CONTENTS

ビジュアルヘルプ – データのインポートとエクスポート（3）

Matlab MAT ファイルの読み込み

MLLoadWave コマンドは、Matlab MAT ファイルを Igor Pro に読み込みます。
このコマンドは、MLLoadWave コマンドを直接選択するか、Load Matlab MAT File ダイアログを表示する Data →Load Waves→Matlab MAT File を選択することでアクセスできます。

MLLoadWave は Matlab アプリケーションが提供するダイナミックライブラリに依存しています。
MLLoadWave を使うには、使用するマシンに互換性のあるバージョンの Matlab がインストールされている必要があります。
Matlab をお持ちでない場合、Matlab のバージョンが Igor と互換性がない場合、または単に HDF5 ファイルでの作業を行いたい場合は、回避策として「バージョン 7.3 の MAT ファイルを HDF5 ファイルとして読み込む」のセクションを参照してください。

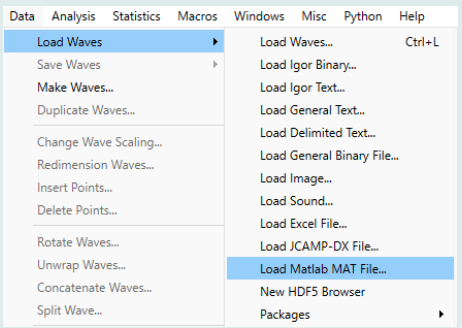
MLLoadWave コマンドは、Igor Pro 7.0 で Igor に組み込まれました。
それ以前のバージョンでは XOP として実装されていました。
この XOP は、Yves Peysson 氏と Bernard Saoutic 氏によって最初に作成されました。

Matlab ダイナミックライブラリの検索

MLLoadWave は、Matlab のインストーラーによって Mathworks 社が提供するライブラリと動的にリンクします。
Igor に以下の通り検索場所を指定する必要があります：

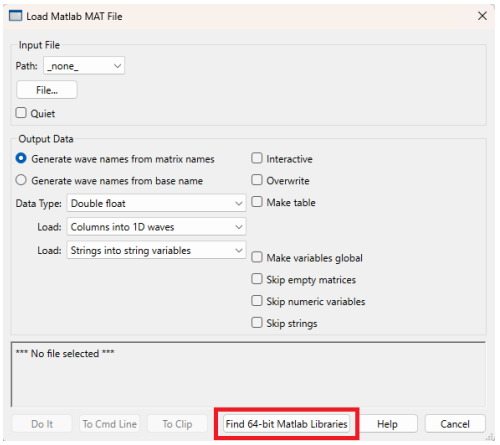
1. **メニュー Data → Load Waves → Load Matlab MAT File を選択します。**

   Load Matlab MAT File が表示されます。



2. **Find 32-bit Matlab Libraries ボタンまたは Find 64-bit Matlab Libraries ボタンをクリックしてください。**

   ボタンのタイトルは、IGOR32（Windows 専用）または IGOR64 を実行しているかによって異なります。
   クリックすると「Find Matlab」ダイアログが表示されます。



3. Choose Folder　ダイアログを表示するために、Folder ボタンをクリックします。

4. Matlab フォルダーに移動し、それを選択してください。

これは次のようなものになるでしょう：

```
C:\Program Files\MATLAB\<version>             // 64-bit Windows
C:\Program Files (x86)\MATLAB\<version>       // 32-bit Windows
```

ここで \<version\> は Matlab のバージョン（例：R2015a）です。

5. Choose ボタンをクリックします。

Igor は必要なダイナミックライブラリを見つけるため、Matlab フォルダーを検索します。
見つかった場合、それらを読み込もうとします。
検索と読み込みが成功すると、Accept ボタンが有効になります。
検索と読み込みが失敗すると、Accept ボタンは無効になります。
Igor が必要な Matlab ダイナミックライブラリを見つけられない場合、またはシステムが Matlab ダイナミックライブラリが必要とする他のダイナミックライブラリを見つけられない場合、検索は失敗します。

有効な Matlab フォルダーを選択しているにもかかわらず、Accept ボタンが有効にならない場合は、「Matlab ダイナミックライブラリの問題」のセクションを参照してください。

6. Accept ボタンをクリックします。

Igor は、将来のセッションで使うために、Matlab ダイナミックライブラリの場所をプレファレンスに記録します。

Matlab のダイナミックライブラリ場所を指定する前に MLLoadWave を呼び出すと、MLLoadWave は Find Matlab ダイアログを表示します。
上記のステップに従って Matlab のインストール場所を特定してください。

Matlab ダイナミックライブラリの問題

MLLoadWave は、Igor Pro バージョンと同じアーキテクチャ向けにビルドされた互換性のある Matlab ダイナミックライブラリを必要とします。
IGOR32（32 ビット版 Igor Pro）には 32 ビット版 Matlab ライブラリが、IGOR64（64 ビット版 Igor Pro）には 64 ビット版 Matlab ライブラリが必要です。

Matlab を持っていない場合、または使っている Matlab のバージョンが Igor と互換性がない場合、あるいは単に HDF5 ファイルでの作業を行いたい場合は、回避策として「バージョン 7.3 の MAT ファイルを HDF5 ファイルとして読み込む」のセクションを参照してください。

Windows における Matlab ダイナミックライブラリの問題

Igor Pro 7.02 以前のバージョンでは、Matlab のダイナミックライブラリディレクトリへのパスを Windows の PATH 環境変数に設定する必要がありました。
7.02 以降では、この設定は不要です。

ただし、すべてのバージョンの Matlab でテストすることは不可能であり、将来のバージョンでは動作が異なる可能性があります。
上記の「Matlab ダイナミックライブラリの検索」に記載されている手順に従っても、Igor が Matlab ダイナミックライブラリにリンクできない場合は、Windows PATH 環境変数に Matlab ライブラリのパスを追加してみてください。
IGOR32 には 32 ビット版 Matlab ライブラリが、IGOR64 には 64 ビット版 Matlab ライブラリが必要であることを忘れないでください。
（Igor 10 は 64 ビット専用であることにも注意してください。）
再テストする前に Igor を再起動してください。

Matlab 2021 が Igor Pro 9 以降と互換性がないとの報告を受けています。
これはダイナミックライブラリの競合が原因であるようです。
回避策については「バージョン 7.3 の MAT ファイルを HDF5 ファイルとして読み込む」のセクションを参照してください。

サポートしている Matlab データ形式

MLLoadWave は 1D、2D、3D、4D の数値データおよび文字列データをロードできます。
MLLoadWave は４次元を超えるデータのロードはできません。

Matlab の文字列データを Igor ウェーブに読み込む時、Igor ウェーブの次元は Matlab データセットよりも１つ小さくなります。
これは、Matlab 文字列データセットの各要素が１バイトであるのに対し、Igor 文字列ウェーブの各要素は文字列（任意のバイト数）であるためです。

MLLoadWave は、次の種類の Matlab データの読み込みをサポートしていません：セル配列、構造体、疎データセット、オブジェクト、64 ビット整数。

数値データの読み込みモード

Load Matlab MAT File ダイアログには、数値データを読み込む方法をコントロールするポップアップメニューが表示されます。
メニュー項目は以下の通りです：

Load columns into 1D wave	Matlab 行列の各列は、個別の１次元 Igor ウェーブに読み込まれます。
Load rows into 1D wave	Matlab 行列の各行は、個別の１次元 Igor ウェーブに読み込まれます。
Load matrix into one 1D wave	Matlab の行列全体が１つの１次元 Igor ウェーブに読み込まれます。
Load matrix into matrix	Matlab の行列が Igor 行列に読み込まれます。
Load matrix into transposed matrix	Matlab の行列が Igor 行列に読み込まれますが、行と列が転置されます。

３次元または４次元のデータをロードする時、最初の３つのモードでは各層（Matlab 用語では「ページ」）を個別の行列として扱います。
3D Matlab データの場合、以下の動作となります：

Load columns into 1D wave	Matlab データセットの各レイヤーの各列は、個別の１次元 Igor ウェーブとして読み込まれます。
Load rows into 1D wave	Matlab データセットの各レイヤーの各行は、個別の１次元 Igor ウェーブに読み込まれます。

