INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
A new approach to slant stack processing: An X-window (OSF/motif) project

Ni, Lideng, M.A.
Rice University, 1992
RICE UNIVERSITY

A NEW APPROACH TO SLANT STACK PROCESSING
- An X-window (OSF/Motif) Project

by

LIDENG NI

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE
MASTER OF ARTS

APPROVED, THESIS COMMITTEE

Gerald H.F. Gardner, Professor, Director
Department of Geology & Geophysics

Dale S. Sawyer, Associate Professor
Department of Geology & Geophysics

J.C. De Bremaecker, Professor
Department of Geology & Geophysics

Houston, Texas

April, 1992
ABSTRACT

A New Approach to Slant Stack Processing
- An X-window (OSF/Motif) Project

by

Lideng Ni

A new definition for slant stack and inversion has been formulated. The different p ranges with different frequencies in the forward slant stack provide accurate reconstruction of the original data. The computing time is faster than the conventional method for the same p traces made in the forward transform. Its applications presenting in this thesis have been satisfactory in seismic data processing.
ACKNOWLEDGEMENTS

I would like to express my sincere thank to my department for the two year's financial support. I would not have this thesis and my career had I not had this support.

My thank to my advisor, Dr. Gerald H. F. Gardner, can not be simply expressed in any kind of language. He changed my life. He is so wonderful and so considerate. He helped me not just on my studies, but many others. I owe him too much.

He is full of knowledge. I can never graduate under him because he has so much to learn from.

Should I be a Christian, he would have been my God. I am so proud to be his student and to know such a wonderful person.

Yes, no language can described true gratefulness. But I still want to take this chance to say to him, I love you, more than a son to a wonderful father. He will be in my memory wherever I go. This thesis is truly dedicated to him.

I also thank Dr. Dale S. Sawyer for his valuable suggestions and crises in this study. He understands my topic very well and this thesis would have been in another way if I had not had his suggestions.

I also thank Dr. De Bremaecker for his serving in my thesis committee and spending time reading my script.
To Dr. Gerald H. F. Gardner

My Advisor
TABLE OF CONTENTS

ABSTRACT .................................................................................................................. ii
ACKNOWLEDGEMENT .............................................................................................. iii
TABLE OF CONTENTS .............................................................................................. v
LIST OF FIGURES ................................................................................................... vii
INTRODUCTION ........................................................................................................ 1

CHAPTER 1  SLANT STACK AND ITS INVERSION ............................................. 2
  1.1 Slant Stack ........................................................................................................ 2
  1.2 Inversion of a Slant Stack ................................................................................... 6
  1.3 Slant-stack Inversion by a Second Slant Stack
      with an "Rho" Filter Is an Exact Inverse of the
      Forward Transform ......................................................................................... 9
  1.4 Relation between Slant Stack and 2-D Fourier
      transform .......................................................................................................... 12

CHAPTER 2  SCHEMES TO CONSTRUCT SLANT STACK
      AND ITS INVERSION ....................................................................................... 15

Introduction ............................................................................................................. 15
  2.1 Slant Stack and its Inversion by direct mapping
      in time domain .................................................................................................. 15
  2.2 Slant Stack and its Inversion by phase shift in
      the frequency domain ....................................................................................... 17
  2.3 Slant stack and Inversion by (F, K)
      Implementation .................................................................................................. 22

CHAPTER 3  PROBLEMS WITH CONVENTIONAL
      SLANT STACK ............................................................................................... 24
Introduction.................................................................24

3.1 Problems with Direct Mapping.................................25
3.2 Problems by Phase Shift...........................................30
3.3 Problems with Conventional (F, K) Implementation.........32

CHAPTER 4 THE NEW APPROACH TO SLANT STACK ..............34
4.1 Introduction .........................................................34
4.2 Method ................................................................37
  4.2.1 Forward Slant Stack ............................................37
  4.2.2 Inverse Slant Stack .............................................40

CHAPTER 5 EXAMPLES.....................................................42
5.1 Model 1, Several Sinusoids ........................................43
5.2 Model 2, Spike Response ..........................................47
5.3 Model 3, Impulse Response ......................................53
5.4 Model 4, Input Is Single Frequency Sinusoid
   in All Traces..........................................................59
5.5 Model 5, Input Data Have Events with Negative
   Slopes ................................................................63

CHAPTER 6 APPLICATIONS...............................................67
6.1 Muting Undesired Events .........................................67
  6.1.1 Removal of a Linear Event .................................67
  6.1.2 Removal Strong Linear Event ..............................74
6.2 Trace Interpolation ..................................................81
6.3 Velocity Analysis .....................................................85

CHAPTER 7 SUMMARY.....................................................86

APENDIX I THE USER GUIDE...........................................87

APENDIX II THE COMPLETE SOURCE CODE (MOTIF
INTRODUCTION

Slant stack usually refers to a discrete, finite form of the Radon transform. It involves linear moveout and summing amplitudes over an offset axis in the time domain. By slant stack, the original offset axis (x-axis) is replaced by a ray parameter p axis and the time axis is replaced by a new time (intercept time) τ axis. Thus the transformed domain by slant stack is sometimes called the tau-p domain. A family of traces in the tau-p domain is called a slant-stack gather or p gather (Yilmaz, 1987).

Slant stack and its inversion have applications in seismic data processing such as velocity filtering, trace interpolation, noise suppression, migration etc.. Both forward and inverse slant stacks can be carried out by several different methods.

A slant stack is usually constructed from a minimum slope $p_{\text{min}}$ to a maximum slope $p_{\text{max}}$, which are chosen to include the main features of the data. In this method, the range of p values is the same for every frequency in the data. A primary objective of this thesis is to formulate a new definition of slant stack in which each frequency has a different range of p values. The slant stack constructed by this modified procedure can be inverted in a simple way to recover the original data. The properties of this new transform and its applications to the problems of data interpolation and data enhancement are also discussed in this thesis.
Chapter 1
Slant Stack And Its Inversion

[1.1] Slant Stack

1. Definition

The most straight forward description for slant stack in seismology is to sum all the amplitude in the (x, t) domain along a slanting line described by its slope $p_i$ and intercept time $\tau_i$, where $x$ represents offset axis and $t$ represents time axis in a seismic section. The result is plotted at a point with coordinates $(p_i, \tau_i)$ in the $\tau$-$p$ domain. The $x$ axis is replaced by the $p$ axis, the transformed ray parameter axis, and the $t$ axis is replaced by the $\tau$ axis, the intercept time axis (Figure 1.1.1).

![Diagram](image-url)

Figure 1.1.1 Slant stack is a process summing amplitudes along a slanting line with slope $p_i$ and intercept time $\tau_i$ in (x, t) domain (a). The result is plotted at the point $(p_i, \tau_i)$ in the corresponding ($\tau$, $p$) domain (b).

In most geophysical applications, the process is a digitization of a line integral over the $x$ axis. That is:
\[ d(p, \tau) = \int_{-\infty}^{\infty} f(x, t=\tau+px) \, dx , \]  

where \( f(x, t) \) is the amplitude at time \( t \) and horizontal range \( x \). The function \( d(p, \tau) \) is the mathematical slant stack of \( f(x, t) \).

2. Slant stack by phase shift

Slant stack equation (1.1.1) can be carried out in the following manner: First shift the amplitudes in each column of the \((x, t)\) domain vertically so that a slanting line with slope \( p \) becomes horizontal and then add all the columns together. The shift for a column at range \( x \) is \( \Delta t = px \). The result gives column \( p \) in the \((p, \tau)\) domain (Figure 1.1.2).

Figure 1.1.2 Implementation of slant stack by shifting the amplitudes on the slanting line \( AB \) to the horizontal line \( AC \) and adding these amplitudes horizontally
The shifting is accomplished by convolution in time domain, which is equivalent to multiplication in the frequency domain.

To shift any column vertically by $\Delta t$, its 1-D FFT coefficients $G(x, F_i)$ are multiplied by $e^{-i2\pi F_i \Delta t}$, where $F_i$ are the frequencies in Hertz. Hence, summing over $x$, the Fourier coefficients $D(p, F_i)$ of the slant stack are given by:

$$D(p, F_i) = \sum_x G(x, F_i) e^{-i2\pi F_i p x} \quad (1.1.2)$$

The final slant stack $d(p, \tau)$ is obtained by applying 1-D inverse FFT to each column of the matrix $D(p, F_i)$.

![Diagram of slant stack construction](image)

**Figure 1.1.3** Scheme to construct slant stack by phase shift. $(x, F)$ data are obtained by 1-D FFT along each column of the $(x, t)$ data. $(p, F)$ data are obtained by phase shift, which is accomplished by matrix multiplication. $(p, \tau)$ data are obtained by 1-D IFFT along each column of the $(p, F)$ data.

To simplify for digitized data:
\[
f(x, t) \xrightarrow{1-D \text{ FFT}} G(x, F) \xrightarrow{\text{phase shift and summation}} D(p, F) \xrightarrow{1-D \text{ FFT}} d(p, \tau)
\]

To actually carry out this process for the mathematical case, using integration instead of summation, we can separate it into the following steps:

1. Forward Fourier transform:

   This process transforms data from \((x, t)\) domain into \((x, F)\) domain by the equation:

   \[
   G(x, F) = \int_{-\infty}^{\infty} f(x, t)e^{i2\pi Ft}dt
   \]

   where \(G(x, F)\) is the Fourier transform along each column of the original data \(f(x, t)\).

2. Phase shifting and summing horizontally:

   The amplitudes along the slanting line with slope \(p\) are shifted to become horizontal. Their summation defines the data in \((p, F)\) domain. It is carried out by the equation:

   \[
   D(p, F) = \int_{-\infty}^{\infty} G(x, F)e^{-i2\pi Fpx}dx
   \]

   where \(D(p, F)\) is the slant stack in frequency domain.

3. Inverse Fourier transform

   Apply the inverse Fourier transform on the data in \((p, F)\) domain, we obtain the \(\tau\)-\(p\) gather:
where \( d(p, \tau) \) is the slant stack of \( f(x, t) \).

These three steps provide the theoretical framework for performing slant stack in the \((x, F)\) domain.

[1.2] Inversion of a Slant Stack

1. Definition

Slant stack inversion for the mathematical case can also be done by a similar process: adding all the amplitudes in the \((\tau, p)\) domain along a slanting line described by its slope \(-x_i\) and intercept time \(t_i\) after convolving the columns of the \((\tau, p)\) domain with a "Rho" filter (each Fourier component of frequency \(F\) multiplied by its frequency \(F\), discussed below), The result is directly plotted at a point with coordinates \((x_i, t_i)\) in the \((x, t)\) domain (Figure 1.2.1). This process is also called a second slant stack, in which a "Rho" filter is also applied.

![Figure 1.2.1 Slant stack inversion using a second slant stack. The process is summing amplitudes along slanting line with slope \(-x_i\) and intercept time \(t_i\) in the \((\tau, p)\) domain (a) along with a "Rho" filter. The result is a point plotted on the corresponding \((x, t)\) domain (b).](image-url)
In agreement with equation (1.1.1), this process can also be expressed as line integral over the p axis:

\[ f(x, t) = \int_{-\infty}^{\infty} d'(p, \tau=t-px) \, dp \]  

(1.2.1)

where \( d'(p, \tau) \) is equal to \( d(p, \tau) \) filtered by the "Rho" filter. That is:

\[ d'(p, \tau) = \rho(\tau) \ast d(p, \tau) \]  

(1.2.2)

and the "Rho" filter is defined by the equation (1.2.3):

\[ \rho(t) = \int_{-\infty}^{\infty} F \, e^{-i2\pi F t} \, dF \]  

(1.2.3)

while this procedure is an exact inversion for the mathematical case, it is not exact when it is digitized and integrals are replaced by summations.

2. Slant stack inversion by phase shift

As for the forward process, we may also attempt to carry out slant stack inversion by phase shift in frequency domain for digitized data: performing 1-D FFT along \( \tau \) axis, followed by phase shifting, then summing these amplitudes horizontally for each frequency with a "Rho" filter applied, and finally 1-D IFFT will reconstruct the original data (Figure 1.2.2). However, as explained later, this attempt at inversion is not exact.
Figure 1.2.2 Scheme to construct slant stack inversion by phase shift. (p, F) data are obtained by 1-D FFT along each column of the (p, \( \tau \)) data. (x, F) data are obtained by phase shift, which is accomplished by matrix multiplication with "Rho" filter applied. (x, t) data are obtained by 1-D IFFT along each column of the (x, F) data.

The implementation of the inverse process in the mathematical case can also be separated into 3 steps:

(1) **Forward Fourier transform:**

To transform data from (p, \( \tau \)) domain into (p, F) domain:

\[
D(p, F) = \int_{-\infty}^{\infty} d(p, \tau) e^{i2\pi F \tau} d\tau
\]  

(1.2.4)

where \( D(p, F) \) is the Fourier transform of the \( d(p, \tau) \) data.
(2) Phase shifting and summing horizontally:

In this process, \((p, F)\) data are transformed by function \(e^{i2\pi Fp_x}\) and a "Rho" filter to obtain the data in \((x, F)\) domain:

\[
G(x, F) = F \int_{-\infty}^{\infty} D(p, F)e^{i2\pi Fp_x} dp
\]  \hspace{1cm} (1.2.5)

where \(G(x, F)\) is the data in \((x, F)\) domain, \(F\) is the so-called "Rho" filter.

(3) Inverse Fourier transform

IFFT transform data from \((x, F)\) domain into \((x, t)\) domain by equation:

\[
f(x, t) = \int_{-\infty}^{\infty} G(x, F)e^{-i2\pi Ft} dF
\]  \hspace{1cm} (1.2.6)

where \(f(x, t)\) is the reconstruction from the \(\tau-p\) domain.

[1.3] Slant-stack Inversion by a Second Slant Stack with an "Rho" Filter Is an Exact Inverse of the Forward Transform

Equations (1.1.3) and (1.2.6), (1.1.5) and (1.2.4) are forward and inverse Fourier transforms and are reversible. So we need to prove that equations (1.1.4) and (1.2.5), transformed by the kernel functions \(e^{-i2\pi Fp_x}\) and \(e^{i2\pi Fp_x}\) with "Rho" filter applied, are also reversible.

Changing the integral variable \(x\) into \(y\) in equation (1.1.4) (Because it is different from the \(x\) in the equation (1.2.5)) and substitute the equation (1.1.4) into the RHS of equation (1.2.5), we have:
The RHS of equation (1.2.5) = \[ F \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(y, F) e^{-i2\pi Fp} dy e^{i2\pi Fp} dp \]

= \[ F \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(y, F) e^{i2\pi Fp(y-x)} dp dy \]

= \[ F \int_{-\infty}^{\infty} \left( \int_{-\infty}^{\infty} e^{i2\pi Fp(y-x)} dp \right) G(y, F) dy \]

(Setting \( Fp = q \))

= \[ F \int_{-\infty}^{\infty} \left( \int_{-\infty}^{\infty} e^{i2\pi q(y-x)} dq \right) G(y, F) dy \]

= \[ \int_{-\infty}^{\infty} \left( \int_{-\infty}^{\infty} e^{i2\pi q(y-x)} dq \right) G(y, F) dy \]

(1.2.7)

where \( \int_{-\infty}^{\infty} e^{i2\pi q(y-x)} dq \) is the standard form of a delta function \( \delta(x - y) \). A delta function has the characteristic that any function convolved with it equals that function.

That is:

\[ f(x) = \int_{-\infty}^{\infty} f(y) \delta(y - x) dy \]

(1.2.8)

Re-write equation (1.2.7) from equations (1.2.8), we have:
The RHS of equation (1.2.5) =

\[ \int_{-\infty}^{\infty} G(y, F) \delta(x - y) dy \]

= \( G(x, F) \)

= The LHS of equation (1.2.5)

Thus, equations (1.1.4) and (1.2.5) are also reversible. This mathematical derivation shows that slant stack inversion by a second slant stack with "Rho" filter applied is the exact inverse of the forward transform.
[1.4] Relation between Slant Stack and 2-D Fourier transform

If we combine equations (1.1.3) and (1.1.4), the slant stack in frequency domain is expressed as:

$$D(p, F) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, t) e^{i2\pi(Ft - Fp)} \, dx \, dt$$

(1.3.1)

The 2-D Fourier transform of $f(x, t)$ is expressed as:

$$D'(K, F) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, t) e^{i2\pi(Ft + Kx)} \, dx \, dt.$$  

(1.3.2)

Hence the coefficient of $2\pi it$ is the frequency $F$ and the coefficient of $2\pi ix$ is the frequency $K$. Picking out these coefficients in equation (1.3.1), we see that $K = -Fp$ (Wade and Gardner, 1988). Then from equations (1.3.1) and (1.3.2), we have:

$$D(p, F) = D'(K=-Fp, F).$$  

(1.3.3)

Equation (1.3.3) implies that the radial direction of the 2-D Fourier transform is exactly the 1-D Fourier transform of the slant stack. On the other hand, the data along the hyperbola defined by $p = -\frac{K}{F}$ in the 1-D Fourier transform of the slant stack (Figure 1.3.1c) are exactly a column of the 2-D Fourier transform at their corresponding
wavenumber K's (Figure 1.3.1). This relation between slant stack and Fourier transform is called the "projection-slice" theorem.

Figure 1.3.1 The relation between 2-D Fourier transform and the 1-D Fourier transform of τ-p transform. A radial line in (F, K) domain (a) is a column in (p, F) domain (b). A hyperbola in (p, F) domain (c) is a column in (F, K) domain (d).

To simplify:

\[ f(x, t) \text{ 2-D FFT} > G(K, F) \text{ Radial direction} > D(p = -\frac{K}{F}, F) \text{ 1-D IFFT} > d(p, τ), \]

or:

\[ d(p, τ) \text{ 1-D FFT} > D(p, F) \text{ Hyperbolic line} > G(K = -pF, F) \text{ 2-D IFFT} > f(-x, t) \]
This relation provides the fundamental basis for performing slant stack in \((F, K)\) domain, as in the paper by Wade and Gardner (1988).
Chapter 2
SCHEMES TO CONSTRUCT SLANT STACK
AND ITS INVERSION

Introduction

Based on the previous discussion, slant stack and its inversion can be carried out by three different methods:

1) Direct mapping in the time domain;
2) Phase shift and summation in the frequency domain;
3) 2-D Fourier transform (F, K) implementation.

The following discusses these three methods for constructing conventional slant stack and inversion, in which the discrete formulas are used to replace the mathematical formula because actual data are always discretely sampled.

[2.1] Slant Stack and its Inversion by direct mapping in time domain

Slant stack by direct mapping is based on its definition by equation (1.1.1). In practice, real seismic data are sampled discretely. The discrete \( \tau-p \) transform, obtained by changing equation (1.1.1), is given by:

\[
d(p, \tau_p) = \sum_{i=-n}^{n} f(x_i, t_i = \tau_p + p x_i)
\]  
(2.1.1)

where \((2n+1) = \) the number of seismic traces used in the transform,

- \(x_i = \) position of the seismic trace or offset,
- \(f(x_i, t_j) = \) amplitude at point \((x_i, t_j)\) in the \((x, t)\) domain, where \(t_j = \tau_p + p x_i,\)
- \(d(p, \tau_p) = \) amplitude at point \((p, \tau_p)\) in the tau-p domain.
By equation (2.1.1), we construct the slant stack gather directly from the data. However, during this process, data interpolation is required because the slanting line does not necessarily intersect with the traces at the grid points (Figure 2.1.1a).

