情報処理概論

九州大学　工学部地球環境工学科　講義資料　担当：木村

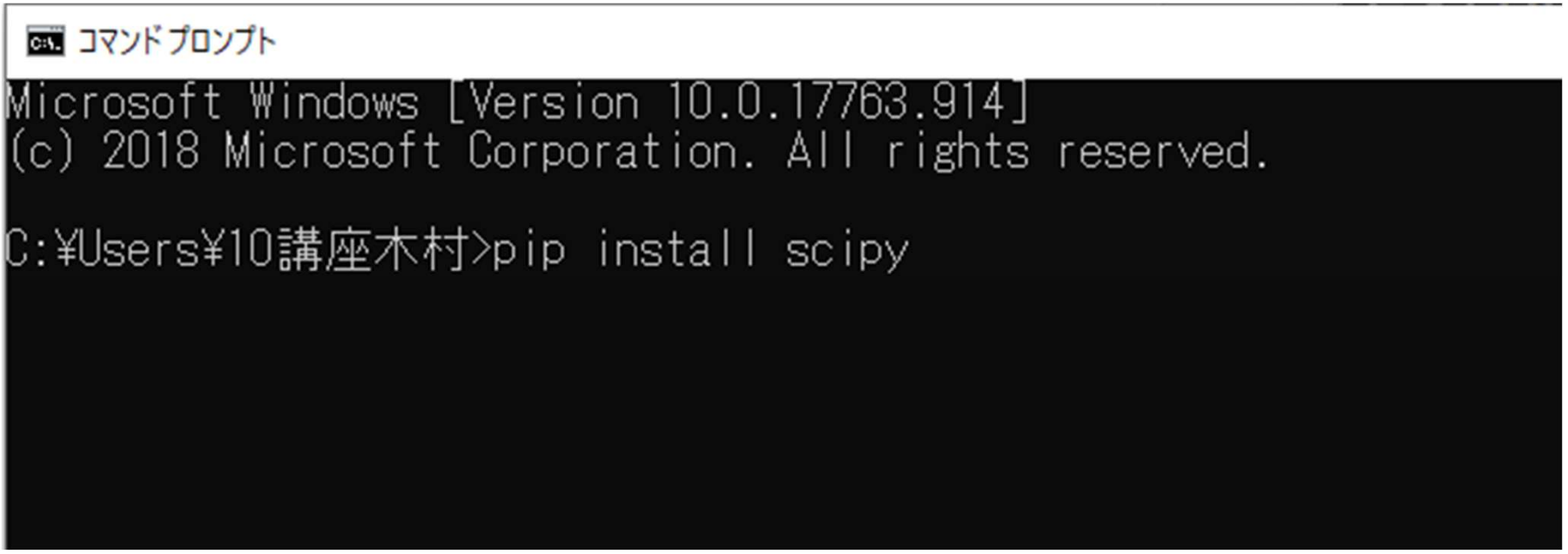
09 数値最適化／科学技術計算ライブラリ

科学技術計算モジュール SciPy

Pythonで様々な科学技術計算を行うためのパッケージ群
最適化、方程式の根、計算幾何、確率統計、補間、積分、クラスタリング、
フーリエ変換、画像処理など

科学技術計算モジュール SciPy をインストールする（前回）

Windows10の場合：コマンドプロンプト上で
＞　pip install scipy
を実行



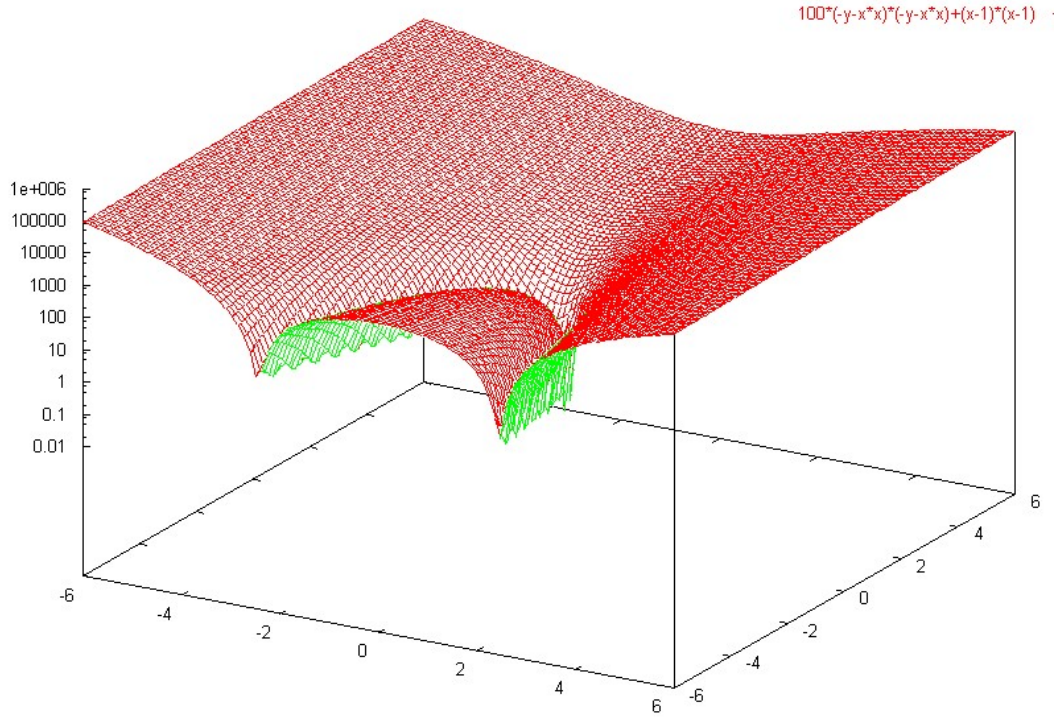
インストールがうまく行かない場合、
インターネットを検索して解決してください

ベンチマーク問題： Rosenbrock関数

コスト関数

$$f(x_1, x_2, \cdots x_n) = \sum_{i=1}^{n-1} \left\{ 100\left(x_i^2 - x_{i-1}^2\right) + (1 - x_i)^2 \right\}$$

ただし　$-2.048 < x_i < 2.048$



n=2の場合の
コスト関数の景観

極小点＝最小点は1つしか無いが、
そこへたどり着くには
長くて曲がりくねった谷を
辿らねばならない
=変数間の依存関係が強い

例1　共役勾配法 in Rosenbrock関数

```
import numpy as np          # 数値計算ライブラリ
from scipy import optimize  # scipyの最適化ライブラリ

#-----最小化を目指すコスト関数
def costFunc( x_list, *args ):  # 最適化ライブラリoptimizeで使用可能な関数形式
        #パラメータ以外のデータはargsを通して渡す
        # この例題ではRosenbrock関数を記述
        cost = 0.0;
        for n in range( 0, len( x_list )-1 ):
                cost += 100.0 * ((x_list[n]*x_list[n] - x_list[n+1])**2)
                cost += (( 1.0 - x_list[n])*( 1.0 - x_list[n]))
        return( cost )

#----パラメータの初期値
initial_x = np.array( [0.0, 0.0, 2.0, -1.0 ] )

#----Conjugate Gradient (CG法=共役勾配法)による関数最小化
x_list_cg = optimize.fmin_cg( costFunc, initial_x )

#-----最適化結果を画面へ表示
print( "conjugate gradient:", x_list_cg)
```

