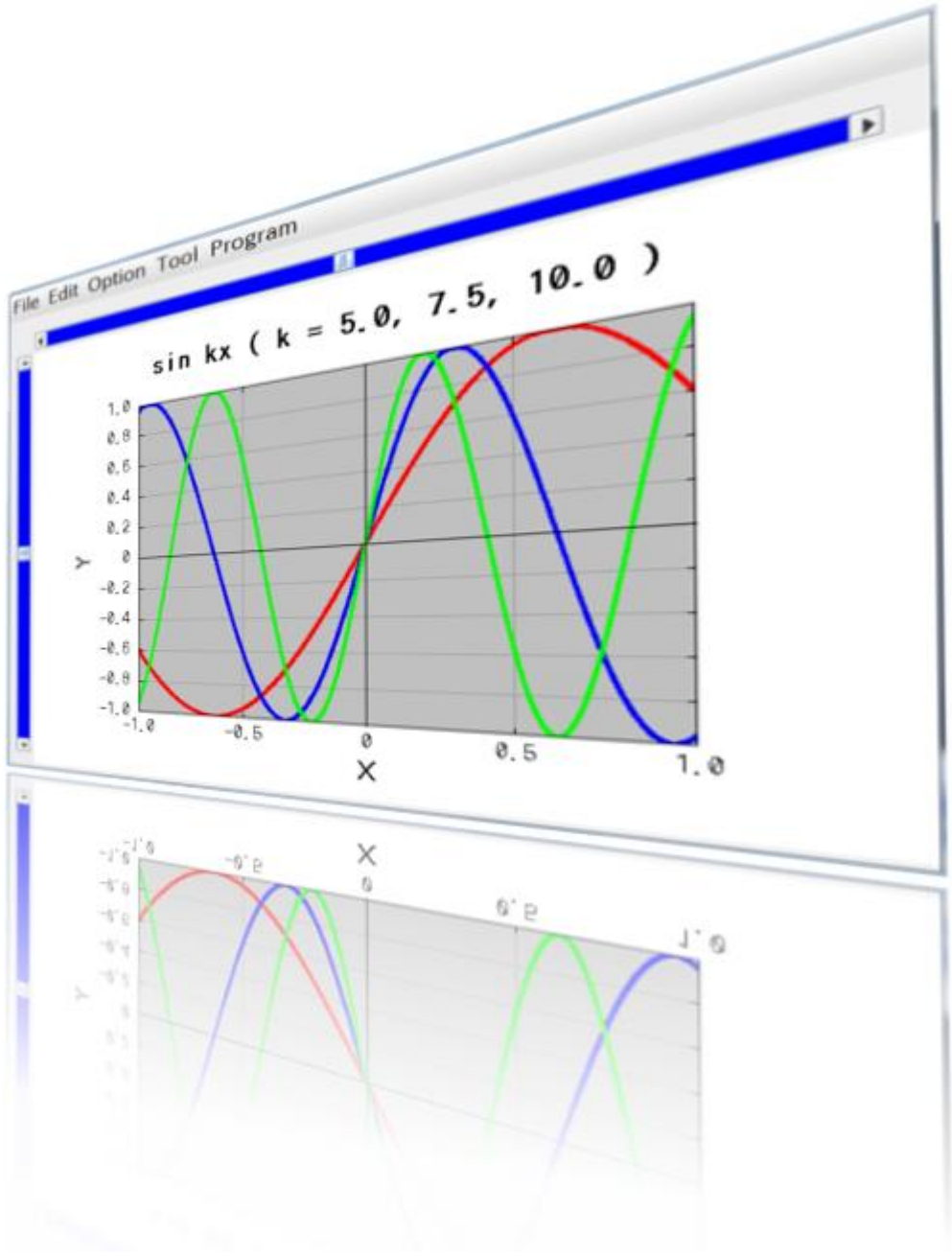
RINEARN
www.rinearn.com



RINEARN Graph 2D　5.6

取扱説明書　第 6 版

■目次

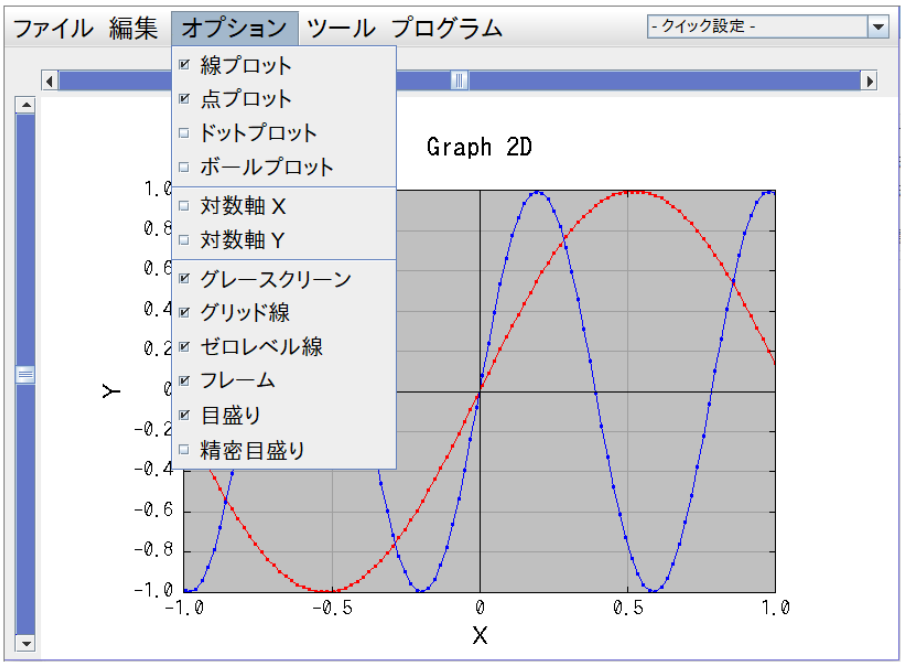
1. はじめに

リニアングラフ 2D とは

リニアングラフ 2D（RINEARN Graph 2D）は、数値計算プログラムや表計算ソフトなどで作成された座標値ファイルから、2 次元の散布図・折れ線グラフをプロットできる、データ解析・数値計算分野向けの 2D グラフソフトです。全て Java 言語で開発されているため、様々な種類の PC 用オペレーティングシステム上においてインストール不要で動作し、USB メモリーなどに入れて持ち運んでの利用も可能です。また、商用・非商用を問わず、どなたでも無料で利用できます。

リニアングラフ 2D 公式 Web サイト：
https://www.rinearn.com/ja-jp/graph2d/

リニアングラフ 2D は、厳格な文書用に高度な図をプロットするグラフソフトよりは、むしろそういったソフトと併用する事を前提とした、「普段使い」のためのシンプルで軽快なプロットツールを目指しています。プロットオプションはメニューバーからすぐに切り替え可能で、プロット範囲の移動や拡大・縮小もマウス操作で直感的に行えます。



グラフ画面の描画エンジンも同様に、「普段使い」を重視してレスポンスやプロット速度などを優先させた特性になっています。高機能系グラフソフトと比べると見栄えや滑らかさなどは劣りますが、マウスドラッグやスクロールでの範囲・倍率操作には軽快に反応し、アニメーションプロット機能なども備えています。百万点を超える高密度ドットプロットなどにも、さすがに少し重くなりますが耐えられます。

ライセンスに関して

リニアングラフ 2D は、どなたでも無償にてご利用頂けます。リニアングラフ 2D を利用して作成された画像に関しても、ご自由にご使用頂けます（※フォントの著作権にはご注意ください）。

また、データファイルや、API を使用するプログラム等に、リニアングラフ 2D を同梱して二次配布する事も可能です。ただし、同梱ではなく単体で二次配布する事はご遠慮ください。

詳しくは、ダウンロードしたパッケージ内の「 License 」フォルダ内にライセンス文書が添付されていますので、ご参照ください。二次配布の際も、そのライセンス文書が含まれる形にしてください。

※ なお、このソフトウェアの起動時に、Java®実行環境（JRE）を「jre」フォルダ内にダウンロードして使用するか尋ねられる場合がありますが、そこでダウンロードされる JRE には、その JRE 自身の別のライセンスが適用されます。商用・非商用問わず無償で使用できますが、配布等には条件が存在します。詳細は ReadMe や「jre」フォルダ内の文書、および JRE 付属の説明書等をご参照ください。

Ver.5.6 の特徴

Ver.5.6 では、リニアングラフ 3D と同様、各画面のデザインがリファインされ、より扱いやすくなりました。メニューの階層構造や内容は保たれているため、これまでのバージョンに慣れた方でも違和感なく馴染む事ができます。ユーザーインターフェース面での大きな変更として、リニアングラフ 3D で先に採用されていた「クイック設定」セレクタが、画面右上に追加されました。このセレクタから、複数の設定を手軽に切り替える事ができます。

加えて、こちらもリニアングラフ 3D 同様に、Java 言語のコードから制御するための API が新たにサポートされました。これにより、リニアングラフ 2D を単体のグラフソフトとしてだけでなく、Java 言語でのプログラミング時のグラフ描画ライブラリとしても使用できるようになりました。プログラムからは、通常の基本操作やファイルからのプロットだけでなく、配列データを直接渡してプロットさせたり、アニメーションさせる事もできます。

描画処理関連では、これまでご要望の多かった線幅の設定に対応しました。また、アンチエイリアス処理が使用可能（標準で有効）になり、文字や線・点などの描画が滑らで綺麗になりました。

ワンポイント! 3 次元版のリニアングラフ 3D について

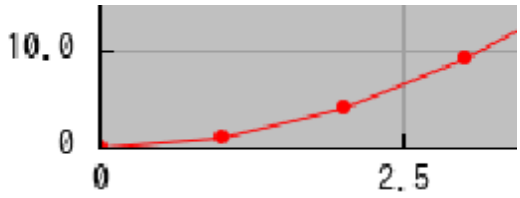
リニアングラフ 2D には、兄弟ソフトとして、3 次元グラフをプロットできる「 リニアングラフ 3D 」も存在します。両者は同じ感覚で併用できるよう、共通の画面デザインや操作インターフェースを採用しています。ぜひ併せてご活用ください。

リニアングラフ 3D 公式 Web サイト：

https://www.rinearn.com/ja-jp/graph3d/

■ グラフ画面の画質が粗い場合は…

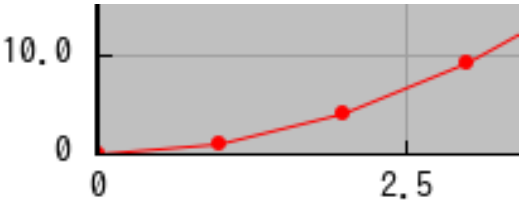
リニアングラフ 2D でグラフを描画した際、環境によっては、ウィンドウ上に表示されるグラフの文字や線の境界などが、ギザギザが目立つ印象で粗く表示される場合があります（例：下図）。



これは、ご使用の PC において、ディスプレイに内容を拡大して表示する設定になっている際などに生じます。この設定は、例えば Microsoft® Windows® をご使用の場合、デスクトップを右クリックして「 ディスプレイ設定 」などから拡大率の設定を確認・変更できます。

※ OS 側のこの機能に問題があるわけでは全くなく、ソフトウェア側を含めた色々な事情でやむを得ない現象です。

ディスプレイの拡大率がちょうど 100%でない場合、上のように文字や線の境界が粗く表示される事があります。100%に設定した際の画質は下図の通りです（これが本来の画質です）：



ただし、画質が粗くなるのは、あくまでもウィンドウ上での見かけだけであって、グラフ画像をファイルに保存したり、右クリックでコピーしたりする際の画質には全く影響しません（100%表示時と同じ画質になります）。そのため、見づらくて作業が難しいレベルでなければ、無理にディスプレイの拡大率を 100%にして使用する必要はありません。