![Figure 2.1.1 Slant stack in (x, t) domain requires data interpolation (a) Forward process, (b) inverse process.](image)

The original data can approximately be reconstructed based on its definition by equation (1.2.1). If we follow the mathematical inverse transform, we obtain equation (2.1.2) by digitizing equation (1.2.1):

\[
f(x_i, t_j) = \sum_{\alpha=1}^{np} d'(p_\alpha, t_j - p_\alpha x_i), \quad (2.1.2)
\]

where \(np\) is the number of \(p\) gather, and \(d'\) is equal to \(d\) convolving with "Rho" filter as described in equations (1.2.2) and (1.2.3). During this process, data interpolation is also required because the slanting line with slope \(-x_i\) does not necessarily intersect with the \(p\) traces at the grid points in the \(\tau-p\) domain (Figure 2.1.1b).

In most cases this digitized inverse gives a fairly good result. But, as we show later, it is not an exact inverse. An exact inverse of the linear equations is not practical.
when np (number of p traces), nx (number of input traces), and nt (number of sampling points) are large numbers.

[2.2] Slant Stack and its Inversion by phase shift in the frequency domain

Slant stack by this method is based on equations (1.1.2) to (1.1.5). In this method, the input traces are transformed from time domain to frequency domain and then each row vector corresponding to a particular frequency is treated separately. Each row (vector) in (x, F) domain is multiplied by phase shift factors (matrix) to produce a row (vector) in (p, F) domain (Figure 2.2.1). Then perform 1-D inverse Fourier transform to make the τ-p data.

![Diagram showing slant stack process](image)

**Figure 2.2.1** In (x, F) domain, each row vector (a) is multiplied by phase shift operator (b) to produce a row in (p, F) domain (c).

The inversion is done by a second slant stack with "Rho" filter applied, based on equations (1.2.4) to (1.2.6).

Both forward and inverse transform can be efficiently calculated by the matrix multiplication in frequency domain. Following are the digitizing formula during each process:
1) Forward Fourier transform

In this process, we carry out 1-D Fourier transform along each input trace in the (x, t) domain (forward) to obtain the data in the (x, F) domain, or along each p trace in the (τ, p) domain (inverse) to obtain the data in the (p, F) domain.

In the forward transform, the discrete FFT formula is obtained by digitizing equation (1.1.3):

$$G(x, F_j) = \sqrt{\frac{1}{nt}} \sum_{\alpha=0}^{nt-1} f(x, t_\alpha) e^{i2\pi \frac{\alpha}{nt}} \quad (j = 0, 2, 3, ..., nt-1)$$

(2.2.1)

where nt is total sampling points, t_\alpha is the sampling time series, F_j is the corresponding frequency series, G(x, F_j) is the data in (x, F) domain.

In the inverse transform, the discrete FFT formula is obtained by digitizing equation (1.2.4):

$$D(p, F_j) = \sqrt{\frac{1}{nt}} \sum_{\alpha=0}^{nt-1} d(p, \tau_\alpha) e^{i2\pi \frac{\alpha}{nt}} \quad (j = 0, 2, 3, ..., nt-1)$$

(2.2.2)

2) Phase shift and summing horizontally

The forward process is carried out by the multiplication of the row vector G(x, F_i) defined by each individual frequency F_i in (x, F) domain with its operator \[e^{i2\pi F_ipx}\] (matrix) to obtain a row D(p, F_i) in the (p, F) domain at the corresponding frequency F_i (Figure 2.2.1). The digitizing formula can be obtained from equation (1.1.4) by matrix multiplication as:

$$D(p_k, F_i) = G(x_j, F_i) \ast M$$

(2.2.3)
And the inverse process by digitizing equation (1.2.5):

\[ G(x_j, F_i) = D(p_k, F_i)^*M' \]  

(2.2.4)

where:

\[ M = \begin{bmatrix} e^{i2\pi F_i p_k x_j} \end{bmatrix} \]

i=0, 2, 3, ..., NT/2,

x_j = x_0, x_2, x_3, ..., x_{nx-1},

p_k = p_0, p_2, p_3, ..., p_{np-1}.

and matrix M' is conjugate-transpose of the matrix M, nx is the total input traces, np is the number of p traces made, and p_0 and p_{np-1} are minimum and maximum p values in the transformed domain respectively. Matrix M has np columns and nx rows as:

\[ M = \begin{bmatrix} e^{i2\pi F_i p_0 x_0} & e^{i2\pi F_i p_1 x_0} & e^{i2\pi F_i p_2 x_0} & \cdots & e^{i2\pi F_i p_{np-1} x_0} \\ e^{i2\pi F_i p_0 x_1} & e^{i2\pi F_i p_1 x_1} & e^{i2\pi F_i p_2 x_1} & \cdots & e^{i2\pi F_i p_{np-1} x_1} \\ e^{i2\pi F_i p_0 x_2} & e^{i2\pi F_i p_1 x_2} & e^{i2\pi F_i p_2 x_2} & \cdots & e^{i2\pi F_i p_{np-1} x_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e^{i2\pi F_i p_0 x_{nx-1}} & e^{i2\pi F_i p_1 x_{nx-1}} & e^{i2\pi F_i p_2 x_{nx-1}} & \cdots & e^{i2\pi F_i p_{np-1} x_{nx-1}} \end{bmatrix} = e^A \]

where:

\[ A = w_0^* \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix} + w_1^* \begin{bmatrix} 0 & 1 & 2 & \cdots & (np-1) \\ 0 & 1 & 2 & \cdots & (np-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 2 & \cdots & (np-1) \end{bmatrix}. \]
\[
\begin{pmatrix}
0 & 0 & 0 & \ldots & 0 \\
1 & 1 & 1 & \ldots & 1 \\
2 & 2 & 2 & \ldots & 2 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
(n-1) & (n-1) & (n-1) & \ldots & (n-1)
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & \ldots & 0 \\
0 & 1 \times 1 & 1 \times 2 & 1 \times 3 & \ldots & 1 \times (n_{p}-1) \\
0 & 2 \times 1 & 2 \times 2 & 2 \times 3 & \ldots & 2 \times (n_{p}-1) \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & (n-1) \times 1 & (n-1) \times 2 & (n-1) \times 3 & \ldots & (n-1)(n_{p}-1)
\end{pmatrix}
^*
\]

(2.2.5)

and:

\[
w_0 = i2\pi F_i p_0 x_0,
\]

\[
w_1 = i2\pi F_i d p_0 x_0,
\]

\[
w_2 = i2\pi F_i p_0 dx,
\]

\[
w_3 = i2\pi F_i dp dx.
\]

and \(x_0\) is the offset of the trace on which the slant stack takes place (i.e., as intercept time axis), \(dp\) is the \(p\) increment, and \(dx\) is the trace spacing.

In practice, we do not need to generate the matrix by evaluating the exponential functions at every frequency. For example, if we have generated the matrix at frequency \(F_i\), then the matrix at \(F_{i+1}\) can be obtained by matrix multiplication as:

\[
[e^{i2\pi F_{i+1} p x}] = [e^{i2\pi (F_i + dF) p x}] = [e^{i2\pi F_{ip} p x}] \ast [e^{i2\pi dF p x}]
\]

(2.2.6)

where \(dF\) is the frequency increment, which is determined by:
\[ dF = \frac{1}{NT^*DT}. \] (2.2.7)

Because \( F_i \) is from \( F_0 \) to \( F_{NT/2} \) with increment \( dF \), we only need to generate the matrix at zero frequency \( (F_0) \), the elements of which are all one's, and the matrix \( [e^{i2\pi FDp}] \).

Then the matrix at all other frequencies can be obtained by the multiplication of the corresponding elements of the matrices as in the equation (2.2.6).

Notice that the exact numerical inversion of the forward linear equations does not use a "Rho" filter and no interpolation is needed. This method of inversion is practical because each frequency gives a fairly large set of linear equations which must be solved. Without transformation from \( t \) to \( F \), one giant set of equations must be solved (Beylkin, 1987).

(3) Inverse Fourier transform

In the forward transform, this process constructs the data from \( (p, F) \) domain into \( (p, \tau) \). The following is the discrete formula for the inverse Fourier transform base on equation (1.1.5):

\[ d(p, \tau_j) = \sqrt{\frac{1}{nt}} \sum_{\beta=0}^{nt/2} D(p, F_\beta) e^{-i2\pi \beta \frac{j}{nt}} \quad (j = 0, 2, 3, ..., nt-1) \] (2.2.8)

where \( d(p, \tau) \) is the slant stack of \( f(x, t) \).

In the inverse transform, this process reconstructs data in \( (x, t) \) domain from \( (x, F) \) domain. Its discrete formula is obtained from the equation (1.2.6):

\[ f(x, t_j) = \sqrt{\frac{1}{nt}} \sum_{\beta=0}^{nt/2} G(x, F_\beta) e^{-i2\pi \beta \frac{j}{nt}} \quad (j = 0, 2, 3, ..., nt-1) \] (2.2.9)
[2.3] Slant stack and inversion by (F, K) implementation

Slant stack in this domain is based on the relation between 2-D Fourier transform and slant stack, or "projection-slice" theorem, as stated in Section [1.3]. In this method the 1-D Fourier transform of the slant stack data is obtained by picking out these data along the radial line of the Fourier coefficients of the 2-D Fourier transform of the input data. The result makes a column at the corresponding p value in the (p, F) domain (Figure 1.3.1). The p value is the slope of the radial line, which is determined by:

\[
p = -\frac{K}{F}
\]  

(2.3.1)

The discrete formula for 2-D Fourier transform is:

\[
G(K_j, F_i) = \sum_{m=0}^{nx-1} \sum_{n=0}^{nt-1} f(x_m, t_n) e^{-i2\pi mj/nx} e^{-i2\pi ml/nt}
\]  

(2.3.2)

After the (p, F) data have been mapped at all p values, the (p, τ) data are obtained by 1-D inverse Fourier transform along each column of the (p, F) data (Wade and Gardner, 1988).

As that of the (x, t) direct mapping, slant stack by this method also requires data interpolation because the (F, K) data are discrete and not all the radial lines intersect at the grid points.

The inverse transform is the exactly reverse process of the forward as in the Figure 1.3.1. First we perform 1-D Fourier transform along each column of the (p, τ) data to produce the (p, F) data, then put each column of the (p, F) data into the corresponding radial direction of the (F, K) domain. After the (F, K) data have been
made, 2-D inverse Fourier transform will reconstruct the \((x, t)\) data. The discrete formula for 2-D inverse Fourier transform is given by:

\[
f(x_m, t_n) = \sum_{j=0}^{nx-1} \sum_{l=0}^{nt-1} G(K_j, F_l) e^{-i2\pi mj/nx} e^{-i2\pi ml/nt}
\]  

(2.3.3)

Alternatively, we can pick out those data in the \((p, F)\) domain along hyperbolic line with constant \(K\) value. The result makes a column at the corresponding \(K\) value in the \((F, K)\) domain (Figure 1.3.1). The \(K\) value can be determined by:

\[
K = -pF
\]  

(2.3.4)

As we can see, this process also needs data interpolation in the frequency domain.
Chapter 3
Problems with the Conventional Slant Stack

Introduction

When a slant stack is inverted using a "Rho" filter and a second slant stack, the results are sometimes disappointing even though the procedure is a direct discretization of the exact mathematical formula. Beylkin (1987) has proved that poor results are to be expected in general, and he developed an alternative inversion procedure based on the solution of a system of simultaneous equations for each frequency.

The problem with inversion using a second slant stack is illustrated in Figure 3. The input data in Figure 3a consist of a 20 Hz sinusoid in trace 6, a 40 Hz sinusoid in trace 9, a 60 Hz sinusoid in trace 12 and their sum in trace 15, with all traces normalized to unit amplitude. The other traces have zero amplitudes. There are a total of 31 traces with a spacing of 15 m between traces and 4 ms between time samples. A slant stack of these data was made in (x, F) domain from a minimum slope of zero to a maximum of $68.27 \times 10^{-3}$ s/m in 186 steps with $3.67 \times 10^{-5}$ s/m as p increment (Figure 3b) (See Model 1 in Chapter 5 for more detail), a "Rho" filter was applied, and a second slant stack was made back to the same trace spacing.

Figure 3c shows the result of this inversion. The amplitudes of the original sinusoids are not recovered accurately, and some traces that were originally zero now contain appreciable signals.

The reasons for these poor results are discussed in this chapter.
[3.1] Problems with direct mapping

The poor results associated with the conventional slant stack inverse process in Figure 3 can be illustrated from the (x, t) domain. For simplification, consider that the (x, t) data consist of a single frequency sinusoid in all the traces with the peaks of the sinusoids lining up along a slanting line with slope Q. These (x, t) data can be expressed as:

\[ f(x, t) = e^{i2\pi F(t-Qx)} \]  (3.1.1)

Then the tau-p data are given by the equation (2.1.1) as:

\[
d(p, \tau) = \sum_{j=-n}^{n} f(x_j, t_j=\tau+px_j)
\]

\[
= \sum_{j=-n}^{n} e^{i2\pi F(\tau+px_j-Qx_j)}
\]

\[
= e^{i2\pi F\tau} \sum_{j=-n}^{n} e^{i2\pi F(p-Q)x_j},
\]  (3.1.2)

Writing \( x_j = j\Delta x \), where \( \Delta x \) is the trace spacing, equation (3.1.2) becomes:

\[
d(p, \tau) = e^{i2\pi F\tau} \sum_{j=-n}^{n} e^{i2\pi F(p-Q)(j\Delta x)}
\]

\[
= e^{i2\pi F\tau} \frac{\sin[(2n+1)\pi F(p-Q)\Delta x]}{\sin[\pi F(p-Q)\Delta x]},
\]  (3.1.3)

where \((2n+1)\) is the number of the traces.
Figure 3 Reconstruction of original data by standard slant stack and inversion with a "Rho" filter applied, showing the poor result even though the procedure is a direct discretization of the exact mathematical formula. (a) test data; (b) standard slant stack; (c) the inversion, using standard slant stack (see Model 1 in Section 5.1).
The expression \( \frac{\sin[(2n+1)\pi F(p-Q)\Delta x]}{\sin[\pi F(p-Q)\Delta x]} \) consists of two sine functions, \( \sin[(2n+1)\pi F(p-Q)\Delta x] \) has a period \( \Delta p_1 = \frac{2}{F(2n+1)\Delta x} \) with respect to \( p \), and \( \sin[\pi F(p-Q)\Delta x] \) has period of \( \Delta p_2 = \frac{2}{F\Delta x} \) with respect to \( p \). There are two zero crossings within one period. Thus the expression in equation (3.1.3) has a period of \( \Delta p = \frac{1}{F\Delta x} \) with respect to \( p \). Within one period \( \Delta p \), there are \( 2n+1 \) periods of the numerator (Figure 3.1.1). So when performing slant stack, if we chose a \( p \) range larger than the period of this expression, we will get repetitive information. On the other hand, if we chose a \( p \) range smaller than the period, we will lose some information.

However, this period depends on frequency and seismic data contain a range of frequencies, say from \( F_{\text{min}} \) to \( F_{\text{max}} \) such as in Figure 3. Thus, during the transformation, we need to vary the \( p \) range according to frequency in order that only one period be retained in the \( (p, \tau) \) domain. That is:

\[
p_{\text{range}} = \frac{1}{F\Delta x}.
\] (3.1.4)

Moreover, \( \Delta p \) (the increment of \( p \)) must be small enough to avoid the aliasing. As stated, we have a period \( \frac{2}{(2n+1)F\Delta x} \) for the numerator. According to the sampling theory, we need to have at least 2 sampling points per period, so \( \Delta p \) must satisfy:

\[
\Delta p \leq \frac{\Delta p}{2n+1} = \frac{1}{(2n+1)F\Delta x}.
\] (3.1.5)

Otherwise, we will not be able to recover the original data.

But \( \Delta p \) is also changing with frequency according to equation (3.1.5). The lowest is \( \frac{1}{F_{\text{max}}\Delta x} \). So if the frequencies in the data range from \( F_{\text{min}} \) to \( F_{\text{max}} \), we can set:
Figure 3.1.1 Periodical characteristic of slant stack operator, from expression $\frac{\sin((2n+1)\pi F(p-Q)\Delta x)}{\sin(\pi F(p-Q)\Delta x)}$ in the equation (3.1.3), (a) plot of bottom in the expression with respect to $p$ for $F = 40$ Hz; (b) plot of top in the expression with respect to $p$ for $F = 40$ Hz; (c) plot of the expression with respect to $p$ for $F = 40$ Hz.
\[ \Delta p = \frac{1}{(2n+1)F_{\text{max}} \Delta x}. \]  

(3.1.6)

where \((2n+1)\) is the number of input traces in the \((x, t)\) domain, and use this \(\Delta p\) for all frequencies.

The problem is that the conventional slant stack has a fixed \(p\) range for all the frequencies and there is no exact criterion for the best \(p\) range or \(p\) increment. This means that different periods of data are used for different frequencies during the transformation (Figure 3.1.1). It ignores the frequency factor which is key to making the transform invertible.

The above analysis shows that we need to vary the \(p\) range according to the frequency and use a proper \(p\) increment in order to obtain all necessary information.

[3.2] Problem by phase shift

To obtain tau-\(p\) data from \((x, F)\) domain, we denote a row of the \((x, F)\) data by the vector \(C\), a row of the corresponding \((p, F)\) data by the vector \(D\), and matrix \(M\) as the transform operator, that is:

\[ CM = D \]  

(3.2.1)

where:

\[ M = \begin{bmatrix} e^{i2\pi p F_i} \end{bmatrix}, \]  

(3.2.2)

\[ x = [x_0, x_1, \ldots, x_{nx-1}]', \]  

(3.2.3)

\[ p = [p_0, p_1, \ldots, p_{np-1}]. \]  

(3.2.4)

Equation (3.2.1) represents a set of linear equations which can be inverted by standard method:
where matrix $M'$ denotes the conjugate-transpose of matrix $M$.

As usual, $x = \alpha \Delta x \; (\alpha = -n:1:n)$ for seismic data. So:

$$M = \left[ e^{i2\pi \alpha \Delta x p F_i} \right].$$

Matrix $M$ consists of $(2n+1)$ (the number of seismic traces) rows and $np$ (number of $p$ traces) columns. As $p$ increases, the columns of this matrix will repeat when $p$ increases by $\Delta p$

$$\Delta p = \frac{1}{F_i \Delta x}.$$  

(3.2.6)

If we chose $p$ range larger than $\Delta p$, the matrix $M$ will be ill-conditioned. The problem is then obvious. Meanwhile, $\Delta p$ is changing with frequency. Thus the $p$ range has also to be changed accordingly.

If we chose a fixed range of $p$ to construct $p$ gather, this will inevitably either cause aliasing (or repetition) at high frequency (matrix $M$ becomes singular) or/and not have enough range of $p$ at low frequency because higher frequency has smaller period while lower frequency has larger period with respect to $p$. Thus the constructed slant stack either has repetitive information (at high frequencies) or/and does not have enough information (at lower frequencies).
[3.3] Problem with conventional (F, K) implementation

Using (F, K) implementation for slant stack, the Fourier transform in time of the slant stacked data is obtained by picking out radial lines of the Fourier coefficients from the 2-D Fourier transform of the input data. In order to get an exact inverse, the mapping from (F, K) domain to its transformed (p, F) domain must be in 1-1 correspondence. However, in the standard slant stack, when a radial line meets one side of the (K, F) diagram it is wrapped around to the other side and intersects with other radial lines as illustrated in Figure 3.3.1.

