Perturb_Problem2

October 19, 2020

1 Solving a Fourth Order Elliptic Singular Perturbation Problem

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

[123]:

```
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

[124]:
```
@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

[125]:
```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),
```

```python
        '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': 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

[126]:
```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)
```

```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': ElementTriP1(), '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']) + 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))
    # uh0 = solve(*condense(K2 + P, f2, D=m.boundary_nodes()),␣
↪solver=solver_iter_krylov(Precondition=True))
    return uh0, basis

# easy_boundary(basis['u'])

# easy_boundary_penalty(basis['u'])

# m.boundary_nodes()
```

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

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

[127]:

```
@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)

[133]:
```
refine_time = 5
epsilon_range = 4
for j in range(epsilon_range):
    epsilon = 1 * 10**(-j*2)

    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_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  1.90  1.32  0.83  0.80
2^-3  1.86  1.82  0.97  0.94
2^-4  1.96  1.93  1.00  0.98
2^-5  1.99  1.98  1.01  0.99
epsilon = 0.01
  h     L2u   H1u   H2u   epu
2^-2  1.71  1.43  0.90  1.40
2^-3  2.26  1.77  0.95  1.70
2^-4  2.20  1.88  1.04  1.75
2^-5  2.05  1.92  1.03  1.60
epsilon = 0.0001
  h     L2u   H1u   H2u   epu
2^-2  1.70  1.43  0.90  1.41
2^-3  2.22  1.75  0.92  1.72
2^-4  2.20  1.85  0.96  1.84
2^-5  2.10  1.91  0.99  1.90
epsilon = 1e-06
  h     L2u   H1u   H2u   epu
2^-2  1.70  1.43  0.90  1.41
2^-3  2.22  1.75  0.92  1.72
2^-4  2.20  1.85  0.96  1.84
2^-5  2.10  1.91  0.99  1.90
```

2.2.2 With penalty (Problem2)

[137]:
```
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.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
        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

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

    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  2.52  1.78  1.21  1.17
2^-3  1.76  1.85  1.01  0.98
```

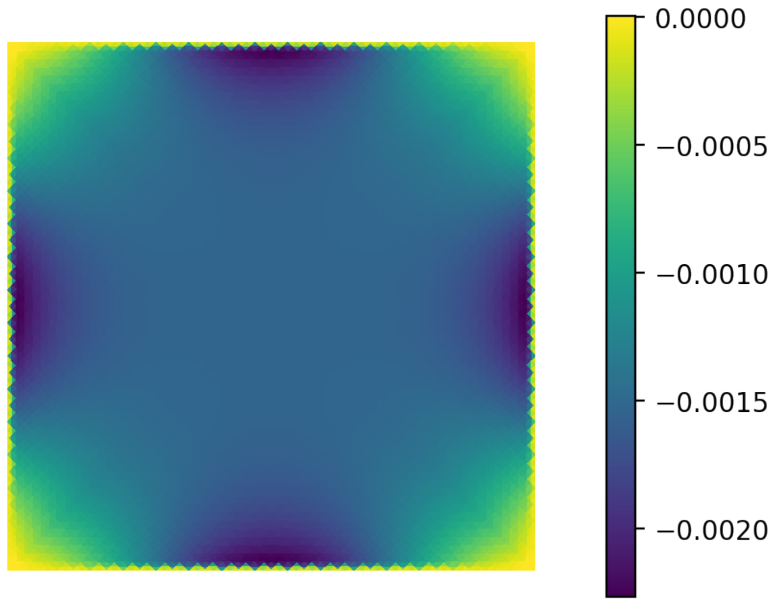
```
2^-4  1.81  1.91  0.98  0.96
2^-5  1.90  1.96  0.99  0.98
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.62  1.29  0.37  1.22
2^-3  1.76  1.59  0.63  1.51
2^-4  1.85  1.60  0.67  1.39
2^-5  1.70  1.51  0.66  1.12
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.62  1.29  0.35  1.24
2^-3  1.77  1.59  0.58  1.56
2^-4  1.93  1.62  0.57  1.58
2^-5  1.98  1.60  0.55  1.57
epsilon = 1e-06
  h    L2u   H1u   H2u   epu
2^-2  1.62  1.29  0.35  1.24
2^-3  1.77  1.59  0.58  1.56
2^-4  1.93  1.62  0.57  1.58
2^-5  1.98  1.60  0.55  1.57
```

2.2.3 Analysising result uh_0 with and without penalty

uh0_penalty-uh0_no_penalty

[138]:
```
plot(basis['u'], uh0_penalty-uh0_no_penalty, colorbar=True)
```

[138]: <matplotlib.axes._subplots.AxesSubplot at 0x1e9b6fe1348>

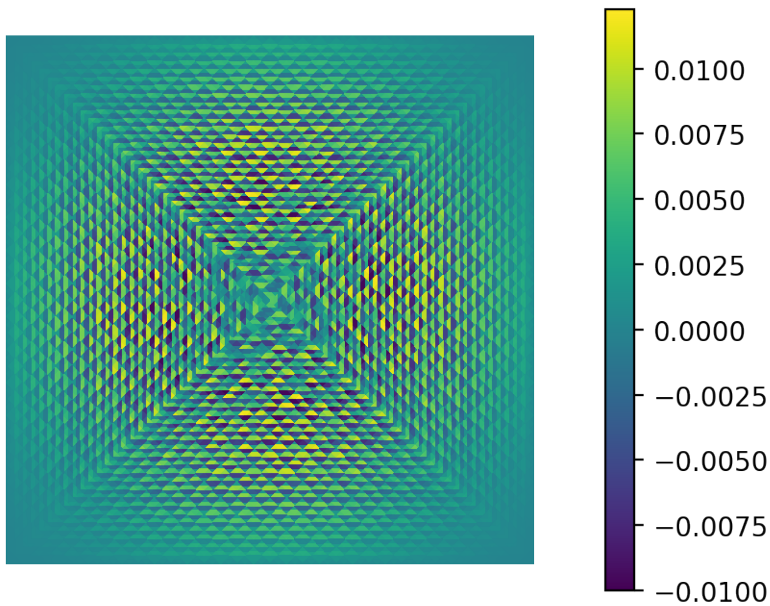


u-uh0_penalty

[145]:
```
x = basis['u'].doflocs[0]
y = basis['u'].doflocs[1]
u = exact_u(x, y)

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

[145]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e9ba829b48>
```



Value of uh0_penalty on boundary nodes

[146]:
```
uh0_penalty[m.boundary_nodes()]
```

[146]:
```
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., 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.])
```

