PGE 382 - Numerical Methods in Petroleum and Geosystems Engineering

Renato Poli - rep2656

CP7 - Apr, 3rd

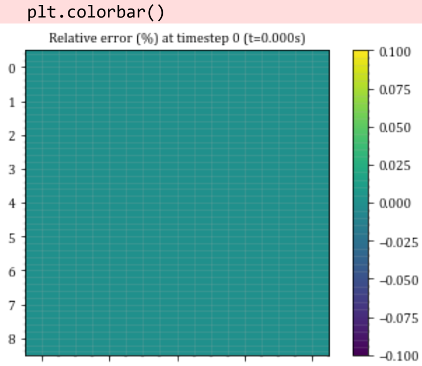
In [3]:
```
from math import pi, sin, cos, exp
import numpy as np
from numpy import linspace, zeros, arange
from numpy import ix_ as ix
np.set_printoptions(threshold=10000, linewidth=10000)
from numpy import exp, linspace, vectorize
import matplotlib.pyplot as plt
plt.style.use('paper.mplstyle')

dx = 0.125 ; dy = dx ; dz = dx
Tf = 0.25 ; Nt=10
mu = 0.5

X = np.arange(0,1+dx,dx) ; Ni = len(X)
Y = np.arange(0,1+dy,dy) ; Nj = len(Y)
Z = np.arange(0,1+dz,dz) ; Nk = len(Z)

Nijk = Ni * Nj * Nk

dt = Tf/Nt ; Nt = Nt + 1

# Global index
def _(i,j,k) : return i + Ni*j+ Ni*Nj*k
def exact( t,x,y,z) :
    return 1/( 1+exp(x/2/mu + y/2/mu + z/2/mu - 3*t/4/mu) )

def build_exact() :
    global Uexact, Nt, Ni, Nj, Nk
    Uexact = zeros( [Nt,Ni,Nj,Nk] )
    for n in arange(0,Nt) :
        for i in arange(Ni) :
            for j in arange(Nj) :
                for k in arange(Nk) :
                    Uexact[n,i,j,k] = exact(n*dt,i*dx,j*dy,k*dz)

# The list of free dofs
def build_Df() :
    global Df, Dp, Nt, Ni, Nj, Nk

    Df=[]
    for i in arange(1,Ni-1) :
        for j in arange(1,Nj-1) :
            for k in arange(1,Nk-1) :
                Df.append( _(i,j,k) )
    Dp=[]
    for i in arange(0,Ni) :
        for j in arange(0,Nj) :
            Dp.append( _(0,i,j) )
            Dp.append( _(Ni-1,i,j) )

            Dp.append( _(i,0,j) )
            Dp.append( _(i,Nj-1,j) )

            Dp.append( _(i,j,0) )
            Dp.append( _(i,j,Nk-1) )

# Assign BCs to solution vector U
def init_bcs() :
    global U, Uexact
    U = zeros( [Nt,Ni,Nj,Nk] )
    U[0,:,:,:] = Uexact[0,:,:,:] #IC
    U[:,0,:,:] = Uexact[:,0,:,:] #I=0
    U[:,:,0,:] = Uexact[:,:,0,:] #J=0
    U[:,:,:,0] = Uexact[:,:,:,0] #K=0
    U[:,Ni-1,:,:] = Uexact[:,Ni-1,:,:]
    U[:,:,Nj-1,:] = Uexact[:,:,Nj-1,:]
    U[:,:,:,Nk-1] = Uexact[:,:,:,Nk-1]

# Global solution vector
build_exact()
init_bcs()
build_Df()

Dff_ix = ix( Df, Df )
Df_ix = ix( Df )
Dp_ix = ix(Dp)
for n in arange(1,Nt) :
    print(f"Solving timestep {n} ...")

    # Solution from the previous TS
```

