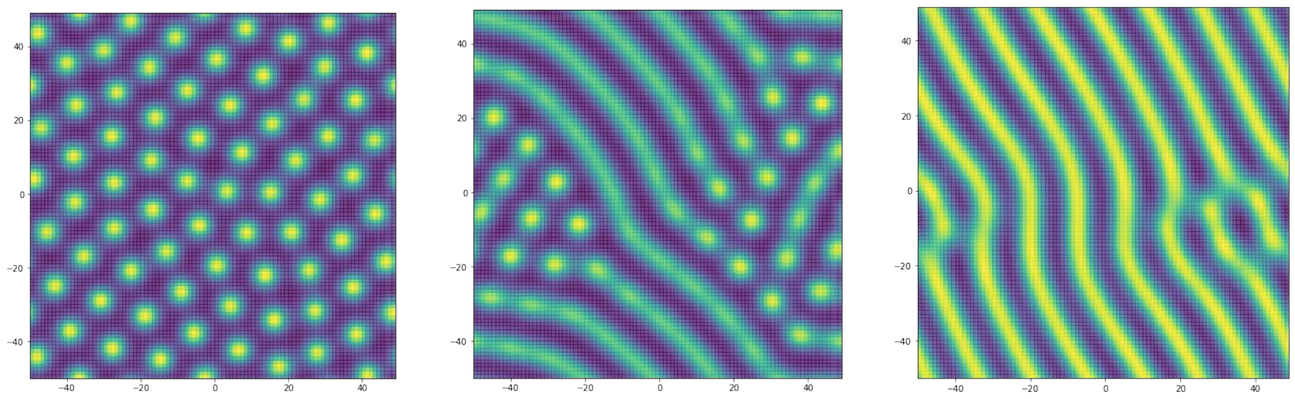
数理生物学演習

第１２回　空間構造の数理モデル（３）：

パターン形成，反応拡散モデル



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１２回：パターン形成

本日の目標

- 2次元配列
- 有限差分法による離散化
- 分子の拡散
- 濃度勾配モデル
- 反応拡散モデル

拡散方程式

熱伝導方程式

$$\frac{\partial u}{\partial t} = D\nabla^2 u$$

拡散が生じる分子などのダイナミクスを記述する
集団遺伝学で出てくることもある

空間微分演算子

$$\nabla = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \cdots, \frac{\partial}{\partial x_n} \right)$$

スカラ量（例えば，拡散性分子の濃度）の勾配

$$\mathrm{grad} u = \nabla u = \frac{\partial u}{\partial x_1}\mathbf{e}_1 + \frac{\partial u}{\partial x_2}\mathbf{e}_2 + \cdots + \frac{\partial u}{\partial x_n}\mathbf{e}_n$$

より詳しく知りたい人は物理数学やベクトル解析などを調べよう

熱伝導方程式の解析解と記述される現象の例

境界条件により異なる解を得ることができる.
ここでは１次元の場合について２つ紹介する.

$u(0,t) = u_0(>0)$

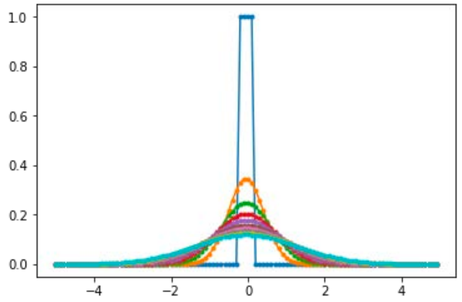
$x \to -\infty, \infty$ で $u(x,t) = 0, \dfrac{\partial u}{\partial t} = 0$

熱伝導方程式

$$\frac{\partial u}{\partial t} = D\nabla^2 u$$

過渡解

$$u(x,t) = \frac{u_0}{\sqrt{4\pi D t}} \exp\left(-\frac{x^2}{4Dt}\right)$$

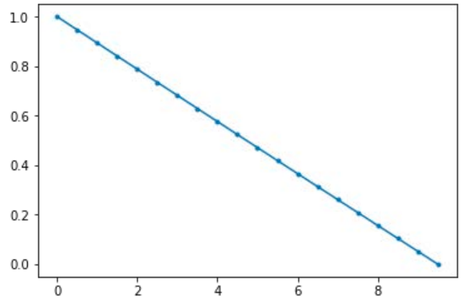


ある部位で生産された分子が広がっていく様子 など

$u(x_0,t) = u_0(>0)$
$u(x_1,t) = u_1(>0)$

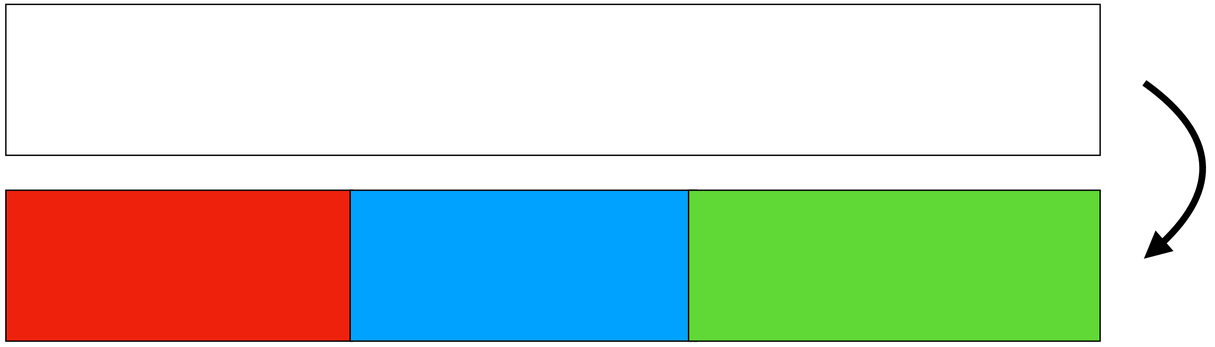
定常解

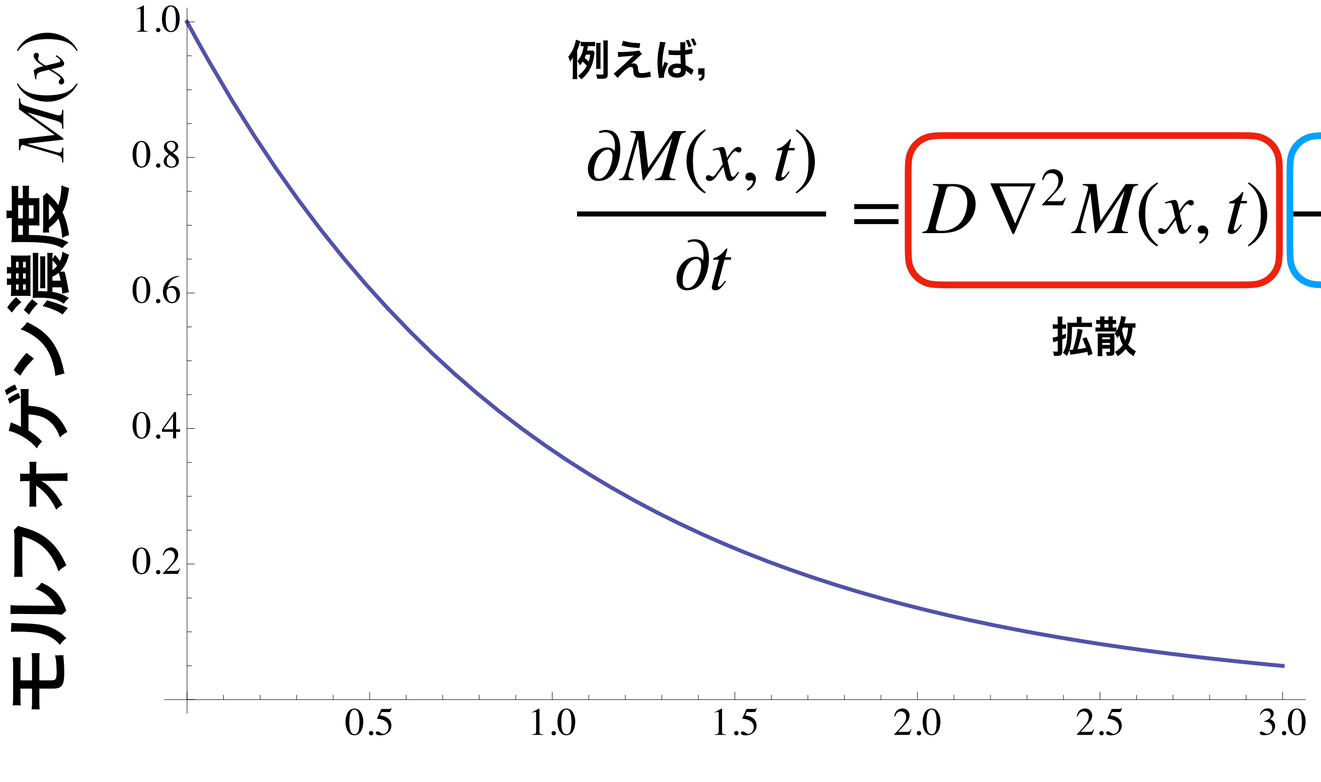
$$u(x) = \frac{u_1 - u_0}{x_1 - x_0}x + u_0$$



膜を隔てた分子の勾配 など

モルフォゲンによるパターン形成





例えば,

$$\frac{\partial M(x,t)}{\partial t} = D\nabla^2 M(x,t) - dM(x,t)$$

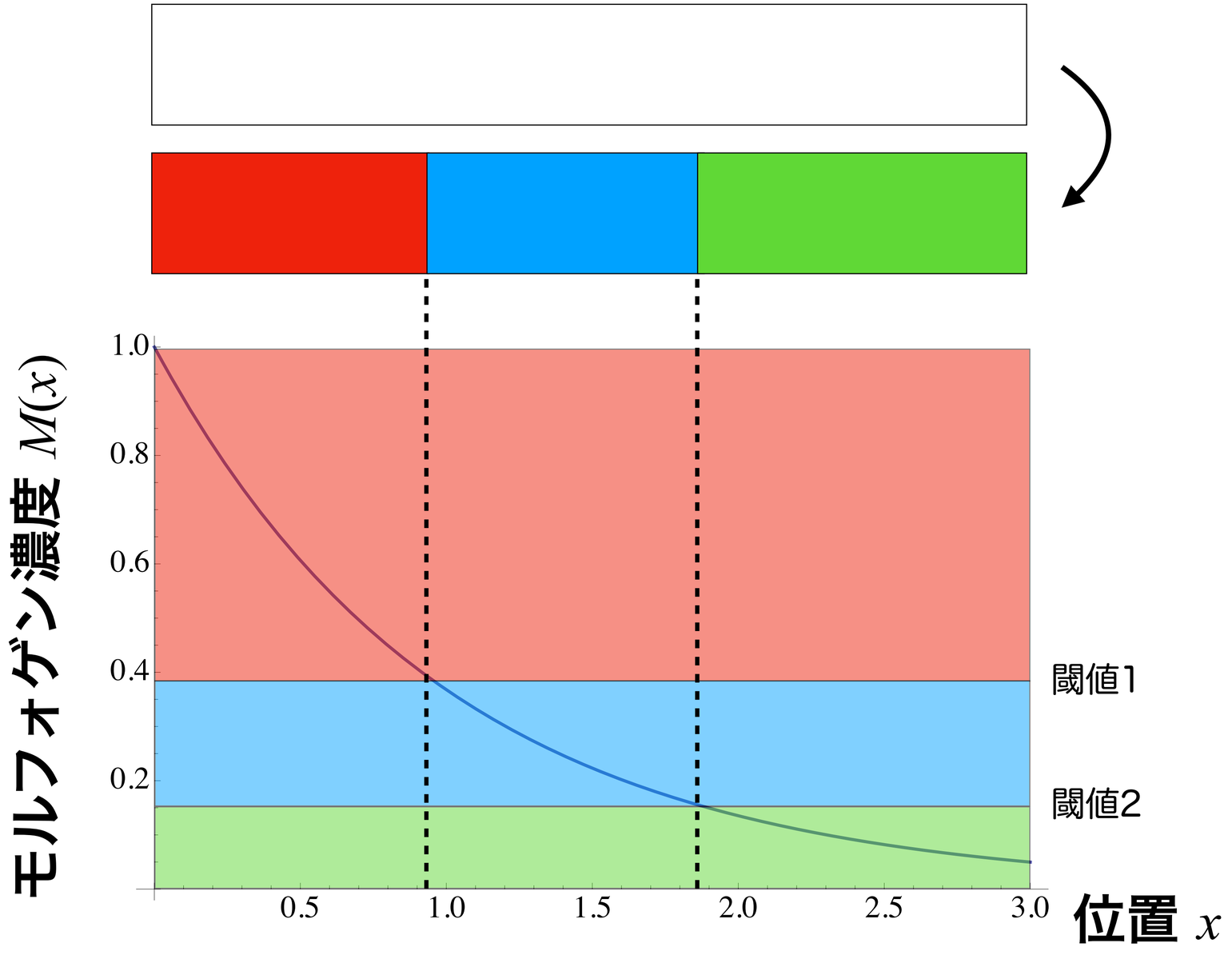
拡散　　分解

t: 時間

D: 拡散係数

d: 分解速度

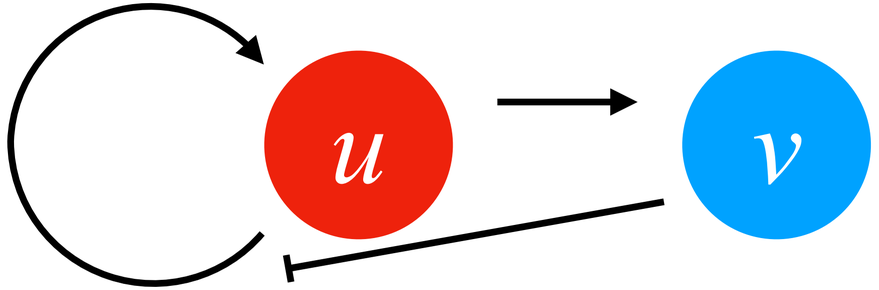
モルフォゲンによるパターン形成



反応拡散モデル ギーラー-マインハルト系

仮定

- uとvは共に拡散する
- uとvは共に一定速度で分解される
- uは自己活性化する
- vはuの合成を抑制する
- uはvの合成を促進する
- uは一定速度で生成される



活性化因子
アクチベーター

抑制因子
インヒビター

自己活性化　合成の抑制

拡散　分解

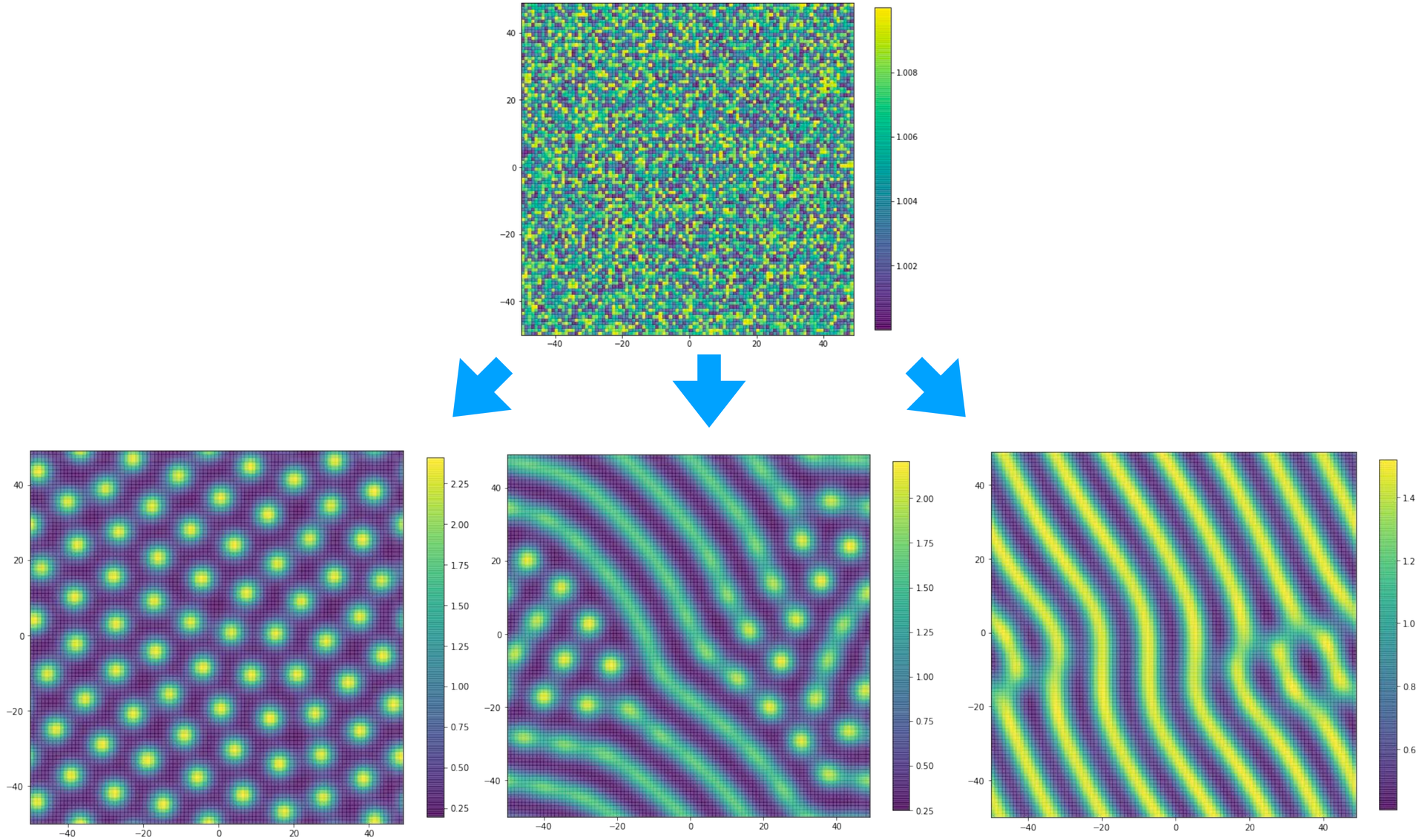
$$\begin{cases} \dfrac{\partial u}{\partial t} = D_u \nabla^2 u - d_u u + k_1 \dfrac{u^2}{v} + k_2 \\ \dfrac{\partial v}{\partial t} = D_v \nabla^2 v - d_v v + k_3 u^2 \end{cases}$$

生成

合成の促成

チューリングパタン

ほぼ一様な構造のない状態から，自発的に空間的パタンが生じる



実際にプログラムを組んでみよう！

有限差分法による空間の離散化

差分商により微分を近似することで，微分方程式を離散化する

差分を刻み幅で割ったもの

本演習では空間方向の離散化に用いる
時間方向へはこれまで通りオイラー法を使う

ある関数$u(x, y)$を2次元空間上で離散化する．
$u_{i,j}=u(x_i, y_j)$, (x_i, y_j)でのuのx方向への偏微分を $\left(\frac{\partial u}{\partial x}\right)_i$, x方向への刻み幅をΔxとすれば，

前進差分による近似

$$\left(\frac{\partial u}{\partial x}\right)_i \approx \frac{u_{i+1,j} - u_{i,j}}{\Delta x}$$

オイラー法と同じ

中心差分による近似

$$\left(\frac{\partial u}{\partial x}\right)_i \approx \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x}$$

後退差分による近似

$$\left(\frac{\partial u}{\partial x}\right)_i \approx \frac{u_{i,j} - u_{i-1,j}}{\Delta x}$$

2階の中心差分による2階偏微分の近似

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_i \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}$$

導出については補足資料参照

y方向へも同様に考える

#01-01．１次元の拡散方程式のプログラムを書く．
固定境界条件（$u(0,t) = u_0, u(x_e, t) = 0$）でシミュレーションしてみよう．

- **時間方向の離散化**

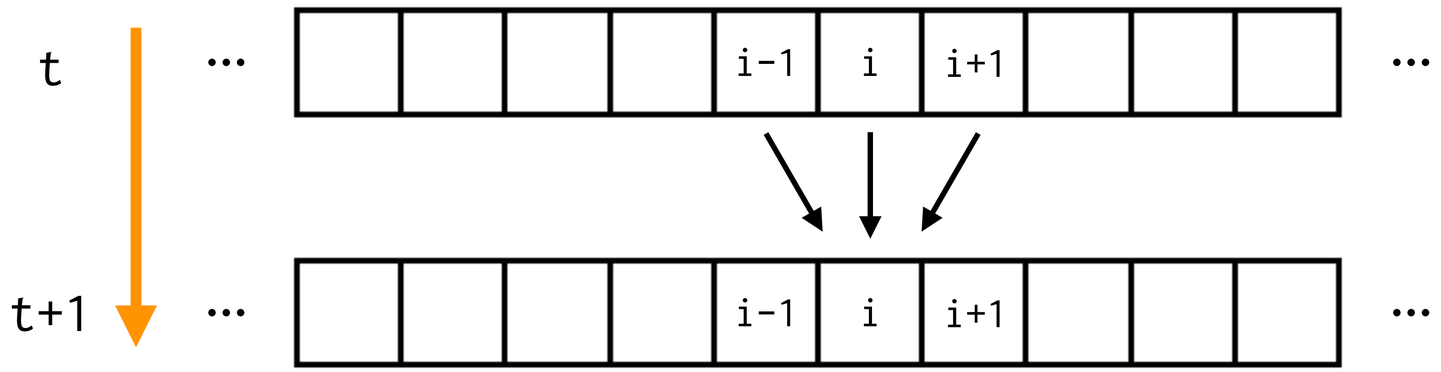
  前進差分により近似（いつものオイラー法）
- **空間方向の離散化**

  2階の中心差分により近似

$$\frac{\partial u}{\partial t} = D\nabla^2 u$$

$$\xrightarrow{\text{離散化}} \quad u_{i,t+\Delta t} = u_{i,t} + D\left[\frac{u_{i-1,t} - 2u_{i,t} + u_{i+1,t}}{\Delta h^2}\right]\Delta t$$

```python
# 01-01-01. １次元の拡散方程式の関数定義
# 時間方向はオイラー法，空間方向は中心差分


def diff_eq_1d(u_arr, D, dh, dt):
    new_u = u_arr[1] + D * ((u_arr[0] + u_arr[2] - 2 * u_arr[1]) / (dh**2)) * dt
    return new_u
```



#01-01．１次元の拡散方程式のプログラムを書く．
固定境界条件（$u(0,t) = u_0, u(x_e, t) = 0$）でシミュレーションしてみよう．

```python
# 01-01-02. １次元拡散モデルのシミュレーション実行
import numpy as np

D = 1.0  # 拡散係数
dt = 0.05  # 時間方向の刻み幅
dx = 0.5  # 空間方向の刻み幅
u_0 = 1  # 境界条件 u(0,t) = u_0
x_e = 10  # xの終端
t_end = 50  # tの終端

x = np.arange(0, x_e, dx)  # x座標
t_list = []  # 時刻を記録するリスト
u_list = []  # uを記録するリスト

u = np.zeros(len(x), dtype=float)  # uの初期化
u[0] = u_0  # 境界条件 u(0,t) = u_0
u[-1] = 0  # 境界条件 u(x_e,t) = 0

t_list.append(0)
u_list.append(np.copy(u))
```

