Perturb_Problem2

October 22, 2020

1 Solving a Fourth Order Elliptic Singular Perturbation Problem

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

[1]:
```
from skfem import *
import numpy as np
from skfem.visuals.matplotlib import draw, plot
from skfem.utils import solver_iter_krylov
from skfem.helpers import d, dd, ddd, dot, ddot, grad, dddot, prod
from scipy.sparse.linalg import LinearOperator, minres
from skfem import *
from skfem.models.poisson import *
from skfem.assembly import BilinearForm, LinearForm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
plt.rcParams['figure.dpi'] = 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

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

1.3 Forms and errors

[2]:

```
@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

[3]:
```python
def easy_boundary(basis):
    '''
    Input basis
    ----------------
    Return D for boundary conditions
    '''

    dofs = basis.find_dofs({
        'left': m.facets_satisfying(lambda x: x[0] == 0),
```

```
        'right': m.facets_satisfying(lambda x: x[0] == 1),
        'top': m.facets_satisfying(lambda x: x[1] == 1),
        'buttom': m.facets_satisfying(lambda x: x[1] == 0)
    })

    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_type='P1'):
    if element_type == 'P1':
        element = {'w': ElementTriP1(), 'u': ElementTriMorley()}
    elif element_type == 'P2':
        element = {'w': ElementTriP2(), 'u': ElementTriMorley()}
    else:
        raise Exception("The element not supported")

    basis = {
        variable: InteriorBasis(m, e, intorder=5)
        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=basis['w'].find_dofs()),
               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, tol=1e-8))  # cg
    return uh0, basis
```

1.5 Solver for problem2

[4]:
```
def easy_boundary_penalty(basis):
    '''
    Input basis
    ----------------
    Return D for boundary conditions
    '''
```

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

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


def solve_problem2(m, element_type='P1'):
    if element_type == 'P1':
        element = {'w': ElementTriP1(), 'u': ElementTriMorley()}
    elif element_type == 'P2':
        element = {'w': ElementTriP2(), 'u': ElementTriMorley()}
    else:
        raise Exception("The element not supported")

    basis = {
        variable: InteriorBasis(m, e, intorder=5)
        for variable, e in element.items()
    }

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

    wh = solve(*condense(K1, f1, D=basis['w'].find_dofs()),
               solver=solver_iter_krylov(Precondition=True, tol=1e-8))

    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, tol=1e-8))
    # 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

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

[5]:

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


def exact_u(x, y):
    return (sin(pi * x) * sin(pi * y))**2


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


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

2.2.1 P1 element

Without penalty (Problem1)

[6]:
```
refine_time = 6
epsilon_range = 5
element_type = 'P1'
print('element_type:', element_type)
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, element_type)
        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]))
```

```
element_type: P1
epsilon = 1
  h     L2u   H1u   H2u   epu
2^-2  1.83  0.87  0.71  0.70
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  1.45  0.74  0.48  0.66
2^-3  2.26  1.68  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  1.45  0.73  0.47  0.65
2^-3  2.29  1.66  0.81  1.61
2^-4  2.31  1.89  0.94  1.87
2^-5  2.14  1.97  0.98  1.96
2^-6  2.04  1.99  1.00  1.99
epsilon = 1e-06
  h     L2u   H1u   H2u   epu
2^-2  1.45  0.73  0.47  0.65
2^-3  2.29  1.66  0.81  1.61
2^-4  2.31  1.89  0.94  1.87
2^-5  2.14  1.97  0.98  1.96
2^-6  2.04  1.99  1.00  1.99
epsilon = 1e-08
  h     L2u   H1u   H2u   epu
2^-2  1.45  0.73  0.47  0.65
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
```

With penalty (Problem2)

[7]:

```python
sigma = 5
element_type = 'P1'
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, element_type)
        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]))
```

```
epsilon = 1
  h     L2u   H1u   H2u   epu
2^-2  1.72  0.82  0.73  0.72
2^-3  2.25  1.80  1.07  1.02
2^-4  2.24  1.95  1.04  1.01
2^-5  2.11  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  1.28  0.55  0.23  0.47
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  1.27  0.54  0.22  0.46
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
epsilon = 1e-06
  h     L2u   H1u   H2u   epu
2^-2  1.27  0.54  0.22  0.46
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
epsilon = 1e-08
  h     L2u   H1u   H2u   epu
2^-2  1.27  0.54  0.22  0.46
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
```

2.2.2 P2 element

Without penalty (Problem1)

