CONTENTS

コマンド

ErrorBars [/W=*winName* /CLIP=*clip* /L=*lineThick* /T=*thick* /X=*xWidth* /Y=*yWidth* /RGB=*strokeColor*] *traceName*, *mode* [*errorSpecification* [, *errorSpecification*]]

ErrorBars コマンドは、指定されたグラフ内の指定されたトレースにエラーバーを追加または削除し、エラーバーの設定を変更します。

最も一般的な形式は「エラーバー」であり、各データポイントから「キャップ」へと伸びる線です。
この線（または「バー」）の長さは、通常、測定値の不確かさ、すなわち測定の「誤差」の範囲を示すために使われます。

エラーバーは、水平方向、垂直方向、あるいはその両方に伸びることがあります。
また、対称の場合もあれば、非対称の場合もあります。

誤差は、両方向の誤差を表すボックス、半透明の帯（垂直方向の誤差のみ）、または誤差を表す楕円によって示される場合もあります。

パラメーター

traceName は、グラフ上のトレースの名前です（ヘルプ Trace Names を参照）。

traceName を含む文字列を $ 演算子と組み合わせて使うことで、*traceName* を指定することができます。

mode は以下のキーワードのいずれかです。

OFF	エラーバーなし
X	水平エラーバーのみ
Y	垂直エラーバーのみ
XY	水平＆垂直エラーバー
BOX [=*fillColor*]	ボックスのエラーバー。必要に応じて、(r,g,b) または (r,g,b,a) で表される塗りつぶし色を指定できます。色が指定されていない場合、ボックスは塗りつぶされません。*fillColor* パラメーターは Igor Pro 8.0 で追加されました。

ELLIPSE={*mode*, *p*, *alpha*}
エラー楕円をプロットします。

以下の説明にある「ewave=*ew*」というエラー指定を使って、3 列のウェーブデータとしてエラー値を指定する必要があります。

mode=0： *ew* には、X の標準偏差、Y の標準偏差、X と Y の相関係数が含まれます。

mode=1： *ew* には、X の分散、Y の分散、X と Y の共分散が含まれます。

p は、誤差楕円で表される信頼水準です。*p*=0.6837、*p*=0.95、*p*=0.997 は、それぞれ 1 標準偏差、2 標準偏差、3 標準偏差を表します。*p* の値が大きいほど、楕円は大きくなり、信頼水準が高くなります。

alpha は、0（完全に透明）から 65535（完全に不透明）までの範囲の不透明度値です。詳細は、ヘルプ Error Ellipse Color を参照してください。

楕円モードは、Igor Pro 9.0 で追加されました。

詳細は、ヘルプ Error Ellipses を参照してください。

NOCHANGE　このモードでは、モードや誤差の指定に影響を与えることなく、フラグ（後述の「フラグ」を参照）を使ってエラーバーの各要素をプログラムで変更することができます。

NOCHANGE モードは Igor Pro 9.0 で追加されました。

SHADE={*options*, *fillMode*, *fgColor*, *bkColor* [, *negFillMode*, *negFgColor*, *negBkColor*]}

options は将来の使用のために予約されており、必ずゼロに設定する必要があります。

fillMode は塗りつぶしパターンを設定します。

fillMode=0：　ソリッドブラック

fillMode=1：　ソリッドブラック

fillMode=2：　ソリッドブラック

fillMode=3：　75% グレー

fillMode=4：　50% グレー

fillMode=5：　25% グレー

fillMode>=6：　ヘルプ Fill Patterns を参照

fgColor は (r,g,b) または (r,g,b,a) です。アルファ値を含めすべてが 0、つまり (0,0,0,0) の場合、実際の色はアルファ値 20000 のトレース色になります。

bkColor はパターンのみに使われ、単色塗りつぶしの場合は単に (0,0,0) と指定できます。

negFillMode は *fillMode* と同じですが、負のエラーシェーディング用です。

negFgColor は *fgColor* と同じですが、負のエラーシェーディング用です。

negBkColor は *bkColor* と同じですが、負のエラーシェーディング用です。

後述する *errorSpecification* は、エラーシェーディングの Y 方向の振幅にのみ影響します。

シェーディングが適用されるトレースの X 値は単調でなければなりません。X 値が単調でない場合の結果は未定義となります。

詳細や例は、ヘルプ Error Shading を参照してください。アルファカラーパラメーターに関する情報は、ヘルプ Color Blending を参照してください。

SHADE は Igor Pro 7.0 で追加されました。

OFF と NOCHANGE 以外のモードについては、以下の形式のエラー仕様があります：

```
keyword [=value]
```

エラー仕様パラメーター

OFF と NOCHANGE 以外のモードを指定する場合は、キーワードの後に 0 個以上のパラメーターが続くエラー仕様を指定する必要があります。

pct=*value*　この値は、エラーバーの長さを、トレースの X 値または Y 値に対する割合（パーセンテージ）として表したものです。

sqrt　エラーバーの長さは、トレースの X 値または Y 値の平方根です。

const=*value*　*value* は、トレースの X 値や Y 値とは無関係な、エラーバーの長さです。

ewave=*ew*　*ew* は、エラー楕円パラメーターを持つ3列のウェーブデータです。各行は、1つのトレースデータポイントに関する情報を表します。*ew* の各列の解釈は、ELLIPSE キーワードとともに指定されたモードパラメーターによって異なります。

mode=0：　*ew* には、X の標準偏差、Y の標準偏差、X と Y の相関係数が含まれます。

mode=1：　*ew* には、X の分散、Y の分散、X と Y の共分散が含まれます。

ew には、各トレースデータポイントに対して3列と1行からなる実質的な 2D ウェーブが得られる限り、サブレンジ（一部）指定を含めることができます。ヘルプ Subrange Display Syntax を参照してください。

詳細は、ヘルプ Error Ellipses を参照してください。

wave=(*w1*, *w2*)　各エラーバーの長さは、正のバーの場合はウェーブ *w1* の対応するポイントから、負のバーの場合はウェーブ *w2* の対応するポイントから算出されます。

w1 を省略すると、正の棒グラフは描画されません。*w2* を省略すると、負の棒グラフは描画されません。

w1 と *w2* には、サブレンジ（一部）表示構文を使用できます。

mulWave=*ew*　各エラーバーの長さは、ウェーブ ew の対応するポイントから得られた乗数に基づいて計算されます。
トレースウェーブが *tw* の場合、エラーバーの長さは次のように計算されます。

```
positive error bar length = tw * (ew - 1)
negative error bar length = tw * (1 - 1/ew)
```

これは、対数正規分布に従う誤差の場合に適しているように、幾何平均と標準偏差を算出した場合に適切です。このようなエラーバーは、対数軸上では左右対称に見えます。

mulWave キーワードは、Igor Pro 9.0 で追加されました。

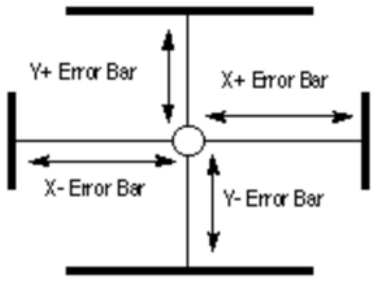
nochange　*errorSpecification* は既存の値から変更されていません。nochange を使うと、たとえば、X と Y 方向の誤差に対して *errorSpecification* を繰り返し指定することなく、ボックスモードの塗りつぶし色を変更することができます。

nochange は Igor Pro 8.0 で追加されました。

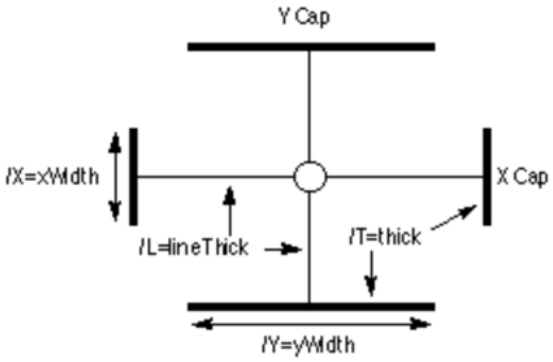
モードが XY または BOX の場合、カンマで区切った2つの誤差指定が必要です。
その他のモードでは、誤差指定は1つだけで十分です。
2つ指定するとエラーとなります。

mode と *errorSpecification* を使った例については、以下の「例」のセクションを参照してください。

mode と *errorSpecification* は、「キャップ」につながる水平線および垂直線（「バー」）の長さのみをコントロールします。
その他のサイズや太さはすべて、フラグによってコントロールされます。



mode と *errorSpecification* で
コントロールされるサイズ



フラグ値でコントロールされるサイズ

フラグ

/CLIP=*clip*
/CLIP={*clipH*, *clipV*}　プロット領域の外側にエラーバーが伸びることを許容する距離をポイント単位で設定します。最初の形式を使う場合、垂直方向と水平方向の値は両方とも *clip* に設定されます。2番目の形式を使うと、水平方向と垂直方向のクリッピングを個別に設定できます。*clipH* と *clipV* のデフォルト値は 2 ポイントです。プロット領域内のみにエラーバーを描画するように制限するには、負の値を指定できます。

/CLIP フラグは Igor Pro 9.0 で追加されました。

/L=*lineThick*　*lineThick* は、ウェーブ上のポイントから端点まで描画される X 軸と Y 軸のエラーバーの両方の太さを指定します。ELLIPSE キーワードを使ってエラー楕円を描画する場合、*lineThick* は楕円の輪郭線の太さを設定します。

/RGB=*strokeColor*　エラーバーの描画に使う線の色を設定します。*strokeColor* は (r,g,b) または (r,g,b,a) の形式で指定します。/RGB を省略した場合に使われるデフォルトの色は、エラーバーが関連付けられているトレースの色と同じになります。

ELLIPSE キーワードを使う場合、*strokeColor* は楕円の輪郭の色を設定します。

/RGB フラグは Igor Pro 8.0 で追加されました。

/T=*thick*　*thick* は、X 軸と Y 軸のエラーバーの「キャップ」の両方の太さを表します。ボックスモードでは、/T はボックスの外枠の太さを設定します。

/W=*winName*　指定されたグラフウィンドウまたはサブウィンドウのエラーバーを変更します。省略された場合、このアクションはアクティブなウィンドウまたはサブウィンドウに適用されます。プロシージャ、マクロ、またはコマンドラインで使う場合、このフラグは最初に指定する必要があります。

winName を使ってサブウィンドウを特定する時は、ウィンドウ階層の構成に関する詳細について、ヘルプ Subwindow Syntax を参照してください。

/X=*xWidth*　*xWidth* は、ポイントの左右にあるキャップの幅（実際には高さ）です。

/Y=*yWidth*　*yWidth* は、ポイントの上または下にあるキャップの幅です。

太さと幅はポイント単位で指定します。
太さのパラメーターは整数である必要はありません。
標準解像度の画面では整数の太さしか正確に表示できませんが、高解像度のデバイスでは整数以外の太さも正しく表示されます。

/T=0 を指定するとキャップを完全に非表示にし、/L=0 を指定するとキャップへの線を完全に非表示にします。

詳細

waveName に対してエラー値を指定するウェーブが *waveName* よりも短い場合、利用できないポイントについては、そのエラーウェーブの最後の値が使われます。
エラーウェーブ内のポイントに NaN（Not a Number）が含まれている場合、そのポイントに関連する半分のバーは表示されません。

