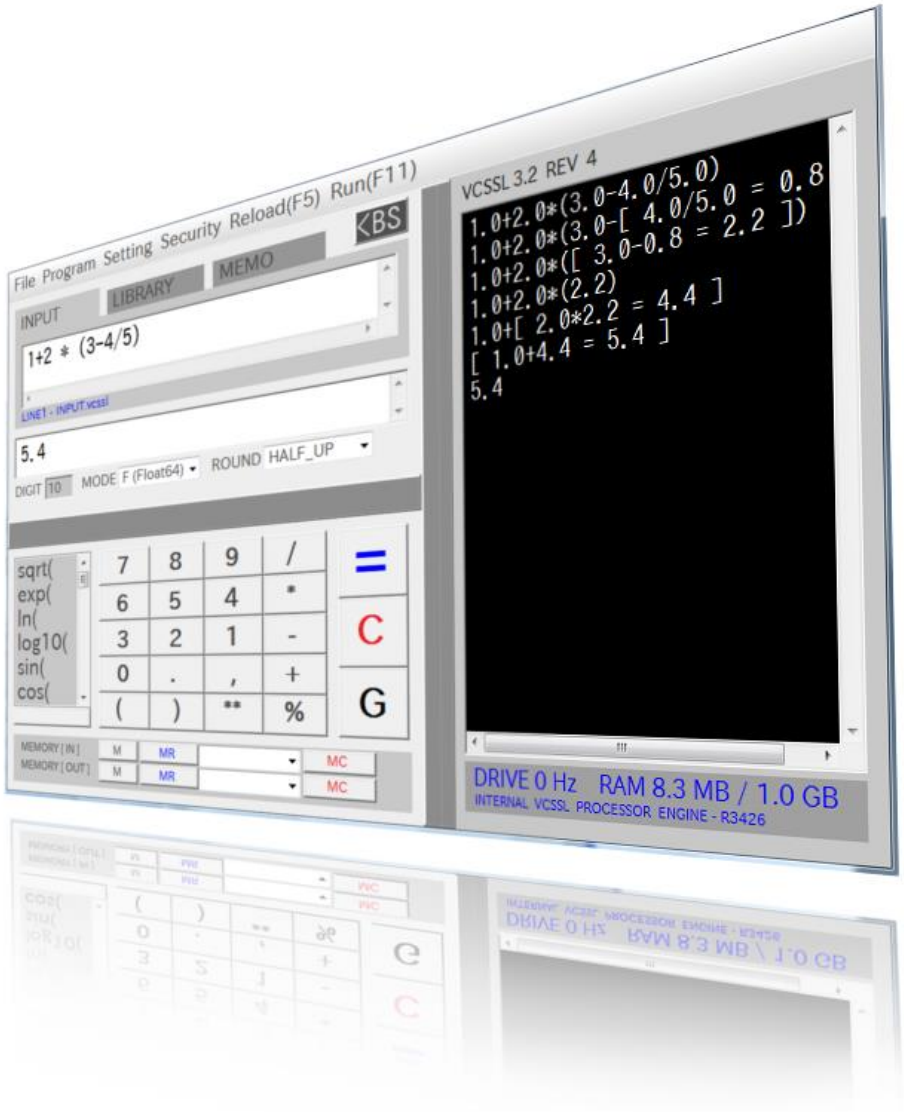
RINEARN
www.rinearn.com



RINEARN Processor 4.3

取扱説明書

第 6 版

■ 目次

はじめに

■ リニアンプロセッサーとは

リニアンプロセッサーは、途中式表示やグラフ描画機能をはじめ、プログラミングでの関数自作や自動処理などにも対応した、各種 OS 対応・無料の高機能な関数電卓ソフトです。インストール不要で利用でき、USB メモリーなどで持ち運んでの利用も可能です。

リニアンプロセッサー公式 Web サイト:
https://www.rinearn.com/ja-jp/processor/

■ ソフトウェアの利用ライセンスなどに関して

リニアンプロセッサーは、商用・非商用問わず、どなたでも無償で利用できます。ただし現時点のライセンスでは、特別に許諾された場合を除いて、ソフトウェア本体の不特定多数への再配布などは認められていないため、ご注意ください（後述の通り、リニアンプロセッサーで作成したグラフ画像やプログラムは自由に配布可能です）。

※ 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。

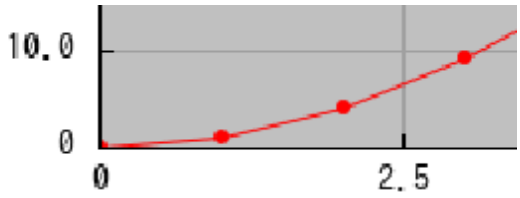
■ リニアンプロセッサーによる作成物の、権利と免責事項などに関して

リニアンプロセッサー開発元は、ユーザーがリニアンプロセッサーを使用して作成した数値、グラフ画像、データ、プログラムなどに関して一切の権利を主張しません。また、それら作成物に関する一切の責任も負担しません。それら作成物の著作権と責任は完全にユーザーに帰属します。

例えば、リニアンプロセッサーで描画・保存したグラフ画像の著作権は、その作成者に完全に帰属し、作成者の判断でご自由にご使用頂けます。報告やクレジット表記などは一切不要です（※ただし、使用するフォントの著作権にはご注意ください）。

■ グラフ画面の画質が粗い場合は…

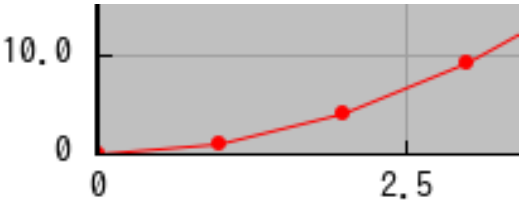
リニアンプロセッサーでグラフを描画した際、環境によっては、ウィンドウ上に表示されるグラフの文字や線の境界などが、ギザギザが目立つ印象で粗く表示される場合があります（例：下図）。



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。

起動方法

まずは、リニアンプロセッサーを起動してみましょう。リニアンプロセッサーは、各種オペレーティングシステムにおいて、ダウンロードしてすぐに起動できます（ インストール不要 ）。

■ 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プロセッサーの配布ファイルは、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　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）。

■ 起動

ZIP ファイルを展開できたら、さっそくリニアンプロセッサーを起動してみましょう ！

・Microsoft® Windows® をご使用の場合

ダウンロード・展開したフォルダ内にある、「 **RinearnProcessor_?.?.?.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）** 」をダブルクリックしてください。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、リニアンプロセッサーの画面が起動します（右図）。2 回目以降はすぐに起動します。

・Linux®等やその他の OS をご使用の場合

展開フォルダ内の「 **RinearnProcessor.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 RinearnProcessor.jar
```

と入力して実行できます。必要に応じて、「-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 RinearnProcessor.jar 」）。

※ java コマンドが使用できない場合は JRE の導入が必要です。apt コマンドが使える環境では：

```
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、リニアンプロセッサー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）。

■ コマンドライン端末上から起動したい場合は…

プログラミング用途などでは、リニアンプロセッサーを、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から起動したいかもしれません。その場合は後述の手順で、「bin」フォルダのパスを環境変数 Path/PATH に設定します。すると「 rinproc 」とコマンド入力するだけでリニアンプロセッサーが起動し、さらに以下のように引数にVCSSLプログラムを指定して開く事も可能になります：

```
rinproc  Test.vcssl
```

・Microsoft® Windows® をご使用の場合のパス設定

まずは、リニアンプロセッサー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、リニアンプロセッサーの「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、間違うと大変な事になるので、慣れた方に頼める場合は恐らく頼むのが無難です。

・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、リニアンプロセッサーのディレクトリを適当な場所へ配置してください。以下では例として「 /usr/local/bin/rinearn/rinearn_processor_?_?_?/ （?_?_?の箇所はバージョン番号） 」に配置したとします。この場所のさらに bin ディレクトリ内まで cd して以下のコマンドを実行します：

```
chmod +x rinproc          （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）
```

そして、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」（どれが有効かは環境依存）を開き、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_processor_?_?_?/bin/
```

※ ?_?_?の箇所はバージョン番号で置き換えて下さい

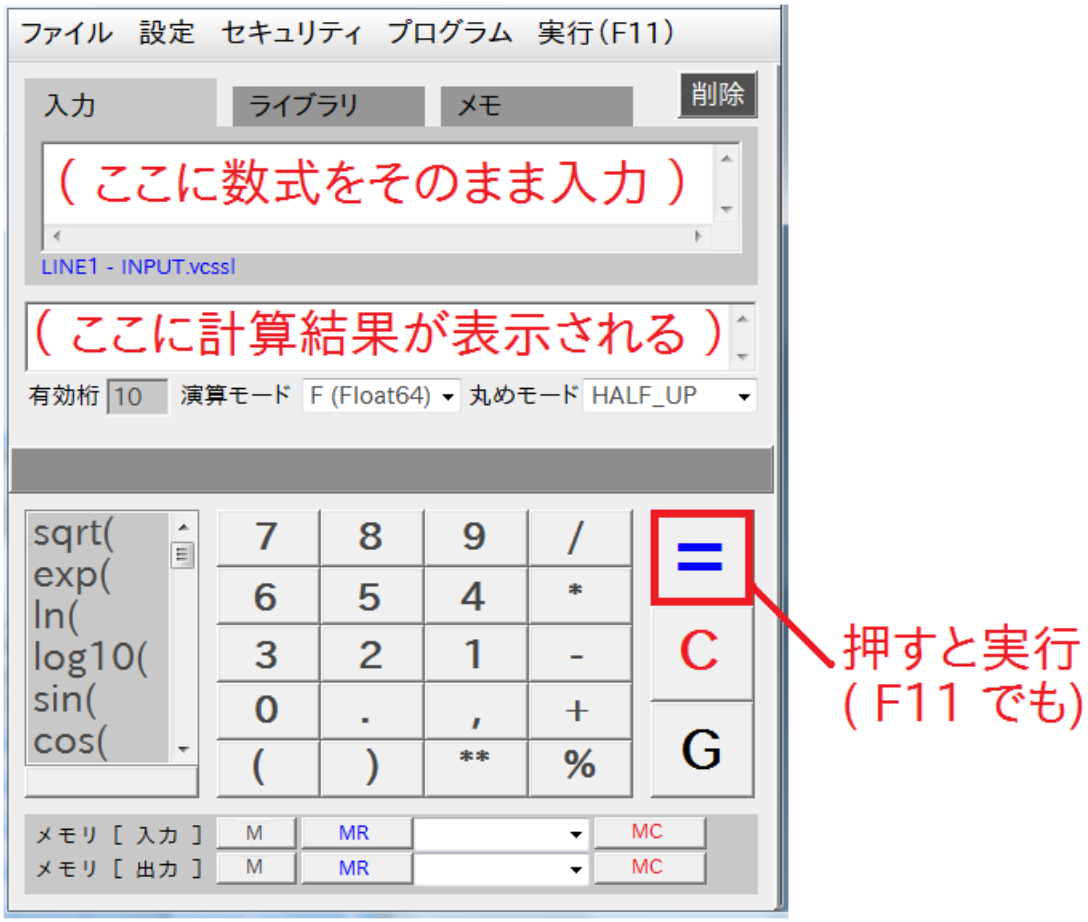
※ $PATH:以降の内容は、リニアンプロセッサーのディレクトリを配置した場所に合わせてください

数式を計算する

ここでは、電卓としての最も基本的な機能である、数式の計算を行ってみましょう。画面デザインや使い方は非常にシンプルで、基本的に普通の電卓と変わりません。

■ 起動時の画面

それではまず、リニアンプロセッサーを起動してください。すると、下図の画面が表示されます。



上図のように、起動時のデザインは、スタンダードな普通の電卓と全く変わりません。実際、基本的な使い方も、普通の電卓と同じです。**なお、数式を入力する「 入力 （INPUT） 」欄の文字サイズは、「Ctrl」+「U」キーで大きく、「Ctrl」+「D」キーで小さく調整できます。**

■ とりあえず計算してみる

それでは、実際に簡単な数式を計算してみましょう。上図で**「 （ ここに数式をそのまま入力 ） 」**と記載されている**「 入力 」欄**に、以下の数式を入力してください。

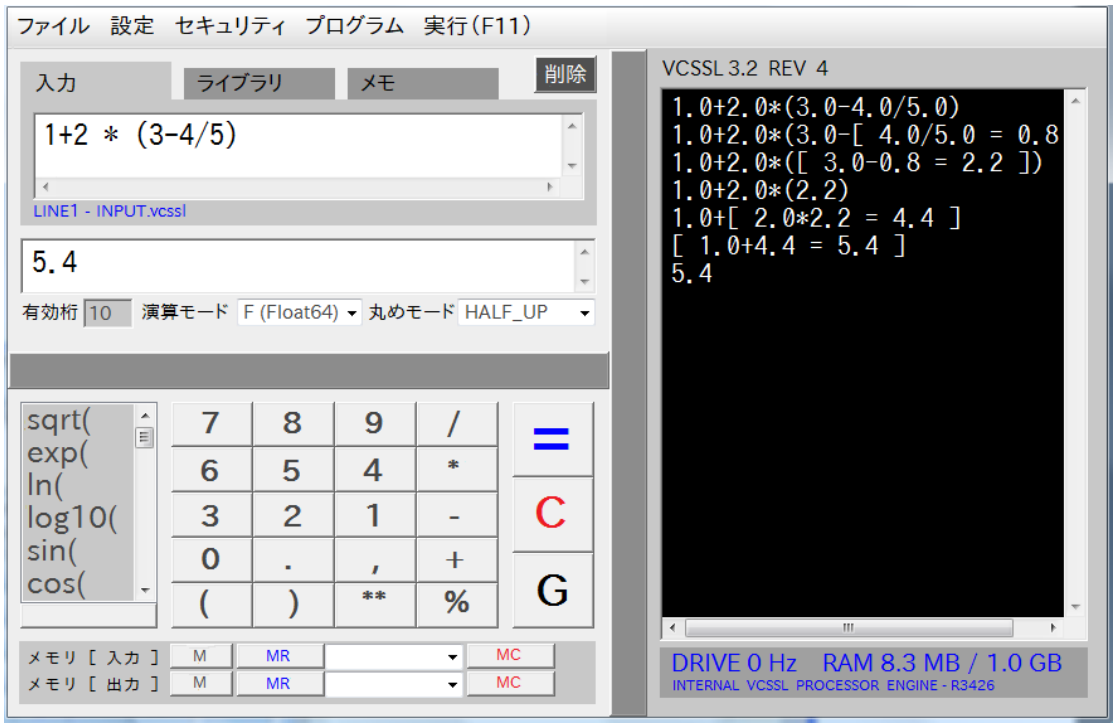
- 入力 -

