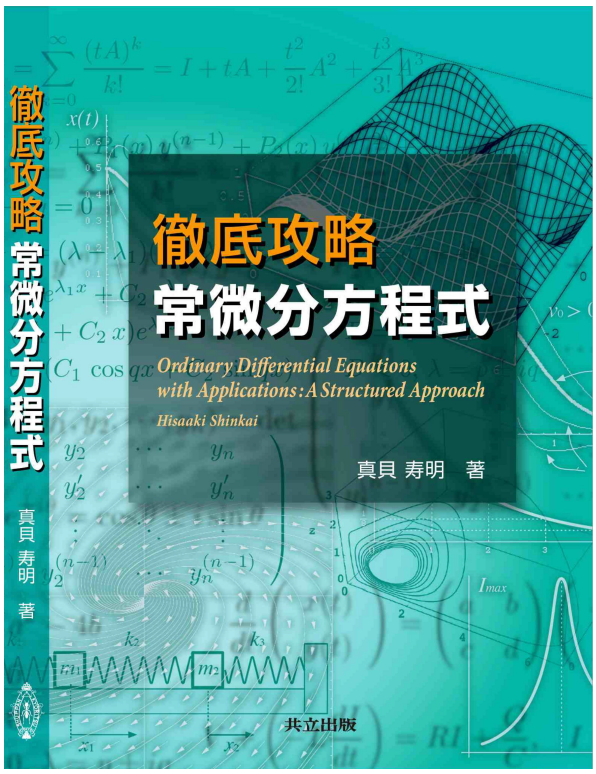
常微分方程式　数値的手法：Python を使う方法

『徹底攻略　常微分方程式』（真貝寿明，共立出版，2010）の第７章２節は，
「数値的手法：Mathematicaを使う方法」
として，Mathematicaの基本的な使い方と微分方程式を解く方法の解説をしています．

その部分を，宇都宮大学大学院工学研究科の上村佳嗣先生が，Pythonを用いる形式に翻訳されました．比較して眺めるのも面白いと思い，ご本人了承のもと，Python版をwebからダウンロード可能としたのが，このファイルです．



このファイルの内容に関するご質問等は，上村さん宛にお願いします．

上村佳嗣　　gami_at_is.utsunomiya-u.ac.jp
宇都宮大学大学院 工学研究科 情報システム科学専攻
321-8585 宇都宮市陽東7-1-2

本書の内容についてのご質問等は，著者までどうぞ．

真貝寿明　　hisaaki.shinkai_at_oit.ac.jp
大阪工業大学 情報科学部 情報システム学科
573-0196 大阪府枚方市北山 1-79-1

なお，本書の正誤表を
http://www.oit.ac.jp/is/~shinkai/book/index.html#ODE
にて用意しておりますので，ご利用ください．上村さんにはミスプリの指摘も多数いただきました．この場を借りまして，改めて御礼申し上げます．

2018年11月
真貝寿明

7 数値的手法

7.2 *Python* を使う方法

Python は，Python Software Foundation より配信されているフリーソフトウェアであり，書きやすく読みやすいため，プログラミング初心者にもおすすめな高水準言語である．開発者は，オランダ人の Guido van Rossum であり，名前の由来は BBC のコメディ番組「空飛ぶモンティ・パイソン」である．数値計算や数式処理，グラフィクス等の拡張パッケージを追加することで，幅広い領域の科学技術計算をこなすことができるのも人気の理由である．最近では，機械学習の基本ツールとしてよく用いられている．*Python* は，Linux/Mac OS/Windows などの多くの OS に対応している．Anaconda(miniconda) などの Python パッケージ・インストーラも提供されているので，個人での導入は非常に容易となっている．この機会にぜひ試してほしい．

2018 年 11 月時点のバージョンは 3.7.1.

♣Guido van Rossum ヴァンロッサム (1956-) Google を経て Dropbox に勤務．Python's BDFL.

本章では，*Python* の基本的な使い方と，微分方程式を解くまでの実例を紹介する．以下では，*Python*3 と Jupyter Notebook の利用を前提に記述する．

数式処理ができるソフトウェアとしては，他に Mathematica, Maple, MATLAB 上の MuPAD なども有名．

7.2.1 基本的な数式処理

Python と Jupyter Notebook を利用するにあたって，いくつかルールを書き出しておく．

- 数式処理や数値計算，グラフ描画を行うにはそれぞれ専用の拡張パッケージを呼び出す．呼び出せない場合は `pip` などで事前にインストールする．
- 入力する関数や命令は，大文字と小文字をきちんと区別する．
- 関数の引数は丸括弧 `(···)` でくくる．`sin(x)`, `cos(x)`, `tan(x)`, `log(x)` (=自然対数 $\log_e x$), `log(x,10)`(=常用対数 $\log_{10} x$), `exp(x)`(=指数関数 e^x)
- 行列やリストは `[···]` でくくる．
- 命令を実行するときは，「shift」キーを押しながら「return」，または「▷|」ボタンをクリックする．

