Try_Perturb_Problem2

October 18, 2020

1 Solving a Fourth Order Elliptic Singular Perturbation Problem

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

[1]:
```python
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 d, dd, ddd, dot, ddot, grad, dddot, prod
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

pi = np.pi
sin = np.sin
cos = np.cos
exp = np.exp
```

1.1 Problem 2

The modified Morley-Wang-Xu element method is also equivalent to

$$\begin{aligned} (\nabla w_h, \nabla \chi_h) &= (f, \chi_h) & \forall \chi_h \in W_h \\ \varepsilon^2 \tilde{a}_h(u_h, v_h) + b_h(u_h, v_h) &= (\nabla w_h, \nabla_h v_h) & \forall v_h \in V_h \end{aligned}$$

where

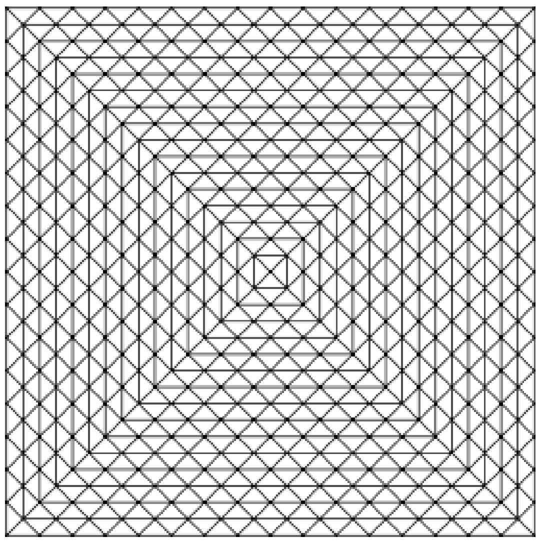
$$\tilde{a}_h(u_h, v_h) := (\nabla_h^2 u_h, \nabla_h^2 v_h) - \sum_{F \in \mathcal{F}_h^\partial} (\partial_{nn}^2 u_h, \partial_n v_h)_F - \sum_{F \in \mathcal{F}_h^\partial} (\partial_n u_h, \partial_{nn}^2 v_h)_F + \sum_{F \in \mathcal{F}_h^0} \frac{\sigma}{h_F} (\partial_n u_h, \partial_n v_h)_F$$

1.1.1 Setting ϵ and generating mesh

[2]:
```
epsilon = 0

# m = MeshTri.init_symmetric()
m = MeshTri()
m.refine(4)
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 Forms for $(\nabla w_h, \nabla \chi_h) = (f, \chi_h)$

[3]:
```
@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 = pi * w.x[0]
    piy = pi * w.x[1]
    lu = 2 * (pi)**2 * (cos(2*pix)*((sin(piy))**2) + cos(2*piy)*((sin(pix))**2))
    llu = - 8 * (pi)**4 * (cos(2*pix)*sin(piy)**2 + cos(2*piy)*sin(pix)**2 -␣
 ↪cos(2*pix)*cos(2*piy))
    return (epsilon**2 * llu - lu) * v
```

1.1.3 Solving w_h

[4]:
```
%%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))
```

```
Wall time: 13.5 ms
```

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

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

[5]:
```
@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))
```

1.1.5 Setting boundary conditions

[6]:
```
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
```

1.1.6 Adding penalty

[7]:
```
sigma = 5
gamma1 = 1e-3
gamma2 = gamma1

def divM(T): # input K
    return np.array([T[0, 0, 0] + T[0, 1, 1], T[1, 0, 0] + T[1, 1, 1]])

def gradM(T): # input K
    return np.array([[[T[0, 0, 0] , T[0, 0, 1]],
                      [T[0, 1, 0], T[0, 1, 1]]],
                     [[T[1, 0, 0], T[1, 0, 1]],
                      [T[1, 1, 0], T[1, 1, 1]]]])

def Mnn(ddu, n):
    return ddot(-ddu, prod(n, n))