[ ]:
```
m = MeshTri.init_symmetric()
m.refine(refine_time)
```

```python
# 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

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

[149]:
```python
dot(fbasis_dof.interpolate(uh0_no_penalty).grad, mem.n)
```

[149]:
```
array([[ 8.15825591e-04,  0.00000000e+00, -8.15825591e-04],
       [ 8.15825591e-04,  0.00000000e+00, -8.15825591e-04],
       [ 8.15825591e-04,  0.00000000e+00, -8.15825591e-04],
       [ 8.15825591e-04,  0.00000000e+00, -8.15825591e-04],
       [ 8.15825591e-04, -8.19789829e-18, -8.15825591e-04],
       [ 8.15825591e-04, -8.19789829e-18, -8.15825591e-04],
       [ 8.15825591e-04,  0.00000000e+00, -8.15825591e-04],
       [ 8.15825591e-04, -4.09894915e-18, -8.15825591e-04],
       [-7.14345468e-04,  0.00000000e+00,  7.14345468e-04],
       [-7.14345468e-04,  0.00000000e+00,  7.14345468e-04],
       [-7.14345468e-04,  0.00000000e+00,  7.14345468e-04],
       [-7.14345468e-04,  0.00000000e+00,  7.14345468e-04],
       [-7.14345468e-04,  0.00000000e+00,  7.14345468e-04],
       [-7.14345468e-04, -7.66533237e-16,  7.14345468e-04],
       [-7.14345468e-04, -3.83266618e-16,  7.14345468e-04],
       [-7.14345468e-04,  3.79677535e-16,  7.14345468e-04],
       [-7.40150307e-03,  0.00000000e+00,  7.40150307e-03],
       [ 7.36232292e-03,  0.00000000e+00, -7.36232292e-03],
       [-7.40150307e-03,  0.00000000e+00,  7.40150307e-03],
       [ 7.36232292e-03,  0.00000000e+00, -7.36232292e-03],
       [-7.40150307e-03,  0.00000000e+00,  7.40150307e-03],
       [ 7.36232292e-03,  0.00000000e+00, -7.36232292e-03],
       [-7.40150307e-03,  0.00000000e+00,  7.40150307e-03],
       [ 7.36232292e-03, -7.39809772e-17, -7.36232292e-03],
       [-7.40150307e-03, -3.90224561e-16,  7.40150307e-03],
       [ 7.36232292e-03, -4.64205537e-16, -7.36232292e-03],
       [-7.40150307e-03,  3.90224561e-16,  7.40150307e-03],
       [ 7.36232292e-03,  0.00000000e+00, -7.36232292e-03],
       [-7.40150307e-03,  0.00000000e+00,  7.40150307e-03],
```

```
[ 7.36232292e-03,  0.00000000e+00, -7.36232292e-03],
[-7.40150307e-03,  0.00000000e+00,  7.40150307e-03],
[ 7.36232292e-03,  4.27215049e-16, -7.36232292e-03],
[-4.89813934e-03,  0.00000000e+00,  4.89813934e-03],
[ 5.89314768e-03,  0.00000000e+00, -5.89314768e-03],
[-4.89813934e-03,  0.00000000e+00,  4.89813934e-03],
[ 5.89314768e-03,  0.00000000e+00, -5.89314768e-03],
[-4.89813934e-03,  0.00000000e+00,  4.89813934e-03],
[ 5.89314768e-03,  0.00000000e+00, -5.89314768e-03],
[-4.89813934e-03,  0.00000000e+00,  4.89813934e-03],
[ 5.89314768e-03,  0.00000000e+00, -5.89314768e-03],
[-4.89813934e-03,  0.00000000e+00,  4.89813934e-03],
[ 5.89314768e-03, -1.76977567e-16, -5.89314768e-03],
[-4.89813934e-03, -1.17759733e-16,  4.89813934e-03],
[ 5.89314768e-03,  1.76977567e-16, -5.89314768e-03],
[-4.89813934e-03,  0.00000000e+00,  4.89813934e-03],
[ 5.89314768e-03,  0.00000000e+00, -5.89314768e-03],
[-4.89813934e-03, -1.84572759e-17,  4.89813934e-03],
[ 5.89314768e-03,  7.40222938e-18, -5.89314768e-03],
[ 4.63554148e-03,  0.00000000e+00, -4.63554148e-03],
[-5.65954429e-03,  0.00000000e+00,  5.65954429e-03],
[ 4.63554148e-03,  0.00000000e+00, -4.63554148e-03],
[-5.65954429e-03,  0.00000000e+00,  5.65954429e-03],
[ 4.63554148e-03,  0.00000000e+00, -4.63554148e-03],
[-5.65954429e-03,  0.00000000e+00,  5.65954429e-03],
[ 4.63554148e-03,  0.00000000e+00, -4.63554148e-03],
[-5.65954429e-03,  0.00000000e+00,  5.65954429e-03],
[ 4.63554148e-03, -6.56964676e-16, -4.63554148e-03],
[-5.65954429e-03,  0.00000000e+00,  5.65954429e-03],
[ 4.63554148e-03,  0.00000000e+00, -4.63554148e-03],
[-5.65954429e-03, -6.56964677e-16,  5.65954429e-03],
[ 4.63554148e-03, -7.03545340e-16, -4.63554148e-03],
[-5.65954429e-03, -3.28482338e-16,  5.65954429e-03],
[ 4.63554148e-03,  0.00000000e+00, -4.63554148e-03],
[-5.65954429e-03,  3.00047113e-16,  5.65954429e-03],
[-2.30942843e-03,  0.00000000e+00,  2.30942843e-03],
[ 3.69563525e-03,  0.00000000e+00, -3.69563525e-03],
[-2.30942843e-03,  0.00000000e+00,  2.30942843e-03],
[ 3.69563525e-03,  0.00000000e+00, -3.69563525e-03],
[-2.30942843e-03,  0.00000000e+00,  2.30942843e-03],
[ 3.69563525e-03,  0.00000000e+00, -3.69563525e-03],
[-2.30942843e-03,  8.19789810e-18,  2.30942843e-03],
[ 3.69563525e-03,  0.00000000e+00, -3.69563525e-03],
[-2.30942843e-03,  0.00000000e+00,  2.30942843e-03],
[ 3.69563525e-03,  0.00000000e+00, -3.69563525e-03],
[-2.30942843e-03, -3.14044016e-17,  2.30942843e-03],
[ 3.69563525e-03,  0.00000000e+00, -3.69563525e-03],
```

