図を作成してTEXのレポートの質を上げよう

芝浦工業大学　数理科学研究会

～METAPOST, emath での図の作成～
ver.18 (METAPOST のみ)

2011 年 2 月 1 日 作成　　2011 年 2 月 26 日 完成　　2011 年 3 月 4 日 発表

作成開始日: ’11 年 1 月 31 日　　補訂：’11 年 2 月　4 日 補訂：’11 年 2 月　6 日 補訂：’11 年 2 月　8 日
全面改訂：’11 年 2 月 12 日 補訂：’11 年 2 月 14 日 補訂：’11 年 2 月 18 日 改訂：’11 年 2 月 19 日
補訂：’11 年 2 月 20 日 補訂：’11 年 2 月 21 日 補訂：’11 年 2 月 22 日 補訂：’11 年 2 月 23 日
補訂：’11 年 2 月 24 日 補訂：’11 年 2 月 25 日 補訂：’11 年 2 月 26 日 補訂：’11 年 3 月　1 日
補訂：’11 年 3 月　2 日 補訂：’11 年 3 月　3 日

作成：太田悠暉

はじめに

テストや課題・レポートなどで図を用いながら詳しく説明するのは, 基本的には重要である. しかし, 手書きでは楽でも TEX で図を作画するのは難しく, また大変時間を要する. そこで, 筆者が METAPOST, emath を用いて作画したもののうち, 理解した内容を記載する. そのため, もしかしたら分かりづらい説明になっている可能性があることがないとは言い切れない. これを見て実行させる際, 分からない・説明不足である部分があれば質問してもらって構わないが, 勘違い・不具合・OS が異なる等での質問は一切受け付けないことを強調しておく. よく考えてから質問に来てもらいたい. ただ与えられたものを当たり前のようにこなすことより, 自分にできることを自覚し, また自分にできないことを自覚することがまず必要である. そしてできることを 1 つでも増やしていこうとしていくことが重要ではないだろうか. これを読む人がこのようなことを理解した上で実践してくれることを望み, 願うばかりである.

目 次

第 1 章
METAPOST の場合

1 METAPOST

1.1 METAPOST とは？

METAPOST とはグラフや図を PostScript 形式で出力する METAFONT のようなプログラムである．METAPOST の特徴は, 直接図を描くのではなく図を作成するプログラムを書くことにある．そのため, 少しでもプログラムを書ける人ならば使える．プログラムであるため, 図を微調整することが可能で, 実例としては校章を作成 (半分を作り鏡映しにして完成) された方がいる．あまり難しく考えずに取り組んでもらいたい．

1.2 METAPOST の使い方

先取りになるが, METAPOST について少しだけ踏み込んで説明しよう．まず, テキストエディタ（例えばメモ帳）で“ ○○.mp ”というファイルを作成する．テキストエディタには例えば次のように記述する．

```
beginfig(1);
  u=30;
  draw (u,0)--(2u,u)--(u,2u)--(0,u);
endfig;
beginfig(2);
  u=30;
  draw (u,0)--(2u,u)--(u,2u)--(0,u)--cycle;
endfig;
end.
```

1 つ目は u=30bp(1bp(ビックポイント)=1/72 インチ , 1 インチ=2.54cm だから 30bp はおよそ 0.15mm) として, 4 点 (u,0),(2u,u),(u,2u),(0,u) を順に線分で結ぶことを表わしている．2 つ目は 4 点 (u,0),(2u,u),(u,2u),(0,u),(u,0) の順に線分で結び, 閉路をつくることを表わしている．これを実際に出力するには, コマンドプロンプトで『mpost ○○』と打ち込むことにより作成される○○.1 という EPS ファイルを includegraphics を用いて TEX に取り込めばよい[1] ．拡張子の .mp は書いても良いが, 省略して構わない．(後述するかもしれないが, フォントを埋め込んだ EPS ファイルを作成する場合には, コマンドプロンプトで『jmpost -tex=platex ○○』と jmpost コマンドでオプション『-tex=platex』を指定する必要がある) GSveiw 等がない場合はそのままでは表示されないので, さらに PDF に変換しないと見られないだろうが説明は省略する．

ちなみに beginfig(1) と beginfig(2) には違う内容を書き込むことができ, beginfig(1); から最初の endfig; までに描かれた内容が○○.1, beginfig(2); から 2 番目の endfig; までに描かれた内容が○○.2 という EPS ファイルに出力される．また, beginfig(-1) のように負の値を用いた場合は, ○○.ps というファイルになる．これらを TEX に取り込むには, 例えば次のようにする．

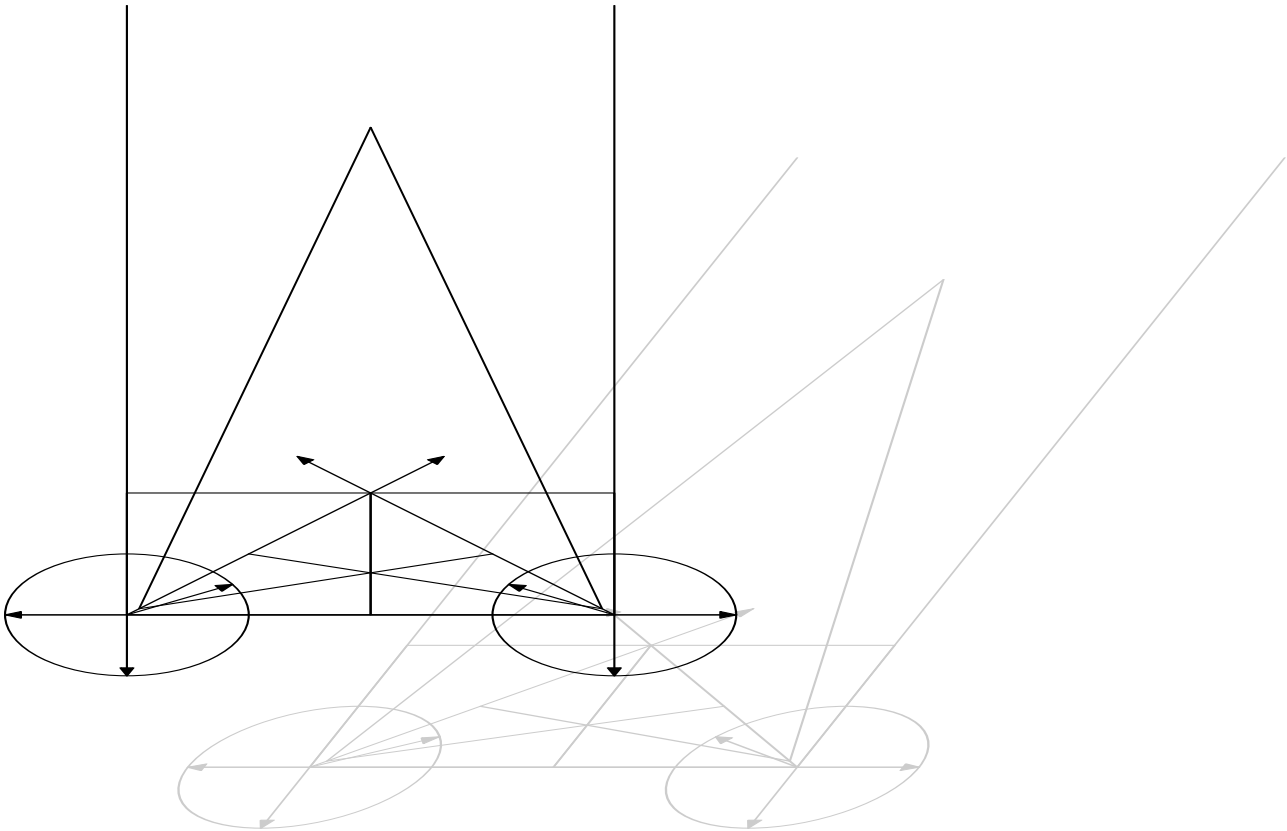
```
\documentclass{jarticle}
\usepackage[dvips]{graphicx}
\begin{document}
\includegraphics[width=5cm]{○○.1}
\includegraphics[width=5cm]{○○.ps}
\end{document}
```

[1] 参考にしたサイト [3] では TEX のソース中で記述すると METAPOST を動かして図を作成し, TEX 文書に貼り付けまでを行う METAPOST を拡張した MEPOTEX というパッケージが紹介されている．そちらから始めるのもよいだろう

1.3 EPS とは？

EPS とは Encapsulated PostScript(カプセル化されたポストスクリプト) の略称で, 印刷業界で広く使われている 1 つの図だけを含む限定された PostScript の画像形式であり, 拡大してもギザギザにならないベクトルグラフィックが特徴である. EPS ファイルは EPSF とも書く.

すると, 「PostScript とは何か？」と疑問になるかもしれないが, 筆者にはよく分からなかったので自分で理解してほしい.



2 METAPOST に必要な知識

2.1 文・マクロ・変数・タイプ

ここでは, METAPOST ではどのような「文」や「変数」「マクロ」などが許されているのかを知ってもらおうと思う. さまざまな図を作成するためには基礎的な知識が必要である. この内容を理解し, 必要な部分を各自で補えばどんな図でも描けるようになる.

1. 文

   METAPOST はおおむね C 言語のように扱われていると筆者は感じる. TEX とは異なることに注意してもらいたい. METAPOST における「文」とは, 1 つの引数をもつコマンド (引数を持たないコマンドもある) または数式のみからなる基本単位であることが特徴的である. TEX では文字列 (トークン) を処理するために, 文字や命令を次々と繋げて記述するが, METAPOST では 1 つの「文」にはひとつながりの命令か数式のどちらかしか書けない. すなわち, 1 つの「文」で 1 つの処理しかできないことになる. 1「文」は, 直前の ; (またはファイルの先頭) から ; (またはファイルの末尾) までと認識される. 例えば,

   ```
   a=2; b=4;
   ```

   のように 1 行に書かれていても 2「文」とみなされ,

   ```
   pickup pencircle
          scaled 5pt;
   ```

   のように複数行にわたって書かれていても 1「文」とみなされることになる. 実行したときにエラーが発生した場合は ; のつけ忘れの可能性が高いので注意深くチェックすることが必要だろう.

