46 微分方程式

46.1 常微分方程式とベクトル場

微分方程式

$$\frac{dy}{dx} = y - x \tag{10}$$

を考える。これの一般解は，$y(x) = x + 1 + Ce^x$ である。ここに C は任意定数。初期値問題

$$\frac{dy}{dx} = y - x, \qquad y(0) = \frac{2}{3} \tag{11}$$

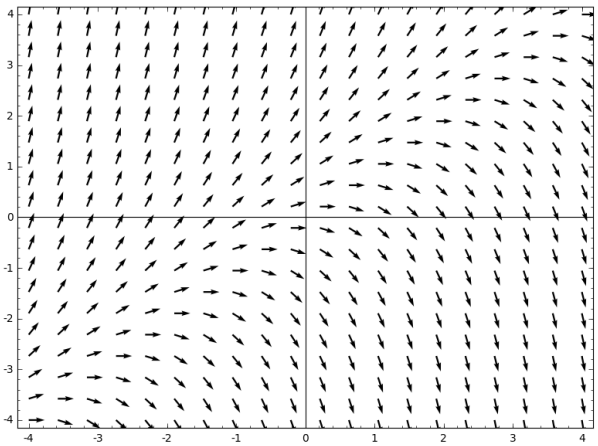
を考える。上の一般解に $x = 0$ の値を代入して $y(0) = 1 + C = 2/3$ から C を求めると，$C = -1/3$ となることがわかる。よってこの初期値問題の解は

$$y(x) = x + 1 - \frac{1}{3}e^x \tag{12}$$

である。このようにして代数的な計算によって微分方程式の解を求めることができるが，コンピューターによる描画を用いると解の様子を視覚的に観察することができる。

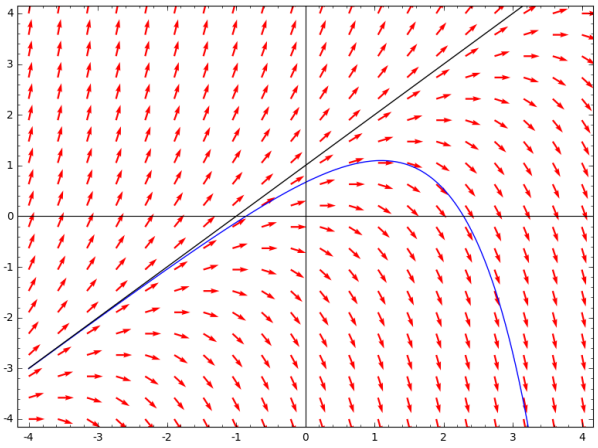
微分方程式 (10) の解曲線 $y = y(x)$ は平面上の点 (x, y) で傾き $y - x$ を持つ。そこで，点 (x, y) に $(1, y - x)$ の向きの単位ベクトルが生えているベクトル場を描いてみよう。そのようなベクトル場は勾配場 (slope field) と呼ばれる。つまり，$m = (1 + (y - x)^2)^{-1/2}$ として，ベクトル場 $V(x, y) = (m, m(y - x))$ を描けばよい。

```
var('y')
m = sqrt(1+(y-x)^2)^(-1)
plot_vector_field( (m,m*(y-x)), (x,-3,3), (y,-3,3) )
```



微分方程式 (10) の解は，初期条件 $(0, y(0))$ から出発して，上の絵の矢印をその向きにたどることで得られる。上の絵を見れば，解の振る舞いは一目瞭然であろう。具体的には直線 $y = x + 1$ を境に，その上と下で $x \to \infty$ のときの振る舞いは異なる。この勾配場 (赤) に $y = x + 1$ の直線 (黒) および解 (12)(青) を重ねてみよう。

```
var('y')
m = sqrt(1+(y-x)^2)^(-1)
vecf = plot_vector_field( (m,m*(y-x)), (x,-4,4), (y,-4,4) ,color="red")
sol = plot(x+1-(1/3)*e^x, (x,-4,4))
asym = plot(x+1, (x,-4,4), color='black')
(vecf+sol+asym).show(ymin=-4, ymax=4)
```