@BilinearForm
def penalty_1(u, v, w):
    return gamma1 * ddot(-dd(u), prod(w.n, w.n)) * dot(grad(v), w.n)
```

```python
@BilinearForm
def penalty_2(u, v, w):
    return gamma2 * ddot(-dd(v), prod(w.n, w.n)) * dot(grad(u), w.n)

@BilinearForm
def penalty_3(u, v, w):
    return (sigma/w.h)*dot(grad(u), w.n)*dot(grad(v), w.n)
```

1.1.7 Solving u_{h0}

[8]:
```python
%%time

fbasis = FacetBasis(m, element['u'])
p1 = asm(penalty_1, fbasis)
p2 = asm(penalty_2, fbasis)
p3 = asm(penalty_3, fbasis)
P = -p1 - p2 + p3

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 + P, f2, D=D),
            solver=solver_iter_krylov(Precondition=True))  # cg
```

```
Wall time: 52 ms
```

1.1.8 Computing L_2 H_1 H_2 error with u_{h0} and u

[9]:
```python
def exact_u(x, y):
    return (sin(pi * x) * sin(pi * y))**2

def dexact_u(x, y):
    dux = 2 * pi * cos(pi * x) * sin(pi * x) * sin(pi * y)**2
    duy = 2 * pi * cos(pi * y) * sin(pi * x)**2 * sin(pi * y)
    return dux, duy

def ddexact(x, y):
    duxx = 2*pi**2*cos(pi*x)**2*sin(pi*y)**2 - 2*pi**2*sin(pi*x)**2*sin(pi*y)**2
    duxy = 2*pi*cos(pi*x)*sin(pi*x)*2*pi*cos(pi*y)*sin(pi*y)
    duyx = duxy
    duyy = 2*pi**2*cos(pi*y)**2*sin(pi*x)**2 - 2*pi**2*sin(pi*y)**2*sin(pi*x)**2
    return duxx, duxy, duyx, duyy

@Functional
def L2uError(w):
    x, y = w.x
```

```
    return (w.w - exact_u(x, y))**2

def get_DuError(basis, u):
    duh = basis.interpolate(u).grad
    x = basis.global_coordinates().value
    dx = basis.dx  # quadrature weights
    dux, duy = dexact_u(x[0], x[1])
    return np.sqrt(np.sum(((duh[0] - dux)**2 + (duh[1] - duy)**2) * dx))

def get_D2uError(basis, u):
    dduh = basis.interpolate(u).hess
    x = basis.global_coordinates().value  # coordinates of quadrature points [x,␣
 ↪y]
    dx = basis.dx  # quadrature weights
    duxx, duxy, duyx, duyy = ddexact(x[0], x[1])
    return np.sqrt(
        np.sum(((dduh[0][0] - duxx)**2 + (dduh[0][1] - duxy)**2 +
                (dduh[1][1] - duyy)**2 + (dduh[1][0] - duyx)**2) * dx))
```

2 Numerical results

2.1 Parameters

$$\tilde{a}_h\left(u_h, v_h\right) := \left(\nabla_h^2 u_h, \nabla_h^2 v_h\right) - \sum_{F \in \mathcal{F}_h^{\partial}} \left(\partial_{nn}^2 u_h, \partial_n v_h\right)_F - \sum_{F \in \mathcal{F}_h^{\partial}} \left(\partial_n u_h, \partial_{nn}^2 v_h\right)_F + \sum_{F \in \mathcal{F}_h^0} \frac{\sigma}{h_F} \left(\partial_n u_h, \partial_n v_h\right)_F$$

- gamma1 times $\sum_{F \in \mathcal{F}_h^{\partial}} \left(\partial_{nn}^2 u_h, \partial_n v_h\right)_F$
- gamma2 times $\sum_{F \in \mathcal{F}_h^{\partial}} \left(\partial_n u_h, \partial_{nn}^2 v_h\right)_F$
- sigma in $\sum_{F \in \mathcal{F}_h^0} \frac{\sigma}{h_F} \left(\partial_n u_h, \partial_n v_h\right)_F$

[10]:
```
gamma1 = 1
gamma2 = 1
sigma = 5
```

2.2 Example 1

Using example

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

[11]:
```
for i in range(6):
    epsilon = 1*10**(-i)

