内容

ビジュアルヘルプ – カーブフィッティング（3） 3
暗黙的関数のフィッティング 3
例：楕円にフィッティングする 3
フィッティング関数のフィッティング和 6
線形依存性：重大な問題点 7
フィッティング関数の和に適用される制約 7
例：加算された指数関数 8
例：加算された指数関数 9
複素数値関数によるフィッティング 10
複素数フィッティングに対するダイアログのサポートはありません 10
複素数基本フィッティング関数 10
複素数基本多変量フィッティング関数 11
複素数一括フィッティング関数 11
その他のウェーブ 12
複数プロセッサーを用いたカーブフィッティング 12
自動マルチスレッド処理によるカーブフィッティング 12
プログラムされたマルチスレッド処理によるカーブフィッティング 13
制約とスレッドセーフ関数 13
ユーザー定義フィッティング関数 14
ユーザー定義フィッティング関数の形式 14
基本フィッティング関数の形式 15
非常に長い式の中間結果 16
条件文 16
New Fit Function ダイアログは特殊なコメントを追加 17
フィッティング関数ダイアログが適切に扱えない関数 18
多変量フィッティング関数の形式 19
一括フィッティング関数 20
多変量一括フィッティング関数 25
構造体フィッティング関数 26
基本的な構造体フィッティング機能の例 27
WMFitInfoStruct 構造体 29

多変量構造体フィッティング関数 29
コマンドを使ったカーブフィッティング 30
バッチフィッティング 30
カーブフィッティングの例 31
カーブフィッティングにおける特異点 32
多項式フィッティングに関する特別な考慮事項 32
大きなオフセットを持つ X 値による誤差 32
カーブフィッティングのトラブルシューティング 33
イプシロンウェーブ 34
カーブフィッティング参考文献 35

ビジュアルヘルプ – カーブフィッティング（3）

暗黙的関数のフィッティング

時々、y = f(x) の形で表現できないモデルにデータをフィッティングさせる必要がある場合があります。
例としては、次のような方程式を用いて円や楕円を適合させる場合が挙げられます。

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$$

この式は y = f(x) の形で表せないため、標準的なユーザー定義のフィッティング関数を記述することはできません。

この問題は、従属変数と独立変数の両方に測定誤差が存在する「変数誤差問題」に関連しています。
２つの入力（独立変数 x と y）があり、両方に（おそらく）類似した測定誤差が存在します。
変数誤差フィッティングとの違いは、関数の出力が従属変数ではなくゼロである点にあります。

ODRPACK95 パッケージは、/ODR=3 フラグを使って、陰関数へのフィッティングもサポートしています。
複数の独立変数（上記の楕円の場合、x と y の 2 つ）を持つフィッティング関数を作成します。
フィッティング処理は、x と y の残差を見つけ、データから関数のゼロコンター線までの距離を最小化するフィッティング係数を求めようとします。

この時点で、変数フィッティングにおける誤差に関するセクションを読むとよいかもしれません。
詳細は「変数の誤差：直交距離回帰」のセクションを参照してください。
その多くは暗黙的フィッティングにも適用されます。

通常のフィッティングと暗黙的フィッティングにはいくつかの違いがあります。
暗黙的フィッティングでは自動宛先が行われず、履歴に印刷されるレポートには、フィッティングされたモデルの値を算出する方法を示すウェーブ代入が含まれません（実際、これを実現するのは非常に困難です）。

ODRPACK95 の動作の詳細により、暗黙的カーブフィッティングを行うと、カーブフィッティングの進捗ウィンドウは更新されず、フィッティングが 1 回の反復で完了したように見えます。
これは、すべての処理が ODRPACK95 の呼び出し内部で行われるためです。

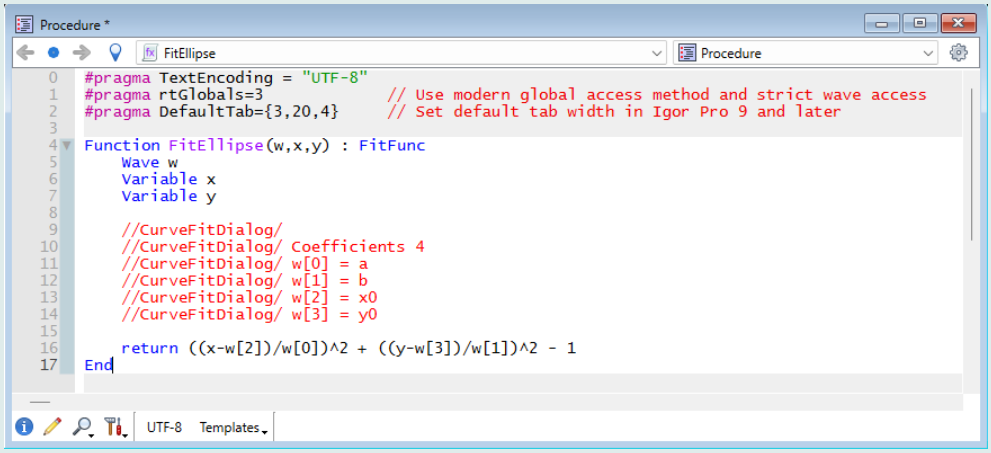
フィッティング関数は、関数の解が見つかったときにゼロを返すように記述する必要があります。
したがって、$f(x_i)=0$ という形の方程式があれば準備は完了です。
単に $f(x_i)$ を実装するフィッティング関数を作成すればよいためです。
$f(x_i)$=定数 という形の場合は、定数を方程式の反対側に移動する必要があります：$f(x_i)$-定数=0。
$f(x_i)=g(x_i)$ のような形式の場合、フィッティング関数は $f(x_i)-g(x_i)$ を返すように記述する必要があります。

例：楕円にフィッティングする

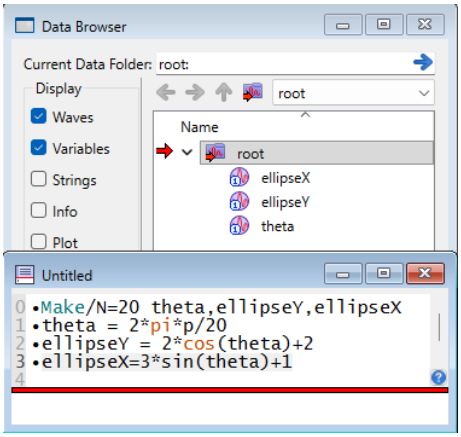
この例では、上記のような方程式をフィッティングさせる方法を示します。
この例では、楕円の中心がゼロでない座標 x_0, y_0 に位置することを許容します。

1. まず、フィッティング関数を定義します。
この関数には、フィッティング係数に覚えやすい名前を付けるための特別なコメントが含まれています（「New Fit Function ダイアログが特別なコメントを追加」のセクションを参照）。
下の関数をコピーして Procedure ウィンドウに貼り付けます。

```
Function FitEllipse(w,x,y) : FitFunc
        Wave w
        Variable x
        Variable y

        //CurveFitDialog/
        //CurveFitDialog/ Coefficients 4
        //CurveFitDialog/ w[0] = a
        //CurveFitDialog/ w[1] = b
        //CurveFitDialog/ w[2] = x0
        //CurveFitDialog/ w[3] = y0

        return ((x-w[2])/w[0])^2 + ((y-w[3])/w[1])^2 - 1
End
```



2. 暗黙的なフィッティングを行う時、Igor は入力データとフィッティング係数の調整を求め、フィッティング関数がゼロを返すようにします。
上記の楕円関数を実装するには、式右辺の「= 1」を考慮するため 1.0 を引く必要があります。

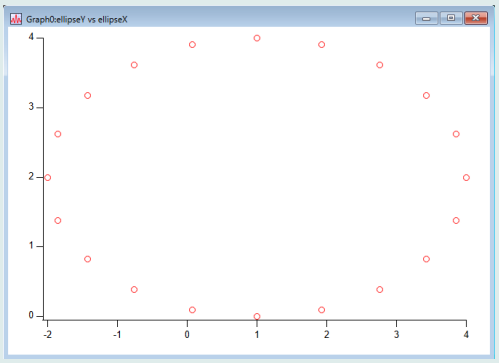


例を作成する上で難しいのは、楕円上に位置する偽のデータを生成することです。
標準的なパラメトリック方程式（y = acos(theta), x=bsin(theta)）を使ってこれを実現します。
現時点では、グラフ上で楕円を明確に確認できるよう、データに「測定誤差」は加えません。

次のコマンドを実行します。

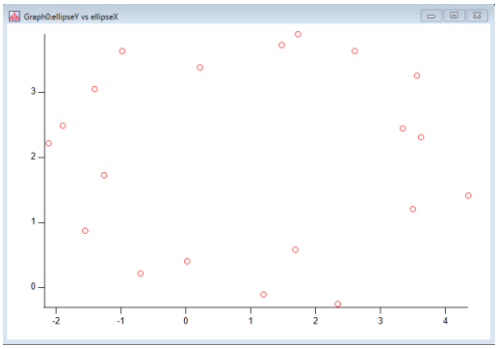
```
Make/N=20 theta,ellipseY,ellipse
theta = 2*pi*p/20
ellipseY = 2*cos(theta)+2
ellipseX=3*sin(theta)+1
```

3. **Windows→New Graph メニューを使い、次に Modify Trace Appearance ダイアログを使ってグラフを表示します。**
コマンドでは次のようになります。



```
Display ellipseY vs ellipse
ModifyGraph mode=3,marker=8
ModifyGraph
width={perUnit,72,bottom},height={perUnit,72,left}
```

4. 最後のコマンドラインは、楕円が本来のアスペクト比で表示されるように、グラフの幅と高さのモードを設定します。
次に、データに「測定誤差」を追加します。



```
// 「ランダム」なデータが常に同じになるように...
SetRandomSeed 0.5
ellipseY += gnoise(.3)
ellipseX += gnoise(.3)
```

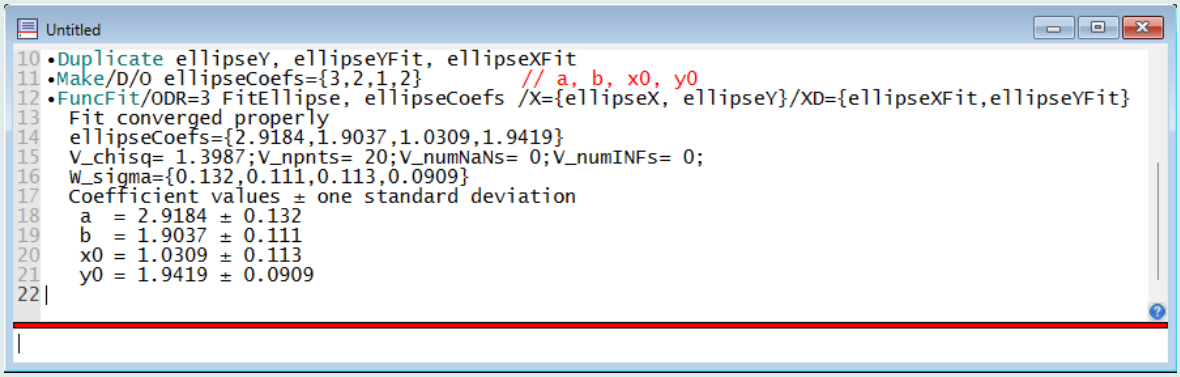
結果はひどい楕円となります。

5. さて、いよいよフィッティングを行います。
これには係数ウェーブを作成して初期推定値を埋めるとともに、フィッティング時に推定される X と Y の値を受け取るためのウェーブペアを作成する必要があります。
コマンドラインで次を実行します。

```
Duplicate ellipseY, ellipseYFit, ellipseXFit
Make/D/O ellipseCoefs={3,2,1,2}          // a, b, x0, y0
FuncFit/ODR=3 FitEllipse, ellipseCoefs /X={ellipseX, ellipseY}/XD={ellipseXFit,ellipseYFit}
```