例

```
// X は wave1 の 10%、Y は wave1 の 5% である
ErrorBars wave1, XY pct=10, pct=5

// X 軸のエラーバーのみ、wave1 の平方根
ErrorBars wave1,X sqrt

// Y 軸のエラーバーのみ、誤差の定数値 = 4.3
ErrorBars wave1,Y const=4.3

// X 軸と Y 軸のエラーバー。X 軸の誤差はウェーブ w1 から、Y 軸の誤差はウェーブ w2 から算出され、左右対称
ErrorBars wave1,XY wave=(w1, w1), wave=(w2, w2)

// エラーボックス：水平方向 10%、垂直方向 5%
ErrorBars wave1,BOX pct=10,pct=5

// アルファ値（透明度）が 50% の青色で塗りつぶされたエラーボックス
// 水平方向に 10%、垂直方向に 5%
ErrorBars wave1,BOX=(0,0,65535,32767) pct=10,pct=5

// エラーの計算方法を変更することなく、エラーボックスの塗りつぶし色を、不透明度 50% の赤に変更
ErrorBars wave1,BOX=(65535,0,0,32767) nochange, nochange

// Y 軸のエラーバーのみ。ウェーブ w1 は、Y+ のバーに対する誤差を持つ
// ウェーブ w2 は、Y- のバーに対する誤差を持つ
ErrorBars wave1,Y wave=(w1,w2)

// Y 軸のエラーバーのみ表示。Y+ と Y- の両方で同じウェーブを使用
// トレースの色を上書きして、エラーバーを黒にする
ErrorBars/RGB=(0,0,0) wave1,Y wave=(w1,w1)

// Y 軸のエラーバーのみ、Y+ のエラーバーはなし。ウェーブ w2 は Y- のエラーバーを持つ
ErrorBars wave1,Y wave=(,w2)

// wave1 のエラーバーを非表示にする
ErrorBars wave1,OFF
```

エラー楕円の例

ヘルプ Error Ellipse Example を参照してください。

参照

ヘルプ Error Bars、Trace Names、Programming With Trace Names

EstimatePeakSizes [/B=*baseWave*] [/X=*xWave*] [/E=*bothEdgesWave*] *edgePct*, *maxWidth*, *box*, *npks*, *peakCentersWave*, *peakWave*, *peakAmplitudesWave*, *peakWidthsWave*

EstimatePeakSizes コマンドは、推定中心が指定されたピークの振幅と幅を推定します。

EstimatePeakSizes コマンドは、主に Igor テクニカルノート #20 とその派生版で使われます。

パラメーター

edgePct は、ベースラインに対する、エッジが検出されるピーク高さの割合です。
この値は 1 から 99 の間でなければならず、通常は 50 に設定されます。

maxWidth は、*peakWidthsWave* で返される最大幅（X 座標）です。

box は、*peakWave* と *baseWave* を平滑化する時に、スライディング平均に含めるピーク値の数です。
偶数を指定した場合は、その直上の奇数が使われます。

npks は、ピークの大きさを推定するピークの数です。
この値は 1 以上でなければなりません。

peakCentersWave には、ピークの中心のポイント番号が含まれており、その長さは少なくとも *npks* である必要があります。
ピークのサイズは、これらのピークの中心ポイントからピークの端点の探索を開始することで推定されます。
i 番目のピーク中心ポイントは、*peakCentersWave*[i] に格納されていなければなりません（ここで、i は 0 から *npks*-1 までの範囲）。
peakCentersWave 内のピークの中心ポイントの値は、単調増加または単調減少でなければなりません。

peakWave は、ピークを含むウェーブです。

peakAmplitudesWave は、ピークのベースライン補正済みピーク振幅を含む出力ウェーブです。
その長さは、少なくとも *npks* 以上でなければなりません。
i 番目のピーク振幅は、*peakAmplitudesWave*[i] に格納されます。

peakWidthsWave は、X 座標におけるピークの幅を含むウェーブです。
その長さは、少なくとも *npks* 以上でなければなりません。
i 番目のピークの幅は、*peakWidthsWave*[i] に格納されます。

フラグ

/B=*baseWave*　派生データを算出するために、*peakWave* から *baseWave* を差し引き、そのデータからエッジが検出されます。

/E=*bothEdgesWave*　*bothEdgesWave* は、既存の出力ウェーブを指定します。
その長さは、少なくとも *npks**2 である必要があります。
i 番目のピークエッジのポイント座標は、*bothEdgesWave*[i*2] と *bothEdgesWave*[i*2+1] に格納されます。

X の値がポイント番号とともに増加する場合、*bothEdgesWave*[i*2] は *bothEdgesWave*[i*2+1] よりも大きくなり、これは予想外の結果となる可能性があります。

/X=*xWave*　*xWave* は、*peakWave* と *baseWave* 内の対応するポイントの X 座標を指定します。この配列の長さは *peakWave* と同じでなければならず、値は単調増加または単調減少していなければなりません。

参照

FindPeak コマンド

Execute [/Z] *cmdStr*

Execute コマンドは、*cmdStr* の内容を、あたかもコマンドラインで入力されたかのように実行します。

Execute の最も一般的な用途は、ユーザー定義関数からマクロや外部コマンドを呼び出すことです。
Igor Pro ではこのような呼び出しを直接行うことができないため、これが必要となります。
その他の用途については、ヘルプ The Execute Operation を参照してください。

/Z フラグが使われた場合、V_flag にエラーコードが格納されます。
パラメーターが不足しているスタイルマクロが呼び出され、ユーザーが Quit Macro をクリックした場合は、エラーコードは -1 となり、エラーがなかった場合は 0 となります。

フラグ

/Z　　エラーは致命的ではなく、エラーダイアログは表示されません。

詳細

コマンドラインとコマンドバッファは 1 行あたり 2500 バイトに制限されているため、cmdStr も同様に、実行可能バイト数が最大 2500 バイトに制限されます。

cmdStr 内でローカル変数を参照しないでください。
このコマンドは、マクロやユーザー定義関数によって提供されるローカル環境では実行されません。

Execute は、マクロを含む文字列式を受け入れることができます。
文字列は Macro、Proc、または Window で始まり、マクロに関する通常の規則に従っている必要があります。
最後の行を含め、すべての行はキャリッジリターンで終了している必要があります。
マクロの名前は重要ではありませんが、存在している必要があります。
/Z フラグを使っている場合を除き、エラーが報告されます。
/Z フラグを使うと、エラー状態において V_Flag に 0 以外の数値が割り当てられます。

例

実行するコマンドをローカルの文字列変数に格納し、その文字列を Execute コマンドに渡すのが良いでしょう。
次の例は、デバッグのためにその文字列を履歴に出力します。

```
String cmd
sprintf cmd, "GBLoadWave/P=%s/S=%d \"%s\"", pathName, skipCount, filename
Print cmd           // デバッグ用
Execute cmd
// マクロで実行：
Execute "Macro junk(a,b)\rvariable a=1,b=2\r\rprint \"hello from macro\",a,b\rEnd\r"
```

Execute /P[/Q/Z] *cmdStr*

Execute/P は Execute と似ていますが、コマンド文字列 *cmdStr* が即座に実行されるのではなく、コマンドキューに投入される点が異なります。
コマンドキュー内の項目は、他に何も処理が行われていない場合にのみ実行されます。
マクロや関数が実行中であってはならず、コマンドラインは空でなければなりません。

フラグ

/Q　　コマンドは、コマンドラインや履歴エリアには表示されません。

/Z　　エラーダイアログは非表示になります。

参照

コマンドキューを使った Execute/P の詳細は、ヘルプ Operation Queue を参照してください。

ExecuteScriptText [/B /W=*waitTime* /UNQ /Z] *textStr*

ExecuteScriptText コマンドは、コンパイルと実行のために指定したテキストを Windows コマンドラインに渡します。

スクリプトによって生成されたテキスト（エラーメッセージを含む）は、文字列出力変数 S_value に返されます。
S_value は、/B（バックグラウンドで実行）フラグを指定した場合にのみ設定されます。

パラメーター

textStr には、有効な Windows コマンドラインが含まれている必要があります。

フラグ

/B　スクリプトをバックグラウンドで実行します。つまり、Igor Pro をアクティブなアプリケーションのままにします。

/B を省略すると、スクリプトによって呼び出されたプログラムがアクティブなプログラムになります。/B を指定すると、Igor Pro がアクティブなプログラムのままになります。

/W=*waitTime*　*waitTime* は、ExecuteScriptText が戻るまでの最大待機時間（秒単位）です。

/UNQ　このフラグは効果を持ちません。

/UNQ フラグは、Igor Pro 7.0 で追加されました。

/Z　コード内でエラーを処理したい場合は、/Z を指定してください。ExecuteScriptText にエラーを Igor Pro に報告させたい場合は、これを省略してください。詳細は、以下の「ExecuteScriptText のエラー処理」のセクションを参照してください。

ExecuteScriptText のエラー処理

textStr には、バッチファイルへのフルパス、または実行ファイルの名前が格納されます。
これには、オプションで Windows 形式のパスや引数を指定することもできます。

```
[path]executableName [.exe] [arg1 ]...
```

path を省略し、*executableName* が ".exe" で終わる場合、または拡張子がない場合、まずレジストリを検索し、次に PATH 環境変数に沿って実行ファイルを検索します。
見つからない場合は、Igor Pro フォルダーが実行ファイルの場所として想定されます。
例えば、実行ファイル名のみで構成されるコマンドの例は以下の通りです。

```
ExecuteScriptText "calc"    // calc.exe, calculator
```

バッチファイルや *.exe 以外のファイルを呼び出す時は、フルパスを指定してください。
パス（またはファイル名）にスペースが含まれている場合は、パスを引用符で囲む必要があります。

```
ExecuteScriptText "\"C:\\Program Files\\my.bat\""
```

外側の二重引用符は、ExecuteScriptText によって処理されます。
"\" エスケープシーケンスで表される内側の二重引用符は、オペレーティングシステムに渡されるコマンド内に残ります。

/B フラグを使うと、コマンドをバックグラウンドで実行し、Igor Pro をアクティブなアプリケーションのままにすることができます。
/B を省略すると、コマンドによって起動されたプログラムがアクティブなプログラムになります。
バッチファイルを実行する場合、/B を指定すると DOS ウィンドウが表示されなくなります。

/B を指定すると、コマンドの出力は S_value 出力文字列変数を通じて返されます。
/B を省略すると、S_value は "" に設定されます。

/W=*waitTime* フラグを使うと、コマンドの実行に時間がかかりすぎた場合にタイムアウトさせることができます。
これは、GUI 以外のプログラムを呼び出す場合にのみ役立ちます。
GUI プログラムの場合、*waitTime* に指定された値にかかわらず、GUI プログラムがイベントの処理を開始するとすぐに ExecuteScriptText は処理を終了します。

