Tiny Basic for Windows グラフィック操作編

tbasic.org [*1]

Ver.1.64 用 [2026 年 10 月版]

Tiny Basic for Windows 入門編，初級編，ファイル操作編では，プログラムを書く上で，文字を使った基本的な処理について説明しました。ここではグラフや絵を描くグラフィック操作について説明します。

本文書に掲載されている主要なプログラムは，同梱の

```
samples¥tutorial¥Graphic
```

に収められています。これらのプログラムを実行しながらの本書をご利用をお勧めします。

目次

1 tbasic グラフィック処理の概要 2
- 1.1 グラフィックの原理 . . . 2
- 1.2 tbasic のグラフィック画面 . . . 3
- 1.3 点の位置指定 . . . 4
- 1.4 グラフィック画面への描画 . . . 5
- 1.5 グラフィック画面の閉じ方 . . . 6
- 1.6 グラフィック画面の活用 . . . 7
- 1.7 高解像度画像の利用 . . . 9

2 tbasic グラフィック処理法 11
- 2.1 色の指定 . . . 11
- 2.2 初期設定 . . . 13
- 2.3 グラフィック画面への描画 . . . 13
- 2.4 グラフィック画面の閉じ方 . . . 18

3 グラフィック基本コマンド・文の使い方 18
- 3.1 Cls . . . 19
- 3.2 PSet . . . 20
- 3.3 Line . . . 21
- 3.4 Circle . . . 23
- 3.5 Paint . . . 24
- 3.6 GLocate と GPrint . . . 25

4 活用コマンド 27
- 4.1 LoadPicture . . . 27

[*1] https://www.tbasic.org

4.2 SavePicture 33
4.3 ReSizeImage 34
4.4 RotateFlip 35

5 グラフィック処理プログラム例（平面グラフの描画） 37
5.1 初期設定 37
5.2 グラフを点で描く 38
5.3 グラフを線で描く 39
5.4 比較的汎用性のあるプログラム 42
5.5 媒介変数や極座標表示関数のグラフ 43
5.6 演習問題 49

今まで tbasic で色々な計算や処理ができることを説明しました。そこでは，その計算や処理の結果を，数値や文字として実行画面に表示したり，テキストファイルに出力したりしました。しかし計算や処理した結果を文字としてだけでなく，グラフにしてそれを出力できれば，その結果をよく理解する助けになりますし，楽しいものです。

tbasic では文字を画面に表示したり，ファイルに出力するだけではなく，色々なグラフや絵を画面に表示したり，画像ファイルとして出力することができます（これをグラフィック操作と言います)。そして，ちょっとしたことを，手軽に行なえる十分な能力を持っています。

ここでは tbasic を使ったグラフィック操作の基本を説明します。

1　tbasic グラフィック処理の概要

tbasic のグラフィックは N88-BASIC（或いは，Quick BASIC）を基本にして Windows 用に変更・拡張したものです。Visual Basic のグラフィックは Quick BASIC の後継とも言えますが，かなり大きな変更・拡張が行われていて，tbasic のグラフィックとは少し違います。しかし，ルーツは同じですので，移行は難しくありません。

1.1　グラフィックの原理

コンピュータに接続されているディスプレイは小さな点 (pixel：画素) が縦横に規則正しく並んだ集まりです。その各点を種々の色で表すことにより，グラフィックを表現します。私たちがコンピュータの画面上に，色々な曲線や，また立体的な図形が描かれているのを見ますが，実際は，平面上に色々な点を描くことで，そのように見えるようにしています。ですから，「すべてのものを点で描く」ことになります。

線や絵などを点でいちいち描いていてはとても大変です。でも心配は要りません。線や円を描く便利な命令が用意されています。そして，tbasic は命令に従ってそれを点に変換してグラフを描きます。でも実際に最終的には，

すべてのものを点で描く

わけです。

どのくらいの個数の点で描くのかを決めるのが解像度です。ディスプレイの解像度が 800×600 といった言い方がありますが，これは横が 800 ピクセル，縦が 600 ピクセルからなる画面であることを意味します。現在の Windows で使われるディスプレイは，800×600，1024×768，1152×864，1280×960，1280×1024，1600×1200 等の解像度があります[*2]。どの解像度が使えるかは，ディスプレイとコンピュータに装備されているビデオカードの性能（メモリ）に依存します。

Windows では，ディスプレイのある部分にフォームと言われる小さなウインドウ画面を配置し，その中のグラフィック画面にグラフを描画します。

実際の画面上にある画素の位置は左上からの座標で表現されますが，ユーザーが操作する場合，フォームの中に配置されたグラフィック画面での左上からの座標で操作します。

[*2] また最近では，4K(3840×2160) の解像度を備えるものもあります。

1.2 tbasic のグラフィック画面

今までは，Print 文を使って実行画面に色々な数値や文字を表してきました。しかし，tbasic の実行画面では数値や文字しか表示できません。これに対して，グラフィック画面では，数値や文字だけでなくグラフや絵など表示できます。tbasic のグラフィック画面は実行画面とは役割や使い方が異なり，そのことからグラフィック画面は起動時には表示さません。

実行画面はプログラムの実行結果を表示するものです。これはエディタの画面のように文字を表示し，画面に収まらない場合はスクロールします。

これに対して，グラフィック画面は特別にグラフィック表示が必要な場合に使います。そしてこの画面では改行やスクロールはありません。常にある場所を指定して，そこに描きます。ですから，以前に描いたものの上に描くと，前のものは消えてしまいます。

■ tbasic では，
・実行画面は文字を表示し，画面に収まらない場合はスクロールする。
・グラフィック画面は点を表示し，スクロールせず，固定された座標をもっている。

グラフィック画面は起動時には表示されません。ですから，グラフィック画面を使う場合は，まずそれを表示する必要があります。これをグラフィック画面を開くと言います。グラフィック画面を開くには tbasic では GScreen コマンドを使います。GScreen は Graph Screen の意味で

```
GScreen(横ピクセル数，縦ピクセル数)
```

の形で使います。ここで横ピクセル数，縦ピクセル数は使用するグラフィック画面の大きさの指定です。

例 1.1 (400×200). 例えば

```
GScreen(400,200)
```

とすると，横 400 縦 200 ピクセルのグラフィック画面が表示され，使用できるようになります。グラフィック画面は次のようになります。

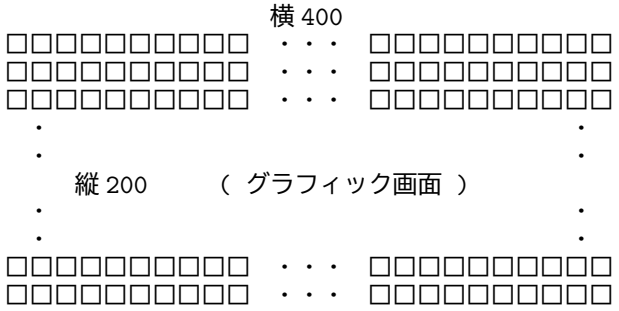


400×200 のグラフィック画面

1.3 点の位置指定

グラフィック画面の点（画素）の位置指定について，横 400 縦 200 のグラフィック画面で説明します。

横 400 縦 200 のグラフィック画面は，以下のように，400×200=80000 個の画素で構成されます。



これらの各点を様々な色で表示して，図や絵やグラフを描きます。これらの点の横の並びを行と言います。今の場合，1 つの行は 400 個の点からなります。また，点の縦の並びを列と言います。今の場合 1 つの列は 200 個の点からなります。グラフィック画面全体は 200 行，400 列の点からなると言えます。

これらの各点には，その位置を表す座標があります。座標は左から数えて何列目かということと，上から数えて何行目かによって表します。一般に

x 列，y 行の位置にある点を座標 $(x,\ y)$

で表します。但し，列・行とも 0 から数え始めます。ですから今のグラフィック画面の場合，

左上隅の座標は $(0,0)$ で，右下隅の座標は $(399,199)$

になります。

この位置指定の方法は一般によく使われる指定方法で，OS レベルでの基本的方法はこの方法です。しかし，実際に使用するとなると，この方法は余り便利ではありません。

例えば，グラフィック画面を，画面全体を普通の数学で使われる $(x,\ y)$ 座標として表すとします。そして，中央の点を原点 $(0,0)$，左端中央を $(-2,0)$，右端中央を $(2,0)$ として，$y=f(x)$ のグラフを描くとしましょう[*3]。このとき，$-2 \leqq x \leqq 2$ に対して，$(x,\ f(x))$ を表す点は，このグラフィック画面での座標の表し方では，大体

$$(100x+200,\ -100f(x)+100)$$

となります。形式的な変換ですが，分かりにくいですし，面倒です。

実は，このような問題の回避するために便利なコマンドが用意されています。それが `Window` です。`Window` は窓と言う意味ですが，ここでは使う座標系を設定します。

[*3] この場合，この画面サイズでは，左下隅の座標は $(-2,-1)$，右上隅の座標は $(2,1)$ となります。

```
Window (x0,y0)-(x1,y1)
```

の形で使います。ここで，(x0,y0)，(x1,y1) は x0<x1，y0<y1 であれば，どの様な数値でも構いません。ただし，普通は GScreen で設定した縦横比とこの Window での縦横比が同じに設定する必要があります。そうでないと，歪んだグラフになります。

このように設定すると，グラフィック画面の左上隅は (x0,y0)，右下隅は (x1,y1) と表されます。例えば，今の場合

```
Window (-2,-1)-(2,1)
```

と設定すると[*4]，$-2 \leqq x \leqq 2$ に対して，$(x,\ f(x))$ を表す点は，

$$(x, -f(x))$$

になります。ここで y 座標の部分にマイナスの符号が付くことに注意しましょう。これは，普通の $(x,\ y)$ 座標では y は下から上へと増加しましたが，この座標では上から下へと増加することによります。

このマイナスも回避する方法があります。それは MathGraph コマンドです。これは tbasic 独自のコマンドですが，最近の BASIC では同様なことは可能です。MathGraph は数学風グラフ座標と言う意味を込めました。使い方は

```
MathGraph On
```

とするだけです。普通の座標設定に戻すときは，MathGraph Off とします。

```
GScreen(400,200)
Window(-2,-1)-(2,1)
MathGraph On
```

このように設定すると，$-2 \leqq x \leqq 2$ に対して，$(x,\ f(x))$ を表す点のグラフィック画面の座標は，

$$(x, f(x))$$

になります。つまり，普通の数学での方法の座標と，グラフィック画面での方法の座標が一致します。

これなら分かりやすいですね。

1.4 グラフィック画面への描画

グラフィック画面が開かれ，上のように座標系が設定されると，そのグラフ画面に対して様々な処理を行うことができます。

最終的にはグラフィック画面上に点として表示するわけですが，それらを簡易化するコマンドが用意されています。これらの処理を行うコマンドとして，次があります。

[*4] GScreen(400,200) の縦横比 2 : 1 と同じ縦横比です。

CLS	画面を消去する。
PSet	色々な色を使って点を描く。
Line	線や長方形を描く。
Circle	円や円弧を描く。
Paint	領域を塗りつぶす。
GLocate	グラフィック画面位置を指定する。
GPrint	グラフィック画面に文字列を表示する。

などがあります。このように tbasic でサポートされるコマンドは基本的なもののみで，複雑な処理は得意ではありませんが，それでも色々なことができます。これらについては「基本コマンド・文の使い方」の項で説明します

1.5 グラフィック画面の閉じ方

利用が終わったグラフィック画面を閉じるには，CloseGScreen を使います。tbasic ではグラフィック画面は 1 つしかありませんので，使い方は

CloseGScreen

とするだけです。このコマンドを実行すると，グラフィック画面が閉じられます。つまり消えてしまいます。

ファイルの場合はオープンしたファイルは「必ず閉じる」と説明しました。しかし，グラフィック画面の場合は少し事情が違います。グラフィック画面の場合，描画が終わったことと画面の利用が終わったこととは違います。プログラム終了後も，グラフィック画面を消したくないこともあると思います。そのような場合，CloseGScreen を実行しなくても構いません。実際，CloseGScreen を実行しない方がよいプログラムの方が多いかもしれません。ですから

グラフィック画面は必ずしも，閉じなくても良い

となります。

では，グラフィック画面が閉じていない状況で，新たに

GScreen(横ピクセル数，縦ピクセル数)

を実行したらどうなるでしょうか。GScreen はグラフィック画面の初期化も行いますから，前のグラフィック画面も初期化されるでしょうか。

これは tbasic の仕様の問題です。初期化する方法も考えられますが，私は初期化しないで，新たな画面を作るように仕様を決めました。つまり

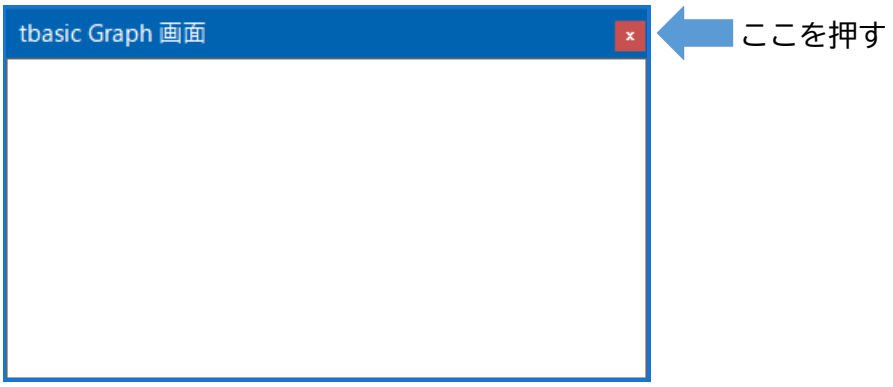
グラフィック画面が閉じていない状態で，GScreen を実行すると，
前あったグラフィック画面と全く独立な新たなグラフィック画面が作られ，
それが操作の対象となります。

