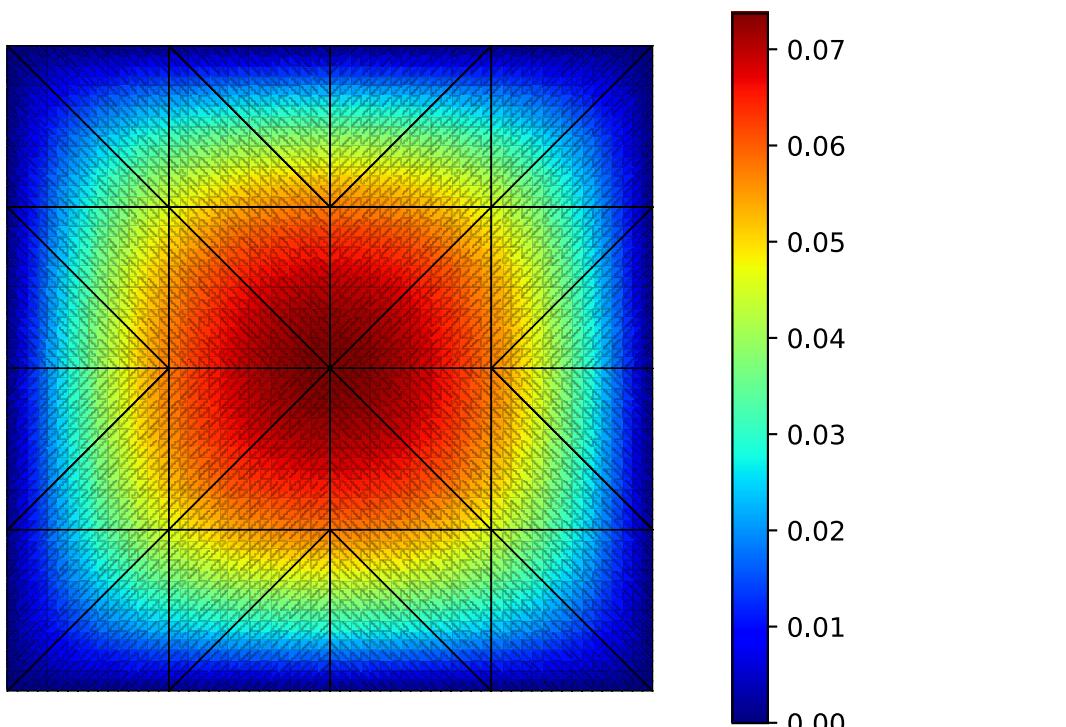


In [17]:

```

1 """Discontinuous Galerkin method."""
2
3 from skfem import *
4 from skfem.helpers import grad, dot, jump
5 from skfem.models.poisson import laplace, unit_load
6
7 m = MeshTri.init_sqsymmetric().refined()
8 e = ElementTriDG(ElementTriP4())
9 alpha = 1e-3
10
11 ib = Basis(m, e)
12 bb = FacetBasis(m, e)
13 fb = [InteriorFacetBasis(m, e, side=i) for i in [0, 1]]
14
15 @BilinearForm
16 def dgform(u, v, p):
17     ju, jv = jump(p, u, v)
18     h = p.h
19     n = p.n
20     return ju * jv / (alpha * h) - dot(grad(u), n) * jv - dot(grad(v), n) * ju
21
22 @BilinearForm
23 def nitscheform(u, v, p):
24     h = p.h
25     n = p.n
26     return u * v / (alpha * h) - dot(grad(u), n) * v - dot(grad(v), n) * u
27
28 A = asm(laplace, ib)
29 B = asm(dgform, fb, fb)
30 C = asm(nitscheform, bb)
31 b = asm(unit_load, ib)
32
33 x = solve(A + B + C, b)
34
35 M, X = ib.refinterp(x, 4)
36
37 def visualize():
38     from skfem.visuals.matplotlib import plot, draw
39     %config InlineBackend.figure_formats = ['svg']
40
41     ax = draw(M, boundaries_only=True)
42     return plot(M, X, shading="gouraud", ax=ax, colorbar=True)
43
44 if __name__ == "__main__":
45     visualize().show()
46

```



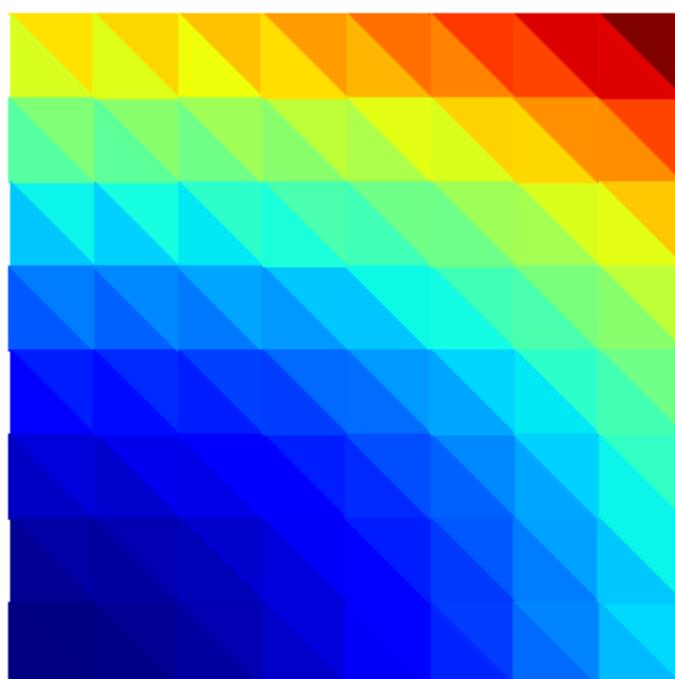
The Github repository of [gdmcbain/fenics-tuto-in-skfem](https://github.com/gdmcbain/fenics-tuto-in-skfem) contains eight exercises from the FEnics tutorial book. FEnics's great popularity no doubt stems largely from the book by Langtangen and Logg, entitled *Solving PDEs in Python: The FEnics Tutorial I*, a full-length book about using FEnics for finite element solutions of PDEs. This book can be accessed online or downloaded for free. Since FEnics is such a popular package, it is natural for someone to wonder if the new, little-known and somewhat brash program scikit-fem could reproduce the results of the FEnics tutorials. Below are six of the eight problems of the collection. (Note: problems 2 and 6 could not be attempted because they require a commercial program for execution.)

FEnics exercise #1 below, executed by scikit-fem.

```
In [1]: 1 import numpy as np
2
3 from skfem import *
4 from skfem.models.poisson import laplace, unit_load, mass
5
6 mesh = MeshTri().refined(3)
7 mesh
8
9 V = InteriorBasis(mesh, ElementTriP1())
10
11 u_D = 1 + [1, 2] @ mesh.p ** 2
12
13 boundary = mesh.boundary_nodes()
14
15 u = np.zeros_like(u_D)
16 u[boundary] = u_D[boundary]
17
18 a = asm(laplace, V)
19 L = -6.0 * asm(unit_load, V)
20
21 u = solve(*condense(a, L, u, D=boundary))
22
23 ax = mesh.plot(u)
24 ax.get_figure().savefig("poisson.svg")
25
26 mesh.save("fenics01.ply")
27
28 error = u - u_D
29 print("error_L2 =", np.sqrt(error.T @ asm(mass, V) @ error))
30 print("error_max =", np.linalg.norm(error, np.inf))
```

Warning: PLY doesn't support 64-bit integers. Casting down to 32-bit.

```
error_L2 = 3.090730095650652e-16
error_max = 1.1102230246251565e-15
```



Type *Markdown* and *LaTeX*: α^2

FEnics exercise #3 below, executed by scikit-fem.