Load matrix into one 1D wave	Matlab データセットのレイヤーが 1 次元 Igor ウェーブに読み込まれます。
Load matrix into matrix	Matlab 3D データセットが 3 次元 Igor ウェーブに読み込まれます。
Load matrix into transposed matrix	Matlab 3D データセットは 3 次元 Igor ウェーブに読み込まれますが、行と列が転置されます。

3D または 4D データセットを読み込む時、最後の 2 つのモードにおける「行列」という用語は適切ではありません。
MLLoadWave は、3D または 4D データセット全体を 3D または 4D の Igor ウェーブに読み込みます。

バージョン 7.3 の MAT ファイルを HDF5 ファイルとして読み込む

2006 年、Matlab は MAT ファイル形式のバージョン 7.3 を追加しました。
バージョン 7.3 の MAT ファイルは、ファイルの先頭に 512 バイトの Matlab 固有情報が付加された HDF5 ファイルです。
HDF5 ライブラリはアプリケーション固有データを先頭に追加することを許可しているため、バージョン 7.3 の MAT ファイルは HDF5 ファイルとして読み込むことが可能です。

Igor は MAT ファイルよりも HDF5 ファイルのサポートが優れているため、使っているマシンに Matlab がインストールされていない場合、あるいは Igor の Matlab サポートがお使いの Matlab 環境と互換性がない場合などには、こうしたファイルを HDF5 形式で読み込むと便利です。
Igor の HDF5 サポートに関する詳細は、ヘルプ HDF5 in Igor Pro を参照してください。

バージョン 7.3 の MAT ファイルは、バイトオフセット 512 に HDF5 シグネチャを含みます。
HDF5 シグネチャは、8 バイトのパターンであり、以下で説明されています。

https://support.hdfgroup.org/documentation/hdf5/latest/_f_m_t11.html#subsec_fmt11_boot_super

HDF5 Browser で 7.3 MAT ファイルを開くには、HDF5 Browser の Open File ダイアログのポップアップメニューから All Files を選択する必要があります。
バージョン 7.3 以外の MAT ファイルを HDF5 ファイルとして開こうとすると、HDF5 ライブラリはエラーを返します。

例えば、バージョン 7.3 形式で Matlab データを保存する方法については、
https://mathworks.com/help/matlab/ref/save.html
を参照してください。
Matlab のプレファレンスを変更して、バージョン 7.3 をデフォルト形式にすることも可能です。

一般的なバイナリファイルの読み込み

一般的なバイナリファイルとは、他のプログラムによって作成されたバイナリファイルです。
バイナリファイル形式を理解していれば、そのデータを Igor に読み込むことが可能です。
ただし、バイナリファイル形式を正確に理解している必要があります。
これは通常、比較的単純な形式の場合にのみ可能です。

一般的なバイナリファイルからデータを読み込むには、次の 2 つの方法があります：

- FBinRead コマンドを使う
- GBLoadWave コマンドを使う

FBinRead の使用は、GBLoadWave の使用よりもやや難しいですが、より柔軟性があります。
特に、構造体として保存されたデータを読み込む場合に役立ちます。

詳細については、FBinRead コマンドのヘルプを参照してください。
このセクションでは、GBLoadWave コマンドの使用に焦点を当てます。

GBLoadWave は、一般的なバイナリファイルからデータをウェーブに読み込みます。
「GB」は「general binary」を意味します。

GBLoadWave コマンドは直接呼び出すか、または Data→Load Waves→Load General Binary File を選択することで呼び出せます。
これにより Load General Binary ダイアログが表示されます。

GBLoadWave を正しく使うには、バイナリファイルの形式を正確に把握する必要があります。
したがって、主に自身のプログラムで作成したバイナリファイルを読み込む時に役立ちます。
ファイル形式を正確に把握している場合、サードパーティ製ファイルの読み込みにも GBLoadWave を利用できます。

GBLoadWave が処理できるファイル

GBLoadWave は次の種類のバイナリデータを扱います：

- 8 bit、16 bit、32 bit、64 bit 符号付き・符号なし整数
- 32 bit、64 bit IEEE 浮動小数点数
- 32 bit、64 bit VAX 浮動小数点数

さらに、GBLoadWave は高バイト先頭（Motorola）および低バイト先頭（Intel）タイプのバイナリ数値を扱います。

GBLoadWave は現在、IEEE または VAX 拡張精度値を処理できません。
詳細は「VAX 浮動小数点」のセクションを参照してください。

GBLoadWave は、Igor がサポートする任意の数値データ型（64 ビットおよび 32 ビット IEEE 浮動小数点、64 ビット、32 ビット、16 ビット、8 ビットの符号付きおよび符号なし整数）を使ってウェーブを作成できます。
ウェーブのデータ形式は、ファイルのデータ形式と同じである必要はありません。
例えば、整数の A/D 読み取り値を含むファイルがある場合、そのデータを単精度または倍精度の浮動小数点ウェーブに読み込むことができます。

一般的に、ウェーブは浮動小数点として読み込むのが最善です。
ほぼすべての Igor コマンドは、浮動小数点でより高速に動作するためです。
ただし、画像、特に画像のスタックを扱う場合は例外です。
例えば、ファイルに 512x512x1024 バイトの画像スタックがある場合、それをバイトウェーブとしてロードする必要があります。

GBLoadWave は Igor の多次元ウェーブについて何も知りません。
これは１次元のみを扱います。
GBLoadWave ダイアログで使われる「配列」という用語は「１次元配列」を意味します。
１次元ウェーブとしてデータを読み込んだ後、必要に応じて次元の変更を行うことができます。

GBLoadWave は、ファイルから１つ以上の１次元配列を読み込むことができます。
複数の配列を読み込む場合、それらはファイル内で順次格納されるか、またはインターリーブされる可能性があります。
「順次」とは、ある配列のすべてのポイントがファイルに現れた後、次の配列のすべてのポイントが現れることを意味します。
インターリーブとは、各配列のポイント 0 がファイルに現れた後、各配列のポイント 1 が現れることを意味します。

GBLoadWave と非常に大きなファイル

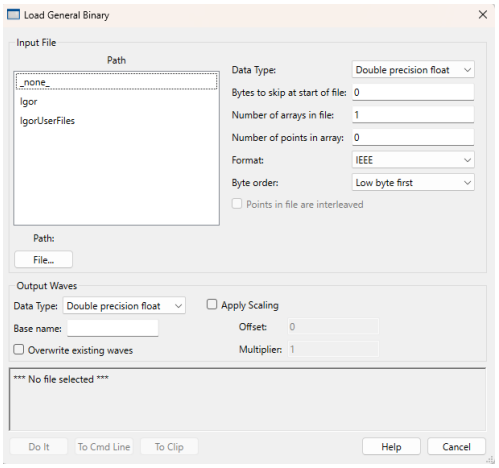
ほとんどのデータファイルは、GBLoadWave や Igor にとって大きな問題となるほど大きくはありません。
ただし、データファイルが数億バイトから数十億バイトに近づく場合、サイズやメモリの問題が発生する可能性があります。

GBLoadWave でデータの型を変換する場合（例えば：16 ビット符号付きから 32 ビット浮動小数点へ）、読み込み処理中に追加のバッファが必要となり、より多くのメモリを消費します。

非常に大きなファイルを扱う場合、GBLoadWave /S および /U フラグを使って、データファイルの一度に一部だけを Igor に読み込む必要がある場合があります。

Load General Binary ダイアログ

Data→Load Waves→Load General Binary File を選択すると、Igor は Load General Binary ダイアログを表示します。
このダイアログでは、読み込むファイルを選択し、ファイルのデータ型および作成するウェーブ（または複数のウェーブ）のデータ型を指定できます。



ダイアログ内のいくつかの項目について、説明が必要です。

Number of Arrays in File テキストボックスと Number of Points in Array テキストボックスは、初期設定でどちらも「auto」に設定されています。
auto とは、GBLoadWave がファイル内のバイト数に基づいてこれらを自動的に決定することを意味します。

両方を auto 設定のままにすると、GBLoadWave はファイル内に 1 つの配列が存在し、そのポイント数はファイルのバイト数と各ポイントのデータ長によって決定されると仮定します。

Number of Arrays in File を 0 より大きい数値に設定し、Number of Points in Array を auto に設定した場合、GBLoadWave はファイル内の総バイト数と指定された配列数に基づいて、各配列のポイント数を決定します。