GUI 以外のプログラムの場合、*waitTime* に正の値を指定すると、ExecuteScriptText はその秒数までコマンドが完了するのを待ちます。
その時間内にコマンドが完了しなかった場合、ExecuteScriptText はエラーを返します。
/W フラグを省略した場合、または *waitTime* に 0 を指定した場合、ExecuteScriptText は、時間がかかってもコマンドが完了するまで待機します。
実行中のスクリプトが、例えば cmd.exe などを介して Windows コマンドウィンドウを作成する場合、Windows プロセスはスクリプトのコマンドが実行された後ではなく、コマンドウィンドウが閉じられた時点で初めて完了します。

例

```
// DOS コマンドをバックグラウンドで実行する
ExecuteScriptText/B "hostname"; Print S_value

// バックグラウンドで MatLab を起動する
ExecuteScriptText/B "C:\\Matlab\\bin\\matlab.exe myFile.m"

// スクリプトを Windows Script Host に渡す
ExecuteScriptText/W=5 "WScript.exe \"C:\\Test Script.vbs\""

// バッチファイルを実行し、コマンドウィンドウを開いたままにする
ExecuteScriptText "cmd.exe /K \"C:\\mybatch.bat\""

// DOS コマンドを実行し、出力がある場合はそれを取得する
//   ExecuteDOSCommand(command, maxSecondsToWait)
// DOS コマンドを実行し、出力されたテキストを関数の結果として返す。
// DOS コマンドがテキストを返さない場合、"" を返す。
// maxSecondsToWait は、DOS がコマンドの実行を完了するまで待機する最大秒数。
// それ以上の時間がかかった場合、エラーが発生する。
// この関数は、Igor Pro User Folder 内にファイルを作成する：
//      IgorBatch.bat          DOS が実行すべきコマンドを保持する
//      IgorBatchOutput.txt    DOS コマンドによって生成された出力があれば、それを保持する

// 例：
//      Print ExecuteDOSCommand("echo %PATH%", 3)
Function/S ExecuteDOSCommand(command, maxSecondsToWait)
    String command                      // 例： "echo %PATH%"
    Variable maxSecondsToWait           // DOS 処理がこれより時間がかかった場合はエラーとなる

    String quoteStr = "\""

    // "Igor Pro User Files" 内のバッチファイルへのパスを取得する
    String dirPath = SpecialDirPath("Igor Pro User Files", 0, 0, 0)
    dirPath = ParseFilePath(5, dirPath, "\\", 0, 0)        // Windows パスに変換する
    String batchFilePath = dirPath + "IgorBatch.bat"
    String batchOutputFilePath = dirPath + "IgorBatchOutput.txt"

    DeleteFile/Z batchOutputFilePath

    // DOS コマンドをバッチファイルに記述する
    String dosCommand = command + " > " + quoteStr + batchOutputFilePath + quoteStr
    Variable refNum
    Open refNum as batchFilePath
```

```
    FBinWrite refNum, dosCommand
    Close refNum

    // バッチファイルを実行する
    // DOS コマンドは、/W オプションで指定された秒数以内に完了する必要がある
    // /C を指定すると、コマンドの実行後に cmd.exe が終了する
    String text
    sprintf text, "cmd.exe /C \"%s\"", batchFilePath
    ExecuteScriptText/W=(maxSecondsToWait) text

    // 出力を取得する
    String result = ""
    Open/R/Z refNum as batchOutputFilePath
    if (V_flag != 0)
        result = ""
        // result = "<No output was generated by batch file>"      // デバッグ用
    else
        // バッチファイルの内容を文字列として読み込む
        FStatus refNum
        Variable numBytesInFile = V_logEOF
        result = PadString("", numBytesInFile, 0x20)
        FBinRead refNum, result
        Close refNum
    endif

    return result
End
```

ExperimentInfo [/Q[=*quiet*]] [*keyword=value* [, *keyword=value* ...]]

ExperimentInfo コマンドは、現在のエクスペリメントに関する情報を返します。
この情報は、以下で説明するように、出力変数 V_Flag と出力文字列変数 S_Value に格納されて返されます。

ExperimentInfo は Igor Pro 8.0 で追加されました。

現在のエクスペリメントの名前を取得するには、IgorInfo(1) を使います。

フラグ

/Q[=*quiet*]　/Q を省略するか、静粛モードとして 0 を指定した場合、ExperimentInfo はコマンドウィンドウの履歴エリアに情報を出力します。

/Q または /Q=1 を指定した場合、ExperimentInfo は履歴エリアに何も出力しません。ただし、V_Flag と S_Value を通じて情報は引き続き返されます。

キーワード

getRequiredIgorVersion　V_Flag を通じて、現在のエクスペリメントを開くために必要な Igor Pro のバージョンを返します。S_Value を通じて、その要件を説明するメッセージを返します。

詳細は、以下の「必要な Igor バージョンの入手方法」のセクションを参照してください。

getLongNameUsage[={*dfRef*, *mask*, *options*, *length*}
現在のエクスペリメントで使われている長いオブジェクト名に関する情報を、V_Flag と S_Value を通じて返します。長いオブジェクト名（31 バイトを超えるもの）を使うエクスペリメントでは、Igor Pro 8.0 以降が必要となるため、この情報が参考になるかもしれません。

V_Flag には、検出された長い名称の数が設定されます。

S_Value には、検出された長い名前の説明が設定されます。/Q を省略するか、/Q=0 を指定した場合、その説明はコマンドウィンドウの履歴エリアにも表示されます。

すべてのパラメーターはオプションです。ただし、いずれかを指定する場合は、すべてを指定する必要があります。

dfRef は、開始点となるデータフォルダーを指定するもので、デフォルトは root: です。これは、開始点となるデータフォルダーへのフルパスまたは部分パスです。*dfRef* に root: または * を指定すると、デフォルト値が適用されます。: を指定すると、現在のデータフォルダーが使われます。

getLongNameUsage は、*dfRef* で指定されたデータフォルダーの内容について報告します。したがって、指定されたデータフォルダー自体の名前が長くても、そのフォルダー自体は報告されません。

mask は、どのタイプの長いオブジェクト名について情報を得たいかを指定します。これは、次のように定義されるビット単位のパラメーターです：

Bit 0： ウェーブ名
Bit 1： 変数名
Bit 2： データフォルダー名
Bit 3： ターゲットウィンドウ名
Bit 4： シンボリックパス名

mask のデフォルト値は 31 で、これは上記のすべてを指定します。

ビット設定の詳細は、ヘルプ Setting Bit Parameters を参照してください。

options は、返される情報のさまざまな側面をコントロールします。これは、次のように定義されるビット単位のパラメーターです：

Bit 0： 人間が読みやすい形式のテキストを生成する
Bit 1： ビット 0 がセットされている場合はヘッダーを含める
Bit 2： ウェーブ、変数、データフォルダーにフルパスを使う
Bit 3： 再帰的（データフォルダーを再帰的に処理する）

options のデフォルト値は 15 です。

ビット設定の詳細は、ヘルプ Setting Bit Parameters を参照してください。

length は、出力に含まれる名前がこれ以上の長さ（バイト単位）でなければならないことを指定します。
デフォルト値は 31 です。*length* に 0 を指定した場合、名前の長さに関係なく、すべてのウェーブが一覧表示されます。

詳細は、以下の「長い名前の使用状況情報の取得」のセクションを参照してください。

getLongDimensionLabelUsage[={*dfRef*, *mask*, *options*, *length*}
現在のエクスペリメントにおいて、長い次元ラベルを持つウェーブに関する情報を、V_Flag と S_Value を通じて返します。このようなエクスペリメントには Igor Pro 8.0 以降が必要となるため、この情報が参考になるかもしれません。

V_Flag には、長い次元ラベルを持つウェーブの数が入力されます。

S_Value には、検出されたウェーブの説明が設定されます。/Q を省略するか、/Q=0 を指定した場合、その説明はコマンドウィンドウの履歴エリアにも表示されます。

すべてのパラメーターはオプションです。ただし、いずれかを指定する場合は、すべてを指定する必要があります。

dfRef は、処理を開始するデータフォルダーを指定するもので、デフォルトは root: です。これは、開始データフォルダーへのフルパスまたは部分パスです。*dfRef* に root: または * を指定すると、デフォルト値が適用されます。: を指定すると、現在のデータフォルダーが使われます。

mask は、コンテキストを提供するために、長い次元ラベルを持つウェーブデータとともにデータフォルダーも一覧表示されるかどうかを決定します。これは、次のように定義されるビット単位のパラメーターです：

Bit 2：　　データフォルダー名を含む

他のすべてのビットは無視されます。

mask のデフォルト値は 4 です。これは、ビット 2 がセットされ、データフォルダーが一覧表示されることを意味しますが、これらが一覧表示されるのは、*options* パラメーターのビット 0 もセットされている場合に限られます。

ビット設定の詳細は、ヘルプ Setting Bit Parameters を参照してください。

options は、返される情報のさまざまな側面をコントロールします。これは、次のように定義されるビット単位のパラメーターです：

Bit 0：　　人間が読みやすい形式のテキストを生成する
Bit 1：　　ビット 0 がセットされている場合はヘッダーを含める
Bit 2：　　ウェーブ、変数、データフォルダーにフルパスを使う
Bit 3：　　再帰的（データフォルダーを再帰的に処理する）

options のデフォルト値は 15 です。

ビット設定の詳細は、ヘルプ Setting Bit Parameters を参照してください。

length は、出力にウェーブを含めるために、次元ラベルがこれ以上の長さ（バイト単位）でなければならないことを指定します。
デフォルト値は 31 です。*length* に 0 を指定した場合、次元ラベルの長さに関係なく、ウェーブが一覧表示されます。

必要な Igor バージョンの入手方法

キーワード 'getRequiredIgorVersion' は、V_Flag を通じて現在のエクスペリメントを開くために必要な Igor Pro のバージョンを返します。
特定のバージョンが要求されていない場合は、0 を返します。
V_Flag が 0 以外の場合、S_Value には、特定のバージョンの Igor Pro が必要な理由の説明が含まれます。

ここでいう「現在のエクスペリメントを開くために必要な Igor Pro のバージョン」という概念は、非常に限定的なものです。
getRequiredIgorVersion が V_Flag を通じて 0 以外の値を返す場合、これは、V_Flag で指定されたバージョンより古いバージョンの Igor Pro では、現在のエクスペリメントをまったく開くことができないことを意味します。
現在、getRequiredIgorVersion がフラグを立てる唯一の状況は、長いオブジェクト名、具体的には長いウェーブ、変数、データフォルダー、ターゲットウィンドウ、シンボリックパス名、あるいは長いウェーブの次元ラベルが使われている場合です。

エクスペリメントで長い名前や長い次元ラベルが使われている場合、そのエクスペリメントは Igor 8.0 以前のバージョンでは開くことができず、getRequiredIgorVersion は V_Flag を 8.00 に設定します。
背景情報は、ヘルプ Long Object Names と Long Dimension Labels を参照してください。

getRequiredIgorVersion が 0 を返す場合、そのエクスペリメントは Igor Pro の以前のバージョンでも開くことができることを意味します。
ただし、そのエクスペリメントがエラーなしで確実に開くという保証はありません。
例えば、プロシージャ内で Igor Pro 8.0 で追加された関数やコマンドを使っている場合、そのエクスペリメントは以前のバージョンでも開くことはできますが、コンパイルの時にエラーが発生します。
getRequiredIgorVersion は、特定のバージョンを必要とするすべての Igor Pro 機能の使用を検出するわけではありません。
この関数は、前段落で説明した特定の状況のみを検出します。

