Perturb_Problem2

October 19, 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 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'] = 200

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

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

1.3 Forms and errors

[3]:
```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 ddot(-dd(u), prod(w.n, w.n)) * dot(grad(v), w.n)

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


@BilinearForm
def penalty_3(u, v, w):
    global mem
    global nn
    global memu
    nn = prod(w.n, w.n)
    mem = w
    memu = u
    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

[70]:
```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


def solve_problem1(m):

    element = {'w': ElementTriP2(), '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, tol=1e-8))

    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

[71]:
```
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)
```

```python
    })

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


def solve_problem2(m):
    global fbasis
    element = {'w': ElementTriP2(), 'u': ElementTriMorley()}
    basis = {
        variable: InteriorBasis(m, e, intorder=4)
        for variable, e in element.items()
    }

    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']) + epsilon**2 * P + asm(b_load,␣
↪basis['u'])
    f2 = asm(wv_load, basis['w'], basis['u']) * wh
    uh0 = solve(*condense(K2, f2, D=easy_boundary_penalty(basis['u'])),␣
↪solver=solver_iter_krylov(Precondition=True))
    # uh0 = solve(*condense(K2 + P, f2, D=m.boundary_nodes()),␣
↪solver=solver_iter_krylov(Precondition=True))
    return uh0, basis
```

2 Numerical results

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

2.1 Parameters

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

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

2.2 Example 1

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

[64]:
```
@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)

[65]:
```
refine_time = 6
epsilon_range = 3
for j in range(epsilon_range):
    epsilon = 1 * 10**(-j*2)

    L2_list = []
    Du_list = []
```

```
    D2u_list = []
    h_list = []
    epu_list = []
    m = MeshTri()

    for i in range(1, refine_time+1):

        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])))
#         print(
#             '2^-' + str(i + 2), ' {:.5f}  {:.5f}  {:.5f}  {:.5f}'.format(
#                 L2s[i + 1], H1s[i + 1],
#                 H2s[i + 1],
#                 epus[i + 1]))

uh0_no_penalty = uh0
```

```
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 = 0.01
  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 = 0.0001
  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
```

2.2.2 With penalty (Problem2)

[66]:
```
sigma = 5
for j in range(epsilon_range):
    epsilon = 1 * 10**(-j * 2)
    ep = epsilon
    L2_list = []
    Du_list = []
    D2u_list = []
    h_list = []
    epu_list = []
    m = MeshTri()

    for i in range(1, refine_time + 1):

        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())
```

```python
        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])))
#         print(
#             '2^-' + str(i + 2),
#             ' {:.5f}  {:.5f}  {:.5f}  {:.5f}'.format(L2s[i + 1], H1s[i + 1],
#                                                       H2s[i + 1], epus[i + 1]))

uh0_penalty = uh0
```

```
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  1.67  0.87  0.70  0.69
2^-3  2.26  1.80  1.07  1.02
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 = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.25  0.59  0.21  0.51
2^-3  2.31  1.56  0.69  1.48
2^-4  2.24  1.90  0.98  1.79
2^-5  1.85  2.18  1.36  1.95
2^-6  1.83  2.20  1.46  1.80
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.24  0.58  0.20  0.50
2^-3  2.30  1.50  0.60  1.45
2^-4  2.37  1.67  0.66  1.65
```

