vector que contiene ...
 float diff;
 for (int i=0; i<Nober; i++) {
 diff = abs(MvelCom[i] - Mvel[i]);
 diffVel.push_back(diff);//adiciona este elemento al final del vector
 }


//:::::(2-g.2):::::Imprime un fichero que contiene la diferencia entre la solucion del modelo completo y un subset
 FILE *fichero3;
 FILE *outtomo3;
 outtomo3= fopen("outJacknifing.dat","w");//Escribir
 if((fichero3=fopen("myvelini.dat", "r"))==NULL)//Leido
 printf("\n\nCaution1: don't read file: Look input file name\n\n");
 else {
        while (!feof(fichero3)) {
        //:::(2):::Lee informacion de la coordenadas x de la grilla
          float velAire = 0.00;
          for(int i=0; i<9180; i++) {
              fscanf(fichero3,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                  if (leer.veltmp == 333) {//Condicion para que quede con el mismo formato
                  fprintf(outtomo3, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                  }//end_if
                  else {
```

```
                        fprintf(outtomo3, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, diffVel[i]);
                        }
                    }
                  }//fin_while
            }//fin_else
            fclose(fichero3);
            fclose(outtomo3);

//:::::(2-g.3):::::Adiciona coordenadas a los vectores q contiene topografia, y genera un fichero requerido para Graficar solucion
            int tamVcxg = Vcxg.size();
            int pto = tamVcxg-1;
            int pto1 = Vcxg[pto];

            Vcxg.push_back(pto1);//adiciona extremo derecho inferior
            Vccxg.push_back(0.00);//adiciona extremo derecho inferior

            Vcxg.push_back(0);//adiciona extremo izquierdo inferior
            Vccxg.push_back(0.00);//adiciona extremo izquiredo inferior

            float pto2 = Vccxg[0];
            Vcxg.push_back(0);//adiciona extremo izquierdo superior
            Vccxg.push_back(pto2);//adiciona extremo izquiredo superior

            FILE *outtomo4;
            outtomo4= fopen("outPoligono.dat","w");
            int tamVnew = Vcxg.size();
            for (int i=0; i < tamVnew; i++) {
            fprintf(outtomo4, "%d %f\n", Vcxg[i], Vccxg[i]); }
            fclose(outtomo4);
}
```

NOMBRE DEL FICHERO: (jacknifingORI.cpp)

```
//jacknifingORI.cpp
//Write by: CarlosBecerra
//Date: Dec/2007
#include<iostream>
#include "jacknifing.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;
```

```
typedef struct {
        int nx, nz, Vwater, Vair; //No. de nodos de la grilla y velocidades en el agua y en al aire
        int cxg, czg; //Coordenadas X y Z de la Grilla
        int coox;
        float ccxg; //Cota de la coordenada X de la grilla
        float velg;
        float cooz, veltmp;
        float diffeacm;
}lectura ;

void jacknifing::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }

jacknifing::jacknifing(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double jacknifing::leerCompleto() { //Lee el fichero que contiene la solucion del set completo
  lectura leer;
  vector<int> Vcxg;//Vector que contiene Coordenadas X de la Grilla
  vector<int> Vcxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg;//Vector que contiene Cota de las Coordenadas X de la Grilla
  vector<float> Vccxg1;//Vector que contiene Cota de las Coordenadas X de la Grilla
  vector<int> Vczg;//Vector que contiene Coordenadas Z de la Grilla
  vector<int> Vczg1;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> MvelCom;//Vector que contiene Coordenadas X de la Grilla
  int cxgi;
  float ccxgi;
  int czgi;
  float veli;

//:::::(1-l):::::Lee el fichero que contiene el modelo solucion del set de Datos Completo
cout << "Inicio de lectura archivo Completo" << endl;
        FILE *fichero;
        if((fichero=fopen("invtomoCom.dat", "r"))==NULL)
        printf("\n\nCaution1: don't read file: Look input file name\n\n");
        else {
                //while (!feof(fichero)) {
                //:::(1):::Lee informacion de la grilla
                  fscanf(fichero,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero,"%d", &leer.cxg);//Lee coordenadas X de la grilla
```

```
                    cxgi = leer.cxg;
                    Vcxg.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada X de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas z de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero,"%d", &leer.czg);//Lee coordenadas Z de la grilla
                    czgi = leer.czg;
                    Vczg.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la velocidad
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                          for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                          fscanf(fichero,"%f", &leer.velg);//Lee coordenadas X de la grilla
                     veli = leer.velg;
                     MvelCom.push_back(veli);//adiciona este elemento al final del vecto
                          }
                  }
                //}//fin_while
        }//fin_else
        fclose(fichero);


//:::::(2-l):::::Lee el fichero que contiene el modelo solucion de 1 subset
  FILE *fichero1;
  vector<float> Mvel;//Vector que contiene Coordenadas X de la Grilla

        if((fichero1=fopen("inv_out.smesh.1.1", "r"))==NULL)
        printf("\n\nCaution2: don't read file: Look input file name\n\n");
        else {
                //while (!feof(fichero1)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero1,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
```

```
                    }
                //:::(4):::Lee informacion de la coordenadas z de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero1,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                          for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                          fscanf(fichero1,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                     veli = leer.velg;
                     Mvel.push_back(veli);//adiciona este elemento al final del vecto
                          }
                  }
                //}//fin_while
          }//fin_else
          fclose(fichero1);


//:::::(2-r):::::Calcula: a)la diferencia entre la solucion del modelo completo y un subset b) la suma de las diferencias
          int Nober = Mvel.size();
          vector<float> diffVel;//Vector que contiene la diferencia entre la solucion del modelo completo y un subset
          float diff;
          for (int i=0; i<Nober; i++) {
          diff = abs(MvelCom[i] - Mvel[i]);
          diffVel.push_back(diff);//adiciona este elemento al final del vector
          }
          //a.1)Generacion del fichero que contiene las diferencias entre el modelo Completo y 1 subset
          cout << "Reescribiendo... diferencias.dat" << endl;
          FILE *outtomo1;
          outtomo1= fopen("diferencias.dat","w");
          int mole = diffVel.size();
            for(int z=0; z<mole; z++) {//Una iteracion para cada nodo Z de la grilla
            fprintf(outtomo1,"%f", diffVel[z]);
            }
          fclose(outtomo1);

          //a.2)Generacion del fichero que contiene las diferencias entre el modelo Completo y 1 subset en formato de grilla
          FILE *outtomoDIFF;
          outtomoDIFF= fopen("tomoDIFF.dat","w");
          //:::(1):::Imprime informacion de la grilla.
          fprintf(outtomoDIFF, "%d %d %d %d\n", leer.nx, leer.nz, leer.Vwater, leer.Vair);
          //:::(2):::Imprime informacion de la coordenadas x de la grilla
          for(int x=0; x<leer.nx; x++) {//Imprime coordenadas X de la grilla
            if (x < (leer.nx-1)) { //Ultimo elemto del arreglo
```

```
  fprintf(outtomoDIFF, "%d ", Vcxg[x]); }
  else { //Demas elementos del arreglo
  fprintf(outtomoDIFF, "%d\n", Vcxg[x]); }
 }
 //:::(3):::Imprime informacion de la Cota para cada coordenada x de la grilla
 for(int x=0; x<leer.nx; x++) {//Imprime cota para cada coordenadas X de la grilla
  if (x < (leer.nx-1)) { //Ultimo elemto del arreglo
  fprintf(outtomoDIFF, "%f ", Vccxg[x]); }
  else { //Demas elementos del arreglo
  fprintf(outtomoDIFF, "%f\n", Vccxg[x]); }
 }
 //:::(4):::Imprime informacion de la coordenadas z de la grilla
 for(int z=0; z<leer.nz; z++) {//Imprime coordenadas Z de la grilla
  if (z < (leer.nz-1)) { //Ultimo elemto del arreglo
  fprintf(outtomoDIFF, "%d ", Vczg[z]); }
  else { //Demas elementos del arreglo
  fprintf(outtomoDIFF, "%d\n", Vczg[z]); }
 }
 //:::(5):::Imprime Diferencias entre el modelo Completo y 1 Subset
 int multi = leer.nz;
 for(int d=1; d<=mole; d++) {//Imprime diferencias Z de la grilla
  if (d == multi) { //Ultimo elemento del arreglo
  fprintf(outtomoDIFF, "%f\n", diffVel[d]);
  multi = multi + leer.nz; }
  else { //Demas elementos del arreglo
  fprintf(outtomoDIFF, "%f ", diffVel[d]); }
 }
 fclose(outtomoDIFF);


 //b)Lectura del fichero de diferencias Acumuladas (...Para calculo de la media)
 cout << "leyendo... differenAcm.dat" << endl;
vector<float> diffAcm;//Vector que contiene...
 FILE *readdiffAcm;
 readdiffAcm= fopen("differenAcm.dat","r");
  for (int i=0; i<Nober; i++) {
  fscanf(readdiffAcm,"%f", &leer.diffeacm);//Lee
  diffAcm.push_back(leer.diffeacm);//adiciona este elemento al final del vector
  }
 fclose(readdiffAcm);

 //b)Generacion del fichero que contiene las diferencias Acumuladas
 cout << "Reescribiendo... differenAcm.dat" << endl;
 FILE *outAcum;
 outAcum= fopen("differenAcm.dat","w");
 float dAcm;
  for(int z=0; z<mole; z++) {//Una iteracion para cada nodo Z de la grilla
  dAcm = diffAcm[z] + diffVel[z];
  fprintf(outAcum,"% f", dAcm);
```

```
            }
          fclose(outAcum);


//:::::(2-m):::::Calcula la media de las diferencias
           float tmpdiff = 0.00;
           for(int z=0; z<mole; z++) {///Una iteracion para cada nodo Z de la grilla
             tmpdiff = tmpdiff + diffVel[z];
           }
            float mediaDiff = tmpdiff/mole;
            printf("La media de la diferencias es: %-10.2f\n", mediaDiff);

//:::::(2-g.2):::::Imprime un fichero que contiene la diferencia entre la solucion del modelo completo y un subset
          FILE *fichero3;
          FILE *outtomo3;
          outtomo3= fopen("outJacknifing.dat","w");//Escribir
          if((fichero3=fopen("myvelini.dat", "r"))==NULL)//Leido
          printf("\n\nCaution1: don't read file: Look input file name\n\n");
          else {
                    while (!feof(fichero3)) {
                    //:::(2):::Lee informacion de la coordenadas x de la grilla
                      float velAire = 0.00;
                      int NoTo = leer.nx * leer.nz;
                      for(int i=0; i<NoTo; i++) {
                          fscanf(fichero3,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                                if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                                fprintf(outtomo3, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                                }//end_if
                                else {
                                fprintf(outtomo3, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, diffVel[i]);
                                }
                      }
                    }
          }
          fclose(fichero3);
          fclose(outtomo3);

}
```

NOMBRE DEL FICHERO: (jacknifingFIN.cpp)

```
//jacknifingFIN.cpp
//Write by: CarlosBecerra
//Date: Dec/2007
#include<iostream>
#include "jacknifing.h"
#include <cstdio>
```

```
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        int nx, nz, Vwater, Vair; //No. de nodos de la grilla y velocidades en el agua y en al aire
        int cxg, czg; //Coordenadas X y Z de la Grilla
        int coox;
        float ccxg; //Cota de la coordenada X de la grilla
        float promg;
        float cooz, veltmp;
        float diffeacm;
}lectura ;

void jacknifing::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }

jacknifing::jacknifing(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double jacknifing::leerCompleto() { //Lee el fichero que contiene la solucion del set completo
  FILE *fichero;
  lectura leer;
  vector<int> Vcxg;//Vector que contiene Coordenadas X de la Grilla
  vector<int> Vcxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vczg;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> Vczg1;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> PromDiFF;//Vector que contiene el promedio de las Diferencias
  vector<float> StdDiFF;//Vector que contiene la Desviacion Standar
  vector<float> SumPromDiFF;//Vector que contiene el promedio de las Diferencias / coordenada X
  int cxgi;
  float ccxgi;
  int czgi;
  float promi;

//:::::(1-read):::::Lee el fichero que contiene el promedio de las diferencias
```

```
//double jacknifing::leerSubset() { //Lee el fichero que contiene la solucion de un subset
    FILE *fichero1;
        if((fichero1=fopen("promDiFF", "r"))==NULL)
        printf("\n\nCaution2: don't read file: Look input file name\n\n");
        else {
                while (!feof(fichero1)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero1,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero1,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                        float sumProm = 0;
                        for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                        fscanf(fichero1,"%f ", &leer.promg);//Lee coordenadas X de la grilla
                     promi = leer.promg;
                      PromDiFF.push_back(promi);//adiciona este elemento al final del vector
                        sumProm = sumProm + promi;
                        }
                        sumProm = sumProm / leer.nz;
                     SumPromDiFF.push_back(sumProm);//adiciona este elemento al final del vector
                        //cout << sumProm << " ";
                  }

                }//fin_while
        }//fin_else
        fclose(fichero1);


//:::::(1-write):::::Imprime un fichero que contiene la media de las diferencias
        FILE *fichero2;
        FILE *outprom;
        outprom= fopen("outJackkprom.dat","w");
```

