

## RINEARN Graph 3D 5.6

取扱説明書 初版

## ■目次

1. はじめに	2
2. 起動方法	4
3. 画面の操作方法	9
4. 各メニュー機能解説	11
4-1. File メニュー	11
4-2. Edit メニュー	12
4-3. Option メニュー	16
4-4. Tool メニュー	20
5. 座標値ファイル書式	21
5-1. CSV ファイルと TSV ファイル	21
5-2. マトリックス書式	22
5-3. 3 カラム書式	24
5-4. 3 カラム書式で面を表現するには	24
6. 数式文法	29
6-1. 基礎文法	29
6-2. X 軸変数と Y 軸変数	30
6-3. 関数コマンド一覧	31
7. VCSSL での制御・自動処理	33
7-1. VCSSL と Graph3D API について	33
7-2. プログラムの作成と実行	33
7-3. プログラム例	34
8. Java 言語での制御・自動処理と自由な 3D 描画	38
8-1. 開発の準備とコンパイル・実行	38
8-2. ファイルのプロット	39
8-3. 配列データのプロット	41
8-4. 画像の出力	42
8-5. 範囲やオプションなどの詳細設定	43
8-6. アニメーションプロット	44
8-7. 描画エンジンの直接操作による自由な 3D 描画	48
9. 商標	54

# 1. はじめに

## リニアングラフ 3D とは

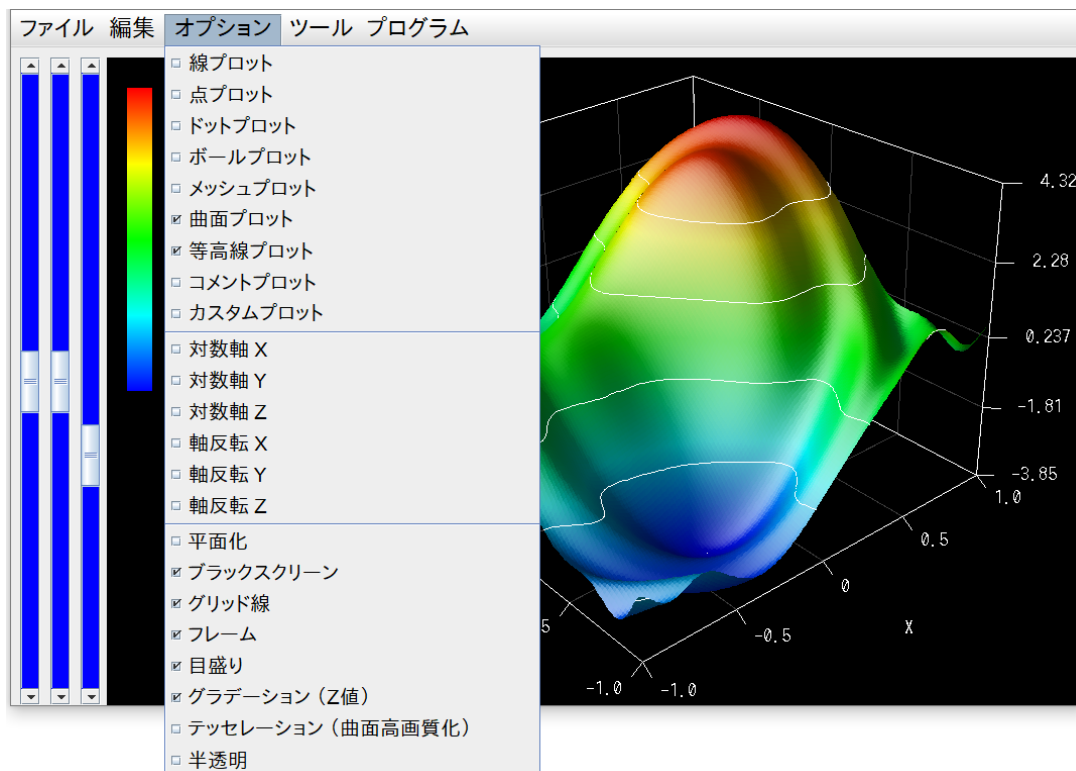
リニアングラフ 3D (RINEARN Graph 3D) は、表計算ソフトや数値計算プログラムなどで作成された座標値ファイルから、立体的な 3 次元グラフを描画できるソフトウェアです。様々な種類の PC 用オペレーティングシステム上において、インストール不要で動作し、USB メモリーなどに入れて持ち運んでの利用も可能です。また、商用・非商用を問わず、どなたでも無料で利用できます。

リニアングラフ 3D 公式 Web サイト (WEB 版ユーザーガイドも公開)

<https://www.rinearn.com/graph3d/>

リニアングラフ 3D は、内蔵の 3D 描画エンジンも含めて全て Java 言語で開発されているため、PC のハードウェア構成などに依存せず、どこでも同様の動作と描画結果が見込めます。3D 描画エンジンは専用開発ですが、グラフ用のものであれば高品質な光線反射・陰影処理付きのものなので、立体感に優れる 3D グラフを描画でき、複雑微妙な凹凸も一目で把握できるようサポートします。

画面デザインや操作系は、データ解析作業における日常的なツールとして手軽に扱えるよう、シンプルさと操作手順の少なさを重視したものとなっています。例えば、各種プロットオプションは、メニューバーから 2 クリックで選択切り替えが可能です。3 次元では不自然になりがちな、マウスによる視点の回転操作も、自然な操作感になるよう工夫した処理を採用しています。



## ライセンスに関して

リニアグラフ 3D は、どなたでも無償にてご利用頂けます。リニアグラフ 3D 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、リニアグラフ 3D 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、そちらをご参照ください。リニアグラフ 3D を二次配布する際も、配布物の中にこのライセンス文書が含まれるようにしてください。

## Ver.5.6 の特徴

リニアグラフ 3D の現行バージョンである Ver.5.6 は、5.x 系における 6 度目のマイナーアップグレード版であり、基本的な部分は踏襲しつつ、メイン画面やメニュー画面および操作などにリファイン的な改修が実施されました。その中でも大きな点として、複数のグラフ設定を、画面右上のセレクトからすぐに切り替えられる「 クイック設定 」機能も実装されました。

もう一つの大きな変化点として、新たに Java 言語のコードによる制御や自動処理を行うための API がサポートされました。この API により、ファイルやデータを読み込んだりプロットオプションを設定したりといった基本操作の他に、配列データを直接転送してプロットさせたり、リアルタイムでアニメーションさせる事もできます。

また、この API によって、描画エンジンを直接操作し、自由に 3D 空間上に点や線、三角形や四角形などの図形を描画する事も可能になりました。これにより、リニアグラフ 3D を、従来のように普通のグラフソフトとしてだけでなく、プログラミングでの簡易 3D 描画環境のような用途にも活用できるようになりました。

### ワンポイント!

## 2 次元版の「リニアグラフ 2D」も !

リニアグラフ 3D は 3 次元のグラフ専用ですが、兄弟ソフトとして、2 次元グラフをプロットするための「 リニアグラフ 2D 」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。

リニアグラフ 2D 公式 Web サイト（WEB 版ユーザーガイドも公開）

<https://www.rinearn.com/graph2d/>

## 2. 起動・プロット方法

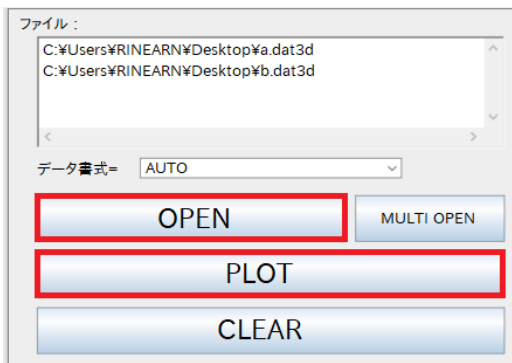
リニアグラフ3Dを起動してグラフをプロットするには、いくつかの方法が用意されています。ご利用の用途に合わせた方法で起動してください。

### 2-1. 基本的な起動・プロット方法(インストール不要)

#### ・JAR ファイルをダブルクリックで起動し、ファイルをプロット

まずは、最も単純な方法で起動し、ファイルをプロットしてみましょう。Microsoft® Windows® をご利用の場合は、ダウンロード・展開したフォルダ直下にある「RinearnGraph3D.jar (JAR ファイル)」をダブルクリックすると起動します。

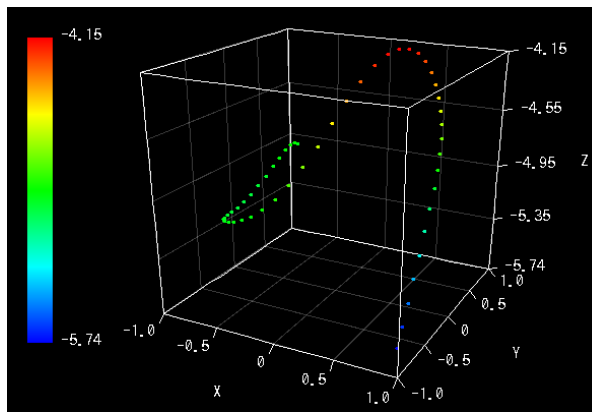
Linux® 等をご利用の場合は、上記 JAR ファイルを右クリックしたメニューから、Java の実行環境で開くよう指定すると起動します。権限などでエラーが出る場合は、JAR ファイルを右クリックし、「プロパティ」→「アクセス権」の項目などから、プログラムとして実行する権限を付加してください（コマンドラインで `sudo chmod +x RinearnGraph3D.jar` でも可能です）。



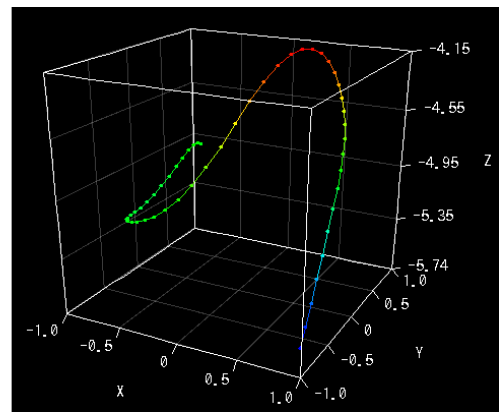
起動後、メニューバーから「ファイル/File」>「ファイルを開く/Open File」メニューを選択し、プロットしたいファイルを選択してください（「/」の後は英語モードでの表記です）。

「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。標準では左側の図の通り、各座標の位置に点がプロットされます。この点の色は、座標の Z 値に応じたグラデーションで色分けされています（後で説明する通り、無効化して単色にもできます）。メニューバーから「オプション/Option」>「線プロット/With Lines」のチェックを入れると、右側の図の通り、各点が線でつながれます。

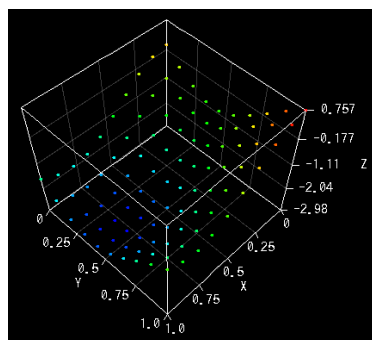


点プロット(起動時の標準オプション)

