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]:

```
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 Problem1

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

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

1.2 Problem2

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

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

1.3 Forms and errors

[2]:
```python
@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))


@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))
```

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


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


@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)


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

1.4 Solver for problem1

[3]:
```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)
    })
```

```python
    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


def solve_problem1(m):

    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))

    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=easy_boundary(basis['u'])),
                solver=solver_iter_krylov(Precondition=True))  # cg
    return uh0, basis
```

1.5 Solver for problem2

[4]:
```python
def easy_boundary_penalty(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']))
    return D
```

```python
def solve_problem2(m):

    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

    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=easy_boundary_penalty(basis['u'])),
                solver=solver_iter_krylov(Precondition=True))  # cg
    return uh0, basis
```

2 Numerical results

setting boundary condition: $u = 0 \; on \; \partial\Omega$

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 $\sum_{F \in \mathcal{F}_h^0} \frac{\sigma}{h_F} \left(\partial_n u_h, \partial_n v_h\right)_F$

2.2 Example 1

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

[16]:
```python
@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


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
```

2.2.1 Without penalty (Problem1)

[17]:
```python
for j in range(6):
    epsilon = 1 * 10**(-j)

    L2_list = []
    Du_list = []
    D2u_list = []
    h_list = []
    epu_list = []
    m = MeshTri()

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

```
        m.refine()
        uh0, basis = solve_problem1(m)
        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 =', 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   1.79   0.91   0.69   0.67
2^-3   2.19   1.76   1.02   0.98
2^-4   2.16   1.93   1.05   1.02
2^-5   2.06   1.98   1.02   1.01
2^-6   2.02   2.00   1.01   1.00
epsilon = 0.1
  h     L2u    H1u    H2u    epu
2^-2   1.38   0.84   0.60   0.66
2^-3   2.06   1.76   1.07   1.22
2^-4   2.06   1.93   1.08   1.15
2^-5   2.03   1.98   1.04   1.05
2^-6   2.01   2.00   1.01   1.01
```

```
epsilon = 0.01
  h     L2u   H1u   H2u   epu
2^-2  1.43  0.77  0.45  0.70
2^-3  2.26  1.67  0.86  1.61
2^-4  2.08  1.94  1.09  1.86
2^-5  1.76  2.03  1.22  1.85
2^-6  1.82  2.02  1.14  1.59
epsilon = 0.001
  h     L2u   H1u   H2u   epu
2^-2  1.43  0.77  0.45  0.70
2^-3  2.29  1.66  0.81  1.61
2^-4  2.31  1.89  0.94  1.87
2^-5  2.12  1.97  0.99  1.96
2^-6  1.98  2.00  1.03  1.99
epsilon = 0.0001
  h     L2u   H1u   H2u   epu
2^-2  1.43  0.77  0.45  0.70
2^-3  2.29  1.66  0.81  1.61
2^-4  2.31  1.89  0.94  1.87
2^-5  2.14  1.97  0.98  1.96
2^-6  2.04  1.99  1.00  1.99
epsilon = 1e-05
  h     L2u   H1u   H2u   epu
2^-2  1.43  0.77  0.45  0.70
2^-3  2.29  1.66  0.81  1.61
2^-4  2.31  1.89  0.94  1.87
2^-5  2.14  1.97  0.98  1.96
2^-6  2.04  1.99  1.00  1.99
```

2.2.2 With penalty (Problem2)

[18]:

```python
gamma1 = 10
gamma2 = 10
sigma = 50
for j in range(6):
    epsilon = 1 * 10**(-j)
    ep = epsilon
    L2_list = []
    Du_list = []
    D2u_list = []
    h_list = []
    epu_list = []
    m = MeshTri()

    for i in range(1, 8):

        m.refine()
```

```python
        uh0, basis = solve_problem2(m)
        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 =', 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  -3.32  -2.71  -2.26  -2.21
2^-3  1.22  0.64  -0.24  -0.34
2^-4  2.59  3.03  2.99  2.99
2^-5  1.26  1.31  0.94  0.91
2^-6  1.09  1.10  0.61  0.59
2^-7  1.04  1.05  0.56  0.54
epsilon = 0.1
  h    L2u   H1u   H2u   epu
2^-2  0.31  0.39  0.26  0.34
2^-3  1.35  1.32  0.67  0.94
2^-4  1.41  1.36  0.69  0.83
2^-5  1.25  1.23  0.63  0.68
2^-6  1.14  1.12  0.58  0.59
2^-7  1.07  1.06  0.54  0.55
```

```
epsilon = 0.01
  h     L2u   H1u   H2u   epu
2^-2  0.39  0.50  0.25  0.52
2^-3  1.62  1.46  0.42  1.41
2^-4  1.91  1.62  0.65  1.50
2^-5  1.91  1.58  0.71  1.36
2^-6  1.75  1.48  0.66  1.09
2^-7  1.49  1.33  0.61  0.82
epsilon = 0.001
  h     L2u   H1u   H2u   epu
2^-2  0.39  0.50  0.25  0.52
2^-3  1.62  1.46  0.39  1.42
2^-4  1.94  1.62  0.57  1.56
2^-5  1.99  1.61  0.60  1.56
2^-6  2.00  1.58  0.58  1.53
2^-7  1.99  1.55  0.58  1.47
epsilon = 0.0001
  h     L2u   H1u   H2u   epu
2^-2  0.39  0.50  0.25  0.52
2^-3  1.62  1.46  0.38  1.42
2^-4  1.94  1.62  0.57  1.57
2^-5  1.99  1.61  0.60  1.56
2^-6  2.00  1.58  0.57  1.54
2^-7  2.00  1.55  0.54  1.52
epsilon = 1e-05
  h     L2u   H1u   H2u   epu
2^-2  0.39  0.50  0.25  0.52
2^-3  1.62  1.46  0.38  1.42
2^-4  1.94  1.62  0.57  1.57
2^-5  1.99  1.61  0.60  1.56
2^-6  2.00  1.58  0.57  1.54
2^-7  2.00  1.55  0.54  1.52
```