![Diagram of (K, F) domain showing radial lines and intersections](image)

Figure 3.3.1 In the standard slant stack, values in the (K, F) domain of the data are picked out for all frequencies along radial line AB and its wraparound paths CD and EF. They intersect with another radial line AG. Thus the mapping from (K, F) to (p, F) is not 1:1.

Thus no matter how nearly horizontal the radial line may be, Fourier components are picked out at all frequencies by wrapping the line repeatedly around the diagram until it
reaches the Nyquist frequency. This means that each point can be picked out by several radial lines because of the wraparound, and hence mapping from \((K, F)\) to \((p, F)\) is not \(1:1\).
Chapter 4
The New Approach to the Slant Stack

[4.1] Introduction

If the data were increasingly finely sampled in space, the mathematical inversion theorem would be more and more closely approached. For example, if ten traces were interpolated to one thousand traces then the slant stack of the one thousand traces would be more accurately inverted than the original ten by a "Rho" filter and a second slant stack. One basic idea in the new method is to create a transformation as if the data traces were interpolated to a fine sampling and slant stacked without actually doing the interpolation.

To explain this approach in the \((F, K)\) domain, the original traces are interpolated to a fine sampling interval in the \(x\) direction by padding zero traces in the plus and minus \(K\) directions. Then a radial line does not wrap around the original \((K, F)\) diagram but picks up zero coefficients from the padding columns (Figure 4.1.1).

In short, the radial lines are not wrapped around. When a steep radial line meets one side of the \((K, F)\) diagram, the higher frequency components are put equal to zero. Thus the modified slant stack is the same as a standard slant stack except that some \(p\) traces have a high cut filter applied. By this modified method, each point in the \((K, F)\) diagram is picked out by only one radial line, and hence the mapping from \((K, F)\) to \((p, F)\) is 1:1.
Figure 4.1.1 In the modified method, the values outside the (K, F) diagram are padded with zeros so the diagram has wider K range. The radial line AB then does not wrap around and only values along AB are used. The values along BC are all zeros. Thus no radial lines will intersect with each other.

In practice, the phase shift method may be preferred to (K, F) implementation because no actual interpolation of Fourier coefficients is required. In (x, F) domain, we can change the p range based on the variation of the period of the slant stack operator with frequency, discussed in Chapter 3. Their relation determines a hyperbolic p range (Figure 4.1.2).
Figure 4.1.2 In modified method, the $p$ range is changing with frequencies and defines a hyperbolic zone. Outside this zone, zeros are assigned to all the grid points. Events with slope falling inside the $\text{min}_p\_\text{range}$ will have concentrated $\tau$-$p$ data. The minimum $p$ value is chosen to suit the different applications.

By assigning zeros to all grid points outside the hyperbolic boundaries, we are able to:
(1) get rid of the repetitive information at higher frequencies, and (2) include all necessary information at lower frequencies. Thus the 1:1 is satisfied.
[4.2] Method

[4.2.1] Forward Slant Stack

The discrete formula for constructing a slant stack in \((x, F)\) domain has been discussed in Section [2.2]. Following are steps that actually carry out these processes, in which the problems with the conventional method are eliminated.

1. Transform the data from \((x, t)\) domain into \((x, F)\) domain:

   This is carried out by performing 1-D FFT along each input seismic trace or column. The sequence of the resulting coefficients along each column is: zero frequency, positive frequencies, Nyquist frequency, and negative frequencies in reversed order. Because the input seismic data are real numbers, the positive and negative frequencies are complex conjugates symmetrically placed around the Nyquist frequency.

2. Rearrange the Fourier coefficients for the transformation.

   Because of the symmetry of the Fourier coefficients, we do not need to use those negative frequency coefficients. The zero frequency component is zero for seismic data. The Nyquist frequency is the highest frequency in the data set, and its coefficient is zero according to the sampling theorem because we usually assume that seismic data are not aliased in frequency. Thus we only have at most those coefficients of the positive frequencies for the transformation.

3. Select the maximum and minimum frequency for the transformation:

   This process is usually determined by the main frequency range in the data set. The maximum frequency \((F_{\text{max}})\) is used to determined the \(p\) increment \((dp)\), the minimum frequency \((F_{\text{min}})\) is used to determined the total number of \(p\) traces \((np)\), as discussed below.
(4) Determination of number of p traces and their increment:

The p increment as discussed in Section [3.1] is given by the equation (3.1.6).
The maximum p range is at $F_{\text{min}}$, given by the equation:

$$\max_p\_\text{range} = \frac{1}{F_{\text{min}} \Delta x}$$  \hspace{1cm} (4.2.1)

The total p traces is then determined as:

$$np = \frac{\max_p\_\text{range}}{dp}$$

$$= (2n+1) \frac{F_{\text{max}}}{F_{\text{min}}}$$  \hspace{1cm} (4.2.2)

(5) Select the minimum p value and the trace on which the slant stack takes place as the intercept time axis:

The choice of the minimum p value is dependent on the purposes of the transformation. We will note the minimum p value chosen will not affect the reconstruction as long as no muting occurs. We will also note that if the slope of a linear event falls into the center portion of the p range (the p range at the maximum frequency, Figure 4.1.2), this event will be concentrated near a point in the $\tau$-p domain (discussed below). So in order to mute an event, we need to chose the minimum p value so that this event will be concentrated in the $\tau$-p domain in the center portion of the p range.

The choice of the trace on which the slant stack takes place has no effect on the reconstruction. It merely serves as the intercept time axis. So we can chose any seismic trace as the intercept time axis.
(6) Calculate the matrix \([e^0]\) and \([e^{i2\pi dFp_x}]\)

The matrix \([e^0]\) is the slant stack operator at zero frequency. It has np (total \(p\) traces as calculated on equation (4.2.4)) columns and nx (input seismic traces) rows, its elements are all one's.

The matrix \([e^{i2\pi dFp_x}]\) has the same dimension as matrix \([e^0]\). The \(dF\) is calculated using equation (2.2.6), the \(p\) (vector) using the equation (3.2.4) with increment \(dp\) using equation (3.1.5), and the \(x\) (transposed vector) using equation (3.2.3) with increment of trace spacing. This matrix is calculated based on the equation (2.2.4).

(7) Slant stack by matrix multiplication with \(p\) range varying with frequency:

During this process, each row vector \(D(p, F_i)\) defined by each individual frequency \(F_i\) in the \((p, F)\) domain is obtained by the multiplication of the corresponding row vector \(G(x, F_i)\) of the \((x, F)\) data and the matrix \([e^{i2\pi F_ip_x}]\), in which the range of the \(p\) is determined by \(\frac{1}{F_idx}\). That is:

\[
D(p, F_i) = G(x, F_i)*[e^{i2\pi F_ip_x}] \tag{4.2.3}
\]

The \(p\) range in this equation which is varying with frequency will remove the repetitive information at higher frequencies while including sufficient information at lower frequencies.

If no filtering is desired, we chose the \(p\) range the same for all frequencies lower than the minimum frequency \(F_{\text{min}}\), and equal to that at the minimum frequency \(F_{\text{min}}\). The \(p\) range at frequencies higher than the maximum frequency \(F_{\text{max}}\) are all equal to that at the maximum frequency \(F_{\text{max}}\).

If we want to filter out those frequencies outside the band defined by the \(F_{\text{min}}\) and \(F_{\text{max}}\), simply zero out their Fourier coefficients.
Meanwhile, a new matrix \( [e^{i2\pi(F_i+dF)p_x}] \), which is slant stack operator at frequency \( F_i+dF \), is generated by the multiplication of the corresponding elements of matrices \( [e^{i2\pi F_i p_x}] \) and \( [e^{i2\pi dF p_x}] \).

(8) After the \((p, F)\) data have been generated, perform 1-D inverse FFT along each column of the \((p, F)\) matrix to obtain the slant stack gather:

Advantages of the new scheme are: (i) we can choose any starting \( p \) values, as long as their range is \( -\frac{1}{F_i \Delta x} \). Thus we do not need to find out what is the upper and lower limit of \( p \) as conventional. (ii) The new transform contains all information in the original data. (iii) Muting undesired events can still be performed inside the range and then inverse back to original \((x, t)\) domain.

[4.2.2] Inverse Slant Stack

The inverse process is very similar to the forward process. But some cautions need to be taken:

(1) The \( p \) range and its increment, and the \( x \) range and its increment are all fixed regardless of frequency.

During forward transform, we assigned zeros to all those points outside the hyperbolic \( p \) range (Figure 4.1.2), so we do not need to worry about the repetitions or losses of information in the inverse process. The \( p \) increment has also been determined during the forward process. The \( x \) range and its increment are fixed because we want to reconstruct the original seismic traces which have fixed offset and fixed number of traces. Even the trace interpolation will not cause problems because the increase in the number of traces will be compensated for by the decrease in the trace spacing.
In a word, the inverse process will not have repetitive information at higher frequencies or losses of information at lower frequency because all these dangers have been removed during the forward process.

(2) The tau-p data are now shifted by the amount defined by matrix $[e^{-i2\pi Fxp}]$.

Matrix $[e^{-i2\pi Fxp}]$ is the conjugate transpose of matrix $[e^{i2\pi Fpx}]$. It has nx columns and np rows. So for the inverse process, we do not need to generate the matrix again, but simply take the conjugate transpose of the matrix generated during the forward process.

(3) Spectrum restoration

Mathematical derivation has shown that the "Rho" filter is required for the inverse slant stack transformation in order to properly restore the frequency spectrum. It is carried out by the multiplication of the magnitude of each frequency component with its absolute frequency. In reality, we can apply this Rho filter either during the forward process or during the inverse process (Novotny, 1990).

Appended is a complete listing of the program for the slant stack and inversion.
Chapter 5
Examples

In order to visualize how the new method of slant stack works and to make a comparison with the ordinary method, several synthetic models were generated to do both slant stack and its inversion. All the input models consist of 31 traces with 15 m trace spacing and 512 sampling points in each trace with 4 ms sampling interval.

The strategy in choosing the test models is based on the following consideration:

(1) Simple models: such as an impulse, a single wavelet, or a single frequency sinusoids. This is because it is easy to visualize what the forward or inverse transform will be with these inputs.

(2) A simple synthetic seismic section: with events having both positive and negative slope in the test data, examine the forward and inverse transform with all positive transformed ray parameter.

In the modified method, the $F_{\text{min}}$ and $F_{\text{max}}$ were set equal to 10 Hz and 60 Hz, respectively, which were determined by the main frequency range in the data set. Thus the $p$ range in the transformed domain is equal to $\frac{1}{F_{\text{min}} \Delta x} = \frac{1}{10 \times 15} = 6.67 \times 10^{-3}$ m/s.

The minimum value of $p$ was chosen to suit different applications (mainly for muting a particular event, discussed in Chapter 6). During the inverse process, a "Rho" filter was applied and a second slant stack was made back to the same trace spacing.

In the standard slant stack, the slant stack gather was constructed with exactly the same transformed parameters as those of the modified method. The inverse process is also the same as that of the modified method, with a "Rho" filter applied and a second slant stack made back the same trace spacing.

In both methods, trace 16 was chosen to serve as the intercept time axis, along which the slant stack takes place.
[5.1] Model 1, Several Sinusoids

This input model has been shown in the introduction section of Chapter 3 in Figure 3a.

(1) Standard slant stack and inversion

The standard slant stack has also been shown in Figure 3b in Chapter 3. The inversion, as shown in the Figure 3c, demonstrates the general problems of the standard method.

(2) Modified slant stack and inversion

Figure 5.1a, the result of the modified slant stack, shows that each frequency has a different p range.

The inverse result is shown in Figure 5.1b. The sinusoids were recovered with the correct amplitude and the other traces were almost zero as in the original data.

(3) Spectrum view

The spectrum of trace 15 of the test data is shown in figure 5.1c, which consists of three frequency components (20, 40, 60 Hz) with the same amplitudes. That of the reconstruction by the modified method is shown in figure 5.1d, showing the spectrum was accurately reconstructed. However, figure 5.1e, the spectrum of the reconstruction by the ordinary method, shows that the original spectrum was distorted.
Figure 5.1 Model 1: the test data, the standard slant stack, and its inversion were shown in Figure 3, (a) modified slant stack; (b) the inversion, using modified slant stack, (c) the spectrum of the test data, (d) the spectrum of the reconstruction by modified method, (e) the spectrum of the reconstruction by standard method.
Spectrum of Original Data

(c)

Spectrum of Reconstruction by Modified Method

(d)

Spectrum of Reconstruction by Ordinary Method

(e)
[5.2] Model 2, Spike Response

In this model, all the input grid points have zero amplitudes except one point in trace 16 (the intercept time axis) normalized to unit amplitude (Figure 5.2a).

(1) Standard slant stack and inversion

   Each p trace by this method consists of a spike in the transformed $\tau$-p domain, lining up to produce a linear event. All the spikes have the same amplitude (Figure 5.2b).

   However, the inversion did not reconstruct the original spike (Figure 5.2c).

(2) The modified slant stack and inversion

   Each p trace still consists of a spike to produce a linear event in the transformed $\tau$-p domain, but their amplitudes gradually decrease outside a central p range as if a high cut filter had been applied to the previous linear event produced by the standard slant stack (Figure 5.2d).

   The inverse satisfactorily reconstructed the original spike (Figure 5.2e).

(3) Spectrum view

   The spectrum of trace 16 which contains the original spike is a line parallel to the frequency axis (Figure 5.2f). The modified method almost reconstructed the spectrum (Figure 5.2g). The bumps were due to the fact that the transformation was discrete. However, that by the standard slant stack produced a slanting spectrum which makes an angle with the frequency axis, instead of parallel to it (Figure 5.2h).

   From this model, we can see that the modified slant stack consists of the standard slant stack plus a high cut filter (which is actually carried out by reducing the p range with increasing frequency, thus it may be faster to implement the modified slant stack than the standard one, if both methods make the same total number of p traces).
Furthermore, the modified method also removed the edge truncation effect while the standard one imposed an edge truncation effect during the reconstruction process (Figures 5.2c and 5.2e).
Figure 5.2 Model 2: (a) test data, (b) standard slant stack; (c) the inversion, using standard slant stack; (d) modified slant stack; (e) the inversion, using modified slant stack; (f) spectrum of the test data; (g) spectrum of the inversion by modified method; (h) spectrum of the inversion by standard method.
(b) Slant Stack by the Ordinary Method

(c) Reconstruction of Slant Stack by the Ordinary Method
Spectrum of Original Data

(f)

Spectrum of Reconstruction by Modified Method

(g)

Spectrum of Reconstruction by Ordinary Method

(h)
[5.3] Model 3, Impulse Response

Figure 5.3 shows the forward and inverse transforms of a single wavelet in trace 6 with trace 16 still serving as the intercept time axis for both the standard and modified slant stack.

Figure 5.3a shows the input wavelet, which is the second derivative of a Gaussian function, generated from the formula:

\[ f(t) = [1 - 2\alpha(t - t1)^2] e^{-\alpha(t-t1)^2}, \]

where \( \alpha = 5000 \), and \( t1 = 1 \) second. The \( t \) is calculated from 0 to the trace length, which is equal to \( 512 \times 0.004 = 2.048 \) seconds.

(1) Standard slant stack and inversion

The standard slant stack produced a linear event, with each of the \( p \) traces containing the same wavelet as the input (Figure 5.3b).

However, the inversion produced distorted image of the original wavelet as shown in Figure 5.3c (see the spectrum), which was complicated by the edge truncation effect.

(2) Modified slant stack and inversion

The modified method produced a linear tapered event, with amplitudes and frequencies decreasing gradually on the both ends of the \( p \) range (Figure 5.3d) as that in the spike model.

The inversion produced a correct image of the input section, as shown in Figure 5.3e (see spectrum).
(3) Spectrum view

Although the spectra of the original input and the two reconstructions have similar shapes, the peak frequency of the original input is 22.5 Hz (Figure 5.3f), that of the reconstruction by the modified method is also 22.5 Hz (Figure 5.3g), but the reconstruction by the standard method produced a peak frequency of 27.5 Hz (Figure 5.3h).
Figure 5.3 Model 3: (a) test data, (b) standard slant stack; (c) the inversion, using standard slant stack; (d) modified slant stack; (e) the inversion, using modified slant stack; (f) spectrum of test data with peak frequency 22.5 Hz; (g) spectrum of the inversion by modified method with peak frequency 22.5 Hz; (h) spectrum of the inversion by standard method with peak frequency 27.5 Hz.
[5.4] Model 4, Input is single frequency sinusoid in all traces

Model 4 consists of single frequency sinusoids (30 Hz) in all the input traces (Figure 5.4a).

(1) Standard slant stack and inversion

The standard method produced 3 sinusoid traces in the transformed domain (Figure 5.4b), two of which were caused by spatial aliasing. In this input model, the period of the slant stack operator at 30 Hz can be obtained as:

\[
p_{\text{range at 30Hz}} = \frac{1}{15 \times 30} = 2.222 \times 10^{-3}
\]

The given p range (6.667 x 10^-3) is three times as large as this period. Thus there are 3 none-zero p traces in the transformed domain.

The result of its inversion is shown on Figure 5.4c. The amplitudes at the ends of some of the reconstructed traces affect the whole data set.

(2) Modified slant stack and inversion

The modified method produced one non-zero trace with the rest almost zero (Figure 5.4d). i.e., the transformed data are not aliased. The inversion accurately reconstructed the original data (Figure 5.4e).
Figure 5.4 Model 4: (a) test data with single frequency sinusoid (40 Hz) in all traces; (b) standard slant stack; (c) the inversion, using standard slant stack; (d) modified slant stack; (e) the inversion, using modified slant stack.
[5.5] Model 5, Input data have events with negative slopes

Model 5 consists of two linear events and two hyperbolae with positive slopes and two linear events and two hyperbolae with negative slopes which are mirror image with each other (Figure 5.5a).

(1) Standard slant stack and inversion

Figure 5.5b is the result of standard method using the previous slant stack parameters, in which the ray parameters are all positive number. The figure shows that only the events with positive slopes were constructed (the events with negative slopes were dispersed over the \( \tau-p \) domain).

Figure 5.5c is the inverse of Figure 5.5b. It shows that the original section was reconstructed with distorted wavelets. It also introduced extra noise to the data set.

(2) Modified slant stack and inversion

In this method, the ray parameters were still positive. The result is shown in Figure 5.5d, in which all the events were constructed, especially those with negative slopes, which were distributed over the whole transformed domain.

However, the inverse very well reconstructed the original data (Figure 5.5e).

In fact, all the reconstruction will be the same regardless of positive or negative ray parameters, or the minimum p value chosen in the modified method.
Figure 5.5 Model 5: (a) test data with both positive and negative slope events; (b) standard slant stack with positive ray parameters used; (c) the inversion, using standard slant stack; (d) modified slant stack with positive ray parameters used, showing dispersion of negative slope events over the transformed domain; (e) the inversion, using modified slant stack.
Chapter 6
Applications

Slant stack and inversion have many applications in various fields. In seismology, they can be used for plane wave decomposition as 2-D FFT, velocity analysis, trace interpolation, and noise removal, etc.

The strategy in choosing the input models is based on the following consideration:

(1) In the filtering (muting) application: strong aliased linear events representing common air wave in seismic record are generated. How the slant stack and inversion, both standard method and modified method, may help in removing these events.

(2) In the application of trace interpolation: a simple synthetic seismic section is generated. How the two methods may help in the trace interpolation.