    L2_list = []
    Du_list = []
    D2u_list = []
    h_list = []
```

```python
epu_list = []
m = MeshTri()

for i in range(1, 7):
    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))


    fbasis = FacetBasis(m, element['u'])
    p1 = asm(penalty_1, fbasis)
    p2 = asm(penalty_2, fbasis)
    p3 = asm(penalty_3, fbasis)
    P = -p1 - p2 + p3

    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 + P, f2, D=D), solver=solver_iter_krylov(Precondition=True))  # cg

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

    L2u = np.sqrt(L2uError.assemble(basis['u'], w=U))
    Du = get_DuError(basis['u'], uh0)
    H1u = Du + L2u
    D2u = get_D2uError(basis['u'], uh0)
    H2u = Du + L2u + D2u
    epu = np.sqrt(epsilon**2 * D2u**2 + Du**2)
    h_list.append(m.param())
    Du_list.append(Du)
    L2_list.append(L2u)
    D2u_list.append(D2u)
    epu_list.append(epu)

hs = np.array(h_list)
L2s = np.array(L2_list)
Dus = np.array(Du_list)
```

```python
    D2us = np.array(D2u_list)
    epus = np.array(epu_list)
    H1s = L2s + Dus
    H2s = H1s + D2us
    print('epsilon =', epsilon)
    print('  h     L2u    H1u    H2u    epu')
    for i in range(H2s.shape[0] - 1):
        print(
            '2^-' + str(i + 2),
            ' {:.2f}  {:.2f}  {:.2f}  {:.2f}'.format(-np.log2(L2s[i + 1] /␣
↪L2s[i]),
                                                    -np.log2(H1s[i + 1] /␣
↪H1s[i]),
                                                    -np.log2(H2s[i + 1] /␣
↪H2s[i]),
                                                    -np.log2(epus[i + 1] /␣
↪epus[i])))
```