```
        if((fichero2=fopen("myvelini.dat", "r"))==NULL)
        printf("\n\nCaution1: don't read file: Look input file name\n\n");
        else {
                while (!feof(fichero2)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  float velAire = 0.00;
                  int NoD = leer.nx * leer.nz;
                  for(int i=0; i<NoD; i++) {
                     fscanf(fichero2,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                          if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                          fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                          }//end_if
                          else {
                          fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, PromDiFF[i]);
                          }
                  }
                }//fin_while
        }//fin_else
        fclose(fichero2);
        fclose(outprom);

//:::::(1-writeSum):::::Imprime un fichero que contiene la media de las diferencias / corrdenada X
        FILE *outSumprom;
        outSumprom= fopen("outSumProm.dat","w");
                for(int s=0; s<leer.nx; s++) {
                fprintf(outSumprom, "%-10.2f\n", SumPromDiFF[s]);
                }
        fclose(outSumprom);

//:::::(1-writePen):::::Imprime un fichero que contiene la pendientes
        FILE *outPend;
        outPend= fopen("outPendientes.dat","w");
        float sx, sy, sx2, sy2, pxy, a, b;//Variables temporales de la regresion
        int paso = 25;//Definir
        int NumFor = 19;//Definir
        int ini = 0;
        int fin = paso;
        for(int n=0; n<NumFor; n++) {
                sx = sy = sx2 = sy2 = pxy = 0.0;
                for(int p=ini; p<fin; p++) {
                sx = sx + Vcxg1[p];
                sy = sy + Vccxg1[p];
                sx2 = sx2 + (Vcxg1[p] * Vcxg1[p]);
                sy2 = sy2 + (Vccxg1[p] * Vccxg1[p]);
                pxy = pxy + (Vcxg1[p] * Vccxg1[p]);
                }
                a = ((paso * pxy) - (sx*sy)) / ((paso * sx2)-(sx*sx));
                b = (sy - (a * sx)) / paso;
```

```
                cout << n << ") " << "y = " << a << "* x +" << b << endl;
                float y1, y2;
                y1 = a * ini + b;
                y2 = a * fin + b;
                        float y;
                        int xCorr;
                        fprintf(outPend, "%10.6f\n", a);
                ini = ini + paso;
                fin = fin + paso;
        }
        fclose(outPend);


//:::::(2-read):::::Lee el fichero que contiene la desviacion estandar
    FILE *fichero3;
        if((fichero3=fopen("desstdDiFF", "r"))==NULL)
        printf("\n\nCaution2: don't read file: Look input file name\n\n");
        else {
                while (!feof(fichero3)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero3,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero3,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero3,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero3,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                        for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                        fscanf(fichero3,"%f ", &leer.promg);//Lee coordenadas X de la grilla
                     promi = leer.promg;
                     StdDiFF.push_back(promi);//adiciona este elemento al final del vecto
                        }
                  }

                }//fin_while
```

```
        }//fin_else
        fclose(fichero3);

//:::::(2-write):::::Imprime un fichero que contiene la la desviacion estandar
        FILE *fichero4;
        FILE *outstd;
        outstd= fopen("outJackkstd.dat","w");
        if((fichero4=fopen("myvelini.dat", "r"))==NULL)
        printf("\n\nCaution1: don't read file: Look input file name\n\n");
        else {
                while (!feof(fichero4)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  float velAire = 0.00;
                   int NoD = leer.nx * leer.nz;
                   for(int i=0; i<NoD; i++) {
                      fscanf(fichero4,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                                if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                                fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                                }//end_if
                                else {
                                fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, StdDiFF[i]);
                                }
                   }
                }//fin_while
        }//fin_else
        fclose(fichero4);
        fclose(outstd);
}
```

NOMBRE DEL FICHERO: (figrayos.sh)

```
#!/bin/csh -f

gmtset COLOR_BACKGROUND = 150/150/150

#Carlos Becerra
#Noviembre/2007
set ps = rayos.ps
#set veldata = finalmodel.dat

#inicial_model:
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x [m]":/a100:"z [m]":WeSn -X4 -Y18 -K -P > $ps

blockmean myvelini.dat -R0/10000/-900/17 -I300/300 -V  > tmp.mybmean
surface tmp.mybmean -Gmyvelini.grd -I20 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps
```

```
makecpt -Crainbow -T600/4300/1000 -Z > mypalet.cpt

grdimage myvelini.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour myvelini.grd -JX -R -C500 -L0/10 -O -K >> $ps

#psxy myrayout -JX -R -W1/0 -M -O -K >> $ps

psscale -Cmypalet.cpt -D7/-3/18/0.2h -B:"Velocity [m/s]": -O -K >> $ps
#**********_________**********#
#final_sln:

psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x [m]":/a100:"z [m]":WeSn -X0 -Y18 -K -P -O >> $ps

blockmean myvelinifin.dat -R0/10000/-900/17 -I300/300 -V  > tmp.mybmean
surface tmp.mybmean -Gmyout2.grd -I20 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps

grdimage myout2.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour myout2.grd -JX -R -C500 -L0/10 -O >> $ps
```

NOMBRE DEL FICHERO: (figJacknifing.sh)

```
#!/bin/csh -f

gmtset COLOR_BACKGROUND = 150/150/150

#Carlos Becerra
#Noviembre/2007
set ps = viewJack.ps

#Promedio_Diferencias:

psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Media de las diferencias -velocidad [m/s]-":/a100:"z [m]":WeSn -X3 -Y18 -K -P > $ps

blockmean outJackkprom.dat -R0/10000/-900/17 -I200/200 -V  > tmp.mybmean
surface tmp.mybmean -GoutJackkprom.grd -I10 -R0/10000/-900/17 -V
psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Cgray -T0/20/2 -Z -I > mypalet.cpt

grdimage outJackkprom.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outJackkprom.grd -JX -R -O -K >> $ps

psscale -Cmypalet.cpt -D8/-2.5/15/0.2h -O -K >> $ps
#**********_________**********#
#Desviacion_Estandar:
```

```
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Desviacion Estandar [m/s]":/a100:"z [m]":WeSn -X0 -Y-14
-K -P -O >> $ps

blockmean outJackkstd.dat -R0/10000/-900/17 -I100/100 -V  > tmp.mybmean
surface tmp.mybmean -GoutJackkstd.grd -I10 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Cocean -T0/10/2 -Z -I >mypalet1.cpt

grdimage outJackkstd.grd -JX -R0/10000/-900/17 -Cmypalet1.cpt -O -K -B >> $ps
grdcontour outJackkstd.grd -JX -R -O -K >> $ps

psscale -Cmypalet1.cpt -D8/-2.5/15/0.2h -O >> $ps
```

NOMBRE DEL FICHERO: (Jackknifing.sh)

```
#! /bin/sh
# File: jackknifing.sh
# Invierte n subset de datos (Jacknifing)
# Carlos Becerra
# Diciembre/2007

# Nombre de fichero, input & output
set inicial = mymodelinicial.dat

# Numero de veces que se ejecuta el bucle i
nums=50

        #Compilacion y ejecucion: Generacion de modelo inicial y demas input files
        echo "Iniciando proceso... Compilacion: analisis "
    g++ tiempo.cpp -o exetime
        ./exetime
        g++ analisis.cpp analisisTest.cpp -o exe
        ./exe

        #Inversion del set de datos Completo
        echo "Trabajando sobre modelo Completo"
    g++ tiempo.cpp -o exetime
        ./exetime

        echo "***Trazado de Rayos sobre modelo Completo***"
        ./tt_forward -Mmymodelinicial.dat -GoutCompleto.dat -Rmyrayout -Imyvelini.dat -V

        echo "***Invirtiendo set de datos Completo***"
        ./tt_inverse -Mmymodelinicial.dat -GoutCompleto.dat -V1 -Q1e-3 -SV2000 -SD100 -CVcorrefile.dat

        cp inv_out.smesh.1.1 invtomoCom.dat
```

```
	echo "***Trazado de Rayos sobre modelo final para el set Completo***"
	./tt_forward -MinvtomoCom.dat -GoutCompleto.dat -Rmyrayoutfin -Imyvelinifin.dat -V

	echo "Set de datos Completo listo!"

	echo "Generando vista tomograma (sln): Set Completo"
	./figrayos.sh
	#gv rayos.ps
	cp rayos.ps totaltomo.ps
	rm listDiFF


	#Inversion de los subset de datos Jacknifing
	i=1
while [ "$i" -ne "$nums" ]
do

	echo "Trabajando sobre modelo $i"
g++ tiempo.cpp -o exetime
	./exetime
	echo "***Trazado de Rayos sobre modelo inicial***"
	#./tt_forward -Mmymodelinicial.dat -Gout$i.dat -Rmyrayout -Imyvelini.dat

	echo "***Invirtiendo subset de datos $i***"
	./tt_inverse -Mmymodelinicial.dat -Gout$i.dat -V1 -Q1e-3 -SV2000 -SD100 -CVcorrefile.dat

	#Copiar la grilla solucion a otro fichero
	cp inv_out.smesh.1.1 invtomo$i.dat

	echo "***Trazado de Rayos sobre modelo final para el subset $i***"
	#./tt_forward -Minvtomo$i.dat -Gout$i.dat -Rmyrayoutfin -Imyvelinifin.dat

	echo "subset $i listo!"

	echo "Generando vista tomograma (sln): subset $i"
	#./figrayos.sh
	#gv rayos.ps

	echo "Compilacion: jacknifing "
	g++ jacknifingORI.cpp jacknifingTest.cpp -o exe1
	./exe1

	#Copiar diferencias entre la Solucion Completa y la de el i-Subset
	cp tomoDIFF.dat tomoDIFF$i.dat
	echo "tomoDIFF$i.dat"  >> listDiFF

	i=`expr $i + 1`
	done
```

```
        echo "Calculando Promedio de Diferencias"
        ./stat_smesh -LlistDiFF -Ca > promDiFF
        echo "Calculando Desviacion Estandar"
        ./stat_smesh -LlistDiFF -CrpromDiFF > desstdDiFF

        echo "Generando vista solucion Jackknifing"
        g++ jacknifingFIN.cpp jacknifingTest.cpp -o exe
        ./exe
        ./figJacknifing.sh
        gv viewJack.ps

        g++ tiempo.cpp -o exetime
        ./exetime
        echo "Proceso Terminado"

#****** fin *****#
```

ANEXO 3

Código para realizar análisis Checkerboard

NOMBRE DEL FICHERO: (checkerboardTest.cpp)

```
/* Lee 2 ficheros grilla: (1) modelo checkerboard (2) modelo solucion de checkerboard y genera los outputs para
graficarlos usando GMT */
#include<iostream>
#include "checkerboard.h"

using namespace std;

int main()
{
        checkerboard a(1);
        a.leerGrillas();
}
```

NOMBRE DEL FICHERO: (checkerboard.h)

```
//definicion de la clase checkerboard
#if !defined( _CHECKERBOARD_H_ )
```

```
#define _CHECKERBOARD_H_

class checkerboard
{
        //atributos
        private:
                int Numol; //NUmero de Modelos
    protected:
                void mensage1();
        //miembros publicos
        public:
        checkerboard() {}
        checkerboard(int numol);
        double leerGrillas();
};
#endif // _CHECKERBOARD_H_
```

NOMBRE DEL FICHERO: (checkerboard.cpp)

```
//checkerboard.cpp
//Write by: CarlosBecerra
//Date: Dec/2007
#include<iostream>
#include "checkerboard.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        int nx, nz, Vwater, Vair; //No. de nodos de la grilla y velocidades en el agua y en al aire
        int cxg, czg; //Coordenadas X y Z de la Grilla
        int coox;
        float ccxg; //Cota de la coordenada X de la grilla
        float velg;
        float cooz, veltmp;
        float diffeacm;
}lectura ;

void checkerboard::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }
```

```
checkerboard::checkerboard(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double checkerboard::leerGrillas() { //Lee el fichero que contiene la solucion del set completo
  FILE *fichero;
  lectura leer;
  vector<int> Vcxg;//Vector que contiene Coordenadas X de la Grilla
  vector<int> Vcxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vczg;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> Vczg1;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> ModChecker;//Vector que contiene las velodades del modelo Checkerboard
  vector<float> ModSolucion;//Vector que contiene la solucion producto de la inversion
  vector<float> ModDiferencias;//Vector que contiene las direfencias
  int cxgi;
  float ccxgi;
  int czgi;
  float velmodi;

//:::::(1-read):::::Lee el fichero que contiene el modelo checkerboard
     FILE *fichero1;
        if((fichero1=fopen("modelcheckerboard.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: modelcheckerboard.dat\n\n");
        else {
                while (!feof(fichero1)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero1,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero1,"%d", &leer.czg);//Lee coordenadas X de la grilla
```

```
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                        for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                        fscanf(fichero1,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                    velmodi = leer.velg;
                    ModChecker.push_back(velmodi);//adiciona este elemento al final del vecto
                        }
                  }

                }//fin_while
          }//fin_else
          fclose(fichero1);