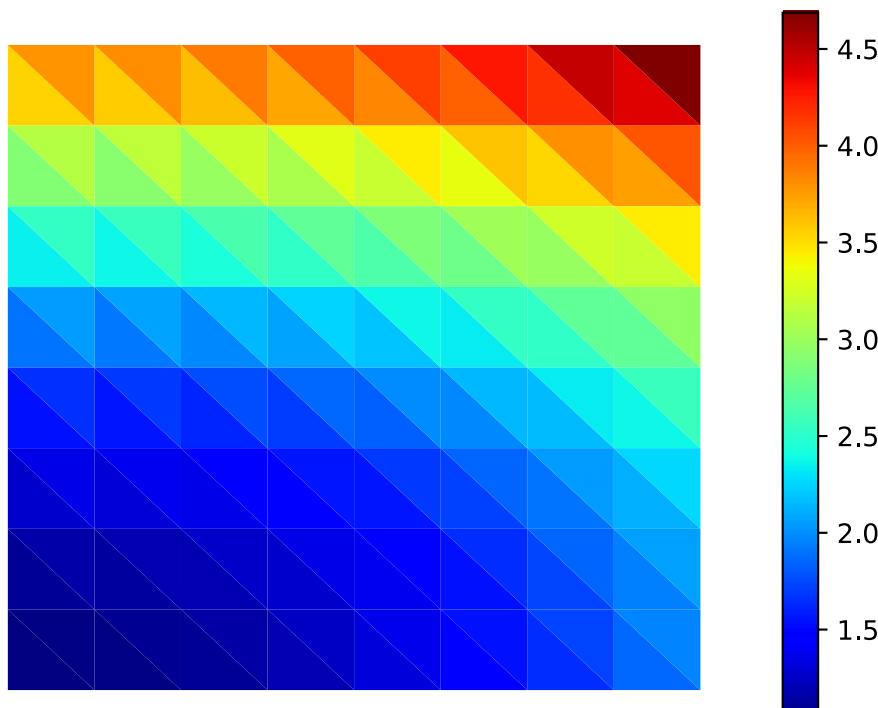
```
In [2]: 1 from matplotlib.pyplot import subplots, pause
2 # %matplotlib qt
3 import numpy as np
4 %config InlineBackend.figure_formats = ['svg']
5
6 from skfem import *
7 from skfem.models.poisson import laplace, mass, unit_load
8
9 alpha = 3.0
10 beta = 1.2
11
12 nx = ny = 2 ** 3
13
14 time_end = 2.0
```

```

15 num_steps = 10
16 dt = time_end / num_steps
17
18 mesh = (
19     MeshLine(np.linspace(0, 1, nx + 1)) * MeshLine(np.linspace(0, 1, ny + 1))
20 ).to_meshtri()
21 basis = InteriorBasis(mesh, ElementTriP1())
22
23 boundary = basis.get_dofs().all()
24 interior = basis.complement_dofs(boundary)
25
26 M = asm(mass, basis)
27 A = M + dt * asm(laplace, basis)
28 f = (beta - 2 - 2 * alpha) * asm(unit_load, basis)
29
30 fig, ax = subplots()
31
32
33 def dirichlet(t: float) -> np.ndarray:
34     return 1.0 + [1.0, alpha] @ mesh.p ** 2 + beta * t
35
36
37 t = 0.0
38 u = dirichlet(t)
39
40 zlim = (0, np.ceil(1 + alpha + beta * time_end))
41
42 for i in range(num_steps + 1):
43
44     ax.cla()
45     ax.axis("off")
46     fig.suptitle("t = {:.4f}".format(t))
47     mesh.plot(u, ax=ax, zlim=zlim)
48     if t == 0.0:
49         fig.colorbar(ax.get_children()[0])
50     fig.show()
51     pause(1.0)
52
53     t += dt
54     b = dt * f + M @ u
55
56     u_D = dirichlet(t)
57     u = solve(*condense(A, b, u_D, D=boundary))
58     error = np.linalg.norm(u - u_D)
59     print("t = %.2f: error = %.3g" % (t, error))
60
C:\Users\gary\AppData\Local\Temp\ipykernel_7460\3832573378.py:50: UserWarning: Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
fig.show()

```

t = 0.0000



```

t = 0.20: error = 9.44e-15
t = 0.40: error = 1.25e-14
t = 0.60: error = 1.45e-14
t = 0.80: error = 1.7e-14
t = 1.00: error = 1.68e-14
t = 1.20: error = 1.89e-14
t = 1.40: error = 2.01e-14
t = 1.60: error = 2.01e-14
t = 1.80: error = 2.16e-14
t = 2.00: error = 2.34e-14
t = 2.20: error = 2.46e-14

```

FEnics exercise #4 below, executed by scikit-fem.