```
1 + 2 * ( 3 - 4 / 5 )
```

数字や記号の入力は、もちろん画面に並んでいるボタンを押してもいいですし、キーボードから直接入力する事もできます。入力できたら、青色の**「 ＝ 」ボタン**を押してください。すると、上図で**「 (ここに計算結果が表示される) 」**と記載されている**「 出力 」欄**に、下記の計算結果が表示されます（ 下図参照 ）。なお、キーボードの**「 F11 」キー**で実行する事もできます。

- 出力 -

```
5.4
```



ところで、「 ＝ 」ボタンを押すと同時に、画面が横に拡がり、黒いエリアが表示されました。これは**「 コンソール 」欄**と言うもので、**計算の途中式や、エラーメッセージなどが表示される**領域です。実際、今回の計算では、以下のような途中式が表示されたはずです。

- コンソール -

```
1.0+2.0*(3.0-4.0/5.0)
1.0+2.0*(3.0-[ 4.0/5.0 = 0.8 ])
1.0+2.0*([ 3.0-0.8 = 2.2 ])
1.0+2.0*(2.2)
1.0+[ 2.0*2.2 = 4.4 ]
[ 1.0+4.4 = 5.4 ]
5.4
```

■ サポートされている演算と優先度

リニアンプロセッサーでは、下記の演算をサポートしています。

演算の記号（演算子）	演算の内容
+	加算（たし算）
-	減算（引き算）
*	乗算（かけ算）
/	除算（割り算）
%	剰余算（余り）
**	べき乗算（指数）

演算は、**優先度の高いものから先に処理**されます。優先度は、以下の順になっています。

べき乗算 ＞ 乗算 ＝ 除算 ＝ 剰余算 ＞ 加算 ＝ 減算

例えば、乗算（かけ算）は加算（たし算）よりも先に処理されます。なお、**同じ優先度の演算が並んでいる場合は、左から順に処理**されます。

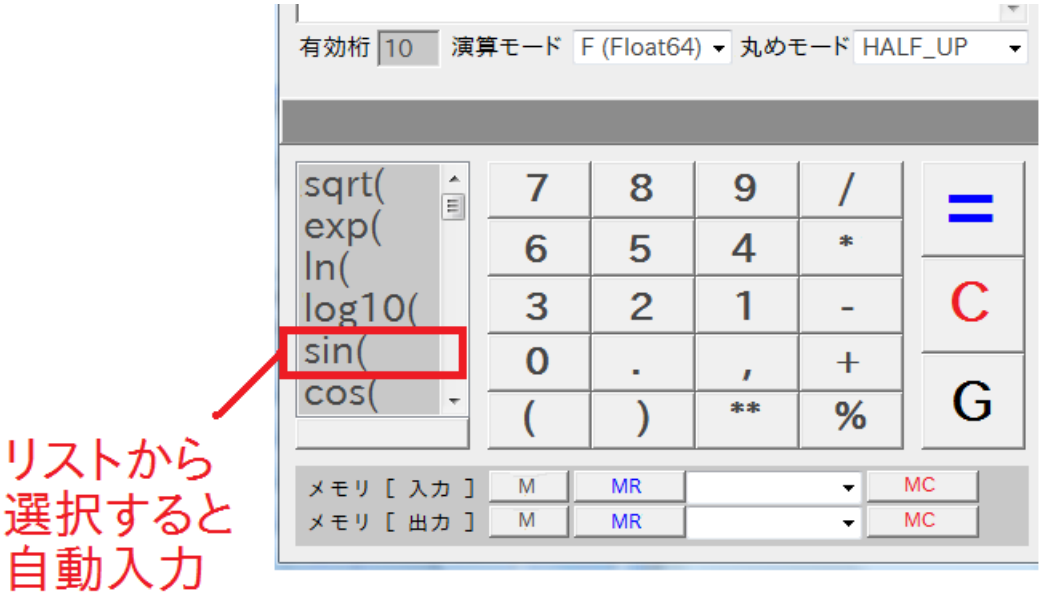
■ 関数を使う

今度は、sin や cos など、リニアンプロセッサーに最初から用意されている関数（一覧は後半の章参照）を使って計算してみましょう。先と同様に、以下の数式を計算してみてください。

- 入力 -

```
sin( 1 / 2 ) + cos( 3 / 4 )
```

この「sin」や「cos」という文字列は、キーボードから直接入力してもいいですが、画面左下にあるリストから選択すると、自動で入力されます。（下図参照）



この計算の結果は以下のようになります。

- 出力 -

```
1.211114407
```

そしてコンソールには、以下の途中式が表示されます。

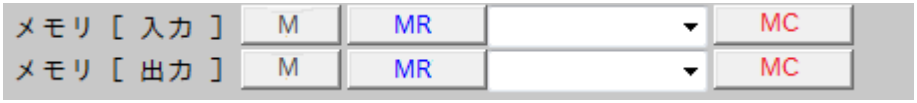
- コンソール -

```
sin(1.0/2.0)+cos(3.0/4.0)
sin([ 1.0/2.0 = 0.5 ])+cos([ 3.0/4.0 = 0.75 ])
[ sin(0.5) = 0.479425538604203 ]+[ cos(0.75) = 0.7316888688738209 ]
[ 0.479425538604203+0.7316888688738209 = 1.211114407478024 ]
1.211114407478024
```

■ 結果を記憶する ― メモリー機能

本章の最後に、計算結果や数式を記憶する、メモリー機能を使ってみましょう。リニアンプロセッサーのメモリー機能は、いくつでも無制限に記憶でき、さらに再起動後も内容が保持されるなど、とても強力です。

メモリー機能は、画面下部のパネルから使用します。(下図参照)



上下 2 つの段がありますが、上段が INPUT(入力)項目の内容を記憶するメモリー、下段がOUTPUT(出力)項目の内容を記憶するメモリーとなっています。各機能は以下の通りです。

「 M 」ボタン ― 現在の「入力」/「出力」欄の内容を記憶します。記憶内容は「 MR 」ボタンの横にあるリストに追加されます。

「 MR 」ボタン ― を押すと、横のリストで選択されている値が、「入力」欄に入力されます。

「 MC 」ボタン ― を押すと、リストに記憶されている値が全てクリアされます。

グラフを描く

ここでは、リニアンプロセッサーの特徴の一つである、グラフ描画機能を使ってみましょう。

■ y (x) 形式の 2 次元グラフを描画する

リニアンプロセッサーには、数式から 2D/3D グラフを描画できる機能が用意されています。ここでは、実際にこの機能を使用してみましょう。

まず、画面右下にある「 **G** 」ボタンを押してください。するとファイルを選択するウィンドウが表示されます。その中には、以下のような複数のファイルが存在するはずです。これらは「 **グラフプログラム** 」と言い、描画するグラフの種類に応じて選択します。

グラフプログラム一覧

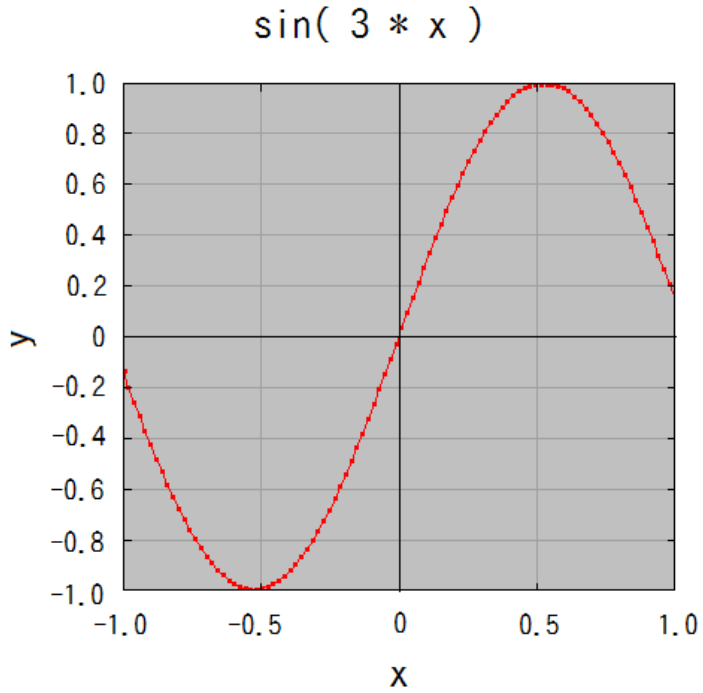
- **y(x)形式の 2D グラフ描画**
- **y(x,t)形式の 2D グラフ描画**
- **x(t),y(t)形式の 2D グラフ描画**
- **z(x,y)形式の 3D グラフ描画**
- **z(x,y,t)形式の 3D グラフ描画**
- **x(t),y(t),z(t)形式の 3D グラフ描画**

ここでは「 **y(x)形式の 2D グラフ描画** 」を選択しましょう。選択するとシステムがグラフモードになり、入力項目が並ぶウィンドウ（入力ウィンドウ）や、グラフ画面が出現します。入力ウィンドウでは、x-max 項目と x-min 項目でプロットする x 範囲、x-N 項目でプロット点数を指定できます。

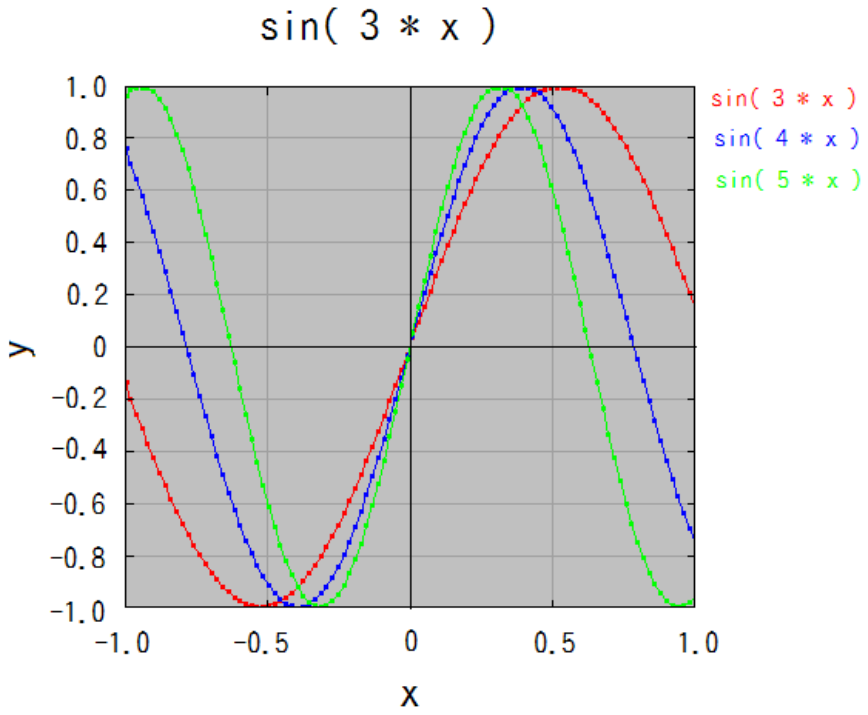
とりあえずここでは、「 y(x) = 」と書かれた項目に、以下のように式を記述し、「 **PLOT - プロット** 」ボタンを押してください。すると、グラフが描画されます。

- 「y(x) =」の項目 -