長い名前の使用状況情報の取得

長いオブジェクト名に関する背景情報は、ヘルプ Long Object Names を参照してください。

長いオブジェクト名（31 バイトを超えるもの）を使うエクスペリメントでは、Igor Pro 8.0 以降が必要です。
getLongNameUsage キーワードを指定して ExperimentInfo コマンドを実行することで、現在のエクスペリメントで長いオブジェクト名が使われているかどうか、また使われている場合はその内容を特定できます。
これは、以前のバージョンの Igor Pro でもエクスペリメントファイルを開けるようにエクスペリメントを修正したい場合に役立ちます。

キーワード getLongNameUsage は、長いウェーブ名、変数名、データフォルダー名、ターゲットウィンドウ名、シンボリックパス名に関する情報を報告します。
長い軸名、注釈名、コントロール名、特殊文字名、XOP 名、長い次元ラベルに関する情報は報告しません。

getLongNameUsage は、出力変数 V_Flag と S_Value を通じて情報を返します。
また、/Q を省略するか、/Q=0 を指定した場合は、コマンドウィンドウの履歴エリアに情報を出力します。

「人間が読みやすいモード」（options パラメーターセットのビット 0）で、出力にデータフォルダーを含める場合（*mask* パラメーターセットのビット 2）、ExperimentInfo は、フォルダー名が長いかどうかに関わらず、各データフォルダーごとに出力を生成し、インデントを使います。
これにより、データフォルダーの階層全体という文脈の中で、長いフォルダー名を確認することができます。

「人間が読みやすいモード」がオフの場合、ExperimentInfo は、指定されたデータフォルダーの名前が長い場合にのみ、そのフォルダーに対する出力を生成します。
このモードは、人間が読むためではなく、コードでの使用を目的としています。
このモードでは、フルパスを指定して呼び出した場合にのみ、階層構造（つまり、どのデータオブジェクトがどのデータフォルダーにあるか）を判別できます。

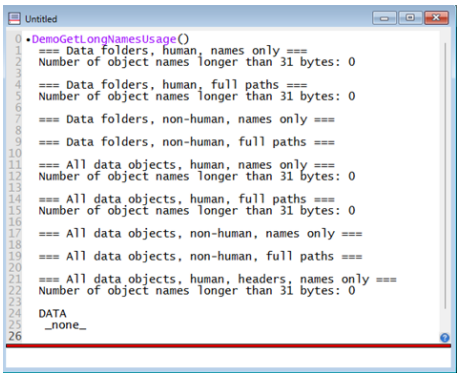
dfRef パラメーターを使って開始データフォルダーを指定し、かつ「人間が読みやすいモード」がオフになっている場合、ExperimentInfo は、開始データフォルダー名が長くても、そのフォルダー名に関する出力を生成しません。

フルパスを指定した場合（options パラメーターセットのビット 1）、リベラル名は引用符で囲まれて表示されますが、単純名のみを指定した場合は引用符で囲まれません。
単純名が引用符で囲まれない理由については、ヘルプ Accessing Global Variables and Waves Using Liberal Names で説明しています。

GetLongNameUsage の例

```
// すべてのデータオブジェクト（ウェーブ、変数、データフォルダー）、ターゲットウィンドウと
// シンボリックパスの長い名前を表示する
ExperimentInfo getLongNameUsage

// 長い名前の数のみを出力し、それ以外は出力しない
ExperimentInfo/Q getLongNameUsage; Print V_Flag
```

```
//  より専門的なテスト - 長い名前を使うエクスペリメントの Procedure ウィンドウに以下を貼り付け、
// DemoGetLongNamesUsage() を実行してください

Constant kLongNamesLength = 31
Constant kHumanMask = 1          // 人間が読みやすいモード
Constant kHeadersMask = 2        // ヘッダーを含む
Constant kFullPathsMask = 4      // フルパスを使う
Constant kRecursiveMask = 8      // 再帰的

Function DemoGetLongNamesUsage()
    Print "=== Data folders, human, names only ==="
    ExperimentInfo getLongNameUsage={*, 4, kHumanMask | kRecursiveMask, kLongNamesLength}
    Printf "\r"

    Print "=== Data folders, human, full paths ==="
    ExperimentInfo getLongNameUsage={*, 4, kHumanMask | kFullPathsMask | kRecursiveMask,
kLongNamesLength}
    Printf "\r"

    Print "=== Data folders, non-human, names only ==="
    ExperimentInfo getLongNameUsage={*, 4, 0 | kRecursiveMask, kLongNamesLength}
    Printf "\r"

    Print "=== Data folders, non-human, full paths ==="
    ExperimentInfo getLongNameUsage={*, 4, kFullPathsMask | kRecursiveMask, kLongNamesLength}
    Printf "\r"

    Print "=== All data objects, human, names only ==="
    ExperimentInfo getLongNameUsage={*, 7, kHumanMask | kRecursiveMask, kLongNamesLength}
    Printf "\r"

    Print "=== All data objects, human, full paths ==="
    ExperimentInfo getLongNameUsage={*, 7, kHumanMask | kFullPathsMask | kRecursiveMask,
kLongNamesLength}
    Printf "\r"

    Print "=== All data objects, non-human, names only ==="
    ExperimentInfo getLongNameUsage={*, 7, 0 | kRecursiveMask, kLongNamesLength}
    Printf "\r"

    Print "=== All data objects, non-human, full paths ==="
    ExperimentInfo getLongNameUsage={*, 7, kFullPathsMask | kRecursiveMask, kLongNamesLength}
    Printf "\r"

    Print "=== All data objects, human, headers, names only ==="
    ExperimentInfo getLongNameUsage={*, 7, kHumanMask | kHeadersMask | kRecursiveMask,
kLongNamesLength}
    // Printf "\r"      // ヘッダーモードでは末尾に空白行が出力されるため、これをスキップする
End
```



参照

ExperimentModified、IgorInfo コマンド
ヘルプ Long Object Names

ExperimentModified [*newModifiedState*]

ExperimentModified コマンドは、現在のエクスペリメントの修正（保存）状態を取得し、必要に応じて設定します。

このコマンドを使うと、保存する必要のない変更を加えた後に、現在のエクスペリメントを保存するかどうかを尋ねてくるのを防ぐことができます。
逆に、通常であれば保存の確認を行わない場合でも、強制的に保存の確認を行わせることもできます。

変数 V_flag は、常に ExperimentModified コマンドの実行前に有効だった「エクスペリメントによる変更」の状態に設定されます。
変更済みの場合は 1、変更されていない場合は 0 です。

文字列 S_Info には、前回の保存以降、または ExperimentModified 0 の実行以降に行われた、現在のエクスペリメントに対する変更点のリストが、常にセミコロン区切りで設定されます。
これは、"Do you want to save changes to experiment" というダイアログに表示されるリストと同じものです。

パラメーター

newModifiedState が存在する場合、エクスペリメントによって変更された状態を次のように設定します。

newModifiedState=0： 終了したり別のエクスペリメントを開いたりする前に、エクスペリメントの保存を求めるメッセージは表示されず、Save Experiment メニュー項目は無効になります。

newModifiedState=1： 終了したり別のエクスペリメントを開いたりする前に、エクスペリメントを保存するよう求められ、Save Experiment メニュー項目が有効になります。

詳細

コマンドラインで ExperimentModified 0 を実行しても、コマンドが履歴エリアにエコーされ、エクスペリメントが「変更済み」としてマークされてしまうため、機能しません。
履歴エリアにテキストをエコーしない関数やマクロ内でこのコマンドを使ってください。

例

/Q フラグは不可欠です。これにより、エクスペリメントが再度変更されたとみなされるような履歴エリアへの出力が行われなくなります。

```
Menu "Modified", dynamic
    // "Save Experiment" を有効にし、V_Flag と S_Info を設定する
    "Mark Experiment Modified",/Q,ExperimentModified 1
    // "Save Experiment" を無効にする
    "Mark Experiment Saved",/Q,ExperimentModified 0
    "-"
    ShowModifiedItems(),/Q, ;           // なにもしない
End

Function/S ShowModifiedItems()
    String dnm= \\M0:1(:                // メタ文字なしを無効にする
    ExperimentModified
    if( V_Flag )
        return dnm+ReplaceString(";", S_info, ";"+dnm)
```

```
    endif
    return dnm+"Not Modified"
End
```

参照

SaveExperiment、ExperimentInfo コマンド
ヘルプ Menu Definition Syntax

ExportGizmo [*flags*] *keyword* [=*value*]

ExportGizmo コマンドは非推奨となっていますが、部分的な下位互換性を確保するため、引き続きサポートされています。

フラグ

/N=*gizmoName* 対象となる Gizmo ウィンドウの名前を指定します。/N を省略した場合、最前面にある Gizmo ウィンドウに対して処理を行います。

/P=*pathName* *pathName*（既存のシンボリックパスの名前）で指定されたフォルダーにファイルを保存します。

/Z エラーを抑制します。

キーワード

clip 現在の画像をクリップボードにコピーします。コピーは現在の解像度倍率で行われ、TIFF 形式で保存されます。

EPS Gizmo ウィンドウでは、EPS はサポートされなくなりました。

PDF Gizmo ウィンドウでは、PDF はサポートされなくなりました。

print 印刷ダイアログを表示せずに、Gizmo ウィンドウをデフォルトのプリンタで印刷します。

wave=*wName* 現在の画像を、*wName* で指定された 3D 画像ウェーブとして、解像度 1x で保存します。Gizmo ウィンドウが表示されている必要があります。Gizmo 画像に透明部分が含まれている場合、結果として生成されるウェーブは、符号なしバイトの４レイヤー RGBA ウェーブになります。それ以外の場合は、ウェーブは符号なしバイトの３レイヤー RGB ウェーブになります。

参照

ModifyGizmo、NewGizmo、AppendToGizmo、GetGizmo、GizmoInfo、RemoveFromGizmo コマンド
ヘルプ 3D Graphics

Extract [/FREE /INDX /O][*typeFlags*] *srcWave*, *destWave*, *LogicalExpression*

Extract コマンドは、LogicalExpression の評価結果が TRUE となる箇所において *srcWave* 内のデータを検索し、一致するデータを順次 *destWave* に格納します。
destWave は、まだ存在しない場合は作成されます。

パラメーター

srcWave はウェーブの名前です。

destWave は、結果が格納される新規または既存のウェーブの名前です。

LogicalExpression では、式の中で任意の比較演算子や論理演算子を使用できます。

フラグ

/FREE　　フリーの *destWave* を作成します（ヘルプ Free Waves を参照）。

/FREE は関数内でのみ使用可能であり、かつ *destWave* に対して単純な名前またはウェーブ参照構造体のフィールドが指定されている場合に限ります。

/INDX　　*srcWave* のデータではなく、インデックスを *destWave* に格納します。

/O　　*destWave* を *srcWave* と同じにする（ソースを上書きする）ことができます。

型フラグ（関数のみ）

Extract では、ユーザー関数内でさまざまな型フラグを使って、宛先のウェーブ参照変数の型を指定することもできます。
これらの型フラグは、同名の別のウェーブ参照変数と一致させる必要がある場合や、ウェーブ代入に対してどのような式をコンパイルすべきかを指示する必要がある場合を除き、使う必要はありません。
型フラグの完全な一覧と詳細は、ヘルプ WAVE Reference Types と WAVE Reference Type Flags を参照してください。