In [3]:

```

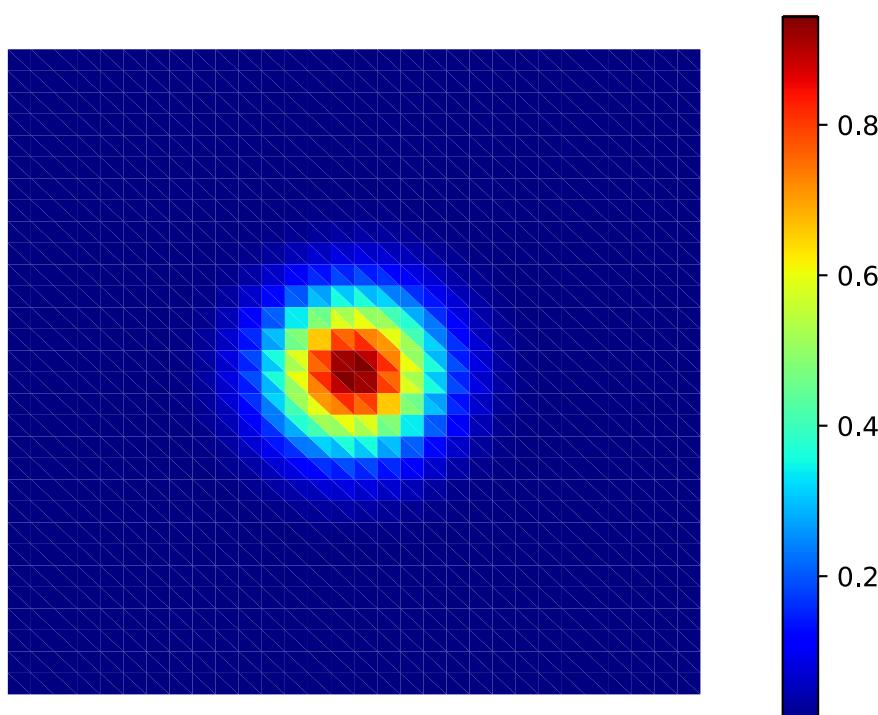
1 from pathlib import Path
2
3 from matplotlib.pyplot import subplots, pause
4 import numpy as np
5
6 from skfem import *
7 from skfem.models.poisson import laplace, mass
8
9 a = 5.0
10
11 nx = ny = 30
12
13 time_end = 2.0
14 num_steps = 50
15 dt = time_end / num_steps
16
17 mesh = (
18     MeshLine(np.linspace(-2, 2, nx + 1)) * MeshLine(np.linspace(-2, 2, ny + 1))
19 ).to_meshtri()
20 basis = InteriorBasis(mesh, ElementTriP1())
21
22 boundary = basis.get_dofs().all()
23 interior = basis.complement_dofs(boundary)
24
25 M = asm(mass, basis)
26 A = M + dt * asm(laplace, basis)
27
28 fig, ax = subplots()
29
30 t = 0.0
31 u = np.exp(-a * (np.sum(mesh.p ** 2, axis=0))) # initial condition, P1 only
32
33 output_dir = Path("heat_gaussian")
34 try:
35     output_dir.mkdir()
36 except FileExistsError:
37     pass
38
39 for i in range(num_steps + 1):
40
41     ax.cla()
42     ax.axis("off")
43     fig.suptitle("t = {:.4f}".format(t))
44     mesh.plot(u, ax=ax, zlim=(0, 1))
45     if t == 0.0:
46         fig.colorbar(ax.get_children()[0])
47         fig.savefig("initial.png")
48     fig.show()
49     pause(0.01)
50
51     t += dt
52     b = M @ u
53
54     u = solve(*condense(A, b, D=boundary))
55
56     mesh.save(str(output_dir.joinpath(f"solution{i:06d}.msh")), {"temperature": u})
57

```

C:\Users\gary\AppData\Local\Temp\ipykernel_7460/3973729471.py:48: UserWarning: Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.

fig.show()