```
sin( 3 * x )
```



続いて、入力ウィンドウの「 y (x) = 」項目の内容を、「 sin(４ * x) 」と書き換えて、再度「 PLOT - プロット 」ボタンを押してください。すると、新しいグラフ線が、古いグラフ線の上に重ねて描画されます。さらに「 sin(５ * x) 」も重ねて描画させてみましょう。最終的に、下図のグラフが得られます。



■ z (x, y) 形式の 3 次元グラフを描画する

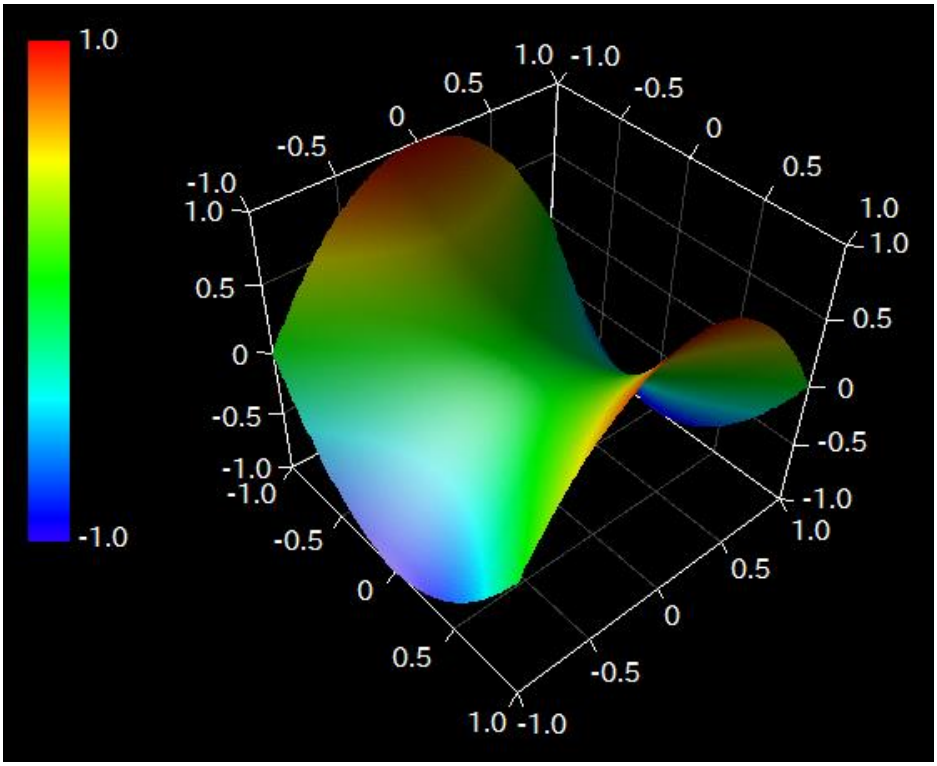
続いて、z(x, y)形式の 3 次元グラフを描かせてみましょう。

まず「 G 」ボタンを押し、今度は「 **z(x,y)形式の 3D グラフ描画** 」というグラフプログラムを選択してください。すると先ほどと同様、入力項目が並ぶウィンドウ（入力ウィンドウ）が表示されます。

とりあえずここでは、「 **z (x, y) =** 」と書かれた項目に、以下のように式を記述し、「 **PLOT - プロット** 」ボタンを押してください。すると、グラフが描画されます。

- 「z(x,y)=」の項目-

```
x * x - y * y
```



なお、ここで表示された 2D/3D グラフ表示画面は、単体のグラフソフトウェアとしても公開されており、様々なオプションや機能が利用できます。詳細は下記公式サイトをご参照ください。

・リニアングラフ 2D 公式サイト / https://www.rinearn.com/ja-jp/graph2d/
・リニアングラフ 3D 公式サイト / https://www.rinearn.com/ja-jp/graph3d/

■ その他の形式のグラフを描画する（ アニメーションなど ）

ここまでに使った他にも、アニメーション可能な「 **y(x,t)形式の 2D グラフ描画**」や「 **z(x,y,t)形式の 3D グラフ描画**」など、各種グラフプログラムが存在します。また、グラフプログラムは新たに入手したり、自分で独自に開発したりする事も可能です。詳しくは後半の「 プログラミング 」の章をご参照ください。ここでは例として、２次元のアニメーショングラフを描画する「 **y(x,t)形式の 2D グラフ描画**」の扱い方に触れておきます。まず「 G 」ボタンを押し、この名前のグラフプログラムを選択してください。すると入力項目が並ぶウィンドウが出現します。

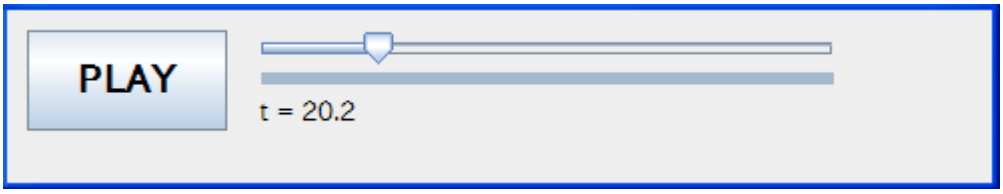
ここで「 y (x, t) = 」項目に以下のように入力してください。

- 「y (x,t) =」項目 -

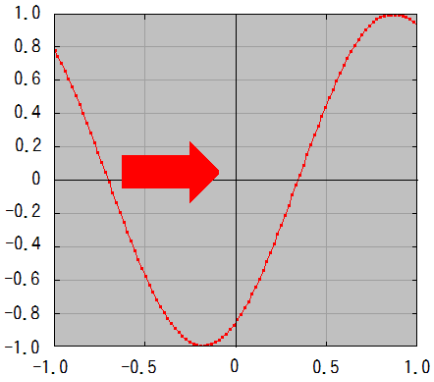
```
sin( 3 * x - t )
```

この「 t 」というのが時刻を表す文字です。アニメーション中は、この t の数値が時間経過に伴って変化していきます。

式を入力したら、続いて「 **SET - セット** 」ボタンを押すと、グラフ表示画面と、その上に「 **ANIMATION** 」と書かれたウィンドウが表示されます。



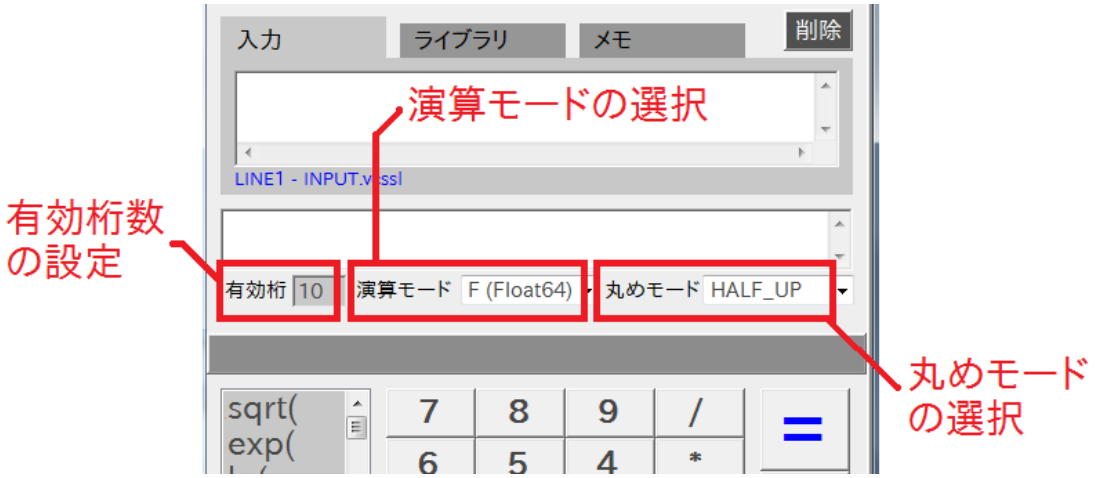
「 ANIMATION 」ウィンドウの「 **PLAY** 」ボタンを押すと、アニメーションの再生がスタートします。もう一度押すと再生がストップします。今回の例では、サイン波が右に進んでいきます。



桁数の設定と演算モード、演算精度

ここでは、表示する桁数の設定や、各演算モード時の精度について説明します。リニアンプロセッサーは、VF モードを使用する事で、大きな桁の演算も扱う事ができます。

「出力」欄の下で、桁数や演算モード、および丸めモードを選択できます。



■ 有効桁数の設定

有効桁数とは、簡単に言うならば **「 先頭から数えて何個有効な数字が続くか 」** という桁数です。厳密な解説はここでは割愛しますが、例えば 1.2345 も 123.45 も有効桁数は 5 桁です。

一般に、割り切れないような計算は何桁でも数字が続きますが、計算結果が表示される際には、「出力」欄の左下にある**「 有効桁 」**項目で設定した有効桁数に揃えられます。

なお、綺麗に割り切れる計算などで、計算結果の桁数が、設定された桁数よりも少なくなる場合もあります。そのような場合、標準設定では、少ないままの桁数で表示されます。もし、末尾に 0 の列を追加して、強制的に設定された桁数まで揃えたい場合（つまり有効桁数が何桁であるかを強調したい場合）は、**「 設定 」** メニューの **「 ゼロ揃え 」** 項目を有効にしてください。

■ 固定小数点表示

上で述べたように、「有効桁」項目の設定値は有効桁数であって、いわゆる**「 小数点以下が何桁か 」** というような桁数とは全く無関係です。計算結果を、小数点以下の決まった桁数で切り揃えたい場合（固定小数点表示）は、**「 設定 」** メニューの **「 固定小数点表示 」** 項目を有効にしてく

ださい。すると、「有効桁」項目があった場所（VF モード時はその下）、下に **「少数桁」** 項目が出現するので、ここで小数点以下の桁数を指定してください。

なお、固定小数点表示は、結果を小数点以下の決まった桁で **“ 切り揃えて ”** 丸めるだけです。なので、例えばそもそも計算結果において小数点以下 2 桁までしか数字が無いのに、少数桁項目で小数点以下 5 桁までに設定したとしても、もともと数字が無いので小数点以下 2 桁までしか表示されません。

従って VF モード時などには、もともとの有効桁数も足りるように設定しておく必要があります。ただし、有効桁数を大きく設定しても、「（1 以下の部分という意味での）小数点以下」が何桁になるかは、計算する内容によって異なるのでご注意下さい（コンソール上の表示値で確認できます）。

■ 丸めモードの選択

計算結果を、設定された有効桁数に揃える際、**端数がどのように丸められるか**は、「出力」欄の右下にある**「 丸めモード 」**項目で選択します。例えば「 UP 」を指定すれば切り上げ、「 DOWN 」を指定すれば切り捨てを行います。

それ以外の「 HALF_～ 」のどれかを指定した場合、値によって切り上げか切り捨てのどちらかが行われ、**最も近い数値**に丸められます。例えば設定桁以降の端数が 500…1 と続くなら切り上げ、499…9 と続くなら切り捨てが行われます。ただし、設定桁以降の端数がちょうど 500…0 と続く場合、切り上げと切り捨てのどちらが近いというわけでもありません。このような場合をどのように扱うかは、「 HALF_～ 」のどれを指定するかで異なります。

「 HALF_UP 」を指定すれば、設定桁以降の端数が 500…0 と続く場合は切り上げられます。そして設定桁以降が 499…9 と続く場合は切り捨てられます。つまりこの場合、**設定桁の次の桁を（それ以降の桁は無視して）四捨五入したのと同様の結果**が得られます。

「 HALF_DOWN 」を指定すれば、設定桁以降の端数が 500…0 と続く場合は切り捨てられます。そして 500…1 と続く場合は切り上げられます。

「 HALF_EVEN 」を指定した場合は少し特別です。この場合では、設定桁以降の端数が 500…0 と続くとき、切り上げるか切り捨てるかは、その前の桁が偶数か奇数かによって反転します（偶数方向へ丸めます）。

なお、「 NONE 」を指定した場合は丸めを行わず、**内部処理における演算結果をそのまま**返します。演算結果を自分で丸めたい場合に使用して下さい。なお「 NONE 」の場合は、指定された桁数にはならず、ゼロ揃え等のオプションも効きません。つまり、結果の値に何も行われません。

■ 演算モードの選択

少しコンピュータの仕組みに関わる内容になりますが、リニアンプロセッサーにおける数値の演算は、標準（F モード）では **「 2 進数 64bit 浮動小数点数 」** という形式で扱われます。これはコンピュータにおいて非常に高速に計算できる形式で、10 進数換算で約 16 桁前後の値を扱えます（ただし、2 進数特有の誤差を考慮する必要があります）。

場合によっては、16 桁を超える大きな桁数の小数を扱いたい場合や、2 進数誤差を嫌って 10 進数で演算してほしい場合もあります。また、小数ではなく整数として演算してほしい場合もあるでしょう。こういった様々な用途に対応するため、リニアンプロセッサーには、以下の 3 つの**「 演算モード 」** が存在します。

演算モード	**説明**
F (float64)	すべての入力値を約 16 桁の小数(2 進 64bit 浮動小数点数)として扱います。
VF (varfloat)	すべての入力値を任意桁の小数（10 進多倍長浮動小数点数）として扱います。
DIRECT	整数や小数など、異なる種類（型）の値を混在して扱います。型は C 言語系の**数値リテラル表記**で判断されます。例えば整数なら、「123」は 10 進数、「0b101」は 2 進数、「0o123」は 8 進数（※）、「0x123abc」は 16 進数とみなされます。「1.23」は 64bit 浮動小数点数、「1.23vf」は多倍長浮動小数点数となります。 ※ 8 進数リテラルについては、バージョン 4.2 までは 「 0（ゼロ）を頭に付ける 」という標準的なルールを採用していましたが、慣れない場合の混乱を防ぐため、バージョン 4.3 以降では「 0o（ゼロ・オー） 」を頭に付けるという記法に移行しました（VCSSL も同様です）。従来の記法も使用できますが、警告メッセージが表示されます。表示したくない場合は「設定」メニューから「互換性に関する警告を無視」を有効にしてください。

普通の電卓として特に重要なのは F モード（標準）と VF モードでしょう。これらは両者とも小数の演算を行うモードですが、**精度（有効桁数の限界など）と処理速度**が大きく異なります。特徴を次ページにまとめます。

■ F モード と VF モードの主な特徴

下の表は、一般的な用途向けである、F モードと VF モードの特徴について詳しくまとめたものです。

	F モード（標準）	VF モード
概要	内部処理に 64bit の 2 進数が使用されます。最大 16 桁程度まで扱えます。非常に高速で、数学関数も軽快で正確です。反面、末尾の数桁に、2 進数演算に特有の誤差などが生じます。	内部処理に任意桁数の 10 進数が使用されます。何桁でも扱えて、高精度です。反面、大きな桁数では、処理速度が低下します。特に数学関数は独自開発ライブラリのため、比較的重く、精度検証もまだ完全ではありません。
演算精度（桁）	限界で 16 桁程度です。信頼できるのは 14 桁程度で、現実的には、安全マージンを取ると 10 桁程度です。	任意桁数なので、必要に応じていくらでも桁数を増やせます。ただし桁数を増やすと、演算速度は低下します。
内部処理の型	64bit, 2 進数（ double ）	多倍長, 10 進数
内部処理における丸め(※)	開発言語における組み込み型（ double ）の丸め処理がそのまま使用されています。	Ver.4.2.48（VCSSL 3.3.12)以降から段階的に Half-even に移行中です。それ以前は単純な切り捨てです。
想定すべき誤差など	10 進/2 進変換に特有の誤差などに注意が必要です。**末尾の数桁には常に誤差が含まれると考えるべきです。**	10 進/2 進変換の誤差を気にする必要はありません。ただしあくまで有限桁なので、丸め誤差などは存在します。
演算速度	100 M FLOPS 程度 （ 数億回 / 秒 ）	最大 1 M FLOPS 程度 ※桁数が大きくなると遅くなります。
数学関数	高速で正確です。 ※ ただし、一般のプログラミングでの数学関数の使用時と同様、必ずしも 64bit の限界までの精度が出るわけではなく、精度は関数によります。	歴史の浅い、独自開発のライブラリを使用しているため、比較的低速であり、精度検証も完全ではありません（使用前のテストが推奨されます）。 その代わり、sin 関数の値を千桁求める等、普通の電卓では不可能なほどの長い桁数の値を求める事が可能です。

※ここでの**「内部処理における丸め」**とは、先に述べた**「丸めモード」の設定項目とは無関係**であり、「内部で演算を行う際の丸め」の事を意味します（内部処理では、設定桁数よりも余裕を持った桁数で演算が行われます）。そうして演算を行った最終的な結果が、「丸めモード」で設定した方法で、設定桁数に丸められて表示されます。

■ 実際には設定桁数よりも大きな桁数で演算され、表示時に丸められる

さて、上の表では、精度や誤差など、細かい点が色々と記載されていますが、10 桁程度の計算で、小数を普通に扱うような場合であれば、このような点をあまり気にし過ぎる必要は無いでしょう。

なぜならリニアンプロセッサーは、**内部処理では設定桁数よりも大きな桁数で演算を行い、結果表示時に指定桁まで丸めてから**、「出力」欄に出力するからです。例えば、Fモードで有効桁が 10 桁に指定された状態で、以下の数式を計算してみましょう。

- 入力 -