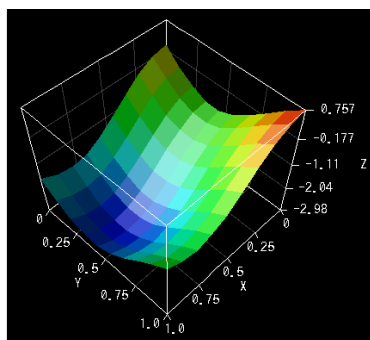


点+線プロット

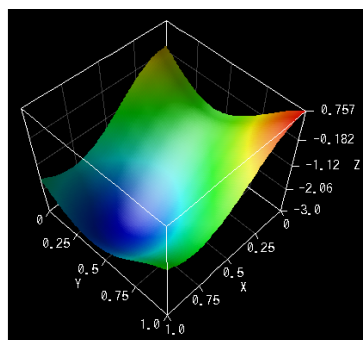
データがメッシュ形式（「5. 座標値データ書式」参照）に対応している場合は、同様に「オプション/Option」>「曲面プロット/With Membranes」を選択すると、下図中央のように各座標を四角形をつないだ曲面が描画されます。さらに「テッセレーション/Tessellation」を有効化すると、下図右側のように曲面の座標が密に補間され、なめらかになります。補間の結果、面が元のプロット範囲からはみ出て切れる場合は、「編集/Edit」>「範囲の設定/Set Range」で調整してください。



点プロット

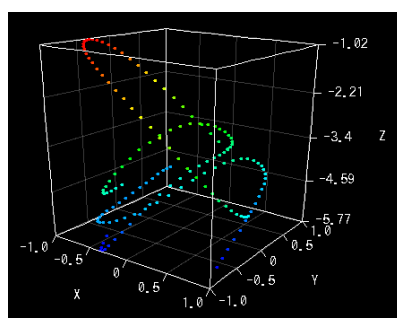


曲面プロット

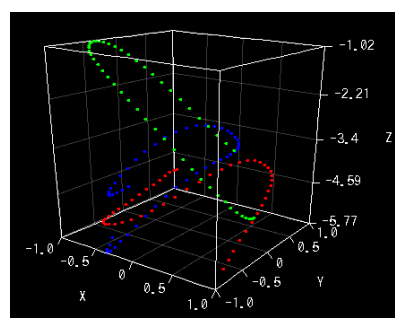


曲面+テッセレーション

ファイルを複数選択してプロットした場合は、下図左側の通り、それぞれのファイルのデータが重ねてプロットされます。それぞれのファイルのデータは、内部ではちゃんと異なる系列のものとして区別されているため、色分けする事もできます。メニューバーから「オプション/Option」>「グラデーション(Z 値)/Gradation Z」のチェックを外すと、下図右側のように色分けされます。



グラデーションON



グラデーションOFF

なお、1 つのファイル内に複数系列のデータが含まれている場合（「5. 座標値データ書式」参照）、同様にグラデーションを無効化すると、系列ごとに色分けされます。

### ワンポイント!

#### メモリーが不足する場合は…

ここで用いた起動方法では、メモリー使用量の上限が Java 実行環境のデフォルト値に限定されるため、ファイルによってはメモリーが不足する場合があります。メモリーが不足すると、「メモリーエラー」や「Memory Error」などとメッセージが表示され、ソフトウェアが終了してしまいます。

その場合は、RinearnGraph3D.jar( JAR ファイル )の代わりに RinearnGraph3D\_5.\*.bat ( バッチファイル )をダブルクリックして起動するか、後述するようにコマンド入力端末から多めのメモリー割り当て量を指定して起動してください。

### ・JAR ファイルをうまく起動できない場合

JAR ファイルをダブルクリックなどでうまく起動できない場合は、コマンド入出力端末でダウンロード・解凍したフォルダ(ディレクトリ)に移動し、以下の通りコマンドラインで起動する事もできます。

```
java -jar RinearnGraph3D.jar
```

なお、上の例では、デフォルトのメモリー容量が割り当てられますが、それでは不足する場合もあります(メモリーエラーなどと表示されます)。その場合は以下のように `-Xmx...` オプションを付け、任意のメモリー容量を割り当てる事もできます。以下の例は 256MB を割り当てて起動する例です。1GB を割り当てるには `-Xmx1g` とします。

```
java -Xmx256m -jar RinearnGraph3D.jar
```

ただし、毎回リニアグラフ 3D のフォルダまで移動したり、長いコマンドを打つのは面倒です。コマンドラインで常用する場合には、後に述べるように、固定場所に置いて「bin」フォルダのパスを環境変数 Path に登録すると便利です。そうすると、どこからでも短いコマンドで起動可能になります。

### ・それでも起動できない場合、Java 実行環境(JRE)のバージョン確認や導入を

なお、この方法でもうまく起動できない場合は、ご使用の PC に Java® の実行環境(JRE)が入っていないか、入っていても古い可能性があります。Windows をご使用の場合、以下の Web ページで最新の Java 実行環境を入手・導入できます: <https://java.com/ja/>  
64bit 版 OS に 32bit 版が入ってしまう場合は、下記ページから 64bit 版を選択してみてください:  
<https://java.com/ja/download/manual.jsp>

Linux 等をご使用の場合、上記でも導入可能ですが、apt が使用できる場合はコマンドラインで

```
apt search jre
```

 (または apt の代わりに apt-cache)

で入手可能なものの一覧を確認した上で、選択して導入できます。導入例は以下の通りです:

```
sudo apt install default-jre
```

 (または apt の代わりに apt-get)  

```
sudo apt install openjdk-*-jre
```

 ( \* の箇所にはバージョンの数字が入ります)

※ 2つとも入れる必要はありません。片方のみで十分です。

導入するのは他のものでも構いませんが、リニアグラフ 3D 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)。

## 2-2. インストールして使用する (Windows をご使用の場合)

### ・座標値データファイルのダブルクリックで自動的に起動・プロット

リニアングラフ 3D を Windows の PC にインストールすると、特定の拡張子の座標値データファイルをダブルクリックするだけで、自動的にリニアングラフ 3D でプロットできるようになります。また、大量のメモリー容量を割り当てる事も可能になります。

インストールと言っても、どこか適当な場所にリニアングラフ 3D のフォルダを設置しておくだけでよく、非常に簡単です。デスクトップでも OK ですが、ずっと動かさない場所のほうが良いでしょう。設置したら、プロットしたい座標値データファイルを右クリックし、メニューから「**プログラムから開く**」を選択し、そこでリニアングラフ 3D のフォルダ内にある下記のファイルを選択してください。

・RinearnGraph3D\_5.\*.bat (バッチファイル)

選択したら、「常にこのアプリを使って～のファイルを開く」や「この種類のファイルを開くときは、選択したプログラムをいつも使う」の項目を有効にして、「OK」を押して完了します。以降、同じ拡張子のファイルをダブルクリックすると、自動的にリニアングラフ 3D でプロットされます。

### ・コマンド起動

コマンドプロンプトなどのコマンド入出力端末から、リニアングラフ 3D で座標値ファイルをプロットできるようにする事も可能です。それにはまず、手動でどこか適当な場所にリニアングラフ 3D のフォルダを設置し、そしてその中の「**bin**」フォルダのパスを、Windows の環境変数 Path に登録して下さい。登録手順については、「Windows 環境変数 path」などと Web 検索してください。

すると、ring3d コマンドが使用可能になるので、以下のように入力して起動できます：

```
ring3d a.dat3d
```

この例では、座標値データファイル「a.dat3d」をプロットしています。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：

```
ring3d a.dat3d b.dat3d
```



## 2-2. インストールして使用する (Linux 等をご使用の場合)

Linux 等をご使用の場合にも、リニアングラフ 3D をインストールし、コマンド入力端末からいつでも使用できるよう事が可能です。なお、ここでは bash 互換のシェルの利用を想定しています。それ以外のシェルをご利用の場合は、シェルスクリプトを適切な内容へ改変してご利用ください。

まずは、リニアングラフ 3D のディレクトリをどこか適当な場所へ設置してください。ここでは例として以下の場所に設置したとします。

```
/usr/local/bin/rinearn/rinearn_graph_3d_5_*/ (5_*_*の箇所はバージョン番号です)
```

続いて cd コマンドでこのディレクトリ内の bin ディレクトリまで移動し、以下のコマンドを実行してください。

```
chmod +x ring3d  
chmod +x rinearngraph3d (旧版と同じコマンド名 rinearngraph3d も使用したい場合)
```

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_3d_5_*/bin/  
(5_*_*の箇所はバージョン番号です)
```

\$PATH:以降の内容は、リニアングラフ 3D のディレクトリを配置した場所に合わせてください。

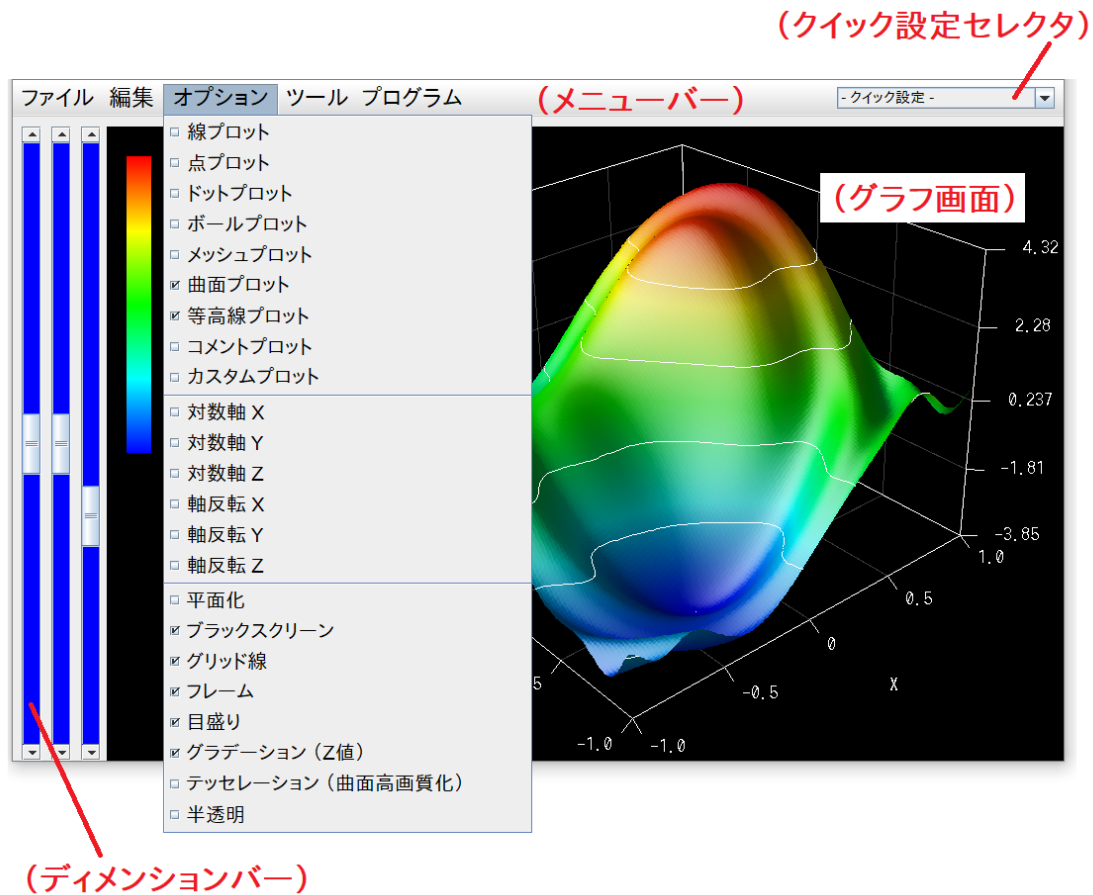
これで作業は完了です。再起動後、コマンド入力端末のどこからでも ring3d コマンドが使用可能となっています。以下のように使用できます：

