Modelling, migration, and inversion using Linear Algebra

John C. Bancroft and Rod Blais

ABSTRACT

Modelling migration and inversion can all be accomplished using Linear Algebra. Key to these processes is the diffraction array that is multidimensional. Two dimensional poststack migrations require a four dimension diffraction array. Current processing practices for Least-Squares analysis require a diffraction array that is two dimensional (a matrix), with one dimensional vectors for the reflectivity and seismic data. These matrices and vectors can be derived from multidimensional data by helical unwrapping. The field of Multilinear Algebra may allow the data to retain their multidimensional arrays, but require defining processes such as a two dimensional transpose of a three, or higher, dimensional array. Modelling, migration, and inversion are demonstrated using Linear Algebra with MATLAB software, along with a corresponding 2D transpose of a 4D diffraction matrix.

INTRODUCTION

Claerbout has stated that one of his greatest contributions to geophysics was the use of cycle unwrapping to enabling Least-Squares (LS) solutions of seismic data (Claerbout 1998). This process converts multidimensional arrays of diffractions, reflectivity, and seismic data, into a matrix and two vectors. The LS method provides an optimum inversion for the reflectivity when given the diffraction matrix and seismic data.

Diffraction matrix

A seismic diffraction is the surface recording of energy returned from a subsurface scatterpoint. Scatterpoints are defined on a grid of the subsurface that represents the reflectivity, and with amplitudes that align to form reflecting surfaces. The location and amplitude of a diffraction is defined for every scatterpoint. This information is then used to locate and extract the recorded diffracted energy.

The location and amplitudes of a diffraction for two dimensional data can be estimated from analytic equations or from modelling using ray-tracing or wavefield propagation as illustrated in the following three examples.

1. A hyperbolic equation for the traveltime T of a diffraction is

$$T^2 = T_0^2 + \frac{4x^2}{v^2}, \tag{1}$$

   where T_0 is the vertical zero-offset traveltime, x is the spatial location, and v is the locally constant velocity, usually defined as an RMS velocity. The amplitude is estimated from both T_0 and T. This is a very efficient form for the diffraction as it can be efficiently computed when required.

2. The diffraction can also be estimated from modelling using ray-tracing or wave-propagation methods with the results stored in two 1D arrays, such as:

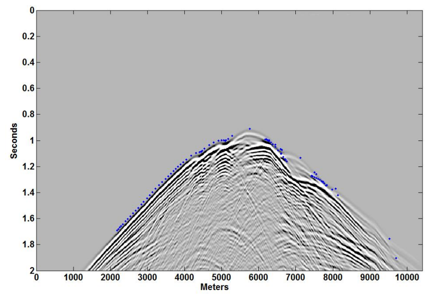
$$a)\ \begin{bmatrix} 3 & 2 & 2 & 2 & 3 \end{bmatrix} \text{ time or sample number}$$

$$b)\ \begin{bmatrix} 0.2 & 0.5 & 1.0 & 0.5 & 0.2 \end{bmatrix} \text{ spatial amplitude of the diffraction}$$

where the elements in the array are defined at spatial locations. The above arrays assume a single arrival time for the diffraction. Multiple arrival times that represent first arrivals or maximum energy can also be incorporated with additional arrays.

3. A third option is to store the diffraction in a matrix as illustrated in Figure 1a, which contains the data from the vectors displayed above. A more realistic diffraction matrix is illustrated in Figure 1b, formed from wave-propagation modelling. This diffraction matrix contains approximately 1000 columns and 1000 rows, and contains all the scattered energy. The wavefield in (b) also includes blue dots that were computed using raytracing.

$$\begin{bmatrix} . & . & . & . & . \\ . & 0.5 & 1.0 & 0.5 & . \\ 0.2 & . & . & . & 0.2 \\ . & . & . & . & . \end{bmatrix}$$



a) b)

FIG. 1 Two diffraction matrices with a) containing a simplified matrix, and b) a large matrix with results from wavefield modelling.