//:::::(1-write):::::Imprime un fichero que contiene velocidades del modelo checkerboard
          FILE *fichero2;
          FILE *outprom;
          outprom= fopen("outModelCheckerBoard.dat","w");
          if((fichero2=fopen("myvelini.dat", "r"))==NULL)
          printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
          else {
                while (!feof(fichero2)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  float velAire = 0.00;
                  int NoD = leer.nx * leer.nz;
                  for(int i=0; i<NoD; i++) {
                     fscanf(fichero2,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                        if (leer.veltmp == 333) {//Condicion para que quede con el mismo formato
                        fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                        }//end_if
                        else {
                        fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, ModChecker[i]);
                        }
                  }
                }//fin_while
          }//fin_else
          fclose(fichero2);
          fclose(outprom);

//:::::(2-read):::::Lee el fichero que contiene la solucion de la inversion del modelo chekerboard
      FILE *fichero3;
          if((fichero3=fopen("inv_sln.checkerboard.dat", "r"))==NULL)
          printf("\n\nerror: fichero no encontrado: inv_sln.checkerboard.dat\n\n");
          else {
                while (!feof(fichero3)) {
```

```
                    //:::(1):::Lee informacion de la grilla.
                      fscanf(fichero3,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                    //:::(2):::Lee informacion de la coordenadas x de la grilla
                      for(int x=0; x<leer.nx; x++) {
                        fscanf(fichero3,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                        cxgi = leer.cxg;
                        Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                      }
                    //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                      for(int x=0; x<leer.nx; x++) {
                        fscanf(fichero3,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                        ccxgi = leer.ccxg;
                        Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                      }
                    //:::(4):::Lee informacion de la coordenadas x de la grilla
                      for(int z=0; z<leer.nz; z++) {
                        fscanf(fichero3,"%d", &leer.czg);//Lee coordenadas X de la grilla
                        czgi = leer.czg;
                        Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                      }
                    //:::(5):::Lee informacion de la coordenadas x de la grilla
                      for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                              for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                              fscanf(fichero3,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                         velmodi = leer.velg;
                         ModSolucion.push_back(velmodi);//adiciona este elemento al final del vecto
                              }
                      }

                    }//fin_while
            }//fin_else
            fclose(fichero3);

//:::::(2-write):::::Imprime un fichero que contiene las velocidades de la inversion del modelo chekerboard
            FILE *fichero4;
            FILE *outstd;
            outstd= fopen("outSlnCheckerBoard.dat","w");
            if((fichero4=fopen("myvelini.dat", "r"))==NULL)
            printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
            else {
                    while (!feof(fichero4)) {
                    //:::(2):::Lee informacion de la coordenadas x de la grilla
                      float velAire = 0.00;
                      int NoD = leer.nx * leer.nz;
                      for(int i=0; i<NoD; i++) {
                         fscanf(fichero4,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                              if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                              fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
```

```
                                }//end_if
                                else {
                                fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, ModSolucion[i]);
                                }
                        }
                    }//fin_while
            }//fin_else
            fclose(fichero4);
            fclose(outstd);

//:::::(3-Diferencias):::::calcula las diferencias entre el modelo checkerboard y la inversion del modelo
            float Diffc;
            int NoD = leer.nx * leer.nz;
            for(int c=0; c<NoD; c++) {
            Diffc = abs(ModChecker[c] - ModSolucion[c]);
            ModDiferencias.push_back(Diffc);//adiciona este elemento al final del vector
            }
//:::::(3-write):::::Imprime un fichero que contiene diferencias calculadas
            FILE *fichero5;
            FILE *outdiff;
            outdiff= fopen("outDiffCheckerBoard.dat","w");
            if((fichero5=fopen("myvelini.dat", "r"))==NULL)
            printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
            else {
                    while (!feof(fichero5)) {
                    //:::(2):::Lee informacion de la coordenadas x de la grilla
                      float velAire = 0.00;
                      int NoD = leer.nx * leer.nz;
                      for(int i=0; i<NoD; i++) {
                          fscanf(fichero5,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                                if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                                fprintf(outdiff, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                                }//end_if
                                else {
                                fprintf(outdiff, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, ModDiferencias[i]);
                                }
                        }
                    }//fin_while
            }//fin_else
            fclose(fichero5);
            fclose(outdiff);

}
```

NOMBRE DEL FICHERO: (figCheckerBoard.sh)

```
#!/bin/csh -f
```

```
gmtset COLOR_BACKGROUND = 150/150/150

#Carlos Becerra
#Dic/2007
set ps = viewChecker.ps

#Modelo_Checerboard:
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Modelo Checkerboard":/a100:"z [m]":WeSn -X2 -Y18 -K
-P > $ps

blockmean outModelCheckerBoard.dat -R0/10000/-900/17 -I50/20 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -GoutModelCheckerBoard.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -GoutModelCheckerBoard.grd -I10 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Crainbow -T600/4500/500 -Z >mypalet.cpt

grdimage outModelCheckerBoard.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outModelCheckerBoard.grd -JX -R -O -K >> $ps

psscale -Cmypalet.cpt -D17/5/10/0.2 -O -K >> $ps

#**********_________**********#
#Inversion_Checkerboard:

psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Solucion Checkerboard":/a100:"z [m]":WeSn -X0 -Y-14 -K
-P -O >> $ps

blockmean outSlnCheckerBoard.dat -R0/10000/-900/17 -I50/20 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -GoutSlnCheckerBoard.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -GoutSlnCheckerBoard.grd -I10 -R0/10000/-900/100 -V
#psclip outPoligono.dat -R -J -O -K >> $ps


grdimage outSlnCheckerBoard.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outSlnCheckerBoard.grd -JX -R -O >> $ps
#grdcontour outSlnCheckerBoard.grd -JX -R -C500 -L0/10 -O -K >> $ps

psscale -Cmypalet.cpt -D17/5/10/0.2  -O >> $ps
```

NOMBRE DEL FICHERO: (figCheckerBoardDiff.sh)

```
#!/bin/csh -f

gmtset COLOR_BACKGROUND = 150/150/150

#Carlos Becerra
```

```
#Dic/2007
set ps = viewCheckerDiff.ps

#Diferencias:
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Diferencias Checkerboard":/a100:"z [m]":WeSn -X2 -Y18
-K -P > $ps

blockmean outDiffCheckerBoard.dat -R0/10000/-900/17 -I20/5 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -GoutDiffCheckerBoard.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -GoutDiffCheckerBoard.grd -I10 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Cjet -T0/400/50 -Z >mypalet.cpt

grdimage outDiffCheckerBoard.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outDiffCheckerBoard.grd -JX -R -O -K >> $ps

psscale -Cmypalet.cpt -D17/5/10/0.2 -O >> $ps
```

NOMBRE DEL FICHERO: (checkerboard.sh)

```
#! /bin/sh
# File: checkerboard.sh
# (Checkerboard Test)
# Carlos Becerra
# Diciembre/2007


# Nombre de fichero, input & output
set inicial = mymodelinicial.dat

        echo "Iniciando proceso..."
    g++ tiempo.cpp -o exetime
        ./exetime

        echo "***Trazado de Rayos sobre modelo Completo***"
    #./tt_forward -Mmymodelinicial.dat -GoutCompleto.dat -Rmyrayout -Imyvelini.dat -V

        echo "***Invirtiendo set de datos Completo***"
    #./tt_inverse -Mmymodelinicial.dat -GoutCompleto.dat -V1 -Q1e-3 -SV2000 -SD100 -CVcorrefile.dat

        #cp inv_out.smesh.1.1 invtomoCom.dat
        #cp invtomoCom.dat inv_sln.tomoCom

        echo "Generando modelo checkboard..."
        #./edit_smesh inv_sln.sintetico -Ca >modelcheckerboard.dat
        #./edit_smesh inv_sln.sintetico2 -Cd50/4000/6000/-1300/-1100 >modelcheckerboard.dat
        ./edit_smesh inv_sln.tomoCom -Cc20/1000/40 >modelcheckerboard.dat
```

```
        echo "***Invirtiendo modelo checkerboard***"
        ./tt_inverse -Mmodelcheckerboard.dat -GoutCompleto.dat -V1 -Q1e-3 -SV2000 -SD100 -CVcorrefile.dat
        cp inv_out.smesh.1.1 inv_sln.checkerboard.dat
        echo "leyendo grillas..."
        g++ checkerboard.cpp checkerboardTest.cpp -o exe
        ./exe

        ./figCheckerBoard.sh
        gv viewChecker.ps
        ./figCheckerBoardDiff.sh
        gv viewCheckerDiff.ps


    g++ tiempo.cpp -o exetime
        ./exetime
        echo "Proceso Terminado"
#****** fin *****#
```

ANEXO 4

Código para realizar análisis Monte Carlo

NOMBRE DEL FICHERO: (montecarlogenTest.cpp)

```
// montecarlogenTest.cpp-programa
/* Lee y perturba velocidades en el modelo inicial*/
#include<iostream>
#include "montecarlogen.h"

using namespace std;


int main()
{
        montecarlo t(5);
        t.perturba(5);//Amplitud de perturbaciones (%)
}
```

NOMBRE DEL FICHERO: (montecarlogen.h)

```
//definicion de la clase montecarlo
#if !defined( _MONTECARLOGEN_H_ )
#define _MONTECARLOSGEN_H_
```

```
class montecarlo
{
        //atributos
        private:
                int Amp; //Amplitud de perturbaciones (%)
    protected:
                void mensage1();
        //miembros publicos
        public:
        montecarlo() {}
        montecarlo(int Amp);
        double perturba(int Ampl);
};
#endif // _MONTECARLOGEN_H_
```

NOMBRE DEL FICHERO: (montecarlogen.cpp)

```
//montecarlogen.cpp
//Write by: CarlosBecerra
//Date: Dec/2007
#include<iostream>
#include "montecarlogen.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        int nx, nz, Vwater, Vair; //No. de nodos de la grilla y velocidades en el agua y en al aire
        int cxg, czg; //Coordenadas X y Z de la Grilla
        int coox;
        float ccxg; //Cota de la coordenada X de la grilla
        float velg;
        float cooz, veltmp;
        float diffeacm;
}lectura ;

void montecarlo::mensage1() {
        cout << "Amplitud negetiva.\n";
        }

montecarlo::montecarlo(int Amp) {
```

```
        if (Amp < 0) {
                mensage1();
                Amp= -Amp;
                }
                Amp= Amp;
}

double montecarlo::perturba(int Ampl) { //Genera un fichero con los primeros arribos perturbados
  lectura leer;
  vector<int> Vcxg;//Vector que contiene Coordenadas X de la Grilla
  vector<int> Vcxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vczg;//Vector que contiene Coordenadas Z de la Grilla
  vector<int> Vczg1;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> VelOri;//Vector que contiene las velodades del modelo montecarlo
  //vector<float> ModSolucion;//Vector que contiene la solucion producto de la inversion
  //vector<float> ModDiferencias;//Vector que contiene las direfencias
  int cxgi;
  float ccxgi;
  int czgi;
  float velmodi;

    FILE *fichero1;//Lee el fichero que contiene los primero arribos (tiempos de viaje)
        if((fichero1=fopen("mymodelinicial.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: outCompleto.dat\n\n");
        else {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero1, "%d %d %d %d\n", &leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1, "%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas Z de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero1,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de velocidades
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
```

```
                    for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                    fscanf(fichero1,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                velmodi = leer.velg;
                VelOri.push_back(velmodi);//adiciona este elemento al final del vecto
                    }
                }

        }//fin_else
        fclose(fichero1);


long tm;
int ntime;
time_t segundos;
time(&segundos);
srand((unsigned)time(0)); //cambio semilla
float Lo, A, B, Velpert;

 FILE *outpert;
 outpert= fopen("modelmontecarlo.dat","w");//nombre del fichero que contiene los tiempos de viaje perturbados
                //:::(1):::Escribe informacion de la grilla.
                  fprintf(outpert, "%d %d %d %d\n", leer.nx, leer.nz, leer.Vwater, leer.Vair);
                //:::(2):::Escribe informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    if (x == (leer.nx-1))
                    fprintf(outpert, "%d\n", Vcxg1[x]);
                    else {
                    fprintf(outpert, "%d ", Vcxg1[x]); }
                  }
                //:::(3):::Escribe informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    if (x == (leer.nx-1))
                    fprintf(outpert, "%.2f\n", Vccxg1[x]);
                    else {
                    fprintf(outpert, "%.2f ", Vccxg1[x]); }
                  }
                //:::(4):::Escribe informacion de la coordenadas Z de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    if (z == (leer.nz-1))
                    fprintf(outpert, "%d\n", Vczg1[z]);
                    else {
                    fprintf(outpert, "%d ", Vczg1[z]); }
                  }
                //:::(5):::Escribe informacion de velocidades
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                            for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla

                            time(&segundos);
                            int nmaximo= 100;
```

```
                                int mynum; //Variable del Numero aleatorio usado para eliminar un dato
                                mynum = rand()%nmaximo + 1;
                                Lo = VelOri[z] * (Ampl/100.0);
                                A = VelOri[z] + Lo;
                                B = VelOri[z] - Lo;
                                Velpert = B + ((mynum*(A-B))/(nmaximo-1));

                            if (z == (leer.nz-1))
                            fprintf(outpert, "%-10.2f\n", Velpert);
                            else {
                            fprintf(outpert, "%-10.2f ", Velpert); }

                                }
                        }
        fclose(outpert);
}
```

NOMBRE DEL FICHERO: (montecarloTest.cpp)

