CONTENTS

ビジュアルヘルプ – Signal Processing

このヘルプファイルでは、信号変換に重点を置いた基本的な分析操作について説明しています。

フーリエ変換

Igor は、高速フーリエ変換（FFT）アルゴリズムを使って、離散フーリエ変換（DFT）を計算します。
FFT は通常、信号の振幅や位相の検出など、より大規模なプロセスの１ステップとして、プロシージャから呼び出されます。
Igor の FFT は、素因数分解多次元アルゴリズムを使っています。
素因数分解により、このアルゴリズムはほぼすべてのデータポイントに対して機能します。
古いバージョンの Igor では、データポイントは２のべき乗の数に制限されていました。

このセクションでは、1 次元 FFT に焦点を当てています。
FFT の多次元的な側面に関する情報は、多次元フーリエ変換を参照してください。

Analysis メニューから Fourier Transforms を選択すると、ウェーブに対してフーリエ変換を実行できます。
これにより、Fourier Transforms ダイアログが表示されます。

Forward または Reverse ラジオボタンをクリックして、変換のタイプを選択します。
変換するウェーブを Wave リストから選択します。
Wave リストの下にある From Target ボックスを有効にすると、ターゲットウィンドウ内の適切なウェーブのみがリストに表示されます。

一部のウェーブがリストにない理由

「適切な」ウェーブとはどういう意味でしょうか？

データは実数または複素数です。
データが実数の場合、データポイントの数は偶数でなければなりません。
これは、順変換されたウェーブの逆変換が元のウェーブと等しくなることを保証するために導入された人工的な制限です。
多次元データの場合、行の数のみが偶数でなければなりません。
逆 FFT の制限の一部は、コマンドラインを使って回避できます。

逆 FFT には複素数データが必要です。
データポイントの数に制限はありません。
ただし、歴史的理由と互換性のため、ポイントの数に関する特定の値は、次のセクションで説明するように、異なる扱いを受けます。

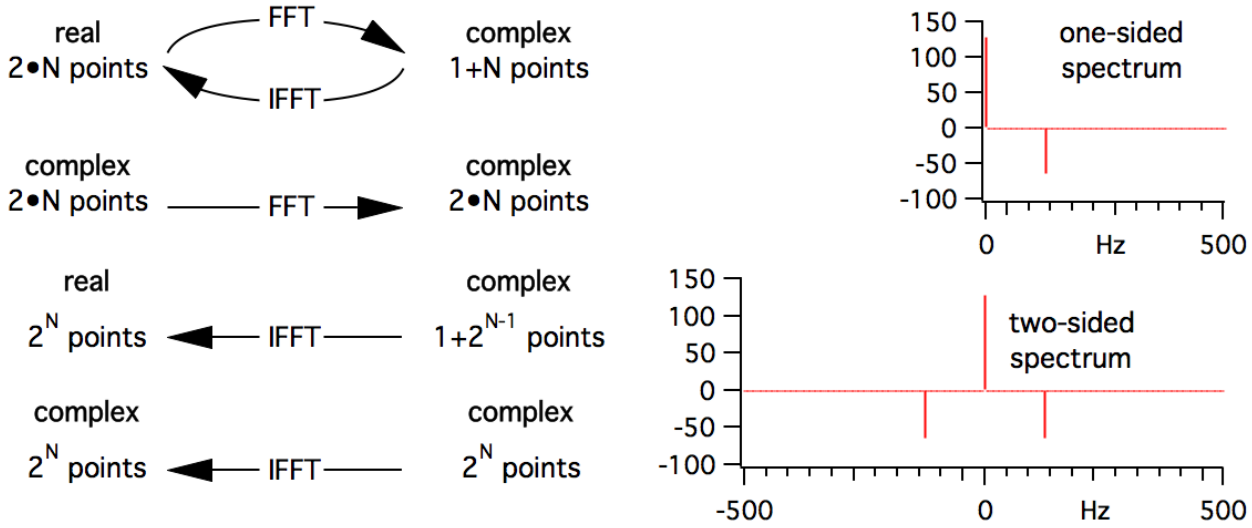
ウェーブタイプとポイント数の変化

ウェーブが N ポイントの１次元実数ウェーブ（N は偶数でなければならない）の場合、FFT 演算の結果は、「片側スペクトル」を含む N/2+1 ポイントからなる複素数ウェーブになります。
負のスペクトルは、実数入力ウェーブでは同一であるため、計算されません。

ウェーブが複素数（虚数が 0 であっても）の場合、そのデータ型とポイント数は、順 FFT によって変更されません。

FFT の結果は「両側スペクトル」となり、正と負の周波数スペクトルが両方含まれます。
複素数の入力データの虚数部が 0 以外の場合、これらのスペクトルは異なります。

下の図は、虚数成分がゼロの 128 ポイントのデータの両側スペクトルを示しています。



ポイントのマジックナンバーと IFFT

逆 FFT を実行する時、入力は常に複素数ですが、結果は実数または複素数のいずれかになります。
（3.0 以前のバージョンの Igor では、2 の整数乗（2n）しか順変換できなかったため、Igor は順変換されたウェーブのポイント数から、逆変換でどのような結果を作成すべきかを判断していました。
互換性を確保するため、Igor バージョン 3.0 以降では、特定の数のポイントは引き続きマジックナンバーとして扱われます。）

ウェーブのポイント数が 2 の整数乗（2^n）の場合、IFFT によるウェーブは複素数になります。
ウェーブ内のポイント数が 2 の整数乗（$1+2^n$）より 1 つ多い場合、IFFT によって得られるウェーブは、長さ（2^{n+1}）の実数になります。

ポイントの数が 2 つのマジックナンバーのいずれでもない場合、フーリエ変換の逆変換の結果は、フーリエ変換ダイアログで複素数結果が選択されていない限り、実数になります。

X スケーリングと単位の変更

FFT コマンドは、変換されたウェーブの X スケーリングを変更します。
変換されたウェーブの X 単位が時間（s）、周波数（Hz）、長さ（m）、または逆数（m^{-1}）の場合、結果のウェーブの単位は、それぞれの共役単位に設定されます。
その他の単位は現在無視されます。
X スケーリングの X0 値は、ウェーブが実数か複素数かによって変化しますが、Δx は常に同じに設定されます：

$$\Delta x_{FFT} = \frac{1}{N \cdot \Delta x_{original}}$$

N はウェーブの元の長さです。

元のウェーブが実数である場合、FFT 処理後の最小 X 値（X0）は 0 となり、最大 X 値は次のようになります：

$$\Delta x_{N/2} = \frac{N}{2}\Delta x_{FFT} = \frac{N}{2}\frac{1}{N\Delta x_{original}} = \frac{1}{2\Delta x_{original}}$$

$$= Nyquist\ Frequency$$

元のウェーブが複素数の場合、FFT 処理後の最大 X 値は XN/2 - ΔXFFT、最小 X 値は -XN/2 となり、ポイント N/2 の X 値は 0 になります。

IFFT コマンドは、FFT コマンドが行った X 軸のスケール変更を逆転させますが、ポイント 0 の X 値は常に 0 になります。

FFT 振幅スケーリング

さまざまなプログラムでは、FFT ウェーブフォームの振幅のスケーリングにさまざまなアプローチを採用しています。
分析によって適切なスケーリング方法は異なり、その方法については一般的な合意はありません。
Igor は、この点において他の多くの参考文献とは異なる、「Numerical Recipes in C」で説明されている方法を採用しています。

FFT によって計算される N ポイントの複素数ウェーブ $wave_{org}$ の DFT 方程式は、次のとおりです：

$$wave_{FFT}[n] = \sum_{k=0}^{N-1} wave_{orig}[k] \cdot \exp\left[2\pi ikn / N\right]$$

ここで、

$$i = \sqrt{-1}$$

であり、$wave_{orig}$ および $wave_{FFT}$ は、FFT 操作の前後の同じ波形を指します。

N ポイントの複素数ウェーブの FFT に対して IFFT によって計算される IDFT 方程式は、次のとおりです。

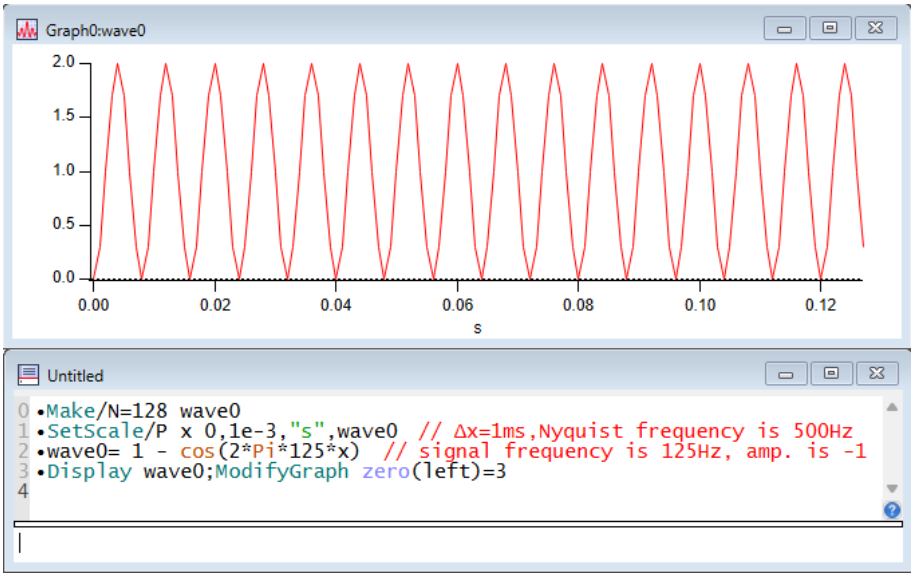
$$wave_{IFT}[n] = \frac{1}{N}\sum_{k=0}^{N-1} wave_{FFT}[k] \cdot \exp\left[-2\pi ikn / N\right]$$

$wave_{FFT}$ を連続フーリエ変換と同じ結果を得るようにスケールするには、$wave_{org}$ のウェーブデータ（waveorig）のサンプル数 N でスペクトル値を割る必要があります。

ただし、実際のウェーブの FFT では、正の周波数を含む正のスペクトルだけが計算されます。
これは、フーリエ変換と FFT の振幅を比較する時、負の周波数に対応する同一の負のスペクトルを考慮するために、正のスペクトルを 2 倍にする必要があります（ただし、負のスペクトル値を持たない $wave_{FFT}[0]$は除く）。

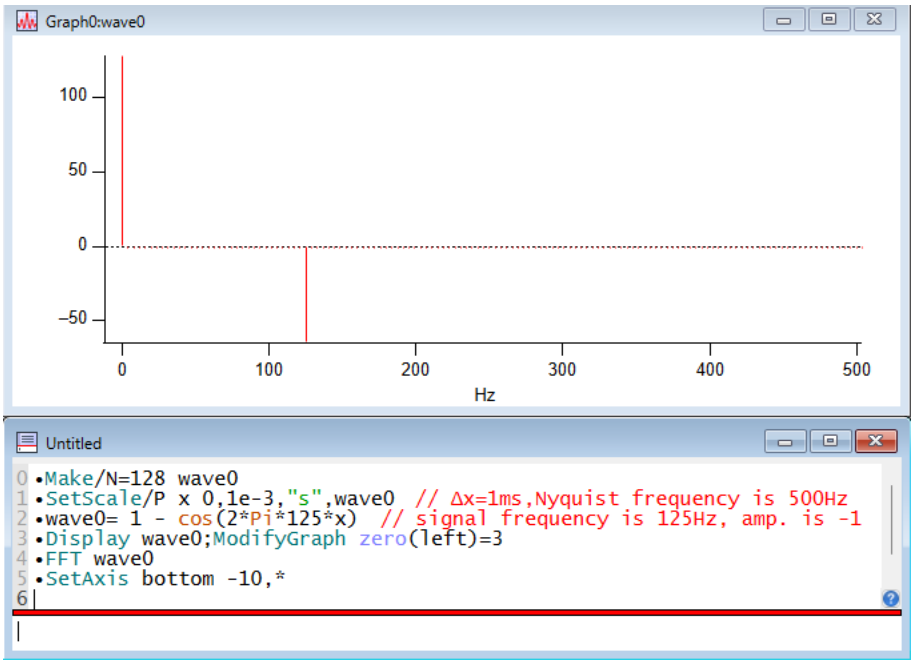
例えば、ここでは、実際のウェーブの片側スペクトルを計算し、予想されるフーリエ変換値と比較しています：

```
Make/N=128 wave0

SetScale/P x 0,1e-3,"s",wave0              // Δx=1ms, ナイキスト周波数は 500Hz

wave0= 1 - cos(2*Pi*125*x)                 // 信号周波数は 125Hz, 振幅は -1
Display wave0;ModifyGraph zero(left)=3
```



FFT wave0

Igor は「片側」のスペクトルを計算し、グラフを更新します：



フーリエ変換は、ゼロ周波数（DC）の結果として 1 を予測します。
これは、FFT 値 128 を入力値の数（これも 128）で割った結果と一致します。
一般に、フーリエ変換のゼロ周波数における値は：

$$Fourier\ Transform\ Amplitude(0) = \frac{1}{N} real\left(r2polar\left(wave_{FFT}(0)\right)\right)$$

フーリエ変換は、-125Hz の周波数で振幅が(-0.5 + i0)のスペクトルピークを予測し、正のスペクトルにおいても +125Hz で同じピークを予測します。
それらの予測されるピークの和は(-1+i0)となります。

（この例は、虚数を 0 に保つために意図的に作成したものです。実数は、入力信号に +cos(...) ではなく -cos(...) が含まれているため、負の値になります。）

Igor は正のスペクトルピークのみを計算したため、負の周波数ピークの双子を考慮して２倍します。
２倍した -128 のピークを入力値の数で割ると、フーリエ変換の予測と一致する (-1+i0) になります。
一般に、0 以外の周波数 f におけるフーリエ変換の値は次のとおりです：

$$Fourier\ Transform\ Amplitude(f) = \frac{2}{N} real\left(r2polar\left(wave_{FFT}(f)\right)\right)$$

唯一の例外は、ナイキスト周波数値（片側 FFT 結果の最後の値）で、片側変換での値は両側変換での値と同じです（他のすべての周波数値とは異なり、両側変換では１つのナイキスト周波数値のみが計算されるため）。
従って、

$$Fourier\ Transform\ Amplitude(f_{Nyquist}) = \frac{1}{N} real\left(r2polar\left(wave_{FFT}(f_{Nyquist})\right)\right)$$

周波数分解能は $\Delta X_{FFT} = 1/(N_{original} \cdot \Delta x_{original})$、または 1/(128×1e-3) = 7.8125 Hz。
これは、以下のコマンドを実行することで確認できます：

```
Print deltax(wave0)
```

これは履歴エリアに次のように出力されます：

```
        7.8125
```

入力信号が周波数分解能の倍数ではない場合（この例では 7.8125 Hz の倍数）、信号のエネルギーは FFT 結果の２つの最も近い周波数に分割されることに注意してください。
これは、連続フーリエ変換の挙動とは異なります。

位相極性

結果の位相に関するフーリエ変換には２つの異なる定義があります。
Igor は、他の多くの参考文献とは符号が異なる方法を使っています。
これは、Igor の FFT の結果と他のプログラムの FFT の結果を比較する場合に主に重要になります。
２つの方法間の変換は、次のように行います。

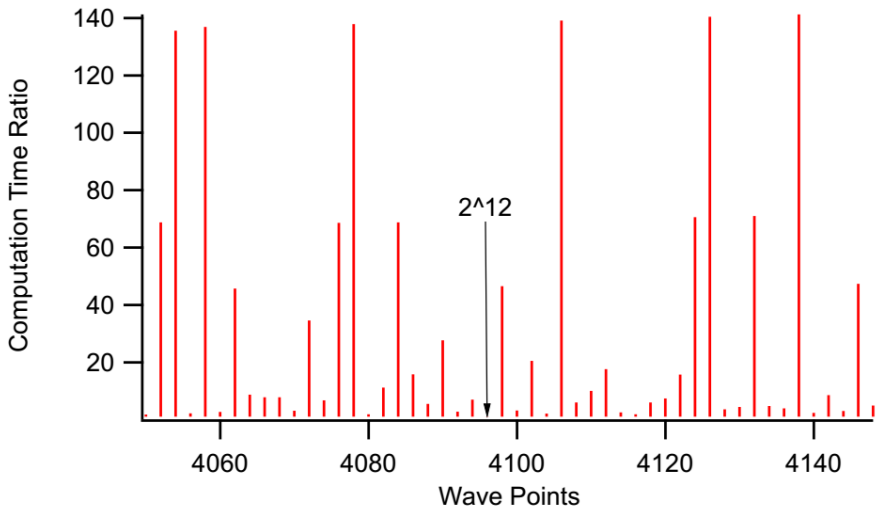
```
FFT wave0; wave0=conj(wave0)          // 位相角を、虚数部の符号を変更することで逆相にする
```

FFT と IFFT がグラフに与える影響

Igor は、デフォルトでは「Lines between points」モードで複素数ウェーブを表示します。
しかし、上記のように、グラフに表示されているウェーブに対して FFT を実行し、そのウェーブの表示モードが「Lines between points」になっている場合、Igor はその表示モードを「Sticks to zero」に変更します。
また、グラフに表示されているウェーブに対して IFFT を実行し、そのウェーブの表示モードが「Sticks to zero」になっている場合、Igor はその表示モードを「Lines between points」に変更します。

FFT の速度に対するポイント数の影響

プライムファクターFFT アルゴリズムは、ポイントの数が 2 のべき乗である必要はありません。
しかし、ポイントの数が小さな素数に因数分解できない場合、FFT の速度が劇的に低下する可能性があります。
次のグラフは、入力ポイントの数に応じて、複素数ベクトルに対する FFT の相対的な速度を示しています（横軸：ポイント数、縦軸：計算速度の比率）：

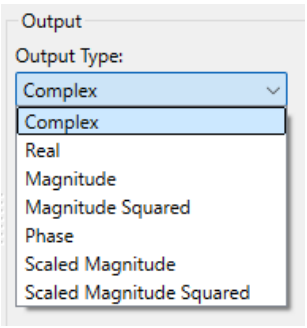


矢印は 4096 ポイント (2^12) の位置にあります。
この話の教訓は、素因数の大きいウェーブサイズは避けるべきだということです。
例えば、4078 のウェーブは、素因数 2039 と 2 を含むため、比較的長い時間がかかります。
ウェーブのサイズを２ポイント増やすと、計算は 71 倍速くなります（4080 の素因数は 2*2*2*2*3*5*17 です）。
最高のパフォーマンスを得るには、ポイントの数は 2 のべき乗、たとえば 4096 にする必要があります。

振幅と位相の測定

FFT 演算は、出力タイプを選択することで、直接的に複素数、実数、振幅、振幅の二乗、または位相の結果を生成できます。

FFT コマンドの複素数ウェーブの結果を使う場合、WaveTransform (キーワード magnitude、magsqr、および phase) または WaveMetrics Procedures フォルダーにあるさまざまなプロシージャ (次のセクションで説明) を使って、振幅と位相を計算できます。



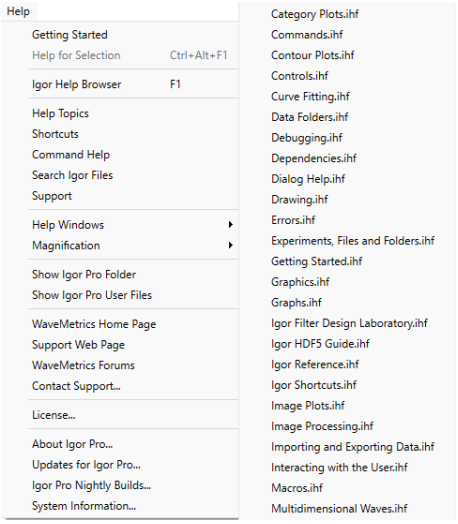
位相ウェーブを展開（±180 度間で発生する位相のジャンプを排除）したい場合は、Unwarp コマンドまたは Data メニューの Unwarp Waves ダイアログを使います。
コマンドのヘルプ Unwarp を参照してください。
２次元では、ImageUnwrapPhase コマンドを使用できます。

プロシージャを使った振幅と位相

下位互換性のため、WaveMetrics が提供する WaveMetrics Procedure：Analysis：:DSP (Fourier Etc) フォルダー内のプロシージャを使って、FFT の振幅と位相を計算することができます。

#include メカニズムを使って、それらにアクセスすることができます。プロシージャファイルを含める方法については、ヘルプ Include Statement を参照してください。

Help → Help Windows メニューからアクセスできる WM プロシージャインデックスヘルプファイルは、使用可能なルーチンとその使用方法を確認するのに便利です。



FTMagPhase 関数

FTMagPhase 関数は、Igor の FFT 操作への簡単なインターフェイスを提供します。
FTMagPhase には、以下の機能があります。

- 結果の自動表示
- 元のデータは変更されない
- デシベル単位での振幅表示が可能
- オプションの度またはラジアンで位相を表示
- オプションの１次元位相のアンラッピング
- 解像度の向上
- オプションのウィンドウ処理で２の「べき」ではないデータに対応

これらの関数にアクセスするには、プロシージャファイルで `#include <FTMagPhase>` を使ってください。

FTMagPhaseThreshold 関数

FTMagPhaseThreshold 関数は、FTMagPhase プロシージャと同じですが、次の追加機能があります：

- 低振幅の信号の位相値を無視

これらの関数にアクセスするには、プロシージャファイルで `#include <FTMagPhaseThreshold>` を使ってください。

DFTMagPhase 関数

DFTMagPhase 関数は、FTMagPhase プロシージャと似ていますが、計算にはより遅い離散フーリエ変換が使われる点が異なります。

- ユーザーが選択可能な周波数の開始と終了
- ユーザーが選択可能な周波数帯の数

このプロシージャには、ユーザーが選択した 1 つの周波数における振幅と位相を計算する DFTAtOneFrequency プロシージャも含まれています。

これらの関数にアクセスするには、プロシージャファイルで `#include <DFTMagPhase>` を使ってください。

CmplxToMagPhase 関数

CmplxToMagPhase 関数は、FFT の結果である複素数ウェーブを、振幅と位相の別々のウェーブに変換します。FTMagPhase の機能の多くを備えていますが、FFT は実行しません。

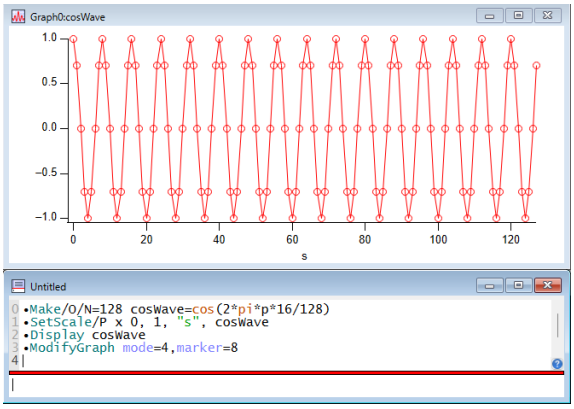
これらの関数にアクセスするには、プロシージャファイルで `#include <CmplxToMagPhase>` を使ってください。

スペクトルのウィンドウ処理

FFT 計算では、入力データが繰り返されるという仮定が立てられます。
これは、データの初期値と最終値が同じでない場合に重要です。
この繰り返しデータ仮定の結果の簡単な例を以下に示します。

データが 16 サイクルの完全なコサイン（cos）ウェーブをサンプリングしたものだとします：

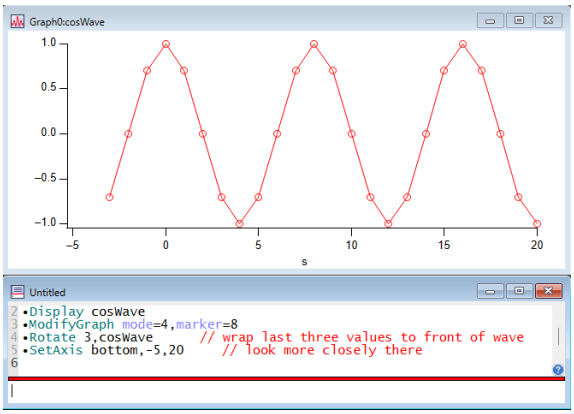
```
Make/O/N=128 cosWave=cos(2*pi*p*16/128)
SetScale/P x 0, 1, "s", cosWave
Display cosWave
ModifyGraph mode=4,marker=8
```



cosWave の最後の数ポイントを先頭にコピーすると、最初の数ポイントと完全に一致することに注目してください。

実際、Rotate コマンドでその操作を試してみます：

```
// 最後の 3 つの値をウェーブの先頭にラップ
Rotate 3,cosWave

// その部分を拡大
SetAxis bottom,-5,20
```

