CONTENTS

Volume II User's Guide: II-5 Waves #2

ウェーブフォームの演算と代入

Igor Pro マニュアル：II-72 ページ以降をもとに編集

ウェーブフォームの演算は、Igor の解析機能の中でも非常に柔軟で強力な部分です。
標準的なプログラミング言語で 1 つの変数に代入するためのコードを書くのと同じように、ウェーブ全体またはウェーブの一部に対して代入文を書くことができます。

ウェーブの代入文では、左辺にウェーブが現れ、右辺に数式が現れます。
以下、いくつか例を挙げます：

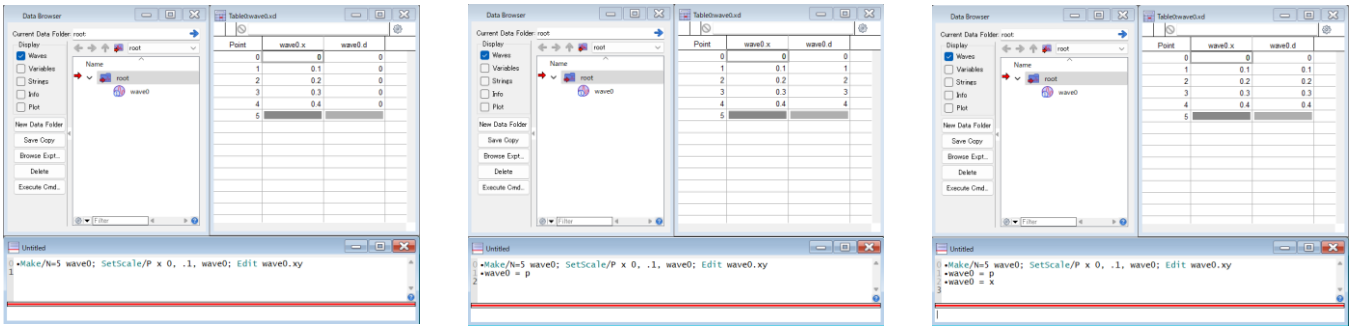
```
wave0 = sin(x)
wave0 = log(wave1/wave2)
wave0[0,99] = wave1[100 + p]
```

左側のウェーブは目標（Destination）ウェーブと呼ばれます。
右側のウェーブはソースウェーブと呼ばれます。

Igor がウェーブ代入文を実行すると、目標ウェーブの各ポイントについて、右辺の式が 1 回ずつ評価されます。
各評価の結果は、目標ウェーブの対応するポイントに格納されます。

実行中、シンボル p は設定をしようとしている目標ウェーブのポイントの番号に等しい値を持ち、シンボル x は、そのポイントの X 値に等しい値を持ちます。
指定したポイントの X 値は、そのポイントの番号とウェーブの X スケーリングによって決定されます。
これを確認するには、次のコマンドを 1 つずつ実行してください：

```
Make/N=5 wave0; SetScale/P x 0, .1, wave0; Edit wave0.xy
wave0 = p
wave0 = x
```



最初の代入文（2 行目）では、wave0 の各ポイントの値をポイント番号に設定します。
2 番目の代入文（3 行目）は、wave0 の各ポイントの値をそのポイントの X 値に設定します。

ソースウェーブは、評価対象のポイントでデータ値を返します。
例えば、

```
wave0 = log(wave1/wave2)
```

の時、Igor は wave0 の各ポイントについて、右辺式を一度評価します。
式の評価が行われるたびに、wave1 と wave2 は評価対象のポイントにおけるデータ値を返します。
したがって、以下の 2 つのコマンドは同等です：

```
wave0 = log(wave1/wave2)
wave0 = log(wave1[p]/wave2[p])
```

ウェーブの代入の理解

ウェーブの代入を理解するには、p と x の意味を理解することが重要です。
そのためには、Igor 内部で何が起こっているかを考えることが役立つかもしれません。
ウェーブの代入文は、Igor 内部でループを実行しています。

p は内部ループのループインデックスであり、目標ウェーブに設定されているポイントの範囲を順に実行します。
先の例では、p は 0 から始まり、内部ループを回るたびにインクリメントされ、最大値である N-1（ここでは N は wave0 の点の数）まで増加します。
p の各値について、右辺の式が評価され、その結果が wave0 の対応するポイントに格納されます。

ウェーブフォームの代入文の右辺に wave1 または wave1[p] がある場合、内部ループの反復中に目標ウェーブ（wave0）のポイント p における wave1 の値を返します。

x は p と似ていますが、内部ループのループインデックスではなく、ループインデックスから目標ウェーブの X スケーリングを使って計算されます：

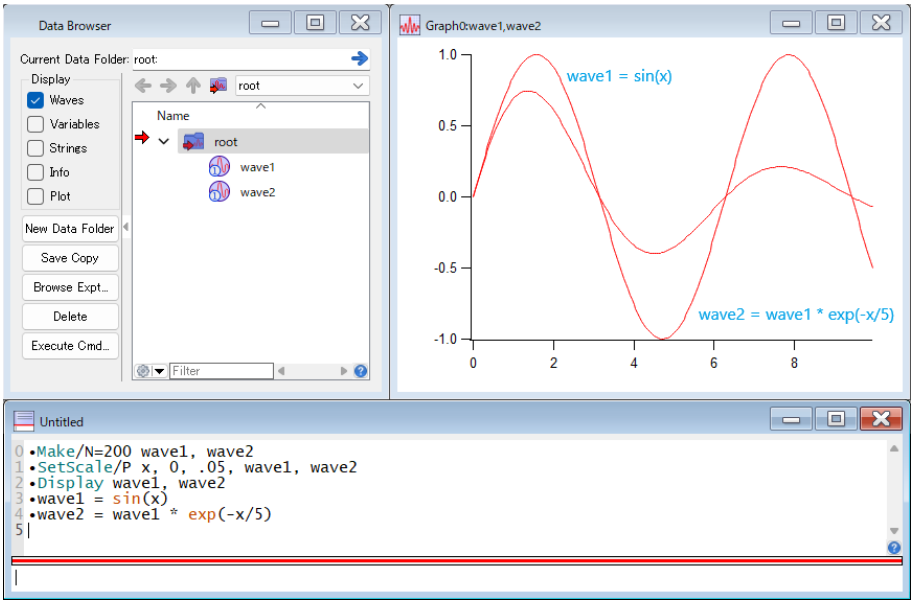
```
x = x0 + p*dx
```

ウェーブの代入は、Igor の重要な機能であり、理解する価値があります。

ウェーブ代入の例

次の一連のコマンドは、上記で説明した考え方の一部を示しています（1 行ずつ実行するとわかりやすいです）。

```
Make/N=200 wave1, wave2              // 各 200 ポイントの 2 つのウェーブを作成
SetScale/P x, 0, .05, wave1, wave2 // X 値を 0～10 に設定（0.05x200）
Display wave1, wave2                 // ウェーブのグラフを作成
wave1 = sin(x)                       // wave1 に値を代入
wave2 = wave1 * exp(-x/5)            // wave2 に値を代入
```



wave1 には 200 個のポイントがあるため、ウェーブの代入文 wave1=sin(x) は、wave1 の各ポイントに 1 回ずつ、合計 200 回 sin(x) を評価します。

wave1 の最初のポイントはポイント番号 0、wave1 の最後のポイント番号は 199 です。
この例では示されていないシンボル p は、0 から 199 までの値をとります。
シンボル x は、SetScale コマンドで指定されているように、0 から始まり、0.05 刻みで変化する wave1 の 200 個の X 値を順にたどります。
各評価の結果は、wave1 の対応するポイントに格納され、wave1 は正弦波の約 1.5 サイクル分となります。

wave2 も 200 ポイントあるため、ウェーブの代入文 wave2=wave1*exp(-x/5) は wave1*exp(-x/5) を wave2 のポイントごとに 200 回評価します。
この Experiment では右辺の式に wave1 というウェーブが含まれています。
Igor が代入を実行すると、p は 0 から 199 まで変化します。
右辺が評価される 200 回のそれぞれにおいて、wave1 は対応するポイントの値を返します。
各評価の結果は、wave2 の対応するポイントに格納され、wave2 は減衰正弦波の約 1.5 サイクル分となります。

ウェーブの代入文の機能は、目標のウェーブのデータ値を設定することです。
Igor は、代入から暗示される機能的な関係は記憶しません。
今回の例では、代入後に wave1 を変更しても wave2 は自動的に変更されません。
wave2 が wave1 を変更する前と同じ機能関係を持つように変更する場合は、wave2=wave1*exp(-x/5) の代入を再度実行する必要があります。

関数的な関係を確立する特別な種類のウェーブ代入文があります。
ただ、これは控えめに使うべきです。
詳細はマニュアル II-84 Wave Dependency Formulas を参照してください。

その他のウェーブ代入機能

シンボル p が現在の行の番号を返すのと同様に、シンボル q、r、s は多次元ウェーブの列、レイヤー、チャンクの現在の番号を返します。
行の次元におけるシンボル x は、列、レイヤー、チャンクの次元における類似のシンボル y、z、t に対応しています。
詳細はマニュアル II-6 Multidimensional Waves を参照してください。

複数のプロセッサを使って、実行に時間がかかるウェーブフォーム代入文を実行することができます。
詳細はマニュアル IV-323 Automatic Parallel Processing with MultiThread を参照してください。

右辺は、目標ウェーブを含むデータフォルダーのコンテキストで評価されます。
詳細はマニュアル II-111 Data Folders and Assignment Statements を参照してください。