2. 起動・プロット方法

リニアングラフ 2D は、インストール不要ですぐに使えます。さっそく起動し、使用してみましょう。

2-1. ダウンロードした ZIP 形式の配布ファイルの展開方法

ダウンロードしたままのリニアングラフの配布ファイルは、圧縮された ZIP 形式のファイルになっています。まずは、その **ZIP ファイルを右クリックして「すべて展開」や「ここに展開」などを選び、展開してください。**すると、元の ZIP ファイルと同じ名前のフォルダが出現し、その中に ZIP ファイルの中身が入っています。以降、その中のものを使用してください。元の ZIP ファイルはもう不要です。

※「問題を引き起こす可能性～」などのエラーが表示されて展開を完了できない場合、ZIP ファイルを右クリックして「プロパティ」を選択し、プロパティの画面の下にあるセキュリティ項目の「許可する」にチェックを入れて「OK」すると、以降は展開可能になります。このエラーは、インターネット等から入手した不明なプログラムを安易に実行しないための、OS のセキュリティ機能によるものです。

※ Linux®等をご使用の場合で、展開結果のファイル名の日本語が文字化けしてしまう場合があります。その場合、コマンドライン端末から以下のように unzip コマンドで展開してみてください：

```
cd　ZIP ファイルのある場所
unzip　-O　cp932　ZIP ファイル名
```

効果が無かった場合は、他の展開ソフトを使用するか、展開後に convmv コマンドなどで文字化けの修復処理を試してみてください（日本語ファイル名は CP932 でエンコードされています）。

2-2. 簡単な起動方法

ZIP ファイルを展開できたら、まずは最も簡単な方法でリニアングラフ 2D を起動してみましょう ！

・Microsoft® Windows® をご使用の場合（※コマンドでの利用は後の 2-4 参照）

展開したフォルダ内の「 **RinearnGraph2D_?.?.?.bat（種類はバッチファイル、「?」の箇所はバージョン番号の数字）**」をダブルクリックすると起動します。最初に設定や初回処理について尋ねられるので、適時答えると、リニアングラフ 2D が起動します。2 回目以降はすぐに起動します。

特定の拡張子のファイルをダブルクリックするとリニアングラフ 2D で開かせたい場合は、そのファイルを右クリックして「プログラムから開く」メニューで、このバッチファイルで開くよう指定します。

・Linux®等やその他の OS をご使用の場合 （※常用する場合は後の 2-4 参照）

展開フォルダ内の「 **RinearnGraph2D.jar（JAR ファイル） 」** を、コマンドライン端末から：

```
cd　展開したフォルダ
java -jar RinearnGraph2D.jar
```

と入力して実行できます。必要に応じて、java コマンドに「-Xmx」オプションを追加してメモリー容量を指定します （例：512MB 使用するなら「 java -Xmx512m -jar RinearnGraph2D.jar 」）。

※ もし、java コマンドが使用できない旨のエラーが出る場合は、Java®実行環境（JRE）を導入する必要があります。apt コマンドが使える環境では、まずコマンドライン端末上で：

```
apt search jre　　(または apt の代わりに apt-cache)
```

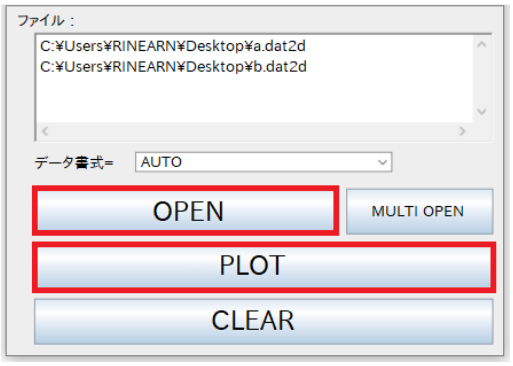
して入手可能な一覧を確認の上で、コマンドラインで以下の例のように導入できるかもしれません：

```
sudo apt install default-jre　　(または apt の代わりに apt-get)
または
sudo apt install openjdk-?-jre　　(?の箇所にはバージョンの数字が入ります)
```

他のものでも構いませんが、リニアングラフ 2D が動作しないものもあります（※末尾に -headless が付いているものでは動作しないので、付けないようご注意ください）。

2-3. 起動後、ファイルをプロット（グラフに描画）する方法

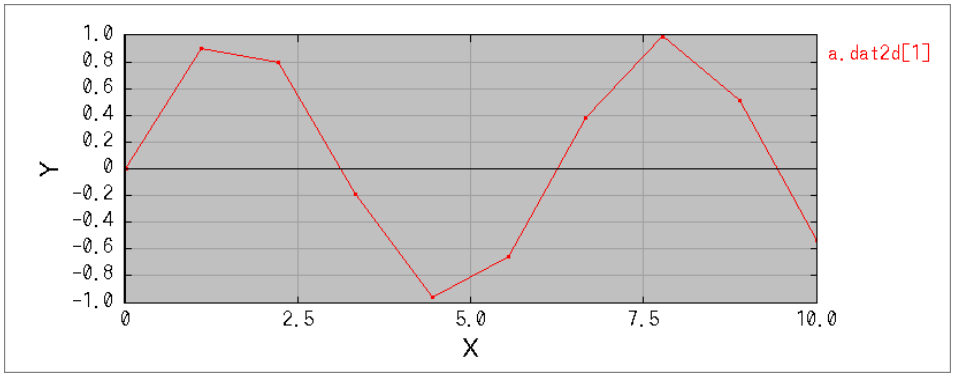
より高度な起動設定などは後に回して、まずはファイルをプロットしてみましょう ！



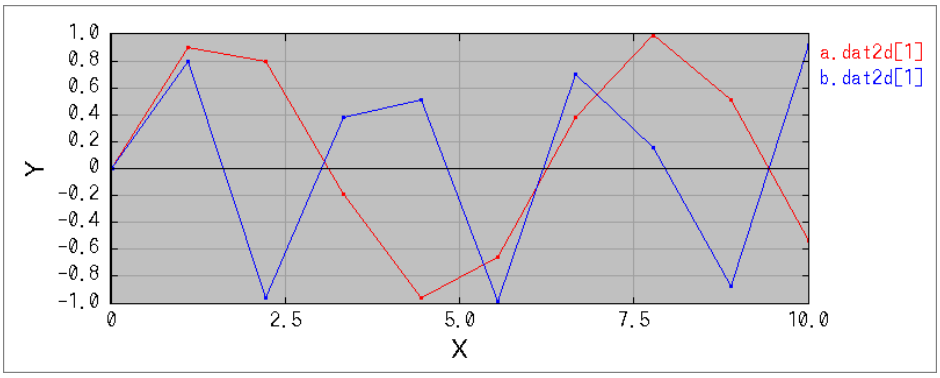
起動後、メニューバーから「ファイル/File」 > 「ファイルを開く/Open File」メニューを選択し、プロットしたいファイルを選択してください（「/」の後は英語モードでの表記です）。

「OPEN」ボタンでファイルを選択すると、それが上のテキストエリアに追加されます。プロット対象ファイルを全て追加したら、「PLOT」ボタンを押して、グラフにプロットしてください。（ ※ ファイルを開かずに、表計算ソフト上などのデータを直接プロットしたい場合は、表計算ソフト上でデータを選択コピーし、グラフ画面を右クリックして 「 データの貼り付け（Paste Data） 」 を選択してください。）

以下の図は、適当なファイルを 1 個プロットした例です。単純な（単系列の）ファイルでは、このように赤色の点が各座標の位置にプロットされ、折れ線でつながれます。折れ線でつなぎたくない場合は、メニューバーから「オプション / Option」＞「線プロット / With Lines」のチェックを外してください。同様に点が要らない場合は、「点プロット/With Points」のチェックを外してください。



ファイルを複数選択してプロットした場合、ファイルごとに異なる系列と見なされ、それぞれ別の色でプロットされます。下図は実際に２つのファイルをプロットした例です。なお、１つのファイル内に複数の系列を含ませる事もできます。詳細は「5. 座標値ファイル書式」の説明をご参照ください。



複数系列のグラフは、系列番号を時刻番号と見なしてアニメーションさせる事もできます。アニメーションモードは、メニューバーから「ツール / Tool」＞「アニメーション / Animation」を選択すると使用できます。

ワンポイント！ **メモリーが不足する場合は…**

プロットするファイルのサイズによっては、メモリーが不足し、「メモリーエラー」や「Memory Error」などとメッセージが表示されてソフトウェアが終了してしまいます。

その際、Microsoft Windows をご使用の場合は、「SetMemorySize_メモリー容量設定.bat（バッチファイル）」をダブルクリック実行し、メモリー容量を大きめの値に再設定してください。

Linux 等やその他 OS をご使用の場合は、「簡単な起動方法」の項目で述べた通り、java コマンドに -Xmx オプションを付加して実行してください。ただし後述のパス設定を行って ring2d コマンドをご使用の場合は、リニアングラフ 2D の「bin」フォルダ内の「ring2d」というファイル（拡張子なし）をテキストエディタで開き、中の java コマンド実行部に -Xmx オプションを追記してください。

2-4. コマンドライン端末上で常用する場合の設定

リニアングラフ 2D を、コマンドプロンプトやシェルなどのコマンドライン端末上で常用したい場合は、リニアングラフ 2D をダウンロード・展開したフォルダ内の「bin」というフォルダのパス（場所）を、環境変数 Path または PATH に設定します（後述）。すると ring2d コマンドが使用可能になり、

```
ring2d  a.dat2d
```

とコマンド入力するだけで、リニアングラフ 2D を起動してファイル「 a.dat2d 」をプロットできるようになります。空白を挟んで、複数のファイルをプロットする事もできます：

```
ring2d  a.dat2d  b.dat2d
```

・Microsoft® Windows® をご使用の場合のパス設定

まずは、リニアングラフ 2D のフォルダをどこか適当な場所へ配置（ずっとそこへ置きます）してください。その後にパス設定を行いますが、バージョンによって画面の開き方や手順が異なるため、詳細は「Windows Path 設定」などで検索してみてください。Windows 10 をご使用の場合は：

スタートボタン ＞ 歯車アイコン（設定） ＞「環境変数を編集」と検索して移動 ＞ 開かれた画面で「ユーザー環境変数」の一覧から「Path」（無ければ作成）を選び「編集」 ＞「新規」を押し、リニアングラフ 2D の「bin」フォルダのパスを入力※
（※ Shift キーを押しながら bin フォルダを右クリックすると、メニューからパスのコピーを行えます）

で行えますが、誤って他の値をいじったり、削除してしまったりすると大変な事になるのでご注意ください（上の説明で、システム環境変数への登録もできますが、その点を考慮してユーザー環境変数に登録しています）。 手慣れた方に依頼できる場合は、行ってもらう方が無難かもしれません。

・Linux® 等やその他の OS をご使用の場合のパス設定（および実行権限設定）

まずは、リニアングラフ 2D のディレクトリ（フォルダ）をどこか適当な場所へ配置（ずっとそこへ置きます）してください。ここでは例として以下の場所に配置したとします。

/usr/local/bin/rinearn/rinearn_graph_2d_?_?_?/ （?_?_?の箇所はバージョン番号です）

続いて cd コマンドでこの場所のさらに bin ディレクトリ内まで移動し、以下のコマンドを実行します：

```
chmod +x ring2d          （起動用のシェルスクリプトに実行権限を付加しています）
```

