

KETCindy リファレンスマニュアル

KETCindy Project Team

2016 年 12 月 20 日

目次

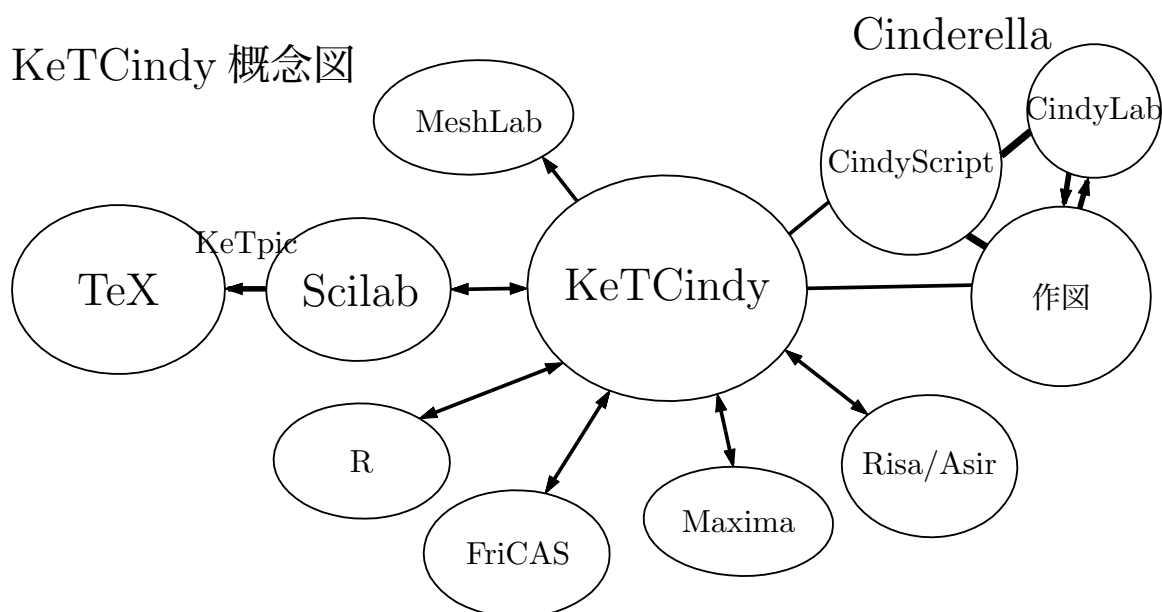
1	KETCindy について	3
1.1	システムの構成	3
1.2	Cindyscript のスロットの概念	3
1.3	プロットデータ	4
1.4	Cinderella の作図ツール	5
1.5	用語解説	6
2	定数と変数	7
3	関数リファレンス	9
3.1	設定・定義	9
3.2	描画	22
3.3	プロットデータの操作	64
3.4	微積分など	79
3.5	作表	82
3.6	値の取得と入出力	91
3.7	その他	102
4	他の数式処理ソフトなどとの連携	109
4.1	R との連携	109
4.2	Maxima との連携	119
4.3	Scilab との連携	130
4.4	Risa/Asir との連携	131
4.5	FriCAS(Axiom) との連携	132
4.6	MeshLab との連携	133
4.7	表計算ソフトとの連携	138

5	アニメーション PDF : KeTCindymv	141
5.1	概要	141
5.2	設定	142
5.3	描画	143
6	KeTCindy3D	146
6.1	概要	146
6.2	設定・定義	148
6.3	描画	149
7	逆引辞典	181
7.1	点の作図	181
7.2	線の作図	184
7.3	多角形を描く	188
7.4	円の作図	190
7.5	領域を塗る	191
7.6	関数のグラフ・曲線	193
7.7	その他	194
7.8	値の取得・変換	195
7.9	スライダを作る	197
8	関数一覧	198

1 KeTCindy について

1.1 システムの構成

KeTCindy は, Cinderella での作図データを Scilab 版 KeTpic に渡し, \LaTeX ファイルを作成するためのスクリプトライブラリである。Cinderella によるインタラクティブな作図機能と, CindyScript によるプログラミングにより, \LaTeX 文書の挿入図を効率よく作成することができる。また, R, Maxima, Risa/Asir などの数式処理ソフトと連携して計算を行うことができる。



Cinderella で作図した図のデータは, Scilab のファイル (拡張子 sci) に書き出される。これを Scilab で処理して \LaTeX ファイルを作成する (Scilab 版 KeTpic)。できた \LaTeX ファイルを, 本文中に input コマンド で挿入すれば図が表示される。

Cinderella と Scilab やその他のソフトウェアとの連携には, バッチファイル (Mac ではシェルフファイル) を用いている。(概念図の両方向矢印) バッチファイルは kc.bat, シェルフファイルは kc.sh で, KeTCindy が目的に応じてこれらのファイルを書き出し, プラグイン KetCindyPlugin.jar でそれらを実行する手順になっている。

1.2 Cindyscript のスロットの概念

Cindyscript は Cinderella のプログラミング言語で, スクリプトエディタで記述する。スクリプトエディタは, 「スクリプト」メニューの「Cindyscript」を選択して開くか, Ctrl+9

(Windows) / ⌘+ 9 (Mac) で開く。

スクリプトエディタの左方に並んでいるフォルダアイコンのメニュー風のを「スロット」といい、Cindyscript の実行タイミングによりスクリプトを書き分けることができる。通常は Initialization スロットと Draw スロットを使う。

Initialization スロットは、スクリプトを実行すると最初に一度だけ実行される。従って、関数定義や変数の初期設定などを書く。ひな形やサンプルの cdy ファイルではここに KETlib というページがあり、パス名の設定やライブラリの読み込みなど、KeTCindy の初期設定に関する記述がある。

Draw スロットは、Cinderella の画面上でなにか操作が行われるたびに実行される。通常はここにスクリプトを書く。ひな形の cdy ファイルでは、Ketinit() などが記述された figures ページが用意されている。



1つのスロットに複数のページを作ることができる。KETlib 以外に初期設定のスクリプトを書く場合は、Initialization スロットに新しいページを作るのがよい。

1.3 プロットデータ

プロットデータ (Plot Data) とは、関数のグラフや幾何要素を描くデータのことであり、KeTCindy の処理の中核をなしている。本マニュアルでは PD と略すことがある。たとえば曲線は、描画範囲を分割して線分の集まりとして描画しており、このときのプロットデータはそれらの線分の端点のリストである。

プロットデータの名称は KeTCindy が自動的に命名する。その命名規則は次の通りである。

- ・名称の頭部は、プロットデータを作成する関数ごとに決まっている。
- ・第1引数に name が与えられる場合、name を頭部に付加する。

例: `Listplot("1", [[0,0],[1,2]]);` のとき, sg1

- ・関数によっては、第1引数の name を略すことができる。この場合、引数で用いられた点の名前を頭部に付加する。

例：Listplot([A,B,C]); のとき、sgABC

プロットデータを生成したときは、Cindyscript エディタのコンソールにその名称を表示する。上の例では、

```
generate Listplot sgABC
```

と表示される。プロットデータを操作する関数では、この PD の名称を用いる。

プロットデータの内容を知りたい場合は、Cindyscript の

```
println(プロットデータ名)
```

でコンソールに表示することができる。

プロットデータは、Cindyscript によるプログラムで作成してそれを KeTCindy で利用することもできる。Listplot() の例を参照のこと。ただし、要素の数が大きいと Scilab でエラーとなるので、1つのプロットデータの要素は 200 程度とするのがよい。これより多い場合は分割する。

1.4 Cinderella の作図ツール



動かすモード（選択モード）にする：標準状態



点を加える



直線を加える



線分を加える



中点を加える



交点を加える



平行線を加える



垂線を加える



角の二等分線を加える



円を加える



半径つき円を加える



焦点と通る点で決まる楕円



焦点と通る点で決まる双曲線



焦点と準線で決まる放物線



多角形を加える



角に印をつける



角度を測る



選択した要素を消去する



点をまとめて選択する





線分をまとめて選択する

設定メニューから「上のツールバーのカスタマイズ」を選び、「すべて表示」にすると現れる
ツール



鏡映



点の極線を描く

画面ツール（下のツールバー）



原点を移動する



矩形領域を画面サイズに拡大



画面を矩形領域サイズに縮小



軸と方眼を表示し格子点にスナップする

1.5 用語解説

インシデント

点が曲線上に乗っている状態を表す。

インスペクタ

幾何要素の属性などを管理するウィンドウ。

幾何要素

Cinderella の作図ツールで作図した点や直線などの要素

幾何点

幾何要素としての点。マウスドラッグで動かすことができる。

固定点

マウスドラッグで移動することのできない点

コンソール

スクリプトエディタの右下のエリア。

自由点

マウスドラッグで任意に動かすことのできる点。

スロット

Cindyscript で、スクリプトを書くとき、実行タイミングによりに分類するもの

スナップ

マウスポイントが格子点の近くに来ると格子点上にぴったり移動する。

Cinderella の画面の下方ツールのうち、磁石アイコンによりこのモードになる。

2 定数と変数

KeTCindy は Cindyscript で記述されている。CindyScript では、変数名は大文字と小文字を区別するが、関数名は大文字小文字を区別しない。Cindyscript のマニュアルでは組み込み関数名はすべて小文字で表記されている。例示されたスクリプトでは、変数も小文字である。そこで、KeTCindy では、組み込みの変数名・関数名と区別しやすいように、次の規則により名前を付けている。

- ・グローバルな変数はすべて大文字か、大文字で始まるものとする。
- ・局所変数は小文字で、関数定義の冒頭で `regional()` により局所変数として宣言する。
- ・関数名は大文字で始まる。

なお、CindyScript は関数型プログラミング言語であり、命令はすべて関数を用いて行われるが、本マニュアルでは、文脈により「コマンド」という表現も用いる。

定数

つぎのものが CindyScript に予約されており、小文字で表されている。

- `pi` : 円周率。Scilab には `% pi` で書き出される。
- `i` : 虚数単位。Scilab には `% i` で書き出される。

一般のプログラミングでは変数 `i` をループ変数としてよく使うが、CindyScript では `i` は予約定数と考え、変数として用いないことを勧める。ただし、変数としてまったく使えないわけではなく、変数として用いた後、必要があれば `i=complex([0,1])` を実行することにより虚数単位として再定義することができる。

予約変数

KeTCindy が内部的に使用する予約変数がある。そのうち次のものはユーザーが値を変更または設定することができる。設定は Initialization スロットの「KETlib」ページでおこなうが、`Fhead` と `Texparent` は `Draw` スロットでもよい。

- `Fhead` : 書き出されるファイル名の頭部
- `Texparent` : 親プロセスのファイル名
- `Dirhead` : パスの頭部

Dirlib	ライブラリ ketlib のパス
Dirbin	ketbin のパス
Dirwork	作業ディレクトリのパス
Shellfile	シェルフファイル名

以下の予約変数は、ライブラリが使用するグローバル変数であるので、ユーザーはこれらの変数名を使ってはいけない。なお、変数は大文字小文字を区別するので、小文字で書く分には支障はない。したがって、ユーザーが作るプログラムでは、すべて小文字か、先頭だけが大文字の変数を使うことを勧める。

ADDAXES, ArrowlineNumber, ArrowheadNumber, BezierNumber,
 COM0thlist, COM1stlist, COM2ndlist, Dq, FUNLIST,
 Fnamesc, Fnamescibody, Fnameout, Fnametex, GDATA LIST,
 GLIST, GCLIST, GOUTLIST, KCOLOR, KETPICCOUNT,
 KETPICLAYER, LETTERlist, LFmark, MilliIn, PenThick,
 PenThickInit, POUTLIST, SCALEX, SCALEY, SCIRELIST,
 SCIWRLIST, TenSize, TenSizeInit, ULEN, XMAX, XMIN, YaSize,
 YaThick, YMAX, YMIN, VLIST

3 関数リファレンス

3.1 設定・定義

関数 Addax(0 または 1)

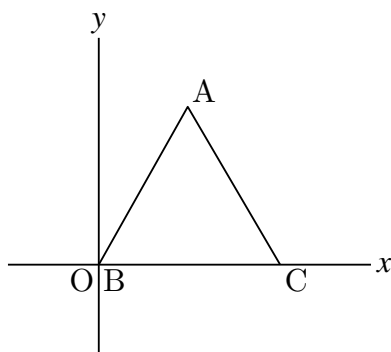
機能 座標軸を書くかどうかを定める

説明 Scilab の Closefile() の引数に対応する。

引数が 0 のとき座標軸を書かない（デフォルトは 1）

```
Listplot([B,A,C]);
```

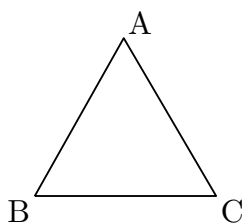
```
Letter([A,"ne","A",B,"se","B",C,"se","C"]);
```



```
Listplot([B,A,C,B]);
```

```
Letter([A,"ne","A",B,"sw","B",C,"se","C"]);
```

```
Addax(0);
```



[⇒ 関数一覧](#)

関数 Addcolor(描画コマンド , カラーコード)

機能 描画コマンドで描かれる線を指定色で描く

説明 描画コマンドはダブルクォートでくくって文字列とする。描画コマンド内にダブ

ルクウォートがある場合は、シングルルクウォートにする。option のある描画コマンドで option を指定しない場合は、必ず空リストを option として書く。描画色は、RGB または CMYK。描画色は画面と図版の両方に有効。

例：Addcolor("Plotdata('2','x^2','x',[])",[1,1,0]);

関数	Colorcode(文字 1, 文字 2, カラーコード)
機能	文字 1 から文字 2 へカラーコードを変換する。戻り値は変換されたコード。
説明	文字は, "rgb","cmyk","hsv" のいずれか。

例：Colorcode("rgb","cmyk",[1,0,0]);
RGB コードの [1,0,0] を CMYK に変換したコードを返す
Colorcode("cmyk","rgb",[0,1,1,0]);
CMYK コードの [0,1,1,0] を RGB に変換したコードを返す
Colorcode("rgb","hsv",[1,0,0]);
RGB コードの [1,0,0] を HSV に変換したコードを返す

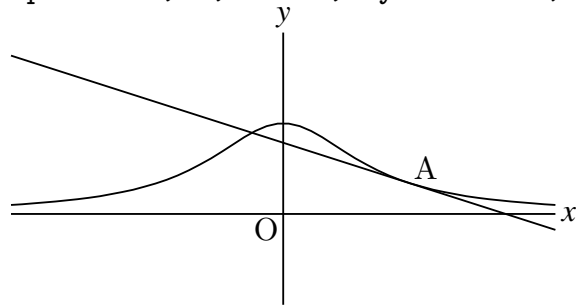
関数	Deffun(関数名, 定義のリスト)
機能	関数を定義する
説明	関数定義は, CindyScript の関数定義 $f(x):=式$ でもできるが, Deffun() を使うことにより, Scilab 側に渡すファイルに function 定義式 endfunction; が記述されるので, Scilab 側でこの関数を利用することができる。目的に応じて使い分けるとよい。 式のリストには if 文を用いた場合分けの関数式を記述することもできる。

例： $f(x) = \frac{1}{x^2 + 1}$ を定義し, グラフを描いて $x=1$ における微分係数を求める。

```
Deffun("f(x)",["regional(y)","y=1/(x^2+1)","y"]);  
Plotdata("1","f(x)","x");  
coeff=Derivative("f(x)","x",1);
```

点 A を作図しておく, 点 A をドラッグしたとき常に曲線上に乗せ, その点での接線を引くことができる。

```
A.xy=[A.x,f(A.x)];
coeff=Derivative("f(x)","x",A.x);
Lineplot("1",[A,[A.x+1,A.y+coeff]]);
```



例： $f(x) = \begin{cases} 1 & (x \geq 0) \\ -1 & (x < 0) \end{cases}$ を定義する。

```
Deffun("f(x)",["regional(y)","if(x>=0,y=1,y=-1)","y"]);
```

if 文はネストすることができる。

```
Deffun("f(x)",["regional y","if(x>1,y=1,
if(x>-1,y=x,y=-1))","y"]);
```

関数	Defvar(文字列)
機能	変数を定義する
説明	変数の定義を Scilab と共有する。また、Scilab の Assign リストに追加する。

例：Defvar("const=3");

複数の変数を定義するときはリストにする。

例：Defvar(["a",3,"b",1]);

関数	Drwxy()
機能	座標軸を描く
説明	座標軸はデフォルトでは最後に描かれるが、座標軸上に白抜きの点を表示するなど、先に描くことが必要な場合に用いる。

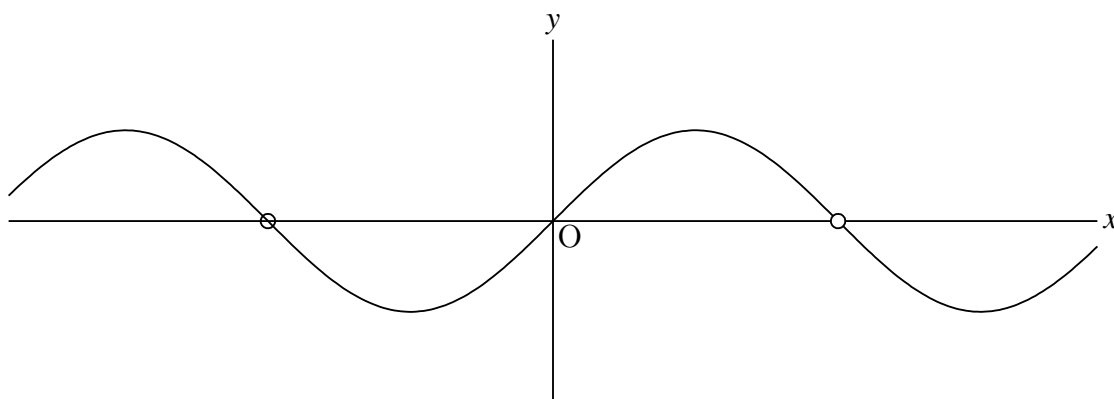
例：点 $(-\pi, 0)$ と $(\pi, 0)$ を白抜きの点で表示する。

```

Setax([7,"se"]);
Setpt(8);
Drwpt([-pi,0],0);
Drwxy();
Plotdata("1","sin(x)","x",["dr","Num=300"]);
Drwpt([pi,0],0);

```

このスクリプトでは, $\text{Drwpt}([-pi,0],0);$ を実行したのち座標軸を描き, 次に, $y = \sin x$ のグラフを描いてから $\text{Drwpt}([pi,0],0);$ を実行するので, 点 $(-\pi,0)$ の上を座標軸が通り, 点 $(\pi,0)$ は座標軸とグラフの上を通るので白抜きになる。



関数	FontSize(記号)
機能	フォントサイズを設定する
説明	次に Fontsize() を実行するまで有効 記号は, "t" , "ss" , "f" , "s" , "n" , "la" , "La" , "LA" , "h" , "H"

例:小さい方からいくつか表示する。

```

Psize(2);
Drawpoint([A,B,C,D,E,F,G]);
FontSize("t"); Letter([A,"s2","A"]);
FontSize("ss"); Letter([B,"s2","B"]);
FontSize("s"); Letter([C,"s2","C"]);
FontSize("la"); Letter([D,"s2","D"]);
FontSize("La"); Letter([E,"s2","E"]);
FontSize("h"); Letter([F,"s2","F"]);
FontSize("H"); Letter([G,"s2","G"]);

```

\dot{A} \dot{B} \dot{C} \dot{D} \dot{E} \dot{F} \dot{G}

[⇒ 関数一覧](#)

関数	Ketinit(options)
機能	K _ε TCindy を初期化する
説明	option 縦方向の倍率と描画領域を設定

例：Ketinit() : 倍率 1, 描画領域 $-5 \leq x \leq 5, -5 \leq y \leq 5$ (デフォルト)
Ketinit(2) : 倍率 2, 描画領域 $-5 \leq x \leq 5, -5 \leq y \leq 5$
Ketinit(2, [-2, 3], [-2, 4]) : 倍率 2, 描画領域 $-2 \leq x \leq 3, -2 \leq y \leq 4$

描画領域 (TeX に出力する領域) は制御点 SW (左下) と NE (右上) を対角とする矩形領域。描画領域を指定すると、制御点がなければその位置に作り、すでに存在する場合は何もしない。作成された制御点はドラッグして描画領域を変更することができる。

倍率は、Setscaling(倍率) を実行するのと同じ。ただし、Cinderella で作図した幾何要素に対しては無効。(Setscaling() の項参照)

[⇒ 関数一覧](#)

関数	Ptsize(n) , Setpt(n)
機能	表示する点の大きさを設定する。
説明	Ptsize() と Setpt() は同じである。デフォルトは 1 Ptsize() は CindyScript 風の語法, Setpt() は K _ε Tpic 風の語法 全体の点の大きさを設定する。点の大きさを個々に変えたい場合は、size オプションを用いる。

例：1 から 4 までの点の大きさ
あらかじめ、Cinderella の作図ツールで点 A,B,C,D を作図しておく。

```
Pointdata("1",A,["size=1"]);
Pointdata("2",B,["size=2"]);
Pointdata("3",C,["size=3"]);
Pointdata("4",D,["size=4"]);
```

Pointsize	1	2	3	4
	.	.	•	•

[⇒ 関数一覧](#)

関数 Setax()
機能 座標軸の書式を設定する。
説明 Scilab のみで実行する。Cinderella の描画面に反映されない。

引数は順番に

1. 軸の形状（直線は "l" , 矢印は "a"）デフォルトは直線
2. 横軸名 デフォルトは x
3. 横軸名の位置
4. 縦軸名 デフォルトは y
5. 縦軸名の位置
6. 原点名 デフォルトは O
7. 原点名の位置

それぞれダブルクォートでくくる。

7つの引数のうち n 番目だけを指定する場合は, [n,"内容"] で指定できる。

また, 後方はデフォルトなら省略できる。

例：座標軸の先端を矢印にし, 原点の北西に O を書く。

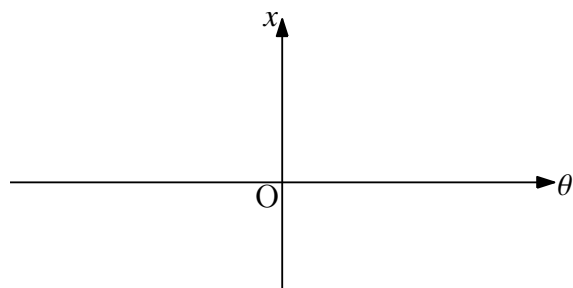
```
Setax(["a","","","","","nw"]);
```

例：原点の北西に O を書く。

```
Setax([7,"nw"]);
```

例：先端を矢印にし, 横軸を θ , 縦軸を x にして矢じりの左側に書く。

```
Setax(["a"," $\theta$ ","","x","w"]);
```



[⇒ 関数一覧](#)

関数 Definecolor(色名, 定義のリスト)
機能 色名を定義する
説明 ユーザー命名の色名を定義する。定義リストは RGB または CMYK のリスト
 各色 0 ～ 1 の範囲で指定する。定義した色名は, Setcolor(color,options) で使
 うことができる。なお, KeTCindy では, 68 色を色名で使うことができる。次の
 Setcolor(color,options) 参照)

例：暗い紫色を darkmaz の名称で定義して使う。

```
Definecolor("darkmaz",[0.8,0,0.8]);
Setcolor("darkmaz");
```

関数 Setcolor(color,options)
機能 描画色の設定
説明 引数 color はカラーコードまたは色の名称。
 カラーコードは RGB または CMYK をリストで与える。各色 0 ～ 1。
 色の名称は次頁の 68 色が指定できる。ただし, Cinderella の画面には色は反映され
 ない。
 color に色の名称を用いた場合は, option として, 透明度を 0 ～ 1 の数で指定でき
 る。1 が最も濃く, 0 は結果として色塗りをしない。

例 Cinderella の描画ツールと CindyScript で線分 AB,AC を 60° の角をなすよう
 に描いておき, 点 D と E を弧の両端になるように設定して

```
Setcolor([1,0,0]);
Circledata([A,D],["Rng=[0,pi/3]"]);
Arrowhead(E,[-1,0.8],[2,1]);
```

を実行すると, 矢じりつきの弧を赤で表示することができる。

1 行目は, Setcolor("red"); でもよい。

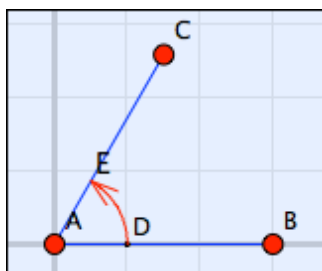
ただし, Cinderella の描画面では着色されない。

描画面でも着色したい場合は, オプション "color->[R,G,B]" を用いる。









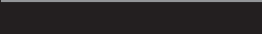


上の例の場合,

```
Circledata([A,D],["Rng=[0,pi/3]","color->[1,0,0]"]);
Arrowhead(E,[-1,0.8],[2,1,"color->[1,0,0]"]);
```


とすれば、描画面でも赤で表示される。



色の名前 (和名)	T _E X コマンド (英語)	[C,M,Y,K] のベクトル	文字 (例)	塗りつぶし (濃さ 1)
	greenyellow	[0.15,0,0.69,0]	あいうえお	
黄色	yellow	[0,0,1,0]	あいうえお	
	goldenrod	[0,0.1,0.84,0]	あいうえお	
	dandelion	[0,0.29,0.84,0]	あいうえお	
	apricot	[0,0.32,0.52,0]	あいうえお	
	peach	[0,0.5,0.7,0]	あいうえお	
	melon	[0,0.46,0.5,0]	あいうえお	
	yelloworange	[0,0.42,1,0]	あいうえお	
橙	orange	[0,0.61,0.87,0]	あいうえお	
	burntorange	[0,0.51,1,0]	あいうえお	
	bittersweet	[0,0.75,1,0.24]	あいうえお	
	redorange	[0,0.77,0.87,0]	あいうえお	
	mahogany	[0,0.85,0.87,0.35]	あいうえお	
	maroon	[0,0.87,0.68,0.32]	あいうえお	
	brickred	[0,0.89,0.94,0.28]	あいうえお	
赤	red	[0,1,1,0]	あいうえお	
	orangered	[0,1,0.5,0]	あいうえお	
	rubinered	[0,1,0.13,0]	あいうえお	
	wildstrawberry	[0,0.96,0.39,0]	あいうえお	
	salmon	[0,0.53,0.38,0]	あいうえお	
	carnationpink	[0,0.63,0,0]	あいうえお	
	magenta	[0,1,0,0]	あいうえお	
	violetred	[0,0.81,0,0]	あいうえお	
	rhodamine	[0,0.82,0,0]	あいうえお	
	mulberry	[0.34,0.9,0,0.02]	あいうえお	
	redviolet	[0.07,0.9,0,0.34]	あいうえお	
	fuchsia	[0.47,0.91,0,0.08]	あいうえお	
	lavender	[0,0.48,0,0]	あいうえお	
	thistle	[0.12,0.59,0,0]	あいうえお	
	orchid	[0.32,0.64,0,0]	あいうえお	
	darkorchid	[0.4,0.8,0.2,0]	あいうえお	
紫	purple	[0.45,0.86,0,0]	あいうえお	
	plum	[0.5,1,0,0]	あいうえお	
	violet	[0.79,0.88,0,0]	あいうえお	

色の名前 (和名)	T _E X コマンド (英語)	[C,M,Y,K] のベクトル	文字 (例)	塗りつぶし (濃さ 1)
	royalpurple	[0.75,0.9,0,0]	あいいうえお	
	blueviolet	[0.86,0.91,0,0.04]	あいいうえお	
	periwinkle	[0.57,0.55,0,0]	あいいうえお	
	cadetblue	[0.62,0.57,0.23,0]	あいいうえお	
	cornflowerblue	[0.65,0.13,0,0]	あいいうえお	
	midnightblue	[0.98,0.13,0,0.43]	あいいうえお	
	navyblue	[0.94,0.54,0,0]	あいいうえお	
	royalblue	[1,0.5,0,0]	あいいうえお	
青	blue	[1,1,0,0]	あいいうえお	
	cerulean	[0.94,0.11,0,0]	あいいうえお	
	cyan	[1,0,0,0]	あいいうえお	
	processblue	[0.96,0,0,0]	あいいうえお	
	skyblue	[0.62,0,0.12,0]	あいいうえお	
	turquoise	[0.85,0,0.2,0]	あいいうえお	
	tealblue	[0.86,0,0.34,0.02]	あいいうえお	
	aquamarine	[0.82,0,0.3,0]	あいいうえお	
	bluegreen	[0.85,0,0.33,0]	あいいうえお	
	emerald	[1,0,0.5,0]	あいいうえお	
	janglegreen	[0.99,0,0.52,0]	あいいうえお	
	seagreen	[0.69,0,0.5,0]	あいいうえお	
緑	green	[1,0,1,0]	あいいうえお	
	forestgreen	[0.91,0,0.88,0.12]	あいいうえお	
	pinegreen	[0.92,0,0.59,0.25]	あいいうえお	
	limegreen	[0.5,0,1,0]	あいいうえお	
	yellowgreen	[0.44,0,0.74,0]	あいいうえお	
	springgreen	[0.26,0,0.76,0]	あいいうえお	
	olivegreen	[0.64,0,0.95,0.4]	あいいうえお	
	rawsienna	[0,0.72,1,0.45]	あいいうえお	
	sepia	[0,0.83,1,0.7]	あいいうえお	
茶色	brown	[0,0.81,1,0.6]	あいいうえお	
	tan	[0.14,0.42,0.56,0]	あいいうえお	
灰色	gray	[0,0,0,0.5]	あいいうえお	
黒	black	[0,0,0,1]	あいいうえお	
白	white	[0,0,0,0]		

⇒ 関数一覧

関数 Setmarklen(数)
機能 座標軸の目盛の長さを設定する
説明 Htickmark() , Vtickmark() で座標軸に目盛を入れるとき、その長さを設定する。
⇒ [Htickmark\(\[横座標, 方向, 文字\]\)](#)

関数 Setorigin(座標)
機能 描画する座標軸の原点を設定（移動）する
説明 描画する座標軸の原点を引数の座標とする。座標は点の識別名でもよい。

例：原点を (3,2) として座標軸を描く。

```
Setorigin([3,2]);
```

原点を点 A の位置にして座標軸を描く。

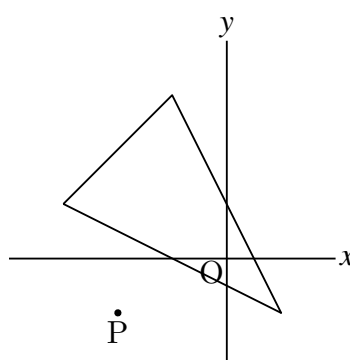
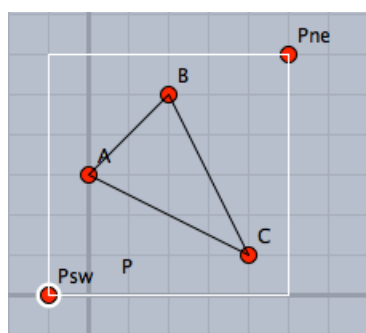
```
Setorigin(A);
```

注意：座標軸とともに、原点の O、軸名なども移動するが、座標系が変更されるわけではない。

例：原点は (3,2) に移動するが、スクリプトではもとの座標系を使う。

```
Setorigin([3,2]);  
Listplot([A,B,C,A]);  
Psize(3);  
Drawpoint([1,1]);  
Letter([[1,1], "s2", "P"]);
```

左が実行時の Cinderella の画面、右が $\text{T}_\text{E}\text{X}$ の結果。



関数 Setpen(数)
機能 線の太さを設定する

関数 Setpt(数)
機能 表示する点の大きさを設定する

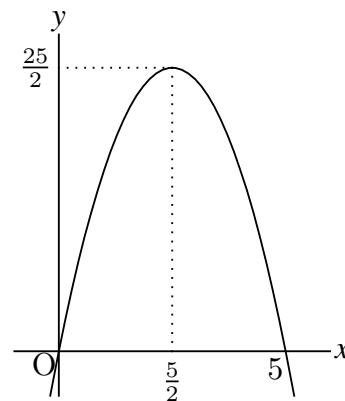
関数 Setscaling(倍率)
機能 縦方向の倍率を設定する

説明 2次関数の応用問題などでは、グラフが縦に大きくなる場合があり、y 軸方向のスケールを変えることがよくある。次のスクリプトは、 $f(x) = -x^2 + 10x$ のグラフを縦軸方向を半分にして描くものである。

```

Setscaling(0.5);
A.xy=[0,25/4];
B.xy=[5/2,25/4];
C.xy=[5/2,0];
Listplot([A,B],["do"]);
Listplot([C,B],["do"]);
Plotdata("1",-2*x^2+10*x,"x");
Letter([[5,0], "s2w", "5", [0,25/2], "w2",
      "$\frac{25}{2}$", C, "s4", "$\frac{5}{2}$"]);

```



ここで、点 A,B の座標が

```

A.xy=[0,25/4];
B.xy=[5/2,25/4];

```

となっていることに注意されたい。y 座標をあらかじめ半分になっている。すなわち、Cinderella で作図した幾何要素に対しては Setscaling は無効である。これは、Putpoint 関数を用いて点の位置を決めても同じである。

たとえば、次のスクリプトでは、Cinderella の画面上では 2 本の線分が点 B でつながるが、書き出された T_EX の図では離れてしまう。

```

Setscaling(0.5);
Putpoint("A", [0,2]);
Putpoint("B", [2,2]);

```

```
Listplot([A,B]);  
Listplot("1",[[0,0],[2,2]]);
```

関数 Setunitlen(文字列)
機能 単位長を設定する。デフォルトは 1cm

関数 Setwindow()
機能 出力する描画領域を設定する

説明 出力する描画領域は、通常は2点 SW と NE を対角とする矩形領域である。この2点をドラッグすることによりビジュアルに描画領域を決められる。
しかし、これとは別に出力範囲を設定したい場合にこの関数を用いる。
また、表を作成したときは、表の範囲が出力範囲として優先される (Tabledata() を実行したとき) ので、表外に図を描いた場合は、最後にこの関数で出力範囲を指定して書き出す。

[⇒ 関数一覧](#)

3.2 描画

描画関数は曲線などを作図する関数である。

基本的な書式は

関数名 (name , 点リストなど , options);

であるが, name の不要なものもある。

name は, プロットデータの名称を指定するもので, 関数ごとに決められた頭部のあとに付けられる。name が不要の場合は K_ETCindy が自動的に名称を作成する。

点リストなどは, 点の座標, 点の識別名, 複数の点のリスト, 複数の点を示す文字列などがあり, 関数によって異なる。点は Cinderella で作図した幾何要素の点を利用できる。

options は, 線種・表示する文字列・解像度・出力の有無などを指定するオプション群。

線種はつぎの4通り。デフォルトは実線。

"dr, n" 太さ n の実線で描く。Scilab のファイルに Drwline() を出力する。

"da,m,n" 破線を描く。Scilab のファイルに Dashline() を出力する。

m は破線の長さ, n は破線の間隔 (m,n は省略可)

m,n オプションは Cinderella の描画面には反映されない。

"id,m,n" ギャップからはじまる破線を描く。Scilab のファイルに Invdashline() を出力する。

"do,m,n" 点線で描く。Scilab のファイルに Dottedline() を出力する。

m は点の間隔, n は太さ (m,n は省略可)

描画色指定は, Cindyscript の表記と同様で, RGB のリストで指定するか, 色名を用いる。

例: "color->[0,0.7,0]" で暗い緑になる。

出力の有無は

"notex" Cinderella 画面上で補助線として用いた図形を Scilab に出力しない

"nodisp" Cinderella 画面上にも出力しない

"nodisp" は画面上にも, Scilab へのデータにも出力されないが, プロットデータは作成され, それを戻り値とするので, プロットデータだけを利用したい場合に有効である。

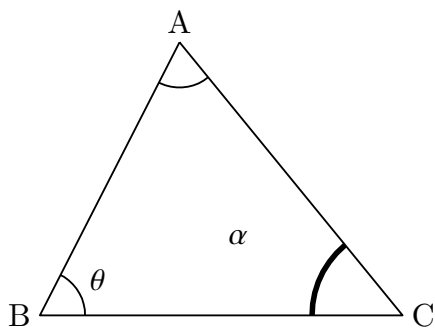
例 pdata=Circledata([A,B],["nodisp"]);

として, プロットデータ pdata を処理する。

関数 Anglemark(点リスト , options)
機能 点リスト [A,B,C] で示された角に弧の形状の角の印をつける。
説明 options は次の通り。
 数値 角の印の大きさ。デフォルトは 1
 線種 "dr, n" , "da,m,n" , "do,m,n"
 "Expr=文字" : 文字を入れる
 "Expr=位置 , 文字" : 位置を指定して文字を入れる。位置は頂点からの距離。

例：三角形の内角に印をいれ，文字を書き込む。

```
Listplot([A,B,C,A]);
Letter([A,"n1","A",B,"w1","B",C,"e1","C"]);
Anglemark([B,A,C]);
Anglemark([C,B,A],["Expr=\theta"]);
Anglemark([A,C,B],[2,"dr,3","Expr=2,\alpha"]);
```



※角の印には平行四辺形の形状のものもある。Paramark() を参照のこと。

[⇒ 関数一覧](#)

関数 Arrowdata(name,[始点 , 終点] , options)
機能 2 点間を結ぶ矢線を描く。プロットデータ名の頭部は ar
説明 name は，座標を数値で与えるときに必要。幾何要素の識別名で与えるときはなくてもよい。
 options は矢じりの形状などの指定で
 [矢じりの大きさ, 開き角, 矢じり位置, 線種, 線の表示色] のリストで与える。
 開き角は 60 分法で与える。ただし，° はつけない。5 未満の時は 18° の倍数指定とする。
 矢じり位置は，線分の長さを 1 とした始点からの距離。

ただし、Cinderella の画面上には全ては反映されない。たとえば、太さ指定をしても太さは同じ。

例：線分 AB を矢線にする。

```
Arrowdata([A,B]);
```

始点が (2,0) , 終点が (4,3) , 開き角 45° , EF の中点に矢じりの先端

```
Arrowdata("1",[2,0],[4,3]],[1,45,0.5]);
```

大きさ 2 , 太さ 2

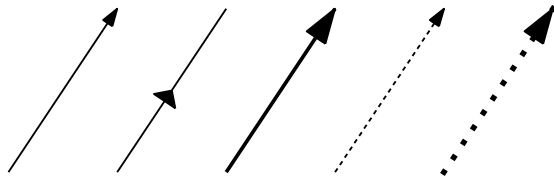
```
Arrowdata([C,D],[2,1,1,"dr,2"]);
```

破線で太さ 0.5

```
Arrowdata([E,F],["da,0.5"]);
```

少し間が空いて太い点線で、Cinderella の画面上では赤で表示する。

```
Arrowdata([G,H],[2,1,"do,2,3","color-$$[1,0,0]"]);
```

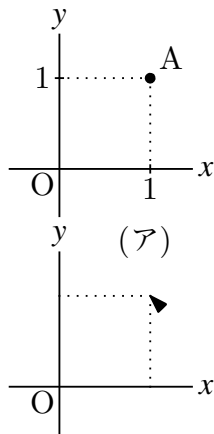


[⇒ 関数一覧](#)

関数	Arrowhead(点 , 方向 , options) , Arrowhead(点 , プロットデータ,options)
機能	点に矢じりだけを描く
説明	<p>指定された位置に、指定された方向を向いた矢じりだけを描く。</p> <p>点は座標または幾何要素名。方向は原点から見て座標 [a,b] の方向。</p> <p>options は [大きさ, 矢じりの開き角, 形状と位置] のリスト。</p> <p>矢じりの開き角は 60 分法で片側半分の角。</p> <p>形状は, "f" : 塗りつぶしの三角形 (デフォルト) または "1" : ラインのみ。</p> <p>位置は, "t" (デフォルト) または "c" , "b"</p> <p>"t" は矢じりの先端が終点に一致, "c" は三角形の中心が終点と一致, "b" は終点が矢じりの底辺にのる。</p> <p>プロットデータを指定したときは、曲線上の点に矢じりをつける。</p> <p>曲線には向きがあり、それによって矢じりの向きが決まる。" Invert(プロットデータ)" とすると反対向きの矢じりになる。</p>

曲線の向きとは, 曲線を描くときの順序で, プロットデータの順序でもある。invert() はこのプロットデータを逆順にするものである。

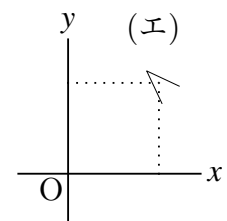
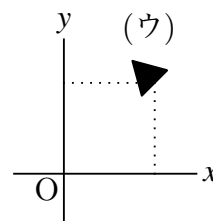
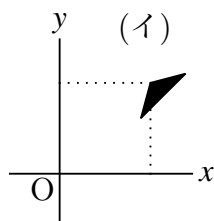
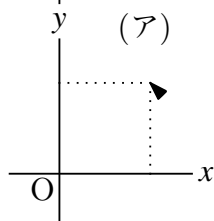
例：点 A が下図の位置のとき (ア) Arrowhead(A,[-1,1]);



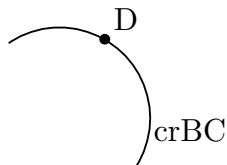
(イ) Arrowhead([1,1],[-1,1],[2,60]);

(ウ) Arrowhead(A,[-1,1],[2,30,"b"]);

(エ) Arrowhead([1,1],[-1,1],[2,20,"lc"]);



曲線 crBC 上の点 D が
下図のようなとき



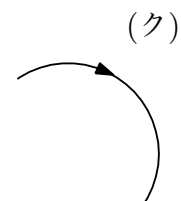
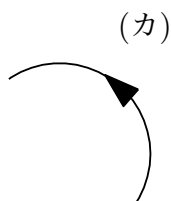
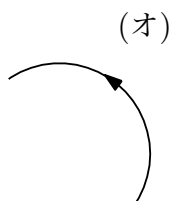
(オ) Arrowhead(D,"crBC");

(カ) Arrowhead(D,"crBC",[2]);

(キ) Arrowhead(D,"crBC",[2,30,"l"]);

(ク) Arrowhead(D,"Invert(crBC)");

D の座標がわかれば, それでもよい



関数 Bezier(名前, 節点リスト, 制御点リスト, [オプション])

機能 単独のベジエ曲線を描く

説明 制御点は, 各区間に対して, 3 次の場合 2 個, 2 次の場合 1 個のリストで与える。

[[F,G], [H], ...]

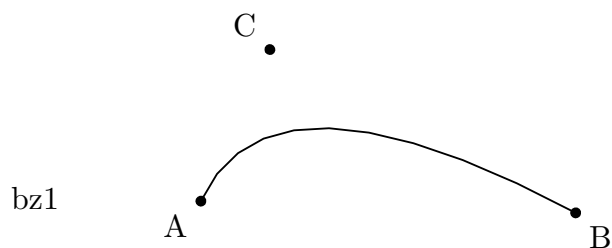
オプション

"Num=..." : 節点間の分割数 (分点数 -1) を指定できる。(デフォルトは 10)

例：

2 次ベジエ曲線

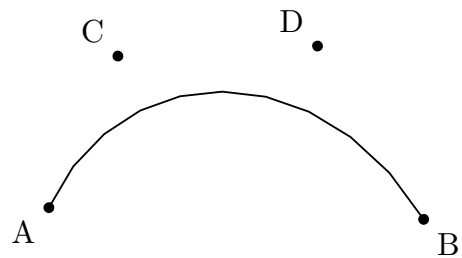
Bezier("1",[A,B],[C]);



3 次ベジエ曲線

Bezier("c",[A,B],[C,D]);

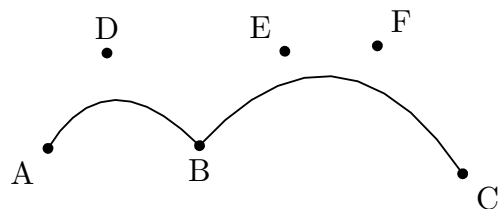
bzc



つなげる

Bezier("3",[A,B,C],[[D],[E,F]]);

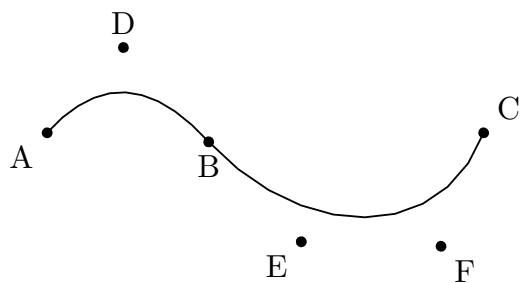
bz3



D,B,E を 1 直線上にとると，滑らかにつながる

Bezier("S",[A,B,C],[[D],[E,F]]);

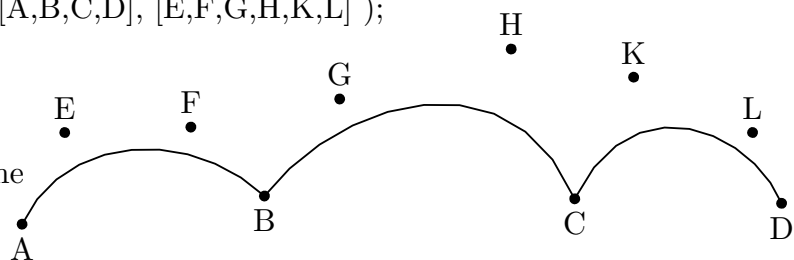
bzS



全て同じ次数の場合，次のようにしてもよい.