拡張パッケージには，SymPy (数式処理), NumPy (数値計算), SciPy (科学技術計算), Matplotlib (グラフ描画) などがある．

例えば，Jupyter Notebook を起動し，「New」ボタンをクリックして，「Python 3」を選び，入力欄 (Cell) に

```
from sympy import *
from sympy.abc import x,y
init_printing()
```

と入力して「shift+return」したのち

```
(x+y)**2
```

と入力し再び「shift+return」してみよう．すると，画面に返される出力は

数式処理を行うには，`from sympy import *` により SymPy 直下のモジュールをすべて呼び出す．

変数は使用する前に `from sympy.abc import x,y` などでシンボル化する．

結果を LaTeX 風に表示するには `init_printing()` を入力する．

`**`は，べき乗を表す記号．

```
In [1]:  from sympy import *
         from sympy.abc import x,y
         init_printing()
In [2]:  (x+y)**2
```

```
Out[2]:  (x+y)**2
```

となる．**は，べき乗を表す記号である．入力行には，In [2]:などと表示され，出力行には Out[2]:と表示される．

さて，このように出力された直後，

```
expand(_)
```

と入力して「shift+return」しよう．expand は展開せよ，という関数コマンドであり，_ の記号は直前の計算出力を表す．画面に返される出力は，

直接 expand((x+y)**2) と入力しても同じ．

```
In [3]:  expand(_)
Out[3]:  x**2 + 2*x*y + y**2
```

となる．

続けていくつか入出力を試してみよう．

乗算を表すときは，*記号を使う．

```
In [4]:  _-4*x*y
Out[4]:  x**2 - 2*x*y + y**2
```

この答を因数分解するのが factor( ) 関数である．

```
In [5]:  factor(_)
Out[5]:  (x - y)**2
```

三角関数も試してみよう．π を使うときは，pi とする．

```
In [6]:  sin(pi/2)
Out[6]:  1
```

この例からわかるように三角関数の引数はラジアンで定義されている．度数表示を使うならば，pi/180 をかけて

```
In [7]:  sin(60*pi/180)
Out[7]:  sqrt(3)/2
In [8]:  N(_)
Out[8]:  0.866025403784439
```

などとする．N( ) は，数値評価をする関数コマンドである．

引数を x として変数のままにして，計算することも可能である．

初等関数

SymPy 関数	関数
exp(x)	e^x
a**x	a^x
sqrt(x)	$\sqrt{x}$
log(x)	$\log_e x$
log(x,a)	$\log_a x$
sin(x)	$\sin x$
cos(x)	$\cos x$
tan(x)	$\tan x$
asin(x)	$\sin^{-1} x$
acos(x)	$\cos^{-1} x$
sinh(x)	$\sinh x$
cosh(x)	$\cosh x$
abs( )	絶対値
re( )	実数部
im( )	虚数部

代数計算

SymPy 関数	意味
expand( )	展開
factor( )	因数分解
simplify( )	簡素化
.subs( )	代入
N( )	数値評価

特殊記号

SymPy	意味
pi	$\pi = 3.1415\cdots$
E	$e = 2.7182\cdots$
I	$i = \sqrt{-1}$

```
In [9]:  (1+x+x**2)**3
Out[9]:  (x^2 + x + 1)^3
```

この式の x に $x = 2$ を代入する方法がある．

```
In [10]:  _.subs(x,2)
Out[10]:  343
```

出力された答をさらに簡略化して表すのが，`simplify` 関数である．

```
In [11]:  sin(x)**2+cos(x)**2
Out[11]:  sin^2(x) + cos^2(x)
In [12]:  simplify(_)
Out[12]:  1
```

Python 独特の数式表現もあるが，すぐに慣れることだろう．

7.2.2 代数計算

方程式を解く機能として，次の 3 つの関数コマンドがある．

`solve(f(x),x)`	方程式 $f(x) = 0$ を x について解く．
`linarg.solve(A,b)`	線形方程式 $Ax = b$ を x について数値的に解く．
	NumPy の linarg モジュールを呼び出して使う．
`optimize.fsolve(f,x0)`	非線形方程式 $f(x) = 0$ の数値解を $x =$`x0` 付近で求める．
	SciPy の optimize モジュールを呼び出して使う．

例えば，方程式 $ax + b = 0$ を解いてみよう．

厳密には $a = 0$ のときは不定になるが，`solve` の出力は，そのような例外的な場合はユーザが除去することを前提としている．

```
In [1]:  from sympy import *
         from sympy.abc import a,b,c,x,y
         init_printing()
In [2]:  solve(a*x+b,x)
Out[2]:  [-b/a]
```

