CONTENTS

ビジュアルヘルプ – 多次元ウェーブ

「ウェーブ」のヘルプでは、複数の行からなる一次元ウェーブに焦点を当てています。
これらの行は「ポイント」と呼ばれ、記号 p は行番号を表し、これは「ポイント番号」と呼ばれます。
スケーリングされた行番号は X 値と呼ばれ、記号 x で表されます。

このヘルプでは、列、レイヤー、チャンクの次元を追加することで、これまでの概念を最大 4 次元までのウェーブへと拡張します。
記号 q、r、s は、それぞれ列番号、レイヤー番号、チャンク番号を表します。
スケーリングされた列番号、レイヤー番号、チャンク番号は、Y 値、Z 値、T 値と呼ばれ、記号 y、z、t で表されます。

2 次元のウェーブを「行列（マトリックスとも）」と呼びます。
これは行（第 1 次元）と列（第 2 次元）から構成されています。
2 次元を超えると、用語の定義は多少恣意的になります。
次の 2 次元については、「レイヤー」と「チャンク」と呼びます。

用語の概要は以下の通りです。

次元番号	0	1	2	3
次元名	行	列	レイヤー	チャンク
次元インデックス	p	q	r	s
スケーリングされた次元インデックス	x	y	z	t

1D ウェーブの各要素には、行インデックスという 1 つのインデックスと、 1 つのデータ値があります。

2D ウェーブの各要素には、行インデックスと列インデックスの 2 つのインデックスと、 1 つのデータ値があります。

3D ウェーブの各要素には、 3 つのインデックス（行、列、レイヤー）と 1 つのデータ値があります。

4D ウェーブの各要素は、 4 つのインデックス（行、列、レイヤー、チャンク）と 1 つのデータ値で構成されます。

多次元ウェーブの作成

Make コマンドを使うと、多次元ウェーブを作成できます。

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

N 次元のウェーブを作成する時は、/N フラグに N 個の値を指定します。
例えば：

```
// 20 行（計 20 ポイント）の 1D ウェーブの作成
Make/N=20 wave1

// 20 行 3 列（計 60 要素）の行列（2D）ウェーブの作成
Make/N=(20,3) wave2
```

Redimension コマンドの /N フラグも同様に機能します。

```
// wave1 と wave2 の両方を、10 行、4 列になるように変更する
Redimension/N=(10,4) wave1, wave2
```

InsertPoints と DeletePoints コマンドでは、要素を挿入する次元を指定するためにフラグ（/M=dimensionNumber）を指定します。
例えば：

```
InsertPoints/M=1 2,5,wave2      // M=1 は列の次元を意味する
```

このコマンドは、２番目の列の前に５つの新しい列を挿入します。
/M=1 を省略した場合、または /M=0 を使った場合、２番目の行の前に５つの新しい行が挿入されます。

データ値のリストを使って Make コマンドを実行することで、多次元ウェーブを作成することもできます。
例えば：

```
// ３行１列で構成される 1D ウェーブを作成
Make wave1 = {1,2,3}

// ３行２列で構成される 2D ウェーブを作成
Make wave2 = {{1,2,3},{4,5,6}}
```

中括弧の使用に関する詳しい説明については、ヘルプ Lists of Values を参照してください。

Duplicate コマンドでは、多次元ウェーブの完全なコピーを作成したり、/R フラグを使ってサブレンジ（一部）を抽出したりすることができます。
/R フラグの構文は以下の通りです。

```
Duplicate/R=[startRow,endRow][startCol,endCol] +その他のオプション
```

末尾のフィールドには、指定した次元の最後の要素を指定するために「*」を使うか、末尾のフィールドを省略することができます。
また、指定した次元全体を含めるには、単に「 [] 」を指定することもできます。
ソースのウェーブが /R フラグで指定した数よりも多くの次元を持っている場合、それらの次元はすべてコピーされます。
例えば：

```
// 試してみるための 3D ウェーブを作成
Make/N=(5,4,3) wave3A = p+ 10*q + 100*r

// １～２行目まで、２列目から末尾まで、およびすべてのレイヤーを複製
Duplicate/R=[1,2][2,*][] wave3A, wave3B        // 2 rows, 2 columns, 3 layers

// レイヤー 0 の列 2 のすべての行で構成される 3D のウェーブを作成
Duplicate/R=[][2,2][0,0] wave3A, wave3C        // 5 rows, 1 column, 1 layer
```