46.2 Euler 法

微分方程式の多くは代数的に解くことはできない。しかし，その初期値問題の解を数値的に計算することは可能である。ここでは，微分方程式の数値計算法の一つである Euler 法について解説する。$f(x,y)$ を与えられた関数として，微分方程式

$$\frac{dy}{dx} = f(x,y) \tag{13}$$

及び初期条件 $y(x_0) = y_0$ を考える。(x_0, y_0) は与える与えられた数値である。上の微分方程式は無限小 dx を用いて

$$y(x+dx) = y(x) + f(x,y)dx \tag{14}$$

と書くことができる。これは y の点 x での値から無限小だけ進んだ点 $x+dx$ での値を決める方程式と考えられる。Euler 法はこれに基づいて $y(x)$ を近似する方法である。Δx を小さい数とし，$x_1 = x_0 + \Delta x$ での y の値は

$$y_1 = y_0 + f(x_0, y_0)\Delta x \tag{15}$$

で近似される。$x_2 = x_1 + \Delta x$ での y の値は

$$y_2 = y_1 + f(x_1, y_1)\Delta x \tag{16}$$

で近似される。同様にして $x_n = x_0 + n\Delta x$ での y の値は

$$y_n = y_{n-1} + f(x_{n-1}, y_{n-1})\Delta x \tag{17}$$

によって近似される。

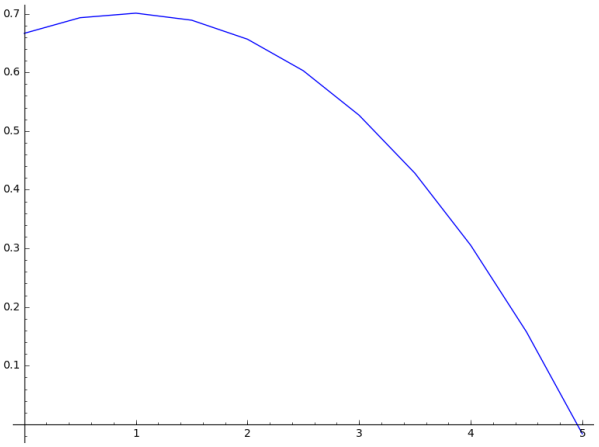
Euler 法に従って微分方程式の初期値問題 (11) の解 $y(x)$ を近似してみよう。以下では，Δx と n の値を適当に指定して，リスト $((x_0,y_0),(x_1,y_1),\cdots,(x_n,y_n))$ を作る。

```
x0, y0 = 0, 2/3    # 初期条件をセット
Dx= 0.5            # Delta xを設定
var('y')
f(x,y) = y-x       # f(x,y)を定義
SOL = [(x0,y0)]            # 初期条件だけからなるリスト
for j in range(1,11):
```

```
    X = x0+Dx                # 次のxの値
    Y = y0 + f(x0,y0)*dx     # 次のyの値
    SOL.append((X,Y))        # (X,Y)をリストに加える
    x0,y0 = X, Y             # x0, y0をX,Yに置き換える
print SOL      # 解曲線を近似したリストをプリント
approx = list_plot(SOL,plotjoined=True)    # SOLをプロット
approx.show()
```

実行結果の例

```
[(0, 2/3), (0.500000000000000, 0.693333333333333), (1.00000000000000,
0.701066666666667), (1.50000000000000, 0.689109333333333),
(2.00000000000000, 0.656673706666667), (2.50000000000000,
0.602940654933333), (3.00000000000000, 0.527058281130667),
(3.50000000000000, 0.428140612375893), (4.00000000000000,
0.305266236870929), (4.50000000000000, 0.157476886345766),
(5.00000000000000, -0.0162240382004033)]
```