```
[-2.30942843e-03,  0.00000000e+00,  2.30942843e-03],
[ 3.69563525e-03,  0.00000000e+00, -3.69563525e-03],
[-2.30942843e-03,  8.70243880e-18,  2.30942843e-03],
[ 3.69563525e-03,  0.00000000e+00, -3.69563525e-03],
[ 2.11543286e-03,  0.00000000e+00, -2.11543286e-03],
[-3.43855254e-03,  0.00000000e+00,  3.43855254e-03],
[ 2.11543286e-03,  0.00000000e+00, -2.11543286e-03],
[-3.43855254e-03,  0.00000000e+00,  3.43855254e-03],
[ 2.11543286e-03,  0.00000000e+00, -2.11543286e-03],
[-3.43855254e-03,  0.00000000e+00,  3.43855254e-03],
[ 2.11543286e-03,  0.00000000e+00, -2.11543286e-03],
[-3.43855254e-03,  0.00000000e+00,  3.43855254e-03],
[ 2.11543286e-03, -7.38097950e-16, -2.11543286e-03],
[-3.43855254e-03,  7.03545340e-16,  3.43855254e-03],
[ 2.11543286e-03,  7.59355071e-16, -2.11543286e-03],
[-3.43855254e-03,  7.38097950e-16,  3.43855254e-03],
[ 2.11543286e-03,  0.00000000e+00, -2.11543286e-03],
[-3.43855254e-03, -3.69048975e-16,  3.43855254e-03],
[ 2.11543286e-03,  7.38097950e-16, -2.11543286e-03],
[-3.43855254e-03,  3.51772670e-16,  3.43855254e-03],
[ 7.16411341e-03,  0.00000000e+00, -7.16411341e-03],
[-6.65596381e-03,  0.00000000e+00,  6.65596381e-03],
[-7.04922022e-03,  0.00000000e+00,  7.04922022e-03],
[ 6.47393763e-03,  0.00000000e+00, -6.47393763e-03],
[ 7.16411341e-03,  0.00000000e+00, -7.16411341e-03],
[-6.65596381e-03,  0.00000000e+00,  6.65596381e-03],
[-7.04922022e-03,  0.00000000e+00,  7.04922022e-03],
[ 6.47393763e-03,  0.00000000e+00, -6.47393763e-03],
[ 7.16411341e-03,  0.00000000e+00, -7.16411341e-03],
[-6.65596381e-03,  0.00000000e+00,  6.65596381e-03],
[-7.04922022e-03,  0.00000000e+00,  7.04922022e-03],
[ 6.47393763e-03,  0.00000000e+00, -6.47393763e-03],
[ 7.16411341e-03,  3.15849879e-16, -7.16411341e-03],
[-6.65596381e-03,  2.43860629e-16,  6.65596381e-03],
[-7.04922022e-03,  0.00000000e+00,  7.04922022e-03],
[ 6.47393763e-03,  0.00000000e+00, -6.47393763e-03],
[ 7.16411341e-03, -3.15849879e-16, -7.16411341e-03],
[-6.65596381e-03,  0.00000000e+00,  6.65596381e-03],
[-7.04922022e-03, -5.35040272e-16,  7.04922022e-03],
[ 6.47393763e-03,  6.00094228e-16, -6.47393763e-03],
[ 7.16411341e-03,  3.15849879e-16, -7.16411341e-03],
[-6.65596381e-03,  0.00000000e+00,  6.65596381e-03],
[-7.04922022e-03,  0.00000000e+00,  7.04922022e-03],
[ 6.47393763e-03, -3.00047114e-16, -6.47393763e-03],
[ 7.16411341e-03,  0.00000000e+00, -7.16411341e-03],
[-6.65596381e-03,  0.00000000e+00,  6.65596381e-03],
[-7.04922022e-03,  0.00000000e+00,  7.04922022e-03],
```

```
       [ 6.47393763e-03,  0.00000000e+00, -6.47393763e-03],
       [ 7.16411341e-03,  1.79973127e-17, -7.16411341e-03],
       [-6.65596381e-03, -1.67207657e-17,  6.65596381e-03],
       [-7.04922022e-03,  1.77086840e-17,  7.04922022e-03],
       [ 6.47393763e-03, -1.62634890e-17, -6.47393763e-03]])
```

∂u_n of `uh0_penalty` on boundary nodes

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

```
[150]: array([[ 9.07738686e-07,  3.40561544e-07, -2.26615599e-07],
       [ 9.05261427e-07,  3.38081218e-07, -2.29098991e-07],
       [ 8.90125283e-07,  3.22956388e-07, -2.44212507e-07],
       [ 9.22832583e-07,  3.55646422e-07, -2.11539740e-07],
       [ 8.76852049e-07,  3.09687700e-07, -2.57476649e-07],
       [ 9.36266599e-07,  3.69075321e-07, -1.98115958e-07],
       [ 9.18893184e-07,  3.51710757e-07, -2.15471671e-07],
       [ 8.94105525e-07,  3.26931109e-07, -2.40243306e-07],
       [-1.28157444e-05, -1.27461504e-05, -1.26765564e-05],
       [-1.28157112e-05, -1.27461216e-05, -1.26765320e-05],
       [-1.28144853e-05, -1.27453600e-05, -1.26762348e-05],
       [-1.28159842e-05, -1.27452390e-05, -1.26744939e-05],
       [-1.28160241e-05, -1.27460777e-05, -1.26761313e-05],
       [-1.28145742e-05, -1.27461349e-05, -1.26776957e-05],
       [-1.28160297e-05, -1.27460997e-05, -1.26761698e-05],
       [-1.28157989e-05, -1.27461253e-05, -1.26764517e-05],
       [-6.37685554e-06, -5.65173562e-06, -4.92661569e-06],
       [-6.21314212e-06, -6.93406209e-06, -7.65498206e-06],
       [-6.37716686e-06, -5.65191608e-06, -4.92666530e-06],
       [-6.21164453e-06, -6.93397063e-06, -7.65629673e-06],
       [-6.37685965e-06, -5.65181382e-06, -4.92676799e-06],
       [-6.21300426e-06, -6.93392848e-06, -7.65485269e-06],
       [-6.37813742e-06, -5.65187258e-06, -4.92560773e-06],
       [-6.21198631e-06, -6.93398920e-06, -7.65599210e-06],
       [-6.37602663e-06, -5.65180088e-06, -4.92757513e-06],
       [-6.22917176e-06, -6.93401619e-06, -7.63886062e-06],
       [-6.36064585e-06, -5.65186333e-06, -4.94308081e-06],
       [-6.21252902e-06, -6.93390533e-06, -7.65528164e-06],
       [-6.37493040e-06, -5.65195116e-06, -4.92897192e-06],
       [-6.21274960e-06, -6.93391055e-06, -7.65507149e-06],
       [-6.37704393e-06, -5.65188977e-06, -4.92673561e-06],
       [-6.22880868e-06, -6.93392669e-06, -7.63904471e-06],
       [-1.67975419e-06, -1.20002378e-06, -7.20293374e-07],
       [-1.53714802e-06, -2.11442721e-06, -2.69170641e-06],
       [-1.67983162e-06, -1.19977114e-06, -7.19710670e-07],
       [-1.53750566e-06, -2.11450002e-06, -2.69149439e-06],
       [-1.68002913e-06, -1.19993608e-06, -7.19843032e-07],
```