```
2^-5  2.23  1.66  0.62  1.65
2^-6  2.11  1.61  0.58  1.60
```

[22]:
```python
mem.n[1][mem.x[0] == 0]   # ny when x = 0
```

[22]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

[23]:
```python
mem.n[0][mem.x[0]==0] # nx when x = 0
```

[23]:
```
array([-1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1.])
```

[24]:
```python
mem.n[0][mem.x[0]==1] # nx when x = 1
```

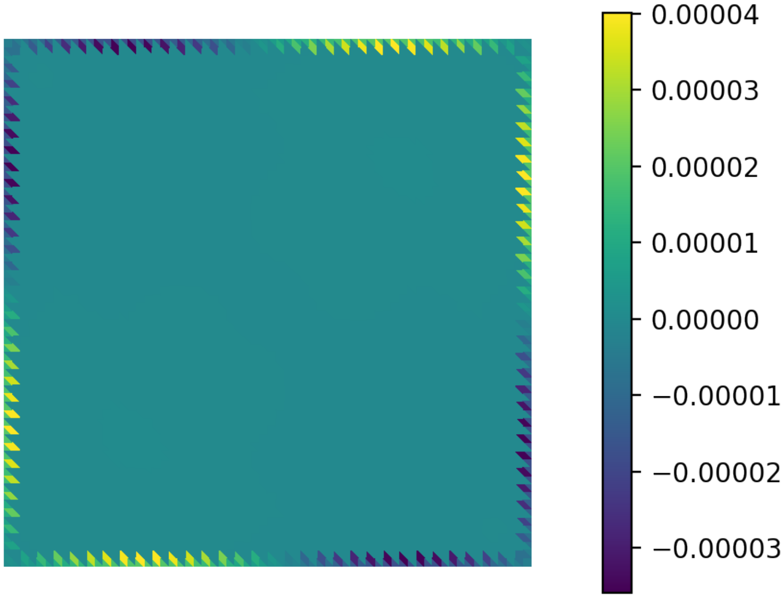
[24]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[25]:
```python
mem.n[1][mem.x[1]==1] # ny when y = 1
```

[25]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[26]:
```python
plot(basis['u'], uh0_penalty-uh0_no_penalty, colorbar=True)
```

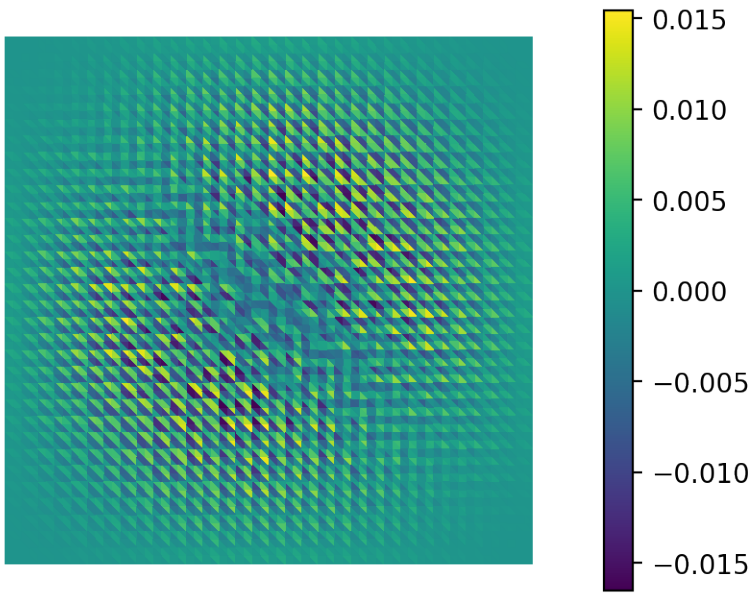
[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22b3d5abb88>
```



[29]:
```
x = basis['u'].doflocs[0]
y = basis['u'].doflocs[1]
u = exact_u(x, y)

plot(basis['u'], u-uh0_penalty, colorbar = True)
```

[29]: <matplotlib.axes._subplots.AxesSubplot at 0x22b3df72d08>



[30]:
```
# uh0_penalty[m.boundary_nodes()]

m = MeshTri()
m.refine(refine_time)

# fbasis_dof = FacetBasis(m,
#                       ElementTriMorley())

fbasis_dof = FacetBasis(m,
                        ElementTriMorley(),
                        quadrature=(np.array([[0.0, 0.5, 1.0]]), np.array(
                            [1, 1, 1])))  # quadrature: points and weights

p3 = asm(penalty_3, fbasis_dof)
```

∂u_n of uh0_without_penalty on boundary nodes

Data structure: $[n1, n2, n3]$ for each facet

- $n1, n3$: ∂u_n on two ends of a facet
- $n2$: ∂u_n on the middle point of a facet

