Chapter
II-6

Multidimensional Waves

Overview

Chapter II-5, **Waves**, concentrated on one-dimensional waves consisting of a number of rows. In Chapter II-5, **Waves**, the rows were referred to as "points" and the symbol p stood for row number, which was called "point number". Scaled row numbers were called X values and were represented by the symbol x.

This chapter now extends the concepts from Chapter II-5, **Waves**, to waves of up to four dimensions by adding the column, layer and chunk dimensions. The symbols q, r and s stand for column, layer and chunk numbers. Scaled column, layer and chunk numbers are called Y, Z and T values and are represented by the symbols y, z and t.

We call a two-dimensional wave a "matrix"; it consists of rows (the first dimension) and columns (the second dimension). After two dimensions the terminology becomes a bit arbitrary. We call the next two dimensions "layers" and "chunks".

Here is a summary of the terminology:

Dimension Number	0	1	2	3
Dimension Name	row	column	layer	chunk
Dimension Index	p	q	r	s
Scaled Dimension Index	x	y	z	t

Each element of a 1D wave has one index, the row index, and one data value.

Each element of a 2D wave has two indices, the row index and the column index, and one data value.

Each element of a 3D wave has three indices (row, column, layer) and one data value.

Each element of a 4D wave has four indices (row, column, layer, chunk) and one data value.

Creating Multidimensional Waves

Multidimensional waves can be created using the Make operation:

```
Make/N=(numRows,numColumns,numLayers,numChunks) waveName
```

When making an N-dimensional wave, you provide N values to the /N flag. For example:

```
// Make a 1D wave with 20 rows (20 points total)
Make/N=20 wave1

// Make a matrix (2D) wave with 20 rows and 3 columns (60 elements total)
Make/N=(20,3) wave2
```

The Redimension operation's /N flag works the same way.

```
// Change both wave1 and wave2 so they have 10 rows and 4 columns
Redimension/N=(10,4) wave1, wave2
```

The operations InsertPoints and DeletePoints take a flag (/M=*dimensionNumber*) to specify the dimension into which elements are inserted. For example:

```
InsertPoints/M=1 2,5,wave2    //M=1 means column dimension
```

This command inserts 5 new columns in front of column number 2. If the /M=1 had been omitted or if /M=0 had been used then 5 new rows would have been inserted in front of row number 2.

You can also create multidimensional waves using the Make operation with a list of data values. For example:

```
// Create a 1D wave consisting of a single column of 3 rows
Make wave1 = {1,2,3}
```

```
// Creates a 2D wave consisting of 3 rows and 2 columns
Make wave2 = {{1,2,3},{4,5,6}}
```

The Duplicate operation can create an exact copy of a multidimensional wave or, using the /R flag, extract a subrange. Here is the syntax of the /R flag:

```
Duplicate/R=[startRow,endRow][startCol,endCol] and so on...
```

You can use the character * for any end field to specify the last element in the given dimension or you can just omit the end field. You can also specify just [] to include all of a given dimension. If the source wave has more dimensions than you specify in the /R flag, then all of those dimensions are copied. For example:

```
// Make a 3D wave to play with
Make/N=(5,4,3) wave3A = p + 10*q + 100*r

// Duplicate rows 1 through 2, columns 2 through the end, and all layers
Duplicate/R=[1,2][2,*] wave3A, wave3B     // 2 rows, 2 columns, 3 layers

// Create a 3D wave consisting of all rows of column 2, layer 0
Duplicate/R=[][2,2][0,0] wave3A, wave3C   // 5 rows, 1 column, 1 layer
```

Igor considers wave3C to be 3 dimensional wave and not 1 dimensional, even though it consists of just one column of data, because the number of columns and layers are greater than zero. This is a subtle distinction and can cause confusion. For example, you may think you have extracted a 1D wave from a 3D object but you will find that wave3C will not show up in dialogs where 1D waves are required.

You can turn the 3D wave wave3C into a 1D wave using the following command:

```
Redimension/N=(-1,0) wave3C
```

The -1 value does not to change the number of rows whereas the 0 value for the number of columns indicates that there are no dimensions past rows (in other words, no columns, layers or chunks).