Igor Pro は、wave3C がデータ列を１つしか持たないにもかかわらず、列数とレイヤー数が 0 より多いため、これを１次元ではなく３次元と見なします。
これは微妙な違いであり、混乱を招く可能性があります。
例えば、3D オブジェクトから 1D のウェーブを抽出したつもりでも、1D のウェーブが必要なダイアログでは wave3C が表示されません。

次のコマンドを使うと、3D ウェーブ wave3C を 1D ウェーブに変換できます。

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

値「-1」は、Redimension コマンドにおいて行数を変更しないことを指定し、列数の値「0」は、行を超える次元が存在しないこと（つまり、列、レイヤー、チャンクがないこと）を示します。

プログラマー向けメモ

歴史的な理由から、記号 x や p をグローバル変数のように扱うことができます。
つまり、これらに値を格納したり、参照して値を取得したりすることが可能です。
しかし、これには何の意味もなく、推奨されません。

x や p とは異なり、y、z、t、q、r、s は関数のように振る舞い、これらには値を格納することはできません。

多次元ウェーブを用いたプログラミングで役立つ関数やコマンドをいくつか紹介します。

DimOffset　　DimDelta　　DimSize

FindDimLabel　　SetDimLabel　　GetDimLabel

次元ラベル

次元ラベルとは、次元（行、列、レイヤー、またはチャンク）や、特定の次元インデックス（行番号、列番号、レイヤー番号、またはチャンク番号）に関連付けられた名称のことです。

次元ラベルは、主に、特定の要素がそれぞれ異なる目的を持つウェーブを扱う時に、Igor のプロシージャプログラマーを支援するためのものです。
次元ラベルはファイルから読み込む時に設定でき、テーブル内で表示、作成、または編集することができます（ヘルプ Showing Dimension Labels を参照）。

多次元または 1D のウェーブにおいて、個々の次元インデックスに名前を付けることができます。
例えば、3 列のウェーブがある場合、列 0 に「red」、列 1 に「green」、列 2 に「blue」という名前を付けることができます。
ウェーブの代入式では、リテラルな数値の代わりにこれらの名前を使用できます。
これを行うには、名前の前に % 記号を付けます。
例えば：

```
wave2D[][%red] = wave2D[p][%green]              // red の列を green の列と同じにする
```

特定の次元の特定のインデックスにラベルを設定するには、SetDimLabel コマンドを使います。

例えば：

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

1 は次元数（列数）、0 は次元インデックス（0 列目）、red はラベルを表します。

関数 GetDimLabel は、指定された次元とインデックスに関連付けられた名前を含む文字列を返します。
例えば：

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

これは、履歴エリアに「red」と出力します。

FindDimLabel 関数は、指定されたラベルに関連付けられたインデックス値を返します。
ラベルが見つからない場合は、特別な値 -2 を返します。
この関数は、ループ内でウェーブにアクセスする時に、次元ラベルの代わりに数値インデックスを使用できるようにするため、ユーザー定義関数で役立ちます。
数値インデックスを使ってウェーブデータにアクセスする方が、次元ラベルを使う場合よりもはるかに高速です。

個々の次元インデックス値の名前を設定するだけでなく、インデックス値として「-1」を使うことで、次元全体の名前を設定することもできます。
例えば：

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

これにより、列の次元のラベルが「ColorComponents」に設定されます。
次元ラベルを表示している場合、このラベルはテーブルに表示されます。

CopyDimLabels コマンドを使うと、ある次元から別の次元へ、あるいはあるウェーブから別のウェーブへ、次元ラベルをコピーすることができます。

次元ラベルには最大 255 バイトまで指定でき、名前をシングルクオートで囲むか、$ 演算子を使って文字列式を名前に変換すれば、スペースや通常は使用できない文字を含めることができます。
例えば、

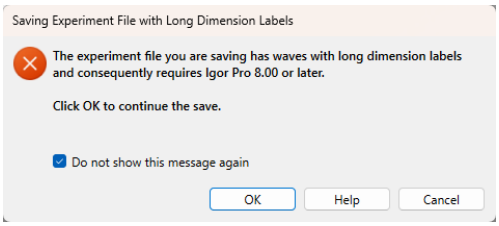
```
wave[%'a name with spaces']
wave[%$"a name with spaces"]
```