ですから，グラフ画面は必要でなくなったときに閉じれば良いことになります。多くのグラフを同時に見た

い場合は，それを消さずに次々と新しいグラフ画面を開き描いていけば，いくつもの画面を同時に見ることができます。[*5]

このことを繰り返すと，グラフィック画面がどんどんと溜まっていくことになります。いくつグラフィック画面ができても tbasic が終了すると，全て自動的に閉じられます。ですから，余り数が多くない場合はそのままにして置けばよいのですが，数が多くなると，グラフィック画面を消去しないと，システムが不安定になることも考えられます。そこでグラフィック画面が多く溜まった場合は手動で消去して下さい。その方法は標準的で

グラフィック画面の右上にある「×」ボタンを押す



となります。

1.6 グラフィック画面の活用

グラフィック画面に描画されたものは，tbasic が終了すると全て閉じられ，すべて消えてしまいます。でもその画面に描かれたグラフ等を利用したい場合もあります。その活用法について少し説明します。基本は

グラフィック画面は画像として保存できる

ということです。この保存の仕方は次の２通りが考えられます。

(1) クリップボードの利用

グラフィック画面をクリップボードにコピーできます。ですから，その画像を別の画像処理ソフトで処理することができます。

クリップボードへのコピーは，グラフィック画面をアクティブにして（グラフィック画面の一部をクリックする），`Alt+PrintScreen` キーを押せば可能です。後は，画像処理ソフト（ペイントでも可能）を起動して，貼り付けを実行すれば画像が取り込めます。

この方法ですと，tbasic のグラフ画面の枠も含めた全体を画像として保存します。保存される画像の大きさは，PC の画面設定に依存します。

[*5] 但し，tbasic でのグラフ操作の対象は最後に開いている画面のみであることに注意してください。

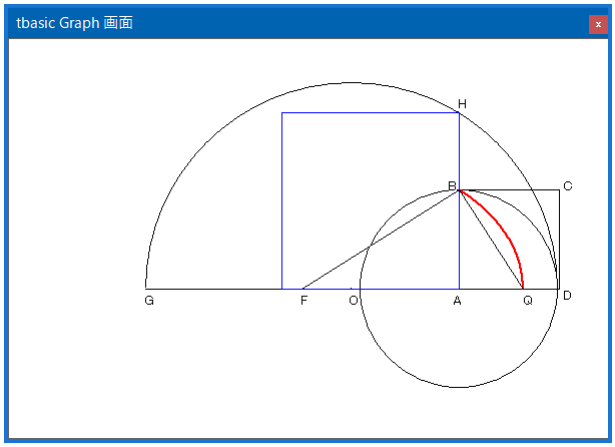
(2) tbasic のコマンドを利用

tbasic では SavePicture コマンドをサポートしています。プログラムの中，或はダイレクトモードでこのコマンドを利用することで，グラフィック画面の中身を jpg，png，gif または bmp ファイルに保存することができます。

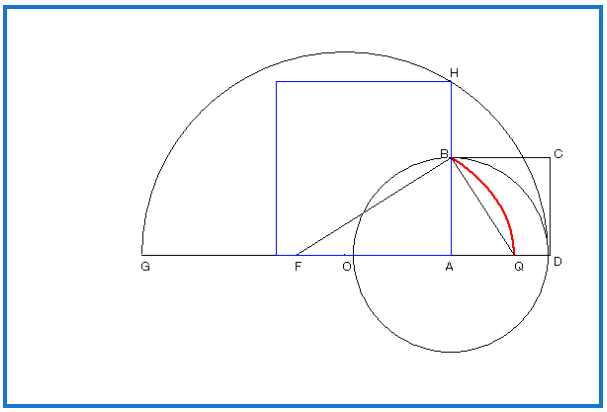
この方法ですと，tbasic のグラフ画面の枠を含まない，画面を描画した解像度と同じ大きさの画像を保存することができます。

現在は色々な種類の画像処理ソフトがあり，色々な画像を作成してくれます。しかし，それらを使っても，種々の正確な計算から得られる画像の作成は，必ずしも簡単ではありません。

簡単な BASIC でのグラフィック処理から得られる画像がそれらに勝ることもあります。以下の図は sample にある enseki.tbt から作成した画像です[*6]。



Alt+PrintScreen キー利用



SavePicture コマンド利用

この画像は，右下にある円と中央の青い正方形が，丁度同じ面積じであることを示すものです。この作図には右中にある赤色の円積線（クァドラトリックス）を使っていますが，これらの線はプログラムで，正確に計算して描いたものですから，図も正確なものです。

このような図を描くのは BASIC のグラフィックの得意とするところです。皆さんも色々なグラフィックに挑戦してみて下さい。

[*6] 以下の画像に水色枠がありますが，これは見やすさのために付けたもので実際の画像にはありません。

1.7 高解像度画像の利用

現在のパソコンで利用されているディスプレイの解像度はフル HD（1920×1080）が標準的と言われています。例えば，このようなパソコンで，横 1600 縦 800 のグラフィック画面を使いたいとすれば，ただ単に

```
GScreen(1600,800)
```

とすれば良いでしょう。しかし，最近では標準的なケータイのカメラでも 3000×4000 ピクセルもの解像度を持ちます[*7]。このよう設定で撮影された画像は上の方法では，パソコンの画面に表示することはできません。

現在，パソコン上でこのような高解像度な画像を表示する場合は，普通は縮小して表示し，必要に応じて拡大利用します。このような利用法は tbasic でも拡張型の GScreen を利用すると可能です。

拡張型の GScreen は，

```
GScreen (x,y,x1,y1)
```

の形で使います。このコマンドを実行すると，横 x ピクセル，縦 y ピクセルの大きさのグラフ画面を，横 x1 ピクセル，縦 y1 ピクセルのスクリーンの中に開きます。

この指定は，操作対象とする画像を横 x ピクセル，縦 y ピクセルの大きさとし，それをディスプレイ上に表示する場合，横 x1 ピクセル，縦 y1 ピクセルのスクリーンを利用するというものです。

この場合，大きい画像をそれより小さい画面で表示しますから，どのように表示するのかが問題になりますが，tbasic では，この状況に対して次の 3 つの対応方法があります。これに関連するコマンドとして，GStretch と GScroll があります。

(1) 特に何もしない。これは，GScreen (x,y,x1,y1) でスクリーンを開いた後の状態です。表示されるのは，全体の横 x ピクセル，縦 y ピクセルの内，左上部分の横 x1 ピクセル，縦 y1 ピクセルの部分です。
(2) 全体を縮小して表示する。GScreen (x,y,x1,y1) でスクリーンを開いた後，GStretch ON コマンドを実行した結果です。全体の画像を縮小して横 x ピクセル，縦 y ピクセルの部分に表示します。
(3) スクリーンをスクロールさせ，必要な部分を表示する。GScreen (x,y,x1,y1) でスクリーンを開いた後，GScroll ON コマンドを実行した結果です。横 x1 ピクセル，縦 y1 ピクセルの Window 右，下に表示されるスライダーを操作することで，表示部分を移動させることができます。

例えば，ここに bigMandelbrot.png という 4000×4000 ピクセルの画像があるとします。この画像を，400×400 の画面に表示することにして，

```
GScreen(4000,4000,400,400)
LoadPicture("bigMandelbrot.png")
```

を実行したときの，画像が図 1 です[*8]。図 1 は，1/10 の左上の部分を表示していますから，黒が表示されているだけです。

更に，

```
GStretch ON
```

を追加したのが，図２です。図２は全体を縮小して表示しますから，画像の全体が分かります。

[*7] tbasic で利用可能な画像の最大は ver.1.61 より 4500×4500 ピクセルになりました。
[*8] LoadPicture については，4 節を参照してください。



図 1: 無指定

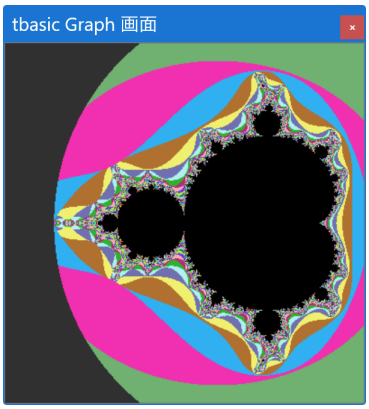


図 2: GStretch On

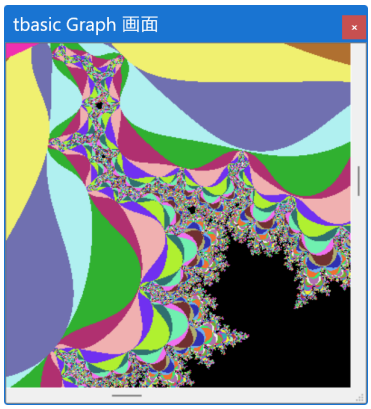


図 3: GScroll On

図 3 は，GStretch ON の代わりに，GScroll ON を追加して，スクロールバーのスライダーを少し動かしたときのものです。スライダーを適当に動かすことで，全体画像の 1/10 の部分を縮小せず表示できます。

上の操作は，メモリ上にある画像をディスプレイに表示する方法を指定するもので，画像そのものを変化させているわけではありません[*9]。画像そのものに対する処理，操作は画像の大きさに関わらず，以下の節で説明する各種コマンドで可能です。実際，上の画像 bigMandelbrot.png も，tbasic で作成したものです。ですから，大きな画像に対する処理も，その画像の状況を見ながら操作することが可能です。

[*9] 対象とする画像そのものの大きさを変える方法としては ResizeImage コマンドがあります。

2　tbasic グラフィック処理法

前項では，グラフィックの概要を説明しました。ここでは続いて，具体的な使い方を説明します。

2.1　色の指定

グラフィック画面では**背景色**，**前景色**，**境界色**と３種類の色の項目があります。

- 背景色はグラフィック画面の地の色
- 前景色は点など書くときに使う色，
- 境界色は `Paint` による塗りつぶしのときに境界として使う色です。

これらは，特に指定しない場合，
・背景色：白，　・前景色：黒，　・境界色：黒
となります。必要に応じてこれらの色を指定して下さい。

■背景色
背景色はグラフィック画面をオープンするときの，地の色です。ですから，画面をオープンする前に指定する必要があります。オープン後に背景色を指定することも可能ですが，その場合は，`Cls 2` を実行する必要があります。グラフィック画面は `Cls 2` を行ったとき初期化されます

■前景色
前景色は点などを書くときの色です。様々な図形を描くとき適宜指定します。

■境界色
境界色も塗りつぶしのときに適宜指定します。特に指定しない場合は前景色と同じになります。

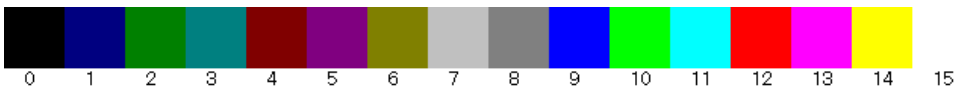
背景色，前景色，境界色の指定はそれぞれ

```
GBackColor
GForeColor
GBorderColor
```

で設定します。設定方法は色番号による指定，色の名前による指定，html 形式による指定があります。

■色番号による方法
使える色は 16 色で，色番号 0 から 15 まで指定できます。色は次のように割り当てられています。

番号	:	名前	番号	:	名前	番号	:	名前	番号	:	名前
0	:	Black	1	:	Navy	2	:	Green	3	:	Teal
4	:	Maroon	5	:	Purple	6	:	Olive	7	:	Silver
8	:	Gray	9	:	Blue	10	:	Lime	11	:	Cyan(Aqua)
12	:	Red	13	:	Fuchsia	14	:	Yellow	15	:	White



この方法では，例えば背景色に黒を指定するには，

```
GBackColor=0
```

とします。但し，色番号による指定は，旧式の方法です。可能なら，次の色の名前による指定を推奨します。

■色の名前による指定

この方法色の名前を使って設定します。例えば背景色に黒を指定するには，

```
GBackColor="Black"
```

とします。

tbasic Ver.1.5 より，名前で指定できる色が増えました。この名前は Microsoft の .NET Framework で使用されている Windows PC 上では標準的なものです。全部で 140 種類の名前が定義されていますが，同じ色で 2 種類の名前を持つものが二つあります。Aqua と Cyan，Fuchsia と Magenta は同じ色です。ですから実際は 138 種の色が名前で利用できます。詳細はヘルプを参照してください。

次は利用できる色の一覧です。Samples にある，ColorSample15.tbt を実行したものです。



■ html 形式による指定

html 文書で色を指定する方法と同じ RGB の輝度を使って設定します。この方法だと 138 色より多いカラー指定ができます。例えば，赤：FF，緑：CC，青：AA を使って

```
GBackColor="#FFCCAA"
```

と指定します。これは肌色 を指定しています。

2.2 初期設定

グラフィック画面を使うときの標準的な初期設定は次のようになります。

(1) 背景色の設定
(2) 前景色の設定
(3) グラフィック画面のオープン
(4) 座標の設定

このうち，色の設定は省略することができます。設定しない場合，背景色（画面の地の色）は白，前景色（画面に描く点などの色）は黒になります。

色の設定方法はいくつかありますが，色の名前を指定する方法か html 式に RGB で指定するのが良いでしょう。色番号を使う方法もありますが，この方法は旧式な方法です。

詳しい色の指定方法は以下にある「色の指定の例」，更に，ヘルプの「色の指定」の項目を参照して下さい。

例えば初期設定として，背景色を紺，前景色を水色，グラフィック画面の大きさは，600×400，座標は数学的座標で $(-3,-2)$〜$(3,2)$ で指定するとしましょう。この場合の初期設定は

```
GBackColor="Navy"
GForeColor="Cyan"
GScreen(600,400)
Window (-3,-2)-(3,2)
MathGraph On
```

となります。

2.3 グラフィック画面への描画

グラフィック画面が開かれ座標系が設定されると，そのグラフ画面に対して様々な処理が行えます。

これらの処理を行うコマンドとして

CLS	画面を消去する。
PSet	色々な色を使って点を描く。
Line	線や長方形を描く。
Circle	円や円弧を描く。
Paint	領域を塗りつぶす。
GLocate	グラフィック画面位置を指定する。
GPrint	グラフィック画面に文字列を表示する。