Waveform data is efficiently stored in the matrix where each column is the spatial location, and each row represents the sampled travel time, with each element containing the amplitudes. Multiple arrivals can be easily included in the matrix form. Use of a diffraction matrix is desirable for processing with simple Linear Algebra equations, but very limited in practical applications because of the large storage requirements. A Kirchhoff migration using the full wavefield can be a true matched filter and extract all the scattered energy.

Most diffraction energy is processed using either the hyperbolic equation in a time migration, or in the two arrays for a depth migration. These diffractions in 2D data may contain wavelets and filter operators for spectral shaping and anti-alias filtering.

A SIMPLE EXAMPLE

The following example will create, migrate, and invert a simple example of 2D post stack seismic data. We will use a reflectivity matrix **r** to define a diffraction matrix at each location of a reflecting element, and then combine the diffraction matrices into a 4D array **D**. The modelled seismic data **s** are then created from the reflectivity matrix and the diffraction array. We then reverse the process using migration and inversion to reconstruct the reflectivity matrix.

Defining the reflectivity matrix

We start with a very simple geological cross section **r** (for reflectivity) with 5 traces each with 3 samples to form a matrix. Each element in the matrix is a scatterpoint. There are two scatterpoints with amplitudes of 2 and 4. I will use the MATLAB method of dimensioning these data which is similar to FORTRAN but opposite to that used in "C". We have ***K***=3 depth samples and ***L***=5 traces, to form **r**(*K L*), as shown in Figure 2.

$$\mathbf{r} = \begin{matrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

FIG. 2 Reflectivity matrix with 5 traces, each with three depth samples.

Defining the 4D diffraction array

Simplified diffractions are defined at the location of each scatter point in **r** and plotted below in a matrix form in Figure 3. A diffraction matrix is defined for each reflection point **d**(*i*, *j*) in Figure 2, then each diffraction matrix is located at the corresponding element to create a 4D array. The size of the 4Darray is **D**(*I J K L*) with each element accessed as **D**(*i, j, k, l*). The velocity is assumed to be constant, and the shape of all diffractions, at the same depth level, are identical, but the location of the apex is aligned with the location of the scatterpoint.

$$\begin{array}{cc}
 & \begin{matrix} \quad\quad j & & & & \quad\quad\quad\quad\quad\quad l & & \end{matrix} \\
\begin{matrix} i \\ \\ k \\ \\ \end{matrix} &
\left[\begin{matrix}
\begin{bmatrix} 1&.&.&.&. \\ .&1&.&.&. \\ .&.&1&.&. \\ .&.&.&1&. \end{bmatrix} &
\begin{bmatrix} .&1&.&.&. \\ 1&.&1&.&. \\ .&.&.&1&. \\ .&.&.&.&1 \end{bmatrix} &
\begin{bmatrix} .&.&1&.&. \\ .&1&.&1&. \\ 1&.&.&.&1 \\ .&.&.&.&. \end{bmatrix} &
\begin{bmatrix} .&.&.&1&. \\ .&.&1&.&1 \\ .&1&.&.&. \\ 1&.&.&.&. \end{bmatrix} &
\begin{bmatrix} .&.&.&.&1 \\ .&.&.&1&. \\ .&.&1&.&. \\ .&1&.&.&. \end{bmatrix} \\
\begin{bmatrix} .&.&.&.&. \\ 1&1&.&.&. \\ .&.&1&.&. \\ .&.&.&1&. \end{bmatrix} &
\begin{bmatrix} .&.&.&.&. \\ 1&1&1&.&. \\ .&.&.&1&. \\ .&.&.&.&1 \end{bmatrix} &
\begin{bmatrix} .&.&.&.&. \\ .&1&1&1&. \\ 1&.&.&.&1 \\ .&.&.&.&. \end{bmatrix} &
\begin{bmatrix} .&.&.&.&. \\ .&.&1&1&1 \\ .&1&.&.&. \\ 1&.&.&.&. \end{bmatrix} &
\begin{bmatrix} .&.&.&.&. \\ .&.&.&1&1 \\ .&.&1&.&. \\ .&1&.&.&. \end{bmatrix} \\
\begin{bmatrix} .&.&.&.&. \\ .&.&.&.&. \\ 1&1&1&.&. \\ .&.&.&1&. \end{bmatrix} &
\begin{bmatrix} .&.&.&.&. \\ .&.&.&.&. \\ 1&1&1&1&. \\ .&.&.&.&1 \end{bmatrix} &
\begin{bmatrix} .&.&.&.&. \\ .&.&.&.&. \\ 1&1&1&1&1 \\ .&.&.&.&. \end{bmatrix} &
\begin{bmatrix} .&.&.&.&. \\ .&.&.&.&. \\ .&1&1&1&1 \\ 1&.&.&.&. \end{bmatrix} &
\begin{bmatrix} .&.&.&.&. \\ .&.&.&.&. \\ .&.&1&1&1 \\ .&1&.&.&. \end{bmatrix}
\end{matrix}\right]
\end{array}$$

