物体運動のシミュレーション

宇都宮大学　吉田勝俊

学習内容

1. 運動と運動方程式
2. 微分方程式の数値計算
3. 数値解のアニメーション
4. 大気中の質点の放物運動
5. 運動の設計(≒条件の調整)

※ Python では，必要な機能を「インポート」して使います．

```
import numpy as np                  #数値計算機能のインポート 別名np
import matplotlib.pyplot as plt     #グラフ描画機能のインポート 別名plt
from scipy.integrate import odeint #常微分方程式の数値解法をインポート
from matplotlib.animation import FuncAnimation #アニメーション機能のインポート
from matplotlib import rc           #各種設定機能のインポート
rc('animation', html='jshtml')     #Colabでアニメーション表示可能にするための設定
```

---

§1 運動と運動方程式

- 運動　⇔　物体の位置 x の時間変化 $x(t)$ のこと!
- 運動方程式　⇔　物体の運動 $x(t)$ を求める方程式のこと!

高校物理の運動方程式

$ma(t) = f$　　(質量)×(加速度)=(力)

記号	名称	説明
m	質量	運動(位置の時間変化)ではない
f	力	同上
$a(t)$	加速度	同上

結論

- 高校の運動方程式を解いても，「加速度」しか求まらない．
- 運動方程式なのに，「運動」は求まらない!

大学物理の運動方程式

$mx''(t) = f$　　(質量)×(加速度)=(力)

記号	名称	説明
m	質量	運動(位置の時間変化)ではない
f	力	同上
$x(t)$	運動	位置の時間変化!

微分法

記号	名称	説明	数学
$x(t)$	運動	位置の時間変化	
$v(t) = x'(t)$	速度	位置 $x(t)$ のグラフの傾き	運動の微分
$a(t) = v'(t) = x''(t)$	加速度	速度 $v(t)$ のグラフの傾き	速度の微分

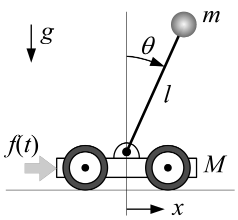
- $a(t) = v'(t) = \boxed{x'(t)}' = x''(t)$

結論

- 大学の運動方程式を解くと,「運動」が求まる!

運動方程式の解き方

- 手計算
  - 長所･･･運動が数式で求まる　→条件の逆算(設計という)がしやすい
  - 短所･･･対象がちょっとでも複雑だと解けない
- 数値解法(コンピュータによる近似計算のこと)
  - 短所･･･運動は数式ではなく,数列として求まる
  - 長所･･･ロボットだろうが,スペースシャトルだろうが解ける

> ★機械構造としては,超シンプルなのに,手計算では解けない運動方程式の例
>
> 
>
>
> 図:台車型倒立振子(しんし / ふりこ)
>
> $$\begin{cases} (M + m)x'' + (ml\cos\theta)\theta'' - ml\theta'^2 \sin\theta = f(t) \\ (ml\cos\theta)x'' + (ml^2)\theta'' - mlg\sin\theta = 0 \end{cases}$$
>
> - 一次式でない項 $\sin\theta, \cos\theta, \theta'^2$ があるため,この方程式は数学的に解けない(非線形方程式といいます)
>
> [〔YouTube〕inverted pendulum with mini4WD model car]

§2 微分方程式の数値計算

- 未知数の **微分を含む** 方程式を **微分方程式** といいます.
- 運動方程式は,微分方程式の一種です.

《例題1》$v' = -0.5v$ （水の抵抗を受けるボートの運動方程式）

- 運動方程式より簡単な方程式で,解き方を説明します.
- 未知数が速度 $v(t)$ だけなので,簡単に解けます.