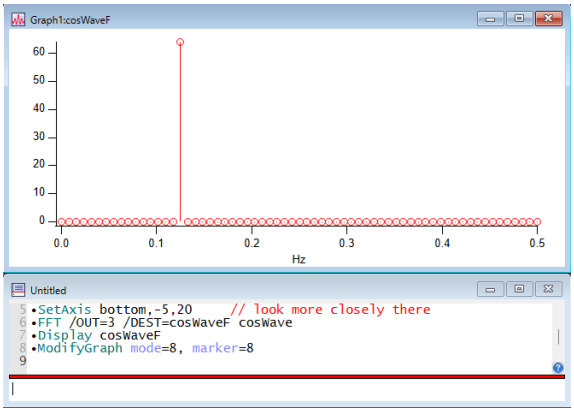


回転したポイントは x=-3、-2、および -1 の位置に現れます。
これは、FFT の観点からは不連続性が存在しないことを示しています。

不連続性が存在しないため、FFT の振幅結果は理想的な期待値と一致しています：

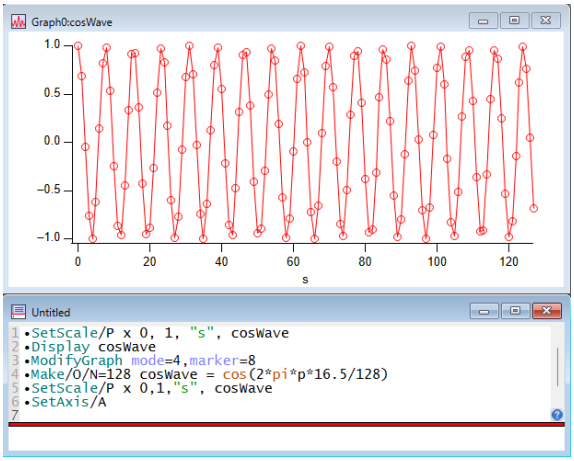
理想的な FFT 振幅 = コサイン振幅 × ポイント数 / 2
= 1 × 128 / 2 = 64

```
FFT /OUT=3 /DEST=cosWaveF cosWave
Display cosWaveF
ModifyGraph mode=8, marker=8
```



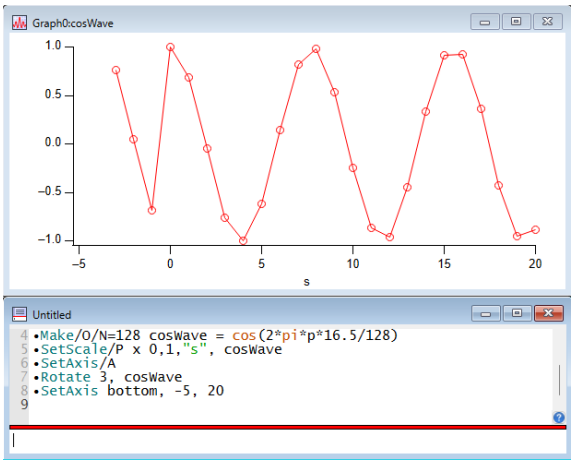
他のすべての FFT の振幅がゼロであることに注目してください。
次に、データを変更して、16.5 のコサイン周期になるようにします（改めて最初から実行してみます）：

```
// 最初の例の部分
Make/O/N=128 cosWave=cos(2*pi*p*16/128)
SetScale/P x 0, 1, "s", cosWave
Display cosWave
ModifyGraph mode=4,marker=8

// 周期の変更
Make/O/N=128 cosWave = cos(2*pi*p*16.5/128)
SetScale/P x 0,1,"s", cosWave
SetAxis/A
```



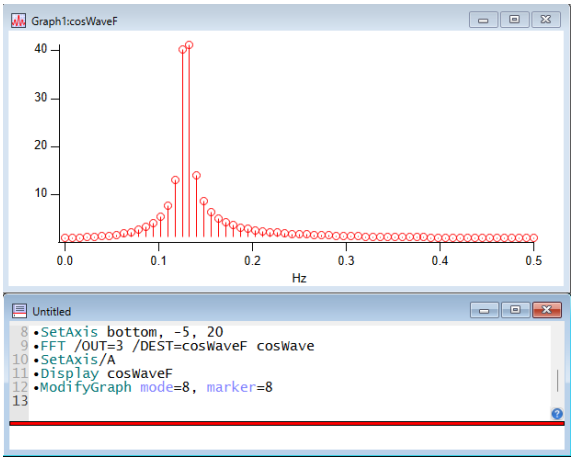
このデータを前述のように回転すると、FFT が回転していないデータのポイント 127 とポイント 0 の間に不連続性があると認識することがわかります。
次のグラフでは、元のポイント 127 が x=-1 に回転され、ポイント 0 は依然として x=0 の位置にあります。

```
Rotate 3, cosWave
SetAxis bottom, -5, 20
```



このデータの FFT を計算すると、不連続部分により、主コサインピークが周囲の振幅値に「漏れ」ます。

```
FFT /OUT=3 /DEST=cosWaveF cosWave
SetAxis/A
Display cosWaveF
ModifyGraph mode=8, marker=8
```

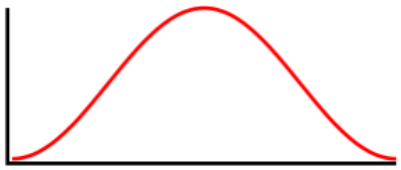


これらはスペクトルのウィンドウ処理とどのように関連しているのでしょうか？
スペクトルのウィンドウ処理は、この漏れを軽減し、より正確な FFT 結果を提供します。
具体的には、ウィンドウ処理は、漏れの影響を受ける隣接する FFT 値の数を減少させます。
一般的なウィンドウは、データの左右両端を滑らかに減衰させてゼロに近づけることで、この機能を実現します。

ハニング（Hanning）ウィンドウ

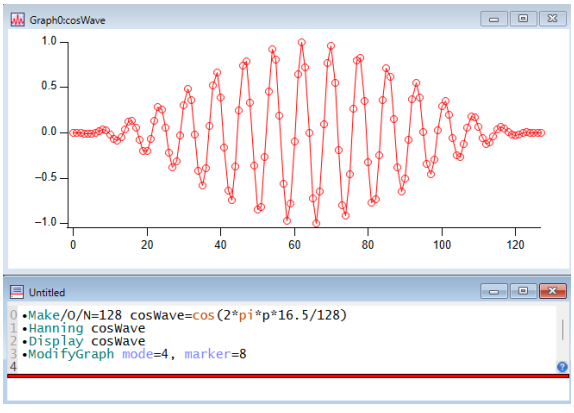
FFT を計算する前にデータをウィンドウ処理すると、上記の漏れを軽減することができます。
ハニングウィンドウは、次の式で定義される単純なコサイン関数です。

$$hanning[p] = \frac{1 - \cos\left(\frac{2\pi p}{N-1}\right)}{2}$$



16.5 サイクルのコサインウェーブデータにハンニングウィンドウを適用してみます：

```
Make/O/N=128 cosWave=cos(2*pi*p*16.5/128)
Hanning cosWave
Display cosWave
ModifyGraph mode=4, marker=8
```

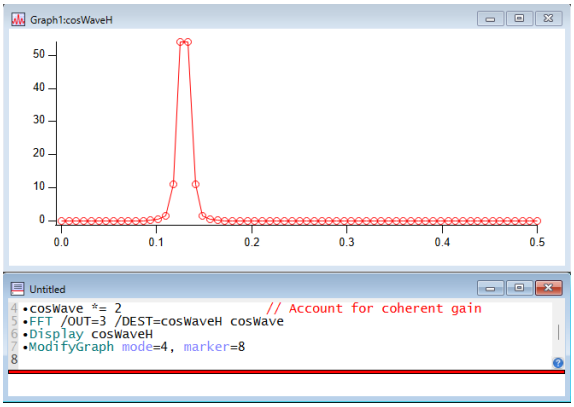


ウェーブの端をゼロに滑らかにすることで、端を折り返しても不連続感がありません。

データにウィンドウを適用すると、エネルギーが失われます。
アプリケーションによっては、コヒーレントまたは非コヒーレントな増幅を考慮して出力をスケールする必要がある場合があります。
コヒーレント利得は、時々、振幅因子で表され、これはウィンドウ関数の係数の和に等しくなります。
非コヒーレント利得は、同じ係数の二乗和として定義されるパワー因子です。
検討している場合、補正係数はハンニングウィンドウのコヒーレント利得の逆数です。

したがって、FFT の振幅を 2 倍にすることで補正できます：

```
cosWave *= 2                    // コヒーレント利得を考慮
FFT /OUT=3 /DEST=cosWaveH cosWave
Display cosWaveH
ModifyGraph mode=4, marker=8
```



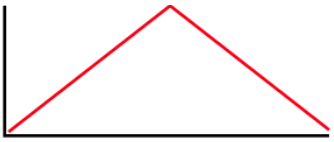
ピーク付近の周波数値は漏れの影響を比較的受けにくく、振幅は理想値の 64 に近くなります。

その他のウィンドウ

ハンニングウィンドウは究極の窓ではありません。
漏れをより抑える他のウィンドウは、ピークを広くする傾向があります。
FFT および WindowFunction コマンドには、以下の組み込みウィンドウ関数が用意されています：Hanning、Hamming、Bartlett、Blackman、Cosa(x)、KaiserBessel、Parzen、Riemann、Poisson。

ユーザー定義関数を記述するか、三角形のウィンドウを適用する次のような単純なウェーブ代入文を実行することで、他のウィンドウを作成することができます。

```
data *= 1-abs(2*p/numpnts(data)-1)
```



X スケーリングの複雑化を避けるために、ポイントインデックスを使ってください。
ウィンドウの効果は、完全なコサインウェーブ、できればサンプリング周波数の 1/4 (ナイキスト周波数の 1/2) のコサインウェーブに適用して確認することができます。

その他のウィンドウは、WaveMetrics が提供する「DSP Window Functions」プロシージャファイルに用意されています。

多次元ウィンドウ処理

画像に対して FFT を行う場合、画像の境界がシャープであるためにアーチファクトが発生することがよくあります。
１次元ウェーブの場合と同様、画像にウィンドウを適用することで、FFT の結果を改善することができます。

画像にウィンドウを適用するには、Hanning、Hamming、Bartlett、Blackman、および Kaiser ウィンドウフィルターを実装する ImageWindow コマンドを使う必要があります。
詳細については、ImageWindow コマンドのヘルプを参照してください。

パワースペクトル

周期グラム

信号 s(t) の周期グラムは、次式で与えられるパワースペクトルの推定値です。

$$P(f)=\frac{|F(f)|^2}{N}$$

ここで、F(f) は離散フーリエ変換 (DFT) によって計算された s(t) のフーリエ変換であり、N は正規化 (通常はデータポイントの数) です。

FFT を使って周期グラムを計算することもできますが、DSPPeriodogram コマンドを使うとより簡単です。
これは同じウィンドウ関数を持ちますが、DC 項を抑制したり、結果を次のように dB で表示したりするための独自の正規化を選択することもできます：

20log10(F/F0)

または

10log10(P/P0)

P0 は、P の最大値またはユーザーが指定した参照値のいずれかです。

DSPPeriodogram は、クロスパワースペクトルも計算できます。
これは、２番目の信号のフーリエ変換の複素共役と１番目の信号のフーリエ変換の積です。

$$P(f)=\frac{F(f)G^*(f)}{N}$$

ここで、F(f) および G(f) は２つのウェーブの DFT です。

パワースペクトル密度関数

「Power Spectral Density」プロシージャファイルで提供される PowerSpectralDensity ルーチンは、入力データのセグメントのパワースペクトルを平均してパワースペクトル密度を計算します。
これは、FFT または DSPPeriodogram コマンドの新しい組み込み機能を利用していない初期のプロシージャファイルです。
このプロシージャは、下位互換性のために引き続きサポートされています。

PowerSpectralDensity 関数は、入力として長いデータウェーブを受け取り、パワースペクトル密度関数を計算します。
これらのプロシージャには、以下の機能があります：

- 結果の自動表示
- 元のデータは変更されない
- ウィンドウ処理関数のポップアップリスト
- ユーザー設定可能なセグメント長

これらの関数にアクセスするには、プロシージャファイルで `#include <Power Spectral Density>` を使ってください。
プロシージャファイルを含める方法については、ヘルプ Include Statement を参照してください。

PSD Demo Experiment

PSD Demo (Examples:Analysis: フォルダー内) は、PowerSpectralDensity プロシージャを使い、結果に適用されるスケーリングの根拠を含め、その動作を詳しく説明しています。

ヒルベルト変換

関数 f(x) のヒルベルト変換は、次のように定義されます：

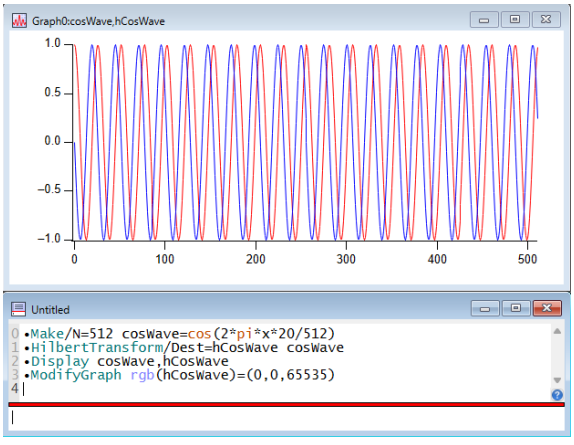
$$F_H(x) = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{f(t)}{t-x}dt$$

積分は、Cauchy の主値として評価されます。
数値計算では、積分を畳み込みとして表現するのが通例です。

$$F_H(x) = \left(\frac{-1}{\pi x}\right) \otimes f(x)$$

フーリエ変換の逆変換が i*sgn(x) であることを踏まえると、フーリエ変換の畳み込み定理を用いてヒルベルト変換を評価することができます。
HilbertTransform コマンドは、便利なショートカットとして機能します。
次の例では、コサイン関数のヒルベルト変換を計算し、正弦関数を得ます：

```
Make/N=512 cosWave=cos(2*pi*x*20/512)
HilbertTransform/Dest=hCosWave cosWave
Display cosWave,hCosWave
ModifyGraph rgb(hCosWave)=(0,0,65535)
```



時間周波数解析

信号のフーリエスペクトルを計算すると、フーリエ変換に含まれるすべての位相情報が失われます。
信号に含まれる周波数は特定できますが、これらの周波数が信号にいつ現れるかはわかりません。
例えば、次の信号を考えてみます：

f(t)のスペクトル表現は、2 つの周波数 f_1 と f_2 を交換しても本質的に変化しません。
つまり、フーリエスペクトルは、スペクトルが時間とともに変動する信号の分析ツールとして最適ではありません。
この問題に対する解決策のひとつが、いわゆる「短時間フーリエ変換」です。
これは、スライドする時間ウィンドウを使ってフーリエスペクトルを計算します。

時間周波数解析のための追加のツールとして、ウィグナー変換と連続ウェーブレット変換（CWT）があります。

ウィグナー（Wigner）変換

ウィグナー変換（ウィグナー分布関数または WDF とも呼ばれます）は、 1 次元時間信号 U(t)を 2 次元時間周波数表現に変換します。
概念的に、WDF は音楽の楽譜に類似しており、時間軸が水平方向に、周波数（音符）が垂直方向にプロットされています。
WDF は次の式で定義されます：

$$W(t,\nu)=\int_{-\infty}^{\infty} U(t+x/2)U^{*}(t-x/2)e^{-i2\pi x\nu}\,dx$$

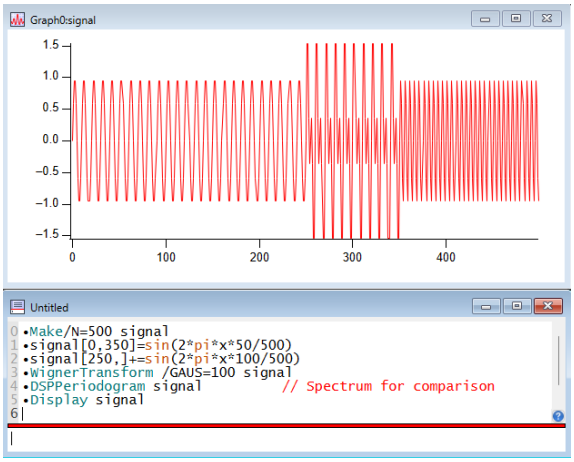
WDF W(t,v) は実数であることに注目してください（これは、それがエルミート（Hermitian）量の数値のフーリエ変換であることからわかります）。
WDF はまた、Ambiguity（あいまいさ）関数の 2 次元フーリエ変換です。

局所化されたスペクトルは、WDF を有限領域 dtdn で積分することで導出できます。
t と n の両方にガウス重み関数を使い、最小不確実性条件 dtdn=1 を選択することで、局所スペクトルの推定値を得ます。

$$\widehat{W}(t,\nu;\delta t)\propto\left|\int U(t')\exp\left[-2\pi\left(\frac{t-t'}{\delta t}\right)\right]\exp(-i2\pi\nu t')\,dt'\right|^{2}$$

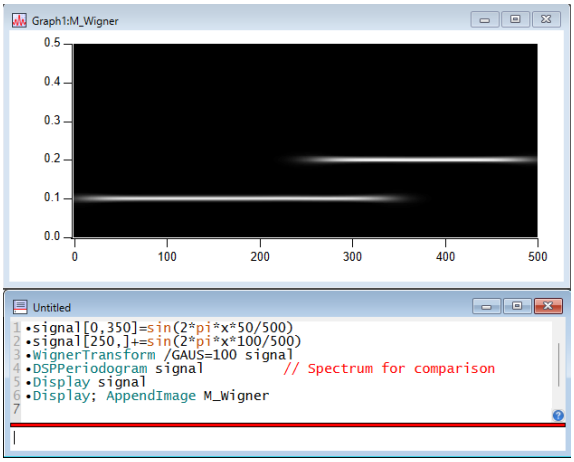
WignerTransform コマンドの応用例として、 2 つの周波数からなる信号を考えます：

```
Make/N=500 signal
signal[0,350]=sin(2*pi*x*50/500)
signal[250,]+=sin(2*pi*x*100/500)
WignerTransform /GAUS=100 signal
DSPPeriodogram signal          // 比較用スペクトル
Display signal
```



この例で使っている信号は、時間的にわずかに重なる 2 つの「純粋な」周波数で構成されています：

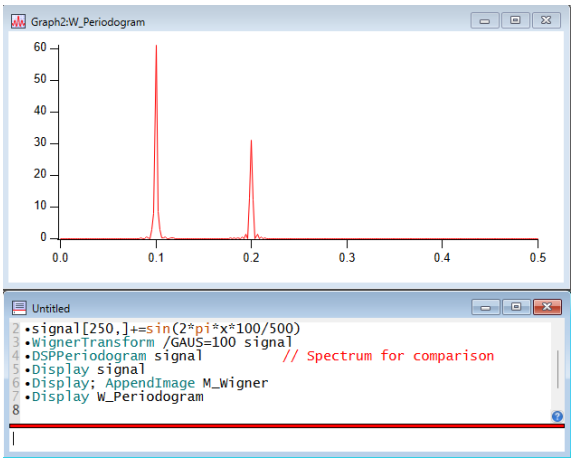
```
Display; AppendImage M_Wigner
```



時間依存性はウィグナー変換において明確に観察されます。
水平方向（時間）の移行は急峻ではありません。
これは主に、最小不確定関係 dtdn=1 を適用したことによるものですが、計算上のエッジ効果も一因となっています。

比較すると、信号のスペクトルは 2 つの周波数の存在を明確に示しているものの、信号の周波数成分の時間的変動については何らの示唆も与えていません。
さらに、2 つの周波数におけるパワーの違いは、持続時間の違いまたは振幅の違いのいずれかに起因する可能性があります。

```
Display W_Periodogram
```



連続ウェーブレット変換

連続ウェーブレット変換（CWT）は、信号の時間周波数表現であり、グラフ上ではウィグナー変換と表面的な類似性を持っています。

ウェーブレット変換は、信号 s(t) と、主関数の並進と拡大によって生成される関数の集合との畳み込みです。
主関数は「マザーウェーブレット」と呼ばれ、変換または拡大された関数は「ウェーブレット」と呼ばれます。
数学的には、CWT は次のように表されます。

$$W(a,b)=\frac{1}{\sqrt{a}}\int s(t)\psi\left(\frac{t-b}{a}\right)dt$$

ここで、b はウェーブレットの時間変換、a はウェーブレットの拡大率を表します。

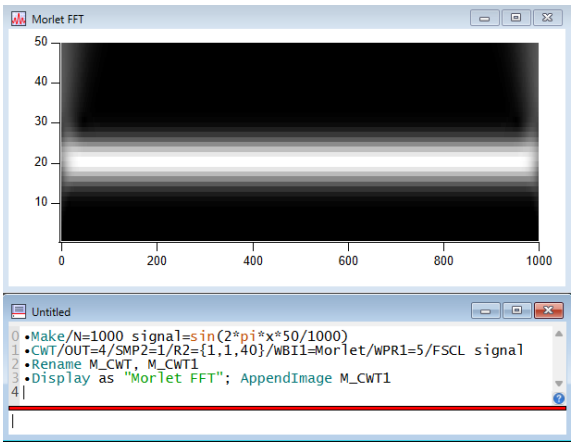
計算上の観点からは、FFT を使って畳み込みを計算するのが当然であり、その結果は s(t) の適切なサンプリングに依存することが示唆されます。

マザーウェーブレットが複素数である場合、CWT も複素数値関数となります。
そうでない場合、CWT は実数値関数です。
CWT の 2 乗の大きさはパワースペクトルに相当するため、CWT の典型的な表示（画像）は、時間オフセット b の

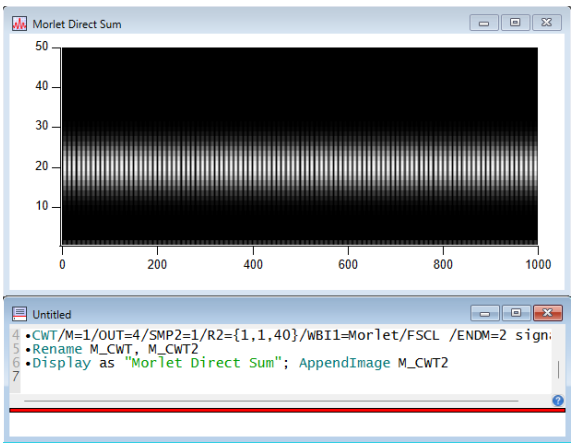
関数としてのパワースペクトルを表しています。
ただし、CWT の正確な形式はマザーウェーブレット γ の選択に依存するため、CWT の二乗振幅とパワースペクトルの間の等価性の程度は応用依存である点に注意すべきです。

CWT 演算は、FFT と直接和法の両方を使って実装されています。
どちらの方法でも、デルタ関数を入力として使って有効なウェーブレットの表現を得ることができます。
２つの CWT の結果を比較する場合は、必ず、両方がまったく同じウェーブレット関数の定義、同じ正規化、および同じ計算方法を使用していることを確認してください。
例えば、

```
Make/N=1000 signal=sin(2*pi*x*50/1000)
CWT/OUT=4/SMP2=1/R2={1,1,40}/WBI1=Morlet
       /WPR1=5/FSCL signal
Rename M_CWT, M_CWT1
Display as "Morlet FFT"; AppendImage M_CWT1
```

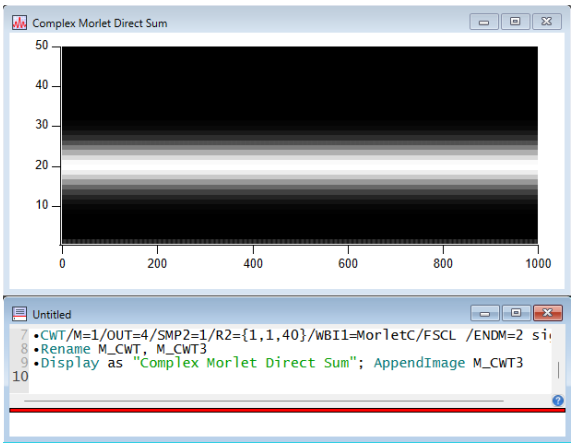



```
CWT/M=1/OUT=4/SMP2=1/R2={1,1,40}/WBI1=Morlet
       /FSCL /ENDM=2 signal
Rename M_CWT, M_CWT2
Display as "Morlet Direct Sum";
       AppendImage M_CWT2
```



Morlet ウェーブレットの複雑な波形を直接和法（/M=1）で用い、二乗振幅を表示すると、次のように得られます：

```
CWT/M=1/OUT=4/SMP2=1/R2={1,1,40}
       /WBI1=MorletC/FSCL /ENDM=2 signal
Rename M_CWT, M_CWT3
Display as "Complex Morlet Direct Sum";
       AppendImage M_CWT3
```



最後の画像は、FFT アプローチを使って生成された画像と本質的に同じ結果を示していますが、この場合、エッジ効果は完全に消失しています。

離散ウェーブレット変換

DWT はフーリエ変換と同様に、信号を基底関数の集合を用いて分解する手法です。
フーリエ変換では、基底は正弦波と余弦波で構成され、展開は１つのパラメーターで表されます。
ウェーブレット変換では、展開には２つのパラメーターがあり、２つのパラメーターに対応する拡大とオフセットを用いて、１つの「マザー」ウェーブレットから関数（ウェーブレット）が生成されます。

$$f(t)=\sum_a\sum_b c_{ab}\psi_{ab}(t)$$

ここで、2 つのパラメーターによる展開係数は、次のように表されます：

$$c_{ab}=\int f(t)\psi_{ab}(t)dt$$

そして、ウェーブレットは次の条件を満たします：

$$\psi_{ab}(t)=2^{a/2}\Psi\left(2^a t-b\right)$$

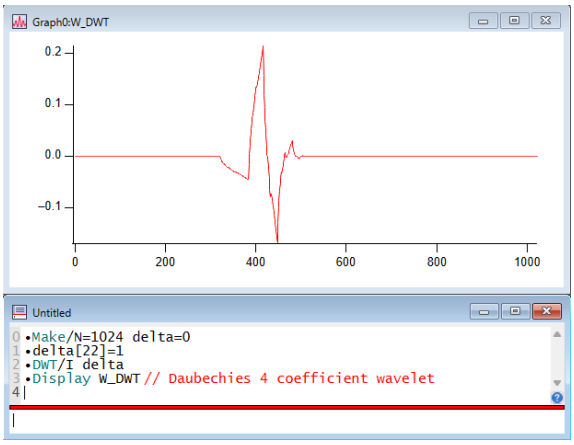
ここで、Y はマザーウェーブレット、a は拡張パラメーター、b はオフセットパラメーターです。