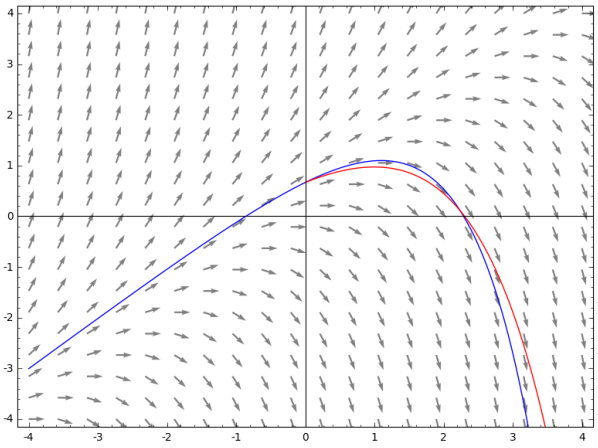


Euler 法による近似解と真の解とを比較してみよう。

```
var('y')
m = sqrt(1+(y-x)^2)^(-1)
vecf = plot_vector_field( (m,m*(y-x)), (x,-4,4), (y,-4,4) ,color="gray")
sol = plot(x+1-(1/3)*e^x, (x,-4,4))    # 真の解のグラフ

x0, y0 = 0, 2/3    # 初期条件をセット
Dx= 0.05          # Delta xを設定
f(x,y) = y-x       # f(x,y)を定義
SOL = [(x0,y0)]            # 初期条件だけからなるリスト
for j in range(1,81):
    X = x0+Dx              # 次のxの値
    Y = y0 + f(x0,y0)*Dx   # 次のyの値
    SOL.append((X,Y))      # (X,Y)をリストに加える
    x0,y0 = X, Y           # x0, y0をX,Yに置き換える
approx = list_plot(SOL,plotjoined=True,color='red')    # SOLをプロット

(vecf+sol+approx).show(ymin=-4, ymax=4) # 赤がEuler法によって計算したもの
```



Euler 法によって計算した値は，厳密な値のグラフと比べると誤差があるのがわかる。この誤差は Δx を小さく取ることにより，小さくすることができる。例えば，上のプログラムで `Dx=0.001` としてみよう。（このとき 10 行目の `range(1,81)` を変えて，計算のステップ数も増やす必要がある。）

46.3 微分方程式の厳密解

この節では `desolve` 関数を使って微分方程式を解きます。`desolve` の文法とオプションは次の通りです：

微分方程式を解く (desolve)

```
1 desolve(de, dvar, options)
```

ここで

- `de`: 微分方程式 (differential equation)
- `dvar`: 従属変数 (dependent variable:求める関数)

です。オプション `options` は次が用意されています：

- `ics`: 初期条件 (initial conditions) もしくは境界条件を指定するときに設定します
- `ivar`: 独立変数 (independent variable)
- `show_method`: 解法に使用した手法 (微分方程式の型) を表示する
- `contrib_ode`: Clairaut, Lagrange, Riccati 型の方程式も含めて解を探す場合に `True` にする。

これらのオプションは省略可能で，指定しない場合 `False` になります。

さて簡単な微分方程式

$$y'(x) = y(x) \tag{18}$$

を解いてみましょう。これ解くには次のようにします：

```
1 x = var('x')                  # x を変数とする（省略可）
2 y = function('y')(x)          # y を x の関数とする
3 desolve( diff(y,x) == y, y)   # 微分方程式 y'=y を y について解く
```

実行結果の例

```
_C*e^x
```

上のプログラムの 2 行目で，y は x の関数であるという宣言をしています。3 行目の diff(y,x) は微分 $y'(x)$ を意味していて，diff(y,x)==y が解きたい微分方程式ということになります。実行結果の_C は任意定数です。

desolve では微分方程式を指定するときに==0 を省略することができます。例えば，desolve(diff(y,x)-y, y) と desolve(diff(y,x)-y == 0, y) は同じ意味です。

```
y = function('y')(x)
desolve( diff(y,x) - y , y)   # 微分方程式 y'-y =0 を yについて解く
```

実行結果の例

```
_C*e^x
```

46.3.1 微分方程式の解法 1

次の微分方程式を解く事を考えましょう。

$$y'(x) + 2y(x)\sin(x) = 0 \tag{19}$$

微分方程式を勉強した事がある人は，変数分離型の解法によって直ちにこれを解く事ができると思います。忘れた人もいるかもしれないので念のために変数分離型の解法を復習しておきます。まず上の微分方程式を

$$\frac{y'}{y} = -2\sin(x) \tag{20}$$

と変形して[*13]，両辺を x で積分する事により

$$\log y = \int \frac{y'}{y} dx = 2\cos(x) + c \tag{21}$$

となります。これを y について解くことにより答えは

$$y(x) = \exp(2\cos(x) + c) \tag{22}$$

となります。これも一般解といえますが，これでは y が恒等的に 0 となる自明な解を含みません。そこで，$C = e^c$ を新たに任意定数とすると，(19) の解は

$$y(x) = Ce^{2\cos(x)} \tag{23}$$

となります。さて，上の微分方程式を Sage で解いてみましょう：

```
y = function('y')(x)
DE = diff(y,x) + 2*y*sin(x) == 0 # 微分方程式を DEと名付ける
desolve(DE, y)                   # DEをyについて解く
```