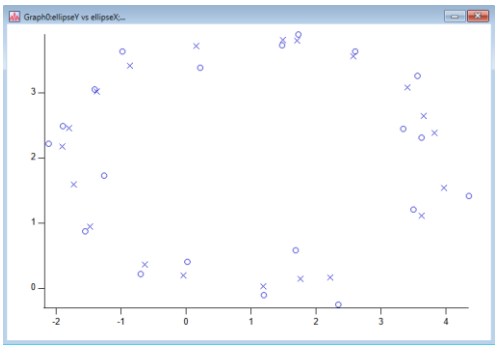
FuncFit コマンドの呼び出しでは Y ウェーブが指定されていません（通常は係数ウェーブ ellipseCoefs の直後に配置されます）。
これは暗黙的なフィッティングであるためです。

結果は次のようになります。



6. そして、**/XD フラグで指定された宛先ウェーブをグラフに追加します。**

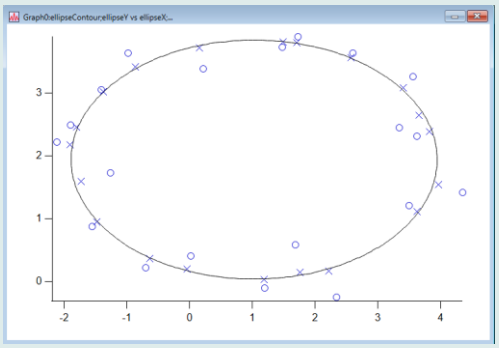
```
AppendToGraph ellipseYFit vs ellipseXFit
ModifyGraph mode=3,marker(ellipseYFit)=1
ModifyGraph rgb=(1,4,52428)
```



解を表すモデルの計算は、陰関数の根を求める必要があるため困難です。
滑らかなモデルカーブをグラフに素早く追加する方法は、X と Y の範囲におけるフィッティング関数の値を行列に埋め込み、その行列のコンタープロットをグラフに追加することです。
その後、コンタープロットを修正してゼロのコンターのみを表示させます。

7. **上記のグラフに例示関数の等高線を追加するコマンドは以下の通りです。**

```
Make/N=(100,100) ellipseContour
SetScale/I x -3,4.5,ellipseContour
SetScale/I y -.2, 5, ellipseContour
ellipseContour = FitEllipse(ellipseCoefs, x, y)
AppendMatrixContour ellipseContour
ModifyContour ellipseContour

labels=0,autoLevels={*,*,0},moreLevels=0,moreLevels={0}
ModifyContour ellipseContour rgbLines=(0,0,0)
```



結果のグラフは右のようになります。

フィッティング関数のフィッティング和

適切なモデルは、より単純なモデルの組み合わせ、典型的にはそれらの和である場合があります。
指数関数的な減衰関数の和、あるいは様々な位置にある複数の異なるピークの和である可能性があります。
ピークフィッティングには、別のユーザー定義フィッティング関数として実装されたベースライン関数が含まれる場合があります。
標準的なカーブフィッティングでは、和を実装するユーザー定義関数を記述する必要があります。

代わりに、FuncFit コマンドの代替構文を使って、自動的に合計されるフィッティング関数のリストにフィッティングできます。
各フィッティング関数の結果は個別の係数ウェーブを通じて返されるため、和における各項の結果を分離して保持することが容易です。
この構文については、FuncFit コマンドの参考文献に記載されています。

フィッティング関数の和の機能では、あらゆる種類のフィッティング関数（組み込み関数、または「ユーザー定義フィッティング関数」のセクションで説明されている派生）を組み合わせることができます。
ただし、組み込みフィッティング関数のみを使う場合でも、初期推定値を指定する必要があります。

これは、初期推定値の生成コードが、データの様々な特徴がフィッティング関数のリスト内でどのように分割されているかを判別できないためです。

線形依存性：重大な問題点

フィッティング関数のリストをフィッティングさせる場合、関数リストのフィッティング係数間に線形依存性が生じないように注意する必要があります。
最も発生しやすい依存性は、すべての組み込みフィッティング関数で使われる定数 Y オフセット、およびユーザー関数に一般的に含まれる定数 Y オフセットに起因します。
この Y オフセットは、実世界の測定でよくあるオフセット誤差を補正するものです。
たとえば、組み込み関数 exp の式は次のとおりです：

$$y_0 + A \exp(-Bx)$$

２つの項を合計すると、２つの y_0 が得られます（関数の２番目のコピーにはプライム記号が使われています）：

$$y_0 + A \exp(-Bx) + y'_0 + A' \exp(-B'x)$$

B と B′ が異なる限り、２つの指数項は問題になりません。
しかし y_0 と y'_0 は線形依存であり、数学的に区別できません。
一方を増やし他方を同量減らしても、結果は全く同じになります。
これによりフィッティング時に特異行列エラーが発生することが確実です。

解決策は、y_0 係数の１つをホールドすることです。
リスト内の各フィッティング関数には固有の係数ウェーブがあり、各関数に対してホールド文字列を指定できるため、これは簡単です（下記の最初の例を参照）。

線形依存性を導入する方法は数多く存在します。
それらは必ずしも簡単に見分けられるとは限りません。

フィッティング関数の和に適用される制約

フィッティング関数の和の機能には、関数ごとに制約を指定するキーワードは含まれていません。
ただし通常の制約指定は使用可能です。
関数リストを考慮した制約式に変換する必要があります。
（制約の使用に興味があるが構文がわからない場合は、「制約付きフィッティング」のセクションを参照してください。）

制約式では、フィッティング係数を指定するために Kn を使います。
ここで n は、係数リスト内のフィッティング係数の単純な順序番号です。
n は 0 から始まります。
合計されたフィッティング関数のリスト内で特定のフィッティング関数のフィッティング係数を参照するには、制約を適用しようとするフィッティング関数の前に、リスト内の全てのフィッティング関数に含まれる全てのフィッティング係数を考慮に入れる必要があります。
例えば、これは３つの指数項の和をフィッティングさせるコマンドの一部です：

```
FuncFit {{exp, expTerm1},{exp, expTerm2, hold="1"},{exp, expTerm3, hold="1"}} ...
```

第二の指数項の係数を制約するには、まず組み込みの exp フィッティング関数には３つのフィッティング係数があることを知っておく必要があります。
exp フィッティング関数の式は以下の通りです：

```
f(x) = y0 + A*exp(-x*invTau)
```

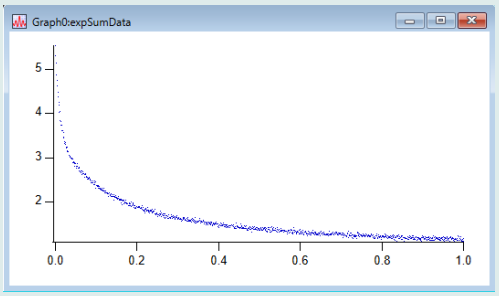
最初の加算された指数項の垂直オフセット（y0）は制約式において「K0」で表され、振幅は「K1」、逆時間(invTau)は「K2」で表されます。
２番目の指数項の制約式では、y0 の値は「K3」からカウントを継続し（ただし「線形依存性：重大な問題点」のセクションを参照し、フィッティング係数を保持と制約を同時に設定できない点に注意）、振幅には「K4」、invTau には「K5」を使います。

例：加算された指数関数

この例では、組み込み関数 exp フィッティングを用いて３つの指数関数の和をフィッティングします。
実際の作業では exp_XOffset 関数を使うことを推奨します。
この関数は X=0 ではないデータへの対応が優れており、減衰定数フィッティング係数が実際の減衰定数となります。
exp フィッティング関数は減衰定数の逆数を返します。

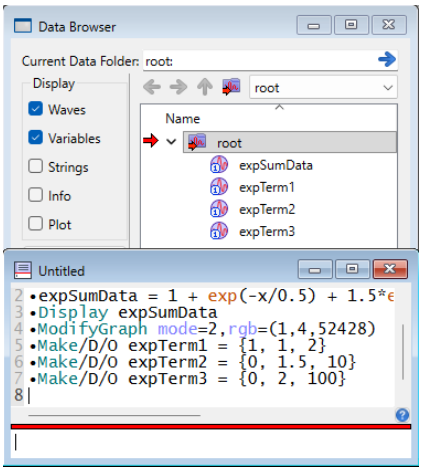
1. まず、**偽のデータを作成し、それをグラフ化します。**

   偽のデータは、フィッティング関数における垂直オフセット項の処理方法を示すために、意図的に Y オフセットを 1.0 に設定して作成しました。




```
Make/D/N=1000 expSumData
SetScale/I x 0,1,expSumData
expSumData = 1 + exp(-x/0.5) + 1.5*exp(-x/.1) +
2*exp(-x/.01)+gnoise(.03)
Display expSumData
ModifyGraph mode=2,rgb=(1,4,52428)
```

2. **次に、各フィッティング関数に対して係数ウェーブを作成します。**
   各関数のコピーにおける垂直オフセット間の線形依存性にもかかわらず、すべてのフィッティング係数を係数ウェーブに含める必要があります。
   そうしないと、関数が評価された際にフィッティングに必要な情報がすべて揃わなくなります。




```
Make/D/O expTerm1 = {1, 1, 2}
Make/D/O expTerm2 = {0, 1.5, 10}
Make/D/O expTerm3 = {0, 2, 100}
```

これらの各線は、３つの要素を持つ１つの係数ウェーブを生成します。
最初の要素は y_0、２番目は振幅、３番目は減衰定数の逆数です。
データが偽物であるため、初期推定値についてはかなり明確な見当がついています。

ベースラインオフセット 1.0 を反映するため、最初の指数係数ウェーブのみについて y_0 係数を 1 に設定しました。
もし 3 つ全てで y_0 を 1.0 に設定した場合、オフセットは 3.0 となります。

これでフィッティングを実行できます。
フィッティング関数と係数ウェーブのリストを伴う FuncFit コマンドはかなり長くなる可能性があります：

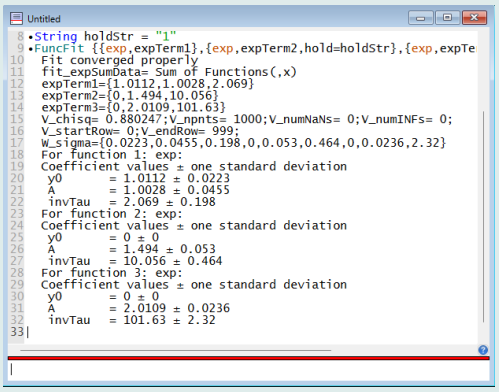
```
FuncFit {{exp, expTerm1},{exp, expTerm2, hold="1"},{exp, expTerm3, hold="1"}} expSumData/D
```

