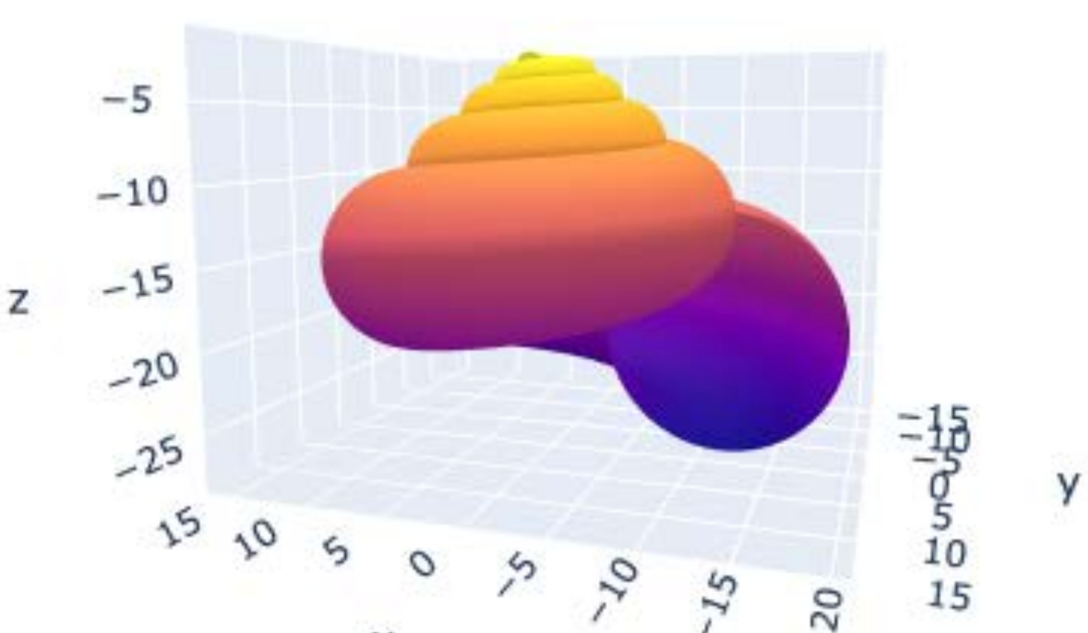
数理生物学演習

第7回　空間構造の数理モデル（1）：
理論形態学，Raupのモデル



野下 浩司（Noshita, Koji）

noshita@morphometrics.jp

https://koji.noshita.net

理学研究院 数理生物学研究室

第7回：理論形態モデル

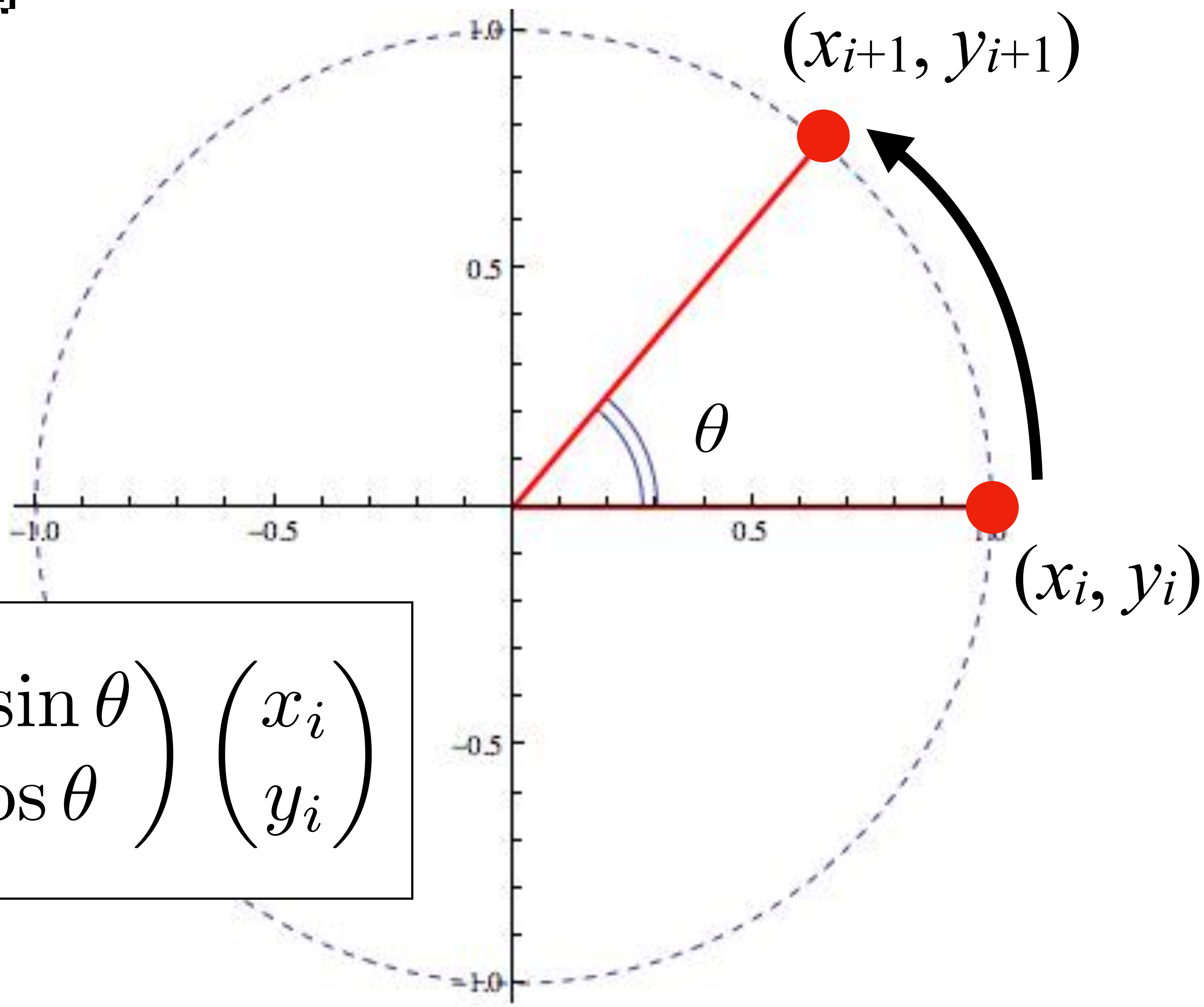
本日の目標

- Raupのモデル
- 回転行列
- 3Dプロット

回転行列 2次元

原点周りにθだけ回転させる回転行列

$$\mathbf{R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$



$$\boxed{\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}}$$

θだけ逆回転させる場合や

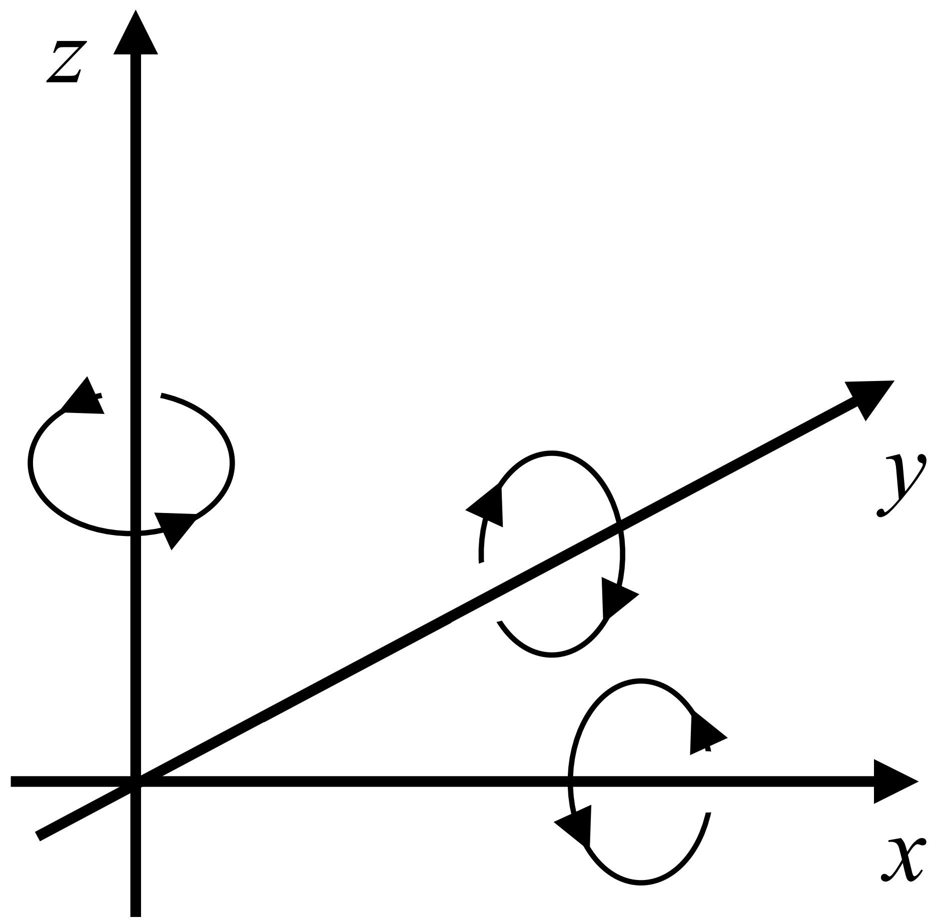
2θ回転だけ回転させる場合を考えてみよう

$$\mathbf{R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

$$\mathbf{R}(-\theta) = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

$$\begin{aligned} \mathbf{R}(2\theta) &= \begin{pmatrix} \cos 2\theta & -\sin 2\theta \\ \sin 2\theta & \cos 2\theta \end{pmatrix} \\ &= \begin{pmatrix} (\cos\theta)^2 - (\sin\theta)^2 & -2\sin\theta\cos\theta \\ 2\sin\theta\cos\theta & (\cos\theta)^2 - (\sin\theta)^2 \end{pmatrix} \\ &= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \\ &= \mathbf{R}(\theta)\mathbf{R}(\theta) \end{aligned}$$