実行結果の例

```
_C*e^(2*cos(x))
```

46.3.2 微分方程式の解法 2

次の微分方程式を考えます：

$$y'(x) = \frac{x - y(x)}{x + y(x)}. \tag{24}$$

これも変数分離型として解く事ができますが，Sage に解かせてみましょう：

[*13] $y(x) = 0$ のときはどうするんだと指摘されそうですが，とりあえず $y(x) \neq 0$ と仮定しましょう。

```
y = function('y')(x)
desolve( diff(y,x) == (x-y)/(x+y), y)
```

実行結果の例

```
-1/2*x^2 + x*y(x) + 1/2*y(x)^2 == _C
```

微分方程式 (24) の解 $y(x)$ は，微分を含まない方程式

$$-\frac{1}{2}x^2 + xy(x) + \frac{1}{2}y(x)^2 = C \tag{25}$$

を満たす関数として陰 (implicitly) に解かれました。上式は $y(x)$ についての 2 次関数なので解の公式を使って直ちに解く事ができますが，Sage の solve にそれをやらせてみましょう：

```
y = function('y')(x)
SOL = desolve( diff(y,x) == (x-y)/(x+y), y)   # 微分方程式の答えをSOLと名づける
solve(SOL,y)                                  # 方程式SOLをyについて解く
```

実行結果の例

```
[y(x) == -x - sqrt(2*x^2 + 2*_C), y(x) == -x + sqrt(2*x^2 + 2*_C)]
```

したがって，微分方程式の答えは，$y(x) = -x \pm \sqrt{2x^2 + 2C}$ となることが分かります。

± と定数 C に応じて，微分方程式の解は無数にありますが，これをグラフに描いてみましょう：

```
p = Graphics()     # 空のグラフィックスオブジェクトを作る
for C in range(4):          # 定数Cを0から3まで変えながら解のグラフをpに加える
  p += plot( -x - sqrt(2*x^2 + 2*C), (x,-3,3),
       color=hue(C/4,1 ), legend_label='C='+str(C)+',$+$') #Cごとに色を変える
  p += plot( -x + sqrt(2*x^2 + 2*C), (x,-3,3),
       color=hue(C/4,0.5), legend_label='C='+str(C)+',$-$')#Cごとに色を変える

p.show()      # pを描画
```

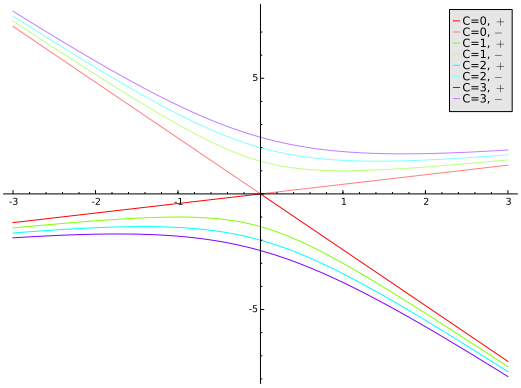


図 34　出力結果：微分方程式 (24) の解のグラフ

46.3.3　微分方程式の解法 3

次の微分方程式を考えます：

$$y' = \sin(x + y) + \sin(x - y) \tag{26}$$

これも変数分離型で解く事ができますが，このままでは Sage は解く事ができません，

```
y = function('y')(x)
desolve( diff(y,x) == sin(x+y)+sin(x-y), y)
```

実行結果の例

```
NotImplementedError: Maxima was unable to solve this ODE. Consider to
set option contrib_ode to True.
```

右辺を加法定理で因数分解してやれば解く事ができます：

```
y = function('y')(x)
DE = diff(y,x) == sin(x)*cos(y)+sin(y)*cos(x)+sin(x)*cos(y)-sin(y)*cos(x)
desolve(DE, y)
```

実行結果の例

```
-1/4*log(sin(y(x)) - 1) + 1/4*log(sin(y(x)) + 1) == c - cos(x)
```

46.4　二階の微分方程式

$y(x)$ の二階までの微分を含む方程式を二階の微分方程式といいます。x の関数 y の二階微分は diff(y,x,2) です。通常，二階の微分方程式の解は 2 つの任意定数を持ちます。それでは，二階の微分方程式

$$y''(x) - 8y'(x) + 15y(x) = 0 \tag{27}$$

を解いてみましょう。

```
y = function('y')(x)
desolve( diff(y,x,2)-8*diff(y,x)+15*y == 0, y)
```