Bezier("name", [A,B,C,D], [E,F,G,H,K,L]);

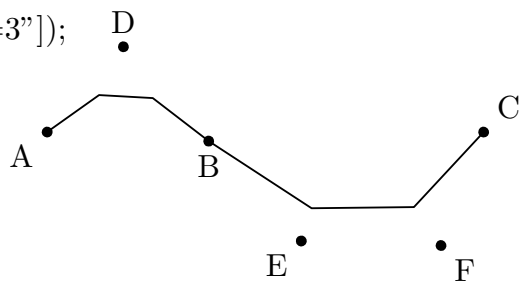
bzname



オプション

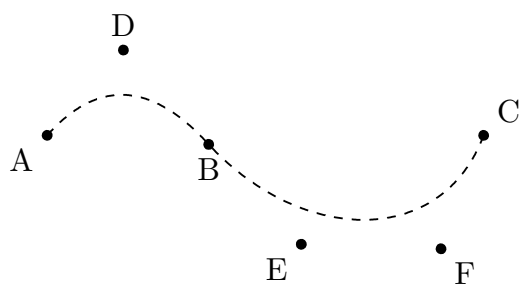
Bezier("1a",[A,B,C],[[D],[E,F]],["Num=3"]);

bz1a



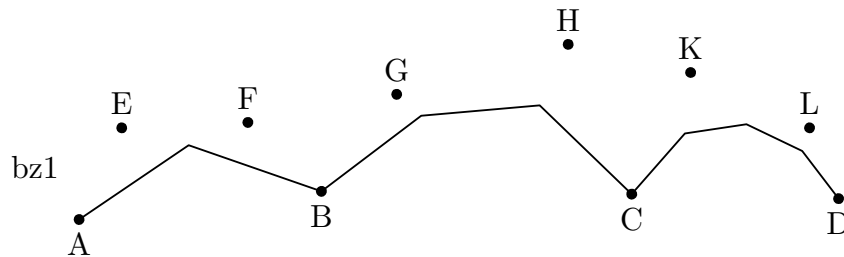
Bezier("d5e",[A,B,C],[[D],[E,F]],["Num=200","da"]);

bzd5e



Num を（ベクトルとして）区間ごとに与えることもできる。

Bezier("1", [A,B,C,D], [E,F,G,H,K,L], ["Num=[2,3,4]"]);



⇒ 関数一覧

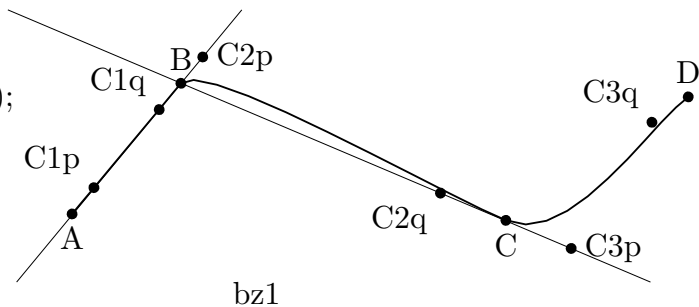
関数 Beziersmooth(名前, 節点リスト, [オプション])

機能 節点間を 3 次ベジエ曲線でスムーズに結んだ曲線を描く

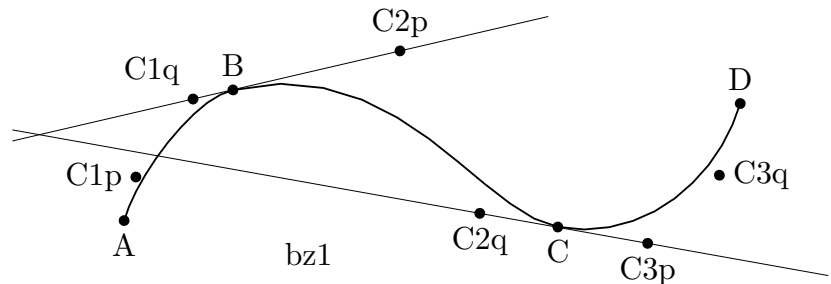
説明 節点をはさむ制御点は 1 直線上にとる（したがって、1 つは半自由点で、直線上しか動けない）。制御点は自動的に配置される。その後、節点や制御点を動かして、描きたいものにする。

例：

Beziersmooth("1",[A,B,C,D]);



その後、節点や制御点を動かして、描きたいものにする。ただし、C2p は C1q と B を通る直線上しか動けない。C3p は C2q と C を通る直線上しか動けない。



関数 Beziersym(名前, 節点リスト, [オプション])

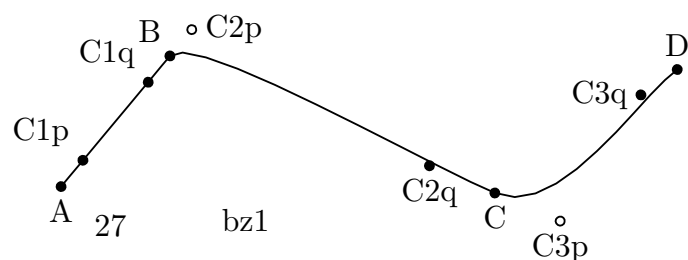
機能 節点間を 3 次ベジエ曲線でスムーズに結んだ曲線を描く

説明 節点をはさむ制御点は節点に関し対称（片方は表示されず、動かせない）。制御点は自動的に配置される。その後、節点や制御点を動かして描きたいものにする。

例：

Beziersym("1",[A,B,C,D]);

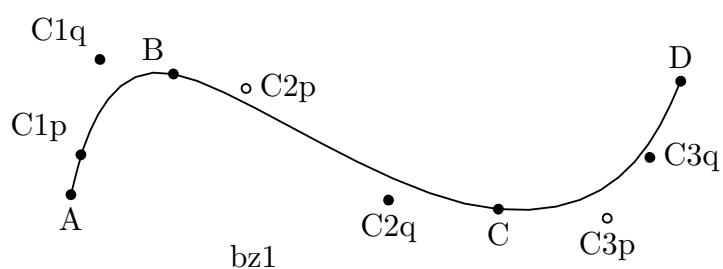
C2p と C3p は表示され



ない。

その後、節点や制御点を動かして、描きたいものにする。

C2p と C3p は表示されず、動かさない。



⇒ 関数一覧

関数 Bowdata(点リスト , options)

機能 弓形を描く

説明 点リストで与えられた2点を結ぶ弓形を描く。

2点を反時計回りに回る方向に弓形を描く。

options は, [曲がり , 空白サイズ , 文字, 線種]

曲がり は弧の曲がり具合の指定。デフォルトは1

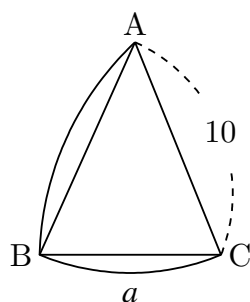
空白サイズ は中央にあける空白の大きさ

文字は, "Expr=文字"

また, "Expr=位置 , 文字" で位置を指定して文字を入れる。位置は e,w,n,s,c

例 三角形 ABC の各辺に弓形マークをつけ記号を入れる。

```
Listplot([A,B,C,A]);  
Letter([A,"n1","A",B,"w1","B",C,"e1","C"]);  
Bowdata([A,B]);  
Bowdata([B,C],[1,"Expr=s3,a"]);  
Bowdata([C,A],[2,1.2,"Expr=10","da"]);
```



以上が基本。これに加え、文字を回転して表示する方法がある。

ただし、Cinderella の画面には反映されない。

文字をを回転するには次のように書く。

”Exprrot=微小移動，文字”

微小移動は t, n, u

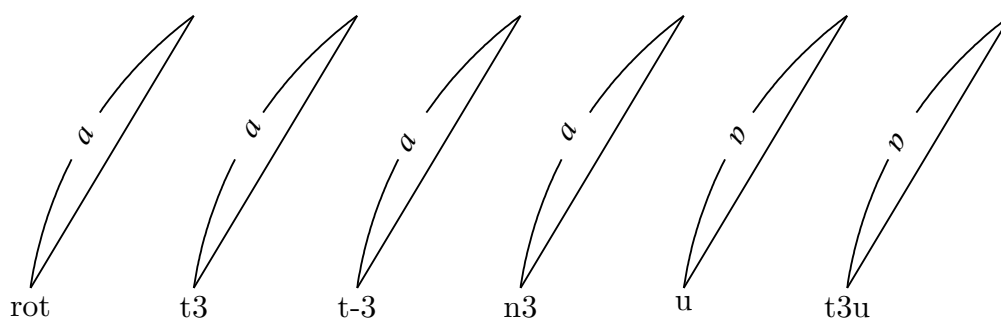
t は線分方向の微小移動。移動量は数字をつける。正負が可。

n は線分と垂直方向の微小移動

u は上下反転

t, n, u は組み合わせることができる。 以下にいくつか例を示す。

```
Bowdata([B,A],[1,1,"Exprrot=a"]);  
Bowdata([D,C],[1,1,"Exprrot=t3,a"]);  
Bowdata([F,E],[1,1,"Exprrot=t-3,a"]);  
Bowdata([H,G],[1,1,"Exprrot=n3,a"]);  
Bowdata([L,K],[1,1,"Exprrot=u,a"]);  
Bowdata([N,M],[1,1,"Exprrot=t3u,a"]);
```



[⇒ 関数一覧](#)

関数 Bspline(名前, 制御点リスト, [オプション])

機能 2次 B-spline 曲線を描く

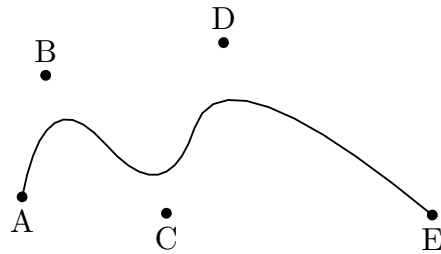
説明 節点は自動的に計算され、表示されない

例：Bspline("1",[A,B,C,D,E]);

Bezier("1",[A,(B+C)/2,(C+D)/2,E],[B,C,D]);

と同じ。曲線の名前が bz1 ではなく bzb1 となる。

通常の B-spline 曲線の端の制御点の代わりに、端点を動かせるようにしている。

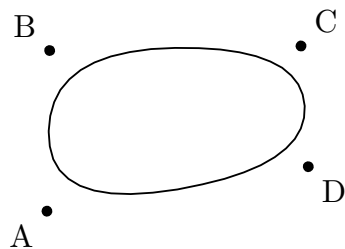


例：Bspline("1",[A,B,C,D,A]);

リストの最初と最後が同じ場合は閉曲線になる。

Bezier("1",[(D+A)/2,(A+B)/2,(B+C)/2,(C+D)/2,(D+A)/2],[A,B,C,D]);

と同じ。



関数 CRspline(名前, 節点リスト, [オプション])

機能 単独の Catmull-Rom スプライン曲線を描く

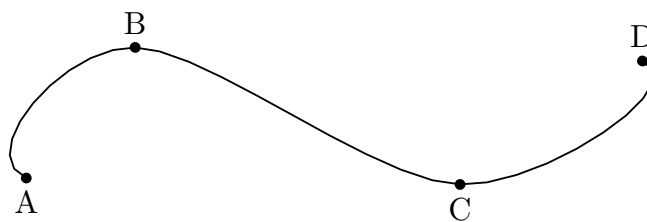
説明 自由点は、節点のみで、制御点は節点から作られ移動はできない。

オプションに、通常のオプションのほか、次の2つが使える。

"size->n" 画面上での線の太さを指定する。

"pointsize->n" 制御点の大きさを指定する。0 のとき非表示となる。

例：CRspline("3",[A,B,C,D]);



関数 Circledata(name, リスト,options)

機能 円または多角形を描く。

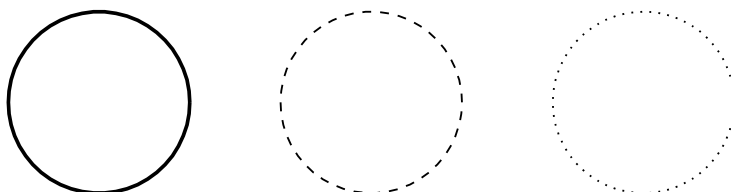
説明

中心の点と、円周上の 1 点、または 3 点をリストで与えて円を描く。
プロットデータの名前は、"cr" に引数の name を付加したものとなる。
中心と円周上の点を、座標ではなく幾何要素名で指定する場合は name は省略可。
options は以下のものをリストで与える。省略した場合は実線で円が描かれる。

"Rng=[θ_1, θ_2]" 角 θ_1 から θ_2 の範囲の弧を描く。角は弧度法で与える。
"Num=分割数" 円を描くときの分割数。値が小さい場合は多角形になる。
線種 "dr, n" , "da,m,n" , "do,m,n"

例：原点中心，半径 2 の円を描く `Circledata("1",[[0,0],[2,0]]);`
A 中心，半径 AB の円 を描く `Circledata([A,B]);`
A 中心，半径 2 の円 を描く `Circledata([A,A+[2,0]]);`
3 点 A,B,C を通る円を描く `Circledata([A,B,C]);`
この場合，できた円の中心を `Pointdata("1",[crABCcenter]);` で
作図できる。

下図左より，A 中心，半径 AB の円 を
太さ 2 の実線で描く `Circledata([A,B],["dr,2"]);`
破線で描く `Circledata([A,B],["da"]);`
点線で描く `Circledata([A,B],["do"]);`



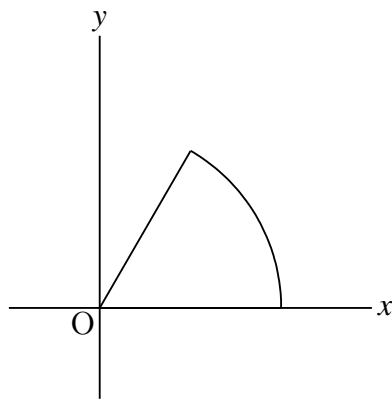
例：A 中心，半径 AB，中心角 60° の弧を描く。

`Circledata([A,B],["Rng=[0,pi/3]"]);`

このとき，扇型を描くのであれば，2 本の半径を引く必要がある。そのためには，A が原点，B が x 軸上にあれば，点 A,B 以外にもうひとつ点 C をとり，その位置を CindyScript を用いて

`C.xy=|A,B|*[cos(pi/3),sin(pi/3)]`

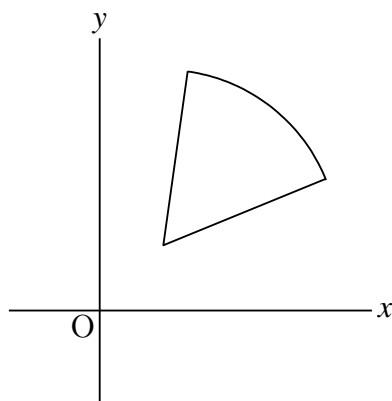
で指定し，扇型ができたのを確かめてから，`Listplot([B,A,C])` を付加すればよい。



辺 AB が x 軸と平行でない場合は、角の範囲は 0 からではなく、AB が x 軸となす角から始める必要があり、ちょっとした工夫が必要である。中心 A が原点でない場合も含め、次のようなスクリプトで実現できる。

```
th=arctan2(B.xy-A.xy);
str="Rng=["+text(th+0)+", "+text(th+pi/3)+"]";
C.xy=A.xy+|A,B|*[cos(th+pi/3),sin(th+pi/3)];
Circledata([A,B],[str]);
Listplot([B,A,C]);
```

1 行目は、AB が x 軸となす角を `arctan2` 関数 によって求めている。
2 行目は引数の文字列を作っている。`th+0` , `th+pi/3` により弧度法にしている。ここで、`+0` が必要である。



複数のオプションをリストで与えることもできる。

例：弧を太く描く

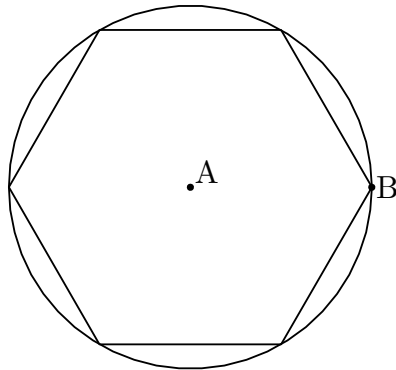
```
Circledata([C,D],["dr,3","Rng=[0,pi/3]"]);
```

円は N が大きな値の正 N 多角形として描いている。option の `["Num=数値"]` に

よってその細かさを指定できる。N の値が小さければ正多角形が描けることになる。

例：A 中心，半径 AB の円と，その円に内接する正六角形

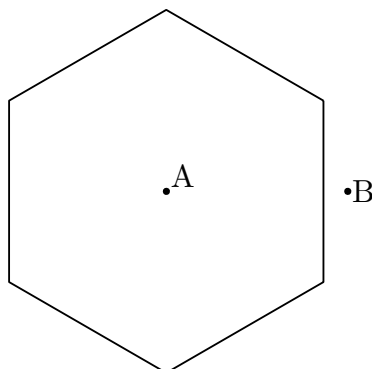
```
Circledata("1",[A,B]);  
Circledata("2",[A,B],["Num=6"]);
```



ここで，同じ [A,B] を使うため，name を付与して区別する必要がある。

また，頂点の位置を変えるのであれば，Rng= オプションを使う。

```
Circledata("2",[A,B],["Num=6","Rng=[pi/6,13/6*pi]"]);
```



[⇒ 関数一覧](#)

関数	Crosspoint(name , PD1 ,PD2 , 範囲)
機能	2 曲線の交点を作る
説明	曲線 1 と曲線 2 の範囲にある交点を作る。曲線 1 と曲線 2 はプロットデータの名称。 範囲は，交点が存在する範囲を指定する。

例：3 次曲線の接線がその曲線と交わる点を求める。

2 点 A,B を作図ツールで適当なところにとり，次のスクリプトで 3 次曲線を描く。

```

f(x):=x^3-4*x;
g(x):=d(f(#),A.x)*(x-A.x)+A.y;
A.y=f(A.x);
B.y=g(B.x);
Plotdata("1","f(x)","x");
Lineplot([A,B]);

```

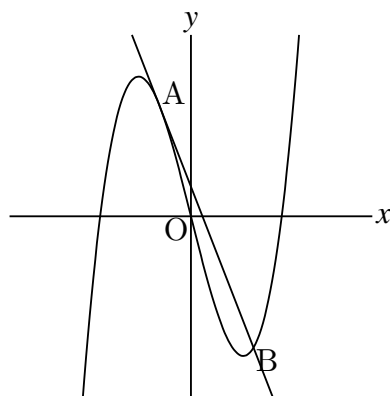
1行目と2行目で3次曲線とその接線を定義し、点A,Bをその上に乗せている。
 Plotdata() と Lineplot() により、3次曲線と接線のプロットデータができる。
 それぞれの名称はコンソールに出力される。3次曲線は gr1 , 接線は lnAB である。
 また、図を見て、交点のある場所を確認して範囲を決める。
 プロットデータ名と範囲を用いて次のスクリプトを追加する。

```

Crosspoint("C",gr1,lnAB,[1,2]);

```

これで、交点Cが新たに作られる。
 必要に応じ、Letter 関数で点の名前を表示すると次のようになる。



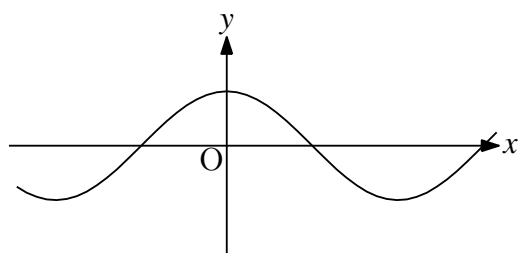
交点が存在しない場合、PD1 の端点に点が作られる。
 2曲線の交点を作るのに、[Putintersect\(点名,PD1,PD2\)](#) 関数を用いることもできる。

[⇒ 関数一覧](#)

関数	Deqplot(name, 式, 変数名, 初期値, options)
機能	微分方程式の解曲線を描く
説明	微分方程式と初期値を与えて解曲線を描く。

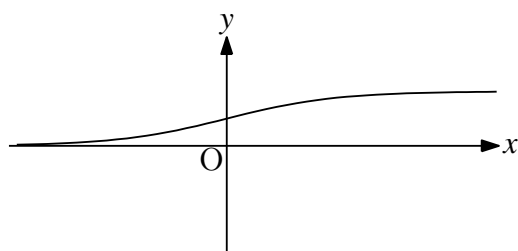
例： $y'' = -y$ で、初期値が $x = 0$ のとき $y = 1, y' = 0$ の解曲線

```
Deqplot("1","y'=-y","x",0,[1,0]);
```



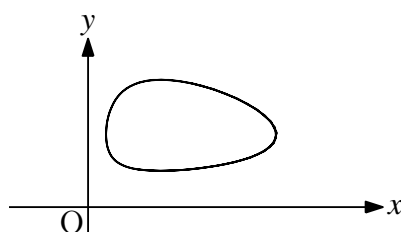
例： $y' = y * (1 - y)$ で、 $x = 0$ のとき、 $y = 0.5$ の解曲線

```
Deqplot("2","y'=y*(1-y)","x",0,0.5,["Num=100"]);
```



例： $[x, y]' = [x(1 - y), 0.3y(x - 1)]$ で、変数は t 、 $t = 0$ （区間の左端）のときの x, y の値が 1 と 0.5 であるときの解曲線

```
Deqplot("3","[x,y]'=[x*(1-y),0.3*y*(x-1)]","t=[0,20]",
[1,0.5],["Num=200"]);
```



なお、この例では、Cinderella の画面上の図は誤差が大きくて正確な図にはならないが、 \TeX に出力する図は正確な図になる。解像度を上げる (Num の数を大きくする) ことにより、Cinderella の画面上の図も正確な図に近づく。

[⇒ 関数一覧](#)

関数	Drwpt(点,option) , Drawpoint(点,options)
機能	点を表示する
説明	座標または幾何点の識別名を与えて点を表示する。これだけでは Cinderella の描画

面には描かれないので、描画面にも表示するには Cinderella の作図ツールで作図するか、Pointdata() または Putpoint() を用いる。

複数の点の場合は座標または識別名はリストで与える。

option に数字 0 を入れると、白抜きで表示する。なお白抜きの場合は、Psize() で点の大きさを少し大きめにとるとよい。

可読性を高めるときは Drawpoint を推奨する。

例：座標 (1,1) と (4,3) に点を表示する。Cinderella の描画面には描かれない。

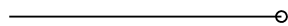
```
Drwpt([[1,1],[4,3]]);
```

例：Cinderella で点 A,B,C を作図しておき、T_EX で表示する。

```
Drwpt([A,B,C]);
```

例：線分 AB の右端 (B) を白抜きで表示する

```
Psize(5);  
Listplot([A,B]);  
Drawpoint(B,0);
```



※ Drawpoint([A,B],0); とすれば、両端が白抜きになる。

[⇒ 関数一覧](#)

関数	Drawsegmark(name, リスト,options) または Segmark(name, リスト,options)
機能	線分に印をつける
説明	リストで与えられた 2 点を端点とする線分に印をつける。印には 4 種類がある。 options は,

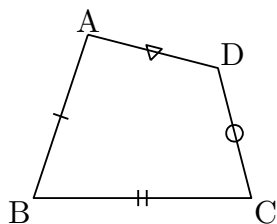
Type=n : n=1~4 : 印の種類

Width : 二本線のときの線の幅

例：Listplot() で四角形 ABCD を描き線分に印をつけた。

```
Segmark("1",[A,B],["Type=1"]);  
Segmark("2",[B,C],["Type=2","Width=1.5"]);
```

```
Segmark("3",[C,D],["Type=3"]);
Segmark("4",[D,A],["Type=4"]);
```



⇒ 関数一覧

関数 Expr([座標, 位置, 文字列])

機能 T_EX 記法の文字列を与えて数式を書く。

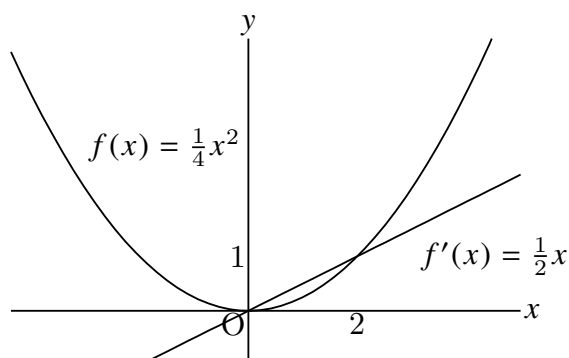
説明 Letter で文字列の前後に \$ \$ をおくのと同じ。

導関数の記号 (シングルクウォート) ' は, Scilab でのシングルクウォートの使用とぶつかるので, ' (バッククウォート) を用いる。

複数の箇所に文字を書く場合は, Letter() と同様, 引数をリストにして与える。

例 $f(x) = \frac{1}{4}x^2$ とその導関数 $f'(x) = \frac{1}{2}x$ の式, 軸上に必要な数を入れる。

```
Expr([-3,3], "e", "f(x)=\frac{1}{4} x^2", [3,1.5], "s2e2",
    "f '(x)=\frac{1}{2}x", [2,0], "s", "2", [0,1], "w", "1"]);
```



※原点 O が線と重なっている。位置をずらすには, Setax() の項を参照のこと。

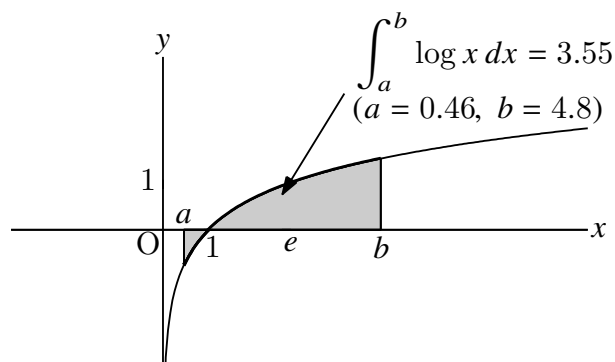
例 対数関数の定積分の記号および積分値を図に書き込む。

下図で, 定積分の表示の部分は矢線を PQ として,

```
Arrowdata(Q,P);
Expr([Q+[0.2,0], "ne", "\displaystyle \int_a^b \log x\,dx="+
    text(L.x*(log(L.x)-1)-G.x*(log(G.x)-1)) ]]);
```

で表示している。

$L.x*(\log(L.x)-1)-G.x*(\log(G.x)-1)$ は、点 L,G(図の a, b) をドラッグして積分範囲を決めるようにしているので、そこから計算した値。



関数 Exprrot([座標, 向き, 文字列])

機能 T_EX 記法の文字列を与えて傾いた数式を書く。

説明 「座標」の位置に、指定された向きで数式を書く。

向きはベクトルで与える。

座標, 向きとも, Cinderella で作図した幾何点を用いることができる。

A(1,1), B(3,2), C(0,2) のとき, 次の2つのスクリプトは同じ結果になる。

```
Exprrot(C, B-A, "\sqrt{3}");
Exprrot([0,2], [2,1], "\sqrt{3}");
```

[⇒ 関数一覧](#)

関数 Ellipseplot(name, 点リスト, 定義域, options)

機能 焦点と通る点を与えて楕円を描く。

説明 点リストで2つの焦点と通る点を与える。点は Cinderella の幾何点が使える。

また, 通る点のかわりに, 焦点からの距離の和を実数で与えることもできる。

実際には, 媒介変数表示 $x = a \cos \theta, y = b \sin \theta$ を, 回転・平行移動して描いている。

定義域はこのときの t の定義域で, 省略も可能。省略したときの初期値は [-5,5]

例: 点 A,B を焦点とする楕円を描く。

```
Ellipseplot("1", [A,B,C]);
```

 点 C を通る楕円を描く。

```
Ellipseplot("1", [A,B,4]);
```

 焦点からの距離の和が4である楕円を描く。

```
Ellipseplot("1", [A,B,C], "[0,pi]");
```

 楕円の半分を描く。

例: Cinderella の作図ツールに, 焦点と通る点で楕円を描くものがある。また, 点の

極線を描くツールがある。これを利用すると、楕円上にとった点をインシデントにできるので、インタラクティブに図を変更することができる。この Cinderella の作図機能と合わせて、一方の焦点から出た光が楕円上で反射して他方の焦点に至る、という図を次のようにして描くことができる。

まず、3つの点 A,B,C を作図する。次に「焦点と通る点で決まる楕円」ツールを選び、点 A,B,C を順に指定すると、楕円が描かれる。

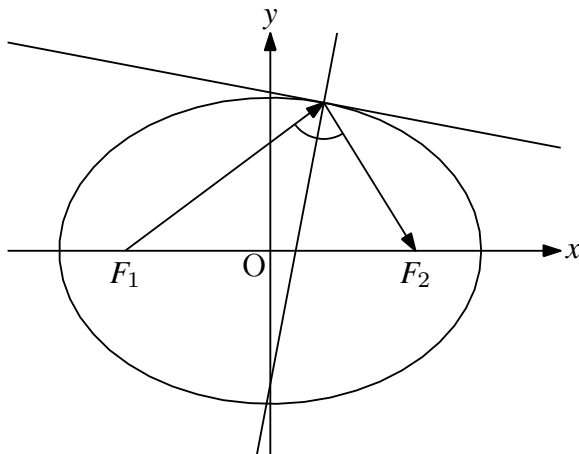
モードメニューの「直線」から「点の極線」を選び、点 C と楕円を順に指定すると接線が引かれる。

「垂線を加える」ツールを用いて、点 C で垂線、すなわち法線を引く。

「点を加える」ツールを用いて、接線、法線上に適当に点を取る。(D,E となったとする)

次のスクリプトを書いて実行すると、楕円に関して入射角と反射角が等しくなるように光が反射する様子を図にすることができる。

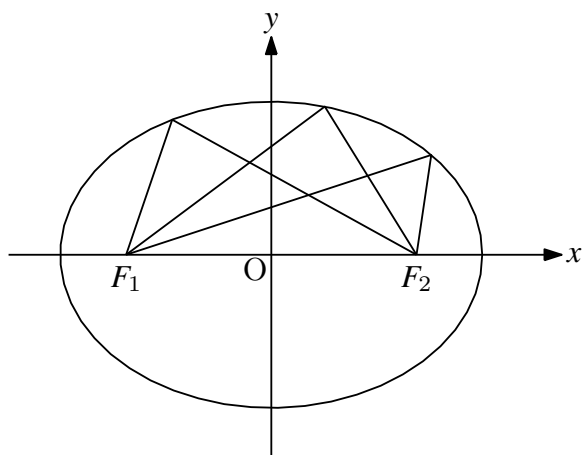
```
Ellipseplot("1",[A,B,C]);
Lineplot([C,D]);
Lineplot([C,E]);
Arrowdata([A,C]);
Arrowdata([C,B]);
Anglemark([A,C,B]);
Expr([A,"s2","F_1",B,"s2","F_2"]);
```



また、接線、法線を描かず、この楕円上に点 D,E,・・・をとり（個数は任意）次のスクリプトを書けば、何本かの光線が一方の焦点を出て他方の焦点に集まる様子を描くことができる。

```
Ellipseplot("1",[A,B,C]);
```

```
Listplot([A,C,B]);
Listplot([A,D,B]);
Listplot([A,E,B]);
Expr([A,"s2","F_1",B,"s2","F_2"]);
```



関数 Framedata(name , リスト)

関数 Framedata2(name ,[P1,P2])

機能 矩形を描く

説明 中心の座標, 横, 縦をリスト [中心 , 横 , 縦] で与え, 矩形を描く。横, 縦は中心からの距離。

中心の座標は点の名前でもよい。

中心を座標で与える場合は name は省略できない。

横, 縦は, 矩形の頂点を示す点にすることもできる。

中心, 横, 縦を省略した場合は, 描画範囲と同一の矩形を描く

Framedata2() では, 対角点 (左下 P1 と右上 P2) をリストで与える。

以下にいくつか例を示す

Framedata("1"); 描画範囲と同一の矩形を描く

Framedata("2",[0,0],2,2); 原点を中心とする縦横幅4の正方形を描く

Framedata("3",[A,3,2]); 点 A を中心とする横6, 縦4の矩形を描く

Framedata([A,3,2]); 中心が点の名称の場合は name は省略できる。

Framedata([A,B]); 点 A を中心, 点 B を頂点のひとつとする矩形を描く

矩形の角を丸めたい場合は, Ovaldata(name, 点リスト,options) を使う。

Framedata2("1",[A,B]); 点 A, B を対角点とする矩形を描く

関数 Hyperbolaplot(name, 点リスト , 定義域, options)

機能 焦点と通る点を与えて双曲線を描く。

説明 点リストで2つの焦点と通る点を与える。点は Cinderella の幾何点が使える。また、通る点のかわりに、焦点からの距離の差を実数で与えることもできる。

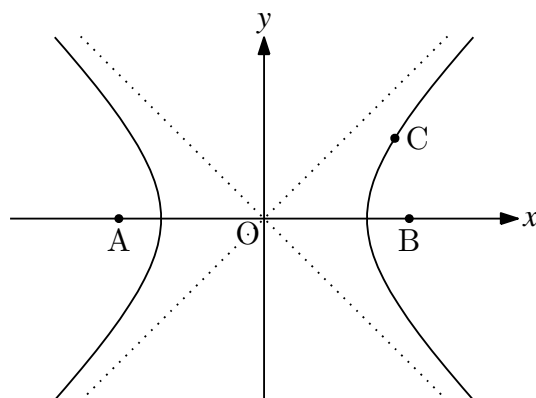
実際には、ハイパボリック関数を用いた媒介変数表示 $x = \cosh t, y = \sinh t$ を回転・平行移動している。option として、"Asy=線種" を与えると、漸近線を指定した線種で表示する。デフォルトでは漸近線は非表示。

例：点 A,B を焦点とする双曲線を描く。

`Hyperbolaplot("1",[A,B,C]);` 点 C を通る双曲線を描く。

`Hyperbolaplot("1",[A,B,2]);` 焦点からの距離の差が2の双曲線を描く。

`Hyperbolaplot("1",[A,B,C],["Asy=do"]);` 漸近線を点線で描く。



関数 `Parabolaplot(name, 点リスト, 定義域, options)`

機能 点リスト [A,B,C] で示された焦点, 準線で決まる放物線を描く。

説明 焦点 A と準線 BC で決定する放物線を描く。

実際には、2次関数 $y = x^2$ のグラフを回転・平行移動して描いており、定義域は、 $y = x^2$ での定義域と考えてよい。定義域は省略することもできる。省略したときの初期値は [-5,5]

例：点 A を焦点, 直線 BC を準線とする放物線を描く

`Parabolaplot("1",[A,B,C]);`

`Parabolaplot("1",[A,B,C],["-5,5"]);` 定義域を $-5 \leq x \leq 5$ とする。

点 (0,1) を焦点, 直線 $y = -1$ を準線とする放物線を描く

`Parabolaplot("1",[[0,1],[-1,-1],[1,-1]]);`

例：放物線上の2点で引かれた接線と放物線で囲まれた領域を斜線で描く。

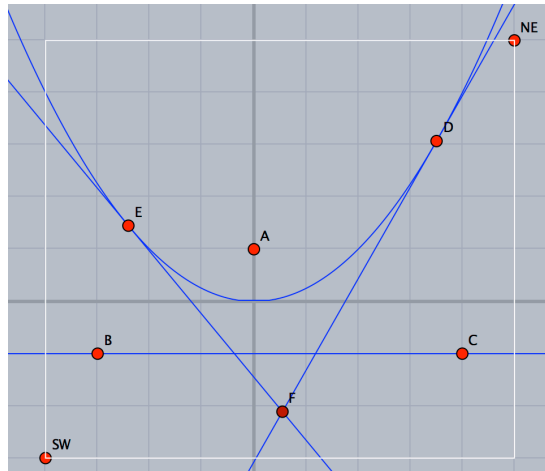
Cinderella の作図ツールに、焦点と準線で放物線を描くものがある。また、点の極線を描くツールがある。これを利用すると、放物線上にとった点をインシデントにできるので、インタラクティブに図を変更することができる。この Cinderella の作図機

能と合わせて、次の手順で図を描く。

まず、焦点 $A(0,1)$ と準線 $y = 1 : BC$ を作図する。次に「焦点と準線で決まる放物線」ツールを選び、点 A と直線 BC を指定すると、放物線が描かれる。方程式では $y = \frac{1}{4}x^2$ の放物線である。

次に、放物線上に点 D, E をとる。Cinderella の作図機能を用いているので、この2点は放物線上だけを動かすことができる。(インシデント)

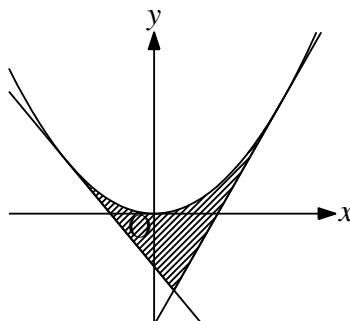
モードメニューの「直線」から「点の極線」を選び、点 D と放物線、点 E と放物線を順に指定すると接線が引かれる。その交点に点を取る。



以上で作図ができたので、次のスクリプトを書いて実行する。

```
Parabolaplot("1",[A,B,C]);  
Lineplot([D,F]);  
Lineplot([E,F]);  
Listplot([E,F,D]);  
Hatchdata("1",["ii"],[["gr1para","s"],["sgEFD","n"]]);
```

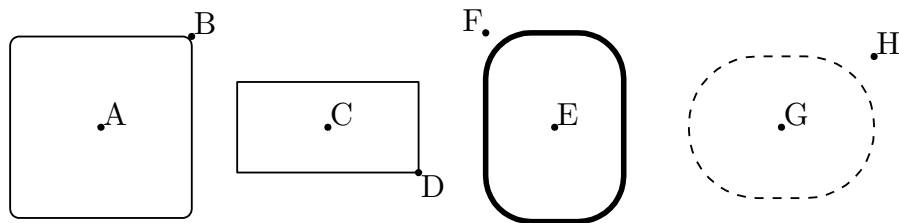
これで、次図ができる。このあと、文字などは適当に追加する。



関数 Ovaldata(name, 点リスト,options)
機能 角を丸くした矩形を描く
説明 中心と対角の1点を指定し、角を丸くした矩形を描く
 options は、角の落とし具合と線種など。デフォルトは 0.2

例：いくつかの例を示す。

```
Ovaldata("1", [A,B]);  
Ovaldata("2", [C,D],[0]);  
Ovaldata("3", [E,F],[1,"dr,3"]);  
Ovaldata("4", [G,H],[1.5,"da"]);
```



関数 Htickmark([横座標, 方向, 文字])
機能 横軸に目盛を書く。
説明 Scilab のみで実行する。Cinderella の描画面には反映されない。引数は位置（横座標）、方向、文字。複数点の情報を [] 内にまとめて記入できる。

例 点 (2, 0) の南西側に 2 を表示する。

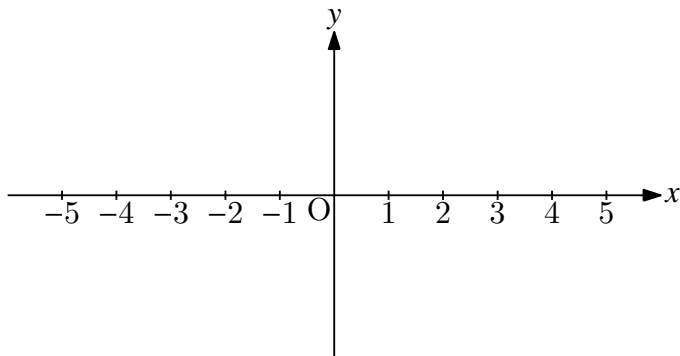
```
Htickmark([2,"sw","2"]);
```

例 -5 から 5 までの目盛を打つ。

Cindyscript のリスト処理を使って、次のように引数のリストを作って渡す。

```
memori=apply(-5..5,x,[x,"s",text(x)]);  
memori=flatten(remove(memori,[[0,"s","0"])]);  
Htickmark(memori);
```

1 行目, apply のカッコ内の -5..5 でリスト [-5,-4,-3,-2,-1,0,1,2,3,4,5] ができる。
 それを用いて, apply で [数, "s", 数の文字] からなるリストができる。text(x) は x
 を文字にする関数。
 2 行目で, このリストから, [0,"s","0"] を除き, リストを平滑化する。
 結果は次のようになる。



関数 Vtickmark([横座標, 方向, 文字])
機能 縦軸に目盛を書く。
説明 Htickmark と同様。縦軸に目盛を書く。

例：点 (0, 1), (0, 2) の西側に 1, 2 を表示する。

```
Htickmark([1,"w","1",2,"w","2"]);
```

[⇒ 関数一覧](#)

関数 Implicitplot(name, 式, x の定義域, y の定義域, options)
機能 陰関数のグラフを描く。
説明 陰関数の式を与えてグラフを描く。式, 定義域とも文字列。
 options は, "r","m","Wait=n" が指定できる。Wait の初期値は 10。
 "r","m" に関しては, オプションなしまたは, "" のとき
 i) データファイルがなければ, 新しく作る
 ii) データファイルが既にあればそれを読み込む
 "m" のとき, 強制的にデータファイルを作り直す。
 "r" のとき, すでにあるデータファイルを読み込む。

例：楕円を描いて, 中にハッチをかける。

```
Implicitplot("1","x^2+2*y^2=4","x=[-2,2]","y=[-2,2]");  
Hatchdata("1","i",["imp1"]);
```

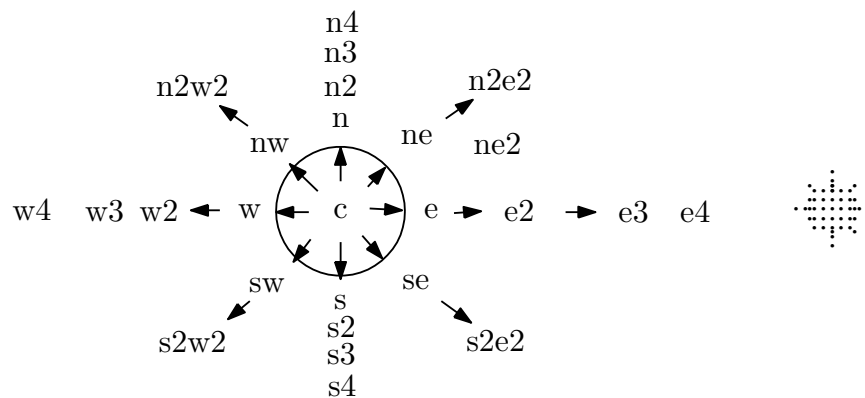
注) ここで, y の定義域は実際より広くとってある。” $y=[-1,1]$ ” とすると, 上下が欠けた楕円になる。曲線だけならそれでよいが, その場合, 閉曲線ではないので Hatchdata() で無用に時間がかかってしまうので要注意。

関数 Letter([位置, 方向, 文字列])

機能 文字列を表示する

説明 「位置 (座標)」と方向で指定された場所に文字を書き込む。

位置 (座標) は点の名前で指定することもできる。場所は上下左右中央 (n/s/w/e/c) の方向で表す。(n/s/w/e は東西南北の記号)



指定位置からの距離を, 数値で与えることもでき, e2, e3 は e より少し離して置く。複数の文字列をリストの形にして渡すことができる。

注) 導関数の記号' は, 数式モード (\$ ではさむ) で' (バッククウォート) を用いる。

例: 座標 (2,1) の南東に P を表示

```
Letter([2,1] ,"se","P");
```

点 C を中央として C を表示

```
Letter(C ,"c", "C");
```

点 A の南西に A, E の南に数式を表示

```
Letter([A,"sw","A",E,"s","$ f(x)=\frac{1}{4} x^2 $"]);
```

関数 Letterrot(座標, 方向ベクトル, 移動量, 文字列)

機能 文字列を回転して表示する

説明 座標で示された位置に, 方向ベクトルで指定された向きに回転して文字を書き込む。

第 3 引数は微小移動量で, 略することもできる。

A(1,1),B(3,2),C(0,2) のとき, 次のスクリプトは同じ結果になる。

```
Letterrot([0,2],[2,1],2,5,"AB");
Letterrot([0,2],B-A,2,5,"AB");
Letterrot(C,B-A,2,5,"AB");
Letterrot(C,B-A,"t2n5","AB");
```

移動量を略して

```
Letterrot(C,B-A,"AB");
```

とすることもできる。この場合は、微小な移動はされない。

[⇒ 関数一覧](#)

関数 Lineplot (name , 2 点のリスト , options)
機能 2 点のリストで示された点を結ぶ直線を描く。プロットデータ名の頭部は ln
説明 2 点のリストは座標または幾何要素の名前で与える。

options は次の通り。

線種 "dr, n" , "da,m,n" , "do,m,n"

"+" 半直線を描く。

"dr" , "da" , "do" と "+" はリストにして両方指定することができる。

点のリストが、座標ではなく幾何要素名のリストの場合は、name は省略できる。

いくつか例を示す。

各座標を結ぶ直線を引く

```
Lineplot("1",[[0,0],[1,2]])
```

Cinderella の描画ツールで 2 点 A,B をとっておき、直線 AB を引く

```
Lineplot([A,B]);
```

option の働きの例

```
Lineplot([A,B],["dr,0.5","+"]);
```

A を端点とする半直線を描く

```
Lineplot([C,D],["dr,2"]);
```

直線 CD を太さ 2 で描く

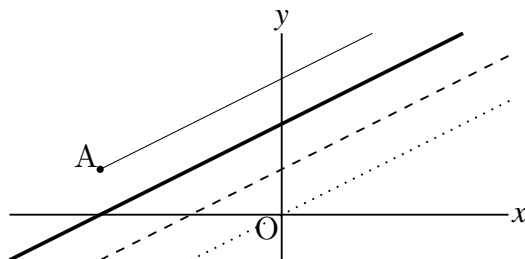
```
Lineplot([E,F],["da"]);
```

直線 EF を破線で描く

```
Lineplot([G,H],["do"]);
```

直線 GH を点線で描く

結果は、次図左上から。



⇒ 関数一覧

関数 Listplot (name , 点のリスト , options)

機能 点のリストで示された点を結ぶ。プロットデータ名の頭部は sg

説明 点のリストは座標または幾何要素名のリストで与える。点が、座標ではなく幾何要素名の場合は、name は省略可

プロットデータの名前は、"sg" に引数の name を付加したものとなる。

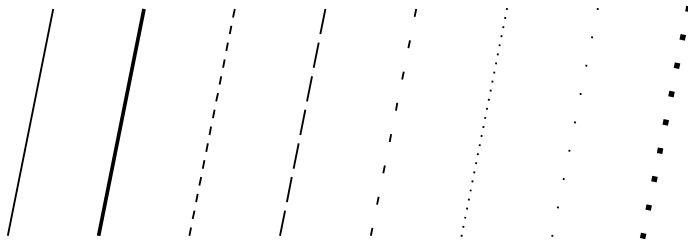
options は次の通り。

線種 "dr, n" , "da,m,n" , "do,m,n"

options の使用例

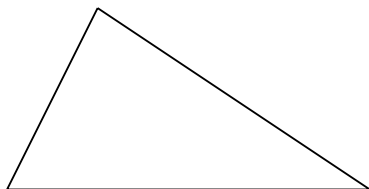
Listplot([A,B]);	線分 AB を描く。太さはデフォルト。
Listplot([C,D],["dr,2"]);	線分 CD を描く。太さ 2
Listplot([E,F],["da"]);	線分 EF を破線で描く
Listplot([G,H],["da,3,1"]);	線分 GH を破線で描く。線を長く
Listplot([K,L],["da,1,3"]);	線分 KL を破線で描く。間隔を空ける
Listplot([M,N],["do"]);	線分 MN を点線で描く。
Listplot([O,P],["do,3"]);	線分 OP を点線で描く。間隔を空ける
Listplot([Q,R],["do,3,3"]);	線分 QR を点線で描く。間隔を空けて太く

結果は次図左から。



例：Cinderella の作図ツールで三角形 ABC を描いておく。

```
Listplot([A,B,C,A]);
Addax(0);
```



各座標を頂点とする三角形を描く

```
Listplot("1",[[0,0],[2,0],[1,2],[0,0]]);
```

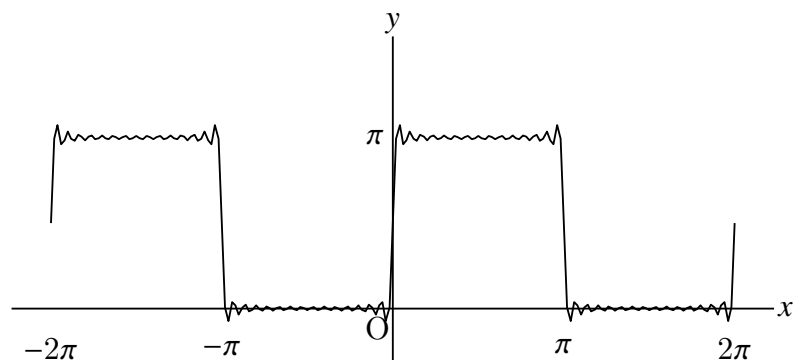

プロットデータは点の座標のリストである。したがって、プロットデータを自作して Listplot() で表示することができる。

例：有限フーリエ級数展開

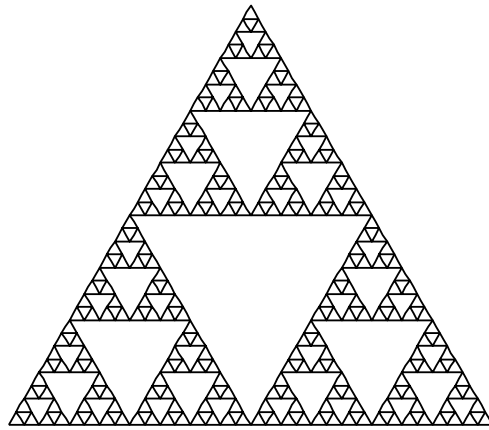
$$\frac{\pi}{2} + \sum_{n=0}^{30} \frac{1 - (-1)^n}{n} \sin nx$$

次のように Cindyscript で関数を定義し、プロットデータ pd を作って引数に渡す。

```
f(x):=(
  s=pi/2;
  repeat(30,n,s=s+(1-(-1)^n)/n*sin(n*x));
);
pd=apply(0..200,t,
  x=-2*pi+t*4*pi/200;
  [x,f(x)]);
);
Listplot("1",pd);
Expr([[ -2*pi,-0.5],"s",-2\pi],[-pi,-0.5],"s",-\pi],[pi,-0.5],"s",
  "\pi",[2*pi,-0.5],"s",2\pi],[0,pi],"w2","\pi"]);
```



リストの長さには制限があるため、あまり長いリストを用いたり、何度も用いたりすることができない。たとえば、タートルグラフィックスを用いたシェルピンスキーのギャスケットでは、次のサイズくらいは可能だが、植物の成長モデルでは分岐が多いためあまり大きな図はできない。スクリプトを工夫して、200 くらいずつのリストに分割する。



関数 Mkssegments()

機能 幾何線分のすべての PD を作成

説明 Ketinit() の直後に置くことにより, Cinderella の作図ツールで描いたすべての線分をそのままプロットデータとする。たとえば, 線分 AB を作ると, プロットデータ sgAB が作成される。その後, インспекタで点 B の識別名を変更 (たとえば Q に) すると, プロットデータ名も変更される。線分はすでに描かれていてもよい。

関数 Mkcircles()

機能 幾何円のすべての PD を作成

説明 Ketinit() の直後に置くことにより, Cinderella の作図ツールの「円を加える」で描いたすべての円をそのままプロットデータとする。たとえば, 中心 A, 円周上の点を B とした円を作ると, プロットデータ crAB が作成される。その後, インспекタで点 B の識別名を変更 (たとえば Q に) すると, プロットデータ名も変更される。円はすでに描かれていてもよい。

[⇒ 関数一覧](#)

関数 Mkbeziercrv(名前,[[節点リスト,制御点リスト],[節点リスト,制御点リスト],...],
[オプション])

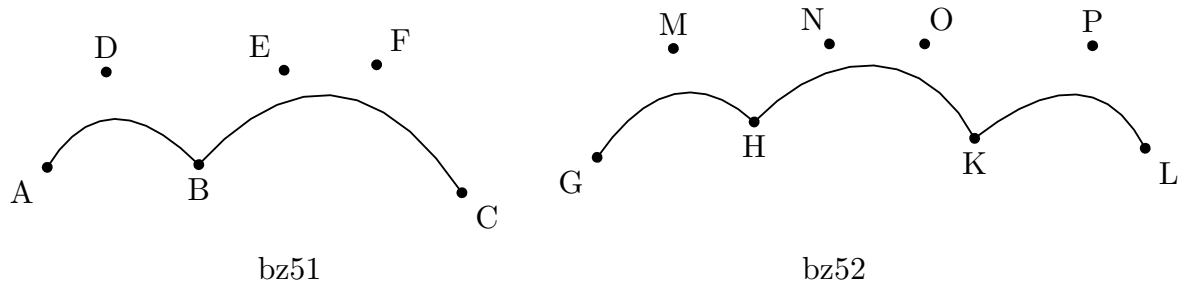
機能 複数のベジエ曲線を描く

説明 単独のベジエ曲線の場合はリストの外側の [] はなくてもよい。

Mkbeziercrv—(名前, [節点リスト, 制御点リスト], [オプション]) は
Bezier(名前, 節点リスト, 制御点リスト, [オプション]) と同じ。

例：Mkbeziercrv("n",[A,B,C],[[D],[E,F]]); は
 Bezier("n",[A,B,C],[[D],[E,F]]); と同じ。
 曲線の名前が bzn ではなく bzn1 となる。

複数の場合は, [[ptlist1,ctrlist1], [ptlist2,ctrlist2],...]
 Mkbeziercrv("5",[[A,B,C],[[D],[E,F]]],[[G,H,K,L],[[M],[N,O],[P]]]);



```
ptlist1=[A,B,C];ctrlist1=[[D],[E,F]];list1=[ptlist1,ctrlist1];
ptlist2=[G,H,K,L];ctrlist2=[[M],[N,O],[P]];list2=[ptlist2,ctrlist2];
list=[list1,list2];
Mkbeziercrv("5",list);
```

などとしても同じ。

[⇒ 関数一覧](#)

関数 Mkbezierptcrv(節点リスト ptlist, [オプション])

機能 ベジエ曲線を描く

説明 制御点は、自動的に配置される。その後、節点や制御点を動かして、描きたいものにする。

複数の場合は [ptlist1, ptlist2....]