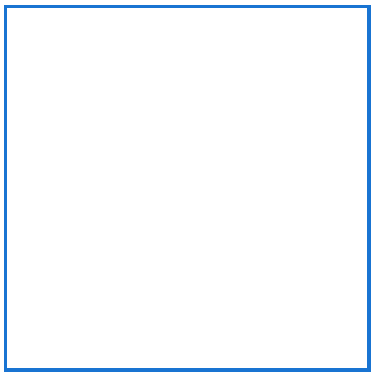
などがあります。これらに全体については「基本コマンド・文の使い方」の項で説明します。

ここでは，そのうち最も基本的なコマンドである PSet，Line と Circle を使った例を説明します。

画面を第一象限の正方形領域，座標は正の部分として，初期設定したものが次です。Rnd 関数を使うため[*10]，座標系を左下 $(0,0)$，右上 $(1,1)$ と設定します。

[*10] Rnd は $0 \leqq$ Rnd < 1 の範囲の値を取ります。

今の場合，何も表示していないので白紙の画面です。



```
GBackColor="White"
GForeColor="Black"
GScreen(300,300)
Window (0,0)-(1,1)
MathGraph On
```

実行結果

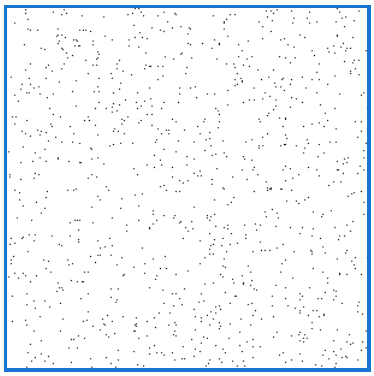
この画面に，ランダムに 1000 個の点を描きましょう。

点を描くコマンドは PSet です。PSet は Point Set の意味です。PSet (x,y) の形で使います

```
PSet (Rnd,Rnd)
```

としましょう。これを 1000 回続けるために For 文を使います。これをまとめると次のプログラムが得られます。

例 2.1 (1000Points.tbt [*11]).



```
'1000 個の点
GBackColor="White"
GForeColor="Black"
GScreen(300,300)
Window (0,0)-(1,1)
MathGraph On
For i=1 To 1000
   PSet (Rnd,Rnd)
Next i
End
```

実行結果

これだけだと，黒地に白の点があるだけなので，色を変化させて見ましょう。色指定は GForeColor で設定することを推奨します。

色を変化させる方法は色々考えられますが，ここでは配列変数を使って色を指定する方法の例をあげます。

配列変数に色の名前を 3 色設定して，その名前を順次選択します。そのための処理が，

```
j=((i-1) mod 3)+1
```

です。この値は，i=1,2,3,4,5,6,... とすると，j=1,2,3,1,2,3,... となります。

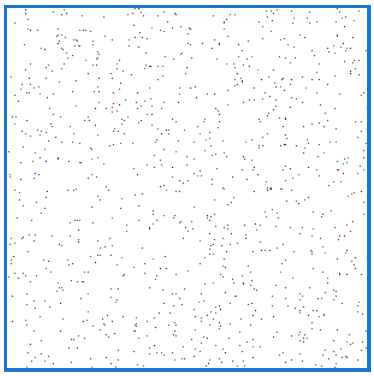
[*11] このファイルは 同梱の samples¥Tutorial¥Graphic にあります。以下のファイルも同様です。

GForeColor で点を描く色を設定し，点を描きます。GForeColor は 数値定数，数値変数，文字列定数，文字列変数しか右辺に置けませんので，まず，色を表す文字列を計算し，その結果を予め変数に入れて置きます。

また，途中経過を見るために，実行を遅らせるコマンド Sleep を使っています。

例 2.2 (1000ColorPoints.tbt).

```
'1000 個の点
Dim CLR$(3)
CLR$(1)="DarkGreen"
CLR$(2)="DarkRed"
CLR$(3)="Navy"
GBackColor="White"
GForeColor="Black"
GScreen(300,300)
Window (0,0)-(1,1)
MathGraph On
For i=1 To 1000
   Sleep(10)
   C$ = CLR$(((i-1) mod 3)+1) :'1,2,3 を順次取る。
   GForeColor= C$
   PSet (Rnd,Rnd)
Next i
End
```



実行結果

このプログラムでは,

```
C$ = CLR$(((i-1) mod 3)+1)
```

で色を決めています。$i = 1, 2, 3, \ldots$ に対して，C$は”DarkGreen”，”DarkRed”，”Navy” の順に繰り返し，設定されます。拡大すると色は確認できます。

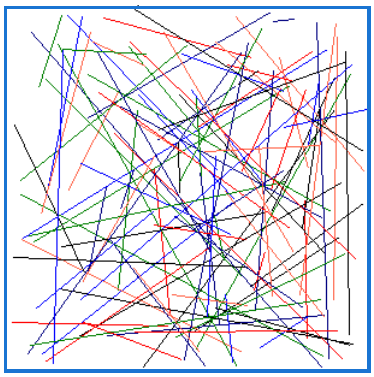
同様のことを 線分を使って行いましょう。線分を描くには Line を使います。Line は線の意味ですが,

```
Line (x1,y1)-(x2,y2)
```

の形で使います。(x1,y1) から (x2,y2) への線を引きます。これを使うと次のプログラムが得られます。1000 本の線は多いので，今度は 100 本の線にします。また色は 6 色使うことにし，その色の名前を予め配列変数に格納して置きます。

例 2.3 (100Lines.tbt).

```
'100 本の線
Dim CLR$(6)
 CLR$(1)="Black" : CLR$(2)="Green" : CLR$(3)="Navy"
 CLR$(4)="Tomato": CLR$(5)="Red"   : CLR$(6)="Blue"
BackColor="White"
ForeColor="Black"
GScreen(300,300)
Window (0,0)-(1,1)
MathGraph On
For i=1 To 100
   Sleep(10)
   C$ = CLR$(((i-1) mod 6)+1) :'1,2,...,6 を順次取る。
   GForeColor= C$
   Line (Rnd,Rnd)-(Rnd,Rnd)
Next i
End
```



実行結果

プログラムで，

```
CLR$(1)="Black" : CLR$(2)="Green" : CLR$(3)="Navy"
CLR$(4)="Tomato": CLR$(5)="Red"   : CLR$(6)="Blue"
```

と，文区切り記号コロン: を使って複文にしています。これは配列変数の設定をまとめて見やすくするためのものです。コロンを使わずに，6 行で書いても同じです。

また，今回は 6 色使うので，6 色の名前を配列変数に設定して，

```
C$ = CLR$(((i-1) mod 6)+1)
```

で，6 色を順に繰り返し C$に設定しています。

Line は線分を描くのが第一ですが，長方形を描くこともできます。

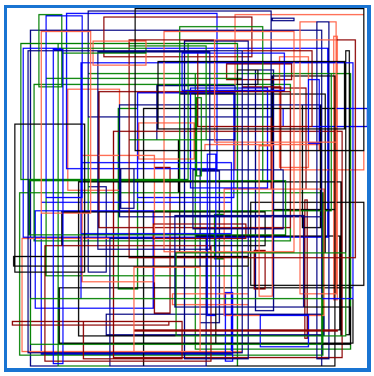
```
Line (x1,y1)-(x2,y2),,B
```

とすると，(x1,y1) と (x2,y2) を対角頂点とする長方形を書きます。[*12]

これを使って，長方形を描くプログラムにしてみると，次のようになります。上のプログラムで，Line に B オプションを付けただけです。

例 2.4.

```
'100 個の長方形
Dim CLR$(6)
 CLR$(1)="White" : CLR$(2)="Green": CLR$(3)="Blue"
 CLR$(4)="Yellow": CLR$(5)="Red"  : CLR$(6)="Cyan"
BackColor="Black"
ForeColor="White"
GScreen(300,300)
Window (0,0)-(1,1)
MathGraph On
For i=1 to 100
   Sleep(10)
   C$ = CLR$(((i-1) mod 6)+1) :'1,2,...,6 を順次取る。
   GForeColor= C$
   Line (Rnd,Rnd)-(Rnd,Rnd),,B
Next i
End
```



実行結果

最後に Circle を使った例をあげます。Circle は円の意味ですが，

```
Circle (x,y),r
```

の形で使い，(x,y) を中心とする半径 r の円を描きます。また，

```
Circle (x,y),r,,,,,F
```

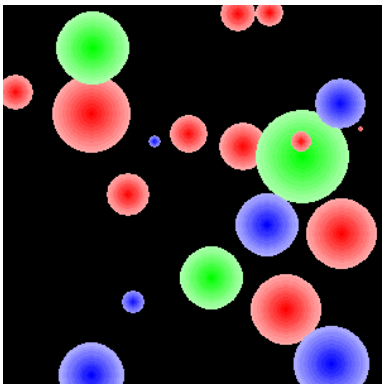
の形で使うと，(x,y) を中心とする半径 r の中を塗りつぶした円を描きます。

[*12] また Line (x1,y1)-(x2,y2),,BF とすると，(x1,y1) と (x2,y2) を対角頂点とする長方形を書き中を塗りつぶします。

ここでは，html 形式による色指定を使って，グラデーション風の円を描いてみましょう。次のプログラムは赤，緑，青の円の中をグラデーション風に塗りつぶした図形を 20 個描いています。この例では背景は黒に設定しています。３色の中でどれを使うかと，位置，半径は乱数を使って決めています。また，実行時に，Randomize を行っているので[*13]，毎回異なる絵が描かれ，同じ絵を 2 度描くことはできません。

例 2.5. [20Circles.tbt]

```
'20 個の円
Dim CLR$(2)
GBackColor="Black": GForeColor="White"
GScreen(300,300)  : Window (0,0)-(1,1)
MathGraph On
Randomize
For i=1 To 20
   X=Rnd:Y=Rnd:R0=Rnd/80: CN=Int(Rnd * 3)
   For j=10 To 0 step -1
      CLR$(CN)="FF"
      CLR$((CN+1) mod 3)=Right$("0"+Hex$(j*17),2)
      CLR$((CN+2) mod 3)=CLR$((CN+1) mod 3)
      C$="#"+CLR$(0)+CLR$(1)+CLR$(2)
      R=R0*(j+1)
      GForeColor= C$
      Circle (x,y),R,,,,,F
   Next j
Next i
End
```



実行結果

上のプログラムでは，CLR$(0)：赤の輝度，CLR$(1)：緑の輝度，CLR$(2)：青の輝度をそれぞれ設定して，C$="#"+CLR$(0)+CLR$(1)+CLR$(2) で html 形式の色 C$を決めています。まず，0,1,2 をランダムにとる変数 CN を決め，それにより，色を決めます。それに従って，それぞれ，

赤：#FFAAAA, #FF9999, …, #FF1111, #FF0000
緑：#AAFFAA, #99FF99, …, #11FF11, #00FF00
青：#AAAAFF, #9999FF, …, #1111FF, #0000FF

の文字列を生成し，それを C$として，GForeColor に設定しています。そのために，Hex$(j*17) を使っています。Hex$は，16 進数の文字列に変換する関数です。ですから，$17 = 16 + 1 = 11(16進)$ より，

- j=1,…,9 に対して，Hex$(j*17) は 16 進表示として，jj を表します。

更に，j=10 の場合は，AA を表します。j=0 の場合は，0 となるので，その場合，左に 0 を加えるようにします。

```
Right$("0"+Hex$(j*17),2)
```

とすると，j=0,…,10 に対して，16 進数文字列として jj を表します。

html 形式の色指定の自動化には，上のような文字列処理をする必要があり，幾分煩雑ですが，細かい色指定ができます。工夫により美しい絵を作ることが可能でしょう。

「基本コマンド・文の使い方」では個々のコマンドについて更に説明をします。

[*13] Randomize は時刻を使って乱数の初期化を行います。

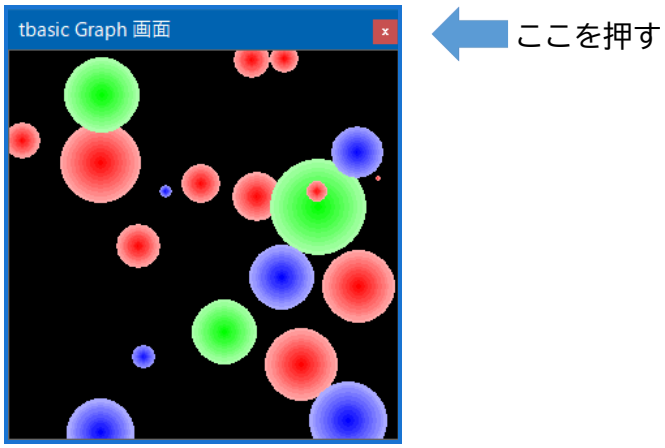
2.4　グラフィック画面の閉じ方

利用が終わったグラフィック画面を閉じるには，CloseGScreen を使いますが，上の例では使っていませんでした。つまり，

グラフィック画面は，必ずしも閉じなくても良い

ということでした。でも必要なくなったら，手で閉じてください。

グラフィック画面の右上にある「×」ボタンを押します。



3　グラフィック基本コマンド・文の使い方

前節では，グラフィックと使用法の概要を説明しました。ここでは続いて，以下の基本コマンドの使い方を説明します。

(1) Cls
(2) PSet
(3) Line
(4) Circle
(5) Paint
(6) GLocate と GPrint

3.1 Cls

Cls は Clear Screen の意味で画面消去を行います。グラフィック画面の消去は Cls 2 を使います[*14]。

```
Cls 2
```

とするとグラフィック画面が消去されます。

Cls 2 の実際の処理は，背景色でグラフィック画面全体を塗りつぶします。背景色の設定は，GScreen の前に行うのが基本です。しかし，同じグラフ画面で背景色を変える場合など，GScreen の設定後に GBackColor を設定することもあるかも知れません。そのようなとき合，グラフ画面を初期化する場合は，Cls 2 を実行します。Cls 2 を実行すると，その GBackColor の色で塗りつぶされます。