実行結果の例

```
_K1*e^(5*x) + _K2*e^(3*x)
```

ここで_K1，_K2 は二つの任意定数です。

46.5　オプション 1 （Riccati, Clairaut, Lagrange を含めた解法）

desolve は，さまざまなタイプの微分方程式を解くことができますが，Riccati 型微分方程式

$$\frac{dy}{dx} + a(x)y^2 + b(x)y + c(x) = 0$$

や Clairaut 型の微分方程式

$$y = x\frac{dy}{dx} + f\left(\frac{dy}{dx}\right)$$

や力学の講義で学ぶ Lagrange 方程式の可能性も含めて解かせるにはオプションで contrib_ode=True と指定する必要があります。このとき，どの手法を用いたかを知るためには，オプションとして show_method=True を指定します。

では微分方程式 $y'^2 + xy' - y = 0$ を解いてみましょう：

```
y = function('y')(x)
DE = diff(y,x)^2 + x*diff(y,x)-y == 0      # 微分方程式DEを定義
desolve(DE, y, contrib_ode=True, show_method=True)
```

実行結果の例

```
[[y(x) == _C^2 + _C*x, y(x) == -1/4*x^2], 'clairault']
```

上の微分方程式を解くときに Clairaut 型の微分方程式の解法が使われたことがわかります。

46.6 オプション２：方程式が定数を含む場合

$y'(x) = ay(x)$ のように微分方程式が定数 a を含む場合は，その変数を `var('a')` で指定するのと同時に，オプションとして独立変数を `ivar=x` と指定します。

```
var('a')      # aを変数とする
y = function('y')(x)
desolve(diff(y,x) + a*y == 0, y, ivar=x)
```

実行結果の例

```
_C*e^(-a*x)
```

微分方程式 $y'' = ay$ は a が正か負か 0 かによって解が劇的に変化します。a の符号がわからないと微分方程式が解けない場合があります：

```
var('a')
y = function('y')(x)
desolve(diff(y,x,2) + a*y == 0, y, ivar=x)
```

実行結果の例

```
Traceback (click to the left of this block for traceback)
...
Is a positive, negative or zero?
```

このようなメッセージが出た場合，a に対する条件を指定すると解けるかもしれません。例えば $a > 0$ と仮定すると次のようになります：

```
var('a')
y = function('y')(x)
assume(a>0)           # a>0と仮定
desolve(diff(y,x,2) + a*y == 0, y, ivar=x)
```

実行結果の例

```
_K2*cos(sqrt(a)*x) + _K1*sin(sqrt(a)*x)
```

46.7 オプション３：初期条件と境界条件 (ics)

$y' = y$ の一般解は $y = Ce^x$ です。初期条件が $y(0) = 1$ なら $C = 1$ です。一階の微分方程式を初期条件

$$y(x_0) = y_0 \tag{28}$$

の下で解くときには，オプション (`ics`) に $[x_0, y(x_0)]$ を指示します。例えば微分方程式 $y' = y$ を初期条件 $y(0) = 1$ で解くなら次のようにします。

```
y = function('y')(x)
desolve( diff(y,x) + y == 0, y, ics=[0,1] )  # 初期条件icsを指定
```

実行結果の例

```
e^x
```

二階の微分方程式の初期条件は `ics`$= [x_0, y(x_0), y'(x_0)]$ のように指定します。

例えば微分方程式 $y'' + 2y' + y = 0$ で，初期条件を $y(0) = 3, y'(0) = 1$ とするなら

```
y = function('y')(x)
desolve( diff(y,x,2) + 2*diff(y,x) + y , y, ics=[0,3,1] ) # 初期条件を指定
```

実行結果の例

```
(4*x + 3)*e^(-x)
```

とします。また境界条件[*14]を設定するには ics= $[x_0, y(x_0), x_1, y(x_1)]$ をオプションとして書きます。上と同じ微分方程式を境界条件 $y(0) = 3, y(\pi/2) = 2$ として解くと

```
y = function('y')(x)
desolve( diff(y,x,2)+2*diff(y,x)+y , y, ics=[0,3,pi/2,2] ) # 境界条件を指定
```