詳細

srcWave は、テキストを含む任意の型である可能性があります。

destWave は *srcWave* と同じ型ですが、常に 1 次元です。
/INDX を指定すると、*destWave* の型は 32 ビットの符号なし整数に設定され、その値は *srcWave* の次元数に関係なく、*srcWave* 内の線形インデックスを表します。

例

```
Make/O source= x
Extract/O source,dest,source>10 && source<20
print dest
```

は、履歴エリアに以下を出力します：

```
dest[0]= {11,12,13,14,15,16,17,18,19}
```

参照

Duplicate コマンド

FastGaussTransform [/AERR=*aprxErr*, /WDTH=*h*, /OUTW=*locWave*, /OUT1={*x1*,*nx*,*x2*}, /OUT2={*x1*,*nx*,*x2*,*y1*,*ny*,*y2*}, /OUT3={*x1*,*nx*,*x2*,*y1*,*ny*,*y2*,*z1*,*nz*,*z2*}, /Q, /RX=*rx*, /RY=*ry*, /TET=*nTerms*, /Z] *srcLocationsWave*, *srcWeightsWave*

FastGaussTransform コマンドは、次式で与えられる離散ガウス変換を計算するための効率的なアルゴリズムを実装しています。

$$G(y_j) = \sum_{i=0}^{N-1} q_i \exp\left(-\frac{\left\| y_j - x_i \right\|^2}{h} \right),$$

ここで、G は M 次元のベクトル、y は観測位置を表す N 次元のベクトル、$\{q_i\}$ は M 次元の重み、$\{x_i\}$ は発生源の位置を表す N 次元のベクトル、h はガウス幅です。
ウェーブ M_FGT には、現在のデータフォルダー内の出力が格納されています。

フラグ

/AERR=*aprxErr* 近似誤差を設定します。この値によって、計算においてガウス関数のテイラー展開の項をいくつ使うかが決まります。デフォルト値は 1e-5 です。

/WDTH=*h* ガウスの幅を設定します。デフォルト値は 1 です。

/OUTW=*locWave* 出力が計算される位置を指定します。*locWave* の列数は *srcLocationsWave* と同じでなければなりません。その他の /OUT フラグは相互に排他的であるため、一度に 1 つだけを使ってください。

/OUT1={*x1*,*nx*,*x2*}
/OUT2={*x1*,*nx*,*x2*,*y1*,*ny*,*y2*}
/OUT3={*x1*,*nx*,*x2*,*y1*,*ny*,*y2*,*z1*,*nz*,*z2*}
指定した次元のグリッド形式の出力を指定します。いずれの場合も、その次元の開始値と終了値、区間の数を設定します。入力ソースの次元と一致しない出力を指定することはできません。

/Q 履歴エリアに結果が表示されません。

/RX=*rx* クラスターの最大半径を設定します。最大半径が *rx* 未満になると、クラスタリングアルゴリズムは終了します。/RX を指定しない場合、最大半径は検出された最大半径と同じになります。

/RY=*ry* クラスターが変換値に寄与する、観測ポイントとクラスタ中心との距離の上限を設定します。*ry* のデフォルト値は、/WDTH フラグで指定されたガウス幅の 4 倍です。

/TET=*nTerms* テイラー展開の項数を設定します。/TET を使うと、項数を設定するとともに、近似誤差値（/AERR）から推定されるデフォルトの誤差推定値をバイパスできます。

/Z エラーの報告はありません。

詳細

離散ガウス変換は、直和として計算することができます。
厳密な計算が実用的となるのは、光源や観測点の数がある程度少なく、空間次元が低い場合に限られます。
次元が高くなり、光源の数が増えるにつれて、ガウス関数のいくつかの性質を活用する方が効率的です。
FastGaussianTransform コマンドは、これを 2 つの方法で実現します。
まず、遠方のクラスターに属するすべての光源点の寄与を計算する必要がないように、光源を N 次元の空間クラスターに整理します（FPClustering コマンドを参照）。
アルゴリズムの 2 つ目の構成要素は、和を「ソースポイントのみに依存する因子」と「観測ポイントのみに依存する因子」に因数分解する近似手法です。
ソースポイントのみに依存する因子は一度だけ計算されるのに対し、観測ポイントのみに依存する因子は各観測ポイントごとに一度ずつ評価されます。

計算効率と精度のバランスは、複数のパラメーターを使って調整することができます。
デフォルトでは、このコマンドはガウス関数のテイラー展開に使う項の数を自動的に算出します。
/AERR を使ってデフォルトの近似誤差値を修正するか、/TET を使って展開の項数を直接設定することができます。

FastGaussianTransform は、ウェーブの最大許容次元を超える次元での計算をサポートしています。
したがって、*srcLocationsWave* は 2D の実数型（単精度または倍精度）のウェーブでなければなりません。
このウェーブでは、各行が 1 つのソース位置に対応し、各列が各次元の成分を表します（たとえば、トリプレッ

トウェーブは 3D のソース位置を表します）。
srcWeightsWave は *srcLocationsWave* と同じ行数を持つ必要があり、実数の単精度または倍精度のウェーブである必要があります。
ほとんどのアプリケーションでは、*srcWeightsWave* は単一の列を持つため、出力 G はスカラーになります。
ただし、*srcWeightsWave* が複数の列を持つ場合、G はベクトルになります。
これは、複数の係数セットを一度にテストする必要がある場合に便利です。
/OUTW を使って観測ポイントを指定する場合、*locWave* は *srcLocationsWave* と同じ列数でなければなりません（出力の行数は任意です）。
このコマンドでは、ウェーブのスケーリングはサポートされていません。

参照

CWT、FFT、ImageInterpolate、Loess、FPClustering コマンド

参考文献

Yang, C., R. Duraiswami, and L. Davis, Efficient Kernel Machines Using the Improved Fast Gauss Transform, Advances in Neural Information Processing Systems, 16, 2004.

```
FastOp destWave = prod1 [± prod2  [± prod3]]
```

```
FastOp destWave += prod1 [± prod2]
```

FastOp コマンドを使うと、特定のウェーブ代入文の実行速度を向上させることができます。
この構文は、構文要件を満たす任意のウェーブ代入文の前に FastOp を挿入するだけでよいように設計されています。
「+=」構文は、Igor Pro 9.0 で追加されました。

パラメーター

destWave　代入式に対する既存の宛先ウェーブです。ウェーブの長さがすべて同じでない場合、または型が数値型でない場合は、実行時にエラーが発生します。

prod1, *prod2*, *prod3*　以下の形式の生成物：

```
constexpr * wave1 * wave2
```

または

```
constexpr * wave1 / wave2
```

constexpr は、リテラル数値定数、または括弧で囲まれた定数式であることができます。このような式は一度だけ評価されます。

prod 式の各構成要素は、どれでも省略可能です。

フラグ

/C　Igor Pro 9.0 以降では、/C フラグは非推奨となっており、無視されます。浮動小数点ウェーブの場合、式の種類（実数または複素数）は、宛先となるウェーブの型によって決定されます。

Igor Pro 8 およびそれ以前のバージョンでは、複雑な宛先ウェーブを使う場合、/C が必要です。

FastOp は、浮動小数点ウェーブについては実数および複素数ウェーブをサポートしていますが、整数ウェーブについては実数のみをサポートしています。

詳細

FastOp は、浮動小数点ウェーブについては実数と複素数ウェーブをサポートしています。
整数ウェーブについては、実数のみをサポートしています。

ウェーブが複雑な場合は、ユーザー定義関数内でウェーブ参照を宣言する際に、必ず WAVE/C を使ってください。

以下の表に示す特定の組み合わせについては、より汎用的ではありますが処理速度の遅い汎用コードではなく、高速化された最適化コードを使って評価されます。
SP は「単精度浮動小数点」、DP は「倍精度浮動小数点」を意味します。

記述	**最適化されるデータ型**
dest = 0	すべて
dest = waveA	すべて
dest += C0 * waveA	SP と DP、実数と複素数
dest = dest + C0 * waveA	SP と DP、実数と複素数
dest = dest + waveA * C0	SP と DP、実数と複素数
dest = C0	SP と DP、整数、読み取り専用
dest = waveA + C1	SP と DP、整数、読み取り専用
dest = C0 * waveA + C1	SP と DP、読み取り専用
dest = waveA + waveB + C2	整数のみ、読み取り専用

上記の式において、プラスはマイナスとなる場合があり、末尾の定数（C0、C1、C2）は省略可能です。

注記

整数ウェーブに対して整数コマンドを使う上記に挙げた最適化されるケースを除き、整数の結果は倍精度の中間値を使って評価されます。
これにより、64 ビット整数ウェーブの精度は 53 ビットに制限されます。

処理速度の向上は、一般的に、FastOp を削除した同等の記述と比較して 10 倍から 40 倍の範囲です。

例

有効な記述：

```
FastOp dest = 3
FastOp dest = waveA + waveB
FastOp dest = 0.5*waveA + 0.5*waveB
FastOp dest = waveA*waveB
FastOp dest = (2*3)*waveA + 6
FastOp dest = (locvar)*waveA
FastOp dest += waveA + waveB
```

無効な記述：

```
FastOp dest = 3*4
FastOp dest = (waveA + waveB)
FastOp dest = waveA*0.5 + 0.5*waveB
FastOp dest = waveA*waveB/2
FastOp dest = 2*3*waveA + 6
FastOp dest = locvar*waveA
FastOp dest += waveA + waveB + waveC
```

参照

MatrixOp、MatrixMultiply、MatrixMultiplyAdd コマンドを使うことで、さらに効率的な行列演算が可能になります。

MultiThread コマンド

FBinRead [/B[=*b*]/F=*f*] *refNum*, *objectName*

FBinRead コマンドは、*refNum* で指定されたファイルからバイナリデータを読み取り、指定されたオブジェクトに格納します。

バイナリデータを数値ウェーブに読み込むという単純な用途であれば、GBLoadWave コマンドの方が実装が簡単かもしれません。

パラメーター

refNum は、ファイルを開くために実行された Open コマンドから取得されたファイル参照番号です。

objectName は、数値変数、文字列変数、構造体、またはウェーブの名前です。

フラグ

/B[=*b*] ファイルのバイト順を指定します。

b=0： ネイティブ（/B がないときと同じ）
b=1： 逆順（/B と同じ）
b=2： ビッグエンディアン（Motorola）
b=3： リトルエンディアン（Intel）

/F=*f* 読み込むバイト数と、そのバイトの解釈方法をコントロールします。

f=0： オブジェクトのネイティブバイナリ形式（デフォルト）
f=1： 符号付き 1 バイト整数
f=2： 符号付き 16 ビットワード。2 バイト
f=3： 符号付き 32 ビットワード。4 バイト
f=4： 32 ビット IEEE 浮動小数点。4 バイト
f=5： 64 ビット IEEE 浮動小数点。8 バイト
f=6： 64 ビット整数。8 バイト。Igor Pro 7.0 以降が必要

