LabTalk スクリプトガイド

最終更新 2019 年 4 月

このマニュアルは、(株)ライトストーンの協力により、翻訳・制作したものです。

OriginLab Corporation
One Roundhouse Plaza
Northampton, MA 01060
USA
(413) 586-2013
(800) 969-7720
Fax (413) 585-0126
www.OriginLab.com

目次

1 LabTalk スクリプトガイド

このガイドでは、Origin 内のスクリプト言語である LabTalk を紹介します。LabTalk は、データを解析またはグラフを作図するうえでスクリプトを記述、実行するユーザ向けに設計されています。このガイドは、Origin のスクリプト機能を利用し、スクリプト言語でのプログラミングに慣れているユーザのためのヘルプ機能です。Origin の基本的な知識があり、各々のニーズにあったソフトウェアのカスタマイズを始めようとしているユーザに対して、詳しい情報を提供しています。

ガイドは LabTalk の簡単な紹介から始まり、続いて言語の基本的な説明と、Origin の環境でスクリプトを整理、実行する様々な方法を紹介する章が続きます。残りの章は Origin の機能、例えばインポート、グラフ作成、データ分析、ユーザからの入力、自動化ごとに整理されます。

最後にいくつかのリファレンステーブルを含みます。しかし、このガイドは完全なる言語のリファレンスではありません。完全な LabTalk 言語のリファレンス文書は、Origin 内の**ヘルプ**メニューからアクセスできます。Origin のバージョンが新しくなると共に、LabTalk の新しい機能は追加されますので、言語リファレンスヘルプファイルには通常、バージョン番号のスタンプ（例えば 8.1、通常太字または赤文字）が記されています。

このガイドは、LabTalk を学ぶ際に他の資料と共に使用してください。他の資料については、**LabTalk を学ぶ上でのリソース**章を参照してください。

このガイドでは、複数のスクリプトサンプルを提供します。これらのサンプルを試すには、スクリプトを手動で入力するか、Origin を起動し、**ヘルプ**メニューからアクセスできるヘルプファイルからコピー＆ペーストし、ご利用ください。

- Excel と一緒に操作する
- Origin で R を使う
- Python を使って操作する
- 自動化とバッチ処理
- 関数リファレンス
- 索引

2 LabTalk を使いましょう

内容

2.1. Hello World

まず、よく使用されるサンプルを使用し、LabTalk スクリプトを実行する方法を紹介します。

1. Origin を開き、**ウィンドウ**メニューから、**スクリプトウィンドウ**を選択し、**スクリプトウィンドウ**を開きます。
2. このウィンドウで、次のように入力し、Enter キーを押します。

```
type "Hello World"
```

3. ここで LabTalk コマンド **type** を使用しました。コマンドは 2 文字の省略形でも記述できます。その場合、次のコマンドを実行してください。

```
ty "Hello World"
```

Origin は、コマンドの下に直接 **Hello World** という文字を出力します。

Enter を押すと Origin はウィンドウのスクリプト行の最後にセミコロン(;)を追加して実行します。
このスクリプト行を再実行するには、カーソルをその行のどこかに移動し、もう一度 Enter を押してください。
カーソルを行の最後に移動する場合、Enter を押してスクリプトを実行する前に、セミコロン(;)を削除する必要があります。

では、スクリプトウィンドウで複数の行を実行する方法を確認しましょう。

1. Origin 内でワークブックがアクティブな状態で **スクリプトウィンドウ**を開きます。

2. スクリプトウィンドウに次の 3 行を入力します。各行の最後に;を入力してから、Enter キーを押します。これにより、それぞれの行が実行されることなく複数行のスクリプトを入力できます。後から全ての行を一度に実行します。

```
type "The current workbook is %h";
type "This book has $(page.nlayers) sheet(s)";
type "There are $(wks.ncols) columns in the active sheet";
```

3. マウスでドラッグし、すべてのスクリプト行を選択します。キーボードを使用している場合、スクリプトの最初にカーソルを合わせ、Shift キーを押しながら矢印キーを使用して全ての行を選択します。
4. 選択した全てのスクリプト行を実行するために Enter キーを押します。現在どのワークブックがアクティブであるかによって、スクリプトウィンドウに出力される情報は異なりますが、以下のような出力になります。

```
The current workbook is Book1
This book has 1 sheet(s)
There are 2 columns in the active sheet
```

上記の例の場合、**%H** という文字列レジスタを使用しました。これは現在アクティブなウィンドウ名（ワークブック、行列、グラフのいずれか）を保持します。そして、LabTalk オブジェクトの **page** と **wks** を使用し、ブックのシートの数と、シート内の列数を表示しました。**$()**は、置換表記の 1 つで、**()**内の変数を評価し、その値を返すよう Origin に通知します。LabTalk:置換表記

スクリプトウィンドウに複数の行を入力する場合、各行の最後に;を追加してから Enter を押すと、その行の実行せずに改行する事ができます。これにより、それぞれの行で実行せず、すべての行を入力してから一括で実行できます。

2.2. = 記号を使った素早い出力

スクリプトウィンドウは対話形式で結果を返す計算機のように使うことができます。スクリプトウィンドウに、以下のスクリプトを入力し、Enter キーを押します。LabTalk:インタラクティブな実行

```
3 + 5 =
```

Origin は計算を行い、結果を次の行に出力します。

```
3 + 5 = 8
```

= 記号は、通常、左辺と右辺を持ち、代入演算子として使われます。右辺が無い場合、インタラプタが＝の左側（左辺）の式を評価し、スクリプトウィンドウに結果を出力します。

以下の例では、LabTalk での変数の考え方を説明します。例えば、スクリプトウィンドウに次の代入文を入力します。LabTalk:データ型と変数

```
double A = 2
```

これは変数 A を作成し、A に 2 が割り当てられます。そして、変数 A に対して PI(Origin で定義される定数, π) で掛け算するなどの算術演算を行い、変数 A に再割り当てすることができます。

```
A = A*PI
```

A の現在の値を表示するには、次のように入力します。

```
A =
```

Enter キーを押すと、次のように値が返ってきます。

```
A = 6.2831853071796
```

さらに、変数や変数の値のリストを見る List コマンドもあります。以下のコマンドを入力して Enter を押します。

```
list
```

Origin は、**LabTalk の変数と関数**ダイアログを開き、Origin の変数をすべて表示します。
また、特定のタイプの変数を切り捨てる事もできます。

```
list v
```

数値変数をリストするにはこれを使用します。

2.3. スクリプトを実行する他の方法

先程の例題では、スクリプトを**スクリプトウィンドウ**から実行しました。Origin は他にも LabTalk スクリプトを整理して実行する機能を備えています。詳細については **LabTalk スクリプトの実行とデバッグ**の章を確認してください。ここでは、スクリプトを実行するメソッド、(1)ツールバーボタンにある**カスタムルーチン**、(2)カスタムメニューアイテム、(3)グラフページのボタンを紹介します。

2.3.1. カスタムルーチン

Origin には、スクリプトを保存し、ツールバーのボタンを押すだけで実行する便利な方法があります。

1. キーボードで **Ctrl+Shift** キーを押しながら、標準ツールバーの**カスタムルーチン**ボタン () を押します。
2. これにより Origin の標準のスクリプトエディタである**コードビルダ**が開きます。編集しているファイルは、**Custom.ogs** です。コードには、1 つのセクション **[Main]**があり、以下のスクリプト 1 行を含みます。

   ```
   [Main]
   type -b $General.Userbutton;
   ```

3. これを次の行で置き換えます。

   ```
   [Main]
   ```

```
type -b "Hello World";
```

4. そして、**コードビルダ**ウィンドウの**保存** ()ボタンをクリックします。
5. Origin のウィンドウに戻り、  ボタンをクリックします。

これにより、Origin は再び Hello World のテキストを出力しますが、ここでは **type** コマンドに**-b** スイッチが付いているので、ポップアップウィンドウに表示されます。

2.3.2. カスタムメニューアイテム

LabTalk スクリプトはカスタムメニューアイテムを実行できます。

1. メニューから**ツール**:**カスタムメニューオーガナイザ**を選択し、**カスタムメニューオーガナイザ**ダイアログを開きます。
2. **カスタムメニュー追加**タブを開きます。左側パネルの内部を右クリックし、**新しいメインポップアップ**をコンテキストメニューから選択します。
3. 右側のパネルでは、**ポップアップテキスト**に名前、例えば **My Menu** を入力してから編集ボックスの外側をクリックします。
4. 左側パネルから **My Menu** を選択してから右クリックし、**項目の追加**をコンテキストメニューから選びます。
5. 右側のパネルでは、**項目テキスト**を **Hello World** に変更し、次のスクリプトを **LabTalk スクリプト**ボックスに追加します。

```
type -b "Hello World";
```

6. **閉じる**ボタンをクリックした後、ポップアップするウィンドウでは **Yes** をクリックして、メニューの変更を**デフォルト**メニューに設定します。開いたファイルダイアログでは、**保存**ボタンを押し、デフォルト名でデフォルトフォルダ(ユーザファイルフォルダ)に保存します。
7. これで新しいメニュー(**My Menu**)がメニューバーの**ウィンドウ**メニューの左側に追加されます。この新しいメニューアイテムをクリックし、**Hello World** エントリーをドロップダウンから選択します。Hello World と表示されたダイアログがポップアップします。

2.3.3. グラフにボタンを追加する

Origin はグラフやワークシートにボタンを追加する事ができ、ボタンを押す事で LabTalk スクリプトの実行が可能です。これにより、スクリプトを特定のプロジェクトやウィンドウに保存できます。

1. 標準ツールバーの**新規グラフ**ボタン  をクリックし、新しいグラフを作成します。
2. プロット操作・オブジェクト作成ツールバーの中の**テキストツール**ボタン（  ）をクリックし、作成したグラフ上でクリックして **My Button** と入力します。テキストの入力が終わったら、テキストの外側をクリックして編集を終了します。
3. テキストを右クリックし、コンテキストメニューから、**オブジェクトのプログラム制御**を選択して**オブジェクトのプログラム制御**ダイアログを開きます。
4. ダイアログでは、**の後でスクリプトを実行**のドロップダウンリストから、**ボタンアップ**を選択し、以下のスクリプトを編集ボックスに入力します。

```
type -b "Hello World";
```

5. OK をクリックしてダイアログを閉じます。テキストラベルがボタンになりました。ボタンをクリックすると、Hello World と表示されたダイアログがポップアップします。

2.4. スクリプトサンプル

ここで紹介するスクリプトサンプルでは、インポートとデータ処理に関連する特定のシナリオをステップごとに実行し、プロジェクトを保存します。このサンプルでは、いくつかの LabTalk 言語、例えば**コマンド**、**オブジェクト**、**X ファンクション**に焦点を当てています。これらの言語についての詳細は、この後に続く章で説明します。

NOTE:ここでは、コマンド文を実行するために**スクリプトウィンドウ**を使用します。コードの 1 行分だけを実行する場合、; は入力せずに、Enter キーを押してください。複数行にわたるコードの場合、各行の末尾に;を入力してから Enter を押すと、そのまま入力を続ける事ができます。全ての行を入力後、Enter キーを押すと実行します。

それでは **doc** コマンドと**-n** スイッチを使用して、新しいプロジェクトを始めましょう。現在のプロジェクトを保存する必要がある場合、このコマンドはユーザに保存をするように促します。

```
doc -n
```

サンプルフォルダからデータファイルをインポートしましょう。インポートするファイルを参照するために、X ファンクション **dlgfile** を使用します。

```
dlgfile gr:=ASCII
```

インストールフォルダの、**\Samples\Import and Export** サブフォルダから、**S15-125-03.dat** を選択し、**開く**をクリックします。

上記操作により、ファイルのパスをロードし、**fname$**変数にファイル名が入力されます。この変数の値を確認するには、以下のコードを実行します。

```
fname$=
```

このファイルをアクティブなワークブックにインポートします。**impasc**X ファンクションを命名制御オプションと共に使用し、ファイル名はワークブックネイに設定されません。

```
impasc Options.Names.FNameToBk:=0
```

では、**Position** 列のデータを処理しましょう。まず、範囲変数を定義し、この列を指定します。

```
range rpos = "Position"
```

指定する列は、現在のワークシートの 4 番目の列なので、インデックス番号を使用し、以下のように定義することもできます。

```
range rpos = 4
```

次のコマンドを使用すると、範囲変数に現在定義されている値を確認できます。

```
list a
```

列データを正規化し、データが 0 から 100 までの値をになるようにします。正規化に必要な X ファンクションを確認するには、次のコマンドを使用します。

```
lx *norm*
```

上記コマンドにより、**norm** を名前に含む X ファンクションが表示されます。データを正規化する X ファンクションは複数あり、今回の例では、**rnormalize**X ファンクションを使用します。

この X ファンクションのシンタックスに関するヘルプを参照するには、以下のように入力します。

```
rnormalize -h
```

これにより、情報が一覧表示されます。あるいは、以下のように入力します。

```
help rnormalize
```

対応するヘルプファイルが開きます。

では、posotion 列を正規化しましょう。

```
rnormalize irng:=rpos method:=range100 orng:=<input>
```

正規化されたデータは同じ列、つまり、元のデータを置き換え、出力されます。これは出力範囲変数 **orng** を**<input>**に設定したためです。

X ファンクションを使用する場合、現在の順序で変数値を指定している時は、変数名を入力する必要はありません。つまり、上記コマンドは次のように省略して記述できます。

```
rnormalize rpos range100 orng:=<input>
```

ここでも **orng** を指定した理由は、この特定の変数に先立つ他の変数があるからです。それらの変数は、ここで行いたい計算には関係ないため、今回のコマンドには含めていません。

次に、プロジェクトの中のフォルダに変更を加えます。プロジェクトを管理するいくつかの X ファンクションがあり、そのうちのいくつかを使用します。

```
// 現在のワークシート名を入手
string name$ = wks.name$;
// ルートフォルダに移動
pe_cd ..;
// Folder1 をワークシートと同じ名前にする
pe_rename Folder1 name$;
```

では、ルートフォルダにある全てのサブフォルダとワークブックをリストしましょう。

```
pe_dir
```

最後に、Origin プロジェクトをユーザファイルフォルダに保存します。ユーザファイルフォルダの場所は文字列レジスタ**%Y** に保存されています。この変数を確認すると、ユーザファイルフォルダの場所を確認できます。

```
%Y =
```

save コマンドを使用し、現在のプロジェクトを MyProject.opj という名前でユーザファイルフォルダに保存します。

```
save %yMyProject
```

上記コマンドで**%Y** はユーザファイルフォルダのパスに置き換えられ、このプロジェクトはこのパスに保存されます。

2.5. これから行うこと

LabTalk を習得し、利用するには、LabTalk スクリプトガイドの残りの項目を学習することが重要です。上述の例は、表面的なものですが、LabTalk の学習を始めるにあたっては十分な情報といえます。次の章では、LabTalk を学ぶ上で便利なリソースの一覧を表示します。

3 LabTalk を学ぶ上でのリソース

このガイドにより提供される内容以外で、LabTalk を学ぶ際に下記のリソースが利用可能です。

内容

3.1. オンラインドキュメント

このガイドの更新分を含む、最新の LabTalk のヘルプドキュメントです。次のリンク先で確認できます。
http://www.originlab.com/doc/labtalk

3.2. チュートリアル

いくつかの LabTalk チュートリアルが Origin と共にお手元に届きます。これらは**ヘルプ**メニューからアクセスできます。

3.3. スクリプトのサンプル

Origin には、様々なスクリプトのサンプルが付いています。これらは、**ヘルプ**メニューからアクセスできます。また、サブフォルダ**<Origin インストールフォルダ>\Samples\LabTalk Script Examples**にも含まれています。

3.4. X ファンクションのスクリプトサンプル

Origin 内で **F11** キーを押すと、**XF スクリプトダイアログ**が開きます。このダイアログは、特に X ファンクションを呼び出すための多くのスクリプトを提供します。これらの関数はインポート、フィット、信号処理、スペクトル解析などのカテゴリに分けられています。

3.5. LabTalk フォーラム

質問がある場合、LabTalk フォーラムをご利用ください。http://www.originlab.com/forum ページを開き、**LabTalk Forum** を選択します。このフォーラムは、OriginLab 社の技術担当者もがモニタリングしている他、ハードユーザからアイデアや答えが提示されるかもしれません。

3.6.トレーニングとコンサルティング

OriginLab 社と、各国の販売代理店は LabTalk を使用するためのトレーニングとコンサルティングサービスを提供しています。詳細については、お問い合わせください。

4 アップグレードユーザ向け情報

内容

4.1. イントロダクション

これらのノートは、Origin のバージョン 8.0 での LabTalk スクリプト言語に加えられた変更や改善に関連しています。もしスクリプトコードを 7.5 などの古いバージョンで開発している場合は、新しい機能を活用するだけでなく、Origin の新しいバージョンでスクリプトの互換性の問題を修正するようにコードを更新したい場合があります。

4.2. バージョン 7.x での主な制限事項

多くの LabTalk コマンドはワークシート、グラフデータや列など「アクティブな」オブジェクトで動作します。このため、ワークシートやグラフ（または 2 番目のワークシート）などのウィンドウを記憶することによってウィンドウを管理し、"window -a" コマンドを使用してこれらのウィンドウ間を行ったり来たり切り替えるためのスクリプトが頻繁に要求されます。

変数はスコープ内で宣言されていないとグローバル変数となっていました。

いくつかのコマンドは、データが消滅するため、元のデータが必要な場合は、コピーで作業をする必要があります。

4.3. バージョン 8.0.x での主な改善事項

3 つの追加事項により Origin8 では前項で説明されている問題が解決しています。LabTalk プログラマはそれらのコンセプトを知る必要があります。

- 変数の宣言とスコープ
- 範囲変数
- X ファンクション

宣言された変数は、それらが宣言されたプロシージャでのみコンテキストを持っています。

範囲変数は様々な Origin のオブジェクトを指し示すことができ、オブジェクトがアクティブである必要がなくなります。

X ファンクションは、範囲変数を入力、出力変数として受け入れ、分析結果からデータを分離し、自動更新や高レベルのプログラム機能を提供します。

4.4. 例

複数ファイルのデータの導関数の折れ線グラフを確認したいとします。

4.4.1. 過去の方法

Origin 7.5 では、複数 ASCIIA ファイルを新しいシート、新しい列または行にインポートできました。これらのオプションは、ワークシート内の特別な構造を読み込み、書き込むことで行われていましたが、後者の 2 つは、管理が難しいため、新しいシート（デフォルト）にします。どのシートが作成されるか知るのが難しいため、最初に新しい Origin プロジェクト内の空のワークシートを削除します。

次のようにインポートします。

```
win -cd %H; // デフォルトのワークシートウィンドウを削除
// ファイルを指定しインポート
run.section(file,MultiImportToolbar);
```

次のようにしてインポートされた全てのシートをループできます。

```
doc -e W { // 全ワークシートでループ
                ...
        }
```

しかし、必要な導関数のコマンドは元のデータを置き換えます。ここではインポートされたデータを保持したいので、データをコピーし、これを導関数の計算に使用します。

```
wo -a 1; // 列を追加
copy col(2) col(3); // 列 3 は列 2 のコピー
derivative col(3); // 列 3 は導関数で置換えられる
```

データをプロットするには、グラフウィンドウを作成し（グラフウィンドウの名前を憶えて置く必要があります）、全てのワークシートでループし、グラフに追加するデータの名前を取得し、データを追加するグラフに切り替えます。

その代わりに、ワークシートの最初のパスにいるときに変数(count)を使用し、グラフ作成のためにループします。そして、続くループで導関数データの名前だけに注意し、グラフにデータを追加するために一時的にグラフに切り替えます。

スクリプトは次のようになります。

```
win -cd %H;
run.section(file,MultiImportToolbar);
count = 1;
doc -e W {
        wo -a 1;
        copy col(2) col(3);
        derivative col(3);
        if( count == 1 )
        {
                wo -s 3 0 3 0;
```

```
                wo -p 200 LINE;
                %M = %H;
        }
        else
        {
                %N = col(3);
                win -o %M
                {
                        layer -i %N;
                }
        }
        count++;
}
win -a %M;
layer -a;
```

以下は、上述のスクリプトで(一時的に)アクティブになるウィンドウです。

Worksheet1
Worksheet2
Worksheet3
Worksheet1
Graph1
Worksheet1
Worksheet2
Graph1
Worksheet2
Worksheet3
Graph1
Worksheet3
Graph1

このシーケンスを理解することは、コードを記述する要件でした。

4.4.2. 現在の方法

次に範囲(range)と X ファンクションを使用して、新しいバージョンでどのように処理を行うかを見てみましょう。

ここでは、2 つの X ファンクションを使用します。最初のプロンプトは、インポート、次はファイルをどのようにインポートするかを指定します。

```
dlgfile gr:=*.dat mu:=1;
impASC options.ImpMode:=4 options.FileStruct.NumericSeparator:=0
options.Names.FNameToBk:=0;
```

以前のバージョンでの、"run.section(file,MultiImportToolbar);"は、2 つに分けられますが、"dlgfile"関数は、他の多くの場合でファイルの指定に使用できます。"impasc" X ファンクションは、引数の一部として様々なオプションを統合し、オプションが定義する必要のある時に多くの行を置換えます。

次に、作図に使用する文字列変数を宣言します。

```
string strRanges;
```

ワークブック内の各ワークシート(インポートされたファイル)をループします。

```
loop(ii,1,page.nlayers) // ワークブック(ページ)は、複数ワークシート(レイヤ)を持つ
{
                ...
}
```

ループ内では、導関数を計算するデータを指定する範囲を宣言し、differentiate X ファンクションで使用します。

```
range raD = $(ii)!(1,2); // 列 1 を X、列 2 を Y として使用したインデックスでシートを参照した範囲
differentiate iy:=raD; // デフォルトで新しい列を作成する X ファンクションに範囲を渡す
```

文字列変数は、各範囲をカンマ区切りリストとして統合する範囲シンタックスにあたる文字列の定義に使用されます。

```
if(ii == 1) strRanges$ = 1!(1,3);
else strRanges$ = %(strRanges$),$(ii)!(1,3);
```

ループが終了すると、文字列は次のようになります。

```
1!(1,3),2!(1,3),3!(1,3)
```

最後の行は括弧で囲みます。

```
strRanges$ = (%(strRanges$));
```

次のようになります。

```
(1!(1,3),2!(1,3),3!(1,3))
```

X ファンクション plotxy は、1 つのコマンドで最後のグラフを作成するために複数 XY 範囲として理解します。

最終的なコードは、次のようになります。

```
dlgfile gr:=*.dat mu:=1;
impASC options.ImpMode:=4 options.FileStruct.NumericSeparator:=0
options.Names.FNameToBk:=0;
string strRanges;
loop(ii,1,page.nlayers)
{
        range raD = $(ii)!(1,2);
        differentiate iy:=raD;
        if(ii == 1) strRanges$ = 1!(1,3);
        else strRanges$ = %(strRanges$),$(ii)!(1,3);
}
strRanges$ = (%(strRanges$));
plotxy %(strRanges$) plot:=200;
```

4.5. 不適合

言語の新機能への適合や問題の修正のための変更により、古いスクリプトが動作しない場合があります。

いくつかの修正方法は以下の通りです。

4.5.1. 文字列の代入

文字列変数へ値を代入する場合、変数名と'$'の間にスペースは必要ありません。

過去の方法

```
str $ = "Hello"; // 現在は動作しません
```

現在の方法

```
str$ = "Hello";
```

4.5.2. 文字列レジスタ

%D は以前は最後のセットのデータセットの名前が含まれていました。現在は、コマンドウィンドウやスクリプトウィンドウでコマンドを実行する場合には、現在の作業ディレクトリが含まれています。

4.5.3. 列の両端揃え

以前は"wks.col#.justify"プロパティで列の左/中央/右揃えを操作しました。

新バージョンでは以下のように操作します。

```
DoMenu 35152; // 左
DoMenu 35153; // 中央
DoMenu 35154; // 右
```

5 言語の基本

この章では、LabTalk 言語の構造に関する様々な要素を紹介します。最初のセクションでは、全般的な言語の機能、例えばデータ型、変数、演算子、条件付きループ構造、マクロ、関数等について学びます。2 番目のセクションでは、LabTalk 固有の機能、例えば範囲や置換表記、オブジェクト、メソッドと設定、X ファンクションにアクセスする方法などを紹介します。

この章で以下の項目を説明します

- 一般的な言語機能
- 特殊な言語機能
- LabTalk スプリクトの優先順位

5.1. 一般的な言語機能

LabTalk スクリプト言語の一般的な機能についての情報を確認できます。ほとんどすべてのプログラミング言語の機能と同様です。

このセクションで以下の項目を説明します

- データ型と変数
- プログラミングシンタックス
- 演算子
- 条件およびループ構造
- マクロ
- 関数

5.1.1. データ型と変数

内容

LabTalk のデータ型

LabTalk は 9 つのデータ型をサポートしています。

タイプ	コメント
Double	double 型の精度を持つ浮動小数点数
Integer	整数値
Constant	数値のデータ型は一度定義されると変更できません
Dataset	数値の配列
String	連続した文字
StringArray	文字列の配列
Range	Origin オブジェクト（ワークブック、ワークシート等）の特定の範囲を参照します。
Tree	枝(ブランチ)と葉(リーフ)を持つ木(ツリー)のような構造に倣ったもの
Graphic Object	オブジェクト、例えばラベル、矢印、線及びその他ユーザーによって作成されたグラフィックエレメントなどがあります。

Numeric

LabTalk は、次の 3 つの数値データ型 **double**、**int**、**const** をサポートしています。

1. Double: double 型の精度を持つ浮動小数点数。Origin のデフォルトの変数型です。
2. Integer: 整数 (**int**)は LabTalk では double 型で保存され、代入中に切り詰めされます。
3. Constant: 定数(**const**)は、LabTalk で利用できる 3 番目の数値データ型です。一度宣言されると、**constant** の値は変更できません。

```
// 新しい変数データ型として double を宣言:
double dd = 4.5678;
// 新しい integer 変数を宣言:
int vv = 10;
// 新しい constant を宣言:
const em = 0.5772157;
```

Note:LabTalk には、**complex** データ型はありません。列は数値複素数タイプとして設定でき、基本的な演算子 (+,-,*,/) を使用した列間の演算を行ったり、リテラル値を使用できます。複素数を入力した列値や文字式から実数部や虚数部を抽出する関数は準備されていますが、**複素数**変数が必要な場合、**OriginC** を利用してください。

```
// 直接的な文字式
(3-13i) / (7+2i) =;
// (3-13i) / (7+2i)=-0.094339622641509-1.8301886792453i
// 複素数式に文字列を割り当てる
col(A)[3] = (3-13i) * (7+2i);
// 複素数列または文字式の実数部分を取得する
dReal1 = imreal(col(A)[3]);
dReal2 = imreal((3-13i) + (7+2i));
// 複素数列または文字式の虚数部分を取得する
dImag1 = imaginary(col(A)[3]);
dImag2 = imaginary((3-13i) - (7+2i));
```

Dataset

Dataset データ型は、数値配列を保持するように設計されています。

一時データセット

dataset 変数を宣言すると、それはローカルな一時データセットとして内部的に保存されます。一時データセットは、Origin プロジェクトと一緒に保存されず、特定のワークシートとは結びついていません。一時データセットは、計算のみに使用され、プロット用に使用することはできません。LabTalk:Datasets

以下のサンプルでは、データ型の使用法を紹介します(Dataset 型の宣言と$ 置換表記を使用します。)

```
// 1 から 10 で増分 0.2 の値を持つ
// データセット'aa'を宣言
dataset aa={1:0.2:10};

// integer 型の 'nSize' を宣言し、
```

```
// これに新しい配列の長さを割り当て
int nSize = aa.GetSize();

// スクリプトウィンドウに 'aa' の値を出力
type "aa has $(nSize) values";
```

プロジェクトレベルの非接続データセット

vector データを代入(宣言せずに)または Create (コマンド) を使って、データセットを作成すると、それは、プロジェクトレベルのデータセットとなり、計算またはプロット用に使用することができます。

代入によるプロジェクトレベルデータセットの作成

```
bb = {10:2:100}
```

あるいは **Create コマンド**を使います

```
create %(strWks$) -wdn 10 aa bb;
```

プロジェクトレベルまたはローカルレベルの変数についての詳細は、以下の変数のスコープのセクションをご覧下さい。

データセットの操作については、Datasets をご覧下さい。

%( )の操作については、置換表記をご覧下さい。

String

LabTalk では、文字列変数および文字列レジスタの 2 つの方法で文字列を扱います。

文字列変数

文字列変数は、宣言と代入で作成されたり、代入のみ(定義された変数スコープによる)で作成されます。そして、LabTalk では連続した文字の名前に**$**記号を付けて表されます。(下記の命名規則を参照) (例、*stringName$*):

```
// 宣言と代入でローカル/セッションスコープを持つ文字列を作成

// 文字列 "greeting"を作成し
//  "Hello" という値を割り当て
string greeting$ = "Hello";

// 宣言では、$ 記号の付加は任意
string FirstName, LastName;
FirstName$ = Isaac;
LastName$ = Newton;

// 宣言無しで代入してプロジェクト文字列を作成
greeting2$ = "World";// グローバルスコープは OPJ ファイルと共に保存

// 文字列は文字列変数のクラスメソッドを使い作成
string str$ = Johann Sebastian Bach;
str.Find('Sebastian')=;
```

文字列変数の操作については、文字列の処理項目を参照してください。

文字列レジスタ

文字列を文字列レジスタに保存することもできます。これは、%記号にアルファベット文字を付けて表します。(例、%A-%Z)文字列レジスタは常のグローバルスコープになります。

```
/* 文字列レジスタ %A に文字列"Hello World"を割り当て */
%A = "Hello World";
```

Origin の現在のバージョンでは、文字列の操作については、いくつかの役立つメソッドがあるので、文字列変数を使用することをお勧めします。詳細は、String(オブジェクト)をご覧下さい。しかし、文字列レジスタを既に使っている場合、その使用法については、文字列レジスタをご覧下さい。

StringArray

StringArray データ型は、Datasets データ型が数値配列を扱うのと同じように、文字列の配列を扱います。String データ型と同様に、StringArray もいくつかの組込のメソッドがあります。詳細は StringArray (オブジェクト)をご覧下さい。

次のサンプルは、StringArray の使用方法を示すものです。

```
// "aa"という名前の文字列配列を宣言
// 組込のメソッド Add と GetSize を使用
StringArray aa;              // aa は空の文字列配列
aa.Add("Boston");            // aa に 1 つの要素を追加:"Boston"
aa.Add("New York");          // aa 2 つ目の要素を追加:"New York"

/* 各行に"aa has 2 strings in it:"と入力*/
type "aa has $(aa.GetSize()) strings in it:";
loop(ii,1,aa.GetSize())
{
        ty aa.GetAt(ii)$;
}
```

Range

range データ型は、ワークブック、ワークシート、グラフ、レイヤ、ウィンドウ等にある特定の領域を参照する Origin のデータに関連するオブジェクトにアクセスできます。

一般的なシンタックスは、

range *rangeName* = [*WindowName*]*LayerNameOrIndex*!*DataRange*[*subRange*]

これは、ワークブック、行列、グラフ内の特定のデータにアクセスします。

range *rangeName* = [*BookName*]*SheetNameOrIndex*!*ColumnNameOrIndex*[*RowBegin*:*RowEnd*]

range *rangeName* = [*MatrixBookName*]*MatrixSheetNameOrIndex*!*MatrixObjectNameOrIndex*[*CellBegin*:*CellEnd*]

range *rangeName* =[*GraphName*]*LayerNameOrIndex*!*DataPlotIndex*[*RowBegin*:*RowEnd*]

一時データセットにアクセスする range 変数を作成するためには特別なシンタックス **[*??*]** が使われます。

例えば、

```
// Book1 の Sheet2 の列 3 にアクセス
range cc = [Book1]Sheet2!Col(3);
// Graph1 のレイヤ 1 の 2 番目の曲線にアクセス
```

```
range ll = [Graph1]Layer1!2;
// MBook1 の MSheet1 の 2 番目の行列オブジェクトにアクセス
range mm = [MBook1]MSheet1!2;
// 非接続データセット tmpdata_a にアクセス
range xx = [??]!tmpdata_a;
```

Note:

- CellRange は、1 つのセル、行または列(の一部)、セルグループ、非連続な選択セルにすることができます。
- ワークシート、行列シート、グラフレイヤは、それぞれ、名前またはインデックスで参照することができます。
- range 変数 を定義して、origin オブジェクトを表したり、直接、X ファンクションの引数として range を使用することができます。
- **range データ型**および **range 変数**の詳細については、範囲表記にあります。

Tree

LabTalk は、標準的な tree データ型をサポートしており、これは枝(ブランチ)と葉(リーフ)を持つ木(ツリー)のような構造に倣ったものです。ブランチは複数のリーフを含み、それぞれのリーフにはデータが含まれます。ブランチとリーフを合わせたものノードと呼びます。

リーフ: 子ノードを持たないノード。値を含むことができます。

ブランチ: 子ノードを持つノード。値は含みません。

リーフノードは**数値**、**文字列**、**データセット**(ベクター)型を変数をもつことができます。

ツリーは、複数のパラメータをセットしたり、保存するために Origin で使われます。例えば、データセットを Origin ワークスペースにインポートするとき、**オプション**と呼ばれるツリーは、インポートをどのように実行するかを決定するパラメータを保持します。

具体的には、次のコマンドはファイル"SampleData.dat"から ASCII データをインポートし、インポートが扱われる方法を制御するため**オプションツリー**内の値をセットします。**ImpMode** リーフに 4 という値をセットすると、新しいワークシートにデータをインポートするよう Origin に通知されます。(Cols ブランチにある)NumCols リーフに 4 という値をセットすると、*SampleData.dat* ファイルの最初の 3 列のみをインポートするよう Origin に通知されます。

```
string str$ = system.path.program$ + "Samples\Graphing\Group.dat";
impasc fname:=str$
/* 新しいシートを開始 */
options.ImpMode:=4
/* 最初の 3 列のみインポート*/
options.Cols.NumCols:=3;
```

aa という名前のツリー変数を宣言します。

```
// 空のツリーを宣言
tree aa;
// 代入しているときにツリーノードが追加
aa.bb.cc=1;
aa.bb.dd$="some string";

// 新しいツリー 'trb' を宣言し、ツリー'aa'からのデータをそれに割り当てる
tree trb = aa;
```

tree データ型は、入力および出力データ構造の両方として、X ファンクションで頻繁に使われます。例えば:

```
// インポートするファイルの情報を 'trInfo'に割り当て
impinfo t:=trInfo;
```

ツリーノードは文字列にすることができます。以下の例は、文字列データのツリーノードをワークシート列にコピーする方法を示します。

```
//ワークシートにデータファイルをインポート
newbook;
string fn$=system.path.program$ + "\samples\statistics\automobile.dat";
impasc fname:=fn$;
tree tr;
//列の統計を実行し、その結果をツリー変数に保存する
discfreqs irng:=2 rd:=tr;
// ワークシート列に文字列を割り当てる
newsheet name:=Result;
col(1) = tr.freqcount1.data1;
col(2) = tr.freqcount1.count1;
```

ツリーノードはベクター型にすることもできます。**Origin 8.1 SR1** のより前までは、Tree 変数内のベクターにアクセスする唯一の方法は、以下のサンプルに示すように、直接代入することでした。

```
tree tr;
// データセットをツリーノードに代入すると
// 自動的にベクターノードになる:
tr.a=data(1,10);
// ベクターツリーノードは列に割り当てできる:
col(1)=tr.a;
// ベクターツリーノードは非接続データセットに代入でき、
//  これは、ツリーノードが直接計算するのに使えないので、便利です。
dataset temp=tr.a;
// 非接続データセットで計算を実行
col(2)=temp*2;
```

ベクターツリーノードの要素には、次のような記述で直接アクセスできます:

```
// 上記の例題のつづく操作
col(3)[1] = tr.a[3];
```

これは、ベクター **tr.a** の 3 番目の要素を現在のワークシートの 1 行 3 列目に代入します。

また、下記のように分析結果をツリー変数に出力することもできます。

```
newbook;
//ワークシートにデータファイルをインポートする
string fn$=system.path.program$ + "\samples\Signal Processing\fftfilter1.dat";
impasc fname:=fn$;
tree mytr;
//FFT を実行し、結果をツリー変数に保存
fft1 ix:=col(2) rd:=mytr;
page.active=1;
col(3) = mytr.fft.real;
col(4) = mytr.fft.imag;
```

ツリーに関する詳細は Origin プロジェクトの章、**メタデータにアクセスする**セクションをご覧ください。

グラフィックオブジェクト

新しい LabTalk の変数型を使うと、ブック/レイヤ内のグラフィックオブジェクトを GObject で制御することができます。

一般的なシンタックスは:

GObject *name* = [*GraphPageName*]*LayerIndex*!*ObjectName*;

GObject *name* = [*GraphPageName*]*LayerName*!*ObjectName*;

GObject *name* = *LayerName*!*ObjectName*; // アクティブグラフ

GObject *name* = *LayerIndex*!*ObjectName*; // アクティブグラフ

GObject *name* = *ObjectName*; // アクティブレイヤ

すでに存在するオブジェクトと、まだ作成されていないオブジェクト両方に GObject 変数を宣言することができます。

例えば:

```
GObject myLine = line1;
draw -n myLine -l {1,2,3,4};
win -t plot;
myLine.X+=2;
/* myLine が、アクティブでない別のグラフにあったとしても、それを制御することができます。*/
```

グラフィックオブジェクトとそのプロパティおよびメソッドについての説明については、グラフィックオブジェクトをご覧下さい。

変数

変数は、単に特定のデータ型のインスタンスです。各変数には、名前または識別子があり、識別子は変数にデータを割り当てたり、変数からデータにアクセスするのに使われます。この代入演算子は等号 (=)で、同時に変数を作成(存在しない場合)して、それに値を割り当てるのに使われます。

変数の命名規則

変数、データセット、コマンド、マクロ名は、識別子として一般に参照されます。LabTalk で識別子を割り当てるとき

- 文字と数値のどの組合せでも使えますが、以下の注意が必要です。
    - 識別子は 25 文字以上にすることはできません。
    - 最初の 1 文字を数字にすることはできません。
    - アンダスコア "_" はデータセット名で特別な意味を持ちますので、使用を避けて下さい。
- **Exist** (関数) を使って、識別子が、ウィンドウ、マクロ、ツール、データセット、変数に名前を付けるのに使われているかどうかをチェックすることができます。
- いくつかの一般的な識別子がシステム利用として Origin に予約されています。これについては、システム変数をご覧下さい。

変数名競合の取り扱い

@ppv システム変数は、プロジェクト変数、セッション変数、ローカル変数の名前の競合を Origin がどのように扱うかを制御します。他のすべてのシステム変数と同様に**@ppv** は、スクリプトからいつでも変更でき、直ちに反映されます。

変数	説明
@ppv=0	これはデフォルトのオプションで、セッション変数とローカル変数の両方ともが既存のプロジェクト変数の名前を使うことができます。競合のイベントが発生すると、セッション変数またはローカル変数が使われます。
@ppv=1	このオプションは、既存のプロジェクト変数同じ名前を持つセッション変数を宣言することを不正とします。新しいプロジェクトをロードするとき、競合した名前を持つセッション変数は、プロジェクトが閉じられるか、同じ名前を持つプロジェクト変数が削除されるまで、無効になります。
@ppv=2	このオプションは、既存のプロジェクト変数同じ名前を持つローカル変数を宣言することを不正とします。新しいプロジェクトをロードするとき、競合した名前を持つローカル変数は、プロジェクトが閉じられるか、同じ名前を持つプロジェクト変数が削除されるまで、無効になります。
@ppv=3	これは@ppv=1 と @ppv=2 の組合せです。この場合、すべてのセッション変数とローカル変数は、プロジェクト変数の名前を使うことができません。新しいプロジェクトがロードされる場合、同じ名前の既存のセッションまたはローカル変数は無効になります。

変数の一覧表示と削除

LabTalk コマンドの **list** と **del** を使って、変数を一覧表示したり、変数を削除します。

```
/* LabTalk コマンドの"list"には変数を表示するさまざまなオプションがあります。
リストはデフォルトでスクリプトに出力されます。*/

list a;         // 全てのセッション変数を表示
list v;         // 全てのプロジェクト変数とセッション変数を表示
list vs;        // 全てのプロジェクト文字列変数とセッション文字列変数を表示
list vt;        // 全てのプロジェクトツリー変数とセッションツリー変数を表示

// LabTalk コマンド "del" を使用して変数を削除

del -al <variableName>;   //   特定のローカルまたはセッション変数を削除
del -al *;                //   すべてのローカルまたはセッション変数を削除

// LabTalk 変数のビューアもあります。
// "ed" コマンドもビューアを開くことができます。
list;                        // LabTalk 変数ビューアを開く
```

一覧表示および削除のすべてのオプションについては(言語リファレンス:コマンドリファレンス)の **List (コマンド)**および **Del (コマンド)** をご覧ください。

オプションが指定されていない場合、List または Edit コマンドを実行すると LabTalk の変数と関数ダイアログが開き、すべての変数および関数を一覧表示します

変数のスコープ

変数が宣言される方法により、そのスコープが決まります。宣言無しで作成された変数 (**double**, **string**, **dataset** のみ)は、プロジェクト変数になり、Origin のプロジェクトファイルとともに保存されます。宣言された変数は、ローカルまたはセッション変数になります。LabTalk のスコープは、3 つの可視のレベルで構成されます。

- プロジェクト変数

- セッション変数
- ローカル変数

プロジェクト変数

- プロジェクト変数(グローバル変数)は、Origin プロジェクトファイル(*.OPJ)と一緒に保存されます。プロジェクト変数は、プロジェクトスコープを持ちます。
- プロジェクト変数は、**double**, **string**, **dataset** の変数型に対して宣言無しで自動的に作成されます。

```
// double 型のプロジェクト(プロジェクトスコープ)変数を定義:
myvar = 3.5;
// 非接続データセット(グローバルスコープ)を定義:
temp = {1,2,3,4,5};
// string 型のプロジェクト(プロジェクトスコープ)変数を定義:
str$ = "Hello";
str$ = "Hello";
```

- 他のすべての変数型は宣言する必要があり、そのデフォルトのスコープをセッションまたはローカルにします。@global システム変数を使ってローカル変数を強制的にセッション変数として利用可能にできます(下記参照)。

セッション変数

- セッション変数 は、Origin プロジェクトと一緒に保存されず、プロジェクトを切り替えても、現在の Origin セッションで利用できます。つまり、一度セッション変数が定義されると、その変数は、Origin アプリケーションが終了するか、変数を削除するまで、存在します。セッション変数は、変数宣言で定義されます。

```
// double 型の変数を定義
double var1 = 4.5;
// 非接続データセットを定義
dataset mytemp = {1,2,3,4,5};
```

プロジェクト変数とセッション変数を同じ名前にすることができます。このような場合、セッション変数が優先されます。

```
aa = 10;
type "First, aa is a project variable equal to $(aa)";
double aa = 20;
type "Then aa is a session variable equal to $(aa)";
del -al aa;
type "Now aa is project variable equal to $(aa)";
```

そして、出力は

```
First, aa is a project variable equal to 10
Then aa is a session variable equal to 20
Now aa is project variable equal to 10
```

ローカル変数

ローカル変数は、特定のスクリプトの現在のスコープ内でのみ存在します。
スクリプトレベルのスコープが次のようなスクリプトに対して存在します。

- 中括弧{}で囲まれている
- 別の *.OGS ファイルまたは OGS ファイル内の個々のセクション
- 列/行列の値ダイアログの内部
- カスタムボタン(ボタンスクリプト)

ローカル変数が宣言されると、セッション変数と同じ方法で値を割り当てます。

```
loop(i,1,10){
     double a = 3.5;
     const e = 2.718;
     // 他のスクリプト行
}
//"a" および "e" は、{}で囲まれたコードの内部にだけ存在する
```

セッション変数またはプロジェクト変数と同じ名前を持つローカル変数を持つことが可能です。この場合、スクリプトのスコープ内で、同じ名前のローカル変数、セッション変数、プロジェクト変数が存在する場合、ローカル変数が優先されます。例えば、以下のスクリプトを実行する場合（以下のようなスクリプトの実行について詳細は、LabTalk スクリプトをファイルから実行を参照してください）:

```
[Main]
double aa = 10;
type "In the main section, aa equals $(aa)";
run.section(, section1);
run.section(, section2);

[section1]
double aa = 20;
type "In section1, aa equals $(aa)";

[section2]
type "In Section 2, aa equals $(aa)";
```

Origin は次のように出力します。

```
In the main section, aa equals 10
In section1, aa equals 20
In Section 2, aa equals 10
```

ローカル変数と関数をセッション内で使用

変数や関数を *.OGS ファイルで定義し、スクリプトウィンドウやテキストラベルなどから使用できます(通常、ローカル変数や関数は OGS 完了まで実行されたときに存在しなくなります)。このようにするには、**@global** システム変数を **1** にします(デフォルト値は **0**)。これにより、OGS ファイルのローカル変数と関数をセッション内で使用できます。

```
[Main]
@global = 1;
// 以下の宣言はセッション内で利用可能になります
range a = 1, b= 2;
if(a[2] > 0)
{
  // ローカルスコープを開始
  range c = 3; // この宣言はまだセッション内で利用可能
}
```

*.OGS を終了すると、**@global** 変数は自動的にそのデフォルト値である **0** に戻ります。

@global を次のように開始位置と終了位置に配置して、コードブロックを制御できます。

```
@global=1;
double alpha=1.2;
double beta=2.3;
Function double myPeak(double x, double x0)
{
   double y = 10*exp(-(x-x0)^2/4);
   return y;
}
@global=0;
double gamma=3.45;
```

上記の場合、変数 alpha, beta およびユーザ定義関数 myPeak は、セッション内で利用可能です。一方、変数 gamma は、@global がデフォルト値である 0 に戻ってから宣言されたため、利用できません。

概要表:変数のスコープ

	データ型	宣言	定義される場所	サンプル	有効な期間
定数	定数	する (const)	ORGSYS.CNF	const av = 6.022×10^{23}	Origin が起動している間
プロジェクト変数	double, string, dataset	しない	• スクリプトウィンドウ • コマンドウィンドウ • OGS ファイル • 様々な GUI ダイアログ	av = 6.022×10^{23}	OPJ が開いている間有効。OPJ とともに保存可能。

セッション変数	全てのタイプ	する	• スクリプトウィンドウ • コマンドウィンドウ	double av = 6.022×10^{23}	セッションの間有効。OPJ とともに保存不可。
ローカル変数	全てのタイプ	する	• OGS ファイル • 様々な GUI ダイアログ	double av = 6.022×10^{23}	スクリプトが実行されている間
セッション変数としてのローカル変数	全てのタイプ	する	• OGS ファイル • 様々な GUI ダイアログ	**@global=1** のときに, double av = 6.022×10^{23}	セッションの間有効。OPJ とともに保存不可。

5.1.2. プログラミングシンタックス

プログラミングシンタックス

LabTalk スクリプトは、LabTalk のインタプリタにより受け取られ、解釈･実行される、ひとまとまりの LabTalk テキストのことです。LabTalk スクリプトは、1 つ以上のプログラミングステートメントで構成され、それぞれに割り当てられた操作を実行します。

スクリプト中の各文の文末には、他の文と分割するためにセミコロン(;)を付ける必要があります。ただし、スクリプトウィンドウ中で実行される文が単一の場合、その文末にはセミコロンの必要はありません。

スクリプト中の各々の文は「語」から構成されます。語は、ホワイトスペース(スペース、タブ、キャリッジリターン)で区切られているテキストのグループのことです。括弧で囲まれているテキストは、ホワイトスペースの有無に関係なく、1 語として扱われます。例えば、

```
type This is a statement;                    // 1 つの LabTalk のステートメント

ty s1; ty s2; ty s3;                        // 3 つのステートメント
```

括弧は、ホワイトスペースを含む長い語を作成するために使用されます。例えば、以下のスクリプトをみてみましょう。

```
menu 3 (Long Menu Name);
```

ここで、左括弧は単語の開始を意味し、右括弧はその語の終了を意味します。

このセクションで以下の項目を説明します

- ステートメントの種類
- LabTalk 中でのセミコロンの使用

- 一つのステートメントを数行にわたって入力する
- コメント
- ステートメントの評価の順序

ステートメントの種類

LabTalk は次の 5 種類のステートメントに対応しています。

- 代入文
- マクロ文
- コマンド文
- 算術文
- 関数ステートメント

代入文

代入文は一般的に次のような構文をしています。

LHS = expression ;

expression (RHS, 右辺) は計算され、*LHS* (左辺)に代入されます。*LHS* が存在しない場合、可能なら作成され、不可ならエラーが返されます。

代入文により新しいデータオブジェクトが作られた場合、作成されたオブジェクトは、次のいずれかの型となります。

- *LHS* が *stringVar*$ = "Hello."のように**$**で終わる場合、文字列変数
- *expression* がスカラーの数値として評価された場合は、数値変数
- *expression* がデータ領域として評価された場合は、データセット

新しい値が既存のデータオブジェクトに割り当てられた場合は、以下の規則に従います。

- *LHS* がデータセットで、*expression* がスカラー数値の場合、*LHS* 中の全ての要素の値は *expression* になります。
- *LHS* が数値変数の場合、*expression* もスカラー数値に評価されなければなりません。もし、*expression* がデータセットの場合、*LHS* はそのデータセットの第 1 要素の値となります。
- *LHS* と *expression* の両方がデータセットの場合、*LHS* の各要素の値は、*expression* 中の対応する位置の値に等しく設定されます。
- *LHS* が文字列の場合、*expression* は文字列式とみなされます。

- *LHS* が**オブジェクトプロパティ**の表記の場合、終わりに$があるか、ないかに関係なく、この表記は、wks.ncols=3;のようにワークシート列数などのオブジェクトプロパティを**セット**するのに使われます。

代入文の使用例

変数 **B** に値 2 を割り当てます。

```
B = 2;
```

Test を変数 B の三乗とします。

```
Test = B^3;
```

文字列変数**%A** に Austin TX を割り当てます。

```
%A = Austin TX;
```

データセット **Book1_B** の全ての要素に 4 を割り当てます。

```
Book1_B = 4;
```

Book1_B の各要素に、**Book2_B** の各々の要素の値を割り当てます。

```
Book1_B = Book2_B;
```

ワークシートオブジェクトの **rhw** のプロパティを使用して、**Book1** のワークシートの行見出しの幅を 100 に設定します。**doc -uw** コマンドはウィンドウをリフレッシュします。

```
Book1!wks.rhw = 100;  doc -uw;
```

more と **yetmore** 中の対応するインデックスにある値に対して演算が行なわれます。**myData** 中の同じインデックスのセルにその結果を割り当てます。

```
myData = 3 * more + yetmore;
```

Note:代入演算の左辺にある文字列変数が括弧で囲まれている場合、その文字列変数は、代入の前に置換操作を施されます。例えば、

```
%B = DataSet;
(%B) = 2 * %B;
```

この例では、DataSet の値に 2 が掛けられ、DataSet に再度代入されます。文字列変数**%B** は、文字列"DataSet"のままです。

文字列レジスタと同様、代入文も次のように文字列変数で使われます。

```
fname$=fdlg.path$+"test.csv";
```

この場合、*expression* は、文字列そのもの、文字列変数、複数文字列を**+**文字で連結した文字列の文字列式です。

マクロ文

マクロとは、与えられたスクリプトに特定の名前を関連付けて、スクリプトを別名で呼ぶ一つの方法です。マクロ名は、そのスクリプトを呼び出すコマンドのように使用できます。

詳細については、マクロをご覧ください

コマンド文

3 番目の文の種類は、コマンド文です。LabTalk には、プログラムを制御、修正するため、数多くのコマンドが用意されています。

コマンド文は、識別子であるコマンド名で始まります。この名前は、(一意に識別できる限り)最高 2 文字までに省略できます。ほとんどのコマンドは、オプション(スイッチとも言います)を取ることができ、これはコマンドの機能の詳細を制御する 1 文字です。オプションスイッチは、ダッシュ記号"-"に引き続いて、1 文字ないし数文字アルファベットで指定されます。コマンドは引数を持つこともできます。引数は、データオブジェクトか、ひとまとまりのスクリプトです。多くの場合、オプションにも引数を持たせることができます。

コマンド文は、一般的に次の構文で表されます。

```
command [option] [argument(s)];
```

ここで、カギ括弧[]内の項目は、オプションを示し、これは省略可能です。全てのコマンドにオプションと引数の両方が付いているわけではありません。カギ括弧は、実際のコマンドの入力には含めません。

オブジェクトメソッド は別種のコマンド文です。オブジェクトメソッドは、(名前の付き)オブジェクトに対し、直接操作を施します。オブジェクトメソッドは、次の構文で表されます。

```
ObjectName.Method([options]);
```

例えば、

次のスクリプトは、アクティブワークシートに new と呼ばれる列を追加し、ウィンドウをリフレッシュします。

```
wks.addcol(new);
doc -uw;
```

以下に、様々なコマンド文の例を示します。

ゼロを基準に、データセット myData を積分します。

```
integ myData;
```

integ コマンドに、オプション -r と引数 baseline を追加することにより、myData を baseline という基準曲線を基に積分します。

```
integ -r baseline myData;
```

repeat コマンドは次の 2 つの引数を取って実行します。

1. 実行する回数
2. 繰り返す手順を示すスクリプト

このコマンド文は、3 度「Hello World」とダイアログボックスに表示します。

```
repeat 3 {type -b "Hello World"}
```

算術文

算術文は一般的に次の構文を持ちます。

```
dataObject1 operator dataObject2;
```

ここで

- *dataObject1* は、データセットか数値変数
- *dataObject2* は、データセット、変数、定数
- *operator* は、+、-、*、/、^ のいずれか

計算結果は、*dataObject1* に代入されます。*ここで、dataObject1* は関数ではないことに注意してください。例えば、**col(3)+25** は、この型の文の使用法としては間違っています。

算術文の異なる形式を表す例

myData をデータセットとすると、**myData** の各々のセル値を 10 で割ります。

```
myData / 10;
```

myData の各セル値から **otherData**b をの対応する値を引き、結果を **myData** に代入します。2 つのデータセットは Y 同士(あるいは Z 同士)の XY 属性を持っていなければなりません。(下の注意**参照**。)

```
myData - otherData;
```

A が変数の場合、A に 1 を加えます。A がデータセットの場合、A 列の各値に 1 を加えます。

```
A + 1;
```

Note:算術文と代入文では、2 つのデータセットを扱う時、その扱い方が異なります。例えば、算術文 **data1_b+data2_b** と代入文 **data1_b=data1_b+data2_b** の結果は、違った方式で算出されます。後者のケースは、2 つのデータセットのそれぞれの X 値は無視して、各点の合計を計算します。前者の場合、**data1_b + data2_b** は、2 つのデータセットが XY 平面上の曲線であるかのようにそれぞれを加えます。そのため、**data1_b** と **data2_b** の X 値が異なる場合、2 つのデータセットの一方を補間する必要があります。このイベントでは、Origin は最初のデータセット(この場合、**data1_b**)の X 値を補間します。

関数ステートメント

関数ステートメントは、関数名--識別子--に引き続いて括弧(...)で引数を指定します。

次は関数ステートメントの例です。

```
sum(dataset);
```

LabTalk の関数についての詳細は、関数をご覧下さい。

LabTalk でのセミコロンの使用

セミコロンでステートメントを分割する

C 言語の構文と同様、LabTalk でもセミコロンが、ステートメントを分割するために使用されます。通常、各ステートメントはセミコロンで終わります。しかし、セミコロンの使用について次のような規則があります。

- スクリプトウィンドウで単一のステートメントスクリプトを実行する時はセミコロンを使用しないでください。
    - 適切な構文の例は次のとおりです: **`type "hello"`** (ENTER).
    - それが実行されると、インタプリタによって自動的にセミコロンが文末に付けられます。
- { } ブロックで終了するステートメントは、セミコロンを省略できます。
- { } ブロック内の最後のステートメントもセミコロンを省略できます。

次のサンプルで、3 つの type コマンド間の違いに注意してください。

```
if (m>2) {type "hello";} else {type "goodbye"}
type "the end";
```

上記は次のように書くこともできます。

```
if (m>2) {type "hello"} else {type "goodbye"}
type "the end";
```

または

```
if (m>2) {type "hello"} else {type "goodbye"};
type "the end";
```

遅延実行のための文頭のセミコロン

スクリプトの実行を遅らすために、スクリプトの前にセミコロンを ';' 置くことができます。これは、ウィンドウを閉じたり、新しくプロジェクトを作成するといったボタン自体が消去されるボタン内でスクリプトを実行する必要がある場合に、必要となります。例えば、ボタン内に以下のスクリプトを置くと、おそらくクラッシュするでしょう。

```
// このウィンドウを閉じるボタン
type "closing this window";
win -cn %H;
```

これを解決するため、スクリプトを次のように記述します。

```
//このウィンドウを閉じるボタン
type "closing this window";
;win -cn %H;
```

実行を遅くするため、文頭のセミコロン ';' の後にすべてのスクリプトを記述します。ステートメントの特定グループを遅延させたい場合、セミコロン';'に続けて、{script} の内側にスクリプトを置くことができます。例えば

```
//このウィンドウを閉じるボタン
type "closing this window";
;{type "from delayed execution";win -cn %H;}
type "actual window closing code will be executed after this";
```

関連項目: LabTalk システム変数 @LT

一つのステートメントを数行にわたって入力する

スクリプトを読み易くするために、一つの文を、いくつかの行に分けて入力したい場合があります。この方法の一つは、中括弧{}を使用することです。スクリプトファイルに左中括弧**{**が見つかると、Origin はその先の右中括弧**}**を検索し、その間のブロック全体が一つの文として実行されます。例えば、次のマクロ文について考えてみましょう。

```
def openDialog {layer -s 1;  axis x;};
```

次のように記述することもできます。

```
def openDialog {
                  layer -s 1;
                  axis x;
                };
```

この 2 つ目の例文は数行にわたっていますが、両スクリプト共、同じ1つの文として実行されます。

Note: { } の間に含まれるスクリプトの長さには制限があります。LabTalk スクリプト内の { } に含まれるスクリプトは内部的に変換され、変換後のスクリプトは合計 1140 バイト以下(置換後)でなくてはなりません。この制限を回避し、長く扱いにくい LabTalk コードのブロックを扱う代わりに、マクロ文や、run.section()、run.file()などのオブジェクトメソッドを使用することができます。詳細は 引数の受け渡し を参照してください。

コメント

LabTalk スクリプトでは、2 つのコメントフォーマットを利用できます。

あるスクリプト中のテキストをコメントとして扱うには「//」の記号を使用し、//から行末までのテキスト全体が無視されます。例えば、

```
type "Hello World"; //ここにコメントを記述
```

「/*」と「*/」の文字を組合せを使って、コメントの開始と終了を表し、実行したくないコードブロックまたはテキストブロックを囲みます。例えば、

```
type Hello /* ここにコメントのテキスト
              あるいは実行しないコード
              などを記述 */*/
World;
```

Note:スクリプトのデバッグ行を指定するには「#!」を使用します。`system.debug=1` の場合にのみ、この記号以降のスクリプト行が実行されます。

ステートメントの評価の順序

スクリプトが実行される場合、スクリプトは解釈･実行されるため LabTalk のインタープリタに送られ、以下のような手続きで評価されます。

スクリプトは、その構成要素のステートメントに分割されます。

各ステートメントは,代入文、マクロ文、コマンド文、算術文、関数文の順序で文型が確定されます。インタプリタは最初に明示されている(括弧や引用符中に隠されていない)代入演算子を検索します。これが見つからない場合、次に最初の語がマクロ名かどうかを調べます。その次にそれがコマンド名かどうかを調べます。次にインタプリタは算術演算子を検索し算術文かどうかをチェックし、最後にその文が関数文かどうかを調べます。

ステートメント解釈の優先順序が、スクリプトの実行に大きく影響することもあります。例えば、次の代入文について考えてみましょう。

```
type = 1;
```

この代入文は、変数 type に 1 を割り当てます。ステートメント解釈の順序に従い、代入はコマンドの前に行われるので、type が LabTalk のコマンドであるにも関わらず、代入操作が行なわれます。しかし、次のステートメントでは、コマンド文が算術表現より先に評価されるので、コマンド type が実行されます。

```
type + 1;
```

つまり、この 2 番目の例では、文字列「+1」が出力されます。

次の評価ルールに従い、受け取られた順に評価されていきます。

- 代入文: 代入演算子の左辺の文字列変数は、括弧で囲まれていない限り展開されません。それ以外の全ての文字列変数は展開され、置換記法(**%()**と**$()**)は置換処理されます。
- マクロ文: マクロの引数は置換プロセスを経て、渡されます。
- コマンド文: コマンドがリテラルである場合、それは置換プロセッサには送られません。そうでない場合は、展開･置換処理が行なわれます。
- 算術文: 数式は、全ていったん置換プロセスを経て展開されます。

5.1.3. 演算子

内容

イントロダクション

LabTalk では、代入演算子、算術演算子、論理演算子、関係演算子、条件演算子の 5 種があります。

算術演算子	+ - * / ^ & |
文字列の連結	+
代入演算子	= += -= *= /= ^=
論理・関係演算子	> >= < <= == != && ||
条件演算子	?:

これらの演算子は、スカラーで計算されますが、多くの場合、ベクトル(データセット)でも実行できます。また、Origin にはデータセットに対して実行可能な数値関数、数学関数、三角関数、統計関数が組み込まれています。

LabTalk では、数式の評価にあたり、以下の優先順位の規則を適用します。

1. 括弧のない代入演算子が評価されます。
2. 括弧内(...)の演算が括弧外の演算より優先されて評価されます。
3. 乗算(*) と除算(/) が、加算(+) と減算(-) より優先されて実行されます。
4. 関係演算子（>,>=,<,<=）が評価されてから、等号･不等演算子（==と!=）が評価されます。
5. 論理演算子 || は&&より先に評価されます。
6. 最後に、条件演算子（?:）が評価されます。

算術演算子

Origin では、以下の算術演算子を利用できます。

演算子	説明
+	加算
-	減算
*	乗算
/	除算

^	ベキ乗 (X^Y は X を Y 乗します。) (下記 Note を参照)
&	ビットごとの論理積(And)演算子。数値を構成するビットに対して実行されます。
|	ビットごとの論理和(Or)演算子。数値を構成するビットに対して実行されます。

Note:0 の累乗 n (0^n)の場合:n > 0 の場合、0 が返されます。If n < 0 の場合、欠損値が返されます。n = 0 の場合、1 が返されるか(@ZZ = 1 の場合) または 欠損値が返されます(@ZZ = 0 の場合)

これらの演算子はスカラーおよびベクトル(データセット)で演算することができます。スカラーやベクトル計算に関する詳細は、計算の実行 を参照してください。

次のサンプルはべき乗演算子の使用法を示しています。コマンドウィンドウに次のスクリプトを入力します。

```
1.3 ^ 4.7 =
```

ENTER キーを押すと、3.43189 がコマンドウィンドウに表示されます。次のサンプルは、ビットごとの**論理積 AND** 演算子の使用法を示しています。コマンドウィンドウに次のスクリプトを入力します。

```
if (27&41 == 9)
{type "Yes!"}
```

ENTER キーを押すと、**Yes!** がコマンドウィンドウに表示されます。

Note:27&41 == 9 because

```
27 = 0000000000011011
41 = 0000000000101001
```

ビットごとに&(and)を施すと

```
0000000000001001 （となり、 9 と等しくなります）
```

Note:乗算では掛算記号の*を明示しなくてはいけません。例えば、変数 X に定数 2 を掛ける場合、2X ではなく、2*X と記述します。

定数の定義

ORGSYS.CNF ファイル内に定数を定義することができます。

const pi = 3.141592653589793

対数変換についての注意

- データセットを対数スケールに変換するには、次の構文を使用します。

```
col(c) = log(col(c));
```

- データセットを線形スケールに変換し直すには、次の構文を使用します。

```
col(c) = 10^(col(c));
```

文字列の連結

まれに文字列変数または文字列レジスタ型のいずれかの複数の文字列を連結する必要がある場合があります。このセクションのコード部分すべてで、"Hello World."という文字列を返します。

文字列連結演算子は、プラス記号(+)で、2 つの文字列を連結するために使用します。

```
aa$ ="Hello";
bb$="World";
cc$=aa$+" "+bb$;
cc$=;
```

2 つの文字列レジスタを連結するには、単にそれらを並べるだけです。

```
%J="Hello";
%k="World";
%L=%J %k;
%L=;
```

文字列変数と文字列レジスタの両方を操作する必要がある場合、%( ) 置換表記を利用して以下のようにします。

```
aa$ ="Hello";
%K="World";
dd$=%(aa$) %K;
dd$=;

dd$=%K;
dd$=aa$+" "+dd$;
dd$=;

%M=%(aa$) %K;
%M=;
```

代入演算子

Origin では、以下の算術演算子を利用できます。

演算子	説明
=	変数(やデータセット)に引数を割り当てる。
+=	変数(やデータセット)の内容に引数を加えて、割り当てる。
-=	変数(やデータセット)の内容から引数を引いて、割り当てる。
*=	変数(やデータセット)の内容に引数を掛けて、割り当てる。
/=	変数(やデータセット)の内容を引数で割って、割り当てる。
^=	変数(やデータセット)の内容を引数でベキ乗して、割り当てる。

これらの演算子はスカラーおよびベクトル(データセット)で演算することができます。スカラーやベクトル計算に関する詳細は、計算の実行 を参照してください。

次のサンプルは-= 演算子の使用法を示しています。

この例では、変数 A の値から 5 を引き、結果を A に割り当てています。

```
A -= 5;
```

次のサンプルでは、Data1_B 中の各値を Book1_A 中の対応する値で割り、結果を Book1_B に割り当てるものです。

```
Book1_B /= Book1_A;
```

これらの代入演算子に加えて、LabTalk は、インクリメントおよびデクリメント演算子にも対応しています。(スカラーのみ、ベクトルは不可)

演算子	説明
++	変数の内容に 1 を加えて、割り当てる。
--	変数の内容から 1 を引いて、割り当てる。

次の **for** ループの式は、インクリメント演算子++の一般的な使用法を示しています。スクリプトは現在のワークシートの 2 番目の列のデータをコマンドウィンドウに出力します。

```
for (ii = 1; ii <= wks.maxrows; ii++)
   {type ($(col(2)[ii])); }
```

比較･論理演算子

Origin では、以下の比較･論理演算子が利用できます。

演算子	説明
>	より大きい
>=	以上
<	より小さい
<=	以下
==	等しい
!=	等しくない
&&	かつ
||	または

論理･関係演算子を使った式は、真(非ゼロの値)、偽(ゼロ)のいずれかに評価されます論理演算子は、通常、条件構造とループ構造の中で使用します。

数値の比較

もっと一般的な比較は 2 つの数値を比較することです。一般的には、少なくとも一方が変数です。例えば

```
if aa<3 type "aa<3";
```

または、両方の項目を変数にして比較することもできます。

```
if aa<=bb type "aa<=bb";
```

括弧を使って、同じ論理ステートメント内で複数比較することもできます。

```
if (aa<3 && aa<bb) type "aa is lower";
```

文字列の比較

2 つの文字列の比較には、==と!=の演算子が使えます。演算子の前か後に置かれて比較される文字列定数は(数値の比較の場合とは違い)、引用符(")で囲まれなければなりません。次のスクリプトは、%A が空の文字列であるかどうか調べます。

```
if (%A == ""){type "empty"};
```

次のサンプルは等号(==)演算子の使用法を示しています。

```
x = 1;      // 変数 x を 1 にセット
%a = x;     // 文字列 a に"x"をセット
if (%a == 1);
      type "yes";
else
      type "no";
```

Origin は%a の値(x の値)を探し、これが 1 なので、結果は、yes となります。次のスクリプトを確認します。

```
x = 1;      // variable x is set to 1
%a = x;     // string a is set to "x"
if ("%a" == 1)
      type "yes";
else
      type "no";
```

このスクリプトでは、%a が二重引用符で囲まれているので、Origin はこれを文字列とみなし、その結果 %a は1ではなく x となり、結果は no となります。

条件演算子 (?:)

三項演算子 または 条件演算子 (?:) は、次のような構文で使用されます。

Expression1 ?Expression2 :Expression3

最初に Expression1 が評価されます。Expression1 が真(ゼロでない)の場合、Expression2 が評価されます。Expression2 の値が条件式の値になります。Expression1 が偽(ゼロ)であると、Expression3 が評価され、Expression3 がこの条件式の値となります。Note：Expressions1 と 2 の表現自体を条件演算子にしてネストすることも可能です。次のサンプルでは、大きい方の値（m または n）を変数に割り当てます。

```
m = 2;
n = 3;
variable = (m>n?m:n);
variable =
```

LabTalk は次のように返します。**variable = 3**

次の例では、スクリプトが A 列にある全ての 5.5 から 5.9 の値を 5.6 に置き換えます。

```
col(A) = col(A)>5.5&&col(A)<5.9?5.6:col(A);
```

Note:しきい値置換関数 treplace(dataset, value1, value2 [, condition]) でもデータセットの値を評価することができ、条件によって、他の値にそれらを置換することができます。treplace(dataset, value1, value2 [, condition]) 関数では、データセットの各値は条件に応じて value1 と比較されます。比較が真の時、その値は condition の値に応じて Value2 または -Value2 に置換されます。比較が偽の時、その値は condition の値に応じてそのままの値を持つかまたは欠損値に置換されます。treplace() 関数は条件演算子よりも高速に処理されます。

計算の実行

LabTalk を使って、次の両方の演算を実行できます。

- スカラー計算 (一変数の数学演算) および
- ベクトル計算 (データセット全体の数学演算)

スカラー演算

LabTalk を使って、演算を実行し、結果を数値変数に格納することができます例えば、以下のスクリプトを見てみましょう。

```
inputVal = 21;
myResult = 4 * 32 * inputVal;
```

この例の 2 行目で計算を実行し、変数 myResult を作成します。計算結果は、myResult に格納されます。

変数が被演算子(オペランド)であり、結果が格納される場合、簡易表記を利用できます。例えば、次のスクリプトを見てみましょう。

```
B = B * 3;
```

次のように記述することができます。

```
B *= 3;
```

このサンプルでは、変数 B に割り当てられた結果に対して乗算がなされています。同様に、+=、-=、/=、^=も利用可能です。簡易表記を使用するとスクリプトを高速に実行できます。

ベクトル計算

変数への演算の実行および結果の実行(スカラー演算)に加えて、LabTalk を使って、データセットへの演算を行うことができます。

ベクトル計算は、(1) 行ごとに行う、(2) 線形補間を使う、のいずれかの方法で実行されます。

行ごとの計算

ベクトル演算は一般的な記法を用いている場合には常に行ごとに実行されます。

datasetB = scalarOrConstant <operator> datasetA;

datasetC = datasetA <operator> datasetB;

これはデータセットの要素数が異なる場合でも当てはまります。ワークシートに A, B, C の 3 つの空の列があるものとします。次のスクリプトを実行します。

```
col(a) = {1, 2, 3};
col(b) = {4, 5};
```

```
col(c) = col(a) + col(b);
```

列 C の結果は、{5, 7, --}となります。つまり、Origin はデータセットに値が含まれない場合、行に対して欠損値を出力します。
Vector calculations can also involve a scalar.上記の例で、次のように入力します。

```
col(c) = 2 * col(a);
```

列 A を 2 倍して、結果を列 C の対応する行に出力します。
または、次のスクリプトを実行します(*newData* は以前には作成されていないものとします):

```
newData = 3 * Book1_A;
```

newData という名前の一時データセットが作成され、ベクトル演算の結果が割り当てられます。

補間を使った計算

Origin は、範囲表記と interp1 および interp1xy のような X ファンクションを使った補間をサポートします。詳細は、補間を参照して下さい。

5.1.4. 条件およびループ構造

LabTalk 言語の構造は C 言語のそれと似ています。LabTalk は、以下をサポートしています。

- 処理を繰り返し実行するループ構造
- 条件の真、偽に応じて処理の実行を制御する分岐構造

内容

ループ構造

すべての LabTalk のループは、引数としてスクリプトを取ります。これらのスクリプトは、指定した条件下で繰り返し実行されます。LabTalk には次の 4 つのループコマンドがあります。

コマンド	構文
repeat	repeat value {*script*};
loop	loop (variable, startVal, endVal) {*script*};
doc -e	doc -e object {*script*};
for	for (expression1; expression2; expression3) {*script*};

LabTalk の for ループは、他のプログラミング言語の for ループと同じです。repeat, loop, doc -e ループは、他の言語では馴染みがないものですが、簡単に使うことができます。

Repeat

repeat ループは、一連の操作が変更無く繰り返されるときに使われます。

シンタックス:**repeat *value* {*script*};**

スクリプト *script* を、*value* で指定した回数分、またはエラーが起こるまで、または *break* コマンドが実行されるまで実行します。

例えば、次のスクリプトは、文字列を 3 回出力します。

```
repeat 3 { type "line of output"; };
```

Loop

loop ループは、各ループの処理が問題なく実行されると 1 つの変数が増加するようなときに使われます。

シンタックス:**loop (*variable*, *startVal*, *endVal*) {*script*};**

簡単なカウンタ変数を持つループ構造です。*starVal* の値で変数 *variable* を初期化し、スクリプトを実行します。*script* を実行します。*variable* を増加し、それが *endVal* より大きいかどうかをチェックします。大きくない場合、*script* を実行し、ループを継続します。

例えば、次のスクリプトは 1 から 4 までの数を出力します。

```
loop (ii, 1, 4)  {type "$(ii)";};
```

Note:**loop** コマンドは、**for** コマンドよりもスクリプトブロックの実行が高速となります。ループを停止ための条件を判定するため LabTalk 式の解析を行う必要がないため、高速となります。

Doc –e

doc -e ループは、スクリプトを実行してグラフウィンドウのような特定のオブジェクトに対して処理を行うときに使われます。**doc -e** ループは、指定したオブジェクトの各インスタンスに対してスクリプトを実行するように Origin に通知します。

シンタックス:**doc -e *object* {*script*};**

さまざまなオブジェクトタイプがドキュメントコマンドに一覧表示されています。

例えば、次のスクリプトはプロジェクト内のすべてのグラフウィンドウのウィンドウタイトルを出力します。

```
doc -e P {%H=}
```

For

for ループは、上記以外のすべての状況で使われます。

シンタックス：**for (*expression1*; *expression2*; *expression3*) {*script*};**

for ステートメントの中で、*expression1* が評価されます。これによって、ループの初期値が指定されます。次に *expression2* が評価され、その結果が真(0 でない)である場合、*script* が実行されます。そして、カウンターを増加させるため **expression3** が実行されます。処理が 2 番目のステップから繰り返し行われます。*expression2* の結果が偽(0)になった時点でこのループは終了します。3 つの評価式はいずれもカンマで区切られている複数の文で構成することができます。

例えば、次のスクリプトは 1 から 4 を出力します。

```
for(ii=1; ii<=4; ii++)
{
   type "$(ii)";
}
```

Note:**loop** コマンドは、スクリプトブロックの実行が高速となります。

分岐構造

条件の真、偽に応じて処理の実行を制御する分岐構造 LabTalk には、if、if-else、switch の 3 つの判断分岐構造があります。

- **if** コマンドは、指定した条件が成り立った場合、つまり、真(非ゼロ値)の場合に、指定したスクリプトを実行する場合に使用します。
- if-else コマンドは、指定した条件が真(非ゼロ値)の場合、あるスクリプトを実行し、偽(ゼロ)の場合、別のスクリプトを実行する場合に使用します。
- **switch** コマンドは、2 つ又はそれ以上の条件があって、条件を満たした指定スクリプトを実行をする場合に使用します。

If, If-Else

シンタックス：

1. **if (*testCondition*) *sentence1*; [else *sentence2*;]**
2. **if (*testCondition*) {*script1*} [else {*script2*}]**

TestCondition が真である場合、*script1* が実行されます。表現が条件演算子を含まない場合、その結果がゼロでなければ真と評価されます。

オプション **else** が付加されていて、*testCondition* が偽(ゼロ)である場合、*script2* が実行されます。このとき、else の後にスペースを付けなくてはいけません。文字列は引用符で囲まれていなくてはいけません。文字列の比較には大文字と小文字の区別はされません。

一つの文からなるスクリプトの引数の終わりには、セミコロンを付けます。複数の文からなるスクリプトの引数は、大括弧{ }で囲みます。この場合、各々のスクリプトはセミコロンで区切ります。スクリプトを囲む大括弧の後にセミコロンを付ける必要はありません。

例えば、下記のスクリプトは、「Yes!」を表示するメッセージボックスを開きます。

```
%M = test;
if (%M == "TEST")  type -b "Yes!";
else type -b "No!";
```

下記のスクリプトは、列 A 内の-1.95 以上である最初のポイントを見つけます。

```
newbook;
col(1)=data(-2,2,0.01);
val = -1.95;
get col(A) -e numpoints;
for(ii = 1 ; ii <= numpoints ; ii++)
{
   // 真のときにループを終了する
   if (Col(A)[ii] > val)  break;
}
if(ii > numpoints - 1)
   ty -b No number exceeds $(val);
else
   type -b The index number of first value > $(val) is $(ii)
The value is $(col(a)[ii]);
```

例えば、1 つの **if** ステートメントで複数の条件を確かめることができます。

```
if(a>1 && a<3) b+=1;   // 真の場合、b の値を 1 つ増加させる
```

&& (論理和) 演算子は、LabTalk でサポートされているいくつかの論理演算子の 1 つです。

Switch

switch コマンドは、2 つ又はそれ以上の条件があって、条件を満たした指定スクリプトを実行をする場合に使用します。例えば、次のスクリプトは b を返します。

```
ii=2;
switch (ii)
{
     case 1:
         type "a";
         break;
     case 2:
         type "b";
         break;
     case 3:
         type "c";
         break;
     default:
         type "none";
         break;
}
```

Break とプログレスバー

LabTalk には、break コマンドがあります。これを実行すると、ループから抜け、任意でスクリプトを終了します。これは、ループ内の分岐構造でよく利用されます。ループのテスト条件を無効にする条件から保護するために使われます。break コマンドは、ループの処理状況を示すダイアログボックス(プログレスバー)を表示するのに使うことができます。

Exit

exit コマンドは、フラグを強制的に終了するようセットされていなければ、Origin を終了する際のメッセージボックスを表示します。

Continue

continue コマンドは、ループ内で使うことができます。これを実行すると、ループスクリプト内の残りは無視され、インタプリタは次のループを実行します。これは、ループ内の分岐構造と一緒に使われ、ループスクリプトでの処理から不正な値を除外できます。

例えば、下記の for ループにおいて、ii がゼロより小さい場合に、continue 文は type 文を飛ばします。

```
for (ii = -10; ii <= 10; ii += 2)
{
   if (ii < 0)
     continue;

   type "$(sqrt(ii))";
}
```

スクリプトファイル内のセクション

ラベル制御ダイアログにスクリプトを入力することに加え、スクリプトを Origin スクリプト(OGS)ファイルに保存することもできます。Origin スクリプトファイルは、ASCII テキストファイルで、1 行以上の LabTalk ステートメントで構成されています。複数ステートメントを複数セクションに分けることができます。セクション は、次のようにセクション名を角括弧で囲んで宣言します。

[*SectionName*]

別のセクションの宣言が現れるまで、宣言したセクション内にあるスクリプトがそのセクションに属します。セクションを持つスクリプトのフレームワークは、次のようなものです。

```
...
Scripts;
...
[Section 1]
...
Scripts;
...
[Section 2]
...
Scripts;
...
```

各スクリプトは、新しいセクションが現れるか、return ステートメントが実行されるか、エラーが発生するまで順番に実行されます。セクション内のスクリプトを実行するには、次のようなコマンドを使います。

run.section(*FileName*,*SectionName*);

command.FileName が含まれていないとき、現在実行されているスクリプトファイル内のセクションであると見なされます。

run.section(,Init);

次のスクリプトは、OGS ファイル内のセクションを呼び出す方法を示しています。

```
type "Hello, we will run section 2";
run.section(, section2);

[section1]
type "This is section 1, End the script.";

[section2]
type "This is section 2, run section 1.";
run.section(, section1);
```

スクリプトを実行するには、Origin のユーザフォルダに、test.ogs のように保存し、コマンドウィンドウに次のように入力します。

```
run.section(test);
```

セクション内のコードが意図せずセクションを終了させるようなエラーを引き起こす場合、テスト用の変数を使うことができます。

```
[Test]
SectionPassed = 0;
// 実行時にエラーとなるコード
...
SectionPassed = 1;
```

コードがエラーの場合、SectionPassed には 0 の値がセットされます。コードが問題ない場合、SectionPassed には 1 の値がセットされます。

5.1.5. マクロ

マクロの定義

コマンドシンタックス

```
define macroName {script}
```

このコマンドは、*macroName* と呼ばれるマクロを定義し、それを指定した*スクリプト*に関連づけます。この操作により、macroName はコマンドとして利用でき、スクリプトを呼び起こすこともできます。

下記のスクリプトは、指定した文字列を 3 回出力するマクロ hello を定義しています。

```
def hello
{
      loop (ii, 1, 3)
            { type "$(ii).Hello World"; }
};
```

マクロ hello を定義した後、スクリプトウィンドウに文字列 *hello* を入力すると、次の結果が出力されます。

```
1.Hello World

2.Hello World

3.Hello World
```

マクロを定義したら、次のように入力することで、マクロとして定義されているスクリプトを表示することができます。

```
define macroName;
```

マクロに引数を渡す

マクロには最高 5 つまで引数を含むことができます。各引数の値にアクセスするためのマクロ内で、%1-%5 シンタックスを使用します。マクロの引数には、数値、文字列、変数、データセット、関数、及びスクリプトが使用可能です。マクロに引数を渡すことは、スクリプトに引数を渡すことに似ています。

マクロに引数が渡される場合、オブジェクトのプロパティ macro.nArg を使用して、引数の数を調べることができます。

下記のスクリプトは、数値の引数を 1 つ持つ *myDouble* というマクロを定義します。このマクロにより、入力した引数の 2 倍の値が出力されます。

```
def myDouble { type "$(%1 * 2)"; };
```

上記のマクロを定義した後、スクリプトウィンドウに下記のように入力します。

```
myDouble 5
```

Origin はスクリプトウィンドウに下記の結果を出力します。

```
10
```

このマクロを 2 つの引数を取るように修正します。

```
def myDouble { type "$(%1 * %2)"; };
```

マクロの定義後、スクリプトウィンドウに下記のように入力します。

```
myDouble 5 4
```

Origin は、次のように出力します。

```
20
```

マクロのプロパティ

マクロオブジェクトは、マクロに渡された引数の数を保持するプロパティをもちます。

プロパティ	アクセス	説明
Macro.nArg	読み取り専用, 数値	マクロに渡された引数の値を持つプロパティ

例えば、

次のスクリプトは、*TypeArgs* と呼ばれるマクロを定義します。マクロ TypeArgs に 3 つの引数が渡されると、このマクロはそれらの引数をスクリプトウィンドウに表示します。

```
Def TypeArgs
{
	if (macro.narg != 3)
	{
		type "Error!You must pass 3 arguments!";
	}
	else
	{
		type "The first argument passed was %1.";
		type "The second argument passed was %2.";
		type "The third argument passed was %3.";
	}
};
```

上述した例のようにマクロ *TypeArgs* を定義して、下記をスクリプトウィンドウに入力すると、

```
TypeArgs One;
```

下記の結果がスクリプトウィンドウに表示されます。

```
Error!引数を 3 つ渡す必要があります。
```

上述した例のようにマクロ *TypeArgs* を定義して、下記をスクリプトウィンドウに入力すると、

```
TypeArgs  One (This is argument Two) Three;
```

下記の結果がスクリプトウィンドウに表示されます。

```
The first argument passed was One.
The second argument passed was This is argument Two.
The third argument passed was Three.
```

5.1.6. 関数

関数は、ほとんどのプログラミング言語の中核的なものです。以下は、LabTalk での関数のシンタックスと用法についての説明です。

内容

組み込み関数

LabTalk は、組み込み関数を使って、多くの操作をサポートしています。関数の一覧およびそれぞれの説明は、関数リファレンスにあります。関数は、次のようなシンタックスで呼ばれます。

***outputVariable* = *FunctionName*(Arg1, Arg2, ..., Arg N);**

以下は、組み込み関数の使用例です。

Count (関数) は、ベクターデータ内の要素の数を整数で返します。

```
// アクティブなワークシートの列 A にある要素の数を返す
int cc = count(col(A));
```

Ave (関数) は、データセットのグループの平均を計算し、range 変数で結果を返します。

```
range ra = [Book1]Sheet1!Col(A);
range rb = [Book1]Sheet1!Col(B);
// グループの平均値を返す
rb = ave(ra, 5);     // 5 = グループサイズ
```

Sin (関数) は、入力角度の sin を **double** 型で返します。(入力角度の単位は、**system.math.angularunits** の値で決まります)

```
system.math.angularunits=1;   // 1 = 度数で入力
double dd = sin(45);          // ANS:DD = 0.7071
```

ユーザ定義関数

複数の引数を持つユーザ定義関数が、Origin8.1 の LabTalk からサポートされています。ユーザ定義関数のシンタックスは次の通りです。

function *dataType funcName*(*Arg1*, *Arg2*, ..., *ArgN*) {*script*;}

必要な Origin のバージョン:8.6SR0

Note:

1. 関数名は 42 文字以下である必要があります。
2. 引数と戻り値は、**string, double, int, dataset**, **tree** のデータ型をサポートしています。引数のデフォルトデータ型は、**double** です。戻り値のデフォルトデータ型は、**int** です。
3. デフォルトでユーザ定義関数の引数は値で渡され、関数内のその引数の値は、関数の外部では利用できません。 しかし、リファレンスで渡す引数は、関数内部で引数の値に変わり、関数の外部でも利用できます。これは **REF** というキーワードを使って可能です。

数値関数を使った簡単なサンプルがあります。

```
// この関数は数値の 3 乗根を計算する
function double dCubeRoot(double dVal)
{
    double xVal;
    if(dVal<0) xVal = -exp(ln(-dVal)/3);
    else xVal = exp(ln(dVal)/3);
    return xVal;
}
// 次のようになる
dcuberoot(-8)=;
```

以下の関数は、データセットの幾何平均を計算します。

```
function double dGeoMean(dataset ds)
{
    double dG = ds[1];
    for(int ii = 2 ; ii <= ds.GetSize() ; ii++)
        dG *= ds[ii]; // データセット内のすべての値が掛け算される
    return exp(ln(dG)/ds.GetSize());
}
// 引数はデータセットを返すもの
dGeoMean(col("Raw Data"))=;
```

この例は、範囲を引数で受け付けて、その範囲内のデータの平均を返す関数を定義します。

```
// 範囲の平均を計算
function double dsmean(range ra)
{
   stats ra;
   return stats.mean;
}
// アクティブブックの最初のシートの
// 全ての列を指定する範囲を渡す
range rAll = 1!(1:end);
dsMean(rAll)=;
```

この例は、日付データセット内の特定の曜日の出現を数える関数を定義します。

```
function int iCountDays(dataset ds, int iDay)
{
    int iCount = 0;
    for(int ii = 1 ; ii <= ds.GetSize() ; ii++)
    {
        if(weekday(ds[ii], 1) == iDay) iCount++;
    }
    return iCount;
}
// ここでは、金曜日を数える
iVal = iCountDays(col(1),6); // weekday(data, 1) で 6 は金曜日
iVal=;
```

関数は、データセットを返すこともできます。

```
// データセットから負の値のみ取得
function dataset dsSub(dataset ds1)
{
    dataset ds2;
    int iRow = 1;
    for(int ii = 1 ; ii <= ds1.GetSize() ; ii++)
    {
        if(ds1[ii] < 0)
        {
            ds2[iRow] = ds1[ii];
            iRow++;
        }
    }
    return ds2;
}
// 列 1 の負の値すべてを列 2 に割り当て
col(2) = dsSub(col(1));
```

または文字列を返します。

```
// サブ文字列が現れるデータセット内のすべての値を取得
function string strFind(dataset ds, string strVal)
{
    string strTest, strResult;
    for( int ii = 1 ; ii <= ds.GetSize() ; ii++ )
```

```
    {
        strTest$ = ds[ii]$;
        if( strTest.Find(strVal$) > 0 )
        {
            strResult$ = %(strResult$)%(CRLF)%(strTest$);
        }
    }
    return strResult$;
}
// "hadron"が現れる列 3 のすべてのインスタンスを取得

string MyResults$ = strFind(col(3),"hadron")$; // '$' 記号で終わることに注意
MyResults$=;
```

リファレンスで引数を渡す

この例は、**tree** ノードの値を整数として返す関数を示します。(tree 変数の 1 要素)さらに、リファレンスで渡すことは、**REF** キーワードを使って、示しています。

```
// 関数定義
Function int GetMinMax(range rr, ref double min, ref double max) {
  stats rr;
  //stats X ファンクションを実行した後、同じ名前を持つ
  //LabTalk の tree 変数を作成/更新
  min = stats.min;
  max = stats.max;
  return stats.N;
}

// GetMinMax 関数を呼び、ワークシート全体に対する最小値と最大値を探す
double y1,y2;
int nn = getminmax(1:end,y1, y2);
type "Worksheet has $(nn) points, min=$(y1), max=$(y2)";
```

LabTalk 関数で tree 変数を使った詳細なサンプルやリファレンスで変数を渡すサンプルが Originlab の wiki ページにあります。

リファレンスで文字列の引数を渡す別のサンプルが以下にあり、これは関数呼び出しで、変数の最後に$を付けません。

```
//最初のシートの範囲文字列を返す
//実際の新しいブックのショートネームが Name$で返される
Function string GetNewBook(int nSheets, ref string Name$)
{
        newbook sheet:= nSheets result:=Name$;
        string strRange$ = "[%(Name$)]1!";
        return strRange$;
}
```

上記関数を呼び出すとき、Name$引数が次に示すように、$記号を付けないことが重要です。

```
string strName$;
string strR$ = GetNewBook(1, strName)$;
strName$=;
strR$=;
```

データセット関数

Origin は、double 型の引数を受け付け、double 型で返す数学関数もサポトしています。このような関数の一般的なシンタックスは次の通りです。

funcName(X) = ***expressionInvolvingX***.

これらのデータセット関数が定義されるとき、これらを呼び出し、名前でデータセットが作成されます。このデータセットは、関数と結びつき、Origin プロジェクトの一部として保存されます。一度定義したら、データセット関数は、名前で参照され、組み込み関数のように使用することができます。

例えば、スクリプトウィンドウに次のスクリプトを入力して、**Salary** という関数を定義します。

```
Salary(x) = 52 * x
```

一度定義したら、この関数をいつでも次のような形式で呼び出すことができます。

```
Salary(100)=
```

これは、**Salary(100)=5200** のような結果となります。この場合には、結果のデータセットは 1 つの要素のみを持ちます。しかし、ベクターデータ(またはデータセット)が入力引数として渡されると、出力は入力と同じ数の要素のデータセットとなります。

他のデータセットと同様、ユーザ定義のデータセット関数は、**作図のセットアップ**のようなダイアログや**レイヤ n** のようなダイアログ内の**利用可能なデータ**内に表示されます(そして、他のデータセットのようにプロットできます。).