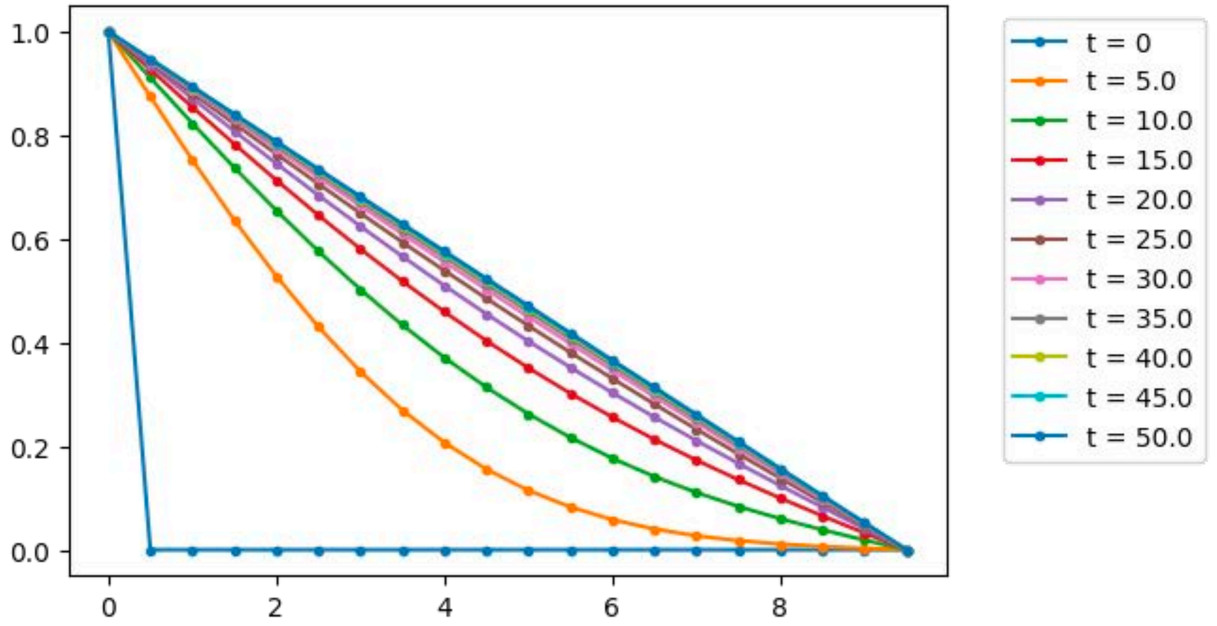
```python
for n in range(1, int(t_end / dt) + 1):
    t = n * dt  # 時刻tの計算
    u_tmp = np.zeros(len(x), dtype=float)

    u_tmp[0] = u_0  # 境界条件 u(0,t) = u_0
    for i in range(1, len(u) - 1):
        u_tmp[i] = diff_eq_1d(
            u[(i - 1) : (i + 2)], D, dx, dt
            )
    u_tmp[-1] = 0  # 境界条件 u(x_e,t) = 0

    u = np.copy(u_tmp)
    t_list.append(t)
    u_list.append(np.copy(u))
```

！注意：拡散係数に対して，時間解像度が大き過ぎる or 空間解像度が小さすぎると本当の解と異なる挙動を示す．
拡散係数を大きくする場合，時間解像度を小さく（ほんの少しの未来だけを計算）するか，空間解像度を大きく（大雑把に計算）する．

少なくとも$\frac{D\Delta t}{\Delta x^2} < \frac{1}{2}$を満たす必要がある．

#01-01. 1次元の拡散方程式のプログラムを書く.
固定境界条件（$u(0,t) = u_0, u(x_e, t) = 0$）でシミュレーションしてみよう.

```python
# 01-01-03. 結果の可視化
import matplotlib.pyplot as plt

plt.figure(dpi=100)
plt.ylim(-0.05, 1.05)
for i in range(0, len(u_list), 100):
    plt.plot(x, u_list[i], ".-", label="t = " + str(t_list[i]))
    plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")
```



時間発展に伴い線形な濃度勾配に近づいていくことが観察できる（$t \to \infty$ で $u(x) = -\frac{u_0}{x_e} + u_0$）.

#01-02. ２次元の拡散方程式のプログラムを書く．周期境界条件でシミュレーションしてみよう．
x軸方向，y軸方向とも空間方向の刻み幅１で，範囲はともに(-25,25)とする．
初期条件はどこか１つの区画で100（$u(x_s, y_s, 0) = 100$）とする．

- **時間方向の離散化**

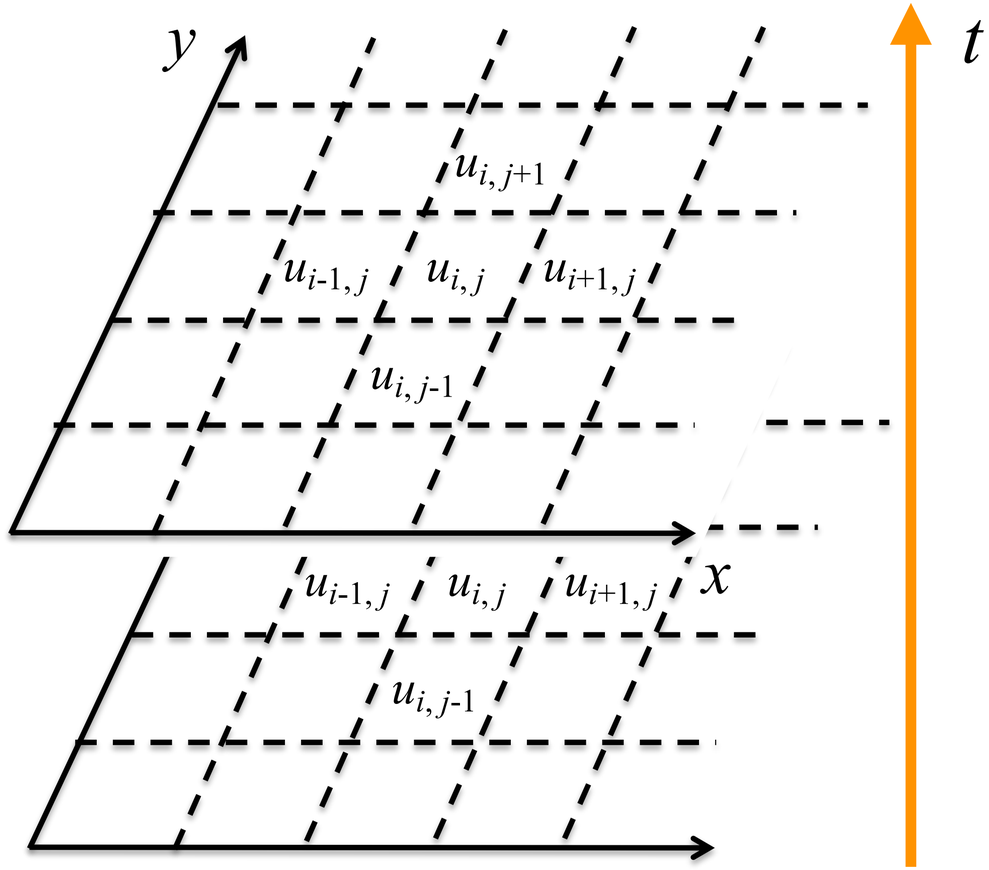
  前進差分により近似

  （いつものオイラー法）
- **空間方向の離散化**

  2階の中心差分により近似

$$\frac{\partial u}{\partial t} = D\nabla^2 u$$

離散化



$$u_{i,j,t+\Delta t} = u_{i,j,t} + D\left(\frac{u_{i-1,j,t} + u_{i+1,j,t} + u_{i,j-1,t} + u_{i,j+1,t} - 4u_{i,j,t}}{\Delta h^2}\right)\Delta t$$

ただし，$\Delta h = \Delta x = \Delta y$.