[6.1] Muting undesired events

Slant stack separates each individual linear event in the transformed domain by its different slope and intercept time which often interfere with one another in the original record. This separation sometimes allows for effective muting of an undesired event.

[6.1.1] Removal of a linear event

Figure 6.1.1a is the test data. It consists of 31 traces with 15 m trace spacing and 512 sampling points with 4 ms sampling interval. There are two linear events with velocities 800 m/s and 2000 m/s, two hyperbole from horizons at depth 1000 m with velocity 2000 m/s and depth 2000 m with velocity 3000 m/s, respectively. The purpose is to mute the linear event with velocity 800 m/s.
(1) Modified method

By proper selection of the minimum p value as discussed in chapter 4, the event with 800 m/s velocity appears in the center of the transformed domain (Figure 6.1.1b). Figure 6.1.1c is the reconstruction of Figure 6.1.1b without muting.

Now zero out the \( \tau-p \) data inside the rectangle (Figure 6.1.1b'), the reconstruction is shown in Figure 6.1.1c'. The result shows that the event was removed.

(2) Standard method

The forward process used exactly the same transformed parameters as those of the modified method. The resulting \( \tau-p \) gather is shown in Figure 6.1.1d and inverse without muting shown in Figure 6.1.1e.

The muting is also done by zeroing out the \( \tau-p \) data inside the rectangle (Figure 6.1.1d') and then performing inverse transform. The result is shown in Figure 6.1.1e'. The result shows that the event was not removed.
Figure 6.1.1 Application of slant stack and inversion: (a) test data with 15m trace spacing, the purpose is to mute the linear event L with velocity 800 m/s; (b) modified slant stack with proper initial p value; (c) inversion, using modified method; (b') zero out the data inside the rectangle; (c') inversion, using modified method, showing the event was removed; (d) standard slant stack; (e) inversion, using standard method; (d') zero out the data inside the rectangle; (e') inversion, using standard method, showing the event was not removed.
(b) Slant Stack by the Modified Method

(c) Reconstruction of Slant Stack by the Modified Method
Slant Stack by the Ordinary Method

Reconstruction of Slant Stack by the Ordinary Method
[6.1.2] Remove strong event

The input has 50 m trace spacing with one strong event and several other weak ones (the background) (Figure 6.1.2a). The background is shown in figure 6.1.2b. The purpose is to remove the strong event so that the background can be seen clearly.

(1) Modified method

The minimum p value was chosen so that the strong event located on the center of the transformed domain (Figure 6.1.2c). The reconstruction without muting is shown in Figure 6.1.2d.

Now zero out this event in τ-p domain (Figure 6.1.3c'). The reconstruction is shown in Figure 6.1.3d'. This event was completely removed and the background events is now standing out.

(2) Standard method

In this method, we used exactly the same transform parameters as those of the modified method. Figure 6.1.3e is the resulting τ-p data. Figure 6.1.3f is the reconstruction without muting.

Zero out the data inside the rectangle in Figure 6.1.3e'. The reconstruction is shown in Figure 6.1.3f'. The event was not removed.
Figure 6.1.2 Application of slant stack and inversion: (a) the test data with one strong event; (b) the background of the test data without the strong event, the purpose is to mute the strong linear event \( L \) in (a) to see if we can get the background of figure (b); (c) modified slant stack with proper initial \( p \) value; (d) inversion, using modified method; (c') zero out the data inside the rectangle; (d') inversion, using modified method, showing the event was removed; (e) standard slant stack with the same transformed parameters as those of the modified method; (f) inversion, using standard method; (e') zero out the data inside the rectangle; (f') inversion, using standard method, showing the event was not removed;
Original Model for Slant Stack and its Reconstruction

(a)

Background signals

(b)
Slant Stack by the Ordinary Method

(e')

Reconstruction of Slant Stack by the Ordinary Method

(f')
[6.2.] Trace Interpolation

During reconstruction, we do not have to use the same trace spacing as the forward transform. Thus, slant stack can be used for trace interpolation (Figure 6.2).

Spatial interpolation is similar to interpolation in frequency domain. If the frequency is aliased, there is no way to interpolate the frequency correctly. All interpolations suffer the same restriction as the corresponding data should not be aliased. So, in order for the data to be properly interpolated spatially, the input must not be spatially aliased.

To illustrate slant stack and inversion in the application of trace (spatial) interpolation, we take Figure 6.2a as test data. It consists of 11 traces with 15 m trace spacing and 512 sampling points in each trace with 4 ms sampling interval. Both standard and modified methods are used to do the testing. Their forward transforms were the same as all the examples shown previously, but in the inverse process, the trace spacing was set equal to 7.5 m and trace number equal to 22 (2x11).

(1) Modified method

By proper selection of the minimum p value, the forward result is shown in Figure 6.2b, which has concentrated $\tau$-p data. The interpolated data are shown on Figure 6.2c.

(2) Standard method

In this method, we also used the same transform parameters as those of the modified method. The resulting slant stack is shown on Figure 6.2d. The interpolated result is shown in Figure 6.2e.

Comparing Figures 6.2c and 6.2e, we see that in the modified method, the data were correctly interpolated. But in the standard method, the interpolated traces are not correct.
Figure 6.2 Application of slant stack and inversion: (a) test data, the purpose is
do trace interpolation with half trace spacing and double trace number; (b)
modified slant stack with proper initial p; (c) interpolation by modified
method, showing data were correctly interpolated; (d) standard slant stack
with same initial p; (e) interpolation by standard method, showing data were
not correctly interpolated
(b)

(c)
[6.3] Velocity analysis

The slant stack parameter $p$ is the slowness, which is the reciprocal of stacking velocity. By slant stack, the velocity of any events can be directly read from the transformed domain.
Chapter 7

SUMMARY

The difference between the standard method for slant stack and the modified method is that the ranges of ray parameter \( p \) are fixed for all frequencies for the standard method while in the modified method each frequency was redefined to have different \( p \) range \((=l/F_i\Delta x)\) based on the periodical characteristic of slant stack operator.

The advantages for the modified method are that: (1) we do not have to worry about what the lower and upper limits for the ray parameters are, as long as their ranges are kept in one period; (2) The only information needed to be supplied is the range of frequencies, which can easily be determined by looking at the frequency spectrum; (3) It may be faster to implement the modified method than the standard method if both methods make the same number of the total \( p \) traces; and (4) The minimum \( p \) value can be easily determined to suit different applications.
Appendix I

A X Window Project for A New Approach to Slant Stack

The User Guide

Appendix II is the complete source code for this thesis project, which was written in C. It is an interactive processing program involving slant stack and its inversion by standard method and by my modified method. The interactive part was written using OSF/Motif.

The code is placed in /home/nld/slnstk/Motif directory of RUF system. There is also a makefile. You may copy the source code and then type "make -f makefile". A executable file slnstk will be generated.

Before running the slnstk, you can set up motif resource by typing "xrdb AppResource", which is to set up the general appearance. You may change the resource file in the way you like.

Appendix II

A X Window Project for A New Approach to Slant Stack

The Complete Source Code (Motif version)
AppResource

! A Motif resource file for slant stack application
! general appearance and behavior defaults
!
slstk*allowShellResize:true
slstk*borderWidth:0
slstk*thickness:3
slstk*traversalOn:true
slstk*keyboardFocusPolicy:explicit
slstk*menuAccelerator:<key>KP_F2
slstk*XmRowColumn*background:skyblue
slstk*XmFrame*background:wheat
slstk*XmDrawingArea*background:white
slstk*shadowType:SHADOW_IN
slstk*highlightThickness:3
slstk*shadowThickness:3
!
slstk*fontList:variable
slstk*topShadowColor:white
slstk*bottomShadowColor:black
makefile

### X Project Makefile for the New Approach to the Slant Stack
### Lideng Ni 2/22/1991
###
### EXEC= slnstk
###
### DEFINES
###
###
### CC = gcc -D_NO_PROTO -g
###
### -g = Debugging info
### -O = optimize (cannot use -g then)
### CFLAGS= -I/usr/include
###
### Libraries
###
### LFLAGS= -L/usr/lib
###
### LIBS= -L/usr/lib -lxm -lxkt -lx11 -lm
###
###
### OBJECTS=
###
### save.o
### createStuff.o
### drawItem.o
### drawx.o
### eventx.o
### fourier.o
### HandleJob.o
### model.o
### plot_job.o
### textx.o
### slnstk.o
### xmMain.o
### xmMenu.o
###
### Command to compile and link objects together
###
### $(EXEC): $(OBJECTS)
### $(CC) -o $(EXEC) $(OBJECTS) $(CFLAGS) $(LIBS)
###
###
### end of make file
###
constant.h

#include <Xm/Xm.h>

#define NT 512
#define TR 31
#define DT .004
#define FMIN 10.
#define FMAX 60.
#define DIM 6*TR
#define OFFSET 50
#define PI 3.1415926
#define VStart 35
#define boxStart 30
#define pixmap1_width 800
#define pixmap1_height 200
#define pixmap2_width pixmap1_width
#define pixmap2_height 580
#define boxlen (pixmap2_width - 2*boxStart)

#define MUTE 1
#define PICK 0
#define XOR 99

extern Widget draw1, draw2;
extern GC draw_gc, undraw_gc, XorGC;
extern Pixmap pixmap1, pixmap2;

typedef struct _complex { float x, y; } complex;

void
RedrawPicture(Widget w, XExposeEvent *event,
   String *params, Cardinal *num_params),
processDown(Widget w, XButtonEvent *event,
   String *params, Cardinal *num_params),
processMotion(Widget w, XButtonEvent *event,
   String *params, Cardinal *num_params),
processUp(Widget w, XButtonEvent *event,
   String *params, Cardinal *num_params),
CreateMenus(Widget w), save_cb();

XFontStruct *initFont(Widget w, String s);
```c
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include "constant.h"

Pixmap pixmap1, pixmap2;
GC draw_gc, XorGC, undraw_gc;

Widget
CreateDrawWindow(Widget parent, int width, int height, String trans )
{
    IntArg args[10];
    int n;

    n = 0;
    XtSetArg(args[n], XmNwidth, width); n++;
    XtSetArg(args[n], XmNheight, height); n++;
    if (trans)
    {
        XtSetArg(args[n], XmNtranslations, XtParseTranslationTable(trans)); n++;
    }

    return XmCreateDrawingArea(parent, "DrawingArea", args, n);
}

void
RedrawPicture(Widget w, XExposeEvent *event, String *params, Cardinal *num_params)
{
    int x, y, width, height;
    Pixmap pixmap;
    GC gc;

    if (event)
    {
        /* drawing because of expose or button press */
        x = event->x;
        y = event->y;
        width = event->width;
        height = event->height;
    }
    else
    {
        /* always the whole window! */
        x = 0;
        y = 0;
        width = 10000;
        height = 10000;
    }

    if (w == draw1)  pixmap = pixmap1;
    else if (w == draw2)  pixmap = pixmap2;
    else return;

    XCopyArea(XtDisplay(w), pixmap, XtWindow(w),
        draw_gc, x, y, width, height, x, y);
}
static void
createGC( Widget w, GC *gc, XGCValues theGCValues, caddr_t type )
{
    *gc = XCreateGC( XtDisplay(w),
                     RootWindowOfScreen(XtScreen(w)),
                     GCForeground | GCBBackground,
                     &theGCValues );

    if ((int)type == XOR)
        XSetFunction( XtDisplay(w), *gc, GXxor );
}

unsigned long
COLOR(w, name)
Widget w;
char *name;
{
    XrmValue fromVal, toVal;
    unsigned long *pixel;

    fromVal.size = sizeof(char*),
    fromVal.addr = name;

    XtConvert(w, XmRString, &fromVal, XmRPixel, &toVal);
    pixel = (unsigned long*)toVal.addr;
    if( pixel == NULL )
    {
        fromVal.addr = XtDefaultBackground;
        XtConvert(w, XmRString, &fromVal, XmRPixel, &toVal);
        pixel = (unsigned long*)toVal.addr;
    }
    return(*pixel);
}

set_up_things(w)
Widget w;
{
    XGCValues values;
    int theScreen, theDepth;
    Display *display = XtDisplay(w);
    Pixel bg, fg;
    Arg args[2];
    int n;

    n = 0;
    XtSetArg(args[n], XmNbackground, &bg); n++;
    XtSetArg(args[n], XmNforeground, &fg); n++;
    XtGetValues(w, args, n);

    theScreen = DefaultScreen ( display );
    theDepth = DefaultDepth ( display, theScreen );
createStuff.c

pixmap1 = XCreatePixmap(display, RootWindowOfScreen(XtScreen(w)),
                        pixmap1_width, pixmap1_height, theDepth);

pixmap2 = XCreatePixmap(display,
                        RootWindowOfScreen(XtScreen(w)),
pixmap2_width, pixmap2_height, theDepth);

/** Create GC for draw **/
values.foreground = fg;
createGC ( w, &draw_gc, values, NULL);

/** Create GC for undraw **/
values.foreground = bg;
createGC ( w, &undraw_gc, values, NULL);

XFillRectangle(display, pixmap1, undraw_gc, 0, 0,
pixmap1_width, pixmap1_height);

XFillRectangle(display, pixmap2, undraw_gc, 0, 0,
pixmap2_width, pixmap2_height);

/** Create rubber band GC **/
values.foreground = COLOR(w, "black");
values.background = COLOR(w, "black");
createGC ( w, &XorGC, values, XOR );
}

void plot_node( Display *display, Drawable w, GC gc, int x, int y, String name )
{
    XDrawString( display, w, gc, x, y, name, strlen(name) );
}
drawitem.c

/*
** X Project for the "New Approach to Slant Stack"
**
** This software package is a interactive processing program, which involves
** in the slant stack and inversion by the standard method and by the modified
** method
**
**
**
Lideng Ni 2/22/1991
*/
#include <Xm/Xm.h>
#include "constant.h"

drawitem( Widget w, int theCommand, GC theGC, int x1, int y1, int x2, int y2 )
{
    /* -- function drawItem */

    if ((theCommand == MUTE) && ((x2 <= x1) || (y2 <= y1)))
        return;

    switch (theCommand)
    {
    case PICK:
        drawLine( XtDisplay(w), XtWindow(w), theGC, x1, y1, x2, y2 );
        drawLine( XtDisplay(w), pixmap2, theGC, x1, y1, x2, y2 );
        break;

    case MUTE:
        drawRectangle( XtDisplay(w), XtWindow(w), theGC, x1, y1,
                       x2 - x1, y2 - y1 );
        drawRectangle( XtDisplay(w), pixmap2, theGC, x1, y1,
                       x2 - x1, y2 - y1 );
        break;
    }
}
/* -- function drawIt */
drawx.c

/*
 * X Project for the "New Approach to Slant Stack"
 * This software package is a interactive processing program, which involves
 * in the slant stack and inversion by the standard method and by the modified
 * method
 * Lideng Ni 2/22/1991
 */

#include <Xm/Xm.h>

drwLine(Display *display, Drawable w, GC theGC, int x1, int y1, int x2, int y2)
{
    /* -- function drawLine */
    XDrawLine(display, w, theGC, x1, y1, x2, y2);
    /* -- function drawLine */
}

drawLines(Display *display, Drawable w, GC theGC, XPoint points, int n)
{
    XDrawLines(display, w, theGC, points, n, CoordModeOrigin);
    /* -- function multiple drawLines */
}

drawRectangle(Display *display, Drawable w, GC theGC, int x, int y, int width,
int height)
{
    /* -- function drawRectangle */
    XDrawRectangle(display, w, theGC, x, y, width, height);
    /* -- function drawRectangle */
}

fillpolygon
/**
 * Fills the framed outline of a polygon. It is bounded
 * by a curve drawn by drawLines
 */
fillpolygon ( Display *display, Drawable w, GC theGC, XPoint points, int    numberOfPoints, float avg1 )
{
    register    int    avg = avg1 + 1;
    XPoint fillPoints[512];
    register    int    i, j = 0, l = 0;

    if ( points[0].x >= avg )
    {
        fillPoints[j].x = avg;
        fillPoints[j].y = points[0].y;
        l = j = 1;
    }

    for ( i = l; i < numberOfPoints - 1; i++ )
    {
        if ( points[i].x < avg )
        {
            continue;
        }
        else if ( points[i].x > avg && points[i-1].x < avg )
        {
            fillPoints[j].x = avg;
            fillPoints[j].y = points[i-1].y;
            ++j;
        }
        else if ( points[i].x >= avg && points[i+1].x < avg )
        {
            fillPoints[j].x = avg;
            fillPoints[j].y = points[i+1].y;
            j++;
            XFillPolygon ( display, w, theGC, fillPoints, j,
                            Convex, CoordModeOrigin );
            j = 0;
        }
        else
        {
            fillPoints[j].x = points[i].x;
            fillPoints[j].y = points[i].y;
            j++;
        }
    }
}

/*
**    end of file drawx.c
*/
#include <Xm/Xm.h>
#include "constant.h"
#define NOT_DRAW_ON 0
#define DRAW_ON 1

static int draw_on = NOT_DRAW_ON;

static int currentX, currentY, lastX, lastY;
static float traceIncr = boxlen/(TR-1.);
int DrawCommand = PICK;
int traceBegin, traceEnd, traceNumber, sampleBegin, sampleEnd;
int mutex1[30], mutex2[30], mutexY[30], mutex2[30], muteTimes = 0;

void processDown(Widget w, XButtonEvent *DrawEvent, String *params,
                 Cardinal *num_params)
{
    draw_on = DRAW_ON;
    lastX = DrawEvent->x;
    lastY = DrawEvent->y;
    currentX = lastX;
    currentY = lastY;

    drawitem ( w, DrawCommand, XorGC, lastX, lastY, currentX, currentY);
}

void processMotion(Widget w, XButtonEvent *DrawEvent, String *params,
                   Cardinal *num_params)
{
    if (draw_on != DRAW_ON) return;
    drawitem ( w, DrawCommand, XorGC, lastX, lastY, currentX, currentY);
    currentX = DrawEvent->x;
    currentY = DrawEvent->y;

    drawitem ( w, DrawCommand, XorGC, lastX, lastY, currentX, currentY);
}

void processUp(Widget w, XButtonEvent *DrawEvent, String *params,
               Cardinal *num_params)
{
    if (draw_on != DRAW_ON) return;
    currentX = DrawEvent->x;
    currentY = DrawEvent->y;
    drawitem ( w, DrawCommand, XorGC, lastX, lastY, currentX, currentY);
if (DrawCommand == MUTE)
{
    drawitem (w, DrawCommand, XorGC, lastX, lastY, currentX, currentY);
    mutex1[muteTimes] = lastX;
    mutex2[muteTimes] = currentX;
    mutex1[muteTimes] = lastY;
    mutex2[muteTimes] = currentY;
    muteTimes++;
}
else if (DrawCommand == PICK)
{
    traceBegin = round ((lastX - boxStart) / traceIncr);
    sampleBegin = lastY - VStart;
    traceEnd = round ((currentX - boxStart) / traceIncr);
    sampleEnd = currentY - VStart;
    traceNumber = round ((traceBegin + traceEnd) / 2);
    muteTimes = 0;
}
currentX = currentY = lastX = lastY = 0;
draw_on = NOT_DRAW_ON;
/*
**  X Project for the "New Approach to Slant Stack"
**
**  This software package is a interactive processing program, which involves
**  in the slant stack and inversion by the standard method and by the modified
**  method
**
**  These subroutines are used to do FFT and IFFT
**
** Ldeng Ni  2/22/1991
*/

#include "constant.h"
#include <math.h>

/***
*****
*****  fft is to perform forward Fourier transform while ifft is to inverse transform
*****
*****  If the input data are matrix, then fft (or ifft) will perform transform along
*****  each column. All input data will be padded with 0 to make the length be power
*****  of 2 if the input data are not power of 2.
*****
*****/

fft ( numberOfColumn, numberOfRow, ftmat )
complex ftmat[NT][NT];
int    numberOfColumn, numberOfRow;
{
    int i;

    for ( i = 0; i < numberOfColumn; i++ )
        fftvector ( numberOfRow, ftmat[i], 1. );
}

ifft ( numberOfColumn, numberOfRow, ifftmat )
complex ifftmat[NT][NT];
int numberOfColumn, numberOfRow;
{
    int i;

    for ( i = 0; i < numberOfColumn; i++ )
        fftvector ( numberOfRow, ifftmat[i], -1. );
}
```c
fftvector ( length, coeff, type )
complex coeff[];
int length;
float type;
{
    int i, j, m, l, increment;
    int power2 = 1;
    float fac, arg;
    complex temp;
    complex mid;