２つのパラメーターによる表現は、１次元信号からより高い次元に移行すると、すぐに複雑になる可能性があります。
さらに、各スケールにおける係数の数が 2 の累乗として変化するため、1 次元信号の DWT は 2 次元画像として便利に表現できません（CWT の場合とは異なります）。
従って、変換の結果を「パッキング」して、入力と同じ次元数を持つようにすることが一般的です。
例えば、入力が 128 (=27) ポイントの 1D ウェーブの場合、7-1=6 の重要なスケールが次のように配置されます：

スケール	保存場所
1	64-127
2	32-63
3	16-31
4	8-15
5	4-7
6	2-3

DWT の定義の興味深い結果として、デルタ関数の適切な形を変換することで、ウェーブレットの形状を特定することができます。
例えば、

```
Make/N=1024 delta=0
delta[22]=1
DWT/I delta
// Daubechies 4 次ウェーブレット係数
Display W_DWT
```

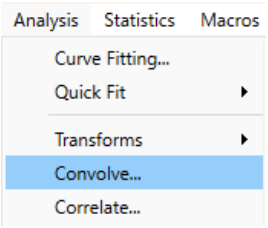


畳み込み

畳み込みを使って、入力信号に対する線形システムの応答を計算することができます。
線形システムは、そのインパルス応答によって定義されます。

入力信号とインパルス応答の畳み込みが、出力信号の応答です。
畳み込みは、周波数領域におけるフィルタリングの時間領域の同等のものです。

平滑化は、畳み込みの一種です – 詳細はヘルプ Smoothing を参照してください。

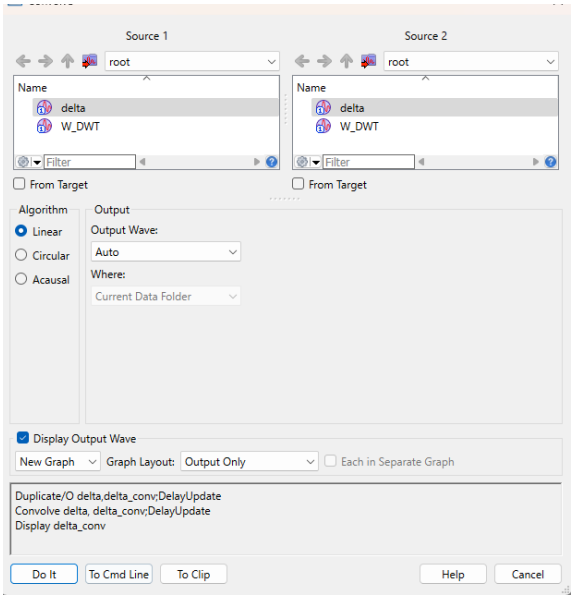


FilterFIR コマンドは、時間領域での畳み込みを実装します – 「デジタルフィルタリング」を参照してください。

Igor は、Convolve コマンドで一般的な畳み込みを実装しています。
Convolve コマンドを使うには、メニュー Analysis → Convolve を選択します。

内蔵の畳み込みコマンドは、「ソース」と「宛先」という 2 つのウェーブの畳み込みを計算し、その結果で宛先のウェーブを上書きします。
このコマンドでは、1 つのソースウェーブを複数の宛先ウェーブと畳み込むこともできます（その場合は、各ケースの結果で対応する宛先ウェーブが上書きされます）。

Convolve ダイアログでは、2 番目のウェーブを新しい宛先ウェーブに事前に複製することで、より柔軟な操作を行うことができます。



ソースウェーブが実数値の場合、各宛先ウェーブも実数値でなければなりません。
ソースウェーブが複素数の場合、各宛先ウェーブも複素数でなければなりません。
倍精度および単精度のウェーブは自由に混在させることができます。
計算は、より高い精度で行われます。

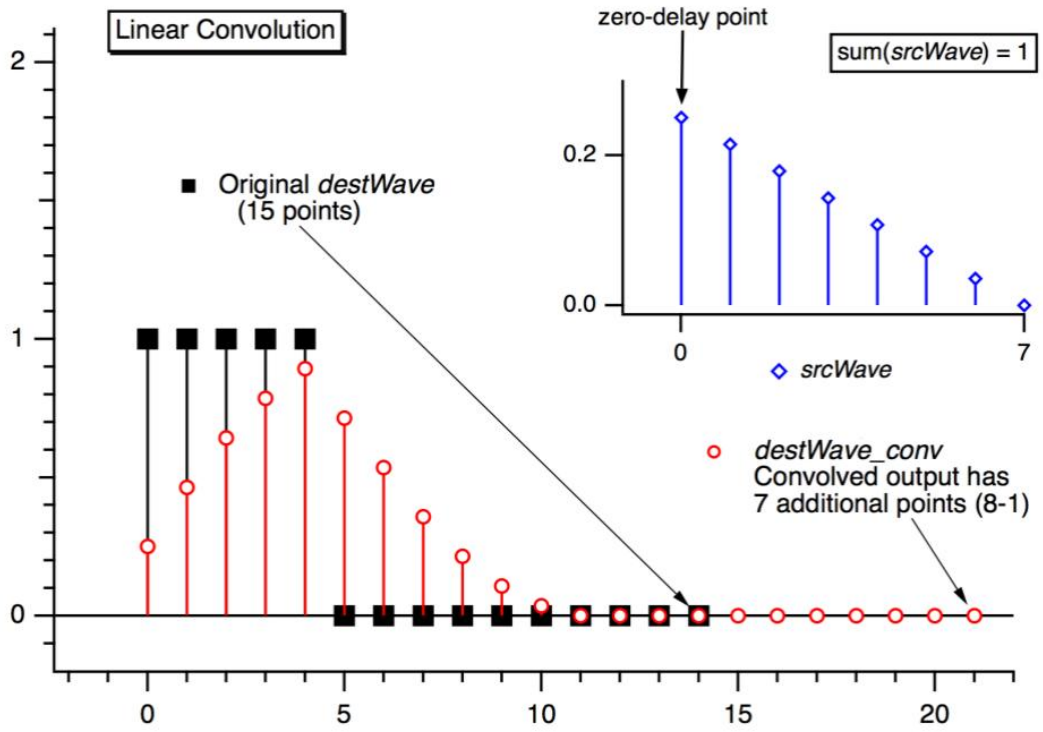
畳み込みは、畳み込みの対象となるポイントの前後の隣接するポイントを組み合わせるため、ウェーブの端には十分な隣接ポイントが存在しません。
「Convolve」ダイアログの「Algorithm」グループには、これらの欠落したポイントに対応するための 3 つのアルゴリズムが表示されます。
選択したアルゴリズムに応じて、目的のウェーブの長さは、ソースウェーブの長さから 1 を引いた分だけ長くなる場合があります。

Linear（線形）アルゴリズムは、Smooth コマンドのゼロ端効果メソッドと類似しています。
欠落した隣接するポイントの値にゼロが代入されます。

Circular（円形）アルゴリズムは、ラップ（Wrap）エンド効果法と類似しています。
このアルゴリズムは、データが無限に繰り返し続けると仮定される場合に適しています。

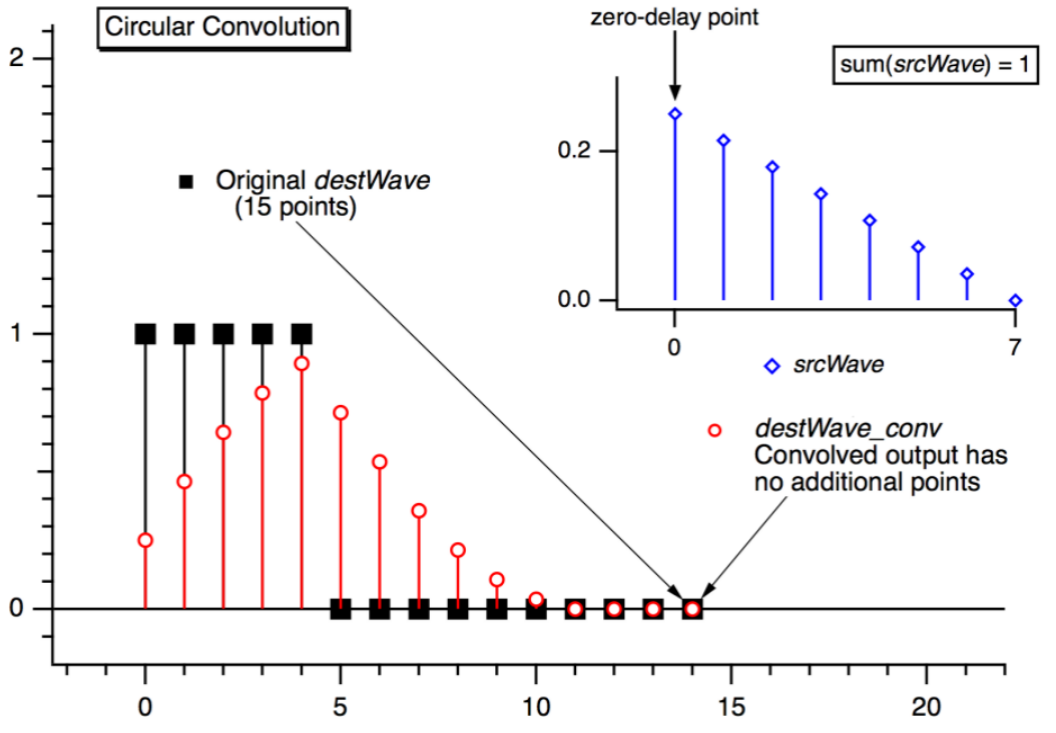
非因果アルゴリズムは、Linear の特殊なケースであり、Linear が導入する時間遅延を排除したものです。

選択したアルゴリズムに応じて、宛先ウェーブの点数は、ソースウェーブのポイント数から 1 を引いた数だけ増加する場合があります。線形および非因果の畳み込みでは、
まず、宛先ウェーブは、ソースウェーブのポイント数より 1 少ない数だけゼロで埋められます。
これにより、円形畳み込みで発生する「ラップアラウンド」効果が防止されます。
非因果畳み込み後、ゼロパディングされたポイントは削除され、線形畳み込み後には保持されます。

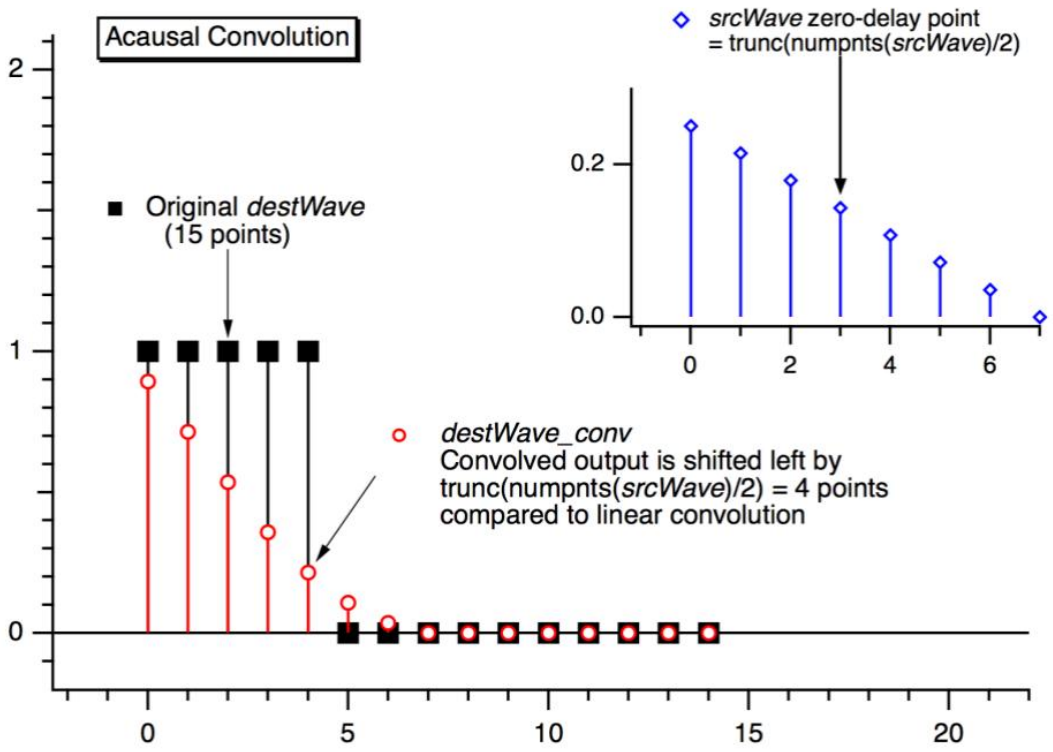


srcWave の最初のポイントが遅延なし（t = 0）に対応するインパルス応答（またはフィルター係数）がソースウェーブに含まれている場合は、線形畳み込みを使います。

ソースウェーブと宛先ウェーブのデータが無限に繰り返される（または、終了ポイントから開始ポイントに戻って繰り返される）場合、つまりゼロパディングが必要ない場合、円形畳み込みを使います。

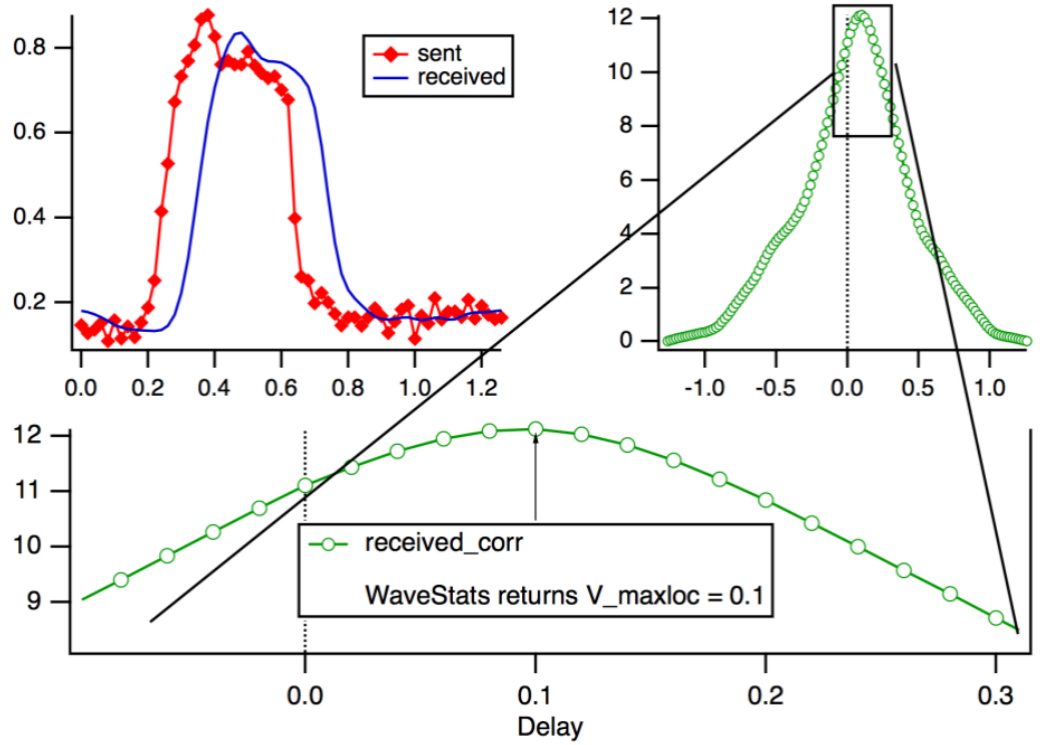


ソースウェーブに、ソースウェーブの中央のポイントが遅延なし（t = 0）に対応するインパルス応答が含まれている場合は、非因果畳み込みを使います。

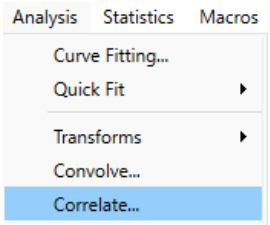


相関関係

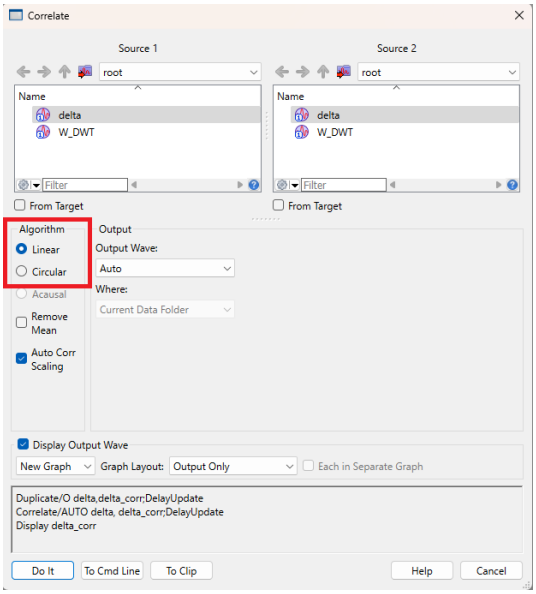
相関を使って、2 つのデータセットの類似性を比較することができます。
相関は、2 つの入力信号が互いにシフトされたときの類似性の尺度を計算します。
相関結果は、2 つの信号が最も一致する時点で最大値に達します。
2 つの信号が同一の場合、この最大値は t=0（遅延なし）で到達します。
2 つの信号の形状が類似しているが、一方の信号が時間的に遅延しており、さらにノイズが加わっている場合、相関は遅延を測定する適切な方法です。



Igor は、Correlate コマンドで相関を実装しています。
Analysis メニューから Correlate を選択できます。
ソースウェーブが、宛先ウェーブである場合もあります。
その場合、そのウェーブには「自己相関」が含まれます。
ソースと宛先が異なる場合、「相互相関」と呼ばれます。



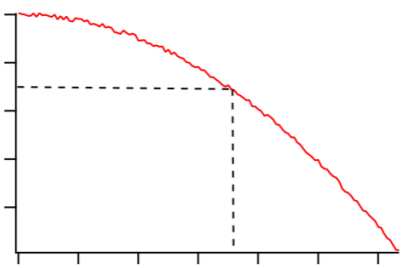
異なるタイプのソースウェーブと宛先ウェーブを組み合わせる場合、畳み込みと同様に相関についても同様の考慮事項が適用されます。
相関はまた、端効果に対処する必要があり、これらは円形相関と線形相関のアルゴリズム選択によって処理されます。
ヘルプ Convolution を参照してください。



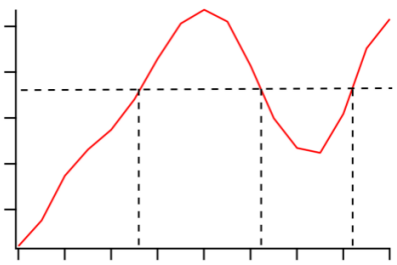
レベル検出

レベル検出は、データが指定された Y 値を通過または到達する X 座標の位置を特定するプロセスです。
これは「逆補間」と呼ばれることもあります。
別の言い方をすれば、レベル検出は「Y レベルが与えられた場合、それに対応する X 値は何か」という質問に答えるものです。
Igor は、この質問に対して 2 種類の答えを提供します。

1 つ目の回答では、Y データが単一の値からなるリストであり、その値が単調増加または単調減少するものと仮定しています。
この場合、Y 値に対応する X 値は 1 つだけです。
検索位置と方向は関係ないため、二分探索が最も適しています。
このようなデータには、BinarySearch または BinarySearchInterp 関数を使ってください。



別の回答では、Y データが不規則に変化すると仮定しています。
これは取得データの場合に該当します。
この場合、Y レベルを横切る X 値が複数存在し、返される X 値は検索の開始位置とデータ内の検索方向によって異なります。
FindLevel、FindLevels、EdgeStats、PulseStats コマンドは、このようなデータに対応しています。



関連するものの、異なる質問として「関数 y=f(x) が与えられたとき、y が 0（または他の値）となる x を求める」があります。
この質問は FindRoots コマンドで回答されます。
ヘルプ Finding Function Roots と FindRoots コマンドを参照してください。

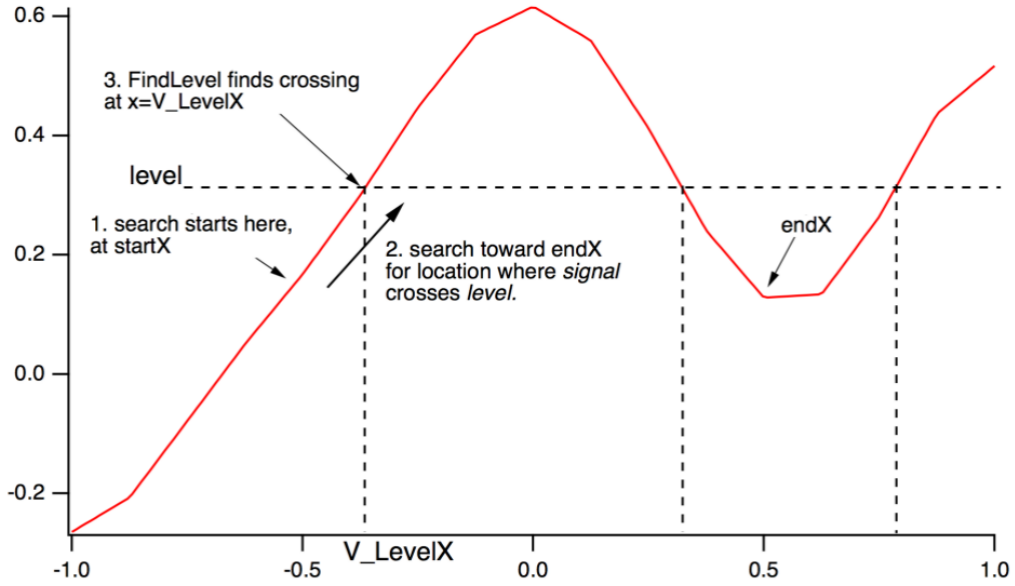
以下のセクションは、不規則に変化するデータにおけるレベルクロスを検出する方法について説明します。
ここで説明するコマンドはピークを検出するように設計されていません。
ピーク測定については、ヘルプ Peak Measuremen を参照してください。

ウェーブフォームデータ内のレベルを見つける

FindLevel コマンドを使って、ウェーブフォームデータ内の単一のレベル交差を見つけることができます。
また、FindLevels コマンドを使って、ウェーブフォームデータ内の複数のレベル交差を見つけることができます。
これらのコマンドはどちらも、ノイズの影響を軽減するために、検索するウェーブを滑らかにするオプションがあります。
データのサブレンジは、コマンドの /R フラグに指定した startX および endX 値に応じて、X 値の昇順または降順で検索できます。

FindLevel は、startX から開始し、endX の方向へ検索範囲内を移動し、最初のレベル交差点を検出します。
検索はシーケンシャルに実行されます。
FindLevel の出力は、2 つの特別な数値変数です：V_Flag と V_LevelX です。
V_Flag は検索の成功または失敗を示し（0 は成功）、V_LevelX にはレベル交差の X 座標が含まれます。

例えば、以下のデータがある場合：




```
FindLevel/R=(-0.5,0.5) signal,0.30
```

は、レベル交差情報を履歴エリアに出力します：

```
V_Flag=0; V_LevelX=-0.37497; V_rising=1;
```

XY データにおけるレベルを見つける

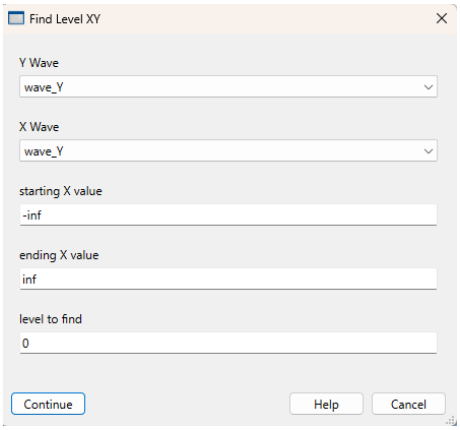
XY データでレベル交差を見つけるには、Y ウェーブを検索し、その X 値が X ウェーブのどこにあるかを特定します。
これには、X ウェーブの値を昇順または降順で並べ替える必要があります。
これを確実にするために、コマンド：

```
Sort xWave,xWave,yWave
```

により、xWave の値が昇順になり、XY の対応関係が保持されるようにウェーブをソートします。

次のプロシージャは、X 範囲内で Y レベルが交差する X 位置を見つけ、その結果を出力変数 V_LevelX に格納します。

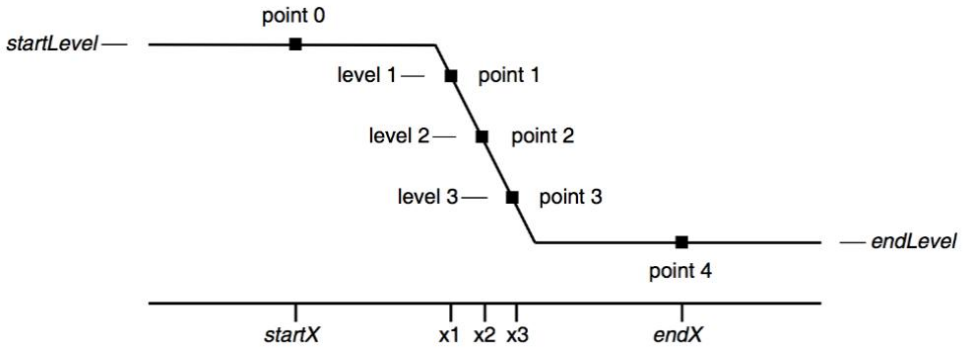
```
Function FindLevelXY()
        String swy,swx                          // 文字列はウェーブの名前を持つ
        Variable startX=-inf,endX=inf           // startX,endX は wx 内の値に対応
                                                // X スケーリングを含まない

        Variable level
        // ユーザーから情報を得るためにダイアログを表示
        Prompt swy,"Y Wave",popup WaveList("*",";","")
        Prompt swx,"X Wave",popup WaveList("*",";","")
        Prompt startX, "starting X value"
        Prompt endX, "ending X value"
        Prompt level, "level to find"
        DoPrompt "Find Level XY", swy,swx,startX, endX, level

        WAVE wx = $swx
        WAVE wy = $swy


        // ここからが面白い部分です
        Variable startP,endP                    // startX、endX をカバーするポイント範囲を計算
        startP=BinarySearch(wx,startX)
        endP=BinarySearch(wx,endX)
        FindLevel/Q/R=[startP,endP] wy,level    // Y ウェーブを探す、成功が前提
        Variable p1,m
        p1=x2pnt(wy,V_LevelX-deltaX(wy)/2)      // x2pnt ラウンド；それを回避
        // wy の V_levelX を挟む wx の 2 つのポイントの間を線形補間する
        m=(V_LevelX-pnt2x(wy,p1))/(pnt2x(wy,p1+1)-pnt2x(wy,p1))          // スロープ
        V_LevelX=wx[p1] + m * (wx[p1+1] -wx[p1] )       // ポイント-スロープ方程式
End
```



