Try_Perturb_Examples

October 18, 2020

1 Solving a Fourth Order Elliptic Singular Perturbation Problem

$$\left\{\begin{array}{ll}\varepsilon^2\Delta^2 u-\Delta u=f & \text{in }\Omega\\ u=\partial_n u=0 & \text{on }\partial\Omega\end{array}\right.$$

[18]:

```
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 dd, ddot, grad
from scipy.sparse.linalg import LinearOperator, minres
from skfem import *
from skfem.models.poisson import *
from skfem.assembly import BilinearForm, LinearForm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
plt.rcParams['figure.dpi'] = 100

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

1.1 Problem 1

The modified Morley-Wang-Xu element method is equivalent to finding $w_h \in W_h$ and $u_{h0} \in V_{h0}$ such that

$$\begin{array}{ll}(\nabla w_h,\nabla\chi_h)=(f,\chi_h) & \forall\chi_h\in W_h\\ \varepsilon^2 a_h\left(u_{h0},v_h\right)+b_h\left(u_{h0},v_h\right)=\left(\nabla w_h,\nabla_h v_h\right) & \forall v_h\in V_{h0}\end{array}$$

where

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

Using example

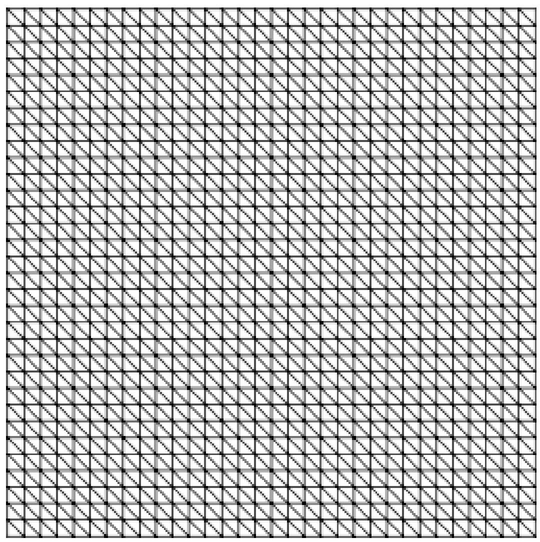
$$u\left(x_1,x_2\right)=\left(\sin\left(\pi x_1\right)\sin\left(\pi x_2\right)\right)^2$$

1.1.1 Setting ϵ and generating mesh

[55]:
```
epsilon = 0

# m = MeshTri.init_symmetric()
m = MeshTri()
m.refine(5)
element = {'w': ElementTriP1(), 'u': ElementTriMorley()}
basis = {variable: InteriorBasis(m, e, intorder=4)
    for variable, e in element.items()}  # intorder: integration order for␣
 ↪quadrature

draw(m)
plt.show()
```



1.1.2 Forms for $(\nabla w_h, \nabla \chi_h) = (f, \chi_h)$

[5]:
```
@BilinearForm
def laplace(u, v, w):
    '''
    for $(\nabla w_{h}, \nabla \chi_{h})$
    '''
    return dot(grad(u), grad(v))


@LinearForm
def f_load(v, w):
```

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

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

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

[6]:
```
@BilinearForm
def a_load(u, v, w):
    '''
    for $a_{h}$
    '''
    return ddot(dd(u), dd(v))


@BilinearForm
def b_load(u, v, w):
    '''
    for $b_{h}$
    '''
    return dot(grad(u), grad(v))


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

1.1.4 Setting boundary conditions

[7]:
```
def easy_boundary(basis):
    '''
    Input basis
    ----------------
    Return D for boundary conditions
    '''
```

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

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

1.1.5 Solving w_h and u_{h0}

[9]:
```
%%time

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

wh = solve(*condense(K1, f1, D=m.boundary_nodes()),␣
 ↪solver=solver_iter_krylov(Precondition=True))
```

```
Wall time: 20.9 ms
```

[10]:
```
%%time

D = easy_boundary(basis['u'])
K2 = epsilon**2 * asm(a_load, basis['u']) + asm(b_load, basis['u'])
f2 = asm(wv_load, basis['w'], basis['u']) * wh
uh0 = solve(*condense(K2, f2, D=D),␣
 ↪solver=solver_iter_krylov(Precondition=True)) # cg
```

```
Wall time: 64.8 ms
```

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

[11]:
```
def exact_u(x, y):
    return (sin(pi * x) * sin(pi * y))**2

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

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

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

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

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

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

[12]:
```
for i in range(6):
    epsilon = 1*10**(-i)