回転行列 3次元



右ねじ

x軸周り

$$\mathbf{R}_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

y軸周り

$$\mathbf{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

z軸周り

$$\mathbf{R}_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

指数増殖モデルのおさらい

$$\frac{dx}{dt} = ax$$

初期条件

$$x(0) = x_0$$

$$x(t) = x_0 e^{at}$$

解いてみよう

対数らせん

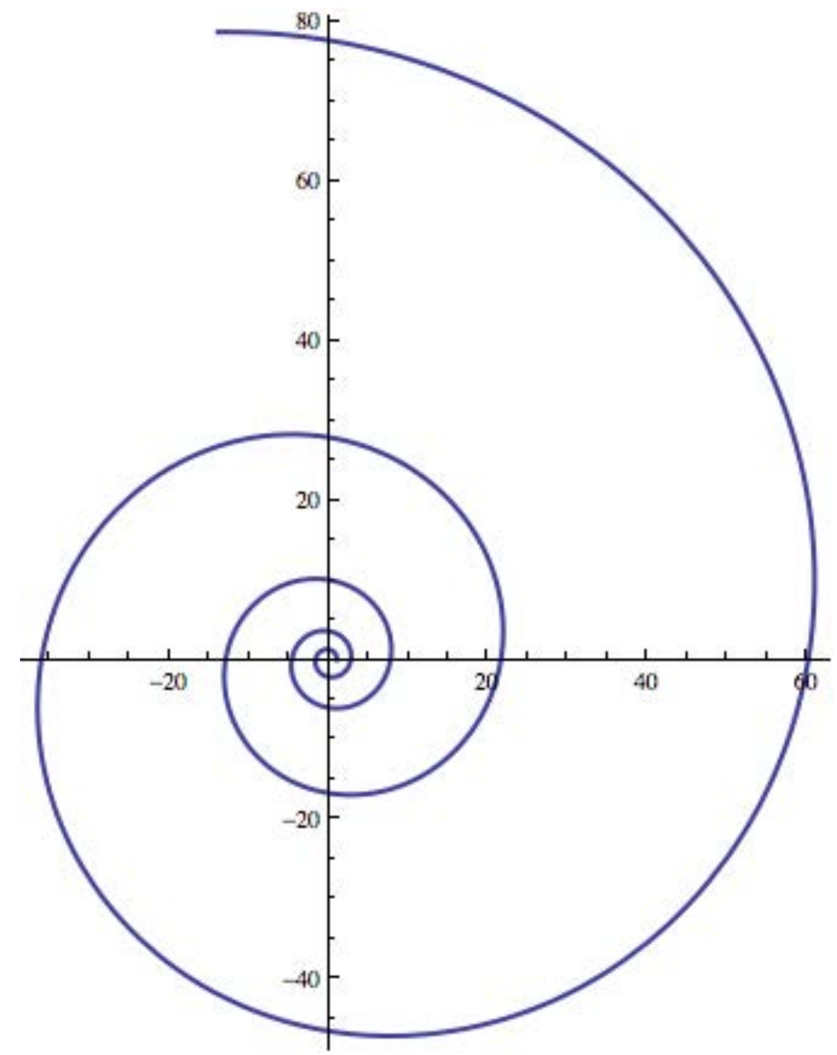
対数らせんで近似できる“巻き”パタン

$$\frac{dr}{d\theta} = a\theta \quad (a\text{は定数})$$

オウムガイ

初期条件

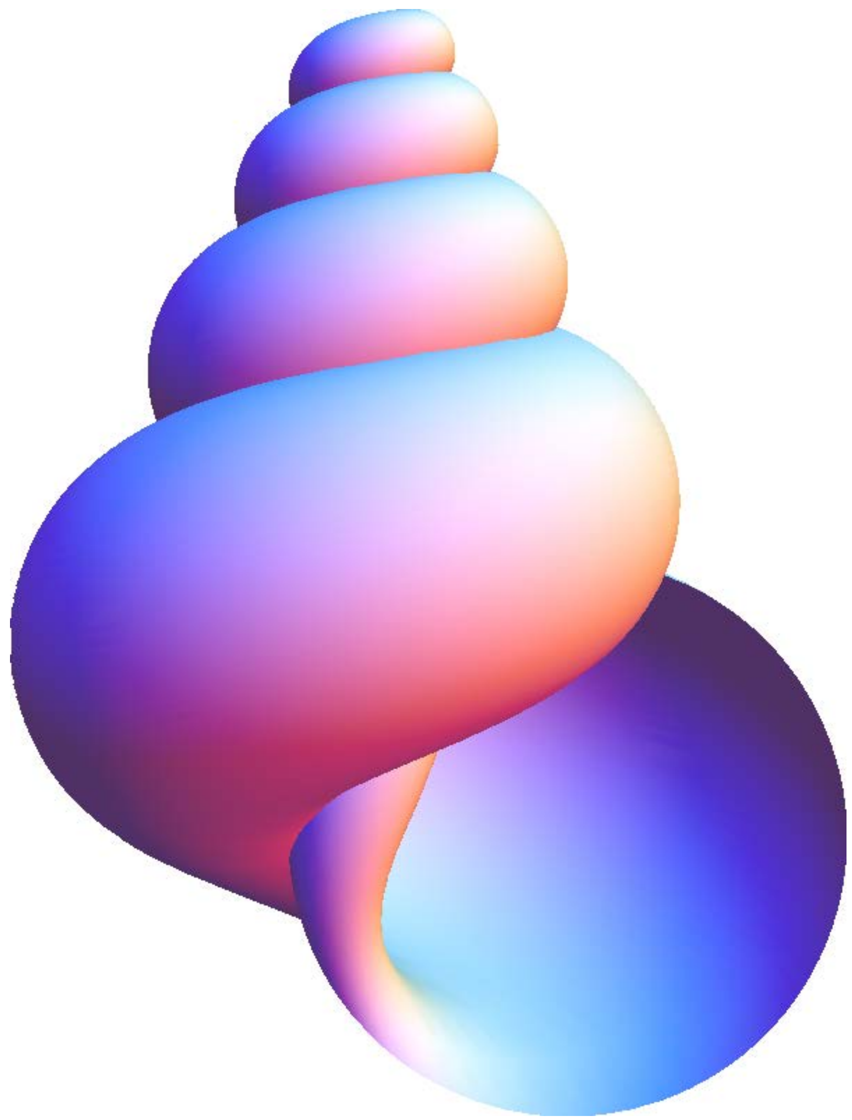
唐沢 與希 氏（三笠市立博物館）提供

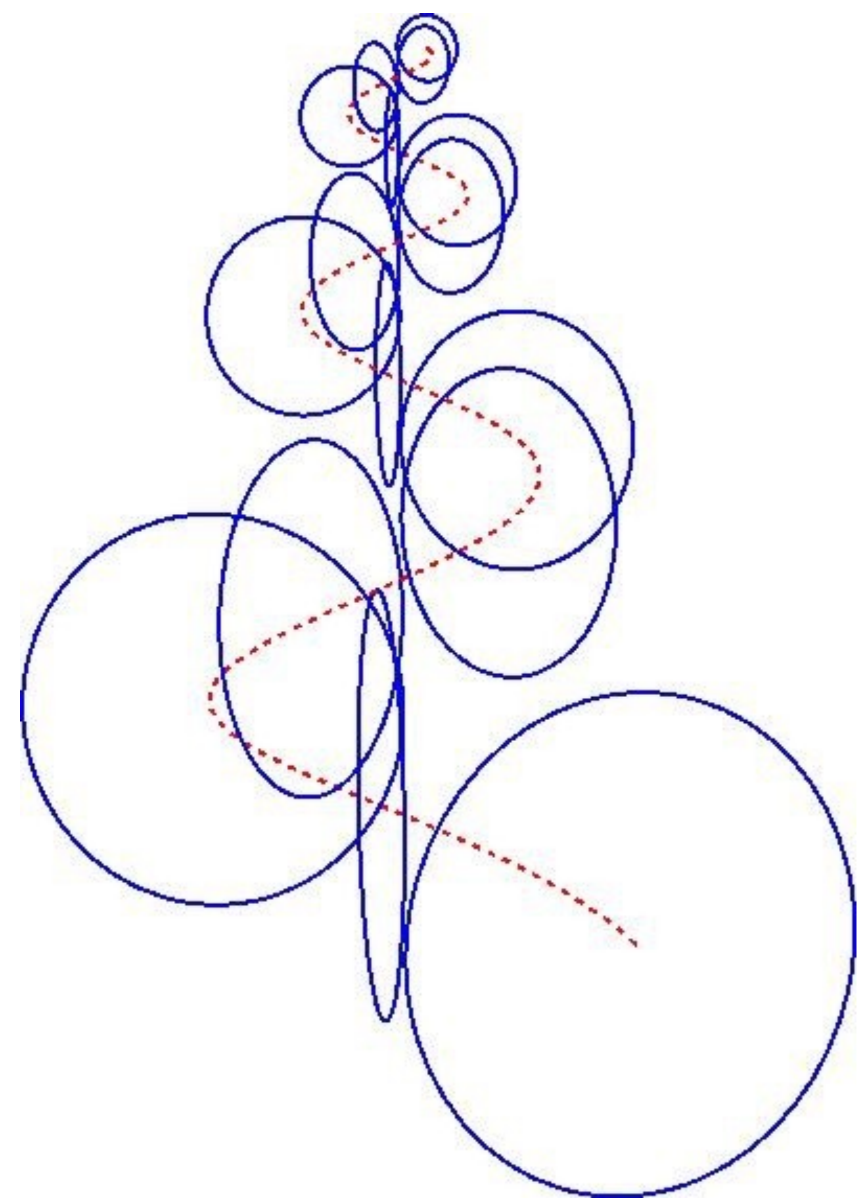


$$r(\theta) = r_0 e^{a\theta}$$

Raupのモデル

Raup (1962, 1966), Raup & Michelson (1965)