この関数は、見つからないレベル交差は処理しません。
FindLevel で Y ウェーブを検索した後、V_Flag をテストする部分だけが欠けています。

エッジ統計

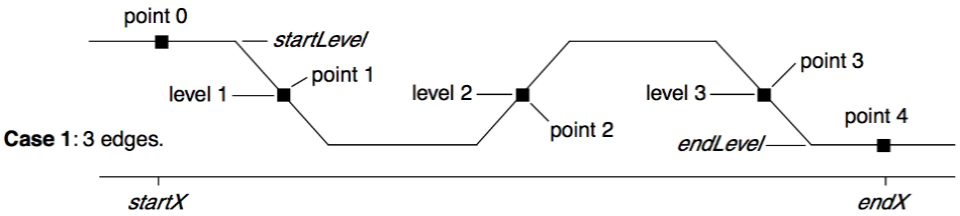
EdgeStats コマンドは、以下に示すように、１つのエッジを含むと予想されるウェーブの領域について、単純な統計情報（実際には測定値）を生成します。
複数のエッジが存在する場合、EdgeStats は最初に検出されたエッジに対して処理を行います。
エッジ統計情報は、EdgeStats のヘルプで説明している特別な変数に格納されています。
統計データは、エッジのレベル、検出された「ポイント」の X 座標またはポイントの位置、およびそれらの間の距離です。
これらの検出ポイントは、実際にはレベル交差の位置であり、通常は実際のウェーブフォームポイントの間に位置します（補間位置）。



EdgeStats は、FindLevel と同じ原理に基づいています。
EdgeStats は XY ペアでは機能しません。
ヘルプ Converting XY Data to a Waveform を参照してください。

パルス統計

PulseStats コマンドは、以下に示すように、3 つのエッジを含むと予想されるウェーブの領域について、単純な統計情報 (測定値) を生成します。
3 つを超えるエッジが存在する場合、PulseStats は最初に検出された最初の 3 つのエッジに対して処理を行います。
PulseStats は、エッジが 1 つまたは 2 つしかない他の 2 つのケースにも対応しています。
パルス統計情報は、PulseStats のヘルプで説明している特別な変数に格納されます。

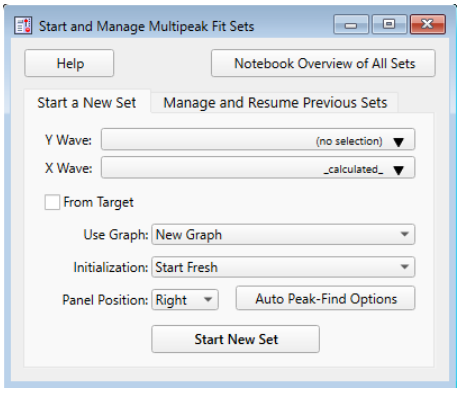


PulseStats は EdgeStats と同じ原理に基づいています。
PulseStats は XY ペアでは機能しません。
ヘルプ Converting XY Data to a Waveform を参照してください。

ピーク測定

ピーク測定の構築ブロックは、FindPeak コマンドです。
これを使って、独自のピーク測定プロシージャを構築することも、WaveMetrics が提供するプロシージャを使うこともできます。

Multipeak Fitting パッケージは、ピークデータへのカーブフィッティングのための強力な GUI およびプログラミングインターフェイスを提供します。
これは、さまざまなピーク形状やベースライン関数に適合します。
デモ Experiment（Multipeak Fitting Demo Experiment）で概要を見ることができます。



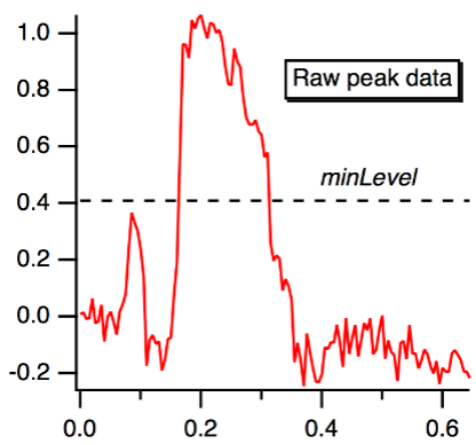
また、複数の Gauss、Lorentz、および Voigt のピークにフィッティングを行う、Multi-peak Fit という Experiment の例もあります。
Multi-peak Fit は、Tech Note 20 よりも包括的ではありませんが、使用が簡単です。

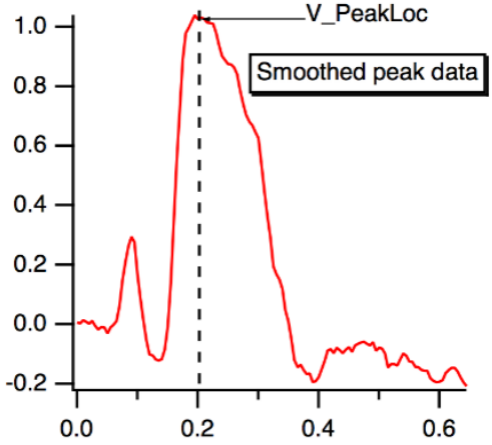
FindPeak コマンドは、ウェーブの平滑化された 1 次および 2 次導関数を分析して、ウェーブの最小値または最大値を検索します。
平滑化と微分は、入力ウェーブのコピーに対して行われます（入力ウェーブは変更されません）。
ピークの最大値は、滑らかにした 1 階微分のゼロ交点で検出され、その時点で滑らかにした 2 階微分が負の値となります。
最小値または最大値の位置は、特殊変数 V_PeakLoc に返されます。
FindPeak によって設定されるこの変数を含むその他の特殊変数については、コマンドのリファレンスに記載されています。

以下の説明は、FindPeak がこのようなコマンドを実行する時に経るプロセスを説明したものです：

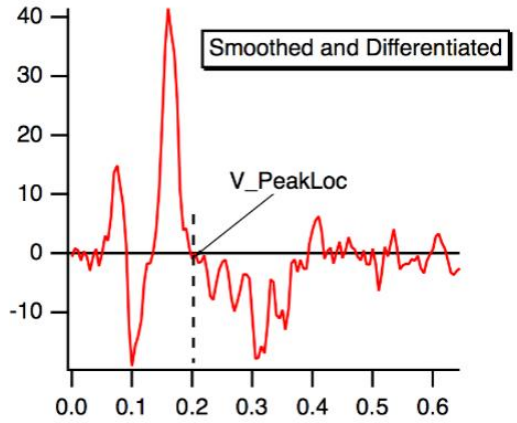
```
FindPeak/M=0.5/B=5 peakData          // 5 ポイントのスムージング、最小レベル = 0.5
```

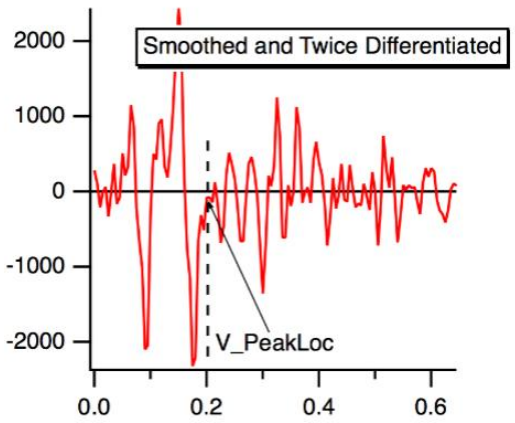
まずボックススムージングが実行されます：





次に、中央差分法を用いて 2 回の微分を行い、第 1 微分と第 2 微分を求めます：

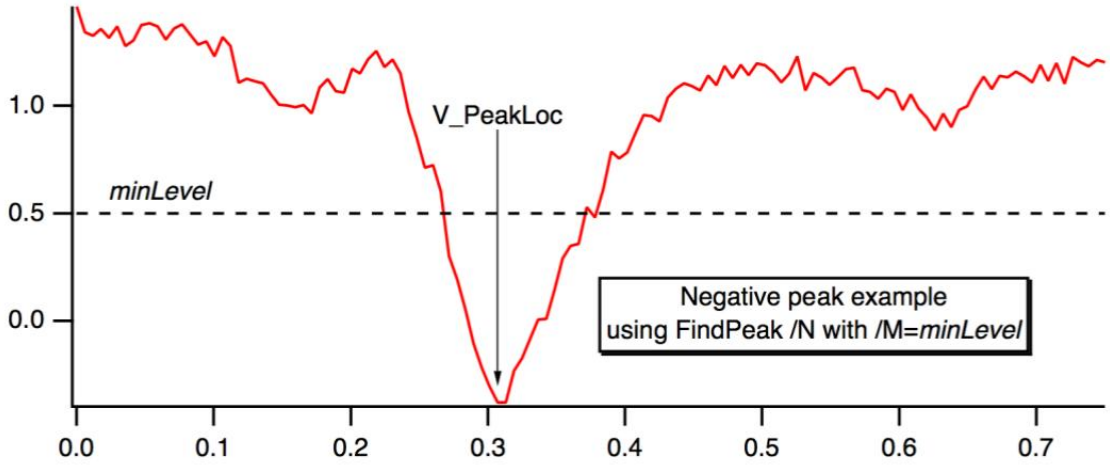




/M=minLevel フラグを使う場合、FindPeak は minLevel 未満のピークを無視します（つまり、検出されたピークの Y 値は minLevel を超える必要があります）。
minLevel の値は平滑化されたデータと比較されるため、元のデータでは十分に大きいように見えるピークでも、minLevel に非常に近い場合、検出されない可能性があります。
/N オプションも指定されている場合（最小値または「負のピーク」を検索する場合）、FindPeak は振幅が minLevel を超えるピークを無視します（つまり、検出されたピークの Y 値は minLevel 未満になります）。
負のピークの場合、ピークの最小値は、滑らかにした 1 階微分のゼロ交点にあり、その点で滑らかにした 2 階微分が正の値をとります。

次のコマンドは、負のピークを見つける例を示しています：

```
FindPeak/N/M=0.5/B=5 negPeakData          // 5 ポイントのスムージング、最大レベル = 0.5
```

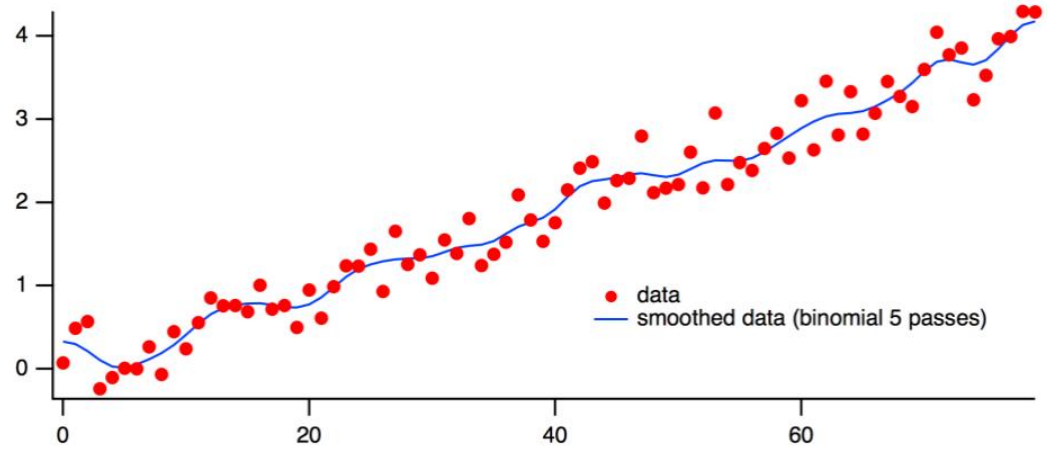


複数のピークを見つけるには、ループ内で FindPeak を呼び出すプロシージャを作成します。
ピークが見つかったら、/R を使って検索範囲を制限し、見つかったピークを除外して、再度検索します。
V_Flag が、ピークが見つからなかったことを示す場合は、ループを終了します。

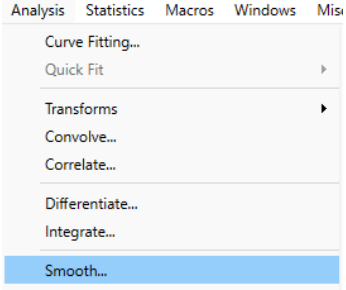
FindPeak コマンドは XY ペアに対して機能しません。
ヘルプ Converting XY Data to a Waveform を参照してください。

平滑化（スムージング）

平滑化は、データの変動を軽減するための専門的なフィルタリング操作です。
ノイズを軽減するためにも使われることがあります。



このセクションでは、Smooth、FilterFIR、および Loess コマンドによる 1 次元ウェーブフォームデータの平滑化について説明します。
FIlterIIR および Resample コマンドも参照してください。

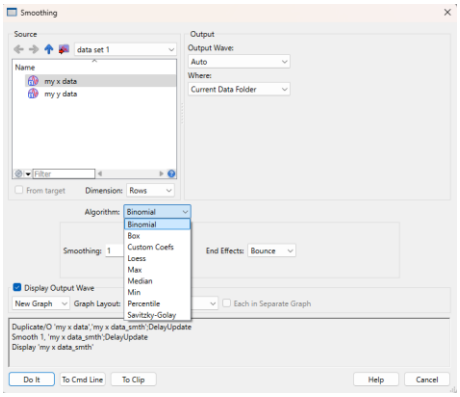


注記： XY データの平滑化は、Loess コマンドおよび Median.ipf プロシージャファイル（Median Smoothing を参照）で処理できます。

注記： 画像と 3D データの平滑化は、MatrixFilter、MatrixConvolve、および ImageFilter コマンドによって処理されます。

Igor には、いくつかの 1D 平滑化アルゴリズムが組み込まれています。
さらに、独自の平滑化係数を指定することもできます。

Analysis メニューから Smooth を選択します。
選択した平滑化アルゴリズムによっては、ダイアログで追加のパラメーターを指定する必要がある場合があります。



ビルトイン平滑化アルゴリズム

Igor には、1 次元ウェーブフォーム用の数多くのスムージングアルゴリズムが組み込まれています。
そのうちの 1 つは、XY データモデルに対応しています。

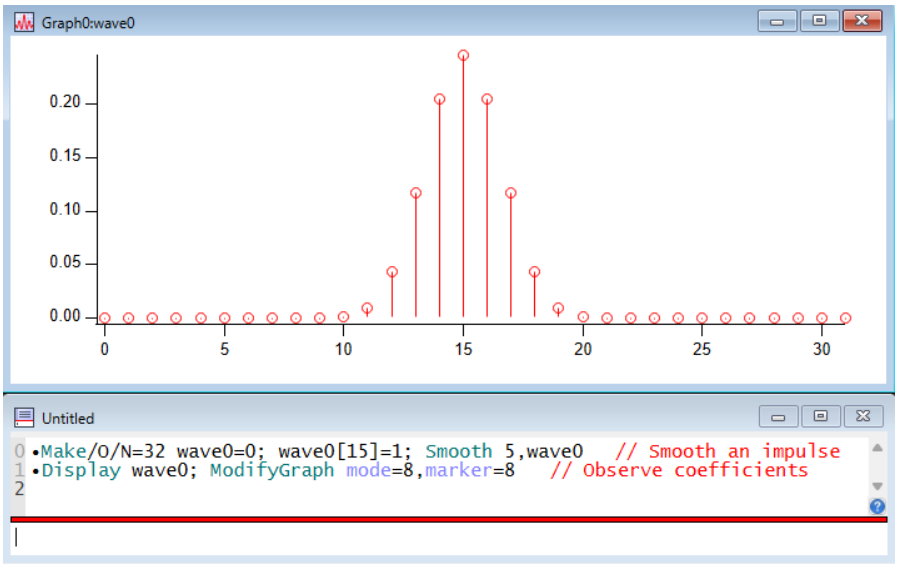
アルゴリズム	**コマンド**	**データ**
Binomial	Smooth	1D ウェーブフォーム
Savitzky-Golay	Smooth/S	1D ウェーブフォーム
Box (Average)	Smooth/B	1D ウェーブフォーム
Custom Smoothing	FilterFIR	1D ウェーブフォーム
Median	Smooth/M	1D ウェーブフォーム
Percentile, Min, Max	Smooth/M/MPCT	1D ウェーブフォーム
Loess	Loess	1D ウェーブフォーム、XY 1D ウェーブ、 偽色画像*、行列サーフェス*、多変量データ*

* Loess コマンドはこれらのデータ形式をサポートしていますが、Smooth ダイアログにはそれらを選択するためのインターフェイスは用意されていません。

最初の 4 つのアルゴリズムは、平滑化パラメーターに従って 1 組の平滑化係数を事前に計算または適用し、各データウェーブを、その係数によるウェーブの畳み込みで置き換えます。

インパルスを含むウェーブを平滑化することで、計算された係数を特定することができます。
例えば：

```
Make/O/N=32 wave0=0; wave0[15]=1; Smooth 5,wave0          // インパルスを平滑化
Display wave0; ModifyGraph mode=8,marker=8                // 係数を表示
```



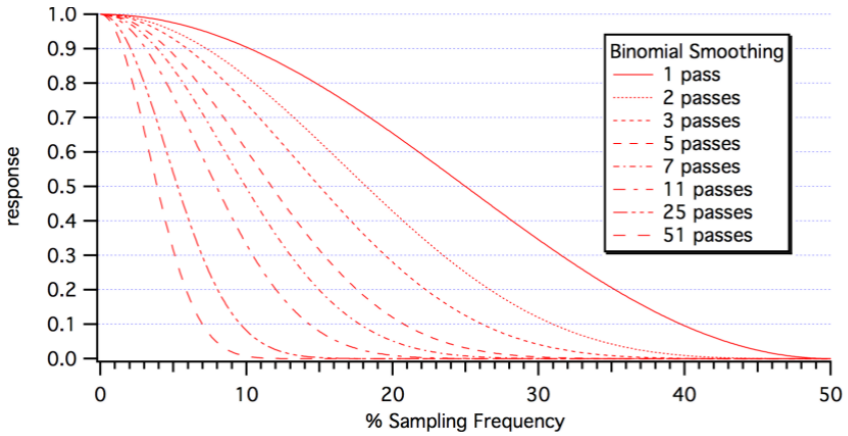
振幅出力で係数の FFT を計算して、周波数応答を表示します。
「振幅と位相の測定」を参照してください。

最後の 2 つのアルゴリズム（Smooth/M と Loess コマンド）は、固定された平滑化係数のセットを作成し、畳み込みを行う手法に基づかないため、この技術は適用できません。

Binomial（二項）平滑化

Binomial 平滑化コマンドは、Gaussian フィルターです。
このコマンドは、平滑化パラメーターと同じレベルで、パスカルの三角形から導出された正規化された係数を用いて、データを畳み込みます。
このアルゴリズムは、Marchand and Marmet (1983) の論文から導出されています。

次のグラフは、サンプリング周波数のパーセンテージとして表した、Binomial 平滑化アルゴリズムの周波数応答を示しています。
例えば、データのサンプリング周波数が 1000 Hz で、5 パスを使う場合、200 Hz (サンプリング周波数の 20%) の信号は、およそ 0.1 になります。



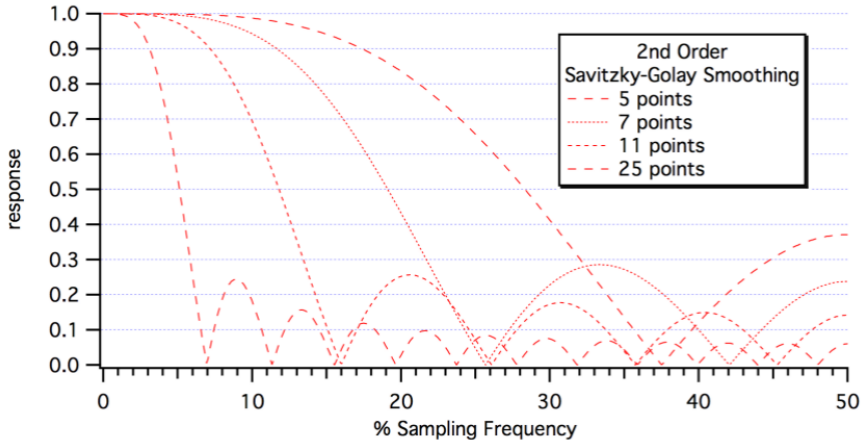
Savitzky-Golay（サヴィツキー・ゴレイ）平滑化

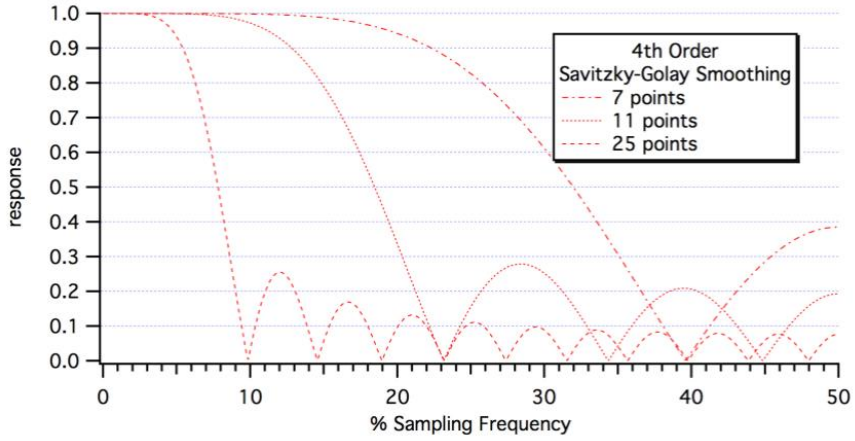
Savitzky-Golay 平滑化は、化学の分野で広く用いられている異なるセットの事前計算済み係数を使います。
これは、最小二乗多項式（Least Squares Polynomial）平滑化の一種です。
平滑化の程度は、多項式の次数と、各平滑化出力値の計算に使うポイント数の 2 つのパラメーターによってコントロールされます。
このアルゴリズムは、1964 年に A. Savitzky と M.J.E. Golay によって最初に提案されました。
その後、1972 年と 1978 年に他の研究者によって係数が修正され、Igor では修正後の係数が使われています。

最大ポイント値は 32767 です。
最小値は 2 次の場合は 5、4 次の場合は 7 です。
2 次と 3 次の係数は同じため、2 次の選択のみを記載します。
同様に、4 次と 5 次の係数は同一です。

Savitzky-Golay 平滑化は広く使われてきましたが、Marchand と Marmet が論文で説明している Binomial 平滑化には利点があります。

以下のグラフは、Savitzky-Golay アルゴリズムの 2 次平滑化と 4 次平滑化における周波数応答を示しています。
高周波数域での大きな応答は、Binomial 平滑化がしばしばより良い選択となる理由を示しています。

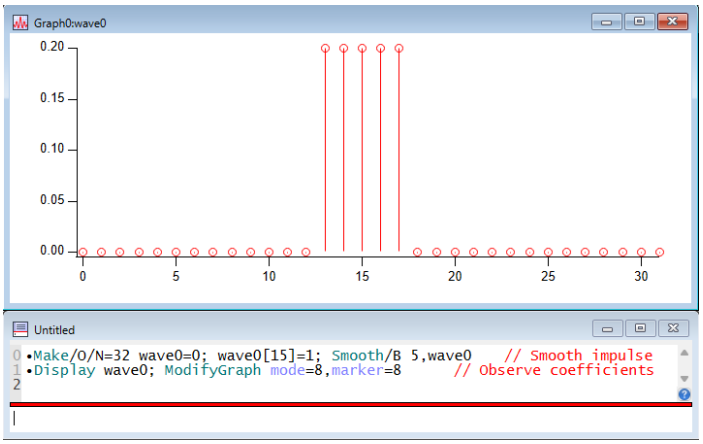




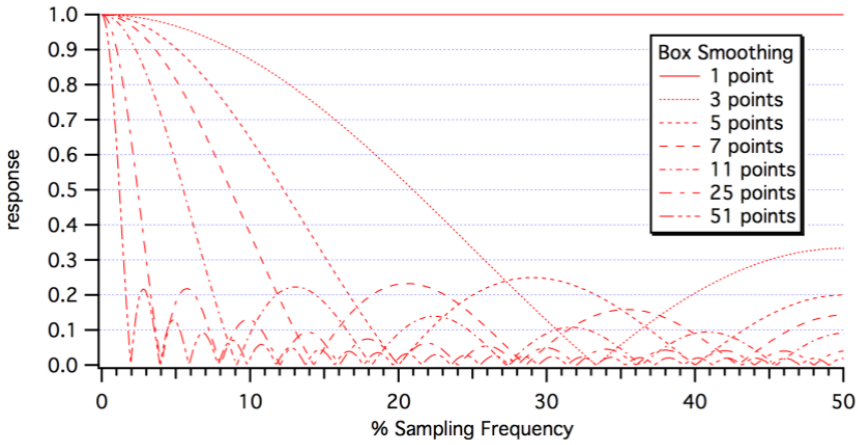
Box 平滑化

Box 平滑化は移動平均に似ていますが、平滑化対象の値の前後にある等数のデータポイントを、その値と共に平均化する点が異なります。
Points パラメーターは、平均化する値の総数です。
その値は奇数であるはずです。
なぜなら、その値には前のポイント、中央のポイント、そして後のポイントが含まれているからです。
例えば、値が 5 の場合、中心ポイントの前後 2 つの値の平均値と、中心ポイント自体の値を平均します。

```
Make/O/N=32 wave0=0; wave0[15]=1; Smooth/B 5,wave0          // インパルスを平滑化
Display wave0; ModifyGraph mode=8,marker=8                  // 係数を表示
```



