36　2 つの関数

ここでは，以前に Python で定義した関数とは異なる『関数』について解説します。まず，Python における関数の復習をします。Python における関数は，いくつかの処理をひとかたまりにして名前を付けたものでした。たとえば，次のように関数を定義しました：

```
1  def fib(n):                  # 関数fibの定義，引数n
2      a, b = 0, 1              # ここから
3      for i in range(n):       #
4          a, b = b, a+b        # ここまでが一連の処理
5      return a                 # 上の処理が終わったらaの値を返す
6
7  for i in range(1,10):        #i=1,…,9 に対して
8      print fib(i)             # fib(i)をプリント
9
10 print type(fib)              #fibのデータの型を表示
```

実行結果の例

```
1 1 2 3 5 8 13 21 34
<type 'function'>
```

上のプログラムでは，関数 fib が呼び出されると，そのときの引数 n に対して 2 行 ~4 行の処理を行い，それが終わると a の値を返す (return) という事をしています。

はじめて Python の関数を習ったときに，違和感を憶えた人も多いと思います。それは，高校以前の数学で『関数』と呼ばれているものは**文字式**として定義されていて，方程式を解いたり微分したりといった文字式としての取り扱い方を学習するからだとおもいます。たとえば

$$4x+3, \quad x^2, \quad \sin(x), \quad \frac{1}{x^2+1} \tag{6}$$

はどれも文字式による関数です。ここでは，x は関数の変数 (variable) や不定元 (indeterminate) と呼ばれます。これらの関数は，x に具体的な数値を代入すれば，計算によりその関数の値が定まりますが，Python の関数のように計算手順を記述したものではありません。そして，これらは x が何者であるかはとりあえずは特定せずに x の文字式としての扱われます。

Sage では，文字式としての関数に対応するものを定義することができます。

文字式としての関数の定義 1

```
1 var('x')          #  xを変数＝不定元にする宣言
2 f = x^2           #  文字式x^2を変数fに代入
```

上のプログラムの 1 行目は文字 x を変数として取り扱いますという宣言です。これにより x の文字式が扱えるようになります。2 行目で関数 $f = x^2$ を定義しています。

上に続いて次を実行します

```
1 print f             #  fをプリント
2 print f(x=5)        #  x=5のときのfをプリント
3 print type(f)       #  fの型を調べる
4 print type(x)
```

実行結果の例

```
x^2
25
<type 'sage.symbolic.expression.Expression'>
<type 'sage.symbolic.expression.Expression'>
```

このようにして定義した関数 f の特定の x に対する値を計算するときには 2 行目のように書きます。3,4 行目の結果は，f と x のデータの型は sage.symbolic.expression.Expression というものになっています。一方，上で定義した fib のデータの型は'function' です。

次のように文字式を定義する方法もあります：

文字式としての関数の定義 2

```
1 var('x')
2 f(x) = x^2          # 変数の指定(x)があるのが前と異なる
3
4 print f
5 print f(5)          # これで関数の値が返される
6 print type(f)
```

実行結果の例

```
x |--> x^2
25
<type 'sage.symbolic.expression.Expression'>
```

また，2 変数 x,y の関数を定義するには次のようにします

```
1 var('x y')      #x, yを文字式として取り扱う宣言
2 g = (x+y)^2      # 関数gの定義
3 print g
4 print g(x=3, y=6)      #x=3, y=6のときのgの値
5 print g(y=3)      #y=3のときのgの値
```

実行結果の例

```
(x + y)^2
81
(x + 3)^2
```

36.1 Sage で使える関数

Sage でははじめから様々な初等関数・特殊関数が定義されています。Sage で使える初等関数には三角関数とその逆関数 sin, cos, tan, csc, sec, cot ,arcsin, arccos，対数関数・指数関数 log，ln，exp，双曲関数 sinh，cosh，tanh やその逆関数 arcsinh があります。また代表的な特殊関数であるガンマ関数 gamma(z) ゼータ関数 zeta や楕円関数，いくつかの直交多項式が使えます。これら以外にも数多くの関数が用意されています。

37 グラフの描画とデータの可視化

データの可視化を解説します。Sage では手軽に高機能なグラフ描画機能を利用することが出来ます。以下では出力は Sage ノートブック上で実行していると仮定しています。

