PSIM®

Version 2020a

For Power Electronics & Motor control

スクリプト機能マニュアル

Powersim Inc.

Mywayプラス株式会社

Script Functions User's Guide

Version 2020a

February 2021

Disclaimer

Powersim Inc. (Powersim) and Myway Plus Corporation (Myway) make no representation or warranty with respect to the adequacy or accuracy of this documentation or the software which it describes. In no event will Powersim and Myway or their direct or indirect supplies be liable for any damages whatsoever including, but not limited to, direct, indirect, incidental, or consequential damages of any character including, without limitation, loss of business profits, data, business information, or any and all other commercial damages or losses, or for any damages in excess of the list price for the license to the software and documentation.

お問い合わせ先

Myway プラス株式会社

〒222-0022 神奈川県横浜市西区花咲町 6-145　横浜花咲ビル

Tel 045-548-8836, Fax 045-548-8832

Email: sales@myway.co.jp

URL: https://www.myway.co.jp/

目次

1 はじめに

PSIM は様々なスクリプト機能があり、スクリプトツールによって計算の実行、実行シミュレーション、グラフ表示を行うことができます。スクリプトツールは、PSIM の[スクリプト]メニュー から呼び出すことができます。

スクリプトツールは、以下の機能をサポートしています。

- 算術演算子
- 演算機能
- 数式機能
- 制御機能
- 関数配列
- 文字列関数
- 複素数
- ファイル関数
- グラフ機能
- シミュレーション機能

このチュートリアルでは、関数定義と使用方法について説明します。

各スクリプト関数にはヘルプページがあります。関数のヘルプページに移動するには、関数をハイライトし、**F1** キーを押すか、[**ヘルプ**]メニューから[**ヘルプ**]を選択します。

以下がスクリプトの例です。降圧コンバータの電流ループ設計になります。詳細につきましては、別紙の「Tutorial – Control Loop Design Using Script Functions.pdf」をご覧ください。

```
// Buck converter - current loop design
PI = 3.14159;

// Parameters
Vin = 250;
L1 = 200u;
C1 = 245u;
R1 = 0.6;
Ksen_i = 1/165; // current sensor gain
Ksen_v = 1/100; // voltage sensor gain
// PWM gain: Kpwm = Vin/Vcarrier, with Vcarrier=1
Kpwm = Vin;
fsw = 20k;
// Current PI controller
Ki_pi = 1.24292;
Ti_pi = 142.139u;
// Plant: Hi(s) = iL / Vos
Hi = formula(1/(s*L1 + R1/(s*R1*C1+1))); // define Hi(s) as a formula
// Current PI: Gi = Ki * (1+sTi) / (sTi)
Gi = formula(Ki_pi*(1+s*Ti_pi)/(s*Ti_pi)) // define Gi(s) as a formula
// Current loop transfer function Ti_loop
```

```
Ti_loop = Hi*Gi*Ksen_i*Kpwm // perform formula calculation
// Bode plot
wmin = 100 *2*PI;
wmax = 50e3 *2*PI;
Freq_rad = ArrayLog(wmin, wmax); // define frequency array (rad/sec) from wmin to
wmax, in log
Freq_Hz = Freq_rad/(2*PI); // define frequency array (Hz) from fmin to fmax
s = Complex(0, Freq_rad); // define Laplace operator

BodePlot("Current Loop", Freq_Hz, Hi, Gi, Ti_loop); // generate Bode plots for Hi, Gi, and
Ti_loop
```

スクリプトでは、ダブルスラッシュ“ // “ は、コメントを表します。スクリプトステートメントの各行はセミコロン” ; “で終わることを推奨します。これによりスクリプトが読みやすくなり、誤りが発生する可能性を低減できます。

2 データタイプ

スクリプトで使えるデータタイプは :

Float、integer、complex、string、array　です。

データタイプの宣言は必要ありません。変数が割り当てられるとデータタイプは自動的に決まります。同じ変数は別の割り当てを通じてデータタイプを変更することができます。

以下がデータタイプ割り当てのスクリプト例となります。

```
// This script shows how variable data types are assigned.

i1 = 5; // i1 is an integer

r1 = 4.6; // r1 is a float

c1 = complex(1.1,2.2); // c1 is a complex number

s1 = “test1”; // s1 is a string

s2 = s1+”test2”; // s2 is also a string

s1 = c1; // s1 is now a complex number

a1 = {1, 2, 3, 5, 7}; // a1 is an integer array. a1 = {1,2,3,5,7}

a2 = array(1, 5, 1); // a2 is an integer array. a2 = {1,2,3,4,5}

a3 = array(complex(2,-1), complex(10,-5), complex(2,-1)); // a3 is a complex array.

// a3 = {2-j1, 4-j2, 6-j3, 8-j4, 10-j5}

Months = {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec} // Months is a
string array
```

スクリプト機能名は大文字小文字の区別はありません。変数名は大文字小文字の区別をします。

スクリプト機能を次の節で説明します。

3 スクリプト実行画面

デフォルトでは、計算の各ステップの変数値が表示ウィンドウに表示されます。表示ウィンドウをクリアするには、[**編集**]メニューから次を選択します。

[**メッセージウィンドウをクリア**]：メッセージウィンドウをクリアします。

[**毎回実行前にメッセージウィンドウをクリア**]：毎回実行前にメッセージウィンドウを自動的にクリアします。

各ステップで変数値を表示すると、スクリプトの実行が遅くなります。関数 *ScriptOption* は、表示を無効または有効にします。

ScriptOption(“Nolog”) 変数値の表示が無効になります

ScriptOption(“log”) 変数値の表示が有効になります。

次の例は、関数の使用方法を示しています。

```
a = 1;
ScriptOption(“Nolog”);        // disable display
b = 2;
ScriptOption(“log”);           // enable display

c = a + b;
```