Number of Points in Array を 1 より大きい数値に設定し、Number of Arrays in File を auto に設定した場合、GBLoadWave はファイルの総バイト数と各配列に指定されたポイント数に基づいて、ファイル内の配列数を決定します。

各設定で「auto」の代わりに数値を入力することで、ファイル内の配列の数と各配列のポイント数を明示的に指定することも可能です。

GBLoadWave は 1 つ以上の 1 次元ウェーブを作成し、指定された基本名に番号を付加して生成した名前をウェーブに付与します。
例えば、基本名が「wave」の場合、wave0、wave1 などの名前を持つウェーブを作成します。

Overwrite Existing Waves チェックボックスがオンの場合、GBLoadWave は既存のウェーブ名を使い、それらを上書きします。
オフの場合、GBLoadWave は既に使われている名前をスキップします。

Apply Scaling チェックボックスをオンにすると、オフセットと乗数を指定でき、GBLoadWave がデータを意味のある単位にスケーリングします。
このチェックボックスがオフの場合、GBLoadWave はスケーリングを行いません。

VAX 浮動小数点

GBLoadWave は、VAX「F」フォーマット（32 ビット、単精度）および「G」フォーマット（64 ビット、倍精度）の数値を読み込むことができます。

VAX データに対して GBLoadWave のバイトスワップ機能（/B フラグ）を使わないでください。
これは Intel から Motorola へのバイトスワップ（リトルエンディアンからビッグエンディアンへの変換）を行います。
VAX データは Igor のデータ保存方式に対してバイトスワップされていますが、同じ意味でのスワップではありません。
具体的には、各 16 ビットワードはビッグエンディアンですが、各 8 ビットバイトはリトルエンディアンです。
入力データが VAX データであることを /J=2 で指定すると、GBLoadWave は VAX データに必要なスワッピングを行います。

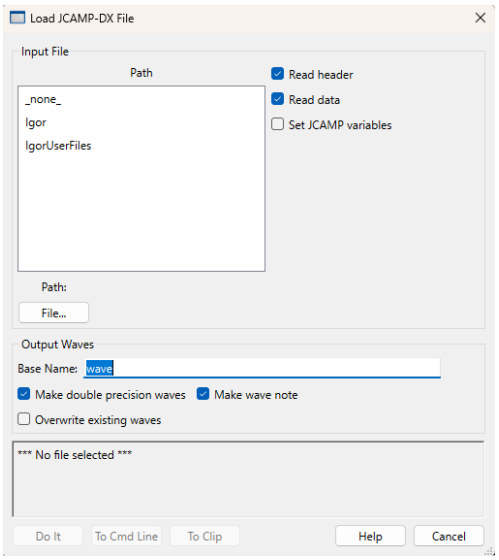
GBLoadWave は、VAX「D」フォーマット（別の 64 ビット形式）を読み取れません。
ただし、VAX D フォーマットは F フォーマットと同じで、小数部として追加の 4 バイトを持つだけです。
これにより、余分な小数部ビットを破棄して、VAX D フォーマットを F フォーマットとして読み込むことが可能です。
例を以下に示します：

```
GBLoadWave/W=2/V/P=VAXData/T={2,2}/J=2/N=temp "VAX D File"
KillWaves temp1
Rename temp0, VAXDData_WithoutExtraFractBits
```

/W=2 フラグは、ファイル内に 2 つの配列が存在することを GBLoadWave に伝えます。
/V フラグは、それらがインターリーブされていることを伝えます。
ファイル内の各データポイントの最初の 4 バイトは、temp0 ウェーブに格納されます。
D フォーマットの余分な小数部ビットを含む 2 番目の 4 バイトは、temp1 に格納されますが、これは破棄します。

JCAMP ファイルの読み込み

Igor は JCAMPLoadWave コマンドを使って JCAMP-DX ファイルを読み込むことができます。
JCAMP-DX フォーマットは主に赤外分光法で使われます。
これは ASCII 文字のみを使うプレーンテキスト形式です。



JCAMPLoadWave コマンドは直接呼び出すか、または Data→Load Waves→Load JCAMP-DX File を選択することで呼び出せます。
これにより Load JCAMP-DX ダイアログが表示されます。

JCAMPLoadWave は、JCAMP-DX ファイルヘッダを十分に理解してデータを読み込み、ウェーブのスケールを適切に設定します。
JCAMP-DX は主に等間隔のデータ用であるため、データセットごとに 1 つのウェーブが生成されます。

ウェーブの X スケールは、JCAMP-DX ファイルヘッダーの情報に基づいて設定されます。
ヘッダー情報は、オプションでウェーブのノートに保存され、またオプションで一連の Igor 変数に保存されます。
これらの変数を作成することを選択した場合、ヘッダー内の各 JCAMP-DX ラベルに対応する変数が 1 つずつ生成されます。

JCAMPLoadWave が処理できるファイル

JCAMPLoadWave は、１つのファイルから１つ以上のウェーブを読み込むことができます。
JCAMP-DX 規格では、新しいデータセットごとに新しいヘッダーで開始することが求められています。
各ヘッダーは ##TITLE= ラベルで開始する必要があります。
当社の知る限り、ほとんどの分光器システムはファイルごとに１つのデータセットのみを書き込みます。

さらに、JCAMP-DX 規格には、JCAMPLoadWave がサポートする簡易なオプション圧縮技術が含まれています。
圧縮を使わないファイルは人間が直接読める形式です。

JCAMPLoadWave は、標準的な JCAMP-DX フォーマットで保存されたほとんどのファイルを読み込めるはずです。
正しく読み込めない JCAMP-DX ファイルをお持ちの場合は、support@wavemetrics.com までお送りください。

一部のシステムは、データ自体がバイナリファイルに保存され、JCAMP-DX スタイルのヘッダーのみを含む ASCII ファイルが添付されるハイブリッド形式を生成します。
特定の Bruker NMR 分光計がこれを行っていることがわかっています。
これらのシステムに対応するために、ヘッダー情報のみを読み込むオプションを選択することができます。
その場合、おそらく GBLoadWave を使って、データを別途読み込む必要があります。

JCAMP ヘッダー情報を読み込む

JCAMPLoadWave は、ヘッダー情報を Igor に読み込むための２つのメカニズムを提供します：

- すべてのヘッダーテキストをウェーブノートに保存する
- ヘッダー内で遭遇した各 JCAMP ラベルに対して１つの Igor 変数を作成する

Load JCAMP-DX File ダイアログで Make Wave Note チェックボックスをオンにすると、/W フラグが呼び出され、ヘッダー全体がウェーブノートに保存されます。

Set JCAMP Variables のチェックは /V フラグを呼び出し、ヘッダー内で検出された各 JCAMP ラベルに対して１つの Igor 変数を作成します。
詳細は次のセクションで説明します。

JCAMPLoadWave で設定される変数

JCAMPLoadWave は、Igor ファイルローダーの出力変数である S_fileName、S_path、V_flag、S_waveNames を設定します。
これらは JCAMPLoadWave のリファレンスドキュメントに記載されています。

/V フラグを使うと（ダイアログの Set JCAMP Variables チェックボックスに対応）、ヘッダー変数も設定されます。
ヘッダー変数とは、JCAMPLoadWave が JCAMP ヘッダーから取得したデータを含む変数です。

JCAMPLoadWave がマクロから呼び出された場合、ヘッダー変数はローカル変数として作成されます。
コマンドラインまたはユーザー定義関数から呼び出された場合、ヘッダー変数はグローバル変数として作成されます。
「関数からヘッダー変数を使う」セクションで、この点についてさらに詳しく説明しています。

ヘッダー変数名は、JCAMP ラベルを基に、文字列変数には「SJC_」、数値変数には「VJC_」のプリフィックスが付与されます。

したがって、##TITLE ラベルを検出すると、JCAMPLoadWave はラベル内容を含む「SJC_TITLE」という文字列変数を作成します。

特定の JCAMP ラベルは数値情報として解析され、数値変数が作成されます。
作成される可能性のある数値変数には以下が含まれます：

VJC_NPOINTS	データセット内のデータポイント数に設定されます。これはヘッダー情報から設定されます。ファイル内の実際のデータポイント数が異なる場合、この変数はそれを反映しません。
VJC_FIRSTX	データセットの最初のデータポイントの X 値に設定します。
VJC_LASTX	データセットの最後のデータポイントの X 値に設定します。
VJC_DELTAX	連続する横座標値間の間隔に設定します。これは (VJC_LASTX - VJC_FIRSTX) / (VJC_NPOINTS - 1) から計算されるため、##DELTAX=label で指定される値とはわずかに異なる場合があります。
VJC_XFACTOR	ファイル内の X データ値に実世界の値を得るために適用すべき乗数を設定します。
VJC_YFACTOR	ファイル内の Y データ値に実世界の値を得るために適用すべき乗数を設定します。
VJC_MINY	データセット内で見つかった最小の Y 値に設定します。
VJC_MAXY	データセット内で見つかった最大の Y 値に設定します。