```
1.01 - 0.001
```

- 出力 -

```
1.009
```

- コンソール -

```
1.01-0.0010
[ 1.01-0.0010 = 1.0090000000000001 ]
1.0090000000000001
```

さて、コンソールに注目してください。この場合、内部処理では 17 桁の結果が得られた事が分かります。そして、その**最終桁は、本来「 0 」であるべきなのに「 1 」になっている**事が見て取れます（これはコンピューターで２進数で小数点付きの数値を処理する事に起因する、有名な誤差の一種でによるものです）。こういった誤差は、内部処理における末尾桁の数桁まで影響する可能性があります。しかしながら、**結果は 10 桁に四捨五入（丸めモードに依存）されてから出力されたため、末尾の誤差「…001」の影響は切り捨てられた**事が分かります。

上のように、多くの場合は丸めによって補正されますが、**しかしながら、F モードにはこういった類の誤差が常に存在するという事は、あらかじめ認識した上で使用して下さい。**そして、これが問題となる場合には、小数演算では VF モード、整数演算では DIRECT モードなどを使い分けて下さい。

■ 大きな桁数と精度が必要な場合は、VF モードを使う

F モード時には、内部処理でも最大で 16～18 桁までしか扱えないため、**10 桁よりも大きい桁数で計算させる際は注意が必要**です。極端な例では、F モード時に 16 桁などで計算させると、桁数がギリギリで丸める余地が無くなってしまい、誤差が結果に見える領域に現れる可能性があります。

どうしてもそのような大きな桁数を扱いたければ、VF モードを使う事が推奨されます。VF モードでは、内部処理で無制限に桁を使えるため、ちゃんと安全マージンを取って計算した上で、丸めて表示する事ができます。

実際、先ほどの例では、VF モード時の場合、16 桁に設定しても内部ではより大きな桁数で計算され、16 桁に丸められます（丸めモードに依存、標準の HALF_UP 時は四捨五入されます）。

また、VF モードでは数千桁の計算も扱う事ができます。例として、**VF モード**を使い、**桁数を 1000 桁に指定**して、以下の数式を計算してみてください。（ 少し時間がかかります ）

- 入力 -

```
sin(1)**2 + cos(1)**2
```

- 出力 -

```
1.0...（ 1000 桁続く ）
```

このように、1000 桁の正しい値を得る事ができました。

ただし、VF モード時における数学関数の処理には、まだ歴史の浅い、独自開発のライブラリが使用されています。そのため、あらゆる入力値と桁数について、精度が完全に確認されているわけではありません。

VF モードで数学関数を使う場合、厳密性を要求されるような場面では、上のように結果が分かっている数式を用いて、予め動作検証を十分に行う事が推奨されます。

2 進/8 進/10 進/16 進数の整数演算

ここでは、DIRECT モードを使用した、2 進/8 進/10 進/16 進数の整数演算を扱います。

■ 数値ごとに異なる種類（型）を混在させる「 DIRECT 」モード

ここまでは、入力された数値をすべて小数として扱う、F モードと VF モードのみに絞って解説してきました。その理由は、それらが電卓ソフトとして分かりやすい挙動をするからです。対して、ここで扱う「 **DIRECT 」モード**は、電卓というよりはプログラミング言語に近い挙動をするため、慣れないと少し分かり辛い挙動をします。しかし、2 進数や 16 進数などが扱えるなど、強力な特徴もあります。

DIRECT モードでは、一行の数式の中でも、数値ごとに異なる種類（型）のものとして扱われます。特にリニアンプロセッサーで常用するのは、**整数（int 型）**と**小数（float 型）**という 2 種類が挙げられます。例えば、以下の数式において、「21」は整数、「1.2」は小数として扱われます。

- 入力 -

```
21 + 1.2
```

このように、小数の場合は「 **小数点があれば少数として扱われる** 」という、分かりやすいルールがあります。これに対して、少しややこしいのは整数です。

■ 整数には、何進法で記述するかによって、複数の記述法がある

整数には、以下のように複数の記述ルールがあります。

基数	例（括弧内は10進法表現）	記述ルールの詳細
10 進数	12345　　(12345)	普通に整数を記述する（小数点は付けない）。
2 進数	0b1010010　(82)	先頭に「0b」を付け、それ以降を 2 進数で記述する。
8 進数	0o23532　　(10074)	先頭に「0o」を付け、それ以降を 8 進数で記述する。
16 進数	0x12ac14　　(1223700)	先頭に「0x」を付け、それ以降を 16 進数で記述する。

このように整数では、10 進数（10 進法で記述した数）の場合を除いて、先頭に特殊な表記を付加します。そうすると、その通りにリニアンプロセッサーに解釈され、扱われます。

例として、16 進数（16 進法で記述した数）の整数を使用してみましょう。「入力」欄に以下のように記述し、その下の**「 演算モード 」項目を「 DIRECT 」にした上で**実行してみてください。

- 入力 -

```
0xff
```

これは 16 進数で「ff」、つまり 10 進数の「255」と同じ数値です。なお、16 進法での一桁は

1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f

というように表します。10 進法では 9 の次に 10 に繰り上がりますが、16 進法では 16 ではじめて次の桁に繰り上がります。つまり 16 進数の f は 15 を意味し、16 進数の 10 は 16 を意味します。

- 出力 -

```
255
```

結果は上の通り、入力値を 16 進数と解釈した上で、それを 10 進数に変換した「255」が得られました。このように、出力は標準では 10 進数に変換して表示されます。

逆に、普段日常で使っている 10 進数の数を、2 進数/8 進数/16 進数で表示したければ、以下のように対応する変換関数で包みます：

- 入力 … 2 進数として表示したい場合 -

```
bin( 255 )
```

- 入力 … 8 進数として表示したい場合 -

```
oct( 255 )
```

- 入力 … 16 進数として表示したい場合 -

```
hex( 255 )
```

これで望む基数で表現した数値が得られます。結果には先頭に「0b」「0」「0x」が付加されます。

上の **hex( )** ように、括弧（かっこ）の前に特定のキーワードが付いたものは、数学と同様に関数と呼びます。ここでの hex 関数は、sin や cos といった、通常の数学関数と同じようなものだと思っても問題ありません。つまり **hex は 括弧の中身を 16 進数に変換する関数**という事です。

■ 基数が異なる整数の混合演算

整数の四則演算では、異なる基数（16 進数と 10 進数など）の整数を混在させる事が可能です（内部処理ではすべて 2 進数に変換して扱われます）。

- 入力 … 16 進数と 10 進数の混合四則演算 -

```
0xf2 + 14
```

- 出力 -

```
256
```

このように、16 進数の 0xf2（10 進数で 242）と、10 進数の 14 で加算を行い、正しい答えとして 256 を得られました。

これを 16 進数で表示するには、**全体を hex( ) で囲みます**。

- 入力 … 16 進数と 10 進数の混合四則演算結果を 16 進数で表示 -

```
hex( 0xf2 + 14 )
```

- 出力 -

```
0x100
```

この通り、256 を 16 進法で表現した正しい答え「100」を得ました。先頭の 0x は、それが 16 進数である事を現すだけで、読む際は数に含めない事に注意してください。

■ 整数と小数の混合演算

整数と小数を混在させた演算も可能です。10 進数同士だけでなく、他のものも混在可能です。

- 入力 -

```
0x100 * 1.7
```

- 出力 -

```
435.2
```

0x100（10 進数の 256 と同じ） に 1.7（10 進数） を掛けた結果が、10 進数として表示されました。このように、**整数と小数の混合演算では、結果は小数として扱われる**というルールがあります。

なお、これをそのまま 16 進数にしようと hex( )で囲うとエラーとなります。なぜなら **hex( ) 関数は、括弧の中身が必ず整数でなければ受け付けない**という決まりになっているからです。つまり小数を 10 進数以外で表示する事はできません。

しかしどうしても必要な場合には、以下のようにして**小数点以下を切り捨て、無理やり整数に変換**して使う事もできます。これは**キャスト**と呼ばれる強引な演算で、切捨ての分だけ精度が落ちます：

- 入力 -

```
hex( (int)( 0x100 * 1.7 ) )
```

- 出力 -

```
0x1b3
```

このように、(int)( ) で囲った中身の小数点以下が切り捨てられ、435 になったうえで、それを 16 進法で表現した結果が得られました。

■ ビットごとの論理演算 （ ビット AND、ビット OR、ビット XOR ）

整数では、**「 ビットごとの論理演算 」**という、特殊な演算を扱う事ができます。これはやや専門的な分野で使われるものであり、一般には馴染みが無いため、あまり深くまで踏み込んだ解説は割愛します。

リニアンプロセッサーでは、ビットごとの論理演算は、演算子ではなく関数として提供されます。これは、C 言語系の言語でビット排他的論理和にマッピングされる「 ^ 」演算子が、一般の電卓ソフトではべき乗（指数）の演算子として普及しているため、誤用を避けるための仕様です。

and( )がビット論理積、or( )がビット論理和、xor( )がビット排他的論理和を返す関数です。これらはパラメータ（引数）を 2 つ要するので、コンマ記号「 , 」で区切って記述します。

※ これらの関数は、「 設定 」 > 「 カンマ付き数値 」項目を OFF にしないとエラーになります。

- 入力 -