/U 整数の形式（/F=1、2、3、または 6）は符号なしです。/U を省略した場合、整数は符号付きとなります。

詳細

objectName が文字列変数の名前である場合、/F は適用されません。
読み込まれたバイト数は、FBinRead コマンドが呼び出される前の文字列に含まれるバイト数です。
PadString 関数を使って、文字列のサイズを設定することができます。

FBinRead が数値変数に対して使うバイナリ形式は、/F フラグによって決まります。
/F を省略した場合、変数のネイティブデータ型（8 バイトの倍精度浮動小数点型）が使われます。
したがって、実数型変数に読み込む場合、/F フラグの設定に応じて、FBinRead はファイルから 1、2、4、または 8 バイトを読み取り、必要に応じてそれらのバイトを倍精度浮動小数点数に変換し、その結果の値を変数に格納します。
複素数型変数に読み込む場合、この処理は実部と虚部についてそれぞれ 1 回ずつ、計 2 回繰り返されます。

実数ウェーブの読み取りは、実数変数の読み取りと同様に機能しますが、実数ウェーブには複数の要素があり、各要素はウェーブのデータ型に応じて 1、2、4、または 8 バイトであるという点だけが異なります。

実数ウェーブの各要素について、FBinRead は /F またはウェーブのネイティブデータ型によって指定されたバイト数を読み取り、必要に応じてそれらのバイトをウェーブのデータ型に変換し、その結果の値を対応するウェーブ要素に格納します。
複素数ウェーブに読み込む場合、この処理は２回繰り返されます。
１回目は各要素の実部に対して、２回目は虚部に対して行われます。

構造体の読み込みは異なります。/F フラグは効果を持ちません。
FBinReads は、構造体を埋めるために必要なバイト数を読み込みますが、そのバイト数は個々のフィールドのサイズや、２バイト単位で構造体をアラインメントしているという事実によって決まります。
ファイルから構造体へバイトが読み込まれた後、/B フラグを指定すると、FBinRead は個々のフィールドのバイト順を入れ替えます。

FBinRead コマンドは多次元性を認識しません。
詳細については、ヘルプ Multidimensional Waves、特にヘルプ Analysis on Multidimensional Waves を参照してください。

参照

FBinWrite、Open、FGetPos、FSetPos、FStatus、GBLoadWave コマンド

FBinWrite [/B[=*b*]/F=*f*] *refNum*, *objectName*

FBinWrite コマンドは、指定されたオブジェクトをバイナリ形式でファイルに書き込みます。

パラメーター

refNum は、ファイルを開くために実行された Open コマンドから取得されたファイル参照番号です。

objectName は、数値変数、文字列変数、構造体、またはウェーブの名前です。

フラグ

/B[=*b*] ファイルのバイト順を指定します。

b=0： ネイティブ（/B がないときと同じ）
b=1： 逆順（/B と同じ）
b=2： ビッグエンディアン（Motorola）
b=3： リトルエンディアン（Intel）

/F=*f* 読み込むバイト数と、そのバイトの解釈方法をコントロールします。

f=0： オブジェクトのネイティブバイナリ形式（デフォルト）
f=1： 符号付き 1 バイト整数
f=2： 符号付き 16 ビットワード。2 バイト
f=3： 符号付き 32 ビットワード。4 バイト
f=4： 32 ビット IEEE 浮動小数点。4 バイト
f=5： 64 ビット IEEE 浮動小数点。8 バイト
f=6： 64 ビット整数。8 バイト。Igor Pro 7.0 以降が必要

/P データに IgorBinPacket を追加します。これはプログラム間通信 (PPC) の結果パケット（*refNum*=0）で使われていましたが、通常、ファイルへの書き込み時には使われません。

/U 整数の形式（/F=1、2、または 3）は符号なしです。/U を省略した場合、整数は符号付きとなります。

詳細

refNum の値が 0 の場合は、プログラム間通信（PPC）と ActiveX オートメーション（Windows）と組み合わせて使われます。
通常はファイルに書き込まれるはずのデータは、PPC または ActiveX オートメーションの結果パケットに追加されます。

オブジェクトが文字列変数である場合、/F は適用されません。
書き込まれるバイト数は、その文字列のバイト数と同じになります。

FBinWrite が数値変数やウェーブに対して使うバイナリ形式は、/F フラグの設定によって異なります。
/F フラグが指定されていない場合、FBinWrite は指定されたオブジェクトのネイティブバイナリ形式を使います。

バイト順とは、マルチバイトデータがファイルに書き込まれる順序を指します。
例えば、16 ビットのワード（「ショート」と呼ばれることもあります）は、上位バイトと下位バイトで構成されます。
ビッグエンディアンのバイト順では、上位バイトが先にファイルに書き込まれます。
リトルエンディアンのバイト順では、下位バイトが先にファイルに書き込まれます。

FBinWrite は、構造体全体をディスクファイルに書き込みます。
/B フラグが指定された場合、構造体の個々のフィールドのバイト順が反転します。

FBinWrite コマンドは多次元性を認識しません。
詳細は、ヘルプ Multidimensional Waves、特にヘルプ Analysis on Multidimensional Waves を参照してください。

参照

FBinRead、Open、FGetPos、FSetPos、FStatus、GBLoadWave コマンド

FFT [*flags*] *srcWave*

WaveMetrics Web サイトの日本語のコマンドのヘルプセクションにある別ファイルを参照してください。

FGetPos *refNum*

FGetPos コマンドは、ファイルの位置を返します。

ファイルの位置のみを知りたい場合、FGetPos は FStatus よりも高速な代替手段となります。

FGetPos コマンドは、Igor Pro 7.0 で追加されました。

パラメーター

refNum は、Open コマンドによって取得されたファイル参照番号です。

詳細

FGetPos は、理論上、最大で約 4.5E15 バイトの長さを持つ非常に大きなファイルに対応しています。

FGetPos は、以下の変数を設定します。

V_flag　　*refNum* が有効な場合、0 以外（true）を返します。

V_filePos　　ファイルの先頭からの現在のファイル位置（バイト単位）です。

参照

Open、FSetPos、FStatus コマンド

FIFO2Wave [/R=[*startPoint*,*endPoint*]/S=*s*] *FIFOName*, *channelName*, *waveName*

FIFO2Wave コマンドは、指定された FIFO の指定されたチャネルから、指定されたウェーブへ FIFO データをコピーします。

フラグ

/R=[startPoint,*endPoint*]
指定された FIFO ポイントをウェーブに書き込みます。

/S=*s*
s=0： /S を指定しない場合と同じです。
s=1： ウェーブの X 軸スケーリング値 x0 を、FIFO 内の最初のサンプルの番号に設定します。
s=2： ウェーブの数値型を、FIFO チャネルの型に合わせて変更します。
s=3： *s*=1 と *s*=2 を組み合わせたものです。

詳細

FIFO2Wave が動作するには、FIFO が有効な状態である必要があります。
NewFIFO を使って FIFO を作成すると、初期状態では無効です。
CtrlFIFO コマンドを使って開始コマンドを発行すると、有効になります。
CtrlFIFO を使って FIFO パラメーターを変更するまで、有効な状態が維持されます。

/R=[*startPoint*,*endPoint*] を使って FIFO データポイントの範囲を指定すると、FIFO2Wave は *startPoint* と *endPoint* を有効なポイント番号にクリップした後、指定された FIFO ポイントをウェーブに出力します。

有効なポイント数は、FIFO が実行中かどうか、FIFO がファイルに接続されているかどうかに依存します。
FIFO が実行中の場合、*startPoint* と *endpoint* は FIFO 内のポイント数に合わせて切り詰められます。
FIFO がファイルをバッファリングしている場合、その範囲にはファイルの全範囲を含めることができます。

範囲を指定しない場合、FIFO2Wave は、最も最近取得された FIFO データをウェーブに転送します。
転送されるデータポイントは、FIFO 内のデータポイント数とウェーブ内のデータポイント数のうち、少ない方の数となります。

FIFO2 ウェーブは、現在の X スケーリングと /S フラグの設定に応じて、ウェーブの X スケーリングや数値の型を変更する場合もあれば、変更しない場合もあります。

ウェーブの X 方向のスケーリングは、x0 と Δx の 2 つの値によってコントロールされると考えてください。
ここで、ポイント p の X 座標は x0 + p*Δx となります。
FIFO2Wave は、ウェーブの Δx 値を常に FIFO の deltaT 値（CtrlFIFO コマンドで設定された値）と同じに設定します。
/S フラグを指定しない場合、FIFO2Wave はウェーブの x0 値を設定せず、ウェーブの数値型も設定しません。

FIFO2Wave を使ってグラフ内のウェーブを可能な限り高速に更新する場合、/S=0 フラグを設定すると最高の更新レートが得られます。
その他の /S 値を設定すると、再計算が増え、更新速度が低下します。

ウェーブ（FIFO チャネルに合わせて変更された場合など）の数値型が浮動小数点型である場合、FIFO2 ウェーブは、FIFO データをウェーブに転送する前に、次のようにスケーリングを行います。

```
scaled_value = (FIFO_value - offset) * gain
```

FIFO チャネルのゲインが 1 で、オフセットが 0 の場合、スケーリングは影響を与えないため、FIFO2Wave はこれをスキップします。

指定された FIFO チャネルがイメージストリップチャネル（NewFIFOChan のオプションパラメーター vectPnts を使って定義されたもの）である場合、結果として得られるウェーブは、行数が vectPnts で設定さ

れ、列数が前述の１次元ウェーブにおけるポイント数によって設定される行列となります。
チャート内の対応するチャンネルと同じ外観の画像プロットを作成するには、ウェーブを転置する必要があります（MatrixTranspose コマンド）。

参照

FIFO とデータ収集に関する説明に関しては、ヘルプ FIFOs and Charts を参照してください。

SetDrawEnv、SetDrawLayer コマンド

ウェーブや X スケーリングに関する説明は、ヘルプ The Waveform Model of Data を参照してください。

FIFOStatus [/Q] *FIFOName*

FIFOStatus コマンドは、FIFO とそのチャネルに関する各種情報を返します。
FIFO はデータ取得に使われます。

フラグ

/Q　　履歴エリアに S_Info 文字列が表示されません。

詳細

FIFOStatus は、指定された名前の FIFO が存在する場合、変数 V_flag を 0 以外に設定します。
指定された FIFO が存在する場合、FIFOStatus はその FIFO に関する情報を以下の変数に格納します。

V_FIFORunning　　FIFO が実行中の場合、0 以外となります。

V_FIFOChunks　　これまでに FIFO に格納されたデータチャンクの数です。

V_FIFOnchans　　FIFO 内のチャネル数です。

S_Info　　キーワードをパックした情報の列です。

キーワードをパックした情報文字列は、次のような形式のセクションが連続して構成されています：keyword:value

NumberByKey 関数や StringByKey 関数を使うと、キーワードをパックした文字列から値を取り出すことができます。
S_Info のキーワードは以下の通りです。