通常、ソースウェーブは、目標ウェーブと同じポイント数と X スケーリングを持っています。
これが当てはまらない場合は、ウェーブの代入文を別の方法で記述することが有効です。
詳細は本ファイルの後半「長さが一致しないウェーブ」かマニュアル II-83 Mismatched Waves を参照してください。

インデックスとサブレンジ

Igor では、1D ウェーブの特定のポイントまたはポイントの範囲を参照する 2 つの方法が用意されています。
X 値インデックスとポイント番号インデックスです。
次の例を考えてみてください。[] と () で差があります：

```
wave0[54] = 92              // wave0 のポイント 54 を 92 に設定
wave0(54) = 92              // wave0 の X=54 を 92 に設定
wave0[1,10] = 92            // wave0 のポイント 1 から 10 までを 92 に設定
wave0(1,10) = 92            // wave0 の X=1 から X=10 までを 92 に設定
```

ブラケット [] は、ポイント番号を使ってウェーブをインデックス付けしていることを示しています。
ブラケット内の数字は、インデックス付きウェーブのポイント番号として解釈されます。

括弧 () は、X 値を使ってウェーブをインデックス付けしていることを示しています。
括弧内の数字は、インデックス付きウェーブの X 値として解釈されます。
X 値をインデックスとして使う場合、まず、インデックスが付けられたウェーブの X スケーリングに基づいて、その X 値に対応するポイント番号を見つけ、そのポイント番号をポイントインデックスとして使用します。

ウェーブにポイントスケーリングが適用されている場合、この 2 つの方法は同じ結果となります。
ただし、ウェーブの X スケーリングをポイントスケーリング以外に設定すると、これらのコマンドの動作は違うものになります。
どちらの場合も、範囲は両端を含みます。

範囲だけでなく、ポイント数のインクリメントも指定できます。
例えば、

```
wave0[0,98;2] = 1           // wave0 内の偶数のポイントを 1 に設定（0 から 2 ずつ足す）
wave0[1,99;2] = -1          // wave0 内の奇数のポイントを -1 に設定（1 から 2 ずつ足す）
```

セミコロンの後の数字はインクリメントを表します。
Igor は開始ポイントの番号から始まり、インクリメントごとに飛ばしながら終了ポイントの番号まで進みます。
各結果のポイント番号で、ウェーブ代入文の右辺を評価し、その結果を目標ポイントに設定します。

X 値で範囲を指定する場合にもインクリメントを使うことができますが、インクリメントは常にポイント数で指定されます。
例えば、

```
wave0(0,100;5) = PI         // 指定された X 値の wave0 のポイントを PI に設定
```

Igor は x=0 に対応するポイント番号から開始し、x=100 に対応するポイント番号までを処理します。
反復処理のたびに、ポイント番号は 5 ずつ増加します。

目標ウェーブの範囲を指定するときに、いくつかのショートカットを使うことができます。
サブレンジの開始値と終了値は、どちらも省略できます。
開始が省略された場合は、ポイント番号 0 が使われます。
終了が省略された場合は、ウェーブの最後のポイントが使われます。
最後のポイントを指定するときに、* 文字または INF を使うこともできます。
インクリメント値が省略された場合は、1 ポイントにデフォルト設定されます。

これらのショートカットを説明する例をいくつか示します：

```
wave0[ ,50] = 13                    // wave0 のポイント 0～50 に 13 を設定
wave0[51,] = 27                     // wave0 のポイント 51～最終ポイント に 27 を設定
wave0[51,*] = 27                    // wave0 のポイント 51～最終ポイント に 27 を設定
wave0[51,INF] = 27                  // wave0 のポイント 51～最終ポイント に 27 を設定
wave0[ , ;2] = 18.7                 // wave0 のすべての偶数ポイントに 18.7 を設定
wave0[1,*;2] = 100                  // wave0 のすべての奇数ポイントに 18.7 を設定
```

目標ウェーブのサブレンジは、１つのポイントまたはポイントの範囲で構成される場合がありますが、ソースウェーブのサブレンジは、１つのポイントで構成されなければなりません。
つまり、ウェーブの代入文は、

```
wave1(4,5) = wave2(5,6)             // 正しくない！
```

は正しくありません。
この代入では、x は 4～5 の範囲です。

次のようにすると、望み通りの結果を得ることができます：

```
wave1(4,5) = wave2(x+1)             // OK!
```

左辺で指定された範囲により、x は 4～5 になります。
したがって、x+1 は 5～6 になり、右辺の式は 5～6 の wave2 の値を返します。

インデックスウェーブでのインデックス化

次の構文でインデックス値を提供する別のウェーブを使って、目標ウェーブの特定の要素を設定することができます：

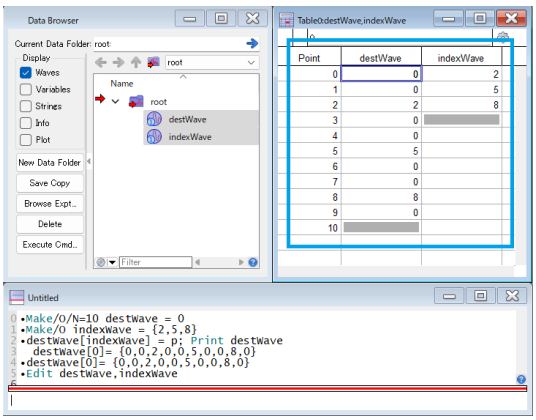
```
destWave[indexWave] = <expression>
```

この機能は Igor Pro 8.0 で追加されました。

目標ウェーブが 1D の場合、インデックスウェーブには有効なポイント番号のリストを含める必要があります。

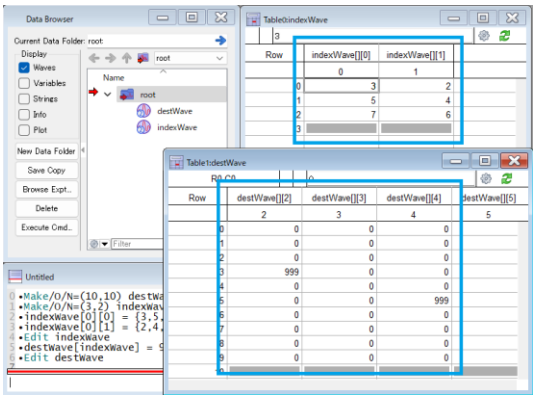
例えば、

```
Make/O/N=10 destWave = 0
Make/O indexWave = {2,5,8}
destWave[indexWave] = p; Print destWave
destWave[0]= {0,0,2,0,0,5,0,0,8,0}
```



目標ウェーブが多次元の場合、インデックスウェーブは、最初の列に有効な行インデックス、２番目の列に有効な列インデックスを持つ 2D とすることができます。
次の例では、999 に設定される (3,2)、(5,4)、(7.6) の要素を除いて、目標ウェーブのすべての要素をゼロに設定します。

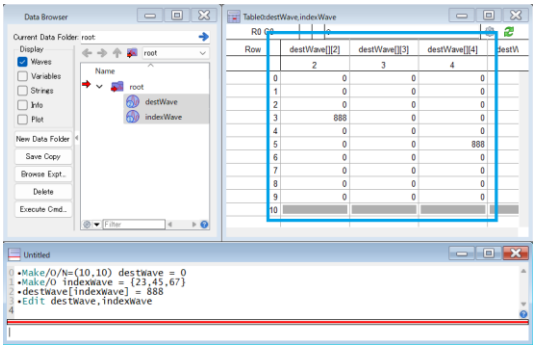
```
Make/O/N=(10,10) destWave = 0
Make/O/N=(3,2) indexWave
// 列 0 に行インデックスを保存
indexWave[0][0] = {3,5,7}
// 列 1 に列インデックスを保存
indexWave[0][1] = {2,4,6}
Edit indexWave
destWave[indexWave] = 999
Edit destWave
```



目標ウェーブが多次元である場合、インデックスウェーブは、目標ウェーブ自体が 1D であるかのように、線形ポイント番号を含む 1D にすることが認められます。
代入文は、目標ウェーブが 1D であるかのように評価されるため、q、r、s、y、z、t は右辺でゼロを返します。

次の例では、1D インデックスウェーブを使って前の例と同じ要素を設定します。

```
Make/O/N=(10,10) destWave = 0
Make/O indexWave = {23,45,67}
destWave[indexWave] = 888
// 10x10 を 1D の 100 ポイントにし、23, 45, 67 番目
```



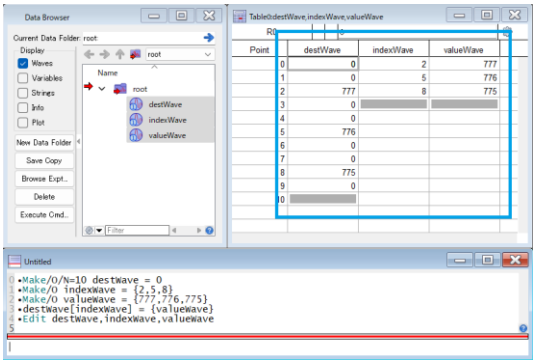
左辺のインデックスウェーブを使うだけでなく、右辺の値ウェーブも次の構文で使うことができます：

```
destWave[indexWave] = {valueWave}
```

値ウェーブは、値の 1D ベクトルです。

インデックス値と同じ数の値を含み、目標ウェーブと同じ型（数値、文字列など）である必要があります。
例を示します。

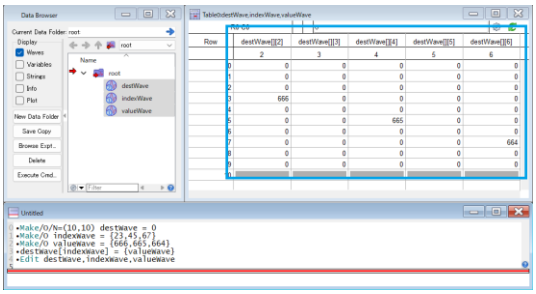
```
Make/O/N=10 destWave = 0
Make/O indexWave = {2,5,8}
Make/O valueWave = {777,776,775}
destWave[indexWave] = {valueWave}
```



インデックスウェーブを使う場合、個々の値をリストで指定する機能はサポートされていません。

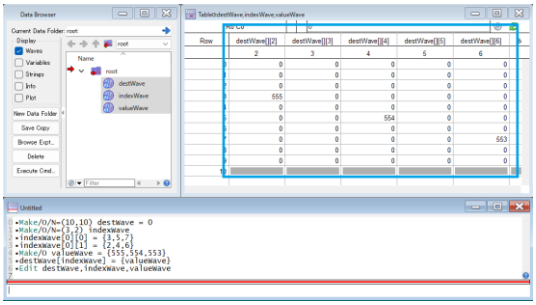
```
destWave[indexWave] = {777,776,775}        // エラー – 括弧内は値ウェーブを指定する必要がある
```

次の例では、1D インデックスウェーブを提供することで、2D の目標ウェーブを 1D として扱っています。



```
Make/O/N=(10,10) destWave = 0
Make/O indexWave = {23,45,67}
Make/O valueWave = {666,665,664}
destWave[indexWave] = {valueWave}
```

次の例では、2D インデックスウェーブを提供することで、2D の目標ウェーブを 2D として扱っています。



```
Make/O/N=(10,10) destWave = 0
Make/O/N=(3,2) indexWave
// 列 0 に行インデックスを保存
indexWave[0][0] = {3,5,7}
// 列 1 に列インデックスを保存
indexWave[0][1] = {2,4,6}
Make/O valueWave = {555,554,553}
destWave[indexWave] = {valueWave}
```

ウェーブ代入の補間

例えば、wave1[1.75] は、ポイント 1 のデータ値からポイント 2 のデータ値までの 3/4 の地点における wave1 の値を返します。
この補間は 1D ウェーブに対してのみ行われます。多次元データを使った代入については、マニュアル II-96 Multidimensional Wave Assignment を参照してください。

これは強力な機能です。
キャリブレーションと呼ばれる等間隔のキャリブレーション曲線があり、xData と呼ばれるウェーブに保存されている特定の X 座標のセットにおけるキャリブレーション値を見つけたいとします。
キャリブレーションウェーブの X スケーリングを設定している場合は、次の操作が可能です。

```
Duplicate xData, yData
yData = calibration(xData)
```

これは、xData データウェーブの各 X 座標に対して、キャリブレーションウェーブの線形補間値を求めるために、Igor のウェーブ代入文の補間機能を使います。

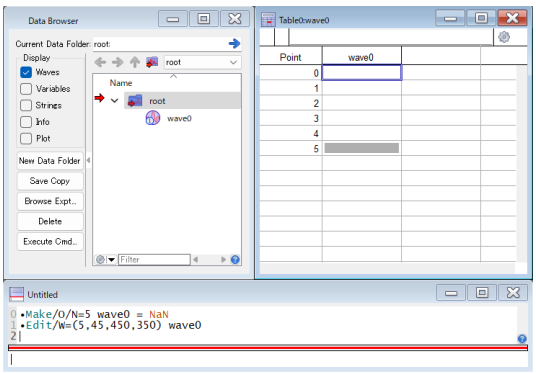
値のリスト

ウェーブまたはウェーブのサブレンジに値を代入するには、{} 内のリストを使います。
また、行や列を追加することもできます。

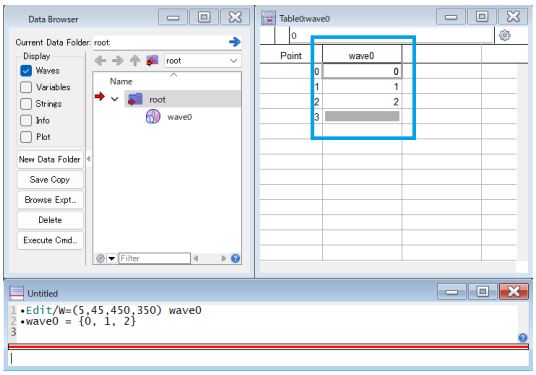
1D の値のリスト

1D ウェーブの要素を設定し、値のリストを使って行を追加する方法を説明します。
次に示すように、1D の値のリストは、{} 内の行の値のリストで構成されます。

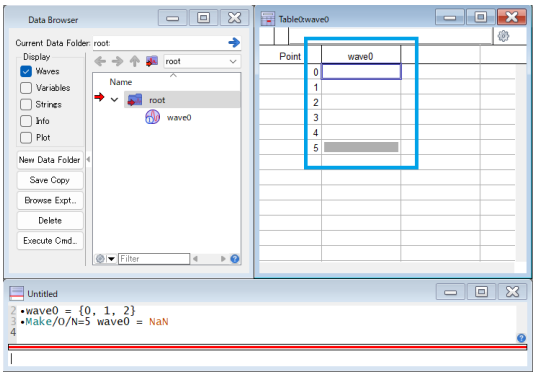
```
// 5 ポイントのウェーブを作成し、テーブルに表示
Make/O/N=5 wave0 = NaN
Edit/W=(5,45,450,350) wave0
// 右図では大きさを調整してキャプチャ
```



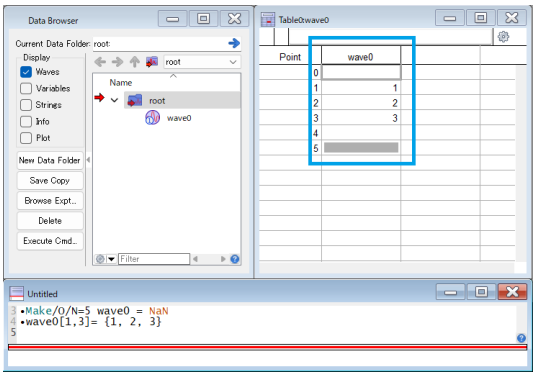

```
// wave0 を 3 行に次元を変更し、Y 値を設定
wave0 = {0, 1, 2}
```



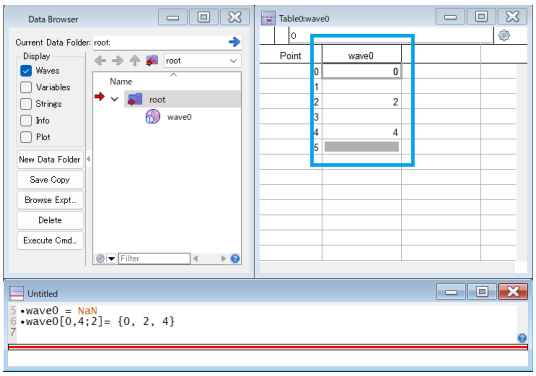

```
// 5 ポイントに戻し、NaN を設定
Make/O/N=5 wave0 = NaN
```




```
// ポイント 1 ～ 3 に値を設定
wave0[1,3]= {1, 2, 3}
```

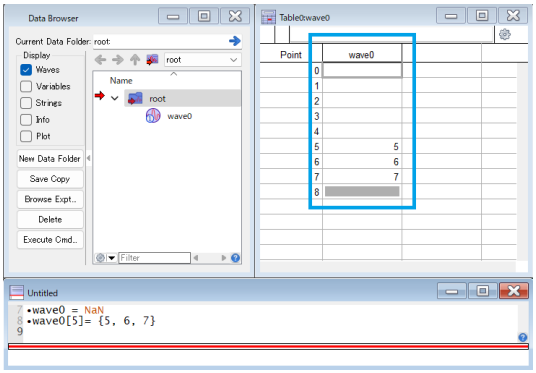


```
// 空白に戻す
wave0 = NaN
// ポイント１～３に値を設定（前の手順の別表現）
wave0[1]= {1, 2, 3}
```

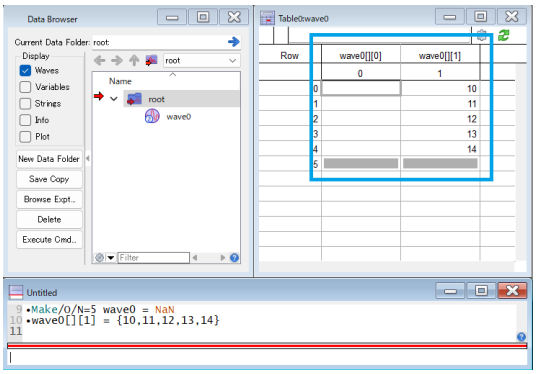


```
// 空白に戻す
wave0 = NaN
// 間隔を指定してポイント０，２，４を設定
wave0[0,4;2]= {0, 2, 4}
```

```
// 空白に戻す
wave0 = NaN

// wave0 を５行から８行に拡張し、新しい Y 値を設定
// 行を追加。インデックス 5 はウェーブの行番号と等しい
wave0[5]= {5, 6, 7}
```



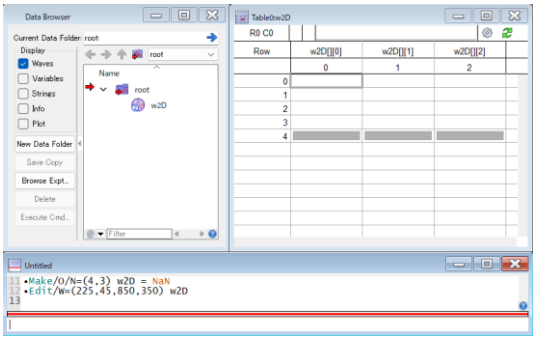
```
// ５ポイントに戻す
Make/O/N=5 wave0 = NaN

// wave0 を 1D から 2D に変えるために列を追加
// 列インデックス 1 はウェーブの列番号と等しい
wave0[][1] = {10,11,12,13,14}
```



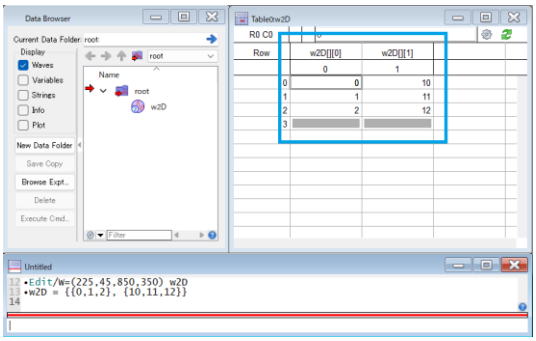
2D の値のリスト

2D ウェーブの要素を設定し、値のリストを使って行と列を追加する方法を説明します。
次に示すように、2D の値のリストは、{} で囲まれたリストの入れ子リストで構成されています。
各内側のリストは、１つの列の行の値を定義します。

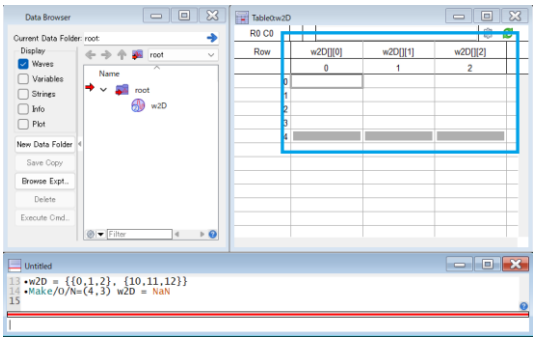
```
// ４行３列のウェーブを作成
Make/O/N=(4,3) w2D = NaN
Edit/W=(225,45,850,350) w2D
```



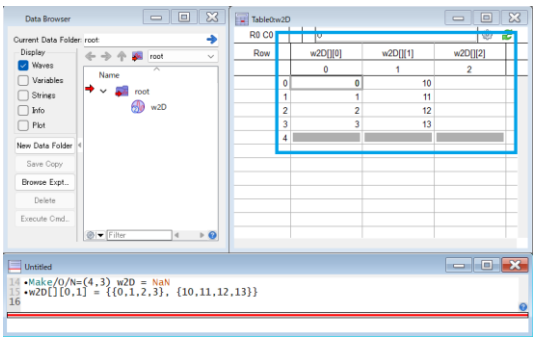
```
// w2D の次元を変え、要素を設定
// 列 0 を {0,1,2} に、列 1 を {10,11,12} に設定
w2D = {{0,1,2}, {10,11,12}}
```



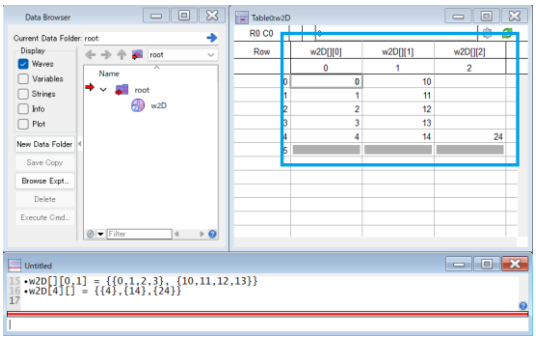
```
// 4行3列に戻す
Make/O/N=(4,3) w2D = NaN
```



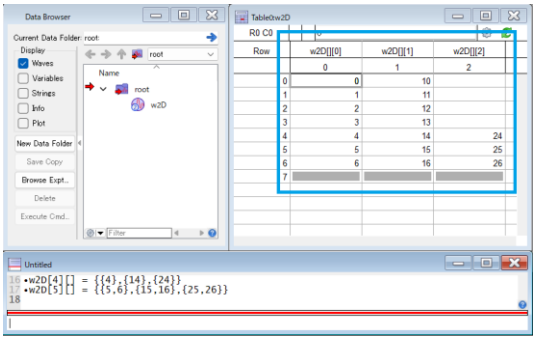
```
// 列 0 を {0,1,2,3} に、列 1 を {10,11,12,13} に設定
// [] は「すべての行」を示し、列 0 から列 1 の
// すべての行に設定することを意味する
w2D[][0,1] = {{0,1,2,3}, {10,11,12,13}}
```



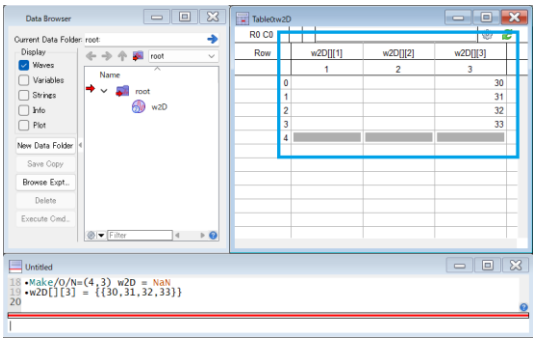
```
// w2D を4行から5行に拡張し、新しい Y 値を設定
// 行を追加。インデックス 4 はウェーブの行番号と等しい
// [] は「すべての列」を示し、すべての列の4行に設定
// することを意味する
w2D[4][] = {{4},{14},{24}}
```



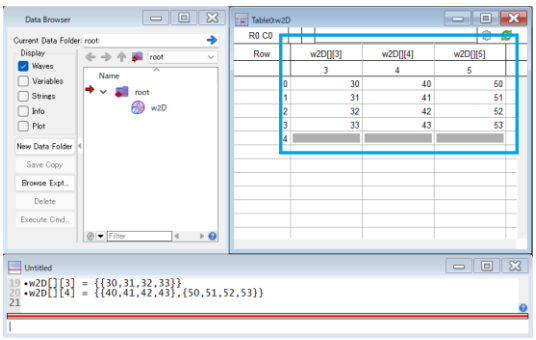
```
// w2D を5行から7行に拡張し、新しい Y 値を設定
w2D[5][] = {{5,6},{15,16},{25,26}}
```



```
// ４行３列に戻す
Make/O/N=(4,3) w2D = NaN

// w2D を３列から４列に拡張し、新しい Y 値を設定
// インデックス 3 はウェーブの列番号と等しい
w2D[][3] = {{30,31,32,33}}
```



```
// w2D を４列から６列に拡張し、新しい Y 値を設定
w2D[][4] = {{40,41,42,43},{50,51,52,53}}
```



ウェーブの初期化

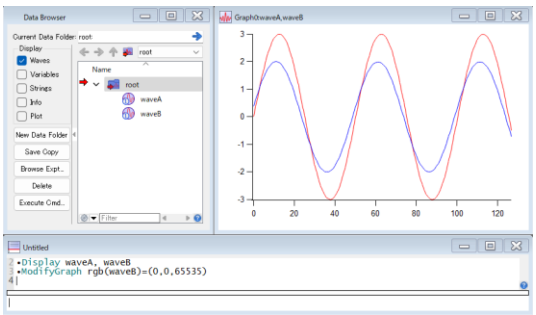
Igor のコマンドラインまたはプロシージャから、次の例のように、１つのコマンドでウェーブを作成し、初期化することができます。

```
Make wave0=sin(p/8)          // wave0 はデフォルトのポイント数を持つ
Make coeffs={1,2,3}          // coeffs は３つのポイントだけ持つ
```

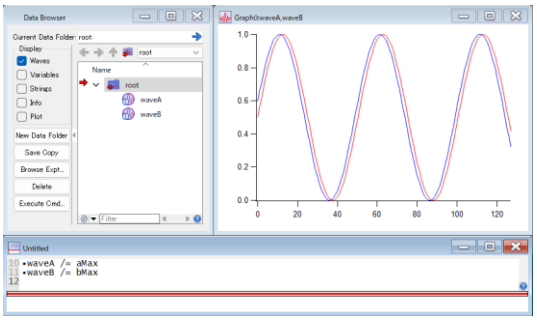
例：ウェーブの正規化

複数のウェーブの形状を比較するときには、それらを正規化して共通の範囲を共有できるようにすることが望ましい場合があります。
例えば、

```
// サンプルデータの作成
Make waveA = 3*sin(x/8)
Make waveB = 2*sin(pi/16 + x/8)

// ウェーブの表示
Display waveA, waveB
ModifyGraph rgb(waveB)=(0,0,65535)
```



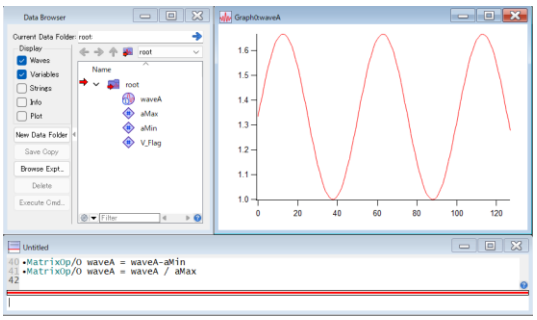
```
// ウェーブを正規化
Variable aMin = WaveMin(waveA)
Variable bMin = WaveMin(waveB)
waveA -= aMin
waveB -= bMin
Variable aMax = WaveMax(waveA)
Variable bMax = WaveMax(waveB)
waveA /= aMax
waveB /= bMax
```



一時変数 aMin と bMin を使い方に注目してください。
これらは２つの理由で必要です。
まず、もし `waveA -= WaveMin(waveA)` と記述した場合、WaveMin は waveA のポイントごとに１回ずつ呼び出されることになり、処理時間が長くなります。
さらに悪いことに、ウェーブフォームの代入文の実行中に、waveA の最小値が変化し、不正確な結果が返されます。

ウェーブフォーム演算には、より高速な方法が存在する場合があります。
大きなウェーブの場合、FastOp と MatrixOp コマンドにより処理速度が向上します。

```
Make waveA = 3*sin(x/8)

Display waveA

Variable aMin = WaveMin(waveA)
Variable aMax = WaveMax(waveA)
// FastOp はウェーブ変数をサポートしていない
waveA -= aMin
// ウェーブの演算
FastOp waveA = (1/aMax) * waveA
MatrixOp/O waveA = waveA - aMin
MatrixOp/O waveA = waveA / aMax
```



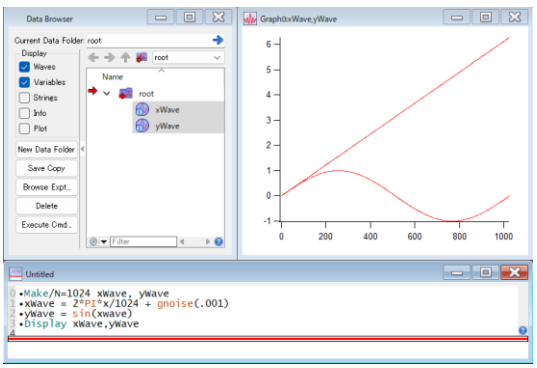
例：XY データをウェーブフォームデータに変換する

XY データを等間隔のウェーブフォームデータに変換することが望ましいことがあります。
例えば、高速フーリエ変換（FFT）には等間隔のデータが必要です。
時間ドメインで XY データを測定した場合は、FFT を実行する前にこの変換を行う必要があります。

次のようなサンプル XY データを作成することができます。

```
Make/N=1024 xWave, yWave
xWave = 2*PI*x/1024 + gnoise(.001)
yWave = sin(xwave)
```

（右図は xWave と yWave をプロットしたもの）

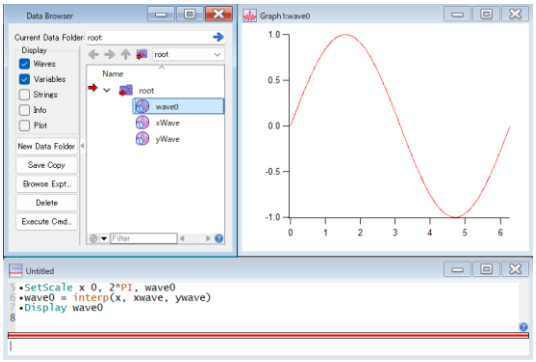


xWave は 0 から 2π までの値を持ち、若干のノイズが含まれています。
このデータは x 軸方向に等間隔で配置されているわけではありませんが、単調増加、つまり常に増加しています。
もし、これが単調でなければ、XY ペアをソートできます。

XY データを表すウェーブフォームは次のようにして作成できます。

```
Duplicate ywave, wave0
SetScale x 0, 2*PI, wave0
wave0 = interp(x, xwave, ywave)
```

（右図は wave0 をプロットしたもの）



SetScale コマンドは、wave0 のスケーリングを設定し、その X 値が 0 から 2 の範囲で動くようにします。
そのデータ値は、yWave vs xWave で表される曲線から、これらの X 値のそれぞれで直線補間を使って値を取り出すことで生成されます。

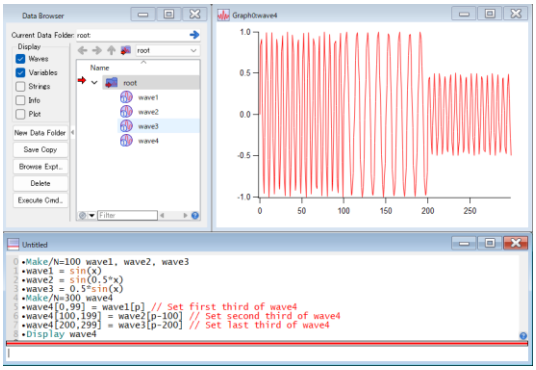
他の補間方法については、マニュアル III-109 Converting XY Data to a Waveform を参照してください。

例：ウェーブの結合

ウェーブの結合は、Concatenate コマンド（本ファイルの末尾かマニュアル V-82 を参照）を使うことでより簡単に実行することができます。
次の例は、主にウェーブ代入文の使い方を示すことを目的としています。

仮に 100 ポイントのウェーブが 3 つ（wave1, wave2, wave3）あるとします。
ここで、3 つの元のウェーブを結合した第 4 のウェーブ wave4 を作りたいと考えています。
これを実行するためのコマンドは以下の通りです。

```
Make/N=100 wave1, wave2, wave3
wave1 = sin(x)
wave2 = sin(0.5*x)
wave3 = 0.5*sin(x)
Make/N=300 wave4
wave4[0,99] = wave1[p]          // wave4 の最初の 1/3
wave4[100,199] = wave2[p-100]   // 次の 1/3
wave4[200,299] = wave3[p-200]   // 最後の 1/3
```

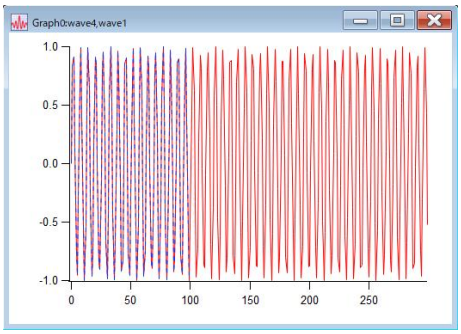


この例では、wave4 のサブレンジを、ウェーブ代入文の代入先として使っています。
右辺は wave1、wave2、wave3 の値を示しています。
p は、評価対象のポイントの範囲を指していることを思い出してください。
つまり、最初の代入では p は 0～99、2 番目の代入では 100～199、3 番目の代入では 200～299 の範囲となります。
各代入において、右辺のウェーブは 0 から 99 までの 100 ポイントのみです。
したがって、ソースウェーブの 100 の値を抽出するには、右辺の p をオフセットする必要があります。

例：ウェーブの分解

例えば、300 ポイントの wave4 を、それぞれ 100 ポイントの 3 つのウェーブ、wave1、wave2、wave3 に分解するとします。
これを実行するためのコマンドは以下の通りです。

```
Make/N=300 wave4
wave4 = sin(x)
Make/N=100 wave1, wave2, wave3
wave1 = wave4[p]          // wave4 の最初の 1/3
wave2 = wave4[p+100]      // 次の 1/3
wave3 = wave4[p+200]  // 最後の 1/3
```



（右図は wave4 と wave1 をプロットしたもの）

この例では、wave4 のサブレンジをデータのソースとして使います。
ポイント番号インデックスを使って、wave4 の必要なセグメントをインデックスします。
wave1、wave2、wave3 はそれぞれ 100 ポイントあるため、p は 0～99 の範囲となります。
最初の代入では、wave4 のポイント 0～99 にアクセスします。
2 番目の代入では、wave4 のポイント 100～199 にアクセスします。
3 番目の代入では、wave4 のポイント 200～299 にアクセスします。

別のウェーブのサブレンジからウェーブを作成するには、Duplicate コマンド（マニュアル V-185 を参照）を使うこともできます。

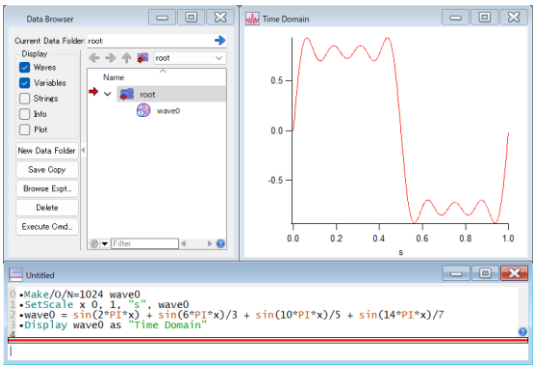
例：複素数ウェーブの演算

Igor には、複素数や複素数ウェーブを操作するための多くのビルトイン関数が用意されています。
これらの例を以下に示します。

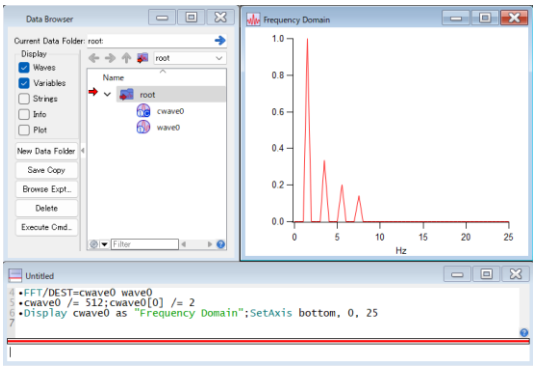
時間ドメインのウェーブフォームを作成し、それを FFT することで複素数ウェーブを生成します。
下記の関数は、複素数ウェーブの実数部と虚数部の抽出方法、平方和の求め方、直交表現から極座標表現への変換方法を示しています。
周波数ドメイン処理の詳細については、マニュアル III-270 Fourier Transform を参照してください。

（以下、画面は Function の内部を個別にコマンドラインで実行したもので示します）

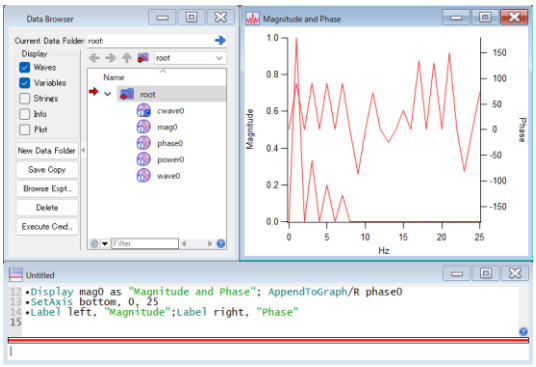
```
Function ComplexWaveCalculations()
    // 時間ドメインウェーブフォームを作成
    Make/O/N=1024 wave0
    // 0 から 1 秒の範囲
    SetScale x 0, 1, "s", wave0
    wave0 = sin(2*PI*x) + sin(6*PI*x)/3 +
            sin(10*PI*x)/5 + sin(14*PI*x)/7
    Display wave0 as "Time Domain"
```

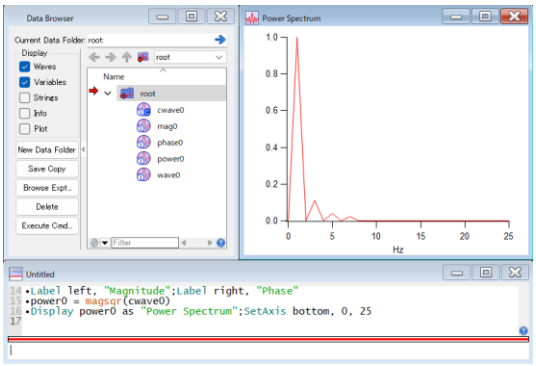


```
    // FFT の実行
    // cwave0 は複素数、513 ポイント
    FFT/DEST=cwave0 wave0
    cwave0 /= 512;cwave0[0] /= 2      // 振幅を正規化
    Display cwave0 as "Frequency Domain";SetAxis
    bottom, 0, 25
```



```
    // マグニチュードと位相を計算
    // 実数ウェーブが存在する
    Make/O/N=513 mag0, phase0, power0
    CopyScales cwave0, mag0, phase0, power0
    mag0 = real(r2polar(cwave0))
    phase0 = imag(r2polar(cwave0))
    phase0 *= 180/PI                  // 角度に変換
    Display mag0 as "Magnitude and Phase";
        AppendToGraph/R phase0
    SetAxis bottom, 0, 25
    Label left, "Magnitude";Label right, "Phase"
    // パワースペクトルを計算
    power0 = magsqr(cwave0)
    Display power0 as "Power Spectrum";SetAxis
        bottom, 0, 25
End
```





例：比較演算子とウェーブの合成

比較演算子 ==、>=、>、<=、< は、ウェーブの合成に役立ちます。
x<0 の場合は -π、x>=0 の場合は +π となるようにウェーブを設定したいと仮定します。
次のウェーブの代入文がこれを実現します。

```
wave1 = -pi*(x<0) + pi*(x>=0)
```

これは、条件文が真であれば 1 を、偽であれば 0 を返し、その後、乗算が実行されるため、正しく動作します。

条件演算子を使って、次のような代入を行うこともできます。

```
wave0 = (x>0) ? pi : -pi
```

mod 関数と == を使って、一連のインパルスを作成することができます。
このウェーブ方程式は、0 から始まる 10 進数の各ポイントに 5 を代入し、その他すべてのポイントに 0 を代入します。

```
wave1 = (mod(p,10)==0) * 5
```

例：ウェーブの代入とラベルを使ったインデックス化

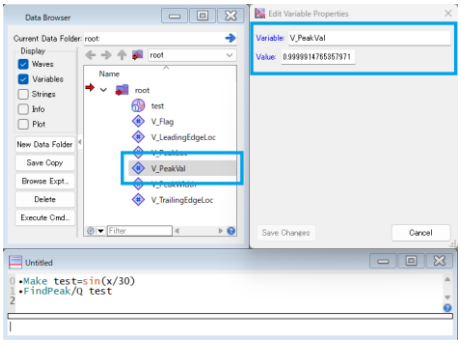
次元ラベルはウェーブの値を意味のある名前で参照するために使うことができます。
したがって、例えば、係数値を保持するウェーブを作成し、混乱を招きやすく意味の乏しい数値インデックス（例：coef[1]）ではなく、係数の名前（例：coef[%Friction]）でこれらの名前を直接参照することができます。
ウェーブの値とラベルをテーブルで表示することもできます。

ウェーブの次元ラベルは SetDimLabel コマンド（本ファイルの末尾かマニュアル V-838 を参照）を使って作成します。
詳細はマニュアル II-93 Dimension Labels を参照してください。
次元ラベルの長さは、最大 255 バイトです。
スペースを含むような自由な名前を使う場合は、必ずシングルクオートで囲んでください。

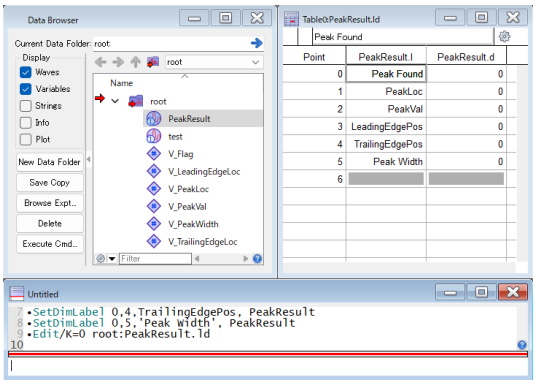
Igor Pro 8.0 より前のバージョンでは、次元ラベルは 31 バイトに制限されていました。
長い次元ラベルを使う場合、ウェーブファイルと Experiment には Igor Pro 8.0 以降が必要です。

次の例では、ウェーブを作成し、FindPeak コマンド（本ファイルの末尾かマニュアル V-247 を参照）を使って、ウェーブのピークパラメーターを取得します。
次に、適切なラベルを付けた出力パラメーターウェーブを作成し、そのラベルを使って FindPeak の結果を出力ウェーブに代入します。

```
// ウェーブを作成し、ピークパラメーターを取得
Make test=sin(x/30)
FindPeak/Q test
```



```
// 適切な行ラベルを持つウェーブを作成
Make/N=6 PeakResult
SetDimLabel 0,0,'Peak Found', PeakResult
SetDimLabel 0,1,PeakLoc, PeakResult
SetDimLabel 0,2,PeakVal, PeakResult
SetDimLabel 0,3,LeadingEdgePos, PeakResult
SetDimLabel 0,4,TrailingEdgePos, PeakResult
SetDimLabel 0,5,'Peak Width', PeakResult

// テーブル内に PeakResult 値とラベルを表示
Edit PeakResult.ld
```



上記で説明した方法に加えて、テーブルにウェーブを表示し、データと一緒に次元ラベルを表示することで、次元ラベルを作成、編集することもできます。
ラベルを持つテーブルに関する詳細は、マニュアル II-235 Showing Dimension Labels を参照してください。

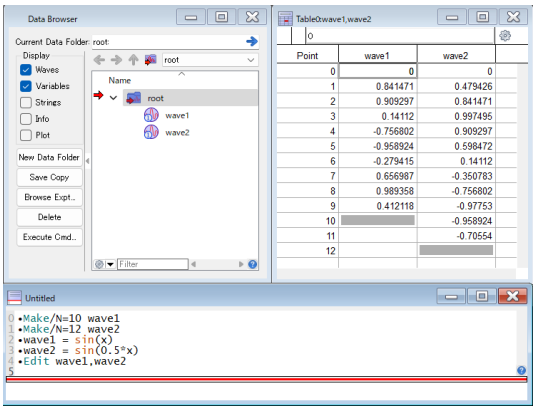
長さが一致しないウェーブ

ほとんどの作業においては、異なる長さのウェーブを混ぜる必要はありません。
実際、このようなことをするのは、意図的な場合よりも、ミスによる場合のほうが多いです。
しかし、作業において混ぜることが必要な場合は、Igor がどのように処理するのかを知っておく必要があります。

別のウェーブの値を代入する場合について考えてみます。

wave1 のポイント数を 10、wave2 を 12 とする

```
Make/N=10 wave1
Make/N=12 wave2
wave1 = sin(x)
wave2 = sin(0.5*x)
Edit wave1,wave2
```



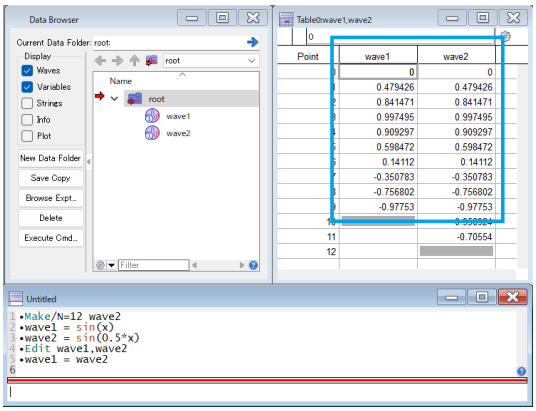
次の代入を実行します。

```
wave1 = wave2
```

この代入では、明示的なインデックス付けは行われていないため、Igor は次のように式を記述したかのように処理します。

```
wave1 = wave2[p]
```