2. マクロ

   TEX と同様に METAPOST でもマクロを作成することができる. そのため一度マクロを作成すれば, どんな複雑な“ もの ”でもマクロを呼び出すだけでよくなり大変便利である. マクロに対して, もともと METAPOST に備わっている命令はプリミティブという. マクロはこのプリミティブを組み合わせる, あるいはプリミティブと他のマクロを組み合わせることにより作られる. 色々とマクロをソースファイル内で定義しても良いのだが, それではプログラミングの良さが生かされるとは言えない. マクロの定義を別のファイルして `input` 命令でソースファイルに読み込むことにしたほうが利口である. マクロの定義を保存するファイルの名前に制約はないが, 拡張子を `.mp` にすると `input` 命令での拡張子が省略できる. 例えば `macro1.mp` というファイルを METAPOST のソースに読み込むときは,

   ```
   input macro1;
   ```

   とすればよい. マクロについてこれ以上説明する事は避け, 5 **マクロについて**(23 ページ) において説明する.

3. 変数・タイプ

   METAPOST で使える変数は `numeric`(数値), `pair`(点・ベクトル), `path`(経路), `pen`(ペン先の形状), `picture`(画像), `boolean`(論理値), `transform`(変形), `string`(文字列), `color`(色) の 9 つのタイプが主である.

   これらのタイプの変数を使うためにはまず「宣言」が必要である. 宣言方法は

   ```
   <タイプ名> <変数名>;
   ```

   である. 例えば `pair` 変数を宣言するには

   ```
   pair a;
   ```

   とすればよい. 同じタイプの変数は同時に宣言でき

   ```
   pair t,e,x;
   ```

   のように書くことにより行える. 特に宣言を行わずに `<変数名>` を使用すると, 数値タイプと認識されるので必ず `<タイプ名>`を書くことをすすめる.

   `<変数名>` には, t1 のように数値の<添字>を付けることも, t1s のように数値と文字の<サフィックス>を付けることも, t.a のように文字だけの<サフィックス>を付けることも可能である. これらは次のように宣言する.

   ```
   pair t[], t[]s, t.a;
   ```

このように数値の部分は [] に代える必要がある. その代わり, pair t[]；と宣言したなら t1, t2, t3, ··· がすべて宣言できることになるので便利ではある.

数値の<添字>は [<数値式>] としても構わない. すなわち t[1+2] は t[3] と, t[1] は t1 と同じ意味になる. そのため, t[s] と ts と t.s は違う意味になる. また, t1a と t1.a と t[1]a と t[1].a は同じ意味であることは理解できるだろう. ちなみに, t.1 は t1 を意味するのではなく, t[0.1] を意味することに注意が必要である.

2.2 変数について 1 (numeric, pair, path, color)

ここでは, 変数のうち numeric, pair, path, color の 4 つについて説明する. この 4 つは変数の中でもよく使われるものなので, 最低限として次の内容は理解したほうが良いと思われる.

2.2.1 numeric

numeric は数値データを扱うことができるタイプである. 以下 numeric 変数を数値と呼ぶこともある. ただし, 扱える数値の桁数は少なく, 最小単位を $\frac{1}{65536}$ とする絶対値が 4096 未満の実数値に限定されている. よく使う定数として, 最小の正数 $\frac{1}{65536}$ を返す epsilon, 最大の正数を返す infinity, epsilon の 32 倍を返す eps などが定義されている. また mm や pt などの単位も numeric の定数となっている.

2.2.2 pair

pair は（数値, 数値）の値を持つ変数のタイプである. 平面上の点やベクトルを表わすのに用いられる形式である. 以下, 単に点またはベクトルと呼ぶこともある. pair 変数の定数として, (1,0) を表わす right, (-1,0) を表わす left, (0,1) を表わす up, (0,-1) を表わす down, (0,0) を表わす origin が定義されている.

2.2.3 path

path は複数の点を線分や曲線でつなだものを表わす変数のタイプである. 点と点の間に線を引くだけでなく領域を塗りつぶすことが可能で, それらの基本単位として扱われる. 例えば, 3 点（0,0）,（2,0）,（1,$\sqrt{3}$）を直線で結ぶなら (0,0)--(2,0)--(1, $\sqrt{3}$) のように, 曲線で結ぶなら (0,0)..(2,0)..(1, $\sqrt{3}$) のようにすればよい. これらを閉じて領域を作成するのなら (0,0)--(2,0)--(1, $\sqrt{3}$)--cycle のように最後に cycle を書く. これは, はじめの通過点とさいごの通過点を同じにすることで, 閉じるように制御している. 以下, path を経路と呼び, 閉じた path を閉路 (巡回経路) と呼ぶこともある. path の定数として (0,0)--(1,0)--(1,1)--(0,1)--cycle である正方形 unitsquare, 直径 1 の円(半径0.5の円)fullcircle などが定義されている.

2.2.4 color

color は（数値, 数値, 数値）の値を持つ変数のタイプである. それぞれを 0 以上 1 以下の値を入れることにより, 色を指定することができる. rgb 形式で表したとき, 各成分に赤・緑・青の順に並べて色を指定する. (通常は rgb 形式だが, 色に関するマクロを読み込むことにより cmy 形式などに変更することも可能である.) color は形式上 3 次元ベクトルであり, 最初に示したように各成分が数値と同じ値をとることができるので, 空間ベクトルとして使うことも可能である. ただし本来の使い方ではないため, 演算がほとんど定義されていない. 使い慣れてからの使用を勧める. 以上より色としての使い方と, 3 次元ベクトルとしての使い方があると理解できると思うが, 以下では色についてのみ扱うので色と呼ぶ. また, color の定数として red, blue, green, white, black, background が定義されている.

2.2.5 よく使う演算

以下は単項演算子や 2 項演算子として使う「関数」とも思える演算についてまとめたものである．他のプログラミング言語との類似点・相違点に注意してもらいたい．表において x, y は数値, p はベクトルである．

演算式	意味	演算式	意味
$x+y$	$x+y$	length(p)	ベクトル p の長さ
$x-y$	$x-y$	sind(x)	$\sin x$　x は角度 弧度ではない
$x*y$	$x\times y$	cosd(x)	$\cos x$　x は角度 弧度ではない
x/y	$x\div y$	dir(x)	x 度方向の単位ベクトル
$x**y$	x^y	angle(p)	ベクトル p の方向角 (イメージはベクトル p の偏角)
$x++y$	$\sqrt{x^2+y^2}$	xpart(p)	ベクトル p の x 成分
$x+-+y$	$\sqrt{x^2-y^2}$	ypart(p)	ベクトル p の y 成分
sqrt(x)	$\sqrt{x}$	mlog(x)	$256\ln x$
abs(x)	$\lvert x\rvert$	mexp(x)	$e^{x/256}$
abs(p)	$\lvert p\rvert$	⋮	⋮

2.3 変数について 2 (pen, picture, bloolean, string, transform)

ここでは, 変数のうち pen, picture, bloolean, string, transform について説明する．はじめは理解しなくてもよいが, さまざまな図やグラフを作成するためには必要になってくる．とはいえ, pen については理解してほしい．

2.3.1 pen

pen は線を引くときの線の形状を格納しておく変数である．以下ペン先と呼ぶこともある．pen 値には通常プリミティブを変形したものを使う．また, 経路を指定することにより, 自由な形状のペン先を設定できる．ペン先は変数 `currentpen` に格納されているが, 描画している途中でペン先を変更するときは `pickup` ペン先； とする．ペン先を一時的に変更しそのペン先を使用した後, もとに戻したい場合は, `withpen` ペン先 で指定する方がよい．これについては, 後で説明する．pen の定数には, 直径 1 の円の `pencircle` , 1 辺の長さが 1 の `pensquare` , 長さが 1 の横向きの線分の `penrazor` などがある．

2.3.2 picture

picture は画像を保存するための変数である．以下画像と呼ぶこともある．描画途中の画像を保存しているのは, `currentpicture` である．picture のプリミティブは画像が何もない状態の `nullpicture` などがある．

2.3.3 bloolean

bloolean は `true` または `false` のいずれかの値をもつ変数である．よく `if` 文や `for` 文の制御に使われる．しかし, 変数として使用するのではなく bloolean 値をとる式として使われることが多い．例えば `x**2>4` ならば, `x` が $|$`x`$| > 2$ の値なら `true` , $|$`x`$| \leq 2$ の値なら `false` の値をとる．値が論理値をとるので, 以下論理値と呼ぶ．また, picture にはプリミティブは `true, false` ぐらいしかない．

2.3.4 string

string は文字列を格納するための変数である. 以下文字列と呼ぶ. 他のファイルに文字列を書きだしたり, ファイル名を管理したり, フォントを埋め込んだりするのに用いる. string の定数・プリミティブもあまりなく, プリミティブに `jobname`(現在のソースファイルのベースネーム) がある程度である.

2.3.5 transform

transform は 1 次変換と平行移動を行う affine 変換（アフィン変換）を値にもつ変数のタイプである. アフィン変換とは次の 6 つの数値によって定められる変換である.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} p \\ q \end{pmatrix}$$

アフィン変換は単純に説明すると, 拡大・縮小・回転・引き伸ばし・平行移動などの変形を行う変換のことである. このような変換を点・経路・ペン先・画像に対して行うので, 以下変形と呼ぶ. 変形を行うには先ほどの 6 つの数値を設定して使うのも良いが, プリミティブ・基本マクロで定義されている〈変形演算子〉を合成して用いることが一般的である.

2.3.6 〈変形演算子〉

〈変形演算子〉は pair, path, pen, picture に作用する変換で以下のようなものがある. t は数値, p,q はベクトルである. また, `Obj` は変形させたい pair, path, pen, picture を書く. 通常はこれらの合成のみで対応可能である. これらをさらに一般化したものが transform 値になる. ただし, transform 値については扱わない.