キーワード	型	意味
DATE	数字	CtrlFIFO を通して開始指示が出された日時
DELTAT	数字	CtrlFIFO で設定された FIFO の deltaT 値
DISKTOT	数字	FIFO のファイルに書き込まれたチャンクの現在の数
FILENUM	数字	CtrlFIFO で設定された出力ファイルの refNum またはレビューファイルの refNum。FIFO がどのファイルにも接続されていない場合、これは 0 になる
NOTE	文字列	CtrlFIFO で設定された FIFO のノート文字列
VALID	数字	FIFO が無効な場合は 0
DATATYPE	数字	NewFIFOCHAN/Y=(*numType*) で設定されたかのように、チャネルのデータ型を指定する。ここで、*numType* は WaveType 関数によって返される値

さらに、FIFOStatus は、FIFO 内の各チャネルについて、S_Info にフィールドを書き込みます。
フィールドのキーワードは、そのフィールドと、それが参照するチャネルを識別するための名前と番号の組み合わせです。
例えば、チャネル 4 の名前が “Pressure” の場合、S_Info 文字列には次のように表示されます：
NAME4:Pressure

次の表では、チャネルの番号は「#」で表されています。

キーワード	型	意味
FSMINUS#	数字	NewFIFOChan で設定された、チャネルのフルスケール値からのマイナス値
FSPLUS#	数字	NewFIFOChan で設定されたチャネルのプラス側のフルスケール値
GAIN#	数字	NewFIFOChan で設定されたチャネルのゲイン値
NAME#	文字列	チャネル名
OFFSET#	数字	NewFIFOChan によって設定されたチャネルのオフセット値
UNITS#	文字列	NewFIFOChan で設定されたチャネルの単位

参照

NumberByKey、StringByKey コマンド

ヘルプ FIFOs and Charts

FilterFIR [/COEF[=*coefsWaveName*] /DIM=*d* /E=*endEffect* /ENDV={*sv*[,*ev*]} /HI={*f1*, *f2*, *n*} /LO={*f1*, *f2*, *n*} /NMF={*fc*, *fw* [, *eps*, *nMult*]} /WINF=*windowKindName*] *waveName* [, *waveName*]...

WaveMetrics Web サイトの日本語のコマンドのヘルプセクションにある別ファイルを参照してください。

FilterIIR [/CASC /COEF[=*coefsWaveName*] /DIM=*d* /ENDV=*ev* /HI=*fHigh* /LO=*fLow* /N={*fNotch*, *notchQ*} /ORD=*order* /Z=*z* /ZP] [*waveName* , ...]

WaveMetrics Web サイトの日本語のコマンドのヘルプセクションにある別ファイルを参照してください。

FindContour [/DSTX=*destXWave* /DSTY=*dstYWave*] *matrixWave*, *level*

FindContour コマンドは、matrixWave=*level* の解の軌跡を表す XY ウェーブのペアを生成します。

FindContour コマンドは、Igor Pro 7.0 で追加されました。

フラグ

/DSTX=*destX* 指定された宛先ウェーブに、出力 X データを保存します。宛先ウェーブが存在しない場合は作成され、すでに存在する場合は上書きされます。

/DSTY=*destY* 指定された宛先ウェーブに、出力 Y データを保存します。宛先ウェーブが存在しない場合は作成され、すでに存在する場合は上書きされます。

詳細

FindContour は、等高線追跡アルゴリズムを用いて、*matrixWave=level* の解の軌跡を表す一対のウェーブを生成します。

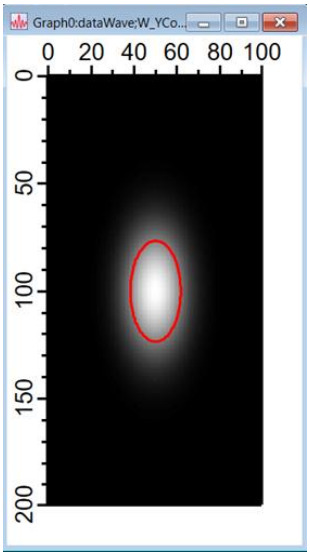
/DSTX を省略した場合、出力 X データは現在のデータフォルダー内の W_XContour に書き込まれます。

/DSTY を省略した場合、出力 Y データは現在のデータフォルダー内の W_YContour に書き込まれます。

出力ウェーブは倍精度浮動小数点数として記述されます。
異なる連続した解の点を区別するために、NaN が使われます。

例

```
Make/N=(100,200) dataWave = 1e4*gauss(x,50,10,y,100,20)
FindContour dataWave,4      // dataWave=4 の解を求める
NewImage dataWave
AppendToGraph/T W_YContour vs W_XContour
```



参照

AppendMatrixContour、ContourZ コマンド

FindDuplicates [*flags*] *srcWave*

FindDuplicates コマンドは、ウェーブ内の重複値を特定し、必要に応じてさまざまな出力ウェーブを作成します。
srcWave は、数値（実数または複素数）またはテキストのいずれかです。

srcWave が数値の場合、/DN、/INDX、/RN、/SN フラグは、以下に説明する通りウェーブを生成します。
これらのフラグをすべて省略した場合、FindDuplicates はデフォルトで /INDX フラグを使います。

srcWave がテキストの場合、/DT、/INDX、/RT、/ST フラグは、以下に説明する通りウェーブを生成します。
これらのフラグをすべて省略した場合、FindDuplicates はデフォルトで /INDX フラグを使います。

FindDuplicates コマンドは、Igor Pro 7.0 で追加されました。

フラグ

/FREE　すべての出力ウェーブをフリーウェーブとして生成します。
/FREE フラグは Igor Pro 8.0 で追加されました。

/FREE はユーザー定義関数でのみ使用できます。/FREE を使う場合、すべての出力ウェーブパラメーターは、パスや $ 式ではなく、単純な名前でなければなりません。

フリーウェーブに関する詳細は、ヘルプ Free Waves を参照してください。

/INDX=*indexWave*　検出された重複値のインデックスを含む数値出力ウェーブを作成します。このインデックスは、*srcWave* 内で重複値が検出された位置の番号です。このフラグは、数値入力とテキスト入力の両方に適用されます。

/INDX は、出力ウェーブを指定するフラグが指定されていない場合のデフォルトの動作です（数値ウェーブの場合は /DN、/RN、/SN、または /INDX、テキストウェーブの場合は /DT、/RT、/ST、または /INDX）。

/Z　　エラーを出力しません。

数値ソースウェーブのフラグ

/DN=*dupsWave*　　重複を含む数値ウェーブを作成します。

/RN=*dupsRemovedWave*
　　重複をすべて除去したソースデータを含む数値ウェーブを作成します。

/SN=*replacement*　　すべての重複値を *replacement* に置き換えた数値ウェーブを作成します。
replacement には、NaN、INF、複素数を含む任意の数値を指定できます。

　　/SNDS フラグを使って別の出力ウェーブを指定しない限り、出力ウェーブは現在のデータフォルダー内の W_ReplacedDuplicates となります。

/SNDS=*dupsReplacedWave*
　　/SN によって生成される出力ウェーブを指定します。/SNDS を省略した場合、/SN によって作成される出力ウェーブは、現在のデータフォルダー内の W_ReplacedDuplicates となります。/SN を指定せずに /SNDS のみを指定しても、何の効果もありません。

/TOL=*tolerance*　　単精度と倍精度の数値ウェーブに対する許容誤差値を指定します。

　　以下の場合、 2 つの値は重複しているとみなされます。

```
abs(value1-value2) <= tolerance
```

　　デフォルトでは、許容誤差はゼロです。

/UN=*uniqueNumbersWave*
　　srcWave に含まれる一意の数値を、小さい順に並べ替えた数値ウェーブを作成します。*srcWave* に NaN の要素が含まれている場合、それらは *uniqueNumbersWave* の最後のデータポイントとして扱われます。

　　/UN は、*srcWave* 内のエントリの順序を維持する /RN と互換性がありません。

　　/UN フラグは Igor Pro 9.0 で追加されました。

/UNC=*uniqueCounts*
　　/UN フラグによって作成された *uniqueNumbersWave* 内の各エントリの個数を格納した数値ウェーブを作成します。

　　/UNC フラグは、Igor Pro 9.0 で追加されました。

テキストソースウェーブのフラグ

/CI　　ASCII 文字のみを対象に、大文字と小文字を区別しないテキスト比較を行います。例えば、「A」と「a」は同一とみなされます。

　　/CI フラグは、Igor Pro 9.0 で追加されました。

/DT=*dupsWave*　　重複を含むテキスト出力ウェーブを作成します。

/LOC　　ASCII 文字と非 ASCII 文字の両方について、大文字と小文字の区別を考慮した、ロケールに応じたテキスト比較を実行します。例えば、非 ASCII 文字の「Å」と「å」は、ASCII 文字の「A」と「a」と同様に、同一とみなされます。

　　/CI も指定しない限り、/LOC は無視されます。

　　/LOC フラグは、Igor Pro 9.0 で追加されました。

/RT=*dupsRemovedWave*
重複をすべて除去したソースデータを含むテキスト出力ウェーブを作成します。

/ST=*replacementStr*
すべての重複部分を *replacementStr* に置き換えたテキストウェーブを作成します。*replacementStr* には、"" を含む任意のテキスト値を指定できます。

/STDS フラグを使って別の出力ウェーブを指定しない限り、出力ウェーブは現在のデータフォルダー内の T_ReplacedDuplicates となります。

/STDS=*dupsReplacedWave*
/ST によって生成される出力ウェーブを指定します。/STDS を省略した場合、/ST によって作成される出力ウェーブは、現在のデータフォルダー内の T_ReplacedDuplicates となります。/ST を指定せずに /STDS のみを指定しても、何の効果もありません。

詳細

FindDuplicates は *srcWave* をスキャンし、重複する値を特定します。
各値の最初の出現は重複とは見なされません。
重複とは、整数型やテキスト型のウェーブの場合のように値が完全に同一である場合、あるいは単精度または倍精度の数値型ウェーブの場合のように、指定された許容範囲内にある値のことです。

/CI または /CI/LOC を使わない限り、テキストの比較では大文字と小文字が区別されます。

このコマンドにより、上記の各種フラグで指定されたウェーブに対するウェーブ参照が作成されます。
詳細は、ヘルプ Automatic Creation of WAVE References を参照してください。

例

```
Function DemoFindDuplicates(mode)
    int mode            // 0=大文字・小文字を区別; 1=/CI; 2=/CI/LOC

    Make/O/T sourceText={"A","a", "Å","å", "B","b"}
    switch(mode)
        case 0:         // 大文字・小文字を区別する
            // {"A","a","Å","å","B","b"} を返す
            FindDuplicates/FREE/RT=output sourceText
            break
        case 1:         // ASCII のみ大文字・小文字を区別しない
            // {"A","Å","å","B"} を返す
            FindDuplicates/FREE/RT=output/CI sourceText
            break
        case 2:         // 大文字小文字を区別せず、ロケールに対応
            // {"A","Å","B"} を返す
            FindDuplicates/FREE/RT=output/CI/LOC sourceText
            break
    endswitch

    Print sourceText
    Print output
End
```