次のグラフは、Box 平滑化アルゴリズムの周波数特性曲線を示しています。



Median（中央値）平滑化

Median 平滑化は、係数のセットとの畳み込みを使いません。
代わりに、各ポイントについて、そのポイントを中心とする指定された範囲の隣接する値の値の中央値を計算します。
ウェーブフォームデータ内の NaN 値は許容され、中央値の計算からは除外されます。

単純な XY データの中央値平滑化を行うには、Median.ipf プロシージャファイルを include します：

```
#include <Median>
```

また、Analysis → Packages → Median XY Smoothing メニュー項目を使います。
現在、このプロシージャファイルはデータ内の NaN を処理せず、以下で説明する方法の 1 のみを実装しています。

画像（2 次元行列）の Median 平滑化には、Median 方式を使った MatrixFilter または ImageFilter コマンドを使います。
ImageFilter は 3 次元行列データも平滑化できます。

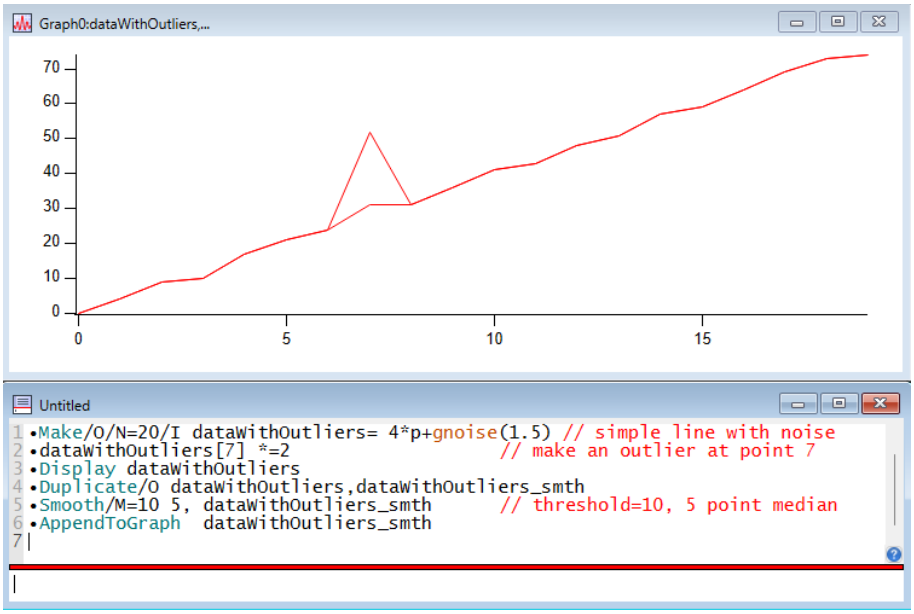
1D ウェーブフォームデータで Median 平滑化（Smooth/M）を使うには、いくつかの方法があります：

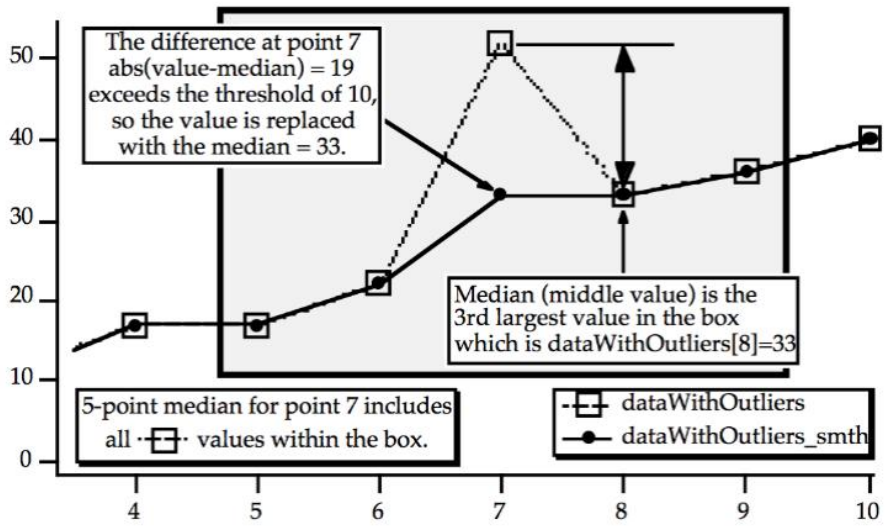
1. すべての値を、隣接する値の中央値で置き換えます。
2. 値が NaN の場合、その値を中央値で置き換えます。「中央値平滑化を使って欠損値を置き換える方法」を参照してください。
3. 各値が中央値から指定された閾値分だけ異なる場合、その値を中央値で置き換えます。
4. 値を計算された中央値に置き換える代わりに、0、NaN、+inf、または -inf を含む指定した数値に置き換えます。

Median 平滑化は、データ内の「外れ値」を置き換えるために使用できます。
外れ値とは、他のデータとは「外れた」ように見えるデータのことです。
この「外れ」の尺度の一つは、隣接する値の中央値からの過度の偏差です。
Threshold（閾値）パラメーターは、「過度の偏差」とみなされる値を定義します。

```
// 例では、結果のチェックを簡略化するために整数ウェーブを使う
Make/O/N=20/I dataWithOutliers= 4*p+gnoise(1.5)   // ノイズを持つ単純な線
dataWithOutliers[7] *=2                           // ポイント 7 に外れ値を作成
Display dataWithOutliers
```

```
Duplicate/O dataWithOutliers,dataWithOutliers_smth
Smooth/M=10 5, dataWithOutliers_smth              // threshold=10, 5 ポイントの Median

AppendToGraph  dataWithOutliers_smth
```





Percentile（パーセンタイル）、Min（最小値）、Max（最大値）平滑化

Median 平滑化は、実際には、Min（最小値）および Max（最大値）と同様、Percentile 平滑化の特殊化です。

Percentile 平滑化は、値の最小パーセンタイル % より大きい、平滑化ウィンドウ内の最小値を返します：

percentile=0: (Min) 平滑化値は、平滑化ウィンドウ内の最小値です。パーセンタイルの最小値は 0 です。

percentile=50: (Median) 平滑化値は、平滑化ウィンドウ内の値の中央値です。

percentile=100: (Max) 平滑化された値は、平滑化ウィンドウ内の最大値です。パーセンタイルの最大値は 100 です。

例えば、percentile=25、平滑化ウィンドウのポイント数が 7、1 つの入力ポイントについて、ソート後のウィンドウ内の値が以下の場合を考えます：

{0, 1, 8, 9, 10, 11, 30}

25 パーセンタイルは、順位 R を計算して求めます：

R = (percentile /100)*(num +1)

この例では、R は 2 と評価されるため、ソートされたリストの 2 番目の項目（この例では 1）が入力ポイントのパーセンタイル値となります。

パーセンタイルアルゴリズムは、0 および 100 以外のパーセンタイルの値を計算するために、補間された順位を使用します。
詳細については、Smooth コマンドを参照してください。

Loess 平滑化

Loess コマンドは、局所加重回帰平滑化を使ってデータを平滑化します。
このアルゴリズムは、「ノンパラメトリック回帰」プロシージャに分類されることもあります [Cleveland]。

回帰は、定数、線形、または 2 次関数にすることができます。
外れ値を無視する堅牢なオプションも利用できます。
さらに、データセットが小規模の場合は、Loess を使って信頼区間を生成することができます。

基本的で堅牢なアルゴリズム、例、および参考資料については、Loess コマンドのヘルプを参照してください。

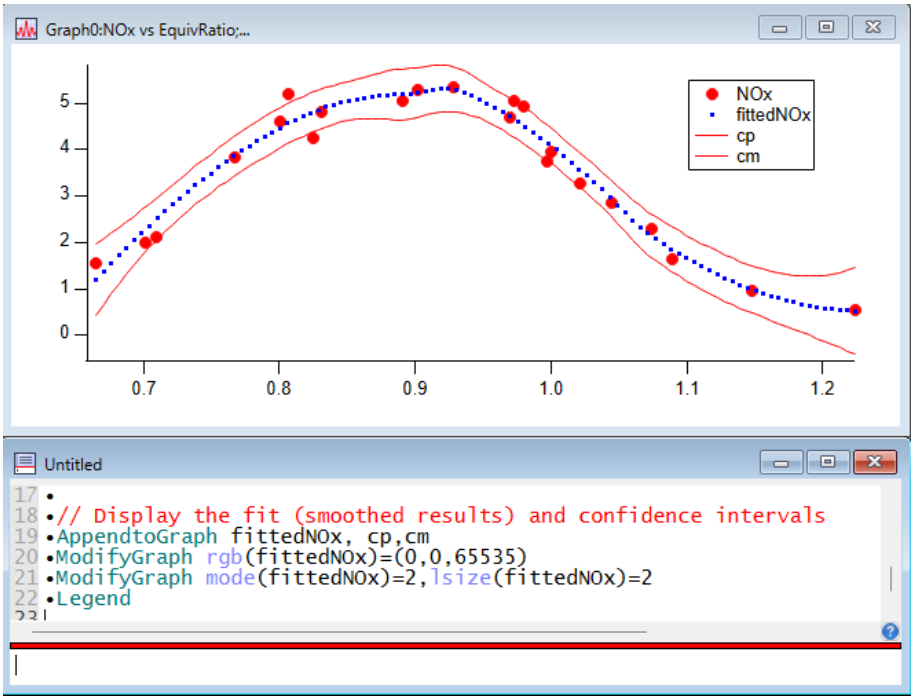
この実装は、ウェーブフォーム、ウェーブの XY ペア、偽色画像、行列表面、および多変量データ (1 つの従属データウェーブと複数の独立変数データウェーブ) で機能します。
Loess は NaN 入力値を破棄します。

ただし、Smooth ダイアログでは、ウェーブフォームと XY ペアのウェーブ (「データの XY モデル」を参照) 用のインターフェイスしか提供されておらず、信頼区間やその他のあまり一般的ではないオプション用のインターフェイスは提供されていません。

XY ペアを補間（平滑化）し、補間された 1 次元ウェーブフォーム（Y vs X スケーリング）を作成する、Loess コマンドのヘルプからの例です。

```
// 3. 99% の信頼区間出力（cp および cm）を持つウェーブフォームに補間（Y 対 X スケーリング）
//     された 1-D Y vs X ウェーブデータ
// NOx = f(EquivRatio)

// Y ウェーブ
Make/O/D NOx = {4.818, 2.849, 3.275, 4.691, 4.255, 5.064, 2.118, 4.602, 2.286, 0.97, 3.965,
5.344, 3.834, 1.99, 5.199, 5.283, 3.752, 0.537, 1.64, 5.055, 4.937, 1.561};

// X ウェーブ （X ウェーブはソートされていないことに注意）
Make/O/D EquivRatio = {0.831, 1.045, 1.021, 0.97, 0.825, 0.891, 0.71, 0.801,  1.074, 1.148, 1,
0.928, 0.767, 0.701, 0.807, 0.902, 0.997, 1.224, 1.089, 0.973, 0.98, 0.665};

// 入力データをグラフ出力
Display NOx vs EquivRatio; ModifyGraph mode=3,marker=19

// X 範囲にわたって高密度ウェーブフォームに補間
Make/O/D/N=100 fittedNOx
WaveStats/Q EquivRatio
SetScale/I x, V_Min, V_max, "", fittedNOx
Loess/CONF={0.99, cp, cm}/DEST=fittedNOx/DFCT/SMTH=(2/3) srcWave=NOx,factors={EquivRatio}
```

```
// フィット（滑らかな結果）と信頼区間を表示する
AppendtoGraph fittedNOx, cp,cm
ModifyGraph rgb(fittedNOx)=(0,0,65535)
ModifyGraph mode(fittedNOx)=2,lsize(fittedNOx)=2
Legend
```



注記： Loess はメモリを大量に消費します。特に信頼区間を生成する時には注意が必要です。信頼区間を使う場合、Loess コマンドのヘルプにある「メモリの詳細」セクションを確認してください。

カスタム平滑化係数

カスタム係数アルゴリズムを選択することで、独自の平滑化係数セットを使ってデータを平滑化できます。
このオプションは、別のプログラムまたは Igor Filter Design Laboratory (IFDL) で作成されたローパスフィルター（平滑化）係数がある場合に使います。

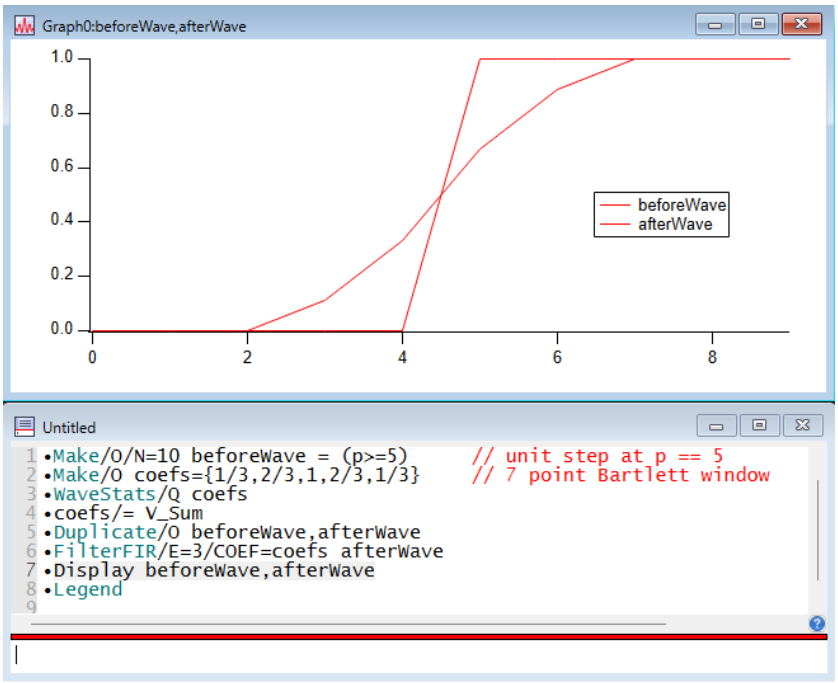
表示されるポップアップメニューから、係数を含むウェーブを選択します。
Igor は、FilterFIR コマンドを使って、これらの係数を入力ウェーブと畳み込みます。
短いウェーブを、より長いウェーブと畳み込む場合は、FilterFIR を使ってください。
2 つのウェーブを畳み込む場合、ポイント数が同じであれば、Convolve コマンドを使うと高速に処理できます。

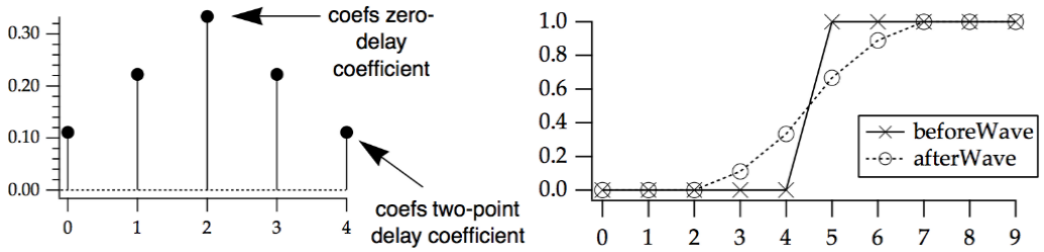
係数ウェーブのすべての値が使われます。
FilterFIR は、係数ウェーブの中間ポイントが「遅延=0」ポイントに対応すると仮定します。
これは通常、係数ウェーブに、奇数個のポイントを持つフィルタの双方向インパルス応答が含まれている場合に発生します。
（ポイントの数が偶数の係数ウェーブの場合、「中央」のポイントは numpnts(coefs)/2-1 になりますが、これにより、通常望ましくない遅延が平滑化されたデータに発生します）

次の例では、coefs ウェーブは、単純な 7 ポイントの Bartlett (三角) ウィンドウ (最初の Bartlett ウィンドウの値と最後の Bartlett ウィンドウの値は 0 なので省略) によってデータを平滑化しています。

```
// この例は、7 ポイント Bartlett ウィンドウによって平滑化された単位ステップ信号を示しています
Make/O/N=10 beforeWave = (p>=5)          // p==5 での単位ステップ
Make/O coefs={1/3,2/3,1,2/3,1/3}                  // 7 point Bartlett window
```

```
WaveStats/Q coefs
coefs/= V_Sum

Duplicate/O beforeWave,afterWave
FilterFIR/E=3/COEF=coefs afterWave

Display beforeWave,afterWave
Legend
```

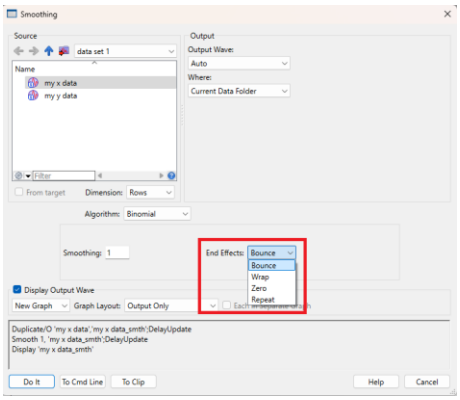




終端効果

最初の 4 つの平滑化アルゴリズムは、指定されたポイントの隣にあるポイントを使って、そのポイントの出力値を計算します。
各アルゴリズムは、滑らかにするポイントの前後にある隣接するポイントを同じ数だけ組み合わせています。
ウェーブの開始ポイントまたは終了ポイントでは、一部のポイントには十分な隣接ポイントがないため、隣接値の生成方法を実装する必要があります。

Smoothing ダイアログの End Effect ポップアップメニューで、Igor がこれらの値をどのように作成するかを選択します。
以下の説明では、i は小さな正の整数であり、wave[n] は平滑化されるウェーブの最後の値です。



Bounce 法は、欠落しているウェーブ [-i] の値の代わりにウェーブ [i] を使い、欠落しているウェーブ [n+i] の値の代わりにウェーブ [n-i] を使います。
これは、データがウェーブの開始点と終了点について対称であると想定する場合に最も効果的です。
終了効果の方法を指定しない場合、Bounce が使われます。

Wrap 法は、欠落しているウェーブ [-i] の値の代わりにウェーブ [n-i] を使い、その逆も同様です。
これは、ウェーブが無限に繰り返されると仮定する場合に最も効果的です。

Zero 法では、欠落値には 0 を使います。
この方式は、ウェーブが 0 で始まり 0 で終わる場合に最適です。

Repeat 法は、欠落している wave[-i] 値の代わりに wave[0] を使い、欠落している wave[n+i] 値の代わりに wave[n] を使います。
これは、1 つのイベントを表すデータに最適です。

迷ったときは、Repeat を使ってください。

デジタルフィルタリング

デジタルフィルターは、ウェーブフォームに含まれる周波数を強調または弱めるために使われます。
例えば、ローパスフィルターは低周波数を保持し、高周波数を排除します。

入力ウェーブフォームにフィルターを適用すると、「response」出力ウェーブフォームが生成されます。

Igor は、有限インパルス応答 (FIR) および無限インパルス応答 (IIR) デジタルフィルターの設計と適用が可能です。

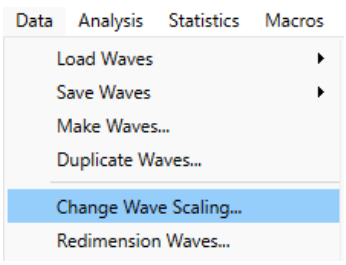
Igor には、他の形式のデジタルフィルタリングも存在します。
Savitzky-Golay、Loess、中央値、移動平均平滑化など、さまざまな平滑化操作が挙げられます。

Convolve コマンドを直接使うことは、デジタルフィルタリングを行う別の方法ですが、これには、以下で説明する「フィルター設計と適用ダイアログ」を使う場合よりも、より高度な知識が必要です。

Igor Filter Design Laboratory (IFDL) パッケージも、デジタルフィルターの設計および適用に使用できます。

サンプリング周波数と設計周波数帯域

デジタルフィルタリングでは、データの XY モデルは使われません。
ウェーブフォームモデルを使い、SetScale または Change Wave Scaling ダイアログを使ってサンプリング周波数を設定してください。



例えば、44.1KHz（コンパクトディスクの音楽サンプルレート）でサンプリングされたウェーブフォームの場合、X スケーリングは次のようなコマンドで設定する必要があります。

```
SetScale/P x, 0, 1/44100, "s", musicWave
```

通常、フィルターは、周波数の範囲を定義する「周波数帯域」と、その帯域における望ましい応答振幅（ゲイン）および位相を指定して設計されます。

可能な周波数の範囲は、信号のサンプリング周波数の 0 からその半分の範囲までであり、これを「ナイキスト（Nyquist）周波数」と呼びます。

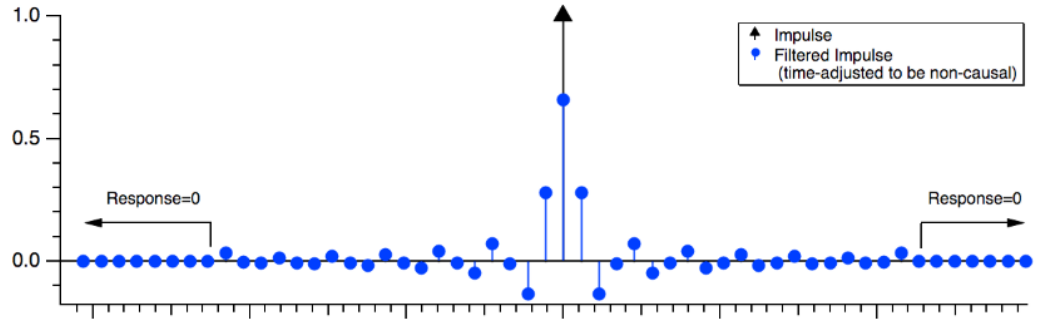
musicWave の例では、ナイキスト周波数は 22,050 Hz であるため、そのウェーブフォーム用のフィルター設計では、22,050 Hz 以下の周波数帯域を定義します。

フィルター設計出力

フィルター設計の結果、フィルタリングの実装に使われる一連のフィルター「係数」が生成されます。
係数の値と形式は、フィルターの設計タイプ、帯域数、帯域周波数、およびフィルターの応答を定義するその他のパラメーターによって異なります。
FIR 設計と IIR 設計のフォーマットは、かなり異なります。

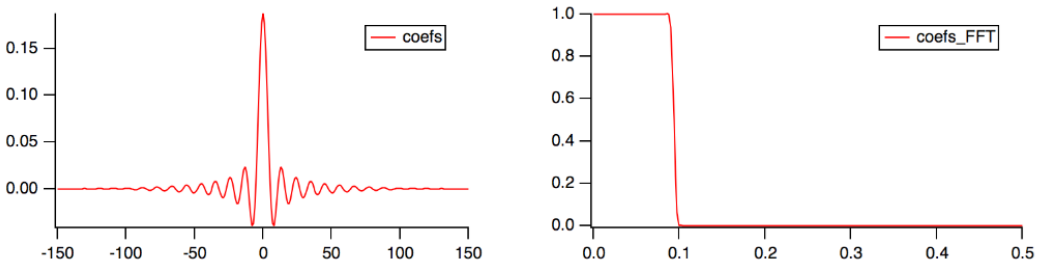
FIR フィルター

有限インパルス応答（Finite Impulse Response: FIR）とは、フィルターがインパルス（スパイク）に対する時間領域応答が、有限の時間後にゼロになることを意味します。



FIR フィルターは、振幅が変化する有限の長さの等間隔のインパルス時系列であり、入力信号と畳み込みを行って、フィルタリングされた出力信号を生成します。

インパルス応答の振幅は「重み係数」または「係数」と呼ばれます。
これらは、フィルターが単位インパルスに応答したときの応答とまったく同じです。
FIR フィルターは、係数の FFT を計算するだけで周波数応答を観察することができます。
係数の X 方向のスケールを、適用するデータのサンプリング周波数に一致するように設定すると、FFT 結果の周波数範囲はデータのナイキスト周波数にスケールされます。
デフォルトの X スケーリングの場合、周波数範囲は 0 から 0.5 Hz になります：