フーリエ領域データをロードする場合、データが光位相遅延と振幅を表すことを反映するために、以下の変数が作成される可能性があります：VJC_FIRSTR、VJC_LASTR、VJC_DELTR、VJC_RFACTOR、VJC_AFACTOR。

ヘッダー内で検出されたその他のラベルは、SJC_<ラベル名> という名前の文字列変数として生成されます。
ここで <ラベル名> は JCAMP ラベル名に置き換えられます。
例えば、##YUNITS ラベルは SJC_YUNITS という名前の文字列変数を生成します。

１つのファイル内の連続するデータセットは同じ標準ラベルを持つため、変数の内容はファイル内の特定のラベルの最後のインスタンスによって決定されます。

関数からヘッダー変数を使う

ユーザー定義関数から JCAMPLoadWave を実行し、/V フラグでヘッダー変数の作成を指示した場合、変数は現在のデータフォルダー内のグローバル変数として作成されます。
これらの変数にアクセスするには、NVAR および SVAR 参照を使う必要があります。
これらの参照は JCAMPLoadWave の呼び出し後に記述しなければなりません。
例えば：

```
Function LoadJCAMP()
        JCAMPLoadWave/P=JCAMPFiles "JCAMP1.dx"
        if (V_Flag == 0)
                Print "No waves were loaded"
                return -1
        endif

        NVAR VJC_NPOINTS
        Printf "Number of points: %d\r", VJC_NPOINTS

        SVAR SJC_YUNITS
        Printf "Y Units: %s\r", SJC_YUNITS

        return 0
End
```

上記のコードは、ヘッダーに ##NPOINTS ラベルが含まれており、そこから変数 VJC_NPOINTS と SJC_YUNITS が作成されることを前提としています。
ファイルにそのようなラベルが含まれていることを保証できない場合は、変数を使う前に NVAR/Z および NVAR_Exists を使って変数の存在を確認する必要があります。

実行時に作成された変数を特定する必要がある場合は、GetIndexedObjName 関数を使い、各名前が SJC_ または VJC_ で始まるかどうかをテストしてください。

関数内のヘッダー変数に関するもう一つの問題は、周囲に多くの不要なコードを残すことです。
次のように整理できます：

```
KillVariables/Z VJC_NPOINTS
KillStrings/Z SJC_YUNITS
```

音声ファイルを読み込む

Igor Pro 7 で追加された SoundLoadWave コマンドは、様々な音声ファイル形式からデータをロードします。

Igor の音声関連機能に関する一般的な情報については、ヘルプ Sound を参照してください。

Igor プロシージャを使ってウェーブを読み込む

Igor の強みのひとつは、データを自動的に読み込み、処理し、グラフ化するプロシージャを記述できる点です。
これは、同一または類似の構造を持つデータファイルを大量に蓄積している場合や、業務で定期的にそのようなファイルを生成する場合に役立ちます。

プロシージャへの入力は、１つ以上のデータファイルです。
出力は、グラフやページレイアウトの印刷物、あるいは計算結果のテキストファイルとなる可能性があります。

その状況に合わせたプロシージャが必要となります。
このセクションでは、出発点として役立つ可能性のあるいくつかの例を紹介します。

ファイルローダーによって設定される変数

LoadWave コマンドは、ウェーブを自動的に読み込むプロシージャに有用な情報を提供するため、数値変数 V_flag と文字列変数 S_fileName、S_path、S_waveNames を作成します。
関数内で使われる場合、LoadWave コマンドはこれらをローカル変数として作成します。

他のほとんどのファイルローダーは、同じまたは類似の出力変数を作成します。

LoadWave は、読み込まれるファイル名を文字列変数 S_fileName に設定します。
これはグラフやページレイアウトの注釈付けに便利です。

LoadWave は、読み込まれたファイルを含むフォルダーのフルパスを文字列変数 S_path に設定します。
これは、最初のファイルと同じフォルダーから２つ目のファイルを読み込む必要がある場合に便利です。

LoadWave は変数 V_flag に読み込まれたウェーブの数を設定します。
これにより、ファイル内のウェーブの数を事前に知らなくても、プロシージャがウェーブを処理できるようになります。

LoadWave はまた、文字列変数 S_waveNames を、読み込まれたウェーブ名のセミコロン区切りリストに設定します。
プロシージャから、このリスト内の名前を後続の処理に使用できます。

ウェーブフォームデータの読み込みとグラフ作成

以下は、データファイルの内容を自動的に読み込みグラフ化する Igor 関数の基本形式を示すための非常に簡単な例です。
区切り文字付きテキストファイルからウェーブフォームデータを読み込み、そのウェーブのグラフを作成します。

この例では Igor のシンボリックパスを使っています。
この概念に慣れていない場合は、ヘルプ Symbolic Paths を参照してください。

この関数では、読み込むファイルがウェーブフォームデータの 3 列で構成されていると仮定します。
特定のデータファイル形式に合わせて関数を調整することで、関数を非常にシンプルに保つことができます。

```
Function LoadAndGraph(fileName, pathName)
        String fileName                         // 読み込むファイル名、またはダイアログを表示する場合は ""
        String pathName                         // パス名、またはダイアログを表示する場合は ""

        // ウェーブを読み込み、ローカル変数を設定する
        LoadWave/J/D/O/P=$pathName filename
        if (V_flag==0)                          // ウェーブが読み込まれない。おそらくユーザーがキャンセル
                return -1
        endif

        // 3 つのウェーブの名前を文字列変数に入れる
        String s0, s1, s2
        s0 = StringFromList(0, S_waveNames)
        s1 = StringFromList(1, S_waveNames)
        s2 = StringFromList(2, S_waveNames)

        Wave w0 = $s0                           // ウェーブの参照を作成
        Wave w1 = $s1
        Wave w2 = $s2

        // ウェーブの X スケーリング、X 単位、データ単位を設定
        SetScale/P x, 0, 1, "s", w0, w1, w2
        SetScale d 0, 0, "V", w0, w1, w2

        Display w0, w1, w2                      // 新しいグラフを作成

        // グラフに注釈を追加
        Textbox/N=TBFileName/A=LT "Waves loaded from " + S_fileName

        return 0                                // 成功
End
```

s0、s1、s2 はローカル文字列変数であり、読み込まれたウェーブの名前に代入します。
その後、$ 演算子を使って各ウェーブへの参照を作成し、後続のコマンドで使用できます。

関数を Procedure ウィンドウに入力すると、コマンドラインから実行したり、別の関数から呼び出したりできます。

```
LoadAndGraph("", "")
```

を実行すると、LoadWave コマンドは Open File ダイアログを表示し、ファイルを選択できるようにします。
適切なパラメーターで LoadAndGraph を呼び出すと、LoadWave はダイアログを表示せずにファイルを読み込みます。

Procedure ウィンドウに以下のメニュー宣言を記述することで、「Load And Graph」メニュー項目を追加できます：

```
Menu "Macros"
        "Load And Graph...", LoadAndGraph("", "")
End
```

LoadWave コマンドで Auto name & go オプションを使っていないため、LoadWave は新規ウェーブに名前を入力できる別のダイアログを表示します。
プロシージャをより自動化したい場合は、/A または /N オプションを使って Auto name & go を有効にしてください。
ロードされたウェーブの名前の指定をユーザーが行いたい場合は、/B フラグを使ってください。
詳細は LoadWave コマンドの説明を参照してください。

機能をシンプルに保つため、新規ウェーブに対して X スケーリング、X 単位、データ単位をハードコードしています。
自身のデータに合わせて SetScale コマンドのパラメーターを変更する必要があります。
柔軟性を高めるには、関数に追加パラメーターを設ける必要があります。