```
[-1.53699213e-06, -2.11457631e-06, -2.69216049e-06],
[-1.67973563e-06, -1.19990446e-06, -7.20073290e-07],
[-1.53711704e-06, -2.11451098e-06, -2.69190493e-06],
[-1.67812585e-06, -1.19993244e-06, -7.21739039e-07],
[-1.55035032e-06, -2.11443847e-06, -2.67852662e-06],
[-1.67671929e-06, -1.19997437e-06, -7.23229447e-07],
[-1.55052065e-06, -2.11447947e-06, -2.67843828e-06],
[-1.67837384e-06, -1.19987110e-06, -7.21368357e-07],
[-1.53704818e-06, -2.11449802e-06, -2.69194785e-06],
[-1.67999120e-06, -1.19995499e-06, -7.19918777e-07],
[-1.53677300e-06, -2.11451134e-06, -2.69224967e-06],
[-1.08606610e-05, -1.13144516e-05, -1.17682423e-05],
[-1.09722592e-05, -1.04184563e-05, -9.86465338e-06],
[-1.08605636e-05, -1.13142583e-05, -1.17679530e-05],
[-1.09731814e-05, -1.04184253e-05, -9.86366928e-06],
[-1.08594297e-05, -1.13133967e-05, -1.17673637e-05],
[-1.09722874e-05, -1.04181226e-05, -9.86395772e-06],
[-1.08593777e-05, -1.13133005e-05, -1.17672234e-05],
[-1.09717384e-05, -1.04184447e-05, -9.86515098e-06],
[-1.08608899e-05, -1.13143378e-05, -1.17677856e-05],
[-1.09744873e-05, -1.04184127e-05, -9.86233813e-06],
[-1.08617067e-05, -1.13143787e-05, -1.17670506e-05],
[-1.09723855e-05, -1.04184033e-05, -9.86442104e-06],
[-1.08631117e-05, -1.13142768e-05, -1.17654419e-05],
[-1.09726226e-05, -1.04184113e-05, -9.86419995e-06],
[-1.08596183e-05, -1.13142505e-05, -1.17688828e-05],
[-1.09699690e-05, -1.04183668e-05, -9.86676463e-06],
[-1.75748529e-07,  5.05669283e-08,  2.76882386e-07],
[-1.06865040e-07, -4.69043094e-07, -8.31221147e-07],
[-1.75471928e-07,  5.05830770e-08,  2.76638082e-07],
[-1.06967368e-07, -4.68911090e-07, -8.30854812e-07],
[-1.75784195e-07,  5.03354340e-08,  2.76455063e-07],
[-1.07040824e-07, -4.68972763e-07, -8.30904702e-07],
[-1.70339706e-07,  5.04875380e-08,  2.71314782e-07],
[-1.10264934e-07, -4.68982663e-07, -8.27700392e-07],
[-1.75722224e-07,  5.05409737e-08,  2.76804171e-07],
[-1.06938486e-07, -4.69026461e-07, -8.31114436e-07],
[-1.74262333e-07,  5.05089630e-08,  2.75280258e-07],
[-1.06699630e-07, -4.68974441e-07, -8.31249252e-07],
[-1.75684105e-07,  5.05785512e-08,  2.76841208e-07],
[-1.06636583e-07, -4.68839715e-07, -8.31042848e-07],
[-1.75774736e-07,  5.04176444e-08,  2.76610025e-07],
[-1.07062899e-07, -4.68964689e-07, -8.30866480e-07],
[-1.22917423e-05, -1.24994620e-05, -1.27071817e-05],
[-1.23538813e-05, -1.20165529e-05, -1.16792245e-05],
[-1.22917067e-05, -1.24994983e-05, -1.27072899e-05],
[-1.23536752e-05, -1.20166234e-05, -1.16795715e-05],
```

```
[-1.22931391e-05, -1.25002018e-05, -1.27072645e-05],
[-1.23546238e-05, -1.20173397e-05, -1.16800555e-05],
[-1.22938834e-05, -1.25003096e-05, -1.27067357e-05],
[-1.23538229e-05, -1.20171895e-05, -1.16805561e-05],
[-1.22939449e-05, -1.24994582e-05, -1.27049715e-05],
[-1.23497605e-05, -1.20166126e-05, -1.16834647e-05],
[-1.22940631e-05, -1.24994347e-05, -1.27048062e-05],
[-1.23462444e-05, -1.20166004e-05, -1.16869564e-05],
[-1.22921211e-05, -1.24994560e-05, -1.27067908e-05],
[-1.23536730e-05, -1.20166800e-05, -1.16796870e-05],
[-1.22961004e-05, -1.24994029e-05, -1.27027054e-05],
[-1.23510621e-05, -1.20166963e-05, -1.16823306e-05],
[-3.68683640e-06, -4.38856278e-06, -5.09028916e-06],
[-3.84584281e-06, -3.19366021e-06, -2.54147760e-06],
[-8.87702104e-06, -8.18697146e-06, -7.49692189e-06],
[-8.72854808e-06, -9.36265082e-06, -9.99675356e-06],
[-3.68651662e-06, -4.38861848e-06, -5.09072034e-06],
[-3.84551313e-06, -3.19369117e-06, -2.54186920e-06],
[-8.87769653e-06, -8.18690128e-06, -7.49610602e-06],
[-8.72854081e-06, -9.36343921e-06, -9.99833761e-06],
[-3.68672512e-06, -4.38859086e-06, -5.09045660e-06],
[-3.84569301e-06, -3.19361312e-06, -2.54153323e-06],
[-8.87751161e-06, -8.18689971e-06, -7.49628781e-06],
[-8.72765988e-06, -9.36285315e-06, -9.99804642e-06],
[-3.69149455e-06, -4.38858079e-06, -5.08566703e-06],
[-3.83675894e-06, -3.19362219e-06, -2.55048544e-06],
[-8.87435670e-06, -8.18689541e-06, -7.49943412e-06],
[-8.72862893e-06, -9.36284958e-06, -9.99707023e-06],
[-3.69655392e-06, -4.38863533e-06, -5.08071675e-06],
[-3.84313810e-06, -3.19365705e-06, -2.54417601e-06],
[-8.87341331e-06, -8.18693903e-06, -7.50046474e-06],
[-8.74427766e-06, -9.36279077e-06, -9.98130388e-06],
[-3.70319708e-06, -4.38864473e-06, -5.07409237e-06],
[-3.84555172e-06, -3.19356834e-06, -2.54158497e-06],
[-8.87733065e-06, -8.18686637e-06, -7.49640209e-06],
[-8.72789323e-06, -9.36295745e-06, -9.99802166e-06],
[-3.68705374e-06, -4.38878679e-06, -5.09051985e-06],
[-3.84529332e-06, -3.19352524e-06, -2.54175716e-06],
[-8.87813534e-06, -8.18676934e-06, -7.49540334e-06],
[-8.73007065e-06, -9.36338384e-06, -9.99669702e-06],
[-3.68661073e-06, -4.38859980e-06, -5.09058886e-06],
[-3.84592435e-06, -3.19360343e-06, -2.54128251e-06],
[-8.87704131e-06, -8.18684212e-06, -7.49664294e-06],
[-8.72889146e-06, -9.36297530e-06, -9.99705913e-06]])
```

[162]:
```
# ### 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')
```

2.3 Example 2

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

where

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

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

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

[173]:
```
@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 /
```