Programmer Notes

For historical reasons, you can treat the symbols x and p like global variables, meaning that you can store into them as well as retrieve their values by referencing them. But this serves no purpose and is not recommended.

Unlike x and p, y, z, t, q, r and s act like functions and you can't store into them.

Here are some functions and operations that are useful in programming with multidimensional waves:

```
DimOffset, DimDelta, DimSize
FindDimLabel, SetDimLabel, GetDimLabel
```

Dimension Labels

A dimension label is a name associated with a dimension (rows, columns, layers or chunks) or with a specific dimension index (row number, column number, layer number or chunk number).

Dimension labels are primarily an aid to the Igor procedure programmer when dealing with waves in which certain elements have distinct purposes. Dimension labels can be set when loading from a file, and can be displayed, created or edited in a table (see **Showing Dimension Labels** on page II-170).

You can give names to individual dimension indices in multidimensional or 1D waves. For example, if you have a 3 column wave, you can give column 0 the name "red", column 1 the name "green" and column 2 the name "blue". You can use the names in wave assignments in place of literal numbers. To do so, you use the % symbol in front of the name. For example:

```
wave2D[][%red] = wave2D[p][%green]     //Set red column equal to green column
```

To create a label for a given index of a given dimension, use the SetDimLabel operation.

For example:

```
SetDimLabel 1, 0, red, wave2D
```

1 is the dimension number (columns), 0 is the dimension index (column 0) and red is the label.

The function GetDimLabel returns a string containing the name associated with a given dimension and index. For example:

```
Print GetDimLabel(wave2D,1,0)
```

prints “red” into the history area.

The FindDimLabel function returns the index value associated with the given label. It returns the special value -2 if the label is not found. This function is useful in user-defined functions so that you can use a numeric index instead of a dimension label when accessing a wave in a loop. Accessing wave data using a numeric index is much faster than using a dimension label.

In addition to setting the name for individual dimension index values, you can set the name for an entire dimension by using an index value of -1. For example:

```
SetDimLabel 1, -1, ColorComponents, wave2D
```

This sets the label for the columns dimension to “ColorComponents”. This label appears in a table if you display dimension labels.

Dimension names can contain up to 31 bytes and may contain spaces and other normally illegal characters if you surround the name in single quotes or if you use the $ operator to convert a string expression to a name. For example:

```
wave[%'a name with spaces']
wave[%$"a name with spaces"]
```

Dimension names have the same characteristics as object names. See **Object Names** on page III-443 for a discussion of object names in general.

Graphing Multidimensional Waves

You can easily view two-dimensional waves as images and as contour plots using Igor’s built-in operations. See Chapter II-14, **Contour Plots**, and Chapter II-15, **Image Plots**, for further information about these types of graphs. You can also create waterfall plots where each column in the matrix wave corresponds to a separate trace in the waterfall plot. For more details, see **Waterfall Plots** on page II-255.

Additional facilities for displaying multi-dimensional waves in Igor Pro are provided by the Gizmo extension, which create surface plots, slices through volumes and many other 3D plots. To get started with Gizmo, see **3D Graphics** on page II-317.

It is possible to graph a subset of a wave, including graphing rows or columns from a multidimensional wave as traces. See **Subrange Display** on page II-250 for details.

Analysis on Multidimensional Waves

Igor Pro includes the following capabilities for analysis of multidimensional data:

- Multidimensional waveform arithmetic
- Matrix math operations
- Image processing
- Multidimensional Fast Fourier Transform
- The **MatrixOp** operation

There are many analysis operations for 1D data that we have not yet extended to support multiple dimensions. Multidimensional waves do not appear in dialogs for these operations. If you invoke them on multidimensional

waves from the command line or from an Igor procedure, Igor treats the multidimensional waves as if they were 1D. For example, the Smooth operation treats a 2D wave consisting of n rows and m columns as if it were a 1D wave with n*m rows. In some cases the operation will be useful. In other cases, it will make no sense.