スクリプト画面は以下のように表示します：

```
a = 1

c = 3
```

b の値は表示されません。

4 算術演算子

スクリプトツールでサポートされる算術演算子は以下の通りです。

+	加算
-	減算
*	乗算
/	徐算
%	剰余算（例:　5%2 = 1）
^	乗数 (x^y は x の y 乗を返します)
**	乗数 (x**y は x の y 乗を返します)
=	等号
==	イコール
!=	ノットイコール
>	大なり
>=	大なりイコール
<	小なり
<=	小なりイコール
!	否定演算子
&&	論理積（AND）
||	論理和（OR）

5 演算機能

スクリプトツールでは以下の演算機能がサポートされています。

abs(x)	絶対値
acos(x)	arccos (ラジアン)
acosd(x)	arccos (摂氏)
arccos(x)	arccos (ラジアン)
asin(x)	arcsin (ラジアン)
asind(x)	arcsin (摂氏)
arcsin(x)	arcsin (ラジアン)
atan(x)	arctan (ラジアン)
atand(x)	arctan (摂氏)
arctan(x)	arctan (ラジアン)
atan2(y,x)	arctan (x と y が定義、単位はラジアン）)
atan2d(y,x)	arctan (x と y が定義、単位は摂氏）)
ceil(x)	実数 x より大きい整数を返す関数
cos(x)	cos (x 単位はラジアン)
cosd(x)	cos (x 単位は摂氏)
cosh(x)	双曲線 cos
eval	数式評価
exp(x)	e（自然対数の底数）の x 乗
FFT	FFT 解析

floor(x)	床関数 実数 x に対して x 以下の最大の整数を出力
formula	変数の関数として数式を定義
hypot(x1,x2,x3,...)	x1 の 2 乗、x2 の 2 乗、x3 の 2 乗の和の平方根 (sqrt(x1^2+ x2^2+ x3^2…..)
hypot(x_array)	入力がアレイになります。アレイセルの 2 乗和の平方根の値を返します。(sqrt(x_array[0]^2+x_array[1]^2+x_array[2]^2…..)
ln(x)	底が e である自然対数
log(x)	底が e である自然対数
log10(x)	底が 10 である常用対数
max(x1,x2,x3,…..)	x1,x2,x3,…の最大値。(入力データ数は制限なし)
max(x_array)	入力がアレイでアレイセルの最大値を返します。
min(x1,x2,x3,…..)	x1,x2,x3,…の最小値。(入力データ数は制限なし)
min(x_array)	入力がアレイでアレイセルの最小値を返します。
mod(x,y)	x/y リマインダを返すモジュロ関数。これは剰余算と同じです。(mod(5, 2) = 1)
pow(x,y)	x の y 乗
pwr(x,y)	x の y 乗の絶対値、abs(x)^y y
sign(x)	符号関数 1(x > 0); -1(x < 0); 0(x = 0)
sin(x)	sine (x 単位はラジアン)
sinh(x)	双曲線 sine
sqr(x)	x の 2 乗 x^2
sqrt(x)	平方根
tan(x)	tan(x 単位はラジアン)
tanh(x)	双曲線 tan

6 FFT 解析

FFT 関数は FFT 解析を実行します。以下に示すように、FFT 関数を使用するには 2 つの方法があります。

FFT(arr, Xmin, Xmax)
FFT(array_y, array_x, Xmin, Xmax)

FFT(arr、Xmin、Xmax)の場合、arr はシミュレーション結果の SIMVIEW のデータを使います。これはデータの最初の列は時間であると想定されています。

FFT(array_y、array_x、Xmin、Xmax)の場合、array_y は FFT 解析のためのデータ配列、array_x は時間データとなります。

Xmin と Xmax は、FFT 解析に使用されるデータの開始時間と終了時間を定義します。これらは省略が可能で、定義されていない場合は、データ全体が使用されます。

Xmax-Xmin として定義されるデータ範囲は、基本周期の整数倍でなければなりません。 たとえば、基本周波数が 1kHz の場合、データ範囲は 1m、2m、3m などにする必要があります。

次の例は､シミュレーション結果の SimView データを使って FFT 解析をする場合の設定例です。

```
file_smv = GetLocal("PARAMPATH") + "buck converter.smv";

 // Read the SIMVIEW file into the object arr.
arr = GraphRead(file_smv);

// Perform FFT on the curve Vos. It is assumed that the first column of the data is time.
Only the data from 0.4ms to 0.5ms is used for FFT.
fft_result = FFT(arr["Vos"], 0.4m, 0.5m);
```

次の例は、時間とデータ配列を別々に定義した場合の設定例です。

```
// Define the time array and data array
array_x = Array(0, 6*pi, pi/1000);
 array_y = sin(array_x);

 // Perform FFT on array_y
fft_result = FFT(array_y, array_x, 2*pi, 4*pi);
plot(fft_result);
```

7 数式機能

数式機能は式を定義し計算します｡スクリプトツールの数式機能では以下がサポートされています。

Formula(math expression)

この機能は変数の関数として式を定義します。変数値は定義されていません。

```
f1 = Formula(a+b*c); // f1=a+b*c    ここで a, b, c は未知の変数です。
```

Eval(formula name, variable1=value1, variable2=value2, …)