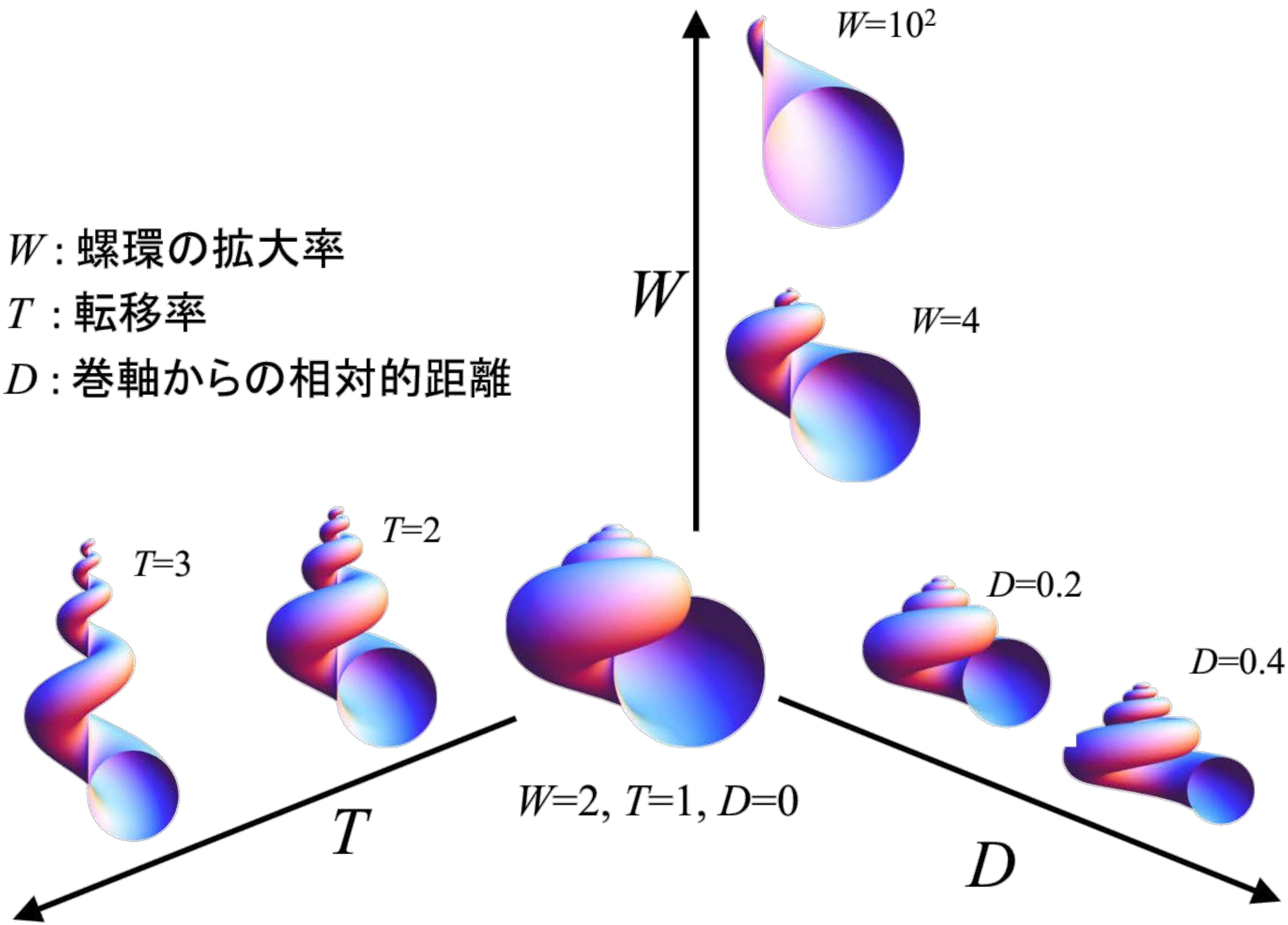
母曲線を巻軸周りに回転させながら成長させることで
“巻き”のパタンを記述

パラメータを変えることで様々な巻きパタンを表現できる

W : 螺環の拡大率

T : 転移率

D : 巻軸からの相対的距離



NumPy，NumPy配列

NumPy：数値計算・行列計算ライブラリ

NumPy

- 多次元配列を効率よく計算するためのパッケージ
- 様々な数値計算用の便利な関数も実装されている

多次元配列 ndarray

固定長の配列．要素は同じ型でなければならない．

Pythonのリストは可変長
要素の型も別々で良かった

```
# 01-01. ndarray

import numpy as np

# 7.1 ndarray
a = np.array([1,2,3])
b = np.array([6, 3.3, 1])
C = np.array([[1, 5, 6],
              [7, 8, 9],
              [4, 2, 3]])
D = np.array([[2.3, 4, 7.2],
              [7, 9, 1],
              [11, 2, 9]])
```

- import numpy as np
  NumPyを使用する際はnpという略称でインポートすることが一般的.

numpy
- array(リスト)：リストに基づき多次元配列を作成する関数．要素は同じ型でなければならない（型が異なる場合はより基本的な型へ変換（アップキャスト）される）.

NumPy：数値計算・行列計算ライブラリ

多次元配列 ndarray

固定長の配列．要素は同じ型でなければならない．

Pythonのリストは可変長
要素の型も別々で良かった

```
# 01-02. ndarrayの属性

# 配列の形状
print(a.shape)
print(C.shape)

# 次元
print(b.ndim)
print(D.ndim)
```

```
# （要素の）型
print(a.dtype)
print(D.dtype)

# 配列のキャスト
e = a.astype(float)
F = D.astype(int)
print(e)
print(F)
```

```
#出力
(3,)
(3, 3)
1
2
int64
float64
[1. 2. 3.]
[[ 2  4  7]
 [ 7  9  1]
 [11  2  9]]
```

numpy
- 配列.shape 配列の形状（高さ，幅，深さ，…など）を記録したタプル
- 配列.ndim 配列の次元
- 配列.dtype 配列の型（要素にどの型を持つか）
- 配列.astype 配列のキャスト．特定の型へ変換できる．

NumPy：数値計算・行列計算ライブラリ

基本的な演算（1）

```
# 01-03. 基本的な演算

# 同次元の加算・減算
print("a + b: ", a + b)
print("b - a: ",b - a)
print("C + D: \n", C + D)
print("C - F: \n",C - F)

# 異なる次元の加算・減算
print("a + C: \n", a + C)
print("D - b: \n", D - b)

# 乗算・除算
print("a*b: ", a * b)
print("C/a: \n", C / a)
```

！注意：ベクトルや行列の演算とは別物．これらは後ほど．

次元が異なる場合は一番大きな次元に合わせ，同一要素が繰り返される．

要素ごとの演算

a+Cは
の出力結果は
array([a+C[0],
a+C[1],
a+C[2]])
となるイメージ

```
#出力
a + b:  [7.  5.3 4. ]
b - a:  [ 5.   1.3 -2. ]
C + D:
 [[ 3.3  9.  13.2]
 [14.  17.  10. ]
 [15.   4.  12. ]]
C - F:
 [[-1  1 -1]
 [ 0 -1  8]
 [-7  0 -6]]
a + C:
 [[ 2  7  9]
 [ 8 10 12]
 [ 5  4  6]]
D - b:
 [[-3.7  0.7  6.2]
 [ 1.   5.7  0. ]
 [ 5.  -1.3  8. ]]
a*b:  [6.  6.6 3. ]
C/a:
 [[1.  2.5 2. ]
 [7.  4.  3. ]
 [4.  1.  1. ]]
```

このあたりの細かいルールを知りたい場合は公式ドキュメントを参照．

https://docs.scipy.org/doc/numpy/reference/ufuncs.html#broadcasting

NumPy：数値計算・行列計算ライブラリ

基本的な演算（2）

NumPyの関数は基本的に配列の要素ごとに適用される．

```
# 01-04. 基本的な関数による演算

# 指数
print("a**2: ", a**2)
print("np.exp(2): ", np.exp(2))
print("np.exp(a): ", np.exp(a))

# 対数
print("np.log(2): ", np.log(2))
print("np.log(C): \n", np.log(C))

# 平方根
print("np.sqrt(2): ", np.sqrt(2))
print("np.sqrt(b): ", np.sqrt(b))

# 三角関数
print("np.sin(np.pi/2): ", np.sin(np.pi/2))
print("np.sin(D): \n", np.sin(D))
print("np.cos(e): ", np.cos(e))
```

同じ関数で（複数の要素を）処理したいときに便利な機能．こうした処理をベクトル化した（vectrized）計算と呼ぶことがある．

```
#出力
a**2:  [1 4 9]
np.exp(2):  7.38905609893065
np.exp(a):  [ 2.71828183  7.3890561  20.08553692]
np.log(2):  0.6931471805599453
np.log(C):
 [[0.         1.60943791 1.79175947]
 [1.94591015 2.07944154 2.19722458]
 [1.38629436 0.69314718 1.09861229]]
np.sqrt(2):  1.4142135623730951
np.sqrt(b):  [2.44948974 1.81659021 1.        ]
np.sin(np.pi/2):  1.0
np.sin(D):
 [[ 0.74570521 -0.7568025   0.79366786]
 [ 0.6569866   0.41211849  0.84147098]
 [-0.99999021  0.90929743  0.41211849]]
np.cos(e):  [ 0.54030231 -0.41614684 -0.9899925 ]
```