```
bin( or( 0b101010 , 0b010101 ) )
```

- 出力 -

```
0b111111
```

このように、2 つの 2 進数「101010」と「010101」に対して、ビット（2 進数での桁）ごとに論理和（どちらか片方でも 1 なら 1 になる）をとった結果「111111」が得られました。なお、and を使うとビットごとの論理積（両方 1 の場合だけ 1 になる）となり、 0 （ 0b0 ） が得られます。

■ 整数同士の除算（割り算）には注意が必要

DIRECT モードで整数同士で除算（割り算）を行う場合には、注意が必要です。詳細は「VCSSL のプログラミングガイド」フォルダ内のガイド参照ですが、いくつかのプログラミング言語（C 言語も）では、**整数同士の演算は整数になるというルールがあり、それにより、整数同士の除算では、結果の小数点以下が切り捨てられます。**これは DIRECT モードでも同様です。

つまり DIRECT モードでは、1/2 が 0 になります。

このルールは慣れるまでミスを招きがちなため、注意が必要です。

変数や関数を使用する

ここでは、リニアンプロセッサーで標準サポートされている変数（定数）や関数をご紹介します。

■ 標準でサポートされている変数（定数）や関数

リニアンプロセッサーでは、いくつでも独自の変数や関数を定義する事が可能ですが、一般的なものは最初から用意されており、最初から使用できます。ここでは、それらをご紹介します。

■ 変数（定数）

・PI

使用例 ： PI

内容 ： 円周率の値（3.141592653589793）です。

・pi( )

使用例 ： pi( )

内容 ： こちらも円周率ですが、定数では無く、円周率を求める関数です。VF モードで使用すると、1 万桁程度まで円周率の値を求める事ができます。この関数は定数値を求めるため、引数を取りません。この関数が求める桁数は、VF モードの設定桁数がそのまま適用されます。なお、計算のアルゴリズムにはガウス=ルジャンドル法が使用されます。

・E

使用例 ： E

内容 ： 自然対数の底（自然対数の底）の値（2.718281828459045)です。

■ 基本的な関数

・ sqrt(平方根を求める値)　※日本語で「 平方根(…) 」も使用可
使用例 ： sqrt(2)　または　平均(2)
内容 ： 平方根を求めます。

・ exp(指数を求める値)
使用例 ： exp(2)
内容 ： 自然対数の底（ネイピア数）による指数関数を求めます。

・ ln(自然対数を求める値)　※日本語で「 自然対数(…) 」も使用可
使用例 ： ln(100)　または　自然対数(100)
内容 ： 自然対数を求めます。

・ log10(常用対数を求める値)　※日本語で「 常用対数(…) 」も使用可
使用例 ： log10(100)　または　常用対数(100)
内容 ： 常用対数（10 を底とした対数）を求めます。

・ fac(階乗を求める値)　※日本語で「 階乗(…) 」も使用可。
使用例 ： fac(5)　または　階乗(5)
内容 ： 階乗の値を求めます。

・ abs(絶対値を求める値)　※日本語で「 絶対値(…) 」も使用可
使用例 ： abs(-2)　または　絶対値(-2)
内容 ： 絶対値を求めます。

・ rad(ラジアン値を求める度数)　※日本語で「 ラジアン(…) 」も使用可
使用例 ： rad(45)　または　ラジアン(45)
内容 ： 角度の度数をラジアン値に変換します。

・ deg(度数を求めるラジアン値)　※日本語で「 度(…) 」も使用可
使用例 ： deg(PI / 4)　または　度(PI/4)
内容 ： 角度のラジアン値を度数に変換します。

■ 三角関数

・ sin(角度のラジアン値)

使用例 ： sin(PI/2)

内容 ： sin 関数の値を求めます。角度はラジアンで指定してください。

・ cos(角度のラジアン値)

使用例 ： cos(PI)

内容 ： cos 関数の値を求めます。角度はラジアンで指定してください。

・ tan(角度のラジアン値)

使用例 ： tan(PI/4)

内容 ： tan 関数の値を求めます。角度はラジアンで指定してください。

■ 逆三角関数

・ asin(角度を求めたい sin 値)

使用例 ： (asin(1) + acos(1))*2

内容 ： arcsin 関数の値を求めます。sin 値を指定すると、対応する角度をラジアンで返します。

・ acos(角度を求めたい cos 値)

使用例 ： (asin(1) + acos(1))*2

内容 ： arccos 関数の値を求めます。cos 値を指定すると、対応する角度をラジアンで返します。

・ atan(角度を求めたい tan 値)

使用例 ： atan(1) * 4

内容:arctan 関数の値を求めます。tan 値を指定すると、対応する角度がラジアンで返されます。

■ 双曲線関数

・ sinh(値)
使用例 ： sinh(0.1)
内容 ： ハイパボリックサイン関数の値を求めます。

・ cosh(値)
使用例 ： cosh(0.1)
内容 ： ハイパボリックコサイン関数の値を求めます。

・ tanh(値)
使用例 ： tanh(0.1)
内容 ： ハイパボリックタンジェント関数の値を求めます。

■ 逆双曲線関数

・ asinh(対応値を求めたい sinh 値)
使用例 ： asinh(5.0)
内容 ： arsinh 関数の値を求めます。

・ acosh(対応値を求めたい cosh 値)
使用例 ： acosh(5.0)
内容 ： arcosh 関数の値を求めます。

・ atanh(対応値を求めたい tanh 値)
使用例 ： atanh(0.1)
内容 ： artanh 関数の値を求めます。

■ 統計関数

・ **sum(値 1, 値 2, 値 3, … , 値 N)　※日本語で「 和（…） 」も使用可**
使用例 ： sum(1, 2, 3, 4, 5)　または　和(1, 2, 3, 4, 5)
内容 ： 複数の値を受け取り、それらの和を返します。

・ **mean(値 1, 値 2, 値 3, … , 値 N)　※日本語で「 平均（…） 」も使用可**
使用例 ： mean(1, 2, 3, 4, 5)　または　平均(1, 2, 3, 4, 5)
内容 ： 複数の値を受け取り、それらの平均値を返します。

・ **van(値 1, 値 2, 値 3, … , 値 N)　※日本語で「 分散（…） 」も使用可**
使用例 ： van(1, 2, 3, 4, 5)　または　分散(1, 2, 3, 4, 5)
内容 ： 複数の値を受け取り、それらの分散(分母は N)を返します。

・ **van1(値 1, 値 2, 値 3, … , 値 N)　※日本語で「 不偏分散（…） 」も使用可**
使用例 ： va(1, 2, 3, 4, 5)　または　不偏分散(1, 2, 3, 4, 5)
内容 ： 複数の値を受け取り、それらの不偏分散(分母は N-1)を返します。

・ **sdn(値 1, 値 2, 値 3, … , 値 N)　※日本語で「 標準偏差（…） 」も使用可**
使用例 ： sdn(1, 2, 3, 4, 5)　または　標準偏差(1, 2, 3, 4, 5)
内容 ： 複数の値を受け取り、それらの標準偏差(平方根内の分母は N)を返します。

・ **sdn1(値 1, 値 2, 値 3, … , 値 N) ※日本語で「 不偏標準偏差（…） 」も使用可**
使用例 ： sd(1, 2, 3, 4, 5)　または　不偏標準偏差(1, 2, 3, 4, 5)
内容 ： 複数の値を受け取り、それらの不偏標準偏差(平方根内の分母は N-1)を返します。

変数や関数を定義する

ここでは、独自の変数や定数を定義し、数式中で使用する方法について解説します。

■「 ライブラリ 」欄

リニアンプロセッサーでは、独自の変数や関数を、いくつでも自由に定義し、数式中で使用する事ができます。それらは**「 ライブラリ 」**欄に定義します。「ライブラリ」欄には、「入力」欄の上にある**「 ライブラリ 」**と書かれたタブをクリックすると書き込めます。

なお、「ライブラリ」欄の編集内容は自動的に保存されます。そのため、特に保存したり、読み込んだりする必要はありません。万一、色々と間違ってエラーだらけになってしまった場合は、etc フォルダ内にある 「 LIBRARY.vcssl 」 を削除すれば、初回起動時のものが再生成されます。

■「 import Math ; 」は消してはいけない

「ライブラリ」欄の先頭付近には、以下の一行が記述されているはずです。

- ライブラリ -

```
import Math ;
```

この行は、**sin や cos など、リニアンプロセッサーで最初から使える関数を読み込む**ためのものです。
従って、**この行を消してはいけません。**消してしまうと sin や cos が使えなくなります。

■ 変数を定義する

それでは、変数を定義してみましょう。変数の定義は、以下のような 1 行を書き加えて行います。

```
float 変数名 = 値 ;
```

最初の「 float 」というのは、**小数を扱う変数**を定義するための用語だと思ってください（型と呼びます）。また、**行末の「 ; 」記号はついつい忘れがち**なので、注意が必要です。

実際に、**x という名前**で、**1.0 の値**を持つ変数を定義してみましょう。「ライブラリ」欄に一行だけ追記し、以下のようにします。

- ライブラリ -

```
import Math ;

float x = 1.0 ; // x という名前で、1.0 という値の変数を定義
```

追記された最終行の、**// （ ダブルスラッシュ ）以降はコメントであり、無視されます。**これで、**x という名前**で、**1.0 の値**を持つ変数が定義できました。

定義した変数 x を使って、実際に計算を行ってみましょう。「 入力 」タブをクリックして「 入力 」欄に戻り、以下の数式を計算してみてください。

- 入力 -

```
1 + x
```

この計算の結果は以下の通りです。

- 出力 -

```
2.0
```

このように、正しい値が得られました。

■ 関数を定義する

続いて、関数を定義してみましょう。関数は、以下のような形で定義します。

```
float 関数名( float 引数名 ) {
    return 計算内容 ;
}
```

ここでも「 float 」というキーワードが登場していますが、とりあえず今はあまり気にしないでください。小数を受け取って、小数を返すという意味なのですが、最初は全体を暗記した方が早いです。

「 引数（ひきすう） 」という耳慣れない言葉も登場していますが、これは関数を使う際、( ) の中に書いた値が格納される、特別な変数です。パラメータと言った方が分かりやすいかもしれません。

それでは例として、「 half 」という名前で、受け取った値を２で割って返す関数を定義してみましょう。ライブラリに内容を追記し、以下のようにします。

- ライブラリ -

```
import Math ;

// 値を 2 で割って返す関数
float half( float a ) {
  return a / 2.0 ;
}
```

ここで、a/2 ではなく、a/2.0 としている事に注意が必要です。詳しい説明は割愛しますが、**整数同士で割り算を行った場合、余りが切り捨てられ、値が整数になってしまいます。**上のように数値に小数点を付けておくと、うっかりそうなってしまう事を防げます。これは不具合ではなく、VCSSL というプログラミング言語の仕様上、そういうルールになっているためです（C 言語などでも同様）。

実は、上では a を整数ではなく小数点付きの数値と見なす事を「float」という語で宣言していて、 a / 2 としても問題は無いのですが、慣れるまではうっかりミスがよくあるので、今はとりあえず**「 ライブラリ欄の中で割り算を行う際、数値には小数点を付けておく 」**と覚えておいてください。

それでは、実際に half 関数を使用してみましょう。

- 入力 -

```
half( 1 )
```

ここで、括弧内に「1」と書いて、half 関数を使用しました。**この呼び出し時に書いた「1」という値が、half 関数の「a」に入ります。**この性質が、関数の最も重要な点です。「1」に対する計算、「2」に対する計算…というように、何度も同じような処理を書かなくても、**とりあえず「a」と置いて処理を書けばいい**わけです。そして、**使う時に a に具体的な数値が入る**というわけです。

さて、この計算の結果は以下の通りになります。

- 出力 -

```
0.5
```

このように正しい値が得られました。

■ 引数が複数ある関数

関数では、引数を複数使用する事もできます。それには、引数をカンマ記号で区切って定義します。例として、2 つの値の積を返す関数 mul を定義してみましょう。「ライブラリ」欄に以下のように記述します。

- ライブラリ -

```
import Math ;

// 2 つの値の積を返す関数
float mul( float a, float b ) {
  return a * b ;
}
```

それでは「入力」タブをクリックして「入力」欄に戻り、以下の数式を計算してみましょう。

- 入力 -

```
mul( 2, 8 )
```

この計算の結果は以下の通りです。

- 出力 -

```
16
```

このように正しい値が得られました。

■ VF モード時に使用する変数と関数

ここまでで扱ってきた変数と関数の定義は、**F モードと DIRECT モードの時**にしか使用できません。使おうとしても、引数の型が一致しないという旨のエラーが表示されます。

それでは、**VF モードに使用する変数と関数の定義**は、どうすればよいのでしょうか。

それにはまず、これまで何度も登場した「 **float** 」というキーワードの代わりに、「 **varfloat** 」というキーワードを使用する必要があります（ ※ 実は VF モードの「 **VF** 」は、「 **V**ar**F**loat 」の略です ）。そして、1.0 や 2.0 などの数字の末尾に、**vf** の 2 文字を付けるようにします。2 進数や 16 進数の整数では先頭に 2 文字を付けましたが、それと同じ感覚で、今度は末尾に付けるだけです。

実際に、上で定義した変数 x と half 関数を、VF モード ON 時に使用するために書き直してみましょう。内容は以下のようになります。

- ライブラリ -

```
import Math ;

// x という名前で、1.0 という値の変数を定義（VF モード ON 用）
varfloat x = 1.0vf ;

// 値を 2 で割って返す関数（VF モード ON 用）
varfloat half( varfloat a ) {
  return a / 2.0vf ;
}
```

これらの変数と関数は、VF モード時に使用できます。

プログラム機能を使用する

ここまでは、いわゆる電卓ソフトとしての、普通の機能を扱ってきました。しかし、リニアンプロセッサーでは、そういった普通の機能よりも高度な「 プログラム 」というものを利用する事ができます。

ここでは、プログラムの仕組みと利用方法を解説します。

■「 プログラム 」とは「 する事のリスト 」

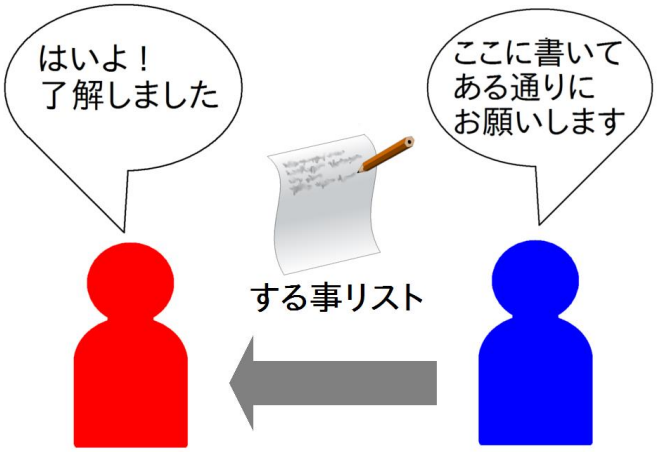
そもそも、「 プログラム 」とは一体何なのでしょうか。それは、「 する事のリスト 」です。

例えばあなたが、自分で電卓のボタンを押して複雑な計算を行う代わりに、他の人にやってもらう場合を考えてみてください。その場合、どういった数値に、どういった内容の計算をして、結果をどのようにまとめるのか、「 する事のリスト 」を紙に書いて渡す事でしょう。例えば、「 1 から 100 までの素数の和を求めよ 」といった計算の場合は、下記のようなリストになるでしょう。

▼ 人間が人間に処理を頼む時の、「 する事のリスト 」（※内容は 100 までの素数の和の計算）

・ここからの内容を、2 から 100 まで繰り返してください。(繰り返し中の回数を i とします)
　・もし i を、2 以上 i-1 以下の全ての数で割り、余りが一回も 0 にならないなら、それは素数です。
　・もし i が素数なら、i の値を合計に足してください。
・繰り返しはここまでです。
・最後に、合計の値を教えてください。

※ 実はこの場合には 2,3,5,7 で割った余りのみで素数か判定できるのですが、簡単のため上のようにしています。



これを受け取った人は、そこに書かれている通りに電卓を叩いて、計算を行ってくれるわけです。実は、**上のような「 する事のリスト 」が、役割的にはまさに一種のプログラム**と見なせるのです。

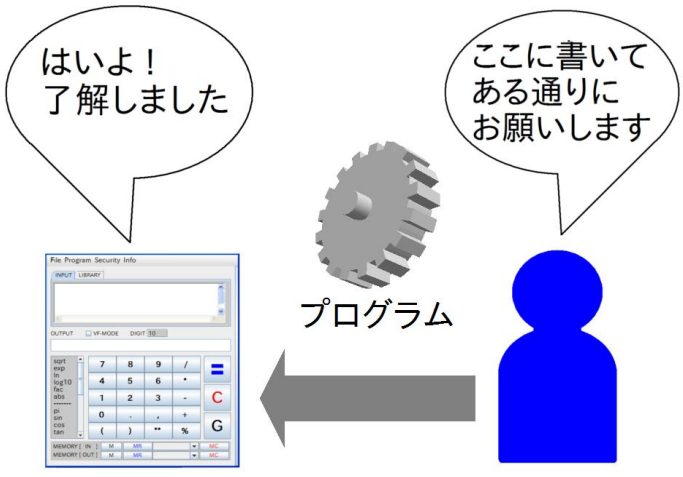
ただし、上のような普通の文章は、人間が人間に伝える場合には良いのですが、機械に伝える場合には、言い回しなどが不必要に複雑だったり、曖昧だったりして、理解するのが難しいという問題があります。よく「 日本語は難しい 」と言われますが、ただでさえ人間にとっても難しいのに、機械にとっては難し過ぎるのです。そこで、機械やソフトウェアに「 する事のリスト 」を伝えたい場合は、**「 プログラミング言語 」と呼ばれる特別な言語で記述**します。プログラミング言語は、人間の話す言語（自然言語）に特有の曖昧さや複雑さを排除した、機械にとって都合の良い言語です※。

※ あくまで「自然言語に比べて」という意味であって、プログラミング言語の中でも機械寄りのもの（機械語、アセンブリ言語）から、比較的人間寄りのもの（高級言語、スクリプト言語）まで、色々なものがあります。

先ほど例に挙げた、人間用の「 するの事リスト 」を、ちゃんとプログラミング言語で書き直すと下記のようになります。これが、一般に「 プログラム 」と呼ばれるものです。

▼ 人間が機械に処理を頼む時の、「 する事のリスト 」 = プログラム

