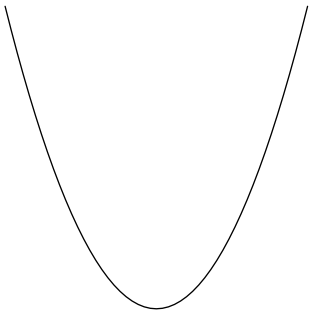
A User's Manual for MEPOTEX グラフ編

～ METAPOST in TEX ver 4.00 ～

みなも

1　最低限の命令でグラフを描こう



まずは放物線 $y = x^2 - 1$ ($-2 \leqq x \leqq 2$) のグラフ (右図) を描いてみます.

座標軸も原点「O」も座標も何もない寂しいグラフですが，これなら，下の 3 行の命令で描けます.

```
\begin{MPpic}<10mm>(2|2,3|1)
  \sendMP{xdraw() kansu(t*t-1)(-2,2,50);}
\end{MPpic}
```

では，命令の意味を説明しましょう.

● **MPpic 環境**

これは LaTeX の picture 環境を拡張したもので，図のための領域を確保したり，生成した図を貼り込んだりといった下準備をするものです.

使い方は，次の通りです.

\begin{MPpic}<単位長>(正横幅|負横幅, 正縦幅|負縦幅)
**　（各種描画命令）**
\end{MPpic}

単位長は，図中での長さ 1 の線分 (例えば，2 点 (0, 0) , (1, 0) 間の距離) が，紙面でどれくらいの長さになるかを示す値です[*1]．したがって，例えば**単位長**を 2 倍にすると，図全体が 2 倍に拡大されます．図をかいたあとで，まわりの文章との兼ね合いでもうちょっと図を小さくとか大きくといった微調整に便利です.

なお，縦方向と横方向の単位長を変えることもできます．この場合は **<横単位長, 縦単位長>** のように半角コンマで区切って指定します.

正横幅は $x \geqq 0$ の範囲の幅，**負横幅**は $x \leqq 0$ の範囲の幅を指定します．例えば，**正横幅**が 3 , **負横幅** が 2 なら，$-2 \leqq x \leqq 3$ の範囲の幅を確保します[*2]．実寸法ではなく，座標で指定していることに注意してください．例えば，先の例では $-2 \leqq x \leqq 3$ で x 座標にして 5 の幅が確保されていますが，これは，**単位長** が 1cm なら 5 cm の幅，**単位長** が 2cm なら 10 cm の幅に相当します.

正縦幅は $y \geqq 0$ の範囲の幅 (というより高さ) , **負縦幅**は $y \leqq 0$ の範囲の幅 (というより深さ[*3]) を指定します．例えば，**正縦幅**が 4 , **負縦幅** が 5 なら，$-5 \leqq y \leqq 4$ の範囲の幅を確保します.

なお，**|負横幅**，**|負縦幅**は省略可能で，省略するとこれらの幅は 0 と見なされます.

● **\sendMP{(MEPOTEX に送る命令)}**

描画命令は，TEX ではなく METAPOST に対して行います．\sendMP は，その引数に書かれた内容を METAPOST に送る命令です.

なお，METAPOST では,「1 つながりの命令」が終わるごとに ; を入れなければなりませんので注意してください[*4].

[*1] cm や mm や pt で指定します．truecm , truemm , truept なども使えます.

[*2] TEX は図全体を 1 つの「文字」のように扱います．ここで確保しているのは，この「文字」のサイズです．中身の図とは独立に指定できることからわかると思いますが，実際の図はここで確保したサイズより小さくても大きくても (確保サイズからはみ出しても) かまいません.

[*3] TEX ではアルファベット — y や g のような下にはみ出す部分をもつ文字を含む — を扱う関係上，ベースライン (p.3 の図を参照) の上側部分の縦方向のサイズを「高さ」，ベースラインの下側部分の縦方向のサイズを「深さ」と呼び，別々に扱えるようになっています．MEPOTEX のデフォルト設定では，$y = 0$ の線がベースラインに重なるようになっています.

[*4] ; が出てくるまでは複数行に渡っていても 1 文ですし，a:=a+1; b:=b+1; のように 1 行に 2 文以上を書くこともできます.

● kansu(**定義式**)(**定義域左端**, **定義域右端**, **分割数**)

METAPOST に対する命令の 1 つで，**定義式** に指定した関数のグラフを生成します．ただし，生成されるものは〈パス〉とよばれる仮想的な曲線であり，これ単独でグラフを描く命令ではありません．METAPOST ではあらかじめ〈パス〉を作り，これに対し，描画（draw）や塗りつぶし（fill）といった，描画命令を指定します．

定義式 は関数の定義式のことです．上の例では t^2-1 という関数を表しています．変数が t であることに注意してください[*5]．

定義式 に使える関数ですが，初等関数はだいたいそろっています（下表を参照）．

関数・演算	表記	関数・演算	表記	関数・演算	表記
$+$	+	$-$	-	$\times$	*
$\div$	/	$\sqrt{t}$	sqrt(t)	s^t	s**t
$\sqrt{s^2+t^2}$	s++t	$\sqrt{s^2-t^2}$	s+-+t	$\lvert t \rvert$	abs(t)
$\sin t$（度数）	sind(t)	$\cos t$（度数）	cosd(t)	$\tan t$（度数）	tand(t)
$\sin t$（弧度）	sin(t)	$\cos t$（弧度）	cos(t)	$\tan t$（弧度）	tan(t)
$\mathrm{Arcsin}\, t$（度数）	Arcsind(t)	$\mathrm{Arccos}\, t$（度数）	Arccosd(t)	$\mathrm{Arctan}\, t$（度数）	Arctand(t)
$\mathrm{Arcsin}\, t$（弧数）	Arcsin(t)	$\mathrm{Arccos}\, t$（弧数）	Arccos(t)	$\mathrm{Arctan}\, t$（弧数）	Arctan(t)
e^t	exp(t)	$\log_e t$	ln(t)	$\log_{10} t$	Log(t)

他にもいろいろありますが，とりあえずはこれで十分かと思います．計算の優先順位は，通常の数式と同じく * や / が + や - より先に行われます．この優先順序を変えるときも，通常の数式と同じく ( ) を使います．ただし，通常の数式とは違い，( ) , { } , [ ] を使い分けたりはしません．何重に重なっても，((2 + 3)*5 + 4)/2 のように ( ) のみで表記します．

なお，METAPOST では $1.2t$ のような「具体的な数値」と変数の積は 1.2t のように * を省略できます[*6]．また，関数の引数が「具体的な数値」のとき，( ) を省略できます[*7]．例えば $\sqrt{31}$ は sqrt(31) でも sqrt31 でもかまいません．

定義域左端 , **定義域右端** はグラフを描く関数の定義域で，$a \leqq t \leqq b$ でグラフを描きたいときは，**定義域左端** に a , **定義域右端** に b を指定します[*8]．

関数のグラフは一見曲線に見えますが，実はこれは細かい線分をつなげたものです．**分割数** には定義域を何個に分割して折れ線を描くかを指定します．ちなみに,「**分割数** が大きいほど精密なグラフが描ける」ように思われがちですが，METAPOST は有効数字の桁数が意外と小さいので，余り細かくしても意味がないところがあります．目安としては，通常の曲線で 50 くらい，曲線の弧長が長い場合でも 100 位の分割数で十分です．

● xdraw(**オプション**) **パス**

METAPOST に対する命令の 1 つで，**パス** を描画します[*9]．**オプション** には線の太さや色などが指定できるのですが，これについては追々紹介していきます．上の例では空白になっていますので，デフォルトの太さ（0.6pt）, デフォルトの色（黒）での描画です．

[*5] METAPOST では x がある意味予約済みなため，t を用いています．

[*6] もちろん，省略せずに 1.2*t と書いてもかまいません．

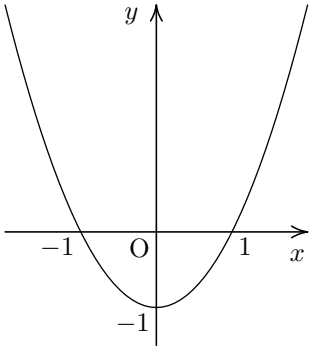
[*7] 「具体的な数値」以外にも省略できるときがあるのですが，このあたり，計算の優先順位の細かい規則と絡んできますので，深くは触れません．以降で ( ) のない例が出てきたら,「省略できるんだな」と思ってください．

[*8] ここで，ふつうに考えると **左端** < **右端** になるはずですが，プログラム的には **左端** > **右端** も受け入れます．この場合 **左端** を a , **右端** を b として，$a \geqq t \geqq b$ でグラフを描くことになります．

[*9] METAPOST には本来 draw という描画命令があるのですが，使いやすいようにいろいろ拡張した（eXpand）ので xdraw という名前です．

2　座標や座標軸を書いてみよう

今回は，前講の放物線 $y = x^2 - 1$ ($-2 \leqq x \leqq 2$) のグラフに座標軸と原点「O」，座標軸との交点の座標を書き込んでみます (右図) . これは次のような命令で描けます.




```
\begin{MPpic}<10mm>(2|2,3|1.5)
  \mptLabel{(-w,0)}[tr]<-0.5mm,-0.5mm>{$-1$}
  \mptLabel{(w,0)}[tl]<0.5mm,-0.5mm>{1}
  \mptLabel{(0,-h)}[tr]<-0.5mm,-0.5mm>{$-1$}
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[tr]<0mm,-2mm>{$x$} \mptYaxis|==>[tr]<-2mm,0mm>{$y$}
  \sendMP{xdraw() kansu(t*t-1)(-2,2,50);}
\end{MPpic}
```

では，今回付け加わった命令の意味を説明しましょう.

● **\mptLabel{配置点}[ラベル位置]<横補正，縦補正>{ラベル文字列}**

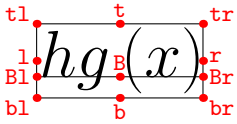
METAPOST では図中に書き込む文字列のことを「ラベル」とよんでいます. \mptLabel は図中にこの「ラベル」を入れるための命令です. **ラベル文字列** の部分がその「ラベル」の内容で，今回は座標の数値を入れるのに使っています.

まず，**配置点**ですが，これは座標の形で指定します. (1cm,2cm) のような実寸法で書くこともできますが，通常，配置点の座標は，**単位長** に追従して変化すべきです[*10]ので，実寸法でなく **単位長** で指定します.

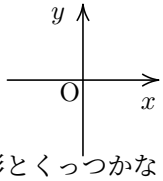
MPpic 環境で単位長を **<横単位長，縦単位長>** の形で指定した場合は，w が **横単位長** , h が **縦単位長** になります. 単位長を **<単位長>** の形で指定した場合は，w と h がともに **単位長** に等しくなります. したがって，x 座標が 2 **単位長**，y 座標が 3 **単位長**の点の座標は (2w,3h) と表記されます.

配置点は 2 次元ベクトルを表す定数でもかまいません. よく使うのは，origin でこれは (0,0) と同じ意味です.

ラベル位置 はラベルの文字列のどの位置を配置点に合わせるかを指定します. t , b , B , l (エル) , r の 1 つまたは 2 つ，あるいは c を指定します. **[ラベル位置]** は省略可能で，省略した場合は [c] と解釈されます. t はラベル上部，b は下部，B はベースライン，l は左部，r は右部，c は中央を意味します. 2 つ組み合わせると例えば bl で左下部になります.



最後に **<横補正，縦補正>** ですが，これは補正がある場合 (右上図) とない場合 (右図) と比べてもらう方がいいでしょう.



O が座標軸にぴったりくっついて，少々見栄えが悪いですね. こういった場合のラベル位置の微調整を行う部分が **<横補正，縦補正>** です. なお，**単位長** を 2 倍にして図を大きくしても,「文字を図形とくっつかないようにちょっとずらす」ときの補正量は変える必要がないため，この **<横補正，縦補正>** は，通常，実寸法で指定します. なお，**<横補正，縦補正>** は省略可能で，省略すると <0mm,0mm> の意味になります.

● **\mptXaxis|==>[ラベル位置]<横補正，縦補正>{ラベル文字列}**
\mptYaxis|==>[ラベル位置]<横補正，縦補正>{ラベル文字列}

これは座標軸を描き，さらに変数名を書き添えます. \mptXaxis が横軸，\mptYaxis が縦軸を描きます. 書き添える変数名は **ラベル文字列** に指定します. **[ラベル位置]<横補正，縦補正>** の意味は，上の \mptLabel と同じです.

なお，座標軸の範囲は，とくに指定のない限り MPpic 環境で確保した範囲いっぱいに描きます[*11]. そのため，例えば横軸の矢印の右側に x を書き添えますと，x という文字は MPpic 環境で確保した範囲からはみ出します.

図の右側が空白ならはみ出しても問題は無いのですが，図の右側に別の図を配置するなどするときは隣と重なる心配があります. こういう場合，座標軸の長さを確保した領域より短くする手があります. そのためには |==> の部分を **|開始座標==終端座標>** のようにします. 例えば，\mptXaxis|-2w==w>[tr]<0mm,-2mm>{x} だと，MPpic 環境で確保した範囲と関係なく[*12]，(-2w,0) から (w,0) まで横軸を描きます.