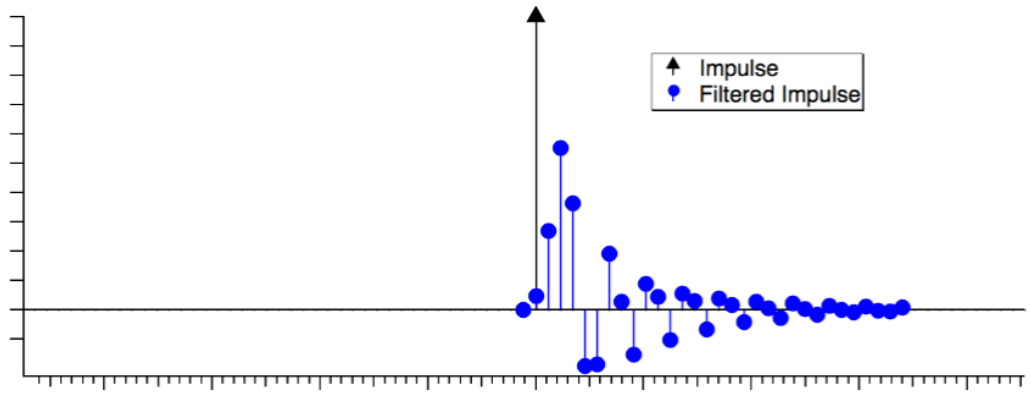
FIR フィルターは、完全に線形位相（すべての周波数で一定の遅延）で評価されていますが、同様の周波数応答を実現するには、IIR フィルターよりもはるかに多くの係数が必要となります。

その結果、FIR フィルターを電子的にデジタル化すると、通常、対応する IIR フィルターよりもコストが高くなります。

FIR 係数を FilterFIR コマンドに入力ウェーブとともに供給すると、フィルタリングされた出力ウェーブが計算されます。

IIR フィルター

無限インパルス応答（Infinite Impulse Response: IIR）フィルターの応答は、インダクターやコンデンサーを使うアナログ電子フィルターと同様、無期限に続きます。



IIR フィルターは、デジタル実装トポロジーに応じて値および使用方法が異なる、係数または重み a0、a1、a2… および b0、b1、b2… のセットです。
FIR フィルターとは異なり、これらの係数は、単位インパルスに対するフィルターの応答と同じではありません。
詳細については、「IFDL IIR フィルター設計」のトピックを参照してください。

IIR フィルターは、ごくわずかな係数で非常に洗練された周波数応答を実現できます。
欠点は、非線形位相、有限精度演算で実現した場合の数値不安定性（発振）の可能性、および間接的な設計手法（従来のアナログフィルター手法の周波数変換）であることです。

Igor は 2 つの IIR 実装を使っています：

- Direct Form I (DF I)
- Cascaded Bi-Quad Direct Form II (DF II)

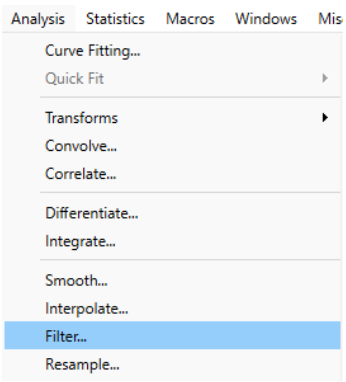
IIR 係数は 3 つの形式で表されます：

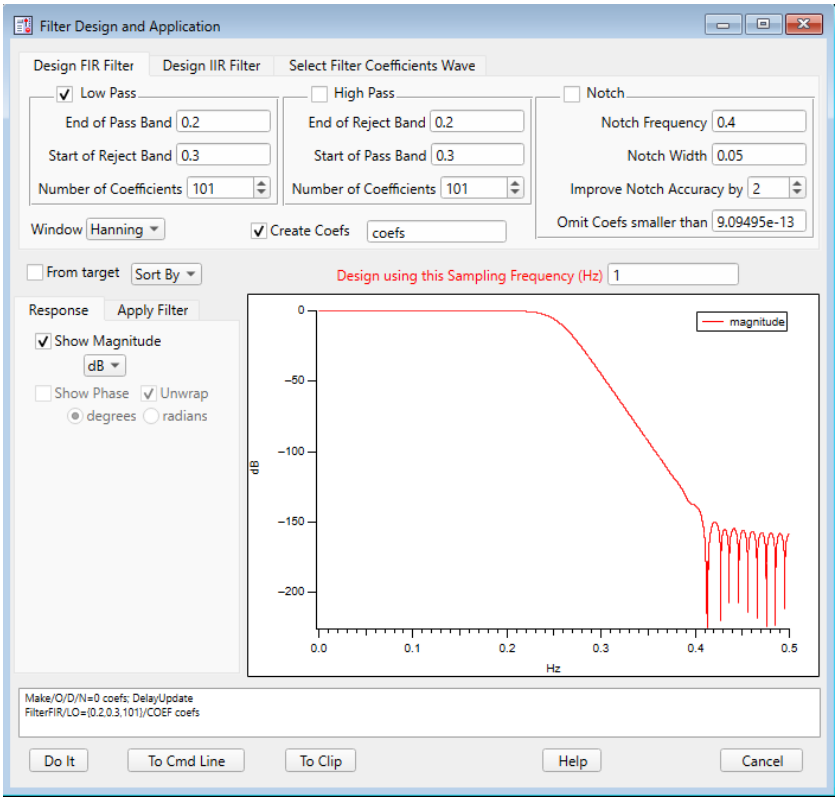
- DF I
- DF II
- "zeros and poles" form（IFDL の「IIR Analog Prototype Design Graph」のトピックで議論されています）

IIR 係数を FilterIIR 演算に、入力ウェーブフォームとともに供給して、フィルタリングされた出力ウェーブフォームを計算します。
IIR 設計係数のフォーマットは、実装によって異なります。
これは、同じフィルター設計の Direct Form I、Cascaded Bi-Quad Direct Form II、および Pole-zero の係数を示すテーブルで確認できます。

Filter Design and Application ダイアログ

Filter Design and Application ダイアログは、デジタルフィルターを設計および適用するためのシンプルなユーザーインターフェイスを提供します。
メニュー Analysis → Filter を選択して表示します。





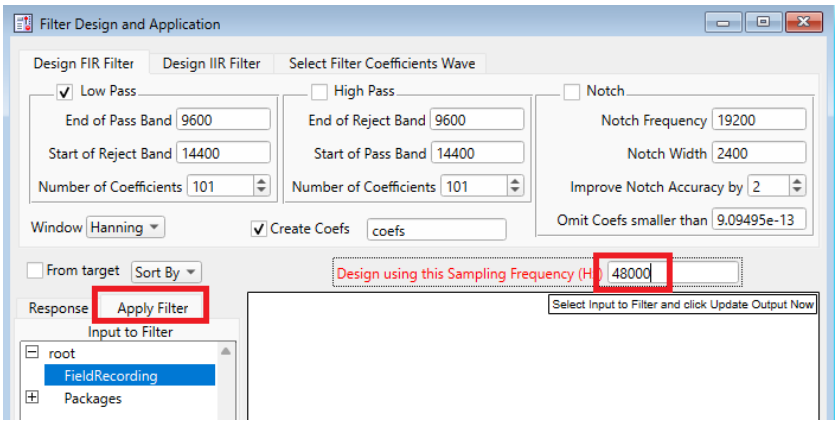
このダイアログでは、Igor Filter Design Laboratory (IFDL) フィルターの一部を設計することができます。よりシンプルで、ほとんどの用途に十分なフィルターが用意されています。

最初は、[Design FIR Filter] タブが表示され、単純なローパスフィルターが事前に選択されています。「Design using this Sampling Frequency (Hz)」は 1 に設定されており、デフォルトの設計サンプリング周波数は 1 Hz であるため、表示されている周波数は 0 から 0.5 Hz のデフォルト範囲です。

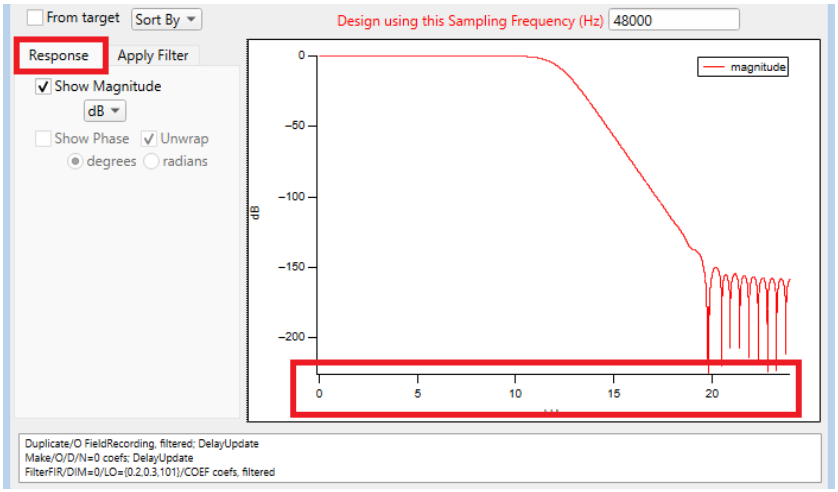
フィルターの設計を開始するには、次のいずれかを実行します：

- サンプリング周波数を手動で入力
- Apply Filter をクリックし、上記で正しく設定したサンプリング周波数を持つ、フィルターをかけるウェーブを選択

このウェーブは 48000 Hz でサンプリングされているとします：

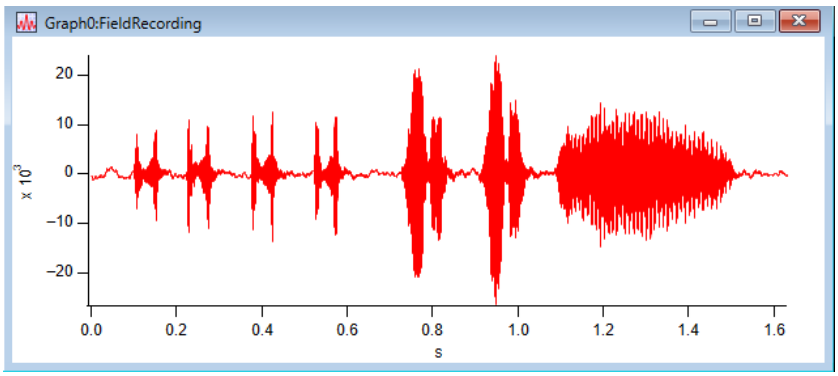


[Response] タブに戻り、入力したサンプリング周波数を使ってデフォルトのローパスフィルターを表示します。
周波数範囲は 0～24000 Hz になりました。

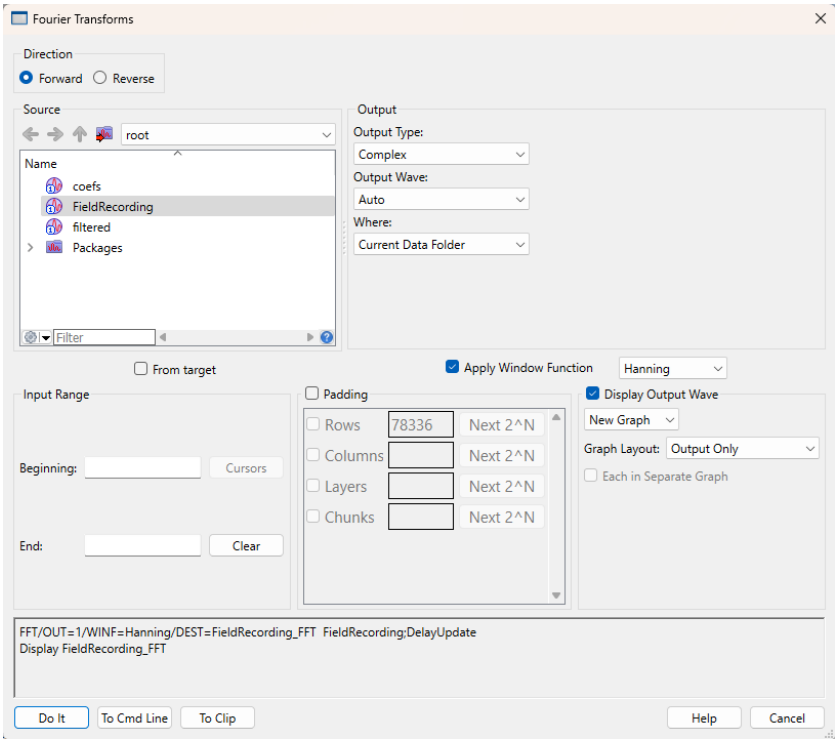


1 つのローパス帯域、1 つのハイパス帯域、および 1 つのノッチを任意に組み合わせて、サンプリングされたデータウェーブの周波数成分を通過または除去することができます。
ローパス帯域とハイパス帯域の両方を使うことで、帯域パスフィルターと帯域ストップフィルターを作成することができます。

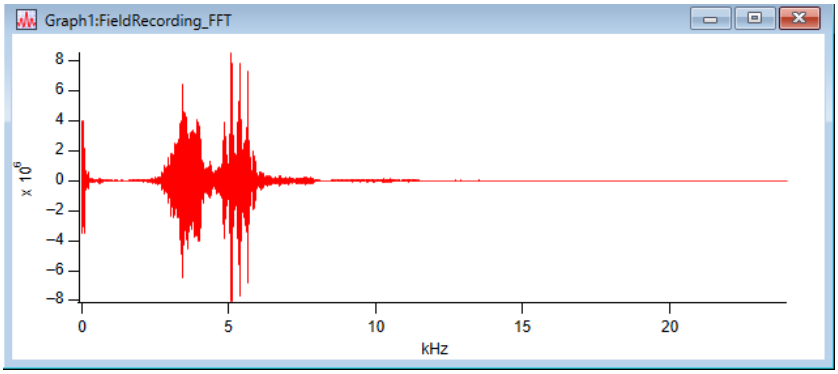
fieldRecording にフィルターを適用する前に、元のウェーブフォームをグラフ化してみます。



フィルタリングする前に、入力ウェーブの周波数成分を知っておくと便利です。
メニュー Analysis → Transforms → Fourier Transforms を選択して、Fourier Transforms ダイアログを使ってください。



この信号には、2 つの興味深い周波数帯があります：0 から 4.5kHz と、4.5 から約 10kHz です。



説明のために、各帯域を分離する 2 つのフィルター、ローパスフィルターとハイパスフィルターを設計してみます。

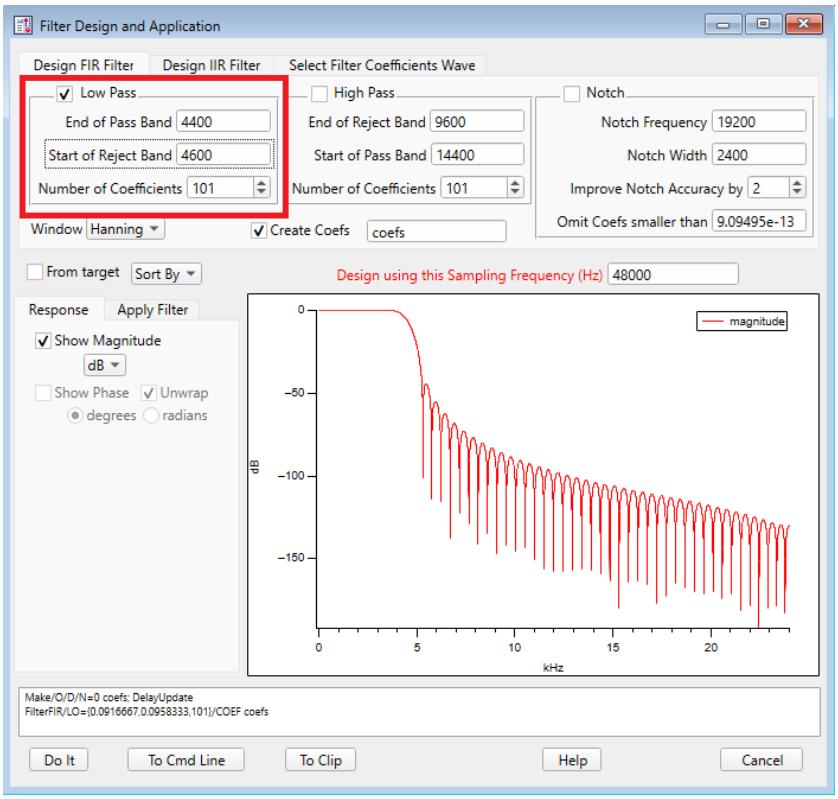
例：ローパス FIR フィルター

ローパスフィルターは、信号周波数を 4.5KHz 以下に抑え、それ以上の周波数を排除するように設計することができます。

無限に鋭い周波数境界は現実的ではありません（無限の係数が必要になるため）。
そのため、通過帯域から遮断帯域への移行が起きる 2 つの周波数を指定します。
この移行帯域が狭くなるほど、高周波数帯域を有効に除去するために必要な係数の数が増加します。

200 Hz の移行帯域幅（4400 Hz から 4600 Hz）を選択し、適切な数の係数（デフォルトの 101）を使って周波数特性を確認します：

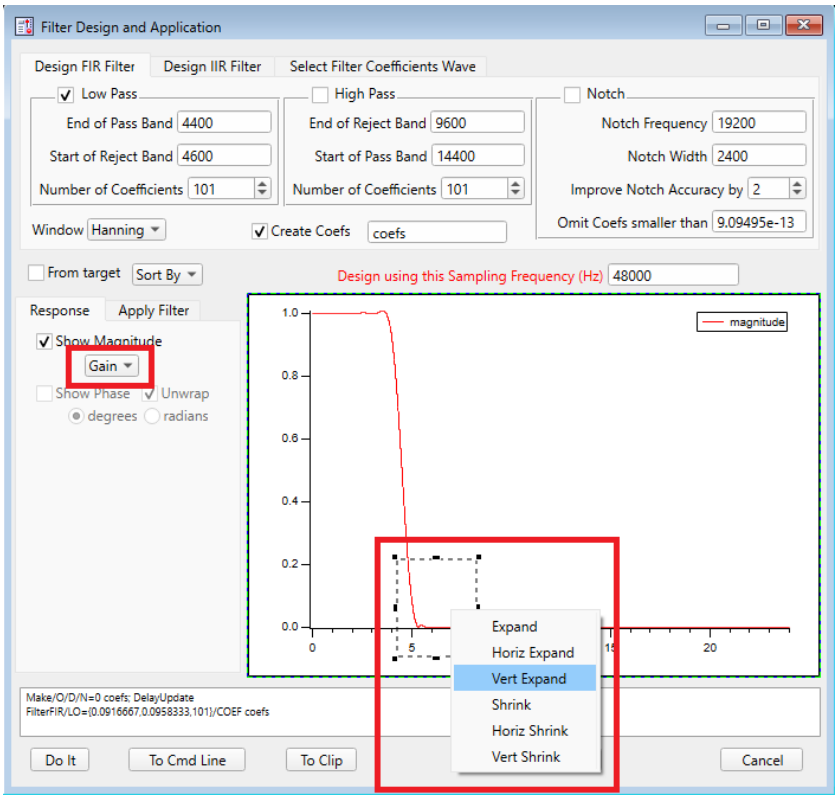
Low Pass Filter：チェック
End of Pass Band：4400
Start of Reject Band：4600



dB とは何ですか？

dB は「デシベル」の略で、ある値と別の値の比率を表す対数単位です。
フィルタリングでは、特定の周波数における出力振幅と入力振幅の比率を表します。
0 dB は比率に変化がないことを意味し、通過帯域のフィルターの応答です。

-100dB とは、10^{-5} または 0.00001 の比率を意味します。
dB から Gain（ゲイン）に表示を切り替えて、垂直方向の範囲を大幅に拡大（範囲を選択して、右クリックでメニューを表示）すると、この関係が確認できます：

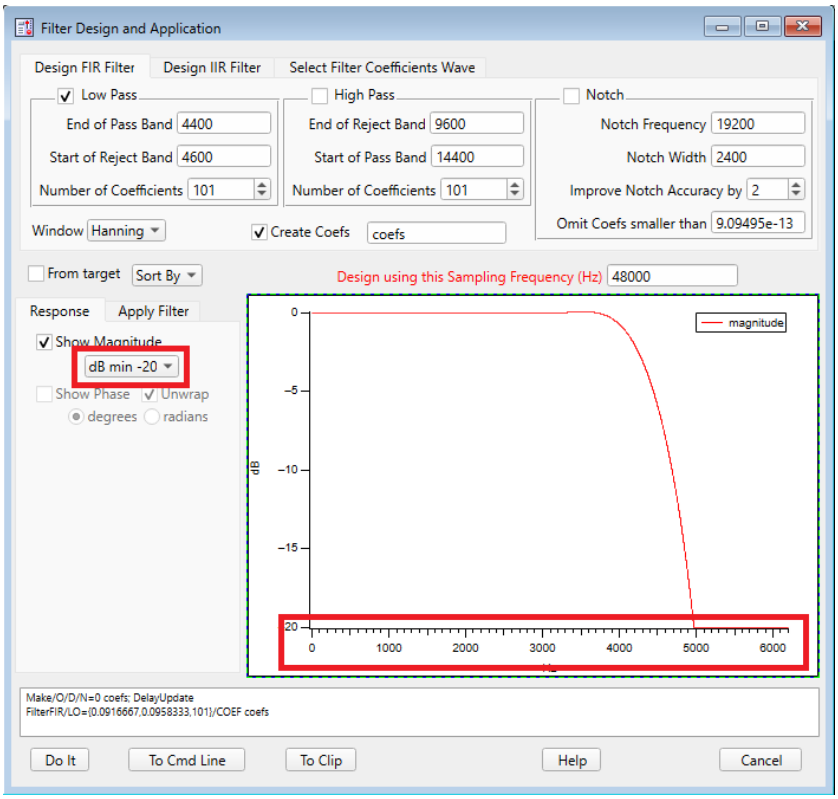


FIR 帯域周波数の選択

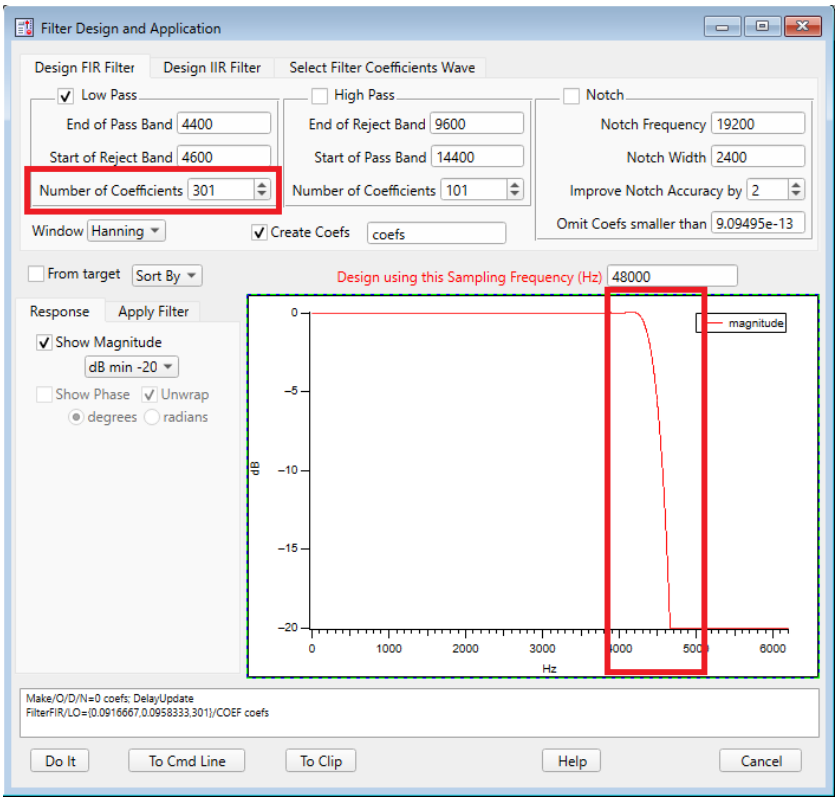
通過帯域の終わり付近の周波数の振幅を低下させないようにするには、通過帯域の終わり周波数を増加させます。より多くの係数が必要になる場合があります。

FIR フィルター設計における係数の選択

係数の数を増やすことで、通過帯域から遮断帯域への移行を急峻にすることができます。
（グラフを右クリックし、Autoscale Axes を選択して元に戻し、X 軸をダブルクリックして X 軸の範囲を 6200Hz にします）



係数の数を 101 から 301 に増やします。

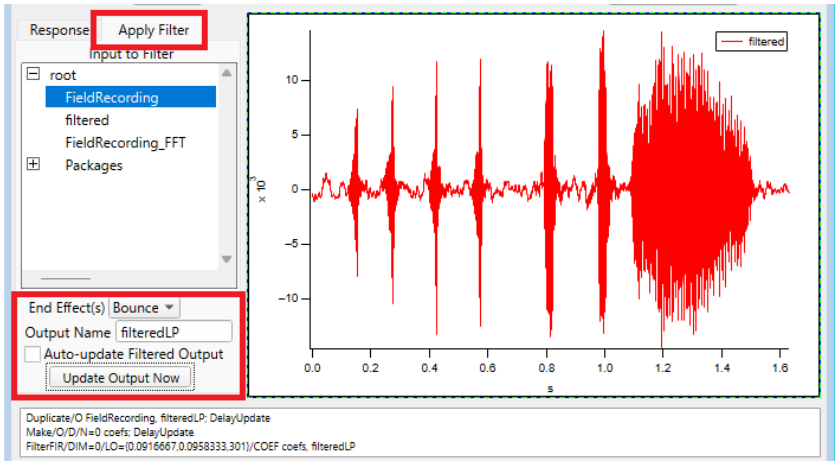


FIR フィルターでは、フィルタリングされた出力ウェーブフォームがサンプル半分分遅延しないように、係数の数は奇数を使ってください。

データに FIR フィルターを適用

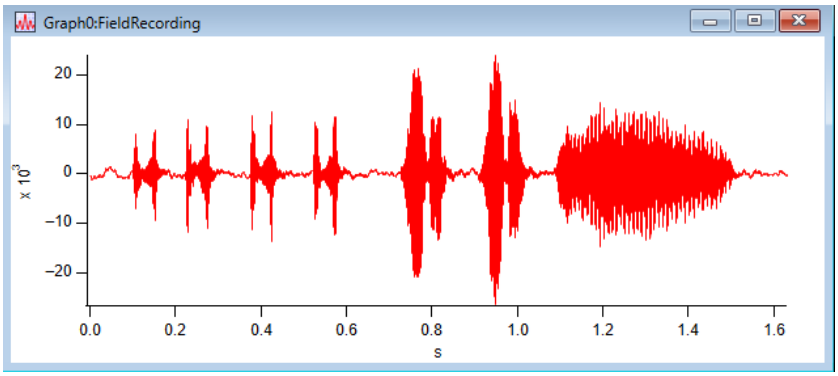
Apply Filter（フィルターを適用）タブをクリックすると、設計したフィルターをウェーブフォームに適用した結果を確認できます。