[8]:
```
refine_time = 6
epsilon_range = 5
element_type = 'P2'
```

```python
print('element_type:', element_type)
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, element_type)
        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
```

```
element_type: P2
epsilon = 1
  h     L2u   H1u   H2u   epu
2^-2  1.82  1.71  1.03  0.98
2^-3  1.73  1.74  0.94  0.90
2^-4  1.90  1.88  0.99  0.97
2^-5  1.97  1.96  1.00  0.99
2^-6  1.99  1.99  1.00  1.00
epsilon = 0.01
  h     L2u   H1u   H2u   epu
2^-2  2.83  1.88  0.93  1.75
2^-3  2.62  1.99  0.91  1.81
2^-4  1.11  2.04  1.08  1.71
2^-5  0.92  1.72  1.05  1.30
2^-6  1.47  1.60  0.90  1.00
epsilon = 0.0001
  h     L2u   H1u   H2u   epu
2^-2  2.86  1.87  0.91  1.77
2^-3  3.06  1.92  0.85  1.87
2^-4  3.36  1.94  0.93  1.91
2^-5  3.28  1.98  0.98  1.97
2^-6  3.12  1.99  0.99  1.99
epsilon = 1e-06
  h     L2u   H1u   H2u   epu
2^-2  2.86  1.87  0.91  1.77
2^-3  3.06  1.92  0.85  1.87
2^-4  3.37  1.94  0.93  1.91
2^-5  3.28  1.98  0.98  1.97
2^-6  3.13  1.99  0.99  1.99
epsilon = 1e-08
  h     L2u   H1u   H2u   epu
2^-2  2.86  1.87  0.91  1.77
2^-3  3.06  1.92  0.85  1.87
2^-4  3.37  1.94  0.93  1.91
2^-5  3.28  1.98  0.98  1.97
2^-6  3.13  1.99  0.99  1.99
```

With penalty (Problem2)

[9]:
```
sigma = 5
element_type = 'P2'
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, element_type)
        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_penalty = uh0
```

```
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  1.81  1.59  1.00  0.95
2^-3  1.57  1.67  0.91  0.87
2^-4  1.80  1.82  0.97  0.95
2^-5  1.91  1.91  0.99  0.98
2^-6  1.96  1.96  1.00  0.99
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  2.98  1.84  0.94  1.69
2^-3  2.59  1.99  0.93  1.81
2^-4  1.06  2.05  1.10  1.72
2^-5  0.91  1.73  1.06  1.30
2^-6  1.46  1.59  0.90  1.00
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  3.01  1.83  0.93  1.70
2^-3  3.07  1.91  0.87  1.86
2^-4  3.36  1.94  0.94  1.92
2^-5  3.25  1.98  0.99  1.97
2^-6  3.09  2.00  1.00  1.99
epsilon = 1e-06
  h    L2u   H1u   H2u   epu
2^-2  3.01  1.83  0.93  1.70
2^-3  3.07  1.91  0.87  1.86
2^-4  3.36  1.94  0.94  1.92
2^-5  3.26  1.98  0.99  1.97
2^-6  3.10  2.00  1.00  1.99
epsilon = 1e-08
  h    L2u   H1u   H2u   epu
2^-2  3.01  1.83  0.93  1.70
2^-3  3.07  1.91  0.87  1.86
2^-4  3.36  1.94  0.94  1.92
2^-5  3.26  1.98  0.99  1.97
2^-6  3.10  2.00  1.00  1.99
```

2.3 Example 2

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

where

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

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

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

epsilon needs to be set smaller in P2-penalty case

[10]:
```
@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 P1 element

Without penalty (Problem1)

[11]:
```
refine_time = 6
epsilon_range = 5
element_type = 'P1'
print('element_type:', element_type)
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, element_type)
        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)
```

```python
        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]))
```

```
element_type: P1
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  0.00  0.00  0.00  0.00
2^-3  -0.00  -0.00  -0.00  -0.00
2^-4  -0.00  -0.00  -0.00  -0.00
2^-5  -0.00  -0.00  -0.00  -0.00
2^-6  -0.00  -0.00  -0.00  -0.00
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.64  0.75  -0.23  0.64
2^-3  1.46  0.61  -0.44  0.52
2^-4  1.04  0.52  -0.42  0.38
2^-5  -0.13  0.39  -0.30  0.19
2^-6  -0.42  0.14  -0.13  0.03
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.66  0.75  -0.23  0.65
2^-3  1.53  0.61  -0.47  0.56
2^-4  1.50  0.54  -0.49  0.51
2^-5  1.50  0.52  -0.49  0.50
2^-6  1.50  0.51  -0.50  0.50
```