Multidimensional Wave Indexing

You can use multidimensional waves in wave expressions and assignment statements just as you do with 1D waves (see **Indexing and Subranges** on page II-71). To specify a particular element of a 4D wave, use the syntax:

`wave[rowIndex][columnIndex][layerIndex][chunkIndex]`

Similarly, to specify an element of a 4D wave using *scaled* dimension indices, use the syntax:

`wave(xIndex)(yIndex)(zIndex)(tIndex)`

To index a 3D wave, omit the chunk index. To index a 2D wave, omit the layer and chunk indices.

rowIndex is the number, starting from zero, of the row of interest. It is an unscaled index. *xIndex* is simply the row index, offset and scaled by the wave's X scaling property, which you set using the SetScale operation (Change Wave Scaling in Data menu).

Using scaled indices you can access the wave's data using its natural units. You can use unscaled or scaled indices, whichever is more convenient. column/Y, layer/Z and chunk/T indices are analogous to row/X indices.

Using bracket notation tells Igor that the index you are supplying is an unscaled dimension index. Using parenthesis notation tells Igor that you are supplying a scaled dimension index. You can even mix the bracket notation with parenthesis notation.

Here are some examples:

```
Make/N=(5,4,3) wave3D = p + 10*q + 100*r
SetScale/I x, 0, 1, "", wave3D
SetScale/I y, -1, 1, "", wave3D
SetScale/I z, 10, 20, "", wave3D
Print wave3D[0][1][2]
Print wave3D(0.5)[2](15)
```

The first Print command prints 210, the value in row 0, column 1 and layer 2. The second Print command prints 122, the value in row 2 (where x=0.5), column 2 and layer 1 (where z=15).

Since wave3D has three dimensions, we do not, and must not, specify a chunk index.

There is one important difference between wave access using 1D waves versus multidimensional waves. For 1D waves alone, Igor performs linear interpolation when the specified index value, whether scaled or unscaled, falls between two points. For multidimensional waves, Igor returns the value of the element whose indices are closest to the specified indices.

When a multidimensional wave is the destination of a wave assignment statement, you can specify a subrange for each dimension. You can specify an entire dimension by using []. For example:

```
wave3D[2][][1,2] = 3
```

This sets row 2 of all columns and layers 1 and 2 to the value 3.

Note that indexing of the form [] (entire dimension) or [1,2] (range of a dimension) can be used on the left-hand side only. This is because the indexing on the left side determines which elements of the destination are to be set whereas indexing on the right side identifies a particular element in the source which is to contribute to a particular value in the destination.

Multidimensional Wave Assignment

As with one-dimensional waves, you can assign a value to a multidimensional wave using a wave assignment statement. For example:

```
Make/O/N=(3,3) wave0_2D, wave1_2D, wave2_2D
wave1_2D = 1.0; wave2_2D = 2.0
wave0_2D = wave1_2D / wave2_2D
```

The last command sets all elements of wave0_2D equal to the quotient of the corresponding elements of wave1_2D and wave2_2D.

Important: Wave assignments as shown in the above example where waves on the right-hand side do not include explicit indexing are defined only when all waves involved have the same dimensionality. The result of the following assignment is undefined and may produce surprising results.

```
Make/O/N=(3,3) wave33
Make/O/N=(2,2) wave22
wave33 = wave22
```

Whenever waves of mismatched dimensionality are used you should specify explicit indexing as described next.

In a wave assignment, Igor evaluates the right-hand side one time for each element specified by the left-hand side. During this evaluation, the symbols p, q, r and s take on the value of the row, column, layer and chunk, respectively, of the element in the destination for which a value is being calculated. For example:

```
Make/O/N=(5,4,3) wave3D = 0
Make/O/N=(5,4) wave2D = 999
wave3D[][][0] = wave2D[p][q]
```

This stores the contents of wave2D in layer 0 of wave3D. In this case, the destination (wave3D) has three dimensions, so p, q and r are defined and s is undefined. The following discussion explains this assignment and presents a way of thinking about wave assignments in general.

