Try_Perturb

October 17, 2020

1 Solving a Fourth Order Elliptic Singular Perturbation Problem

$$\begin{cases} \varepsilon^2\Delta^2 u - \Delta u = f & \text{in } \Omega \\ u = \partial_n u = 0 & \text{on } \partial\Omega \end{cases}$$

[2]:
```
from skfem import *
import numpy as np
from skfem.visuals.matplotlib import draw, plot
from skfem.utils import solver_iter_krylov
from skfem.helpers import dd, ddot, grad
from scipy.sparse.linalg import LinearOperator, minres
from skfem import *
from skfem.models.poisson import *
from skfem.assembly import BilinearForm, LinearForm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
plt.rcParams['figure.dpi'] = 100
```

1.1 Problem 1

The modified Morley-Wang-Xu element method is equivalent to finding $w_h \in W_h$ and $u_{h0} \in V_{h0}$ such that

$$\begin{aligned} (\nabla w_h, \nabla \chi_h) &= (f, \chi_h) & \forall \chi_h \in W_h \\ \varepsilon^2 a_h(u_{h0}, v_h) + b_h(u_{h0}, v_h) &= (\nabla w_h, \nabla_h v_h) & \forall v_h \in V_{h0} \end{aligned}$$

where

$$a_h(u_{h0}, v_h) := \left(\nabla_h^2 u_{h0}, \nabla_h^2 v_h\right), \quad b_h(u_{h0}, v_h) := \left(\nabla_h u_{h0}, \nabla_h v_h\right)$$

Using example

$$u(x_1, x_2) = \left(\sin(\pi x_1)\sin(\pi x_2)\right)^2$$

1.1.1 Setting ϵ and generating mesh

[36]:
```
m = MeshTri.init_symmetric()
# m = MeshTri()
m.refine(5)
element = {'w': ElementTriP1(), 'u': ElementTriMorley()}
```

```
basis = {variable: InteriorBasis(m, e, intorder=4)
    for variable, e in element.items()}  # intorder: integration order for␣
 ↪quadrature

# draw(m)
# plt.show()
```

1.1.2 Exact u

[37]:
```
def exact_u(x, y):
    return (np.sin(np.pi * x) * np.sin(np.pi * y))**2
```

1.1.3 Forms for $(\nabla w_h, \nabla \chi_h) = (f, \chi_h)$

[59]:
```
epsilon = 0
@BilinearForm
def laplace(u, v, w):
    '''
    for $(\nabla w_{h}, \nabla \chi_{h})$
    '''
    return dot(grad(u), grad(v))


@LinearForm
def f_load(v, w):
    '''
    for $(f, x_{h})$
    '''
    pix = np.pi * w.x[0]
    piy = np.pi * w.x[1]
    lu = 2 * (np.pi)**2 * (np.cos(2*pix)*((np.sin(piy))**2) + np.cos(2*piy)*((np.
 ↪sin(pix))**2))
    llu = - 8 * (np.pi)**4 * (np.cos(2*pix)*np.sin(piy)**2 + np.cos(2*piy)*np.
 ↪sin(pix)**2 - np.cos(2*pix)*np.cos(2*piy))
    return (epsilon**2 * llu - lu) * v
```

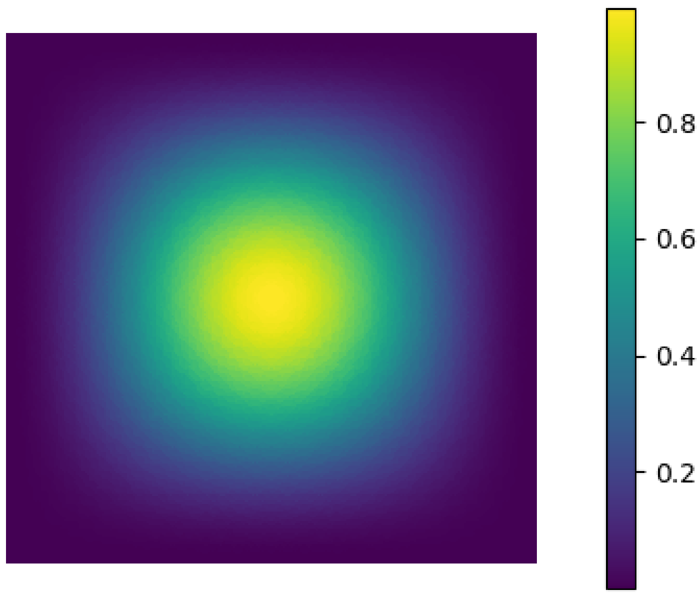
1.1.4 Solving w_h

[60]:
```
%%time

K1 = asm(laplace, basis['w'])
f1 = asm(f_load, basis['w'])

wh = solve(*condense(K1, f1, D=m.boundary_nodes()),␣
 ↪solver=solver_iter_krylov(Precondition=True))
```

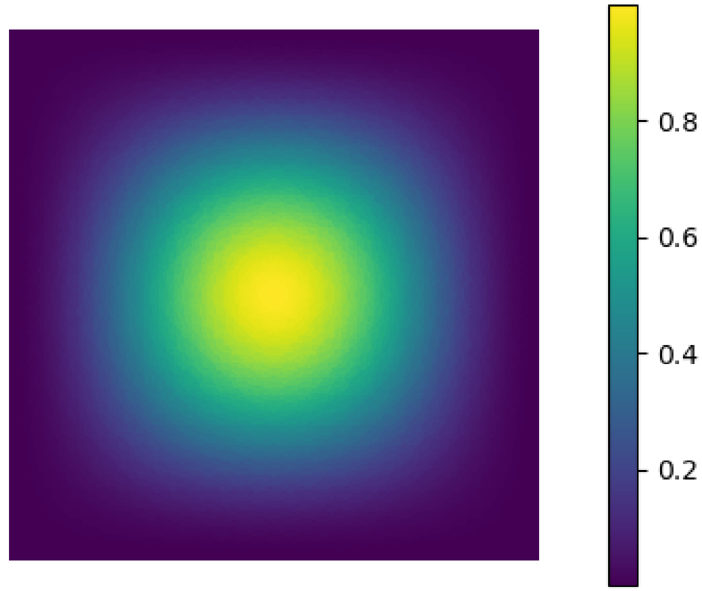
```
build_pc_diag(A) enabled
Wall time: 23.8 ms
```

[61]:
```python
def exact_w(x, y):
    pix = np.pi * x
    piy = np.pi * y
    lu = 2*(np.pi)**2 * (np.cos(2*pix)*np.sin(piy)**2 + np.cos(2*piy)*np.
 ↪sin(pix)**2)
    u = (np.sin(pix) * np.sin(piy))**2
    return - (epsilon**2 * lu - u)
```

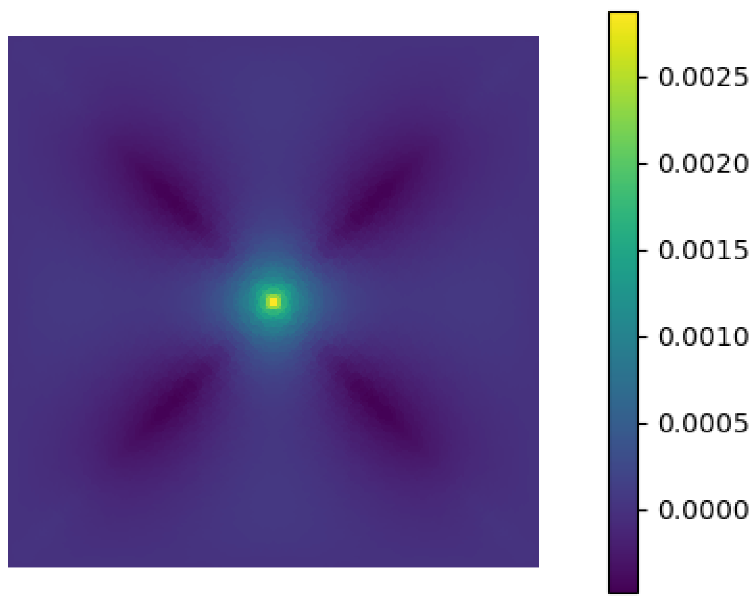
[62]:
```python
plot(basis['w'], wh, colorbar=True)
plt.show()
```