Cls 2 を使った例をあげます。

例 3.1 (Cls2.tbt).

```
Dim CLR$(2)
GScreen(300,300)
For i=1 to 10
   CN=Int(Rnd * 3)
   For j=15 To 0 step -1
      Sleep(100)
      CLR$(CN)="FF"
      CLR$((CN+1) mod 3)=Right$("0"+Hex$(j*17),2)
      CLR$((CN+2) mod 3)=CLR$((CN+1) mod 3)
      C$="#"+CLR$(0)+CLR$(1)+CLR$(2)
      GBackColor= C$
      Cls 2
   Next j
Next i
End
```

このプログラムでも，CLR$(0)：赤の輝度，CLR$(1)：緑の輝度，CLR$(2)：青の輝度を設定することにします。

```
CN=Int(Rnd * 3)
```

で，ランダムに 0,1,2 を選択し，その色に対して

```
CLR$(CN)="FF"
```

と最大輝度を指定します。CLR$(CN) 以外の 2 つの成分は CLR$((CN+1) mod 3) と CLR$((CN+2) mod 3) になります。今回は各場合に対して，

赤：#FFFFFF, #FFEEEE, …, #FF1111, #FF0000
緑：#FFFFFF, #EEFFEE, …, #11FF11, #00FF00
青：#FFFFFF, #EEEEFF, …, #1111FF, #0000FF

を生成し，C$="#"+CLR$(0)+CLR$(1)+CLR$(2) で html 形式の色 C$を決めています。

[*14] Cls，Cls 1 は実行画面の消去，Cls 2 はグラフィック画面の消去，Cls 3 は両画面の消去です。

このプログラムを実行すると，背景色がグラデーション風に順次変化していることが確認できます。

3.2 PSet

PSet は Point Set (点を打つ) の意味で，グラフィック画面に点を表示します。

```
PSet (x,y)
```

の形で使います。前景色を指定することで様々な色の点を表示できます[*15]。

例を挙げましょう。ここでは，説明のために行番号を付けますが，実際のプログラムでは行番号はありません。

以下のプログラムは原点 (0,0) を中心とする半径 1 の円と，点 (1,1) を中心とする半径 1 の円で囲まれる図形の中にある点をランダムに表示するものです。

```
01 GBackColor = "Black"
02 GForeColor = "White"
03 GScreen(300,300)
04 Window(0,0)-(1,1)
05 MathGraph On
06 For i=1 to 10000
07    x = RND
08    y = RND
09    If (x^2+y^2 <= 1) and ((x-1)^2+(y-1)^2 <=1) then
10       PSet (x,y)
11    End if
12 Next i
13 End
```

説明.

- 1 行〜5 行はグラフィック画面の初期設定です。背景色は黒，前景色は白に設定しています。グラフィック画面は (0,0) と (1,1) を端点とする正方形です。
- 6 行から 12 行で点を 10000 個取り，条件を満たすとき，描画をします。
- 7 行，8 行で x, y をそれぞれランダムに決めます。
- 9 行で，(x^2+y^2 <= 1) and ((x-1)^2+(y-1)^2 <=1) のとき，原点 (0,0) を中心とする半径 1 の円と，点 (1,1) を中心とする半径 1 の円で囲まれる図形の中（境界も含む。）に (x,y) があることの判定をします。
- 10 行で，その場合，前景色 白で点 (x,y) を描画します。

□

上の例では白一色で点を表示していますが，html 形式を使うともっと多くの色を使うことができます。このプログラムランダムに色を指定するものにしてみましょう。

16 種の文字列”0”,”1”,”2”,...,”D”,”E”,”F” をランダムに与えるために Hex$(16*RND) を使います[*16]。これらを並べて

[*15] 色番号 C を直接指定して PSet (x,y),C の形で使うこともできますが，この方法では色番号しか使えません。色の指定は，GForeColor 等を使うことを推奨します。

[*16] x =16*RND は $0 \leqq x < 16$ となる x を与えますが，Hex$(x) は 16 進整数文字列を返します。

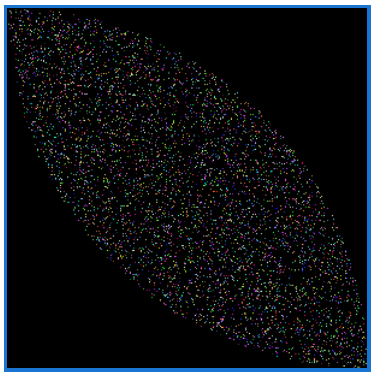
```
C$="#"+Hex$(16*RND)+Hex$(16*RND)+Hex$(16*RND)+Hex$(16*RND)+Hex$(16*RND)+Hex$(16*RND)
```

とすると，C$は，ランダムに html 形式による色指定の文字列を与えます。これを加えると，次のプログラムが得られます。

例 3.2 (10000PSet.tbt).

```
GBackColor="Black"
GScreen(300,300)
Window(0,0)-(1,1)
MathGraph On
For i=1 to 10000
   x = RND
   y = RND
   If (x^2+y^2 <= 1) and ((x-1)^2+(y-1)^2 <=1) then
      C$="#"+Hex$(16*RND)+Hex$(16*RND)+Hex$(16*RND)+Hex$(16*RND)+Hex$(16*RND)+Hex$(16*RND)
      ForeColor = C$
      PSet (x,y)
   End if
Next i
End
```

以下が実行結果です。



3.3 Line

Line は線や長方形を描きます。

```
Line (x1,y1)-(x2,y2)
```

の形で使うと，(x1,y1) から (x2,y2) への線を引きます。長方形を描くこともできます。

```
Line (x1,y1)-(x2,y2),,B
```

とすると，(x1,y1) と (x2,y2) を対角頂点とする長方形を書きます。ここでの B は Box の意味です。また

```
Line (x1,y1)-(x2,y2),,BF
```

とすると，(x1,y1) と (x2,y2) を対角頂点とする長方形を書き中を塗りつぶします。ここで，BF は Box Fill の意味です。これらの場合，色は前景色で描きます。この色の指定は，GForeColor で設定します[*17]。

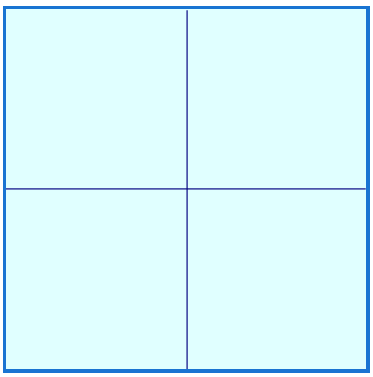
[*17] 色番号 C で直接指定することもできます。例えば Line (x1,y1)-(x2,y2),C とすると，(x1,y1) から (x2,y2) への線を色番号 C の色で引きます。

Line を使う典型的な例を２つ挙げましょう。

まず，次は座標軸を描いています。平面グラフを描くときによく使うタイプのものです。

例 3.3 (座標軸の描画).

```
' 座標軸を描く
GBackColor="LightCyan"
GScreen(300,300)
Window (-2,-2)-(2,2)
MathGraph On
GForeColor = "Navy"
Line (-2,0)-(2,0) :'x 軸
Line (0,-2)-(0,2) :'y 軸
GForeColor = "Black"
End
```



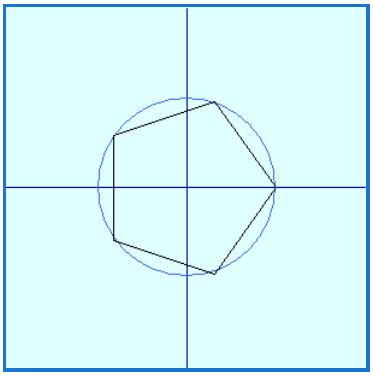
実行結果

このプログラムでは，背景を明るい水色 (LightCyan) とし，xy 軸を紺色 (Navy) で描いています。

次に，半径 1 の円周上に正多角形を描いてみましょう。この例では正５角形です。n を変更することで，色々な正多角形を描くことができます。

例 3.4 (正多角形の描画 (Polygon.tbt)).

```
' 正 n 多角形
n=5
GBackColor="LightCyan"
GScreen(300,300)
Window (-2,-2)-(2,2)
MathGraph On
GForeColor = "Navy"
Line (-2,0)-(2,0) :'x 軸
Line (0,-2)-(0,2) :'y 軸
GForeColor = "RoyalBlue"
Circle(0,0),1
ForeColor = "Black"
For i=0 To n-1
   x0=Cos(2*Pi*i/n)
   y0=Sin(2*Pi*i/n)
   x1=Cos(2*Pi*(i+1)/n)
   y1=Sin(2*Pi*(i+1)/n)
   Line (x0,y0)-(x1,y1)
Next i
End
```



実行結果

正多角形を描く方法は色々考えられますが，円周を n 等分する点を結ぶ方法が最も簡単で，汎用性があります。

座標軸を描画した後に，正多角形が円周上にあることを確認するため，GForeColor = "RoyalBlue"で半径 1 の円を描いています。その後，正多角形を描画します。

半径 1 の円周上の正多角形の点は円周 2*Pi を n 等分した点から得られるので，その点の座標 (x,y) を Cos(2*Pi*i/n),Sin(2*Pi*i/n) で求め，それらを順次 Line で結んでいます。

3.4 Circle

Circle は円の意味ですが，円や円弧を描くときに使います。細かいオプション指定が可能ですので，必要な場合はヘルプを参照してください。普通は，

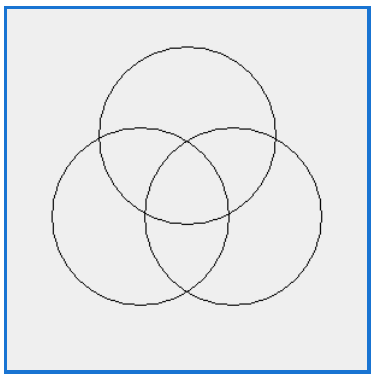
```
Circle (x,y),r
```

の形で使い，(x,y) を中心とする半径 r の円を書きます。オプションで色番号による描画色を指定することもできますが，色番号では 16 色のみの指定になります。色番号の指定がない場合は，前景色によって描かれます。前景色は GForeColor によって，細かく指定できますので，色は GForeColor で指定することを推奨します。

例を挙げましょう。次の例は３つの円を描いています。

例 3.5 (３円の描画).

```
' ３つの円を描く
GBackColor="#EFEFEF"
GScreen(300,300)
Window (-2,-2)-(2,2)
MathGraph On
GForeColor = "Black"
For i=0 to 2
   x=Cos(2*Pi*i/3+Pi/2)*0.6
   y=Sin(2*Pi*i/3+Pi/2)*0.6
   Circle(x,y),1
Next i
End
```



実行結果

Window (-2,-2)-(2,2) として，原点 $(0,0)$ を中心として一辺が 4 の正方形を指定しています。この中に半径 1 の円を３つ描きます。均等に配置するように中心 $(0,0)$ から半径 0.6 の円周上に３円の中心を置きます。３円の中心点は，2π を 3 等分し決めますが，2π を 3 等分した $2\pi * i/3$ の位置は $i = 0$ のとき x 軸上にあるので，y 軸上にあるように $\pi/2$ だけ回転させて，

```
x=Cos(2*Pi*i/3+Pi/2)*0.6
y=Sin(2*Pi*i/3+Pi/2)*0.6
```

で座標 (x, y) を決めています。ここで，背景色は薄い灰色 (GBackColor="#EFEFEF") を指定しています。

3.5 Paint

Paint は色を塗ると言う意味で，ある領域を塗りつぶします。

`Paint (x,y)`

の形で使い，点 (x,y) を含む境界色で囲まれた領域を前景色で塗りつぶします。境界色は GBorderColor，塗りつぶす色は GForeColor で指定することを推奨します[*18]。ここで領域が境界色で閉じていないと，グラフィック画面全体が塗りつぶされてしまうことに注意が必要です。

上の３つの円の各領域を光の３原色が加算混合になるように塗りつぶしてみましょう。

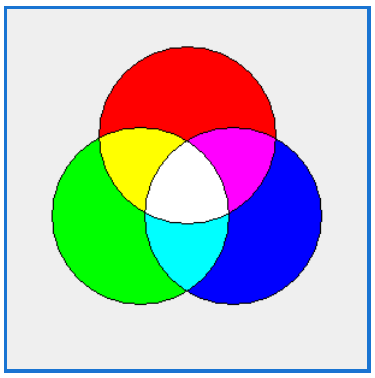
例 3.6 (光の３原色（加算混合）(AdditivePrimaryColors.tbt)).

```
' 光の３原色（加算混合）
GBackColor="#EFEFEF"
GScreen(300,300)
Window (-2,-2)-(2,2)
MathGraph On
GForeColor = "Black"
For i=1 to 3
   x=Cos(2*Pi*i/3+Pi/2)*0.6
   y=Sin(2*Pi*i/3+Pi/2)*0.6
   Circle(x,y),1
Next i
GBorderColor = "Black"

GForeColor ="#FF0000":' 赤
Paint (Cos(Pi/2)*0.8,Sin(Pi/2)*0.8)
GForeColor ="#00FF00":' 緑
Paint (Cos(2*Pi/3+Pi/2)*0.8,Sin(2*Pi/3+Pi/2)*0.8)
GForeColor ="#0000FF":' 青
Paint (Cos(2*Pi*2/3+Pi/2)*0.8,Sin(2*Pi*2/3+Pi/2)*0.8)

GForeColor ="#FF00FF":' 紫
Paint (Cos(Pi/3)*0.6,Sin(Pi/3)*0.6)
GForeColor ="#FFFF00":' 黄
Paint (Cos(2*Pi/3+Pi/3)*0.6,Sin(2*Pi/3+Pi/3)*0.6)
GForeColor ="#00FFFF":' 水色
Paint (Cos(2*Pi*2/3+Pi/3)*0.6,Sin(2*Pi*2/3+Pi/3)*0.6)

GForeColor ="#FFFFFF":' 白
Paint (0,0)
End
```



