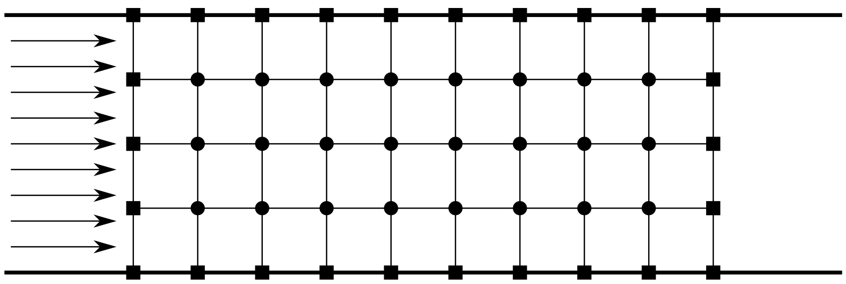
Incompressible Flow

田浦

1　やりたいこと

2 次元ハーゲン・ポアズイユ流を以下の領域でシミュレートしたい



2　方程式

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \nabla p + c\Delta u \tag{1}$$

$$\nabla \cdot u = 0 \tag{2}$$

3　解法

非圧縮の方程式の普通の解き方.

右辺の p を求めるために式 (1) の両辺の $\nabla\cdot$ をとる.

$$\frac{\partial \nabla \cdot u}{\partial t} + \nabla \cdot (-(u \cdot \nabla)u - \nabla p + c\Delta u)h \tag{3}$$

連続の式 (2) より $\nabla \cdot u = 0$ なので

$$0 = \nabla \cdot (-(u \cdot \nabla)u - \nabla p + c\Delta u) \tag{4}$$

よって

$$-\nabla \cdot (u \cdot \nabla)u - \Delta p + c\nabla \cdot \Delta u = 0 \tag{5}$$

さらに

$$\nabla \cdot \Delta u = \Delta(\nabla \cdot u) = 0 \tag{6}$$

なので,

$$\Delta p = -\nabla \cdot (u \cdot \nabla)u \tag{7}$$

よって, 1 ステップの計算は,

1. 式 (8) を用いて p を求める.

$$\Delta p = -\nabla \cdot (u \cdot \nabla) u \tag{8}$$

2. 求まった p を用いて, $u(t+h)$ を求める.

$$u(t+h) \approx u(t) + (-(u \cdot \nabla)u - \nabla p + c\Delta u)h \tag{9}$$

4 計算の方法＋よくわからないこと

一番単純に各格子点に p, u を割り当てて (スタッガード格子とかは必要だとわかるまでとりあえずやらないで), 差分法で解こうと思っている.

その際に境界条件の与え方がよくわからない.

1. 圧力の式 (8) を解くときの境界条件は?

   どこぞには, ノイマン条件を課すみたいなことが書いてある.

$$\frac{\partial p}{\partial n} = 0 \tag{10}$$

2. u に関する境界条件は?

$$\text{上下の壁沿いについては } u = 0 \tag{11}$$

   でよいなのだろう (多分).

   では左右の空いているところはどうしたら良い? (ここが一番良くわからない)

   とりあえずよくわからないながら,

$$\text{左 (流入して来る所) については } u = (1, 0), \tag{12}$$

$$\text{右 (流出して行く所) については} \frac{\partial u}{\partial n} = 0 \tag{13}$$

   という条件を課している. 後者の n は x 軸方向のベクトル. 最終的に得たい答えはポアズイユ流の, $u_x = (y - 1/2)^2$ みたいな答えなので, 明らかに違うのだが, と言って, まさかこれを境界条件というわけにも行かず, どうしたものかという感じ.

ちなみに差分法で計算するときの表面的な問題として, 格子点の数を $m \times n$ とすると, 式 (8) の右辺を作る時に微分を差分で計算する関係で, 右辺が計算できるのは, 内部の点 $((m-2) \times (n-2))$ だけになる. 従って, 連立一次方程式を用いて求まる p も内部の点だけとなる (ノイマン条件を課しているので, 外周の点は内部の点と同じとして計算する. つまり, $p[0, j] = p[1, j]$, $p[m-1, j] = p[m-2, j]$, $p[i, 0] = p[i, 1]$, $p[i, n-1] = p[i, n-2]$).