実行結果の例

```
(2*(2*e^(1/2*pi) - 3)*x/pi + 3)*e^(-x)
```

となります。

46.8　その他の方法

微分方程式 $xy' - x\sqrt{y^2 + x^2} - y = 0$ はそのままでは解けない：

```
y = function('y')(x)
desolve( x * diff(y,x) - x * sqrt(y^2+x^2) - y == 0, y, contrib_ode=True )
```

実行結果の例

```
Traceback (click to the left of this block for traceback)
...
TypeError: ECL says: Maxima asks: Is  y  zero or nonzero?
```

しかし，x と y の範囲に条件を付ければ解ける：

```
y = function('y')(x)
assume(x>0);  assume(y>0);          #x>0, y>0 と仮定する
desolve( x * diff(y,x) - x * sqrt(y^2+x^2) - y == 0, y, contrib_ode=True )
```

実行結果の例

```
x - arcsinh(y(x)/x) == c
```

上の結果は Sage4.6 の結果ですが，Sage6.7 では次のようになりました。

実行結果の例

```
[1/2*(2*x^2*sqrt(x^(-2)) - 2*x*sqrt(x^(-2))*arcsinh(y(x)/sqrt(x^2)) -
2*x*sqrt(x^(-2))*arcsinh(y(x)^2/(sqrt(y(x)^2)*x)) +
log(4*(2*x^2*sqrt((x^2*y(x)^2 + y(x)^4)/x^2)*sqrt(x^(-2)) + x^2 +
2*y(x)^2)/x^2))/(x*sqrt(x^(-2))) == _C]
```

この左辺に simlify_full() を施せばもう少し単純な形になりますが，旧バージョンの結果ほどきれいにはなりませんでした。おそらく Sage のバージョンによって使用しているアルゴリズムが変更されたために，見た目が異なる結果が得られたのだと思われます。

47　インタラクティブな操作：@interact

Sage ノートブックの独自の機能の一つに@interact があります。これによって，式やグラフィックスなどをマウスを使ってインタラクティブに操作する事ができるようになります。これは Mathematica での Manipulate 機能に相当するものです。

簡単な例を使って@interact の使い方を紹介します。関数 f(x) を，引数 x に対して x^2 を画面に表示する関数である，と定義します：

```
def f(x):          # 関数 f を定義
    print x^2      # return ではなく print であることに注意
```

[*14] 境界 x_0, x_1 での値 $y(x_0)$ と $y(x_1)$ を指定することを境界条件を決めるという。

次に，この関数に対してインタラクティブな操作を行ってみましょう。Sage ノートブックで次を実行してみましょう：

```
@interact          # インタラクティブな操作をするためのおまじない
def f(x=[1,2,3,4,5]):    # 関数の中で引数の取る値を指定
    print x^2
```

番号付きのボタンをクリックすると，その下の実行結果が変わります。プログラムの中のリスト [1,2,3,4,5] がボタンの番号に対応しています。

上のものは最も簡単なインタラクティブな操作ですが，一般には次のような手順によって行います：

@interact の使い方

1. 実行したい処理を一つの関数として定義する
2. 関数の定義の直前に@interact と書く
3. 変化させたいパラメーターを関数の引数とする
4. パラメーターが変化する範囲を『引数＝』で指定する（例 x=(1,2,3,4,5)）

パラメーターがとる範囲は次のように指定することができる：

- リスト：x=[1,2,3,4,5]　(ボタン)
- タプル：x=(1,2,3,4,5)　(スライダー)
- 1 から 5 までの自然数：(1..5)　(操作はスライダー)
- 0.0 から 10.0 までの連続的な実数：x=(0,10)　(操作はスライダー)
- 5 から 10 までの 0.2 刻みの実数：(5,10,0.2)　（操作はスライダー）
- 1 から 8 までの実数をとるスライダーで，初期位置は 5　⇒　slider(1,8,default=5)
- 文字を入力する input_box(type=str)

slider と input_box はオプションを次のように指定するすることができる：

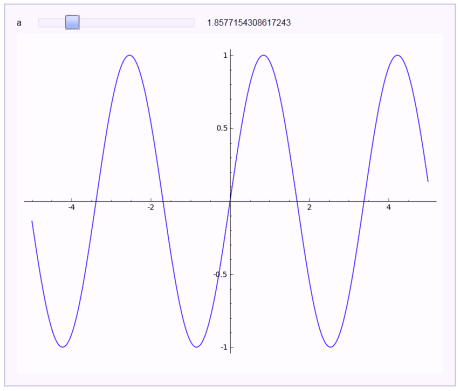
- input_box(default = "sin(x)",label = "hoge", type=str)
- slider(0,1,.01,.5,label = 'start value')