```
int goukei = 0;
for( int i = 2; i <= 100; i++ ){
  bool sosuu = true;
  for( int j = 2; j <= i-1; j++ ){
    if( i % j == 0 ){
      sosuu = false;
    }
  }
  if( sosuu ){
    goukei += i;
  }
}
output( goukei );
```



このように、リニアンプロセッサーは、**「 する事のリスト 」＝ プログラム** を読んで、複雑な計算を自動で行う事ができます。このように、処理内容をプログラムにまとめる事で、次のような利点があります。

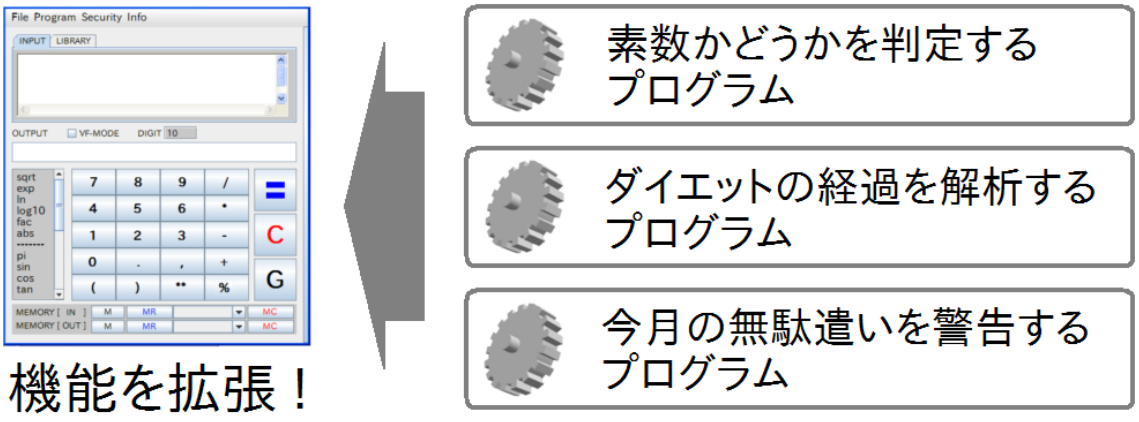
▼プログラムの利点

利点 1. 一手ごとに人間の操作や入力を待つ必要がないので、処理速度がはるかに速い
利点 2. いろいろな処理を複雑に組み合わせて、より高度な処理を実現できる
利点 3. 一度プログラムとして書いた処理は、何度でもすぐに使用できる

利点１に関してはこの通りで、現在のリニアンプロセッサーでは、**最大で毎秒１億回程度の、小数点付き数値（浮動小数点数）の演算処理**が可能です。一方で、「＋」や「－」などのボタンを毎秒 1 億回押して計算させるのは絶対に無理ですよね。プログラムなら、それができます。また、数値の入力についても、**ファイルに記述された膨大なデータを読み込んで、自動で次々と処理**していく事なども可能です。これも、人間が手でいちいち数値を大量に打ち込むのと比べて、はるかに高速です。

続いて利点 2 ですが、プログラムは文章として記述するという形態上、**ボタンの直接打ち込みに比べて、はるかに長く複雑な処理内容を、正確に伝える**事ができます。また、ボタンやウィンドウといった GUI 部品も使う事ができます。なので、頑張って数百行から数千行規模のプログラムを書けば、もはや一つのソフトウェアと呼べるような、高度なものでも作る事ができます。

最後に利点 3 です。**一度頑張ってプログラムに書いておいた処理は、それ以降は何度でもすぐに使えます。**これは一般のプログラムについても言える非常に大きな利点ですが、リニアンプロセッサーでは特に重要です。というのもリニアンプロセッサーでは、作ったプログラムを、「 プログラム 」メニューに追加して常用する事ができるからです（方法も、書いたプログラムを「 RinearnProcessorProgram 」フォルダ内に置くだけです）。これはつまり、**機能をユーザーが自由自在に拡張していける**事を意味していて、リニアンプロセッサーの最大の特徴になっています。



■ リニアンプロセッサーでは、プログラミング言語「 VCSSL 」をサポート

リニアンプロセッサーでは、C 言語系のプログラミング言語である「 **VCSSL** 」を標準でサポートしており、この言語でプログラムを書いたり、実行したりする事ができます。

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

実はこの VCSSL は、もともとリニアンプロセッサー上でのプログラム機能のために開発された言語です。そのため、C 言語系のオーソドックスな文法を採用しつつも、電卓上でちょっとしたプログラムを手軽に書いて動かせるように、即席・簡易用途重視のシンプルな言語になっています。

■ 色々なプログラムを無料で配信している「 コードアーカイブ 」も ！

上記の VCSSL 公式サイトでは、VCSSL で開発された色々なプログラムを無料で配信している、「 **コードアーカイブ** 」も運営しています。

コードアーカイブ： https://www.vcssl.org/ja-jp/code/

コードアーカイブでは、サンプル的なコードだけではなく、ツールとして使えるものや、コンテンツとして楽しめるものなども含めて、それなりに多くのプログラムを配信しており、だいたい毎月数本程度のペースで新しいプログラムが追加されます。

コードアーカイブから入手したプログラムは VCSSL 製なので、もちろんリニアンプロセッサー上でも動かして使えます。気に入ったプログラムはリニアンプロセッサーのフォルダ内にある「 RinearnProcessorProgram 」フォルダに入れておくだけで、メニューバーの「 プログラム 」メニューからすぐに使えるため、コードアーカイブは**リニアンプロセッサーの拡張機能配信サービス**のような感覚でも利用する事ができます。実際、リニアンプロセッサーのプログラムメニューに標準で付属しているプログラムは、ほとんどがコードアーカイブで配信されているものです。

ぜひ気に入ったプログラムを入手してみてください ！

■ プログラムを書いて実行する

プログラムは、コードアーカイブから入手する以外に、もちろん自分で独自のものを作成する事もできます。ここでは実際に、プログラムを書いて実行してみましょう ！

・ 非常に単純なプログラムを書いて実行してみる

プログラムの作成・実行方法はとても簡単で、これまで通りに**リニアンプロセッサーの画面の「入力」欄に、数式の代わりにプログラムを書いて「 = 」ボタン（または F11 キー）を押すだけ**です。すると、書いたプログラムが実行されます。試しに以下のように書いて実行してみましょう：

- 入力 -

```
output ( "Hello, World !" ) ;
```

※ 入力が面倒な場合、コピーして入力欄に「貼り付け」してください。

VCSSLのプログラムは、個々の文（処理単位）の末尾に「 ；（セミコロン）」 記号で付きます。そのため、**このように「 ； 」記号を含む内容を入力欄に記述すると、リニアンプロセッサーはそれを数式ではなくプログラムとして解釈します。**

さて、「 = 」ボタン（または F11 キー）を押して実行すると、出力欄に以下のように表示されます：

- 出力 -

```
Hello, World !
```

このように、「出力」欄にメッセージが表示されましたね。上で書いたプログラムは、このようにメッセージを表示する内容だったのです。

・ 少し長いプログラムを書いて実行してみる

VCSSLプログラムの詳しい書き方については、後述する通り別のガイドに譲るとして、さすがに上の例では短すぎてプログラムという感じがしないので、もう少し長いだけ内容を書いてみましょう。

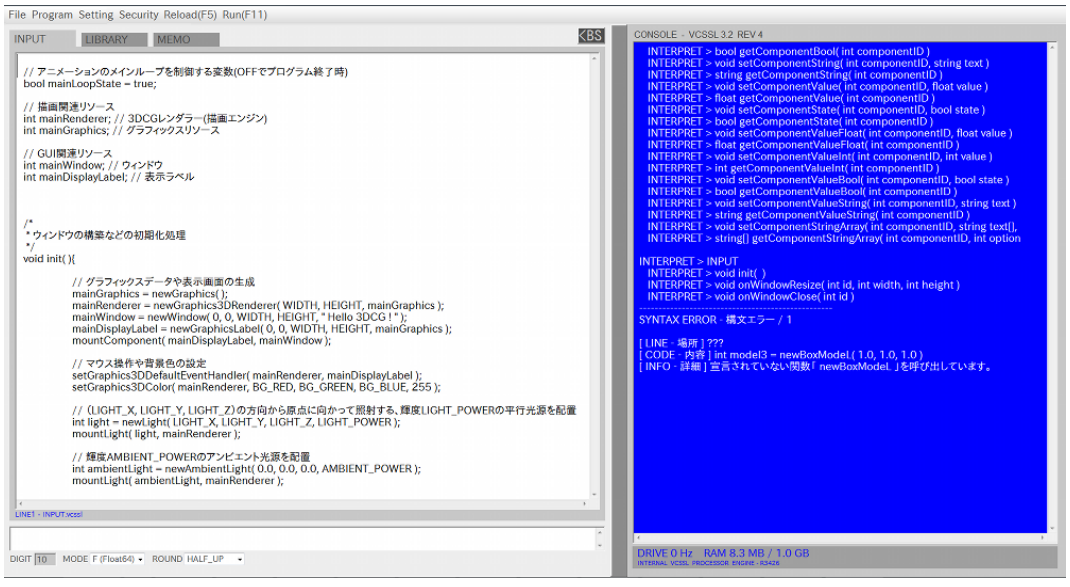
- 入力 -

```
import tool.Graph2D;

// グラフにプロットするサンプル座標値配列を用意
float xData[11];
float yData[11];
for(int i=0; i<=10; i++){
      xData[i] = i;
      yData[i] = xData[i] * xData[i];
}