```
epsilon = 1e-06
  h     L2u   H1u    H2u    epu
2^-2  1.66  0.75  -0.23  0.65
2^-3  1.53  0.61  -0.47  0.56
2^-4  1.50  0.54  -0.49  0.51
2^-5  1.50  0.52  -0.49  0.50
2^-6  1.50  0.51  -0.50  0.50
epsilon = 1e-08
  h     L2u   H1u    H2u    epu
2^-2  1.66  0.75  -0.23  0.65
2^-3  1.53  0.61  -0.47  0.56
2^-4  1.50  0.54  -0.49  0.51
2^-5  1.50  0.52  -0.49  0.50
2^-6  1.50  0.51  -0.50  0.50
```

With penalty (Problem2)

[12]:
```
sigma = 5
element_type = 'P1'
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, element_type)
        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)
```

```python
    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]))
```

```
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  1.66  1.21  0.75  0.72
2^-3  1.46  1.61  0.93  0.90
2^-4  1.67  1.78  0.95  0.93
2^-5  1.84  1.89  0.97  0.96
2^-6  1.93  1.95  0.99  0.98
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  2.37  1.52  -0.59  1.36
2^-3  1.41  1.19  -0.70  0.61
2^-4  0.04  0.28  -0.13  0.01
2^-5  0.57  0.74  0.39  0.48
2^-6  1.17  1.13  0.71  0.77
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  2.16  1.40  0.49  1.31
2^-3  2.38  1.59  0.55  1.53
2^-4  2.39  1.59  0.55  1.56
2^-5  2.64  1.57  0.53  1.55
2^-6  0.73  1.50  0.52  1.51
epsilon = 1e-06
  h    L2u   H1u   H2u   epu
2^-2  2.16  1.40  0.49  1.31
2^-3  2.36  1.59  0.55  1.53
2^-4  2.28  1.59  0.55  1.56
```

```
2^-5  2.18  1.56  0.53  1.55
2^-6  2.12  1.54  0.52  1.53
epsilon = 1e-08
  h    L2u   H1u   H2u   epu
2^-2  2.16  1.40  0.49  1.31
2^-3  2.36  1.59  0.55  1.53
2^-4  2.28  1.59  0.55  1.56
2^-5  2.17  1.56  0.53  1.55
2^-6  2.09  1.54  0.52  1.53
```

2.3.2 P2 element

Without penalty (Problem1)

[13]:
```
refine_time = 6
epsilon_range = 5
element_type = 'P2'
print('element_type:', element_type)
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, element_type)
        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]))
```

```
element_type: P2
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  -0.01  -0.00  -0.00  -0.00
2^-3  -0.01  -0.00  -0.00  -0.00
2^-4  -0.00  -0.00  -0.00  -0.00
2^-5  -0.00  -0.00  -0.00  -0.00
2^-6  -0.00  -0.00  -0.00  -0.00
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.53  0.61  -0.29  0.50
2^-3  1.51  0.55  -0.40  0.46
2^-4  1.13  0.51  -0.40  0.37
2^-5  -0.23  0.38  -0.29  0.19
2^-6  -0.47  0.13  -0.12  0.03
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.52  0.61  -0.30  0.51
2^-3  1.53  0.56  -0.42  0.50
2^-4  1.51  0.53  -0.47  0.50
2^-5  1.50  0.51  -0.49  0.50
2^-6  1.50  0.51  -0.49  0.50
epsilon = 1e-06
  h    L2u   H1u   H2u   epu
2^-2  1.52  0.61  -0.30  0.51
2^-3  1.53  0.56  -0.42  0.50
2^-4  1.51  0.53  -0.47  0.50
2^-5  1.50  0.51  -0.49  0.50
```

```
2^-6  1.50  0.51  -0.49  0.50
epsilon = 1e-08
  h     L2u   H1u    H2u    epu
2^-2  1.52  0.61  -0.30  0.51
2^-3  1.53  0.56  -0.42  0.50
2^-4  1.51  0.53  -0.47  0.50
2^-5  1.50  0.51  -0.49  0.50
2^-6  1.50  0.51  -0.49  0.50
```

With penalty (Problem2)

[14]:
```python
sigma = 5
element_type = 'P2'
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, element_type)
        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
```