37.1 関数のグラフを描画（plot）

関数 $f(x)$ のグラフをプロットするには『plot』を使います。

plot——関数 $f(x)$ のグラフをプロット

```
plot(f(x), (x,a,b))
```

- 関数 $f(x)$ を x の範囲 $[a,b]$ で描画する。

例えば $\sin(x)$ を区間 $[-6,6]$ でプロットするには次のようにします：

例 1 ：
```
plot(sin(x), (x,-6,6))
```

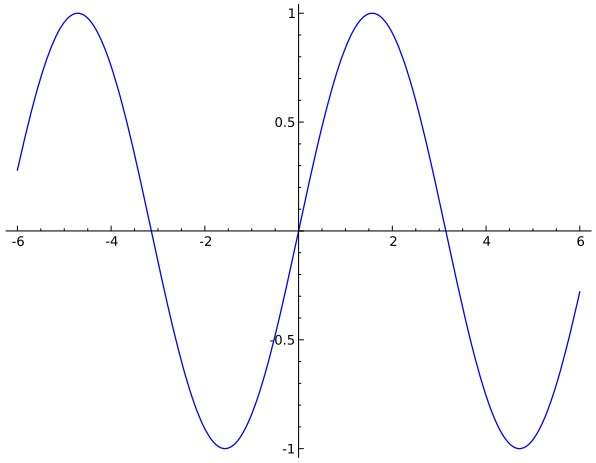

図 9　出力結果：$\sin(x)$ のグラフ

つぎに原点で発散している関数 $1/x$ のグラフを描いてみましょう：

例 **2**：
```
plot(1/x, (x,-2,2))
```

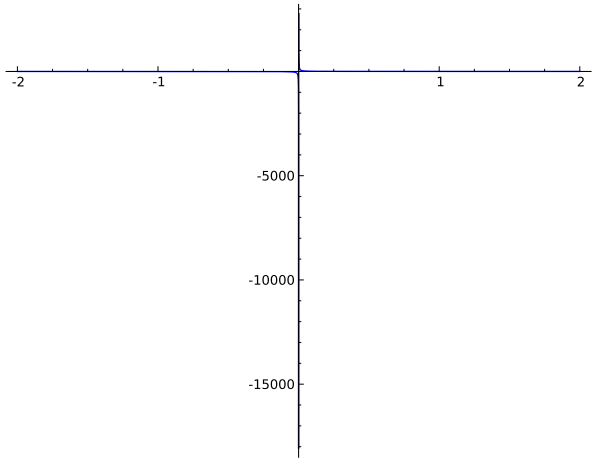

図 10　出力結果：$1/x$ のグラフ

グラフの縦軸が異常に大きい数値になってグラフがつぶれてしまいました。これは $1/x$ のグラフが原点で発散しているためです。このような関数を描画するには関数の値の範囲—**値域**—を指定しないといけません：

関数 $f(x)$ のグラフを値域 $[c,d]$ の範囲で描画

```
plot(f(x), (x,a,b), ymin=c, ymax=d)
```

- 関数 $f(x)$ を x の範囲を $[a,b]$ として描画。
- 描画する関数の y 軸の最小値・最大値をそれぞれ `ymin=c, ymax=d` とする。

関数 $1/x$ のグラフを値域を $[-3,4]$ に制限して描画します。

例３：
```
plot(1/x, (x,-2,2), ymin=-3, ymax=4)
```

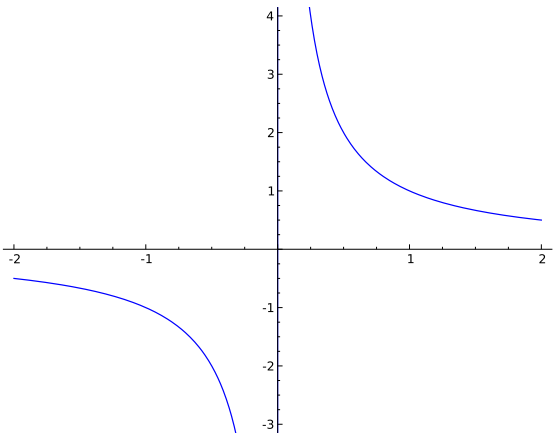

図 11　出力結果：$1/x$ を値域 $[-3,4]$ に制限したグラフ

複数のグラフを一度に表示するには次のようにします：

関数 $f(x), g(x), h(x)$ のグラフを重ねて描画

```
plot( (f(x),g(x),h(x)), (x,a,b) )
```

三つ以上でも同様にカンマで区切って描くことができます。例えば 3 つの関数 $\sin(x), \cos(x), \tan(x)$ を区間 $[-7,7]$，値域 $[-5,5]$ で重ねて描画するには次のようにします：

例４：
```
plot( (sin(x),cos(x),tan(x)), (x,-7,7), ymin=-5,ymax=5)
```

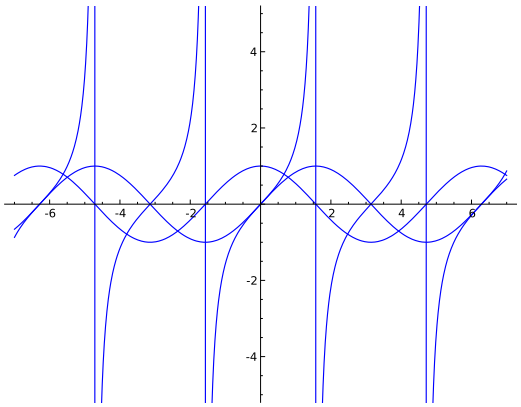

図 12　出力結果：$\sin(x), \cos(x), \tan(x)$ のグラフ

plot——様々なオプション

plot では，オプションを指定することにより色を付けたり，グラフや座標軸に名前を付けたりすることが出来ます。そのときの書式は次の通りです

```
plot( f(x), (x,a,b), color='グラフの色',
      axes_labels=['横軸名','縦軸名'], legend_label='グラフ名')
```