導出してみよう

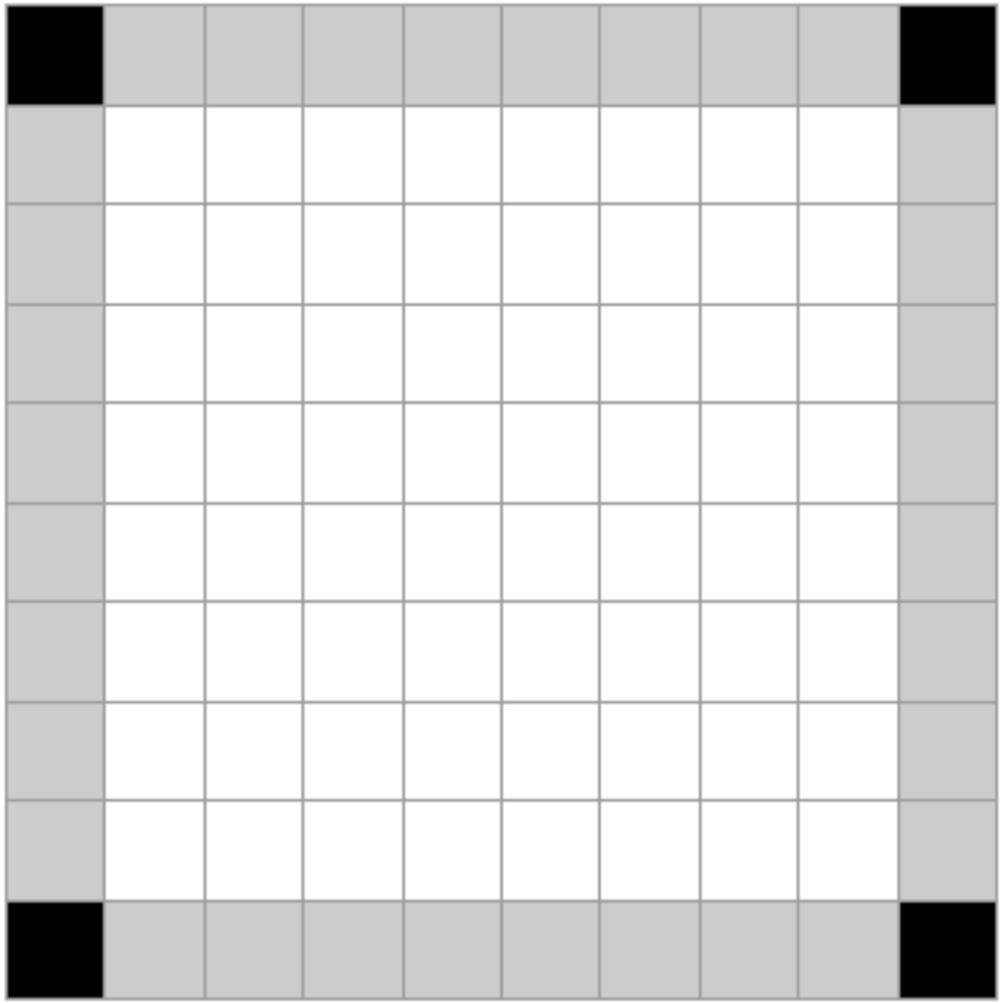
```python
# 01-02-01. ２次元の拡散方程式の関数定義


def diff_eq_2d(u_arr, D, dh, dt):
    new_u = (
        u_arr[1, 1]
        + D
        * (
            (u_arr[0, 1] + u_arr[2, 1] + u_arr[1, 0] + u_arr[1, 2] - 4 * u_arr[1, 1])
            / (dh**2)
        )
        * dt
    )
    return new_u
```

$$u_{i,j,t+\Delta t} = u_{i,j,t} + D\left(\frac{u_{i-1,j,t} + u_{i+1,j,t} + u_{i,j-1,t} + u_{i,j+1,t} - 4u_{i,j,t}}{\Delta h^2}\right)\Delta t$$

```
# 01-02-p. ２次元の拡散方程式の擬似コード

各種パラメータ，初期値の設定
場の設定（x,y）

結果を記録するリストの定義

uの初期化

for ステップ数 in 繰り返し回数:
  時刻tの計算

  # -- 状態遷移 --
  u_tmp = update(u, D, dh, dt)

  情報の更新
  結果のリストへの記録
```

こんな感じのプログラムを作ってみよう．
今回は，update()という関数を定義して，uを一気に計算してみる．

2次元空間での境界条件



境界条件の処理を適切に分岐させよう

```python
# 01-02-02. ２次元の拡散方程式に基づく更新
def update(field, D, dh, dt):
  n_i, n_j = field.shape
  new_field = np.zeros((n_i, n_j), dtype=field.dtype)

  for i in range(n_i):
    for j in range(n_j):
      if i == 0:
        if j == 0:
          # 境界条件処理 i=0,j=0
          new_field[i,j] = diff_eq_2d(field[[-1,0,1],:][:,[-1,0,1]], D, dh, dt)
        elif 0<j<n_j-1:
          # 境界条件処理 i=0,0<j<n_j-1
          new_field[i,j] = diff_eq_2d(field[[-1,0,1],:][:,[j-1,j,j+1]], D, dh, dt)
        elif j == n_j-1:
          # 境界条件処理 i=0,j=n_j-1
          new_field[i,j] = diff_eq_2d(field[[-1,0,1],:][:,[n_j-2,n_j-1,0]], D, dh, dt)
      elif 0 < i < n_i-1:
        if j == 0:
          # 境界条件処理 1<i<n_i-1,j=0
          new_field[i,j] = diff_eq_2d(field[[i-1,i,i+1],:][:,[-1,0,1]], D, dh, dt)
        elif 0 < j< n_j-1:
          # メイン （1<i<n_i-1,1<j<n_j-1）
          new_field[i,j] = diff_eq_2d(field[[i-1,i,i+1],:][:,[j-1,j,j+1]], D, dh, dt)
        elif j == n_j-1:
          # 境界条件処理 1<i<n_i-1,j=n_j-1
          new_field[i,j] = diff_eq_2d(field[[i-1,i,i+1],:][:,[n_j-2,n_j-1,0]], D, dh, dt)
      elif i == n_i-1:
        if j == 0:
          # 境界条件処理 i=n_i,j=0
          new_field[i,j] = diff_eq_2d(field[[n_i-2,n_i-1,0],:][:,[-1,0,1]], D, dh, dt)
        elif 0<j<n_j-1:
          # 境界条件処理 i=n_i-1,1<j<n_j-1
          new_field[i,j] = diff_eq_2d(field[[n_i-2,n_i-1,0],:][:,[j-1,j,j+1]], D, dh, dt)
        elif j == n_j-1:
          # 境界条件処理 i=n_i-1,j=n_j-1
          new_field[i,j] = diff_eq_2d(field[[n_i-2,n_i-1,0],:][:,[n_j-2,n_j-1,0]], D, dh, dt)

  return new_field
```