関数が定義されたとき、2D グラフレイヤがアクティブレイヤの場合、100 ポイントのデータセットが X 軸スケールを使って、X 範囲として作成され、関数データセットが自動的にプロットレイヤに追加されます。

関数グラフテンプレート(FUNCTION.OTP, **標準**ツールバーまたは**ファイル**:**新規**からアクセス可能) も作成され、データセット関数をプロットします。

Origin の**関数グラフ**の機能は、新しいデータセット関数に組み込みおよびユーザ定義関数の組合せから簡単に作成できます。さらに、新しく作成した関数はリファレンスに対してすぐにプロットされます。

この機能には 2 つのどちらかの方法でアクセスできます。

1. 「**標準**」ツールバーの「**新しい関数ウィンドウ**」ボタンをクリックします。 
2. Origin メニューから、**ファイル**:**新規**を選び、リストから**関数**を選び、OK をクリックします。

開いた**作図の詳細**ダイアログの**関数**タブに、*F1(x) = 5*sin(x)+1* のように関数定義を入力し、**OK** をクリックします。関数がグラフにプロットされます。

グラフの**新しい関数**ボタンをクリックして、**作図の詳細**に別の関数を追加すれば、別の関数を定義できます。**OK** を押すと、新しい関数プロットがグラフに追加されます。他の関数に対しても行う場合には繰り返します。

フィット関数

多くの一般的な関数をサポートするだけでなく、Origin は非線形曲線フィットで使用できる自分のフィット関数を作成することができます。ユーザ定義のフィット関数は、新しいデータセットを生成するのに使われますが、それらを呼び出すのは、特別なシンタックスが必要です。

nlf_FitFuncName(ds, ***p1*****,** ***p2*****, ...,** ***pn*****)**

ここで、フィット関数は **FitFuncName** という名前で、**ds** は独立変数として使われるデータセット、*p1--pn* はフィット関数のパラメータです。

単純な例として、y 切片と勾配が入力パラメータとなる **MyLine** という単純な直線のフィット関数を定義する場合、アクティブワークシートの列 C を独立変数、列 D を関数出力に使うには、次のように入力します。

```
// 切片 = 0, 傾き = 4
Col(D) = nlf_MyLine(Col(C), 0, 4)
```

関数のスコープ

ユーザ定義関数は制御可能なスコープ(変数のような)を持ちます。スコープについては、データ型と変数をご覧下さい。プロジェクトレベル変数があるので、プロジェクトレベル関数はないことに注意してください。

セッション変数と同様に、関数のスコープは、**@global=1** という割り当てを使って関数定義を行うことで、現在の Origin セッションのプロジェクトで一般的に使えるように拡張することができます。

@global=1 を使ってセッションレベルで関数を使用できるように、プロジェクトの ProjectEvents.OGS ファイルの中で、関数を定義することで、関数とプロジェクトを結びつけることができます。

セッションをまたいで使用使用するためのユーザ定義関数を作成するには、.OGS ファイルの関数定義を実行するために、**MACROS.CNF** ファイル内にコマンドを追加します。

ユーザ定義関数は、LabTalk スクリプトをサポートしている場所なら Origin プロジェクトのどこからでもアクセスでき、定義のスコープはこのような使用に適用できます。例えば、double 型または dateset を返す**@global=1** で定義された関数は、**値の設定**ダイアログの**列の条件式**で使うことができます。

@global=1 を先に定義をしておけば、その関数はどこからでも使用できるようになります。

```
[Main]
        @global=1;  // 次の関数をセッションレベルとして定義
        function double dGeoMean(dataset ds)
        {
            double dG = ds[1];
            for(int ii = 2 ; ii <= ds.GetSize() ; ii++)
                dG *= ds[ii]; // データセット内の全ての値は全て掛け算する
            return exp(ln(dG)/ds.GetSize());
        }
        // [main]セクション内の関数を呼び出す
        dGeoMean(col(1))=;
[section1]
        // この関数はこのセクションでも呼び出せる
        dGeoMean(col(1))=;
```

関数が **@global=1** なしで ***.ogs** ファイルに定義されている場合、そのセクション内でのみ呼び出せます。

```
[Main]
        function double dGeoMean(dataset ds)
        {
            double dG = ds[1];
            for(int ii = 2 ; ii <= ds.GetSize() ; ii++)
                dG *= ds[ii]; // データセット内の全ての値は全て掛け算する
            return exp(ln(dG)/ds.GetSize());
        }
        // [main] セクション内の関数を呼び出す
        dGeoMean(col(1))=;
[section1]
        // 関数はこのセクションでは呼び出せない
        dGeoMean(col(1))=;  // エラー:未定義の関数
```

@global=1 を使用しないでブロック内で定義されると、そのブロックの外では呼び出せません。

```
[Main]
{   // 括弧内の関数を定義
        function double dGeoMean(dataset ds)
        {
            double dG = ds[1];
            for(int ii = 2 ; ii <= ds.GetSize() ; ii++)
                dG *= ds[ii]; // データセット内の全ての値は全て掛け算する
            return exp(ln(dG)/ds.GetSize());
        }
}
        // 括弧の外では関数を呼び出すことはできない
        dGeoMean(col(1))=;   // エラー:未定義の関数
```

チュートリアル:複数の関数機能を使う

次のチュートリアルは、Origin のプロジェクトレベルにユーザ定義関数を追加し、その関数を使って関数グラフを作成する方法を示しています。

1. 新しいプロジェクトを開始し、**表示**:**コードビルダ**メニューから、コードビルダを開きます。
2. 左側のツリーパネルの Project ブランチを開き、**ProjectEvents.OGS** ファイルをダブルクリックして開きます。このファイルはデフォルトで新しいプロジェクトに存在しています。
3. **[AfterOpenDoc]** セクションの下に次のコード行を追加します。

   ```
   @global=1;
   Function double myPeak(double x, double x0)
   {
   double y = 10*exp(-(x-x0)^2/4);
   return y;
   }
   ```

4. ファイルを保存し、コードビルダを閉じます。
5. Origin で、プロジェクトを希望の場所に保存します。OGS ファイルがプロジェクトと一緒に保存され、ユーザ定義関数がプロジェクト内で利用できるようになります。
6. 保存したプロジェクトを再び開きます。これが契機となり、**[AfterOpenDoc]** セクションが実行され、myPeak 関数が定義されます。
7. 「**標準**」ツールバーの「**新しい関数ウィンドウ**」ボタンをクリックします。

8. 作図の詳細ダイアログの関数タブに、次のような関数定義を入力します。

   F1(x) = myPeak(x, 3)

   OK をクリックすると、関数がグラフにプロットされます。

9. グラフの**新しい関数**ボタンをクリックして、**作図の詳細**に別の関数を追加します。

   F2(x) = myPeak(x, 4)

   そして OK をクリックします。

10. 2 つ目の関数プロットがグラフに追加されます。

11. 再びプロジェクトを保存し、再度開きます。2 つの関数プロットは、プロジェクトと一緒に保存されたユーザ定義関数を参照しているので、これらは利用可能になっています。

12. 最初に Origin を終了し、再び Origin を開き、プロジェクトを再び実行し、**myPeak** 関数がプロジェクトをロードしたときに定義されることをチェックすることで、本当に上記の動作が行われるかを自分で確認することができます。

5.2. 特別な言語機能

これらのページは、LabTalk スクリプト言語の一般的な機能についての情報があります。このセクションのコンセプトおよび機能は Origin 固有のものです。

このセクションで以下の項目を説明します

- 範囲表記
- 置換表記
- プロットラベルで使用される構文と表記法
- LabTalk オブジェクト
- Origin オブジェクト
- 文字列レジスタ
- X ファンクションの紹介

5.2.1. 範囲表記

内容

範囲についてのイントロダクション

Origin プロジェクトの内部では、データはワークシートの列、行列、非接続データセット、グラフの 4 つの主要な場所に存在します。これらの形式のいずれも、範囲データ型を使って、簡単に標準的な方法でデータにアクセスすることができます。

範囲変数を作成すると、範囲データを直接操作することができ、範囲を読み書きできます。以下のサンプルは、範囲変数の作成と多くのデータ型の使用方法を示すものです。

Origin 8.0 以前のバージョンでは、データは cell(), col() , wcol() 関数だけでなく、データセットを使ってアクセスしていました。現在でも cell(), col(), wcol() 関数は、アクティブブックのアクティブシートを操作する場合に、データアクセスするのに効果的です。範囲表記は、本質的には、これらの関数を拡張して、Origin プロジェクト内のどのブック、シート、プロットでもアクセスできるようにするものです。

Note : *すべての X ファンクションが複数列や非隣接列などの複雑な範囲を扱える訳ではありません。ドキュメントにサポートを示す表記がない場合は、いくつか試してみてください。*

Note :グラフ内のデータはデータプロットの形式で、それらは本質的には列、行列、非接続データセットへの参照となります。グラフには実際のデータは保存されません。

宣言とシンタックス

他のデータタイプと同様、次のシンタックスを使って範囲(**Range**)変数を宣言できます。

range [-*option*] *RangeName* = *RangeString*

範囲割り当ての左側は、範囲割り当てのすべての形式に対して同じ形です。大括弧([])はオプションスイッチはオプション設定できるパラメータで、データタイプによって利用可能なオプションスイッチは異なります。詳細は範囲の**データ型セクション**をご覧ください。範囲名は、Origin の変数名の規則に従います。システム変数名は使用しないでください。

範囲割り当ての右側、***RangeString*** は、範囲が示すオブジェクトの種類により変わります。個々の範囲文字列は、下記のセクションの範囲のデータ型で説明されています。

範囲表記は、範囲変数を定義するために、排他的に使われます。シンタックスの代入文のどちらか一方では、データアクセスのための一般表記として使うことはできません。

Origin オブジェクトにアクセスする

範囲変数は、Origin オブジェクトの次の種類に割り当てられます。

- 列
- ワークシート
- ページ
- グラフレイヤ
- 非接続データセット

一度割り当てると、範囲はそのオブジェクトを表すので、範囲変数を使って、そのオブジェクトのプロパティとメソッドにアクセスできます。

範囲は、標準的な Origin オブジェクトのいくつかのサブセットまたは組合せで構成されます。サンプルには以下が含まれます。

- 列の部分範囲
- セルブロック

- XY 範囲
- XYZ 範囲
- 複合範囲

範囲のデータ型

ワークシートデータ

ワークシートデータに対して、***RangeString*** は次の形式を取ります。

[*WorkBookName*]*SheetNameOrIndex*!*ColumnNameOrIndex*[*CellIndex*]

ここで、*ColumnName* は、列の*ロングネーム*または*ショートネーム*のどちらかにすることができます。

どの ***RangeString*** でも、シート、列、行の一組のインデックスを、それぞれ ***index1*:*index2*** のようにコロンで分けて、シート、列、行の連続した区間の開始と終了を指定することができます。キーワード **end** を ***index2*** に使うと、指定したオブジェクトのすべてを範囲とすることを示しています。例えば、

```
range rs = [Book1]4:end!            // Sheet4 から最後まで
range rd = [Book2]Sheet3!5:10;      // 列 5~10 まで
```

行の場合、インデックスは角括弧で囲む必要があり、ワークシート列のいくつかの行に対する全範囲割り当てのステートメントは、次のようになります。

```
range rc1 = [Book1]Sheet2!Col(3)[10:end];    // 10 行目から最後まで
range rc2 = [Book1]Sheet2!Col(3)[10:20];     // 10 行目から 20 行目まで
```

セルの内容にアクセスする古い方法ですが、Cell 関数 は現在でもサポートされています。

範囲を使って、列ラベル行にアクセスする方法については、メタデータにアクセスする および 列ラベル行参照テーブルをご覧下さい。

列

アクティブワークシートの列に対する範囲変数を宣言するとき、ブックとシートの部分を次のように省略することができます。

```
range rc = Col(3)
```

さらに、実際の列を識別でき、次のように単純な表記にすることができます。

```
range aa=1;          // アクティブワークシートの col(1)
range bb=B;          // アクティブワークシートの col(B)
range cc="Test A"; // アクティブワークシートのロングネーム ("Test A") の列
```

Note: 文字列範囲の表現で使用される引用符はロングネームを表します。このロングネームがワークシート内の表記と同じ(例えば、1!)でも、範囲文字列"1!" は 1!というロングネームがついている列を表します。システム変数**@RPQ** を使用するとこの動作を止めることができます。詳細はこの表をご確認ください。

複数の範囲変数をカンマで分けて、同じ行で宣言できます。上記の例は、次のように書くことができます。

```
range aa = 1, bb = B, cc = "Test A";
```

異なるブックのシートや同じシートのすべてを参照する必要がある場合、ブックのシート部分を次のように組み合わせることができます。

```
range [Book2]Sheet3 aa=1, bb=B, cc="Test A";
```

Origin の列のロングネームは固有にする必要はない(例: 同じワークシートの複数列に同じロングネームを使うことが可能)ので、ショートネームとロングネームを一緒に指定するとより正確になります。

```
range dd = D"Test 4";  // Col(D), ロングネーム'Test 4'を範囲に割り当て
```

列の範囲を割り当てたら、それを使って列のパラメータにアクセスしたり、変更することができます。

```
range rColumn = [Book1]1!2;          // 範囲は列
rColumn.digitMode = 1;               // 表示のための小数点桁数を使用
rColumn.digits = 2;                  // 2 桁を使用
```

または、計算を実行します。

```
// 各アクティブワークブックのシート 1, 2, 3 の列に割り当てる
range aa = 1!col(1);
range bb = 2!col(1);
range cc = 3!col(1);
cc = aa+bb;
```

異なるシートにあるデータを算術演算するときには、範囲変数を使う必要があります。範囲文字列への直接参照はまだサポートされていません。例えば、スクリプト **Sheet3!col(1) = Sheet1!col(1) + Sheet2!col(1);** は動作しません。範囲変数を宣言せずに 1 行で書く必要がある場合、別の方法として、データセット置換を使用します。

ページとシート

1 列のデータだけでなく、範囲はページオブジェクトのどの部分にでもアクセスするのに使うことができます。

範囲変数を使って、ワークブック全体にアクセスします。

```
// 'rPage' は、'Book1' というワークブックを示す
range rPage = [Book1];

// 'Book1' のロングネームを "My Analysis Worksheets" にセット
rPage.LongName$ = My Analysis Worksheets;
```

範囲変数を使ってワークシートにアクセスします。

```
range rSheet = [Book1]Sheet1!;       // 範囲はワークシート(WKS オブジェクト)
rSheet.name$ = "Statistics";         // Sheet1 の名前 を "Statistics"に変更
rSheet.AddCol(StdDev);               // StdDev という列を追加
```

列の部分範囲

範囲変数を使って、次のように列の部分範囲を指定することができます。

```
// book1 sheet2 の col(a) の部分範囲
range cc = [book1]sheet2!col(a)[3:10];
```

または、目的のワークブックとワークシートがアクティブな場合、短い表記を使うことができます。

```
// book1 sheet2 の col(a) の部分範囲
range cc = col(a)[3:10];
```

範囲変数を使用して、列の一部分だけで計算やその他の操作を実行できます。例えば、

```
range r1=1[5:10];
range r2=2[1:6];
r1 = r2; // 列 2 の 1 行目から 6 行目までの値を列 1 の 5 から 10 行目までに移動
r1[1]=;
// これで 1 行 2 列目と同じ値を 5 行 1 列目に出力
```

セルのブロック

次のように範囲を使って 1 つのセルまたはセルブロック(複数の列と行をまたがるセル)にアクセスできます。

```
range aa = 1[2];                    // cell(2,1), 列 1 の 2 行目
range bb = 1[1]:3[10];              // cell(1,1) から cell(10,3)まで
```

Note:セルブロックを表す範囲変数は、X ファンクションの引数としてのみ利用でき、直接計算することはできません。

オプションスイッチ –v

必要な Origin のバージョン:9.1SR0

ワークシートデータでは、**-v** スイッチで 1 つのブロックを範囲として定義し、その値を一時ベクトル内に保存できます。これにより、同じ大きさでもブロックの形が異なるブロック間のデータ移行が行えます(つまり、行から列への値の割り振りが可能になります)。

次のサンプルは、このオプションスイッチをどのように使用するかを示します。

```
//新しいブックにサンプルデータをインポート
fname$=system.path.program$ + "\Samples\Statistics\automobile.dat";
newbook;
impasc;
//列 B と C の全行をブロックとして定義
range -v r1 = B[1]:C[end];
// 新しいシートを作成
newsheet;
// ブロックを大きさの決まった列 A と B を定義
range -v r2 = 1[1]:2[r1.GetSize()/2]; // ブロックの大きさは 2 列×行数
//最初のブロックの値を 2 つ目のブロックに定義
r2 = r1;
```

ベクトルはデータを列の順番で保存し、”形”にかかわらず移動先のブロックへ移動します。

```
// 新しいブックにサンプルデータをインポート
fname$=system.path.program$ + "\Samples\Statistics\abrasion_raw.dat";
newbook;
```

```
impasc;
//  1 つのブロックを列 A と列 B のすべての行で定義
range -v ra1 = 1[1]:2[end];
// 新しいシートを作成
newsheet;
// ブロックを 1 列で定義、ra1 ブロックを大きさとする
range -v ra2 = 1[1:ra1.GetSize()];
// 1 番目のブロックの値を 2 番目のブロックに割り振る
ra2 = ra1;
col(1)[L]$ = Combined;
```

Note:目的のブロックで定義される列は割り振られる前に存在する必要があります。

行列データ

行列データに対して、***RangeString*** は、

[MatrixBookName]MatrixSheetNameOrIndex!MatrixObject

次のような構文で代入文を作成できます。

```
// MBook1、MSheet1 の 2 番目の行列オブジェクト
range mm = [MBook1]MSheet1!2;
```

RangeName[row, col]の表記を使って行列範囲のセルの内容にアクセスします。例えば、

```
range mm=[MBook1]1!1;
mm[2,3]=10;
```

行列に複素数が含まれる場合、複素数を表す文字列は次のようにアクセスできます。

```
string str$;
str$ = mm[3,4]$;
```

グラフデータ

グラフデータに対して、***RangeString*** は、

[GraphWindowName]LayerNameOrIndex!DataPlot

サンプルの代入文は次のようなものです。

```
range ll = [Graph1]Layer1!2;        // Graph1, Layer1 の 2 番目の曲線
```

オプションスイッチ –w, –wx, –wy, –wz

グラフウィンドウに対して、range -w、range -wx、range -wy、range -wz オプションを使って、プロットしたデータセットのワークシート列範囲を取得できます。

range -w は常にワークシートの従属変数（2D プロットでは Y 値、3D プロットでは Z 値あるいは行列オブジェクト）のワークシート範囲を取得します。Origin9.0SR0以降では、range -w による複数の範囲がサポートされました。

range -wx、range -wy、range -wz はそれぞれ対応するワークシートの X、Y、Z 値を取得します。

range -wx、range -wz 必要な Origin のバージョン: 9.0SR0

```
// グラフウィンドウをアクティブウィンドウにします。 ...
// 最初のデータプロットの Y 値のワークシート範囲を取得
```

```
range -w rW = 1;

// 対応する X 値のワークシート範囲を取得
range -wx rWx = 1;

// 対応する Y 値のワークシート範囲を取得
range -wy rWy = 1;

// 対応する Z 値のワークシート範囲を取得
range -wz rWz = 1;

// 最初のデータプロットのグラフ範囲を取得
range rG = 1;

// 現在の選択(%C)を取得; マーカー間のデータを決定
range -w rC = %C;
```

上記のスクリプトで、**rW = [Book1]Sheet1!B** ですが、**rG = [Graph1]1!1** であることに注意して下さい。

グラフのデータセレクタの範囲

データセレクタツールを使って、グラフの 1 つ以上の範囲を選択し、LabTalk から参照することができます。選択した範囲が 1 つでは、システム変数 MKS1, MKS2 を使うことができます。v8.0 SR6 以降では、新しい X ファンクション **get_plot_sel** が追加され、解析するために選択範囲を文字列に取得できます。次のサンプルは、現在のグラフの各範囲を選択する方法を示しています。

```
string strRange;
get_plot_sel str:=strRange;
StringArray sa;
sa.Append(strRange$,"|"); // トークン化する
int nNumRanges = sa.GetSize();
if(nNumRanges == 0)
{
        type "there is nothing selected";
        return;
}
type "Total of $(nNumRanges) ranges selected for %C";
for(int ii = 1; ii <= nNumRanges; ii++)
{
        range -w xy = sa.GetAt(ii)$;
        string strWks$ = "Temp$(ii)";
        create %(strWks$) -wdn 10 aa bb;
        range fitxy = [??]!(%(strWks$)_aa, %(strWks$)_bb);
        fitlr iy:=xy oy:=fitxy;
        plotxy fitxy p:=200 o:=<active> c:=color(red) rescale:=0 legend:=0;
        type "%(xy) fit linear gives slope=$(fitlr.b)";
}
// 完了したら全データマーカーを消去
mark -r;
```

次の文書も参考にして下さい。Create (Command) (非接続データセットの作成), [??] 範囲表記 (非接続データセットから範囲を作成), **fitlr** X ファンクション, StringArray (オブジェクト) (特に、**Append** メソッドは、Origin 8.0 SR6 から使用できます)

X 値を使用した部分範囲指定

XY 範囲で操作するとき、部分範囲を X 値で指定できます。シンタックスは次のようになります。

1. ワークシートからの場合
   [*WorkBookName*]*SheetNameOrIndex*!*YColumnNameOrIndex*[x*X1:X2*]

サンプル:

```
// XY データ列 1 と列 2 の部分範囲 x=0.15 から 0.2 を使用
range rxy = (1, 2)[x0.15:0.2];
```

1. グラフからの場合
   [*GraphWindowName*]*LayerNameOrIndex*!*DataPlot*[x*X1:X2*]

サンプル:

```
// Graph1 Layer1 の 2 番目の部分範囲 XY
range rxy2 = [Graph1]Layer1!2[x5:20];
```

以下のサンプルでは、plotxy X ファンクションを使用してグラフを作図し、smooth X ファンクションで、データの部分範囲をスムージングします。

```
// 新しいブックにデータをインポート
newbook;
fname$ = system.path.program$ + "\Samples\Signal Processing\EMG Recording.dat";
impasc;

// XY 部分範囲として、X が 5 から 5.5 と、9.3 から 9.8 を定義
range rxy1 = (1, 2)[x5:5.5];
range rxy2 = 2[x9.3:9.8];
plotxy rxy1 plot:=200;  // 最初の XY 部分範囲で折れ線グラフを作図
smooth -r 2 rxy2 method:=le;  // 2 番目の XY 部分範囲で Loess のスムージング
```

X 値を使用して部分範囲を指定するときは、X データは単調である必要があります。

非接続データセット

非接続データセットは、ワークシートの列に似ていますが、ブック－シート－列の構成のオーバーヘッドがありません。これは、通常、create コマンド で作成されるか、Dataset 宣言せずに代入文で自動的に作成されます。

非接続データセットの ***RangeString*** は

[??]!*LooseDatasetName*

代入文は以下のようになります。

```
range xx = [??]!tmpdata_a;        // 非接続データセット 'tmpdata_a'
```

この動作を確認するために、**plotxy** X ファンクションを使って、非接続データセットのグラフをプロットします。

```
// 2 つの非接続データセットを作成
```

```
create tmpdata -wd 50 a b;
tmpdata_a=data(50,1,-1);
tmpdata_b=normal(50);
// 非接続データセットの範囲と明示ポイントを宣言
range aa=[??]!(tmpdata_a, tmpdata_b);
// 散布図を作図
plotxy aa;
```

非接続データセットはプロジェクトに属し、宣言される Dataset 変数とは異なり、セションまたはローカルスコープを持ちます。Dataset 変数も内部的には非接続データセットですが、非接続データセットは計算のみの使用に限られ、例えば、プロットを作成するのに使用することはできません。

範囲のメソッド

範囲変数が作成されると、以下のメソッドがこの範囲内で使用できるようになります。

メソッド	説明
range.getSize()	範囲の大きさを返します。このメソッドは、列、行列オブジェクト、グラフプロット、セルのブロック、非接続データセットなどのデータセット範囲で動作します。なお、セルのブロックに関しては、範囲宣言で指定した最初のサブ列の大きさだけ返します。
range.setSize()	範囲の大きさをセットします。このメソッドは、列、行列オブジェクト、グラフプロット、セルのブロック、非接続データセットなどのデータセット範囲で動作します。セルのブロックの場合、範囲宣言で指定した最初のサブ列の大きさだけセットします。
range.getLayer()	範囲にレイヤ（グラフレイヤ、ワークシート、行列レイヤ）が付属している場合、このメソッドはそのレイヤの **UID** を返します。レイヤ名を取得するには、メソッドの後に**$**記号が必要です。例えば、"rng.getLayer()$ = "のようになります。
range.getPage()	範囲にページ（グラフページ、ワークブック、行列ブック）が付属している場合、このメソッドはそのレイヤの **UID** を返します。ページ名を取得するには、メソッドの後に**$**記号が必要です。例えば、"rng.getPage()$ = "のようになります。
range.sub(*name*/*index*)	このメソッドは、名前(*name*)または、インデックス(*index*)でデータ範囲から部分範囲を取得します。これは、仮想行列で便利です。例えば、*ztitle* という名前の仮想行列がある場合、*ztitle.sub(y);* (名前) または *ztitle.sub(1);* (インデックス)を使用すると Y 値のデータセットを返します。さらに、*ztitle.sub(y)[3]=;* や *ztitle.sub(y)[3]$=;* のような表現を使用することでこのデータセットの 3 番目の値を取得できます。
range.reverse()	このメソッドはデータセット範囲、例えば列、行列オブジェクト、グラフプロット、セルのブロック、非接続データセット等に動作し、範囲内のデータ順を反対にします。もし範囲がセルのブロックの場合、宣言した範囲で指定した最初のサブ列のデータ順だけを反対にします。colReverse X ファンクションは同じ事を行います。

範囲固有の使用法

範囲データを操作する

列の範囲は直接データを操作するのに使用することができます。列名を直接使うのではなく、範囲を使う大きなメリットは、どのページまたはレイヤがアクティブであるかを考慮する必要がないということです。

例えば、

```
// 2 つの範囲変数 v1 と v2 を宣言
range [Book1]Sheet1 r1=Col(A), r2=Col(B);

// [book1]sheet1 がアクティブであれば、col(A)=data(1,30) と同じ
r1 = data(1,30);
r2 = uniform(30);

// グラフを作成するので [Book1]Sheet1 はアクティブではなくなる
plotxy 2;
sec -p 1.5;             // 1.5 秒待つ
r2/=4;                  // 範囲は動作するが、 col(A)/=4 は動作しない
sec -p 1.5;             // 1.5 秒待つ
r2+=.4;
sec -p 1.5;             // 1.5 秒待つ
r1=10+r1/3;
```

セルの範囲を示す列範囲変数を直接計算することができます。例えば、

```
range aa = Col(A)[10:19]; // A 列の 10 から 19 行目
aa += 10;                 // aa のすべての要素は 10 ずつ増加
```

列の部分範囲のサポートが拡張されました。

```
// 列 1 の 7 行目から 13 行目と列 2 の 3 行目から 4 行目で構成される範囲
// 括弧とカンマの区切りを使用
range rs = (1[7:13], 2[3:4]);
del rs; // バージョン 8.0 SR6 からサポート

// 部分範囲間をコピー
range r1 = 1[85:100];
range r2 = 2;
// r1 を列 2 の先頭にコピー
r2 = r1; // 8.1 からサポート
// v8.1 も部分範囲に完全/不完全コピー
range r2 = 2[17:22];
r2 = r1; // r1 から 6 つの値をコピー
range r2 = 3[50:200];
r2 = r1; // 元は 16 個の値があるので、65 行までのみコピー
```

範囲の動的割り当て

列番号を変化させたり、別の範囲変数の名前を使って、実行時に自動的な方法で新しい範囲を作成できると役立つ場合があります。

列インデックスの式を使って新しい範囲を定義する

関数 wcol()は、実際の列インデックスを実行時に分割するのに使用されます。

```
int nn = 2;
range aa=wcol(2*nn +1);
```

既存の範囲を使って新しい範囲を定義する

次のスクリプトは、%( ) 置換表記 と wks (オブジェクト)メソッドを使って、別の範囲に基づいた新しい範囲を作成する方法を示しています。**%( )** 置換で範囲変数を使用すれば、タイプしなくても、常に**[Book]Sheet!**文字列を認識します。

```
range rwks = sheet3!;
range r1= %(rwks)col(a);
```

この場合、新しい範囲 *r1* は **Sheet3!Col(A)**となります。

既存の範囲を基にした新しい範囲を構成するこのメソッドは、ワークシート範囲を最初に宣言するようコードを集中し、列範囲を宣言するのに使用することができるので、とても役に立ちます。ここでは、*rwks* 変数を使って、列を Sheet 3 に追加します。

```
rwks.addcol();
```

そして、範囲 **rwks** の最後の列(一番右側)を認識する別の列を定義します。これは新しく作成された列を指します。

```
range r2 = %(rwks)wcol( %(rwks)wks.ncols );
```

適当な位置に範囲の割り当てがあると、次のように計算や代入がしやすくなります。

```
r2=r1/10;
```

これは範囲 **r1** のデータを 10 で割り、その結果を範囲 **r2** として宣言されている列に入れます。

X ファンクションの引数

多くの X ファンクションは引数として範囲を使用します。例えば、X ファンクション *stats* は、入力としてベクトルデータを取り、その指定した範囲の記述統計量を計算します。次のように入力すると、

```
stats [Book1]Sheet2!(1:end); // book1 の 2 シート目の統計量
stats Col(2);                // アクティブなワークシートの列 2 の統計量

// // 列 1-2, 5-10 行目のセルブロックの統計量
stats 1[5]:2[10];
```

または、範囲変数を使って、同じような操作を行うことができます。

```
/* 1 番目と 2 番目のシートの列 2 の 3-5 行目に対する範囲変数を定義
その範囲に stats X ファンクション を実行*/
range aa = (1,2)!col(2)[3:5]; stats aa;
```

この X ファンクションに対する入力ベクトル引数は範囲変数で指定されます。

いくつかの X ファンクションは、XYRange という特別なタイプの範囲を使います。これは本質的には X と Y を含む複合範囲でエラーバーを含めることもできます。
XYRange の一般的なシンタックスは次の通りです。

```
(rangeX, rangeY)
```

しかし、rangeX 部分を飛ばし、標準的な範囲表記を使って、XYRange を指定します。この場合デフォルトの X データが使われます。
次の 2 つの表記は、XYRange と認識します。

```
(, rangeY)
rangeY
```

例えば、integ1 X ファンクションは入力と出力の両方で XYRange を取ります。

```
// col(1) を X、col(2)を Y として積分
// 列 3 を X, 列 4 を Y として積分曲線を出力
integ1 iy:=(1,2) oy:=(3,4);

// 結果の積分曲線を列 3 に Y, 列 1 の入力 X を共有することを除き、
// 上記と同じ
integ1 iy:=2 oy:=3;
```

範囲変数の一覧、削除、変換

範囲変数の一覧表示

LabTalk の **list** コマンドを使って、範囲変数を含むすべてのセッション変数の名前の一覧とそれらが定義されている内容を出力します。例えば、

```
list a;        // すべてのセッション変数を一覧表示
```

コマンドウィンドウでこのコマンドを実行する場合、次のように出力されます。

```
Session:
 1        MYRANGE    [book1]sheet1!col(b)
 2          MYSTR   "abc"
 3             PI    3.1415926535898
```

Origin 8.1 から、多くのスイッチ (下記) が追加され、特定のセッション変数を出力できます。

オプション	表示される対象	オプション	表示される対象
a	すべてのセッション変数	aa	String 配列 (セッション)
ac	定数 (セッション)	af	ローカル関数 (セッション)
afc	ローカル関数ー全内容 (セッション)	afp	ローカル関数のプロトタイプ (セッション)
ag	グラフィックオブジェクト (セッション)	ar	範囲変数 (セッション)
as	文字列変数 (セッション)	at	ツリー変数 (セッション)
av	数値変数 (セッション)	--	--

範囲変数の削除

範囲変数を削除するには、LabTalk の del コマンドを-ra スイッチと一緒に使います。例えば、LabTalk:Delete (コマンド)

```
range aa=1;  // aa = アクティブワークシートの Col(1)
range ab=2;  // ab = アクティブワークシートの Col(2)
range ac=3;  // ac = アクティブワークシートの Col(3)
```

```
range bb=4;  // bb = アクティブワークシートの Col(4)
list a;      // aa, ab, ac, bb を含む全てのセッション変数を一覧表示
del -ra a*;  // 文字 "a" で始まるすべての範囲変数を削除

// 最後のコマンドは aa, ab, ac を削除
```

下記の表は、範囲変数を削除するオプションの一覧です。

オプション	削除/消去される対象	オプション	削除/消去される対象
ra	ローカル/セッション変数	al	-ra と同じ
rar	範囲変数	ras	文字列変数
rav	数値変数	rac	定数
rat	ツリー変数	raa	String 配列
rag	グラフィックオブジェクト	raf	ローカル/セッション関数

範囲を UID に変換する

各 Origin オブジェクトには、ショートネームとロングネーム、ユニバーサル ID (UID)があります。範囲変数とその UID 間を変換することができ、関数 **range2uid**, **uid2name**, **uid2range** を使って、ページとレイヤの名前を取得できます。使用列については、LabTalk オブジェクト をご覧下さい。

範囲の特殊な表記

XY と XYZ 範囲

特定の X ファンクションへの入力として設計されているので、XY 範囲は、2 つのワークシート列に XY データの属性にした一対データです。また、XY 部分範囲は、X 値を使用して指定できます。同様に、XYZ 範囲は、3 つの要素をもち、XYZ データに対応する 3 つのワークシート列を持っています。

例えば、**fitpoly** X ファンクションは、入力と出力両方の XY 範囲を取ります。

```
// 列 1 と 2 の XY データを 2 次多項式でフィット
// 係数を列 3、XY フィットデータを列 4 と列 5 に入れる
fitpoly iy:=(1,2) polyorder:=2 coef:=3 oy:=(4,5);
```

X に対する#と ?を使った XY 範囲

X ファンクションの引数として範囲を使うため (8.0 SR3) で 2 つの特殊文字'?' と '#' が導入されました。'?' は、強制的にワークシート属性を使うことを示し、範囲の属性が要求に合わない場合、エラーとなります。'#' は、範囲が属性を無視し、X 属性として行番号を使うことを意味します。しかし、Y 列がサンプリング情報の場合、そのサンプリング情報は X を提供するのに使われます。

例えば、

```
plotxy (?, 5);          // if col(5) が X 列の場合エラー
plotxy (#, 3);          // 行番号を X とし、col(3) を Y として作図
```

これらの表記は、ここでのサンプルのように、**plotxy** X ファンクションで特に役立ちます。

```
// 列の属性を使ってワークシートのすべての列をプロット
plotxy (?,1:end);
```

範囲出力でのタグ表記

多くの X ファンクションは、*テンプレート*, *名前* , *インデックス* を含むタグで修正する出力範囲があります。以下は、離散度数を求める X ファンクション, **discfreqs** で使用する例です。

```
discfreqs irng:=1 freq:=1 rd:="[Result]<new template:=table.otw index:=3>";
```

これは、TABLE.OTW というテンプレートをロードして、**Result** というワークブックの 3 番目のシートに直接出力されます。

これらタグの表記のサポートは特定の X ファンクションに依存しますので、作成コードをに含める前に試すことができます。

複合範囲

複合範囲は、複数の部分範囲で構成される範囲です。次のシンタックスを使って、複合範囲を構成します。

```
// 3 つの範囲の基本的な組合せ
(range1, range2, range3)

// 複数シートから共通の列範囲
(sheet1,sheet2,sheet3)!range1

// シート範囲からの共通の列範囲
(sheet1:sheetn)!range1
```

これがどのように動作するかを表示するには、**wcellcolor** X ファンクション を使って範囲を表示し、**plotxy** を使って XYRange を表示します。少なくとも 4 つの列が数値データで入力されている、アクティブなブック/シートを操作しているものとします。

```
// いくつかの異なるブロックを青にする
wcellcolor (1[1]:2[3], 1[5]:2[5], 2[7]) color(blue);

// そのうちのいくつかをフォントの色を赤にセット
wcellcolor (1[3]:4[5], 2[6]:3[7]) color(red) font;
```

plotxy を試すには、最初のシートにいくつか数値を入れ、新しいシートを追加し、2 番目のシートにはより多くの数値を入れます。

```
// 両シートの A(X)B(Y) を同じグラフにプロット
plotxy (1:2)!(1,2);

// ワークブックを再びアクティブにし、シートを追加して、それらにデータを入力
// すべてのシートの 2 行目から 10 行目までの A(X)B(Y) をプロット
plotxy (1:end)!(1,2)[2:10];
```

Note:複合範囲には特有の不明瞭さがあります。例えば、(**r1**,**r2**)のように範囲 **r1** と範囲 **r2** で構成される範囲、**r1** と **r2** という名前の列で構成される XY 範囲例えば、(**r1**,**r2**)です。そのため、与えた範囲変数に割り当てられているオブジェクトの種類を覚えておくことが大切です。

5.2.2. 置換表記

内容

イントロダクション

スクリプトが実行される時、LabTalk のインタプリタに送らます。そして、その操作の中でインタプリタは、%と$で始まる 2 種類の特別な置換表記を探します。置換表記が見つかると、次のセクションで説明しているように、インタプリタは元の文字列を別の文字列で置き換えます。置換された文字列の値は、その文が実際に実行されるまで分かりません。この操作は実行時文字列置換と呼ばれます。

以下に説明するように、3 つの置換法があります。

- 文字列レジスタ置換, %A - %Z
- %( )置換、**%(str$)**, **%(range)**、ワークシート情報、列データセット名、ワークシートセル、凡例などの置換に利用する表記
- $( )置換, **$(*expression*)** 、数式を解釈し、結果を文字列の形式にする

%A - %Z

文字列レジスタは、最も簡単な置換の形式です。文字列レジスタは、スクリプト実行中にその内容によって置換されます。例えば、

```
FDLOG.Open(A);    // ダイアログからファイル名を%A に入れる
%B=FDLOG.path$;   // %B にファイルパスを入れる
doc -open %B%A;   // %B%A はフルパスとファイル名の形式
```

文字列レジスタは、Origin8 でより信頼性の高い文字列変数が導入される前の古いスクリプトでよく使われています。文字列変数を解釈するには、%( )置換が使われます。これについては、次のセクションで説明します。

%( ) 置換

文字列式の置換

LabTalk コマンドは数値の式を受け付けますが、文字列の式は受け付けません。そのため、文字列を引数とする必要がある場合、%( )置換を使った文字列変数または文字列式を渡して、実行時の値を解釈する必要があります。文字列式の最も簡単な形式は、次の例のような 1 つの文字列変数です。

```
string str$ = "Book2";
win -o %(str$) {wks.ncols=;}
```

キーワード置換

%( )置換表記は、タブや改行コードなどの 非出力文字 (制御文字とも言う)を挿入するためにも使われます。LabTalk キーワードを使って、これらの非出力文字にアクセスします。例えば、

```
// 改行、ラインフィード(CRLF)を文字列に挿入
string ss$ = "Hello%(CRLF)Goodbye";
ss$=;      // ANS:'Hello', 'Goodbye' が別の行に出力
// 直接入力可能
type ss$;
// 文字が混ざっている場合 %() 表記を使用
ty I say %(ss$) you say;
```

ワークシート列とセルの置換

次の表記は、文字列としてワークシートセルにアクセスしたり、どのワークブックのシートからでも列のデータセット名を取得できます。Origin 8 以前は、各ブックには 1 つのシートしかなかったので、ブック名だけでその内容を参照できました。Origin 8 は複数のワークシートをサポートしているので、そのブックに 1 つのシートだけが含まれていることが確実でなければ、**[workbookname]sheetname** の形式で特定のシートにアクセスします。

ワークシートのセルの内容を返すには、次のシンタックスを使用します。

- この表記は、ブックのアクティブシートを参照します。

  %(*workbookName*, *column*, *row*)

- ブックとシートを指定する新しい Origin 8 の表記は次のシンタックスです。

  %([*workbookname*]*sheetname*, *column*, *row*[,*format*])

例えば、Book1 のアクティブワークシートの 4 番目の列の 3 行目のセルに 25 という値が含まれている場合、スクリプトウィンドウに次のステートメントを入力すると、A を 25 にセットし、その値を 2 倍して Book1 の別のシートに入れます。

```
A = %(Book1, 4, 3);
%([Book1]Results, 1, 4) = 2 * A;
```

テキストセルの内容を得るには、文字列変数に代入します。

```
string strVar$ = %(Book1, 2, 5); // Note :ここでは最後の '$' は必要ない
strVar$ = ;
```

8.1 より以前では、列と行のインデックスを使う必要があり、数値のセルは常に完全精度を返しました。Origin 8.1 では、*列*に対しては、インデックスと列名の両方がサポートされ、*行*に対しては、ラベル行の文字(例えばロングネームは L)もサポートされています。

任意の *format* 引数もサポートされており、数値セルを文字列に変換する際に指定することができます。Book2 の sheet3 の col(Signal)[3]に 12.3456789 という数値があるものとしますが、小数点以下 2 桁が表示されます(この設定は**列プロパティ**ダイアログで行います)。

```
//現在の列フォーマットを使用して文字列 W をフォーマット
//12.34 が返されます
type "Col(Signal)[3] displayed value is %([Book2]Sheet3,Signal,3,W)";
//フォーマット指定がない場合フル精度
A=%([Book2]Sheet3,Signal,3);
A=;// 12.3456789 が表示される
//あるいは、他のフォーマット表記を使用
type "Showing 3 decimal places:%([Book2]Sheet3,Signal,3,.3)";
```

他のシンタックスを使用してデータのセルフォーマットを管理します。**セルのフォーマット**ダイアログまたは**列プロパティ**ダイアログを使用して編集します。

%([*workbookName*]*sheetName*, @WL, *column*[*row*], W)

以下は、同様のサンプルです。

```
//または表示フォーマットを保つため、@WL オプション付きの他の表現を使用
type "Col(Signal)[3] displayed value is %([Book2]Sheet3, @WL, Signal[3], W)";
//フォーマット指定がない場合フル精度
B=%([Book2]Sheet3,@WL, Signal[3]);
B=;// 12.3456789
```

Note:9.1 SR0 で紹介されるフォーマット文字 *W* は、以前のバージョンの *C* の使用に代わるものです。しかし、Origin は、現在の列フォーマットの値を返すために、**%([workbookName]sheetName, column, row,C)** の使用をサポートしています。

データセットの名前を得るには、次の構文を使用します。

- ブックのアクティブシートに対する古い表記

  %(*workbookName*, *column*)

- 新しい Origin 8 のブックのシート表記

  %([*workbookName*]*sheetName*, *column*)

- インデックスを使うこともできます。

  %([*workbookName*]*SheetIndex*, *column*)

ここで、***column*** は、列名をサポートしている Origin 8.1 より前のインデックスである必要があります。

例えば、

```
%A = %(%H, 2);              // アクティブブックのあくていぶシートの列 2
type %A;
%B = %([Book1]Sheet3,2); // Book1 の Sheet3 の列 2
type %B;
```

上記の例では、アクティブワークシートの列 2 のデータセットの名前が評価式の右側で置換され、%A と%B に割り当てられます。2 番目のケースでは、ブックまたはシートが存在しない場合、エラーは起こりませんが、置換は無効となります。

Note:括弧を使って、文字列レジスタ自体に割り当てを実行するのではなく、文字列レジスタ変数に含まれる名前を持つデータセットで実行される割り当てを強制的に行うことができます。

```
%A = %(Book1,2);    // 列 2 のデータセット名を取得
type %A;            // データセットの名前を出力
(%A) = %(Book1,1); // 列 1 のデータを列 2 にコピー
```

別のシートのデータセットを含めた計算

ブックまたはシートからデータセット名を取得する(データセット置換)機能は、次のような異なるシートにある列間の計算を行うのにとても役立ちます。

```
// Sheet2 と Sheet3 の col(1)を合計し、アクティブシートの col(1)に入れる
col(1)=%([%H]sheet2, 1) + %([%H]sheet3, 1);

// Book2 の最初のシートで col "signal"を減算
//  そして、アクティブブックの Sheet3 の列"calibrated"に結果を入れる
%([%H]sheet3, "calibrated")=col(signal) - %([Book2]1,signal);
```

ロングネームを使う場合、***column*** 名は引用符で囲む必要があります。引用符が無い場合、Origin は最初にショートネームを探し、見つからなければ、ロングネームを使用しようとします。上記のサンプルで、

```
%([%H]sheet3, "calibrated")
```

は、強制的にロングネームを探しますが、

```
%([Book2]1,signal)
```

は、ショートネームを持つ列が無かった場合のみロングネームを使います。

ワークシート情報の置換

置換表記でのワークシート列とセルへのアクセスと同様、@置換 (ワークシート情報置換)は、**@**文字を使って、ワークシート情報やメタデータにアクセスするさまざまなオプションを指定する 2 番目の引数にある列インデックスまたは列名とは区別されます。

以下のシンタックスは、ワークシート情報置換に使用され、アクティブシートでサポートされます。

%(*workbookName*, *@option*, *columnNumber*)

Origin8 で導入された新しい表記を使うことをお勧めします。

%([*workbookName*]*worksheetName*, *@option*, *columnNumber*)

ここで、*option* は次のいずれかです。

オプション	返される値
@#	ワークシート列の総数を返します。*ColumnNumber* は省略できます。
@CS	*columnNumber* 列から右に数えて、一番最初の選択されている Y 列の列インデックス（columnNumber 列を含む）を返します。
@E#	*columnNumber* =1 の場合、ワークシート内の Y エラー列数を返します。*columnNumber* =2 の場合、現在選択されている領域内の Y エラー列数を返します。*columnNumber* が省略されている場合、*columnNumber* は 1 と見なされます。
@H#	**columnNumber** = 1 の場合、ワークシート内の X エラー列数を返します。*columnNumber* =2 の場合、現在選択されている領域内の X エラー列数を返します。*columnNumber* が省略されている場合、*columnNumber* は 1 と見なされます。
@PC	ページのコメント
@PCn	*n* 番目のコメント
@PL	ページのロングネーム
@PN	ページのショートネーム
@SN	シート名
@SC	シートコメント
@OY	選択されている列の 1 番左の Y 列から数えた、*columnNumber* 列までのオフセットを返します。
@OYX	選択されている列の 1 番左の Y 列から数えた、*columnNumber* Y 列までの Y 列のみのオフセットを返します。
@OYY	選択されている列の 1 番左の Y 列から数えた、*columnNumber* X 列までの X 列のみのオフセットを返します。
@T	列属性(1 = Y , 2 = 無属性, 3 = Y エラー, 4 = X , 5 = ラベル, 6 = Z, 7 = X エラー)
@X	ワークシートの最も左にある X 列のインデックス番号を返します。列は左から右へ、列番号が 1 から順に大きくなります。次の範囲のシンタックスを使用します。%(worksheetName, @X);
@Xn	ワークシートの最初の X 列のショートネームを返します。次の範囲のシンタックスを使用します。%(worksheetName, @Xn);
@Y	選択されている列の 1 番左の列から数えた、*columnNumber* 列までのオフセットを返します。
@Y-	指定した列を左に探していき最初に見つかった Y 列の列番号を返します。列が Y 列であれば、columnNumber を返します。Y 列が見つからなければ、0 を返します。次の範囲のシンタックスを使用します。%(worksheetName, @Y-, ColumnNumber);

@Y#	*columnNumber* =1 の場合、ワークシート内の Y 列数を返します。*columnNumber* =2 の場合、現在選択されている領域内の Y 列数を返します。*columnNumber* が省略されている場合、*columnNumber* は 1 と見なされます。
@Y+	指定した列をに探していき最初に見つかった Y 列の列番号を返します。列が Y 列であれば、columnNumber を返します。Y 列が見つからなければ、0 を返します。次の範囲のシンタックスを使用します。%(worksheetName, @Y+, ColumnNumber);
@YS	*columnNumber* 列から右に数えて、一番最初の選択されている Y 列の番号（columnNumber 列を含む）を返します。
@Z#	*columnNumber* =1 の場合、ワークシート内の Z 列数を返します。*columnNumber* =2 の場合、現在選択されている領域内の Z 列数を返します。*columnNumber* が省略されている場合、*columnNumber* は 1 と見なされます。

この表のオプションは、@ option または @ variables で識別できます。テキストラベルオプションページの@ option はワークシート情報置換で使用されます。

情報の保存とインポートしたファイルの情報

@W 変数は、Origin のワークブックおよびワークシート、列だけでなく、インポートしたファイルに保存されているメタデータにアクセスします。

上記と同様のシンタックスを使って、列番号を変数またはノードの情報で置き換えます。

%([*workbookName*]*worksheetName*!*columnName*, *@option*, *varOrNodeName*)

オプション	返される値
@W	*varOrNodeName* の情報を返します。この変数は、ワークブックレベルにあると解釈され、ワークブックオーガナイザで確認できます。これが使用されていると、*worksheetName!ColumnName* を指定する必要がありません。
@WF*n*	*n* 番目のインポートしたファイルに対して *varOrNodeName* の情報を返します。この変数はワークブックオーガナイザで確認できます。
@WS	*varOrNodeName* の情報を返します。この変数は、ワークシートレベルにあると解釈され、ワークブックオーガナイザで確認できます。これが使用されていると、*ColumnName* を指定する必要がありません。
@WC	*varOrNodeName* の情報を返します。この変数は、列レベルにあると解釈され、**列プロパティ**ダイアログで確認できます。

@置換の例

次のスクリプトは、現在選択されている範囲の最初の列の列名を返します。(**selc1** 数値システム変数に関する情報に関しては、システム変数 を参照してください。)

```
%N = %(%H, @col, selc1);  %N =;
```

次の行は、アクティブページのロングネームを文字列変数に返します。

```
string PageName$ = %(%H, @PL);
```

以下のスクリプトは Book2、Sheet3 の 4 番目の列に対して列の種類を返します。

```
string colType$ = %([Book2]Sheet3, @T, 4);
colType$=;
```

インポートフィルタは、ワークブックと一緒に保存されるインポートファイルについての情報のツリー構造を作成することができます。ここで、複数ファイルのインポートに対して、現在のブックにインポートされた 3 番目のデータセットのポイント数を返します。

```
%z=%(%H,@WF3,variables.header.noofpoints);
%z=
```

現在のアクティブワークシートが 6 つの列(XYYYYY)を持っているとします。もし、列 2,4,5 が選択されているとすると、以下のスクリプトは、*columnNumber* 列（3 つ目の引数）から右に数えて、1 番最初の選択された Y 列番号（columnNumber 列を含む）を返します。

```
loop(ii,1,6)
{
    type -l %(%H, @YS, ii),;
}
type;
```

次のように出力されます。

```
2,2,4,4,5,0,
```

凡例の置換

グラフの凡例にも、%( ) 置換表記を使用できます。最初の引数は整数値で、最初の引数がワークシート指定子であるような他の%( )置換とは区別されなければなりません。凡例の置換のシンタックスは次のようなものです。

%(*n*[, *@option*])

ここで、*n* は、現在のレイヤにあるデータプロットのインデックスです。多くのオプションを変数 *n* の後に指定できます。たとえば、データプロットに関連するプロット属性の文字(X, Y, Z) などで、オプションを指定していなければ、Y と見なされます。*@option* は、凡例の内容を制御する引数です。例えば、

```
// 現在のグラフレイヤの凡例で
// 最初の従属データセットのロングネームを表示
legend.text$ = %(1Y, @LL)

// 同等のコマンド（デフォルトの Y が使われる）
legend.text$ = %(1, @LL)
```

凡例に 2 番目の独立データセット（X など）のショートネームを表示するには、以下のように実行します。

```
legend.text$ = %(2X, @LS)
```

@オプションの完全なリストと詳細情報は、凡例の置換表記ページを確認してください。

Note:この凡例の変更スタイルは、単一の凡例項目のみの変更に限られますが、作図の詳細ダイアログで使用することができるよう構成されています。

legendupdate X ファンクションは、スクリプトから凡例を編集するより簡単で包括的な方法を提供します。

$( ) 置換

数値を文字列に変換するには$()置換表記を使用します。この表記は、与えられた数式を実行時に評価し、その結果を文字列に変換して置き換えます。

この表記は、次の構文で使用します。

$(*expression* [, *format*])

ここで、*expression* は数式ですが、通常、1 つの数値または変数（データセットとデータ範囲）であり、*format* は、Origin の出力形式または C 言語形式にすることができます。

もし、*expression* がデータセットあるいは範囲変数の場合、スペースで区切られた値のリストを返します。(必要な Origin のバージョン: 9.1SR0)

例えば、

```
//データセット定義
dataset ds1 = {1, 3, 5, 7};
//置換表記の結果を出力
type $(ds1);//ANS:1 3 5 7;
type $(ds1, *2);//ANS:1.0 3.0 5.0 7.0

// 最初の列にある値を範囲変数 rx に渡す
range rx = col(1);
//範囲表記の結果を出力
type $(rx);
```

デフォルトの形式

角括弧は、*format* が**$()** 置換表記に対して任意の引数であることを示します。もし、*format* を除外すると、Origin は**@SD** システム変数（デフォルト値は 14）により指定された桁または有効桁数で *expression* を実行します。例えば、

```
double aa =  3.14159265358979323846;
type $(aa);    // ANS:3.1415926535898
```

Origin の形式

必要な Origin のバージョン:9.1SR0

Origin は出力形式を整えるためのオプションがいくつかあります。

フォーマット	説明
**n*	*n* 桁の有効桁数を表示
n	*n* 桁の有効桁数を表示, 桁数以下の 0 を消去
S**n*	科学表記で *n* 桁の有効桁数を表示
E**n*	工学表記で *n* 桁の有効桁数を表示

.n	*n* 桁の小数桁数を表示
.n,	整数部分に 1000 の区切り(,)を表示し、*n* 桁の少数桁数を表示
S.*n*	科学表記で *n* 桁の小数桁数を表示
E.*n*	工学表記で *n* 桁の小数桁数を表示
D*c*	文字列 *c* によってカスタマイズされたフォーマットで日付を表示
D*n*	**列のプロパティ**ダイアログボックスの**表示フォーマット**ドロップダウンリストから *n* の形式の日付を表示
T*c*	文字列 *c* によってカスタマイズされたフォーマットで時間を表示
T*n*	**列のプロパティ**ダイアログボックスの**表示フォーマット**ドロップダウンリストから *n* の形式の時間を表示
#n	整数値を *n* 桁で表示、必要な場所で、0 を付加
<prefix>##<sep>###<suffix>	指定したセパレータ(*<sep>*)を桁の間に表示し、オプションで接頭語(<prefix>) や接尾語(<suffix>)を追加します。一つの **#** 記号は、1 桁を意味します。最後の **#** 記号は、単位桁を参照します。最初と 2 番目の **#** の数は変更できます。

このスクリプトブロックは、Origin 形式の例を示しています。

```
xx = 1.23456;
type "xx = $(xx, *2)";  // ANS:1.2
type "xx = $(xx, .2)";  // ANS:1.23

yy = 1.10001;
type "yy = $(yy, *4)";   // ANS:1.100
type "yy = $(yy, *4*)";  // ANS:1.1

zz = 203465987;
type "zz = $(zz, E*3)";  // ANS:203M
type "zz = $(zz, S*3)";  // ANS:2.03E+08

type "$(date(2009/7/20), D1)";  // ANS:2009 年 7 月 20 日

type "$(date(2009/7/20), Dyyyy'-'MM'-'dd)";  // ANS:2009-07-20

type "$(time(14:31:04), T4)";   // ANS:02 PM

type "$(time(14:31:04), Thh'.'mm'.'ss)";  // ANS:02.31.04

type "$(45, #5)";    // ANS:00045

type "$(56000, ##+###)"; //ANS:56+000

type "$(4000, ##+##M)"; //ANS:40+00M

type "$(10000, .0,)"; //ANS:10,000
```

Note:日付と時間では、*n* は 0 から始まります。

C 言語の形式

$() 置換の *format* の部分は、C 言語形式のステートメントもサポートします。

オプション	符号の有無	出力	入力 Range
d, i	符号付き	整数値(小数の整数部分または整数値)	-2^31 -- 2^31 -1
f, e, E, g, G	符号付き	十進数、科学表記、十進または科学表記	+/-1e290 -- +/-1e-290
o, u, x, X	符号なし	8 進数、整数、16 進数、16 進数(大文字)	-2^31 -- 2^32 - 1

Note:最後のカテゴリでは、負の値は 2 つの成分で表されます。

ここに LabTalk で使用する C コードのサンプルがいくつかあります。

```
double nn = -247.56;
type "Value:$(nn,%d)";    // ANS:-247

double nn = 1.23456e5;
type "Values:$(nn, %9.4f), $(nn, %9.4E), $(nn, %g)";
// ANS:123456.0000, 1.2346E+005, 123456

double nn = 1.23456e6;
type "Values:$(nn, %9.4f), $(nn, %9.4E), $(nn, %g)";
// ANS:123456.0000, 1.2346E+006, 1.23456e+006

double nn = 65551;
type "Values:$(nn, %o), $(nn, %u), $(nn, %X)";
// ANS:200017, 65551, 1000F
```

Origin と C 言語の形式の組合せ

Origin は、C 言語形式の指定子を使って、*E* と *S* の形式の使用をサポートしています。例えば、

```
xx = 1e6;
type "xx = $(xx, E%4.2f)";  // ANS:1.00M
```

負の値の表示

type コマンド(およびその他)のコマンド解析は、オプションスイッチ指示子として **-** 記号を探します。変数 K に負の値を割り当て、type コマンドを使ってその値を表す場合、引用符または括弧で置換表記を囲み **-** 記号を保護する必要があります。例えば、

```
K = -5;
type "$(K)"; // これは動作します
type ($(K)); // これも動作します
type $(K);   // これは type コマンドが-5 オプションを持たないのでエラーになる
```

変数の動的な名前付けと作成

置換表記$()は、代入演算子の右辺、左辺のいずれの側にあるかに関係なく、置換操作をし、値を与えます。

次のスクリプトで値 2 を持つ変数 A を作成します。

```
A = 2;
```

そして、この表記で値 3 を持つ変数 A2 を作成できます。

```
A$(A) = 3;
```

スクリプトウィンドウに **A$(A) =** または **A2 =** を入力して、それを確認します。

$()置換の他のサンプルについては、数値から文字列への変換をご覧下さい。

%n マクロとスクリプト引数

%*n* の形式の置換は、マクロ または スクリプトセクションに渡される引数で使われます。ここで *n* は整数 1-5 です(最大 5 つまでの引数をマクロまたはスクリプトに渡すことができます。)。

次のスクリプト例は、二つの引数(%1 と%2)を取り、その引数の和をダイアログボックスに出力するマクロを定義しています。

```
def add {type -b "(%1 + %2) = $(%1 + %2)"}
```

いったん、このマクロが定義されると、次のように入力するだけでそれを実行することができます。

```
add -13 27;
```

結果は次のように出力されます。

(-13 + 27) = 14

つまり、置換表記 $(%1+%2) は、14 という文字表現が結果となります。

5.2.3. プロットラベルで使用される構文と表記法

フォーマット構文と表記法を組み合わせた固定文字列は、**作図の詳細ダイアログのラベルタブのフォーマット指定編集ボックス**に入力して、プロットに目的のラベルを作成することができます。

フォーマット文字列と表記法は以下のいづれかを含めることができます。

目次

事前定義された文字

文字	説明
n, *i*	ソースワークシートの列および行番号
x, *y*, *z*	データポイントの X、Y、Z 座標

ix, iy	プロットが行列から作成されるときのデータポイントの列と行番号
zh	3D 三点グラフのデータポイントの Zh 座標

wcol と *col* 列の参照に使用されます。*wcol(n)* は数値式を引数として取り、*n* は現在の列番号を表す動的変数です。

数値と日時フォーマットの構文

LabTalk 置換表記ページを参照して数値と日時の値をフォーマットする構文を確認できます。

以下は頻繁に使用されるフォーマット文字列です。

フォーマット文字列	説明	サンプル
3	n 桁の有効桁数を表示, 桁数以下の 0 を消去	12.56 --> 12.6 13 --> 13
.4	n 桁の小数桁数を表示	12.56 --> 12.5600 13.46857 --> 13.4686
2.2	小数点以下 2 桁の 2 のべき乗で表示	13 --> 1.63×2^3
p*3	有効桁数 3 で科学的表記 10^3 で表示	52832 --> 5.28×10^4
S.4	有効桁数 4 で科学的表記 10^3 で表示	6215369 --> 6.2154E+06
E.3	小数点以下 3 桁の 2 のべき乗で表示	6215369 --> 6.215M
L.2	小数点以下 2 桁の 2 のべき乗で表示	13 --> $1.76 \times e^2$
##+##	10 と 100 の位の間セパレータ+を追加して表示	6251 --> 62+51
#+##M	10 と 100 の位の間セパレータ+を追加して表示し、接尾語 M をつける	625 --> 6+25M
#.0%	%を追加して数値をパーセンテージ表記する	0.2 --> 20.0% 0.253 --> 25.3%
# ##/##	2 桁までの分数で表記する整数の部分が混在している場合、残りの部分はスペースで区切られます。	1.25 --> 1 1/4 0.2 --> 1/5 0.3 --> 3/10
# #/8	8 の分数で表示されます	1.25 --> 1 2/8 0.2 --> 2/8 0.3 --> 2/8
DMS	degree、minute、second（度分秒）の形式で値を表示します	37.34255 --> 37°20'30

D1	**列のプロパティ**ダイアログボックスの**表示**ドロップダウンリストから 2 番目の形式で日付を表示	1/20/2018 --> Saturday, January 20, 2018 (in 英語 Windows7 OS)
T2	**列のプロパティ**ダイアログボックスの**表示**ドロップダウンリストから 3 番目の形式で時間を表示	10:05 --> 10:05:00

Origin の数値表記について

- Origin はプロットのラベルとワークシート列の両方で、カスタム表記をサポートします。**表示**: ***.n***, ***.n****, ****n***, ****n****.

備	A(X)	B(Y)	C(Y)	D(Y)	E(Y)
Comments		.3 (3 dec places)	.3* (3 dec places but truncate trail 0s)	*3 (3 sig digits)	*3* (3 sig but truncate trail 0s)
1	1.2	1.200	1.2	1.20	1.2
2	1.23	1.230	1.23	1.23	1.23
3	1.239	1.239	1.239	1.24	1.24
4	1.2395	1.240	1.24	1.24	1.24

- **#%**に小数点桁を加えるには、**#.00%**を使います（0 は一つの小数点桁）
- /100ths のように分数で表すには、**#/100** または**# #/100** を使います
- 過分数として表示するには**#/#**を使いますより正確にするには、**#** サインを複数使います(e.g. ###/###).
- **DMS** の基本構文は D [スペース] M [S] [F] [n]です。ここでカッコはオプションの要素を囲みます。S =秒、n =端数小数点を表す桁、F =ファイル。単位は表示されず、要素はスペースで区切られます。

  サンプル:

  DMS = 37°20'30

  D MS = 37° 20' 30

  DMSF = 37 20 30

  DMSF5 = 37 20 30.12345

  DMF1 = 27 20.5

- 時間が経つと、カスタムエントリのリストが乱雑になり過ぎることがありますリストからカスタムエントリを削除するには、ユーザファイルフォルダ（ヘルプ:フォルダを開く:ユーザファイルフォルダ）を参照し、テキストエディタを使用して Origin.ini ファイルを開き、[列カスタムフォーマットリスト]セクションを編集します。

列ラベル行の文字

例として、次のワークシートを使用します。

	A(X)	B(Y)
Long Name		Delta Temperature
Units		K
Comments		YBCO milled
Parameters		Version 2.1
Parameters 2		12/15/2004
Time		03:00:39 PM
UserDefined		S21
UserDefined1		235
RunNo		07
1	0.01	40
2	0.02	40
3	0.03	39.9
4	0.04	40.1

頻繁に使用される列ラベル行は以下のとおりです。

列のラベル行	文字	サンプル
ショートネーム	**G**	%(col(B)[G]$) --> B
ロングネーム	**L**	%(col(B)[L]$) --> Delta Temperature
単位	**U**	%(col(B)[U]$) --> K
コメント	**C**	%(col(B)[C]$) --> YBCO milled
パラメータ	**P***n*、ここで *n* はパラメータインデックス	%(col(B)[P1]$) --> Version 2.1 %(col(B)[P2]$) --> 12/15/2004
ユーザ定義パラメータ	**D***n*、ここで *n* はパラメータインデックス *または* **<Real Parameter Name>**、ユーザ定義パラメータの名前を変更した場合	%(col(B)[Time]$)/%(wcol(n)[D1]$) --> 03:00:39 PM %(col(B)[D2]$) --> S21 %(col(B)[D3]$) --> 235 %(col(B)[RumNo]$)/%(wcol(n)[D4]$) --> 07

このページを参照すると、より多くの列ラベル行にアクセスできます。

サンプル

フォーマット文字列	説明
$(wcol(n)[i],*4)	現在の Y 列の *i* 番目の行にある数値をラベルとして使用します。値は、有効桁数 4 桁で表示されます。
My Label: $(col(Pressure)[i],.3)	*Pressure* の *i* 番目の行の値に接頭辞"My Label:"をつけ、ラベルとして使用します。値は小数点桁数 3 で表示されます。

$(col(1)[i],D2)	現在の列の *i* 番目の行の日付をラベルとして使用します。**セルのフォーマット**ダイアログボックスの**フォーマット**を日付にしたときに**表示**ドロップダウンリストの 3 番目の形式で日付を表示します。
$(col(2)[i], T15)	2 番目の列の *i* 番目の行の時間の値をラベルとして使用します。**セルのフォーマット**ダイアログボックスの**フォーマット**を時間にしたときに**表示**ドロップダウンリストの 16 番目の形式で時間を表示します。
%(wcol(n+1)[i]$) %(x$)	*n+1* 番目の列の i 番目の行にあるテキストと X 値のテキストを組み合わせてラベルとして使用します。*n* は現在の Y 列のインデックスです。
%(book2,iy,ix)	行列からグラフを作成したときに、*ix* と *iy* は現在のデータポイントの元の行列ブックでの X、Y インデックスを参照します。このサンプルではワークブック、*book2* にラベルを配置できます。*%(book2,iy,ix)*は、Book2Sheet1 の *iy* 列 *ix* 行の値をデータポイントのラベルとして使用します。
%([book1]sheet2,iy,ix)	上述の*%(book2,iy,ix)* と同様ですこのサンプルではワークシートを指定する必要があります。
%(MBook2,ix,iy)	行列ブック *MBook2*（第一シート、第一オブジェクト）のセル(ix, iy)の値をラベルとして使用します。*ix* と *iy* は、元の行列のデータポイントの X と Y インデックスです。
$(Y)%(CRLF)$(p,0)%	積み上げグラフを作成するときは、この文字列を使用して Y 値と対応するパーセンタイルをラベルとして表示できます。"%(CRLF)"はラベルを 2 つの行に分けるために使用されます。パーセンタイルは整数で表示されます。
%(wcol(n)[L]$)	現在の Y 列のロングネームをラベルとして使用します。
$(Y*100, .1)%	小数点以下 1 桁で Y 値を 100%で表示します。

5.2.4. LabTalk オブジェクト

LabTalk スクリプトプログラミングでは、さまざまなオブジェクトおよびそのプロパティにアクセスできます。これらのオブジェクトは、ワークシート列やグラフのデータプロットなど GUI に表示されている Origin プロジェクトのコンポーネントを含みます。このようなオブジェクトは、**Origin オブジェクト**として参照され、次のセクション、Origin オブジェクトのサブジェクトとなっています。

オブジェクトのコレクションには、INI オブジェクトまたはシステムオブジェクトのようなインターフェースに表示されない別のオブジェクトも含みます。LabTalk スクリプトからアクセス可能なオブジェクトのすべてはアルファベット順のオブジェクトリストにあります。

通常、各オブジェクトにはプロパティがあり、そしてオブジェクトを操作するメソッドがあります。オブジェクトのプロパティとメソッドは、個々のオブジェクトにより異なります。例えば、データ列はグラフとは異なるプロパティを持ち、それぞれに実行する操作も異なります。どちらの場合でも、オブジェクトのプロパティにアクセスしたり、メソッドを呼び出すための一般的なシンタックスが必要です。これらは以下にまとめられています。

また、オブジェクトの名前を変更することができるので、異なるスコープのオブジェクトで名前を共有すると、オブジェクト名が紛らわしくなります。そのような理由から、各オブジェクトには固有のユニバーサル ID(UID)が割り当てられ、オブジェクト名とその UID 間の相互に参照する機能があります。

内容

プロパティ

プロパティは、次のようなシンタックスを持つオブジェクトに結びついた数値またはテキスト文字列を設定するか、返します。

objName.property (数値のプロパティ)
objName.property$ (テキストのプロパティ)

ここで *objName* はオブジェクトの名前で、*property* はオブジェクトの種類に対する有効なプロパティです。テキストオブジェクトにアクセスするとき、*プロパティ*の後ろに**$**記号を付ける必要があります。

例えば、次の方法でオブジェクトプロパティを設定できます。

```
// アクティブワークシートの列数を 10 に設定
wks.ncols = 10;
// アクティブワークシートを MySheet という名前にする
wks.name$ = MySheet;
```

またはプロパティの値を取得できます。

```
pn$ = page.name$;   // アクティブページ名を取得
layer.x.from = ;    // X 軸の開始値を取得して表示
```

メソッド

メソッドは、直ちに実行可能なコマンドです。実行されるとき、メソッドはオブジェクト関連する関数を実行し、値を返します。オブジェクトメソッドは、次の構文で表されます。

objName.method(arguments)

ここで *objName* は、オブジェクトの名前で、*method* は、オブジェクトの種類に対して有効なメソッド、*arguments* はメソッドがどのような動作をさせるかを決めます。一部の引数はオプションで、一部のメソッドは引数を必要としません。ただし、全てのオブジェクトメソッドのステートメントは括弧 "()" で囲む必要があります。

例えば、次のコードは、**run** オブジェクトの **section** メソッドを使って、**computeCircle**, という名前のスクリプトにある **Main** セクションを呼び出して、3 つの引数を渡します。

```
double RR = 4.5;
string PA$ = "Perimeter and Area";
run.section(computeCircle, Main, PA$ 3.14 R);
```

オブジェクト名とユニバーサル ID(UID)

各オブジェクトには、ショートネームとロングネームがあり、ほとんどのオブジェクトにはユニバーサル ID (UID)があります。ショートネームとロングネームの両方とも変更できますが、オブジェクトの UID はプロジェクト(OPJ ファイル)内では同じままです。オブジェクトの UID は、あるプロジェクトを別のプロジェクトに変更する場合に変わります。これは、すべてのオブジェクト UID がリフレッシュされ、新しく組み合わせられたプロジェクト内で各オブジェクトが固有になるように再設定されるためです。

多くの LabTalk 関数には引数としてオブジェクト名が必要で、オブジェクトの名前が変更されると、2 つの間で変換されるために、次の関数が提供されます。

- ***nVal* = range2uid(*rangeName*$)**
- ***str*$ = uid2name(*nVal*)$**
- ***str*$ = uid2range(*nVal*)$**

関連する関数は、次の一般シンタックスを持つ **NameOf(range$)** です。

- ***str*$ = nameof(*rangeName*$)**

その利用法は次のサンプルで示しています。

```
// (Book1, Sheet1 内) 列 1 の範囲変数を確立
range ra=[Book1]1!1;
// その範囲と結びついた内部名を取得
string na$ = NameOf(ra)$;
// na$ は 'Book1_A' になる
na$ =;
// 内部名を与える UID を取得
int nDataSetUID = range2uid(na$);
```

範囲名に加え、列、シート、ブックそれら自体の名前から UID を取得することができます。

```
// 列 2 の UID を返す
int nColUID = range2uid(col(2));
// シートまたはレイヤの UID を返す
int nLayerUID = range2uid([book2]Sheet3!);
// アクティブシートあるいはレイヤの UID を返す
nLayerUID =range2uid(!);
// アクティブワークブックの sheet3 の UID を返す
nLayerUID =range2uid(sheet3!);
//  特定シート内のインデックス 'jj' を持つ列の UID を返す
nColUID = range2uid([Book1]sheet2!wcol(jj));
```

また、関数 **range2uid** は、アクティブデータプロット名やデータ列名を含むシステム変数**%C** と一緒に動作します。

```
// アクティブデータプロットまたは選択した列の UID を返す
nDataSetUID = range2uid(%C);
```

範囲変数からページとレイヤを取得

範囲変数を与えると、対応するページとレイヤの UID を取得することができます。次のコードは、現在のシートの XY データから非表示のプロットを作成し、非表示のプロットのグラフページ名を取得する方法を示しています。

```
plotxy (1,2) ogl:=<new show:=0>; // A(x)B(y) を新しい非表示プロットとして作図
range aa=plotxy.ogl$;
int uid=aa.GetPage();
string str$=uid2Name(uid)$;
type "Result graph name is %(str$)";
```

プロットからブックとシートを取得

データプロットに関連したワークブックとワークシートを範囲変数として取得することもできます。次のコード(Origin 8 SR2 以降で利用可)は、アクティブプロット(%C)を列範囲として取得する方法を示し、そこから対応するワークシートとブック変数を取り出し、データをプロットする完全なアクセスを許可します。

```
// アクティブプロットの列範囲、 -w スイッチはデフォルトで Y 列を取得
range -w aa=%C;
// シートに対する wks 範囲、列は以下に属する
range ss = uid2range(aa.GetLayer())$;
// シート名を表示
ss.name$=;
// その列からのブック範囲文字列
range bb = uid2range(aa.GetPage())$;
// ブック名を表示
bb.name$=;
```

文字列形式で GetLayer と GetPage から返す範囲文字列を直接使うより簡単な方法もあります。

```
// アクティブプロットの列範囲、 -w スイッチはデフォルトで Y 列を取得
range -w aa=%C;
// シートに対する sheet 範囲文字列は以下に属する
range ss = aa.GetLayer()$;
// シート名を表示
ss.name$=;
// その列からのブック範囲文字列
range bb = aa.GetPage()$;
// ブック名を表示
bb.name$=;
```

ページにマッピングされた範囲を作成するとき、範囲変数は PAGE (オブジェクト)のプロパティがあります。
グラフレイヤにマッピングされた範囲を作成するとき、範囲変数は LAYER (オブジェクト)のプロパティを持ちます。
ワークブックレイヤ(ワークシートまたは行列シート)にマッピングされた範囲を作成するとき、範囲変数は WKS (オブジェクト)のプロパティを持ちます。

5.2.5. Origin オブジェクト

Origin でスクリプトを作成するのに欠くことのできない LabTalk のオブジェクトがあります。これを Origin オブジェクトと言います。これらのオブジェクトはグラフィカルインターフェースとして表示され、Origin プロジェクト(OPJ)に保存されます。Origin オブジェクトは、Origin プロジェクトの主要なコンポーネントです。これには以下のようなものがあります。

1. ページオブジェクト(ワークブック/グラフウィンドウ/行列ブック)

2. ワークシートオブジェクト
3. 列オブジェクト
4. レイヤオブジェクト
5. 行列オブジェクト
6. データセットオブジェクト
7. グラフィックオブジェクト

ルーズデータセットを除き、Origin オブジェクトは 3 つの階層で構成されています。

ワークブック -> ワークシート -> 列

行列ブック -> 行列シート -> 行列オブジェクト

グラフウィンドウ -> レイヤ -> データプロット

以下のセクションでは、オブジェクトメソッドを一覧にし、スクリプトでこれらのオブジェクトを使用例があります。

5.2.6. 文字列レジスタ

目次

イントロダクション

文字列レジスタは、Origin で文字データを扱う方法の 1 つです。V8.0 より前のバージョンでは、これが唯一の方法で、現在の Origin のバージョンでも使うことができます。しかし、文字列処理ルーチンを適切な文字列変数に移行することをお勧めします。使用の比較については文字列の処理をご覧下さい。

長年、文字列レジスタ名が%文字とそれに続く一つのアルファベット文字（A から Z までの文字）で構成されていることが、LabTalk スクリプト作成者に知られています。これらの基本となる 26 個の文字列レジスタ、すなわち%A-%Z のうち、いくつかはシステム文字列レジスタとして予約されています（以下の表にリストアップされています）。

Origin 2016 SR2 以降、必要に応じて "%@*N*"個の読み取り専用のシステム文字列レジスタが追加されています。これらは元の "%*N*"文字列レジスタの下の 2 番目の表にリストアップされています。

文字列レジスタはグローバルスコープであり、これは、いつでも、どのスクリプトでも変更できることを意味しています。これは役に立つこともありますが、あるスクリプトが別のスクリプトで使われている文字列レジスタの値を変更し、エラーや予期せぬ結果を招くので、使用には注意が必要です。

いくつかの文字レジスタはシステム変数として予約されており、自作のスクリプトでそれらに値を代入しようとするとエラーが発生する可能性があります。それらは、%C-%I、および%X-%Z の範囲でグループ化されるか、または前に%@が付きます（例：%@A = Apps ルートフォルダ）。詳しくは以下の表を確認してください。

システム変数としての文字列レジスタ

文字列レジスタは最大 260 バイト文字保持します（%Z は最大 6290 保持します）。文字列レジスタ名は、**%**文字（または**%@**）とそれに続く単一の英字（A から Z までの文字）で構成されます。このため文字列レジスタは、%変数としても知られています。

%文字列レジスタ

26 個の文字列レジスタのうち、次が特別な意味を持つシステム変数として予約され、これはスクリプト内で再割り当てをしてはいけません。しかし、これらが保持している値にアクセスしたり、操作する場合に、役立ちます。

%変数	説明
%C	現在のアクティブデータセットの名前
%D	現在の作業ディレクトリ, cd コマンドでセットされる(Origin8 で新しく追加)
%E	最近選択したワークシートを含むウィンドウの名前
%F	現在のフィッティングセッションにあるデータセットの名前
%G	現在のプロジェクト名
%H	現在のアクティブウィンドウのタイトル
%I	現在の基線のデータセット
%X	現在のプロジェクトのパス
%Y=	User Files フォルダへのフルパス名。ユーザの.INI ファイルやその他ユーザが編集したファイルがある場所。%Y は Origin を初回起動時に選択した場所によって、各ユーザごとに別々に設定することができる。 Origin 7.5 より前は、さまざまなユーザの INI ファイルは Origin.EXE がある場所にありました。Origin7.5 から、1 つの PC で複数ユーザが使えるように、User Files フォルダを作成することををサポートしています。 Origin の EXE のパス(プログラムパス)を取得するには、以下の LabTalk ステートメントを使います。

	`%a = system.path.program$` Origin C では、GetAppPath()関数へ適切な引数を渡します(INI パスまたは EXE パスを返すためです)。
%Z	長い文字列を一時的に保存する領域(最大 6290 バイト文字)。

システム変数を含む文字列レジスタは、名前が利用可能であれば、どこでも使用することができます。以下ご参照ください。

```
//現在のアクティブなワークシートを削除:

del %C;
```

%@文字列レジスタ

Origin 2016 SR2 以降では、**%@**文字とそれに続く一つの英字で構成される以下の読み取り専用のシステム文字列レジスタが追加されました。

%@変数	説明
%@A	アプリのルートインストールフォルダ。
%@B	OriginLab の AppData フォルダ。
%@C	コンピュータ名。
%@D	Origin 操作のデータと設定が保存されている ProgramData フォルダ。
%@E	グループリーダーのマシンのグループフォルダのパス詳しくは Set Group Folder Location Dialog を参照してください。
%@F	アクティブなプロジェクトエクスプローラのフォルダの名前。
%@G	アクティブなグラフシート(シートとしてワークブックに追加されたグラフ)のグラフページショートネーム。このタイプのシートでない場合、または含まれているグラフがない場合は空です。
%@H	グラフや行列が埋め込まれているアクティブなページを含むブックの名前。
%@I	最後にアクティブになったウィンドウのショートネーム。
%@L	OriginLab ソフトウェアライセンスの所有者名。
%@M	拡張子を含む、新しく追加されたファイルパス。
%@N	表示、非表示、削除にかかわらず、最も新しく作成されたウィンドウの名前。%H はこれとは異なり、アクティブウィンドウの名前を格納します。
%@O	Origin.EXE の EXE の名前を含めたフルパス。32-bit または 64-bit のどちらのバージョンでも動作します。もし両方のバージョンがインストールされている場合は、%@O は実行している EXE のパスを返します。

%@P	アクティブなフォルダのプロジェクトエクスプローラのフルパス。
%@Q	現在の Origin プロジェクトファイルの拡張子（小文字の opj または opju）。
%@R	Origin がパッチファイル(\Updates) やヘルプファイル(\Updates) をダウンロードして保存する ProgramData のルートフォルダ。メニューの**ヘルプ: フォルダを開く:プログラムデータフォルダ**で開くものと同じです。
%@S	インストール中の Origin セットアップダイアログボックスで入力されたユーザ名。
%@T	アクティブなグラフシート（シートとしてワークブックに追加されたグラフ）のタイトル（存在する場合はショートネーム + ロングネーム）。このタイプのシートでない場合、または含まれているグラフがない場合は空です。グラフにロングネーム無い場合は%@G と同じです。
%@U	プロジェクトの未保存ファイルのパス。
%@V	一時保存フォルダ。このフォルダはプロジェクトの保存中にファイルを保存するために使用されます。
%@W	インストール中の Origin セットアップダイアログボックスで入力された会社名。
%@X	現在実行中の Apps フォルダ。
%@Y	ユーザの AppData ルートフォルダ。このフォルダには、インストールされているアプリとそのバージョン番号をリストした xml ファイルが格納されています。

文字列変数としての文字列レジスタ

システム変数を除く文字列レジスタは、文字列レジスタを文字列変数として使用できます。以下ご参照ください。

文字列変数への値の代入

例えば、スクリプトウィンドウに次の代入文を入力して下さい。

```
%A = John
```

上記は、文字列レジスタ**%A** の内容を **John** にします。

文字列変数は置換表記で使うこともできます。置換表記を使って、次の代入文をスクリプトウィンドウに入力します。

```
%B = %A F Smith
```

これは**%B** の内容を **John F Smith** にします。つまり、代入演算子の*右側*で文字列変数が評価されると、その結果が代入演算子の*左側の識別子*に割り当てられます。

数値変数のように使うことができます。スクリプトウィンドウに次の代入文を入力して下さい。

```
%B =
```

Origin は変数の値を返します。

John F Smith

割り当て前の変数の評価

小括弧()で囲むと、代入演算子の左辺の文字列変数は、代入される前にいったん評価されます。例えば、次の代入文をスクリプトウィンドウに入力します。

```
%B = Book1_A
```

このスクリプトにより、値 **Book1_A** が、文字列レジスタ%B に割り当てられます。**Book1_A** がデータセット名だとして、スクリプトウィンドウに次の代入文を入力します。次の例を見てください。

```
(%B) = 2*%B
```

これにより、データセットの結果として、**2** が掛けられた値に更新されます。しかし、文字列レジスタ**%B** の内容は、文字列 **Book1_A** のままです。

文字列の比較

文字列レジスタの比較をするには、等価演算子(==)による比較を使用します。

- 文字列レジスタが引用符で囲まれていると(例えば"%a")、Origin は変数名を構成する文字列を文字通りに比較します。例えば、

```
aaa = 4;
bbb = 4;
%A = aaa;
%B = bbb;
if ("%A" == "%B")
    type "YES";
else
    type "NO";
```

この例の結果は **NO** となります。なぜならば、この比較式の場合、文字列変数の文字を比較するので、**aaa !=bbb** となります。

- 文字列レジスタが引用符で囲まれていないと(例えば%a)、Origin は文字列レジスタに含まれる変数の値を比較します。例えば、

```
aaa = 4;
bbb = 4;
%A = aaa;
%B = bbb;
if (%A == %B)
    type "YES";
else
    type "NO"
```

この例の結果は **YES** となります。なぜならば、この比較式は、得られた(数値)変数の内容を比較しているので、**aaa==bbb==4** となるからです。

部分文字列

部分文字列 (substring)表記は、その文字列ある特定の部分とまります。部分文字列の一般的な形式は、下のような構文をしています。

%[*string*,*argument*];

ここで、***string*** は、文字列そのものであり、***argument*** は、どの部分の文字列を返すのかを指定します。
例を挙げてみましょう。次の代入文をスクリプトウィンドウに入力しましょう。

```
%A="Results from Data2_Test"
```

以下に、この%A の部分文字列における argument の指定の仕方を説明します。

以下を行うには	入力するスクリプト	返される値
文字を検索し、その文字の左側にある全てのテキストを返す。	`%B = %[%A, &apos;_&apos;]; %B =`	Results from Data2
文字を検索し、その文字の右側にある全てのテキストを返す。	`%B = %[%A, >&apos;_&apos;]; %B =`	Test
指定された文字位置の左側にある全ての文字を返す。	`%B = %[%A, 8]; %B =`	Results
指定された 2 つの文字の間（指定された位置を含む）にある全ての文字を返す。	`%B = %[%A, 14:18]; %B =`	Data2
左から数えて *#n* トークンを返す。	`%B = %[%A, #2]; %B =`	from
文字列の長さ(文字数)を返す。	`ii = %[%A]; ii =`	ii = 23

以下は、部分文字列の別の例です。

以下を行うには	入力するスクリプト	返される値

指定された区切り記号（この例の場合は、タブ(\t)）で区切られた、i 番目のトークンを返す。	`%A = 123 342 456;`
	`for (ii = 1; ii <= 3; ii++)`
	`{`
	`  Book1_A[ii] = %[%A, #ii,\t]`
	`};`
		Book1_a[1]に **123** の値を、**Book1_a[2]**に **342** の値を、**Book1_a[3]**に **456** の値を入れます。

@n 行目を返す。	`%Z = "First line`
	`second line";`
	`%A = %[%Z, @2];`
		文字列**%Z** の 2 行目を文字列**%A** に代入これを確認するには、**%A=**をスクリプトウィンドウに入力します。

注意:

引用符の部分文字列や置換表記への使用する場合、

- スペースは 1 つの文字とみなされ、無視されません。
- 文字列の長さ(バイト数)は、スペースも 1 文字として数えます。

例えば、**%A** に **5** をセットして、**%A** の長さを見るには、スクリプトウィンドウに次のように入力して、*Enter* を押します。

```
%A = " 5 ";
ii = %[%A];
ii = ;
```

Origin は **ii = 3** のように返します。

トークンについての注意

トークンとは、ホワイトスペース(スペース又はタブ)、丸括弧(...)、カギ括弧[...]、引用符(")で囲まれた文字群のことです。例えば、次のトークン群について考えてみます。

```
%A = These (are all) "different tokens"
```

スクリプトウィンドウに以下のようにスクリプトを入力すると、

スクリプト	返される値
`%B = %[%A, #1]; %B=`	These
`%B = %[%A, #2]; %B=`	are all
`%B = %[%A, #3]; %B=`	different tokens

5.2.7. X ファンクションの紹介

X ファンクションは、バージョン 8 で導入された新しい機能で、Origin のツールを構築するフレームワークを提供します。ほとんどの X ファンクションは、オブジェクト操作やデータ分析のような操作を実行するために LabTalk スクリプトからアクセスできます。

スクリプトから X ファンクションを実行する一般的なシンタックスは、以下のような形です。角括弧 [ ] はオプションです。

xfname [-options] arg1:=value arg2:=value ... argN:=value;

X ファンクションを実行するとき、Origin はコロンと等号 の記号 ":="を使って、引数の値を割り当てます。例えば、単純な線形フィットを実行するには、fitlr X ファンクション が使われます。

```
// フィットするデータ、アクティブワークシートの Col(A) と Col(B) が
//  :=記号を使って、変数 'iy'に割り当てらる
fitlr iy:=(col(a), col(b));
```

ほとんどの X ファンクションが任意の引数を持ちますが、Origin がデフォルトの値と設定を持つ場合には、引数なしで X ファンクションを呼ぶこともできます。例えば、新しいワークブックを作成するために、newbook X ファンクションを呼びます。

```
newbook;
```

X ファンクションは実行するのが簡単で使いやすいので、以下のスクリプトサンプル内でそれらの多くを使用しています。X ファンクションを実行するためのオプション (ヘルプの取得、ダイアログを開く、自動更新出力の作成を含む)と引数の詳細は、X ファンクションと Origin C 関数の呼び出し セクションで説明しています。

5.3. LabTalk スプリクトの優先順位

マクロや OriginC 関数、X ファンクション、OGS ファイルなど、いくつかのオブジェクトがあるため、これらのオブジェクト間で同じ名前を使用しないように注意する必要があります。同じ名前のオブジェクトがあると混乱を招き、正しくない結果が導き出されることもあります。どうしても同じ名前でないといけない場合、LabTalk は予め定められたルールに従い、オブジェクトを実行していきます。次のリストは LabTalk のオブジェクトで、上から順に優先順位が高くなっています。

1. マクロ
2. OGS ファイル
3. X ファンクション
4. run.file(*FileName*)のような LT オブジェクトメソッド
5. Crigin C 関数を呼ぶことができる LT
6. LT コマンド(省略可)

6 X ファンクションおよび Origin C 関数の呼び出し

LabTalk から X ファンクションおよび Origin C 関数を呼び出す方法を説明します。

- X ファンクション
- Origin C 関数

6.1. X ファンクション

X ファンクションは、LabTalk スクリプトからタスクを実行したり、Origin の機能にアクセスする主要なツールです。次のセクションでは、LabTalk で X ファンクションを利用する助けとなる概要を説明します。

この章で以下の項目を説明します

- X ファンクションの概要
- X ファンクションの入力と出力
- X ファンクションの実行オプション
- X ファンクションの例外の扱い

6.1.1. X ファンクションの概要

X ファンクションは、LabTalk スクリプトから Origin の機能のほとんどすべてを同じ方法で利用することができます。X ファンクションを使う最も良い方法は、X ファンクションを使う多くのサンプルを利用することです。スクリプトからアクセスできる X ファンクションの一覧は LabTalk でサポートしている X ファンクションのセクションにあります。

内容

シンタックス

X ファンクションの固有のシンタックスからスクリプトサンプルで X ファンクションを認識できます。

xFunctionName input:=<*range*> argument:=<*name/value*> output:=<*range*> -switch;

一般的な注意:

- X ファンクションは、複数の入力、出力、引数を指定することができます。
- X ファンクションは、その利用可能な引数のリストのサブセットで呼び出すことができます。
- 値が指定されていない場合、デフォルト値が使われます。
- 各 X ファンクションは、異なる入出力の変数を持ちます。

X ファンクションの引数の順番についての注意:

- X ファンクションはデフォルトで、その入力と出力の引数が特定の順序で入力されます。
- 引数の順序は、個々の X ファンクションのヘルプファイルにあり、スクリプトウィンドウで、**XFunctionName -h** のように表示します。
- Origin が指定している順番で引数を入力すれば、引数名を入力する必要はありません。
- 引数名を明示的に入力すると、引数はどの順番でも指定することができます。
- Origin に決められた順序であれば、最初のいくつかの引数名を省略可能です。その後に続く引数については、引数名を記述すればどのような順番でも指定可能です。
- 引数名は、引数名の後ろにあるいくつかの文字をトリミングして短縮可能ですが、短縮名は固有である必要があります。

次のサンプルは、**fitpoly** X ファンクションを使って、これらについて説明しています。

サンプル

fitpoly X ファンクションは、次の特定のシンタックスを持ち、これは Origin が指定している引数の順番です。

fitpoly iy:=(inputX,inputY) polyorder:=n coef:=columnNumber oy:=(outputX,outputY) N:=numberOfPoints;

指定された順番で入力する場合、X ファンクションは次のように書きます。

```
// 適切な順序で、「切片を固定」と「切片の値を指定」を
// 0 に指定する必要がある
fitpoly (1,2) 4 0 0 3 (4,5) 100;
```

これは、アクティブワークシートの列 1 と列 2 にある XY データの 100 ポイントで 4 次多項式のフィットを行い、列 3 に多項式の係数を配置し、列 4 と列 5 にフィット結果の XY データを配置します。

これに対して、すべてのオプションを入力したコマンドは、長くなりますが、上記と同じ操作を実行します。

```
fitpoly iy:=(1,2) polyorder:=4 coef:=3 oy:=(4,5) N:=100;
```

入力と出力の引数名を入力するのと引き替えに、LabTalk はどの順番でも受け付けられ、期待する結果を返します。

```
fitpoly coef:=3 N:=100 polyorder:=4 oy:=(4,5) iy:=(1,2);
```

他の方法として、いくつかの引数名を省略し、その後に続く引数は、順不同に名称と共に指定します。これにより、以下は上記と同じ結果を取得するスクリプトです。

```
fitpoly (1,2) 4 oy:=(4,5) N:=100 coef:=3;
```

引数リストの中で固有の短縮名である場合、以下のように引数を短縮可能です。

```
fitpoly iy:=(1,2) poly:=4 co:=3 o:=(4,5) N:=100;
```

poly は **polyorder** 、**co** は **coef**、**o** は **oy** の短縮です。以下のスクリプトを使用すると、エラーが返されます。

```
fitpoly i:=(1,2) poly:=4 co:=3 o:=(4,5) N:=100;
```

これは、i から始まる引数が 2 つ(**iy** と **intercept**)含まれていることが原因です。ここで、Origin は どちらの引数 i が標準であるか、つまり、i が一意ではないということはできません。

また、明示的に指定すれば、入力と出力はそれぞれ別の行でどの順番にでもすることができます。

```
fitpoly
coef:=3
N:=100
polyorder:=4
oy:=(4,5)
iy:=(1,2);
```

X ファンクションの呼び出しを終了するセミコロンは、X ファンクションと結びついた最後のパラメータの後にだけ配置します。

オプションスイッチ

-h や **-d** のようなオプションスイッチは、スクリプトから X ファンクションを実行する代替のモードにアクセスします。それらは、他の引数と一緒または引数無しで使うことができます。オプションスイッチ(とその値)は引数リストのどこにでも配置することができます。このテーブルは、主要な X ファンクションのオプションスイッチをまとめたものです。

名前	関数
-h	スクリプトウィンドウにヘルプファイルの内容を出力
-d	ダイアログボックスを表示してパラメータを入力
-s	サイレントモードで実行。結果は結果ログに送られない
-t <*themeName*>	事前にセットしたテーマを使用
-r <*value*>	入力が変更したら、自動再計算するよう出力をセット

オプションスイッチについての詳細は、X ファンクションの実行オプションをご覧下さい。

ダイアログ設定からスクリプトを生成

Xファンクションを呼ぶ最も簡単な方法は、*-d* オプションを使い、ユーザインターフェース(GUI)を使ってその設定を行うことです。

GUI で、ダイアログ設定が行われると、ダイアログテーマのフライアウトメニューの**スクリプトを生成** を選んで、設定に対応する LabTalk スクリプトを生成することができます。そして、現在の GUI 設定に合致するスクリプトがスクリプトウィンドウに出力され、それをバッチ用の OGS ファイル、または他のプロジェクトにコピー＆ペーストすることができます。

6.1.2. X ファンクションの入力と出力

内容

X ファンクションの変数

X ファンクションは LabTalk の変数型(StringArray を除く)を引数として受け付けます。LabTalk 変数だけでなく、X ファンクションは複雑なデータ構造の特別な変数型を使うこともできます。

これらの特別な変数型は、下表にあるように X ファンクションへの引数としてのみ動作します。(使用可能なキーワードについての詳細は、この章にある**範囲の特殊なキーワード**セクションをご覧ください。)

変数型	説明	サンプル構造	コメント
XYRange	X と Y の組合せ。任意で Y エラーバーデータを含む。	1. (1,2) 2. <new> 3. (1,2:end) 4. (<入力>, <新規>) 5. [book2]sheet3!<new>	グラフに対して、インデックスを直接使って、プロット範囲(1,2)がグラフ上の 1 番目と 2 番目のプロットであることを示します。
XYZRange	X, Y, Z データの組合せ。	1. (1,2,3) 2. <new> 3. [book2]sheet3!(1,<new>,<new>)	4.
ReportTree	階層レポートのツリーベースオブジェクトはワークシート範囲または LabTalk ツリー変数に結びついている必要があります。	1. <new> 2. [<入力>]<新規> 3. [book2]sheet3	4.

変数型	説明	サンプル構造	コメント
ReportData	ベクターのコレクションのツリーベースオブジェクトは ワークシート範囲または LabTalk ツリー変数に結びついている必要があります。ReportTree とは違って、ReportData は通常のワークシートに出力し、ワークシート内の既存のデータの終わりに追加するのに使うことができます。ReportData オブジェクト内のすべての列はグループ化されている必要があります。	1. <new> 2. [<入力>]<新規> 3. [book2]sheet3 4. [<input>]<input>!<new>	5.

変数型についてより理解を深めるには、この章の **ReportData** の出力セクションのサンプルを参照してください。

範囲の特殊なキーワード

<new>

新しいオブジェクトを追加/作成

<active>

アクティブオブジェクトを使う

<input>

同じ X ファンクションの入力範囲と同じ

<same>

X ファンクションの前の変数と同じ

<optional>

オブジェクトが入力または出力で任意であることを示す

<none>

オブジェクトは作成されない

ReportData の出力

多くの X ファンクションが **ReportData** オブジェクトの形式で、複数の出力ベクターを生成します。通常、ReportData オブジェクトは、**NLFit** X ファンクションからの **Fit Curves** 出力のように、ワークシートと結びついています。例えば、**fft1** X ファンクションからの出力を考えてみましょう。

```
// ReportData 出力を Book2, Sheet3 に送る
fft1 rd:=[book2]sheet3!;
// ReportData 出力を Book2 の新しいシートに送る
fft1 rd:=[book2]<new>!;
// ReportData 出力をアクティブワークブック/シートの列 4 に送る
fft1 rd:=[<active>]<active>!Col(4);
// ReportData 出力をアクティブワークブックの新しいシートに送る
fft1 rd:=[<active>]<new>!;
// ReportData 出力を tr1 というツリー変数に送る
```

```
// 'tr1' が存在しなければ、作成されます。
fft1 rd:=tr1;
```

レポートデータをツリー変数に送る

ReportData 出力を中間変数として、このようなデータを一時的に保持するワークシートのオーバーヘッドを含みたくないかもしれません。

そして、1 つの代替手段は、複数ベクターを 1 つにまとめ、このようなベクターデータの追加の属性をサポートするツリー変数を使って、レポートデータオブジェクトを作り上げるデータセットを保存することです。

ワークシートの出力範囲の仕様は、通常、**[Book]Sheet!**, **<new>** あるいは **<active>** のような形式となっています。出力文字列がこれらの通常のブックーシートの仕様の 1 つを持たない場合、出力は自動的に LabTalk ツリー名であると考えられます。

以下は、**avecurves** X ファンクションのサンプルの機能です。このサンプルでは、結果の ReportData オブジェクトがツリー変数への最初の出力であり、そして、そのツリーからの 1 つのベクターは入力データがある同じシートの特定の列位置に置かれます。ReportData は通常新しいシートに出力されます。

```
int nn = 10;
col(1)=data(1,20);  //いくつかのデータを入力
loop(i,3,nn){wcol(i)=normal(20);};
range ay=col(2); //'avecurves'の Y 出力
Tree tr; // 出力ツリー
avecurves (1,3:end) rd:=tr;
// ツリーノード(ベクター)'aveY'を範囲'ay'に割り当て
//  'tr.='を使ってツリー構造を表示
ay=tr.Result.aveY;
ay[L]$="Ave Y"; // ロングネームをセット

//  デフォルト X を使って散布図としてデータをプロット
plotxy (?,3:end) p:=201;
// 範囲'ay'のデータを折れ線で同じグラフに追加
plotxy ay o:=<active> p:=200;
```

レポートデータを直接ブック/シート/列の特定の位置に送る

X ファンクションから結果を新しい列として入力シートに配置するだけなら、次のように行います。

```
avecurves (1,2:5) rd:=[<input>]<input>!<new>;
```

レポートデータを出力する入力シートの特定の列を指定する場合、次のように指定します。

```
avecurves (1,2:5) rd:=[<input>]<input>!Col(3);
```

これらの新しい列を見つけるための追加のコードを記述する必要があるので、この後に続くデータアクセスは複雑になります。

ReportData タイプの出力が、使われている特定の X ファンクションに依存して異なるデータの量(列)を含むことが分かります。結果を既存のシートに送る場合、生成される ReportData 列で既存データを上書きしないよう注意が必要です。

6.1.3. X ファンクションの実行オプション

内容

X ファンクションのオプションスイッチ

次のオプションスイッチは、スクリプトから X ファンクションにアクセスするときに利用できます。

スイッチ	名前	関数
-cf	--	入力範囲の列フォーマットをコピーし、それを出力範囲に適用
-d	-dialog	X ファンクションパラメータを選択するダイアログを開く
-db	--	ダイアログの種類; 現在のワークブック内でパネルとして X ファンクションダイアログを開く
-dc *IsCancel*	--	ダイアログの種類; X ファンクションパラメータを選択するダイアログを開く。**OK** ボタンをクリックする場合、*IsCancel* を 0 にセットし、**キャンセル**ボタンをクリックする場合は 1 をセットする。**キャンセル**ボタンをクリックする場合、**#User Abort!** のようなエラーメッセージはスクリプトウィンドウに表示されず、そのまま X ファンクションが実行される
-e	--	X ファンクションビルダで X ファンクションを開く。edit -x コマンドと同じように動作する。
-h	-help	スクリプトウィンドウにヘルプファイルの内容を出力
-hn	--	X ファンクションをロードしコンパイルするが、それだけしかしない。すでにロードとコンパイル済みの場合、何もしない
-hs	--	-h の変形; スクリプト使用例のみを出力
-ht	--	-h の変形; 存在していれば、ツリーノード情報のみを出力
-hv	--	-h の変形; 変数リストのみ出力
-hx	--	-h の変形; 関連する X ファンクション情報のみ出力
-r 1	-recalculate 1	入力が変更したら、自動再計算するよう出力をセット

-r 2	-recalculate 2	手動で再計算が必要な場合に再計算するよう出力をセット
-s	-silent	サイレントモードで実行。結果は結果ログに送られません。
-sb	--	-s の変形; エラーメッセージと結果ログの出力を出しません。
-se	--	-s の変形; エラーメッセージを出しません。結果ログの出力は行います。
-sl	-silent	-s と同じ
-ss	--	-s の変形; スクリプトウィンドウに情報メッセージを出しません。
-t <*Name*>	-theme	事前に設定されたテーマを使用します。

Recalculate is not supported when <input> is used an an <output>.