[31]:
```
dot(fbasis_dof.interpolate(uh0_no_penalty).grad, mem.n)
```

[31]:
```
array([[-3.98486361e-05,  0.00000000e+00,  3.98486361e-05],
       [-3.98486361e-05,  0.00000000e+00,  3.98486361e-05],
       [ 1.41473766e-03,  0.00000000e+00, -1.41473766e-03],
       [ 1.41473766e-03,  0.00000000e+00, -1.41473766e-03],
       [ 1.41473766e-03,  0.00000000e+00, -1.41473766e-03],
       [ 1.41473766e-03,  1.25653946e-18, -1.41473766e-03],
       [-3.98486361e-05,  4.00422675e-19,  3.98486361e-05],
       [-3.98486361e-05,  0.00000000e+00,  3.98486361e-05],
       [-1.38149039e-03,  0.00000000e+00,  1.38149039e-03],
       [ 6.16103823e-06,  0.00000000e+00, -6.16103824e-06],
       [-1.38149039e-03,  0.00000000e+00,  1.38149039e-03],
       [ 6.16103873e-06,  0.00000000e+00, -6.16103873e-06],
       [ 6.16103848e-06, -1.07659996e-32, -6.16103848e-06],
       [-1.38149039e-03, -1.54391690e-15,  1.38149039e-03],
       [ 6.16103900e-06,  1.54461980e-15, -6.16103900e-06],
       [-1.38149039e-03,  0.00000000e+00,  1.38149039e-03],
       [-6.91102596e-03,  0.00000000e+00,  6.91102596e-03],
       [ 7.04922718e-03,  0.00000000e+00, -7.04922718e-03],
       [-6.91102596e-03,  0.00000000e+00,  6.91102596e-03],
       [ 7.04922718e-03,  0.00000000e+00, -7.04922718e-03],
       [-7.05141428e-03,  0.00000000e+00,  7.05141428e-03],
       [ 6.91610036e-03,  0.00000000e+00, -6.91610036e-03],
```

```
[-7.05141428e-03,  2.60158330e-32,  7.05141428e-03],
[ 6.91610036e-03,  8.51898473e-16, -6.91610036e-03],
[-7.05141428e-03,  0.00000000e+00,  7.05141428e-03],
[ 6.91610035e-03,  0.00000000e+00, -6.91610035e-03],
[-7.05141428e-03, -1.74957081e-16,  7.05141428e-03],
[ 6.91610035e-03, -2.12974618e-16, -6.91610035e-03],
[-6.91102596e-03,  6.92740116e-16,  6.91102596e-03],
[ 7.04922718e-03,  1.04021350e-31, -7.04922718e-03],
[-6.91102596e-03, -6.92740116e-16,  6.91102596e-03],
[ 7.04922718e-03,  0.00000000e+00, -7.04922718e-03],
[-3.89736613e-03,  0.00000000e+00,  3.89736613e-03],
[ 4.97007004e-03,  0.00000000e+00, -4.97007004e-03],
[-3.89736613e-03,  0.00000000e+00,  3.89736613e-03],
[ 4.97007004e-03,  0.00000000e+00, -4.97007004e-03],
[-4.98630277e-03,  0.00000000e+00,  4.98630277e-03],
[ 5.86438870e-03,  0.00000000e+00, -5.86438870e-03],
[-4.98630277e-03,  0.00000000e+00,  4.98630277e-03],
[ 5.86438870e-03,  0.00000000e+00, -5.86438870e-03],
[-4.98630277e-03,  0.00000000e+00,  4.98630277e-03],
[ 5.86438870e-03,  0.00000000e+00, -5.86438870e-03],
[-4.98630277e-03, -4.37194717e-17,  4.98630277e-03],
[ 5.86438870e-03, -1.41462141e-16, -5.86438870e-03],
[-3.89736613e-03,  6.68289582e-32,  3.89736613e-03],
[ 4.97007004e-03, -2.77505244e-16, -4.97007004e-03],
[-3.89736613e-03,  1.70491581e-16,  3.89736613e-03],
[ 4.97007004e-03,  0.00000000e+00, -4.97007004e-03],
[ 4.98861660e-03,  0.00000000e+00, -4.98861660e-03],
[-5.86465643e-03,  0.00000000e+00,  5.86465643e-03],
[ 3.91291238e-03,  0.00000000e+00, -3.91291238e-03],
[-4.98277885e-03,  0.00000000e+00,  4.98277885e-03],
[ 4.98861661e-03,  0.00000000e+00, -4.98861661e-03],
[-5.86465643e-03,  0.00000000e+00,  5.86465643e-03],
[ 3.91291238e-03,  0.00000000e+00, -3.91291238e-03],
[-4.98277885e-03,  0.00000000e+00,  4.98277885e-03],
[ 3.91291238e-03,  0.00000000e+00, -3.91291238e-03],
[-4.98277885e-03,  0.00000000e+00,  4.98277885e-03],
[ 4.98861661e-03,  1.36965800e-15, -4.98861661e-03],
[-5.86465643e-03,  8.04960916e-32,  5.86465643e-03],
[ 3.91291238e-03,  0.00000000e+00, -3.91291238e-03],
[-4.98277885e-03,  0.00000000e+00,  4.98277885e-03],