FIG. 3 A 4D array containing diffraction matrices at the location of each scatterpoint.

Converting the reflectivity matrix to a 2D vector

We can create seismic data by converting the multidimensional arrays to data in a matrix and two vectors (Claerbout 1998). Consider the **r** matrix as colour coded in Figure 4. We can convert this data to a vector $\mathbf{r}_v$ by concatenating the columns as in (b), but it is more efficient to display the transpose of this vector $\mathbf{r}_v^{\mathbf{T}}$ as in (c). This is referred to as unwrapping the matrix into a vector.

$$\mathbf{r} = \begin{matrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

a)

$$\mathbf{r_v} = \begin{matrix} 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 4 \\ 0 \end{matrix}$$

b)

$$\mathbf{r_v}^{\mathbf{T}} = \begin{matrix} 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \end{matrix}$$

c)

FIG. 4 Vectorizing the reflector matrix, a) the matrix **r** b) the column vector $\mathbf{r_v}$, and c) the transpose $\mathbf{r_v}^{\mathbf{T}}$.

Creating a 2D matrix from the 4D diffraction array

We can place a diffraction matrix at each of these reflectivity elements as in Figure 5.

$$\mathbf{Dv}^{\mathbf{T}} = D_{11} \quad D_{21} \quad D_{31} \quad D_{21} \quad D_{22} \quad D_{23} \quad \boxed{D_{31}} \quad D_{32} \quad D_{33} \quad D_{41} \quad D_{42} \quad D_{43} \quad D_{51} \quad D_{52} \quad D_{53}$$

FIG. 5 Diffraction matrices placed at the location of the reflectivity vector.

We now convert each diffraction matrix $\mathbf{D_{kl}}$ to a column vector as described above to complete the process of converting the 4D diffraction array into a 2D matrix. For example, diffraction $\mathbf{D_{13}}$, identified by the green square above, becomes a column vector, which is plotted as a row vector in Figure 6 for convenience. The complete 2D diffraction matrix $\mathbf{D_{2D}}$ is displayed in Figure 7, with $\mathbf{D_{13}}$ highlighted as a column.

$$\mathbf{D}_{13}{}^{T} = \begin{matrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{matrix}$$

FIG. 6 Vectorized form of the $\mathbf{D}_{13}$ diffraction matrix.

$$\mathbf{D}_{2D} = \begin{matrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}$$

FIG. 7 Complete 2D form of the 4D diffraction matrix.

Creating the seismic data

Using the reduced forms of the 2D diffraction matrix $\mathbf{D}_{2D}$, and the vector form of the reflectivity $\mathbf{r}_v$, we can define the seismic data $\mathbf{s}_v$ from the linear equation

$$\mathbf{s}_v = \mathbf{D}_{2D}\mathbf{r}_v \tag{2}$$

The seismic data resulting from forward modelling is converted back to a matrix form $\mathbf{s}$ using the reverse process of unwrapping, and is shown in Figure 8.

$$\mathbf{s} = \begin{matrix}
0 & 2 & 0 & 0 & 0 \\
2 & 0 & 2 & 4 & 4 \\
0 & 0 & 4 & 2 & 0 \\
0 & 4 & 0 & 0 & 2
\end{matrix}$$

FIG. 8 The modelled seismic section.