[63]:
```python
w = exact_w(basis['w'].doflocs[0], basis['w'].doflocs[1])

plot(basis['w'], w, colorbar=True)
plt.show()
```



[64]:
```
plot(basis['w'], w - wh, colorbar=True)
plt.show()
```



1.1.5 Forms for $\varepsilon^2 a_h\left(u_{h0}, v_h\right) + b_h\left(u_{h0}, v_h\right) = \left(\nabla w_h, \nabla_h v_h\right)$

$$a_h\left(u_{h0}, v_h\right) := \left(\nabla_h^2 u_{h0}, \nabla_h^2 v_h\right), \quad b_h\left(u_{h0}, v_h\right) := \left(\nabla_h u_{h0}, \nabla_h v_h\right)$$

[65]:
```python
@BilinearForm
def a_load(u, v, w):
    '''
    for $a_{h}$
    '''
    return ddot(dd(u), dd(v))


@BilinearForm
def b_load(u, v, w):
    '''
    for $b_{h}$
    '''
    return dot(grad(u), grad(v))


@BilinearForm
def wv_load(u, v, w):
    '''
    for $(\nabla \chi_{h}, \nabla_{h} v_{h})$
    '''
    return dot(grad(u), grad(v))
```

1.1.6 Setting boundary conditions

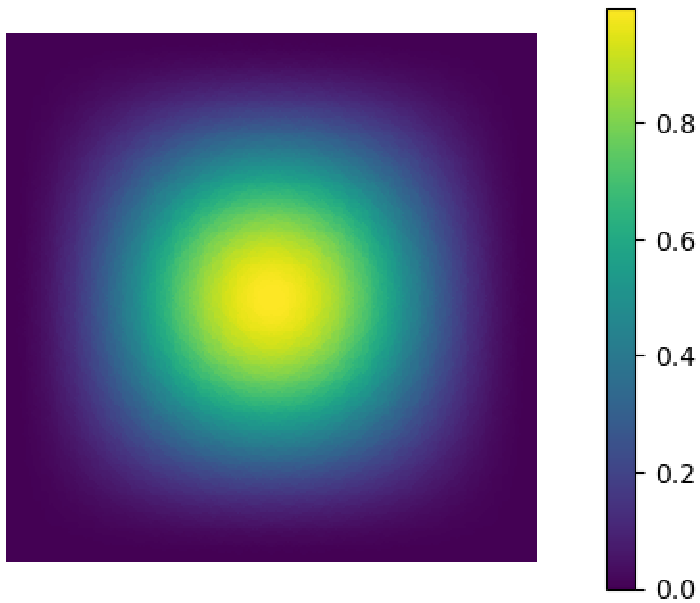
[67]:
```python
def easy_boundary(basis):
    '''
    Input basis
    ----------------
    Return D for boundary conditions
    '''

    dofs = basis.find_dofs({
        'left': m.facets_satisfying(lambda x: x[0] == 0),
        'right': m.facets_satisfying(lambda x: x[0] == 1),
        'top': m.facets_satisfying(lambda x: x[1] == 1),
        'buttom': m.facets_satisfying(lambda x: x[1] == 0)
    })

    D = np.concatenate((dofs['left'].nodal['u'], dofs['right'].nodal['u'],
                        dofs['top'].nodal['u'], dofs['buttom'].nodal['u'],
                        dofs['left'].facet['u_n'], dofs['right'].facet['u_n'],
                        dofs['top'].facet['u_n'], dofs['buttom'].facet['u_n']))
    return D
```

1.1.7 Solving u_{h0}

[68]:
```
%%time

D = easy_boundary(basis['u'])
K2 = epsilon**2 * asm(a_load, basis['u']) + asm(b_load, basis['u'])
f2 = asm(wv_load, basis['w'], basis['u']) * wh
uh0 = solve(*condense(K2, f2, D=D), solver=solver_iter_krylov(Precondition=True))
```