```
// montecarlo.cpp-programa
/* Lee 2 ficheros grilla: (1) modelo montecarlo (2) modelo solucion de montecarlo y genera los outputs para graficarlos usando GMT */
#include<iostream>
#include "montecarlo.h"

using namespace std;


int main()
{
        montecarlo a(1);
        a.leerGrillas();
}
```

NOMBRE DEL FICHERO: (montecarlo.h)

```
//definicion de la clase montecarlo
#if !defined( _MONTECARLO_H_ )
#define _MONTECARLO_H_

class montecarlo
{
        //atributos
        private:
                int Numol; //NUmero de Modelos
    protected:
                void mensage1();
```

```
        //miembros publicos
        public:
        montecarlo() {}
        montecarlo(int numol);
        double leerGrillas();
};
#endif // _MONTECARLO_H_
```

NOMBRE DEL FICHERO: (montecarlo.cpp)

```
//montecarlo.cpp
//Write by: CarlosBecerra
//Date: Dec/2007
#include<iostream>
#include "montecarlo.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        int nx, nz, Vwater, Vair; //No. de nodos de la grilla y velocidades en el agua y en al aire
        int cxg, czg; //Coordenadas X y Z de la Grilla
        int coox;
        float ccxg; //Cota de la coordenada X de la grilla
        float velg;
        float cooz, veltmp;
        float diffeacm;
}lectura ;

void montecarlo::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }

montecarlo::montecarlo(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double montecarlo::leerGrillas() { //Lee el fichero que contiene la solucion del set completo
```

```
FILE *fichero;
lectura leer;
vector<int> Vcxg;//Vector que contiene Coordenadas X de la Grilla
vector<int> Vcxg1;//Vector que contiene Coordenadas X de la Grilla
vector<float> Vccxg;//Vector que contiene Coordenadas X de la Grilla
vector<float> Vccxg1;//Vector que contiene Coordenadas X de la Grilla
vector<float> Vczg;//Vector que contiene Coordenadas Z de la Grilla
vector<float> Vczg1;//Vector que contiene Coordenadas Z de la Grilla
vector<float> Modmontecarlo;//Vector que contiene las velodades del modelo montecarlo
vector<float> ModSolucion;//Vector que contiene la solucion producto de la inversion
vector<float> ModDiferencias;//Vector que contiene las direfencias
int cxgi;
float ccxgi;
int czgi;
float velmodi;

//:::::(1-read):::::Lee el fichero que contiene el modelo montecarlo
    FILE *fichero1;
        if((fichero1=fopen("modelmontecarlo.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: modelmontecarlo.dat\n\n");
        else {
                while (!feof(fichero1)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero1,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero1,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {///Una iteracion para cada nodo X de la grilla
                          for(int z=0; z<leer.nz; z++) {///Una iteracion para cada nodo Z de la grilla
                          fscanf(fichero1,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                     velmodi = leer.velg;
                     Modmontecarlo.push_back(velmodi);//adiciona este elemento al final del vecto
                          }
```

```
                    }
                }//fin_while
        }//fin_else
        fclose(fichero1);

//:::::(1-write):::::Imprime un fichero que contiene velocidades del modelo montecarlo
        FILE *fichero2;
        FILE *outprom;
        outprom= fopen("outModelmontecarlo.dat","w");
        if((fichero2=fopen("myvelini.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
        else {
                while (!feof(fichero2)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  float velAire = 0.00;
                  int NoD = leer.nx * leer.nz;
                  for(int i=0; i<NoD; i++) {
                     fscanf(fichero2,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                          if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                          fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                          }//end_if
                          else {
                          fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, Modmontecarlo[i]);
                          }
                  }
                }//fin_while
        }//fin_else
        fclose(fichero2);
        fclose(outprom);

//:::::(2-read):::::Lee el fichero que contiene la solucion de la inversion del modelo chekerboard
      FILE *fichero3;
        if((fichero3=fopen("inv_sln.montecarlo.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: inv_sln.montecarlo.dat\n\n");
        else {
                while (!feof(fichero3)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero3,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero3,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero3,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
```

```
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero3,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                          for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                          fscanf(fichero3,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                     velmodi = leer.velg;
                     ModSolucion.push_back(velmodi);//adiciona este elemento al final del vecto
                          }
                  }

                }//fin_while
        }//fin_else
        fclose(fichero3);

//:::::(2-write):::::Imprime un fichero que contiene las velocidades de la inversion del modelo chekerboard
        FILE *fichero4;
        FILE *outstd;
        outstd= fopen("outSlnmontecarlo.dat","w");
        if((fichero4=fopen("myvelini.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
        else {
                while (!feof(fichero4)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  float velAire = 0.00;
                  int NoD = leer.nx * leer.nz;
                  for(int i=0; i<NoD; i++) {
                     fscanf(fichero4,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                          if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                          fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                          }//end_if
                          else {
                          fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, ModSolucion[i]);
                          }
                  }
                }//fin_while
        }//fin_else
        fclose(fichero4);
        fclose(outstd);

//:::::(3-Diferecias):::::calcula las diferencias entre el modelo montecarlo y la inversion del modelo
        float Diffc;
```

```
        int NoD = leer.nx * leer.nz;
        for(int c=0; c<NoD; c++) {
        Diffc = abs(Modmontecarlo[c] - ModSolucion[c]);
        ModDiferencias.push_back(Diffc);//adiciona este elemento al final del vector
        }
//:::::(3-write):::::Imprime un fichero que contiene diferencias calculadas
        FILE *fichero5;
        FILE *outdiff;
        outdiff= fopen("outDiffmontecarlo.dat","w");
        if((fichero5=fopen("myvelini.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
        else {
                while (!feof(fichero5)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  float velAire = 0.00;
                  int NoD = leer.nx * leer.nz;
                  for(int i=0; i<NoD; i++) {
                      fscanf(fichero5,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                            if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                            fprintf(outdiff, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                            }//end_if
                            else {
                            fprintf(outdiff, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, ModDiferencias[i]);
                            }
                  }
                }//fin_while
        }//fin_else
        fclose(fichero5);
        fclose(outdiff);
}
```

NOMBRE DEL FICHERO: (montecarloFIN.cpp)

```
//montecarloFIN.cpp
//Write by: CarlosBecerra
//Date: Dec/2007
#include<iostream>
#include "montecarlo.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;
```

```
typedef struct {
        int nx, nz, Vwater, Vair; //No. de nodos de la grilla y velocidades en el agua y en al aire
        int cxg, czg; //Coordenadas X y Z de la Grilla
        int coox;
        float ccxg; //Cota de la coordenada X de la grilla
        float promg;
        float cooz, veltmp;
        float diffeacm;
}lectura ;

void montecarlo::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }

montecarlo::montecarlo(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double montecarlo::leerGrillas() { //Lee el fichero que contiene la solucion del set completo
  FILE *fichero;
  lectura leer;
  vector<int> Vcxg;//Vector que contiene Coordenadas X de la Grilla
  vector<int> Vcxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vczg;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> Vczg1;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> Prommontecarlo;//Vector que contiene el promedio de las Diferencias
  vector<float> Stdmontecarlo;//Vector que contiene la Desviacion Standar
  int cxgi;
  float ccxgi;
  int czgi;
  float promi;

//:::::(1-read):::::Lee el fichero que contiene el promedio de las diferencias
    FILE *fichero1;
        if((fichero1=fopen("prommontecarlo", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: prommontecarlo\n\n");
        else {
                while (!feof(fichero1)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero1,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
```

```
                    fscanf(fichero1,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero1,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                          for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                          fscanf(fichero1,"%f ", &leer.promg);//Lee coordenadas X de la grilla
                     promi = leer.promg;
                     Prommontecarlo.push_back(promi);//adiciona este elemento al final del vecto
                          }
                  }

                }//fin_while
          }//fin_else
          fclose(fichero1);


//:::::(1-write):::::Imprime un fichero que contiene la media de las diferencias
          FILE *fichero2;
          FILE *outprom;
          outprom= fopen("outprommontecarlo.dat","w");
          if((fichero2=fopen("myvelini.dat", "r"))==NULL)
          printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
          else {
                  while (!feof(fichero2)) {
                  //:::(2):::Lee informacion de la coordenadas x de la grilla
                    float velAire = 0.00;
                    int NoD = leer.nx * leer.nz;
                    for(int i=0; i<NoD; i++) {
                        fscanf(fichero2,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                          if (leer.veltmp == 333) {//Condicion para que quede con el mismo formato
                          fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                          }//end_if
                          else {
                          fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, Prommontecarlo[i]);
```

```
                        }
                    }
                }//fin_while
            }//fin_else
            fclose(fichero2);
            fclose(outprom);

//:::::(2-read):::::Lee el fichero que contiene la desviacion estandar
        FILE *fichero3;
            if((fichero3=fopen("desstdmontecarlo", "r"))==NULL)
            printf("\n\nerror: fichero no encontrado: desstdmontecarlo\n\n");
            else {
                    while (!feof(fichero3)) {
                    //:::(1):::Lee informacion de la grilla.
                      fscanf(fichero3,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                    //:::(2):::Lee informacion de la coordenadas x de la grilla
                      for(int x=0; x<leer.nx; x++) {
                        fscanf(fichero3,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                        cxgi = leer.cxg;
                        Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                      }
                    //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                      for(int x=0; x<leer.nx; x++) {
                        fscanf(fichero3,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                        ccxgi = leer.ccxg;
                        Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                      }
                    //:::(4):::Lee informacion de la coordenadas x de la grilla
                      for(int z=0; z<leer.nz; z++) {
                        fscanf(fichero3,"%d", &leer.czg);//Lee coordenadas X de la grilla
                        czgi = leer.czg;
                        Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                      }
                    //:::(5):::Lee informacion de la coordenadas x de la grilla
                      for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                              for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                              fscanf(fichero3,"%f ", &leer.promg);//Lee coordenadas X de la grilla
                         promi = leer.promg;
                         Stdmontecarlo.push_back(promi);//adiciona este elemento al final del vecto
                              }
                      }

                    }//fin_while
            }//fin_else
            fclose(fichero3);

//:::::(2-write):::::Imprime un fichero que contiene la la desviacion estandar
            FILE *fichero4;
            FILE *outstd;
```

```
        outstd= fopen("outdesmontecarlo.dat","w");
        if((fichero4=fopen("myvelini.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
        else {
                while (!feof(fichero4)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  float velAire = 0.00;
                  int NoD = leer.nx * leer.nz;
                  for(int i=0; i<NoD; i++) {
                     fscanf(fichero4,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                          if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                          fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                          }//end_if
                          else {
                          fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, Stdmontecarlo[i]);
                          }
                  }
                }//fin_while
        }//fin_else
        fclose(fichero4);
        fclose(outstd);
}
```

NOMBRE DEL FICHERO: (figmontecarlo.sh)

```
#!/bin/csh -f

gmtset COLOR_BACKGROUND = 150/150/150

#Carlos Becerra
#Dic/2007
set ps = viewmontecarlo.ps

#Modelo_MonteCarlo:
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Modelo remuestreado - MonteCarlo":/a100:"z [m]":WeSn
-X2 -Y18 -K -P > $ps

blockmean outModelmontecarlo.dat -R0/10000/-900/17 -I10/10 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -GoutModelmontecarlo.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -GoutModelmontecarlo.grd -I10 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Crainbow -T500/4500/500 -Z >mypalet.cpt

grdimage outModelmontecarlo.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outModelmontecarlo.grd -JX -R -O -K >> $ps
```

```
psscale -Cmypalet.cpt -D17/5/10/0.2 -O -K >> $ps

#**********_________**********#
#Inversion_MonteCarlo:

psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Solucion modelo MonteCarlo":/a100:"z [m]":WeSn -X0 -Y-
14 -K -P -O >> $ps

blockmean outSlnmontecarlo.dat -R0/10000/-900/17 -I10/10 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -GoutSlnmontecarlo.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -GoutSlnmontecarlo.grd -I10 -R0/10000/-900/100 -V
#psclip outPoligono.dat -R -J -O -K >> $ps


grdimage outSlnmontecarlo.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outSlnmontecarlo.grd -JX -R -O >> $ps
#grdcontour outSlnmontecarlo.grd -JX -R -C500 -L0/10 -O -K >> $ps

psscale -Cmypalet.cpt -D17/5/10/0.2  -O >> $ps
```

NOMBRE DEL FICHERO: (figmontecarloDiff.sh)

```
#!/bin/csh -f

gmtset COLOR_BACKGROUND = 150/150/150

#Carlos Becerra
#Dic/2007
set ps = viewmontecarloDiff.ps

#Diferencias:
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba2000:"x[m] Diferencias modelo MonteCarlo":/a100:"z [m]":WeSn -X2
-Y18 -K -P > $ps

blockmean outDiffmontecarlo.dat -R0/10000/-900/17 -I10/10 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -GoutDiffmontecarlo.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -GoutDiffmontecarlo.grd -I10 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Cjet -T0/1000/100 -Z >mypalet.cpt

grdimage outDiffmontecarlo.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outDiffmontecarlo.grd -JX -R -O -K >> $ps

psscale -Cmypalet.cpt -D17/5/10/0.2 -O >> $ps
```

NOMBRE DEL FICHERO: (figmontecarloFIN.sh)