```python
                                    (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)

[176]:
```
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 = []
    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  1.52  1.14  0.76  0.74
2^-3  1.82  1.76  0.94  0.91
2^-4  1.95  1.92  0.99  0.97
2^-5  1.99  1.98  1.00  0.99
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.12  0.98  -0.66  0.89
```

```
2^-3  0.68  1.06  -0.27  0.71
2^-4  0.79  1.10  0.24  0.57
2^-5  1.25  1.29  0.57  0.69
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.29  0.63  -0.48  0.60
2^-3  1.39  0.54  -0.51  0.52
2^-4  1.47  0.51  -0.50  0.50
2^-5  1.48  0.51  -0.50  0.50
epsilon = 1e-06
  h    L2u   H1u   H2u   epu
2^-2  1.29  0.63  -0.48  0.60
2^-3  1.38  0.54  -0.51  0.52
2^-4  1.46  0.51  -0.50  0.50
2^-5  1.49  0.51  -0.50  0.50
```

2.3.2 With penalty (Problem2)

[181]:
```
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.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
        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)
```

```python
    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

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

    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  2.07  1.95  1.24  1.19
2^-3  1.72  1.80  1.09  1.06
2^-4  1.84  1.88  1.02  1.00
2^-5  1.92  1.94  1.00  0.99
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  0.81  0.50  -0.83  0.42
2^-3  0.90  0.64  -0.34  0.52
2^-4  1.10  0.78  0.01  0.57
2^-5  1.17  0.93  0.19  0.58
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  0.74  0.42  -0.50  0.36
2^-3  0.78  0.50  -0.44  0.46
2^-4  0.91  0.53  -0.47  0.48
2^-5  0.97  0.53  -0.49  0.49
epsilon = 1e-06
  h    L2u   H1u   H2u   epu
2^-2  0.74  0.42  -0.50  0.36
2^-3  0.78  0.50  -0.44  0.46
```

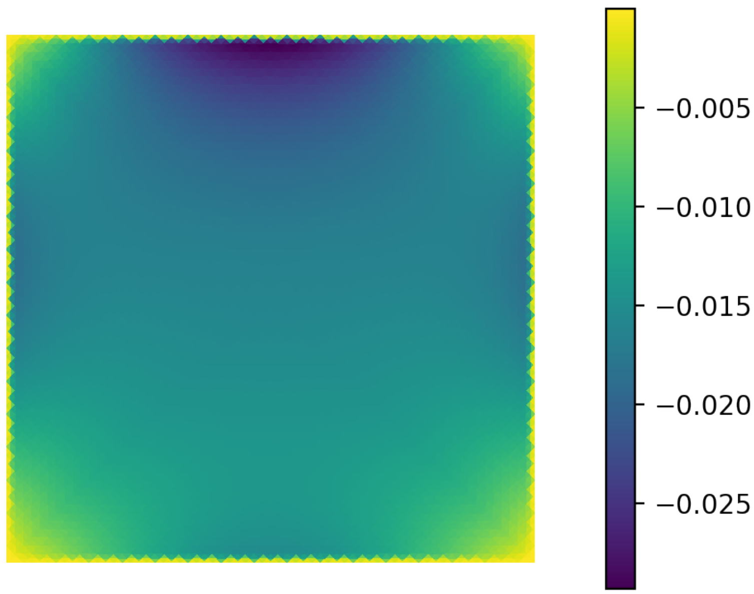
```
2^-4  0.91  0.53  -0.47  0.48
2^-5  0.96  0.53  -0.49  0.49
```

2.3.3 Analysising result uh_0 with and without penalty

uh0_penalty-uh0_no_penalty

[182]:
```python
plot(basis['u'], uh0_penalty-uh0_no_penalty, colorbar=True)
```

[182]: <matplotlib.axes._subplots.AxesSubplot at 0x1e9b3545b08>

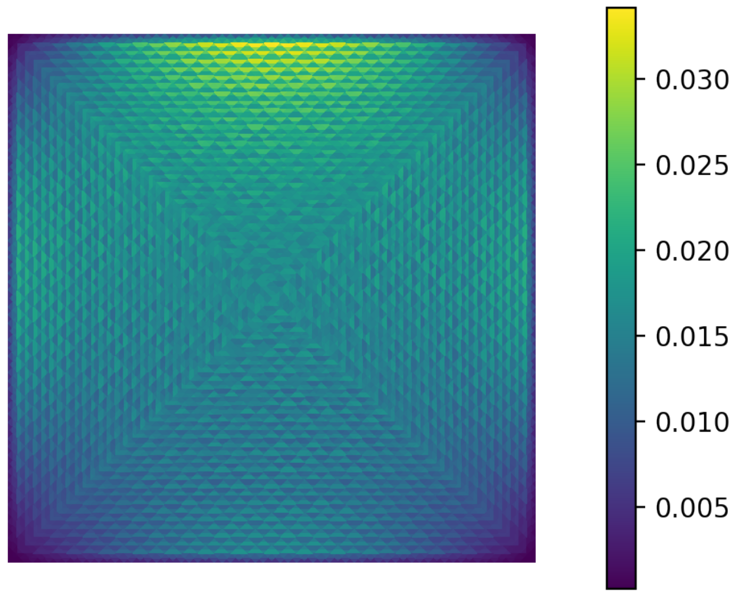


u-uh0_penalty

[183]:
```python
x = basis['u'].doflocs[0]
y = basis['u'].doflocs[1]
u = exact_u(x, y)

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

[183]: <matplotlib.axes._subplots.AxesSubplot at 0x1e9c0effb88>



Value of `uh0_penalty` on boundary nodes

[184]:
```
uh0_penalty[m.boundary_nodes()]
```

[184]:
```
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., 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.])
```

[185]:
```
m = MeshTri.init_symmetric()
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