    j = 0;

    if ( type == 1 ) fac = 1.;
    else fac = 1./length;

    while ( power2 < NT ) power2 *= 2;

    for (i = length; i<power2; i++)
    {
        coeff[i].x = 0;
        coeff[i].y = 0;
    }

    length = power2;

    for ( i=0; i < length; i++ )
    {
        if ( i <= j )
        {
            temp.x = coeff[j].x*fac;
            temp.y = coeff[j].y*fac;
            coeff[j].x = coeff[j].x*fac;
            coeff[j].y = coeff[j].y*fac;
            coeff[j] = temp;
        }
        m = length/2;
        again:
        if ( j > m-1 )
        {
            j = j-m;
            m = m /2;
            if (m >= 1) goto again;
        }
        j = j+m;
    }

    l = 1;

    step: increment = 2*l;

    for ( m =0; m < l; m++)
    {
        arg = PI*type*m/(l*1.);
        mid.x = cos(arg);
        mid.y = sin(arg);
    }
```
for ( i = m; i < length; i += increment) 
{
    temp.x = mid.x*coeff[i+1].x - mid.y*coeff[i+1].y;
    temp.y = mid.x*coeff[i+1].y + mid.y*coeff[i+1].x;
    coeff[i+1].x = coeff[i].x-temp.x;
    coeff[i+1].y = coeff[i].y-temp.y;
    coeff[i].x = coeff[i].x + temp.x;
    coeff[i].y = coeff[i].y + temp.y;
}

l = increment;

if ( l < length ) goto step;

for ( i = 1; i < length/2; i++)
{
    temp = coeff[length-i];
    coeff[length-i] = coeff[i];
    coeff[i] = temp;
}

return;

/*
  **
  end of file fourier.c
 */
/*
 **  X Project for the "New Approach to Slant Stack"
 **
 **  This software package is a interactive processing program, which involves
 **  in the slant stack and inversion by the standard method and by the modified
 **  method
 **
 **  Lideng Ni    2/22/1991
 */
#include <stdio.h>
#include <Xm/Xm.h>
#include <math.h>
#include "constant.h"

extern float new_dp, old_dp, old_p1, new_p1, detf;
extern int oldnp, newnp, fstart, fend, Yes;

doingOldSST() /* Standard forward tau-p transform */
{
    extern complex gt[][NT];
    complex input[TR][NT];
    register int i, j;

    dtcopy ( gt, input, sizeof(complex)*NT*TR );
    oldtaup ( input );
}

/* end of doingOldSST */
doingOldInvSST( int *x1, int *y1, int *x2, int *y2, int muteTime )
{
    register int i, j, k;
    float oldnpIncr;
    complex input[DIM][NT];
    extern complex oldtp[][NT];

    dtcopy ( oldtp, input, sizeof(complex)*NT*oldnp );

    if ( muteTime >= 1 )
    {
        /* Muting */
        oldnp = 6*TR;
        oldnpIncr = boxlen/(oldnp - 1.);
        for ( k = 0; k < muteTime; k++ )
        {
            x1[k] = ( x1[k]-boxStart )/oldnpIncr;
            x2[k] = ( x2[k]-boxStart )/oldnpIncr;
            x1[k] = x1[k] >= 0 ? x1[k] : 0;
            x2[k] = x2[k] >= 0 ? x2[k] : 0;
            for ( i = x1[k]; i < x2[k]; i++ )
            {
            }
        }
    }
}
HandleJob.c

    for ( j = y1[k]-VStart; j < y2[k]-VStart; j++ )
    {
        input[i][j].x = 0;
        input[i][j].y = 0;
    }
}

oldxt ( input );

} /* end of doingOldInvSST */

doingNewSST( ) /* Modified forward tau-p transform */
{
    extern complex gt[][NT];
    complex input[TR][NT];
    register int i, j;
    dcopy ( gt, input, sizeof(complex)*NT*TR );

    newtaup ( input );
} /* end of doingNewSST */


doingNewInvSST( int *x1, int *y1, int *x2, int *y2, int muteTime )
{
    complex input[DIM][NT];
    extern complex newtp[][NT];
    float newnplncre;
    register int i, j, k;
    dcopy ( newtp, input, sizeof(complex)*NT*newnp );

    if ( muteTime >= 1 )
    {
        /* Muting */
        newnplncre = boxlen / ( newnp - 1.);
        for ( k = 0; k <= muteTime - 1; k++ )
        {
            x1[k] = ( x1[k]-boxStart )/ newnplncre;
            x2[k] = ( x2[k]-boxStart )/ newnplncre;
            x1[k] = x1[k] >= 0 ? x1[k] : 0;
            x2[k] = x2[k] >= 0 ? x2[k] : 0;
            for ( i = x1[k]; i < x2[k]; i++ )
            {
                for ( j = y1[k]-VStart; j < y2[k]-VStart; j++ )
                {
                    input[i][j].x = input[i][j].y = 0;
                }
            }
        }
    }

    newxt ( input );
} /* end of doingNewInvSST */
HandleJob.c

dtcopy ( a, b, len )       /* Copy from a into b */
char      *a, *b;
int       len;
{
    bcopy ( a, b, len );
} /* end of dtcopy */

/*
 ** End of file eventjob.c
 */
/*
 ** X Project for the "New Approach to Slant Stack"
 **
 ** This software package is a interactive processing program, which involves
 ** in the slant stack and inversion by the standard method and by the modified
 ** method
 **
 ** Lideng Ni  2/22/1991
 */

#include <math.h>
#include "constant.h"

extern complex gt[][NT];

#define ALPHA 5000
#define V0 800
#define V1 2000
#define V2 3000
#define H1 1000
#define H2 1000

void gen1()
{
    gt[TR/2][NT/2].x = 1;
    return;
}

void gen2()
{
    register int i, j;
    float t3, t31, t32, t;
    float a3, b3;
    float scale;

    scale = pow(1.*OFFSET,3./4);
    t31 = pow(2.*H1, 2.);
    t32 = pow(1.*OFFSET, 2.);
    i = 4;
    t3 = sqrt(pow(1.*i, 2.)*t32 + t31)/V1;
for (j=0; j<NT; j++)
{
    t = j*1.*DT;
    a3 = 1.-2.*ALPHA*pow((t - t3),2.);
    b3 = exp(-ALPHA*pow((t - t3),2.));
    gt[i][j].x = scale*(a3*b3);
    gt[i][j].y = 0;
}
return;

gen3()
/**
  gen3 generates 4 events, first one is air wave with velocity 800 m/s, second one is
direct arrival with velocity 2000 m/s, third one is reflection from 1000m depth
with velocity 2000 m/s, and the last one is another reflection from 2000 depth with
velocity 3000 m/s
**/
{
    register int i, j;
    float x, t, t1, t2, t3, t4, t11, t21, t31, t32;
    float xx, xx1, xx2, t41, t42;
    float a1, b1, a2, b2, a3, b3, a4, b4;
    float scale;

    scale = pow(1.*OFFSET,3./4);
    t11 = 1.*OFFSET/V0;
    t21 = 1.*OFFSET/V1;
    t31 = pow(2.*H1,2.);
    t32 = pow(1.*OFFSET,2.);
    xx = 1.*H1/(H1+H2)*(OFFSET/2.);
    xx1 = pow (( OFFSET / 2. - xx ), 2.);
    xx2 = pow(xx,2.);
    t41 = pow(1.*H1,2.);
    t42 = pow(1.*H2,2.);

    for (i=0; i<TR; i++)
    {
        t1 = (i+1.)*t11;
        t2 = (i+1.)*t21;
        t3 = sqrt(pow(1.*i, 2.)*t32 + t31)/V1;
        t4 = 2*(sqrt(pow(1.*i,2.)*xx2 +t41)/V1 +
              sqrt(pow(1.*i,2.)*xx1 +t42)/V2);

        for (j=0; j<NT; j++)
        {
            t = j*1.*DT;
            a1 = 1 - 2.*ALPHA*pow((t - t1),2.);
model.c

b1 = exp(-ALPHA*pow((t - t1),2));
a2 = 1-2*ALPHA*pow((t - t2),2);
b2 = exp(-ALPHA*pow((t - t2),2));
a3 = 1-2*ALPHA*pow((t - t3),2);
b3 = exp(-ALPHA*pow((t - t3),2));
a4 = 1-2*ALPHA*pow((t - t4),2);
b4 = exp(-ALPHA*pow((t - t4),2));

gt[i][j].x = scale*(a1*b1 + a2*b2 + a3*b3 + a4*b4);
gt[i][j].y = 0;
}
return;
}

gen4()
/**
 *** gen4 generates a sinusoid at one trace with frequency 40 Hz, all other traces are
 *** zeros.
 **/
{
    register int i, j;

    for ( i = 0; i < TR; i++ )
        for ( j = 0; j < NT; j++ )
            gt[i][j].x = gt[i][j].y = 0;

    for ( j = 0; j < NT; j++ )
    {
        gt[4][j].x = 1.0*OFFSET/3.0*sin(2.0*PI*40*j*DT);
        gt[4][j].y = 0;
    }
}
gen5()
/**
 *** input has 4 traces, first one consists of a single frequency (40 Hz), second one
 *** consists of only 20 Hz, third one 60 Hz, the last one combines all the previous
 *** frequency.
 **/
{
    register int i, j;

    for ( i = 0; i < TR; i++ )
        for ( j = 0; j < NT; j++ )
            gt[i][j].x = gt[i][j].y = 0;
for (j = 0; j < NT; j++)
{
    gt[5][j].x = 2.5*OFFSET/3.*sin(2.*PI*20*j*DT);
    gt[8][j].x = 2.5*OFFSET/3.*sin(2.*PI*40*j*DT);
    gt[11][j].x = 2.5*OFFSET/3.*sin(2.*PI*60*j*DT);
    gt[14][j].x = 1.*OFFSET/3.*sin(2.*PI*20*j*DT) +
                  1.*OFFSET/3.*sin(2.*PI*40*j*DT) +
                  1.*OFFSET/3.*sin(2.*PI*60*j*DT);
}

}

g6()  
/**
 ** gen6 generates 4 events, first one is air wave with velocity 800 m/s, second one is
 ** direct arrival with velocity 2000 m/s, third one is reflection from 1000m depth
 ** with velocity 2000 m/s, and the last one is another reflection from 2000 depth with
 ** velocity 3000 m/s
 ***/
{
    register int i, j, k;
    float x, t, t1, t2, t3, t4, t11, t21, t31, t32;
    float xx, xx1, xx2, t41, t42;
    float a1, b1, a2, b2, a3, b3, a4, b4;
    float scale;

    scale = pow(1.*OFFSET, 3./4);

    t11 = 1.*OFFSET/V0;
    t21 = 1.*OFFSET/V1;
    t31 = pow(2.*H1, 2.);
    t32 = pow(1.*OFFSET, 2.);
    xx = 1.*H1/(H1+H2)*(OFFSET/2.);
    xx1 = pow((OFFSET/2. - xx), 2.);
    xx2 = pow(xx, 2.);
    t41 = pow(1.*H1, 2.);
    t42 = pow(1.*H2, 2.);

    for (i= (TR-1)/2; i< TR; i++)
    {
        t1 = (i+1.)*t11;
        t2 = (i+1.)*t21;
        t3 = sqrt(pow(1.*i, 2.)*t32 + t31)/V1;
        t4 = 2*(sqrt(pow(1.*i, 2.)*xx2 +t41)/V1 +
                     sqrt(pow(1.*i, 2.)*xx1 + t42)/V2);

        for (j=0; j<NT; j++)
        {

model.c

t = j*1.*DT;
a1 = 1 - 2*ALPHA*pow((t - t1),2.);
b1 = exp(-ALPHA*pow((t - t1),2.));
a2 = 1-2*ALPHA*pow((t - t2),2.);
b2 = exp(-ALPHA*pow((t - t2),2.));
a3 = 1-2*ALPHA*pow((t - t3),2.);
b3 = exp(-ALPHA*pow((t - t3),2.));
a4 = 1-2*ALPHA*pow((t - t4),2.);
b4 = exp(-ALPHA*pow((t - t4),2.));

g[i][j].x = scale*(a1*b1 + a2*b2 + a3*b3 + a4*b4);
g[i][j].y = 0;
g[TR-i-1][j] = g[i][j];
}
}
return;
}

gen7()
/**
*** gen7 generates 4 events, first one is air wave with velocity 800 m/s, second one is
*** direct arrival with velocity 2000 m/s, third one is reflection from 1000m depth
*** with velocity 2000 m/s, and the last one is another reflection from 2000 depth with
*** velocity 3000 m/s. There are a lot of random noise. This model is used to test the
*** effect of slant stack filtering.
**/
{
    register int    i, j;
    float    x, t, t1, t2, t3, t4, t11, t21, t31, t32;
    float    xx, xx1, xx2, t41, t42;
    float    a1, b1, a2, b2, a3, b3, a4, b4, an, bn;
    float    scale;

    scale = pow(1.*OFFSET,3./4);
    t11 = 1.*OFFSET/V0;
    t21 = 1.*OFFSET/V1;
    t31 =  pow(2.*H1, 2.);
    t32 = pow(1.*OFFSET, 2.);
    xx = 1.*H1/(H1+H2)*(OFFSET/2.);
    xx1 = pow( ( OFFSET / 2. - xx ), 2.);
    xx2 = pow(xx, 2.);
    t41 = pow(1.*H1,2.);
    t42 = pow(1.*H2,2.);

    for (i=0; i<TR; i++)
    {
        t1 = (i+1.)*t11;
        t2 = (i+1.)*t21;
        t3 = sqrt(pow(1.*i, 2.)*t32 + t31)/V1;
        t4 = ...
    }
}
\[ t_4 = 2^* (\sqrt{\text{pow}(1.0, i, 2.0) * xx2 + t41}) \div V1 + \sqrt{\text{pow}(1.0, i, 2.0) * xx1 + t42}) \div V2); \]

for \( j = 0; j < NT; j++ \)
{
    \[ t = j * 1.0 \text{DT}; \]
    \[ a1 = 1 - 2^* \text{ALPHA}^\text{pow}(t - t1), 2.0); \]
    \[ b1 = \exp(-\text{ALPHA}^\text{pow}(t - t1), 2.0); \]
    \[ a2 = 1 - 2^* \text{ALPHA}^\text{pow}(t - t2), 2.0); \]
    \[ b2 = \exp(-\text{ALPHA}^\text{pow}(t - t2), 2.0); \]
    \[ a3 = 1 - 2^* \text{ALPHA}^\text{pow}(t - t3), 2.0); \]
    \[ b3 = \exp(-\text{ALPHA}^\text{pow}(t - t3), 2.0); \]
    \[ a4 = 1 - 2^* \text{ALPHA}^\text{pow}(t - t4), 2.0); \]
    \[ b4 = \exp(-\text{ALPHA}^\text{pow}(t - t4), 2.0); \]
    \[ an = 1 - 2^* 5^* \text{ALPHA}^\text{pow}(t - j \text{DT}), 2.0); \]
    \[ bn = \exp(-5^* \text{ALPHA}^\text{pow}(t - j \text{DT}), 2.0); \]
    \[ \text{gt[i][j].x} = \text{scale} \ast (a1 \ast b1 + a2 \ast b2 + a3 \ast b3 + a4 \ast b4 + .2 \ast an \ast bn + .5 \ast \sin(2.0 \text{PI} \ast 3^* \text{DT}) + .5 \ast \sin(2.0 \text{PI} \ast 6^* \text{DT}) + .5 \ast \sin(2.0 \text{PI} \ast 90^* \text{DT}) + .5 \ast \sin(2.0 \text{PI} \ast 85^* \text{DT}) + .5 \ast \sin(2.0 \text{PI} \ast 100^* \text{DT})); \]
    \[ \text{gt[i][j].y} = 0; \]
}

return;

gen8()

/**
*** gen8 generates very complicate linear and hyperbolic events which interfere with each other.
***
***/
{
    \[ i, j; \]
    \[ x, t, l1, l2, l3, t4, t11, t21, t31, t32, xx, xx1, xx2, t41, t42; \]
    \[ a10, b10, a20, b20, a30, b30, a40, b40; \]
    \[ a11, b11, a21, b21, a31, b31, a41, b41; \]
    \[ a12, b12, a22, b22, a32, b32, a42, b42; \]
    \[ a13, b13, a23, b23, a33, b33, a43, b43; \]
    \[ \text{scale} = \text{pow}(1.0 \text{OFFSET}, 3.0) / 4); \]
    \[ t11 = 1.0 \text{OFFSET} \div V0; \]
    \[ t21 = 1.0 \text{OFFSET} \div V1; \]
    \[ t31 = \text{pow}(2.0 \text{H1}, 2.0); \]
    \[ t32 = \text{pow}(1.0 \text{OFFSET}, 2.0); \]
    \[ xx = 1.0 \text{H1} / (\text{H1} + \text{H2}) \ast \text{OFFSET} / 2.0; \]
    \[ xx1 = \text{pow}((\text{OFFSET} / 2.0 \text{ - xx}) \ast 2.0); \]
\[
xx2 = \text{pow}(xx, 2.);
\]
\[
t41 = \text{pow}(1.*H1, 2.);
\]
\[
t42 = \text{pow}(1.*H2, 2.);
\]

for (i=0; i<TR; i++)
{
  t1 = (i+1.)*t11;
  t2 = (i+1.)*t21;
  t3 = sqrt(pow(1.*i, 2.)*t32 + t31)/V1;
  t4 = 2*(sqrt(pow(1.*i, 2.)*xx2 +t41)/V1 +
    sqrt(pow(1.*i, 2.)*xx1 + t42)/V2);

  for (j=0; j<NT; j++)
  {
    t = j*1.*DT;
    a10 = 1 - 2*ALPHA*\text{pow}((t - t1), 2.);
    b10 = \text{exp}(-ALPHA*\text{pow}((t - t1), 2.));
    a20 = 1-2*ALPHA*\text{pow}((t - t2), 2.);
    b20 = \text{exp}(-ALPHA*\text{pow}((t - t2), 2.));
    a30 = 1-2*ALPHA*\text{pow}((t - t3), 2.);
    b30 = \text{exp}(-ALPHA*\text{pow}((t - t3), 2.));
    a40 = 1-2*ALPHA*\text{pow}((t - t4), 2.);
    b40 = \text{exp}(-ALPHA*\text{pow}((t - t4), 2.));

    a11 = 1 - 2*ALPHA*\text{pow}((t - .5*t1), 2.);
    b11 = \text{exp}(-ALPHA*\text{pow}((t - .5*t1), 2.));
    a21 = 1 - 2*ALPHA*\text{pow}((t - .5*t2), 2.);
    b21 = \text{exp}(-ALPHA*\text{pow}((t - .5*t2), 2.));
    a31 = 1 - 2*ALPHA*\text{pow}((t - .5*t3), 2.);
    b31 = \text{exp}(-ALPHA*\text{pow}((t - .5*t3), 2.));
    a41 = 1 - 2*ALPHA*\text{pow}((t - .5*t4), 2.);
    b41 = \text{exp}(-ALPHA*\text{pow}((t - .5*t4), 2.));

    a12 = 1 - 2*ALPHA*\text{pow}((t - 1.5*t1), 2.);
    b12 = \text{exp}(-ALPHA*\text{pow}((t - 1.5*t1), 2.));
    a22 = 1 - 2*ALPHA*\text{pow}((t - 1.5*t2), 2.);
    b22 = \text{exp}(-ALPHA*\text{pow}((t - 1.5*t2), 2.));
    a32 = 1 - 2*ALPHA*\text{pow}((t - 1.5*t3), 2.);
    b32 = \text{exp}(-ALPHA*\text{pow}((t - 1.5*t3), 2.));
    a42 = 1 - 2*ALPHA*\text{pow}((t - 1.5*t4), 2.);
    b42 = \text{exp}(-ALPHA*\text{pow}((t - 1.5*t4), 2.));

    a13 = 1 - 2*ALPHA*\text{pow}((t - .25*t1), 2.);
    b13 = \text{exp}(-ALPHA*\text{pow}((t - .25*t1), 2.));
    a23 = 1 - 2*ALPHA*\text{pow}((t - .25*t2), 2.);
    b23 = \text{exp}(-ALPHA*\text{pow}((t - .25*t2), 2.));
    a33 = 1 - 2*ALPHA*\text{pow}((t - .25*t3), 2.);
    b33 = \text{exp}(-ALPHA*\text{pow}((t - .25*t3), 2.));
    a43 = 1 - 2*ALPHA*\text{pow}((t - .25*t4), 2.);
    b43 = \text{exp}(-ALPHA*\text{pow}((t - .25*t4), 2.));
  
  
}
model.c

gt[i][j].x = scale*(a10*b10 + a20*b20 + a30*b30 + a40*b40 +
a11*b11 + a21*b21 + a31*b31 + a41*b41 +
a12*b12 + 10.*a22*b22 + a32*b32 + a42*b42 +
a13*b13 + a23*b23 + a33*b33 + a43*b43);
gt[i][j].y = 0;
}
}
return;
}

gen9()
/**
*** input has 4 traces, first one consists of a single frequency (40 Hz), second one
*** consists of only 20 Hz, third one 60 Hz, the last one combines all the previous
*** frequency.
***/
{
    register int i, j;

    for (i = 0; i < TR; i++)
        for (j = 0; j < NT; j++)
            
            gt[i][j].x = gt[i][j].y = 0;
        
    for (j = 0; j < NT; j++)
        
        
        gt[1][j].x = sin(2.*PI*5.*j*DT) +
                   sin(2.*PI*40.*j*DT) +
                   sin(2.*PI*60.*j*DT);
        gt[4][j].x = 2.5*sin(2.*PI*40.*j*DT);
        gt[7][j].x = 2.5*sin(2.*PI*20.*j*DT);
        gt[10][j].x = 2.5*sin(2.*PI*60.*j*DT);
        gt[13][j].x = sin(2.*PI*20.*j*DT) +
                      sin(2.*PI*40.*j*DT) +
                      sin(2.*PI*60.*j*DT);
        gt[16][j].x = 2.5*sin(2.*PI*5.*j*DT);
        gt[19][j].x = 2.5*sin(2.*PI*30.*j*DT);
        gt[22][j].x = 2.5*sin(2.*PI*70.*j*DT);
        gt[25][j].x = 2.5*sin(2.*PI*85.*j*DT);
        gt[28][j].x = 2.5*sin(2.*PI*115.*j*DT);
    
}

gen10()
{
    register int i, j;
    float dis, t;

for ( i = 0; i < TR; i++ )
{
    dis = OFFSET*i*1.;
    for ( j = 0; j < NT; j++ )
    {
        t = j*DT*1.;
        gt[i][j].x = sin (2.*PI*80*(t-1./1000*dis));
        gt[i][j].y = 0.;
    }
}
gen11()
{
    register int i, j;
    float dis, t;
    for ( i = 0; i < TR; i+=2 )
    {
        dis = OFFSET*i*1.;
        for ( j = 0; j < NT; j++ )
        {
            t = j*DT*1.;
            gt[i][j].x = sin (2.*PI*40*(t-1./1000*dis));
            gt[i][j].y = 0.;
        }
    }
}
gen12()
{
    register int i, j;
    float dis, t;
    for ( i = 0; i < TR; i+=4 )
    {
        dis = OFFSET*i*1.;
        for ( j = 0; j < NT; j++ )
        {
            t = j*DT*1.;
            gt[i][j].x = sin (2.*PI*40*(t-1./1000*dis));
            gt[i][j].y = 0.;
        }
    }
}
/*
**     end of file model.c
*/
/\* X Project for the "New Approach to Slant Stack" \*