この機能は変数値を元に式の値を計算します。例えば

```
f1 = Formula(a+b*c); // f1 = a+b*c  a, b, cは変数です。

f1_value = Eval(f1, a=1, b=2, c=3); //  f1a, b, c の値を定義して f1 を計算します。
```

この機能で加減乗除演算ができます。

以下が数式機能を使ったスクリプトの例です。

```
H = formula(R / ((s * R * C) + 1)); // 伝達関数 H(s)を定義しています
                                    //  R, Cは変数です。

G = formula(kpi *((s*Tpi)+1) / (s*Tpi)); // 伝達関数G(s)の定義

T = G*H; // 数式同士の積    G(s)*H(s)

Tcl = T/(1+T); // 数式演算の実行
```

8 制御/エラー/警告機能

スクリプトツールでサポートされる制御/エラー/警告機能以下の通りです。

if (...)
{...}
else if (...)
{...}
else
{...}
	条件付き if 文

iif (condition, value1, value2)	インライン if 宣言 (注：“iif”が正しく“if”ではありません)
While (...) {...}	While ループ
Error (“text %f, %f”, var1, var2)	エラー宣言。最大 5 つの変数がサポートされます。
Warning (“text %f, %f”, var1, var2)	警告宣言。最大 5 つの変数がサポートされます。
Return(var1)	var1 の値を戻します。

例:

if と while を使用した構文の例を以下に示します。

```
if (k1 > 10)
{
        a1 = 10
}
else
{
        a1 = 5
}

iflag = 1
b1 = 0
while (iflag == 1)
{
        b1 = b1 + 0.1
        if (b1 > 10)
                iflag = 0
}
```

9 関数配列

配列は、中括弧のペア({})で定義されカンマ(,)で項目を別々に分割します 。以下にいくつかの例を示します。

```
Months = {"Jan", "Feb" , "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",  "Oct", "Nov", "Dec"};

a = {1, 2, 3, 5, 7, 11};

b = {10.5, 8.3, 33.59};
```

配列内のセルの値は 大括弧([])を 使用してアクセスします。

例えば、上記の例では、Months[0]の値は“Jan”、a[1] の値は 2、b[1]の値 は 8.3 になります。

同様にセルの値を変更することも可能です。例えば、a[1] = 5;

多次元アレイは、括弧の内側に括弧を入れ子にすることで定義できます。例えば。

```
M = {{1,2}, {3,4}, {5,6}};
```

この場合、M[0][1]の値は 2 になります。値を 50 に変更するには次のようにします。M[0][1] = 50;

```
a = {1, 2, 3, 4, 5};        // 元の配列

a[1] = 10;                  // a[1]を 10 に変更します。

c = a;                      // 新しい配列が作成されます。C への変更は a には影響しません。

c[2] = 50;                  // a[2]は影響を受けません。
```

サポートされている配列関数の機能について。

Array(size)
Array(size, InitialValue)
Array(FirstValue, LastValue, increment)

配列関数は配列の一次元配列のサイズを定義します。‘InitialValue’を指定すると、すべてのセルの値を InitialValue'に初期化することができます。'FirstValue'の初期値で始まり、それが 'LastValue'の最終値に達するまで、'increment'の増分幅で配列を初期化することも出来ます。

例えば、次の例では 10 個のセルの配列を定義し、while ループでそれらを初期化します。結果の配列は次のとおりです。

x1 = {15,17,19,21,23,25,27,29,31,33}

```
x1 = Array(10);
i = 0;
while(i < 10)
```

```
{
x1[i] = 15 + (2 * i);
i++;
}
```

別の例として、x2 = Array(15, 33, 2)は 10 セルの配列を定義し、Array 関数として初期化します。結果は次のとおりです。x2 = {15,17,19,21,23,25,27,29,31,33}

この関数は、減算することもできます。例えば、x = Array(9, 3, 1.5) の結果は次の通りです。
x = {9, 7.5, 6, 4.5, 3}.

次の例では、10 個のセルの配列を定義 し、行(“Row “)として初期化します。その後、while ループで数値を付加します。結果は次のとおりです。
L = {Row 1,Row 2,Row 3,Row 4,Row 5,Row 6,Row 7,Row 8,Row 9,Row 10}.

```
L = Array(10, "Row ");
i = 0;
while(i < 10)
{
L[i] = L[i] + string(i+1);
i++;
}
```

ArrayLog(FirstValue, LastValue)

この関数は 、1 次元配列を 定義 し、‘FirstValue’で始まり、‘LastValue’で終わる値で配列を初期化します。数値は指数関数的に増加します。

例えば、X = ArrayLog(8,300) のように関数を使用すると、
X={8,9,10,20,30,40,50,60,70,80,90,100,200,300}のように初期化されます。
Y = ArrayLog(10000, 500) のように関数を使用すると、
Y = {10000,9000,8000,7000,6000,5000,4000,3000,2000,1000,900,800,700,600,500}のように初期化されます。

SizeOf(Var1)

この関数は、‘Var1’が 1 次元配列の場合、配列のサイズを返します。‘Var1’'が文字列の場合、文字列のサイズを返します。‘Var1’が'曲線の場合は、ポイント数を返します。

例 :

```
Var1 = {1,2,3,4};

a = sizeof(Var1); // a = 4
```

Var1 が以下を含むグラフの場合：

Time	V1	V2	V3
0.01	0.11	0.12	0.13
0.02	0.21	0.22	0.23
0.03	0.31	0.32	0.33

その次に、

```
num_column = sizeof(Var1); //num_column=4

num_row = sizeof(Var1[0]); //num_row=3. Var1[0] is the 1st column. Var1[0] =
{0.01,0.02,0.03}
```

AddToArray(Var1, value1, ….)