ソース	意味
`Obj scaled t`	`Obj` を原点を中心に t 倍に拡大したもの
`Obj xscaled t`	`Obj` を x 軸方向に x 軸から t 倍に引き伸ばしたもの
`Obj yscaled t`	`Obj` を y 軸方向に y 軸から t 倍に引き伸ばしたもの
`Obj zscaled p`	`Obj` に p を複素数として掛け算したもの
`Obj shifted p`	`Obj` をベクトル p だけ平行移動したもの
`Obj rotated t`	`Obj` を原点を中心に t 度だけ回転したもの
`Obj rotatedaround (p,t)`	`Obj` を点 p を中心に t 度だけ回転したもの
`Obj reflectedabout (p,q)`	`Obj` を 2 点 p,q を結ぶ直線に関して対称移動したもの

2.4 補間

以上からさまざまな値を使うことができると理解できると思うが, それらを補間した値も用いることができる. numeric, pair, path(3 次元ベクトルとみるとき color も) について補間を行うことができる.

pair を補間する方法は, 分点である. これは 2 点 A, B に対して, 線分 AB を $t : (1-t)$ に分ける点を `t[A,B]` で表す. このとき t は数値なので, 内分点・外分点のどちらも表すことができる. 同様に numeric も補間することができる (color も同様である).

path を補間する場合は, `interpath` というマクロを使う. p, q を経路とするとき `interpath(t,p,q)` のように表現する. これは, 2 つの経路を構成する点を対応させて, それを $t : (1-t)$ に分ける点を作り, 新たに経路を得る式である.

2.5 その他の重要な使い方

ここまでに書かれた内容で色々と作成できるが, 値の使い方や, 制御方法について簡単にここで説明しておく.

2.5.1 代入と等値

変数に値を設定するとき, 代入と等値の 2 通りがある.

代入は, 式を計算して得られる値または具体的な値を変数に格納する方法である. 「代入先の変数 `:=` 式または値」 のようにして代入する. 例えば `a,b,c` を数値としたとき `a:=b+c;` や `a:=a+1;` のように使う.

等値は, 条件を連立方程式で与え, それにより値を設定する方法である. 図を作成するのに有効である. 例えばベクトル `p` を `p.x, p.y` の 2 方向に分解するとしよう. 分解されたベクトルをマクロの `z0x, z0y` に値を入れるとする. これは次の 3 つの式により, 設定できる.

```
z0x+z0y=p;  z0x=s*p.x;  z0y=t*p.y;
```

ここで, `s,t` は一時的に使う数値である. これは余分に変数を使っていることになり, メモリ的にもよくない. そこで, 毎回異なる定数を生成してくれる `whatever` を用いて次のようにするのが一般的である.

```
z0x+z0y=p;  z0x=whatever*p.x;  z0y=whatever*p.y;
```

2.5.2 for 文・if 文

for 文・if 文は他のプログラム言語と同様に使える.

for 文は, 例えば次のようにして使う.

```
for n=0 step 4 until 360: z[n]=dir(n); endfor
```

は `z0=dir(0); z4=dir(4); …… z360=dir(360);` と書いたことと同じになる. また,

```
draw z0 for n=1 upto 4: ..z[n] endfor;
```

とすれば, `z0..z1..z2..z3..z4;` と書いたことと同じになる. ここで ; の位置に注意してもらいたい. どの範囲が「文」になるのかを考えて理解してもらいたい.

if 文は, 例えば次のようにして使う.

```
if pair c       : z0=c;
elseif numeric c: z0=c*right;
else            : z0=(0,0);
fi
```

このように `if ○○: ××; elseif ○○: ××; else ○○: ××;` で条件とその時の制御を書き, `fi` によって if 文を終了する.

2.5.3 単位

前述したように, `pt` や `mm` などの単位は TEX と同じで数値定数として扱われる. `pt` や `mm` などの数値は, 「デフォルトの単位 (`bp`)」との比率を表わす.

3　実用的ないくつかの例

ここからは, 実際に (高校数学の授業で) 使えそうな例を挙げていく. そのため複雑になっている部分の説明を省こうと思う. 分からない場合は, 質問してもらって構わない.

まずは簡単な例から始めよう. 次のファイルは $y = x$ のグラフを作成するものである.

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  v=1cm;
  drawarrow (-.5v,0)--(4v,0);
  drawarrow (0,-.5v)--(0,4v);
  pickup pencircle scaled 1;
  draw (0,0){dir 45}    for i=1 upto 8: ..(i/2,i/2)*v endfor;
  label.lrt(btex $y=x$ etex, (3,3)*v);
  label.rt(btex $x$ etex, (4,0)*v);
  label.top(btex $y$ etex, (0,4)*v);
  label.llft(btex O etex, (0,0));
endfig;
end.
```

`draw (0,0){dir 45}`は, (0,0) から引く直線 (曲線) を制御するためにある. この場合, (0,0) から 45° の角度で引くことを表している. このほかに up((1,0) 方向), down((-1,0) 方向), right((0,1) 方向), left((0,-1) 方向), dir 角度, curl 数値 (曲がり具合), tention 数値 (張り具合) などがある. これらは自由に使うことができて, 複数の制御をすることが可能である. 例えば draw (0,0){up}..{right}(2,2)..{down}(4,1) のように使う. ふつうこれらの制御は, 思い通りの図が作成できない時に使う.

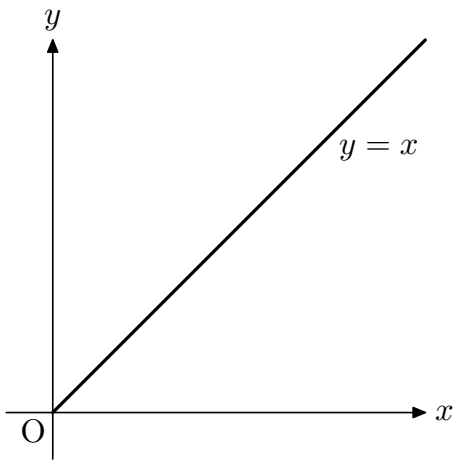


図 1: y=x

文字列を座標 `(a,b)` の中心におくとき, `label(文字列, (a,b))` などとすればよい. これ以外にもさまざまなラベル (文字列を埋め込む方法) ある. 説明が長くなるので仮に, 点 (またはベクトル)p=(x,y) としておこう. 点 p の上側に文字列を置きたいときは, `label.top(文字列,p)` とすればよい. 同様に, 下側なら `label.bot(文字列,p)`, 左側なら `label.lft(文字列,p)`, 右側なら `label.rt(文字列,p)`, 左上側なら `label.ulft(文字列,p)`, 左下側なら `label.llft(文字列,p)`, 右上側なら `label.url(文字列,p)`, 右下側なら `label.lrt(文字列,p)` とすればよい. ただし上記のように `verbatimtex \documentclass{article} \begin{document} etex;` を入れ, `label.rt(btex $x$ etex, (4,0)*v);` のように書くことで, 文字を埋め込むことができるようになっている.

また, for 文を使うと楽に作図することができる. この場合, (0,0) と (1/2,1/2) を繋ぎ, 次に (1/2,1/2) と (1,1) を繋ぎ……と繰り返して $y = x$ のグラフを作成している.

これらを用いて作成したのが図 1 の $y = x$ のグラフである. $y = x$ のグラフを作成するだけならもっと簡単にできるが, 練習だと思ってほしい.

次も簡単な例にしよう. 次のファイルは $y = \sqrt{x}$ のグラフを作成するものである.

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  u=1cm;
  drawarrow (-.5u,0)--(4u,0);
  drawarrow (0,-.5u)--(0,2u);
  pickup pencircle scaled 1;
  draw (0,0){up}  for i=1 upto 8: ..(i/2,sqrt(i/2))*u endfor;
  label.lrt(btex $y=\sqrt{x}$ etex, (3,sqrt(3))*u);
  label.rt(btex $x$ etex, (4,0)*u);
  label.top(btex $y$ etex, (0,2)*u);
  label.llft(btex O etex, (0,0));
endfig;
end.
```

書き忘れていたが, 『0.5』のように小数の整数部分が『0』である場合は省略することが可能で, 『.5』と書くだけでよい. こんな内容, 言われるまでもないと思うが……

METAPOST の数学関数の代表例として『sqrt ()』を使った例である. 他にもあり, これ以降の例でも出てくるが, 各自で調べてもらいたい.

図 2 のようにできただろうか?

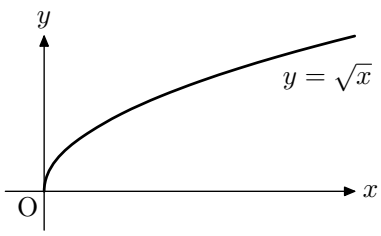


図 2: $y = \sqrt{x}$

ここからは徐々に面倒になる. 心してかかるようにしてもらいたい. 次のファイルは $y = x^2$ のグラフのうち $x \geq 0$ の部分を作成するものである.