```
#!/bin/csh -f
gmtset COLOR_BACKGROUND = 150/150/150

#Carlos Becerra
#Dic/2007
set ps = viewmontecarloFIN.ps

#Modelo_Rta MonteCarlo:
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Modelo Promedio MonteCarlo":/a100:"z [m]":WeSn -X2
-Y18 -K -P > $ps

blockmean outprommontecarlo.dat -R0/10000/-900/17 -I100/100 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -Goutprommontecarlo.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -Goutprommontecarlo.grd -I10 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Crainbow -T0/4500/500 -Z >mypalet.cpt

grdimage outprommontecarlo.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outprommontecarlo.grd -JX -R -O -K >> $ps

psscale -Cmypalet.cpt -D17/5/10/0.2 -O -K >> $ps

#**********_________**********#
#Desviacion_Estandar:

psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Desviacion Estandar MonteCarlo":/a100:"z [m]":WeSn
-X0 -Y-14 -K -P -O >> $ps

blockmean outdesmontecarlo.dat -R0/10000/-900/17 -I20/5 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -Goutdesmontecarlo.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -Goutdesmontecarlo.grd -I10 -R0/10000/-900/100 -V
psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Cocean -T0/5000/500 -Z -I>mypalet1.cpt

grdimage outdesmontecarlo.grd -JX -R0/10000/-900/17 -Cmypalet1.cpt -O -K -B >> $ps
grdcontour outdesmontecarlo.grd -JX -R -O >> $ps
#grdcontour outdesmontecarlo.grd -JX -R -C500 -L0/10 -O -K >> $ps

psscale -Cmypalet1.cpt -D17/5/10/0.2  -O >> $ps
```

NOMBRE DEL FICHERO: (montecarlo.sh)

```
#!/bin/csh -f
#! /bin/sh
# File: montecarlo.sh
```

```
# (Analisis Monte Carlo)
# Carlos Becerra
# Diciembre/2007

# Nombre de fichero, input & output
set inicial = mymodelinicial.dat

# Numero de veces que se ejecuta el bucle i
nums=50

        echo "Iniciando proceso..."
    g++ tiempo.cpp -o exetime
        ./exetime

        echo "***Trazado de Rayos sobre modelo Completo***"
    #./tt_forward -Mmymodelinicial.dat -GoutCompleto.dat -Rmyrayout -Imyvelini.dat -V

        rm listMontecarlo


        #Inversion de los i modelos MonteCarlo
        i=1
    while [ "$i" -ne "$nums" ]
    do

        echo "Generando modelo $i MonteCarlo..."
        g++ montecarlogen.cpp montecarlogenTest.cpp -o exe
        ./exe

        echo "***Invirtiendo modelo $i ***"
        ./tt_inverse -Mmodelmontecarlo.dat -GoutCompleto.dat -V1 -Q1e-3 -SV2000 -SD100 -CVcorrefile.dat
        cp inv_out.smesh.1.1 inv_sln.montecarlo.dat
        echo "leyendo grillas..."
        g++ montecarlo.cpp montecarloTest.cpp -o exe
        ./exe

        #./figmontecarlo.sh
        #gv viewmontecarlo.ps
        #./figmontecarloDiff.sh
        #gv viewmontecarloDiff.ps

        #Copiar Solucion de la inversion para la muestra i
        cp inv_sln.montecarlo.dat slnmontecarlo$i.dat
        echo "slnmontecarlo$i.dat"  >> listMontecarlo
        echo "modelo $i listo!"

        i=`expr $i + 1`
        done
```

```
        echo "Calculando Promedio"
        ./stat_smesh -LlistMontecarlo -Ca > prommontecarlo
        echo "Calculando Desviacion Estandar"
        ./stat_smesh -LlistMontecarlo -Crprommontecarlo > desstdmontecarlo

        echo "Generando vista sln"
        g++ montecarloFIN.cpp montecarloTest.cpp -o exe
        ./exe
        ./figmontecarloFIN.sh
        gv viewmontecarloFIN.ps

    g++ tiempo.cpp -o exetime
        ./exetime
        echo "Proceso Terminado"

#****** fin *****#
```

ANEXO 5

Código para realizar análisis Bootstrapping

NOMBRE DEL FICHERO: (bootstrapgenTest.cpp)

```
// bootstrapgenTest.cpp-programa
/* Lee y perturba tiempos de viaje*/
#include<iostream>
#include "bootstrapgen.h"

using namespace std;


int main()
{
        bootstrap t(10);
        t.perturba(10);//Amplitud de perturbaciones (%)
}
```

NOMBRE DEL FICHERO: (bootstrapgen.h)

```
//definicion de la clase bootstrap
#if !defined( _BOOTSTRAPGEN_H_ )
```

```
#define _BOOTSTRAPGEN_H_

class bootstrap
{
        //atributos
        private:
                int Amp; //Amplitud de perturbaciones (%)
    protected:
                void mensage1();
        //miembros publicos
        public:
        bootstrap() {}
        bootstrap(int Amp);
        double perturba(int Ampl);
};
#endif // _BOOTSTRAPGEN_H_
```

NOMBRE DEL FICHERO: (bootstrapgen.cpp)

```
//bootstrapgen.cpp
//Write by: CarlosBecerra
//Date: Dec/2007
#include<iostream>
#include "bootstrapgen.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        int nShot; //Numero de disparos en el set
        char id[1]; //Letra que identifica: fuente o receptor
        float coorXsou, cotasou; //Coordenadas X y elevacion de la fuente
        float coorXrec, cotarec; //Coordenadas X y elevacion del receptor
        int nrec; //Numero de receptores por disparo
        int code; //INDICA: 0/refraccion; 1/refleccion
        float tpa; //Tiempo de primer arribo
        float dt;
}lectura ;

void bootstrap::mensage1() {
        cout << "Amplitud negetiva.\n";
        }
```

```
bootstrap::bootstrap(int Amp) {
        if (Amp < 0) {
                mensage1();
                Amp= -Amp;
                }
                Amp= Amp;
}

double bootstrap::perturba(int Ampl) { //Genera un fichero con los primeros arribos perturbados
  FILE *outpert;
  outpert= fopen("modelbootstrap.dat","w");//nombre del fichero que contiene los tiempos de viaje perturbados
  lectura leer;
 long tm;
 int ntime;
 time_t segundos;
 time(&segundos);
 srand((unsigned)time(0)); //cambio semilla
 float Lo, A, B, tpapert;

    FILE *fichero1;//Lee el fichero que contiene los primero arribos (tiempos de viaje)
        if((fichero1=fopen("outCompleto.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: outCompleto.dat\n\n");
        else {
                //:::(0):::Lee informacion numero de disparos del Set de datos
                  fscanf(fichero1, "%d", &leer.nShot);
                  fprintf(outpert, "%d\n", leer.nShot);
                //:::(1):::Lee informacion de cada receptor por cada disparo
                  for(int s=0; s<leer.nShot; s++) {//Una iteracion para cada disparo del set
                  time(&segundos);
              tm = clock(); //Esperar 1 segundo para generar otro NUmero Aleatorio
                   fscanf(fichero1, "%s %f %f %d", &leer.id, &leer.coorXsou, &leer.cotasou, &leer.nrec);//Lee
informacion de la fuente
                  fprintf(outpert, "%s %10.2f %10.2f %10d\n", leer.id, leer.coorXsou, leer.cotasou, leer.nrec);
                          for(int r=0; r<leer.nrec; r++) {//Una iteracion para cada receptor
                          fscanf(fichero1, "%s %f %f %d %f %f", &leer.id, &leer.coorXrec, &leer.cotarec,
&leer.code, &leer.tpa, &leer.dt);//informacion fuente
                          int nmaximo= 100;
                          int mynum; //Variable del Numero aleatorio usado para eliminar un dato
                          mynum = rand()%nmaximo + 1;
                          Lo = leer.tpa * (Ampl/100.0);
                          A = leer.tpa + Lo;
                          B = leer.tpa - Lo;
                          tpapert = B + ((mynum*(A-B))/(nmaximo-1));
                          fprintf(outpert, "%s %10.2f %10.2f %5d %12.5f %10.2f\n", leer.id, leer.coorXrec,
leer.cotarec, leer.code, tpapert, leer.dt);
                          }
                  }
        }//fin_else
```

```
        fclose(fichero1);
        fclose(outpert);

}
```

NOMBRE DEL FICHERO: (bootstrappingTest.cpp)

```
// bootstrapping.cpp-programa
/* Lee 2 ficheros grilla: (1) modelo bootstrap (2) modelo solucion de bootstrap y genera los outputs para graficarlos
usando GMT */
#include<iostream>
#include "bootstrapping.h"

using namespace std;


int main()
{
        bootstrap a(1);
        a.leerGrillas();
}
```

NOMBRE DEL FICHERO: (bootstrapping.h)

```
//definicion de la clase bootstrap
#if !defined( _BOOTSTRAPPING_H_ )
#define _BOOTSTRAPPING_H_

class bootstrap
{
        //atributos
        private:
                int Numol; //NUmero de Modelos
    protected:
                void mensage1();
        //miembros publicos
        public:
        bootstrap() {}
        bootstrap(int numol);
        double leerGrillas();
};
#endif // _BOOTSTRAPPING_H_
```

NOMBRE DEL FICHERO: (bootstrapping.cpp)

```
//bootstrapping.cpp
//Write by: CarlosBecerra
```

```
//Date: Dec/2007
#include<iostream>
#include "bootstrapping.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        int nx, nz, Vwater, Vair; //No. de nodos de la grilla y velocidades en el agua y en al aire
        int cxg, czg; //Coordenadas X y Z de la Grilla
        int coox;
        float ccxg; //Cota de la coordenada X de la grilla
        float velg;
        float cooz, veltmp;
        float diffeacm;
}lectura ;

void bootstrap::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }

bootstrap::bootstrap(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double bootstrap::leerGrillas() { //Lee el fichero que contiene la solucion del set completo
  FILE *fichero;
  lectura leer;
  vector<int> Vcxg;//Vector que contiene Coordenadas X de la Grilla
  vector<int> Vcxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vczg;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> Vczg1;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> ModBootstrap;//Vector que contiene las velodades del modelo bootstrap
  vector<float> ModSolucion;//Vector que contiene la solucion producto de la inversion
  vector<float> ModDiferencias;//Vector que contiene las direfencias
  int cxgi;
  float ccxgi;
```

```
  int czgi;
  float velmodi;

//:::::(1-read):::::Lee el fichero que contiene el modelo bootstrap
      FILE *fichero1;
            if((fichero1=fopen("invtomoCom.dat", "r"))==NULL)
            printf("\n\nerror: fichero no encontrado: modelbootstrap.dat\n\n");
            else {
                        while (!feof(fichero1)) {
                        //:::(1):::Lee informacion de la grilla.
                          fscanf(fichero1,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                        //:::(2):::Lee informacion de la coordenadas x de la grilla
                          for(int x=0; x<leer.nx; x++) {
                            fscanf(fichero1,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                            cxgi = leer.cxg;
                            Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                          }
                        //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                          for(int x=0; x<leer.nx; x++) {
                            fscanf(fichero1,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                            ccxgi = leer.ccxg;
                            Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                          }
                        //:::(4):::Lee informacion de la coordenadas x de la grilla
                          for(int z=0; z<leer.nz; z++) {
                            fscanf(fichero1,"%d", &leer.czg);//Lee coordenadas X de la grilla
                            czgi = leer.czg;
                            Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                          }
                        //:::(5):::Lee informacion de la coordenadas x de la grilla
                          for(int x=0; x<leer.nx; x++) {///Una iteracion para cada nodo X de la grilla
                                    for(int z=0; z<leer.nz; z++) {///Una iteracion para cada nodo Z de la grilla
                                    fscanf(fichero1,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                             velmodi = leer.velg;
                             ModBootstrap.push_back(velmodi);//adiciona este elemento al final del vecto
                                    }
                          }

                        }//fin_while
            }//fin_else
            fclose(fichero1);


//:::::(1-write):::::Imprime un fichero que contiene velocidades del modelo bootstrap
            FILE *fichero2;
            FILE *outprom;
            outprom= fopen("outModelbootstrap.dat","w");
            if((fichero2=fopen("myvelini.dat", "r"))==NULL)
            printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
```

```
        else {
                while (!feof(fichero2)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  float velAire = 0.00;
                  int NoD = leer.nx * leer.nz;
                  for(int i=0; i<NoD; i++) {
                     fscanf(fichero2,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                          if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                          fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                          }//end_if
                          else {
                          fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, ModBootstrap[i]);
                          }
                  }
                }//fin_while
        }//fin_else
        fclose(fichero2);
        fclose(outprom);

//:::::(2-read):::::Lee el fichero que contiene la solucion de la inversion del modelo chekerboard
      FILE *fichero3;
        if((fichero3=fopen("inv_sln.bootstrap.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: inv_sln.bootstrap.dat\n\n");
        else {
                while (!feof(fichero3)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero3,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero3,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero3,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero3,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                          for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
```

```
                        fscanf(fichero3,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                    velmodi = leer.velg;
                    ModSolucion.push_back(velmodi);//adiciona este elemento al final del vecto
                        }
                }

            }//fin_while
        }//fin_else
        fclose(fichero3);

