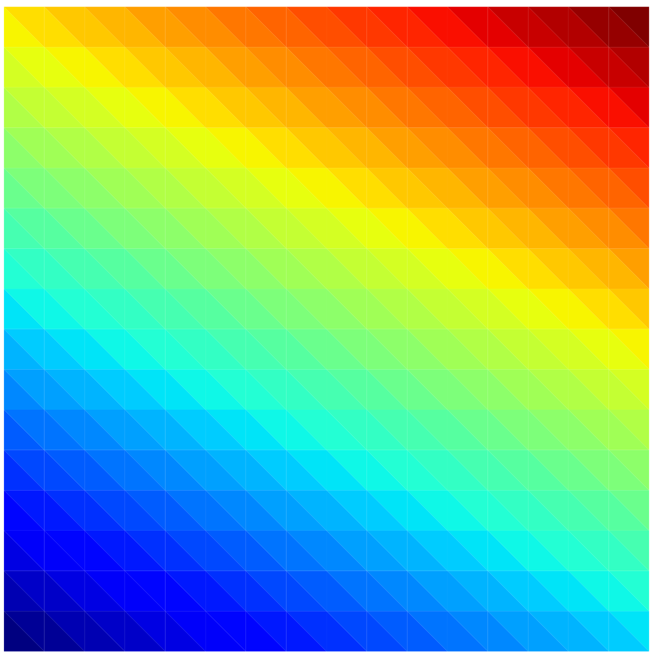
FEnics exercise #5 below, executed by scikit-fem.

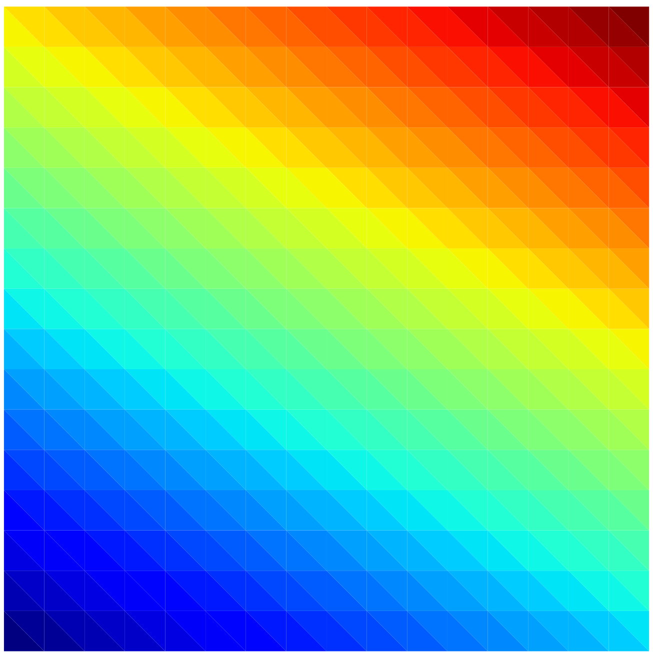
In [18]:

```python
from pathlib import Path

import numpy as np
from scipy.optimize import root

from sympy import symbols
from sympy.vector import CoordSys3D, gradient, divergence
from sympy.utilities.lambdify import lambdify

from skfem import (
    MeshTri,
    InteriorBasis,
    ElementTriP1,
    BilinearForm,
    LinearForm,
    asm,
    solve,
    condense,
)
from skfem.models.poisson import laplace
from skfem.visuals.matplotlib import plot
%config InlineBackend.figure_formats = ['svg']


output_dir = Path("poisson_nonlinear")

try:
    output_dir.mkdir()
except FileExistsError:
    pass


def q(u):
    """Return nonlinear coefficient"""
    return 1 + u * u


R = CoordSys3D("R")


def apply(f, coords):
    x, y = symbols("x y")
    return lambdify((x, y), f.subs({R.x: x, R.y: y}))(*coords)


u_exact = 1 + R.x + 2 * R.y  # exact solution
f = -divergence(q(u_exact) * gradient(u_exact))  # manufactured RHS

mesh = MeshTri().refined(3)  # refine thrice

V = InteriorBasis(mesh, ElementTriP1())

boundary = V.get_dofs().all()
interior = V.complement_dofs(boundary)


@LinearForm
def load(v, w):
    return v * apply(f, w.x)


b = asm(load, V)


@BilinearForm
def diffusion_form(u, v, w):
    return sum(v.grad * (q(w["w"]) * u.grad))


def diffusion_matrix(u):
    return asm(diffusion_form, V, w=V.interpolate(u))


dirichlet = apply(u_exact, mesh.p)  # P1 nodal interpolation
plot(V, dirichlet).get_figure().savefig(str(output_dir.joinpath("exact.png")))


def residual(u):
    r = b - diffusion_matrix(u) @ u
    r[boundary] = 0.0
    return r