[ 4.98861661e-03, -6.84828999e-16, -4.98861661e-03],
[-5.86465643e-03,  1.26163943e-15,  5.86465643e-03],
[-1.34639673e-03,  0.00000000e+00,  1.34639673e-03],
[ 2.67380499e-03,  0.00000000e+00, -2.67380499e-03],
[-1.34639673e-03,  0.00000000e+00,  1.34639673e-03],
[ 2.67380499e-03,  0.00000000e+00, -2.67380499e-03],
[-2.69459694e-03,  0.00000000e+00,  2.69459694e-03],
```

```
[ 3.91604443e-03,  0.00000000e+00, -3.91604443e-03],
[-2.69459694e-03,  0.00000000e+00,  2.69459694e-03],
[ 3.91604443e-03,  0.00000000e+00, -3.91604443e-03],
[-2.69459694e-03,  0.00000000e+00,  2.69459694e-03],
[ 3.91604443e-03,  0.00000000e+00, -3.91604443e-03],
[-2.69459694e-03,  1.34400484e-17,  2.69459694e-03],
[ 3.91604443e-03,  1.06487686e-17, -3.91604443e-03],
[-1.34639673e-03, -2.90782449e-17,  1.34639673e-03],
[ 2.67380499e-03,  8.66094691e-17, -2.67380499e-03],
[-1.34639673e-03,  2.90782449e-17,  1.34639673e-03],
[ 2.67380499e-03,  0.00000000e+00, -2.67380499e-03],
[ 2.70331593e-03,  0.00000000e+00, -2.70331593e-03],
[-3.92120091e-03,  0.00000000e+00,  3.92120091e-03],
[ 1.36976255e-03,  0.00000000e+00, -1.36976255e-03],
[-2.69316344e-03,  0.00000000e+00,  2.69316344e-03],
[ 2.70331593e-03,  0.00000000e+00, -2.70331593e-03],
[-3.92120091e-03,  0.00000000e+00,  3.92120091e-03],
[ 1.36976255e-03,  0.00000000e+00, -1.36976255e-03],
[-2.69316344e-03,  0.00000000e+00,  2.69316344e-03],
[ 1.36976255e-03, -1.51564347e-15, -1.36976255e-03],
[-2.69316344e-03,  1.45810122e-15,  2.69316344e-03],
[ 2.70331593e-03, -2.98529357e-32, -2.70331593e-03],
[-3.92120091e-03,  1.45472193e-15,  3.92120091e-03],
[ 1.36976255e-03, -7.57821735e-16, -1.36976255e-03],
[-2.69316344e-03,  7.29050609e-16,  2.69316344e-03],
[ 2.70331593e-03,  7.56781016e-16, -2.70331593e-03],
[-3.92120091e-03,  7.27360963e-16,  3.92120091e-03],
[ 6.50637559e-03,  0.00000000e+00, -6.50637559e-03],
[-5.85108687e-03,  0.00000000e+00,  5.85108687e-03],
[-6.91544607e-03,  0.00000000e+00,  6.91544607e-03],
[ 6.51542562e-03,  0.00000000e+00, -6.51542562e-03],
[ 6.50637559e-03,  0.00000000e+00, -6.50637559e-03],
[-5.85108687e-03,  0.00000000e+00,  5.85108687e-03],
[-6.91544607e-03,  0.00000000e+00,  6.91544607e-03],
[ 6.51542562e-03,  0.00000000e+00, -6.51542562e-03],
[ 6.91686986e-03,  0.00000000e+00, -6.91686986e-03],
[-6.51625802e-03,  0.00000000e+00,  6.51625802e-03],
[-6.51437140e-03,  0.00000000e+00,  6.51437140e-03],
[ 5.86130379e-03,  0.00000000e+00, -5.86130379e-03],
[ 6.91686986e-03,  0.00000000e+00, -6.91686986e-03],
[-6.51625802e-03, -4.09769151e-16,  6.51625802e-03],
[-6.51437140e-03,  0.00000000e+00,  6.51437140e-03],
[ 5.86130379e-03,  0.00000000e+00, -5.86130379e-03],
[ 6.91686986e-03,  0.00000000e+00, -6.91686986e-03],
[-6.51625802e-03,  0.00000000e+00,  6.51625802e-03],
[-6.51437140e-03,  0.00000000e+00,  6.51437140e-03],
[ 5.86130379e-03,  0.00000000e+00, -5.86130379e-03],
```

```
[ 6.91686986e-03,  0.00000000e+00, -6.91686986e-03],
[-6.51625802e-03,  1.02442288e-16,  6.51625802e-03],
[-6.51437140e-03, -5.00453619e-16,  6.51437140e-03],
[ 5.86130379e-03,  2.85282257e-16, -5.86130379e-03],
[ 6.50637559e-03,  0.00000000e+00, -6.50637559e-03],
[-5.85108687e-03,  0.00000000e+00,  5.85108687e-03],
[-6.91544607e-03,  9.94066249e-16,  6.91544607e-03],
[ 6.51542562e-03, -1.13481753e-15, -6.51542562e-03],
[ 6.50637559e-03,  5.43745735e-16, -6.50637559e-03],
[-5.85108687e-03,  0.00000000e+00,  5.85108687e-03],
[-6.91544607e-03, -9.94066249e-16,  6.91544607e-03],
[ 6.51542562e-03,  0.00000000e+00, -6.51542562e-03]])
```