名前は、A から順に自動的につける。

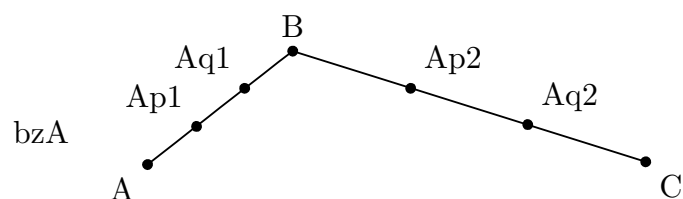
オプション

"Deg=..." 次数指定ができる。(デフォルトは 3 次)

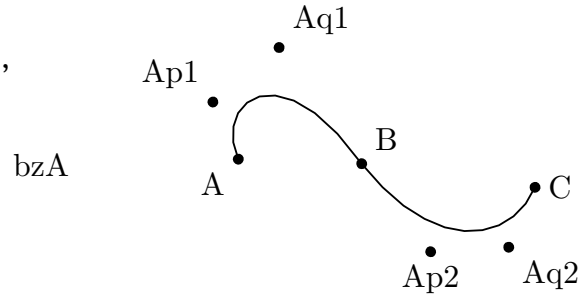
"Num=..." 各区間の区間数 (分点数-1) を指定できる。(デフォルトは 10)

例：

Mkbezierptcrv([A,B,C]);



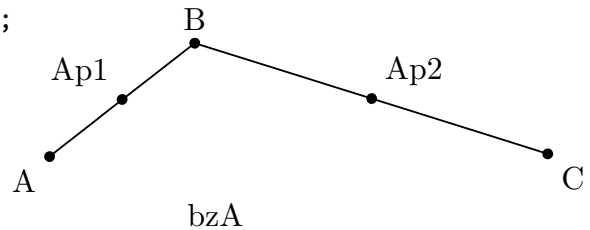
その後、節点や制御点を動かして、
描きたいものにする。



`Mkbezierptcrv([A,B,C],["Deg=2"]);`

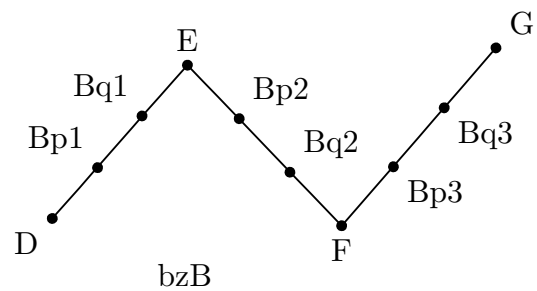
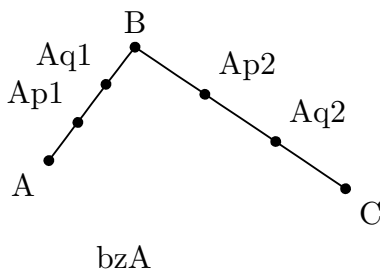
Deg=2 とすると 2 次になる。

制御点は各区間に 1 個ずつできる。



複数の場合は [ptlist1, ptlist2....]

`Mkbezierptcrv([[A,B,C], [D,E,F,G]]);`



[⇒ 関数一覧](#)

関数 Paramark(点リスト , options)

機能 点リスト [A,B,C] で示された角に平行四辺形の形状の角の印をつける。

説明 options は次の通り。

数値 角の印の大きさ。デフォルトは 1

線種 "dr, n" , "da,m,n" , "do,m,n"

"Expr=文字" : 文字を入れる

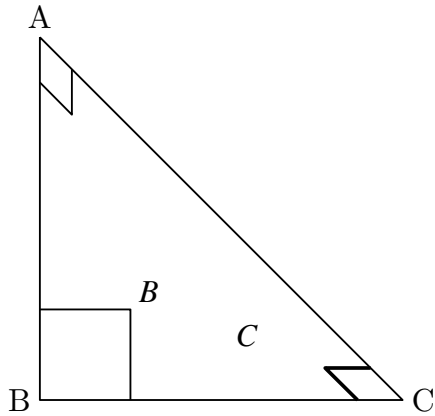
"Expr=位置 , 文字" : 位置を指定して文字を入れる。位置は頂点からの距離。

例：三角形の内角に印をいれ、文字を書き込む。

`Paramark([B,A,C]);`

`Paramark([C,B,A],[3,"Expr=B"]);`

```
Paramark([A,C,B],["dr,2","Expr=2,C"]);
```



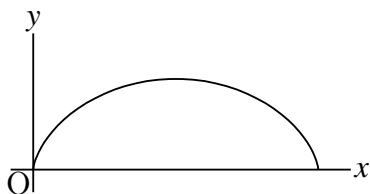
※角の印には弧の形状のものもある。Anglemark() を参照のこと。

[⇒ 関数一覧](#)

関数 Paramplot(name , 式 , 変数と定義域,options)
機能 媒介変数表示の曲線を描く。プロットデータの頭部は gp
説明 式は""でくくった媒介変数表示のリストで与える。
 定義域も "" でくくって文字列とし、t=に続いてリストで指定する。
 options は線種が有効

例：サイクロイド曲線を描く。

```
Paramplot("1",["t-sin(t),1-cos(t)","t=[0,2*pi]");
```

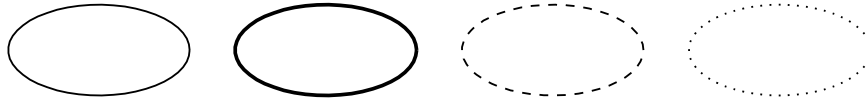


```
Deffun("fx(t)", ["regional(x)","x=t-sin(t)","x"]);  
Deffun("fy(t)", ["regional(y)","y=1-cos(t)","y"]);  
Paramplot("1",["fx(t),fy(t)","t=[0,2*pi]");
```

とすれば、関数定義も Scilab に引き継がれる。

options の使用例。左から、デフォルト、太線、破線、点線の楕円

```
Paramplot("1","[2*cos(t),sin(t)]","t=[0,2*pi]");  
Paramplot("2","[2*cos(t)+5,sin(t)]","t=[0,2*pi]","dr,2");  
Paramplot("3","[2*cos(t),sin(t)+3]","t=[0,2*pi]","da");  
Paramplot("4","[2*cos(t)+5,sin(t)+3]","t=[0,2*pi]","do");
```



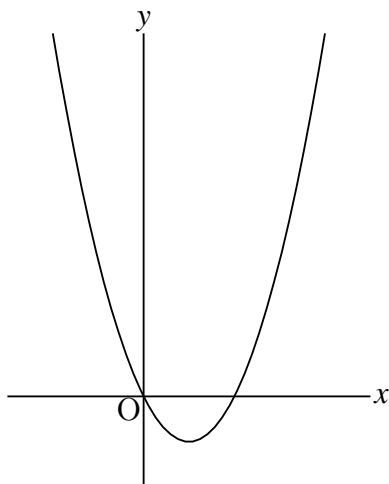
[⇒ 関数一覧](#)

関数 Plotdata(name , 式 , 変数と定義域 , options)
機能 関数のグラフを描く。プロットデータの名前は, gr
説明 式で表された関数のグラフを, 指定された定義域で描く。
式, 定義域は " " でくくって文字列とする。定義域は x=に続いてリストで指定。
options は次の通り。

線種	"dr, n" , "da,m,n" , "do,m,n"
"Num=数値"	描画時の分割数
"Dis=数値"	値が指定数値以上ジャンプする場合は不連続点とみなす。
"Exc=数値リスト	リストで示された点は除外する。
"Exc=関数"	関数の零点は除外する。

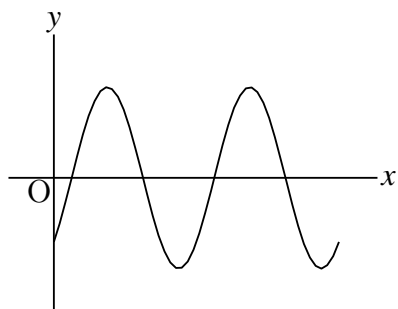
例：2 次関数 $f(x) = x^2 - 2x$ のグラフを定義域指定なしで描く。

```
Plotdata("1","x^2-2*x","x");
```



例：三角関数 $2\sin\left(2x - \frac{\pi}{4}\right)$ のグラフを，定義域 $0 \leq x \leq 2\pi$ で描く。

```
Plotdata("3","2*sin(2*x-pi/4)","x=[0,2*pi]");
```



CindyScript では，`plot(式 , 定義域);` で描くが，`KETCindy` を用いるときは，CindyScript の `plot` 関数のかわりに，この `Plotdata` を使えばよい。

軸に数字を入れるのであれば，`Letter()` を用いる。

options の使用例

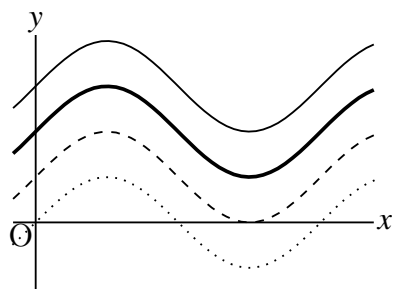
```
Plotdata("1","sin(x)","x");    f(x)=sinx のグラフを描く（デフォルト）
```

```
Plotdata("2","sin(x)+1","x",["dr,2"]);    同じく，太さ2で描く
```

```
Plotdata("3","sin(x)+2","x",["da"]);    同じく，破線で描く
```

```
Plotdata("4","sin(x)+3","x",["do"]);    同じく，点線で描く
```

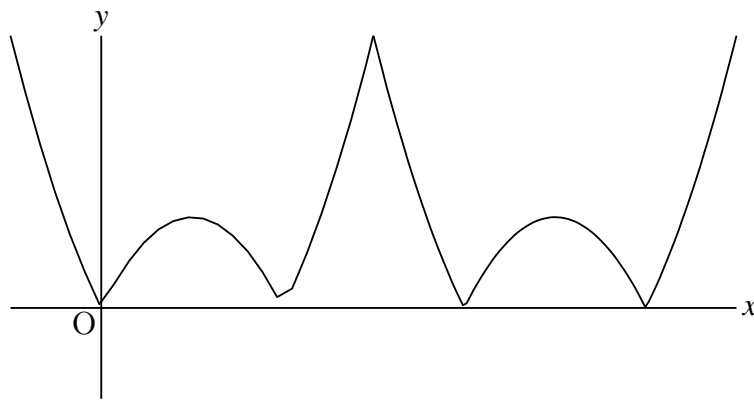
結果は次図上から。



Num=分割数の指定

グラフの描画は，区間を分割して関数値をとり，各点を結ぶという通常の方法によって行っている。N の指定はこの分割数の指定である。デフォルトは 50。思うような結果が得られない場合はこの値を大きく指定するとよい。

下図左はデフォルト，右は Num=200 とした。



不連続点の指定

Dis オプションにより、値がジャンプする不連続点を線で結ばないようにする。
Num オプションと合わせて使うと効果が上がる。

例： $f(x) = \tan x$ のグラフは、そのままではあたかも漸近線が描かれたようになるが、これは、不連続点をそのまま結んでいるためである。

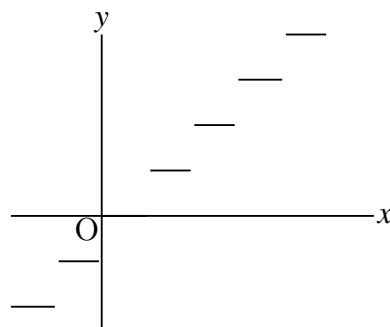
```
Plotdata("1","tan(x)","x",["Num=200","Dis=50"]);
```

で漸近線が描かれなくなる。

例：ガウス記号 $[x]$ で表される関数のグラフは、不連続な階段状になる。この関数は天井関数 `floor()` であるので、

```
Plotdata("1","floor(x)","x",["Num=100","Dis=0.9"]);
```

できれいに描ける。



関数に文字係数がついており、文字係数の値を変化させながらグラフを描くには、Assign を使う。

例：直線 $y = bx - b^2$ の係数 b を変化させて描き、包絡線をうかびあがらせる。

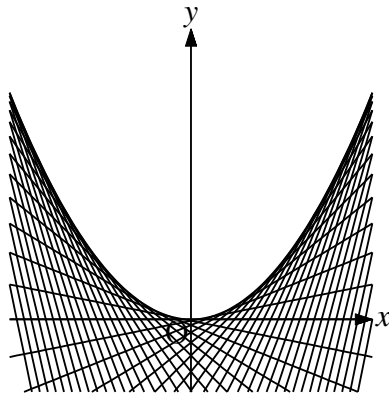
```
repeat(50,t,
```



```

cb=t/5-5;
Plotdata(text(t),Assign("b*x-b\verb|^|^2","b",cb),"x");\
);

```



[⇒ 関数一覧](#)

関数 Pointdata(name , 点リスト , optionss)

機能 点を作って Scilab に点のリスト Pointdata(list) を出力する。

説明 点データを作成する。この点は、幾何要素ではない。プロットデータの頭部は pt
 例：Pointdata("1",[[1,2],[-2,3]]);

2つの点データを作る。Cinderella の描画面では緑の点で表示される。

```
Pointdata("1",[A,B]);
```

作図した点 A,B について、点データを作る。

Cinderella の描画面上では既存の点 A,B に緑の点が重なって表示される。

```
Pointdata("1",A,["size=4"]);
```

A の位置に大きさ 4 で点を作る。Psize(Setpt) の項を参照のこと。

```
Pointdata("1",[A,B],[0]);
```

点データを作り、TeX にオプション 0 (白抜き) で描く

```
Pointdata("1",[[3,4],[5,6]],["notex"]);
```

点データを作るが、TeX には描かない

```
Pointdata("1",[A,B],["nodisp"]);
```

点データを作らない。(戻り値として返す)

生成されるリストは、座標のリストがネストした形になっている。
 その内容は、println() を用いてコンソールに表示することができる。たとえば

```

Pointdata("1",[[1,2],[3,4],[6,2]]);
println(pt1);

```

で、コンソールに次のように表示される。

```
generate pointdata pt1
[[[1,2]], [[3,4]], [[6,2]]]
```

ネストされたリストは、Cindyscript の `flatten()` 関数を利用して平滑化（ネストを解除）することにより、`Listplot()` で利用することができる。

例：`Ptsize(3);`

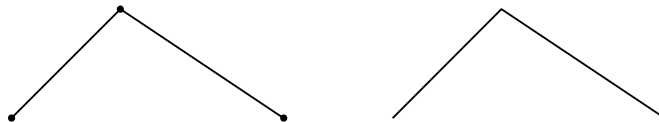
```
Pointdata("1", [[1,2], [3,4], [6,2]]);
```

```
Listplot("1", flatten(pt1));
```

これで、節点を明示した折れ線が描ける。(下図左)

```
Listplot("1", [[1,2], [3,4], [6,2]]);
```

では線分のみが描かれる。(下図右)



[⇒ 関数一覧](#)

関数 `Polygonplot(name, 点リスト, 整数, optionss)`

機能 2点を指定して正多角形を描く。

説明 円に内接する正多角形を描き、頂点の幾何点を作る。

例：`Polygonplot("1", [A,B], 12);`

点 A を中心とし、半径 AB の円周上に、点 B からスタートして正十二角形の頂点 B1～B11 を反時計回りに作り、これを結んだ正多角形を描く。頂点は幾何点。

整数でない数を指定した場合は、きちんと閉じない折れ線が描かれる。

関数 `Putintersect(点名, PD1, PD2, [No])`

機能 2曲線の交点を作る

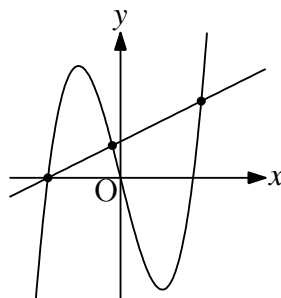
説明 2曲線はプロットデータ名で指定する。指定した点名で交点を作る。この点は幾何点。

交点が2つ以上ある場合でも、描画範囲にそのうち1つだけが存在するなら、その点となる。描画範囲に2つ以上の交点がある場合は、それらのうちいずれかになる。このとき、コンソールに交点の座標のリストと、「Choose point number」というガイドが表示される。引数の No として、その番号を指定すると、その点が採用される。

この関数で作成されるのは幾何点だけなので、 $\text{T}_{\text{E}}\text{X}$ の図に点として明示するためには `Pointdata()` で書き出す必要がある。

次の例は、3次曲線と直線の交点を3つとも取ったものである。

```
Plotdata("1","x^3-4*x","x",["Num=200"]);
Plotdata("2","1/2*x+1","x");
Putintersect("P","gr1","gr2",1);
Putintersect("Q","gr1","gr2",2);
Putintersect("R","gr1","gr2",3);
Pointdata("1",[P,Q,R],["size=4"]);
```



交点が存在しない場合は、「No intersect point」がコンソールに表示される。

2曲線の交点を作るのに、[Crosspoint\(\)](#) 関数を用いることもできる。

[⇒ 関数一覧](#)

関数	<code>Putpoint(点名, 座標1, 座標2)</code>
機能	点を作る
説明	識別名が点名の点を、既存でなければ座標1に作る。既存ならば座標2に移動する。 Tex には出力されない。

例：(1,1) に固定点 A を作る。この点は動かすことができない。

```
Putpoint("A",[1,1]);
```

(1,1) に自由点を作る。

```
Putpoint("A",[1,1],[A.x,A.y]);
```

この点は座標2の効果により、自由点となり、ドラッグして動かすことができる。

実際には、Cinderella の作図ツールで点を取り、インスペクタで名前を変更すればよいわけだが、この形でとればスクリプトの可読性が増す意味がある。

注) 点名は半角アルファベットとする。数字や漢字でも Cinderella では点ができるが、

Scilab でエラーとなる。

関数 PutonLine(点名 , 座標 1 , 座標 2)

機能 直線上に点を作る

説明 座標 1 , 座標 2 を通る直線上に点名の点を作る。できた点は直線に対してインシデントとなる。

例 点 A,B を通る直線上に点 P をとる。

```
PutonSeg("P",A,P);
```

関数 PutonSeg(点名 , 座標 1 , 座標 2)

機能 線分上に点を作る

説明 座標 1 , 座標 2 を通る線分上に点名の点を作る。できた点は線分に対してインシデントとなる。

例 点 A,B を通る線分上に点 P をとる。

```
PutonSeg("P",A,P);
```

[⇒ 関数一覧](#)

関数 Reflectpoint(点, 対称点または対称軸)

機能 点の鏡映を作成

説明 点を指定された点または軸に関して対称移動する。対称軸は [点 1, 点 2] で指定

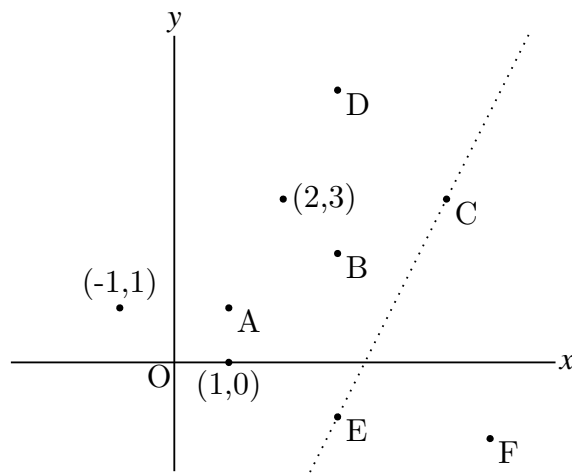
例：C は B に関して A と対称な点

D は点 (2,3) に関して A と対称な点

E は点 (1,0) に関して (-1,1) と対称な点

F は直線 CE に関して A と対称な点

```
C.xy=Reflectpoint(A,B);  
D.xy=Reflectpoint(A,[[2,3]]);  
E.xy=Reflectpoint([-1,1],[[1,0]]);  
F.xy=Reflectpoint(A,[C,E]);  
Lineplot([C,E],["do"]);
```



⇒ 関数一覧

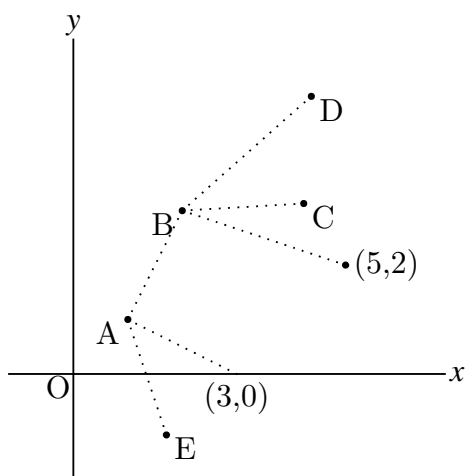
関数 Rotatepoint(点 , 角度 , 中心)
機能 点の位置を回転する
説明 点を, 中心で示された点の周りに回転する。角度は弧度法で与える

例：点 C は A を, B に関して $\frac{2}{3}\pi$ だけ回転した点

点 D は点 (5,2) を, B に関して $\frac{\pi}{3}$ だけ回転した点

点 E は点 (3,0) を A に関して $-\frac{\pi}{4}$ だけ回転した点

```
C.xy=Rotatepoint(A,2*pi/3,B);
D.xy=Rotatepoint((5,2),pi/3,B);
E.xy=Rotatepoint([3,0],-pi/4,A);
```



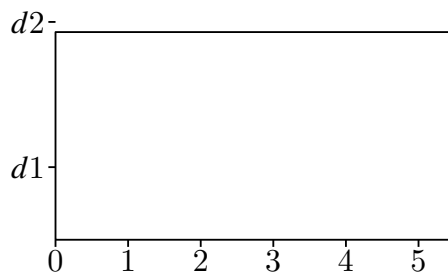
関数 Rulerscale(始点 , 横軸目盛 , 縦軸目盛)

機能 目盛を打つ

説明 始点の位置を縦横の起点として目盛りを打つ。
 目盛はリストで与える。
 ["r",a,b,c] の形式では, a から b まで c 間隔で目盛を打つ。
 ["f",n1,"str",n2,"str",...] の形式では, n と "str" がセットで, n の位置に "str" を書く。ただし, 位置は Cinderella の描画面の原点を 0 とする。
 Framedata() または Framedata2() とともに用いると矩形に目盛を打つことができる。

例：A を原点に置いた矩形枠を描き, 横に 0,1,2,3,4,5, 縦に d1, d2 の目盛を打つ。

```
Framedata2("1",[A,B]);
Rulerscale(A,["r",0,5,1],["f",1,"d1",3,"d2"]);
```



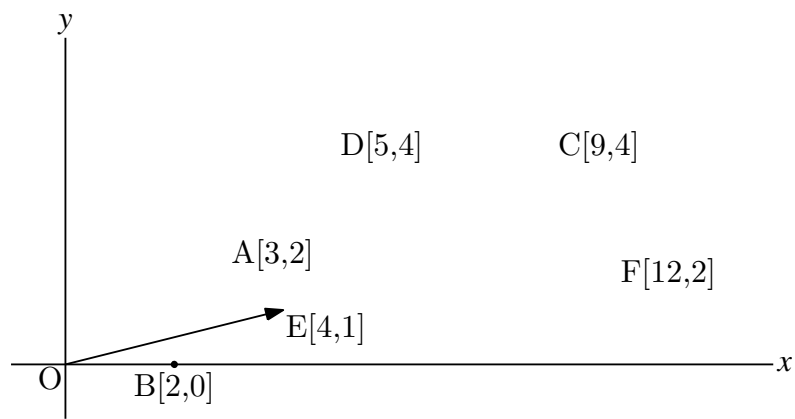
関数 Scalepoint(点, 比率ベクトル, 中心)

機能 点の位置を拡大・縮小する

説明 点を, 指定された中心を原点とする座標系で, 比率ベクトルの分だけ拡大・縮小した位置に置く。

例：点 C を, 点 A を原点中心に横に 3 倍, 縦に 2 倍した位置に置く。
 点 D を, 点 A を点 B 中心に横に 3 倍, 縦に 2 倍した位置に置く。
 点 F を, 点 A を原点中心にベクトル \overrightarrow{OE} で示された比率の位置に置く。

```
C.xy=Scalepoint(A,[3,2],[0,0]);
D.xy=Scalepoint(A,[3,2],B);
F.xy=Scalepoint(A,E.xy,[0,0]);
```

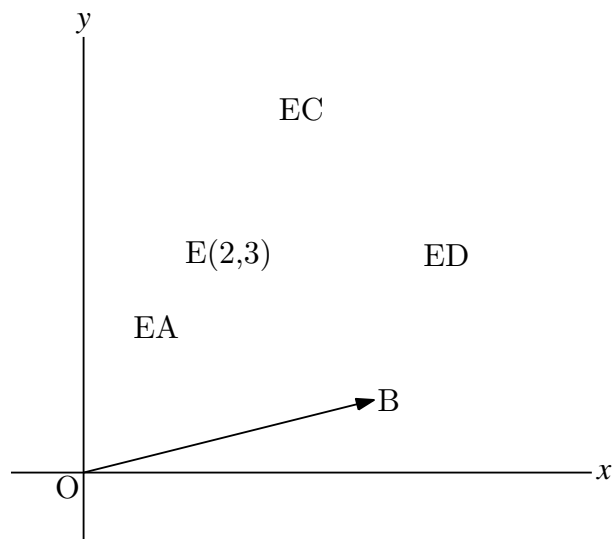


⇒ [関数一覧](#)

関数 Translatepoint(点 , 移動ベクトル)
機能 点を平行移動する
説明 点を移動ベクトルで示された分だけ平行移動する。

例：点 C は点 A を x 軸方向に 2 , y 軸方向に 3 だけ平行移動した点
 点 D は点 A をベクトル \overrightarrow{OB} だけ平行移動した点

```
C.xy=Translatepoint(A,[2,3]);
D.xy=Translatepoint(A,B.xy);
```



⇒ [関数一覧](#)

3.3 プロットデータの操作

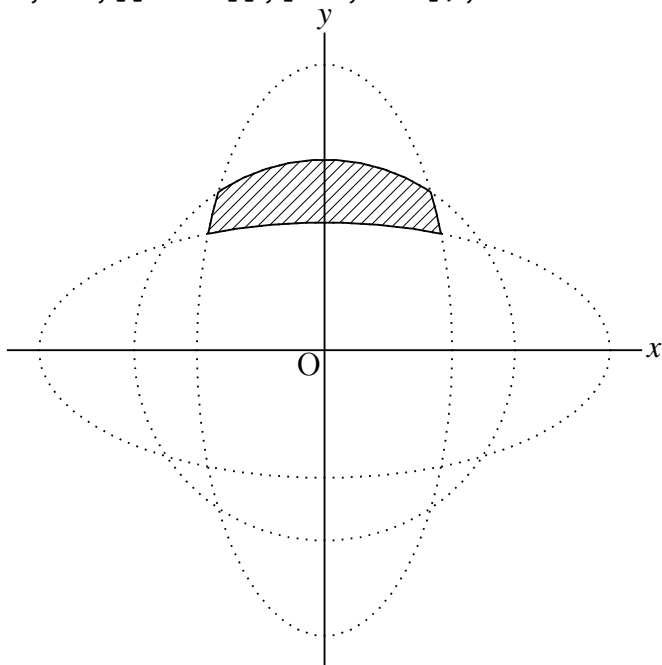
関数	Changestyle(PD リスト, optionss)
機能	描画オプションを変更する
説明	複数の図形の描画オプションを一括して変更する。

例：線分 AB, 円 AB の線を破線にして T_EX に書き出さないようにする。

```
Changestyle(["sgAB", "crAB"], ["da", "notex"]);
```

関数	Enclosing(name, PD リスト, [位置, 方向, 数式])
機能	複数の曲線から閉曲線を描く。
説明	引数の位置, 方向, 数式は, 複数点の情報を [] 内にまとめて記入できる。以下は一例。

```
Circledata([A,B], ["do"]);  
Scaledata("1", "crAB", 1.5, 1/1.5, ["do"]);  
Scaledata("2", "crAB", 1/1.5, 1.5, ["do"]);  
Tmp1=Intersectcrvs(sc2, crAB);  
Putpoint("C", Tmp1\_1);  
Enclosing("1", ["sc2", "crAB", "sc2", "Invert(sc1)"], [C]);  
Hatchdata("1", "i", [{"en1"}], ["dr, 0.5"]);
```

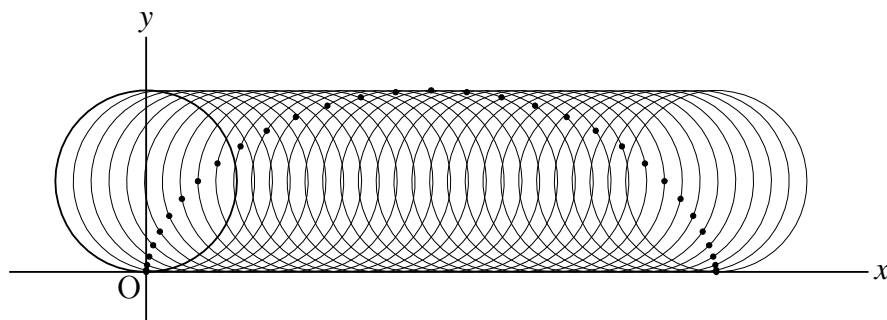


⇒ [関数一覧](#)

関数 AddGraph(name , プロットデータのリスト,option)
機能 複数のプロットデータをまとめる
説明 複数のプロットデータをまとめて扱う。たとえば、円と、円周上の点の2つのプロットデータをまとめて扱えば、スライダを用いたり動画にするときに、個々に移動や回転をしなくてすむ。Joincrvs() では、プロットデータをつなげて1つのプロットデータにするが、AddGraph() では、プロットデータをリスト化して扱うという違いがある。
 プロットデータのリストは、プロットデータ名を文字列化して渡す。たとえば、円のプロットデータが cr1 のとき、"cr1" とする。

例：サイクロイドの図を描く。

```
Setpt(3);
Circledata("1",[[0,1],[0,0]]);
Pointdata("1",[0,0]);
AddGraph("1",["pt1"],"cr1",["nodisp"]);
nn=32;
forall(1..nn,
  t=2*pi/nn*#;
  Rotatedata(text("#"),"ad1",-t,[[0,1],"nodisp"]);
  Translatedata(text("#"),"rt"+text("#),[t,0],["dr,0.3"]);
);
```



ここで、AddGraph() の引数で与えるプロットデータのリストで、点のプロットデータ pt1 を"pt1" としていることに注意。円のプロットデータが、点の座標のリストであるのに対し、点のプロットデータは一つの座標だけなので、このようにしてリスト化して渡す。

[⇒ 関数一覧](#)

関数 Hatchdata(name , 方向リスト , プロットデータ , optionss)
機能 閉曲線の内部に斜線を引く。
説明 引数は、曲線名、内部外部のパターンを与える”i”, ”o” の文字列、閉曲線を与える曲線分と領域の内部を定める方向のリストとオプション。

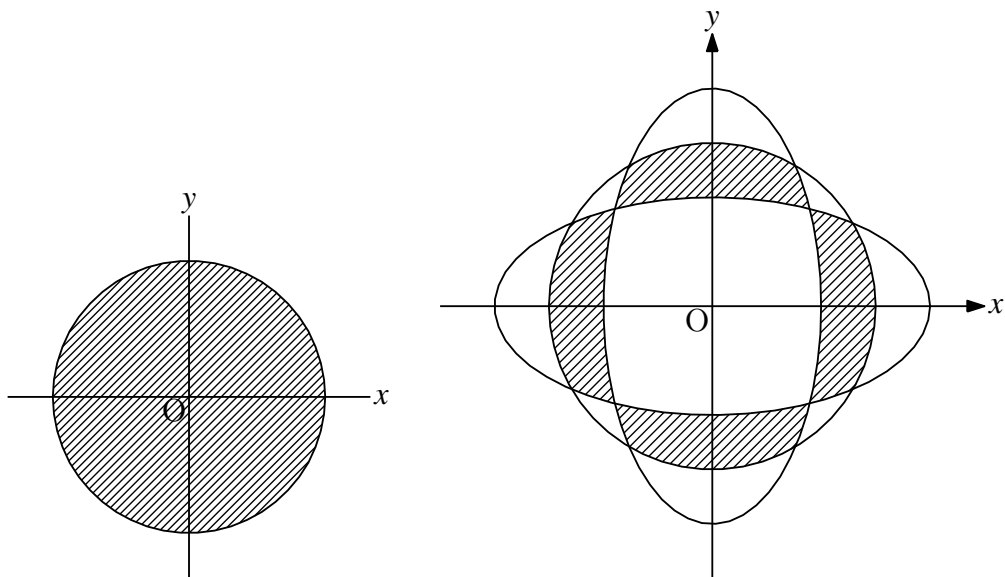
Scilab でハッチデータを作成して読み込む。同じ名前のハッチデータがすでにある場合、”m”をつけるとハッチデータを強制的に更新することができる。

例：円の内部。(下图左)

```
Circledata([A,B],["dr"]);
Hatchdata("1",["i"],[["crAB"]],["dr,0.7","out"]);
```

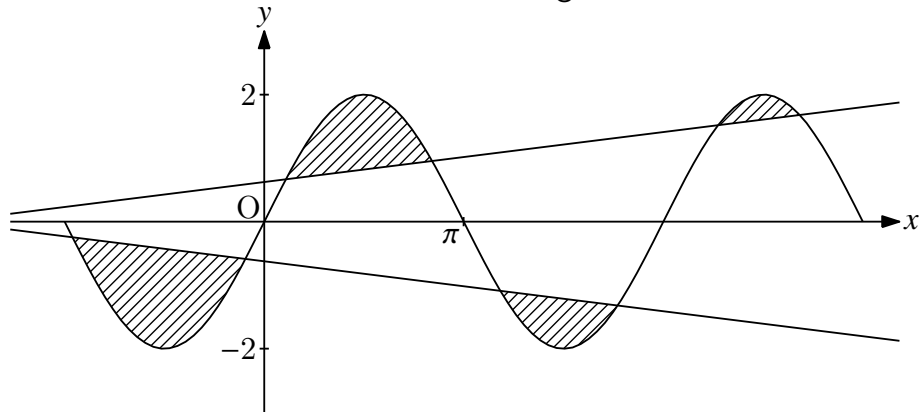
例：3つの閉曲線の内側・外側のパターンが同一である領域(下图右)

```
Circledata([A,B],["dr"]);
Paramplot("1","[4*cos(t),2*sin(t)]","t=[0,2*pi]");
Paramplot("2","[2*cos(t),4*sin(t)]","t=[0,2*pi]");
Hatchdata("1",["ioi"],[["crAB"],["gp1"],["gp2"]],["dr,0.7","out"]);
Hatchdata("2",["iio"],[["crAB"],["gp1"],["gp2"]],["dr,0.7","out"]);
```



例：複数のパターンに分かれた領域。

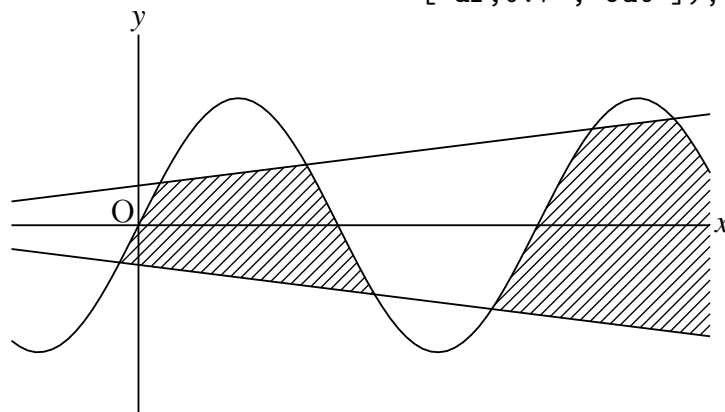
```
Plotdata("1","2*sin(x)","x=[-pi,3*pi]","Num=100");
Listplot([A,B]);
Listplot([A,C]);
io="out";
Hatchdata("1",["ii"],["sAB","n"],["gr1","s"],["dr,0.7",io]);
Hatchdata("2",["ii"],["sAC","s"],["gr1","n"],["dr,0.7",io]);
```



複数領域の斜線塗の場合、それらの領域をまとめて出力するか、まとめて Cinderella に読み込む場合が多く、そのような場合には 4 行目のような指定の仕方もある。

例：描画範囲とあわせて閉領域となる場合。

```
Plotdata("1","2*sin(x)","x=[-pi,3*pi]","Num=100")
Listplot([A,B]);
Listplot([A,C]);
Hatchdata("1",["iio"],["sAB","s"],["sAC","n"],["gr1","[0,2]"],["dr,0.7","out"]);
```

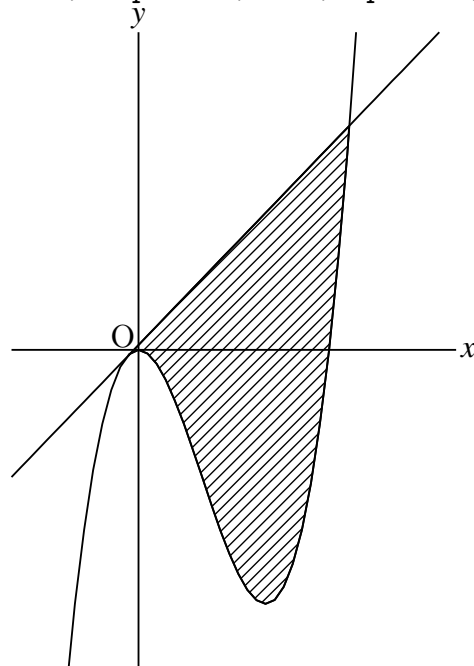


例：接線で囲まれた領域の場合。

```

Def fun("f(x)",["regional(y)","y=x ^ 2*(x-3)","y"]);
Plotdata("1","f(x)","x");
Put onCurve("A",gr1,-1);
coef=Derivative("f(x)","x","A.x");
Def var("DC=coef");
Def fun("g(x)",["regional(y)","y=DC*(x-A.x)+A.y","y"]);
Plotdata("2","g(x)","x");
Tmp=Intersectcrvs(gr1,gr2);
Partcrv("1",A,Tmp __ 3,"gr1");
Partcrv("2",A,Tmp __ 3,"gr2");
Hatchdata("1",["ii"],[["part1","n"],["part2","s"]],["dr,0.7","out"]);

```



斜線の向きや間隔を変えることもできる。

例：円の内部または円と直線で区切られた図形

```

Circledata([A,B]);   で円データの名前が crAB となる。これを用いて
Hatchdata("1",["i"],[["crAB"]]); 円内に傾き 45° の斜線を引く (下図 ha1)
Hatchdata("2",["i"],[["crAB"]],[-40,2]); 傾き -40°, 間隔を 2 倍に (ha2)
                                   (間隔は実数で指定できる)
Hatchdata("3",["i"],[["crAB"]],["dr,0.5"]); 線の太さを 0.5 倍に (ha3)
Hatchdata("4",["i"],[["crAB"]],[-45,2,"dr,0.5"]); (ha4)

```

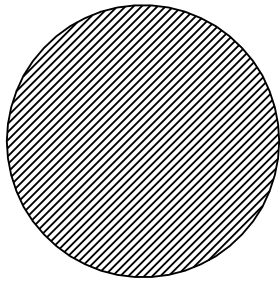


図 ha1

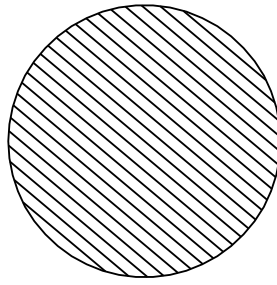


図 ha2

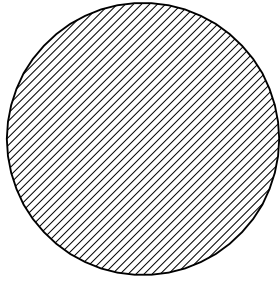


図 ha3

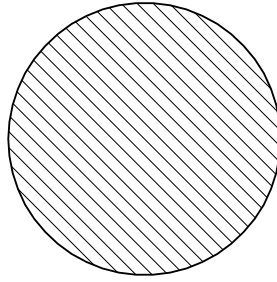


図 ha4

この円に直線を加え領域を分ける。

`Lineplot("1",[A,B]);` 直線データの名称は `ln1`

`Lineplot("2",[A,C]);` 直線データの名称は `ln2`

円の内部で右上 (`ln1` と `ln2` の北側) (下図 ha5)

`Hatchdata("5",["iii"],[["crAB"],["ln1","n"],["ln2","n"]]);`

円の内部で右上と左下 (下図 ha6)

`Hatchdata("6",["iii","ioo"],[["crAB"],["ln1","n"],["ln2","n"]]);`

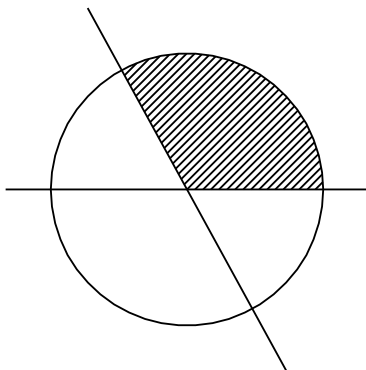


図 ha5

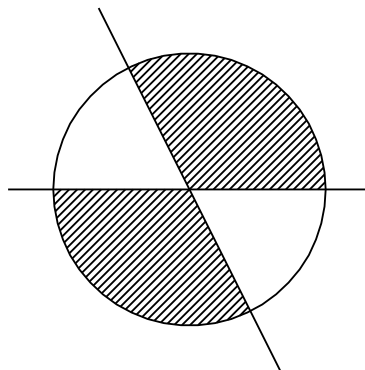


図 ha6

円データ `crAB` を作るが描かなければ境界線が含まれない。(下図 ha7)

```
Circledata([A,B],["notex"]);
Hatchdata("7",["i"],[["crAB"]]);
```

境界線を破線で描くと開集合のようになる（下図 ha8）

```
Circledata([A,B],["da"]);
Hatchdata("8",["i"],[["crAB"]]);
```



図 ha7

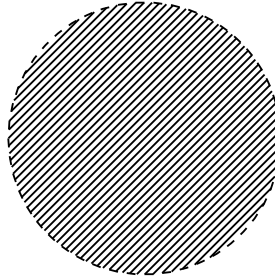
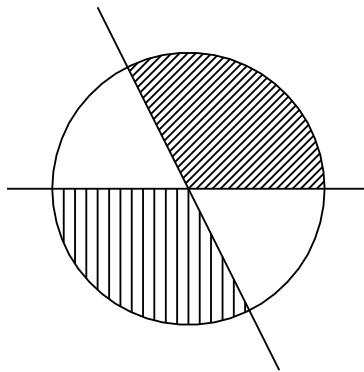


図 ha8

問題：次図のように上と下で斜線の描き分けをするには？

答は図の後に



問題の答：次のように 2 つのデータを作り両方描く

```
Hatchdata("9a",["iii"],[["crAB"],["ln1","n"],["ln2","n"]]);
Hatchdata("9b",["ioo"],[["crAB"],["ln1","n"],["ln2","n"]],[90,2]);
```

（この答は図 ha7 と図 ha8 との関係と同じ）

関数 Dotfilldata(name, 方向リスト, プロットデータ, optionss)

機能 領域を点で敷き詰める。

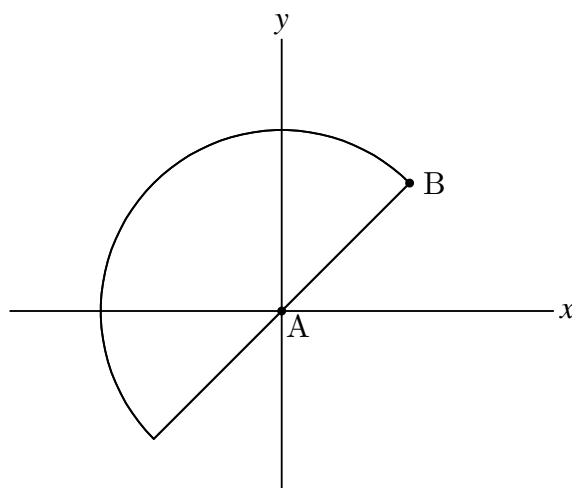
説明 Scilab とデータの授受をおこなって描画する。書式は Hatchdata() と同じ

[⇒ 関数一覧](#)

関数 Joincrvs(name, プロットデータのリスト, options)
機能 隣接する曲線プロットデータの リストを繋いで 1 本の曲線を作る。
説明 曲線のリストは隣接する順番で指定する。プロットデータ名の頭部は join。
options は線種 "dr, n" , "da,m,n" , "do,m,n"

例：線分 $y = x$ ($-\sqrt{2} \leq x \leq \sqrt{2}$) と半円を繋いで得られる閉曲線を描く。

```
Plotdata("1", "x", "x=[-sqrt(2),sqrt(2)]");
Putpoint("B", [sqrt(2),sqrt(2)]);
Circledata("2", [A,B], ["Rng=[pi/4,pi/4*5]"]);
Joincrvs("3", ["gr1", "cr2"]);
```



[⇒ 関数一覧](#)

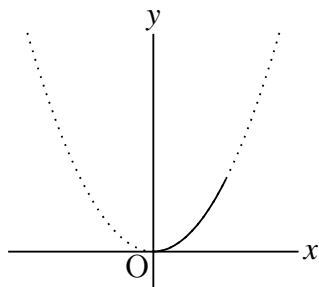
関数 Partcrv(name, A, B, プロットデータ, options)
機能 曲線プロットデータ上の点 A, B の間の部分曲線を描く。
説明 プロットデータ名の頭部は part。
options は線種 "dr, n" , "da,m,n" , "do,m,n"

例 1：2 点 A, B の順序と曲線の向きによって部分曲線が決まる。曲線の向きは、 $y = f(x)$ のグラフでは x 座標が増加する向き。ここで、PDname="gr1" のように”で囲むことが必要。

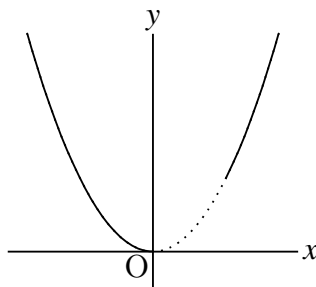
```
Plotdata("1", "x^2", "x", ["do"]); (放物線の名前は gr1)
```

Partcrv("1", [0,0], [1,1], "gr1"); (部分曲線の名前は part1)

Partcrv("2", [1,1], [0,0], "gr1"); (部分曲線の名前は part2)



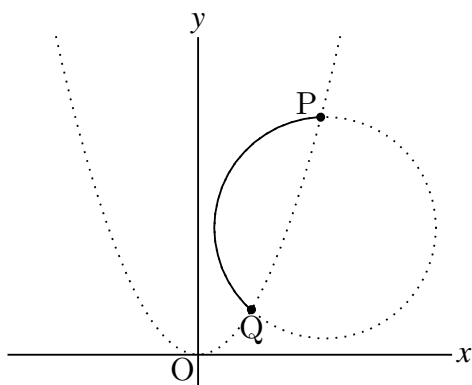
part1 の図



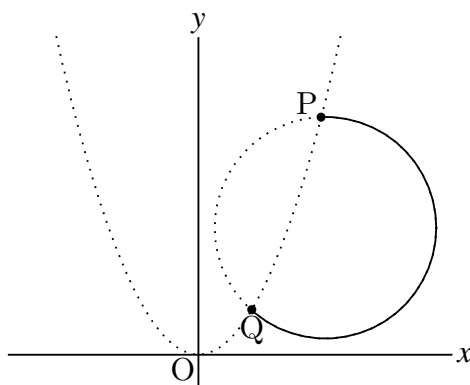
part2 の図

例 2：閉曲線の例. パラメータ表示曲線はパラメータの増える向きをもつ。

```
Circledata([A,B], ["do"]);
Plotdata("1", "x^2", "x", ["do"]);
tmp=Intersectcrvs(crAB, gr1);
P.xy=tmp_1;
Q.xy=tmp_2;
Partcrv("1", P, Q, "crAB");
Partcrv("2", Q, P, "crAB");
```



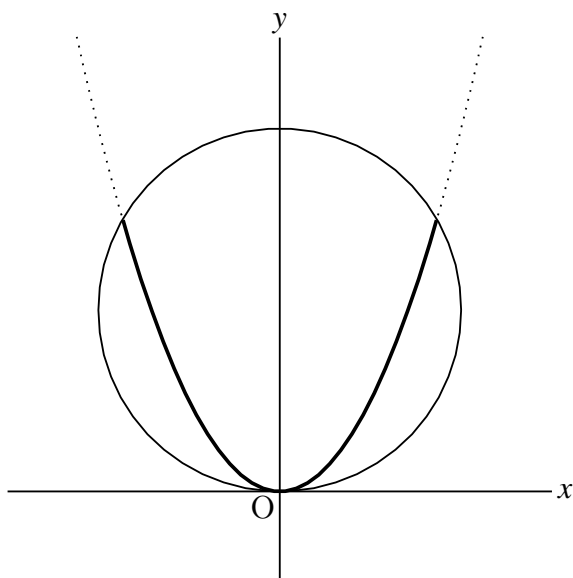
part1 の図



part2 の図

例 3：放物線 $y = x^2$ が円で切り取られる部分を描く。

```
Circledata([A,B]);
Plotdata("1", "x^2", "x", ["do"]);
tmp3=Intersectcrvs("crAB", "gr1");
Partcrv("2", tmp3_2, tmp3_1, "gr1", ["dr, 2"]);
```

[⇒ 関数一覧](#)

関数 PutonCurve(点の名前, プロットデータ, options)

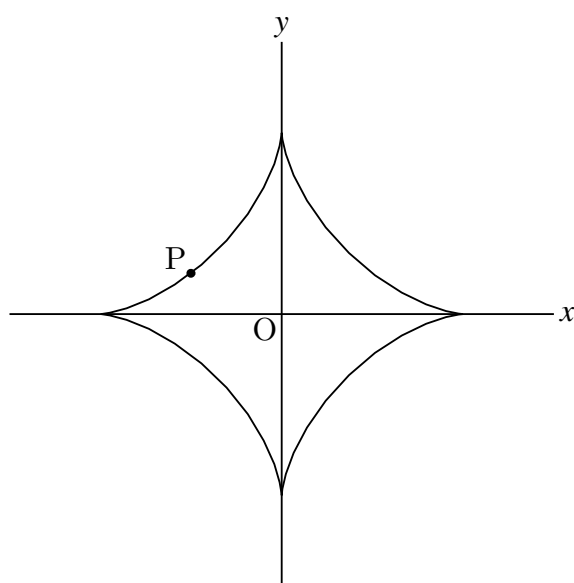
機能 曲線上に点を乗せる。

説明 点が存在しない場合は新たに作る。すでにその点が存在する場合は, その点の x 座標を使う。初期値の x 座標のデフォルトは 0。
options は [x 座標の範囲]。