```

```python
u = np.zeros(V.N)
u[boundary] = dirichlet[boundary]
result = root(residual, u, method="krylov")

if result.success:
    u = result.x
    print("Success.  Residual =", np.linalg.norm(residual(u), np.inf))
    print("Nodal Linf error =", np.linalg.norm(u - dirichlet, np.inf))
    plot(V, u).get_figure().savefig(str(output_dir.joinpath("solution.png")))
else:
    print(result)

```

```
Success.  Residual = 1.4058151617812875e-07
Nodal Linf error = 1.2228777324096995e-08
```





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

In [3]:

```python
from pathlib import Path

import numpy as np

import skfem
from skfem.models.poisson import vector_laplace, laplace
from skfem.models.general import divergence

from meshio.xdmf import TimeSeriesWriter


@skfem.BilinearForm
def vector_mass(u, v, w):
    return sum(v * u)


@skfem.BilinearForm
def port_pressure(u, v, w):
    return sum(v * (u * w.n))


p_inlet = 8.0

```

```python
#mesh = skfem.MeshTri()
#mesh.refine(3)
mesh = MeshTri().refined(3)

boundary = {
    "inlet": mesh.facets_satisfying(lambda x: x[0] == 0),
    "outlet": mesh.facets_satisfying(lambda x: x[0] == 1),
    "wall": mesh.facets_satisfying(lambda x: np.logical_or(x[1] == 0, x[1] == 1)),
}
boundary["ports"] = np.concatenate([boundary["inlet"], boundary["outlet"]])

element = {"u": skfem.ElementVectorH1(skfem.ElementTriP2()), "p": skfem.ElementTriP1()}
basis = {
    **{v: skfem.InteriorBasis(mesh, e, intorder=4) for v, e in element.items()},
    **{
        label: skfem.FacetBasis(mesh, element["u"], facets=boundary[label])
        for label in ["inlet", "outlet"]
    },
}


M = skfem.asm(vector_mass, basis["u"])
L = {"u": skfem.asm(vector_laplace, basis["u"]), "p": skfem.asm(laplace, basis["p"])}
B = -skfem.asm(divergence, basis["u"], basis["p"])
P = B.T + skfem.asm(
    port_pressure,
    *(
        skfem.FacetBasis(mesh, element[v], facets=boundary["ports"], intorder=3)
        for v in ["p", "u"]
    )
)

t_final = 1.0
dt = 0.05

dirichlet = {
    "u": basis["u"].get_dofs(boundary["wall"]).all(),
    "p": np.concatenate([basis["p"].get_dofs(boundary["ports"]).all()]),
}
inlet_pressure_dofs = basis["p"].get_dofs(boundary["inlet"]).all()

uv_, p_ = (np.zeros(basis[v].N) for v in element.keys())  # penultimate
p__ = np.zeros_like(p_)  # antepenultimate

K = M / dt + L["u"]

t = 0

with TimeSeriesWriter("channel.xdmf") as writer:

    writer.write_points_cells(mesh.p.T, {"triangle": mesh.t.T})

    while t < t_final:

        t += dt

        # Step 1: Momentum prediction (Ern & Guermond 2002, eq. 7.40, p. 274)

        uv = skfem.solve(
            *skfem.condense(
                K,
                (M / dt) @ uv_ - P @ (2 * p_ - p__),
                np.zeros_like(uv_),
                D=dirichlet["u"],
            )
        )

        # Step 2: Projection (Ern & Guermond 2002, eq. 7.41, p. 274)

        dp = np.zeros(basis["p"].N)
        dp[inlet_pressure_dofs] = p_inlet - p_[inlet_pressure_dofs]

        dp = skfem.solve(*skfem.condense(L["p"], B @ uv, dp, D=dirichlet["p"]))

        # Step 3: Recover pressure and velocity (E. & G. 2002, p. 274)

        p = p_ + dp
        print(min(p), "<= p <= ", max(p))

        du = skfem.solve(*skfem.condense(M / dt, -P @ dp, D=dirichlet["u"]))
        u = uv + du

        uv_ = uv
        p_, p__ = p, p_

        # postprocessing
        writer.write_data(
            t,
            point_data={
                "pressure": p,
```

```python
                    "velocity": np.pad(
                        u[basis["u"].nodal_dofs].T, ((0, 0), (0, 1)), "constant"
                    ),
                },
            )

        print(min(u[::2]), "<= u <= ", max(u[::2]), "||v|| = ", np.linalg.norm(u[1::2]))


# References

#  Ern, A., Guermond, J.-L. (2002). _Éléments finis : théorie,
#  applications, mise en œuvre_ (Vol. 36). Paris: Springer. ISBN:
#  3540426159

