AFEM

Skouroliakou Vasiliki/09419021

May 2020

1 Personal Info

My name is Skouroliakou Vasiliki and I am currently a student at MSc program "Applied Mathematical Sciences" at Technical University of Athens. I am also a graduated student of University of Thessaly and specifically the Department of Electrical and Computer Engineering.

2 The equation

Consider the nonlinear diffusion equation:

$$u_t = \bigtriangledown \cdot (a(u) \bigtriangledown u) + f$$

$(t, x, y) \in [0, 1] \times \Omega \subset \mathbf{R^2}$. We also assume homogeneous Neumann conditions on the entire boundary.

The non-linear term a(u) is usually of the form $a(u) = m|u|^{m-1} + u^3,\ m > 1$.

3 Discretization

We use a Backward Euler scheme to discrete the equation in time domain [2], [1]. The Backward Euler states:

$$u_t = \frac{u^n - u^{n-1}}{\Delta t}$$

so we get:

$$\frac{u^n - u^{n-1}}{\Delta t} = \bigtriangledown \cdot (a(u^n) \bigtriangledown u^n) + f \longrightarrow$$
$$u^n - \Delta t \bigtriangledown \cdot (a(u^n) \bigtriangledown u^n) - \Delta t f = u^{n-1}$$

We consider the finite element space V to be the H_0^1.

We multiply the above equation with a test function $v \in V$ which is zero on the boundary:

$$\int_\Omega (u^n - \Delta t \bigtriangledown \cdot (a(u^n) \bigtriangledown u^n) - \Delta t f - u^{n-1}) v d\hat{x} = 0$$

We use Green's formula to obtain:

$$\int_\Omega (u^n v + \Delta t a(u^n) \bigtriangledown u^n \cdot \bigtriangledown v - \Delta t f v - u^{n-1} v) d\hat{x} = 0$$

for every $v \in V$.

Now we use set $v = \phi_i$ and we also set $u^n = u$ and $u^{n-1} = u^0$:

$$F_i = \int_\Omega (u\phi_i + \Delta t a(u^n) \bigtriangledown u \cdot \bigtriangledown \phi_i - \Delta t f \phi_i - u^0 \phi_i) d\hat{x}$$

The above equation has anon-linear term, i.e. $a(u^n)$. In order to deal with this term we use its most recent approximation at every step. We denote the approximation u^-, starting from $u^- = 0$. Moreover we use the following representation for u:

$$u = \sum_j c_j \phi_j$$

and the equation is now written:

$$F_i \approx \hat{F}_i = \sum_j c_j \int_\Omega (\phi_j \phi_i + \Delta t a(u^-) \bigtriangledown \phi_j \cdot \bigtriangledown \phi_i d\hat{x} - \int_\Omega \Delta t f \phi_i + u^0 \phi_i d\hat{x} \tag{1}$$

The Newton method solves the linear system:

$$J^k (\delta u)^{k+1} = -F^k$$

for every iteration k where $(\delta u)^{k+1} = u^{k+1} - u^k$. The Jacobian matrix is denoted by J and it is equal to $J_{ij} = \frac{\partial F_i}{\partial c_j}$.

The Newton method stops when the $|(\delta u)^{k+1}| < tol$, for a given tolerance.

The Newton method is used to find u at every time step n, starting from $u^0 = 0$.

4 Python Code

At this section the Python code using Netgen/NGSolve is presented. The package is downloaded from here. The code below is based on this documentation.

We set a(u) for m=3. We also set the initial guess for Newton method and the solution for time t_0 equal to zero. The right-hand side function f is set equal to 1. We will also add inhomogeneous Dirichlet conditions on top and bottom boundary.

Run the script using the command

```
netgen diffusion.py
```

```python
from netgen import gui
from ngsolve import *
from netgen.geom2d import SplineGeometry

geo = SplineGeometry()
geo.AddRectangle( (-1, -1), (1, 1),
                  bcs = ("bottom", "right", "top", "left"))
mesh = Mesh( geo.GenerateMesh(maxh=0.25))
Draw(mesh)
fes = H1(mesh, order=3)

u,v = fes.TnT()

time = 0.0
dt = 0.01

gfu = GridFunction(fes)
gfuold = GridFunction(fes)

a = BilinearForm (fes, symmetric=False)
a +=( u*v +dt*3*u**2*grad(u)*grad(v) +dt*u**3*grad(u)*grad(v) -dt*1*v -gfuold*v ) * dx

from math import pi
gfu = GridFunction(fes)
gfu.Set(sin(2*pi*x))
```

```python
Draw(gfu,mesh,"u")
SetVisualization (deformation=True)
t = 0

def SolveNonlinearMinProblem(a,gfu,tol=1e-13,maxits=25):
    res = gfu.vec.CreateVector()
    du  = gfu.vec.CreateVector()

    for it in range(maxits):
        print ("Newton iteration {:3}".format(it),end="")
        print ("energy = {:16}".format(a.Energy(gfu.vec)),end="")

        #solve linearized problem:
        a.Apply (gfu.vec, res)
        a.AssembleLinearization (gfu.vec)
        inv = a.mat.Inverse(fes.FreeDofs())
        du.data = inv * res

        #update iteration
        gfu.vec.data -= du

        #stopping criteria
        stopcritval = sqrt(abs(InnerProduct(du,res)))
        print ("<A u",it,", A u",it,">_{-1}^0.5 = ", stopcritval)
        if stopcritval < tol:
            break
        Redraw(blocking=True)


for timestep in range(50):
    gfuold.vec.data = gfu.vec
    SolveNonlinearMinProblem(a,gfu)
    Redraw()
    t += dt
    print("t = ", t)
```

Script 1: diffusion.py: Python script

The following picture shows the solution at the final time.

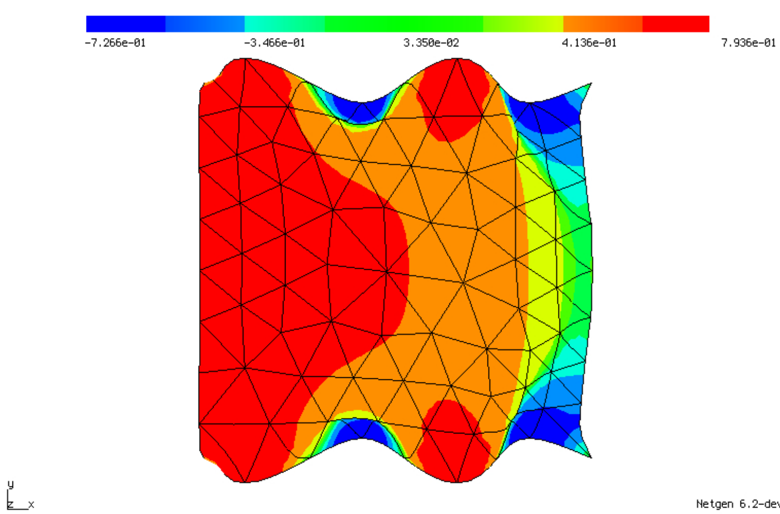

Figure 1: Solution at the final time

References

[1] Emmrich Etienne and David Šiška. Full discretization of the porous medium/fast diffusion equation based on its very weak formulation. *Communications in Mathematical Sciences*, 10(4):1055–1080, 2012.

[2] Hans Petter Langtangen. Solving nonlinear ode and pde problems. *Center for Biomedical Computing, Simula Research Laboratory and Department of Informatics, University of Oslo*, 2016.