t = 0.0000



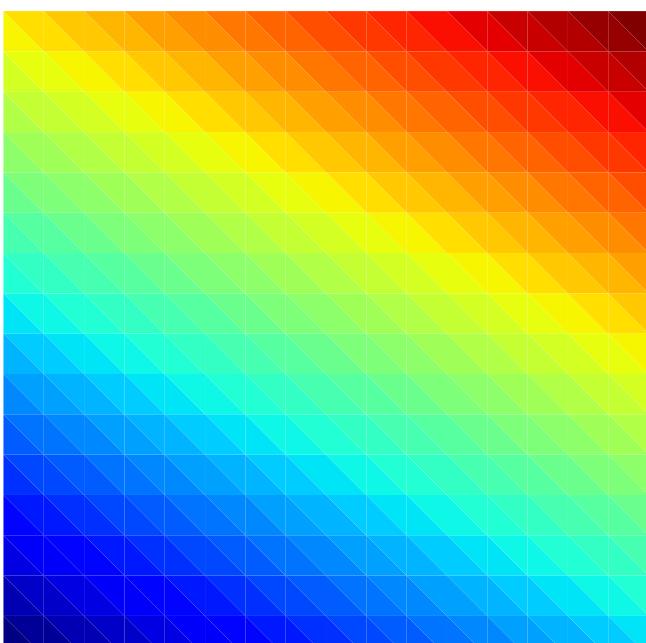
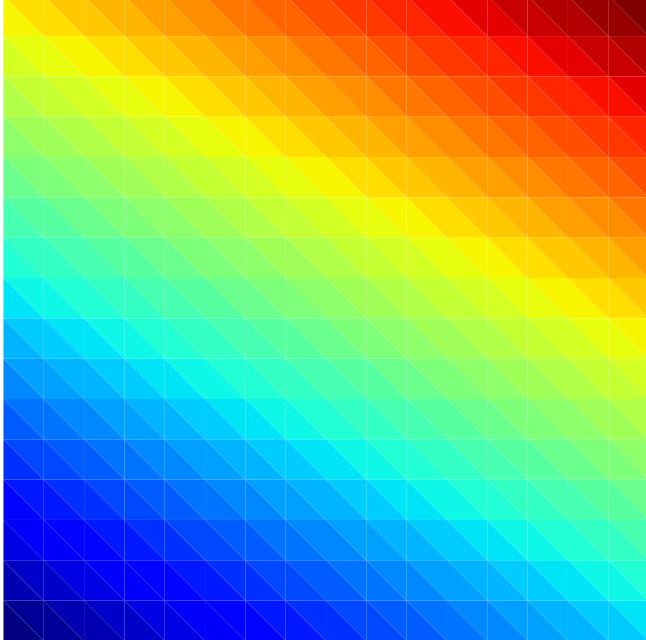
FEnics exercise #5 below, executed by scikit-fem.

```
In [18]: 1 from pathlib import Path
2
3 import numpy as np
4 from scipy.optimize import root
5
6 from sympy import symbols
7 from sympy.vector import CoordSys3D, gradient, divergence
8 from sympy.utilities.lambdify import lambdify
9
10 from skfem import (
11     MeshTri,
12     InteriorBasis,
13     ElementTriP1,
14     BilinearForm,
15     LinearForm,
16     asm,
17     solve,
18     condense,
19 )
20 from skfem.models.poisson import laplace
21 from skfem.visuals.matplotlib import plot
22 %config InlineBackend.figure_formats = ['svg']
23
24
25 output_dir = Path("poisson_nonlinear")
26
27 try:
28     output_dir.mkdir()
29 except FileExistsError:
30     pass
31
32
33 def q(u):
34     """Return nonlinear coefficient"""
35     return 1 + u * u
36
37
38 R = CoordSys3D("R")
39
40
41 def apply(f, coords):
42     x, y = symbols("x y")
43     return lambdify((x, y), f.subs({R.x: x, R.y: y}))(*coords)
44
45
46 u_exact = 1 + R.x + 2 * R.y # exact solution
47 f = -divergence(q(u_exact)) * gradient(u_exact) # manufactured RHS
48
49 mesh = MeshTri().refined(3) # refine thrice
50
51 V = InteriorBasis(mesh, ElementTriP1())
52
53 boundary = V.get_dofs().all()
54 interior = V.complement_dofs(boundary)
55
56
57 @LinearForm
58 def load(v, w):
59     return v * apply(f, w.x)
60
61
62 b = asm(load, V)
63
64
65 @BilinearForm
66 def diffusion_form(u, v, w):
67     return sum(v.grad * (q(w["w"])) * u.grad)
68
69
70 def diffusion_matrix(u):
71     return asm(diffusion_form, V, w=V.interpolate(u))
72
73
74 dirichlet = apply(u_exact, mesh.p) # P1 nodal interpolation
75 plot(V, dirichlet).get_figure().savefig(str(output_dir.joinpath("exact.png")))
76
77
78 def residual(u):
79     r = b - diffusion_matrix(u) @ u
80     r[boundary] = 0.0
81     return r
82
83
```

```

84 u = np.zeros(V.N)
85 u[boundary] = dirichlet[boundary]
86 result = root(residual, u, method="krylov")
87
88 if result.success:
89     u = result.x
90     print("Success. Residual =", np.linalg.norm(residual(u), np.inf))
91     print("Nodal Linf error =", np.linalg.norm(u - dirichlet, np.inf))
92     plot(V, u).get_figure().savefig(str(output_dir.joinpath("solution.png")))
93 else:
94     print(result)
95
Success. Residual = 1.4058151617812875e-07
Nodal Linf error = 1.2228777324096995e-08

```



FEnics exercise #7 below, executed by scikit-fem.

In [3]:

```

1 from pathlib import Path
2
3 import numpy as np
4
5 import skfem
6 from skfem.models.poisson import vector_laplace, laplace
7 from skfem.models.general import divergence
8
9 from meshio.xdmf import TimeSeriesWriter
10
11
12 @skfem.BilinearForm
13 def vector_mass(u, v, w):
14     return sum(v * u)
15
16
17 @skfem.BilinearForm
18 def port_pressure(u, v, w):
19     return sum(v * (u * w.n))
20
21
22 p_inlet = 8.0
23