例：アステロイド上の動点 P をとる。

```
Paramplot("1", "[2*cos(t)^3, 2*sin(t)^3]", "t=[0, 2*pi]");
PutonCurve("P", "gp1", [-1, 1]);
```

点 P がアステロイド上にでき, この点はドラッグするとアステロイド上を $-1 \leq x \leq 1$ の範囲で動かすことができる。

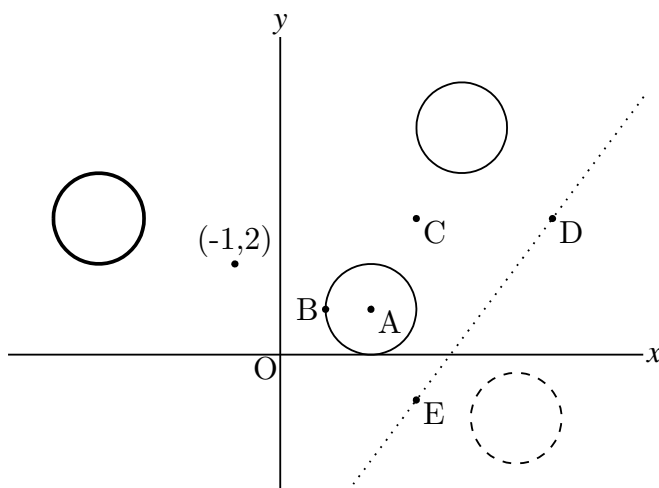


[⇒ 関数一覧](#)

関数 Reflectdata(name , プロットデータ , 対称点または対称軸,options)
機能 プロットデータの鏡映を作成
説明 プロットデータを指定された点または軸に関して対称移動する。
対称点は座標または、点の識別名。ただし、対称点を座標で示すときは要素がひとつのリストにする。
対称軸はリスト [点 1, 点 2] で指定
options は線種

例：中心 A , 半径 AB の円を描き、そのプロットデータを用いて、
点 C に関して対称な円を
点 (-1,2) に関して対称な円を太い実線で
直線 DE に関して対称な円を破線で
それぞれ描く。
作成されるプロットデータの名称を rf1 ,rf2 ,rf3 とする。

```
Circledata([A,B]);\\  
Reflectdata("1","crAB",[C]);\\  
Reflectdata("2","crAB",[[-1,2]],["dr,2"]);\\  
Reflectdata("3","crAB",[D,E],["da"]);\\
```



[⇒ 関数一覧](#)

関数 Rotatedata(name , プロットデータ , 角度 , [中心 , options])
機能 プロットデータの位置を回転する

説明 図形を，中心で示された点の周りに回転する。角度は弧度法で与える
中心と options はまとめてリストで与える。options は線種

例：中心 A，半径 AB の円のプロットデータを描き，

点 C を中心に $\frac{\pi}{2}$ だけ回転した円

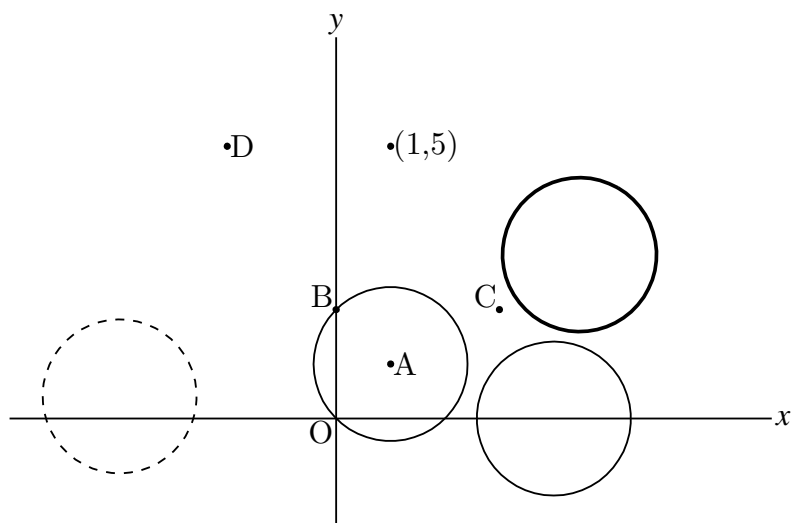
点 (1,5) を中心に $\frac{\pi}{3}$ だけ回転し，線を太くした円

点 D を中心に $-\frac{\pi}{3}$ だけ回転し，破線にした円

をそれぞれ描く。

作成されるプロットデータの名称を rt1 ,rt2 ,rt3 とする。

```
Circledata([A,B]);\\
Rotatedata("1","crAB",pi/2,[C]);\\
Rotatedata("2","crAB",pi/3,[1,5],"dr,2");\\
Rotatedata("3","crAB",-pi/3,[D,"da"]);\\
```



[⇒ 関数一覧](#)

関数 Scaledata(name ,プロットデータ, x 方向比率, y 方向比率, [中心, options])

機能 図形の位置を拡大・縮小する

説明 図形の位置をプロットデータを用いて指定された比率で拡大・縮小する

中心と options はまとめてリストで与える。options は線種

例：中心 A, 半径 AB の円を描いておく。プロットデータ名は crAB となる。

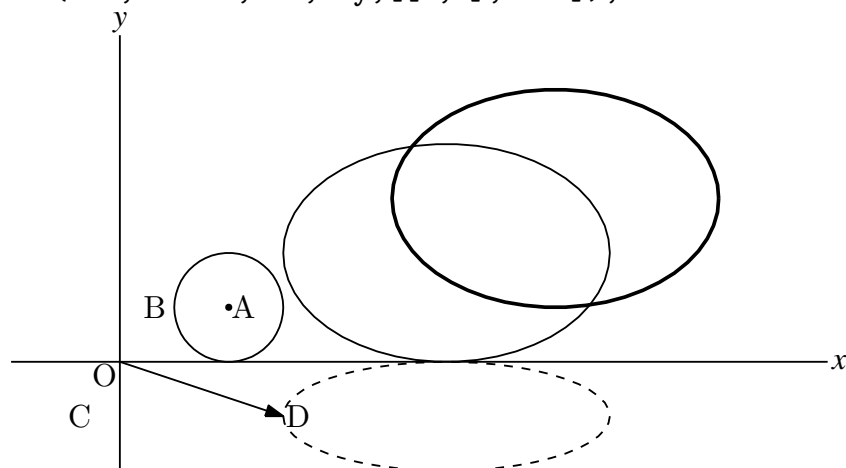
この円を

原点中心に x 軸方向に 3, y 軸方向に 2 だけ平行移動する。

C を中心に x 軸方向に 3, y 軸方向に 2 だけ平行移動し, 実線で太く描く。

原点中心にベクトル \overrightarrow{OD} だけ平行移動し, 破線で描く。

```
Circledata([A,B]);  
Scaledata("1","crAB",3,2,[[0,0]]);  
Scaledata("2","crAB",3,2,[C,"dr,2"]);  
Scaledata("3","crAB",D.x,D.y,[[0,0],"da"]);
```



関数 `Shade(プロットデータのリスト, optionss)`

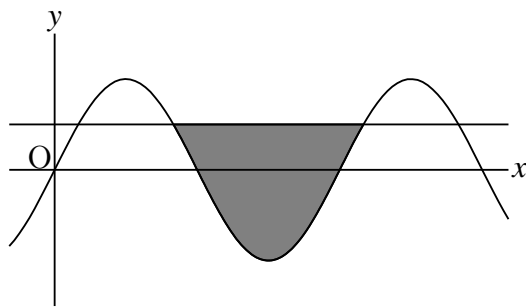
機能 閉曲線で囲まれた領域を塗りつぶす。

説明 第 1 引数には, 閉曲線を与える曲線分のリストを並べる。

optionss は, Cinderella の画面上での描画色, 濃さをリストで与える。

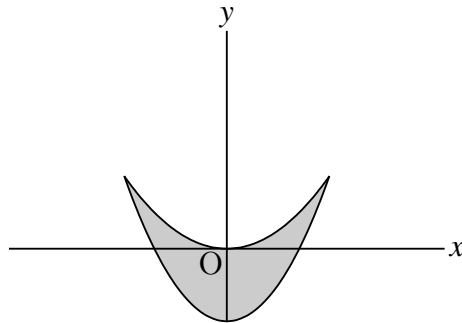
例: $y = 2 \sin x$ のグラフと直線 $y = 1$ (直線 AB の一部) とで囲まれた部分に 0.5 の濃さで色を塗る。

```
Setax([7,"nw"]);  
Plotdata("1","2*sin(x)","x",["Num=100"]);  
Plotdata("2","2*sin(x)","x=[5*pi/6,13*pi/6]",["Num=100"]);  
Lineplot("1",[[0,1],[1,1]]);  
Listplot("1",[[5*pi/6,1],[13*pi/6,1]]);  
Shade(["gr2","sg1"],[0.5]);
```



例：2つの放物線で囲まれた部分に 0.2 の濃さで色を塗る。Cinderella の画面では赤で少し薄く塗る。

```
Plotdata("1","x^2-1","x=[-sqrt(2),sqrt(2)]",["Num=100"]);
Plotdata("2","x^2/2","x=[-sqrt(2),sqrt(2)]",["Num=100"]);
Shade(["gr2","gr1"],[0.2,"color->[1,0,0]","alpha->0.4"]);
```



⇒ [関数一覧](#)

関数 Translatedata(name , プロットデータ , 移動ベクトル , options)
機能 プロットデータを平行移動する
説明 プロットデータを移動ベクトルで示された分だけ平行移動する。
name はこれによってつくるプロットデータの名称を設定。単なる番号は不可。
options は線種

例：中心 A, 半径 AB の円を描いておく。プロットデータ名は crAB となる。

この円を

x 軸方向に 2, y 軸方向に 3 だけ平行移動する。

ベクトル \overrightarrow{OC} だけ平行移動し、実線で太く描く。

ベクトル \overrightarrow{OD} だけ平行移動し、破線で描く。

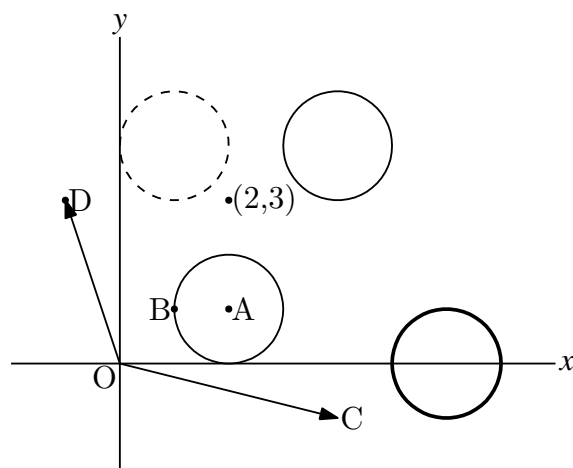
作成されるプロットデータの名称を tr1 , tr2 , tr3 とする。

```
Circledata([A,B]);
```

```
Translatedata("tr1","crAB",[2,3]);
```

```
Translatedata("tr2","crAB",C,["dr,2"]);
```

```
Translatedata("tr3","crAB",D,["da"]);
```



[⇒ 関数一覧](#)

3.4 微積分など

関数 Derivative(関数式, 変数, 値)

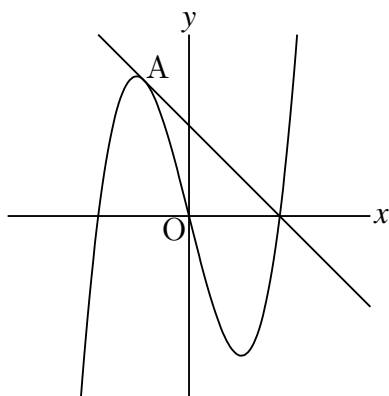
機能 関数の微分係数を求める

説明 関数式で与えられた関数の, 「変数=値」における微分係数を求める。

値は, 点の座標を用いることができる。点 A の x 座標であれば, A.x とする。

例: 3 次曲線上の点 A で接線を引く。点 A,B は作図ツールでとっておく。

```
Deffun("f(x)", ["regional(y)", "y=x^3-4*x", "y"]);  
coef=Derivative("f(x)", "x", A.x);  
A.y=f(A.x);  
B.y=coef*(B.x-A.x)+A.y;  
Plotdata("1", "f(x)", "x", ["Num=200"]);  
Lineplot([A,B]);  
Letter([A,"ne","A"]);
```



[⇒ 関数一覧](#)

関数 integrate(関数式, 変数, 範囲, options)

関数 integrate(PD, 範囲, options)

機能 関数式またはプロットデータで与えられた曲線の積分値を求める

説明 関数式またはプロットデータで与えられた曲線の, 指定された範囲の定積分の値を求める。

関数を与えたときの options は次の通り。

”Rule=s” : シンプソン法による。デフォルトは台形公式。

”R=s” と略記できる。

”Num=数値” : 分割数の指定。初期値は 100

一般にはシンプソン法の方が精度がよいが、そうではない場合もある。

例: $f(x) = x^3 - 2x^2 + 2$ について, 0 から 3 までの定積分の値を求める。

CindyScript では, guess() 関数を用いると, 可能な場合は値を分数表記にする。

```
f(x):=x^3-2*x^2+2;  
println(guess(Integrate("f(x)","x",[0,3],["Num=200"])));  
println(guess(Integrate("f(x)","x",[0,3],["Num=5","Rule=s"])));
```

を実行すると, コンソールに結果が表示される。シンプソン法では $33/4$ と表示される。整関数の場合, シンプソン法では分割数がかなり少なくても高い精度が得られる。整関数でない場合, 分割数を区間幅の 10 の整数倍にするとよい精度になる。たとえば, $f(x) = \sqrt{x}$ の場合, 区間が $[2,9]$ であれば, Num=100 よりも Num=70 の方が精度がよい。

プロットデータを与えたときは, アルゴリズムが異なるので上記オプションは無効。

例: 上の例と同じ関数をプロットデータで指定する。

```
plotdata("1","x^3-2*x^2+2","x");  
println(Integrate("gr1",[0,3]));
```

プロットデータで与える方法は, R を利用して正規分布曲線を描いたときなどに記述が楽になる。[PlotdataR\(\)](#) の項を参照のこと。

[⇒ 関数一覧](#)

関数 Inversefun(関数, 範囲, 値)

機能 関数の逆関数値を求める

説明 関数は文字列で, 関数式もしくは定義された関数名とする。

指定された範囲の中で逆関数値を求める。存在しない場合は一方の端点を戻り値とし, コンソールに「not found」と表示される。

数式処理ではなく数値探索のアルゴリズムを使っているので, 単調関数でない場合は範囲をできるだけ狭くとるとよい。値が複数ある場合は, 小さいほうが表示される。

例: `x=Inversefun("sin(x)","x=[0,pi/2]",0.5);`

実行すると $x = \frac{\pi}{6}$ となる。

```
Defun("f(x)", ["regional(y)", "y=sin(x)", "y"]);  
x=Inversefun("f(x)", "x=[0,pi/2]", 0.5);
```

でも同じ結果が返される。

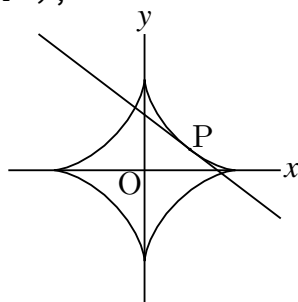
```
x=Inversefun("f(x)", "x=[pi/3,pi]", 0.97);
```

は区間設定が不適切で正しい結果が得られない。

例: アステロイド上に点 P をとり, P の x 座標から, 逆関数値 t を求めて接線を引く。

できた点 P はドラッグしてアステロイド上を動かすことができる。

```
Defun("fx(t)", ["regional(x)", "x=2*cos(t)^3", "x"]);  
Defun("fy(t)", ["regional(y)", "y=2*sin(t)^3", "y"]);  
Paramplot("1", ["fx(t),fy(t)", "t=[0,2*pi]");  
PutonCurve("P", gp1, 1);  
if(P.y>=0, range="t=[0,pi]", range="t=[pi,2*pi]");  
t0=Inversefun("fx(t)", range, P.x);  
dx=Derivative("fx(t)", "t", t0); dy=Derivative("fy(t)", "t", t0);  
Putpoint("Q", [0,0], P.xy+[dx,dy]);  
Lineplot([P,Q]);  
Drawpoint(P);  
Letter(P, "ne", "P");
```



3.5 作表

関数 `Tabledata(name, 縦横データ, 除外線, options)`

機能 表の枠を作成し, 表のデータ list を返す

説明 Cinderella の描画面上に左下を原点とする表を作成する。

他の関数との引数の整合性, $\text{KE}T_{\text{pic}}$ のコマンドとの整合性などから, 先頭に `name` の引数をつけるが, 実際にはあまり利用しないので, 空文字""でもよい。

`options` は線種と"notex"など, および"Rng=no"。

"Rng=no" をつけると, NE,SW による出力領域指定が有効になる。つけない場合は, 表の部分だけが出力される。

縦横データには, 次の 2 通りの書式がある。

(1) 横のセル数, 縦のセル数, 表の横幅, 表の縦幅 を指定する。

例: `Tabledata("", 4, 5, 80, 50, [])`;

(2) 横と縦の幅を指定したリストを使う

例: `Yoko=[20, 20, 20, 20]`;

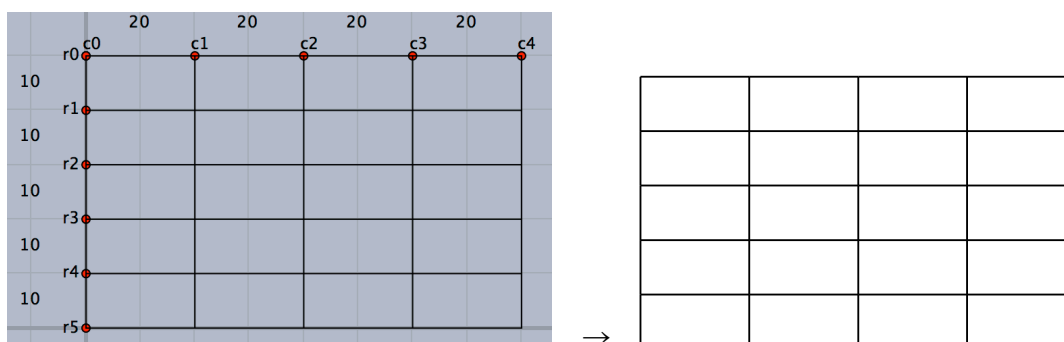
`Tate=[10, 10, 10, 10, 10]`;

`Tabledata("", Yoko, Tate, [])`;

上の 2 つのスクリプトは同じ表を作成する。幅は Cinderella の描画面の 0.1 を単位とする。

作成された表には, 行, 列の制御点がつく。画面上では, 横罫線の番号 `r0, r1, ...` 縦罫線の番号 `c1, c2, ...` と見ることもできる。また, 縦幅, 横幅が数字で示される。ただし, これらは $\text{T}_{\text{E}}\text{X}$ には出力されない。

また, 作表は Cinderella の描画面上では座標平面上に置かれるが, $\text{T}_{\text{E}}\text{X}$ への出力は座標平面上には置かないことが多いので, 座標軸は非表示としている。



表のサイズ, 行幅, 列幅は, 作成後にそれぞれの制御点をドラッグすることにより任意に変えることができる。

除外線は、除外するセルの罫線を、r と c で位置指定する。

横罫線の場合、横罫線の番号、範囲（から、まで）

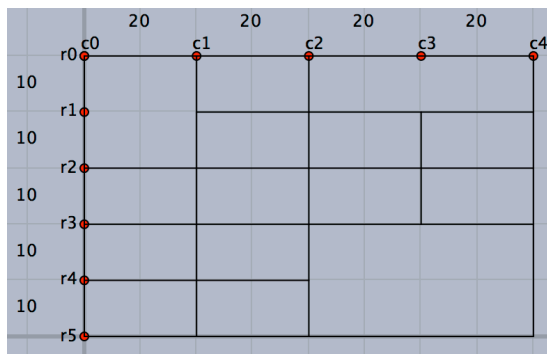
縦罫線の場合、縦罫線の番号、範囲（から、まで）

とする。

例：Rmv=["r1c0c1","c3r0r1","c3r3r5","r4c2c4"];

Tabledata("",4,5,80,50,Rmv);

で、次の表ができる。



除外する罫線がない場合は、空リストとする。

例：Tabledata("",Yoko,Tate,[]);

Tabledata() 関数は、制御点 r0,r1,...,c0,c1,... がなければ新しく作り、すでに存在する場合はそのままとする。したがって、一度表を作成したのち、行数・列数を修正して作り直す場合は、一度既存の点を消去する必要がある。そのためには、「すべての点を選択する」ツールをクリックして点を消去するのがよい。クリックすると、消去後すぐに新規作成される。(誤って「すべての要素のを選択する」を選ばないこと)

他の点が描画されている場合は、表の部分だけドラッグで選択するか、表示メニューの「式による表示」で一覧表を出して、制御点を選択して消去する。

ただし、行数、列数が多すぎた場合は、必ずしも作り直す必要はない。Settable() 関数により、実際に書き出す表の範囲を指定できる。

[⇒ 関数一覧](#)

関数	Tabledatalight(name, 縦横データ, 除外線, options)
機能	幾何点を持たない表の枠を作成し、表のデータ list を返す
説明	Tabledata() が Cinderella の幾何点を生成するのに対し、こちらは幾何点を生成しない。

幾何点を作成しないメリットは、スクリプトだけで縦横幅を変更できること。デメ

リットはインタラクティブな微調整ができないこと。

option として、ラベルのスキップ値（スキップするところは表示されない）を指定することができる。ただし、ラベルは Cinderella の画面上だけの問題。

```
例：Yoko=[20,20,20,20];
      Tate=[10,10,10,10,10];
      Tabledatalight("",Yoko,Tate,[],[2]);
      とすると、r1,r3,c1,c2 が非表示となる。
```

[⇒ 関数一覧](#)

関数	ChangeTablestyle(罫線リスト, 変更オプション)
機能	Table の罫線の描画オプションを変更
説明	罫線の部分的に指定して描画オプションを変更できる。

```
例：ChangeTablestyle(["r0c0c3"],["da"]);
      の実行により、r0 の位置の横線が c0 から c3 まで破線に変わる。
      ChangeTablestyle(["c1r0r2"],["da"]);
      の実行により、c1 の位置の縦線が r0 から r2 まで破線に変わる。
```

関数	Findcell(列番号, 行番号)
機能	セルの情報 list（中心, 横幅／2, 縦幅／2）を返す
説明	列番号, 行番号は左上のセルを 1 列 1 行として数える。

```
例：Tabledata(4,5,80,50,[]);
      println(Findcell(tb,2,1));
      とすると、2 列 1 行のセルの中心の座標と横幅の半分, 縦幅の半分の値がリスト
      としてコンソールに表示される。結果は [[3,4.5],1,0.5]
```

[⇒ 関数一覧](#)

関数	Putcell (列番号, 行番号, 位置, 文字データ)
機能	セルに文字列を入れる
説明	複数のセルにまたぐ位置指定の場合, 列番号, 行番号は, セル左上と右下の制御点の名称で指定する。 位置は c, r, l, t, b（中央 center, 右 right, 左 left, 上 top, 下 bottom）

位置の例を以下に示す。

```
Tabledata("",5,2,100,40,["c1r1r2","c4r1r2"]);
Putcell(1,1,"c","A");
Putcell(2,1,"r","B");
Putcell(3,1,"l","C");
Putcell(4,1,"t","D");
Putcell(5,1,"b","E");
Putcell(c0r1,c2r2,"c","F");
Putcell(c2r1,c3r2,"lb","G");
Putcell(c3r1,c5r2,"rt","H");
```

R0					
	A	BC	D		
R1				E	H
R2	F				
	C0	C1	C2	C3	C4

※ R0,C0, C1 は実際には表示されない

[⇒ 関数一覧](#)

関数 PutcoL (列番号, 文字位置, 文字列リスト)
機能 1 列に順に文字を書き入れる
説明 列番号で指定した列に, 第 1 行から順に文字列リストの文字を書き入れる
 数の場合はダブルクォートでくくなくてもよい。
 セルを飛ばす場合は, ヌル文字列 "" を書く。

関数 PutcoLexpr (列番号, 文字位置, 文字列リスト)
機能 1 列に順に文字を書き入れる
説明 文字列に T_EX 書式を使うことができる

関数 Putrow (行番号, 文字位置, 文字列リスト)
機能 1 行に順に文字を書き入れる
説明 行番号で指定した行に, 第 1 列から順に文字列リストの文字を書き入れる。

関数 Putrowexpr (行番号, 文字位置, 文字列リスト)
機能 1 行に順に文字を書き入れる

説明 文字列に T_EX 書式を使うことができる

文字を入れる例を示す。

```
Tabledata("",5,3,100,45,["c1r1r2","r1c2c3","r2c2c3"]);
Putcol(3,"c",["A","B","C"]);
PutcoLexpr(4,"1",["x^2","y=\sqrt{x^3}"]);
Putrow(1,"c",[1,"二"]);
Putrowexpr(3,"c",["",""\frac{\pi}{2}","", "", "\sum{x^2}"]);
```

R0					
	1	二	A	x^2	
R1			B	$y = \sqrt{x^3}$	
R2		$\frac{\pi}{2}$	C		$\sum x^2$
R3	C0	C1	C2	C3	C4

※ R0,C0,・・・は実際には表示されない

また、この例では、C4 列の罫線を、制御点 C4 をドラッグすることにより右にずらしている。

グラフや文を入れた表の作成例

PutcoLexpr(),Putrowexpr() では、数式だけでなく、一般の T_EX の文を入れることができる。

また、グラフの位置を適当に合わせて描画することにより、表のセルの中にグラフを入れることができる。

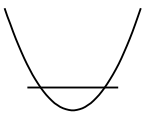
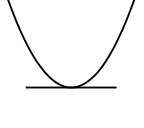
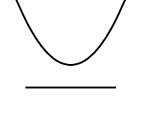
例：2 次関数のグラフと 2 次方程式の判別式の関係

セルの中にグラフを描く例。実際には、セルの位置にグラフを描いているだけ。

```
Tabledata("",3,3,120,90,["r1c0c3","r2c0c3"],["dr,2"]);
ChangeTablestyle(["r1c0c3"],["dr"]);
ChangeTablestyle(["r2c0c3"],["da"]);
Plotdata("1","(x-2)^2+1.5","x=[0.5,3.5]");
Plotdata("2","(x-6)^2+2","x=[4.5,7.5]");
Plotdata("3","(x-10)^2+2.5","x=[8.5,11.5]");
Listplot([A,B]);
```

```
Listplot([C,D]);
Listplot([E,F]);
Putrowexpr(1,"c",["D>0","D=0","D<0"]);
Putrow(2,"c",["2点で交わる","接する","共有点なし"]);
Letter(G,"c","判別式と$x$軸の交点");
```

判別式と x 軸の交点

$D > 0$	$D = 0$	$D < 0$
2点で交わる	接する	共有点なし
		

注意：この例を実行するとわかるが、セルのサイズと文字サイズの関係などにより、Cinderella の画面上と $\text{T}_\text{E}\text{X}$ への書き出しは必ずしも同一にはならない。

例：増減表とグラフ

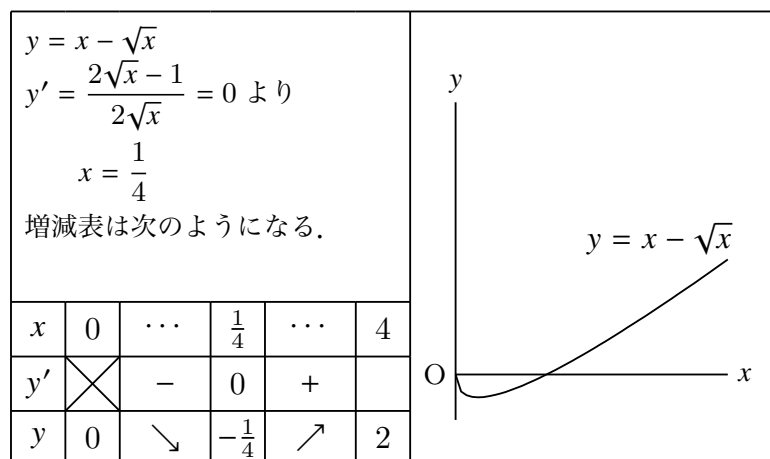
関数の増減表とグラフを 1 つの表の中に入れた例。

```
Tate=[6,6,10,6,10,6,40];
Yoko=[30,6,6,6];
Rmv=["c1r0r1","c2r0r1","c3r0r1","c4r0r1","c5r0r1", "r1c6c7","r2c6c7","r3c6c7"];
Tabledata("",Tate,Yoko,Rmv,["dr"]);
Tlistplot("23d",["c1r2","c2r3"]);
Tlistplot("23u",["c1r3","c2r2"]);
Putrowexpr(2,"c",["x",0,"\cdots","\tfrac{1}{4}","\cdots",4]);
Putrowexpr(3,"c",["y'", "", "-", 0, "+"]);
Putrowexpr(4,"c",["y",0,"\searrow","-\tfrac{1}{4}","\nearrow",2]);
Putcell(1,1,"l2t2", "{\small\begin{minipage}{44mm}$y=x-\sqrt{x}$\\$y'=\frac{2\sqrt{x}-1}{2\sqrt{x}}=0$|より\vspace{1mm}\\hspace*{2zw}$x=\frac{1}{4}$\vspace{1mm}\\増減表は次のようになる\end{minipage}}" );
Plotdata("1","x-sqrt(x)","x=[0,3]","do","notex");
Listplot("2",[[0,0],[3,0]],["do","notex"]);
Listplot("3",[[0,-0.5],[0,3]],["do","notex"]);
```

```

Translatedata("1","gr1",[4.9,1],["dr"]);
Translatedata("2","sg2",[4.9,1],["dr"]);
Translatedata("3","sg3",[4.9,1],["dr"]);
Letter(Ptend(tr2),"e1","\small{$x$}");
Letter(Ptend(tr3),"n1","\small{$y$}");
Letter(Ptstart(tr2),"w1","\small 0");
Expr(Ptend(tr1),"nw-2","y=x-\sqrt{x}");

```







例：凹凸を含めた増減表

```

Tabledata("",8,4,80,40,[]);
Putrowexpr(1,c,["x","\cdots","-1","\cdots","0","\cdots","1","\cdots"]);
Putrowexpr(2,c,["y'","+","+","+","0","-","-","-"]);
Putrowexpr(3,c,["y'',"+"+"0","-","-","-","0","+""]);
Putrowexpr(4,c,["y","\nelarrow","\frac{1}{\sqrt{e}}","\nerarrow",
"1","\serarrow","\frac{1}{\sqrt{e}}","\selarrow"]);

```

x	...	-1	...	0	...	1	...
y'	+	+	+	0	-	-	-
y''	+	0	-	-	-	0	+
y		$\frac{1}{\sqrt{e}}$		1		$\frac{1}{\sqrt{e}}$	

ここで、凹凸を示す矢印は、ketpic.sty で定義されているものである。

nelarrow,nerarrow,selarrow,serarrow,NELarrow,NERarrow,SELarrow,SEarrow
がある。

先頭の ne,se で北東・南東（右上・右下）次の r,l は回転の向き（r:right:反時計回り, l:left:時計回り）の矢印 (arrow) と覚えるとよい。

直線系の矢印は

NEarrow,SEarrow

少しずつ違うので試されたい。

なお、これらの矢印は CindyTeX にはないので、Cinderella の描画面には表示されない。

[⇒ 関数一覧](#)

関数	Settable((左上), 右下)
機能	表データを書き出す
説明	指定された範囲で表の枠線を書き出す。指定しなければ全体が書き出される。 表の出力範囲は、NE,SW による描画範囲にかからわず、全体もしくは Settable() で指定した範囲となる。 位置は、制御点による位置表示で、c2r3 のように指定する。 c,r は小文字で c,r の順。 左上を省略すると、デフォルトの c0r0 と解釈される。

例: `Tabledata("",4,5,80,50,[]);`
`repeat(5,s, Putrow(s,"c",[s*4-3,s*4-2,s*4-1,s*4]));`
`Settable(c4r5);`

2行目で各セルに数字を入れている。

左上と右下（罫線 C4 と R5 の交点）を対角とする範囲、すなわち全体を書き出す。

`Settable(c3r4);`

とすると、左上と右下（罫線 C3 と R4 の交点）を対角とする範囲。

出力するのは罫線だけなので、セルの中身はそのまま表示される。（下図左）

`Settable(c1r1,c3r4);`

とすると、左上（罫線 C1 と R1 の交点）と右下（罫線 C3 と R4 の交点）を対角とする範囲。

罫線だけなので、セルの中身はそのまま表示される。（下図右）

1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8
9	10	11	12	9	10	11	12
13	14	15	16	13	14	15	16
17	18	19	20	17	18	19	20

関数 Tgrid(セルラベル)

機能 表のセルの座標を返す

説明 指定されたセルの左上の座標を返す。実際には、セルラベルは罫線を示しているの
で、指定した罫線の交点（格子点）ということもできる。

関数 Tlistplot(セルラベル 1, セルラベル 2)

機能 指定された 2 つの格子点を線分で結ぶ

説明 セルに斜線を引くのに用いる。

例：Tlistplot(["c0r1","c1r2"]);

[⇒ 関数一覧](#)

3.6 値の取得と入出力

計算値やプロットデータの値を取得したり、Scilab 用とのデータのやりとりをする。

関数 Crossprod(リスト, リスト)
機能 2つのベクトルの外積を求める。
説明 Cindyscript の組み込み関数 cross(リスト, リスト) と同じ。
例: `Crossprod([1,0,0],[1,1,1]);`
結果は `[0,-1,1]`

関数 Dotprod(リスト, リスト)
機能 2つのベクトルの内積を求める。
説明 Cindyscript では、積の演算で内積が求められる。
例: `Dotprod([1,2,3],[1,-1,1]);`
結果は `2`
`[1,2,3]*[1,-1,1]` でも同じ結果を得る。

関数 Findarea(プロットデータ)
機能 プロットデータで囲まれる部分の面積を求める。
説明 閉曲線をなすプロットデータで囲まれる部分の面積を求める。
例: ベジエ曲線で囲まれた部分の面積を求めて表示する。
`Bezier("1",[A,B,C],[[D],[E,F]]);`
`drawtext([2,1],Findarea("bz1"));`

関数 Findlength(プロットデータ)
機能 プロットデータの曲線の長さを求める。
説明 プロットデータが描く曲線の長さを求める。
例: ベジエ曲線で描いた曲線の長さを求めて表示する。
`Bezier("1",[A,B,C],[[D],[E,F]]);`
`drawtext([2,1],Findlength("bz1"));`

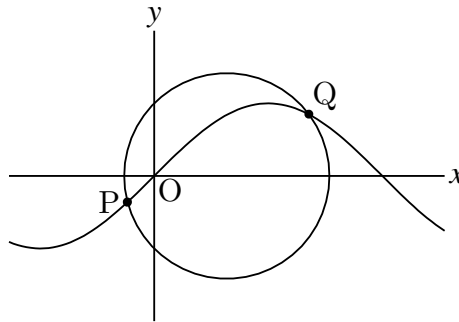
関数 Intersectcrvs(プロットデータ 1, プロットデータ 2, optionss)
機能 2 曲線の交点リストを取得する。
説明 オプションは共有点があるかどうかを判断するための限界値
ただし、オプションは通常は使わないので気にしなくてよい。

例：円と曲線の交点を P,Q とする。

```
Plotdata("1", "sin(x)", "x", ["Num=100"]);
Circledata([A, B]);
tmp=Intersectcrvs(gr1, crAB);
P.xy=tmp_1;
Q.xy=tmp_2;
```

リスト内に 2 交点のデータが $\text{tmp} = [[-0.37, -0.36], [2.13, 0.85]]$ のように入っている。交点の順序は PD1, PD2 の順序と曲線の向きによって決まる。曲線の向きは、 $y = f(x)$ のグラフでは x 座標が増加する向きで、パラメータ表示曲線ではパラメータの増加する向き。また、PD1 上から探し始めて PD2 との交点を拾ってゆく。

交点がひとつの場合も $\text{tmp} = [[2.45, 0.63]]$ と 2 重のリストに入っているの、点として取出すには $P=\text{tmp}_1$; とする。



関数 Invert(PD)
機能 プロットデータの点を逆順にする (reverse と同じ)

関数 MeetCurve(曲線,x0, y0)
機能 曲線上の点を返す。
説明 x 座標が x_0 で、点 (x_0, y_0) に近い曲線上の点を返す。
注) x_0, y_0 は文字列でもよい。

関数 Lcrd()
機能 幾何点, リスト点の論理座標を取得する。

関数 Pcrd()
機能 幾何点, リスト点の物理 (表示) 座標を取得する。

[⇒ 関数一覧](#)

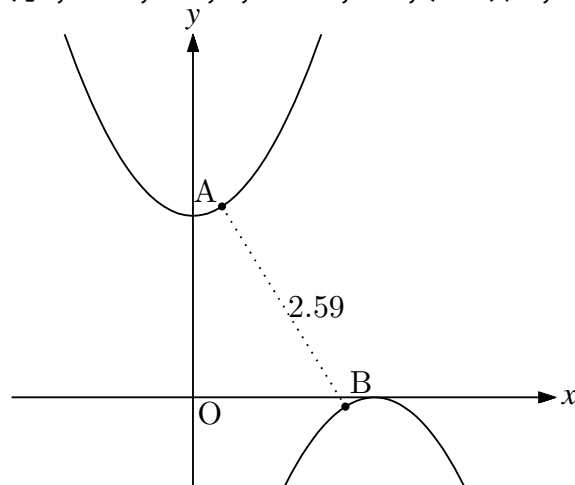
関数 Nearestpt(PD1, PD2)

機能 2 曲線に対し、最も近い点とそのパラメータ、距離のリストを返す

説明 戻り値は、それぞれの曲線上の点の座標とプロットデータ中の位置、その距離からなるリスト。

例：2つの放物線上の点の最短距離とその位置を求める。点 A,B を作図ツールでとっておく。

```
Plotdata("1", "x^2+2", "x=[-2,2]");
Plotdata("2", "-(x-2)^2", "x=[0,3]");
plist=Nearestpt("gr2","gr1");
B.xy=plist_1;
A.xy=plist_3;
Listplot([A,B],["do"]);
Ptsize(4);
Drwpt([A,B]);
Letter([A,"nw","A",B,"n2e","B", (A+B)/2,"e",text(plist_5)]);
```



ここで plist に代入されたリストは次の形である。5 番目の要素が 2 点間の距離、すなわち最短距離。

```
[[1.68,-0.1],29,[0.32,2.1],30,2.59]
```

関数 Nearestptcrv(座標, プロットデータ)

機能 第 1 引数の座標に最も近い曲線プロットデータ上の点座標を返す

説明 次に例示する。中心 A, 半径 AB の円と点 C,D は Cinderella で作図しておく。

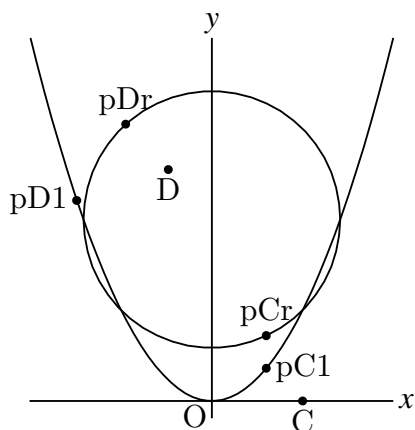
```
Circledata([A, B]);
Plotdata("1", "x^2", "x");
```

```

pCr=Nearestptcrv(C.xy, "crAB")
pC1=Nearestptcrv(C.xy, "gr1")
pDr=Nearestptcrv(D.xy, "crAB");
pD1=Nearestptcrv(D.xy, "gr1");

```

それぞれ、点 C、点 D から最も近い曲線上の点を表す。



[⇒ 関数一覧](#)

関数	Numptcrv (プロットデータ)
機能	曲線の節点の個数を返す
説明	Cindyscript の length(PD) と同じ

例：曲線上のいくつかの点で分割し破線で図示する。

区間 $[-1, 2]$ に対する 2 次関数のグラフを描く

```

Deffun("f(x)",["regional(y)","y=-x^2+4","y"]);
Plotdata("1","f(x)","x=[-1,2]","Num=200");

```

曲線の端点を結ぶ直線を描く

```

Lineplot("3",[Ptstart(gr1),Ptend(gr1)],["do"]);

```

節点の 4 分の 1 で曲線を分割して破線を描く

```

Lineplot("4",[Ptstart(gr1),Ptcrv(0.25*Numptcrv(gr1),gr1)],["da"]);

```

節点の 2 等分で曲線を分割して破線を描く

```

Lineplot("5",[Ptstart(gr1),Ptcrv(0.5*Numptcrv(gr1),gr1)],["da"]);

```

節点の 4 分の 3 で曲線を分割して破線を描く

```

Lineplot("6",[Ptstart(gr1),Ptcrv(0.75*Numptcrv(gr1),gr1)],["da"]);

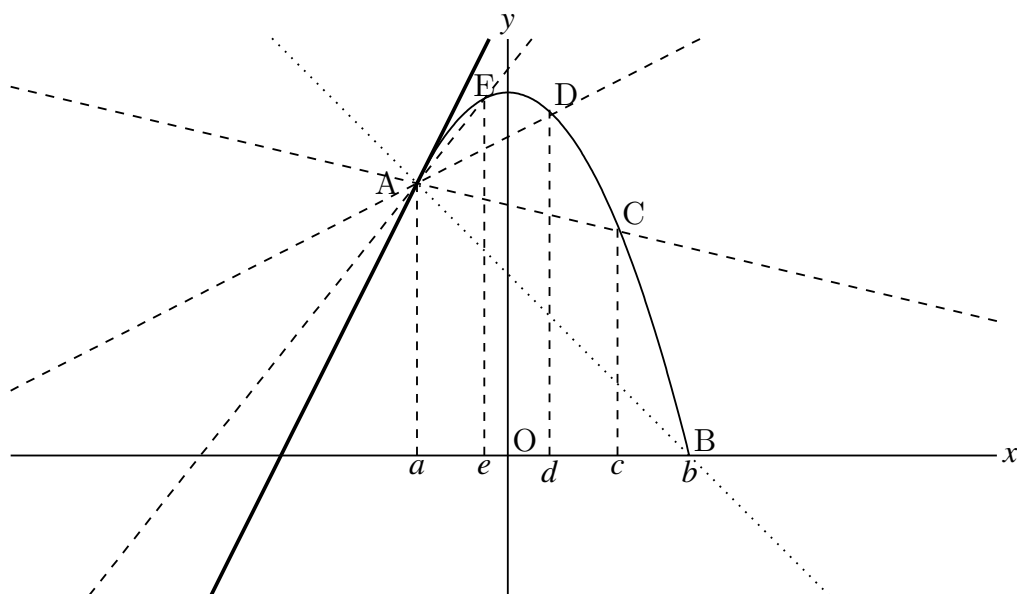
```

曲線上の点 A の接線を引く。

```

coef=Derivative("f(x)","x",G.x);
Defvar("DC=coef");
Deffun("g(x)",["regional(y)","y=DC*(x-G.x)+G.y","y"]);
Plotdata("2","g(x)","x",["dr,2"]);

```



⇒ 関数一覧

関数 Paramoncrv(点の座標, 曲線の名前)

機能 曲線上の点のパラメータ値を返す。

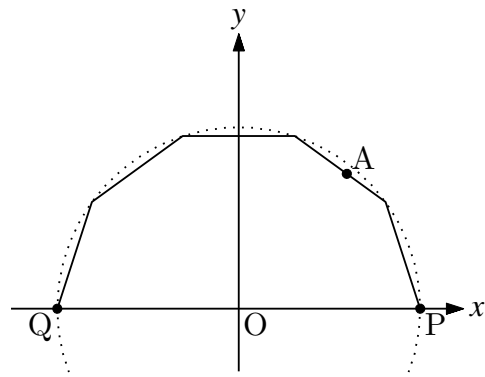
説明 曲線は折れ線として描かれるが、曲線上の各点はこの折れ線の節点を基準としたパラメータ値を持つ。パラメータ値は整数部分が節点の番号，小数部分が節間の位置を表す。

例：図のような点 P から Q に至る円周上の 5 等分点を節点とする折れ線 sg1 において， n 番目の線分上の点は $n \leq t \leq n+1$ の範囲のパラメータ値を持つ。

図の点 A は 2 番目の線分上にあり，パラメータ値は 2.45 である。この値は

```
Paramoncrv(A.xy,"sg1");
```

によって得られる。

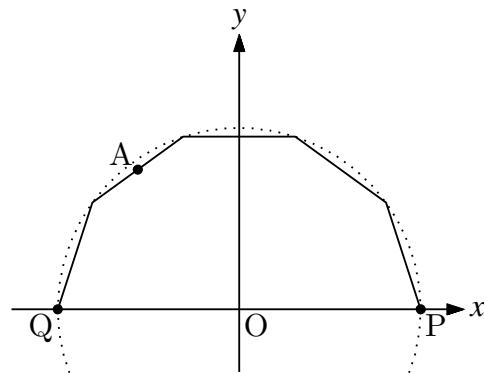


関数 Pointoncrv(点のパラメータ値, PD)
機能 曲線上のパラメータ値を持つ点の座標を返す。
説明 曲線（折れ線）上の節点を基準としたパラメータ値により点の位置が定まる。

例：図のような点 P から Q に至る半円周上の 5 等分点を節点とする折れ線 sg1 において、パラメータ値 4.5 を持つ点 A は 4 番目の線分の中点である。したがって

```
A.xy=Pointoncrv(4.5,"cr1");
```

によって、点 A を中点に置くことができる。



[⇒ 関数一覧](#)

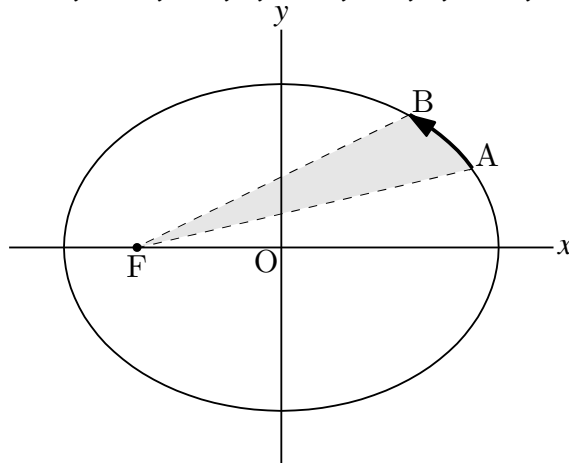
関数	Ptcrv(n, プロットデータ)
機能	曲線プロットデータの n 番目の節点を返す
説明	Cindyscript の PD_n と同じ

例：楕円上の点で分割する。あらかじめ必要な点を作図しておく。

```

Circledata([0,P],["do","Num=100","notex"]);
Scaledata("1","crOP",4/3,1);
F.xy=[-sqrt(7),0];
A=Ptcrv(9,sc1);
B=Ptcrv(16,sc1);
Listplot("1",[A,F,B],["da"]);
Partcrv("1",A,B,"sc1",["dr,3"]);
Shade(["part1","sg1",0.1);
Arrowhead(B,"sc1",[1.5]);
Letter([A,"ne","A",B,"ne","B",F,"s2","F"]);

```



[⇒ 関数一覧](#)

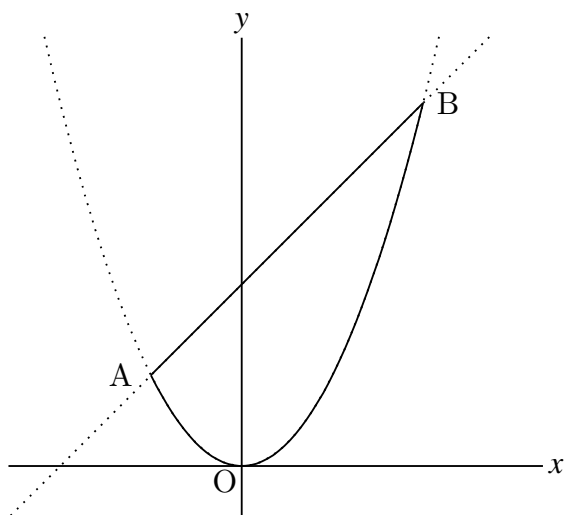
関数	Ptstart(プロットデータ), Ptend(プロットデータ)
機能	プロットデータの最初の点, 最後の点を取得する。
説明	プロットデータの最初の点, 最後の点の座標を返す。

例：定義域を限定したグラフの両端の点を取得し線分 AB を引く。

```

Deffun("f(x)",["regional(y)","y=x^2","y"]);
Plotdata("1","f(x)","x=[-1,2]",["dr"]);
Lineplot("2",[Ptstart(gr1),Ptend(gr1)]);

```



[⇒ 関数一覧](#)

関数 ReadOutData(ファイル名)

機能 外部データを読み込む

説明 Scilab の WriteOutData で作ったプロットデータ列のデータファイルを読み込む。
引数を省略した場合は、Fhead で定義したファイル名のテキストファイルから読み込む。ファイル名にはコンマで区切ってパスを与えることができる。たとえば、

```
ReadOutData("/datafolder","file.txt");
```

関数 Sprintf(実数, 長さ)

機能 小数点以下の長さを固定した文字列に変換

説明 実数を、小数点 n 位までの数とした文字列に変換する

例 Sprintf(pi,2) は 3.14 を返す

 Sprintf(pi,7) は 3.1415927 を返す

注) pi は Cindyscript の予約変数で、円周率を表す。

関数 WritetoSci(引数) ,WritetoScibody(引数)

機能 Scilab 用のソースファイルに書き出す

説明 ソースファイルへの書き出し方に 3 つのタイプがある

・ 引数なしの場合

Fhead で定義したファイル名に拡張子 .sci を付加して全体を書き出す。

末尾にコメントアウトした quit() を付加する。

- WritetoSci の引数に次の数を渡す

WritetoSci(1) : WritetoSci() と同じ

WritetoSci(2) : 全体を書き出し、末尾に quit() を付加する。

WritetoSci(3) : WritetoScibody() と同じ。body 部分のみを書き出す。

- ファイル名を引数とする

引数のファイル名で書き出す。ただし、拡張子 .sci をつける必要がある。

- 引数を "sh" とする

WritetoSci(2) と同じ。

- WritetoSci(filename,"sh")