LoadAndGraph は、任意の列数のファイルを処理できるように記述することが可能です。
これにより関数はより複雑になりますが、汎用性が高まります。

より高度なプログラマー向けに、LoadAndGraph のより一般的なバージョンを以下に示します。

```
Function LoadAndGraph(fileName, pathName)
        String filename                 // 読み込むファイル名、またはダイアログを表示する場合は ""
        String pathname                 // パス名、またはダイアログを表示する場合は ""

        // ウェーブを読み込み、ローカル変数を設定する
        LoadWave/J/D/O/P=$pathName filename
        if (V_flag==0)                  // ウェーブが読み込まれない。おそらくユーザーがキャンセル
                return -1
        endif

        Display                         // 新しいグラフを作成

        String theWave
        Variable index=0
        do                              // グラフにウェーブをアペンド
                theWave = StringFromList(index, S_waveNames)    // 次のウェーブ
                if (strlen(theWave) == 0)                       // さらにウェーブがあるか？
                        break                                   // ループの外に出る
                endif
                Wave w = $theWave
                SetScale/P x, 0, 1, "s", w                      // X スケーリングを設定
                SetScale d 0, 0, "V", w                         // データの単位を設定
                AppendToGraph w
                index += 1
        while (1)                       // 無条件に "do" までループバック

        // グラフに注釈を追加
        Textbox/A=LT "Waves loaded from " + S_fileName

        return 0                        // 成功
End
```

do ループは S_waveNames 内の名前リストから順次名前を取り出し、対応するウェーブをグラフに追加します。
S_waveNames にはファイルから読み込まれた各列に対応する名前が 1 つずつ含まれます。

LoadAndGraph 関数には重大な欠点があります。
非常にシンプルなデフォルトのグラフしか作成できません。
この問題を克服するには 4 つの方法があります：

- プレファレンスを使う
- スタイルマクロを使う
- プロシージャ内にグラフのフォーマットを直接設定する
- 既存のグラフ内のデータを上書きする

通常、Igor はプロシージャの実行中にプレファレンスを使いません。
LoadAndGraph 関数の中でプレファレンスを有効にするには、関数の先頭付近に「Preferences 1」というステートメントを記述する必要があります。
これにより、関数の実行中のみプレファレンスが有効になります。
この設定により、Display と AppendToGraph コマンドがグラフプレファレンスを使うようになります。

関数内でプレファレンスを使うと、プレファレンスを変更した時にその関数の出力も変化します。
また、この関数を同僚に渡した場合、異なる結果が生成されることも意味します。
このプレファレンスへの依存性は、達成しようとしている目標によって、機能として捉えることも問題として捉えることもできます。
通常、プロシージャはプレファレンスから独立している状態を保つことが望ましいです。

スタイルマクロを使う方がより堅牢な手法です。
これを行うには、まずプロトタイプグラフを作成し、Igor にそのグラフ用のスタイルマクロを作成するよう指示します（ヘルプ Graph Style Macros を参照）。
次に、LoadAndGraph マクロの末尾にスタイルマクロへの呼び出しを配置します。
スタイルマクロは、そのスタイルを新しいグラフに適用します。

コードを自己完結させるには、グラフの書式設定をコード内で直接設定できます。
LoadAndGraph 関数を煩雑にしないために、これはサブルーチン内で行うべきです。

最後のアプローチは、新しいグラフを作成するのではなく、既存のグラフのデータを上書きすることです。
これを行う最も簡単な方法は、ウェーブに常に同じ名前を使うことです。
例えば、３つのウェーブを含むファイルを読み込み、それらを wave0、wave1、wave2 と名前を付けるとします。
次に、ウェーブのグラフを作成し、グラフのすべてを好みに合わせて設定します。
次に別のファイルを読み込み、同じ名前を使用し、LoadWave の「上書き」オプションを選択します。
新しいファイルのデータが既存のウェーブデータを置き換え、自動的に既存のグラフを更新します。
この方法を使うと、関数は以下のように簡略化されます：

```
Function LoadAndGraph(fileName, pathName)
        String fileName                         // 読み込むファイル名、またはダイアログを表示する場合は ""
        String pathName                         // パス名、またはダイアログを表示する場合は ""

        // ウェーブを読み込み、既存のウェーブを上書きする
        LoadWave/J/D/O/N/P=$pathName filename
        if (V_flag==0)                          // ウェーブが読み込まれない。おそらくユーザーがキャンセル
                return -1
        endif

        Textbox/C/N=TBFileName/A=LT "Waves loaded from " + S_fileName

        return 0                                // 成功
End
```

これには微妙な変更点があります。
LoadWave コマンドで /N オプションを使いました。
これにより入力ウェーブが自動的に wave0、wave1、wave2 と命名されます。

この手法はこれ以上ないほどシンプルであることがお分かりいただけるでしょう。
欠点は、wave0 のような意味のない名前になってしまうことです。
LoadWave /B フラグを使えば、より適切な名前を指定できます。

Igor バイナリウェーブファイルまたはパックされた Igor エクスペリメントのデータからデータをロードする場合、LoadWave の代わりに LoadData コマンドを使用できます。
これは強力なコマンドで、特に実験の複数回実行によって生成されるような、同一構造のデータセットが複数存在する場合に役立ちます。
LoadData コマンドのヘルプを参照してください。

XY データの読み込みとグラフ作成

前のセクションの例では、ファイル内のすべての列を同じもの、つまりウェーブフォームとして扱いました。
XY データがある場合、状況は少し変わります。
ファイル内の列について、さらにいくつかの仮定を立てる必要があります。
例えば、２組の XY ペアを表す４列のファイル群があるとします。
最初の２列が最初の XY ペア、次の２列が２番目の XY ペアとなります。

このケースに対応するための関数の修正版を以下に示します。

```
Function LoadAndGraphXY(fileName, pathName)
        String fileName                 // 読み込むファイル名、またはダイアログを表示する場合は ""
        String pathName                 // パス名、またはダイアログを表示する場合は ""

        // ウェーブを読み込み、グローバルを設定
        LoadWave/J/D/O/P=$pathName filename
        if (V_flag==0)                  // ウェーブが読み込まれない。おそらくユーザーがキャンセル
                return -1
        endif

        // ウェーブの名前を文字列変数に入れる
        String sx0, sy0, sx1, sy1
        sx0 = StringFromList(0, S_waveNames)
        sy0 = StringFromList(1, S_waveNames)
        sx1 = StringFromList(2, S_waveNames)
        sy1 = StringFromList(3, S_waveNames)

        Wave x0 = $sx0                          // ウェーブ参照を作成
        Wave y0 = $sy0
        Wave x1 = $sx1
        Wave y1 = $sy1
        SetScale d 0, 0, "s", x0, x1            // ウェーブのデータ単位を設定
        SetScale d 0, 0, "V", y0, y1

        Display y0 vs x0                        // 新しいグラフを作成
        AppendToGraph y1 vs x1

        Textbox/A=LT "Waves loaded from " + S_fileName          // グラフに注釈を追加

        return 0                                // 成功
End
```

この関数とウェーブフォームベースの LoadAndGraph 関数の主な違いは、ウェーブフォームを XY ペアとしてグラフに追加する点です。
また、ウェーブフォームをウェーブフォームとしてではなく XY ペアとして扱うため、ウェーブフォームの X スケーリングを設定しません。

任意の数の XY ペアを処理できる、より汎用的な関数を書くことが可能です。
繰り返しになりますが、汎用性を高めることは複雑さを増すことになります。
以下に、より汎用的なバージョンの関数を示します。

```
Function LoadAndGraphXY(fileName, pathName)
        String filename                 // 読み込むファイル名、またはダイアログを表示する場合は ""
        String pathname                 // パス名、またはダイアログを表示する場合は ""

        // ウェーブを読み込み、グローバルを設定
        LoadWave/J/D/O/P=$pathName filename
        if (V_flag==0)                  // ウェーブが読み込まれない。おそらくユーザーがキャンセル
                return -1
        endif

        Display                         // 新しいグラフを作成

        String sxw, syw
        Variable index=0
        do                              // グラフにウェーブをアペンド
```

```
                sxw=StringFromList(index, S_waveNames)          // 次の名前
                if (strlen(sxw) == 0)                           // 次があるか？
                        break                                   // ループの外に出る
                endif
                syw=StringFromList(index+1, S_waveNames)        // 次の名前

                Wave xw = $sxw                                  // ウェーブ参照を作成
                Wave yw = $syw

                SetScale d 0, 0, "s", xw                // X ウェーブの単位を設定
                SetScale d 0, 0, "V", yw                // Y ウェーブの単位を設定
                AppendToGraph yw vs xw

                index += 2
        while (1)                                       // 無条件に "do" までループバック

        // グラフに注釈を追加
        Textbox/A=LT "Waves loaded from " + S_fileName

        return 0                                        // 成功
End
```