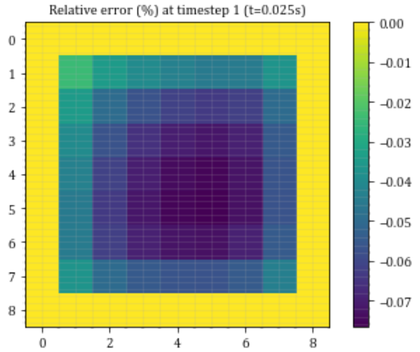
```python
    Un = U[n-1,:,:,:].flatten()
    # Apply new BCs in Uk
    Uk = Un.copy()
    Uk[Dp_ix] = Uexact[n,:,:,:].flatten()[Dp_ix]

    nk = 0 #newton loop index
    while(1) :
        # Jacobians and functions
        J = zeros([ Nijk, Nijk ] )
        F = zeros( Nijk )

        for i in arange(1,Ni-1) :
            for j in arange(1,Nj-1) :
                for k in arange(1,Nk-1) :
                    _ijk = _(i,j,k)
                    _0jk = _(i-1,j,k)
                    _1jk = _(i+1,j,k)
                    _i0k = _(i,j-1,k)
                    _i1k = _(i,j+1,k)
                    _ij0 = _(i,j,k-1)
                    _ij1 = _(i,j,k+1)

                    J[_ijk,_ijk] += -2*Uk[_ijk]*(1/dx + 1/dy + 1/dz) +\
                                    Uk[_0jk]/dx + Uk[_i0k]/dy + Uk[_ij0]/dz +\
                                    (-2*mu)*(1/dx/dx + 1/dy/dy + 1/dz/dz) +\
                                    (-1/dt)

                    J[_ijk,_0jk] += Uk[_ijk]/dx + mu/dx/dx
                    J[_ijk,_1jk] += mu/dx/dx
                    J[_ijk,_i0k] += Uk[_ijk]/dy + mu/dy/dy
                    J[_ijk,_i1k] += mu/dy/dy
                    J[_ijk,_ij0] += Uk[_ijk]/dz + mu/dz/dz
                    J[_ijk,_ij1] += mu/dz/dz

                    F[_ijk] += (-Uk[_ijk])*(Uk[_ijk]-Uk[_0jk])/dx
                    F[_ijk] += (-Uk[_ijk])*(Uk[_ijk]-Uk[_i0k])/dy
                    F[_ijk] += (-Uk[_ijk])*(Uk[_ijk]-Uk[_ij0])/dz
                    F[_ijk] += mu*(Uk[_0jk] - 2*Uk[_ijk] + Uk[_1jk])/dx/dx
                    F[_ijk] += mu*(Uk[_i0k] - 2*Uk[_ijk] + Uk[_i1k])/dy/dy
                    F[_ijk] += mu*(Uk[_ij0] - 2*Uk[_ijk] + Uk[_ij1])/dz/dz
                    F[_ijk] += ( Un[_ijk] - Uk[_ijk] )/dt

        # Solve for the unknowns, update vectors
        Kf = J[Dff_ix] ; Ff = F[Df_ix]
        delta = np.linalg.solve( Kf, -Ff )
        # Collect solution
        Uk[Df_ix] += delta
        err = np.linalg.norm(delta)

        # Check for convergence
        nk += 1
        print(f"   Newton iteration #{nk} ... (err={err:.3e})")
        if err < 1e-15 : break
        if nk > 50 : break

    U[n,:,:,:] = Uk.reshape(Ni,Nj,Nk)
    # Compare with exact
    DIFF = U[n,:,:,:] - Uexact[n,:,:,:]
    print(f"Difference from exact: {np.linalg.norm(DIFF)}")
```