[186]:
```
dot(fbasis_dof.interpolate(uh0_no_penalty).grad, mem.n)
```

[186]:
```
array([[ 4.90335655e-02,  0.00000000e+00, -4.90335655e-02],
       [ 4.90799502e-02,  0.00000000e+00, -4.90799502e-02],
       [ 4.90335655e-02,  0.00000000e+00, -4.90335655e-02],
       [ 4.90799502e-02,  0.00000000e+00, -4.90799502e-02],
       [ 9.28847628e-02, -9.33361060e-16, -9.28847628e-02],
       [ 9.60331552e-02, -9.64997970e-16, -9.60331552e-02],
       [ 9.28847628e-02,  0.00000000e+00, -9.28847628e-02],
       [ 9.60331552e-02, -4.82498985e-16, -9.60331552e-02],
       [-2.40221159e-03,  0.00000000e+00,  2.40221159e-03],
       [-2.40221159e-03,  0.00000000e+00,  2.40221159e-03],
       [-1.44835149e-02,  0.00000000e+00,  1.44835149e-02],
       [ 9.88742273e-03,  0.00000000e+00, -9.88742273e-03],
       [-1.44835149e-02,  0.00000000e+00,  1.44835149e-02],
       [ 9.88742273e-03, -5.91180348e-15, -9.88742273e-03],
       [-4.69528652e-03, -4.90198389e-15,  4.69528652e-03],
       [-4.69528652e-03,  4.87839339e-15,  4.69528652e-03],
       [-3.62899124e-02,  0.00000000e+00,  3.62899124e-02],
       [ 3.28884327e-02,  0.00000000e+00, -3.28884327e-02],
       [-4.09229316e-02,  0.00000000e+00,  4.09229316e-02],
       [ 3.86213890e-02,  0.00000000e+00, -3.86213890e-02],
       [-3.62899124e-02,  0.00000000e+00,  3.62899124e-02],
       [ 3.28884327e-02,  0.00000000e+00, -3.28884327e-02],
       [-4.09229316e-02,  0.00000000e+00,  4.09229316e-02],
       [ 3.86213890e-02, -3.88090570e-16, -3.86213890e-02],
       [-3.72364520e-02, -5.17075590e-15,  3.72364520e-02],
       [ 3.03383716e-02, -5.47561380e-15, -3.03383716e-02],
       [-7.09065752e-02,  6.93404202e-15,  7.09065752e-02],
       [ 6.42651338e-02,  0.00000000e+00, -6.42651338e-02],
       [-3.72364520e-02,  0.00000000e+00,  3.72364520e-02],
       [ 3.03383716e-02,  0.00000000e+00, -3.03383716e-02],
       [-7.09065752e-02,  0.00000000e+00,  7.09065752e-02],
       [ 6.42651338e-02,  7.25692905e-15, -6.42651338e-02],
       [-4.61411866e-02,  0.00000000e+00,  4.61411866e-02],
       [ 4.42921529e-02,  0.00000000e+00, -4.42921529e-02],
       [-4.72692382e-02,  0.00000000e+00,  4.72692382e-02],
       [ 4.61108147e-02,  0.00000000e+00, -4.61108147e-02],
       [-4.61411866e-02,  0.00000000e+00,  4.61411866e-02],
       [ 4.42921529e-02,  0.00000000e+00, -4.42921529e-02],
       [-4.72692382e-02,  0.00000000e+00,  4.72692382e-02],
       [ 4.61108147e-02,  0.00000000e+00, -4.61108147e-02],
```

```
[-6.77151095e-02,  0.00000000e+00,  6.77151095e-02],
[ 5.96618183e-02, -3.82869731e-15, -5.96618183e-02],
[-9.01123686e-02, -3.75495806e-15,  9.01123686e-02],
[ 8.65145196e-02,  4.62430713e-15, -8.65145196e-02],
[-6.77151095e-02,  0.00000000e+00,  6.77151095e-02],
[ 5.96618183e-02,  0.00000000e+00, -5.96618183e-02],
[-9.01123686e-02, -3.39563401e-16,  9.01123686e-02],
[ 8.65145196e-02,  1.08668636e-16, -8.65145196e-02],
[ 1.64945756e-02,  0.00000000e+00, -1.64945756e-02],
[-2.09347877e-02,  0.00000000e+00,  2.09347877e-02],
[ 1.64945756e-02,  0.00000000e+00, -1.64945756e-02],
[-2.09347877e-02,  0.00000000e+00,  2.09347877e-02],
[ 2.65492148e-02,  0.00000000e+00, -2.65492148e-02],
[-2.99974852e-02,  0.00000000e+00,  2.99974852e-02],
[ 5.62573665e-03,  0.00000000e+00, -5.62573665e-03],
[-1.13722283e-02,  0.00000000e+00,  1.13722283e-02],
[ 2.65492148e-02, -5.08139489e-15, -2.65492148e-02],
[-2.99974852e-02,  0.00000000e+00,  2.99974852e-02],
[ 5.62573665e-03,  0.00000000e+00, -5.62573665e-03],
[-1.13722283e-02, -6.00323220e-15,  1.13722283e-02],
[ 3.22375333e-02, -9.38205912e-15, -3.22375333e-02],
[-4.09139438e-02, -4.52905866e-15,  4.09139438e-02],
[ 3.22375333e-02,  0.00000000e+00, -3.22375333e-02],
[-4.09139438e-02,  4.32349490e-15,  4.09139438e-02],
[-4.85087749e-02,  0.00000000e+00,  4.85087749e-02],
[ 4.75497752e-02,  0.00000000e+00, -4.75497752e-02],
[-4.87393669e-02,  0.00000000e+00,  4.87393669e-02],
[ 4.81439671e-02,  0.00000000e+00, -4.81439671e-02],
[-4.85087749e-02,  0.00000000e+00,  4.85087749e-02],
[ 4.75497752e-02,  0.00000000e+00, -4.75497752e-02],
[-4.87393669e-02,  4.93184376e-16,  4.87393669e-02],
[ 4.81439671e-02,  0.00000000e+00, -4.81439671e-02],
[-8.46975576e-02,  0.00000000e+00,  8.46975576e-02],
[ 7.60590524e-02,  0.00000000e+00, -7.60590524e-02],
[-9.46896127e-02, -1.91649522e-15,  9.46896127e-02],
[ 9.28448968e-02,  0.00000000e+00, -9.28448968e-02],
[-8.46975576e-02,  0.00000000e+00,  8.46975576e-02],
[ 7.60590524e-02,  0.00000000e+00, -7.60590524e-02],
[-9.46896127e-02,  3.56811472e-16,  9.46896127e-02],
[ 9.28448968e-02,  0.00000000e+00, -9.28448968e-02],
[ 7.18360361e-03,  0.00000000e+00, -7.18360361e-03],
[-1.18960778e-02,  0.00000000e+00,  1.18960778e-02],
[ 7.18360361e-03,  0.00000000e+00, -7.18360361e-03],
[-1.18960778e-02,  0.00000000e+00,  1.18960778e-02],
[ 1.87923899e-02,  0.00000000e+00, -1.87923899e-02],
[-2.28141821e-02,  0.00000000e+00,  2.28141821e-02],
[-5.00396458e-03,  0.00000000e+00,  5.00396458e-03],
```

```
[-1.66990164e-04,  0.00000000e+00,  1.66990164e-04],
[ 1.87923899e-02, -5.57742750e-15, -1.87923899e-02],
[-2.28141821e-02,  5.34817710e-15,  2.28141821e-02],
[-5.00396458e-03,  6.01115815e-15,  5.00396458e-03],
[-1.66990164e-04,  6.06144095e-15,  1.66990164e-04],
[ 1.40406625e-02,  0.00000000e+00, -1.40406625e-02],
[-2.32508971e-02, -4.80784894e-15,  2.32508971e-02],
[ 1.40406625e-02,  9.61569789e-15, -1.40406625e-02],
[-2.32508971e-02,  4.69102956e-15,  2.32508971e-02],
[ 3.93429705e-02,  0.00000000e+00, -3.93429705e-02],
[-4.20188735e-02,  0.00000000e+00,  4.20188735e-02],
[-2.91710279e-02,  0.00000000e+00,  2.91710279e-02],
[ 2.51737281e-02,  0.00000000e+00, -2.51737281e-02],
[ 4.29379681e-02,  0.00000000e+00, -4.29379681e-02],
[-4.46671263e-02,  0.00000000e+00,  4.46671263e-02],
[-3.60333000e-02,  0.00000000e+00,  3.60333000e-02],
[ 3.31587593e-02,  0.00000000e+00, -3.31587593e-02],
[ 3.93429705e-02,  0.00000000e+00, -3.93429705e-02],
[-4.20188735e-02,  0.00000000e+00,  4.20188735e-02],
[-2.91710279e-02,  0.00000000e+00,  2.91710279e-02],
[ 2.51737281e-02,  0.00000000e+00, -2.51737281e-02],
[ 4.29379681e-02,  3.28537127e-15, -4.29379681e-02],
[-4.46671263e-02,  2.85390517e-15,  4.46671263e-02],
[-3.60333000e-02,  0.00000000e+00,  3.60333000e-02],
[ 3.31587593e-02,  0.00000000e+00, -3.31587593e-02],
[ 4.44226962e-02, -4.79658200e-15, -4.44226962e-02],
[-5.18977362e-02,  0.00000000e+00,  5.18977362e-02],
[-2.37282263e-02, -5.71404906e-15,  2.37282263e-02],
[ 1.74062472e-02,  5.88895733e-15, -1.74062472e-02],
[ 7.68653567e-02,  6.22153080e-15, -7.68653567e-02],
[-8.20846461e-02,  0.00000000e+00,  8.20846461e-02],
[-5.70050248e-02,  0.00000000e+00,  5.70050248e-02],
[ 4.91962963e-02, -4.32349490e-15, -4.91962963e-02],
[ 4.44226962e-02,  0.00000000e+00, -4.44226962e-02],
[-5.18977362e-02,  0.00000000e+00,  5.18977362e-02],
[-2.37282263e-02,  0.00000000e+00,  2.37282263e-02],
[ 1.74062472e-02,  0.00000000e+00, -1.74062472e-02],
[ 7.68653567e-02,  1.93097147e-16, -7.68653567e-02],
[-8.20846461e-02, -2.06208774e-16,  8.20846461e-02],
[-5.70050248e-02,  1.43205055e-16,  5.70050248e-02],
[ 4.91962963e-02, -1.23588374e-16, -4.91962963e-02]])
```