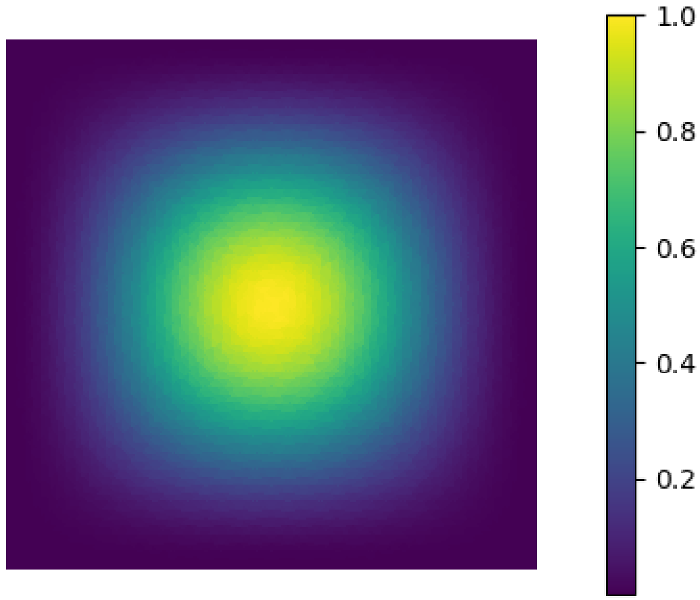
```
build_pc_diag(A) enabled
Wall time: 74.8 ms
```

[69]:
```
plot(basis['u'], uh0, colorbar=True)
plt.show()
```



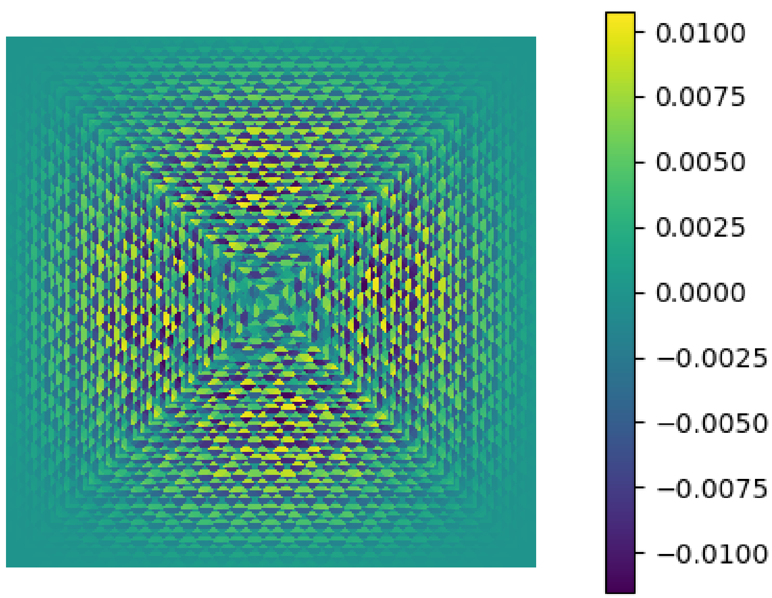
1.1.8 Showing exact u

[70]:
```
u = exact_u(basis['u'].doflocs[0], basis['u'].doflocs[1])

plot(basis['u'], u, colorbar=True)
plt.show()
```



1.1.9 Visualizing error $u - u_{h0}$

[72]:
```
plot(basis['u'], u-uh0, colorbar=True)
plt.show()
```



1.1.10 Computing L_2 error with u_{h0} and u

[77]:
```
@Functional
def L2uError(w):
    x, y = w.x
    return (w.w - exact_u(x, y))**2
```

[78]:
```
U = basis['u'].interpolate(uh0).value

L2u = np.sqrt(L2uError.assemble(basis['u'], w=U))
print('L2 error of uh0:', L2u)
```

```
L2 error of uh0: 0.0005506239320336788
```