2 次方程式も解くことができる．

`solve` は，方程式に含まれるシンボルの数が式の数と同じときは第 2 引数を省略できる．

```
In [3]:  solve(a*x**2+b*x+c,x)
Out[3]:  [1/(2a)(-b + sqrt(-4ac + b^2)),  -1/(2a)(b + sqrt(-4ac + b^2))]
In [4]:  solve(x**2+2*x+3)
Out[4]:  [-1 - sqrt(2)i,  -1 + sqrt(2)i]
```

連立方程式 $\begin{cases} x+2y=-3 \\ x-2y=5 \end{cases}$ は次のようにして解くことができる．

連立方程式は，行列を用いて解いた方が見通しがよい．§7.2.7 で紹介する．

```
In [5]:  solve([x+2*y+3,x-2*y-5])
Out[5]:  [x : 1,  y : −2]
```

7.2.3 関数の定義，微分・積分

関数を微分するのは，**diff(関数，微分する変数)** という関数コマンドである．合成関数の微分も，偏微分も同じ関数コマンドで計算できる．なお，一変数関数の場合は第 2 引数が省略できる．

記号として計算を行った結果を出力していることに感動しませんか?

```
In [1]:  from sympy import *
         from sympy.abc import x,y
         init_printing()
In [2]:  diff(x**2)
Out[2]:  2x
In [3]:  diff(sin(x))
Out[3]:  cos(x)
In [4]:  diff(sin(x*y),y)
Out[4]:  x cos(xy)
```

代数計算

Python 関数	機能
`diff( , )`	導関数
`integrate( , )`	積分 (記号)
`quad( , , )`	積分 (数値)
`def f(x):`	関数の定義

関数を定義するのは，`def f(x):···` の形式である．

```
In [5]:  def f(x):
             return x/(1+x**2)
In [6]:  f(2)
Out[6]:  0.4
In [7]:  diff(f(x))
Out[7]:  −2x^2/(x^2+1)^2 + 1/(x^2+1)
```

積分は，**integrate(関数，積分変数)** という関数コマンドで実行できる．なお，一変数関数の場合は第 2 引数が省略できる．

```
In [8]:  integrate(f(x))
Out[8]:  (1/2) log(x^2+1)
```

積分定数が省かれているが，これが不定積分になる．定積分を行う場合は，例えば，$\int_0^2 f(x)dx$ であれば，

```
In [9]:  integrate(f(x),(x,0,2))
Out[9]:  1/2 log(5)
In [10]:  N(_)
Out[10]:  0.80471895621705
```

この結果を数値積分によって求めるには，SciPy の integrate モジュール内にある quad(関数名，下限，上限) 関数を用いる．同じ結果を出すことを確認しよう．

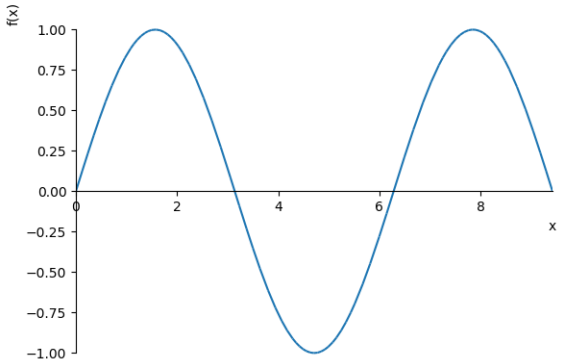
```
In [1]:  from scipy.integrate import quad
In [2]:  def f(x):
             x/(1+x**2)
In [3]:  quad(f,0,2)[0]
Out[3]:  0.8047189562170503
```

7.2.4　基本的な描画・プロット

次に，グラフを描いてみよう．

グラフを描く関数コマンドは plot(関数,(x,xmin,xmax)) 軸や線などに工夫をしたいときは，plot(関数,(x,xmin,xmax),option=value) として option を追加する．

```
In [1]:  %matplotlib inline
         from sympy import *
         from sympy.abc import x,y
         from sympy.plotting import plot3d
In [2]:  plot(sin(x),(x,0,3*pi))
```



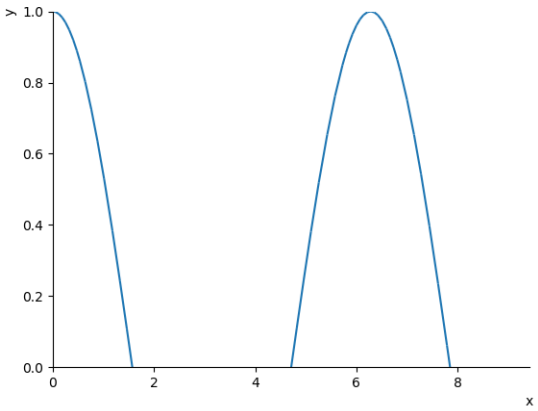

```
Out[2]:  <sympy.plotting.plot.Plot at 0x..........>
```

plot 関数は，x 軸と y 軸の比率や，y 軸の範囲を自動で調整してグラフを描くが，ユーザーが必要に応じてこれらの設定をすることもできる．次は y 軸の範囲を $0 \leq y \leq 1$ として $\cos x$ をプロットする例である．オプションで y 軸に y と記入する．

plot の主なオプション

オプション	機能
ylim	縦軸の範囲
xlabel	横軸の名称
ylabel	縦軸の名称
axis_center	軸の交差点
show	表示フラグ
line_color	線の色