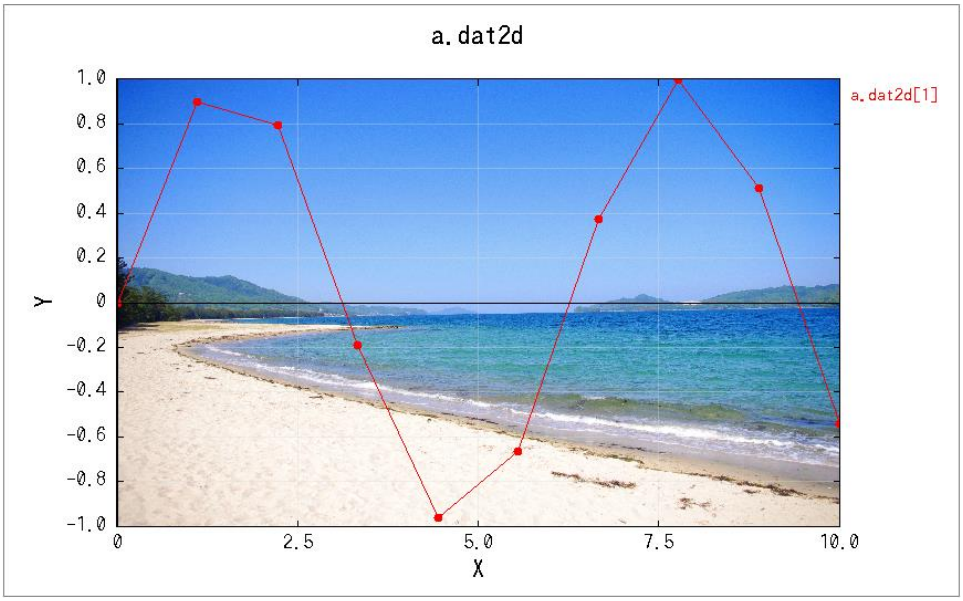
最後に、ユーザーのホームディレクトリにある、「.bashrc（隠しファイル）」または「.bash_profile」もしくは「.profile」（どのファイルが有効かはオペレーティングシステムによって異なります）をテキストエディタで開き、最終行に下記の一行を追記してください。

```
export PATH=$PATH:/usr/local/bin/rinearn/rinearn_graph_2d_?_?_?/bin/
```
（?_?_?の箇所はバージョン番号で置き換えて下さい）

$PATH:以降の内容は、リニアングラフ 2D のディレクトリを配置した場所に合わせてください。

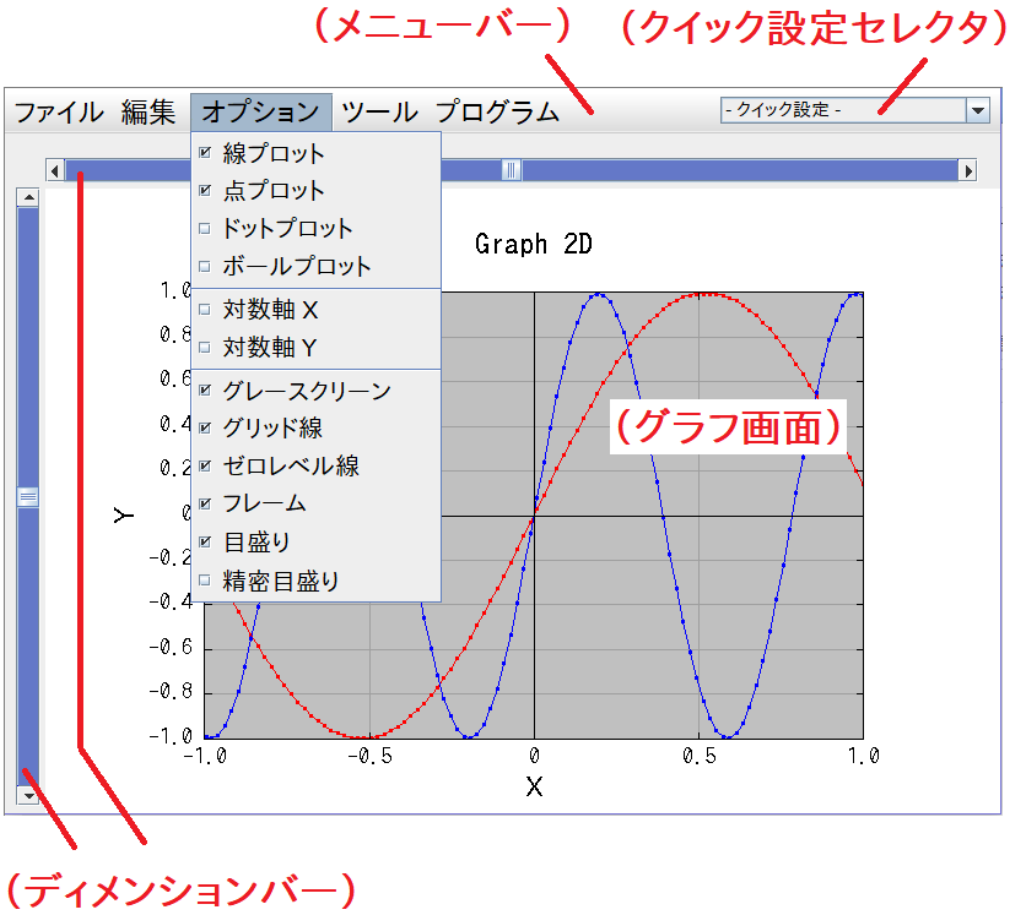
ワンポイント！ **背景に画像を表示する「壁紙モード」**

リニアングラフ 2D には、背景に画像ファイルを表示する「壁紙モード」が用意されています。上述のどの起動方法においても、ダウンロード・解凍したリニアングラフ 2D のフォルダ直下（RinearnGraph2D.jarと同じ場所）に「RinearnGraph2DScreen」という名前のJPEG形式画像ファイル（拡張を含めた名前は「RinearnGraph2DScreen.jpg」）を入れておくと、起動的に壁紙として読み込まれて表示されます。



壁紙モードは、本来は長時間作業時の快適性確保のための機能でしたが、グラフ背景に解説文や矢印、基準線などを合成したい場合にも便利です。また、別のグラフの画像を背景にして、その上に座標値データファイルの内容を重ねてプロットするなどの用途も考えられます（※余白などを除去しておく必要があります）。

3. 画面の操作方法



・メニューバー

画面上部のメニューバーから、座標値ファイルを開いたり、プロットオプションの指定等の各種設定を行ったりする事ができます。詳しくは次章で解説します。

・ディメンションバー

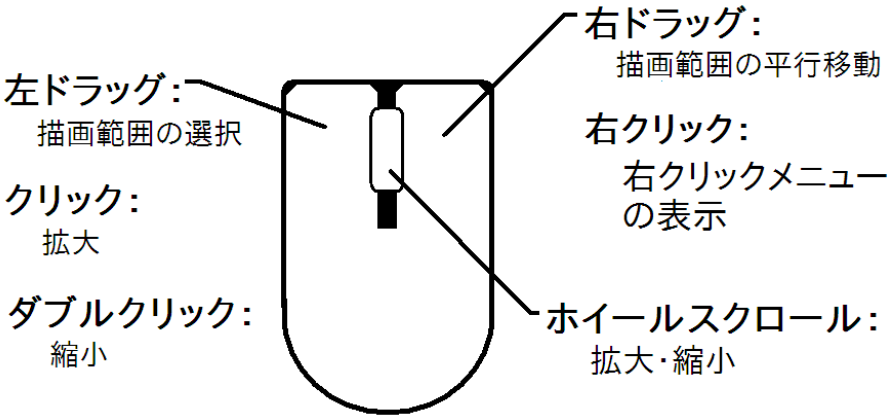
画面上端と左端に並ぶ 2 本のディメンションバーで、それぞれグラフの X、Y 各方向のプロット範囲を独立に拡大・縮小できます。※ディメンションバーの操作は非常に敏感（指数的）に反映されるので、繊細に操作してください。

・クイック設定セレクタ

画面右上のクイック設定セレクタで、設定をすぐに切り替える事ができます。設定の追加は、セレクタから「設定の追加」を選ぶと行えます。追加した設定は「 RinearnGraph2DQuickSetting 」フォルダ内に保存されますので、バージョン更新後に引き継ぎたい場合はコピーしてください。

・グラフ画面

画面中央のグラフ画面にグラフが描画されます。このエリアではマウス操作により、描画範囲の変更や拡大・縮小等を行えます。また、マウスカーソルの下に十字型のカーソルと、その地点の座標値が描画されます。



以下では、マウス操作の各項目について解説します。

・左ドラッグ

マウスの左ドラッグは、一般的なグラフソフトと同様、範囲を選択し、その範囲を全画面に再プロットします。再プロットは新ウィンドウでは無く、同一のウィンドウとして行われます。

・右ドラッグ

右ドラッグでは、グラフ画面をつかむようにして、上下左右にプロット範囲を平行移動できます。

・左クリック/ダブルクリック

左クリックでは、クリックした点を中心として、約 1.7 倍拡大したグラフを再プロットします。また、ダブルでは、逆に約 1/1.7 に縮小したグラフを再プロットします。

・右クリック

右クリックでは、コピーや貼り付けなどを行えるメニューが表示されます。詳細は下記をご参照ください。

・ホイールスクロール

ホイールスクロールの操作では、左クリック/ダブルクリックよりも連続的に、拡大/縮小操作を行えます。

ワンポイント！

右クリックメニューを活用しましょう！

グラフ画面においてマウスの右ボタンをクリックすると、右クリックメニューが表示されます。右クリックメニューには、他のソフトと連携した作業において便利な機能をまとめてあります。右クリックメニューを積極的に活用する事で、資料作成を格段にスムーズに行う事ができます。

・Copy Image

画像をコピーし、別のソフトウェアにそのまま貼り付ける事が可能です。

・Paste Data

別のソフトウェアで選択コピーした数列データを、ファイルを作る事無く、そのままプロットする事が可能です。**「表計算ソフトなどでプロットしたい範囲を選択コピーし、リニアングラフに貼り付けてグラフ化する」**といった使い方が可能です。

4. 各メニュー機能解説

4-1. ファイル メニュー / File Menu

・ファイルを開く / Open File

座標値ファイルを開きます。一度に複数のファイルを指定可能です。「OPEN」ボタンを押してプロットしたい座標値ファイルを追加していき、最後に「PLOT」ボタンを押してください。

・データを開く / Open Data

ファイル化されていない数列データをそのままプロットする事ができます。「DATA:」と書かれている下の入力ウィンドウに、座標値ファイルの中身を貼り付けたり、表計算ソフトで領域をコピーして張り付けてください。「PLOT」を押すとグラフにプロットされます。
※この操作は、グラフ画面を右クリックすると出現するメニューから、「Paste Data」を選択しても行えます。

・画像の保存 / Save Image

現在のグラフ画面を画像ファイルに出力します。形式は BMP、JPEG、PNG が指定可能です。任意の名前を入力して「Save」ボタンを押してください。

画像は特に指定しない場合、開いている座標値ファイルのある場所に保存されます。任意の場所に保存したい場合は「Location」項目の「SET」ボタンを押し、保存場所を指定してください。

また、JPEG 形式画像ファイルでは、「Quality」項目で品質の指定が可能です。品質を上げるほど綺麗な画像に仕上がりますが、ファイルサイズが大きくなります。なお、BMP 及び PNG 形式では全く非劣しない完全品質で出力されます。

・設定の保存 / Save Setting

現在の設定を保存します。設定ファイルの保存場所は、以下から選択できます。

1: ソフトウェアの場所 / Software Directory
2: データファイルの場所 / Data-File Directory
3: ホームディレクトリ / Home Directory
4: クイック設定 / Quick Setting

1と 3 は、リニアングラフ2D の起動時に毎回自動で読み込まれるため、基本設定を保存するのに用います。 2 は現在開いているデータファイルと同じフォルダに保存され、そのフォルダ内のファイルを開いた際に読み込まれます。 4 は画面右上のクイック設定セレクタに設定を追加します。

4-2. 編集メニュー / Edit Menu

・クリア / Clear

現在のグラフ情報を全てクリアします。

・範囲の設定 / Set Range

グラフにプロットする範囲を指定します。X、Y それぞれに最小値と最大値を入力してください。

・色の設定 / Set Color

グラフの色を設定します。同時に複数の座標値ファイルを読み込んだ場合や、複数系列の座標値ファイルを読み込んだ場合に、ここでの設定に基づいて色分けされます。

・フォントの設定 / Set Font

グラフ画像中の目盛り表示やタイトル、ラベルなどに使用するフォントを設定します。一般にフォントは著作権により保護されており、個別にライセンスが存在します。**(重要)リニアングラフで作成した画像を、論文やインターネット等を通じて第三者に公開する場合は、その行為がライセンスにより許されているフォントを使用してください。**