フォルダー内のすべてのファイルを読み込む

次の例では、複数のファイルを含むフォルダーがあると仮定します。
各ファイルにはウェーブフォームデータの３列が含まれています。
フォルダー内の各ファイルを読み込み、グラフを作成して出力します。
この例では LoadAndGraph 関数をサブルーチンとして使います。

```
Function LoadAndGraphAll(pathName)
        String pathname                                 // シンボリックパス名、またはダイアログを表示する場合は ""

        String filename
        String graphName
        Variable index=0

        if (strlen(pathName)==0)                        // パスが指定されていない場合、作成
                NewPath/O temporaryPath                 // ダイアログが表示
                if (V_flag != 0)
                        return -1                       // ユーザーがキャンセル
                endif
                pathName = "temporaryPath"
        endif

        Variable result
        do                                              // フォルダー内の各ファイルをループ
                fileName = IndexedFile($pathName, index, ".dat")
                if (strlen(fileName) == 0)                              // さらにファイルがあるか？
                        break                                           // ループの外に出る
                endif
                result = LoadAndGraph(fileName, pathName)
                if (result == 0)                                        // LoadAndGraph が成功したか？
                        // グラフを出力
                        graphName = WinName(0, 1)                       // 最前面のグラフの名前を取得
                        String cmd
                        sprintf cmd, "PrintGraphs %s", graphName
                        Execute cmd                                     // 下記で説明

                        KillWindow $graphName                   // グラフをキル
                        KillWaves/A/Z                           // すべての未使用のウェーブをキル
                endif
                index += 1
        while (1)
```

```
        if (Exists("temporaryPath"))                    // 存在する場合は一時パスをキル
                KillPath temporaryPath
        endif

        return 0                        // 成功
End
```

この関数は、特定のフォルダー内の特定の種類の連続するファイル名を見つけるために IndexedFile 関数に依存しています。
IndexedFile の最後のパラメーターは、拡張子が「.dat」のファイルを探していることを示しています。

ファイル名を取得したら、LoadAndGraph 関数に渡します。
グラフを出力した後、そのグラフを終了させ、現在のデータフォルダー内の全てのウェーブを削除します。
これにより、次のファイルで新たに開始できます。
より洗練されたバージョンでは、グラフ内のウェーブのみを削除します。

グラフを印刷するには、PrintGraphs コマンドを使います。
PrintGraphs は、関数内で直接使用できない数少ない組み込みコマンドの１つです。
したがって、PrintGraphs コマンドを文字列変数に格納し、Execute を呼び出して実行します。

Igor バイナリウェーブファイルまたはパックされた Igor エクスペリメントデータからデータをロードする場合は、LoadData コマンドを使用できます。
LoadData コマンドのヘルプを参照してください。

データファイル読み込み時のウェーブの名前設定

このセクションでは、区切り文字付きテキストファイルから読み込んだウェーブデータの名前の設定方法をプログラムで実現する方法を示します。
背景の情報については、「LoadWave によるウェーブ名生成」のセクションを参照してください。

ファイルには列ラベルのない３列の数値が含まれていると仮定し、Stimulus、CellA、CellB という名前のウェーブを作成します。
LoadWave /B フラグを使ってウェーブ名を設定します。

```
Function/S GetColumnInfoStr1()
        String columnInfoStr = ""
        columnInfoStr += "N='Stimulus';"
        columnInfoStr += "N='CellA';"
        columnInfoStr += "N='CellB';"
        return columnInfoStr
End

Function LoadAndSetNames1(pathName, fileName)
        String pathName                 // シンボリックパス名、またはダイアログを表示する場合は〝〟
        String fileName                 // ファイル名、またはダイアログを表示する場合は〝〟

        String columnInfoStr = GetColumnInfoStr1()
        LoadWave/J/O/P=$pathName/B=columnInfoStr filename
        if (V_Flag == 0)
                return -1               // 失敗
        endif

        return 0                        // 成功
End
```

次に、読み込むファイルの名前（拡張子を除く）をウェーブ名に含めます。
「Data.txt」というファイルの場合、Data_Stimulus、Data_CellA、Data_CellB という名前のウェーブが生成されます。

```
Function/S GetColumnInfoStr2(String baseName)
        String columnInfoStr = ""
        columnInfoStr += "N='" + baseName + "_" + "Stimulus';"
        columnInfoStr += "N='" + baseName + "_" + "CellA';"
```

```
        columnInfoStr += "N='" + baseName + "_" + "CellB';"
        return columnInfoStr
End

Function LoadAndSetNames2(pathName, fileName)
        String pathName                 // シンボリックパス名
        String fileName                 // ファイル名

        // このバージョンでは、実際のシンボリックパスとファイル名を指定する必要がある
        if (strlen(pathName)==0 || strlen(fileName)==0)
                return -1                               // 失敗
        endif

        String fileNameMinusExtension = ParseFilePath(3, fileName, ":", 0, 0)
        String baseName = CleanupName(fileNameMinusExtension, 0)

        String columnInfoStr = GetColumnInfoStr2(baseName)
        LoadWave/J/A/O/P=$pathName/B=columnInfoStr filename
        if (V_Flag == 0)
                return -1                               // 失敗
        endif

        return 0                                        // 成功
End
```

最後に、Igor Pro 9.0 以降で利用可能な LoadWave /NAME フラグを使い、読み込むファイル名（拡張子を除く）をウェーブ名に含めます。
「Data.txt」というファイルの場合、前の例と同様に Data_Stimulus、Data_CellA、Data_CellB という名前のウェーブが生成されます。

```
Function/S GetColumnInfoStr3()
        String columnInfoStr = ""
        columnInfoStr += "N='Stimulus';"
        columnInfoStr += "N='CellA';"
        columnInfoStr += "N='CellB';"
        return columnInfoStr
End

Function LoadAndSetNames3(pathName, fileName)
        String pathName                 // シンボリックパス名、またはダイアログを表示する場合は ""
        String fileName                 // ファイル名、またはダイアログを表示する場合は ""

        String columnInfoStr = GetColumnInfoStr3()
        LoadWave/J/A/O/P=$pathName/B=columnInfoStr/NAME={":filename:_","",1} filename
        if (V_Flag == 0)
                return -1               // 失敗
        endif

        return 0                        // 成功
End
```

/NAME フラグの詳細については、「ウェーブ名でファイル名を使う」のセクションを参照してください。

データのエクスポート

エクスペリメントを保存すると、Igor は現在のエクスペリメントのウェーブを自動的にディスクに保存します。
多くの Igor ユーザーは、ファイルからデータを Igor に読み込み、グラフやレイアウトを作成して印刷します。
これで処理は完了です。
ウェーブを明示的に保存する必要はありません。

Igor のパックされたエクスペリメントファイルにウェーブを保存してアーカイブするには、SaveData コマンドを使うか、Data Browser の Save Copy ボタンを使います。
パックされたエクスペリメントのデータは、LoadData コマンドまたは Data Browser の Load Expt ボタンを使って Igor に再読み込みできます。

または、File→Open Experiment を使ってファイルをエクスペリメントとして読み込むこともできます。
詳細は SaveData コマンドのヘルプを参照してください。

エクスペリメントデータとは別にウェーブを保存する主な理由は、Igor から別のプログラムへデータをエクスポートするためです。
ウェーブを明示的にディスクに保存するには、Igor の Save コマンドを使います。

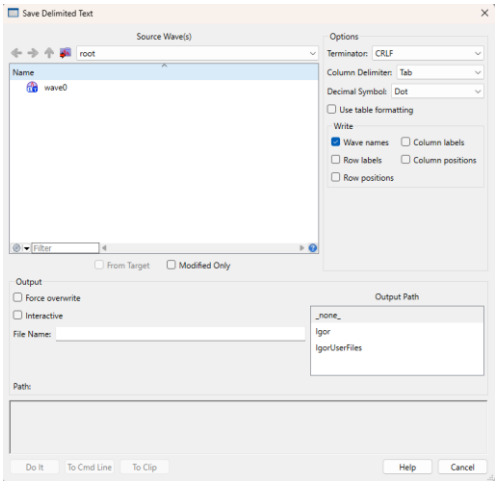
Data メニューの Save Waves サブメニューから、すべての組み込みルーチンにアクセスできます。