```
epsilon = 1
  h     L2u    H1u    H2u    epu
2^-2  1.34  0.84  0.69  0.68
2^-3  2.24  1.79  1.04  0.99
2^-4  2.24  1.95  1.04  1.01
2^-5  2.12  1.99  1.02  1.00
2^-6  2.04  2.00  1.01  1.00
epsilon = 0.1
  h     L2u    H1u    H2u    epu
2^-2  1.37  1.13  0.72  0.90
2^-3  2.64  2.04  1.13  1.36
2^-4  2.42  2.03  1.11  1.19
2^-5  2.24  2.03  1.05  1.07
2^-6  2.10  2.02  1.02  1.02
epsilon = 0.01
  h     L2u    H1u    H2u    epu
2^-2  1.65  1.21  0.55  1.13
2^-3  3.11  1.92  0.89  1.78
2^-4  2.23  1.97  1.15  1.87
2^-5  1.75  2.03  1.26  1.85
2^-6  1.81  2.02  1.16  1.59
epsilon = 0.001
  h     L2u    H1u    H2u    epu
2^-2  1.66  1.21  0.55  1.13
2^-3  3.16  1.90  0.85  1.79
2^-4  2.45  1.91  1.00  1.89
2^-5  2.10  1.97  1.02  1.96
2^-6  1.94  2.00  1.05  1.99
epsilon = 0.0001
```

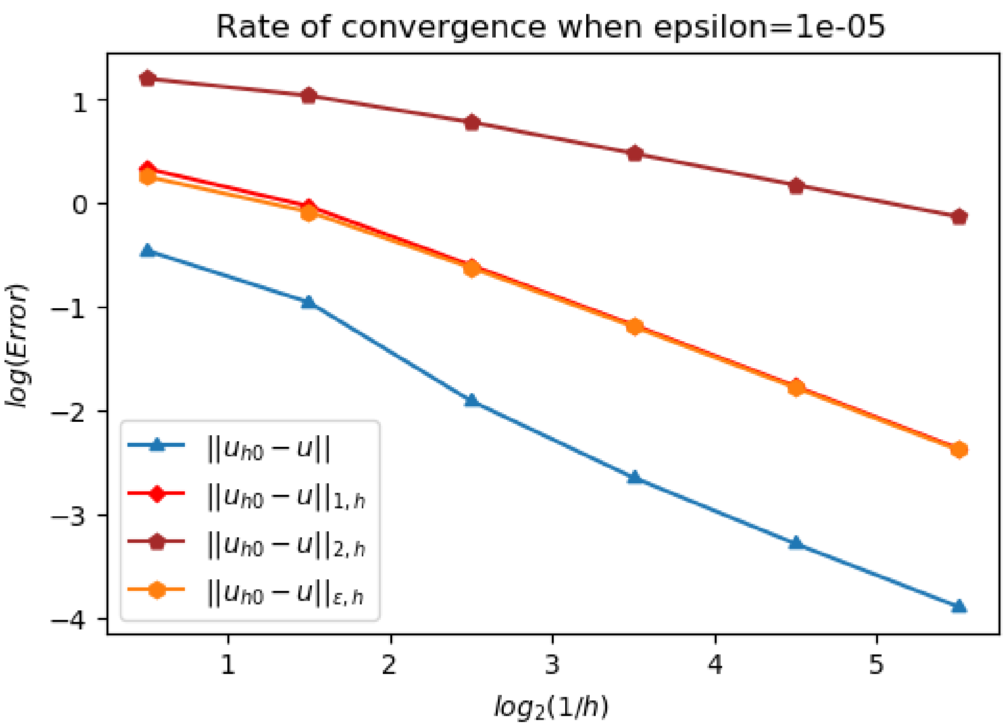
```
  h     L2u    H1u    H2u    epu
2^-2   1.66   1.21   0.55   1.13
2^-3   3.16   1.90   0.85   1.79
2^-4   2.46   1.91   1.00   1.89
2^-5   2.12   1.97   1.01   1.96
2^-6   2.01   1.99   1.01   1.99
epsilon = 1e-05
  h     L2u    H1u    H2u    epu
2^-2   1.66   1.21   0.55   1.13
2^-3   3.16   1.90   0.85   1.79
2^-4   2.46   1.91   1.00   1.89
2^-5   2.12   1.97   1.01   1.96
2^-6   2.01   1.99   1.01   1.99
```

[12]:

```
hs_Log = np.log2(hs)

L2plot, = plt.plot(-hs_Log,
                   np.log10(L2s),
                   marker=(3, 0),
                   label='$|\|u_{h0}-u\||}$')
H1plot, = plt.plot(-hs_Log,
                   np.log10(H1s),
                   marker=(4, 0),
                   label=r'$|\left\|{u}_{h0}-u\right\||_{1, h}$',
                   color='red')
H2plot, = plt.plot(-hs_Log,
                   np.log10(H2s),
                   marker=(5, 0),
                   label=r'$|\left\|{u}_{h0}-u\right\||_{2, h}$',
                   color='brown')
epplot, = plt.plot(-hs_Log,
                   np.log10(epus),
                   marker=(6, 0),
                   label='$\|\|{u}_{h0}-u\|\|_{\epsilon, h}$')