関数のリスト全体は中括弧で囲まれ、リスト内の各フィッティング関数仕様も同様に中括弧で囲まれています。
リスト内の各関数に対して、少なくともフィッティング関数と係数ウェーブを提供する必要があります。
ここでは最初の指数項に対してそのように行われています。

各関数の仕様には様々なキーワードを含めることも可能です。
ここで第 2 項と第 3 項にはホールド文字列を含めるため、hold キーワードが含まれています。
使われる文字列により、第 2 項と第 3 項の y_0 係数がゼロで固定されます。
これにより、 3 つのフィッティング関数間の線形依存性による問題が防止されます。
第 2 項および第 3 項の係数ウェーブはそれぞれの y_0 をゼロに設定するため、垂直方向のオフセットは全て第 1 項の 1 つの係数に集約されます。

3. この例では、ホールド文字列はリテラルで引用された文字列です。
任意の文字列式を使用できますが、文字列内に含まれる関数リストを使う場合の制限については以下を参照してください。

```
String holdStr = "1"
FuncFit
{{exp,expTerm1},{exp,expTerm2,hold=holdStr},{exp,expTer
m3,hold="1"}} expSumData/D
```



例：加算された指数関数

最初の例における FuncFit コマンドの関数リストは中程度の長さですが、さらに長くなる可能性があります。
20 個のピーク関数と 1 個のベースライン関数を合計したい場合、リストはコマンドラインの 2500 バイト制限を容易に超える可能性があります。

幸いなことに、関数の仕様を文字列変数に構築し、string キーワードを使用できます。
最初の例をコマンドラインで実行すると、次のようになります。

```
String myFunctions="{exp, expTerm1}"
myFunctions+="{exp, expTerm2, hold=¥"1¥"}"
myFunctions+="{exp, expTerm3, hold=¥"1¥"}"
FuncFit {string = myFunctions} expSumData/D
```

これらのコマンドは、関数の仕様を 1 つずつ追加して関数リストを構築します。
2 行目と 3 行目は、リストに関数を追加するために演算子 += を使っています。
各関数の仕様は、それぞれ対応する中括弧を含みます。

ホールド文字列の扱いに注意してください。
引用符付き文字列式に引用符を含めるには、引用符をエスケープする必要があります。
そうしないと、コマンドラインパーサーはホールド文字列を囲む最初の引用符を閉じ引用符と認識します。

関数リスト文字列は、FuncFit が実行されているコンテキストの外で実行時に解析されます。
したがって、ユーザー定義関数内でローカル変数を参照することはできません。
ホールド文字列は、ここに示すように引用符付きリテラル文字列であるか、完全なデータフォルダーパスを含むグローバル変数への参照である可能性があります。

```
String/G root:myDataFolder:holdStr="1"
String myFunctions="{exp, expTerm1}"
myFunctions+="{exp, expTerm2, hold=root:myDataFolder:holdStr}"
myFunctions+="{exp, expTerm3, hold=root:myDataFolder:holdStr}"
FuncFit {string = myFunctions} expSumData/D
```

複素数値関数によるフィッティング

Igor Pro 9.0 より前では、複素数値データに複素数値関数をフィッティングするには、実数部と虚数部を 1 つの実数値ウェーブにまとめる特殊なデータ構成を用いた実数値フィッティング関数を記述する必要がありました。
現在では、Igor は複素数値フィッティング関数への直接フィッティングをサポートしています。

複素数値関数のフィッティングにおける基本要件は、複素結果を返すフィッティング関数を記述することです。
Igor は、ユーザー定義フィッティング関数の従来形式（基本フィッティング関数の形式を参照）と一括形式（一括フィッティング関数）をサポートしていますが、基本フィッティング関数の戻り値の型または一括フィッティング関数における Y ウェーブは複素数でなければなりません。
構造フィッティング関数やフィッティング関数の和の形式では、複素数フィッティング関数はサポートされていません。

複素数フィッティングに対するダイアログのサポートはありません

Curve Fitting ダイアログでは、複素数ウェーブを選択したり、複素値を返す関数をフィッティングしたりすることはできません。
ダイアログ内で実数値関数とウェーブを使ってフィッティングを設定し、To Cmd Line ボタンをクリックして生成されたコマンドをコマンドラインにコピーし、そこでコマンドを編集する方法が考えられます。
おそらく、FuncFit コマンドのリファレンスドキュメントを参照して、コマンドを直接作成する方が簡単でしょう。

複素数基本フィッティング関数

複素数値関数の基本的な形式は次のようになります。

```
Function/C F(Wave w, Variable xx) : FitFunc
	<body of function>
	<return statement>
End
```

「Function/C」の /C は、関数が複素数値を返すことを Igor に伝えます。
上記のように、この関数は実数値の係数ウェーブと実数値の独立変数を受け取ります。
特定の関数で係数ウェーブや独立変数を複素数値にする必要がある場合は、パラメーター宣言に /C を追加してください。

```
Function/C F(Wave/C w, Variable/C xx) : FitFunc
	<body of function>
	<return statement>
End
```

ここでは係数ウェーブと独立変数の両方が複素数であることを示します。
どちらか一方または両方が複素数となり得ます。

複素数の独立変数を指定する場合、複素数値の X ウェーブを指定する必要があります。
Igor のウェーブスケーリングは複素数に対応していないため、フィッティング関数が複素数の独立変数を必要とする場合は、FuncFit コマンドへの入力として別途 X ウェーブを用意する必要があります。

複素数基本多変量フィッティング関数

通常の実数値フィッティング関数と同様に、複素数値を返す多変量フィッティング関数を記述できます。

```
Function/C F(Wave w, Variable x1, Variable x2) : FitFunc
	<body of function>
	<return statement>
End
```

従来と同様に、係数ウェーブは実数または複素数のどちらかであり、独立変数は実数または複素数のどちらかです。
独立変数は、すべて実数値であるか、すべて複素数値であるかのどちらかでなければなりません。

複素数一括フィッティング関数

特定のケースでは、フィッティング関数がモデル値を一度にすべて生成する必要がある場合があります。
複素数値関数を実装する一括フィッティング関数を記述できます。

```
Function AAOFitFunc(Wave pw, Wave/C yw, Wave xw) : FitFunc
	yw = <expression involving pw and xw>
End
```

基本形式とは異なり、この関数は複素値を返さず、Function/C で宣言されません。
代わりに、一括フィッティング関数は複素モデル値を Igor に yw ウェーブを通じて返します。
このウェーブは /C を使って複素数として宣言する必要があります。

基本形式と同様に、係数ウェーブ pw を複素数または実数に設定でき、独立変数ウェーブ xw を複素数または実数に設定できます。
ただし、FuncFit コマンドに渡すウェーブの型と一致している必要があります。

一括フィッティング関数も、x 入力ウェーブを追加することで多変量関数とすることができます。
基本形式と同様に、独立変数入力はすべて実数または複素数のどちらかでなければなりません。

その他のウェーブ

フィッティング中に、さまざまな他のウェーブを使うことができます。
例えば、重み付けウェーブ、マスキングウェーブ、イプシロンウェーブ、残差ウェーブ、そしてフィッティング解のモデル値を受け取る宛先ウェーブです。
これらのウェーブはすべて、フィッティング関数の性質に応じて複素数または実数でなければなりません。

従属変数値に適用される重みは常に複素数でなければなりません。
というのも、複素数フィッティング関数は定義上、複素数の従属変数値を返すからです。
残差ウェーブと宛先ウェーブも同様の理由で複素数でなければなりません。

イプシロンウェーブを使う場合、係数ウェーブと一致させる必要があります。

マスクウェーブは実数でなければなりません。
データポイントを選択する時に、入力データの虚数部のみを選択することはできません。

従属変数と独立変数の両方に誤差が含まれる場合、X 重み付けウェーブは独立変数入力のタイプと一致させる必要があります。

複数プロセッサーを用いたカーブフィッティング

複数のプロセッサーを搭載したコンピューターを使っている場合、それらを活用したいと考えるでしょう。
カーブフィッティングは、複数のプロセッサーを以下の 2 つの方法で活用できます。

- 自動マルチスレッド処理
- プログラムされたマルチスレッド処理

自動マルチスレッド処理によるカーブフィッティング

基本形式または一括形式で組み込みのフィッティング関数、あるいはスレッドセーフなユーザー定義フィッティング関数を使う場合、データセットが「十分に大きい」と Igor が判断すれば、自動的に複数のプロセッサーが使われます。
「十分に大きい」とされるデータポイント数は、MultiThreadingControl コマンドによって設定されます。

MultiThreadingControl がカーブフィッティングに使うキーワードは 2 つあります。
CurveFit1 キーワードは、組み込みおよび基本形式のユーザー定義フィッティング関数による自動マルチスレッド処理をコントロールします。
CurveFitAllAtOnce キーワードは、一括形式のユーザー定義フィッティング関数による自動マルチスレッド処理をコントロールします。
これらの制限の適切な設定は、フィッティング関数の複雑さと問題の性質によって異なります。
最適な設定は実験によってのみ決定できます。

Igor 7 で追加された自動マルチスレッド処理により、CurveFit /NTHR フラグは廃止されました。
この自動マルチスレッド処理は、Igor 6 で使用可能なスレッド処理よりも効果的です。

ODR フィッティング（「変数の誤差：直交距離回帰」のセクションを参照）における自動マルチスレッド処理は、多数のフィッティング係数を持つフィッティング関数や、計算コストの高いユーザー定義のフィッティング関数で最も効果的です。
意外にも、入力データポイントが少ない場合にも効果的である可能性があります。

ユーザー定義のフィッティング関数で Threadsafe キーワードを使い、かつそのフィッティングに対して Igor の自動マルチスレッド機能が有効になっている場合、その関数はウェーブデータやグローバル変数にアクセスしてはなりません。
したがって、その関数では WAVE、NVAR、または SVAR 宣言を使ってはなりません。

プログラムされたマルチスレッド処理によるカーブフィッティング

CurveFit、FuncFit、FuncFitMD コマンドはスレッドセーフです。
したがって、複数のプロセッサーを使う別の方法は、異なるプロセッサーを使って複数のカーブフィッティングを同時に実行することです。
これには、「スレッドセーフ関数とマルチタスク」のセクションで説明しているスレッドセーフなプログラミングが必要で、少なくとも中級レベルの Igor プログラミングスキルが求められます。

フィッティング関数は、組み込みのフィッティング関数、またはスレッドセーフなユーザー定義のフィッティング関数のどれかです。

例としては、File→Example Experiments→Curve Fitting→MultipleFitsInThreads を選択してください。

制約とスレッドセーフ関数

カーブフィッティングに対する制約を指定する通常の方法は、テキストウェーブ内の式によるものです（「制約付きカーブフィッティング」のセクションを参照）。
これらの式を解析し、カーブフィッティングで使う準備プロセスの一環として、式の一部を評価する必要があります。
そのためには、Execute コマンドの動作と非常に似たプロセスで、それらを Igor のコマンドラインインタプリタに送信する必要があります。
残念ながら、これはスレッドセーフではありません。