NumPy：数値計算・行列計算ライブラリ

ベクトル・行列計算（1）

配列をベクトルや行列あるいはテンソルとみなして計算をおこなうこともできる

```
# 01-05. ベクトル・行列計算


# ベクトルの基本演算

print("a+b: ", a + b)
print("a-b: ", a - b)
print("3*a: ",3 * a)

# ベクトルの内積・外積

print("a.b, np.dot(a,b): ", np.dot(a,b))
print("axb, np.cross(a,b): ", np.cross(a,b))

# 行列の基本演算

print("C+D: \n", C + D)
print("C-D: \n", C - D)
print("2*C: \n", 2 * C)

# 行列の乗算

print("C.a, np.dot(C,a): ", np.dot(C,a))
print("C.D, np.dot(C,D): \n", np.dot(C,D))
print("D.C, np.dot(D,C): \n", np.dot(D,C))
```

```
#出力
a+b:  [7.  5.3 4. ]
a-b:  [-5.  -1.3  2. ]
3*a:  [3 6 9]
a.b, np.dot(a,b):  15.6
axb, np.cross(a,b):  [-7.9 17.  -8.7]
C+D:
 [[ 3.3  9.   13.2]
 [14.   17.   10. ]
 [15.    4.   12. ]]
C-D:
 [[-1.3  1.   -1.2]
 [ 0.   -1.    8. ]
 [-7.    0.   -6. ]]
2*C:
 [[ 2 10 12]
 [14 16 18]
 [ 8  4  6]]
C.a, np.dot(C,a):  [29 50 17]
C.D, np.dot(C,D):
 [[103.3  61.   66.2]
 [171.1 118.  139.4]
 [ 56.2  40.   57.8]]
D.C, np.dot(D,C):
 [[ 59.1  57.9  71.4]
 [ 74.  109.  126. ]
 [ 61.   89.  111. ]]
```

NumPy：数値計算・行列計算ライブラリ

ベクトル・行列計算（2） 線形代数向けの便利な関数も用意されている

```
# 01-06. 線形代数向け関数
# 転置行列
print("C^T, C.transpose(): ", C.transpose())
print("C^T, np.transpose(C): ", np.transpose(C))
# 行列式
print("|D|, np.linalg.det(D): ", np.linalg.det(D))
# 逆行列
print("F^-1, np.linalg.inv(F): ”, np.linalg.inv(F))
# 固有値・固有ベクトル
print(“np.linalg.eig(C): ", np.linalg.eig(C))
print("固有値のみ, np.linalg.eigvals(C): ”, np.linalg.eigvals(C))
```

```
#出力
C^T, C.transpose():  [[1 7 4]
 [5 8 2]
 [6 9 3]]
C^T, np.transpose(C):  [[1 7 4]
 [5 8 2]
 [6 9 3]]
|D|, np.linalg.det(D):  -638.3000000000005
F^-1, np.linalg.inv(F) [[-0.12248062  0.03410853  0.09147287]
 [ 0.08062016  0.09147287 -0.07286822]
 [ 0.13178295 -0.0620155   0.01550388]]
np.linalg.eig(F) (array([14.72735221, -3.28537742,  0.55802521]), array([[ 0.43801562,  0.85468529, -0.00703173],
       [ 0.84944136, -0.12913467, -0.76794748],
       [ 0.29426465, -0.50282928,  0.64047421]]))
固有値のみ, np.linalg.eigvals(F) [14.72735221 -3.28537742  0.55802521]
```

NumPy：数値計算・行列計算ライブラリ

その他の関数

その他にも配列関係の計算を便利におこなうための関数が多数用意されている.

numpy
- zeros(shape)：形状がshapeのすべての要素がゼロの配列を生成する
- identity(n)：n x nの単位行列を生成する
- linspace(start, stop, num)：startからstopまでの間にnum個の値をもつ配列を生成する（stopを含む）.

```
# 01-07. その他の便利な関数

# 配列の生成
Z = np.zeros([3,4])
I = np.identity(3)
r = np.linspace(1, 2, 10)
print("Z: \n", Z)
print("I: \n", I)
print("r: ", r)

# 集約・統計
print("np.max(a)", np.max(a), a)
print("a.max()", a.max(), a)
print("np.min(C)", np.min(C), C)
print("C.min()", C.min(), C)
print("np.sum(b): ", np.sum(b), b)
print("b.sum(): ", b.sum(), b)
print("np.mean(b): ", np.mean(b))
print("b.mean(): ", b.mean(), b)
print("np.median(b): ", np.median(b))
print("np.std(D): ", np.std(D))
```

```
#出力
Z:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
I:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
r:  [1.         1.11111111 1.22222222 1.33333333 1.44444444
1.55555556
 1.66666667 1.77777778 1.88888889 2.        ]
np.max(a) 3 [1 2 3]
a.max() 3 [1 2 3]
np.min(C) 1 [[1 5 6]
 [7 8 9]
 [4 2 3]]
C.min() 1 [[1 5 6]
 [7 8 9]
 [4 2 3]]
np.sum(b):  10.3 [6.  3.3 1. ]
b.sum():  10.3 [6.  3.3 1. ]
np.mean(b):  3.4333333333333336
b.mean():  3.4333333333333336 [6.  3.3 1. ]
np.median(b):  3.3
np.std(D):  3.3973846149975753
```

関数を鍛える：docstring

関数定義へのdocstringの追加

```
def 関数名(パラメータ):
   """
   文章（docstring）
   """
   処理1
   処理2
   ...
   処理n
   return 戻り値
```

Jupyter Notebook上でもdocstringを呼び出すことができる.