- 『色』は red，yellow，blue，green や RGB カラー'#3F4A46' などを指定。
- axes_labels で軸のラベルを指定。
- 座標軸を消すには axes=False を追加します。
- ラベル名ではドル記号$ ··· $で囲むことで簡単な LaTeX の数式環境が使えます。そこでは\sin(x) で $\sin(x)$，\\tan(x) で $\tan(x)$，\\frac{a}{b}で $\frac{a}{b}$ を表します。

これらのオプションは例えば次のように使用します：

例５：
```
plot(sin(x^2),(x,0,5), color='red', axes_labels=['time','amplitude'],
     legend_label='$\sin(x^2)$')
```

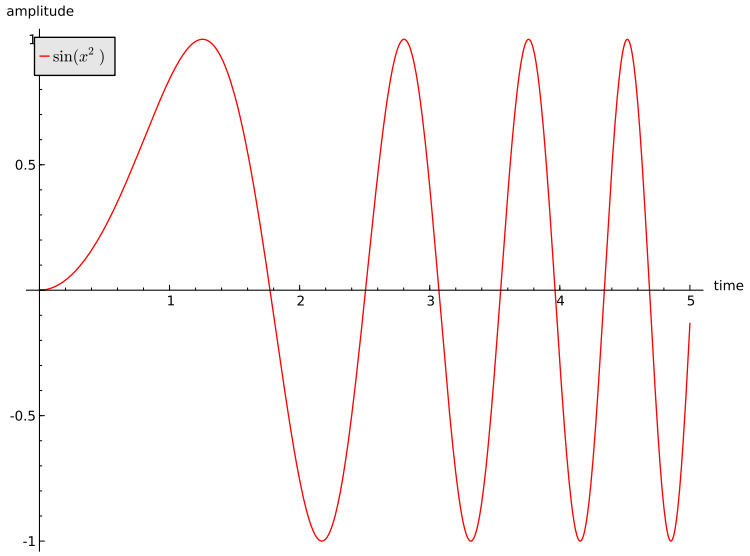

図 13　出力結果：$\sin(x^2)$ のオプション付きプロット

37.2　グラフィックスで使える色

red, blue 意外にも様々な色が用意されています。使える色は colors という辞書に記録されています：

```
for i in sorted(colors):
    print i
```

実行結果の例

```
aliceblue
antiquewhite
aqua
...
...
...
yellow
yellowgreen
```

37.3　グラフ描画：応用編

まず show() の使い方を説明します。グラフなどのオブジェクトに対して，それを画面上に描画する命令が show() です。以下を実行してみましょう：

show の使い方 1

```
1 p1 = plot( sin(x), (x,-6,6) )
2 show(p1)
```

- 1 行目で $\sin(x)$ のグラフデータを変数 p1 に代入しています。
- 2 行目で p1 を画面に表示させています。
- 2 行目は p1.show() としても同じです（試してみましょう）。

次のように show() の中にオプションを書くこともできます：

show の使い方 2

```
1 p1 = plot( tan(x), (x,-6,6) )
2 p1.show(ymin=-5, ymax=5)
```

次の例のように，それぞれのグラフに色を付けて重ねて表示することができます：

例 6：

```
1 p1 = plot(sin(x), (x,-5,5), legend_label='sin', color="red")
2 p2 = plot(cos(x), (x,-5,5), legend_label='cos', color="blue")
3 p3 = plot(tan(x), (x,-5,5), legend_label='$\\tan(x)$', color="green")
4 p4 = p1+p2+p3                #p4はグラフp1,p2,p3を重ねたものです
5 p4.show(ymin=-3,ymax=3)      #p4を表示させるにはshowを使います
```

上の命令では $\sin, \cos, \tan$ のそれぞれのグラフは個別に計算されて変数 p1,p2,p3 の中に入ります。4 行目のように足し合わせる事によって重ねられたグラフ p4 が出来ます。最後に『show』コマンドを使うことにより p4 が描画されます。

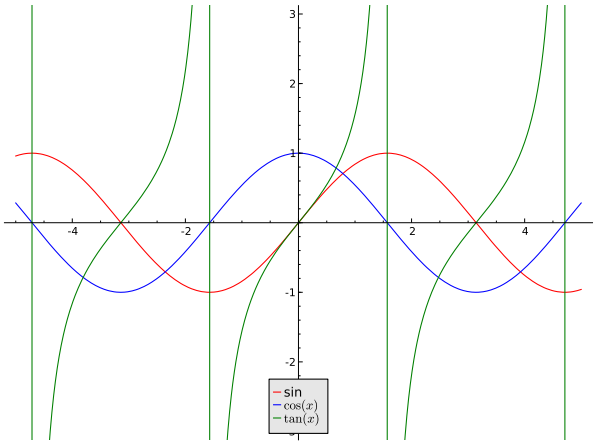

図 14　出力結果：$\sin(x), \cos(x), \tan(x)$ の色を変えて重ねて描画

グラフを重ねるのではなく，横に並べたいときは graphics_array を使います：

graphics_array——二つのグラフを横に並べて表示

例７：
```
1 p1 = plot( sin(x), (x,-4,4) )
2 p2 = plot( cos(x), (x,-4,4) )
3 p3 = graphics_array([p1,p2])
4 p3.show(aspect_ratio=1)
```