引数のファイル名で shell (bat) コマンド用のファイルを書き出す。

- WritetoSci(数字 ,filename) は不可 (エラーとなる)

関数 Makeshell(ファイル名) / Makebat(ファイル名)

機能 Mac の場合はシェルフファイル、Windows の場合はバッチファイルを書き出す。

説明 書き出されるファイルは次の通り。

- 書き出される場所とファイル名は、Shellparent / Batparent で指定したもの。
- 内容は、Shellchild / Batchild に、子プロセスの引数として Fhead と引数のファイル名を付加したもの。たとえば、Shellchild="sh ketcindy.sh" で、Fhead="fig" , 引数が "fig2tex" であれば

```
#!/bin/sh
cd /Users/Hoge/Desktop/KeTCindy
sh ketcindy.sh fig fig2tex
exit 0
```

が書き出される。2行目のディレクトリ (フォルダ) 名は、Dirwork で指定したもの。ただし、上記の fig は fig.sci のことだが、fig.sci は出力されないので、別途 WritetoSci(2) で書き出す必要がある。

関数 Textformat(数, 桁数)

機能 小数点以下の桁数を指定して数を文字列化する。

説明 Cindyscript の組み込み関数にも、format() という同様の関数があるが、こちらは、数のリストにも対応する。

例：円周率を小数点以下 5 位までで文字列化する。

```
Textformat(pi,5);
```

戻り値は, 3.14159

例：円周率と、ネピア数をリストにして、共に小数点以下 5 位までで文字列化したりリストを返す。

```
Textformat([pi,exp(1)],5);
```

戻り値は, [3.14159,2.71828]

関数 Viewtex()

機能 T_EX のソースファイルを書き出す。引数なし。

説明 グローバル変数 Fhead で定義したファイル名に "main" を付加した T_EX のソースファイルとバッチファイル (Mac の場合はシェルフファイル) を作成する。

Fhead のほか、ディレクトリ指定などのため、次のグローバル変数に必要事項が代入されていることが条件である。

Dirwork=作業フォルダのパス

Shellparent=親プロセス (kc.sh / kc.bat) を置くディレクトリのパス

Shellchild=子プロセス名

Libname=Scilab のライブラリ ketpicsciL5 へのパス

たとえば、Fhead="fig" とすると

- ・図を表示する fig.sci を Dirwork で指定したフォルダにを生成する。(すでにある場合は上書き)

- ・Dirwork で指定したフォルダに figmain.tex を生成する。(すでにある場合は上書き)

figmain.tex は T_EX のプリアンブルと `begin{document}` を設定したソースファイルで、中には `input{"fig.tex"}` が書かれており、これをコンパイルすれば作成した図を表示することができる。

- ・親プロセスの生成 (すでにある場合は上書き)

ketcindy.sh (Mac) ketcindy.bat(Windows) を子プロセスとして figmain.tex と fig.tex を引数として実行するプロセス kc.sh(Mac) kc.bat (Windows) を作業ディレ

クトリに生成する。これを実行すると図を表示する PDF ファイルが作られる。

以上の手続きにより、Viewtex() を実行すれば、デスクトップ上に生成された kc.sh / kc.bat を使って、fig.sci の生成から、fig.tex の作成、T_EX のコンパイルまでの一連の操作を自動実行して、図を表示することができることになる。kc.sh / kc.bat の実行は、kc() で行うことができる。

関数	Workprocess()
機能	作図の経過を取得する
説明	作図ツールを用いた作図の経過を取得する。 println(Workproccess()); とすると、コンソールに作図手順が表示される。

[⇒ 関数一覧](#)

3.7 その他

関数 Assign(文字列, 文字, 文字)

機能 文字列の中のある文字を他の文字で置き換える

説明 第1引数の文字列中の第2引数の文字を, 第3引数の文字で置き換える。

第3引数が数値の場合, 文字列に変換される。

第2引数と, 第3引数をリストにして, 複数の置き換えをすることができる。

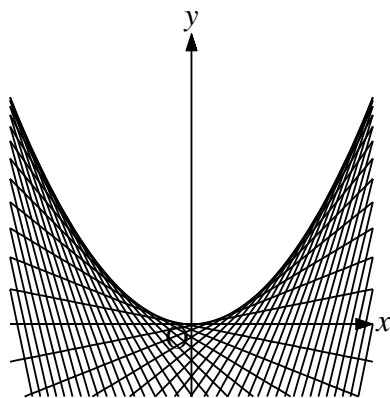
例: $a*x$ を $1.3*x$ とした文字列を返す。次のいずれも同じ結果になる。

```
Assign("x^2+a*x","a","1.3");
```

```
Assign("x^2+a*x","a",1.3);
```

例: 直線 $y = bx - b^2$ の係数 b を変化させて描き, 包絡線をうかびあがらせる。

```
repeat(50,t,  
  cb=t/5-5;  
  Plotdata(text(t),Assign("b*x-b^2","b",cb),"x");  
);
```



例: 文字で表された x と y の係数をまとめて数値で置き換える。

```
Assign("a*x^2+b*x",["a",1,"b",2]);
```

[⇒ 関数一覧](#)

関数 BBdata(ファイル名,option)

機能 画像ファイルのサイズを求める

説明 TeX 文書において, inputgraphics コマンドで画像を貼り込むときの BB サイズを求める。

TeX 処理系の extractbb を用いて画像ファイルから BB データを作り, テキストファイルとして作業ディレクトリに書き出す。これを読んで, コンソールに includegraphics のコマンドを書き出す。

option は, 幅または高さの指定。

"w=40mm" で width=40mm が, "h=40mm" で height=40mm が付加される。

例: pic.pdf のサイズを求める。

```
BBdata("pic.pdf")
```

を実行すると, コンソールに includegraphics のコマンド文

```
\includegraphics[bb=.....]{pic.pdf}
```

が表示される。これをそのままコピーすればよい。

なお, bb の値は整数値ではなく, 高精細の値を小数点以下 2 桁に四捨五入して示される。

画像ファイルは, PDF に限らず, PNG, JPG などでもよい。

例: BBdata("fig.jpg",["h=40mm"]);

で

```
\includegraphics[bb=0.00 0.00 578.16 592.56,height=40mm]{fig.jpg}
```

が表示される。

関数 Changework(パス名)

機能 作業ディレクトリを指定 (変更) する

説明 作業ディレクトリは, Initialization スロットの KETlib ページに Dirwork= で指定されているが, これを変更する。Draw スロットの, Ketinit() の前に記述することにより, Ketlib ページの記述を変更しなくてすむ。

これにより, 作成したファイルを他の人とやり取りしやすくなる。

関数 Com0th(文字列)

機能 Scilab の Openfile の前に置くコマンド (文字列) を定義する。

説明 例: Com0th("Setax('a')");

Openfile の前の部分の先頭に書き出す。これにより, KETCindy でサポートされていない Scilab 版 KETpic のコマンドを利用することができる。

関数 Com1st(文字列)
機能 Scilab の Openfile の前に置くコマンド (文字列) を定義する。
説明 例: Com1st("Setax('a')");
これにより, KeTCindy でサポートされていない Scilab 版 KeTpic のコマンドを利用することができる。

関数 Com2nd(文字列)
機能 Scilab の Openfile のあとに置くコマンド (文字列) を定義する。

関数 Com2ndpre(文字列)
機能 Scilab の Openfile のあとに置くコマンド (文字列) を定義する。 Openfile の直後 (グループの先頭に) 書き出す。

関数 Figpdf(option)
機能 出力枠サイズの PDF を作る。

説明 KeTCindy では, 通常, 出力された fig.tex ファイルを閲覧する PDF を A4 サイズで作成する。これに対し, Figpdf() を実行すると, 出力サイズの PDF を作成する。閲覧用だけではなくワープロなどに貼り込むときにそのまま使用できるので便利である。ただし, そのための親子プロセスを生成して実行するため, 次の手続き (1)(2) が必要となる。

(1) 変数 Texparent を設定する。

これは, 出力する PDF のファイル名の指定である。たとえば,

```
Fhead="fig";  
Texparent="pic";
```

とすると, fig.tex を表示した pic.pdf が作成される。pic.pdf が目的の PDF。

Texparent は Fhead とは異なるものにする。

(2) 出力は, 「Parent」「Exekc」の順にボタンを押す。「Texview」は押さない。

なお, これらのボタンを使わずにスクリプトで実行するか, オリジナルのボタンを作る場合は, 次のコマンドを実行する。

```
Writetosci(2);  
Makeshell();  
kc();
```

Writetosci(2); と Makeshell(); で fig.sce と pic.tex が作成される。

Windows の場合は Makebat(); とする。

kc(); により kc.sh が実行され, Scilab で fig.sce から fig.tex を作るころか

とする。引数なしで

```
Windispg();
```

とすればすべてのプロットデータ列が表示される。

なお、いずれの場合も、作図したプロットデータも同時に表示される。

作図した図を全てではなく選択して表示する場合は、それらのプロットデータ名をリストにして引数とする。

たとえば、sg1, gr1, crAB が定義されているとき、

```
Windispg(["sg1","gr1"]);
```

とすれば、sg1,gr1 のみが表示される。 次の3つの関数は、プロットデータの操作で、単独で用いることもできるが、描画関数、プロットデータの操作関数のオプションに組み込まれているので、特殊な場合を除いてはほとんど利用しないと思われる。

関数	Drwline(プロットデータ)
機能	プロットデータ (文字列) を実線で描く
説明	Scilab の Drwline を書き出す 例: <code>Drwline("sABCA,1");</code>

関数	Dashline(プロットデータ)
機能	プロットデータ (文字列) を破線で描く
説明	Scilab の Dashline を書き出す 例: <code>Dashline("sABCA");</code>

関数	Dottedline(プロットデータ)
機能	プロットデータ (文字列) を点線で描く
説明	Scilab の Dottedline を書き出す 例: <code>Dottedline("sABCA");</code>

関数	helplist()
機能	ヘルプデータを作成する
説明	関数の簡単な説明データを作成する。 この関数は、Initialization スロットに書く。

[⇒ 関数一覧](#)

関数 Help(文字列)

機能 関数の使用例を取得する

説明 文字列で始まる関数の使用例をコンソールに表示する。

```
println(Help("C"));
```

のようにすると、コンソールに、次のように「C」で始まる関数の使用例が表示される。

```
CRspline("1",[A,B,C,A]);
ChangeTablestyle(["r0c0c3"],["da"]); \
Changestyle("sgAB",["da"]);
Changstyle("geoseg3","ax3d"),["notex"]); \
Circledata([A,B,C]);
Circledata([A,B],["Rng=[0,pi/2]"]); \
Com2nd("\color[cmyk]{0,0,0,0.5}"); \
Crossprod(vec1,vec2); \
```

さらに,

```
println(Help("Ci")); とすると
```

```
Circledata([A,B,C]);
Circledata([A,B],["Rng=[0,pi/2]"]);
```

だけが表示される。

```
println(Help("*")); とすると、すべての関数の使用例が表示される。
```

引数がない場合は、`Helplist(Dirlib,["+","+3d"],"helpJ")`; とみなされる。`+` は `ketcindylib` の意味。

関数 Helpkey(文字列)

機能 関数の使用例をキーワードで検索する

説明 文字列に与えたキーワードで関数の使用例を検索し、コンソールに表示する。

例：`Helpkey("直線")`; とすると、コンソールに次のように表示される。

```
IntersectsgpL("",[p1,p2],[p3,p4,p5],"draw");
IntersectsgpL("R","P-Q","A-B-C");
IntersectsgpL("R","P-Q","A-B-C","put");
```

空間の直線と平面の交点

```
Lineplot("1",[2,1],[3,3]);
Lineplot([A,B]);
```

直線データを作成

`PtonLine("C",pA,pB);`

直線上に点をとる

関数 `Indexall(str1,str2);`
機能 文字列 `str1` から `str2` を検索しその位置をすべて返す
説明 Cindyscript の `indexof()` の拡張版。`indexof()` が最初に見つかった位置を返すのに
対し、`Indexall()` は存在する位置をすべてリストにして返す。

例：`str="abcabcabc"` から `"b"` を検索する。

`indexof(str,"b")` では、2 が返る。

`Indexall(str,"b")` では、`[2,5,8]` が返る。

関数 `Ketcindylogo()`
機能 `KETCindy` のロゴを書き出す
説明 `KETCindy` のロゴを表示する `TEX` のコマンド行を書き出す。
内容は
`\def\ketcindy{\{K\kern-.20em \lower.5ex\hbox{E}\kern-.125em{TCindy}\}}`

関数 `Op(n,list or str)`
機能 リストまたは文字列から要素を抜き出す
説明 第2引数のリストまたは文字列の `n` 番目の要素（文字）を返す。
`Cindyscript` の アンダーバーの演算子 (`list_n` , `str_n`) と同様。`Silab` との整合性のため追加

[⇒ 関数一覧](#)

4 他の数式処理ソフトなどとの連携

4.1 R との連携

R は主に統計解析のためのソフトウェアで、`binorm` (二項分布), `pois` (ポアソン), `unif` (一様分布), `chisq` (カイ 2 乗), `f` (F 分布), `t` (t 分布) など、多くの確率分布をサポートしている。

正規分布 (normal distribution) では

`dnorm` 確率密度関数

`pnorm` 分布関数

`qnorm` 分布関数の逆関数

`rnorm` 乱数発生

というように、分布名の頭に `d`, `p`, `q`, `r` をつけると上記 4 つの関数が得られる。

各分布には、自由度などの引数があり、たとえば、平均 `m`, 標準偏差 `s` の正規分布 (の密度関数) は `dnorm(x, m, s)` となる。

KeTCindy では、`kc.bat/sh` によってコマンドを R に渡し、結果をテキストファイルで受け取る。このとき、R とのやりとりで、次のようなファイルが作業ディレクトリに作成される。

拡張子 `r` : `r` 用のファイル

拡張子 `dat`, 拡張子 `txt` : データファイル

このデータのやり取りに関する次のオプションがある。

オプションなしまたは、`"` のとき

i) データファイルがなければ、新しく作る

ii) データファイルが既にあればそれを読み込む

`"m"` のとき、強制的にデータファイルを作り直す。

`"r"` のとき、すでにあるデータファイルを読み込む。

このとき、ファイルの読み書きで不具合があると、数秒の後「`==> file.txt not generated (5 s)`」のようなエラーメッセージがコンソールに表示される。このような場合は作業ディレクトリの設定などを確認していただきたい。この待ち時間については、`Wait` オプションで設定することもできる。

関数 `Boxplot`(名前, データ, 垂直位置, 高さ, option)

機能 箱ひげ図を描く

説明 データは、直接的に変数で渡す場合とファイルから読み込む場合がある。

例：乱数で作成した 5 未満の実数のデータを箱ひげ図にする。

```
dt1=apply(1..100,5*random());  
Boxplot("1",dt1,1,1/2);
```

例：外部ファイルとして用意したデータを読み込んで箱ひげ図にする。データファイルは csv 形式とする。

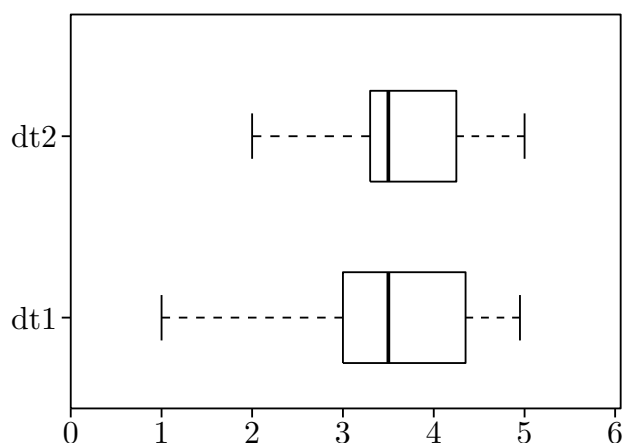
```
Boxplot("2","datafile.csv",3,1/2);  
Boxplot("1",dt1,1,1/2);
```

複数列から成る csv ファイルを読み込むには、Readcsv を使う。csv ファイルは、ketwork フォルダに入れておく。戻り値は読み込んだファイル。

データの値を画面に入るように調節するには、次のようにリストの計算を利用すればよい。

また、Framedata2(), Rulerscale() を併用することで目盛を入れることができる。Framedata2() のために、表示領域の対角点 A,B を Cinderella の作図ツールで作図しておく。

```
data=Readcsv("datafile.csv");  
dt1=apply(data,#_1);  
dt2=apply(data,#_2);  
Boxplot("1",dt1/20,1,1/2);  
Boxplot("2",dt2/20,3,1/2);  
Framedata2("1",[A,B]);  
Rulerscale(A,["r",0,6,1],["f",1,"\mbox{dt1}",3,"\mbox{dt2}"]);
```



[⇒ 関数一覧](#)

関数 CalcbyR(変数名, コマンド列, option)
機能 R のコマンドを実行して結果を返す
説明 バッチファイル kc.bat / シェルファイル kc.sh を利用して R とデータをやり取りし、計算結果を取得する。

例：R を用いて標準正規分布から 10 個の乱数を発生し、平均値と標準偏差を求めてコンソールに表示する。

```
cmdL=[ "=rnorm",[10] ];
CalcbyR("dt",cmdL);
nx=length(dt);
mx=sum(dt)/nx;
sx=sqrt(dt*dt/nx-mx^2);
println("データ："+dt);
println("平均："+format(mx,4)+"      標準偏差："+format(sx,4));
```

1 行目の `cmdL=["=rnorm",[10]]` で、標準正規分布から 10 個の乱数を発生するコマンド列を定義。

2 行目で、R で計算した結果がリスト `dt` に入る。

例：R を用いて $N(50,5^2)$ から 10 個の乱数を発生し、平均と不偏分散も R で計算してその結果をコンソールに表示する。

```
cmdL=[
```

```

    "tmp1=rnorm",[10,50,5],
    "tmp2=mean",["tmp1"],
    "tmp3=var",["tmp1"],
    "=c(tmp1,tmp2,tmp3)",[]
];
CalcbyR("rd",cmdL);
dt=rd_(1..(length(rd)-2));
mx=rd_(-2);
vx=rd_(-1);
println("データ："+dt);
println("平均："+format(mx,4)+"    不偏分散："+format(vx,4));

```

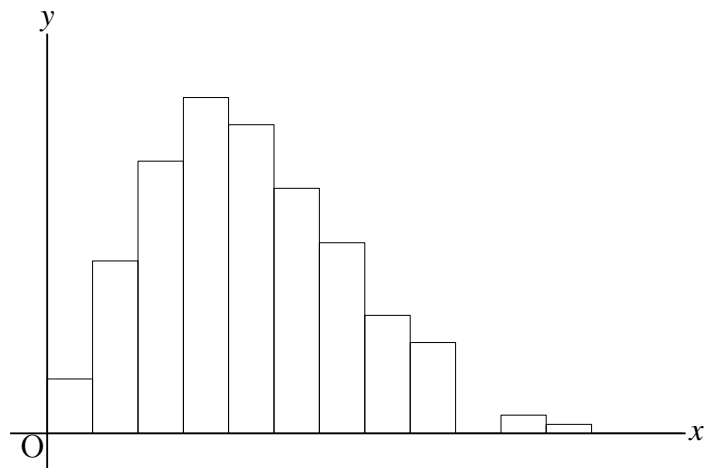
CalcbyR() によって、データと平均、不偏分散からなるリストが作成されるので、mx に平均、vx に不偏分散を代入している。rd_₍₋₁₎ は、リスト rd の末尾の要素。

例：Rでポアソン分布から 200 個の乱数を取り、標本平均の分布の様子＝分散が小さくなって、正規分布に近づいている様子＝をヒストグラムで見る。

```

cmdL=[
    "tmp1=rpois",[200,5],
    "tmp2=mean",["tmp1"],
    "tmp3=var",["tmp1"],
    "=c(tmp2,tmp3,tmp1)",[]
];
CalcbyR("rd",cmdL);
dt=rd_(3..length(rd));
nn=length(dt);
mx=rd_1;
vx=rd_2*(nn-1)/nn;
sx=sqrt(vx);
println(dt);
println(["m="+format(mx,4),"v="+format(vx,4)]);
Setscaling(1/5);
Histplot("1",dt,["Breaks=seq(0,14,1)","dr,0.5"]);

```

例：ポアソン分布で乱数を 2000 個発生させ、10 個ずつの平均を R で計算し、

```
cmdL=[
  "tmp1=rpois", [2000,5],
  "tmp2=c()", [],
  "for(k in 1:200){", [],
  "  tmp=tmp1[(10*(k-1)+1):(10*k)]", [],
  "  tmp2=c(tmp2,mean(tmp))", [],
  "}", [],
  "=tmp2", []
];
CalcbyR("rd2",cmdL);
Setscaling(1/10);
Histplot("2",rd2);
```

[⇒ 関数一覧](#)

関数 Histplot(name,data,option)

機能 R を利用してヒストグラムを描く

説明 data はリストにして作成するか、外部ファイルから Readcsv() で読み込む。

階級範囲（ブレイクポイント）は、通常スタージェスの公式によるが、オプションで、

”breaks=[0,10,20,30,40,50,60,70,80,90,100]”

などと指定することもできる。

この他のオプションは

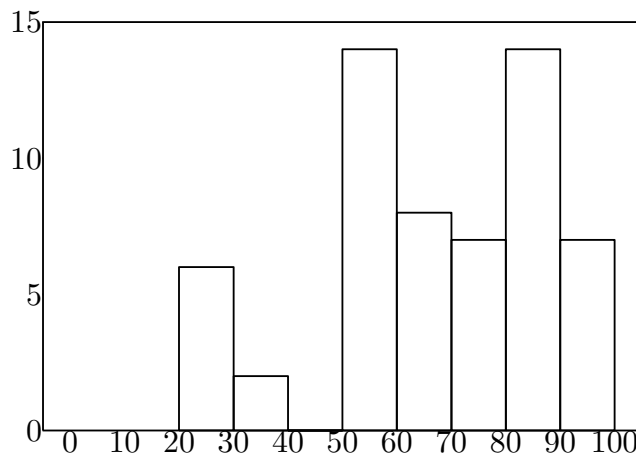
”Den=yes/no”：密度の指定（初期値は no）

”Rel=yes/no”：相対度数にする/しない（初期値は no）

例：csv ファイル（datafile.csv）を読み込み、ヒストグラムを作る。Framedata2()

と `Rulerscale()` を併用して、目盛付きの枠の中に表示する。表示枠の対角点 A,B は Cinderella の作図ツールで作図しておく。

```
Addax(0);
Setscaling(5);
Setunitlen("0.6mm");
data=Readcsv("datafile.csv");
Histplot("1",data,[""]);
Framedata2("1",[A,B]);
Rulerscale(A,["r",0,100,10],["r",0,15,5]);
```



2 行目と 3 行目は、データに合わせて縦方向を 5 倍にし、TeX の単位長を 0.6mm にしている。

Den,Rel オプションを yes にしたときは、`Setscaling(100)` くらいにするのがよい。csv ファイルが複数のデータからなる場合は、

`dt1=apply(rc1,#_1);` として、リストの第 1 要素を取得する。第 2 要素のヒストグラムであれば `#_2` とする。

[⇒ 関数一覧](#)

関数	<code>PlotdataR(name, 式, 変数)</code>
機能	R の関数のグラフを描く
説明	Cindyscript の組み込み関数にはない関数のグラフを R を利用して描く。

例：平均 5, 標準偏差 2 の正規分布の密度関数と分布関数のグラフを描く。

```
PlotdataR("1", "dnorm(x,5,2)", "x=[0,10]");
```

```
PlotdataR("2", "pnorm(x,5,2)", "x=[0,10]");
```

例：標準正規分布のグラフ上の点と x 軸を結んだ線分を描く。

点 A,B は Cinderella の作図ツールで作図しておき、点 A をグラフ上のおよその位置に置いてから実行する。

```
PlotdataR("1","dnorm(x)","x=[-5,5]");
Putoncurve("A","grR1",[-3,3]);
Putpoint("B",[A.x,0]);
Listplot("1",[A,B]);
```

2 行目の最後の引数の [-3,3] は、その範囲を動かすことを意味する。

A はグラフ上を動かすことができ、B はそれに伴って動く。ただし、少し動かす度に バッチ/シェル ファイルを実行するので、煩雑な場合は、Plotdata() の行をコメント化してから点 A を動かしたあと再実行するとよい。

例：上と同様で、x 軸上の点を自由点 A とし、曲線上に B を置く。

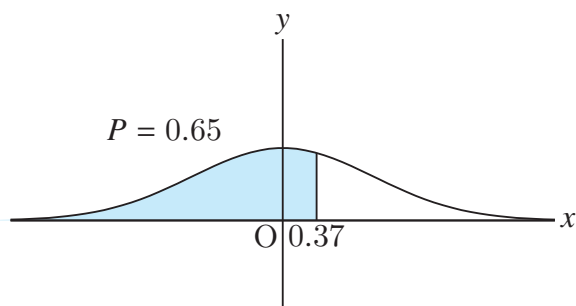
```
PlotdataR("1","dnorm(x)","x=[-5,5]");
PlotdataR("1","dnorm(x)","x=[-5,5]");
A.xy=[A.x,0];
Lineplot("1",[A,A+[0,1]],["nodisp"]);
Putintersect("B","grR1","ln1");
Listplot("1",[A,B]);
```

例：前の例のグラフで、AB の左側に Shade をかけ、Shade の部分の面積を求める。

P の値を表示する位置に、Cinderella の作図ツールで点 C をとっておく。

```
tmp1=[0.2,0,0,0];
tmp2=Colorcmyk2rgb(tmp1);
SetColor(tmp1);
SetColor([0.2,0,0,0]);
Shade(["en1"],["color->" + text(tmp2)]);
SetColor("black");
PlotdataR("1","dnorm(x)","x=[-5,5]",["Num=100"]);
Putpoint("A",[0,0],[A.x,0]);
Lineplot("1",[A,A+[0,1]],["nodisp"]);
Putintersect("B","grR1","ln1");
Listplot("1",[A,B]);
```

```
Listplot("2", [[-5,0],[5,0]], "nodisp");
Enclosing("1", ["Invert(grR1)", "sg2", "sg1"], [B, "notex"]);
tmp=0.5+Integrate("grR1", [0,A.x]);
Expr([A, "s", text(A.x), C, "e", "P="+text(tmp)]);
```



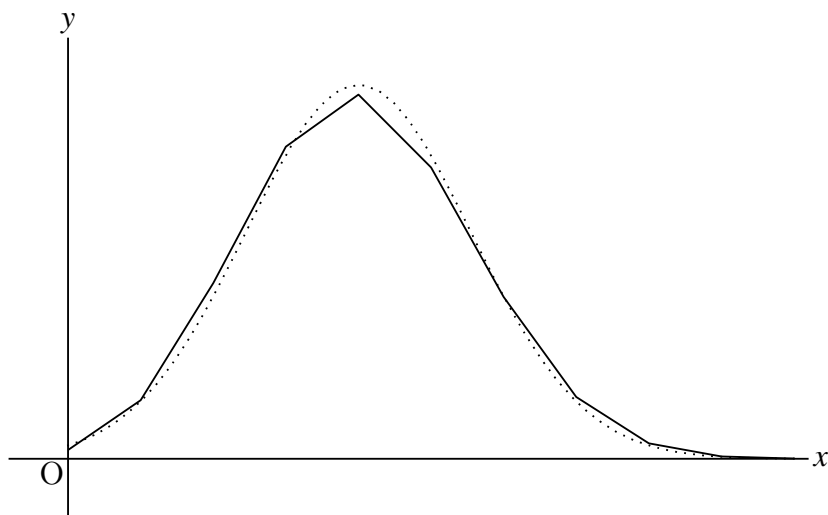
Shade() はプロットデータを定義しないので、先にかけておく塗ってから線を描くので、にじみがなくきれいに仕上がる。("en1"は Enclosing("1"・・・) でできるプロットデータ)

[⇒ 関数一覧](#)

関数	PlotdiscR(name, 式, 変数)
機能	R を利用して離散型のグラフを描く
説明	dbinom (二項分布), dpois (ポアソン分布), dgeom (幾何分布) など離散型確率分布のグラフを描く。

例：二項分布のグラフと正規分布のグラフを比較する。

```
Setscaling(20);
PlotdiscR("1", "dbinom(k,10,0.4)", "k=[0,10]");
PlotdataR("1", "dnorm(x,10*0.4,sqrt(10*0.4*0.6))", "x=[0,10]", ["do"]);
```



関数部分が長くなるときは、Assign() を利用して次のように書くこともできる。

```
tmp="dnorm(x,n*p,sqrt(n*p*(1-p)))";
tmp=Assign(tmp,["n",10,"p",0.4]);
PlotdataR("2",tmp,"x=[0,10]",["do"]);
```

次は、ポアソン分布および幾何分布のグラフ。

```
PlotdiscR("2","dpois(k,4)","k=[0,10]");
PlotdiscR("3","dgeom(k,0.3)","k=[0,10]");
```

[⇒ 関数一覧](#)

関数 Readcsv(path,filename,option)

機能 R を利用して csv ファイルを読む。

説明 R を使って csv ファイルを読みこむ。戻り値は読み込んだデータのリスト。

第1引数の path は、ファイルを network フォルダに置いた場合は省略することができる。そうでない場合は、フルパスで指定する。たとえば、"/Users/Hoge/Desktop" option は、"Flat=" で、"Flat=y" の場合は、読み込んだデータをリスト化したときに平滑化（1次元のリスト）にする。デフォルトは "Flat=n"

また、R を使わず Scilab を使って読み込むこともできる。その場合は、option に "Use=S" をつけるが、Mac で作成したファイルで、改行コードが CR のみの場合はうまく読み込むことができない。Windows 標準の CR+LF なら読み込める。

利用例は Boxplot() などの例を参照のこと。

関数 Scatterplot(name,filename,option)

機能 2次元データを読み込み、散布図を描く

説明 外部ファイル filename (csv 形式) を読み、散布図を描く。

オプションは "Reg=no" : 回帰直線を描くかどうか (yes/no) 初期値は yes

A, B 表示枠の対角点 (左下と右上の点) (Cinderella で作図)

C 相関係数と回帰直線の式を表示する点

"Size=n" : 点の大きさ。Cinderella の描画面には反映されない。

描画面の点も大きくする場合は, "size->n" オプションを追加する。

例: data.csv を読んで散布図を描き、回帰直線を引く。

```
Scatterplot("1","data.csv",["Reg=yes",A,B,C,"Size=5"]);  
Rulerscale(A,["r",0,10,1],["r",1,10,1]);
```

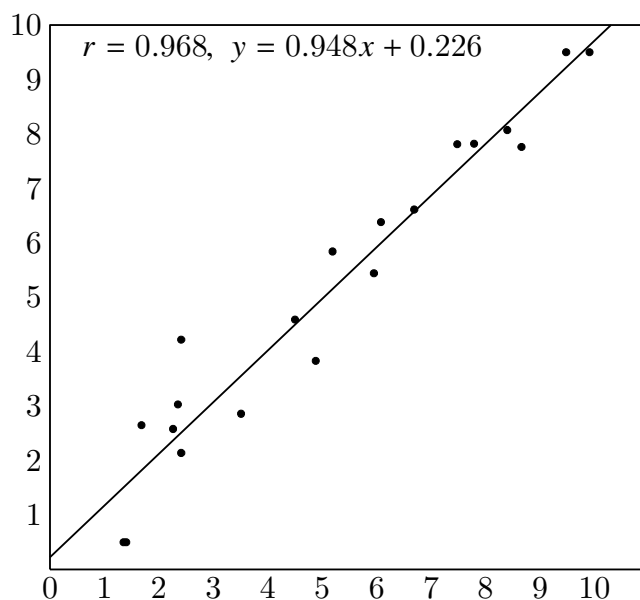
option の点 A,B は Cinderella の作図ツールでとった表示領域の対角点。(A が左下。なお、対角点の名称は A,B には限らない。)

回帰直線を描く場合 ("Reg=yes") は、回帰直線が枠からはみ出すので、A を SW, B を NE に一致させるとよい。

C は相関係数と回帰直線の式を表示する点として Cinderella の作図ツールで取る。これも、名称は C には限らない。

```
Rulerscale(A,["r",0,10,1],["r",1,10,1]);
```

で表示枠に目盛を打っている。"r" は等幅, "f" は Tickmark と同じ指定。



4.2 Maxima との連携

Maxima は数式処理ソフトで、KeTCindy においては微積分の計算など、Cindyscript では不十分な点を補うことができる。

KeTCindy では、kc.bat/sh によってコマンドを Maxima に渡し、結果をテキストファイルで受け取る。このとき、Maxima とのやりとりで、次のようなファイルが作業ディレクトリに作成される。

拡張子 max : Maxima 用のファイル

拡張子 txt : データファイル

このデータのやり取りに関する次のオプションがある。

オプションなしまたは, "" のとき

i) データファイルがなければ、新しく作る

ii) データファイルが既にあればそれを読み込む

"m" のとき、強制的にデータファイルを作り直す。

"r" のとき、すでにあるデータファイルを読み込む。

このとき、ファイルの読み書きで不具合があると、数秒の後「==> file.txt not generated (5 s)」のようなエラーメッセージがコンソールに表示される。このような場合は作業ディレクトリの設定などを確認していただきたい。この待ち時間については、Wait オプションで設定することもできる。

関数 CalcbyM(name, コマンド, option)

機能 Maxima のスクリプトを実行する

説明 第2引数は Maxima で実行するコマンド。

コマンドと引数リストの繰り返しからなるリスト (例えば cmdL) を作って、一度に実行する。

戻り値はない。(未定義値) 結果は、コマンドリストの最後に記述した変数 (引数は空リスト) の値が name で指定された変数に代入される。複数の結果を戻すときは、:: で区切って記述するとリストにして代入される。

例: $\sin x$ とその導関数を表示する。結果は 変数 fdf に f と df のリストが代入される。

```
cmdL=[
  "f:sin(x)", [],
  "df:diff", ["sin(x)", "x"],
  "f::df", []
];
```

```
CalcbyM("fdf",cmdL);
println(fdf);
```

実行すると、コンソールに、 $[\sin(x), \cos(x)]$ と表示される。

例：2 次方程式 $x^2 - x - 4 = 0$ の解を求める。

```
cmdL=[
  "ans:solve",["x^2-x-4","x"],
  "ans",[]
];
CalcbyM("ans",cmdL);
println("ans="+ans);
```

コンソールには

```
ans=[x = -(sqrt(17)-1)/2,x = (sqrt(17)+1)/2]
が表示される。
```

応用例 1：曲線の接線を引く

$f(x) = \frac{e^x + e^{-x}}{2}$ の、 $x = a$ における接線の方程式を作る。

Maxima でその処理を行うコマンドを定義し、CalbyM で実行する。

```
fx="(exp(x)+exp(-x))/2";
cmdL=[
  "df:diff",[fx,"x"],
  "c:ev",["df","x=a"],
  "b:ev",[fx,"x=a"],
  "eq:c*(x-a)+b",[],
  "eq",[]
];
CalcbyM("tn1",cmdL);
println(tn1);
```

コンソールには

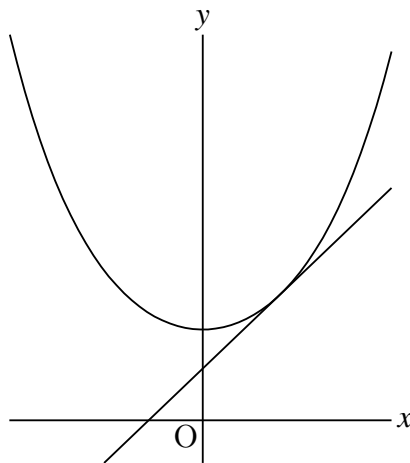
```
(%e^a-%e^-a)*(x-a))/2+(%e^a+%e^-a)/2
```

が表示される。

この、CalbyM の戻り値 `tn1` を用いて、曲線上の 1 点 A における接線のグラフを描く。以下のスクリプトを追加する。なお、点 A を Cinderella の作図ツールで適当なところにとっておく。

```
tn1=Assign(tn1,["%e^a","exp(a)","%e^-a","exp(-a)"]);
Plotdata("1",fx,"x");
Putoncurve("A","gr1");
tmp=Assign(tn1,["a",A.x]);
plotdata("2",tmp,"x",["Num=2"]);
```

1 行目では Maxima で作成した式を、Cindyscript でプロットできる式にしている。



なお、接線の方程式を求めるだけであれば、`Mxfun()` を使うこともできる。`Mxfun()` の解説を参照のこと。

応用例 2：パラメトリックの場合の接線

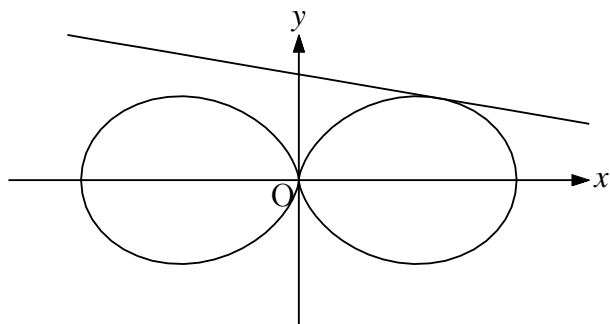
媒介変数の値を決めるために、点 A を Cinderella の描画面の x 軸上にとっておき、その x 座標を媒介変数 t の値とする。スライダを作ってもよい。

```
fn="3*cos(t)^2*[cos(t),sin(t)]";
cmdL=[
  "f:",[fn],
  "df:diff",["f","t"],
  "df:trigsimp",["df"],
  "tn:f+s*df",[],
  "tn",[]
];
CalcbyM("tn2",cmdL);
```

```

Paramplot("1",fn,"t=[0,2*pi]",["Num=100"]);
gn=Assign(tn2,["t",A.x]);
Paramplot("2",gn,"s=[-3,3]");

```



cmdL で定義している Maxima のコマンド (trigsimp など) については, Maxima の解説書などを参照されたい。

[⇒ 関数一覧](#)

関数 Example("Mxfun", 文字)
機能 Mxfun の使用例を表示。文字は "a", "b", など。
説明 たとえば, Example("Mxfun","a") とすると, Mxfun の使用例がコンソールに表示される。

関数 Mxbatch(ファイル名)
機能 Maxima のファイルを実行するコマンド作る
説明 Dirlib で指定されたフォルダの中の maximaL フォルダにあるファイルを実行するための, CalcbyM 用のコマンドを作成する。

例: cmd=Mxbatch("fourier_sec")

を実行すると, cmd に

```
[batch, ["/Applications/ketcindy/ketlib/maximaL/fourier_sec.max"]]
```

が代入される。そこで,

```
CalcbyM("ret",cmd,[]);
```

を実行すれば, fourier_sec.max が実行されて, 結果を ret に取得できる。

fourier_sec.max に続けて, コマンド列 cmd2 を実行することもでき, その場合は

```
cmdL=Concat(Mxbatch("fourier_sec"),cmd2);
```

```
CalcbyM("ret",cmdL,[]);
```

とする。(Concat() はリストを結合する Cindyscript の関数)

[⇒ 関数一覧](#)

関数 Mxfun(name, 式, リスト, option)
機能 Maxima の関数を実行する
説明 第2引数の「式」は Maxima の関数名。第3引数のリストは関数に渡す引数のリスト。

戻り値は、第1引数の式に1つでも文字があると文字列となる。すべて数字(+,-, .を含む)の場合は16桁以下であれば数、それ以上の場合は文字列となる。また、戻り値は、変数 mxname にも代入される。

オプションに "Disp=no" をつけると、結果をコンソールに表示しない。

例：10! を求める。

```
Mxfun("1","10!",[],[""]);
```

を実行すると、コンソールに mx1 is 362880 と表示される。この値は変数 mx1 に代入されているので、

```
drawtext([0,1],mx1);
```

とすれば Cinderella の描画面上に表示される。

また、戻り値を別の変数に代入して使うこともできる。

```
fact10=Mxfun("1","10!",[],[""]);
```

```
drawtext([0,1],fact10);
```

例： $f(x) = \sin x$ を微分する

```
Mxfun("1", "diff",["sin(x)","x"])
```

とすると

```
diff(sin(x),x)
```

というコマンドを Maxima に渡して、戻り値を Cindy の変数 mx1 に代入する。

```
Mxfun("1", "diff(sin(x),x)",[])
```

と、第1引数にまとめても同じ結果になる。ただし、この場合、第2引数は空リストとする。

文字列を引数とする場合、例えば、文字列を連結するコマンド concat では、

```
concat("a","b")
```

とするが、Cindyscript の文字列の処理の関係で、第 1 引数ではこの形で記述できない。

したがって、このような場合は、第 2 引数を使って

```
Mxfun("1","concat",["a","b"]) とすればよい。
```

Cindyscript の微分との違い

Cindyscript でも微分はできる。たとえば、

```
f(x):=sin(x);  
g(x):=d(f(#),x);  
plot(g(#));
```

とすると、 $\cos(x)$ のグラフが描かれる。

しかし、Cindyscript の微分が、微分の定義による数値計算であるのに対し、Maxima では数式処理として微分ができる。

その意味の違いは、次のスクリプトで確かめられる。

```
f(x):=sin(x);  
g(x):=d(f(#),x);  
println(g(x));
```

では、コンソールに表示されるのは未定義値 (_ _) である。
一方、

```
Mxfun("1", "diff",["sin(x)","x"]);  
println(mx1);
```

では、コンソールに $\cos(x)$ と表示される。

mx1 は文字列であるので、

```
g(x):=parse(mx1);
```

とすれば、 $g(x)$ を導関数とすることができ、 $\text{plot}(g(\#))$ でグラフを描くことができる。

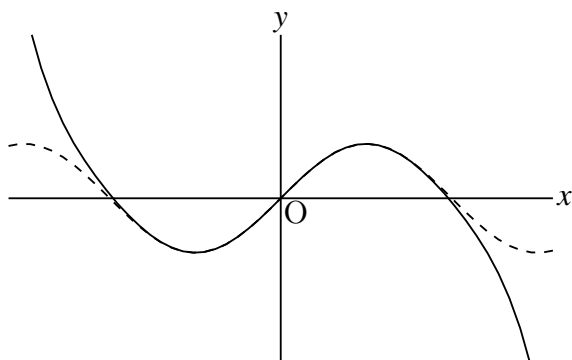
また、Cindyscript の微分では、3 階か 4 階までの導関数が計算上の限度であるのに対し、Maxima なら何階でも微分ができるので、テイラー展開などで有利である。

例 $\sin x$ の テイラー展開を行い、グラフを表示する。

```

Mxfun("1","taylor",["sin(x)","x",0,7],[""]);
Plotdata("1","sin(x)","x",["da"]);
Plotdata("2",mx1,"x");

```



Mxtex() を用いて、Mxfun() の結果の mx1 を TeX 書式にして表示することもできる。

```
Expr([[1,2],"e",Mxtex("1",mx1)]);
```

を追加すれば [1,2] の位置に式が表示される。

応用例：接線の方程式を作る

$f(x) = \frac{e^x + e^{-x}}{2}$ の、 $x = a$ における接線の方程式を作る。

関数式を文字列にしておき、Assign() を用いて変数 x を a に変えれば、 $f(a)$ の式を作ることができる。導関数についても同様にする。

```

fx="(exp(x)+exp(-x))/2";
gx=Mxfun("1","diff",[fx,"x"]);
fa=Assign(fx,["x","a"]);
ga=Assign(gx,["x","a"]);
tf=ga+"*(x-a)+(+fa+)";
println(tf);

```

コンソールには

```
(%e^a-%e^-a)/2*(x-a)+((exp(a)+exp(-a))/2)
```

が表示される。

[⇒ 関数一覧](#)

関数	Mxtex(name, 式)
機能	式を TeX 書式にする
説明	第 2 引数の式は、直接書いた式もしくは Mxfun の戻り値。これを TeX の書式にする。 戻り値は、変数 txname にも代入される。

例：部分分数への分解

部分分数 $\frac{x^3}{(x+1)(x+2)}$ の分解を Maxima で行い、その結果を TeX 書式にして画面に表示する。画面に表示された結果はそのまま KeTCCindy で出力できる。

```
Mxfun("1","partfrac",["x^3/((x+1)*(x+2))","x"]);
Mxtex("1",mx1);
Expr([0,1],"e",tx1);
```

ここで、mx1, tx1 はそれぞれ Mxfun("1", · · ·), Mxtex("1", · · ·) の結果 (戻り値) である。mx1, tx1 はコンソールにも表示され、tx1 は次のようになっている。

```
\frac{8}{x+2}-\frac{1}{x+1}+x-3
```

Cindyscript は TeX 書式をサポートしているのでこれで描画面に分数式が表示されるが、Tex の文書では、`\frac{}{}` ではなく、`\dfrac{}{}` を使うことが多い。そこで、Assign() を用いて、"frac" を "dfrac" に変えれば、そのまま Tex 文書で使える。ただし、Cindyscript は `\dfrac{}{}` をサポートしていないので、画面上では分数表記にならない。そのあたりの事情を次のスクリプトで示す。

```
fx="x^3/((x+1)*(x+2))";
pfx=Mxfun("1","partfrac",[fx,"x"]);
form=Mxtex("1",fx)+"="+Mxtex("2",pfx);
dform=Assign(form,["frac","dfrac"]);
Letter([0,5],"e","部分分数への分解  $" + form + "$");
Letter([0,3],"e","部分分数への分解  $" + dform + "$");
```

Cinderella の描画面では次のように表示される。

部分分数への分解	$\frac{x^3}{(x+1)(x+2)} = \frac{8}{x+2} - \frac{1}{x+1} + x - 3$
部分分数への分解	$x^3(x+1)(x+2) = 8x+2-1x+1+x-3$

出力した TeX 挿入図では次のようになる。

部分分数への分解 $\frac{x^3}{(x+1)(x+2)} = \frac{8}{x+2} - \frac{1}{x+1} + x - 3$

部分分数への分解 $\frac{x^3}{(x+1)(x+2)} = \frac{8}{x+2} - \frac{1}{x+1} + x - 3$

なお、文字列を置換するのに、`Assign(form,["frac","dfrac"])`ではなく、`Cindyscript`の文字列の関数 `replace` を用いて、

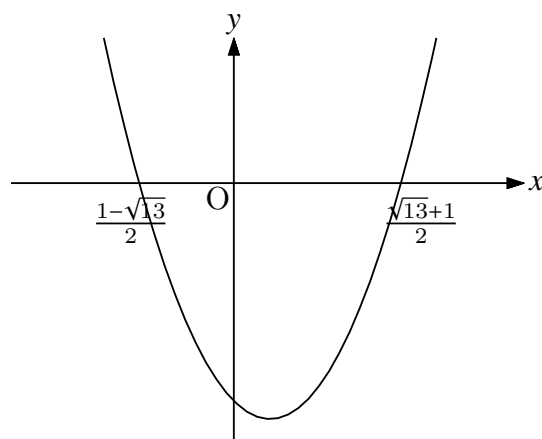
```
dform=replace(form,"frac","dfrac");
```

としてもよい。

例：2次関数のグラフを表示し、 x 軸との交点の x 座標を表示する。

```
fx="x^2-x-3";
cmdL=[
  "ans:solve",[fx,"x"],
  "ans",[]
];
CalcbyM("ans",cmdL);
p1=indexof(ans,"[");
p2=indexof(ans,",");
p3=indexof(ans,"]");
s1=substring(ans,p1,p2-1);
s2=substring(ans,p2,p3-1);
s1=replace(s1,"x =", "");
s2=replace(s2,"x =", "");
Mxtex("1",s1);
Mxtex("2",s2);
Plotdata("1",fx,"x");
```

```
Expr([-2,-0.5],"e",tx1);
Expr([2,-0.5],"e",tx2);
```



ここで, `CalcbyM("ans",cmdL);` で得られる `ans` は, 次のような文字列である。

```
"[x = -(sqrt(13)-1)/2,x = (sqrt(13)+1)/2] "
```

そこで, ここから 2 つの式だけを抽出する作業を行ったのち, `Mxtex()` で TeX の式を得ている。

さらに応用として, 点 A を Cinderella の作図ツールで作図し,

```
if(A.y<0,
  fx="(x-"+text(A.x)+")^2"+guess(A.y),
  fx="(x-"+text(A.x)+")^2"+"+guess(A.y);
);
```

とすると, 点 A を頂点とする放物線と軸との交点の座標が描かれる。Maxima とのデータのやり取りをするためのタイムラグがあるが, インタラクティブに放物線的位置を変えることができる。

<参考>

2 次関数のような簡単な関数であれば, Cindyscript の `roots()` 関数を用いて 2 次方程式が解けるので, 次のスクリプトでほぼ同じ動作をするものを作ることができる。「ほぼ」というのは点 A の位置によっては, `guess()` で解釈しきれないことがあるためである。Maxima を使えば数式処理で解を求めるので, A がどこにあってもきれいに表示できる。

```
fx="x^2-2*A.x*x+A.x^2+A.y";
cf=[A.x^2+A.y,-2*A.x,1];
sol=roots(cf);
s1=guess(sol_2);
```



```
s2=guess(sol_1);  
Mxtex("1",s1);  
Mxtex("2",s2);  
Plotdata("1",fx,"x");  
Expr([-2,-0.5],"e",tx1);  
Expr([2,-0.5],"e",tx2);
```

[⇒ 関数一覧](#)

4.3 Scilab との連携

KeTCindy は、もともと Scilab と連携して、Cinderella からの出力を TeX のテキストとするシステムである。しかし、それだけでなく、Maxima などとの連携と同様、Scilab の関数を実行して、データをやりとりすることができる。

KeTCindy では、kc.bat/sh によってコマンドを Scilab に渡し、結果をテキストファイルで受け取る。このとき、Scilab とのやりとりで、次のようなファイルが作業ディレクトリに作成される。