次元ラベルは、オブジェクト名と同じ特性を持っています。
オブジェクト名に関する一般的な説明については、ヘルプ Object Names を参照してください。

長い次元ラベル

Igor Pro 8 より前では、ウェーブの次元ラベルの長さは 31 バイトに制限されていました。
31 バイト以下の長さの次元ラベルを作成した場合、Igor Pro 8 以降で保存されたウェーブは、Igor Pro 7 以前と互換性のある形式で保存されます。
31 バイトを超える長さの次元ラベルを作成した場合、そのウェーブは Igor Pro 7 以前と互換性のない形式で保存されます。

長い次元ラベルを持つウェーブを含む Igor バイナリウェーブファイルまたはエクスペリメントファイルを保存しようとすると、Igor Pro は、そのエクスペリメントには Igor Pro 8.0 以降が必要であることを知らせる警告ダイアログを表示します。
この警告ダイアログは、Igor バイナリウェーブファイルまたはエクスペリメントをインタラクティブに保存する場合にのみ表示され、SaveExperiment コマンドを使ってプログラムから保存する場合は表示されません。
Do not show this message again チェックボックスをクリックすることで、このダイアログを表示させないようにすることができます。



多次元ウェーブのグラフ化

Igor Pro の組み込み演算機能を使えば、２次元ウェーブを画像やコンタープロットとして簡単に表示できます。
これらのグラフの種類に関する詳細については、ヘルプ Contour Plots と Image Plots を参照してください。
また、行列ウェーブの各列がウォーターフォールプロット内の個別のトレースに対応するようなウォーターフォールプロットを作成することもできます。
詳細については、ヘルプ Waterfall Plots を参照してください。

Igor Pro では、「Gizmo」機能により、多次元ウェーブの表示に関する追加機能が提供されています。
Gizmo を使うと、サーフェスプロットやボリュームのスライス、その他多くの 3D プロットを作成できます。
Gizmo の使い方を学ぶには、ヘルプ 3D Graphics を参照してください。

ウェーブのサブセットをグラフ化することが可能です。
これには、多次元ウェーブの行や列をトレースとしてグラフ化することも含まれます。
詳細については、ヘルプ Subrange Display を参照してください。

多次元ウェーブでの分析

Igor Pro には、多次元データの分析を行うための以下の機能が含まれています。

- 多次元ウェーブフォーム演算
- 行列の演算
- 画像処理
- 多次元高速フーリエ変換
- MatrixOp コマンド

1D データに対する解析コマンドの多くは、まだ多次元に対応するように拡張されていません。
これらのコマンドのダイアログには、多次元ウェーブは表示されません。
コマンドラインや Igor Pro プロシージャから多次元ウェーブに対してこれらのコマンドを呼び出すと、多次元ウェーブを 1D ウェーブとして扱います。
例えば、Smooth コマンドは、n 行 m 列からなる 2D ウェーブを、n×m 行の 1D ウェーブとして扱います。
場合によっては、このコマンドが役立つこともあります。
しかし、そうでない場合には意味をなさないこともあります。

多次元ウェーブのインデックス付け

多次元ウェーブは、1D ウェーブと同様に、ウェーブ式や代入文で使用できます（ヘルプ Indexing and Subranges を参照）。
4D ウェーブの特定の要素を指定するには、次の構文を使います。

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

同様に、スケーリングされた次元インデックスを使って 4D ウェーブの要素を指定するには、次の構文を使います。

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

3D ウェーブをインデックス付けする場合は、チャンクインデックスを省略してください。
2D ウェーブをインデックス登録する場合は、レイヤーインデックスとチャンクインデックスを省略してください。

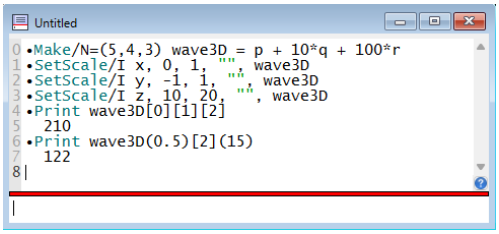
rowIndex は、対象となる行の番号（0 から始まる）です。
これはスケーリングされていないインデックスです。
xIndex は、SetScale コマンド（Data メニューの Change Wave Scaling）で設定したウェーブの X 軸スケーリングプロパティによってオフセットおよびスケーリングされた行インデックスです。