既存の**完全な名前**を持つオプションでは、省略形のスイッチまたは完全な名前のどちらかをスクリプトで使います。例えば、X ファンクション **smooth** では

```
smooth -h
```

これは、次のコマンドと同じです。

```
smooth -help
```

サンプル

テーマを使う

FivePtAdjAve という名前のテーマを使って、アクティブワークシートの列 1 および列 2 にある XY データにスムージング操作を実行します。

```
smooth (1,2) -t FivePtAdjAve
```

Note:Origin は自動的にテーマを保存し、取り出すので、テーマファイルへのパスを指定する必要はありません。あるプロジェクト(*.OPJ)で保存したテーマは他のプロジェクトで同様に使うことができます。

再計算モードの設定

入力列のデータが変わったときに、**freqcounts** X ファンクションの出力列を自動再計算にセットします。

```
freqcounts irng:=col(1) min:=0 max:=50 stepby:=increment inc:=5
   end:=0 count:=1 center:=1 cumulcount:=0 rd:=col(4) -r 1;
// '-r 1' で再計算を自動にセット
```

X ファンクションダイアロを開く

スクリプトから X ファンクションを実行している間、入力を促すためにダイアログを開きたい場合があります。この例では、パーセンタイルフィルタ(method:=2)を使って、25 個のデータポイントでの移動ウィンドウで、スムージング操作を実行します。さらに、smooth X ファンクションに結びついているダイアログ(-d)を開き、入力および出力データの選択に対して、他のオプションを使えるようにします。

```
smooth method:=2 npts:=25 -d
```

入力から出力にフォーマットをコピー

-cf オプションスイッチを持つ FFT フィルタを使って、出力データをフォーマットし、入力データのフォーマットと一致させます。

```
// *.wav ファイルをインポート。インポートされた*.wav データフォーマットは、short(2)
fname$ = system.path.program$ + "Samples\Signal Processing\sample.wav";
newbook s:=0; newsheet col:=1; impWav options.SparkLines:=0;
string bkn$=%H;

// デフォルトで、すべての分析結果はデータ型 double として出力
// ここで-cf は、出力データを short(2)にする
fft_filters -cf [bkn$]1!col(1) cutoff:=2000
oy:=(<input>,<new name:="Lowpass Sound Frequency">);
```

6.1.4. X ファンクションの例外の扱い

以下のサンプルは LabTalk を使って X ファンクションのエラーを捕まえる例です。これにより、エラーを引き起こす可能性のある X ファンクションを呼び出しによってスクリプト全体を中断させません。

エラーコードを返さ**ない** X ファンクションに対して、最後に実行した X ファンクションのエラーをチェックする次の 2 つの関数があります。**xf_get_last_error_code()** と **xf_get_last_error_message()$**これらの関数は、特定の X ファンクションがエラーとなる可能性がある場所で使います。

この例では、ユーザにインポートするファイルを選択するオプションが与えられます。しかし、そのインポートがエラーとなる場合(例、ユーザがインポートに適さないファイルタイプを選択する) 、残りのコードを取り扱う必要があります。

```
dlgfile gr:=*.txt;  // ユーザからファイル名とパスを取得
impasc -se;  // -se スイッチを使用して実行を継続する必要がある、下記 note 参照
if( 0 != xf_get_last_error_code() )
{
    strError$ = "XFunction Failed:" + xf_get_last_error_message()$;
    type strError$;
    break 1;  // 実行停止
}
// データインポートは恐らく成功し、スクリプトの実行を継続可能
type continuing...;
```

通常の X ファンクションのオプション **-se** を使って、エラーメッセージを出さないようにします。**-sl** オプションを使うと、エラーログを出力しません。**-sb** オプションは両方を出力しません。スクリプトの実行を継続して次の行へ移動するには、これら 2 つの内のどちらかのオプションを使用する必要があります。

ループしてピークを見つける

次のサンプルでは、ワークシートのすべての列をループしてピークを見つけます。特定の列にピークが見つからない場合、スクリプトは残りの列に対して継続されます。ここでは適切なデータを持つワークシートがアクティブであると見なしています。

```
for(int ii=2; ii<=wks.ncols; ii++)
{
  // 現在の列のにピークが見つけ、XF からエラーメッセージを出力しない
  Dataset mypeaks;
  pkfind $(ii) ocenter:=mypeaks -se; // 継続するには-se を使用する必要がある


  // XF がエラーになったかどうかをチェック
  if( 0 != xf_get_last_error_code() )
```

```
  {
    type "Failed on column $(ii):%(xf_get_last_error_message()$)";
  }
  else
  {
    type Found $(mypeaks.getsize()) peaks in column $(ii);
  }
}
```

6.2. Origin C 関数

次のセクションでは、LabTalk スクリプトから Origin C 関数 を呼び出す方法を説明しています。

この章で以下の項目を説明します

- Origin C 関数のロードとコンパイル
- 変数を Origin C 関数に渡す/受け取る
- 既存の Origin C ファイルを更新する
- Origin C 関数を使用する

6.2.1. Origin C 関数のロードとコンパイル

内容

OriginC 関数またはワークスペースのロードとコンパイル

Origin から Origin C 関数を呼び出す前に、現在の Origin セッションで関数をコンパイルリンクしておく必要があります。ソースファイルをプログラムでコンパイルリンクしたり、LabTalk からワークスペースをビルドするには、LabTalk の run オブジェクトの run.loadOC メソッドを使います。

```
err = run.LoadOC("myFile",[option]);
```

サンプル

ロードされる Origin C ファイルを走査するオプションを使って、Origin C ファイルの依存ファイルも自動的にロードされます。

```
// オプションを 16 とし、iw_filter.c の Origin C 関数を
// ロードし、コンパイルする
// これにより、iw_filter.c 内の.h ファイルをスキャンする
```

```
if(run.LoadOC(OriginLab\iw_filter.c, 16) != 0)
{
        type "Failed to load iw_filter.c!";
        return 0;
}
```

メニューの**表示**:**コードビルダ**を選択して**コードビルダ**を開きます。そして、**ワークスペース**パネル(表示されていない場合、**コードビルダ**のメニュー**表示**:**ワークスペース**で表示できます)で、**Temporary** フォルダにある *iw_filter.c* ファイルを確認できます。

Origin C ソースファイルをシステムフォルダに追加する

コードビルダでファイルを開いたら、コードビルダワークスペースの **System** ブランチにファイルをドラッグアンドドロップします。そうすることで新しい Origin セッションでもファイルをロードしコンパイルすることが保証されます。詳細は、コードビルダ の文書を参照して下さい。

プログラムでソースファイルを system フォルダに追加して、Origin が実行しているときにいつでも利用できるようにすることができます。

```
run.addOC(C:\Program Files\Originlab\Source Code\MyFunctions.c);
```

これは、ユーザに ツールを配布したり、値の設定ダイアログで動作するように作られた関数を永久に利用できるようにするときに役に立ちます。

Origin C ファイルをプロジェクト(OPJ)に追加する

Origin C ファイル(またはどんなファイルでも) を Origin プロジェクトファイルに追加することができます。そして、ファイルは OPJ と一緒に保存され、プロジェクトが開いたときに抽出することができます。Origin C ファイルの場合、ファイルをコンパイル リンクすることもでき、ファイルにある関数にアクセスすることができます。プロジェクトにファイルを追加するには、単にファイルをコードビルダの **Project** ブランチにドラッグアンドドロップするか、ブランチを右クリックして、ファイルを追加します。詳細は、コードビルダ の文書を参照して下さい。

6.2.2. 変数を Origin C 関数に渡す/受け取る

どのような関数を呼び出す場合でも、その関数に変数を渡したり、関数が出力する変数を受け取る必要がある場合がしばしば起こります。以下は、LabTalk 変数を Origin C 関数に渡すシンタックスおよび特徴をまとめたものです。

LabTalk から Origin C 関数を呼び出すシンタックス

Origin C 関数は次のようなシンタックスを使って LabTalk から呼び出します。

```
// 複数のパラメータがある場合、カンマ(,)でパラメータを分ける
int iret = myfunc(par1, par2....);

// 代入がなければ、括弧やカンマは不要
myfunc par1;

// 次の関数は、値を返さず、パラメータもない。括弧は任意
myfunc;
```

LabTalk から渡すまたは受け取る変数のデータ型

次の表は、Origin C 関数を呼び出すときの LabTalk から渡されるまたは受け取る Origin C 変数型の一覧です。

変数型	OC 関数への引数	OC からの戻り
double	可	可
int	可	可
bool (真または偽)	不可。代わりに int を渡す。偽の場合 0、真の場合他の整数。	不可。代わりに int を返す。偽の場合 0、真の場合 1。
string	可	可
int 型、double 型の配列	可	可
String Array	可。参照では渡せない	可

Note:

1. Origin C 関数を LabTalk をから呼び出すために持つことができる引数の最大数は 80 です。
2. LabTalk の変数は参照によって Origin C の数値関数に渡すことができます。

6.2.3. 既存の Origin C ファイルを更新する

イントロダクション

グループリーダーや開発者は他の Origin ユーザに Origin C ファイルの新しいバージョンを配布したい場合があります。このような場合、エンドユーザが既に古いバージョンの Origin C ファイルをインストールしていると、そのファイルに対応する OCB ファイルがユーザファイルフォルダ(UFF)にあります。新しい Origin C ファイルのタイムスタンプが、その OCB ファイルのタイムスタンプより古いという可能性があります。この場合、Origin は OCB ファイルが既に更新されていると考え、新しい Origin C ファイルを再コンパイルしません。このようなことを避けるために、新しい Origin C ファイルをインストールするときに、OCB ファイルを削除することが最も良い方法です。削除されると、Origin は必ず OCB ファイルを再作成するので、そのために新しい Origin C ファイルをコンパイルします。

OCB ファイルを手動で削除する

Origin C ファイルに対応する OCB ファイルをエンドユーザのコンピュータのユーザファイルフォルダ内の OCTEMP フォルダから手動で削除することができます。Origin C ファイルの場所にもよりますが、OCB ファイルを OCTemp フォルダ内のサブフォルダに置くことも可能です。配置されたら、エンドユーザは OCB ファイルを削除でき、ワークスペースをリビルドし、OCB ファイルを更新します。

プログラムyで OCB ファイルを削除する

グループリーダーや開発者は、LabTalk の Delete コマンドの OCB オプションを使って、プログラム的に対応する OCB ファイルを削除することができます。これは Origin パッケージ内の Origin C ファイルを配布する場合にとても役立ちます。つまりエンドユーザが手動で削除できないような OCB ファイルを削除できます。

以下は、LabTalk の Delete コマンドの OCB オプションを呼び出す方法のサンプルです。

```
del -ocb filepathname1.c
del -ocb filepathname1.ocw
del -ocb filepathname1.c filepathname2.c // 複数ファイルを削除
del -ocb %YOCTEMP\filename.c // %Y を使用してユーザファイルフォルダのパスを取得
```

6.2.4. Origin C 関数を使用する

機能を拡張するため、1 つの値を返す Origin C 関数を定義して（詳細は Orign C の使用と作成を参照）、コマンドウィンドウからその関数を呼び出すことができます。例えば、

1. メニューの**表示**:**コードビルダ**を選択して**コードビルダ**を開きます。
2. **コードビルダ**メニューの**ファイル**:**新規**を選択して新しい*.c ファイルを作成します。**新規ファイル**ダイアログで、*MyFuncs* などのファイル名を入力し、OK ボタンをクリックします。
3. このファイルの最後で新しい行を開始し、以下のコードを入力します。

```
double MyFunc (double x)
{
    return sin(x) + cos(x);
}
```

4. メニューの**ビルド**:**ビルド** を選択してファイルのコンパイルとリンクを行います。
5. エラーがなければ、上記で定義した関数を LabTalk で使用できます。コマンドウィンドウに次のスクリプトを入力して実行します。

```
newbook;  // 新しいワークブックを作成
col(A) = data(1, 32);  // 行番号で埋める
col(B) = MyFunc(col(A));  // OriginC 関数を呼び、B 列に結果を出力
```

7 LabTalk スクリプトの実行とデバッグ

この章で以下の項目を説明します

- スクリプトの実行
- スクリプトのデバッグ

Origin は、LabTalk スクリプトを実行したり、保存するためのいくつかオプションがあります。この章の最初には、これらのオプションの説明があります。この章の 2 番目に Origin でサポートしているスクリプトデバッグ機能の概要があります。

7.1. スクリプトの実行

7.1.1. スクリプトの実行

次のセクションには LabTalk スクリプトを実行したり、保存する 11 個の方法をまとめています。しかし、最初にスクリプトとスクリプトが動作するオブジェクトとの関係に注意することが重要です。

このセクションで以下の項目を説明します

- スクリプトウィンドウやコマンドウィンドウから
- ファイルから
- 値の設定ダイアログから
- ワークシートスクリプトから
- スクリプトパネルから
- 図形オブジェクトから
- ProjectEvents スクリプト
- インポートウィザードから
- 非線形フィルタから
- 外部アプリケーションから
- コンソールから
- タイマー操作
- Origin 起動中の処理

- カスタムメニューアイテムから
- ツールバーボタンから

デフォルト

ワークブックやグラフページなどの Origin オブジェクトを操作するとき、スクリプトは常にデフォルトでアクティブウィンドウに対して行われます。ウィンドウがアクティブでない場合、win -a を使ってアクティブにします。

```
win -a book2;                  // book2 のウィンドウをアクティブにする
col(b) = {1:10};        // book2 の B 列に 1 から 10 を入力
```

しかし、*win -a* を使ってアクティブウィンドウを操作するのは、確実でない場合があります。スクリプトの実行シーケンスで、アクティブウィンドウまたはレイヤを切り替えることは、動作が遅くなり、正確でない結果を導く可能性があります。

スクリプトを括弧で囲み、常に win -o *winName* {*script*} を使うことが好ましい方法です。こうすると、Origin は指定したウィンドウを一時的にアクティブウィンドウ(内部的)としてセットし、そのウィンドウ以外のウィンドウを排除して、括弧内のスクリプトを実行します。例えば、次のサンプルコードは、新しいプロジェクトを作成し、デフォルトのブックにいくつかのデータを入力して、プロットします。その後ブックに戻り、新しいシートを追加して、2 番目のシートで 2 つ目のグラフを作成します。。

```
doc -s;doc -n;//デフォルトワークシートを含む新しいプロジェクト
string bk$=%H;//ブックショートネームを保存
//データを入力してショート新しいプロットを作成
wks.ncols=2;col(1)=data(1,10);col(2)=normal(10);
plotxy (1,2) o:=<new>;
//ここで新たに作成したグラフがアクティブなウィンドウになる
//元のワークブックでいくつかスクリプトを実行
win -o bk$ {
  newsheet xy:="XYY";
  col(1)=data(0,1,0.1);col(2)=col(1)*2;col(3)=col(1)*3;
  plotxy (1,2:3) plot:=200 o:=<new>;
}
```

win -o だけが文字列変数を使用することができる LabTalk コマンドです。上記のように、記述する必要はありません。

```
win -o %(bk$)
```

この特定のコマンドは頻繁に使われ、Origin8.0 で文字列変数を利用できるように、修正されました。文字列変数が LabTalk コマンドへの引数として使われる場合、すべての場所で、%( )置換表記を使う必要があります。

LabTalk スクリプトを実行する場所

Origin にはスクリプトを保存したり、実行する場所がいくつもあり、それらは同じではありません。次のサブセクションでは、よく使用される順番に配置しています。

最初の 2 つ、(1) スクリプトウィンドウおよびコマンドウィンドウからスクリプトを実行する (2) ファイルからスクリプトを実行する, は、スクリプトを記述する方には、他の場所よりとてもよく使われます。まず、この章の 2 つのサブセクションを読むことをお勧めします。その他については、必要に応じて読んでも構いません。

7.1.2. スクリプトウィンドウやコマンドウィンドウから

LabTalk を直接実行する 2 つのウィンドウがあります。(古い) スクリプトウィンドウ と (新しい) コマンドウィンドウです。各ウィンドウは、1 行以上のスクリプトを実行できます。コマンドウィンドウにはプロンプトがあり、プロンプトに入力したすべてのコードを実行します。

スクリプトウィンドウには、カーソルだけがあり、Enter キーを押すと、マウスで選択したコードまたはカーソルが存在する行のコードを実行します。どちらのウィンドウも Ctrl+Enter キーで実行無しで改行します。Ctrl+Enter キーを使って改行するとき、ステートメントの最後にセミコロン;を含める必要があります。また、編集メニューの「スクリプトの実行」をオフにすることもできます。

コマンドウィンドウには、X ファンクションのオートコンプリート機能や実行したコマンドのコマンド履歴や再呼び出しの機能がありますが、スクリプトウィンドウにはありません。スクリプトウィンドウは、複数行や長いスクリプトを編集するのが簡単です。

以下は、1 つの X 列と複数の Y 列の形式でデータを持つワークシートを処理するサンプルスクリプトです。コードは、すべての Y データから Y 値の最大値および最小値を見つけ、すべての Y をその範囲で正規化するものです。

```
// 最も小さい最小値と最も大きい最大値を探す
double absMin = 1E300;
double absMax = -1E300;
loop(ii,2,wks.ncols)
{
    stats $(ii);
    if(absMin > stats.min) absMin = stats.min;
    if(absMax < stats.max) absMax = stats.max;
}
// 各列をその範囲に正規化
loop(ii,2,wks.ncols)
{
    stats $(ii);
    wcol(ii)-=stats.min;                    // ゼロの最小値にシフト
    wcol(ii)/=(stats.max - stats.min);      // 1 に正規化
    wcol(ii)*=(absMax - absMin);            // 範囲に正規化
    wcol(ii)+=absMin;                       // 最小値にシフト
}
```

スクリプトウィンドウで実行するには、コードを貼り付け、マウスですべてのコードを選択し、Enter キーを押します。(選択したテキストは反転表示されます)

コマンドウィンドウでスクプトを実行するには、コードを貼り付け Enter キーを押します。コードに誤りがある場合、スクリプトウィンドウではすべてを利用して編集することができますが、コマンドウィンドウの履歴パネルでは編集できず、スクリプトすべてを再呼び出しすることもできません。

Origin には、コードビルダというスクリプトエディタがあり、これは LabTalk や Origin C コードを編集したり、デバッグすることができます。コードビルダを開くには、スクリプトまたはコマンドウィンドウに **ed.open()** と入力するか、標準ツールバーから  ボタンを選択します。

7.1.3. ファイルから

LabTalk スクリプトは通常、Origin オブジェクトを要求し、開いたプロジェクトに制限されます。スクリプトはファイルに保存され、どのプロジェクトからでも呼び出すことができます。スクリプトファイルは 5 つまでの引数を伴って呼び出すことができます。このセクションは、ファイルに保存した LabTalk スクリプトの使用の概要です。

内容

スクリプトファイルを作成/保存する

LabTalk スクリプトは、Origin の コードビルダを含むテキストエディタで作成し、保存できます。コードビルダを開くには、標準ツールバーの  ボタンをクリックします。エディタウィンドウにコードを入力したり、貼り付けて新しいドキュメントを作成し、フォルダおよびファイル名を付けて保存します(OGS という拡張子になります)。

OGS ファイルの拡張子

LabTalk スクリプトは、ファイルに保存でき、どんな拡張子でも構いませんが、OGS という拡張子が標準のもので、柔軟性が高くなります。ですので、**OGS ファイル**を使うことをお勧めします。ファイルシステムのアクセス可能なフォルダにスクリプトファイルを保存できますが、特定の位置にすると利点があります。OGS ファイルがユーザファイルフォルダにある場合、スクリプトを実行する際に、パスを指定する必要はありません。

OGS ファイルは、ファイルとしてディスクに保存するのではなく、Origin プロジェクト(OPJ)に接続することができます。ファイルはコードビルダの **Project** ノードに追加することができ、プロジェクトと一緒に保存できます。保存の際には、ユーザフォルダからファイルをドラッグし、プロジェクトフォルダにドロップします。このように接続された OGS ファイル内のスクリプトのセクションをファイル内のセクションを呼び出すのと同様、**run.section()** オブジェクトメソッド を使って呼び出すことができます。Origin は最初にプロジェクト内のファイルを探すので、ファイル名だけを指定する必要があります。プロジェクトに接続したファイルとセクションが見つかったらコードを実行します。

OGS ファイル内のセクション

コードがモジュール方式で書かれていると、スクリプトの実行は分かり易く、デバッグもしやすくなります。モジュールスクリプトをサポートするため、LabTalk スクリプトファイルをセクションに分割します。セクションは各括弧[ ]の中にセクション名を付けて宣言します。

[*SectionName*]

セクション宣言の下のスクリプト行がそのセクションに属します。セクション内の LabTalk スクリプトは、別のセクション宣言が見つかったときや、return ステートメントが実行されたとき、コマンドエラーが発生したときに終了します。通常、複数セクションを持つ OGS ファイルのフレームワークは次のようなものです。

```
[Main]
// スクリプト行
ty In section Main;

[Section 1]
// スクリプト行
ty In section 1;

[Section 2]
// スクリプト行
ty In section 2;
```

ここで、**ty** は **type** コマンドのことで、文字'ty'で始まるコマンドが他に存在しない場合に、このように省略することができます。

OGS ファイルを実行する

OGS ファイルを実行する

run オブジェクトを使ってスクリプトファイルを実行するか、ある環境下では、LabTalk がファイル名を**コマンド**オブジェクトとして解釈します。ファイルをコマンドオブジェクトとして使用する場合、ファイル拡張子は OGS である必要があります。詳細については、以下の **Note** を参照してください。

次の呼び出しフォーマットを比較しましょう。

run.section(OGSFileName, SectionName[,arg1 arg2 ... arg5])

run.file(OGSFileName[arg1 arg2 ... arg5])

OGSFileName.SectionName [arg1 arg2 ... arg5]

OGSFileName [arg1 arg2 ... arg5]

特に、**test.ogs** ファイルを呼び出すファイルをユーザファイルフォルダに保存している場合

```
// コマンドシンタックスを使って、test.ogs の [Main] セクションを実行、
// それ以外はファイルの先頭のセクションではないコードを実行、それ以外は何もしない
test;

// コマンドシンタックスを使って、test.ogs の section1 のみ実行
test.section1;

// run.section() シンタックスを使って、test.ogs の section1 のみ実行
run.section(test, section1)
```

Note:OGS ファイルを保存した後、**cd**X ファンクション(`cd 1;`)を実行し、ファイルが保存されるフォルダを変更する必要があります。あるいは、現在の作業フォルダにファイルがある場合、**dir** を使用する事もできます。これを実行しないと、Origin はファイルを探すことができず、実行可能なコマンドを見つける事ができません。

Origin が、OGS ファイル名をオブジェクトとして認識した後、OGS ファイル名か name.sectionname をスクリプトウィンドウかコマンドウィンドウで実行します。ファイル名またはセクション名に空白(スペース)を含む場合、ダブルクオテーションで囲む必要があります。例えば、

```
// フォルダにある 'My Script.ogs' という LabTalk スクリプトを実行
//'D:\OgsFiles'.

// 現在の作業フォルダを 'D:\OgsFiles'に変更
cd D:\OgsFiles; // Origin はフォルダ内に OGS ファイルを検索
//  'My Scripts.ogs'内の'Beta Test' セクションのコードを実行
// スペースで区切られた 3 つの引数を受け渡す（必要に応じてダブルクオテーションを使用）
"My Scripts.Beta Test" "Redundant Test" 5 "Output Averages";
```

次のコマンドを実行してアクセスできる Origin の **Samples\LabTalk Script Examples** フォルダには多くのサンプルがあります。

```
cd 2;
```

スクリプトに引数を渡す

run.section() オブジェクトメソッドを使って、スクリプトファイル(またはそのセクションの 1 つ)またはマクロを呼び出すとき、引数を渡すことができます。引数は、テキスト、数値、数値変数、文字列変数のいずれかです。

引数をスクリプトファイルセクションやマクロに引き渡す際

- セクションやマクロの呼び出しで、引数と引数の間はスペースで区切る必要があります。run.section を使う場合、カンマは、最初の引数からセクション名を区切ります。
- 引数がテキスト、あるいは文字列変数の場合、個々の引数は(内容が 1 語以上、又は負の数の場合)二重引用符で囲まれている必要があります。数値や数値変数を引き渡す場合は、値が負の数の時を除いて、2 重引用符で囲む必要はありません。
- スクリプトファイルセクションやマクロに引き渡せる引数の数は、**スペース**で区切った 5 つまでです。引き渡された引数は、スクリプトファイルセクション、または、マクロの定義内の引数の受け渡し変数に置かれます。引数の受け渡し変数には%1,%2,%3,%4,%5 があります。最初に引き渡された引数は%1 に、2 番目の引数は%2 に、といった要領で、割当てが行なわれます。受け渡し変数は文字列変数と同じような働きをします。つまり、これらの変数は、埋め込まれているコマンドの実行前に置換されます。引き渡された引数の数は *macro.narg* に登録されます。

文字列を%1, %2, などが受け取る引数として渡すサンプルです。TEST.OGS ファイルには次のセクションが含まれます。

```
[output]
type "%1 %2 %3";
```

次のスクリプトを実行したとします。

```
run.section(test.ogs, output, "Hello World" from LabTalk);
```

ここで、%1 には"Hello World"、%2 には"from"、%3 には"LabTalk"が、引き渡たされます。文字列の置換後、type コマンドによってスクリプトウィンドウに出力されます。

```
Hello World from LabTalk
```

をスクリプトウィンドウに出力します。もし、スクリプトファイルセクション呼び出し内で、2 重引用符が省略されると、%1 に"Hello",%2 に"World",%3 に"from"が受け渡されてしまい、Hello World from となり、Origin は次のように出力します。

```
Hello World from
```

数値変数の参照による引き渡し

数値変数引数の参照による引き渡しは、スクリプトファイルセクションやマクロの中から、元の変数の値の変更を可能にします。

例えば、あるアプリケーションで **LastRow** という変数が、値を含む列 B 内の最後の行番号を保持しているものとします。更に、現在の **LastRow** の値が 10 だとします。列 B に 5 つの値を(現在の最後の行に)追加するようなスクリプトファイルセクションに、変数 **LastRow** を引き渡したとします。値の追加後、スクリプトファイルセクションによって、変数 **LastRow** の値を増やして、**LastRow** の値を 15 に更新することができます。

サンプル:

つまり、TEST.OGS が次のような adddata セクションを含んでいるとします。

```
[adddata]
        loop (n, 1, 5)
        {
            %1[%2 + n] = 100;
        };
        %2 = %2 + (n - 1);
        return 0;
```

次に以下のスクリプトを実行するとします。

```
col(b) = data(1, 10);    // データ 1_b を値で埋める
get col(b) -e lastrow;   // lastrow に最後の行の値を入れる
run.section(test.ogs, adddata, col(b) lastrow);
lastrow = ;
```

変数 **LastRow** は参照によって引き渡され、値に 15 を持つように更新されます。

数値変数の値による引き渡し

数値変数引数の値による引き渡しは、置換表記$()を使って行われます。この表記は、引数をスクリプトファイルセクション、又はマクロに引き渡す前に、インタープリタに引数の評価を強制します。この引き渡しのテクニックは、計算の結果値を変更せずそのまま後で使うような場合に有効です。それに引き換え、引数が参照によって引き渡されると、式が解釈される度に、式全体が計算されます。

次のスクリプトファイルのサンプルでは、数値変数 *var* が参照および値で渡されます。%1 は、参照で渡される引数を保持し、%2 は値で渡される引数を保持します。更に、2 語からなる文字列変数(%A)も 1 つの引数として%3 に引き渡されています。

```
[typing]
        type -b "The value of %1 = %2 %3";
        return 0;
```

セクションを *Test.OGS* に保存し、コマンドウィンドウで次のスクリプトを実行します。

```
var = 22;
%A = "degrees Celsius";
run.section(test.ogs, typing, var $(var) "%A");
```

すると、ダイアログボックスが開き、次のメッセージを出します。"The value of var = 22 degrees Celsius".

OGS ファイルとセクションの名前のガイドライン

OGS スクリプトファイルの命名規則は、その呼び出し方によって異なります。上記のセクションでは、2 つの方法を説明しています。**run.section()** メソッドを使った呼び出し、スクリプトまたはコマンドウィンドウからの呼び出し (**コマンド** による方法)

Run.section() メソッドを使うとき

- OGS ファイルの名前に使用する文字の種類や長さには制限がありません。
- ファイル名の拡張子の指定は、OGS という拡張子を持つファイルに対して任意です。
- OGS ファイル内で run.section( )を使って同じ OGS ファイルの別のセクションを呼び出すとき、ファイル名を省略できます。例えば

```
[main]
run.section( , calculate);

[calculate]
cc = aa + bb;
```

コマンドによる方法を使うとき

- OGS ファイルの名前はコマンド名の制限に従います。25 文字以下、先頭の文字が数値または特殊文字は不可、スペースやアンダースコアを含まない
- ファイル名の拡張子は OGS にしなければならず、指定する必要はありません。

セクション名の規則(どちらかの方法を使うとき)

- セクション名が省略されているとき

  1. Origin は **main** というセクションを探し、それが見つかると実行します。
  2. **main** セクションが見つからなく、セクションの名前のないファイルの最初にコードがある場合は、そのコードを実行します。
  3. 何もない場合には、Origin は何も実行せずエラーも出ません。

OGS ファイルは、既存の Origin の関数 や X ファンクションと同じ名前にすることはできません。

パスを設定する

Origin7.5 では、スクリプトファイル(.OGS)は Origin システムとユーザファイルフォルダの両方から実行できました。これがデフォルトで現在の作業フォルダにあたります。スクリプトファイルがそこにあれば、パスを変更する必要はありません。しかし、スクリプトファイルが、これら 2 つのフォルダのどちらにも存在しない場合、run.section()オブジェクトメソッド フルパスで指定する必要がありました。Origin 8 以降、現在の作業フォルダ(CWF)という考えが導入され、選択した CWF の場所で、スクリプトファイルや X ファンクションを実行することができます。

MS-DOS での慣習にならい、Origin でも cd X ファンクションを使って CWF を表示することができます。

```
// このコマンドを入力し、スクリプトウィンドウに
//現在の作業フォルダを表示
cd
```

デフォルトから変更されていない限り、出力は以下のようなディレクトリです。

```
現在の作業ディレクトリ:
C:\Documents and Settings\User\My Documents\OriginLab\Origin8.1\User Files\
```

しかし、多くのスクリプトを書いていると、ファイルをフォルダ内に統合し、そのファイルが存在する場所からスクリプトを呼び出します。また、Origin はそれぞれのディレクトリから実行できるサンプルスクリプトを提供しています。

run.section( )スクリプトがユーザファイルフォルダのサブフォルダにある場合、相対リファレンスを使う事ができます。例えば、

```
run.section(subfolder1\scriptA,main); // ScriptA.ogs は subfolder1 にある
run.section(subfolder2\scriptA,main); // ScriptA.ogs は subfolder2 にある
```

スクリプトを使用して、現在の作業フォルダをセットできます。例えば、Origin のシステムサブフォルダ **Samples\LabTalk Script Examples** にある **ave_curves.ogs** という OGS ファイルを実行するために、次のスクリプトを実行します。

```
// 文字列変数を作成し、目的のスクリプトファイルに
//Origin のシステムパスにフォルダのパスを追加して
// 完全なパスを保持する
path$ = system.path.program$ + "Samples\LabTalk Script Examples\";
// 目的のパスを現在の作業フォルダに設定
cd path$;
// 関数を呼び出す
ave_curves;
```

事前定義のパスのセットを作成できます。cdset X ファンクションは、事前定義したパスを全てリストし、CWF を追加/削除することができます。次のように入力します。

```
//  'cdset' コマンドはスクリプトウィンドウに
//事前定義のパスを表示
cdset
```

変更していないならば、以下のようなパスが表示されます。

```
1 = C:\Documents and Settings\User\My Documents\OriginLab\Origin8.1\User Files\
2 = C:\Program Files\OriginLab\Origin81\Samples\LabTalk Script Examples\
3 = C:\Program Files\OriginLab\Origin81\
```

上記の 2 番目のパスを CWF にセットするには、次のように入力します。

```
// CWF を事前定義パス 2 に変更
cd 2
```

事前定義してあるフォルダセットに、新しいパスを追加するには、新しいパスを変更し、次に **cdset** X ファンクションを特定のインデックスでセットします。例えば、

```
cd D:\Files\Filetype\Script;  // このパスを CWF としてセット
// このパスを事前定義リスト 4 番目の位置（index 4）に追加
// 既に index 4 がある場合、上書き
cdset 4;
// CWF が手動で変更された場合、次のように再セット可能
// 'cd 4'と入力する事で以下のように再セット可能  'D:\Files\Filetype\Script\'
```

cdset コマンドの操作については、知っておくと便利なポイントがいくつかあります。

- あるプロジェクトで事前定義パスに新しいパスを追加すると、それは保存され、別のプロジェクトでも使用できます。
- 現在のパスをスクリプトウィンドウに表示するには、スクリプトウィンドウで 'cdset' と入力します。
- 最大 9 つまでのパスを定義することができます。
- インデックス番号を自分で割り当てることはできません。
- 現在のパスが存在しているインデックスに新しいパスを割り当てると、現在のパスが上書きされます。

上記で記載された 3 つの事前定義パスで、2 つ目は複数のスクリプトのファイルを有します（拡張子 **OGS**）。DOS と同じように、*cd 2* でこのフォルダに移動できます。有効な OGS の確認には dir X ファンクションを使う事ができます。そして使用可能なスクリプトを実行します。

```
// 2 番目のフォルダを CWF としてセット
cd 2;
// CWF 内の全ての ogs と X ファンクションをリスト
dir;
// スクリプトファイルの実行
// 呼び出す時にファイルの拡張子は必要ない
autofit;
```

また、ed.open()メソッドを使用すると、CWF のスクリプトファイルをコードビルダにロードできます。例えば、

```
// この場合、ファイル名の OGS 拡張子が必要です
ed.open(pick_bad_data.ogs)
```

Origin C から LabTalk を実行する

.OGS ファイルを直接実行するだけでなく、LabTalk コマンドおよびスクリプトは Origin C から実行することもできます。詳細については、Origin C ヘルプファイルの LabTalk インターフェース のグローバル関数をご覧ください。

7.1.4. 値の設定ダイアログから

値の設定ダイアログは、他のデータセットへの参照を含む関数でデータの列の計算を行うときに役立ちます。

値の設定によって割り当てられた列は、入力した計算式の結果が入力されます(数式はデータセットを返す)。数式は、自動的に更新(自動)、ユーザの要望に応じて実行(手動)、または、更新しない(なし)にすることができます。

1 つの計算式では行えない複雑な計算に対しては、ダイアログ内の実行前の処理スクリプトパネルに、LabTalk スクリプトを含めて使用することができます。

自動と手動の設定では、それぞれ および のような錠前のアイコンが、列の上に作成されます。緑色の錠前アイコンは、データが更新されている事を表し、黄色い錠前 は、更新する必要があることを示します。また、赤い錠前アイコンは、参照先が削除されていたり、計算式が壊れている事を示します。

コードが自己参照する場合(例えば、セットされる列が計算式に含まるなど)、自動と手動のオプションは「なし」にセットされます。

以下は、値の設定ダイアログに対するスクリプトサンプルです。通常、短いスクリプトがこのダイアログに入力されます。

他の列を使った表現

以下のように数式(計算式の右側)のみの場合:

```
// 列 3 で
// 列の値を大きくまたは小さくする
// 大きな値や小さな値に問題があるような場所でフィットする場合に役立ちます。
col(2)*1e6;
```

いくつかの場合、条件を使用した数式便利です。

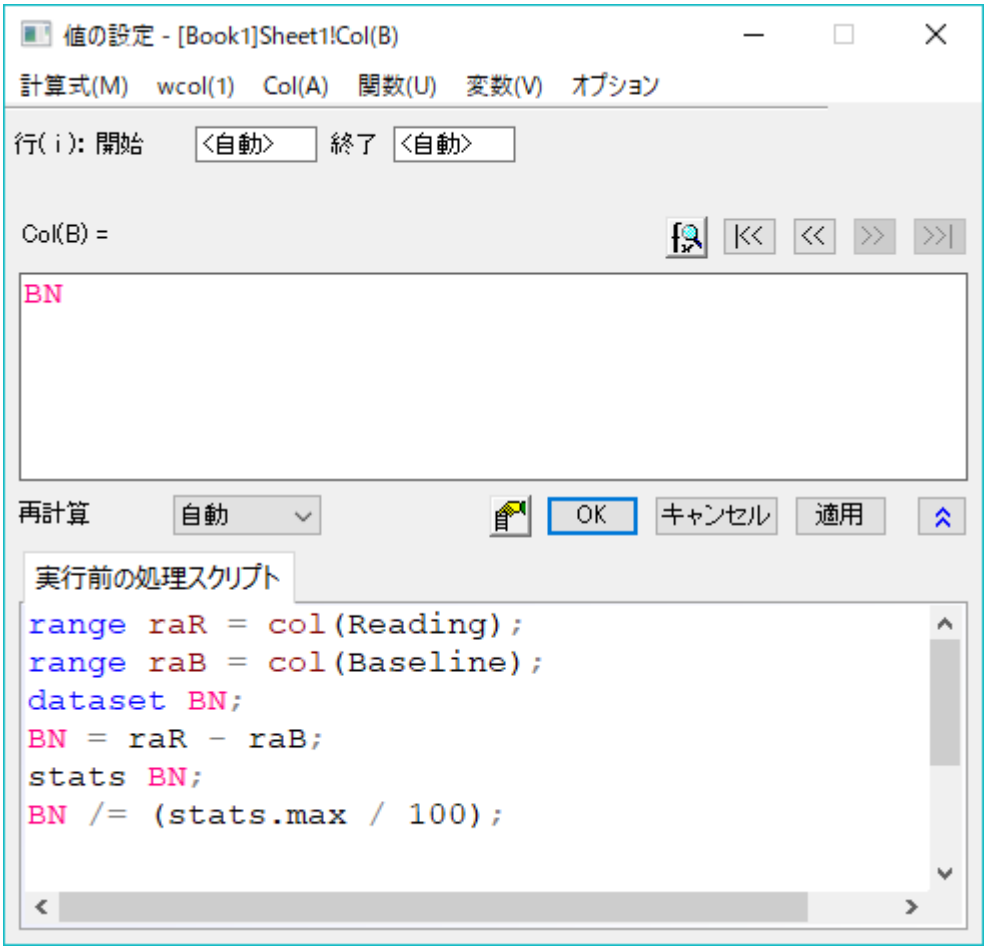
```
// 負の値をゼロに設定する
col(2)<0?0:col(2);
```

実行前の処理スクリプトセクションを使う

値の設定ダイアログの**実行前の処理スクリプト**セクションでは、その数式が実行される前に実行するスクリプトを入力します。この機能は、数式を適切にセットアップする操作を実行するのに役立ちます。次のサンプルは、そのようなスクリプトの使用例を示します。

```
// 数式セクションで...
// BaseNormal 列
BN
// 実行前の処理スクリプトセクション内で...
range raR = col(Reading);    // 信号
range raB = col(Baseline);   // ベースライン
dataset BN;
BN = raR - raB;              // 信号からベースラインを減算
stats BN;                    // 結果の統計量を出力
BN /= (stats.max / 100);     // 最大値を 100 に正規化
```

次の画像は、値の設定ダイアログに入力した上記コードのスクリーンショットです。



7.1.5. ワークシートスクリプトから

値の設定ダイアログの自動更新機能やインポート後に実行するスクリプトを設定するインポートフィルタが無かった古いバージョンの Origin との下位互換性のために提供されています。

スクリプトは、ワークシートに保存でき(そのためワークブックには各シートに対して別々のワークシートスクリプトを持つ)、このワークシートにインポートするか、特定のデータセット(このワークシート内でなくても)に変更があったときに実行するようセットできます。

以下は、Sheet3 へのインポートで実行されるか、Sheet2 の A 列が変更したときに実行される Sheet3 に接続したスクリプトです。

```
range ra1 = Sheet1!1;
range ra2 = Sheet1!2;
range ra3 = Sheet2!A; //  'Change in Range' 列
range ra4 = 3!2;      // インポートでシート名を変更
range ra5 = 3!3;      // 数値のシートで参照
ra5 = ra3 * ra2 / ra1 * ra4;
```

7.1.6. スクリプトパネルから

(ワークブックタイトルバーのコンテキストメニューからアクセス)は、スクリプトウィンドウとコマンドウィンドウの両方の機能があります。

- スクリプトウィンドウのように、複数行を保持でき、コード行を選択して、Enter キーで実行します。
- また、コマンドウィンドウのように、実行済みの履歴が残ります。
- スクリプトウィンドウとは異なり、Origin を閉じたときに、スクリプトの内容は保存されませんが、これらのスクリプトはプロジェクトに保存されます。

```
// 列 2 を 10 でスケール
col(2)*=10;

// 'mV' 列の最小値を 0 にシフト
stats col(mV);
col(mV)-=stats.min;

// 列 3 を 1 に正規化した列 2 にセット
stats 2;
col(3) = col(2)/stats.max;
```

7.1.7. 図形オブジェクトから

(テキスト、線、図形)をイベントと結びつけ、そのイベントで実行されるスクリプトを含めることができます。図形オブジェクトはページと結びつけられるので、テンプレート、ウィンドウファイル、プロジェクトファイルに保存されます。

ボタン

頻繁に使われるスクリプトのいくつかを Origin のユーザインターフェース(GUI)のボタンに割り当てて、スクリプトの実行を自動化することもできます。これには、次のステップを行います。

新しく開いた Origin プロジェクトから

1. プロジェクトウィンドウの左にあるツールメニューからテキストツールを選択します。--> 
2. **Book1** ウィンドウの空白ワークシートの 2 つの空の列の右にある領域をクリックします。テキストボックスが開きます。

   テキストボックスに"Hello"と入力し、Enter を押します。これでボタン用のラベルが入力できました。
3. 次に、ALT キーを押しながら、今作成したテキストをダブルクリックします。**オブジェクトのプログラム制御**というウィンドウが開きます。
4. ダイアログの左上にある「オブジェクト名」の項目を、**Greeting** に変更します。
5. **オブジェクトのプログラム制御** ウィンドウの下側のテキストボックスで、スクリプトをそのまま入力します。

   ```
   type -b "Hello World";
   ```

6. また、**オブジェクトのプログラム制御**ウィンドウで、**～のあとでスクリプトを実行**ボックスで、**ボタンアップ**を選び、**OK** をクリックします。

7. これでボタンが作成できました。ボタンを押すと、スクリプトが実行され、ポップアップウィンドウに "Hello World" と出力されます。

スクリプトウィンドウにのみ存在しているテキストスクリプトとは異なり、このボタンとスクリプトは、Origin プロジェクトを保存するときに保存されます。

また、グラフィックオブジェクトに設定されたスクリプトは、オブジェクト名と run メソッドを使用して実行できます。**コマンドウィンドウ**を開き、以下のように入力します。

```
greeting.run()
```

そして ENTER キーを押して実行します。

線

以下は、直線を移動すると、その直線がある X 位置にあるデータの補間値を出力するようにグラフ上に垂直線を作成するスクリプトです。

```
//グラフ上に垂直線を作成
draw -n MyCursor -l -v $(x1+(x2-x1)/2);
MyCursor.HMOVE = 1;                 // 水平移動可能
MyCursor.color = color(orange);     // オレンジ色に変更
MyCursor.linewidth = 3;             // 線を太くする
// グラフにラベルを追加
label -sl -a $(MyCursor.x) $(Y2+0.05*(Y2-Y1)) -n MyLabel $(%C(MyCursor.x));
// 線にスクリプトを割り当て
MyCursor.script$="MyLabel.x = MyCursor.x;
MyLabel.y = Y2 + MyLabel.dy;
doc -uw;";
// 線が動いた後スクリプトを実行するようにする
MyCursor.Script = 2;
```

その他のオブジェクト

グラフィカルオブジェクト(テキスト、線、図形)に、イベントが発生したときに実行するスクリプトを添付することができます。

この例では、グラフ上の四角形(オブジェクト名 RECT)が移動またはサイズ変更した時にスクリプトが実行されるように設定されています。

1. **プロット操作・オブジェクト作成**ツールバーの四角形ツールを使って、グラフに矩形を描きます。
2. **オブジェクト編集**ツールバーのプロット後部に移すボタンを使って、データの後ろに矩形を配置します。
3. Alt キーを押しながら、矩形をダブルクリックして、**オブジェクトのプログラム制御**を開きます。オブジェクト名が Rect であることを確認します。
4. 次のスクリプトを入力します。

```
%B = %C;
%A = xof(%B);
dataset dsRect;
dsRect = ((%A >= rect.x1) && (%A <= rect.x2) &&
          (%B >= rect.y3) && (%B <= rect.y1))?%B:0/0;
stats dsRect;
delete dsRect;
type -a Mean of $(stats.mean);
```

5. ～のあとでスクリプトを実行ドロップダウンリストで**移動／サイズ変更**を選びます。
6. OK をクリックします。

この矩形を移動したり、サイズ変更すると、スクリプトが矩形の範囲内にあるすべてのデータポイントの平均を計算し、スクリプトウィンドウに出力します。グラフィックオブジェクト範囲を定義して以下のように実行することもできます。

```
GObject gobj = [Graph1]1!rect;
gobj.run();
```

7.1.8. ProjectEvents スクリプト

関数を定義して、操作を実行したり、Origin プロジェクトを開いたり、閉じたり、保存するコマンドを実行することができます。Origin 8.1 以降、デフォルトで、ProjectEvents.ogs というファイルが Origin プロジェクトに添付されています。

このファイルのテンプレートは Origin に付属しており、**EXE** フォルダにあります。このテンプレートファイルは各新規プロジェクトに添付されています。ファイルは、**コードビルダ**で開いて、左パネルの **Project** ノードを開き、見たり、編集することができます。

ProjectEvents.ogs のセクション

デフォルトで、**ProjectEvents.ogs** ファイルは、そのプロジェクトと結びついた 3 つの異なるイベントに対応する 3 つのセクションを含んでいます。

1. AfterOpenDoc:このセクションは、プロジェクトを開いたら直ちに実行されます。
2. BeforeCloseDoc:このセクションは、プロジェクトを閉じる前に実行されます。
3. BeforeSaveDoc:このセクションは、プロジェクトを保存する前に実行します。

ProjectEvents.ogs を利用する

このファイルと内容を実行できるようにするため、ユーザはファイルを編集し、コードビルダで保存する必要があり、そして、プロジェクトを保存します。次回プロジェクトを開いたら、添付 OGS ファイルに含まれるスクリプトコードは、(事前定義のセクション名に従って)指定したイベントで実行されます。

例えば、ユーザが **ProjectEvents.ogs** ファイルの[AfterOpenDoc]セクションで新しい関数を定義して、コードビルダで保存し、Origin のプロジェクトを保存すると、その関数は(グローバルで定義されていれば)プロジェクトが再度開いたときに利用できるようになります。現在のセッションで利用可能な関数を実行するには、実行するセクションにカーソルを置き、**デバッグ**メニュ

ーから、**現在のセクションを実行**を選択します。そして、Origin のスクリプトウィンドウで、**list a** コマンドを発行して、新しい関数が現れ、プロジェクト内で使用できることを確認します。

関数についての簡単なチュートリアルでは、LabTalk のデータセットベースの関数と一緒に使用すると、**ProjectEvents.ogs** の値を表示します。

この OGS ファイルに自分のセクションを追加してカスタムルーチンを保存したり、プロジェクト固有のスクリプトコードを保存することができます。このようなセクションは、イベントドリブンではありませんが、LabTalk スクリプトを実行できる場所からアクセスすることができます。例えば、[MyScript]というセクションをこのファイルに追加すれば、スクリプトウィンドウから次のように実行して、そのセクションのコードはプロジェクトを開いた後に実行されます。

```
run.section(projectevents,myscript);
```

ProjectEvents.ogs スクリプトは、Origin の外部のコマンドコンソールから Origin プロジェクトを開いて、実行することもできます。

7.1.9. インポートウィザードから

は、ASCII、バイナリ、カスタムファイル形式(Origin C で記述したカスタムプログラムを使用)をインポートするのに使用します。ウィザードは、選択した位置でフィルタを保存でき、インポートした後に実行するスクリプトを含めることができます。フィルタを作成すると、データをインポートするのに使うことができ、自動的にスクリプトを実行します。Origin にファイルをドラッグ&ドロップしたとき、**フィルタマネジャー**がそのファイル形式をサポートしていれば、これと同じ機能が適用されます。

例えば、

- インポートウィザードを開始します。
- Origin の Samples\Spectroscopy フォルダを開き、**Peaks with Base.DAT** を選びます。
- 追加をクリックし、OK をクリックします。
- 進むを 6 回クリックして、*フィルターの保存*ページに移動します。
- **フィルタの保存**チェックボックスにチェックを付けます。
- **Subtract Base and Find Peaks** のような適切なフィルタファイル名を入力します。
- **詳細なフィルタオプション**チェックボックスにチェックを付けます。
- 進むをクリックします。
- テキストボックスに次のスクリプトを貼り付けます。

```
range raTime = 1;                // Time 列を範囲として取得
range raAmp = 2;                 // Amp 列を範囲として取得
range raBase = 3;                // Base 列を範囲として取得
wks.addcol(Subtracted);          // Subtracted という列を作成
range raSubtracted = 4;          // Subtracted 列を範囲として取得
raSubtracted = raAmp - raBase;   // Amp から Base を減算
pkFind iy:=(1,4);                // 減算されたデータからピークを探す
range raPeaks = 5;               // ピークインデックス列を範囲として取得
for( idx = 1; idx <= raPeaks.GetSize() ; idx++ )
```

```
{
    pkidx = raPeaks[idx];
    ty Peak found at $(raTime[pkidx]) with height of $(raSubtracted[pkidx]);
}
```

- 完了をクリックします。

これにより、次のことが行われます。

- フィルタの保存
- このフィルタを使ってインポートを実行
- インポート後、スクリプトはデータを減算するスクリプトを実行し、**pkFind** 関数がピークインデックスを位置づけます。結果がスクリプトウィンドウに出力されます。

7.1.10. 非線形フィルタから

非線形フィットには、NLFit ダイアログの**コード**ページに**フィット後のスクリプト**セクションがあります。これは、フィット直後に常に何かを実行した場合に役立ちます。例えば、フィットパラメータ値にアクセスし、別の計算を行ったり、別の分析のために結果を積み上げることができます。

この例では、**フィット後のスクリプト**セクションは、フィットデータセットの名前を追加し、計算したピークの中心を **GaussResults** という名前のワークブックに追加します。

```
// これは初回のみ新しいブックを作成
if(exist(GaussResults)!=2)
{
    newbook name:=GaussResults sheet:=1 option:=1 chkname:=1;
    GaussResults!wks.col1.name$= Dataset;
    GaussResults!wks.col2.name$= PeakCenter;
}

// 最後のレポートシート (this fit)からツリーを取得
getresults iw:=__REPORT$;

//  'GaussResults'の 2 つの列に範囲を割り当て
range ra1 = [GaussResults]1!1;
range ra2 = [GaussResults]1!2;

// 1 列目の行数と次の行番号を取得
size = ra1.GetSize();
size++;

// 最初の列に入力データ範囲を書き出す
ra1[size]$ = ResultsTree.Input.R2.C2$;
// 2 番目の列にピークの中心を書き出す
ra2[size] = ResultsTree.Parameters.xc.Value;
```

7.1.11. 外部アプリケーションから

は Origin を COM サーバにしてデータのやり取りができます。Origin の COM オブジェクトは、他のアプリケーションと接続するためのさまざまなプロパティとメソッドを公開しています。完全に制御するため、Origin には LabTalk から呼び出して実行できる X ファンクションおよび Origin C 関数を含み、LabTalk で利用できる **Execute** メソッドがあります。このサンプルでは、

Visual Basic のシンタックスを使って、Origin を開始し、いくつかのデータをインポートし、ガウスフィットして、ピークの中心を出力します。

```
' Start Origin
Dim oa
Set oa = GetObject("", "Origin.Application")
'oa.Execute ("doc -m 1") ' Uncomment if you want to see Origin
Dim strCmd, strVar As String
Dim dVar As Double

' Wait for Origin to finish startup compile
' (30 seconds is specified here,
'  but function may return in less than 1 second)
oa.Execute ("sec -poc 30")

'Project is empty so create a workbook and import some data
oa.Execute ("newbook")
strVar = oa.LTStr("SYSTEM.PATH.PROGRAM$") +
         "Samples\Curve Fitting\Gaussian.DAT"
oa.Execute ("string fname") ' Declare string in Origin
oa.LTStr("fname$") = strVar ' Set its value
oa.Execute ("impasc")       ' Import

' Do a nonlinear fit (Gauss)
strCmd = "nlbegin 2 Gauss;nlfit;nlend;"
oa.Execute (strCmd)
' Get peak center
dVar = oa.LTVar("nlt.xc")
strVar = "Peak Center at " + CStr(dVar)
bRet = MsgBox(strVar, vbOKOnly, "Gauss Fit")

oa.Exit
Set oa = Nothing
End
```

Samples\COM Server and Client フォルダには、COM クライアントアプリケーションの詳細なサンプルがあります。

7.1.12. コンソールから

外部のコンソール(Windows のコマンドプロンプトなど)のコマンドラインから Origin を起動するとき、オプションの引数が指定されているかどうかを確認するため、**Origin.exe** を呼び出す *以外* のコマンドオプションを読み込みます。

内容

- 3.4 外部 Excel ファイルのサマリーレポートを出力するバッチ処理

コマンドライン引数のシンタックス

コマンドラインのすべての引数はオプションです。Origin に引数を渡すシンタックスは以下の通りです。

Origin.exe [-switch *arg*] [*origin_file_name*] [*labtalk_scripts*]

- ***-switch arg***
  複数のスイッチを渡せます。-r, -r0, -rs(LabTalk スクリプトを引数として使用し、Origin C の起動時のコンパイルの後にスクリプトを実行するスイッチ)を除くほとんどのスイッチは上記の表記に従います。利用可能なスイッチとその機能については、下記のスイッチの表とサンプルをご覧ください。

- ***origin_file_name***
  このファイル名は、Origin プロジェクトまたは Origin ウィンドウファイルを参照します。パスを含めることができます。ファイル拡張子を指定する必要があります。

- ***labtalk_scripts***
  OPJ が開いた後に実行する任意の スクリプトです。これはスクリプトが長い場合に役立ちます。

スイッチ

スイッチ	引数	関数
-A	*cnf file*	INI ファイル中で指定されるリストに追加される設定ファイルを指定します。 設定ファイルには、どの LabTalk コマンドでも含めることができますが、通常メニューコマンドやマクロ定義を含めます。パスを指定することはできません。ファイル拡張子を含めることはできません。ファイルは Origin フォルダ内になければならず、拡張子 CNF でなければなりません。例えば、 *C:\Program Files\OriginLab\Origin8\Origin8.exe -a myconfig* **Note:**-a スイッチを使ってコマンドラインで .cnf ファイルを渡すとき、Origin C は、起動時のコンパイルを終了しないかもしれません。そして、.cnf ファイルが処理される時までにライセンスは処理されません。ですから、自分の.cnf ファイルに X ファンクションを含めたい場合、-a スイッチではなく、-r または -rs スイッチを使ったほうが良いです。
-B	<なし>	-R と同様、OPJ ファイルが開いた後に OPJ ファイルの後のスクリプトを実行しますが、OPJ に結びついた ProjectEvents.ogs の前に実行します。このオプションを使って、ProjectEvents.ogs への変数に渡すことができます。このオプションは、すべてのコマンドライン文字列を使える利点がありますので、-R で必要な括弧は不要となります。(8.1)
-C	*cnf file*	INI ファイル中の指定を無効にする新しいコンフィギュレーションファイルを指定します。 設定ファイルには、どの LabTalk コマンドでも含めることができますが、通常メニューコマンドやマクロ定義を含めます。
-H	<なし>	Origin アプリケーションを非表示にします。スクリプトウィンドウは、内部制御で開く場合には、表示されます。

-HS	<なし>	-h と同じですが、追加して**スクリプトウィンドウ**を開く事も阻止します。これは、スケジュールタスクを正確に実行する際に重要になります。(9.0 SR1)
-I	*ini file*	ORIGIN.INI に変わる新しい初期化ファイルを指定します。通常、複数のスイッチを使用する際には、初期化スイッチ(-i)は他のスイッチより前に設定しなくてはなりません。
-L	*level*	指定したメニューレベルで Origin を起動します。
-M	<なし>	Origin を最小化して実行します。(8,1)
-OCW	*ocw file*	Origin C ワークスペースファイルをロードします。
-P	*フルパス*	Origin のネットワーク版が特定のパスのクライアント側のファイルを探します。
-R	(*script*)	指定した OPJ がロードされたあと、LabTalk *スクリプト*を実行します。 **Note:**このスクリプトは Origin C の起動時のコンパイルの後実行します。
-R0	(*script*)	指定した OPJ がロードされる前、LabTalk *スクリプト*を実行します。 **Note:**このスクリプトは Origin C の起動時のコンパイルの後実行します。
-RS	*script*	-R に似ていますが、OPJ を指定しません。コマンド行の残りの文字列は、LabTalk スクリプトとして使われ、起動時の Origin C コンパイルが完了したら実行します。(8,1)
-SLOG	*file name*	スクリプトウィンドウの出力をファイルに送ります。パスが提供されない場合、ファイルはユーザファイルフォルダに書き込まれます。*ファイル名*が指定されていない時に例えば **-slog -hs** のように他のスイッチが続く場合、**Script_Log.txt** はユーザファイルフォルダに保存されます。(9.0 SR1)
-TL	*file name*	デフォルトのページテンプレートを指定します。
-TM	*otm ファイル*	デフォルトの行列テンプレートを指定します。
-TG	*otp ファイル*	デフォルトのグラフテンプレートを指定します。-TP と同じ
-TP	*otp ファイル*	デフォルトのグラフテンプレートを指定します。-TG と同じ
-TW	*otw ファイル*	デフォルトのワークシートテンプレートを指定します。
-W	<なし>	Origin のネットワーク版が開始フォルダまたは作業ディレクトリのクライアント側のファイルを探します。

サンプル

Origin のプロジェクトファイルをロード

次は、DOS の*.bat ファイルのサンプルです。最初に現在のディレクトリを Origin 実行ファイルのディレクトリに変更します。そして、Origin を呼び出し、次のコマンド行引数を渡します。

- ***-r0 (type -b "opj will open next")***
  -r0 を使って、Origin プロジェクトがロードされる前にスクリプトを実行
- ***-r (type -b "OPJ is now loaded")***
  -r を使って、Origin プロジェクトがロードされた後にスクリプトを実行
- ***c:\mypath\test.opj***
  開く Origin プロジェクトの名前を指定

これら -r, -r0, -rs, -b スイッチは起動時の Origin C コンパイルが完了するまで待つので、このスクリプトの中で X ファンクションを使うことができます。

> *cd "C:\Program Files\OriginLab\Origin8" origin8.exe -r0 (type -b "opj will open next") -r (type -b "OPJ is now loaded") c:\mypath\test.opj*

より複雑なスクリプトに対しては、OGS ファイルにスクリプトを記述し、コマンドラインからその OGS ファイルを実行した方が良いでしょう。

次のサンプルは、ユーザファイルフォルダにある **startup.ogs** ファイルの main セクションにあるスクリプトを実行します。**run.section** メソッドに渡されるファイル名にパスが含まれない場合、ユーザファイルフォルダにファイルがあるものと仮定します。次のコマンド行の引数は、**run.section** オブジェクトメソッドの使用方法を示しています。

> *C:\Program Files\OriginLab\Origin8\Origin8.exe -rs run.section(startup.ogs, main)*

上記のサンプルを示す簡単な **startup.ogs** ファイルは次のようになります。

```
[main]
    type -b "hello, from startup.ogs";
```

コマンドラインコンソールを使って、OPJ ベースのカスタムプログラムを実行

OPJ ファイルに結びついた **ProjectEvents.ogs** スクリプトは、OPJ 中心の処理ツールを作成するのに使うことができます。次のサンプルでは、OPJ を使って、OPJ を直接開くか、Origin 外部のコマンドラインコンソールからそれを呼び出してプログラムを実行することができます。さらに、コマンドラインにプロジェクト変数をセットして、OPJ ファイルがユーザのメニュー操作で開いたのか、コマンドライン引数で開いたのかを示すことができます。

次の ProjectEvents.ogs コードで OPJ を作成します。

```
[AfterOpenDoc]
Function doTask()
{
        type -a "Doing some task...";
        // 実行するコード
        type "Done!";
}
//%2 = 2 はコマンドライン、OPJ のダブルクリック
// この変数で直接制御する方が良い
CheckVar FromCmdLine 0;
if(FromCmdLine)
{
        type -b "Coming from command line";
```

```
        doTask();
        sec -p 2;//閉じる前に少し待つ
        exit;
}
else
{
        type -N "Do you want to do the task now?";
        doTask();
}
```

コマンドラインからこの OPJ(*test* という)を実行するには、-B スイッチを使って、**[AfterOpenDoc]** が実行される前に **FromCmdLine** 変数が定義されるようにします。

<exepath>Origin81.exe -b <opjpath>test.opj FromCmdLine=1

Origin でサマリーレポートを出力するバッチ処理

次のサンプルは、**-rs** スイッチを含む長いスクリプト文字列を入力することで、コマンドラインシェル(例えば、Windows の **cmd**) から Origin を起動する方法を示しています。

スクリプトは数回操作を実行します。

1. 複数のファイル名を持つ文字列変数 (*fname$*) をセットアップします。
2. X ファンクション (batchprocess)を呼び出し、既存の分析テンプレートを使って、バッチ処理を実行します。
3. X ファンクション(*expasc*)を呼び出し、結果を CSV ファイル(*c:\test\my batch\output.csv*)にエクスポートします。
4. Origin プロジェクトファイル(OPJ)を保存する(*doc -s*)ためのプロンプトを出さないようにします。
5. Origin アプリケーションを終了します。

開始するには、外部のシステムレベルのコマンドプロンプト(Windows の **cmd**)でこのコマンドを発行し、Origin のインストールパスをコンピュータまたはネットワークのパスに置き換えます。

C:\Program Files\OriginLab\OriginPro81\Origin81.exe -m -rs template$="C:\Program Files\OriginLab\OriginPro81\Samples\Curve Fitting\autofit.ogw"; fname$="C:\Program Files\OriginLab\OriginPro81\Samples\Curve Fitting\step01.dat%(CRLF)C:\Program Files\OriginLab\OriginPro81\Samples\Curve Fitting\step02.dat%(CRLF)C:\Program Files\OriginLab\OriginPro81\Samples\Curve Fitting\step03.dat%(CRLF)C:\Program Files\OriginLab\OriginPro81\Samples\Curve Fitting\step04.dat%(CRLF)C:\Program Files\OriginLab\OriginPro81\Samples\Curve Fitting\step05.dat%(CRLF)C:\Program Files\OriginLab\OriginPro81\Samples\Curve Fitting\step06.dat"; batchprocess batch:=template name:=template$ fill:="Data" append:="Summary" ow:=[Summary Book]"Summary Sheet"!; expasc iw:=[Summary Book]"Summary Sheet"! type:=csv path:="c:\test\my batch output.csv"; doc -s; exit;

外部 Excel ファイルのサマリーレポートを出力するバッチ処理

このサンプルは、外部の Excel ファイルを使用して、バッチ処理でサマリーレポートを生成する方法を示しています。

1 つの連続したコマンドラインで、次のように実行されます。

1. Origin が起動し、既存のプロジェクトファイル(OPJ)がロードされ、それには以下の 2 つが含まれます。
    - 分析テンプレートとして使用される Origin ワークブック

- レポートファイルとして使用される外部リンクの Excel ファイル

2. 特定のワイルドカード(この場合、拡張子 **.csv* を持ち,ファイル名が *T* で始まる)に合っているすべてのファイルが見つかります。
3. batchProcess X ファンクションはファイルのバッチ処理を実行するために呼び出されます。
4. Excel ウィンドウは、バッチ処理操作の最後でサマリーレポートを含みます。外部 Excel ファイルにリンクしたこのウィンドウが保存され、origin プロジェクトは保存なしで閉じられます。
5. **\Samples\Batch Processing** サブフォルダから直接 Excel ファイルを開き、結果を表示します。

開始するには、外部のシステムレベルのコマンドプロンプト(Windows の **cmd**)でこのコマンドを発行し、Origin のインストールパスをコンピュータまたはネットワークのパスに置き換えます。

```
C:\Program Files\OriginLab\OriginPro81\Origin81.exe -rs string
path$=system.path.program$+"Samples\Batch Processing\";string opj$=path$+"Batch Processing with
Summary Report in External Excel File.opj";doc -o %(opj$);findfiles ext:="T*.csv";win -a
Book1;batchProcess batch:=0 fill:="Raw Data" append:="My Results" ow:="[Book2]Sheet1!" number:=7
label:=1;win -o Book2 {save -i}; doc -s; exit;
```

スクリプトからバッチ処理を行う追加の情報 (ループ と X ファンクションの両方を使う)は、バッチ処理 の章にあります。

7.1.13. タイマー操作

Timer (コマンド) は **TimerProc** マクロを実行し、組合せによって、**n** 秒ごとにスクリプトを実行するのに使用することができます。

次のサンプルは、2 秒ごとに timer 操作を実行し、ディスク上のデータファイルが修正されたかどうかをチェックし、新しければ再インポートします。

このスクリプトサンプルを実行するには、まず次のステップを実行します。

1. 単純な 2 列の ASCII ファイル **c:\temp\mydata.dat** を作成します。別の名前や場所でも構いません。
2. 新しいプロジェクトを開始し、デフォルトの ASCII 設定でファイルを新しいブックにインポートします。ブックのショートネームは **mydata** に変わります。
3. データの線＋シンボルプロットを作成し、グラフの XY 軸スケールプロパティを自動にセットし、新しいデータが追加されたらグラフを更新します。
4. グラフをアクティブウィンドウのままにします。
5. プロジェクトに添付した ProjectEvents.OGS ファイルの[AfterOpenDoc]セクションに下記のスクリプトを保存します。

6. ProjectEvents.OGS の[BeforeCloseDoc]に次のコマンドを追加します。

```
timer -k;
```

7. Origin プロジェクトを保存し、閉じ、プロジェクトを再度開きます。プロジェクトを開いた時に、timer が開始し、プロジェクトを閉じたときに、timer が実行を停止します。
8. ディスク上のデータファイルに行き、編集して、2, 3 のデータポイントを追加します。
9. timer 操作により再インポートが実行され、グラフが新しいデータで更新されます。

```
// TimerProc マクロのセットアップ
def TimerProc {
        // ファイルが存在するかをチェック、無ければ終了
        string str$="c:\temp\mydata.dat";
        if(0 == exist(str$) ) return;

        // ディスク上のファイルの日付/時間を取得
        double dtDisk = exist(str$,5);

        // データブックのスクリプトを実行
        // ここでブックのショートネームは mydata であることが前提
        win -o mydata {
                // 最後のインポートの日付/時間を取得
                double dtLast = page.info.system.import.filedate;

                // ディスクのファイルが新しければ、ファイルを再インポート
                if( dtDisk > dtLast ) reimport;
        }
}

// 2 秒ごとに TimerProc を実行するようセット
timer 2;
```

Samples\LabTalk Script Examples サブフォルダには、**Reimport File Using Timer.OPJ** という Origin プロジェクトがあり、これは上記でセットアップしたスクリプトに似ています。この OPJ を開き、スクリプトを表示して、この機能を試します。

7.1.14. Origin 起動中の処理

Origin が起動するとき、トリガーとなる複数のイベントがあります。**OEvents.OGS** ファイルを使って各イベントで実行するように LabTalk スクリプトをセットすることができます。

例えば、すべての Origin C 関数が起動時にコンパイルされたあとで、自分のスクリプトを実行したいかもしれません。次のサンプルは Origin 起動時に、ユーザ定義の LabTalk 関数 を追加するデモです。これらの関数は、すべての Origin セッションで利用できます。

1. 次のスクリプトで新しい OGS ファイル, *MyLTFuncs.OGS* をユーザファイルフォルダに作成します。

```
[DefFuncs]
@global = 1;
function int myswap(ref double a, ref double b)
{
        double temp = a;
        a = b;
        b = temp;
        return 0;
}
```

2. Origin フォルダから *OEvents.OGS* をユーザファイルフォルダにコピーします。別の方法として、Origin フォルダからファイルを開くときに、ユーザファイルフォルダに保存します。

> **Note:**ユーザファイルフォルダにあるすべてのシステム OGS ファイル、CNF ファイルなどをコピーし編集してください。

3. この OEvents.OGS ファイルは、スクリプト実行時に示す *[AfterCompileSystem]*, *[BeforeOpenDoc]*, *[OnExitOrigin]*複数のセクションを含みます。

4. *[AfterCompileSystem]* というセクションで、次のスクリプトを追加します。

```
// LabTalk 関数定義スクリプトファイルを実行
run.section(MyLTFuncs, DefFuncs);
```

5. OEvents.OGS でセクションを実行するには、ユーザファイルフォルダで *Origin.ini* を編集します。実行していたら Origin を終了し、Origin.ini ファイルを編集し、"OEvents"セクションの下にある行を下記のようにコメント解除 (;を消去)します。

```
Ogs1 = OEvents
; Ogs2 = OEvents
; Origin can trigger multiple system events
; Uncomment this line and implement event handlers in OEvents.ogs
```

> **Note:**1 つ以上のイベントハンドラーファイルが *Origin.ini* に存在していて、名前が OEvents.OGS でないかもしれません。

6. 新しい Origin セッションを開始し、次のテストスクリプトを実行し、ユーザ定義関数が動作しているかをチェックします。

```
double a = 1.1;
double b = 2.2;
ty "At the beginning, a = $(a), and b = $(b)";
myswap(a, b);
ty "After swap, a = $(a), and b = $(b)";
```

Note1:イベントに結びついたカスタムスクリプトから Origin C 関数を呼び出す必要がある場合、Origin C ファイルがコンパイルされ、関数がスクリプトアクセスできる状態にあることを確実にする必要があります。詳細は Origin C 関数のロードとコンパイル をご覧ください。

Note2:イベントは間接的に ORIGIN.INI ファイルで決まるので、複数の INI ファイルを作成することでカスタム環境を作成できます。cmd コンソールまたはショートカットのようにコマンド行を指定することによってカスタム INI ファイルを使って Origin を起動できます。コンソールからのスクリプトをご覧ください。

7.1.15. カスタムメニュー項目から

LabTalk スクリプトはカスタムメニューアイテムに割り当てできます。Origin の**ツール**メニューからアクセスできる **カスタムメニューオーガナイザー**は、メニュー項目を簡単に追加したり、編集できます。このダイアログの**カスタムメニュー**タブは、新しいメニュー項目を追加し、ポップアップメニューとセパレータを含むサブメニュー項目を追加します。メニュー項目が追加されると、LabTalk スクリプトはその項目に対して割り当てることができます。メニュー項目は、すべてのウィンドウタイプまたは特定のウィンドウタイプで利用できます。

そしてカスタムメニュー設定を保存し、複数の設定ファイルを作成して、**フォーマット: メニュー** メニューを使って別にロードされます。詳細な情報については、ヘルプファイルのカスタムメニューオーガナイザーダイアログのページをご覧ください。

7.1.16. ツールバーボタンから

LabTalk スクリプトファイルは Origin の ツールバーボタンからも起動できます。LabTalk を使いましょうの章で、ツールバーのカスタムルーチンボタンの使用方法について説明しました。ここでは、その詳細について説明します。3 つのファイルによってこれができます。

1. ボタンの外観を定義するビットマップファイルは、Origin が提供しているボタン または 自分自身で作成したボタンの一方を使用します。
2. ユーザがボタンをクリックするときに実行される LabTalk スクリプトファイル
3. ボタンまたはボタングループについての情報が保存される INI ファイル。Origin は、下の手順に従うときに INI ファイルを作成します。

ボタン自体を定義するビットマップファイル(BMP)を持っているものとします(作成することに関心があれば、以下のステップがサンプルになっています。)

最初に、**コードビルダ** (Origin の標準ツールバーの  を選択して開く) または他のテキストエディタを使って、LabTalk スクリプト (OGS) ファイルを開発します。拡張子 OGS を付けて保存します。1 つのスクリプトファイルをいくつかのセクションに分け、各セクションを異なるツールバーボタンにすることができます。

内容

Origin ツールバーのボタンを配置する

Origin ツールバーにボタンを配置するには、この手順を使います。

1. Origin で、**表示**:**ツールバー**を選び、ツールバーのカスタム化ダイアログを開きます。
2. **ボタングループ**タブをアクティブにします。
3. **ボタングループ**にある**新規**ボタンをクリックし、ボタングループの作成ダイアログを開きます。
4. グループ名を入力します。
5. この新しいグループに対するボタンの数を入力します。
6. 参照ボタンをクリックし、ビットマップファイルの場所に移動します。このファイルは、ユーザフォルダに置いておくほうが良いでしょう。
7. **OK** をクリックします。
8. **名前を付けて保存**ダイアログが開きます。ビットマップファイルと同じ名前を入力します。**OK** をクリックして INI ファイルを保存します。作成したグループがグループリストに追加され、ボタンが表示されます。

OPX ファイルにエクスポートするカスタムボタングループを作成する時には、**ユーザファイル**フォルダの中にサブフォルダを作成し、そこにボタングループの INI ファイル、ビットマップファイル、スクリプトファイル、他のサポートファイルを保存することをお勧めします。他の Origin ユーザが、その OPX ファイルをインストールする時に、**ユーザファイル**フォルダ内に、同じサブフォルダが自動的に作成されます。そして、このサブフォルダ内に、カスタムボタングループに関するファイルがコピーされます。この方法で Origin サブフォルダを使用すれば、自分で作成したファイルと Origin のファイルを分けておくことができます。

LabTalk スクリプト(OGS) ファイルとボタンを合わせる

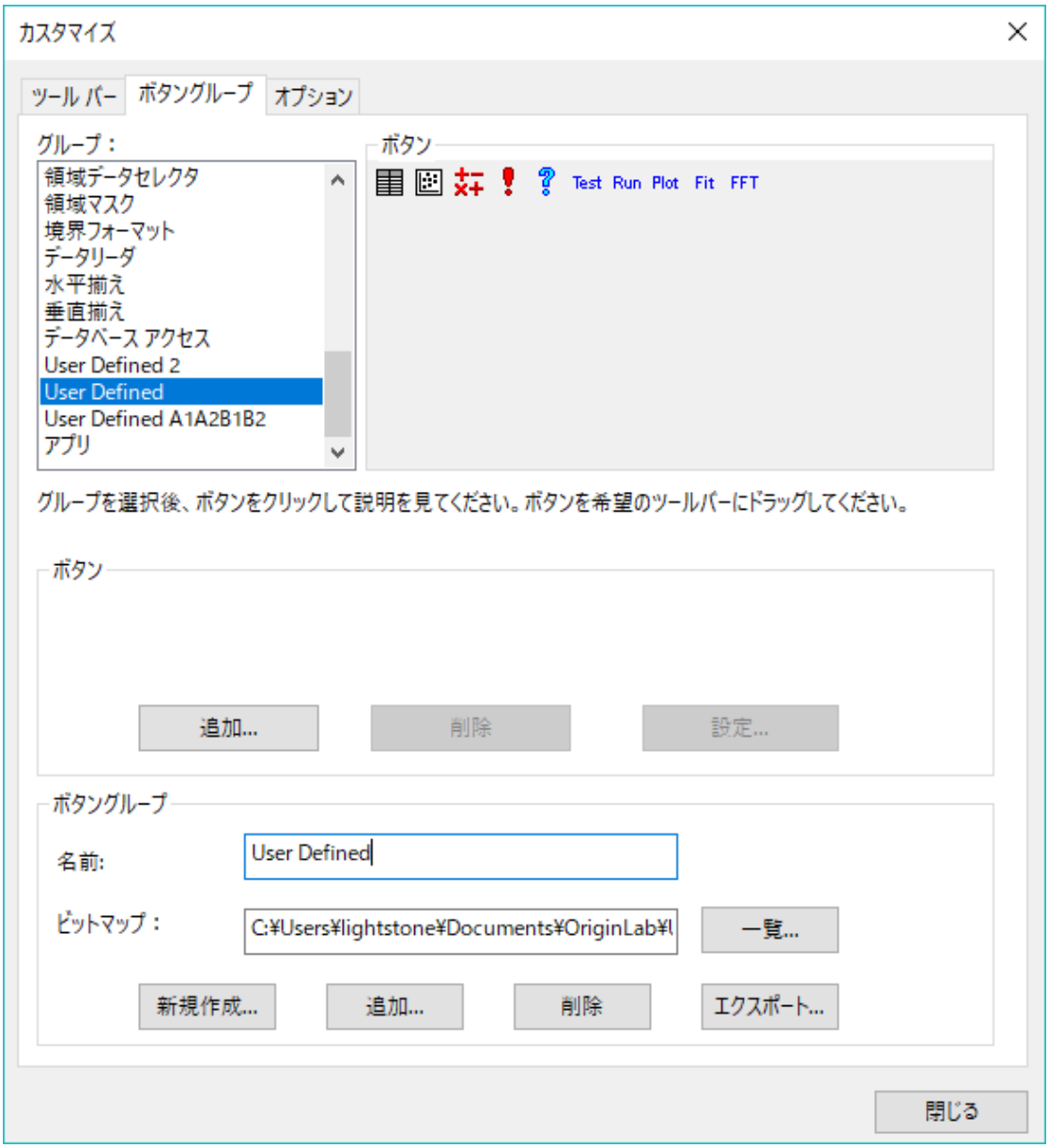
1. ボタンをクリックして選択します。
2. 設定ボタンをクリックし、ボタン設定ダイアログを開きます。
3. 参照ボタンをクリックし、OGS ファイルのある場所に移動します。
4. OGS ファイルのセクション名を、引数リストに引数入力します。
5. ツールチップテキストボックスに簡単な説明を入力します。
6. ステータスバーテキストボックスにステータスバーに出力するメッセージを入力します。
7. **OK** をクリックします。

8. ボタングループの各ボタンに対してこれらのステップを繰り返します。

9. 初めのボタンを Origin のワークスペースにドラッグします。ツールバーが作成されます。これでほかのボタン全てをツールバーにドラッグできます。

Origin で利用できるカスタムボタン

以下のダイアログは、Origin のメインメニュー**表示**:**ツールバー**を選択して開くことができます。**ボタングループ**タブで、**ユーザ定義**グループを選びます。



これらを使って始めるために Origin ツールバーにこれらのボタンをドラッグします。上記の手順を使い、与えられたボタンとスクリプトを関連付けします。

新しいボタン用のビットマップファイルを作成する

ビットマップファイルを作成するには、Windows のペイントのようなビットマップ画像(BMP)を編集し、保存できるプログラムを使用します。以下のステップは始める際の手助けとなります。

1. 組込みのユーザ定義のツールバーのビットマップを使うことは最初に始めるのに適しています。Windows ペイントで、**ファイル:開く**を選び、ユーザファイルフォルダを使って、**Userdef.bmp** を選択します。
2. 画像サイズをセットします。**イメージ: サイズ変更と傾斜**を選びます。画像の高さは 16 のままである必要があるので変更しないでください。各ボタンは 16x16 ピクセルになります。ツールバーに 2 つだけボタンが必要なら、幅を 32 に変更します。幅は常に 16 掛けるボタンの数で、最大 10 個のボタン分、つまり 160 ピクセルです。
3. **表示:ズーム:カスタム:800%**を選びます。これでイメージは操作するのに十分大きくなりました。
4. **表示:ズーム:グリッド表示**を選択します。これでピクセルごとに色を付けることができます。各ボタンの外観をデザインします。
5. **ファイル:名前を付けて保存**を選び、新しいファイル名を入力し、**ファイルの種類** は **16 色ビットマップ**にします。

7.2. スクリプトのデバッグ

このセクションでは、LabTalk スクリプトをデバッグする方法を説明します。最初のパートでは、スクリプトのインタラクティブな実行を紹介します。つぎに、Origin のスクリプトエディタであるコードビルダを含むいくつかのデバッグツールを説明します。最後のパートでは、エラーの取扱いについて説明します。

このセクションで説明している項目

- インタラクティブな実行
- デバッグツール
- エラーの取り扱い

7.2.1. インタラクティブな実行

LabTalk コマンドまたは X ファンクションを 1 行ずつ(または複数行を選択して)実行し、ステップバイステップで処理できます。この処理方法のメリットは、スクリプト開発中などで、発行したコマンドの結果を確認できることや、その結果やエラーを参照して適切な対応をとれることです。

LabTalk コマンドをインタラクティブに実行するには、次の場所にスクリプトを入れます。

- スクリプトウィンドウ
- Origin メインウィンドウにあるコマンドウィンドウ
- コードビルダのコマンド結果ウィンドウ

各ウィンドウの特徴およびメリットには、次のようなものがあります。

スクリプトウィンドウ

スクリプトウィンドウは、**ウィンドウ**メニューから開くことができます。これは、上級ユーザのための LabTalk スクリプトを実行する柔軟な場所です。Enter キーを押すと次のように実行されます。

1. 選択部分がない場合、現在カーソルがある行を実行

2. 選択部分がある場合、選択部分を実行

Ctrl+Enter キーを押すと、実行せずに改行でき、続けてコードを記述できます。また、編集メニューの**スクリプトの実行**オプションで、編集とインタラクティブな実行を切り替えられます。

Origin メインウィンドウにあるコマンドウィンドウ

コマンドウィンドウのコマンドプロンプトに LabTalk コマンドを入力できます。入力したコマンド行のすぐ後に結果が出力されます。コマンドウィンドウには、コマンド履歴パネル、オートコンプリート、前に実行したコマンドを再利用するためのロールバックのサポート、以前に実行したコマンドブロックを実行、以前に実行したコマンドを OGS ファイルに保存するなどさまざまな便利な機能があります。しかし、コマンドウィンドウでは、複数行にわたるスクリプトを編集できません。

コマンドウィンドウの使用方法については、Origin ヘルプファイルの **Origin のコマンドウィンドウ**をご覧下さい。

コードビルダのコマンド結果ウィンドウ

コードビルダーは Origin の統合された開発環境で、LabTalk スクリプト、OriginC コード、X ファンクションコードなどのデバッグをする際に便利です。コードビルダーでは、ブレークポイントの設定といった便利なデバッグツールや、ステップバイステップの実行、変数の値の検証を使用してデバッグできます。

コードビルダを使用する方法については、プログラミングヘルプファイルのコードビルダユーザガイドをご覧下さい。

7.2.2. デバッグツール

Origin は LabTalk スクリプトを開発およびデバッグするさまざまなツールを提供しています。

内容

- 15 Type <ogsFileName> (コマンド)
- 16 ログをファイルに出力

コードビルダ (Origin の機能)

コードビルダ は、Origin の統合開発環境で、LabTalk スクリプト、Origin C のコード、X ファンクションのコード、Origin C で記述されたフィット関数のデバッグを行います。コードビルダでは、ブレークポイントの設定、ステップバイステップによる実行、変数値のインスペクションなど便利なデバッグツールを使うことができます。コードビルダは ed.open() メソッドで開くことができます。

コードビルダを使用する方法については、プログラミングヘルプファイルの**コードビルダユーザガイド**をご覧下さい。

ここではコードビルダで LabTalk スクリプトをデバッグする方法についてサンプルを紹介します。

1. 以下のスクリプトを実行して OGS ファイルを開きます。

```
// ogs ファイルをコードビルダで開く
file$ = system.path.program$ + "Samples\LabTalk Script
Examples\ave_traces.ogs";
ed.open(%(file$));
```

2. 開いたファイルで 22 行目にある、以下のスクリプトの左側の余白をクリックして、ブレークポイントをセットします。

```
fname$ = system.path.program$ + "Samples\Data
Manipulation\not_monotonic_multicurve.dat";
```

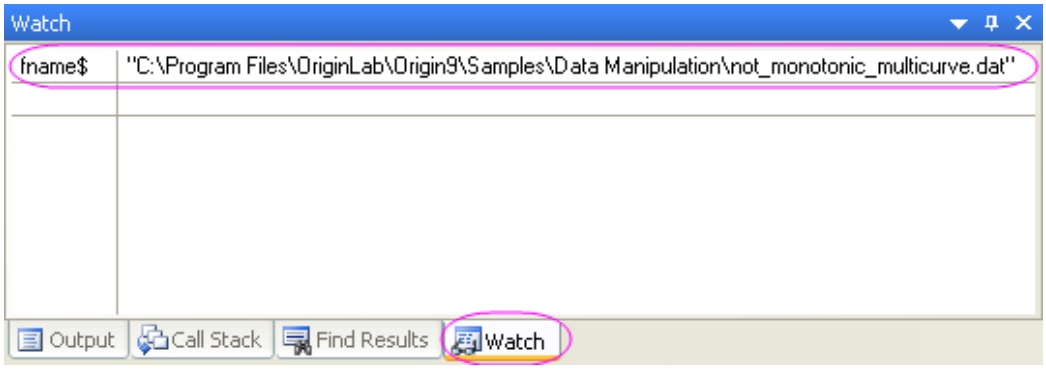
ブレークポイントは次のようになります。

```
//test to make sure OriginPro is installed
if (system.product&1 != 1)
{
    type "This feature is only available in OriginPro 8.";
    break;
}

// Put the path of sample data into fname string variable which is the default used by impASC
fname$ = system.path.program$ + "Samples\Data Manipulation\not_monotonic_multicurve.dat";

newbook;// Create a new book
impASC;// import the file using all defaults
string bkn$ = %H; // save the book name as plotting will create new window to change %H
plotxy [bkn$]!((1,2), (3,4), (5,6), (7,8)) plot:=200;
```

3. 12 行目-**[main]**セクション-にカーソルを合わせます。それから、メニューで**デバッグ**:**現セクションの実行**を選択します。**[Main]**セクションのコードが実行され、ブレークポイントで止まります。

```
//test to make sure OriginPro is installed
if (system.product&1 != 1)
{
    type "This feature is only available in OriginPro 8.";
    break;
}

// Put the path of sample data into fname string variable which is the default used by impASC
fname$ = system.path.program$ + "Samples\Data Manipulation\not_monotonic_multicurve.dat";

newbook;// Create a new book
impASC;// import the file using all defaults
string bkn$ = %H; // save the book name as plotting will create new window to change %H
plotxy [bkn$]!((1,2), (3,4), (5,6), (7,8)) plot:=200;
```

4. では、F10 を押して 1 行ごとにコマンドを実行していきましょう。コードビルダは**ウォッチ**ウィンドウでデバッグ中の変数の値を確認できます。例えば、F10 を 1 回押したあと、まだ開いていないならば**表示**:**ウォッチ**と操作して**ウォッチ**ウィンドウを開きます。このウィンドウ内のテーブルに左側のセルに *fname$* と入力すると、値が同じ行の右側のセルに表示されます。



5. 残りのスクリプトを実行するには、F5 キーを押してください。他のブレークポイントがなければ、最後まで実行されます。

Ed (オブジェクト)

Ed (オブジェクト)は、コードビルダにアクセスするスクリプトを提供し、LabTalk スクリプトと Origin C コード専用のエディタです。

ed オブジェクトメソッドは次の通りです。

メソッド	説明
ed.open()	コードビルダウィンドウを開きます。
ed.open(fileName)	コードビルダウィンドウに指定したファイル(fileName)を開きます。
ed.open(fileName, sectionName)	コードビルダウィンドウに指定した OGS ファイルの指定したセクション(sectionName)を開きます。(見つからなかった場合、デフォルトは最初を開きます。)

コードビルダを開く

```
ed.open()
```

コードビルダで特定のファイルを開く

次のコマンドはファイル **myscript.ogs** を開きます。

```
ed.open(E:\myfolder\myscript.ogs)
```

事前保存のパスでファイルを開く

cd (コマンド) X ファンクションを使って、最初に特定のフォルダに切り替えます。

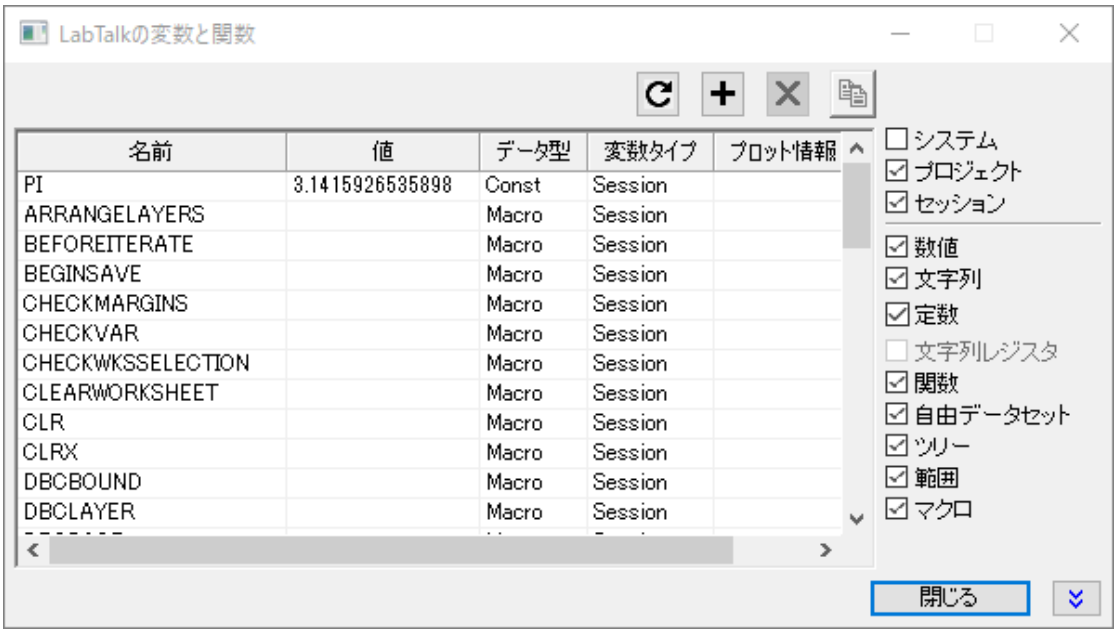
```
cd 2;
ed.open(autofit.ogs);
```

LabTalk 変数と関数のダイアログ

オプション無しの **list** コマンド だけでなく、**ed** (コマンド) (**ed** オブジェクトとは異なる) は、LabTalk 変数ダイアログを開きます。このダイアログは、現在のプロジェクトにあるすべての変数の属性の表です。属性には、変数名、値、データ型、サブデータ型、プロパティ、プロット情報、説明があります。

これは、変数の現在の値とプロパティをリアルタイムで表示できるので、スクリプトのプログラマーにはとても便利です。さらに、変数の属性がテキスト記述子の場合、変数をアルファベット順にソートでき、数値の場合、数値でソートできます。

全変数リストのサブセットを表示できるダイアログの右側にチェックボックスがあります。



Echo (システム変数)

デバッグおよびトレースを行うため、**Echo** 変数は、スクリプトやエラーメッセージをコマンドウィンドウ(または利用可能ならスクリプトウィンドウ)に出力します。echo を有効にするには、次のようにスクリプトウィンドウに入力します。

echo = *Number*

Number は次のいずれかです。

Number	説明
1	エラーとなったコマンドの表示
2	遅延実行のためキューに送られたスクリプトを表示します。
4	実行されたコマンドの表示

8	実行された代入文の表示
16	マクロの表示

Note: これらの値は、2 進数の異なるビット位置に対応しており、それらを足し合わせると、組み合わされた設定となります。例えば、echo = 12 は、コマンドと代入文の両方のスクリプトを表示します。Echo = 7 は、echo = 1, echo = 2, echo = 4 の組合せで、メニューコマンドを選択したときに実行されるスクリプトを表示させたい場合に便利です。echo を解除するには、スクリプトウィンドウに echo = 0 と入力します。

#!script (特別なシンタックス)

この表記を使ってスクリプトにデバッグ用ステートメントを埋め込みます。#記号は、LabTalk インタープリタにその行の最後までのテキストを無視するよう通知するものです。しかし、その後に ! 記号が続いている場合、@B システム変数 (または System.Debug オブジェクトプロパティ)が 1 にセットされている場合にはスクリプトが実行されます。次は、このオプションのサンプルを示しています。

```
@B = 1;
range rr = [Book1]Sheet1!col(A);  // 列 A を Range 変数に
for (ii=1; ii<=10; ii+=1) {
  #!ii=; rr[ii]=;  // デバッグ用埋め込みステートメント
  rr[ii]+=ii*10;
}
@B = 0;
#!type -a This line will not execute
```

このスクリプトは @B を 1 としてセットし、#! 行を実行できるようにしています。@B を 0 としてセットすると、最後の行は実行しません。

{script} (特別なシンタックス)

LabTalk コード内のエラーによって、その場所でコードが止まり、エラーの後のステートメントは実行されません。このような場合でもスクリプトに続けて実行してほしい場合、中括弧({ })でエラーの扱いを止めるところと再開するところを指定します。例えば、次のスクリプトで

```
type Start;
impasc fname:=BadFileName;
type End;
```

Start という語句がスクリプトウィンドウに出力されますが、**BadFileName** が見つからない場合、スクリプトはそこで実行を停止し、**End** は出力されません。

そこで、次のように中括弧({ })で疑わしい行を囲みます。

```
type Start;
{
   impasc fname:=BadFileName;
}
type End;
```

すると、**BadFileName** が適切にインポートされたかどうかに関係なく、**End** が出力されます。

この条件を変数で捕まえることもできます。

```
flag = 1;
{
   impasc fname:=MyFile;
   flag = 0;
}
if(flag)
   type Error ocurred;
else
   type OK;
```

OGS ファイル内のセクションが実行できない場合にも、同様の状況になります。コードは何も出力せず、呼び出しているコンテキストに戻ります。上記の方法を使用してどのコードが実行できないか、確認できます。この場合、括弧は特に必要ありません。

```
[Called Section]
flag = 1;
BadCommand; // この行でエラーが起こり、戻る
flag = 0;  // フラグ（グローバル変数である必要があります）が 1 の場合、// 上記コードが失敗した事を示します。
```

@B(システム変数), System.Debug (オブジェクトプロパティ)

@B システム変数は、デバッグモードを制御して、**#!**で始まる LabTalk ステートメントを実行します。:

```
1 = enable
0 = disable
```

これは **System.Debug** オブジェクトプロパティと同じです。

@OC (システム変数)

@OC (システム変数)は、LabTalk から Origin C 関数を呼び出すことができるかどうかを制御します。

値	説明
@OC = 1 (デフォルト)	Origin C 関数を呼び出すことが**できる**
@OC = 0	Origin C 関数を呼び出すことは**できない**

@V(システム変数), System.Version(オブジェクトプロパティ)

@V は Origin のバージョン番号を示します。@V と System.Version オブジェクトプロパティは同じです。

@VDF (システム変数)

@VDF = 1 にセットされている場合、プロジェクトファイル(OPJ)を開くとき、Origin はファイルを保存した Origin のバージョンを出力します。

VarName= (コマンド)

このコマンドは、変数の値を調べます。スクリプトにこれを埋込みスクリプト実行中にスクリプトウィンドウに中間的な変数の値を表示します。

サンプル 1

次のコマンドは myHight 変数の値を出力します。

```
myHight=
```

LabTalk:List (コマンド)

list コマンドはシステム環境を調べるのに使います。例えば、list s コマンドはプロジェクト内のすべてのデータセット(一時データセットを含む)を表示します。

ErrorProc (マクロ)

マクロタイプ:次のような場合に、特別なイベント ErrorProc マクロが実行されます。 LabTalk インタープリターが#Command Error を検出した時、 ダイアログボックスでキャンセルがクリックされた時、 キャンセルボタンの無いダイアログボックスで「いいえ」がクリックされた時、ErrorProc マクロは実行直後に削除されます。ErrorProc マクロはエラーをトラップするのに有効です。

NotReady (マクロ)

この NotReady マクロは、「This operation is still under development...」というメッセージを OK ボタン付きのダイアログボックスに表示します。

Type <ogsFileName> (コマンド)

このコマンドは *type* コマンドの一種で、実行中のディレクトリ内で指定したスクリプトファイル(.OGS)の内容を、コマンド(もしくはスクリプト)ウィンドウに出力します。ogsFileName 内のファイル拡張子、.OGS は省略されることに注意してください。ファイル名はパスを含むことができず、作業フォルダ内にある必要があります。

例:

次のスクリプトは D の内容を表示します。\temp\mytemp1.ogs and C:\myogs\hello.ogs.

```
cd D:\Temp;
type mytemp1.ogs; // 拡張子を含む
cd C:\temp;
type hello; // 拡張子は含まない
```

ログをファイルに出力

ログの情報をファイルに出力するには type コマンドがあります。type -gb はログファイルの出力先を指定し、出力ルーチンを開始します。type -ge はルーチンを終了し、ファイルへのログを終了します。例えば、

```
type -gb %Ylog.txt;  // テキストをファイル log.txt に書き始める// 存在しない場合は作成する
type aa;  // aa と記入
type bb;  // bb と記入
type cc;  // cc と記入
type -ge;  // 記入終了
```

これは、スクリプトが 30000 バイトのバッファしかないため、スクリプトウィンドウに非常に大きな出力をしている際に特に便利です。

7.2.3. エラーの取扱い

エラーが発生すると LabTalk スクリプトは中断します。しかし、エラーが発生してもスクリプトの実行を続けたいときがあります。この場合、Origin でエラーが発生するスクリプトの一部分を中括弧 ("{" と "}") で囲みます。Origin はエラーがあるセクションに到達すると、"}" までのスクリプトは飛ばし、中括弧の外から実行が再開されます。この意味では、中括弧と run.section() コマンドは同じ動作になります。

以下は、エラーを取り扱う方法を示す単純なサンプルです。スクリプトウィンドウでスクリプトを実行する前に、新しいワークシートを作成し、列 C が存在しないことを確認してください。

```
// エラー取扱いなしのスクリプト
type "Start the section";
stats col(c);
stats.max=;
type "Finished the section";
```

コード行 `stats col(c);` は、列 C が存在しないのでエラーが発生します。そして、スクリプトが中断し、次の出力だけ行われます。

```
Start the section
Failed to resolve range string, VarName = ix, VarValue = col(c)
```

今度は、中括弧を入れて、エラーの取扱いを行います。エラーが発生したかどうかを示す変数を追加し、システム変数を使って Origin のエラーメッセージを一時的に遮断します。

```
// エラーの取り扱いのあるスクリプト
type "Start the section";
int iNOE = @NOE;  // 現在の Origin エラーメッセージ出力フラグを保存する
// エラーの発生するセクション
{
        @NOE = 0; // Origin のエラーメッセージを中断する
        vErr = 1; // エラー変数を true (1)にセット
        stats col(c); // これがエラーを引き起こす可能性があるコード
        stats.max=; // エラーが無い時のみ実行は継続する
        vErr = 0; // エラーが無い場合、変数は false (0)を返す
}
@NOE = iNOE; // Origin のエラーメッセージを再表示
if(vErr) ty An error occurred.Continuing ...;
type "Finished the section";
```

出力は次のとおりです。

```
Start the section
An error occurred.Continuing ...
Finished the section
```

コード `stats col(c)` でのエラーの後、中括弧 (}) の外側のコードが実行され、必要に応じてエラーをトラップして処理を行います。もし、発生したすべてのエラーの記録を保持したい場合、@ NOE に関連する行を、メッセージログでコメントアウトすることができます。

8 文字列の処理

この章は文字列で作業を行う方法、つまり文字列変数について、登録と配列、数値を文字列に変換、文字列を数値に変換を含む、文字列を処理する際に使用できる方法を紹介します。

この章で説明している項目

- 文字列変数と文字列レジスタ
- 文字列の処理
- 文字列を数値に変換
- 数値を文字列に変換
- 文字列配列

8.1. 文字列変数と文字列レジスタ

Origin では、文字列の処理は、文字列変数または文字列レジスタの 2 つの方法でサポートされています。一般には、直感的に使用できる文字列変数を使うことをお勧めします(他のプログラミング言語での文字列のように使えます)。そして、事前定義されている多くの文字列メソッドをサポートしています。どちらも文字列レジスタより高度な機能です。

8.1.1. 文字列変数

文字列変数は、宣言または代入によって作成され、その名前には常に$記号が付けられます。例えば、

```
// 文字列タイプの'aa'という変数を宣言することで作成
//'aa' は空 (i.e., "")
string aa$;

//  'aa' に文字列シーケンスを割り当て
aa$ = "Happy";

// 文字列変数 'bb'を作成し値を割り当て
//同じ行でこれをすべて行う
string bb$ = "Yes";

// 宣言なしで文字列変数 'cc' を作成し値を割り当て
//(下記参照)
cc$ = "Global";
```

Note:文字列変数 **cc** は宣言されなかったので、プロジェクトスコープが与えられ、これはすべてのルーチン、関数などがプロジェクトが開いている限り見えることを意味しています。宣言された変数 **aa** と **bb** は、ローカル(セッション)スコープが与えられます。スコープについての詳細は、変数とスコープをご覧下さい。

8.1.2. 文字列レジスタ

Origin 8.0 より前のバージョンでは、Origin は文字列レジスタで文字列の処理をサポートしていました。そのため、文字列レジスタは最新のバージョンでもサポートされており、スクリプトプログラミング例で使われているのが分かります。英語のアルファベット 26 文字に対応した 26 個の文字列レジスタがあり、%記号を使って、%A--%Z のように表します。これらは、文字列変数のように文字列シーケンスを割り当てることもできますが、下記のサンプルで示すように、扱い方が少し異なります。26 個の文字列レジスタのいくつか、%C--%I, と %X--%Z の範囲は、システム利用のため予約されています。この使い方についての説明は、文字列レジスタをご覧ください。

8.2. 文字列の処理

内容

文字列メソッドを使う

これらのサンプルは、長い文字列(この例ではフルパスのファイル名)から、部分文字列(ファイル名)を取得する方法をいくつか示しています。最後に、2 つの文字列を連結する方法を示します。

getFileName()を使って部分文字列を探す

この例では、文字列メソッドは、特定の事例として作成していますが、一般に必要な操作に使うことができます。

```
// 組込みの文字列メソッド GetFileName()を使用
string fname$="C:\Program Files\Origin 8\Samples\Import\S15-125-03.dat";
string str1$ = fname.GetFileName()$;
str1$=;
```

reverseFind(), mid()メソッドを使って、部分文字列を探す

ここでは、文字列メソッドを組み合わせて使用します。

```
// 関数 ReverseFind と Mid を使ってファイル名を抽出
string fname$="C:\Program Files\Origin 8\Samples\Import\S15-125-03.dat";
// 右から探すことで、最後の '\' の位置を見つける
int nn=fname.ReverseFind('\');
// その位置の後に開始し、終わりまでの部分文字列を取得
string str2$=fname.Mid(nn1)$;
// ファイル名をスクリプトウィンドウに出力
str2$=;
```

トークンを使って部分文字列を探す

ここでは、別の一般的な探索方法を使って、操作を完了させます。

```
// トークンを使って、ファイル名を抽出
string fname$="C:\Program Files\Origin 8\Samples\Import\S15-125-03.dat";
//  '\' 記号で区切られたトークンの数を取得
int nn=fname.GetNumTokens('\');
// 最後のトークンを取得
string str3$ = fname.GetToken(nn, '\')$;
// そのトークンの値をスクリプトウィンドウに出力
str3$=;
```