//:::::(2-write):::::Imprime un fichero que contiene las velocidades de la inversion del modelo chekerboard
        FILE *fichero4;
        FILE *outstd;
        outstd= fopen("outSlnbootstrap.dat","w");
        if((fichero4=fopen("myvelini.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
        else {
            while (!feof(fichero4)) {
            //:::(2):::Lee informacion de la coordenadas x de la grilla
              float velAire = 0.00;
              int NoD = leer.nx * leer.nz;
              for(int i=0; i<NoD; i++) {
                fscanf(fichero4,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                    if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                    fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                    }//end_if
                    else {
                    fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, ModSolucion[i]);
                    }
              }
            }//fin_while
        }//fin_else
        fclose(fichero4);
        fclose(outstd);

//:::::(3-Diferecias):::::calcula las diferencias entre el modelo bootstrap y la inversion del modelo
        float Diffc;
        int NoD = leer.nx * leer.nz;
        for(int c=0; c<NoD; c++) {
        Diffc = abs(ModBootstrap[c] - ModSolucion[c]);
        ModDiferencias.push_back(Diffc);//adiciona este elemento al final del vector
        }
//:::::(3-write):::::Imprime un fichero que contiene diferencias calculadas
        FILE *fichero5;
        FILE *outdiff;
        outdiff= fopen("outDiffbootstrap.dat","w");
        if((fichero5=fopen("myvelini.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
```

```
        else {
                while (!feof(fichero5)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  float velAire = 0.00;
                  int NoD = leer.nx * leer.nz;
                  for(int i=0; i<NoD; i++) {
                     fscanf(fichero5,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                          if (leer.veltmp == 333) {///Condicion para que quede con el mismo formato
                          fprintf(outdiff, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                          }//end_if
                          else {
                          fprintf(outdiff, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, ModDiferencias[i]);
                          }
                  }
                }//fin_while
        }//fin_else
        fclose(fichero5);
        fclose(outdiff);

}//FIN_FIN
```

NOMBRE DEL FICHERO: (bootstrapFIN.cpp)

```
//BootstrapFIN.cpp
//Write by: CarlosBecerra
//Date: Dec/2007
#include<iostream>
#include "bootstrapping.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        int nx, nz, Vwater, Vair; //No. de nodos de la grilla y velocidades en el agua y en al aire
        int cxg, czg; //Coordenadas X y Z de la Grilla
        int coox;
        float ccxg; //Cota de la coordenada X de la grilla
        float promg;
        float cooz, veltmp;
        float diffeacm;
}lectura ;
```

```
void bootstrap::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }

bootstrap::bootstrap(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double bootstrap::leerGrillas() { //Lee el fichero que contiene la solucion del set completo
  FILE *fichero;
  lectura leer;
  vector<int> Vcxg;//Vector que contiene Coordenadas X de la Grilla
  vector<int> Vcxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vczg;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> Vczg1;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> PromBootstrap;//Vector que contiene el promedio de las Diferencias
  vector<float> StdBootstrap;//Vector que contiene la Desviacion Standar
  int cxgi;
  float ccxgi;
  int czgi;
  float promi, desi;
  float sumdes = 0.0;

//:::::(1-read):::::Lee el fichero que contiene el promedio de las diferencias
    FILE *fichero1;
        if((fichero1=fopen("mediaBootstrap", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: mediaBootstrap\n\n");
        else {
                while (!feof(fichero1)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero1,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                   }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero1,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
```

```
                    }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero1,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                        for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                        fscanf(fichero1,"%f ", &leer.promg);//Lee coordenadas X de la grilla
                     promi = leer.promg;
                     PromBootstrap.push_back(promi);//adiciona este elemento al final del vecto
                        }
                  }

                }//fin_while
        }//fin_else
        fclose(fichero1);


//:::::(1-write):::::Imprime un fichero que contiene la media de las diferencias
        FILE *fichero2;
        FILE *outprom;
        outprom= fopen("outmediaBootstrap.dat","w");
        if((fichero2=fopen("myvelini.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
        else {
                while (!feof(fichero2)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  float velAire = 0.00;
                  int NoD = leer.nx * leer.nz;
                  for(int i=0; i<NoD; i++) {
                     fscanf(fichero2,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                        if (leer.veltmp == 333) {//Condicion para que quede con el mismo formato
                        fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                        }//end_if
                        else {
                        fprintf(outprom, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, PromBootstrap[i]);
                        }
                  }
                }//fin_while
        }//fin_else
        fclose(fichero2);
        fclose(outprom);

//:::::(2-read):::::Lee el fichero que contiene la desviacion estandar
      FILE *fichero3;
```

```
        if((fichero3=fopen("desstdBootstrap", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: desstdBootstrap\n\n");
        else {
                while (!feof(fichero3)) {
                //:::(1):::Lee informacion de la grilla.
                  fscanf(fichero3,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                //:::(2):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero3,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                    cxgi = leer.cxg;
                    Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                  }
                //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                  for(int x=0; x<leer.nx; x++) {
                    fscanf(fichero3,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                    ccxgi = leer.ccxg;
                    Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                  }
                //:::(4):::Lee informacion de la coordenadas x de la grilla
                  for(int z=0; z<leer.nz; z++) {
                    fscanf(fichero3,"%d", &leer.czg);//Lee coordenadas X de la grilla
                    czgi = leer.czg;
                    Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                  }
                //:::(5):::Lee informacion de la coordenadas x de la grilla
                  for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                          for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                          fscanf(fichero3,"%f ", &leer.promg);//Lee coordenadas X de la grilla
                     desi = leer.promg;
                          sumdes = sumdes + desi;
                     StdBootstrap.push_back(desi);//adiciona este elemento al final del vecto
                          }
                  }

                }//fin_while
        }//fin_else
        fclose(fichero3);
        float promdes = sumdes / (leer.nx * leer.nz);
        cout << "El promedio de la desviacion es: " << promdes << endl;

//:::::(2-write):::::Imprime un fichero que contiene la la desviacion estandar
        FILE *fichero4;
        FILE *outstd;
        outstd= fopen("outdesBootstrap.dat","w");
        if((fichero4=fopen("myvelini.dat", "r"))==NULL)
        printf("\n\nerror: fichero no encontrado: myvelini.dat\n\n");
        else {
                while (!feof(fichero4)) {
                //:::(2):::Lee informacion de la coordenadas x de la grilla
```

```
                    float velAire = 0.00;
                    int NoD = leer.nx * leer.nz;
                    for(int i=0; i<NoD; i++) {
                        fscanf(fichero4,"%d %f %f\n", &leer.coox, &leer.cooz, &leer.veltmp);//Lee coordenadas X
de la grilla
                                if (leer.veltmp == 333) {//Condicion para que quede con el mismo formato
                                fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, velAire);
                                }//end_if
                                else {
                                fprintf(outstd, "%d %-10.2f %-10.2f\n", leer.coox, leer.cooz, StdBootstrap[i]);
                                }
                    }
                   }//fin_while
          }//fin_else
          fclose(fichero4);
          fclose(outstd);

}
```

NOMBRE DEL FICHERO: (figbootstrap.sh)

```
#!/bin/csh -f

gmtset COLOR_BACKGROUND = 150/150/150

#Carlos Becerra
#Dic/2007
set ps = viewBootstrap.ps

#Inversion_Modelo:
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Solucion Modelo no perturbado":/a100:"z [m]":WeSn -X2
-Y18 -K -P > $ps

blockmean outModelbootstrap.dat -R0/10000/-900/17 -I200/200 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -GoutModelbootstrap.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -GoutModelbootstrap.grd -I10 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Crainbow -T500/4500/500 -Z >mypalet.cpt

grdimage outModelbootstrap.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outModelbootstrap.grd -JX -R -O -K >> $ps

psscale -Cmypalet.cpt -D17/5/10/0.2 -O -K >> $ps

#**********_________**********#
#Inversion_Bootstrap:
```

```
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Solucion para 1 muestra Bootstrap":/a100:"z [m]":WeSn
-X0 -Y-14 -K -P -O >> $ps

blockmean outSlnbootstrap.dat -R0/10000/-900/17 -I200/200 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -GoutSlnbootstrap.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -GoutSlnbootstrap.grd -I10 -R0/10000/-900/100 -V
#psclip outPoligono.dat -R -J -O -K >> $ps


grdimage outSlnbootstrap.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outSlnbootstrap.grd -JX -R -O >> $ps
#grdcontour outSlnbootstrap.grd -JX -R -C500 -L0/10 -O -K >> $ps

psscale -Cmypalet.cpt -D17/5/10/0.2 -O >> $ps
```

NOMBRE DEL FICHERO: (figbootstrapDiff.sh)

```
#!/bin/csh -f

gmtset COLOR_BACKGROUND = 150/150/150

#Carlos Becerra
#Dic/2007
set ps = viewBootstrapDiff.ps

#Diferencias:
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Diferencias perturnado vs. no-perturbado":/a100:"z
[m]":WeSn -X2 -Y18 -K -P > $ps

blockmean outDiffbootstrap.dat -R0/10000/-900/17 -I10/10 -V  > tmp.mybmean
#nearneighbor tmp.mybmean -GoutDiffbootstrap.grd -I10 -R0/10000/-900/17 -S1000 -V
surface tmp.mybmean -GoutDiffbootstrap.grd -I10 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Cjet -T0/400/50 -Z >mypalet.cpt

grdimage outDiffbootstrap.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outDiffbootstrap.grd -JX -R -O -K >> $ps

psscale -Cmypalet.cpt -D17/5/10/0.2 -O >> $ps
```

NOMBRE DEL FICHERO: (figbootstrapFIN.sh)

```
#!/bin/csh -f

gmtset COLOR_BACKGROUND = 150/150/150
```

```
#Carlos Becerra
#Dic/2007
set ps = viewBootstrapFIN.ps

#media Bootstrap:
psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] media Bootstrap [m/s]":/a100:"z [m]":WeSn -X2 -Y18 -K
-P > $ps

blockmean outmediaBootstrap.dat -R0/10000/-900/17 -I300/300 -V  > tmp.mybmean
surface tmp.mybmean -GoutmediaBootstrap.grd -I10 -R0/10000/-900/17 -V
#psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Crainbow -T600/4600/1000 -Z >mypalet.cpt

grdimage outmediaBootstrap.grd -JX -R0/10000/-900/17 -Cmypalet.cpt -O -K -B >> $ps
grdcontour outmediaBootstrap.grd -JX -R -O -K >> $ps

psscale -Cmypalet.cpt -D17/5/10/0.2 -O -K >> $ps

#**********_________**********#
#Desviacion Estandar:

psbasemap -JX16/-10 -R0/10000/-900/17 -Ba1000:"x[m] Desviacion Estandar [m/s]":/a100:"z [m]":WeSn -X0 -Y-14
-K -P -O >> $ps

blockmean outdesBootstrap.dat -R0/10000/-900/17 -I200/200 -V  > tmp.mybmean
surface tmp.mybmean -GoutdesBootstrap.grd -I10 -R0/10000/-900/17 -V
psclip outPoligono.dat -R -J -O -K >> $ps

makecpt -Cocean -T0/50/10 -Z -I> mypalet1.cpt

grdimage outdesBootstrap.grd -JX -R0/10000/-900/17 -Cmypalet1.cpt -O -K -B >> $ps
grdcontour outdesBootstrap.grd -JX -R -C500 -L0/10 -O -K>> $ps

psscale -Cmypalet1.cpt -D17/5/10/0.2 -O >> $ps
```

NOMBRE DEL FICHERO: (bootstrapping.sh)

```
#! /bin/sh
# File: bootstrapping.sh
# (Analisis Bootstrap)
# Carlos Becerra
# Diciembre/2007


# Nombre de fichero, input & output
set inicial = mymodelinicial.dat
```

```
# Numero de veces que se ejecuta el bucle i
nums=50

        echo "Iniciando proceso..."
    g++ tiempo.cpp -o exetime
        ./exetime

        echo "***Trazado de Rayos sobre modelo Completo***"
    #./tt_forward -Mmymodelinicial.dat -GoutCompleto.dat -Rmyrayout -Imyvelini.dat -V

        echo "***Invirtiendo set de datos Completo***"
    #./tt_inverse -Mmymodelinicial.dat -GoutCompleto.dat -V1 -Q1e-3 -SV2000 -SD100 -CVcorrefile.dat

        #cp inv_out.smesh.1.1 invtomoCom.dat
        #cp invtomoCom.dat inv_sln.tomoCom
        rm listBootstrap


        #Inversion de los i modelos Bootstrap
        i=1
    while [ "$i" -ne "$nums" ]
    do

        echo "Generando muestra bootstrap..."
        g++ bootstrapgen.cpp bootstrapgenTest.cpp -o muestra
        ./muestra

        echo "***Invirtiendo modelo $i bootstrap***"
        ./tt_inverse -Minv_sln.tomoCom -Gmodelbootstrap.dat -V1 -Q1e-3 -SV2000 -SD100 -CVcorrefile.dat
        cp inv_out.smesh.1.1 inv_sln.bootstrap.dat
        echo "leyendo grillas..."
        g++ bootstrapping.cpp bootstrappingTest.cpp -o exe
        ./exe

