parabolic_solution

April 6, 2022

1 Problem

$$\partial_t u - \nabla \cdot K(\nabla u)\nabla u = f \tag{1}$$

$f = f(x,t)$ is a time dependent forcing term and the diffusion tensor is

$$K(\nabla u) = \frac{2}{1+\sqrt{1+4|\nabla u|}} \tag{2}$$

On the boundary we prescribe Neumann boundary conditions and require initial conditions $u(\cdot,0) = u_0(\cdot)$.

We solve this problem in variational form using Implicit Euler in time

$$\int_\Omega \frac{u^{n+1}-u^n}{\Delta t} v + K(\nabla u^{n+1})\nabla u^{n+1}\cdot\nabla v\, dx = \int_\Omega f(x,t^n+\Delta t)v\, dx + \int_{\partial\Omega} g(x,t^n+\Delta t)\cdot n\, v\, ds\ .$$

on a domain $\Omega = [0,1]^2$. We choose f, g so that the exact solution is

$$u(x,t) = e^{-2t}\left(\frac{1}{2}(x^2+y^2) - \frac{1}{3}(x^3-y^3)\right) + 1$$

Setup grid and space

[1]:
```
import numpy as np
from dune.grid import cartesianDomain
from dune.alugrid import aluConformGrid as leafGridView
from dune.fem.space import lagrange as solutionSpace
from dune.fem.scheme import galerkin as solutionScheme
from dune.fem.function import integrate, uflFunction

domain = cartesianDomain([0, 0], [1, 1], [10, 10])
view = leafGridView(domain)
space = solutionSpace(view, order=2)
```

```
Created parallel ALUGrid<2,2,simplex,conforming> from input stream.


GridParameterBlock: Parameter 'refinementedge' not specified, defaulting to
'ARBITRARY'.
WARNING (ignored): Could not open file 'alugrid.cfg', using default values 0 <
```

```
[balance] < 1.2, partitioning method 'ALUGRID_SpaceFillingCurve(9)'.

You are using DUNE-ALUGrid, please don't forget to cite the paper:
Alkaemper, Dedner, Kloefkorn, Nolte. The DUNE-ALUGrid Module, 2016.
```

Import UFL variables and define spatial coordinate, test/trial function

[2]:
```
from dune.ufl import DirichletBC, Constant
from ufl import TestFunction, TrialFunction, SpatialCoordinate,\
      FacetNormal, dx, ds, div, grad, dot, inner, sqrt, exp, sin

x = SpatialCoordinate(space)
n = FacetNormal(space)
u = TrialFunction(space)
v = TestFunction(space)
```

Define initial condition and exact solution

[3]:
```
initial = 1/2*(x[0]**2+x[1]**2)-1/3*(x[0]**3-x[1]**3)+1
exact   = lambda t: exp(-2*t)*(initial - 1) + 1
dtExact = lambda s: -2*exp(-2*s)*(initial - 1)
```

Define discrete functions, one for u^{n+1} and one for u^n. We also define two constants which can be used as floats in ufl expressions but can be changed at a later stage without requiring any recompilation.

[4]:
```
dt = Constant(0, name="dt")     # time step
t  = Constant(0, name="t")      # current time

u_h   = space.interpolate(exact(t), name='u_h')
u_h_n = u_h.copy(name="previous")
```

Now setup the model and the scheme:

$$\int_\Omega \frac{u^{n+1}-u^n}{\Delta t} v + K(\nabla u^{n+1}) \nabla u^{n+1} \cdot \nabla v \, dx = \int_\Omega f(x, t^n + \Delta t) v \, dx + \int_{\partial\Omega} g(x, t^n + \Delta t) \cdot n \, v \, ds \, .$$

[5]:
```
abs_du = lambda u: sqrt(inner(grad(u), grad(u)))
K = lambda u: 2/(1 + sqrt(1 + 4*abs_du(u)))
f = lambda s: dtExact(t) - div( K(exact(s))*grad(exact(s)) )
g = lambda s: K(exact(s))*grad(exact(s))
```

1.1 Task

Implement the form **a** and right hand side **b** as described above, using terms K, f, and g.

[6]:
```
a = ( inner((u - u_h_n)/dt, v) + inner(K(u)*grad(u), grad(v)) ) * dx
b = f(t+dt)*v*dx + inner( dot(g(t+dt),n), v) * ds
```

A Galerkin scheme is created that allows us to solve the equation $a = b$.

[7]:
```
scheme = solutionScheme(a == b, solver='cg')
```

Time loop: first choose a time step and then iterate from $t^0 = 0$ to $t^N = N\Delta t = 0.25$. We write vtk files every after every 0.05 time interval.

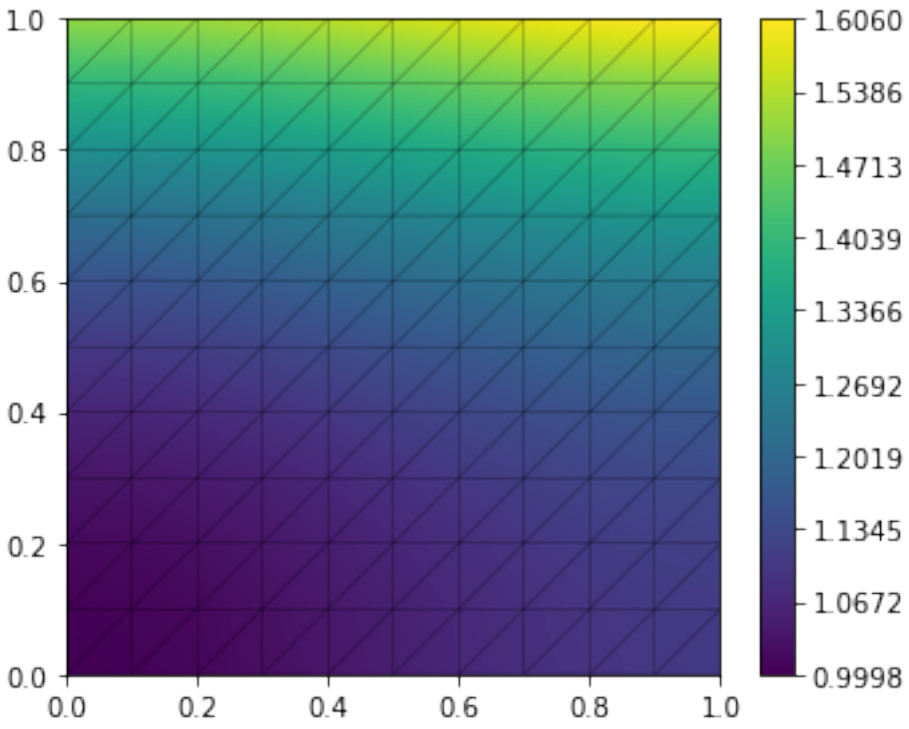
[8]:
```
scheme.model.dt = 0.0025
saveInterval = 0.05
vtk = view.sequencedVTK("reactionsystem", pointdata=[u_h])
nextSaveTime = saveInterval
vtk()

time = 0
while time <= 0.25:
    u_h_n.assign(u_h)
    scheme.solve(target=u_h)
    time += scheme.model.dt
    scheme.model.t = time
    if time >= nextSaveTime or time >= 0.25:
        vtk()
        nextSaveTime += saveInterval
```

Plot result and compute error

[9]:
```
u_h.plot()

error = dot(u_h-exact(t),u_h-exact(t))
error = np.sqrt( integrate(view, error, order=5) )
print("error at final time=", error)
```




```
error at final time= 0.16418668639785838
```