先ほどの例で，次のようにするとスライダーの取る範囲が-5 から 5 までの実数になります。

```
@interact
def f(x=(-5,5)):
    print x^2
```

@interact はパラメーターを含むグラフを描いて様子を見たいときに非常に便利です：

```
@interact
def f(a=(1,5)):                  # aをパラメーターとして1から5まで動かす
    p1 = plot(sin(a*x),-5,5) # p1をsin(ax)のグラフとする
    p1.show()                    # p1を表示する
```



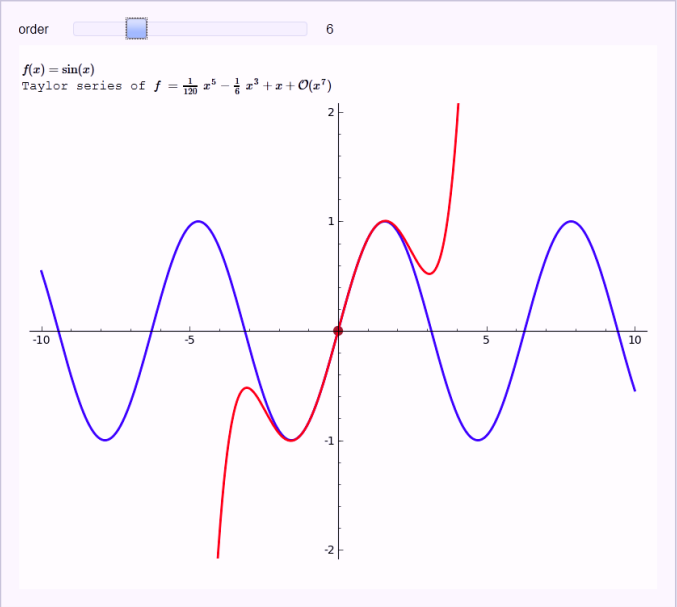
$\sin(x)$ とテイラー級数を同時に描いてみましょう：

```
x0 = 0
f    = sin(x)
p    = plot(f,-10,10)   # pはsin(x)のグラフ

@interact      # 以下で定義する関数をインタラクティブに操作
def show_taylor(nn=(1..20)):
    ft = f.taylor(x,x0,nn)                  #ftはfのnn次のテイラー級数
    pt = plot(ft,-10, 10)                   #ftのグラフをptとする
    show(p + pt, ymin = -2, ymax = 2)     # グラフを描画
```

次は，上のグラフの色を変えたりオプションを様々に追加したものです。

```
x0 = 0
f    = sin(x)
p    = plot(f,-10,10, thickness=2)   # pはsin(x)のグラフ
dot = point((x0,f(x=x0)), pointsize=80,rgbcolor='brown')    # 原点に点を打つ
@interact      # 以下で定義する関数をインタラクティブに操作
def show_taylor(nn=(1..20)):
   ft = f.taylor(x,x0,nn)                                #ftはfのテイラー級数
   pt = plot(ft,-10, 10, color='red', thickness=2) #ftのグラフをpt
   pretty_print( html('$f(x) = %s$'%latex(f)) )
   pretty_print(html('Taylor series of $f$ $= %s+O(x^{%s})$'%(latex(ft),nn+1)))
   show(dot + p + pt, ymin = -2, ymax = 2)     # グラフを描画
```



上のプログラムの 10 行目の %s は直後の latex(f) に置き換わります。同じく，11 行目の一つ目の %s は latex(ft)，二つ目の %s は order+1 に置き換わります。この記法についての解説は省略します。

次のようにすると，関数 sin(x) も固定したままではなく変化させることができます：

```
x0 = 0
@interact
def show_taylor(func=input_box(default="sin(x)", label="function", type=str),
                    order=(1..20), xmin=(-5,5), xmax=(0,5)):
    f(x) = eval(func)                  # 文字列 func を式に直したものが f(x)
    dot = point((x0,f(x=x0)),pointsize=80,rgbcolor='brown')  # 点
    p = plot(f(x),x,xmin,xmax, thickness=2)                  # f をプロット
    ft = f.taylor(x,x0,order)                      #ft は f の Taylor 展開
    pt = plot(ft,xmin,xmax, color='red', thickness=2)  #ft をプロット
    pretty_print(html('$f(x) = %s$'%latex(f)))
    pretty_print(html('Taylor series of $f$ $= %s+\mathcal{O}(x^{%s})$'
                      %(latex(ft),order+1)))
    show(dot + p + pt, ymin = -2, ymax = 2)   # グラフを表示
```