以下の表は、Igor で利用可能なデータ保存ルーチンとその主な特徴を一覧表示したものです。

ファイル形式	**説明**
区切り記号付きテキスト	結果のアーカイブ化や他のプログラムへのエクスポートに使われます。 フォーマット：<data><delimiter><data><terminator> 任意の行数と列数を持つ１つのデータブロックを含みます。 列ラベルの行はオプションです。 列の長さは等しくても不等でもかまいません。 1D または 2D のウェーブを出力できます。 「区切り記号付きテキストファイルにウェーブを保存」のセクションを参照してください。
一般的なテキスト	結果のアーカイブ化や他のプログラムへのエクスポートに使われます。 フォーマット：<number><tab><number><terminator> 任意の行数と列数を持つ１つ以上の数値ブロックを含みます。 列ラベルの行はオプションです。 ブロック内の列は長さが等しくなければなりません。 1D または 2D のウェーブを出力できます。 「一般的なテキストファイルにウェーブを保存」のセクションを参照してください。
Igor テキスト	ウェーブのアーカイブ化、またはある Igor エクスペリメントから別の Igor エクスペリメントへのウェーブエクスポートに使われます。 フォーマット：「Igor テキストファイル形式」のセクションを参照してください。 １つ以上のウェーブブロックを含み、各ブロックは任意の数のウェーブと行を持ちます。 特定のブロックは数値データまたはテキストデータのいずれかを格納できます。 特別な Igor キーワード、数値、および Igor コマンドで構成されます。 1D から 4D のウェーブをエクスポートできます。 「Igor テキストファイルにウェーブを保存」のセクションを参照してください。
Igor バイナリ	ある Igor エクスペリメントから別のエクスペリメントへウェーブをエクスポートするために使われます。 １つの Igor ウェーブデータを含みます。 フォーマット：Igor Technical Note #003, "Igor Binary Format" を参照 「Igor バイナリウェーブファイルにウェーブを保存」のセクションを参照してください。

画像	別のプログラムへウェーブをエクスポートするために使われます。 フォーマット：TIFF、PNG、raw PNG、JPEG. 「画像ファイルにウェーブを保存」のセクションを参照してください。
HDF4	Igor は HDF4 形式でのデータエクスポートをサポートしていません。
HDF5	ヘルプ HDF5 in Igor Pro を参照してください。
音声	別のプログラムへウェーブをエクスポートするために使われます。 フォーマット：AIFC、WAVE 「サウンドファイルの保存」のセクションを参照してください。
TDMS	データを National Instruments TDMS ファイルに保存します。 Igor 拡張機能の有効化が必要です。 詳細はヘルプ TDM XOP を参照してください。
SQL データベース	データを SQL データベースに書き込みます。 Igor 拡張機能の有効化とデータベースプログラミングの専門知識が必要です。 「SQL データベースへのアクセス」のセクションを参照してください。

区切り記号付きテキストファイルにウェーブを保存



区切り記号付きテキストファイルを保存するには、Data→Save Waves→Save Delimited Text を選択し、Save Delimited Text ダイアログを表示します。
区切り記号付きテキストの保存ルーチンは、タブまたは選択した別の区切り記号で区切られた数値で構成されるファイルに書き込みます。
各テキスト行の末尾には、選択可能な行終端文字が付けられます。
１次元ウェーブを書き込む場合、列ラベルの行をオプションで含めることができます。
行列を書き込む場合、列ラベルに加えて行ラベル、および行と列の位置情報をオプションで書き込むことができます。

区切り記号付きテキストの保存は、任意の次元のウェーブを保存できます。
多次元ウェーブは、１ブロックあたり１ウェーブで保存されます。
データは行/列/レイヤー/チャンクの順序で書き込まれます。
区切り記号付きテキストとして保存された多次元ウェーブは、区切り記号付きテキストとして Igor に再読み込みすることはできません。
これは、Load Delimited Text ルーチンが複数のブロックをサポートしていないためです。
それらは一般的なテキストとして再読み込みすることは可能です。
ただし、後で Igor に再読み込みすることを意図したデータについては、Igor Text、Igor Binary、または Igor Packed Experiment 形式が推奨されます。

ファイル内の列の順序は、Save コマンドでウェーブ名が表示される順序に依存します。
このダイアログは、Source Waves リストでウェーブを選択した順序に基づいてウェーブ名を生成します。

デフォルトでは、Save コマンドは数値データを倍精度データには「%.15g」形式で、精度が低いデータには「%.7g」形式で書き込みます。
これらの形式により、ファイル内で最大 15 桁または 7 桁の精度が得られます。

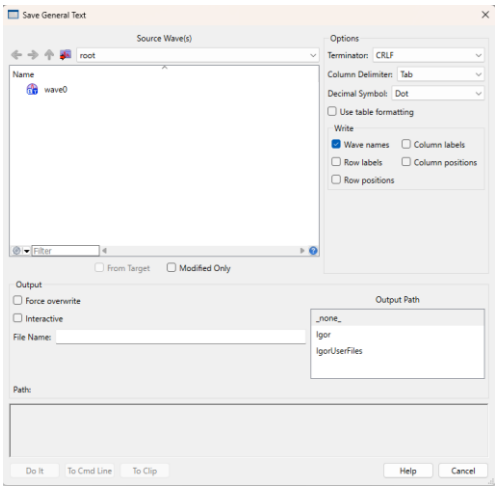
異なる数値書式を使うには、エクスポートしたいデータのテーブルを作成します。
テーブルの列の数値書式を必要に応じて設定します。
データはテーブルに表示されているとおりにファイルに書き込まれるため、テーブルには十分な桁数を表示してください。
Save Delimited Text ダイアログで、Use table formatting チェックボックスを選択します。
複数列ウェーブ（1D 複素数ウェーブまたは多次元ウェーブ）を保存する場合、ウェーブの最初のテーブルの列のテーブル形式を使って、ウェーブのすべての列が保存されます。

SaveTableCopy と wfPrintf コマンドは、特定の数値形式を使ってウェーブをテキストファイルに保存するためにも使用できます。

Save コマンドでは、既存ファイルを上書きするのではなく、ファイルの末尾に追加することが可能です。
これは、定期的に実行する分析結果を 1 つのファイルに蓄積するのに便利です。
また、fPrintf コマンドで生成したヘッダー情報を含むファイルに数値ブロックを追加するのにも使用できます。
ダイアログ経由では追加オプションは利用できません。
この操作を行うには、Save コマンドの説明を参照してください。

一般的なテキストファイルにウェーブを保存

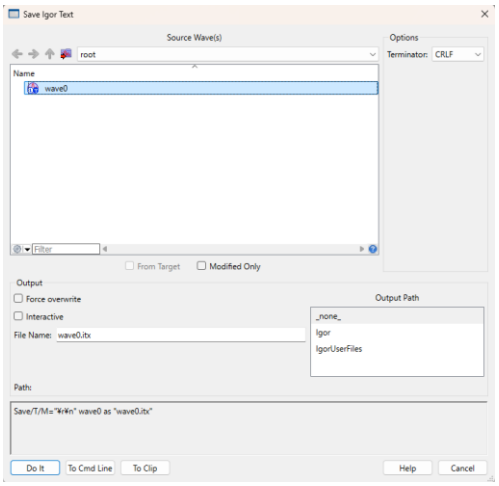
一般的なテキストファイルへのウェーブ保存は、区切り記号付きテキストファイルへの保存と非常に似ています。
Save General Text ダイアログは、Save Delimited Text ダイアログと同じです。



一般的なテキストファイルの 1 つのブロック内のすべての列は、同じ長さでなければなりません。
Save General Text ルーチンは、指定されたすべてのウェーブを保存するために必要な数のブロックを書き込みます。
例えば、100 ポイントの 1D ウェーブ 2 つと 50 ポイントの 1D ウェーブ 2 つを保存するように指示した場合、2 つのデータブロックが書き込まれます。
多次元ウェーブは、1 ブロックに 1 つのウェーブが書き込まれます。