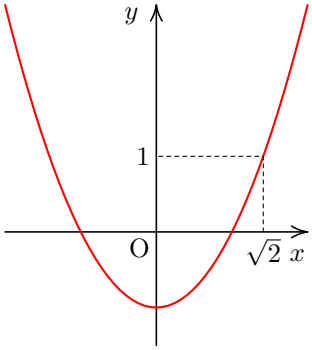
[*10] 例えば **単位長** を 2 倍にして図を 2 倍に拡大したなら，ラベルを配置する座標も 2 倍に拡大されるべきです.

[*11] 正確には「描く範囲」の初期値が「MPpic 環境で確保した範囲」というだけで,「描く範囲」を上書きすることもできます. これについては p.11 の SetArea 命令の説明を参照してください.

[*12] MPpic 環境で確保した範囲より長くも短くもできます

3　線の太さや線種を変えてみよう

今回は，前講の放物線 $y = x^2 - 1$ ($-2 \leqq x \leqq 2$) のグラフに補助線をつけて，グラフも目立つよう少し太くし，色をつけてみます (右図) . これは次のような命令で描けます.




```
\begin{MPpic}<10mm>(2|2,3|1.5)
  \mptLabel{(w*sqrt2,0)}[t]<0mm,-0.5mm>{$\sqrt{2}$}
  \mptLabel{(0,h)}[r]<-0.5mm,0mm>{1}
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[tr]<0mm,-2mm>{$x$}
  \mptYaxis|==>[tr]<-2mm,0mm>{$y$}
  \sendMP{xdraw(0.8pt,red) kansu(t*t-1)(-2,2,50);
          xdraw(0.4pt,hasen()) Pline(w*sqrt2,h);}
\end{MPpic}
```

ここで説明するのは，第 1 講で保留した xdraw のオプションと新しい METAPOST 命令 Pline です.

● xdraw では，線の太さや色，線種をオプションで指定できます[*13].

● まず，線の太さですが，これは xdraw( ) の ( ) 内に線の太さを記入することで指定できます. 例えば上の例では 0.8pt や 0.4pt が線の太さの指定です[*14].

● 次に色ですが，これは定義済みの色の名前を記入するか，RGB 値などを直接書き込みます. 例えば上の例では red がそれに当たります. すでに，線の太さなどをオプションに入れているときは，半角のコンマで区切って指定します. オプションの順序は問いません.

ここで色について説明を. コンピュータ上で色を指定する方法としては，以下の 3 つがよく知られています.

• RGB 方式 — 光の三原色である赤 (Red) , 緑 (Green) , 青 (Blue)[*15] で表現する方法です. METAPOST では (0.2,0.3,0.6) のように各成分が 0 ～ 1 であるような 3 次元ベクトル[*16]として表現します.

• CMYK 方式 — インク (?) の三原色であるシアン (Cyan) , マゼンタ (Magenta) , 黄 (Yellow) に黒 (Kuro あるいは blacK) を加えた 4 色で表現する方法です. METAPOST では (0.2,0.3,0.6,0.1) のように各成分が 0 ～ 1 であるような 4 次元ベクトル[*17]として表現します.

• HSB 方式 — 色相 (Hue) , 彩度 (Saturation) , 輝度 (Brightness)[*18] で表現する方法で，直感的に色がわかりやすい形式です. METAPOST では HSB 形式を直接扱う仕組みはないので，MEPOTEX ではマクロ hsb で RGB 形式に変換して使います. 例えば，hsb(0.2,0.9,0.7) で色相 0.2 , 彩度 0.9 , 輝度 0.7 の色を RGB の 3 次元ベクトルに変換してくれますので，hsb(0.2,0.9,0.7) を xdraw のオプションとして使えます.

METAPOST で定義済みの色として，white , black , red , green , blue が順に (1,1,1) , (0,0,0) , (1,0,0) , (0,1,0) , (0,0,1) の意味で使えます[*19]. MEPOTEX では，これに加えて LATEX の color パッケージを usenames オプション付きで使ったとき使用可能となる，以下の 68 色も定義してあります.

GreenYellow , Yellow , Goldenrod , Dandelion , Apricot , Peach , Melon , YellowOrange , Orange , BurntOrange , Bittersweet , RedOrange , Mahogany , Maroon , BrickRed , Red , OrangeRed , RubineRed , WildStrawberry , Salmon , CarnationPink , Magenta , VioletRed , Rhodamine , Mulberry , RedViolet , Fuchsia , Lavender , Thistle , Orchid , DarkOrchid , Purple , Plum , Violet , RoyalPurple , BlueViolet , Periwinkle , CadetBlue , CornflowerBlue , MidnightBlue , NavyBlue , RoyalBlue , Blue , Cerulean , Cyan , ProcessBlue , SkyBlue , Turquoise , TealBlue , Aquamarine , BlueGreen , Emerald , JungleGreen , SeaGreen , Green , ForestGreen , PineGreen , LimeGreen , YellowGreen , SpringGreen , OliveGreen , RawSienna , Sepia , Brown , Tan , Gray , Black , White .

[*13] 他に矢印をつけたりもできるのですが，話が長くなりますので，矢印についてはまた後ほど.

[*14] 単位は pt でなくても — 例えば mm などでも — かまいません. もっといえば **numeric** 値 (スカラー値) なら，それを線の太さと見なします. なお，METAPOST のデフォルト単位は bp (ビッグポイント — 1 bp は 1/72 インチ) ですから，単位をつけず 1.2 と指定すると，1.2bp の意味になります. ちなみに，METAPOST では mm などは実は「単位」ではなく「**numeric** 値定数」で，$\mathtt{mm} = \dfrac{1\mathrm{mm}}{1\mathrm{bp}}$ という比率を表し，この比率によりすべて bp 単位に置き換えているにすぎません.

[*15] JIS 規格で定義されている「赤」は red より若干暗い赤なんですが，ここでは red の訳として「赤」を使っています. 緑，青も同様です.

[*16] **color** 値といいます.

[*17] **cmykcolor** 値といいます.

[*18] 色相は 0 ～ 1 で赤 ～ 橙 ～ 黄 ～ 緑 ～ 青 ～ 紫 ～ 赤 と 1 周します. 彩度は色の鮮やかさで，0 は無彩色 (黒 ～ 灰 ～ 白) , 1 で最も鮮やかです. 輝度は色の明るさで，0 のときは色相・彩度に関わらず黒. 1 で最も明るくなります.

[*19] したがって，xdraw((1,0,0)) は xdraw(red) と同じ意味です.

xdraw のオプションに，これらの定義済みの色名，3 次元ベクトル (**color** 値) , 4 次元ベクトル (**cmykcolor** 値) のいずれかが入っていれば，線の描画にその色が使われます．これらが入っていなければ，デフォルトの色である黒が使われます．

● 線種としては，(オプション指定なしのときの) 実線の他に，破線と点線が使えます．

線種を破線にしたいときは，xdraw のオプションに hasen() を指定します[*20]．hasen() は 1.5 bp の線分と 1.5bp の空白の繰り返しからなる破線を意味します．ただし，実際には線を引く ⟨ ペン ⟩ の太さのため，線分は少し長く，空白は少し短くなります．例えば，⟨ ペン ⟩ の太さの直径 (線の太さ) が 0.4 bp のときは，線分の両側に 0.2bp ずつ線が伸びることになりますので，1.9 bp の線分と 1.1bp の空白となります[*21]．⟨ ペン ⟩ の太さが 1 bp になると，2.5 bp の線分と 0.5 bp の空白となり，かなり差が激しくなるため，この場合は線分・空白の長さを大きくとった方がいいでしょう．そのためには，hasen(**倍率**) のように，オプションの **倍率** を指定します．例えば hasen(1.2) とすると，線分と空白の長さが通常の 1.2 倍 ($1.5 \times 1.2 = 1.8$ bp) になります．

線種を点線にしたいときは，xdraw のオプションに tensen() を指定します．これは現在の ⟨ ペン ⟩ の太さ (線の太さ) の点と 2.5bp の空白[*22]を交互に並べた点線を生成します．空白を広げたいときは，hasen と同様に tensen(**倍率**) とします．

xdraw のオプションについてはとりあえずここまでで，次は新しい METAPOST 命令の Pline について説明します．

● Pline(**座標**) は **座標** に指定した点から横軸，縦軸 (座標軸) に引いた垂線の ⟨ パス ⟩ を生成する METAPOST 命令です[*23]．ちなみに，Pline(**座標**) は「横軸，縦軸両方に引いた垂線をつなげた折れ線」ですが，横軸だけに垂線を引く Pxline(**座標**) , 縦軸だけに垂線を引く Pyline(**座標**) もあります．

[*20] ( ) は省略できません．
[*21] ⟨ ペン ⟩ の太さ分はみ出すかどうかは，実は内部定数で制御できるのですが，ここでは初期設定 (はみ出す) を前提で話を進めています．
[*22] こちらも，空白は ⟨ ペン ⟩ の太さ分小さくなります．
[*23] Pline の P は射影 (projection) の P です．

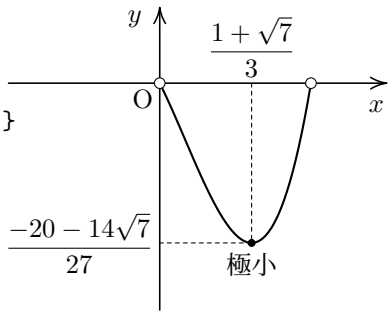
4　点に名前をつけてみよう

グラフを描いていると，1 つの点に「コメント (ラベル) をつける」「座標軸への垂線 (補助線) を引く」「白丸や黒丸をつける」など，複数の図形要素をつけることはよくあります．

このとき，例えばその点の座標が複雑だったりしますと，いちいち座標を描くのが面倒だったりしますし，入力ミスも起こしやすくなります．また，図をかいている途中で気が変わってグラフの関数を変更した場合，全部の座標を修正するのは大変です．

こういう場合，点に「名前」をつけておき，以下はその「名前」で点を指定する方法があります．これなら，座標を直接書き込むのは最初に名前を「定義」する部分だけですので，間違いも減りますし，修正する場合も 1 ヶ所の修正ですみます．

今回の例は右のような図です．




```
\begin{MPpic}<10mm>(3|2,1|3)
  \sendMP{z0=origin; z1=(w*(1+sqrt7)/3,h*(-20-14sqrt7)/27); z2=(2w,0);}
  \mptLabel{(x1,0)}[b]<0mm,0.5mm>{$\dis\frac{1+\sqrt{7}}{3}$}
  \mptLabel{(0,y1)}[r]<-0.5mm,0mm>{$\dis\frac{-20-14\sqrt{7}}{27}$}
  \mptLabel{z1}[t]<0mm,-1mm>{極小}
  \mptLabel{z0}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[tr]<0mm,-2mm>{$x$} \mptYaxis|==>[tr]<-2mm,0mm>{$y$}
  \sendMP{xdraw(0.8pt) kansu(t*t*t-t*t-2t)(0,2,50);
          xdraw(0.4pt,hasen()) Pline(z1); tanten(z0); tanten(z2); Tanten(z1)(1.5pt);}
\end{MPpic}
```

(上で \dis は \displaystyle のことです．このマニュアルのプリアンブルで \let\dis=\displaystyle と宣言しています．$\mathcal{AMS}$LaTeX をお使いの方なら，\dis\frac の部分は \dfrac でも同じ結果になります．)

さて，まずは予備知識から．

(2w,3h) のような平面上の点 — これは 2 次元のベクトルと言ってもいいですが — は，METAPOST において **pair** 値とよばれています．点，すなわち，**pair** 値に名前をつけるには，次の 2 つの方法があります．

(1) 新たな **pair** 値を宣言して使う．

例えば，Max という名前の変数を **pair** 値として宣言するには，\sendMP 内で

```
pair Max;
```

とします．これで，Max という変数を **pair** 値として使えるようになります[*24]ので，具体的な値を代入，例えば Max:=(2w,3h); とすれば[*25]，以降 Max を (2w,3h) の意味に使えます．

(2) すでに定義済みの **pair** 値マクロ z⟨ サフィックス ⟩ を用いる[*26]．

METAPOST では z⟨ サフィックス ⟩ は **pair** 値をとるマクロとして定義済みです．具体例で説明すると，z1 は (x1,y1) と同じ意味になるようマクロが定義されています[*27]．

こちらの場合は，宣言不要で，z1=(2w,3h); と書けます[*28]し，x1=2w; y1=3h; のように，x 座標，y 座標別々に値を設定することもできます．

[*24] この場合，それまでに Max が持っていた意味は — すでに命令として定義されていたり，別の変数として宣言されていたとしても — リセットされてしまいますのでご注意ください．例えばこの例で max のように小文字を使わなかったのは，max が「最大値を求める」という関数として定義済みであり，その意味を壊したくなかったからです．

[*25] METAPOST では代入は := で，等置は = で表します．ここで，代入とは「左辺の変数に右辺の (現在の) 値を設定する」という意味であり，等置とは「左辺と右辺が等しいという方程式を立てる (変数を決定する十分な個数の方程式が集まれば変数が自動で確定する)」という意味です．

上の Max:=(2w,3h); は Max=(2w,3h); でも同じ結果になりますが，Max の値を書き換える場合は代入でないといけません．

[*26] ⟨ サフィックス ⟩ とは，.A や .bl のようなピリオドと英字 (列) , または，[1] や [-0.1] のような **numeric** 値を [ ] で囲ったものです．なお，**numeric** 値のうち，符号なしの具体的な数値は [ ] を省けます．例えば，z1 は z[1] と同じ意味です．