代わりに、「制約行列とベクトル」のセクションで説明している方法を使う必要があります。
残念ながら、この方法は理解が難しく、設定が面倒で、ミスをしやすいものです。
最善の方法は、通常のテキストウェーブ法（「制約式」のセクションを参照）で制約式を設定し、/C フラグを付けて試行フィッティングを実行することです。
Igor が必要な行列とベクトルを生成します。

ほとんどの場合、基本的な式はフィッティングごとに変化せず、制約条件の限界値のみが変化します。
その場合は、Igor が提供する行列を使い、ベクトル内の数値を変更することで制約条件の限界値を変更できます。

ユーザー定義フィッティング関数

New Fit Function ダイアログを使ってユーザー定義関数を作成すると、ダイアログで入力した情報に基づいて、Procedure ウィンドウに関数のコードが生成されます。
New Fit Function ダイアログを使うのがユーザー定義のフィッティング関数を作成する最も簡単な方法ですが、Procedure ウィンドウに直接関数を記述することも可能です。

特定の種類の複雑さについては、関数を自分で記述する必要があります。
そのような関数を作成する最も簡単な方法は、ダイアログを使って関数の骨組みを作成し、その後、Procedure ウィンドウで編集して修正することかもしれません。

このセクションでは、ユーザー定義のフィッティング関数の形式について説明します。
これにより、New Fit Function ダイアログの出力を理解できるだけでなく、自分でフィッティング関数を作成できるようになります。

ユーザー定義のフィッティング関数には、様々な状況に合わせて、記述や使用の複雑さが異なる多様な形式を使用できます。
次のセクションでは、最も単純な形式について説明します。
これは New Fit Function ダイアログで作成される形式でもあります。

ユーザー定義フィッティング関数の形式

ユーザー定義のフィッティング関数には、３つの形式を使用できます。
前述の基本形式、一括形式、そして構造体を唯一の入力パラメーターとして使う構造体フィッティング関数です。
さらに、構造体フィッティング関数には基本形式と一括形式のバリエーションがあります。
これらの各形式は特定の状況に対応します。

基本形式（「基本フィッティング関数の形式」のセクションを参照）が元の形式です。
これは一度に 1 つのモデル値のみを返します。

一括形式（「一括フィッティング関数」のセクションを参照）は、畳み込み、積分、FFT などの操作が、モデル値をすべて一度に計算する性質を持つ問題に対処します。
関数呼び出しのオーバーヘッドが低減されるため、大規模データセットでは基本形式よりも若干高速です。

構造体ベースの形式（「構造体フィッティング関数」のセクション参照）は、構造体を唯一の関数パラメーターとして使い、フィッティング中に任意の情報を関数に渡すことを可能にします。
これにより非常に柔軟になりますが、FuncFit をユーザー定義関数から呼び出す必要があるという制約も生じます。

基本フィッティング関数	**一括フィッティング関数**	**構造体フィッティング関数**
Curve Fitting ダイアログ内で選択、作成、編集が可能	Curve Fitting ダイアログ内で選択は可能だが、作成や編集は不可能	Curve Fitting ダイアログからは使用不可
適切なコメント付き、記憶補助係数名	記憶補助係数名なし	ユーザー定義関数から呼び出される FuncFit と一緒に使う必要がある
ストレートなプログラミング：1 つの X 値、1 つの戻り値	プログラミングにはウェーブの代入に関する十分な理解が必要であり、	プログラミングが最も困難。構造体の理解と、FuncFit を呼び出すドライバ関数の記述の両方が必要

	回避が難しい問題がいくつか存在する	
畳み込み、積分、FFT、または単一の操作で全データ値を使うあらゆる演算を利用するフィッティング関数を記述する効率的な方法ではない	畳み込み、積分、FFT などの演算を伴う問題において最も効率的。必要としない問題であっても、基本形式よりもはるかに高速な場合が多い	非常に柔軟。任意の情報をフィッティング関数に伝達可能。構造体経由でフィッティング進捗に関する詳細情報を伝達

基本フィッティング関数の形式

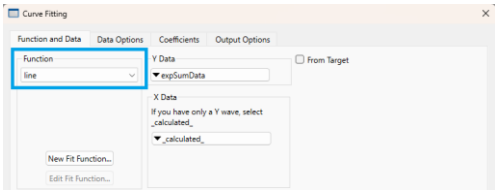
基本的なユーザー定義のフィッティング関数は以下の形式を持ちます。

```
Function F(w, x) : FitFunc
	WAVE w; Variable x

	<body of function>
	<return statement>
End
```

もちろん、関数にはより説明的な名前を選ぶこともできます。

上記の単変量の場合、関数は正確に 2 つのパラメーターを持つ必要があります。
最初のパラメーターは係数ウェーブで、慣例的に w と呼ばれます。
２番目のパラメーターは独立変数で、慣例的に x と呼ばれます。
関数がこの形式であれば、カーブフィッティング関数として認識され、FuncFit コマンドで使用できるようになります。

FitFunc キーワードは、関数がカーブフィッティングを目的としていることを示します。
FitFunc　キーワードが付与され、かつ正しい形式を持つ関数は、Curve Fitting ダイアログの Function メニューに含まれます。



FitFunc キーワードは必須ではありません。
FuncFit コマンドでは、ウェーブと変数をパラメーターとする任意の関数を使用できます。
Curve Fitting ダイアログでは、Function メニューから Show Old-Style Functions を選択すると、FitFunc キーワードのない関数を表示できますが、正しい形式に偶然一致しているだけで、フィッティング関数ではない関数も表示される場合があります。

この関数はカーブフィッティングに関する知識を一切持ちません。
入力パラメーターから戻り値を計算する方法だけを知っています。
カーブフィッティング中に、特定の X 値と特定のフィッティング係数セットに対する値が必要になったときに呼び出されます。

以下は対数関数をフィッティングさせるユーザー定義関数の例です。
対数関数は組み込みのフィッティング関数として提供されていないため、これが役立つ場合があります。

```
Function LogFit(w,x) : FitFunc
	WAVE w
	Variable x
```

```
        return w[0]+w[1]*log(x)
End
```

この例では、２つのフィッティング係数が使われています。
最初の係数は係数ウェーブのゼロ要素にあることに注意してください。
係数ウェーブのインデックスを省略することはできません。
Igor は係数ウェーブのサイズに基づいて、フィッティングが必要なフィッティング係数の数を決定します。
したがって、ウェーブの未使用の要素は特異行列エラーを引き起こします。

非常に長い式の中間結果

関数の本体は通常、かなり単純ですが、複雑になることもあります。
必要に応じて、結果を部分ごとに構築するためにローカル変数を使用できます。
他の関数や演算を呼び出したり、ループや条件分岐を使うことも可能です。

関数に多くの項がある場合、関数全体を１行に収めようとせず、中間結果を保存するためにローカル変数を使う方が便利かもしれません。
例えば、

```
return w[0] + w[1]*x + w[2]*x^2
```

の代わりに、

```
Variable val            // 結果値を蓄積するためのローカル変数
val = w[0]
val += w[1]*x
val += w[2]*x^2
return val
```

と書くことができます。

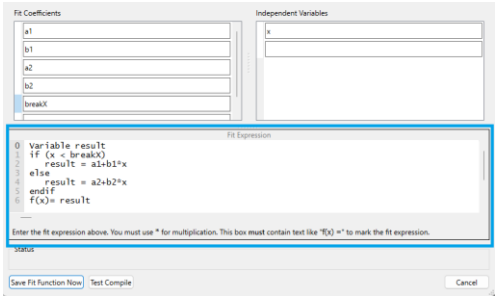
お客様から、Mathematica で導出された非常に長い式を用いたフィッティング関数を頻繁に提供されます。
これらの式は読解や理解が困難で、デバッグや修正が極めて困難です。
中間結果への代入を使って分解する努力は十分に価値があります。

条件文

フィッティング関数では if 文を含むフローコントロール文の使用が許可されています。
これにより、区分関数のフィッティングや、関数内の特異点における戻り値のコントロールが可能となります。

データ群の異なる区間に２本の直線をフィッティングする関数の例です。
パラメーターの１つを使って、どちらの直線に切り替えるかを決定します。

```
Function PieceWiseLineFit(w,x) : FitFunc
	WAVE w
	Variable x

	Variable result
	if (x < w[4])
		result = w[0]+w[1]*x
	else
		result = w[2]+w[3]*x
	endif
	return result
End
```



この関数は New Fit Function ダイアログに入力できます。
上の関数を作成した時のダイアログは右のようになっています。

New Fit Function ダイアログは特殊なコメントを追加

上記の区分線形フィッティング関数の例では、New Fit Function ダイアログは係数ウェーブのインデックスを使う代わりに係数名を使いますが、フィッティング関数内で係数に名前を付ける方法はありません。
New Fit Function ダイアログは、追加情報を含む特別なコメントをフィッティング関数に追加します。
例えば、ダイアログによって作成された PieceWiseLineFit 関数は次のようになります。

```
Function PieceWiseLineFit(w,x) : FitFunc
	WAVE w
	Variable x

	//CurveFitDialog/ These comments were created by the Curve Fitting dialog. Alteri
	//CurveFitDialog/ make the function less convenient to work with in the Curve Fit
	//CurveFitDialog/ Equation:
	//CurveFitDialog/ variable result
	//CurveFitDialog/ if (x < breakX)
	//CurveFitDialog/  result = a1+b1*x
	//CurveFitDialog/ else
	//CurveFitDialog/  result = a2+b2*x
	//CurveFitDialog/ endif
	//CurveFitDialog/ f(x) = result
	//CurveFitDialog/ End of Equation
	//CurveFitDialog/ Independent Variables 1
	//CurveFitDialog/ x
	//CurveFitDialog/ Coefficients 5
	//CurveFitDialog/ w[0] = a1
	//CurveFitDialog/ w[1] = b1
	//CurveFitDialog/ w[2] = a2
	//CurveFitDialog/ w[3] = b2
	//CurveFitDialog/ w[4] = breakX

	Variable result
	if (x < w[4])
		result = w[0]+w[1]*x
	else
		result = w[2]+w[3]*x
	endif
```

```
        return result
End
```

Edit Fit Function ボタンをクリックすると、関数コードが解析され係数の数が決定されます。
係数を命名するコメントが存在する場合、ダイアログはそれらの名前を使います。
存在しない場合、係数ウェーブの名前が使われ、それにインデックス番号が付加されて係数名となります。

フィッティング係数に覚えやすい名前を付けると、履歴エリアのカーブフィッティングレポートを確認する時に非常に役立ちます。
ダイアログと履歴に名前を表示させるために必要な最小限のコメントは、先頭コメント行に加えて係数コメント行です。
例えば、上記の関数を以下のように変更すると、Curve Fitting ダイアログと履歴レポートでフィッティング係数名を使用できるようになります。

```
Function PieceWiseLineFit(w,x) : FitFunc
        WAVE w
        Variable x

        //CurveFitDialog/
        //CurveFitDialog/ Coefficients 5
        //CurveFitDialog/ w[0] = a1
        //CurveFitDialog/ w[1] = b1
        //CurveFitDialog/ w[2] = a2
        //CurveFitDialog/ w[3] = b2
        //CurveFitDialog/ w[4] = breakX

        variable result
        if (x < w[4])
         result = w[0]+w[1]*x
        else
         result = w[2]+w[3]*x
        endif
        return result
End
```