なお、ソフトウェアとの相性により使用不可能なフォントが存在するため、設定したものと別のフォントが使用される事があります。リニアングラフで初めて使用するフォントがある場合、画像を公開する前に、設定したフォントが正しく使用されているかを必ずご確認ください。

・目盛りの設定 / Set Scale

X 方向と Y 方向への、目盛りの数を設定できます。さらに、「Format」項目で目盛り数字の書式を指定可能です。書式を指定するには、「Auto」項目を選択解除し、「Format=」の右にある入力エリアに、以下の例に基づいて入力してください。絶対値が 0.1 以上 10 以下の場合を独立に指定可能です。

入力例:

「 1.23 」のように表示したい場合、「 0.00 」と入力してください。

「 1.23E4 」のように表示したい場合、「 0.00E0 」と入力してください。

小数を切り捨てて「 1 」のように表示したい場合、「 0 」と入力してください。

・描画の設定 / Set Rendering

グラフスクリーンの幅と高さや、余白の設定などを行う事ができます。アンチエイリアス(文字や点・線などの描画の際、境界を滑らかにし、ピクセルによるギザギザを目立たなくする処理)の有効/無効もここで切り替えられます。

4-3. オプション メニュー / Option Menu

・線プロット / With Lines

座標点を線で結ぶオプションです。

・点プロット / With Points

座標点を点で描画するオプションです。

・ドットプロット / With Dots

座標点を微小な点で描画するオプションです。

・ボールプロット / With Balls

座標点を任意サイズの丸で描画するオプションです。

・対数軸 X, Y / Log X, Y

グラフを対数軸で表示するオプションです。**このモードを使用するには、座標値データが 0 や負の数を含んでいない必要があります。**

なおリニアングラフでは、対数軸の底が 10 であるか自然数（ネイピア数）であるかを気にする必要はありません。というのも、底の変換公式により、両者のグラフはそもそも比例関係にあり、目盛りの数字が異なるだけです。そしてリニアングラフでは、目盛りの数字にその位置の対数値では無く、その位置の真の値を表示する仕様になっています（例えば 1000 の位置には、3 ではなくそのまま 1000 と表示されます）。従って、対数軸の底によらず全く同じグラフとなるため、底を気にする必要が無いのです。

ワンポイント！ 対数軸表示で、目盛りの数字を綺麗に揃えるには

通常軸表示では、目盛りの数字が等差数列となるため、プロット範囲の最大値・最小値を綺麗な値に揃えていれば、目盛りの数字も綺麗な値に揃います。しかしこれを対数軸表示で行っても、端数の多い数字になってしまします。

これを解決するには、対数軸表示における目盛りの数字が等比数列となる事を利用します。まず、最大値・最小値を 10 の何乗かに設定しておきます。そして例えば最大値÷最小値が 10 の 5 乗になったとすれば、目盛りの本数を 5 本に設定します。こうすると、各目盛りが公比 10 の等比数列となるので、目盛り数字を綺麗に揃える事ができます。

・グレースクリーン / Gray　Screen

背景を灰色にするオプションです。OFF にすると白い背景になります。灰色の背景は液晶画面での作業に適しており、白色の背景は印刷に適しています。

・グリッド線 / Grid Line

背景にグリッド線（目盛りと目盛りを繋ぐ垂直・水平線）を描画するオプションです。

・ゼロレベル線 / Zero Line

X=0 及び Y=0 の垂直・水平線を描画するオプションです。

・フレーム / Frame

灰色の領域と余白との境界線を描画するオプションです。

・目盛り / Scale

目盛りを描画するオプションです。

・精密目盛り / 8byte-Scale

目盛りの表示桁数を、8Byte 浮動小数点数の有効桁（約 16 桁）まで表示するモードです。

4-4. ツールメニュー / Tool Menu

・アニメーション / Animation

グラフをアニメーションとして表現するツールです。メニューからこの項目を選択すると、アニメーションを制御するためのウィンドウが出現します。まず、アニメーションツールの下部にある「モード/MODE」の選択項目で、アニメーションの形式を選択してください。

「TRACE」: 始点から終点まで、徐々に描き足していくようにアニメーションします。
「INDEX」: 始点から終点まで、点を動かすようにアニメーションします。
「INDEX -SERIES」: 時刻ごとに系列を切り替えてアニメーションします。

なお、系列を時刻とするアニメーション(INDEX-SERIES)では、2 カラム書式の座標値ファイルにおいて、空白改行が系列の区切りと見なされます。マトリックス書式の座標値ファイルでは、異なる列に属する Y 値は別の系列と見なされます。

その後、「PLAY」ボタンで再生、「STOP」ボタンで一時停止ができます。一般的な動画プレイヤーのように、右側の大きなバーで任意の時刻に飛ぶ事ができます。またその下の小さなバーで再生スピードを調整できます。アニメーションを終了したい場合は、アニメーションウィンドウを閉じてください。

・数式プロット / Math

数式から座標値データを生成し、グラフにプロットするツールです。「Y(<X>)=」と書かれた入力ウィンドウ中に数式を記述して「PLOT」ボタンを押すと、その数式のグラフが描画されます。なお、数式中には X 軸変数を<X>と記述する必要があります。

※ 現在は「 プログラム / Program 」メニューから、新しい数式プロットツールが使用できます。このツールは旧バージョンですが、こちらの操作感に慣れた方のために互換サポートしています。

4-5. プログラム メニュー / Program Menu

リニアングラフ 2D が標準サポートしているスクリプト言語である、VCSSL で記述されたプログラムを実行します。プログラムはユーザーが作成可能で、リニアングラフ 2D を制御して、自動処理などを行わせる事ができます。詳細は後の章を参照してください。

VCSSL で記述したプログラムを「 RinearnGraph2DProgram 」フォルダ内に入れておくと、このメニューからすぐに選択して実行する事ができます。標準では、数式プロットツールとして使えるプログラムなどが同梱されています。

5. 座標値ファイル書式

この章では、リニアングラフ 2D で読み込み可能なファイルの書式（記述方法）について説明します。ここでの書式に基づいて座標値ファイルを作成してください。

ワンポイント！ 表計算ソフトからは、コピー&ペーストでグラフ化できる

グラフ作成に表計算ソフトを使用する場合、値の入力されている領域を選択してコピーし、リニアングラフ 2D の画面上で右クリックして、メニューから「Paste Data」を選ぶことで、ファイルを作らずにそのままグラフにプロットできます。

5-1. CSV ファイルと TSV ファイル

グラフ用の座標値ファイルには、座標値を並べた数列を記述します。このように数列をファイルに記述するには、主に CSV と TSV という 2 つのファイル形式があります※。

CSV 形式： データを「,（カンマ）」で区切って並べるファイル形式です。
一般に csv という拡張子が付きます。

TSV 形式： データをタブ文字（タブ空白）で区切って並べるファイル形式です。
拡張子は様々なケースがあります。

グラフ作成において、CSV は表計算ソフトで作成するデータなどでよく使用され、TSV はプログラミングなどで作成するデータなどでよく使用されます。リニアングラフ 2D では、CSV と TSV の両方に対応しています。

※ なお、厳密な TSV 形式では数値をタブ文字のみで区切りますが、数値ファイルではタブ文字と半角空白が混在していたり、逆に半角空白のみで区切ったり（いわゆる SSV）、空白スペースを並べてタブのように整形したファイルもよく存在します。以下では、簡単のためにそれらを区別せず一緒に扱います（全てリニアングラフ 2D で対応しています）。

5-2. マトリックス書式

CSV/TSV といった区切り文字の書式の他にも、グラフ用のデータには「数列をどう並べるか」といった点について、複数の書式が存在します。リニアングラフ 2D では、一般的に広く使用されている「マトリックス書式」及び「2 カラム書式」に対応しています。ここではまず、前者について解説します。

・マトリックス書式とは

「マトリックス書式」とは、最も左側の 1 列目に X 値数列を、そして 2 列目以降に Y 値数列を記載していく書式です。一つの列で一つの系列を表現します。マトリックス書式は、表計算ソフトなどで簡単に作成可能というメリットがあります。以下に、表計算ソフトでマトリックス書式のファイルを作成する手順を例示します。

・表計算ソフトでの作成例

まず表計算ソフトウェアを起動し、**最も左側の列に X 値を記述し、それより右の各列に各系列の Y 値を記述します。**必要であれば、最も上の行に系列名称を記述する事も可能です（省略可）。

図：表計算ソフトウェアの記述例

表計算ソフトウェア

年度	受注量	出荷量	純利益	成長率
2007	1.431	1.432	1.551	1.714
2008	1.881	1.883	1.772	1.701
2009	1.532	1.228	1.113	0.988
2010	1.111	1.233	1.453	1.882

上図において、赤い行が系列名称、青い列が X 値、そして緑の領域が各系列の（その行の X における）Y 値を意味します。上図のように記述し、ファイルを **CSV（カンマ区切り）形式**で保存してください（タブ区切りの TSV 形式で保存しても問題はありませんが、この書式では CSV を推奨します）。作成された CSV ファイルは、リニアングラフ 2D で読み込む事ができ、全てのプロットオプションが使用可能です。

5-3. 2 カラム書式

・2 カラム書式とは

表計算ソフトにおいてマトリックス書式がよく使用されるのに対して、プログラミング言語を用いた分野では、2 カラム書式がよく使用されます。2 カラム書式はファイル各行に左から X、Y と記載し、1 行につき 1 個の座標点を表現する書式です。

この書式は、プログラミングにより非常に効率的に作成できる事がメリットとして挙げられます。

2 カラム書式の例（タブ区切り）:

X1	Y1
X2	Y2
X3	Y3
…	…
X100	Y100

・複数系列の表現

2 カラム形式では、空白行を挟む事により、複数のデータを一枚のファイルに記載する事が容易にできます。座標値ファイル中に空白改行があると、そこが系列の区切りと見なされ、それ以降の座標値データは別系列のグラフのものとして読み込まれます。系列はいくつでも使用可能です。

アニメーションモードでは、複数系列のグラフを時系列と見なして再生します。つまり各時刻ごとに対応する系列のグラフがのみ描画されます。例えば系列が 101 個あるデータなら、時刻0には第 0 系列のみが、時刻 1 には第 1 系列のみが…といった具合で時刻 100 まで順々に描画されます。

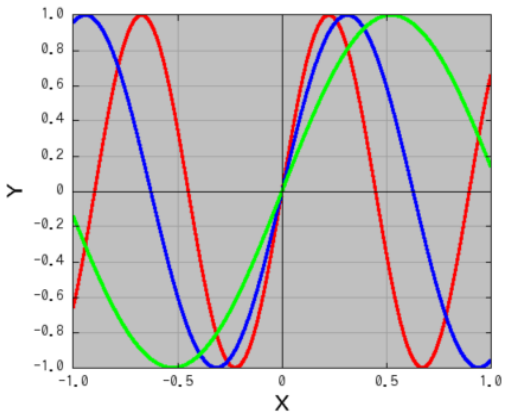
6. 数式のグラフ描画

リニアングラフ 2D では、データファイルからだけではなく、数式からグラフを描画する事もできます。これを数式プロットと呼びます。

6-1. 数式プロットツール

数式プロットで使用するツールは、リニアングラフで標準サポートしているプログラミング言語「 VCSSL 」(8 章参照)で開発されており、グラフ画面上部の「 プログラム(プログラム) 」メニューから選んで起動できます。数式の形に応じて、複数のツールがあります。これらは単体でも公開しているもので、Web ページ上で詳しい使用方法も解説しています。一覧と概要は以下の通りです。