〔運動方程式のプログラミング〕

```
def Boat(v, t):
  '''
  《例題１》の運動方程式
  '''
  dvdt = -0.5*v # 運動方程式
  return dvdt
```

〔時間軸の作成〕

```
t0 = 0              # 初期時刻
dt = 0.05           # 時間ステップ
tn = 130            # データ長
t1 = t0 + dt*(tn-1) # 終端時刻

t = np.linspace(t0, t1, tn) # 時間軸を表す等差数列
print(t)
```

```
[0.   0.05 0.1  0.15 0.2  0.25 0.3  0.35 0.4  0.45 0.5  0.55 0.6  0.65
 0.7  0.75 0.8  0.85 0.9  0.95 1.   1.05 1.1  1.15 1.2  1.25 1.3  1.35
 1.4  1.45 1.5  1.55 1.6  1.65 1.7  1.75 1.8  1.85 1.9  1.95 2.   2.05
 2.1  2.15 2.2  2.25 2.3  2.35 2.4  2.45 2.5  2.55 2.6  2.65 2.7  2.75
 2.8  2.85 2.9  2.95 3.   3.05 3.1  3.15 3.2  3.25 3.3  3.35 3.4  3.45
 3.5  3.55 3.6  3.65 3.7  3.75 3.8  3.85 3.9  3.95 4.   4.05 4.1  4.15
 4.2  4.25 4.3  4.35 4.4  4.45 4.5  4.55 4.6  4.65 4.7  4.75 4.8  4.85
 4.9  4.95 5.   5.05 5.1  5.15 5.2  5.25 5.3  5.35 5.4  5.45 5.5  5.55
 5.6  5.65 5.7  5.75 5.8  5.85 5.9  5.95 6.   6.05 6.1  6.15 6.2  6.25
 6.3  6.35 6.4  6.45]
```

〔初速度の設定〕

```
v0 = 1 # 初速度〔m/s〕
```

〔運動方程式の数値解法〕 速度の時間変化 $v(t)$ を求める

```
vt = odeint(Boat, v0, t) # これで解ける
```

解 $v(t)$ は，時間軸 `t` の各時刻における速度の「数列」として得られる．

時刻の数列 `t`	0.05	0.1	0.15	0.2	⋯
速度の数列 `v`	$v(0.05)$	$v(0.1)$	$v(0.15)$	$v(0.2)$	⋯

```
print(vt)
```

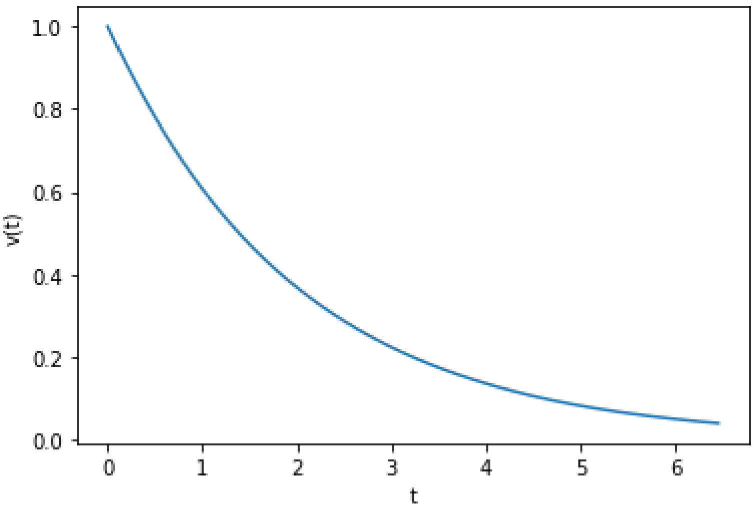
```
 [0.16529888]
 [0.16121764]
 [0.15723716]
 [0.15335496]
 [0.14956861]
 [0.14587575]
 [0.14227406]
 [0.1387613 ]
 [0.13533527]
 [0.13199383]
 [0.12873489]
 [0.12555642]
 [0.12245642]
 [0.11943296]
 [0.11648415]
 [0.11360814]
 [0.11080315]
 [0.10806741]
 [0.10539921]
 [0.1027969 ]
 [0.10025883]
 [0.09778343]
 [0.09536915]
 [0.09301448]
 [0.09071794]
 [0.08847811]
 [0.08629357]
 [0.08416298]
 [0.08208499]
 [0.0800583 ]
 [0.07808165]
 [0.07615381]
 [0.07427357]
 [0.07243974]
 [0.0706512 ]
 [0.06890682]
 [0.0672055 ]
 [0.06554619]
 [0.06392785]
 [0.06234946]
 [0.06081005]
 [0.05930864]
 [0.05784431]
 [0.05641613]
 [0.05502321]
 [0.05366468]
```

```
 [0.05233969]
 [0.05104742]
 [0.04978706]
 [0.04855781]
 [0.04735892]
 [0.04618962]
 [0.0450492 ]
 [0.04393693]
 [0.04285212]
 [0.0417941 ]
 [0.0407622 ]
 [0.03975578]]
```