```python
    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]))
```

```
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  1.44  1.35  0.87  0.83
2^-3  1.59  1.57  0.90  0.86
2^-4  1.78  1.73  0.94  0.91
2^-5  1.90  1.87  0.97  0.95
2^-6  1.95  1.94  0.99  0.98
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.39  1.26  -1.04  0.86
2^-3  0.37  -0.15  -0.73  -0.48
2^-4  0.32  0.21  -0.12  -0.01
2^-5  0.67  0.80  0.40  0.50
2^-6  1.20  1.15  0.72  0.78
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  3.10  1.91  0.87  1.81
2^-3  3.05  1.95  0.96  1.91
2^-4  1.43  1.96  0.99  1.97
2^-5  0.14  1.85  1.00  1.96
2^-6  0.02  1.29  0.78  1.54
epsilon = 1e-06
  h    L2u   H1u   H2u   epu
2^-2  3.14  1.91  0.87  1.81
2^-3  3.35  1.96  0.96  1.91
2^-4  3.21  1.99  0.99  1.97
2^-5  2.96  2.00  1.00  1.99
2^-6  1.58  2.00  1.00  2.00
epsilon = 1e-08
  h    L2u   H1u   H2u   epu
2^-2  3.14  1.91  0.87  1.81
2^-3  3.35  1.96  0.96  1.91
```

```
2^-4  3.23  1.99  0.99  1.97
2^-5  3.09  2.00  1.00  1.99
2^-6  3.02  2.00  1.00  2.00
```

2.4 Example3

$$u^0(x_1, x_2) = \sin(\pi x_1)\sin(\pi x_2)$$
$$f(x_1, x_2) = -\Delta u^0 = 2\pi^2 \sin(\pi x_1)\sin(\pi x_2)$$

epsilon needs to be set smaller in P2-penalty case

[15]:
```
def exact_u(x, y):
    return sin(pi * x) * sin(pi * y)


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


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

[16]:
```
@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
    dx = basis.dx
    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))


@LinearForm
def f_load(v, w):
    pix = pi * w.x[0]
    piy = pi * w.x[1]
    return (2 * pi**2 * sin(pix) * sin(piy)) * v


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

2.4.1 P1 element

Without penalty (Problem1)

[17]:
```
refine_time = 6
epsilon_range = 5
element_type = 'P1'
print('element_type:', element_type)
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, element_type)
        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])))
#         print(
#             '2^-' + str(i + 2), ' {:.5f}  {:.5f}  {:.5f}  {:.5f}'.format(
#                 L2s[i + 1], H1s[i + 1],
#                 H2s[i + 1],
#                 epus[i + 1]))
```

```
element_type: P1
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  0.00  0.00  0.01  0.01
2^-3  -0.00  -0.00  0.00  0.00
2^-4  -0.00  -0.00  -0.00  0.00
2^-5  -0.00  -0.00  -0.00  -0.00
2^-6  -0.00  -0.00  -0.00  -0.00
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.72  0.81  -0.19  0.70
2^-3  1.49  0.65  -0.44  0.56
2^-4  1.02  0.53  -0.42  0.39
2^-5  -0.14  0.39  -0.30  0.19
2^-6  -0.41  0.14  -0.13  0.03
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.74  0.81  -0.20  0.71
2^-3  1.57  0.65  -0.46  0.60
```

```
2^-4  1.51  0.55  -0.49  0.52
2^-5  1.51  0.52  -0.49  0.50
2^-6  1.50  0.51  -0.50  0.50
epsilon = 1e-06
  h    L2u   H1u   H2u    epu
2^-2  1.74  0.81  -0.20  0.71
2^-3  1.57  0.65  -0.46  0.60
2^-4  1.51  0.55  -0.49  0.52
2^-5  1.51  0.52  -0.49  0.50
2^-6  1.50  0.51  -0.50  0.50
epsilon = 1e-08
  h    L2u   H1u   H2u    epu
2^-2  1.74  0.81  -0.20  0.71
2^-3  1.57  0.65  -0.46  0.60
2^-4  1.51  0.55  -0.49  0.52
2^-5  1.51  0.52  -0.49  0.50
2^-6  1.50  0.51  -0.50  0.50
```

With penalty (Problem2)

[18]:

```
sigma = 5
element_type = 'P1'
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, element_type)
        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]))
```

```
epsilon = 1
  h     L2u    H1u    H2u    epu
2^-2  0.00   0.00   0.00   0.00
2^-3  -0.00  0.00   0.00   0.00
2^-4  -0.00  -0.00  0.00   0.00
2^-5  -0.00  -0.00  -0.00  -0.00
2^-6  -0.00  -0.00  -0.00  -0.00
epsilon = 0.01
  h     L2u    H1u    H2u    epu
2^-2  2.15   1.44   0.54   1.34
2^-3  1.90   1.48   0.58   1.42
2^-4  0.34   0.18   -0.65  0.10
2^-5  -0.48  -0.54  -1.30  -0.71
2^-6  -0.42  -0.25  -0.63  -0.39
epsilon = 0.0001
  h     L2u    H1u    H2u    epu
2^-2  2.19   1.46   0.53   1.37
2^-3  2.39   1.67   0.63   1.62
2^-4  2.26   1.67   0.62   1.65
2^-5  2.13   1.63   0.58   1.61
2^-6  2.03   1.58   0.55   1.57
epsilon = 1e-06
  h     L2u    H1u    H2u    epu
```