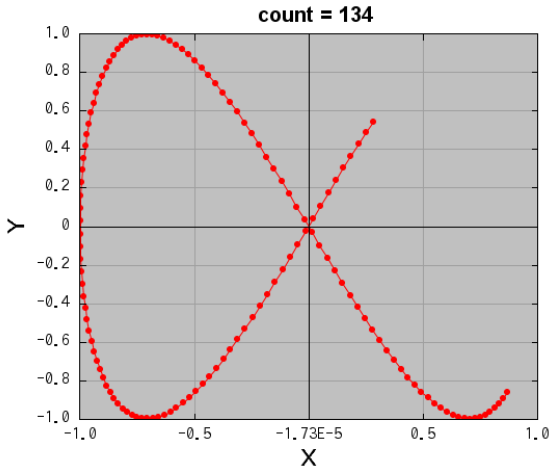
・y(x)形式の 2D グラフ描画

概要	y(x)形式の数式を、2D の線グラフとして描画します。 
使用方法	起動後、入力画面の「 y(x) = 」の欄に数式を入力して「 プロット 」ボタンを押すと、その数式が 2D グラフとして描画されます。範囲やプロット密度なども指定可能です。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/5000-graph2d-input-yx/

・y(x,t)形式の 2D グラフ描画

概要	y(x,t)形式の数式を、2D の線グラフとしてアニメーション描画します。
使用方法	起動後、入力画面の「 y(x,t) = 」の欄に数式を入力して「 プロット 」ボタンを押すと、その数式が 2D グラフとして描画され、「 t 」を時刻変数としてアニメーション表示されます。範囲やプロット密度なども指定可能です。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/5400-graph2d-input-yxt/

・x(t),y(t)形式の 2D グラフ描画

概要	x(t),y(t)形式の数式を、t を媒介変数として 2D グラフにアニメーションします。 
使用方法	起動後、入力画面の「 x(t) = 」および「 y(t) = 」の欄に数式を入力して「 プロット 」ボタンを押すと、その数式が「 t 」を媒介変数として 2D グラフに描画されます。そこで「 アニメーション 」ボタンを押すと、グラフを始点から終点へ徐々に描いていくように、アニメーション再生されます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/5200-graph2d-input-xt-yt/

・旧バージョンの数式プロットツール

必要に応じて、グラフ画面上部の「 ツール 」>「 (旧)数式プロット 」メニューから、リニアングラフが VCSSL をサポートする前から搭載されていた、旧バージョンの数式プロットツールも使用できます。そのツールでは y(x) 形式のグラフしか描けず、また数式内の x を < > で囲って <x> のように書かなければいけませんが、昔からご使用されている方を想定して、今も使用可能な状態になっています。この旧ツールでは、log 関数が自然対数の ln ではなく常用対数の log10 と同じになっている事に注意が必要です(新しいツールでは log10 と同じです)。

6-2. 数式の書き方と関数一覧

各数式プロットツールでは、入力画面の「 y(x) = 」などの数式欄に、基本的には普通の書き方で数式を入力してください。ただし、かけ算は「 * 」、割り算は「 / 」記号を使います。かけ算・割り算は、足し算・引き算よりも先に計算されます。足し算・引き算を先に計算したい場合は、その部分を、例えば「 (y + 2) * 3 」のようにかっこ ( ) で囲ってください。

数式内では、例えば「 sin(x) + cos(2 * x) 」のように、一般的な数学関数を使用できます。サポートされている関数の一覧は以下の通りです:

sqrt( )

二乗根、ルート

exp( )

指数関数

Log10( )

10 を底とした対数関数

※「ツール」メニューの旧版の数式プロットツールでは、単に log とした場合も log10 と同等となります。しかし、「プログラム」メニューの新しい数式プロットツールでは（VCSSLの仕様に従うため）、logはlnと同等となります。これに起因する混乱を避けるため、現在では単なる log は使わず、log10 と ln の使い分けが推奨されます。

ln( )

自然数を底とした対数関数

abs( )

絶対値

fac( )

階乗演算　※「ツール」メニューの旧版の数式プロットツールでは、!( ) を使用して下さい。

**

べき乗演算　※使い方は正なら 2**3、負の数なら(-2)**(-3) など。

※「ツール」メニューの旧版の数式プロットツールでは、2^3 のように「^」と記述して下さい。

sin( )

サイン/三角関数

cos( )

コサイン/三角関数

tan( )

タンジェント/三角関数

asin( )

アークサイン/逆三角関数

acos( )
アークコサイン/逆三角関数

atan( )
アークタンジェント/逆三角関数

sinh( )
ハイパボリックサイン/双曲線関数

cosh( )
ハイパボリックコサイン/双曲線関数

tanh( )
ハイパボリックタンジェント/双曲線関数

asinh( )
ハイパボリックサインの逆関数/逆双曲線関数

acosh( )
ハイパボリックコサインの逆関数/逆双曲線関数

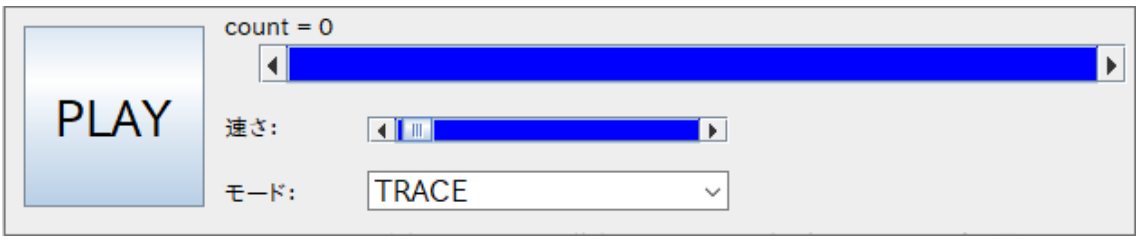
atanh( )
ハイパボリックタンジェントの逆関数/逆双曲線関数

7. アニメーション

リニアングラフ 2D では、グラフをアニメーション描画する事もできます。

7-1. 単一ファイル内のデータのアニメーション

単一のファイルに記載されたデータを、系列を時刻とみなしたり、座標点の順序を時刻と見なしたりしてアニメーションさせる事ができます。それにはまず、グラフ画面上部の「 ツール（ツール） 」>「 アニメーション 」メニューを選択して、アニメーションツールを起動してください。



続いて、アニメーションツールの下部にある「MODE/モード」の選択項目で、アニメーションの形式を選択してください。

「INDEX」: 曲線/点プロットにおいて、始点から終点までをアニメーションします。
「TRACE」: 曲線/点プロットにおいて、始点から終点までをアニメーション（上描き）します。
「INDEX -SERIES」: 時刻ごとに系列を切り替えてアニメーションします。

その後、「PLAY」ボタンを押すとアニメーションが開始されます。もう一度押すと一時停止となり、さらにもう一度押すと再開されます。アニメーションを終了したい場合は、アニメーションウィンドウを閉じてください。

なお、複数系列のアニメーションを行う場合は、あらかじめデータを複数系列の書式で記載しておく必要があります。詳細は「 5. 座標値ファイル書式 」の章をご参照ください。例えば、「 x y 」の形で 1 行に着き 1 つの座標点を書いていく 2 カラム書式では、空白行をはさむと、そこが系列の区切りと見なされます。表のようにデータを書いていくマトリックス書式も使用できますが、アニメーション用のデータは大量になるため、プログラミングなどで自動生成するには 2 カラム書式がおすすめです。

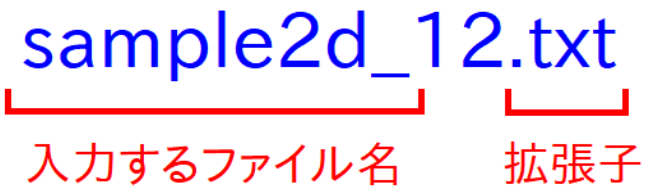
7-2. 連番の複数ファイルのアニメーション

また、例えば「 sample2d_0.txt 」「 sample2d_1.txt 」「 sample2d_2.txt 」… などのように、連番のファイル名が付いた複数のファイルを、次々と読み込んで高速にグラフにプロットする事で、アニメーション描画を行う事もできます。

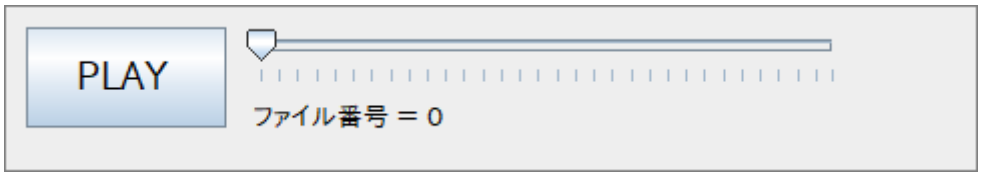
それにはまず、グラフ画面上部の「 プログラム（Program） 」メニューから、「 **連番 2D データファイルのアニメーション** 」を選択してください。

するとツールの処理が起動し、まずデータファイルを読み込むフォルダを尋ねられるため、ある場合は選択してください。無い場合は「 いいえ 」を押すと、「 RinearnGraph2DProgram 」フォルダ内の「 animation2d_input 」フォルダが使用されますが、標準ではこの中にサンプルデータファイルが入っています。

続いて、ファイル名と拡張子を尋ねられるため、入力してください。ここでファイル名には、連番の番号部分を除いた部分（下図）を指定します。



あとは自動で連番の始点と終点が検索され、以下のアニメーションツールの画面が起動します：



基本的には先の単一ファイルのアニメーションと同様、「PLAY」ボタンを押すと再生開始、「STOP」ボタンで停止します。詳細な使用方法については以下の Web ページをご参照ください：

https://www.vcssl.org/ja-jp/code/archive/0001/7200-graph-file-animator-2d/

なお、このツールでは「 画像保存 」ボタンを押す事で、アニメーションの各コマを、連番の画像ファイルとして保存する事もできます。保存した画像をアニメーション再生するためのツールも、同じく「 プログラム（Program） 」メニューに「 **連番画像ファイルのアニメーション** 」として同梱されています。こちらについての詳しい使用方法は以下の Web ページをご参照ください：

https://www.vcssl.org/ja-jp/code/archive/0001/4100-image-file-animator/

8. VCSSL による制御・自動処理

リニアングラフ 2D は、プログラミング言語 VCSSL による制御をサポートしています。うまく活用すると、連番ファイルを自動処理でプロットさせたり、読み込んだデータを変換（加工）してプロットさせる事などができます。

8-1. VCSSL について

VCSSL は、日常計算から 3 次元コンピューターグラフィックスまでを幅広くカバーするプログラミング言語です。C 言語系のシンプルな文法を採用しており、容易に習得する事ができます。

プログラミング言語 VCSSL　公式サイト
https://www.vcssl.org/ja-jp/

VCSSL のランタイムは、リニアングラフ 2D に同梱されており、リニアングラフ 2D の「 Program / プログラム 」メニューから手軽に利用する事ができます。

8-2. プログラムの作成と実行

VCSSL はスクリプト言語なので、プログラムは一般のテキストエディタで記述し、拡張子「.vcssl」を付けて保存するだけで作成できます。作ったプログラムは、リニアングラフ 2D のメニューバーにある、**「 Program / プログラム 」** メニューから選択実行できます。

なお、便利なプログラムを作成した場合、**「 RinearnGraph2DProgram 」**フォルダに入れておくと、「 Program / プログラム 」メニューに表示され、すぐに実行できるようになります。

8-3. Graph2D API について

VCSSL からリニアングラフ 2D を制御するには、tool.Graph2D API を使用します。 ここでは Graph2D API の一部の機能のみを使用します。全機能を参照するには、下記 URL をご参照ください。

tool.Graph2D API 詳細仕様
https://www.vcssl.org/ja-jp/lib/tool/Graph2D

8-4. プログラム例

・ファイルをプロットする

以下は座標値データファイル「test.tsv」をプロットする、最も単純なプログラムです。
なお、「//」で始まる行はコメントであり、処理の際には読み飛ばされます。

```
//リニアングラフ 2D を制御するライブラリをインポート
import tool.Graph2D;

//リニアングラフ 2D を起動（ x 位置, y 位置, 幅, 高さ, タイトル ）
int graph = newGraph2D( 0, 0, 700, 700, "Graph" ) ;

//座標値データファイル「 test.tsv 」をプロットする
setGraph2DFile( graph, "test.tsv" );
```