plt.legend(handles=[L2plot, H1plot, H2plot, epplot])
plt.title('Rate of convergence when epsilon='+str(epsilon))
plt.xlabel('$log_{2}(1/h)$')
plt.ylabel('$log(Error)$')
plt.show()
```



2.3 Example 2

$$u = g(x)p(y)$$

where

$$g(x) = \frac{1}{2}\left[\sin(\pi x) + \frac{\pi\varepsilon}{1 - e^{-1/\varepsilon}}\left(e^{-x/\varepsilon} + e^{(x-1)/\varepsilon} - 1 - e^{-1/\varepsilon}\right)\right]$$

$$p(y) = 2y\left(1 - y^2\right) + \varepsilon\left[ld(1 - 2y) - 3\frac{q}{l} + \left(\frac{3}{l} - d\right)e^{-y/\varepsilon} + \left(\frac{3}{l} + d\right)e^{(y-1)/\varepsilon}\right]$$

$$l = 1 - e^{-1/\varepsilon}, q = 2 - l \text{ and } d = 1/(q - 2\varepsilon l)$$

[13]:
```
@LinearForm
def f_load(v, w):
    '''
    for $(f, x_{h})$
    '''
    x = w.x[0]
    y = w.x[1]
```

```
    return ((sin(pi*x)/2 - (ep*pi*(exp(-x/ep) + exp((x - 1)/ep) - exp(-1/ep) -␣
↪1))/(2*(exp(-1/ep) - 1)))*(12*y + ep*((exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/
↪(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)))/ep**2 + (exp((y - 1)/ep)*(3/(exp(-1/
↪ep) - 1) - 1/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)))/ep**2)) -␣
↪((pi**2*sin(pi*x))/2 + (ep*pi*(exp(-x/ep)/ep**2 + exp((x - 1)/ep)/ep**2))/
↪(2*(exp(-1/ep) - 1)))*(ep*(exp((y - 1)/ep)*(3/(exp(-1/ep) - 1) - 1/(exp(-1/ep)␣
↪+ 2*ep*(exp(-1/ep) - 1) + 1)) + exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/(exp(-1/ep)␣
↪+ 2*ep*(exp(-1/ep) - 1) + 1)) - (3*exp(-1/ep) + 3)/(exp(-1/ep) - 1) - ((2*y -␣
↪1)*(exp(-1/ep) - 1))/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)) + 2*y*(y**2 -␣
↪1)) - ep**2*(((pi**4*sin(pi*x))/2 - (ep*pi*(exp(-x/ep)/ep**4 + exp((x - 1)/ep)/
↪ep**4))/(2*(exp(-1/ep) - 1)))*(ep*(exp((y - 1)/ep)*(3/(exp(-1/ep) - 1) - 1/
↪(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)) + exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/
↪(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)) - (3*exp(-1/ep) + 3)/(exp(-1/ep) -␣
↪1) - ((2*y - 1)*(exp(-1/ep) - 1))/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)) +␣
↪2*y*(y**2 - 1)) - 2*(12*y + ep*((exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/(exp(-1/
↪ep) + 2*ep*(exp(-1/ep) - 1) + 1)))/ep**2 + (exp((y - 1)/ep)*(3/(exp(-1/ep) -␣
↪1) - 1/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)))/ep**2))*((pi**2*sin(pi*x))/2␣
↪+ (ep*pi*(exp(-x/ep)/ep**2 + exp((x - 1)/ep)/ep**2))/(2*(exp(-1/ep) - 1))) +␣
↪ep*(sin(pi*x)/2 - (ep*pi*(exp(-x/ep) + exp((x - 1)/ep) - exp(-1/ep) - 1))/
↪(2*(exp(-1/ep) - 1)))*((exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/(exp(-1/ep) +␣
↪2*ep*(exp(-1/ep) - 1) + 1)))/ep**4 + (exp((y - 1)/ep)*(3/(exp(-1/ep) - 1) - 1/
↪(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)))/ep**4))) * v


def exact_u(x, y):
    return -(sin(pi*x)/2 - (ep*pi*(exp(-x/ep) + exp((x - 1)/ep) - exp(-1/ep) -␣
↪1))/(2*(exp(-1/ep) - 1)))*(ep*(exp((y - 1)/ep)*(3/(exp(-1/ep) - 1) - 1/(exp(-1/
↪ep) + 2*ep*(exp(-1/ep) - 1) + 1)) + exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/(exp(-1/
↪ep) + 2*ep*(exp(-1/ep) - 1) + 1)) - (3*exp(-1/ep) + 3)/(exp(-1/ep) - 1) -␣
↪((2*y - 1)*(exp(-1/ep) - 1))/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)) +␣
↪2*y*(y**2 - 1))