[*27] x⟨ サフィックス ⟩ , y⟨ サフィックス ⟩ は **numeric** 値として宣言されています．

[*28] 代入の場合,「左辺は変数名でなければならない」 $\Longleftrightarrow$ 「マクロ名ではダメ」なため，z1 などに値を設定するときは，代入でなく等置を使います．x1 や y1 は変数名ですので，代入でもかまいません．

では，今回の例について詳しく見てみましょう．図形は 3 次関数のグラフの一部で，極小点を含むのですが，その座標がかなり汚いです．何回も入力したくないです．そこでその点を z1 と名付けています．

```
z1=(w*(1+sqrt7)/3,h*(-20-14sqrt7)/27);
```

ついでにグラフの端点にも z0 , z2 と名前をつけています．

z1 には次の「装飾」を施します．

● z1 から x 軸に下ろした垂線の足の位置に z1 の x 座標 $\dfrac{1+\sqrt{7}}{3}$ を書き込みます．

```
\mptLabel{(x1,0)}[b]<0mm,0.5mm>{$\dis\frac{1+\sqrt{7}}{3}$}
```

z1=(x1,y1) ですので，z1 を定めた時点で x1 と y1 も定まっていることに注意してください．

同様に y 軸に下ろした垂線の足の位置に z1 の y 座標 $\dfrac{-20-14\sqrt{7}}{27}$ を書き込みます．

● z1 のすぐ下に「極小」という文字 (ラベル) を書き込みます．

```
\mptLabel{z1}[t]<0mm,-1mm>{極小}
```

● z1 から x 軸，y 軸に，細破線で垂線を引きます．垂線のパスを生成する命令は，前講で紹介した Pline を使います．

```
xdraw(0.4pt,hasen()) Pline(z1);
```

● グラフ上 z1 の位置に点 (黒丸) を打ちます．黒丸や白丸を打つ命令はここが初出ですので詳しく説明しますと，次のようになります．

白丸を打つ命令は，　tanten(**座標**);　または　tanten(**座標**)(**大きさ**);
黒丸を打つ命令は，　Tanten(**座標**);　または　Tanten(**座標**)(**大きさ**);

座標のところは tanten(2w,3h) のように直接座標を入れてもかまいませんし，今回のように **pair** 値の変数やマクロでもかまいません．

大きさ は丸 (円) の半径です．(**大きさ**) をつけなかった場合は半径 2pt の円になります．ちなみに，黒丸は白丸より「重い」印象があるので，黒丸は白丸よりちょっと小さい方がバランスがいいかもしれません．先の例ではそれを踏まえて，白丸はデフォルトの半径 2pt で，黒丸は (**大きさ**) を 1.5pt に指定して使っています．

● 余談ですが，この講ではこれまで **pair** 値を (2w,3h) のような直交座標で表現していましたが，極座標風の表現もできます．z1 を「原点 (極) からの距離が 2w , (横軸からの) 偏角が 30°」である点と定めたいならば，

```
z1=2w*dir30;
```

とします．

dir(**角度**) は (**角度**) で指定された偏角をもつ単位ベクトルを生成し，これに距離を掛ければ実質的に極座標ということです．なお，(**角度**) の ( ) は，**角度** が具体的な数値の場合は上の例のように省略できます．

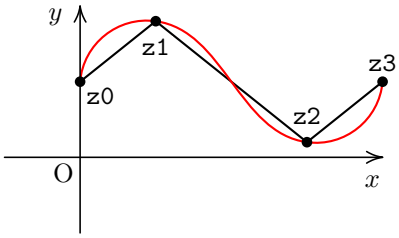
5　領域を塗りつぶしてみよう

今回は，折れ線，曲線の描画と，領域の塗りつぶしについて扱いましょう．

まず，おさらいです．METAPOST では〈パス〉という仮想的な曲線を生成し，それに対して xdraw など描画命令を作用させます．塗りつぶしの場合は，描画命令が塗りつぶし用の命令に変わります．

ところで，〈パス〉はこれまで kansu や Pline などの命令を用いて生成してきましたが，〈パス〉を構成する点から直接生成することもできます．

例えば，次の例を見てください．



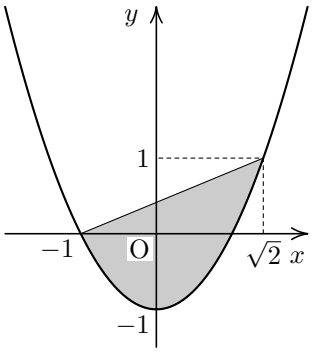

```
\begin{MPpic}<10mm>(4|1,3|1)
  \sendMP{z0=(0,h); z1=(w,1.8h); z2=(3w,0.2h); z3=(4w,h);
          xdraw(0.8pt) z0--z1--z2--z3;
          xdraw(0.8pt,red) z0..z1..z2..z3;
          Tanten(z0); Tanten(z1); Tanten(z2); Tanten(z3);}
  \mptLabel{z0}[tl]<0.5mm,-0.5mm>{\texttt{z0}} \mptLabel{z1}[t]<0mm,-2mm>{\texttt{z1}}
  \mptLabel{z2}[b]<0mm,2mm>{\texttt{z2}}       \mptLabel{z3}[b]<0mm,1mm>{\texttt{z3}}
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[tr]<0mm,-2mm>{$x$} \mptYaxis|==>[tr]<-2mm,0mm>{$y$}
\end{MPpic}
```

この例では，まず前講の方法で，点 z0 , z1 , z2 , z3 の 4 つの点を定め，これらを結んで作った〈パス〉を xdraw で描いてます．

〈パス〉を作るには点を -- または .. で結んでいきます．-- で結ぶと〈パス〉は折れ線（上の例では黒線）となり，.. で結ぶと曲線（上の例では赤線）になります[*29]．

ちなみに，kansu は内部でグラフ上の点を計算し，-- で（つまり折れ線で）つなげたものです[*30]．したがって，例えば kansu(t*t)(-2,2,50) -- z1; なら，$y = x^2$ のグラフの $-2 \leqq x \leqq 2$ の部分の曲線弧（に見える折れ線）の終端（点 $(2,\ 4)$）から点 z1 につながる折れ線が得られます．

次に塗りつぶしです．これまで通り，まずは例を挙げましょう．




```
\begin{MPpic}<10mm>(2|2,3|1.5)
  \sendMP{xfilldraw(0.8)(0.4pt) kansu(t*t-1)(-1,sqrt2,50)--cycle;
          xdraw(0.8pt) kansu(t*t-1)(-2,2,100);
          z0=(w*sqrt2,h);
          xdraw(0.4pt,hasen()) Pline(z0);}
  \mptLabel{(-w,0)}[tr]<-0.5mm,-0.5mm>{$-1$}
  \mptLabel{(0,-h)}[tr]<-0.5mm,-0.5mm>{$-1$}
  \mptLabel{(x0,0)}[t]<0mm,-0.5mm>{$\sqrt{2}$}
  \mptLabel{(0,y0)}[r]<-0.5mm,0mm>{1}
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[tr]<0mm,-2mm>{$x$} \mptYaxis|==>[tr]<-2mm,0mm>{$y$}
\end{MPpic}
```

[*29] 曲線の形状は，デフォルトのパラメータにしたがって「なるべくなめらかに与えられた点を結ぶ」ものになります．デフォルトのパラメータは変更可能ですが，話が脇道にそれますのでここでは触れません．

[*30] 曲線でなく折れ線でつないでいるのは，そのほうがきれいにグラフを描けることが多いからです．とくに，変化の激しい曲線などでは，上記の「なるべくなめらかに」が悪い方に働いて，思わぬ曲線になることが多いです．上の例でも人によっては「z0 から z1 までの部分が膨らみすぎ」と感じるかも知れません．

ちなみに，今までの例を見てもわかるとおり，50 分割程度の折れ線でも，人の目には曲線に見えますので，実用上折れ線で問題ありません．

当たり前のことですが，塗りつぶしは閉じた〈パス〉に対して行われます．

ここで，閉じた〈パス〉についてですが，METAPOST では

```
z0--z1--z2--z3--z0
```

のように〈パス〉の終点と始点が一致したからと言って「閉じている」とは見なしません．METAPOST で閉じた〈パス〉を作るには，明確に「閉じた」ことを宣言しないといけません．この閉じたことを宣言する命令が cycle で，

```
z0--z1--z2--z3--cycle
```

のように使います．この例での cycle は，〈パス〉の始点 z0 を表すと同時に,「閉じた」ことを宣言しています．

先の例では,

```
kansu(t*t-1)(-1,sqrt2,50)--cycle
```

のように使っています．kansu(t*t-1)(-1,sqrt2,50) が ($-1 \leqq x \leqq \sqrt{2}$ を 50 分割した) 51 個の点からなる折れ線であり，その終点 (点 ($\sqrt{2}$, 1) に当たる) から始点 (点 (-1, 0) に当たる) に戻る線分を追加し，〈パス〉を閉じる働きをしているのが --cycle の部分です．

閉じた〈パス〉ができれば，次は塗りつぶしです．METAPOST の本来の塗りつぶし命令は fill ですが，描画命令と同じく，ここでは MEPOSTEX で拡張された塗りつぶし命令を紹介します．

● **xfill(色) 閉じたパス**

閉じたパス で囲まれた領域を **色** に指定した色で塗りつぶします．

色 に指定できるのは，**color** 値，**cmykcolor** 値，**numeric** 値[*31]で，**color** 値，**cmykcolor** 値の場合は，そのまま指定通りの色で塗りつぶします．**numeric** 値の場合は 0 ～ 1 の値で指定し，0 が黒，1 が白，0 と 1 間の値が灰色を意味します[*32]．

● **xfilldraw(色)(境界オプション) 閉じたパス**

xfill と xdraw を合わせたような命令です．xfill は **閉じたパス** の内部を塗るだけなのに対し，xfilldraw は **閉じたパス** の内部を塗った後，境界線である **閉じたパス** 自体も描きます．**色** は塗りつぶしの色で xfill の場合と同じ使い方，**境界オプション** は境界線の太さ，色，線種などで xdraw のオプションと同じ使い方です．

ちなみに先の例では xfilldraw(0.8)(0.4pt) ですので，白 80% , 黒 20% の灰色で内部を塗りつぶし，境界線は 0.4pt の黒実線で描く設定です[*33]．

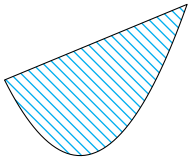
● xfill と xfilldraw の塗りつぶしには，斜線も使えます．ただし，斜線の実現には METAPOST の clip 命令 (閉じた曲線の外を消去する命令) を使っているのですが，これが dvipdfmx と相性が悪く，時々うまく PDF を作れないことがあります．METAPOST 自体に問題があるわけではなくドライバとの相性の問題ですので，うまくいかないときはあきらめてください．

さて，斜線の指定方法ですが，これは **色** の部分に **pair** 値 (2 次元ベクトル) を指定します．**pair** 値 (2 次元ベクトル) の大きさが「斜線の間隔」を，偏角が斜線の向きを表しますので，極座標っぽい表現で入れるとわかりやすいと思います．

例えば，1mm 間隔で 135° 方向のシアンの斜線を入れたい場合は

```
xfill(Cyan,2mm*dir135) パス
```

のように指定します (右図は xfilldraw で描いています) .



[*31]以前説明しましたように，**color** 値は rgb (赤・緑・青) の 3 次元ベクトル，**cmykcolor** 値は cmyk (シアン・マゼンタ・黄・黒) の 4 次元ベクトルで，各成分は 0 ～ 1 の値です．**numeric** 値は通常の数値 (スカラー) です．

[*32]もちろん値 0 に近いほど黒っぽく，1 に近いほど白っぽくなります．ちなみに内部的には，例えば 0.8 は 0.8white — 白 80% , 黒 20 % — に変換されて色指定に使われています．

[*33]もっとも，境界線のうち放物線部分は，あとでもっと太い線で上書きしているので，xfilldraw で描いた境界線が見えるのは，領域上部の線分部分だけです．

6　図を段落の横に配置してみよう

この講では，図を描く話はちょっとお休みして，図の配置を考えてみましょう．

論文なんかでは図は，配置に余裕があるページに「原則下置き」とか「原則上置き」など一定の規則で，左右中央に配置するのが一般的[*34]ですが，高校までの問題集，参考書や授業のプリントでは，図は対応する本文のすぐそば，場合によっては段落幅を短くしてその横に配置したい，ということの方が一般的です．今回はそのあたりのテクニックを説明しましょう．

まず，図を中央に配置する方法について．これは，図全体を center 環境に入れたり，別行立て数式扱いにしたり，無限伸長の空白 \hfill を使ったり[*35]で実現できます．

```
\begin{center}
  \begin{MPpic}<10mm>(3|1,3|1)
    ( 図 )
  \end{MPpic}
\end{center}
```

```
\[
  \begin{MPpic}<10mm>(3|1,3|1)
    ( 図 )
  \end{MPpic}
\]
```

```
\noindent\hfill
  \begin{MPpic}<10mm>(3|1,3|1)
    ( 図 )
  \end{MPpic}
\hfill\null
```

次に，段落の幅を短くする方法について．これは MEPOTEX の命令 \mptAke で実現できます．使い方は，例えば，段落の初めの 1 行は行幅いっぱい，次の 2 行を 6cm 短く，次の 5 行を 10cm 短くするには，

```
\mptAke{1:*,2:6cm,5:10cm}
```