Input to Filter（フィルター入力）リストボックスで入力ウェーブを選択し、Auto-update Filtered Output （フィルター出力の自動更新）チェックボックスまたは Update Output Now（出力を今すぐ更新）ボタンをクリックします。
これにより、フィルター処理の結果のプレビューが更新されます。

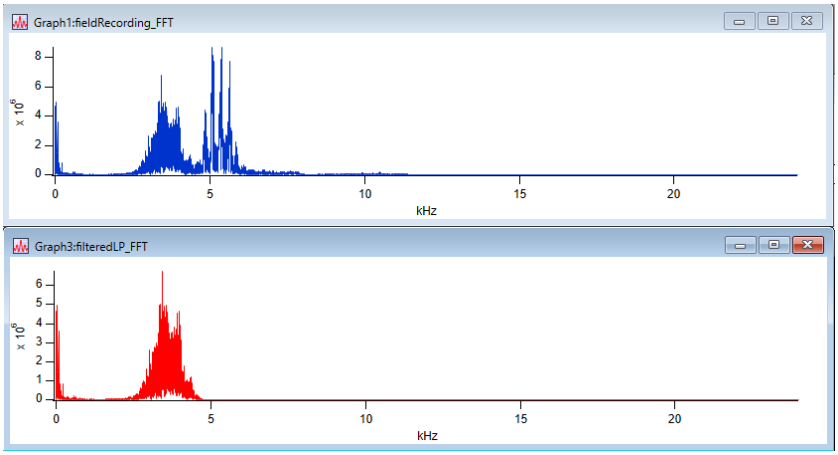


Do It（実行）をクリックして、現在のデータフォルダーに最終的な出力ウェーブを作成します。
最終的な出力ウェーブの名前は、Output Name（出力名）フィールドで設定できます。
ここでは「filteredLP」を使いました。

比較のため、以下にフィルタリングされていない fieldRecording を示します：



ダイアログのプレビューでは、フィルター処理された出力から高周波成分が除去されていることがわかります。
フィルター処理された LP の FFT により、この変化が確認できます。
Analyze メニューから Fourier Transform を選択して、Fourier Transforms ダイアログを開きます。
ウェーブのリストから fielteredLP を選択し、ポイント数が偶数（ここでは 1 を足して 302）になるように設定します。
Do It をクリックすると、fielteredLP_FFT というウェーブが作成されます。

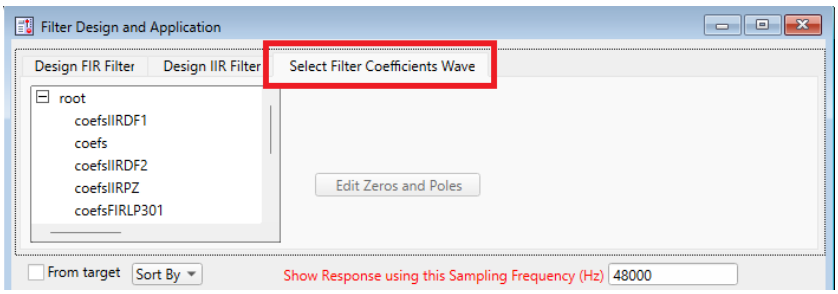


他のデータに FIR フィルターを適用

設計の出力係数のコピーを保存しておけば、フィルターを再利用することができます。
例えば：

```
Duplicate/O coefs, savedFIRfilter          // フィルターの設計のコピーを保持
```

保存した FIR フィルターは、FilterFIR コマンドを直接使うか、Filter ダイアログの Select Filter Coefficients Wave タブを使って、他のデータに適用することができます。



FilterFIR コマンドを使う場合：

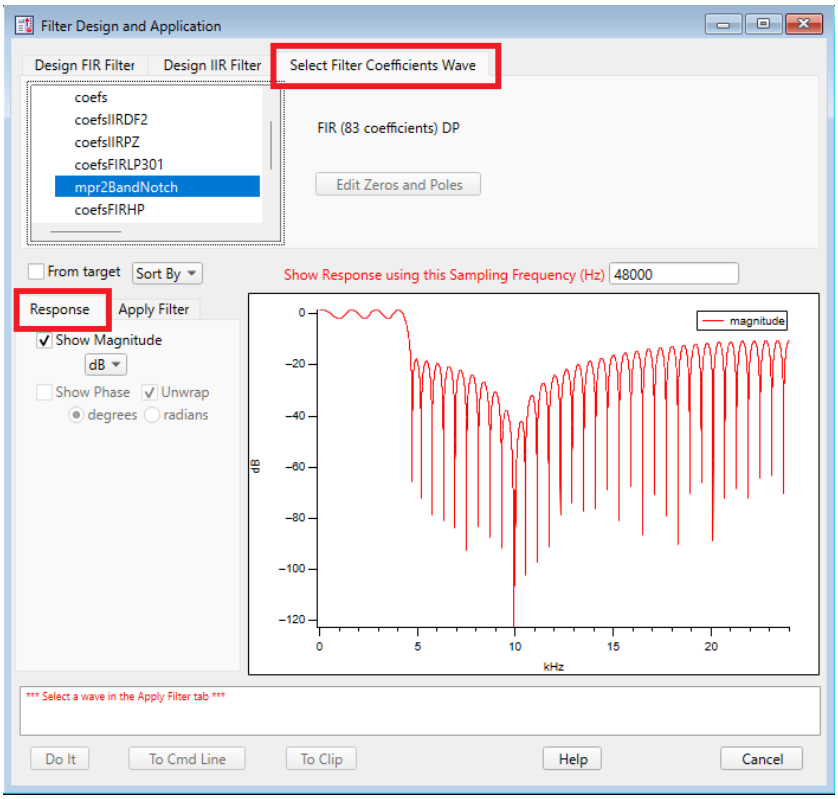
```
Duplicate/O otherData, otherDataFiltered
FilterFIR /DIM=0 /COEF=savedFIRfilter otherDataFiltered
```

フィルター係数ウェーブの選択

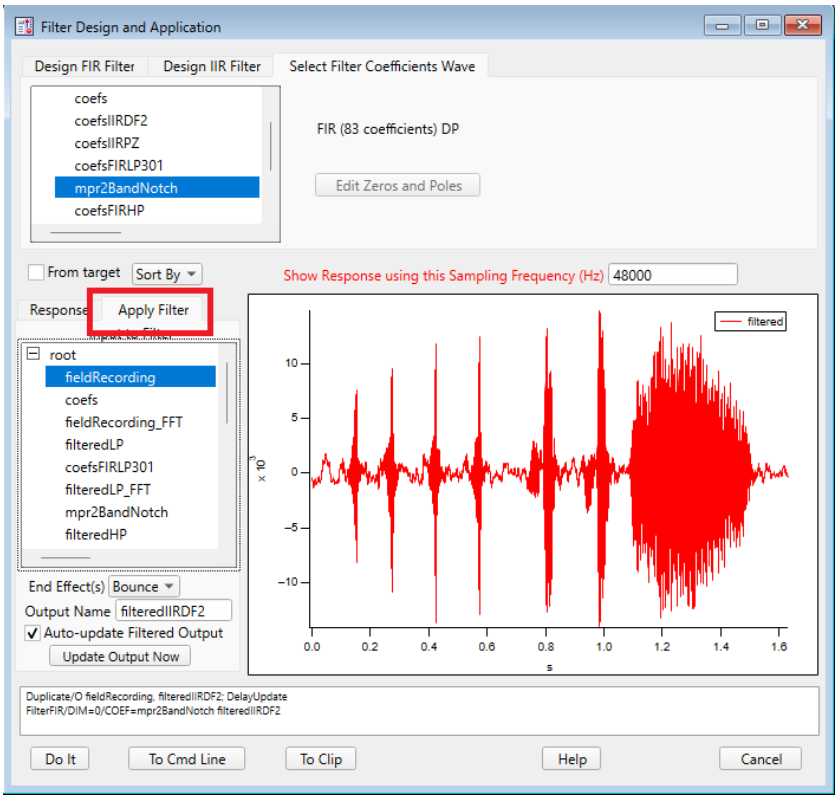
このセクションでは、Filter Design and Application ダイアログを使って、保存した FIR または IIR フィルターを他のデータに適用する方法を示します。

まず、ダイアログの Select Filter Coefficients Wave タブで、保存したフィルター設計ウェーブを選択します。

Response タブを選択すると、フィルターの応答を確認できます。



次に、下の Apply Filter タブから、フィルターをかけるウェーブを選択します。



例：ハイパス FIR フィルター

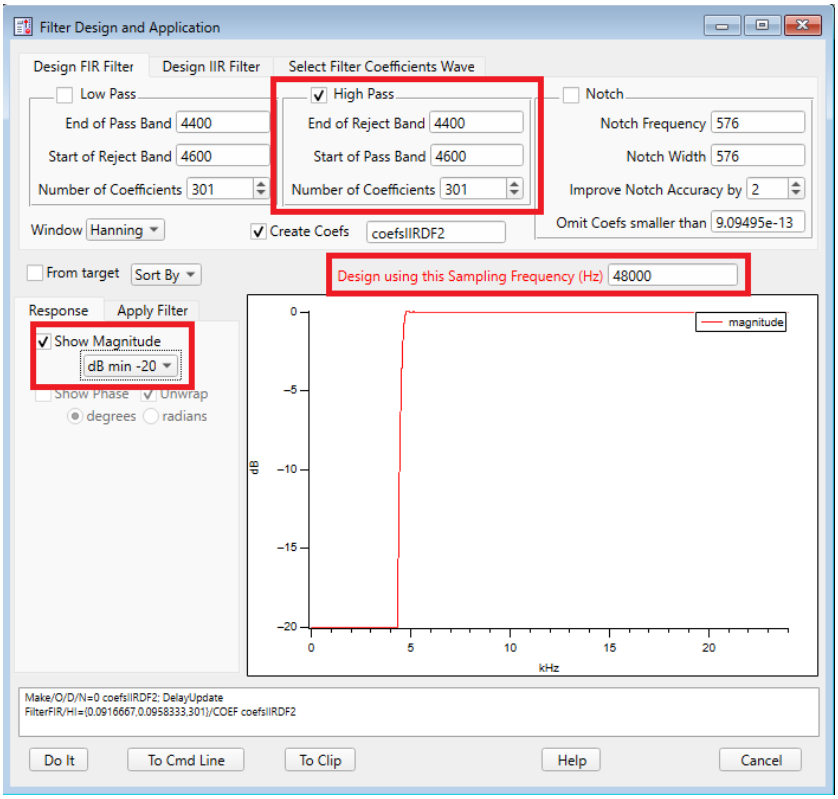
上記のローパスフィルターによって除去された信号成分のみを残すハイパスフィルターを設計します。

メニュー Analysis → Filter を選択し、Design using this Sampling Frequency (Hz) を 48000 に設定します。

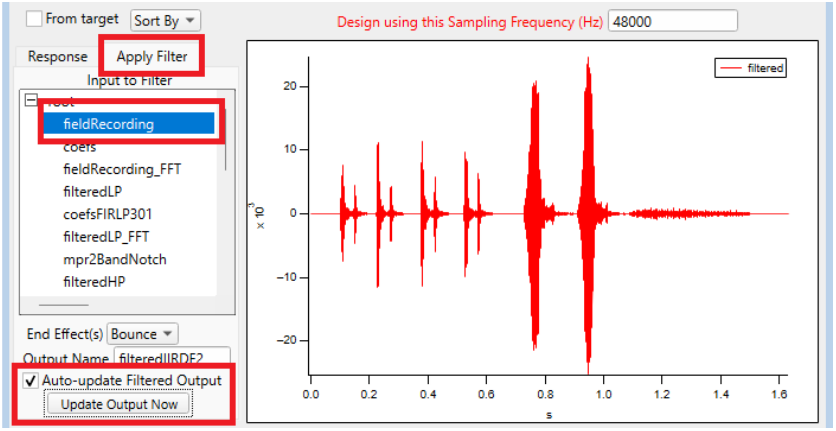
Low Pass をオフにし、High Pass をオンにし、Notch はオフのままにします。

ハイパス FIR 帯域周波数の選択

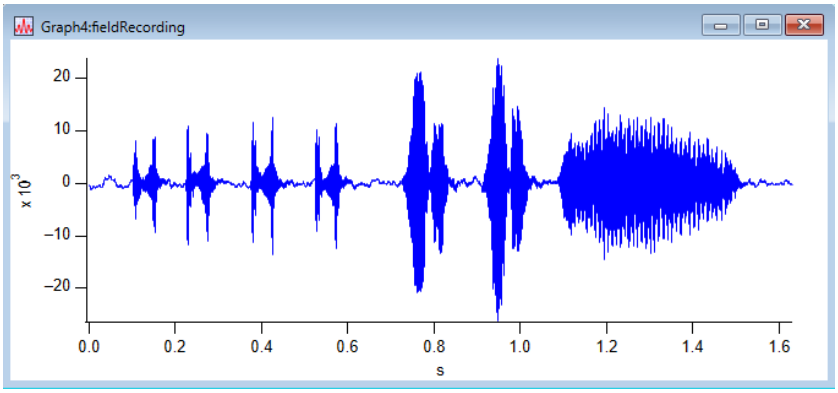
End of Reject Band を 4400 に設定し、Start of Pass Band を 4600 に設定して、ローパスフィルターの例と同じ移行帯域を定義します。
Number of Coefficients を 301 にして、低周波数域の遮断と高周波数域の通過の間に急峻な移行を実現します。
db min -20 を選択すると次のようになります。



Apply Filter タブをクリックして、設計したフィルターをウェーブフォームに適用した結果を確認します。
fieldRecording フィールドを選択し、Update Output Now をクリックします。



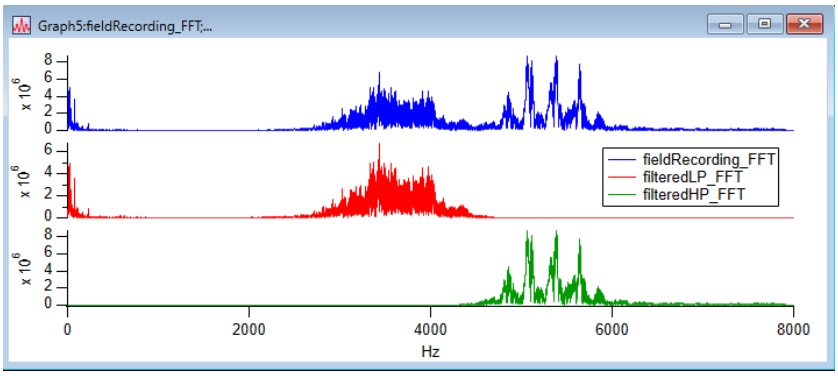
比較のため、フィルタリングされていない fieldRecording を示します：



ダイアログのプレビューでは、フィルター処理された出力から低周波の要素が除去されていることがわかります。

Do It をクリックして、fieldRecording ウェーブフォームをハイパスフィルタリングした結果、filteredHP を作成します。

filteredHP の結果の FFT で、変更を確認します：
（fieldRecording_FFT, fielteredLP_FFT, filteredHP_FFT をプロットした結果は次のようになります）



フィルタリングの結果を評価するもう 1 つの方法は、元のウェーブフォームとフィルタリングされたウェーブフォームに対して PlaySound コマンドを使うことです。

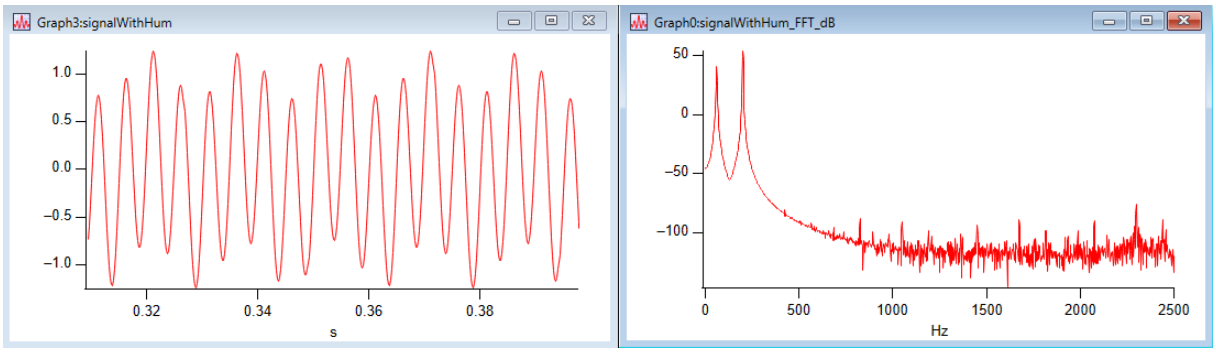
```
PlaySound fieldRecording
PlaySound filteredLP
PlaySound filteredHP
```

例：Notch FIR フィルター

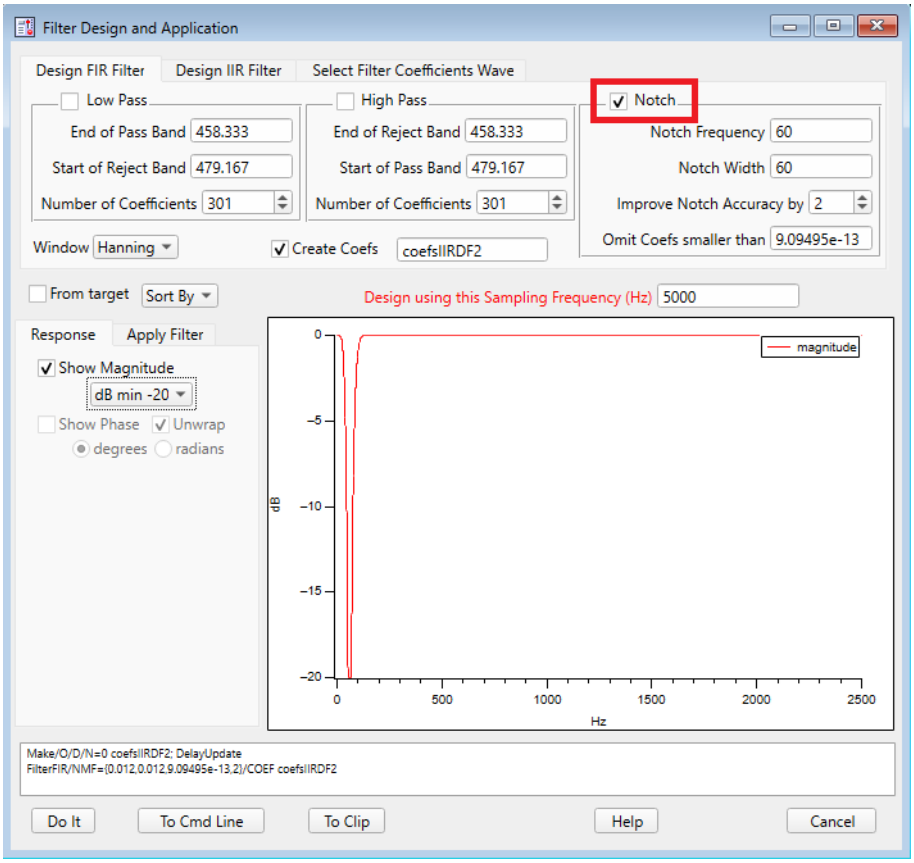
Notch（ノッチ）フィルターは通常、目的の信号に干渉する非常に狭い周波数範囲を排除するために使われます。
生理学的ウェーブフォームから 50 または 60 Hz の電力信号の干渉を除去する場合が、その使用例のひとつです。

サンプリング周波数 5000 Hz、信号 200 Hz、干渉 60 Hz の合成ウェーブフォームがあるとします。
右のグラフは、そのスペクトル成分を示しています。signalWithHum を FFT し、dB になるようにします。

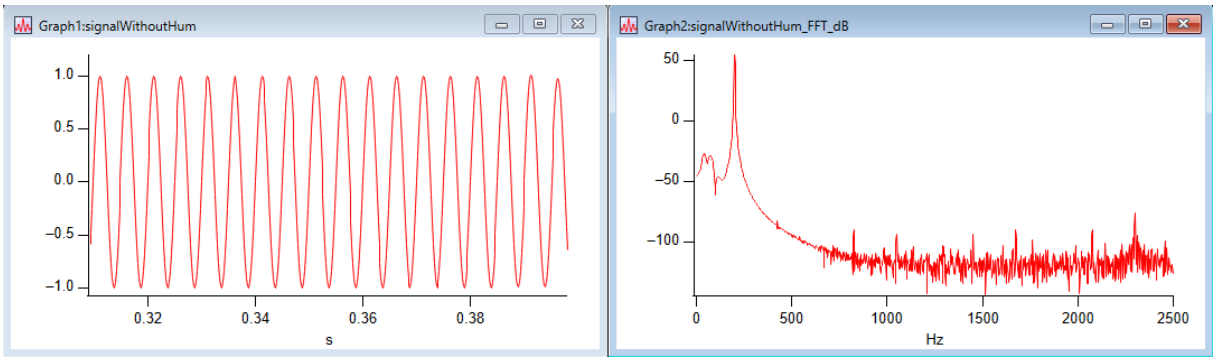
```
Duplicate/O signalWithHum_FFT, signalWithHum_FFT_dB
signalWithHum_FFT_dB = 20*log(signalWithHum_FFT_dB)
```



Notch チェックボックスをオンにし、Low Pass と High Pass チェックボックスをオフにして、ノッチのみのフィルターを作成します。



FilterFIR コマンドは、高精度な計算を使って、選択した周波数で深いノッチを取得し、より深いノッチを取得するために周波数を調整することを優先します。
Improve Notch Accuracy の値（FilterFIR ドキュメントの nMult）は、ノッチを実装するために使われる係数の数を間接的に設定します。
このフィルター設計を適用した結果は次の通りです。



干渉する 60 Hz の信号は大幅に軽減されました。

IFDL を使ったその他の FIR 設計

Filter Design and Application ダイアログを使って作成された FIR フィルターは、切り捨てられた sin(x)/x カーネルに「ウィンドウ」形状（Hanning WindowFunction など）を適用して作成された単純なフィルターです。

フィルターは機能しますが、高い性能（急峻なフィルター遷移帯域、不要な周波数の良好な除去）を得るには多くの係数が必要となります。
多くの場合、これらは重要な欠点ではありませんが、設計したフィルターを実際の電子機器に実装する場合、余分な係数によりコストが高くなります。

Igor Filter Design Laboratory (IFDL) パッケージを使うと、はるかに少ない係数で高性能の FIR フィルターを計算することができます。
これは、McClellan、Parks、および Rabiner の先駆的な論文で説明されている Remez Exchange アルゴリズムを使って、フィルターの応答とフィルター係数の数を最適化します。
詳細は、Remez コマンドを参照してください。

IIR 設計

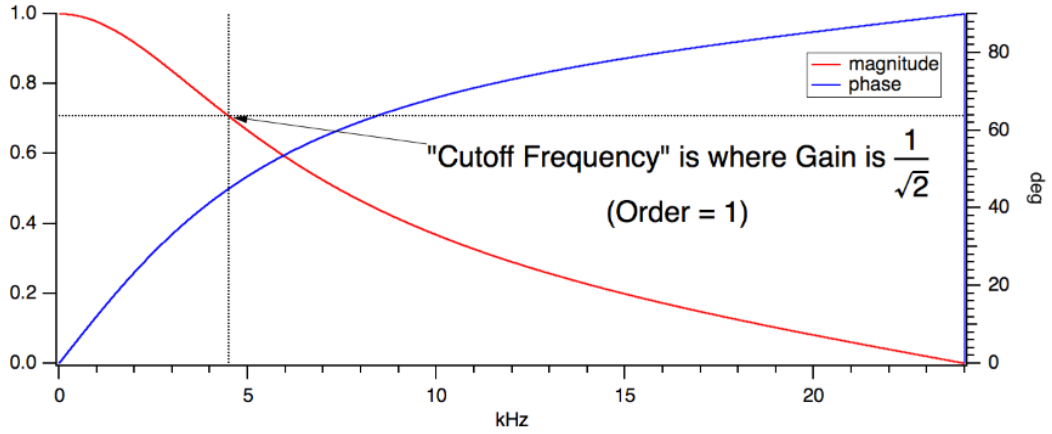
Filter Design and Application ダイアログを使って作成される IIR フィルターは、電気工学および機械工学の世界で標準的な滑らかな応答フィルターであるアナログ Butterworth（バターワース）フィルターの変換に基づいています。
詳細は、ヘルプ IIR Filters を参照してください。

Igor Filter Design Laboratory (IFDL) を使って、アナログの Bessel and Chebyshev フィルターの変換に基づく IIR フィルターを設計します。

FIR フィルターと同様に、設計が簡単な IIR フィルターは、フィルタータイプ（ローパス、ハイパス、ノッチ）および設計周波数で指定されます。

IIR 帯域周波数の選択

FIR 設計とは異なり、IIR 設計では通過帯域と遮断帯域を定義するために 1 つのカットオフ周波数を使います。
カットオフ周波数は、信号の周波数成分の応答が「カットオフ」（遮断）し始める周波数と考えることができます。
「Begins」は、応答の-3 dB 点、いわゆる「半電力」振幅に設定されます。
このポイントでの利得は 1/√2 = 0.707107 です。