```

```

24 #mesh = skfem.MeshTri()
25 #mesh.refine(3)
26 mesh = MeshTri().refined(3)
27
28 boundary = {
29     "inlet": mesh.facets_satisfying(lambda x: x[0] == 0),
30     "outlet": mesh.facets_satisfying(lambda x: x[0] == 1),
31     "wall": mesh.facets_satisfying(lambda x: np.logical_or(x[1] == 0, x[1] == 1)),
32 }
33 boundary["ports"] = np.concatenate([boundary["inlet"], boundary["outlet"]])
34
35 element = {"u": skfem.ElementVectorH1(skfem.ElementTriP2()), "p": skfem.ElementTriP1()}
36 basis = {
37     **{v: skfem.InteriorBasis(mesh, e, intorder=4) for v, e in element.items()},
38     **{
39         "label": skfem.FacetBasis(mesh, element["u"], facets=boundary[label])
40         for label in ["inlet", "outlet"]
41     },
42 }
43
44
45 M = skfem.asm(vector_mass, basis["u"])
46 L = {"u": skfem.asm(vector_laplace, basis["u"]), "p": skfem.asm(laplace, basis["p"])}
47 B = -skfem.asm(divergence, basis["u"], basis["p"])
48 P = B.T + skfem.asm(
49     port_pressure,
50     *(
51         skfem.FacetBasis(mesh, element[v], facets=boundary["ports"], intorder=3)
52         for v in ["p", "u"]
53     )
54 )
55
56 t_final = 1.0
57 dt = 0.05
58
59 dirichlet = {
60     "u": basis["u"].get_dofs(boundary["wall"]).all(),
61     "p": np.concatenate([basis["p"].get_dofs(boundary["ports"]).all()]),
62 }
63 inlet_pressure_dofs = basis["p"].get_dofs(boundary["inlet"]).all()
64
65 uv_, p_ = (np.zeros(basis[v].N) for v in element.keys()) # penultimate
66 p__ = np.zeros_like(p_) # antepenultimate
67
68 K = M / dt + L["u"]
69
70 t = 0
71
72 with TimeSeriesWriter("channel.xdmf") as writer:
73     writer.write_points_cells(mesh.p.T, {"triangle": mesh.t.T})
74
75     while t < t_final:
76
77         t += dt
78
79         # Step 1: Momentum prediction (Ern & Guermond 2002, eq. 7.40, p. 274)
80
81         uv = skfem.solve(
82             *skfem.condense(
83                 K,
84                 (M / dt) @ uv_ - P @ (2 * p_ - p__),
85                 np.zeros_like(uv_),
86                 D=dirichlet["u"],
87             )
88         )
89
90         # Step 2: Projection (Ern & Guermond 2002, eq. 7.41, p. 274)
91
92         dp = np.zeros(basis["p"].N)
93         dp[inlet_pressure_dofs] = p_inlet - p_[inlet_pressure_dofs]
94
95         dp = skfem.solve(*skfem.condense(L["p"], B @ uv, dp, D=dirichlet["p"]))
96
97         # Step 3: Recover pressure and velocity (E. & G. 2002, p. 274)
98
99         p = p_ + dp
100        print(min(p), "<= p <= ", max(p))
101
102        du = skfem.solve(*skfem.condense(M / dt, -P @ dp, D=dirichlet["u"]))
103        u = uv + du
104
105        uv_ = uv
106        p_, p__ = p, p_
107
108        # postprocessing
109        writer.write_data(
110            t,
111            point_data={
112                "pressure": p,

```

```

114         "velocity": np.pad(
115             u[basis["u"].nodal_dofs].T, ((0, 0), (0, 1)), "constant"
116         ),
117     },
118 )
119
120     print(min(u[::2]), "<= u <= ", max(u[::2]), "||v|| = ", np.linalg.norm(u[1::2]))
121
122
123 # References
124
125 # Ern, A., Guermond, J.-L. (2002). _Éléments finis : théorie,
126 # applications, mise en œuvre_ (Vol. 36). Paris: Springer. ISBN:
127 # 3540426159
128 <= p <= 8.0
0.0 <= u <= 0.5322508243494332 ||v|| = 4.026720129295742e-14
0.0 <= p <= 8.0
0.0 <= u <= 0.6309113809656185 ||v|| = 1.436588597740157e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.7616343911111542 ||v|| = 1.90715233764314e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.8422995060746478 ||v|| = 2.4372603252675528e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.8947839812242627 ||v|| = 2.810672594345035e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9296216090981441 ||v|| = 3.052360984815118e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9528899755044427 ||v|| = 3.190213298235965e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9684589961632456 ||v|| = 3.2489532651000665e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9788815550706454 ||v|| = 3.2490550626528445e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9858598213872569 ||v|| = 3.206844857583792e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9905321915694375 ||v|| = 3.135004182450136e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9936606578490389 ||v|| = 3.0431839802365636e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9957553825354923 ||v|| = 2.9386031114155746e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9971579467570855 ||v|| = 2.826577517080062e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9980970614624239 ||v|| = 2.7109626371259528e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9987258643521538 ||v|| = 2.594510013397419e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9991468917776665 ||v|| = 2.479147375473926e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9994287989893178 ||v|| = 2.3661944160698367e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9996175555793892 ||v|| = 2.256526542369644e-05
0.0 <= p <= 8.0
0.0 <= u <= 0.9997439454833684 ||v|| = 2.150697652766544e-05

```

FEnics exercise #8 below, executed by scikit-fem. Note: In this case it is necessary to set the inline backend to "retina", because the .svg backend scrambles the graphic. (Whatever it is, "retina" seems to match vector image quality, so it is considered a player of equal skill.)