文字列の連結

'+' 演算子を使って、文字列を連結することができます。以下に示すように行います。

```
string aa$="reading";
string bb$="he likes " + aa$ + " books";
type "He said " + bb$;
```

insert 文字列メソッドを使って、2 つの文字列を連結することができます。

```
string aa$ = "Happy";
string bb$ = " Go Lucky";
// 文字列 'aa' を文字列 'bb' に位置 1 で挿入
bb.insert(1,aa$);
bb$=;
```

サポートされている文字列メソッドの一覧と説明については、String (オブジェクト)をご覧下さい。

文字列レジスタを使う

文字列レジスタ は、より簡単で強力に使うことができますが、文字列変数とそのメソッドに比べ、読み込みが難しくなります。また、これらは、グローバル(セッションスコープ)なので、他のプログラムにより修正されることが少なくなります。

```
// 2 つの文字列を文字列レジスタを使って連結
%A="Left";
%B="Handed";
%N="%A %B";
%N=           // "Left Handed"
// 長いファイルパスの文字列からファイル名の部分文字列を抽出
%N="C:\Program Files\Origin 8\Samples\Import\S15-125-03.dat";
for(done=0;done==0; )
{
    %M=%[%N,>'\'];
    if(%[%M]>0) %N = %M;
    else done = 1;
}
%N=;
```

文字列から数値を抽出

このサンプルは、文字列から数値を抽出する複数の方法を示しています。

```
// 文字列変数は多くのメソッドをサポート
string fname$="S15-125-03.dat";

int nn=fname.Find('S');
string str1$ = fname.Mid(nn+1, 2)$;
type "1st number = %(str1$)";

string str2$ = fname.Between("-", "-")$;
type "2nd number = %(str2$)";

int nn = fname.ReverseFind('-');
int oo = fname.ReverseFind('.');
string str3$ = fname.Mid(nn + 1, oo - nn - 1)$;
type "3rd number = %(str3$)";

type $(%(str2$) - %(str1$) * %(str3$));

// 文字列レジスタを使って、部分文字列の表記を使用可能
%M = "S15-125-03.dat";
%N = %[%M,2:3];  // 開始と終了を指定
type "1st number = %N";
%N = %[%M,>'S']; // 'S'の後の文字列を探す
%N = %[%N,'-'];  //  '-'の前の残りを探す
type "1st number = %N";
%O = %[%M,#2,\x2D];  //  '-' (16 進数で 2D) で区切られた 2 番目のトークンを探す
type "2nd number = %O";
%P = %[%M,'.'];  // 拡張子を取る
%P = %[%P,>'-']; // 最初の '-'の後
%P = %[%P,>'-']; // 2 番目の '-'の後
type "3rd number = %P";
type $(%O - %N * %P);
```

8.3. 文字列を数値に変換

次のサンプルは、数値の文字列を実際の数値に変換する方法です。

8.3.1. 置換表記を使う

文字列型の変数を数値型(double, int, const)の変数に変換するため、次の簡単なサンプルを考えます。

```
// myString は文字 456 を文字列として含む
string myString$ = "456";

// myStringNum は整数値 456 を含む
int myStringNum = %(myString$);
```

シンタックス **%(string$)** は、LabTalk でサポートされている 2 つの置換表記の 1 つです。もう一つは、$(*num*)で、これは 反対の変換で、数値から文字列に変換するのに使います。

8.3.2. 文字列レジスタを使う

このサンプルは、文字列レジスタにある文字列を数値に変換する方法を示しています。

```
// 上記に似ているが文字列レジスタを使って実行
string myString$ = "456";
// クォート無しでの割り当ては右側を評価
%A = myString$;
// %A は置換され、右側が評価される
int aa = %A;
// 'aa' は他の整数値で操作可能
int bb = aa + 100;
bb=;                    // ANS（答え）:556
```

8.4. 数値を文字列に変換

次は、数値の桁数と小数点以下の桁数のフォーマットの表記を含んで、数値変数を文字列に変換するサンプルです。

内容

8.4.1. 数値を文字列に変換する

置換表記を使う

さまざまな数値の型 (double, int, or const) を文字列に変換するには、以下のサンプルを参考にして下さい。

```
// myNum は整数値 456 を含む
int myNum = 456;
// myNumString は文字 456 を文字列として含む
string myNumString$ = $(myNum);
```

シンタックス **$(num)** は、LabTalk でサポートされている 2 つの 置換表記 の 1 つです。もう 1 つの **%(string$)**は、文字列変数をその内容で置換する逆の変換(文字列から数値)を行うのに使われます。

フォーマットも型変換中に指定することができます。

```
$(number [,format])     // 角括弧はフォーマットが任意であることを示す
```

フォーマットは、C 言語のフォーマット指定子変換に従います。これは、どの C 言語参照でも見つけることができます。例えば

```
string myNumString2$ = $("3.14159",%3d);
myNumString2$=                              // "3"
```

```
string myNumString2$ = $("3.14159",%3.2f);
myNumString2$=                              // "3.14"

string myNumString2$ = $("3141.59",%6.4e);
myNumString2$=                              // "3.1416e+003"
```

フォーマットのこの型についての情報は、$()置換を参照して下さい。

Format 関数を使う

数値変数を文字列変数に変換する別の方法は、**format** 関数を使うことです。

```
// format を呼び出し、有効桁数 3 を指定
string yy$=Format(2.01232, "*3")$;
// "2.01"
yy$=;
```

format 関数についての詳細な情報については、Format (関数)をご覧下さい。

8.4.2. 有効桁数、小数点桁数、数値のフォーマット

LabTalk には、独自のフォーマット指定子があり、これは LabTalk の置換表記の一部として使われ、数値を書式化する簡単な方法です。

有効桁数を設定するのに＊表記を使う

```
x = 1.23456;
type "x = $(x, .2)";
```

この例では、変数 x の後に、有効数字 2 桁を設定する *2 が続いています。出力結果は次のようになります。

```
x = 1.2
```

さらに、")"の前に * を置くと、10 のべき乗の前の 0 を切り取ります。例えば

```
y = 1.10001;
type "y = $(y, *4*)";
```

この例では、出力結果は次のようになります。

```
y = 1.1
```

結果は、y が*4 ではなく*4* に従うので、有効桁数が 2 となります。

小数点位置を設定するのに．表記を使う

```
x = 1.23456;
type "x = $(x, .2)";
```

この例では、変数 x の後に、小数点以下 2 桁を設定する .2 が続いています。出力結果は次のようになります。

```
x = 1.23
```

記号 E を使って変数を工学表記に変更する

記号 E は記号*と同じように、書式を変更する変数に続けて入力します。例えば、

```
x = 1e6;
type "x = $(x, E%4.2f)";
```

上記 type のコマンド内において、%は置換記号の開始を意味し、4 は全桁数を、.2 は小数点以下の桁数 2 桁を指定しており、f は浮動小数記号の表記です。出力は次のようになります。

```
x = 1.00M
```

$(x, S*n)を使って、工学表記から科学(指数)表記に変換する

このシンタックスでは、*n* は全桁数を指定します。

```
x = 1.23456;
type "x = $(x,S*3)";
```

Origin は次のように返します。

```
x = 1.23E0
```

8.5. 文字列配列

これは、文字列配列を作成し、そこに要素を追加し、ソートし、配列の内容を一覧にする方法を示しています。

```
// 既存のサンプルファイルをインポート
newbook;
fpath$ = "Samples\Data Manipulation\US Metropolitan Area Population.dat"
string fname$ = system.path.program$ + fpath$;
impasc;

// 最後の列をループし、すべての州のデータを探す
range rMetro=4;
stringarray saStates;
for( int ir=1; ir<=rMetro.GetSize(); ir++ )
{
        string strCell$ = rMetro[ir]$;
        string strState$ = strCell.GetToken(2,',')$;
        // 名前に '-' のインスタンスを見つける
        int nn = strState.GetNumTokens("-");
        // States 文字列配列に追加
        for( int ii=1; ii<=nn; ii++ )
        {
                string str$ = strState.GetToken(ii, '-')$;
                // すでに存在していなければ追加
                int nFind = saStates.Find(str$);
                if( nFind < 1 )
                        saStates.Add(str$);
        }
}
```

```
// States 文字列配列をソートし、出力
saStates.Sort();
for(int ii=1; ii<=saStates.GetSize(); ii++)
        saStates.GetAt(ii)$=;
```

9 ワークブック、ワークシート、ワークシート列

この章ではワークブック-> ワークシート-> 列の階級を説明し、これらのオブジェクトにスクリプトからアクセスする方法を紹介します。ワークシートのデータを、*仮想行列*として扱うコンセプトについても説明します。

この章で以下の項目を説明します

- ワークブック
- ワークシート
- ワークシート列

9.1. ワークブック

この章で以下の項目を説明します

- ワークブックの基本操作
- ワークブックの操作

9.1.1. ワークブックの基本操作

ワークブックは Page オブジェクトと Window コマンドで操作できます。また、データ操作用の X ファンクションを使うこともできます。これらのツールを使用すると、新しいワークブックの作成、ワークブックの複製、ワークブックをテンプレートとして保存等を行えます。実践的なサンプルは以下の通りです。

内容

新しいワークブックを作成する

newbook X ファンクションを使うと、新しいワークブックを作成できます。この X ファンクションの引数として、新たに作成するワークブックのロングネーム、シート数、使用するテンプレート、非表示にするか等を選択できます。

```
//ロングネーム"MyResultBook"の新しいワークブックを作成
newbook MyResultBook;

// 3 つのワークシートを持つ新しいワークブックを作成
// ロングネームとショートネームに "MyData" を使用
newbook name:="MyData" sheet:=3 option:=lsname;

// 非表示の新しいワークブックを作成
// ワークブックの名前は myBkName$ 変数に格納
newbook hidden:=1 result:=myBkName$;
// ワークブック名を出力
myBkName$ = ;

// デフォルトでは、ビルトインテンプレートである"Origin"が使われる
// 新しいワークブックを作成すると特定のテンプレートを使用可能
// XYZ テンプレートを使用して、新しいワークブックを作成
newbook template:=XYZ;
```

win -ti コマンドはテンプレートファイルから最小化した新しいワークブックを作成できます。

```
//  FFT テンプレートを使用して新しいワークブックを作成
// ロングネームとショートネームを MyFFT に設定し、最小化
win -ti wks FFT MyFFT;
```

ワークブックを開く

ワークブックのデータが拡張子 ogw のファイルとして保存されている場合、doc -o コマンドで開く事ができます。

```
// 開くワークブックのパス
string strName$ = system.path.program$;
strName$ += "Samples\Graphing\Automobile Data.ogw";
// ワークブックを開く
doc -o %(strName$);
```

ワークブックを保存する

Origin はワークブックのデータをファイル（拡張子 .ogw）として保存、データなしのテンプレートで保存（拡張子 .otw）、分析を行ったワークブックに関しては分析テンプレート（拡張子 .ogw）として保存できます。

1. save -i コマンドはアクティブなワークブックを ogw ファイルとして保存できます。

```
// 新しいワークブックを作成
newbook;
// col(1)に適当なデータを入力
col(1) = uniform(32);
// このデータ付きのワークブックをユーザファイルフォルダ内に MyData.ogw
として保存
```

```
save -i %YMyData.ogw;
```

2. template_saveas X ファンクションはワークブックをテンプレートとして保存する時に使用します。

```
// 3 つのシートを持った新しいワークブックを作成
newbook sheet:=3;
// このワークブックをテンプレートとして My3SheetsBook という名前で
// ユーザファイルフォルダ（デフォルト）に保存
template_saveas template:=My3SheetsBook;
```

3. 行った分析と共にワークブックを保存するには、save -ik を使用します。

```
// プロジェクトを作成
string strOpj$ = system.path.program$ +
"Samples\Analysis.opj";
doc -o %(strOpj$);

// 分析テンプレートとしてワークブックを保存するために、ワークブックをアク
ティブにする
win -a Book1J;

// このワークブックを分析テンプレートとして
// MyAnalysis.ogw というファイル名でユーザファイルフォルダに保存
save -ik %YMyAnalysis.ogw;
```

ワークブックを閉じる

ワークブックを閉じるには、ワークブックの右上にある閉じるボタンをクリックするだけです。この挙動は win -ca コマンドで実行でき、ダイアログがポップアップしてユーザに削除か非表示を選択するように求めます。

```
// ワークブックを作成し、名前は変数 MyBook$ に格納
newbook result:=MyBook$;

// 閉じるボタンをクリックする様子をシミュレートする
win -ca %(MyBook$);
```

ポップアップをせずにワークブックを閉じて全てのデータを削除するには win -cd コマンドを使用できます。この操作は以下にあるワークブックを削除する項目と同じです。

```
// 閉じるために新しいワークブックを作成
newbook;

// ポップアップなしでワークブックを閉じて全てのデータを削除
win -cd %H;
```

ワークブックの表示と非表示

win コマンドには 3 つのスイッチ（ *-ch*, *-h*, *-hc*）があり、ワークブックの表示と非表示を切り替える事ができます。

```
// 非表示にするために 3 つのワークブックを作成
newbook name:=MyBook1 option:=lsname;  // 1 番目のワークブック、 MyBook1
newbook name:=MyBook2 option:=lsname;  // 2 つ目のワークブック、MyBook2
newbook name:=MyBook3 option:=lsname;  // 3 つ目のワークブック、 MyBook3;

//  -ch を使ってアクティブなワークブック、MyBook3 を非表示にする
// プロジェクト・エクスプローラの表示モードは非表示
win -ch 1;

// -hc を使用して 1 番目のワークブック（アクティブではありません）// MyBook1 を非表示にする
// プロジェクト・エクスプローラの表示モードは非表示
win -hc 1 MyBook1;

// -h を選択して 2 番目のワークブック（アクティブなワークブック）// MyBook2 を非表示にする
// プロジェクト・エクスプローラの表示モードはまだ通常通り
win -h 1;

// 実際、 MyBook2 はまだアクティブなワークブックなので、
// 以下のようにすると表示可能
win -h 0;

// MyBook1 と MyBook3 を表示するには、-hc スイッチを使用して、
// ワークブック名を指定する必要がある
win -hc 0 MyBook1;
win -hc 0 MyBook3;
```

ワークブックに名前とラベルを付ける

ワークブックにはショートネーム、ロングネーム、コメントがあります。ワークブックの名前（ショートネーム）を変更するには win -r コマンドを使用し、page オブジェクトを使用すると、ワークブックのタイトル（ショートネーム、ロングネーム、両方）表示を含む、ロングネームとコメントを制御できます。

```
// "Data"と名前がつけた新しいワークブックを作成
// 両方の名前をタイトルに表示
// ショートネームとロングネームは同じです
// ワークブックのタイトルはショートネームのみを表示
newbook name:=Data option:=lsname;

// ワークブックの名前を"RawData"に変更
win -r Data RawData;

// ロングネームを "FFT Data"に変更
page.longname$ = "FFT Data";
// コメント"1st group data for fft"を追加
page.comments$ = "1st group data for fft";

// ワークブックのタイトルにロングネームのみを表示するように変更
page.title = 1;  // 1 = ロングネーム、 2 = ショートネーム、3 = 両方
```

ワークブックをアクティブにする

ワークブックをアクティブにするには、win -a を使用します。

```
// 開くプロジェクトのパスを表示
string strOpj$ = system.path.program$;
strOpj$ += "Samples\Curve Fitting\Intro to Nonlinear Curve Fit Tool.opj";
// プロジェクトを開く
doc -o %(strOpj$);

// プロジェクトの 2 番目のサブフォルダにある Book1 をアクティブにする
win -a Book1;

// ワークブック名を変数に入れる事も可能
// プロジェクトにワークブック Gaussian の名前を入れる
string strGau$ = Gaussian;
// 1 番目のサブフォルダにある Gauusian ワークブックをアクティブにする
win -a %(strGau$);
```

ほとんどの Origin コマンドはアクティブウィンドウに対して操作しますので、*win -a* コマンドを使って、ワークブックをアクティブにし、スクリプトを実行するとアクティブなワークブックになると思い、実行する事もあるでしょう。これは、簡単なコードで動作しますが、長いスクリプトに対しては、タイミングの問題があるかもしれません。代わりに、window -o *winName* {*script*} を使うことをお勧めします。詳細は、スクリプトが操作するオブジェクトについての注意をご覧下さい。

ワークブックを削除する

ワークブックを削除するには win -c コマンドを使用します。このコマンドはポップアップを表示せずに、直接ウィンドウを削除します。

```
// プロジェクトを開くパス
string strOpj$ = system.path.program$ + "Samples\Curve Fitting\2D Bin and
Fit.opj";
// プロジェクトを開く
doc -o %(strOpj$);
// ワークブック Book1 プロジェクトから削除
win -c Book1;

// アクティブなワークブックを削除する場合、ワークブック名は無視可能
// または、 %H を使用してワークブック名を参照
win -a MatrixFit1;  // ワークブック MatrixFit1 をアクティブにする
win -c;  // ワークブックを削除
// または、次を使用
// win -c %H;

// これは、変数にワークブック名が格納されているワークブックも削除可能
// newbook X-Function を使用して新しいワークブックを作成
// このワークブックの名前は文字列変数 ToDel$ に格納される
newbook result:=ToDel$;
// いま作成したワークブックを削除
win -c %(ToDel$);
```

9.1.2. ワークブックの操作

Origin は分割、統合、複製など、LabTalk スクリプトを使用してブックを操作するための機能を提供しています。

ワークブックの複製

アクティブなワークブックを複製するために、win -d コマンドを使用します。複製されたワークブックの名前指定が可能で、新たなワークブックは複製後アクティブになります。コマンド win -da は同じように動作しますが、複製後もアクティブなウィンドウは変わりません。

```
// プロジェクトを開く
string strOpj$ = system.path.program$;
strOpj$ += "Samples\LabTalk Script Examples\Loop_wks.opj";

doc -o %(strOpj$);

// ワークブック S2Freq1 をアクティブにする
win -a S2Freq1;

// このワークブックを複製して、名前を"MyCopy"とし、
// これをアクティブウィンドウにする
win -d MyCopy;

// ワークブック MyCopy を複製し、名前を"MyCopy2"にする
// アクティブウィンドウは MyCopy のままにする
win -da MyCopy2;
```

ワークブックを統合

複数のワークブックをひとつのワークブックに統合するには、X ファンクション merge_book を使用します。

```
// プロジェクトを開く
string strOpj$ = system.path.program$;
strOpj$ += "Samples\LabTalk Script Examples\Loop_wks.opj";

doc -o %(strOpj$);

// Sample1 フォルダをアクティブにする
pe_cd /Sample1;

// 2 つのサブフォルダ内にある、2 つのワークブック(S1Freq1 と S1Freq2)を統合する
// 統合されたブック内のワークシート名には、元のワークブック名を使用する
merge_book fld:=recursive rename:=sname;

// Sample2 フォルダをアクティブにする
pe_cd /Sample2;

// 2 つのサブフォルダ内にある、2 つのワークブック(S2Freq1 と S2Freq2)を統合する
// 統合されたブック内のワークシート名には、元のワークブック名を使用する
merge_book fld:=recursive rename:=sname;

// root フォルダをアクティブにする
pe_cd /;
```

```
// 上記スクリプトで作成された 2 つのワークブックは
// それぞれ"mergebook"という名前をはじめに持つ
// これら 2 つのワークブックを統合し、1 つのワークブックにまとめる
// 最終的得られるワークシートには、
// 元のワークシート名を付ける
merge_book fld:=project single:=0 match:=wkbshort key:="mergebook*"
rename:=wksname;
```

ワークブックの分割

上記のサンプルでは複数のワークブックを統合して一つにまとめましたが、ワークブックを分割し、単一ワークシートを持つ複数のブックにすることができます。この場合には、X ファンクション wsplit_book を使用します。

```
// プロジェクトを開く
string strOpj$ = system.path.program$;
strOpj$ += "Samples\COM Server and Client\Basic Stats on Data.opj";

doc -o %(strOpj$);

// アクティブなワークブック、RawData には 3 つのワークシートがあり、
// このブックを分割し、3 つのブックを作成する
// それぞれのワークブックには、元のワークブックからワークシートを 1 つずつ格納する
wsplit_book fld:=active;
```

9.2. ワークシート

このセクションで以下の項目を説明します

- ワークシートの基本操作
- ワークシートデータの操作
- ワークシートから行列に変換
- 仮想行列

9.2.1. ワークシートの基本操作

ワークブックへのシート追加や、ワークシートのアクティブ化、ワークシートプロパティの設定、ワークシート削除などが基本的なワークシート操作に含まれます。これらの操作は、データ操作用の X ファンクションとあわせ、Page および Wks オブジェクトを使用して実行します。以下にいくつか実践的なサンプルがあります。

内容

新しいワークシートを追加する

newsheet X ファンクションは、新しいシートを追加するのに使います。

```
// 3 つのワークシートを持つワークブックを作成
// そしてロングネームとショートネームを"mydata"にする
newbook name:="mydata" sheet:=3 option:=lsname;
// 4 列持つ "source" という名前のシートを現在のワークブックに追加
newsheet name:=source cols:=4;
```

ワークシートをアクティブにする

ワークブックは Origin のオブジェクトで列を含むワークシートで構成されています。ブック内のワークシートは、page にある内部的なレイヤです。言い換えれば、ワークシートはレイヤオブジェクトから派生し、ワークブックはページオブジェクトから派生したものです。ページのアクティブレイヤは、*page.active* または *page.active$* プロパティで表され、ワークシートをアクティブにするのに使用されます。

```
// 4 つのワークシートをもつ新たなブックを作成
newbook sheet:=4;

page.active = 2;       // インデックスでワークシートをアクティブ
page.active$ = sheet3;        // 名前でワークシートをアクティブ
```

ワークシートプロパティを修正する

ワークシートオブジェクトを使う

ワークシートがアクティブなとき、**wks.=** と入力し、Enter キーを押して、すべてのワークシートプロパティを一覧表示します。これらのプロパティのほとんどは書き込み可能で、直接修正することができます。例えば、

```
// ワークシートの名前を変更
wks.name$ = Raw Data
// 列の数を 4 にセット
wks.ncols = 4
// 列幅を 8 文字に修正
wks.colwidth = 8
// ワークシートヘッダの最初のユーザ定義パラメータを表示
wks.userparam1 = 1
```

wks.maxRows と wks.nRows の 2 つのプロパティは似ています。前者は、ワークシート内で値がある最も大きな行インデックス、後者はワークシートの行数の読み書きを行います。次のスクリプトで違いを見ることができます。

```
newbook;  // 新しいワークブックを作成
col(b) = {1:10};  // 列 B の第 1 セルから 10 セルまでに、1 から 10 を入力
wks.maxRows = ;
```

```
wks.nRows = ;
```

Origin は、wks.maxRows に対して 10 を出力し、wks.nRows に対しては 32 を出力します。

もし、ワークシートがアクティブではない場合、**wks** オブジェクトの前に、ワークシート名（またはワークブック名）を指定するために以下のシンタックスを使用します。

[*WorkbookName*]*WorksheetNameOrIndex*!wks

または、ワークシートの range 変数を使用できます。例えば、

```
// プロジェクトを開く
string strOpj$ = system.path.program$;
strOpj$ += "Samples\COM Server and Client\Basic Stats on Data.opj";
doc -o %(strOpj$);

wks.nCols = ;  // アクティブなワークシートの列数を出力
// ワークシート[RawData]Data!の列数を出力
[RawData]Data!wks.nCols = ;
// RawData ワークブックの第 1 ワークシートの名前を出力
[RawData]1!wks.name$ = ;

// range を使用
range rWks = [RawData]Data!;  // RawData ワークブックの Data シートのための range 変数
rWks.userparam1 = 1;  // ワークシートの第一ユーザパラメータを表示
```

X ファンクションを使用する

wks オブジェクト以外に、X ファンクションを使って、ワークシートプロパティを修正することができます。これらの X ファンクションの名前は、通常、"w"で始まります。wcolwidth, wcellformat, wclear などになります。wks.colwidth を使わずに、下記のように列のサイズを変更することもできます。

```
wcolwidth 2 10; // 2 番目の列幅を 10 にセット
```

ワークシートを削除する

layer -d コマンドは、ワークシートまたはグラフレイヤを削除するのに使用します。

```
// 6 つのワークシートをもつワークブックを作成し、
// ワークブック名を MyBook$とする
// 最初のワークシートがアクティブになる
newbook sheet:=6 result:=MyBook$;

// 新しくワークシートを作成し、名前を "My Sheet"とする
newsheet name:="My Sheet";

page.active = 1;  // 最初のワークシートをアクティブにする
layer -d;  // アクティブワークシートの削除

// インデックスによるワークシートの削除
// アクティブワークブック（またはグラフ）内の 3 番目のシート（またはレイヤ）を削除
layer -d 3;

// ワークシート名で削除
```

```
layer -d "Sheet5";

// range による特定ワークシートの削除
range rs = [%(MyBook$)]"My Sheet"!;  // 特定シートの range を定義
layer -d rs;

// 文字列変数に保存された名称のワークシートの削除
string strSheet$ = "Sheet3";
layer -d %(strSheet$);
```

文字列変数として保存された名称のワークシートを削除するには、

```
//__report$ には Origin が作成した最後のレポートシートの名前を保持
layer -d %(__report$);
```

変数 **__report$** は、特定のオブジェクトの最後に使ったインスタンスを記録するシステム作成の文字列変数の例です。このような変数の一覧は、リファレンステーブルにあります。

9.2.2. ワークシートデータの操作

このセクションでは、基本的なデータ処理の X ファンクションのサンプルを説明します。ワークシートデータに直接アクセスするには、範囲表記をご覧下さい。ワークシート

内容

ワークシートデータをコピーする

ワークシートをコピーする

wcopy X ファンクションは、指定したワークシートの コピー を作成するのに使用します。

次のサンプルは、現在のワークシートを複製し、コピーしたワークシートをもつ新しいブックを作成します。

```
wcopy 1! [<new>]1!;
```

セルの範囲をコピーする

wrcopy X ファンクションは、あるワークシートから別のワークシートにセル範囲をコピーするのに使用します。目的のワークシートのロングネームとして使用する元の行を指定できます。

次のスクリプトは、[book1]sheet1! の 5 から 9 行目を Book1 の CopiedValues というワークシートにコピーし(ワークシートが存在しなければ作成されます)、[book1]sheet1! の 4 行目の値を目的のワークシート[book1]CopiedValues!のロングネームに割り当てます。

```
wrcopy iw:=[book1]sheet1! r1:=5 r2:=10 name:=4 ow:=CopiedValues!;
```

列や行列オブジェクトのコピーは、列をコピーする と 行列をコピーする をご覧ください。

ワークシートデータを削減する

ワークシートデータの削減ワークシート、削減

Origin は、 reduce_ex, reducedup, reducerows, reducexy のようなデータ削減の X ファンクションがあります。これらの X ファンクションは、大きなデータセットから小さなデータセットを作成する異なる方法を提供します。どれを選択するかは、持っている入力データの種類や出力したいデータの種類に依存します。

サンプル

次のスクリプトは、Y 値が複数 X 値のそれぞれに対して平均値となるよう新しく XY 列を作成します。

```
reducedup col(B);
```

次のスクリプトは、3 を係数として、アクティブな選択(XY のプロット属性とは関係なく、複数列やワークシート全体にできます。)を削減します。これは、行 2,3、行 5,6 を削除し、行 1,4 を残します。デフォルトで、削減された値は、新しいワークシートに入れられます。

```
reducerows npts:=3;
```

次のスクリプトは列 A の *n* 個毎の数字(下記例では 5 個)の平均を取り、各グループの平均を列 B に出力します。これは ave LabTalk 関数と同じで、col(b)=ave(col(a),5)のように書くことができます。

```
reducerows irng:=col(A) npts:=5 method:=ave rd:=col(b);
```

ワークシートデータの抽出

ワークシート、データの抽出抽出、ワークシートデータ

wxt X ファンクションを使って、データ列を含む条件でワークシートから部分データを抽出することができます。

```
// サンプルデータファイルのインポート
newbook;
```

```
string fname$ = system.path.program$ + "samples\statistics\automobile.dat";
impasc;
// 列のいくつかを使って範囲を定義
range rYear=1, rMake=2, rHP=3;
type "Number of rows in raw data sheet= $(rYear.GetSize())";
// 条件文字列を定義し、同じブック内の
// 新しく名前を付けたシートにデータを抽出
string strCond$="rYear >= 1996 and rHP<70 and rHP>60 and rMake[i]$=Honda";
wxt test:=strCond$ ow:="Extracted Rows"! num:=nExtRows;
type "Number of rows extracted = $(nExtRows)";
```

新しいブックに出力する

次の行を変更して、既存のワークブックの新しいシートではなく、新しいワークブックに直接出力できます。

```
wxt test:=strCond$ ow:=[<new name:="Result">]"Extracted"! num:=nExtRows;
```

見て分かるように、前のコードとの違いは、**<new>** キーワードを持つ変数 **ow** の範囲表記のワークブックの部分を追加しています。(<new>修飾語、オプション、テンプレート、名前などにリンクとインデックスを表します。)

ワイルドカードを使って検索する

LabTalk は、文字列の比較に、ワイルドカードとして* と？文字を使います。以下のように、**strCond** を変更することができます。

```
string strCond$ = "rYear >= 1996 and rHP<70 and rHP>60 and rMake[i]$=*o*";
```

文字 **o** を使って、自動車のメーカすべてを表示できます。

ワークシートデータを削除する

ワークシートデータを削除する

N 番目の行を削除するには、上記で述べたように、**reducerows** X ファンクションを使って行います。

このサンプルでは、for ループを使ってワークシート内の *N* 番目の列毎に削除する方法を示しています。

```
int ndel = 3; // 必要に応じてこの数字を変更
int ncols = wks.ncols;
int nlast = ncols - mod(ncols, ndel);
// 右から左に削除する必要がある
for(int ii = nlast; ii > 0; ii -= ndel)
{
   delete wcol($(ii));
}
```

ワークシートをソートする

ワークシートをソートするワークシート、ソート

次のサンプルでは、**wsort** X ファンクションを使って、ワークシート内のデータをネストソートする方法を示しています。

```
// 新しくブックを作成し、サンプルファイルをインポート
newbook;
string fname$ = system.path.program$ + "Samples\Statistics\automobile.dat";
impasc;
```

```
// ベクターデータをセットアップして、列のネストとソート順を指定
// ソートネスト: 主キー 列 2, 次に col 1, その次に col 3
dataset dsCols = {2, 1, 3};

// col2 を昇順、col 1 を昇順、col 3 を降順にソート
```

ワークシートを分割する

X ファンクション wsplit は、1 つのワークシートの列を複数シートに分割するのに使用されます。

以下のサンプルは、複数の CSV ファイルをインポートし、全てのデータファイル内にある Amplitude データをワークシートに出力します。さらに、等高線図の作図のためにこれらを統合し、行列として出力します。

```
// 新しいワークブックを作成
newbook;

// 特定フォルダ内の全ての CSV ファイルを探す
string strPath$ = system.path.program$ + "Samples\Batch Processing\";
findfiles path:=strPath$ fname:=csvFiles$ ext:=csv;

// 見つかった CSV ファイルを、1 つのワークシートにインポートする
impCSV  fname:=csvFiles$  // 見つかったすべての CSV ファイル
        options.Mode:=1  // 第 2 ファイルから、新たな列を開始
        options.names.FNameToSht:=0  // ワークシート名の変更なし
        options.names.FNameToBk:=0  // ワークブック名の変更なし
        options.HeaderLines.SubHeaderLines:=2  // 2 行のサブヘッダオプション
        options.HeaderLines.LongNames:=1  // 第 1 サブヘッダ行はロングネーム
        options.HeaderLines.Units:=2;  // 第 2 サブヘッダ行は単位

// ロングネームによりワークシートを分割
// 同じロングネームの列を同一ワークシートに出力する
// すべての結果ワークシートを新たなワークブックひとつに出力
wsplit mode:=label label:=L;

// ワークシート Amplitude をアクティブにする
page.active$ = Amplitude;

// ワークシートデータを行列に直接変換する
w2m;

// Amplitude データの等高線図を作成する
worksheet -p 226 contour;
```

カテゴリーデータをアンスタック/スタックする

アンスタックデータスタック、データ

ワークシート列をアンスタックする

分析およびプロットを行う際に、カテゴリーデータをアンスタックすることが望ましい場合があります。スクリプトからこれを実行するには、wunstackcol X ファンクションが最も便利な方法です。

この例では、カテゴリーデータがインポートされ、入力範囲を **irng2** で指定した特定のカテゴリーでデータをアンスタックします。(カテゴリーで)表示されるデータは、入力範囲 **irng1** で参照されます。この例では、列の範囲が直接入力となりますが、範囲変数が使われます。

```
// 自動車データをインポート
newbook;
string fpath$ = "\Samples\Statistics\Automobile.dat";
string fname$ = system.path.program$ + fpath$;
impasc;

// 列 2 に保存されている自動車の Make を使って他のすべての列をアンスタック
// 出力シートのコメント行に "Make" を配置
wunstackcol irng1:=(1, 3:7) irng2:=2 label:="Comments";
```

結果はアンスタックデータを持つ新しいワークシートです。

ワークシート列をスタックする

カテゴリーデータをスタックする操作は、アンスタック操作の逆を行うようなものです。元のデータセットにおいて、異なるグループに属したサンプルが、別々の列に格納されています。スタッキング後、サンプルはグループ情報を提供する追加の列とともに、同じ列の異なる行に格納されます。wstackcol を使用してワークシート列をスタックします。

以下のサンプルではまず、カテゴリーデータの入力されたワークブックを開きます。第 1 ワークシートがアクティブな状態で B,C,D 列を行でスタックし、A 列はその他の列として含めます。

```
// ワークブックを開く
string strBook$ = system.path.program$;
strBook$ += "Samples\Statistics\Body.ogw";
doc -o %(strBook$);

// ワークシート Male の B,C,D 列をスタック
// A 列を他の列として含める
// メソッドは行ごと
wstackcol irng:=(2:4) tr.identifiers:={L} include:=1 method:=1;
```

結果は、ソースと同じブック内にスタックデータを持った新しいワークシートとして出力されます。

ピボットテーブル

X ファンクション wpivot は、データ概要の把握、分析、比較、データ間の関係確認を素早く行うことがきます。これによりデータ情報を簡単に把握できます。

```
// 新しいブックを作成
// データファイルをインポート
newbook;
fname$ = system.path.program$ + "Samples\Statistics\HouseholdCareSamples.xls";
impExcel lname:=1;

// ワークシート"HQ Family Mart"がアクティブなことを確認
// このワークシートのコピーを作成
page.active$ = "HQ Family Mart";
wcopy ow:=[<new>]"HQ Family Mart"!;

// ピボットテーブルの行のソースは Make、
```

```
// 列ソースは Brand、データは Number in shelf
// 結果として、異なるブランドと型式ごとに
// 棚の製品数が出力される
wpivot  row:=col(D) col:=col(F) data:=col(K)
        method:=sum total:=1 sort_total:=no sum:=1;

// ソースデータシートをアクティブにする
page.active$ = "HQ Family Mart";

// 行のソース、列のソース、データ列が同じピポッドテーブル
// 合計の 10%より小さい値の場合、
// それらを結合
// 結果シートでは、列の情報をユーザ定義パラメータとして格納
// 行の名前は、ソースシートの列ロングネームを使用
wpivot  row:=col(D)
        col:=col(F)
        data:=col(K)
        method:=sum total:=1 sort_total:=no sum:=1
        dir:=col threshold:=10  // 列を越えて 10%より小さい値の場合、
// それらを結合
        // 列の情報（列ロングネーム）をユーザ定義パラメータ行に出力
        pos:=udl udlabel:=L;
```

ワークシートフィルタ

Origin のワークシートフィルタ (データフィルタ) は、列ベースのフィルタで、特定条件により行の削減が可能です。そして、削減されたデータは分析やグラフからは除かれます。3 つのデータフォーマット（数値、テキスト、日付/時間）をサポートしています。LabTalk では、wks.col (wks.col.filter, wks.col.filter$, wks.col.filterenabled, wks.col.filterprescript$, と wks.col.filterx$) オブジェクトにより、データフィルタの制御が可能です。フィルタの実行/再適用には、wks.runfilter() メソッドを使用します。

```
// ワークブックを作成し、データをインポート
newbook;
string fname$ = system.path.program$ + "Samples\Statistics\Automobile.dat";
impasc;

// 1 列目にデータフィルタをセット（数値）
wks.col1.filter = 1;  // フィルタ追加
wks.col1.filterx$ = year;  // 既存の第 1 列を変数 "year"にセット
// フィルタ条件を、1995 から 2000 の間に設定
wks.col1.filter$ = "year.between(1995,2000)";

// 2 列目にデータフィルタをセット（テキスト）
wks.col2.filter = 1;  // Add filter
wks.col2.filterx$ = make;  // 既存の第 2 列を変数 "make" としてセット
// 事前スクリプトをセット
wks.col2.filterprescript$ = "string strFavorite$ = GMC";
wks.col2.filter$ = "make = strFavorite$";  // フィルタクエリ文字列をセット

// ワークシートフィルタを実行
wks.runfilter();

// 1 列目のフィルタを無効化
```

```
wks.col1.filterenabled = 0;

// フィルタを再適用
wks.runfilter();
```

ワークシート内にフィルタがあるかどうかを検出するには、wks.hasfilter()メソッドを使用することができます。

```
// ワークシートフィルタがある場合 1 を返し、そうでない場合は 0 を返す
wks.hasfilter() = ;
```

9.2.3. ワークシートから行列に変換

分析またはグラフ作成時に、ワークシートから行列に変換またはその反対の操作をして、データを再構成する必要があるかもしれません。このページはワークシートを行列に変換する時のサンプルと情報を紹介します。なお、反対の操作をする際は行列をワークシートに変換するを確認してください。

ワークシートから行列

ワークシートに含まれるデータは、一連の グリッディング X ファンクションを使って行列に変換することができます。

w2m X ファンクションは行列の形式で入力されているワークシートを行列に変換します。元のワークシートのデータには、最初の列、最初の行、ヘッダ行に X または Y 座標値を含めることができます。しかし、行列の座標は等間隔でなければならいので、元のワークシートに等間隔な X/Y 値が入力されている必要があります。

X/Y 座標値が等間隔でなければ、行列に変換するのではなく、仮想行列 を使用してください。

次のサンプルはワークシートから直接変換する方法を示しています。

```
// 新しいワークシートを作成
newbook;
// サンプルデータをインポート
string fname$ = system.path.program$ +
       "\samples\Matrix Conversion and Gridding\DirectXY.dat";
impasc;
// ワークシートを行列に変換、最初の行が X で最初の列が Y
w2m xy:=xcol xlabel:=row1 ycol:=1;
// 行列ウィンドウに X/Y 値を表示
page.cntrl = 2;
```

ワークシートデータが XYZ 列形式で構成されているとき、グリッディング法を使ってこのようなデータを行列に変換します。多くのグリッディング法が利用でき、これにより元のデータを補間し、指定した次数での等間隔な XY で、値の配列を生成します。

次のサンプルは、Renka-Cline グリッディング法で XYZ ワークシートデータを変換し、新しい行列から 3D グラフを作成します。

```
// シート無しで新しいワークシートを作成
newbook;
// サンプルデータをインポート
string fname$ = system.path.program$ +
       "\samples\Matrix Conversion and Gridding\XYZ Random Gaussian.dat";
impasc;
```

```
// ワークシートデータを Renka-Cline グリッディング法で 20 x 20 の行列に変換
xyz_renka 3 20 20;
// 3D カラーマップグラフを作成
worksheet -p 242 cmap;
```

9.2.4. 仮想行列

ワークシートセルのグループとして配置されたデータを行列として扱うことができ、3D 曲面, 3D 棒グラフ, 等高線などのさまざまなプロットをこのようなデータから作成できます。この機能については、「仮想行列」を参照してください。X および Y 座標の値は、任意で最初の列と最初の行のデータブロックに含めることができます。行はワークシートのヘッダ行にすることができます。

Origin 内の行列オブジェクトは XY 座標の線形マッピングのみをサポートしていますが、仮想行列は非線形または不等間隔の XY 座標をサポートしています。

ワークシートにあるデータを使ってプロットを作成するとき、仮想行列が定義されます。plotvm X ファンクションを使ってプロットを作成します。

次のサンプルは、X ファンクション plotvm の使用例です。

```
// 新しいワークブックを作成し、サンプルデータをインポート
newbook;
string fname$=system.path.program$ + "Samples\Graphing\VSurface 1.dat";
impasc;
// シート全体を仮想行列として扱い、カラーマップ曲面を作成
plotvm irng:=1! format:=xacross rowpos:=selrow1 colpos:=selcol1
 ztitle:="VSurface 1" type:=242 ogl:=<new template:=cmap>;
// X 軸スケールを対数に変更
layer.x.type=2;
```

9.3. ワークシート列

この章で以下の項目を説明します

- 基本ワークシート列操作
- ワークシート列データの操作
- 日付と時間データのフィット

9.3.1. 基本ワークシート列操作

ワークシート列に操作を行う場合、ほとんどの状況で wks.col オブジェクトを使用できます。または、範囲表記を列オブジェクトに指定して操作します。

内容

列の追加・挿入

ワークシートの最後に列を追加するには、wks.addCol()方法を使用します。これは指定の名前を有する列を追加し、その名前が使用済みまたは無視された場合、追加される列には一般的な名前が選択されます。

```
// 新しいワークブックを作成
newbook;

// Result という名前を指定し、新しい列を最後に追加
wks.addCol(Result);
```

上記方法は最後に 1 列ずつ追加する事しかできません。複数の列を追加する場合、wks.nCols プロパティを使用してワークシートの列数を指定する事で列を追加できます。例えば、以下のスクリプトはアクティブなワークシートに 3 列を追加し、一般的な名前を付けます。(Note:この方法で名前を指定する事はできません。以下の**列に名前とラベルをつける**を参照してください。)

```
// 新しいワークブックを作成
newbook;

// ワークシートの最後に 3 列分追加
wks.nCols = wks.nCols + 3;
```

ワークシートの最後に列を追加する他に、現在の列の前に指定した数の列を挿入する事もできます。まず、wks.col を使用して現在の列を指定します。そして wks.insert()メソッドを使用して現在の列の前に 新たな列を挿入します。この方法では、それぞれの列の名前をスペース区切りでリストする必要があります。

```
// 新しいワークブックを作成
newbook;

// 列 2 を現在の列として設定
wks.col = 2;

// 列 2 の前に、名前を指定した列を 3 つ挿入
wks.insert(DataX DataY Result);
```

列の移動

colmove X ファクションは、ワークシート内のデータ列を移動するのに使用します。(範囲変数とは対照的に)明示的な範囲と入力としての移動操作を受け付けます。

```
// 最初の列をワークシートの最後(左から右)にする
colmove rng:=col(1) operation:=last;

// 列 2-4 をワークシートの最も左に移動
colmove rng:=Col(2):Col(4) operation:=first;
```

列の名前を変更する・ラベルを付ける

列の名前(ショートネーム)を変更するために、Origin は *name$*プロパティと共に wks.col オブジェクトを使用できます。また、列ラベル行の文字、***G*** を使用して列のショートネームを変更できます。

```
// 新しいワークブックを作成
newbook;

// 列 1 の名前を DataX に変更
wks.col1.name$ = DataX;
// 範囲を使用して列２の名前を DataY に変更
range rY = 2;  // 範囲を列 2 に設定する
rY.name$ = DataY;

// 新しい列を追加する
wks.addCol();
// "G"で名前を変更する
col(3)[G]$ = "Result";
```

列ラベル行の文字は列ラベルに簡単にアクセスできる方法です。列ラベルにはロングネーム、単位、コメント、列パラメータ、ユーザ定義パラメータなどが含まれます。

```
// 新しいワークブックを作成
newbook result:=BkName$;

// 以下のラベル行を表示する
// ロングネーム、単位、コメント、最初の列パラメータ
// 最初のユーザ定義パラメータ
wks.labels(LUCP1D1);

// 列 1 と列 2 を定義する
range r1 = [%(BkName$)]1!1;
range r2 = [%(BkName$)]1!2;
// col を使用してロングネームをセットする
col(1)[L]$ = Time;
col(2)[L]$ = Voltage;
// 範囲で単位をセット
r1[U]$ = Sec;
r2[U]$ = V;
// 範囲を使用して列コメントをセット
r1[C]$ = Sample1;
r2[C]$ = Sample1;
```

```
// 範囲を使用して列パラメータをセット
r1[P1]$ = "Machine1";
r2[P1]$ = "Machine1";

// 最初のユーザ定義パラメータの名前を変更
wks.UserParam1$ = Current;

// 現在のラベル行をセット
r1[Current]$ = 1mA;
r2[Current]$ = 1mA;
```

列の表示/非表示

列を表示/非表示にするには、colHide X ファンクションを使用します。

```
// 新しいワークブックを作成
newbook;

// ワークシートの列番号をセット
wks.nCols = 6;

// 2 列目を非表示にする
colHide 2 hide;

//3 列目と 5 列目を非表示にする
colHide (3, 5) hide;
```

非表示にした列を表示にするには、2 つ目の引数を *hide* から *unhide* に変更して実行します。

列の入れ替え

2 つの特定の列を入れ替えるには colSwap X ファンクションを使用します。

```
// 新しいワークブックを作成
newbook;

// 列 1 と列 2 の位置を入れ替える
colSwap (1, 2);
```

ここで指定する 2 つの列は、必ずしも隣り合っている必要はありません。

```
// 新しいワークブックを作成
newbook;

// 列の数を 6 としてセット
wks.ncols = 6;

// 2 列目と 4 列目を入れ替える
colswap (2, 4);
```

列の形式を変更する

プロットの割り当て

列へのプロット属性は選択したデータがデフォルトでどのように扱い、作図をするのか決定します。プロット属性は、X、Y、Z、Z エラー、Y エラー、ラベルなどを含みます。これらは wks.col.type を使用して変更できます。

```
// データをインポート
newbook;
string fname$ = system.path.program$;
fname$ += "Samples\Matrix Conversion and Gridding\XYZ Random Gaussian.dat";
impasc;

// プロット属性をセット
wks.col = 3;  // 列 3 を現在の列にセット
wks.col.type = 6;  // Z

// 3 番目の列（Z 列）を選択
worksheet -s 3 1 3 -1;
// 温度を元にしたカラーマップ曲面図を OpenGL で作成
worksheet -p 103 glcmap;
```

列幅

列幅をセットするには、wcolwidth X ファンクションを使用します。または、wks.col.width も使用できます。

```
// ワークブックを開く
string strPath$ = system.path.program$;
strPath$ += "Samples\Graphing\Automobile Data.ogw";
doc -o %(strPath$);

// 列 2 を ###ではなく、数字を全て表示する
// 列 2 の長さを 6 文字分にセット
wcolwidth irng:=col(2) width:=6;
```

データ形式と表示

列に正しいデータ形式（データ型）を設定して、列データを正しく表示できます。また、作図やデータ分析などの操作を行う際の助けになります。列は、数値、テキスト、日付、時間、月、曜日等多くのデータ型を選択できます。データ型をセットするには wks.col オブジェクトの *format* プロパティを使用してください。

```
// データをインポート
newbook;
string fname$ = system.path.program$;
fname$ += "Samples\Signal Processing\Average Sunspot.dat";
impasc;

// 列 2 に数値（現在はテキストと数値）に設定
wks.col2.format = 1;  // 数値 = 1

// 桁数指定法を "小数桁数"にし、
// 桁数を 2 にする
wks.col2.digitMode = 1;  // 小数点以下の桁数を指定
```

```
wks.col2.digits = 2;  // 書数点以下 2 桁
```

以下のサンプルはそれぞれの対応する形式に対応する設定を紹介します。

1. 数値

```
// データをインポート
newbook;
string fname$ = system.path.program$;
fname$ += "Samples\Curve Fitting\Enzyme.dat";
impasc;

//  列 2 に数値（現在はテキストと数値）にセット
wks.col2.format = 1;  // 数値 = 1
// 表示フォーマットをカンマ付にする
wks.col2.subformat = 4;  // 小数点として表示 1,000
// データ型は短い整数にする
wks.col2.numerictype = 3;


// 列 3 にも同じような操作をする
wks.col3.format = 1;  // 数値 = 1
// 表示フォーマットはカンマ付にする
wks.col3.subformat = 4;  // 小数桁として表示 1,000
// データ型を短い整数にする
wks.col3.numerictype = 3;
```

2. 日付

日付と時間のフォーマットに関して、列内のデータがユリウス歴（日付と時間のフォーマットに見えますが、実際には時間です）ではない場合、直接日付や時間としてセットできません。あるいは日付と時間フォーマットに見える場合、数値が欠損値になるか、別のものになります。この問題を避けるために、Origin は **wks.col.setformat()** メソッドを提供しています。

```
// データをインポート
newbook;
string fname$ = system.path.program$;
fname$ += "Samples\Import and Export\Custom Date and Time.dat";
impasc;

// 列 1 のデータ型を日付にセット
// カスタム表示形式では、
// 現在のようなテキスト表示を列に行う
wks.col1.setformat(4, 22, dd'.'MM'.'yyyy HH':'mm':'ss'.'##);
// データ型 yyyy/MM/dd HH:mm:ss をセット
wks.col1.subformat = 11;
```

3. 時間

上記の日付の説明を参照してください。

```
// データをインポート
newbook;
string fname$ = system.path.program$;
fname$ += "Samples\Import and Export\IRIG Time.dat";
impasc;

// 列 1 のデータ型を時間にセット
wks.col1.format = 3;  // 時間 = 3
// IRIG 時間形式「DDD:HH:mm:ss.##」で表示
wks.col1.subformat = 16;
```

4. 月

```
// 列 1 のデータ型を月にセット
// そして、月の名前を全て表示
wks.col1.format = 5;  // 月 = 5
wks.col1.subformat = 2;  // 月の名前を全て表示
```

5. 曜日

```
// 列 1 のデータ型を曜日にセット
// 各曜日の最初の文字のみを表示
wks.col1.format = 6;  // 曜日 = 6
wks.col1.subformat = 3;  // 各曜日の最初の文字を表示
```

列にスパークラインを追加する

sparklines X ファンクションは指定した列にスパークラインを追加するのに使用します。

```
// ワークブックを開く
string strPath$ = system.path.program$;
strPath$ += "Samples\Graphing\Automobile Data.ogw";
doc -o %(strPath$);

//ロングネームに"Year" とついている列以外全てにスパークラインを追加
for(ii = 2; ii <= wks.nCols; ii+=5)
{
        sparklines sel:=0 c1:=ii c2:=ii+3;
}
```

列の削除

delete コマンドはワークシートから列を削除する事ができます。

```
// ワークブックを作成
newbook;

// 列 B を削除
delete col(B);

// 新しいワークシートに列を 4 つ追加
newsheet cols:=4;

// 範囲を使用して列 3 を削除
range rr = 3;  // 列 3 はワークシートに新たに追加される
delete rr;
```

削除したい列がワークシートの最後にある場合、削除する列を wks.nCols により指定すると、削除できます。

```
// ワークブックを開く
string strPath$ = system.path.program$;
strPath$ += "Samples\Graphing\Automobile Data.ogw";
doc -o %(strPath$);

// 開いたワークシートの最後の 20 列分を削除
wks.nCols = wks.nCols-20;
```

9.3.2. ワークシート列データの操作

内容

基本的な操作

いくつかの数値データをロードしたり、作成したら、ここに示すような操作を行うことができます。

基本的な算術計算

ほとんどのデータは列に保存されており、そのデータに対して行毎にさまざまな操作を実行することができます。LabTalk スクリプトで 2 つの方法でこれを行うことができます。(1) 演算子を使って直接ステートメントまたは (2) 範囲を使う例えば、列 A の各行に値を追加し、列 B にはそれに対応する値を追加し、列 C に結果の値を配置します。

```
Col(C) = Col(A) + Col(B);  // 加算
Col(D) = Col(A) * Col(B);  // 乗算
Col(E) = Col(A) / Col(B);  // 除算
```

「 - 」と「 ^ 」の演算子は、それぞれ減算と指数として上記のように動作します。

範囲変数を使えば、異なるシートにある列に対して同じ操作を行うことができます。

```
// sheet1, 2, 3 の列 1 をポイント
range aa = 1!col(1);
range bb = 2!col(1);
range cc = 3!col(1);
cc = aa+bb;
cc = aa^bb;
cc = aa/bb;
```

異なるシートにあるデータを算術演算するときには、範囲変数を使う必要があります。範囲文字列への直接参照はサポートされていません。例えば、スクリプト **Sheet3!col(1) = Sheet1!col(1) + Sheet2!col(1);** は動作しません。

関数

標準的な演算子に加え、LabTalk は、**sin** や **cos** のような三角関数、Bessel 関数、**uniform** や **Poisson** のような統計分布データを生成する関数など、データを操作する多くの一般的な関数をサポートしています。すべての LabTalk 関数は、1 つの引数で動作しますが、多くはその中でベクター化され、ワークシート列、ルーズデータセット、行列で動作します。例えば、三角関数 **sin** は次のようになります。

```
// 数値の sin を見つける:
double xx = sin(0.3572)
// 列データの sin を見つける （行毎）
Col(B) = sin(Col(A))
// 行列データの sin を見つける （要素毎）
[MBook2] = sin([MBook1])
```

関数の例として、主要な機能がデータ生成である **uniform** 関数を考えます。この形式では、作成する値の数として入力 *N* を取り、そして、0 から 1 までの範囲で均一に分布した乱数を *N* 個生成します。

```
/* 列 B の最初の 20 行に一様に分布した乱数を入力:*/
Col(B) = uniform(20);
```

LabTalk でサポートする関数の完全なリストについては、アルファベット順の関数リストをご覧下さい。

列に式をセットする

Origin のメニュー操作で、列値の設定ダイアログを使って、指定した数式でワークシート列にデータを生成または変換することができます。このような変換は、csetvalue X ファンクションを使って LabTalk で実行することもできます。ここに LabTalk を使って列値をセットする方法を示すサンプルがいくつかあります。

```
newbook;
wks.ncols = 3;
// 列 1 に乱数を入力
csetvalue formula:="rnd()" col:=1;
```

```
// 列 1 のデータを 0 ~ 100 の整数値に変換
csetvalue formula:="int(col(1)*100)" col:=2;
// 列値を設定するときに実行前の処理スクリプトを指定
// 再計算モードを手動にセット
csetvalue formula:="mm - col(2)" col:=3 script:="int mm = max(col(2))"
recalculate:=2;
string str$ = [%h]%(page.active$)!;
newsheet cols:=1;
// range 変数を使って別のシートの列を参照
csetvalue f:="r1/r2" c:=1 s:="range r1=%(str$)2; range r2=%(str$)3;" r:=1;
```

列の式をセットするのに論理宣言が使用されているとき、0.0, NANUM(欠損値)や-1.0E-290 から 1.0E-290 の値などは、*False* として評価されます。つまり、LabTalk はコマンドは、1(真)の代わりに 0(偽)を返します。

```
type $(-1e-290?1:0); // 0 (偽)を返す
type $(1/0?1:0); // 0 (偽)を返す, ここで 1/0 == NANUM
```

列のコピー

colcopy X ファンクションは、列ラベル行と日付や文字と数値のような列フォーマットを含むデータ列をコピーします。

次の例は、アクティブワークシートの 2 列目から 4 列目を book2 の sheet1 の 1 列目から 3 列目にコピーします。

```
// 列のロングネーム、単位とコメントだけでなく
// データと書式の両方をコピー
colcopy irng:=(2:4) orng:=[book2]sheet1!(1:3) data:=1
        format:=1 lname:=1 units:=1 comments:=1;
```

列をソートする

特定の列をソートするには、X ファンクション wsort が使用できます。この X ファンクションを使って 1 列のみをソートするには、引数 C1 と C2 は、ワークシートの同じ列でなければならず、bycol も C1 と同じである必要があります。

```
// 新しいワークブックを作成
newbook;

// 第 1 列は行番号、2 列目は一様乱数で埋める
col(1) = {1:32};
col(2) = uniform(32);

// 2 列目を降順にソートする
wsort c1:=2 c2:=2 bycol:=2 descending:=1;
```

列を逆順にする

X ファンクション colreverse を使用して列を逆順にします。

```
// 新しいワークブックを作成
newbook;

// 第 1 列は行番号、2 列目は一様乱数で埋める
```

```
col(1) = {1:32};
col(2) = uniform(32);

// 1 列目をインデックスを使用して逆順にする
colreverse rng:=1;  // colreverse rng:=col(A);  // このスクリプトも使用可能

// 2 列目を range 変数を使用して逆順にする
range rr = 2;
colreverse rng:=rr;
```

9.3.3. 日付と時間データ

やの情報を表示するために使われる、さまざまな文字列フォーマットは、情報をユーザに通知するのに役立ちますが、これらの値をプロットしたり分析するには、これらの値の数学的基礎が必要です。Origin は、天文学的なユリウス暦を改訂して日付と時刻を保存しています。これは、時刻 0 は、BC4713/01/01 の正午 12:00 です。数字の整数部分は時刻 0 からの日数を表し、小数部分は 24 時間表示での分数を表します。Origin は、正午ではなく、夜中の 0:00 を表すように、12 時間(0.50 日)減算してこの値を使っています。

次のいくつかのサンプルでは、LabTalk スクリプトで日付と時刻のデータを扱う方法を示しています。

Note:日付 または **時刻** で表されるテキストは、実際には、Origin で数値として扱われない**文字** または **文字と数値** の形式です。列プロパティダイアログ（列名をダブルクリックするか列を選択し、「フォーマット：列フォーマット」）を使用して、**テキスト**または**文字と数値**列を、**日付**か**時間**フォーマットに変換します。表示フォーマットは、変換時に列のテキストフォーマットに一致する必要があります。

内容

日付と時刻

例として、アクティブシートの列 1 に日付データ、列に時刻データがあるものとします。そして、日付と時刻を 1 つの列に保存するものとします。

```
/* 日付と時刻の両方は数学的基礎で成り立っているので加算可*/
Col(3) = Col(1) + Col(2);

// デフォルトで、新しい列は日数を表示
/* フォーマットとサブフォーマット法を使って選択した日付と時間の表示をセット*/

// Format #4 は日付フォーマット
wks.col3.format = 4;
// サブフォーマット #11 は MM/dd/yyyy hh:mm:ss
wks.col3.subformat = 11;
```

上記の列番号は、format ステートメントにハードコーディングされています。列数を **cn** という変数に持っている場合、**wks.col$(cn).format = 4**.のようにして、数字の **3** を**$(cn)**で置き換えることができます。他の **format** および **subformat** オプションに対しては、LabTalk 言語リファレンス: オブジェクトリファレンス: Wks.col (オブジェクト)をご覧下さい。

日付と時刻の列は、列 1 で **MM/dd/yyyy** のフォーマット、列 2 で **hh:mm:ss** のフォーマットを持つテキストの場合、同じ操作がいくつかのコード行で可能です。

```
// ループする行数を取得
int nn = wks.col1.nrows;

loop(ii,1,nn){
  string dd$ = Col(1)[ii]$;
  string tt$ = Col(2)[ii]$;
  // 日付と時間の文字列をテキストとして保存
  Col(3)[ii]$ = dd$ + " " + tt$;
  // 日付関数は日付と時間の文字列を数値の日付値に変換
  Col(4)[ii] = date(%(dd$) %(tt$));
};
// 列 4 を真の日付列に変換
wks.col4.format = 4; // 日付列に変換
wks.col4.subformat = 11; // M/d/yyyy hh:mm:ss として表示
```

ここで、中間的な列は、文字列として日付と時刻の組合せを保持するように形式化され、結果の日付と時刻(数値)の値は 4 番目の列に保存されます。これらは同じテキストになるよう現れますが、列 C は文字通りのテキストで、列 D は日付です。

この数学的な方法によって、2 つの日付の差(2 つの日付の日数、時間) を計算し、その結果を時間列に入れることができます。そして、時刻を日付に加え、新しい日付の値を計算できます。また、時刻データを時刻データに追加して、有効な時刻データを取得することができますが、日付データを日付データに加えることはできません。

出力用のフォーマット

利用可能な形式

D 表記を使って、Origin の組込の日付フォーマットの 1 つを使って、数値の日付の値 を文字列の日付時刻に変更することができます。

```
type "$(@D, D10)";
```

これは、現在の日付と時間(システム変数 **@D** に保存) を読み込み可能な文字列として返します。

```
7/20/2009 10:30:48
```

D10 オプションは、**MM/dd/yyyy hh:mm:ss** に対応しています。上記スクリプトの D 文字の後の数値を変更することで、多くの他の出力フォーマットが利用できます。これはワークシート列フォーマットダイアログの**日付フォーマット**ドロップダウンリストの選択肢のインデックス(0 から)です。最初の項目(インデックス = 0)は、Windows の短縮形式の日付で、2 番目は Windows の長い形式の日付です。

Note:*D は大文字でなければなりません。wks.col3.subformat = #, のようなワークシートのサブフォーマットを設定する場合、これらのインデックス値は 1 から始まります。*

例えば

```
type "$(date(09/07/20), D1)";
```

は、日本の地域の設定

```
09/07/20, 月曜日となります。
```

同様に、時刻の値の出力フォーマットに対して、類似の T 表記があります。

```
type "$(time(12:04:14), T5)";    // ANS:12:04 PM
```

このように日付と時刻をフォーマットすることは、一般的な $() 置換 表記の特殊な形式を使います。

カスタムフォーマット

3 つのカスタム日付と時刻のフォーマットがあります。そのうち 2 つはスクリプト編集可能なプロパティで、もう 1 つは列のプロパティダイアログで編集するか、ワークシート列オブジェクトメソッドで編集します。

1. **system.date.customformat*n*$**
2. **wks.col.SetFormat** オブジェクトメソッド

両方のメソッドは、カスタムフォーマットを示すために、**yyyy'.'MM'.'dd** のような日付と時刻の指定子を使います。以下に注意して下さい。

- 日付時刻指定子のテキストの一部 (非スペースの区切り)は、必要に応じて変更できますが、引用符で囲む必要があります。
- 指定子のトークン自体 (i.e., yyyy, HH, etc.)は、大文字小文字を区別し、表示通りに使う必要があります。— 利用可能なすべての指定子のトークンは、リファレンステーブル: 日付と時刻フォーマットの指定子にあります。日付と時間フォーマットの指定子
- 最初の 2 つのフォーマットは、ローカルなファイルにその説明を保存し、他のログインアカウントで別の表示することができます。3 番目のフォーマットには列自体の説明が保存されます。

D*nn* 表記

Origin は、これらのカスタム日付の表示に対して、**D19** から **D21** を予約しています(D の後の整数値は 0 から数えるので、サブフォーマット 20 から 22)。オプションの D19 と D20 は、それぞれシステム変数 **system.date.customformat1$** と **system.date.customformat2$** で制御されます。このオプションを使うには、以下のサンプルをご覧下さい。

```
system.date.customformat1$ = MMM dd hh'.'mm tt;
type "$(Date(7/25/09 14:47:21),D19)";       // 出力 Jul 25 02.47 PM

system.date.customformat2$ = yy','MM','dd H'.'mm'.'ss'.'####;
type "$(Date(7/27/09 8:22:37.75234),D20)"; // 出力 09,07,27 8.22.37.7523
```

Wks.col.SetFormat オブジェクトメソッド

ワークシート列に保存されている日付列のカスタム日付表示を指定するには、Wks.Col.SetFormat オブジェクトメソッドを使います。カスタム日付フォーマット指定子を入力するとき、非日付文字を引用符で囲んで下さい。このオブジェクトメソッドは、アクティブワークシートの列に対してのみ動作します。

次のサンプルでは、アクティブワークシートの列 4 は、カスタム日付/時刻フォーマットを表示するようセットされています。

```
// wks.format=4 (日付), wks.subformat=22 (カスタム)
wks.col4.SetFormat(4, 22, yyyy'-'MM'-'dd HH':'mm':'ss'.'###);
```

```
doc -uw; // リフレッシュして変更を表示
```

10 行列ブック、行列シート、行列オブジェクト

ワークブックやワークシートと同様に、Origin の行列は階層構造でデータを保持します。行列データの場合、行列ブック -> 行列シート -> 行列オブジェクト の順に構成されます。そのため、Page や Wks のようなオブジェクトは、ワークブックやワークシートだけでなく、行列ブックや行列シートの意味の含まれます。さらに、Origin には 行列データを扱う X ファンクションが数多くあります。

この章で以下の項目を説明します

- 行列ブックの基本操作
- 行列シート
- 行列オブジェクト

10.1. 行列ブックの基本操作

行列ブックは Origin のワークブックと同じデータ構造レベルに属し、どちらもウィンドウです。つまり、ワークブックと同じように行列ブックを Page オブジェクトと Window コマンドで操作することができます。

10.1.1. ワークブックのような操作

行列ブックとワークブックはどちらもウィンドウで、たくさんの類似した操作方法を共有しています。LabTalk スクリプトも、同じスクリプトを使用することが良くあります。異なる部分を下記に抜粋しました。同じスクリプトを使用している場合はワークブックの基本操作を参照してください。

1. 新しい行列ブックを作成
   X ファンクションの newbook を使用して新しい行列ブックを作成する場合、引数 **mat** を 1 にする必要があります。
   以下は、ワークブックでのサンプルと同様のサンプルです。

```
//ロングネーム"MyMatrixBook"を持つ、新しい行列ブックを作成
newbook mat:=1 name:=MyMatrixBook;

//  3 つの行列シートを持つ、新しい行列ブックを作成
// "Images"をロングネームとショートネームとして使用
newbook mat:=1 name:=Images sheet:=3 option:=lsname;

// 非表示の新しい行列部ブックを作成
// 行列ブック名は 「myBkName$」変数内に格納される
newbook mat:=1 hidden:=1 result:=myBkName$;
// 行列ブック名を出力
myBkName$ = ;
```

2. 行列ブックを開く
   行列ブックを開くときはワークブックを開くときと同じように、doc -o コマンドを使用します。違いとしては、行列ブックを開くときの拡張子は *ogm* です。

3. 行列ブックを保存
   Origin のデータがある行列ブックを示す拡張子は **ogm** で、データが無いテンプレートは **otm** です。行列ブックを ogm または otm ファイルとして保存するには、それぞれ save -i コマンドと template_saveas X ファンクションを使用します。しかし、行列ブックは分析テンプレートとしては保存できません。

4. 行列ブックを閉じる
   これはワークブックと同じです。win -ca と win -cd コマンドを参照してください。

5. 行列ブックを表示/非表示にする
   これはワークブックと同じです。詳細は win コマンドの*-ch*, *-h*, *-hc* スイッチを確認してください

6. 行列ブックに名前やラベルを付ける
   これはワークブックと同じです。win -r コマンドと page オブジェクトを参照してください。

7. 行列ブックをアクティブにする
   これはワークブックと同じです。win -a コマンドを参照してください。コマンド *window -o winName {script}* は、名前を付けた行列ブックに対して指定したスクリプトを実行するのに使用できます。より詳細な説明については、スクリプトの実行の章 をご覧ください。

8. 行列ブックを削除
   これはワークブックと同じです。win -a コマンドを参照してください。

10.1.2. イメージサムネールを表示する

イメージサムネールを表示または非表示にするには matrix -it コマンドを使用出来ます。

```
// 新しい行列ブックを作成
newbook mat:=1;
// イメージをインポート
string strImg$ = system.path.program$;
strImg$ += "Samples\Image Processing and Analysis\bamboo.jpg";
impImage fname:=strImg$;

// イメージサムネールを非表示
matrix -it 0;
```

10.2. 行列シート

行列シートは、Origin のワークシートと同様の構造レベルのものです。そのため、多くの共通項を有しています。

このセクションで以下の項目を説明します

- 行列シートの基本操作
- 行列シートのデータ操作

10.2.1. 行列シートの基本操作

このセクションのサンプルは、多くのオブジェクトプロパティと X ファンクションがワークシートと行列シートの両方に適用するので、ワークシート基本操作 セクションにあるサンプルに似ています。**wks** オブジェクトのすべてのプロパティが行列シートに適用するわけではないので、コード内で使用する前に確認する必要があります。

内容

新しい行列シートを追加する

mat:=1 オプション付き **newsheet** X ファンクションは、行列シートを行列ブックに追加するのに使用します。

```
// 3 つの行列シートを持つ新しい行列ブックを作成し
// ロングネームとショートネームに "myMatrix" を使用
newbook name:="myMatrix" sheet:=3 option:=lsname mat:=1;
// 現在の行列ブックに "newMatrix" という名前の 100*100 のシートを追加
newsheet name:=newMatrix cols:=100 rows:=100 mat:=1;
```

行列シートをアクティブにする

ワークシートと同様、行列シートは、page にある内部的なレイヤで、*page.active$* プロパティにより行列シートにアクセス可能です。例えば、

```
// 3 つの行列シートをもつ、行列ブックを作成
newbook sheet:=3 mat:=1;

page.active = 2;       // レイヤ番号で行列をアクティブにする
page.active$ = MSheet3;       // 名前で行列をアクティブにする
```

行列シートのプロパティを修正する

行列のプロパティを修正するには、wks オブジェクトを使います。これはワークシートと同じように行列シートに対して動作します。例えば、

```
// 行列シートの名前の変更
wks.name$ = "New Matrix";
// 列幅の修正
wks.colwidth = 8;
```

次数/座標値の設定

wks オブジェクトと mdim X ファンクションの両方は、行列の次数をセットするのに使用できます。

```
// wks オブジェクトを使って次数をセット
wks.ncols = 100;
wks.nrows = 200;
// mdim X ファンクションを使って次数をセット
mdim cols:=100 rows:=100;
```

同じ行列シートに含まれる複数行列オブジェクトの場合には、すべての行列オブジェクトは同じ次数でなければなりません。

XY マッピングをセットする

行列は、番号が付けられた列と行を持ち、これらは等間隔で線形にマッピングされた X 値および Y 値です。LabTalk では、mdim X ファンクションを使ってマッピングをセットできます。

```
// 行列シートの XY マッピング
mdim cols:=100 rows:=100 x1:=2 x2:=4 y1:=4 y2:=9;
```

行列シートを削除する

layer -d コマンドにより、行列シートの削除が可能です。例えば、

```
layer -d;  // アクティブレイヤを削除、ワークシート、行列シート、グラフレイヤにできる
layer -d 3;  // インデックスで、アクティブ行列ブックの 3 番目の行列シートを削除
layer -d msheet1;  // 名前で行列シートを削除
range rs = [mbook1]msheet3!;
layer -d rs;  // 範囲で行列シートを削除
// 文字列変数に保存された行列ブック名
string str$ = msheet2;
layer -d %(str$);
```

10.2.2. 行列シートのデータ操作行列シート

行列シートと行列オブジェクト間の変換

Origin で行列シートは複数の行列オブジェクトを持つことができます。mo2s X ファンクションを使うと、複数の行列オブジェクトを別々の行列シートに分けます。

ms2o X ファンクションを使って、複数の行列シートを一つにすることができます(与えられたすべての行列は同じ次数を共有)。

```
// 行列シート 2, 3, 4 を統合
ms2o imp:=MBook1 sheets:="2,3,4" oms:=Merge;
// MSheet1 の行列オブジェクトを新しいシートに分割
mo2s ims:=MSheet1 omp:=<new>;
```

10.3. 行列オブジェクト

行列オブジェクトは、強力な行列データの基本的な単位で、この上位の入れものとして行列シートがあります。これはワークシートと列の関係と同様です。以下のページでは、行列オブジェクトの操作に関する実践的なサンプルを紹介します。

この章で以下の項目を説明します

- 行列オブジェクトの基本操作
- 行列オブジェクトデータ操作
- 行列をワークシートに変換

10.3.1. 行列オブジェクトの基本操作

行列シートは複数の行列オブジェクトを持つことができ、それらは共通の次数を持ちます。行列オブジェクトはワークシートと同じように認識することができ、追加や削除ができます。以下のセクションは行列オブジェクトの基本的な操作について、実践的なサンプルを通して紹介します。

内容

行列オブジェクトの追加または挿入

wks.nmats を使用すると行列シート内に複数の行列オブジェクトをセットして行列オブジェクトを追加する事ができます。また、wks.addcol()の方法は行列オブジェクトを追加する際にも使用できます。

```
// 行列シート内に 5 つの行列オブジェクトをセット
wks.nmats = 5;
// 行列シートに新しい行列オブジェクトを追加
wks.addCol();
// 名前のついた行列オブジェクトを行列シートに追加
wks.addCol(Channel2);
```

デフォルトでは、行列シート内にある最初の行列オブジェクトが現在の行列オブジェクトとなり、wks.col を使用します。また、wks.insert()方法は現在の行列オブジェクトの前に行列オブジェクトを挿入します。

```
// 新しい行列ブックを作成し、イメージサムネールを表示
newbook mat:=1;
matrix -it 1;


// アクティブな行列シートの中の 1 番目の行列オブジェクトの前に行列オブジェクトを挿入
wks.insert();

// 2 番目の行列オブジェクトを現在のオブジェクトにする
wks.col = 2;
```

```
// 2 番目の行列オブジェクトの前に行列オブジェクトを挿入
wks.insert();
```

行列オブジェクトをアクティブ化する

アクティブな行列シートの中にある行列オブジェクトをアクティブ化するには **wks.active** を使用します。

```
// 新しい行列ブックを作成
newbook mat:=1;

// アクティブな行列シートにさらに 2 つの行列オブジェクトを追加
wks.addCol();
wks.addCol();

// イメージサムネールを表示
matrix -it 1;

// 2 番目の行列オブジェクトをアクティブにする
wks.active = 2;
```

イメージモードとデータモードを切り替える

matrix コマンドは行列オブジェクトのイメージモードとデータモードを切り替えるオプションを提供しています。アクティブな行列オブジェクトのみが行列シートに現れます。

```
matrix -ii 1;  // イメージモードで表示
matrix -ii 0;  // データモードで表示
```

ラベルのセット

各行列オブジェクトについてのロングネーム、コメント、単位は範囲表記を使用してセットします。

```
// 新しい行列ブックを作成
newbook mat:=1;

// 1 番目の行列シートに含む行列オブジェクトの数を 3 つにする
wks.nMats = 3;

// イメージサムネールを表示
matrix -it 1;

// 最初の行列オブジェクトをアクティブにする
wks.active = 1;

// ロングネーム、単位、コメントをセット
range rx = 1;  // アクティブな行列シートの最初の行列オブジェクト
rx.lname$ = X;  // ロングネーム = X
rx.unit$ = cm;  // 単位= cm
rx.comment$ = "X Direction";  // コメント = "X Direction"

// 同じことを行列オブジェクト 2 と 3 にも行う
wks.active = 2;
```

```
range ry = 2;
ry.label$ = Y;  // ロングネームはこの方法でもセット可能
ry.unit$ = cm;
ry.comment$ = "Y Direction";
wks.active = 3;
range rz = 3;
rz.label$ = Z;
rz.unit$ = Pa;
rz.comment$ = Pressure;
```

行列オブジェクト削除する

行列オブジェクトを削除するには、delete コマンドを使用します。

```
// 行列オブジェクトを範囲で削除
range rs=[mbook1]msheet1!1;  // 最初の行列オブジェクト
del rs;
// または、行列オブジェクトの名前で削除
range rs=[mbook1]msheet1!Channel2;  // オブジェクト名は Channel2
del rs;
```

10.3.2. 行列オブジェクトのデータ操作

matrix コマンドに加え、Origin は行列オブジェクトデータの特定の操作を実行する X ファンクションを提供します。このセクションでは、行列オブジェクトデータを操作するのに使用される利用可能な X ファンクションのサンプルを紹介します。

内容

行列オブジェクトの値を設定する

行列のセルの値は、matrix -v コマンドまたは msetvalue X ファンクションのどちらかを使ってセットできます。matrix -v コマンドはアクティブな行列オブジェクトのみを操作しますが、X ファンクションはどの行列シートの値でも設定できます。

このサンプルは行列の値をセットして、行列ウィンドウにイメージサムネールを表示する方法を示します。

```
// 行列ブックを作成
mat:=1;
int nmats = 10;
range msheet=1!;
// 行列オブジェクトの値をセット
msheet.Nmats = nmats;
// 最初の行列オブジェクトに値をセット
matrix -v x+y;
range mm=1; mm.label$="x+y";
double ff=0;
// 他のオブジェクトをループ
loop(i, 2, nmats-1) {
        msheet.active = i;
        ff = (i-1)/(nmats-2);
        // 値をセット
        matrix -v (5/ff)*sin(x) + ff*20*cos(y);
        // ロングネームをセット
        ange aa=$(i);
        aa.label$="$(5/ff,*3)*sin(x) + $(ff*20)*cos(y)";
}
// 乱数の値で最後の 1 つを入力
msheet.active = nmats;
matrix -v rnd();
range mm=$(nmats); mm.label$="random";
// ウィンドウにサムネールを表示
matrix -it;
```

行列データのコピー

mcopy X ファンクションは行列データを コピー する時に使用します。

```
// mbook1 から別の行列 mbook2 にデータをコピー
mcopy im:=mbook1 om:=mbook2; // このコマンドはコピー先の次数を自動設定
```

行列オブジェクトとベクターを変換する

2 つの X ファンクション、m2v と v2m は、それぞれ行列データをベクターに、ベクターデータを行列に変換する時に利用できます。Origin は、行列を保存するために行を主とした順番を使いますが、どちらの関数も列を主とした順番を指定することができます。

```
// 行列全体を列ごとにワークシート列にコピー
m2v method:=m2v direction:=col;
// col(1) からデータを指定した行列オブジェクトにコピー
v2m ix:=col(1) method:=v2row om:=[Mbook1]1!1;
```

数値データとイメージデータ間を変換する

Origin では、行列はイメージデータ(RGB など)または数値データ(整数値など)を含むことができます。次の関数は 2 つのフォーマット間を変換する際に利用できます。

```
// グレースケール画像を数値データに変換
img2m img:=mat(1) om:=mat(2) type:=byte;
// 数値行列をグレースケール画像に変換
m2img bits:=16;
```

複素数値を持つ行列を操作する

複素数値を持つ行列を操作する X ファンクションには map2c, mc2ap, mri2c, mc2ri があります。これらの X ファンクションは、2 つの行列(振幅と位相や実数と虚数)を 1 つの複素行列に統合したり、1 つの複素行列を振幅/位相や実数/虚数成分に分けます。

```
// 振幅と位相を複素データに組合せる
map2c am:=mat(1) pm:=mat(2) cm:=mat(3);
// 異なる行列の実数と虚数を新しい行列の複素データに組合せる
mri2c rm:=[MBook1]MSheet1!mat(1) im:=[MBook2]MSheet1!mat(1) cm:=<new>;
// 複素数を振幅と位相を持つ 2 つの新しい行列に変換
mc2ap cm:=mat(1) am:=<new> pm:=<new>;
// 複素数を実数部と虚数部の 2 つの行列オブジェクトに変換
mc2ri cm:=[MBook1]MSheet1!Complex rm:=[Split]Real im:=[Split]Imaginary;
```

行列オブジェクトデータを変換する

次の X ファンクションを使って、物理的に行列の次数または内容を変更します。以下の変換では、行列オブジェクトの反転を除き、行列シートの次数が変更し、同一行列ブック内の他のマトリックスオブジェクトにも変更が加えられます。

データまたはイメージの行列からコピーまたは抽出する

行列が行列内にイメージを含む場合、X ファンクション mcrop は、行列の矩形領域を抽出またはコピーするのに使うことができます。

```
// 左から 10 ピクセル、上から 20 ピクセルの位置から開始し
// 50×25 のイメージ行列をコピー
mcrop x:=10 y:=20 w:=50 h:=25 im:=<active> om:=<input>; // <input> はコピー
// 行列の中央部を新しい行列に抽出
// 行列ウィンドウがアクティブであること
matrix -pg DIM px py;
dx = nint(px/3);
dy = nint(py/3);
mcrop x:=dx y:=dy h:=dy w:=dx om:=<new>; // <new> は抽出
```

データ行列を拡張する

X ファンクション mexpand は、指定した列と行の係数を使って、データ行列を拡張できます。複二次補間を使って、新しいセルの値を計算します。

```
// アクティブ行列を係数 2 で拡張
mexpand cols:=2 rows:=2;
```

データまたはイメージの行列を反転する

X ファンクション mflip は行列を水平方向または垂直方向に反転し、その行列の対称な行列を作成します。

```
// 行列を垂直に反転
```

```
mflip flip:=vertical;

// "matrix" コマンドも使用可能
matrix -c h;  // 水平に
matrix -c v;  // 垂直に
```

データまたはイメージの行列を反転する

X ファンクション mrotate90 を使って、行列を 90/180 度時計回りまたは反時計回りに回転できます。

```
// 行列を 90 度時計回りに回転
mrotate90 degree:=cw90;

//  "matrix" コマンドでも 90 度回転可能
matrix -c r;
```

データ行列を縮小する

X ファンクション mshrink は、指定した列と行の係数を使って、データ行列を縮小できます。

```
// アクティブな行列を列の係数 2、行の係数 1 で縮小
mshrink cols:=2 rows:=1;
```

データ行列を転置する

X ファンクション mtranspose は行列を転置するのに使用することができます。

```
// [MBook1]MSheet1!の 2 番目の行列オブジェクトを転置
mtranspose im:=[MBook1]MSheet1!2;

// "matrix" コマンドを使って行列転置も可能
matrix -t;
```

RGB のイメージをチャンネルに分割する

imgRGBsplit X ファンクションはカラー画像を別々の R, G, B チャンネルに分割します。例えば:

```
// チャンネルを分割して、赤、緑、青の別々の行列を作成
imgRGBsplit img:=mat(1) r:=mat(2) g:=mat(3) b:=mat(4) colorize:=0;
// チャンネルを分割して、赤、緑、青のパレットを結果行列に適用
imgRGBsplit img:=mat(1) r:=mat(2) g:=mat(3) b:=mat(4) colorize:=1;
```

イメージの取扱いについての詳細は、イメージ処理の X ファンクション をご覧ください。

10.3.3. 行列からワークシートに変換する

分析または、グラフ作成時に、行列からワークシートに変換またはその反対の操作により、データを再構成する必要があるかもしれません。このページでは、行列をワークシートに変換する時のサンプルと情報を紹介します。なお、反対の操作は、ワークシートを行列に変換するを確認してください。

行列からワークシート

行列のデータは、m2w X ファンクションでワークシートに変換できます。この X ファンクションは、X/Y マッピング有り、または無しで、直接データをワークシートに変換できます。または、データを再配置してワークシートの XYZ 列に入るように変換します。

次のサンプルは行列をワークシートに変換し、変換後のデータ形式により、異なる方法でグラフを作成する方法を示しています。

```
// 新しい行列ブックを作成
win -t matrix;
// 行列の次数と X/Y 値をセット
mdim cols:=21 rows:=21 x1:=0 x2:=10 y1:=0 y2:=100;
// 行列 X/Y 値を表示
page.cntrl = 2;
// 行列 Z 値をセット
msetvalue formula:="nlf_Gauss2D(x, y, 0, 1, 5, 2, 50, 20)";
// 行列ウィンドウ名を保持
%P = %H;
// 直接法で行列をワークシートに変換
m2w ycol:=1 xlabel:=row1;
// 仮想行列を使ってワークシートからグラフを作成
plot_vm irng:=1! xy:=xacross ztitle:=MyGraph type:=242 ogl:=<new template:=cmap>;
// 行列を XYZ ワークシートデータに変換
sec -p 2;
win -a %P;
m2w im:=!1 method:=xyz;
// 3D 散布図を作成
worksheet -s 3;
worksheet -p 240 3D;
```

行列データを直接ワークシートセルに変換した場合、仮想行列 の機能を使用してデータをプロットできます。

11 グラフ作成

この章で以下の項目を説明します

- グラフを作成する
- グラフのフォーマット
- レイヤの管理
- グラフィックオブジェクトの作成とアクセス

Origin のグラフ作成機能は幅広く、詳細の設定ができることは良く知られています。強力かつ柔軟な Origin のグラフ機能には、ユーザインターフェースからだけでなく、スクリプトからも簡単にアクセスすることができます。これらのセクションでは、LabTalk スクリプトを使って、グラフの作成および編集を行うサンプルを提供します。

11.1. グラフを作成する

グラフ作成は、おそらく Origin で最も頻繁に実行される操作です。このセクションは、LabTalk スクリプトから直接グラフを作成する次の 2 つの X ファンクション、**plotxy** および **plotgroup** の例を示します。一度、プロット が作成されると、ページ、レイヤ、軸 オブジェクトのようなオブジェクトプロパティを使って、set コマンド で グラフを整えることができます。

内容

11.1.1. PLOTXY X ファンクションを使ってグラフを作成する

plotxy は一般的な目的のグラフを作成するのに使用します。これは グラフウィンドウを作成したり、グラフテンプレートにプロットしたり、新しい グラフレイヤにプロットするのに使用されます。すべての X ファンクションと共通のシンタックスがあります。

plotxy オプション 1:=*optionValue* オプション 2:=*optionValue* ... オプション *N*:=*optionValue*

利用可能なオプションと値は X-Function ヘルプの **plotxy** にまとめられています。あまり直観的ではないので、**plot** オプションとその一般的な値はこ以下にまとめられています。

plot:=	グラフタイプ
200	折れ線
201	散布図
202	線+シンボル
203	列

plot オプションで利用可能なすべての値は、プロットタイプ ID にあります。

X Y データをプロットする

X および Y を参照する入力 XYRange

次のサンプルは、アクティブワークシートの最初の 2 列を、最初の列を X として、2 列目を Y として、折れ線グラフでプロットします。

```
plotxy iy:=(1,2) plot:=200;
```

Input XYRange referencing just the Y

次のサンプルは、アクティブワークシートの 2 列目を、それに結びついた X に対する Y として、折れ線グラフでプロットします。明示的に X を指定しないとき、Origin はワークシート内の Y と結びついている X 列を使用します。結びついた X 列がない場合には <自動> X が使われます。デフォルトで、<自動> X は行番号です。

```
plotxy iy:=2 plot:=200;
```

X YY データをプロットする

次のサンプルは、Book1, Sheet1 の最初の 3 列を、最初の列を X として、2 列目と 3 列目を Y として、グループ化した散布図でプロットします。

```
plotxy iy:=[Book1]Sheet1!(1,2:3) plot:=201;
```

XY XY データをプロットする

次のサンプルは、アクティブワークシートの最初の 4 列を、最初の列を X とし、2 列目を 1 列目と結びついている Y として、さらに 3 列目を X とし、4 列名を 3 列目と結びついている Y として、グループ化した線＋シンボルグラフでプロットします。

```
plotxy iy:=((1,2),(3,4)) plot:=202;
```

ワークシート列の XY 属性を使ってプロットする

次のサンプルは、アクティブワークシートのすべての列をワークシートの列プロット属性を使って縦棒グラフでプロットします。'?' はワークシートの XY 属性を使うことを示しています。'1:end' はすべての列をプロットすることを示しています。

```
plotxy iy:=(?,1:end) plot:=203;
```

列の一部分をプロットする

次のサンプルは、アクティブワークシートのすべての列のうち 1 行目から 12 行目までを、グループ化した折れ線グラフでプロットします。

```
plotxy iy:=(1,2:end)[1:12] plot:=200;
```

Note:部分範囲についての詳細は、X 値を使用した部分範囲指定を確認してください。

グラフテンプレートにプロットする

次のサンプルは、アクティブワークシートの最初の列を theta(X)、2 番目の列を r(Y) として、*極座標*グラフにプロットし、グラフウィンドウの名前を *MyPolarGraph* とします。

```
plotxy (1,2) plot:=192 ogl:=[<new template:=polar name:=MyPolarGraph>];
```

既存のグラフレイヤにプロットする

次のサンプルは、アクティブワークシートの 10 列目から 20 列目までを列の XY 属性を使って、Graph1 の 2 番目のレイヤにプロットします。これらの列は、すべて Y 列にすることができ、ワークシート内の結びついた X 列に対してプロットされます。

```
plotxy iy:=(?,10:20) ogl:=[Graph1]2!;
```

新しいグラフレイヤを作成する

次のサンプルは、アクティブグラフウィンドウに新しく下 X 軸左 Y 軸のレイヤを加え、Book1、Sheet2 から最初の列を X として 3 列目を Y として、折れ線グラフでプロットします。グラフウィンドウがアクティブで出力グラフレイヤを指定しない場合には、新しいレイヤが作成されます。

```
plotxy iy:=[Book1]Sheet2!(1,3) plot:=200;
```

二重 Y グラフの作成

```
// データファイルをインポート
string fpath$ = "Samples\Import and Export\S15-125-03.dat";
string fname$ = system.path.program$ + fpath$;
impASC;

// ブックとシート名を記憶
string bkname$ = page.name$;
string shname$ = layer.name$;

// 1 番目と 2 番目の列を X および Y としてプロット
// ワークシートがアクティブなら列範囲を指定するだけ
plotxy iy:=(1,2) plot:=202 ogl:=[<new template:=doubleY>];
```

```
// 2 番目のレイヤに最初と 3 番目の列をプロット
// グラフウィンドウがアクティブなので
// ブックとシートを指定
plotxy iy:=[bkname$]shname$!(1,3) plot:=202 ogl:=2;
```

11.1.2. PLOTGROUP X ファンクションを使ってグラフグループを作成する

グループ化変数(データセット)に従って、**plotgroup** X ファンクションはページ、レイヤ、データプロットのプロットを作成します。正しく動作させるには、ワークシートを最初にグラフグループデータでソートし、次にレイヤグループデータ、最後にデータプロットグループデータでソートします。

このサンプルはグループでプロットする方法を示しています。

```
// サンプルデータをセット
fn$ = system.path.program$ + "Samples\Statistics\body.dat";
newbook;
impASC fn$;       // 新しいワークブックにインポート

// ワークシートをソート--ソートはとても重要!
wsort bycol:=3;

// グループ毎にプロット
plotgroup iy:=(4,5) pgrp:=Col(3);
```

この次のサンプルは、1 つのグループでグラフウィンドウを作成し、2 番目のグループでグラフレイヤを作成します。

```
// サンプルデータを取り入れる
fn$ = system.path.program$ + "Samples\Graphing\Categorical Data.dat";
newbook;
impASC fn$;
// ソート
dataset sortcol = {4,3}; // drug、gender 順で
dataset sortord = {1,1}; // ソート（昇順）する
wsort nest:=sortcol ord:=sortord;
// 性別ごとにグラフをレイヤで分けて各薬をプロット
plotgroup iy:=(2,1) pgrp:=col(drug) lgrp:=col(gender);
```

Note:各グループ変数は任意です。例えば、ページグループとデータグループを省略することで、レイヤ内のデータを統合する 1 つのグループ変数を使うことができます。使用するオプションがどれであっても同じソート順にすることが重要です。

11.1.3. Worksheet -p コマンドを使って 3D グラフを作成する

3D グラフを作成するには、Worksheet (コマンド) (-p スイッチ)を使います。

最初に、単純な 3D 散布図を作成します。

```
// 新しいブックを作成
newbook r:=bkn$;
// bkn$でスクリプトを実行
win -o bkn$ {
        // サンプルデータをインポート
        string fname$ = system.path.program$ +
```

```
                "\samples\Matrix Conversion and Gridding" +
                "\XYZ Random Gaussian.dat";
        impasc;
        // 新しいブック名を保存
        bkn$ = %H;
        // 列タイプを Z に変更
        wks.col3.type = 6;
        // 列 3 を選択
        worksheet -s 3;
        // 3D 散布図を"3d"という名前のテンプレートでプロット
        worksheet -p 240 3d;
};
```

また、3D カラーマップや 3D ワイヤーフレームも作成することができます。3D グラフは ワークシート または行列のどちらかから作成することができます。プロットするまえに **グリッディング** を行う必要があるかもしれません。

上記のサンプルのあと次のスクリプトを実行し、行列から 3D ワイヤーフレームを作成します。

```
win -o bkn$ {
        // Shepard 法でグリッディング
        xyz_shep 3;
        // 3D ワイヤーフレームグラフを作図
        worksheet -p 242 wirefrm;
};
```

11.1.4. 仮想行列から 3D および等高線グラフを作成する

Origin では、plotvm X-ファンクションにより、ワークシートから 3D カラーマップ、等高線図、3D ワイヤーフレームといった 3D グラフを作成することもできます。この関数は、仮想行列を作成し、行列のような形式からプロットします。例えば、

```
// 新しいワークブックを作成し、サンプルデータをインポート
newbook;
string fname$=system.path.program$ + "Samples\Graphing\VSurface 1.dat";
impasc;
// 仮想行列としてシート全体を扱い、カラーマップ曲面プロットを作成
plotvm irng:=1! format:=xacross rowpos:=selrow1 colpos:=selcol1
 ztitle:="VSurface 1" type:=242 ogl:=<new template:=cmap>;
// X 軸スケールを対数に変更
// 仮想行列から作成された 3D グラフに対してサポートされる非線形の軸タイプ
LAYER.X.type=2;
```

11.2. グラフのフォーマット

内容

11.2.1. グラフウィンドウ

グラフウィンドウは、**Page** (オブジェクト)に結びついたビジュアルページで構成されます。各々のグラフページは、**layer** オブジェクトと関連づけられる、少なくとも 1 つのビジュアルページを含んでいます。グラフレイヤには、**layer** オブジェクトのサブオブジェクトである **layer.x** および **layer.y** オブジェクトと結びついた 1 セットの XY 軸を含みます。

グラフページまたはグラフレイヤにマッピングされた範囲変数がある場合、**page** や **layer** という言葉の変わりに、範囲変数名を使うことができます。

11.2.2. ページプロパティ

page オブジェクトは、アクティブグラフウィンドウのプロパティにアクセスしたり、修正したりできます。このオブジェクトのすべてのプロパティの一覧を出力するには

```
page.=
```

一覧は数値とテキストのプロパティの両方を含みます。テキスト(文字列)のプロパティ値をセットするとき、プロパティ名の後に **$** 記号を付けます。

アクティブウィンドウのショートネームを変更するには

```
page.name$="Graph3";
```

アクティブウィンドウのロングネームを変更するには

```
page.longname$="This name can contain spaces";
```

page オブジェクトの代わりに範囲変数を使ってグラフのプロパティまたは属性を変更することもできます。範囲変数を使うことの利点は、目的のグラフがアクティブであるかどうかに関係なく操作できることです。

以下のサンプルは、範囲変数を使って、名前で目的のグラフを指しアクティブグラフレイヤをレイヤ 2 にセットします。宣言されると、範囲変数が **page** の代わりに使うことができます。

```
//グラフを指す Range 変数を作成
range rGraph = [Graph3];
//範囲はページオブジェクトのプロパティ
rGraph.active=2;
```

11.2.3. レイヤのプロパティ

layer オブジェクトは、グラフレイヤのプロパティにアクセスしたり、修正したりできます。

グラフレイヤのサイズをセットするには

```
//レイヤ領域の単位を cm にセット
layer.unit=3;
//幅をセット
layer.width=5;
//高さをセット
layer.height=5;
```

レイヤの背景色を塗る

laycolor X ファンクションはレイヤの背景色を塗りつぶすのに使います。関数に渡される色指定の値は、作図の詳細で表示されるのと同じ Origin のカラーリスト に対応しています(1=黒, 2=赤, 3=緑など)。

レイヤ 1 の背景色を緑にするには

```
laycolor layer:=1 color:=3;
```

スピードモードのプロパティをセット

speedmode X ファンクションは、レイヤのスピードモードのプロパティをセットするのに使います。

凡例を更新

legendupdate X ファンクションは、ページ/レイヤにあるグラフの凡例を更新したり、再構築するのに使います。

11.2.4. 軸のプロパティ

layer オブジェクトの **layer.x** および **layer.y** サブオブジェクトは、軸のプロパティを修正するのに使うことができます。

アクティブレイヤの X スケールを修正するには

```
//スケールを Log10 にセット
layer.x.type = 2;
//開始値をセット
layer.x.from = .001;
//終了値をセット
layer.x.to = 1000;
//増分値をセット
layer.x.inc = 2;
```

Y スケールを操作するには、上記スクリプトの x を y に変更するだけです。アクティブでないレイヤを操作するには場合、layer*N*.x.from.のようにレイヤインデックスを指定します。サンプルは次の通りです。layer3.y.from = 0;

Axis コマンドは、軸ダイアログの設定にアクセスするのに使います。

Book1 の Sheet1 にある col(A)に対して col(B)を col(C)のテキスト付きでプロットし、X 軸ラベルを列 C の値に変更するには、次のようにします。

```
range aa = [Book1]Sheet1!col(C);
axis -ps X T aa;
```

11.2.5. データプロットのプロパティ

Set (コマンド) は、データプロットの属性を変更するのに使います。次のサンプルは、同じデータプロットのプロパティを数回変更し、**Set** コマンドがどのように動作するのかを示します。スクリプトでは、プロットスタイルを変更する前に、sec コマンドを使って、1 秒間停止します。

```
// いくつかデータを作成
newbook;
col(a) = {1:5};
col(b) = col(a);
// 散布図を作成
plotxy col(b);

// シンボルサイズをセット
// %C はアクティブデータセット
sec -p 1;
set %C -z 20;
// シンボルの形状をセット
sec -p 1;
set %C -k 3;
// シンボルの色をセット
sec -p 1;
set %C -c color(blue);
// シンボルを接続
sec -p 1;
set %C -l 1;
// グラフ線の色を変更
sec -p 1;
set %C -cl color(red);
// 線の太さを 4 ポイントにセット
sec -p 1;
set %C -w 2000;
// 実線を破線に変更
sec -p 1;
set %C -d 1;
```

2 つのレイヤを持つテンプレート, *DoubleY*, にプロットし、2 番目のレイヤのデータプロットのスタイルをセットする別のサンプルがあります。

```
// データをインポート
newbook;
string fn$=system.path.program$ + "Samples\Curve Fitting\Enzyme.dat";
impasc fname:=fn$;
//アクティブワークシート範囲を宣言
range rr = !;
//テンプレートにプロット
plotxy iy:=(1,2) plot:=200 ogl:=[<new template:=DoubleY>];
//アクティブグラフの 2 番目のレイヤにプロット、これは上記の折れ線から作成したグラフ
plotxy iy:=%(rr)(1,3) plot:=200 ogl:=2!;
//レイヤ 2 の最初のデータプロットの範囲を宣言
range r2 = 2!1;
//線を破線にセット
set r2 -d 1;
```

11.2.6. 凡例とラベル

凡例とラベルのフォーマットについては、グラフィカルオブジェクトの作成とアクセス をご覧下さい。

11.3. レイヤを管理する

内容

11.3.1. 区分プロットを作成する

newpanel X ファンクションは、n x m のレイヤ配置を持つ新しいグラフを作成します。

6 区分グラフを作成します。

次のサンプルは、2 列 3 行で配置した 6 つのレイヤを持つ新しいグラフを作成します。この関数は、どのウィンドウがアクティブかに関係なく実行できます。

```
newpanel col:=2 row:=3;
```

X ファンクションを使うとき、値の割り当てに変数名を使う必要はありませんが、col:= と row:= で明示的に指定するとコードが読みやすくなります。入力する字数を減らすため、上記のコードではなく、次のように入力することもできます。

```
newpanel 2 3;
```