実行結果

３つの円で分けられた７つの領域に対して対応する色を html 形式で指定し，塗りつぶしています。外側の円の部分が３原色，赤 "#FF0000"，緑 "#00FF00"，青 "#0000FF" です。

その内側の２円の共通部分で決まる領域が，黄，水色，紫です。これらの色は例えば，赤 "#FF0000"と緑"#00FF00"の共通部分は黄色"#FFFF00"のように，丁度，可算方式で決まっています。

中心にある３円の共通部分は３原色の加算で白"#FFFFFF"となります。

ここで境界色 BorderColor の設定を忘れると，全画面を塗りつぶしてしまいますので，注意が必要です。

[*18] 境界色や塗りつぶす色を色番号で直接指定することもできますが，非推奨です。

3.6 GLocate と GPrint

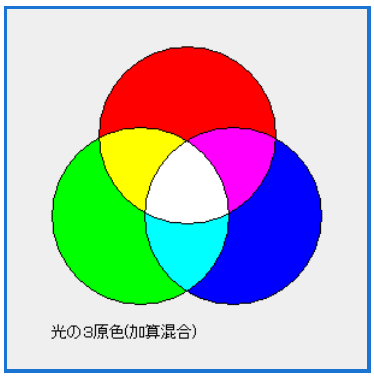
GPrint は Graph Print の意味で，グラフィック画面に前景色で文字列を表示します。

`GPrint 文字列（定数または変数）`

の形で使います。GLocate は Graph Locate の意味で，`GLocate（x,y）`の形で使い，`GPrint` での表示位置を指定します。例えば上のプログラムの End の前に

```
GForeColor = "Black"
GLocate (-1.5,-1.5)
GPrint "光の３原色（加算混合）"
```

を入れると次の結果が得られます。

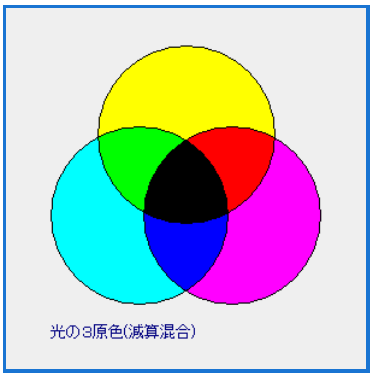


同様に色を入れ替えて，減算混合の３原色（黄，水色，紫）の表示プログラムができます。減算混合では，黄色"#FFFF00"と水色"#00FFFF"の共通部分が，緑"#00FF00"となるように，AND を取る減算方式になります。

例 3.7 (光の３原色（減算混合）(SubtractivePrimaryColors.tbt)).

```
' 光の３原色（減算混合）
GBackColor="#EFEFEF"
GScreen(300,300)
Window (-2,-2)-(2,2)
MathGraph On
GForeColor = "Black"
For i=1 to 3
   x=Cos(2*Pi*i/3+Pi/2)*0.6
   y=Sin(2*Pi*i/3+Pi/2)*0.6
   Circle(x,y),1
Next i
GBorderColor = "Black"
GForeColor ="#FFFF00":' 黄
Paint (Cos(Pi/2)*0.8,Sin(Pi/2)*0.8)
GForeColor ="#00FFFF":' 水色
Paint (Cos(2*Pi/3+Pi/2)*0.8,Sin(2*Pi/3+Pi/2)*0.8)
GForeColor ="#FF00FF":' 紫
Paint (Cos(2*Pi*2/3+Pi/2)*0.8,Sin(2*Pi*2/3+Pi/2)*0.8)

GForeColor ="#FF0000":' 赤
Paint (Cos(Pi/3)*0.6,Sin(Pi/3)*0.6)
GForeColor ="#00FF00":' 緑
Paint (Cos(2*Pi/3+Pi/3)*0.6,Sin(2*Pi/3+Pi/3)*0.6)
GForeColor ="#0000FF":' 青
Paint (Cos(2*Pi*2/3+Pi/3)*0.6,Sin(2*Pi*2/3+Pi/3)*0.6)
GForeColor ="#000000":' 黒
Paint (0,0)
GForeColor = "Navy"
GLocate (-1.5,-1.5)
GPrint "光の３原色（減算混合）"
End
```



実行結果

4 活用コマンド

ここでは，グラフを更に活用するための，状況によっては便利なコマンドを 4 つ説明します。

4.1 LoadPicture

tbasic でのグラフ操作は主としてグラフを作成するものでしたが，既に作成してある，画像などを利用したいこともあるでしょう。LoadPicture はこのような場合利用します。

使い方は，例えば，

```
LoadPicture("Sample.jpg")
```

のように使い，これで，"Sample.jpg"をグラフ画面に表示させることができます

ここで，注意しなければならないことがあります。それは表示する画像とそれを表示するグラフ画面の大きさを一致させることです。一致していないと，一部分しか表示されないとか，余白ができることになります。

このことから，LoadPicture を実行するときは，次を確認する必要があります。

(1) 画像ファイルのパス確認する[*19]。

(2) 画像ファイルの大きさを知る。
画像の縦・横サイズ xs,ys が不明の場合は，GetImageSize$関数で取得可能です。

(3) 適切なグラフ画面を開く。
xs,ys が比較的小さい場合は，GScreen(xs,ys) でグラフ画面を開きます。xs,ys が大きい場合は，比較的小さい xst,yst に対して，GScreen(xs,ys,xst,yst) でグラフ画面を開きます[*20]。

(4) LoadPicture を実行する。その際，ファイルパスを確認する

[*19] 以下の例では，画像ファイルはカレントディレクトリにあるとします。そうでないときはファイルを開くときと同じように，ファイルはフルパスで指定する必要があります。

[*20] 1.7 節も参照してください。xs : ys = xst : yst となるように xst,yst を取ります。

例を挙げましょう。

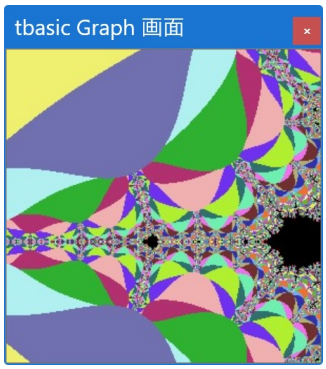
■画像サイズが小さくて，サイズが既知の場合

画像ファイル SSample.jpg があるとして，画像サイズが例えば，300 × 300 だったとします。このような場合は，画像の表示は簡単です[*21]。GScreen(300,300) でグラフ画面を開けば，そのサイズが，画像のサイズと同じですから，LoadPicture を行えば，画像は丁度画面に収まります。

以下がそのプログラムと実行結果です[*22]。

例 4.1 (画像サイズが既知の小さい画像の LoadPicture(LoadSSample.tbt)).

```
' 画像サイズが既知の小さい画像の LoadPicture
GScreen(300,300)
LoadPicture("SSample.jpg")
End
```



実行結果

[*21] 以下の例で使う SSample.jpg, MSample.jpg, LSample.jpg, HSample.jpg は 1.7 節で利用した"bigMandelbrot.png"をそれぞれ，300 × 300，400 × 300，2500 × 1500，4000 × 3000 のサイズにトリミングしたものです。

[*22] ここで画像ファイル SSample.jpg はカレントディレクトリにあるとします。

■画像サイズが小さくて，サイズが不明の場合

画像サイズは比較的小さいことは分かっていても，具体的には不明な場合，正確な画像サイズを知る必要があります。tbasic では，GetImageSize$で画像サイズを知ることができます。GetImageSize$は”x???:y???”の形の文字列で，”x???”，”y???” はそれぞれ，4 桁の数字で，幅，高さを返します。

このことから，

```
IMS$=GetImageSize$(F$)
xs =Val(Left$(IMS$,4))
ys =Val(Right$(IMS$,4))
```

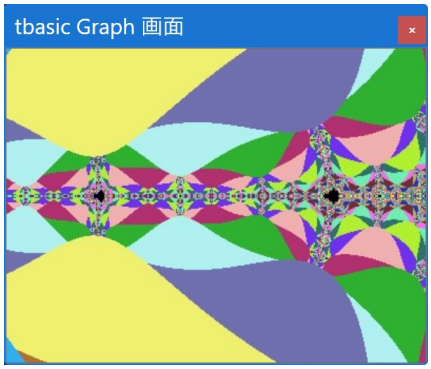
によって，F$の xs,ys を求めることができます。

次の例は，画像サイズが小さいことだけは分かっている MSample.jpg の画像サイズを求めて画面を開くものです。

例 4.2 (画像サイズが小さいことが分かっている画像の LoadPicture(LoadMSample.tbt)).

```
' 小さい画像の LoadPicture
F$ = "MSample.jpg"
IMS$=GetImageSize$(F$)
xs =Val(Left$(IMS$,4))
ys =Val(Right$(IMS$,4))

GScreen(xs,ys)
LoadPicture(F$)
End
```



実行結果

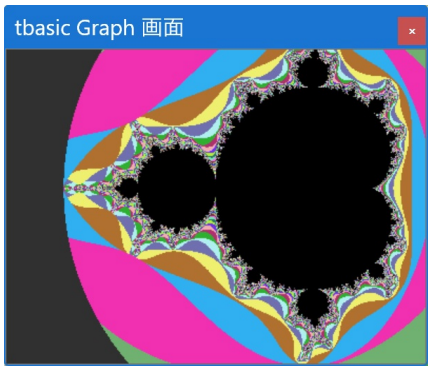
■画像サイズが大きくて，サイズが既知の場合

画像のサイズが大きい場合は，工夫が必要になります。画像ファイル HSample.jpg があるとして，画像サイズが例えば，4000 × 3000 だったとします。この画像を上のような方法で LoadPicture のために画面を開くと，大きすぎて，ディスプレイに収まりません。

このような場合は，拡張型の GScreen を使って，小さな画面に画像を縮小して読み込ませることができます。縮小は，GStretch コマンドを使います。次の例では，1/10 の画面に縮小して表示しています。

例 4.3 (画像サイズが既知の大きい画像の LoadPicture(LoadHSample.tbt)).

```
' 画像サイズが既知の大きな画像の LoadPicture
GScreen(4000,3000,400,300)
GStretch ON
LoadPicture("HSample.jpg")
End
```



実行結果

■画像サイズが大きくて，サイズが不明の場合

画像サイズが大きいことは分かっていても，具体的は分かっていない場合は，GetImageSize$を使って，同様なことが可能ですが，どのくらいの縮小率を選ぶかが問題です。ここでは簡単のため 1/5 として例を挙げます。

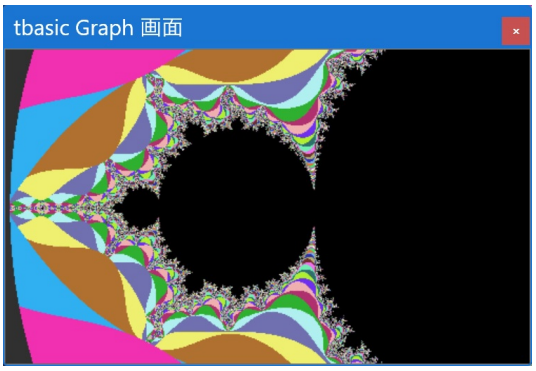
例 4.4 (大きな画像の LoadPicture(LoadLSampe.tbt)).

```
' 大きな画像の LoadPicture
' 拡張型の GScreen を開く

F$ = "LSample.jpg"
IMS$=GetImageSize$(F$)
xs =Val(Left$(IMS$,4))
ys =Val(Right$(IMS$,4))

' ここでは 1/5 の画面に表示する。
xst=xs/5
yst=ys/5

GScreen(xs,ys,xst,yst)
GStretch ON
' 小さい画面に縮小して全体を表示する。
LoadPicture(F$)
End
```



実行結果

■ SelectOpenFile の活用

今までの例は，画像ファイルがカレントディレクトリにある場合でした。画像がカレントディレクトリにない場合はそのファイルまでの Path(パス) を指定する必要があります。一つのファイルを手動で選択して，利用する場合は SelectOpenFile を利用すると簡単に Path を指定することができます。

SelectOpenFile は，

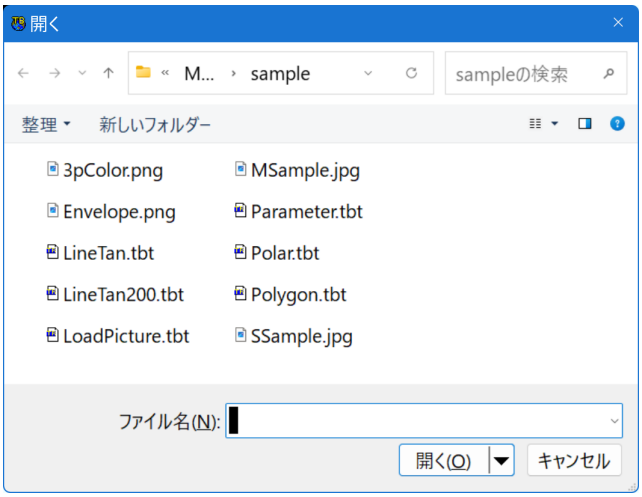
SelectOpenFile または，SelectOpenFile(ExtensionFilter)

の形で使い，Open ダイアログを使って，ファイルを選択したとき，選んだファイルの完全パスを返します。

次のプログラムは簡単なものですが，実行してみると SelectOpenFile の使い方が良く分かります。右の画像は SelectOpenFile の実行の例です。