この関数は、配列'Var1'にセルを追加します。'Var1'は必ず'配列でなければならない事に注意してください。以下の例では、3 つの値が配列 a に追加されます。

```
a = {1,2,3,4}

AddToArray(a, 5,6,7);   // a が{1,2,3,4,5,6,7}に変更されます
```

AppendArray(array1,array2,array3,・・・・)

この関数は､パラメータリストにあるすべてのアレイを表示と同じ順で追加してアレイを返します。パラメータリストにあるすべてのアレイは同じタイプである必要があります。

この関数はグラフにも使用できます。

次の例ではアレイarr2はarr1に追加されます。

```
arr1 = {1,2,3,4};
arr2 = {10, 11};
arr3 = AppendArray(arr1,arr2); //  arr3 is: {1,2,3,4,10,11}
```

Copy (arr)
Copy (arr , StartIndex, Length)

コピー機能は、配列の完全または部分的なコピーを行います。
パラメータは次のとおりです。

arr :	入力配列
StartIndex:	配列のコピーを開始するセルの位置
Length:	返される配列の長さ

Copy(arr)関数は、配列の完全なコピーを返します。Copy(arr, StartIndex, Length)関数は 'StartIndex'のセルの場所から、'Length'の長さ分の配列のコピーを返します。
'StartIndex'に-1 を入力した場合配列の最後から'Length'の長さの配列を返します'Length'に-1 を入力した場合'StartIndex'自身とその前にあるすべてのセルを返します。

配列演算

異なる大きさの二つの配列で演算を行った場合、結果は小さい方のアレイのサイズに変更されます。

以下に例を示します。

```
arr1 = {10, 100, 1000, 10000};
arr2 = {-1, -2};
arrA = arr1 + arr2;        //結果は{9,98}です。
arrB = arr1 - arr2;        //結果は{11,102}です。
arrC = arr1 * arr2;        //結果は{-10,-200}です。
arrD = arr1 / arr2;        //結果は{-10,-50}です。
arrE = arr2 / arr1;        //結果は{-0.1,-0.02}です。
arrF = log10(arr1);        //結果は{1,2,3,4}です。
arrG = 6 * arr2;           //結果は{-6,-12}です。
arrH = 6 / arr2;           //結果は {-6,-3}です
```

10 文字列関数

文字列は 2 つのダブルクォーテーション(")で挟んで定義します。 例えば"This is a string"のように定義します。以下に示すように文字列は、変数に代入することができます。

```
Var1 = "Apple";
Var2 = "Orange";
```

プラス記号(+)は文字列を結合するために使用します 。その他の算術演算(- * /)は文字列に対しては使用できません。例えば、

```
Var3 = Var1 + Var2; //  Var3 の内容は次のとおりです。 "AppleOrange"
```

_CRLF は、 復帰改行（キャリッジリターン）として特別な処理が行われます、文字列操作で使用することができます。

文字列に関連する機能は以下になります。

String(val)

この関数は、数値を文字列に変換します。例えば、

```
a = String(12.7m);              //  a は"0.0127"の文字列
b = String( 15 + 6 );           //  b は"21"の文字列
arr = {1, 5, 9};
c = String(arr);                //  c は"{1,5,9}"の文字列
```

Value(str)

この関数は、文字列を数値に変換します。

テキストファイルから数値を読み込むとき、通常は文字列として読み取られます。文字列として読み込まれた数値は、算術演算で使用される前に‘Value()’関数を使用して数値に変換する必要があります。例えば、

```
a = "12";
b = "5";
r  = a + b;                    // r は文字列"125"です。 文字列"12"と文字列"5"
                               // が連結され、文字列"125"になります。
s = Value(a) + Value(b);       // s は数値の 17 です。 "12"と"5"は、最初数値に変
                               // 換されその後算術演算が行われます。
```

Find(InputString, StringToFind, StartSearchIndex(Optional))

Findreverse(InputString, StringToFind, StartSearchIndex(optional))

Find 関数は入力文字列内の文字や文字列を検索します。‘Start search index’が指定されている場合は指定された場所の後ろから検索を開始します。文字や文字列を見つけることができなかった場合、関数はインデックスとして 0 を返すか-1 を返します。

Fundreverse 関数は、逆方向に検索することを除いては、Fund 関数と同じように機能します。

関数のパラメータは次のとおりです。

Input string:	入力文字列
String to find:	検索する文字列
Start search index:	0 を開始位置とする、文字れる検索を開始する位置

関数は、最初にマッチした文字列の 0 を開始位置とするインデックスを返します。文字列がマッチしなかった場合は-1 を返します。例は以下のとおりです。

```
Var3 = "AppleOrange"
iA = Find(Var3, "A");          // "AppleOrange"の中で'A'は最初の文字なので、
                               //  iA には 0 が入ります。
ia = Find(Var3, "a");          // "AppleOrange"の中で'a'は 8 番目の文字なので、
                               //  ia には 7 が入ります。
ie1 = Find(Var3, "e");         //   同様に ie1 は 4 です。
ie2 = Find(Var3, "e", ie1+1);  //   ie2 は 10 です。 関数は、最初の 'e'の後の文字か
                               //  ら検索を開始します。
```

SizeOf(str)

関数は文字列のサイズを返します。

SubStr (InputString, StartIndex, Length)