係数の数が記載された行の前にある空白のコメントは必須です。
これらのコメントを解析するパーサーは、１行のリードイン行を破棄する必要があります。
その行には、リードインの //CurveFitDialog/ が含まれていれば、何を含めても構いません。

フィッティング関数ダイアログが適切に扱えない関数

例で示した関数では、関数コードを見るだけで、その関数に必要なフィッティング係数の数が明確にわかります。
係数のウェーブがリテラル数値でインデックス付けされているためです。
係数の数は、関数内で使われる最大のインデックス値に 1 を加えた値となります。

時々、フィッティング関数は係数ウェーブのインデックス付けにリテラル数値以外の構造を使います。
これにより、Curve Fitting ダイアログは必要な係数の数を把握できなくなります。
この場合、必要な係数の数をダイアログに指定するまで、Coefficients タブの内容は設定されません。
これを行うには、Coefficient Wave メニューから適切なポイント数の係数ウェーブを選択します。

Edit Fit Function ボタンをクリックして、そのような関数を編集することはできません。
Procedure ウィンドウで関数を記述および編集する必要があります。

以下は任意の数のガウスピークをフィッティングできる関数の例です。
係数ウェーブの長さからフィッティングするピーク数を決定します。
したがって、係数へのアクセスには定数ではなく変数を使います。

```
Function FitManyGaussian(w, x) : FitFunc
        WAVE w
        Variable x

        Variable returnValue = w[0]
        Variable i
        Variable numPeaks = floor((numpnts(w)-1)/3)
        Variable cfi

        for (i = 0; i < numPeaks; i += 1)
                cfi = 3*i+1
                returnValue += w[cfi]*exp(-((x-w[cfi+1])/w[cfi+2])^2)
        endfor
        return returnValue
End
```

多変量フィッティング関数の形式

多変量フィッティング関数は単変量関数と同じ形式を持ちますが、複数の独立変数を持ちます。

```
Function F(w, x1, x2, ...) : FitFunc
        WAVE w;
        Variable x1
        Variable x2
        Variable ...

        <body of function>
        <return statement>
End
```

データセットに平面トレンドをフィッティングさせる関数は、次のように記述できます。

```
Function Plane(w, x1, x2) : FitFunc
        WAVE w
        Variable x1, x2

        return w[0] + w[1]*x1 + w[2]*x2
End
```

独立変数の数に制限はありません。
ただし、関数宣言の行全体が 2500 バイトの単一コマンド行内に収まる必要があります。

New Fit Function ダイアログは、基本フィッティング関数と同様のコメントを多変量フフィッティング関数に追加します。
上記の `Plane()` 関数は以下のように見えるかもしれません（最初の 2 行の特殊コメントを省略して表示しています）。

```
Function Plane(w,x1,x2) : FitFunc
        WAVE w
        Variable x1
        Variable x2

        //CurveFitDialog/ These comments were created by the Curve...
        //CurveFitDialog/ make the function less convenient to work...
        //CurveFitDialog/ Equation:
        //CurveFitDialog/ f(x1,x2) = A + B*x1 + C*x2
        //CurveFitDialog/ End of Equation
        //CurveFitDialog/ Independent Variables 2
        //CurveFitDialog/ x1
        //CurveFitDialog/ x2
        //CurveFitDialog/ Coefficients 3
        //CurveFitDialog/ w[0] = A
        //CurveFitDialog/ w[1] = B
        //CurveFitDialog/ w[2] = C

        return w[0] + w[1]*x1 + w[2]*x2
End
```

一括フィッティング関数

１つの Y 値を順次計算する方式は、一部のフィッティング関数ではうまく機能しません。
これは、理論信号と計測器応答の畳み込みをフィッティングしようとする場合などに生じる畳み込みを含む関数に当てはまります。
微分方程式の解へのフィッティングも別の例となり得ます。

このケースでは、「一括」フィッティング関数を作成できます。
この関数は、X ウェーブと Y ウェーブを提供します。
X ウェーブは関数への入力であり、関数が Y 値を計算すべきすべての X 値を含みます。
Y ウェーブは出力用であり、すべての Y 値を Y ウェーブに入力します。

特定のフィッティング係数セットに対して一括関数は一度だけ呼び出されるため、基本的なフィッティング関数よりもはるかに少ない回数で呼び出されます。
関数呼び出しのオーバーヘッドが削減されるため、一括関数が必須でない問題であっても、一括関数の方が高速になる場合があります。

一括フィッティング関数の形式は以下の通りです。

```
Function myFitFunc(pw, yw, xw) : FitFunc
        WAVE pw, yw, xw

        yw = <expression involving pw and xw>
End
```

関数の結果がウェーブ yw に格納されるため、return 文は存在しません。
return 文を含めても、カーブフィッティング中は戻り値は無視されます。

X ウェーブには、カーブフィッティングに X ウェーブを提供したかどうかに関わらず、フィッティングの全 X 値が含まれます。
X ウェーブを提供しなかった場合、xw には Y ウェーブの X スケーリングから導出された等間隔の値が含まれます。

一括フィッティング関数にはいくつかの制限があります。

1. Curve Fitting ダイアログでは、一括処理関数の作成や編集はできません。
   Procedure ウィンドウで編集して作成する必要があります。
   ただし、一括処理関数は Curve Fitting ダイアログの Function メニューに一覧表示されます。

2. 「古い書き方の」一括関数は存在しません。
   FitFunc キーワードが必須です。

3. 記憶補助係数名は得られません。

以下は、一括関数を用いた指数関数的な減衰の例です。
この例は実際には意味がありません。
これを一括関数にする理由はありません。
単に、計算上の複雑さを排除した実際の関数の動作を示す例に過ぎません。
以下が一括関数としての実装例です。

```
Function allatonce(pw, yw, xw) : FitFunc
        WAVE pw, yw, xw

        // ウェーブの代入が処理を行う
        yw = pw[0] + pw[1]*exp(-xw/pw[2])
End
```

次は標準的なユーザーフィッティング関数として記述された同じ関数です。

```
Function notallatonce(pw, x) : FitFunc
        WAVE pw
        Variable x

        return pw[0] + pw[1]*exp(-x/pw[2])
End
```

一括処理関数では、exp() の引数にウェーブ xw が含まれることに注意してください。
基本形式では入力パラメーター x を使います。
これは 1 つの値を含む変数です。
一括処理関数における xw の使用は、ウェーブフォーム代入における暗黙のポイント番号機能を使っています（ヘルプ Waveform Arithmetic and Assignment を参照）。

一括フィッティング関数を作成する時には、以下の点に注意が必要です。

1. 値を代入する場合を除き、yw ウェーブを変更してはいけません。
   サイズの変更も禁止です。
   以下の操作は問題を引き起こします。

```
    Function allatonce(pw, yw, xw) : FitFunc
            WAVE pw, yw, xw
            Redimension/N=2000 yw               // 誤り！
            yw = pw[0] + pw[1]*exp(-xw/pw[2])
    End
```

2. 入力データとして提供するウェーブと同じ数のポイントを yw と xw で得られない場合があります。
   制限された範囲にフィッティングする場合、入力データに NaN が含まれる場合、またはマスクウェーブを使って入力データのサブセットを選択する場合、ポイント数が減少したウェーブが得られます。
   この状況に対応するようにフィッティング関数を記述するか、これらの機能を使わないようにしてください。

3. 自動宛先設定機能（つまり、Destination メニューの _auto_ オプション、または FuncFit コマンド単体での /D オプション）と正しく連携する一括フィッティング関数を記述するのは難しいですが、不可能ではありません。

4. xw および yw ウェーブは、ユーザーのデータウェーブではありません。
   これらは、CurveFit、FuncFit または FuncFitMD の実行中に Igor によって作成され、ユーザーの入力ウェーブからのデータで埋められます。
   これらは、フィッティングが終了すると破棄されます。
   デバッグのために、Duplicate を使ってウェーブのコピーを保存することができます。
   xw ウェーブの内容を変更しても、何の影響も与えません。

上記の例では、exp 関数の引数として xw を使い、ウェーブ代入文の特殊な特徴を利用して 2 番目の条件を満たしています。

次の例は、ガウスピークと指数関数的減衰の畳み込みをフィッティングさせます。
ベースラインオフセット、ガウスピークの振幅と幅、および指数関数の減衰定数をフィッティングさせます。
これは、インパルス信号の幅を復元するために、指数関数的に減衰するインパルス応答を持つ機器をモデル化する場合に役立ちます。

```
Function convfunc(pw, yw, xw) : FitFunc
        WAVE pw, yw, xw

        // pw[0] = ガウスベースラインオフセット
        // pw[1] = ガウス振幅
        // pw[2] = ガウス位置
        // pw[3] = ガウス幅
        // pw[4] = 機器の応答の指数減衰定数

        // 指数関数を含むウェーブを生成し、減衰定数を pw[4] とする
        // ウェーブは、あらゆる妥当な減衰定数が実質的にゼロに近づくのに十分なポイントを必要とする
        // スケーリングはゼロを中心に対称化され、Convolve/A からの X オフセットを回避する
        Variable dT = deltax(yw)
        Make/D/O/N=201 expwave          // 減衰がゼロになるのに十分な長さ
        SetScale/P x -dT*100,dT,expwave

        // expwave を指数関数的な減衰で埋める
        expwave = (x>=0)*pw[4]*dT*exp(-pw[4]*x)

        // 畳み込みの結果の振幅が変わらないように、指数関数を正規化する
        Variable sumexp
        sumexp = sum(expwave)
        expwave /= sumexp

        // 出力ウェーブにガウスピークを挿入する
        yw = pw[1]*exp(-((x-pw[2])/pw[3])^2)

        // 指数関数で畳み込む。注記 /A に注意
        Convolve/A expwave, yw

        // 端部効果を回避するため、畳み込みの後に垂直方向のオフセットを追加する
        yw += pw[0]
End
```

この関数に関して注意すべき点として：

1. 関数内部でウェーブを生成し、指数関数的減衰を保存します。
   ウェーブ生成は時間がかかる操作となり得ます。
   関数使用者にとっては不便ですが、適切なウェーブを事前に作成し、関数内で単に参照するだけで計算時間を節約できます。

2. 指数関数的な減衰を含むウェーブが畳み込み演算に渡されます。
   /A フラグを使うと、畳み込み演算によって出力ウェーブ yw の長さが変更されるのを防ぎます。
   ヘルプ Convolution を参照することを推奨します。

3. 出力ウェーブ yw は畳み込み演算のパラメーターとして使われます。
   畳み込み演算はデータが等間隔であることを前提とするため、この yw の使用は、この関数が上記の 2) または 3) を満たさないことを意味します。
   欠損点や不均等な間隔の X 値を含む入力データをフィッティングに使うと、この関数は失敗します。