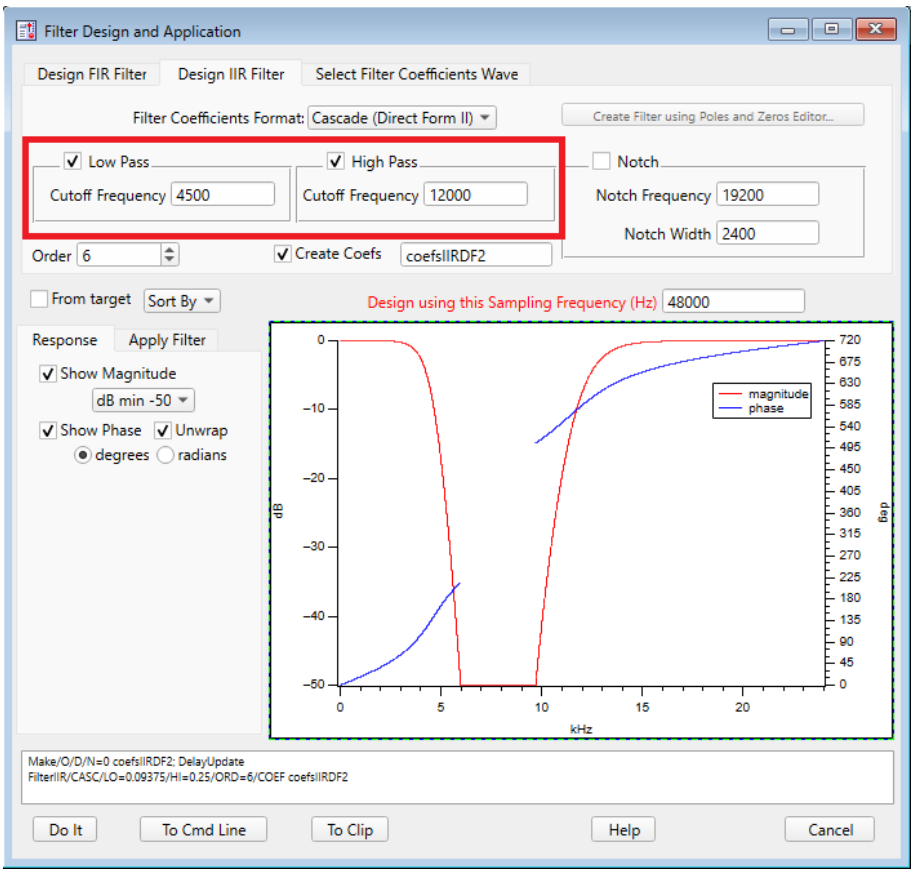


バンドパスおよびバンドストップフィルター

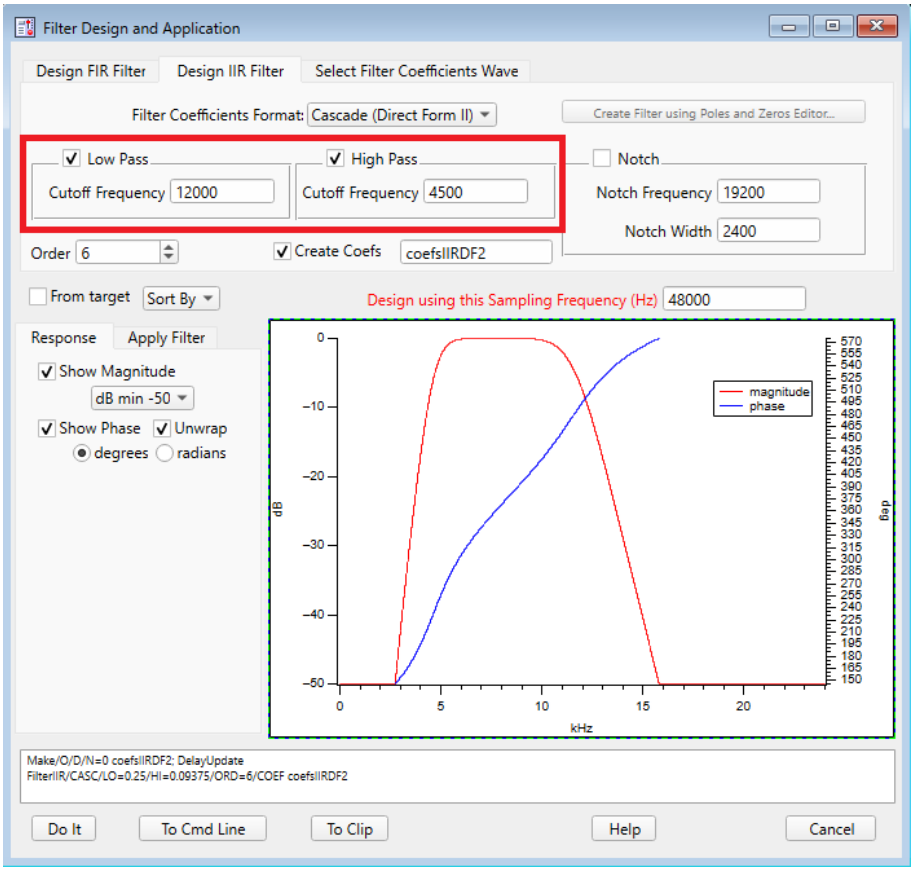
0 またはナイキスト周波数を含まない周波数範囲を通過または遮断するフィルターは、バンドパスフィルターまたはバンドストップフィルターと呼ばれます。
このようなフィルターは、狭い周波数範囲の信号を保持または除去する場合にのみ有用です。
ノッチフィルターは、独自の特別な実装を持つバンドストップフィルターの一種です。

バンドパスフィルターとバンドストップフィルターは、どちらもダイアログのローパスおよびハイパス設定を使います。
違いは、どちらのカットオフ周波数が他よりも低いかという点です。

ローパスカットオフ周波数がハイパスカットオフ周波数よりも低い場合、結果はバンドストップフィルターになります。

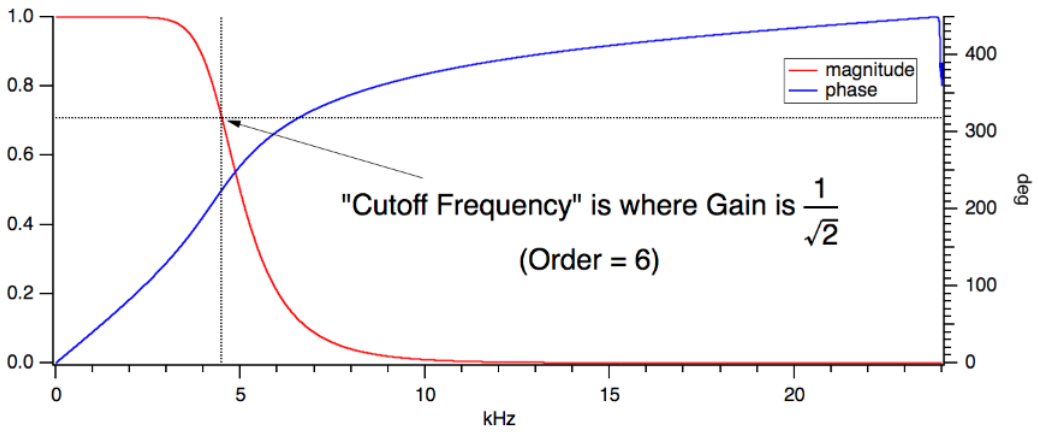


ハイパスカットオフ周波数がローパスカットオフ周波数よりも低い場合、結果はバンドパスフィルターになります。



IIR 設計における次数の選択

IIR 設計では、フィルタリングの性能を変更するために係数の数を調整する代わりに、フィルターの「次数」を使います。
基本的に、各次数は再帰的なフィルタリングの別のレイヤーを表しています。
高次フィルターは、帯域の遷移が急で、位相シフトが大きく、数値的に安定性が低下する傾向があります。
次数を 1 から 6 に増やすと、遷移がさらに急になります。

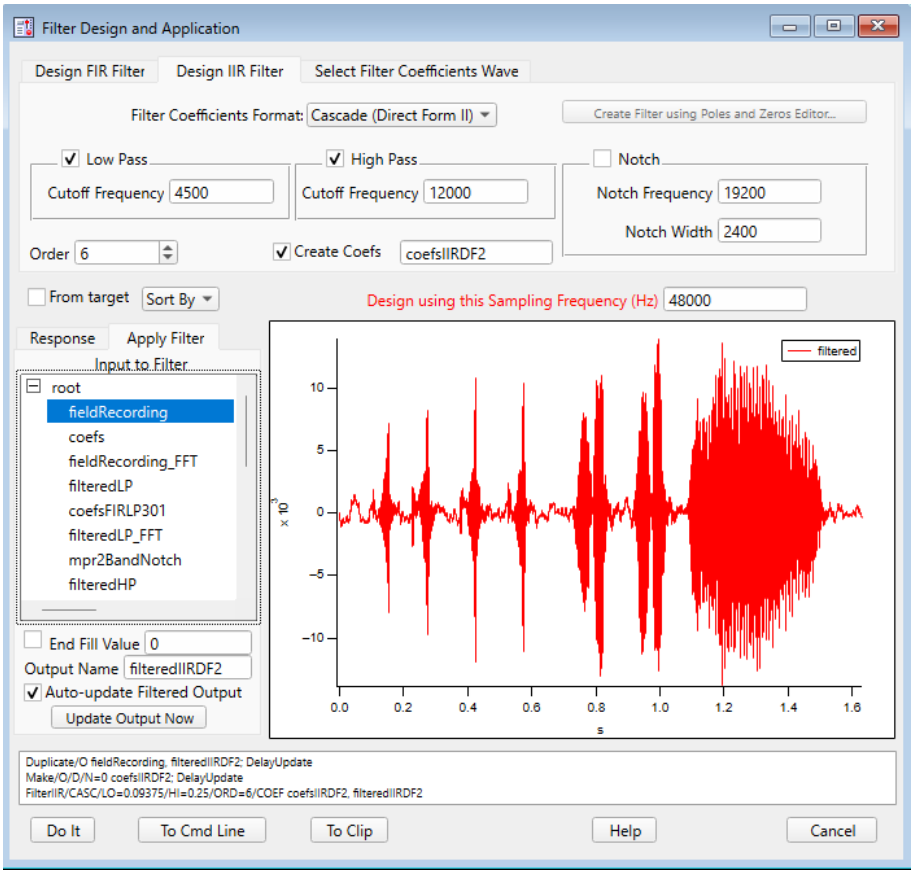


IIR フィルターをデータに適用

Apply Filter タブをクリックすると、設計したフィルターをウェーブフォームに適用した結果を確認できます。

Input to Filter リストボックスで入力ウェーブを選択し、Auto-update Filtered Output チェックボックスまたは Update Output Now ボタンをクリックします。
これにより、フィルタリングされた結果のプレビューが更新されます。

Do It をクリックして、現在のデータフォルダーに最終的な出力ウェーブを作成します。
Output Name フィールドで、最終的な出力ウェーブの名前を設定できます。
ここでは、filteredIIRDF2 を使いました。



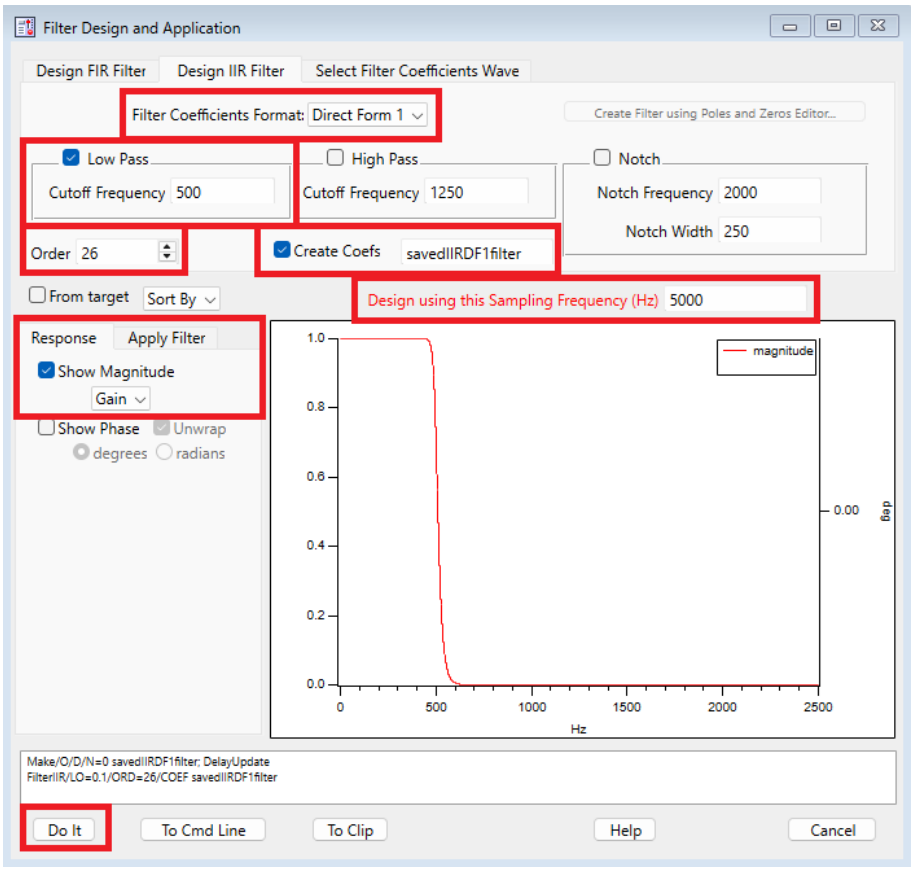
デジタルフィルターの評価

Filter Design and Application ダイアログの Response タブにあるグラフは、フィルターが入力ウェーブフォームにどのような影響を与えるかを評価する最も直接的な方法です。

また、元のデータとフィルタリングしたデータをグラフ化して、詳細な視覚的比較を行うこともできます。

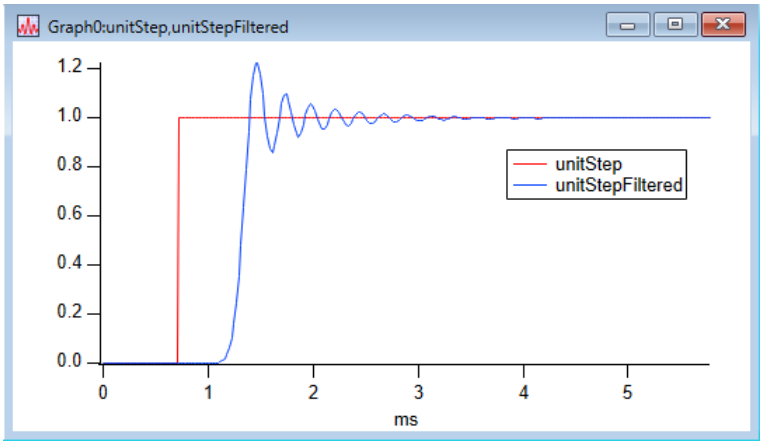
電気工学から借用した有用な手法として、フィルターが理想的な「単位ステップ」ウェーブフォームにどのように反応するかを確認する方法があります。

次のサンプルを動作させるためにフィルターsavedIIRDF1filter を作成しておきます。



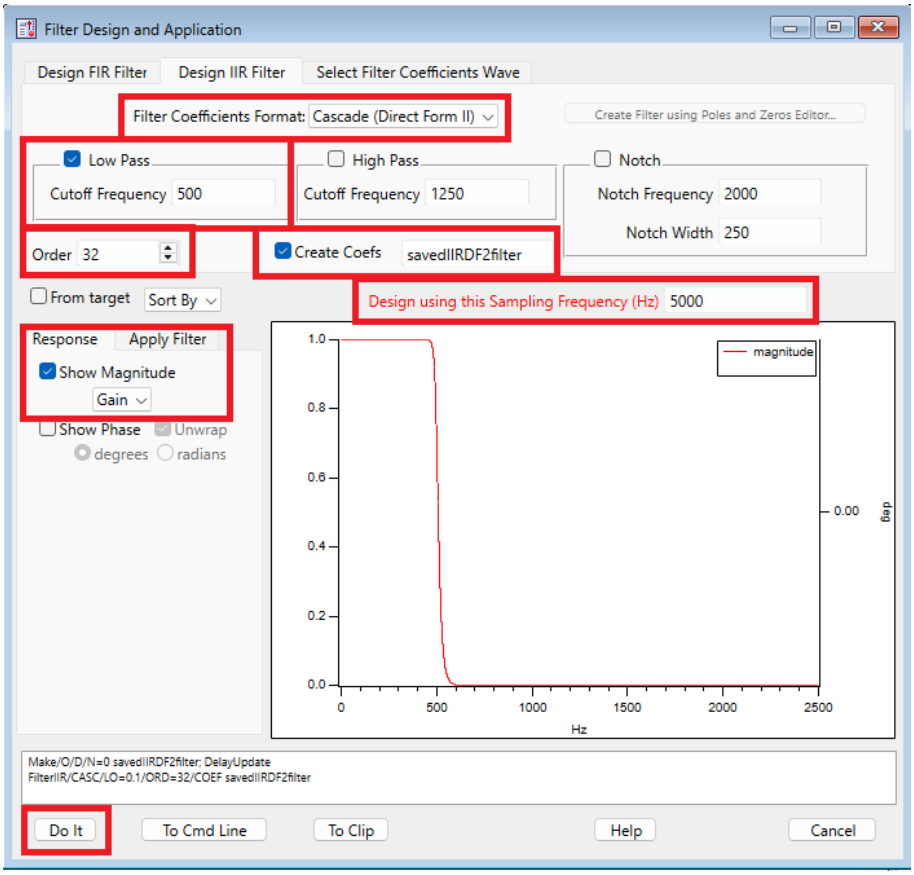

```
Make/O yourData; Setscale/P x, 0, 1/44100, "s", yourData      // サンプルのデータを生成

Make/O/N=256 unitStep = p >= 32              // 因果 IIR フィルターに適した単位ステップウェーブフォームを作成する
CopyScales/P yourData, unitStep              // データと同じサンプリング周波数を使う
Duplicate/O unitStep, unitStepFiltered       // FilterIIR は、フィルタリング結果で入力を上書きする
FilterIIR/DIM=0/COEF=savedIIRDF1filter unitStepFiltered      // DF I フィルターを単位ステップウェーブフォームに
                                                             // 適用する
Display unitStep, unitStepFiltered           // フィルタリングされていないウェーブとされたウェーブを表示する
```

トレースの色を変更し、凡例を追加すると次のようになります。

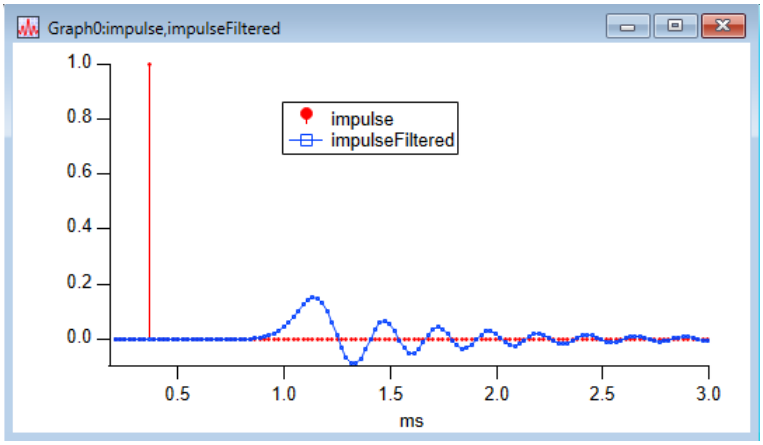


次の例は、フィルターが理想的な「単位インパルス」ウェーブフォームにどのように反応し、その FFT の大きさを表示するかを、Filter Design and Application ダイアログで示しています。

次のサンプルを動作させるためにフィルターsavedIIRDF2filter を作成しておきます。

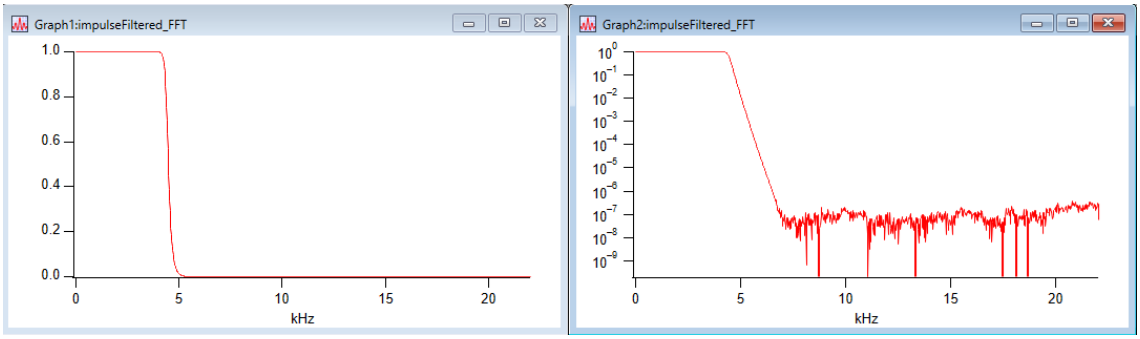



```
Make/O yourData; Setscale/P x, 0, 1/44100, "s", yourData      // サンプルのデータを生成

Make/O/N=2048 impulse = p == 16              // 因果 IIR フィルターに適したインパルスウェーブフォームを作成
CopyScales/P yourData, impulse               // インパルス長をステップ長よりも長くすることで、FFT の解像度を向上
Duplicate/O impulse, impulseFiltered
FilterIIR/CASC/DIM=0/COEF=savedIIRDF2filter impulseFiltered          // DF II 実装は /CASC が必要
Display impulse, impulseFiltered
```



（Help ファイル内のグラフは元データが異なるため上記とは異なります）

```
FFT/MAG/DEST=impulseFiltered_FFT impulseFiltered     // インパルス応答の大きさを計算
Display impulseFiltered_FFT
Display impulseFiltered_FFT                          // 対数軸は 20*log(応答)の計算と同じ形状をしている
ModifyGraph log(left)=1
```



他のデータに IIR フィルターを適用

設計の出力係数のコピーを保存しておけば、フィルターを再利用することができます。
例えば：

```
Duplicate/O coefs, savedIIRfilter          // フィルター設計のコピーを保持
```

保存した IIR フィルターは、FilterIIR コマンドを使って他のデータに適用することができます。

```
Duplicate/O otherData, otherDataFiltered    // FilterIIR は、入力にフィルタリング結果を上書きする
FilterIIR/DIM=0/COEF=savedIIRfilter otherDataFiltered        // Apply saved filter to copy of otherData
```

保存した IIR フィルターは、Filter ダイアログの Select Filter Coefficient Wave タブを使って、他のデータにも適用することができます。

Rotate コマンド

Rotate コマンドは、選択したウェーブのデータ値を指定したポイント数だけローテーションさせます。
Rotate Waves（Data メニュー）を選択すると、Igor は Rotate ダイアログを表示します。

ウェーブのデータの値を、数字の列として考えてみます。
指定したポイント数が正の場合、ウェーブ内のポイントは下方向にローテーションします。
指定したポイント数が負の場合、ウェーブ内のポイントは上向きにローテーションします。
列の片方の端でローテーションされた値は、もう一方の端に巻き込まれます。

ローテーション操作は、ローテーションしたウェーブの X スケールをシフトし、折り返すポイントを除き、指定したポイントの X 値がローテーションによって変化しないようにします。
これを確認するには、ウェーブの X スケーリングとデータ値をテーブルに表示し、X 値に対するローテーションの効果を確認します。

この X スケーリングの変更は、期待する内容である場合もあれば、そうでない場合もあります。
XY ペアをローテーションさせる場合、通常は期待する内容ではありません。
この場合、SetScale コマンドを使って X スケーリングの変更を元に戻す必要があります。

```
SetScale/P x,0,1,"",waveName          // waveName をご自身のウェーブ名に置き換えてください
```

スペクトルウィンドウのローテーションの例も参照してください。

多次元ウェーブのローテーションについては、MatrixOP の rotateRows、rotateCols、rotateLayers、rotateChunks 関数をご覧ください。

Unwrap コマンド

Unwrap コマンドは、指定された各ウェーブをスキャンして、モジュロ演算の効果を元に戻そうとします。
例えば、ウェーブに対して FFT を実行すると、結果は直交座標系の複素数ウェーブになります。
次のコマンドで、FFT の結果の位相を含む実際のウェーブを作成できます：

```
wave2 = imag(r2polar(wave1))
```

ただし、矩形座標から極座標への変換では、位相情報が 2π の倍数で残ります。
連続した位相情報を復元するには、次のコマンドを使用できます：

```
Unwrap 2*Pi, wave2
```

Unwrap コマンドは 1D ウェーブ専用です。
2D データの Unwrap は、かなり困難です。
詳細については、ImageUnwrapPhase コマンドを参照してください。

信号処理参考文献

Cleveland, W.S., Robust locally weighted regression and smoothing scatterplots, J. Am. Stat. Assoc., 74, 829-836, 1977.

Marchand, P., and L. Marmet, Binomial smoothing filter: A way to avoid some pitfalls of least square polynomial smoothing, Rev. Sci. Instrum., 54, 1034-41, 1983.

Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, Numerical Recipes in C, 2nd ed., 994 pp., Cambridge University Press, New York, 1992.

Savitzky, A., and M.J.E. Golay, Smoothing and differentiation of data by simplified least squares procedures, Analytical Chemistry, 36, 1627-1639, 1964.

Wigner, E. P., On the quantum correction for thermo-dynamic equilibrium, Physics Review, 40, 749-759, 1932.