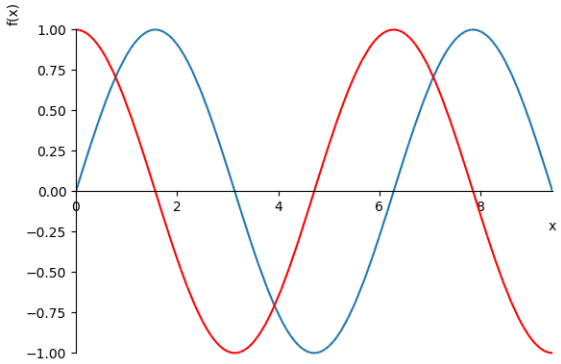
```
In [3]:  plot(cos(x),(x,0,3*pi),ylim=(0,1),ylabel='y')
```




```
Out[3]:  <sympy.plotting.plot.Plot at 0x..........>
```

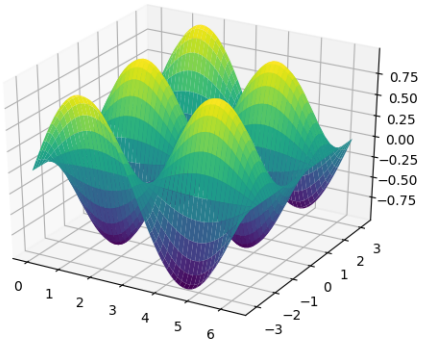
複数の図を重ねて表示することも可能である．plot(sin(x),cos(x),(x,0,3*pi)) としてもよいが，次の例を紹介する．

```
In [4]:  a1=plot(sin(x),(x,0,3*pi),show=False)
         a2=plot(cos(x),(x,0,3*pi),show=False, line_color='r')
         a1.extend(a2)
         a1.show()
```



2 変数関数のグラフは，plotting モジュール内の 3 次元プロット関数 plot3d を使う．

```
In [5]:  plot3d(sin(x)*cos(2*y),(x,0,2*pi),(y,-pi,pi)
```




```
Out[5]:  <sympy.plotting.plot.Plot at 0x..........>
```

7.2.5　微分方程式を解く

いよいよ微分方程式を解く関数を紹介しよう．

`dsolve(eqn, x(t))`

t に関する微分方程式 `eqn=0` を $x(t)$ について解析的に解く．

Sympy パッケージを呼び出して使う．

`odefun(func, t0, x0)`

t に関する微分方程式 x' =`func` を初期条件 x(`t0`) =`x0` のもとで，$x(t)$ について級数展開による近似解を求める．

Mpmath パッケージを呼び出して使う．

`odeint(func, x0, t, args=(...))`

t に関する微分方程式 x' =`func` を初期条件 $x(0)$ =`x0` のもとで，x(`t`) について数値的に解く．

Scipy の Integrate パッケージを呼び出して使う．

まず，1 階の微分方程式 $\dfrac{dx}{dt} = ax$ を解いてみよう．`dsolve` については，求める関数が 1 つならば第 2 引数は省略できる．

```
In [1]:  from sympy import *
         init_printing()
         t,a,k=symbols('t,a,k',real=True)
         x,y=symbols('x,y',cls=Function)
In [2]:  dsolve(diff(x(t)) - a*x(t))
Out[2]:  x(t) = C1*e^(a*t)
```

この一般解に初期条件を代入して，未定の定数 C_1 を確定できる．

```
In [3]:  dsolve(diff(x(t)) - k*x(t), ics={x(0):2})
```
Out[3]: $x(t) = 2e^{at}$

2 階の微分方程式 $\dfrac{d^2x}{dt^2} = -k^2x$ と $\dfrac{d^2x}{dt^2} = k^2x$ を解いてみよう．

```
In [4]:  dsolve(diff(x(t),t,2) + k**2*x(t))
```
Out[4]: $x(t) = C_1 \sin(t|k|) + C_2 \cos(kt)$
```
In [5]:  dsolve(diff(x(t),t,2) - k**2*x(t))
```
Out[5]: $x(t) = C_1 e^{-kt} + C_2 e^{kt}$

ところで，上の 2 階の微分方程式の右辺は，係数が $k^2 > 0$ のために，解が三角関数の場合と指数関数の場合で明確に区別できている．ところがこれを $\dfrac{d^2x}{dt^2} = kx$ として解こうとするとき，人間ならば，k の正負で場合分けして解くことになるが，$Python$ では残念ながら勝手に $k > 0$ と解釈してしまうようだ．

```
In [6]:  dsolve(diff(x(t),t,2) - k*x(t))
```
Out[6]: $x(t) = C_1 e^{-\sqrt{k}t} + C_2 e^{\sqrt{k}t}$