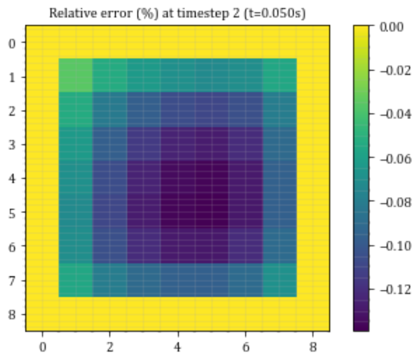
```
Solving timestep 1 ...
   Newton iteration #1 ... (err=1.105e-01)
   Newton iteration #2 ... (err=5.436e-05)
   Newton iteration #3 ... (err=3.936e-11)
   Newton iteration #4 ... (err=1.731e-16)
Difference from exact: 0.0017434691489342754
Solving timestep 2 ...
   Newton iteration #1 ... (err=1.122e-01)
   Newton iteration #2 ... (err=5.558e-05)
   Newton iteration #3 ... (err=4.055e-11)
   Newton iteration #4 ... (err=1.951e-16)
Difference from exact: 0.002993242288226833
Solving timestep 3 ...
   Newton iteration #1 ... (err=1.141e-01)
   Newton iteration #2 ... (err=5.703e-05)
   Newton iteration #3 ... (err=4.204e-11)
   Newton iteration #4 ... (err=1.835e-16)
Difference from exact: 0.003914008727057912
Solving timestep 4 ...
   Newton iteration #1 ... (err=1.161e-01)
   Newton iteration #2 ... (err=5.857e-05)
   Newton iteration #3 ... (err=4.366e-11)
   Newton iteration #4 ... (err=2.137e-16)
Difference from exact: 0.00460208331625954
Solving timestep 5 ...
   Newton iteration #1 ... (err=1.182e-01)
   Newton iteration #2 ... (err=6.017e-05)
   Newton iteration #3 ... (err=4.532e-11)
   Newton iteration #4 ... (err=2.122e-16)
Difference from exact: 0.005120862885483042
Solving timestep 6 ...
   Newton iteration #1 ... (err=1.203e-01)
   Newton iteration #2 ... (err=6.179e-05)
   Newton iteration #3 ... (err=4.700e-11)
   Newton iteration #4 ... (err=2.206e-16)
Difference from exact: 0.005514408063422826
Solving timestep 7 ...
   Newton iteration #1 ... (err=1.224e-01)
   Newton iteration #2 ... (err=6.342e-05)
   Newton iteration #3 ... (err=4.866e-11)
   Newton iteration #4 ... (err=1.963e-16)
Difference from exact: 0.005814073071049053
Solving timestep 8 ...
   Newton iteration #1 ... (err=1.245e-01)
   Newton iteration #2 ... (err=6.505e-05)
   Newton iteration #3 ... (err=5.030e-11)
   Newton iteration #4 ... (err=2.258e-16)
Difference from exact: 0.006042380575947817
Solving timestep 9 ...
   Newton iteration #1 ... (err=1.266e-01)
   Newton iteration #2 ... (err=6.666e-05)
   Newton iteration #3 ... (err=5.190e-11)
   Newton iteration #4 ... (err=2.145e-16)
Difference from exact: 0.006215548934590986
Solving timestep 10 ...
   Newton iteration #1 ... (err=1.287e-01)
   Newton iteration #2 ... (err=6.825e-05)
   Newton iteration #3 ... (err=5.345e-11)
   Newton iteration #4 ... (err=2.372e-16)
Difference from exact: 0.006345240087418409
```

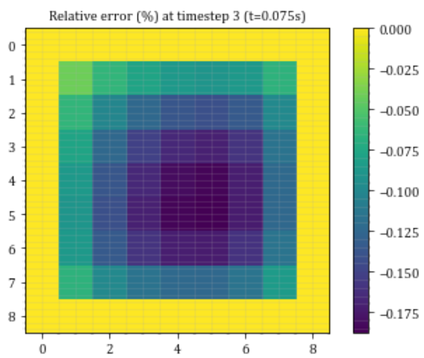
In [4]:
```python
k=5
for n in arange(Nt) :
    plt.figure()
    err = ( 1 - U[n,:,:,k] / Uexact[n,:,:,k] ) * 100
    plt.imshow( err )
    plt.colorbar()
    plt.title(f"Relative error (%) at timestep {n} (t={n*dt:.3f}s)")
```

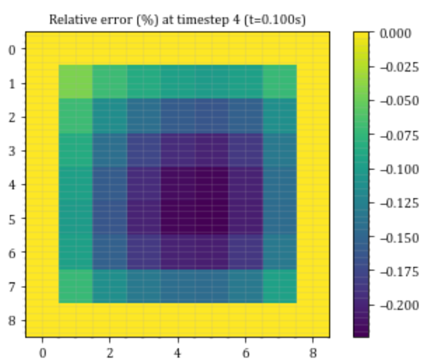
```
C:\Users\rebpo\AppData\Local\Temp\ipykernel_16848\1638510875.py:6: MatplotlibDeprecationWarning: Auto-removal of grids by pcolor() and pcolormesh
() is deprecated since 3.5 and will be removed two minor releases later; please call grid(False) first.
  plt.colorbar()
```