例 4.5.

```
F$=SelectOpenFile
Print F$
```



SelectOpenFile を使って，上のすべての例に対応するプログラムを以下に挙げます。

例 4.6 (SelectOpenFile を利用した画像の LoadPicture(LoadImage.tbt)).

```
ChDir GetProgramDir
Const MaxScr = 400
GFName$=SelectOpenFile(" |*.BMP;*.JPG;*.PNG;*.GIF")
If Trim$(GFName$)<>"" then
   ISize$=GetImageSize$(GFName$)
   XSize=Val(Left$(ISize$,4))
   YSize=Val(Right$(ISize$,4))
   ScrSize = Max(XSize,YSize)
   If ScrSize <= MaxScr then
      GScreen(XSize,YSize)
   Else
      R = MaxScr/ScrSize
      XSt = XSize * R
      YSt = YSize * R
      GScreen(XSize,YSize,XSt,YSt)
      GStretch ON
   End If
   LoadPicture(GFName$)
End If
End
```

4.2 SavePicture

SavePicture は 1.6 でも説明しましたが，LoadPicture の逆で，グラフ画面の画像を画像ファイルとして保存するものです。

SavePicture(F$)

の形で使い，グラフ画面の画像を F$として保存します。保存画像は拡張子 bmp，png，gif，jpg で指定します。ファイル名のみだとカレントディレクトリに保存します。それ以外の場所に保存する場合はパスを含めて指定します。また，同名のファイルがあった場合は，上書きをします。

SavePicture(F$) は現在表示されているグラフ画面をそのまま保存するだけの単純なコマンドですが，パス指定や，上書き注意のが必要な場合は，対応するプログラムを追加する必要があります。

次の例は，既にグラフ画面が表示されている状況で，パスを SelectSaveFile を使って指定し，上書き確認を FileExists 関数を使って確認するものです。

SelectSaveFile は保存するファイルのパスを指定するのが主目的です。ダイアログでパスを選び，そこでファイル名を指定すると，指定したファイルのフルパスを返します。

例 4.7.

```
' 部分プログラム
' グラフ画面が既に表示されていること
GFName$=SelectSaveFile(" |*.BMP;*.JPG;*.PNG;*.GIF")
If Trim$(GFName$)<>"" then
   If FileExists(GFName$) then
      Print "同名のファイルが存在します"
   Else
      SavePicture(GFName$)
   End If
End If
End
```

4.3 ReSizeImage

ReSizeImage は現在グラフ画面に表示されている画像を縮小します[*23]。このコマンドは Ver.1.6 で新たにサポートされたものです。

ResizeImage(Ratio)

の形で使います。

現在携帯のカメラで撮影した画像は 4000 ピクセルを超えるものもあり，大きすぎることもあります，そのような場合に必要なサイズへと縮小するものです。

例えば，

ResizeImage(1/2)

を実行すると，現在のグラフ画面に表示されている画像が，縦横それぞれ 1/2 になり，面積は 1/4，保存した場合のファイルサイズは約 1/4 になります。

実際に，現在ある画像ファイルを小さいものしたい場合，

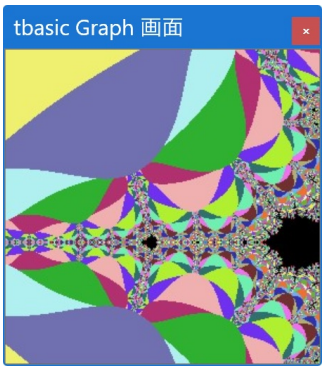
- Gscreen でグラフ画面を開く
- LoadPicture で画像を表示
- ResizeImage で画像を縮小
- SavePicture で画像を保存

で可能です。

次の例は例 4.1 の後に，ResizeImage(1/2) を追加したときのグラフ画面の様子です。

例 4.8.

```
'
GScreen(300,300)
LoadPicture("SSample.jpg")
ResizeImageg(1/2)
End
```



ResizeImage 処理前



ResizeImage 処理後

[*23] 拡大も可能ですが，使用は推奨しません。

4.4 RotateFlip

Ver.1.62 で新たにサポートされたものです。tbasic の RotateFlip は C#でサポートされているものと機能的にほぼ同じですが，指定方法は少し異なります。

RotateFlip(Angle)

または，

RotateFlip(Reflection)

の形で指定します。

- RotateFlip(Angle) は，現在のグラフ画面を Angle だけ時計回りに回転させます。Angle の取れる値は，90 度，180 度，270 度のみです。tbasic では，次のように文字列の形式で記述します。
  RotateFlip("90"), RotateFlip("180"), RotateFlip("270")
- RotateFlip(Reflection) は現在のグラフ画面を Reflection で指定された方法で反転させます。反転は x 軸，y 軸に関する反転のみです。
  RotateFlip("x") は画像を x 軸に関して反転させます（上下反転)。
  RotateFlip("y") は画像を y 軸に関して反転させます（左右反転)。

次の例は,元画像をグラフ画面に表示し,順次 RotateFlip("90"),RotateFlip("180"),RotateFlip("270"),RotateFlip("x"),RotateFlip("y") を施したグラフ画面の様子です。

画像サイズが 300×300 の画像 B300.png がカレントディレクトリにあったとしたとき，次のプログラムのコメントを適宜外すことにより，以下のようなグラフ画面が得られます。

例 4.9.

```
GScreen(300,300)
LoadPicture("B300.png")
'RotateFlip("90")
'RotateFlip("180")
'RotateFlip("270")
'RotateFlip("x")
'RotateFlip("y")
End
```



元画像



時計回り 90 度回転



時計回り 180 度回転



時計回り 270 度回転



上下反転



左右反転

5　グラフィック処理プログラム例（平面グラフの描画）

グラフィック画面には様々な図形を描くことができますが，ここでは最も基本的なものとして $y = f(x)$ の形の平面グラフを描くプログラムを説明しましょう。

グラフは色々な状況で描くことが考えられますが，ここではまず簡単のため具体的数値を使った状況で説明します。次に，それら色々な状況に対応できる汎用的なプログラムを挙げます。また媒介変数，$x = x(t)$，$y = y(t)$ で表されるグラフのプログラムも挙げます。

- 初期設定
- グラフを点で描く
- グラフを線で描く
- 線で描くのが難しい例
- 比較的汎用性のあるプログラム
- 媒介変数表示関数のグラフ

5.1　初期設定

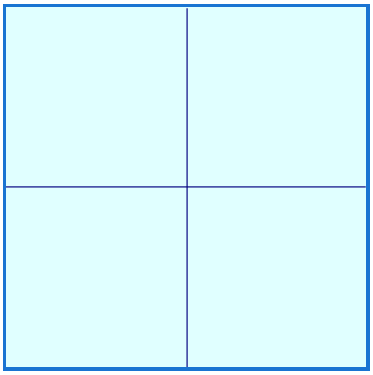
平面グラフを描く場合，標準的には次の設定が必要です。

- グラフ画面の大きさ：GScreen
  必要に応じて，使うグラフ画面の大きさを指定します。画面の縦横比は 1 : 1 や 2 : 1 など比較的単純なものが良いでしょう。
- 座標系の設定
  Window を使って座標設定をます。座標系の縦横比とグラフ画面の縦横比は普通は一致させます[*24]。
- 座標軸の描画
  グラフを見やすくするために，x 軸と y 軸を表示します。

ここでは，例として次の設定を使います。

例 5.1 (座標軸の設定).

```
' 座標軸の設定
GBackColor="LightCyan"
GScreen(300,300)
Window (-10,-10)-(10,10)
MathGraph On
GForeColor = "Navy"
Line (-10,0)-(10,0) :'x 軸
Line (0,-10)-(0,10) :'y 軸
GForeColor = "Black"
End
```



実行結果

[*24] 特殊なグラフを描く場合，座標系の敢えて縦横比を変えることはあります。

5.2　グラフを点で描く

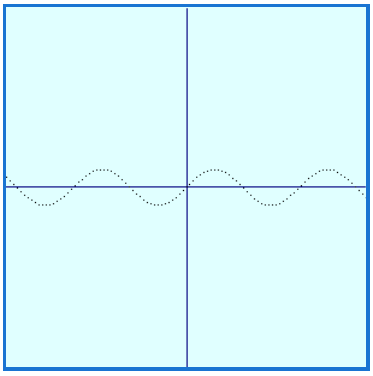
グラフィックの原理の項では，グラフ画面は点（pixcel）で構成されると説明しました。ですから，グラフを描くとき，点で描くのは基本的です。このことから，xy 座標平面上に $y = f(x)$ のグラフを描くには

> x を順次動かし，各 x で，$(x, f(x))$ に点を表示する

ことが考えられます。まず，この方法で，$y = \sin(x)$ のグラフを描いてみましょう。点の表示には PSet を使います。

例 5.2 ($\sin(x)$ のグラフ（点て描く）).

```
GBackColor="LightCyan"
GScreen(300,300)
Window (-10,-10)-(10,10)
MathGraph On
GForeColor = "Navy"
Line (-10,0)-(10,0) :'x 軸
Line (0,-10)-(0,10) :'y 軸
GForeColor = "Black"
For i=0 To 100
   x = -10 + 2*i/10
   y = Sin(x)
   PSet(x,y)
Next i
End
```

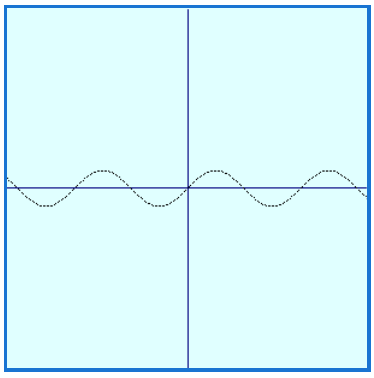


実行結果

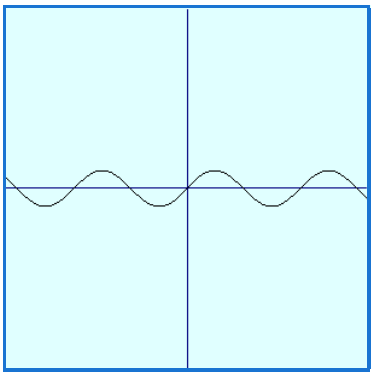
ここでは 101 個の点を描いています。$x = -10$ から 10 までの間を 100 等分し，その等分点の x 座標を $x = -10 + 2 * i/10$ で求めます。その各点について $y = \sin(x)$ を計算し，その座標 (x, y) に点を表示します。

この結果を見ると，表示されたグラフは連続していません。実際，点は離散的なものですから，多くの点を並べなければ連続した線を表すことはできません[*25]。

これを連続したグラフにする方法としては，更に点の個数を増やすことが考えられます。次は点の個数を 201 個，301 個にしたときの結果です。



201 個の点



301 個の点

[*25] もちろん，数学的には点をいくら並べても線にはなりません。グラフ画面が有限個の点で表されているという状況の下での話です。

201 個では，連続にはなりませんが，300 個にすると，連続した線になりました。もともとこのグラフィック画面には横 300 個の点しかありませんから，これ以上点の個数を増やしても（この場合は[*26]）余り意味はありません。まとめると，

> 点でグラフを描く場合，画面の横軸の画素数以上の点で描く必要がある。

となります。

5.3 グラフを線で描く

連続したグラフにするもう一つの方法として，点を線で結ぶ方法が考えられます。そこで上のプログラムを点を線で結ぶように書き直してみましょう。線分を使って描くには，

> x_1, x_2 を動かし，$(x_1, f(x_1))$ と $(x_2, f(x_2))$ を結ぶ線分を描いていく。

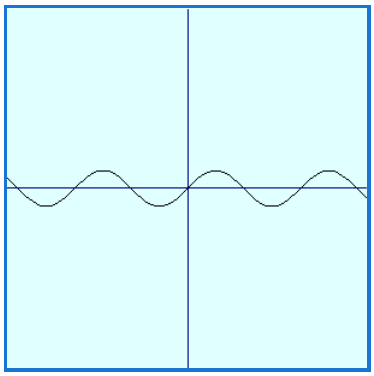
とします。

ここでは PSet の代わりに Line を使います。

次のプログラムは 100 本の線分を使うものにしたものです。

例 5.3 ($\sin(x)$ のグラフ（線て描く）(LineSin.tbt)).

```
GBackColor = "LightCyan"
GScreen(300,300)
Window (-10,-10)-(10,10)
MathGraph On
GForeColor = "Navy"
Line (-10,0)-(10,0)
Line (0,-10)-(0,10)
GForeColor = "Black"
For i=0 to 100-1
   x1 = -10 + 2*i/10
   y1 = Sin(x1)
   x2 = -10 + 2*(i+1)/10
   y2 = Sin(x2)
   Line (x1,y1)-(x2,y2)
Next i
End
```



実行結果

$x = -10$ から $x = 10$ までの長さ 20 の線分を 100 等分した長さ $20/100 = 2/10$ の各線分の両端

$i = 0$ のとき，$(x_1, x_2) = (-10, -10 + 2/10)$

...