∂u_n of uh0_penalty on boundary nodes

[32]:
```
dot(fbasis_dof.interpolate(uh0_penalty).grad, mem.n)

# ### Showing examples of facets used in caculating penalty and also $\partial
 ↪u_{n}$

# for i in [0,8]:
#     plt.scatter(mem.x[0][i], mem.x[1][i], s=4, marker='*')
#     plt.axis('square')
```

[32]:
```
array([[-2.66740775e-04, -2.25581062e-04, -1.84421350e-04],
       [-2.66740775e-04, -2.25581062e-04, -1.84421350e-04],
       [ 2.86686179e-03,  1.45356139e-03,  4.02609856e-05],
       [ 2.86686179e-03,  1.45356139e-03,  4.02609856e-05],
       [ 2.86686179e-03,  1.45356139e-03,  4.02609856e-05],
       [ 2.86686179e-03,  1.45356139e-03,  4.02609856e-05],
       [-2.66740775e-04, -2.25581062e-04, -1.84421350e-04],
       [-2.66740775e-04, -2.25581062e-04, -1.84421350e-04],
       [-3.55755888e-03, -2.17584926e-03, -7.94139631e-04],
       [-7.55719389e-04, -7.62617081e-04, -7.69514774e-04],
       [-3.55755888e-03, -2.17584926e-03, -7.94139631e-04],
       [-7.55719389e-04, -7.62617081e-04, -7.69514774e-04],
       [-7.55719389e-04, -7.62617081e-04, -7.69514774e-04],
       [-3.55755888e-03, -2.17584926e-03, -7.94139631e-04],
       [-7.55719389e-04, -7.62617081e-04, -7.69514774e-04],
       [-3.55755888e-03, -2.17584926e-03, -7.94139631e-04],
       [-1.43866520e-02, -7.47729539e-03, -5.67938795e-04],
       [-6.38497885e-04, -7.68624773e-03, -1.47339976e-02],
       [-1.43866520e-02, -7.47729539e-03, -5.67938795e-04],
       [-6.38497885e-04, -7.68624773e-03, -1.47339976e-02],
       [-1.61103543e-04,  6.89017829e-03,  1.39414601e-02],
       [ 1.35927039e-02,  6.67643929e-03, -2.39825319e-04],
       [-1.61103543e-04,  6.89017829e-03,  1.39414601e-02],
```

```
[ 1.35927039e-02,  6.67643929e-03, -2.39825319e-04],
[-1.61103543e-04,  6.89017829e-03,  1.39414601e-02],
[ 1.35927039e-02,  6.67643929e-03, -2.39825319e-04],
[-1.61103543e-04,  6.89017829e-03,  1.39414601e-02],
[ 1.35927039e-02,  6.67643929e-03, -2.39825319e-04],
[-1.43866520e-02, -7.47729539e-03, -5.67938795e-04],
[-6.38497885e-04, -7.68624773e-03, -1.47339976e-02],
[-1.43866520e-02, -7.47729539e-03, -5.67938795e-04],
[-6.38497885e-04, -7.68624773e-03, -1.47339976e-02],
[-8.05958478e-03, -4.16268900e-03, -2.65793226e-04],
[-3.37563374e-04, -5.30719595e-03, -1.02768285e-02],
[-8.05958478e-03, -4.16268900e-03, -2.65793226e-04],
[-3.37563374e-04, -5.30719595e-03, -1.02768285e-02],
[ 5.38774082e-05,  5.03964573e-03,  1.00254141e-02],