解きたい数値最適化問題
（パラメータx→コスト）
をここへ記述

勾配法で探索を開始する点

最適化結果のパラメータ

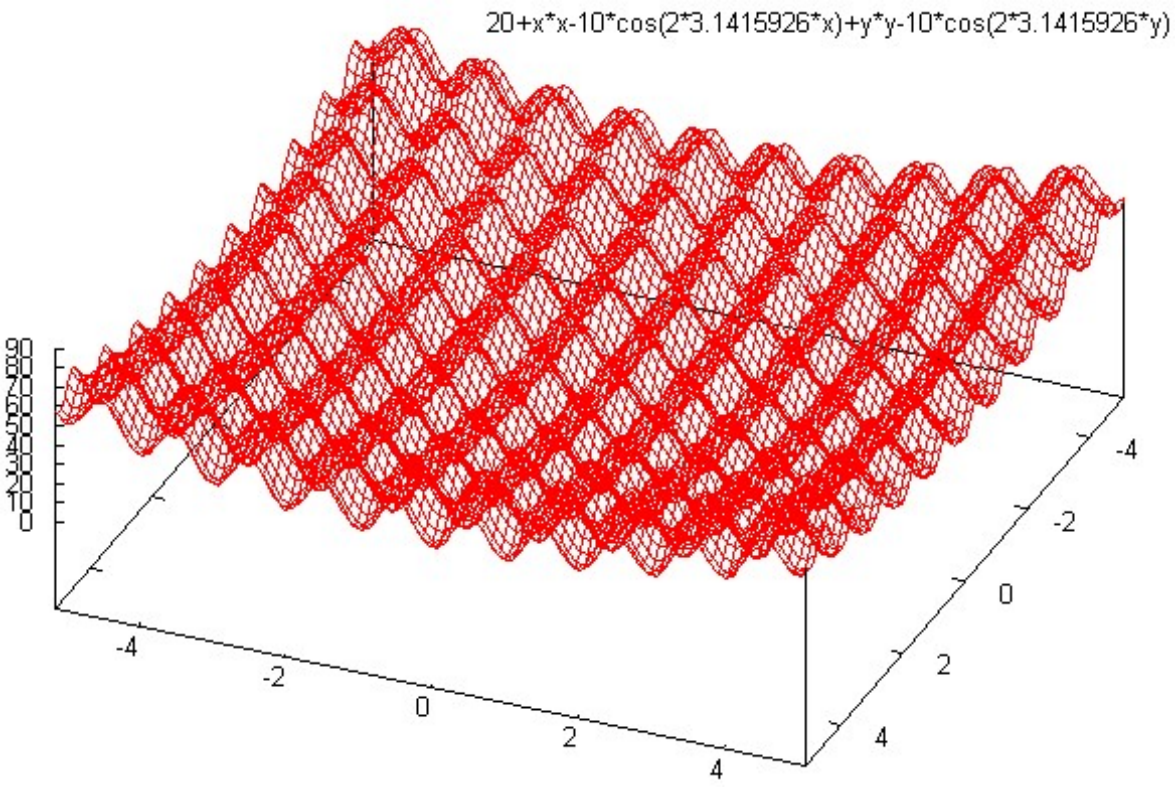
共役勾配法による最適化エンジン

ベンチマーク問題： Rastrigin関数

コスト関数

$$f(x_1, x_2, \cdots x_n) = 10n + \sum_{i=1}^{n} \left(x_i^{\,2} - 10\cos(2\pi x_i) \right)$$

ただし　　$-5.12 < x_i < 5.12$



n=2の場合の
コスト関数の景観

・局所解が多数存在する多峰性関数

・探索域中心（全て0）に最適解が存在し、その周辺に格子状に複数の局所解を持つ。

例2　共役勾配法と滑降シンプレックス法

in Rastrigin関数

```
import numpy as np          # 数値計算ライブラリ
from scipy import optimize  # scipyの最適化ライブラリ

#-----最小化を目指すコスト関数
def costFunc2( x_list, *args ):  # 最適化ライブラリoptimizeで使用可能な関数形式
        # パラメータ以外のデータはargsを通して渡す
        # この例題ではRastrigin関数を記述
        cost = 10.0 * len( x_list );
        for i in range( len( x_list ) ):
                cost += (x_list[i]*x_list[i] - 10.0 * np.cos(2.0*np.pi*x_list[i] ))
        return( cost )

#----パラメータの初期値
initial_x = np.array( [1.0, 0.5, -1.0, -0.8 ] )

#----Conjugate Gradient (CG法=共役勾配法)による関数最小化
x_list_cg = optimize.fmin_cg( costFunc2, initial_x )

#-----共役勾配法による最適化結果を画面へ表示
print( "conjugate gradient:", x_list_cg)
```