拡張子 sci : Scilab 用のファイル

拡張子 txt : データファイル

このデータのやり取りに関する次のオプションがある。

オプションなしまたは, "" のとき

i) データファイルがなければ、新しく作る

ii) データファイルが既にあればそれを読み込む

"m" のとき、強制的にデータファイルを作り直す。

"r" のとき、すでにあるデータファイルを読み込む。

このとき、ファイルの読み書きで不具合があると、数秒の後「==> file.txt not generated (5 s)」のようなエラーメッセージがコンソールに表示される。このような場合は作業ディレクトリの設定などを確認していただきたい。この待ち時間については、Wait オプションで設定することもできる。

関数 CalcbyS(name, コマンド, option)

機能 Scilab のコマンドスクリプトを実行する

説明 第2引数は Scilab で実行するコマンドのリスト。

関数 PlotdataS(name, 関数, 変数)

機能 Plotdata と同様の書式で、Scilab の関数を実行する

説明 Cindyscript の組込関数にない関数のグラフを描くことができる。

例：ベッセル関数のグラフを描く

```
PlotdataS("1","besselj(1,x)","x");
```

[⇒ 関数一覧](#)

4.4 Risa/Asir との連携

関数 CalcbyA(name, コマンド, option)

機能 Risa/Asir のスクリプトを実行する

説明 第2引数は Risa/Asir で実行するコマンド。

コマンドと引数リストの繰り返しからなるリスト (例えば cmdL) を作って、一度に実行する。

戻り値はない。(未定義値) 結果は、コマンドリストの最後に記述した変数 (引数は空リスト) の値が name で指定された変数に代入される。複数の結果を戻すときは、:: で区切って記述するとリストにして代入される。

関数 Asirfun(name, 式, リスト, option)

機能 Risa/Asir の関数を実行する

説明 第2引数の「式」は Risa/Asir の関数名。第3引数のリストは関数に渡す引数のリスト。

戻り値は、第1引数の式に1つでも文字があると文字列となる。すべて数字 (+, -, . を含む) の場合は16桁以下であれば数、それ以上の場合は文字列となる。また、戻り値は、変数 asname にも代入される。

オプションに "Disp=no" をつけると、結果をコンソールに表示しない。

[⇒ 関数一覧](#)

4.5 FriCAS(Axiom) との連携

関数 CalcbyF(name, コマンド, option)

機能 FriCAS のスクリプトを実行する

説明 第2引数は FriCAS で実行するコマンド。

コマンドと引数リストの繰り返しからなるリスト (例えば cmdL) を作って、一度に実行する。

戻り値はない。(未定義値) 結果は、コマンドリストの最後に記述した変数 (引数は空リスト) の値が name で指定された変数に代入される。複数の結果を戻すときは、:: で区切って記述するとリストにして代入される。

関数 Frfun(name, 式, リスト, option)

機能 FriCAS の関数を実行する

説明 第2引数の「式」は FriCAS の関数名。第3引数のリストは関数に渡す引数のリスト。

戻り値は、第1引数の式に1つでも文字があると文字列となる。すべて数字 (+, -, . を含む) の場合は16桁以下であれば数、それ以上の場合は文字列となる。また、戻り値は、変数 friname にも代入される。

オプションに "Disp=no" をつけると、結果をコンソールに表示しない。

[⇒ 関数一覧](#)

4.6 MeshLab との連携

MeshLab は、3D データ (obj データなど) を読み込んでレイトレーシングで表示・編集するソフトウェアである。レイトレーシングで 3D グラフィクスを描くには、Cinderella と親和性の高い Cindy3D を利用するのがよいが、MeshLab を使うメリットは 3D プリント用の STL ファイルを出力できることである。また、KeTCindy で描いた 3D の図がレイトレーシングでどのようなになるのかを見ることも比較的簡単にできる。

MeshLab との連携は、KeTCindy から Obj 形式のデータを書き出すことで行う。Mkobj**() 関数でデータを作り、Mkviewobj() 関数で MeshLab を呼び出して表示を行う。

なお、Mkviewobj() 関数で MeshLab を呼び出して表示を行う場合、これを Draw スロットに書くと頻繁に呼び出しが行われるため非効率となる。そこで、if(1==0,・・・) で・・・の部分に MeshLab の呼び出し関係のスクリプトを書いて、実際に呼び出すときに if(1==1,・・・) とする方法と、呼び出し関係のスクリプトを関数化してボタンに割り当てる方法がある。ketcindy パッケージに含まれる sample にボタンをつけたものがある。

なお、3D であるので、Initialization スロットに

Ketinit(); Ketinit3d(); を記述しておく。

関数 Mkobjcmd(name, 式, option)

機能 厚みを持たない曲面の obj ファイルのためのコマンドを作成する

説明 オプションは [分割数 1, 分割数 2, 表側の方向の指定]

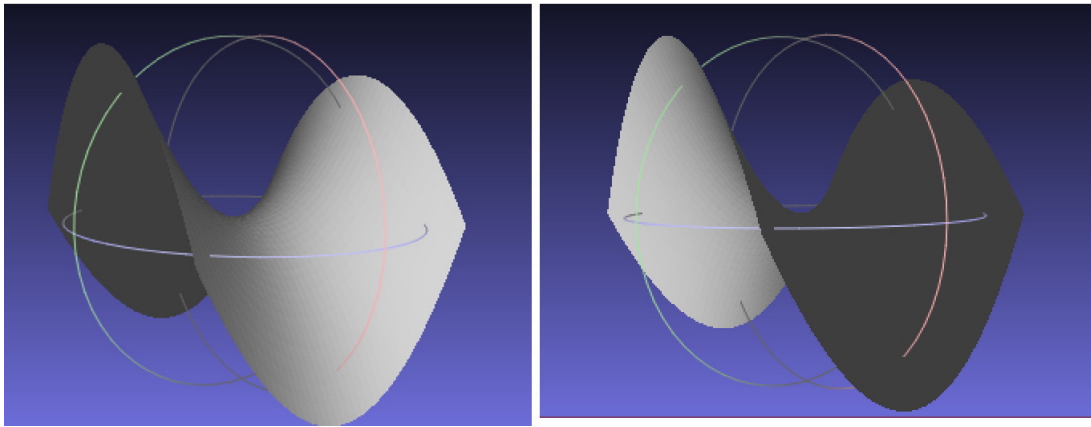
表側の方向は、変数に対して、右手系の方向が”+”

作成されるデータは”oc”+name のファイル名の obj データである。この名称は、Mkviewobj() で用いる。(以下、Mkobj**() 関数では同様)

【例】：サドル面

```
fd=[ "z=x^2-y^2", "x=[-1,1]", "y=[-1,1]", " "];
Sf3data("1",fd);
Windispg();
Mkobjcmd("1",fd,[40,40,"-"]);
Meshlab():=(
    Mkviewobj("saddle",oc1, ["m","v"]);
);
```

このうち、Sf3data("1",fd); は Cinderella の画面に表示するためであって、なくてもよい。次図で、左が option + の場合、右が - の場合である。



⇒ 関数一覧

関数 Mkobjcrvcmd(name, PD, option)

機能 空間曲線（直線）の obj ファイルのためのコマンドを作成

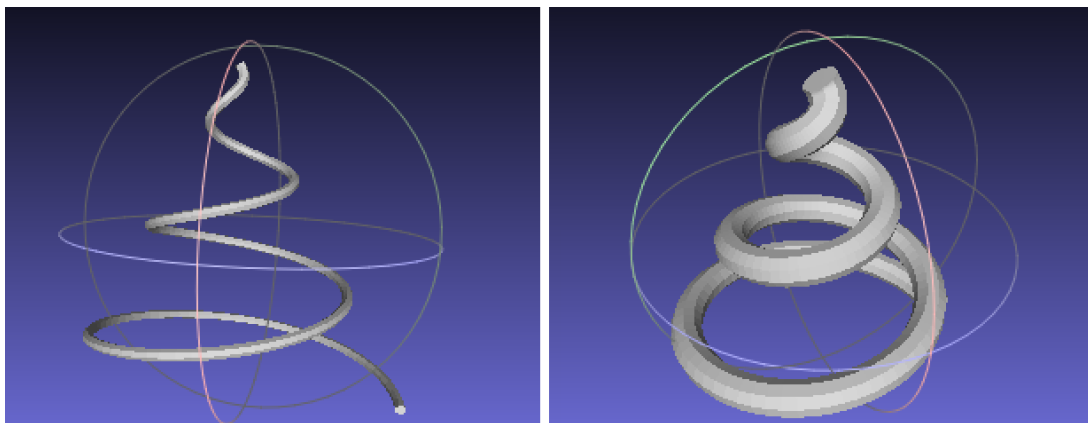
説明 オプションは [太さ, 断面の形状（正多角形）の辺の数, 断面の正面]

曲線は紐のようなもので表す。その断面は正多角形で、デフォルトは正 6 角形である。断面の正面は"xy","yz","zx"のいずれかで指定する。太くなった時に形状の差が現れる。

例 太さ 0.03 で螺旋を描く

```
Spacecurve("1", "[(6*pi-t)/(6*pi)*cos(t), (6*pi-t)/(6*pi)*sin(t), 0.1*t]",
    "t=[0,6*pi]", ["Num=200"]);
Windisp();
Mkobjcrvcmd("1", "sc3d1", [0.03]);
Meshlab() := (
Mkviewobj("spiral", oc1, ["m", "v"]);
);
```

Mkobjcrvcmd("1", "sc3d1", [0.1, 8, "yz"]); としたのが下図右。



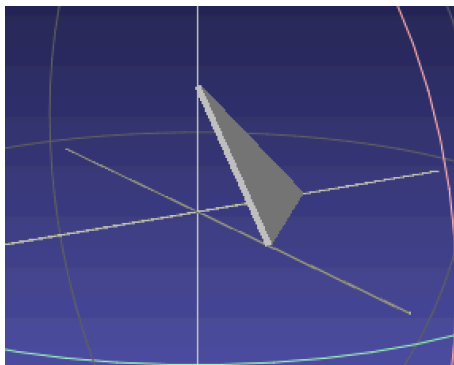
関数 `Mkobjnrm(name, 式)`
機能 法線ベクトルのデータを作成
説明 式は曲面を表す式。これに対し、法線ベクトルを表す式を求める。

関数 `Mkobjplatecmd(name, 面データ, options)`
機能 面を描く
説明 面データを渡して面を描く。

`options` は、面の厚みの指定。厚みは中心線に対し、両側につけることができる。
たとえば、`[0.05]` はプラス側に `0.05` の厚み、`[0.05,-0.04]` はマイナス側にも `0.04` の厚みをつける。

【例】 三角形のプレートを描く

```
Xyzax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,5]");  
p1=[2,0,0];  
p2=[0,2,0];  
p3=[0,0,2];  
plane=[[p1,p2,p3],[[1,2,3]]];  
Mkobjplatecmd("1",plane,[0.05]);  
Mkobjcrvcmd("2","ax3d");  
Mkviewobj("plane",Concatcmd([oc1,oc2]),["m","v"]);
```



関数 `Mkobjpolycmd(name, PD, options)`
機能 多面体を描く
説明 `VertexEdgeFace()` の戻り値を `PD` として渡して多面体を描く。

関数 Mkobjsymbcmd(PD, 実数, 実数, ベクトル, ベクトル)

機能 文字等の obj データのためのコマンドを作成

説明 引数の PD を描く。第 2 引数は大きさ, 第 3 引数は回転角, 第 4 引数は正面方向のベクトル, 第 5 引数は PD の中心の位置。

PD は, 平面の描画コマンドによるプロットデータが使える。また, PD に半角アルファベットを文字として与えることができる。この場合, 文字は n,p,q,r,t,x,y,z で, 該当するフォントが data フォルダの fontF フォルダに用意されている。この中にあるフォントは使えない。

【例】軸の名前 x,y,z を表示し, 円を描く。

円は Circledata(); でプロットデータを作成する。

関数 Mkobjthickcmd(name, 式)

機能 厚みを持つ曲面の obj ファイルのためのコマンドを作成

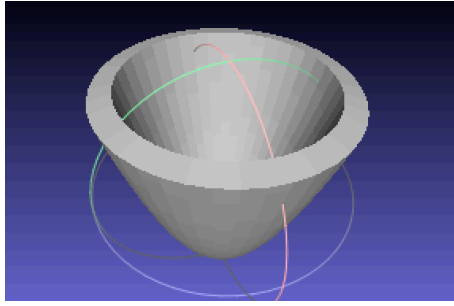
説明 オプションは [分割数 1, 分割数 2, 厚み, 表側の方向の指定, 条件] 表側の方向は, 変数に対して, 右手系の方向が”+”。厚みを持つため, nsew のそれぞれについて, ”+n+s-e-w” のように指定する。

条件として, "Assume(R>0)" をつけると, R が 0 以下になるための不具合を回避できる。

また, "ratsimp" をつけると有理関数について, "trigsimp" をつけると三角関数について, 処理を速くすることができる。なお, この関数は Maxima を使うので, Maxima をインストールしていることが前提。

【例】回転放物線

```
fd=[
  "z=(x^2+y^2)",
  "x=R*cos(T)", "y=R*sin(T)",
  "R=[0,2]", "T=[0,2*pi]", "e"
];
Mkobjthickcmd("1",fd,[40,40,0.2,"+n+s-e-w+", "assume(R>0)"]);
Mkviewobj("pala",oc1,["m","v","Wait=5"]);
```

関数 Mkviewobj(name, PD, options)

機能 obj ファイルを作成。option により MeshLab を立ち上げて表示する。

説明 第2引数に複数のプロットデータを与えるときは、Concatcmd() により1つにまとめる。オプションは

"m" または "make"	データを作る（指定しない場合もデータがなければ作る）
"v" または "view"	MeshLab を立ち上げて表示する
"W=n"	作成するための待ち時間。n 秒。これを過ぎると終了する
"Unit=mm"	Setunitlen() と連動して3Dプリンタの数値の単位を mm で指定する 3Dプリンターがインチで認識する場合は "Unit=in" とする。

[⇒ 関数一覧](#)

4.7 表計算ソフトとの連携

表計算ソフトでは、複数のセルを選択してコピー（Windows では Ctrl+ C , Mac では Command+C）すると、セルの内容は tab 区切りのテキストデータとしてクリップボードにコピーされる。これを Cindyscript エディタにペーストすることで表計算ソフトのデータを KeTCindy で利用できる。逆に、Cindyscript のコンソールへの出力を表計算ソフトのシートにコピーすることもできる。

また、表計算ソフトから書き出した CSV ファイルについても同様にして CSV 形式のデータを扱うことができる。

関数 Tab2list(str, option)

機能 str の内容をリストに変換する

説明 tab やコンマ区切りになっている文字列 str をリストに変換する。
option は、次の通り。

Blank=a : NULL のセルを a に置き換える。

Sep=b : セパレータ（区切り文字）を b とする。デフォルトは tab コード

次のような手順で表計算ソフトや CSV ファイルからデータを KeTCindy に移すことができる。

(1) Cindyscript エディタで、適当な文字変数を用意する。

たとえば, data="";

```
1 Fhead="template"; // write a head name
2 Texparent="";
3 Ketinit();
4
5 data="";
6
7 Windispg();
8
```

(2) 表計算ソフトで、適当な範囲を指定しクリップボードにコピーする。

Windows なら Ctrl+C, Mac なら Command+C

	A	B	C	D	E	F
1		A	T	G	C	
2	トリ結核菌	15.5	14.3	36.4	33.8	
3	大腸菌	24.7	23.6	26	25.7	
4	コムギ	27.4	27.1	22.7	22.8	
5	サケ	29.7	29.1	20.8	20.4	
6	ヒト	30.9	29.4	19.9	19.8	
7						
8						

(3) data=""; のダブルクォートの間にペーストする。

最後の行は右図のように, " の前で改行されていてもよい。

```
5 data=" A T G C
6 トリ結核菌 15.5 14.3 36.4 33.8
7 大腸菌 24.7 23.6 26 25.7
8 コムギ 27.4 27.1 22.7 22.8
9 サケ 29.7 29.1 20.8 20.4
10 ヒト 30.9 29.4 19.9 19.8";
```

```
5 data=" A T G C
6 トリ結核菌 15.5 14.3 36.4 33.8
7 大腸菌 24.7 23.6 26 25.7
8 コムギ 27.4 27.1 22.7 22.8
9 サケ 29.7 29.1 20.8 20.4
10 ヒト 30.9 29.4 19.9 19.8
11 ";
```

(4) この文字変数 data に対し, Tab2list(data) を実行すると, 行列を表すリストが返される。

これを適当な変数に代入し, 作表コマンドで表にするなど, 目的に応じて利用する。
数値だけなら行列として計算もできる。

```
5 data=" A T G C
6 トリ結核菌 15.5 14.3 36.4 33.8
7 大腸菌 24.7 23.6 26 25.7
8 コムギ 27.4 27.1 22.7 22.8
9 サケ 29.7 29.1 20.8 20.4
10 ヒト 30.9 29.4 19.9 19.8";
11 dlist=Tab2list(data);

[[0,A,T,G,C],[トリ結核菌,15.5,14.3,36.4,33.8],[大腸菌,24.7,23.6,26,25.7],
[コムギ,27.4,27.1,22.7,22.8],[サケ,29.7,29.1,20.8,20.4],[ヒト,
30.9,29.4,19.9,19]]
```

空文字のセル (NULL) が含まれる場合, デフォルトではそのまま空文字になるが, アンケート処理などで無回答を 0 にしたいような場合は

```
dlist=Tab2list(data, ["Blank=0"]);
```

とする。

CSV ファイルから CSV 形式 (コンマ区切り) のデータをコピーした場合は

```
dlist=Tab2list(data,["Sep="]);
```

とする。

なお、文字列をセパレータで区切ってリスト化する Cindyscript の関数に `tokenize()` がある。上の例で、

```
dlist=tokenize(data,[unicode("000a"),unicode("0009")]);
```

とすると、改行コード (000a) と tab コード (0009) で切り分けてリスト化する。このとき、リストの各要素はつぎのようになる。

文字列→文字列

数値形式の文字→実数 例： 14 → 整数 14 12.3 → 実数 12.3

計算式の形 → 文字列 例： 437-0023 → 437-0023 (文字列)

これに対し、`Tab2list()` では、計算式の形の文字列は数値と見なして計算結果を取得する。

例： 437-0023 → 414 (数値)

したがって、郵便番号や日付 (28/12/5) のようなものは計算されてしまうので、`tokenize()` を用いるのがよい。なお、`tokenize()` の場合、空行は空リストになるので、最後の行でダブルクォートの前で改行されていると空リストが入るので注意されたい。

関数 `Dispmat(list)`

機能 リストを行列の形で tab 区切りにしてコンソールに表示する。

説明 行列を表すリスト（たとえば `dlist`）を引数として `Dispmat(dlist)` を実行すると、コンソールに行列型で内容が表示される。

実際には TAB 区切りの文字列。(println としなくても直接コンソールに表示される)

これを表計算ソフトのシートにコピーする。

11	<code>dlist=Tab2list(data);</code>				
12	<code>Dispmat(dlist);</code>				
0	A	T	G	C	
トリ結核菌		15.5	14.3	36.4	33.8
大腸菌	24.7	23.6	26	25.7	
コムギ	27.4	27.1	22.7	22.8	
サケ	29.7	29.1	20.8	20.4	
ヒト	30.9	29.4	19.9	19	

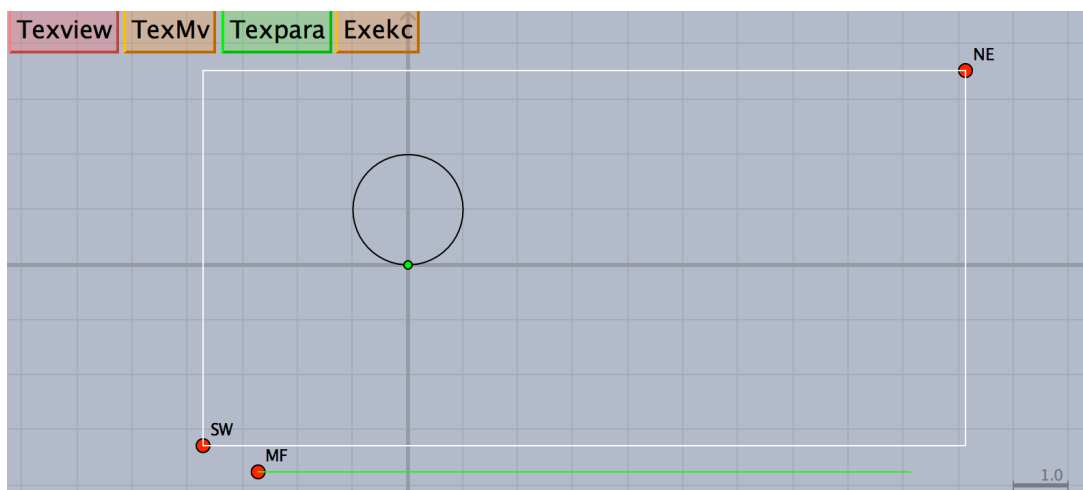
5 アニメーション PDF : KeTCindymv

5.1 概要

アニメーションのできる PDF を作る。

Cinderella の作図機能と Cindyscript を用いてアニメーションができるが、PDF にすることで Cinderella がなくても PDF ビュアーがあればアニメーションを実行できるので、プレゼンテーションや教材の受け渡しなどに便利である。

KeTCindymv の画面は次のようになっている。(templatemv.cdy)



画面上方のボタンには、次のようなスクリプトが割り当てられているので、ボタンを自作することもできる。

Texview	: Viewtex();	現在の画面の PDF データを作る
TexMv	: Texmovie();	アニメーション PDF データを作る
Texpara	: Texpara();	フレームに分割した PDF データを作る
Exekc	: kc();	バッチファイルを実行する

アニメーションの作成は、フレームを定義する関数を作成し、Moviedata() 関数を実行する。実行後、TexMv、Exekc のボタンを順に押すことで、Fhead+”moviefigs.tex” と Fhead+”moviemain.tex”, Fhead+”moviemain.pdf” が生成されて表示される。たとえば、Fhead が ”abc” の場合、TeX の文書には

```
\input{abcmoviefigs.tex}
```

で動画を挿入することができる。

ただし、動画のできる PDF を作成するには、ドキュメントクラスと使用パッケージについて注意が必要である。

ドキュメントクラスは、article または jarticle とする。jsarticle は使えない。

パッケージは animate に dvipdfmx オプションをつける。

```
\usepackage[dvipdfmx]{animate}
```

また、アニメーション PDF でアニメーションを行うには Adobe Acrobat Reader など、アニメーションに対応した PDF リーダーが必要である。Windows の SumatraPDF, Mac の プレビューでは動かない。

次に, Texpara, Exekc のボタンを順に押すことで, フレームに分割した Fhead+"parafigs.tex" と Fhead+"paramain.tex" ,Fhead+"paramain.pdf" が生成されて表示される。この図を利用するにはパッケージの指定がやや面倒である。 Fhead+"mvparamain.tex" を参照されたい。

画面下方のスライダによりアニメーションの途中図を描くことができる。Texview と Exekc ボタンによりその図のファイルが作成される。

5.2 設定

KeTCindymv では, KeTCindy と若干異なる設定が必要である。

まず, Initialization スロットでは, KeTCindy で

```
Shellfile="";
```

になっているのを

```
Shellfile="mv";
```

とする。

次に, Draw スロットでは, 次のように Ketinit(); に加え, Ketinitmv(); が必要である。また, 画面表示のために, Windisp(); のかわりに Mvdisp(); を用いる。

```
Fhead="templatemv";
```

```
Texparent="";
```

```
Ketinit();
```

```
Ketinitmv();
```

ここにスクリプトを書く

```
Mvdisp();
```

なお, ひながたの templatemv.cdy ではこの設定がすでにしてある。

5.3 描画

関数 Moviedata(str1,str2,options)

機能 アニメーションデータを作る

説明 str1 はアニメーションを定義する関数名を文字列とする。str2 は定義域。

options は, Cut と Div

Cut : 1 秒のフレーム数。初期値は 20。

Div : 全体のフレーム数。初期値は 80。

したがって, 初期値では 4 秒間のアニメーションとなる。

【例】 定円上を動く点 P と, 定点 A を結ぶ線分の中点を Q として動きを見る。

まず, 定円を, 原点中心、半径 2 として描いておく。

アニメーションを定義する関数は, 時間を t とすれば, 時刻 t における図 (動くものだけ) を定義する。時刻は単なる媒介変数であるので, t でなく s などでもよい。

```
Mf(t):=(  
  pt=2*[cos(t),sin(t)];  
  Listplot("2",[[4,0],pt]);  
  Pointdata("2",mid(pt,[4,0]));  
  Letter([[4,0],"s","A",pt,"en","P",mid(pt,[4,0]),"ne","Q"]);  
);
```

ここで, $Mf(t)$ の中で使っているユーザー定義関数 $mid()$ は, 端点を引数として線分の中点を返すもので,

```
mid(p1,p2):=(p1+p2)/2;
```

として定義しておく。

また, 点の大きさを適宜設定しておく。

以上の準備の後,

```
Moviedata("Mf(t)","t=[0,2*pi]");
```

を実行する。

ここでは, 角度を媒介変数としているので, 時間の t でなく s として

```
Moviedata("Mf(s)","s=[0,2*pi]");
```

としてもよい。

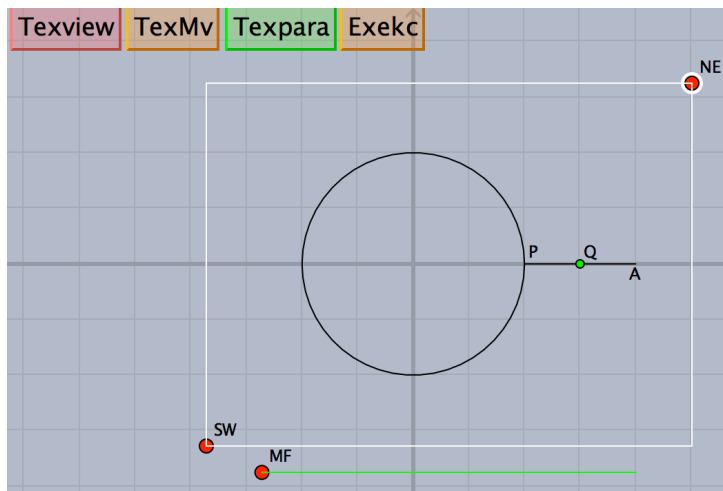
次のようにオプションを指定すると, 5 秒間のアニメーションとなる。

```
Moviedata("Mf(s)","s=[0,2*pi]","Div=50","Cut=10");
```

["Div=150","Cut=30"] とすると, やはり 5 秒間のアニメーションとなるが, 1 秒間のフレーム数が多いためなめらかな動きとなる。ビデオのフレームレートである。た

だし、ファイルサイズは約3倍となる。

Cinderella の画面は次のようになる。



TexMv ボタン または TexPara ボタンを押すと画面上でもアニメーションが実行される。その後 Exekc ボタンを押すとファイルが作成される。

この図では、点が明示されるのは Q だけである。A,P とも点を明示する場合のスク립トの全体を示しておく。

```
Fhead="mid";
Texparent="";
Ketinit();
Ketinitmv();
mid(p1,p2):=(p1+p2)/2;
Circledata("1",[[0,0],[2,0]]);
Ptsize(4);
Pointdata("1",[4,0]);
Mf(t):=(
  pt=2*[cos(t),sin(t)];
  Listplot("2",[[4,0],pt]);
  Pointdata("2",pt);
  Pointdata("3",mid(pt,[4,0]));
  Letter([[4,0],"s","A",pt,"en","P",mid(pt,[4,0]),"ne","Q"]);
);
Moviedata("Mf(s)","s=[0,2*pi]","Div=30","Cut=10");
Mvdispq();
```

⇒ [関数一覧](#)

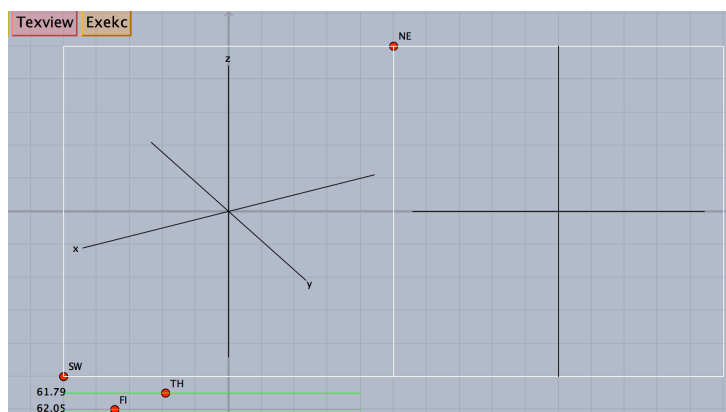
関数	Setpara(str)
機能	parapa のタイトルを指定する
説明	パラパラ動画のタイトルを str にする。デフォルトは Fhead。 "" にした場合はタイトルを表示しない。

6 K_ETCindy3D

6.1 概要

K_ETCindy3D の画面は次のように構成される。

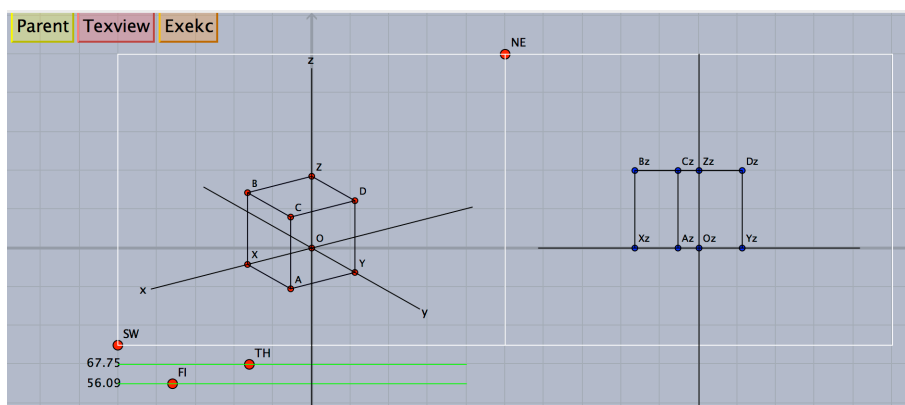
Cinderella の描画面に、白の矩形で囲んだ領域が2つできる。NE,SW を対角とする左側の領域を主画面、右側の領域を副画面という。



主画面は平面の場合と同様、TeX に出力される範囲を示し、NE,SW の2点をドラッグすることにより変更できる。

副画面は、Start3d() 関数により作成される。座標軸は、Xyzax3data() 関数によって描くことができる。主画面の下方のスライダーで視点が移動でき、主画面上では軸が回転する。副画面は、xy 平面上に視点を置いたものと考えればよい。

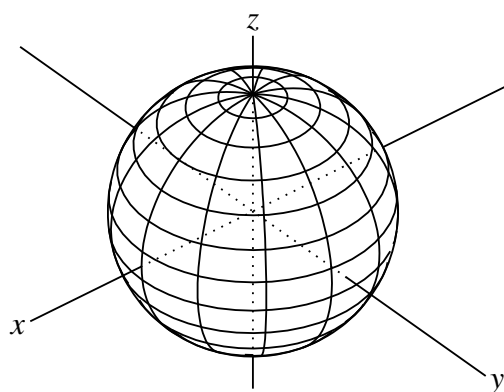
主画面上に Cinderella の作図ツールで点や線分を作図すると、Start3d() 関数により副画面に対応する点が作図される。主画面上の点をドラッグすると x,y 座標を変更でき、副画面上の点をドラッグすると z 座標を変更できる。空間内の点は、Putpoint3d() 関数によって座標を指定して作成することもできる。次のような図は Cinderella の作図ツールだけでも作ることができるし、Putpoint3d() 関数を用いても作ることができる。



主画面と副画面の対応する点をドラッグすることにより、インタラクティブに図形を描くことができるが、実際には描画関数を用いて図形を描き、スライダで視点を変えて見やすい図を TeX に出力することになるだろう。

KeTCindy3D では、点・直線・曲線・面の描画を描画関数を用いて行うことができる。線や面については陰線処理を行い、立体的な図を作成することができる。しかし、陰線処理は処理にかなりの時間がかかるので、Cindyscript ではなく Scilab で行っているが、それでも相当時間がかかることを覚悟しなければならない。Scilab での計算は、バッチファイル/シェルスクリプトで行っているが、Cindyscript ではその計算結果が出るのを待つため一時的に反応がなくなる。したがって、反応がなくなってもハングアップではないので、Cinderella を強制終了しないように。

一例を示すと、球面をメッシュ入りで座標軸とともに陰線処理して描いた場合、次のような図が画面に表示されるまでの時間は、MacBookPro13' (Late 2013) Core i7 2.8GHz 8GB (OSX 10.11.6) の場合で約 124 秒であった。スクリプトは [Wireparadata\(\)](#) の例を参照されたい。



6.2 設定・定義

関数 Ketinit3d()

機能 KeTCindy3D の使用宣言

説明 Cinderella の画面を 3D モードにする。

Cinderella の描画面に、視点移動のための 2 つのスライダを作る。スライダは初期位置が左端になる。


KeTCindy の使用宣言 Ketinit() とともに用いるが、この関数は一度だけ実行すればよいので、通常は Initialization スロットに置く。Ketinit() も、平面の場合と異なり Initialization スロットに置けばよい。ただし、平面の関数を多用する場合は、Ketinit() は Draw スロットに書いておく。場合によって変数の初期化などが必要なのである。KeTCindy3D における変数の初期化などは、Start3d() で行われる。

関数 Start3d(option)

機能 3D の画面設定と空間点の認識

説明 副画面を作り、幾何点を 3D の点として認識する。この関数は必須で、Draw スロットのはじめの方に置く。

Cinderella の作図ツールで、点・線分を作図すると、内部関数の Ptseg3data() によってそれらを空間の点として認識し、副画面上に対応する点をとる。ただし、始めは z 座標を 0 とする。点の名前が A であれば、副画面上の点は Az となる。点をポイントして選択すると副画面の上に座標が表示される。

作図した点の名称をインスペクタで変更した場合、新しい名称に対応する点を副画面上に作成するが、以前の点は消えないので要注意。たとえば、点 A を作図した後、主画面上の点 A をインスペクタで点 D に変えた場合、副画面上に新たに Dz ができるが、以前の Az も残る。残った Az は、選択しておいて作図ツールの消去ボタン  で消すことができる。

option に、除外点のリストを与えると、その点は空間点としない。(始点を移動しても位置は変わらない)

[⇒ 関数一覧](#)

6.3 描画

関数 Bezier3d(name, リスト 1, リスト 2)
機能 空間ベジェ曲線を描く
説明 引数はリスト 1 が端点リスト, リスト 2 が制御点リスト
1 組の端点につき, 2 つの制御点を使う。

【例】いくつかの点をベジェ曲線で結ぶ

端点 A,B に対し, 制御点を D,E とする。

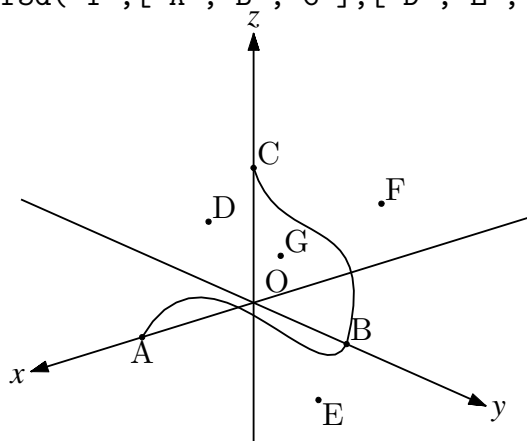
```
Bezier3d("1",["A","B"],["D","E"]);
```

端点 A,B に対し, 制御点を D,E とし, 端点 BC に対し制御点を E,F とする。

```
Bezier3d("1",["A","B","C"],["D","E","E","F"]);
```

端点 A,B に対し, 制御点を D,E とし, 端点 BC に対し制御点を F,G とする。(図)

```
Bezier3d("1",["A","B","C"],["D","E","F","G"]);
```



[⇒ 関数一覧](#)

関数 Changstyle3d(リスト, リスト)
機能 3D プロットデータの属性を変更
説明 第 1 引数のプロットデータの属性を, 第 2 引数に変更する。

たとえば, 補助線など, 画面には描いても TeX に書き出さない線を描画するときは, option に ["notex"] をつけるが, これをあとから付加したい場合に利用する。プロットデータはリストにできるので, 複数のプロットデータの属性をまとめて変更することができて便利である。

【例】4 つの点で四面体の辺を描き, まとめて notex にする。点 A,B,C,D はとってあるものとする。

```
Spaceline("1",[A,B]);
```

```

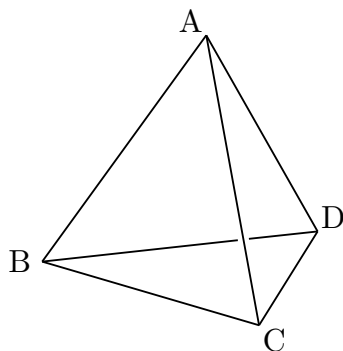
Spaceline("2",[A,C]);
Spaceline("3",[B,C]);
Spaceline("4",[A,D]);
Spaceline("5",[B,D]);
Spaceline("6",[C,D]);
edges=apply(1..6,"sl3d"+text(#));
Changestyle3d(edges,["notex"]);

```

関数 Concatobj(リスト,option)
機能 いくつかの obj データを結合する
説明 多面体の各面の頂点リストから面データ（頂点リストと面リスト）を作る。

【例】 4 点 A,B,C,D を頂点とする四面体を描く。

四面体は 4 つの面からなっている。頂点を A,B,C,D とすると、4 つの面は
 $\triangle ABC$, $\triangle ABD$, $\triangle ACD$, $\triangle BCD$
 である。



そこで

```
Concatobj([[A,B,C],[A,B,D],[A,C,D],[B,C,D]]);
```

とすると、面データ [[A,B,C,D],[1,2,3],[1,2,4],[1,3,4],[2,3,4]] が返される。

この面データを使って四面体を描くことができる。コード例は、[VertexEdgeFace\(\)](#) を参照のこと。

⇒ [関数一覧](#)

関数 Crvsfparadata(name,PD1,PD2, 式,options1,options2)
機能 曲線の曲面による陰線処理
説明 曲線 PD1 を表示するにあたり、曲面 PD2 によって隠れる部分を非表示にする。
 通常は曲面 PD2 も表示するので、Sfbdparadata() も同時に用いることになる。曲面を表示しなければ、曲線だけが陰線処理された状態で表示される。
 第 4 引数の式は、PD2 を描くための式。

options1 には"r","m","Wait=n" が指定できる。Wait の初期値は 20

"r","m"に関しては、オプションなしまたは,"" のとき

- i) データファイルがなければ、新しく作る
- ii) データファイルが既にあればそれを読み込む

"m" のとき、強制的にデータファイルを作り直す。

"r" のとき、すでにあるデータファイルを読み込む。

options2 には 軸の陰線の表示について "nodisp" または線種が指定できる。デフォルトは "nodisp"。

options2 だけを指定したい場合は、options1 を空リスト [] にする。

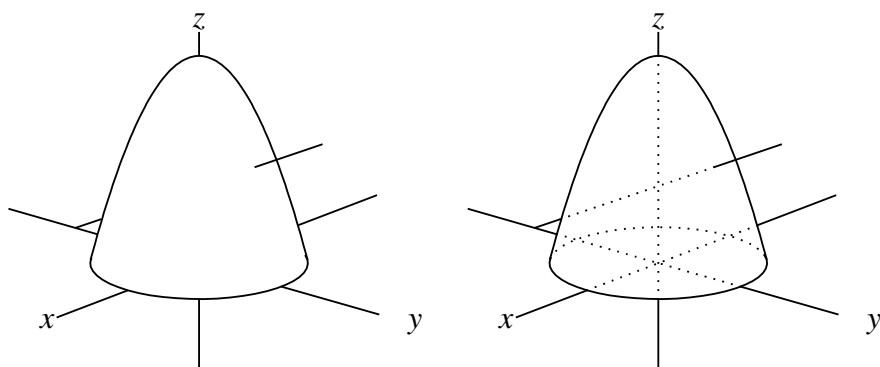
【例】回転放物面と座標軸，線分を描く。

デフォルトでは陰線は非表示である。(下図左)

```
Xyzax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,5]");  
Putpoint3d(["A", [0,-3,0], "B", [0,3,3]], "fix");  
Spaceline([A,B]);  
fd=["z=4-(x^2+y^2)", "x=R*cos(T)", "y=R*sin(T)", "R=[0,2]", "T=[0,2*pi]", "e"];  
Sfbdparadata("1", fd);  
Crvsfparadata("1", "AB3d", "sfbd3d1", fd);  
Crvsfparadata("2", "ax3d", "sfbd3d1", fd);
```

options2 に線種指定 ["do"] をつけると、陰線は点線で表示される。(下図右)

```
Sfbdparadata("1", fd, [], ["do"]);  
Crvsfparadata("1", "AB3d", "sfbd3d1", fd, [], ["do"]);  
Crvsfparadata("2", "ax3d", "sfbd3d1", fd, [], ["do"]);
```



⇒ [関数一覧](#)

関数 Datalist2d()

機能 画面上のプロットデータのリストを取得する

説明 画面に描かれているすべてのプロットデータのリストを返す。

空間図形は、Cinderella の画面上に射影し表示する。そのため、KeTCindy3D は、空間におけるプロットデータと、画面上に表示するプロットデータの 2 つを作っている。Datalist2d() では、画面上に表示するプロットデータのリストを返す。

【例】

```
Xyzax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,5]");  
Putpoint3d(["A", [0, -3, 0], "B", [0, 3, 3]], "fix");  
Spaceline([A, B]);  
println("PD="+Datalist2d());
```

とすると、コンソールに PD=[ax2d, AB2d] と表示される。ax2d は座標軸のプロットデータ ax3d に、AB2d は線分 AB のプロットデータ AB3d に対応している。

関数 Datalist3d()

機能 空間のプロットデータのリストを取得する

説明 空間に描かれているすべてのプロットデータのリストを返す

【例】

```
Xyzax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,5]");  
Putpoint3d(["A", [0, -3, 0], "B", [0, 3, 3]], "fix");  
Spaceline([A, B]);  
println("PD="+Datalist3d());
```

とすると、コンソールに PD=[ax3d, AB3d] と表示される。

[⇒ 関数一覧](#)

関数 Dist3d(a1, a2)

機能 空間の 2 点間の距離を返す

説明 引数 a1, a2 は作図点の名称、空間点の名称のいずれでもよい。
次の 3 通りの記法は同じ結果を返す。混在も可

```
Dist3d("A", "B");
```



```
Dist3d(A,B);
Dist3d(A3d,B3d);
```

関数 Drawpoint3d(座標)

機能 空間点を描く

説明 引数で与えた空間座標の点を描く。この点は幾何点ではない。また、TeX にも出力されない。幾何点にするには [Putpoint3d\(\)](#) と用いる。TeX に点を出力するには、[Pointdata\(\)](#) または [Drawpoint\(\)](#) を用いる。
引数は、座標のリストにすることもできる。

【例】

```
Drawpoint3d([1,1,1]);
Drawpoint3d([1,1,1],[0,1,0]);
```

関数 Embed(name,PD リスト, 式, 変数リスト)

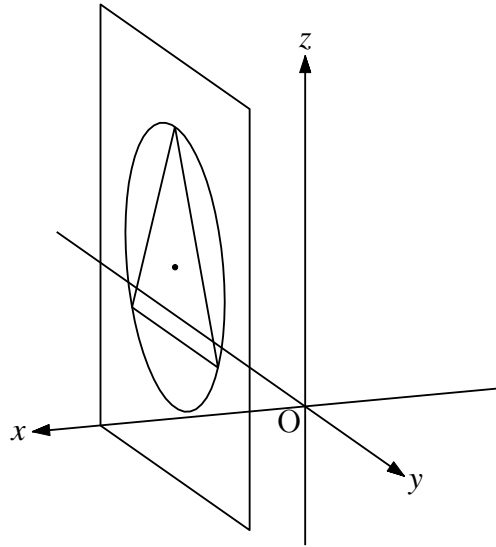
機能 2 D 図形の空間内平面へ埋め込む

説明 第 2 引数は 2 D の図形のプロットデータのリスト、式と変数は平面を記述する式と変数。平面は原点 vo と 2 つの基本ベクトル \vec{vx}, \vec{vy} を用いて、 $vo + x \cdot \vec{vx} + y \cdot \vec{vy}$ の形で表すことができる。変数（基本ベクトルの係数）は x, y でなく、 s, t でもよい。式、変数リストともに文字列にする。また、基本ベクトルは直交していなくてもよいし、長さが異なってもよいが、縦横同じスケールの直交座標系にするのがわかりやすいだろう。

【例】 正三角形と外接円を空間内の平面に埋め込む

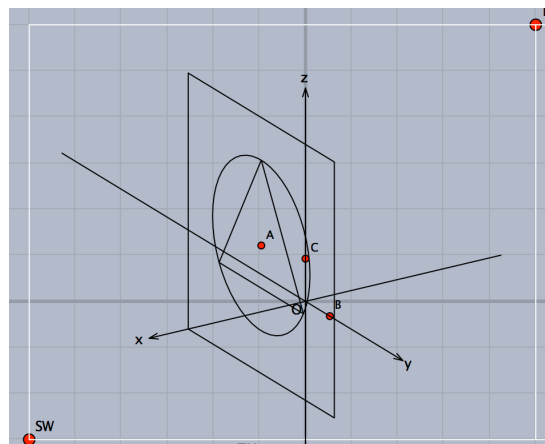
```
Xyzax3data("", "x=[-5,4]", "y=[-10,4]", "z=[-5,5]", ["a", "0"]);
Spaceline("1", [[3,0,0],[3,6,0],[3,6,6],[3,0,6],[3,0,0]]);
Defvar("vo=[3,3,3]");
Defvar("vx=[0,1,0]");
Defvar("vy=[0,0,1]");
Putpoint3d(["A", [3,3,3]], "fix");
Circledata("1", [[0,0],[2,0]], ["nodisp"]);
Listplot("1", [[0,2],[sqrt(3),-1],[sqrt(3),-1],[0,2]], ["nodisp"]);
Embed("1", ["cr1", "sg1"], "vo+x*vx+y*vy", "[x,y]");
```

```
Ptsize(3);
Drawpoint(A);
```



ここで、Embed() で引き渡す vo,vx,vy については、Scilab での変数定義が必要なので (KeTCindy では行わない) Defvar() によって定義をしている。原点，基本ベクトルを，点を作図して次のようにすることもできる。この場合は Defvar() は不要。

```
Putpoint3d(["A",[3,3,3],"B",[0,1,0],"C",[0,0,1]],"fix");
Embed("1",["cr1","sg1"],"A3d+x*B3d+y*C3d","[x,y]");
```



この場合，点 B,C の座標がそのまま基本ベクトルとなっているが，原点 A に対して描画平面上には B,C がないので図がわかりにくい。図をわかりやすくするならば次のようにする。

```
Putpoint3d(["A",[3,3,3],"B",[3,4,3],"C",[3,3,4]],"fix");
Embed("1",["cr1","sg1"],"A3d+x*(B3d-A3d)+y*(C3d-A3d)","[x,y]");
```

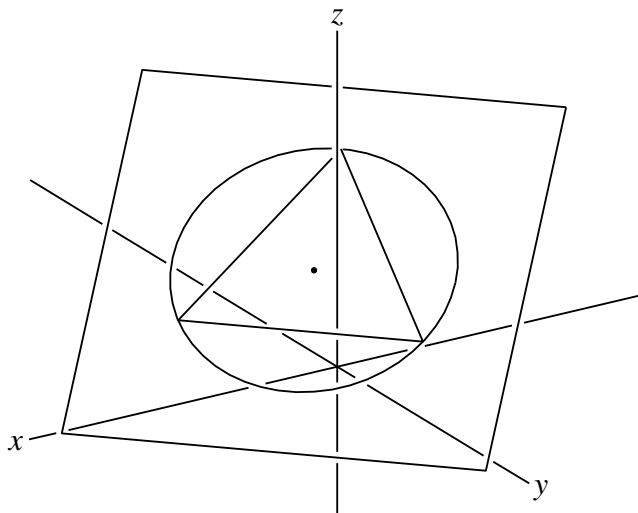
また，平面を記述するのに，平面の原点と法線ベクトルを用いて Perpplane() を用

いと、基本ベクトルが生成されるので、これを利用することができる。次のスクリプトでは、Skeletonparadata() を用いて陰線処理もしている。

```

Xyzax3data("", "x=[-5,5]", "y=[-8,5]", "z=[-5,5]");
Putpoint3d(["O", [0,0,0], "P", [1,1,2]], "fix");
Perpplane("E-F", "P", P3d-O3d, "put");
vec1=3*(E3d-P3d);
vec2=3*(F3d-P3d);
Putpoint3d(["A", P3d+vec1+vec2], "fix");
Putpoint3d(["B", P3d+vec1-vec2], "fix");
Putpoint3d(["C", P3d-vec1-vec2], "fix");
Putpoint3d(["D", P3d-vec1+vec2], "fix");
Spaceline([A,B,C,D,A]);
Circledata("1", [[0,0], [2,0]], ["nodisp"]);
Listplot("1", [[0,2], [-sqrt(3), -1], [sqrt(3), -1], [0,2]], ["nodisp"]);
Embed("1", ["cr1", "sg1"], "P3d+x*(E3d-P3d)+y*(F3d-P3d)", "[x,y]");
Ptsize(3);
Drawpoint(P);
Skeletonparadata("1");

```



[⇒ 関数一覧](#)

関数	Intersectcrvsf(name,PD, 式)
機能	曲線と曲面の交点の座標を求める
説明	PD は曲線のプロットデータ。式は曲面の式。

【例】回転放物面と線分の交点の座標を表示する。曲面は表示されていなくてもよい。

```
Putpoint3d(["A",[0,-3,0],"B",[0,3,2]],"fix");
Spaceline([A,B]);
fd=["z=4-(x^2+y^2)","x=R*cos(T)","y=R*sin(T)","R=[0,2]","T=[0,2*pi]","e"];
println("Intersect="+Intersectcrvsf("1","AB3d",fd));
```

実行すると、コンソールに

```
Intersect=[[0,-1.91,0.36,1.18],[0,1.57,1.52,1.76]] と表示される。
```

関数 IntersectsgpL(点名, 線分, 面, 描画方法)