```

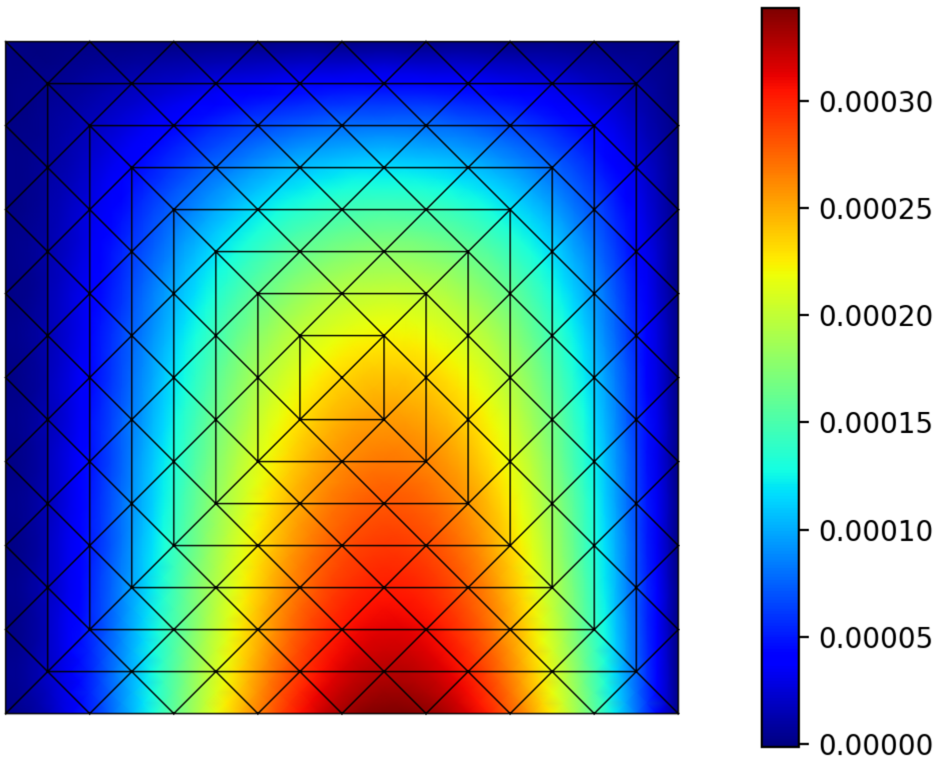
```
0.0 <= p <=  8.0
0.0 <= u <=  0.5322508243494332 ||v|| =  4.026720129295742e-14
0.0 <= p <=  8.0
0.0 <= u <=  0.6309113809656185 ||v|| =  1.436588597740157e-05
0.0 <= p <=  8.0
0.0 <= u <=  0.7616343911111542 ||v|| =  1.90715233764314e-05
0.0 <= p <=  8.0
0.0 <= u <=  0.8422995060746478 ||v|| =  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]:

```python
r"""
This example demonstrates the solution of a slightly more complicated problem
with multiple boundary conditions and a fourth-order differential operator. We
consider the `Kirchhoff plate bending problem
<https://en.wikipedia.org/wiki/Kirchhoff%E2%80%93Love_plate_theory>`_ which
finds its applications in solid mechanics. For a stationary plate of constant
thickness :math:`d`, the governing equation reads: find the deflection :math:`u
: \Omega \rightarrow \mathbb{R}` that satisfies
.. math::
    \frac{Ed^3}{12(1-\nu^2)} \Delta^2 u = f \quad \text{in $\Omega$},
where :math:`\Omega = (0,1)^2`, :math:`f` is a perpendicular force,
:math:`E` and :math:`\nu` are material parameters.
In this example, we analyse a :math:`1\,\text{m}^2` plate of steel with thickness :math:`d=0.1\,\
The Young's modulus of steel is :math:`E = 200 \cdot 10^9\,\text{Pa}` and Poisson
ratio :math:`\nu = 0.3`.
In reality, the operator
.. math::
    \frac{Ed^3}{12(1-\nu^2)} \Delta^2
is a combination of multiple first-order operators:
.. math::
    \boldsymbol{K}(u) = - \boldsymbol{\varepsilon}(\nabla u), \quad \boldsymbol{\varepsilon}(\bol
.. math::
    \boldsymbol{M}(u) = \frac{d^3}{12} \mathbb{C} \boldsymbol{K}(u), \quad \mathbb{C} \boldsymbol
where :math:`\boldsymbol{I}` is the identity matrix. In particular,
.. math::
    \frac{Ed^3}{12(1-\nu^2)} \Delta^2 u = - \text{div}\,\textbf{div}\,\boldsymbol{M}(u).
```