\* This software package is a interactive processing program, which involves \* in the slant stack and inversion by the standard method and by the modified \* method \*/

Lideng Ni 2/22/1991

#include <stdio.h>
#include <Xm/Xm.h>
#include <math.h>
#include "constant.h"

static complex originalgt[NT];
static complex myinvgt[NT];
static complex wronginvgt[NT];

static XFontStruct *fontStruct;
char num_str[15];

findFFTWindow ()
{
    extern int traceNumber, sampleBegin, sampleEnd;
    float traceIncre = boxlen(TR+1.);

    sampleBegin = sampleBegin >= 0 ? sampleBegin : 0;
    sampleEnd = sampleEnd <= NT ? sampleEnd : NT;

    if ( traceNumber < 0 || traceNumber >= TR ) return;
    doingFFT( traceNumber, sampleBegin, sampleEnd );

    plot_FFT( draw_gc );
}

doingFFT( traceNumber, sampleBegin, sampleEnd )

int traceNumber, sampleBegin, sampleEnd;
{
    register int i;
    extern complex gt[][NT];
    extern complex newinvgt[][NT];
    extern complex oldinvgt[][NT];

    if ( sampleBegin > sampleEnd )
    {
        i = sampleEnd;
        sampleEnd = sampleBegin;
        sampleBegin = i;
    }
for ( i = sampleBegin; i < sampleEnd; i++)
{
    originalgt[i-sampleBegin] = gt[traceNumber][i];
    myinvgt[i-sampleBegin] = newinvgt[traceNumber][i];
    wronginvgt[i-sampleBegin] = oldinvgt[traceNumber][i];
}
if (sampleEnd - sampleBegin < NT )
{
    for ( i = sampleEnd - sampleBegin; i < NT; i++)
    {
        originalgt[i].x = 0.;
        originalgt[i].y = 0.;
        myinvgt[i].x = 0.;
        myinvgt[i].y = 0.;
        wronginvgt[i].x = 0.;
        wronginvgt[i].y = 0.;
    }
}
fftvector ( NT, originalgt, 1. );
fftvector ( NT, myinvgt, 1. );
fftvector ( NT, wronginvgt, 1. );

plot_FFT( whichGC )
GC whichGC;
{
    float def;
    XPoint orgpoints[NT], mypoints[NT], wrongpoints[NT];
    register float a = 0.;
    register float b = 0.;
    register float c = 0.;
    register int i;
    int x1, x2, x3, b_width, b_height, b_dis;
    int b_startx, b_starty, myTitleWidth, spHeight;
    register float temp;
    char *title, *subtitle;

    *******************************************************************************/
    Three boxes to hold the three spectrums
    (b_startx, b_starty): first box starting point
    b_width, b_height: box size
    b_dis: distance between boxes
    spHeight: spectrum height inside the boxes < b_height
    *******************************************************************************/
    b_width = 250*pixmap1_width/900;
    b_height = pixmap1_height - 100;
    b_dis = 50*pixmap2_width/900;
    b_startx = 20*pixmap2_width/900;
b_starty = 50;
spHeight = b_height - 15;

def = 1/(NT*DT);
for (i = 0; i < NT/2 + 1; i++)
{
    temp = absolute (originalgt[i]);
    orgpoints[i].x = def*i*b_width*2*DT + b_startx;
    a = temp > a ? temp : a;

    temp = absolute (myinvgt[i]);
    mypoints[i].x = def*i*b_width*2*DT + b_startx + b_width + b_dis;
    b = temp > b ? temp : b;

    temp = absolute (wronginvgt[i]);
    wrongpoints[i].x = def*i*b_width*2*DT + b_startx +
        2*b_width + 2*b_dis;
    c = temp > c ? temp : c;
}

a = a==0. ? 1. : spHeight/a;
b = b==0. ? 1. : spHeight/b;
c = c==0. ? 1. : spHeight/c;
for (i = 0; i < NT/2 + 1; i++)
{
    orgpoints[i].y = -absolute (originalgt[i])*a + b_starty + b_height;
    mypoints[i].y = -absolute (myinvgt[i])*b + b_starty + b_height;
    wrongpoints[i].y = -absolute (wronginvgt[i])*c + b_starty + b_height;
}

XClearWindow(XtDisplay(draw1), XtWindow(draw1));
XFillRectangle(XtDisplay(draw1), pixmap1, undraw_gc, 0, 0, 10000, 5000);

fontStruct = initFont (draw1, "8x13B");
drawRectangle (XtDisplay(draw1), XtWindow(draw1), whichGC, b_startx, b_starty, b_width, b_height);
drawRectangle (XtDisplay(draw1), XtWindow(draw1), whichGC, b_startx + b_width + b_dis, b_starty, b_width, b_height);
drawRectangle (XtDisplay(draw1), XtWindow(draw1), whichGC, b_startx + 2*b_width + 2*b_dis, b_starty, b_width, b_height);
drawRectangle (XtDisplay(draw1), pixmap1, whichGC, b_startx, b_starty, b_width, b_height);
drawRectangle (XtDisplay(draw1), pixmap1, whichGC, b_startx + b_width + b_dis, b_starty, b_width, b_height);
drawRectangle (XtDisplay(draw1), pixmap1, whichGC, b_startx + 2*b_width + 2*b_dis, b_starty, b_width, b_height);

for (i = 0; i <= 125; i += 25)
{
    x1 = b_width*2*DT*i + b_startx;
    x2 = x1 + b_width + b_dis;
}
plot_job.c

x3 = x2 + b_width + b_dis;
sprintf(num_str, "%3d", i);
drawLine(XtDisplay(draw1), XtWindow(draw1), whichGC, x1,  
   b_starty + b_height + 5, x1, b_starty + b_height);
drawLine(XtDisplay(draw1), XtWindow(draw1), whichGC, x2,  
   b_starty + b_height + 5, x2, b_starty + b_height);
drawLine(XtDisplay(draw1), XtWindow(draw1), whichGC, x3,  
   b_starty + b_height + 5, x3, b_starty + b_height);
drawLine(XtDisplay(draw1), pixmap1, whichGC, x1, b_starty +  
   b_height + 5, x1, b_starty + b_height);
drawLine(XtDisplay(draw1), pixmap1, whichGC, x2, b_starty +  
   b_height + 5, x2, b_starty + b_height);
drawLine(XtDisplay(draw1), pixmap1, whichGC, x3, b_starty +  
   b_height + 5, x3, b_starty + b_height);

x1 = i == 0 ? x1 - 5 : x1;
x2 = i == 0 ? x2 - 5 : x2;
x3 = i == 0 ? x3 - 5 : x3;
plot_node(XtDisplay(draw1), XtWindow(draw1), draw_gc, x1-15,  
   b_starty + b_height + 19, num_str);
plot_node(XtDisplay(draw1), XtWindow(draw1), draw_gc, x2-15,  
   b_starty + b_height + 19, num_str);
plot_node(XtDisplay(draw1), XtWindow(draw1), draw_gc, x3-15,  
   b_starty + b_height + 19, num_str);
plot_node(XtDisplay(draw1), pixmap1, draw_gc, x1-15,  
   b_starty + b_height + 19, num_str);
plot_node(XtDisplay(draw1), pixmap1, draw_gc, x2-15,  
   b_starty + b_height + 19, num_str);
plot_node(XtDisplay(draw1), pixmap1, draw_gc, x3-15,  
   b_starty + b_height + 19, num_str);

sprintf(num_str, "%9.1f", spHeight/a);
drawLine(XtDisplay(draw1), XtWindow(draw1), whichGC, b_startx,  
   b_starty + b_height - spHeight, 
   b_width + b_startx, b_starty + b_height - spHeight);
drawLine(XtDisplay(draw1), pixmap1, whichGC, b_startx, b_starty +  
   b_height - spHeight, 
   b_width + b_startx, b_starty + b_height - spHeight);
plot_node(XtDisplay(draw1), XtWindow(draw1), draw_gc, b_startx + 10,  
   b_starty + b_height - spHeight - 3, num_str);
plot_node(XtDisplay(draw1), pixmap1, draw_gc, b_startx + 10,  
   b_starty + b_height - spHeight - 3, num_str);

sprintf(num_str, "%9.1f", spHeight/b);
drawLine(XtDisplay(draw1), XtWindow(draw1), whichGC, b_startx +  
   b_width + b_dis, 
   b_starty + b_height - spHeight, 
   b_starxt + 2*b_width + b_dis, 
   b_starty + b_height - spHeight);
drawLine(XtDisplay(draw1), pixmap1, whichGC, b_startx + b_width + b_dis,  
   b_starty + b_height - spHeight, 
   b_starxt + 2*b_width + b_dis, 
   b_starty + b_height - spHeight);
plot_job.c

plot_node ( XtDisplay(draw1), XtWindow(draw1), draw_gc, b_startx +
b_dis + b_width + 10,
b_stary + b_height - spHeight - 3, num_str );
plot_node ( XtDisplay(draw1), pixmap1, draw_gc, b_startx + b_dis +
b_width + 10, b_stary + b_height - spHeight - 3, num_str );

sprintf(num_str, "%9.1f", spHeight/c);
drawLine ( XtDisplay(draw1), XtWindow(draw1), whichGC,
b_startx + 2*b_width + 2*b_dis,
b_stary + b_height - spHeight,
b_startx + 3*b_width + 2*b_dis,
b_stary + b_height - spHeight );
drawLine ( XtDisplay(draw1), pixmap1, whichGC, b_startx +
2*b_width + 2*b_dis,
b_stary + b_height - spHeight,
b_startx + 3*b_width + 2*b_dis,
b_stary + b_height - spHeight );
plot_node ( XtDisplay(draw1), XtWindow(draw1), draw_gc, b_startx +
2*b_dis + 2*b_width + 10,
b_stary + b_height - spHeight - 3, num_str );
plot_node ( XtDisplay(draw1), pixmap1, draw_gc, b_startx +
2*b_dis + 2*b_width + 10,
b_stary + b_height - spHeight - 3, num_str );

fontStruct = initFont ( draw1, "8x13B");
title = "Spectrum of Original Data"
myTitleWidth = textBoxWidth ( title, fontStruct );
i = textBoxHeight ( fontStruct ); x1 = 1.5*i;
plot_node ( XtDisplay(draw1), XtWindow(draw1), draw_gc,
b_width/2 - myTitleWidth/2 + b_startx,
b_stary - x1, title );
plot_node ( XtDisplay(draw1), pixmap1, draw_gc, b_width/2 -
myTitleWidth/2 + b_startx,
b_stary - x1, title );
title = "Spectrum of Reconstruction"
myTitleWidth = textBoxWidth ( title, fontStruct );
plot_node ( XtDisplay(draw1), XtWindow(draw1), draw_gc,
3*b_width/2 - myTitleWidth/2 +
b_startx + b_dis, b_stary - i - x1, title );
plot_node ( XtDisplay(draw1), pixmap1, draw_gc, 3*b_width/2 -
myTitleWidth/2 +
b_startx + b_dis, b_stary - i - x1, title );
subtitle = "by Modified Method"
myTitleWidth = textBoxWidth ( subtitle, fontStruct );
plot_node ( XtDisplay(draw1), XtWindow(draw1), draw_gc,
3*b_width/2 - myTitleWidth/2 +
b_startx + b_dis, b_stary - i, subtitle );
plot_node ( XtDisplay(draw1), pixmap1, draw_gc, 3*b_width/2 -
myTitleWidth/2 +
b_startx + b_dis, b_stary - i, subtitle );
plot_job.c

title = "Spectrum of Reconstruction";
myTitleWidth = textWidth ( title, fontStruct );
plot_node ( XDisplay(draw1), XtWindow(draw1), draw_gc,
  5*b_width/2 - myTitleWidth/2 +
  b_startx + 2*b_dis, b_starty - i - x1, title );
plot_node ( XDisplay(draw1), pixmap1, draw_gc, 5*b_width/2 -
  myTitleWidth/2 +
  b_startx + 2*b_dis, b_starty - i - x1, title );
subtitle = "by Ordinary Method";
myTitleWidth = textWidth ( subtitle, fontStruct );
plot_node ( XDisplay(draw1), XtWindow(draw1), draw_gc,
  5*b_width/2 - myTitleWidth/2 +
  b_startx + 2*b_dis, b_starty - i, subtitle );
plot_node ( XDisplay(draw1), pixmap1, draw_gc, 5*b_width/2 -
  myTitleWidth/2 +
  b_startx + 2*b_dis, b_starty - i, subtitle );
drawLines ( XDisplay(draw1), XtWindow(draw1), whichGC, orgpoints,
  NT/2+1 );
drawLines ( XDisplay(draw1), XtWindow(draw1), whichGC, mpoints,
  NT/2+1 );
drawLines ( XDisplay(draw1), XtWindow(draw1), whichGC,
  wrongpoints, NT/2+1 );
drawLines ( XDisplay(draw1), pixmap1, whichGC, orgpoints, NT/2+1 );
drawLines ( XDisplay(draw1), pixmap1, whichGC, mpoints, NT/2+1 );
drawLines ( XDisplay(draw1), pixmap1, whichGC, wrongpoints, NT/2+1 );

}

plot_OldSST( )
{
  extern complex oldtp[][NT];
  extern float old_p1, old_dp;
  extern int oldnp;
  char *name;

  name = "Slant Stack by the Ordinary Method";
  plotData ( oldtp, oldnp, NT, old_p1, old_dp, name, "tp" );
}

plot_OldInvg( )
{
  extern complex oldinvg[][NT];
  char *name;

  name = "Reconstruction of Slant Stack by the Ordinary Method";
  plotData ( oldinvg, TR, NT, 0., OFFSET*1., name, "gt" );
}
plot_NewSST( )
{
    extern complex newtp[][NT];
    extern float new_p1, new_dp;
    extern int newnp;
    char *name;

    name = "Slant Stack by the Modified Method";
    plotData ( newtp, newnp, NT, new_p1, new_dp, name, "tp" );
}

plot_NewInvgt( )
{
    extern complex newinvgt[][NT];
    char *name;

    name = "Reconstruction of Slant Stack by the Modified Method";
    plotData ( newinvgt, TR, NT, 0., OFFSET*1., name, "gt" );
}

plot_gt( )
{
    extern complex gt[][NT];
    char *name;

    name = "Original Model for Slant Stack and its Reconstruction";
    plotData ( gt, TR, NT, 0., OFFSET*1., name, "gt" );
}

horizon ( GC whichGC )
{
    int y1, y2;

    y1 = 1./DT + 30.;
    y2 = 2./DT + 30.;

    drawLine( XDisplay(draw2), XtWindow(draw2), whichGC,
        boxStart - 15, VStart, boxlen + boxStart + 15, VStart );
    drawLine( XDisplay(draw2), XtWindow(draw2), whichGC, boxStart - 15, y1,
        boxlen + boxStart + 15, y1 );
    drawLine( XDisplay(draw2), XtWindow(draw2), whichGC, boxStart - 15, y2,
        boxlen + boxStart + 15, y2 );
    drawLine( XDisplay(draw2), pixmap2, whichGC, boxStart - 15, VStart,
        boxlen + boxStart + 15, VStart );
drawLine( Xtdisplay(draw2), pixmap2, whichGC, boxStart - 15, y1, 
        boxlen + boxStart + 15, y1 );
drawLine( Xtdisplay(draw2), pixmap2, whichGC, boxStart - 15, y2, 
        boxlen + boxStart + 15, y2 );
fontStruct = initFont ( draw2, "variable" );
plot_node( Xtdisplay(draw2), XtWindow(draw2), draw_gc, boxStart - 15, 
        VStart - 5, "0" );
plot_node( Xtdisplay(draw2), XtWindow(draw2), draw_gc, boxlen + 
        boxStart + 11, VStart - 5, "0" );
plot_node( Xtdisplay(draw2), XtWindow(draw2), draw_gc, boxStart - 15, 
        y1 - 5, "1" );
plot_node( Xtdisplay(draw2), XtWindow(draw2), draw_gc, boxlen + 
        boxStart + 11, y1 - 5, "1" );
plot_node( Xtdisplay(draw2), XtWindow(draw2), draw_gc, boxStart - 15, 
        y2 - 5, "2" );
plot_node( Xtdisplay(draw2), XtWindow(draw2), draw_gc, boxlen + 
        boxStart + 11, y2 - 5, "2" );
plot_node( Xtdisplay(draw2), pixmap2, draw_gc, boxStart - 15, VStart - 5, "0" );
plot_node( Xtdisplay(draw2), pixmap2, draw_gc, boxlen + boxStart + 11, 
        VStart - 5, "0" );
plot_node( Xtdisplay(draw2), pixmap2, draw_gc, boxStart - 15, y1 - 5, "1" );
plot_node( Xtdisplay(draw2), pixmap2, draw_gc, boxlen + boxStart + 11, 
        y1 - 5, "1" );
plot_node( Xtdisplay(draw2), pixmap2, draw_gc, boxStart - 15, y2 - 5, "2" );
plot_node( Xtdisplay(draw2), pixmap2, draw_gc, boxlen + boxStart + 11, 
        y2 - 5, "2" );
}

plotData ( input_d, numberOfColumn, numberOfRows, offsetStart, offset, name, type )
complex           input_d[i][NT];
int                numberOfColumn, numberOfRows;
float              offsetStart, offset;
char               *name, *type;
{
    register float    b = 0., scale, scale1, fli;
    register int      i, j;
    XPoint            nld_Points[NT];
    float              fabs(), scale2;
    int                myTitleWidth;

    XClearWindow( Xtdisplay(draw2), XtWindow(draw2) );
    XFillRectangle(Xtdisplay(draw2), pixmap2, undraw_gc, 0, 0, 10000, 5000);

    scale = boxlen / ( numberOfColumn - 1 );

    for (i=0; i<numberOfRow;i++)
        nld_Points[i].y = i + VStart;

    for (j=0; j < numberOfColumn;j++)
        for (i=0; i<numberOfRow;i++)
        {
            scale1 = fabs(input_d[i][j].x);
            b = scale1 > b ? scale1 : b;
        }
b = b == 0 ? 1 : scale/b;
scale1 = 0;
if (strcmp (type, "gt", 2) == 0) scale2 = .5;
if (strcmp (type, "tp", 2) == 0) scale2 = 2;
for (j=0; j < numberOfColumn; j++)
{
    for (i=0; i<numberOfRow;i++)
    {
        nld_Points[i].x = input_dt[j][i].x*b*scale2 +
        boxStart + scale1;
    }
    drawLines( XtDisplay(draw2), XtWindow(draw2), draw_gc,
              nld_Points, NT);
    drawLines( XtDisplay(draw2), pixmap2, draw_gc, nld_Points, NT);
    fillpolygon( XtDisplay(draw2), XtWindow(draw2), draw_gc,
               nld_Points, NT, boxStart + scale1);
    fillpolygon( XtDisplay(draw2), pixmap2, draw_gc, nld_Points,
               NT, boxStart + scale1);
    scale1 = scale1 + scale;
}
fontStruct = initFont (draw2, "8x13B");
if (strcmp (type, "gt", 2) == 0)
    for (i=0; i < numberOfColumn; i++)
    {
        j = i*offset + offsetStart;
        sprintf(num_str, "%d", j);
        j = i == 0 ? boxStart - 2 : boxStart + i*scale - 10;
        plot_node( XtDisplay(draw2), XtWindow(draw2), draw_gc,
                   j, VStart + NT + 20, num_str);
        plot_node( XtDisplay(draw2), pixmap2, draw_gc, j, VStart +
                   NT + 20, num_str);
    }
else if (strcmp (type, "tp", 2) == 0)
{
    for (fli = 0; fli < numberOfColumn; fli += (numberOfColumn-1)/10.)
    {
        sprintf(num_str, "%2f", (fli*offset + offsetStart)*1e4);
        j = fli == 0 ? boxStart - 10 : boxStart + fli*scale - 18;
        plot_node( XtDisplay(draw2), XtWindow(draw2), draw_gc,
                   j, VStart + NT + 20, num_str);
        plot_node( XtDisplay(draw2), pixmap2, draw_gc, j, VStart +
                   NT + 20, num_str);
    }
    plot_node( XtDisplay(draw2), XtWindow(draw2), draw_gc,
              boxlen + boxStart, VStart + NT + 35, "e-4");
    plot_node( XtDisplay(draw2), pixmap2, draw_gc, boxlen + boxStart,
              VStart + NT + 35, "e-4");
}
horizon ( draw_gc);
fontStruct = initFont ( draw2, "Rom22" );
myTitleWidth = textWidth ( name, fontStruct );
i = ( pixmap2_width - myTitleWidth ) / 2;
plot_job.c

plot_node ( XtDisplay(draw2), XtWindow(draw2), draw_gc, i, VStart - 15, name );
plot_node ( XtDisplay(draw2), pixmap2, draw_gc, i, VStart - 15, name );
fontStruct = initFont ( draw2, "8x13B");

}

int round ( x ) /* round a float number to its nearest integer */
float x;
{
int y;
y = x;
if ( ( x - y ) >= .5 ) return ( y + 1);
else return ( y );
}

absolute ( points ) /* calculate the modul of complex number */
complex points;
{
float absresult;
absresult = sqrt ( pow (1.*points.x, 2.) + pow (1.*points.y, 2.) ) ;
return (absresult);
}

/*
** end of file plot_job.c
*/
```c
#include <stdio.h>
#include "constant.h"