def dexact_u(x, y):
    dux = -((pi*cos(pi*x))/2 + (ep*pi*(exp(-x/ep)/ep - exp((x - 1)/ep)/ep))/
↪(2*(exp(-1/ep) - 1)))*(ep*(exp((y - 1)/ep)*(3/(exp(-1/ep) - 1) - 1/(exp(-1/ep)␣
↪+ 2*ep*(exp(-1/ep) - 1) + 1)) + exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/(exp(-1/ep)␣
↪+ 2*ep*(exp(-1/ep) - 1) + 1)) - (3*exp(-1/ep) + 3)/(exp(-1/ep) - 1) - ((2*y -␣
↪1)*(exp(-1/ep) - 1))/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)) + 2*y*(y**2 -␣
↪1))
    duy = (sin(pi*x)/2 - (ep*pi*(exp(-x/ep) + exp((x - 1)/ep) - exp(-1/ep) - 1))/
↪(2*(exp(-1/ep) - 1)))*(ep*((2*(exp(-1/ep) - 1))/(exp(-1/ep) + 2*ep*(exp(-1/ep)␣
↪- 1) + 1) + (exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/(exp(-1/ep) + 2*ep*(exp(-1/ep)␣
↪- 1) + 1)))/ep - (exp((y - 1)/ep)*(3/(exp(-1/ep) - 1) - 1/(exp(-1/ep) +␣
↪2*ep*(exp(-1/ep) - 1) + 1)))/ep) - 6*y**2 + 2)
    return dux, duy
```

```python
def ddexact(x, y):
    duxx = ((pi**2*sin(pi*x))/2 + (ep*pi*(exp(-x/ep)/ep**2 + exp((x - 1)/ep)/ep**2))/(2*(exp(-1/ep) - 1)))*(ep*(exp((y - 1)/ep)*(3/(exp(-1/ep) - 1) - 1/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)) + exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)) - (3*exp(-1/ep) + 3)/(exp(-1/ep) - 1) - ((2*y - 1)*(exp(-1/ep) - 1))/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)) + 2*y*(y**2 - 1))
    duxy = ((pi*cos(pi*x))/2 + (ep*pi*(exp(-x/ep)/ep - exp((x - 1)/ep)/ep))/(2*(exp(-1/ep) - 1)))*(ep*((2*(exp(-1/ep) - 1))/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1) + (exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)))/ep - (exp((y - 1)/ep)*(3/(exp(-1/ep) - 1) - 1/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)))/ep) - 6*y**2 + 2)
    duyx = duxy
    duyy = -(sin(pi*x)/2 - (ep*pi*(exp(-x/ep) + exp((x - 1)/ep) - exp(-1/ep) - 1))/(2*(exp(-1/ep) - 1)))*(12*y + ep*((exp(-y/ep)*(3/(exp(-1/ep) - 1) + 1/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)))/ep**2 + (exp((y - 1)/ep)*(3/(exp(-1/ep) - 1) - 1/(exp(-1/ep) + 2*ep*(exp(-1/ep) - 1) + 1)))/ep**2))
    return duxx, duxy, duyx, duyy