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  u=1cm;
  drawarrow (-.5u,0)--(4u,0);
  drawarrow (0,-.5u)--(0,8u);
  pickup pencircle scaled 1;
  draw (0,0){right}  for i=1 upto 8: ..(i/2,i**2/(2**3))*u endfor;
  label.lrt(btex $y=x^2$ etex, (3,4)*u);
  label.rt(btex $x$ etex, (4,0)*u);
  label.top(btex $y$ etex, (0,8)*u);
  label.llft(btex O etex, (0,0));
endfig;
end.
```

ここで, for 文の中で `(i/2,i**2/(2**3))*u` としているのは, `(i/2,i**2/(2**2))*u` とするとグラフが大きく変化し見づらくなるのを防ぐためである. どうなるかは, 実際に実行して確かめてみてもらいたい.

ともあれ, このファイルから作成されるのは図 3 の $y = x^2$ (ただし $x \geq 0$) である. このファイルに少し手を加えると図 4 のように $x \leq 0$ の部分も描ける. これは, 各自で取り組んでもらおう.

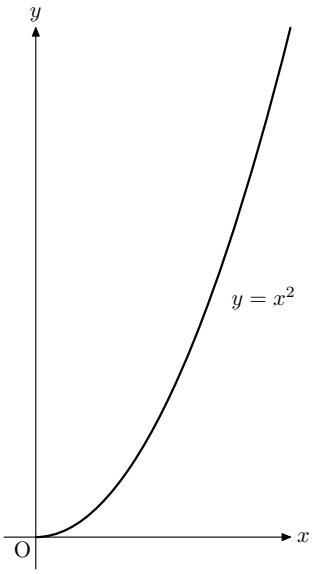


図 3: $y = x^2$ (ただし $x \geq 0$)

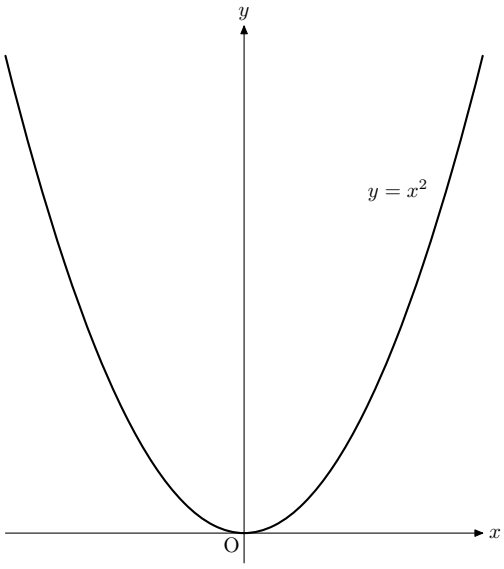


図 4: $y = x^2$

ここまでは, 理解しただろうか. ここでは, プリミティブを用いずに単位円を作成しようと思う. 次のファイルは単位円 $x^2 + y^2 = 1$ のグラフを作成するものである.

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  u=1cm;
  drawarrow (-1.4*u,0)--(1.4*u,0);
  drawarrow (0,-1.4*u)--(0,1.4*u);
  pickup pencircle scaled 1;
  draw (u,0){up}  for i=1 upto 7: ..(cosd(i*45),sind(i*45))*u endfor ..cycle;
  label.llft(btex O etex scaled 0.7, (0,0));
  label.urt(btex $1$ etex scaled 0.7, (1,0)*u);
  label.urt(btex $1$ etex scaled 0.7, (0,1)*u);
  label.ulft(btex $-1$ etex scaled 0.7, (-1,0)*u);
  label.llft(btex $-1$ etex scaled 0.7, (0,-1)*u);
  label.rt(btex $x$ etex scaled 0.7, (1.4,0)*u);
  label.top(btex $y$ etex scaled 0.7, (0,1.4)*u);
endfig;
end.
```

結果は図 5 のようになる. わずか 8 点のみによる制御しかないにもかかわらず, 図 5 を見ると綺麗な円が描けていることが分かるだろう. 筆者ははじめ非常に驚き, 理由を考えて納得したのだが, これを読んでいる人はどう感じるのだろうか? ただ『そんなものなのだろう』や『そうなるに決まっている』としか感じられないのだろうか? さて, 図 5 は制御点が 9 点であるが, 制御点の点数を変えたらどうなるのだろうか? 滑らかになるのだろうか? それとも不自然な曲線になるのだろうか? 各自で予想をたてた上で, やってもらいたい.

`cosd(),sind()` は角度を引数にもち, それぞれ sin, cos の値を返す関数である.

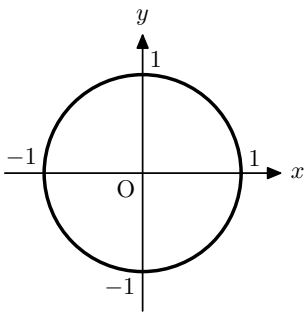


図 5: $x^2 + y^2 = 1$

では, この流れで $\sin x$ のグラフを描こうと思う. まず次のファイルは単位円と正弦値 $\sin x$ $(0 \leq x \leq 2\pi)$ を対比させたグラフを作成するものである.

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  u=1cm;
  PI=3.14159265358979323846264338327 9;
  a=180/PI;
  drawarrow (-1.4*u,0)--(8*u,0);
  drawarrow (0,-1.4*u)--(0,1.4*u);
  pickup pencircle scaled 1;
  draw (u,0){up} for i=1 upto  8: ..(cosd(i*45),sind(i*45))*u endfor;
  draw (1.3*u,0){dir 40} for i=1 upto 36: ..(1.3+i*10/a,sind(i*10))*u endfor;
  label.top(btex $y$ etex,(0,1.4)*u);
  label.llft(btex O etex, (0,0));
endfig;
end.
```

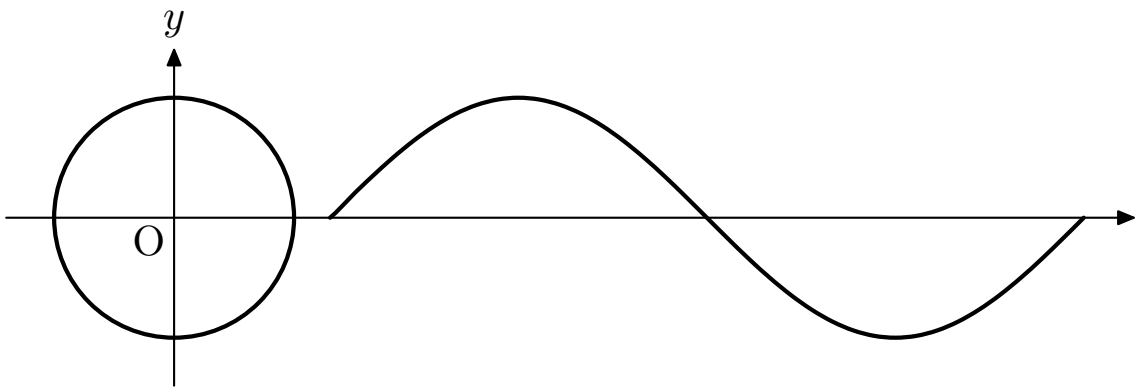


図 6: 単位円と $\sin x$

さらに $0 \leq x \leq 4\pi$ とし, 軸を別にとったものが次になる.

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  u=1cm;
  PI=3.14159265358979323846264338327 9;
  a=180/PI;
  drawarrow (-1.4*u,0)--(8*u,0);
  drawarrow (0,-1.4*u)--(0,1.4*u);
  pickup pencircle scaled 1;
  draw (u,0){up}  for i=1 upto 8: ..(cosd(i*45),sind(i*45))*u endfor;
  draw (1.3*u,0){dir 40}  for i=1 upto 36: ..(1.3+i*10/a,sind(i*10))*u endfor;
  label.top(btex $y$ etex,(0,1.4)*u);
  label.llft(btex O etex, (0,0));
endfig;
end.
```

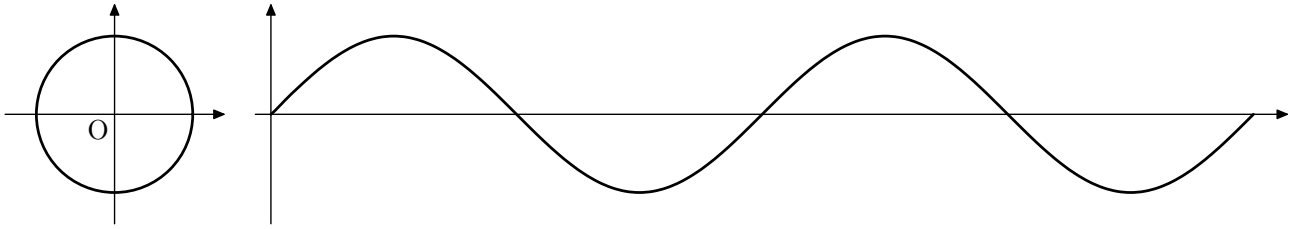


図 7: 単位円と $\sin x$

さらに, θ 回転したときの単位円上の点 P とそれに対応する $\sin\theta$ を結ぶようにしたのが次のファイルである.