〔解のグラフ表示〕 （横軸, 縦軸）=（時刻, 速度）のグラフ用紙に解を描く

```
v = vt[:,0] # 時間軸 t に合せて，解も横ベクトルにしておく

plt.plot(t, v)      # グラフの作成
plt.xlabel('t')     # 横軸のラベル
plt.ylabel('v(t)')  # 縦軸のラベル
```

```
Text(0, 0.5, 'v(t)')
```



〔解であることのチェック〕

$v' - (-0.5v) = v' + 0.5v$ が 0 になるはず!

```
dvdt = np.gradient(v, t, edge_order=2) # v の時間微分

print(dvdt + 0.5*v)
```

```
[ 1.02193195e-04 -5.05474894e-05 -4.92982552e-05 -4.82729374e-05
 -4.71608744e-05 -4.60468050e-05 -4.48805639e-05 -4.37352330e-05
 -4.26547830e-05 -4.16160376e-05 -4.07045752e-05 -3.98699213e-05
 -3.89156168e-05 -3.77595820e-05 -3.65365051e-05 -3.54070149e-05
 -3.44849295e-05 -3.38759989e-05 -3.33081707e-05 -3.25430887e-05
 -3.16495721e-05 -3.07606112e-05 -2.99807740e-05 -2.92766220e-05
 -2.85794559e-05 -2.78738675e-05 -2.71868357e-05 -2.65111198e-05
 -2.58553595e-05 -2.52189632e-05 -2.45916823e-05 -2.39848768e-05
```

```
 -2.33968030e-05 -2.28160168e-05 -2.22521059e-05 -2.16904222e-05
 -2.11214384e-05 -2.05717959e-05 -2.00650780e-05 -1.96142935e-05
 -1.91851579e-05 -1.87525574e-05 -1.83206292e-05 -1.78650653e-05
 -1.73800602e-05 -1.69292526e-05 -1.65286747e-05 -1.61325144e-05
 -1.57247491e-05 -1.53098865e-05 -1.49195239e-05 -1.45556411e-05
 -1.41999169e-05 -1.38481476e-05 -1.35035276e-05 -1.31732181e-05
 -1.28541260e-05 -1.25397830e-05 -1.22258719e-05 -1.19181978e-05
 -1.16241276e-05 -1.13396657e-05 -1.10606568e-05 -1.07850778e-05
 -1.05171067e-05 -1.02598816e-05 -1.00095921e-05 -9.76311288e-06
 -9.51930238e-06 -9.28614536e-06 -9.06632845e-06 -8.85240888e-06
 -8.64000294e-06 -8.42372163e-06 -8.20145674e-06 -7.99060197e-06
 -7.79811010e-06 -7.61179960e-06 -7.42787199e-06 -7.24252434e-06
 -7.05450250e-06 -6.87625240e-06 -6.71216473e-06 -6.55300570e-06
 -6.39545257e-06 -6.23578498e-06 -6.07301560e-06 -5.91906899e-06
 -5.77787147e-06 -5.64074841e-06 -5.50487150e-06 -5.36706503e-06
 -5.22673653e-06 -5.09412726e-06 -4.97235143e-06 -4.85406597e-06
 -4.73699511e-06 -4.61849942e-06 -4.49827213e-06 -4.38479350e-06
 -4.28034702e-06 -4.17875124e-06 -4.07802531e-06 -3.97577325e-06
 -3.87176839e-06 -3.77400336e-06 -3.68418849e-06 -3.59652635e-06
 -3.50969553e-06 -3.42161186e-06 -3.33263119e-06 -3.24987685e-06
 -3.17448407e-06 -3.10116945e-06 -3.02862044e-06 -2.95537524e-06
 -2.87959217e-06 -2.79996327e-06 -2.72160422e-06 -2.64781291e-06
 -2.57587948e-06 -2.50603934e-06 -2.43999206e-06 -2.38025236e-06
 -2.33003975e-06 -2.28185603e-06 -2.22642659e-06 -2.16793978e-06
 -2.11085988e-06  4.23515631e-06]
```