・線プロットオプションを設定する

続いて、点プロットオプションを無効にし、線プロットオプションのみを有効にするプログラムです。

```
import tool.Graph2D;

int graph = newGraph2D( 0, 0, 700, 700, "Graph" ) ;
setGraph2DFile( graph, "test.tsv" );

//点プロットを無効にする
setGraph2DOption( graph, "WITH_POINTS", false );

//線プロットを有効にする
setGraph2DOption( graph, "WITH_LINES", true );
```

・グラフを画像ファイルに出力する

以下はグラフを画像ファイル「test.png」に出力するプログラムです。

```
import tool.Graph2D;

int graph = newGraph2D( 0, 0, 700, 700, "Graph" ) ;

setGraph2DFile( graph, "test.tsv" );

//グラフを画像ファイル「 test.png 」に出力（ グラフ ID, 画像ファイル名, 画像形式 ）
exportGraph2D( graph, "test.png", "PNG" );
```

・連番の座標値データファイルを次々とプロットし、連番の画像ファイルに出力する

続いて応用です。VCSSL プログラムから見て「test」フォルダの中にある、連番の座標値データファイル「 test0.tsv 」～「 test100.tsv 」を次々とプロットし、画像ファイル「 test0.png 」～「test100.png」に出力していくプログラムです。

```
import tool.Graph2D;

int graph = newGraph2D( 0, 0, 700, 700, "Graph" ) ;

//変数 i を 1 から 100 まで変更しながらループ
for( int i=0; i<=100; i++ ){

   setGraph2DFile( graph, "./test/test"+i+".tsv" );
   exportGraph2D( graph, "./test/test" + i + ".png", "PNG" );

   //100 ミリ秒スリープ（アニメーションウェイト）
   sleep( 100 );

}
```

・座標値データファイルを読み込み、値を変換してプロットする

　最後に、座標値データファイル「test.tsv」を読み込み、Y 値を sin(Y 値)に変換した上でプロットするプログラムです。最初に、「test.tsv」が**空白改行を含む場合**を扱います。

```
import tool.Graph2D;
import Math;

//  座標値データファイル「test.tsv」を、TSV 形式の読み込みモードで開く
int file = open( "test.tsv", READ_TSV );

//  ファイル行数を取得
int lineN = countln( file );

double value[]; // 1 行の X、Y 値を控える数値配列
string newData = ""; //  新しいデータを控える文字列変数

for( int i=0; i<lineN; i++ ){
   value = readln( file );    //1 行のデータを配列で取得し、次の行へ
   if( length(value,0) == 0 ){
      //空白行はそのまま改行 （EOL はシステムによって定義されている改行コード値）
      newData += EOL;
   }else{
      //空白でない行は Y 値を sin(Y 値)に変換して書き出し、改行
      newData += value[0] + "          " + sin( value[1] ) + EOL;
   }
}

//ファイルアクセスを閉じる
close( file );

int graph = newGraph2D( 0, 0, 700, 700, "Graph" );

//変換後の文字列データからグラフにプロット
setGraph2DData( graph, newData );
```

上述のプログラムは、文字列の結合処理を伴うために低速です。test.tsv が**空白改行を含まない場合**は、以下のように数値の配列を直接リニアングラフ 2D へ転送する方が、はるかに高速です。

```
import tool.Graph2D;
import Math; //sin 新関数を使用するのに必要

//  座標値データファイル「test.tsv」を、TSV 形式の読み込みモードで開く
int file = open( "test.tsv", READ_TSV );

//  ファイル行数を取得
int lineN = countln( file );

double value[ ]; // 1 行の X、Y、Z 値を控える数値配列

double x[lineN]; //  全行の X 値を控える数値配列
double y[lineN]; //  全行の X 値を控える数値配列

for( int i=0; i<lineN; i++ ){

   //1 行のデータを配列で取得し、次の行へ
   value = readln( file );
   x[i] = value[0];
   y[i] = sin( value[1] ); // Y 値は sin(Y 値)に変換

}

//ファイルアクセスを閉じる
close( file );

int graph = newGraph2D( 0, 0, 700, 700, "Graph" );

//変換後の数値データ配列からグラフにプロット
setGraph2DData( graph, x, y );
```

今度の処理は非常に高速で、起動とほぼ同時に変換が完了します。

9. Java 言語による制御・自動処理

リニアングラフ 2D は、前章で扱った VCSSL に加えて、Java 言語での制御もサポートしています。

9-1. 開発の準備とコンパイル・実行

Java 言語でリニアングラフ 2D を制御するには、別途 Java 言語の開発環境（JDK）が必要です。JDK の入手やインストールについては、Java 言語の解説書や解説サイトなどをご参照ください。なお、ここでは解説の単純化のため IDE は使用せず、コンパイルと実行はコマンドラインで行います。

はじめに、環境が揃っている事の確認として、リニアングラフ 2D を起動するための簡単なサンプルを作成し、コンパイル・実行してみましょう。「Sample0.java」という名前で、以下のコードを記述したテキストファイルを作成してください：

```
import com.rinearn.graph2d.RinearnGraph2D;

public class Sample0 {
        public static void main(String[] args) {

                // グラフを起動
                RinearnGraph2D graph = new RinearnGraph2D();
        }
}
```

続いて、リニアングラフ 2D の配布パッケージ内にある「 RinearnGraph2D.jar（JAR ファイル）」を、上のコードと同じ場所に置いてください。そして、コマンドライン端末上でその場所に cd などで移動し、以下のように入力してコンパイルしてください：

```
javac -classpath ".;RinearnGraph2D.jar" Sample0.java        (Windows の場合)
javac -classpath ".:RinearnGraph2D.jar" Sample0.java        (Linux 等の場合)
```

※ 上の二つは、-classpath の後の部分での区切り文字が、「 ; 」と「 : 」で異なっています。

エラー無くコンパイルできた事を確認した上で、そのまま以下のように入力して実行してください：

```
java -classpath ".;RinearnGraph2D.jar" Sample0        (Windows の場合)
java -classpath ".:RinearnGraph2D.jar" Sample0        (Linux 等の場合)
```

実行の結果、グラフ画面が起動すれば成功です。以降のサンプルコードでは、上のコマンドの Sample0 の部分を Sample1～Sample6 に置き換えて、同様にコンパイル・実行できます。その際、ファイル名は必ず「 クラス名.java 」とするようにご注意ください。

この章で使用するリニアングラフ 2D の Java 言語用 API ライブラリは、以下の URL で仕様書を参照できます。この章では触れきれない細かい機能の一覧や詳細などは、そちらをご参照ください：

・ **リニアングラフ 2D　API 仕様書（ Java 言語用 ）**
https://www.rinearn.com/ja-jp/graph2d/api/

API ライブラリの中でも特に使用する RinearnGraph2D クラスの仕様書は、以下で参照できます：
https://www.rinearn.com/ja-jp/graph2d/api/com/rinearn/graph2d/RinearnGraph2D

以下では、いくつかの基本となる使い方について、サンプルコードを例示しながら解説していきます。なお、以下で掲載するサンプルコードは、配布パッケージ内にも同梱されています。

9-2. ファイルのプロット

まずは、実用における最も単純な例として、CSV ファイルに記載された座標値データをプロットしてみましょう。以下のサンプルデータファイルを使用します：

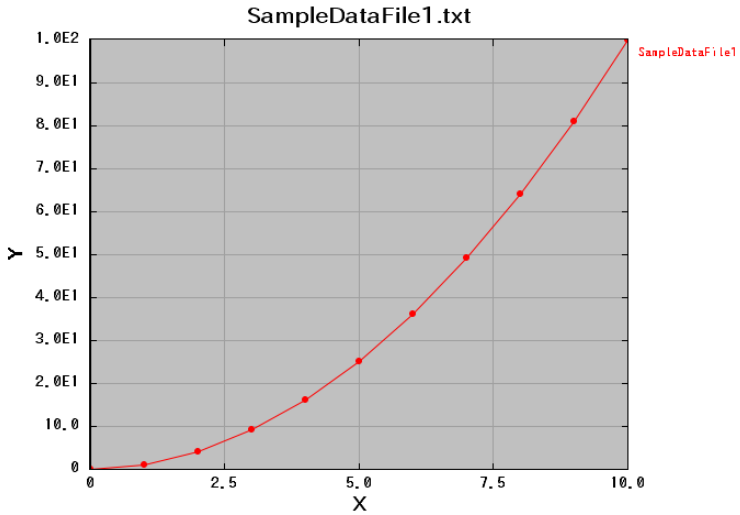
- SampleDataFile1.csv -

```
0.0,0.0
1.0,1.0
2.0,4.0
3.0,9.0
4.0,16.0
5.0,25.0
6.0,36.0
7.0,49.0
8.0,64.0
9.0,81.0
10.0,100.0
```

この通り、2 カラム書式（左から x　y ）で、0 <= x <= 10 の範囲で関数 $y = x^2$ の座標値が書き出されています。このファイルを開いてグラフにプロットするプログラムは以下の通りです：

```
import com.rinearn.graph2d.RinearnGraph2D;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Sample1 {
        public static void main(String[] args) {
                // グラフを起動
                RinearnGraph2D graph = new RinearnGraph2D();
                try {
                        // データファイルを読み込んでプロット
                        graph.openDataFile(new File("SampleDataFile1.csv"));

                // 異常時の例外処理
                } catch (FileNotFoundException fnfe) {
                        System.err.println("ファイルが見つかりませんでした。");
                } catch (IOException ioe) {
                        System.err.println("ファイルが開けませんでした。");
                }
        }
}
```

▼実行結果



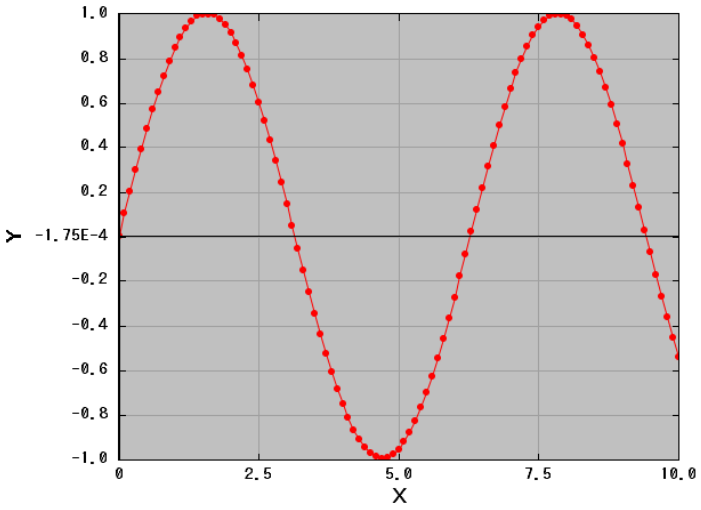
上のコードでは、まず RinearnGraph2D クラスのインスタンスを生成し(この時点でグラフ画面が起動します)、そして openDataFile メソッドを呼び出して、開きたいファイルを引数に渡しています。ファイルを開く過程での例外については、必要に応じて適切に処理してください。

9-3. 配列データのプロット

ファイルを介さず、配列に格納された座標値データを、直接渡してプロットする事もできます：

```
import com.rinearn.graph2d.RinearnGraph2D;

public class Sample2 {
        public static void main(String[] args) {
                // 座標値データの用意
                int n = 101;
                double[] x = new double[n];
                double[] y = new double[n];
                for(int i=0; i<n; i++) {
                        x[i] = i*0.1;
                        y[i] = Math.sin(x[i]);
                }

                // グラフにプロット
                RinearnGraph2D graph = new RinearnGraph2D();
                graph.setData(x, y);
        }
}
```