ここで aspect ratio とは縦横比のことです。横に並べるときはデフォルトだと横に縮小するため aspect ratio を 1 に指定するとよいでしょう。

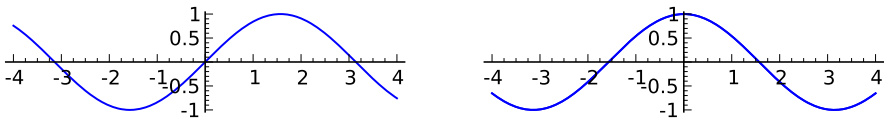

図 15　出力結果：$\sin(x), \cos(x)$ のグラフを横に並べる

`fill` コマンドを使うことによって，グラフの上下やグラフによって囲まれた領域を塗りつぶすことが出来ます：

グラフとフィリング（塗りつぶし）

例８：
```
1 p1 = plot(sin(x), (x,-5,5), fill = 'axis') # 横軸との間を塗りつぶし
2 p2 = plot(sin(x), (x,-5,5), fill = 'min') # グラフの下部を塗りつぶし
3 p3 = plot(sin(x), (x,-5,5), fill = 'max') # グラフの上部を塗りつぶし
4 p4 = plot(sin(x), (x,-5,5), fill = 0.5) #0.5との間を塗りつぶし
5 graphics_array([[p1, p2], [p3, p4]]).show()
```

上のプログラムでは，`graphics_array([[a,b],[c,d]])` によってグラフを $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ と並べています。

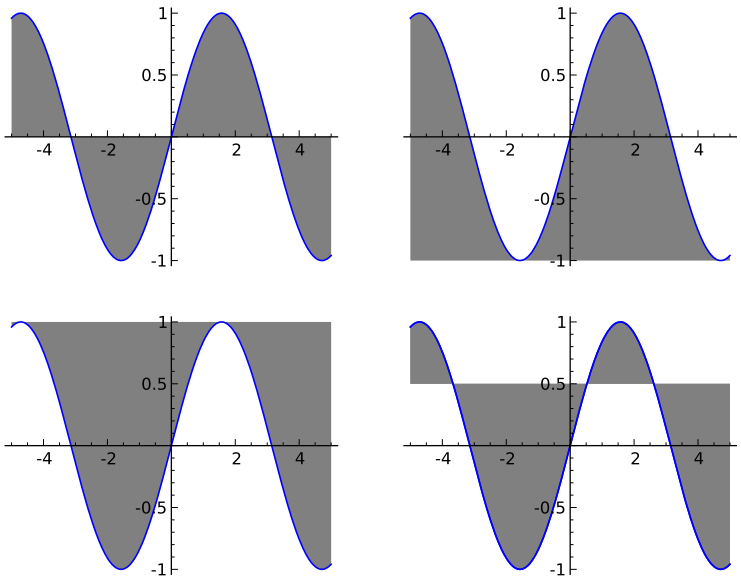

図 16　出力結果：fill の使い方

他の関数との間をフィリングにはつぎのようにします：

$\sin(x)$ のグラフを描き，関数 x^2-1 との間を色づけする

例 9：
```
1 plot( sin(x), (x,-2,2), fill = x^2-1)
```

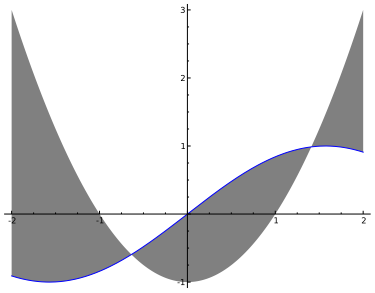

図 17　出力結果：$\sin(x)$ のグラフ，x^2-1 との間を色づけしている

他にもグラフの線の太さ (thickness) を変えたり，線を点線 (dashed) にする等のさまざまなオプションがあります。これらのオプションを参照するには，plot のヘルプをみてください：

plot のヘルプを表示する

```
1 plot?
```

37.4　陰関数のグラフ

$f(x,y)=0$ で定義される x,y 平面の曲線を描くには `implicit_plot` を使います

implicit_plot——$f(x,y)=0$ のグラフ

```
1 var("x y")
2 implicit_plot(f(x,y), (x,a,b), (y,c,d))
```

- 2 変数関数 $f(x,y)$ のグラフを $(x,y)\in[a,b]\times[c,d]$ の範囲で描画する。

ここで `x,y = var('x y')` というのは『x,y を変数として取り扱います』という宣言です。たとえばデカルトの正葉線 (x^3+y^3-3xy) を描くには次のようにします：

$x^3+y^3-3xy=0$ のグラフ

例 **10**：
```
1 var("x y")
2 implicit_plot(x^3+y^3-3*x*y, (x,-2,2), (y,-2,2))
```