6 区分グラフを作成しプロットする

次のサンプルは、新しいワークブックにいくつかデータをインポートし、6 つのレイヤを持つ新しいグラフウィンドウを作成し、2 列 3 行で配置し、各レイヤ(区分)をループして、インポートしたデータをプロットします。

```
// 新しいワークブックを作成
newbook;
```

```
// ファイルをインポート
path$ = system.path.program$ + "Samples\Graphing\";
fname$ = path$ + "waterfall2.dat";
impasc;

// 新しいパネルが%H を変更するのでワークブック名を保存
string bkname$=%H;

// 2*3 区分グラフを作成
newpanel 2 3;

// データをプロット
for (ii=2; ii<8; ii++)
{
        plotxy iy:=[bkname$]1!wcol(ii) plot:=200 ogl:=$(ii-1);
}
```

11.3.2. グラフウィンドウにレイヤを追加する

layadd X ファンクションは、グラフウィンドウに新しいレイヤ を作成/追加します。この関数は、**グラフ操作:新規レイヤ(軸) メニュー**に相当します。

プログラムで、グラフにレイヤを追加することは一般的ではありません。事前にグラフテンプレートを作成し、**plotxy** X ファンクションを使って、グラフテンプレートにプロットすることをお勧めします。

次のサンプルは、独立した右 Y 軸スケールを追加します。右 Y 軸のみを表示する新しいレイヤを追加します。これは寸法がリンクされ、レイヤが追加された時に、X 軸が現在のアクティブレイヤにリンクされます。新しく追加したレイヤがアクティブレイヤになります。

```
layadd type:=rightY;
```

11.3.3. レイヤを配置する

layarrange X ファンクションは、グラフページ上のレイヤを配置するのに使います。

プログラムで、グラフレイヤを配置することは一般的ではありません。事前にグラフテンプレートを作成し、**plotxy** X ファンクションを使って、グラフテンプレートにプロットすることをお勧めします。

次のサンプルは、アクティブグラフに既存レイヤを 2 行 3 列で配置します。アクティブグラフに 6 つのレイヤが無くても、新しいレイヤは追加されません。存在しているレイヤのみが配置されます。

```
layarrange row:=2 col:=3;
```

11.3.4. レイヤを移動する

laysetpos X ファンクションは、ページの相対位置で、グラフ内の 1 つ以上のレイヤの位置をセットするのに使われます。

次のサンプルは、アクティブグラフウィンドウのすべてのレイヤを左に整列し、ページの左側から 15%の位置にセットします。

```
laysetpos layer:="1:0" left:=15;
```

11.3.5. 2 つのレイヤを入れ替える

layswap X ファンクションは、2 つのグラフレイヤの位置と場所を入れ替えます。名前と番号でレイヤを参照できます。

次のサンプルは、レイヤ 1 と 2 のページの位置をインデックス番号を使って入れ替えます。

```
layswap igl1:=1 igl2:=2;
```

次のサンプルは、Layer1 および Layer2 という名前のレイヤの位置を名前を入れ替えます。

```
layswap igl1:=Layer1 igl2:=Layer2;
```

レイヤは、レイヤ管理ツールだけでなく、作図の詳細ダイアログでも名前を変更することができます。レイヤ管理ツールでは、レイヤ選択リストにある名前をダブルクリックして、名前を変更します。作図の詳細ダイアログの左側のパネルで、レイヤを 2 回クリックして、名前を変更します。

LabTalk から名前を変更するには、layer*n*.name$を使います。*n* はレイヤのインデックスです。例えば、以下のようにしてレイヤ 1 の名前を Power に変更します。layer1.name$="Power";

11.3.6. レイヤを整理する

layalign X ファンクションは、元のレイヤ/参照レイヤに対して 1 つ以上のレイヤを整列するのに使います。

次のサンプルは、アクティブグラフレイヤのレイヤ 1 とレイヤ 2 を下辺に合わせて整列します。

```
layalign igl:=1 destlayer:=2 direction:=bottom;
```

次のサンプルは、アクティブグラフレイヤのレイヤ 1 とレイヤ 2, 3, 4 を左辺に合わせて整列します。

```
layalign igl:=1 destlayer:=2:4 direction:=left;
```

次のサンプルは、レイヤ 1 に関して、グラフ 3 のすべてのレイヤを左辺に合わせて整列します。2:0 という表記は、グラフ内のレイヤ 2 から最後のレイヤまでのすべてのレイヤという意味です。

```
layalign igp:=graph3 igl:=1 destlayer:=2:0 direction:=left;
```

11.3.7. レイヤをリンクする

laylink X ファンクションは、お互いのレイヤをリンクするのに使用します。レイヤ領域/位置だけでなく、軸スケールもリンクします。

次のサンプルは、アクティブグラフのすべてのレイヤの X 軸をレイヤ 1 の X 軸にリンクします。単位は、リンクしたレイヤの%にセットされます。

```
laylink igl:=1 destlayers:=2:0 XAxis:=1;
```

11.3.8. レイヤ単位をセットする

laysetunit X ファンクションは、複数のレイヤのレイヤ領域の単位をセットします。

11.4. グラフィックオブジェクトの作成とアクセス

内容

11.4.1. グラフィックオブジェクトの作成

テキストや四角形、線といったグラフィックオブジェクトを追加します。

次のサンプルでは、アクティブなグラフに四角形を追加する方法を紹介します。他のグラフオブジェクトタイプについては、oc_const.h ファイル内の GROT_* (例:GROT_TEXT, GROT_LINE, GROT_POLYGON) を確認してください。

```
GraphLayer gl = Project.ActiveLayer();
string strName = "MyRect";
GraphObject goRect = gl.CreateGraphObject(GROT_RECT, strName);
```

現在のグラフウィンドウにテキストラベルを追加します。

```
GraphLayer gl = Project.ActiveLayer();
GraphObject go = gl.CreateGraphObject(GROT_TEXT, "MyText");
go.Text = "This is a test";
```

以下のサンプルでは、グラフに矢印を追加する方法を示します。矢印のオブジェクトタイプは、GROT_LINE で、線と同じです。線と矢印ともに、データポイントの数は 2 つ必要です。

```
GraphPage gp;
gp.Create();
GraphLayer gl = gp.Layers();

string strName = "MyArrow"; // グラフオブジェクトの名前
GraphObject go = gl.CreateGraphObject(GROT_LINE, strName);

go.Attach = 2; // 接続先はレイヤとスケール

Tree tr;
tr.Root.Dimension.Units.nVal = 5; // 単位はスケールにセット

// スケール値で位置をセット
vector vx = {2, 6};
vector vy = {6, 2};
tr.Root.Data.X.dVals = vx;
tr.Root.Data.Y.dVals = vy;
```

```
tr.Root.Arrow.Begin.Style.nVal = 0;
tr.Root.Arrow.End.Style.nVal = 1;

if( 0 == go.UpdateThemeIDs(tr.Root) )
{
        go.ApplyFormat(tr, true, true);
}
```

次のサンプルでは、グラフに曲線矢印を追加する方法を示します。曲線矢印は、データポイントの数は 4 つ必要です。

```
GraphPage gp;
gp.Create();
GraphLayer gl = gp.Layers();

string strName = "MyArrow"; // グラフオブジェクトの名前
GraphObject go = gl.CreateGraphObject(GROT_LINE4, strName);

go.Attach = 2; // 接続先はレイヤとスケール

Tree tr;
tr.Root.Dimension.Units.nVal = 5; // 単位をスケールにセット

// スケール値で位置をセット
vector vx = {2, 4, 6, 5};
vector vy = {7, 6.9, 6.8, 2};
tr.Root.Data.X.dVals = vx;
tr.Root.Data.Y.dVals = vy;

tr.Root.Arrow.Begin.Style.nVal = 0;
tr.Root.Arrow.End.Style.nVal = 1;

if( 0 == go.UpdateThemeIDs(tr.Root) )
{
        go.ApplyFormat(tr, true, true);
}
```

11.4.2. プロパティ設定

テキストフォントや、色、線の幅などのグラフィカルオブジェクトのプロパティを設定します。

```
// グラフオブジェクトの色とフォントを設定
GraphLayer gl = Project.ActiveLayer();
GraphObject goText = gl.GraphObjects("Text");
goText.Text = "This is a test";
goText.Attach = 2; // レイヤスケールに接続

Tree tr;
tr.Root.Color.nVal = SYSCOLOR_RED; // テキストの色
tr.Root.Font.Bold.nVal = 1;
tr.Root.Font.Italic.nVal = 1;
tr.Root.Font.Underline.nVal = 1;
tr.Root.Font.Size.nVal = 30; // テキストのフォントサイズ
```

```
if( 0 == goText.UpdateThemeIDs(tr.Root) )
{
        bool bRet = goText.ApplyFormat(tr, true, true);
}
```

11.4.3. 位置とサイズを設定

```
GraphLayer gl = Project.ActiveLayer();
GraphObject go = gl.GraphObjects("Rect");
go.Attach = 2; // レイヤスケールに接続

// テキストオブジェクトをレイヤの左上に移動
Tree tr;
tr.Root.Dimension.Units.nVal = UNITS_SCALE;
tr.Root.Dimension.Left.dVal = gl.X.From; // 左
tr.Root.Dimension.Top.dVal = gl.Y.To/2; // 上
tr.Root.Dimension.Width.dVal = (gl.X.To - gl.X.From)/2; // 幅
tr.Root.Dimension.Height.dVal = (gl.Y.To - gl.Y.From)/2; // 高さ

if( 0 == go.UpdateThemeIDs(tr.Root) )
{
        bool bRet = go.ApplyFormat(tr, true, true);
}
```

11.4.4. 接続先プロパティを更新

接続先には、ページ、レイヤ枠、レイヤスケールの 3 つの選択肢があります。

```
// グラフオブジェクトに異なるオブジェクトを接続
// 0 はレイヤ。レイヤを移動するとグラフオブジェクトも一緒に移動
// 1 はページ。レイヤを移動してもグラフオブジェクトに影響はない
// 2 はレイヤスケール。スケールを変更するとグラフオブジェクトの位置が
// 対応して移動.
go.Attach = 2;
```

11.4.5. 無効なプロパティの取得と設定

```
// 移動可能、選択可能といった無効なプロパティを確認
Tree tr;
tr = go.GetFormat(FPB_OTHER, FOB_ALL, true, true);
DWORD   dwStats = tr.Root.States.nVal;

// 垂直と水平移動を確認
// プロパティの詳細は、oc_const.h ファイルの GOC_* を確認
if( (dwStats & GOC_NO_VMOVE) && (dwStats & GOC_NO_HMOVE) )
{
        out_str("This graph object cannot be move");
}
```

11.4.6. プログラム制御

```
// 1.線を追加
GraphLayer gl = Project.ActiveLayer();
GraphObject go = gl.CreateGraphObject(GROT_LINE);
go.Attach = 2; // 接続先をレイヤスケールにセット
go.X = 5; // 初期位置 X = 5 にセット

// 2.線図のプロパティをセット
Tree tr;
tr.Root.Direction.nVal = 2; // 1 は水平、2 は垂直
tr.Root.Span.nVal = 1; // レイヤまで間隔
tr.Root.Color.nVal = SYSCOLOR_RED; // 線の色

if( 0 == go.UpdateThemeIDs(tr.Root) )
{
        go.ApplyFormat(tr, true, true);
}

// 3.イベントモードと LT スクリプトをセット
// 線移動すると線の位置の X 値を印字
Tree trEvent;
trEvent.Root.Event.nVal = GRCT_MOVE;// 詳細は、oc_const.h の GRCT_*
trEvent.Root.Script.strVal = "type -a $(this.X)";

if( 0 == go.UpdateThemeIDs(trEvent.Root) )
{
        go.ApplyFormat(trEvent, true, true);
}
```

11.4.7. 凡例の更新

凡例は、グラフウィンドウ内にある"Legend"という名前のグラフィックオブジェクトです。データプロットを追加/削除後、**legend_update** 関数を使用して、現在データプロットに基づいた凡例に更新することが可能です。

```
// 凡例の更新の簡単な使用方法
// Origin C ヘルプでは、この関数を検索して引数や
// 使用例について確認可能
legend_update(gl); // gl は、GraphLayer オブジェクト
```

11.4.8. グラフに表オブジェクトを追加

```
// 1.表テンプレートとともにワークシーとを作成
Worksheet wks;
wks.Create("Table", CREATE_HIDDEN);
WorksheetPage wksPage = wks.GetPage();

// 2.表のサイズを取得し、テキストを埋める
wks.SetSize(3, 2);
wks.SetCell(0, 0, "1");
wks.SetCell(0, 1, "Layer 1");
```

```
wks.SetCell(1, 0, "2");
wks.SetCell(1, 1, "Layer 2");

wks.SetCell(2, 0, "3");
wks.SetCell(2, 1, "Layer 3");

//3.グラフへのリンクとして表を追加
GraphLayer gl = Project.ActiveLayer();
GraphObject grTable = gl.CreateLinkTable(wksPage.GetName(), wks);
```

12 インポート

この章で以下の項目を説明します

- データをインポートする
- 画像のインポート

Origin には、ASCII, CSV, Excel, National Instruments DIAdem, pCLAMP といった、多様なファイル形式のデータをインポートする X ファンクションがあります。各ファイル形式の X ファンクションは、インポートするファイルの名前をブックまたはシートにするといった共通の設定に加え、そのファイル形式固有のオプションを提供しています。

インポートに関するすべての X ファンクションは、**imp** で始まる名前を持ちます。下表はこれらの X ファンクションのリストです。すべての X ファンクションのヘルプ情報が使用でき、X ファンクション名に**-h** オプションを付けて実行すれば、スクリプトウィンドウまたはコマンド行に表示することができます。例えば、スクリプトウィンドウに **impasc -h** と入力すると、コマンドのすぐ下にヘルプを表示します。

X ファンクション名	簡単な説明
impASC	ASCII ファイルをインポートします。
impBin2d	バイナリ 2D 配列ファイルのインポート
impCSV	csv ファイルのインポート
impDT	データトランスレーション Version 1.0 ファイルをインポートします。
impEP	EarthProbe ファイル (EPA)をインポートします。EPA ファイルだけが EarthProbe データに対してサポートされます。
impExcel	Microsoft Excel 97-2007 ファイルをインポートします。
impFamos	Famos Version 2 ファイルをインポートします。
impFile	事前に定義したフィルタを使ってファイルをインポートします。
impHEKA	HEKA (dat)ファイルをインポートします。
impIgorPro	WaveMetrics IgorPro (pxp, ibw)ファイルをインポートします。
impImage	画像ファイルをインポートします。
impinfo	ファイルインポートに関する情報を読み取ります。
impJCAMP	JCAMP-DX Version 6 ファイルをインポートします。

impJNB	SigmaPlot ファイル (JNB)をインポートします。SigmaPlot 8.0 以下のバージョンをサポートしています。
impKG	カレイダグラフのファイルをインポートします。
impMatlab	Matlab ファイルのインポート
impMDF	ETAS INCA MDF (DAT, MDF)ファイルをインポートします。INCA 5.4 (ファイル version 3.0)をサポートしています。
impMNTB	Minitab ファイル(MTW)またはプロジェクト(MPJ)をインポートします。Minitab 13 以前のバージョンをサポートします。
impNetCDF	netCDF ファイルをインポートします。バージョン 3.1 以下のファイルをサポートします。
impNIDIAdem	National Instruments DIAdem 10.0 データファイルをインポートします。
impNITDM	National Instruments の TDM および TDMS ファイル(TDMS は日時フォーマットをサポートしていない)をインポートします。
impODQ	*.ODQ ファイルをインポートします。
imppClamp	pCLAMP ファイルをインポートします。pClamp 9 (ABF 1.8 ファイル形式)および pClamp 10 (ABF 2.0 ファイル形式)をサポートしています。
impSIE	Import nCode Somat SIE 0.92 file
impSPC	Thermo ファイルをインポートします。
impSPE	Princeton Instruments (SPE)ファイルをインポートします。Minitab 2.5 以前のバージョンをサポートします。
impWav	ウェーブ形式のオーディオファイルをインポートします。
reimport	現在のファイルを再インポートします。

X ファンクションの形式で自身のインポートルーチンを記述することもできます。ユーザが作成した X ファンクション名が **imp** から始まっていると、それはシステムフォルダ、ユーザファイルフォルダ、グループフォルダの**\X-Functions\Import and Export** サブフォルダに配置され、**ファイル**:**インポート**メニューに表示されます。

これらのセクションでは、データ、グラフ、イメージをインポートする関数のスクリプトでの利用例を提供します。

12.1. データのインポート

次のサンプルは、外部ファイルからデータをインポートする X ファンクションの使用方法を説明するものです。サンプルは ASCII ファイルをインポートしますが、目的のファイルタイプ(例えば csv や Matlab)に応じて置き換えることができます。シンタックスやサポートしているコマンドは同じです。これらのサンプルは Origin サンプルファイルをインポートするので、このサンプルをスクリプトウィンドウまたはコマンドウィンドウに直接入力したり、貼り付けて、実行することができます。

内容

12.1.1. ASCII データファイルをワークシートまたは行列にインポートする

このサンプルは、ASCII ファイル(この場合、拡張子 ***.txt**)をアクティブワークシートや行列にインポートします。別の X ファンクション **findfiles** を使って、他の多くのファイルを含むディレクトリ(変数 **path$**に割り当て)にある特定のファイルを探すことができます。findfiles X ファンクションの出力は、希望のファイル名を含む文字列変数で、デフォルトで **fname$**という変数名に割り当てます。**impASC** X ファンクションのデフォルトの入力引数も **fname$**という変数名となっています。

```
string path$ = system.path.program$ + "Samples\Import and Export\";
findfiles ext:=matrix_data_with_xy.txt;
impASC;
```

12.1.2. オプションを指定して ASCII データをインポートする

このサンプルは、**impASC** X ファンクションの多くの詳細オプションの使用方法を説明します。これはファイルを新しいブックにインポートし、**impASC** X ファンクションのオプションを使って、名前を変更します。すべては **impASC** の呼び出しの一部であるのでセミコロンは 1 つしかありません(すべての**オプション**を指定した後にセミコロン)。

```
string fn$=system.path.program$ + "Samples\Spectroscopy\HiddenPeaks.dat";
impasc fname:=fn$
options.ImpMode:=3                          /* 新しいブックで開始 */
options.Sparklines:=0                       /* スパークラインをオフ */
options.Names.AutoNames:=0                  /* 自動名前変更をオフ */
options.Names.FNameToSht:=1                 /* シートをファイル名に変更 */
options.Miscellaneous.LeadingZeros:=1;      /* 先行ゼロを消去 */
```

12.1.3. 複数データファイルをインポートする

このサンプルでは、複数のデータファイルを新しいワークブックにインポートするものです。各ファイルに対して新しいワークシートを作成します。

```
string fns, path$=system.path.program$ + "Samples\Curve Fitting\";
findfiles f:=fns$ e:="step1*.dat";    // 'path$'内でマッチするファイルを検索
int n = fns.GetNumTokens(CRLF);       // 見つかったファイル数
string bkName$;
newbook s:=0 result:=bkName$;
impasc fname:=fns$                    // impasc には多くのオプションがある
options.ImpMode:=4                    // 新しいシートで開始
options.Sparklines:=2                    // 50 列以下ならスパークラインを追加
options.Cols.NumCols:=3        // 最初の 3 列のみをインポート
options.Names.AutoNames:=0           // 自動名前変更をオフ
options.Names.FNameToBk:=0            // ワークブック名を変更しない
options.Names.FNameToSht:=1           // シートをファイル名に変更
options.Names.FNameToShtFrom:=4       // 4 文字目以降のファイル名を削除
options.Names.FNameToBkComm:=1        // ファイル名をワークブックコメントに追加
options.Names.FNameToColComm:=1       // ファイル名を列コメントに追加
options.Names.FPathToComm:=1          // ファイルパスをコメントに含める
orng:=[bkName$]A1!A[1]:C[0] ;
```

12.1.4. ASCII ファイルをワークシートにインポートして、行列に変換する

このサンプルでは、**impASC** と一緒に動作する 2 つの役立つ X ファンクションを示します。1 つは、**dlgFile** で、これはインポートするファイルを選択するダイアログを作成します。もう 1 つは **w2m** で、これはワークシートを行列に変換します。**w2m** X ファンクションは、最初の列に線形に増加する Y 値と最初の行に線形に増加する X 値が必要です。これを確認するには、**Samples\Import and Export** フォルダの **matrix_data_with_xy.txt** を使います。

```
dlgfile g:=ascii; // ファイルを開くダイアログ
impAsc; // 選択したファイルをインポート
// ワークシートを行列にする X ファンクション, 'w2m'を使い変換を行う
w2m xy:=0 ycol:=1 xlabel:="First Row" xcol:=1
```

12.1.5. Open コマンド

Origin にデータを入力する別の方法は、**Open** (コマンド)を使う方法です。

Open コマンドにはいくつかオプションがあり、そのうちの 1 つを使って、ファイルを開いてノートウィンドウに表示します。

open -n fileName [winName]

このスクリプト行は、ASCII ファイル *fileName* を**ノート** ウィンドウに開きます。任意の *winName* が指定されていない場合、新しい**ノート**ウィンドウが作成されます。

既存のファイルを使って操作するには、以下を行います。

```
%b = system.path.program$ + "Samples\Import and Export\ASCII simple.dat";
open -n "%b";
```

12.1.6. テーマとフィルタを使ってインポートする

テーマを使ったインポート

Origin GUI からインポートするとき、インポート設定を**テーマファイル**に保存できます。このようなテーマファイルは拡張子 *.OIS を持ち、Origin の**ユーザファイルフォルダ**(UFF)の**\Themes\AnalysisAndReportTable** サブフォルダに保存されます。そして、それらは **-t** オプションスイッチ付きの X ファンクションを使ってアクセスできます。指定したテーマファイルに保存された設定に従ってインポートが実行されます。

```
string fn$=system.path.program$ + "Samples\Spectroscopy\HiddenPeaks.dat";
// "My Theme.OIS" というテーマファイルがあることが前提
impasc fname:=fn$ -t "My Theme";
```

インポートウィザードフィルタファイルを使ったインポート

ASCII ファイルおよび単純形式のバイナリファイルのカスタムインポートは**インポートウィザード**を使って実行することができます。このツールは、ファイル名やヘッダから変数を抽出したり、インポートの最後にスクリプトを実行するといったインポートのカスタマイズを行うことができます。これは、インポートしたデータの後処理を行うのに利用できます。マウス操作で行ったすべての設定は、**インポートフィルタファイル**としてディスクに保存できます。このようなファイルは拡張子**.OIF** を持ち、複数の場所に保存できます。

インポートウィザードのインポートフィルタファイルを作成すると、**impfile** X ファンクションを使って、インポートフィルタにアクセスでき、インポートフィルタファイルに保存した設定を使ってカスタムインポートを実行できます。

```
string fname$, path$, filtername$;
// ファイルパスを指す
path$ = system.path.program$ + "Samples\Import and Export\";
// 仕様に合ったファイルを探す
findfiles ext:="S*.dat";
// インポートウィザードのインポートフィルタを指示
string str$ = "Samples\Import and Export\VarsFromFileNameAndHeader.oif";
filtername$ = system.path.program$ + str$;
// データフォルダにあるフィルタを使ってすべてのファイルをインポート
impfile location:=data;
```

12.1.7. データベースからインポートする

Origin はデータベースクエリに対して 4 つの機能を提供しています。データベースインポートの基本機能は、以下の Microsoft Office で提供される標準の Northwind データベースを使った例に示すように 2 つの関数にカプセル化されています。

```
// dbedit 関数はクエリと接続文字列を作成し
// 詳細をワークシートに接続
dbedit exec:=0
sql:="Select Customers.CompanyName, Orders.OrderDate,
[Order Details].Quantity, Products.ProductName From
((Customers Inner Join Orders On Customers.CustomerID = Orders.CustomerID)
Inner Join [Order Details] On Orders.OrderID = [Order Details].OrderID)
Inner Join Products On Products.ProductID = [Order Details].ProductID"
connect:="Provider=Microsoft.Jet.OLEDB.4.0;User ID=;
Data Source=C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb;
Mode=Share Deny None;Extended Properties="";
Jet OLEDB:System database="";
Jet OLEDB:Registry Path="";
Jet OLEDB:Database Password=***;
```

```
Jet OLEDB:Engine Type=5;
Jet OLEDB:Database Locking Mode=1;
Jet OLEDB:Global Partial Bulk Ops=2;
Jet OLEDB:Global Bulk Transactions=1;
Jet OLEDB:New Database Password="";
Jet OLEDB:Create System Database=False;
Jet OLEDB:Encrypt Database=False;
Jet OLEDB:Don't Copy Locale on Compact=False;
Jet OLEDB:Compact Without Replica Repair=False;
Jet OLEDB:SFP=False;Password="

// dbimport 関数でインポートを完了
dbimport;
```

2 つの追加の関数は、接続およびクエリ文字列の詳細を取り出し、プレビューと部分インポートを実行します。

名前	簡単な説明
dbEdit	ワークシートのクエリを作成、編集、ロード、削除
dbImport	特定のワークシートに保存したデータベースクエリを実行
dbInfo	ワークシートのデータベースクエリに含まれる SQL 文字列と接続文字列を読む
dbPreview	クエリの制限付き実行(デフォルトで 50 行)クエリが必要な情報を返すのかを確認するのに便利です。

12.2. 画像のインポート

ImpImageX ファンクションは、スクリプトを使って画像ファイルを Origin にインポートします。デフォルトで、画像ファイルは Origin に画像(RGB 値など)として保存されます。その画像をグレースケールに変換するオプションがあります。

Origin は複数ファイルのインポートをサポートしています。デフォルトで、複数の画像ファイルは、新しいレイヤを作成して、目的のページに追加されます。行列にインポートする場合、各行列のレイヤはインポートするファイル名になります。

内容

12.2.1. 行列に画像をインポートしデータに変換する

このサンプルは、1 つの画像ファイルを行列にインポートし、(RGB カラー)画像をグレースケールに変換し、それらを新しい行列に保存します。

```
newbook mat:=1;                 // 新しい行列ブックを作成
fpath$ = "Samples\Image Processing and Analysis\car.bmp";
string fname$ = system.path.program$ + fpath$;

// アクティブウィンドウにパス「fname$」のイメージをインポート
//(新しい行列ブック)
impimage;

// イメージをグレースケールに変換し、新しい行列に入れる
// 「type」はビット深度を指定。:  0=short 型 (2-byte/16-bit, デフォルト);
// 1=byte 型 (1-byte/8-bit)
img2m type:=byte;
```

12.2.2. 1 つの画像を行列にインポートする

このサンプルは新しい行列ブックに一連の *.TIF 画像をインポートします。上記で示した **img2m** X ファンクションの代替として、キーボードショートカット **Ctrl+Shift+d** と **Ctrl+Shift+i** を使って、行列データと画像データの表示を切り替えることができます。

```
newbook mat:=1;
fpath$ = "Samples\Image Processing and Analysis\";
string fns, path$ = system.path.program$ + fpath$;
// 「myocyte」からはじまるファイル名のファイルを検索
findfiles f:=fns$ e:="myocyte*.tif";
// 各ファイルを新しいシートにインポート (options.Mode = 4)
impimage options.Mode:=4 fname:=fns$;
```

12.2.3. 複数の画像を行列ブックにインポートする

このサンプルは、JPG 画像のフォルダを異なる行列ブックにインポートします。

```
string pth1$ = "C:\Documents and Settings\All Users\"
string pth2$ = "Documents\My Pictures\Sample Pictures\";
string fns, path$ = pth1$ + pth2$;
// すべての *.JPG ファイルを探す (デフォルトで、「path$」にある)
findfiles fname:=fns$ ext:="*.jpg";
// 見つかったファイル数を int 変数「n」に割り当てる
// 「CRLF」 ==> ファイルを「carriage-return line-feed」で分ける
int n = fns.GetNumTokens(CRLF);
string bkName$;
string fname$;
// すべてのファイルをループし、それぞれを新しい行列ブックにインポート
for(int ii = 1; ii<=n; ii++)
{
    fname$ = fns.GetToken(ii, CRLF)$;

    //新しい行列ページを作成
    newbook s:=0 mat:=1 result:=bkName$;
    //行列ページの最初のレイヤにイメージをインポート
    //デフォルトのファイル名は fname$
    impimage orng:=[bkName$]msheet1;
}
```

12.2.4. 画像をグラフレイヤにインポートする

既存のグラフレイヤに画像をインポートすることもできます。この画像は表示のためだけのものです。(行列に変換しなければ、データは表示されません。)

```
string fpath$ = "Samples\Image Processing and Analysis\cell.jpg";
string fn$ = system.path.program$ + fpath$;
insertImg2g fname:=fn$ ipg:=graph1;
```

13 エクスポート

この章で以下の項目を説明します

- ワークシートのエクスポート
- グラフのエクスポート
- 行列のエクスポート
- 動画のエクスポート

Origin は、データ、グラフ、イメージをエクスポートする X ファンクションを提供していますエクスポートに関するすべての X ファンクションの名前は、**exp** という文字が先頭に付きます。下表は、これらの X ファンクション一覧です。すべての X ファンクションのヘルプ情報が使用でき、X ファンクション名に-h オプションを付けて実行すれば、スクリプトウィンドウまたはコマンド行に表示することができます。例えば、スクリプトウィンドウに **expgraph -h** と入力すると、コマンドのすぐ下にヘルプを表示します。

X ファンクション名	簡単な説明
expASC	ワークシートデータを ASCII ファイルとしてエクスポートします。
expGraph	グラフを画像ファイルにエクスポートします。
expImage	アクティブなイメージを画像ファイルにエクスポート
expMatASC	行列データを ASCII ファイルとしてエクスポートします。
expNITDM	ワークブックを National Instruments 社の TDM および TDMS ファイルにエクスポートします。
expWAV	Microsoft PCM wav ファイルとしてデータをエクスポートします。
expWks	アクティブシートをラスターまたはベクターの画像ファイルとしてエクスポートします。
img2GIF	アクティブなイメージを gif ファイルにエクスポート

13.1. ワークシートのエクスポート

内容

- 1.3 ワークシートをデータファイルにエクスポート

13.1.1. ワークシートのエクスポート

ワークシートのデータを画像(例、PDF)やデータファイルのどちらかにエクスポートすることができます。

ワークシートを画像ファイルにエクスポート

expWks X ファンクションは、ワークシート全体、ワークシートの表示されている部分、ワークシートの選択部分を、JPEG, EPS, PDF などの画像ファイルにエクスポートするのに使われます。

```
// アクティブワークシートを TEST.EPS という名前の EPS ファイルにエクスポート
// D:\ ドライブに保存
expWks type:=EPS export:=active filename:="TEST" path:="D:";
```

expWks X ファンクションは、**export** オプションを使って、同時に多くのワークシートをエクスポートすることもできます。指定しなければ、アクティブワークシートがエクスポートされます。

次のサンプルでは、*export:=book* が現在のワークブックにあるすべてのワークシートを指定したフォルダ *path* にエクスポートします。

```
expWks type:=PDF export:=book path:="D:\TestImages" filename:=Sheet#;
```

ワークシートはワークブック内で並んだ順に(左から右へ)保存されます。ここで、名前はその順番で、'Sheet1', 'Sheet2'のようにシートを数えるように番号が付けられます。*filename:=Sheet##* のようにすると、'Sheet01'のような名前になります。

export の他のオプションには *project*, *recursive*, *folder*, *specified* があります。

expWks X ファンクションは、作成したグラフが埋め込まれていて、セルを統合したり、色付けして、プレゼンテーション用の分析結果があるカスタムレポートをエクスポートする際にも役立ちます。

ワークシートを複数ページの PDF ファイルにエクスポート

expPDFw X ファンクションは、ワークシートを複数ページの PDF ファイルにエクスポートできます。この X ファンクションは、現在のプリンタの設定では 1 ページに入りきれないほどの内容がワークシートにある大きなワークシートをエクスポートするのに役立ちます。この X ファンクションには、ブック内のすべてのシートまたはプロジェクト内のすべてのシートを出力するオプション、表紙やページ番号を追加するオプションがあります。

ワークシートをデータファイルにエクスポート

このサンプルでは、**expAsc** X ファンクションを使って、ワークシートデータをタブ区切りで ASCII ファイルに出力します。

```
// タブ区切りで Book 2, Sheet 3 のデータを
// TEST.DAT という名前の ASCII ファイルにエクスポートし、D:\ ドライブに保存

expASC iw:=[Book2]Sheet3 type:=0 path:="D:\TEST.DAT" separator:=TAB;
```

このサンプルでは、*type* オプションはファイルの拡張子を示し、次の値のいずれかにセットします(type:=dat は type:=0 と同じ)。

- 0=dat:*.dat,
- 1=text:Text File(*.txt),

- 2=csv:*.csv,
- 3=all:All Files(*.*)

13.2. グラフのエクスポート

LabTalk から呼ばれる **expGraph** X ファンクションを使ってグラフをエクスポートする 3 つのサンプルです。

13.2.1. 幅と解像度(DPI)を指定してグラフをエクスポート

expGraph X ファンクションを使ってグラフを画像としてエクスポートイメージサイズのオプションは **tr1** というツリー変数のノードに保存され、解像度のオプション(すべてのラスター画像に対して)は、**tr2** というツリー変数のノードに保存されます。

一般的なアクリケーションは、画像の幅と解像度の *両方*を指定して、希望の画像形式にグラフをエクスポートします。例えば、グラフを高解像度(1200DPI)の 3.2 インチ幅の*.tif ファイルと指定された 2 段組の刊行物を考えます。

```
// アクティブグラフウィンドウを D:\TestImages\TEST.TIF にエクスポート
// 幅 = 3.2 インチ, 解像度 = 1200 DPI

expGraph type:=tif path:="D:\TestImages" filename:="TEST"
         tr1.unit:=0
         tr1.width:=3.2
         tr2.tif.dotsperinch:=1200;
```

tr1.unit の利用可能な値

- 0 = インチ
- 1 = cm
- 2 = ピクセル
- 3 = ページの比

Note: これは特定の出力タイプを指定するためにツリー構造に保存されているデータにアクセスする良い例です。**tr1** の説明は製品のオンラインヘルプにあります。

13.2.2. プロジェクト内のすべてのグラフをエクスポートします。

プロジェクト内の指定したすべてのオブジェクトをループする **doc -e** コマンドと **expGraph** X ファンクションを組み合わせて使用すれば、Origin プロジェクトからすべてのグラフをエクスポートすることができます。

例えば、現在のプロジェクトのすべてのグラフをビットマップ(BMP)にエクスポートするには

```
doc -e P
{
  // %H はアクティブウィンドウ名を保持する文字列レジスタ
  expGraph type:=bmp path:="d:\TestImages" filename:=%H
           tr1.unit:=2
           tr1.width:=640;
}
```

doc -e のいくつかのサンプルを見ると オブジェクト間をループしているのが分かります。

13.2.3. パスとファイル名を付けてグラフをエクスポートする

文字列レジスタ, %G と %X は、現在のプロジェクトファイル名とパスを保持しています。label コマンドと組み合わせて、グラフのエクスポート時にこれらの情報をページ内に配置することができます。例えば、

```
// プロジェクトのパス
string proPath$ = system.path.program$ + "Samples\Graphing\Multi-Curve
Graphs.opj";
// プロジェクトを開く
doc -o %(proPath$);
// グラフにファイルパスと名前を追加
win -a Graph1;
label -s -px 0 0 -n ForPrintOnly \v(%X%G.opj);
// グラフをドライブ D にエクスポート
expGraph type:=png filename:=%H path:=D:\;
// ファイルパスと名前を削除
label -r ForPrintOnly;
```

13.3. 行列のエクスポート

Origin の行列には、数値データだけでなく、画像データを保存することができます。実際、Origin 内の *すべての* 画像は、行列で保存され、図mたはピクセルデータで表示されます。行列は、保持している内容に関係なくエクスポートできます。

スクリプトで行列をエクスポートすることは、2 つの X ファンクションで行うことができます。画像ではない行列には **expMatAsc** 、画像の行列には **expImage** を使います。

13.3.1. 画像でない行列のエクスポート

画像でない行列を ASCII ファイルにエクスポートするには、**expMatAsc** X ファンクションを使います。エクスポートする際の拡張子は、*.dat (type:=0), *.txt (type:=1), *.csv (type:=2), すべての種類 (type:=3)を利用できます。

```
// XY グリッディングを有効にして TEST.CSV という名前の *.csv 形式の
// ファイルに行列 (Matrix Book 1 の Matrix Sheet 1) をエクスポート
expMatASC im:=[MBook1]MSheet1 type:=2 path:="D:\TEST.CSV" xygrid:=1;
```

13.3.2. 画像の行列のエクスポート

Origin の行列ウィンドウには、複数シートを含めることができ、各シートには複数の行列オブジェクトを含めることができます。行列オブジェクトは RGB 値(デフォルト、1 つの行列セルに 3 つの数字を持つ、各行列セルはピクセルに対応)またはグレースケールデータ(各行列セルには 1 つのグレースケール値)の画像を含めることができます。

例えば、画像を行列オブジェクト(RGB 値として)にインポートして、後で、**イメージ**メニューを使って、グレースケールデータ(例えば、グレースケールのピクセル値)に変換できます 。行列オブジェクトに RGB 値またはグレースケールデータが含まれていれば、**expImage** X ファンクションを使って、行列の内容を画像ファイルにエクスポートできます。例えば、次のスクリプトコマンドは、行列ブック MBook 1 の Sheet1 の最初の行列オブジェクトをエクスポートします。

```
// 画像行列を *.tif 画像としてエクスポート
expImage im:=[MBook1]1!1 type:=tif fname:="c:\flower"
```

ラスター画像形式(JPEG, GIF, PNG, TIF を含む)をエクスポートする時、ビット階調や解像度(DPI)を指定したい場合があります。これは、**expImage** のオプションツリー **tr** で指定することができます。これらのオプションを指定した X ファンクションの呼び出しは次のようになります。

```
expImage im:=[MBook1]MSheet1! type:=png fname:="D:\TEST.PNG"
tr.PNG.bitsperpixel:="24-bit Color"
tr.PNG.dotsperinch:=300;
```

ツリー**tr** のすべてのノードは、オンラインヘルプにあります。

13.4. 動画のエクスポート

グラフのグループを動画としてエクスポートするには、ビデオライター(vw) オブジェクトを使用する必要があります。実際のフレームと共に動画をエクスポートするには、必ずビデオライターオブジェクトを作成する必要があります。いくつかのウィンドウをフレームに見立てて作成し、ビデオライターに渡します。1 度に 1 つのビデオライターでのみ作業できます。つまり、**vw.Create( )** メソッドでビデオライターオブジェクトを作成した後、新たな **vw.Create( )**メソッドを実行する前に、**vw.Release( )** を実行してビデオライターをリリースする必要があります。

ここに、ビデオライターオブジェクトの作成、グラフの作成、ビデオライターオブジェクトのリリース方法を示すスクリプトのサンプルがあります。また、LabTalk サンプルカテゴリー内にあるサンプル全体も合わせてご確認ください。

13.4.1. ビデオライターオブジェクトを作成する

動画をエクスポートする時に最初に行うべきことは、ビデオライターオブジェクトを **vw.Create( )**方法で作成する事です。最低限、ビデオのファイル名（ファイル名を含む）は指定する必要があります。また、ここでは圧縮方法のためのコーデック値、1 秒ごとのフレーム数等を指定できます。

例えば、このスクリプトは動画ファイル"test.avi"を既存のファイルパス、*D:\Exported Videos* に作成し、他の設定はデフォルトを維持します（つまり、圧縮なし、1 フレーム毎秒、640px 横×480px 縦の大きさ）。

```
vw.Create("D:\Exported Videos\test.avi");
//上記スクリプトは次のスクリプトと同義
//vw.Create("D:\Exported Videos\test.avi", 0, 1, 640, 480);
```

FourCC コードを使用すると、圧縮方法を指定できます。例えば、以下のスクリプトは WMV1 圧縮形式を使用して動画を作成します。

```
//4 つの記号コードで codec を定義
int codec = vw.FourCC('W', 'M', 'V', '1');
//800*600 動画ファイルをユーザファイルフォルダ内の test.avi を作成
vw.Create(%Y\My Video.avi, codec, 1, 800, 600)
```

ビデオ作成が成功したかチェックするには、**vw.Create( )**を使用します。**vw.Create( )**メソッドうまく作成できている場合は 0 を返してきます。例:

```
//ファイルのパス、 D:\AAA があるなら、以下のスクリプトは 0 を返すはず
//パスが存在しない場合、エラーコードを返す
int err = vw.Create(D:\AAA\test.avi);
if(err==0)
   type "video creation is successful";
else
   type "video creation failure, the error code is $(err)";
```

13.4.2. ビデオライターオブジェクトでグラフを作成する

ビデオライターオブジェクトが作成されると、**vw.WriteGraph( )** 方法を使用してビデオライターオブジェクトの中にグラフを作成します。ぐらふウィンドウに加え、関数グラフや行列、レイアウトページなどの他のウィンドウを使用することもできます。

例えば、このスクリプトは現在のアクティブなウィンドウをビデオに渡します。

```
vw.WriteGraph( );
```

ウィンドウ名や書き出すフレームの数を指定できます。例えば、以下のスクリプトは Graph1 を 5 フレーム分追加します。

```
vw.WriteGraph(Graph1,5);
```

ループ構造を使用してグラフをビデオに追加する事もできます。これにより、複数のスクリプト行を記入する手間を省く事ができます。以下のサンプルはプロジェクト内の全てのグラフウィンドウを動画にまとめ、各グラフは 2 フレームとして挿入されます。

```
doc -e P
{
   vw.WriteGraph(,2);
}
```

このメソッドで、先程のサンプルと同じように、エラーコードの有無を確認できます。返ってくる値が 0 の場合、グラフ作成（または他のウィンドウ）がうまくいっている事を示します。

13.4.3. ビデオライターオブジェクトをリリースする

各ビデオライターオブジェクトでは、リリースしないと実際のビデオが作成されないので、この操作はとても重要です。このステップで使用するメソッドは、**vw.Release( )**です。

以下のスクリプトは、空のグラフウィンドウと共に、動画ファイル"example.avi"をユーザファイルフォルダに作成するスクリプトのサンプルです。

```
int err = vw.Create(%Y\example.avi);
//ビデオが作成できる場合、既存のグラフをビデオに書き出す
if(0 == err)
{
    //デフォルトのテンプレートで空のグラフウィンドウを作成
    win -t plot;
    vw.WriteGraph( );
}
//ビデオライターのリリース
vw.Release( );
```

vw.Release( ) メソッドも値を返します。0 は動画作成がうまくいったことを示しますが、1 の場合は作成に失敗した事を示します。

14 Origin のプロジェクト

Origin プロジェクトは、すべてのデータ、操作、グラフ、レポートを含みます。ここでは、スクリプトを使ってプロジェクト内の要素を管理するテクニックについて、以下のセクションで説明します。

- プロジェクトを管理する
- メタデータにアクセスする
- オブジェクトのループ
- プロジェクトデータの保護

14.1. プロジェクトを管理する

内容

14.1.1. DOCUMENT コマンド

Document コマンドは、LabTalk コマンドで、Origin プロジェクトに関するさまざまな操作を実行します。**document** コマンドのシンタックスは次のようになっています。

```
document -option value;
```

Note:

- *value* は、いくつかのオプションに対しては利用できず、コマンドから除外されます。
- 詳細は Doc (Object)を確認してください。

内部的に、Origin はプロジェクトを修正した事を示すプロパティを更新します。現在のプロジェクトが修正されているときにプロジェクトを開こうとすると、通常、現在のプロジェクトを保存を促します。document コマンドは、このプロパティを制御するオプションがあります。

新規プロジェクトを開始する

```
// 注意!保存するかどうかを尋ねるプロンプトがオフ
document -s;
// ''doc'' は ''document'' の短縮形、 ''n'' は ''new''の短縮形
doc -n;
```

プロジェクトを開く/保存する

doc -o コマンドを使って、プロジェクトを開く/保存、save コマンドでプロジェクトを保存します。

```
// Origin プロジェクトファイルを開く
string fname$ = SYSTEM.PATH.PROGRAM$ + "Origin.opj";
doc -o %(fname$);    // ''document -open''の短縮形
// 変更を行う
%(Data1,1) = data(0,100);
%(Data1,2) = 100 * uniform(101);
// 新しい場所に新しい名前でプロジェクトを保存
fname$ = SYSTEM.PATH.APPDATA$ + "My Project.opj";
save %(fname$);
```

プロジェクトを追加する

前のスクリプトに続けて、別のプロジェクトを追加することができます。Origin は、同時に 1 つのプロジェクトファイルのみをサポートするため、既存のプロジェクトと追加されたプロジェクトがひとつのプロジェクトになります。

```
// Origin プロジェクトファイルを現在のファイルに追加
fname$ = SYSTEM.PATH.PROGRAM$ + "Origin.opj";
doc -a %(fname$); // Abbreviation of ''document -append''
// 現在のプロジェクトを保存 - これは ''My Project.opj''
save;
// 新しい名前で現在のプロジェクトを新しい場所に保存
save C:\Data Files\working.opj;
```

子ウィンドウを保存する/ロードする

Origin で、グラフ、ワークブック、行列、Excel ブックなどの子ウィンドウを 1 つのファイルに保存することができます。Append は、別のプロジェクトにファイルを追加するのに使うことができます。ワークブック、行列、グラフに対しては、適切な拡張子が自動的に追加されますが、Excel ウィンドウには.XLS を指定する必要があります。

```
// save コマンドはアクティブウィンドウに対して行われる
save -i C:\Data\MyBook;
```

Append は、次のウィンドウの種類をロードするのに使うことができます。

```
// ワークブック(*.OGW), 行列(*.OGM), グラフ(*.OGG), Excel(*.XLS)
dlgfile group:=*.ogg;
// fname は dlgfile Xファンクションでセットされる文字列変数
doc -a %(fname$);
```

Excel に対して、Excel ファイルを Excel として開くのではなく、インポートするように指定することができます。

```
doc -ai "C:\Data\Excel\Current Data.xls";
```

ノートウィンドウは、特殊なオプションスイッチを使った特別なケースです。

```
// Notes1 という名前のノートウィンドウを保存
save -n Notes1 C:\Data\Notes\Today.TXT;
// MyNotes という名前のノートウィンドウにテキストファイルを読み込む
open -n C:\Data\Notes\Today.txt MyNotes;
```

外部 Excel ブックを保存する

これは Origin 8.1 から導入された機能で、外部的にリンクした Excel ブックを現在のファイル名で保存します。

```
save -i;
```

ウィンドウのリフレッシュ

次のコマンドを使って、ウィンドウをリフレッシュすることができます。

```
doc -u;
```

14.1.2. プロジェクトエクスプローラの X ファンクション

次の X ファンクションは、DOS コマンドのように利用して、プロジェクトのサブフォルダを作成したり、削除したり、移動することができます。

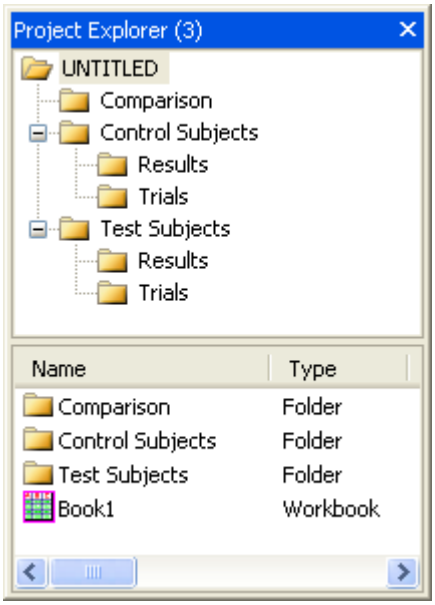
名前	簡単な説明
pe_dir	アクティブフォルダの内容を表示します。
pe_cd	別のフォルダに変更します。
pe_move	フォルダまたはウィンドウを移動します。
pe_path	現在のパスをレポートします。
pe_rename	フォルダまたはウィンドウの名前を変更します。
pe_mkdir	フォルダを作成します。
pe_rmdir	フォルダを削除します。

この例では

```
doc -s;                          // Origin の保存フラグを消去
doc -n;                          // 新しいプロジェクトを開始
pe_cd /;                         // トップレベルに移動
pe_mkdir "Test Subjects";        // フォルダを作成
pe_cd "Test Subjects";           // フォルダに移動
```

```
pe_mkdir "Trials";              // サブフォルダを作成
pe_mkdir "Results";             //  もう一つ作成
pe_cd /;                        // トップレベルに移動
pe_mkdir "Control Subjects"; // 別のフォルダを作成
pe_cd "Control Subjects";       // そのフォルダに移動
pe_mkdir "Trials";              // サブフォルダを作成
pe_mkdir "Results";             //  別のフォルダを作成
pe_cd /;                        // トップレベルに移動
pe_mkdir "Comparison";          // フォルダを作成
```

プロジェクトエクスプローラ内で下図のようにフォルダ構造を作成しています。:



ツール :オプション :オープン/クローズで、**サブフォルダ中にオープンする**が有効の場合、オプション:**Folder1** という名前のフォルダを追加します。

14.2. メタデータにアクセスする

メタデータは、他のデータを参照する情報です。サンプルには、データが元々収集された時刻、データを収集した機器を操作した人、調べた標本の温度が含まれます。メタデータはプロジェクト、ページ、レイヤ、列に保存することができます。

内容

- 3.2 レポートページツリーにアクセスする
- 3.3 ページストレージのユーザツリー
- 3.4 ワークシートのユーザツリー
- 3.5 ワークシート列のユーザツリー

14.2.1. 列ラベル行

メタデータは、ロングネーム (L), 単位 (U), コメント(C), サンプリング間隔 ユーザ定義パラメータを含むさまざまな パラメータ行などといった情報を 列ヘッダに含むワークシート内で表示できます。

列ラベル行は文字によるインデックスが割り当てられます。この文字は列ラベル行のリファレンステーブルで確認できます。使用例

列ラベル行の読み込み/書き出し

スクリプトから または 列ヘッダ の文字列を捕まえたり、セットしたい場合があります。対応するラベル行の文字を行インデックスとして、ラベル行にアクセスします。

Note:数値セルへのアクセスはラベル行の文字を使うことをサポートしていません。

以下は列ヘッダ文字列を読み書きするサンプルです。

```
Book1_A[L]$ = Time;    // 列 A のロングネームを Time にセット
Book1_A[U]$ = sec;     // 列 A の単位を sec にセット
string strC$ = col(2)[C]$;    // 2 番目の列のコメント行を読み strC$に
// 最初のシステムパラメータ行から値を取得
double syspar1 = %(col(2)[p1]$);
Col(1)[L]$="Temperature";    // Col(1) のロングネームを"Temperature"に
range bb = 2;                // Col(2)の範囲変数を宣言
// Col(2) のロングネームに文字列 " Data" を追加して
//  Col(1) のロングネームにセット
bb[L]$=Col(1)[L]$+" Data";
```

Note:Origin 8.0 では、LabTalk 変数が列ラベル行の文字に優先されます。例えば、

```
int L = 4;     // Origin 8.0 以前では
Col(B)[L]$=    // Col(B)の 4 行目の値を文字列として返す
```

しかし、Origin 8.1 では、これは変更され、列ラベル行(L, U, C, など)が優先されます。

```
int L = 4;     // Origin 8.1 では
Col(B)[L]$=    // Col(B)のロングネームを文字列として返す
```

次のサンプルはユーザパラメータ行の作成およびアクセス方法を示しています。

```
// 最初のユーザパラメータ行を表示
wks.userParam1 = 1;
// 最初のユーザパラメータ行をカスタムネームに割り当て
wks.userParam1$ = "Temperature";
// 列の特定のユーザパラメータ行に書き込み
col(2)[Temperature]$ = "96.8";
```

```
// ユーザ定義パラメータ行の値を取得
double temp = %(col(2)[Temperature]$);
```

wks.labels オブジェクトメソッドで、どの列ヘッダ行をどの順番で表示するかをセットすることができます。アクティブワークシートに対して、このスクリプトは列ヘッダ行、ロングネーム、単位、最初のシステムパラメータ、最初のユーザパラメータ、コメントの順で表示します。ロングネーム, 単位, 最初のシステムパラメータ, 最初のユーザパラメータ, コメント

```
range ww = !;
ww.labels(LUP1D1C);
```

14.2.2. 等間隔なサンプリング間隔

Origin のユーザはデータ系列(Y)に対するサンプリング間隔をデータポイントの対応する行以外の値にセットすることができます。

サンプリング間隔列ラベル行にアクセスする

これを行うと、特別なヘッダ行が作成され、カスタム間隔(と初期の値)が適用されていることが分かります。このヘッダ行のテキストにアクセスするには、単に **E** 行インデックス文字を使います。このヘッダ行は、実際には読み込み専用で、自由な文字列をセットできません。しかし、構成されているこの文字列のプロパティを列のプロパティ (wks.col オブジェクトを参照) または colint X ファンクションのどちらかで変更できます。

```
// 1 番目の列のサンプリング間隔ヘッダテキストを読み取り、文字列変数に
string sampInt$ = Col(1)[E]$;
// サンプリング間隔が Col(1)に開始値 2、増分 0.5 で
// セットされている場合、
sampInt$=;
// "x0 = 2"
// "dx = 0.5"// と出力される
```

サンプリング間隔のヘッダを見るには、以下のステップを試します。

1. 新しいワークシートを作成し、X 列を削除します。
2. 残りの列(例：B(Y))の一番上をクリックして列全体を選択し、右クリックしてメニューから**サンプリング間隔に設定**を選択します。
3. 初期値および増分値を 1 以外の値にします。
4. OK をクリックし、指定した値が入力されている新しいヘッダ行が表示されます。

次のサンプルは X ファンクションを使って、スクリプトからこれを行う方法を示しています。

また、あるデータタイプ(例えば*.*wav*)をインポートするとき、サンプリング間隔がヘッダ行として表示されます。

X ファンクションによるサンプリング間隔

サンプリング間隔は、その表示がユーザ情報で書式化されるという点で特別なものです。プログラム的に、以下のようにアクセスできます。

```
// X ファンクションの完全表記を使用
colint rng:=col(1) x0:=68 inc:=.25 units:=Degrees lname:="Temperature";
// 以下は短い表記
colint 1 68 .25 Degrees "Temperature";
```

初期値と増分値はワークシート列プロパティを使って読み込むことができます。

```
double XInitial = wks.col1.xinit;
double XIncrement = wks.col1.xinc;
string XUnits$ = wks.col1.xunits$;
string XName$ = wks.col1.xname$;
```

Note:サンプリング間隔が設定されている場合を除き、これらのプロパティが列プロパティのリスト内で表示されます(スクリプトウィンドウに **wks.col1.=** と入力し、列 1 のプロパティ名を表示)。

- 文字列 **wks.col1.xunits$** および **wks.col1.xname$** には値がありません。
- 数値 **wks.col1.xinit** および **wks.col1.xinc** は、それぞれ行番号の最初の値と増分値に対応する 1 という値を持ちます。

14.2.3. ツリー

ツリーは LabTalk でサポートされるデータタイプで、これは既存のデータを構造化するものなのでメタデータの形式のツリーも考えます。これらは、データタイプと変数で扱うセクションで簡単に紹介されていますが、X ファンクションに対しても重要なので再度説明しています。

多くの X ファンクションは、データをツリー形式で入力、出力します。X ファンクションは LabTalk スクリプトからアクセスできる主要なツールの 1 つなので、効率よくツリー変数を認識し、使用することが重要です。

ファイルインポートのツリーノードにアクセスする

ワークシートにデータをインポートしたあと、Origin はページレベルで特別なツリーのような構造のメタデータを保存します。ファイルについての基本情報は、この構造から直接取り出すことができます。

```
string strName, strPath;
double dDate;
// 構造からファイル名、パス、日付を取得
strName$ = page.info.system.import.filename$;
strPath$ = page.info.system.import.filepath$;
dDate = page.info.system.import.filedate;
//  % と $ の両方の置換法が使われる
ty File %(strPath$)%(strName$), dated $(dDate,D10);
```

このツリー構造は、インポートについて追加の情報を持つツリーを含みます。このツリーは X ファンクションを使ってツリー変数として抽出できます。

```
Tree MyFiles;
impinfo ipg:=[Book2] tr:=MyFiles;
MyFiles.=; // ツリーの内容をスクリプトウィンドウに出力
```

注意:*impinfo* ツリーの内容は、インポートに使われる関数に依存します。

複数ファイルを 1 つのワークブックにインポートする場合(新しいシート、新しい列または新しい行)、オーガナイザーは最後にインポートしたシステムメタデータのみを表示するため、各ファイルに対する特定のツリーをロードする必要があります。

```
Tree trFile;
int iNumFiles;
// ファイルの数を探すために最初にこの関数を使用
impinfo trInfo:=trFile fcount:=iNumFiles;
// すべてのファイルをループ - インデックス 0 から始まる
for( idx = 0 ; idx < iNumFiles ; idx++ )
{
   // 次のファイルのツリーを取得
   impinfo findex:=idx trInfo:=trFile;
   string strFileName, strLocation;
   //
   strFileName$ = trFile.Info.FileName$;
   strLocation$ = trFile.Info.DataRange$;
   ty File %(strFileName$) was imported into %(strLocation$);
}
```

レポートページツリーにアクセス

分析レポートページは、ツリー構造に基づく特別な形式のワークシートです。この構造を getresults X ファンクションを使って、ツリー変数を取得し、結果を抽出できます。

```
// Origin サンプルファイルをインポート
string fpath$ = "Samples\Curve Fitting\Gaussian.dat";
string fname$ = SYSTEM.PATH.PROGRAM$ + fpath$;
impasc;
// データの Gauss フィットを実行しレポートシートを作成
nlbegin (1,2) gauss;
nlfit;
nlend 1 1;
// 自動作成の文字列変数 __REPORT$,
// は最後に作成されたレポートシートの名前を保持
string strLastReport$ = __REPORT$;
// これはレポートをツリーに取得する X ファンクション
getresults tr:=MyResults iw:=%(strLastReport$);
// これらの結果にアクセスできる
ty Variable\tValue\tError;
separator 3;
ty y0\t$(MyResults.Parameters.y0.Value)\t$(MyResults.Parameters.y0.Error);
ty xc\t$(MyResults.Parameters.xc.Value)\t$(MyResults.Parameters.xc.Error);
ty w\t$(MyResults.Parameters.w.Value)\t$(MyResults.Parameters.w.Error);
ty A\t$(MyResults.Parameters.A.Value)\t$(MyResults.Parameters.A.Error);
```

ページストレージのユーザツリー

ツリー構造を使って情報をワークブック、行列ブック、グラフページに保存できます。次のサンプルは、セクションを作成し、アクティブページ保存領域にサブセクションと値を追加する方法を示します。

```
// Experiment という新しいセクションを追加
page.tree.add(Experiment);
// Sample というサブセクションを追加
page.tree.experiment.addsection(Sample);
// サブセクションに値を追加
page.tree.experiment.sample.RunNumber = 45;
page.tree.experiment.sample.Temperature = 273.8;
```

```
// Detector という別のサブセクションを追加
page.tree.experiment.addsection(Detector);
// 値を追加
page.tree.experiment.detector.Type$ = "InGaAs";
page.tree.experiment.detector.Cooling$ = "Liquid Nitrogen";
```

情報を保存すると、保存内容をダンプすることで取り出すことができます。

```
// ページストレージのすべての内容を出力
page.tree.=;
// またはプログラムでアクセス
temperature = page.tree.experiment.sample.temperature;
string type$ = page.tree.experiment.detector.Type$;
ty Using %(type$) at $(temperature)K;
```

ワークブックまたは行列ブックのページオーガナイザでツリーを表示できます。

ワークシートのユーザツリー

ワークブックのページレベルに保存されているツリーは、シートがアクティブであるかどうかに関係なく、アクセスできます。シートレベルでツリーを保存することもできます。

```
// ここではアクティブシートに 2 つのツリーを追加
wks.tree.add(Input);
// ブランチと値を動的に作成
wks.tree.input.Min = 0;
// 別の値を追加
wks.tree.input.max = 1;
// 2 番目のツリーを追加
wks.tree.add(Output);
// 2 つ以上の値を追加
wks.tree.output.min = -100;
wks.tree.output.max = 100;

// ツリーを出力
wks.tree.=;
// またはアクセス
ty Input $(wks.tree.input.min) to $(wks.tree.input.max);
ty Output $(wks.tree.output.min) to $(wks.tree.output.max);

// range を使ってシートレベルのツリーにアクセス
range rs = [Book7]Sheet2!;
rs!wks.tree.=;
```

ワークブックまたは行列ブックのページオーガナイザでツリーを表示できます。

ワークシート列のユーザツリー

個々のワークシート列はツリー形式でメタデータを含むことができます。ツリーノードを割り当てたり、読みだすことはページレベルのツリーと非常に似ています。

```
// COLUMN ツリーを作成
wks.col2.tree.add(Batch);
// ブランチを追加
wks.col2.tree.batch.addsection(Mix);
// ブランチに 2 つの値を追加
wks.col2.tree.batch.mix.ratio$ = "20:15:2";
wks.col2.tree.batch.mix.BatchNo= 113210;
// 動的にブランチと値を追加
wks.col2.tree.batch.Line.No = 7;
wks.col2.tree.batch.Line.Date$ = 3/15/2010;

// スクリプトウィンドウにツリーを出力
wks.col2.tree.=;
// またはツリーにアクセス
batch = wks.col2.tree.batch.mix.batchno;
string strDate$ = wks.col2.tree.batch.Line.Date$;
ty Batch $(batch) made on %(strDate$) [$(date(%(strDate$)))];
```

列のプロパティダイアログのユーザツリータブでこれらのツリーを表示できます。

14.3. オブジェクトのループ

Origin プロジェクトに存在している特定のオブジェクトに、ある操作を次々と実行することができます。例えば、プロジェクトのすべてのグラフレイヤを再スケールしたり、プロジェクト内のすべてのワークシートに新しい列を追加することができます。LabTalk の document コマンド(または **doc**)は、このような操作を行うものです。**doc** コマンドのいくつかの例がここに示されています。

内容

14.3.1. プロジェクト内のオブジェクトをループする

-e または -ef スイッチ付きの document コマンド(または doc -e コマンド)は、Origin プロジェクト内のさまざまなオブジェクトのコレクションをループするための一般的な方法です。このコマンドは、コレクション内に見つかった Origin の各インスタンスに複数行の LabTalk スクリプトを実行します。

ワークブックとワークシートをループする

doc -e LB コマンドを使って、プロジェクト内のすべてのワークシートをループできます。以下のスクリプトは、すべてのワークシートをループし、行列レイヤを読み飛ばします。

```
//プロジェクト内のすべてのワークシートをループし
//その名前および各シートの列数を出力
doc -e LB {
        if(exist(%H,2)==0) //ワークブックではなく、行列でなければならい
                continue;
        int nn = wks.nCols;
        string str=wks.Name$;
        type "[%H]%(str$) has $(nn) columns";
}
```

次のサンプルは、プロジェクトの異なるワークブックにあるデータ列をループして、操作する方法を示します。

Origin 8.1 SR2 以降のバージョンで利用可能なサンプルプロジェクトファイルを次のパスから開きます。
\\Samples\LabTalk Script Examples\Loop_wks.opj

プロジェクトには、Sample1、Sample2 フォルダとバックグラウンド信号に **Bgsignal** というフォルダがあります。Sample1、Sample2 フォルダには **Freq1** および **Freq2** という 2 つのフォルダがあり、これは特定のサンプルに対する 1 セットの周波数でのデータに対応しています。

Freq1、**Freq2** フォルダ内のワークブックは **DataX**, **DataY** および定数の周波数を含む 3 列があります。**Bgsignal** フォルダのワークブックの名前は **Bgsig** です。**Bgsig** ワークブックでは、**DataX** と 2 つの Y 列を含む 3 列があり、そのロングネームは、各 **Freq** フォルダのワークブックの周波数に対応しています。

目的は、各ワークブックの列を加算し、特定の周波数に対するバックグラウンド信号を同じ周波数のサンプルデータから減算することです。次の Labtalk スクリプトはこの操作を実行します。

```
doc -e LB
{ //各ワークシートをループ
   if(%H != "Bgsig") //バックグラウンド信号のワークブックをスキップ
   {
      Freq=col(3)[1]; //周波数を取得
      wks.ncols=wks.ncols+1;  //サンプルシートに列を追加
      //ロングネームを使って Freq に対する bg 信号列を取得
      range aa=[Bgsig]1!col("$(Freq)");
      wcol(wks.ncols)=col(2)-aa; //bg 信号を減算
      wcol(wks.ncols)[L]$="Remove bg signal"; //ロングネームをセット
   }
}
```

上記コードよりもやや遅いですが、コード内でブックをループし、そしてシートをループすることもできます。

以下の例では、現在アクティブなプロジェクトエクスプローラのフォルダにあるすべてのワークブックをループし、見つかった各ブック内のシートをループする方法を示しています。

```
int nbooks = 0;
// このフォルダの名前を取得
string strPath;
pe_path path:=strPath;
// すべてのワークブックをループ
// 現在のフォルダの中のみ
doc -ef W {
```

```
   int nsheets = 0;
   // 各ワークブック内のすべてのワークシートをループ
   doc -e LW {
      type Sheet name:%(layer.name$);
      nsheets++;
   }
   type Found $(nsheets) sheet(s) in %H;
   type %(CRLF);
   nbooks++;
}
type Found $(nbooks) book(s) in folder %(strPath$) of project %G;
```

さらにワークブックプロパティを使って内部ループを置き換えることができます。

```
int nbooks = 0;
// このフォルダの名前を取得
string strPath;
pe_path path:=strPath;
// すべてのワークブックをループ
// 現在のフォルダの中のみ
doc -ef W {
   // 各ワークブックのすべてのワークシートをループ
   loop(ii,1,page.nlayers) {
      range rW = [Book1]$(ii)!;
      type Sheet name:%(rw.name$);
   }
   type Found $(page.nlayers) sheet(s) in %H;
   type %(CRLF);
   nbooks++;
}
// 最終的なレポート- %G はプロジェクト名を含む
type Found $(nbooks) book(s) in folder %(strPath$) of project %G;
```

グラフウィンドウをループ

ここでは、すべての グラフウィンドウをループします(プロットウィンドウには、すべてのグラフ, 関数グラフ, レイアウトページ、埋込グラフが含まれます)。

```
doc -e LP
{
    // 埋め込みグラフまたはレイアウトウィンドウをスキップ
    if(page.IsEmbedded==0&&exist(%H)!=11)
    {
        string name$ = %(page.label$);
        if(name.Getlength()==0 ) name$ = %H;
        type [%(name$)]%(layer.name$);
    }
}
```

下記のスクリプトは、プロジェクト内の全てのグラフウィンドウ中の内容をデフォルトプリンタに印刷します。

```
doc -e P print; // ''document -each Plot Print''の省略形
```

ワークブックウィンドウをループ

document -e command コマンドをネストすることができ、この例では、すべてのワークシート内のすべての Y データセットをループします。

```
doc -e W
{
    int iCount = 0;
    doc -e DY
    {
        iCount++;
    }
    if( iCount < 2 )
      { type Worksheet %H has $(wks.ncols) columns,;
        type $(iCount) of which are Y columns; }
    else
      { type Worksheet %H has $(wks.ncols) columns,;
        type $(iCount) of which are Y columns; }
}
```

列と行をループする

この例は、すべての列をループし、n 番目の列ごとに削除します。

```
int ndel = 3; // 必要に応じてこの数字を変更
int ncols = wks.ncols;
int nlast = ncols - mod(ncols, ndel);
// 右から左に削除する必要がある
for(int ii = nlast; ii > 0; ii -= ndel)
{
   delete wcol($(ii));
}
```

この例では、ワークシートの行ごとに削除する方法を示しています。

```
int ndel = 3;            // 必要に応じてこの数字を変更
range rr = col(1);       // 列 1 の範囲を取得
nrows = rr.GetSize();    // 数値行の数を取得
int nlast = nrows - mod(nrows, ndel);
// 下から上に削除する必要がある
for(int ii = nlast; ii > 0; ii -= ndel)
{
   range rr = wcol(1)[$(ii):$(ii)];
   mark -d rr;
}
```

このスクリプトは、Sheet1 の 4 つの列の対数を計算し、Sheet2 の対応する列に結果を配置する

```
for(ii=1; ii<=4; ii++)
{
        range ss = [book1]sheet1!col($(ii));
        range dd = [book1]sheet2!col($(ii));
        dd = log(ss);
}
```

図形オブジェクトをループする

アクティブレイヤ内のすべての 図形オブジェクト をループすることができます。2 つのオプションでこれを包むことによって、すべてのグラフウィンドウをカバーできます。

```
// 各プロットに対して
doc -e P
{
    // 各プロットの各レイヤに対して
    doc -e LW
    {
        // 各プロットの各レイヤの各図形オブジェクトに対して
        doc -e G
        {   // 凡例の背景に陰を付ける設定
            if("%B"=="Legend") %B.background = 2;
            // 日時スタンプの色を青にセット
            if("%B"=="timestamp") %B.color = color(blue);
        }
    }
}
```

14.3.2. グラフのすべてのレイヤにピーク分析を実行する

この例は、グラフ内のすべてのレイヤをループし、事前に保存したピークアナライザテーマファイルを使って、各レイヤのデータセットにピーク分析を実行します。アクティブウィンドウが複数レイヤグラフで、各レイヤに 1 つのデータ曲線があるものとします。また、ピークアナライザテーマファイルが存在しているものとします。

```
// ループを入力する前にリマインダメッセージを表示しないようにする
// これは Origin がレポートシートに切り替わることについての
// リマインダーメッセージをポップアップしないようにする
type -mb 0;
// グラフウィンドウのすべてのレイヤをループ
doc -e LW
{
   // 事前セットしたテーマでピーク分析を実行
   sec;
   pa theme:="My Peak Fit";
   watch;
   /* sec と watch は任意
      これらは各レイヤでデータフィットにかかる時間を出力 */
}
// リマインダーメッセージを戻す
type -me;
```

14.4. プロジェクトデータの保護

Origin 9.1 で紹介されているように、LabTalk コマンドはワークシートやワークブックの様々な保護が行えます。これには、オブジェクトレベルの保護を修正できるプロジェクトレベルの **Admin** モードを含みます。Admin モードなしでも Origin オブジェクトの状態を保護制御できますが、これらは安全ではありません。誰でも LabTalk コマンドを実行して保護を解くことができます。

安全な保護のためには、**Admin** パスワードを Origin プロジェクトに設定し、正しいパスワードが入力されない限り Origin オブジェクトを変更できないようにします。安全性が問題ではない場合(例えば、意図しない変更やブックおよびシートの削除から保

護する場合)、Admin パスワードなしでオブジェクトレベルの保護を追加できます。Admin パスワードの設定についての情報は、Admin モードを確認してください。

内容

14.4.1. プロジェクトレベルの保護

2 種類のプロジェクトレベルのパスワード保護があります。

- Origin プロジェクトにパスワードを追加し、権限のない人間が OPJ を開くのを防ぎます。この方法は、GUI からと LabTalk スクリプトからの方法があります。
- プロジェクトに Admin パスワードを追加して、権限のない人間の OPJ の編集を防ぎます。この方法は OPJ のアクセスは制限しませんが、OPJ の編集や新しいファイルへの保存を防ぎます。この方法は LabTalk からのみ利用できます。

OPJ 開封を防ぐパスワード保護

この種類のパスワード保護が有効な場合、Origin プロジェクト(OPJ)を開くときにパスワードの入力が必要です。これは、追加による開封や、ス繰王とによる開封などでも適用されます。権限のない開封からプロジェクトを保護するには、

- プロジェクトファイルを開き、スクリプトウィンドウかコマンドウィンドウで、次のように入力します。

```
doc -pwd
```

これによりパスワードダイアログが開き、プロジェクトパスワードの設定と変更が可能です。

- GUI でのアクセスは、**ツール:データ保護:パスワード保護**を選択し、パスワードを入力します。

OPJ 編集を防ぐパスワード保護:Admin モード

Admin モードの概念は、Origin プロジェクトに追加されます。プロジェクトが Admin モードで保護されると、Admin として OPJ にログインしない限り、プロジェクトの編集やワークブック、ワークシートの保護機能を変更することはできません。

OPJ の Admin 保護を行うには、スクリプトウィンドウで以下のように入力します。

```
doc -pwa [password] // Admin パスワード ''password'' を OPJ に追加
```

角括弧で示したように、パスワードの選択はオプションです。もし、パスワードを選択しない場合、Admin モードでのログイン時に要求されません。

パスワードを作成した場合、またはデフォルト("origin")を使用した場合、次のようなコマンドを入力して Admin モードにログインします。

```
doc -pw [password] // Admin モードにログインパスワードが指定された場合、パスワードを与える
```

追加のコマンド

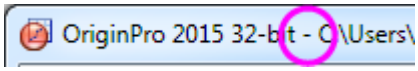
```
doc -pwx // Admin モードのログアウト保護へのアクセスなし

doc -pwr // パスワード削除。実行のためにはログインが必要

doc -pwta // 保護が有効なブックとシートを表示
```

> **document** コマンドについての詳細は、document コマンドを確認してください。

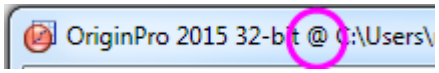
Admin モードを示すタイトルバー

Origin のタイトルバーは、Origin プロジェクトの Admin 状態を表示します。

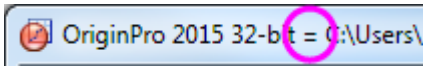
- OPJ の Admin モードなし(通常の状態)



- OPJ の Admin モードが有効で Admin としてのログインなし



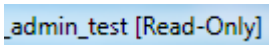
- OPJ の Admin モードが有効で Admin としてログイン



Admin モードと OPJ の保存

プロジェクトの Admin モードを有効にすると、ログインなしでの Origin 上での上書きは行えません。Admin が有効な OPJ を開き、ログインせずに保護されていない領域(ワークブックおよびワークシートレベルの保護を参照)で OPJ を修正すると、Origin のタイトルバーのファイル名にアスタリスク(*) が付きます。

- 未修正ファイル

- 修正されたファイル

アスタリスクは、OPJ が修正されたことを示していますが、保存前にログインしない限り OPJ を保存できません。保存やログインせずに OPJ を閉じることはできます。

14.4.2. ワークシートおよびワークブックレベルの保護

以下の保護方法は Admin モードにすることなく適用できます。しかし、Admin モードがないので、これらの保護削除からのファイルアクセスを防ぐことはできません。

次のコマンド構文を使用してシートレベルで保護を適用できます。

```
layer -lw hex(hex value)
```

次のコマンド構文を使用してブックレベルで保護を適用できます。

```
page -lw hex(hex value)
```

保護フラグには、一般の継承の規則があります。ブックに保護を設定すると、ブック内の全てのシートはブックからフラグが継承されます。この決まりの例外は **Delete** フラグです。ブックとシートの削除フラグは分けて制御する必要があります（例えば、ブックの編集を許可しなくても、個別シートの削除は許可されます）。

レイヤおよびページの保護フラグ

レイヤやページの保護には、以下の *hex value* をとります。

Hex Value	説明
--	全ての保護ビットをオン
0	アクティブワークシート/ワークブックから全ての保護を削除
2	**データ**：ワークシート内の全てのセル、データセルやラベルセルを含む
80	**構造**：シート内の列を変更せずに保持。列の挿入/削除、移動を防ぐ。しかし、行は変更可能。
100	**名前の変更**：シート名の変更を防ぐ(ワークブックに)
400	**削除**：オブジェクトの削除を防ぐ
Note:保護を適用する前に、排除ゾーンの注意を確認してください。	

レイヤおよびページの保護サンプル

次のサンプルは複数の保護フラグを統合(hex value を追加)し、保護するものを高度に制御することを示したサンプルです。

サンプル：ワークシート書き込みアクセスフラグの設定

```
lay -lw hex(82); //データと構造保護のためにアクティブシートをセット
lay -lw; //全て保護
lay -lw 2; //データのみ保護
lay -lw hex(180); //列/行の挿入/削除なし、シート名変更なし
lay -lw 0; //現在のシートの全ての保護を削除
```

... ここで *hex value* は上の表で与えられた値です。

サンプル:ワークブック書き込みアクセスフラグの設定

```
page -lw; //ブックと全てのシートをロック。変更不可。ブックやシートの削除不可。
page -lw hex(482); //削除、構造変更、データ編集をロック

// ロックするアクティブシートにブックを用意
// 分析結果のブックへの追加を許可
lay -lw hex(582); //削除、名前変更、列/行の追加/挿入/移動、編集なし

page -lw hex(400); //ブック削除なし
```

... ここで *hex value* は上の表で与えられた値です。

> **Note**:Origin 2015 SR1 では、次のコマンド
>
> ```
> page -lw; // ブックと全シートのロック
> ```
>
> はワークブックの名前変更を防ぐことができません。

現在の保護フラグを印字

```
layer -lt // 現在のシート保護フラグを印字

layer -lt // 現在のシート保護フラグを印字
```

> - **layer** コマンドについての詳細は、layer コマンドを確認してください。
> - **page** コマンドについての詳細は、page コマンドを確認してください。

読み取り専用シートの排除ゾーン

ワークシートの修正をロックしたとき、シート内の全てのセルは読み取り専用になります。選択的なシートの修正をユーザができるようにしたいときがあります(例えばフォームとしてシートを設定)。シートの**排除ゾーン**の作成と管理の方法を説明します。

排除ゾーンを制御するコマンド

```
layer -les n
```

...ここで、*n* は次のいずれかです。

値	説明
0	アクティブシートから排除ゾーンをクリア
1	現在の選択、1 つ範囲または CTRL+選択による複数範囲を追加し、新しい排除ゾーンとする
2	スクリプトウィンドウで排除ゾーンをリスト表示

Note:排除ゾーンの修正は、シートが保護されていないときだけ可能です。シートをフォームとして設定するには、(1) 必要な排除ゾーンを追加し、(2) **lay -lw** でシートを修正から保護します。

グラフの保護

ここで概説されたスクリプトコマンドはワークブックやワークシートに実行されますが、**Delete** フラグは一般にグラフ保護に使用でき、次のようにします。

```
repeat 10 {win -t plot}; //10 このグラフウィンドウを作成
win -o graph4 {page -lw hex(400)}; //graph4 に保護を設定
doc -e P {win -c}; //OPJ 内の全てのグラフの削除をループ Graph4 は保護されます。
```

スクリプトサンプル

以下のスクリプトは、ワークブックに対してデータシートのロックを設定し、ワークブックを削除から保護します。OPJ は保存でき、共有できます。他のユーザは表示、グラフ作図、データ分析は行えますが、編集と削除はできません。

分析結果は追加のシートに出力されるので、このサンプルのデータシートは編集から保護されているので、**Smooth** 等で生成された結果のルーチンは他のシートに出力するように構成されます。

```
// サンプルファイルをインポート
newbook;
string fname$=system.path.program$+"Samples\Curve Fitting\Gaussian.dat";
impasc;
// パスワードを設定しログイン - パスワードン文字列として mypwd を設定
doc -pwa mypwd;
doc -pw mypwd;
// データシート構造保護、編集と削除の禁止
lay -lw hex(482);
// ページ削除禁止
page -lw hex(400);
// ログアウト
doc -pwx;
// これで OPJ を保存でき、他のユーザはデータの表示、グラフ作成、分析操作が可能
// 分析結果は同じシートの新しい列でなく新しいシートに追加される
```

15 分析とアプリケーション

Origin は、ある分野向けのデータ分析および特定の数学・科学アプリケーションに価値のある関数をサポートしています。次のセクションでは、これらの関数について、利用分野に応じたサンプルを提供しています。

このセクションで説明している項目

- 数学
- 統計
- カーブフィット
- 信号処理
- ピークと基線
- 画像処理

15.1. 数学

このセクションには、データ処理に使える 4 つの一般的な数学演算を行うサンプルがあります。

このセクションで説明している項目

- 複数曲線の平均
- 微分
- 積分
- 補間

15.1.1. 複数曲線の平均

avecurves X ファンクションを使って、複数曲線(XY データ)を平均し、1 つの曲線を作成できます。この X ファンクションには、平均する時に曲線の X 値うい計算する方法や等間隔の X 値を指定する方法などのオプションがあります。

以下は線形補間で平均するサンプルです。

```
// 既存のスクリプト 'loadDSC.ogs' を使ってサンプルデータをロード
string fpath$ = "Samples\LabTalk Script Examples\LoadDSC.ogs";
string LoadPath$=system.path.program$ + fpath$;
// データが適切にロードされない場合、スクリプトの実行を止める
if(!run.section(%(LoadPath$), Main, 0)) break 1;
// データがアクティブワークブックにロードされる
// アクティブワークブックの名前を取得、%H はアクティブワークブックを指す
```

```
string dscBook$=%H;

// 線形補間を使ってすべてのデータの平均を実行
avecurves iy:=[dscBook$](1:end)!(1,2)
          rd:=[<input>]<new name:="Averaged Data">!
          method:=ave
          interp:=linear;
```

平均したら、データと結果をプロットできます。

```
//  plotxy X ファンクションを使って、すべてのデータと平均曲線をプロット
plotxy [dscBook$](1:end)!(1,2) plot:=200;
```

15.1.2. 微分

微分を見つける

次のサンプルは、データセットの微分を計算する方法を示しています。**differentiate** X ファンクションが使われ、高次微分を行うこともできます。

```
// データをインポート
newbook;
fname$ = system.path.program$ + "\Samples\Spectroscopy\HiddenPeaks.dat";
impasc;

// 列 2 のデータの一階微分および二階微分を計算:

// 出力はデフォルトで次に利用可能な列、列 3
differentiate iy:=Col(2);
// 出力はデフォルトで列 4 に
differentiate iy:=Col(2) order:=2;

// 元データと結果をプロット

// 各プロットは x 値として列 1 を使用
plotstack iy:=((1,2), (1,3), (1,4)) order:=top;
```

スムージングで微分を検索

differentiate X ファンクションは、Savitsky-Golay スムージングを使って微分を計算できます。この機能を使う場合、**smooth** 変数を 1 にセットします。そして、多項式の次数や Savitzky-Golay スムージング法で使われるウィンドウのポイントを指定して、スムージングをカスタマイズできます。以下はこの例です。

```
// ノイズ付きでサンプルデータをインポート
newbook;
fpath$ = "\Samples\Signal Processing\fftfilter1.DAT";
fname$ = system.path.program$ + fpath$;
impasc;
bkname$=%h;

// Savitsky-Golay スムージングを使って微分
differentiate iy:=col(2) smooth:=1 poly:=1 npts:=30;
```

```
// 元データと結果をプロット
newpanel row:=2;
plotxy iy:=[bkname$]1!2 plot:=200 ogl:=1;
plotxy iy:=[bkname$]1!3 plot:=200 ogl:=2;
```

15.1.3. 積分

integ1 X ファンクションは、積分を使って曲線下の面積を計算します。数学上および絶対値の面積の両方を計算します。次のサンプルでは、絶対値の面積を計算しています。

```
//サンプルデータをインポート
newbook;
fname$ = system.path.program$ + "Samples\Mathematics\Sine Curve.dat";
impasc;

//曲線の面積を計算し、積分曲線をプロット
integ1 iy:=col(2) type:=abs plot:=1;
```

積分を実行すると、結果は integ1 ツリー変数から取得できます。

```
// integ1 ツリーを出力
integ1.=;
// 指定した値を取得
double area = integ1.area;
```

X ファンクションは、関心のある量の変数名を指定します。

```
double myarea, ymax, xmax;
integ1 iy:=col(2) type:=abs plot:=1 area:=myarea y0:=ymax x0:=xmax;
type "area=$(myarea) %(CRLF)ymax=$(ymax) %(CRLF)xmax=$(xmax)";
```

行列の 2 次元データの積分は、**integ2** X ファンクションを使って実行できます。この X ファンクションは、行列で定義される曲面図の下の z=0 平面までの体積を計算します。

```
// 最初の行列シートの最初の行列オブジェクトの体積積分を実行
range rmat=[MBook1]1!1;
integ2 im:=rmat integral:=myresult;
type "Volume integration result:$(myresult)";
```

15.1.4. 補間

補間はデータに実行する一般的な数学関数の一つで、Origin は 2 つの方法(1)範囲表記で一つの値とデータセットの補間、(2)X ファンクションによる全曲線の補間、で補間をサポートしています。

内容

XY 範囲を使う

一度宣言した XY 範囲(X 値による部分範囲指定を利用可能)は、関数として使うことができます。この関数への引数はスカラー(スカラーを返す)またはベクター(ベクターを返す)にすることができます。どちらの場合でも、X データセットは単調に増加、減少している必要があります。例えば、

```
newbook;
wks.ncols = 4;
col(1) = data(1,0,-.05);
col(2) = gauss(col(1),0,.5,.2,100);
range rxy = (1,2);
rxy(.67)=;
range newx = 3; // 列を X 列データとして使用
newx = {0, 0.3333, 0.6667, 1.0}; // 新しい X データ範囲を作成
range newy = 4; // これは補間する空の列
newy = rxy(newx);
```

そして、この範囲変数を以下の形式を持つ関数として使うことができます。

XYRangeVariable(RangeVariableOrScalar[,*connect*[,*param*]])

ここで *connect* は、以下のいずれかを指定します。

line

直線の接続

スプライン

スプライン接続

bspline

b-スプライン接続

そして、*param* はスムージングパラメータで、bspline 接続法にのみ適用されます。*param*=-1 の場合、単純な bspline が使われます。これはプロット内で bspline line 接続と同じになります。*param* >=0 の場合、NAG 関数 **nag_1d_spline_function** が使われます。

> **Note:**XY 範囲の補間を使うとき、***spline*** または ***bspline*** を接続法として使用する場合、***x*** 値が重複してはいけません。代わりに、補間の X ファンクションを使うことができます。

ワークシートデータから

次のサンプルは、関数として範囲を使い、ワークシートからのデータを引数として補間を実行する方法を示します。

サンプル 1:次のコードは、**bspline** のさまざまなスムージングパラメータの使用法を紹介します。

```
col(1)=data(1,9);                  // 列 1 に行番号を入力
col(2)=normal(9);                  //  列 2 に乱数を入力
col(3)=data(1,9,0.01);             // 希望の X 値で Col(3)に入力
wks.col3.type = 4;
range bb=(1,2);                    // cols 1,2 を使用する範囲を宣言;
// 異なるパラメータ設定を使って補間した値を計算
loop(i, 4, 10) {
   wcol(i)=bb(col(3), bspline, $(i*0.1));
}
```

サンプル 2:XY 範囲を使って、次のようなコードを使ってどんな X 値からでも Y 値のデータを取得することができます。

```
// いくつかのデータを生成
newbook;
wcol(1)={1, 2, 3, 4};
wcol(2)={2, 3, 5, 6};
// XY 範囲を定義
range rr =(1,2);
// 線形補間で指定して X 値に対する Y 値を探す
rr(1.23) = ;     // ANS: rr(1.23)=2.23
// 線形補間で X 値の配列に対して Y 値を探す
wcol(3)={1.5, 2.5, 3.5};
range rNewX = col(3);
// 計算した Y 値を保持する新しい列を追加する
wks.addcol();
wcol(4) = rr(rNewX);
```

サンプル 3:指定した Y 値に対する X 値を探す場合、上記の例の引数を逆にします。Y から X を探す場合、Y データセットが単調に増加または減少している必要があります。

```
// いくつかのデータを生成
newbook;
wcol(1)={1, 2, 3, 4};
wcol(2)={2, 3, 5, 6};
// XY 範囲を定義
range rr =(2,1); //X と Y を入れ替え
// 線形補間で指定した Y 値に対する X 値を探す
rr(2.23) = ;     // ANS: rr(2.23)=1.23;
// 計算した X 値を保持する新しい列を追加
wks.addcol();
range rNewX = wcol(3);
// 線形補間で Y 値の配列に対する X 値を探す
wcol(4)={2.5, 3.5, 5.5};
range rNewY = wcol(4);
rNewX = rr(rNewY);
```

グラフから

グラフページがアクティブなときに範囲補間を使うこともできます。

サンプル 1:配列の値の補間

```
// アクティブプロットの範囲を定義
range rg = %C;
```

```
// 線形で使ったスカラー値に対する補間
rg(3.54)=;
// 配列の値に対する補間
// 新しい X 値の位置を与える:
range newX = [Book2]1!1;
// 新しい Y 値(出力)が行くべき場所を与える:
range newY = [Book2]1!2;
// 新しい Y 値を計算する:
newY = rg(newX);
```

サンプル 2:補間法を指定します。

```
// 指定したプロットの範囲を定義
range -wx rWx = 2;      // アクティブレイヤの 2 番目のプロットの X 値を使う
range -w rWy = 2;       // アクティブレイヤで 2 プロット目の Y を使用
range rr = (rWx,rWy);   // 2 つの範囲から XY 範囲を構築
// 新しい X 値の出力場所を定義
range newX = [Book2]1!1;
newX = {5,15,25};
range newY1 = [Book2]1!2; // 新しい Y の範囲
range newY2 = [Book2]1!3; // 新しい Y の範囲
// 線形補間で X 値の配列に対する新しい Y の値を探す
newY1 = rr(newX);
// bspline 補間で X 値の配列に対する新しい Y の値を出力
newY2 = rr(newX,bspline);
```

任意のデータセットを使う

単調増加または減少している同じサイズの 2 つの任意データセットに対して、2 つのデータセットを補間し、片方のデータセットにおける補間値を与える事で、もう片方のデータセットの補間値を出力できます。データセットは範囲変数、データセット変数、列にすることができます。このような補間を実行する形式は次の通りです。

dataset1(value, dataset2)

これは、**dataset2** と **dataset1** で構築される XY データのグループを補間し、与えられた X (**dataset2**) **値**.での Y (**dataset1**) 値を返します。例えば、

```
// データセットを使う
dataset ds1 = {1, 2, 3, 4};
dataset ds2 = {2, 3, 5, 6};
// ds1 の X が 1.23 での補間された値を ds2 に返す
ds2(1.23, ds1) = ;  // 2.23 を返す
// ds2 の X が 5.28 での補間した値を ds1 に返す
ds1(5.28, ds2) = ;  // 3.28 を返す

// 範囲を使う
newbook;
wks.ncols = 3;
range r1 = 2;  // アクティブワークシートの列 2
r1 = {1, 2, 3, 4};
range r2 = 3;  // アクティブワークシートの列 3
r2 = {2, 3, 5, 6};
r2(1.23, r1) = ;
r1(5.28, r2) = ;
```

```
// 列を使う
col(3)(1.23, col(2)) = ;
col(2)(5.28, col(3)) = ;
```

補間曲線を作成する

曲線補間用の X ファンクション

Origin には、XY データを補間して、新しい出力 XY データを作成する 3 つの X ファンクションがあります。

名前	簡単な説明
interp1xy	XY データの補間を実行し、均一な等間隔 X に対して出力します。
interp1	XY データの補間を実行し、入力 X 値に対して出力します。
interp1trace	X が単調でない XY データの補間を実行します。

既存の X データセットを使用

以下の例は、既存の X データセットを使って補間した Y 値を探すものです。

```
// 指定した列の属性を持つ新しいワークブックを作成
newbook sheet:=0;
newsheet cols:=4 xy:="XYXY";
// サンプルデータファイルをインポート
fname$ = system.path.program$ + "Samples\Mathematics\Interpolation.dat";
impasc;

// col(3)の X 値で col(1) と col(2) のデータを補間
range rResult=col(4);
interp1 ix:=col(3) iy:=(col(1), col(2)) method:=linear ox:=rResult;

//元のデータと結果をプロット
plotxy iy:=col(2) plot:=202 color:=1;
plotxy iy:=rResult plot:=202 color:=2 size:=5 ogl:=1;
```

均一で等間隔な X 出力

次のサンプルは、均一で等間隔な X 値を出力として生成し、補間を実行します。

```
//新しいワークブックを作成し、データファイルをインポート
fname$ = system.path.program$ + "Samples\Mathematics\Sine Curve.dat";
newbook;
impasc;

//列 2 でデータを補間
interp1xy iy:=col(2) method:=bspline npts:=50;
range rResult = col(3);

//元のデータと結果をプロット
plotxy iy:=col(2) plot:=202 color:=1;
```

```
plotxy iy:=rResult plot:=202 color:=2 size:=5 ogl:=1;
```

非単調データを補間

次のサンプルは X 値が単調でないデータに対してトレース補間を実行します。

```
//新しいワークブックを作成し、データファイルをインポート
fname$ = system.path.program$ + "Samples\Mathematics\circle.dat";
newbook;
impasc;

//トレース補間で列 2 の循環データを補間
interp1trace iy:=Col(2) method:=bspline;
range rResult= col(4);

//元のデータと結果をプロット
plotxy iy:=col(2) plot:=202 color:=1;
plotxy iy:=rResult plot:=202 color:=2 size:=1 ogl:=1;
```

補間の X ファンクションは、入力データの X 範囲の外側にある Y 値を補外するのに使用することもできます。

行列の補間

minterp2 X ファンクションは、行列の補間/補外を実行するのに使うことができます。

```
// 新しい行列ブックを作成し、サンプルデータをインポート
newbook mat:=1;
filepath$ = "Samples\Matrix Conversion and Gridding\Direct.dat";
string fname$=system.path.program$ + filepath$;
impasc;
// 元の X および Y データサイズの 10 倍で行列を補間
range rin = 1; // 入力データで行列をポイント
int nx, ny;
nx = rin.ncols * 10;
ny = rin.nrows * 10;
minterp2 method:=bicubic cols:=nx rows:=ny ;
```

OriginPro は、**interp3** X ファンクションも提供しており、これは、4 次の散布データを補間するのに使うことができます。

15.2. 統計

これは、X ファンクションを呼び出して、スクリプトで組み込まれたいくつかの統計検定を行うサンプルのセクションです。

このセクションで説明している項目

- 記述統計量
- 仮説検定
- ノンパラメトリック検定

- 生存分析

15.2.1. 記述統計量

Origin には記述統計量を計算するいくつかの X ファンクションがあり、最も一般的なものには以下のものがあります。

X ファンクション	簡単な説明
colstats	列の統計
corrcoef (Pro のみ)	相関係数
freqcounts	データセットの度数カウント
mstats (Pro のみ)	行列の記述統計を計算する
rowstats	データの行の統計
stats	選択したデータセットを完全なデータセットとして扱い、データセットの統計を計算します。

これらの X ファンクションのそれぞれの説明については、記述統計量をご覧下さい。

列と行の記述統計量

X ファンクション **colstats** は列の統計を実行できます。デフォルトで、これは各入力列の平均、標準偏差、データポイント数、中央値を出力します。しかし、変数に異なる値を割り当て、出力をカスタマイズすることができます。次の例では、**colstats** を使って、4 つの列の平均、標準偏差、平均の標準誤差、中央値を計算します。

```
//4 列あるサンプルデータをインポート
newbook;
fname$ = system.path.program$ + "Samples\Statistics\nitrogen_raw.txt";
impasc;

//列 1 から 4 の記述統計を実行
colstats irng:=1:4 sem:=<new> n:=<none>;
```

rowstats X ファンクションを同様の方法で使うことができます。次の例はアクティブなワークシートの平均を算出します。結果は、追加された新しい列に入力されます。

Note:平均 と標準偏差は、デフォルトで出力が<new>になります。必要無い場合は<none>に設定してください。

```
newbook;
fname$ = system.path.program$ + "Samples\Statistics\engine.txt";
impasc; //サンプルデータをインポート

wunstackcol irng1:=1 irng2:=2; //列のアンスタック
wtranspose type:=all ow:=<new>; //ワークシートを置き換える
range rr1 = 1:2;
delete rr1;
range rr2 = 2;
```

```
delete rr2; //空列を削除
int nn = wks.ncols;
wks.addcol();
wks.col$(nn+1).lname$ = Mean;
wks.col$(nn+1).index = 2; //平均列を追加
wks.addcol();
wks.col$(nn+2).lname$ = Sum;
wks.col$(nn+2).index = 3; //合計列を追加

//行の統計をとり、合計と平均を算出後、対応する列に保存
rowstats irng:=4[1]:end[end] sum:=3 mean:=2 sd:=<none>;
```

度数カウント

データ範囲の度数カウントを計算する場合、**freqcounts** X ファンクションを使います。

```
//サンプルワークブックを開く
%a = system.path.program$ + "Samples\Statistics\Body.ogw";
doc -a %a;

//列 4 にあるデータの度数を数える
freqcounts irng:=4 min:=35 max:=75 stepby:=increment intervals:=5;
```

相関係数

corrcoef X ファンクションは、2 つのデータセット間の相関係数 を計算するのに使うことができます。

```
//サンプルデータのインポート
newbook;
fname$ = system.path.program$ + "Samples\Statistics\automobile.dat";
impasc;

//相関係数
corrcoef irng:= (col(c):col(g)) rt:= <new name:=corr>
```

15.2.2. 仮説検定

Origin/OriginPrio は、仮説検定のためのいくつかの X ファンクションをサポートしています。

名前	簡単な説明
rowttest2 (Pro のみ)	行の 2 標本の t 検定を実行します。
ttest1	仮説を立てる母平均の標本の平均を比較します。
ttest2	2 つの標本の平均を比較します。
ttestpair	2 つの平均が一致する場合に、2 つの標本が等しいかどうかを検定します。
vartest1 (Pro のみ)	標本の分散が指定した値に等しいかどうかを検定します。

vartest2 (Pro のみ)	2 つの標本の分散が等しいかどうかを検定します。

これらの X ファンクションの入出力の引数など完全な説明は、オンラインヘルプの 仮説検定をご覧下さい。

内容

1 集団の t 検定

与えられた信頼水準で標本の平均が仮説値と一致するかどうかを知る必要がある場合、**1 集団の T 検定**を使うことを考えます。この検定は、標本が正規分布しているものと見なして行います。1 集団の t 検定を行う前に、この前提を確認する必要があります。

```
//サンプルデータをインポート
newbook;
fname$ = system.path.program$ + "Samples\Statistics\diameter.dat";
impasc;

//正規性の検定
swtest irng:=col(a) prob:=p1;
if (p1 < 0.05)
{
        type "The sample is not likely to follow a normal distribution."
}
else
{
        // 平均が 21 かどうかを調べる
        ttest1 irng:=col(1) mean:=21 tail:=two prob:=p2;
        if (p2 < 0.05) {
                type "At the 0.05 level, the population mean is";
                  type "significantly different from 21."; }
        else {
                type "At the 0.05 level, the population mean is NOT";
                  type "significantly different from 21."; }
}
```

2 集団の t 検定

rowttest2 X ファンクションは、行に対して 2 集団の t 検定を実行するのに使うことができます。次の例は、各行に対応する確率値を計算する方法を示しています。

```
// サンプルデータをインポート
newbook;
string fpath$ = "Samples\Statistics\time_raw.dat";
```

```
string fname$ = system.path.program$ + fpath$;
impAsc;

// 行に対する 2 集団の t 検定を実行
// 標本分散が等しいとは仮定されていません
ttest2 irng:=(col(1), col(2)) equal:=0;

// 結果を出力
type "Value of t-test statistic is $(ttest2.stat)";
type "Degree of freedom is $(ttest2.df)";
type "P-value is $(ttest2.prob)";
type "Conf. levels in 95% is ($(ttest2.lcl), $(ttest2.ucl))";
```

rowttest2 X ファンクションは、行に対して **2 集団の t 検定**を実行するのに使うことができます。次の例は、各行に対応する確率値を計算する方法を示しています。

```
// サンプルデータのインポート
newbook;
string fpath$ = "Samples\Statistics\ANOVA\Two-Way_ANOVA_raw.dat";
fname$ = system.path.program$ + fpath$;
impasc;

// 行に対して 2 集団の t 検定
rowttest2 irng1:=(col(a):col(c)) irng2:=(col(d):col(f))
          tail:=two prob:=<new>;
```