という命令を段落の先頭にでも書いておきます[*36]．

段落整形かかってます

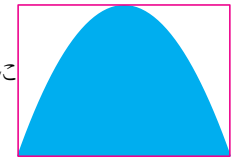
{ } 内には，**行数:縮めたい幅** という形式を，コンマで区切って書いていきます．縮めなくていい行については **縮めたい幅** のところに * を書き込みます[*37]．ちなみに，{ } 内に指定した行数の合計より実際の段落の行数が少ないときには，余った指定は無視されます．逆に，指定した行数の合計より実際の段落の行数が多いときには，一番最後の指定が繰り返し使われます．例えば，先程の例では 1 行目は縮みなし，2 , 3 行目は 6cm 短く，4 ～ 8 行目は 10cm 短くなるのですが，9 行目以降がある場合には，最後の指定の「10 cm 短い」が適用されます．なお，\mptAke* のように，* をつけて用いると，{ } 内の最後に 1:* が追加されたのと同じ意味を持ちます．

\mptAke*{1:*,2:6cm,5:10cm} ⟺ \mptAke{1:*,2:6cm,5:10cm,1:*}

つまり，1 + 2 + 5 = 8 行目までは上と同じですが，9 行目以降は「縮みなし」になります．ちなみに，気付かれているかも知れませんが，この段落には \mptAke*{1:*,2:6cm,5:10cm} がつけられています．10 cm 短い行が別行立て数式も含めて 3 行しかないように見えますが，これは「別行立て数式は 3 行分にカウントされる」という TEX の仕様のためです．

\mptAke に関して，もう 1 つ注意です．この「段落の整形」は段落が変わったらリセットされて，次の段落には引き継がれないのですが，LATEX の enumerate 環境や itemise 環境では，このリセットが掛からないようにして見出し付き段落の形状を維持しているため，enumerate 環境や itemise 環境内で，\mptAke を使うと，それ以降，環境内のすべての段落が整形されてしまいます．これを阻止し，\mptAke を使う前の状態に戻すには，\modosu という命令を，元に戻したい段落の先頭または元に戻したい項目に対応する \item の直後に置きます．

最後に，段落を整形して開けた空白に図を入れる方法についてお話しします．

まず，何も考えずに行の途中にこのように  図を入れた場合どうなるかというと，MPpic 環境で図のために確保したサイズ (マゼンタの枠) だけ，幅と高さ・深さ[*38]がとられるため，上下の行との間に大きく空白ができます．

[*34] 配置位置を探すのも LATEX にお任せで，執筆者は配置位置を気にしないのが，LATEX 本来のスタイルです．ちなみにそのための環境が figure 環境です．

[*35] \hfill は，「可能な限りいくらでも伸びる水平方向の空白」を表します．これを図の左右に入れることで，行内で左右中央に配置できます．ただし，「行末の空白は無視される」という TEX の規則で，図の右側の空白が無視されないよう，\null (何も出力しないけれども「幅 0 の文字」として機能します) をつけています．

[*36] 段落のどこに書いてもいいのですが，目立つところにないと後で見づらいでしょう．ちなみに，同じ段落で複数回指定した場合は，最後の指定が有効でそれ以外は無視されます．

[*37] もちろん * の代わりに 0mm でも同じ意味になります．

[*38] 縦方向のサイズのうち，ベースラインより上の部分のサイズを高さ，ベースラインより下の部分のサイズを深さといいます．

ここで，MPpic 環境で確保したサイズは中身の図とは無関係で，確保した領域からはみ出した図もかけることを思い出してください．極端な話 \begin{MPpic}<10mm>(0,0) のように確保サイズを 0 にすると，のように行間やまわりの文字に影響を与えず，図を書き込めます[*39]．デフォルトでは，この図を置いたところが座標の原点になるように図が配置されるのですが，これを図の左下など，別の点 — 例えば点 $(-1.4,\ -1)$ — になるように配置するには，

\begin{MPpic}<10mm>(0,0)(-1.4,-1)

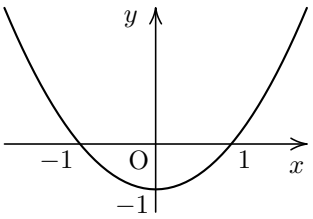
のように，MPpic 環境の最後にオプション座標 (-1.4,-1) を指定します．今回紹介したオプションも含め，MPpic 環境の書式をまとめておきますと，次のようになります．

\begin{MPpic}<単位長>(正横幅|負横幅, 正縦幅|負縦幅)(配置点横座標, 配置点縦座標)

もうおおよそ予測は立ったかも知れませんが，段落の右側に図を配置するには，\mptAke で段落の幅を短くしておき，段落の最後に無限伸張の空白 \hfill を入れ，さらにその後に，確保サイズ 0 の図を，

図本体が配置点から右方向と上方向にはみ出す $\Longleftrightarrow$ 図の左下を配置点とする

ようにすればいいことになります．実際，そのように図を配置してみましょう．配置点には赤の丸をつけています．



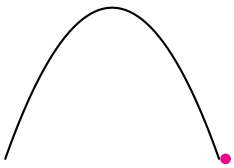
上の段落と図は，次のようなソースで生成されています．

```
\mptAke{1:48mm}
もうおおよそ予測は立ったかも知れませんが，段落の右側に図を配置するには，
\verb+\mptAke+ で段落の幅を短くしておき，段落の最後に無限伸張の空白 \verb+\hfill+ を入れ，
さらにその後に，確保サイズ 0 の図を，
$$\hbox{図本体が配置点から右方向と上方向にはみ出す} \iff \hbox{図の左下を配置点とする}$$
ようにすればいいことになります．
実際，そのように図を配置してみましょう．配置点には赤の丸をつけています．
\hfill
\begin{MPpic}<10mm,6mm>(0,0)(-2.5,-1.5)
  \sendMP{SetArea(-2,2)(-1.5,3);}
  \mptLabel{(-w,0)}[tr]<-0.5mm,-0.5mm>{$-1$} \mptLabel{(w,0)}[tl]<0.5mm,-0.5mm>{1}
  \mptLabel{(0,-h)}[tr]<-0.5mm,-0.5mm>{$-1$} \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[tr]<0mm,-2mm>{$x$}            \mptYaxis|==>[tr]<-2mm,0mm>{$y$}
  \sendMP{xdraw(0.8pt) kansu(t*t-1)(-2,2,50); xfill((0,1,0,0)) circle((-2.5w,-1.5h),2pt);}
\end{MPpic}
```

なお，座標軸 (など) を描く範囲の初期値は「MPpic 環境で確保した範囲」なのですが，今回の場合それは大きさ 0 なので,「描く範囲」としては不適切です．このような場合，SetArea 命令で「描く範囲」を上書きします[*40]．使い方は次の通りです．

SetArea(x **最小値**,x **最大値**)(y **最小値**,y **最大値**);

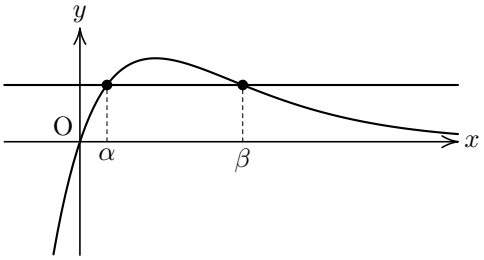
ちなみに，図を段落の右に配置するには，段落の最下行を「縮みなし」にして，\hfill で右端に飛ばし，そこを図の右端にする手もあります．この方法は段落の行数を確認してから調整する分手間は掛かりますが，図の右端を印刷する領域の右端にそろえやすいというメリットがあります．



[*39]図のために確保した領域の大きさは 0 で，マゼンタの点の位置に配置されています．実際の図は確保した領域からはみ出しています．
[*40]「MPpic 環境の確保したサイズ」は「描く範囲」の初期値に流用しただけですので，SetArea を使って「描く範囲」を上書きしても,「MPpic 環境の確保したサイズ」は変わりません．

7　曲線の交点を求めてみよう

ここでは，2 曲線の交点に，目立つよう点を打ったり，座標をつけたりすることを考えます．2 曲線の方程式を連立して直接解けるのなら，求まった座標を直接用いればよいので何ら問題は無いのですが，ここでは，交点が代数的に求まらないようなケースについて，交点を求めることを考えます．




```
\begin{MPpic}<10mm,30mm>(5|1,0.5|0.5)(0,0)
  \sendMP{path ptha,pthb;
          ptha:=kansu(t*exp(-t))(-0.35,5,100);
          pthb:=(-w,1/4h)--(5w,1/4h);
          z0=ptha intersectionpoint pthb;
          z1=reverse ptha intersectionpoint pthb;
          xdraw(0.8pt) ptha; xdraw(0.8pt) pthb;
          for n=0,1: Tanten(z[n]); xdraw(0.4pt,hasen()) Pxline(z[n]); endfor}
  \mptLabel{origin}[br]<-0.5mm,0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \mptLabel{(x0,0)}[t]<0mm,-0.5mm>{$\alpha$} \mptLabel{(x1,0)}[t]<0mm,-0.5mm>{$\beta$}
\end{MPpic}
```

上の例では，新しいテクニックがいくつか使われていますので，それらを順に説明しましょう．

● METAPOST では曲線や線分は〈パス〉とよばれています．今回，曲線の交点（2 つ）を求めるときと，曲線の描画で，同じ〈パス〉を 3 回使います．〈パス〉はすでに説明しましたように kansu などを使って作るのですが，結構煩雑なので，同じ〈パス〉の定義を何度も書くのは面倒です．そこで〈パス〉に名前をつけて繰り返し使うことができます[*41]．

〈パス〉に名前をつけるというのは，実際には「**path** 値変数を用意し，この変数に〈パス〉を代入する」ということです．

上の例では，ptha , pthb という名前の変数を **path** 値として使っています[*42]．

まず，**path** 値であることの宣言は，\sendMP 内で

```
path ptha,pthb;
```

とします．このように，複数の変数を **path** 値として宣言したいときは，半角コンマで区切ってまとめて宣言できます．

次に代入ですが，これは **pair** 値のときと同じく（\sendMP 内で）:= で OK です．

ptha は $y = xe^{-x}$（$-0.35 \leqq x \leqq 5$）　—　`ptha:=kansu(t*exp(-t))(-0.35,5,100);`

pthb は $\left(-1,\ \dfrac{1}{4}\right)$ と $\left(5,\ \dfrac{1}{4}\right)$ を結ぶ線分　—　`pthb:=(-w,1/4h)--(5w,1/4h);`

● 2 つの〈パス〉の交点を求めるには \sendMP 内で，そのものずばり intersectionpoint という命令を使います．例えば，ptha と pthb の交点を求め，それに z0 という名前をつけるには，

```
z0 = ptha intersectionpoint pthb;
```

とします．

ところで，上の例では ptha と pthb の交点は 2 つありますが，どちらが求まるのでしょう．

答は，ptha（intersectionpoint の左側に書いた〈パス〉）上で最初に現れる交点です．ptha は $y = xe^{-x}$ のグラフの $x = -0.35$ に対応する点から始めて $x = 5$ に対応する点で終わる「左から右へ」描く曲線ですので，2 つの交点のうち早く現れるのは，左の方（x 座標が小さい方）です．

では，右の方（x 座標が大きい方）の交点を得るにはどうすればいいでしょうか．METAPOST には曲線の向きを変える reverse という命令があり，これをつけた〈パス〉は始点が終点に，終点が始点に逆転します．reverse ptha であれば，$y = xe^{-x}$ のグラフの $x = 5$ に対応する点から始めて $x = -0.35$ に対応する点で終わる「右から左へ」描く曲線を意味します．これを intersectionpoint の左側に書けば右側の交点が先に現れるため，右側の交点が得られます．

[*41] ちょうど「点」（**pair** 値）に名前をつけたのと同様です．
[*42] 名前は比較的自由につけられますが，**path** 値であることがわかる名前の方が，後で見直すときわかりやすいと思います．

● 最後は「交点を求める」ことと直接関係ありませんが，覚えておくと便利な構文 (for 文) です.

先の例において求めた交点は 2 つあり，それを z0 , z1 としたのですが，これらに対して「点を打つ」「x 軸に垂線を下ろす」操作を行うことを考えます．2 点それぞれについて記述すれば

```
Tanten(z1); xdraw(0.4pt,hasen()) Pxline(z1);
Tanten(z2); xdraw(0.4pt,hasen()) Pxline(z2);
```

となりますが，これらをまとめて

```
for n=0,1: Tanten(z[n]); xdraw(0.4pt,hasen()) Pxline(z[n]); endfor
```

と書けます.

意味はおおよそわかると思いますが，: と endfor の間に書かれた内容を,「n を 0 に置き換えたもの」「n を 1 に置き換えたもの」の 2 つの内容に置き換えて解釈してくれる構文です．同じ文を 2 回書かなくてすむので，ちょっと楽になります.

もちろん，もっと多くのものをまとめることもできます．例えば，n=0,1,2,3: なら，n の部分を 0 , 1 , 2 , 3 の 4 つに置き換えた内容をまとめて書いたものになります.

ちなみに，z0 と z[0] は同じ意味でしたので，n=0 のとき，z[n] は z0 と同じ意味ですが，zn は違う意味になります．これは，zn の 2 文字で 1 つの変数名と解釈され，n に 0 が代入されないからです．これが for 文で n に [ ] がついている理由です.