        #./figBootstrap.sh
        #gv viewBootstrap.ps
        #./figBootstrapDiff.sh
        #gv viewBootstrapDiff.ps

        #Copiar Solucion de la inversion para la muestra i
        cp inv_sln.bootstrap.dat slnBootstrap$i.dat
        echo "slnBootstrap$i.dat"  >> listBootstrap
        echo "Bootstrap $i listo!"

        i=`expr $i + 1`
        done

        echo "Calculando media Bootstrap"
        ./stat_smesh -LlistBootstrap -Ca > mediaBootstrap
```

```
        echo "Calculando Desviacion Estandar Bootstrap"
        ./stat_smesh -LlistBootstrap -CrmediaBootstrap > desstdBootstrap

        echo "Generando vista analisis Bootstrap"
        g++ bootstrapFIN.cpp bootstrappingTest.cpp -o exe
        ./exe
        ./figBootstrapFIN.sh
        gv viewBootstrapFIN.ps


    g++ tiempo.cpp -o exetime
        ./exetime
        echo "Proceso Terminado"
```

ANEXO 6

Código para calcular correcciones estáticas

NOMBRE DEL FICHERO: (analisisTest.cpp)

```
// analisisTest.cpp-programa que utiliza la clase analisis
#include<iostream>
#include "analisis.h"

using namespace std;

int main()
{
        analisis a(51); //NUmero de modelos q va a generar
        a.leerEntrada();
        a.genInputGrilla(10, 10);//size cell en x, profundidad del modelo, size cell(en z), Velocidad maxima,
Velocidad minima.
}
```

NOMBRE DEL FICHERO: (analisis.h)

```
//definicion de la clase Analisis
#if !defined( _ANALISIS_H_ )
#define _ANALISIS_H_

class analisis
```

```
{
        //atributos
        private:
                int Numol; //NUmero de Modelos
    protected:
                void mensage1();
        //miembros publicos
        public:
        analisis() {}
        analisis(int numol);
        double leerEntrada();
        double genInputGrilla(int dx, int dz);
        double genttfile();
        //double jackknifing(int shift, int); //shift: Numero de trazas que se desean sacar de cada disparo
        int shot;
        int cont[800]; //Contador de: Numero de receptores por disparo
        float coorXmin;
        float coorXref;
        float coorYref;
        float Cotamax, Cotamin, Cotadiff;
};
#endif // _ANALISIS_H_
```

NOMBRE DEL FICHERO: (analisis.cpp)

```
//analisis.cpp
//Write by: CarlosBecerra
//Date: Jan/2007
#include<iostream>
#include "analisis.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        char id[5];
        int estacion;
        int cooX;
        int cooY;
        float cota;
        float tpa; //Tiempo de Primer Arribo
        float tup; //Tiempo de uphole, solo para SHOT's
```

```
}lectura ;

void analisis::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }

analisis::analisis(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double analisis::leerEntrada() { //Lee un fichero secuencial
  FILE *fichero;
  lectura leer;
  int trace =0;
  shot =0; //Iniciando contador
  int sumpick =0; //Contador NUmero Total de picks
  coorXmin= 100000000000.00;
  vector<float> COTAS;//Contiene Cotas del set de datos

        if((fichero=fopen("clientes2.geo", "r"))==NULL)
        printf("\n\nCaution1: don't read file: Look input file name\n\n");
        else {

      while (!feof(fichero)) {

                fscanf(fichero,"%s",&leer.id);
                int resu = strncmp(leer.id, "SHOT", 4); //Compara 2 cadenas de caracteres
                fseek(fichero, -strlen(leer.id)*sizeof(char), SEEK_CUR); //devuelve el puntero al inicio del
fichero
                if (resu != 0) { //lee y imprime RECEIVER
                fscanf(fichero,"%s %d %d %d %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa);
                trace ++;
                COTAS.push_back(leer.cota);

                        if(leer.cooX < coorXmin) { //Encontrar coordenadas de punto de referencia, En los
shot's
                        coorXmin = leer.cooX;
                        coorXref = coorXmin;
                        coorYref = leer.cooY;
                        }
                }
                if (resu == 0) { //lee y imprime SHOT
                fscanf(fichero,"%s %d %d %d %f %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa, &leer.tup);
```

```
                cont[shot] = trace;
                shot ++;
                trace =0;
                COTAS.push_back(leer.cota);

                        if(leer.cooX < coorXmin) { //Encontrar coordenadas de punto de referencia, En los
receptores's
                        coorXmin = leer.cooX;
                        coorXref = coorXmin;
                        coorYref = leer.cooY;
                        }
                }
        }
        cont[shot]=trace;
            }
        fclose(fichero);

                cout << "\n\n********************** info 1 **********************" << endl;

                cout << "NUmero de receptores por disparo: " << endl;
                for(int i=1; i<=shot; i++) { //Imprime Informacion leida del fichero
                printf("%d ", cont[i]); //Imprime Numero de trazas por disparo.
                sumpick = sumpick + cont[i];
                }
                cout << "\n\n********************** info 2 **********************" << endl;
                cout << "El set de Datos leido contiene " << shot << " disparos, " << sumpick << " picks" <<
endl;
                cout << "Las coordenadas (x,y) del punto de referencia son: ";
                printf( "%10.2f %10.2f\n\n", coorXref ,coorYref);

        //Vector de Cotas en orden Ascendente --> Para encontrar Cotas maxima & minima
        int Numecotas = COTAS.size();//tamaño del vector
        float tempO;
        for(int i=0; i<(Numecotas-1); i++) {
          for(int j=0; j<(Numecotas-1); j++) {//Ordenar vector Cotas
            if(COTAS[j] > COTAS[j+1]) {
            tempO=COTAS[j];
            COTAS[j]=COTAS[j+1];
            COTAS[j+1]=tempO;
           }
          }
        }
        Cotamax = COTAS[Numecotas-1];
        Cotamin = COTAS[0];
        Cotadiff = Cotamax - Cotamin;

        cout << "\n********************** info 3 **********************" << endl;
        printf("La Cota maxima es: %8.2f y la minima es %8.2f la diferencia es %8f\n", Cotamax, Cotamin,
Cotadiff);
```

```
        cout << "*********************  info  *********************\n" << endl;


}

double analisis::genInputGrilla(int dx, int dz) {
  FILE *fichero;
  lectura leer;
  int tempEst;
  float COORx, tempCoordX, tempCOTASREC, cotatmp, negcota;
  vector<int> RecEst;//Contiene Estacion de Receptores
  vector<int> SouEst;//Contiene Estacion de Fuentes
  vector<float> RecCoordX;//Contiene Coordenada X de Receptores
  vector<float> SouCoordX;//Contiene Coordenada X de Fuentes
  vector<int> RecEstacion;//Contiene Estacion de Receptores
  vector<float> RecCoordenadaX;//Contiene Coordenada X de Receptores
  vector<float> COTASSOU;//Contiene Cotas de las fuentes
  vector<float> COTASREC;//Contiene Cotas de los receptores
  vector<float> CotasReceptores;//Contiene Cotas de los receptores

        FILE *outSources;
        outSources= fopen("OUTSOU.dat","w");
        fprintf(outSources,"%d\n", shot);

        if((fichero=fopen("clientes2.geo", "r"))==NULL)
        printf("\n\ncaution1: don't read file, Revisar el nombre del fichero de entrada!\n\n");
        else {
          while (!feof(fichero)) {

                for(int i=1; i<=shot; i++) { //lee e imprime informacion de 1 disparo
              fscanf(fichero,"%s %d %d %d %f %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa, &leer.tup);
                fprintf(outSources, "%d ", leer.estacion);
                SouEst.push_back(leer.estacion);
           COORx = sqrt(pow(coorXref-(leer.cooX),2)+pow(coorYref-(leer.cooY),2));
           fprintf(outSources,"%f ", COORx);
                SouCoordX.push_back(COORx);
                cotatmp = -(Cotamax + Cotamin - leer.cota);
                COTASSOU.push_back(cotatmp);
           fprintf(outSources,"%f\n", cotatmp);

                  for(int j=1; j<=cont[i]; j++) { //lee e imprime informacion de los receptores
                  fscanf(fichero,"%s %d %d %d %f %f\n",&leer.id, &leer.estacion, &leer.cooX, &leer.cooY,
&leer.cota, &leer.tpa);
                  RecEst.push_back(leer.estacion);
              COORx = sqrt(pow(coorXref-(leer.cooX),2)+pow(coorYref-(leer.cooY),2));
                  RecCoordX.push_back(COORx);
                  cotatmp = Cotamax + Cotamin - leer.cota;
                  COTASREC.push_back(cotatmp);
```

```
                    }//fin for

                        }//fin for
                  }//fin while
                }//fin else
  fclose(fichero);
  fclose(outSources);


            int NumElem = RecEst.size();//tamaño del vector
            for(int i=0; i<(NumElem-1); i++) {
               for(int j=0; j<(NumElem-1); j++) {//Ordenar vector de Estaciones con su correspondiente: (1) Coordenada X y (2) COTA
                if(RecEst[j] > RecEst[j+1]) {
                tempEst=RecEst[j];
                tempCoordX=RecCoordX[j];
                tempCOTASREC=COTASREC[j];
                RecEst[j]=RecEst[j+1];
                RecCoordX[j]=RecCoordX[j+1];
                COTASREC[j]=COTASREC[j+1];
                RecEst[j+1]=tempEst;
                RecCoordX[j+1]=tempCoordX;
                COTASREC[j+1]=tempCOTASREC;
                }
               }
            }

            //Crear Nuevos vectores que no contengan estaciones de receptores con coordenadas_X repetidas
            for(int k=0; k<NumElem; k++) {//No incluir Repetidos en vectores
              if(RecEst[k] != RecEst[k+1]) {
              RecEstacion.push_back(RecEst[k]);
              RecCoordenadaX.push_back(RecCoordX[k]);
              CotasReceptores.push_back(COTASREC[k]);
              }
            }

            int tam1 = RecEstacion.size();
            FILE *outReceptores;
            outReceptores= fopen("OUTREC.dat","w");
            fprintf(outReceptores, "%d\n", tam1);
            for(int r=0; r<tam1; r++) {//No incluir Repetidos en vectores
               negcota = -1 * CotasReceptores[r];
              fprintf(outReceptores, "%d %f %f\n", RecEstacion[r], RecCoordenadaX[r], negcota);
            }
      fclose(outReceptores);

}
```

NOMBRE DEL FICHERO: (delayTest.cpp)

```
// delayTest.cpp-programa
/* Lee: (1) modelo de velocidad (2) Informacion Fuentes (3) Informacion Receptores*/
/* Calcula: Correcion por Estrato Somero*/
#include<iostream>
#include "delay.h"

using namespace std;


int main()
{
        delay a(1);
        a.correcion(3000.00, 10, 10, 679.70, 109.55, 500.0, 3200.00);
}
```

NOMBRE DEL FICHERO: (delay.h)

```
//definicion de la clase delay
#if !defined( _DELAY_H_ )
#define _DELAY_H_

class delay
{
        //atributos
        private:
                int Numol; //NUmero de Modelos
    protected:
                void mensage1();
        //miembros publicos
        public:
        delay() {}
        delay(int numol);
        double correcion(float lim, int dx, int dz, float Cotamax, float Cotamin, float datum, float VelRemp);
};
#endif // _DELAY_H_
```

NOMBRE DEL FICHERO: (delay.cpp)

```
//delay.cpp
//Write by: CarlosBecerra
//Date: Jan/2008
#include<iostream>
#include "delay.h"
#include <cstdio>
#include <cstring>
#include<cstdlib>
```

```
#include<iostream>
#include<ctime>
#include<cmath>
#include<vector>

using namespace std;

typedef struct {
        int nx, nz, Vwater, Vair; //No. de nodos de la grilla y velocidades en el agua y en al aire
        int numRec, numSou;
        int estacion;
        float coorX;
        int cxg, czg; //Coordenadas X y Z de la Grilla
        int coox;
        float ccxg; //Cota de la coordenada X de la grilla
        float velg, vela;
        float cooz, veltmp;
        float diffeacm;
        float cotasou, cotarec;
}lectura ;

void delay::mensage1() {
        cout << "Parametro No Valido, convirtiendo a Positivo\n";
        }

delay::delay(int numol) {
        if (numol < 0) {
                mensage1();
                numol= -numol;
                }
                Numol= numol;
}

double delay::correcion(float lim, int dx, int dz, float Cotamax, float Cotamin, float datum, float VelRemp) { //Lee el
fichero que contiene la solucion del set completo
  FILE *fichero;
  lectura leer;
  vector<int> Vcxg;//Vector que contiene Coordenadas X de la Grilla
  vector<int> Vcxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vccxg1;//Vector que contiene Coordenadas X de la Grilla
  vector<float> Vczg;//Vector que contiene Coordenadas Z de la Grilla
  vector<float> Vczg1;//Vector que contiene Coordenadas Z de la Grilla
  vector< vector<float> > ModeloVel;//Matrix que contiene las velodades del modelo
  vector<int> VestSou;//Vector que contiene estaciones de fuentes
  vector<int> VestRec;//Vector que contiene estaciones de los receptores
  vector<float> VcoorXSou;//Vector que contiene Coordenadas X de las fuentes
  vector<float> VcoorXRec;//Vector que contiene Coordenadas X de los receptores
  vector<float> COTASOU;//Vector que contiene Cotas de las fuentes
```