The notation of the diffraction data in an uppercase letter (**D**) is intended to convey it as a matrix, while the lower case letters for the reflectivity and seismic data (**r** and **s**) are intended to convey them as vectors. In these forms, they can be processes with conventional Linear Algebra.

The migration

We will now migrate the data in an attempt to recover the reflectivity. This is accomplished with the transverse of the 2D diffraction array $\mathbf{D_{2d}}^{\mathbf{T}}$, i.e.,

$$\mathbf{m} = \mathbf{D_{2D}}^{\mathbf{T}}\mathbf{s_v} \tag{3}$$

The migration result is displayed in Figure 9. It is very different from the reflectivity matrix in Figure 4a. The migrated amplitudes at the scatterpoints are larger, and there is noise contaminating all the other scatterpoints. However, the amplitudes at the scatterpoints locations do have a slight relative similarity.

```
        4   10    4    6   12
m  =    6    8    6   10   16
        4    8    6    6   10
```

FIG. 9 The migrated matrix.

The inversion

We continue our processing with Linear Algebra to recover the reflectivity using a least squares approach (Yousefzadeh 2012). The least squares solution is

$$\left(\mathbf{D_{2D}}^{\mathbf{T}}\mathbf{D_{2D}}\right)\mathbf{r_v} = \mathbf{D_{2D}}^{\mathbf{T}}\mathbf{s_v}\,, \tag{4}$$

or

$$\mathbf{r_v} = \left(\mathbf{D_{2D}}^{\mathbf{T}}\mathbf{D_{2D}}\right)^{-1}\mathbf{D_{2D}}^{\mathbf{T}}\mathbf{s_v}\,, \tag{5}$$

where some form of stabilization is required to invert $\mathbf{D^TD}$ matrix. The result is virtually the same as the initial reflectivity and is displayed below with “e” format to show how small the zero values are.

```
      -2.2204e-016    2.0000e+000    2.6645e-015   -6.6613e-016   -6.6613e-016
Inv = -8.4377e-015    9.4369e-015   -4.4409e-015   -1.7764e-015    4.0000e+000
       1.0936e-014   -8.4377e-015   -1.5543e-015    8.8818e-016    2.8866e-015
```

FIG. 10 The least squares inversion for the reflectivity.

Code

Code to compute the inversion in MATLAB computes the $\mathbf{D^TD}$ matrix with,

```
      DTD = D2d'*D2d; ,
```

inverts it with

```
      DTDI = inv(DTD); ,
```

then computes the reflectivity vector with

```
     Rvec = DTDI*(D2d'*S1d); ,
```

and then converts the reflectivity vector to the reflectivity matrix with

```
   R = reshape(Rvec, 3,5); .
```

MATLAB prefers to use its internal inversion process with a black-slash operation

```
     Rvec = DTD\(D2d'*S1d); ,
```

that is apparently more stable than the previous method we used for the inversion.

We included this last piece of code to emphasize that the data was in a vector-matrix format that required the reflectivity to be converted from a vector, back into its correct form as a matrix. In prestack and or 3D data, the reflectivity would be a 3D array, the seismic data would have a higher dimension, and the diffraction matrix 3w2ould have an even higher dimension.

All the above processes, using unwrapped data, are a well-established procedures (Yousefzadeh 2012). It is "hoped" that these processing sequences can be accomplished without the need to unwrap the data.