Convergence with

$$\epsilon = 1$$

[79]:
```
epsilon = 1

currentL2u = 1
formerL2u = 1
m = MeshTri()

for i in range(1, 6):
    m.refine()

    element = {'w': ElementTriP1(), 'u': ElementTriMorley()}
    basis = {variable: InteriorBasis(m, e, intorder=4)
        for variable, e in element.items()}  # intorder: integration order for␣
↪quadrature

    K1 = asm(laplace, basis['w'])
    f1 = asm(f_load, basis['w'])

    wh = solve(*condense(K1, f1, D=m.boundary_nodes()),␣
↪solver=solver_iter_krylov(Precondition=True))

    D = easy_boundary(basis['u'])
    K2 = epsilon**2 * asm(a_load, basis['u']) + asm(b_load, basis['u'])
    f2 = asm(wv_load, basis['w'], basis['u']) * wh
    uh0 = solve(*condense(K2, f2, D=D),␣
↪solver=solver_iter_krylov(Precondition=True))

    U = basis['u'].interpolate(uh0).value

    L2u = np.sqrt(L2uError.assemble(basis['u'], w=U))
    print('case 2^-' + str(i))
    print('L2 error of uh0:', L2u)
```

```
    currentL2u = L2u
    if i != 1:
        print('rate', -np.log2(currentL2u/formerL2u))
    formerL2u = L2u
```

```
build_pc_diag(A) enabled
build_pc_diag(A) enabled
case 2^-1
L2 error of uh0: 0.14312157458419236
build_pc_diag(A) enabled
build_pc_diag(A) enabled
case 2^-2
L2 error of uh0: 0.041409209806485096
rate 1.7892175869478015
build_pc_diag(A) enabled
build_pc_diag(A) enabled
case 2^-3
L2 error of uh0: 0.00905868775220735
rate 2.192577691947967
build_pc_diag(A) enabled
build_pc_diag(A) enabled
case 2^-4
L2 error of uh0: 0.0020300577442618198
rate 2.1577813106674504
build_pc_diag(A) enabled
build_pc_diag(A) enabled
case 2^-5
L2 error of uh0: 0.0004859434175848888
rate 2.0626605211396885
```

Convergence with

$$\epsilon = 10^{-6}$$

[81]:
```
epsilon = 1e-6

currentL2u = 1
formerL2u = 1
m = MeshTri()

for i in range(1, 6):
    m.refine()

    element = {'w': ElementTriP1(), 'u': ElementTriMorley()}
    basis = {variable: InteriorBasis(m, e, intorder=4)
        for variable, e in element.items()}  # intorder: integration order for␣
 ↪quadrature
```

```python
    K1 = asm(laplace, basis['w'])
    f1 = asm(f_load, basis['w'])

    wh = solve(*condense(K1, f1, D=m.boundary_nodes()),␣
 ↪solver=solver_iter_krylov(Precondition=True))

    D = easy_boundary(basis['u'])
    K2 = epsilon**2 * asm(a_load, basis['u']) + asm(b_load, basis['u'])
    f2 = asm(wv_load, basis['w'], basis['u']) * wh
    uh0 = solve(*condense(K2, f2, D=D),␣
 ↪solver=solver_iter_krylov(Precondition=True))

    U = basis['u'].interpolate(uh0).value

    L2u = np.sqrt(L2uError.assemble(basis['u'], w=U))
    print('case 2^-' + str(i))
    print('L2 error of uh0:', L2u)
    currentL2u = L2u
    if i != 1:
        print('rate', -np.log2(currentL2u/formerL2u))
    formerL2u = L2u
```

```
build_pc_diag(A) enabled
build_pc_diag(A) enabled
case 2^-1
L2 error of uh0: 0.1589487110205496
build_pc_diag(A) enabled
build_pc_diag(A) enabled
case 2^-2
L2 error of uh0: 0.058990938135133085
rate 1.4299960594294439
build_pc_diag(A) enabled
build_pc_diag(A) enabled
case 2^-3
L2 error of uh0: 0.01202629748858373
rate 2.294300801260063
build_pc_diag(A) enabled
build_pc_diag(A) enabled
case 2^-4
L2 error of uh0: 0.0024264810981080463
rate 2.309255024423518
build_pc_diag(A) enabled
build_pc_diag(A) enabled
case 2^-5
L2 error of uh0: 0.0005506239402151528
rate 2.1397263793908747
```