▼実行結果



※ Ｙ軸の中心が０になっていないのは、上下の端が僅かに±1 からずれているため、後で扱うようにプロット範囲を設定すると、綺麗に揃えられます。

このように、先ほどファイルを開いていた openDataFile メソッドの代わりに、setData メソッドで配列データを渡します。

配列データは、ここでは単系列なので x[座標点インデックス] の形ですが、系列を分けたい場合は x[小系列インデックス][座標点インデックス] の形の 2 次元配列にします（y も同様）。プロット結果では系列の区切りで色が変わり、また、線プロット時では系列の区切りで線が切れます。

なお、上のコードで、座標値データの点数 n を 101 と半端な値にしているのは、単純に端の点での X 値をきりのいい値にするためで、重要な意味はありません（※ X の値を単純に配列インデックス×0.1 としているので、インデックスの範囲が 0～100 ならちょうどきりがいいのですが、そのために必要な配列要素数は 101 個です）。

9-4. 画像の出力

グラフを画像ファイルに出力するには、RinearnGraph2D クラスの exportImageFile メソッドを使用します。例えば、8-2 や 8-3 のサンプルコードにおいて、main メソッドの末尾に以下の一行を追加すれば、画像ファイル「graph.png」が出力されます：

```
		try {
			graph.exportImageFile(new java.io.File("graph.png"), 1.0);
		} catch (java.io.IOException ioe) {
			System.err.println("ファイル出力に失敗しました。");
		}
```

画像形式は拡張子から自動で判断されますが、現時点では JPEG と PNG 形式にのみ対応しています。2 番目の引数は画質で、1.0 で最高、0.0 で最低となります（JPEG 形式のみで有効です）。

なお、getImage メソッドにより、グラフ画像を java.awt.Image クラスのインスタンスとして取得する事もできます。これにより、グラフ画像を加工したり、別の GUI 画面上に埋め込んだりする事も（それなりの処理を書く必要がありますが）可能です。

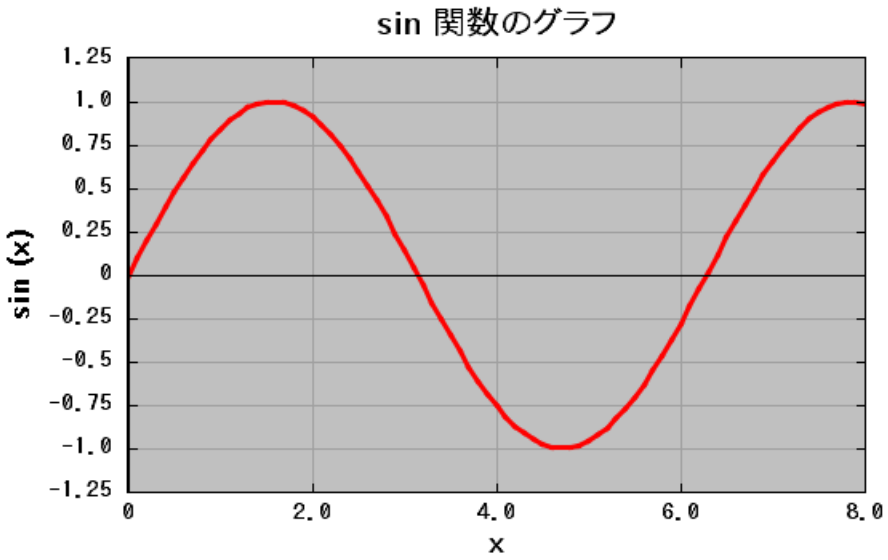
9-5. 範囲やオプションなどの詳細設定

setX/YRange メソッドや setOptionSelected メソッドなどを用いて、プロット範囲やオプションなどの設定を行う事ができます。例として、先ほどと同様の配列データをプロットし、点プロットオプションを無効化しつつ線プロットオプションは有効である状態にして、範囲も設定してみましょう：

```
import com.rinearn.graph2d.RinearnGraph2D;
import com.rinearn.graph2d.RinearnGraph2DOptionItem;
```

```
public class Sample3 {
        public static void main(String[] args) {
                // 座標値データを用意し、グラフに渡してプロット
                int n = 101;
                double[] x = new double[n];
                double[] y = new double[n];
                for(int i=0; i<n; i++) {
                        x[i] = i*0.1;
                        y[i] = Math.sin(x[i]);
                }
                RinearnGraph2D graph = new RinearnGraph2D();
                graph.setData(x, y);

                // プロット範囲を設定
                graph.setXRange(0.0, 8.0);
                graph.setYRange(-1.25, 1.25);

                // グラフスクリーンのサイズを設定
                graph.setScreenSize(680, 380);

                // タイトルとラベルを設定
                graph.setTitle("sin 関数のグラフ");
                graph.setXLabel("x");
                graph.setYLabel("sin (x)");

                // プロットオプションを選択（点プロット無効化と線プロット有効化）
                graph.setOptionSelected(RinearnGraph2DOptionItem.POINT, false);
                graph.setOptionSelected(RinearnGraph2DOptionItem.LINE, true);

                // プロットオプションのパラメータを設定（線プロットの線幅を指定）
                RinearnGraph2DOptionParameter parameter
                                = new RinearnGraph2DOptionParameter();
                parameter.setLineWidth(3.0);
                graph.setOptionParameter(parameter);
        }
}
```

▼実行結果



上のコードでは、グラフに座標値データを渡してプロットさせた後、まず setXRange / setYRange メソッドでそれぞれ X/Y 軸方向のプロット範囲を設定しています。

続いて、setScreenSize メソッドでグラフスクリーンの幅と高さを設定しています。グラフスクリーンは、グラフが描画されているスクリーンの事で、グラフを画像ファイルに出力する際などには、ここで指定したサイズが出力画像のサイズになります。

その後は、setOptionSelected メソッドでプロットオプションの有効/無効状態を操作しています。操作するオプションの項目は RinearnGraph2DOptionItem 列挙子の要素で指定します。上では点プロットオプションを無効化し、線プロットオプションを（こちらはデフォルトでも有効ですが）有効化しています。また、プロットオプションでの線の太さなどの詳細設定を、setOptionParameter メソッドで行っています。このメソッドには、まず RinearnGraph2DOptionParameter クラスのインスタンスを生成して設定内容を保持させた上で、それを引数に渡します。線の太さの他に、点プロットでの点の大きさなども設定できます。

設定系のメソッドは他にも存在します。詳細については、RinearnGraph2D クラスの API 仕様書をご参照ください。なお、細かい設定を一括で行うには、リニアングラフ 2D の設定ファイルを loadConfigurationFile メソッドで読み込む方法もあります。設定周りの処理があまり多くなってくると、コードをシンプルにする上でも、設定ファイル経由の方が便利です。また、API では細かすぎて未サポートになっているような設定も、設定ファイル経由なら行えます。一方で、設定ファイルの記載内容やその有効性は、（できるだけ互換は保たれますが）リニアングラフ 2D のバージョンに依存する事に留意する必要があります。

9-6. アニメーションプロット

アニメーションプロットを行うには、スレッドを生成して、その中で 8-2 や 8-3 で扱ったファイルや配列データのプロットを繰り返し、少しずつ異なる内容を連続で描かせ続ければ OK です。

ただし標準では、setData メソッドで配列データを渡した際、データの描画処理が完了するまで、呼び出し元に処理が戻らない事に留意する必要があります。これは、アニメーションの各画面をコマ落ちなく画像に出力したい場合などには便利です。しかし、例えば外部の装置などから入力されるデータをリアルタイムでプロットしたい場合などには、次々と入ってくるデータに対して描画速度が追い付かない、つまり描画がボトルネックになってしまう可能性もあります。そのような場合には、setAsynchronousPlottingEnabled メソッドの引数に true を指定すると、setData メソッド呼び出し時はすぐに処理が戻り、描画処理は別スレッドで非同期に、適当なタイミングで行われるようになります。これは、一般にアニメーション速度を描画所要時間に依存させたくない場合にも有効です。

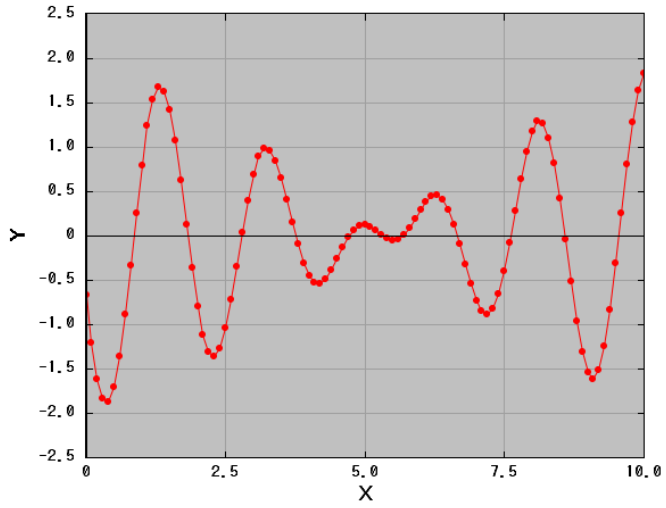
実際に、毎時刻の配列データを計算で生成し、非同期描画でアニメーションさせてみましょう:

```
import com.rinearn.graph2d.RinearnGraph2D;
import com.rinearn.graph2d.RinearnGraph2DOptionItem;
import java.awt.event.WindowListener;
import java.awt.event.WindowEvent;
public class Sample4 implements Runnable, WindowListener {
        Thread thread = null;         // アニメーション用のスレッド
        RinearnGraph2D graph = null;  // グラフ
        volatile boolean continuesLoop = true;     // ループの継続/終了を制御する
        public static void main(String[] args) {
                Sample4 sample = new Sample4();
        }
        // 初期化・実行開始処理
        public Sample4() {
                // グラフを起動し、描画範囲の設定と自動調整機能の無効化を行う
                this.graph = new RinearnGraph2D();
                graph.setXRange(0.0, 10.0);
                graph.setYRange(-2.5, 2.5);
                this.graph.setXAutoRangingEnabled(false);
                this.graph.setYAutoRangingEnabled(false);
                // データ更新と描画処理の関係を非同期にする(リアルタイムアニメーション用)
                //(※ 連番で画像出力する用途などでは行わない方がコマ落ちや欠けを防げる)
                this.graph.setAsynchronousPlottingEnabled(true);
```

```
            // グラフを閉じたら独自終了処理を行うリスナーを登録、デフォルト処理は無効化
            this.graph.addWindowListener(this);
            this.graph.setAutoDisposingEnabled(false);
            // アニメーションスレッドを生成して実行開始
            this.thread = new Thread(this);
            this.thread.start();
    }
    // アニメーションスレッドの処理
    @Override
    public void run() {
            int n = 101;
            double[] x = new double[n];
            double[] y = new double[n];

            // アニメーションループ（continuesLoop が true の間継続）
            for(int frame=0; this.continuesLoop; frame++) {
                    double t = frame * 0.05; // 時刻変数

                    // 座標値データを更新してグラフに転送（非同期）
                    for(int i=0; i<n; i++) {
                            x[i] = i * 0.1;
                            y[i] = Math.sin(3.0*x[i]+1.5*t)+Math.cos(3.5*x[i]+t);
                    }
                    this.graph.setData(x, y);

                    // 50 ミリ秒だけ停止(時間は適時調整)
                    try {
                            Thread.sleep(30);
                    } catch(InterruptedException e) {
                            // 割り込み例外の処理
                    }
            }
            // アニメーションループが終了したらグラフを破棄
            this.graph.dispose();
            // スレッド処理終端: 他にスレッド・リソースが残っていなければ自然に実行終了
    }
```

```
        // ※ 以下のようなイベント処理の実装が面倒な場合は、多少強引でよければ、
        // this.graph.setAutoExittingEnabled(true); により、グラフを閉じたら
        // アプリケーションの実行を即終了するよう設定可能です（即席の場合向けです）。

        // グラフのウィンドウが閉じられた際に行うイベント処理
        @Override
        public void windowClosing(WindowEvent e) {
                // アニメーションループを脱出させ、スレッドを終了させる（結果、実行も終了）
                this.continuesLoop = false;
        }
        // その他のウィンドウイベント処理（ここでは何もしない）
        @Override
        public void windowDeactivated(WindowEvent e) { }
        @Override
        public void windowActivated(WindowEvent e) { }
        @Override
        public void windowDeiconified(WindowEvent e) { }
        @Override
        public void windowIconified(WindowEvent e) { }
        @Override
        public void windowOpened(WindowEvent e) { }
        @Override
        public void windowClosed(WindowEvent e) { }
}
```