@interact については力学系やフラクタルなどおもしろい使い方が次の Web ページで紹介されています：

http://wiki.sagemath.org/interact

48　動画作成 (animate)

グラフィックスのオブジェクトのリストをつなぎ合わせて次のように簡単に動画（gif ファイル）を作ることができます：

```
def pic_sin(c):
    return plot(sin(x+c),(x,0,2*pi))

lis1 = [ pic_sin(c) for c in srange(0,2*pi,0.1)]
b = animate(lis1, figsize=[3,2])
```

```
b.show()
b.show(delay=2)
b.show(delay=3)
```

描画には計算時間が多少かかります。オプション delay でコマ送りの早さを指定します。ブラウザ上で保存したい gif 動画上で右クリックして，「名前を付けて画像を保存」を選ぶことで，gif ファイルを保存することができます。

48.1 高木関数の描画

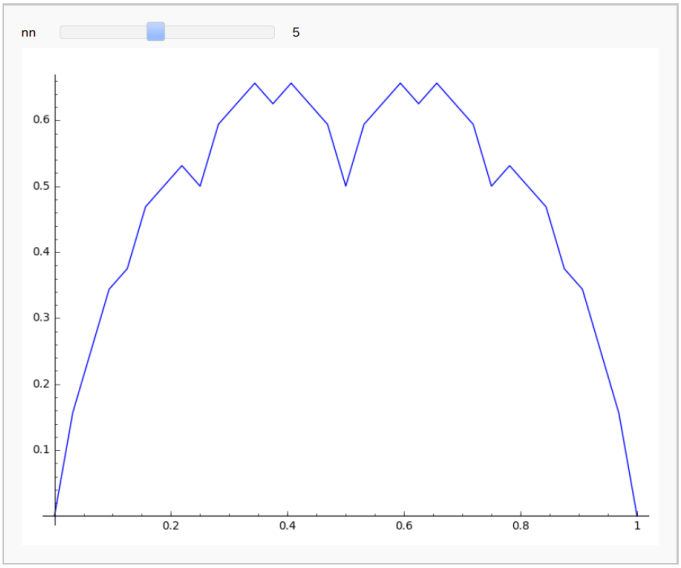
高木関数とは

$$T(x) = \sum_{n=0}^{\infty} \frac{s(2^n x)}{2^n} \tag{29}$$

で定義される関数である。ここに $s(x) = \min_{n\in\mathbb{Z}} |x - n|$。無限和を有限の和（最初の n 項の和）で近似することで，高木関数の様子を描画してみよう。

```
s = lambda x: abs(x-round(x))

@interact
def plotTakagi(nn=(1..10)):
    f = lambda x: sum( [s(2^k*x)/2^k for k in range(nn)] )
    plot(f,x,0,1).show()
```



高木関数のグラフをアニメーションにしてみよう。

```
def plotTakagi(nn):
    s = lambda x: abs(x-round(x))
    f = lambda x: sum( [s(2^k*x)/2^k for k in range(nn)] )
    return plot(f,x,0,1, legend_label='n='+str(nn))

figs = [plotTakagi(nn) for nn in range(1,17)]
animate(figs).show(delay=70)
```

48.2 高木関数をプロットする

高木関数のグラフ (takagi.gif) を生成する Sage プログラムは次のようになります。

- ファイル名：**PicTakagi.sage**

```
def plotTakagi(nn):
    s = lambda x: abs(x-round(x))
    f = lambda x: sum( [s(2^k*x)/2^k for k in range(nn)] )
    return plot(f,x,0,1, legend_label='n='+str(nn))

figs = [plotTakagi(nn) for nn in range(1,17)]
pp = animate(figs)
pp.gif(savefile='./takagi.gif', delay=70)
```

上のファイルを端末から

```
$ sage PicTakagi.sage
$
```

と実行すると，実行したフォルダに takagi.gif が作られます。