- `e-05` は $\times 10^{-5}$ $(\times 0.00001)$ のコンピュータ表記.
- ゆえに, 数値解の誤差は 0.00001 程度だったということ.
- 無視はできない大きさだが, 実用上はこの程度の誤差でよしとする.

《例題1の続き》　位置$x(t)$も知りたいんですけど!

- 元の運動方程式 $v' = -0.5v$ に,
- 位置と速度の関係式 $x' = v$ を連立します.

$$\begin{cases} x' = v \\ v' = -0.5v \end{cases}$$

〔運動方程式のプログラミング〕　今度は2連立

```
def Boat2(xv, t):
  '''
  《例題１の続き》の運動方程式
  '''
  x, v = xv

  dxdt = v      # 追加した方程式
  dvdt = -0.5*v #《例題１》の運動方程式

  return [dxdt, dvdt] #２式をまとめて返す
```

〔時間軸の作成〕

```
t0 = 0              # 初期時刻
dt = 0.05           # 時間ステップ
tn = 130            # データ長
t1 = t0 + dt*(tn-1) # 終端時刻

t = np.linspace(t0, t1, tn) # 時間軸を表す等差数列
print(t)
```

```
[0.   0.05 0.1  0.15 0.2  0.25 0.3  0.35 0.4  0.45 0.5  0.55 0.6  0.65
 0.7  0.75 0.8  0.85 0.9  0.95 1.   1.05 1.1  1.15 1.2  1.25 1.3  1.35
 1.4  1.45 1.5  1.55 1.6  1.65 1.7  1.75 1.8  1.85 1.9  1.95 2.   2.05
 2.1  2.15 2.2  2.25 2.3  2.35 2.4  2.45 2.5  2.55 2.6  2.65 2.7  2.75
 2.8  2.85 2.9  2.95 3.   3.05 3.1  3.15 3.2  3.25 3.3  3.35 3.4  3.45
 3.5  3.55 3.6  3.65 3.7  3.75 3.8  3.85 3.9  3.95 4.   4.05 4.1  4.15
 4.2  4.25 4.3  4.35 4.4  4.45 4.5  4.55 4.6  4.65 4.7  4.75 4.8  4.85
 4.9  4.95 5.   5.05 5.1  5.15 5.2  5.25 5.3  5.35 5.4  5.45 5.5  5.55
 5.6  5.65 5.7  5.75 5.8  5.85 5.9  5.95 6.   6.05 6.1  6.15 6.2  6.25
 6.3  6.35 6.4  6.45]
```

〔初期位置と初速度の設定〕

```
x0 = 0 #初期位置〔m〕
v0 = 1 #初速度〔m/s〕
```

〔運動方程式の数値解法〕　位置 $x(t)$ と, 速度 $v(t)$ を, 同時に求める

```
xvt = odeint(Boat2, [x0, v0], t) # これで解ける
```

解(**数値解** という)は, 時間軸 `t` の各時刻における(位置, 速度)の「ベクトル列」として得られる.