機能 空間の線分と平面の交点を作図する

説明 引数の線分は線分の端点を "A-B" の形もしくは座標のリスト形で与える。

引数の面は、面内の 3 点を "C-D-E" の形もしくは座標のリストで与える。

描画方法は、"put" または "draw" で、描画方法を指定しない場合は "draw" と同じ。"draw" では交点が緑の点で表示されるだけで、幾何点はできない。"put" では幾何点を作る。

【例】 座標のリストで与える記述例

```
IntersectsgpL("P",[p1,p2],[p3,p4,p5],"draw");
```

【例】 立方体を平面で切った図を描く。

いろいろな手順が考えられるが、ここでは次の手順で描く。

(1) 立方体の頂点をとる。1 辺の長さを Hn とする。

ここでは軸上の点は Putaxes3d() でとる。

(2) 切断面を決める点 E,F,G を辺上の自由点として Putonseg3d() でとる。

(3) E,F,G を通る平面と、辺 AC,DY との交点を取り、M,N とする。

(4) 全体を多面体として面データを作って描画する。

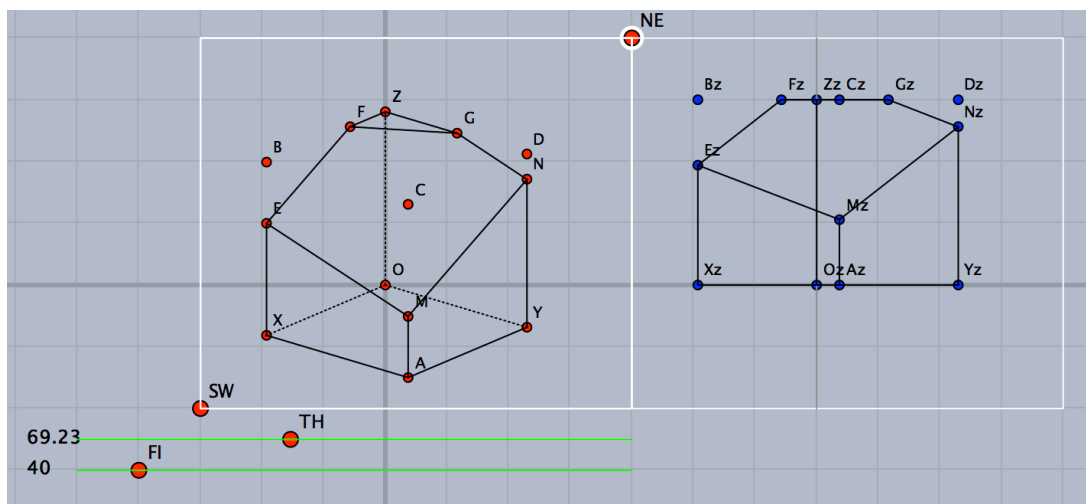
```
Hn=3;
Putaxes3d(Hn);
Putpoint3d("A",[Hn,Hn,0],"fix");
Putpoint3d("B",[Hn,0,Hn],"fix");
Putpoint3d("C",[Hn,Hn,Hn],"fix");
Putpoint3d("D",[0,Hn,Hn],"fix");
Putonseg3d("E",X,B);
Putonseg3d("F",Z,B);
Putonseg3d("G",Z,D);
IntersectsgpL("M","A-C","E-F-G","put");
IntersectsgpL("N","D-Y","E-F-G","put");
```

```

phd=Concatobj([ [O,X,A,Y], [X,A,M,E], [A,Y,N,M], [Y,N,G,Z,O],
[O,Z,F,E,X], [Z,F,G], [E,M,N,G,F]]);
VertexEdgeFace("1",phd,["Edg=nogeo"]);
Nohiddenbyfaces("1","phf3d1");

```

Cinderella の描画面はつぎのようになる。点 E,F,G をドラッグして、適当な位置の断面にできる。ただし、M,N は辺上にあることが条件である。

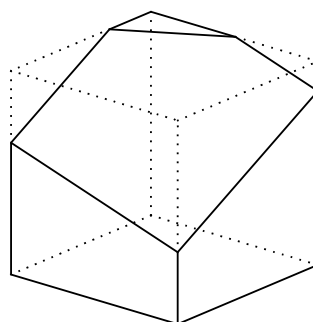
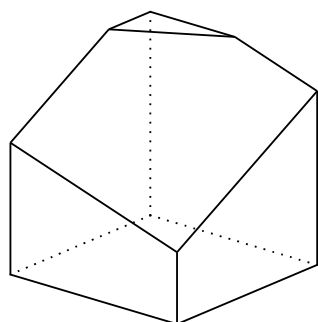


できた図は下図左。これに、次のスクリプトを追加すれば、断面上方の立方体の各辺も点線で描かれる。(下図右)

```

Spaceline([E,B,F],["do"]);
Spaceline([B,C,M],["do"]);
Spaceline([C,D,N],["do"]);
Spaceline([D,G],["do"]);

```



⇒ [関数一覧](#)

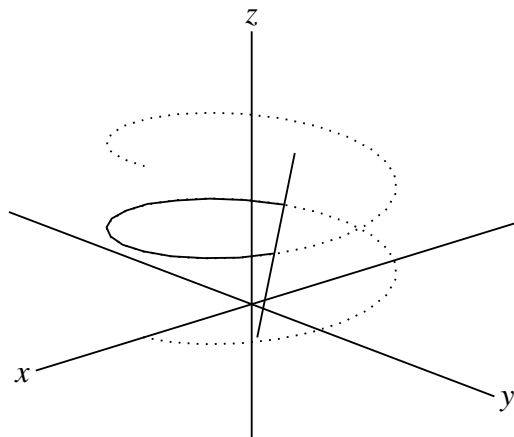
関数 Invparapt(座標, PD)

機能 描画面上の座標に対応する曲線上の点の座標を返す

説明 Cinderella の描画面上の座標を与えて、それに対応する曲線上の 3 次元座標を返す。空間内の曲線を作図すると、曲線の空間内のプロットデータとともに、描画面上に描くためのプロットデータも作られる。これを利用すると、描画面上の位置から曲線上の座標を求めることができる。

【例】螺旋と線分を描いたとき、描画面上での交点（空間内の交点ではない）に対応する螺旋上の点の座標を求め部分曲線を描く。

```
Spaceline("1", [[-1,-1,-1],[1,2,3]]);  
Spacecurve("1", "[2*cos(t),2*sin(t),0.2*t]", "t=[0,4*pi]", ["do"]);  
tmp=Intersectcrvs("sl2d1","sc2d1");  
p1=Invparapt(tmp_1,"sc3d1");  
p2=Invparapt(tmp_2,"sc3d1");  
Partcrv3d("1",p1,p2,"sc3d1");
```



ここで、sl2d1,sc2d1 は線分と螺旋の描画面上での（平面の）プロットデータである。Intersectcrvs() で平面上の交点の座標（複数あるのでリストが返る）を求め、Invparapt() で対応する螺旋上の点の座標を求めて部分曲線を描いている。実際に交わる点での部分曲線ではないことに注意。

関数 Mkbezierptcrv3d(点リスト)

機能 制御点を自動的にとる空間ベジェ曲線

説明 リストで与えた点に対し、制御点を自動的に生成してベジェ曲線を描く。

制御点は、2つの点に対して、その点を端点とする線分上に2つ作られる。これを適宜移動して任意の曲線にすることができる。[空間ベジェ曲線 Bezier3d\(\)](#) を参照のこと。

【例】 `Mkbezierptcrv3d(["A","B","C","D"]);`

線分 AB 上に 2 点 `a1p,a2p`, 線分 BC 上に 2 点 `a2p,a2q`, 線分 CD 上に 2 点 `a3p,a3q` ができる。

[⇒ 関数一覧](#)

関数 `Nohiddenbyfaces(name,PD1,PD2,option1,option2)`

機能 面に対し曲線を陰線処理する

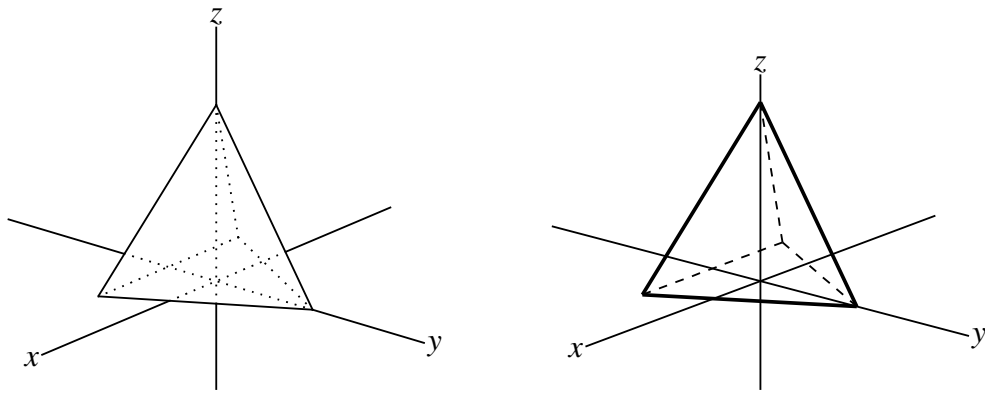
説明 PD2 で与えられた面に対し、曲線 PD1 の面に隠れている部分を陰線処理する。
引数 PD1 を省略するとすべての曲線が対象となる。陰線処理された線は初期設定では点線で表される。この線種は option2 で変更できる。たとえば、["da"] とすると破線になる。option1 は曲線全体の option であるので、option2 だけを指定する場合は、option1 として空リスト [] が必要である。

【例】座標平面上に正四面体を描き、各軸と正四面体の辺を陰線処理する。(下図左)

```
Xyzax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,4]");  
Putpoint3d("A", 2*[-1,-1/sqrt(3),0], "fix");  
Putpoint3d("B", 2*[1,-1/sqrt(3),0], "fix");  
Putpoint3d("C", 2*[0,sqrt(3)-1/sqrt(3),0], "fix");  
Putpoint3d("D", 2*[0,0,sqrt(3)], "fix");  
phd=Concatobj([A,B,C], [A,B,D], [A,C,D], [B,C,D]);  
VertexEdgeFace("1", phd, ["Edg=nogeo"]);  
Nohiddenbyfaces("1", "phf3d1");
```

`VertexEdgeFace("1", phd, ["Edg=nogeo"]);` によって、辺、頂点、面のプロットデータが作られる。`phf3d1` は、面のプロットデータである。

ここで、`Nohiddenbyfaces("1", "phe3d1", "phf3d1", ["dr,2"], ["da"]);` とすると、座標軸は陰線処理されず、正四面体の辺 (`phe3d1`) だけが陰線処理されて破線で描かれる。四面体は太く描かれる。(下図右)



同様に、

```
Nohiddenbyfaces("1","ax3d","phf3d1",[],["da"]);
```

とすれば、座標軸だけが陰線処理されて破線で描かれる。

[⇒ 関数一覧](#)

関数 Parapt(座標)
機能 点の投影面での座標
説明 引数の空間座標に対応する Cinderella の描画面の座標を返す。

【例】`Parapt([2,1,5]);` により、点 (2,1,5) が表示されている描画面の座標、たとえば `[-0.52,3.27]` が返される。

[⇒ 関数一覧](#)

関数 Perpplane(点名, 点, ベクトル, option)
機能 点を通り線分に垂直な平面上に基準点を 2 つとる
説明 引数の点名は、作成する 2 点で "A-B" の形
 第 2 引数は通る点の名称または座標
 第 3 引数は法線ベクトル
 option は "put" で、2 つの幾何点を作図する。option がない場合は幾何点は作らず、無名の点のみを表示する。put 以外の文字列を書いたときは無効な命令とし、何も作成されない。

記述例を示すと

```
Perpplane("A-B","P",[1,1,1],"put");
```

点 P を通り、法線ベクトル (1,1,1) に垂直な平面上に点 A,B をとる。

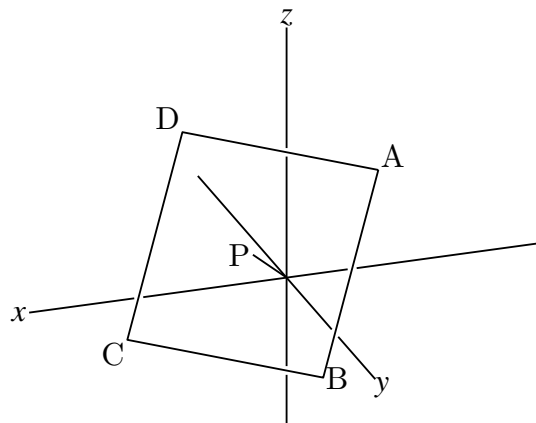
```
Perpplane("A-B","P",P3d-03d);
```

点 P を通り、線分 OP に垂直な平面上に点 A,B をとる。

これらにおいて、PA と PB は垂直で、 $PA=PB=1$ となる。

【例】 ベクトル $\vec{p} = (1, 1, 1)$ に垂直で点 $(1, 1, 1)$ を通る平面を描く。

```
Putaxes3d(2);
Putpoint3d(["P",[1,1,1]],"fix");
Ptseg3data();
Perpplane("E-F","P",P3d-03d,"put");
vec1=2*(E3d-P3d);
vec2=2*(F3d-P3d);
Putpoint3d(["A",P3d+vec1+vec2],"fix");
Putpoint3d(["B",P3d+vec1-vec2],"fix");
Putpoint3d(["C",P3d-vec1-vec2],"fix");
Putpoint3d(["D",P3d-vec1+vec2],"fix");
Spaceline([A,B,C,D,A]);
Letter([P,"w","P",A,"ne","A",B,"e","B",C,"ws","C",D,"nw","D",]);
Skeletonparadata("1");
```



[⇒ 関数一覧](#)

関数 Perppt(点名, 点, 点リスト,option)

機能 平面に下ろした垂線の足を求める

説明 第2引数の点から, 第3引数の点リストで決まる平面に下した垂線の足を, 第1引数の名前の点とする。

オプションは次の通り。デフォルトは "draw"

draw : 点を打つ。幾何点は作らない

put : 幾何点を作る

none : 計算だけ行い, 点は作図しない。

【例】原点から点 ABC を通る平面に下した垂線の足 H の座標を求める。

`Perpnt("H","O","A-B-C","none");` 表示はされない。

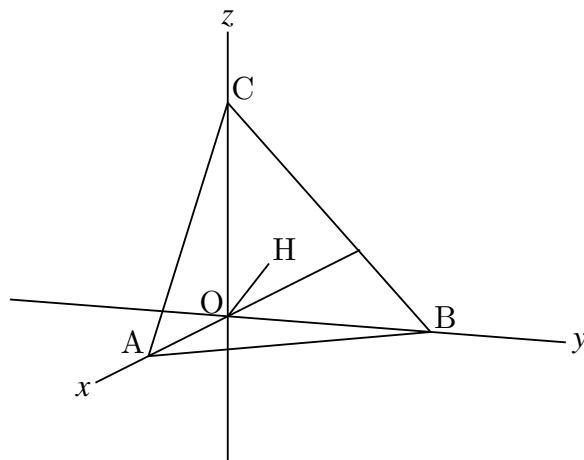
`Perpnt("N","O","A-B-C");` H の位置に緑色の点が表示される。

`Perpnt("N","O","A-B-C","put");` 幾何点 H が作図される。

いずれの場合も、H の座標は変数 H3d に代入される

作図例

```
Xyzax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,4]");  
Putpoint3d("O", [0,0,0], "fix");  
Putpoint3d("A", [3,0,0], "fix");  
Putpoint3d("B", [0,3,0], "fix");  
Putpoint3d("C", [0,0,3], "fix");  
Perpnt("H", "O", "A-B-C", "put");  
Spaceline([A,B,C,A]);  
Spaceline([O,H]);  
Letter([A,"nw","A",B,"ne","B",C,"ne","C",O,"nw","O",H,"ne","H"]);
```



関数 Partcrv3d(name, 始点, 終点, PD)

機能 部分曲線のプロットデータを作成する

説明 曲線 PD において、始点から終点までのプロットデータを作成する。

始点と終点は、プロットデータの番号もしくは曲線上にとった点の識別名で示す。

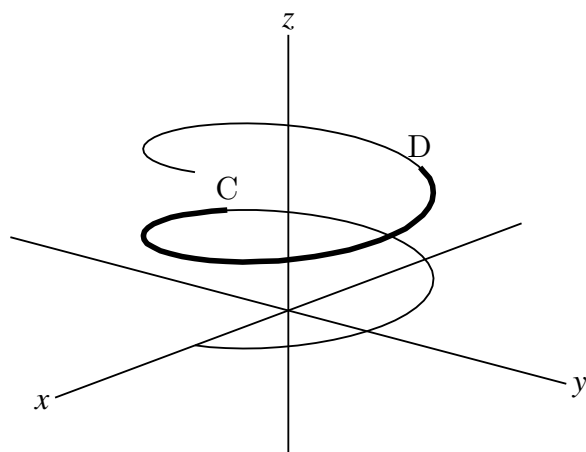
【例】螺旋を描き、C から D までの部分を太くする。PutonCurve3d() で螺旋上に点 C,D をとり、ドラッグして適当な位置に移動する。

```

Xyzax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,4]");
Spacecurve("1", "[2*cos(t),2*sin(t),0.2*t]", "t=[0,4*pi]", ["Num=100"]);
PutonCurve3d("C", "sc3d1");
PutonCurve3d("D", "sc3d1");
Partcrv3d("1", C, D, "sc3d1", ["dr,3"]);
Letter([C, "n2", "C", D, "n2", "D"]);

```

ここで, "sc3d1" は, 螺旋のプロットデータ, "part3d1" は, 部分曲線のプロットデータである。

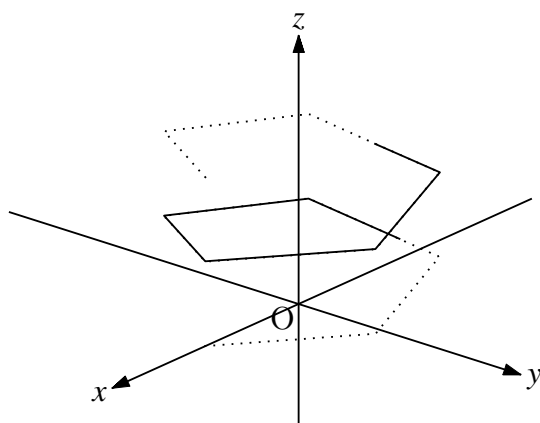


【例】稲妻状の螺旋を点線で描き, その一部を実線にする。位置はプロットデータの番号で示す。小数にすると曲線を分割している線分の途中の位置になる。

```

Spacecurve("1", "[2*cos(t),2*sin(t),0.2*t]", "t=[0,4*pi]", ["Num=10", "do"]);
Partcrv3d("1", 3.3, 8.5, "sc3d1");

```



関数 Phparadata(name,name2,options)

機能 多面体を陰線処理して描く

説明 多面体を陰線処理して描く。多面体は面データ（頂点リストと面リスト）を与えて, VertexEdgeFace() でプロットデータを作る。このプロットデータに対し, 隠れている

面（辺）を非表示にして表示する。第 1 引数は通常の name, 第 2 引数の name2 は, VertexEdgeFace() で与えた name と同じものとする。

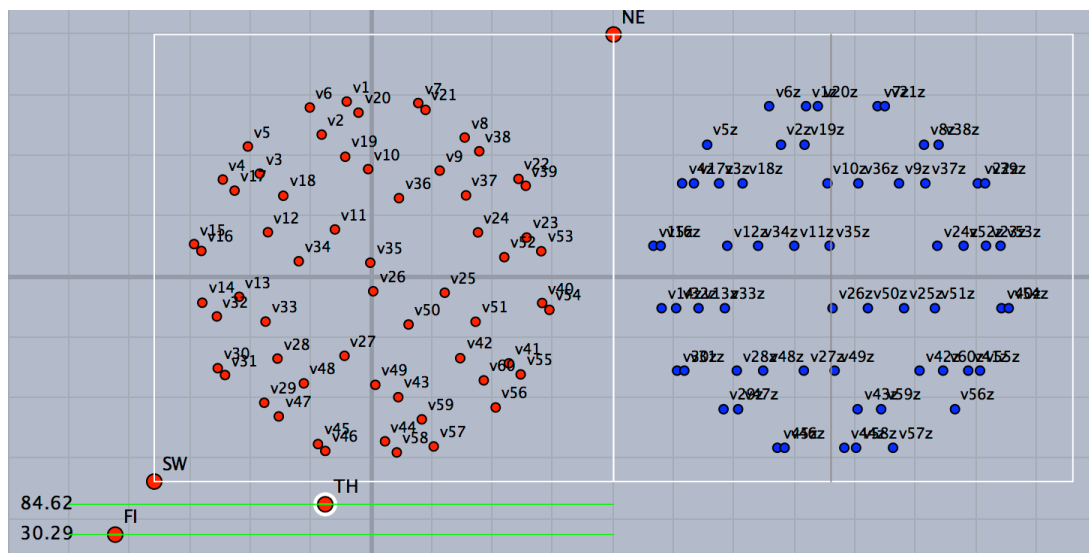
options は, 全体の線種 ("dr,2" など) と, 陰線の線種を "Hidden=線種" で指定できる。デフォルトでは陰線は表示しない。

【例】小林・鈴木・三谷による多面体データ polyhedrons obj から, s06 の切頂二十面体（サッカーボール型）を描く。

```
Setdirectory( Dirhead+"/data/polyhedrons_obj");
phd=Readobj("s06.obj",["size=3"]);
Setdirectory(Dirwork);
VertexEdgeFace("s06",phd,["Edg=nogeo"]);
Phparadata("1","s06");
```

VertexEdgeFace() の name は通常の "1" でもよい。その場合は, Phparadata("1","1"); とするが, わかりにくいので上のようにした。

実行すると, Cinderella の描画面は次のように頂点だけが描かれる。



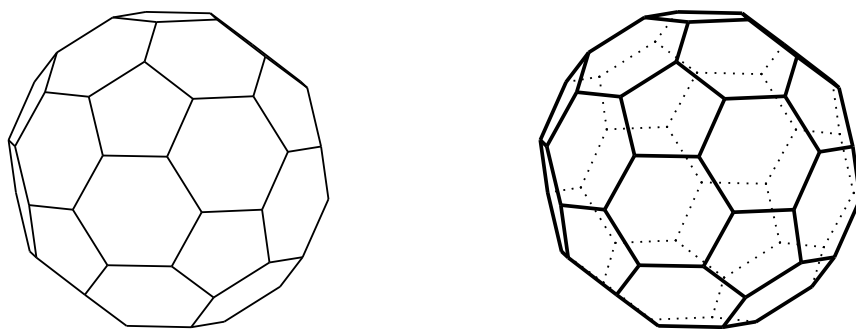
「Texview」「Exekc」ボタンを押すと陰線処理したデータが Fhead.txt に書き出され, これを用いて Fhead.tex が作成される。そのため, Exekc ボタンを押してから処理が完了するまでにかなり時間がかかる。(下図左)

なお, もう一度実行すると, こんどは, Fhead.txt ができているために, これを読み込んで Cinderella の画面でも陰線処理された図が描かれる。

全体の線種と, 陰線の線種を

```
Phparadata("1","s05",["dr,2","Hidden=do"]);
```

で指定したのが下図右である。



【注意】

polyhedrons obj のデータを使って、続けて異なる多面体を描きたい場合は注意が必要である。Readobj() だけを変更して別のデータを読めばよさそうであるが、VertexEdgeFace() の name も（したがって、Phparadata() の第 2 引数も）書き換えないと、前のデータが残っていてうまくいかない。たとえば、上のコードで切頂十二面体を描いた後、正八面体 (r02) を描こうとするならば、

```
Setdirectory( Dirhead+"/data/polyhedrons_obj");
phd=Readobj("r02.obj",["size=3"]);
Setdirectory(Dirwork);
VertexEdgeFace("2",phd,["Edg=nogeo"]);
Phparadata("1","2");
```

のようにする。

[⇒ 関数一覧](#)

関数 Projcoordpara(座標)

機能 投影座標を求める

説明 空間座標を投影した座標を求める。

戻り値の第 1, 第 2 要素は Cinderella の描画面の x,y 座標。第 3 要素は xy 平面に垂直な z の座標で、投影面からの（符号付）距離を表す。

【例】Projcoordpara([3,1,2]);

戻り値は [-0.65,1.7,3.27] となる。

関数 Putaxes3d([x,y,z])

機能 軸上に幾何点を作る。

説明 引数のリスト $[x,y,z]$ に対し、点 $X(x,0,0)$, $Y(0,y,0)$, $Z(0,0,z)$ および 原点 O を主画面上にとり、副画面上に対応する点 X_z , Y_z , Z_z , O_z を作る。すでに同じ名称の点がある場合は、指定された位置に移動する。

引数は、実数にすることもでき、 $Putaxes3d(a)$ は、 $Putaxes3d([a,a,a])$ と同じになる。

【例】 $Putaxes3d(5);$ 原点と、 $x(5,0,0)$, $y(0,5,0)$, $z(0,0,5)$ を作る。

$Putaxes3d([1,2,3]);$ 原点と、 $x(1,0,0)$, $y(0,2,0)$, $z(0,0,3)$ を作る。

⇒ [関数一覧](#)

関数 $PutonCurve3d(\text{点名}, PD)$

機能 空間曲線上に点をとる

説明 プロットデータ PD の曲線上に、点名の点をとる。

とった点は固定点ではなく、曲線上にインシデントとなる。したがって、ドラッグして曲線上を動かすことができる。例は [Partcrv3d\(\)](#) を参照のこと。

関数 $Putonseg3d(\text{点名}, \text{点1}, \text{点2})$

機能 線分上に点を取る

説明 点1と点2の中点に、指定された名前の点を取る。点1と点2が線分として結ばれていなくてもよい。とった点は線分にインシデントとなる（線分が描かれていなくても）。点1と点2はリストにすることもできる。

【例】 A と B の中点に点 C をとる。つぎのいずれでもよい。

$Putonseg3d("C", A, B);$

$Putonseg3d("C", [A, B]);$

関数 $Putpoint3d(\text{リスト}, \text{option})$

機能 空間に幾何点を作図する

説明 点の名称と座標を与えて点を作図する。複数の点を一度に作図できる。

option は、"fix" または "free" (デフォルト)。

"fix" をつけると、固定点（ドラッグで移動できない点）とする。同じ名称の点がすでに存在する場合は、指定した位置に移動して固定点とする。

"free" をつけると、自由点（ドラッグで移動できる）とする。同じ名称の点がすでに存在する場合はなにもしない。

【例】いくつか記述例を示す。

```
Putpoint3d(["A",[2,1,3]]);  
Putpoint3d(["A",[1,1,1],"C",[1,0,1]],"fix");  
Putpoint3d(["A",[2,1,3]],"free");
```

なお、この関数は幾何点を作るものであり、TeX には出力されない。TeX に点を出すするには、[Pointdata\(\)](#) または [Drawpoint\(\)](#) を併用する。

空間における点の座標は、点名に"3d"を付加した名前の変数に代入される。たとえば、点 A の座標は A3d である。これにより、点の座標を取得できる。

[⇒ 関数一覧](#)

関数 Readobj(ファイル名)

機能 obj ファイルを読み込む。

説明 小林・鈴木・三谷による多面体データ polyhedrons_obj を読み込む。

オプションは ["size=n"] で、n 倍したデータにする。負の数にすると上下が反転される。

データは KeTCindy の data フォルダの中にある。したがって、次のようなスクリプトを書く。読み込むのは一度だけなので、Draw スロットではなく Initialization スロットに置けばよいが、コードの可読性を高めるには Draw スロットでもよい。

```
Setdirectory( Dirhead+"/data/polyhedrons_obj");  
polydt=Readobj("r02.obj");  
Setdirectory(Dirwork);
```

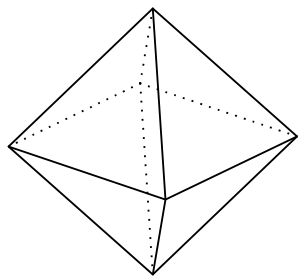
これで、r02.obj データが、変数 polydt に代入される。

この多面体データは、頂点リストと面リストからなっているが、頂点リストは座標のリストなので、読み込んで表示するときには、点の名称を v1,v2,・・・とする。

読み込んだあとの使い方を含めて例を示す。

【例】正八面体を描く

```
Setdirectory( Dirhead+"/data/polyhedrons_obj");  
polydt=Readobj("r02.obj",["size=2"]);  
Setdirectory(Dirwork);  
VertexEdgeFace("1",polydt,["Edg=nogeo"]);  
Nohiddenbyfaces("1","phf3d1");
```



[⇒ 関数一覧](#)

関数 Reflectpoint3d(座標, リスト)

機能 点の鏡映点を求める

説明 第 2 引数のタイプにより, 点に関する鏡映, 直線に関する鏡映, 面に関する鏡映のそれぞれの点の座標を返す。

【例】点 A,B,C,D を空間にとり, 点 A の鏡映点の座標を求める。

点 B に関する鏡映点 Reflectpoint3d(A3d,[B3d]);

直線 BC に関する鏡映点 Reflectpoint3d(A3d,[B3d,C3d]);

平面 BCD に関する鏡映点 Reflectpoint3d(A3d,[B3d,C3d,D3d]);

関数 Rotate3pt(座標,vec, 角度, 中心点)

機能 点を回転する

説明 第 1 引数の座標の点を回転する。vec は回転軸の方向ベクトル。中心点は方向ベクトルの始点。第 4 引数を省略した場合は原点とする。

【例】点 A を (0,-1,0) に置いたときの記述例と結果 (戻り値)

Rotate3pt(A3d,[0,0,1],pi/2); 戻り値は [1,0,0]

Rotate3pt(A3d,[0,0,1],pi/2,[1,1,1]); 戻り値は [3,0,0]

関数 Rotatedata3d(name,PD リスト,vec, 角度,options)

機能 プロットデータを回転

説明 プロットデータを, 原点を始点とするベクトル vec 周りに回転する。複数のプロットデータをまとめて回転することができる。

options として, 中心点 (vec の始点), 線種を指定することができる。

【例】コード例と結果を示す。

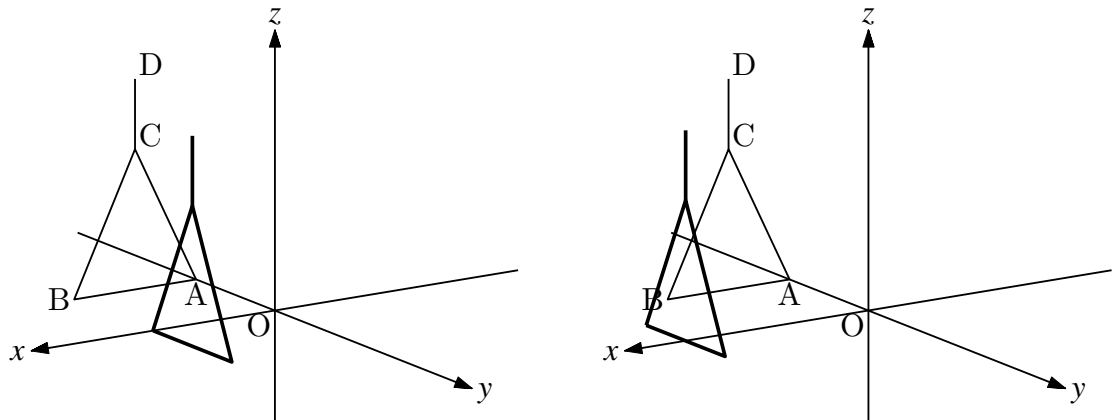

```

Xyzax3data("", "x=[-5,4]", "y=[-5,5]", "z=[-5,4]", ["a", "0"]);
Putpoint3d(["A", [0,-2,0], "B", [2,-2,0], "C", [1,-2,2], "D", [1,-2,3]], "fix");
Spaceline("1", [A,B,C,A]);
Spaceline([C,D]);
Rotatedata3d("1", ["sl3d1", "CD3d"], [0,0,1], pi/2, ["dr,2"]);
Letter([A, "s", "A", B, "w", "B", C, "ne", "C", D, "ne", "D"]);

```

これを

`Rotatedata3d("1", ["sl3d1", "CD3d"], [0,0,1], pi/2, [[1,0,0], "dr,2"]);`
 とした場合が右図である。



関数 Sf3data(name, リスト, options)

機能 陰線処理なしの曲面をワイヤースケルトンモデルで描く

説明 リストは、関数式と定義域、境界指定をそれぞれ文字列にしたもののリスト。

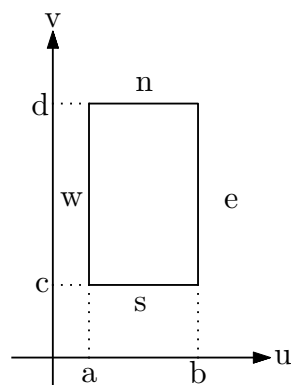
options は、解像度（各変数に対応する分割数）とメッシュの密度、境界指定。

解像度は、"Num=[a,b]" で指定。初期値は a,b とも 25

メッシュ密度は、縦横で "Wire=[a,b]" で指定。初期値は a,b とも 20

境界指定は "ewsn" で行う。"ewsn" の意味は次のように考える。

変数が u, v のとき、 u, v 平面において、 $a \leq u \leq b, c \leq v \leq d$ の矩形を考え、
 その東西南北 (ewsn) の境界を示す。

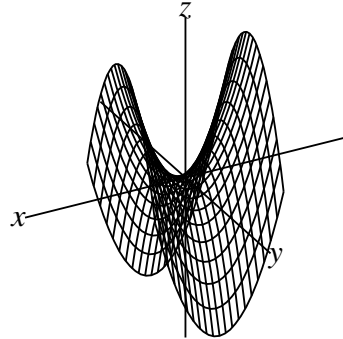


関数式のパターンはつぎの3通り。

(1) $z = f(x, y)$

【例】： $z = x^2 - y^2$ を定義域 $x = [-2, 2], y = [-2, 2]$ で描画する。

```
Sf3data("1", ["z=x^2-y^2", "x=[-2,2]", "y=[-2,2]"]);
```



【例】：同じく、解像度を x,y とも 10、メッシュの数を縦横とも 10 にする。

解像度、メッシュ密度とも下げるので粗い描画となる。

```
Sf3data("1", ["z=x^2-y^2", "x=[-2,2]", "y=[-2,2]",  
  ["Num=[10,10]", "Wire=[10,10]"]);
```

(2) $z = f(x, y), x = g(R, T), y = h(R, T)$

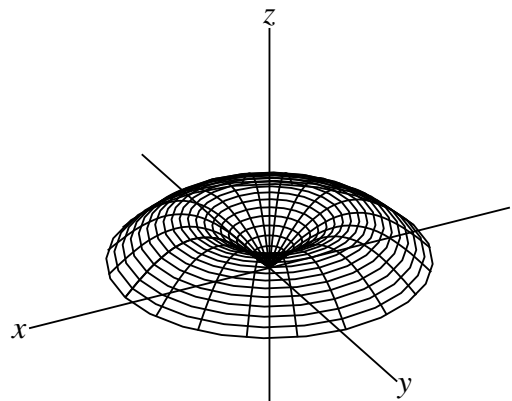
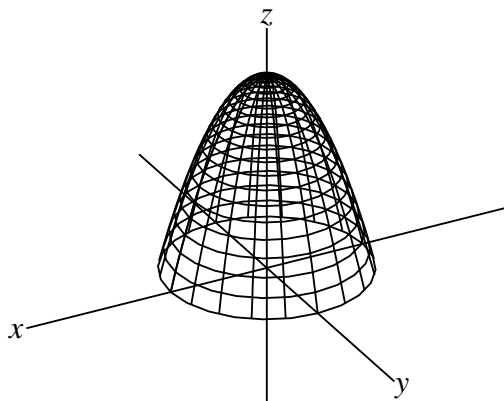
x, y の媒介変数 r, t は大文字にする。

【例】：次図左

```
fd=["z=4-(x^2+y^2)", "x=R*cos(T)", "y=R*sin(T)", "R=[0,2]", "T=[0,2*pi]";  
Sf3data("1", fd);
```

【例】：次図右

```
fd=["z=sin(sqrt(abs(x^2+y^2)))", "x=r*cos(t)", "y=r*sin(t)",  
  "r=[0,3]", "t=[0,2*pi]";  
Sf3data("1", fd);
```

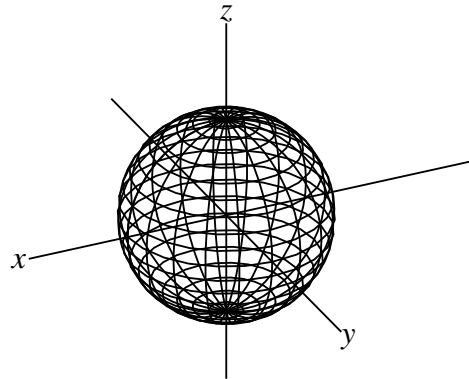


(3) $x = f(u, v), y = g(u, v), z = h(u, v),$

この場合, (2) と区別するために, "p" を先頭につけておく。

【例】球面

```
fd=["p","x=2*sin(u)*cos(v)","y=2*sin(u)*sin(v)","z=2*cos(u)",  
    "u=[0,pi]","v=[0,2*pi]"];  
Sf3data("1",fd);
```



⇒ 関数一覧

関数 Sfbdparadata(name, 式,options1,options2)

機能 空間曲面の陰線処理

説明 曲面について陰線処理を行う。 曲面の式は Sf3data() の場合と同じ。ただし, (2) と (3) のパターンでは式の最後に境界の指定 ewns を明示する必要がある。(3) では境界の指定を" "(中は半角スペース) とする。省略すると不要な境界線が表示されることがある。境界指定については, Sf3data() の項を参照のこと。

options1 は, "Wait=n","r","m", 線種。Wait の初期値は 30。

"r","m"に関しては, オプションなしまたは,"" のとき

- i) データファイルがなければ, 新しく作る
- ii) データファイルが既にあればそれを読み込む

"m" のとき, 強制的にデータファイルを作り直す。

"r" のとき, すでにあるデータファイルを読み込む。

options2 は, 陰線に関する option で, "nodisp" または 線種。"nodisp" にすると陰線は消去されるため, 輪郭線だけが描かれることになる (デフォルト)。options2 だけを指定する場合は, options1 を空リストにしてつける必要がある。

【注意】 この処理は Scilab で行うが, 非常に時間がかかるため, この関数を実行した状態で画面上のスライダを動かしたりしないこと。何か操作を行う場合は, いったん

行頭に//を書いてコメント化してから行う。

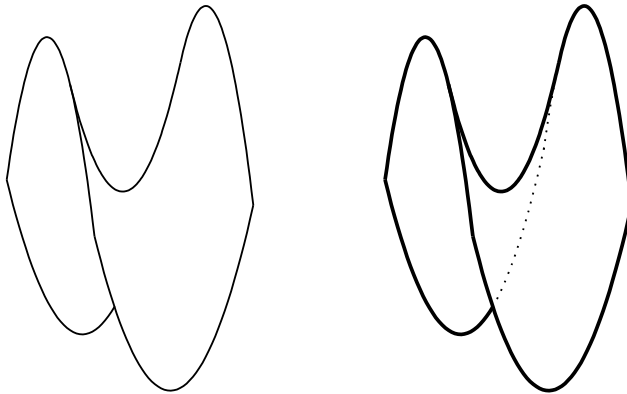
【例】サドル面

陰線を消去して表示（デフォルト）（下図左）

```
fd=["x=x^2-y^2","x=[-2,2]","y=[-2,2]";  
Sfbdparadata("1",fd);
```

全体を実線で太めにして、陰線を点線で表示（下図右）

```
fd=["x=x^2-y^2","x=[-2,2]","y=[-2,2]";  
Sfbdparadata("1",fd,["dr,2"],["do"]);
```



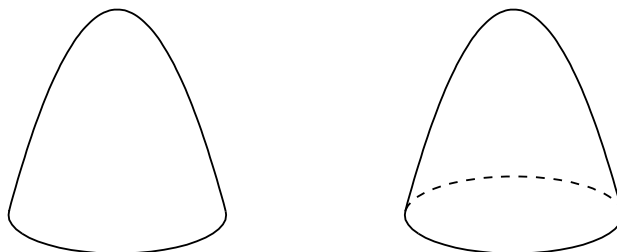
【例】放物面

デフォルト（下図左）

```
fd=["z=4-(x^2+y^2)","x=R*cos(T)","y=R*sin(T)","R=[0,2]","T=[0,2*pi]","e"];  
Sfbdparadata("1",fd);
```

陰線を破線で表示（下図右）

```
fd=["z=4-(x^2+y^2)","x=R*cos(T)","y=R*sin(T)","R=[0,2]","T=[0,2*pi]","e"];  
Sfbdparadata("1",fd,[],["da"]);
```



⇒ [関数一覧](#)

関数 Skeletonparadata(name,PD リスト,PD リスト,option)

機能 陰線処理（スケルトン処理）をおこなう

説明 描画されている線と軸について陰線処理をおこなう。

第2引数の線（プロットデータ）が、第3引数の線（プロットデータ）によって隠される部分を消去する。第2、第3引数を省略した場合は、すべての線について、互いの陰線処理をおこなう。option で消去する部分の長さを指定できる。

【例】螺旋と線分、座標軸の陰線処理

次のように螺旋と線分、座標軸を描いておく。

```
Xyzax3data("", "x=[-5,5]", "y=[-5,4]", "z=[-5,3]");  
Putpoint3d(["A", [0,-2,-2]], "fix");  
Putpoint3d(["B", [-1,1,3]], "fix");  
Spaceline([A,B]);  
Spacecurve("1", "[2*cos(t), 2*sin(t), 0.2*t]", "t=[0,4*pi]", ["Num=100"]);
```

座標軸のプロットデータは ax3d, 線分は AB3d, 螺旋は sc3d1 である。これに対し、

```
Skeletonparadata("1");
```

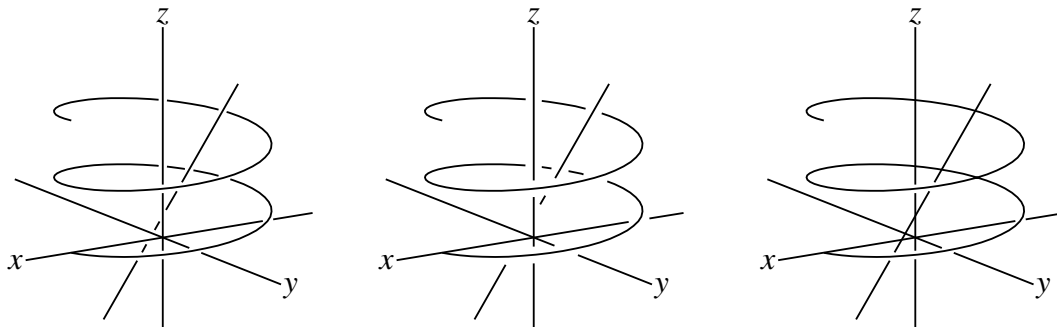
描画されている線と軸について陰線処理をおこなう。(図左)

```
Skeletonparadata("1", [2]);
```

重なった部分の空きを2にする。(図中央)

```
Skeletonparadata("1", ["AB3d", "ax3d"], ["sc3d1"]);
```

螺旋によって隠れる部分だけ消去する。(図右)



このほか、

```
Skeletonparadata("1", ["AB3d", "ax3d"], ["sc3d1"], [2]);
```

```
Skeletonparadata("1", ["AB3d"], ["ax3d", "sc3d1"]);
```

も可能である。

[⇒ 関数一覧](#)

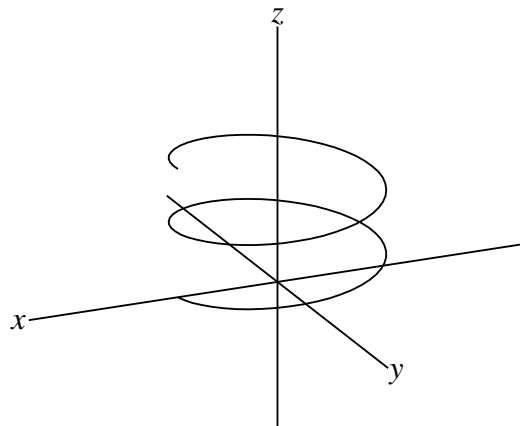
関数 Skeletoindatacindy(name,PD リスト,PD リスト,option)

機能 陰線処理（スケルトン処理）を Cindyscript でおこなう
説明 描画されている線と軸について陰線処理をおこなう。
 Scilab を呼ばず、Cindyscript だけで行うので、多面体などでインタラクティブに図を動かすことができる。ただし、曲線では処理が遅くなる。

関数 Spacecurve(name, 式, 定義域,options)
機能 空間曲線を描く
説明 媒介変数で表された曲線を描く。option は解像度 Num

【例】螺旋を描く

```
Spacecurve("1", "[2*cos(t), 2*sin(t), 0.2*t]", "t=[0, 4*pi]", ["Num=100"]);
```



関数 Spaceline(リスト)
機能 折れ線を描く
説明 点の名称または座標のリストを与えて折れ線を描く。平面での Listplot() にあたる。
 【例】Spaceline("1", [[2,5,1],[4,2,3]]); 指定された2点を結んだ線分を描く。
 Spaceline([A,B,C,A]); 作図されている2点 A,B,C を結んだ三角形を描く。
 options は線種 (dr,da,do)

関数 Translatedata3d(name,PD, 平行移動量)
機能 空間プロットデータを平行移動
説明 PD で表される図形を，平行移動する。

【例】曲線 sc3d1 を y 軸方向に 2 だけ平行移動する。

```
Translatedata3d("1", ["sc3d1"], [0, 2, 0]);
```

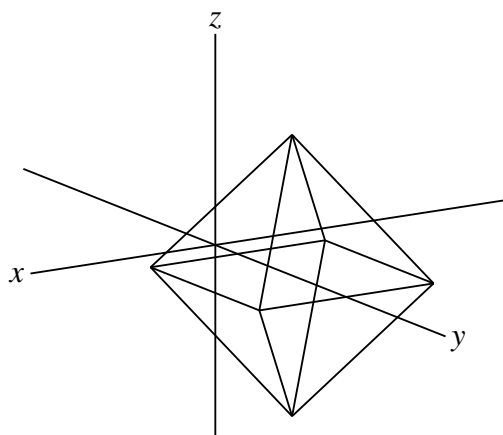
結果として，もとの曲線と平行移動した曲線の 2 つが描かれる。

【例】多面体の平行移動

VertexEdgeFace() で描いた多角形はこの関数では平行移動できないので、面データを直接操作して平行移動を行う。

たとえば、小林・鈴木・三谷による多面体データ polyhedrons obj を用いて正八面体を描く場合、次のようにする。y 軸方向に 2 だけ平行移動する場合である。

```
Setdirectory( Dirhead+"/data/polyhedrons_obj");  
phd=Readobj("r02.obj",["size=2"]);  
Setdirectory(Dirwork);  
dn=length(phd_1);  
repeat(dn,s,phd_1_s=phd_1_s+[0,2,0]);  
VertexEdgeFace("1",phd,["Edg=nogeo"]);
```



[⇒ 関数一覧](#)

関数	Translatept3d(座標, 平行移動量)
機能	空間点を平行移動
説明	点を平行移動する。

【例】点 A(1,0,0) を (-1,1,1) だけ平行移動した点を B とする。点 A の空間座標は A3d で表される。

```
Putpoint3d(["A",[1,0,0]],"fix");  
pt=Translatept3d(A3d,[-1,1,1]);  
Putpoint3d(["B",pt],"fix");
```

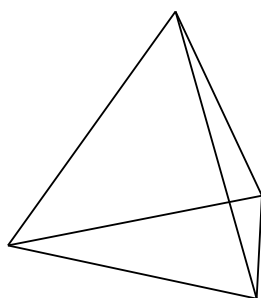
関数 VertexEdgeFace(name, 面データ,options)

機能 多面体の面データを用いて多面体を描く

説明 面データは、頂点リストと、面リストからなる。面リストは、各面を構成する頂点を、外側から見て反時計回り（左回り）に並べたリストである。
 たとえば、四面体 ABCD の面データは、[[A,B,C,D],[1,2,3],[1,2,4],[1,3,4],[2,3,4]] となる。
 4点 A,B,C,D をとっておき、このリストを引数に与えると、四面体の辺が描かれる。このとき、option がなければ、Cinderella の幾何要素として辺が描かれるが、それだけでは TeX には書き出されないので注意が必要である。(変数未定義のエラーになる)
 TeX に書き出すには、option として ["Edg=nogeo"] をつける。このときは各辺は幾何要素にならない。
 また、"Pt=free" option をつけると、頂点は自由点となり、マウสดラッグで動かすことができる。
 生成されるプロットデータは、
 phv3d：頂点のリスト
 phe3d：辺のリスト
 phf3d：面リスト
 なお、それぞれ末尾に name が付加される。

【例】：正四面体を描く

```
Putpoint3d("A",2*[-1,-1/sqrt(3),0],"fix");
Putpoint3d("B",2*[1,-1/sqrt(3),0],"fix");
Putpoint3d("C",2*[0,sqrt(3)-1/sqrt(3),0],"fix");
Putpoint3d("D",2*[0,0,sqrt(3)],"fix");
phd=Concatobj([ [A,B,C], [A,B,D], [A,C,D], [B,C,D] ]);
VertexEdgeFace("1",phd,["Edg=nogeo"]);
```



関数 Wireparadata(name,PD, 式, 整数, 整数,options1,options2)
機能 曲面のワイヤフレームを陰線処理して描く
説明 PD は、第3引数の式で描いたワイヤフレームモデルのプロットデータ名。第4, 第5引数は分割線の数。
options1 には "r","m","Wait=n" が指定できる。Wait の初期値は 30。
"r","m"に関しては、オプションなしまたは,"" のとき
i) データファイルがなければ、新しく作る
ii) データファイルが既にあればそれを読み込む
"m" のとき、強制的にデータファイルを作り直す。
"r" のとき、すでにあるデータファイルを読み込む。
options2 には陰線の表示方法について "nodisp" または線種 がある。デフォルトは"nodisp" 。何も指定しないときは, [""] (空文字) を書いておく。
曲面の陰線処理を行うので, Sfbdparadata() とペアで使う。