Igor テキストファイルにウェーブを保存

Igor Text 形式は、ウェーブデータだけでなく、その他のプロパティも保存することができます。
各ウェーブの次元スケーリング、単位、ラベル、データのフルスケールと単位、およびウェーブのノート（存在する場合）を保存します。
SaveData コマンドを使ってパックされたエクスペリメントとして保存すると、これらのデータはバイナリデータとしてより効率的に保存されます。



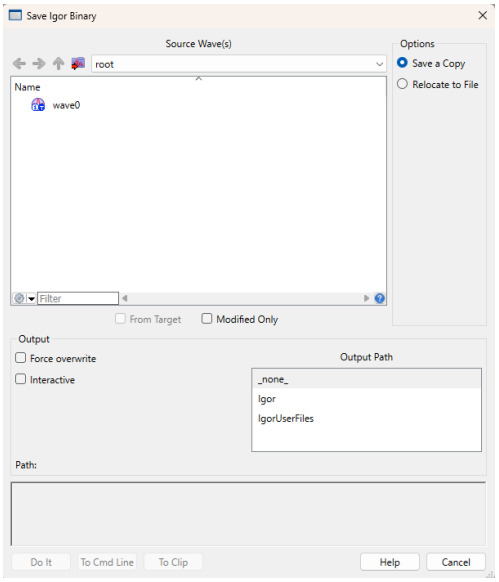
一般的なテキスト形式と同様に、Igor Text ファイルの 1 つのブロック内のすべての列は同じ長さでなければなりません。
Save Igor Text ルーチンは、必要な数のブロックを書き込むことでこの要件を処理します。

Save Igor Text は任意の次元のウェーブを保存できます。
多次元ウェーブはブロックごとに 1 ウェーブずつ保存されます。

ブロック先頭の /N フラグがウェーブの次元を識別します。
データは行/列/レイヤー/チャンクの順で書き込まれます。

Igor バイナリウェーブファイルにウェーブを保存

Save Igor Binary ルーチンは、ウェーブを Igor バイナリウェーブファイルに保存します。
１ファイルにつき１ウェーブです。
ほとんどのユーザーは、Igor エクスペリメントを保存すると自動的にウェーブが保存されるため、この操作は必要ありません。
同僚に送るために、ウェーブを Igor バイナリウェーブファイルに保存したい場合があるかもしれません。



Save Igor Binary ダイアログは Save Delimited Text ダイアログと似ています。
ファイル名の付け方には違いがあります。
Igor Binary の場合、各ウェーブが個別のファイルに保存されるためです。
ダイアログのリストから１つのウェーブを選択した場合、ファイル名を入力できます。

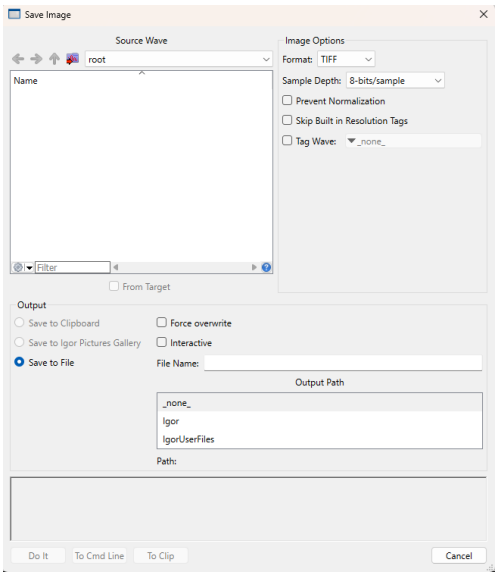
ただし、複数のウェーブを選択した場合、ファイル名を入力することはできません。
Igor は「wave0.ibw」という形式のデフォルトのファイル名を使います。

エクスペリメントをパックされたエクスペリメントファイルに保存すると、すべてのウェーブが Igor バイナリ形式で保存されます。
その後、Data Browser または LoadData コマンドを使って、別の Igor エクスペリメントにこれらのウェーブを読み込むことができます。

.ibw ファイルは 20 億要素を超えるウェーブをサポートしていません。
代わりに、非常に大きなウェーブをパックされたエクスペリメントファイル（.pxp）に保存するには、SaveData コマンドまたはデータブラウザーの Save Copy ボタンを使用できます。

画像ファイルにウェーブを保存

ウェーブを TIFF、PNG、raw PNG、または JPEG 形式で保存するには、Data→Save Waves→Save Image を選択し、Save Image ダイアログを表示します。



JPEG は非可逆圧縮を使い、TIFF、PNG、raw PNG は可逆圧縮を使います。
圧縮による画質劣化を避けるには、JPEG を使わないでください。

JPEG はサンプルあたり 8 ビットのみをサポートします。

PNG はサンプルあたり 24 ビットおよび 32 ビットをサポートします。
Raw PNG はサンプルあたり 8 ビットおよび 16 ビットをサポートします。

拡張 TIFF ファイル形式は、サンプルあたり 8 ビット、16 ビット、32 ビットをサポートし、画像スタックを 3D および 4D ウェーブでエクスポートできます。

詳細は ImageSave コマンドのヘルプを参照してください。

サウンドファイルの保存

SoundSaveWave コマンドを使って、ウェーブをサウンドファイルとして保存できます。

テキストウェーブのエクスポート

Igor は、テキストウェーブを区切り記号付きテキストファイルまたは一般的なテキストファイルとしてエクスポートする時、テキストを引用符で囲みません。
Igor テキストファイルとしてエクスポートする時は、テキストを引用符で囲みます。

タブ、キャリッジリターン、ラインフィードなどの特定の特殊文字は、プログラム間のデータ交換時に問題を引き起こします。
ほとんどのプログラムは、これらの文字を、ある値と次の値、あるいはあるテキスト行と次のテキスト行を区切るものとして扱うためです。
Igor Text ウェーブには、特殊文字を含むあらゆる文字を含めることができます。
ほとんどの場合、これは問題にはなりません。テキストウェーブに特殊文字を保存する必要がないか、あるいは保存する場合でも、それらを他のプログラムにエクスポートする必要がないためです。

Igor がテキストウェーブを含むテキストファイルを作成する時、ウェーブの内部で以下の文字が現れると、それらは関連するエスケープコードに置換されます：

文字	名前	ASCII コード	エスケープシーケンス
CR	キャリッジリターン	13	\r
LF	ラインフィード	10	\n
tab	タブ	9	\t
\	バックスラッシュ	92	\\

Igor がこれを行うのは、エスケープシーケンスに変換しないと誤解釈されるためです。
Igor がテキストファイルをテキストウェーブに読み込む時、このプロセスを逆行させ、エスケープシーケンスを対応する ASCII コードに変換します。

このエスケープコードの使用は、Save コマンドの /E フラグを使って抑制できます。
これは、バックスラッシュを含むテキストをエスケープコードを解釈しないプログラムにエクスポートするために必要です。

現在、テキストファイルの書き込み時には、Save コマンドは常に UTF-8 テキストエンコーディングを使います。
ウェーブに非 ASCII テキストが含まれており、UTF-8 をサポートしていないプログラムにインポートする必要がある場合は、保存後にファイルのテキストエンコーディングを変換する必要があります。
これを行うには、ノートブックとしてファイルを開き、テキストエンコーディングを変更して再度保存するか、外部テキストエディタを使います。

多次元ウェーブのエクスポート

多次元ウェーブを区切り記号付きテキストファイルまたは一般的なテキストファイルとしてエクスポートする時、行ラベル、行位置、列ラベル、列位置をファイルに書き込むオプションがあります。

これらの各オプションは、Save Waves ダイアログ内のチェックボックスでコントロールされます。
行/列ラベルと位置については、「2D ラベルと位置の詳細」のセクションで説明しています。

Igor は多次元ウェーブを列/行/レイヤー/チャンクの順に書き込みます。

SQL データベースへのアクセス

Igor Pro には SQL XOP と呼ばれる XOP が含まれており、Igor プロシージャからリレーショナルデータベースへのアクセスを提供します。
このアクセスを実現するために、ODBC（Open Database Connectivity）ライブラリとドライバを使います。

SQL XOP の設定と使用方法の詳細については、ヘルプ SQL XOP を参照してください。