参照

FindLevels、FindValue、Sort、TextHistogram コマンド

FindLevel [/B=*box* /EDGE=*e* /P/Q/R=(*startX*,*endX*)/T=*dx*] *waveName*, *level*

FindLevel コマンドは、指定されたウェーブを検索し、指定された Y レベルが交差する X 値を特定します。

フラグ

/B=*box*　移動平均のボックスサイズを設定します。/B=box が省略された場合、または box が 1 の場合、平均化は行われません。ボックスサイズとして偶数を指定した場合、その直上の（奇数の）整数が使われます。ボックスサイズとして 1 より大きい値を使うと、ウェーブの最初または最後の (box-1)/2 ポイントで発生するレベル交差について、FindLevel は検出できなくなります。これは、これらのポイントには、派生した平均ウェーブ値を計算するのに十分な近傍データがないためです。

/EDGE=*e*　レベル交差が上昇するか下降するかのいずれかの変化を検索対象として指定します。

e=0：　/EDGE フラグを指定しない場合と同じ（レベル交差が上昇するものと下降するものを検索します）。

e=1：　ウェーブの開始ポイントから終了ポイントに向かって *level* を通過する時に、Y 値が増加している交差箇所のみを検索します。

e=2：　ウェーブの開始ポイントから終了ポイントに向かって *level* を通過する時に、Y 値が減少している交差箇所のみを検索します。

/P　X 軸上の交差位置をポイント番号に基づいて計算します。/P を省略した場合、レベル交差位置は X 値に基づいて計算されます。

/Q　履歴に結果が出力されるのを防ぎ、レベルが見つからない場合のエラーを防止します。

/R=(*startX*, *endX*)　検索対象のウェーブの X 座標範囲を指定します。*startX* と *endX* を入れ替えることで、検索方向を逆にすることもできます。

/R=[*startP*, *endP*]　検索対象のウェーブの範囲を指定します。*startP* と *endP* を入れ替えることで、検索方向を逆転させることができます。

範囲を /R=[*startP*] と指定した場合、その範囲の終わりがウェーブの終わりとみなされます。/R を省略した場合は、ウェーブ全体が検索対象となります。

/T=*dx*　最初のレベル交差ポイントを検出した後、２回目の検索を実行します。２回目の検索は、最初のレベル交差ポイントから *dx* 単位先から開始し、最初の交差ポイントの方向に向かって遡って検索します。FindLevel が２つ目のレベル交差ポイントを検出した場合、V_LevelX を最初の交差ポイントと２つ目の交差ポイントの平均値に設定します。そうでない場合は、V_LevelX を最初の交差ポイントの値に設定します。

詳細

FindLevel は、ウェーブをスキャンしながら、*level* をウェーブの Y 値から導出された値と比較します。
各導出値は、Y 値の移動平均です。

FindLevel は、*level* を挟む２つの派生ウェーブ値を検索します。
これらの値が見つかった場合、その２つの Y 値の間を線形補間することで、*level* が位置する X 値を算出します。

FindLevel は、*level* と完全に一致する値を検索するのではなく、*level* を通過する遷移を検索します。
正確な値を検索する方法については、BinarySearch を参照してください。

FindLevel は、以下の変数を設定することで結果を出力します。

V_flag　0： *level* が検出された
1： *level* が検出されなかった

V_LevelX　*level* が検出された補間された X 値、または /P が指定されている場合は、それに対応するポイント番号

V_rising　　0： 交点における Y 値は、ウェーブの始点から終点に向かって減少している
1： 交点付近の　Y　値が増加している

/Q フラグを指定しない場合、FindLevel は実行結果を履歴エリアに表示して報告します。

指定されたレベルが見つからず、かつ /Q フラグを省略した場合、FindLevel はエラーを発生させ、エラーアラートを表示するとともに、実行中のコマンドラインやマクロの実行を中断します。

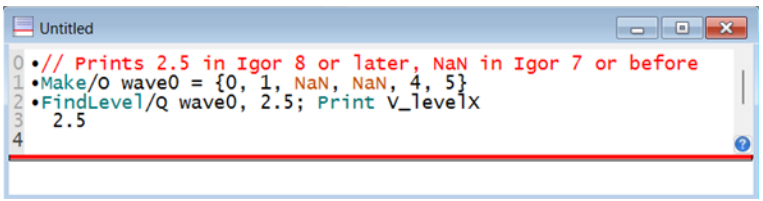
/P フラグを使わない限り、V_LevelX は指定されたウェーブの X 軸スケーリングに基づいて返されます。
/P フラグを使う場合は、ポイント番号に基づいて返されます。

FindLevel の NaN の処理

Igor Pro 8.0 以降では、FindLevel 関数は NaN 値の処理をそれ以前のバージョンとは異なる方法で行います。
現在、レベルが 2 つの NaN ではないウェーブ Y 値の間に位置し、その間に NaN が存在する場合、これら 2 つの Y 値とそれに関連付けられた X スケーリング値を使って、レベル交差の X 座標を線形補間します。
Igor Pro 7 以前のバージョンでは、レベル交差を検出できず、V_LevelX を NaN に設定します。

例えば、

```
// Igor 8 以降では 2.5 と表示され、Igor 7 以前では NaN と表示される
Make/O wave0 = {0, 1, NaN, NaN, 4, 5}
FindLevel/Q wave0, 2.5; Print V_levelX
```



次のコマンドを実行すると、Igor Pro 8 以前の動作に戻すことができます。

```
SetIgorOption UseIP6FindLevel = 1
```

FindLevel と多次元ウェーブ

FindLevel コマンドは多次元性を認識しません。
詳細は、ヘルプ Multidimensional Waves、特に Analysis on Multidimensional Waves を参照してください。

参照

FindLevels、FindValue、BinarySearch、BinarySearchInterp、EdgeStats、PulseStats コマンド

FindLevels [/B=*box* /D=*destWaveName* /DEST=*destWaveName* /EDGE=*e* /M=*minWidthX* /N=*maxLevels* /P/Q /R=(*startX*,*endX*)/T=*dx*] *waveName*, *level*

FindLevels コマンドは、指定されたウェーブを検索し、指定された Y レベルが交差する 1 つ以上の X 値を特定します。

ウェーブが指定された値と等しくなる箇所を見つけるには、代わりに FindValue を使ってください。

フラグ

/B=*box*　　移動平均のボックスサイズを設定します。FindLevel コマンドを参照してください。

/D=*destWaveName*
　　FindLevels がレベル交差値を格納するウェーブを指定します。/D と /DEST を省略し

た場合、FindLevels は W_FindLevels という名前のウェーブを作成し、そこにレベル交差値を格納します。

/DEST=*destWaveName*

/D と同じです。/D と /DEST の両方は、ユーザー関数内で、宛先のウェーブに対して実際のウェーブ参照を作成します。詳細は、ヘルプ Automatic Creation of WAVE References を参照してください。

/EDGE=*e*　レベル交差が上昇するか下降するかのいずれかの変化を検索対象として指定します。

e=0：　/EDGE フラグを指定しない場合と同じ（レベル交差が上昇するものと下降するものを検索します）。

e=1：　ウェーブの開始ポイントから終了ポイントに向かってレベルを通過する時に、Y 値が増加しているレベル交差箇所のみを検索します。

e=2：　ウェーブの開始ポイントから終了ポイントに向かってレベルを通過する時に、Y 値が減少しているレベル交差箇所のみを検索します。

/M=*minWidthX*　レベル交差間の最小 X 方向の距離を設定します。これは、FindLevels がレベル交差を検出した後、次の交差をどこで検索するかを決定します。検索は、その交差から X 単位離れた位置から開始されます。

/N=*maxLevels*　FindLevels が検出する交差の最大数を設定します。*maxLevels* のデフォルト値は、*waveName* の指定された範囲内のポイント数です。

/P　交点を基に交差を計算します。FindLevel コマンドを参照してください。

/Q　履歴には記録されず、レベルが見つからない場合でも処理は中止されません。FindLevel 操作を参照してください。

/R=(*startX*, *endX*)　X の範囲を指定します。FindLevel コマンドを参照してください。

/R=[*startP*, *endP*]　ポイントの範囲を指定します。FindLevel コマンドを参照してください。

/T=*dx*　２つのレベル交差ポイントを検索します。*dx* は *minWidthX* より小さくなければならないため、/T を使う場合は /M も指定する必要があります（FindLevels は、２回目の検索の開始位置が、次のエッジに対する１回目の検索の終了位置を超えないように *dx* を制限します）。/T の詳細については、FindLevel を参照してください。

詳細

レベル交差ポイントを検出するアルゴリズムは、FindLevel コマンドで使われているものと同じです。

FindLevels が *maxLevels* 個のレベル交差を検出するか、それ以上のレベル交差が見つからない場合、検索を中止します。

FindLevels は、以下の変数を設定します。

V_flag　0： *maxLevels* 個のレベル交差が見つかりました。
1： *maxLevels* 未満のレベル交差が少なくとも１か所見つかりました。
2： レベル交差は見つかりませんでした。

V_LevelsFound　検出されたレベル交差の数です。

例

```
Macro FindLevelsExample()
    // サンプルデータ
    Make/O/N=1024 peaks;SetScale/P x,0,0.001,"" peaks      // 1ms サンプリング
```

```
        peaks=(exp(sawtooth(-(x-1)*300/pi)^3)-1)/(exp(1)-1)
        SetRandomSeed 0; peaks += gnoise(0.05)

        // y=0.5 の X 軸との交点のうち、増加区間と減少区間の交点を特定する
        Variable minDX=20*deltax(peaks)         // 交差間のサンプル数は最小 20
        Variable level = 0.5                    // ピークの Y 範囲は、名目上 0..1
        FindLevels/Q/D=risingEdges/EDGE=1/M=(minDX) peaks, level
        FindLevels/Q/D=fallingEdges/EDGE=2/M=(minDX) peaks, level

        // 検出されたエッジを示す
        Duplicate/O risingEdges, risingYs; risingYs = peaks(risingEdges[p])
        Duplicate/O fallingEdges, fallingYs; fallingYs = peaks(fallingEdges[p])
        // レベルを表示
        Make/O/N=2 showLevel = level;CopyScales/I peaks, showLevel

        Display /W=(35.25,41,856.5,249.5) peaks
        AppendToGraph risingYs vs risingEdges
        AppendToGraph fallingYs vs fallingEdges
        AppendToGraph showLevel
        ModifyGraph mode(risingYs)=8,mode(fallingYs)=8
        ModifyGraph marker(risingYs)=17,marker(fallingYs)=23
        ModifyGraph lStyle(showLevel)=1
        ModifyGraph
rgb(risingYs)=(19675,39321,1),rgb(fallingYs)=(0,0,0),rgb(showLevel)=(1,16019,65535)
        Legend/C/N=text0/X=3.55/Y=1.90
    End
```

参照

レベル交差検知アルゴリズムと /B、/P、/Q、/R、/T フラグの値に関する詳細は、FindLevel コマンドを参照してください。

FindLevels コマンドは多次元性を認識しません。
詳細は、ヘルプ Multidimensional Waves、特に Analysis on Multidimensional Waves を参照してください。