そうやって求まった ($m \times n$ 点の)p を用いて式 (9) の右辺を計算するときも, やはり微分を差分で作る関係で, 右辺が計算できるのは内部の点だけとなる. そのため外周の点に対しては $u(t+h)$ の値が (少なくとも素直には) 求まらない. 壁沿いについては物理的に直感にあう境界条件 (式**??**) があるわけだが, 左右の空いているところはさてどうするのが良いのだろう.

コードを添付します. matplotlib が必要です. python3 で走らせてくれれば勝手にアニメーションが始まるはずです.

やってみると最初のうち中なか変化がないけど, よく見ると徐々に色が変わってきます. そしてそのうちに破綻した感じにぐちゃぐちゃになります.

何が悪いのか.

- 境界条件が悪い
- dt の値が大きすぎる
- オイラー法が荒すぎる
- 空間微分が荒すぎる
- ポアソン方程式の行列の係数のところでバグを入れている
- . . .

など色々あるとおもうのですが. あまりこれ以上一人で悶々と悩みたくないので, 頭の整理を兼ねてこんな文章を書いてみたというわけです.

5 付録: コード

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import scipy.sparse
import scipy.sparse.linalg
import time
import pdb

# 次元で 2 ハーゲン・ポアズイユ流をシミュレート
#
#             ux = 0
#    +----------------------+
#  ---->
#  ---->
#  ---->
#  ---->
#    +----------------------+
#             ux = 0

def partial_by_x(f):
    """
    f : scalar field (m x n array)
    returns : ∂ f∂/x  ((m - 1) x n array)
    """
    m,n = f.shape
    p = f[1:,:] - f[:-1,:]
    assert(p.shape == (m - 1, n)), (p.shape, (m - 1, n))
    return p

def partial_by_y(f):
    """
    f : scalar field (m x n array)
    returns : ∂ f∂/y  (m x (n - 1) array)
```

```
    """
    m,n = f.shape
    p = f[:,1:] - f[:,:-1]
    assert(p.shape == (m, n - 1)), (p.shape, (m, n - 1))
    return p

def grad_f(f):
    """
    f : scalar field (m x n array)
    returns : ∂ (f∂/x, ∂ f∂/y)  (2 x (m - 1) x (n - 1) array)
    """
    m,n = f.shape
    gx = partial_by_x(f)
    gy = partial_by_y(f)
    r =  np.vstack([ gx[:,1:], gy[1:,:] ]).reshape((2, m - 1, n - 1))
    return r

def u_dot_grad_f(u, f):
    """
    u : vector field (2 x m x n)
    f : scalar field (m x n)
    compute ( · ∇ u) f = (u0 ∂ ∂/x + u1 ∂ ∂/y) f
    """
    d,m,n = u.shape
    assert(d == 2), d
    assert(f.shape == (m, n)), (f.shape, (m, n))
    g = u[0,1:,1:] * partial_by_x(f[:,1:]) + u[1,1:,1:] * partial_by_y(f[1:,:])
    assert(g.shape == (m - 1, n - 1)), (g.shape, (m - 1, n - 1))
    return g

# v : vector field
# compute ( · ∇ u) v
def u_dot_grad_v(u, v):
    """
    u : vector field (2 x m x n)
    v : vector field (2 x m x n)
    compute ( · ∇ u) v = ((u0 ∂ ∂/x + u1 ∂ ∂/y) v0, (u0 ∂ ∂/x + u1 ∂ ∂/y) v1)
    """
    d,m,n = v.shape
    assert(d == 2), d
    assert(u.shape == (d,m,n)), (u.shape, (d,m,n))
    gx = u_dot_grad_f(u, v[0,:,:])
    gy = u_dot_grad_f(u, v[1,:,:])
    assert(gx.shape == (m - 1, n - 1)), (gx.shape, (m - 1, n - 1))
    assert(gy.shape == (m - 1, n - 1)), (gy.shape, (m - 1, n - 1))
    r =  np.vstack([ gx, gy ]).reshape((2, m - 1, n - 1))
    return r