時刻の数列 `t`	0.05	0.1	0.15	0.2	⋯	
解の数列 `xvt`	$x(0.05)$	$x(0.1)$	$x(0.15)$	$x(0.2)$	⋯	位置
	$v(0.05)$	$v(0.1)$	$v(0.15)$	$v(0.2)$	⋯	速度

```
print(xvt) # [位置, 速度] が時刻毎に縦に並んでいる
```

```
 [1.66940225 0.16529888]
 [1.67756474 0.16121763]
 [1.68552569 0.15723715]
 [1.69329009 0.15335495]
 [1.70086279 0.14956861]
 [1.70824851 0.14587574]
 [1.71545188 0.14227406]
 [1.7224774  0.1387613 ]
 [1.72932946 0.13533527]
 [1.73601234 0.13199383]
 [1.74253022 0.12873489]
```

```
 [1.74888717 0.12555641]
 [1.75508717 0.12245641]
 [1.76113409 0.11943295]
 [1.76703171 0.11648414]
 [1.77278372 0.11360814]
 [1.77839371 0.11080315]
 [1.78386519 0.10806741]
 [1.78920158 0.10539921]
 [1.79440621 0.10279689]
 [1.79948234 0.10025883]
 [1.80443314 0.09778343]
 [1.8092617  0.09536915]
 [1.81397105 0.09301448]
 [1.81856412 0.09071794]
 [1.82304379 0.0884781 ]
 [1.82741286 0.08629357]
 [1.83167405 0.08416298]
 [1.83583003 0.08208498]
 [1.8398834  0.0800583 ]
 [1.84383669 0.07808165]
 [1.84769237 0.07615381]
 [1.85145286 0.07427357]
 [1.8551205  0.07243975]
 [1.85869759 0.07065121]
 [1.86218636 0.06890682]
 [1.86558899 0.06720551]
 [1.8689076  0.0655462 ]
 [1.87214429 0.06392786]
 [1.87530105 0.06234947]
 [1.87837988 0.06081006]
 [1.88138269 0.05930865]
 [1.88431136 0.05784432]
 [1.88716773 0.05641614]
 [1.88995356 0.05502322]
 [1.89267062 0.05366469]
 [1.89532059 0.0523397 ]
 [1.89790514 0.05104743]
 [1.90042587 0.04978707]
 [1.90288436 0.04855782]
 [1.90528215 0.04735892]
 [1.90762075 0.04618963]
 [1.9099016  0.0450492 ]
 [1.91212614 0.04393693]
 [1.91429575 0.04285213]
 [1.91641179 0.0417941 ]
 [1.91847559 0.0407622 ]
 [1.92048844 0.03975578]]
```

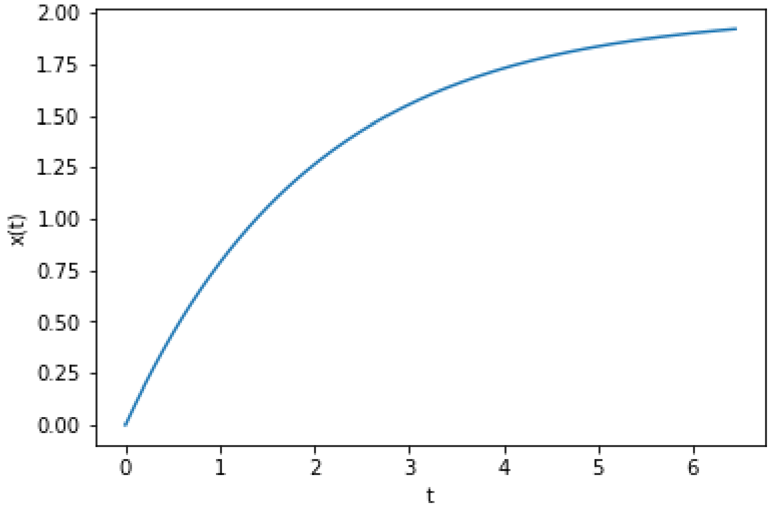
〔解のグラフ表示〕

（横軸,縦軸）=（時刻,位置）

```
xt = xvt[:,0] #最初の列（位置）を取り分ける

plt.plot(t, xt) #位置 x(t) グラフのプロット
```

```
plt.xlabel('t')  #横軸のラベル
plt.ylabel('x(t)')  #縦軸のラベル
```