▼実行結果（干渉する sin 波のグラフがプロットされ、形状がアニメーションで時間変化します。）



9-7. 描画エンジンの直接操作による描画

これまでの内容では、リニアングラフに渡した配列データやファイルから、オプション設定などに応じて、ある程度「おまかせ」でグラフを描画してもらっていました。これは手短で簡単なのですが、その代わり自由度には限界があります。

例えば、「ここに、こういう大きさと色の点を描きたい」 や 「こことここを、こういう色と太さの線で結びたい」 といったように、もっと細かく描画内容を制御したい場合があるかもしれません。そのような用途のために、グラフの描画エンジンを直接操作する事もできます。そのための API は RinearnGraph2DRenderer クラスとして提供されており、以下で仕様書を参照できます:

- **RinearnGraph2DRenderer クラス API 仕様書**

  https://www.rinearn.com/ja-jp/graph2d/api/com/rinearn/graph2d/renderer/RinearnGraph2DRenderer

・簡単な例

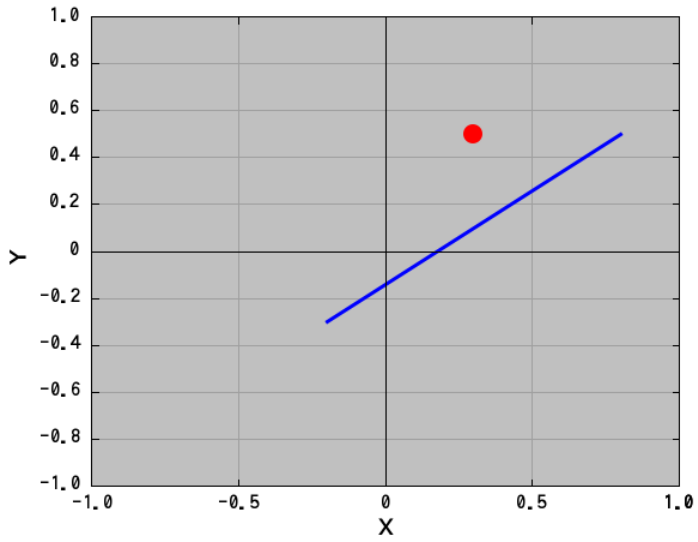
描画エンジンの処理は、もちろんリニアングラフ 2D 側からも呼び出されます。そのため、ここで扱う描画エンジンの直接操作と、リニアングラフ 2D の他の機能（ファイルを開いてプロットしたり、メニューから手動でグラフ範囲やプロットオプションを変更したり、等々）とを併用したい場合には、少し工夫が必要です。

詳しくは次の項で説明しますが、まずはそのような事は一旦置いておいて、とりあえず一番簡単な方法で、点や線を描いてみましょう。

```
import com.rinearn.graph2d.RinearnGraph2D;
import com.rinearn.graph2d.RinearnGraph2DOptionItem;
import com.rinearn.graph2d.renderer.RinearnGraph2DRenderer;
import java.awt.Color;

public class Sample5 {
        public static void main(String[] args) {

                // グラフを起動してレンダラー（描画エンジン）を取得
                RinearnGraph2D graph = new RinearnGraph2D();
                RinearnGraph2DRenderer renderer = graph.getRenderer();
```

```
                // グラフ領域内の(0.3, 0.5)の位置に、半径 8 ピクセルで赤色の点を描画
                renderer.drawPoint(0.3, 0.5, 8.0, Color.RED);

                // グラフ領域内の(-0.2,-0.3)と(0.8,0.5)を結ぶ、
                // 太さ 3 ピクセルで青色の線を描画
                renderer.drawLine(-0.2, -0.3, 0.8, 0.5, 3.0, Color.BLUE);

                // スクリーンの再描画（レンダリング）
                renderer.render();
        }
}
```

▼実行結果



上のコードでは、まずグラフを生成して getRenderer メソッドで描画エンジンを取得し、そしてその描画エンジンの各描画メソッドを呼び出して、点や線などを描いています。

最後に、render メソッドでスクリーンの再描画を行わせていますが、これを忘れるとリニアングラフ 2D の画面が更新されず、従ってせっかく描いた点や線も表示されないのでご注意ください。スクリーンの再描画は、マウスカーソルがグラフ画面上で動いた場合など、必要に応じて自動で行われるタイミングもありますが、しかし点や線を 1 つ 1 つ描くごとに毎回行われたりはしません。従って、描きたい内容を全部描き終えた後で、このように明示的に render メソッドで行ってください。

drawPoint などの各描画メソッドの引数に渡す座標は、グラフ空間における座標と見なされます。そのため、描画される絶対的な位置はグラフの範囲設定によって変化します。また、グラフの範囲より外側にはみ出した内容は描画されません。

・グラフ範囲やプロットオプションを変更しても描画内容が消えないようにするには

ところで、上のサンプルコードでは、後からグラフ範囲やプロットオプションの変更操作などを行うと、描画内容が消えてしまいます。これは、マウス操作で画面をスライドさせたり、拡大・縮小させたりした際についても同様です。それで消えてしまうのは、ちょっと不便ですね。他にもファイルのデータを開いてプロットした際にも消えてしまうので、「 データに補助線などを重ねて描きたい 」といった場合にも不便です。

実は上のような操作は、ファイルや配列データから描画されている他の内容なども含めて、グラフ全体の描きなおし（再描画）が必要になるため、最初にリニアングラフ 2D 側から描画内容をリセットする処理が呼ばれます。その際に、描画エンジンを操作して描いた点や線も消えてしまうわけです。

そこで、グラフの再描画が必要なタイミングをイベントとして受け取って、その度に先ほどのコードの点や線も描画しなおすようにすれば、上で述べたような操作を行っても、消えずにきちんと対応できるようになります。以下がその例です：

```
import com.rinearn.graph2d.RinearnGraph2D;
import com.rinearn.graph2d.RinearnGraph2DOptionItem;
import com.rinearn.graph2d.renderer.RinearnGraph2DRenderer;
import com.rinearn.graph2d.event.RinearnGraph2DPlottingEvent;
import com.rinearn.graph2d.event.RinearnGraph2DPlottingListener;
import java.awt.Color;

public class Sample6 implements RinearnGraph2DPlottingListener {
        RinearnGraph2D graph;
        RinearnGraph2DRenderer renderer;

        public static void main(String[] args) {
                new Sample6();
        }

        // グラフの起動と初期設定
        public Test() {
                // グラフを起動してレンダラー（描画エンジン）を取得
                this.graph = new RinearnGraph2D();
                this.renderer = graph.getRenderer();
```

```
                // 再描画が必要になったらイベントで受け取れるようにリスナー登録
                this.graph.addPlottingListener(this);

                // 描画処理を実行してスクリーンを 3DCG レンダリング
                this.draw();
                renderer.render();
        }

    // 描画エンジンによる描画処理
        public void draw() {

                // グラフ領域内の(0.3, 0.5)の位置に、半径 8 ピクセルで赤色の点を描画
                renderer.drawPoint(0.3, 0.5, 8.0, Color.RED);

                // グラフ領域内の(-0.2,-0.3)と(0.8,0.5)を結ぶ、
                // 太さ 3 ピクセルで青色の線を描画
                renderer.drawLine(-0.2, -0.3, 0.8, 0.5, 3.0, Color.BLUE);
        }

        // 再描画が必要になった際に呼ばれるイベント処理
        @Override
        public void plottingRequested(RinearnGraph2DPlottingEvent e) {
                // 描画処理を再実行
                this.draw();
        }

        // 以下のイベント処理はここでは何もしない
        @Override
        public void plottingCanceled(RinearnGraph2DPlottingEvent e) {
        }
        @Override
        public void plottingFinished(RinearnGraph2DPlottingEvent e) {
        }
}
```

このコードを実行した結果、描画される内容は、先ほどの Sample5 と全く同様です。しかし、マウス操作やメニュー操作によってグラフ範囲を移動・拡大・縮小したり、ファイルを開いて別のデータをプロットしたりしても、描いた点や線が消えてしまう事は無く、ちゃんと常に正しい位置に表示されます。

ワンポイント！ **plottingRequested 等の中で RinearnGraph2D のメソッドは呼べない!**

ここで一つ注意が必要です。RinearnGraph2DPlottingListener インターフェースを実装する際、plottingRequested メソッドなどの各イベント処理メソッド内から、描画エンジンではなくリニアングラフ 2D 本体側である RinearnGraph2D クラスのメソッドを呼ばないようにしてください（描画エンジンである RinearnGraph2DRenderer クラスのメソッドは呼んでも OK です）。

というのも、例えば plottingRequested メソッド内から、RinearnGraph2D クラスの setXRange メソッドでグラフ範囲を変更すると、それによって再描画が必要になるため plottingRequested が呼ばれ、またその中で setXRange メソッドを呼んで … と無限ループに陥ってしまいます。

他にも、RinearnGraph2D クラスの多くのメソッドは、処理が中途半端なタイミングで行われてしまうのを避けるために、描画エンジンの操作中は一旦待機して、描画完了してから処理を行うようになっています。そして、plottingRequested などの実行中は描画エンジンが操作中であると見なされるため、その中で RinearnGraph2D クラスのメソッドを呼ぶと、描画完了をいつまでも待機してしまい、処理が進まなくなってしまいます。

RinearnGraph2DPlottingListener インターフェースを実装した結果、グラフが動かなくなってしまったら、上の点を踏まえて見直してみてください。

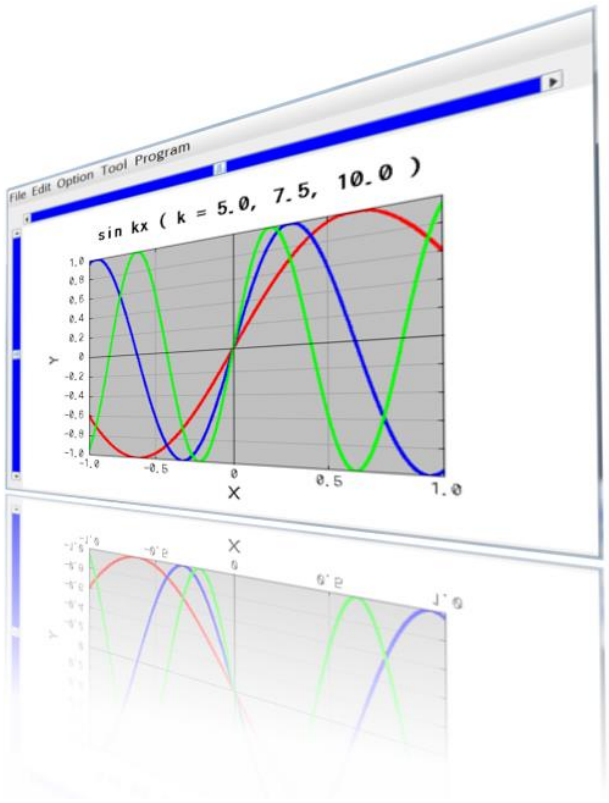
10.商標

[1] Oracle と Java は、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。

[2] Windows は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。

[3] Linux は、Linus Torvalds 氏の米国およびその他の国における商標または登録商標です。

その他、文中に使用されている商標は、その商標を保持する各社の各国における商標または登録商標です。



RINEARN Graph 2D　5.6　取扱説明書 第 6 版

著者　　松井文宏

この説明書に記載されている内容は、以下の Web サイトでも閲覧することができます。

https://www.rinearn.com/ja-jp/graph2d/

RINEARN
www.rinearn.com