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  u=1cm;
  PI=3.14159265358979323846264338327 9;
  a=180/PI;
  drawarrow (-1.4*u,0)--(1.4*u,0);
  drawarrow (0,-1.4*u)--(0,1.4*u);
  drawarrow (1.8*u,0)--(15*u,0);
  drawarrow (2*u,-1.4*u)--(2*u,1.4*u);
  pickup pencircle scaled 1 ;
  draw (u,0){up}  for i=1 upto 8: ..(cosd(i*45),sind(i*45))*u endfor;
  draw (2*u,0){dir 43}  for i=1 upto 72: ..(2+i*10/a,sind(i*10))*u endfor;
  pickup pencircle scaled 0.3 ;
  drawarrow (0,0)..(cosd(140),sind(140))*u withcolor red;
  draw (cosd(140),sind(140))*u{right}--(2+140/a,sind(140))*u withcolor blue;
  draw (2+140/a,sind(140))*u{down}--(2+140/a,0)*u withcolor blue;
  draw (1,0)*0.3*u{up}..(cosd(140),sind(140))*0.3*u;
  draw (-0.5,0.7)*0.3*u--(cosd(140),sind(140))*0.3*u;
  draw (-0.73,0.9)*0.3*u--(cosd(140),sind(140))*0.3*u;
  label.ulft(btex P($\cos \theta,\sin \theta$) etex,(-1,0.5)*u);
  label.llft(btex O etex, (0,0));
  label.urt(btex P etex,(2+140/a,sind(140))*u);
  label.bot(btex $\theta$ etex, (2+140/a,0)*u);
  label.urt(btex $\theta$ etex,(cosd(60),sind(60))*0.25*u);
  label.rt(btex $y$軸 etex,(0,1.4*u));
  label.top(btex $x$軸 etex,(1.4*u,0));
  label.llft(btex O etex, (2*u,0));
  label.rt(btex $y$軸 etex,(2*u,1.4*u));
  label.top(btex $\theta$軸 etex,(14.5*u,0));
  label.ulft(btex $\sin \theta$ etex, (2,sind(140))*u);
endfig;
end.
```

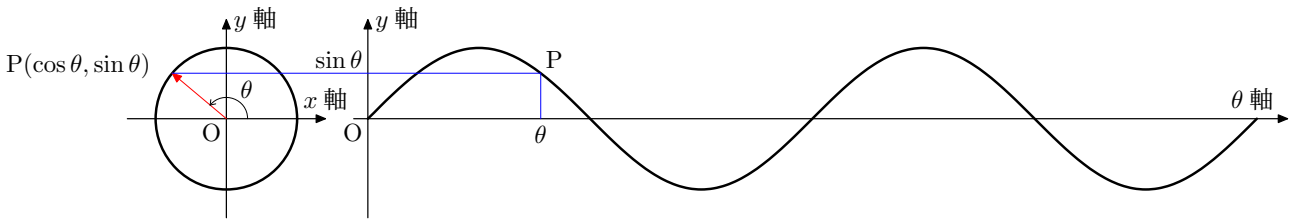


図 8: 単位円と $\sin x$

それでは最後に, 渦巻き曲線の 1 つを描くことにしよう. 次は, (現行の) 高校数学 3 年, 数学 C で習う内容, 媒介変数や極方程式で表す曲線や直線の 1 つである渦巻き曲線を描くファイルである. 渦巻き曲線は $(r(\theta)=)\ r=a\theta\ (a\geq 0)$ で与えられるが, a=1 とした場合を作成した.

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  u=1cm;
  PI=3.14159265358979323846264338327 9;
  b=180/PI;
  c=1/8;
  drawarrow (-3.5*u,0)--(3.5*u,0);
  drawarrow (0,-3.5*u)--(0,3.5*u);
  pair xa,xb,xc,xd,xe,xf,xg,xh,xi,xj,xk,xl,xm,xn,xo,xp;
  xa=(0,PI/2)*c*u;
  xb=(-PI,0)*c*u;
  xc=(0,-3*PI/2)*c*u;
  xd=(2*PI,0)*c*u;
  xe=(0,5*PI/2)*c*u;
  xf=(-3*PI,0)*c*u;
  xg=(0,-7*PI/2)*c*u;
  xh=(4*PI,0)*c*u;
  xi=(0,9*PI/2)*c*u;
  xj=(-5*PI,0)*c*u;
  xk=(0,-11*PI/2)*c*u;
  xl=(6*PI,0)*c*u;
  xm=(0,13*PI/2)*c*u;
  xn=(-7*PI,0)*c*u;
  xo=(0,-15*PI/2)*c*u;
  xp=(8*PI,0)*c*u;
%%%%
  pickup pencircle scaled 0.3 ;
  draw (0,0){right}
    for i=1 upto 1450: ..((i/b)*cosd(i),(i/b)*sind(i))*c*u endfor;
  label.llft(btex O etex scaled 0.8, (0,0));
  label.urt(btex  $r=a\theta (a=1,0\leq \theta \leq 8\pi+\alpha)$ etex scaled 0.8, (2,2)*u);
  label.urt(btex  渦巻く曲線 etex scaled 0.8, (2,1.6)*u);
  label.urt(btex  $   \frac{\pi}{2}$ etex scaled 0.8, xa);
  label.ulft(btex $        -\pi    $ etex scaled 0.8, xb);
  label.lrt(btex  $ -\frac{3}{2}\pi$ etex scaled 0.8, xc);
  label.urt(btex  $        2\pi    $ etex scaled 0.8, xd);
  label.urt(btex  $  \frac{5}{2}\pi$ etex scaled 0.8, xe);
  label.ulft(btex $       -3\pi    $ etex scaled 0.8, xf);
  label.lrt(btex  $ -\frac{7}{2}\pi$ etex scaled 0.8, xg);
  label.urt(btex  $        4\pi    $ etex scaled 0.8, xh);
  label.urt(btex  $  \frac{9}{2}\pi$ etex scaled 0.8, xi);
  label.ulft(btex $       -5\pi    $ etex scaled 0.8, xj);
  label.lrt(btex  $-\frac{11}{2}\pi$ etex scaled 0.8, xk);
  label.urt(btex  $        6\pi    $ etex scaled 0.8, xl);
  label.urt(btex  $ \frac{13}{2}\pi$ etex scaled 0.8, xm);
  label.ulft(btex $       -7\pi    $ etex scaled 0.8, xn);
  label.lrt(btex  $-\frac{15}{2}\pi$ etex scaled 0.8, xo);
  label.urt(btex  $        8\pi    $ etex scaled 0.8, xp);
endfig;
end.
```

上のように記述すると, 図 9 が得られる. ちなみにファイル中で `c=1/8` としているのは, そうしなければ曲線がすぐに大きくなってしまうからである. これは本当に各自で確認してもらいたい. (`c=1` とすれば, 全ての値が変更できるので確認しやすいと思う)

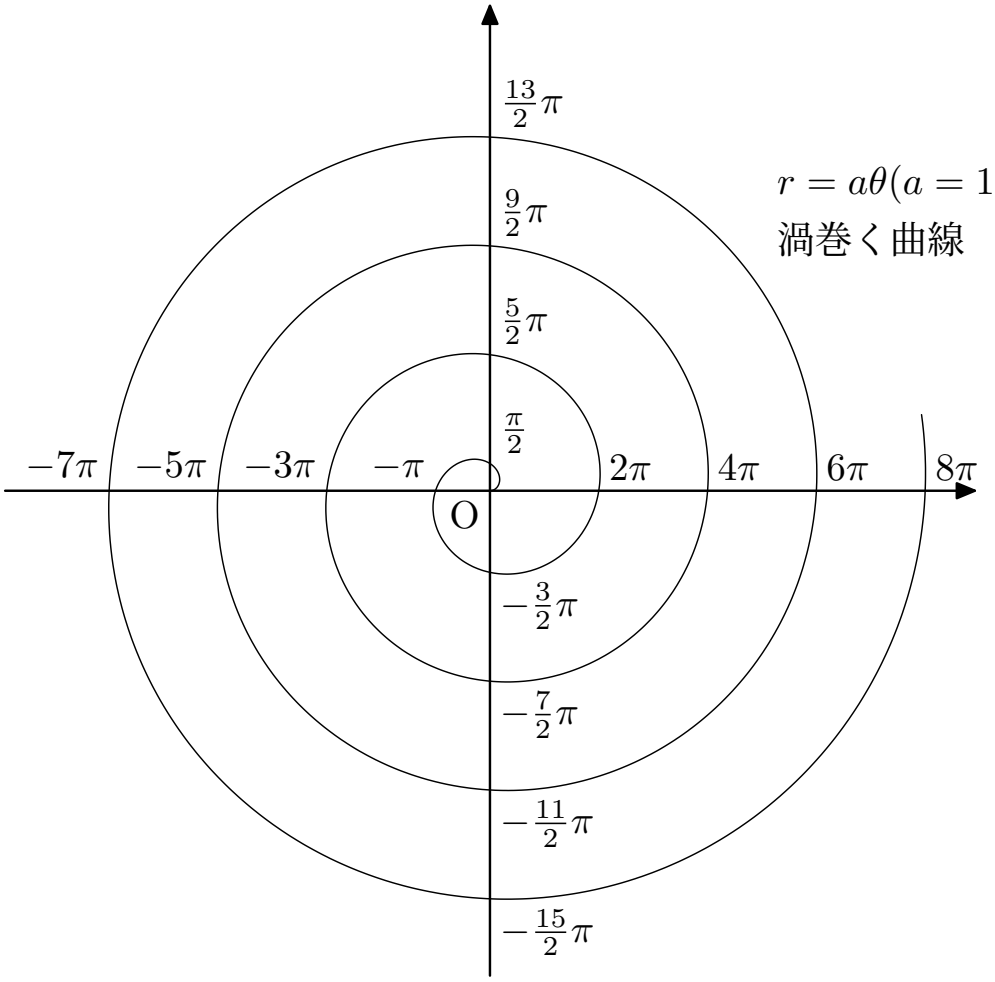

$\frac{13}{2}\pi$
$\frac{9}{2}\pi$
$\frac{5}{2}\pi$
$\frac{\pi}{2}$
-7π
-5π
-3π
$-\pi$
O
2π
4π
6π
8π
$-\frac{3}{2}\pi$
$-\frac{7}{2}\pi$
$-\frac{11}{2}\pi$
$-\frac{15}{2}\pi$
$r = a\theta(a = 1, 0 \leq \theta \leq 8\pi + \alpha)$
渦巻く曲線

図 9: $r = \theta$

4　あると便利なコマンド集

以下は筆者がさまざまなコマンドの中から使えそうなもの集めて記述した. ただし, 他のマクロを用いて使う方が良いこともある.

点を描画するコマンドから, 点列を描くもの↓

```
  drawdot (0,0);
    for i=1 upto 50: drawdot (i,i); endfor;
```

ちなみに, これを用いると図 8 は図 10 のように見やすくなるのが分かるだろう. ただ, 微調整が少々煩わしい気がしないでもない.