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

    for i in range(1, 7):
        m.refine()

        element = {'w': ElementTriP1(), 'u': ElementTriMorley()}
        basis = {variable: InteriorBasis(m, e, intorder=4)
            for variable, e in element.items()}  # intorder: integration order␣
 ↪for quadrature

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

```
        wh = solve(*condense(K1, f1, D=m.boundary_nodes()),␣
↪solver=solver_iter_krylov(Precondition=True))

        D = easy_boundary(basis['u'])
        K2 = epsilon**2 * asm(a_load, basis['u']) + asm(b_load, basis['u'])
        f2 = asm(wv_load, basis['w'], basis['u']) * wh
        uh0 = solve(*condense(K2, f2, D=D),␣
↪solver=solver_iter_krylov(Precondition=True))

        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)
    #     print('Case 2^-' + str(i))
    #     print('L2 error of uh0:', L2u)
    #     print('H1 error of uh0:', H1u)
    #     print('H2 error of uh0:', H2u)
    #     print('Ep error of uh0:', epu)
        h_list.append(m.param())
        Du_list.append(Du)
        L2_list.append(L2u)
        D2u_list.append(D2u)
        epu_list.append(epu)

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

```
                                        -np.log2(epus[i + 1] /
↪epus[i])))
```

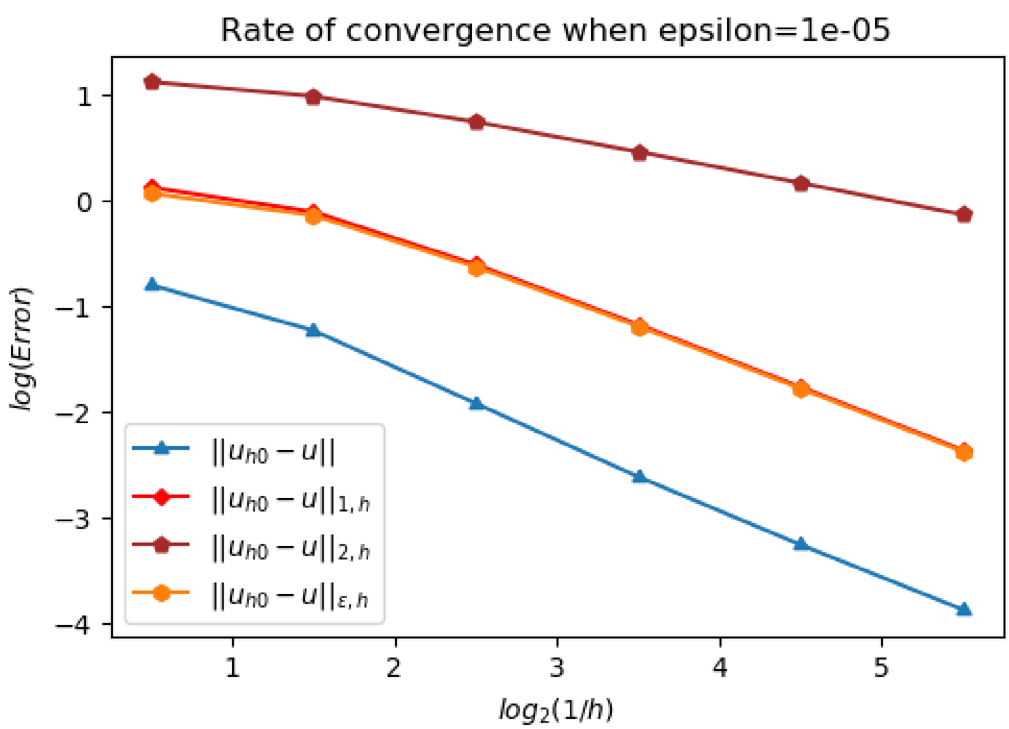
```
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  1.79  0.91  0.69  0.67
2^-3  2.19  1.76  1.02  0.98
2^-4  2.16  1.93  1.05  1.02
2^-5  2.06  1.98  1.02  1.01
2^-6  2.02  2.00  1.01  1.00
epsilon = 0.1
  h    L2u   H1u   H2u   epu
2^-2  1.38  0.84  0.60  0.66
2^-3  2.06  1.76  1.07  1.22
2^-4  2.06  1.93  1.08  1.15
2^-5  2.03  1.98  1.04  1.05
2^-6  2.01  2.00  1.01  1.01
epsilon = 0.01
  h    L2u   H1u   H2u   epu
2^-2  1.43  0.77  0.45  0.70
2^-3  2.26  1.67  0.86  1.61
2^-4  2.08  1.94  1.09  1.86
2^-5  1.76  2.03  1.22  1.85
2^-6  1.82  2.02  1.14  1.59
epsilon = 0.001
  h    L2u   H1u   H2u   epu
2^-2  1.43  0.77  0.45  0.70
2^-3  2.29  1.66  0.81  1.61
2^-4  2.31  1.89  0.94  1.87
2^-5  2.12  1.97  0.99  1.96
2^-6  1.98  2.00  1.03  1.99
epsilon = 0.0001
  h    L2u   H1u   H2u   epu
2^-2  1.43  0.77  0.45  0.70
2^-3  2.29  1.66  0.81  1.61
2^-4  2.31  1.89  0.94  1.87
2^-5  2.14  1.97  0.98  1.96
2^-6  2.04  1.99  1.00  1.99
epsilon = 1e-05
  h    L2u   H1u   H2u   epu
2^-2  1.43  0.77  0.45  0.70
2^-3  2.29  1.66  0.81  1.61
2^-4  2.31  1.89  0.94  1.87
2^-5  2.14  1.97  0.98  1.96
2^-6  2.04  1.99  1.00  1.99
```

[13]:

```
hs_Log = np.log2(hs)

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

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



1.2 Expamle 2 :

$$u^0\left(x_1, x_2\right) = \sin\left(\pi x_1\right)\sin\left(\pi x_2\right)$$

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

[14]:
```
@LinearForm
def f_load(v, w):
    '''
    for $(f, x_{h})$
    '''
    pix = pi * w.x[0]
    piy = pi * w.x[1]
    return (2 * pi**2 * sin(pix) * sin(piy)) * v

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


@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))
```

```python
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))
```

[16]:
```python
for i in range(6):
    epsilon = 1*10**(-i)

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

    for i in range(1, 7):
        m.refine()

        element = {'w': ElementTriP1(), 'u': ElementTriMorley()}
        basis = {variable: InteriorBasis(m, e, intorder=4)
            for variable, e in element.items()}  # intorder: integration order␣
↪for quadrature

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

        wh = solve(*condense(K1, f1, D=m.boundary_nodes()),␣
↪solver=solver_iter_krylov(Precondition=True))

        D = easy_boundary(basis['u'])
        K2 = epsilon**2 * asm(a_load, basis['u']) + asm(b_load, basis['u'])
        f2 = asm(wv_load, basis['w'], basis['u']) * wh
        uh0 = solve(*condense(K2, f2, D=D),␣
↪solver=solver_iter_krylov(Precondition=True))

        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)
    #     print('Case 2^-' + str(i))
    #     print('L2 error of uh0:', L2u)
    #     print('H1 error of uh0:', H1u)
    #     print('H2 error of uh0:', H2u)
    #     print('Ep error of uh0:', epu)
        h_list.append(m.param())
        Du_list.append(Du)
        L2_list.append(L2u)
        D2u_list.append(D2u)
        epu_list.append(epu)

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

```
epsilon = 1
  h    L2u   H1u   H2u   epu
2^-2  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 = 0.1
  h    L2u   H1u   H2u   epu
2^-2  0.54  0.44  0.08  0.30
2^-3  -0.05  0.12  -0.06  0.06
2^-4  -0.09  -0.01  -0.03  -0.01
```

```
2^-5  -0.03  -0.01  -0.01  -0.01
2^-6  -0.01  -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 = 0.001
  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  1.50  0.52  -0.49  0.50
2^-6  1.43  0.51  -0.48  0.48
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-05
  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
```

1.3 Example 3

$$u\left(x_1, x_2\right) = \epsilon\left(e^{-x_1/\epsilon} + e^{-x_2/\epsilon}\right) - x_1^2 x_2$$
$$f = 2x_2$$

[50]:
```
@LinearForm
def f_load(v, w):
    '''
    for $(f, x_{h})$
    '''
    x = w.x[0]
    y = w.x[1]
    return (2 * y) * v


def exact_u(x, y):
```

```python
    return ep * (exp(-x / ep) + exp(-y / ep)) - x**2 * y


def dexact_u(x, y):
    dux = -exp(-x / ep) - 2 * x * y
    duy = -x**2 - exp(-y / ep)
    return dux, duy


def ddexact(x, y):
    duxx = exp(-x / ep) / ep - 2 * y
    duxy = -2 * x
    duyx = duxy
    duyy = exp(-y / ep) / ep
    return duxx, duxy, duyx, duyy


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


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


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

[44]:
```python
for i in range(6):
    epsilon = 1 * 10**(-i)
    ep = epsilon
    L2_list = []
    Du_list = []
    D2u_list = []
```

