Stokes_P1CRP0

October 11, 2020

1 Solving Stokes equation with (P_1^{CR}, P_0)

[1]:
```
from skfem import *
import numpy as np
from skfem.visuals.matplotlib import draw, plot
from skfem.utils import solver_iter_krylov, solver_eigen_scipy, solver_iter_pcg
from skfem.helpers import dd, ddot, div, grad
from scipy.sparse.linalg import LinearOperator, minres
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.sparse import bmat
import dmsh
from skfem.assembly import BilinearForm, LinearForm
plt.rcParams['figure.dpi'] = 100
```

1.1 Problem description

$$\begin{cases} -\Delta \boldsymbol{u} + \nabla p = \boldsymbol{f} \; in \; \Omega \\ \nabla \cdot \boldsymbol{u} = 0 \; in \; \Omega \\ u = 0 \; on \; \partial\Omega \end{cases}$$

where

$$\begin{cases} \Delta \mathbf{u} = \sum_{i=1}^{N} \frac{\partial^2 \mathbf{u}}{\partial x_i \partial x_i} \\ \nabla p = \left(\frac{\partial p}{\partial x_1}, \frac{\partial p}{\partial x_2}, \dots, \frac{\partial p}{\partial x_N} \right) \\ \nabla \cdot \mathbf{u} = \sum_1^N \frac{\partial u_i}{\partial x_i} \end{cases}$$

u is the velocity vector and p is the pressure

Testing case in $\Omega = [0,1] \times [0,1]$:

$$\begin{cases} u1 = 10x^2(x-1)^2 y(y-1)(2y-1) \\ u2 = -10y^2(y-1)^2 x(x-1)(2x-1) \\ p = x^2 - y^2 \end{cases}$$

$$\begin{pmatrix} A & -B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} F_1 \\ 0 \end{pmatrix}$$

1.2 Adding P_1^{CR} element

[2]:
```
class ElementTriP1CR(ElementH1):

    facet_dofs = 1
    dim = 2
    maxdeg = 1
    dofnames = ['u']
    doflocs = np.array([[.5, 0.], [.5, .5], [0., .5]])
    mesh_type = MeshTri

    def lbasis(self, X, i):
        x, y = X

        if i == 0:
            phi = 1. - 2. * y
            dphi = np.array([0. * x, -2. + 0. * y])
            dphi = np.array([0. * x, -2. + 0. * y])
        elif i == 1:
            phi = 2. * x + 2. * y - 1.
            dphi = np.array([2. + 0. * x, 2. + 0. * y])
        elif i == 2:
            phi = 1. - 2. * x
            dphi = np.array([-2. + 0. * x, 0. * x])
        else:
            self._index_error()
        return phi, dphi
```

1.3 Defining forms

[3]:
```
@BilinearForm
def vector_laplace(u, v, w):
    '''
    a
    '''
    return ddot(grad(u), grad(v))


@BilinearForm
def divergence(u, v, w):
    '''
    b
    '''
    return div(u) * v


@LinearForm
```

```python
def body_force(v, w):
    '''
    for f.*v
    '''
    x, y = w.x
    f1 = 10 * (12 * x**2 - 12 * x + 2) * y * (y - 1) * (2 * y - 1) + 10 * (
        x**2) * ((x - 1)**2) * (12 * y - 6) + 2 * x
    f2 = -(10 * (12 * y**2 - 12 * y + 2) * x * (x - 1) * (2 * x - 1) + 10 *
           (y**2) * ((y - 1)**2) * (12 * x - 6)) - 2 * y
    return f1 * v.value[0] + f2 * v.value[1]


@BilinearForm
def mass(u, v, w):
    '''
    c
    '''
    return u * v
```

1.4 Defining exact value and L_2 error

[4]:
```python
def exactu(x, y):
    u1 = 10 * (x**2) * ((x - 1)**2) * y * (y - 1) * (2 * y - 1)
    u2 = -10 * (y**2) * ((y - 1)**2) * x * (x - 1) * (2 * x - 1)
    return -u1, -u2


@Functional
def L2Error_u(w):
    x, y = w.x
    u1, u2 = exactu(x, y)
    #     print((w.w[0] - u1)**2)
    #     print((w.w[1] - u2)**2)
    return (w.w[0] - u1)**2 + (w.w[1] - u2)**2
```

[5]:
```python
def exactp(x, y):
    return x**2 - y**2


@Functional
def L2Error_p(w):
    x, y = w.x
    return (w.w - exactp(x, y))**2
```

1.5 Caculating error and convergence rate

result: h^2 for L2u and h^1 for L2p

[6]:
```
formerL2p = 1
currentL2p = 1
formerL2u = 1
currentL2u = 1
for i in range(7):
    mesh = MeshTri()
    mesh.refine(i)
    element = {'u': ElementVectorH1(ElementTriP1CR()), 'p': ElementTriP0()}
    basis = {
        variable: InteriorBasis(mesh, e, intorder=4)
        for variable, e in element.items()
    }  # intorder: integration order for quadrature

    A = asm(vector_laplace, basis['u'])
    B = asm(divergence, basis['u'], basis['p'])
    C = asm(mass, basis['p'])

    K = bmat([[A, -B.T], [-B, 1e-6 * C]],
             'csr')  # get the sparse format of the result by 'csr'
    f = np.concatenate([asm(body_force, basis['u']), np.zeros(B.shape[0])])

    up = solve(*condense(K, f, D=basis['u'].find_dofs()),
               solver=solver_iter_pcg())

    uh, ph = np.split(up, [A.shape[0]])
    # p = exactp(basis['p'].doflocs[0], basis['p'].doflocs[1])
    # print((np.sqrt(np.sum((p-ph)**2)))/len(ph))
    P = basis['p'].interpolate(ph).value
    L2p = np.sqrt(L2Error_p.assemble(basis['p'], w=P))
    U = basis['u'].interpolate(uh).value
    L2u = np.sqrt(L2Error_u.assemble(basis['u'], w=U))
    # print('L2u Error:', L2u)
    # print('L2p Error', L2p)
    currentL2p = L2p
    currentL2u = L2u
    if i != 0:
        print('2^-' + str(i + 1) + ' case')
        print('L2u rate', -np.log2(currentL2u / formerL2u))
        print('L2p rate', -np.log2(currentL2p / formerL2p))
    formerL2p = L2p
    formerL2u = L2u
```

```
2^-2 case
L2u rate 0.18787386562449362
```

```
L2p rate 0.7046441015927761
2^-3 case
L2u rate 1.4658187575037458
L2p rate 0.9262013358519015
2^-4 case
L2u rate 1.7393264813889655
L2p rate 1.051941171228305
2^-5 case
L2u rate 1.8985120438854328
L2p rate 1.0680058302460897
2^-6 case
L2u rate 1.9651255941767054
L2p rate 1.038799014444043
2^-7 case
L2u rate 1.9900983167360333
L2p rate 1.0159843850208907
```