スケーリングされたインデックスを使うと、ウェーブのデータを本来の単位で参照することができます。
スケーリングされていないインデックスとスケーリングされたインデックスのどちらでも、使いやすい方を使用できます。
column/Y、layer/Z、chunk/T インデックスは、row/X インデックスと同じです。

角括弧表記を使うと、指定したインデックスがスケーリングされていない次元インデックスであることを示します。
丸括弧表記を使うと、指定したインデックスがスケーリングされた次元インデックスであることを示します。
角括弧表記と丸括弧表記を混在させることも可能です。

以下にいくつかの例を挙げます：

```
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)
```



最初の Print コマンドは、行 0、列 1、レイヤー 2 にある値「210」を出力します。
２番目の Print コマンドは、行 2（x=0.5）、列 2、レイヤー 1（z=15）にある値「122」を出力します。

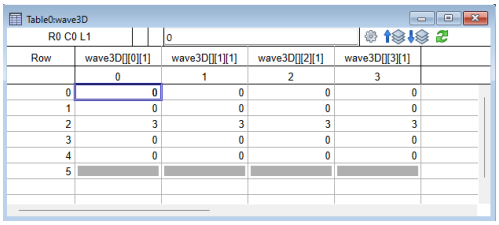
wave3D は 3D であるため、チャンクインデックスを指定することはできず、また指定してはいけません。

1D ウェーブを使ったウェーブアクセスと多次元ウェーブを使ったウェーブアクセスには、１つの重要な違いがあります。
1D ウェーブのみの場合、指定されたインデックス値（スケーリングの有無にかかわらず）が２つのポイントの間に位置する場合、線形補間を行います。
多次元ウェーブの場合、指定されたインデックスに最も近いインデックスを持つ要素の値を返します。

ウェーブ代入文の宛先として多次元ウェーブを指定する場合、各次元に対してサブレンジ（一部）を指定できます。
[] を使うことで、次元全体を指定できます。
例えば：

```
Make/N=(5,4,3) wave3D
wave3D[2][][1,2] = 3
```

これは、すべての列の２行目と、レイヤー1 と 2 の値が 3 に設定されます。

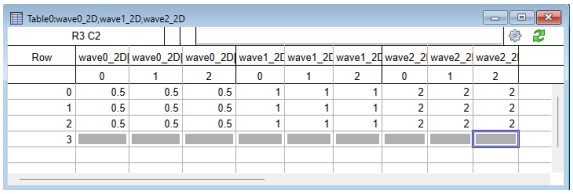


なお、[]（次元全体）や [1,2]（次元の範囲）といったインデックス表記は、左辺でのみ使用できます。
これは、左辺のインデックスが宛先のどの要素を設定するかを決定するのに対し、右辺のインデックスは、宛先の特定の値に寄与するソース内の特定の要素を特定するためです。

多次元ウェーブの代入

１次元ウェーブと同様に、ウェーブの代入文を使って多次元ウェーブに値を代入することができます。
例えば：

```
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
```



最後のコマンドは、wave0_2D のすべての要素を、wave1_2D と wave2_2D の対応する要素の商に等しくします。

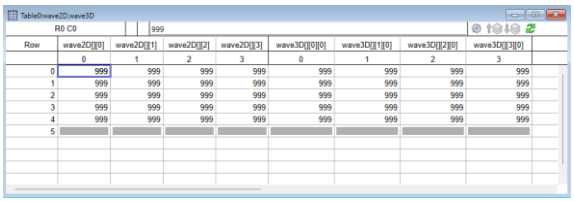
重要： 上記の例のように、右辺のウェーブに明示的なインデックスが含まれていないウェーブの代入は、関与するすべてのウェーブが同じ次元である場合にのみ定義されます。以下の代入の結果は未定義であり、予期しない結果をもたらす可能性があります。

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

次元が一致しないウェーブを使うは、以下に説明するように、明示的なインデックス指定を行う必要があります。

ウェーブ代入では、左辺で指定された各要素について、右辺を 1 回ずつ評価します。
この評価の時、記号 p、q、r、s は、それぞれ、値が計算されている宛先要素の行、列、レイヤー、チャンクの値をとります。
例えば：