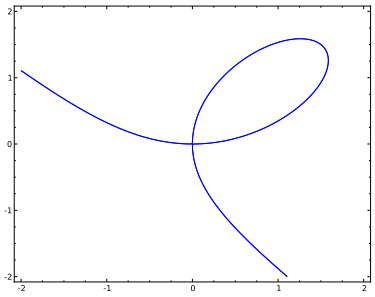

図 18　出力結果：デカルトの正葉線

座標が $(x(t), y(t))$ のように一つのパラメーターに依存して動く点の軌跡を描画するには `parametric_plot` を用います：

$(x(t), y(t))$ で定義される軌跡の描画

例 **11**：
```
1 t = var('t')
2 parametric_plot([cos(t) + 2*cos(t/4), sin(t)-2*sin(t/4)],
3       (t,0, 8*pi), fill=true)
```

上の例では $x(t) = \cos(t) + 2\cos(t/4), y(t) = \sin(t) - 2\sin(t/4)$ です。パラメーター t の動く範囲は 8π にしています。オプション `fill` を使って，囲まれる領域を色づけしています：

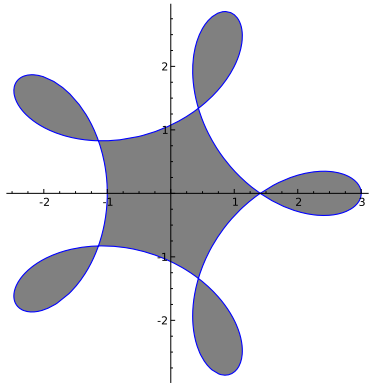

図 19　出力結果：parametric plot(with filling)

37.5　リストのプロット

リスト化されたデータをプロットするには `list_plot` を使います：

リストのプロット

```
1 a = リスト
2 list_plot(a)
```

- プロットできるリストは数値のデータのリスト `[3,2,6,3,12,-2,2]` のようなものや，2 次元ベクトルからなる `[(1,2),(4,1),(3,4),(4,2)]` のようなリストです。または 3 次元のデータのからなるリストに対しては 3 次元的な描画ができます。

例 12：
```
1 a = [1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10]
2 list_plot(a)
```

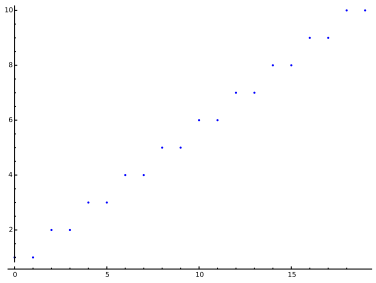

図 20　出力結果：リストのプロット

リストの最初は 0 番目，次は 1 番目となることに注意してください。上と同じリストでプロットのオプションに `plotjoined=True` を指定すると点同士が直線でつながります。

例 13：
```
1 a = [1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10]
2 list_plot(a, plotjoined=True)
```

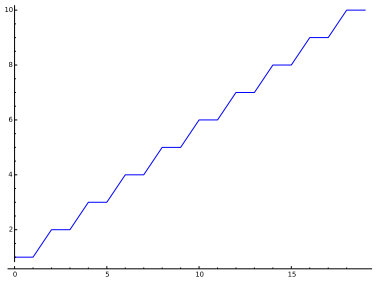

図 21　出力結果：リストのプロット (plotjoined=True)

例えば 2 次元の点データのプロットは次のようにします：

例 14：
```
1 a = [(sqrt(i)*cos(i),sqrt(i)*sin(i)) for i in range(20)];
2 list_plot(a,plotjoined=True)
```

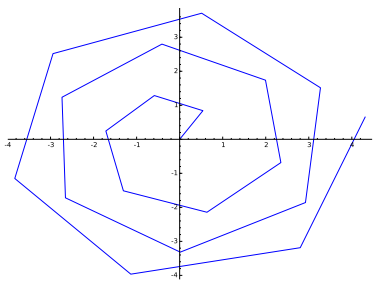

図 22　出力結果：$\{\sqrt{i}(\cos(i), \sin(i))|i = 0, 1, \cdots, 19\}$ をプロット

37.6　変数 (x, y) の取り得る 2 次元領域を描く

関数 $f(x, y)$ が与えられたときに $f(x, y) > 0$ となる領域を描くには `regin_plot` を用います：

region_plot——$f(x, y) > 0$ となる (x, y) の領域を描く

```
x,y = var('x,y')
region_plot(f(x,y)>0, (x,a,b), (y,c,d))
```

たとえば $\sin(x^2 - y^3) > 0$ となる x, y の領域を描くには

例 **14**：
```
x,y = var('x,y')
region_plot(sin(x^2-y^3)>0, (x,-3, 3), (y,-3, 3))
```

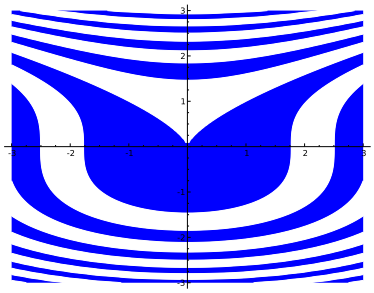

図 23　出力結果：$\sin(x^2 - y^3) > 0$

変数 (x, y) に対する条件は `[f(x,y)>0,g(x,y)>0,h(x,y)>0]` のようにリストにすることにより複数指定することが出来ます。

例 **15**：
```
x,y = var('x,y')
region_plot([x>0, y>0, x^2+y^2>0.5], (x,-1,1), (y,-1,1) )
```