対応のある t 検定

X ファンクション ttestpair を使用して、対応のある t 検定を実行でき、正規分布からとられた同じサンプルサイズの 2 つの標本の平均が等しいかそうでないかを調べ、平均間の差に対する信頼区間を計算します。以下のサンプルでは、まずデータをインポートし、対応のある t 検定を実行して関連する結果を出力します。

```
// サンプルデータをインポート
newbook;
string fpath$ = "Samples\Statistics\abrasion_raw.dat";
string fname$ = system.path.program$ + fpath$;
impasc;

// 1、2 列で対応のある t 検定を実行
// 仮説平均の差は 0.5
// 上側の検定を行う
ttestpair irng:=(col(1), col(2)) mdiff:=0.5 tail:=upper;

// 結果を表示
type "Value of paired-sample t-test statistic is $(ttestpair.stat)";
type "Degree of freedom for the paired-sample t-test is $(ttestpair.df)";
type "P-value is $(ttestpair.prob)";
type "Conf. levels in 95% is ($(ttestpair.lcl), $(ttestpair.ucl))";
```

1 集団の分散検定

X ファンクション vartest1 は、カイ二乗分散検定を実行し、正規分布からの標本データが与えられた仮説の分散値を持つかどうかを調べます。以下のサンプルは **1 集団の分散の検定**を実行し、P 値を求めます。

```
// サンプルデータをインポート
newbook;
string fpath$ = "Samples\Statistics\vartest1.dat";
string fname$ = system.path.program$ + fpath$;
impasc;

// F 検定を実行
// 両側検定
// 仮説分散値は 2.0
// P 値を p とする
vartest1 irng:=col(1) var:=2.0 tail:=two prob:=p;

// P 値を出力
p = ;
```

2 集団の分散の検定(F 検定)

F 検定(または 2 集団の分散の検定)は X ファンクション vartest2 を使用して実行します。

```
// サンプルデータをインポート
newbook;
string fpath$ = "Samples\Statistics\time_raw.dat";
string fname$ = system.path.program$ + fpath$;
impasc;

// F 検定を実行
// 上側を検定
vartest2 irng:=(col(1), col(2)) tail:=upper;

// 結果ツリーを出力
vartest2.=;
```

15.2.3. ノンパラメトリック検定

仮説検定は、母集団が、あるパラメータで特定の分布(正規分布など)に従うと仮定しているパラメトリック検定です。データが正規分布に従うかどうかわからなかったり、正規分布に従わないことを確認していれば、ノンパラメトリック検定を行います。

Origin はノンパラメトリック検定に対して、次の X ファンクションをサポートしています。これらは OriginPro でサポートしている機能です。

名前	簡単な説明
signrank1	母分布の位置(メディアン)が指定した値と同じであるかどうかを検定します。
signrank2/sign2	対の母集団の中央値が等しいかどうかを検定するのに使用します。入力データは素データの形式です。
mwtest/kstest2	2 つの標本が同じ分布であるかどうかを検定します。入力データはインデックス化されています。
kwanova/mediantest	異なる標本の中央値が等しいかどうか検定し、入力データはインデックスモードで配置されます。

friedman	3 つ以上の対のグループを比較します。入力データはインデックスで配置されます。

例として、高校の男子生徒と女子生徒の身長を比較します。

```
//サンプルデータをインポート
newbook;
fname$ = system.path.program$ + "Samples\Statistics\body.dat";
impasc;

//2 標本の Mann-Whitney 検定
//mynw という名前の新しいシートに結果を出力
mwtest irng:=(col(c), col(d)) tail:=two rt:=<new name:=mynw>;

//出力結果シートから結果を取得
page.active$="mynw";

getresults tr:=mynw;

//結果を使用し結論を描画
if (mynw.Stats.Stats.C3 <= 0.05); //確率が 0.05 以下の場合
{
        type "At 0.05 level, height of boys and girls are differnt.";
        //女子の身長が男子の身長の中央値より大きい場合
        if (mynw.DescStats.R1.Median >= mynw.DescStats.R2.Median)
                type "girls are taller than boys.";
        else
                type "boys are taller than girls."
}
else
{
        type "The girls are as tall as the boys."
}
```

15.2.4. 生存分析

生存分析は、バイオサイエンスの分野で、研究対象の母集団の中から生存数を定量化するのに使用されます。Origin は 3 つの幅広い検定をサポートしており、OriginPro では使用できます。

名前	簡単な説明
kaplanmeier	Kaplan-Meier (積極限) 推定量
phm_cox	Cox 比例ハザードモデル
weibullfit	ワイブルフィット

これらの X ファンクションの入出力の引数など完全な説明は、オンラインヘルプの 生存分析をご覧下さい。

Kaplan-Meier 分析

生存率を推定したければ、**kaplanmeier** X ファンクションを使って、生存グラフを作成し、生存関数の性質を比較します。これは、積極限を使って、生存関数を推定し、生存関数の同等性を検定するため、Log Rank, Breslow, Tarone-Ware の 3 つの手法をサポートしています。

例えば、生化学者がより良い抗ガン剤を探しているとします。いくつかのラットに発ガン物質 DMBA を与えた後、ラットの異なるグループに異なる薬を投与し、最初の 60 時間の生存状況を記録します。2 種類の薬で生存率の違いを定量化したいと考えます。

```
// サンプルデータをインポート
newbook;
fname$ = system.path.program$ + "Samples\Statistics\SurvivedRats.dat";
impasc;

//Kaplan-Meier 分析を実行
kaplanmeier irng:=(1,2,3) censor:=0 logrank:=1
            rd:=<new name:="sf">
            rt:=<new name:="km">;

//生存レポートツリーから結果を取得
getresults tr:=mykm iw:="km";

if (mykm.comp.logrank.prob <= 0.05)
{
    type "The two medicines have significantly different"
    type "effects on survival at the 0.05 level ...";
    type "Please see the survival plot.";

    //生存関数をプロット
    page.active$="sf";
    plotxy iy:=(?, 1:end) plot:=200 o:=[<new template:=survivalsf>];

}
else
{
    type "The two medicines are not significantly different.";
}
```

Cox 比例ハザード回帰

phm_cox X ファンクションは、固定の共変量に対する Cox 比例ハザードモデルに関連するパラメータの見積もりおよびその他の統計量を取得するのに使うことができます。いくつかの固定の共変量に沿ってハザード率の変化を予測できます。

例えば、結腸直腸癌の 66 名の患者に対して、効果的な予後因子と最適な予後指標を決めたいとします。(予後因子は、人がある病気であるかどうかを決めるパラメータ)このスクリプトは、関連のある統計量を取得するため、**phm_cox** X ファンクションを実装しています。

```
//サンプルデータをインポート
newbook;
string fpath$ = "Samples\Statistics\ColorectalCarcinoma.dat";
fname$ = system.path.program$ + fpath$;
impasc option.hdr.LNames:=1
       option.hdr.units:=0
       option.hdr.CommsFrom:=2
       option.hdr.CommsTo:=2;

// Cox 回帰を実行
phm_Cox irng:=(col(1),col(2),col(3):end) censor:=0 rt:=<new name:="cox">;

//レポートツリーから結果を取得
```

```
page.active$="cox";
getresults tr:=cox;

type "Prognostic parameters determining colorectal carcinoma are:";

page.active$="ColorectalCarcinoma";
loop(ii, 1, 7)
{
  // 確率が 0.05 以下の場合
  // 生存時間に対して効果ありと言える
  if (cox.paramestim.param$(ii).prob<=0.05)
    type wks.col$(ii+2).comment$;
}
```

Weibull フィット

データがワイブル分布であると*前もって*分かっている場合、**weibullfit** X ファンクションを使って、ワイブルパラメータを推定できます。

```
//サンプルデータをインポート
newbook;
fname$ = system.path.program$ + "Samples\Statistics\Weibull Fit.dat ";
impasc;

//ワイブルフィットを実行
weibullfit irng:=(col(a), col(b)) censor:=1;
```

15.3. カーブフィット

Origin のカーブフィット（非線形曲線フィット）の機能は、とても便利で、幅広く使用されています。多くのユーザは、スクリプトからフィット計算を実行する X ファンクションをユーザインターフェースと同じくらい簡単に使えるとは思っていないかもしれません。次のセクションでは、LabTalk スクリプトを使用してカーブフィットを実行する方法を説明します。

このセクションで説明している項目

- 線形、多項式、線形多重回帰
- 非線形曲線フィット

15.3.1. 線形、多項式、線形多重回帰

LabTalk スクリプトでは、3 つの X ファンクション **fitLR**、**fitPoly**、**fitMR** を使用して線形フィット、多項式フィット、線形多重回帰を実行できます。**-h** スイッチを使用すれば、引数のリストを確認できます。

内容

- 4 オペレーションクラスを実行して回帰を行う

線形回帰

fitLR は、目的のデータセットに対し、もっとも当てはまりがよい直線を作成します。

```
newbook;  // 新しいブックを作成

// ファイル名
string strFile$ = system.path.program$ + "Samples\Curve Fitting\Linear Fit.dat";
impasc fname:=strFile$;  // データインポート

wks.addcol(FitData);  // フィットしたデータ用に FitData という列を作成

// 列 1 (X) と列 2 (Y) の最初の 10 データに対し、線形フィットを実行
// フィットしたデータは FitData 列に出力
fitLR iy:=(1,2) N:=10 oy:=col(FitData);
//ツリーオブジェクト、fitLR を作成し、出力値を格納
fitLR.a = ;  // 切片を出力
fitLR.b = ;  // 傾きを出力
fitLR.= ;  // 切片と傾きを含む、全ての結果を出力
```

線形フィットの他のサンプルは曲線フィットサンプルページにあります。また、**XF スクリプトダイアログ**の **Fitting** カテゴリー内でも確認できます（F11 で開く）。

多項式回帰

では、フィット関数が数学的には非線形ですが、反復無しの分析解を得る特殊なケースです。LabTalk で多項式フィットを行うには、**fitPoly** を利用できます。

```
newbook;  // 新規ブックを作成

// ファイル名
string strFile$ = system.path.program$ + "Samples\Curve Fitting\Polynomial
Fit.dat";
impasc fname:=strFile$;  // データをインポート
wks.addcol(PolyCoef);  // 多項式係数用の新規列を追加
wks.addcol(FittedX);  // フィット曲線の X データ用に新規列を追加
wks.addcol(FittedY);  // フィット曲線の Y データ用に新規列を追加

// 列 1 (X) と列 3 (Y) に多項式フィットを実行
// 多項式次数は 3
fitPoly iy:=(1,3) polyorder:=3 coef:=col(PolyCoef)
oy:=(col(FittedX),col(FittedY));

// fitPoly というツリーに格納された結果を出力
fitPoly.= ;
```

さらに、**fitPoly** では、補正済み残差平方和、決定係数、多項式係数の誤差を出力できます。より詳しいサンプルは、曲線フィットサンプルページにあります。または、**XF スクリプトダイアログ**の **Fitting** カテゴリー内で確認できます（F11 で開く）。

線形多重回帰

多重線形回帰はシンプルな線形回帰を拡張したもので、複数の説明変数と目的変数間の関係性を評価します。

```
// 新しいブックを作成し、データをインポート
newbook;
fn$ = system.path.program$ + "Samples\Curve Fitting\Multiple Linear
Regression.dat";
impasc fn$;
wks.addcol(FitValue);  // フィットした従属変数（Y 値）用の列を追加

// 多重線形回帰を実行
// 列 D は従属、列 A、B、C は独立変数
// 出力結果は tr ツリーに格納される
fitMR dep:=col(D) indep:=col(A):col(C) mrtree:=tr odep:=col(FitValue);
tr.= ;  // 結果ツリーを出力
```

他のサンプルは、曲線フィットサンプルページにあります。または、**XF スクリプトダイアログ**の **Fitting** カテゴリー内で確認できます（F11 で開く）。

オペレーションクラスを実行して回帰を行う

上記で説明した X ファンクションは、シンプルで素早く線形、多項式、多重回帰を行うためのものです。つまり、これら 3 つの X ファンクションを使用する際は、いくつかの値の指定ができません。全ての値にアクセスをするには、**xop** X ファンクションを使用します。これは、内部メニューコマンド（オペレーションコマンド）を呼び出し、回帰を行うためのオペレーションクラスを実行します。次のサンプルは、**xop** X ファンクションを使って線形フィットを実行し、レポートを作成する方法です。

```
// 新しいブックを作成し、データをインポート
newbook;
fname$ = system.path.program$ + "Samples\Curve Fitting\Linear Fit.dat";
impasc fname$;

tree lrGUI;  // 線形フィットのための GUI ツリー
// FitLinear クラスで GUI ツリーを初期化
xop execute:=init classname:=FitLinear iotrgui:=lrGUI;

// GUI ツリーに入力するデータを指定
lrGUI.GUI.InputData.Range1.X$ = col(A);
lrGUI.GUI.InputData.Range1.Y$ = col(C);

// 線形フィットを実行し、準備した GUI ツリーにレポートを生成
xop execute:=report iotrgui:=lrGUI;

xop execute:=cleanup;  //フィット完了後に線形フィットオブジェクトをリセットする
```

15.3.2. 非線形フィット

LabTalk の非線形フィットは、X ファンクションベースのコマンドです。3 つのステップで操作し、各ステップで少なくとも 1 つの X ファンクションを呼び出します。

1. nlbegin:フィット処理を開始します。入力データ、フィット関数の種類、入力パラメータを定義します。
2. nlfit:フィット計算を実行します。

3. nlend どのパラメータをどのような形式で出力するかを選択します。

nlbegin ではなく、以下の X ファンクションでフィットモデルやデータにしたがってフィット処理を開始することもできます。

- nlbeginr:複数の従属/独立変数のモデルをフィットします。
- nlbeginm:行列をフィットします。
- nlbeginz:XYZ ワークシートデータをフィットします。

内容

スクリプトサンプル

以下は、上記で簡単に説明した手順のスクリプトサンプルです。

```
// 非線形フィットを開始、アクティブワークシートから
// 列 1 (X)と列 2 (Y)の入力データを取り
// フィット関数を Gaussian として指定し
// ParamTree という入力パラメータツリーを作成
nlbegin iy:=(1,2) func:=gauss nltree:=ParamTree;
  // 任意: ピークの中心を X = 5 で固定
  ParamTree.xc = 5;      // ピークの中心の X の値に 5 を割り当て
ParamTree.f_xc = 1;    // ピーク中心を固定(f_xc = 0 は固定ではない)
// フィットを実行
nlfit;
// 任意: スクリプトウィンドウに結果を表示
type Baseline y0 is $(ParamTree.y0),;
  type Peak Center is $(ParamTree.xc), and;
  type Peak width (FWHM) is $(ParamTree.w);
// レポートシートなしでフィットセッションを終了
nlend;
```

パラメータツリーの注意

フィットパラメータを保存するデータツリーには、上記サンプルで述べた以外にも多くのオプションがあります。以下のスクリプトコマンドは、ツリーノードのすべて(名前と値)を、**スクリプトウィンドウ**に表示することで、同時に見ることができます。

```
// 値を持つ全ツリー構造を見るには
ParamTree.=;
```

Note: 計算を行うので、(パラメータツリーで固定オプションを 0 にセットして)固定でないフィットのパラメータ値は、初期値から変動します。初期パラメータは、パラメータツリーの個々のノードにアクセスすることで、上記のサンプルで示すように、手動でセットしたり、Origin で自動的にセットできます(下表の **nlfn** X ファンクションをご覧下さい。)。

非線形フィットをサポートする X ファンクションのテーブル

上記で与えられた 3 つに加え、非線形フィットを手助けする X ファンクションがあります。下表は、非線形フィットを制御するのに使用する X ファンクションをまとめたものです。

名前	簡単な説明
nlbegin	ワークシートまたはグラフにある XY データに対して LabTalk の nlfit セッションを開始します。 Note: この X ファンクションは、独立/従属モデルのみをフィットします。複数の従属/独立関数に対しては、代わりに *nlbeginr* を使用します。
nlbeginr	ワークシートデータに対して LabTalk の nlfit セッションを開始します。複数の従属/独立変数の関数をフィットに使用します。
nlbeginm	行列オブジェクトまたはグラフから行列データに対して LabTalk の nlfit セッションを開始します。
nlbeginz	ワークシートまたはグラフにある XYZ データに対して LabTalk の nlfit セッションを開始します。
nlfn	パラメータ自動初期化オプションをセットします。
nlpara	パラメータ値と境界の GUI 編集のためのパラメータダイアログを開きます。
nlfit	反復計算を実行してデータをフィットします。
nlend	フィットセッションを終了し、任意でレポートを作成します。

これらの X ファンクションのそれぞれの説明については、X ファンクションリファレンスをご覧下さい。

線形フィットとの質の違い

線形フィットとは違い、非線形フィットは、分析解が無い方程式を反復計算によって、解きます。しかし、X ファンクションを呼び出して分析を実行するという考え方は同じです。線形フィットが 1 つの X ファンクション（線形フィットのセクションを参照)で 1 行のスクリプトを実行するのに対して、非線形フィットは少なくとも 3 つの X ファンクションが必要です。

15.4. 信号処理

Origin は、信号処理、ノイズデータのスムージングから FFT、ショートタイム FFT、コンボリューションと相関、FFt フィルタリング、ウェーブレット分析の X ファンクションを提供しています。

これらの X ファンクションは、**信号処理**カテゴリーで利用でき、次のコマンドを実行して、一覧表示できます。

```
lx cat:="signal processing*";
```

ショートタイム FFT やウェーブレットなどのいくつかの機能は、OriginPro でのみ利用できます。

次のセクションでは、スクリプトから信号処理の X ファンクションを呼び出すいくつかの簡単なサンプルを提供しています。

15.4.1. スムージング

ノイズデータのスムージング は、**smooth** X ファンクションを使って実行することができます。

```
// 3 次多項式で SavitzkyGolay 法を使って
// ワークシートの列 1,2 の XY データをスムージング
range r=(1,2); // ワークシートに XY があり、アクティブである前提
smooth iy:=r meth:=sg poly:=3;
```

レイヤ内のすべてのプロットをスムージングするため、以下のプロットをループ

```
// レイヤ内のデータプロットの数を数え、結果を
// 変数"count"に保存
layer -c;
// このグラフページの名前を取得
string gname$ = %H;
// smooth という名前の新しいブックを作成 - 実際の名前は bkname$に保存されている
newbook na:=Smoothed;
// 列なしで開始
wks.ncols=0;
loop(ii,1,count) {
    // 入力範囲は'ii'番目のプロットを参照
    range riy = [gname$]!$(ii);
    // 出力範囲は 2 つの新しい列を参照
   range roy = [bkname$]!($(ii*2-1),$(ii*2));
    // 3 次多項式を使った Savitsky-Golay スムージング
    smooth iy:=riy meth:=sg poly:=3 oy:=roy;
}
```

15.4.2. FFT とフィルタリング

次のサンプルは、**fft1** X ファンクションを使ったデータの 1D FFT を実行する方法を示しています。

```
// サンプルファイルをインポート
newbook;
fname$ = system.path.program$ + "Samples\Signal Processing\fftfilter1.dat";
impasc;
// FFT を実行し、名前を付けたツリーに出力
Tree myfft;
fft1 ix:=2 rd:=myfft rt:=<none>;
// コマンド list vt を使ってすべてのツリーを一覧表示
```

ツリーに結果を格納すると、その出力に対して次のように分析を行うことができます。

```
// 目的のツリーベクターノードをデータセットにコピー
// ピークを位置づけ、周波数成分を平均
dataset tmp_x=myfft.fft.freq;
dataset tmp_y=myfft.fft.amp;
// stats を実行し、結果を出力
percentile = {0:10:100};
diststats iy:=(tmp_x, tmp_y) percent:=percentile;
type "The mean frequency is $(diststats.mean)";
```

次のサンプルは、**fft_filters** X ファンクションを使ったデータの信号 フィルタリング を実行する方法を示しています。

```
// ノイズ入りのデータをインポートする
newbook;
string filepath$ = "Samples\Signal Processing\";
string filename$ = "Signal with High Frequency Noise.dat";
fname$ = system.path.program$ + filepath$ + filename$;
impasc;
plotxy iy:=(1,2) plot:=line;

// ローパスフィルタを実行
fft_filters filter:=lowpass cutoff:=1.5;
```

15.5. ピークと基線

このセクションでは、ピークと基線の計算を実行する Origin の X ファンクション、特にスペクトル分析で役立つ分析を扱います。

内容

15.5.1. ピーク分析のための X ファンクション

次の表は、ピーク分析で利用できる X ファンクションを一覧表示します。これらの関数についての情報は、X ファンクションリファレンスヘルプにあります。

名前	簡単な説明
pa	事前に保存した**ピークアナライザ**テーマファイルを使ってピーク分析を実行します。
paMultiY	複数の Y データセットに対してピーク分析のバッチ処理を実行します。
pkFind	ピークを検出します。
fitpeaks	複数ピークをフィットします。
blauto	基線のアンカーポイントを作成します。
interp1xy	基線のアンカーポイントを補間し、基線を作成します。
subtract_ref	既存の基線データセットを元のデータから減算します。
smooth	ピーク分析を実行する前に入力をスムージングします。

integ1	選択した範囲またはピークに積分を実行します。

基線の処理やその他のオプションを必要としないピークに対して、ピーク関数を使って非線形フィットを実行できます。スクリプトから非線形フィットを実行する詳細な情報は、曲線フィットセクションをご覧ください。

次のセクションは、ピーク分析のサンプルを提供するものです。

15.5.2. 基線を作成する

このサンプルは、サンプルデータファイルをインポートし、**blauto** X ファンクションを使って、基線のアンカーポイントを作成します。

```
newbook;
filepath$ = "Samples\Spectroscopy\Peaks on Exponential Baseline.dat";
fname$ = system.path.program$ + filepath$;
impASC;

//20 個の基線のアンカーポイントを作成
range rData = (1,2), rBase =(3, 4);
blauto iy:=rData number:=20 oy:=rBase;
```

同じグラフにデータとアンカーポイントをプロットします。

```
// データの折れ線グラフをプロット
plotxy rData 200 o:=[<new>];
// 基線のポイントを同じレイヤに散布図でプロット
plotxy rBase 201 color:=2 o:=1!;
```

15.5.3. ピークを検索する

このサンプルは、**pkfind** X ファンクションを使って、XY データのピークを探します。

```
// サンプルのパルスデータをインポート
newbook;
fname$ = system.path.program$ + "Samples\Spectroscopy\Sample Pulses.dat";
impASC;
// ピーク高さ 0.2 以上のすべての正ピークを探す
range rin=(1,2);
range routx = 3, routy=4;
pkfind iy:=rin dir:=p method:=max npts:=5 filter:=h value:=0.2
       ocenter:=<none> ocenter_x:=routx ocenter_y:=routy;
```

データを折れ線グラフでプロットし、ピークの xy データを散布図でプロット

```
plotxy iy:=rin plot:=200;
// x 出力列を X としてセットし、Y 列をプロット
routx.type = 4;
plotxy iy:=routy plot:=201 color:=2 o:=1;
```

15.5.4. ピークの積分とフィット

X ファンクションは、直接ピークを積分するか、複数のピークをフィットすることを目的にするよう指定します。そのため、ピークフィットまたは積分を実行するには、まず、**ピークアナライザ**ダイアログを作成し、それをテーマファイルとして保存します。テーマファイルを保存したら、**pa** または **paMultiY** X ファンクションを利用して、スクリプトから積分またはピークフィットを実行します。

15.6. 画像処理

Origin 8 以降のバージョンでは、以前と比べ、画像処理の機能が拡張されています。基本的な画像処理のサンプルを必要な操作を実行する LabTalk スクリプトとして以下に示します。

画像処理を行うのに利用可能なすべての X ファンクションを表示するには、次のコマンドを使います。

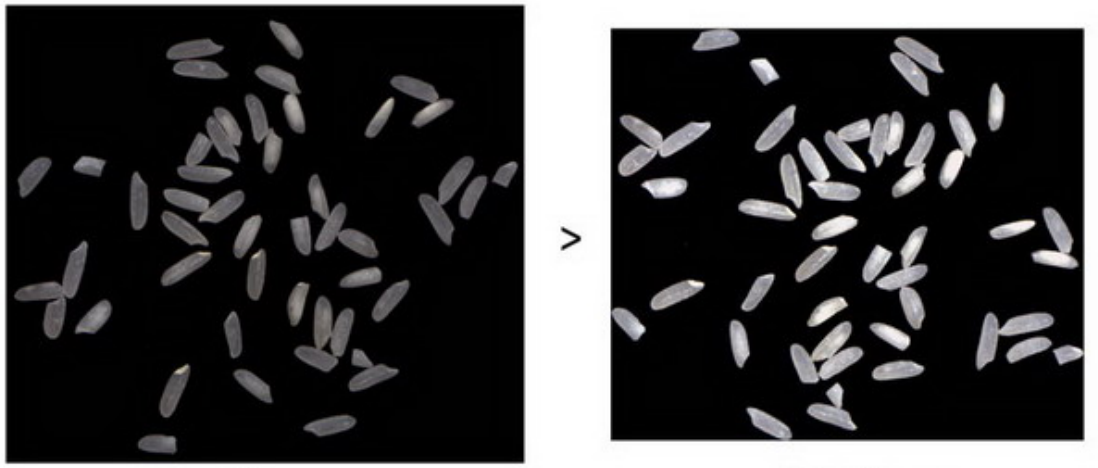
```
lx cat:="image*";
```

いくつかの X ファンクションは OriginPro でのみ利用できます。

内容

15.6.1. 回転および画像を小さくする

この例では、画像を回転し、余白を切り取り、自動レベルを適用して、より小さく、明確にします。



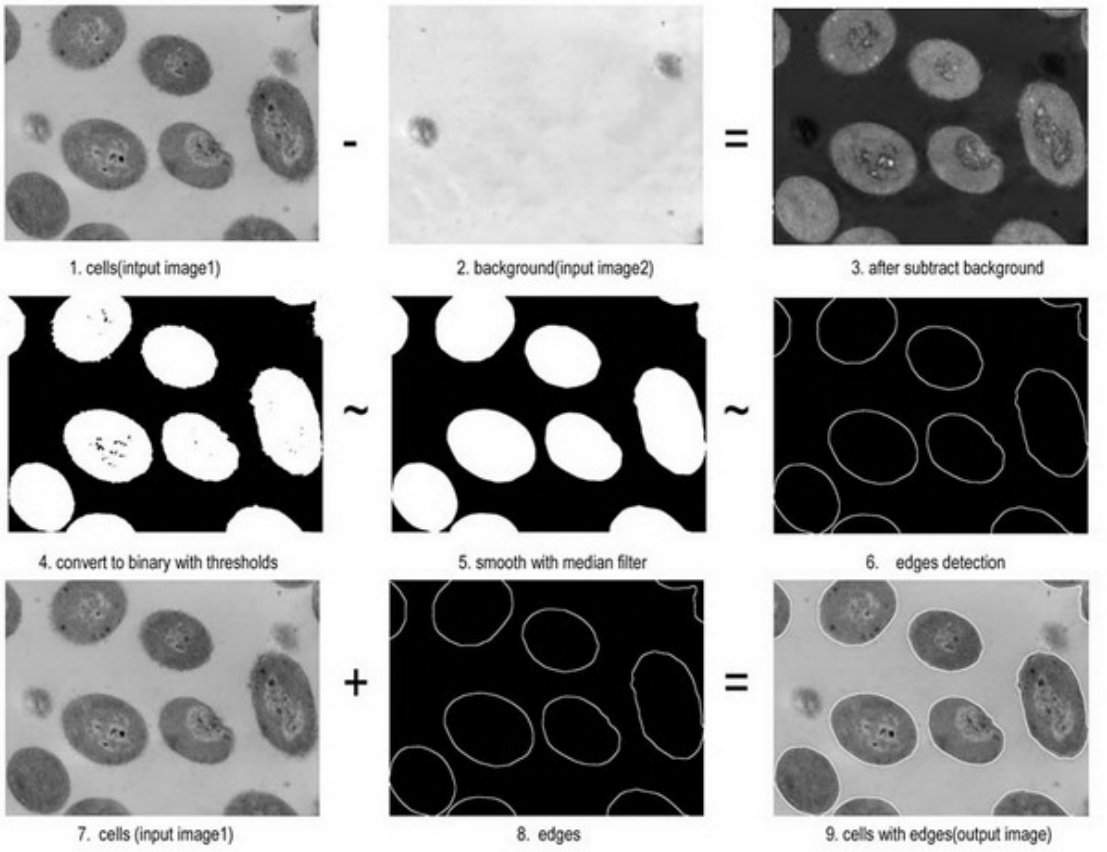
Input Image　　Output Image

```
//プロジェクトエクスプローラに新しいフォルダを作成
pe_mkdir RotateTrim path:=aa$;
pe_cd aa$;

//行列を作成し、そこに画像をインポート
```

```
window -t m;
string fpath$ = "samples\Image Processing and Analysis\rice.bmp";
string fname$ = System.path.program$ + fpath$;
impimage;
window -r %h Original;

//元の画像の次数を取得
matrix -pg DIM nCol1 nRow1;

window -d; //画像を複製
window -r %h Modified;

imgRotate angle:=42;
imgTrim t:=17;

matrix -pg DIM nCol2 nRow2; //修正した画像の次数を取得

imgAutoLevel;// 画像に自動レベルを適用

window -s T;        //ウィンドウを水平に並べて表示

//レポート
window -n n Report;
old = type.redirection;
type.redirection = 2;
type.notes$=Report;
type "Dimension of the original image:";
type "   $(nCol1) * $(nRow1)\r\n";    //  "754 * 668"
type "Dimension of the modified image:";    //  "688 * 601"
type "   $(nCol2) * $(nRow2)\r\n";
type.redirection = old;
```

15.6.2. エッジ検出

細胞の画像から背景を除去し、エッジを検出します。

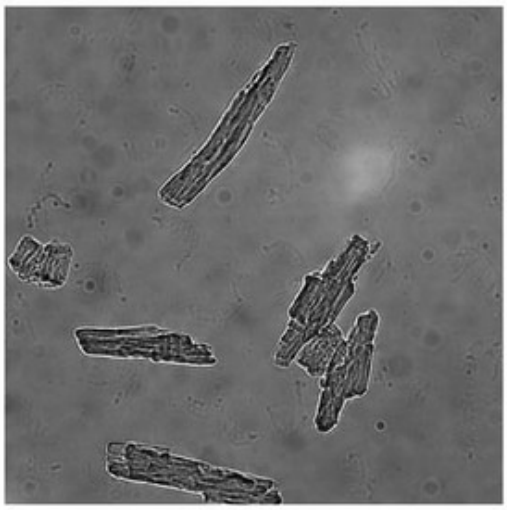



```
//プロジェクトエクスプローラに新しいフォルダを作成
pe_mkdir EdgeDetection path:=aa$;
pe_cd aa$;

//行列を作成し、そこに細胞の画像をインポート
window -t m;
string fpath$ = "samples\Image Processing and Analysis\cell.jpg";
string fname$ = System.path.program$ + fpath$;
impimage;
cell$ = %h;

//行列を作成し、そこに背景の画像をインポート
window -t m;
string fpath$ = "samples\Image Processing and Analysis\bgnd.jpg";
string fname$ = System.path.program$ + fpath$;
impimage;
cellbk$ = %h;


//背景を除去し、前処理を行う
//x, y は Image2 のオフセット
imgSimpleMath img1:=cellbk$ img2:=cell$ func:=sub12 x:=7 y:=13 crop:=1;
//最低および最高の明度をバイナリ 0 または 1 に指定する
imgBinary t1:=65 t2:=255;
```

```
// メディアンフィルタの次数は 18
imgMedian d:=18;

//エッジ検出
// エッジピクセルを決定するのに使用するしきい値 12 を指定
// そして、shv(Sobel 水平&垂直) エッジ検出フィルタを適用
imgEdge t:=12 f:=shv;
edge$ = %h;

//エッジを細胞の画像に追加して戻す
imgSimpleMath img1:=edge$ img2:=cell$ func:=add;
window -z;
```

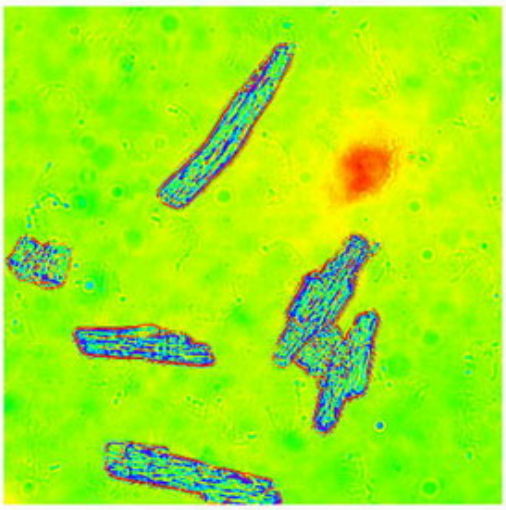
15.6.3. Rainbow パレットをグレー画像に適用する

この例では、グレー画像をレインボーカラーに変換する方法を示します。



Input Image

~



Output Image

```
pe_mkdir Conversion path:=aa$;
pe_cd aa$;

//行列を作成し、サンプル画像をインポート
window -t m;
path$ = System.path.program$;
fname$ = path$ + "samples\Image Processing and Analysis\myocyte8.tif";
impimage;
window -r %h Original;

window -d; //画像の複製
window -r %h newimage;

imgC2gray;       //グレーに変換

//パレット適用
fname$ = System.path.program$ + "palettes\Rainbow.PAL";
imgpalette palfile:=fname$;

window -s T;        //ウィンドウを水平に並べる
```

15.6.4. 画像をデータに変換する

画像を行列オブジェクトに変換するとき、それは **Image** データ型として維持され、ウィンドウの右上にアイコン **I** で示されます。**2D FFT** のようなある数学操作を行うには、データ型を **Data** 型に変換する必要があり、そうすると、ウィンドウの右上のアイコンは **D** で示されます。

このスクリプト例では、複数の画像を行列ブックにインポートし、それらを **data** 型に変換することを示します。

```
// ワイルドカードを使用してファイルを検索
string path$=system.path.program$+"Samples\Image Processing and Analysis";
findFiles ext:="*tif*";

// 新しい行列を作成し、すべての画像を新しいシートとしてインポート
newbook mat:=1;
impImage options.FirstMode:=0 options.Mode:=4;
// すべてのシートをループし、画像をバイトデータに変換
doc -e LW {
   img2m om:=<input> type:=1;
}
```

16 ユーザからの入力

自動化するのが困難なスクリプトに入力を促したい場合があります。例えば、グラフ上の特定のデータポイントを指定したり、ワークシートのあるセルをスクリプトから呼び出す関数への入力にするような場合があります。これを行うため、LabTalk は、スクリプト実行中にユーザに入力を促す方法をサポートしています。

一般に、連続したスクリプト行は、このようなユーザからの入力が必要な場所まで実行します。そして、スクリプトの実行が一時的に中断され、ユーザが何らかの情報を入力したら、処理を続けます。以下のセクションでは、このようなユーザの操作を待つようなプログラミングサンプルをいくつか示しています。

このセクションで説明している項目

- 数値と文字列の入力
- グラフからデータポイントを取得
- ダイアログを開く

16.1. 数値と文字列の入力を取得する

このセクションは、スクリプト実行中に 3 種類のユーザの入力を促すサンプルです。

1. はい/いいえの応答
2. 1 つの文字列
3. さまざまなタイプの入力 (GetN)

ユーザインターフェースモジュール(**UIM**)は、複雑なユーザインターフェースコントロールの構築を可能にします。**UIM オブジェクト** ページを確認してください。

16.1.1. はい/いいえの応答を得る

GetYesNo コマンド は、ユーザから「はい」または「いいえ」の応答を得るのに使うことができます。コマンドは 3 つの引数を取ります。

シンタックス:**getyesno** *stringMessageToUser numericVariableName windowTitle*

例えば、スクリプトウィンドウに次の行を入力すると、**Check Sign of X** というウィンドウタイトルで、ユーザに **Should X be positive?**という「はい」か「いいえ」「キャンセル」で答える質問のポップアップを開きます。**はい** を選択すると、**xpos** に値 1 が割り当てられます。**いいえ**を選択すると、**xpos** に値 0 が割り当てられます。**キャンセル**を選択すると、**xpos** にに値 0 が割り当てられ、**#Command Error!**が出力され、実行が停止します。

```
getyesno "Should X be positive?" xpos "Check Sign of X"
```

追加のスクリプト処理がイベントで必要となる場合、このコマンドを他の場所で呼び出し、数値をテストすることができます。次のサンプルは、**getyesno** を自分自身のコードから呼び出し、2 つの文字列入力が引数としてセクションに渡されます。(複数

のセミコロンの LabTalk スクリプトは、単にスクリプトウィンドウに貼り付けただけでは動作しません。ファイルに保存して実行します。)

```
[Main]
// 呼び出しコード
int iVal = -1;
run.section(,myGetYesNo,"Create a Graph of results?""Graphing Option");
if( iVal > 0 )
{
   type "Graph generated";         // はいの応答
}
else
{
   type "Graph NOT generated";     // いいえまたはキャンセルの応答
}

// 'myGetYesNo' セクション
[myGetYesNo]
getyesno (%1) iVal (%2);
```

16.1.2. 文字列を得る

GetString は、ユーザが入力する 1 つの文字列に対して使用します。

```
%B = "";
GetString (Enter as Last, First) Last (Your Name);
// キャンセルは GetYesNo のテクニックを使わずに中断
if("%B"!="Last")
{
    type User entered %B.;
}
else
{
    type User clicked OK, but did not modify text;
}
```

16.1.3. 複数の値を得る

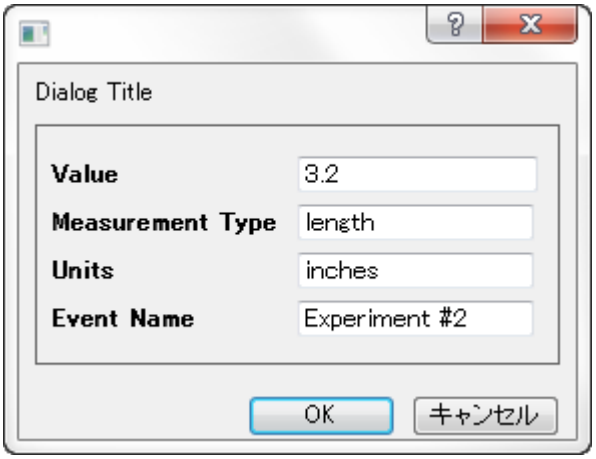
GetN または GetNumber ダイアログ は、数字、文字、リストで入力します。(Origin の以前のバージョンでは、数値のみが可能でした)Origin 8.1 以降では、**GetNumber** は、文字列変数(例、string str1$) と文字列レジスタ (例、%A) の両方を文字列の入力として受け付けます。以前のバージョンは文字列レジスタのみをサポートしていました。**GetN** は、現在、ダイアログタイトルに加えて 7 つまでの変数を受け付けます。

Origin8.1 の ***GetN*** の機能が増え、文字列変数はコマンドの呼び出しで使うことができます。この場合、文字列を最初に宣言しなければなりません。代入による方法ではなく、変数を作成する方法は常に良い方法となります。詳細は、(文字列)変数のスコープを参照して下さい。例えば:

```
// 最初に、使用する変数を宣言
double nn = 3.2;
string measurement$="length", units$="inches", event$="Experiment #2";

// GetN ダイアログを使ってユーザデータを収集
getn
(Value) nn
(Measurement Type) measurement$
```

```
(Units) units$
(Event Name) event$
(Dialog Title);
```

これは、次のダイアログを開き、ユーザに入力を促します。



このダイアログに入力された値は、宣言した変数に割り当てられます。変数は初期値(以前 **GetN** が呼び出されていた)を持つ場合、その値は入力ボックスに表示され、それ以外の場合、入力ボックスは空白となります。どちらの場合でも、初期値を変更したり、そのまま使うことができます。

入力されたデータをチェックするには、スクリプトの次の行を実行します。

```
// データを出力
type In %(event$), the %(measurement$) was $(nn) %(units$);
```

次のサンプルスクリプトは、Graph がアクティブウィンドウで、情報を促して、線とラベルを描画します。**GetN** の呼び出しは、入力として文字列レジスタと事前定義のリストを使用します。

```
%A=Minimum;
iColor = 15;
dVal = 2.75;
iStyle = 2;

// GetN ダイアログを開く
// %A の前の余計な %記号は、変数名として扱うのではなく
// 文字列レジスターを文字通りに解釈
getn (Label Quantile) %%A
(Color) iColor:@C
(Style) iStyle:@D
(Value) dVal
(Set Quantile);

draw -n %A -l -h dVal;      // 名前を付けた水平線を描画
%A.color = iColor;          // 線の色をセット
%A.linetype = iStyle;       // 線種をセット

// 線の右端に QLabel という名前のテキストラベルを
// 作成
label -s -a x2 dVal -n QLabel %A;

%A.Connect(QLabel,1);        // 2 つのオブジェクトを接続
```

Note:スクリプトは、%A が 1 つの語句で、オブジェクト *QLabel* が存在しないことを前提としています。

文字の順番に従い、@から始まる文字はあらかじめ定義されている **GetN** の引数です。

リスト	説明
@B	オブジェクトの背景属性のリスト
@C	基本の色のリスト
@D	線種のリスト
@P	パターンリスト
@S	フォントサイズのリスト
@T	フォントリスト
@W	線の太さのリスト
@Z	シンボルサイズのリスト

GetN ダイアログ内でリスト項目が選択されたときに返される値は、リスト内の項目のインデックスです。例えば、**GetN** のリストの 1 つが下記のようであれば:

```
(Font Size) fs:@S
```

ダイアログのドロップダウンリストから **18** を選ぶと、18 はリストの 8 番目の項目なので、変数 **fs** には値 **8** が入ります。

以下は、シンボルプロットを線＋シンボルに変更したり、その逆に変更するサンプルスクリプトです。

```
get %C -z iSymbolSize;  // 現在のシンボルサイズ
get %C -cl iLineColor;  // 現在の線の色
iUseLine = 0;
// ダイアログをユーザに開く
getn (Symbol Size) iSymbolSize
     (Use Line) iUseLine:2s
     (Line Color) iLineColor:@C
     (Set Plot Style);
// ユーザが線を求める場合
if(iUseLine == 1)
{
    set %C -l 1;            // 線を表示
    set %C -cl iLineColor; // 線の色をセット
}
// そうでなければ
else
    set %C -l 0;            // 線を非表示
set %C -z iSymbolSize;      // シンボルサイズをセット
```

16.2. グラフから点を取得

Origin の**プロット操作・オブジェクト作成**ツールバのどのボタンも、スクリプトから初期化でき、3 つはマクロとリンクして、プログラミングすることができます。

ツールをプログラミングするには、**pointproc** マクロを定義して、適切なコードを実行します。**pointproc** マクロは、ユーザがダブルクリックまたはシングルクリックして Enter キーを押すときに実行します。

16.2.1. スクリーンリーダ

このスクリプトは、スクリーンリーダツールを使って、グラフにラベルを配置します。

```
dotool 2;  // スクリーンリーダツールを開始
dotool -d; // シングルクリックをダブルクリックのように動作させる
// ここで'''pointproc''' マクロを定義
def pointproc {
    label -a x y -n MyLabel Hello;
    dotool 0;  // ツールをポインタにリセット
    done = 1;  // 変数を終わりまで無限ループを許可するようにセット
}
// ツールを使うときスクリプトは停止しない
// そのため、これ以上の実行を止める必要がある
// この無限ループはユーザが点を選択するのを待つ
for( done = 0 ; done == 0; ) sec -p .1;
// A .1 秒の遅延によりループに何かを与える
type Continuing script ...;
// マクロが実行されると無限ループは解放される
```

16.2.2. データリーダ

データリーダツールは、スクリーンリーダに似ていますが、カーソルは実際のデータポイントで固定されます。定義されると、ユーザが **Esc** キーを押すか、ポインタツールをクリックして、データリーダを停止する場合に **quittoolbox** マクロが実行されます。

この例は、グラフウィンドウがアクティブであることが前提で、そのグラフ上の 3 つのポイントを選択することを待ちます。

```
@global = 1;
dataset dsx, dsy;  // X と Y 値を保持する 2 つのデータセットを作成
dotool 3;          // ツールを開始
// 各点の選択に対して実行するマクロを定義
def pointproc {
    dsx[count] = x;  //  X 座標を取得
    dsy[count] = y;  //  Y 座標を取得
    count++;         // 増分をカウント
    if(count == 4) dotool 0;  // 3 つの点を持つかどうかを見ることをチェック
    else type -a Select next point;
}
// ユーザが Esc キーを押すかポインタツールをクリックすると
// 実行するマクロを定義:
def quittoolbox {
    // Error :Not enough points
    if(count < 4) ty -b You did not specify three datapoints;
```

```
    else
    {
        draw -l {dsx[1],dsy[1],dsx[2],dsy[2]};
        draw -l {dsx[2],dsy[2],dsx[3],dsy[3]};
        draw -l {dsx[3],dsy[3],dsx[1],dsy[1]};
        double ds12 = dsx[1]*dsy[2] - dsy[1]*dsx[2];
        double ds13 = dsy[1]*dsx[3] - dsx[1]*dsy[3];
        double ds23 = dsy[3]*dsx[2] - dsy[2]*dsx[3];
        area = abs(.5*(ds12 + ds13 + ds23));
        type -b Area is $(area);
    }
}
count = 1;  // 初期の点
type DoubleClick your first point (or SingleClick and press Enter);
```

次のサンプルは、Esc キー押されるか、ユーザがポインタツールをクリックまで任意のポイントを選択するものです。

```
@global = 1;
dataset dsx, dsy;  // X と Y 値を保持する 2 つのデータセットを作成
dotool 3;          // ツールの開始
// 各ポイントの選択を実行するマクロを定義
def pointproc {
    count++;          // 増分のカウント
    dsx[count] = x;   //  X 座標を取得
    dsy[count] = y;   //  Y 座標を取得
}

// ユーザが Esc キー押すか、ポインタツールをクリックすると
// 実行するマクロを定義
def quittoolbox {
   count=;
   for(int ii=1; ii<=count; ii++)
   {
      type $(ii), $(dsx[ii]), $(dsy[ii]);
   }
}
count = 0;  // 初期の点
type "Click to select point, then press Enter";
type "Press Esc or click on Pointer tool to stop";
```

Enter キーを押してポイントを選択することは、ダブルクリックしてポイントを選択するより確実です。

また、グラフからデータ値を収集する **getpts** コマンドを使うこともできます。

16.2.3. データセレクタ

データセレクタツールは、データセットの範囲をセットするのに使うことができます。範囲は開始行番号(インデックス)と終了行番号で定義されます。データセットの複数範囲を定義でき、Origin の分析ルーチンは、これらの範囲を入力に使い、これらの範囲外のデータは除外されます。

以下は、グラフの範囲を選択させるスクリプトです。

```
// ツールの開始
dotool 4;
// ユーザが行ったとき実行するマクロを定義
def pointproc {
    done = 1;
    dotool 0;
}
// ループで待ち押すことで完了
// (1) Enter キーまたは (2) ダブルクリック
for( done = 0 ; done == 0 ; )
{
    sec -p .1;
}
// ユーザがツールを終了させると、追加のスクリプトが実行
ty continuing ..;
```

領域データセレクタ または 領域マスクツール を使うとき、ユーザが Esc キーを押すことをきっかけとして、quittoolbox マクロを実行できます。

```
// グラフをアクティブにして、領域データ選択ツールを開始
dotool 17;
// ユーザが行ったとき実行するマクロを定義
def quittoolbox {
    done = 1;
}
// ループで待ち押すことで完了 ...
// (1) Esc キーまたは (2) ポインタツールをクリック
for( done = 0 ; done == 0 ; )
{
    sec -p .1;
}
// ユーザがツールを終了させると、追加のスクリプトが実行
ty continuing ..;
```

X ファンクションを使って、これらの範囲を見つけ、使用することができます。

```
// データセット内の範囲を取得
dataset dsB, dsE;
mks ob:=dsB oe:=dsE;
// For each range
for(idx = 1 ; idx <= dsB.GetSize() ; idx++ )
{
    // その範囲に対して曲線以下の積分を取得
    integ %C -b dsB[idx] -e dsE[idx];
    type Area of %C from $(dsB[idx]) to $(dsE[idx]) is $(integ.area);
}
```

プロット操作・オブジェクト作成ツールバーのリスト**太字**はプログラミングで役立ちます。

ツールの番号	説明
0	ポインタ - ポインタはマウスのデフォルトの状態で、マウスはセレクタとして動作します。
1	拡大 - グラフ上で矩形を選択し、その矩形の範囲の軸で再スケールします。(グラフのみ)
2	**スクリーンリーダ** - ページ上のポイントの位置を読み込みます。
3	**データリーダ** - ページ上のデータポイントの位置を読み込みます。(グラフのみ)
4	**データセレクタ** - データ範囲を示すデータマーカをセットします。(グラフのみ)
5	マウスで作図 - グラフ上にデータポイントを描画します。(グラフのみ)
6	テキスト - テキスト注釈をページに追加します。
7	矢印 - 矢印をページに追加します。
8	曲線矢印 - 曲線矢印をページに追加します。
9	直線 - 直線をページに追加します。
10	四角形 - 四角形をページに追加します。
11	円 - 円図形をページに追加します。
12	多角形 - 多角形の図形をページに追加します。
13	折れ線 - 折れ線をページに追加します。
14	自由閉曲線 - 閉曲線をページに追加します。
15	自由曲線 - 自由曲線をページに追加します。
16	縮小 - グラフ上のクリックした場所を中心に縮小します。(グラフのみ)
17	**領域データセレクタ** - データ範囲を選択します。(グラフのみ)
18	**領域マスクツール** - データ範囲のポイントをマスクします。(グラフのみ)

16.3. ダイアログを開く

16.3.1. X-Function ダイアログ

dlg で始まる名前を持つ X ファンクションは、ダイアログでの操作を手助けするためにスクリプトから呼び出すことができます。

X ファンクション名	説明
dlgChkList	一覧からの選択するダイアログ
dlgFile	ファイルを開くダイアログ
dlgPath	パスを開くダイアログ
dlgRowColGoto	指定した行と列に移動するダイアログ
dlgSave	名前を付けて保存ダイアログ
dlgTheme	ダイアログからテーマを選択ダイアログ

このような操作で最も一般的なものは、ディレクトリからファイルを選択することです。次のスクリプト行は、事前に選択している PDF ファイルの拡張子(group)のダイアログを開き、指定したパスの位置(init)で開始します。

```
dlgfile group:=*.pdf init:="C:\MyData\MyPdfFiles";
type %(fname$);  // 選択したファイルパスをスクリプトウィンドウに出力
```

ダイアログで選択したファイルの完全なファイル名(パスを含む)は、変数 **fname$**に保存されます。**init** が X ファンクション呼び出しの外側だったり、見つからない場合、ダイアログは User Files フォルダを開きます。

dlgsave X ファンクションはダイアログを使ってファイルを保存します。

```
dlgsave ext:=*.ogs;
type %(fname$);  // 保存したファイルパスをスクリプトウィンドウに出力
```

16.3.2. Origin C 関数

Type.remind()法は、ユーザーとの対話のためのダイアログを開く別の方法を提供します。この方法は、プライベートリマインダメッセージダイアログを作成します。Ini ファイルは、ダイアログを初期化するために使用されます。ini ファイルの各セクションは、単一のメッセージに使用されます。同じ機能を、グローバル関数を使用して Origin C で作成することができます。**PrivateReminderMessage**.

詳しくはこちらをご参照ください **type.remind()**.

17 Excel と一緒に操作する

Origin は、Origin ワークスペース内で直接 Excel ワークブックを使用できます。Excel ワークブックは、プロジェクト内に保存したり、外部 Excel ファイル(*.xls, *.xlsx)へのリンクを設定して保存することができます。Origin 内で開いた外部の Excel ワークブックを内部用ファイルに変換したり、Origin 内で作成した Excel ワークブックを外部の Excel ファイルに保存することができます。

Origin で新しい Excel ワークブックを作成するには、

```
window -tx;
```

タイトルバーには、**[内部]** と表示され、Excel ワークブックが Origin プロジェクトファイル内に保存されることを示します。

外部 Excel ファイルを開くには、

```
document -append D:\Test1.xls;
```

この場合、タイトルバーに Excel ファイルのパスとファイル名が表示され、Excel ワークブックが Origin プロジェクトファイル外に保存されていることを示します。

内部の Excel ワークブックを、外部ファイルとして保存してリンクすることができます。

```
// Excel ウィンドウがアクティブである必要があり、win -o で一時的にアクティブにできる
window -o Book5 {
    // ファイルパスと拡張子.xls が必要
    save -i D:\Test2.xls;
}
```

元になるファイルをディスクに残して置きながら、新しいファイルとリンクを作成して、外部 Excel ファイルを新しい場所に再保存することができます。

```
// Excel ワークブックがアクティブである場合
// %X は、開いている Origin プロジェクトファイルのパス
save -i %XNewBook.xls;
```

18 Origin で R を使う

18.1. Origin で R を使う

Origjn で R を実行するには 3 通りの方法があります。

- **R コンソール**
- **RServe コンソール**
- LabTalk オブジェクト **R** および **RS (RServe)**

R コンソールツールは、ダイアログインターフェースまたはコマンドを使用して Origin 環境内で R コマンドを投入でき、2 つのアプリケーション間でデータを移せます。

R および RS は、R コマンドを実行できる LabTalk オブジェクトで、変数とデータセットを LabTalk と Origin 間で受け渡します。

R オブジェクトと R コンソールを実行するには、コンピュータ上に R がインストールされている必要があり、RS オブジェクトと Rserve コンソールはクライアント側の Origin と通信できるサーバ側に R パッケージが必要です。

必要な Origin のバージョン：2016 SR0

内容

18.1.1. スクリプトウィンドウで R コマンドを実行

LabTalk オブジェクト **R** および **RS** を使用して R コマンドを実行し、LabTalk と R 間で変数の受け渡しが可能です。以下のサンプルは、R 及び RServe（R の代わりに RS を使用）の両方で有効です。

R または RS の初期化

LabTalk で R コマンドを実行する前に、以下を実行して R アプリケーションを初期化します。

```
if( R.Init()<0 )
        {
                type -b "Please install R software first.";
                return;
        }
```

スクリプトウィンドウで実行します。

RS オブジェクトを初期化するには、以下のスクリプトを参考にしてください。パラメータ設定についてはこのページを確認してください。

```
if(RS.Init(***.***.**.**, 12306, user, password)<0)  // Rserve PC IP,
e.g:192.168.18.75
        {
                type -b "Initialize failed.";
                return;
        }
```

R および RS コマンドを実行

R コマンドを実行

```
R.Exec(rand<-sample(x = 1:6, size = 50, replace = TRUE));
R.Exec(rand);
```

分析を実行します。

```
R.Exec(sum<-summary(rand));
R.Exec(sum);
```

サマリーを表示します。

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
1.00    2.00    3.50    3.56    5.00    6.00
```

複数の R コマンド行を実行

```
R.Exec("array<-1:25;dim(array)<-c(5,5);array") // セミコロンで区切った複数行の R コマンドを実行
```

RS コマンドを実行

```
string fname$ = system.path.program$ + "Samples\Curve Fitting\Sensor01.dat";
impasc;
range r1 = 1:2;

//ワークシートから R ベクトルにデータを送る
RS.Send(%(r1), RR, 2);  //範囲を R data.frame に送る

RS.Exec(fit<-lm(RR$Sensor.Output~RR$Displacement));  //線形フィットを実行
RS.Exec(fit);
```

R/RS コマンドのサンプルを実行

これらに加えて、**Perform Logistic regression in R by using LabTalk** のページに練習用サンプルがあります。

.R ファイルを実行

コンピュータ内の*.R ファイルを実行したい場合、スクリプトウィンドウで以下のシンタックスを使用します。

```
R.Exec(source(".R file path"));
```

例:

```
R.Exec(source("D:\\RData\\test.R"));
```

Origin のデータセットを R コンソールに送る

```
//ワークブックをアクティブにする
R.Send("1!1", arr1, 0); // 現在のワークブックの Sheet 1、列 1 を変数 ''array'' (データ型 vector)として R に送る

range rx=1; // 範囲表記 rx を使用してアクティブワークシートの列 1 を参照
R.Send(%(rx), rlabel, 0); // R 変数 rlabel (type vector)に列 1 を送る

// 最初の列の値:1, 2, 3, --
range rr = 1; // アクティブワークシートの第 1 列を参照
R.Send(%(rr), temp, 0); // R 変数 temp として最初の列を送る
```

Origin のデータセットとして R 変数を受け取る

```
R.Receive("1!1",RMat,1) // R 行列変数 RMat を現在の行列ブックの最初の行列オブジェクトに受け取る
R.Receive("1!1",df$vec1,0); // R データフレーム 'df' 内のデータメンバー 'vec1' をベクトルとして列に受け取る

// 値 [1] FALSE FALSE TRUE の論理的なベクトルがあると仮定
range rl=1;
R.Receive(%(rl),temp,0); // 代わりにアクティブシートの列 1 を 0,0,1 に
```

Labtalk と R 間の変数の受け渡し

R および **RS** オブジェクトは 4 つのメンバー関数をもちます。**GetReal**, **SetReal**, **GetStr**, **SetStr** はそれぞれ Labtalk と R 間で数値と文字列変数を受け渡します。たとえば、

数値または文字列の変数を R Serve コンソールで定義します。

```
Rvar=16
Rstr="height"
```

LabTalk 変数に値を渡します。

```
R.GetReal(LTVar, RVar)
R.GetStr(LTStr$, RStr)
//R リスト 'e1' の実数を LabTalk 変数に渡す
```

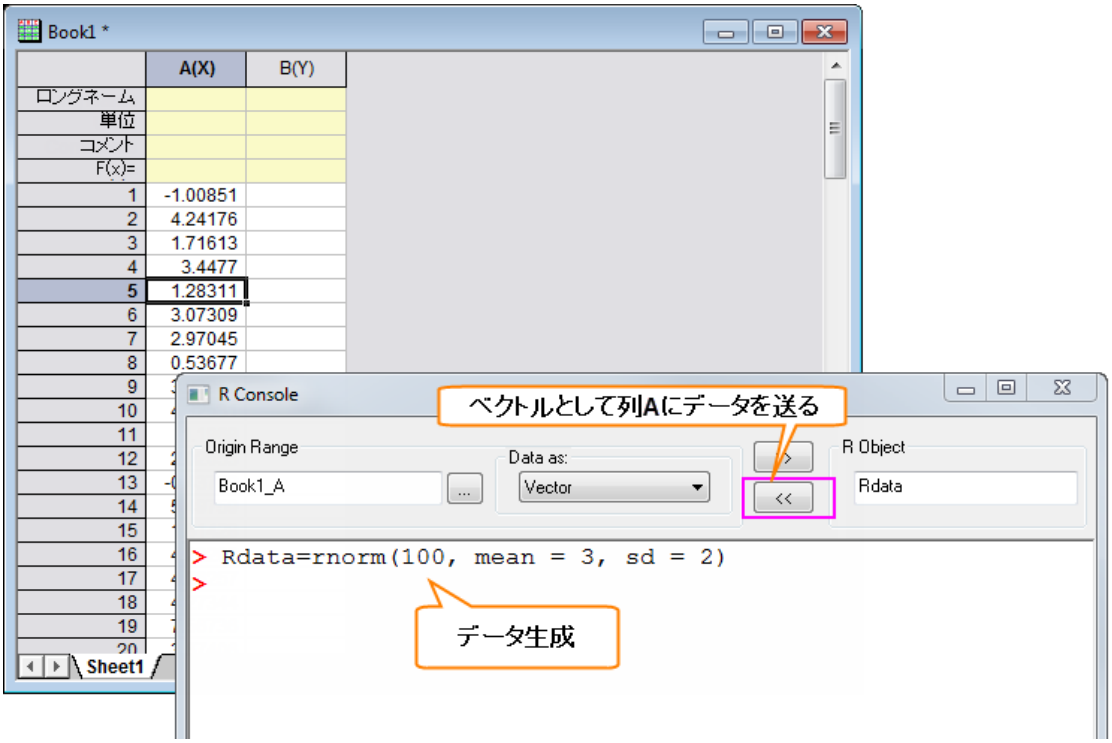
```
R.GetReal(LTVar,e1$a);
```

または、LabTalk で数値または文字列変数を定義し、R 変数に渡すこともできます。

```
R.SetReal(LTVar, RVar)
R.SetStr(LTStr$, RStr)
```

18.1.2. R コンソールまたは Rserve コンソールの利用

R コンソール（または Rserve コンソール）を使用してダイアログ入力ボックスで直接コマンドを実行し、**<<** および **>>**のボタンを使用して Origin のワークシートに結果の値を受け渡すことができます。R コンソール（Rserve コンソール）を使用して、R と Origin ワークシート（行列シート）間でデータを複数のフォーマット (vector, matrix, data.frame) に変換できます。
以下のグラフは、二項分布のランダムデータを生成し、Origin のワークシートに送る方法を説明しています。



18.2. サンプル:LabTalk を使用した R でのロジスティック回帰の実行

ロジスティック回帰モデルは、説明変数とカテゴリ応答変数間の関連です。以下のサンプルでは、スクリプトウィンドウで **R オブジェクト** および **RS オブジェクト**を使用して、*Origin2016\Samples\Statistics\LogRegData.dat* のデータにロジスティック回帰を実行します。

最初に Origin のワークシートにデータを読み込み、R データフレームとしてデータを送ります。R コマンド **glm** を使用してインポートされたデータにロジスティック回帰を実行し、最終的には、結果のために新しい Origin のワークシートを作成して、パラメータ、残差、aic、分散等を含む結果を和マリーシートに送ります。

R オブジェクト

```
// LabTalk で R オブジェクトを呼び出すサンプル
//run.section(testRex,LogisticRoex)

//サンプルデータをインポート
newbook;
string fn = system.path.program$ + "Samples\Statistics\LogRegData.dat";
impasc fname:=fn$ options.sparklines:=0;

//R がインストールされているかチェック
if( R.Init()<0 )
{
        type -b "Please install R software first.";
        return;
}

//Origin にインポートしたデータを R データフレーム変数 dfy として送る
//データフレーム dfy は次の 4 列を含む : Age, Salary, Gender and Career_Change
//Gender と Career_Change 列カテゴリデータ
R.Send("!", dfy, 2);

//インポートデータにロジスティック回帰を実行
//R のリスト変数 yr に結果を格納
R.Exec( "yf<-glm( Career_Change ~ Age + Salary + Gender, family=binomial(logit),
data=dfy)" );
R.Exec("yr<-summary(yf)");

//R の結果を出力するワークブックを作成
newbook;
page.longname$ = "Logistic Regression Result";

//R の係数行列を Origin のワークシートに送り
//ワークシートの最初の列から開始
R.Receive("1", yr$coefficients);

int nc = wks.ncols;
wks.ncols = nc + 6;

// パラメータの指数値を計算
wcol( nc+1 )[L]$ = "Exp of Parameter";
wcol( nc+1 ) = exp( wcol(2) );

//計算結果のためにラベルと列ロングネームをセット
wcol( nc+2 )[1]$ = "Residual";

wcol( nc+3 )[L]$ = "DF";
wcol( nc+4 )[L]$ = "Deviance";
wcol( nc+5 )[L]$ = "AIC";
wcol( nc+6 )[L]$ = "Dispersion Parameter for Binomial Family";

//Origin のワークシートに R の残差結果を送る
double dfr, devr, aic, rlev;
//LabTalk の double 型変数を R のリストオブジェクトの要素変数から取得
```

```
R.GetReal( dfr, yr$df.residual );
R.GetReal( devr, yr$deviance );
R.GetReal( aic, yr$aic );
R.GetReal( rlev, yr$dispersion );

//LabTalk の double 型変数でワークシート節をセット
wcol( nc+3 )[1] = dfr;
wcol( nc+4 )[1] = devr;

wcol( nc+5 )[1] = aic;
wcol( nc+6 )[1] = rlev;

//全ての R 変数をクリア
R.Reset();
```

RS オブジェクト

このサンプルを実行する前に、R Serve の設定が必要です。

```
//LabTalk で Rserve を呼び出すための RS サンプル
//サンプルデータのインポート
newbook;
string fn = system.path.program$ + "Samples\Statistics\LogRegData.dat";
impasc fname:=fn$ options.sparklines:=0;

//Rserve に接続 An Rserve should be set up first.
if( RS.Init( ***.***.***.***, 12306 )<0 ) //サーバ側の IP アドレスを入力
{
        type -b "Fail to connect R server.";
        return;
}

//Origin にインポートしたデータを R のデータフレーム変数 dfy に送る
//データフレーム dfy には次の列があります:Age, Salary, Gender and Career_Change
//Gender と Career_Change 列はカテゴリデータ
RS.Send("!", dfy, 2);

//インポートしたデータにロジスティック回帰を実行
//結果は R のリスト変数 yr に格納
RS.Exec( "yf<-glm( Career_Change ~ Age + Salary + Gender, family=binomial(logit),
data=dfy)" );
RS.Exec("yr<-summary(yf)");

//Rno 結果を出力するために新しいワークブックを作成
newbook;
page.longname$ = "Logistic Regression Result";

//リストオブジェクトの R の係数行列を Origin のワークシートに送る
//ワークシートの 1 列目から開始
RS.Receive("1", yr$coefficients);

int nc = wks.ncols;
wks.ncols = nc + 6;
```

```
//パラメータの指数値を計算
wcol( nc+1 )[L]$ = "Exp of Parameter";
wcol( nc+1 ) = exp( wcol(2) );

//計算された結果のラベルと列のロングネームをセット
wcol( nc+2 )[1]$ = "Residual";

wcol( nc+3 )[L]$ = "DF";
wcol( nc+4 )[L]$ = "Deviance";
wcol( nc+5 )[L]$ = "AIC";
wcol( nc+6 )[L]$ = "Dispersion Parameter for Binomial Family";

//Rno 残差情報を Origin のワークシートに送る
double dfr, devr, aic, rlev;
//LabTalk の double 型変数を R のリストオブジェクトの要素から取得
RS.GetReal( dfr, yr$df.residual );
RS.GetReal( devr, yr$deviance );
RS.GetReal( aic, yr$aic );
RS.GetReal( rlev, yr$dispersion );

//ワークシートセルを LabTalk の double 型変数でセット
wcol( nc+3 )[1] = dfr;
wcol( nc+4 )[1] = devr;

wcol( nc+5 )[1] = aic;
wcol( nc+6 )[1] = rlev;

//全ての R 変数をクリア
RS.Reset();
```

19 Python を使って操作する

Origin 内で Python を実行したり、Python から Origin にアクセスするための **PyOrigin** モジュールを使用するために、Origin は Python の環境を統合しています。

Origin に統合された Python は、バージョン 3.3.5 またはバージョン 2.7.8 です。デフォルトでは、Python のバージョンは 3.3 です。Python のバージョンを 2.7 に切り替えるには、スクリプトウィンドウを開き、メニューから**編集**:**スクリプトの実行**:**Python 2.7** を選択します。また、システム変数**@PYV** で切り替えることもできます。@PYV = 3 (デフォルト)のとき、統合された Python の環境は 3.3.5 で、@PYV = 2 のとき 2.7.8 です。システム変数の値を設定するには、**ツール**:**システム変数**を選択して**システム変数を設定**ダイアログを開きます。

以下のドキュメントは、Python 3.3.5 が Origin で使用されていることが前提の内容です。

必要な Origin のバージョン:2015 SR0

内容

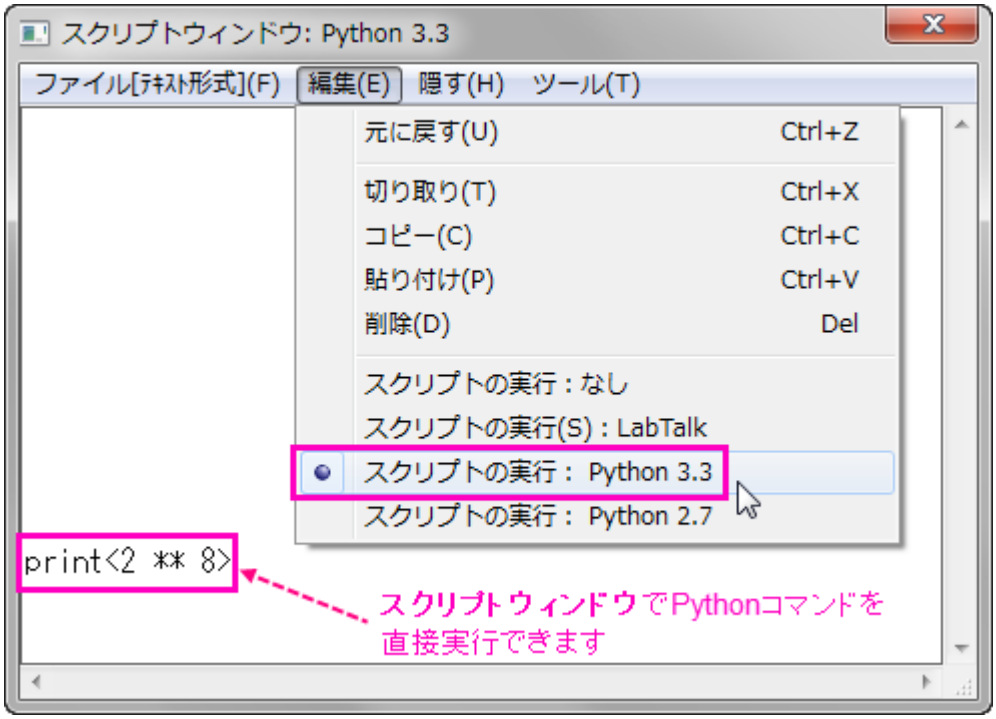
19.1.1. Origin で Python を実行

Origin では、Python コマンドラインや Python ファイル(拡張子.py)を実行できます。また、Origin 内で Python の式を評価することもできます。

Python コマンドラインを実行

Origin 内で、Python コマンドラインを実行するために、次の方法のどれかを使用します。

- **スクリプトウィンドウ**では、**編集:スクリプトの実行:Python 3.3** が選択されていることを確認し、Python コマンドを選択して Enter キー押すと直接コマンドが実行されます。



または

- LabTalk のコマンドオプションスイッチ **run -py** を使用します。たとえば、**編集:スクリプトの実行:LabTalk** が選択されていることを確認して、以下のように実行します。

```
// Python コマンドラインを LabTalk 文字列変数として定義
str$ = "a = 2 ** 8;print(a)";
// Python コマンドラインを実行
run -py %(str$);

// ** は Python におけるべき乗の演算子
//2 の 8 乗である 256 を返す
```

または

- LabTalk のコマンドオプションスイッチ **run.python()** を使用します。たとえば、**編集:スクリプトの実行:LabTalk** が選択されていることを確認して、以下のように実行します。

```
run.python(myname='Origin';print(myname));
//'Origin' を出力
```

複数の Python コマンドまたは、Python モジュールを**スクリプトウィンドウ**で実行するには、システム変数 **@PYI** を **0** にセットして**対話モード**をオフにします。**ツール**:**システム変数**を選択して、**システム変数を設定**ダイアログを開き、変数 = pti と値 = 0 を入力します。

Python ファイルを実行

Python ファイル(拡張子.py)を Origin で実行するために、以下の LabTalk コマンド/オブジェクトのどれかを使用します。

- LabTalk のコマンドオプションスイッチ **run -pyf** を使用します。たとえば、**編集:スクリプトの実行:LabTalk** が選択されていることを確認して、以下のように実行します。

```
// .py ファイルのパスと名前のための文字列を定義
string str$ = system.PATH.PROGRAM$ + "\Samples\Python\ListMember.py";
// .py ファイルを実行
run -pyf "%(str$)";

// この.py ファイルは PyOrigin モジュールで提供されるすべての関数をリスト表示
```

または

- コマンドオプションスイッチ **run -pyp** を使用して、Origin のプロジェクトファイルに追加した Python ファイルを実行します。

または

- LabTalk のコマンドオプションスイッチ **run.python()** を使用します。たとえば、**編集:スクリプトの実行:LabTalk** が選択されていることを確認して、以下のように実行します。

```
// ファイルパスのための文字列を定義
string file_path$ = system.PATH.PROGRAM$ + "\Samples\Python\";
// Python ファイルのパスをセット
run.python("%(file_path$)", 32);
// 選択されたパスにある ListMember.py ファイルを実行
run.python("ListMember.py", 2);

// この.py ファイルは PyOrigin モジュールで提供される全ての関数を表示
```

Python の 式を評価

次の LabTalk コマンド/オブジェクトメソッドのどれかを使用して、Python の式を評価できます。

- LabTalk のコマンドオプションスイッチ **run -pye** を使用します。たとえば、**編集:スクリプトの実行:LabTalk** が選択されていることを確認して、以下のスクリプトを実行します。

```
run -pye 10 % 4;

// Python の式 10 % 4 （% は Python で剰余演算子）を評価
// 評価された値 2 が出力（10 を 4 で割った値）
```

または

- LabTalk のコマンドオプションスイッチ **run.python()** を使用します。たとえば、**編集:スクリプトの実行:LabTalk** が選択されていることを確認して、以下のように実行します。

```
run.python("10 % 4", 1);

// 上の例と同様、評価された値 2 が返される
```

19.1.2. Python で Origin にアクセスする

Python コード内で、**PyOrigin** モジュールをインポートし、Origin オブジェクトにアクセスするために提供された関数を使用できるようにします。

PyOrigin モジュールをインポート

PyOrigin モジュールをインポートするために、.py ファイル内に以下のコマンドラインを入力します。

```
import PyOrigin
```

PyOrigin モジュールは、現 Origin セッションと Python の式の対話のために使用されることが想定されています。そのため、**PyOrigin** モジュールがインポートされた.py ファイルのみ、Origin から呼び出すことができます（前述の **Python ファイルを実行**セクションを参照）。このような.py ファイルは、外部の Python エディタから直接実行することはできません。(e.g. IDLE)

PyOrigin で提供された関数を調べる

全ての関数/グローバル変数とクラス/統合された Python からアクセス可能な関数の一覧を調べるには、**スクリプトウィンドウ**で以下の LabTalk スクリプトを実行します。

```
//スクリプトの実行:LabTalk が選択されていることを確認
string str$ = system.PATH.PROGRAM$ + "\Samples\Python\ListMember.py";
run -pyf "%(str$)";
```

PyOrigin モジュールで提供される関数/クラス/グローバル変数は、Origin におけるものととてもよく似ています。そのため、使用についての詳細は Origin C リファレンスを確認してください。(メインメニューで**ヘルプ:プログラミング:Origin C** を選択し、**Origin C Reference** の章を開きます。)

Origin で Python を使うサンプル

Origin のサンプルフォルダ（*<Origin EXE フォルダ>\Samples\Python*）に、いくつかの Python の組み込みサンプルがあり、Origin で Python を使用して操作する方法確認できます。これには、ワークシートにデータを送る方法、データとグラフをインポート、グラフをエクスポートして web ページでグラフを公開する方法などがあります。サンプル*.py ファイルは、LabTalk コマンドで直接実行でき、Python ファイルが追加された Origin のサンプルファイルはカスタマイズされたボタンをクリックすることで実行できます。

Python ファイルを実行し Python から Origin にデータを送る

<Origin EXE フォルダ>\Samples\Python にある **SendDataToWorksheet.py** は、Python から Origin にデータを送るサンプルで、Python の **tkinter** モジュールを使用して、列フォーマットや行の数といった入力を行うダイアログを作成します。そして、入力されたパラメータを Origin に送信します。

この.py ファイルを実行するために、次の LanTalk スクリプトを**スクリプトウィンドウ**で実行します。

```
// スクリプトの実行が LabTalk になっていることを確認
//Origin でワークシートがアクティブになっていることを確認
string str$ = system.PATH.PROGRAM$ + "\Samples\Python\SendDataToWorksheet.py";
run -pyf "%(str$)";
```

Origin プロジェクトに追加した Python ファイルの実行と表示

<Origin EXE フォルダ>\Samples\Python にある、**ImportMatrixPlotContour.opj** は、Origin に行列をインポートし、等高線図として作図する方法を示しています。また、このサンプルには Python を Origin に追加し、実行する方法も示しています。

このサンプルを使用するには、Origin のメニューから**ファイル**:**開く**を選択し、プロジェクトファイルを開きます。開くと、**Control Panel** というワークブックがあります。**Import Matrix and Plot Contour** ボタンをクリックして付属の Python ファイルを実行します。これは、**スクリプトウィンドウ**で以下の LabTalk スクリプトを実行するのと同等です。

```
run -pyp ImportMatrixPlotContour.py;
```

Show Python Code ボタンをクリックすると、Origin の**コードビルダ**でプロジェクトに付属している Python ファイルを表示できます。これは、次の LabTalk スクリプトと同等です。

```
%z = system.path.projectAttachedFilesPath$;
ed.open(%zImportMatrixPlotContour.py);
```

また、**コードビルダ**を開き(**表示**:**コードビルダ**を選択するか、**Alt+4** を押す)、ワークスペースパネルで **Project** フォルダを開き、付属の Python ファイルにアクセスできます。

Python ファイルを Origin のプロジェクトに追加するには、**Project** フォルダを右クリックして、**ファイルを追加**を選択し、Python ファイルを含めます。

19.1.3. Python の式を使用する再の注意

Origin に統合された Python は、すべての基本的なモジュールと、あなたの Python アプリケーションにインストールされているモジュールを含みます。PyQt4, numpy, scipy などの他の Python モジュールを使用する場合、以下のようにします。

1. ダウンロードした Python モジュールのバージョンが、Origin の統合 Python のバージョンと互換のあるバージョン 3.3 またはバージョン 2.7 か確認します。
2. ダウンロードした Python モジュールが、Origin が操作している、32bit または 64bit と同じ CPU レジスタサイズに関連付いていることを確認してください。現在の Origin のバージョンが 32bit か 64bit かを確認するには、Origin のプログラムウィンドウのタイトルバーを確認するか、コマンドウィンドウまたはスクリプトウィンドウで" system.bits=;" を実行します。
3. ダウンロードされた Python モジュールのパスが、Python の **import** コマンドで直接使用できるように適切に追加されているかを確認します。
4. 現在の Pyhon の実行バージョンが、ダウンロードされた Pythin モジュールと互換性があるかを確認します。デフォルトでは、Python のバージョンは 3.3 です。Python のバージョンを 2.7 に切り替えるには、スクリプトウィンドウを開き、メニューから**編集**:**スクリプトの実行**:**Python 2.7** を選択します。

*C:\Program Files (x86)\Python\Lib\site-packages*のような、デフォルトパスに Python モジュールがインストールされた場合、以下のサンプルのようにして直接パッケージを使用できます。

```
# Python 実行バージョン 3.3 (デフォルト)

import numpy

x = numpy.array([[1, 2, 3], [4, 5, 6]], numpy.int32)

print(x)
```

あるいは、インポートの前にインストレーションフォルダをシステムパスに追加する必要があります。インストレーションフォルダをシステムパスリストに追加するには、Origin に統合された Python 環境に以下の Python ステートメントを実行するのがお勧めのアプローチです(直接ステートメントを実行するか、ファイルから実行します)。

```
import sys

# 文字列変数 py_ext_path の値を
# 実際の Python 拡張インストレーションパスと交換

py_ext_path = "C:\Python33\Lib\site-packages"

if py_ext_path not in sys.path:
    sys.path.append(py_ext_path)
```

これらの Python ステートメントを実行したあと、次のサンプルを使用して、現在の Origin セッションで Python の拡張モジュールにアクセスすることができます。

```
import PyQt4
```

しかし、現在の Origin セッションを閉じた場合、再度 sys.path リストに Python の拡張パスを追加する必要があります。

サンプル使用として、Numpy および Scipy のリンクで目的のバージョンフォルダをクリックして互換パッケージを探し、ダウンロードできます。

19.2. Python オブジェクトのサンプル

必要な Origin のバージョン: Origin 2017 SR1 以降

LabTalk の Python オブジェクトを使用してロジスティック回帰を行う方法を以下の例で示します。

> **Note:** このサンプルでは Python 拡張を使用します。この例を理解しやすくするために、最初に Python 拡張を使用する方法に関する注意 を読むことを強くお勧めします。

Python オブジェクト

最初に Origin のワークシートにデータを読み込み、次に Python リストオブジェクトとしてデータを送ります。それから Python でロジスティック回帰を実行し、最後にリストオブジェクトにある Python のフィッティングされた係数を Origin のワークシートに送ります。

```
//Python object example to call Python in LabTalk
//Check whether Python is installed
if( Python.Init()<0 )
{
        type -b "Initialize failed.";
        return;
}
//Add the installation folder of Python extension to system path list
Python.Exec("import sys");
//Optionally, modify the path of installation folder
Python.Exec("PkPath = "C:\Python33\Lib\site-packages"");
Python.Exec("sys.path.append(PkPath)");
//Import sample data
newbook;
string fn = system.path.program$ + "Samples\Statistics\LogRegData.dat";
impasc fname:=fn$ options.sparklines:=0;
//Send data in worksheet to Python
```

```
for (ii = 1; ii <= wks.ncols; ii++)
{
        range rr = 1!$(ii);
        Python.Send(rr, var$(ii));
}
//Run logistic regression in Python
//Results are stored in Python's list object result
Python.Exec("import pandas as pd");
Python.Exec("import numpy as np");
Python.Exec("import statsmodels.api as sm");
Python.Exec("df = pd.DataFrame.from_items([('Age', var1), ('Salary',
var2),('Gender',var3),('Career_Change',var4)])");
Python.Exec("df['Gender'] = df['Gender'].astype('category')");
Python.Exec("df['Career_Change'] = df['Career_Change'].astype('category')");
Python.Exec("cat_columns = df.select_dtypes(['category']).columns");
Python.Exec("df[cat_columns] = df[cat_columns].apply(lambda x: x.cat.codes)");
Python.Exec("df['intercept'] = 1.0");
Python.Exec("logit = sm.Logit(df['Career_Change'],
df[['Age','Salary','Gender','intercept']])");
Python.Exec("result = logit.fit()");
//New a workbook to output Python's result
newbook;
page.longname$ = "Logistic Regression Result";
//Send Python's fitted coefficients in list object to Origin's worksheet
wks.ncols=4;
//Send Python's fitted coefficients in list object to Origin's worksheet
Python.Receive("1", list(result.params.index));
wks.col2.SetFormat(1);
wks.col2.lname$ = Fitted parameter;
Python.Receive("2", list(result.params));
wks.col3.SetFormat(1);
wks.col3.lname$ = 95% CI lower;
Python.Receive("3", list(result.conf_int()[1]));
wks.col4.SetFormat(1);
wks.col4.lname$ = 95% CI upper;
Python.Receive("4", list(result.conf_int()[1]));
```

LabTalk を使用した R によるロジスティック回帰を実行する方法をこちらのページで学ぶことができます。

20 自動化とバッチ処理

ここでは、いくつか分析テンプレートを作成して、これらのテンプレートでデータのバッチ処理を実行することで、Origin で自動的に分析を行う LabTalk スクリプトを説明します。

このセクションで説明している項目

- 分析テンプレート
- 値の設定機能を使って分析テンプレートを作成する
- バッチ処理

20.1. 分析テンプレート

分析テンプレートは、分析操作で使用するデータシート、レポートシートおよび結果表示のためのカスタムレポートシートなどの複数シートを含めることができる事前設定されたワークブックです。分析操作は、データ変更での再計算をセットすることができ、複数データファイルのバッチ処理や手動での処理に対する分析テンプレートの繰り返し利用を行うことができます。

次のスクリプトサンプルは、組み込みの分析テンプレート、*Dose Response Analysis.ogw* を開き、データファイルをデータシートにインポートします。結果は自動的に新しいデータを元にして更新されます。

```
string fPath$ = system.path.program$ + "Samples\Curve Fitting\";
string fname$ = fPath$ + "Dose Response Analysis.ogw";
// 現在のプロジェクトに分析テンプレートを開く/追加
doc -a %(fname$);
string bn$ = %H;
win -o bn$ {
        // No Inhibitor データをインポート
        fname$ = fPath$ + "Dose Response - No Inhibitor.dat";
        impASC options.Names.FNameToSht:=0
                options.Names.FNameToBk:=0
                options.Names.FNameToBkComm:=0
                orng:=[bn$]"Dose Response - No Inhibitor";
        // Inhibitor データをインポート
        fname$ = fPath$ + "Dose Response - Inhibitor.dat";
        impASC options.Names.FNameToSht:=0
                options.Names.FNameToBk:=0
                options.Names.FNameToBkComm:=0
                orng:=[bn$]"Dose Response - Inhibitor";
        // 結果ワークシートをアクティブにする
        page.active$ = result;
}
```

分析テンプレートを作成する方法を学ぶには、Origin チュートリアルを参照してください。分析テンプレートの作成と利用

20.2. 値の設定を使って、分析テンプレートを作成する

Origin で提供される多くの分析ツールは、再計算オプションを提供しており、新しいデータをインポートして既存のデータと置き換えたときなど元データが変わった時に結果を更新することができます。このような操作を含むワークブックは**バッチ処理**で繰り返し使用するために**分析テンプレート**として保存できます。

列値の設定の機能は、カスタムスクリプトが必要な分析の場合に、このような分析テンプレートを作成するのに使用することもできます。

列値の設定の機能を使って分析テンプレートを作成するために以下のステップをお勧めします。

1. 代表的なデータファイルをインポートするなどして、データシートをセットアップします。
2. データシートまたは同じワークブックの新しいシートに列を追加します。
3. 新しく追加した列から**値の設定**ダイアログを開きます。
4. **実行前の処理スクリプト**パネルに目的の分析スクリプトを入力します。スクリプトは X ファンクションを呼び、データに複数の操作を実行できます。
5. スクリプトでは、新しいデータで置き換えるデータシートの列またはセルを少なくとも 1 つ参照するようにします。データ列を指す範囲変数を定義し、スクリプト内でその範囲変数を使って、カスタム分析出力を計算することでこれを行うことができます。
6. ダイアログボックスの**再計算**ドロップダウンリストで *手動* または *自動*のどちらかをセットし、OK を押します。
7. **ファイル**:**分析テンプレートとしてワークブックを保存...** メニューを選び、分析テンプレートを保存します。

スクリプトを使ったこのようなテンプレートをセットアップするサンプルについては、Origin のチュートリアル、列値の設定を使って分析テンプレートを作成する、を参照してください。列値の設定を使って分析テンプレートを作成する

20.3. バッチ処理

Origin であるデータに対して行った分析処理と同じ処理を複数のデータファイルやデータセットに対してバッチ処理として実行する必要があるかもしれません。これは 3 つの方法で行うことができ、次のセクションで情報とサンプルを提供しています。

20.3.1. 各データセットをループで処理する

バッチ処理を行う 1 つの方法は、複数ファイルやデータセットをループし、ループ処理の中で、適切な X ファンクションや他のスクリプトを呼び出し、必要なデータ処理を実行することです。

次のサンプルは 10 個のファイルをインポートし、カーブフィット操作を実行して、フィット結果を出力します。

```
// ワイルドカードを使ってすべてのファイルを検索
string path$ = system.path.program$ + "Samples\Batch Processing";  // ファイルを検索するパス
// path$変数で指定したフォルダ内を検索
// 結果のファイル名を文字列 fname$に保存する
// CRLF で分ける(デフォルト).ここで使用しているワイルドカード * は、
// "T"からはじまり、拡張子が"csv"のファイルすべてを意味する
findFiles ext:="T*.csv";

// シートなしの新しいブックを開始
newbook sheet:=0;
```

```
// すべてのファイルをループ
for(int iFile = 1; iFile <= fname.GetNumTokens(CRLF); iFile++)
{
        // ファイル名を取得
        string file$ = fname.GetToken(iFile, CRLF)$;
        // ファイルを新しいシートにインポート
        newsheet;
        impasc file$;
        // 現在のデータの列 2 にガウスフィットを実行
        nlbegin iy:=2 func:=gaussamp nltree:=myfitresult;
        // フィットし、レポートなしで終了
        nlfit;
        nlend;
        // ファイル名と結果を出力
        type "File Name:%(file$)";
        type "   Peak Center= $(myfitresult.xc)";
        type "   Peak Height= $(myfitresult.A)";
        type "   Peak Width=  $(myfitresult.w)";
}
```

20.3.2. ループ内で分析テンプレートを使用

代表的なデータセットに対してメニュー操作で必要なデータ処理を実行し、ワークブックまたはプロジェクト全体を**分析テンプレート**として保存し、Origin で分析用のカスタムテンプレートを作成できます。次のサンプルは、10 個のファイルのカーブフィットを実行する既存の分析テンプレートを使う方法です。

```
// ワイルドカードを使ってすべてのファイルを検索
string fpath$ = "Samples\Batch Processing\";
string path$ = system.path.program$ + fpath$;  // ファイルを検索するパス
// path$変数で指定したフォルダ内を検索
// 結果のファイル名を文字列 fname$に保存する
// CRLF で分ける(デフォルト)ここで使用しているワイルドカード * は、
// "T"からはじまり、拡張子が"csv"のファイルすべてを意味します
findFiles ext:="T*.csv";

// 分析テンプレートのパスをセット
string templ$ = path$ + "Peak Analysis.OGW";
// すべてのファイルでループ
for(int iFile = 1; iFile <= fname.GetNumTokens(CRLF); iFile++)
{
   // 分析テンプレートのインスタンスを開く
   doc -a %(templ$);
   // 現在のファイルを第 1 シートにインポート
   page.active = 1;
   impasc fname.GetToken(iFile, CRLF)$
}

// テンプレートで手動再計算に設定した場合
//保留中のすべての操作を更新する
run -p au;
```

20.3.3. X ファンクションのバッチ処理を使用する

Origin は、ファイルやデータセットをループする必要がないところでスクリプトアクセス可能な X ファンクションでバッチ処理を行うことができます。単に処理を行うデータのリストを作成し、適切な X ファンクションを呼び出します。X ファンクションは、テンプレートまたはテーマを使って、指定したデータのすべてを処理します。これらの X ファンクションのいくつかは、任意のサマリーレポートを作成します。サマリーレポートには、ユーザが分析テンプレートでレポートすると指定した各ファイル/データセットからの結果が含まれています。

以下の表は、バッチ分析で利用可能な X ファンクションの一覧です。

名前	簡単な説明
batchProcess	任意のサマリーレポート付きで、分析テンプレート使って複数ファイルまたはデータセットのバッチ処理を実行します。
paMultiY	ピークアナライザテーマを使って複数 Y データセットのピーク分析を実行します。

次のスクリプトは、分析テンプレートを使って、10 個のファイルからデータのカーブフィットを実行し、処理の終わりにサマリーレポートを作成する batchProcess X ファンクション を使用する方法を示しています。

```
// ワイルドカードを使ってすべてのファイルを検索
string path$ = system.path.program$ + "Samples\Batch Processing\";  // ファイルを検索するパス
// path$変数で指定したフォルダ内を検索
// 結果のファイル名を文字列 fname$に保存する
// CRLF で分ける(デフォルト).ここで使用しているワイルドカード * は、
// "T"からはじまり、拡張子が"csv"のファイルすべてを意味する
findFiles ext:="T*.csv";

// 分析テンプレートのパスをセット
string templ$ = path$ + "Peak Analysis.OGW";

// バッチ処理 X-Function を呼び出す
// 最終的なサマリーシートを保持し、中間ブックを削除
batchProcess batch:=1 name:=templ$ data:=0 fill:="Raw Data"
             append:="Summary" remove:=1 method:=impASC;
```

X ファンクションを使ったバッチ処理は、外部コンソールから Origin を呼び出すことで実行できます。詳細は、コンソールからスクリプトを実行するをご覧ください。

21 関数リファレンス

このセクションでは、LabTalk スクリプトでサポートされる関数、X ファンクション、Origin C のリストを表示します。

このセクションで説明している項目

- LabTalk でサポートしている関数
- LabTalk でサポートしている X ファンクション

21.1. LabTalk がサポートする関数

内容

21.1.1. 文字列関数

Note:次のすべての関数は、Origin 8 SR6 以降のバージョンでのみ利用できます。

名前	説明
Char(number)$	1-255 の整数をとり、ASCII 文字を返します。 **サンプル**: • `char(65)$` A.を返します。 • `char(col(B))$` char(col(B))$は列 B にある整数値に対応する ASCII 文字を返します。
Code(str$)	文字列をとり、はじめの文字に対する ASCII コードを返します。**サンプル**: • `str$ = "abc"; code(str$)` 97.を返します。 • `code(col(D))` code(col(D)) は列 D にある文字列の最初の文字に対応する ASCII コードの整数値を返します。
Compare(str1$, str2$ [,Case])	2 つの文字列をとり、一致した場合 1、不一致の場合 0 を返します。 オプション **Case** では、ケースが、1 のとき一致 (デフォルト)、0 のとき不一致となります。 **サンプル**: • `str1$ = "ABC"; str2$ = "abc"; compare(str1$,str2$,0)` 1.を返します。 • `compare(col(F), col(G),1)` compare(col(F), col(G),1)は、文字列または Case が合致しないとき 0 を返し、文字列と Case が合致しない場合 1 を返します。
Exact(str1$, str2$)	2 つの文字列をとり、大文字と小文字の区別を含めて一致した場合 1、不一致の場合 0 を返します。**サンプル**: • `str1$ = "ABC"; str2$ = "abc"; exact(str1$,str2$)` 0.を返します。 • `exact(col(F), col(G))` exact(col(F), col(G)) は完全一致しない場合は 0 を返し、完全一致のン場合 1 を返します(Case を含む)。
Find(str1$, str2$ [,StartPos])	**str1$** で **str2$**を検索し、見つかった場合 **str1$**の最初の文字の位置を返し、見つからなかった場合 0 を返します。 オプション **StartPos** は、検索開始位置を制御します (デフォルト = 1)。大文字小文字の区別はあり、ワイルドカードは無効です。 **サンプル**: • `str1$ = "abcde"; str2$ = "bc"; find(str1$,str2$)` 2.を返します。

	• `find(col(G),col(J))` find(col(G),col(J)) 文字列内で col(J) 文字列を最初の文字から検索します。見つかった場合、col(G)内の col(J)文字列の位置を返します。
IsEmpty(str$)	Excel の ISBLANK 関数に似ています。ワークシートのセルが空であるかどうかを判別するために使用されます。引数 **str** には、セルのアドレスまたは値の列を指定できます。**サンプル**: • `isempty(col(A)[2]$)=; // return 0 if cell row 2, col 1 contains a value; or 1 if empty.`
IsFormula(str$)	ワークシートセルがセル関数入力されているか確認します。**サンプル**: • `isformula(A2)=; // return 1 if cell row 2, col 1 contains cell formula; otherwise return 0.`
Left(str$, n)$	文字列 **str$** をとり、左から数えて **n** 文字までを返します。 **サンプル**: • `str$ = "abcde"; Left(str$,3)$ abc.`を返します。 • `left(col(G),3)$` left(col(G),3)$ は col(G)文字列の左側の 3 文字を返します。
Len(str$)	文字列 **str$** をとり、文字数を返します。**サンプル**: • `str$ = "abc ABC"; Len(str$) 7.`を返します。 • `len(col(G))` len(col(G)) は col(G)文字列の左側の 3 文字を返します。
Lower(str$)$	文字列 **str$** をとり、小文字に変換します。**サンプル**: • `str1$ = "ABCDE"; str2$ = Lower(str1$)$ abcde.`を返します。 • `lower(col(F))$` lower(col(F))$は、文字列 col(F)の小文字を返します。
MakeCSV(str1$[, quote, output_delim, input_delim$])$	区切り文字を持つ文字列をとり、CSV に変換します。 **quote** は、0 (デフォルト) = 囲み文字なし, 1 = 一重引用符, 2 = 二重引用符 を指定します。**output_delim** は、0 = コンマ, 1 = セミコロン です。オプション **input_delim$** は、ソース文字列の区切り文字を指定します。スペース区切りの場合は必要ありません。**サンプル**: • `str1$ = "This is a test value"; MakeCSV(str1$, 1, 0)$ &apos;This&apos;,&apos;is&apos;,&apos;a&apos;,&apos;test&apos;,&apos;value&apos;.`を返します。 • `makecsv(col(N),0,0)$` makecsv(col(N),0,0)$のスペース区切りの文字をとり、カンマ区切りの値を返します。

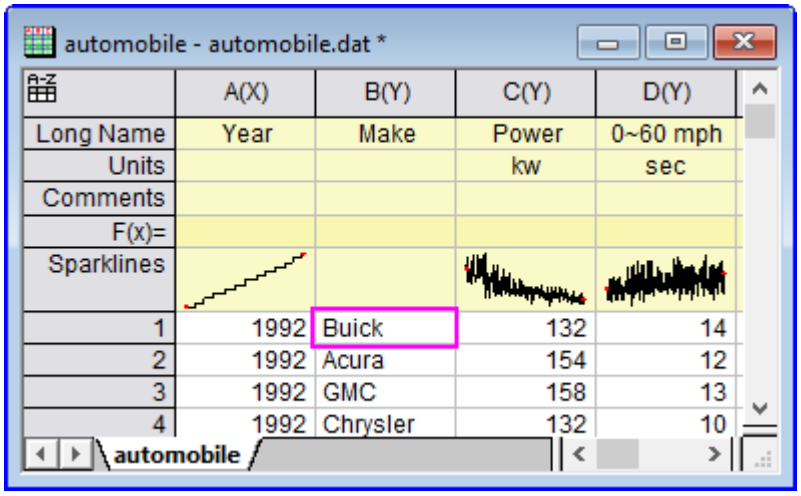
Match(within$,find$[,Case]) (2015 SR0)	この関数は文字列 **find$**と **within$** を比較し、内容が互いに合致するかを確認します。*1* (真, 一致) または *0* (偽, 不一致)をかえします。文字列 **find$**には、"*" や "?" といったワイルドカード文字の使用をサポートしています。**Case** で大文字と小文字の区別を制御し、0(デフォルトのとき不一致、1 のとき一致になります。**サンプル**: • `str1$ = "From: test@Originlab.com"; str2$ = "F*com"; Match(str1$,str2$); 1.`を返します。
MatchBegin(within$,find$[,StartPos,Case])	文字列 **within$** を探し、**find$** の開始位置に応じた整数を返します。見つからない場合は-*1* を返します。 ワイルドカード "*"' と "?" が有効です。オプション **StartPos** は、検索をはじめる文字の位置を指定し、1 (デフォルト)のとき、1 番目の文字から検索します。オプション **Case** では、ケースが、0 のとき不一致 (デフォルト)、1 のとき一致となります。**サンプル**: • `str1$ = "From: test@Originlab.com"; str2$ = "From*@"; MatchBegin(str1$,str2$,1); 1.`を返します。 • `matchbegin(col(a)," ")` matchbegin(col(a)," ") は col(a)の最初のスペース位置を返し、ない場合-1 をかえします。
MatchEnd(within$, find$[, StartPos, Case])	文字列 **within$** を探し、**find$** の終了位置に応じた整数を返します。見つからない場合は-*1* を返します。 ワイルドカード "*" と "?" が有効です。オプション **StartPos** は、検索をはじめる文字の位置を指定し、1 (デフォルト)のとき、1 番目の文字から検索します。オプション **Case** では、ケースが、0 のとき不一致 (デフォルト)、1 のとき一致となります。**サンプル**: • `str1$ = "From: test@Originlab.com"; str2$ = "From*@"; MatchEnd(str1$,str2$,1); 11.`を返します。 • `MatchEnd(col(A),col(B))` MatchEnd(col(A),col(B)) は col(a)内の col(b)文字列の最後の位置を返し、一致しない場合は-1 を返します。
Mid(str$, StartPos [, n])$	文字列 **str$**をとり、**StartPos** から **n** 文字返すまたは、**n** が指定されない場合 **StartPos** から全てを返します。 **サンプル**: • `str$ = "aabcdef"; Mid(str$,2,3)$ abc.`を返します。 • `str$ = "aabcdef"; Mid(str$,2)$ abcdef.`を返します。 • `mid(col(a),1,3)$` mid(col(a),1,3)$ は、col(a)文字列の最初の 3 文字を返します。
NumberValue(str$ [, Decimal$, Group$])	文字列または文字列のベクトルを受け取り、数値として返します。オプションの **Decimal**は文字列の小数点記号を解釈するために使用されます。オプション **Group** はセパレータの解釈に使用されるます。文字列とオプションは引用符で囲みます。 **サンプル**: • `numbervalue("1,000.05")=; // returns 1000.05 (US regional settings)`

	• `numbervalue("5.000,0", ",", ".")=; // returns 5000 (US regional settings)`
Replace(str1$, StartPos, n, replace$)$	**str1$** の **StartPos** 番目の文字から **n** 文字を文字列 **replace$** に取り換えます。 文字列 **replace$** の長さと **n** の長さは異なっていても問題ありません。**サンプル**: • `Replace(abcdefghijklmn,3,5,123456)$` ab123456hijklmn.を返します。 • `replace(col(a),1,4,"Replacement string ")$` replace(col(a),1,4,"Replacement string ")$ は文字列 col(a)の最初の 4 文字を、"Replacement string " (スペースを含む)と置き換えます。
Right(str$, n)$	文字列 **str$** をとり、右から数えて **n** 文字までを返します。 **サンプル**: • `str$ = "abcde"; Right(str$,n)$` cde.を返します。 • `right(col(d),8)` right(col(d),8)の文字列の右側 8 文字を返します。
Search(within$, find$[, StartPos])	文字列 **within$** で、文字列 **find$** を探し、位置を返します。見つからない場合-1 を返します。 大文字小文字の区別はなく、ワイルドカードは無効です。オプション **StartPos** は検索開始位置を制御します (デフォルト = 1)。**サンプル**: • `within$ = "abcde"; find$ = "BC"; Search(within$,find$)` 2.を返します。 • `search(col(c),"sample")` search(col(c),"sample")の文字列内の語句 "sample" の位置を返します。
Substitute(within$,sub$,find$ [, n])$	文字列 **within$** 内の **find$** を探し、**sub$** で置き換えます。 オプションは、**n** 番目に見つかったものだけ置き換えを行います。**サンプル**: • `Substitute(abcdefabcdef,12,bcd,0)$` a12efa12ef.を返します。 • `substitute(col(c),"experiment: ","expt.",1)` substitute(col(c),"experiment: ","expt.",1) col(a)の文字を検索して、最初に検出された時に、"experiment を"expt."に置換します。
Text(d[,fmt$])$ (9.1 SR0)	double 型を文字列に変換します。 **fmt$**オプションは出力をフォーマットし、**これらの値**をとります。指定されていない場合、列フォーマットを使用します。@SD 桁を使用するために、空の文字列""を使用します。Origin 全体の設定を使用するため、」*」を使用します。**サンプル**: • `Text(2.01232,"*3")$` 2.01.を返します。 • `Text(Date(7/10/2014),D1)$` Thursday, July 10, 2014.を返します。

	• `text(date(col(b)),D1)$` text(date(col(b)),D1)$ は日付の列をとり、"Wednesday, March 05, 2014"の形式で文字列を返します。
Token(str$,iToken[, iDelimiter])$	文字列 **str$** をとり、インデックス **iToken** に応じた部分文字列を返します。 オプション iDelimiter は、区切り文字の ASCII 値で、0 (デフォルト)のときは全ての空白スペース、32 のときは単一空白スペース、124 のとき"\|"となります。'_' (ASCII 95), '\|'(ASCII 124)などのほとんどの区切り文字を **iDelimiter** として直接使用できます。**サンプル**: • `str1$="This is my string"; Token(str1$,3)$ my`.を返します。 • `token(col(c),2)` token(col(c),2)の文字列内の 2 番目のトークン(スペース区切り)を返します。 • `token(col(b), 3, &apos;:&apos;)` token(col(b), 3, ':') の文字列内の 3 番目のトークン(コロン区切り)を返します。 Note: いくつかのシンボル文字は直接 **iDelimiter** で使用できませんが、ASCII 値はいつでも適用できます。
Trim(str$[, n])$	文字列 **str$** をとり、スペースを除きます。 パラメータ **n** は、どのように空白を削除するのかをしていします。0 (デフォルト) = 先行 + 後続, 1 = 全て除く です。**サンプル**: • `str1$ = " abc ABC "; Trim(str1$,0)$ abc ABC` を返します。 • `trim(col(a),0)` trim(col(a),0) は先行と後続のスペースが制御された col(c)の文字列を返します。
Upper(str$)$	文字列 **str$** をとり、大文字として返します。 **サンプル**: • `str1$ = "abcde"; Upper(str1$)$` returns `ABCDE`. • `upper(col(c))$` upper(col(c))$ は、文字列 col(F)の大文字を返します。
Value(str$)	文字列数 **str$** をとり、double として返します。 **サンプル**: • `str$ = "+.50"; Value(str$) 0.5`.を返します。**Note:** atof() 関数を確認してください。 • `value(col(e))` value(col(e)) は col(e)内の文字列数をとり double 型の数字として返します。

″$″表記法と文字列関数に関する注釈

Note: 文字列を操作するときに "$"を使用すると、混乱することがあります。