// 2 次元グラフを起動してプロット
int graph = newGraph2D();
setGraph2DData(graph, xData, yData);
```

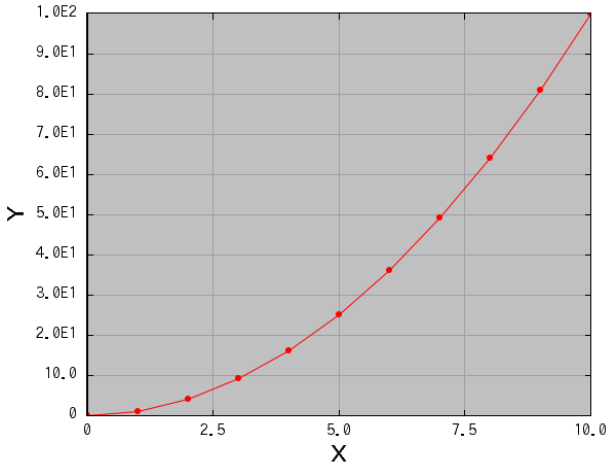
※ 入力が面倒な場合、コピーして入力欄に「貼り付け」してください。

※ このコードの詳細解説は：https://www.vcssl.org/ja-jp/code/archive/0001/2600-graph2d-array/

さて、上の内容を入力する際に、恐らく入力欄が狭すぎると感じたのではないでしょうか。その場合、まずは画面を最大化して、さらに中央付近にある水平のバーをマウスで掴んで、下の方まで引っ張り下げましょう。すると入力欄が広くなり、プログラミングに適したレイアウトに変形します：



ただし、このレイアウトにすると「 ＝ 」ボタンは隠れてしまって押せなくなるので、代わりにキーボードの「 F11 」キーか、メニューバーの「 実行（F11） 」を押してプログラムを実行します。実際に実行してみると：



上図のような 2 次元グラフが表示されます。実は上のプログラムは、数値データの列を「配列」という形で作成して、その内容を２次元グラフにプロット（描画）する内容だったのです。このように、プログラム内でデータの列を処理したり、プログラムからグラフソフトを操作する事などもできます。

■ プログラミングの詳しい解説は、同梱のガイドか VCSSL 公式サイトで

さて、実行方法については以上の通りですが、実際にゼロからプログラムを書くには、**「 色々な処理をどうプログラムに書くか 」**という、いわゆるプログラミングについての解説が必要ですね。

ここでそれを行うと、ガイドが非常に長くなってしまうため、リニアンプロセッサをダウンロード・展開したフォルダ内の「 VCSSL のプログラミングガイド 」というフォルダ内に、以下の通り別冊ガイドとして同梱されています。これらは VCSSL 公式サイトでも公開されており、ブラウザでも読めます。

■ **VCSSL スタートアップガイド（** https://www.vcssl.org/ja-jp/doc/start/ **）：**
プログラミングが初めての方に向けた、いわゆるプログラミング入門書に相当するガイドです。

■ **VCSSL 即席ガイド（** https://www.vcssl.org/ja-jp/doc/cprogrammer/ **）：**
C/C++等、既に C 系言語を扱われている方が、即席で VCSSL を扱うための簡易ガイドです。

■ **VCSSL リファレンスガイド（** https://www.vcssl.org/ja-jp/doc/ **以下 ）：**
VCSSL の文法や機能が網羅的を淡々ととまとめた、リファレンスマニュアル的なガイドです。

なお、標準ライブラリの一覧や仕様は「 https://www.vcssl.org/ja-jp/lib/ 」で参照できます。

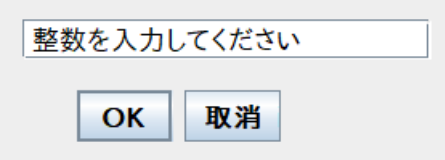
付属のプログラムの一覧と説明

ここでは、リニアンプロセッサーに標準で付属しているプログラムの一覧と、それらについての簡単な説明を記載します。各プログラムは「 プログラム 」メニューから実行できます。

■ 付属のプログラムの使い方

リニアンプロセッサーでは、「 RinearnProcessorProgram 」フォルダ内に入れておいたプログラムを、メニューバーの「 プログラム 」メニューからすぐに実行できます。そして、標準でも色々なプログラムが付属しています。はじめに、試しに 1 つ実行して使ってみましょう。

まずメニューバーから**「 プログラム 」メニュー**をクリックし、開かれるプログラム選択画面から、**「 整数・分数 」フォルダの中の「 素数判定.vcssl 」を選択**してください。すると以下のような入力画面が表示されます：



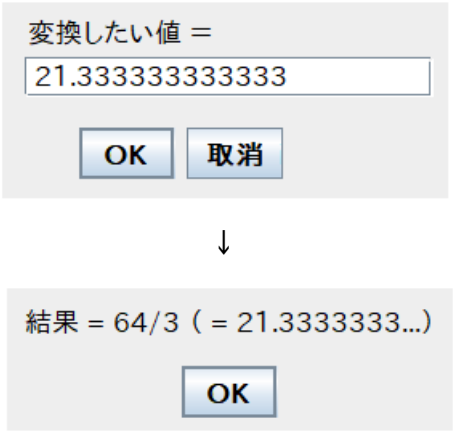
ここで適当に整数を入力すると、それが素数かどうかを判定して、答えてくれます。数値は何度でも入力できて、履歴は画面右の「 コンソール 」欄に表示されます（以下）：