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

for i in range(1, 7):
    m.refine()

    element = {'w': ElementTriP1(), 'u': ElementTriMorley()}
    basis = {
        variable: InteriorBasis(m, e, intorder=4)
        for variable, e in element.items()
    }  # intorder: integration order for quadrature

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

    wh = solve(*condense(K1, f1, D=m.boundary_nodes()),
               solver=solver_iter_krylov(Precondition=True))

    D = easy_boundary(basis['u'])
    K2 = epsilon**2 * asm(a_load, basis['u']) + asm(b_load, basis['u'])
    f2 = asm(wv_load, basis['w'], basis['u']) * wh
    uh0 = solve(*condense(K2, f2, D=D),
                solver=solver_iter_krylov(Precondition=True))

    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)
    #     print('Case 2^-' + str(i))
    #     print('L2 error of uh0:', L2u)
    #     print('H1 error of uh0:', H1u)
    #     print('H2 error of uh0:', H2u)
    #     print('Ep error of uh0:', epu)
    h_list.append(m.param())
    Du_list.append(Du)
    L2_list.append(L2u)
    D2u_list.append(D2u)
    epu_list.append(epu)

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

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

```
epsilon = 1
  h     L2u    H1u    H2u    epu
2^-2  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 = 0.1
  h     L2u    H1u    H2u    epu
2^-2  -0.03  -0.02  -0.00  -0.01
2^-3  0.00  -0.00  0.01  0.00
2^-4  0.01  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  -0.06  -0.01  -0.49  0.00
2^-3  -0.02  -0.01  -0.85  -0.01
2^-4  -0.00  -0.00  -0.36  -0.01
2^-5  0.00  -0.00  -0.05  -0.00
2^-6  0.00  0.00  0.02  0.00
epsilon = 0.001
  h     L2u    H1u    H2u    epu
2^-2  -0.06  -0.01  -0.11  0.01
2^-3  -0.02  -0.00  -0.20  0.00
2^-4  -0.01  0.00  -0.25  0.00
2^-5  -0.00  0.00  -0.67  0.00
2^-6  -0.00  0.00  -1.32  -0.00
epsilon = 0.0001
  h     L2u    H1u    H2u    epu
2^-2  -0.06  -0.01  -0.11  0.01
2^-3  -0.02  -0.00  -0.20  0.00
2^-4  -0.01  0.00  -0.27  0.00
2^-5  -0.00  0.00  -0.34  0.00
```

```
2^-6  -0.00  0.00  -0.39  0.00
epsilon = 1e-05
  h     L2u    H1u    H2u    epu
2^-2  -0.06  -0.01  -0.11  0.01
2^-3  -0.02  -0.00  -0.20  0.00
2^-4  -0.01  0.00  -0.27  0.00
2^-5  -0.00  0.00  -0.34  0.00
2^-6  -0.00  0.00  -0.39  0.00
```

1.4 Example 4

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

where

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

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

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

[52]:
```python
@LinearForm
def f_load(v, w):
    '''
    for $(f, x_{h})$
    '''
    x = w.x[0]
    y = w.x[1]
```

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


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


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

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


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


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


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

[54]:
```python
for i in range(6):
    epsilon = 1 * 10**(-i)
    ep = epsilon
    L2_list = []
    Du_list = []
    D2u_list = []
    h_list = []
    epu_list = []
    m = MeshTri()

    for i in range(1, 7):
        m.refine()

        element = {'w': ElementTriP1(), 'u': ElementTriMorley()}
        basis = {
            variable: InteriorBasis(m, e, intorder=4)
            for variable, e in element.items()
        }  # intorder: integration order for quadrature

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

        wh = solve(*condense(K1, f1, D=m.boundary_nodes()),
                   solver=solver_iter_krylov(Precondition=True))

        D = easy_boundary(basis['u'])
        K2 = epsilon**2 * asm(a_load, basis['u']) + asm(b_load, basis['u'])
        f2 = asm(wv_load, basis['w'], basis['u']) * wh
        uh0 = solve(*condense(K2, f2, D=D),
                    solver=solver_iter_krylov(Precondition=True))

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

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

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

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

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

```
2^-5  1.51  0.52  -0.49  0.50
2^-6  1.51  0.51  -0.50  0.50
epsilon = 1e-05
  h    L2u   H1u   H2u   epu
2^-2  1.66  0.75  -0.23  0.65
2^-3  1.53  0.61  -0.47  0.56
2^-4  1.50  0.54  -0.49  0.51
2^-5  1.50  0.52  -0.49  0.50
2^-6  1.51  0.51  -0.50  0.50
```