勾配法で探索を開始する点

次スライドへ続く

前スライドのプログラムの続き

```
#----次にDownHillSimplex法で最適化してみる

#--DownHillSimplex法の初期simplex(初期の解候補群）をランダムに生成
initial_simplexList=[]
for simplex in range( len( initial_x )+1 ):
        x=[]
        for dim in range( len( initial_x ) ):
                x.append( 2.0*np.random.rand()-1.0 )
        initial_simplexList.append( x )

#-----最適化
x_list_ds = optimize.fmin( costFunc2, initial_x, initial_simplex=initial_simplexList )

# x_list_ds = optimize.fmin( costFunc2, initial_x )

#------滑降シンプレックス法による最適化結果を画面へ表示
print( "DownHillSimplex:", x_list_ds)
```

問題のパラメータ個数がnのとき、n+1個のランダムな初期パラメータを用意してリストへ格納

最適化結果のパラメータ

初期パラメータは省略できるが性能が落ちるおすすめしない

滑降シンプレックス法は初期解候補群の処理に乱数を使用しているので、実行する毎に異なる結果を得る。
何度か実行して確認せよ

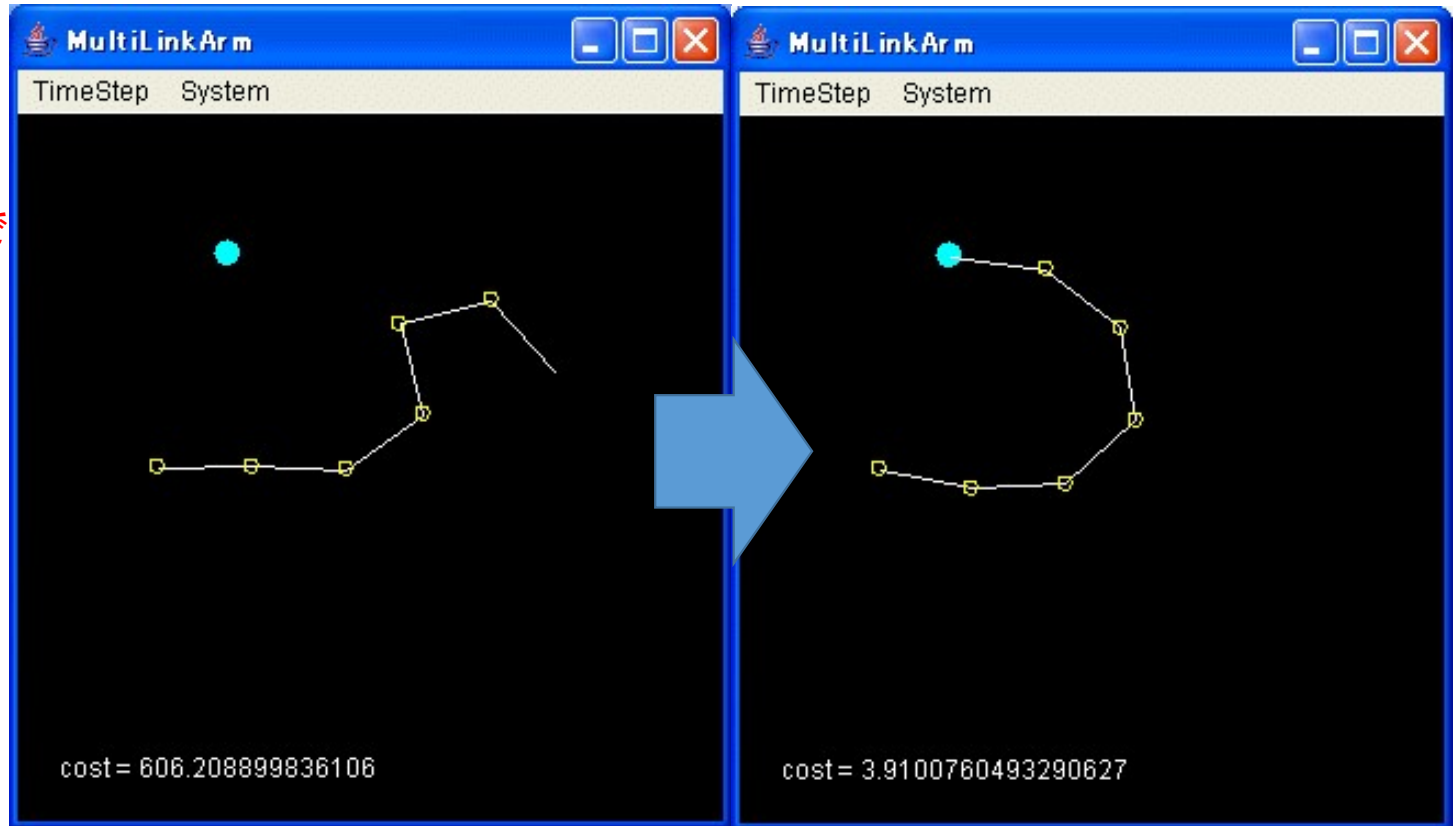
マルチリンクアームのリーチング

●各関節の角度をパラメータとして、アーム先端がターゲット座標に一致するパラメータを求める問題。
ただし、各関節はリンク同士を±90度以内で繋ぐ制約があるので、なるべく関節の可動角中央付近の角度にするのが好ましい。

コスト関数＝（アーム先端座標とターゲット座標の偏差の2乗）
＋（各関節の可動中央角度からの偏差の2乗）

アームの姿勢やコストの計算を行うための
Pythonのプログラム　Robot.py は、
講義資料の置いてあるwebサイトにあるので
これをダウンロードして使うこと。

使い方は次スライド参照



マルチリンクアームの画像表示プログラムサンプル

講義資料の置いてあるwebサイトより Robot.py をダウンロードし、
それとは別に以下のプログラムを作成して Robot.py と同じフォルダに入れて実行すること。

```
import Robot
import matplotlib     # グラフ描画ライブラリ
matplotlib.use('Agg') # グラフを画面に表示しないで画像ファイルとして保存する