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  u=1cm;
  PI=3.14159265358979323846264338327 9;
  a=180/PI;
%%%
  drawarrow (-1.4*u,0)--(1.4*u,0);
  drawarrow (0,-1.4*u)--(0,1.4*u);
  drawarrow (1.8*u,0)--(15*u,0);
  drawarrow (2*u,-1.4*u)--(2*u,1.4*u);
  pickup pencircle scaled 1 ;
  draw (u,0){up}
    for i=1 upto 8: ..(cosd(i*45),sind(i*45))*u endfor;
  draw (2*u,0)
    for i=1 upto 72: ..(2+i*10/a,sind(i*10))*u endfor;
  pickup pencircle scaled 0.3 ;
  drawarrow (0,0)..(cosd(140),sind(140))*u withcolor red;
  pickup pencircle scaled 1 ;
  b=2+(140/a)+abs(cosd(140));
  for i=0 upto 46: drawdot (i*b/46+cosd(140),sind(140))*u withcolor blue; endfor;
  c=sind(140);
  d=2+140/a;
  for i=0 upto 8: drawdot (d,c-i*c/8)*u withcolor blue; endfor;
%%%
  pickup pencircle scaled 0.3 ;
  draw (1,0)*0.3*u{up}..(cosd(140),sind(140))*0.3*u;
  draw (-0.5,0.7)*0.3*u--(cosd(140),sind(140))*0.3*u;
  draw (-0.73,0.9)*0.3*u--(cosd(140),sind(140))*0.3*u;
%%%
  label.ulft(btex P($\cos \theta,\sin \theta$) etex,(-1,0.5)*u);
  label.llft(btex O etex, (0,0));
  label.urt(btex P etex,(2+140/a,sind(140))*u);
  label.bot(btex $\theta$ etex, (2+140/a,0)*u);
  label.urt(btex $\theta$ etex,(cosd(60),sind(60))*0.25*u);
  label.rt(btex $y$軸 etex,(0,1.4*u));
  label.top(btex $x$軸 etex,(1.4*u,0));
  label.llft(btex O etex, (2*u,0));
  label.rt(btex $y$軸 etex,(2*u,1.4*u));
  label.top(btex $\theta$軸 etex,(14.5*u,0));
  label.ulft(btex $\sin \theta$ etex, (2,sind(140))*u);
endfig;
end.
```

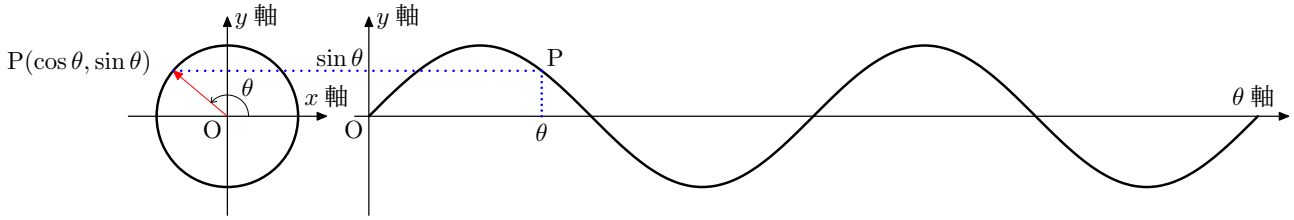


図 10: 単位円と $\sin x$

直線を描画するコマンドから, 点線を描くもの↓

```
draw (0,0)--(1/2,1/2);
  for i=1 upto 50: draw (i,i)--(i+1/2,i+1/2); endfor;
```

ちなみに, これを用いると図 8 は図 11 のようになる. 端点がそろうように制御するのに苦労した. また, 分かりやすくするため $0 \leq \theta \leq 2\pi$ とした. 見やすくなっただろうか?

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  u=1cm;
  PI=3.14159265358979323846264338327 9;
  a=180/PI;
  drawarrow (-1.4*u,0)--(1.4*u,0);
  drawarrow (0,-1.4*u)--(0,1.4*u);
  drawarrow (1.8*u,0)--(9*u,0);
  drawarrow (2*u,-1.4*u)--(2*u,1.4*u);
  pickup pencircle scaled 1 ;
  draw (u,0){up}  for i=1 upto 8: ..(cosd(i*45),sind(i*45))*u endfor;
  draw (2*u,0){dir 43}  for i=1 upto 36: ..(2+i*10/a,sind(i*10))*u endfor;
  pickup pencircle scaled 0.3 ;
  drawarrow (0,0)..(cosd(140),sind(140))*u withcolor red;
%%%
  pickup pencircle scaled 1 ;
  b=2+(140/a)+abs(cosd(140));
  for i=0 upto 30:
    draw (i*b/31+cosd(140)+i*(b/31-1/20)/31,sind(140))*u
       --(i*b/31+cosd(140)+1/20+i*(b/31-1/20)/31,sind(140))*u withcolor blue; endfor;
  c=sind(140);
  d=2+140/a;
  for i=0 upto 4:
    draw (d,c-i*c/5-i*(c/5-1/20)/5)*u--(d,c-i*c/5-1/20-i*(c/5-1/20)/5)*u
    withcolor blue; endfor;
%%%
  pickup pencircle scaled 0.3 ;
  draw (1,0)*0.3*u{up}..(cosd(140),sind(140))*0.3*u;
  draw (-0.5,0.7)*0.3*u--(cosd(140),sind(140))*0.3*u;
  draw (-0.73,0.9)*0.3*u--(cosd(140),sind(140))*0.3*u;
  label.ulft(btex P($\cos \theta,\sin \theta$) etex,(-1,0.5)*u);
  label.llft(btex O etex, (0,0));
  label.urt(btex P etex,(2+140/a,sind(140))*u);
  label.bot(btex $\theta$ etex, (2+140/a,0)*u);
  label.urt(btex $\theta$ etex,(cosd(60),sind(60))*0.25*u);
  label.rt(btex $y$軸 etex,(0,1.4*u));
  label.top(btex $x$軸 etex,(1.4*u,0));
  label.llft(btex O etex, (2*u,0));
```

```
  label.rt(btex $y$軸 etex,(2*u,1.4*u));
  label.top(btex $\theta$軸 etex,(8.8*u,0));
  label.ulft(btex $\sin \theta$ etex, (2,sind(140))*u);
endfig;
end.
```

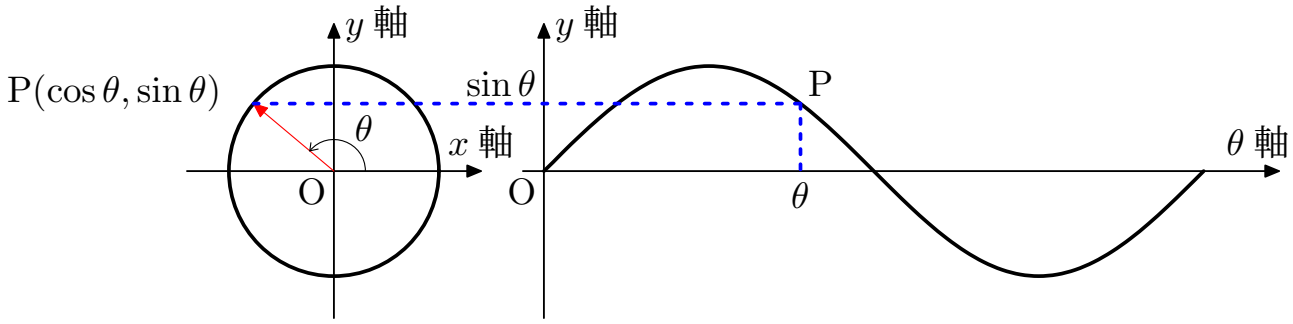


図 11: 単位円と $\sin x$

同様に, 図 9 を点列で出力したのが次の図 12 である. ファイルは 2 行を書き換えるだけなので, 各自で確かめてもらいたい. 図 12 は 1 度ごとに点を描画し, 1450 度までを出力するものである. これから, 通常の関数を点列で描くことは簡単であることが分かる.

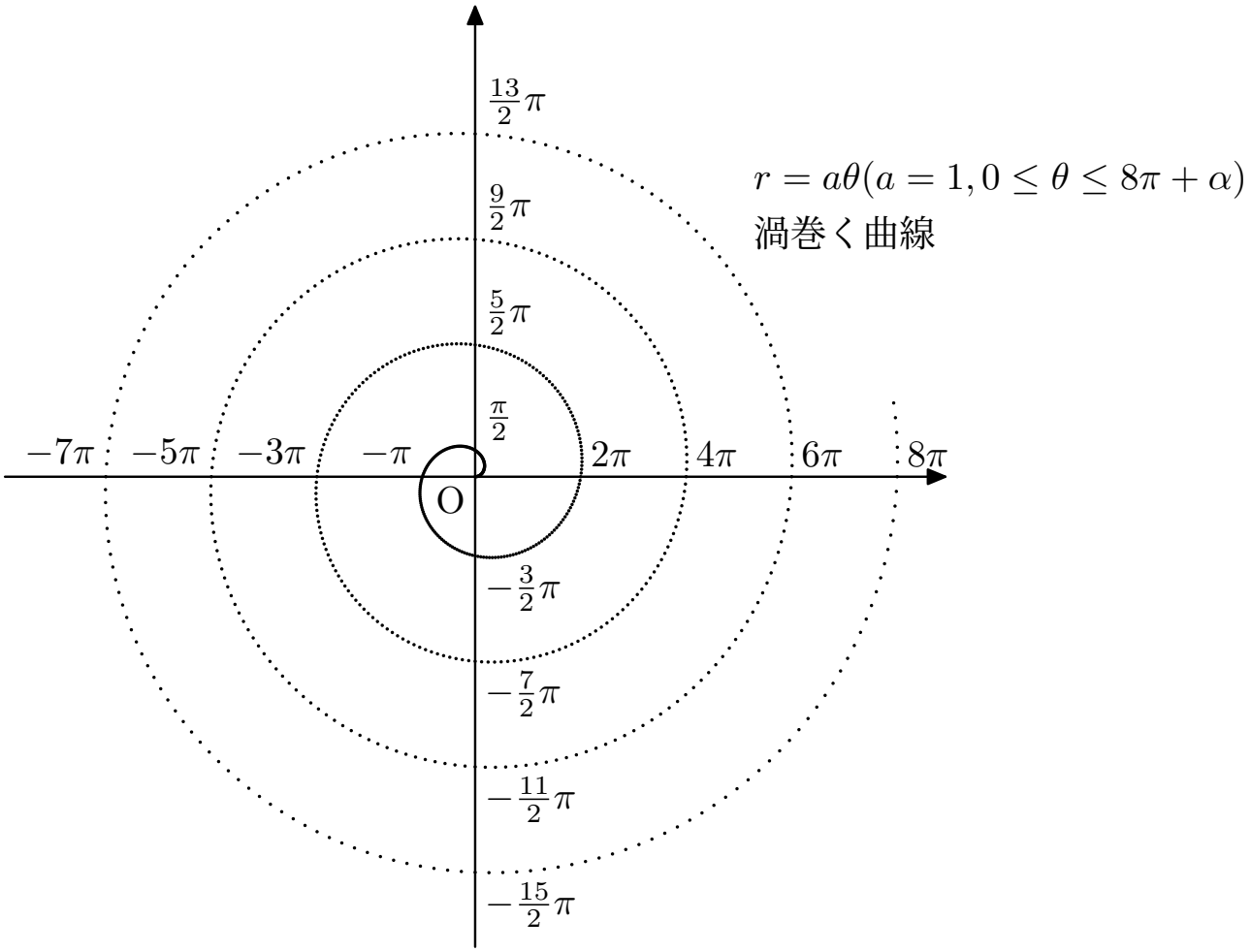


図 12: $r = \theta$

点線 (破線) を描くコマンドを見つけたので, 記述することにした. 以下はそれ. on ○○は実線の長さ, off ○○は空線 (?) の長さを表わす. また, on 0 は点になる. dashpattern( ) は ( ) の中を繰り返すことを表わしており, 前述したものより簡単になったと言えるだろう.

```
  draw (0,0)--(1/2,1/2) dashed (dashpattern(on 2pt off 2pt));
  draw (0,0)--(1/2,1/2) dashed (dashpattern(on 2pt off 2pt on 1pt off 2pt on 2pt));
  draw (0,0)--(1/2,1/2) dashed (dashpattern(on 0 off 1pt));
  draw (0,0)--(1/2,1/2) dashed evenly;
  draw (0,0)--(1/2,1/2) dashed withdots;
```