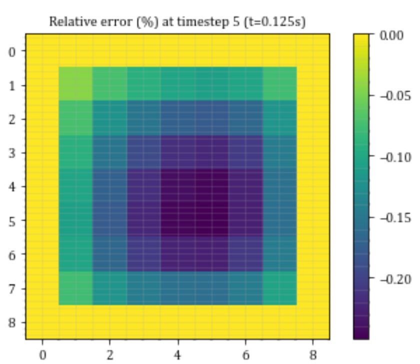



Relative error (%) at timestep 1 (t=0.025s)
0 1 2 3 4 5 6 7 8
0 2 4 6 8
0.00 −0.01 −0.02 −0.03 −0.04 −0.05 −0.06 −0.07

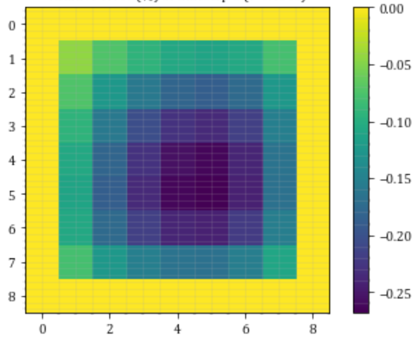

Relative error (%) at timestep 2 (t=0.050s)
0 1 2 3 4 5 6 7 8
0 2 4 6 8
0.00 −0.02 −0.04 −0.06 −0.08 −0.10 −0.12


Relative error (%) at timestep 3 (t=0.075s)
0 1 2 3 4 5 6 7 8
0 2 4 6 8
0.000 −0.025 −0.050 −0.075 −0.100 −0.125 −0.150 −0.175

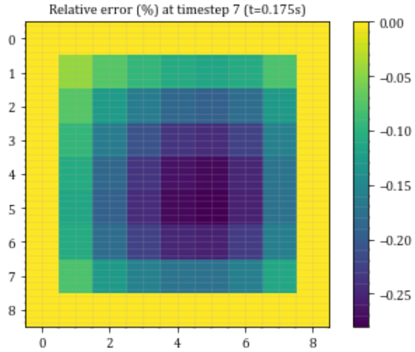

Relative error (%) at timestep 4 (t=0.100s)
0 1 2 3 4 5 6 7 8
0 2 4 6 8
0.000 −0.025 −0.050 −0.075 −0.100 −0.125 −0.150 −0.175 −0.200


Relative error (%) at timestep 5 (t=0.125s)
0 1 2 3 4 5 6 7 8
0 2 4 6 8
0.00 −0.05 −0.10 −0.15 −0.20

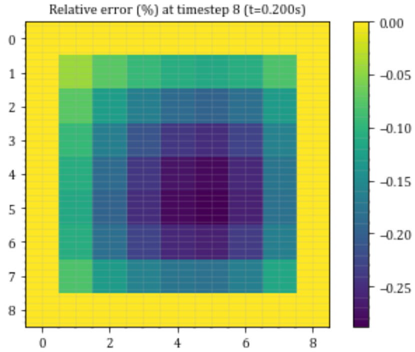
Relative error (%) at timestep 6 (t=0.150s)



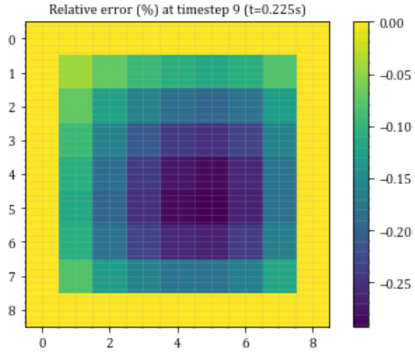
Relative error (%) at timestep 7 (t=0.175s)



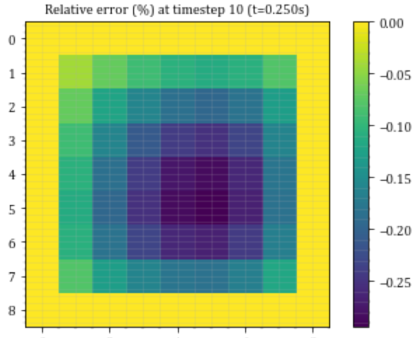
Relative error (%) at timestep 8 (t=0.200s)



Relative error (%) at timestep 9 (t=0.225s)



Relative error (%) at timestep 10 (t=0.250s)