import matplotlib.pyplot as plt
import numpy as np          # 数値計算ライブラリ
from scipy import optimize  # scipyの最適化ライブラリ

#--------グローバル変数の設定
#      以下の関数costFunc()内でもプログラム本体でも共通して使用可能な変数

jointAngleList0 = [-1.0, 1.0, -0.5, -0.5, -0.6] #アームの関節の初期角度1

goal_x = 2.5  #--アーム先端の目標x座標
goal_y = 5.0  #--アーム先端の目標y座標

#---マルチリンクアームクラスの実体を生成してグローバル変数化
robo = Robot.MultiLinkArm( goal_x, goal_y )
```

次スライドへ続く

前スライドのプログラムの続き

```
#-----最小化を目指すコスト関数はここに記述を追加
# def costFunc(param, *args):   # 最適化ライブラリoptimizeで使用可能な関数形式
#     関節の角度のリストと目標座標からコスト関数を計算
```

適切に記述を追加せよ
ヒント：robo.getCost()を利用する

```
#----------関節の角度のリストから、各関節の座標のx座標リストとy座標リストの入ったリストを得る
jointPositionList = robo.getJointPositionList( jointAngleList0 ) # アームの初期姿勢

#----------関節の角度のリストと目標座標からコスト関数を計算
cost = robo.getCost( jointAngleList0 )

#----figure画面figの生成
fig = plt.figure()
#----figure画面へ埋め込まれるデータプロット領域axを生成
ax = fig.add_subplot(1, 1, 1)   # fig.add_subplot(行,列,場所)を表します。
#----x軸とy軸の表示範囲を固定する
plt.xlim( -10,10 )
plt.ylim( -10,10 )
#----データプロット領域にxy軸のラベル設定
ax.set_xlabel('x axis')
ax.set_ylabel('y axis')
#----データプロット領域axにデータのグラフ描画・アームの姿勢を描く
ax.plot( jointPositionList[0], jointPositionList[1], linestyle='-', color='g', label='Initial position',marker='o')
```