これを使うと図 11 は図 13 のようになり, ソースは次のようになる. ただし端点はそろえていない.

```
verbatimtex
  \documentclass{article}
  \begin{document}
etex;
beginfig(-1);
  u=1cm;
  PI=3.14159265358979323846264338327 9;
  a=180/PI;
  drawarrow (-1.4*u,0)--(1.4*u,0);
  drawarrow (0,-1.4*u)--(0,1.4*u);
  drawarrow (1.8*u,0)--(9*u,0);
  drawarrow (2*u,-1.4*u)--(2*u,1.4*u);
  pickup pencircle scaled 1 ;
  draw (u,0){up}
    for i=1 upto 8: ..(cosd(i*45),sind(i*45))*u endfor;
  draw (2*u,0)
    for i=1 upto 36: ..(2+i*10/a,sind(i*10))*u endfor;
  pickup pencircle scaled 0.3 ;
  drawarrow (0,0)..(cosd(140),sind(140))*u withcolor red;
%%%
  pickup pencircle scaled 1 ;
  draw (cosd(140),sind(140))*u--(2+140/a,sind(140))*u
    dashed(dashpattern(on 2pt off 2pt)) withcolor blue;
  c=sind(140);
  d=2+140/a;
  draw (d,0)*u--(d,c)*u dashed(dashpattern(on 2pt off 2pt)) withcolor blue;
%%%
  pickup pencircle scaled 0.3 ;
  draw (1,0)*0.3*u{up}..(cosd(140),sind(140))*0.3*u;
  draw (-0.5,0.7)*0.3*u--(cosd(140),sind(140))*0.3*u;
  draw (-0.73,0.9)*0.3*u--(cosd(140),sind(140))*0.3*u;
%%%
  label.ulft(btex P($\cos \theta,\sin \theta$) etex,(-1,0.5)*u);
  label.llft(btex O etex, (0,0));
  label.urt(btex P etex,(2+140/a,sind(140))*u);
  label.bot(btex $\theta$ etex, (2+140/a,0)*u);
  label.urt(btex $\theta$ etex,(cosd(60),sind(60))*0.25*u);
  label.rt(btex $y$軸 etex,(0,1.4*u));
  label.top(btex $x$軸 etex,(1.4*u,0));
  label.llft(btex O etex, (2*u,0));
  label.rt(btex $y$軸 etex,(2*u,1.4*u));
  label.top(btex $\theta$軸 etex,(8.8*u,0));
  label.ulft(btex $\sin \theta$ etex, (2,sind(140))*u);
endfig;
end.
```

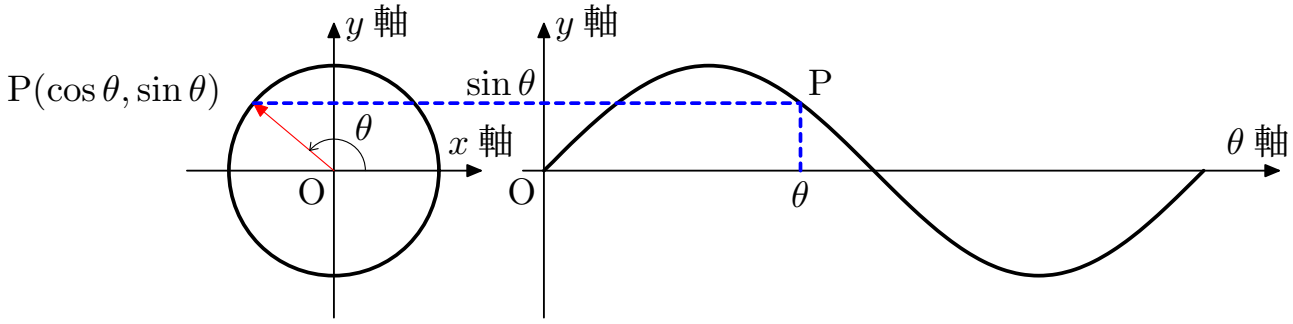


図 13: 単位円と $\sin x$

5　マクロについて

ここまでにさまざまな図を作成するソースファイルを説明してきたが, いちいち全てを入力する方法ではファイルが大きくなりすぎる. 1 連の操作や計算手順などをまとめて定義して名前をつけ, 別の場所に置き, 必要に応じて呼び出すほうが楽である. METAPOST ではこの方法をマクロで実現する. 定義等は後回しにして, 簡単なマクロの例から見ていこう.

5.1　マクロの例１

次のファイルは与えられた 3 点 p,q,r をマクロが受け取って結び, 三角形を作成するものである.

```
beginfig(-1);
%%%
  def drawSankaku(expr a,b,c)=
      draw a--b--c--cycle;
  enddef;
%%%
  u=1cm;
  pair a,b,c;
  a=(0,u);  b=(2u,0);  c=(3u,1.7u);
  drawSankaku(a,b,c);
endfig;
end.
```

上のソースファイルから作成される流れを視覚化したものが図 14 である. 実際には右側の黒の実線のみしか作成されない. 説明すると次のようになる. まず, 座標 a,b,c を指定する. 次に, drawSankaku(a,b,c) でマクロ drawSankaku を呼び出す. 定義に従って a,b,c を結ぶ閉路を描く. 終了する.

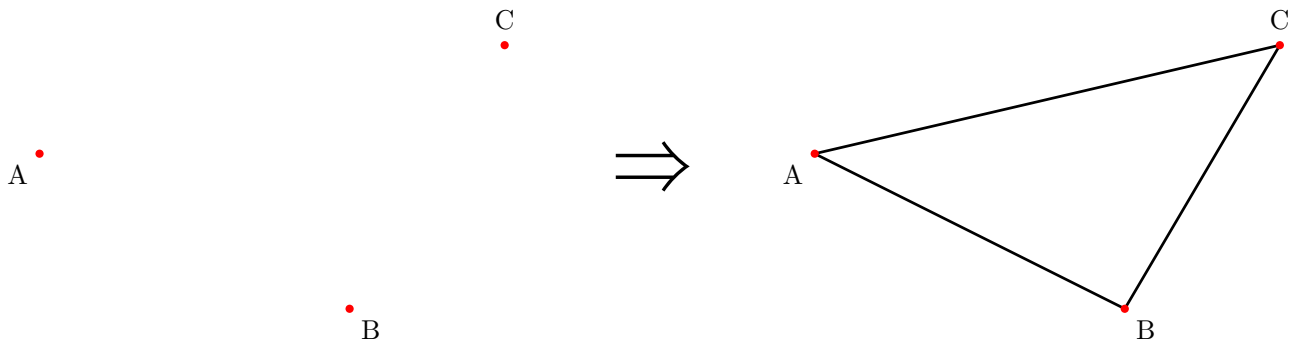


図 14: 3 点から三角形

5.2　マクロの例２

次はマクロ drawSankaku に少し手を加えたマクロ drawTyusen を作成しよう．次のファイルは与えられた 3 点 p,q,r をマクロが受け取って結び, 三角形を作成した後, 頂点と中点を結ぶものである．

```
beginfig(-1);
%%%
  def drawTyusen(expr a,b,c)=
      draw a--b--c--cycle;
      draw a--(0.5[b,c]);
      draw b--(0.5[c,a]);
      draw c--(0.5[a,b]);
  enddef;
%%%
  u=1cm;
  pair a,b,c;
  a=(0,u);  b=(2u,0);  c=(3u,1.7u);
  drawTyusen(a,b,c);
endfig;
end.
```

上のソースファイルから作成される流れを視覚化したものが図 15 である．頂点は強調してある．実際には右下の黒の実線のみしか作成されない．説明すると次のようになる．まず，座標 a,b,c を指定する．次に，drawTyusen(a,b,c) でマクロ drawTyusen を呼び出す．定義に従ってまず a,b,c を結ぶ閉路を描く．次に a を b と c の中点と結び，b を c と a の中点と結び，c を a と b の中点と結ぶ．終了する．