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

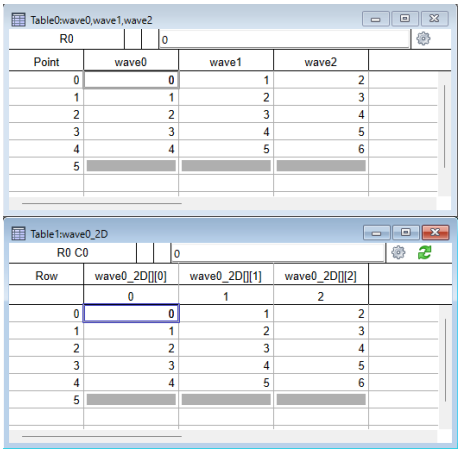


これにより、wave2D の内容が wave3D のレイヤー0 に格納されます。
この場合、格納先（wave3D）は 3 次元であるため、p、q、r は定義されますが、s は未定義となります。
以下の説明では、この代入について解説するとともに、ウェーブの代入全般について考えるためのアプローチを紹介します。

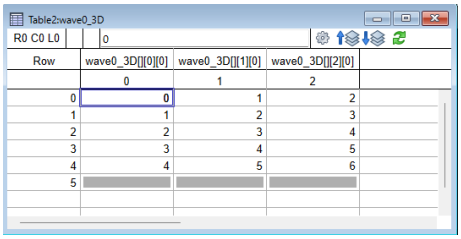
代入式の左辺は、wave3D のレイヤー0（[0]）のすべての行（最初の[]）とすべての列（2 番目の[]）に値を格納することを指定しています。
これらの要素それぞれについて、右辺を評価します。
評価中、シンボル p は設定しようとしている wave3D 内の要素の行番号を返し、シンボル q は列番号を返します。
シンボル r は、プロセス全体を通じて値 0 になります。
したがって、式 wave2D[p][q] は、wave3D 内の対応する行と列にある wave2D からの値を返します。

前述の例が示すように、ウェーブの代入は、ウェーブ間でデータを転送する方法を提供します。
適切なインデックス設定を行うことで、複数の 1D ウェーブから 2D ウェーブを、あるいは複数の 2D ウェーブから 3D ウェーブを構築することができます。
逆に、3D ウェーブのレイヤーを 2D ウェーブとして抽出したり、2D ウェーブの列を 1D ウェーブとして抽出したりすることも可能です。
以下に、これらの操作を説明する例をいくつか示します。

```
// 複数の 1D ウェーブから 2D ウェーブを構築
Make/O/N=5, wave0=p, wave1=p+1, wave2=p+2  // 1D ウェーブ
Make/O/N=(5,3) wave0_2D
wave0_2D[][0] = wave0[p]            // 列 0 のすべての行を代入
wave0_2D[][1] = wave1[p]            // 列 1 のすべての行を代入
wave0_2D[][2] = wave2[p]            // 列 2 のすべての行を代入
```



```
// 複数の 2D ウェーブから 3D ウェーブを構築
Duplicate/O wave0_2D, wave1_2D; wave1_2D *= -1
Make/O/N=(5,3,2) wave0_3D
// レイヤー0 のすべての行/列を代入
wave0_3D[][][0]= wave0_2D[p][q]
// レイヤー1 のすべての行/列を代入
wave0_3D[][][1]= wave1_2D[p][q]
```



```
// 3D ウェーブから 1 つのレイヤーを 2D ウェーブに抽出
wave0_2D = wave0_3D[p][q][0]       // レイヤー0 を 2D ウェーブに抽出
```

```
// 2D ウェーブから 1 つの列を 1D ウェーブに抽出
wave0 = wave0_2D[p][0]              // 列 0 を 1D ウェーブに抽出
```

このような代入式を理解するには、まず左辺のインデックスを見て、宛先のウェーブのうちどの要素が設定されるのかを確認します。
（左辺にインデックスがない場合、すべての要素が設定されます。）
次に、右辺を評価して各宛先要素の値を求める時に、p、q、r、s がどのような値の範囲をとるのかを考えます。
最後に、右辺のインデックスとして使われるこれらの値が、どのようにして宛先のソース要素を選択するのかを考えてみてください。

このような代入を作成するには、まず、設定したい宛先の要素を設定するために、左側で必要なインデックスを決定します。
次に、p、q、r、s がどのような値をとるかを考えます。
そして、p、q、r、s をインデックスとして使い、特定の宛先要素を計算する時に使うソース要素を選択します。