MULTIDIMENSIONAL PROCESSES

It would be simpler to accomplish the above tasks without the need to unwrap or reduce the data to matrix and vector formats. An example would be to convert the 4D diffraction matrix into a transpose of the 4D matrix. This is not possible with Linear Algebra in its current form as the transpose is only defined for a 2D matrix. However, we can define the 2D transpose of a multidimensional array to suit a particular application. For example, the 4D diffraction array can be transposed into a 4D array for migration.

The following Figure 11a shows a 3D diffraction array in (x, z, t) space where the vertical slices represent the diffractions at different depths in z. In a constant velocity medium, the diffractions have a hyperbolic shape, and form on the surface of a cone. These diffractions are the impulse responses, or Green functions, for modelling.

The cone can be rotated (transposed) as displayed in Figure 11b where the vertical slices are now semi-circles, which are the impulse responses for migration.

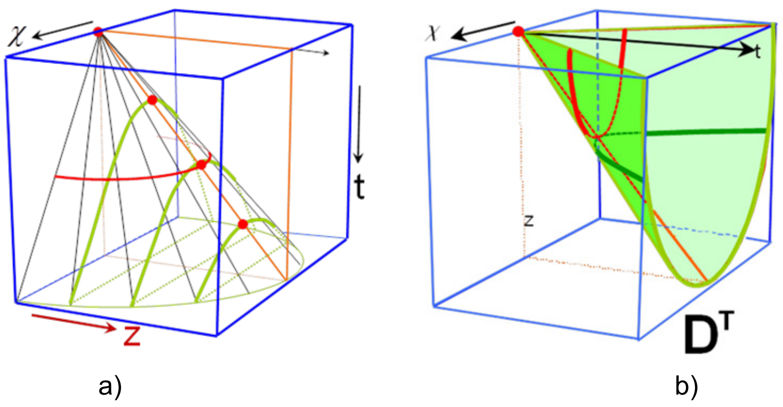


FIG. 11 Three dimensional arrays showing in a) diffractions as vertical slices, and b) the transpose of (a) showing the vertical slices as migration semi-circles.

In Figure 11, it is only the *t* and *z* axis that are transposed.

The 4D diffraction array in Figure 3 can be transposed in a similar manner with the following code that only transposes the *i* and *k* elements:

```
for i = 1:I
    for k = 1:K
        Dt(k,:,i,:) = D(i,:,k,:);
    end
end
```

The result is displayed in Figure 12, where the semi-circles are approximated with box like shapes.

$$
\begin{array}{cc}
 & \qquad j \qquad\qquad\qquad\qquad\qquad l \\
i & \left[\begin{array}{ccccc}
k\begin{bmatrix}1&.&.&.&.\\.&.&.&.&.\\.&.&.&.&.\end{bmatrix} & \begin{bmatrix}.&1&.&.&.\\.&.&.&.&.\\.&.&.&.&.\end{bmatrix} & \begin{bmatrix}.&.&1&.&.\\.&.&.&.&.\\.&.&.&.&.\end{bmatrix} & \begin{bmatrix}.&.&.&1&.\\.&.&.&.&.\\.&.&.&.&.\end{bmatrix} & \begin{bmatrix}.&.&.&.&1\\.&.&.&.&.\\.&.&.&.&.\end{bmatrix} \\
\begin{bmatrix}.&1&.&.&.\\1&1&.&.&.\\.&.&.&.&.\end{bmatrix} & \begin{bmatrix}1&.&1&.&.\\1&1&1&.&.\\.&.&.&.&.\end{bmatrix} & \begin{bmatrix}.&1&.&1&.\\.&1&1&1&.\\.&.&.&.&.\end{bmatrix} & \begin{bmatrix}.&.&1&.&1\\.&.&1&1&1\\.&.&.&.&.\end{bmatrix} & \begin{bmatrix}.&.&.&1&.\\.&.&.&1&1\\.&.&.&.&.\end{bmatrix} \\
\begin{bmatrix}.&.&1&.&.\\.&.&1&.&.\\1&1&1&.&.\end{bmatrix} & \begin{bmatrix}.&.&.&1&.\\.&.&.&1&.\\1&1&1&1&.\end{bmatrix} & \begin{bmatrix}1&.&.&.&1\\1&.&.&.&1\\1&1&1&1&1\end{bmatrix} & \begin{bmatrix}.&1&.&.&.\\.&1&.&.&.\\.&1&1&1&1\end{bmatrix} & \begin{bmatrix}.&.&1&.&.\\.&.&1&.&.\\.&.&1&1&1\end{bmatrix} \\
\begin{bmatrix}.&.&.&1&.\\.&.&.&1&.\\.&.&.&1&.\end{bmatrix} & \begin{bmatrix}.&.&.&.&1\\.&.&.&.&1\\.&.&.&.&1\end{bmatrix} & \begin{bmatrix}.&.&.&.&.\\.&.&.&.&.\\.&.&.&.&.\end{bmatrix} & \begin{bmatrix}1&.&.&.&.\\1&.&.&.&.\\1&.&.&.&.\end{bmatrix} & \begin{bmatrix}.&1&.&.&.\\.&1&.&.&.\\.&1&.&.&.\end{bmatrix}
\end{array}\right]
\end{array}
$$