```
There are several boundary conditions that the problem can take.
The *fully clamped* boundary condition reads
.. math::
    u = \frac{\partial u}{\partial \boldsymbol{n}} = 0,
where :math:`\boldsymbol{n}` is the outward normal.
Moreover, the *simply supported* boundary condition reads
.. math::
    u = 0, \quad M_{nn}(u)=0,
where :math:`M_{nn} = \boldsymbol{n} \cdot (\boldsymbol{M} \boldsymbol{n})`.
Finally, the *free* boundary condition reads
.. math::
    M_{nn}(u)=0, \quad V_{n}(u)=0,
where :math:`V_n` is the `Kirchhoff shear force <https://arxiv.org/pdf/1707.08396.pdf>`_. The exa
definition is not needed here as this boundary condition is a
natural one.
The correct weak formulation for the problem is: find :math:`u \in V` such that
.. math::
    \int_\Omega \boldsymbol{M}(u) : \boldsymbol{K}(v) \,\mathrm{d}x = \int_\Omega fv \,\mathrm{d}
where :math:`V` is now a subspace of :math:`H^2` with the essential boundary
conditions for :math:`u` and :math:`\frac{\partial u}{\partial \boldsymbol{n}}`.
Instead of constructing a subspace for :math:`H^2`, we discretise the problem
using the `non-conforming Morley finite element
<https://users.aalto.fi/~jakke74/WebFiles/Slides-Niiranen-ADMOS-09.pdf>`_ which
is a piecewise quadratic :math:`C^0`-continuous element for biharmonic problems.
The full source code of the example reads as follows:
.. literalinclude:: examples/ex02.py
    :start-after: EOF"""
from skfem import *
from skfem.models.poisson import unit_load
import numpy as np
%config InlineBackend.figure_formats = ['retina']

m = (
    MeshTri.init_symmetric()
    .refined(3)
    .with_boundaries(
        {
            "left": lambda x: x[0] == 0,
            "right": lambda x: x[0] == 1,
            "top": lambda x: x[1] == 1,
        }
    )
)

e = ElementTriMorley()
ib = Basis(m, e)


@BilinearForm
def bilinf(u, v, w):
    from skfem.helpers import dd, ddot, trace, eye
    d = 0.1
    E = 200e9
    nu = 0.3

    def C(T):
        return E / (1 + nu) * (T + nu / (1 - nu) * eye(trace(T), 2))

    return d**3 / 12.0 * ddot(C(dd(u)), dd(v))


K = asm(bilinf, ib)
f = 1e6 * asm(unit_load, ib)

D = np.hstack([ib.get_dofs("left"), ib.get_dofs({"right", "top"}).all("u")])

x = solve(*condense(K, f, D=D))

def visualize():
    from skfem.visuals.matplotlib import draw, plot
    ax = draw(m)
    return plot(ib,
                x,
                ax=ax,
                shading='gouraud',
                colorbar=True,
                nrefs=2)

if __name__ == "__main__":
    visualize().show()
```



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

In [12]:

```
r"""Linear elasticity.
This example solves the linear elasticity problem using trilinear elements.  The
weak form of the linear elasticity problem is defined in
:func:`skfem.models.elasticity.linear_elasticity`.
"""

import numpy as np
from skfem import *
from skfem.models.elasticity import linear_elasticity, lame_parameters

m = MeshHex().refined(3)
e1 = ElementHex1()
e = ElementVector(e1)
ib = Basis(m, e, MappingIsoparametric(m, e1), 3)

K = asm(linear_elasticity(*lame_parameters(1e3, 0.3)), ib)

dofs = {
    'left' : ib.get_dofs(lambda x: x[0] == 0.0),
    'right': ib.get_dofs(lambda x: x[0] == 1.0),
}

u = ib.zeros()
u[dofs['right'].nodal['u^1']] = 0.3

I = ib.complement_dofs(dofs)

u = solve(*condense(K, x=u, I=I))

sf = 1.0
m = m.translated(sf * u[ib.nodal_dofs])

if __name__ == "__main__":
    from os.path import splitext
    from sys import argv

    #m.save(splitext(argv[0])[0] + '.vtk')
    m.save('testimg' + '.vtk')
```