```
ring3d a.dat3d
```

複数ファイルをプロットする例は以下の通りです：

```
ring3d a.dat3d b.dat3d
```

### 3. 画面の操作方法



※バージョンにより、各部のデザインが微妙に異なる場合があります。

#### ・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います。詳しくは次章で解説します。

#### ・ディメンションバー

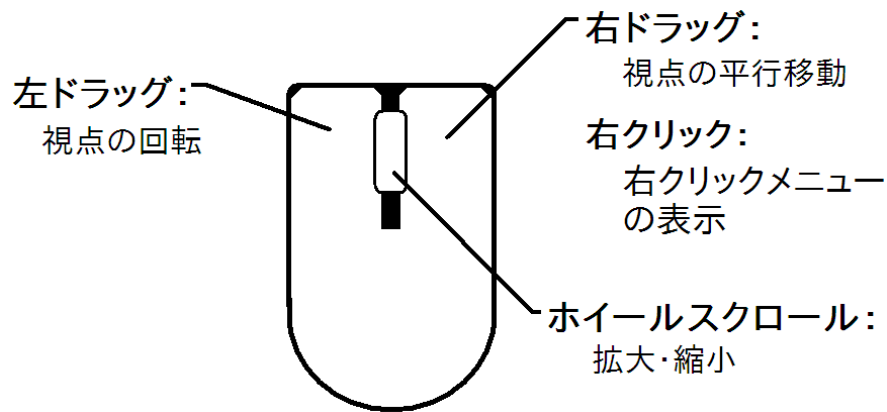
画面左に並ぶ3本のディメンションバーで、それぞれグラフの X、Y、Z 各方向の長さを設定できます。主に Z 方向への扁平率を調整するのに使用します。

#### ・クイック設定セレクト

画面右上のクイック設定セレクトで、設定をすぐに切り替える事ができます。設定の追加は、セレクトから「設定の追加」を選ぶと行えます。追加した設定は「 RinearnGraph3DQuickSetting 」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、視点変更や拡大・縮小等を行えます。



#### 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ことで、資料作成を格段にスムーズに行う事ができます。

### ・画像のコピー / 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。

### ・データの貼り付け / Paste Data

別のソフトウェアで選択コピーした数値データを、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。

- |   |
|---|
| <ul style="list-style-type: none"><li>1: ソフトウェアの場所 / Software Directory</li><li>2: データファイルの場所 / Data-File Directory</li><li>3: ホームディレクトリ / Home Directory</li><li>4: クイック設定 / Quick Setting</li></ul> |
|---|

1 と 3 は、リニアングラフ 3D 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トに設定を追加します。

## 2. 編集メニュー / Edit Menu

### ・Clear / クリア

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、Z それぞれに最小値と最大値を入力してください。

### ・ラベルの設定 / Set Label

X、Y、Z 軸の名称を指定します。前後に空白を入れる事で、表示位置を左右に調整できます。

### ・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、ここでの設定に基づいて色分けされます。ただし「Option」メニューで With Rainbow オプションを有効している場合は、ここでの設定と関係無く虹色に彩色されます。

### ・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。

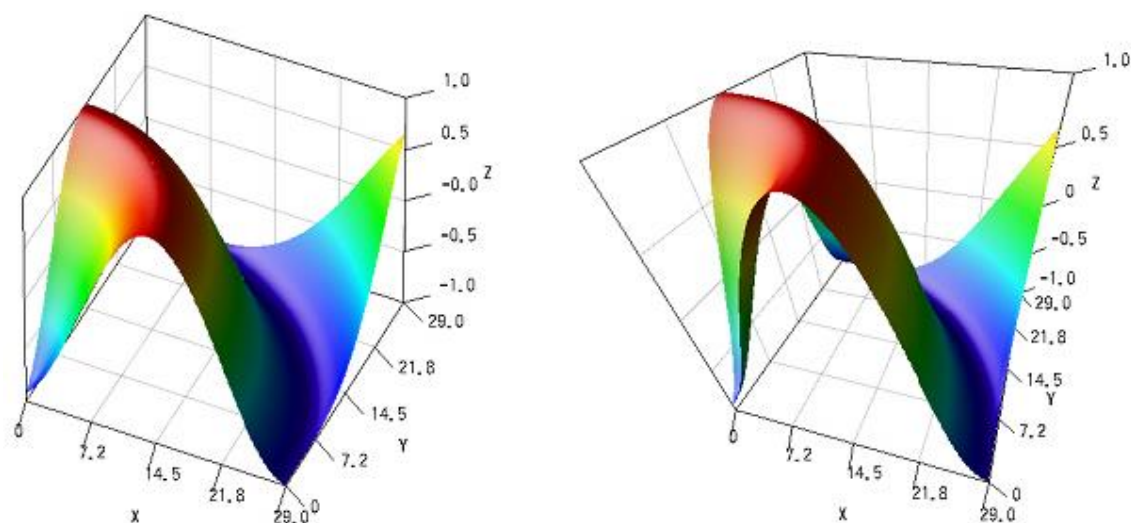
なお、ソフトウェアとの相性により使用不可能なフォントが存在するため、設定したものと別のフォントが使用される事があります。リニアングラフで初めて使用するフォントがある場合、画像を公開する前に、設定したフォントが正しく使用されているかを必ずご確認ください。

### ・カメラの設定 / Set Camera

カメラアングルや、カメラ距離 (Camera Distance)、表示倍率 (Magnification) を指定します。

カメラアングルはディスプレイ上でのマウスドラッグでも制御できますが、カメラアングルを設定として保存したい場合には、このメニューを通して設定を行ってください。

カメラ距離と表示倍率は、主に遠近感の調整に利用します。遠近感とは、「近くの箇所ほど大きく見え、遠くの箇所ほど小さく見える効果 (消失効果)」の事を指します。リニアングラフ 3D では現実のカメラと同様、遠い距離に離して拡大表示すると、遠近感の無いグラフに仕上がります。逆に、近い距離に接近して縮小表示すると、遠近感の強調されたグラフに仕上がります。前者は数学的なデータの表示に、後者は地形図などに向いているでしょう。



▲ 左:遠方から拡大表示(遠近感が殆ど無い)、 右:接近して縮小表示(遠近感が強い)

#### ・Set Scale / 目盛りの設定

目盛りのデザインや書式などを指定します。まず、「Frame / フレーム」項目では、グラフの輪郭タイプを指定可能です。

「BOX TYPE」: 標準の箱型フレームです。

「BACK-WALL TYPE」: 後ろ側の壁のみを描画し、前側の線は描画しないフレームです。

「FLOOR TYPE」: 床のみのフレームです。

「NONE」: フレームを描画しません。

また、「目盛りの区間数 / Number of Sectors」項目では、目盛りの区間の数を指定可能です。

さらに、「書式 / 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、「自動調整 / Auto」項目を選択解除し、「書式 / 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。

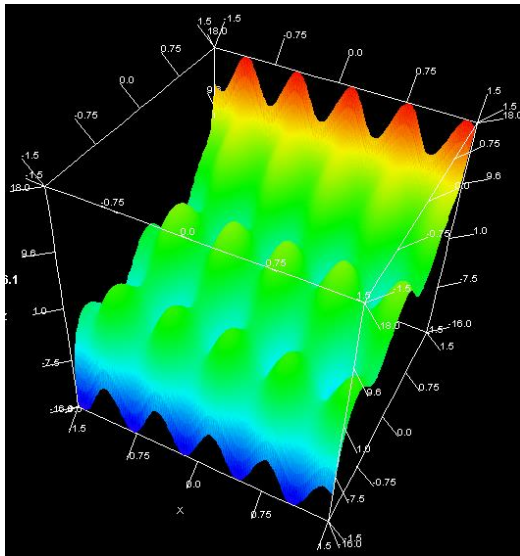
加えて、設定ウィンドウ右側に並ぶ「表示 / Visibility」の選択項目で、目盛りの表示・非表示を場所ごとに指定可能です。それには「自動調整 / Auto」項目を選択解除し、目盛りを表示したい部位のみを選択してください。

### ・光の設定 / Set Light

光線の反射パラメータ設定や、光源位置の設定を行います。

#### - 環境光反射率 / Ambient -

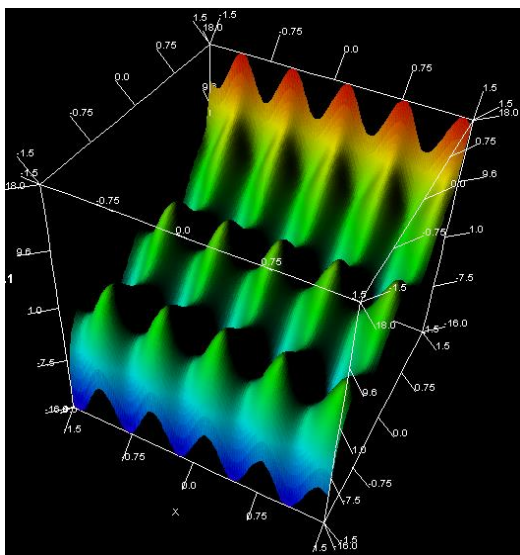
全体を一様に照らす環境光反射の強度を調整します。



▲Ambient のみを 100%にした場合

#### - 方向性反射率 / Directional -

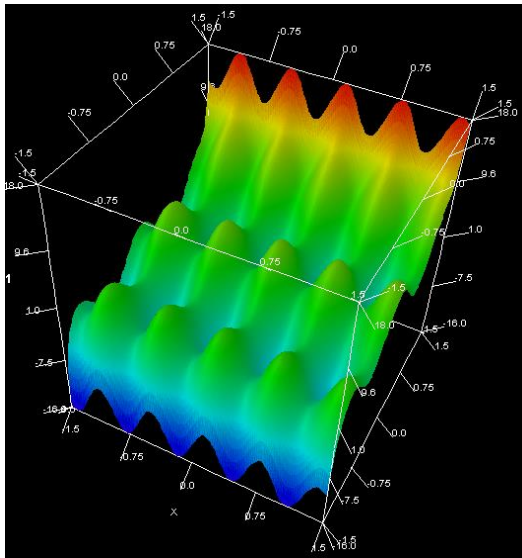
光源に垂直であるほど明るく照らす指向性反射の強度を調整します。



▲Directional のみを 100%にした場合

- 回折性反射率 / Diffractive -

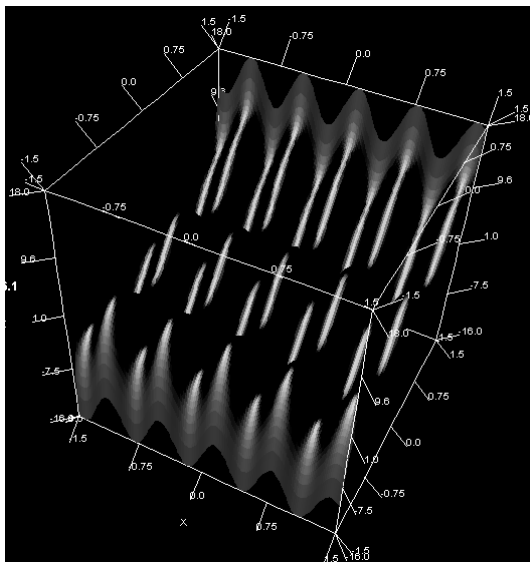
Diffractive で完全に陰となる所も多少回り込んで照らす、回折性反射の強度を調整します。



▲Diffractive のみを 100%にした場合

- 全反射強度 / Shininess, 全反射臨界角 / Shiny Angle -

光沢の強度と、光沢が表れる範囲の広さを調整します。光沢は、光源と視点との間に反射の法則が満たされる点を中心として、ここで指定した広さの範囲に表れます。



▲Shininess のみを 100%にした場合

- 光源の角度 / Light Angle -

グラフを照らしている光源の角度を指定します。



### 4-3. オプション メニュー / Option Menu

※Option メニューは縦に長いため、PC 画面の高さが不足（目安として 720 ピクセル未満）している場合、「Option2」メニューが出現し、内容が分割されて表示される場合があります。その場合、メニュー項目の配置が単に 2 つに分けられるだけであり、各項目の内容は変わりません。

#### ・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。

#### ・メッシュプロット / With Meshes

座標点をメッシュ（網線）で繋ぎ、面を表現するオプションです。 ※座標値ファイルはマトリックス書式もしくは 5 章 5-4 に記載の 3 カラム書式で記載する必要があります。

#### ・曲面プロット / With Membranes

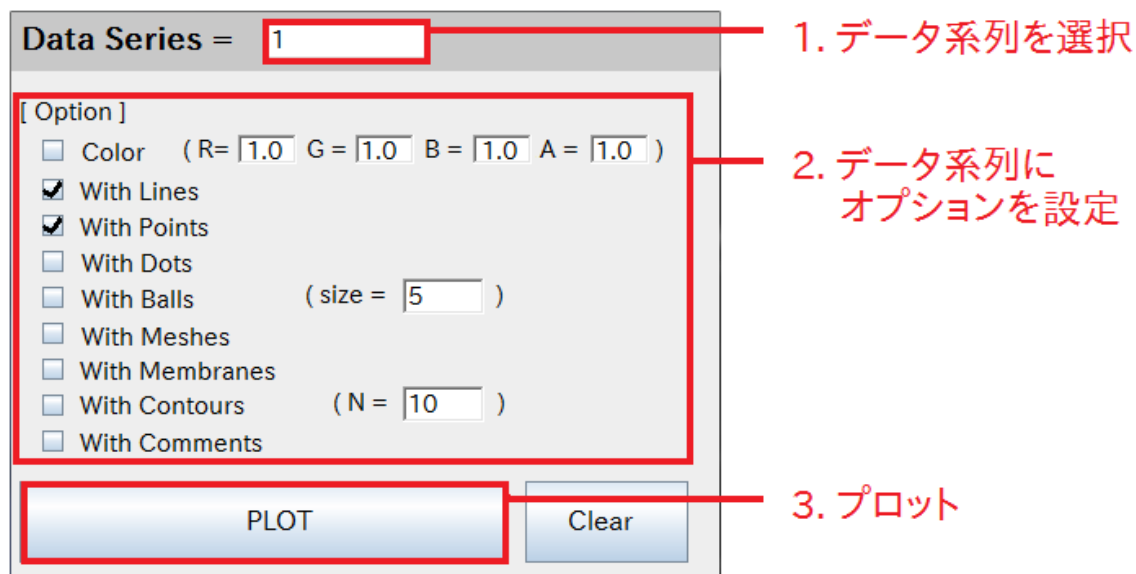
座標点を微小な平面で繋ぎ、曲面を表現するオプションです。光学計算により陰影や光沢が付加されます。「Edit / 編集」メニューの「Set Light / 光の設定」項目で光学計算の詳細な設定を行えます。 ※座標値ファイルはマトリックス書式もしくは 5 章 5-4 に記載の 3 カラム書式で記載する必要があります。

#### ・等高線プロット / With Contour

等高線を表現するオプションです。 ※座標値ファイルはマトリックス書式もしくは 5 章 5-4 に記載の 3 カラム書式で記載する必要があります。

### ・カスタムプロット / With Custom Option

ここまで述べたオプションや描画色を、データ系列ごとに独立に設定できる、上級者向けのオプションです。このオプションでは、下図のようなウィンドウが表示されます。

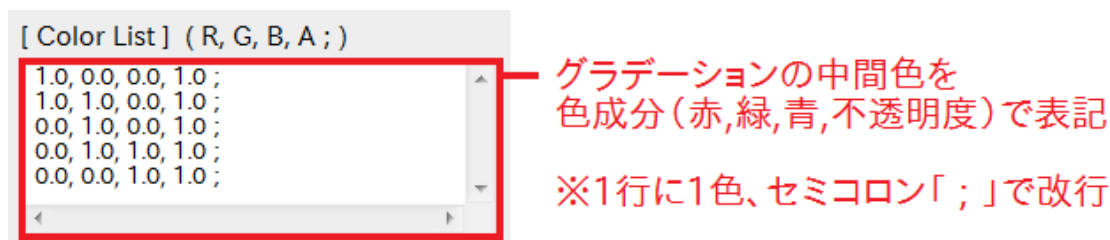


ここでデータ系列の番号と、その系列に使用するオプションを選択して、画面下部のプロットボタンを押すとグラフに描画されます。別のデータ系列を追加するには、データ系列の番号を変えて、上の操作を繰り返します。

データ系列の番号は、1つのファイルに1つの系列が記載されている場合、最初に読み込んだファイルが1番、次のファイルが2番、...といったように割り当てられます。ただし、1つのファイルに複数系列を記載する事も可能です。詳しくは5章「座標値ファイル書式」をご参照ください。

### ・グラデーション / Gradation

Z 値によってグラデーション彩色するオプションです。グラデーションの色は、標準では虹色になっていますが、任意の色を指定する事も可能です。グラデーションの色設定は、このオプションを有効にすると表示されるウィンドウの下部にある、Color List(カラーリスト)項目で行います。



カラーリスト項目では、グラデーションの中間色を、1行に1色の形で指定します。行の終わりにはセミコロン記号「;」を記入する必要があります。

色は、赤(R)、緑(G)、青(B)、不透明度(A)の色成分を、半角コンマ記号「,」区切りで指定します。各色成分は0.0~1.0の範囲で、加法混色(光の色の合成ルール)によって合成されます。

#### ・ブラックスクリーン / Black Screen

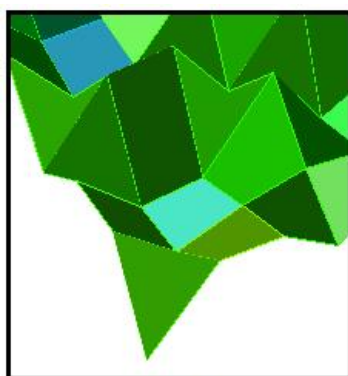
背景を黒色にするオプションです。OFF にすると白い背景になります。黒色の背景は液晶画面での作業に適しており、白色の背景は印刷に適しています。

#### ・半透明 / See-Through

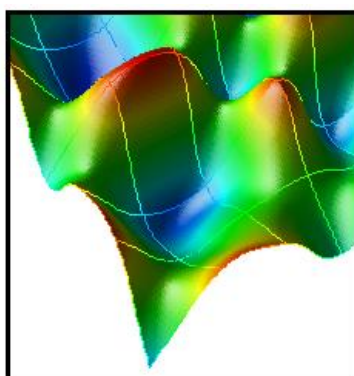
半透明化するオプションです。処理が重いので、画像を保存する直前などにご使用ください。

#### ・テッセレーション(曲面高画質化) / Tessellation

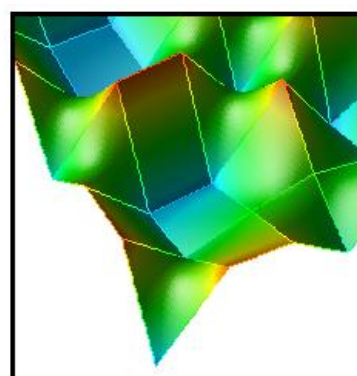