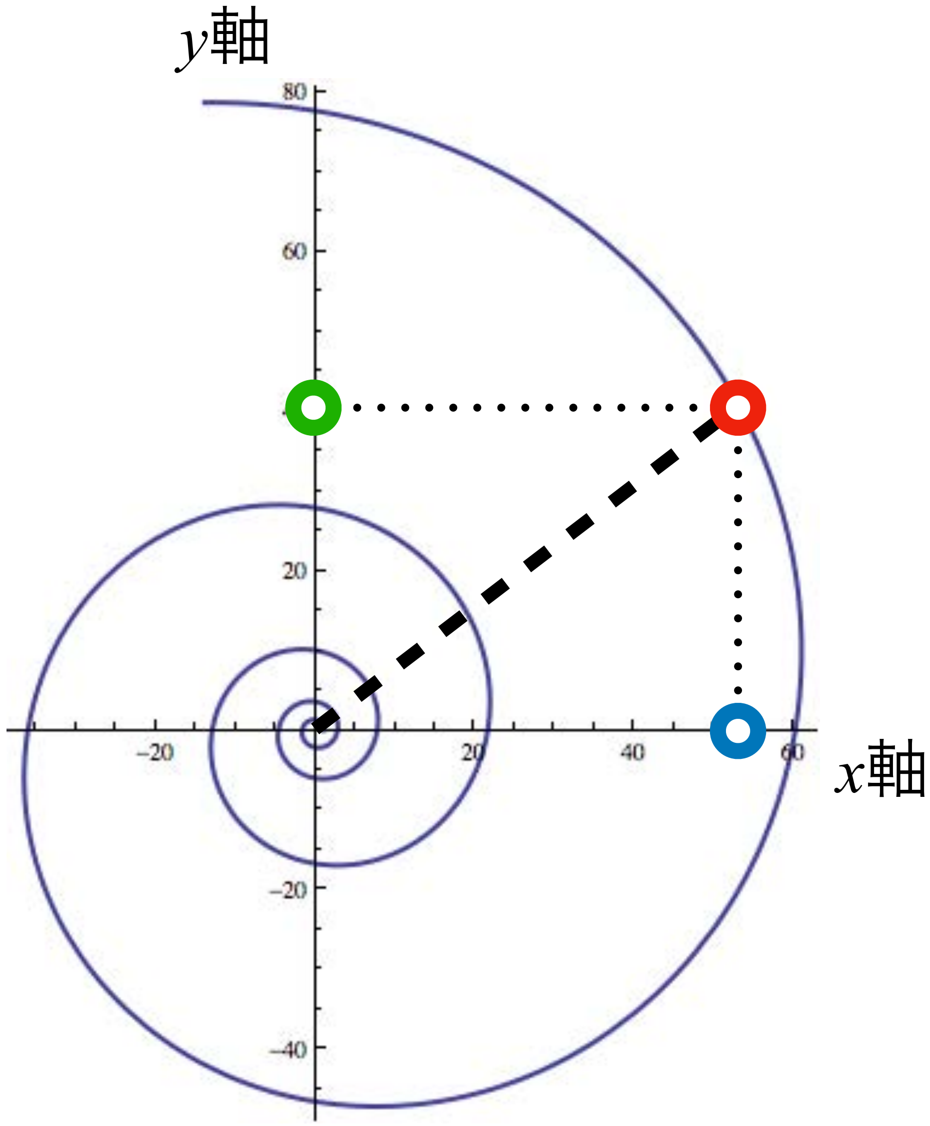
- Shift+Tab

- ?関数名

- docstring：関数などの説明文．宣言後すぐに書き込む．
  NumPyスタイルやGoogleスタイルが有名．
  http://www.sphinx-doc.org/ja/stable/ext/napoleon.html

対数らせんと可視化

対数らせん



$$r(\theta) = r_0 e^{a\theta}$$

```
# 02-01. 対数らせん
import numpy as np

def logSpiral(a, r0, theta):
    """対数螺旋

    対数螺旋の座標値を返す関数

    Args:
        a: 対数螺旋の拡大率
        r0: 動径の初期値
        theta: 回転角

    Returns:
        x, y: 対数螺旋上の座標値

    """
    r = r0*np.exp(a*theta)
    x = r*np.cos(theta)
    y = r*np.sin(theta)

    return (x,y)
```

対数らせんのプロット

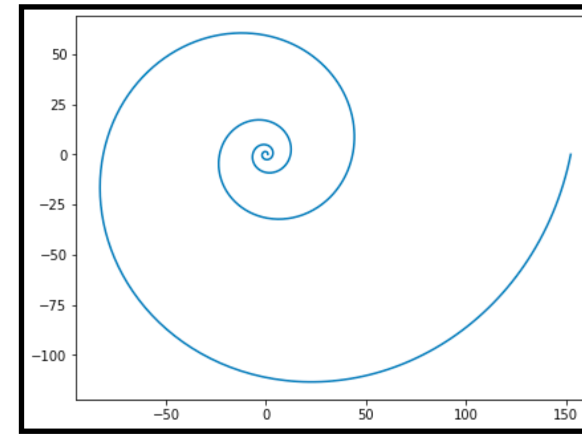
```
# 02-02. 対数螺旋のプロット
import matplotlib.pyplot as plt

# パラメータの設定
r0 = 1
a = 0.2

theta = np.linspace(0, 8*np.pi,1000)

# 座標値の計算
x, y = logSpiral(a, r0, theta)

# プロット
plt.figure(dpi = 200)
plt.axes().set_aspect('equal')
plt.plot(x,y)
```

回転角0〜8πまでプロット

（さっき定義した）logSpiral関数を利用した計算．x,yにはそれぞれ座標値を記録したnumpy配列が代入される

x軸とy軸のスケールを同じにする

出力例



パラメータを変えてプロットしてみよう！

- numpy.linspace(start, stop, num): startからstopまでを等間隔に区切るnum個の要素をもつ配列を作る（num-1分割する）
- matplotlib.pyplot
  - axes() figure環境にaxes（座標軸，様々な作図関連の要素を格納する入れ物）を追加する．
  - axes.Axes
    - set_aspect() axesのアスペクト比（縦/横）を設定する．自動（'auto'），同じ（'equal'），あるいは具体的な値を指定できる．

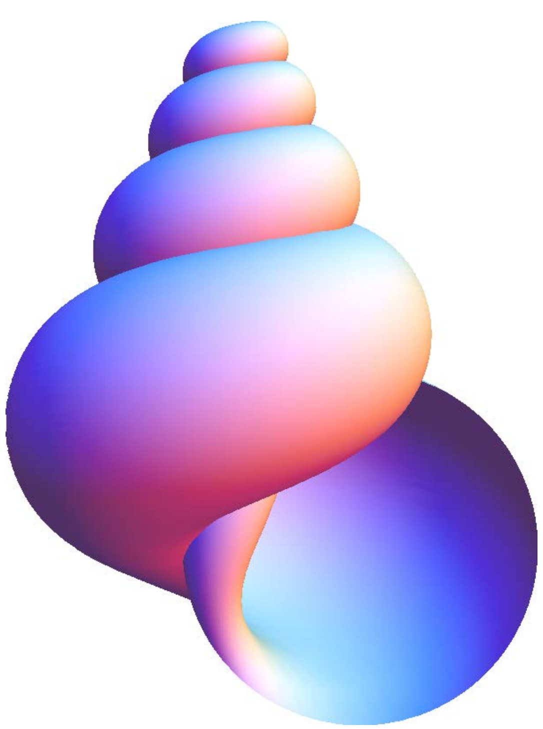
Raupのモデルと可視化

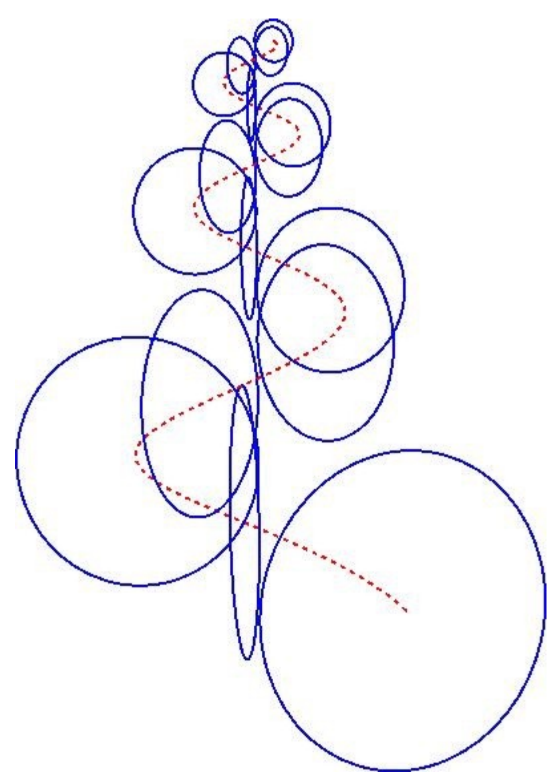
RaupのモデルRaup

θ, φでパラメータ表示された母曲線の軌跡（曲面）で巻貝の殻形態を近似する

$$\mathbf{r}(\theta,\phi|W,T,D) = \underbrace{W^{\frac{\theta}{2\pi}}}_{\text{管の拡大}} \underbrace{\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{z軸周りの回転}} \left[\underbrace{\begin{pmatrix} \cos\phi \\ 0 \\ \sin\phi \end{pmatrix}}_{\text{母曲線}} + \underbrace{\begin{pmatrix} \frac{2D}{1-D}+1 \\ 0 \\ 2T\left(\frac{D}{1-D}+1\right) \end{pmatrix}}_{\text{初期位置}} \right]$$

ただし, $W > 1, T \in \mathbf{R}, -1 < D < 1$　とする.