```
7 が素数かどうか確認します...
7 は素数です。

87 が素数かどうか確認します...
3 で割り切れました。
87 は素数ではありません。
```

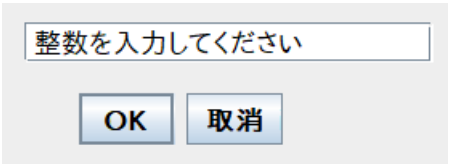
このように、ここで選んだプログラムは素数判定ツールとして使う事ができます。プログラム内容を見たり編集したい場合は、メニューバーから**「 ファイル 」 > 「 ファイルを開く 」** を選び、先ほどのプログラムを開いてください。すると「入力」欄にプログラムの内容が表示され、編集できます。編集しながら「 F11 」キーや「 = 」ボタンで実行可能で、「 ファイル 」メニューから保存もできます。

■「 整数・分数 」フォルダ内のプログラム

・少数から分数への近似変換

概要	少数を入力すると、近い値の分数を探して表示します。 
使用方法	実行すると、分数で近似したい小数（ここでは小数点付きの数値の意味、いわゆる実数）の入力を求められるので、入力してください。その後、分数を探す際の条件を色々と尋ねられるので答えると、その条件下で探した、近い値の分数が表示されます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/2900-float-to-fraction/

・素数判定

概要	入力された整数が、素数かどうか判定します。 
使用方法	実行すると整数の入力を求められるので、入力してください。すると、その整数が素数かどうかが判定されます。ただし扱えるのは、64bit 符号付き整数の範囲内です。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/0300-prime-test/

■ 「 微分・積分 」フォルダ内のプログラム

・微分

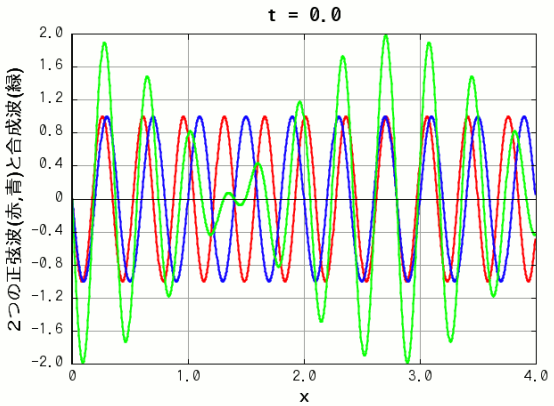
概要	入力された数式を数値的に微分します。
使用方法	起動後、「 f(x) = 」の欄に、微分したい数式を入力し、さらに微分値を求めたい地点の x の値などを入力した上で、「 CALC - 計算 」ボタンを押してください。すると、数値的に求めた微分値が表示され、微分したグラフなども表示されます。
詳細解説	- コードアーカイブ上では未公開 -

・積分

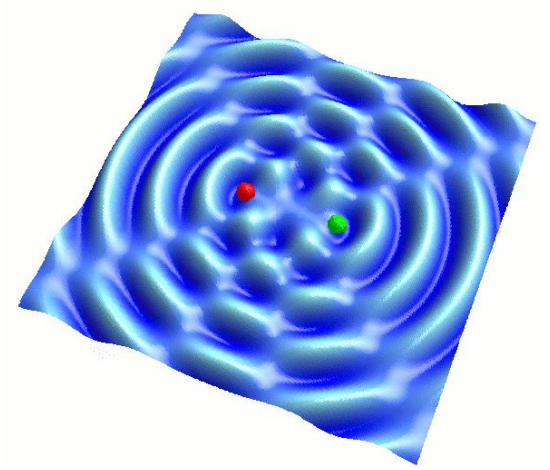
概要	入力された数式を数値的に積分します。
使用方法	起動後、「 f(x) = 」の欄に、積分したい数式を入力し、さらに積分区間などを入力した上で、「 CALC - 計算 」ボタンを押してください。すると、数値的に求めた積分値が表示され、積分したグラフなども表示されます。なお、数値積分アルゴリズムはシンプソン法/台形法/矩形法の 3 種から選択可能で、刻み数 N なども設定できます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/5600-integral-input/

■ 「 シミュレーション 」フォルダ内のプログラム

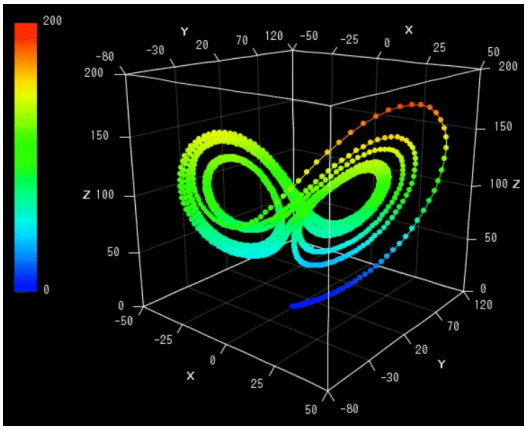
・正弦波の干渉

概要	2 つの正弦波（sin 波）を合成し、干渉する様子をシミュレーションします。 
使用方法	実行すると 2D グラフ画面が立ち上がり、2 つの正弦波が赤色と青色、それらの合成波が緑色でアニメーションで表示されます。画面上のスライダーを操作して、波の振幅や波長などのパラメータを調整できます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/3600-interfering-sine-wave/

・円形波の干渉

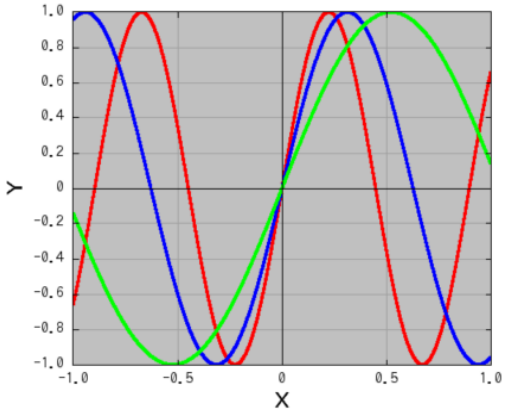
概要	2 つの円形波を合成し、干渉する様子を 3DCG でシミュレーションします。 
使用方法	実行するとウィンドウが立ち上がり、その上で 2 つの円形波が干渉する様子が 3DCG アニメーションで表示されます。画面上のスライダーなどを操作して、波の振幅や波長などのパラメータを調整できます。円形波を単体で表示する事も可能です。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/3800-interfering-circular-wave/

・ローレンツアトラクタ

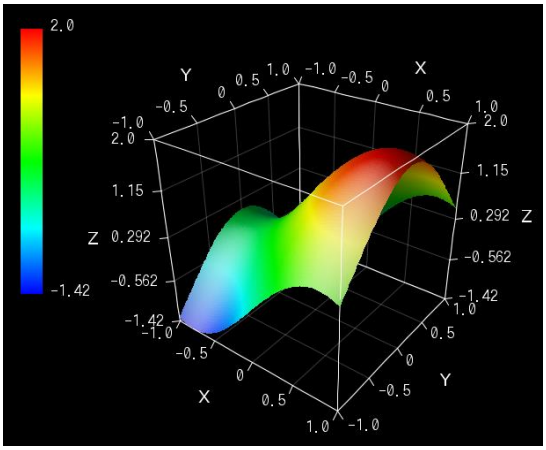
概要	ローレンツ方程式を 4 次ルンゲ＝クッタ法で解き、解曲線を 3D グラフで表示します。 
使用方法	実行すると 3D グラフ画面が立ち上がり、ローレンツ方程式の解曲線が表示されます。画面上のスライダーで、ローレンツ方程式のパラメータを操作すると、リアルタイムでグラフが変化します。解曲線を変数 t についてアニメーションする事も可能です。 このプログラムは実用目的のツールというよりは、微分方程式を GUI で操作しながら可視化するプログラムを作成するための土台として、ユーザーが書き換えて使う事などを想定して同梱されています。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/0100-lorenz-attractor/

■「 数式からのグラフ描画 」フォルダ内のプログラム

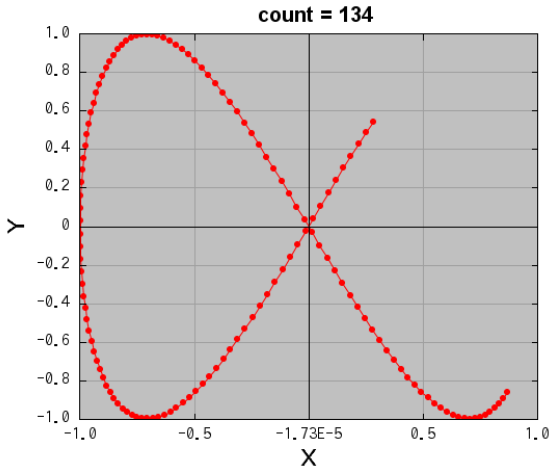
・y(x)形式の 2D グラフ描画

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/

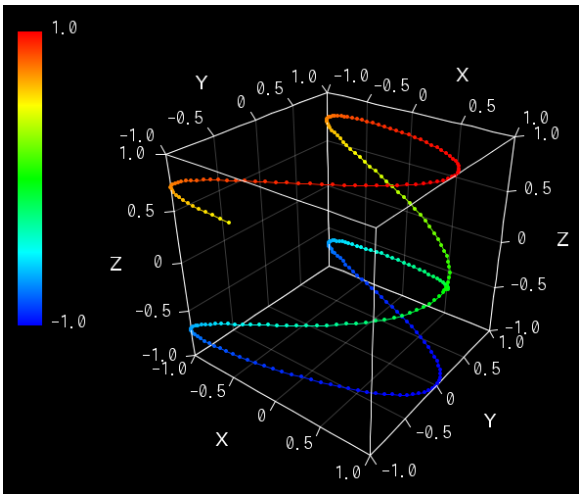
・z(x,y)形式の 3D グラフ描画

概要	z(x,y)形式の数式を、3D の曲面/メッシュグラフとして描画します。 
使用方法	起動後、入力画面の「 z(x,y) =」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 3D 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/0001/5100-graph3d-input-zxy/

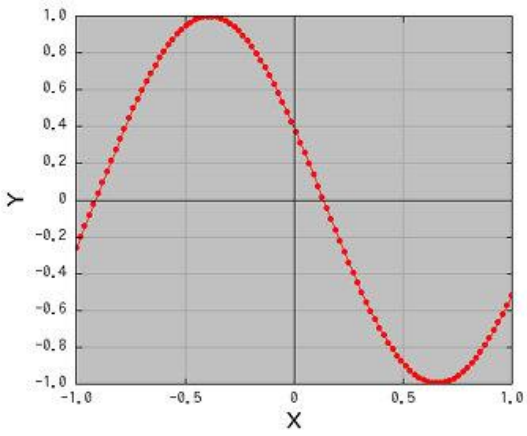
・x(t),y(t)形式の 2D グラフ描画

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/

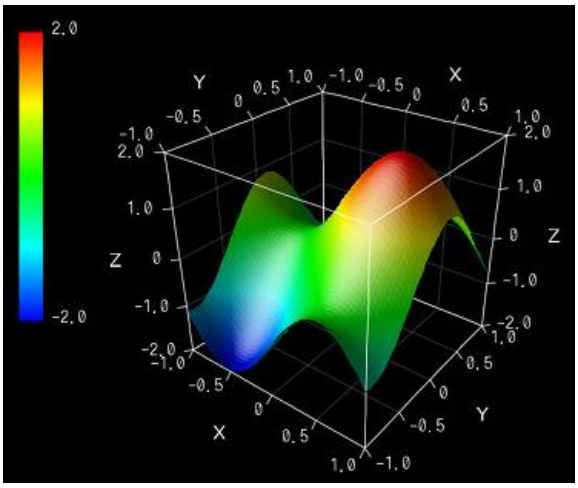
・x(t),y(t),z(t)形式の 3D グラフ描画

概要	x(t),y(t),z(t)形式の数式を、t を媒介変数として 3D 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) = 」、「 y(t) = 」および「 z(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 3D 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/0001/5300-graph3d-input-xt-yt-zt/

・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。  （時間と共に変形します。）
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/

・z(x,y,t)形式の 3D グラフ描画

概要	z(x,y,t)形式の数式を、3D の曲面/メッシュグラフとしてアニメーション描画します。  （時間と共に変形します。）
使用方法	起動後、入力画面の「 z(x,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 3D 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/0001/5500-graph3d-input-zxyt/

■「 座標値ファイルからのグラフ描画 」フォルダ内のプログラム

・座標値ファイルからの 2D グラフ描画

概要	ファイルからデータを読み込んで、2D グラフに描画します。
使用方法	実行するとファイル選択画面が表示されるので、グラフ化したいデータが記載されたファイルを選択すると、グラフ画面が立ち上がって描画されます。なお、ファイルの書式は「 https://www.rinearn.com/ja-jp/graph2d/guide/file 」をご参照ください。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/2400-graph2d-file/

・座標値ファイルからの 3D グラフ描画

概要	ファイルからデータを読み込んで、3D グラフに描画します。
使用方法	実行するとファイル選択画面が表示されるので、グラフ化したいデータが記載されたファイルを選択すると、グラフ画面が立ち上がって描画されます。なお、ファイルの書式は「 https://www.rinearn.com/ja-jp/graph3d/guide/file 」をご参照ください。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/2500-graph3d-file/

・連番の座標値ファイルからの 2D グラフアニメーション

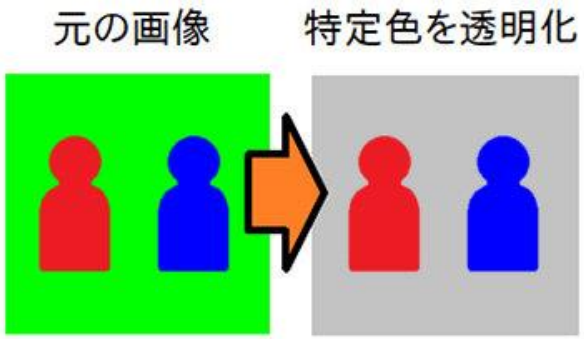
概要	連番のファイルから連続でデータを読み込み、2D グラフをアニメーション描画します。
使用方法	実行するとフォルダ選択画面が表示されるので、グラフ化したい連番データがあるフォルダを選択します。続いて、ファイル名（連番部分を除く）と拡張子を指定すると、グラフ画面が起動します。「PLAY」ボタンで再生開始、「STOP」ボタンで停止します。 なお、連番の各ファイル（＝アニメーションの各瞬間のファイル）の書式は「 https://www.rinearn.com/ja-jp/graph2d/guide/file 」をご参照ください。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/7200-graph-file-animator-2d/

・連番の座標値ファイルからの 3D グラフアニメーション

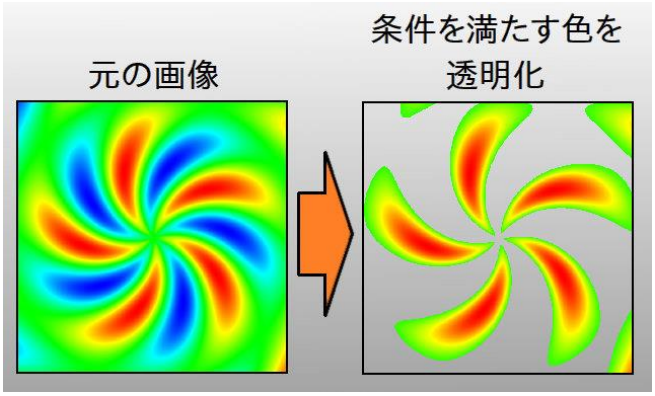
概要	連番のファイルから連続でデータを読み込み、3D グラフをアニメーション描画します。
使用方法	実行するとフォルダ選択画面が表示されるので、グラフ化したい連番データがあるフォルダを選択します。続いて、ファイル名（連番部分を除く）と拡張子を指定すると、グラフ画面が起動します。「PLAY」ボタンで再生開始、「STOP」ボタンで停止します。 なお、連番の各ファイル（＝アニメーションの各瞬間のファイル）の書式は「 https://www.rinearn.com/ja-jp/graph3d/guide/file 」をご参照ください。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/7300-graph-file-animator-3d/

■「 画像処理・グラフィックス 」フォルダ内のプログラム

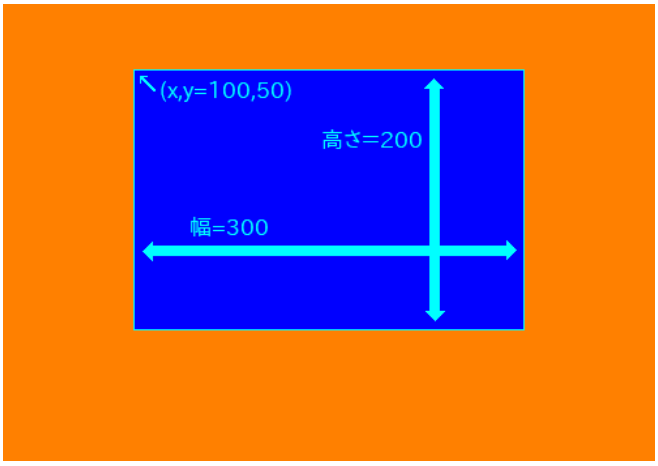
・画像内の特定の色を透明化

概要	画像ファイルを開き、RGB 値で指定した特定の色を透明で置き換えて保存します。 
使用方法	実行するとファイル選択画面が表示されるので、PNG または JEPG 形式の画像ファイルを選択してください。その後、色の RGB（赤/緑/青）値を尋ねられるので入力すると、その色を透明で置き換えた画像が、「 元の画像ファイル名_clear.png 」というファイル名で保存されます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/3300-color-to-clear/

・画像内の条件を満たす色を透明化

概要	画像ファイルを開き、RGB 値の条件式で指定した色を透明で置き換えて保存します。 
使用方法	実行するとファイル選択画面が表示されるので、PNG または JEPG 形式の画像ファイルを選択してください。その後、色の RGB（赤/緑/青）値に関する条件式を尋ねられるので入力すると、その条件を満たす色を透明で置き換えた画像が、「 元の画像ファイル名_clear.png 」というファイル名で保存されます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/3400-conditioned-color-to-clear/

・画像内の四角形領域を切り抜く

概要	画像ファイルを開き、指定された四角形（矩形）領域を切り抜いて保存します。 
使用方法	実行するとファイル選択画面が表示されるので、PNG または JEPG 形式の画像ファイルを選択してください。その後、切り抜く領域の位置（x,y）やサイズ（幅,高さ）を尋ねられるので入力すると、その領域内を切り抜いた画像が、「 元の画像ファイル名_crop.png（または.jpg） 」というファイル名で保存されます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/5700-image-cropper/

・画像を任意サイズに拡大・縮小

概要	画像ファイルを開き、指定されたサイズに拡大・縮小して保存します。 
使用方法	実行するとファイル選択画面が表示されるので、PNG または JEPG 形式の画像ファイルを選択してください。その後、拡大・縮小後のサイズ（幅,高さ）を尋ねられるので入力すると、そのサイズに拡大・縮小された画像が、「 元の画像ファイル名_scale.png（または.jpg） 」として保存されます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/5900-image-scaler/

・画像内の特定の色を透明化（複数ファイル一括処理版）

概要	「画像内の特定の色を透明化」の処理を、フォルダ内の全画像に対して一括で行います。
使用方法	基本的には「画像内の特定の色を透明化」と同様です。途中で入力先/出力先フォルダを尋ねられるので、選択してください。すると、入力先フォルダ内の全画像が処理されて、結果は出力先フォルダ内に保存されます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/3900-color-to-clear-multi/

・画像内の条件を満たす色を透明化（複数ファイル一括処理版）

概要	「画像内の条件を満たす色を透明化」の処理を、フォルダ内の全画像に対して一括で行います。
使用方法	基本的には「画像内の条件を満たす色を透明化」と同様です。途中で入力先/出力先フォルダを尋ねられるので、選択してください。すると、入力先フォルダ内の全画像が処理されて、結果は出力先フォルダ内に保存されます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/4000-conditioned-color-to-clear-multi/

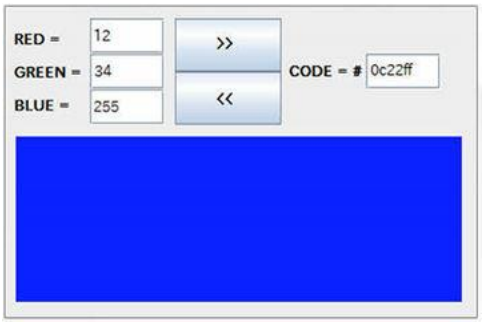
・画像内の四角形領域を切り抜く（複数ファイル一括処理版）

概要	「画像内の四角形領域を切り抜く」の処理を、フォルダ内の全画像に対して一括で行います。
使用方法	基本的には「画像内の四角形領域を切り抜く」と同様です。途中で入力先/出力先フォルダを尋ねられるので、選択してください。すると、入力先フォルダ内の全画像が処理されて、結果は出力先フォルダ内に保存されます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/5800-image-cropper-multi/

・画像を任意サイズに拡大・縮小（複数ファイル一括処理版）

概要	「画像を任意サイズに拡大・縮小」の処理を、フォルダ内の全画像に対して一括で行います。
使用方法	基本的には「画像を任意サイズに拡大・縮小」と同様です。途中で入力先/出力先フォルダを尋ねられるので、選択してください。すると、入力先フォルダ内の全画像が処理されて、結果は出力先フォルダ内に保存されます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/5900-image-scaler/

・RGB やカラーコードの色表示と相互変換

概要	8bit 深度の RGB 値と、16 進数のカラーコードを相互変換し、色の表示も行います。 
使用方法	実行すると、入力項目などが並ぶウィンドウが表示されます。RGB→カラーコードの変換では、画面左に RGB 値を入力して「 >> 」ボタンを押してください。カラーコード→RGB の変換では、画面右にカラーコードを入力して「<<」ボタンを押してください。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/1000-color-display/

・連番画像をアニメーション表示

概要	動画プレイヤーのような GUI で、連番の画像ファイルをアニメーション表示します。
使用方法	起動後、画像を読み込むフォルダを指定してください。続いて、ファイル名（番号部を除く）と拡張子を指定していしてください。すると、アニメーションが始まります。「PLAY/STOP」ボタンで再生/停止の切り替え、スライダーで時間操作ができます。
詳細解説	https://www.vcssl.org/ja-jp/code/archive/0001/4100-image-file-animator/

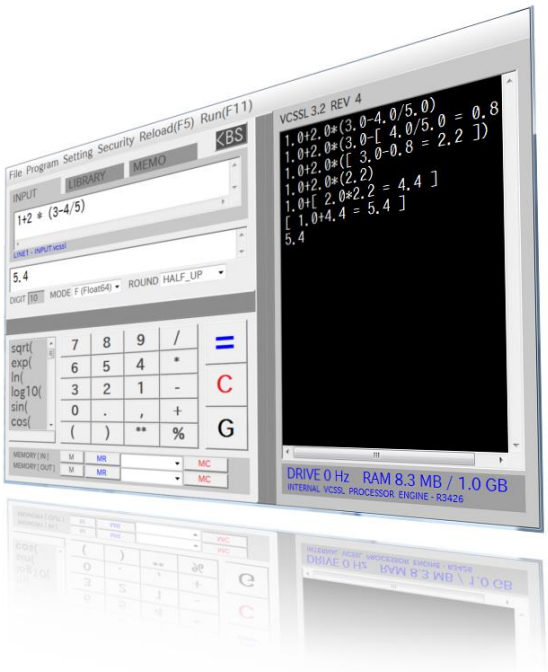
商標などに関して

[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 Processor 4.3　取扱説明書　第 6 版

著者　　松井文宏

このガイド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/processor/