他にもいくつか例を挙げます。

```
// 2D ウェーブの 1 行を 1D ウェーブに抽出
Make/O/N=3 row1
row1 = wave0_2D[1][p]              // 2D ウェーブの行 1 を抽出
```

この例では、宛先の行インデックス（p）を使ってソース列を選択しますが、ソース行は常に 1 となります。

```
// 3D ウェーブの水平方向の断面を 2D ウェーブに抽出
Make/O/N=(2,3) slice_R2            // 行 2 の値からなるスライス
slice_R2 = wave0_3D[2][q][p]       // すべての列/レイヤーの行 2 を抽出
```

左辺に範囲インデックスを指定することで、特定の次元内の要素の範囲にデータを格納することができます。
例えば、wave0_3D の水平スライスをシフトさせるコマンドをいくつか紹介します。

```
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
```

最初の代入では、４行目のすべての要素からなるスライスが０行目に転送されます。
２番目の代入では、スライス n-1 がスライス n に転送されます。
これを理解するには、p が 1 から 4 へと進むにつれて、p-1 がソースの直前の行を参照することを認識してください。

ベクトル（ウェーブフォーム）から行列への変換

場合によっては、ベクトル形式のデータと、同じデータ値の行列形式との間で変換を行う必要があることがあります。
例えば、「sixteenVals」という名前のウェーブフォームに 16 個のデータ値が格納されたベクトルがあり、これを 8 行 2 列の行列として扱いたい場合などです。

通常、Redimension コマンドではデータが次元間で移動することはありませんが、１次元ウェーブへの変換または１次元ウェーブからの変換という特殊なケースでは、Redimension はウェーブの次元を変更する一方で、データは元の位置に残します。
変換を実現するために次のコマンドを使用できます。

```
Make/O/N=16 sixteenVals                // 1D
Redimension/N=(8,2) sixteenVals        // 2D となる。データ損失無し
```

1D ウェーブから再次元化を行う場合、まず列が埋められ、次にレイヤー、そしてチャンクが埋められます。
多次元ウェーブから 1D ウェーブへの再次元化でも、データが失われることはありません。

行列から行列への変換

行列をある形式から別の形式に変換する時は、直接、目的の形式に合わせて次元を変更しないでください。
たとえば、6×6 の行列「ウェーブ」があり、これを 3×12 にしたい場合、次のように試すかもしれません。

```
Make/O/N=(6,6) thirtySixVals            // 2D
Redimension/N=(3,12) thirtySixVals      // これは最後の３行を失う
```

しかし、Igor Pro はまず行数を３行に減らし、最後の３行のデータを削除してから、0 で埋めた６列を追加します。

この問題を回避する最も簡単な方法は、行列を 1D ベクトルに変換し、その後、新しい行列形式に変換することです。

```
Make/O/N=(6,6) thirtySixVals            // 2D
Redimension/N=36 thirtySixVals          // 1D ベクトルがデータを保持
Redimension/N=(3,12) thirtySixVals      // データは保持されている
```

多次元デシメーション

2D と 3D のウェーブを縮小することができます。

pixelate キーワードを指定した ImageInterpolate コマンドでは、行と列のブロックを平均化することで、2D ウェーブまたは 3D ウェーブの各レイヤーの縮小版を作成できます。
ImageInterpolate/PXSZ={*nx*,*ny*} pixelate を使います。
ここで、*nx* と *ny* はそれぞれ平均化する行数と列数を表します。

pixelate キーワードを指定した ImageInterpolate コマンドでは、3D のサブセクションを平均化することで、3D ウェーブの縮小版を作成することもできます。
ImageInterpolate/PXSZ={*nx*,*ny*,*nz*} pixelate を使います。
ここで、*nx*、*ny*、*nz* は、それぞれ平均化する行数、列数、レイヤー数です。

Igor Pro 9.0 で追加された WaveMetrics プロシージャファイル「Reduce Matrix Size.ipf」が提供する ReduceMatrixSize 関数は、行と列のサンプリング、あるいは行と列のブロックの平均化を行うことにより、2D ウェーブまたは 3D ウェーブの各レイヤーのサイズを縮小した表現を作成します。
この関数は、サンプリングにはウェーブ代入を、平均化には ImageInterpolate ピクセル化を使い、両方の手法に対して便利なラッパーを提供します。
ReduceMatrixSize を使うには、グローバルプロシージャファイルとして有効にするか、以下のコードを使って #include してください：

```
#include <Reduce Matrix Size>
```