● 上の例では，同じ〈パス〉を 2 交点を求める際と描画の際の複数回用いるため，あらかじめ名前をつけて利用しましたが，交点だけほしい場合は，もう少しお手軽に求める命令が METAPOST に用意されています．InSecPoint という命令[*43]で，次の 4 通りの使い方ができます.

- 2 つの陽関数のグラフの交点を，指定した定義域の範囲で求める.

  InSecPoint(1 つめの関数)(2 つめの関数)(定義域左端, 定義域右端, 分割数)

  関数 は kansu と同じく t の関数として表現します．陰関数や媒介変数表示には対応していません.
- 陽関数のグラフと線分の交点を，指定した定義域の範囲で求める.

  InSecPoint(線分の一端, 線分の他端)(関数)(定義域左端, 定義域右端, 分割数) または

  InSecPoint(関数)(線分の一端, 線分の他端)(定義域左端, 定義域右端, 分割数)

  定義域 は **関数** に対するものです．線分は単に与えた 2 点を両端とする線分です．**関数** は kansu と同じく t の関数として表現します.
- 2 つの 1 次関数のグラフの交点を，定義域の指定なしで求める.

  InSecPoint(1 つめの関数)(2 つめの関数)()

  3 つ目の ( ) 内が空っぽであることに注意してください[*44].
- 2 つの直線の交点を，定義域の指定なしで求める.

  InSecPoint(直線上の 1 点, もう 1 点)(直線上の 1 点, もう 1 点)()

  1 次関数のグラフは直線なので，本質的に上と同じですが，関数式ではなく直線上の 2 点で直線を指定する点が違います.

[*43] おわかりかと思いますが，intersectionpoint を略した名称です.

[*44] 1 次関数のグラフどうしの交点は 1 次方程式で求まるため，定義域の指定は不要ですが，一般の関数では，交点を求める方程式が代数的に解けなかったり，そもそも (対数の真数条件のように) 実数全体に拡張できない関数もあるため，定義域の指定は必須です.

8　2 曲線で囲まれた領域を塗りつぶしてみよう

前講に続き，この講でも 2 曲線の交点がきれいに求まらないときを考えます．このとき，2 曲線で囲まれた部分を塗りつぶすにはどうしたらいいでしょうか．

このとき必要なのは，2 曲線のうち，領域の境界になっている部分を抜き出し，「境界線」である「閉じたパス」を作ることです．それさえできれば，あとは通常の塗りつぶしで大丈夫です．

では，順を追って説明しましょう．

● まず，曲線の一部を抜き出す方法について．

すでに説明したように，〈 パス 〉(曲線) は，その曲線上のいくつかの点を与え，それを線分もしくは曲線で結ぶことによって作られています．例えば，ptha:=kansu(t*exp(-t))(-0.35,5,100); で生成される ptha という名前の〈 パス 〉は，$-0.35 \leqq x \leqq 5$ の区間を 100 等分してできる (両端を含め) 101 個の点を線分で結んで作られています．この 101 個の点には，出現順に 0 番から 100 番の番号がついています[*45]．このうち，15 番から 38 番までの部分を抜き出した曲線は

```
subpath (15,38) of ptha
```

で得られます．(15,38) の部分の成分は整数でなくてもかまいません．38.7 のような小数なら，METAPOST の方で，38 番と 39 番の間の点から適切なものをピックアップしてくれます．また，(38,15) なら，38 番から始まり 15 番で終わる，もとと逆向きの〈 パス 〉になります．

なお，〈 パス 〉を構成する点に割り振られた番号は，time とよばれています．動点が時間とともにこれらの点を移動していくというイメージでしょう．曲線をいわゆる軌跡ととらえた表現です．

● 次は，2 曲線の囲む部分の境界線を作る話です．

これは 2 曲線の交点[*46]が，各曲線上で何番目の点[*47]に相当するかがわかれば，subpath 命令で必要な部分を切り出せます．

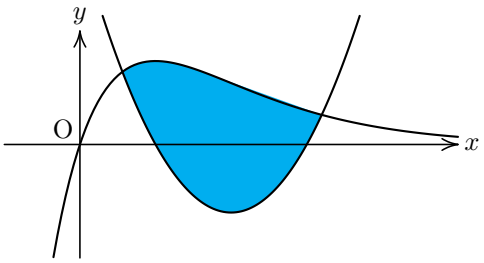
2 つの〈 パス 〉ptha , pthb の交点に対応する各パスの time を知るには，intersectiontimes という命令を使います[*48]．使い方は前講の intersectionpoint とほぼ同じで，違うところは得られるのが交点ではなく交点の times だということです．例えば，

```
(timea,timeb)=ptha intersectiontimes pthb;
```

とすると，交点に対応する ptha 上での time が timea に，pthb 上での time が timeb に記録されます．

もう 1 つの交点に対応する times も，前講のテクニックで求められます．

では，サンプルを見てみましょう．




```
\begin{MPpic}<10mm,30mm>(5|1,0.5|0.5)(0,0)
  \sendMP{path ptha,pthb;
          ptha:=kansu(t*exp(-t))(-0.35,5,100);
          pthb:=kansu(0.3(t-1)*(t-3))(0.3,3.7,100);
          (timea1,timeb1)=ptha intersectiontimes pthb;
          (timea2,timeb2)=reverse ptha intersectiontimes pthb;
          xfill(Cyan) subpath(timea1,timea2) of ptha -- subpath(timeb2,timeb1) of pthb -- cycle;
          xdraw(0.8pt) ptha; xdraw(0.8pt) pthb;}
  \mptLabel{origin}[br]<-0.5mm,0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
\end{MPpic}
```

左側の交点は ptha での time が timea1 , pthb での time が timeb1 であり，右側の交点は ptha での time が timea2 , pthb での time が timeb2 です．塗りつぶす部分の境界線は，ptha 上で timea1 から timea2 まで動き，pthb 上で timeb2 から timeb1 まで戻る点の軌跡ですので，

```
subpath (timea1,timea2) of ptha -- subpath (timeb2,timeb1) of pthb -- cycle;
```

[*45] 例えば〈 パス 〉を z1--z3--z5 のように 3 点 z1 , z3 , z5 を結んだ線分として作った場合，0 番の点は z1 , 1 番の点は z3 , 2 番の点は z5 です．パスの中での番号は，z の後ろの数字とは無関係に，出現順に割り振られることに注意してください．あと，0 番スタートです．1 番からではないので注意です．

[*46] 囲む部分があるので最低 2 個あります

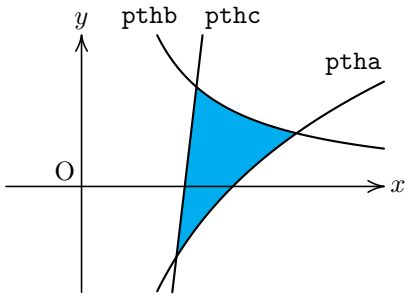
[*47] もちろん整数とは限りませんので，番号というより前述の time の方がしっくりくるかも知れません．

[*48] 命令の最後が time でなく times であることに注意してください．2 つの〈 パス 〉それぞれの時刻を与えるので，複数形です．

となります.

ちなみに，times を取得して ⟨ パス ⟩ を切り貼りして必要な境界線 (⟨⟨ パス ⟩⟩) を作る作業は，境界線に関わる曲線が増えると非常に面倒です．METAPOST にはこの辺を自動化してくれる buildcycle という命令があります.

```
\begin{MPpic}<15mm>(2|0.5,1|0.7)(0,0)
  \sendMP{path ptha,pthb,pthc;
          ptha:=kansu(ln t)(0.5,2,100);
          pthb:=kansu(1/(2t))(0.5,2,100);
          pthc:=(0.8w,h)--(0.6w,-0.7h);
          xfill(Cyan) buildcycle(ptha,pthb,pthc);
          xdraw(0.8pt) ptha; xdraw(0.8pt) pthb; xdraw(0.8pt) pthc;}
  \mptLabel{origin}[br]<-0.5mm,0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \mptLabel{point 100 of ptha}[br]<0mm,0.5mm>{\texttt{ptha}}
  \mptLabel{point 0 of pthb}[b]<-1mm,0.5mm>{\texttt{pthb}}
  \mptLabel{point 0 of pthc}[bl]<0mm,0.5mm>{\texttt{pthc}}
\end{MPpic}
```



buildcycle(ptha,pthb,pthc) がやっていることをもう少し突っ込んで説明すると，まず，pthc と ptha の交点から出発し，ptha 上を移動し，ptha と pthb の交点で，pthb に乗り換え pthb 上を移動し，pthb と pthc の交点で，pthc に乗り換え pthc 上を移動し，pthc と ptha の交点に到る経路が生成されます.

ちなみに，交点が複数あるときは，例によって初めに見つかる交点が採用されます.

注意すべきは，2 本の曲線に対し buildcycle を使うときで，この講の最初の例で

```
buildcycle(ptha,pthb)
```

とやってもうまくいきません.「pthb と ptha の交点のうち ptha 上で最初に見つかるもの」からスタートして「ptha と pthb の交点のうち pthb 上で最初に見つかるもの」に移動して，$\cdots$ となるはずですが，今挙げた 2 つの交点は同じものですから，移動は行われないからです．これを避けるためには

```
buildcycle(ptha,reverse pthb)
```

のように，reverse をつけて pthb の向きを変えます．これにより，ptha 上初めて見つかる交点と pthb 上初めて見つかる交点が異なる点となり，うまく機能します.

以下，やや発展的な内容になりますので，興味のない方は読み飛ばしてください.

例えば，ptha が 101 個の点をつないでできた ⟨ パス ⟩ だとすると，ptha はこれらの点をつなぐ 100 個の曲線弧 (または) 線分から構成されるわけですが，METAPOST では ptha intersectionpoint pthb などとして交点を探すとき，交点を曲線弧単位で探します．したがって，ptha を構成する曲線弧のうち 57 番目の曲線弧 (time が 56 ～ 57) と 81 番目の曲線弧 (time が 80 ～ 81) に交点があれば，57 番目の曲線弧の方の交点が「最初に見つかった交点」になります．ところが，57 番目の曲線弧 (time が 56 ～ 57) に 2 つの交点があった場合 (例えば time 56.2 と 56.7 が交点に対応するとき) , その 2 点の ptha 上での順序を精密に調べるのは保留し，この 2 点が pthb 上どちらが最初に現れるか調べ (もちろん曲線弧単位です) , 先に見つかった方が「最初に見つかった交点」になります．この場合，この交点が本当に ptha 上最初にある交点ではない可能性があります.

kansu で生成する ⟨ パス ⟩ は十分細かいので，同じ曲線弧に 2 交点が現れることはまずないのですが，(折れ線ではないただの) 線分は (当然) 1 つの線分から構成されるので，上記の理由で思った交点を拾ってくれないことがあります．このときは，線分の中程 (2 つの交点の間のどこか) にもう 1 つ点を取って 3 点からなる ⟨ パス ⟩ にするとうまくいきます.

9　曲線のはみ出した部分をカットしてみよう

通常，関数のグラフを描くとき，左右は x 軸の範囲，上下は y 軸の範囲を描画範囲とし，そこからはみ出さないようにグラフを描きます．例えば $y=x^2$ のような 2 次関数を $-1 \leqq y \leqq 5$ からはみ出さないように描くには，$y \leqq 5$ すなわち $x^2 \leqq 5$ を解いて，$-\sqrt{5} \leqq x \leqq \sqrt{5}$ の範囲でグラフを描けばいいのですが,「はみ出さない限界値」を代数的に計算できないときどうするかということを考えます

まず，予備知識．METAPOST は図形要素を **picture** 値変数に記録していきます．イメージ的なことをいうと，METAPOST はキャンバスのようなものをもつことができて，そこに図形を書き込んでいるのです．**picture** 値変数で特に重要なものは currentpicture という名前の変数で，xdraw などこれまで取り上げてきた描画命令は，すべてこの currentpicture という変数 (キャンバス) への描画命令です．最終的にはこの currentpicture に書き込まれた図形要素が出力されます.

さて，METAPOST には「閉じたパス」の外側をすべて消すという命令 clip があります．使い方は (\sendMP 内で) ,

clip **ピクチャー** to **閉じたパス**;

です．**ピクチャー** はここでは currentpicture , **閉じたパス** は描画領域の境界線です.

MEPOTEX にはこれを簡略化した命令 \mptClip があります．\mptClip とだけ書けば，MPpic 環境で確保した領域外を消してくれます[*49]．MPpic 環境で確保した領域とは違う位置に境界線を設定したいときは，\mptClip[**オプション**] の形で使います．**オプション** は以下を使えます.

xmin	描画領域の x 座標の最小値	xmax	描画領域の x 座標の最大値
ymin	描画領域の y 座標の最小値	ymax	描画領域の y 座標の最大値

\mptClip[xmin=0,ymin=-h] のように，コンマで区切って複数指定することもできます．指定しない項目については，MPpic 環境で確保した領域での値が使われます.

さて，これで「この講のテーマは終了」といいたいところですが，実はちょっと問題があります．clip (および，内部で clip を使っている \mptClip) は，dvipdfmx[*50] と相性が悪く，ときどき壊れた PDF をはき出しちゃいます[*51]．運悪くこの不具合に引っかかった場合は，別の方法を考えないといけません.

別の方法としては,「描画領域の境界線」と「描こうと思っている曲線」の交点を求め，前講と同様に曲線の一部[*52]を抜き出す方法です．これを自動化した ClipPATH という命令が MEPOTEX には用意されています．\sendMP 内で，次のようにして使います.

ClipPATH(**描きたいパス**)

これで，**描きたいパス** のうち,「MPpic 環境で確保した範囲」[*53]からはみ出す部分がカットされます．ただし，いくつか制約があります．まず，はみ出す部分は，⟨ パス ⟩(曲線) の描き始め，描き終わりの一方もしくは両方だけに存在するとします．何回も描画領域を出たり入ったりするケースは想定していません．また，⟨ パス ⟩ の中央の点[*54]は描画領域内にあるものとします[*55].

また,「MPpic 環境で確保した範囲」によらず，カットする範囲を直接指定する命令 ClipPath もあります.

ClipPath(**描きたいパス**)(x **最小値**,x **最大値**)(y **最小値**,y **最大値**)

とすれば，x **最小値** $\leqq x \leqq$ x **最大値** , y **最小値** $\leqq y \leqq$ y **最大値** からはみ出す部分がカットされます.

なお，**描きたいパス** の部分が陽関数のグラフ (kansu を使って描く $y=f(x)$ のグラフ) の場合は，x **最小値** , x **最大値** には kansu に指定した定義域の値をそのまま使うことが多いと思います．この場合は,

ClipPath(**描きたいパス**)()(y **最小値**,y **最大値**)

のように，x **最小値** , x **最大値** の指定を省略することもできます.

では，いくつか例を挙げてみましょう．次ページで描いているのは関数 $y=x^3-2x$ のグラフです.

[*49]もちろん，消せるのはそれまでに書いた図形だけです．\mptClip を使った後に書いた図形は消えません.

[*50]windows 版の TEX では，DVI ファイルを PDF に変換するツールとして dvipdfmx が広く使われています．明確に使った記憶がなくても，TeXworks などから自動で呼び出されて使っているケースが多々あります.

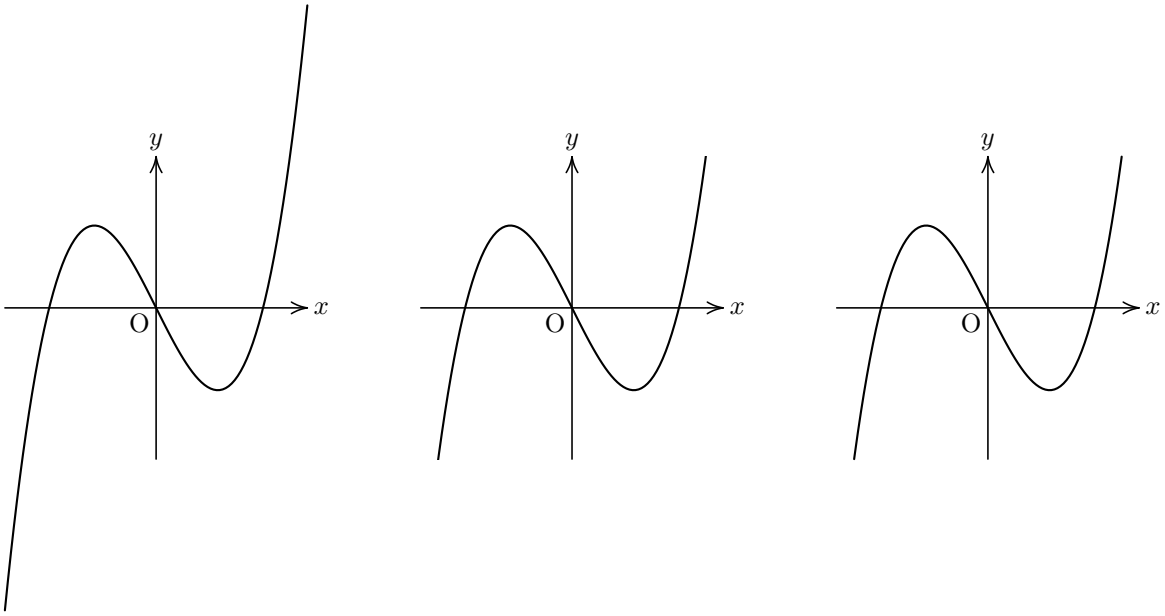
[*51]これは METAPOST の守備範囲外の不具合ですので，METAPOST 側ではどうしようもありません.

[*52]この場合，描画領域の内部にある部分.

[*53]正確には，カットする領域の境界線の初期値として「MPpic 環境で確保した範囲」を使っているだけで，p.11 で紹介した SetArea 命令を使えばこの範囲を上書きできます.

[*54]例えば 101 個の点からなる ⟨ パス ⟩ なら 50 番の点.

[*55]描画領域からはみ出す部分が ⟨ パス ⟩ の半分以上を占めるとなると，実際に描かれている部分の分割精度は半分以下になります．clip の類を使うときは，まず clip を使わず描いて，はみ出しを小さく抑えてから，仕上げに clip を使うといいでしょう.



左の図は，とくにカットせずに描きました．定義域を $-2 \leqq x \leqq 2$ としたんですが，このとき，値域は $-4 \leqq y \leqq 4$ で，描画領域に設定した $-2 \leqq y \leqq 2$ を大きくはみ出しています．

```
\begin{MPpic}<10mm>(2|2,2|2)(0,0)
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \sendMP{xdraw(0.8pt) kansu(t*t*t-2t)(-2,2,100);}
\end{MPpic}
```

これに対し中央の図は，画像全体（currentpicture）に対して \mptClip ではみ出し部分をカットしています．なお，MEPOTEX では画像に書き込む文字は TEX 側で処理していますので，文字はカットされません．座標軸の x や y の文字が残っているのはそういうわけです．

```
\begin{MPpic}<10mm>(2|2,2|2)(0,0)
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \sendMP{xdraw(0.8pt) kansu(t*t*t-2t)(-2,2,100);}
  \mptClip
\end{MPpic}
```

右の図は，$y = x^3 - 2x$ の曲線のはみ出し部分のみをカットしています．曲線（**path** 変数）をカットする命令は，前述の ClipPATH で，これで加工されたパスを描くわけですから，次のようになります．

```
\begin{MPpic}<10mm>(2|2,2|2)(0,0)
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \sendMP{xdraw(0.8pt) ClipPATH(kansu(t*t*t-2t)(-2,2,100));}
\end{MPpic}
```

ちなみに，一般の関数では，対数関数の真数条件とか，分数関数の分母が 0 でないこととか，いろいろな制約がありますので，定義域の設定や分割数の調整は自動化しづらく，上記の

ClipPATH(kansu(t*t*t-2t)(-2,2,100))

のように，煩雑な形になっているのですが，1 次関数（直線）では基本的に「端から端まで」引けばいいことと分割が要らないことから，もっとシンプルな指定が可能です．これを実装したのが MEPOTEX の Linear 命令[*56]で，

Linear(**直線の式**)

の形で使えます．例えば，ClipPATH(kansu(2t-1)(-2,2,1)) は Linear(2t-1) と同じです．

[*56] ClipPATH 命令に対する ClipPath 命令のように，Linear 命令に対しても，x , y の範囲を明示した linear 命令というものがあり，
linear(**直線の式**)(x **最小値**,x **最大値**)(y **最小値**,y **最大値**)
の形で用います．（直線の x **最小値** $\leqq x \leqq$ x **最大値** , y **最小値** $\leqq y \leqq$ y **最大値** の部分を描きます．）

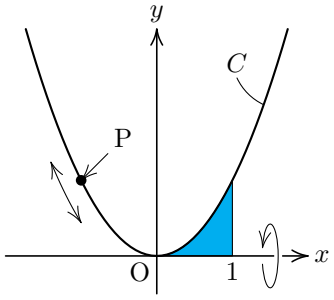
10　矢印を描いてみよう

「関数のグラフ」に必ずしも必要というわけではありませんが，時々欲しくなるのが矢印です．

例えば，グラフ上を点が動くことを表現したり，極値や変曲点の y 座標が接近していて，y 軸のそばに値を記すと重なってしまうときなど，矢印があれば便利です．

以下，例を挙げながら説明していきましょう．

まずは右の例を見てください．放物線上を動く動点を表現するのと，x 軸まわりの回転を表現するために矢印を使っています．また，注釈を参照点と結びつけるのにも矢印を使っています．




```
\begin{MPpic}<10mm>(2|2,3|0.5)
  \sendMP{xfilldraw(Cyan)(0.4pt) kansu(t*t)(0,1,30)--(w,0)--cycle;
          z0=(1.5w,0); z1=(-w,h); z2=(w*sqrt2,2h);}
  \mptLabel{(w,0)}[t]<0mm,-0.5mm>{1}
  \mptLabel{z1}[bl][dist=5mm*dir45,arrowtype=->]<0.5mm,0.5mm>{P}
  \mptLabel{z2}[b][dist=5mm*dir135,arcdir=-20]<0mm,0.5mm>{$C$}
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \sendMP{xdraw(0.8pt) kansu(t*t)(-sqrt3,sqrt3,50); Tanten(z1);
          xdraw(0.4pt,"<->") (kansu(t*t)(-1.2,-0.8,10)) shifted(-2mm,-2mm);
          odraw()(0.4pt,"->(-5)") circulararc(origin,3pt*dir195,330) yscaled4 shifted z0;}
\end{MPpic}
```

上の例では，矢印の他にもいくつか新しいテクニックが現れていますので，それもあわせて説明します．

● 矢印を描くには，xdraw のオプションに，「矢印の付加」を意味する文字列（**string** 値[*57]）を指定します．これにより，xdraw で描く〈パス〉の両端（あるいは一端）に矢印が付きます．

使える文字列は > , < , | , * , o（小文字のオー），>| , |< を - で結んだ，<-> や *->| などです．生成される矢印の形は，文字列から容易に想像されると思いますが，-> なら ⟶ , <-> なら ⟷ , *->| なら ●⟼ といった具合です．

なお，カーブのきつい曲線に矢印をつけると，↻ のように，矢印の片側が曲線に貼り付いて見えることがあります．

この場合，矢印の向きを少し補正（回転）できます．例えば，<-> 型の矢印で〈パス〉の始点側の矢印を 5° , 終点側の矢印を −5° 回転させるには，<(5)->(-5) のように，( ) で囲って補正角を記述します．（上の例では終点側を −5° 補正しています．）

● 上の例では〈パス〉の変形（変換）を行っています．METAPOST では〈パス〉に対してかなり自由な変形[*58]ができますが，ここではよく使うものを挙げます．以下，p を **path** 値（〈パス〉），z を **pair** 値（2 次元ベクトル），t を **numeric** 値（数値）とします．

まず，p shifted z で p を z だけ平行移動します．

また，p xscaled t で p を横方向に t 倍します[*59]．なお，p xscaled -1 は p を x=0 つまり y 軸に関して対称移動したものになります．

同様に，p yscaled t で p を縦方向に t 倍します．また，p scaled t なら p を原点を中心に t 倍します．

ちなみに，これらの変形に対する計算の優先順位は結構高いため，z1--z2--z3 のような〈パス〉に対して，これらの変形を行うときは，(z1--z2--z3) shifted z4 のように〈パス〉を ( ) で囲んでください[*60]．

[*57] **string** 値 とは " " で囲まれた文字列で，上の例では "<->" や "->(-10)" がそれに当たります．
[*58] 具体的にいえば，affine 変換ができます．
[*59] 〈パス〉を構成する点の x 座標が t 倍されると思ってください
[*60] さもないと，z3 のみに変形が加えられます．

● 上の例では楕円弧を描くのに円弧に変形を加えています．円弧は次の命令で描かれます．

circulararc(**中心，始点，中心角**) または circulararc(**中心，始点，終点**)[*61]

上の例では circulararc(origin,3pt*dir195,330) つまり，原点を中心とし 3pt*dir195 を始点とする中心角 330° の円弧 ○ をまず用意しています．これを，縦方向に 4 倍 (yscaled4) し，中心が z0 になるよう平行移動 (shifted z0) しています．

● 円弧が出てきましたので，次いでに円を生成する命令も紹介しておきます．

circle(**中心，半径**) または circle(**中心，通過点**)

で，円の ⟨ パス ⟩ が生成されます．(実は p.11 でこっそり使っています．)

● 次に，回転を表す矢印をじっくり見てみましょう．x 軸と立体的に交差していることを表すために，x 軸が切れているのがわかるでしょうか．これは，先に太めの白線[*62]で ⟨ パス ⟩ を描いた後，本来の太さの線で ⟨ パス ⟩ 描くことで実現しています．これを一気にやってくれる命令が odraw で，

odraw(**始点除外，終点除外，消し幅**)(**オプション**)

の形で使います．このうち **オプション** の部分は xdraw の **オプション** と同じです．

ちなみに，odraw で描く線が，すでに描かれた線に (立体交差ではなく直接) つながるとき，相手の線が切れたらまずいので，消さない部分を指示するのが (**始点除外，終点除外，消し幅**) の部分です．例えば，始点付近の 10 % , 終点付近の 20 %[*63] を除外し，消す線 (白の太線) の幅を 5pt にするなら，(0.1,0.2,5pt) と指定します．

なお，(**始点除外，終点除外，消し幅**) は省略可能で，( ) 内の数値が 2 個なら (**始点除外，終点除外**)[*64] の意味になり，( ) 内の数値が 1 個なら (**始点除外**) の意味になります．すべて省略するなら () です．

● 図に座標の数値や注釈等をつけるには \mptLabel を使えばいい，ということはすでに説明しました．

ところで，文字が混み合った場合，参照点 (注釈等を入れたい点) から離れた位置に文字を入れ，参照点と文字を矢印などつないだりすることがあります．このような場合は \mptLabel の **オプション指定** が使えます．この **オプション指定** は，次の位置に入れます．

\mptLabel{**配置点**}[**ラベル位置**][**オプション指定**]<**横補正，縦補正**>{**ラベル文字列**}

オプション指定 にはいろいろなものを入れられるのですが，さしあたってよく使うのは次の 4 つです．

• dist — 参照点から文字を書き込む位置の移動量を **pair** 値で指定します．例えば，dist=5mm*dir45 なら，45° の向きに 5mm の移動量です[*65]．

• arrowtype — 矢印の形状を指定します．指定の仕方は xdraw のオプションと同様です．ただし，こちらは arrowtype=-> のように使い，" " をつけません．また，向きの補正は次の arrowdiradj を使います．

• arrowdiradj — 矢印の向きを補正します．終点側の矢印のみ補正するときは arrowdiradj=10 のように数値 (角度) を，始点側と終点側両方の矢印を補正するときは arrowdiradj={10,5} のように 2 つの数値 (角度) を半角コンマで区切り { } で囲います．

• arcdir — 参照点と文字を書き込む位置を結ぶ線を曲げます．具体的には文字を書き込む点から参照点まで，まっすぐ行かずに，arcdir に指定した角度だけずれた向きに出発し，カーブを描いて参照点に到ります．

[*61] **終点** を指定した場合，**始点** から **終点** に向かって反時計まわりに回転する円弧となります．なお，**終点** の情報は **中心** と **終点** を結ぶ動径のみが意味を持ちますので，半径が始点側と違っていても大丈夫です．つまり，半径の情報は始点側の値のみが使われます．

[*62] 正確に言うと，background という色で描いています．background はデフォルトで白 (white) です．

[*63] 例えば，101 個の点からなる (100 個の区間からなる)⟨ パス ⟩ の場合，10 % は 10 区間，20% は 20 区間というように解釈してください．

[*64] **消し幅** を省略したときのデフォルトは 3pt です．ついでに，**始点除外** , **終点除外** のデフォルトは 0 (つまり消し線を端まで引く) です．

[*65] x 成分 , y 成分で指定するときは，dist={(2mm,3mm)} のように { } で囲みます．

11　いろいろな関数のグラフを描いてみよう

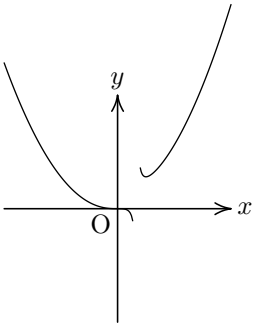
これまでの講では，グラフを描くために必要なスキルの説明が中心で，グラフそのもののバリエーションは乏しかったのですが，この講ではそこを補っていきましょう．

具体的には，いろいろな関数のグラフの描き方 — 媒介変数表示や極方程式も含めて — を紹介していきます．

11.1　有理関数

$y=\dfrac{x^3}{2x-1}$ のような関数を描いてみます．とくに，分母が 0 になる x があるときは，うまく回避しないとオーバーフローエラー[*66]が起こります．具体的には次のようにします．描画領域は $-3 \leqq x \leqq 3$, $-3 \leqq y \leqq 3$ としておきます．

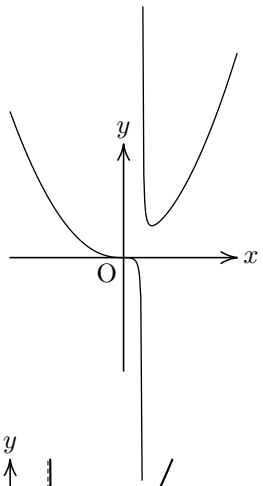
まず，$x=\dfrac{1}{2}$ で分母が 0 になるわけですので，定義域を $x<\dfrac{1}{2}$, $x>\dfrac{1}{2}$ に分割して，2 本の曲線 (パス) としてグラフを完成させます．



漸近線 $x=\dfrac{1}{2}=0.5$ 付近で $y=\pm 3$ に対応する x の値は，3 次方程式の解ですので直接求めるのは厳しいと思います．そこで，ちょっとはみ出すくらいのグラフを描いて，前講で扱った ClipPATH で不要部分をカットすることにします．

ところで,「ちょっとはみ出す」くらいの x の値はいくらぐらいでしょう．これも，直接計算は難しいので，試行錯誤で決めるのが手っ取り早いと思います．まずは，$-3 \leqq x \leqq 0.4$, $0.6 \leqq x \leqq 3$ でグラフを描かせてみます (右図) ．「試し」ですので，最低限の命令で十分です．

```
\begin{MPpic}<5mm>(3|3,3|3)(0,0)
  \sendMP{xdraw() kansu(t*t*t/(2t-1))(-3,0.4,100); xdraw() kansu(t*t*t/(2t-1))(0.6,3,100);}
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
\end{MPpic}
```

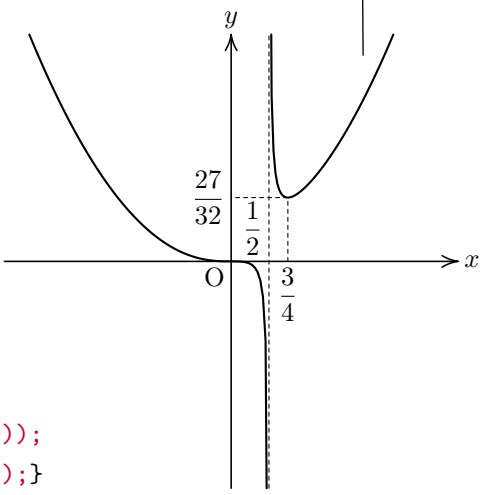


実際描いてみると，まだ全然漸近線に接近できていないので，もうちょっと試行錯誤してグラフがはみ出るくらいの値を探ります．$-3 \leqq x \leqq 0.49$, $0.51 \leqq x \leqq 3$ だといい感じにはみ出ています (右図) ．そこで，これをベースに本番のグラフを描いていくことにします．

今回のグラフに書き込みたいものは，以下の通りです．

- グラフ本体 (⟨ パス ⟩2 つ) を太実線で．
- 極値等を書き込み，破線でグラフと結ぶ．
- 漸近線 $x=\dfrac{1}{2}$ を破線で[*67]．

これらを踏まえてソースとグラフは次のようになります．




```
\begin{MPpic}<10mm>(3|3,3|3)(0,0)
  \sendMP{z0=(3/4w,27/32h);}
  \mptLabel{(1/2w,0)}[br]<-0.5mm,0.5mm>{$\dis \frac{1}{2}$}
  \mptLabel{(x0,0)}[t]<0mm,-0.5mm>{$\dis \frac{3}{4}$}
  \mptLabel{(0,y0)}[r]<-0.5mm,0mm>{$\dis \frac{27}{32}$}
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$}
  \mptYaxis|-3h==3h>[b]<0mm,0.5mm>{$y$}
  \sendMP{xdraw(0.4pt,hasen()) Pline(z0);
          xdraw(0.4pt,hasen()) Vline(1/2)()();
          xdraw(0.8pt) ClipPATH(kansu(t*t*t/(2t-1))(-3,0.49,100));
          xdraw(0.8pt) ClipPATH(kansu(t*t*t/(2t-1))(0.51,3,100));}
\end{MPpic}
```

[*66] 桁あふれ，つまり，計算能力を超えた大きさの数が現れたというエラーです．METAPOST の扱える数値は以外と小さいので，割とすぐにオーバーフローします．

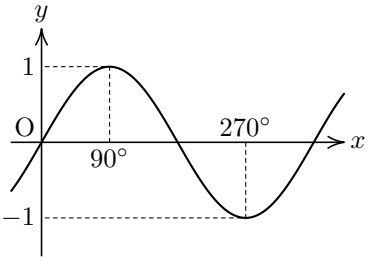
[*67] 鉛直線は Vline(x **の値**)(y **最小値**)(y **最大値**) で生成できます．y **最小値** と y **最大値** は省略可能で，省略すると「確保領域」の端まで引きます．同様に，水平線は Hline(y **の値**)(x **最小値**)(x **最大値**) で生成できます．

11.2　三角関数

三角関数は，横軸を度数法にするか弧度法にするかで少し違ってきます．

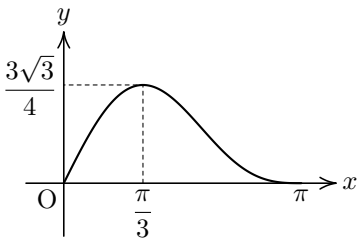
まず，度数法の方ですが，こちらは横軸の数値が結構大きくなること（したがって **横単位長** を小さくとること），三角関数が sind，cosd のように d がついた名称になることに注意が必要です．あと，$y=\tan x$ 等では途中でグラフが切れる[*68]ことにも注意です．

サンプルは次の通りです（ソースの色分けは先程と同様です）．




```
\begin{MPpic}<0.1mm,10mm>(400|40,1.5|1.5)(0,0)
  \sendMP{z0=(90w,h); z1=(270w,-h);}
  \mptLabel{(x0,0)}[t]<0mm,-0.5mm>{$90^\circ$}
  \mptLabel{(x1,0)}[b]<0mm,0.5mm>{$270^\circ$}
  \mptLabel{(0,y0)}[r]<-0.5mm,0mm>{1} \mptLabel{(0,y1)}[r]<-0.5mm,0mm>{$-1$}
  \mptLabel{origin}[br]<-0.5mm,0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \sendMP{xdraw(0.8pt) kansu(sind t)(-40,400,100);
          for n=0,1: xdraw(0.4pt,hasen()) Pline(z[n]); endfor}
\end{MPpic}
```

弧度法の場合，**単位長** は縦横共通でもかまいませんが，横方向の座標に π の整数倍が頻出します．MEPOTEX の場合，PI:=3.14159; と定義してありますので，これを使って座標等を記述すると楽でしょう．

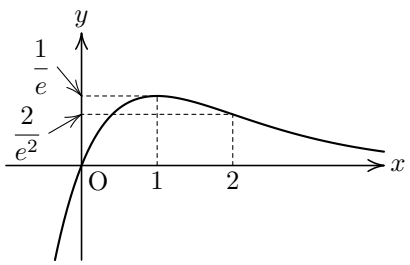



```
\begin{MPpic}<10mm>(3.6|0.5,2|0.7)(0,0)
  \sendMP{z0=(w*PI/3,h/4*3sqrt3);}
  \mptLabel{(x0,0)}[t]<0mm,-0.5mm>{$\dis \frac{\pi}{3}$}
  \mptLabel{(w*PI,0)}[t]<0mm,-0.5mm>{$\pi$}
  \mptLabel{(0,y0)}[r]<-0.5mm,0mm>{$\dis \frac{3\sqrt{3}}{4}$}
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \sendMP{xdraw(0.8pt) kansu(sin t*(1+cos t))(0,PI,100); xdraw(0.4pt,hasen()) Pline(z0);}
\end{MPpic}
```

11.3　指数関数・対数関数

指数関数，対数関数で，とくに自然対数の底 e が関わってくるものについては，関数だけでなく座標の指定にも exp や ln が必要になってきます．あと，はみ出しのカットが必要なことも多いでしょう．

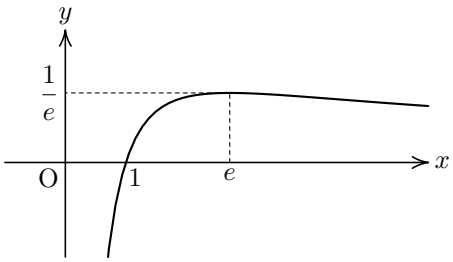
まずは 1 つ目の例として，$y=xe^{-x}$ のグラフを描いてみます．$x\to-\infty$ で $y\to-\infty$ ですが，実際にはかなり早い段階で描画範囲から飛び出してしまいます．あと，縦横比が 1 : 1 では $x>0$ の部分がかなり平べったくなるので，適当に比率を変えることをおすすめします．比率を変えても極値と変曲点の y 座標が接近していますので，ちょっと離れた位置に座標を書いて矢印で結んでいます．

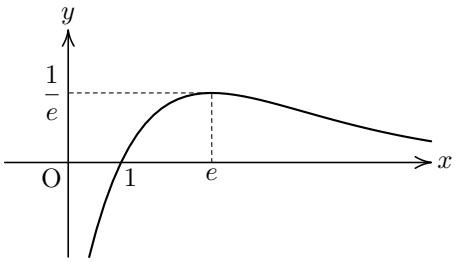



```
\begin{MPpic}<10mm,25mm>(4|1,0.7|0.5)(0,0)
  \sendMP{z0=(w,h*exp(-1)); z1=(2w,2h*exp(-2));}
  \mptLabel{origin}[tl]<0.5mm,-0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \mptLabel{(x0,0)}[t]<0mm,-0.5mm>{1}
  \mptLabel{(x1,0)}[t]<0mm,-0.5mm>{2}
  \mptLabel{(0,y0)}[r][dist=5mm*dir130,arrowtype=->]<-0.5mm,0mm>{$\dis \frac{1}{e}$}
  \mptLabel{(0,y1)}[r][dist=5mm*dir-150,arrowtype=->]<-0.5mm,0mm>{$\dis \frac{2}{e^2}$}
  \sendMP{xdraw(0.8pt) ClipPATH(kansu(t*exp(-t))(-0.5,4,100));
          for n=0,1: xdraw(0.4pt,hasen()) Pline(z[n]); endfor}
\end{MPpic}
```