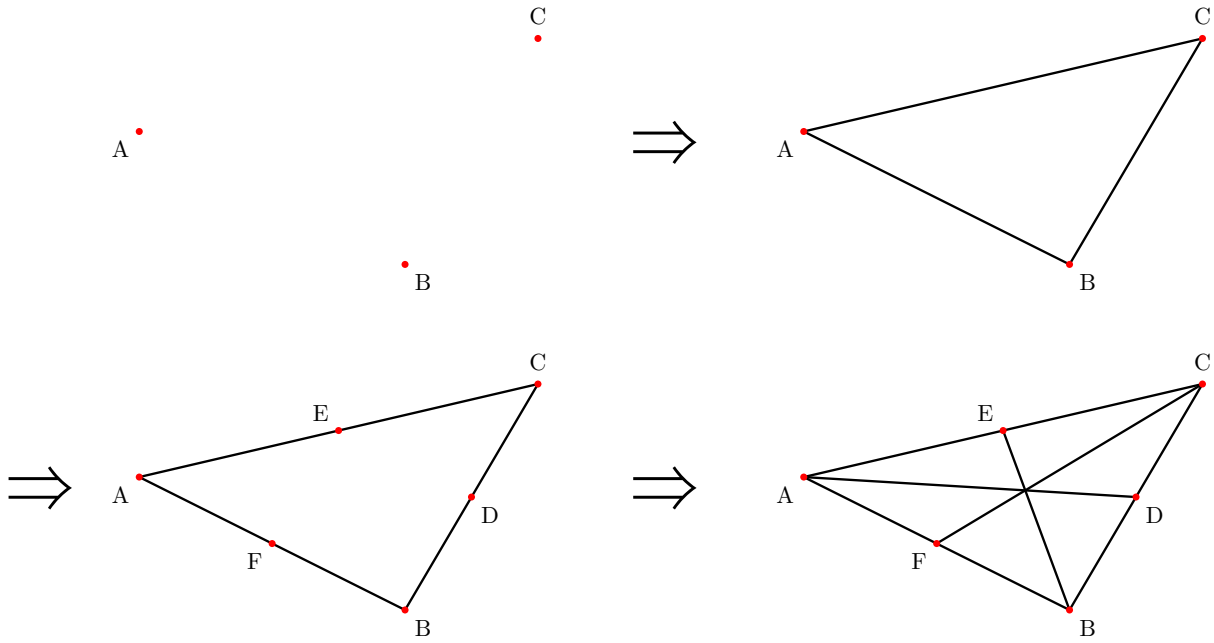


図 15: 3 点から三角形と中線

5.3 マクロの例３

与えられた値から直線や曲線をマクロが描くのもよいが, 与えられた値から別の値を返すマクロも作成できる. ここでは, 与えられた 3 点 p,q,r を受け取って, 三角形を作成する経路を返すマクロ pathSankaku を考えよう. 以下はそのファイルである.

```
beginfig(-1);
%%%
  vardef pathSankaku(expr a,b,c)=
      begingroup
        path p;
        p:=a--b--c--cycle;
        p
      endgroup
  enddef;
%%%
  u=1cm;
  pair a,b,c;
  path q;
  a=(0,u);  b=(2u,0);  c=(3u,1.7u);
  q:=Pathsankakkei(a,b,c);
  draw q;
%  fill q withcolor red;
```

このマクロ pathSankaku は, 経路 q を用意しておいて, q:=pathSankaku(a,b,c); のようにして使う. この場合, draw q や fill q withcolor red のようにして経路を描いたり, 塗りつぶしができるようになる.

ここで begingroup と endgroup があることによって, その間に挟まれる文のうち, 得られる経路を表わす最後の文 (ではないが文のように振る舞う) p 以外の文の計算処理をあたかもないものとして扱い, 最後の文 p のみがあり, その値を返すようになっている. 実質的には, q:=pathSankaku(a,b,c); が q:=p; となるからである.

実際に draw q , fill q withcolor red としたのが次の図 16 である.

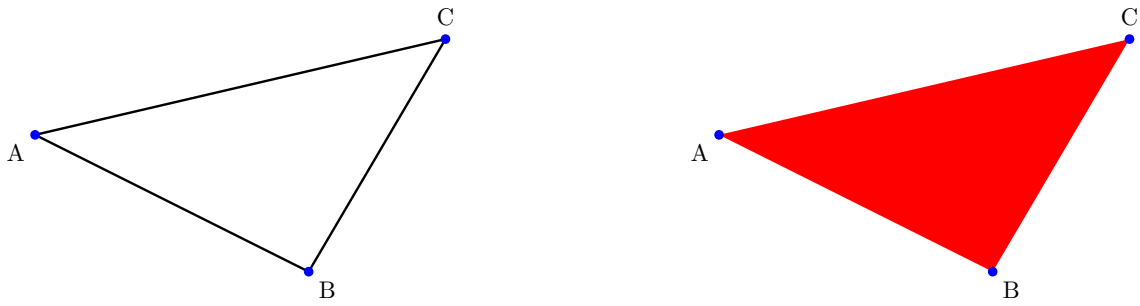


図 16: draw q(左) と fill q withcolor red(右)

5.4 マクロの例４

引数が expr のマクロを使っていたが, 引数が suffix であるマクロもある. ここでは, 点 p の右・上・左・下に長さ u だけ離れた位置に新たな点を定義し, それらを結んで正方形を作成するマクロ enws を考える. 次の 2 つのファイルのどちらでも同じ図 17 を作成できる.

```
beginfig(-1);
%%%
  def enws(suffix zz)(expr len)=
    pair zz.a,zz.b,zz.c,zz.d;
    zz.a=zz+len*right;
    zz.b=zz+len*up;
    zz.c=zz+len*left;
    zz.d=zz+len*down;
    draw zz.a--zz.b--zz.c--zz.d--cycle;
  enddef;
%%%
  u=1cm;
  pair a;
  a=(0,u);
  enws(a,u);
endfig;
end.
```

```
beginfig(-1);
%%%
  def enws(suffix zz)(expr len)=
    zz.a=zz+len*right;
    zz.b=zz+len*up;
    zz.c=zz+len*left;
    zz.d=zz+len*down;
    draw zz.a--zz.b--zz.c--zz.d--cycle;
  enddef;
%%%
  u=1cm;
  pair a;
  pair a.a,a.b,a.c,a.d;
  a=(0,u);
  enws(a,u);
endfig;
end.
```

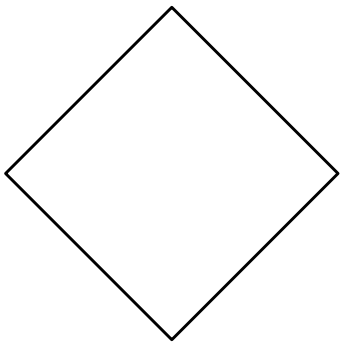


図 17: 傾いた正方形

5.5　マクロの例５

次のように, 引数が text であるマクロもある. ここでは, 点 p から作成する距離 u にある新しい点の位置を角度により定めることにする. 以下はこれを満たすマクロ SUPERenws を利用して正五角形と星を描くファイルである. これにより, 図 18 のような図形が作成される.

```
beginfig(-1);
%%%
  def SUPERenws(suffix zz)(expr len)(text drc)=
    begingroup
      pair zz[];  save k;  numeric k;  k:=0;
      for n=drc:  k:=k+1;  zz[k]=zz+len*dir(n);  endfor
      draw for n=1 upto k: zz[n]-- endfor cycle;
    endgroup
  enddef;
%%%
  u=1cm;  pair a;
  a=(0,u);
  SUPERenws(a,1cm)(18,90,162,234,306);
  SUPERenws(a,1cm)(18,162,306,90,234);
endfig;
end.
```

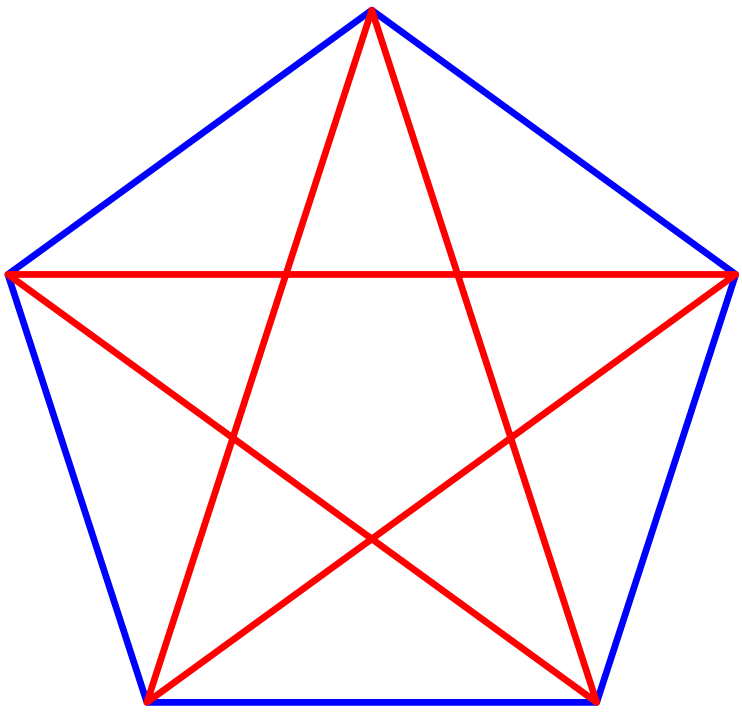


図 18: 正五角形と星 (色付き)

5.6 マクロの定義

ここまでに記述したように, マクロは <マクロ形式><マクロ名><引数列>= マクロ内容 enddef; として定義する. 正確ではないが, 簡単にまとめると以下のようになる. 次の表はマクロの形式である.

def	一連の操作をひとまとめにしたマクロを定義する.
vardef	値を返すようなマクロを定義する. 引数が 1 個の場合, 単項演算子を定めることになる.
	二項演算子を定義できる. 優先順位によって次の 3 つの定義の仕方がある.
primedef	primedef による定義は * や / と同格になる.
secondarydef	secondarydef による定義は + や - と同格になる.
tertiarydef	tertiarydef による定義は > や & と同格になる.

これらの引数については次のようなタイプがある.

expr	引数を「値」として渡す. このタイプの引数には未整理の数式を書き込むことができる. また, 変数だけでなく具体的な値も引数に指定できる.
suffix	引数を <サフィックス> として扱う. このタイプの引数には未整理の数式を使うことができない. また, この引数を <変数> 名を転用することができる.
text	命令や値を並べて引数にすることができる. このタイプの引数は実際に使うときに評価される. また, どのようなものも引数に指定できる.

マクロ名は, プリミティブ・基本マクロ (・他のマクロ) と同じでなければ何でもよい.

きちんと理解すれば, 2 次関数を描くマクロや正 n 角形を描くマクロを作成することもできる. さまざまな関数を描くマクロを作成してみてはどうだろうか.

参考文献

[1] 奥村晴彦 [改訂第 4 版] LaTeX 2_ε 美文書作成入門 2009 技術評論社

[2] http://www.ushape.org/mirror/lagendra.s.kanazawa-u.ac.jp/ogurisu/manuals/metapost/index.html

[3] http://homepage2.nifty.com/domae/

~