∂u_n of `uh0_penalty` on boundary nodes

[187]:
```
dot(fbasis_dof.interpolate(uh0_penalty).grad, mem.n)
```

```
[187]: array([[ 1.50961330e-05,  2.50999023e-06, -1.00761525e-05],
              [ 1.50969121e-05,  2.50624285e-06, -1.00844264e-05],
              [ 1.50961390e-05,  2.50999619e-06, -1.00761466e-05],
              [ 1.50969061e-05,  2.50623686e-06, -1.00844324e-05],
              [ 2.67380762e-05,  3.90989518e-06, -1.89182858e-05],
              [ 2.67884161e-05,  3.65277443e-06, -1.94828672e-05],
              [ 2.67370672e-05,  3.90885054e-06, -1.89193661e-05],
              [ 2.67894253e-05,  3.65381930e-06, -1.94817867e-05],
              [-8.25528504e-05, -8.23182823e-05, -8.20837142e-05],
              [-8.25528505e-05, -8.23182822e-05, -8.20837140e-05],
              [-9.73363558e-05, -9.59220902e-05, -9.45078246e-05],
              [-9.70161335e-05, -9.79816066e-05, -9.89470797e-05],
              [-9.73363509e-05, -9.59220831e-05, -9.45078153e-05],
              [-9.70161401e-05, -9.79816121e-05, -9.89470841e-05],
              [-1.62775473e-04, -1.62316993e-04, -1.61858513e-04],
              [-1.62775470e-04, -1.62316993e-04, -1.61858516e-04],
              [-5.80593532e-05, -5.45157620e-05, -5.09721709e-05],
              [-5.72125115e-05, -6.04239595e-05, -6.36354074e-05],
              [-6.03650271e-05, -5.63690370e-05, -5.23730468e-05],
              [-5.93448888e-05, -6.31161438e-05, -6.68873988e-05],
              [-5.80593535e-05, -5.45157620e-05, -5.09721704e-05],
              [-5.72125114e-05, -6.04239594e-05, -6.36354074e-05],
              [-6.03650361e-05, -5.63690362e-05, -5.23730362e-05],
              [-5.93448866e-05, -6.31161390e-05, -6.68873913e-05],
              [-8.49701169e-05, -8.13340907e-05, -7.76980644e-05],
              [-8.41596586e-05, -8.71220090e-05, -9.00843594e-05],
              [-1.14925228e-04, -1.08001473e-04, -1.01077719e-04],
              [-1.13272264e-04, -1.19547544e-04, -1.25822824e-04],
              [-8.49700931e-05, -8.13340858e-05, -7.76980784e-05],
              [-8.41595591e-05, -8.71220020e-05, -9.00844449e-05],
              [-1.14925269e-04, -1.08001473e-04, -1.01077678e-04],
              [-1.13272302e-04, -1.19547544e-04, -1.25822787e-04],
              [-3.03833586e-05, -2.58778228e-05, -2.13722871e-05],
              [-2.93398094e-05, -3.36647928e-05, -3.79897761e-05],
              [-3.06551595e-05, -2.60394730e-05, -2.14237864e-05],
              [-2.95345865e-05, -3.40371552e-05, -3.85397238e-05],
              [-3.03833599e-05, -2.58778228e-05, -2.13722857e-05],
              [-2.93398100e-05, -3.36647929e-05, -3.79897757e-05],
              [-3.06551597e-05, -2.60394747e-05, -2.14237897e-05],
              [-2.95345862e-05, -3.40371554e-05, -3.85397246e-05],
              [-5.18721331e-05, -4.52599932e-05, -3.86478534e-05],
              [-5.05293503e-05, -5.63549481e-05, -6.21805458e-05],
              [-6.07374718e-05, -5.19383196e-05, -4.31391675e-05],
              [-5.87611607e-05, -6.72089703e-05, -7.56567800e-05],
              [-5.18721365e-05, -4.52599956e-05, -3.86478547e-05],
              [-5.05291663e-05, -5.63549476e-05, -6.21807288e-05],
              [-6.07374968e-05, -5.19383196e-05, -4.31391423e-05],
```

```
[-5.87611101e-05, -6.72089703e-05, -7.56568305e-05],
[-7.58532670e-05, -7.74639086e-05, -7.90745503e-05],
[-7.63217573e-05, -7.42775436e-05, -7.22333298e-05],
[-7.58532672e-05, -7.74639086e-05, -7.90745500e-05],
[-7.63217570e-05, -7.42775437e-05, -7.22333303e-05],
[-8.28284339e-05, -8.54208740e-05, -8.80133142e-05],
[-8.35690665e-05, -8.06399101e-05, -7.77107538e-05],
[-9.86014737e-05, -9.91508078e-05, -9.97001418e-05],
[-9.88137309e-05, -9.77032713e-05, -9.65928117e-05],
[-8.28284259e-05, -8.54208758e-05, -8.80133257e-05],
[-8.35690698e-05, -8.06399066e-05, -7.77107434e-05],
[-9.86014687e-05, -9.91508033e-05, -9.97001379e-05],
[-9.88137356e-05, -9.77032729e-05, -9.65928101e-05],
[-1.49687249e-04, -1.52835130e-04, -1.55983011e-04],
[-1.50606752e-04, -1.46611625e-04, -1.42616499e-04],