2.3 Example 2

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

where

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

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

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

[8]:
```
@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


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
```

2.3.1 Without penalty (Problem1)

[9]:
```
for j in range(6):
    epsilon = 1 * 10**(-j)
    ep = epsilon
    L2_list = []
    Du_list = []
    D2u_list = []
    h_list = []
    epu_list = []
    m = MeshTri()

    for i in range(1, 7):

        m.refine()
        uh0, basis = solve_problem1(m)
        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 =', epsilon)
```

```python
    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  1.11  1.18  0.69  0.65
2^-3  1.56  1.57  0.89  0.86
2^-4  1.85  1.82  0.98  0.95
2^-5  1.95  1.94  1.00  0.99
2^-6  1.99  1.98  1.00  1.00
epsilon = 0.1
  h    L2u   H1u   H2u   epu
2^-2  1.39  1.23  0.46  0.65
2^-3  0.85  1.22  0.66  0.73
2^-4  1.57  1.56  0.83  0.85
2^-5  1.87  1.84  0.95  0.95
2^-6  1.96  1.96  0.99  0.99
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.89  0.86  -0.00  0.75
2^-3  1.54  0.96  -0.68  0.85
2^-4  0.53  1.03  -0.29  0.69
2^-5  0.57  1.02  0.23  0.54
2^-6  1.15  1.16  0.53  0.65
epsilon = 0.001
  h    L2u   H1u   H2u   epu
2^-2  1.69  0.75  -0.23  0.65
2^-3  1.57  0.61  -0.47  0.56
2^-4  1.54  0.54  -0.45  0.51
2^-5  1.35  0.58  -0.18  0.57
2^-6  0.64  0.79  -0.67  0.75
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.67  0.75  -0.23  0.65
2^-3  1.53  0.61  -0.47  0.56
2^-4  1.50  0.54  -0.49  0.51
2^-5  1.51  0.52  -0.49  0.50
2^-6  1.51  0.51  -0.50  0.50
epsilon = 1e-05
  h    L2u   H1u   H2u   epu
2^-2  1.66  0.75  -0.23  0.65
2^-3  1.53  0.61  -0.47  0.56
```

```
2^-4  1.50  0.54  -0.49  0.51
2^-5  1.50  0.52  -0.49  0.50
2^-6  1.51  0.51  -0.50  0.50
```

2.3.2 With penalty (Problem2)

[15]:
```
gamma1 = 10
gamma2 = 10
sigma = 50
for j in range(6):
    epsilon = 1 * 10**(-j)
    ep = epsilon
    L2_list = []
    Du_list = []
    D2u_list = []
    h_list = []
    epu_list = []
    m = MeshTri()

    for i in range(1, 8):

        m.refine()
        uh0, basis = solve_problem2(m)
        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 =', epsilon)
    print('  h     L2u    H1u    H2u    epu')
    for i in range(H2s.shape[0] - 1):
```

```python
        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  -3.73  -3.14  -2.63  -2.56
2^-3  1.43  0.86  -0.01  -0.10
2^-4  2.28  2.73  2.74  2.74
2^-5  1.18  1.20  0.76  0.73
2^-6  1.06  1.06  0.57  0.55
2^-7  1.03  1.03  0.54  0.52
epsilon = 0.1
  h     L2u   H1u   H2u   epu
2^-2  0.20  0.23  0.05  0.15
2^-3  0.93  0.84  0.29  0.49
2^-4  1.11  1.04  0.40  0.52
2^-5  1.08  1.05  0.46  0.51
2^-6  1.04  1.03  0.48  0.50
2^-7  1.02  1.02  0.49  0.50
epsilon = 0.01
  h     L2u   H1u   H2u   epu
2^-2  0.16  0.19  -0.38  0.19
2^-3  0.72  0.52  -0.77  0.45
2^-4  1.01  0.66  -0.34  0.52
2^-5  1.17  0.79  0.01  0.57
2^-6  1.22  0.94  0.20  0.59
2^-7  1.16  1.03  0.32  0.54
epsilon = 0.001
  h     L2u   H1u   H2u   epu
2^-2  0.15  0.16  -0.21  0.16
2^-3  0.67  0.43  -0.43  0.38
2^-4  0.90  0.52  -0.45  0.46
2^-5  0.98  0.55  -0.60  0.50
2^-6  1.03  0.60  -0.88  0.55
2^-7  1.09  0.65  -0.51  0.56
epsilon = 0.0001
  h     L2u   H1u   H2u   epu
2^-2  0.15  0.16  -0.21  0.16
2^-3  0.66  0.43  -0.43  0.38
2^-4  0.89  0.52  -0.45  0.46
2^-5  0.96  0.53  -0.47  0.48
2^-6  0.99  0.53  -0.48  0.49
2^-7  1.00  0.52  -0.49  0.50
epsilon = 1e-05
```

h	L2u	H1u	H2u	epu
2^-2	0.15	0.16	-0.21	0.16
2^-3	0.66	0.43	-0.43	0.38
2^-4	0.89	0.52	-0.45	0.46
2^-5	0.96	0.53	-0.47	0.48
2^-6	0.98	0.53	-0.48	0.49
2^-7	0.99	0.52	-0.49	0.50