【注意】

この処理は Scilab で行うが, Sfbdparadata() とペアで使うために非常に時間がかかる。コンピュータの性能にもよるが2分以上かかることもある。したがって、この関数を実行した状態で画面上のスライダを動かしたりしないこと。何か操作を行う場合は、いったん行頭に//を書いてコメント化してから行う。球面やトーラスのタイプでは、Texview や Exekc ボタンの反応も鈍くなる。Exekc が動作しない（タイムアウト）場合は、kc.sh (Mac) / kc.bat (Windows) を直接起動すれば TeX の図ができる。

【例】 式のタイプ別に例を示す。

サドル面：下図左

```
fd=["z=x^2-y^2","x=[-2,2]","y=[-2,2]";
Sfbdparadata("1",fd);
Wireparadata("1","sfbd3d1",fd,4,5,[""]);
```

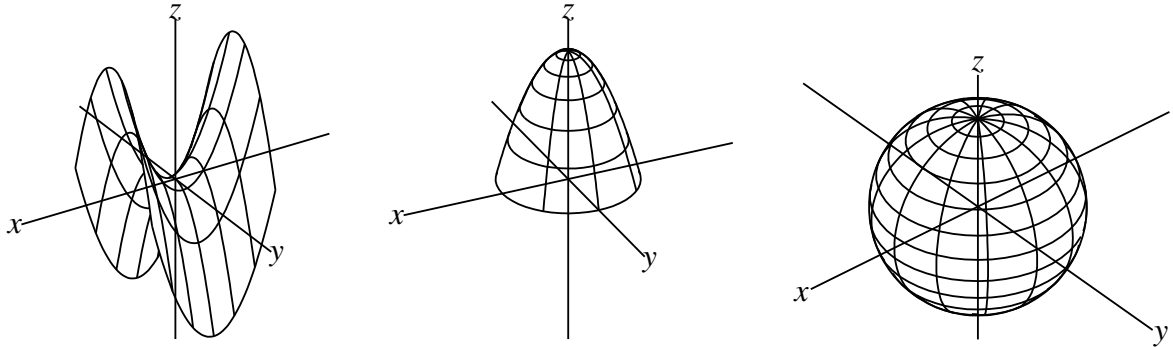
回転放物面：下図中央

```
fd=["z=4-(x^2+y^2)","x=r*cos(t)","y=r*sin(t)","r=[0,2]","t=[0,2*pi]","e"];
Sfbdparadata("1",fd);
Wireparadata("1","sfbd3d1",fd,5,7,[""]);
```

球面：下図右

このタイプでは、媒介変数形式の識別のため、先頭に "p" をつけておくとよい。

```
fd=["p","x=2*sin(u)*cos(v)","y=2*sin(u)*sin(v)","z=2*cos(u)","u=[0,pi]",  
    "v=[0,2*pi]","s"];  
Sfbdparadata("1",fd);  
Wireparadata("1","sfbd3d1",fd,12,12,[""]);
```

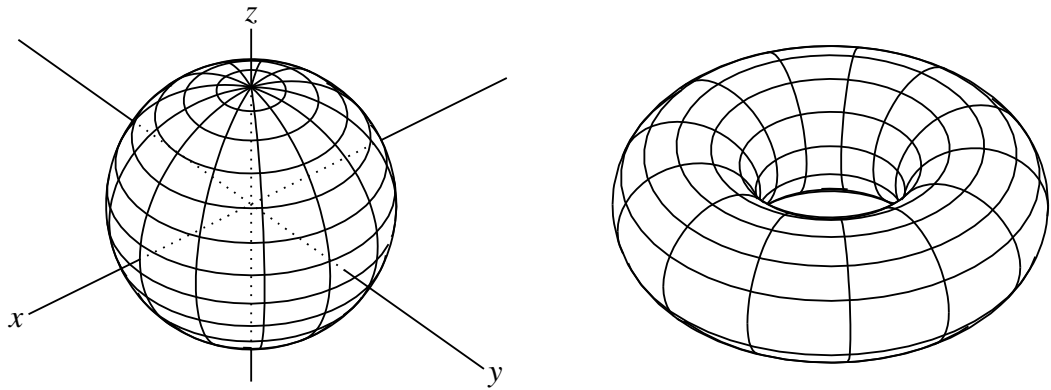


【例】球面で座標軸を陰線処理して点線で表す。(次図左)

```
fd=["p","x=2*sin(u)*cos(v)","y=2*sin(u)*sin(v)","z=2*cos(u)","u=[0,pi]",  
    "v=[0,2*pi]","s"];  
Sfbdparadata("1",fd);  
Wireparadata("1","sfbd3d1",fd,12,12,[""]);  
Crvsfparadata("1","ax3d","sfbd3d1",fd,[],["do"]);
```

【例】トーラスを描く (次図右)

```
fd=["p","x=(2+cos(u))*cos(v)","y=(2+cos(u))*sin(v)","z=sin(u)",  
    "u=[0,2*pi]","v=[0,2*pi]","s"];  
Sfbdparadata("1",fd);  
Wireparadata("1","sfbd3d1",fd,12,12,[""]);
```



[⇒ 関数一覧](#)

関数 `XYZax3data(name, x の範囲, y の範囲, z の範囲,options)`

機能 座標軸を描く

説明 描画面に座標軸を描く。name は空文字列でよい。プロットデータ ax3d が作成される。option は次の 2 つ。

矢じり : "an" : n は数字で矢じりの大きさ。n はなくてもよい。

原点 O : "Onesw" : nesw は微小位置。数字も付けられる。nesw をつけない場合の初期値は sw。

【例】 デフォルトの座標軸

```
XYZax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,5]");
```

矢じりをつける

```
XYZax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,5]", "a");
```

矢じりを大きくする

```
XYZax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,5]", ["a2"]);
```

原点の O を表示する。

```
XYZax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,5]", ["O"]);
```

原点の O の位置を調整して右上に表示する。やじりもつける。

```
XYZax3data("", "x=[-5,5]", "y=[-5,5]", "z=[-5,5]", ["a", "0e2n2"]);
```

【注意】 Putaxes3d() で点を取ると原点に点 O が作成される。この点名 O と表示が重複するのが煩わしい場合は、作図後にこの option をつけてから出力するとよい。

関数 `XYZcoord(P.x,P.y,Pz.y)`

機能 主副画面で決まる点の座標

説明 Cinderella の描画面上の点が表す空間座標を求める

点 P について、主画面の点 P に対応するのが副画面の Pz である。点 P の 2 次元座標は P.x,P.y で、Pz の y 座標は Pz.y で表される。これを引数として与えると、点 P の空間座標が返される。

【例】 点 A をドラッグして動かしたとき、A の座標を求める。

```
println(Xyzcoord(A.x,A.y,Az.y));
```

により、コンソールに座標が表示される。

7 逆引辞典


幾何の作図は、補助線などは Cinderella の作図ツールか Cindyscript を用いて描き、TeX に出力する線を KeTCindy で描くと、効率のよい作図ができる。

関数のグラフも、Cindyscript で描くものと、TeX に出力する KeTCindy で描くものを使い分けるとよい。

以下で、【Cinderella】は、Cinderella の作図ツールを用いた描画、【Cindyscript】は Cindyscript の関数、【KeTCindy】および【KeTCindy3D】は KeTCindy の関数を示す。


7.1 点の作図

7.1.1 点をとる

【Cinderella】	「点を加える」ツール  を用いる
【KeTCindy】	<code>Drwpt(点), Drawpoint(点), Pointdata(名前, リスト), Putpoint(名前, 座標),</code>
【KeTCindy3D】	<code>Drawpoint3d(座標), Putpoint3d(リスト),</code>

点をとる関数はいくつかある。Cinderella の描画面上に単に点を表示するもの、幾何点を作るもの、TeX に出力するためのもの、と用途が異なる。Putpoint(), Putpoint3d() は座標を与えて幾何点を作るが、作図ツールの「点を加える」で作図しておいてスクリプトで座標を指定してもよい。

【例】座標 (1,1) に点 A をとる。

(方法1) 磁石ツール  でスナップモードにしておき、作図ツールで点をとる。

(方法2) KeTCindy の関数で Putpoint("A", [1,1]) とする。

(方法1) では、この点は自由点となる。固定点にするには、インスペクタで「ピンで留める」にする。また、点名は Cinderella が自動的につけるので、変更するには、点を選択しておき、インスペクタを開いて「要素の情報」で変更する。

[1,1] のような格子点でない場合は、作図ツールで適当な位置に点をとったのち、Cindyscript で `A.xy=[2,sqrt(3)]` のようにして座標を指定する。

(方法2) では、この点は固定点となる。自由点とするには Putpoint("A", [1,1], [A.x, A.y]) とする必要がある。

これらの方法で点を取ったとき、画面上には点が表示されるが TeX の図には出力されない。TeX の図に出力するには、KeTCindy の Drwpt() または、Drawpoint() を用いる必要がある。

以下は、それぞれの関数についての比較表である。「描画」は緑色の点で描画するが幾何点は作らない。

関数	描画	幾何点	TeX 出力
Drawpoint	-	-	○
Pointdata	○	-	○
Putpoint	-	○	-
Drawpoint3d	○	-	-
Putpoint3d	-	○	-

[⇒ 関数一覧](#)

7.1.2 点の大きさを変える

【Cinderella】 インспекタの「表示方法」を使う

【Cindyscript】 `A.size=n`

【KeTCindy】 `Ptsize(n)`, `Setpt(n)`

インспекタ、Cindyscript による点の大きさの設定は画面上には反映されるが、TeX には書き出されない。逆に、KeTCindy で設定した点の大きさは、Cinderella の画面上には反映されない。そのことを考慮の上、適宜使い分ける必要がある。画面上の点の大きさと、TeX の図の点の大きさは連動しないので注意が必要である。

KeTCindy の描画関数を使って点をとるときは、`[”size=n”]` オプションをつけて大きさを設定することもできる。

KeTCindy では、`Drwpt()` または `Drawpoint()` に 0 オプションをつけることで白抜きの点を描くことができる。ただし、この場合、これよりあとに書かれた線は白で抜いたあとに書かれてしまうので、記述の順番を考慮する必要がある。

[⇒ 関数一覧](#)

7.1.3 曲線上に点をとる

【Cinderella】 「点を加える」ツールを用いる
 【KeTCindy】 PutonCurve(name,PD, 初期値)
 【KeTCindy3d】 PutonCurve3d(点名,PD)


Cinderella の作図ツールで描いた曲線上には、「点を加える」ツールで点をとることで、その点は曲線上だけを動かすことができるようになる（インシデント）。作図ツールで描いた曲線でなければこの方法では曲線上に点は取れない。

Cindyscript で描いた曲線の場合、たとえば、関数 $f(x)$ のグラフ上に点を置くには、 $A.y=[A.x,f(A.x)]$ のようにすればよい。（あらかじめ点を作図しておく）

KeTCindy の PutonCurve() を使うと、プロットデータを用いて曲線上に点をとることができる。また、KeTCindy で直線、線分上に取りたい場合は、PutonLine(), PutonSeg() を用いることもできる。

[⇒ 関数一覧](#)

7.1.4 交点をとる

【Cinderella】 「2つの曲線の交点を求める」ツール  を用いる
 【KeTCindy】 Intersectcrvs(PD,PD), Crosspoint(name,PD,PD, リスト),
 Putintersect(点名,PD,PD)
 【KeTCindy3D】 Intersectcrvsf(name,PD, 式), IntersectsgpL(点名, 線分, 面, 方法)

Cinderella の作図機能で作図した幾何要素の交点は Cinderella の「2つの曲線の交点を求める」ツールを用いるのが簡便であるが、関数のグラフや方程式による曲線の場合は、Cinderella の作図機能は使えないので、KeTCindy の関数を使うことになる。

KeTCindy の関数は3通りあるが、intersectcrvs() は2曲線の交点のリストを取得する関数なので、既存の点を交点に持てきたい場合は

```
tmp=Intersectcrvs(gr1, crAB);
P.xy=tmp_1;
```

のようにすればよい。


Crosspoint(), Putintersect() では交点に新たな幾何点を作ることができる。

Intersectcrvsf() は曲線と曲面の交点、 IntersectsgpL() は直線と平面の交点を求める。

[⇒ 関数一覧](#)

7.2 線の作図

7.2.1 線分を引く

【Cinderella】	「線分を加える」 ツール  を用いる
【Cindyscript】	<code>draw([A,B])</code>
【KeTCindy】	<code>Listplot([A,B])</code>
【KeTCindy3D】	<code>Spaceline([A,B])</code>

Cinderella の描画ツールを使うと、描かれた線分は幾何要素となり、インスペクタでその属性（色・太さなど）を変えることができる。また、点を線分上にインシデントさせることができる。Scilab に出力しない線分を描くときはこれで十分で、スクリプトも簡素になる。

Cindyscript の `draw` 関数と KeTCindy の `Listplot` 関数では、線の色、太さ、線種を指定できる。また、点の識別名ではなく座標を引数にすることもできる。

推奨するのは、補助線などは Cinderella の描画ツールで描き、最終的に必要な線だけ `Listplot()` で出力する方法。

[⇒ 関数一覧](#)

7.2.2 折れ線を描く

【Cindyscript】	<code>connect(点のリスト)</code>
【KeTCindy】	<code>Listplot(点のリスト)</code>
【KeTCindy3D】	<code>Spaceline(点のリスト)</code>

Cinderella の描画ツールで折れ線を描くのはインタラクティブではあるが本数が多くなると現実的ではない。ただ、描かれた線分は幾何要素となり、インスペクタでその属性（色・太さなど）を変えることができる。折れ線全体の属性を変えるには、すべての折れ線を同時に選択する。（Shift + クリック）

Cindyscript の `connect()` と KeTCindy の `Listplot()` , `Spaceline()` では、引数に点のリストを与えれば折れ線が描ける。

節点を明示した折れ線を描くには、`Listplot()` と `Drawpoint()` を組み合わせることが考えられるが、あらかじめ点のリストを作っておくのが簡便である。

```
Ptsize(3);
```



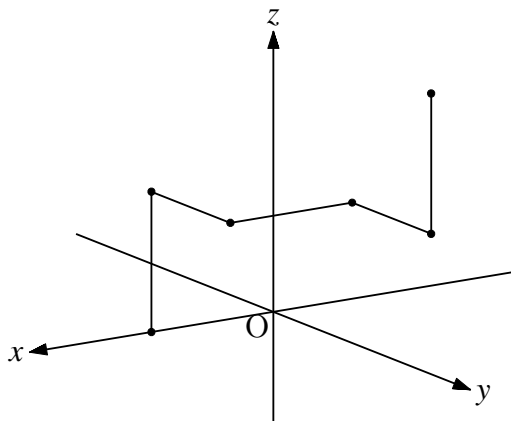
```
pd=[[1,2],[3,4],[6,2]];
Pointdata("1",pd);
Listplot("1",pd);
```

KeTCindy3D の場合は空間座標で点を TeX に書き出すことができないので、次のように点を作って書き出す。

```
Putpoint3d(["A",[2,0,0],"B",[2,0,2],"C",[2,2,2]],"fix");
Spaceline([A,B,C]);
Ptsize(3);
Drawpoint([A,B,C]);
```

節点が多い場合は、空間の点のリストと、投影した点のリストを作って描画する。

```
pt=[[2,0,0],[2,0,2],[2,2,2],[0,2,2],[0,4,2],[0,4,4]];
pt2d=apply(pt,[Projcoordpara(#)_1,Projcoordpara(#)_2]);
Spaceline("1",pt);
Ptsize(4);
Drawpoint(pt2d);
```




点の名前が必要であれば

```
pname=apply(1..6,"P"+text(#));
```

のようにして、名前リストを作る。

[⇒ 関数一覧](#)

7.2.3 直線を引く

【Cinderella】	「直線を加える」ツール  を用いる
【Cindyscript】	<code>draw(join([1,2],[2,-1]))</code>
【KeTCindy】	<code>Lineplot([A,B])</code>

Cinderella の描画ツールを使うと、描かれた線分は幾何要素となり、インスペクタでその属性（色・太さ・線種など）を変えることができる。また、点を直線上にインシデントさせることができる。Scilab に出力しない直線を描くときはこれで十分で、スクリプトも簡素になる。半直線にしたい場合は、線分を描いて一方の端点を描画領域（SW,NE）の外に出して見かけ上半直線にすればよい。

Cindyscript の `draw` 関数と KeTCindy の `Lineplot` 関数では、線の色、太さ、線種を指定することができる。点の識別名ではなく座標を引数にすることもできる。

TeX への出力は、`Lineplot()` に `"+"` オプションをつけると半直線になる。

KeTCindy3D には直線を描く関数はないが、線分の外分点を適当にとって線分として描けば、ほとんど直線として扱うことができる。

[⇒ 関数一覧](#)

7.2.4 点線や破線を描く

【Cinderella】	インスペクタの「特別な表示方法」で線種を変える
【Cindyscript】	修飾子 <code>dashtype</code> を使う
【KeTCindy】	<code>do ,da</code> オプションをつける
【KeTCindy3D】	<code>do ,da</code> オプションをつける

Cinderella の描画ツールで引いた線分・直線の線種は、その要素を選んで「Ctrl+i」(Mac は ⌘ +i) で表示されるインスペクタを用いて、「特別な表示方法」で変更することができる。点線、破線だけでなく、一点鎖線もある。

Cindyscript の `draw()` で引いた線分・半直線は、修飾子を用いて、`draw([A,B],dashtype->2)` のようにして描く。

KeTCindy では、オプションに、`["do"]` (点線: `dotline`) `["da"]` (破線: `dashline`) をつける。

[⇒ 関数一覧](#)

7.2.5 線の太さを変える

【Cinderella】 インспекタの「表示方法」を使う

【Cindyscript】 size オプションをつける

【KeTCindy】 Setpen(n)

インспекタ, Cindyscript による点の大きさの設定は画面上には反映されるが, TeX には書き出されない。逆に, KeTCindy で設定した点の大きさは, Cinderella の画面上には反映されない。そのことを考慮の上, 適宜使い分ける必要がある。画面上の点の大きさと, TeX の図の点の大きさは連動しないので注意が必要である。

KeTCindyn の描画関数を使うときは, "size=n" オプションをつけて大きさを設定できるものもある。

[⇒ 関数一覧](#)

7.2.6 垂線を引く

【Cinderella】 「垂線を加える」ツール  を用いる

【Cindyscript】 perpendicular(A,a)

【KeTCindy】 作図後, Listplot() で線分を引く

【KeTCindy3D】 Perppt() で垂線の足を求めて線分を引く


Cinderella の「垂線を描く」ツールを用いて作図するのが簡便だが, 何か条件を与えて垂線を引くような場合は Cindyscript を使うことになる。ただし, 点と直線 (線分) が幾何要素になっていないと垂線は引けない。perpendicular(A,a) で, A は点の識別子, a は直線 (線分) の識別子。TeX には, 適当な図が描けたところで Lineplot() もしくは Listplot() で出力すればよい。

対象が幾何要素でない場合は, Cindyscript で方程式を求めて引くことになる。

KeTCindy3D の Perppt() 関数は, ある点から平面に下した垂線の足を求める。Perppt() の項にある例を参照のこと。

[⇒ 関数一覧](#)

7.2.7 平行線を引く

【Cinderella】 「平行線を加える」ツール  を用いる

【Cindyscript】 parallel(B,a)

【KeTCindy】 作図後, Listplot() で線分を引く


Cinderella の「平行線を描く」ツールを用いて作図するのが簡便だが, なにか条件を与えて平行線を引くような場合は Cindyscript を使うことになる。ただし, 点と直線 (線分) が幾何要素になっていないと平行線は引けない。parallel(B,a) で, B は点の識別子, a は直線 (線分) の識別子。

TeX には, 適当な図が描けたところで Lineplot() もしくは Listplot() で出力すればよい。対象が幾何要素でない場合は, Cindyscript で方程式を求めて引くことになる。

[⇒ 関数一覧](#)

7.3 多角形を描く

7.3.1 頂点を与えて多角形を描く

【Cinderella】 多角形ツール  を用いる

【Cindyscript】 drawpoly(点リスト)

【KeTCindy】 Listplot(点リスト)

【KeTCindy3D】 Spaceline(点リスト)

Cinderella の描画ツールの多角形ツールは, 既存の点を結んで多角形を作り色塗りをするもの。色塗りをしないのであれば, 単に線分でつないでいく。

Cindyscript の drawpoly(), および KeTCindy の Listplot(), Spaceline() での点リストは, 既存の幾何点でなく座標でもよい。Listplot(), Spaceline() の点リストが必ず閉じたりスト ([A,B,C,D,A]) でなければならないのに対し, Cindyscript の drawpoly() では閉じている必要はない。drawpoly([A,B,C,D]) でよい。

[⇒ 関数一覧](#)

7.3.2 1 辺を与えて正多角形を描く

平面上で, 与えられた線分 AB を 1 辺とする正多角形を描く。

線分 AB は, Cinderella の作図ツールなどで描かれているものとする。ただし, 線分でなく, 両端の点を与えられているだけでもよい。点 A,B が複素平面上にあるものとして, 多角形の頂点の位置を計算すればよい。

Cindyscript の Draw スロットに, 次のスクリプトを書く。(n=5 で正五角形を描く)

```

n=5;
pti=[complex(A),complex(B)];
th=2*pi/n;
repeat(n-2,s,
  z1=pti_s;
  z2=pti_(s+1);
  z=z2+(z2-z1)*(cos(th)+i*sin(th));
  pti=append(pti,z);
);
pt=apply(pti,gauss(#));
pt=append(pt,A.xy);
Listplot("1",pt);


```

pti は、各頂点に対応する複素数のリスト、pt が各頂点の座標のリストである。

[⇒ 関数一覧](#)

7.3.3 接線を引く

曲線上の点における接線を引く

【Cinderella】	幾何要素の 2 次曲線の場合、「点の極線を加える」ツール 	を用いる
【Cindyscript】	微分して接線の方程式を求めて直線を引く	
【KeTCindy】	同上	

Cinderella の作図ツールで描いた 2 次曲線の場合は、曲線上の点において「点の極線を加える」ツールで接線が引ける。このツールは、初期状態では画面に出ていないので、モードメニューの「直線」から選ぶ。また、「設定」メニューの「上のツールバーをカスタマイズする」で、すべてのツールを表示すれば画面上に出る。

Cinderella の作図ツールで描けない曲線の場合は方程式を求めて直線を引くことになる。微分をする関数は、Cindyscript では d(関数, 変数), KeTCindy では Derivative(関数, 変数, 値) を用いる。

例：3 次関数 $f(x) = x^3 - 4x$ 上の点 (1, -3) における接線を引く Cindyscript と KeTCindy の組み合わせ

```
f(x):=x^3-4*x;
```

```

g(x0,x):=d(f(#),x0)*(x-x0)+f(x0);
p1=[1,-3];
p2=[2,g(1,2)];
plotdata("1","x^3-4*x","x");
Lineplot("1",[p1,p2]);

```

このスクリプトでは、 $g(x_0, x)$ を点 $(x_0, f(x_0))$ における接線の方程式としている。
また、接点をインタラクティブに決めるには、2点 A,B を作図しておく

```

Deffun("f(x)","regional(y)","y=x^3-4*x","y");
coef=Derivative("f(x)","x",A.x);
A.y=f(A.x);
B.y=coef*(B.x-A.x)+A.y;
Plotdata("1","f(x)","x");
Lineplot([A,B]);

```

として、接点 A をドラッグして位置決めをすることができる。

<注>

```

Deffun("f(x)","regional(y)","y=x^3-4*x","y");

```

は

```

f(x):=x^3-4*x;


```

でもよい。


[⇒ 関数一覧](#)

7.4 円の作図

7.4.1 円を描く

【Cinderella】	「円を加える」ツール  を用いる
【Cindyscript】	<code>drawcircle([a,b],r)</code>
【KeTCindy】	<code>Circledata([A,B], Circledata(name,[[a,b],[c,d]])</code>

Cinderella の円を描くツールは3種類あるが、あとで TeX に書き出すならば「円を加える」ツールがよい。中心からドラッグすると円周上に点ができる。この2点を使って

Circledata([A,B]) で書き出すことができる。TeX に書き出す必要がない補助円などであれば、「半径付き円を加える」  が簡便。半径を一定にする（円周ドラッグで大きさが変わらないようにする）のであれば、「固定した半径の円を描く」ツールを用いる。Cinderella の円を描くツールで描いた円は、インスペクタの「特別な表示方法」で線種を設定できる。

Cindyscript では、中心と半径を指定して円を描く。drawcircle([a,b],r) は [a,b] が中心の座標、r が半径（実数）。中心は、Cinderella の作図ツールでとった点 A を中心とする場合は、drawcircle(A,r)。

KeTCindy の Circledata([A,B]) は、中心 A で点 B を通る円を描く。A,B を座標にする場合は、name が必要。中心 A で半径 r の円を描くのであれば、座標指定になるので、Circledata("1",[A,A+[3,0]]) または Circledata("1",[A.xy,A.xy+[3,0]]) とする。

[⇒ 関数一覧](#)

7.5 領域を塗る


7.5.1 円の内部を塗る

【Cinderella】	「円を加える」ツールを用いたのち、インスペクタで設定
【Cindyscript】	fillcircle(A,r)
【KeTCindy】	Shade(PD)

画面上で色を塗るだけならインスペクタの「表示方法」で色塗りをするのが簡便。透明度も設定できる。TeX の図版でも領域を塗るのであれば、Circledata() で円を描いた（プロットデータを作成）のち、Shade() で色を塗る。ここで色指定をすれば画面にも反映される。

[⇒ 関数一覧](#)

7.5.2 多角形の内部を塗る

【Cinderella】	「多角形を加える」ツール  を用いる
【Cindyscript】	fillpoly(点リスト)
【KeTCindy】	Shade(PD)

Cinderella の描画ツールの多角形ツールは、既存の点を結んで多角形を作り色塗りをする。

Cindyscript での点リストは、既存の幾何点でなく座標でもよい。点リストは、始点と終点を結ぶので閉じている必要はない。([A,B,C,D])

TeX に書き出すのであれば Shade() を用いる。

[⇒ 関数一覧](#)

7.5.3 曲線で囲まれた領域を塗る

【KeTCindy】 Shade(PD)

Shade() を用いて領域を塗る。Cinderella の画面にも反映される。

[⇒ 関数一覧](#)

7.5.4 領域を斜線で示す

【KeTCindy】 Hatchdata(name, 方向,PD)

領域を斜線で塗る。Cinderella の画面にも反映される。Scilab とデータをやり取りしてハッチデータを作るので、バッチファイル（シェルフファイル）が裏で実行される。

[⇒ 関数一覧](#)

7.5.5 領域を細かい点で埋める

【KeTCindy】 Dotfilldata(name, 方向,PD)

曲線で囲まれた領域を斜線で塗るかわりに細かい点で埋める。使い方は Hatchdata() と同様。

[⇒ 関数一覧](#)

7.6 関数のグラフ・曲線

7.6.1 関数のグラフを描く

【Cindyscript】	関数を定義して、 <code>plot()</code> で表示する
【KeTCindy】	(関数を定義して) <code>Plotdata()</code> で表示する

Cindyscript では、 $f(x):=式$ の書式で関数を定義し、`plot(fx)` でグラフを描くことができる。関数は、`if` 関数による場合分けも可能。また、この関数概念は数学関数ではなく、プログラミングにおける関数なので、一連の操作を関数として定義して実行することもできる。

KeTCindy では、`Deffun()` で関数を定義して、`Plotdata()` でグラフを描く。`Deffun` では、`if` 文による条件分岐も可能だが、`if` のネストはできない。

三角関数などの組み込み関数については、定義は不要で、そのまま `plot()` もしくは `Plotdata()` でグラフを描くことができる。ただし、Cindyscript にあっても KeTCindy にはない逆三角関数 (\arctan など) は Cindyscript で画面には表示できるが TeX にはできない。

[⇒ 関数一覧](#)

7.6.2 媒介変数表示の曲線を描く

【Cindyscript】	<code>plot([f(t),g(t)])</code>
【KeTCindy】	<code>Paramplot(name, 式, 定義域)</code>
【KeTCindy3D】	<code>Spacecurve(name, 式, 定義域)</code>

いずれも option で線種などの指定ができる。

平面における曲線で、Timer Tick スロットに書いてアニメーションを行う場合は、Cindyscript の `plot()` の方を勧める。シンプルな分、実行速度において有利である。

[⇒ 関数一覧](#)

7.6.3 2次曲線を描く

放物線，楕円，双曲線を描く。

【Cinderella】	焦点などを作図してそれぞれの作図ツールを用いる
【Cindyscript】	関数もしくは媒介変数で表して <code>plot()</code> で描く
【KeTCindy】	<code>Plotdata()</code> もしくは <code>Paramplot()</code> , <code>Parabolaplot()</code> など描く

これらの曲線を描く方法は何通りかある。全体ではなく一部分を描く場合は Cinderella の作図ツールではできないので `plot()` または `Paramplot()` で描く。

これらの曲線上に点を取り、接線を引いてインタラクティブに動かすような場合は、Cinderella の作図ツールで描くとよい。点をインシデントにするのが容易であり、接線も簡単に引け、さらにその接線も幾何要素となるので都合がよい。

KeTCindy では、`Plotdata()` で媒介変数表示の式を与えるか、`Parabolaplot()`, `Ellipsplot()`, `Hyperbolaplot()` で焦点などを与えて描く。

[⇒ 関数一覧](#)

7.6.4 陰関数のグラフを描く

【KeTCindy】 `Implicitplot()`

陰関数で与えられた曲線を描く方法は Cinderella, Cindyscript にはない。

[⇒ 関数一覧](#)

7.6.5 プロットデータを自作して表示する

組み込み関数にない関数などを使ってプロットデータを自作して表示するには、Cindyscript で作成したプロットデータを `Listplot()` または `AddGraph()` に渡して描画する。それぞれの項の例を参照のこと。

[⇒ 関数一覧](#)

7.7 その他

7.7.1 幾何要素を非表示にする

描いた点や線を非表示にする。

【Cinderella】 インспекタの「表示方法」で「表示する」のチェックを切る

【Cindyscript】 幾何要素の visible プロパティを false にする

【KeTCindy】 "notex" または "nodisp" オプションをつける

インスペクタの表示方法」で「表示する」のチェックを切った場合、「表示」に戻すには、「表示」メニューの「式による表示」を開いて、該当の幾何要素を選択し、インスペクタで「表示する」のチェックをつける。


幾何要素を Cindyscript で非表示/表示にするには、`A.visible=false / true` とする。幾何要素ではない関数のグラフなどはインスペクタでは非表示にできない。

KeTCindy の "notex" オプションは TeX に出力しない。"nodisp" オプションでは、画面にも表示されない。プロットデータを作成するために使う。

[⇒ 関数一覧](#)

7.7.2 角に印をつける

角に弧または平行四辺形の角の印をつける。

【Cinderella】 「角に印をつける」ツール  を用いる（弧の形状のみ）

【Cindyscript】 なし

【KeTCindy】 `Anglemark()` , `Paramark()` を用いる

TeX に出力しないのであれば、「角に印をつける」ツールを用いるのが簡便。ただし、平行四辺形の印はつけられない。

角に向きを示す矢印を付加する場合は、`Anglemark()` で描いておき、`Arrowhead()` で矢じりをつける。

[⇒ 関数一覧](#)

7.8 値の取得・変換

7.8.1 角度を測る

【Cinderella】 「角度を測る」ツール  を用いる。

【Cindyscript】 `arctan2(ベクトル)` を用いる

Cinderella の「角度を測る」ツールでは、画面上に角度が表示される。この値をスクリプトで利用するためには、その識別名を用いる。識別名は、表示メニューから「式による表示」で出る幾何要素の一覧表で判断する。識別名が $\alpha 0$ であれば、変数 $\alpha 0$ に角の値が入っていると考えてよい。 α は日本語入力モードで「あるふぁ」で入力する。たとえば、`drawtext([1,1], $\alpha 0$)` とすれば、(1,1) の位置に角度が表示される。このときの角は一般角であり、点をドラッグすると 360° 以上の角や負の角で表示される。また、表示は度数法で行われるが、処理によっては弧度法での表示となる。

Cindyscript で角度を取得するには、`arctan2()` を用いるのがよい。引数はベクトルで、x 軸の正の方向とのなす角を $-\pi \sim \pi$ の範囲で弧度法で返す。角 ABC の場合、`th=arctan2(C.xy-B.xy)-arctan2(A.xy-B.xy)` で取得できる。

[⇒ 関数一覧](#)

7.8.2 2点間の距離を測る

【Cindyscript】 `dist(A,B)` または `|A,B|`

【KeTCindy3D】`dist3d(点, 点)`

[⇒ 関数一覧](#)

7.8.3 数値を文字に変換する

関数の引数に文字列として与えるために数値を文字列に変換したい場合がある。このような場合は、`text()` 関数を用いる。引数は実数・複素数いずれでも可。

subsubsection 小数点の桁数を指定して表示

【Cinderella】 なし

【Cindyscript】 `format()` 関数を用いる

【KeTCindy】 `Textformat()` 関数を用いる

Cinderella では、小数点以下の桁数は第 5 位を四捨五入して表示するのがデフォルトである。小数点以下の桁数を指定するには、Cindyscript では `format()` 関数、KeTCindyd では

Textformat() 関数を用いる。Textformat() 関数は、リストで与えた数でも処理できる。

[⇒ 関数一覧](#)

7.9 スライドを作る

7.9.1 線分スライドを作る

変数の値をインタラクティブに決めるスライドを作る。

まず、Cinderella の作図ツールで、適当な線分（または直線）と、その上の点をとる。このとき、点が線分上にインシデントになるように、点を線分上にドラッグして線分がハイライトしたときにマウスボタンを離す。

線分を AB、AB 上の点を C として、A と C の距離 $|A,C|$ を用いて変数の値を決めればよい。

例： $x=|A,C|-5$;

整数を取得したい場合は、天井関数 `ceil()` または、床関数 `floor()` を用いて整数化する。

例： $n=\text{ceil}(|A,C|)$

点 C を端点までドラッグした後、さらに点 C をドラッグしようとするすると端点も同時に動いてしまうことがある。そこで、端点を選択してインスペクタで「ピンで留める」にチェックを入れておくとよい。

[⇒ 関数一覧](#)

7.9.2 円形スライドを作る

角の値をインタラクティブに決めるスライドを作る。

まず、Cinderella の作図ツールで、適当な円を描き、その上に点をとる。このとき、点が円周上にインシデントになるように、点を円周上にドラッグして円周がハイライトしたときにマウスボタンを離す。

円の中心を A、円周上の点を B としたとき、`B.angle` で角度を取得できる。ただし、戻り値は $0^\circ \sim 360^\circ$

`arctan2(B.xy-A.xy)` で角を取得することもできる。この場合の戻り値は、 $-180^\circ \sim 180^\circ$

[⇒ 関数一覧](#)

8 関数一覧

【設定・定義】

Addax(0/1)	座標軸を描くかどうかを定める
Addcolor(com,color)	描画色を設定する
Colorcode(文字 1, 文字 2,color)	カラーコードの変換
Deffun(関数名, 定義リスト)	関数を定義する
Definecolor(色名, 定義リスト)	ユーザー定義色の設定
Defvar(文字列)	変数を定義する
Drwxy()	座標軸を先に描く
FontSize(記号)	フォントサイズを設定する
Ketinit(options)	KE _T Cindy を初期化する
Ptsize(数)	表示する点の大きさを設定する
Setax(リスト)	座標軸の書式を設定する
Setcolor(color,options)	Windisp _g での描画色を設定する
Setmarklen(数)	軸の目盛の長さを設定する
Setorigin(座標)	表示する座標軸の原点の位置を設定する
Setpen(数)	線の太さを設定する
Setpt(数)	表示する点の大きさを設定する
Setscaling(数)	縦方向の倍率を設定する
Setunitlen(数)	単位長を設定する
Setwindow()	描画領域を設定する

【描画】

Anglemark(点リスト, options)	角の印を入れる
AddGraph(name, プロットデータ)	ユーザー定義のプロットデータを描画する
Arrowdata(始点, 終点,options)	2 点間を結ぶ矢線を描く
Arrowhead(点, 方向,options)	点に矢じりだけを描く
Bezier(name, リスト, リスト,options)	単独のベジェ曲線を描く
Beziersmooth(name, リスト,options)	なめらかなベジェ曲線を描く。その 1
Beziersym(name, リスト,options)	なめらかなベジェ曲線を描く。その 2
Bowdata(点リスト,options)	弓形を描く
Bspline(name, リスト, options)	2 次 B スプライン曲線を描く
Changestyle(PD リスト, options)	描画オプションを変更する
Circledata(name, 点リスト,options)	円または多角形を描く
Crosspoint(name,PD,PD, 範囲リスト)	2 曲線の交点を作る

CRspline(name, リスト, options)	単独の Catmull-Rom スプライン曲線を描く
Deqplot(name, 式, 変数名, 初期値,options)]	微分方程式の解曲線を描く
Dotfilldata(name , 方向, PD , optionsss)	領域に点を敷き詰める
Drawsegmark(name, リスト,options)	線分に印をつける
Segmark(name, リスト,options)	線分に印をつける
Drawpoint([点,options])	点を表示する
Drwpt([点,options])	点を表示する
Ellipseplot(name,list,str,options)	楕円を描く
Enclosing(name , [位置, 方向, 数式])	複数の曲線から閉曲線を描く
Expr(文字列)	T _E X 数式を書く
Exprrot(位置, 向き, 文字列)	傾いた T _E X 数式を書く
Framedata(name , リスト)	矩形を描く
Framedata2(name , リスト)	矩形を描く
Hatchdata(name , 方向, PD , optionsss)	領域に斜線を引く
Htickmark([横座標 , 方向 , 文字])	横軸に目盛りを描く
Hyperbolaplot(name,list,str,options)	双曲線を描く
Implicitplot(name,str,,str,str,options)	陰関数のグラフを描く
Joincrvs (name, PD リスト, options)	2つのプロットデータをつなげたデータを作る
Letter(「座標, 位置, 文字列」のリスト)	文字列を表示する
Letterrot(「座標, 方向, 移動量, 文字列」)	文字列を回転して表示する
Lineplot(name, 2点のリスト,options)	2点を結ぶ直線を描く
Listplot(name, 点のリスト,options)	点を線分で結ぶ
Mkbeziercrv(name, リスト,options)	作図した点を使ってベジエ曲線を描く
Mkbezierptcrv(リスト, options)	制御点を自動配置してベジエ曲線を描く
Mkcircle()	幾何円のすべての PD を作成する
Mksegments()	幾何線分のすべての PD を作成する
Ovaldata(name, 点リスト,optionsss)	角を丸くした矩形を描く
Parabolaplot(name,list,str,options)	放物線を描く
Parabolaplot(name, 点リスト,options)	楕円を描く
Paramark(点リスト,options)	角の印を入れる
Paramplot(name, 式, 変数と定義域,options)	媒介変数で表された曲線を表示する
Partcrv(name, 点 1, 点 2,PD)	部分曲線を描く
Plotdata(name, 式, 変数と定義域,options)	関数のグラフを描く
Pointdata(name, 点リスト,optionsss)	点データを作る
Polygonplot(name, 点リスト, 整数,options)	正多角形を描く
Putintersect(点名,PD1,PD2)	2曲線の交点を作る
PutonLine(点名, 座標 1, 座標 2)	直線上に点を作る

PutonCurve(name,PD, 初期値)	曲線上に点を作る
Putpoint(点名, 座標 1, 座標 2)	点を作る
PutonSeg(点名, 座標 1, 座標 2)	線分上に点を作る
Reflectdata(name,PD, 点リスト,options)	プロットデータの鏡映を作成
Reflectpoint(点, 対称点/対称軸)	点の鏡映を作成
Rotatedata(name,PD, 角度, 中心,options)	プロットデータを回転する
Rotatepoint(点, 角度, 中心)	点の位置を回転する
Rulerscale(点, リスト, リスト)	目盛を打つ
Scaledata(name,PD,x,y, 中心,options)	点を拡大・縮小する
Scalepoint(点, 比率ベクトル, 中心)	点の位置を拡大・縮小する
Shade (PD リスト , 数)	閉曲線の内部にシェードをかける
Translatedata(name,PD, ベクトル,options)	プロットデータを平行移動する
Translatepoint(点, ベクトル)	点を平行移動する
Vtickmark([横座標 , 方向 , 文字])	縦軸に目盛りを描く

【微積分など】

Derivative(関数式, 変数, 値)	関数の微分係数を求める
Integrate(関数式, 変数, 範囲,options)	関数の定積分値を求める
Inversefun(関数式, 範囲, 値)	逆関数値を求める

【作表】

Tabledata("", 縦横 , 除外 , options)	表の枠を作成する
Tabledatalight("", 縦横 , 除外 , options)	幾何点を持たない表の枠を作成する
ChangeTablestyle(罫線リスト, options)	Table の罫線の描画オプションを変更する。
Findcell(列番号, 行番号)	セルの情報 list を返す
Putcell (列番号, 行番号, 位置, 文字)	セルに文字列を入れる
PutcoL (列番号, 位置, 文字列リスト)	1 列に順に文字を書き入れる
PutcoLexpr (列番号, 位置, 文字列リスト)	1 列に順に $\text{T}_{\text{E}}\text{X}$ 書式の文字を書き入れる
Putrow (行番号, 位置, 文字列リスト)	1 行に順に文字を書き入れる
Putrowexpr (行番号, 位置, 文字列リスト)	1 行に順に $\text{T}_{\text{E}}\text{X}$ 書式の文字を書き入れる
Settable((左上) , 右下)	表データを書き出す
Tgird(セルラベル	セル (格子点) の座標を返す
Tlistplot(セルラベル 1, セルラベル 2)	セルに斜線を引く

【値の取得と入出力】

Crossprod(リスト, リスト)	ベクトルの外積を計算する
Dotoprod(リスト, リスト)	ベクトルの内積を計算する

Findarea(PD)	プロットデータで囲まれる部分の面積を求める
Findlength(PD)	プロットデータで描く曲線の長さを求める
Intersectcrvs(PD1,PD2,options)	プロットデータを交点のリストを返す
Invert(PD)	プロットデータの点を逆順にする (reverse と同じ)
MeetCurve(PD,x,y)	曲線上の点を取得する
Lcrd()	幾何点, リスト点の論理座標を取得する
Pcrd()	幾何点, リスト点の物理 (表示) 座標を取得する
Nearestpt(PD,PD)	2 曲線間の最も近い点を取得する
Nearestptcrv(点,PD)	点に一番近い曲線上の点を取得する
Numptcrv(PD)	曲線 PD の節点データの個数を取得する
ParamonCurve(PD,n,PtL)	PD 上にある点 P のデータを取得する
Pointoncrv(数,PD)	パラメータ値をもつプロットデータ上の点
Ptcrv(n,PD)	曲線 PD の n 番目の節点を取得する
Ptstart(PD)	プロットデータの始点を取得する
Ptend(PD)	プロットデータの終点を取得する
ReadOutData(ファイル名)	外部データを読み込む
Sprintf(実数, 長さ)	小数点以下の長さを固定した文字列に変換
WritetoSci(ファイル名,options)	すべてのソースを Scilab 用書き出す
WritetoScibody(ファイル名)	ボディのソースを Scilab 用書き出す
Makeshell(ファイル名)	Mac のシェルスクリプトを書き出す
Makebat(ファイル名)	Windows のバッチファイルを書き出す
Textformat(数, 桁数)	小数点以下の桁数を指定して数値を文字列化する
Viewtex()	T _E X のソースファイルを書き出す。引数なし
Workprocess()	作図の経過を取得する

【その他】

Assign(文字列)	文字列中のある文字を値で置き換える
BBdata(ファイル名)	画像のサイズを求める
Changework(パス)	作業ディレクトリを変更する
Com0th(コマンド)	Scilab の Openfile の前に置くコマンドを定義する
Com1st(コマンド)	Scilab の Openfile の前に置くコマンドを定義する
Com2nd(コマンド)	Scilab の Openfile のあとに置くコマンドを定義する
Com2ndpre(コマンド)	Scilab の Openfile のあとに置くコマンドを定義する
Dashline(PD 名)	プロットデータを破線で描く
Dottedline(PD 名)	プロットデータを点線で描く
Drwline(PD 名)	プロットデータを実線で描く
Figpdf(option)	出力枠サイズの PDF を作る

Helplist()	ヘルプデータを作る
Help(str)	コマンドヘルプを表示する
Helpkey(str)	キーワードで関数を検索する
Ketcindylogo()	K _E T _C indy のロゴを書き出す
Indexofall(str1,str2)	文字列 str1 から str2 を検索しその位置をすべて返す
Op(n,list)	リストまたは文字列から要素を抜き出す
Texcom(コード)	T _E X のコードを書き出す
Windispg()	定義されたプロットデータを描画面に描く

【R との連携】

Boxplot(名前, データ, 位置, 高さ,option)	箱ひげ図を描く
CalcbyR(変数名, コマンド列, option)	R のコマンドを実行して結果を返す
Histplot(name,data)	ヒストグラムを描く
PlotdataR(name, 式, 変数)	R の関数のグラフを描く
PlotdiscR(name, 式, 変数)	離散型のグラフを描く
Readcsv(name,filename,option)	csv ファイルを読む
Scatterplot(name,filename,option)	2次元データを読み込み, 散布図を描く

【Maxima との連携】

CalcbyM(name, リスト,option)	Maxima のスクリプトを実行する
Example("Mxfun", 文字)	Mxfun の使用例を表示
Mxbatch(リスト)	Maxima の外部スクリプト用コマンドを作る
Mxfun(name, 式, リスト,option)	Maxima の関数を実行する
Mxtex(num, 式)	式を TeX 書式にする

【Risa/Asir との連携】

CalcbyA(name, リスト,option)	Risa/Asir のスクリプトを実行する
Asirfun(name, 式, リスト,option)	Risa/Asir の関数を実行する

【FriCAS との連携】

CalcbyF(name, リスト,option)	FriCAS のスクリプトを実行する
Frfun(name, 式, リスト,option)	FriCAS の関数を実行する

【MeshLab との連携】

Mkobjcmd(name, 式,option)	厚みを持たない曲面のコマンドを作成
Mkobjcvcmd(name,PD,option)	空間曲線のコマンドを作成
Mkobjnrm(name, 式)	法線ベクトルのデータを作成

Mkobjplatecmd(name, 面データ, options)	面を描く
Mkobjpolycmd(name, PD, options)	多面体を描く
Mkobjsymbcmd(PD, 実数, 実数, vec, vec)	文字等のコマンドを作成
Mkobjthickcmd(name, 式)	厚みを持つ曲面のコマンドを作成
Mkviewobj(name, PD, options)	obj ファイルを作成

【表計算ソフトとの連携】

Tab2list(str, option)	str の内容をリストに変換する
Dispmat(lsit)	リストの内容を行列型にコンソールに表示する

【KeTCindy3D 設定・定義】

Ketinit3d()	KeTCindy3D の使用宣言
Start3d()	3D の開始

【KeTCindy3D 描画】

Bezier3d(name, リスト, リスト)	空間ベジェ曲線を描く
Changstyle3d(リスト, リスト)	3d プロットデータの属性を変更
Concatob(リスト, option)	いくつかの obj データを結合
Crvsfparadata(name, PD, PD2, 式, opt, opt)	曲線の曲面による陰線処理
Datalist2d()	画面に描かれているすべてのプロットデータ
Datalist3d()	画面に描かれているすべてのプロットデータ
Dist3d(点名, 点名)	空間の2点の距離
Drawpoint3d(座標)	空間点を描く
Embed(name, PD, 式)	埋め込みデータ作成
Intersectcrvsf(name, PD, 式)	曲線と曲面の交点を求める
IntersectsgpL(点名, 線分, 面, 描画方法)	空間の直線と平面の交点
Invparapt(座標, PD)	描画面座標に対応する曲線上の座標
Mkbezierptcrv3d(点リスト)	制御点を自動的にとる空間ベジェ曲線
Nohiddenbyfaces(name, PD, PD, opt1, opt2)	多面体と空間曲線を陰線処理
Parapt(座標)	点の投影面での座標
Partcrv3d(name, 始点, 終点, PD)	曲線 PD の部分曲線を作る
Perpplane(点名, 点, ベクトル, option)	点を通り垂直な平面上の基準点
Perppt(点名, 点, 点リスト, option)	平面に下ろした垂線の足
Phparadata(name, name2, options)	多面体を陰線処理して描く
Projcoordpara()	投影座標を求める
Putaxes3d([x,y,z])	軸上に幾何点をとる
PutonCurve3d(点名, PD)	空間曲線上に点をとる
Putonseg3d(点名, 点1, 点2)	線分上に点をとる

Putpoint3d(リスト,option)	空間点をとる
Readobj(ファイル名)	obj ファイルを読み込む
Reflectpoint3d(点, リスト)	点を鏡映
Rotate3pt(点,vec, 角度, 点)	空間点を回転
Rotatedatat3d(name,PD,vec, 角度, 点)	プロットデータを回転
Sf3data(name, リスト,options)	陰線処理なしの空間曲面を描く
Sfbdparadata(name, 式,options)	曲面を陰線処理して描く
Skeletondatacindy(name,PD,PD,options)	スケルトン処理のデータ作成
Skeletonparadata(name,PD,PD,options)	スケルトン処理のデータ作成
Spacecurve(name, 式, 定義域,options)	空間曲線のデータ作成
Spaceline(リスト)	空間の折線データ作成
Translatedata3d(name,PD, 平行移動量)	空間プロットデータを平行移動
Translatept3d(座標, 平行移動量)	空間点を平行移動
VertexEdgeFace(面データ,option)	頂点と面から辺を求め, 辺を描く
Wireparadata(name,PD, 式,int,int,opt,opt)	曲面のワイヤフレームを陰線処理
Xyzax3data(name, 文字, 文字, 文字,options)	座標軸の表示
Xyzcoord(P.x,P.y,Pz.y)	主副画面で決まる点の座標

【KeTCindymv】

Moviedata(str1,str2,options)	動画データの作成
Setpata(str)	パラパラ動画のタイトル指定