[-1.49687240e-04, -1.52835130e-04, -1.55983020e-04],
[-1.50606720e-04, -1.46611625e-04, -1.42616531e-04],
[-1.44713524e-05, -9.73462916e-06, -4.99790594e-06],
[-1.31715625e-05, -1.78146409e-05, -2.24577192e-05],
[-1.45029657e-05, -9.74372749e-06, -4.98448923e-06],
[-1.31642611e-05, -1.78653631e-05, -2.25664650e-05],
[-1.44713514e-05, -9.73462906e-06, -4.99790675e-06],
[-1.31715577e-05, -1.78146407e-05, -2.24577237e-05],
[-1.45029373e-05, -9.74372510e-06, -4.98451286e-06],
[-1.31642809e-05, -1.78653635e-05, -2.25664461e-05],
[-2.71285236e-05, -1.88580839e-05, -1.05876441e-05],
[-2.52667634e-05, -3.26936848e-05, -4.01206062e-05],
[-2.92815169e-05, -2.00354054e-05, -1.07892939e-05],
[-2.70082603e-05, -3.60742591e-05, -4.51402578e-05],
[-2.71285491e-05, -1.88580830e-05, -1.05876169e-05],
[-2.52667711e-05, -3.26936861e-05, -4.01206010e-05],
[-2.92815349e-05, -2.00354054e-05, -1.07892760e-05],
[-2.70082586e-05, -3.60742591e-05, -4.51402597e-05],
[-8.08015491e-05, -8.15030046e-05, -8.22044601e-05],
[-8.10414637e-05, -7.98798507e-05, -7.87182376e-05],
[-8.08015494e-05, -8.15030045e-05, -8.22044597e-05],
[-8.10414640e-05, -7.98798506e-05, -7.87182372e-05],
[-9.12749063e-05, -9.31099221e-05, -9.49449378e-05],
[-9.18211727e-05, -8.95934458e-05, -8.73657189e-05],
[-9.97280941e-05, -9.92394737e-05, -9.87508534e-05],
[-9.96622319e-05, -9.96459259e-05, -9.96296199e-05],
[-9.12749054e-05, -9.31099164e-05, -9.49449275e-05],
[-9.18211681e-05, -8.95934510e-05, -8.73657339e-05],
[-9.97280973e-05, -9.92394790e-05, -9.87508606e-05],
[-9.96622260e-05, -9.96459199e-05, -9.96296139e-05],
[-1.59353662e-04, -1.60724688e-04, -1.62095713e-04],
[-1.59824581e-04, -1.57554199e-04, -1.55283818e-04],
```

```
[-1.59353671e-04, -1.60724688e-04, -1.62095705e-04],
[-1.59824562e-04, -1.57554199e-04, -1.55283836e-04],
[-4.42055535e-05, -4.80472659e-05, -5.18889783e-05],
[-4.51792014e-05, -4.10761962e-05, -3.69731910e-05],
[-6.85667044e-05, -6.57182489e-05, -6.28697934e-05],
[-6.78921371e-05, -7.03502693e-05, -7.28084016e-05],
[-4.50515692e-05, -4.92443216e-05, -5.34370739e-05],
[-4.61475240e-05, -4.17859258e-05, -3.74243276e-05],
[-7.29595582e-05, -6.94410224e-05, -6.59224865e-05],
[-7.20598732e-05, -7.52977196e-05, -7.85355661e-05],
[-4.42055537e-05, -4.80472659e-05, -5.18889782e-05],
[-4.51792018e-05, -4.10761963e-05, -3.69731908e-05],
[-6.85667041e-05, -6.57182490e-05, -6.28697938e-05],
[-6.78921370e-05, -7.03502693e-05, -7.28084016e-05],
[-4.50515896e-05, -4.92443247e-05, -5.34370598e-05],
[-4.61475002e-05, -4.17859268e-05, -3.74243534e-05],
[-7.29595484e-05, -6.94410247e-05, -6.59225009e-05],
[-7.20598716e-05, -7.52977233e-05, -7.85355751e-05],
[-6.99818666e-05, -7.43195403e-05, -7.86572140e-05],
[-7.10833062e-05, -6.60156763e-05, -6.09480464e-05],
[-9.40594662e-05, -9.17424882e-05, -8.94255103e-05],
[-9.35518463e-05, -9.52514994e-05, -9.69511526e-05],
[-8.78499927e-05, -9.53555988e-05, -1.02861205e-04],
[-8.97351033e-05, -8.17198058e-05, -7.37045083e-05],
[-1.35457899e-04, -1.29891545e-04, -1.24325190e-04],
[-1.34136537e-04, -1.38940410e-04, -1.43744283e-04],
[-6.99818100e-05, -7.43195413e-05, -7.86572726e-05],
[-7.10833206e-05, -6.60156761e-05, -6.09480317e-05],
[-9.40594741e-05, -9.17424901e-05, -8.94255061e-05],
[-9.35518417e-05, -9.52515023e-05, -9.69511628e-05],
[-8.78499481e-05, -9.53555988e-05, -1.02861249e-04],
[-8.97351027e-05, -8.17198057e-05, -7.37045087e-05],
[-1.35457899e-04, -1.29891545e-04, -1.24325190e-04],
[-1.34136553e-04, -1.38940410e-04, -1.43744267e-04]])
```

[157]:
```
# ### Analysisising results $u$ and $u_{n}$ with penalty

# # x = mem.x[0]
# # y = mem.x[1]
# plot(basis['u'], uh0-uh1, colorbar=True)

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

# uh0[m.boundary_nodes()]
```

```python
# m = MeshTri.init_symmetric()
# m.refine(2)

# 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)

# for i in range(mem.x.shape[1]):
#     plt.scatter(mem.x[0][i], mem.x[1][i], s=10, marker='*')
#     plt.axis('square')

# dot(fbasis.interpolate(uh1).grad, mem.n)

# dot(fbasis.interpolate(uh0).grad, mem.n)
```