X座標のリスト

Y座標のリスト

次スライドへ続く

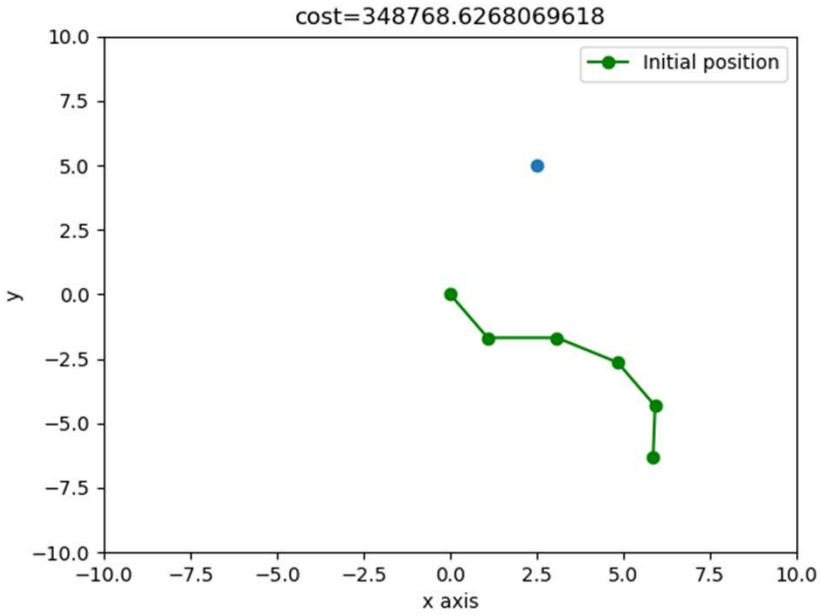
前スライドのプログラムの続き

```
#----データプロット領域axにデータの散布図描画
ax.scatter( [goal_x], [goal_y] ) # アーム先端の目標座標

#----データプロット領域axに legend と title を表示
ax.legend(loc='best')
titlestr="cost="+str(cost)
ax.set_title( titlestr )

#---描画したグラフを png 形式の画像ファイルとして保存
plt.savefig('MultiLinkArm.png') # 【注意】予め matplotlib.use('Agg') を実行しておくこと
```

実行結果



課題：マルチリンクアームのリーチングの最適化

演習資料の置いてあるwebページから Robot.py をダウンロードして作業中のフォルダへ保存後、
そのフォルダで前スライドのマルチリンクアームの画像表示プログラムサンプルを編集、
まずはその動作を確認すること。
次に上記プログラムに記述を追加し、
共役勾配法と滑降シンプレックス法でアームの関節の角度を最適化した後の姿勢を
同じ折れ線グラフの中に描画せよ。

1） まず最適化ライブラリoptimizeで使用可能な関数形式でコスト関数を追加

2） 共役勾配法によるコスト関数最小化 で得た解の関節の角度とそのときのコストを得る

3） 上記の結果を折れ線グラフに追加

4） 滑降シンプレックス法の初期シンプレックスをランダムに生成し、
コスト関数最小化 で得た解の関節の角度とそのときのコストを得る

5） さらに上記の結果を折れ線グラフに追加

余裕があれば、ロボットの関節の数を増やしてみよ。

まとめ

科学技術計算モジュール　SciPy

SciPy中で最適化を扱うモジュール　optimize

第9回　レポート課題提出方法

課題のプログラムと、プログラムの実行結果出力された画像ファイルMultiLinkArm.png　およびRobot.py を
1つのフォルダへまとめ、
下記の課題提出用フォルダへ、課題の番号と提出者が分かるようにフォルダ名を以下のようにしてフォルダごと
アップロードせよ（可能ならフォルダをzipで圧縮したファイルをアップロードするほうが好ましい）
第9回1TE19xxxZ名前
https://share.iii.kyushu-u.ac.jp/public/lRbwAAVITI5A2X4BE45t6TqQIE0UQSQUl5Bap_kZ_sjy

講義資料、および上記フォルダへのリンクは下記ホームページから
http://sysplan.nams.kyushu-u.ac.jp/gen/edu/InfoProcess/2019/index.html

プログラムがエラーで動かないとき

エラーメッセージをよく読もう
エラー箇所の行が表示されているはず（その１行前も怪しい）

文字が全角になっていないか確認すること
カッコ（）やクォーテーション””が全角になっていないか？
空白部分に全角の空白が紛れ込んでいないか？

インデントは「文字数」でそろえないとエラーになる