各座標点の位置を保ったまま、局面の隙間を補完するオプションです。複数のモードがあります。



BASE



PARABOLIC



GRAVITIC

「GRAVITIC」モード： 面要素を重心点で 4 つに分割し、これを繰り返す事によって曲面を補完します。この補完によって得られる曲面は、各座標点の位置を忠実にトレースし、尖り具合を元のままに保ちます。本質的に凹凸が激しいはずであるデータに向いています。

「PARABOLIC」モード： 面要素の辺を、隣の面要素と滑らかに繋がるように 2 次曲線で補完し、得られる 4 辺で囲まれた領域を滑らかに補完する事によって滑らかな曲面を生成します。この補完によって得られる曲面は、各座標点の位置を忠実にトレースしながらも、全域で滑らかなものに仕上がります。尖った箇所も丸みを帯びるので、本来滑らかであるべきデータに向いています。

※このモードは各座標点を滑らかな曲面で繋いで補完する都合上、元のプロット範囲から曲面の一部がはみ出る場合があります。適用後にグララの一部が欠けている場合は、Set Range で Z 方向のプロット範囲を広げてください。

これらのモードを、データに応じて使い分けてください。モードを選択した後は、補完密度を指定してください。「N(X)/Cell」項目に X 方向の補完数を、また「N(Y)/Cell」項目に Y 方向の補完数を指定してください。一つの座標点が  $N(X) \times N(Y)$  個の座標点に補完されます。

※補完数を上げるほど仕上がりが滑らかに仕上がりますが、処理時間を長く要します。また、処理後のメモリー使用量や描画負荷も重くなります。例えば 5×5 で補完すると、メモリー使用量や描画負荷は 25 倍に増大しますのでご注意ください。

### ・軸反転 X,Y,Z / Reverse X,Y,Z

X,Y,Z 軸の方向を反転させるオプションです。なお、目盛りの表示位置を限定している場合、このオプションを使用すると目盛りの表示位置も反転します。

### ・対数軸 X,Y,Z / Log X,Y,Z

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。**

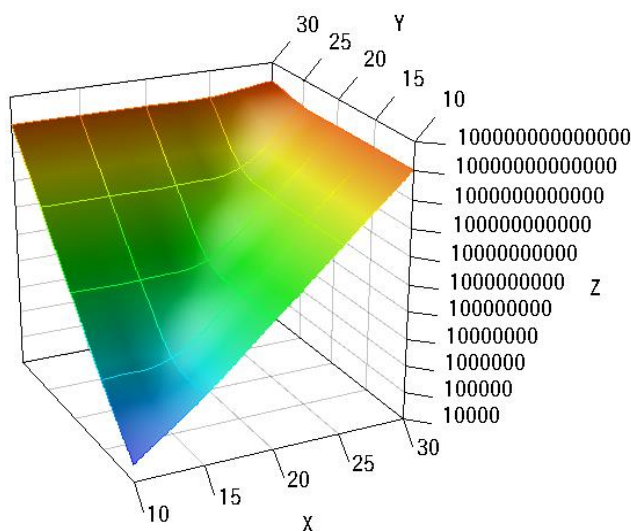
なおリニアグラフでは、対数軸の底が 10 であるか自然数(ネイピア数)であるかを気にする必要はありません。というのも、底の変換公式により、両者のグラフはそもそも比例関係にあり、目盛りの数字が異なるだけです。そしてリニアグラフでは、目盛りの数字にその位置の対数値ではなく、その位置の真の値を表示する仕様になっています(例えば 1000 の位置には、3 ではなくそのまま 1000 と表示されます)。従って、対数軸の底によらず全く同じグラフとなるため、底を気にする必要が無いのです。

#### 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います。

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。



## 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「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。

「SERIES-INDEX」： 時刻ごとに系列を切り替えてアニメーションします。

「X-INDEX」： 曲面/メッシュプロットにおいて、X 値を動かしながらアニメーションします。

「Y-INDEX」： 曲面/メッシュプロットにおいて、Y 値を動かしながらアニメーションします。

「X-RANGE」： プロット範囲を X 方向へ平行移動しながらアニメーションします。

「Y- RANGE」： プロット範囲を Y 方向へ平行移動しながらアニメーションします。

「X-RANGE」または「Y-RANGE」の場合、「Start」項目にプロット範囲始点、「End」項目に終点、「Width」項目にプロット範囲幅、「Shift」項目に時刻あたりの移動量を指定してください。

続いて、「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。

### ・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。「 $Z(<x>,<y>)=$ 」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>、Y 軸変数を<Y>と記述する必要があります。

## 5. 座標値ファイル書式

この章及び次章では、リニアングラフ 3D 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 3D 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、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 という拡張子を付ける決まりがあります。

TSV 形式： データをタブ空白で区切って並べるファイル形式です。  
こちらは特に拡張子の決まりはありません。

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 はプログラミング言語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 3D では、CSV と TSV の両方に対応しています。

## 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 3D 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「3 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、縦方向(行)を X 値、横方向(列)を Y 値と見なし、行列のように Z 値を記載していく書式です。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。反面、X や Y が格子状に並んだ座標しか表現できないというデメリットもあります。

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、最も左上のセルを空白にしたまま、**最も左側の列に X 値を記述し、最も上側の行に Y 値を記述します**。そして、この X 列と Y 行を辺とする長方形領域に、各 X・Y に対応する Z 値を記述します。

図: 表計算ソフトウェアの記述例

表計算ソフトウェア				
	0.0	0.1	0.2	0.3
0.0	1.111	1.233	1.453	1.882
1.0	1.431	1.432	1.551	1.714
2.0	1.881	1.883	1.772	1.701
3.0	1.532	1.228	1.113	0.988

上図において、青い列が X 値、赤い行が Y 値、緑の領域が Z 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 3D で読み込む事ができ、面プロットを含む全てのプロットオプションが使用可能です。

なお、必要が無ければ X 列と Y 行の記述は省略する事が可能です。その場合は Z 値の領域(上図において緑の領域)を、最も左上のセルから詰めて記述してください。その場合、X 値と Y 値にはそれぞれ列番号と行番号が割り振られます。

### ・複数系列の表現

リニアグラフ 3D では、最も左上のセルに「 LEVEL\_INDEX 」と記載すると、複数系列が定義されたマトリックス書式として読み込まれます。このセルの左から順に、1行目に系列番号を記載します。2 行目以降は、通常のマトリックス書式と同様です。ただし1行目に記載した系列番号により、座標値の所属する系列が振り分けられます。系列はいくつでも使用可能です。

※： なお、面プロットを使用するには、各系列において X 及び Y が昇順もしくは降順に並んでいる必要があります。

図： 複数系列におけるマトリックス書式の記述例(表計算ソフト使用)

LEVEL_INDEX	1	1	1	2	2	2
	Y	Y	Y	Y	Y	Y
X	Z	Z	Z	Z	Z	Z
X	Z	Z	Z	Z	Z	Z
X	Z	Z	Z	Z	Z	Z
X	Z	Z	Z	Z	Z	Z
X	Z	Z	Z	Z	Z	Z

系列1

系列2

上図において、一行目が系列番号と見なされます。系列が異なる座標値は別のグラフとして表現されます。系列の異なるデータは、グラフ化した際に別の色で彩色されます(※With Rainbow オプション OFF 時)。



### 5-3. 3 カラム書式

#### ・3 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、3 カラム書式がよく使用されます。3 カラム書式はファイル各行に左から X、Y、Z 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、及び X や Y が格子状に並ばない自由な座標値を表現できる事がメリットとして挙げられます。このメリットから、表計算ソフトでのデータ作成においても使用される事があります。

3 カラム書式の例 (タブ区切り) :

X1	Y1	Z1
X2	Y2	Z2
X3	Y3	Z3
...	...	...
X100	Y100	Z100

#### ・複数系列の表現

3 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。リニアングラフ 3D では、2 行連続した空白行をはさむと、そこが系列の区切りと見なされ、そこ以降は別の系列に属するデータと見なされます。系列の異なるデータは、グラフ化した際に別の色で彩色されます (※With Rainbow オプション OFF 時)。

### 5-4. 3 カラム書式で面を表現するには

3 カラム書式で面を表現するには、まず Y 値を固定し、X 方向に沿って端から端まで座標値を書き出してください。そこで空白改行を 1 行はさみ、Y 値を 1 つ進め、また X 方向に沿って端から端まで座標値を書き出してください。これを繰り返して面全体の座標値を書き出してください。

例として、次ページにある図のような、3×3 メッシュ上における面  $Z(X,Y)$  を表現する座標値データを次ページに示します。

図:3×3 の格子点(各格子点に Z 値が対応)

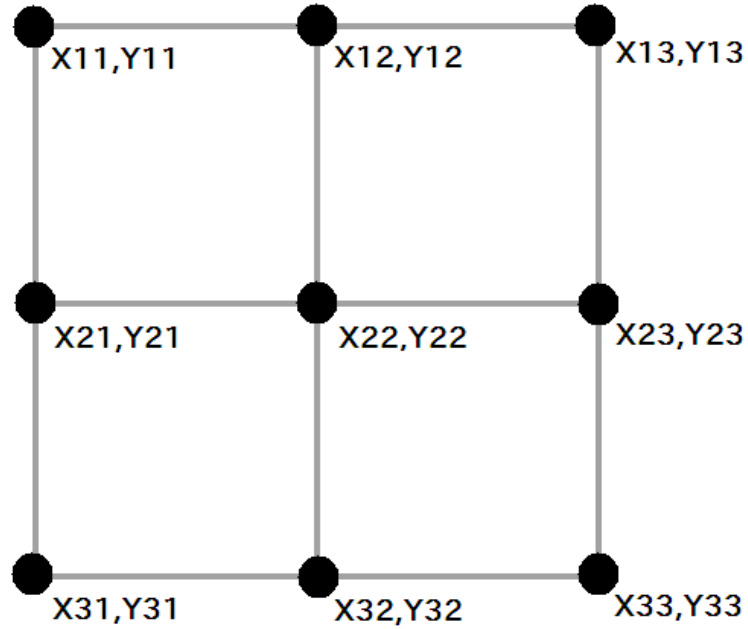


表: 座標値ファイルの内容

X11	Y11	Z11
X12	Y12	Z12
X13	Y13	Z13
( 空 白 行 )		
X21	Y21	Z21
X22	Y22	Z22
X23	Y23	Z23
( 空 白 行 )		
X31	Y31	Z31
X32	Y32	Z32
X33	Y33	Z33

このような座標値ファイルを出力する、C++言語、FORTRAN77 言語、及び VCSSL<sup>※</sup>での簡単なプログラムを次ページに例示します。

※VCSSL は、リニアングラフ 3D との連携機能を備えたプログラミング言語です。詳しくは 7 章をご参照ください。

### C++言語でのプログラム記述例 / $Z = \sin(X) + \cos(Y)$ の曲面グラフを作成