関数は入力文字列の一部を返します。

関数のパラメータは次のとおりです。

Input string	入力文字列
Start index:	ゼロから始まる文字の位置のインデックス
Length:	返される文字列の長さ

この関数は‘StartIndex’から始まる文字から‘Length’で定義された範囲の数分の文字のコピーを返します。

'StartIndex'が-1 の場合、文字列の最後の文字から'Length'分の範囲の文字を返します。
もし'Length'が-1 の場合、関数は'StartIndex'を含む最後の文字までの全ての文字を返します 。例は以下のとおりです。

```
Var3 = "AppleOrange";
s1 = SubStr(Var3, 3, 5);        // 返される文字列は"leOra"です。
s2 = SubStr(Var3, -1, 3);       //返される文字列は最後の文字列"nge"です。
s3 = SubStr(Var3, 3, -1);       // 返される文字列は先頭から 4 番目の文字と、
                                //その文字より後ろのすべての文字を含む
                               //文字列"leOrange"です。
```

Replace(Input string, part to replace, part to replace with, , part to replace, part to replace with, ...)

この関数は、文字列の一部を新しい文字列で置き換えます。

この関数は、複数の‘string to replace’ と‘string to replace with’の組み合わせを無限に指定することが可能です。これにより複数種類の文字列の置き換えが可能です。例えば"(2.0, 3.0)(4.5, 8.3)" を"2.0 3.0 4.5 8.3 "に変換するには以下のようにします。

```
str1 = "(2.0, 3.0)(4.5, 8.3)"
// まず、空の文字列に置き換えることにより、全ての入力かっこを削除します。
str2 = Replace(str1, "(", "");      // 全ての"("を削除
str2 = Replace(str2, ")", " ");     // ")" はスペースで置換
str2 = Replace(str2, ",", " ");     // "," はスペースで置換
str2 = Replace(str2, "  ", " ");    // 2 つ連続するスペースを 1 つのスペースに置換
str2 = Replace(str2, "  ", " ");    // すべてのスペースをスペース無しに置換
str3 = Replace(str1, "(", "", ")", " ", ",", " ", "  ", " ");
//  1 回の関数実行で上記すべての操作を行います。 str1 は変更されません。
```

Split(Input_string, Separator_characters)

この関数は、区切り文字を指定して、文字列を分割します。文字列内の文字のいずれかが検出されると、入力文字列が分割されます。戻り値は文字列の配列です。

例えば、

```
Split("22,33,44,55,66", ",");      //戻り値は次の文字配列です: {"22"," 33"," 44"," 55"," 66"}
```

Split2(Input_string, Start_string, End_string)

Split2(Input_string, Separator_string)

この関数は、文字列を分割します。戻り値は文字列の配列です。Split2 は、 セパレータとして(“Begin”)の文字列全体を使用します。一方 Split はセパレータとして(" \t\r\n")のような 文字のリストを使用します。

例えば、

Split2("(22, 33), (44, 55), (66 , 88)", "(", ")");　　//返り値は: {"22, 33"," 44, 55"," 66 , 88"}です。

11 複素数

次の複素数関数が用意されています。

Complex(a, b)

この関数は複素数を返します。‘a’は実部で‘b’は虚部です。複素数配列を作成する場合、各パラメータを配列で指定することができます。

```
c1 = complex(5, 3);                                  // c1 は:    5 + 3j
arr1 = complex(Array(0,6, 2), Array(-20, -14, 2)); // arr1 は:  {  0–20j, 6-14j, 2+2j }
w1 = complex(5, {1, -5, -7, 22});                    // arr1 は:  {  5+j, 5-5j, 5-7j, 5+22j }
```

Polar(mag, ang)

この関数は、複素数を返します。‘mag’は大きさで‘ang’は角度です。複素数配列を作成する場合、各パラメータを配列で指定することができます。例えば、

```
c1 = Polar(10, 1.4);                    // c1 は:    9.85 + 1.7j
arr1 = Polar ({10, 20, 30}, {0, 0.7, 1.4});
```

Real(c1)

この関数は複素数の実部を返します。

Imag(c1)

この関数は複素数の虚部を返します。

Abs(c1)

この関数は、sqrt(a^2 + b^2)のように定義される複素数の大きさを返します。

Angle(c1)

この関数は、複素数の角度を返します。角度は atan(b, a)のように定義されます。

12 ファイル関数

次のファイル関数が提供されます。

FileRead(FilePath)

この関数は、テキストファイルを読み込み、文字列として変数に取り込みます。例えば、

str = FileRead(“C:\Powersim\MyText.txt”); // str はファイルの内容を含む文字列です。

この関数は、SIMVIEW グラフファイルを読み取るためのものではありません。基本的なテキストファイルを読み取るために使用され、文字列関数を使用して情報を解析できます。たとえば、「MyText.txt」に次のコンテンツが含まれているとします。

```
// This is a parameter file
R1 = 1.2;
R2 = 2.3;
```

次のコードはファイルを読み取り、R1 の値を見つけます。

```
str1 = FileRead("C:\Powersim\MyText.txt"); // str1 contains strings of the entire file
index1 = Find(str1, "R1"); // Find the index of string “R1”
index2 = Find(str1, "=", index1); // Find the equal sign after index1
index3 = Find(str1, ";", index2); // Find the semicolon after index2
R1_value_str = SubStr(str1, index2+1, index3-index2-1); // Find the value of R1 in string
R1_value = Value(R1_value_str); // Convert the string to a number
```

FileReadLines(FilePath)