1D ウェーブのデシメーションに関する解説については、ヘルプ Decimation を参照してください。

多次元フーリエ変換

Igor Pro の FFT と IFFT ルーチンは、混合基数かつ多次元対応です。
混合基数とは、データポイント数（または次元サイズ）が 2 の冪である必要がないことを意味します。

ウェーブの次元に関する制限が 1 つだけあります。
実数データに対して順方向 FFT を実行する場合、行数は偶数でなければなりません。
ただし、次元のサイズが素数であるか、その因数分解に大きな素数が含まれている場合、処理速度は通常の離散フーリエ変換（DFT）と同程度まで低下することに注意してください（つまり、演算回数は N*log(N) ではなく、N^2 のオーダーになります）。

FFT に関する詳細については、ヘルプ Fourier Transforms と FFT コマンドのヘルプを参照してください。

多次元ウェーブを 1D として扱う

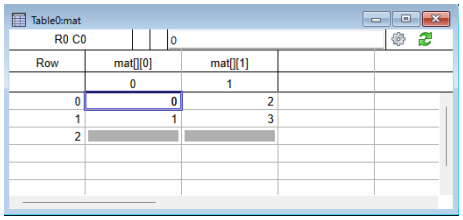
多次元ウェーブを 1D であるかのように扱うことが役立つ場合があります。
たとえば、2D ウェーブ内の NaN の数を調べたい場合、WaveStats は入力を 1D として扱う場合でも、そのウェーブを WaveStats に渡すことができます。

また、多次元ウェーブを 1D として扱うためには、メモリ内のデータの配置を理解する必要がある場合もあります。

2D ウェーブは、複数の列から構成されます。
メモリ上では、データは列ごとに配置されます。
これを「列優先順序（column-major order）」と呼びます。
列優先順序では、ある列の連続する要素はメモリ上で隣接して配置されます。

例えば、次を実行します。

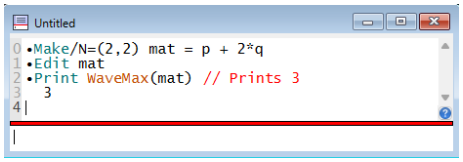
```
Make/N=(2,2) mat = p + 2*q
Edit mat
```



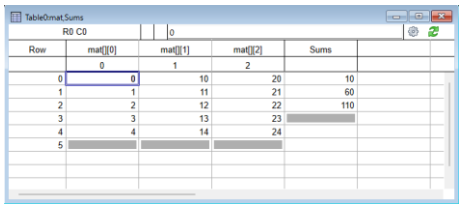
ウェーブ mat は 2 列で構成されています。
1 列目には 0 と 1 の値が含まれています。
2 列目には 2 と 3 の値が含まれています。

このウェーブを多次元対応していないコマンドや関数に渡すと、そのコマンドや関数は、このウェーブを「0、1、2、3」を含む単一の列として扱います。
例を挙げると：

```
Print WaveMax(mat)      // 3 を出力
```



多次元ウェーブがメモリ上にどのように配置されるかという知識を活用する例を次に示します。




```
Function DemoMDAs1D()
        // 2D ウェーブを作成
        Make/O/N=(5,3) mat = p + 10*q
        Variable numRows = DimSize(mat,0)

        // 各列の合計を計算
        Variable numColumns = DimSize(mat,1)
        Make/O/N=(numColumns) Sums
        Sums = sum(mat, p*numRows, (p+1)*numRows-1)
        Edit mat, Sums
End
```

次の記述：

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

は、2D ウェーブ mat を、多次元処理に対応していない sum 関数に渡します。
sum 関数は多次元処理に対応していないため、startX と endX パラメータを、mat を列優先順で配置された 1D ウェーブとして扱うように定義する必要があります。

3D ウェーブや 4D ウェーブも、1D ウェーブとして扱うことができます。
3D ウェーブでは、レイヤー n+1 のデータがレイヤー n のデータに続きます。
4D ウェーブでは、チャンク n+1 のデータがチャンク n のデータに続きます。