def div(v):
    """
    v : vector field (2 x m x n)
    compute ∇ · v =  ∂ v0∂/x + ∂ v1∂/y
    """
    d,m,n = v.shape
    g = partial_by_x(v[0,:,1:]) + partial_by_y(v[1,1:,:])
    assert(g.shape == (m - 1, n - 1)), (g.shape, (m - 1, n - 1))
    return g

def laplace_f(f):
    """
    f : scalar field (m x n)
    returns : Δ f =  ∂^2f∂/x^2 + ∂^2f∂/y^2  ((m - 1) x (n - 1))
    """
    m,n = f.shape
    l = f[:-2,1:-1] + f[2:,1:-1] + f[1:-1,:-2] + f[1:-1,2:] - 4 * f[1:-1,1:-1]
    assert(l.shape == (m - 2, n - 2)), (l.shape, (m - 2, n - 2))
    return l

```

```
def laplace_v(v):
    """
    v : vector field (2 x m x n)
    returns : Δ v =  ∂ (^2v0∂/x^2 + ∂^2v0∂/y^2,
                      ∂^2v1∂/x^2 + ∂^2v1∂/y^2)
              (2 x (m - 1) x (n - 1))
    """
    d,m,n = v.shape
    assert(d == 2), d
    lx = laplace_f(v[0,:,:])
    ly = laplace_f(v[1,:,:])
    assert(lx.shape == (m - 2, n - 2)), (lx.shape, (m - 2, n - 2))
    assert(ly.shape == (m - 2, n - 2)), (ly.shape, (m - 2, n - 2))
    l = np.vstack([ lx, ly ]).reshape((d, m - 2, n - 2))
    return l

def make_laplace_matrix(m, n):
    """
    make matrix A for (m x n) region ((m * n) x (m * n) matrix).
    assume neumann boundary condition
    """
    D = {}
    for i in range(m):
        for j in range(n):
            D[i,j,i,j] = -4
            if i + 1 < m:
                D[i,j,i+1,j] = 1
            else:
                D[i,j,i,j] += 1
            if i > 0:
                D[i,j,i-1,j] = 1
            else:
                D[i,j,i,j] += 1
            if j + 1 < n:
                D[i,j,i,j+1] = 1
            else:
                D[i,j,i,j] += 1
            if j > 0:
                D[i,j,i,j-1] = 1
            else:
                D[i,j,i,j] += 1
    I = {}
    for i in range(m):
        for j in range(n):
            I[i,j] = len(I)
    N = m * n
    A = scipy.sparse.dok_matrix((N, N))
    for (i,j,ii,jj),v in D.items():
        A[I[i,j],I[ii,jj]] = v
    return scipy.sparse.csr_matrix(A)

def solve_poisson(p, f):
    """
    f : scalar field (m x n)
    solve Δ p = f
    """
    m,n = f.shape
    N = m * n
    A = make_poisson_matrix(m, n)
    p[1:-1,1:-1] = scipy.sparse.linalg.spsolve(A, f.reshape(N)).reshape((m, n))
    p[0,1:-1] = p[1,1:-1]
    p[-1,1:-1] = p[-2,1:-1]
    p[1:-1,0] = p[1:-1,1]
    p[1:-1,-1] = p[1:-1,-2]
    p[0,0] = p[1,1]
    p[-1,-1] = p[-2,-2]
    p[0,-1] = p[1,-2]
    p[-1,0] = p[-2,1]
```