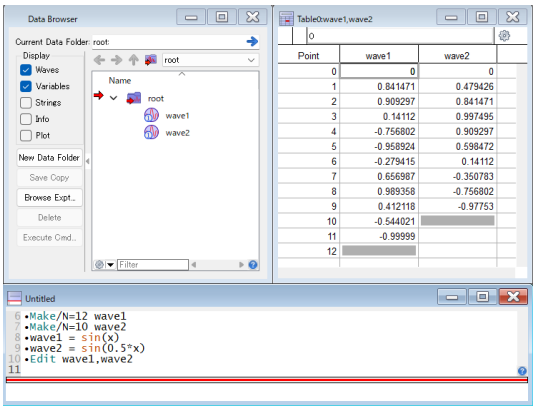
wave2 のポイント数が wave1 のポイント数よりも多い場合、p の範囲は 0 から n-1（n は wave1 のポイント数）であるため、余分なポイントは代入に影響を与えません。



wave2 のポイント数が wave1 のポイント数よりも少ない場合を考えます。

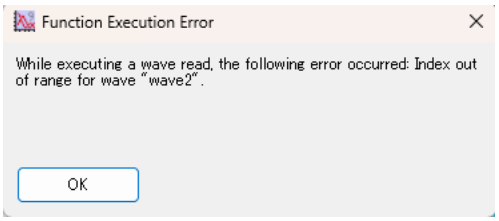
wave1 のポイント数を 12、wave2 を 10 とする

```
Make/N=12 wave1
Make/N=10 wave2
wave1 = sin(x)
wave2 = sin(0.5*x)
Edit wave1,wave2
```



次の代入を実行します。

```
wave1 = wave2
```



wave2 のポイント数が wave1 のポイント数よりも少ない場合、Igor は wave2 の長さよりも大きい値の p について wave2[p] を評価しようとします。

この場合、ウェーブの代入文の実行中に、wave2 の最後のポイントの値が返され、「index out of range」エラーが発生します。

実際には、wave2 の値で補間して、wave1 の値を wave2 の値まで広げたいのかもしれません。
これを実行するには、右辺の適切な X 値に明示的にインデックスを付ける必要があります。

例えば、異なる長さのウェーブを２つ用意すると、次のようなことができます：

```
big = small[p*(numpnts(small)-1)/(numpnts(big)-1)]
```

NaN・INF・欠損値

浮動小数点ウェーブ内のポイントのデータ値は、通常、有限の数値ですが、NaN（非数値）や INF（無限大）の場合もあります。
NaN は「Not a number（数値ではない）」という意味です。
数式として意味をなさない場合は、NaN という値が返されます。
例えば、log(-1) は NaN という値を返します。
また、テーブルまたはウェーブ代入文を使って NaN を指定し、欠損値を表すこともできます。
数学的に意味を成すものの、有限の値を持たない場合、式は値 INF を返します。
log(0) は値 -INF を返します。

IEEE の浮動小数点の標準規格は、NaN 値の表現と動作を定義しています。
数値ウェーブで NaN を表す方法はありません。
整数ウェーブに NaN を格納しようとすると、無意味な値が格納されます。

比較演算子は NaN パラメーターでは動作しません。
定義上、NaN は他の NaN と比較しても、あらゆるものと比較しても常に「偽」となるためです。
値が NaN であるかどうかを確認するには numtype を使います。

Igor は、カーブフィッティングやウェーブの統計操作において、NaN や INF を無視します。
NaN と INF はグラフのスケーリングに影響を与えません。
プロットの時、NaN と　INF をそれぞれ欠損値と無限大として適切に処理します。

Igor は、FFT などの DSP 関連の演算をはじめ、多くの演算で NaN や INF を無視しません。
一般的に、ウェーブのデータポイントのすべて、またはほとんどを数値的に組み合わせる操作は、１つ以上のポイントが NaN または INF である場合、無意味な結果をもたらします。
注目すべき例としては、領域および平均関数、Integrate、FFT 演算が挙げられます。
Smooth や Differenciate など、小数のポイントのみを混ぜるいくつかの演算では、NaN または INF の近傍にあるポイントのみ「汚染」されます。
Interpolate 操作（Analysis メニュー）を使って、NaN を含まないウェーブを作成することができます。
area や mean などの関数、Convolve などの演算、ウェーブのポイントを合計するその他の関数や演算から NaN が返ってきた場合、ウェーブのポイントの一部が NaN であることを示しています。

カーブフィッティングの結果が NaN となった場合、カーブフィッティングが失敗したことを示しています。
トラブルシューティングのヒントは、マニュアル III-266 Curve Fitting Troubleshooting を参照してください。

NaN の処理方法については、マニュアル III-112 Dealing with Missing Values を参照してください。

目標ウェーブをソースウェーブとして使わないでください

ウェーブ代入文の目標が右辺にも表れている場合、予期しない結果となる可能性があります。
次の例を考えてみてください：

```
wave1 -= wave1(5)
wave1 -= vcsr(A)              // カーソル A は wave1 上にある
```

これらの例は、いずれも wave1 の特定のポイントの値を wave1 のすべてのポイントから差し引こうとするものです。
これは、代入の時に、特定のポイントにおける wave1 の値が変更されることになり、期待通りに動作しません。
代入のどこかのポイントで、wave1(5) または vcsr(A) は 0 を返します。
というのも、wave1 はそのポイントの値は、それ自身から差し引かれているからです。

変数を使って、特定のポイントにおける wave1 の値を保存することで、望む結果を得ることができます。

```
Variable tmp
tmp = wave1(5); wave1 -= tmp
tmp = vcsr(A); wave1 -= tmp
```

本セクションで使用したコマンドの説明（Volume V Reference）

Concatenate

Igor Pro マニュアル：V-82 をもとに編集

文法

```
Concatenate [type flags][flags] waveListStr, destWave
Concatenate [type flags][flags] {wave1, wave2, wave3,…}, destWave
Concatenate [type flags][flags] {waveWave}, destWave
```

Concatenate コマンドは、ソースウェーブのデータを destWave に結合します。
destWave が存在しない場合は、新たに作成されます。
destWave が存在し、上書き（/O）が指定されていない場合、ソースウェーブのデータは目標ウェーブの既存のデータに結合されます。

デフォルトでは、可能であれば、結合は目標ウェーブの次元を増やします。
例えば、同じ長さの２つの 1D ウェーブを結合すると、２列の 2D ウェーブが得られます。
目標のウェーブは、より高次元に「昇格」したといえます。

/NP（No promotion）フラグを使うと、目標ウェーブの次元は変更されません。
例えば、同じ長さの２つの 1D ウェーブを /NP で結合すると、結合された 1D ウェーブの長さが２倍になります。

異なる長さのソースウェーブが存在する場合、/NP が指定されているかに関わらず、次元の昇格は行われません。

パラメーター

waveListStr は、セミコロンで区切られた入力ウェーブ名のリストを含む文字列記述です。
末尾にもセミコロンが付きます。
waveListStr 内のウェーブ名の数に制限はありません。

{wave1, wave2, …} 構文を使う場合は、100 ウェーブに制限されています。
{waveWave} 構文では、waveWave は入力ウェーブへの参照を含む１つのウェーブ参照ウェーブです。
この構文は Igor Pro 8.0 で追加されました。

フラグ

/DL	次元ラベルを設定します。 昇格の時には、ソースウェーブ名を新しい次元ラベルとして使い、それ以外では既存のラベルを使います。
/FREE	destWave をフリーウェーブとして作成します（マニュアル IV-91 Free Waves を参照）。 /FREE フラグは Igor Pro 8.0 で追加されました。
/KILL	ソースウェーブを Kill します。
/NP	高次元への昇格を妨げます。
/NP=*dim*	昇格を妨げ、指定された次元（0=行、1=列、2=レイヤー、3=チャンク）に沿ってデータを追加します。

dim で指定された次元以外の次元は、すべてのウェーブで同じである必要があります。

/O　　destWave を上書きします。

形式フラグ（関数でのみ使用される）

Concatenate は、ユーザー関数でさまざまな形式フラグを使って、出力ウェーブの参照変数の形式を指定することができます。
これらの形式フラグは、同じ名前の別のウェーブ参照変数と一致させる必要がある場合、または、ウェーブ代入用にコンパイルする式の種類を識別する必要がある場合を除いて、使用する必要はありません。
形式フラグの完全なリストと詳細については、マニュアル IV-73 WAVE Reference Types と IV-74 WAVE Reference Type Flags を参照してください。

詳細

destWave がまだ存在しない場合、または /O フラグが使われている場合、最初のソースウェーブが複製されて destWave が作成されます。
ソースウェーブのリスト順にウェーブが結合されます。
destWave が存在し、/O フラグが指定されていない場合、結合は destWave から開始されます。

destWave は、ソースウェーブのリストで使うことはできません。

ソースウェーブは、すべて数値、またはすべてテキストでなければなりません。

昇格が許可された場合、すべてのウェーブが共通して持つ下位次元の数が、destWave の次元を決定し、destWave の次元は 1 つ大きくなります。
デフォルトの動作は、ソースウェーブの大きさによって異なります。
同じ長さの 1D ウェーブを結合すると 2D ウェーブが生成されますが、長さが異なる 1D ウェーブを結合すると　1D ウェーブが生成されます。
同様に、同じ大きさの 2D ウェーブを結合すると 3D ウェーブが生成されますが、2D ソースウェーブの列数が異なる場合は、destWave は 2D ウェーブになります。
また、2D ウェーブの行数が異なる場合は、destWave は 1D ウェーブになります。
行数が同じ 1D ウェーブと 2D ウェーブを結合すると 2D ウェーブが生成されますが、行数が異なる場合は、destWave は 1D ウェーブになります。
後半の例を参照してください。