計算すると

$$= \begin{pmatrix} W^{\frac{\theta}{2\pi}}\left(\frac{2D}{1-D}+1+\cos\phi\right)\cos\theta \\ W^{\frac{\theta}{2\pi}}\left(\frac{2D}{1-D}+1+\cos\phi\right)\sin\theta \\ W^{\frac{\theta}{2\pi}}\left(2T\left(\frac{D}{1-D}+1\right)+\sin\phi\right) \end{pmatrix}$$

これに基づきプロットする

Raupモデルの定義

```python
# 02-03. Raupのモデル
def raup_model(W, T, D, theta, phi):
    """Raupのモデル

    Raupのモデルに基づき殻表面の座標（x, y, z）を計算する.

    Args:
        W: 螺層拡大率
        T: 転移率（殻の高さ）
        D: 巻軸からの相対的距離（臍の大きさ）
        theta: 成長に伴う回転角
        phi: 殻口に沿った回転角

    Returns:
        x, y, z: 殻表面のx座標, y座標, z座標の
        それぞれの座標値（の配列）

    """
    w = W**(theta/(2*np.pi))
    x = w * (2*D/(1 - D) + 1 + np.cos(phi))*np.cos(theta)
    y = - w * (2*D/(1 - D) + 1 + np.cos(phi))*np.sin(theta)
    z = - w * (2*T*(D/(1 - D) + 1) + np.sin(phi))
    return (x, y, z)
```

$$\mathbf{R}_x(\pi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

プロット時に見やすくする
ためにx軸周りで180°回転
させている

Raupモデルのプロット

```
# 02-04. Raupのモデルのプロット
import plotly.graph_objs as go

# Raupモデルに基づく殻表面座標の計算
W = 10**0.2
T = 1
D = 0.2

theta_range = np.linspace(0,8*np.pi, 800 )
phi_range= np.linspace(0, 2*np.pi, 60)
theta, phi = np.meshgrid(theta_range, phi_range)

x, y, z = raup_model(W, T, D, theta, phi)

#プロット
fig = go.Figure(
    go.Surface(x=x, y=y, z=z, showscale=False)
    )
fig.update_layout(
    scene = {
        "aspectmode": "data"
        })
fig.show()
```

3次元プロットのためにプロット用パッケージ（plotly）のgraph_objsをgoという略称でインポート

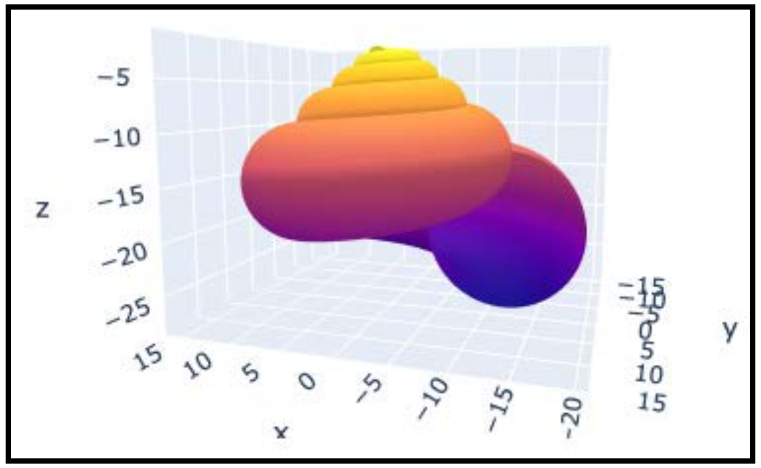
より詳しく知りたい人は公式ドキュメント参照
Plotly https://plotly.com/python/

numpy
- meshgrid(array1d_1, array1d_2)
  一次元配列array1d_1とarray1d_2に従い，それらのなす格子点の（座標ごとの）配列のリストを生成する．

3次元プロットのための
殻両面の３次元座標値を計算

入力した点に張られる表面を作成

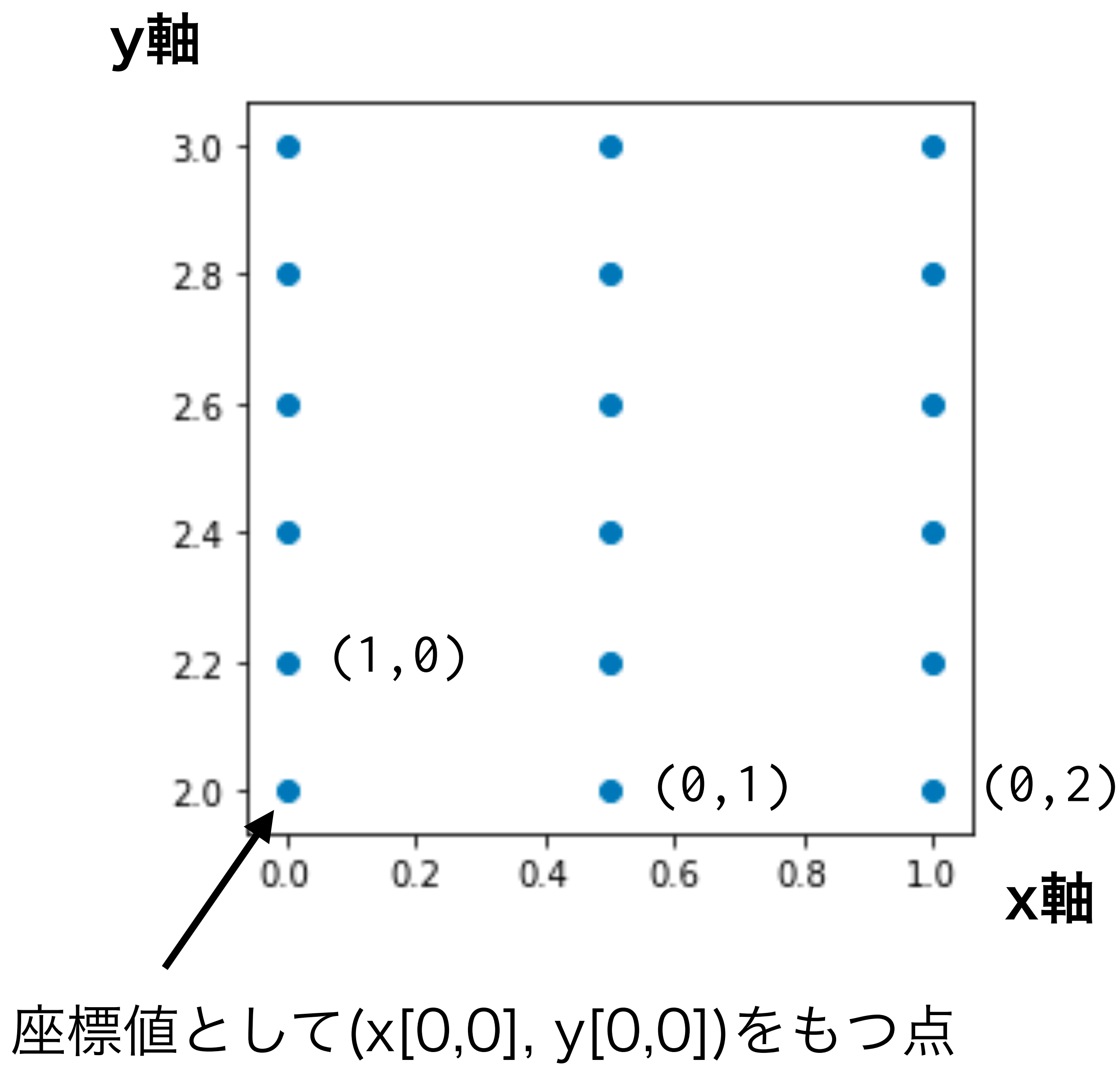
プロット時に各軸の
スケールを揃えるため



meshgridの補足

```
# meshgrid
import matplotlib.pyplot as plt
import numpy as np

a = np.linspace(0,1,3)
b = np.linspace(2,3,6)
mesh = np.meshgrid(a,b)
x, y = np.meshgrid(a,b)

print(mesh)

plt.axes().set_aspect("equal")
plt.scatter(x,y)
```

```
#出力
[array([[0. , 0.5, 1. ],
       [0. , 0.5, 1. ],
       [0. , 0.5, 1. ],
       [0. , 0.5, 1. ],
       [0. , 0.5, 1. ],
       [0. , 0.5, 1. ]]),
array([[2. , 2. , 2. ],
       [2.2, 2.2, 2.2],
       [2.4, 2.4, 2.4],
       [2.6, 2.6, 2.6],
       [2.8, 2.8, 2.8],
       [3. , 3. , 3. ]])]
```



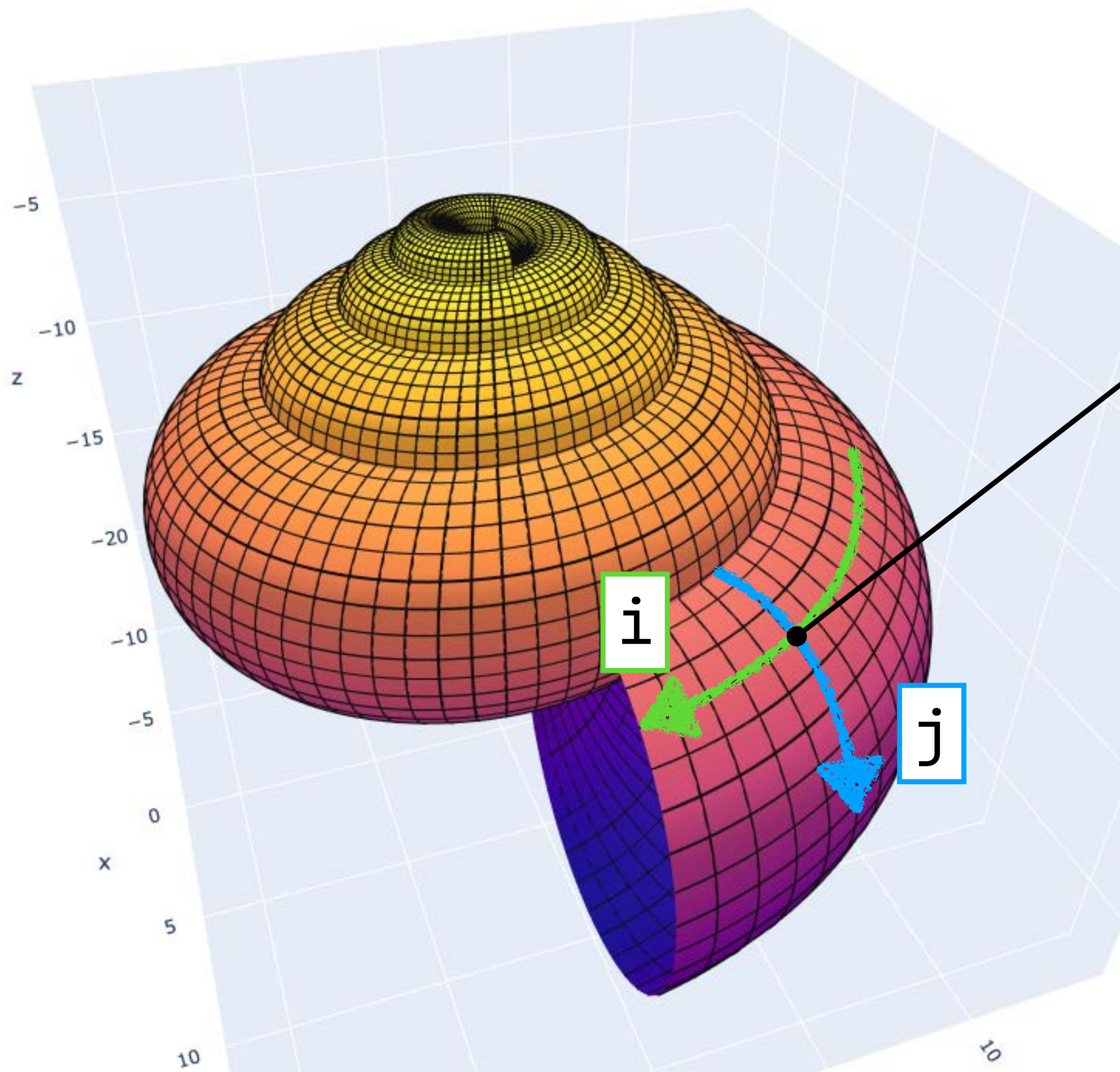
- plt.scatter(X, Y)：配列（やリスト）XとYを座標値とした散布図を描く

n（>2）個の配列からなる格子点の生成にも利用可能

Surfaceによるプロットとの組合せ

```
plotly.graph_objects
```

- `Surface(x=x_arr, y=y_arr, z=z_arr)`
  x，y，zに2次元配列を渡すことで，そのインデックスに対するパラメトリック曲面をプロットする