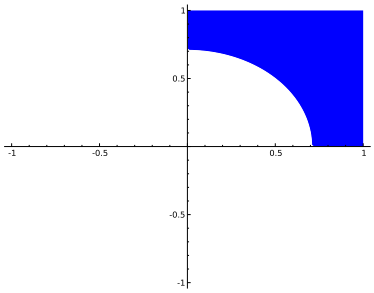

図 24　出力結果：$x > 0, y > 0, x^2 + y^2 > 0.5$ となる x, y の領域のグラフ

region plot は指定された定義域を 100 等分して条件をチェックして描画を行っています。変化の激しい関数の場合は，条件が十分正しくチェックされずに大幅に間違ったグラフが出力される可能性があります。このときは `plot_points=300` のようにオプションを指定することにより，プロットする点の数を増やしまします。`region_plot` についての詳しいことはヘルプを参照してください：

```
region_plot?
```

37.7　基本的なパーツ (円・楕円・矢印・円弧・線分・点・テキスト)

Sage の 2 次元グラフィックスでは，円や線分などの基本的なパーツを手軽に描くことができます。これらのパーツは plot などの命令と簡単に組み合わせることができる事です。

次のようなものが使えます：

- `circle((a,b),r)`：中心 (a,b), 半径 r の円周
- `ellipse((a,b),c,d)`：中心 (a,b)，横の半径 c, 縦の半径 d の楕円
- `arrow((a,b),(c,d))`：始点 (a,b), 終点 (c,d) の矢印
- `arc((a,b),r,sector=(c,d))`：中心 (a,b), 半径 r, 角度 (c,d)。角度 (c,d) は，たとえば 0 度から 90 度までの円弧なら $(0,\pi/2)$ のようにラジアンで表します。
- `line([(a,b),(c,d)])`：(a,b) と (c,d) を結ぶ線分。オプションとして `linestype='--'` を指定すると点線となります。また `marker='o'` のオプションを付けると線分の両端に点がつきます。
- `point(a,b,size=r)`：位置 (a,b)，大きさ r の点
- `text('文字', (a,b), fontsize=r)`：位置 (a,b) にあるサイズ r の文字

例 16：

```
s1 = plot(x^2,(x,-1,1))
s2 = point((0,0), size=100, color='black')
s3 = arrow((-1/2,1/4),(1,1), color='red')
s4 = arc((0,0),0.5,sector=(0,pi), color='green')
s5 = s1 + s2 + s3 + s4
s5.show()
```

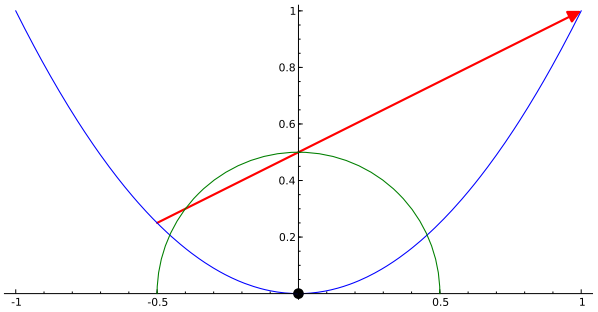

図 25　出力結果：グラフと点と矢印と円弧

38　2 変数関数の可視化

ここでは実数値の 2 変数関数 $f(x,y)$ の可視化を解説します。

38.1　3 次元のプロット

2 変数の実数値関数 $f(x,y)$ を 3 次元的にプロットするには `plot3d` を使います。

plot3d——$f(x,y)$ の 3 次元的な描画

```
var('x y')
plot3d(f(x,y), (x,a,b),(y,c,d))
```

次のようなオプションがあります。

- plot_points：サンプルする点の数 (多くすればするほど精密なグラフになる)
- opacity：透明度 (0 から 1 の値，0.5 なら半透明になる)
- aspect_ratio：縦横高さの比（[1,2,1] のように比を指定する）

例 **17**：
```
var('x y')
plot3d(sin(x-y)*y*cos(x), (x,-3,3), (y,-3,3),
     opacity=0.5, aspect_ratio=[1,1,1])
```

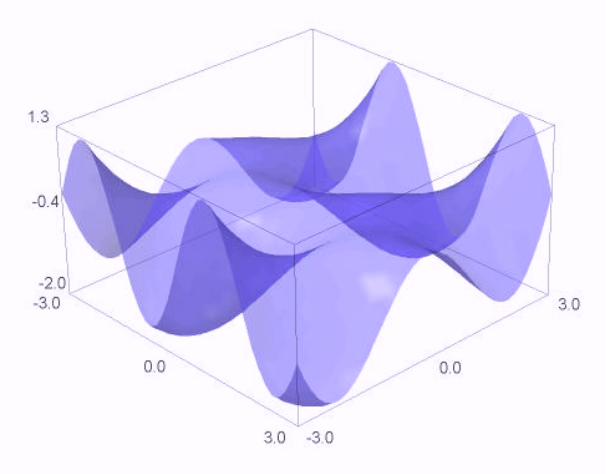

図 26　出力結果：$y\sin(x-y)\cos(x)$ のグラフ

38.2　密度プロット（density plot）と等高線プロット（contour plot）

plot3d の他にも密度プロットは関数の高さを色の濃さであらわす density plot や，等高線であらわす contour plot があります。密度プロットでは高い方が明るく，低い方が濃くなります。