/NP フラグを使うと、次元の昇格が妨げられ、destWave の次元は入力ウェーブと同じになります。

警告

ユーザー定義関数のループなど、特定の状況下では、結合関数が予期しない動作を示すことがあります。

ユーザー定義関数内に次のような記述がある場合：

```
Concatenate/O ..., DestWaveName
```

Igor はコンパイル時に自動でローカルウェーブ参照変数 DestWaveName を作成します。
実行時に、ウェーブ参照変数が NULL の場合、名前はリテラル名とみなされ、その名前のウェーブが現在のデータフォルダーに作成されます。

ループ内で Concatenate を最初に呼び出した後に発生する可能性があるように、ウェーブ参照変数が NULL でない場合、参照されたウェーブはそれがどこにあろうとも上書きされます。

現在のデータフォルダーにウェーブを作成、または上書きする場合は、次の 2 つの方法のうちのどちらかを使う必要があります。

```
Concatenate/O ..., $"DestWaveName"
WAVE DestWaveName                    // 目標ウェーブを後で参照する場合にのみ必要
```

または

```
Concatenate/O ....., DestWaveName
// DestWaveName の使用が完了したら
WAVE DestWaveName=$""
```

例

```
// 次のウェーブが与えられた場合：
Make/N=10 w1,w2,w3
Make/N=11 w4
Make/N=(10,7) m1,m2,m3
Make/N=(10,8) m4
Make/N=(9,8) m5

// 1D ウェーブを結合
Concatenate/O {w1,w2,w3},wdest           // wdest は 10x3 のマトリックスになる
Concatenate {w1,w2,w3},wdest             // wdest は 10x6 のマトリックスになる
Concatenate/NP/O {w1,w2,w3},wdest        // wdest は 30 ポイントの 1D ウェーブになる
Concatenate/O {w1,w2,w3,w4},wdest        // wdest は 41 ポイントの 1D ウェーブになる

// 2D ウェーブを結合
Concatenate/O {m1,m2,m3},wdest           // wdest は 10x7x3 のボリュームになる
Concatenate/NP/O {m1,m2,m3},wdest        // wdest は 10x21 のマトリックスになる
Concatenate/O {m1,m2,m3,m4},wdest        // wdest は 10x29 のマトリックスになる
Concatenate/O {m4,m5},wdest              // wdest は 152 ポイントの 1D ウェーブになる
Concatenate/O/NP=0 {m4,m5},wdest         // wdest は 19x8 のマトリックスになる

// 1D ウェーブと 2D ウェーブを結合
Concatenate/O {w1,m1},wdest              // wdest は 10x8 のマトリックスになる
Concatenate/O {w4,m1},wdest              // wdest は 81 ポイントの 1D ウェーブになる

// 2D ウェーブに行を追加
Make/O/N=(3,2) m6, m7
Concatenate/NP=0 {m6}, m7                // m7 は 6x2 のマトリックスになる

// 2D ウェーブに列を追加
Make/O/N=(3,2) m6, m7
Concatenate/NP=1 {m6}, m7                // m7 は 3x4 のマトリックスになる

// 2D ウェーブにレイヤーを追加
Make/O/N=(3,2) m6, m7
Concatenate/NP=2 {m6}, m7                // m7 は 3x2x2 のボリュームになる
```

```
// 最後のコマンドは次と同様の結果となります：
// Concatenate {m6}, m7
// どちらの形式も m7 に３番目の次元を追加して拡張します
```

SetDimLabel

Igor Pro マニュアル：V-838 をもとに編集

文法

SetDimLabel *dimNumber*, *dimIndex*, *label*, *wavelist*

SetDimLabel コマンドは、次元ラベルまたは次元要素ラベルを指定のラベルに設定します。

パラメーター

行には dimNumber=0、列には 1、レイヤーには 2、チャンクには 3 を使います。

dimIndex が -1 の場合、その次元全体のラベルを設定します。
dimIndex≧0 の場合、その次元の要素のラベルを設定します。

label は、名前（例：time）であり、文字列（例：”time”）ではありません。

label は、255 バイトに制限されています。
31 バイト以上を使う場合には、Igor Pro 8.0 以降が必要です。

詳細

次元ラベルは最大 255 バイトまでで、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、リベラル名で許可されているスペースや、その他の通常は使うことができない文字を含めることができます。

次元ラベルは、オブジェクト名と同じ特性を持ちます。
オブジェクト名の全体の説明はマニュアル III-501 Object Names を参照してください。

Igor Pro 9.0 より前のバージョンでは、不正な文字（ダブルクオート、シングルクオート、コロン、セミコロン、制御文字）を含む作成することができました。
これは、現在、エラーとして扱われます。
不正な名前の次元ラベルを持つ既存の Experiment は、エラーなしで読み込むことができます。

FindPeak

Igor Pro マニュアル：V-247 をもとに編集

文法

FindPeak [*flags*] *waveName*

FindPeak コマンドは、指定されたウェーブの平滑化された１次および２次導関数を分析して、最小値または最大値を検索します。
ピーク位置、振幅、幅に関する情報は、出力変数に返されます。

フラグ

一部のフラグは FindLevel コマンドと同じ意味を持ちます。

/B=*box*	スライディング平均のボックスサイズを設定します。
/I	インパルス（１つのサンプルのピーク）に対応するために、検索条件を変更し、minLevel を超える値を１つだけ要求するようにします。 デフォルトの基準では、ピークを検出するには２つの連続した値が最小レベルを超える必要があります。 負のピークを検出する場合は２つの連続した値が /M レベル未満である必要があります。 minLevel を省略してもインパルスは検出できるため、その場合、/I は不要です。
/M=*minLevel*	ピークの最小レベルを定義します。 /N を指定すると最大レベルに変更されます（下記、「詳細」を参照）。
/N	正のピーク（最大値）ではなく、負のピーク（最小値）を検索します。
/P	場所の出力変数（下記、「詳細」を参照）は、浮動小数点で表示されます。 /P が省略された場合、X 値として表示されます。
/Q	ピークが見つからなかった場合、履歴に表示されず、処理も中断されません。
/R=*(startX,endX)*	検索の X 軸の範囲と方向を指定します。
/R=*[startP,endP]*	検索のポイントの範囲と方向を指定します。

詳細

FindPeak は以下の変数を設定します。

`V_flag`	/Q フラグを使うときのみ設定されます。 0：ピークが検出されたとき ゼロ以外の任意の値はピークが検出できなかったことを意味します。
`V_LeadingEdgeLoc`	startX または startP に最も近いピークエッジの補間位置です。 /P フラグを使う場合、`V_LeadingEdgeLoc` は X 値ではなく、ポイント番号です。 エッジが検出されなかった場合は、NaN（数値ではない）になります。
`V_PeakLoc`	ピークが検出された場所の補間された X 値です。 /P フラグを使う場合、`V_PeakLoc` は X 値ではなく、ポイント番号です。
`V_PeakVal`	検出されたピークのおおよその値です。 ピークが検出されなかった場合は、NaN になります。
`V_PeakWidth`	補間されたピーク幅です。 /P フラグを使う場合、`V_PeakWidth` は X 値ではなく、ポイント数です。

V_PeakWidth は負にはなりません。
ピークエッジがいずれも検出されなかった場合は、NaN になります。

V_TrailingEdgeLoc　endX または endP に最も近いピークエッジの補間位置です。
/P フラグを使う場合、V_TrailingEdgeLoc は X 値ではなく、ポイント番号です。
エッジが検出されなかった場合は、NaN になります。

FindPeak は、BoxSmooth アルゴリズムとボックスパラメーターを使って、入力ウェーブのスライディング平均を計算します。
この平滑化の結果の微分がゼロを横切る場所にピークの中心が存在します。
平滑化された結果の 2 階微分がゼロを横切る場所で、ピークエッジが検出されます。
微分値の線形補間により、中心とエッジをより正確に特定します。
ピーク値は、ピークの中心を囲む 2 つの平滑化されていない値のうち大きいほうの値です（/N の場合は小さいほうの値）。

FindPeak 高精度の測定ルーチンではなく、シンプルなピーク検出器として設計されています。
より正確な統計情報を得るには、PulseStats コマンドを使ってください。

/M がない場合、微分がピークを横切る場所にピークが検出されますが、ピークの高さは関係ありません。

/M=minLevel フラグを使うと、FindPeak はボックススムージングされた入力ウェーブにおいて、minLevel より低いピーク（つまり、検出されたピークの Y 値が minLevel を超える）を無視します。
/N も指定されている場合（最小値を検索）、FindPeak は振幅が minLevel より大きなピーク（つまり、検出されたピークの Y 値が minLevel より小さい）を無視します。

/I を指定しない場合、ピークには minLevel を超える 2 つの連続した値が必要です。
minLevel を超える値が 1 つだけあるピークを検索する場合は /I を使います。

ピークの検索は、startX（または /R が指定されていない場合はウェーブの最初のポイント）から開始され、endX（または /R が指定されていない場合はウェーブの最後のポイント）で終了します。
逆方向の検索も許可されており、V_LoadingEdgeLoc と V_TrailingEdgeLoc の値が交換されます。

Multipeak Fitting パッケージで使われるプロシージャファイル「Peak Autofind.ipf」には、シンプルな自動ピーク検出機能が実装されています。
詳細は、コマンドラインで DisplayHelpTopic "Multipeak Fitting" を実行してください。

FindPeak コマンドは多次元を認識しません。
詳細はマニュアル II-95 Analysis on Multidimensional Waves を参照してください。