The left-hand side of the assignment specifies that Igor is to store a value into all rows (the first []) and all columns (the second []) of layer zero (the [0]) of wave3D. For each of these elements, Igor will evaluate the right-hand side. During the evaluation, the symbol p will return the row number of the element in wave3D that Igor is about to set and the symbol q will return the column number. The symbol r will have the value 0 during the entire process. Thus, the expression wave2D[p][q] will return a value from wave2D at the corresponding row and column in wave3D.

As the preceding example shows, wave assignments provide a way of transferring data between waves. With the proper indexing, you can build a 2D wave from multiple 1D waves or a 3D wave from multiple 2D waves. Conversely, you can extract a layer of a 3D wave into a 2D wave or extract a column from a 2D wave into a 1D wave. Here are some examples that illustrate these operations.

```
// Build a 2D wave from multiple 1D waves (waveforms)
Make/O/N=5, wave0=p, wave1=p+1, wave2=p+2    // 1D waveforms
Make/O/N=(5,3) wave0_2D
wave0_2D[][0] = wave0[p]      // Store into all rows, column 0
wave0_2D[][1] = wave1[p]      // Store into all rows, column 1
wave0_2D[][2] = wave2[p]      // Store into all rows, column 2

// Build a 3D wave from multiple 2D waves
Duplicate/O wave0_2D, wave1_2D; wave1_2D *= -1
Make/O/N=(5,3,2) wave0_3D
wave0_3D[][][0]= wave0_2D[p][q]  // Store into all rows/cols, layer 0
wave0_3D[][][1]= wave1_2D[p][q]  // Store into all rows/cols, layer 1

// Extract a layer of a 3D wave into a 2D wave
wave0_2D = wave0_3D[p][q][0]     // Extract layer 0 into 2D wave
```

```
// Extract a column of a 2D wave into a 1D wave
wave0 = wave0_2D[p][0]           // Extract column 0 into 1D wave
```

To understand assignments like these, first figure out, by looking at the indexing on the left-hand side, which elements of the destination wave are going to be set. (If there is no indexing on the left then all elements are going to be set.) Then think about the range of values that p, q, r and s will take on as Igor evaluates the right-hand side to get a value for each destination element. Finally, think about how these values, used as indices on the right-hand side, select the desired source element.

To create such an assignment, first determine the indexing needed on the left-hand side to set the elements of the destination that you want to set. Then think about the values that p, q, r and s will take on. Then use p, q, r and s as indices to select a source element to be used when computing a particular destination element.

Here are some more examples:

```
// Extract a row of a 2D wave into a 1D wave
Make/O/N=3 row1
row1 = wave0_2D[1][p]          // Extract row 1 of the 2D wave
```

In this example, the *row* index (p) for the destination is used to select the source *column* while the source row is always 1.

```
// Extract a horizontal slice of a 3D wave into a 2D wave
Make/O/N=(2,3) slice_R2        // Slice consisting of all of row 2
slice_R2 = wave0_3D[2][q][p]   // Extract row 2, all columns/layers
```

In this example, the row data for slice_R2 comes from the layers of wave0_3D because the p symbol (row index) is used to select the layer in the source. The column data for slice_R2 comes from the columns of wave0_3D because the q symbol (column index) is used to select the column in the source. All data comes from row 2 in the source because the row index is fixed at 2.

You can store into a range of elements in a particular dimension by using a range index on the left-hand side. As an example, here are some commands that shift the horizontal slices of wave0_3D.

```
Duplicate/O wave0_3D, tmp_wave0_3D
wave0_3D[0][][] = tmp_wave0_3D[4][q][r]
wave0_3D[1,4][][] = tmp_wave0_3D[p-1][q][r]
KillWaves tmp_wave0_3D
```

The first assignment transfers the slice consisting of all elements in row 4 to row zero. The second assignment transfers slice n-1 to slice n. To understand this, realize that as p goes from 1 to 4, p-1 indexes into the preceding row of the source.