ヘルプ Waveform Arithmetic and Assignment も参考になるかもしれません。

これらの問題を解決し、精度においてもより頑健な、より複雑なフィッティング関数のバージョンを以下に示します。
コード内のコメントが関数の動作を説明しています。

この関数は、Make コマンドを使って少なくとも 2 つの異なるウェーブを作成すること、および Make/D を使ってウェーブを倍精度にすることに注意してください。
これは非常に重要です。
パフォーマンスを向上させ、Igor エクスペリメントファイル内の混乱を減らすために、中間ウェーブはフリーウェーブとして作成します。

```
Function convfunc(pw, yw, xw) : FitFunc
        WAVE pw, yw, xw

        // pw[0] = ガウスベースラインオフセット
        // pw[1] = ガウス振幅
        // pw[2] = ガウス位置
        // pw[3] = ガウス幅
        // pw[4] = 機器の応答の指数減衰定数

        // 指数関数を含むウェーブを生成し、減衰定数を pw[4] とする
        // ウェーブは、あらゆる妥当な減衰定数が実質的にゼロに近づくのに十分なポイントを必要とする
        // スケーリングはゼロを中心に対称化され、Convolve/A からの X オフセットを回避する


        // resolutionFactor は、問題のパラメーターに関して指数関数がオーバーサンプリングされる度合いを
        // 設定する。この数値を大きくすると、計算に含まれる時間定数の数が増加する
        // また、問題の時間定数に対するポイント間隔も減少させる
        // 増加させると、コンピューターに必要な時間も増加する

        Variable resolutionFactor = 10

        // dt には重要な時間定数の情報が含まれている
        // モデル計算におけるポイント間隔は、指数関数的時間定数やガウス幅よりもはるかに小さく設定する必要がある
        // 反復計算が無意味だが数学的に許容される負の領域に逸脱した場合の問題を防ぐため、絶対値を使用する

        Variable absDecayConstant = abs(pw[4])
        Variable absGaussWidth = abs(pw[3])
        Variable dT = min(absDecayConstant/resolutionFactor, absGaussWidth/resolutionFactor)

        // 指数ウェーブに適したポイント数を計算する
        // 指数ウェーブの長さは 10 倍定数
        // 指数ウェーブを中央から開始させるため倍にする
        // 奇数にするため +1 し、指数ウェーブを t=0 で開始
        // t=0 は正確に中央点となる

        Variable nExpWavePnts = round((10*absDecayConstant)/dT)*2 + 1

        // 重要：倍精度ウェーブを生成する必要がある
        // パフォーマンス向上のため、また関数実行終了時に中間ウェーブが自動的にクリーンアップされるように、
        // 自由ウェーブを生成する
```

```
	Make/D/FREE/O/N=(nExpWavePnts) expwave		// 倍精度自由ウェーブ

	// この関数のバージョンでは、y 出力ウェーブを自ら生成することで、
	// 計算の解像度と精度をコントロールできるようにする
	// また、これにより後でウェーブ代入を使って、変数 x 間隔や欠損点の問題を解決できるようになる

	Variable nYPnts = max(resolutionFactor*numpnts(yw), nExpWavePnts)
	Make/D/FREE/O/N=(nYPnts) yWave		// 倍精度自由ウェーブ

	// このウェーブスケーリングは、指数関数がウェーブの中間点から開始されるように設定されている

	SetScale/P x -dT*(nExpWavePnts/2),dT,expwave

	// 中間出力のウェーブのウェーブスケーリングを設定し、上記で計算された解像度を持ち、
	// 最初の x 値から開始するようにする

	SetScale/P x xw[0],dT, yWave

	// expwave を指数関数的な減衰で満たし、X=0 から開始する

	expwave = (x>=0)*dT/pw[4]*exp(-x/pw[4])

	// 畳み込みの結果の振幅が変わらないように、指数関数を正規化する

	Variable sumexp
	sumexp = sum(expwave)
	expwave /= sumexp

	// 中間出力ウェーブにガウスピークを挿入する
	// 畳み込みには x 軸方向で等間隔のウェーブが必要だが、入力される x 値は等間隔でない可能性があるため、
	// 独自のウェーブ（yWave）を使う
	// また、ここでは垂直方向のオフセットを追加しない
	// そうすると畳み込みに問題が生じる
	// 畳み込みの前に Igor はウェーブをゼロパディングするため、ここでのベースラインはゼロでなければならない
	// そうしないと、畳み込みの開始ポイントにステップ関数が現れる

	yWave = pw[1]*exp(-((x-pw[2])/pw[3])^2)

	// 指数関数で畳み込む。注記 /A に注意

	Convolve/A expwave, yWave

	// 入力 x データに対応する適切な値を出力 y ウェーブに移動する
	// 入力 x ウェーブを用いたウェーブ代入を使う
	// これにより中間ウェーブから適切な値を抽出し、入力 x ウェーブが要求する x 値において、
	// 中間ウェーブが正確な値を持たない箇所は補間処理を行う
	// このウェーブ代入は自動の宛先設定の問題も解決する
	// 任意の x 間隔を設定する x ウェーブで関数を呼び出せるため、必要な x 値が何であれ問題にならない
	// 垂直オフセットを追加するのに適切な場所

	yw = yWave(xw[p]) + pw[0]
End
```

この関数で生成されるウェーブはすべて倍精度であることに注意してください。
Make コマンドで /D フラグを使わない場合、Igor は単精度ウェーブを生成します。
カーブフィッティングの計算は、浮動小数点丸め誤差に非常に敏感です。
中間ウェーブを倍精度にするだけで、多くのユーザーの問題を解決してきました。

なお、いずれの計算もウェーブ yw を伴わないことに留意してください。
これは、フィッティング対象データの解像度よりも細かい解像度で計算を実行可能とするためです。
別のウェーブを作成することで、yw を変更する必要がなくなります。

実際の戻り値は yWave から取り出され、ウェーブ代入における一次元ウェーブに利用可能な特別な X スケールベースのインデックスを使って yw に格納されます。

これにより、xw 入力ウェーブの任意の X 値を扱うことが可能になります。
ただし、途中の計算は xw に含まれる X の全範囲に対して行われる必要があります。

多変量一括フィッティング関数

多変量一括フィッティング関数は、複数の独立変数を扱います。
作成するには、xw ウェーブの後に追加のウェーブを単純に追加します。

```
Function MultivariateAllAtOnce(pw, yw, xw1, xw2) : FitFunc
        WAVE pw, yw, xw1, xw2

        yw = < pw ウェーブと全ての xw ウェーブを含む式>
End
```

次の例では、xw1 と xw2 の 2 つの独立変数を指定します。
それ以上の変数がある場合は、追加の X ウェーブパラメーターを追加してください。

入力データが行列ウェーブの形式であっても、すべての X ウェーブは Y ウェーブ（yw）と同じポイント数を持つ 1 次元ウェーブです。
FuncFitMD を使って N 行 M 列の行列をフィッティングする場合、Igor は行列を展開して NxM ポイントを含む 1 次元 Y ウェーブを形成します。
これは Y ウェーブパラメーターとして関数に渡されます。
X ウェーブパラメーターは、各列で値が繰り返される NxM ポイントを含む 1 次元ウェーブです。

入力データが完全な行列ウェーブに分類されることが分かっている場合、Redimension コマンドを使って一時的に yw ウェーブを行列に再度折り畳むことができる可能性があります。

```
Function MultivariateAllAtOnce(pw, yw, xw1, xw2) : FitFunc
        WAVE pw, yw, xw1, xw2

        Redimension/N=(rows, columns) yw
        MatrixOP yw = <行列計算式>
        Redimension/N=(rows*columns) yw

        yw = < pw ウェーブと全ての xw ウェーブを含む式>
End
```

ただし、この方法は細心の注意を払って使ってください。
Redimension コマンドに適した行と列を知るためには、元のデータセットのサイズについて仮定を立てる必要があります。
元の入力行列に欠損値（NaN）がある場合、これらの値は Y ウェーブからも欠落し、結果として予想よりも少ないポイント数となります。

この方法を使うと、自動宛先機能（宛先ウェーブを指定しない /D フラグ）は使用できなくなります。
なぜなら、自動宛先ウェーブはほぼ確実に期待するサイズとは異なるサイズになるからです。

上記の 2 つ目の例である一括関数と同様に、中間行列ウェーブを作成し、ウェーブ代入を使ってその行列から値を取得して yw ウェーブに代入することができます。
中間行列から正しい値を抽出する方法を見出すのは難しいかもしれません。

構造体フィッティング関数

場合によっては、フィッティング関数に追加情報を渡す必要があるかもしれません。
これには、実行時の条件を反映するために設定する必要があるが、フィッティング対象とすべきでない定数、あるいは比較に必要なルックアップテーブルや測定済み標準サンプルを含むウェーブなどが含まれます。
あるいは、フィッティング関数が非常に複雑で、何らかの管理データが必要になる場合もあるでしょう。

基本的なフィッティング関数や一括処理関数を使う場合、そのような情報はグローバル変数やウェーブを通じて伝達されなければなりません。
その結果、このグローバル情報はフィッティング関数内で参照される必要があります。
グローバルデータ要件は柔軟性を損ないます。
フィッティング関数は特定のグローバル変数やウェーブに依存し、それらが存在しなければ関数は動作しません。
グローバル情報管理の複雑さと困難さに加え、時間のかかる操作が発生する可能性があります。
グローバル情報の検索にはウェーブや変数のリストを調べ、名前を目的の名前と比較する必要があるためです。

構造体フィッティング関数は、独自の設計による構造体を１つのパラメーターとして受け取るため、このような問題に最適です。
構造体の最初の数個のメンバは特定の要件を満たす必要がありますが、それ以外は自由に設計できます。
構造体にはあらゆる種類のデータを組み込めます。
追加の利点として、数値微分の計算時にどのフィッティング係数が変化しているかをフィッティング関数に識別させるメンバー、自動生成ウェーブを埋めるためにフィッティング関数が呼び出されたことを通知するメンバー、新たなフィッティング反復が開始されたことを識別するメンバーを構造体に含めることが可能です。
フラグを返すことで FuncFit にフィッティングの中止を指示できます。

構造体フィッティング関数を使うには、次の３つのことを行う必要があります。

1. 上部で特定の標準項目を含む構造体を定義する。
2. その構造体を唯一のパラメーターとして使うフィッティング関数を記述する。
3. FuncFit 用のラッパー関数を記述し、構造体のインスタンスを作成、初期化し、/STRC パラメーターフラグ付きで FuncFit を呼び出す。

構造体の使用方法を理解してから、構造体フィッティング関数の記述を試みるべきです（ヘルプ Structures In Functions を参照）。

構造体フィッティング関数には基本型と一括型のバリエーションがあり、その違いは構造の先頭にあるメンバーによって決まります。
基本の構造体フィッティング関数の構造体の形式は次の通りです。

```
Structure myBasicFitStruct
        Wave coefw
        Variable x
        <お好きな追加要素を何でも>
EndStructure
```

構造体の名前は、Igor の命名規則に準拠する任意の名前を付けることができます。
必要なのは、最初の２つのフィールドがウェーブと変数であることです。
慣例として、これらのメンバーの使用法に合わせて、ウェーブを coefw、変数を x と命名します。
この構造体のメンバーは、基本的なフィッティング関数に必要なウェーブと変数のパラメーターに相当します。

通常の一括フィッティング関数を使うのと同じ理由で、一括フィッティング構造体フィッティング関数を使いたい場合があるかもしれません。
同様の懸念は一括構造体フィッティング関数にも当てはまります。
一括構造体フィッティング関数の作成を試みる前に、「一括フィッティング関数」のセクションを読み、理解しておく必要があります。