CP7

Residual

$$R = -uu_x + \mu u_{xx} - uu_y + \mu u_{yy} - uu_z + \mu u_{zz} - u_t$$

reorganize

$$R = (-u)\underbrace{(u_x + u_y + u_z)}_{1} + \mu\underbrace{(u_{xx} + u_{yy} + u_{zz})}_{2} - \underbrace{u_t}_{3}$$

1 $(-u)\left(\frac{u - u_{i-1}}{\Delta x} + \frac{u - u_{j-1}}{\Delta y} + \frac{u - u_{k-1}}{\Delta z}\right)$

$$\boxed{u^2\left(-\frac{1}{\Delta x} - \frac{1}{\Delta y} - \frac{1}{\Delta z}\right) + u\left(\frac{u_{i-1}}{\Delta x} + \frac{u_{j-1}}{\Delta y} + \frac{u_{k-1}}{\Delta z}\right)}$$

2 $\mu \frac{u_{i-1} - 2u + u_{i+1}}{\Delta x^2} + \mu \frac{u_{j-1} - 2u + u_{j+1}}{\Delta y^2} + \mu \frac{u_{k-1} - 2u + u_{k+1}}{\Delta z^2}$

$$\boxed{\mu(-2u)\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}\right) + \frac{\mu}{\Delta x^2}(u_{i-1} + u_{i+1}) + \frac{\mu}{\Delta y^2}(u_{j-1} + u_{j+1}) + \frac{\mu}{\Delta z^2}(u_{k-1} + u_{k+1})}$$

3 $-\frac{u^{n+1} - u^n}{\Delta t}$

Calculate the jacobian:

$$\frac{\partial R}{\partial u} = (-2u)\left(\frac{1}{\Delta x} + \frac{1}{\Delta y} + \frac{1}{\Delta z}\right) + \left(\frac{u_{i-1}}{\Delta x} + \frac{u_{j-1}}{\Delta y} + \frac{u_{k-1}}{\Delta z}\right) +$$

$$+ (-2\mu)\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}\right) - \frac{1}{\Delta t}$$

$$\frac{\partial R}{\partial u_{i-1}} = \frac{u}{\Delta x} + \frac{\mu}{\Delta x^2}$$

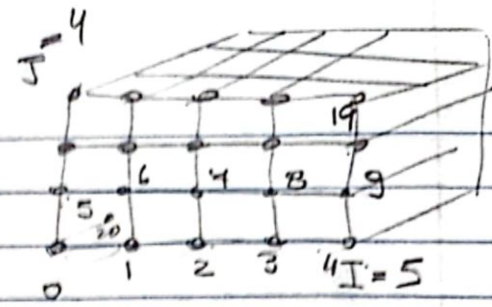
$$\frac{\partial R}{\partial u_{i+1}} = \frac{\mu}{\Delta x^2}$$

$$\frac{\partial R}{\partial u_{j-1}} = \frac{u}{\Delta y} + \frac{\mu}{\Delta y^2}$$

$$\frac{\partial R}{\partial u_{j+1}} = \frac{\mu}{\Delta y^2}$$

$$\frac{\partial R}{\partial u_{k-1}} = \frac{u}{\Delta z} + \frac{\mu}{\Delta z^2}$$

$$\frac{\partial R}{\partial u_{k+1}} = \frac{\mu}{\Delta z^2}$$



Jacobian stencil

$$\boxed{\xi = i + Ij + (I \times J)k}$$

$20 = 0 + 0 + (4 \times 5 \times 1)$.

$\boxed{\text{def } \xi(i,j,k) : \text{return } i + Ij + (IJ)k}$

---

for n in (1, NT) :

$U_K = U_N(n-1)$

While (1) :

for $i = 0 : N_I$

for $j = 0 : N_J$

for $k = 0 : N_K$

$\xi = \xi(i,j,k)$

$J[\xi, \xi] = \partial R/\partial u$

$J[\xi, \xi P00] = \partial R/\partial u_{i+1}$

$J[\xi, \xi N00] = \partial R/\partial u_{i-1}$

$J[\xi, \xi 0P0] = \partial R/\partial u_{j+1}$

⋮

$F[\xi] += f(U_K, U_{N-1}, \xi)$

$dU_K \leftarrow \text{solve}(J, -f)$

$U_K += dU_K$

stop?

$U_N = U_K$.