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


def get_DuError(basis, u):
    duh = basis.interpolate(u).grad
    x = basis.global_coordinates().value
    dx = basis.dx  # quadrature weights
    dux, duy = dexact_u(x[0], x[1])
    return np.sqrt(np.sum(((duh[0] - dux)**2 + (duh[1] - duy)**2) * dx))


def get_D2uError(basis, u):
    dduh = basis.interpolate(u).hess
    x = basis.global_coordinates(
    ).value  # coordinates of quadrature points [x, y]
    dx = basis.dx  # quadrature weights
    duxx, duxy, duyx, duyy = ddexact(x[0], x[1])
    return np.sqrt(
        np.sum(((dduh[0][0] - duxx)**2 + (dduh[0][1] - duxy)**2 +
                (dduh[1][1] - duyy)**2 + (dduh[1][0] - duyx)**2) * dx))
```

[14]:
```
for i in range(6):
    epsilon = 1 * 10**(-i)
    ep = epsilon
    L2_list = []
    Du_list = []
    D2u_list = []
    h_list = []
    epu_list = []
    m = MeshTri()

    for i in range(1, 7):
        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))

        fbasis = FacetBasis(m, element['u'])
        p1 = asm(penalty_1, fbasis)
        p2 = asm(penalty_2, fbasis)
        p3 = asm(penalty_3, fbasis)
        P = -p1 - p2 + p3

        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 + P, f2, D=D), solver=solver_iter_krylov(Precondition=True))  # cg

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

        L2u = np.sqrt(L2uError.assemble(basis['u'], w=U))
        Du = get_DuError(basis['u'], uh0)
        H1u = Du + L2u
        D2u = get_D2uError(basis['u'], uh0)
        H2u = Du + L2u + D2u
        epu = np.sqrt(epsilon**2 * D2u**2 + Du**2)
        h_list.append(m.param())
        Du_list.append(Du)
```

```
        L2_list.append(L2u)
        D2u_list.append(D2u)
        epu_list.append(epu)

    hs = np.array(h_list)
    L2s = np.array(L2_list)
    Dus = np.array(Du_list)
    D2us = np.array(D2u_list)
    epus = np.array(epu_list)
    H1s = L2s + Dus
    H2s = H1s + D2us
    print('epsilon =', ep)
    print('  h    L2u   H1u   H2u   epu')
    for i in range(H2s.shape[0] - 1):
        print(
            '2^-' + str(i + 2), ' {:.2f}  {:.2f}  {:.2f}  {:.2f}'.format(
                -np.log2(L2s[i + 1] / L2s[i]), -np.log2(H1s[i + 1] / H1s[i]),
                -np.log2(H2s[i + 1] / H2s[i]),
                -np.log2(epus[i + 1] / epus[i])))
```