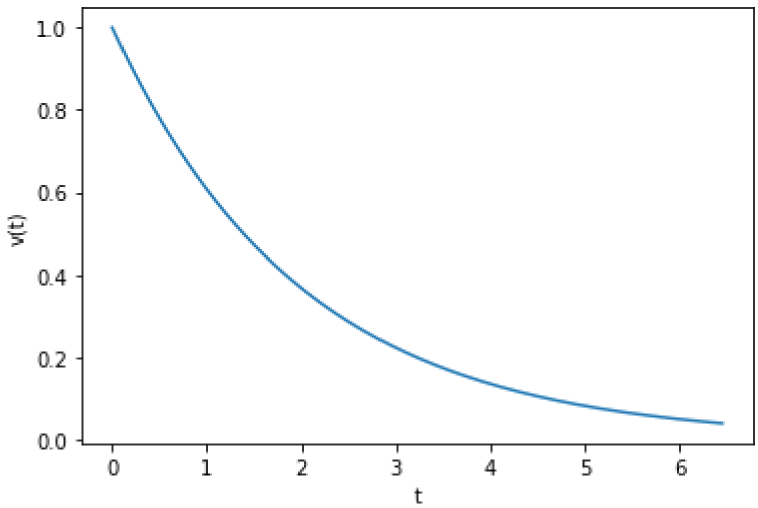
```
Text(0, 0.5, 'x(t)')
```



(横軸,縦軸)=(時刻,速度)　※先ほどと同じグラフ

```
vt = xvt[:,1] #次の列（速度）を取り分ける

plt.plot(t, vt) #速度 v(t) グラフのプロット
plt.xlabel('t')  #横軸のラベル
plt.ylabel('v(t)')  #縦軸のラベル
```

```
Text(0, 0.5, 'v(t)')
```



§3 数値解のアニメーション

パラパラ漫画方式で,解の動きをアニメーション表示する.

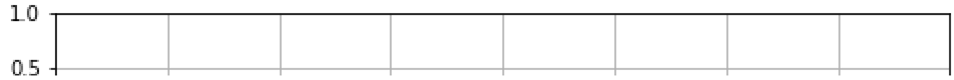
〔アニメーション用のユーザー関数〕

```
def display_Boat(motion):
  '''
```

```
  数値解をアニメーション表示する
  '''

  # グラフ用紙の設定
  # fig グラフ用紙
  # ax  座標軸
  fig, ax= plt.subplots(1, 1, figsize=(8,2)) #グラフ用紙(ax)を1行,1列(1枚)用意

  # アニメーション１コマの描画
  def each_frame(i):
    ax.cla() #グラフ用紙を白紙にリセット
    ax.set_xlim(-1,3) #x軸の範囲
    ax.set_ylim(-1,1) #y軸の範囲
    ax.grid()

    # 質点の描画 (y方向は0)
    x = motion[i] # x座標
    y = 0         # y座標 存在しないので0
    ax.plot(x, y, 'o')

  # アニメーションデータの作成
  n = len(motion) # コマ数
  anim = FuncAnimation(
      fig,
      each_frame,
      interval=80,
      frames=n
  )

  return anim
```

〔アニメーション表示〕

```
display_Boat(xt)
```



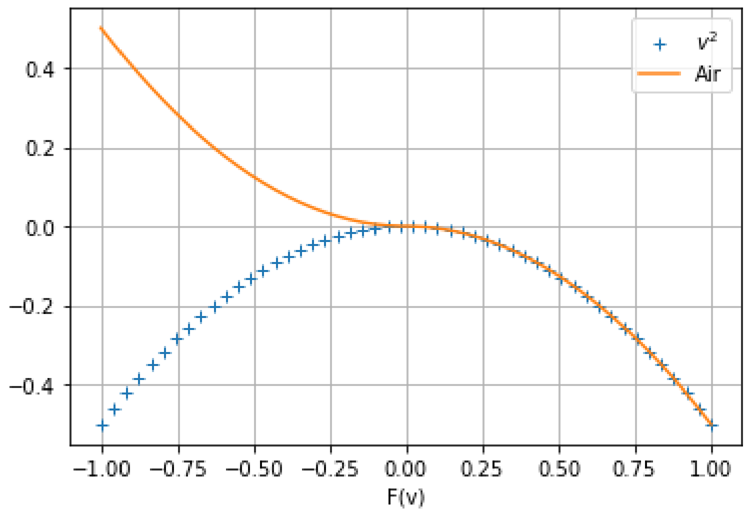
§4 大気中の質点の放物運動

$$\begin{cases} mx'' = -cv|v| \\ my'' = -cw|w| - mg \end{cases} \qquad v = x', \quad w = y'$$

〔空気抵抗のチェック〕

- $F = -cv|v|$ は，速度 v の2乗に比例する空気抵抗
- 抵抗 … 速度と逆向きの力
- 普通の2次関数 $-cv^2$ で与えてしまうと，v の正負によらず，F が同じ向きになり，抵抗にならない．

```
def plot_teikou():
  '''
  空気抵抗と2次関数の違い
  '''
  v = np.linspace(-1,1,50) # 速度軸
  c = 0.5 # 仮の抵抗係数

  Q = -c*v*v         # 普通の2次関数
  F = -c*np.abs(v)*v # 空気抵抗

  plt.plot(v, Q, '+', label='$v^2$')
  plt.plot(v, F, label='Air')
  plt.xlabel('v')
  plt.xlabel('F(v)')
  plt.grid()
  plt.legend()

plot_teikou()
```



- 空気抵抗（実線）は，2次関数のカーブでありつつ，速度と逆向きの特性になっています．

〔運動方程式のプログラミング〕

運動方程式をボートの例と同じ形式に書き直す. 2連立 ×(x,y の2方向) = 4連立.

$$\begin{cases} x' = v \\ v' = -\frac{c}{m} v|v| \\ y' = w \\ w' = -\frac{c}{m} w|w| - g \end{cases} \qquad v = x', \quad w = y'$$

```
def Shoot(xvyw, t):
  '''
  大気中の放物運動の運動方程式
  '''
  x, v, y, w = xvyw
  m = 1
  c = 0.01 #野球ボールよりだいぶ大きいかも？
  g = 9.8

  # x 方向
  dxdt = v
  dvdt = -(c/m)* v*np.abs(v)

  # y 方向
  dydt = w
  dwdt = -(c/m)* w*np.abs(w) -g

  return [dxdt, dvdt, dydt, dwdt] #4式をまとめて返す
```

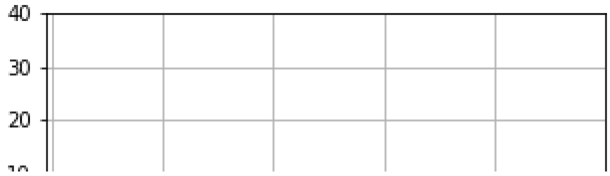
〔運動方程式の数値解法〕

```
t = np.linspace(0, 6, 120)

kmph  = 132               # km/h
mps   = kmph*1000 / 60**2 # m/s
angle = np.pi/4           # π/4 = 45度
x0 = 0                    # x方向の初期位置〔m〕
v0 = mps * np.cos(angle)  # x方向の初速度〔m/s〕
y0 = 2                    # y方向の初期位置〔m〕
w0 = mps * np.sin(angle)  # y方向の初速度〔m/s〕

xvywt = odeint(Shoot, [x0, v0, y0, w0], t) # 運動方程式を解く

xt = xvywt[:,0] # x 方向の位置〔m〕
vt = xvywt[:,1] # x 方向の速度〔m/s〕
yt = xvywt[:,2] # y 方向の位置〔m〕
wt = xvywt[:,3] # y 方向の速度〔m/s〕
```

〔放物運動のアニメーション表示〕

```
def display_Shoot(xt, yt):
  '''
  数値解をアニメーション表示する
  '''
  # グラフ用紙の設定
  # fig グラフ用紙
  # ax  座標軸
  fig, ax= plt.subplots(1, 1, figsize=(5,2)) #グラフ用紙(ax)を1行,1列(1枚)用意

  # アニメーション１コマの描画
  def each_frame(i):
    ax.cla() #グラフ用紙を白紙にリセット
    ax.set_xlim(-1,100) #x軸の範囲
    ax.set_ylim(-1,40) #y軸の範囲
    ax.grid()

    # 運動の描画
    x = xt[i] # x座標
    y = yt[i] # y座標
    ax.plot(x, y, '.r') # 質点
    ax.plot(xt[:i], yt[:i], '-r', lw=0.5) #軌跡

  # アニメーションデータの作成
  n = len(xt) # コマ数
  anim = FuncAnimation(
      fig,
      each_frame,
      interval=80,
      frames=n
  )

  return anim
```

```
display_Shoot(xt, yt)
```

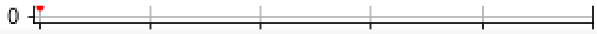


§5 運動の設計（≒条件の調整）

- 原因から結果を予測する問題を，**順問題** といいます．
- 逆に，結果を先に決めて，そうなる原因を逆算する問題を，**逆問題** といいます．
- 逆問題を解くことを **設計** といいます．（工学分野の中心的課題）

〔調整作業用のユーザ関数〕

大気中の放物運動を，例に取り上げます．



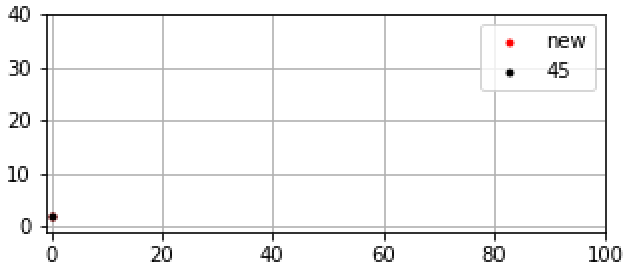

```
def design_Shoot(angle_deg):
  '''
  与えられた角度で，アニメーションまで一括処理する
  '''
  t = np.linspace(0, 7, 140)

  def shoot(deg):

    angle = deg/180 * np.pi # 度→ラジアン

    kmph  = 132              # km/h
    mps   = kmph*1000 / 60**2 # m/s
    x0 = 0                   # x方向の初期位置〔m〕
    v0 = mps * np.cos(angle)  # x方向の初速度〔m/s〕
    y0 = 2                   # y方向の初期位置〔m〕
    w0 = mps * np.sin(angle)  # y方向の初速度〔m/s〕

    xvywt = odeint(Shoot, [x0, v0, y0, w0], t) # 運動方程式を解く

    xt = xvywt[:,0] # x 方向の位置〔m〕
    yt = xvywt[:,2] # y 方向の位置〔m〕

    return [xt, yt]

  xt, yt = shoot( angle_deg )
  xt45, yt45 = shoot( 45 )    # 比較用

  # アニメーション
  fig, ax= plt.subplots(1, 1, figsize=(5,2)) #グラフ用紙(ax)を1行,1列(1枚)用意

  def each_frame(i):
    ax.cla() #グラフ用紙を白紙にリセット
    ax.set_xlim(-1,100) #x軸の範囲
    ax.set_ylim(-1,40) #y軸の範囲
```

```
    # 質点の描画
    ax.plot(xt[i], yt[i], '.r', label='new') #質点
    ax.plot(xt[:i], yt[:i], '-r', lw=0.5)    #軌跡

    # 45度の場合
    ax.plot(xt45[i], yt45[i], '.k', label='45') #質点
    ax.plot(xt45[:i], yt45[:i], ':k', lw=0.5)   #軌跡

    ax.grid()
    ax.legend()

  # アニメーションデータの作成
  n = len(xt) # コマ数
  anim = FuncAnimation(
      fig,
      each_frame,
      interval=80,
      frames=n
  )

  return anim
```

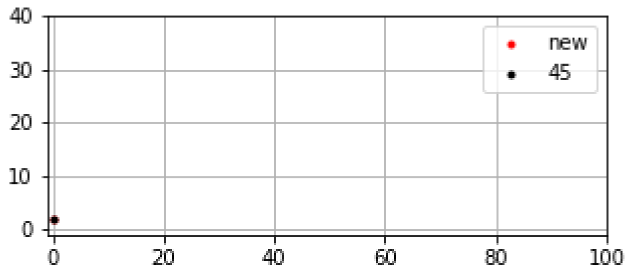
〔運動の設計〕

実習　45度と同等の飛距離で，到達時間の短い射出角度 `angle_deg` を探せ．

```
design_Shoot(angle_deg=45) # この場合は，2つの軌跡が重なります
```



Once Loop Reflect



✓ 0 秒 完了時間: 11:03