(x[i, j], y[i, j], z[i, j])

今回はRaupモデルのθとϕに相当するiとjのインデックスに対してパラメトリックな曲面をプロットしている.

本日の課題 ノーマル

1. Raupモデルのパラメータを変化させて，様々な“かたち”を描け（4つ程度）
選択 2. 1.で描いた“かたち”を巻貝の形態的なモデルとしよう．すると様々なかたちの中には現実の巻貝に存在する“かたち”と，現実には存在しない“かたち”が現れる．では何故現実にはそうした“かたち”の巻貝が存在するのか，または存在しないのかを究極要因と至近要因の両面から考察し，意見を述べよ．
選択 3. 現実の巻貝にはRaupモデルによって描けない“かたち”が存在する．そうした，巻貝を探しだし，何故Raupモデルでは描けないのかを考察せよ．
4. 質問，意見，要望等をどうぞ．

課題をノートブック（.ipynbファイル）にまとめて，Moodleにて提出すること

ファイル名は[回数，01~15]_[難易度，ノーマル nかハード h].ipynb．例．07_n.ipynb

本日の課題 ハード

ノーマル課題2, 3のうち，ノーマルで選択しなかった方に取り組む.

2. （ノーマルの）1.で描いた“かたち”を巻貝の形態的なモデルとしよう．すると様々なかたちの中には現実の巻貝に存在する“かたち”と，現実には存在しない“かたち”が現れる．では何故現実にはそうした“かたち”の巻貝が存在するのか，または存在しないのかを究極要因と至近要因の両面から考察し，意見を述べよ.
3. 現実の巻貝にはRaupモデルによって描けない“かたち”が存在する．そうした，巻貝を探しだし，何故Raupモデルでは描けないのかを考察せよ.

課題をノートブック（.ipynbファイル）にまとめて，Moodleにて提出すること

ファイル名は[回数，01~15]_[難易度，ノーマル nかハード h].ipynb．例．07_nh.ipynb

次回予告
第8回：研究を始めるために
6月10日

復習推奨

- 引用の方法
- 論文の探し方