```
aa$=col(b)[1];
aa$=;
// returns
col(b)[1]
aa$=col(b)[1]$;
aa$=;
// returns
Buick
aa$=upper(col(b)[1]$);
aa$=;
// returns
upper(col(b)[1]$)
aa$=upper(col(b)[1]$)$;
aa$=;
// returns
BUICK
```

21.1.2. 数学関数

名前	説明
abs(x)	x の絶対値を返します。 **サンプル**: • `abs(-2.5)` returns `2.5`; `abs(0/0)` returns `--` (missing value); • `abs(col(b))` abs(col(b)) は col(b)全内容の絶対値を返します。
ceil(x[, sig]) (2019 SR0)	指定された値 **x** を 0 から **x** に近づけ、**d** に最も近い **sig** の倍数に調整して値を返します。 **サンプル**: • `ceil(2.5, 2)` `4` を返します。

	• `ceil(-2.5, 2) -2`.を返します。
Combina(n,k) (2019 SR0)	与えられた **n** 要素から、**k** 要素選んだときの組み合わせの数を返します。 **サンプル**: • `combina(4,2) 10`.を返します。
Combine(n1,n2)	与えられた **n1** 要素から、**n2** 要素選んだときの組み合わせの数を返します。 **サンプル**: • `combine(4,2) 6`.を返します。 • `combine(col(a),2` col(a)の値は、col(a)内の値の 2 要素の組み合わせの数を返します。
Distance(px1, py1, px2, py2)	2 点の XY 座標をとり、最短距離を返します。 **サンプル**: • `distance(0,0,0,1) 1`.を返します。 • `distance(col(g),col(h),col(i),col(j))` distance(col(g),col(h),col(i),col(j)) も、1 を返すことが出来ます。
Distance3D(px1, py1, pz1, px2, py2, pz2)	2 点の XYZ 座標をとり、最短の 3D 距離を返します。 **サンプル**: • `distance3d(0,0,1,0,0,2) 1`.を返します。 • `distance3d(col(a),col(b),col(c),col(d),col(e),col(f))` distance(col(g),col(h),col(i),col(j)) も、1 を返すことが出来ます。
exp(x)	**e** の **x** 乗を返します。 Note: x > 667 の場合欠損値を返します。**サンプル**: • `exp(0) 1`.を返します。 • `exp(col(a))` exp(col(a))の col(a)の値乗を返します。
expm1(x)	**x** の小さい値に正確な exp(**x**)-1 を返します。**サンプル**: • `expm1(0.00574) 0.0057565053651536`.を返します。
fact(n)	非負の整数値の階乗を返します。 Note: n > 170 の時は欠損値を返します。(Log_gamma 関数を確認してください**サンプル**: • `fact(3) 6`.を返します。 • `fact(col(a))` fact(col(a))は col(a)の値の階乗を返します。
factdouble(n)	非負の整数値の二重階乗を返します。

	n = 奇数の場合、シーケンスは 1*3*5...(n-2)*n; **n** = 偶数の場合、シーケンスは 2*4*6...(n-2)*n; **n** = 0 の場合、1 として評価されます。**n** > 299 の場合欠損値を返します。**サンプル**: • `factdouble(6) 48` を返します。 • `factdouble(col(a))` factdouble(col(a))は col(a)の値の階乗を返します。
floor(x[, sig]) (2019 SR0)	与えられた値 **x** を 0 に近づけ、**x** に最も近い有意数の倍数に調整して値を返します。 **サンプル**: • `floor(2.5, 2) = ; 2` を返します。 • `floor(-2.5, -2) = ; -2`.を返します。
gcd(n1, n2[, ...]) (2019 SR0)	与えられた整数 n1、n2、n3 などのグループの最大公約数を返します。**サンプル**: • `gcd(360, 15) 15` を返します。
frac(x)	double 型の整数部分を返します。 **サンプル**: • `frac(3.1415) = ; 0.1415`.を返します。
HaversineDistance(lon1,lat1,lon2,lat2,r[,degree]) (2017 SR0)	半径 **r** を持った球状にある 2 つのポイントの緯度経度を使うと、それらの大円距離を返します。オプションの **度** は緯度経度の単位として、度かラジアンのどちらを設定するかを決定します。(初期設定は度) **サンプル**: • `HaversineDistance(120, 30, 0, -60, 5000) 11388.7402734106` を返します。
int(x)	double 型 **x** をとり、切り捨て整数値を返します。 **サンプル**: • `int(7.9) 7`.を返します。 • `int(col(a))` int(col(a)) は col(a)の切り捨て整数値を返します。
ln(x)	x の自然対数を返します。
ln1p(x)	**x** が 1 に非常に近いときの **x** の自然対数を返します。
log(x)	x の底 10 の x の対数を返します。

mod(n, m)	整数 **n** を整数 **m** で割ったときの整数の剰余(0 に丸めた商)を返します。 **サンプル**: • `mod(16,7) 2`.を返します。 • `mod(col(a),col(b))` mod(col(a),col(b)) は、col(a) の値を col(b)の値で割った整数の剰余を返します。
mod2(n,m)	整数 **n** を整数 **m** で割ったときの整数の剰余(負の無限大への丸めの商)を返します。係数の計算に使用される$\frac{n}{m}$の商は$-\infty$で丸められます。入力が負の場合、mod ($\frac{n}{m}$ の商を 0 への丸めで計算)と異なる値を返します。**サンプル**: • `mod(5,-3)` 戻り値 `-1` • `mod(col(a),col(b))` mod(col(a),col(b)) は、col(a) の値を col(b)の値で割った整数の剰余を返します。
nint(x)	double 型 **x** をとり、最近接整数に丸めます。 nint(x) 関数は、round(x, 0) と同じ結果を返します。
permut(n, k) (2019 SR0)	与えられた **n** 要素の集合から、指定された **k** 要素の置換の数を返します **サンプル**: • `permut(4, 2)` 戻り値 `12`
prec(x, n)	値またはデータセットをとり、**n** 有効数字を返します。 **サンプル**: • `x = 1234567; prec(x, 3) 1230000`.を返します。 • `prec(col(b), 3)` prec(col(b), 3) は有効数字 3 桁の col(b)の値を返します。
rmod2(x, y)	double 型の値 **x** を double 型の値 **y** で割ったときの double 型の値の剰余(0 に丸めた商)を返します。 $\frac{x}{y}$ の商は 0 への丸めです。**サンプル**: • `rmod(4.5, 2) 0.5`.を返します。 • `rmod(col(a),3)` rmod(col(a),3) は col(a)を 3 で割った rmod を返します。
rmod2(x, y)	double 型の値 **x** を double 型の値 **y** で割ったときの double 型の値の剰余(負の無限大への丸めの商)を返します。 $\frac{x}{y}$ の商は $-\infty$ への丸めです。**サンプル**: • `rmod2(-4.5, 2) 1.5`.を返します。 • `rmod(col(a),3)` rmod(col(a),3) は col(a)を 3 で割った rmod を返します。

round(x, n)	値またはデータセットをとり、小数 点以下か多数 n で返します。 **Note:**Origin 9.1 で新しい端数処理のアルゴリズムを導入しています。システム変数 @RNA は、新旧メソッドを切り替えます (旧メソッド @RNA=0; 新メソッド @RNA=1 (デフォルト))。**サンプル**: • `round(1.156, 1) 1.2`.を返します。 • `round(col(a),2)` round(col(a),2) は col(a)の値を 2 桁に丸めた値を返します。
sign(x)	実数 **x** をとり、符号を返します。 **x** > 0 の場合、1 を返し、If **x** < 0 の場合、-1 を、**x** = 0 の場合 0 を返します。
sqrt(x)	double 型 **x** をとり、平方根を返します
Derivative(vd[,n])	ベクトル **vd** をとり、データリストの導関数を返します。 スムージングは無効です。オプション **n** で次数を指定し、デフォルトは 1 です。
DerivativeXY(vx, vy [, n])	2 つのベクトル **vx** と **vy** をとり、曲線の導関数を返します。 オプション **n** で次数を指定し、デフォルトは 1 です。
Integral(integrand,lowerlimit, upperlimit[, arg1, arg2, ...])	1 つの次元積分値を実行し、以下の積分値を返します。 $\int_{LowerLimit}^{UpperLimit} integrand(t, arg1, arg2, ...)dt$
Integrate(vd)	曲線下の面積積分をします。 台形公式を使用します。
IntegrateXY(vx, vy)	曲線 (**vx**, **vy**) 下の面積積分をします。 ベクトル **vx** は曲線の x 座標を含み、**vy** は y 座標を含みます。
Interp(x,vX,vY[,method,bound,smooth,extrap]) (2015 SR0)	x 座標 **vx** と y 座標 **vy** をとり、与えられた座標 **x** において y 座標を補間/補外します。オプション **method** = 0(線形, デフォルト), 1 (3 次スプライン), 2 (3 次 B スプライン), 3 (Akima スプライン)。method = 1 のとき **bound** は 0 (自然) 1 (not-a-knot 条件)。method = 2 のとき **smooth** はスムージング因子(負でない)。**extrap** では、X の範囲が参照範囲外にある場合に適用されます。0 (デフォルト)のとき、最後の 2 点を使用して Y を補外、1 のとき 全ての Y を欠損値としてセット、2 のとき最も近い入力 X の Y 値を使用します。
permutationa(n, k) (2019 SR0)	与えられた **n** 個の要素の集合から、指定された **k** 個の要素についての（反復を伴う）置換の数を返します**サンプル**: • `permutationa(4, 2)` 戻り値 `16`

21.1.3. 特別な数学関数

名前	説明
beta(a, b)	パラメータ **a** と **b** を持つベータ関数 $beta(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1}$
incbeta(x, a, b)	パラメータ **x**, **a**, **b** を持つ不完全ベータ関数 $incbeta(x,a,b) = \int_0^x t^{a-1}(1-t)^{b-1}\,dt$
incf(x, m, n)	不完全 F テーブル関数　パラメータ **x** は、積分範囲の上限、パラメータ **m** は、分子分散の自由度、パラメータ **n** は、分母分散の自由度。
incgamma(a,x)	パラメータ **a** で **x** での不完全ガンマ関数を計算 $incgamma(a,x) = P(a,x) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1}e^{-t}\,dt$ ここで、$\Gamma(a)$ は **a** における Gamma 関数の値で、a > 0 かつ x ≥ 0
inverf(x)	**x** での逆誤差関数を計算
j0(x)	0 次のベッセル関数
j1(x)	1 次のベッセル関数
jn(x, n)	**n** 次のベッセル関数 $J_n(x,n) = (x/2)^n \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{2k}}{k!\Gamma(k+n+1)}$ ここで、Γ は gammaln(*x*) 関数
y0(x)	0 次の第二種ベッセル関数
y1(x)	1 次の第二種ベッセル関数
yn(x, n)	**n** 次の第二種ベッセル関数 $Y_n(x,n) = \lim_{v \to n} Y_n(v,n)$ ここで $Y_n(v,n) = \frac{J_n(x,v)\cos(v\pi) - J_n(x,-v)}{sin(v\pi)}$

21.1.4. 三角関数/双曲線関数

Note: 角度の単位(ラジアン、度、グラジアン)は、**system.math.angularunits** プロパティに依存します(メインメニューの**環境設定**:**オプション**:**数値フォーマット**)

名前	説明
acos(x)	**x** の逆余弦を返します。 x < -1 あるいは x > 1 の場合、欠損値 ("--") を返します。
acosh(x)	**x** の逆双曲線余弦の値を返します。 x < 1 の場合、欠損値 ("--") を返します。
acot(x)	**x** の逆正接を返します。 入力 x はどんな値でもとれます。第一または第二象限の値を返します。
acoth(x)	|x| > 1 の逆双曲線余接の値を返します。
acsc(x)	|x| ≥ 1 の逆余割の値を返します。x < 1 の場合、欠損値 ("--") を返します。第一または第四象限の値を返します。
acsch(x)	**x** の逆双曲線余割の値を返します。 x = 0 あるいは x > ~3E153 の場合、欠損値 ("--") を返します。
angle(x, y)	原点(0,0)と座標(**x,y**)を結ぶ線と正の x 軸の間の角度(ラジアン)を返します。
Angleint1(px1, py1, px2, py2 [, unit, direction])	対の x,y 座標をとり、2 点と X 軸によって定義される線間の角度を返します。 **unit** では、0 のときデフォルトでラジアン、1 のとき角度値 となります。**direction** では、0 (デフォルト) のとき、第一 (+x,+y) と第四 (+x,-y) 象限に角度値を返し、1 のとき、0–2pi ラジアンまたは 0-360 度の値を返します。**サンプル**: • `angleint1(1,1,3,3,1) 45`.を返します。 • `angleint1(col(a),col(b),col(c),col(d),1,1)` angleint1(col(a),col(b),col(c),col(d),1,1) は、2 組の xy 座標で定義される線と X 軸の間の角度 (0 - 360) を返します。
Angleint2(px1, py1, px2, py2, px3, py3, px4, py4 [, unit, direction])	(**px1**, **py1**)と(**px2**, **py2**)を結ぶ線と、(**px3**, **py3**)と(**px4**, **py4**)を結ぶ別の線の 2 つの線の間の角度を返します。 **unit** = 1 の場合、単位は度となり、n=0 の場合、単位はラジアンとなります。オプション **direction** は返す値の方向を指定します。**direction** =0 (デフォルト)の場合、第一象限 (+x,+y) と第四象限 (+x,-y) に制限、**derection** = 1 の場合、0–2pi ラジアンまたは 0-360 度の値を返します。**サンプル**: • `angleint2(0,0,1,0,0,1,0,0,1,1) 90`.を返します。

	• `angleint2(col(a),col(b),col(c),col(d),col(e),col(f),col(g),col(h),1,1)` angleint2(col(a),col(b),col(c),col(d),col(e),col(f),col(g),col(h),1,1) は、col(a) - col(h)の最終ポイント間の角度 (degrees, 0 - 360) を返します。
asec(x)	**x** の逆正割を返します。 x < 1 の場合、欠損値 ("--") を返します。第一または第二象限の値を返します。
asech(x)	**x** の逆双曲線正割の値を返します。0 ≤ **x** ≤ 1 です。他の値をとる x に対しては欠損値("--")を返します。
asin(x)	**x** の逆正弦を返します。-1 ≤ **x** ≤ 1 で、他の値をとる x に対しては欠損値("--")を返します。
asinh(x)	**x** (実数) の逆双曲線正弦を返します。
atan(x)	**x** (実数) の逆正接を返します。
atan2(y,x)	座標 **x**,**y** (double 型) をとり、正の X 軸と点(**x**,**y**) の間の角度を返します。atan(x) 関数の変化形です。-π と π の間の値を返します。角度は反時計回り方向 (y > 0)に対して正で、時計回り方向 (y < 0)に対して負となります。
atanh(x)	**x** の逆双曲線正接の値を返します。-1 ≤ **x** ≤ 1 で、他の値をとる x に対しては欠損値("--")を返します。
cos(x)	**x** の余弦を返します。
cosh(x)	**x** の双曲線余弦の値を返します。
cot(x)	**x** の余接を返します。
coth(x)	**x** の双曲線余接の値を返します。 x 値は 0 ではない値です。絶対値 > 710 の値では、欠損値 ("--") を返します。
csc(x)	**x** の余割を返します。 x = 0 の場合、欠損値 ("--") を返します。
csch(x)	**x** の双曲線余割の値を返します。 x 値は 0 ではない値です。Note: x > 710 (絶対値) の場合、欠損値 ("--") を返します。
Degrees(angle)	ラジアンの **angle** をとり、度を返します。
Radians(angle)	度の **angle** をとり、ラジアンを返します。
secant(x)	**x** の正割を返します。 Note: 日付の秒の値を返す **sec()** 関数と混同しないように気を付けてください。
sech(x)	**x** の双曲線正割の値を返します。 絶対値 > 710 の値では、欠損値 ("--") を返します。
sin(x)	**x** の正弦を返します。
sinh(x)	**x** の双曲線正弦の値を返します。

tan(x)	**x** の正接を返します。
tanh(x)	**x** の双曲線正接の値を返します。

21.1.5. 日付と時間の関数

Origin 2019 以降、Origin には 3 つの日時システムがあります。古くからのデフォルトのシステムは、**Origin の日付と時刻**で説明されているように、**調整済みのユリウス日システム**です。次の例は、長期的なデフォルトの日時スキームに依存しており、"Julian-date value"が表示されている場合、Origin の調整日の値を参照します。これらの機能は、代替システムで動作するはずです...

```
date2str(today(), "MM/dd/yyyy")$ = 09/27/2018 // default date-time system, @DS=0
date2str(today(), "MM/dd/yyyy")$ = 09/27/2018 // "2018" system, @DS=2018
```

...ただし、カレンダーの日付と同じ数値は、システムによって異なります。