In [5]:

```

1 r"""
2 This example demonstrates the solution of a slightly more complicated problem
3 with multiple boundary conditions and a fourth-order differential operator. We
4 consider the `Kirchhoff plate bending problem
5 <https://en.wikipedia.org/wiki/Kirchhoff%20Love\_plate\_theory>`_ which
6 finds its applications in solid mechanics. For a stationary plate of constant
7 thickness :math:`d`, the governing equation reads: find the deflection :math:`u`
8 :math:`\rightarrow \Omega` that satisfies
9 .. math::
10   \frac{Ed^3}{12(1-\nu^2)} \Delta^2 u = f \quad \text{in } \Omega,
11 where :math:`\Omega = (0,1)^2` , :math:`f` is a perpendicular force,
12 :math:`E` and :math:`\nu` are material parameters.
13 In this example, we analyse a :math:`1, \text{m}^2` plate of steel with thickness :math:`d=0.1` ,
14 The Young's modulus of steel is :math:`E = 200 \cdot 10^9 \text{Pa}` and Poisson
15 ratio :math:`\nu = 0.3` .
16 In reality, the operator
17 .. math::
18   \frac{Ed^3}{12(1-\nu^2)} \Delta^2
19 is a combination of multiple first-order operators:
20 .. math::
21   \boldsymbol{K}(u) = - \boldsymbol{\varepsilon}(\nabla u), \quad \boldsymbol{\varepsilon}(\nabla u)
22 .. math::
23   \boldsymbol{M}(u) = \frac{d^3}{12} C \boldsymbol{K}(u), \quad C \boldsymbol{M}(u)
24 where :math:`\boldsymbol{I}` is the identity matrix. In particular,
25 .. math::
26   \frac{Ed^3}{12(1-\nu^2)} \Delta^2 u = - \text{div} \text{div} \boldsymbol{M}(u).

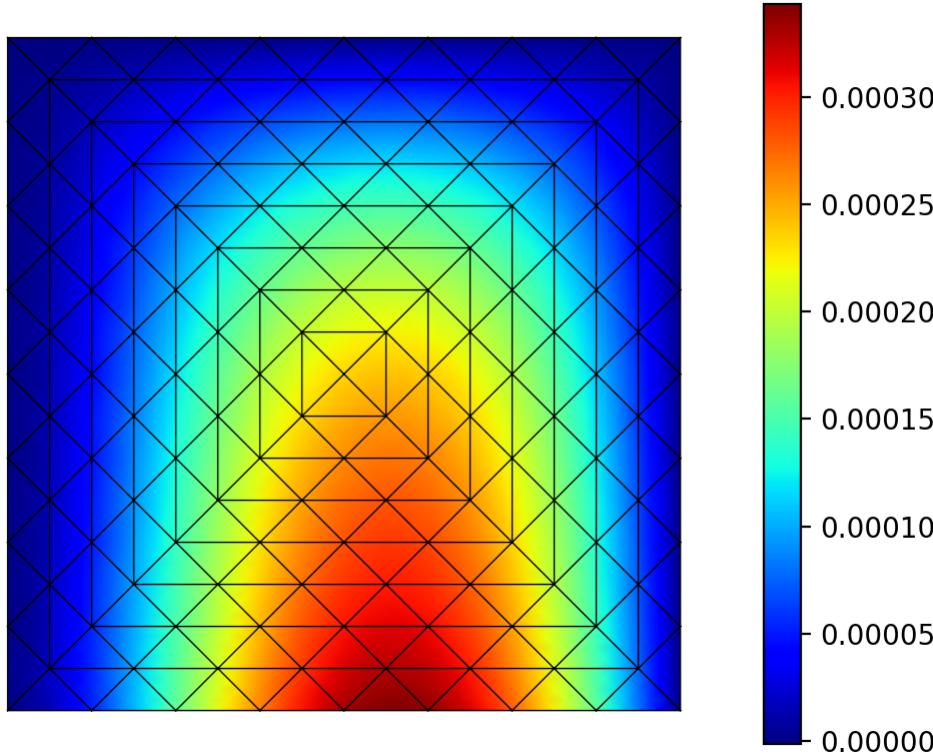
```

```

27 There are several boundary conditions that the problem can take.
28 The *fully clamped* boundary condition reads
29 .. math::
30     u = \frac{\partial u}{\partial \boldsymbol{n}} = 0,
31 where :math:`\boldsymbol{n}` is the outward normal.
32 Moreover, the *simply supported* boundary condition reads
33 .. math::
34     u = 0, \quad M_{nn}(u)=0,
35 where :math:`M_{nn} = \boldsymbol{n} \cdot (\boldsymbol{M} \boldsymbol{n})` .
36 Finally, the *free* boundary condition reads
37 .. math::
38     M_{nn}(u)=0, \quad V_n(u)=0,
39 where :math:`V_n` is the Kirchhoff shear force <https://arxiv.org/pdf/1707.08396.pdf>_. The exa
40 definition is not needed here as this boundary condition is a
41 natural one.
42 The correct weak formulation for the problem is: find :math:`u \in V` such that
43 .. math::
44     \int_\Omega \boldsymbol{M}(u) : \boldsymbol{K}(v) \, , \, dv = \int_\Omega fv \, ,
45 where :math:`V` is now a subspace of :math:`H^2` with the essential boundary
46 conditions for :math:`u` and :math:`\frac{\partial u}{\partial \boldsymbol{n}}` .
47 Instead of constructing a subspace for :math:`H^2` , we discretise the problem
48 using the `non-conforming Morley finite element
49 <https://users.aalto.fi/~jakke74/WebFiles/Slides-Niiranen-ADMOS-09.pdf>`_ which
50 is a piecewise quadratic :math:`C^0` -continuous element for biharmonic problems.
51 The full source code of the example reads as follows:
52 .. literalinclude:: examples/ex02.py
53     :start-after: EOF"""
54 from skfem import *
55 from skfem.models.poisson import unit_load
56 import numpy as np
57 %config InlineBackend.figure_formats = ['retina']
58
59 m = (
60     MeshTri.init_symmetric()
61     .refined(3)
62     .with_boundaries(
63         {
64             "left": lambda x: x[0] == 0,
65             "right": lambda x: x[0] == 1,
66             "top": lambda x: x[1] == 1,
67         }
68     )
69 )
70
71 e = ElementTriMorley()
72 ib = Basis(m, e)
73
74
75 @BilinearForm
76 def bilinf(u, v, w):
77     from skfem.helpers import dd, ddot, trace, eye
78     d = 0.1
79     E = 200e9
80     nu = 0.3
81
82     def C(T):
83         return E / (1 + nu) * (T + nu / (1 - nu) * eye(trace(T), 2))
84
85     return d**3 / 12.0 * ddot(C(dd(u)), dd(v))
86
87
88 K = asm(bilinf, ib)
89 f = 1e6 * asm(unit_load, ib)
90
91 D = np.hstack([ib.get_dofs("left"), ib.get_dofs({"right", "top"}).all("u")])
92
93 x = solve(*condense(K, f, D=D))
94
95 def visualize():
96     from skfem.visuals.matplotlib import draw, plot
97     ax = draw(m)
98     return plot(ib,
99                 x,
100                ax=ax,
101                shading='gouraud',
102                colorbar=True,
103                nrefs=2)
104
105 if __name__ == "__main__":
106     visualize().show()
107
108
109
110
111
112
113
114
115
116