例 **18**：
```
x, y = var('x y')
density_plot(sin(x*y), (x,-3,3), (y,-3,3))
```

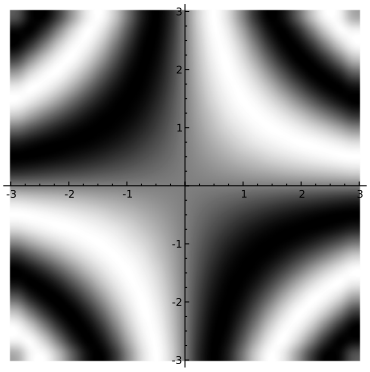

図 27　出力結果：$\sin(xy)$ の密度プロット

例 **19**：
```
x, y = var('x y')
contour_plot(cos(x)+cos(y), (x, -8, 8), (y, -8, 8),contours=20)
```

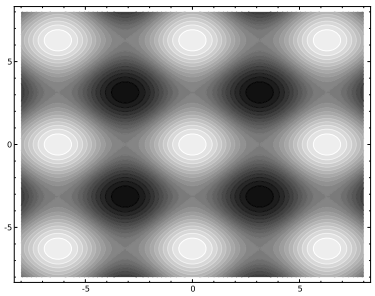

図 28　出力結果：$\cos(x)+\cos(y)$ の等高線プロット

38.3　その他の 3 次元プロット

上記以外の 3 次元描画には次のようなものがあります。

- `implicit_plot3d`
- `parametric_plot3d`
- `plot_vector_field3d`

38.4　画像の保存

描いた画像を保存する方法はいくつかありますが Sage notebook を使っていて png 形式で保存するなら出力された画像を右クリックして『名前を付けて保存』から保存するだけです。png 形式以外では pdf,eps,ps の形式で保存することができます：

notebook での画像の保存の仕方

```
p1 = plot(sin(x), (x,-4,4))
p1.save("ファイル名.pdf")
```

- 上を実行すると画像が表示される代わりにリンクが表示されます。リンクをクリックすると画像ファイルをダウンロードできます。
- eps で保存するなら上で pdf の部分を eps に変えます。
- 3 次元の絵は eps, pdf, ps などの形式では保存することができませんが，png 形式で保存することができます。

Sage のプログラム (**.sage 形式のファイル) を実行することにより，直接ファイルを保存することもできます。これは大量の画像を生成するときに特に便利です。

- ファイル名：**plot.sage**

```
p1 = plot(sin(x), (x,-4,4))
p1.save("plot.pdf")      # p1をpdfファイルとしてセーブする
```

上のファイルを Emacs などで作成したら，端末から

```
user@debian:~$ sage plot.sage    # pdfファイルが生成される（場所は実行したディレクト
user@debian:~$
```

と実行することにより，ディレクトリに直接画像のファイルが生成されます。

39　グラフの描画に関するおもしろい例題

複素関数 e^z は零点を持ちませんが，マクローリン展開を有限項で切った多項式

$$\sum_{k=0}^{n} \frac{z^k}{k!}$$

は n 個の零点を持ちます。不思議なことに関数を $z \to nz$ とスケーリングすると

$$\sum_{k=0}^{n} \frac{(nz)^n}{k!} \tag{7}$$

の零点の集合は $n \to \infty$ の極限で関数 $|ze^{z-1}| = 1$ の滴の部分に一致します（この事はすでに証明されていますが，どうやって証明するんでしょう？）。数値計算でこれを試してみたいと思います。

多項式の零点を求めるには roots を使います。例えば $x^5 + 3x + 1 == 0$ の零点を求めるには

```
eq = x^5+3*x+1==0
eq.roots(x, ring=CC, multiplicities=False)
```

とします。実行結果は

実行結果の例

```
[-0.331989029584509, -0.839072433066608 - 0.943851550132862*I,
-0.839072433066608 + 0.943851550132862*I, 1.00506694785886 -
0.937259156692892*I, 1.00506694785886 + 0.937259156692892*I]
```

となります。multiplicities は多項式の根の多重度を表示させるかどうかのオプションです。いまは必要ないので False にしています。

さて，7 の零点を求めてみましょう：

```
var('z k')    #z, kを変数とする
```

```
nn = 30          #nnの値を30とする
f = sum((nn*z)^k/factorial(k), k,0,nn) # 関数fを定義
eq = f==0                              # 方程式f==0をeqと名付ける
lis = eq.roots(z, ring=CC, multiplicities=False);    # eqの根の集合をlisとする
list_plot(lis)      #lisをプロットする
```

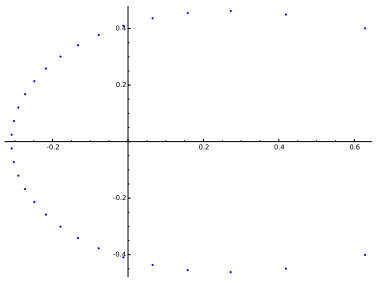

図 29　出力結果：$n = 30$ のときの零点の集合

点の数を増やしてみましょう。

```
var('z k')    #z, kを変数とする
nn = 200         #nnの値を200とする
f = sum((nn*z)^k/factorial(k), k,0,nn) # 関数fを定義
eq = f==0                              # 方程式f==0をeqと名付ける
lis = eq.roots(z, ring=CC, multiplicities=False);    # eqの根の集合をlisとする
list_plot(lis)      #lisをプロットする
```