[ 1.17456369e-02,  5.88139497e-03,  1.71530611e-05],
[ 5.38774082e-05,  5.03964573e-03,  1.00254141e-02],
[ 1.17456369e-02,  5.88139497e-03,  1.71530611e-05],
[ 5.38774082e-05,  5.03964573e-03,  1.00254141e-02],
[ 1.17456369e-02,  5.88139497e-03,  1.71530612e-05],
[ 5.38774082e-05,  5.03964573e-03,  1.00254141e-02],
[ 1.17456369e-02,  5.88139497e-03,  1.71530611e-05],
[-8.05958478e-03, -4.16268900e-03, -2.65793226e-04],
[-3.37563374e-04, -5.30719595e-03, -1.02768285e-02],
[-8.05958478e-03, -4.16268900e-03, -2.65793226e-04],
[-3.37563374e-04, -5.30719595e-03, -1.02768285e-02],
[-8.18514674e-04, -5.80608887e-03, -1.07936631e-02],
[-1.25204028e-02, -6.65723560e-03, -7.94068441e-04],
[ 7.25542571e-03,  3.34350604e-03, -5.68413626e-04],
[-4.89507695e-04,  4.49260073e-03,  9.47470916e-03],
[-8.18514674e-04, -5.80608887e-03, -1.07936631e-02],
[-1.25204028e-02, -6.65723560e-03, -7.94068441e-04],
[ 7.25542571e-03,  3.34350604e-03, -5.68413626e-04],
[-4.89507695e-04,  4.49260073e-03,  9.47470916e-03],
[ 7.25542571e-03,  3.34350604e-03, -5.68413626e-04],
[-4.89507695e-04,  4.49260073e-03,  9.47470916e-03],
[-8.18514674e-04, -5.80608887e-03, -1.07936631e-02],
[-1.25204028e-02, -6.65723560e-03, -7.94068441e-04],
[ 7.25542571e-03,  3.34350604e-03, -5.68413626e-04],
[-4.89507695e-04,  4.49260073e-03,  9.47470916e-03],
[-8.18514674e-04, -5.80608887e-03, -1.07936631e-02],
[-1.25204028e-02, -6.65723560e-03, -7.94068441e-04],
[-2.84318134e-03, -1.49533878e-03, -1.47496226e-04],
[-2.00806156e-04, -2.87512094e-03, -5.54943572e-03],
[-2.84318134e-03, -1.49533878e-03, -1.47496226e-04],
[-2.00806156e-04, -2.87512094e-03, -5.54943572e-03],
[ 8.61496764e-05,  2.78039533e-03,  5.47464099e-03],
[ 7.90711386e-03,  3.99138752e-03,  7.56611685e-05],
```

```
[ 8.61496764e-05,  2.78039533e-03,  5.47464099e-03],
[ 7.90711386e-03,  3.99138752e-03,  7.56611685e-05],
[ 8.61496764e-05,  2.78039533e-03,  5.47464099e-03],
[ 7.90711386e-03,  3.99138752e-03,  7.56611685e-05],
[ 8.61496764e-05,  2.78039533e-03,  5.47464099e-03],
[ 7.90711386e-03,  3.99138752e-03,  7.56611685e-05],
[-2.84318134e-03, -1.49533878e-03, -1.47496226e-04],
[-2.00806156e-04, -2.87512094e-03, -5.54943572e-03],
[-2.84318134e-03, -1.49533878e-03, -1.47496226e-04],
[-2.00806156e-04, -2.87512094e-03, -5.54943572e-03],
[-8.17420251e-04, -3.52095924e-03, -6.22449822e-03],
[-8.66700427e-03, -4.74672014e-03, -8.26436014e-04],