この関数はテキストファイルを文字列の配列として読み込みます。配列の各セルには、1 行のテキストが読み込まれます。例えば、

arr = FileReadLines(“C:\Powersim\MyText.txt”); // arr は文字列の配列です。

この関数は、SIMVIEW グラフファイルを読み取るためのものではありません。基本的なテキストファイルを読み取るために使用され、文字列関数を使用して情報を解析できます。たとえば、「MyText.txt」に次のコンテンツが含まれているとします。

```
// This is a parameter file
R1 = 1.2;
R2 = 2.3;
```

次のコードはファイルを読み取り、R1 と R2 の値を見つけます。

```
str1 = FileReadLines("C:\Powersim\MyText.txt"); // str1 is an array, and each cell
contains
// each line, for example, str1[0] = “// This is a parameter file”,
```

```
// and str1[1] “R1 = 1.2”
nline = sizeof(str1); // Find the size of the array str1 (number of rows)
variable_name = array(); // Define the variable name array. The dimension is
undefined.
variable_value = array(); // Define the variable value array

index = 0;
while (index < nline)
{

    str2 = str1[index]; // Place the cell content of str1 in str2
    index1 = Find(str2, "="); // Find the equal sign
    if (index1 > 0)
    {
name = SubStr(str2, 0, index1); // Get the variable name
AddToArray(variable_name, name); // Add the name to the array

variable_value_str = SubStr(str2, index1+1); // Get the variable value in string
val = Value(variable_value_str); // Convert the string to a number
AddToArray(variable_value, val); // Add the value to the array
    }
    index++;
}
a = variable_name; // Display the variable name array
b = variable_value; // Display the variable value array
```

FileWrite(FilePath, Object, "A" (optional))

この関数は新しいテキストファイルを作成し書き込みます。ファイルが既に存在する場合、上書きされ元の内容は削除されます。フォルダが存在しない場合は、フォルダを作成します。

Object が、文字列または数値の場合、新しいテキストファイルに書き込まれます。Object が配列である場合、各セルは別々の行として書き込まれます。Object がグラフである場合は、最初の行はグラフ名の後に 1 行に 1 つの数字が続きます。

オプションの"A"が指定された場合、ファイルは、ANSI 形式で記述さます。指定しない場合 Unicode 形式が使用されます。例えば、

```
arr = {   "[Diode]",
          "Forward Voltage = 1.2V",
          "Forward Current = 30mA",
          "",
          "[Mosfet]",
          "Forward Voltage = 0.7V",
          ""}
FileWrite("C:\powersim\data\config.ini", arr);
```

ファイル出力結果は以下になります。

[Diode]

[Diode]
Forward Voltage = 1.2V
Forward Current = 30mA

[Mosfet]
Forward Voltage = 0.7V

FileAppend(FilePath, Object, "A" (optional))

この関数は既存のテキストファイルの末尾にオブジェクトを追加します。ファイルが存在しない場合 、新しいファイルを作成します。フォルダが存在しない場合は、フォルダを作成します。

Object が文字列または数値の場合、テキストファイルに追加されます。Object が配列である場合、各セルは別々の行として書き込まれます。Object がグラフである場合は、最初の行はそれに続く数字のグラフ名です。

ファイルが既に存在する場合、既存のフォーマットが使用されます。ファイルが存在せず作成する場合にのみ、オプションの”A”の文字が有効になり ANSI 形式で記述されます。指定しない場合 Unicode 形式が使用されます。

GetLocal(Name)

この関数は、ローカルコンピュータに固有のパラメータを返します。

取得できるパラメータは以下の通りです。

名称	戻り値
PSIMPATH	PSIM フォルダのフルパス 例： "C:\Powersim\PSIM11.1.2_NetworkX64\"
PARAMPATH	パラメータファイルのフルパス 例： "C:\Data\"
SCHPATH	回路図ファイルのフォルダのフルパス パラメータツールウィンドウで使用されている場合、この値は空の文字列を返します。 例： "C:\Schematic files\"
SCHNAME	回路図ファイルの拡張子を含むファイル名 例： "file1.psimsch" パラメータツールウィンドウで使用されている場合、この値は空の文字列を返します。

13 グラフ機能

次の曲線データタイプとファイル機能が提供されます。

曲線のデータタイプ

曲線データは、特定の境界内の一つの変数の挙動を表す数値のセットです。曲線データの構成は次のとおりです。

- 曲線名
- 実数配列
- 検証フラグの配列、上記の配列内の各番号の 1 つのフラグ

MakeCurve(CurveName, NumberOfRows)

MakeCurve(CurveName, ArrayOfValues)

MakeCurve(CurveName, NumberOfRows, Formula,

VarName1, ArrayOfValues,

VarName2, StartValue, Increment …)

関数は、新しい曲線データを作成します。

関数のパラメータは次のとおりです。

CurveName:	カーブの名前
NumberOfRows:	曲線のサイズを変更することは出来ません、したがってこの値は正しく入れる必要があります。
ArrayOfValues:	配列のサイズは、行数やグラフとして使用するために、配列値に初期化されます。
Formula (オプション)	この式は曲線を初期化するために使用されます。StartValue は開始値、Increment は増分値として使用されます。全ての変数に同じ値（定数)を入れるためには、開始値に定数、増分値を 0 に設定します。パラメータとして定義されていない式の任意の変数は、メインスクリプトから取得されます。

Plot(PlotName, X-Axis, Curve1, Curve2, ArrayOfCurves, …)