```
#include <fstream>
#include <math.h>

int main(){

    double x,y,z;
    int xN = 100; // X 方向の座標点数
    int yN = 100; // Y 方向の座標点数

    /* X・Y 方向の計算範囲 */
    double xMin = -5.0;
    double yMin = -5.0;
    double xMax = 5.0;
    double yMax = 5.0;

    std::ofstream ofs( "a.tsv" ); // 出力するファイル

    for( int i=0; i<xN; i++){
        for( int j=0; j<yN; j++){

            x = ( xMax - xMin ) * i / xN ;
            y = ( yMax - yMin ) * j / yN ;
            z = sin(x) + cos(y);

            /* X,Y,Z 値をタブ区切りで書き出して改行 */
            ofs << x << "\t" << y << "\t" << z << std::endl;

        }
        ofs << std::endl; // 空白改行
    }
    ofs.close();
    return 0;
}
```

**FORTRAN77 言語でのプログラム記述例 /  $Z = \sin(X) + \cos(Y)$  の曲面グラフを作成**

```
IMPLICIT NONE

REAL*8 X, Y, Z, X_MAX, Y_MAX, X_MIN, Y_MIN
INTEGER I, J, X_N, Y_N

X_N = 100      ! X 方向の座標点数
Y_N = 100      ! Y 方向の座標点数
X_MAX = 5.0D0  ! X 方向の計算範囲
X_MIN = -5.0D0 ! X 方向の計算範囲
Y_MAX = 5.0D0  ! Y 方向の計算範囲
Y_MIN = -5.0D0 ! Y 方向の計算範囲

OPEN(UNIT=14, FILE='a.tsv', STATUS='UNKNOWN')

DO I = 0, X_N
  DO J = 0, Y_N

    X = (X_MAX-X_MIN) * DBLE(I) / DBLE(X_N)
    Y = (Y_MAX-Y_MIN) * DBLE(J) / DBLE(Y_N)
    Z = SIN(X) + COS(Y)

    ! X,Y,Z を書き出して改行
    WRITE(14,*) X, Y, Z

  ENDDO

  ! 空白改行
  WRITE(14,*)

ENDDO

CLOSE( 14 )

STOP
END
```

VCSSL(3.0 以降)でのプログラム記述例 /  $Z = \sin(X) + \cos(Y)$  の曲面グラフを作成

```
import Math ;

double x ;
double y ;
double z ;

int xN = 100; // X 方向の座標点数
int yN = 100; // Y 方向の座標点数

/* X・Y 方向の計算範囲 */
double xMin = -5.0;
double yMin = -5.0;
double xMax = 5.0;
double yMax = 5.0;

int file = open ( "a.tsv", "wtsv" ); // 出力するファイル (wtsv は TSV 書き込みモード)

for( int i=0; i<xN; i++ ){
    for( int j=0; j<yN; j++ ){

        x = ( xMax - xMin ) * i / xN ;
        y = ( xMax - xMin ) * j / yN ;
        z = sin(x) + cos(y) ;

        /* X,Y,Z 値をタブ区切りで書き出して改行 */
        writeln( file, x, y, z ) ;

    }
    writeln( file, "" ) ; // 空白改行
}

close( file )
```

※ VCSSL2.1 以前では、writeln 関数の代わりに write 関数を使用します。VCSSL2.2 以降では、write 関数では改行がされなくなったため、改行付きの writeln 関数を使用します。

## 6. 数式文法

リニアグラフ 3D には、グラフを理論値等と比較するため、数式をプロットするためのツールが搭載されています。数式プロットツールは「Tool(ツール)」メニューの「Math(数式プロット)」を選択すると起動します。この数式ツールの入力ウィンドウに数式を入力し、「PLOT」を押すと数式のグラフが描画されます。

※ 最近のバージョンでは、「Program(プログラム)」メニューにおいて、VCSSL(次章)によって開発された、より新しい数式プロットツールを利用する事もできます。基本的な使い方は、ここで説明する数式プロットツールと同様ですが、より高機能かつ高速になっています。

ここでは、数式プロットツールで記述する数式の文法を解説します。

### 6-1. 基礎文法

#### ・四則演算の順序

数式プロットツールでは、四則演算の順序は正しくサポートされます。例えば以下の数式：

$$1+2*3$$

を入力すると、掛け算は足し算よりも先に計算しなければならないので、この式の値は 7 であると見なされます。左から順に計算されるわけではありませんのでご注意ください。

#### ・括弧(かっこ)付きの数式

数式プロットツールでは、括弧(かっこ)付きの数式を処理する事が可能です。数式中に括弧があると、まず括弧の中身が先に処理されます。括弧の中にさらに括弧があると、最も内側の括弧から先に処理されます。例えば、

$$((1+2)*3)/(4+5)$$

という数式を入力すると、まず最も内側の括弧(1+2)が処理され、3に置き換えられます。続いて(3\*3)=9と(4+5)=9が処理され、それぞれ置き換えられます。そして最後に9/9=1が処理され、この式の値は1であると見なされます。

### ・数学関数を用いた数式

数式プロットツールでは、様々な数学関数を用いた数式を処理する事ができます。使用可能な数学関数の一覧は次章をご参照ください。

数式中で数学関数を呼び出すには、関数コマンドを記述します。関数コマンドは**関数名**と**括弧**で成り立っており、括弧の中身が数学関数として処理されます。例えば正弦関数(サイン)の関数コマンドは

`sin()`

であり、平方根(ルート)の関数コマンドは

`sqrt()`

です。1.5 の正弦を求めるには

`sin(1.5)`

とし、2 の平方根を求めるには

`sqrt(2)`

とします。関数コマンドの括弧の中で、さらに関数を用いる事や、計算式を記述する事も可能です。例えば、以下のような数式も正しく処理されます。

`sin( ( sqrt(1/2) + sqrt(1/3) ) * 2 )`

## 6-2. X 軸変数と Y 軸変数

実際にグラフ化して意味のある数式は、X 軸変数と Y 軸変数を含んでいる必要があります。数式プロットツールでは、変数を「<」と「>」で挟んで記述してください。z = f(x,y)形式の2次元グラフを作成するには、数式中の X 軸変数の部分に < x >、Y 軸変数の部分に < y > と記述してください。

例: `sin(< x >) + cos(< y >)`

※ 「Program(プログラム)」メニューから起動できる、新しい数式プロット機能では、このように変数 x, y を < > で囲む必要はありません。普通に「`sin(x) + cos(y)`」などと記述してください。

### 6-3. 関数一覧

数式中に以下の関数を記述すると、括弧の中身が数学関数として処理されます。括弧の中にさらに関数を用いたり、式を記述する事も可能です。

#### **sqrt()**

二乗根、ルート

#### **exp()**

指数関数

#### **Log10()**

10 を底とした対数関数

※「Tool(ツール)」メニュー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、「Program(プログラム)」メニューの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 で使い分ける事が推奨されます。

#### **ln()**

自然数を底とした対数関数

#### **abs()**

絶対値

#### **!()**

階乗演算

※「Program(プログラム)」メニューの新しい数式プロットツールでは、fac() 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。

※「Program(プログラム)」メニューの新しい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. VCSSL での制御・自動処理

リニアングラフ 3D 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。

### 7-1. VCSSL と Graph3D API について

VCSSL は、データの解析や可視化、計算処理などを手軽に行うための簡易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://ja.vcssl.org/>

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(メモ用のものでも OK)でざっと書くだけで作成できます。また、リニアングラフ 3D は VCSSL プログラムの実行環境を内蔵しているため、「Program / プログラム」メニューから手軽にプログラムを選択して実行する事ができます。なお、上記 URL から単体の VCSSL の実行環境を入手する事もできます。

VCSSL からリニアングラフ 3D を制御するには、tool.Graph3D API を使用します。ここでは Graph3D API の一部の機能のみを使用します。全機能は下記 URL から参照できます。

tool.Graph3D API 詳細仕様

<https://ja.vcssl.org/lib/tool/Graph3D>

### 7-2. プログラムの作成と実行

VCSSL のプログラムは、一般のテキストエディタで普通に記述し、拡張子(≡名前の末尾)「.vcssl」を付けて保存してください。ソフトによっては、保存の際にファイルの種類などを選択する項目があるので、その場合は「すべてのファイル」などを選んでください。

作ったプログラムを実行するには、まずリニアングラフ 3D のメニューバーから「**Program / プログラム**」メニューをクリックしてください。するとプログラムを選択する画面が表示されるので、そこで自作プログラムを選ぶと、実行されます。

なお、よく使用するプログラムは、「**RinearnGraph3DProgram**」フォルダに入れておくと、上記メニューをクリックした際にすぐに表示され、手短かに実行できるようになります。

## 7-4. 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。

```
//リニアグラフ 3D を制御するライブラリをインポート
import tool.Graph3D;

//リニアグラフ 3D を起動( x 位置, y 位置, 幅, 高さ, タイトル )
int graph = newGraph3D( 0, 0, 700, 700, "Graph" );

//座標値データファイル「 test.tsv 」をプロットする
setGraph3DFile( 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を有効にするプログラムです。

```
import tool.Graph3D;

int graph = newGraph3D( 0, 0, 700, 700, "Graph" );
setGraph3DFile( graph, "test.tsv" );

//点プロットを無効にする
setGraph3DOption( graph, "WITH_POINTS", false );

//線プロットを有効にする
setGraph3DOption( 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.Graph3D;

int graph = newGraph3D( 0, 0, 700, 700, "Graph" );

setGraph3DFile( 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 )
exportGraph3D( 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.Graph3D;

int graph = newGraph3D( 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++){

    setGraph3DFile( graph, "./test/test"+i+".tsv" );
    exportGraph3D( 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、Z 値を  $\sin(Z \text{ 値})$  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.Graph3D;
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", "wtsv" );

// 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
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 Z 値を  $\sin(Z \text{ 値})$  に変換して書き出し、改行
        newData += value[0] + "      " + value[1] + "      " + sin( value[2] ) + EOL;
    }
}

//ファイルアクセスを閉じる
close( file );

int graph = newGraph3D( 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
setGraph3DData( 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グラフ 3D へ転送する方が、はるかに高速です。

```
import tool.Graph3D;
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", "wtsv" );

// 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
double z[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] = value[1];
    z[i] = sin( value[2] ); // Z 値は sin(Z 値)に変換
}

//ファイルアクセスを閉じる
close( file );

int graph = newGraph3D( 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
setGraph3DData( graph, x, y, z );
```

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。

## 8. Java 言語での制御・自動処理と自由な 3D 描画

リニアングラフ 3D 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。

### 8-1. 開発の準備とコンパイル・実行

Java 言語でリニアングラフ 3D 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 3D 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.graph3d.RinearnGraph3D;

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

        // グラフを起動
        RinearnGraph3D graph = new RinearnGraph3D();
    }
}
```

続いて、リニアングラフ 3D の配布パッケージ内にある「RinearnGraph3D.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 ".;RinearnGraph3D.jar" Sample0.java	(Windows の場合)
javac -classpath ".:RinearnGraph3D.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「 : 」で異なります。

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 ".;RinearnGraph3D.jar" Sample0	(Windows の場合)
java -classpath ".:RinearnGraph3D.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 3D 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 3D API 仕様書（Java 言語用）

<https://www.rinearn.com/graph3d/api/>

API ライブラリの中でも特に使用する RinearnGraph3D クラスの仕様書は、以下で参照できます：

<https://www.rinearn.com/graph3d/api/com/rinearn/graph3d/RinearnGraph3D>

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。

## 8-2. 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、座標値ファイルに記載された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.txt -

0.0	0.0	0.0
1.0	1.0	2.0
2.0	4.0	6.0
3.0	9.0	12.0
4.0	16.0	20.0
5.0	25.0	30.0
6.0	36.0	42.0
7.0	49.0	56.0
8.0	64.0	72.0
9.0	81.0	90.0
10.0	100.0	110.0

この通り、3 カラム書式（左から x y z）で、 $0 \leq x \leq 10$  の範囲で適当に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：



```

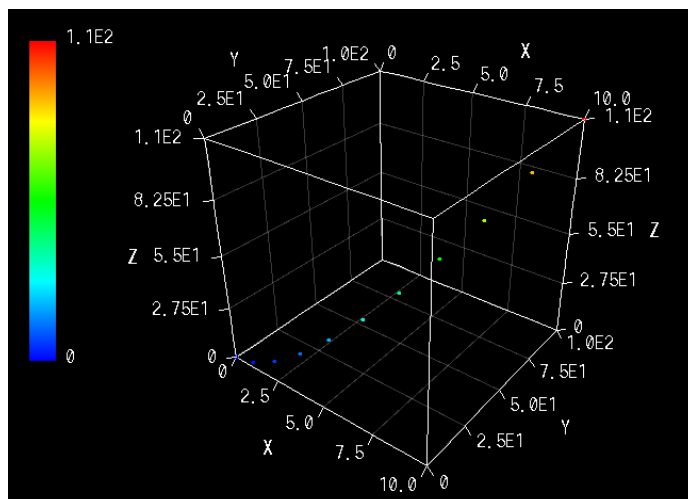
import com.rinearn.graph3d.RinearnGraph3D;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Sample1 {
    public static void main(String[] args) {
        // グラフを起動
        RinearnGraph3D graph = new RinearnGraph3D();
        try {
            // データファイルを読み込んでプロット
            graph.openDataFile(new File("SampleDataFile1.txt"));

            // 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 RinearnGraph3D 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。

### 8-3. 配列データのプロット

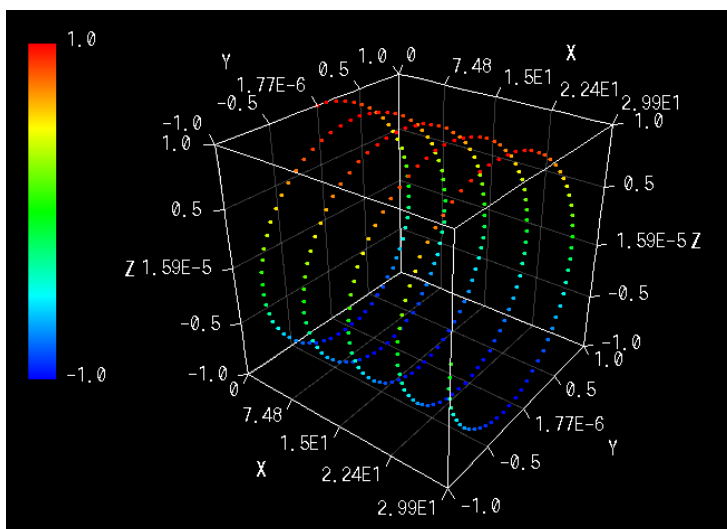
ファイルを介さず、配列に格納された座標値データを、直接渡してプロットする事もできます:

```
import com.rinearn.graph3d.RinearnGraph3D;

public class Sample2 {
    public static void main(String[] args) {
        // 座標値データの用意
        int n = 300;
        double[] x = new double[n];
        double[] y = new double[n];
        double[] z = new double[n];
        for(int i=0; i<n; i++) {
            x[i] = i*0.1;
            y[i] = Math.sin(x[i]);
            z[i] = Math.cos(x[i]);
        }

        // グラフを起動し、データを渡してプロット
        RinearnGraph3D graph = new RinearnGraph3D();
        graph.setData(x, y, z);
    }
}
```

#### ▼実行結果



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 ]` や `x[ 大系列インデックス ][ 小系列インデックス ][ 座標点インデックス ]` の形の多次元配列にします (`y`, `z` も同様)。線プロット時には、小系列の区切りで線が切れ、大系列の区切りで色が変わります。

なお、メッシュプロットや曲面プロットの場合は、`x[ メッシュ縦方向インデックス ][ メッシュ横方向インデックス ]` のように、メッシュの縦横 2 方向のインデックスを持たせてください (`y`, `z` も同様)。その際、どちらが縦か横かは気にしなくても構いません。具体例は 8-5 をご参照ください。

## 8-4. 画像の出力

グラフを画像ファイルに出力するには、`RinearnGraph3D` 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。

## 8-5. 範囲やオプションなどの詳細設定

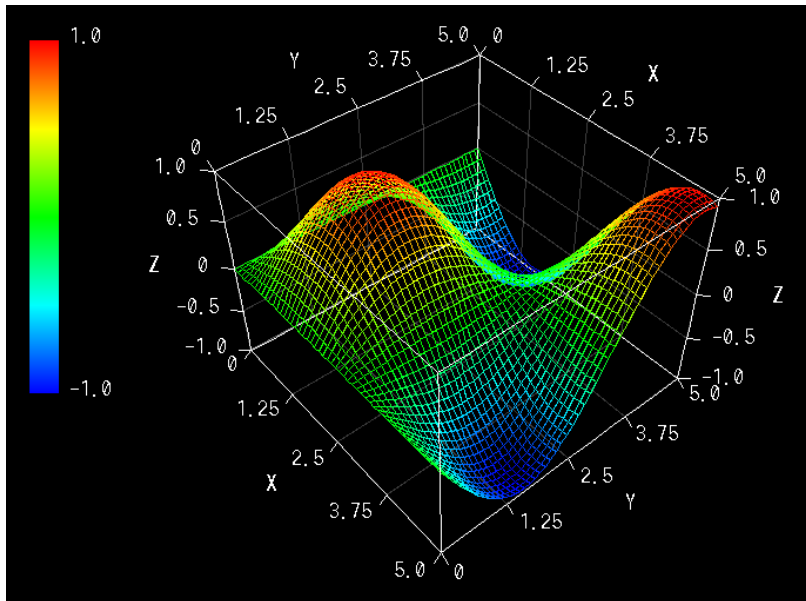
setX/Y/ZRange メソッドや setOptionSelected メソッドなどを用いて、プロット範囲やオプションなどの設定を行う事ができます。例として、8-3 の最後で触れたメッシュ状の配列データをプロットし、メッシュプロットオプションを有効にして、範囲も設定してみましょう：

```
import com.rinearn.graph3d.RinearnGraph3D;
import com.rinearn.graph3d.RinearnGraph3DOptionItem;

public class Sample3 {
    public static void main(String[] args) {
        // 座標値データの用意
        int n = 80; // メッシュの各方向の区間数
        double[][] x = new double[n+1][n+1]; //各方向の頂点数は区間数+1
        double[][] y = new double[n+1][n+1];
        double[][] z = new double[n+1][n+1];
        for(int i=0; i<=n; i++) {
            for(int j=0; j<=n; j++) {
                x[i][j] = i * 0.1;
                y[i][j] = j * 0.1;
                z[i][j] = Math.sin(x[i][j]) * Math.sin(y[i][j]);
            }
        }

        // グラフを起動してデータをプロット
        RinearnGraph3D graph = new RinearnGraph3D();
        graph.setData(x, y, z);
        // オプション設定（点プロット無効化してメッシュプロット有効化）
        graph.setOptionSelected(RinearnGraph3DOptionItem.POINT, false);
        graph.setOptionSelected(RinearnGraph3DOptionItem.MESH, true);
        // 範囲設定
        graph.setXRange(0.0, 5.0);
        graph.setYRange(0.0, 5.0);
        graph.setZRange(-1.0, 1.0);
    }
}
```

## ▼実行結果



上のコードでは、`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 `RinearnGraph3DOptionItem` 列挙子の要素で指定します。上では初期状態で有効な点プロットオプションを無効化し、メッシュプロットオプションを有効化しています。また、`setXRange` / `setYRange` / `setZRange` メソッドでそれぞれ X/Y/Z 軸方向のプロット範囲を設定しています。

設定系のメソッドは他にも存在します。詳細については、`RinearnGraph3D` 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 3D 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。

## 8-6. 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.graph3d.RinearnGraph3D;
import com.rinearn.graph3d.RinearnGraph3DOptionItem;
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
    RinearnGraph3D graph = null;    // グラフ
    boolean loopState = 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() {
        // グラフを起動してプロットオプションを設定（曲面プロット）
        this.graph = new RinearnGraph3D();
        this.graph.setOptionSelected(RinearnGraph3DOptionItem.POINT, false);
        this.graph.setOptionSelected(RinearnGraph3DOptionItem.MEMBRANE, true);
        // 描画範囲の設定と自動調整機能の無効化
        this.graph.setXRange(0.0, 5.0);
        this.graph.setYRange(0.0, 5.0);
        this.graph.setZRange(-1.0, 1.0);
        this.graph.setXAutoRangingEnabled(false);
        this.graph.setYAutoRangingEnabled(false);
        this.graph.setZAutoRangingEnabled(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 = 80; // メッシュの各方向の区間数
    double[][] x = new double[n+1][n+1]; //各方向の頂点数は区間数+1
    double[][] y = new double[n+1][n+1];
    double[][] z = new double[n+1][n+1];

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

        // 座標値データの更新
        for(int i=0; i<=n; i++) {
            for(int j=0; j<=n; j++) {
                x[i][j] = i * (5.0/n);
                y[i][j] = j * (5.0/n);
                z[i][j]
                    = Math.sin(x[i][j]-t) * Math.cos(y[i][j]+t)
                    * Math.sin(Math.cos(x[i][j]+y[i][j])-2*t-0.7);
            }
        }

        // 座標値データをグラフに転送(非同期)
        this.graph.setData(x, y, z);

        // 50 ミリ秒だけ停止(時間は適時調整)
        try {
            this.thread.sleep(50);
        } 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
}

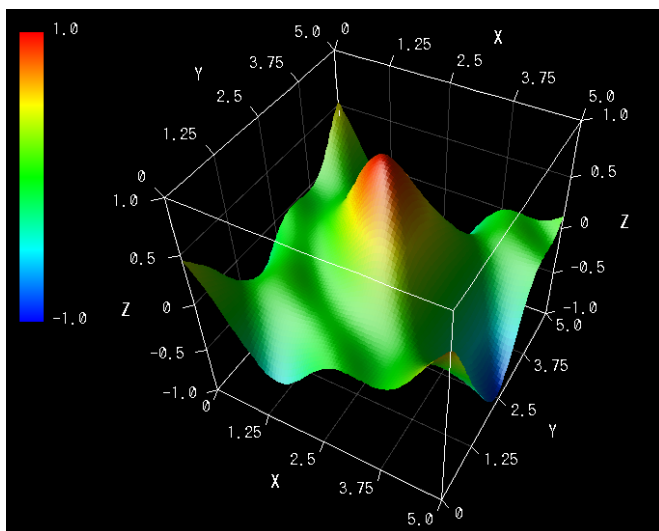
```

```
// ※ 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.setAutoExitingEnabled(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）
    loopState = 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（曲面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。）





## 8-7. 描画エンジンの直接操作による自由な 3D 描画

ここまで、座標値データから、点や線および面などを組み合わせて 3D のグラフを描く処理は、全てリニアングラフ 3D 側が自動で行っていました。これは手軽なので便利ですが、用途によっては、自分で自由に、3D 空間内に点や線および面などを描きたい場合もあるでしょう。

そのような場合には、リニアングラフ 3D の描画エンジンの API を使用します。これは RinearnGraph3DRenderer クラスの各メソッドとして提供されています。このクラスの仕様書は下記で参照できます：

### ・ RinearnGraph3DRenderer クラス API 仕様書

<https://www.rinearn.com/graph3d/api/com/rinearn/graph3d/renderer/RinearnGraph3DRenderer>

この API を用いると、リニアングラフ 3D を通常のグラフソフトとしてだけではなく、プログラミングでの簡易 3D 描画環境としても活用する事ができます。特に、表示ウィンドウやマウスでの視点操作機能などが最初からトータルで備わっているので、「とにかく即席で 3D 図形を描画したい」といった用途などに便利かもしれません。

### ワンポイント!

#### リニアングラフ 3D の描画エンジンの基本特性

リニアングラフ 3D では、全処理を Java 言語で実装した、ソフトウェアレンダリング式の 3D 描画エンジンを採用しています。これにより、動作環境に関する要求は特に無く、どこでも概ね同様の描画結果が得られます。その半面として、描画速度は数十万ポリゴン/秒程度と、そう速くありません。また、描画形式はシンプルな Z ソート形式のフラットシェーディングのみとなっています。遠近判定はピクセル単位ではなくポリゴン単位で行われるため、ポリゴン同士が交差・めり込んでいたり、遠近方向にかなり近接していたり、巨大なポリゴンがある箇所などでは、描画上の手前/奥の関係がラフになります。そのため、細かく遠近判定させたい箇所では、ポリゴンも細かく分割してください。

### ・簡単な例

さて、描画エンジンの処理は、もちろんリニアングラフ 3D 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 3D 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、とりあえずそのような事を一切考えずに、単に 3D 空間に図形を描きたいだけであれば、非常に単純にコードを書く事ができます。

まずは最も簡単な例として、3D 空間内の自由な位置に、点と線、三角形、および四角形を1個ずつ描いてみましょう:

```
import com.rinearn.graph3d.RinearnGraph3D;
import com.rinearn.graph3d.renderer.RinearnGraph3DRenderer;
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
        RinearnGraph3D graph = new RinearnGraph3D();
        RinearnGraph3DRenderer renderer = graph.getRenderer();

        // グラフ空間の範囲を設定
        graph.setXRange(0.0, 10.0);
        graph.setYRange(0.0, 10.0);
        graph.setZRange(0.0, 10.0);

        // グラフ空間内の(1,2,3)の位置に、半径 10 ピクセルで赤色の点を描画
        renderer.drawPoint(1.0,2.0,3.0, 10.0, Color.RED);

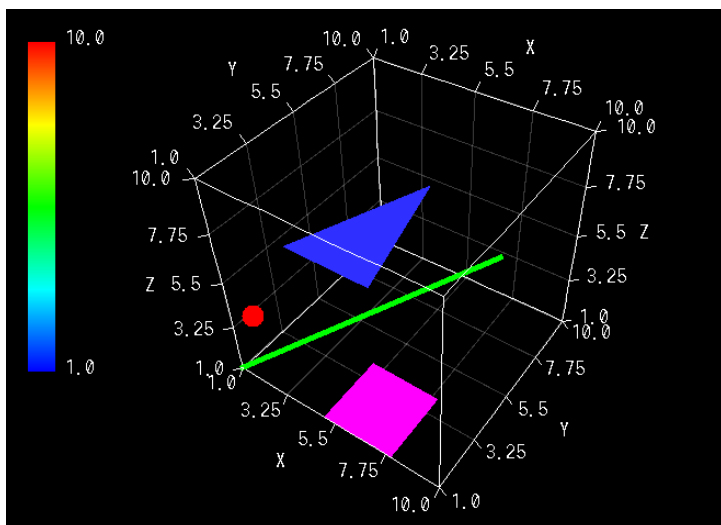
        // (1,1,1)と(3,5,8)の位置を結ぶ、太さ 5 ピクセルで緑色の線を描画
        renderer.drawLine(1.0,1.0,1.0, 10.0,5.0,8.0, 5.0, Color.GREEN);

        // (1,4,5)と(5,4,5)と(5,8,5)を結ぶ、青色の三角形を描画
        renderer.drawTriangle(
            1.0,4.0,5.0, 5.0,4.0,5.0, 5.0,8.0,5.0, Color.BLUE);

        // (5,1,1)と(8,1,1)と(8,4,1)と(5,4,1)を結ぶ、紫色の四角形を描画
        renderer.drawQuadrangle(
            5.0,1.0,1.0, 8.0,1.0,1.0, 8.0,4.0,1.0, 5.0,4.0,1.0, Color.MAGENTA);

        // スクリーンの再描画（3DCG 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、グラフ空間の範囲を設定した上で、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、この段階で初めて、3D 空間内に描いた立体が、平面のグラフ画面に射影されて(2 次元の絵として)描画されます。いわゆる 3DCG のレンダリング処理です。マウスで視点を操作した際などには自動で再レンダリングされますが、点や線の描画メソッドを呼ぶ度に毎回自動で再レンダリングされたりはしないため(処理コストが大きいからです)、このように最後に明示的に再レンダリングさせています。

`drawPoint` などの各描画メソッドの引数に渡す座標は、標準で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、標準では描画されません。このような細かい挙動を変更するには、各描画メソッドの引数で色を指定している部分で、代わりに `RinearnGraph3DDrawingParameter` クラスのインスタンスを渡してください。このクラスには、描画時のグラフ範囲に応じたスケーリングの有無や、はみ出した部分の除去の有無、および色設定や自動彩色の有無などを細かく設定できます。なお、最初に `RinearnGraph3DRenderer` クラスの `clear` メソッドを呼び出しておくと、グラフ範囲の目盛りや枠線を非表示にする事もできます。

### ・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。そのような操作は、グラフ全体の立体形状の造りなおしが必要になるため、最初にリニアングラフ 3D 側から描画内容をリセットする処理が呼ばれるからです。

といっても、別にグラフ範囲やプロットオプションの変更を行う必要が無く、そもそもユーザーに画面上でそのような操作をさせたくない場合であれば、`RinearnGraph3D` クラスの `setMenuVisible` メソッドの引数に `false` を指定して、メニューを非表示にする事もできます。

一方で、ちゃんとグラフ範囲の変更などに対応したい場合は、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、その度に描画しなおす事もできます。以下がその例です：

```
import com.rinearn.graph3d.RinearnGraph3D;
import com.rinearn.graph3d.renderer.RinearnGraph3DRenderer;
import com.rinearn.graph3d.event.RinearnGraph3DPlottingEvent;
import com.rinearn.graph3d.event.RinearnGraph3DPlottingListener;
import java.awt.Color;

public class Sample6 implements RinearnGraph3DPlottingListener {
    RinearnGraph3D graph;
    RinearnGraph3DRenderer renderer;

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

    // グラフの起動と初期設定
    public Sample6() {
        // 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 RinearnGraph3D();
        this.renderer = this.graph.getRenderer();

        // グラフ空間の範囲を設定
        this.graph.setXRange(0.0, 10.0);
        this.graph.setYRange(0.0, 10.0);
        this.graph.setZRange(0.0, 10.0);

        // 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);

        // 3D 形状の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();
    }
}
```

```

// 3D 形状の描画処理
public void draw() {
    // グラフ空間内の(1,2,3)の位置に、半径 10 ピクセルで赤色の点を描画
    this.renderer.drawPoint(1.0,2.0,3.0, 10.0, Color.RED);

    // (1,1,1)と(3,5,8)の位置を結ぶ、太さ 5 ピクセルで緑色の線を描画
    this.renderer.drawLine(1.0,1.0,1.0, 10.0,5.0,8.0, 5.0, Color.GREEN);

    // (1,4,5)と(5,4,5)と(5,8,5)を結ぶ、青色の三角形を描画
    this.renderer.drawTriangle(
        1.0,4.0,5.0, 5.0,4.0,5.0, 5.0,8.0,5.0, Color.BLUE);

    // (5,1,1)と(8,1,1)と(8,4,1)と(5,4,1)を結ぶ、紫色の四角形を描画
    this.renderer.drawQuadrangle(
        5.0,1.0,1.0, 8.0,1.0,1.0, 8.0,4.0,1.0, 5.0,4.0,1.0, Color.MAGENTA);
}

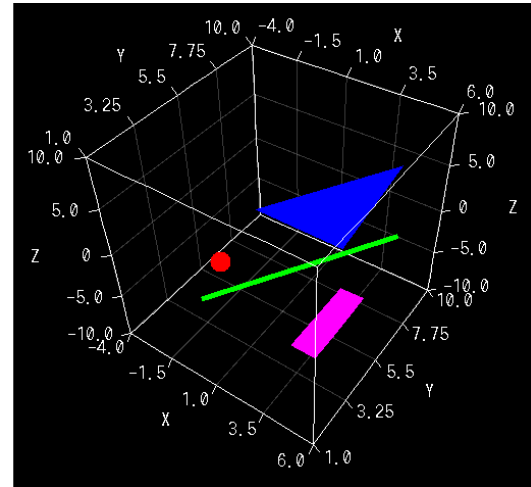
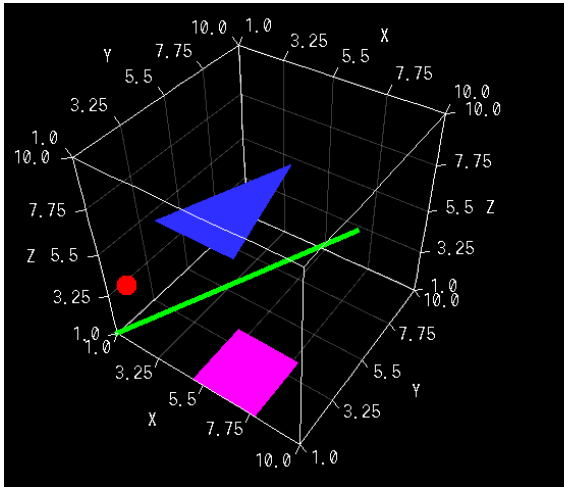
// 再描画が必要になった際に呼ばれるイベント処理
@Override
public void plottingRequested(RinearnGraph3DPlottingEvent e) {
    // 3D 形状の描画処理を再実行
    this.draw();
}

// 以下のイベント処理はここでは何もしない
@Override
public void plottingCanceled(RinearnGraph3DPlottingEvent e) {
}

@Override
public void plottingFinished(RinearnGraph3DPlottingEvent 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、メニューバーから「編集」>「範囲の設定」などを選んでグラフの範囲を変更しても、内容が消えてしまう事は無く、ちゃんと正しいグラフ範囲で表示されます。



これを実現するために、上のコードではまず、グラフの再描画が必要な(今の描画内容が消えてしまうような)タイミングをイベントとして受け取れるように、イベントリスナーの `RinearnGraph3DPlottingListener` インターフェースを実装して、リニアングラフ 3D に登録しています。そして、実際にイベントが発生した際に呼ばれる `plottingRequested` メソッド内で、3D 描画処理を再実行して描き直しています。

### ワンポイント!

**`plottingRequested` 等の中で `RinearnGraph3D` のメソッドは呼べない!**

ここで一つ注意が必要です。`RinearnGraph3DPlottingListener` 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 3D 本体側である `RinearnGraph3D` 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 `RinearnGraph3DRenderer` 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、`RinearnGraph3D` 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、`RinearnGraph3D` 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 `RinearnGraph3D` 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。

`RinearnGraph3DPlottingListener` 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うたら、上の点を踏まえて見直してみてください。

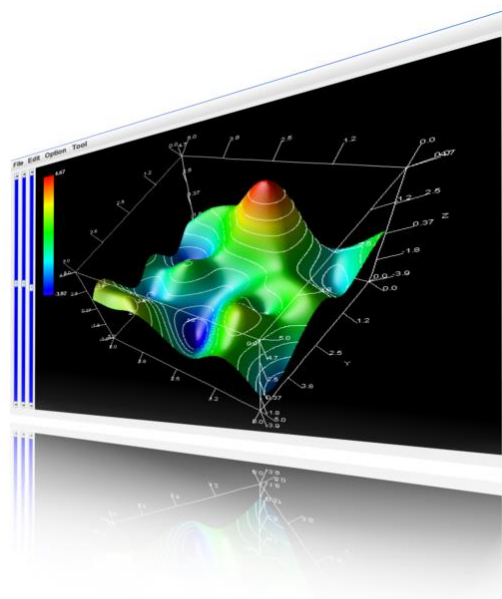
## 8.商標

[ 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 Graph3D 5.6 取扱説明書 初版

---

著者 松井文宏

この書籍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/graph3d/>