[ 2.03404106e-03,  6.65044151e-04, -7.03952757e-04],
[-6.40062222e-04,  2.05195624e-03,  4.74397469e-03],
[-8.17420251e-04, -3.52095924e-03, -6.22449822e-03],
[-8.66700427e-03, -4.74672014e-03, -8.26436014e-04],
[ 2.03404106e-03,  6.65044151e-04, -7.03952757e-04],
[-6.40062222e-04,  2.05195624e-03,  4.74397469e-03],
[ 2.03404106e-03,  6.65044151e-04, -7.03952757e-04],
[-6.40062222e-04,  2.05195624e-03,  4.74397469e-03],
[-8.17420251e-04, -3.52095924e-03, -6.22449822e-03],
[-8.66700427e-03, -4.74672014e-03, -8.26436014e-04],
[ 2.03404106e-03,  6.65044151e-04, -7.03952757e-04],
[-6.40062222e-04,  2.05195624e-03,  4.74397469e-03],
[-8.17420251e-04, -3.52095924e-03, -6.22449822e-03],
[-8.66700427e-03, -4.74672014e-03, -8.26436014e-04],
[-4.91113907e-04, -6.99676871e-03, -1.35024235e-02],
[-1.21133523e-02, -6.26386542e-03, -4.14378561e-04],
[-1.45297441e-02, -7.61591855e-03, -7.02092999e-04],
[-7.54200486e-04, -7.26821183e-03, -1.37822232e-02],
[-4.91113907e-04, -6.99676871e-03, -1.35024235e-02],
[-1.21133523e-02, -6.26386542e-03, -4.14378561e-04],
[-1.45297441e-02, -7.61591855e-03, -7.02092999e-04],
[-7.54200486e-04, -7.26821183e-03, -1.37822232e-02],
[ 1.37429822e-02,  6.82592093e-03, -9.11403228e-05],
[-3.05004265e-05,  6.48520796e-03,  1.30009163e-02],
[-3.22835132e-04,  6.19110947e-03,  1.27050541e-02],
[ 1.13152322e-02,  5.45421432e-03, -4.06803607e-04],
[ 1.37429822e-02,  6.82592093e-03, -9.11403227e-05],
[-3.05004265e-05,  6.48520796e-03,  1.30009163e-02],
[-3.22835132e-04,  6.19110947e-03,  1.27050541e-02],
[ 1.13152322e-02,  5.45421432e-03, -4.06803607e-04],
[ 1.37429822e-02,  6.82592093e-03, -9.11403227e-05],
[-3.05004265e-05,  6.48520796e-03,  1.30009163e-02],
[-3.22835132e-04,  6.19110947e-03,  1.27050541e-02],
[ 1.13152322e-02,  5.45421432e-03, -4.06803607e-04],
[ 1.37429822e-02,  6.82592093e-03, -9.11403228e-05],
```

```
[-3.05004265e-05,  6.48520796e-03,  1.30009163e-02],
[-3.22835132e-04,  6.19110947e-03,  1.27050541e-02],
[ 1.13152322e-02,  5.45421432e-03, -4.06803607e-04],
[-4.91113907e-04, -6.99676871e-03, -1.35024235e-02],
[-1.21133523e-02, -6.26386542e-03, -4.14378561e-04],
[-1.45297441e-02, -7.61591855e-03, -7.02092999e-04],
[-7.54200486e-04, -7.26821183e-03, -1.37822232e-02],
[-4.91113907e-04, -6.99676871e-03, -1.35024235e-02],
[-1.21133523e-02, -6.26386542e-03, -4.14378561e-04],
[-1.45297441e-02, -7.61591855e-03, -7.02092999e-04],
[-7.54200486e-04, -7.26821183e-03, -1.37822232e-02]])
```