```
  vector<float> COTAREC;//Vector que contiene Cotas de las receptores
  int cxgi, esti;
  float ccxgi, coorXi, time;
  int czgi, a;
  float velmodi, veli;
  char namefile[10];
  char namefile2[10];
  float distW, d1;
  float statics;
  datum = -(Cotamax + Cotamin - datum);

//:::::(1-read):::::Lee el fichero que contiene el Modelo de Velocidades
      FILE *fichero1;
            if((fichero1=fopen("prommontecarlo", "r"))==NULL)
            printf("\n\nerror: fichero no encontrado: modeldelay.dat\n\n");
            else {
                        while (!feof(fichero1)) {
                        //:::(1):::Lee informacion de la grilla.
                          fscanf(fichero1,"%d %d %d %d\n",&leer.nx, &leer.nz, &leer.Vwater, &leer.Vair);
                                ModeloVel.reserve(leer.nx);
                        //:::(2):::Lee informacion de la coordenadas x de la grilla
                          for(int x=0; x<leer.nx; x++) {
                            fscanf(fichero1,"%d", &leer.cxg);//Lee coordenadas X de la grilla
                            cxgi = leer.cxg;
                            Vcxg1.push_back(cxgi);//adiciona este elemento al final del vector
                          }
                        //:::(3):::Lee informacion de la Cota para cada coordenada x de la grilla
                          for(int x=0; x<leer.nx; x++) {
                            fscanf(fichero1,"%f\n", &leer.ccxg);//Lee cota para cada coordenadas X de la grilla
                            ccxgi = leer.ccxg;
                            Vccxg1.push_back(ccxgi);//adiciona este elemento al final del vector
                          }
                        //:::(4):::Lee informacion de la coordenadas x de la grilla
                          for(int z=0; z<leer.nz; z++) {
                            fscanf(fichero1,"%d", &leer.czg);//Lee coordenadas X de la grilla
                            czgi = leer.czg;
                            Vczg1.push_back(czgi);//adiciona este elemento al final del vector
                          }
                        //:::(5):::Lee informacion de la coordenadas x de la grilla
                          for(int x=0; x<leer.nx; x++) {//Una iteracion para cada nodo X de la grilla
                                ModeloVel[x].reserve(leer.nz);
                                for(int z=0; z<leer.nz; z++) {//Una iteracion para cada nodo Z de la grilla
                                fscanf(fichero1,"%f ", &leer.velg);//Lee coordenadas X de la grilla
                             velmodi = leer.velg;
                             ModeloVel[x].push_back(velmodi);//adiciona este elemento al final del vector
                                }

                          }
```

```
                }//fin_while
            }//fin_else
            fclose(fichero1);


//:::::(2-read):::::Lee "OUTSOU.dat" Informacion de fuentes
        FILE *fichero2;
            fichero2 = fopen("OUTSOU.dat", "r");
            fscanf(fichero2,"%d", &leer.numSou);
            for(int s=0; s<leer.numSou; s++) {
                fscanf(fichero2,"%d %f %f", &leer.estacion, &leer.coorX, &leer.cotasou); //Lee coordenadas X de la
grilla
                esti = leer.estacion;
                coorXi = leer.coorX;
                VestSou.push_back(esti);
                VcoorXSou.push_back(coorXi);
                COTASOU.push_back(leer.cotasou);
            }
            fclose(fichero2);
//:::::(2-calculo Correcion):::::Calculo de correccion en fuentes
        FILE *out1;
            out1 = fopen("outSOUcorrection.dat", "w");
            float V0, V1;
            for(int s=0; s<leer.numSou; s++) {//Una iteracion para cada fuente
            float TiempoA = 0.0;
            float TiempoB = 0.0;
            float Tiempo = 0.0;
                    for(int x=0; x<leer.nx; x++) {//Una iteracion para cada coordenada X
                        if (VcoorXSou[s] > Vcxg1[x]) { }

                        else {
                        a = x;
                    cout << "a " << a << " (" << Vcxg1[a-1] << " < " << VcoorXSou[s] << " < " << Vcxg1[a]
<< ") " << endl;

                                for(int z=0; z<leer.nz; z++) {//Una iteracion para cada coordenada Z
                                    if (z==0) { V0 = ModeloVel[a-1][z];}
                                    else {
                                if (ModeloVel[a-1][z] < lim) {
                                    V1 = ModeloVel[a-1][z];
                                time = (dz /((V0+V1)/2)) * 1000;
                                TiempoA = TiempoA + time;
                             cout << z << ") : (" << V0 << ", " << V1 << ") time: " << time << " Ttotal: " <<
TiempoA << endl;
                                    V0 = V1;
                                    distW = distW + dz; }
                                    else {
                                    V1 = ModeloVel[a-1][z];
                                time = (((dz*(lim-V0))/(V1-V0))/((V0+V1)/2)) * 1000;
```

```
                                TiempoA = TiempoA + time;
                            cout << z << "m) : (" << V0 << ", " << lim << ") time: " << time << " Ttotal: "
<< TiempoA << endl;
                                    z = (leer.nz-1);
                                    distW = distW + ((dz*(lim-V0))/(V1-V0)); }
                                   }
                            }//fin_for:z
                            for(int z=0; z<leer.nz; z++) {//Una iteracion para cada coordenada Z
                                if (z==0) { V0 = ModeloVel[a][z];}
                                else {
                                if (ModeloVel[a][z] < lim) {
                                    V1 = ModeloVel[a][z];
                                time = (dz /((V0+V1)/2)) * 1000;
                                TiempoB = TiempoB + time;
                            cout << z << ") : (" << V0 << ", " << V1 << ") time: " << time << " Ttotal: " <<
TiempoB << endl;
                                    V0 = V1;
                                    distW = distW + dz; }
                                    else {
                                    V1 = ModeloVel[a][z];
                                time = (((dz*(lim-V0))/(V1-V0))/((V0+V1)/2)) * 1000;
                                TiempoB = TiempoB + time;
                            cout << z << "m) : (" << V0 << ", " << lim << ") time: " << time << " Ttotal: "
<< TiempoB << endl;
                                    z = (leer.nz-1);
                                    distW = distW + ((dz*(lim-V0))/(V1-V0)); }
                                   }
                            }//fin_for:z
                        float inter = (abs(TiempoB-TiempoA)*(VcoorXSou[s]-Vcxg1[a-1]))/(Vcxg1[a]-Vcxg1[a-1]);
                        if (TiempoB<TiempoA) {
                        Tiempo = TiempoB + inter; }
                        else { Tiempo = TiempoA + inter; }
                    cout << "x=" << Vcxg1[a-1] << " TA= " << TiempoA << endl;
                    cout << "x=" << VcoorXSou[s] << " estacion: " << VestSou[s] <<" T= " << Tiempo << endl;
                    cout << "x=" << Vcxg1[a] << " TB= " << TiempoB << endl;
                        d1 = COTASOU[s] + (-1*distW);
                        if (d1 > datum) {
                        statics = -Tiempo - ((abs(datum-d1)*1000)/VelRemp);
                        //statics = -Tiempo;
                        //statics = -((abs(abs(datum)-abs(d1))*1000)/VelRemp);
                        fprintf(out1, "%d %f\n", VestSou[s], statics); }
                            else {
                        statics = -Tiempo + ((abs(d1-datum)*1000)/VelRemp);
                        //statics = -Tiempo;
                        //statics = +((abs(abs(d1)-abs(datum))*1000)/VelRemp);
                        fprintf(out1, "%d %f\n", VestSou[s], statics); }
                    distW = 0.0;
                    x = leer.nx-1;
                    }
```

```
                }//fin_for_coordenada X
            }//fin_for_fuente
            fclose(out1);


//:::::(3-read):::::Lee "OUTREC.dat" Informacion de fuentes
        FILE *fichero3;
            fichero3 = fopen("OUTREC.dat", "r");
            fscanf(fichero3,"%d", &leer.numRec);
            for(int r=0; r<leer.numRec; r++) {
                fscanf(fichero3,"%d %f %f", &leer.estacion, &leer.coorX, &leer.cotarec); //Lee coordenadas X de la
grilla
                esti = leer.estacion;
                coorXi = leer.coorX;
                VestRec.push_back(esti);
                VcoorXRec.push_back(coorXi);
                COTAREC.push_back(leer.cotarec);
            }
            fclose(fichero3);
//:::::(3-calculo Correcion):::::Calculo de correccion en receptores
        FILE *out2;
            out2 = fopen("outRECcorrection.dat", "w");
            //float V0, V1;
            for(int s=0; s<leer.numRec; s++) {//Una iteracion para cada fuente
            float TiempoA = 0.0;
            float TiempoB = 0.0;
            float Tiempo = 0.0;
                    for(int x=0; x<leer.nx; x++) {//Una iteracion para cada coordenada X
                        if (VcoorXRec[s] > Vcxg1[x]) { }

                        else {
                        a = x;
                    cout << "a " << a << " (" << Vcxg1[a-1] << " < " << VcoorXRec[s] << " < " << Vcxg1[a]
<< ") " << endl;

                                for(int z=0; z<leer.nz; z++) {//Una iteracion para cada coordenada Z
                                    if (z==0) { V0 = ModeloVel[a-1][z];}
                                    else {
                                if (ModeloVel[a-1][z] < lim) {
                                    V1 = ModeloVel[a-1][z];
                                time = (dz /((V0+V1)/2)) * 1000;
                                TiempoA = TiempoA + time;
                             cout << z << ") : (" << V0 << ", " << V1 << ") time: " << time << " Ttotal: " <<
TiempoA << endl;
                                    V0 = V1;
                                    distW = distW + dz; }
                                    else {
                                    V1 = ModeloVel[a-1][z];
```

```
                                time = (((dz*(lim-V0))/(V1-V0))/((V0+V1)/2)) * 1000;
                                TiempoA = TiempoA + time;
                             cout << z << "m) : (" << V0 << ", " << lim << ") time: " << time << " Ttotal: "
<< TiempoA << endl;
                                   z = (leer.nz-1);
                                   distW = distW + ((dz*(lim-V0))/(V1-V0)); }
                                  }
                           }//fin_for:z
                           for(int z=0; z<leer.nz; z++) {//Una iteracion para cada coordenada Z
                                if (z==0) { V0 = ModeloVel[a][z];}
                                else {
                                if (ModeloVel[a][z] < lim) {
                                   V1 = ModeloVel[a][z];
                                time = (dz /((V0+V1)/2)) * 1000;
                                TiempoB = TiempoB + time;
                             cout << z << ") : (" << V0 << ", " << V1 << ") time: " << time << " Ttotal: " <<
TiempoB << endl;
                                   V0 = V1;
                                   distW = distW + dz; }
                                   else {
                                   V1 = ModeloVel[a][z];
                                time = (((dz*(lim-V0))/(V1-V0))/((V0+V1)/2)) * 1000;
                                TiempoB = TiempoB + time;
                             cout << z << "m) : (" << V0 << ", " << lim << ") time: " << time << " Ttotal: "
<< TiempoB << endl;
                                   z = (leer.nz-1);
                                   distW = distW + ((dz*(lim-V0))/(V1-V0)); }
                                  }
                           }//fin_for:z
                      float inter = (abs(TiempoB-TiempoA)*(VcoorXRec[s]-Vcxg1[a-1]))/(Vcxg1[a]-Vcxg1[a-1]);
                      if (TiempoB<TiempoA) {
                      Tiempo = TiempoB + inter; }
                      else { Tiempo = TiempoA + inter; }
                cout << "x=" << Vcxg1[a-1] << " TA= " << TiempoA << endl;
                cout << "x=" << VcoorXRec[s] << " estacion: " << VestRec[s] <<" T= " << Tiempo << endl;
                cout << "x=" << Vcxg1[a] << " TB= " << TiempoB << endl;
                      cout << "Datum: " << datum << "statics: " << statics << "distW: " << distW << endl;
                           d1 = COTAREC[s] + (-1*distW);
                        if (d1 > datum) {
                        statics = -Tiempo - ((abs(datum-d1)*1000)/VelRemp);
                        //statics = -Tiempo;
                        //statics = -((abs(abs(datum)-abs(d1))*1000)/VelRemp);
                        fprintf(out2, "%d %f\n", VestRec[s], statics); }
                           else {
                        statics = -Tiempo + ((abs(d1-datum)*1000)/VelRemp);
                        //statics = -Tiempo;
                        //statics = +((abs(abs(d1)-abs(datum))*1000)/VelRemp);
                        fprintf(out2, "%d %f\n", VestRec[s], statics); }
                      distW = 0.0;
```

```
                    x = leer.nx-1;
                    }

                }//fin_for_coordenada X
        }//fin_for_fuente
        fclose(out2);

}
```

NOMBRE DEL FICHERO: (correccion.sh)

```
#! /bin/sh
# File: correcciones.sh
# Calculo de correcciones por estrato somero
# Carlos Becerra
# Enero/2008


        echo "Generando informacion de ubicacion de fuentes y receptores"
        g++ analisis.cpp  analisisTest.cpp -o exe
        ./exe

        echo "Generando correcciones estaticas"
        g++ delay.cpp delayTest.cpp -o exe
        ./exe
```