場（ここではu）の配列の形状（幅と高さ）を取得

配列のスライスを思い出そう

#01-02．２次元の拡散方程式のプログラムを書く．周期境界条件でシミュレーションしてみよう．
x軸方向，y軸方向とも空間方向の刻み幅１で，範囲はともに(-25,25)とする．
初期条件はどこか１つの区画で100（$u(x_s, y_s, 0) = 100$）とする．

これを

```
# 01-02-p. ２次元の拡散方程式の擬似コード

各種パラメータ，初期値の設定
場の設定（x,y）

結果を記録するリストの定義

uの初期化

for ステップ数 in 繰り返し回数:
  時刻tの計算

  # -- 状態遷移 --
  u_tmp = update(u, D, dh, dt)

  情報の更新
  結果のリストへの記録
```

実際に実装すると，こんな感じ．→

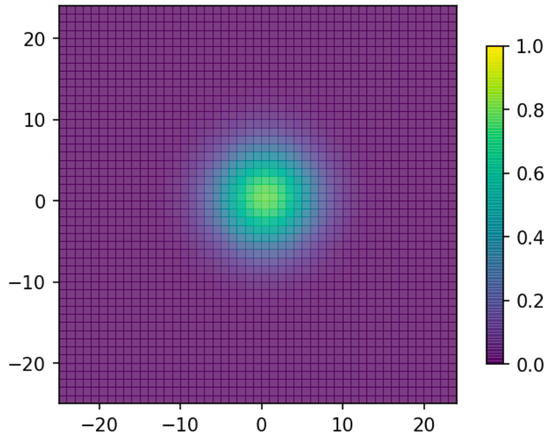
```python
# 01-02-03. ２次元拡散モデルのシミュレーション実行
D = 1.0  # 拡散係数
dt = 0.05  # 時間方向の刻み幅
dh = 1  # 空間方向の刻み幅
u_0 = 100  # uの初期濃度

x = np.arange(-25, 25, dh)
y = np.arange(-25, 25, dh)
xmesh, ymesh = np.meshgrid(x, y)

t_end = 50  # tの終端

t_list = []  # 時刻を記録するリスト
u_list = []  # uを記録するリスト

u = np.zeros([len(x), len(y)], dtype=float)  # uの初期化
u[25, 25] = u_0  # 境界条件 u(0,t) = u_0

t_list.append(0)
u_list.append(np.copy(u))

for n in range(int(t_end / dt)):
    t = (n + 1) * dt  # 時刻tの計算

    # -- 状態遷移 --
    u_tmp = update(u, D, dh, dt)

    # 情報の更新
    t_list.append(t)
    u = np.copy(u_tmp)
    u_list.append(np.copy(u))
```

ここに挙げているのはあくまで一例．
自分がやりやすい方法で実装してOK．

プロットしたいuListの要素（ある時刻のuの結果）を指定する．
ここでは200ステップ目（時刻 200*dt）をプロットしている

```
# 01-02-04a. 結果の可視化
fig, ax = plt.subplots(dpi=150)
ax.set_aspect("equal")
pcm = ax.pcolormesh(xmesh, ymesh, u_list[200], alpha=0.75, vmin=0, vmax=1)
fig.colorbar(pcm, ax=ax, shrink=0.8)
plt.show()
```

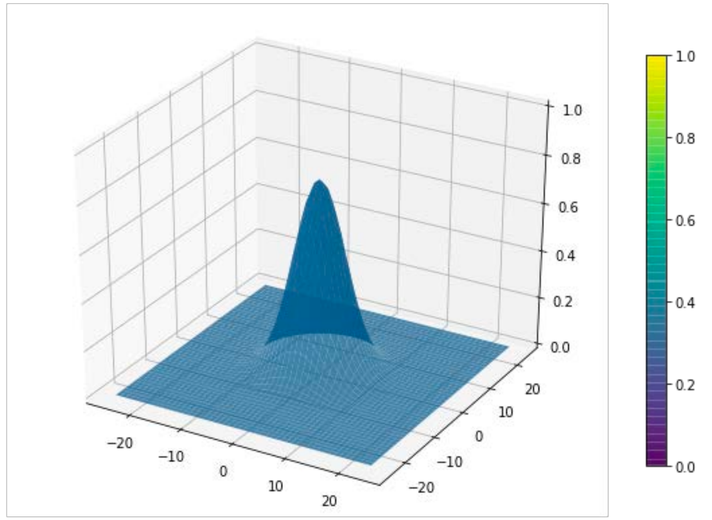
vmin,vmaxはプロットするZの値の範囲．
指定しなければ自動で規格化される．



- matplotlib.pyplot
  - pcolormesh(X, Y, Z) 座標X, Yに対してZの値を色でプロット
- mpl_toolkits.mplot3d.Axes3D
  - plot_surface(X, Y, Z) 座標X, Yに対してZの値を高さで3Dプロット

```
# 01-02-4b. 結果の可視化（3D）
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection="3d")
surf = ax.plot_surface(xmesh, ymesh, u_list[200], vmin=0, vmax=1, alpha=0.75)
ax.plot([0, 0], [0, 0], [0, 1], "w", alpha=0.1)
fig.colorbar(surf, shrink=0.8)
plt.grid()
plt.show()
```