この関数は、グラフをプロットします。

関数のパラメータは次のとおりです。

PlotName:	プロット名、これは文字列です。

X-Axis, Curve1, Curve2:　これらは曲線または数値の配列を使用することができます。表示する曲線の数に制限はありません。X を含む任意の曲線が複素数である場合は、実部と虚部を別々の曲線として表示されます。複数の曲線または数値の配列を含む配列は、パラメータのリストのどこにでも配置することができます。

「examples\Script」フォルダにある「plot sawtooth and triangle.script」の例では、のこぎり波と三角波がプロットされます。

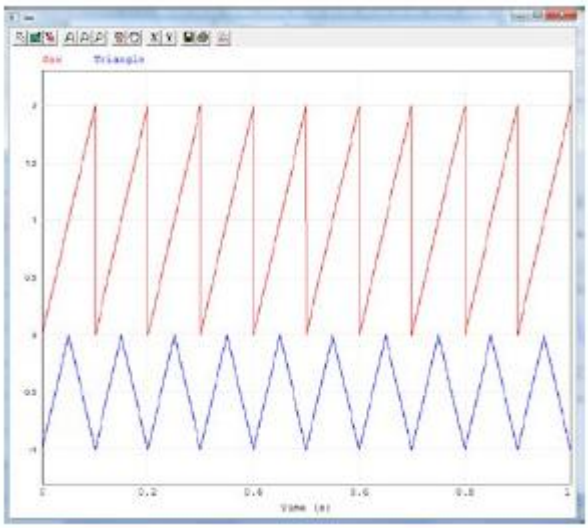
```
// Plotting sawtooth and triangular waveforms
ScriptOption("NoLog*");

t=MakeCurve("Time", Array(100u, 1, 100u)); // t = {1e-4, 2e-4, 3e-4, ..., 1}
Saw1 = ((t / 50m) % 2);
SetCurveName(Saw1, "Saw");

Saw2 = -abs(Saw1-1);
SetCurveName(Saw2, "Triangle");

Plot("Saw", t, Saw1, Saw2);
```

スクリプトを実行すると、以下の波形が生成されます。



BodePlot(PlotName, X-Axis, Curve1, Curve2, ArrayOfCurves, …)

この関数は 2 つのグラフウィンドウを表示します。最初のウィンドウは、大きさの 20 * log10 の複素数列を表示し、第 2 の窓は、複素数列の角度を表示します。

関数のパラメータは次のとおりです。

PlotName:　プロット名 これは文字列です。

X-Axis, Curve1, Curve2: これらは曲線または数値の配列を使用することができます。表示する曲線の数に制限はありません。X を含む任意の曲線が複素数である場合は、実部と虚部が別々の曲線として表示されます。複数の曲線または数値の配列を含む配列は、パラメータのリストのどこにでも配置することができます。

GraphRead(FilePath)

GraphRead(FilePath, “CurveName1”, “CurveName2”, …)

この GraphRead(FilePath)関数は.smv または.txt 拡張子を持つ全体の Simview ファイルを読み込み、曲線の配列に読み込みます。返り値は曲線の配列です。最初のセルは、X 軸になります、その後のセルは、ファイル内の曲線の残りの部分になります。例えば、MyGraph.smv ファイルは以下を含みます。

Time	V1	V2
0.001	1.1	2.2
0.002	3.3	4.4
…	…	

以下の関数呼び出しにて：

```
arr = GraphRead("C:\Powersim\MyGrpah.smv");
```

arr [0]には最初の列（arr [0] = {0.001 0.002…}）が含まれ、arr [1]には 2 番目の列（arr [1] = {1.1 3.3 …}）が含まれます。

大きなデータファイルの場合、GraphRead(FilePath)は完全なデータではなくオーバーサンプリングされたデータを返します。大きなデータファイルは、読み取られたときにコンピューターに残る物理 RAM メモリが 1GB 未満になるように定義されています。この場合に完全なデータをロードするには、関数 GraphRead（FilePath、“ CurveName1”、“ CurveName2”、…）を使用して、選択したカーブのみをロードします。

GraphWrite(FilePath, X-Axis, Curve1, Curve2, ArrayOfCurves, …)

この関数は Simview のグラフデータをファイル出力します。ファイルパスの拡張子が.txt の場合は、テキストファイルを書き込みます。それ以外の拡張子を指定した場合は,.smv 形式のファイルが書き込まれます。

関数のパラメータは次のとおりです。

PlotName: プロット名。 これは文字列です。

X-Axis, Curve1, Curve2: これらは曲線または数値の配列を使用することができます。表示する曲線の数に制限はありません。X を含む任意の曲線が複素数である場合は、実部と虚部が別々の曲線として表示されます。複数の曲線または数値の配列を含む配列は、パラメータのリストのどこにでも配置することができます。

GraphMerge(OutputFileName,InputFile1,InputFile2,InputFile3…)

この関数は2つかそれ以上のSIMVIEWグラフをマージしてファイル名を指定して保存できます。SIMVIEWのマージ機能と同じことができます。 すべての関数パラメータはフルパスで指定してください。

GetLocal() を使う場合はパラメータファイル‘GetLocal(PARAMPATH)’ か回路ファイル ‘GetLocal(SCHPATH)’で始まるフルパスで設定してください。
例えば

```
OutputFile = GetLocal(PARAMPATH) + "mergedGraph.smv";

File1 = GetLocal(PARAMPATH) + "graph1.smv";

File2 = GetLocal(PARAMPATH) + "graph2.smv";

GraphMerge(OutputFile, File1, File2);
```

入力ファイルは.smv か.txt のどちらかとなります。

GetCurve(ArrayOfCurves, CurveName)