Vector (Waveform) to Matrix Conversion

Occasionally you will may need to convert between a vector form of data and a matrix form of the same data values. For example, you may have a vector of 16 data values stored in a waveform named sixteenVals that you want to treat as a matrix of 8 rows and 2 columns.

Though the Redimension operation normally doesn't move data from one dimension to another, in the special case of converting to or from a 1D wave Redimension will leave the data in place while changing the dimensionality of the wave. You can use the command:

```
Make/O/N=16 sixteenVals          // 1D
Redimension/N=(8,2) sixteenVals  // Now 2D, no data lost
```

to accomplish the conversion. When redimensioning from a 1D wave, columns are filled first, then layers, followed by chunks. Redimensioning from a multidimensional wave to a 1D wave doesn't lose data, either.

Matrix to Matrix Conversion

To convert a matrix from one matrix form to another, don't directly redimension it to the desired form. For instance, if you have a 6x6 matrix wave, and you would like it to be 3x12, you might try:

```
Make/O/N=(6,6) thirtySixVals        // 2D
Redimension/N=(3,12) thirtySixVals  // This loses the last three rows
```

But Igor will first shrink the number of rows to 3, discarding the data for the last three rows, and then add 6 columns of zeroes.

The simplest way to work around this is to convert the matrix to a 1D vector, and then convert it to the new matrix form:

```
Make/O/N=(6,6) thirtySixVals        // 2D
Redimension/N=36 thirtySixVals      // 1D vector preserves the data
Redimension/N=(3,12) thirtySixVals  // Data preserved
```

Multidimensional Fourier Transform

Igor's FFT and IFFT routines are mixed-radix and multidimensional. Mixed-radix means you do not need a power of two number of data points (or dimension size).

There is only one restriction on the dimensions of a wave: when performing a forward FFT on real data, the number of rows must be even. Note, however, that if a given dimension size is a prime number or contains a large prime in its factorization, the speed will be reduced to that of a normal Discrete Fourier Transform (i.e., the number of operations will be on the order of N^2 rather than N*log(N)).

For more information about the FFT, see **Fourier Transforms** on page III-239 and the **FFT** operation on page V-190.

Treating Multidimensional Waves as 1D

Sometimes it is useful to treat a multidimensional wave as if it were 1D. For example, if you want to know the number of NaNs in a 2D wave, you can pass the wave to **WaveStats**, even though WaveStats treats its input as 1D.

In other cases, you need to understand the layout of data in memory in order to treat a multidimensional wave as 1D.

A 2D wave consists of some number of columns. In memory, the data is laid out column-by-column. This is called "column-major order". In column-major order, consecutive elements of a given column are contiguous in memory.

For example, execute:

```
Make/N=(2,2) mat = p + 2*q
Edit mat
```

The wave mat consists of two columns. The first column contains the values 0 and 1. The second column contains the values 2 and 3.

You can pass this wave to an operation or function that is not multidimensional-aware and it will treat the wave as if it were one column containing 0, 1, 2, 3. For an example:

```
Print WaveMax(mat)      // Prints 3
```

Here is an example of using the knowledge of how a multidimensional wave is laid out in memory:

```
Function DemoMDAs1D()
   // Make a 2D wave
   Make/O/N=(5,3) mat = p + 10*q
```

```
   Variable numRows = DimSize(mat,0)

   // Find the sum of each column
   Variable numColumns = DimSize(mat,1)
   Make/O/N=(numColumns) Sums
   Sums = sum(mat, p*numRows, (p+1)*numRows-1)
   Edit mat, Sums
End
```

The statement

```
Sums = sum(mat, p* numRows, (p+1)* numRows-1)
```

passes the 2D wave mat to the **sum** function which is not multidimensional-aware. Because sum is not multidimensional-aware, it requires that we formulate the startX and endX parameters treating mat as if it were a 1D wave with the data arranged in column-major order.

You can also treat 3D and 4D waves as 1D. In a 3D wave, the data for layer n+1 follows the data for layer n. In a 4D wave, the data for chunk n+1 follows the data for chunk n.