プロットされるZの値の範囲
を調整するためのトリック



plotlyを使えばインタラクティブな3Dプロットができる．興味ある人は試してみよう．

どうすれば結果を確認しやすいかを考えて，他のプロット方法も検討しよう．

反応拡散モデル

02．ギーラー-マインハルト系の反応拡散モデルについてプログラムを組み，様々なパタンを描く

方針

1. モデルの離散化

アクチベーター

インヒビター

$$\begin{cases} \dfrac{\partial u}{\partial t} = D_u \nabla^2 u - d_u u + k_1 \dfrac{u^2}{v} + k_2 \\ \dfrac{\partial v}{\partial t} = D_v \nabla^2 v - d_v v \ + k_3 u^2 \end{cases}$$

離散化 → **？**

2. 2次元拡散方程式を参考にプログラムを組む

- **基本的には，拡散方程式を反応拡散方程式系に変えるだけ．ただし，拡散性分子が2種類あり，相互に依存することに注意．**
- **どのようなupdate関数を定義すれば良いだろうか？**

3. 初期値の設定

u=1.0, v=1.0にわずかなノイズ(0.0〜0.01程度)を加える．

4. 拡散係数（Du, Dv）を変化させてどのようなパタンが生じるか調べる

とりあえずおすすめのパラメータ

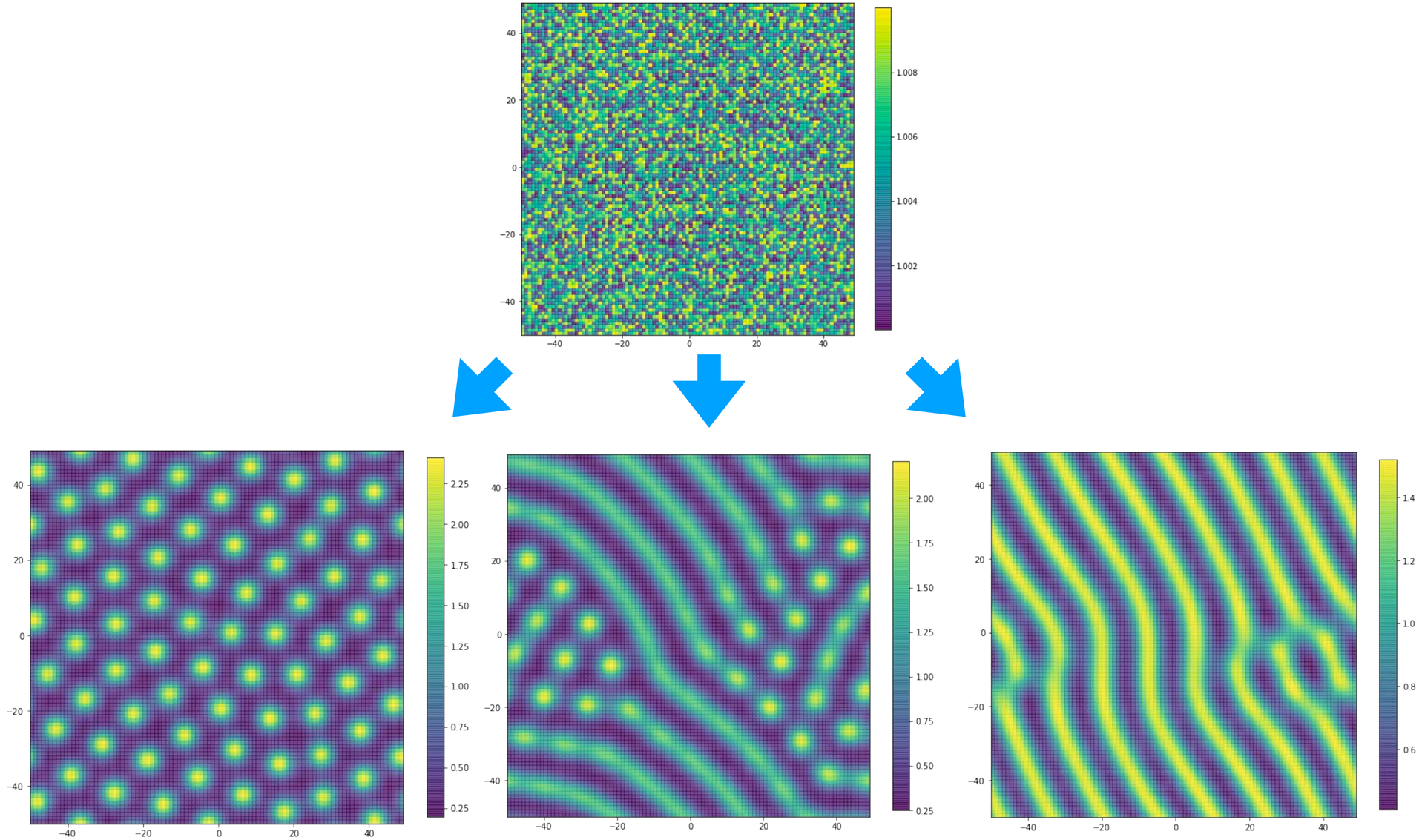
$d_u = 1, d_v = 1, k_1 = 1, k_2 = 0.05, k_3 = 1$

D_u, D_vは，

$D_u < D_v$で色々試してみよう

チューリングパタン

ほぼ一様な構造のない状態から，自発的に空間的パタンが生じる



反応拡散モデル 疑似コード

```
# 02-p. ２次元の反応拡散モデルの擬似コード

各種パラメータ，初期値の設定
場の設定（x,y）

結果を記録するリストの定義

uの初期化
vの初期化

for ステップ数 in 繰り返し回数:
   時刻tの計算

   # -- 状態遷移 --

   情報の更新
   結果のリストへの記録
```

どんな状態遷移用の関数を定義したら良いだろうか？

本日の課題

ノーマル：
１つ選ぶ

ハード：
両方

1. 反応拡散系のパラメータや初期値を変化させた様々なパタンを観察せよ．また，どういった傾向があるかを考察せよ．
2. 反応拡散系のパラメータや初期値を変化させて生物の体表面に観察される模様を幾つか再現せよ．また，それはどういった生物にみられるか例を挙げよ．
3. 質問，意見，要望等をどうぞ．

課題をGitHub Classroomにて提出すること

次回予告

第13回：数理生物学は役に立つのか？（３）

：研究紹介　???

7月28日

復習推奨

- NumPy配列