```
2^-2  2.19  1.46  0.53  1.37
2^-3  2.39  1.67  0.63  1.62
2^-4  2.26  1.67  0.62  1.65
2^-5  2.13  1.63  0.58  1.61
2^-6  2.06  1.58  0.55  1.57
epsilon = 1e-08
  h     L2u    H1u    H2u    epu
2^-2  2.19  1.46  0.53  1.37
2^-3  2.39  1.67  0.63  1.62
2^-4  2.26  1.67  0.62  1.65
2^-5  2.13  1.63  0.58  1.61
2^-6  2.06  1.58  0.55  1.57
```

2.4.2 P2 element

Without penalty (Problem1)

[19]:
```python
refine_time = 6
epsilon_range = 5
element_type = 'P2'
print('element_type:', element_type)
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, element_type)
        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
    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]))
```

```
element_type: P2
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  -0.01  -0.00  0.00  0.00
2^-3  -0.01  -0.00  -0.00  -0.00
2^-4  -0.00  -0.00  -0.00  -0.00
2^-5  -0.00  -0.00  -0.00  -0.00
2^-6  -0.00  -0.00  -0.00  -0.00
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.59  0.67  -0.28  0.56
2^-3  1.54  0.58  -0.39  0.48
2^-4  1.12  0.51  -0.39  0.38
2^-5  -0.25  0.38  -0.29  0.19
2^-6  -0.47  0.13  -0.12  0.03
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.59  0.67  -0.28  0.57
2^-3  1.56  0.58  -0.41  0.52
2^-4  1.52  0.53  -0.47  0.51
2^-5  1.50  0.52  -0.49  0.50
2^-6  1.50  0.51  -0.49  0.50
epsilon = 1e-06
  h    L2u   H1u   H2u   epu
2^-2  1.59  0.67  -0.28  0.57
```

```
2^-3  1.56  0.58  -0.41  0.52
2^-4  1.52  0.53  -0.47  0.51
2^-5  1.50  0.52  -0.49  0.50
2^-6  1.50  0.51  -0.49  0.50
epsilon = 1e-08
  h    L2u   H1u   H2u    epu
2^-2  1.59  0.67  -0.28  0.57
2^-3  1.56  0.58  -0.41  0.52
2^-4  1.52  0.53  -0.47  0.51
2^-5  1.50  0.52  -0.49  0.50
2^-6  1.50  0.51  -0.49  0.50
```

With penalty (Problem2)

[20]:
```
sigma = 5
element_type = 'P2'
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, element_type)
        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)
```

```python
    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]))
```

```
epsilon = 1
  h     L2u    H1u    H2u    epu
2^-2  -0.01  -0.01  -0.00  -0.00
2^-3  -0.00  -0.00  -0.00  -0.00
2^-4  -0.00  -0.00  -0.00  -0.00
2^-5  -0.00  -0.00  -0.00  -0.00
2^-6  -0.00  -0.00  -0.00  -0.00
epsilon = 0.01
  h     L2u    H1u    H2u    epu
2^-2  3.33  1.94  0.91  1.78
2^-3  0.44  0.66  0.59  0.66
2^-4  -0.90  -1.04  -1.67  -1.11
2^-5  -0.77  -0.65  -1.38  -0.81
2^-6  -0.48  -0.26  -0.63  -0.39
epsilon = 0.0001
  h     L2u    H1u    H2u    epu
2^-2  3.15  1.91  0.86  1.79
2^-3  3.40  1.95  0.95  1.90
2^-4  3.30  1.98  0.99  1.97
2^-5  3.13  2.00  1.00  1.99
2^-6  1.27  1.94  1.00  1.94
epsilon = 1e-06
  h     L2u    H1u    H2u    epu
2^-2  3.15  1.91  0.86  1.79
2^-3  3.40  1.95  0.95  1.90
2^-4  3.30  1.98  0.99  1.97
2^-5  3.12  2.00  1.00  1.99
2^-6  3.04  2.00  1.00  2.00
epsilon = 1e-08
```

h	L2u	H1u	H2u	epu
2^-2	3.15	1.91	0.86	1.79
2^-3	3.40	1.95	0.95	1.90
2^-4	3.30	1.98	0.99	1.97
2^-5	3.12	2.00	1.00	1.99
2^-6	3.04	2.00	1.00	2.00