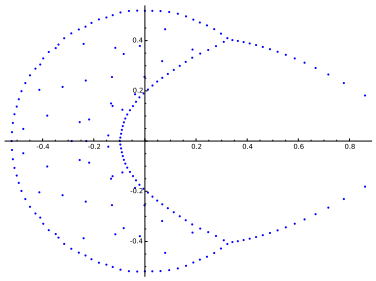

図 30　出力結果：$n = 100$ のときの零点の集合?（誤差で変な結果になる）

この結果は信用できないので，roots のオプションで計算精度を高くしてみたいと思います。そのためには CC で計算していたところを ComplexField(300) のように精度を上げて計算します。

```
var('z k')
nn = 200         # nnの値を200とする
f = sum((nn*z)^k/factorial(k), k,0,nn)
eq = f==0
lis = eq.roots(z, ring=ComplexField(300), multiplicities=False) # 300bitで計算
list_plot(lis)
```

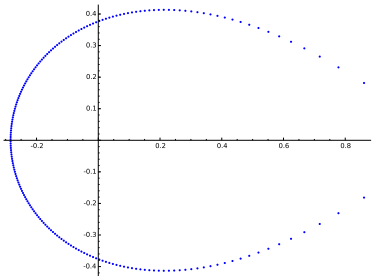

図 31　出力結果：$n = 200$ のときの零点の集合

次に $|ze^{1-z}| = 1$ を平面上に描いてみましょう。$z = x + iy$ のとき $|ze^{1-z}| = \sqrt{x^2+y^2}e^{1-x}$ なので

```
var('x y')
implicit_plot(sqrt(x^2+y^2)*exp(1-x)-1, (x,-1,2), (y,-1,1),color='red')
```

とすればプロットすることができます。これを上で求めた零点と重ねてみましょう。

```
var('z k')
nn =200
f = sum((nn*z)^k/factorial(k), k,0,nn)    # 関数fを定義
eq = f==0
lis = eq.roots(z, ring=ComplexField(300), multiplicities=False);
p1 = list_plot(lis)
var('x y')
p2 = implicit_plot(sqrt(x^2+y^2)*exp(1-x)-1, (x,-1,2), (y,-1,1),color='red')
(p1+p2).show()
```

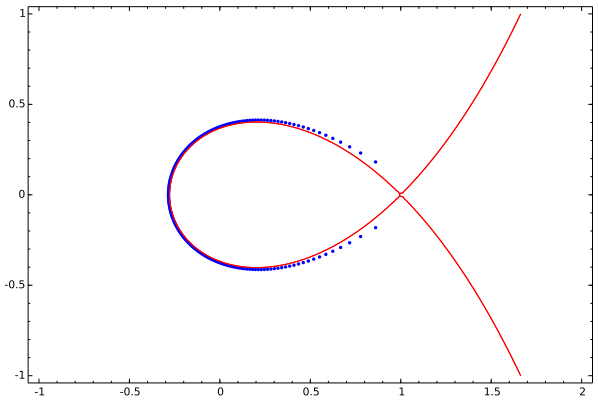

図 32　出力結果：零点の集合と $|ze^{1-z}| = 1$ のグラフを重ねる

青い点（ゼロ点）と赤い線の滴の部分はよく一致していることが見て取れます。

複素関数を色を付けて表示する complex_plot というコマンドがあります，これを使って $\sum_{k=1}^{40}(40z)^k/k!$ を描くと次のようになります。

```
var('k')
nn=40
f = sum((nn*x)^k/factorial(k),k,0,nn)
complex_plot(f,(-1,2),(-1,1),plot_points=500,aspect_ratio=1)
```

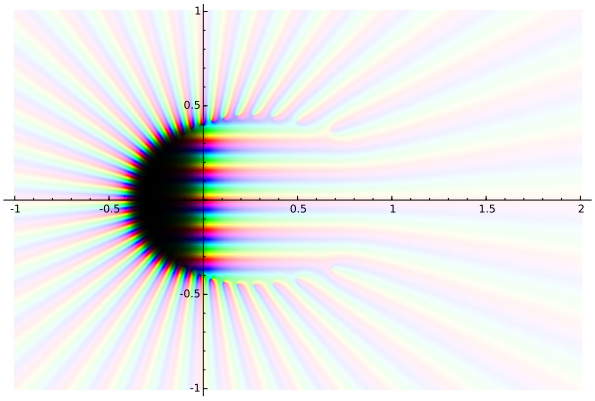

図 33　出力結果：複素関数 $\sum_{k=1}^{40}(40z)^k/k!$ の形

40　練習問題

上の節 37, 38, 39 の解説にあるグラフをすべて描画してみよう。グラフが描画された SageMath のワークシートを提出すること。ファイル名は `sagews02.sws` とすること。