```
date(9/27/2018) = 2458388 // default date-time system, @DS=0
date(9/27/2018) = 269 // "2018" system, @DS=2018
```

Origin の代替日時スキームについては、Origin の代替日時システムを参照してください。

名前	説明
Date(MM/dd/yy[, format$]) と Date(yy,mm,dd)	日付- 時間の次列をとり、ユリウス通日の値を返します。 **format$** が指定されていない場合、文字列はシステムの short date フォーマットを使用して補間されます。**format$**文字列を指定せずに、1 = デフォルト(MM/dd/yyyy) , 2 (dd/MM/yyyy), 3 (yyyy/MM/dd) をとって最初の引数の日付フォーマットを制御します。**サンプル**: • `date(24-09-2009,"dd-MM-yyyy")` 2455098.を返します。 • `date(3/5/14)` date(3/5/14) は、2456721 (US) を返しますが、`date(5/3/14)` は、`2456721` (UK) を返します。 • `date(2/1/1986 13:13, 2)` date(2/1/1986 13:13, 2) は、`2446462.5506944` (US) を返しますが、`date(2/1/1986 13:13, 1)` は、`2446432.5506944` (UK) を返します。 • `date(col(a))` date(col(a)) は col(a)内の日時文字列のユリウス暦を返します。 または 年として double 型 **yy**、月として **mm**、日として **dd** をとり、ユリウス通日の値を返します。

Date2str(d,format$)$	ユリウス通日の値をとり、日付文字列を返します。 **サンプル**: • `date2str(2456573.123, "dd/MM/yyyy HH:mm")$` date2str(2456573.123, "dd/MM/yyyy HH:mm")$ は `08/10/2013 02:57` を返します。 • `date2str(col(b), "dd/MM/yyyy HH:mm")$` date2str(col(b), "dd/MM/yyyy HH:mm")$ は、"dd/MM/yyyy Hh:mm" 形式でデータ文字列を返します。
DatePart(datepart$, d [, n]) (2016 SR1)	ユリウス通に値(double 型)**d** を取り、double 型として **datepart$**で指定された一部日にちを返します。**n** は、週の始まり datepart$ = w or ww を指定します。 **サンプル**: • `datepart("yyyy", 2457360.5107885)` 2015.を返します。 • `datepart("yyyy", A)` A 列の日付-時間データから年次部分を抜き出して返します。 • `datepart("y", Today())=;` datepart(y, Today())=;は、現在の年の日にちを返します。(例:today() = 2457363 = 12/7/2015 = 341). • `datepart("w", 2457360.5107885, 1)=;` date(2/1/1986 13:13, 2) は、`5` (US) を返しますが、`datepart(w, 2457360.5107885)=;` は、`6` (UK) を返します。
Day(d[,n])	ユリウス通日の値をとり、日数を返します。 オプション **n** = 1 の場合、1 から 31 (月)を返します。; **n** = 2 の場合、1 から 366 (年)を返します。**サンプル**: • `Day(2454827.5982639, 2)` は `362` を返します。 • `day(col(b),1)` day(col(b),1)は、ユリウス通日をとり、曜日を返します。
Hour(d)	ユリウス通日の値をとり、整数として時間を返します。 0 から 23 (0 = 12:00 A.M., 23 = 11:00 P.M.) の値を返します。**サンプル**: • `Hour(0.6997854)` 16.を返します。 • `hour(col(b))` hour(col(b)) は col(b)のユリウス通日の値から時間を返します。
Minute(d)	ユリウス通日の値をとり、整数として分を返します。 **サンプル**: • `Minute(2454827.5982639)` 21.を返します。 • `minute(col(b))` minute(col(b)) は col(b)のユリウス通日の値から時間を返します。

Month(d)	ユリウス通日の値をとり、整数として月を返します (0 から 12)。 **サンプル**: • `month(2454821) 12`.を返します。 • `month(col(b)` month(col(b) は col(b)のユリウス通日の値から時間を返します。
MonthName(d[,n])$	ユリウス通日の値をとり、月名を返します。 月のフォーマットはオプション n で指定します。1 のとき単一文字、3 のとき 3 文字（デフォルト)、0 のとき全ての月の名前、-1 のとき言語設定に関係なく英語の 3 文字です。**サンプル**: • `MonthName(2454827.5982639, 0)$ December`.を返します。 • `monthname(col(b),0)$` monthname(col(b),0)$ は col(b)のユリウス通日の値から月名を返します。
Now()	現在の日付/時刻を日付（ユリウス通日）の値として返します。 **サンプル**: • `time2str(now()-date(col(a)),"HH:mm")$` time2str(now()-date(col(a)),"HH:mm")$は現在の時刻と col(a)のユリウス通日間の経過時間文字列(HH:mm) を返します。
Quarter(d)	ユリウス通日をとり、四半期を返します。 **サンプル**: • `Quarter(2454829.5745718) 4`.を返します。 • `quarter(col(b))` month(col(b) は col(b)のユリウス通日の値から時間を返します。
Second(d[,n])	ユリウス通日の値または実数をとり、0 から 59.9999...の値として秒を返します。オプション ***n*** = *0* は 3 桁以上の 10 進数字を表示しますが、ユリウス暦の日付の精度は 4 桁目の小数点以下を四捨五入すると 0.0001 秒に制限されます。 **サンプル**: • `second(2454827.5982639) 30.001`.を返します。 • `second(2454827.5982639, 0) 30.000942349434`.を返します。 • `second(A)` second(col(a)) は col(a)のユリウス通日の秒を返します。
Time(HH,mm,ss) と Time(HH:mm:ss[,Format$])	HH,mm,ss あるいはカスタム日付/鵜時間文字列 (HH:mm:ss = デフォルト) をとり、ユリウス通日を返します。オプション **Format$** 引数は、カスタム文字列フォーマットを指定します。**サンプル**: • `time(20:50:25)` time(20:50:25) は、`0.8683449` を返しますが; `time("20,50,25", "DDD hh,mm,ss")` は、`2.8683449` を返します。 • `time(col(a))` time(col(a))は、col(a)の時間データフォーマットを HH:mm:ss としてユリウス通日を返します。

Time2str(d,format$)$	ユリウス通日をとり、指定されたフォーマットの時間文字列を返します。 **サンプル**: • `time2str(0.1875, "HH:mm")$ 04:30`.を返します。 • `time2str(col(b),"DDD:HH")$` time2str(col(b),DDD:HH)$ は、DDD:HH 形式で時間文字列を返します。
Today()	現在の日付のユリウス通日値を返します。
WeekDay(d[,n])	ユリウス通日をとり、曜日を返します。 週の開始と終了は、オプション **n** で指定します。 0 (デフォルト)は曜 (0-6)、1 は日曜 (1-7)、2 は月曜 (0-6)、3 は月曜 (1-7)です。**サンプル**: • `weekday(2454825, 1) 5`.を返します。 • `weekday(col(b))` weekday(col(b)) は col(b)のユリウス通日をとり曜日を数字として返します。
WeekDayName(d[,n1,n2])$	ユリウス通日(時間を含む)または、**n2** で定義された値をとり、平日を返します。 **n1** で、文字列の長さを指定します。-1 = 3 文字; 0 = 全ての名前で、最初を大文字表記; 1 = 最初の文字 1 文字で、大文字表記; 3(デフォルト)= 3 文字で、最初の文字を大文字表記 です。n2 で、週の開始と終了を指定します。0 = 0(日曜) - 6(土曜); 1 = 1(日曜) - 7(土曜); 2 = 0(月曜) - 6(日曜); 3(デフォルト) = 1(月曜) - 7(土曜).**サンプル**: • `WeekDayName(2454825,-1,0)$ THU`.を返します。 • `weekdayname(col(b),3,0)$` weekdayname(col(b),3,0)$ は、3 文字で最初の文字が大文字の曜日の名前を返します。
WeekNum(d[,n])	ユリウス通日の値をとり、年を通した週数を返します (1 から 53)。 週の始まり(日曜か月曜)を指定するオプションがあります。**サンプル**: • `weeknum(date(1/11/2009)) 3`.を返します。 • `weeknum(date(col(c)))` weeknum(date(col(c)))は、フォーマット "MM/dd/yyyy" の日付データ(col(c))をとり、date()関数を使用してユリウス暦にして、weeknum() 関数で週数を返します。
Year(d)	ユリウス通日の値をとり、整数として年を返します (0100 から-9999)。 **サンプル**: • `year(2454821) 2008`.を返します。 • `year(date(col(c)))` year(date(col(c))) は、フォーマット"MM/dd/yyyy" の日付データ(col(c))をとり、date()関数を使用してユリウス暦にして、4 桁の年を返します。

YearName(d[,n])$	ユリウス通日の値をとり、文字列として年を返します。 **n** によって、文字の桁数を指定します。0 のとき 2 桁、1(デフォルト)のとき 2 桁 (アポストロフィ付)、2 のとき 4 桁 です。**サンプル**: • `YearName(2454827.5982639, 1)$ &apos;08`.を返します。 • `yearName(date(col(c)),0)$` year(date(col(c))) は、フォーマット "MM/dd/yyyy" の日付データ(col(c))をとり、date()関数を使用してユリウス暦にして、4 桁の年を返します。

21.1.6. 論理関数

名前	説明

If(con, val_true$[,val_false$])$
と
If(con, val_true[,val_false])
(2019 SR0)
	条件式 **con** を評価し、比較が TRUE の場合は **val_true**、偽の場合は **val_false** を返します**サンプル**
	• `if(A>B,1,0)` col(A)> col(B)の場合は 1 を返し、そうでない場合は 0 を返します。

Ifs(con1, d1[, con2, d2,]...[, con40, d40]) と Ifs(con1, str1$[, con2, str2$]...[, con40, str40$])$ (2019 SR0)	複数の条件 **conn** を評価し、最初の TRUE 条件の対応する d / str を返します。**サンプル** • `Ifs(A>0.5,"Large",A<0.3,"Small",1,"Other")$`col(A)の値が 0.5 より大きい場合、Large を返し、0.3 より小さい場合は Small を返し、その間の残りの場合は Other を返します。
IfNA(val, val_na$)$ (2019 SR0)	指定された数式 **val** を計算し、指定された文字列 **val_na** を結果がない場合は返し、そうでない場合は **val** の結果の文字列表示を返します。**サンプル** • `IfNA(col(A)/col(B),"not found")$`*col(A)/ col(B)*が見つからない場合は "not found"を返し、そうでない場合は *col(A)/ col(B)*の文字列表示を返します。
switch(expression, val1, res1$[, val2, res2$]...[, val39, res39$][, default$])$ と switch(expression, val1, res1[, val2, res2]...[, val39, res39][, default]) (2019 SR0)	値の集合を評価値 **val** で評価し、一致した値 **valn** がある場合は、対応する **resn** を返します。**サンプル** • `switch(A,1,"A1",2,"B1",3,"C1","Other")$`

21.1.7. 信号処理関数

名前	説明
fftamp(cx [,side]) (2015 SR1)	複素ベクトル **cx**(通常 FFT の複素結果)をとり、振幅を返します。オプション **side** は、スペクトルの出力を定義(1 = 片側, 2 = 両側とシフト)**サンプル**: • `fftamp(fftc(col(B)), 2)` fftamp(fftc(col(B)), 2) は入力信号列 B の FFT 結果(両側)の振幅を返します。 • `col(C) = col(B)-mean(col(B)); fftamp(fftc(col(C)))` col(C) = col(B)-mean(col(B)); fftamp(fftc(col(C))) は、DC オフセットを除去した波形の結果を返します。(片側)
fftc(cx) (2015 SR1)	ベクトル型 **cx** をとり、複素 FFT 結果を返します。出力列のデータ型を complex (16)にする必要があります。**サンプル**: • `fftc(col(B))` fftc(col(B)) は入力信号列 B の FFT 複素結果を返します。 • `fftc(rSignal)` fftc(rSignal) は範囲変数 rSignal の入力信号の FFT 複素結果を返します。
fftfreq(time, n[, side , shift]) (2015 SR1)	サンプリング間隔 **time** と信号サイズ **n** とり、FFT 結果の周波数を返します。オプション **side** は、スペクトルの出力を定義(1 = 片側, 2 = 両側)し、**shift** は両側のときにシフトするか定義します。(0 = シフトなし, 1 = シフト)します。**サンプル:** • `fftfreq(0.001, 100)` fftfreq(0.001, 100) は 0 から 500 で間隔 10 のデータセットを返します (片側, シフトなし) • `fftfreq(0.01, 100, 2, 1)` fftfreq(0.01, 100, 2, 1) はサンプリング間隔 0.01 の両側でシフトされた周波数を返します。
fftmag(cx [,side]) (2015 SR1)	複素ベクトル **cx**(通常 FFT の複素結果)をとり、マグニチュードを返します。オプション **side** は、スペクトルの出力を定義(1 = 片側, 2 = 両側とシフト)。**サンプル**: • `fftmag(fftc(col(B)), 2)` fftamp(fftc(col(B)), 2) は入力信号列 B の FFT 結果(両側)の振幅を返します。 • `col(C) = col(B)-mean(col(B)); fftmag(fftc(col(C)))` col(C) = col(B)-mean(col(B)); fftamp(fftc(col(C))) は、DC オフセットを除去した波形の結果を返します。(片側)

fftphase(cx[, side, unwrap, unit]) (2015 SR1)	複素ベクトル **cx**(通常 FFT の複素結果)をとり、位相を返します。オプション **side** は、スペクトルの出力を定義(1 = 片側, 2 = 両側とシフト)。**unwrap** は位相角度の巻きの有無(0 = 巻き, 1 = 巻きなし)。**unit** は単位 (0 = ラジアン, 1 = 度)。**サンプル**: • `fftphase(fftc(col(B)))` fftphase(fftc(col(B))) は入力信号列 B の FFT 結果(片側、巻きなし、度)の位相を返します。 • `fftphase(fftc(col(B)), 2, 0, 0)` fftphase(fftc(col(B)), 2, 0, 0) は入力信号列 B の FFT 結果(片側、巻きなし、度)の位相を返します。
fftshift(cx) (2015 SR1)	複素ベクトル **cx**(通常 FFT の複素結果または周波数)をとり、シフトされたベクトルを返します。出力列のデータ型を complex (16)にする必要があります。**サンプル**: • `fftshift(fftc(col(B)))` fftshift(fftc(col(B))) はシフトした複素数ベクトルを返します。
ifftshift(cx) (2015 SR1)	複素ベクトル **cx**(通常 FFT の複素結果またはシフトされた FFT 結果)をとり、シフトされていないベクトルを返します。出力列のデータ型を complex (16)にする必要があります。**サンプル**: • `ifftshift(col(B))` ifftshift(col(B)) 列 B に入力されたシフトされた複素ベクトルのシフトされていないベクトルを返します。
invfft(cx) (2015 SR1)	複素ベクトル **cx**(通常 FFT の複素結果)をとり、逆 FFT 結果を返します。出力列のデータ型を complex (16)にする必要があります。**サンプル**: • `invfft(ifftshift(col(B)))` invfft(ifftshift(col(B))) 列 B の FFT 複素結果の逆 FFT を返します。
windata(type, n) (2015 SR1)	**type**(ウィンドウ法)と **n**(ウィンドウサイズ)の整数をとり、サイズ n のベクトルとしてウィンドウの信号を返します。**サンプル**: • `windata(2, 50)` windata(2, 50) ベクトルサイズ 50 の三角ウィンドウの信号を返します。

21.1.8. 統計関数

名前	説明
averageif(vd, con$) (2015 SR0)	ベクトル **vd** と条件 **con$** をとり、**con$**を満たす値の平均を返します。**サンプル**: • `col(A) = data(1,32); con$ = col(A) > 5 && col(A) < 10; averageif(col(A), con$)=; 7.5.`を返します。

Correl(vx, vy) (2015 SR0)	データセット **vx** と **vy** をとり、相関係数を返します。**サンプル**: • `for(ii=1;ii<=30;ii++) col(1)[ii] = ii; col(2)=ln(col(1)); correl(col(A),col(B))=; 0.92064574677971.`を返します。
Count(vd[,n])	ベクトル型 **vd** をとり、要素の数を返します。 オプションの **n** で要素を指定します。0 (デフォルト) = 全て; 1 = 数値; 2 = 欠損値 です。**サンプル**: • `count(col(a),2)` count(col(a),2) は 22 と欠損値の数を返します。
Countif(vd,con$) (2015 SR0)	ベクトル型 **vd** をとり、条件 **vcon** を満たす値のカウントを返します。条件 **vcon** は二重引用符("")で囲む必要があります。 • `countif(col(b),"col(b)>0")=;` • `countif(col(A),"col(A)<10 && col(A)>5")=;`
cov(vx, vy[, avex, avey])	データセット **vx** と **vy** 、それぞれの平均 **avex** と **avey** をとり、共分散を返します。 **サンプル**: • `for(ii=1;ii<=30;ii++) col(1)[ii] = ii; col(2)=ln(col(1)); cov(col(A),col(B))=; 6.8926313172818.`を返します。
Forecast(x,vx,vy) (2019 SR0)	**x** 座標を **vx** と y 座標で **vy** とし、線形回帰を実行して、与えられた座標 x で y 座標を計算または予測します。
Intercept(vx,vy) (2019 SR0)	**vx**(独立)と **vy**(従属)の 2 つのベクトルを取り、線形回帰の切片を返します。
Max(vd)	ベクトル型 **vd** をとり、最大値を返します。 **サンプル**: • `max(col(A))` col(A)の最大値を返します。 • `max(1,2,3,4,9) 9.`を返します。
Maxifs(vd,con$) (2019 SR0)	ベクトル型 **vd** をとり、条件 **con** を満たす最大値を返します。**サンプル** • `maxifs(col(A), "col(A)>5")` col(A)の部分集合の最大値を 5 より大きく返します。
Mean(vd)	ベクトル型 **vd** をとり、平均を返します。 **サンプル**: • `mean(col(A))` mean(col(A)) は、`col(A)` の平均値を返します。

Median(vd[,method])	ベクトル型 **vd** をとり、中央値を返します。 オプション n で補間方法を指定します。0 (デフォルト) = 経験分布の平均; 1 = 近傍法; 2 = 経験分布; 3 = 加重平均右; 4 = 加重平均左; 5 = Tukey hinge です。**サンプル**: • `median(col(A),2)` 中央値を返します(n = 2 で決定される)。補間手法についての詳細情報は、分位数の補間を確認してください。
Min(vd)	ベクトル型 **vd** をとり、最小値を返します。
Minifs(vd,con$) (2019 SR0)	ベクトル型 **vd** をとり、条件 **con** を満たす最小値を返します。**サンプル** • `minifs(col(A), "col(A)>5")` col(A)の部分集合の最小値を 5 より大きく返します。
rms(vd)	ベクトル型 **vd** をとり、二乗平均平方根を返します。
Slope(vx,vy) (2019 SR0)	**vx**(独立)と **vy**(従属)の 2 つのベクトルを取り、線形回帰の切片を返します。
Ss(vd [,ref])	ベクトル型 **vd** をとり、平方和を返します。 平方和は、いくつかの参照値 **ref** を **vd** の各値から減算したあと計算されます。オプション ref は、vd の平均をデフォルトとしますが、定数やデータセット、関数にすることもできます。**サンプル**: • `ss(vd)` ss(vd) は、平均を減算した平方和を返します。 • `ss(vd,4)` ss(vd,4) は *vd* の各要素から 4 を引いた後計算された平方和を返します。 • `ss(vd1,vd2)` ss(vd1,vd2) は *vd1* の各要素から *vd2* を引いた後計算された平方和を返します。 • `AA = 1; BB = 2; ss(vd, AA+BB*x)` AA = 1; BB = 2; ss(vd, AA+BB*x) は *vd* から *1+2x* で説明される行の減算後に平方和を返します。
StdDev(vd)	ベクトル型 **vd** をとり、標本標準偏差を返します。**サンプル**: • `StdDev(col(A))` StdDev(col(A)) はサンプル標準偏差を返します。
StdDevP(vd)	ベクトル型 **vd** をとり、母標準偏差を返します。**サンプル**: • `StdDevP(col(A))` StdDev(col(A)) は母標準偏差を返します。
sumif(vd,con$) (2015 SR0)	ベクトル型 **vd** をとり、条件 **vcon** を満たす値の合計を返します。

Total(vd)	ベクトル型 **vd** をとり、要素の合計を返します。**サンプル**: • `total(col(a))` total(col(a)) 列 A のすべてのデータポイントの合計を返します。
ave(vd, size)	ベクトル型 **vd** をとり、**size** の各グループの平均の範囲を返します。**vd** の要素が **size** の倍数でない場合、mod(vdSize,size) 要素のみ平均した値を返します。**サンプル**: • `ave(col(a),5)` ave(col(a),5) は *col(a)* をサイズ *5* のグループに分け、各グループの平均を計算します。
diff(vd[,n])	ベクトル型 **vd** をとり、隣接要素間を相違の範囲を返します。返された範囲の最初の要素は $\mathbf{vd_{(i+1)}-vd_i}$ などです。オプションのパラメータ **n** の値に応じて、N-1、N、または N + 1 要素を返します。 • 0 =（デフォルト）、N-1 要素を返します。 • 1 =データセット終了時に 0 でパッドし、N 個の要素を返します。 • 2 =データセット開始時に 0 でパッドし、N 個の要素を返します。 • 3 =データセット開始時に 0 でパッドし、要素 N + 1 が N 個の要素を合計して得られる N + 1 要素を返します。
sum(vd) または sum(WorksheetName, col1, col2)	ベクトル **vd** をとり、累積合計(1 から i, i=1,2,...,N)の値を保持するデータセットを返します。*i+1* 番目の要素は最初の i 要素の合計です。または ワークシート名 **WorksheetName** と 2 つの列インデックス **col1** と **col2** をとり、**col1** と **col2** をまたがる行の合計値を保持するデータセットを返します。
Confidence(alpha, std, size)	有意水準 **alpha**、母標準偏差 **std** とサンプル **size** をとり、母平均の信頼区間を返します。**サンプル**: • `confidence(0.05, 1.5, 100)` `0.29399459768101`.を返します。
histogram(vd, inc, min, max)	ベクトル **vd**、ビンの幅 = **inc**、vd の最小 **min** と vd の最大 **max** をとり、データビンを生成します。各ビン上限上のポイントは次のビンに含まれます。
Kurt(vd)	ベクトル型 **vd** をとり、尖度を返します。**サンプル**: • `dataset ds = {1, 2, 3, 2, 3, 4, 5, 6, 4, 8}; kurt(ds)` `0.39502164502164`.を返します。

Percentile(vx, vy)	ベクトル **vx** をとり、**vy** で指定した各パーセント値におけるパーセンタイル値を返します。**サンプル**: • `DATA1_A = normal(1000); DATA1_B = {1, 5, 25, 50, 75, 95, 99}; DATA1_C = percentile(DATA1_A, DATA1_B);` DATA1_A = normal(1000); DATA1_B = {1, 5, 25, 50, 75, 95, 99}; DATA1_C = percentile(DATA1_A, DATA1_B); は、1%, 5%...99%での正規分布のパーセンタイルを含むデータセット *DATA1_C* を返します。
QCD2(n)	サンプルサイズ **n** をとり、品質管理 D2 係数を返します。**サンプル**: • `QCD2(4) 2.05875.`を返します。
QCD3(n)	サンプルサイズ **n** をとり、品質管理 D3 係数を返します。係数 D3 は、X-R 管理図における 3 シグマ下部管理限界です。**サンプル**: • `QCD3(10) 0.223.`を返します。
QCD4(n)	サンプルサイズ **n** をとり、品質管理 D4 係数を返します。係数 D4 は、X-R 管理図における 3 シグマ上部管理限界です。**サンプル**: • `QCD4(10) 1.777.`を返します。
Skew(vd)	ベクトル型 **vd** をとり、歪度を返します。**サンプル**: • `skew(col(a))` skew(col(a)) は*列 A* の歪度を返します。
Emovavg(vd,n[,method])	ベクトル型 **vd**、整数 **n** = 時間幅をとり、指数移動平均を返します。オプション **method** は、計算開始地点を指定します。0 (デフォルト)のときは **n** から、1 のときは最初のポイントから計算を始めます。**サンプル**: • `for(ii=1;ii<=30;ii++) col(1)[ii] = ii; col(3)=emovavg(col(1),10, 1); //method II` for(ii=1;ii<=30;ii++) col(1)[ii] = ii; col(3)=emovavg(col(1),10, 1); //method II は、計算された開始ポイント1を使って、3 番目の列に数値を入れます。
Mmovavg(vd,n)	ベクトル型 **vd**、整数 **n** = 時間幅をとり、修正移動平均のベクトルを返します。**サンプル**: • `for(ii=1;ii<=30;ii++) col(1)[ii] = ii; col(2)=mmovavg(col(1),10);` for(ii=1;ii<=30;ii++) col(1)[ii] = ii; col(2)=mmovavg(col(1),10); は、10 行目から始まるそれぞれの値での修正移動平均値を 2 番目の列に数値を入れます。

Movavg(vd,back,forward)	ベクトル **vd** をとり調整した範囲[i-back, i+forward]の、i ポイントの移動平均を返します(i は現在の行番号)。**サンプル**: • `for(ii=1;ii<=10;ii++) col(1)[ii] = ii; col(2)=movavg(col(1),0, 2);` for(ii=1;ii<=10;ii++) col(1)[ii] = ii; col(2)=movavg(col(1),0, 2); は、それぞれのポイントでの隣接平均を col(2)に入力します。(Note that `col(2)[9] = (col(1)[9]+col(1)[10])/2 and col(1)[10] = col(2)[10]`).
Movcoef(v1,v2,back,forward)	ベクトル **v1** および **v2** をとり調整した範囲[i-back, i+forward]の、i ポイントの移動相関係数のベクトルを返します(i は現在の行番号)。**サンプル**: • `wcol(4) = MovCoef(wcol(2), wcol(3), 20, 0);` wcol(4) = MovCoef(wcol(2), wcol(3), 20, 0); とすると、ウィンドウ[i-20. i]内で、4 番目の列に col(2) と col(3)の移動相関係数が入ります。
Movrms(vd,back[,forward]) (2015 SR2)	ベクトル **vd** をとり調整した範囲[i-back, i+forward]の、i ポイントの二乗平均平方根を返します(i は現在の行番号)。**サンプル**: • `col(B)=movrms(col(A),0, 2);` col(B)=movrms(col(A),0, 2); は列 B にウィンドウ[i, i+2])内のデータの各ポイントにおける RMS を返します。
Movslope(vx,vy[,n])	2 つのベクトル **vx** (独立) と **vy** (従属) をとり、各ポイントの移動傾きのベクトルを返します。オプション **n** はウィンドウ幅を指定します(1 以上)。**n** が偶数の場合、1 が加えられます。n を指定しない場合、入力の線形フィットの傾きの値を返します。**サンプル**: • `col(C)=movslope(col(A),col(B),5);` col(C)=movslope(col(A),col(B),5); は、各ポイントの傾きを列 C に返します(最初と最後のセルは欠損値になります)。
Tmovavg(vd,n)	ベクトル型 **vd**、整数 **n** = 時間幅をとり、三角移動平均を返します。**サンプル**: • `for(ii=1;ii<=30;ii++) col(1)[ii] = ii; col(2)=tmovavg(col(1),9);` for(ii=1;ii<=30;ii++) col(1)[ii] = ii; col(2)=tmovavg(col(1),9); は、9 行目から始まるそれぞれの値での三角移動平均値を 2 番目の列に数値を入れます。
Wmovavg(vd,vw)	ベクトル型 **vd** (スムージングのためのデータ)、**vw** (インデックスされた重み付き因子)をとり、加重移動平均のベクトルを返します。**サンプル**: • `for(ii=1;ii<=30;ii++) col(1)[ii] = ii; //data vector for(ii=1;ii<=10;ii++) col(2)[ii] = ii/10; //weight vector col(3)=wmovavg(col(1),col(2));` for(ii=1;ii<=30;ii++) col(1)[ii] = ii; //data vectorfor(ii=1;ii<=10;ii++) col(2)[ii] = ii/10; //weight vectorcol(3)=wmovavg(col(1),col(2)); は、10 行目から始まるそれぞれの値での加重移動平均値を 3 番目の列に数値を入れます。

21.1.9. 分布関数

累積分布関数(CDF)

名前	説明
betacdf(x,a,b[,tail])	パラメータ *a* と *b* を持つ *x* におけるベータ累積分布関数を計算します。**a** と **b** はすべて正でなければならず、**x** は区間[0,1]上になければなりません。**tail** は返される確率が下側の tail または upper tail であることを決定します。
binocdf(k,n,p)	二項分布で、*n*, p に対応するパラメータを使って、与えられた値 *k* での上側確率/下側確率および点確率を計算します。 $binocdf(k,n,p) = P(X \le k) = \sum_{i=0}^{k} P(X=i)$ $= \sum_{i=0}^{k} \binom{n}{k} p^i (1-p)^{n-i}$
bivarnormcdf(x,y,corre)	二変量正規分布の下側確率を計算します。 $P(X \le x, Y \le y) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^{y}\int_{-\infty}^{x} \exp(\frac{x^2 - 2\rho XY + Y^2}{2(1-\rho^2)})dXdY$
chi2cdf(x,df[,tail])	自由度 **df** を持つカイ二乗分布に対する下側確率を計算します。
fcdf(f,ndf,fdf[,tail])	*f* における分子 **ndf** と分母の変数 **fdf** の自由度の変数 F 累積分布関数を計算します。**tail** は返される確率が下側の tail または upper tail であることを決定します。
gamcdf(g,a,b[,tail])	形状パラメータ *a* とスケールパラメータ *b* を使用し、ガンマ変量 *g* の自由度を持つガンマ分布の下側確率を計算します。**tail** は返される確率が下側の tail または upper tail であることを決定します。
hygecdf(k,m,n,l)	対応するパラメータを使った超幾何分布で、与えられた値での下側確率を計算します。 $hygecdf(k,m,n,l) = P(X \le k) = \sum_{i=0}^{k} P(X=i)$ $= \sum_{i=0}^{k} \frac{\binom{m}{i}\binom{n-m}{l-i}}{\binom{n}{l}}$ ここで、**n** は母集団のサイズ、**m** は母集団の中で成功状態の数、**l** は引き分けのサンプル数です。

logncdf(x,mu,sigma[,tail]) (2015 SR0)	対応するパラメータ *mu* および *sigma* を使用して Lognormal 分布に関連付けられた、指定された値 **x** の指定された **tail** 型テールの確率を計算します。**tail** が指定されていない場合、低いテール確率が返されます。
ncbetacdf(x,a,b,lambda)	非心ベータ分布の下側 cdf を計算します。 $$ncbetacdf(x,\,a,\,b,\,lambda) = P(B \le \beta) = \sum_{j=0}^{\infty} e^{-\lambda/2}\frac{(\lambda/2)^j}{j!}P_b(B \le \beta)$$ ここで $$P_b(B \le \beta) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\int_0^{\beta} B^{a-1}(1-B)^{b-1}dB$$ これは、ベータ分布関数や不完全ベータ関数です。
ncchi2cdf(x,f,lambda)	非心カイ二乗分布の下側確率を計算します。 $$ncchi2cdf(x,\,f,\,lambda) = P(X \le x:\nu;\lambda) = \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}}\frac{(\lambda/2)^j}{j!}P(X \le x:\nu+2j;0)$$ ここで、$P(X \le x:\nu+2j;0)$ は、$\nu+2j$ 自由度の中央 χ^2 です。
ncfcdf(f,df1,df2,lambda)	非心ディガンマまたは分散比分布の下側確率を計算します。 $$ncfcdf(f,\,df1,\,df2,\,lambda) = P(F \le f) = \int_{\lambda}^{f} P(F)dF,$$ Where $$P(F) = \sum_{j=0}^{\infty} e^{-\lambda/2}\frac{(\lambda/2)^j}{j!}\cdot\frac{(\nu_1+2j)^{(\nu_1+2j)/2}\nu_2^{\nu_2/2}}{B((\nu_1+2j)/2,\nu_2/2)} \cdot u^{(\nu_1+2j-2)/2}\left[\nu_2+(\nu_1+2j)u\right]^{-(\nu_1+2j+\nu_2)/2}$$ なお、$B(\cdot,\cdot)$は、ベータ関数です。
nctcdf(t,df,delta[,maxiter])	非心のスチューデントの t 分布に対する下側確率を計算します。 $$nctcdf(t,\nu,\delta,maxiter) = P(T \le t) = C_\nu\int_0^{\infty}\left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\alpha u-\delta} e^{-x^2}dx\right)u^{\nu-1}e^{-u^2/2}du$$ with $$C_\nu = \frac{1}{\Gamma(\frac{1}{2}\nu)2^{(\nu-2)/2}},\ \alpha = \frac{t}{\sqrt{\nu}},\ \nu > 0$$
normcdf(x[,tail])	正規の累積分布に関連付けられた、指定された値 **x** の指定された **tail** 型テールの確率を計算します。

poisscdf(k,lambda)	ポアソン分布で、*lambda* に対応するパラメータを使って、与えられた値 *k* での下側確率を計算します。 $P(X \le k) = \sum_{i=0}^{k} P(X = i) = \sum_{i=0}^{k} e^{-\lambda} \frac{\lambda^k}{k!}$
srangecdf(q,v,group)	スチューデント範囲統計分布の下側確率を計算します。 $P(q) = C \int_0^{+\infty} x^{\nu-1} e^{-\nu x^2/2} \{ r \int_{-\infty}^{+\infty} \Phi(y) \cdot [\Phi(y) - \Phi(y - qx)]^{r-1} dy \} dx$ Where
tcdf(t,df[,tail])	スチューデント t 分布の累積分布関数に自由度 **df** を用いて指定された **tail** 型テールの確率を計算します。
wblcdf(x,a,b)	パラメータ *a* と *b* を使って、*x* 値に対する下側ワイブル累積分布関数を計算します。 $P(X < x \mid a,b) = \int_0^x b a^{-b} t^{b-1} e^{-(\frac{t}{a})^b} dt = 1 - e^{-(\frac{x}{a})^b} I_{(0,+\infty)}(x)$ $I_{(0,+\infty)}(x)$は Weibull CDF がゼロではなく空間です。

確率密度関数(PDF)

名前	説明
betapdf(x,a,b)	パラメータ *a* と *b* を持つベータ分布の確率密度関数を返します。 $f(B:a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} B^{a-1}(1-B)^{b-1}$ $0 \le B \le 1; a,b > 0$
binopdf(x,nt,p) (2015 SR0)	パラメータ *nt* と *p* を持つ二項分布の確率密度関数を返します。 $f(x \mid nt,p) = \binom{nt}{x} p^x (1-p)^{nt-x},$ ここで $0 \le p \le 1$ および $x = 0,1,2,\ldots,nt$
cauchypdf(x,a,b) (8.6 SR0)	Cauchy 確率密度関数 (aka Lorentz 分布) $f(x \mid a,b) = \frac{1}{\pi b \left[1 + \left(\frac{x-a}{b}\right)^2\right]} = \frac{1}{\pi} \left[\frac{b}{(x-a)^2 + b^2}\right]$
exppdf(x,lambda) (8.6 SR0)	X の値で評価される速度パラメータ λ を持つ指数分布の確率密度関数を返します。 $f(x \mid \lambda) = \lambda e^{-x\lambda}, x \ge 0$

gampdf(x,a,b) (8.6 SR0)	パラメータ *a* および *b* を持つ Gamma 確率密度を返します。 $$f(x\|a,b) = \frac{1}{b^a \Gamma(a)} x^{a-1} e^{\frac{-x}{b}}$$ ガンマ分布データセットからスケールと形状パラメータの *a* と *b* を生成するには、推定関数 gamfit を使用します。
ks2density(x,y,vx,vy,wx,wy) (2015 SR0)	スケール(wx, wy)のデータセット (vx, vy)で確率される関数で(x, y)での 2D カーネル密度を返します $$z = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2\pi w_x w_y} \exp\left(-\frac{(x-\mathrm{vx}_i)^2}{2w_x^2} - \frac{(y-\mathrm{vy}_i)^2}{2w_y^2}\right)$$ ここで *n* はベクトル *vx* または *vy* の要素の数で、インデックス *i* はベクトル *vx* または *vy* の i 番目の要素、最適スケール(wx, wy)は推定関数 kernel2width で推定されます。
ksdensity(x,vx,w) (2015 SR0)	ベクトル vx のバンド幅 w の x におけるカーネル密度を返します。 $$f(x\|\mathrm{vx},w) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}w} e^{\frac{-(x-\mathrm{vx}_i)^2}{2w^2}}$$ ここで、*n* はベクトル *vx* のサイズ、インデックス i はベクトル *vx* の *i* 番目の要素、最適バンド幅 *w* は推定関数 kernelwidth により決定できます。
lappdf(x,mu,b) (8.6 SR0)	Laplace 確率密度関数 $$f(x\|a,b) = \frac{1}{2b}\exp\left(-\frac{\|x-a\|}{b}\right)$$ $$= \frac{1}{2b}\begin{cases}\exp\left(-\frac{a-x}{b}\right) & \text{if } x < a \\ \exp\left(-\frac{x-a}{b}\right) & \text{if } x \geq a\end{cases}$$
lognpdf(x,mu,sigma) (8.6 SR0)	分布パラメータ mu と sigma を持つ対数正規確率密度関数の X での値を返します。 $$f(x\|\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\frac{-(\ln x-\mu)}{2\sigma^2}}$$
normpdf(x,mu,sigma) (8.6 SR0)	平均値 *mu* と 標準偏差 *sigma* を持つ正規分布を使って X での各値の確率密度関数を計算します。 $$f(x\|\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$
poisspdf(x,lambda) (8.6 SR0)	*lamda* での平均値パラメータを使って、X での各値の Poisson 確率密度関数を計算します。 $$f(x\|\lambda) = \frac{\lambda^x}{x!} e^{-\lambda} I_{(0,1,\ldots)}(x)$$

wblpdf(x,a,b) (8.6 SR0)	パラメータ *a* と *b* を持つワイブル分布の確率密度関数を返します。 $P(x\|a,b) = ba^{-b}x^{b-1}e^{-(\frac{x}{a})^b}$ ワイブル分布データセットからスケールと形状パラメータの *a* と *b* を生成するには、推定関数 gamfit を使用します。

逆累積分布関数(INV)

名前	説明
betainv(p,a,b)	指定したベータ分布の逆累積分布関数を返します。
chi2inv(p,df)	*nu* で指定されたパラメータを持つ *X* での対応する確率に対するカイ二乗累積密度関数の逆数を計算します。 $P(X \le x_p) = p = \frac{1}{2^{\nu/2}\Gamma(\nu/2)}\int_0^{x_p} X^{\nu/2-1}e^{X/2}dX$
finv(p,df1,df2)	パラメータ *df1* と *df2* を持つ *p* における *F* 累積密度関数を計算します。 $f_p = finv(p, df1, df2)$ $P(F \le f_p) = \frac{\nu_1^{\nu_1/2}\nu_2^{\nu_2/2}\Gamma((\nu_1+\nu_2)/2)}{\Gamma(\nu_1/2)\Gamma(\nu_2/2)} \cdot \int_0^{f_p} F^{(\nu_1-2)/2}(\nu_1 F+\nu_2)^{-(\nu_1+\nu_2)/2}dF$ ここで $\nu_1, \nu_2 > 0$; $0 \le f_p < \infty$
gaminv(p,a,b)	パラメータ a と b を持つ p におけるガンマ累積密度関数を計算します。 $P(G \le g_p) = \frac{1}{\beta^\alpha\Gamma(\alpha)}\int_0^{g_p} G^{\alpha-1}e^{-G/\beta}dG$ ここで $0 \le g_p < \infty$; $\alpha, \beta > 0$
logninv(p,mu,sigma) (2015 SR0)	パラメータ *mu* と *sigma* の対数正規分布の下側確率 p に関連づいた偏度 x を計算します。 $p = \int_0^{x_p} \frac{1}{t\sqrt{2\pi}\sigma}e^{-\frac{(ln(t)-\mu)^2}{2\sigma^2}}dt$ where $0 < x_p$
norminv(p)	標準正規分布の与えられた下側確率 p での偏り x を計算します。 $p = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x_p} e^{-u^2/2}du$ where $-\infty < x_p < \infty$

srangeinv(p,v,ir)	スチューデント範囲統計分布の下側確率での偏り x を計算します。 $q = \frac{\max(x_i) - \min(x_i)}{\hat{\sigma}_e}$
tinv(p,df)	自由度を持つスチューデント t 分布の下側確率の偏りを計算します。 $P(T \le t_p) = \frac{\Gamma((\nu+1)/2)}{\sqrt{\pi\nu}\Gamma(\nu/2)} \int_{-\infty}^{t_p} [1 + \frac{T^2}{\nu}]^{-(\nu+1)/2} dT$, $\nu \ge 1$
wblinv(p,a,b)	パラメータ a と b を使って、与えられた確率に対する逆ワイブル累積分布関数を計算します。 $x_p = [aln(\frac{1}{1-p}]^{(\frac{1}{b})} I_{[0,1]}(p)$

21.1.10. データ生成関数

このカテゴリーの関数の内、**rnd()** / **ran()**および **grnd()** の 2 つは、1 つの値を返します。他の関数は領域を返します。

Note：乱数生成の Origin のメソッドのシードアルゴリズムは、バージョン 2016 で変更されました。詳しくは システム変数 @ran のドキュメントをご覧ください。

名前	説明
Data(x1,x2,inc)	**X1** と **x2** の 2 つの値でデータセットを作ります。**x1** から **x2** 範囲で、増分は **inc** とします。**x1** = **x2** の場合、関数は **x1** の値での **inc** 値を返します。デフォルトは **inc** = 1 です。**サンプル**: `col(A) = data(0,100,5)` col(A) = data(1,100)は、列 A を 0 から 100、増分 5 の値で埋めます。 • `col(A) = data(10, 10, 5)` col(A) = data(10, 10, 5)により、A 列の最初から 5 行目までに 10 が割り当てられます。 • `col(A) = data(1,100)` col(A) = data(1,100)は、列 A を 1 から 100、増分 1 の値で埋めます。
grnd( )	関数 grnd()は、平均=0 と標準偏差=1 を持つ、正規（ガウス）分布の乱数の標本値を返します。初期値と値のシーケンスは各 Origin セッションで同じです。引数は必要有りません。一般に、この関数は、式 grnd()*sd+m を使用して、与えられた平均と標準偏差の正規分布からランダムな値を返すために使用されます。**サンプル**: • `aa=grnd()*0.30855+0.45701` %a=errof(book1_b) は、`0.33882089669989` を返します。

normal(npts[,seed])	**npts** の範囲を返します。値は、平均=0、標準偏差=1 の正規(ガウス)分布で与えられる正規乱数です。**seed** が省かれている場合には、この関数が使われる度に異なる seed が使われます。ランダム値、与えられた平均、標準偏差：normal(npts)*sd+m の正規分布の列を埋めるために使用されます。 • `col(1) = normal(100)*2+5` col(1) = normal(100)*2+5 は、1 列目から 100 列目を、平均 5、標準偏差 2 を持つ正規分布の値で埋めます。
pattern(vd, onerepeat, seqrepeat) と pattern(x1,x2,inc,onerepeat,seqrepeat)	パターン化数値またはテキストデータを返します。*pattern(vd, onerepeat, seqrepeat)* は、文字列シリーズ **vd** と **vd** にあるそれぞれの要素を **Onerepeat** かつ、文字列全体を **seqrepeat** 回繰り返します。*Pattern(x1,x2,inc,onerepeat,seqrepeat)* は増加する **x1** から **x2** の範囲でデータセットを生成しあ m す。データセットの個々の要素は、**onerepeat** で繰り返され、データセット全体は **seqrepeat** 回繰り返されます。文字列シリーズにある要素は、pipe (|)、comma(,)、またはスペースや、範囲変数.によって分割されます。**サンプル**： • `col(a)=pattern("Origin Lab", 2, 2);` col(a)=pattern("Origin Lab", 2, 2); は、"Origin Origin Lab Lab Origin Origin Lab Lab」で A 列を埋めます。 • `col(b)=pattern(1,3,1,2,2);` col(b)=pattern(1,3,1,2,2); は"1 1 2 2 3 3 1 1 2 2 3 3".で B 列を埋めます。
Poisson(n, mean [,seed])	平均値 **mean** の Poisson 分布を持つ **n** 個の乱数整数を返します。**seed** は任意で乱数生成のシードを提供します。**サンプル**： • `col(1)=Poisson(100,5,1)` col(1)=Poisson(100,5,1) は、平均 5 の Poisson 分布を持つ 100 この乱数を列 1 に入力します。
ran([seed]) と rnd([seed])	一様分布で与えられる 0 から 1 の間の値を返します。オプション **seed** が正の場合、0 を返します。**seed** ≤ 0 あるいは、引数が与えられていない場合は、乱数系列中の次の数を返します。
uniform(npts [,seed]) と uniform(npts, vd)	**npts** の範囲を返します。オプション **seed** は値、データ範囲、区切り文字列 ("|", "," or space) 、文字列配列です。**seed** が値の場合、連続一様乱数を返します。**seed** がデータ範囲、文字列配列の場合、データ範囲や文字列から選ばれた値が返されます。**seed** が省かれている場合には、この関数が使われる度に異なる seed が使われます。この関数も、アーギュメントとして、ベクトル **vd** を受け入れます。

21.1.11. ルックアップとデータセット情報関数

名前	説明
Findmasks(vd)	マスクされたデータを含む **vd** をとり、マスクされたポイントのインデックスからなるベクトルを返します。**サンプル**: • `dataset aa=findmasks(col(b)); col(d)=aa` dataset aa=findmasks(book1_b); book1_d=aa は、列 D に列 B のマスクされたデータの行インデックスを返します。
Firstpoint(vd)	ベクトル型 **vd** をとり、最大値を返し、データセット **vd** の最初の値を返します。**サンプル**: • `aa = firstpoint(col(A));` 列 A の最初の値を取得し、変数 aa に代入します。
Index(d,vd[,n])	狭義単調データのベクトル **vd** を使うと、ポイント **d** の番号を介します。オプション **n** = 0（初期設定）が同じ、または近似な場合、**d**; n = 1 looks for ≤ **d**; n = 2 looks for ≥ **d**.**vd** が上記でなくテキストも含まない場合は、-2 を返します。**サンプル**: • `index(170,col(1));` index(170,col(1)); は、列 1 で 170 に等しいか、最も近い値のインデックスを返します。
Lastpoint(vd)	ベクトル型 **vd** をとり、最大値を返し、データセット **vd** の最初の値を返します。**サンプル**: • `aa = lastpoint(col(A));` 列 A の最初の値を取得し、変数 aa に代入します。
List(val,vd)	ベクトル **vd** をとり、値 **val** が存在する最初の場所のインデックス番号を返します。相当するものがない場合は、0 を返します。**サンプル**: • `list(3, col(A))` list(3, col(A)) は、列 A を検索し、値 3 が見つかった場所のインデックス番号を返します。
lookup(str$, vs, vref[, option, Case]) と lookup(str$, vs, vref[, option, Case])$ (2015 SR0)	ベクトル **vs** 内で、文字列 **str$**を検索し、同じインデックスのベクトル **vref** の値を返します。**option** により一致の正確性を決めます。**Case** = 0(デフォルト)の場合、関数は大文字小文字の区別をしません。**サンプル**: • `string str1$ = Lookup("FSA", col(A), col(B))$;` string str1$ = Lookup("FSA", col(A), col(B))$; は、列 A で文字列 *FSA* を探し、同じインデックス番号をもつ列 B の値を返します。
table(vd, vref, d[, option]) と table(vd, vref$, d[, option])$	ベクトル **vd** 内の値 **d** を検索し、同じインデックス番号の **vref** 内の値を返します。返却される値は、**vref** によって数値か文字列です。パラメータ

(2015 SR0)	**option** は、パラメータ **d** の検索を補間します。-1 (デフォルト) = **vd** vs **vref** で線形補間を実行し、補間された値を返す; 0 = ≤ **d** の直近の値を検索; 1 = ≥ **d** の直近の値を検索; 2 = 値に等しいまたは直近の値を検索
unique(vs[, sort, occurrence]) (2018b)	ベクトル型 **vd** をとり、固有値を返します。パラメータ sort は、返された固有値を並べ替えるかどうかを決定します。1(既定)= sort ascendingly; 0 =ソートなし; 2 =降順に並べ替えます。occurrence は重複した値を減らす方法を指定します。0(デフォルト)=最初の重複した値のままにします。 1 =最後の重複した値のまま。
Xindex(x,vd[,option])	ベクトル **vd** (Y データセット)をとり、**x** 値に近い **vd** の X データセット内の値のインデックス番号を返します。**option** で、どのインデックスを返すのかを指定します。0 (デフォルト) = 左から等しいか最も近い; 1 = 右から等しいか最も近い; 2 = 左右から等しいか最も近い。必要条件:(1) **vd** は Y 列である; (2) **vd** の名前は実際の Y データセットと一致; (3) X データセットは昇順。**サンプル**: • `xindex(5,book1_g,1)` xindex(5,book1_g,1) は、5 に等しいか右側の X 値の行インデックスを返します。
Xvalue(n,vd)	ベクトル **vd** (Y または Z データセット)をとり、行番号 **n** に対応する X 値を返します。**サンプル**: • `xvalue(20,book4_c)` xvalue(20,book4_c) は、Book4 列 C の 20 番目の行の X 値を返します。
Errof(vd)	データセット **vd** の誤差値を含むデータセット(エラー列)を返します。**サンプル**: • `%a=errof(book1_b)` %a=errof(book1_b) は、`book1_c` を返します。
hasx(vd)	データセット dataset がアクティブレイヤの X データセットに対してプロットされている場合、1 を返します。そうでなければ、0 を返します。**サンプル**: • `aa=hasx(book1_b)` aa=hasx(book1_b) は、アクティブグラフレイヤが列 B のプロットを含む場合 1 を返します。
IsMasked(n,vd)	ベクトル **vd** をとり、**n** = 0 の場合はマスクされたポイント数を返します。**n** = データポイントのインデックス番号の場合、それがマスクされていれば 1、されていなければ 0 を返します。**サンプル**: • `ismasked(0,book1_b)` ismasked(0,book1_b) は、book1_b データセットに 77 個のマスクされた値がある場合、*77* を返します。 • `ismasked(8,book1_b)` ismasked(8,book1_b) n_8 がマスクされていない場合 *0* を返し、マスクされている場合 *1* を返します。

Xof(vd)	X データセットに関連する Y データセットのベクトル名 **vd** をとり、X データセットの名前を含む文字列を返します。**サンプル**: • `%a = xof(book1_b); book1_c = %a; // e.g. after substitution : book1_c = book1_a` %a = xof(book1_b); book1_c = %a; // e.g. 減算後: book1_c = book1_a 列 B の Y データセットの名前を置き、X 値で列 C を埋めます。

21.1.12. データ操作関数

名前	説明
asc(str$)	入力文字列をとり、文字列の最初の文字に対応する ASCII コード(十進法)を返します。この関数は code 関数と同じことをします。**サンプル**: • `aa = asc($100); aa = 36`.を返します。
corr(vx,vy,k[, n])	2 つのデータセット **vx** と **vy**、ラグサイズ **k** をとり、2 つのデータセット間の相関を返します。オプション **n** は、ポイントの数です。ラグパラメータ **k** は、スカラーまたはベクトルです。k がベクトルの場合、関数はベクトルを返し、スカラーの場合はスカラーを返します。**サンプル**: • `corr(col(1),col(2),data(1,10),50)` corr(col(1),col(2),data(1,10),50)は、遅れの大きさ 1 から 10 までを使って求めた、col(1)と col(2)の最初の 50 個のポイントを返します。
peaks(vd, width, minht)	ベクトル **vd** をとり、**width** と **minht** を使用して検索されたピークのインデックスのデータセットを返します。**width** は、テストポイントの両側のポイント数です。**minht** は Y 軸の単位です。インデックス *i* で表される最高値は、*i*-**width** 又は *i*+**width** のデータ値より大きい **minh** となります。**サンプル**: • `peaks(col(B), 3, 0.1)` peaks(col(B), 3, 0.1)は、ピーク番号データセットを返します。
sort(vd)	昇順でソートされたデータセットを返します。**サンプル**: • `%a=sort(book4_c); book4_d=%a` %a=sort(book4_c); book4_d=%a は、列 C をソートし、結果を列 D に入れます。
treplace(vd,val1,val2[, cnd])	条件 cnd が合致すると、データセットを他の値と入れ替えます。データセット **vd** をとり、各値を **cnd** に対して **val1** と比べ、比較が真の場合 **val2** (または -val2) で置き換えます。偽の場合は値を残すか欠損値("--")で埋めます。

21.1.13. NAG 特別関数

エアリー

名前	説明
airy_ai(x)	Airy 関数 Ai(x)の近似を評価します。
airy_ai_deriv(x)	Airy 関数 Ai(x)の微分の近似を評価します。
airy_bi(x)	Airy 関数 Bi(x)の近似を評価します。
airy_bi_deriv(x)	Airy 関数 Bi(x)の微分の近似を評価します。

ベッセル

名前	説明
bessel_i0(x)	Bessel i0。第一種修正ベッセル関数の近似値 I0(x)を求めます。
bessel_k1_scaled(x)	Bessel i0 scaled。式 $e^{-\|x\|}I_0(x)$ の近似値を求めます。
bessel_i1(x)	Bessel i1。第一種修正ベッセル関数 $I_1(x)$ の近似値を求めます。
bessel_k1_scaled(x)	Bessel i1 scaled。式 $e^{-\|x\|}I_1(x)$ の近似値を求めます。
bessel_i_nu(x,n)	Bessel i nu。第一種修正ベッセル関数の近似値 Iν/4 (x) を求めます。
bessel_i_nu_scaled(x,n)	Bessel i nu scaled。第一種修正ベッセル関数 $e^{-x}I_{\frac{\nu}{4}}(x)$ の近似値を求めます。
bessel_j0(x)	Bessel j0。第一種修正ベッセル関数$J_0(x)$を計算します。
bessel_j1(x)	Bessel j1。第一種修正ベッセル関数 $J_1(x)$ の近似値を求めます。

bessel_k0(x)	Bessel k0。第二種修正ベッセル関数,$K_0(x)$ の近似値を求めます。
bessel_k1_scaled(x)	Bessel k0 scaled。式 $e^x K_0(x)$ の近似値を求めます。
Bessel_k1(x)	Bessel k1。第二種修正ベッセル関数,$K_1(x)$ の近似値を求めます。
bessel_k1_scaled(x)	Bessel k1 scaled。式 $e^x K_1(x)$ の近似値を求めます。
bessel_k_nu(x,n)	Bessel k nu。第二種修正ベッセル関数,$K_{v/4}(x)$ の近似値を求めます。
bessel_k_nu_scaled(x,n)	Bessel k nu scaled。第二種修正ベッセル関数,$e^{-x}K_{v/4}(x)$ の近似値を求めます。
bessel_y0(x)	Bessel y0。第二種ベッセル関数 Y_0 ,x > 0 を計算します。近似値は Chebyshev 拡張に基づいています。
bessel_y1(x)	Bessel y1。第二種ベッセル関数 Y_1 ,x > 0 を計算します。近似値は Chebyshev 拡張に基づいています。

誤差

名前	説明
erf(x)	誤差関数 (または正規誤差積分関数)
erfc(x)	誤差関数の補数に対する近似値を計算します。
erfcinv(dy)	指定した y の逆相補誤差関数の値を計算します。
erfcx(x)	スケーリング相補誤差関数
erfinv(dy)	逆誤差関数

ガンマ

名前	説明
gamma(x)	ガンマ関数。式 $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$ を計算します。
incomplete_gamma(a,x)	不完全ガンマ関数。
log_gamma(x)	ログガンマ関数。$\ln \Gamma(x)$ を計算。x > 0。
real_polygamma(x,k)	ポリガンマ関数 psi 関数 $\psi(x)$ の *k* 次派生の近似を計算します。

積分

名前	説明
cos_integral(x)	NAG 余弦積分関数。式 $C_i(x) = y + \ln x + \int_0^x \frac{\cos u - 1}{u}du$ を計算します。
cumul_normal(x)	累積正規分布関数を計算します。
cumul_normal_complem(x)	累積正規分布関数の補数に対する近似値を計算します。
elliptic_integral_rc(x,y)	一種の NAG 楕円積分。積分 $R_c(x,y) = \frac{1}{2}\int_0^\infty \frac{dt}{\sqrt{t+x}(t+y)}$ の近似値を計算します。
elliptic_integral_rd(x,y,z)	二種の NAG 対称楕円積分。積分 $R_D(x,y,z) = \frac{3}{2}\int_0^\infty \frac{dt}{\sqrt{(t+z)(t+y)(t+z)^3}}$ の近似値を計算します。
elliptic_integral_rf(x,y,z)	一種の NAG 対称楕円積分。積分 $R_F(x,y,z) = \frac{1}{2}\int_0^\infty \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}}$ の近似を計算します。

elliptic_integral_rj(x,y,z,r)	三種の NAG 対称楕円積分。積分 $R_J(x,y,z,\rho)=\frac{3}{2}\int_0^\infty \frac{dt}{(t+\rho)\sqrt{(t+x)(t+y)(t+z)}}$ の近似を計算します。
exp_integral(x)	NAG 指数積分関数。式 $E_1(x)=\int_x^\infty \frac{e^{-u}}{u}du, x>0$ を計算します。
fresnel_c(x)	NAG フレネル積分関数 C。フレネル積分 $C(x)=\int_0^x \cos\left(\frac{\pi}{2}t^2\right)dt$ の近似を計算します。
fresnel_s(x)	NAG フレネル積分関数 S。フレネル積分 $S(x)=\int_0^x \sin\left(\frac{\pi}{2}t^2\right)dt$ の近似を計算します。
sin_integral(x)	NAG 正弦積分関数。式 $Si(x)=\int_0^x \frac{\sin u}{u}du$ を計算します。

ケルビン

名前	説明
kelvin_bei(x)	ケルビン関数 bei x の近似を評価します。
kelvin_ber(x)	ケルビン関数 ber x の近似を評価します。
kelvin_kei(x)	ケルビン関数 kei x の近似を評価します。
kelvin_ker(x)	ケルビン関数 ker x の近似を評価します。

その他

名前	説明
jacobian_theta(k,x,q)	NAG ヤコビのテータ関数。独立変数 x と以下の負でない q をヤコビのテータ関数の値を計算します。$\theta_0(x,q)$, $\theta_1(x,q)$, $\theta_2(x,q)$, $\theta_3(x,q)$, $\theta_4(x,q)$
lambertW(x,branch,offset)	ランベルトの W 関数の実数ブランチの近似値を計算します。

21.1.14. フィット関数

このカテゴリの複数のパラメータを持つ関数は、組込関数として、非線形フィットなどに使用されます。**NLFit(解析:フィット:非線形曲線フィット**)を開いて、複数パラメータ関数の数式、サンプル曲線、関数の詳細を確認することができます。その後、関数選択ページから関心のある関数を選択します。

Origin の非線形フィットから利用できる複数パラメータ関数の詳細については、OriginLab ウェブサイトにある PDF ファイルをご覧下さい。このファイルには、各複数パラメータ関数についての数学的な説明、サンプル曲線、パラメータについての解説、LabTalk 関数のシンタックスが含まれています。

Origin Basic Functions

名前	説明
Allometric1(x,a,b)	古典的な Freundlich モデル。相対成長の研究で使われます。 $y = ax^b$
Beta(x,y0,xc,A,w1,w2,w3)	クロマトグラフィと分光法で使われる Beta ピーク関数 $y = y_0 + A\left[1+\left(\frac{w_2+w_3-2}{w_2-1}\right)\left(\frac{x-x_c}{w_1}\right)\right]^{w_2-1} \cdot \left[1-\left(\frac{w_2+w_3-2}{w_3-1}\right)\left(\frac{x-x_c}{w_1}\right)\right]^{w_3-1}$
Boltzmann(x, A1, A2, x0, dx)	Boltzmann 関数 - シグモイド曲線を生成します。 $y = A_2 + \frac{A_1 - A_2}{1+e^{(x-x_0)/dx}}$
dhyperbl(x,P1,P2,P3,P4,P5)	二重直角双曲線関数 $y = \frac{P_1 x}{P_2 + x} + \frac{P_3 x}{P_{4+} x} + P_5 x$
ExpAssoc(x,y0,A1,t1,A2,t2)	2 次の指数関連等式 $y = y_0 + A_1(1-e^{-x/t_1}) + A_2(1-e^{-x/t_2})$

ExpDec1(x,y0,A1,t1)	一定時間パラメータの 1 次指数減少関数 $y = y_0 + Ae^{-x/t}$
ExpDec2(x,y0,A1,t1,A2,t2)	一定時間パラメータの 2 次指数減少関数 $y = y_0 + A_1 e^{-x/t_1} + A_2 e^{-x/t_2}$
ExpDec3(x,y0,A1,t1,A2,t2,A3,t3)	一定時間パラメータの 3 次指数減少関数 $y = y_0 + A_1 e^{-x/t_1} + A_2 e^{-x/t_2} + A_3 e^{-x/t_3}$
ExpGrow1(x,y0,x0,A1,t1)	時間オフセット付き 1 次指数増加 。x0 は修正。 $y = y_0 + A_1 e^{(x-x_0)/t_1}$
ExpGrow2(x, y0, x0, A1, t1, A2, t2)	時間オフセット付き 2 次指数増加 。x0 は修正。 $y = y_0 + A_1 e^{(x-x_0)/t_1} + A_2 e^{(x-x_0)/t_2}$
Gauss(x, y0, xc, w, A)	Gaussian 関数の面積バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 面積) $y = y_0 + \frac{A}{(w\sqrt{\frac{\pi}{2}})} e^{-2(\frac{x-x_c}{w})^2}$
GaussAmp(x,y0,xc,w,A)	ガウスピーク関数の振幅バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 振幅) $y = y_0 + Ae^{-\frac{(x-x_c)^2}{2w^2}}$
Hyperbl(x, P1, P2)	双曲線関数さらに、酵素反応速度論の Michaelis-Menten モデル $y = \frac{P_1 x}{P_2 + x}$
Logistic(x, A1, A2, x0, p)	薬理学/化学でのロジスティック用量応答 $y = A_2 + \frac{A_1 - A_2}{1 + (\frac{x}{x_0})^p}$
LogNormal(x,y0,xc,w,A)	対数が正規分布するランダム変数の確率密度関数 $y = y_0 + \frac{A}{\sqrt{2\pi} w x} e^{\frac{-\left[\ln \frac{x}{x_c}\right]^2}{2w^2}}$
Lorentz(x, y0, xc, w, A)	ベル型で Gaussian 関数より幅広い Lorentzian ピーク関数 (y0 = オフセット, xc = 中心, w = FWHM, A = 面積) $y = y_0 + \frac{2A}{\pi}\left(\frac{w}{4(x - x_c)^2 + w^2}\right)$
Poisson(x,y0,r)	Poisson 確率密度関数。離散確率分布 $y = y_0 + \frac{e^{-r} r^x}{x!}$

Pulse(x, y0, x0, A, t1, P, t2)	Exponential pulse function(x >= x0 ? y : 0). $y = 0 \quad (x < x_0)$ $y = y_0 + A\left(1 - e^{-\frac{x-x_0}{t_1}}\right)^p e^{-\frac{x-x_0}{t_2}} \quad (x \geq x_0)$
Rational0(x,a,b,c)	1 次の分子と 1 次の分母を持つ Rational 関数 $y = \frac{b + cx}{1 + ax}$
Sine(x,y0,xc,w,A)	特定の値の周囲で振動するサイン波関数 $y = y_0 + A\sin\left(\pi\frac{x - x_c}{w}\right)$
Voigt(x,y0,xc,A,wG,wL)	Gaussian 関数と Lorentzian 関数のコンボリューション (y0 = オフセット, xc = 中心, A =面積, wG = Gaussian FWHM, wL = Lorentzian FWHM) $y = y_0 + A\frac{2\ln 2}{\pi^{3/2}}\frac{W_L}{W_G^2}\int_{-\infty}^{\infty}\frac{e^{-t^2}}{\left(\sqrt{\ln 2}\frac{W_L}{W_G}\right)^2 + \left(\sqrt{4\ln 2}\frac{x-x_c}{W_G} - t\right)^2}dt$

Implicit

名前	説明
Circle(x,y,xc,yc,r)	円中心と半径のパラメータの陰円関数 $f = (x - x_c)^2 + (y - y_c)^2 - r^2$
Ellipse(x,y,xc,yc,a,b)	XY 軸と一致する主軸副軸の陰楕円関数 $f = \left(\frac{x - x_c}{a}\right)^2 + \left(\frac{y - y_c}{b}\right)^2 - 1$
ModDiode(V,I,T,Is,Rs,n,Rsh)	暗黙的に修正されたダイオード方程式 $f = I_s\left(e^{\frac{q(V - I\cdot R_s)}{nkT}} - 1\right) + \frac{V - I\cdot R_s}{R_{sh}} - I$
PlaneMod(x,y,z,theta,phi,d)	正規方向により定義された修正陰平面関数 $f = \sin(\theta)\cos(\phi)x + \sin(\theta)\sin(\phi)y + \cos(\theta)z + d$
SolarCellIV(V,I,T,Is,Rs,n,Rsh,IL)	太陽電池の I-V 曲線 $f = I + I_s\left(e^{\frac{q(V + I\cdot R_s)}{nkT}} - 1\right) + \frac{V + I\cdot R_s}{R_{sh}} - I_L$

Exponential

名前	説明
Asymptotic1(x,a,b,c)	Asymptotic 回帰モデル- 1 次パラメータ $y = a - bc^x$
BoxLucas1(x,a,b)	0 オフセットの 1 次結合関数に等しい Box Lucas モデル $y = a(1 - e^{-bx})$
BoxLucas1Mod(x,a,b)	Box Lucas モデルのパラメータ化 $y = a(1 - b^x)$
BoxLucas2(x,a1,a2)	2 次の Box Lucas モデル $y = \frac{a_1}{a_1 - a_2}(e^{-a_2 x} - e^{-a_1 x})$
Chapman(x,a,b,c)	累積成長曲線を説明する Chapman-Richards 関数 $y = a(1 - e^{-bx})^c$
Exp1p1(x,A)	1 つのパラメータの指数関数 $y = e^{x-A}$
Exp1p2(x,A)	1 つのパラメータの指数関数 $y = e^{-Ax}$
Exp1p2Md(x,B)	1 つのパラメータの指数関数 $y = B^x$
Exp1P3(x,A)	1 つのパラメータの指数関数 $y = Ae^{-Ax}$
Exp1P3Md(x,B)	1 つのパラメータの指数関数 $y = -\ln(B)B^x$
Exp1P4(x,A),	1 つのパラメータの漸近指数関数 $y = 1 - e^{-Ax}$
Exp1P4Md(x,B)	1 つのパラメータの漸近指数関数の別の形 $y = 1 - B^x$
Exp2P(x,a,b)	2 つのパラメータの指数関数 $y = ab^x$

Exp2PMod1(x,a,b),	2 つのパラメータの指数関数 $y = ae^{bx}$
Exp2PMod2(x,a,b),	2 つのパラメータの指数関数 $y = e^{a+bx}$
Exp3P1(x,a,b,c),	指数関数の逆オフセット $y = ae^{\frac{b}{x+c}}$
Exp3P1Md(x,a,b,c),	指数関数の逆オフセットの別の形 $y = e^{a+\frac{b}{x+c}}$
Exp3P2(x,a,b,c),	べき指数が 2 次の多項式の指数関数 $y = e^{a+bx+cx^2}$
ExpAssoc(x,y0,A1,t1,A2,t2)	2 次の指数関連等式 $y = y_0 + A_1(1 - e^{-x/t_1}) + A_2(1 - e^{-x/t_2})$
ExpAssoc1(x,TD,Yb,A,Tau) (2017 SR0)	1 の指数関連等式 $y = Yb + A\left(1 - e^{-\frac{(x-TD)}{Tau}}\right)$
ExpAssoc2(x,TD1,TD2,Yb,A1,A2,Tau1,Tau2) (2017 SR0)	Biphasic exponential association equation. $y = \begin{cases} Yb + A_1\left(1 - e^{-\frac{(x-TD_1)}{Tau_1}}\right) & x < TD_2 \\ Yb + A_1\left(1 - e^{-\frac{(x-TD_1)}{Tau_1}}\right) + A_2\left(1 - e^{-\frac{(x-TD_2)}{Tau_2}}\right) & x \geq TD_2 \end{cases}$
ExpAssocDelay1(x,TD,Yb,A,Tau) (2017 SR0)	One-phase exponential association equation with plateau before exponential begins. $y = \begin{cases} Yb & x < TD \\ Yb + A\left(1 - e^{-\frac{(x-TD)}{Tau}}\right) & x \geq TD \end{cases}$
ExpAssocDelay2(x,TD1,TD2,Yb,A1,A2,Tau1,Tau2) (2017 SR0)	指数関数が始まる前のプラトーを用いた二相指数関数方程式 $y = \begin{cases} Yb & x < TD_1 \\ Yb + A_1\left(1 - e^{-\frac{(x-TD_1)}{Tau_1}}\right) & TD_1 \leq x < TD_2 \\ Yb + A_1\left(1 - e^{-\frac{(x-TD_1)}{Tau_1}}\right) + A_2\left(1 - e^{-\frac{(x-TD_2)}{Tau_2}}\right) & x \geq TD_2 \end{cases}$
Exponential(x,y0,A,R0)	一定率パラメータの指数増加関数 $y = y_0 + Ae^{R_0 x}$

ExpDec1(x,y0,A1,t1)	一定時間パラメータの 1 次指数減少関数 $y = y_0 + Ae^{-x/t}$
ExpDec2(x,y0,A1,t1,A2,t2)	一定時間パラメータの 2 次指数減少関数 $y = y_0 + A_1 e^{-x/t_1} + A_2 e^{-x/t_2}$
ExpDec3(x,y0,A1,t1,A2,t2,A3,t3)	一定時間パラメータの 3 次指数減少関数 $y = y_0 + A_1 e^{-x/t_1} + A_2 e^{-x/t_2} + A_3 e^{-x/t_3}$
ExpDecay1(x,y0,x0,A1,t1)	時間オフセット付き 1 次指数減少 。x0 は修正。 $y = y_0 + A_1 e^{-(x-x_0)/t_1}$
ExpDecay2(x, y0, x0, A1, t1, A2, t2)	時間オフセット付き 2 次指数減少 。x0 は修正。 $y = y_0 + A_1 e^{-(x-x_0)/t_1} + A_2 e^{-(x-x_0)/t_2}$
ExpDecay3(x,y0,x0,A1,t1,A2,t2,A3,t3)	時間オフセット付き 3 次指数減少 。x0 は修正。 $y = y_0 + A_1 e^{-(x-x_0)/t_1} + A_2 e^{-(x-x_0)/t_2} + A_3 e^{-(x-x_0)/t_3}$
ExpGro1(x,y0,A1,t1)	一定時間パラメータの 1 次指数増加関数 $y = y_0 + A_1 e^{x/t_1}$
ExpGro2(x,y0,A1,t1,A2,t2)	一定時間パラメータの 2 次指数増加関数 $y = y_0 + A_1 e^{x/t_1} + A_2 e^{x/t_2}$
ExpGro3(x,y0,A1,t1,A2,t2,A3,t3)	時間一定パラメータの 3 次指増加関数 $y = y_0 + A_1 e^{x/t_1} + A_2 e^{x/t_2} + A_3 e^{x/t_3}$
ExpGrow1(x,y0,x0,A1,t1)	時間オフセット付き 1 次指数増加 。x0 は修正。 $y = y_0 + A_1 e^{(x-x_0)/t_1}$
ExpGrow2(x, y0, x0, A1, t1, A2, t2)	時間オフセット付き 2 次指数増加 。x0 は修正。 $y = y_0 + A_1 e^{(x-x_0)/t_1} + A_2 e^{(x-x_0)/t_2}$
ExpGrow3Dec2(x,y0,xc,Ag1,tg1,Ag2,tg2,Ag3,tg3,Ad1,td1,Ad2,td2) (2015 SR0)	3 次増加と 2 次減少の指数関数 $y = \begin{cases} y_0 + A_{d1} + A_{d2} \\ +A_{g1}\left(e^{-x_c/t_{g1}} - e^{-x/t_{g1}}\right) \\ +A_{g2}\left(e^{-x_c/t_{g2}} - e^{-x/t_{g2}}\right) \\ +A_{g3}\left(e^{-x_c/t_{g3}} - e^{-x/t_{g3}}\right) & x \le x_c \\ y_0 + A_{d1}e^{-(x-x_c)/t_{d1}} + A_{d2}e^{-(x-x_c)/t_{d2}} & x > x_c \end{cases}$

ExpGrowDec(x,y0,xc,Ag,tg,Ad,td) (2015 SR0)	1 つの成長と 1 つの減衰を伴う指数関数。 $$y = \begin{cases} y_0 + A_d + A_g\left(e^{-x_c/t_g} - e^{-x/t_g}\right) & x \le x_c \\ y_0 + A_d e^{-(x-x_c)/t_d} & x > x_c \end{cases}$$
ExpLinear(x,p1,p2,p3,p4)	指数と線形の組合せ $$y = p_1 e^{-x/p_2} + p_3 + p_4 x$$
Langevin(x,y0,xc,C)	3 つのパラメータの常磁性で使用する Langevin 関数 $$y = y_0 + C\left(\coth\left(x - x_c\right) - \frac{1}{x - x_c}\right)$$ $$\coth z = \frac{e^z + e^{-z}}{e^z - e^{-z}}$$
LangevinMod(x,y0,xc,C,s) (2015 SR0)	スケール修正 Langevin 関数 $$y = y_0 + C\left(\coth\left(\frac{x - x_c}{s}\right) - \frac{s}{x - x_c}\right)$$ $$coth z = \frac{e^z + e^{-z}}{e^z - e^{-z}}$$
PIPlatt(x,Pm,alpha) (2017 SR0)	Platt による光合成−光曲線モデル $$y = P_m \cdot tanh(alpha \cdot x / P_m)$$
PIPlatt2(x,Ps,alpha,beta) (2017 SR0)	Platt による光阻害のある光合成−光曲線モデル $$y = P_s\left(1 - e^{-\frac{alpha \cdot x}{P_s}}\right)e^{-\frac{beta \cdot x}{P_s}}$$
PIWebb(x,Pm,alpha) (2017 SR0)	Webb による光合成−光曲線モデル $$y = P_m\left(1 - e^{-\frac{alpha \cdot x}{P_m}}\right)$$
MnMolecular(x,A,xc,k),	単分子の成長関数 $$y = A\left(1 - e^{-k(x - xc)}\right)$$
MnMolecular1(x,A1,A2,k)	単分子の成長関数の別の形 $$y = A_1 - A_2 e^{-kx}$$
Shah(x,a,b,c,r)	線形関数を統合した指数減少関数 $$y = a + bx + cr^x$$
Stirling(x,a,b,k)	パラメータとしてゼロにおける傾きの指数増加関数 $$y = a + b\left(\frac{e^{kx} - 1}{k}\right)$$

YldFert(x,a,b,r)	農業での収穫肥料モデル、心理学での学習曲線 $y = a + br^x$
YldFert1(x,a,b,k)	農業での収穫肥料モデル、心理学での学習曲線 $y = a + be^{-kx}$

Growth/Sigmoidal

名前	説明
BiDoseResp(x,A1,A2,LOGx01,LOGx02,h1,h2,p)	二相容量応答関数 $y = A_1 + (A_2 - A_1)\left[\frac{p}{1 + 10^{(LOGx01 - x)h1}} + \frac{1-p}{1 + 10^{(LOGx02 - x)h2}}\right]$
BiHill(x,Pm,Ka,Ki,Ha,Hi) (2015 SR0)	二相ヒル関数 $y = \frac{P_m}{[1 + (\frac{K_a}{x})^{H_a}][1 + (\frac{x}{K_i})^{H_i}]}$
BoltzIV(x,vhalf,dx,gmax,vrev)	IV データの変換 Boltzmann 関数 $y = \frac{(x - vrev) \cdot gmax}{1 + e^{(x - vhalf)/dx}}$
Boltzmann(x, A1, A2, x0, dx)	Boltzmann 関数 - シグモイド曲線を生成します。 $y = A_2 + (A_1 - A_2)/(1 + e^{(x - x0)/dx})$
DoseResp(x,A1,A2,LOGx0,p)	パラメータ'p'によって与えられた変数 Hill 傾きの容量応答曲線 $y = A_1 + \frac{A_2 - A_1}{1 + 10^{(Logx0 - x)p}}$
DoubleBoltzmann(x,y0,A,frac,x01,x02,k1,k2)	二重 Boltzmann 関数。2 つの Boltzmann 関数の合計 $y = y_0 + A\left[\frac{p}{1 + e^{\frac{x - x_{01}}{k_1}}} + \frac{1-p}{1 + e^{\frac{x - x_{02}}{k_2}}}\right]$
Hill(x,Vmax,k,n)	配位子濃度と結合部位の最大数を決定する Hill 関数 $y = V_{\max}\frac{x^n}{k^n + x^n}$
Hill1(x,START,END,k,n)	オフセット付きの修正 Hill 関数 $y = V_{\max}\frac{x^n}{k^n + x^n}$

Logistic(x, A1, A2, x0, p)	薬理学/化学でのロジスティック用量応答 $y = A_2 + \frac{A_1 - A_2}{1 + \left(\frac{x}{x_0}\right)^p}$
Logistic5(x,Amin,Amax,x0,h,s)	5 つのパラメータのロジスティック関数 $y = A_{\min} + \frac{A_{\max} - A_{\min}}{\left(1 + \left(\frac{x}{x_0}\right)^{-h}\right)^s}$
MichaelisMenten(x,Vmax,Km)	酵素動力学の基本モデル。単一基質 Michaelis-Menten 関数。 $y = \frac{V_{max} x}{K_m + x}$
SGompertz(x,a,xc,k)	動物の成長や人口増加の研究での Gompertz 成長モデル $y = a e^{-\exp(-k(x - x_c))}$
Slogistic1(x,a,xc,k)	タイプ 1 のシグモイドロジスティック関数 $y = \frac{a}{1 + e^{-k(x - x_c)}}$
SLogistic2(x,y0,a,Wmax)	タイプ 2 のシグモイドロジスティック関数 $y = \frac{a}{1 + \frac{a - y_0}{y_0} e^{-4W_{\max} x / a}}$
SLogistic3(x,a,b,k)	タイプ 3 のシグモイドロジスティック関数 $y = \frac{a}{1 + b e^{-kx}}$
SRichards1(x,a,xc,d,k)	タイプ 1 のシグモイドリチャード関数 $y = \left[a^{1-d} - e^{-k(x - x_c)}\right]^{1/(1-d)}, d < 1$ $y = \left[a^{1-d} + e^{-k(x - x_c)}\right]^{1/(1-d)}, d > 1$
SRichards2(x,a,xc,d,k)	タイプ 2 のシグモイドリチャード関数 $y = a\left[1 + (d-1) e^{-k(x - x_c)}\right]^{1/(1-d)}, d \neq 1$
SWeibull1(x,A,xc,d,k)	タイプ 1 のシグモイドワイブル関数 $y = A\left(1 - e^{-(k(x - x_c))^d}\right)$
SWeibull2(x,a,b,d,k)	タイプ 2 のシグモイドワイブル関数 $y = A - (A - B) e^{-(kx)^d}$

Hyperbola

名前	説明
Dhyperbl(x,P1,P2,P3,P4,P5)	二重直角双曲線関数 $y = \frac{P_1 x}{P_2 + x} + \frac{P_3 x}{P_{4+} x} + P_5 x$
Hyperbl(x, P1, P2)	双曲線関数さらに、酵素反応速度論の Michaelis-Menten モデル $y = \frac{P_1 x}{P_2 + x}$
HyperbolaGen(x,a,b,c,d)	一般双曲線関数 $y = a - \frac{b}{(1 + cx)^{1/d}}$
HyperbolaMod(x,T1,T2)	修正双曲線関数 $y = \frac{x}{\theta_1 x + \theta_2}$
RectHyperbola(x,a,b)	直角双曲線関数 $y = a\frac{bx}{1 + bx}$

Logarithm

名前	説明
Bradley(x,a,b)	二重対数相互関数 $y = a\ln(-b\ln(x))$
Log2P1(x,a,b),	2 つのパラメータの対数関数 $y = b\ln(x - a)$
Log2P2(x,a,b)	対数変換関数 $y = \ln(a + bx)$
Log3P1(x,a,b,c)	線形対数変換関数 $y = a - b\ln(x + c)$
Logarithm(x,A)	One-parameter logarithm. $y = \ln(x - A)$

Peak Functions

名前	説明
Asym2Sig(x,y0,xc,A,w1,w2,w3)	非対称の 2 重シグモイド $y = y_0 + A\frac{1}{1+e^{-\frac{x-x_c+w_1/2}{w_2}}}\left(1-\frac{1}{1+e^{-\frac{x-x_c-w_1/2}{w_3}}}\right)$
Beta(x,y0,xc,A,w1,w2,w3)	クロマトグラフィと分光法で使われる Beta ピーク関数 $y = y_0 + A\left[1+\left(\frac{w_2+w_3-2}{w_2-1}\right)\left(\frac{x-x_c}{w_1}\right)\right]^{w_2-1} \cdot\left[1-\left(\frac{w_2+w_3-2}{w_3-1}\right)\left(\frac{x-x_c}{w_1}\right)\right]^{w_3-1}$
Bigaussian(x,y0,xc,H,w1,w2)	非対称ピークのフィットに使用する二重 Gaussian ピーク関数 $y = y_0 + He^{-0.5\left(\frac{x-x_c}{w_1}\right)^2} \quad (x < x_c)$ $y = y_0 + He^{-0.5\left(\frac{x-x_c}{w_2}\right)^2} \quad (x \geq x_c)$
CCE(x,y0,xc1,A,w,k2,xc2,B,k3,xc3)	クロマトグラフィで使われる Chesler-Cram ピーク関数 $y = y_0 + A\left[e^{-\frac{(x-x_{c1})^2}{2w}} + B(1-0.5(1-\tanh(k_2(x-x_{c2}))))e^{-0.5k_3(\lvert x-x_{c3}\rvert+(x-x_{c3}))}\right]$
ECS(x,y0,xc,A,w,a3,a4)	クロマトグラフィで使われる Edgeworth-Cramer ピーク関数 $y = y_0 + \frac{A}{w\sqrt{2\pi}}e^{-0.5z^2}\left(1+\frac{a_3}{3!}z\left(z^2-3\right)+\frac{a_4}{4!}(z^4-6z^3+3)+\frac{10a_3^2}{6!}\left(z^6-15z^4+45z^2-15\right)\right)$ $z = \frac{x-x_c}{w}$
Extreme(x,y0,xc,w,A)	特例の Extreme 関数。Gumbel 確率密度関数 $y = y_0 + Ae^{-e^{-z}-z+1}$ $z = \frac{x-x_c}{w}$

Gauss(x, y0, xc, w, A)	Gaussian 関数の面積バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 面積) $y = y_0 + \frac{A}{\left(w\sqrt{\frac{\pi}{2}}\right)} e^{-2\left(\frac{x-x_c}{w}\right)^2}$
GaussAmp(x,y0,xc,w,A)	ガウスピーク関数の振幅バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 振幅) $y = y_0 + Ae^{-\frac{(x-x_c)^2}{2w^2}}$
Gaussian(x,y0,xc,A,w)	Gaussian 関数の全幅半値(FWHM)バージョン (y0 = 基線, xc = 中心, A = 面積, w = FWHM) $y = y_0 + \frac{Ae^{\frac{-4ln(2)(x-x_c)^2}{w^2}}}{w\sqrt{\frac{\pi}{4ln(2)}}}$
GaussMod(x,y0,A,xc,w,t0)	クロマトグラフィで使用される指数修正ガウス(EMG)ピーク関数 $f(x) = y_0 + \frac{A}{t_0} e^{\frac{1}{2}\left(\frac{w}{t_0}\right)^2 - \frac{x-x_c}{t_0}} \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy$ $z = \frac{x-x_c}{w} - \frac{w}{t_0}$
GCAS(x,y0,xc,A,w,a3,a4)	クロマトグラフィで使われる Gram-Charlier ピーク関数 $f(z) = y_0 + \frac{A}{w\sqrt{2\pi}} e^{\frac{-z^2}{2}} \left(1 + \left\lvert \sum_{i=3}^{4} \frac{a_i}{i!} H_i(z) \right\rvert\right)$ $z = \frac{x-x_c}{w}, H_3 = z^3 - 3z, H_4 = z^4 - 6z^3 + 3$
Giddings(x,y0,xc,w,A)	クロマトグラフィで使われる Giddings ピーク関数 $y = y_0 + \frac{A}{w}\sqrt{\frac{x_c}{x}} I_1\left(\frac{2\sqrt{x_c x}}{w}\right) e^{\frac{-x-x_c}{w}}$
InvsPoly(x,y0,xc,w,A,A1,A2,A3)	中心の逆多項式ピーク関数 $y = y_0 + \frac{A}{1 + A_1\left(2\frac{x-x_c}{w}\right)^2 + A_2\left(2\frac{x-x_c}{w}\right)^4 + A_3\left(2\frac{x-x_c}{w}\right)^6}$
Laplace(x,y0,a,b)	Laplace 確率密度関数 $y = y_0 + \frac{1}{2b} e^{-\frac{\lvert x-a\rvert}{b}}$

Logistpk(x,y0,xc,w,A)	ロジスティックピーク関数。Hubbert 関数とも呼ばれる。 $y = y_0 + \frac{4Ae^{-\frac{x-xc}{w}}}{\left(1+e^{-\frac{x-xc}{w}}\right)^2}$
LogNormal(x,y0,xc,w,A)	対数が正規分布するランダム変数の確率密度関数 $y = y_0 + \frac{A}{\sqrt{2\pi}wx} e^{\frac{-\left[\ln\frac{x}{xc}\right]^2}{2w^2}}$
Lorentz(x, y0, xc, w, A)	ベル型で Gaussian 関数より幅広い Lorentzian ピーク関数 (y0 = オフセット, xc = 中心, w = FWHM, A = 面積) $y = y_0 + \frac{2A}{\pi}\left(\frac{w}{4(x-xc)^2+w^2}\right)$
PearsonIV(x,y0,A,m,v,alpha,lam)	負の判別子のためのピアソンのタイプ IV 分布。偏りのある分布にモデルに適合しやすい $y = y_0 + Ak\left[1+\left(\frac{x-\lambda}{\alpha}\right)\right]^{-m} e^{-v\tan^{-1}\left(\frac{x-\lambda}{\alpha}\right)}$ $k = \frac{2^{2m-2}\lvert\Gamma(m+iv/2)\rvert^2}{\pi\alpha\Gamma(2m-1)}, m > \frac{1}{2}$
PearsonVII(x,y0,xc,A,w,m)	ピアソンのタイプ VII ピーク関数 $y = y_0 + A\frac{2\Gamma(\mu)\sqrt{2^{\frac{1}{\mu}}-1}}{\sqrt{\pi}\Gamma\left(\mu-\frac{1}{2}\right)w}\left[1+4\frac{2^{\frac{1}{\mu}}-1}{w^2}\left(x-x_c\right)^2\right]^{-\mu}$ $m = \mu$
PsdVoigt1(x,y0,xc,A,w,mu)	Pseudo-Voigt 関数。Gaussian 関数と Lorentzian 関数の線形結合 (y0 = オフセット, xc = 中心, A = 面積, w = FWHM, mu = プロファイル形要因) $y = y_0 + A\left[m_u\frac{2}{\pi}\frac{w}{4\left(x-x_c\right)^2+w^2} + \left(1-m_u\right)\frac{\sqrt{4\ln 2}}{\sqrt{\pi}w}e^{-\frac{4\ln 2}{w^2}\left(x-x_c\right)^2}\right]$
PsdVoigt2(x,y0,xc,A,wG,wL,mu)	Pseudo-Voigt 関数。異なる FWHM の Gaussian 関数と Lorentzian 関数の線形結合 (y0 = オフセット, xc = 中心, A = 面積, wG=Gaussian FWHM, wL=Lorentzian FWHM, mu = プロファイル形要因) $y = y_0 + A\left[m_u\frac{2}{\pi}\frac{w_L}{4\left(x-x_c\right)^2+w_L^2} + \left(1-m_u\right)\frac{\sqrt{4\ln 2}}{\sqrt{\pi}w_G}e^{-\frac{4\ln 2}{w_G^2}\left(x-x_c\right)^2}\right]$

Voigt(x,y0,xc,A,wG,wL)	Gaussian 関数(wG は FWHM) と Lorentzian 関数のコンボリューション $y = y_0 + A\frac{2\ln 2}{\pi^{3/2}}\frac{W_L}{W_G^2} \cdot \int_{-\infty}^{\infty} \frac{e^{-t^2}}{\left(\sqrt{\ln 2}\frac{W_L}{W_G}\right)^2 + \left(\sqrt{4\ln 2}\frac{x-x_c}{W_G} - t\right)^2} dt$
Weibull3(x,y0,xc,A,w1,w2)	ワイブルピーク関数の振幅バージョン $S = \frac{x - x_c}{w_1} + \left(\frac{w_2 - 1}{w_2}\right)^{\frac{1}{w_2}}$ $y = y_0 + A\left(\frac{w_2 - 1}{w_2}\right)^{\frac{1-w_2}{w_2}} [S]^{w_2 - 1} e^{-[S]^{w_2} + \left(\frac{w_2 - 1}{w_2}\right)}$

Piecewise

名前	説明
PWL2(x,a1,k1,xi1,k2)	2 つの区分の Piecewise 線形関数 $y = a_1 + k_1 x \quad (x < x_i)$ $y = y_i + k_2(x - x_i) \quad (x \geq x_i)$ $y_i = a_1 + k_1 x_i$
PWL3(x,a1,k1,xi1,k2,xi2,k3)	3 つの区分の Piecewise 線形関数 $y = a_1 + k_1 x \quad (x < x_{i_1})$ $y = y_{i_1} + k_2(x - x_{i_1}) \quad (x_{i_1} \leq x < x_{i_2})$ $y = y_{i_2} + k_3(x - x_{i_2}) \quad (x \geq x_{i_2})$ $y_{i_1} = a_1 + k_1 x_{i_1},\ y_{i_2} = y_{i_1} + k_2(x_{i_2} - x_{i_1})$

Polynomial

名前	説明
Constant(x,y0)	定数の線関数 $y = y_0$
Cubic(x,A,B,C,D)	3 次の多項式 $y = A + Bx + Cx^2 + Dx^3$
Line(x,A,B)	傾きと切片付き線形関数

	$y = A + Bx$
LineMod(x,a,b)	パラメータの X 切片と傾き付き線形関数 $y = a\left(x - b\right)$
Parabola(x,A,B,C)	2 次の多項式 $y = A + Bx + Cx^2$
Poly(x, a0, a1, a2, a3, a4, a5, a6, a7, a8, a9)	9 次の多項式 $y = a_0 + a_1x + a_2x^2 + ... + a_9x^9$
Poly4(x,A0,A1,A2,A3,A4)	4 次の多項式関数 $y = A_0 + A_1x + A_2x^2 + A_3x^3 + A_4x^4$
Poly5(x,A0,A1,A2,A3,A4,A5)	5 次の多項式関数 $y = A_0 + A_1x + A_2x^2 + A_3x^3 + A_4x^4 + A_5x^5$

Power

名前	説明
Allometric1(x,a,b)	古典的な Freundlich モデル。相対成長の研究で使われます。 $y = ax^b$
Allometric2(x,a,b,c)	古典的な Freundlich モデルの拡張モデル $y = a + bx^c$
Belehradek(x,a,b,c)	X がシフトしたべき関数 $y = a\left(x - b\right)^c$
BlNeld(x,a,b,c,f)	収量密度モデルの Bleasdale-Nelder 関数 $y = \left(a + bx^f\right)^{-1/c}$
BlNeldSmp(x,a,b,c)	単純化 Bleasdale-Nelder モデル $y = \left(a + bx\right)^{-1/c}$
FarazdaghiHarris(x,a,b,c)	収量密度研究で使われる Farazdaghi-Harris モデル $y = \left(a + bx\right)^{-1/c}$
FreundlichEXT(x,a,b,c)	拡張された Freundlich 吸着等温式 $y = ax^{bx^{-c}}$

Gunary(x,a,b,c)	Gunary 吸着等温式 $y = \frac{x}{a + bx + c\sqrt{x}}$
LangmuirEXT1(x,a,b,c),	拡張された Langmuir 吸着等温式 $y = \frac{abx^{1-c}}{1 + bx^{1-c}}$
LangmuirEXT2(x,a,b,c)	拡張された Langmuir 吸着等温式の別の形 $y = \frac{1}{a + bx^{c-1}}$
Pareto(x,A)	1 つのパラメータの Pareto 累積密度関数。冪乗則確率分布 $y = 1 - \frac{1}{x^A}$
Pow2P1(x,a,b),	スケールされた Pareto 関数 $y = a\left(1 - x^{-b}\right)$
Pow2P2(x,a,b),	2 つのパラメータの power 関数 $y = a\left(1 + x\right)^b$
Pow2P3(x,a,b)	Pareto 変換関数 $y = 1 - \frac{1}{\left(1 + ax\right)^b}$
Power(x,A)	1 つのパラメータの Power 関数 $y = x^A$
Power0(x,y0,xc,A,P)	オフセット付きの対称 Power 関数 $y = y_0 + A\lvert x - x_c\rvert^p$
Power1(x,xc,A,P)	対称 Power 関数 $y = A\lvert x - x_c\rvert^p$
Power2(x,xc,A,pl,pu)	非対称 Power 関数 $y = A\lvert x - x_c\rvert^{pl}, x < x_c$ $y = A\lvert x - x_c\rvert^{pu}, x > x_c$

Rational

名前	説明
BET(x,a,b)	Brunauer-Emmett-Teller (BET) 吸着式 $y = \frac{abx}{1+(b-2)x-(b-1)x^2}$
BETMod(x,a,b)	修正 BET モデル $y = \frac{x}{a+bx-(a+b)x^2}$
Holliday(x,a,b,c)	Holliday モデル - 農業で使われる収穫密度モデル $y = \left(a+bx+cx^2\right)^{-1}$
Holliday1(x,a,b,c)	拡張 Holliday モデル $y = \frac{a}{1+bx+cx^2}$
Nelder(x,a,b0,b1,b2)	Nelder モデル - 農業で使われる収穫-肥料モデル $y = \frac{x+a}{b_0+b_1(x+a)+b_2(x+a)^2}$
Rational0(x,a,b,c)	1 次の分子と 1 次の分母を持つ Rational 関数 $y = \frac{b+cx}{1+ax}$
Rational1(x,a,b,c)	一定係数で標準化した分子の Rational0 関数の別の形 $y = \frac{1+cx}{a+bx}$
Rational2(x,a,b,c)	x の係数で標準化した分母の Rational0 関数の別の形 $y = \frac{b+cx}{a+x}$
Rational3(x,a,b,c)	x 係数で標準化した分子の Rational0 関数の別の形 $y = \frac{b+x}{a+cx}$
Rational4(x,a,b,c)	定数と rational 関数の合計付き Rational0 関数の別の形 $y = c + \frac{b}{x+a}$

Rational5(x,a,b,c,d)	1 次の分子と 2 次の分母を持つ Rational 関数 $y = \frac{a + bx}{1 + cx + dx^2}$
Reciprocal(x,a,b)	2 つのパラメータの線形逆数関数 $y = \frac{1}{a + bx}$
Reciprocal0(x,A)	1 つのパラメータ(傾き)の線形逆数関数 $y = \frac{1}{1 + Ax}$
Reciprocal1(x,A)	1 つのパラメータ(切片)の線形逆数関数 $y = \frac{1}{x + A}$
ReciprocalMod(x,a,b)	一定係数で標準化した分母の Reciprocal 関数の別の形 $y = \frac{a}{1 + bx}$

Waveform

名前	説明
SawtoothWave(x,x0,y0,A,T)	のこぎり波。非対称な三角形の波で構成される周期関数。 $y = y_0 + \frac{A}{T}\left(x - x_0 - nT\right),$ $x_0 + nT \le x < x_0 + \left(n + 1\right)T,\ n \in Z$
Sine(x,y0,xc,w,A)	特定の値の周囲で振動するサイン波関数 $y = y_0 + A\sin\left(\pi\frac{x - x_c}{w}\right)$
SineDamp(x,y0,xc,w,t0,A)	正弦波収束関数。時間経過とともに振幅が減少する正弦波関数 $y = y_0 + Ae^{\frac{-x}{t_0}}\sin\left(\pi\frac{x - x_c}{w}\right)$ $A > 0,\ t_0 > 0,\ w > 0$
SineSqr(x,y0,xc,w,A)	Sin 二乗関数 $y = y_0 + A\sin^2\left(\pi\frac{x - x_c}{w}\right)$

SquareWave(x,a,b,x0, T)	2 つのレベルで周期的に変化する矩形波関数 $y = \begin{cases} a, & x_0 + nT < x < x_0 + \left(n + \frac{1}{2}\right) T,\ n \in Z \\ b, & x_0 + \left(n + \frac{1}{2}\right) T < x < x_0 + (n+1) T,\ n \in Z \end{cases}$
SquareWaveMod(a, b, x0, duty, T) (2016 SR0)	2 つのレベルで周期的に変化する矩形波関数 $y = \begin{cases} a, & x_0 + nT < x < x_0 + (n + duty) T,\ n \in Z, 0 < duty < 1 \\ b, & x_0 + (n + duty) T < x < x_0 + (n+1) T,\ n \in Z, 0 < duty < \end{cases}$
Step(x,A,B,x1)	2 つの区分の Piecewise 定数関数 $y = \begin{cases} A, & x < x_1 \\ B, & x \geq x_1 \end{cases}$

Surface Fitting

名前	説明
Chebyshev2D(x,y,z0,A1,A2,B1,B2,C1)	Chebyshev 系列の多項式 $z = z_0 + A_1 T_1(x) + B_1 T_1(y) + A_2 T_2(x) + C T_1(x) T_1(y) + B_2 T2(y)$ $T_n(x) = \cos(na \cos(x))$ $T_n(y) = \cos(na \cos(y))$ $-1 \leq x \leq 1, \quad -1 \leq y \leq 1$
Cosine(x,y,z0,A1,A2,B1,B2,C1)	Cos 系列の多項式 $z = z_0 + A_1 \cos(x) + B_1 \cos(y) + A_2 \cos(2x) + C_1 \cos(x) \cos(y) + B_2 \cos(2y)$ $0 \leq x \leq \pi, \quad 0 \leq y \leq \pi$
DoseResp2D(x,y,z0,B,C,D,E,F)	非線形ロジスティック用量応答(Dose Response)関数 $z = z_0 + \frac{B}{\left[1 + \left(\frac{x}{C}\right)^{-D}\right]\left[1 + \left(\frac{y}{E}\right)^{-F}\right]}$
Exponential2D(x,y,z0,B,C,D)	2D 指数減少関数 $z = z_0 + B \exp\left(-\frac{x}{C} - \frac{y}{D}\right)$

Extreme2D(x,y,z0,B,C,D,E,F)	非線形極値関数 $$z = z_0 + By + Cy^2 + D\exp\left[1 - \exp\left(\frac{E-x}{F}\right) - \frac{x-E}{F}\right]$$
ExtremeCum(x,y,z0,B,C,D,E,F,G,H)	非線形極値累積関数 $$z = z_0 + B\exp\left\{-\exp\left\{\frac{C-x}{D}\right\}\right\} + E\exp\left\{-\exp\left\{\frac{F-y}{G}\right\}\right\} + H\exp\left\{-\exp\left\{\frac{C-x}{D}\right\} - \exp\left\{\frac{F-y}{G}\right\}\right\}$$
Fourier2D(x,y,z0,a,b,c,d,w1,w2)	2 つの変数の正弦と余弦関数の合計 $$z = z_0 + a\cos\left(\frac{x}{w_1}\right) + b\sin\left(\frac{x}{w_1}\right) + c\cos\left(\frac{y}{w_2}\right) + d\sin\left(\frac{y}{w_2}\right)$$
Gauss2D(x,y,z0,A,xc,w1,yc,w2)	ガウス曲面 $$z = z_0 + A\exp\left\{-\frac{1}{2}\left(\frac{x-x_c}{w_1}\right)^2 - \frac{1}{2}\left(\frac{y-y_c}{w_2}\right)^2\right\}$$
GaussCum(x,y,z0,B,C,D,E,F)	2D ガウス累積関数 $$z = z_0 + 0.25B\left[1 + erf\left\{\frac{x-C}{\sqrt{2}D}\right\}\right]\left[1 + erf\left\{\frac{y-E}{\sqrt{2}F}\right\}\right]$$
Gaussian2D(x,y,z0,A,xc,w1,yc,w2,theta)	回転ガウス曲面 $$z = z_0 + A\exp\left\{-\frac{1}{2}\cdot\left(\frac{x\cos(\theta) + y\sin(\theta) - x_c\cos(\theta) + y_c\sin(\theta)}{w_1}\right)^2 - \frac{1}{2}\left(\frac{-x\sin(\theta) + y\cos(\theta) + x_c\sin(\theta) - y_c\cos(\theta)}{w_2}\right)^2\right\}$$

LogisticCum(x,y,z0,B,C,D,E,F)	非線形シグモイド(ロジスティック累積)関数 $z = z_0 + \frac{B}{\left[1+\exp\left\{\frac{C-x}{D}\right\}\right]\left[1+\exp\left\{\frac{E-y}{F}\right\}\right]}$
LogNormal2D(x,y,z0,B,C,D,E,F,G,H)	2D 対数正規関数 $z = z_0 + B\exp\left\{-\frac{\left(\ln\frac{x}{C}\right)^2}{2D^2}\right\} + E\exp\left\{-\frac{\left(\ln\frac{y}{F}\right)^2}{2G^2}\right\} + H\exp\left\{-\frac{\left(\ln\frac{x}{C}\right)^2}{2D^2} - \frac{\left(\ln\frac{y}{F}\right)^2}{2G^2}\right\}$
Lorentz2D(x,y,z0,A,xc,w1,yc,w2)	非線形シグモイド(ロジスティック累積)関数 $z = z_0 + \frac{A}{\left[1+\left(\frac{x-x_c}{w_1}\right)^2\right]\left[1+\left(\frac{y-y_c}{w_2}\right)^2\right]}$
Parabola2D(x,y,z0,a,b,c,d)	xy 項のない 2D パラボラ関数 $z = z_0 + ax + by + cx^2 + dy^2$
Plane(x,y,z0,a,b)	平面サーフェス $z = z_0 + ax + by$
Poly2D(x,y,z0,a,b,c,d,f)	2D 二次多項式 $z = z_0 + ax + by + cx^2 + dy^2 + fxy$
Polynomial2D(x,y,z0,A1,A2,A3,A4,A5,B1,B2,B3,B4,B5)	交差項なしの 2D 五次多項式 $z = z_0 + A_1x + A_2x^2 + A_3x^3 + A_4x^4 + A_5x^5 + B_1y + B_2y^2 + B_3y^3 + B_4y^4 + B_5y^5$
Power2D(x,y,z0,B,C,D,E,F)	2D Power 関数 $z = z_0 + Bx^C + Dy^E + Fx^Cy^E$
Rational2D(x,y,z0,A01,B01,B02,B03,A1,A2,A3,B1,B2)	3 次の分子と 3 次の分母を持つ 2D Rational 関数 $z = \frac{z_0 + A_{01}x + B_{01}y + B_{02}y^2 + B_{03}y^3}{1 + A_1x + A_2x^2 + A_3x^3 + B_1y + B_2y^2}$
RationalTaylor(x,y,z0,A01,B01,B02,C02,A1,A2,B1,B2,C2)	Taylor 系列の Rational 関数 $z = \frac{z_0 + A_{01}x + B_{01}y + B_{02}y^2 + C_{02}xy}{1 + A_1x + B_1y + A_2x^2 + B_2y^2 + C_2xy}$

Voigt2D(x,y,z0,A,xc,w1,yc,w2,mu)	voigt 曲面 $z = z_0 + A\left[\frac{\mu}{\left[1+\left(\frac{x-x_c}{w_1}\right)^2\right]\left[1+\left(\frac{y-y_c}{w_2}\right)^2\right]} + (1-\mu)\exp\left(-\frac{1}{2}\left(\frac{x-x_c}{w_1}\right)^2 - \frac{1}{2}\left(\frac{y-y_c}{w_2}\right)^2\right)\right]$
Voigt2DMod(x,y,z0,A,xc,w1,yc,w2,mu) (2016 SR0)	パラメータとして体積のある voigt 曲面 $z = z0 + A\left[\frac{mu}{(1+((x-xc)/w1)^2)*(1+((y-yc)/w2)^2)} + (1-mu)*exp(-\frac{1}{2}*(\frac{x-xc}{w1})^2 - \frac{1}{2}*(\frac{y-yc}{w2})^2)\right]$

PFW

名前	説明
Asym2Sig(x,y0,xc,A,w1,w2,w3)	非対称の 2 重シグモイド $y = y_0 + A\frac{1}{1+e^{-\frac{x-x_c+w_1/2}{w_2}}}(1 - \frac{1}{1+e^{-\frac{x-x_c-w_1/2}{w_3}}})$
Bigaussian(x,y0,xc,H,w1,w2)	非対称ピークのフィットに使用する二重 Gaussian ピーク関数 $y = y_0 + H e^{-0.5\left(\frac{x-x_c}{w_1}\right)^2} \quad (x < x_c)$ $y = y_0 + H e^{-0.5\left(\frac{x-x_c}{w_2}\right)^2} \quad (x \geq x_c)$
BWF(x,y0,xc,H,w,q)	Breit-Wigner-Fano (BWF) 線形 $y = y_0 + \frac{H\left(1+\frac{x-x_c}{qw}\right)^2}{1+\left(\frac{x-x_c}{w}\right)^2}$
CCE(x,y0,xc1,A,w,k2,xc2,B,k3,xc3)	クロマトグラフィで使われる Chesler-Cram ピーク関数 $y = y_0 + A[e^{-\frac{(x-x_{c1})^2}{2w}} + B(1-0.5(1-\tanh(k_2(x-x_{c2}))))e^{-0.5k_3(\lvert x-x_{c3}\rvert+(x-x_{c3}))}]$

ConsGaussian(x,y0, xc, A, w1, w2)	制約付き Gaussian 関数 $y = y_0 + \frac{Ae^{-\frac{0.5(x-x_c)^2}{(w_1+w_2x_c)^2}}}{(w_1+w_2x_c)\sqrt{2\pi}}$
DoniachSunjic(x,y0, xc, H, w, a)	Doniach Sunjic 関数 $y = y_0 + \frac{H\cos\left(\frac{a\pi}{2}+(1-a)\arctan\left(\frac{x-x_c}{w}\right)\right)}{\sqrt{\left(w^2+(x-x_c)^2\right)^{(1-a)}}}$
ECS(x,y0,xc,A,w,a3,a4)	クロマトグラフィで使われる Edgeworth-Cramer ピーク関数 $y = y_0 + \frac{A}{w\sqrt{2\pi}}e^{-0.5z^2}\left(1+\frac{a_3}{3!}z\left(z^2-3\right)+\frac{a_4}{4!}(z^4-6z^3+3)+\frac{10a_3^2}{6!}\left(z^6-15z^4+45z^2-15\right)\right)$ $z = \frac{x-x_c}{w}$
FraserSuzuki(x,y0,xc,A,sig)	FraserSuzuki 非対称関数 $y = y_0 + \frac{Ae^{-\frac{(x-x_c)^2}{2sigL}}}{sig\sqrt{2\pi}}\ (x<0)$ $y = y_0 + \frac{Ae^{-\frac{(x-x_c)^2}{2sigR}}}{sig\sqrt{2\pi}}\ (x\geq 0)$ ここで、 $sigL = sig\,(1-A), sigR = sig\,(1+A)$
Gauss(x, y0, xc, w, A)	Gaussian 関数の面積バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 面積) $y = y_0 + \frac{A}{(w\sqrt{\frac{\pi}{2}})}e^{-2(\frac{x-x_c}{w})^2}$
GaussAmp(x,y0,xc,w,A)	ガウスピーク関数の振幅バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 振幅) $y = y_0 + Ae^{-\frac{(x-x_c)^2}{2w^2}}$
Gaussian(x,y0,xc,A,w)	Gaussian 関数の全幅半値(FWHM)バージョン (y0 = 基線, xc = 中心, A = 面積, w = FWHM) $y = y_0 + \frac{Ae^{\frac{-4ln(2)(x-x_c)^2}{w^2}}}{w\sqrt{\frac{\pi}{4ln(2)}}}$
Gaussian_LorenCross(x,y0, xc, A, w, s)	Gaussian-Lorentzian クロス積

	$$y = y_0 + \frac{A}{1 + \frac{e^{\frac{0.5(1-s)(x-x_c)^2}{w}} s(x-x_c)^2}{w^2}}$$
GaussMod(x,y0,A,xc,w,t0)	クロマトグラフィで使用される指数修正ガウス(EMG)ピーク関数 $$f(x) = y_0 + \frac{A}{t_0} e^{\frac{1}{2}\left(\frac{w}{t_0}\right)^2 - \frac{x-x_c}{t_0}} \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy$$ $$z = \frac{x - x_c}{w} - \frac{w}{t_0}$$
GCAS(x,y0,xc,A,w,a3,a4)	クロマトグラフィで使われる Gram-Charlier ピーク関数 $$f(z) = y_0 + \frac{A}{w\sqrt{2\pi}} e^{\frac{-z^2}{2}} \left(1 + \left|\sum_{i=3}^{4} \frac{a_i}{i!} H_i(z)\right|\right)$$ $$z = \frac{x - x_c}{w}, H_3 = z^3 - 3z, H_4 = z^4 - 6z^3 + 3$$
HVL(x, y0, xc, A, w, d)	Haaroff-Van der Linde 関数 $$y = y_0 + \frac{Ae^{-\frac{0.5(x-x_c)^2}{w^2}} w}{d\sqrt{2\pi} x_c \left(e^{1-\frac{dx_c}{w^2}} + 0.5\left(1 + Erf\left(\frac{x-x_c}{w\sqrt{2}}\right)\right)\right)}$$
InvsPoly(x,y0,xc,w,A,A1,A2,A3)	中心の逆多項式ピーク関数 $$y = y_0 + \frac{A}{1 + A_1\left(2\frac{x-x_c}{w}\right)^2 + A_2\left(2\frac{x-x_c}{w}\right)^4 + A_3\left(2\frac{x-x_c}{w}\right)^6}$$
LogNormal(x,y0,xc,w,A)	対数が正規分布するランダム変数の確率密度関数 $$y = y_0 + \frac{A}{\sqrt{2\pi} w x} e^{\frac{-\left[\ln\frac{x}{x_c}\right]^2}{2w^2}}$$
Lorentz(x, y0, xc, w, A)	ベル型で Gaussian 関数より幅広い Lorentzian ピーク関数 (y0 = オフセット, xc = 中心, w = FWHM, A = 面積) $$y = y_0 + \frac{2A}{\pi}\left(\frac{w}{4(x - x_c)^2 + w^2}\right)$$
PearsonVII(x,y0,xc,A,w,m)	ピアソンのタイプ VII ピーク関数 $$y = y_0 + A\frac{2\Gamma(\mu)\sqrt{2^{\frac{1}{\mu}} - 1}}{\sqrt{\pi}\Gamma\left(\mu - \frac{1}{2}\right) w}\left[1 + 4\frac{2^{\frac{1}{\mu}} - 1}{w^2}(x - x_c)^2\right]^{-\mu}$$ $$m = \mu$$

PsdVoigt1(x,y0,xc,A,w,mu)	Pseudo-Voigt 関数。Gaussian 関数と Lorentzian 関数の線形結合 (y0 = オフセット, xc = 中心, A = 面積, w = FWHM, mu = プロファイル形要因) $y = y_0 + A\left[m_u \frac{2}{\pi} \frac{w}{4(x-x_c)^2 + w^2} + (1-m_u)\frac{\sqrt{4\ln 2}}{\sqrt{\pi}w} e^{-\frac{4\ln 2}{w^2}(x-x_c)^2}\right]$
PsdVoigt2(x,y0,xc,A,wG,wL,mu)	Pseudo-Voigt 関数。異なる FWHM の Gaussian 関数と Lorentzian 関数の線形結合 (y0 = オフセット, xc = 中心, A = 面積, wG=Gaussian FWHM, wL=Lorentzian FWHM, mu = プロファイル形要因) $y = y_0 + A\left[m_u \frac{2}{\pi} \frac{w_L}{4(x-x_c)^2 + w_L^2} + (1-m_u)\frac{\sqrt{4\ln 2}}{\sqrt{\pi}w_G} e^{-\frac{4\ln 2}{w_G^2}(x-x_c)^2}\right]$
Pulse(x,y0,x0,A,t1,P,t2)	Exponential pulse function(x >= x0 ? y : 0). $y = 0 \quad (x < x_0)$ $y = y_0 + A\left(1 - e^{-\frac{x-x_0}{t_1}}\right)^p e^{-\frac{x-x_0}{t_2}} \quad (x \geq x_0)$
SchulzFlory(x,y0,xc,w,A)	重合の後ポリマーの相関的比率を説明するための Schulz Flory 分布関数 $y = y_0 + Ae^{\frac{x_c - x}{w}} \left(\frac{x}{x_c}\right)^{\frac{x_c}{w}}$
Sine(x,xc,w,A,y0)	特定の値の周囲で振動するサイン波関数 $y = y_0 + A\sin\left(\pi\frac{x-x_c}{w}\right)$
SineDamp(x,y0,xc,w,t0,A)	正弦波収束関数。時間経過とともに振幅が減少する正弦波関数 $y = y_0 + Ae^{\frac{-x}{t_0}} \sin\left(\pi\frac{x-x_c}{w}\right)$ $A > 0,\ t_0 > 0,\ w > 0$
Sinesqr(x,xc,w,A,y0)	Sin 二乗関数 $y = y_0 + A\sin^2\left(\pi\frac{x-x_c}{w}\right)$

Voigt(x,y0,xc,A,wG,wL)	Gaussian 関数(wG は FWHM) と Lorentzian 関数のコンボリューション (y0 = オフセット, xc = 中心, A =面積, wG = Gaussian FWHM, wL = Lorentzian FWHM) $y = y_0 + A\frac{2\ln 2}{\pi^{3/2}}\frac{W_L}{W_G^2}\int_{-\infty}^{\infty}\frac{e^{-t^2}}{\left(\sqrt{\ln 2}\frac{W_L}{W_G}\right)^2+\left(\sqrt{4\ln 2}\frac{x-x_c}{W_G}-t\right)^2}dt$
Weibull3(x,y0,xc,A,w1,w2)	ワイブルピーク関数の振幅バージョン $S = \frac{x-x_c}{w_1}+\left(\frac{w_2-1}{w_2}\right)^{\frac{1}{w_2}}$ $y = y_0 + A\left(\frac{w_2-1}{w_2}\right)^{\frac{1-w_2}{w_2}}[S]^{w_2-1}e^{-[S]^{w_2}+\left(\frac{w_2-1}{w_2}\right)}$

基線

名前	説明
Constant(x,y0)	定数の線関数 $y = y_0$
Cubic(x,A,B,C,D)	3 次の多項式 $y = A + Bx + Cx^2 + Dx^3$
ExpDec1(x,y0,A1,t1)	一定時間パラメータの 1 次指数減少関数 $y = y_0 + Ae^{-x/t}$
ExpDec2(x,y0,A1,t1,A2,t2)	一定時間パラメータの 2 次指数減少関数 $y = y_0 + A_1e^{-x/t_1} + A_2e^{-x/t_2}$
ExpGro1(x,y0,A1,t1)	一定時間パラメータの 1 次指数増加関数 $y = y_0 + A_1e^{x/t_1}$
ExpGrow1(x,y0,x0,A1,t1)	時間オフセット付き 1 次指数増加 。x0 は修正。 $y = y_0 + A_1e^{(x-x_0)/t_1}$
ExpGrow2(x, y0, x0, A1, t1, A2, t2)	時間オフセット付き 2 次指数増加 。x0 は修正。 $y = y_0 + A_1e^{(x-x_0)/t_1} + A_2e^{(x-x_0)/t_2}$
Exponential(x,y0,A,R0)	一定率パラメータの指数増加関数 $y = y_0 + Ae^{R_0x}$

Hyperbl(x, P1, P2)	双曲線関数さらに、酵素反応速度論の Michaelis-Menten モデル $y = \frac{P_1 x}{P_2 + x}$
Line(x,A,B)	傾きと切片付き線形関数 $y = A + Bx$
MnMolecular(x,A,xc,k),	単分子の成長関数 $y = A\left(1 - e^{-k(x-xc)}\right)$
Parabola(x,A,B,C)	2 次の多項式 $y = A + Bx + Cx^2$
Poly4(x,A0,A1,A2,A3,A4)	4 次の多項式関数 $y = A_0 + A_1 x + A_2 x^2 + A_3 x^3 + A_4 x^4$
Poly5(x,A0,A1,A2,A3,A4,A5)	5 次の多項式関数 $y = A_0 + A_1 x + A_2 x^2 + A_3 x^3 + A_4 x^4 + A_5 x^5$
Step(x,A,B,x1)	2 つの区分の Piecewise 定数関数 $y = \begin{cases} A, & x < x_1 \\ B, & x \geq x_1 \end{cases}$

Chromatograph

名前	説明
CCE(x,y0,xc1,A,w,k2,xc2,B,k3,xc3)	クロマトグラフィで使われる Chesler-Cram ピーク関数 $y = y_0 + A[e^{-\frac{(x-x_{c1})^2}{2w}} + B(1 - 0.5(1 - \tanh(k_2(x - x_{c2}))))e^{-0.5k_3(\lvert x-x_{c3}\rvert+(x-x_{c3}))}]$
ECS(x,y0,xc,A,w,a3,a4)	クロマトグラフィで使われる Edgeworth-Cramer ピーク関数 $y = y_0 + \frac{A}{w\sqrt{2\pi}}e^{-0.5z^2}\left(1 + \frac{a_3}{3!}z\left(z^2 - 3\right) + \frac{a_4}{4!}(z^4 - 6z^3 + 3) + \frac{10a_3^2}{6!}\left(z^6 - 15z^4 + 45z^2 - 15\right)\right)$ $z = \frac{x - x_c}{w}$

Gauss(x, y0, xc, w, A)	Gaussian 関数の面積バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 面積) $y = y_0 + \frac{A}{(w\sqrt{\frac{\pi}{2}})} e^{-2(\frac{x-x_c}{w})^2}$
GaussMod(x,y0,A,xc,w,t0)	クロマトグラフィで使用される指数修正ガウス(EMG)ピーク関数 $f(x) = y_0 + \frac{A}{t_0} e^{\frac{1}{2}\left(\frac{w}{t_0}\right)^2 - \frac{x-x_c}{t_0}} \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy$ $z = \frac{x-x_c}{w} - \frac{w}{t_0}$
GCAS(x,y0,xc,A,w,a3,a4)	クロマトグラフィで使われる Gram-Charlier ピーク関数 $f(z) = y_0 + \frac{A}{w\sqrt{2\pi}} e^{\frac{-z^2}{2}} \left(1 + \left\lvert \sum_{i=3}^{4} \frac{a_i}{i!} H_i(z) \right\rvert\right)$ $z = \frac{x-x_c}{w}, H_3 = z^3 - 3z, H_4 = z^4 - 6z^3 + 3$
Giddings(x,y0,xc,w,A)	クロマトグラフィで使われる Giddings ピーク関数 $y = y_0 + \frac{A}{w}\sqrt{\frac{x_c}{x}} I_1\left(\frac{2\sqrt{x_c x}}{w}\right) e^{\frac{-x-x_c}{w}}$

Electrophysiology

名前	説明
BoltzIV(x,vhalf,dx,gmax,vrev)	IV データの変換 Boltzmann 関数 $y = \frac{(x - vrev) \cdot gmax}{1 + e^{(x-vhalf)/dx}}$
Boltzmann(x, A1, A2, x0, dx)	Boltzmann 関数 - シグモイド曲線を生成します。 $y = A_2 + (A_1 - A_2)/(1 + e^{(x-x0)/dx})$
DoubleBoltzmann(x,y0,A,frac,x01,x02,k1,k2)	二重 Boltzmann 関数。2 つの Boltzmann 関数の合計 $y = y_0 + A\left[\frac{p}{1 + e^{\frac{x-x_{01}}{k_1}}} + \frac{1-p}{1 + e^{\frac{x-x_{02}}{k_2}}}\right]$
ExpDec1(x,y0,A1,t1)	一定時間パラメータの 1 次指数減少関数 $y = y_0 + Ae^{-x/t}$
ExpDec2(x,y0,A1,t1,A2,t2)	一定時間パラメータの 2 次指数減少関数 $y = y_0 + A_1 e^{-x/t_1} + A_2 e^{-x/t_2}$

ExpDec3(x,y0,A1,t1,A2,t2,A3,t3)	一定時間パラメータの 3 次指数減少関数 $y = y_0 + A_1 e^{-x/t_1} + A_2 e^{-x/t_2} + A_3 e^{-x/t_3}$
Gauss(x, y0, xc, w, A)	Gaussian 関数の面積バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 面積) $y = y_0 + \frac{A}{(w\sqrt{\frac{\pi}{2}})} e^{-2(\frac{x-x_c}{w})^2}$
Goldman(x,b,Nao,Nai,Ki,T)	電気生理学の Goldman-Hodgkin-Katz の式 $E = \frac{RT}{F} \ln\left(\frac{[K]_0 + b[Na]_0}{[K]_i + b[Na]_i}\right)$ $x = [K]_0,\ y = E$ $R = 8.314, F = 96485, b = P_k/P_{Na}$
Hill(x,Vmax,k,n)	配位子濃度と結合部位の最大数を決定する Hill 関数 $y = V_{\max}\frac{x^n}{k^n + x^n}$

Pharmacology

名前	説明
BiDoseResp(x,A1,A2,LOGx01,LOGx02,h1,h2,p)	二相容量応答関数 $y = A_1 + (A_2 - A_1)\left[\frac{p}{1 + 10^{(LOGx01-x)h1}} + \frac{1-p}{1 + 10^{(LOGx02-x)h2}}\right]$
Biphasic(x,Amin,Amax1,Amax2,x0_1,x0_2,h1,h2)	二相シグモイド用量反応(7 つのパラメータのロジスティック方程式) $y = A_{\min} + \frac{(A_{\max 1} - A_{\min})}{1 + 10^{(x-x0\_1)h_1}} + \frac{(A_{\max 2} - A_{\min})}{1 + 10^{(x0\_2-x)h_2}}$
DoseResp(x,A1,A2,LOGx0,p)	パラメータ'p'によって与えられた変数 Hill 傾きの容量応答曲線 $y = A1 + \frac{A2 - A1}{1 + 10^{(Logx0-x)p}}$
MichaelisMenten(x,Vmax,Km)	酵素動力学の基本モデル。単一基質 Michaelis-Menten 関数。 $y = \frac{V_{max}x}{K_m + x}$

OneSiteBind(x,Bmax,k1)	片側直接結合。直角双曲線が等温線や飽和率曲線に接続します。 $$y = \frac{B_{\max} x}{K_1 + x}$$
OneSiteComp(x,A1,A2,logx0)	1 箇所競争曲線。-1 の Hill 勾配付きの用量応答曲線。 $$y = A_2 + \frac{A_1 - A_2}{1 + 10^{(x - \log x_0)}}$$
TwoSiteBind(x,Bmax1,Bmax2,k1,k2)	両側結合関数 $$y = \frac{B_{\max 1} x}{k_1 + x} + \frac{B_{\max 2} x}{k_2 + x}$$
TwoSiteComp(x,A1,A2,logx0_1,logx0_2,fraction)	2 種の受容器のための配位子の競合を説明する両側競合関数 $$y = A_2 + \frac{(A_1 - A_2)\, f}{1 + 10^{(x - \log x_{0\_1})}} + \frac{(A_1 - A_2)\,(1 - f)}{1 + 10^{(x - \log x_{0\_2})}}$$

Rheology

名前	説明
Bingham(x,y0,A) (2015 SR0)	Bingham モデル $$y = y_0 + Ax$$
Cross(x,A1,A2,t,m) (2015 SR0)	Cross モデル $$y = A_2 + \frac{A_1 - A_2}{1 + (tx)^m}$$
Carreau(x,A1,A2,t,a,n) (2015 SR0)	Carreau-Yasuda モデル $$y = A_2 + (A_1 - A_2)[1 + (tx)^a]^{\frac{n-1}{a}}$$
Herschel(x,y0,K,n) (2015 SR0)	Herschel-Bulkley モデル $$y = y_0 + Kx^n$$
VFT(x,A,B,x0) (2015 SR0)	Vogel-Fulcher-Tammann の式 $$\log_{10} y = A + \frac{B}{x - x_0}$$
MYEGA(x,y0,K,C) (2015 SR0)	Mauro-Yue-Ellison-Gupta-Allan の式 $$\log_{10} y = \log_{10} y_0 + \frac{K}{x} e^{C/x}$$

Enzyme Kinetics

名前	説明
CompInhib(x,Vmax,Km,Ki,Ic) (2015 SR0)	1 つの基盤と 1 つの阻害の競合阻害モデル $y = \frac{V_{\max} x}{K_{\mathrm{m}}(1 + \frac{I_{\mathrm{c}}}{K_{\mathrm{i}}}) + x}$ この関数は通常グローバルフィットで使用され、各データセットにおいて Vmax, Km, Ki は共有、Ic は固定です。Ki の初期値は Ic の平均にすることができます。
NoncompInhib(x,Vmax,Km,Ki,Ic) (2015 SR0)	1 つの基盤と 1 つの阻害の非競合阻害モデル $y = \frac{V_{\max} x}{(1 + \frac{I_{\mathrm{c}}}{K_{\mathrm{i}}})(K_{\mathrm{m}} + x)}$ この関数は通常グローバルフィットで使用され、各データセットにおいて Vmax, Km, Ki は共有、Ic は固定です。Ki の初期値は Ic の平均にすることができます。
UncompInhib(x,Vmax,Km,Kia,Ic) (2015 SR0)	1 つの基盤と 1 つの阻害の不競合阻害モデル $y = \frac{V_{\max} x}{(1 + \frac{I_{\mathrm{c}}}{K_{\mathrm{ia}}})(\frac{K_{\mathrm{m}}}{1 + \frac{I_{\mathrm{c}}}{K_{\mathrm{ia}}}} + x)}$ この関数は通常グローバルフィットで使用され、各データセットにおいて Vmax, Km, Ki は共有、Ic は固定です。Kia の初期値は Ic の平均にすることができます。
MixedModelInhib(x,Vmax,Km,Ki,Alpha,Ic) (2015 SR0)	特殊なケースとして、競合、不競合および非競合阻害を含む一般的な方程式。 $y = \frac{\frac{V_{\max} x}{1 + I_{\mathrm{c}}/(Alpha \cdot K_{\mathrm{i}})}}{x + \frac{K_m (1 + I_{\mathrm{c}}/K_{\mathrm{i}})}{1 + I_{\mathrm{c}}/(Alpha \cdot K_{\mathrm{i}})}}$ この関数は通常グローバルフィットで使用されます。各データセットにおいて Vmax, Km, Ki は共有、Ic は固定です。Ki の初期値は Ic の平均にすることができます。
SubstrateInhib(x,Vmax,Km,Ki) (2015 SR0)	高濃度の基質阻害モデル $y = \frac{V_{max} x}{K_m + x(1 + x/K_i)}$
MichaelisMenten(x,Vmax,Km)	酵素動力学の基本モデル。単一基質 Michaelis-Menten 関数。 $y = \frac{V_{max} x}{K_m + x}$
Hill(x,Vmax,k,n)	配位子濃度と結合部位の最大数を決定する Hill 関数 $y = V_{\max} \frac{x^n}{k^n + x^n}$

Spectroscopy

名前	説明
GaussAmp(x,y0,xc,w,A)	ガウスピーク関数の振幅バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 振幅) $y = y_0 + Ae^{-\frac{(x-x_c)^2}{2w^2}}$
InvsPoly(x,y0,xc,w,A,A1,A2,A3)	中心の逆多項式ピーク関数 $y = y_0 + \frac{A}{1 + A_1\left(2\frac{x-x_c}{w}\right)^2 + A_2\left(2\frac{x-x_c}{w}\right)^4 + A_3\left(2\frac{x-x_c}{w}\right)^6}$
Lorentz(x, y0, xc, w, A)	ベル型で Gaussian 関数より幅広い Lorentzian ピーク関数 (y0 = オフセット, xc = 中心, w = FWHM, A = 面積) $y = y_0 + \frac{2A}{\pi}\left(\frac{w}{4(x-x_c)^2 + w^2}\right)$
PearsonVII(x,y0,xc,A,w,m)	ピアソンのタイプ VII ピーク関数 $y = y_0 + A\frac{2\Gamma(\mu)\sqrt{2^{\frac{1}{\mu}} - 1}}{\sqrt{\pi}\Gamma\left(\mu - \frac{1}{2}\right)w}\left[1 + 4\frac{2^{\frac{1}{\mu}} - 1}{w^2}(x - x_c)^2\right]^{-\mu}$ $m = \mu$
PsdVoigt1(x,y0,xc,A,w,mu)	Pseudo-Voigt 関数。Gaussian 関数と Lorentzian 関数の線形結合 (y0 = オフセット, xc = 中心, A = 面積, w = FWHM, mu = プロファイル形要因) $y = y_0 + A\left[m_u\frac{2}{\pi}\frac{w}{4(x-x_c)^2 + w^2} + (1 - m_u)\frac{\sqrt{4\ln 2}}{\sqrt{\pi}w}e^{-\frac{4\ln 2}{w^2}(x-x_c)^2}\right]$
PsdVoigt2(x,y0,xc,A,wG,wL,mu)	Pseudo-Voigt 関数。異なる FWHM の Gaussian 関数と Lorentzian 関数の線形結合 (y0 = オフセット, xc = 中心, A = 面積, wG=Gaussian FWHM, wL=Lorentzian FWHM, mu = プロファイル形要因) $y = y_0 + A\left[m_u\frac{2}{\pi}\frac{w_L}{4(x-x_c)^2 + w_L^2} + (1 - m_u)\frac{\sqrt{4\ln 2}}{\sqrt{\pi}w_G}e^{-\frac{4\ln 2}{w_G^2}(x-x_c)^2}\right]$

Voigt(x,y0,xc,A,wG,wL)	Gaussian 関数(wG は FWHM) と Lorentzian 関数のコンボリューション $y = y_0 + A\frac{2\ln 2}{\pi^{3/2}}\frac{W_L}{W_G^2} \cdot \int_{-\infty}^{\infty} \frac{e^{-t^2}}{\left(\sqrt{\ln 2}\frac{W_L}{W_G}\right)^2 + \left(\sqrt{4\ln 2}\frac{x-x_c}{W_G} - t\right)^2} dt$

統計

名前	説明
Exponential(x,y0,A,R0)	一定率パラメータの指数増加関数 $y = y_0 + Ae^{R_0 x}$
ExponentialCDF(x,y0,A,mu) (2016 SR0)	Exponential 累積分布関数 $y = \begin{cases} y_0 + A\int_0^x \frac{1}{\mu} e^{-\frac{x}{\mu}} dt \\ = y_0 + A\left(1 - e^{-\frac{x}{\mu}}\right) & x \geq 0 \\ y_0 & x < 0 \end{cases}$
Extreme(x,y0,xc,w,A)	特例の Extreme 関数。Gumbel 確率密度関数 $y = y_0 + Ae^{-e^{-z}-z+1}$ $z = \frac{x - x_c}{w}$
GammaCDF(x,y0,A1,a,b) (2016 SR0)	Gamma 累積分布関数 $y = y_0 + \frac{A_1}{b^a \Gamma(a)} \int_0^x t^{a-1} e^{-\frac{t}{b}} dt \quad x > 0$
Gauss(x, y0, xc, w, A)	Gaussian 関数の面積バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 面積) $y = y_0 + \frac{A}{\left(w\sqrt{\frac{\pi}{2}}\right)} e^{-2\left(\frac{x-x_c}{w}\right)^2}$
GaussAmp(x,y0,xc,w,A)	ガウスピーク関数の振幅バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 振幅) $y = y_0 + Ae^{-\frac{(x-x_c)^2}{2w^2}}$
Gumbel(x,a,b)	変換 Gumbel 累積分布関数 $y = 1 - e^{-e^{\frac{x-a}{b}}}$

Laplace(x,y0,a,b)	Laplace 確率密度関数 $y = y_0 + \frac{1}{2b} e^{-\frac{\lvert x-a \rvert}{b}}$
Logistic(x, A1, A2, x0, p)	薬理学/化学でのロジスティック用量応答 $y = A_2 + \frac{A_1 - A_2}{1 + (\frac{x}{x_0})^p}$
LogNormal(x,y0,xc,w,A)	対数が正規分布するランダム変数の確率密度関数 $y = y_0 + \frac{A}{\sqrt{2\pi} w x} e^{\frac{-\left[\ln \frac{x}{x_C}\right]^2}{2w^2}}$
LognormalCDF(x,y0,A,xc,w) (2016 SR0)	LognormalCDF 累積分布関数 $y = y_0 + A \int_0^x \frac{1}{\sqrt{2\pi} w t} e^{-\frac{(ln(t)-x_C)^2}{2w^2}} dt$
Lorentz(x, y0, xc, w, A)	ベル型で Gaussian 関数より幅広い Lorentzian ピーク関数 (y0 = オフセット, xc = 中心, w = FWHM, A = 面積) $y = y_0 + \frac{2A}{\pi} \left(\frac{w}{4(x - x_c)^2 + w^2} \right)$
NormalCDF(x,y0,A,xc,w)	通常の累積分布関数 (y0 = オフセット, A = 振幅, xc = 平均, w = 標準偏差) $y = y_0 + A \int_{-x}^x \frac{1}{\sqrt{2\pi} w} e^{-\frac{(t-x_C)^2}{2w^2}} dt$
Pareto(x,A)	1 つのパラメータの Pareto 累積密度関数。冪乗則確率分布 $y = 1 - \frac{1}{x^A}$
Pareto2(x,a,b)	2 つのパラメータの Pareto 関数 $y = 1 - \left(\frac{b}{x} \right)^a$
PearsonIV(x,y0,A,m,v,alpha,lam)	負の判別子のためのピアソンのタイプ IV 分布。偏りのある分布にモデルに適合しやすい $y = y_0 + Ak \left[1 + \left(\frac{x - \lambda}{\alpha} \right) \right]^{-m} e^{-v \tan^{-1}\left(\frac{x-\lambda}{\alpha}\right)}$ $k = \frac{2^{2m-2} \lvert \Gamma(m + iv/2) \rvert^2}{\pi \alpha \Gamma(2m-1)}, m > \frac{1}{2}$
Poisson(x,y0,r)	Poisson 確率密度関数。離散確率分布 $y = y_0 + \frac{e^{-r} r^x}{x!}$

Rayleigh(x,b)	Rayleigh 累積分布関数 $y = 1 - e^{-\frac{x^2}{2b^2}}$
Weibull(x,y0,a,r,u)	Weibull 確率密度関数 $y = y_0 + \frac{b}{a}\left(\frac{x-c}{a}\right)^{b-1} \exp\left\{-\left(\frac{x-c}{a}\right)^{b}\right\}$
WeibullCDF(x,y0,A1,a,b) (2016 SR0)	Weibull 累積分布関数 $y = \begin{cases} y_0 + A_1 \int_0^x b a^{-b} t^{b-1} e^{-\left(\frac{t}{a}\right)^b} dt \\ = y_0 + A_1\left(1 - e^{-\left(\frac{x}{a}\right)^b}\right) & x > 0 \\ y_0 & x \le 0 \end{cases}$

Quick Fit

名前	説明
Boltzmann(x, A1, A2, x0, dx)	Boltzmann 関数 - シグモイド曲線を生成します。 $y = A_2 + (A_1 - A_2)/(1 + e^{(x-x_0)/dx})$
DoseResp(x,A1,A2,LOGx0,p)	パラメータ'p'によって与えられた変数 Hill 傾きの容量応答曲線 $y = A_1 + \frac{A_2 - A_1}{1 + 10^{(Logx0 - x)p}}$
ExpDecay1(x,y0,x0,A1,t1)	時間オフセット付き 1 次指数減少 。x0 は修正。 $y = y_0 + A_1 e^{-(x-x_0)/t_1}$
ExpGrow1(x,y0,x0,A1,t1)	時間オフセット付き 1 次指数増加 。x0 は修正。 $y = y_0 + A_1 e^{(x-x_0)/t_1}$
Gauss(x, y0, xc, w, A)	Gaussian 関数の面積バージョン (y0 = オフセット, xc = 中心, w = 幅, A = 面積) $y = y_0 + \frac{A}{\left(w\sqrt{\frac{\pi}{2}}\right)} e^{-2\left(\frac{x-xc}{w}\right)^2}$
Hill(x,Vmax,k,n)	配位子濃度と結合部位の最大数を決定する Hill 関数 $y = V_{\max}\frac{x^n}{k^n + x^n}$
Hyperbl(x, P1, P2)	双曲線関数さらに、酵素反応速度論の Michaelis-Menten モデル $y = \frac{P_1 x}{P_2 + x}$

Logistic(x, A1, A2, x0, p)	薬理学/化学でのロジスティック用量応答 $$y = A_2 + \frac{A_1 - A_2}{1 + \left(\frac{x}{x_0}\right)^p}$$
Lorentz(x, y0, xc, w, A)	ベル型で Gaussian 関数より幅広い Lorentzian ピーク関数 (y0 = オフセット, xc = 中心, w = FWHM, A = 面積) $$y = y_0 + \frac{2A}{\pi}\left(\frac{w}{4(x - x_c)^2 + w^2}\right)$$
Sine(x,y0,xc,w,A)	特定の値の周囲で振動するサイン波関数 $$y = y_0 + A\sin\left(\pi\frac{x - x_c}{w}\right)$$
Voigt(x,y0,xc,A,wG,wL)	Gaussian 関数(wG は FWHM) と Lorentzian 関数のコンボリューション $$y = y_0 + A\frac{2\ln 2}{\pi^{3/2}}\frac{W_L}{W_G^2} \cdot \int_{-\infty}^{\infty} \frac{e^{-t^2}}{\left(\sqrt{\ln 2}\frac{W_L}{W_G}\right)^2 + \left(\sqrt{4\ln 2}\frac{x - x_c}{W_G} - t\right)^2} dt$$

複数の変数

名前	説明
GaussianLorentz(y0, xc, A1, A2, w1, w2)	1 つの独立変数と 2 つの従属変数、共有パラメータ $$y_1 = y_0 + \frac{A_1}{w_1\sqrt{\frac{\pi}{2}}}e^{-2\left(\frac{x - x_c}{w_1}\right)^2}$$ $$y_2 = y_0 + 2\frac{A_2}{\pi}\left(\frac{w_2}{4(x - x_c)^2 + w_2^2}\right)$$
Helix(x,x0,y0,A,w,p)	3D Helix Function $$x = A\cos(wz + p) + x_0$$ $$y = A\sin(wz + p) + y_0$$
HillBurk(Vm1, Km1, Vm2, Km2)	2 つの独立した変数と 2 つの従属変数を持つ Hill と Burk モデルの組み合わせ。 $$v_1 = V_{m1}\frac{x_1}{k_{m1} + x_1}$$ $$v_2 = \frac{1 + K_{m2}x_2}{V_{m2}}$$

Line3(a, b, c, d)	傾斜と切片を持つ 3D Line 関数 $y = a + bx$ $z = c + dx$
LineExp(x,Vmax,k,n)	1 つの独立変数と 2 つの従属変数を持つ線と指数モデルの組み合わせ。 $y_1 = a_1 + b_1 x$ $y_2 = a_2 + b_2 e^{\frac{x}{c}}$

21.1.15. その他

名前	説明
BitAND(n1, n2)	2 つの整数のビット積を返します。
BitOR(n1, n2)	2 つの整数のビット和を返します。
BitXOR(n1, n2)	2 つの整数のビット排他的和を返します。
ISNA(d)	数字が NANUM かどうか判定します。
isText(str$) (2019)	値がテキストかどうかを判定します。テキストの場合 1 または空欄、数値の場合 NANUM を返します。
NA()	NANUM を返します。
ocolor2rgb(oColor) (2019 SR0)	内部カラーコード **oColor** を RGB 値に変換します。
xf_get_last_error_code()	X ファンクションエンジンの最後のエラーコードの値を取得します。
xf_get_last_error_message()$	X ファンクションエンジンの最後のエラーメッセージを取得します。
xor(n1,n2) (2019 SR0)	2 つの論理値 **n1** と **n2** の XOR 演算を返します。**サンプル** • `XOR(1>0,7>2)` 両方が TRUE なので 0 を返します。 • `XOR(1<0,7<2)` 両方が FALSE なので 0 を返します。 • `XOR(1>0,7<2)` 1 番目が TRUE で 2 番目が FALSE なので 1 を返します

21.1.16. 工業

名前	説明
Base(num,radix[,len])$ (2019 SR0)	与えられた整数 **num** を指定された基数の文字列表現に変換します。 **radix**.
Bin2Dec(str$)	2 進数を 10 進数に変換します。
BitLShift(num,shift) (2019 SR0)	指定されたビット数だけシフトした 10 進数のシフト
BitRShift(num,shift) (2019 SR0)	指定されたビット数だけシフトした 10 進数のシフト
Convert(d,str1$,str2$)	ある測定系から別の測定系に数値を変換します。
Decimal(text$,radix)	指定された**基数**の文字列表現**テキスト** を 10 進数に変換します。 10 進数
Dec2Bin(n)$	10 進数を 2 進数に変換します。入力範囲は-512 から 511 に制限されます。
Dec2Hex(n[,places])$	10 進数を 16 進数に変換し、オプションで文字数を指定します。
Hex2Dec(str$)	16 進数の文字列表現を 10 進数に変換します。

21.1.17. 複素数

名前	説明
Imabs(c)	複素数の絶対値を取得します。
Imaginary(c)	この関数は、複素数の虚部を取得します。
Imargument(c)	複素数の引数(シータ)を求めます。
Imatanh(c) (2016 SR0)	その複素数の逆タンジェントを返します。
Imatanh(c) (2016 SR0)	その複素数の逆双曲線正接を返します。
Imconjugate(c)	複素数の共役を取得します。

lmcos(c)	複素数の cos 値を計算します。 $ImCos(C) = \frac{e^{iC} + e^{-iC}}{2}$ ここで C は複素数で、$i = \sqrt{-1}$
lmcosh(c) (2019 SR0)	与えられた複素数の双曲線コサインを計算します。
lmcot(c) (2019 SR0)	与えられた複素数のコタンジェントを計算する。
lmcsc(c) (2019 SR0)	与えられた複素数のコセカントを計算します。
lmcsch(c) (2019 SR0)	与えられた複素数の双曲線コセカントを計算します。
lmsec(c) (2019 SR0)	与えられた複素数のセカントを計算します。
lmsech(c) (2019 SR0)	与えられた複素数の双曲線セカントを計算します。
lmsinh(c) (2019 SR0)	与えられた複素数の双曲線サインを計算します。
lmtan(c) (2019 SR0)	与えられた複素数のタンジェントを計算する。
lmdiv(c1,c2)	複素数除算を行います。
lmexp(c)	複素数の指数値を計算します。 $ImExp\,(x+iy) = e^x * (cos(y) + sin(y) * i)$ ここで$i = \sqrt{-1}$です。
lmln(c)	複素数の自然対数を計算します。 $ImLn\,(x+iy) = Ln(ImAbs(x+iy)) + i * atan2(y,x)$ ここで、lmAbs は、複素数の絶対値を計算します。
lmlog10(c)	複素数の底 10 の対数を計算します。 $ImLog10\,(x+iy) = \frac{ImLn(x+iy)}{ImLn(10)}$ ここで、lmLn は、複素数の自然定数を計算し、$i = \sqrt{-1}$です。

Imlog2(c)	複素数の底 2 の対数を計算します。 $ImLog2\left(x+iy\right)=\frac{ImLn(x+iy)}{ImLn(2)}$ ここで、ImLn は、複素数の自然定数を計算し、$i=\sqrt{-1}$です。
ImPower(c,d)	複素数に指定したべき乗を行います。
Improduct(c1,c2)	2 つの複素数の積を求めます(乗算)。 $ImProduct(x1+i*y1,x2+i*y2)$ $=(x1*x2-y1*y2)+i*(x1*y2+x2*y1)$
ImReal(c)	指定した複素数の実数部を取得します。
Imsin(c)	複素数の正弦値を計算します。 $ImSin(C)=\frac{e^{iC}-e^{-iC}}{2i}$ ここで C は複素数で、$i=\sqrt{-1}$です。
Imsqrt(c)	複素数の平方根を計算します。
ImSub(c1,c2)	2 つの複素数の減算を実行します。
ImSum(c1,c2)	2 つの複素数の合計を求めます。
Real2Complex(real,imag)	指定した 2 つ実数を複素数に変換します。出力列のデータ型を complex (16)にする必要があります。**サンプル**: • `real2complex(1, 2)` real2complex(1, 2) 複素数を返します。1 + 2i `1 + 2i` • `real2complex(col(A), col(B))` real2complex(col(A), col(B))列 A の値を実部、列 B の値を虚部に使用した複素数を返します。

21.1.18. 経済

名前	説明
Effect(nrate,npery)	実効年間利率を計算します。
Nominal(erate,npery)	名目年間利率を計算します。
pDuration(rate,pv,fv)	投資が必要とする将来価値に達するために必要な期間数を計算します。

RRI(nper, pv, fv)	投資の成長に対応する金利を計算します。

21.1.19. 使用上の注意

それぞれの関数は、関数のタイプと与えられた引数により、一つの値、あるいは、ある範囲の値（データセット）のどちらかを返します。そうでない場合、何も指定がなければ、すべての関数は、渡される最初の引数が範囲の場合、範囲を返し、値が渡される場合、値を返します。

21.2. LabTalk がサポートする X ファンクション

カテゴリーで分類された以下の X ファンクション は LabTalk スクリプトで頻繁に使用されます。

これは Origin のすべての X ファンクションのリストではありません。LabTalk でサポートしているものだけです。カテゴリーで分類したアルファベット順の *すべての* X ファンクションは、X ファンクションリファレンスをご覧ください。

目次

21.2.1. データ探索

名前	説明
addtool_curve_deriv	グラフに矩形を追加して微分を実行します。
addtool_curve_fft	グラフに矩形を追加して、FFT を実行します。
addtool_curve_integ	グラフに矩形を追加して、その領域内の積分を実行します。
addtool_curve_interp	グラフに矩形を追加して、その領域内の積分を実行します。
addtool_curve_stats	グラフに矩形を追加して、その領域内の基本統計を計算します。
addtool_quickfit	グラフに矩形を追加して、フィットを実行します。
addtool_region_stats	領域統計:グラフに矩形または円形を追加して、その領域内の基本統計を計算します。
dlgRowColGoto	指定した行と列に移動するダイアログ
imageprofile	イメージプロファイルダイアログを開きます。
vinc	ベクターデータの増分の平均を計算します。
vinc_check	ベクターデータの増分の平均を計算します。

21.2.2. データ操作

名前	説明
addsheet	Assays テンプレートでデータフォーマットとフィット関数をセットアップします。
assays	Assays テンプレート設定:Assays テンプレートでデータフォーマットとフィット関数をセットアップします。
copydata	数値データをコピーします。

cxt	異なるモードでアクティブ曲線の X 値を移動します。
levelcrossing	指定したレベルと交差する X 座標を取得します。
m2v	行列をベクターデータに変換します。
map2c	振幅行列と位相行列を複素行列に変換します。
mc2ap	行列の複素数を振幅と位相に変換します。
mc2ri	行列の複素数をその実数部と虚数部に変換します。
mcopy	行列をコピーします。
mks	データプロット内のデータマーカを取得します。
mo2s	複数の行列オブジェクトを持つ行列レイヤを複数の行列レイヤを持つ出力行列ページに変換します。
mri2c	2 つの行列の実数を複素数の行列に組み合わせます。
ms2o	複数行列シートを 1 つの行列シートの複数の行列オブジェクトに統合(移動)します。
newbook	新しいワークブックまたは行列ブックを作成します。
newsheet	新しいワークシートを作成します。
rank	データポイントが指定した範囲内であるかどうか決定します。
reducedup	重複 X データを削減します。
reduce_ex	データポイントを平均し、データの数を減らして、等間隔な X にします。
reducerows	N ポイント毎のデータを基本統計量で置き換えます。
reducexy	X の分布に従って、サブグループの統計量毎に XY データを削減します。
subtract_line	グラフページ上の 2 点を結んだ直線からアクティブプロットを減算します。
subtract_ref	あるデータセットを別のデータセットで減算します。
trimright	Y 列の最後から欠損値を削除します。
v2m	ベクターデータを行列に変換します
vap2c	振幅と位相を複素数のベクターデータの形式に変換します。
vc2ap	複素数のベクターデータを振幅と位相に変換します。
vc2ri	複素数のベクターデータをその実数部と虚数部に変換します。

vfind	指定した値に等しいすべてのベクター要素を見つけます。
vri2c	複素数の実数部と虚数部から複素数のベクターデータを組み立てます。
vshift	ベクターデータをシフトします。
xy_resample	与えられたポリゴンの範囲内でメッシュ化し、データをリサンプリングします。
xyz_resample	メッシュとグリッディングで XYZ データをリサンプリングします。

グリッディング

名前	説明
m2w	行列データをワークシートに変換します。
r2m	ワークシートデータの範囲を直接行列に変換します。
w2m	ワークシートデータを直接行列に変換します。行列の座標は、ワークシートの最初の行と列および行番号で指定できます。
wexpand2m	行と列を拡張して、ワークシートを行列に変換します。
XYZ2Mat	XYZ ワークシートのデータを行列に変換する
xyz_regular	標準のグリッディング
xyz_renka	Renka-Cline グリッディング法
xyz_renka_nag	NAG Renka-Cline グリッディング法
xyz_shep	修正シェパードグリッディング法
xyz_shep_nag	NAG 修正シェパードグリッディング法
xyz_sparse	疎グリッディング
xyz_tps	Thin Plane スプライン補間

行列

名前	説明
mCrop	行列を矩形の領域でトリミングします。
mdim	現在の行列の次数や XY 座標値を設定します。
mexpand	現在の行列の各セルを列と行の倍率に応じて拡張します。

mflip	行列を水平または垂直に反転します。
mproperty	アクティブ行列のプロパティをセットします。
mreplace	指定したデータでアクティブ行列のセルを置き換えます。
mrotate90	行列を 90/180 度回転します。
msetvalue	ユーザ定義の数式でアクティブ行列のセルに値を割り当てます。
mshrink	縮小係数に従って行列を縮小します。
mtranspose	アクティブな行列を転置します。

プロット

名前	説明
plotbylabel	列ラベルをグループ化して複数レイヤグラフをプロットします。
plotgroup	ページグループ毎、レイヤグループ毎、データグループ毎にプロットします。
plotmatrix	データセットの散布図行列をプロットします。
plotmyaxes	複数 Y 軸プロットをカスタマイズします。
plotstack	積み上げグラフをプロットします。
plotxy	特定のプロパティを持つ XY データをプロットします。
plotms	指定した行列シート内のすべての行列オブジェクトの色付き曲面図または、カラーマップ曲面図をプロットします。
plotvm	仮想行列としてのワークシートのセル範囲からプロットします。

ワークシート

名前	説明
colcopy	フォーマットとヘッダを一緒に列のコピーをします。
colint	選択した Y 列に対するサンプリング間隔(暗黙の X 値)を設定します。
colmask	いくつかの条件に基づいて列の範囲をマスクします。
colmove	選択した列を移動します。
colshowx	選択した Y 列に対する X 列(サンプリング間隔を抽出)を表示します。

colswap	指定した 2 列の位置を交換します。
filltext	指定した範囲のセルにランダムな文字を入力します。
getresults	結果ツリーを取得します。
insertArrow	矢印を挿入します。
insertGraph	ワークシートのセルにグラフを挿入します。
insertImg	ファイルからイメージを挿入します。
insertNotes	ワークシートのセルにノートページを埋め込みます。
insertSparklines	ワークシートのセルにスパークラインを挿入します。
insertVar	セルに変数を挿入します。
merge_book	ワークブックを新しいワークブックに統合します。
sparklines	データの上部に各 Y 列のグラフのサムネールを表示します。
updateEmbedGraphs	ワークシートに埋め込んだグラフを更新します。
updateSparklines	データの上部に各 Y 列のグラフのサムネールを表示します。
w2xyz	フォーマットされたデータを XYZ 形式に変換します。
wautofill	オートフィルを行うワークシートの選択します。
wautosize	列の最長の文字列でワークシートの大きさを変更します。
wcellcolor	セルの色をセットして色で塗りつぶすか、選択した文字フォントをフォントの色にセットします。
wcellformat	選択したセルのフォーマットを指定します。
wcellmask	指定した範囲のセルにマスクをかけます。
wcellsel	指定した条件でセルを選択します。
wclear	ワークシートをクリアします。
wcolwidth	ワークシートの列の幅を更新します。
wcopy	指定したワークシートのコピーを作成します。
wdeldup	重複行の削除:1 列内の重複した値を持つワークシートの行を削除します。
wdelrows	指定したワークシート行を削除します。

wkeepdup	重複行の保持:1 列内の重複した値を持つワークシートの行を保持します。
wks_update_link_table	グラフ上のリンクテーブルをワークシートの内容に更新します。
wmergexy	あるワークシートから別のワークシートに XY データをコピーし、空白の行を挿入して一致していない X を統合します。
wmove_sheet	指定したワークシートを目的ワークシートに移動します。
wmvsn	ワークシートのすべての列のショートネームをリセットします。
wpivot	ピボットテーブル:ピボットテーブルを作成して、データのサマリーを視覚化します。
wproperties	スクリプトからツリーを介して、ワークシートプロパティを取得または設定します。
wrcopy	ラベルをコピーするオプションを持つワークシート範囲のコピー
wreplace	ワークシートのセル値を検索し、置換します。
wrow2label	ラベル値をセットします。
wrowheight	行高を設定します。
wsort	ワークシート全体または選択した列をソートします。
wsplit_book	指定したワークブックを 1 つのシートのみを持つ複数のワークブックに分離します。
wtranspose	現在のワークシートデータの行と列を転置します。
wunstackcol	グループデータを複数列にアンスタックします。
wxt	ワークシートデータの抽出します。

21.2.3. データベースアクセス

名前	説明
dbEdit	クエリの作成/編集/削除/読込み
dbImport	クエリを利用したデータベースからのデータインポート
dbInfo	データベース接続情報を表示
dbPreview	特定の先頭行に、クエリからのデータをプレビューするためにインポート

21.2.4. フィッティング

名前	説明
findBase	XY データ領域の基線を探します。
fitcmpdata	2 つのデータセットを同じフィットモデルで比較します。
fitcmpmodel	同じデータセットへの 2 つのフィットモデルを比較します。
fitLR	LabTalk を使った単純な線形回帰
fitpoly	LabTalk で使用する多項式フィット
getnlr	フィットレポートシートから NLFit ツリーを取得します。
nlbegin	LabTalk nlfit セッションを開始します。
nlbeginm	行列データに対して LabTalk の nlfit セッションを開始します。
nlbeginr	LabTalk の nlfit セッションを開始し、複数の従属/独立変数の関数をフィットします。
nlbeginz	xyz に対して LabTalk の nlfit セッションを開始します。
nlend	nlfit セッションを終了します。
nlfit	fit フィットセッションを行います。
nlfn	パラメータ自動初期化オプションをセットします。
nlgui	NLFit の出力結果や出力先を制御します。
nlpara	フィッティングパラメータダイアログを開きます。

21.2.5. グラフ操作

名前	説明
add_graph_to_graph	既存のグラフをレイアウトウィンドウに EMF オブジェクトとして貼り付けます。
add_table_to_graph	リンクしたテーブルをグラフに追加します。
add_wks_to_graph	既存のワークシートをレイアウトウィンドウに貼り付けます。
add_xyscale_obj	新しい XY スケールオブジェクトをレイヤに追加します。
axis_scrollbar	拡大縮小や移動を簡単に行うために軸スクロールバーをグラフに追加します。
axis_scroller	簡単に再スケールできるように下 X 軸に 1 対の逆三角形を追加します。

g2w	グラフをワークシートに移動します。
gxy2w	与えられた X 値に対して、すべての曲線から Y 値を見つけ、それをワークシートの行に追加します。
layadd	アクティブグラフに新しいレイヤを作成します。
layalign	元のレイヤに従って目的のレイヤを整列します。
layarrange	グラフ上のレイヤを再配置します。
laycolor	レイヤの背景色を塗りつぶします。
laycopyscale	レイヤから別のレイヤにスケールをコピーします。
layextract	指定したレイヤを別のグラフウィンドウに抽出します。
laylink	レイヤにいくつかのレイヤをリンクします。
laymanage	アクティブグラフのレイヤの構成を管理します。
laysetfont	レイヤ内のテキストの表示スケールを 1 つに修正します。
laysetpos	1 つ以上のグラフレイヤの位置をセットします。
laysetratio	レイヤの幅と高さの比をセットします。
laysetscale	グラフレイヤの軸スケールを設定します。
laysetunit	グラフレイヤの単位をセットします。
layswap	2 つのグラフレイヤの位置を入れ替えます。
laytoggle	左軸と下軸のオン/オフを切り替えます。
layzoom	レイヤの中央を拡大します。
legendupdate	グラフページ/レイヤの凡例を更新または再構成します。
merge_graph	選択したグラフウィンドウを 1 つのグラフに統合します。
newinset	インセット付きで新しいグラフを作成します。
newpanel	新しい区分グラフを作成します。
palApply	パレットをカラーマップに適用します。既存のパレットファイルを持つ指定したグラフにパレットを適用します。
pickpts	グラフから XY データポイントを取得します。
speedmode	スピードモードのプロパティをセットします。

21.2.6. 画像

調整

名前	説明
imgAutoLevel	画像に自動レベルを適用します。
imgBalance	画像のカラーバランスを調整します。
imgBrightness	画像の明るさを調整します。
imgColorlevel	ユーザ定義のカラーレベリングを画像に適用します。
imgColorReplace	事前定義した色の範囲で色を置換します。
imgContrast	画像のコントラストを調整します。
imgFuncLUT	画像にルックアップテーブル関数を適用します。
imgGamma	ガンマ係数を画像に適用します。
imgHistcontrast	メディアンを計算するヒストグラムを使って、画像のコントラストを調整します。
imgHisteq	ヒストグラム均等化を適用します。
imgHue	画像の色合いを調整します。
imgInvert	画像の色を反転します。
imgLevel	画像の明るさを調整します。
imgSaturation	画像の鮮やかさを調整します。

分析

名前	説明
imgHistogram	イメージヒストグラムを作成します。

代数的変換

名前	説明
imgBlend	2 つの画像を合成し、組み合わせた画像を作成します。
imgMathfun	画像のピクセル値に対していくつかのファクターを使って数学関数を実行します。
imgMorph	数値行列またはグレースケール/バイナリ画像ににモフォロジカルフィルタを適用します。

imgPixlog	ピクセル論理演算を実行します。
imgReplaceBg	背景色を置換します。
imgSimpleMath	2 つの画像間の算術演算を行います。
imgSubtractBg	画像背景を消去します。

変換

名前	説明
img2m	グレースケールの画像を数値の行列データに変換します。
imgAutoBinary	バイナリへ自動変換します。
imgBinary	バイナリへ変換します。
imgC2gray	グレースケール画像に変換します。
imgDynamicBinary	動的なしきい値を使ってバイナリに変換します。
imgInfo	スクリプトウィンドウに画像の基本パラメータを出力します。
imgPalette	画像にパレットを適用します。
imgRGBmerge	RGB チャネルを結合して、カラー画像を再合成します。
imgRGBsplit	カラー画像を別々の R、G、B チャネルに分解します。
imgThreshold	しきい値を使って画像の一部を白黒に変換します。
m2img	数値行列をグレースケールの画像に変換します。

幾何学的変換

名前	説明
imgCrop	画像を矩形の領域までトリミングします。
imgFlip	画像を水平または垂直に反転します。
imgResize	画像のサイズ変更
imgRotate	角度を指定して画像を回転します。
imgShear	画像を水平または垂直にシアーします(平行に歪めます)。
imgTrim	自動しきい値設定で画像をトリミングします。

空間フィルタ

名前	説明
imgAverage	画像に平均フィルタを適用します。
imgClear	画像を消去します。
imgEdge	エッジを検出します。
imgGaussian	ガウスフィルタを適用します。
imgMedian	メディアンフィルタを適用します。
imgNoise	画像にランダムノイズを付加します。
imgSharpen	画像のシャープネスを増加または減少します。
imgUnsharpmask	アンシャープマスクを適用します。
imgUserfilter	ユーザ定義のフィルタを適用します

21.2.7. インポートとエクスポート

名前	説明
batchProcess	分析テンプレートを使ってバッチ処理を行い、サマリーレポートを生成します。
expASC	ワークシートデータを ASCII ファイルとしてエクスポートします。
expGraph	グラフを画像ファイルにエクスポートします。
expImage	アクティブなイメージを画像ファイルにエクスポートします。
expMatASC	行列データを ASCII ファイルとしてエクスポートします。
expNITDM	ワークブックを National Instruments 社の TDM および TDMS ファイルにエクスポートします。
expPDFw	ワークシートを複数ページの PDF ファイルとしてエクスポートします。
expWAV	Microsoft PCM wav ファイルとしてデータをエクスポートします。
expWks	アクティブシートをラスターまたはベクターの画像ファイルとしてエクスポートします。
img2GIF	アクティブなイメージを gif ファイルにエクスポートします。
impASC	ASCII ファイルをインポートします。
impBin2d	バイナリ 2D 配列ファイルをインポートします。

impCDF	CDF ファイルをインポートします。バージョン 3.0 以下のファイルをサポートします。
impCSV	csv ファイルをインポートします。
impDT	データトランスレーション Version 1.0 ファイルをインポートします。
impEDF	EDF ファイルのインポート
impEP	EarthProbe ファイル (EPA)をインポートします。EPA ファイルだけが EarthProbe データに対してサポートされます。
impExcel	Microsoft Excel 97-2007 ファイルをインポートします。
impFamos	Famos Version 2 ファイルをインポートします。
impFile	事前に定義したフィルタを使ってファイルをインポートします。
impHDF5	HDF5 ファイルをインポートします。1.8.2 以下のバージョンをサポートします。
impHEKA	HEKA (dat)ファイルをインポートします。
impIgorPro	WaveMetrics IgorPro (pxp, ibw)ファイルをインポートします。
impImage	画像ファイルをインポートします。
impinfo	ファイルインポートに関する情報を読み取ります。
impJCAMP	JCAMP-DX Version 6 ファイルをインポートします。
impJNB	SigmaPlot ファイル (JNB)をインポートします。SigmaPlot 8.0 以下のバージョンをサポートしています。
impKG	カレイダグラフのファイルをインポートします。
impMatlab	Matlab ファイルをインポートします。
impMDF	ETAS INCA MDF (DAT, MDF)ファイルをインポートします。INCA 5.4 (ファイル version 3.0)をサポートしています。
impMNTB	Minitab ファイル(MTW)またはプロジェクト(MPJ)をインポートします。Minitab 13 以前のバージョンをサポートします。
impNetCDF	netCDF ファイルをインポートします。バージョン 3.1 以下のファイルをサポートします。
impNIDIAdem	National Instruments DIAdem 10.0 データファイルをインポートします。
impNITDM	National Instruments の TDM および TDMS ファイル(TDMS は日時フォーマットをサポートしていない)をインポートします。
impODQ	*.ODQ ファイルをインポートします。

imppClamp	pCLAMP ファイルをインポートします。pClamp 9 (ABF 1.8 ファイル形式)および pClamp 10 (ABF 2.0 ファイル形式)をサポートしています。
impSIE	Import nCode Somat SIE 0.92 file
impSPC	Thermo ファイルをインポートします。
impSPE	Princeton Instruments (SPE)ファイルをインポートします。Minitab 2.5 以前のバージョンをサポートします。
impWav	wave 形式のオーディオファイルをインポートします。
insertImg2g	ファイルからイメージを挿入:グラフウィンドウに画像ファイルを挿入します。
lwfilter	X ファンクションのインポートフィルタを作成します。
plotpClamp	pClamp データのプロットをします。
reimport	現在のファイルを再インポートします。

21.2.8. 数学

名前	説明
avecurves	複数の曲線を平均または連結します。
averagexy	複数の曲線を平均または連結します。
bspline	キュービック B スプライン補間および補外を実行します。
csetvalue	列値の設定をします。
differentiate	入力データの微分を計算します。
filter2	カスタマイズしたフィルタを行列に適用します。
integ1	入力データの積分を計算します。
integ2	ゼロ平面と行列の曲面の体積を計算します。
interp1	3 つの方法を使って、指定した X 値から Y 値を検索する XY データの 1D 補間/補外を実行します。
interp1q	線形補間および補外を実行します。
interp1trace	データにトレース/周期補間を実行します。
interp1xy	3 つの方法を使って、XY データに 1D 補間/補外を実行し、均一 X 値で補間データを生成します。
interp3	3D 補間を実行します。

interpxyz	XYZ データにトレース補間を実行します。
marea	行列曲面の表面積を計算します。
mathtool	データに単純な算術演算を実行します。
medianflt2	メディアンフィルタを行列に適用します。
minterp2	行列に 2D 補間/補外を実行します。
minverse	(擬似)逆行列を生成します。
normalize	入力データを正規化します。
polyarea	閉じたプロット領域の面積を計算します。
reflection	ある区間にデータの範囲を反映します。
rnormalize	列の正規化:列ごとに入力範囲列を正規化します。
specialflt2	事前に定義した特殊フィルタを行列に適用します。
spline	スプライン補間および補外を実行します。
vcmath1	1 つの複素数に対して単純な算術演算を実行します。
vcmath2	2 つの複素数に対して単純な算術演算を実行します。
vmathtool	入力データに単純な算術演算を実行します。
vnormalize	入力ベクターデータを正規化します。
white_noise	データにホワイトノイズ(ガウスノイズ)を付加します。
xyzarea	XYZ 曲面の表面積を計算します。

21.2.9. 信号処理

名前	説明
cohere	コヒーレンスを実行します。
conv	2 つの信号のコンボリューションを計算します。
corr1	2 つの信号の 1 次相関を計算します。
corr2	2 次の相関を計算します。
deconv	デコンボリューションを計算します。

envelope	データの包絡線を取得します。
fft_filter2	2D FFT フィルタを実行します。
fft_filters	FFT フィルタを実行します。
hilbert	ヒルベルト変換を実行または解析信号を計算します。
msmooth	行列を拡張または縮小してスムージングします。
smooth	不規則なデータやノイズの多いデータにスムージングに対して実行します。

FFT

名前	説明
fft1	入力データに対して高速フーリエ変換を実行します。(離散フーリエ変換)
fft2	2 次元高速フーリエ変換を実行します。
ifft1	逆フーリエ変換を実行します。
ifft2	逆 2 次元離散フーリエ変換を実行します。
stft	短時間フーリエ変換を実行します。
unwrap	位相の角度をより滑らかな位相に変換します。

ウェーブレット

名前	説明
cw_evaluate	連続ウェーブレット変換の計算
cwt	実数の 1 次元の連続ウェーブレット係数を計算します。
dwt	1D の離散ウェーブレット変換
dwt2	ウェーブレット変換で行列データを分解します。
idwt	近似係数と詳細係数から 1 次元の逆ウェーブレット変換を行ないます。
idwt2	係数行列から 2D 信号を再構成します。
mdwt	マルチレベルの 1D ウェーブレット分解
wtdenoise	ウェーブレット変換を使ってノイズを除去します。
wtsmooth	詳細係数を切り捨てることで信号をスムージングします。

21.2.10. スペクトル分析

名前	説明
blauto	基線を自動的に作成します。
fitpeaks	ガウス関数またはローレンツ関数で曲線をフィットして、複数ピークを検出します。
pa	ピークアナライザを開きます。
paMultiY	分析テーマを使ってピーク分析のバッチ処理を行い、サマリーレポートを生成します。
pkFind	曲線上のピークを検出します。

21.2.11. 統計

記述統計

名前	説明
colstats	列の統計を実行します。
corrcoef	選択したデータの相関係数を計算します。
discfreqs	離散/カテゴリーデータの度数を計算します。
freqcounts	頻度カウントを求めます。
kstest	1 つのサンプル Kolmogorov-Smirnov 正規性のテストを実行します。
lillietest	リリーフォースの正規性の検定をします。
mmoments	選択したデータのモーメントを計算します。
moments	選択したデータのモーメントを計算します。
mquantiles	選択したデータの四分位を計算します。
mstats	選択したデータの記述統計量を計算します。
quantiles	選択したデータの四分位を計算します。
rowquantiles	行の四分位を計算します。
rowstats	行の記述統計量を計算します。
stats	選択したデータの記述統計量を計算します。
swtest	Shapiro-Wilk 正規性の検定を実行します。

仮説検定

名前	説明
rowttest2	行の 2 標本の t 検定を実行します。
ttest1	対応のある t 検定を実行します。
ttest2	対応のある t 検定を実行します。
ttestpair	対応のある t 検定を実行します。
vartest1	カイ二乗分散検定を実行します。
vartest2	F 検定を実行します。

ノンパラメトリック検定

名前	説明
friedman	フリードマン ANOVA を実行します。
kstest2	入力データに対して 2 標本のコルモゴルフ・スミルノフ検定を実行します。
kwanova	クラスカル・ウォリス検定を実行します。
mediantest	メディアン検定を実行します。
mwtest	マンホイットニー検定を実行します。
sign2	対応のある標本の符号検定を実行します。
signrank1	1 標本の Wilcoxon 符号順位検定を実行します。
signrank2	対応のあるデータの Wilcoxon 符号順位検定を実行します。

生存分析

名前	説明
kaplanmeier	カプランマイヤー推定(積・極限推定) 分析を実行します。
phm_Cox	Cox 比例ハザードモデル分析を実行します。
weibullfit	生存データにワイブルフィットを実行します。

21.2.12. ユーティリティ

名前	説明
customMenu	カスタムメニューの編集ダイアログを開きます。
get_plot_sel	データプロット内でプロット選択します。
get_wks_sel	ワークシートで選択内容を取得します。
themeApply2g	テーマをグラフに適用します。
themeApply2w	テーマをワークシートに適用します。
themeEdit	「**テーマの編集**」ツールを使って指定したテーマファイルを編集します。
xop	クラスに基づいた操作のフレームワークを実行する X ファンクションを実行します。

ファイル

名前	説明
cmpfile	2 つのバイナリファイルを比較して、比較結果を印刷します。
dlgFile	ファイルを開くダイアログでファイルを選択するよう促します。
dlgPath	オープンパスダイアログでパスを選択することを促します。
dlgSave	名前を付けて保存ダイアログで促します。
filelog	文字列を使ってユーザの仕事をメモしたり、記録するテキストファイルを作成します。
findFiles	ファイルを検索します。
findFolders	フォルダを検索します。
imgFile	ファイルを開くダイアログで画像を選択するように促します。
template_saveas	グラフ/ワークブック/行列ウィンドウをテンプレートに保存します。
web2file	Web ページをローカルファイルにコピーします。

システム

名前	説明
cd	作業ディレクトリを変更または表示します。
cdset	現在の作業ディレクトリに指定したインデックスを割り当てます。または、割り当てられたすべてのインデックスおよびそれに結びつけられたパスを一覧表示します。
debug_log	デバッグ用のログファイルを作成するために使用します。OriginLab に問題をレポートする場合のみ使用します。
dir	現在の作業ディレクトリにあるスクリプト(ogs)および X ファンクション(oxf)を一覧表示します。
dlgChkList	チェックボックス付きダイアログを開き、閉じる時に選択されたチェックボックスステータスを返します。
group_server	グループリーダーおよびグループメンバー用のグループフォルダの場所を設定します。
groupmgr	グループフォルダファイルを管理するグループリーダのツールを表示します。
instOPX	Origin XML パッケージをインストールします。
language	Origin の表示言語を変更します。
lc	X ファンクションのカテゴリーまたは指定したカテゴリーの X ファンクションをすべて一覧表示します。
lic	モジュールライセンスの更新:モジュールライセンスファイルを Origin に追加します。
lx	X ファンクションを一覧表示します(名前毎、キーワード毎、場所毎など)
mkdir	現在の作業ディレクトリに新しいフォルダを作成します。
op_change	操作オブジェクトに保存されたツリーを取得およびセットします。
pb	プロジェクトブラウザを開きます。
pe_cd	プロジェクトエクスプローラのディレクトリを変更します。
pe_dir	現在のプロジェクトエクスプローラのフォルダとワークブックを一覧表示します。
pe_load	Origin のプロジェクトを現プロジェクトの既存フォルダにロードします。
pe_mkdir	新しいフォルダを作成します。
pe_move	フォルダ内の指定したページを指定したフォルダに移動します。
pe_path	プロジェクトエクスプローラのパスを探します。
pe_rename	ページまたはサブフォルダの名前を変更します。

pe_rmdir	PE のアクティブフォルダ内のサブフォルダを削除します。
pe_save	現プロジェクトのフォルダを Origin プロジェクトファイルに保存します。
pef_pptslide	フォルダ内のすべてのグラフを PowerPoint にエクスポートします。
pef_slideshow	フォルダ内のすべてのグラフのスライドショー(全画面表示)を行います。
pemp_pptslide	選択したグラフを PowerPoint スライドにエクスポートします。
pemp_slideshow	選択したグラフのスライドショー(全画面表示)を行います。
pep_addshortcuts	お気に入りフォルダに選択したウィンドウのショートカットを追加します。
pesp_gotofolder	このページが存在する元のフォルダに移動します。
updateUFF	旧バージョンのユーザファイルを移行します。
ux	指定した場所にある X ファンクションリストを更新します。

22 索引

LabTalk に関連するヘルプ

リファレンス	場所
X ファンクション	メニュー: ***ヘルプ:X ファンクション*** • 個別の X ファンクションリファレンス
Origin C	メニュー: ***ヘルプ:プログラミング:Origin C*** • セクション **OriginC Reference> Global Functions> LabTalk Interface** Origin C から LabTalk を実行するについて
コードビルダ	メニュー: ***ヘルプ:プログラミング:コードビルダ*** • コードビルダの使用法
チュートリアル	メニュー: ***ヘルプ:チュートリアル*** • セクション ***チュートリアル> プログラミング> コマンドウィンドウと X ファンクション*** LabTalk コマンドと X ファンクションを実行する方法についての初心者向けの簡単なチュートリアル
動画	Web サイト: http://www.originlab.com/index.aspx?go=Products/Origin/ImportingData&pid=1163 • データをインポートした後に LabTalk スクリプトを実行する方法を学びます。(英語のみ)

索引