$i = 100 - 1$ のとき，$(x_1, x_2) = (-10 + 2 * 99/10, -10 + 2 * 100/10) = (10 - 2/10, 20)$

を Line で結んでいます。

この結果から，100 本の線分で描くと，300 個の点で描いたのと殆ど同じ結果が得られることが分かります。

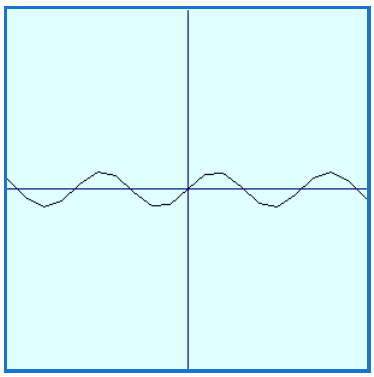
[*26] $\pi/2$ の近くでの $\tan(x)$ のグラフのように値が大きく変化する場合はそうではありません。

もう少し，線分の個数を減らして例えば，20 本，50 本で描いてみましょう。これらは，上のプログラムで For 文を次のように替えたものです。

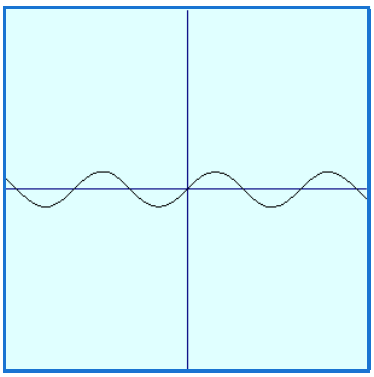
```
'20 本の線
For i=0 to 20-1
   x1 = -10 + 2*i/2
   y1 = Sin(x1)
   x2 = -10 + 2*(i+1)/2
   y2 = Sin(x2)
   Line (x1,y1)-(x2,y2)
Next i
```

```
'50 本の線
For i=0 to 50-1
   x1 = -10 + 2*i/5
   y1 = Sin(x1)
   x2 = -10 + 2*(i+1)/5
   y2 = Sin(x2)
   Line (x1,y1)-(x2,y2)
Next i
```

結果は次の通りです。



20 本の線



50 本の線

20 本の線分で描いたものはギザギザしていますが，50 本の線分でのものは滑らかに見えます。実際，画像を拡大して比較してみると分かりますが，50 本の線分で描いたのと，300 個の点で描いたものは殆ど同じです。

これらのことから次のことが言えます。

> グラフは点で描くよりも線分で描いた方が効率がよい。

しかし，線分でグラフを描く場合，注意することがあります。それは

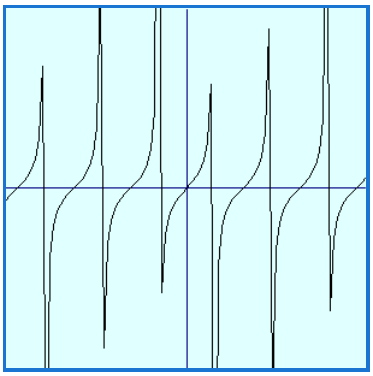
- 連続でないグラフは不連続な部分が正しく描けない。
- 急激な変化をするグラフでは使う線分の個数を増やす必要がある

ということです。

線を使って描くのが難しい例を挙げましょう。次は $\tan(x)$ を上と同様に 100 本の線分で描いたのが次です。

例 5.4 ($\tan(x)$ のグラフ（線て描く）(LineTan.tbt)).

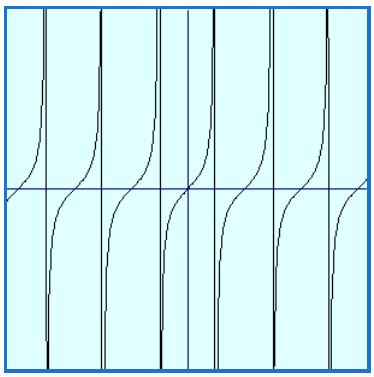
```
GBackColor = "LightCyan"
GScreen(300,300)
Window (-10,-10)-(10,10)
MathGraph On
GForeColor = "Navy"
Line (-10,0)-(10,0)
Line (0,-10)-(0,10)
GForeColor = "Black"
For i=0 To 100-1
   x1 = -10 + 2*i/10
   y1 = Tan(x1)
   x2 = -10 + 2*(i+1)/10
   y2 = Tan(x2)
   Line (x1,y1)-(x2,y2)
Next i
End
```



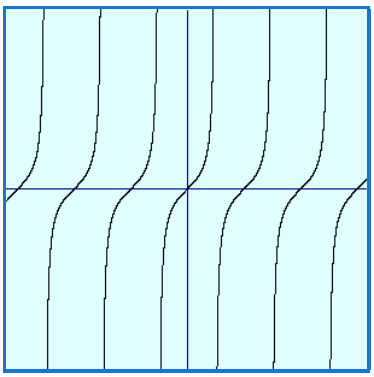
実行結果

$\tan(x)$ は　$\ldots, -\pi/2, \pi/2, 3\pi/2, \ldots$ で不連続で，その付近で急激に増減します。この状況は 100 個の線分では表しきれません。また，線分を結ぶ方法では，不連続点を表せませんから，線分の個数を増やしても，不連続点では正しくはありません。個々の関数の性質が分かっている場合には，その部分をプログラムの中で処理をすれば正しい概形を描くことはできますが，汎用性はありません。

このような場合，多くの点で描くと，かなり正確なグラフを描くことができます。この方法は汎用性がありますが，描画に時間はかかります。以下の左は線分を 200 取ったときの，グラフです。右は点を 40001 個取って描いたグラフです。正確なグラフが得られますが，描画時間は 100 個の点を取ったときの 400 倍かかります[*27]。



200 本の線



40001 個の点

このように，グラフの描画は描く関数に依存しますので，描きながら微調整をする必要があります。

> 不連続点があったり，急激に変化する関数のグラフは，
> 点で描く方が良いこともある。

と言えます。

[*27] 最近の PC はかなり高速で，描画にそれ程時間がかかるわけではありません。ですから場合に依っては試す価値があります。

5.4 比較的汎用性のあるプログラム

前節までのプログラムでいくつかのグラフを描いてきました。ここではそれらを基に，色々な状況で使える汎用的なプログラムを作ってみましょう。

次の前提で作ることにします。

- 線分を使って描くことにし，不連続点のあるグラフでの不連続な部分の不具合は仕方ないものとします。
- 描画範囲内での関数の計算エラー[*28]はないものとします。
- グラフ画面の大きさ，座標，描く関数，使う線分の個数を設定してグラフが描けるものにします。
- これらのものを設定するために，変数とユーザー定義関数を使います[*29]。

汎用的なプログラムと言っても，個々のグラフの描画では色々な状況が異なりますから，当然，それに対応してプログラムの修正は必要です[*30]。その際，できるだけ簡単に修正できるように予めプログラムを書いて置くことが望まれます。

このようなプログラムを書く方法は色々あると思われますが，よく使われる原則は次です。

修正しやすいプログラムの原則

・具体的数値や数式をできるだけ使わない。
・具体的数値，数式の設定はまとめる。

この原則に従って，具体的数値の使用を避け，変数名で記述し，その変数の設定をまとめます。

・グラフ画面の大きさ
　GWW：グラフィック画面の幅
　GWH：グラフィック画面の高さ
・座標
　XLEFT：左端の x 座標
　XRIGHT：右端の x 座標
　YBOTTOM：下端の y 座標
　（YTOP：上端の y 座標はグラフ画面の縦横比から自動設定）
・描く関数
　Function f(x)
・使う線分の個数
　NLINES

この設定で上のプログラムを書き直したものが次です。少しの書き換えで，汎用的なものになります。

[*28] ゼロに依る割り算や，負数の平方根など。
[*29] ユーザー定義関数は「構造化プログラミング」の項で説明するものですが，少し先取りして使用します。
[*30] プログラムの修正ではなく，プログラムを動かして，その中で色々設定する方法もありますが，ここでは，より簡単なプログラム本体を修正する方法を取ります。

例 5.5 (平面グラフ描画プログラム (Explicit.tbt)).

```
' グラフの描画
'------------設定ここから-----------
GWW = 300
GWH = 300
XLEFT = -10
XRIGHT = 10
YBOTTOM = -10
NLINES = 100

Function f(x)
   f = Sin(x)
End Function
'------------設定ここまで-----------
GBackColor = "LightCyan"
GScreen(GWW,GWH)
YTOP = GWH*(XRIGHT-XLEFT)/GWW + YBOTTOM
Window (XLEFT,YBOTTOM)-(XRIGHT,YTOP)
MathGraph On
GForeColor = "Navy"
Line (XRIGHT,0)-(XLEFT,0)
Line (0,YBOTTOM)-(0,YTOP)
GForeColor = "Black"
For i=0 To NLINES-1
   x1 = XLEFT + (XRIGHT-XLEFT)*i/NLINES
   y1 = f(x1)
   x2 = XLEFT + (XRIGHT-XLEFT)*(i+1)/NLINES
   y2 = f(x2)
   Line (x1,y1)-(x2,y2)
Next i
End
'-----------------------------------
```

このプログラムでは，[--設定ここまで--] 以下は，殆ど具体的数値や数式がないことに注意してください。

[設定] の部分を定義変更することで，画面の大きさ，座標系，描画する関数を替えることができます。関数の記述は，Function の f = の右辺を変更すれば可能です[*31]。

5.5 媒介変数や極座標表示関数のグラフ

前節までのプログラムを少し変更すると，媒介変数や極座標表示関数のグラフ用のプログラムになります。

5.5.1 媒介変数

媒介変数 t を使い，その始点と終点を指定します。また，関数は，$f(x)$ ではなく，$x(t)$，$y(t)$ を指定します。

- 媒介変数の動く範囲
  TSTATRT：t の始点
  TEND：t の終点
- 描く関数
  Function x(t)
  Function y(t)

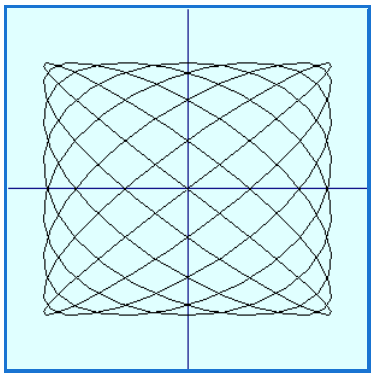
[*31] $f(x) = \sin(x)$ のように設定してはいけません。$f = \sin(x)$ です。

例 5.6 (媒介変数平面グラフ描画プログラム (Parameter.tbt)).

```
' 媒介変数によるグラフの描画
'------------設定ここから-----------
GWW = 300
GWH = 300
XLEFT = -10
XRIGHT = 10
YBOTTOM = -10
TSTART = 0
TEND = 2*Pi
NLINES = 200

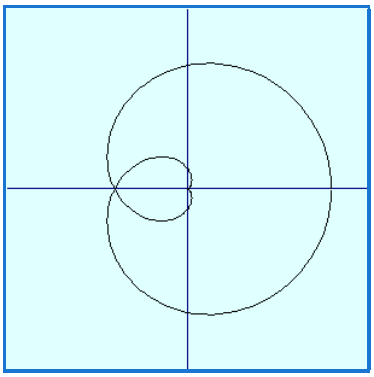
Function x(t)
   x = 8*Sin(4*t)
End Function
Function y(t)
   y = 5*Sin(5*t)
End Function
'------------設定ここまで-----------
BackColor = "LightCyan"
GScreen(GWW,GWH)
YTOP = GWH*(XRIGHT-XLEFT)/GWW + YBOTTOM
Window (XLEFT,YBOTTOM)-(XRIGHT,YTOP)
MathGraph On
ForeColor = "Navy"
Line (XRIGHT,0)-(XLEFT,0)
Line (0,YBOTTOM)-(0,YTOP)
ForeColor = "Black"
For i=0 To NLINES-1
   t1 = TSTART + (TEND-TSTART)*i/NLINES
   x1 = x(t1)
   y1 = y(t1)
   t2 = TSTART + (TEND-TSTART)*(i+1)/NLINES
   x2 = x(t2)
   y2 = y(t2)
   Line (x1,y1)-(x2,y2)
Next i
End
'-----------------------------------
```

このプログラムを使って描いた例を２つ挙げます。`Function x(t)`, `Function y(t)` を変更しただけです。



$x = 8\sin(8t)$

$y = 7\sin(7t)$



$x = 4 * (1 + \cos(t)) * \cos(2t)$

$y = 4 * (1 + \cos(t)) * \sin(2t)$

5.5.2　極座標

極方程式 $r = f(\theta)$ で表されたグラフは，媒介変数のプログラムから簡単にできます。

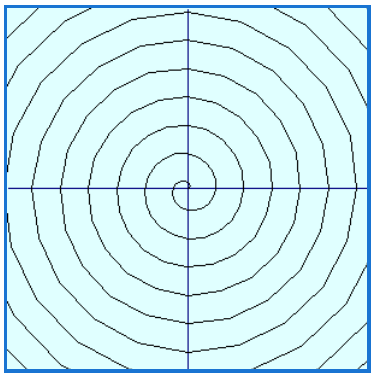
$r = \sqrt{x^2 + y^2}$ ですから，$t = \theta$ として，$x = f(t)\cos(t)$，$y = f(t)\sin(t)$ のグラフを描けばよいことになります。このことから，極方程式で表されたグラフの描画は，媒介変数のプログラムの以下の部分の変更で可能になります。

```
Function r(s)
   r=5*(1+Cos(s))
End Function

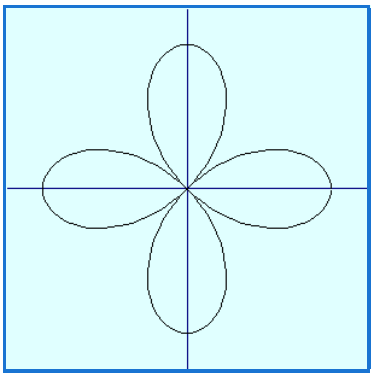
Function x(t)
   x=r(t)*Cos(t)
End Function

Function y(t)
   y=r(t)*Sin(t)
End Function
```