```
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  1.33  1.23  0.73  0.69
2^-3  1.42  1.62  0.90  0.86
2^-4  1.67  1.78  0.95  0.93
2^-5  1.84  1.89  0.98  0.96
2^-6  1.93  1.95  0.99  0.98
epsilon = 0.1
  h    L2u   H1u   H2u   epu
2^-2  1.57  1.21  0.49  0.75
2^-3  3.13  2.03  0.83  0.99
2^-4  1.39  1.80  0.90  0.93
2^-5  1.35  1.76  0.94  0.95
2^-6  1.70  1.86  0.97  0.97
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.32  0.79  -0.21  0.70
2^-3  2.47  1.25  -0.45  1.07
2^-4  1.44  1.21  -0.18  0.83
2^-5  0.61  1.15  0.26  0.61
2^-6  1.14  1.31  0.58  0.69
epsilon = 0.001
  h    L2u   H1u   H2u   epu
2^-2  1.23  0.72  -0.30  0.63
2^-3  2.15  0.92  -0.38  0.80
2^-4  2.02  0.66  -0.37  0.61
2^-5  1.61  0.61  -0.15  0.59
```

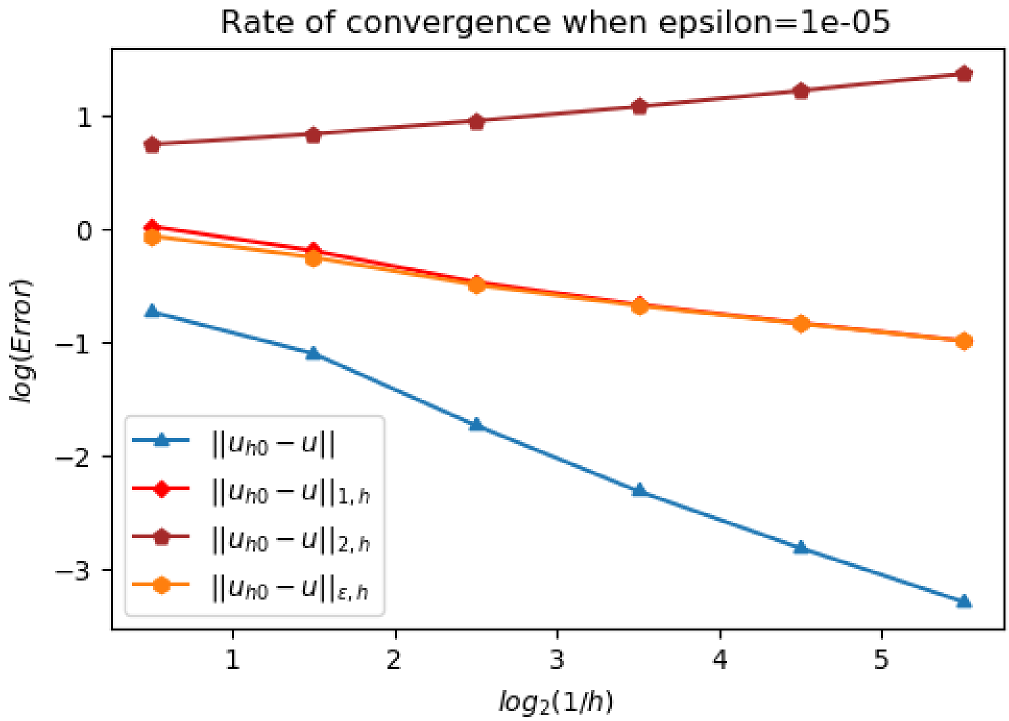
```
2^-6  0.74  0.79  -0.66  0.75
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.22  0.72  -0.31  0.63
2^-3  2.10  0.91  -0.38  0.80
2^-4  1.93  0.66  -0.41  0.61
2^-5  1.68  0.54  -0.47  0.52
2^-6  1.58  0.51  -0.49  0.50
epsilon = 1e-05
  h    L2u   H1u   H2u   epu
2^-2  1.22  0.72  -0.31  0.63
2^-3  2.10  0.91  -0.38  0.80
2^-4  1.92  0.66  -0.41  0.61
2^-5  1.67  0.54  -0.47  0.52
2^-6  1.56  0.51  -0.49  0.51
```

[15]:

```python
hs_Log = np.log2(hs)

L2plot, = plt.plot(-hs_Log,
                   np.log10(L2s),
                   marker=(3, 0),
                   label='$|\|u_{h0}-u\||}$')
H1plot, = plt.plot(-hs_Log,
                   np.log10(H1s),
                   marker=(4, 0),
                   label=r'$|\left\|{u}_{h0}-u\right\||_{1, h}$',
                   color='red')
H2plot, = plt.plot(-hs_Log,
                   np.log10(H2s),
                   marker=(5, 0),
                   label=r'$|\left\|{u}_{h0}-u\right\||_{2, h}$',
                   color='brown')
epplot, = plt.plot(-hs_Log,
                   np.log10(epus),
                   marker=(6, 0),
                   label='$\|\|{u}_{h0}-u\|\|_{\epsilon, h}$')

plt.legend(handles=[L2plot, H1plot, H2plot, epplot])
plt.title('Rate of convergence when epsilon='+str(epsilon))
plt.xlabel('$log_{2}(1/h)$')
plt.ylabel('$log(Error)$')
plt.show()
```



Rate of convergence when epsilon=1e-05
1
0
−1
−2
−3
log(Error)
1
2
3
4
5
log2(1/h)
||uh0 − u||
||uh0 − u||1, h
||uh0 − u||2, h
||uh0 − u||ε, h