FIG. 12 The 2D transpose of the 4D diffraction array in Figure 3. Only the *i* and *k* elements have been transposed.

Migration of the multidimensional arrays is then accomplished with the following code:

```
for i = 1:I              % I = 4
    for l = 1:L          % L = 5
        Mig(:,:) = Mig(:,:) + S(i,l)*Dt(:,:,i,l);
    end
end
```

The results are identical to the previous migration in Figure 9 and are not shown.

The code to accomplish these tasks can be inserted into functions that replicate those in Linear Algebra. To some extent, this has been accomplished in the field of Multilinear Algebra (Hoffman 1971, and Greub1967). A brief note on Multilinear Algebra is contained in an Appendix.

It is anticipated that Multilinear Algebra can be used successfully to process seismic data, especially when using the full waveform modelling of diffractions.

COMMENTS

Defining the data format for use with Multilinear Algebra may difficult.

The storage of data within a computer is basically one dimensional and may already be in an unwrapped form, such as a shot record, or a collection of shot records. It will be the efficient access to these data that will establish the best performance of the multilinear or linear processes.

Will algorithms for Multilinear Algebra be available in MATLAB or other software platforms?

The inversion process may not be computed directly, but may use iterative algorithms such as the conjugate gradient method.

The matrix method poses significant storage problems that limit the size of the applications. The trivial example above contains 5 traces with 4 samples. A typical 2D diffraction will be defined with 200 traces, each with 1,000 samples. If we have 800 traces, each with 1,000 samples, then we need to define 800,000 diffractions. We will require about 1.6×10^{11} samples to store this information. Let's think for a fraction of a millisecond about the number of bytes for each sample, extending the problem to 3D, and then prestack 3D where we are talking about 5×10^{16} bytes. (A tera byte is 10^{12} bytes, a petabyte is 10^{15} bytes, and an exabyte is 10^{18} bytes).

CONCLUSIONS

Linear algebra can be used to process seismic data. The conventional Linear algebra process requires multidimensional data to be unwrapped to vectors and matrices for use with a least-squares application. It is anticipated that the field of Multilinear algebra will bypass this requirement of allow the efficient programming and processing of seismic data.

ACKNOWLEDGEMENTS

We thank the sponsors of the CREWES project for their support.

REFERENCES

Claerbout, J. F., 1998, Multidimensional recursive filters via a helix, Geophysics 63, 1532-1541.

Yousefzadeh, A., 2013, High resolution seismic imaging using LSM, PhD thesis, Department of Geoscience, University of Calgary.