[*68] このあたりの対処法は有理関数のときと同じですので，そちらを参照してください．

対数関数の例も一つ挙げましょう．$y=\dfrac{\log x}{x}$ のグラフです．ちなみに，下の 2 つの図のどっちが正しいグラフでしょう．





正解は左の図なんですが，これを例えば「模範解答」として載せると，たいてい「この図は正しくない」とクレームが付きます．理由は，

$$\text{左の図では関数の特徴である } \lim_{x\to+\infty}\frac{\log x}{x}=0 \text{ が反映されていない}$$

からだそうなんですが，実は $\lim\limits_{x\to+\infty}\dfrac{\log x}{x}$ の収束は非常に遅く，$x=100$ の時点でも関数の値は極大値の $\dfrac{1}{10}$ より大きいような状態です．ただ，この「収束の速さ」は大学入試で問われることはまずなく，「0 に収束する」という結果だけグラフに反映させようとするため，「左の図は正しくない」となるのです．

この図に限らず，実際グラフを「正しく」描くと都合の悪いこと — 2 つの極値が近すぎて識別できない，2 曲線が近すぎて識別できない，etc — は多々起こります．こういう場合，別の関数をもってきて代用することになります．ちなみに，右のグラフは $a=\dfrac{1}{e-1}$ に対し，$y=(x-a)e^{-(x-a)}$ のグラフ[*69]です．

上の 2 つのグラフのソースは次のようになります．

```
\begin{MPpic}<8mm,25mm>(6|1,0.7|0.5)(0,0)
  \sendMP{z0=(w*exp1,h*exp(-1));}
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{0} \mptLabel{(w,0)}[tl]<0mm,-0.5mm>{1}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \mptLabel{(x0,0)}[t]<0mm,-0.5mm>{$e$} \mptLabel{(0,y0)}[r]<-0.5mm,0mm>{$\dis \frac{1}{e}$}
  \sendMP{xdraw(0.8pt) ClipPATH(kansu(ln t/t)(0.5,6,50));
          xdraw(0.4pt,hasen()) Pline(z0);}
\end{MPpic}

\begin{MPpic}<12mm,25mm>(4|0.7,0.7|0.5)(0,0)
  \sendMP{numeric a; a:=1/(exp1-1); z0=(w*(a+1),h*exp(-1));}
  \mptLabel{origin}[tr]<-0.5mm,-0.5mm>{0} \mptLabel{(w*a,0)}[tl]<0mm,-0.5mm>{1}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \mptLabel{(x0,0)}[t]<0mm,-0.5mm>{$e$} \mptLabel{(0,y0)}[r]<-0.5mm,0mm>{$\dis \frac{1}{e}$}
  \sendMP{xdraw(0.8pt) ClipPATH(kansu((t-a)*exp(-t+a))(-0.5,4,50));
          xdraw(0.4pt,hasen()) Pline(z0);}
\end{MPpic}
```

11.4　媒介変数表示された曲線

次は，媒介変数表示された曲線を描いてみましょう．変更点は kansu の代わりに kansuP を使う[*70]ことです．

例えば，**サイクロイド**

$$x=t-\sin t\,,\quad y=1-\cos t$$

を描きたいなら，曲線 (⟨ パス ⟩) を

```
kansuP(t - sin t,1 - cos t)(0,2PI,100)
```

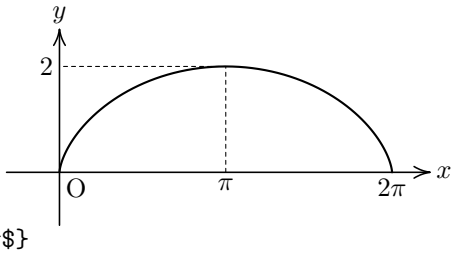
[*69] 増減の様子が同様で，もっと収束の速い関数ということで $y=xe^{-x}$ を考え，これを (x 軸との交点の x 座標) : (極大値をとる x) $=1:e$ となるよう平行移動しました．なお，$y=(x-a)e^{-(x-a)}$ のグラフを下方向に延長するとそのうち y 軸と交わって，$y=\dfrac{\log x}{x}$ のグラフの代用品としてはまずいのですが，上図のように適当なところで切り上げると，あまり気になりません．

[*70] P はパラメータのつもりです．

のようにして作ります.

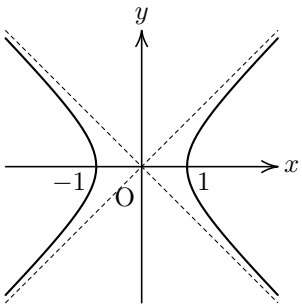
kansuP の直後の（ ）の中に，x , y を表す関数[*71]を半角コンマで区切って書くだけで，それ以外の使い方は kansu と同じです.

座標軸なども含めた完全なソースと実際の図は次の通りです.




```
\begin{MPpic}<7mm>(7|1,2.7|1)(0,0)
  \sendMP{z0=(w*PI,2h);}
  \mptLabel{origin}[tl]<0.5mm,-0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \mptLabel{(x0,0)}[t]<0mm,-0.5mm>{$\pi$} \mptLabel{(0,y0)}[r]<-0.5mm,0mm>{2}
  \mptLabel{(w*2PI,0)}[t]<0mm,-0.5mm>{$2\pi$}
  \sendMP{xdraw(0.8pt) kansuP(t - sin t,1 - cos t)(0,2PI,100);
          xdraw(0.4pt,hasen()) Pline(z0);}
\end{MPpic}
```

ちなみに，$x = f(y)$ のグラフは $x = f(t)$, $y = t$ と媒介変数表示された曲線と考えられますので，kansuP で描けます．例えば，双曲線 $x^2 - y^2 = 1$ は $x = \pm\sqrt{y^2 + 1}$ と表せるため，次のように描けます.




```
\begin{MPpic}<6mm>(3|3,3|3)(0,0)
  \mptLabel{origin}[tr]<-0.5mm,-2.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \mptLabel{(w,0)}[tl]<1mm,-0.5mm>{1}
  \mptLabel{(-w,0)}[tr]<-1mm,-0.5mm>{$-1$}
  \sendMP{xdraw(0.8pt) ClipPATH(kansuP(t++1,t)(-3,3,50));
          xdraw(0.8pt) ClipPATH(kansuP(-(t++1),t)(-3,3,50));
          xdraw(0.4pt,hasen()) (-3w,-3h)--(3w,3h);
          xdraw(0.4pt,hasen()) (-3w,3h)--(3w,-3h);}
\end{MPpic}
```

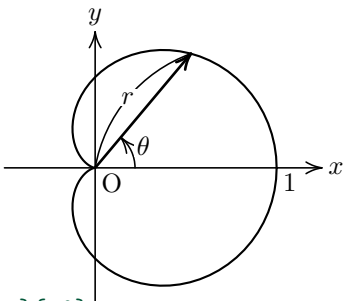
なお，媒介変数表示された曲線の場合，描画領域からのはみ出しは上下だけとは限らず，左右にはみ出す場合もあります．したがって，ClipPath を使うときは，描画領域の指定を（ x 方向を省略せず ）x 方向，y 方向ともに行う必要があります.

11.5　極方程式で表された曲線

最後に，極方程式で表された曲線を描いてみましょう.

極方程式とは，平面上に極とよばれる点（ O とします ）および始線とよばれる半直線（ O が端点です ）を設定し，曲線上の点を極からの距離 r と始線からの回転角 θ で表し，r と θ の関係式 $r = f(\theta)$ で曲線を表すものです.

例えば，次の曲線は $r = 1 + \cos\theta$ で表される**カージオイド**とよばれる曲線です.




```
\begin{MPpic}<12mm>(2.5|1,1.5|1.5)(0,0)
  \sendMP{z0=w*(1+cosd50)*dir50;}
  \mptLabel{origin}[tl]<0.5mm,-0.5mm>{O}
  \mptXaxis|==>[l]<0.5mm,0mm>{$x$} \mptYaxis|==>[b]<0mm,0.5mm>{$y$}
  \mptLabel{(2w,0)}[tl]<0.5mm,-0.5mm>{1}
  \mptFromTo[label=$r$]{origin}{z0}
  \mptAngleMark[label=$\theta$,arrowtype=->,arrowdiradj=-10]{right}{origin}{z0}
  \sendMP{xdraw(0.8pt) kansuR(1+cosd t)(0,360,100);
          xdraw(1pt,"->") origin--z0;}
\end{MPpic}
```

[*71] kansu と同じく，t を変数として記述します

極方程式で表された曲線 (⟨ パス ⟩) を生成するのは kansuR または kansuRR です[*72]．kansuR は角度が度数法，kansuRR は角度が弧度法です．使い方は，kansuR , kansuRR の直後の ( ) に極方程式を t の式で指定することを除いて，kansu と同様です．

ちなみに，上の例では「曲線上の点について極座標の r と θ」を表す矢印を書き込んでいます．矢印自体は xdraw のオプションで書いていますので，すでに解説済みですが，矢印の大きさを書き込む \mptFromTo と，角度を書き込む \mptAngleMark はここが初出ですので，すこし解説しましょう．

● \mptFromTo は $\overset{r}{\frown}$ のような長さを表す記号 (曲線弧) を生成する命令で，基本的な使い方は，

\mptFromTo[**オプション**]{**始点**}{**終点**}

です．このうち，**始点** と **終点** は曲線弧の始点と終点ですからわかりやすいと思います[*73]．[**オプション**] にはいろいろなものが指定できる[*74]のですが，ここではよく使うものに限定して紹介しましょう．

- label — 曲線弧に入れる文字を指定します．コンマを含む文字列のときは，文字列全体を { } で囲ってください．
- arcdir — 曲線弧をどれだけ膨らませるかを指定します．具体的には **始点** から **終点** へのベクトルに対し，曲線弧の接線がなす角度を指定します．指定がない場合は arcdir=30 (つまりなす角 30°) と解釈されます．

● \mptAngleMark は角度マークを生成する命令で，例えば，点 A , B , C を表す **pair** 値を順に z.A , z.B , z.C とするとき，∠ABC[*75] を表すマークを生成するには

\mptAngleMark[**オプション**]{z.A}{z.B}{z.C}

とします．[**オプション**] にはいろいろなものが指定できるのですが，こちらもよく使うものに限定して紹介しましょう．

- markradius — 角度マークを円弧に見立てたときの半径を指定します．指定のないときは markradius=15pt と解釈されます．
- arrowtype — 角度マークに矢印をつけます．指定の仕方は \mptLabel のときと同様です．
- arrowdiradj — 角度マークにつけた矢印の向きを補正します．指定の仕方は \mptLabel のときと同様です．
- label — 角度マークに添える文字を指定します．コンマを含む文字列のときは，文字列全体を { } で囲ってください．
- labeldist — label に指定した文字の中心を，マークからどれくらい離すかを指定します．指定のないときは labeldist=5pt と解釈されます．
- arcdir — 角度マークの曲線弧をどれだけ膨らませるかを指定します．具体的には角を作る半直線と角度マークの端点での接線のなす角を指定します．指定がない場合は arcdir=90 (つまり角を作る半直線と垂直方向) と解釈されます．

11.6 こぼれ話

$0 \leqq t \leqq \dfrac{\pi}{2}$ で $\cos t \geqq 0$ なので $\sqrt{\cos t}$ は問題なく定義されそうですが，kansu(sqrt(cos t))(0,PI/2,50) とするとエラーが生じます．なぜでしょう．

当たり前の話ですが，コンピュータの扱う数値は有限桁です．無限小数であろうと無理数であろうと有限の桁でばっさり切って計算します．上の例においては，PI/2 は内部で四捨五入され 1.5708 となるため，$\dfrac{\pi}{2}$ よりちょっと大きくなります．そのため平方根内が負になり，エラーが出るのです．ちなみに，この場合，定義域の上限を PI/2 よりちょっと小さくすればいいのですが，METAPOST にはこういう目的で使う定数として epsilon と eps があります．epsilon は 2^{-16} , eps は 2^{-11} です．kansu(sqrt(cos t))(0,PI/2-eps,50) ならエラーは出ません．

あと，コンピュータは数値を 2 進法で表しますので，10 進法で有限小数だからといって安心はできません．例えば，10 進法の 0.1 は 2 進法だと $0.0001100110011\cdots_{(2)}$ なので，無限小数であり，コンピュータ内では四捨五入されてしまいますので，例えば METAPOST で 0.1 を 10 回足すと 1.00006 となってしまいます．

[*72] 極座標は Polar coordinate ですが，これだと頭文字が Parameter と同じく P なので，極座標を通常 (r, θ) と表すことをもじって R を命令の末尾につけています．

[*73] ちなみに，曲線弧は始点から終点に向かって左側に張り出します．

[*74] 複数のオプションを指定するときは，半角のコンマで区切ります．

[*75] 半直線 BA から BC へ反時計まわりに角度マークを描きますので，z.A と z.C の順番に注意してください．