2 階の微分方程式で初期条件を与える例も示しておく．当然初期条件を 2 つ与えないと，係数は決まらない．

```
In [7]:  dsolve(diff(x(t),t,2) + k**2*x(t), ics={x(0):1,
             diff(x(t)).subs(t,0):0})
```
Out[7]: $x(t) = \cos(kt)$

連立方程式も解くことができる．$\dfrac{dx}{dt} = y(t), \dfrac{dy}{dt} = x(t)$ を解くのは，次のようにすればよい．

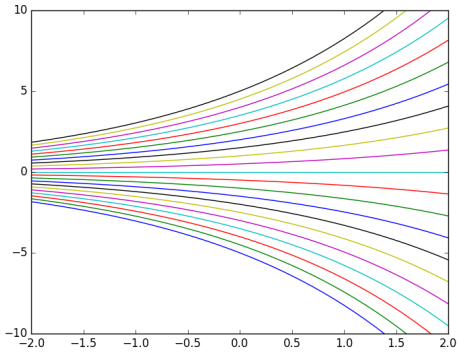
```
In [8]:  dsolve([diff(x(t)) - y(t), diff(y(t)) - x(t)])
```
Out[8]: $[x(t) = C_1 e^t + C_2 e^{-t},\ \ y(t) = C_1 e^t - C_2 e^{-t}]$

7.2.6　ベクトル場を描く

微分方程式 $\dfrac{dy}{dx} = \dfrac{1}{2}y$ の一般解は，$y = Ce^{x/2}$ である．一般解は曲線群であることをグラフにして表そう．

次の例では，定数 C を $C = -5.0, -4.5, -4.0, \cdots, +5.0$ としてグラフを $-2 \leq x \leq 2,\ -10 \leq y \leq 10$ の範囲で描いた．

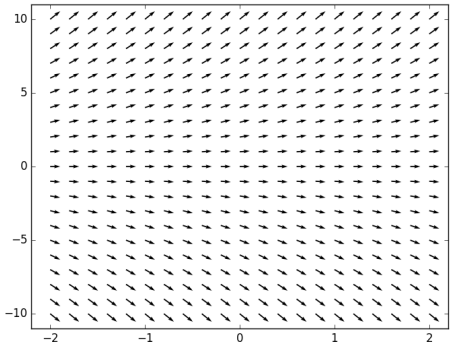
```
In [1]:  from pylab import *
In [2]:  x=linspace(-2,2,41)
In [3]:  [plot(x,C*exp(x/2))
             for C in linspace(-5,5,21)]
         xlim(-2,2)
         ylim(-10,10)
         show()
```



Pylab は，Numpy と Matplot の Pyplot モジュールを同時に呼び出してくれるインターフェースで，MATLAB ライクな使い方が可能になる．

ベクトル場を表示させるためには，meshgrid と quiver を使う．

```
In [4]:  X,Y=meshgrid(linspace(-2,2,21),linspace(-10,10,21))
In [4]:  quiver(X,Y,5/0.8,Y/2)
         xlim(-2.2,2.2)
         ylim(-11,11)
         show()
```



meshgrid は，x 座標と y 座標の目盛配列からグリッドの x 成分行列と y 成分行列を作る．
quiver の引数は，グリッドの x 成分行列と y 成分行列，およびベクトルの x 成分行列と y 成分行列．

7.2.7 行列計算

行列 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ の入力は，[[a,b],[c,d]] のような形式で行う．行列に関する主な関数を挙げておく．

ベクトルも行列も同じ形式で入出力や積の計算が可能．

`A.det()`	行列 A の行列式を出力する.
`A.inv(), Inverse(A)`	行列 A の逆行列を出力する.
`A.T, A.transpose()`	行列 A の転置行列を出力する.
`A.eigenvals()`	行列 A の固有値を出力する.
`A.eigenvects()`	行列 A の固有ベクトルを出力する.
`A*B, A.multiply(B)`	行列 A と行列 B の積を出力する.
`A**n`	行列 A の n 乗を出力する.
`A.multiply_elementwise(B)`	行列 A と行列 B の各要素の積の成分で行列を作る.

`A*A` と `A**2` は，同じ結果となる.

```
In [1]: from sympy import *
        from sympy.abc import a,b,c,d,x,y
        init_printing()
In [2]: A=Matrix([[a,b],[c,d]])
        A
Out[2]: ⎡a  b⎤
        ⎣c  d⎦
In [3]: A*A
Out[3]: ⎡a² + bc   ab + bd⎤
        ⎣ac + cd   bc + d²⎦
In [4]: A.multiply_elementwise(A)
Out[4]: ⎡a²   b²⎤
        ⎣c²   d²⎦
```

§7.2.2 で解いた連立方程式 $\begin{cases} x+2y=-3 \\ x-2y=5 \end{cases}$ を行列とベクトルの積

$\begin{pmatrix} 1 & 2 \\ 1 & -2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}=\begin{pmatrix} -3 \\ 5 \end{pmatrix}$ と書き換えて解いてみよう.

```
In [5]: A=Matrix([[1,2],[1,-2]])
        u=Matrix([x,y])
        B=Matrix([-3,5])
        solve(Eq(A*u,B))
Out[5]: [{x : 1,  y : -2}]
In [6]: A.inv()*B
Out[6]: ⎡1 ⎤
        ⎣-2⎦
```

どちらで解いても同じ答えが得られた.

7.2.8 練習問題

これまでの説明で，*Python* を用いる基本的な操作は理解できたと思う．練習問題を挙げておこう．

問題 7.8

1. $f(x) = x^3 - 3x + 2$ とする．
   （a）グラフを $-3 \leq x \leq 3$ で描け．
   （b）$f(x)$ を因数分解せよ．
   （c）$x^3 - 3x + 2 = 0$ を解け．
2. $f(x) = e^{-x^2}$ とする．
   （a）グラフを $-3 \leq x \leq 3$ で描け．
   （b）$f(x)$ を微分せよ．
   （c）$f'(x)$ のグラフを $-3 \leq x \leq 3$ で描け．
3. $y = 2e^{-3(x-4)^2}$ を微分せよ．また，得られた答を積分してもとに戻ることを確かめよ．

問題 7.9 関数 $x(t)$ についての微分方程式の問題

1. $x' + 3x = 0$ を解け．
2. $x' + 3x = 0$ を初期条件 $x(0) = 5$ のもとで解け．
3. $x'' + 3x = 0$ を解け．
4. $x'' + 3x = 0$ を初期条件 $x(0) = 5, x'(0) = -3$ のもとで解け．
5. $x'' + 3x' + 2x = 0$ を解け．
6. $x'' + 3x' + 2x = e^t$ を解け．
7. $x'' + 3x' + 2x = e^{-t}$ を解け．

次は，微分方程式の解をそのままグラフにしてみよう．

問題 7.10 関数 $x(t)$ に対する 2 階の微分方程式で右辺に強制振動項がある場合を考えよう．いずれも初期条件は，$x(0) = 0, x'(0) = 1$ とする．

1. $x'' + x = 0$ を解き，結果をグラフ $(0 \leq t \leq 20\pi)$ で示せ．
2. $x'' + x = 2\sin 7t$ を解き，結果をグラフで示せ．
3. $x'' + x = 2\sin t$ を解き，結果をグラフで示せ．
4. 上記の (2) と (3) の結果の本質的な違いは何か．

(1) `dsolve` で $x(t)$ について解けたならば，`plot(_.rhs,(t,0,20*pi))` とすればよい．

7.2.9 単振り子の近似限界

例題 3.10 で扱った振り子の問題を *Python* で解いてみよう．振り子の振れ角 θ を時間 t の関数として運動方程式を立てる．重力加速度を g，振り子の長さを ℓ とする．例題 3.10 で説明したように，結果は，θ が小さいときには $\sin\theta \simeq \theta$ と近似して，微分方程式

$$\frac{d^2\theta}{dt^2} + \frac{g}{\ell}\theta = 0 \tag{1}$$

で近似される．これを *Python* を用いて解くと，

```
In [1]: %matplotlib inline
        from sympy import *
        from sympy.abc import t,g,l
        init_printing()
In [2]: theta=Function('theta')
In [3]: eq1=theta(t).diff(t,2)+(g/l)*theta(t)
        dsolve(eq1)
```

Out[3]: $\theta(t) = C_1 e^{-t\sqrt{-\frac{g}{l}}} + C_2 e^{t\sqrt{-\frac{g}{l}}}$

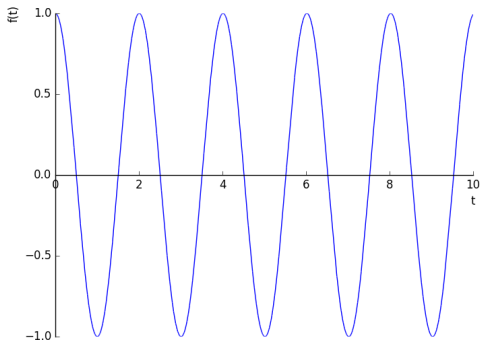
グラフを描くために，$g = 9.8$ m/s^2, $\ell = 1$ m とおき，さらに初期条件を $\theta(0) = 1$, $\theta'(0) = 0$ として具体的に求めてみよう．

```
In [4]: l=1
        g=9.8
        eq1=theta(t).diff(t,2)+(g/l)*theta(t)
        dsolve(eq1, ics={theta(0):1,
            diff(theta(t)).subs(t,0):0})
```

Out[4]: $\theta(t) = \cos\left(\frac{7\sqrt{5}t}{5}\right)$

特殊解は直前の計算出力として記憶されているので，その右辺を `plot` 関数でグラフにする．

```
In [5]: plot(_.rhs,(t,0,10))
```




```
Out[5]:  <sympy.plotting.plot.Plot at 0x.......>
```

約 2 秒を 1 周期とする振り子の解が得られた．

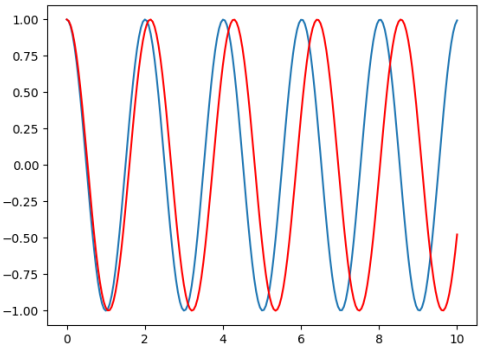
ところで，上の解は数学的には正しいが，実際の振り子に適用するには，初期条件の $\theta = 1$ rad が大きすぎて微分方程式 (7.2.20) が成り立たず，物理的には誤っている．それを確かめるために，θ に関して近似しない微分方程式

$$\frac{d^2\theta}{dt^2} + \frac{g}{\ell}\sin\theta = 0 \tag{2}$$

を数値的に解いてみよう．

数値的に解くために，Scipy の integrate モジュールにある `odeint` を用いる．2 階の微分方程式を $\begin{cases} \dfrac{dx_0}{dt} = x_1(t) \\ \dfrac{dx_1}{dt} = -\dfrac{g}{\ell}\sin(x_0(t)) \end{cases}$ のように 1 階の連立微分方程式とする．

```
In [1]:  from scipy.integrate import odeint
         from pylab import *
In [2]:  def f1(x,t,g,l):
             return [x[1],-g/l*x[0]]
In [3]:  def f2(x,t,g,l):
             return [x[1],-g/l*sin(x[0])]
In [4]:  l=1
         g=9.8
         x0=[1,0]
         t=linspace(0,10,201)
In [5]:  x1=odeint(f1,x0,t,args=(g,l)).T
         x2=odeint(f2,x0,t,args=(g,l)).T
In [6]:  plot(t,x1[0])
         plot(t,x2[0],'r')
         show()
```



となって，近似せずに解いたほうが，周期が若干長くなることがわかる．振り子の等時性は (7.2.20) 式を解いた結果いえたことなので，振れ角が大きいと，振り子の等時性は成り立たないことになる．

(7.2.21) でも，初期条件の θ の値を小さくすれば，(7.2.20) を解いた結果と一致するはずである．各自確かめてみよう．

7.2.10　3 種類の生物の Lotka-Volterra モデル

3 種類の生物系の Lotka-Volterra モデル
2 種類の生物による捕食者/被食者モデル⇒例題 4.14

例題の式の係数は，適当にスケール変換することによって，実質 6 つの係数まで減らすことができるが，ここでは数値を代入するときの意味づけから，10 個のままとしている．

解析的には解けない問題にも挑戦してみよう．§4.6.4 の例題 4.14 で扱った，生物の捕食者/被食者モデル (Lotka-Volterra モデル) を，3 種類の生物の系に拡大して調べてみよう．

例題 7.11　孤立した池に大魚 X，小魚 Y，プランクトン Z が棲む．それぞれの個体数を時間 t の関数として，$x(t), y(t), z(t)$ とする．大魚は，小魚とプランクトンを餌としており，小魚はプランクトンのみを餌としている．それぞれのしたがう微分方程式を

$$\begin{aligned}\frac{dx}{dt} &= -x(a_0 - a_y y - a_z z)\\ \frac{dy}{dt} &= -y(b_0 + b_x x - b_z z)\\ \frac{dz}{dt} &= z(c_0 - c_x x - c_y y - c_z z)\end{aligned}$$

とする．初期時刻の個体数を $(x(0), y(0), z(0)) = (1, 2, 3)$ として，係数に次の値を代入することにより，解の振る舞いを調べよ．

(1) $(a_0, a_y, a_z; b_0, b_x, b_z; c_0, c_x, c_y, c_z) = (1, 1, 1; 1, 1, 1; 1, 1, 1, 1)$

(2) $(a_0, a_y, a_z; b_0, b_x, b_z; c_0, c_x, c_y, c_z) = (1, 1, 1; 1, 1, 3; 2, 2, 2, 1)$

`odeint` に渡す微分方程式の右辺 `func(x,t,a,b,c)` を定義し，さらにいろいろパラメータを変えて計算できるように，`odeint` を含めた一連の計算を 1 つの関数 `f(....)` にしておこう．

```
In [1]:  from scipy.integrate import odeint
         from pylab import *
         from mpl_toolkits.mplot3d import Axes3D
```

```
In [2]:  def func(x,t,a,b,c):
             return [-x[0]*(a[0]-a[1]*x[1]-a[2]*x[2]),
                 -x[1]*(b[0]+b[1]*x[0]-b[2]*x[2]),
                 x[2]*(c[0]-c[1]*x[0]-c[2]*x[1]-c[3]*x[2])]
In [3]:  def f(a0,ay,az,b0,bx,bz,c0,cx,cy,cz,x0,y0,z0,tmax):
             a=[a0,ay,az]
             b=[b0,bx,bz]
             c=[c0,cx,cy,cz]
             x=[x0,y0,z0]
             t=linspace(0,tmax,1001)
             return [t,odeint(func,x,t,args=(a,b,c))]
```

こうしておくと，例題の (1) を $t = 20$ まで解くときには，

```
In [4]:  tmax=20
         t,result=f(1,1,1, 1,1,1, 1,1,1,1, 1,2,3, tmax)
```

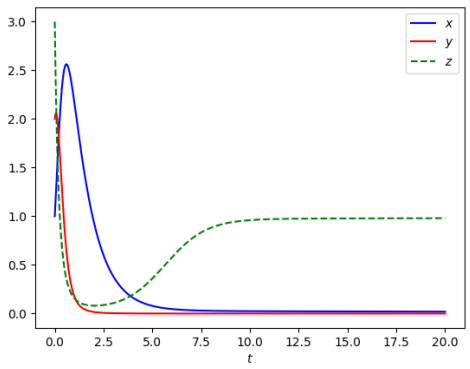
とすればよい．この結果をグラフにするには，例えば，

```
In [5]:  plot(t,result)
In [6]:  xlabel('t')
In [7]:  result=result.T
In [8]:  show()
```

あるいは，

```
In [5]:  result=result.T
In [6]:  plot(t,result[0],'b',label='x')
         plot(t,result[1],'r',label='y')
         plot(t,result[2],'g--',label='z')
In [7]:  xlabel('t')
         legend()
In [8]:  show()
```

とすればよい．

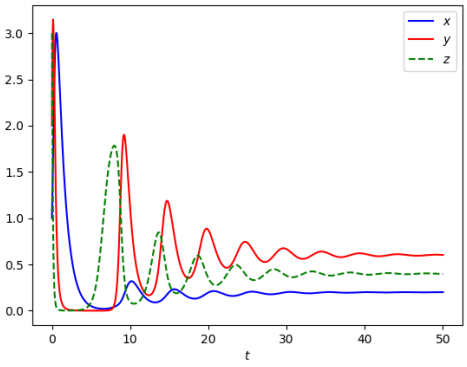


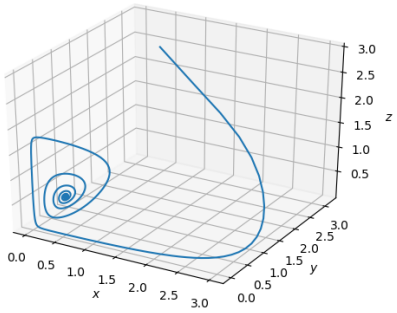
(1) の係数では，大魚 X（図の青線）と小魚 Y（赤線）は絶滅し，プランクトン Z（緑線）ははじめの 3 分の 1 の個体数で残るようである．

(2) の係数では，どうなるだろうか．

```
In [9]:  tmax=50
         t,return=f(1,1,1, 1,1,3, 2,2,2,1, 1,2,3, tmax)
```

として，同様にグラフにすると下図左のようになる．この場合は，共存して一定数に落ち着いていくようだ．下図右は，時間発展の様子を (x, y, z) 空間で 1 本の線として描いたものである．





3 次元空間の軌跡は，Axes3D モジュールの `plot` 関数を用いた．

```
In [10]:  fig=figure()
In [11]:  ax=fig.gca(projection='3d')
In [12]:  ax.plot(result[0],result[1],result[2])
In [13]:  show()
```

問題 7.12　例題 7.11 のモデルで，同じ初期条件のとき，次のような結果になるパラメータを探してみよう．そして，3 次元プロットで，(x, y, z) 空間での解の軌跡も図示せよ．

(1) 小魚が全滅し，大魚とプランクトンが一定数に近づいていくとき．

(2) 3 種が，すべて周期的に変化しながら共存するとき．

参考文献

[1] Python Japan, `https://www.python.jp`

[2] Anaconda home, `https://www.anaconda.com`

[3] Jupyter home, `https://jupyter.org`

[4] SciPy home, `https://scipy.org`

[5] NumPy home, `http://www.numpy.org`

[6] SymPy home, `http://sympy.org`

[7] Matplotlib home, `https://matplotlib.org`

[8] SciPy Lecture Notes, `https://www.scipy-lectures.org`
などを参考にするとよい．