Hoffman, K.M., and R. Kunze, 1971, Linear Algebra, 2nd Edition, Pearson

Greub, W.H., 1967, Multilinear Algebra. Volume 136, Springer-Verlag

APPENDIX

Brief Notes on Multilinear Algebra and Applications (Rod Blais)

1. Summation Conventions

Given two matrices $A = [a_{ij} \mid i=1,\ldots, I; j=1,\ldots, J]$ and $B = [b_{jk} \mid j=1,\ldots, J; k=1,\ldots, K]$ compatible for multiplication as $AB \equiv C = [c_{ik} \mid i=1,\ldots, I; k=1,\ldots, K]$ where $c_{ik} = \sum_{j=1}^{J} a_{ij} b_{jk}$ is conventially written as $c_{ik} = a_{ij}\, b_{jk}$ when there is no possibility of confusion (otherwise, a note is needed), i.e. for repeated subscripts (and superscripts) in different terms, the summation is assumed. Note that e.g. a_{ii} would stand for diagonal elements and not the trace!

2. Vectorization of Arrays and Matrices

A two-dimensional array or matrix $A = [a_{ij} \mid i=1,\ldots, I; j=1,\ldots, J]$ is normally stored (columnwise) as a one-dimensional array $A \approx [a_n \mid n = 1,\ldots, N]$ with $a_n \equiv a_{ij}$ where $n = (j-1)I + i$ and $N = I \cdot J$. This storage strategy can of course be extended to higher dimensional arrays.

3. Multidimensional Arrays

A three-dimensional array $A = [a_{ijk} \mid i=1,\ldots, I; j=1,\ldots, J; k=1,\ldots,K]$ can be post multiplied by a K-vector $U = [u_k \mid k=1,\ldots,K]$ as simply $a_{ijk}\, u_k$ (with each element being summed over k) to give a two-dimensional array $AU \equiv V = [v_{ij} \mid v_{ij} = \sum_{k=1}^{K} a_{ijk} u_k$, $i=1,\ldots, I; j=1,\ldots, J]$. Note the requirement for the compatible dimension K.

Similarly, a four-dimensional array $A = [a_{ijkh} \mid i=1,\ldots, I; j=1,\ldots, J; k=1,\ldots,K; h=1,\ldots,H]$ can be post-multiplied by a two-dimensional array $U = [u_{kh} \mid k=1,\ldots,K; h=1,\ldots,H]$ as simply $a_{ijkh}\, u_{kh}$ (with each element being summed over k and h) to give a two-dimensional array

$$AU \equiv V = [v_{ij} \mid v_{ij} = \sum_{k=1}^{K}\sum_{h=1}^{H} a_{ijkh} u_{kh} , i=1,\ldots, I; j=1,\ldots, J].$$

Note the requirement for the compatible dimensions K and H.

4. Least-Squares Estimation

Given an overdetermined linear system $A X = B$ with $A = [a_{ij} \mid i=1,\ldots, I; j=1,\ldots, J]$ with $I > J$, $X = [x_j \mid j=1,\ldots,J]$ and $B = [b_i \mid i=1,\ldots,I]$ i.e. with $a_{ij}\, x_j = b_i$, the simplest least-squares solution $\hat{X} = (A^T A)^{-1} A^T B$ corresponds to solving $a^T_{ki} a_{ij} x_j = a^T_{ki} b_i$ or simply $a_{ik} a_{ij} x_j = a_{ik} b_i$.

Starting with a four-dimensional system $G Y = F$ with $G = [g_{ijkh} \mid i=1,\ldots,I; j=1,\ldots,J; k=1,\ldots,K; h=1,\ldots,H]$, $Y = [y_{kh} \mid k=1,\ldots,K; h=1,\ldots,H]$ and $F = [f_{ij} \mid i=1,\ldots,I; j=1,\ldots,J]$ overdetermined for the unknowns, then vectorizing the array Y implies that the k and h indices become a single index $n = (h-1)K + k$. Then the system becomes $g_{ijn}\, y_n = f_{ij}$ which can also be vectorized with respect to i and j giving a single index $m = (j-1)I + i$ and the system becomes simply $g_{mn}\, y_n = f_m$, an overdetermined matrix system of linear equations. Hence the previous least-squares solution can be readily obtained as above.