```
def step(u, p, dt):
    """
    calc 1 step.
    u : velocity field (vector field) 2 x m x n
    p : pressure field (scalar field) m x n
    """
    d,m,n = u.shape
    assert(d == 2), d
    assert(p.shape == (m,n)), (p.shape, (m, n))
    c = 1                       # 1/Re  (Re : Reynolds constant)
    # calc righthand side of the poission equation
    # ( · ▽ u)u
    vgu = u_dot_grad_v(u, u)
    assert(vgu.shape == (d, m - 1, n - 1)), (vgu.shape, (d, m - 1, n - 1))
    # - ▽ · ( · ▽ u)u
    div_vgu = - div(vgu)
    assert(div_vgu.shape == (m - 2, n - 2)), (div_vgu.shape, (m - 2, n - 2))
    # solve △ p = - ▽ · ( · ▽ u)u
    solve_poisson(p, div_vgu)
    # ▽ p
    gp = grad_f(p)
    assert(gp.shape == (d, m - 1, n - 1)), (gp.shape, (d, m - 1, n - 1))
    # - ( · ▽ u)u - ▽ p + c △ u
    dudt = - vgu[:,1:,1:] - gp[:,1:,1:] + c * laplace_v(u)
    assert(dudt.shape == (d, m - 2, n - 2)), (dudt.shape, (d, m - 2, n - 2))
    u[:,1:-1,1:-1] += dudt * dt
    # impose boundary condition on the right open boundary
    u[:,-1,1:-1] += u[:,-2,1:-1]

#
# 以下の simulate1 〜 simulate3 は
# アニメーションのやり方を試行錯誤しているもの
# simulate3 が一番シンプル
#

def simulate1():
    d = 2                       # dimension (2)
    m = 200                     # x-axis 200
    n = 50                      # y-axis 50
    dt = 0.001
    u = np.zeros((d, m, n))
    u[0,0,:] = 1.0              # left open boundary
    p = np.zeros((m, n))
    n_steps = 200
    n_imgs = 10
    imgs = []
    fig = plt.figure()
    for i in range(n_steps):
        print("step %d" % i)
        while len(imgs) / n_imgs < (i + 1) / n_steps:
            imgs.append([visualize_f(u[0])])
            print("   add img -> %d" % len(imgs))
        step(u, p, dt)
    ani = animation.ArtistAnimation(fig, imgs, interval=500)
    plt.show()
    return ani

def shrink(z):
    m,n = z.shape
    return z[:m-1,:n-1].reshape((m - 1) * (n - 1))

def visualize_f(f):
    """
    f : scalar field
    """
    m,n = f.shape
    x = np.linspace(0, m - 1, m)
```

```
    y = np.linspace(0, n - 1, n)
    x,y = np.meshgrid(x, y, indexing='ij')
    return plt.pcolor(x, y, f)

def simulate2():
    d = 2                       # dimension (2)
    m = 400                     # x-axis 200
    n = 100                      # y-axis 50
    dt = 0.01
    u = np.zeros((d, m, n))
    u[0,0,:] = 1.0              # boundary
    p = np.zeros((m, n))
    n_steps = 100
    # fig = plt.figure()
    for i in range(n_steps):
        print("step %d" % i)
        if i == 0:
            field = visualize_f(u[0])
            plt.legend()
        else:
            field.set_array(shrink(u[0]))
        yield [field]
        step(u, p, dt)

def do_animation(iterator, **kwargs):
    def anime_fun(*args):
        try:
            return next(iterator)
        except StopIteration:
            return []
    ani = animation.FuncAnimation(plt.gcf(), anime_fun, **kwargs)
    plt.show()
    return ani

def simulate3():
    d = 2                       # dimension (2)
    m = 100                     # x-axis 200
    n = 50                      # y-axis 50
    dt = 0.01
    u = np.zeros((d, m, n))
    u[0,0,:] = 1.0              # boundary
    p = np.zeros((m, n))
    n_steps = 100
    # fig = plt.figure()
    for i in range(n_steps):
        print("step %d" % i)
        if i == 0:
            field = plt.imshow(u[0])
            fig = plt.gcf()
            #plt.clim()   # clamp the color limits
            plt.colorbar()
        else:
            #field.set_array(shrink(u[0]))
            field.set_data(u[0])
        step(u, p, dt)
        plt.pause(0.5)

def main():
    # return simulate1()
    # return do_animation(simulate2(), interval=100)
    return simulate3()

ani = main()
```