このプログラムを使って描いた例を２つ挙げます。`Function r(s)` と `t` の変域を変更しただけです。



$r = s/4$, `TEND = 20*Pi`



$r = 8 * \cos(2s)$

このように関数の値の式が具体的に表された関数（陽関数）のグラフの xy 平面上への描画は簡単に可能です。

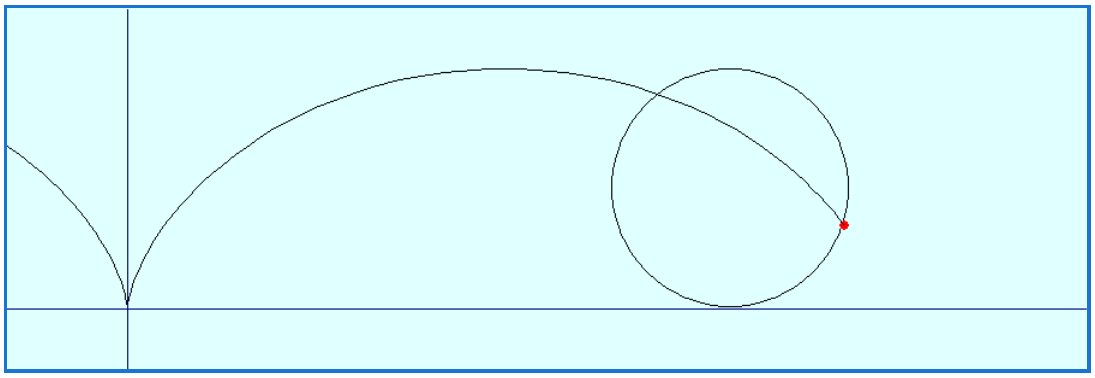
5.5.3　軌跡の描画

数学に軌跡の問題というものがあります。これはある条件を満たす点の集合を記述するものです。平面幾何での軌跡は高校数学で良く扱われます。コンピュータグラフィックで軌跡を描くのは，条件となっている式を上手に表現できれば，描画そのものは難しくありません。ここではそのいくつかについて調べてみましょう。

例 5.7 (サイクロイド). サイクロイドは円が滑らずに回転するとき，円周上の定点が描く図形です。この図形は，半径 1 の円の場合，媒介変数 t を使って，

$$x = t - \sin(t), \qquad y = 1 - \cos(t)$$

と表されることが知られています。このことを使えば，サイクロイドの描画は媒介変数の描画の一例です。前出の媒介変数表示プログラムの式と数値を変えるだけで描画することができます。ここでは，少し，画面を横長にして，回転する円も描いてみると次のようになります。



サイクロイドを動的に描画するプログラムとして Samples の中に，Cycloid.tbt があります。

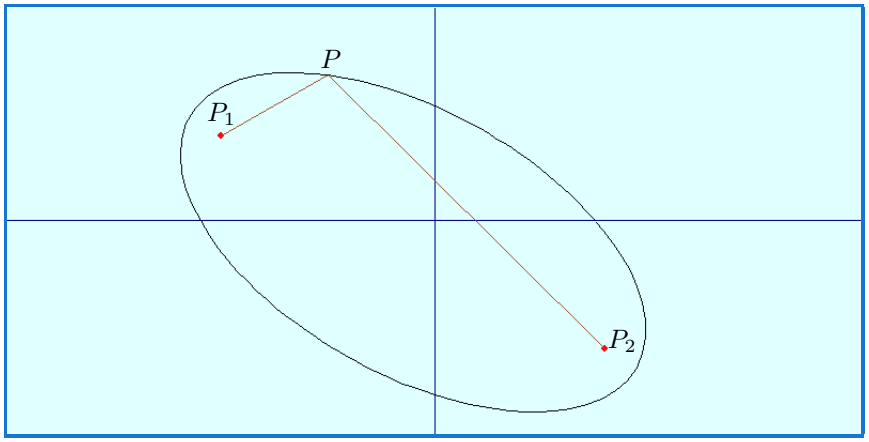
例 5.8 (楕円). 楕円は，２点からの距離の和が一定 C な図形として定められています。この，２点からの距離の和が一定となる点を記述できれば，軌跡が描画できます。これは，例えば，次の Sub によって実現できます。

```
Sub GetXY(s,x,y) ' 角度 s のとき，P0P+P1P=C となる P=(x,y) を返す
   L=0: r=0
   While L<C
      x = r*cos(s)
      y = r*sin(s)
      L= sqr((x-XX0)^2+(y-YY0)^2)+sqr((x-XX1)^2+(y-YY1)^2)
      r=r+1/100
   Wend
End Sub
```

これは中心を通る角度 s の直線上を中心から点を移動させて，C を超えた点を返します。この Sub は

```
適当な初期設定
Public XX0,YY0,XX1,YY1' P0=(XX0,YY0), P1=(XX1,YY1)
・・・
Call GetXY(0,x0,y0)
For i=1 To 100
   r=i*2*Pi/100
   Call GetXY(r,x,y)
   Line (x0,y0)-(x,y)
   x0=x: y0=y
Next i
End
```

のように使います。これを使って，$P_1=(-5,2)$, $P_2=(4,-3)$, $C=12$ として描いた楕円が次です。



5.5.4　物体の運動

アニメーションなど物体の運動は，幾何学の軌跡と同様に，運動による位置の式が得られれば，描いたものを消して，次を描くことで実現できます。またその運動の軌跡は点または，小さな円を使って描くことができます。ここではそのいくつかの例を挙げましょう。

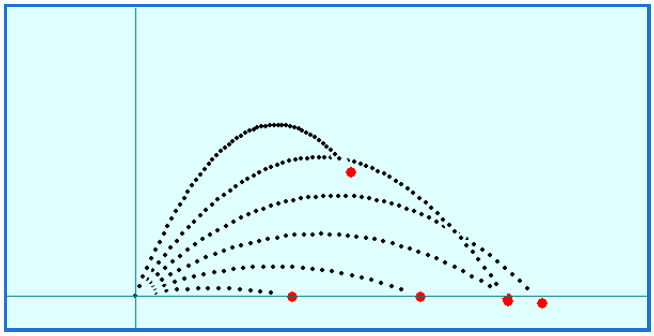
例 5.9 (放物体の運動).

物体を投射したときの角度を s とすると，時間 t の後の位置は，適当な変換をすると，

$$x(t) = v * t * \cos(s),$$
$$y(t) = v * t * \sin(s) - t * t$$

と表されることが知られています。ここで，v は投射速度です。

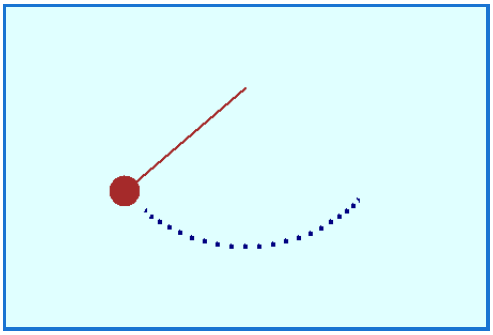
このことを使うと物体の放物運動を描画することができます。Samples にある Parabola.tbt は，$i = 1, 2, \cdots, 8$ に対して $i \cdot \pi/16$ の角度で物体を投射し，その運動とその軌跡を描くものです。下の図はそれを $i = 5$ の途中で止めたときのものです。$i = 4(45°)$ のときが最も遠くに飛ぶ状況が分かります。



例 5.10. 振り子の運動は振り子を吊り下げた点から降ろした垂線と時刻 t での振り子を吊り下げた糸との成す角度を θ とすると，適当な変換をすると，

$$\theta = \theta_0 \cos(t)$$

と表されることが知られています。ここで，θ_0 は最大振幅角度です。Samples にある Pendulum.tbt はこの式を使って，振り子の運動を描画するものです。実際の振り子の動きに近いものが得られます。下の図は最大振幅角度 50° の場合の振り子が運動している様子です。



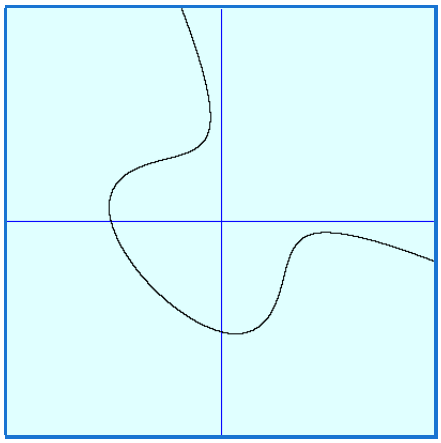
物体の運動の描画も運動の式が分かれば，描画そのものは難しくありません。但し，動く様子を見やすく表示するのは，工夫が必要です。

5.5.5　陰関数の描画

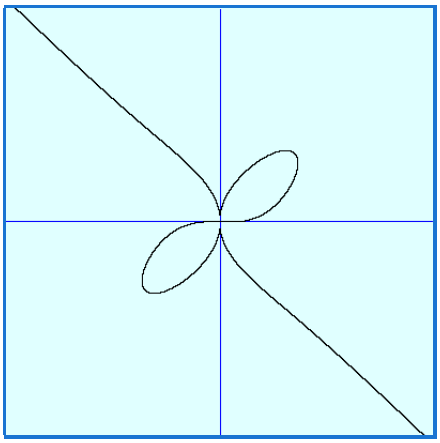
最後に陰関数の描画について考えてみましょう。陰関数のグラフを正確に描画するのは難しい問題で，色々な方法が提案されています。

一方素朴な方法による描画でも多くの関数について，比較的正確な概形を書くことはできます。Samples にある ImplicitPlot.tbt は素朴な原理で，陰関数 $f(x,y)=0$ のグラフを描画します。その原理は，「小さな正方形の４隅の点 (x,y) での $f(x,y)$ の符号が異なるものがあれば，その正方形の中点をプロットする。」というものです。この方法では，抜け落ちる点があることがありますが，多くの自然な関数では，比較的正確な図を描くことができます。

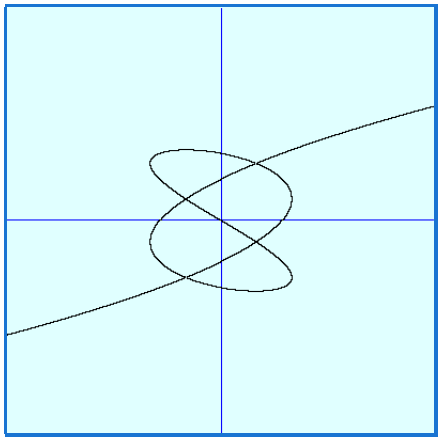
以下は，ImplicitPlot.tbt を使って描画した例です。



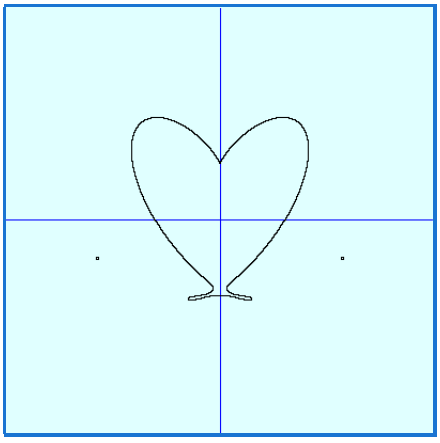
$f(x,y)=(x-y)^2-(x+y)^3+2(x+y)-3$



$f(x,y)=x^5-2x^2y+y^5$



$f(x,y)=4x^3-3x-(16y^5-20y^3+5y)$



(*)

(*):
$f_1(x,y)=(93392896/15625)x^6+((94359552/625)y^2+(91521024/625)y-(249088/125))x^4$
$f_2(x,y)=((1032192/25)y^4-36864y^3-(7732224/25)y^2-(207360)y+(770048/25))x^2$
$f_3(x,y)=65536y^6+49152y^5-135168y^4-72704y^3+101376y^2+27648y-27648$
$f(x,y)=f_1(x,y)+f_2(x,y)+f_3(x,y)-100$

5.6 演習問題

ここまでは tbasic でのプログラミングでの基本的事項，特にこの文書ではグラフィックについて説明してきました。その纏めとして，演習問題を挙げます。基本的には，ここまでの説明でできるものです。以下のプログラムを作りましょう。

(1) 以下の矢印を表示するプログラム。



(2) 角度と拡大率を入力したとき，上の矢印がその角度までゆっくり回転しながら，その拡大率まで拡大していくプログラム。

(3) 図 1 のように長さ 1 の棒 AB の端点 A を y 軸上の点 $(0,1)$ に置き，B を原点 $(0,0)$ に置きます。ここで，B を x 軸上に置くようにしながら，A を y 軸上をずらし，原点まで移動させたときにできる直線族によって浮き出る曲線をアステロイドと言います。この直線の y 軸，x 軸での交点をそれぞれ，$(0,y),(x,0)$ とすると，$x^2+y^2=1$ となりますから，この曲線族の描画は，`Line (0,y)-(sqr(1-y^2),0)` で，y を適宜動かして描けば良いことになります。このとき，第 1 象限だけでなく，第 2，3，4 象限も同様にしてアステロイド全体を描くプログラム。

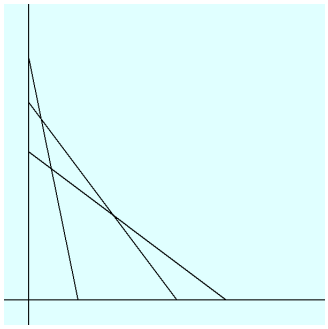


図 1

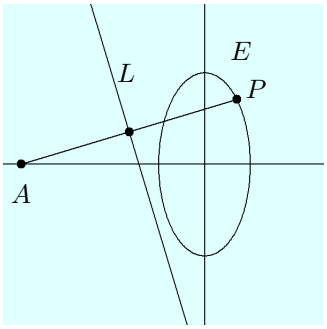


図 2

(4) 図 2 のように楕円 $E:4x^2+y^2=1$ と点 $A(-2,0)$ があります。A と E 上の点 $P(x,y)$ を結んだ線分 AP の垂直 2 等分線を L とします。このとき，P が E 上を動くときの L からなる直線族を描画するプログラム。

「Tiny Basic for Windows グラフィック操作編」更新記録

- （2026 年 10 月版）「高解像度画像の利用」,「活用コマンド」の項を追加。Ver.1.64 用に全体を微調整。プログラムの文書への添付方式を廃止し，Samples への同梱方式へ移行。
- （2020 年 07 月版）pdf 文書の作成方法を変更。色の名前による指定の部分 Ver.1.5 用に追加。グラフィック画面の基本背景色を白系に変更，それに従う画面差し替え。「軌跡の描画」以降の文章を追加。演習問題を追加。それ以外の文書内容は 2014 年 11 月版とほぼ同じ。
- （2014 年 11 月版）初版公開