```

```
117  
118  
119  
120
```



The following cell shows an example from the scikit-fem documentation, *Example 11: Three-dimensional linear elasticity*.

```
In [12]: 1 r"""Linear elasticity.  
2 This example solves the linear elasticity problem using trilinear elements. The  
3 weak form of the linear elasticity problem is defined in  
4 :func:`skfem.models.elasticity.linear_elasticity`.  
5 """  
6  
7 import numpy as np  
8 from skfem import *  
9 from skfem.models.elasticity import linear_elasticity, lame_parameters  
10  
11 m = MeshHex().refined(3)  
12 e1 = ElementHex1()  
13 e = ElementVector(e1)  
14 ib = Basis(m, e, MappingIsoparametric(m, e1), 3)  
15  
16 K = asm(linear_elasticity(*lame_parameters(1e3, 0.3)), ib)  
17  
18 dofs = {  
19     'left' : ib.get_dofs(lambda x: x[0] == 0.0),  
20     'right': ib.get_dofs(lambda x: x[0] == 1.0),  
21 }  
22  
23 u = ib.zeros()  
24 u[dofs['right'].nodal['u^1']] = 0.3  
25  
26 I = ib.complement_dofs(dofs)  
27  
28 u = solve(*condense(K, x=u, I=I))  
29  
30 sf = 1.0  
31 m = m.translated(sf * u[ib.nodal_dofs])  
32  
33 if __name__ == "__main__":  
34     from os.path import splitext  
35     from sys import argv  
36  
37 #m.save(splitext(argv[0])[0] + '.vtk')  
38 m.save('testimg' + '.vtk')  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54
```