一括構造体フィッティング関数の構造は次のようになります。

```
Structure myAllAtOnceFitStruct
        Wave coefw
        Wave yw
        Wave xw
        <お好きな追加要素は何でも>
EndStructure
```

構造体の最初の 3 つの要素は、通常の一括フィッティング関数で要求される pw、yw、xw パラメーターに相当します。

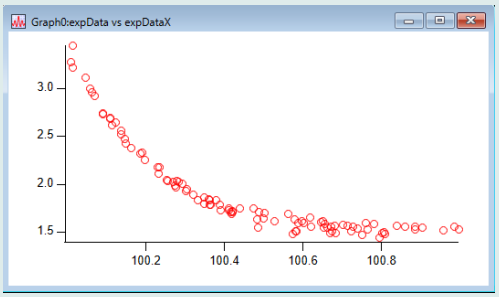
構造体フィッティング関数を用いたフィッティングの簡単な例を以下に示します。
その他の例は File メニュー→Examples→Curve Fitting から適切な例を選択してください。

基本的な構造体フィッティング機能の例

基本的な構造体フィッティング関数の例として、指数関数の X 範囲が X 値に比べて小さい場合に発生する数値的問題を補正するため、X オフセットを用いた指数減衰をフィッティングする関数を説明します。
X オフセットは、減衰振幅と数学的に区別できないため、フィッティング係数にしてはなりません。
この定数を構造体のカスタムメンバーとして保持する構造体を記述します。
この関数は、組み込みの exp_XOffset 関数（「組み込みのカーブフィッティング関数」のセクションを参照）を複製します。

1. まず、試すための偽のデータを用意します。

```
Make/D/O/N=100 expData,expDataX
expDataX = enoise(0.5)+100.5
expData = 1.5+2*exp(-(expDataX-100)/0.2) +
gnoise(.05)
Display expData vs expDataX
ModifyGraph mode=3,marker=8
```

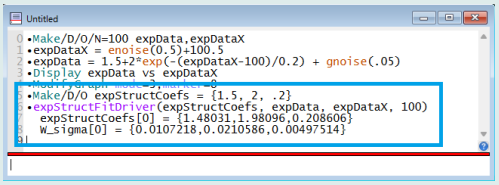


2. 次に、Procedure ウィンドウにコードを入力します。

```
// 構造体の定義
Structure expFitStruct
	Wave coefw		// 必要な係数ウェーブ
	Variable x		// 必要な x 値入力
	Variable x0		// 定数
EndStructure

// フィッティング関数
Function fitExpUsingStruct(s) : FitFunc
	Struct expFitStruct &s

	return s.coefw[0] + s.coefw[1]*exp(-(s.x-s.x0)/s.coefw[2])
End

// FuncFit を呼び出すドライバー関数
Function expStructFitDriver(pw, yw, xw, xOff)
	Wave pw		// 係数ウェーブ - 初期推定値をプリロード
	Wave yw
	Wave xw
	Variable xOff
	Variable doODR

	// 構造体のインスタンス。x0 定数のみを初期化
	// Igor (FuncFit) は必要に応じて coefw と x を初期化する
	STRUCT expFitStruct fs
	fs.x0 = xOff  // 構造体内の x オフセットの値を設定

	FuncFit fitExpUsingStruct, pw, yw /X=xw /D /STRC=fs

	//構造体フィッティング関数の履歴出力はない
	//ここでは独自の簡易レポートを出力
	print pw
	Wave W_sigma
	print W_sigma
End
```

3. 最後に、初期推定値を積んだ係数ウェーブを作成し、ドライバー関数を呼び出します。

```
Make/D/O expStructCoefs = {1.5, 2, .2}
expStructFitDriver(expStructCoefs, expData, expDataX,
100)
```



これは非常に単純な例であり、構造体フィッティング関数を用いたフィッティング処理の最も基本的な側面のみを示すことを目的としています。
上級プログラマーであれば、コントロールパネルのインターフェイスを追加し、初期推定値を自動計算するコードや、定数 x0 のデフォルト値を提供するコードを追加することが可能です。

WMFitInfoStruct 構造体

必須の構造体メンバーに加えて、必須メンバーの直後に WMFitInfoStruct 構造体メンバーを含めることができます。
WMFitInfoStruct 構造体が存在する場合、FuncFit によってフィッティングの進捗状況に関する情報が設定され、フィッティング関数が問題を検出した場合にフィッティングを停止できるメンバーが含まれます。

上記の例にある構造体に WMFitInfoStruct メンバーを追加します。

```
Structure expFitStruct
        Wave coefw                      // 必要な係数ウェーブ
        Variable x                      // 必要な X 値入力
        STRUCT WMFitInfoStruct fi       // オプションの WMFitInfoStruct
        Variable x0                     // 定数
EndStructure
```

そして WMFitInfoStruct のメンバーは：

メンバー	**説明**
char IterStarted	反復処理の最初の呼び出しではゼロではない
char DoingDestWave	自己帰還ウェーブを評価するために呼び出された時にゼロではない
char StopNow	フィッティング関数は、問題が発生しフィッティングを停止すべきであることを示すために、これをゼロ以外にする
Int32 IterNumber	完了した反復回数
Int32 ParamPerturbed	数値微分計算において摂動されるフィッティング係数のインデックス。摂動係数なしのソリューションポイントを評価する場合、-1 に設定する

IterStarted と ParamPerturbed メンバーは、一部の特殊なケースにおいて長大な計算を省略するのに役立つ可能性があります。
DoingDestWave メンバーは、一括処理型の構造体フィッティング関数において役立つ可能性があります。

多変量構造体フィッティング関数

多次元関数（複数の次元または独立変数を持つもの）をフィッティングするには、構造体の X メンバーに配列を使うだけです。
例えば、基本的な 2 次元の構造体フィッティング関数の場合：

```
Structure My2DFitStruct
        Wave coefw
        Variable x[2]
        <お好きな追加要素は何でも>
EndStructure
```

あるいは 2 次元の一括構造体フィッティング関数：

```
Structure My2DAllAtOnceFitStruct
        Wave coefw
```

```
        Wave yw
        Wave xw[2]
        <お好きな追加要素は何でも>
EndStructure
```

コマンドを使ったカーブフィッティング

Curve Fitting ダイアログでは、フィッティング係数の組み合わせに関する制約や、より複雑な構造を含むユーザー定義のフィッティング関数など、いくつかのカーブフィッティング機能が完全にサポートされていません。
また、データセットごとにダイアログを操作することなく、複数のデータセットに対してバッチフィッティングを行いたい場合もあるでしょう。
このような状況では、コマンドラインを使ってフィッティングを行う必要があります。

これを行う最も簡単な方法は、Curve Fitting ダイアログを使って、ほぼ目的の動作を行うコマンドラインを生成することです。
より複雑な制約式など、より複雑な機能を追加したい場合は、To Cmd Line ボタンをクリックし、ダイアログが生成したコマンドを編集して必要な機能を追加してください。

カーブフィッティングを行うユーザープロシージャを作成する場合、ダイアログで生成されたコマンドをコピーするには To Clip ボタンをクリックします。
その後、コマンドを Procedure ウィンドウに貼り付けます。
アプリケーションの必要に応じて編集してください。

カーブフィッティングは、CurveFit、FuncFit、FuncFitMD の 3 つのコマンドによって行われます。
これらのコマンドの詳細については、コマンドのヘルプを参照してください。

バッチフィッティング

バッチフィッティングを行う場合、おそらくループ内でカーブフィッティングコマンドを呼び出すことになるでしょう。
その場合、各フィッティングごとに履歴レポートを出力するのは望ましくないでしょう。
履歴に膨大な量のテキストが生成される可能性があります。
フィッティング中の進行状況ウィンドウも不要でしょう。
フィッティング速度を低下させ、表示、非表示によってウィンドウがフラッシュする原因となります。
最後に、フィッティング中のグラフや表の更新も避けるべきです。
計算速度が大幅に低下する可能性があります。

以下はこれら全てを実行する関数の例です。
さらに、フィッティング処理中のエラーチェックも行います。
$ 演算子の使い方が不明な場合は、ヘルプ Accessing Waves in Functions を参照してください。

```
Function FitExpToListOfWaves(theList)
        String theList

        Variable i=0
        string aWaveName = ""
        Variable V_fitOptions = 4       // 進捗状況ウィンドウを非表示
        Variable V_FitError = 0         // エラーによる中断を防止
```

```
	do
		aWaveName = StringFromList(i, theList)
		WAVE/Z aWave = $aWaveName
		if (!WaveExists(aWave))
			break
		endif

		// /N はフィッティング中に画面更新を抑制
		// /Q はフィッティング中の履歴出力を抑制
		CurveFit/N/Q exp aWave /D/R
		WAVE W_coef

		// 係数を保存
		Duplicate/O W_coef $("cf_"+aWaveName)
		// エラーを保存
		Duplicate/O W_sigma, $("sig_"+aWaveName)
		if (V_FitError != 0)
			// 結果を悪いものとしてマーク
			WAVE w = $("cf_"+aWaveName)
			w = NaN
			WAVE w = $("sig_"+aWaveName)
			w = NaN
			WAVE w = $("fit_"+aWaveName)
			w = NaN
			WAVE w = $("Res_"+aWaveName)
			w = NaN
			V_FitError = 0
		endif
		i += 1
	while(1)
End
```

この関数は非常に制限されています。
単にリスト内の複数のウェーブにフィッティングを行うだけです。
Igor には、バッチフィッティングに高度な機能を追加し、インターフェイスを提供するパッケージが含まれています。
デモを見るには、File→Example Experiments→Curve Fitting→Batch Curve Fitting Demo を選択してください。

カーブフィッティングの例

Igor Pro Folder には、カーブフィッティングの機能を実証する複数のサンプルエクスペリメントが含まれています。
これらのサンプルは、

- 制約付きフィッティング
- 多変量フィッティング
- 複数ピークフィッティング
- グローバルフィッティング
- カーソル間の直線フィッティング
- ユーザー定義関数へのフィッティング

を網羅しています。
これらすべてのエクスペリメントは、Igor Pro Folder/Examples/Curve Fitting にあります。

カーブフィッティングにおける特異点

「特異行列」エラーが発生する状況に遭遇することがあります。
これは、カーブフィッティングを実行するために解かれる連立方程式に一意の解が存在しないことを意味します。
これは通常、カーブフィッティングされた曲線に縮退（例えば、全ての Y 値が等しい場合など）が含まれている場合に発生します。

ユーザー定義関数において、係数の１つ以上が関数の戻り値に影響を与えない場合、特異行列が生成されます。
係数ウェーブは、関数内で実際に使う係数の数と完全に一致するポイント数を持つ必要があります。
そうでない場合、未使用の係数を定数として保持しなければなりません。

特定の関数では、係数の組み合わせによって、１つ以上の係数がフィッティングに影響を与えない場合があります。
ガウス関数を例に考えてみます。

$$K_0 + K_1 \exp((x - K_2)/K_3)^2$$

K_1 がゼロに設定されている場合、以降の指数関数は関数値に影響を与えません。
フィッティングは影響のない係数を報告します。
この例では、K_2 と K_3 がフィッティングに影響を与えないと報告されます。
しかし、この例が示すように、問題となるのは報告された係数ではない場合がしばしばあります。

多項式フィッティングに関する特別な考慮事項

多項式フィッティングは特異値分解法を用います。
特異値が発生し、近似係数のいくつかがゼロになった場合、データが許容する項数を超えた要求をしている可能性が高いです。
「妥当な」フィッティングが得られる最小限の項数を使うべきです。
データが高次項をサポートしていない場合、それらを含めると実際には近似精度が低下する可能性があります。

もしデータが指定した成分数に収まるべきだと本当に考えるなら、特異値のしきい値を調整してみてください。
これを行うには、V_tol という特別な変数を作成し、デフォルト値である 1e-10 より小さい値を設定します。
例えば 1e-15 を試してみてください。

多項式フィッティング中に問題が発生する別の原因は、X 値の範囲がゼロから大きくずれている場合です。
解決策は、X 値を一時的にオフセットし、フィッティングを実行した後、元の X 値を復元することです。
poly_XOffset カーブフィッティング関数がこれを自動的に行います。
詳細は「組み込みカーブフィッティング関数」のセクションを参照してください。

大きなオフセットを持つ X 値による誤差

ゼロから大きく外れた X 値の範囲にフィッティングしようとすると、単一指数関数および二重指数関数のフィッティングが狂うことがあります。

一般的に、ゼロに外挿した時に巨大または無限大の値を返す関数は、問題を引き起こす可能性があります。
解決策は、X 値を一時的にオフセットし、フィッティングを実行した後、元の X 値を復元することです。
係数を修正するために、少し代数演算を行う必要があるかもしれません。

例として、x 値が 100 から 101 の範囲の指数関数へのフィッティングを考えてみます。
x 値を一時的に 100 オフセットし、フィッティングを実行した後、x 値に 100 を加算して復元します。
実際のフィッティングでは、k0 + k1 * exp(-k2 * x) ではなく、c0 + c1 * exp(-c2 * (x - 100)) にフィッティングを行ったことになります。
式を少し整理すると、c0 + c1 * exp(-c2 * x) * exp(c2 * 100) となります。
これらの式を比較すると、k0=c0、k1=c1 * exp(c2 * 100)、k2=c2 であることがわかります。

大きな X オフセットを持つ指数関数をフィッティングする問題に対するより良い解決策は、組み込み exp_Xoffset と dblexp_XOffset フィッティング関数を使うことです。
これらのフィッティング関数は自動的に X シフトを組み込みます。
詳細は「組み込みカーブフィッティング関数」のセクションを参照してください。

高次多項式へのフィッティング時にも同様の問題が発生する可能性があります。
この場合、解の係数をオフセットなしの X 値に変換するために必要な代数演算は自明ではありません。
オフセット X 値を使って問題を再定義する方が望ましいです。

カーブフィッティングのトラブルシューティング

カーブフィッティングの結果が満足のいくものでない場合、諦める前に以下のことを試してみてください。

データが有効であることを確認してください。
すべての値が同一であってはなりません。
フィッティングさせようとしている関数に何らかの類似性を持つ必要があります。

フィッティングが反復処理の場合は、異なる初期推定値を試してください。
一部のフィッティング関数では、正しい解に非常に近い初期推定値が必要です。

ユーザー定義関数にフィッティングさせる場合は、以下を確認してください：

- 係数のウェーブは、未使用の係数を固定しない限り、関数で実際に使う係数の数と完全に同じ数のポイントを持つ必要があります。
- 初期推定値は、期待範囲が 1.0 に近い場合やイプシロンウェーブを指定した場合を除き、ゼロにしてはいけません。
  詳細は「イプシロンウェーブ」のセクションを参照してください。
- 関数が正しく動作していることを確認してください。
  代表的な定義域でプロットしてみてください。
- 関数を検証し、すべての係数が区別可能であることを確認してください。
  例えば、(k0 + k1) * x の部分では、k0 と k1 は区別できません。
  この状況が検出された場合、履歴には「Warning: These parameters may be linearly dependent:（警告: これらのパラメーターは線形に依存している可能性があります）」というメッセージが表示され、その後、区別できないと検出された 2 つのパラメーターが記載された行が続きます。

- ユーザー定義のフィッティング関数の導関数は数値的に計算されるため、関数が係数にわずかにしか依存しない場合、導関数がゼロに見えることがあります。
  解決策は、イプシロンウェーブを作成し、その値を十分に大きく設定して関数の出力にゼロ以外の差を生じさせることです。
  詳細は「イプシロンウェーブ」のセクションを参照してください。

- 前の問題の変形として、段階的に変化する関数、あるいは限られた精度でしか有効でない近似が用いられているため「ノイズの多い」関数があります。
  ここでもイプシロンウェーブを作成し、値を十分に大きく設定して、符号が一致した非ゼロの差分を生成させます。

- 各係数が関数に影響を与えていることを確認してください。
  場合によっては、係数が X 値の限定された範囲でのみ影響を与えることがあります。
  データがその範囲を十分にサンプリングしていない場合、フィッティングがうまく機能しないか、特異行列エラーが発生する可能性があります。

- 係数の最適値が無限大にならないようにしてください（無限大まで増加させるには長い時間がかかります）。

- 係数のいずれかの値に対して関数が NaN または INF を返す可能性があるかどうかを確認してください。
  これを防ぐための制約を追加できるかもしれません。
  フィッティング関数が NaN または INF 値を返した結果として特異行列エラーが発生した場合、警告が表示されます。

- 倍精度係数ウェーブを使ってください。
  カーブフィッティングは数値的に負荷が高く、通常はすべての計算を倍精度数値で行う場合に最良の結果が得られます。
  単精度係数ウェーブを使うと、数値の切り捨てによる失敗が頻繁に発生します。
  Make コマンドはデフォルトで単精度となるため、係数ウェーブを作成する時には /D フラグを使う必要があります。

- ユーザー定義フィッティング関数で中間ウェーブを使う場合、それらすべてが倍精度であることを確認してください。
  Make コマンドはデフォルトで単精度となるため、中間ウェーブを作成する時には /D フラグを使う必要があります。

イプシロンウェーブ

カーブフィッティングでは、カイ二乗面の勾配を求めるために、フィッティング関数の微分係数に対する偏微分を使います。
その後、カイ二乗面の線形化された推定値を解き、解の次の推定値（カイ二乗面の最小値）を求めます。

Igor はあなたのフィッティング関数の数学的方程式を知らないため、有限差分を用いて導関数を数値的に近似しなければなりません。
つまり、現在のフィッティング係数の推定値でモデル値を計算した後、各係数をわずかに摂動し、その差から導関数を計算します。
この小さな摂動（各係数ごとに異なる）は「イプシロン（誤差許容値）」と呼ばれます。

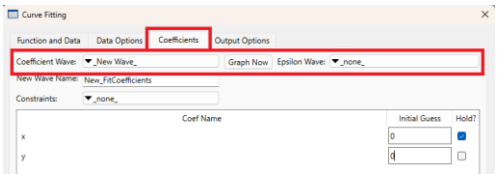
各係数に対するイプシロン値は、/E フラグを使って CurveFit または FuncFit コマンドにイプシロンウェーブを指定することで設定できます。
イプシロンウェーブを指定しない場合、Igor は次のルールに従って各係数に対するイプシロンを決定します。

係数ウェーブが単精度の場合（推奨されません）：

```
if coef[i] == 0
        eps[i] = 1e-4
else
        eps[i] = 1e-4*coef[i]
```

係数ウェーブが倍精度の場合：

```
if coef[i] == 0
        eps[i] = 1e-10
else
        eps[i] = 1e-10*coef[i]
```



Curve Fitting ダイアログで、ユーザー定義のフィッティング関数を使っている場合、Epsilon Wave メニューからイプシロンウェーブを選択できます。
New Wave を選択するか、メニューから既存のウェーブを選択すると、Igor は係数リストに列を追加し、そこでイプシロン値を編集できます。

イプシロンを明示的に設定する理由はいくつかあります。
１つは、フィッティング関数が係数に敏感でない場合です。
つまり、係数をわずかに変化させてもモデル値の変化がごくわずかである場合です。
モデルの依存性が非常に小さい場合、浮動小数点数の切り捨てによってモデル値が変化しないことがあります。
係数を変化させた時にモデルが実際に変化するように、イプシロンを十分に大きな値に設定する必要があります。

モデルが離散的またはノイズを含む場合にも、イプシロンを設定する必要があります。
これは、モデルがテーブル参照や何らかの系列解法を含む場合に発生する可能性があります。
テーブル参照の場合、イプシロンはテーブルから２つの異なる値を取得するのに十分な大きさである必要があります

級数解法の場合、ある時点で級数の項の和を止める必要があります。
級数の切り捨てによって級数の浮動小数点解像度が完全でなくなる場合、モデルの変動が級数の解像度よりも大きくなるよう、イプシロンを十分に大きく設定する必要があります。
系列には、IntegrateODE コマンドを使った常微分方程式の数値解などが含まれる場合があります。
また、FindRoots や Optimize といった近似結果を返すコマンドも含まれる可能性があります。
これらのコマンドは高精度を要求しない方が高速に実行されるため、計算精度を低下させる強い動機が生じ、結果としてイプシロンウェーブが必要となる場合があります。

カーブフィッティング参考文献

レベンバーグ・マルカート法による非線形最小二乗最適化の説明は、以下の文献の第 14.4 章に記載されている：

Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, Numerical Recipes in C, Cambridge University Press, 1988.

制約を適用するために使われるアルゴリズムは以下で示されている：

Shrager, Richard, Quadratic Programming for Nonlinear Regression, Communications of the ACM, v. 15, pp. 41-45, 1972.

この手法は、以下の文献で詳細な数学的記述がされている：

Shrager, Richard, Nonlinear Regression With Linear Constraints: An Extension of the Magnified Diagonal Method, Journal of the Association for Computing Machinery, 17, 446-452, 1970.

直交距離回帰に用いられる ODRPACK95 パッケージの参考文献：

Boggs, P.T., R.H. Byrd, and R.B. Schnabel, A Stable and Efficient Algorithm for Nonlinear Orthogonal Distance Regression, SIAM Journal of Scientific and Statistical Computing, 8, 1052-1078, 1987.

Boggs, P.T., R.H. Byrd, J.R. Donaldson, and R.B. Schnabel, Algorithm 676 - ODRPACK: Software for Weighted Orthogonal Distance Regression, ACM Transactions on Mathematical Software, 15, 348-364, 1989

Boggs, P.T., J.R. Donaldson, R.B. Schnabel and C.H. Spiegelman, A Computational Examination of Orthogonal Distance Regression, Journal of Econometrics, 38, 169-201, 1988.

非線形カーブフィッティングに関する網羅的だが読みにくい情報源：

Seber, G.A.F, and C.J. Wild, Nonlinear Regression, John Wiley & Sons, 1989.

第 5 章の冒頭部分で、非線形関数の信頼区間を計算する際に伴う仮定と近似について論じています。

カーブフィッティングと統計学に関する一般書：

Draper, N., and H. Smith, Applied Regression Analysis, John Wiley & Sons, 1966.

Box, G.E.P., W.G. Hunter, and J.S. Hunter, Statistics for Experimenters, John Wiley & Sons, 1978.