extern float new_dp, old_dp, old_p1, new_p1, deft;
extern int oldnp, newnp, fstart, fend, Yes, muteTimes;
extern complex gt[][NT];
extern complex oldinvgt[][NT];
extern complex newinvgt[][NT];
extern complex oldp[][NT];
extern complex newp[][NT];

static void
saveData(data, rows, columns, fileName)
complex data[][NT];
int rows, columns;
char *fileName;
{
    register int i, j;
    FILE *fp;

    if((fp=fopen(fileName, "wb"))==NULL)
    {
        printf("cannot open file to write\n");
        fclose(fp);
        return;
    }

    for (i=0; i<rows; i++)
    {
        for (j=0; j<columns; j++) fprintf(fp, "%f", data[i][j].x);
        fprintf(fp, "\n");
    }

    fclose(fp);
}

save_cb(w, cd, cld)
Widget w;
caddr_t cd, cld;
{
    saveData(gt, TR, NT, "gt.mat");
    saveData(oldinvgt, TR, NT, "oldinvgt.mat");
    saveData(newinvgt, TR, NT, "newinvgt.mat");
    saveData(oldp, oldnp, NT, "oldp.mat");
    saveData(newp, newnp, NT, "newp.mat");

    printf("TR = %d, NT = %d, oldnp = %d, newnp = %d\n", 
        TR, NT, oldnp, newnp);
    printf("deft = %f, offset = %d\n", deft, OFFSET);
    printf("new_dp = %f, old_dp = %f\n", new_dp, old_dp);
    printf("old_p1 = %f, new_p1 = %f\n", old_p1, new_p1);
}
```
/* X Project for the "New Approach to Slant Stack"*/

This software package is a interactive processing program, which involves in the slant stack and inversion by the standard method and by the modified method

Lideng Ni 2/22/1991

**** Slant Stack function used to transform data in x - t domain to tau - p domain.
**** Function oldtaup is to do conventional tau-p transform while newtaup is to do the tau-p transform proposed by Lideng Ni.
****

#include <math.h>
#include "constant.h"
#include "slstk.h"

extern int muteTimes;

oldtaup( gt_data )

complex . gt_data[][]NT);

[ 

  register int i, j, k;
  float f[NT/2+1], argStep, argStart;
  float pmax, x1;
  float p1x1, dpx1, dpdx, p1dx;
  complex mat[DIM][TR];

  fft ( TR, NT, gt_data);

  for ( i = 0; i < TR; i++ )
  {
    gt_data[i][0].x = .5 * gt_data[i][0].x;
    gt_data[i][0].y = .5 * gt_data[i][0].y;
  }

detf = 1. / ( NT * DT );
  fstart = FMIN/detf;
  fend   = FMAX/detf;

  for ( i = 0; i <= NT/2 + 1; i++ )
    f[i] = detf * i ;

  oldnp = 6*TR;

  pmax = 1. / (f[fstart] * OFFSET );

  p range
pmax = 1/500;
old_dp = pmax / (oldnp - 1);
/*
old_p1 = -1/500;
*/
old_p1 = 0.0;
x1 = -( stack_start - 1 ) * OFFSET;
p1x1 = 2 * PI * defX * old_p1 * x1;
dpx1 = 2 * PI * defX * old_dp * x1;
dpdx = 2 * PI * defX * old_dp * OFFSET;
p1dx = 2 * PI * defX * old_p1 * OFFSET;

if ( Filter_Status == BP || Filter_Status == HP )
{
    for ( i = 0; i < oldnp; i++ )
    {
        for ( j = 0; j < TR; j++ )
        {
            argStep = p1x1 + i * dpdx + j * p1dx;
            oldmult[i][j].x = cos ( argStep );
            oldmult[i][j].y = sin ( argStep );
            argStart = argStep * fffstart/defX;
            mat[i][j].x = cos ( argStart );
            mat[i][j].y = sin ( argStart );
        }
    }
}
else
{
    for ( i = 0; i < oldnp; i++ )
    {
        for ( j = 0; j < TR; j++ )
        {
            argStep = p1x1 + i * dpdx + j * p1dx;
            oldmult[i][j].x = cos ( argStep );
            oldmult[i][j].y = sin ( argStep );
            mat[i][j].x = 1;
            mat[i][j].y = 0;
        }
    }
}

TPSTATUS /* for ( j = 0; j < NT/2+1; j++ ) */

for ( i = 0; i < oldnp; i++ )
{
    for ( k = 0; k < TR; k++ )
    {
        oldsp[i][j].x = oldsp[i][j].x +
            gt_data[k][j].x * mat[i][k].x -
            gt_data[k][j].y * mat[i][k].y;
        }
oldtp[i][j].y = oldtp[i][j].y +
    gt_data[k][j].x * mat[i][k].y +
    gt_data[k][j].y * mat[i][k].x;

temp.x = mat[i][k].x * oldmult[i][k].x -
    mat[i][k].y * oldmult[i][k].y;

temp.y = mat[i][k].y * oldmult[i][k].x +
    mat[i][k].x * oldmult[i][k].y;

mat[i][k] = temp;

ifft (oldnp, NT, oldtp);
/* -- end of oldtaup function */

newtaup (input_gt)
complex input_gt[][NT];
{
    register int i, j, k, l;
    int npmax, npmin, istart, istart0, iend, newnp1, newnp2, iend_end;
    float p1x1, dp1x1, dpdx, p1dx, pmax;
    float x1, fixed, scale;
    float f[NT/2+1], argStep, argStart;
    complex mat[DIM][TR];

    fft (TR, NT, input_gt);

    for (i = 0; i < TR; i++)
    {
        input_gt[i][0].x = .5 * input_gt[i][0].x;
        input_gt[i][0].y = .5 * input_gt[i][0].y;
    }

    detf = 1. / (NT * DT);
    fstart = FMIN/detf;
    fend = FMAX/detf;

    for (i = 0; i <= NT/2 + 1; i++)
        f[i] = detf * i;

    npmin = TR;
    pmax = 1. / (OFFSET * f[fstart]);
    new_dp = 1. / (OFFSET * f[fend] * (npmin - 1));
    npmax = 1. / (OFFSET * f[fstart] * new_dp) + 1;
new_p1 = 0.35 / 450 - 1./FMIN / OFFSET / 2;

/*
new_p1 = 0.;
*/
x1 = - (stack_start - 1) * OFFSET;
p1x1 = 2. * PI * detf * new_p1 * x1;
dpx1 = 2. * PI * detf * new_dp * x1;
dpdx = 2. * PI * detf * new_dp * OFFSET;
p1dx = 2. * PI * detf * new_p1 * OFFSET;

if (Filter_Status == BP || Filter_Status == HP)
{
    for (i = 0; i < npmax; i++)
    {
        for (j = 0; j < TR; j++)
        {
            argStep = p1x1 + i * dpdx + j * p1dx;
            newmult[i][j].x = cos(argStep);
            newmult[i][j].y = sin(argStep);
            argStart = argStep * f[fstart]/detf;
            mat[i][j].x = cos(argStart);
            mat[i][j].y = sin(argStart);
        }
    }
} else
{
    for (i = 0; i < npmax; i++)
    {
        for (j = 0; j < TR; j++)
        {
            argStep = p1x1 + i * dpdx + j * p1dx;
            newmult[i][j].x = cos(argStep);
            newmult[i][j].y = sin(argStep);
            mat[i][j].x = 1;
            mat[i][j].y = 0;
        }
    }
}

fixed = 1./(OFFSET*new_dp);
istart0 = 1./(OFFSET*f[fstart]*new_dp);
ien = 1./(OFFSET*f[fend]*new_dp);
newnp1 = fixed / f[fstart] + 1;
newnp2 = fixed / f[fend] + 1;
start_end = (istart0 - iend)/2;

TPSTATUS

    if (j < fstart)
    {
        newnp = newnp1-1;
    }
istart = 0;
scale = j == 0 ? 0. : f[fstart] / f[j];
}
else if ( j > fend )
{
    newnp = newnp2-1;
    istart = istart_end;
    scale = f[fend] / f[j];
}
else
{
    newnp = fixed / f[j] + 1-1;
    istart = ( istart0 - newnp )/2;
    scale = 1.;
}
for ( i = istart; i < newnp + istart; i++ )
{
    for ( k = 0; k < TR; k++ )
    {
        newtp[i][j].x = newtp[i][j].x +
        scale*input_gt[k][j].x * mat[i][k].x -
        scale*input_gt[k][j].y * mat[i][k].y;

        newtp[i][j].y = newtp[i][j].y +
        scale*input_gt[k][j].x * mat[i][k].y +
        scale*input_gt[k][j].y * mat[i][k].x;

        temp.x = mat[i][k].x * newmult[i][k].x -
        mat[i][k].y * newmult[i][k].y;

        temp.y = mat[i][k].y * newmult[i][k].x +
        mat[i][k].x * newmult[i][k].y;

        mat[i][k] = temp;
    }
}

ifft ( npmax, NT, newtp );
newnp = npmax;
/* -- end of newtauf function */

/****
   Slant Stack inverse function used to transform data in tau - p
   domain to x - t domain.
   Function oldx is to do the inverse transform with conventional
   method while newxt is to do the inverse transform proposed by Lideng Ni.
   ****/
oldxt ( input_tp )

complex input_tp[][NT];
{
    register int i, j, k;
    complex mat[DIM][TR];

    fft ( oldnp, NT, input_tp );

    for ( i = 0; i < oldnp; i++ )
    {
        input_tp[i][0].x = .5 * input_tp[i][0].x;
        input_tp[i][0].y = .5 * input_tp[i][0].y;
    }

    for ( i = 0; i < oldnp; i++ )
        for ( j = 0; j < TR; j++ )
        {
            mat[i][j].x = 1;
            mat[i][j].y = 0;
        }

XTSTATUS

    for ( i = 0; i < TR; i++ )
    {
        for ( k = 0; k < oldnp; k++ )
        {

            oldinvgt[i][j].x = oldinvgt[i][j].x +
                j * ddef * input_tp[k][j].x * mat[k][i].x -
                j * ddef * input_tp[k][j].y * mat[k][i].y;

            oldinvgt[i][j].y = oldinvgt[i][j].y +
                j * ddef * input_tp[k][j].x * mat[k][i].y +
                j * ddef * input_tp[k][j].y * mat[k][i].x;

            temp.x = mat[k][i].x * oldmult[k][i].x +
                mat[k][i].y * oldmult[k][i].y;

            temp.y = mat[k][i].y * oldmult[k][i].x -
                mat[k][i].x * oldmult[k][i].y;

            mat[k][i] = temp;

        }
    }

ifft ( TR, NT, oldinvgt );
} /* -- end of oldxt function */
newxt ( input_tp )

complex input_tp[][NT];
{
    register int i, j, k, l;
    complex mat[DIM][TR];

    fft ( newnp, NT, input_tp);

    for ( i = 0; i < newnp; i++ )
    {
        input_tp[i][0].x = .5 * input_tp[i][0].x;
        input_tp[i][0].y = .5 * input_tp[i][0].y;
    }

    for ( i = 0; i < newnp; i++ )
        for ( j = 0; j < TR; j++ )
            { mat[i][j].x = 1;
              mat[i][j].y = 0;
            }

    XTSTATUS

    for ( i = 0; i < TR; i++ )
    {
        for ( k = 0; k < newnp; k++ )
        {
            newinvgt[i][j].x = newinvgt[i][j].x +
                j * defl * input_tp[k][j].x * mat[k][i].x -
                j * defl * input_tp[k][j].y * mat[k][i].y;

            newinvgt[i][j].y = newinvgt[i][j].y +
                j * defl * input_tp[k][j].x * mat[k][i].y +
                j * defl * input_tp[k][j].y * mat[k][i].x;

            temp.x = mat[k][i].x * newmult[k][i].x +
                mat[k][i].y * newmult[k][i].y;

            temp.y = mat[k][i].y * newmult[k][i].x -
                mat[k][i].x * newmult[k][i].y;

            mat[k][i] = temp;
        }
    }

    if ( muteTimes > 0 )
        for ( j = 0; j < NT/2 + 1 ; j++ )
        {
            newinvgt[0][j].x = newinvgt[0][j].x/4.:
            newinvgt[0][j].y = newinvgt[0][j].y/4.:
newinvgt[1][j].x = newinvgt[1][j].x/2.;
newinvgt[1][j].y = newinvgt[1][j].y/2.;
newinvgt[TR-2][j].x = newinvgt[TR-2][j].x/2.;
newinvgt[TR-2][j].y = newinvgt[TR-2][j].y/2.;
newinvgt[TR-1][j].x = newinvgt[TR-1][j].x/4.;
newinvgt[TR-1][j].y = newinvgt[TR-1][j].y/4.;
}

ifft ( TR, NT, newinvgt );

/\* -- end of newxt function */
complex  gt[TR][NT];
complex  oldinvgt[TR][NT];
complex  newinvgt[TR][NT];
complex  oldp[DIM][NT];
complex  newp[DIM][NT];
complex  oldmult[DIM][TR];
complex  newmult[DIM][TR];
complex  temp;

#define NOFILTER for ( j = 0; j < NT/2+1; j++ ) {
#define BANDPASS for ( j = fstart; j <= fend; j++ ) {
#define LOWPASS for ( j = 0; j <= fend; j++ ) {
#define HIGHPASS for ( j = fstart; j < NT/2+1; j++ ) {

#define NF 0
#define BP 1
#define LP 2
#define HP 3

#define Filter_Status NF /**< Define_Filter_Status */
#define TPSTATUS NOFILTER /**< Filter_Status_Loop */
#define XTSTATUS NOFILTER /**< Don't change this */

/* with respect to which trace you want to stack */
int stack_start = ( TR - 1 ) / 2;

float new_dp, old_dp, old_p1, new_p1, def;
int oldnp, newnp, fstart, fend;
/*
 ** X Project for the "New Approach to Slant Stack"
 ** This software package is an interactive processing program, which involves
 ** in the slant stack and inversion by the standard method and by the modified
 ** method
 **
 ** Lideng Ni 2/22/1991
 */

#include <Xm/Xm.h>
#include "constant.h"

XFontStruct *fontStruct, *initFont();

/*
 ** textHeight returns the max height of the tallest characters
 ** in the given font.
 */

textHeight( fontStructName )
XFontStruct *fontStructName;
{
    /* -- function textHeight */
    int theHeight;

    theHeight = fontStructName->ascent + fontStructName->descent;

    return( theHeight );
} /* -- function textHeight */

/*
 ** textWidth returns the width of the given string in the
 ** given font.
 */

textWidth( theString, fontStructName )
char theString[];
XFontStruct *fontStructName;
{
    /* -- function textWidth */
    int theLength;

    theLength = XTextWidth( fontStructName, theString, strlen( theString ) );

    return( theLength );
} /* -- function textWidth */
XFontStruct *
initFont( Widget w, String fontName )
{
    static XFontStruct *lastFont;

    if (lastFont) XFreeFont (XtDisplay(w), lastFont);

    fontStruct = XLoadQueryFont( XtDisplay(w), fontName );

    if ( fontStruct == 0 )
    {
        printf("Warning: unable to load font %s. The font name 'variable' is\n        used instead.
", fontName);

        fontStruct = XLoadQueryFont( XtDisplay(w), "variable");
    }

    XSetFont( XtDisplay(w), draw_gd, fontStruct->fid );

    lastFont = fontStruct;
    return( fontStruct );
}

/*
 **    end of file textx.c
*/
#include <Xm/Xm.h>
#include <Xm/Frame.h>
#include <Xm/MainW.h>
#include <Xm/PanedW.h>

#include "constant.h"

#define MAIN_TTLE "A X Window Project for A New Approach to Slant Stack by Lideng Ni"

Widget draw1, draw2;

main(argc, argv)
unsigned int argc;
char **argv;
{
    Widget topLevel, pane, mainW, frame;
    XtApplicationContext app_context;
    Arg args[10];
    int n;
    char *num, *item;
    static XtActionsRec window_actions[] = {
        {"RedrawPicture",     RedrawPicture},
        {"processDown",       processDown},
        {"processMotion",     processMotion},
        {"processUp",         processUp},
    };
    static String trans1 = "<Expose>: RedrawPicture();"
    static String trans2 = "<Expose>: RedrawPicture() \n
    <Btn1Down>: processDown() \n
    <Btn1Motion>: processMotion() \n
    <Btn1Up>: processUp()"

    XtSetArg(args[0], XmNtitle, MAIN_TTLE); n++;
    topLevel = XtAppInitialize(&app_context, "XTool",
        NULL, 0,
        argc, argv, NULL, args, 1);

    mainW = XmCreateMainWindow(topLevel, "MainW", NULL, 0);
    XtManageChild(mainW);

    CreateMenus(mainW);

    n = 0;
    pane = XmCreatePanedWindow(mainW, "Vpane", args, n);
    XtManageChild(pane);

    frame = XmCreateFrame(pane, "Frame1", args, 0);
    XtManageChild(frame);
    draw1 = CreateDrawWindow(frame, pixmap1_width, pixmap1_height, trans1);
    XtManageChild(draw1);
frame = XmCreateFrame(pane, "Frame2", args, 0);
XtManageChild(frame);
draw2 = CreateDrawWindow(frame, pixmap2_width, pixmap2_height, trans2);
XtManageChild(draw2);

set_up_things(draw2);

XtAppAddActions(app_context, window_actions, XtNumber(window_actions));

XtRealizeWidget(topLevel);

XtAppMainLoop(app_context);
```c
#include <stdio.h>
#include <Xm/CascadeB.h>
#include <Xm/MenuShell.h>
#include <Xm/MessageB.h>
#include <Xm/RowColumn.h>
#include <Xm/SeparatorG.h>