2.3 Example 2

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

where

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

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

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

[72]:

```
@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)

[73]:
```
refine_time = 5
epsilon_range = 4
for j in range(epsilon_range):
    epsilon = 1 * 10**(-j*2)
    ep = epsilon
    L2_list = []
    Du_list = []
    D2u_list = []
    h_list = []
```

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

    for i in range(1, refine_time+1):

        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)

#     x = basis['u'].doflocs[0]
#     y = basis['u'].doflocs[1]
#     u = exact_u(x, y)
#     plot(basis['u'], u-uh0, colorbar = True)
#     plt.show()

    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])))

uh0_no_penalty = uh0
```

```
epsilon = 1
  h    L2u   H1u   H2u   epu
```

```
2^-2  2.03  1.80  1.11  1.05
2^-3  1.96  1.82  1.01  0.97
2^-4  2.01  1.95  1.03  1.00
2^-5  2.02  2.00  1.02  1.01
epsilon = 0.01
  h    L2u   H1u   H2u    epu
2^-2  0.96  0.71  -0.35  0.62
2^-3  0.99  0.74  -0.28  0.57
2^-4  1.31  1.00  0.08  0.67
2^-5  1.83  1.54  0.57  0.91
epsilon = 0.0001
  h    L2u   H1u   H2u    epu
2^-2  0.97  0.62  -0.32  0.55
2^-3  0.90  0.55  -0.45  0.50
2^-4  0.93  0.54  -0.49  0.50
2^-5  0.95  0.53  -0.50  0.50
epsilon = 1e-06
  h    L2u   H1u   H2u    epu
2^-2  0.97  0.62  -0.32  0.55
2^-3  0.90  0.55  -0.45  0.50
2^-4  0.93  0.54  -0.49  0.50
2^-5  0.96  0.53  -0.50  0.50
```

2.3.2 With penalty (Problem2)

[74]:
```
sigma = 5
for j in range(epsilon_range):
    epsilon = 1 * 10**(-2*j)
    ep = epsilon
    L2_list = []
    Du_list = []
    D2u_list = []
    h_list = []
    epu_list = []
    m = MeshTri.init_symmetric()

    for i in range(1, refine_time+1):

        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
```

```python
        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])))

uh0_penalty = uh0
```

```
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  2.20  2.04  1.15  1.07
2^-3  1.93  1.79  0.99  0.95
2^-4  1.96  1.89  0.99  0.97
2^-5  1.98  1.96  1.00  0.99
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.02  0.70  -0.57  0.58
2^-3  1.01  0.69  -0.33  0.50
2^-4  1.31  0.94  0.08  0.62
2^-5  1.83  1.48  0.57  0.91
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.03  0.65  -0.36  0.56
2^-3  0.92  0.57  -0.46  0.51
2^-4  0.93  0.55  -0.49  0.50
2^-5  0.96  0.54  -0.50  0.50
epsilon = 1e-06
  h    L2u   H1u   H2u   epu
2^-2  1.04  0.65  -0.36  0.56
2^-3  0.92  0.57  -0.46  0.51
```

```
2^-4  0.94  0.55  -0.49  0.50
2^-5  0.96  0.54  -0.50  0.50
```

```
[51]: def exact_f(x, y):
          '''
          for $(f, x_{h})$
          '''
          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))) * 1
```

2.4 Example3

[ ]:

```
```