関数はは ArrayOfCurves 内の CurveName の曲線検索し、曲線を返します。例えば、

```
str = GetCurve(graph, "V1");
```

GetCurveName(Curve1)

この関数は、曲線の名前を返します。例えば、

```
str = GetCurveName(Curve1);
```

SetCurveName(Curve1, NewName)

この関数は、曲線に新しい名前を設定します。例えば、

```
SetCurveName(Curve1, "XAmplitude");
```

14 シミュレーション機能

以下のシミュレーション機能が提供されます。

Simulate(SchematicFilePath, SimviewFilePath, SimulationOptions, ReturnGraph)

SimulateLT(SchematicFilePath, SimviewFilePath, SimulationOptions, ReturnGraph)

この **Simulate** 関数は PSIM エンジンを使用して、回路図ファイルを開き、シミュレーションを実行します。

この **SimulateLT** 関数は LTspice を使用して、のシミュレーションを実行します。

この関数はシミュレーションが終わるまでリターンしません。「SchematicFilePath」(回路ファイルパス)は必須パラメータです。それ以外のすべてのパラメータはオプションです。

パラメータ「SimviewFilePath」は、シミュレーション出力ファイルを定義します。**Simulate**、、および **SimulateSpice** の場合、ファイルの拡張子が.txt の場合、出力ファイルはテキスト形式になります。ファイルの拡張子が.smv の場合、出力ファイルはバイナリ形式になります。**SimulateLT** の場合、拡張子が.raw であっても出力ファイルはバイナリ形式になります。

パラメータ「ReturnGraph」は曲線の配列に設定される変数でなければなりません。AC スイープシミュレーションの場合、「ReturnGraph」は AC スイープのグラフとなります。

パラメータスイープシミュレーションの場合は、別の変数は、パラメータスイープのグラフに従うことができます。

SimulationOptions は文字列であり、それは" "で囲む必要があります。有効なオプションは次のとおりです。:

TotalTime
TimeStep
PrintTime
PrintStep
SaveFlag
LoadFlag

また、ここにも通常の変数を追加することができます。例えば、

```
Simulate("C:\Powersim\files\active filter.psimsch", "C:\Powersim\Graphs\active filter.txt",
"TotalTime=100m; TimeStep=10u; PrintTime=10m; PrintStep=2; SaveFlag=1; LoadFlag=0;
R1=11K;", graph1);
```

変数リストに加えて、シミュレート関数の前に定義されたすべての変数は、シミュレーションされる回路図に渡され参照されます。

ASimulate(SchematicFilePath, SimviewFilePath, SimulationOptions)

この関数は、それがすぐにリターンし、シミュレーションは別のスレッドで並列に実行されることを除き、‘Simulate’と同一です。このシミュレーションによって生成されるグラフがこの関数の後の計算に必要とされている場合はこの関数を使用せず’Simulate’を使用してください。

SimView(FileName,Curve1(optional),Curve2(optional),Curve3(optional) .....)

この関数は SIMVIEW を実行し(実行されていない場合)FileName で設定したファイルを開きます。 ファイル名は SIMVIEW ファイルのフルパスで設定してください。パラメータリストにオプションのカーブ名が含まれている場合はそれらの波形が表示されます。

例えば:

```
File1 = GetLocal(PARAMPATH) + "graph1.smv";
Simview(File1, "vload");
```

となります。

「examples\script」フォルダにある「simulation control example.script」の例は、タイムステップと合計時間を定義し、シミュレーションと計算を実行し、結果をファイルに保存して波形をプロットする方法を示しています。

```
// This example does the following:
// - Define time step and total time
// - Use the defined time step and total time to simulate the circuit sch1.psimsch
// - Create a sine wave V_sine
// - Plot V3 from the simulation result plus V_sine
// - Write the original simulation result plus V_sine to a new file
Step_t = 1E-5; // define simulation time step
Total_t = 0.1; // define simulation total time
//t = Array(1E-5, .1, 1E-5);
t = Array(Step_t, Total_t, Step_t);
File1 = GetLocal("PARAMPATH") + "Files\sch1.psimsch";
smvFile = GetLocal("PARAMPATH") + "Files\graph1.txt";

// Simulate the circuit
Simulate(File1, smvFile, TimeStep = Step_t, TotalTime=Total_t, graph1);
Count = sizeof(graph1); // obtain simulation result
V3 = GetCurve(graph1, "v3"); // obtain v3 curve
V4 = GetCurve(graph1, "v4"); // obtain v4 curve

// Create a sine wave
V_sine = 110 * sin( (t*60*2*pi) + (60/180*pi)); // Sin wave: 60Hz, 60 degrees shift

// Plot V3 and V_sine vs. t
Plot("DD", t, V3, V_sine);
GraphWrite(GetLocal("PARAMPATH") + "Files\Files\graph2.smv", graph1, V_sine);
```

PSIM のサンプルフォルダ「examples \ script」内に、スクリプトが実行できる様々な例がありますので参照してください。

ご注意

1. 本資料に記載された製品の仕様は、予告なく変更することがあります。
2. 本資料の内容については、万全を期しておりますが、万一ご不明な点などがありましたら、弊社までお申しつけください。
3. 本資料に記載された情報に起因する損害または特許権その他権利の侵害に関しては、弊社は一切の責任を負いません。
4. 本資料によって第三者または弊社の特許権その他権利の実施権を許諾するものではありません。
5. 弊社の書面許諾なく、本資料の一部または全部を無断で複製することを固くお断りします。
6. 本資料に記載されている会社名、商品名は、各社の商標または登録商標です。