#include "constant.h"

#define READ_MENU "Read data"
#define JOBS_MENU "Slant stack"
#define PLOT_MENU "Plot data"
#define HELP_MENU "Help"

#define PICK 0  
#define MUTE 1

static XmStringCharSet charset = (XmStringCharSet) XmSTRING_DEFAULT_CHARSET;

static XmString MakeXmString(string)
char **string:
{
    return XmStringLtoRCreate(string, charset);
}

#define BLANK ''
#define NULLC '0'

static char DefaultMnemonic(label)
String label;
{
    char *pos = label;

    while(*pos != NULLC)
        if (*pos++ == BLANK) return *pos;
    return *label;
}

static Widget CreateHelp(parent)
Widget parent;
{
    Widget button, message_box;
    Arg args[10];
    register int n;
    static char message[4*BUFSIZ];
    XmString message_string = NULL;
    XmString button_string = NULL;
    XmString title_string = NULL;

    sprintf(message, \n```
This software package is a X window program written with Xt intrinsic and MOTIF.

Author: Lideng Ni

Department of Geology and Geophysics
Rice University
P.O.Box 1892
Houston, TX 77251

It is an interactive processing program, which involves the slant stack and its inversion by the standard method and by my modified method. Any suggestions, hints and error reports are welcome.

It understands all standard Xt and Motif command line options. The additional options are as follows:

Option Valid Range

-foreground foreground color name

-bg background color name

-background background color name

-fn font name

-display server name used to display

-size and position

-title title of application

-reverse on reverse video

-reverse off no reverse video

-debug no value necessary

Press the 'close' button to close this help facility. Select 'quit' from the menu to exit this application. "O")

message_string = XmStringCreateLtoR(message, charset);
button_string = XmStringCreateLtoR("Close", charset);
title_string = XmStringCreateLtoR ("MOTIF", charset);

n = 0;
XtSetArg(args[n], XmNdialogTitle, title_string); n++;
XtSetArg(args[n], XmNokLabelString, button_string); n++;
XtSetArg(args[n], XmNmessageString, message_string); n++;
message_box = XmCreateMessageDialog(parent, "helpbox", args, n);

button = XmMessageBoxGetChild(message_box, XmDIALOGCANCEL_BUTTON);
XtUnmanageChild(button);

button = XmMessageBoxGetChild(message_box, XmDIALOGHELP_BUTTON);
XtUnmanageChild(button);

if (title_string) XtFree(title_string);
if (message_string) XtFree(message_string);
if (button_string) XtFree(button_string);
return (message_box);
static void
wipeOut_ch(w, client_data, call_data)
Widget w;
int client_data, call_data;
{
    extern GC undraw_gc;
    extern Pixmap pixmap1, pixmap2;

    XBell(XtDisplay(w), 100);

    XClearWindow (XtDisplay(w), XtWindow(draw1));
    XFillRectangle(XtDisplay(w), pixmap1, undraw_gc, 0, 0,
    pixmap1_width, pixmap1_height);

    XClearWindow (XtDisplay(w), XtWindow(draw2));
    XFillRectangle(XtDisplay(w), pixmap2, undraw_gc, 0, 0,
    pixmap2_width, pixmap2_height);
}

#define NErrInternal  "internal"
#define TErrMenu      "Menu"
#define ERROR_CLASS   "MhelpError"

static int
ButtonIndex(buttons, call_data, num)
Widget buttons[];
int num;
XmRowColumnCallbackStruct *call_data;
{
    int i;

    for (i=0; i<num; i++)
        if (buttons[i] == call_data->widget) return i;
    XtErrorMsg(NErrInternal, TErrMenu, ERROR_CLASS, "button not found",
               NULL, NULL);
    return -1;
}

static String
jobMenus [] = {"Standard slant stack","Standard slant stack inversion",
               "Modified slant stack","Modified slant stack inversion"};

static String
plotMenus [] = {"Plot original data","Plot standard slant stack",
                "Plot standard inversion","Plot modified slant stack",
                "Plot modified inversion"};

static String
modelMenus [] = {"Model 1","Model 2","Model 3","Model 4",
                 "Model 5","Model 6","Model 7","Model 8","Model 9",
                 "Model 10","Model 11","Model 12"};

static String
fileMenus [] = {"Save...","Open...","","Print","","Exit"};
static String viewMenus[] = {"View Spectrum"};
static String optionMenus[] = {"Mute taup data", "", "Draw a line", "", "Clear Window"};
static String helpMenus[] = {"help"};

#define NUM_JOBS XtNumber(jobMenus)
#define NUM_PLOTS XtNumber(plotMenus)
#define NUM_MODELS XtNumber(modelMenus)
#define NUM_FILES XtNumber(fileMenus)
#define NUM_VIEWS XtNumber(viewMenus)
#define NUM_OPTIONS XtNumber(optionMenus)
#define NUM_HELPS XtNumber(helpMenus)

static Widget jobButtons[NUM_JOBS], plotButtons[NUM_PLOTS],
    modelButtons[NUM_MODELS], fileButtons[NUM_FILES],
    viewButtons[NUM_VIEWS], optionButtons[NUM_OPTIONS],
    helpButtons[NUM_HELPS];

static void read_cb(Widget w, caddr_t client_data, caddr_t call_data)
{
    int i;
    int whichB = ButtonIndex(modelButtons, call_data, NUM_MODELS);

    XBell(XtDisplay(w), 100);
    XFlush(XtDisplay(w));

    switch (whichB )
    {
    case 0: gen1(); break;
    case 1: gen2(); break;
    case 2: gen3(); break;
    case 3: gen4(); break;
    case 4: gen5(); break;
    case 5: gen6(); break;
    case 6: gen7(); break;
    case 7: gen8(); break;
    case 8: gen9(); break;
    case 9: gen10(); break;
    case 10: gen11(); break;
    case 11: gen12(); break;
    }
    XBell(XtDisplay(w), 100);
    XtSetSensitive(modelButtons[whichB], False);
    XtSetSensitive(plotButtons[0], True);
    XtSetSensitive(jobButtons[0], True);
    XtSetSensitive(jobButtons[2], True);
    XtSetSensitive(jobButtons[1], False);
    XtSetSensitive(jobButtons[3], False);
    for (i=1; i<NUM_PLOTS; i++) XtSetSensitive(plotButtons[i], False);
    XtSetSensitive(optionButtons[0], False);
XtSetSensitive(optionButtons[2], False);
XtSetSensitive(viewButtons[0], False);
}

static void
job_cb(Widget w, caddr_t client_data, caddr_t call_data)
{
extern int mutex1[], mutex2[], mutey1[], mutey2[], muteTimes;
int whichB = ButtonIndex(jobButtons, call_data, NUM_JOBS);

XBell(XtDisplay(w), 100);
XFlush(XtDisplay(w));

switch ( whichB )
{
    case 0:
        doingOldSST();
        XtSetSensitive(jobButtons[0], False);
        XtSetSensitive(jobButtons[1], True);
        break;
    case 1:
        doingOldInvSST(mutex1, mutey1, mutex2, mutey2, muteTimes);
        muteTimes = 0;
        break;
    case 2:
        doingNewSST();
        XtSetSensitive(jobButtons[2], False);
        XtSetSensitive(jobButtons[3], True);
        break;
    case 3:
        doingNewInvSST(mutex1, mutey1, mutex2, mutey2, muteTimes);
        muteTimes = 0;
        break;
}

XtSetSensitive(optionButtons[0], True);
XtSetSensitive(plotButtons[whichB+1], True);
XBell(XtDisplay(w), 100);
}

static void
plot_cb(Widget w, caddr_t client_data, caddr_t call_data)
{
    int whichB = ButtonIndex(plotButtons, call_data, NUM_PLOTS);

    XBell(XtDisplay(w), 100);
    XFlush(XtDisplay(w));

    switch ( whichB )
    {
    case 0:
        plot_gt0();
        break;
case 1:
    plot_OldSST();
    break;
case 2:
    plot_OldInvgt();
    break;
case 3:
    plot_NewSST();
    break;
case 4:
    plot_NewInvgt();
    break;
}
XBell(XtDisplay(w), 100);
XtSetSensitive(optionButtons[0], False);
XtSetSensitive(optionButtons[2], True);
}

static void
option_cb(Widget w, caddr_t cd, caddr_t cld)
{
    int whichB = ButtonIndex(optionButtons, cld, NUM_OPTIONS);
    extern int DrawCommand;

    XBell(XtDisplay(w), 100);
    XFlush(XtDisplay(w));

    switch (whichB)
    {
    case 0:
        DrawCommand = MUTE;
        break;
    case 2:
        DrawCommand = PICK;
        XtSetSensitive(viewButtons[0], True);
        break;
    case 4:
        wipeOut_cb(w, NULL, NULL);
        break;
    default:
        break;
    }
}

static void
file_cb(Widget w, caddr_t cd, caddr_t cld)
{
    int whichB = ButtonIndex(fileButtons, cld, NUM_FILES);

    XBell(XtDisplay(w), 100);
    XFlush(XtDisplay(w));
switch (whichB)
{
    case 0:
        save_cb();
        break;
    case 1:
        break;
    case 5:
        exit();
        break;
    default:
        break;
}

static void
view_cb(Widget w, caddr_t cd, caddr_t cld)
{
    int whichB = ButtonIndex(viewButtons, cld, NUM_VIEWS);

    XBell(XtDisplay(w), 100);
    XFlush(XtDisplay(w));

    switch (whichB)
    {
    case 0:
        findFFTWindow();
        break;
    default:
        break;
    }
    XtSetSensitive(viewButtons[0], False);
}

static void
help_cb(Widget w, caddr_t cd, caddr_t cld)
{
    Widget message_box;
    int whichB = ButtonIndex(helpButtons, cld, NUM_HELPS);

    XBell(XtDisplay(w), 100);
    XFlush(XtDisplay(w));

    switch (whichB)
    {
    case 0:
        message_box = CreateHelp(w);
        XtManageChild(message_box);
        break;
    default:
        break;
    }
}
xmMenu.c

static Widget CreateOneMenu(parent, name, labels, num_labels, buttons, is_toggle)

    Widget parent, buttons[10];
    char *name, *labels[];
    int num_labels;
    Boolean is_toggle;

    Widget shell, casc;
    Arg args[10];
    int row;
    XmString buffer;

    shell = XmCreatePullDownMenu(parent, name, NULL, 0);

    for (row = 0; row < num_labels; row++)
    {
        if (strlen(labels[row]) == 0)
        {
            buttons[row] = XmCreateSeparatorGadget(shell, "msep", NULL, 0);
        }
        else
        {
            buffer = MakeXmString(labels[row]);
            XtSetArg(args[0], XmNlabelString, buffer);
            XtSetArg(args[1], XmNmnemonic,
                DefaultMnemonic(labels[row]));
            if(is_toggle)
            {
                XtSetArg(args[2], XmNset, True);
                buttons[row] = XmCreateToggleButtonGadget(shell, "mToggleButton", args, 3);
            }
            else
            {
                buttons[row] = XmCreatePushButtonGadget(shell, "mButton", args, 2);
            }
        }
        XmStringFree(buffer);
    }

    XtManageChildren(buttons, num_labels);

    buffer = MakeXmString(name);
    XtSetArg(args[0], XmNlabelString, buffer);
    XtSetArg(args[1], XmNmnemonic, *name);
    XtSetArg(args[2], XmNsubMenuId, shell);

    casc = XtCreateManagedWidget("casbutton", xmCascadeButtonWidgetClass,
        parent, args, 3);
    XmStringFree(buffer);
xmMenu.c

if (strcasecmp(name, "help") == 0) {
    XtSetArg(args[0], XmNmenuHelpWidget, casc);
    XtSetValues(parent, args, 1);
}
return shell;

void
CreateMenus(Widget parent)
{
    Widget bar, menu, help[1];
    int n, i;
    Arg args[10];
    extern void exit0;

    n = 0;
    XtSetArg(args[n], XmNspacing, 30); n++;
    bar = XmCreateMenuBar(parent, "bar", args, n);
    XtManageChild(bar);

    menu = CreateOneMenu(bar, "Help", helpMenus,
                          NUM_HELP, helpButtons, True);
    XtAddCallback(menu, XmNentryCallback, help_cb, NULL);

    menu = CreateOneMenu(bar, "File", fileMenus,
                          NUM_FILES, fileButtons, True);
    XtAddCallback(menu, XmNentryCallback, file_cb, NULL);

    menu = CreateOneMenu(bar, "View", viewMenus,
                          NUM_VIEWS, viewButtons, True);
    XtAddCallback(menu, XmNentryCallback, view_cb, NULL);
    for (i = 0; i < NUM_VIEWS; i++)
        XtSetSensitive(viewButtons[i], False);

    menu = CreateOneMenu(bar, "Option", optionMenus,
                          NUM_OPTIONS, optionButtons, True);
    XtAddCallback(menu, XmNentryCallback, option_cb, NULL);
    for (i = 0; i < NUM_OPTIONS-1; i++)
        XtSetSensitive(optionButtons[i], False);

    menu = CreateOneMenu(bar, READ_MENU, modelMenus,
                          NUM_MODELS, modelButtons, True);
    XtAddCallback(menu, XmNentryCallback, read_cb, NULL);

    menu = CreateOneMenu(bar, JOBS_MENU, jobMenus,
                          NUM_JOBS, jobButtons, True);
    XtAddCallback(menu, XmNentryCallback, job_cb, NULL);
    for (i = 0; i < NUM_JOBS; i++) XtSetSensitive(jobButtons[i], False);

    menu = CreateOneMenu(bar, PLOT_MENU, plotMenus,
                          NUM_PLOTS, plotButtons, True);
    XtAddCallback(menu, XmNentryCallback, plot_cb, NULL);
    for (i = 0; i < NUM_PLOTS; i++) XtSetSensitive(plotButtons[i], False);
}
Reference


Cizek, Vaclav, 1986, Discrete Fourier Transforms and Their Applications, Adam Hilger Ltd.


Johnson, B.F. and Reichard, K., 1989, X Window Applications Programming, Management Information Source, Inc..


Wade, C.J., 1988, Forward/inverse Slant stacks in the two-dimensional Fourier Transform Domain Based on an improved Interpolation Operator, MS. thesis, Department of Electrical Engineering, University of Houston
