2.9 Fourth order equations

We consider the Kirchhoff plate equation: Find $w \in H^2$, such that

$$\int \nabla^2 w : \nabla^2 v = \int f v$$

A conforming method requires C^1 continuous finite elements. But there is no good option available, and thus there is no H^2 conforming finite element space in NGSolve.

We have the following two alternatives:

Hybridized C^0-continuous interior penalty method:

A simple way out is to use continuous elements, and treat the missing C^1-continuity by a Discontinuous Galerkin method. A DG formulation is

$$\sum_T \nabla^2 w : \nabla^2 v - \int_E \{\nabla^2 w\}_{nn} [\partial_n v] - \int_E \{\nabla^2 v\}_{nn} [\partial_n w] + \alpha \int_E [\partial_n w][\partial_n v]$$

[Baker 77, Brenner Gudi Sung, 2010]

We consider its hybrid DG version, where the normal derivative is a new, facet-based variable:

$$\sum_T \nabla^2 w : \nabla^2 v - \int_{\partial T} (\nabla^2 w)_{nn} (\partial_n v - \widehat{v_n}) - \int_{\partial T} (\nabla^2 v)_{nn} (\partial_n w - \widehat{w_n}) + \alpha \int_E (\partial_n v - \widehat{v_n})$$

The facet variable is the normal derivative $n_E \cdot \nabla w$, what is oriented along the arbitrarily chosen edge normal-vector. We cannot use the FacetSpace since it does not have the orientation, but we can use the normal traces of an HDiv space. We don't need inner basis functions, so we set order inner to 0:

In [ ]:
```
from ngsolve import *
from ngsolve.webgui import Draw
from netgen.geom2d import unit_square
mesh = Mesh (unit_square.GenerateMesh(maxh=0.1))
```

In [ ]:
```
order = 3

V1 = H1(mesh, order=order, dirichlet="left|bottom|right|top")
V2 = NormalFacetFESpace(mesh, order=order-1, dirichlet="left|bottom|right|t
V = V1*V2

w,what = V.TrialFunction()
v,vhat = V.TestFunction()
```

Some proxy-functions and gridfunctions provide additional differential operators. We can get them via the Operator function. w.Operator("hesse") provides the Hessian, a matrix-valued

function. Note that we can use the InnerProduct(.,.) for $\nabla^2 w : \nabla^2 v$:

In [ ]:
```
n = specialcf.normal(2)
h = specialcf.mesh_size

def jumpdn(v,vhat):
    return n*(grad(v)-vhat)
def hesse(v):
    return v.Operator("hesse")
def hessenn(v):
    return InnerProduct(n, hesse(v)*n)

dS = dx(element_boundary=True)
a = BilinearForm(V)
a += InnerProduct (hesse(w), hesse(v)) * dx \
     - hessenn(w) * jumpdn(v,vhat) * dS \
     - hessenn(v) * jumpdn(w,what) * dS \
     + 3*order*order/h * jumpdn(w,what) * jumpdn(v,vhat) * dS
a.Assemble()

f = LinearForm(V)
f += 1*v*dx
f.Assemble()
```

In [ ]:
```
u = GridFunction(V)
u.vec.data = a.mat.Inverse(V.FreeDofs()) * f.vec

Draw (u.components[0], mesh, "disp_DG")
Draw (grad (u.components[0]), mesh, "grad")
Draw (hesse (u.components[0]), mesh, "hesse")
```

The Hellan-Herrmann-Johnson Mixed Method:

[Hellan 67, Herrmann 67, Johnson 73, Arnold+Brezzi 85, Comodi 89]

We introduce the bending moments as a new, matrix valued variable $\sigma = \nabla^2 w$. Then we can write the plate equation as a saddle point problem:

$$\begin{array}{rcccl} \int \sigma : \tau & - & \int \tau : \nabla^2 w & = & 0 \quad \forall \tau \\ -\int \sigma : \nabla^2 v & & & = & \int f v \quad \forall v \end{array}$$

Integration by parts in upper-right and lower-left terms:

$$\begin{array}{rcccl} \int \sigma : \tau & + & < \operatorname{div} \tau, \nabla w > & = & 0 \quad \forall \tau \\ < \operatorname{div} \sigma, \nabla v > & & & = & \int f v \quad \forall v \end{array}$$

The Hellan-Herrmann-Johnson method uses finite elements σ_h which have continuous normal-normal component. Then, the term $< \operatorname{div} \tau, \nabla w >$ must be understood in distributional form:

$$\sum_T \int_T \operatorname{div} \sigma \, \nabla v - \sum_E \int_E [\sigma_{nt}] (\nabla v)_t$$

Since σ_{nn} is continuous, the jump occurs only in the tangential component of σ_n. Luckily, it hits the tangential component of ∇v, which is single-valued for H^1 finite elements. Thus, we can rewrite the term element-by-element:

$$\sum_T \int_T \operatorname{div} \sigma \nabla v - \int_{\partial T} \sigma_{nt} \nabla_t v$$

The Hellan-Herrmann-Johnson method does not require access to the neighbour element as in DG methods. In [Pechstein-Schöberl 11] the function space $H(\operatorname{div}\operatorname{div})$ was introduced. The HHJ finite elements are (nearly) conforming for this space. A basis was also given in [PS11]

In [ ]:

```
order = 3

V = HDivDiv(mesh, order=order-1)
Q = H1(mesh, order=order, dirichlet="left|bottom|top|right")
X = V*Q

print ("ndof-V:", V.ndof, ", ndof-Q:", Q.ndof)

sigma, w = X.TrialFunction()
tau, v = X.TestFunction()

n = specialcf.normal(2)

def tang(u): return u-(u*n)*n

a = BilinearForm(X, symmetric=True)
a += (InnerProduct (sigma, tau) + div(sigma)*grad(v) \
      + div(tau)*grad(w) - 1e-10*w*v )*dx \
      + (-(sigma*n) * tang(grad(v)) - (tau*n)*tang(grad(w)))*dx(element_bou
a.Assemble()

f = LinearForm(X)
f += -1 * v * dx
f.Assemble()

gfu = GridFunction(X)
gfu.vec.data = a.mat.Inverse(X.FreeDofs()) * f.vec

Draw (gfu.components[0], mesh, name="sigma")
Draw (gfu.components[1], mesh, name="disp")
```

In [ ]: