nonlinear

November 11, 2022

1 3.7 Nonlinear problems

We want to solve a nonlinear PDE.

1.1 A simple scalar PDE

We consider the simple PDE

$$-\Delta u + 3u^3 = 1 \text{ in } \Omega$$

on the unit square $\Omega = (0,1)^2$.

We note that this PDE can also be formulated as a nonlinear minimization problem (cf. 3.8).

[1]:
```
from ngsolve import *
from netgen.geom2d import unit_square
from ngsolve.webgui import Draw
mesh = Mesh (unit_square.GenerateMesh(maxh=0.3))
```

```
importing NGSolve-6.2.2204
```

In NGSolve we can solve the PDE conveniently using the *linearization* feature of `SymbolicBFI`.

The `BilinearForm` (which is not bilinear!) needed in the weak formulation is

$$A(u,v) = \int_\Omega \nabla u \nabla v + 3u^3 v - 1v \; dx \quad (= 0 \; \forall \; v \in H_0^1)$$

[14]:
```
V = H1(mesh, order=3, dirichlet=[1,2,3,4])
u,v = V.TnT()
a = BilinearForm(V)
a += (grad(u) * grad(v) + 3*u**3*v- 1 * v)*dx
```

1.1.1 Newton's method

We use Newton's method and make the loop:

- Given an initial guess u^0
- loop over $i = 0, ..$ until convergence:
  - Compute linearization: $Au^i + \delta A(u^i)\Delta u^i = 0$:

    * $f^i = Au^i$
    * $B^i = \delta A(u^i)$
    * Solve $B^i \Delta u^i = -f^i$
  - Update $u^{i+1} = u^i + \Delta u^i$
  - Evaluate stopping criteria

As a stopping criteria we take $\langle Au^i, \Delta u^i \rangle = \langle Au^i, Au^i \rangle_{(B^i)^{-1}} < \varepsilon$.

[12]:
```
def SimpleNewtonSolve(gfu,a,tol=1e-13,maxits=25):
    res = gfu.vec.CreateVector()
    du = gfu.vec.CreateVector()
    fes = gfu.space
    for it in range(maxits):
        print ("Iteration {:3}  ".format(it),end="")
        a.Apply(gfu.vec, res)
        a.AssembleLinearization(gfu.vec)
        du.data = a.mat.Inverse(fes.FreeDofs()) * res
        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

len(gfu.vec.data)
```

[12]: 127

[9]:
```
gfu = GridFunction(V)
Draw(gfu,mesh,"u")
SimpleNewtonSolve(gfu,a)
print(gfu)
```

```
WebGuiWidget(layout=Layout(height='50vh', width='100%'), value={'gui_settings':␣
↪{}, 'ngsolve_version': '6.2.22…

Iteration   0  <A u 0 , A u 0 >_{-1}^0.5 =  0.18743829125307696
Iteration   1  <A u 1 , A u 1 >_{-1}^0.5 =  9.417800751712506e-05
Iteration   2  <A u 2 , A u 2 >_{-1}^0.5 =  8.541507611851595e-11
Iteration   3  <A u 3 , A u 3 >_{-1}^0.5 =  4.281213551704198e-17
gridfunction 'gfu' on space 'H1HighOrderFESpace(h1ho)'
nested = 0
autoupdate = 0
```

There are also some solvers shipped with NGSolve now:

[6]:
```python
from ngsolve.solvers import *
help(Newton)
```

```
Help on function Newton in module ngsolve.nonlinearsolvers:

Newton(a, u, freedofs=None, maxit=100, maxerr=1e-11, inverse='umfpack',
dirichletvalues=None, dampfactor=1, printing=True, callback=None)
    Newton's method for solving non-linear problems of the form A(u)=0.

    Parameters
    ----------
    a : BilinearForm
      The BilinearForm of the non-linear variational problem. It does not have
to be assembled.

    u : GridFunction
      The GridFunction where the solution is saved. The values are used as
initial guess for Newton's method.

    freedofs : BitArray
      The FreeDofs on which the assembled matrix is inverted. If argument is
'None' then the FreeDofs of the underlying FESpace is used.

    maxit : int
      Number of maximal iteration for Newton. If the maximal number is reached
before the maximal error Newton might no converge and a warning is displayed.

    maxerr : float
      The maximal error which Newton should reach before it stops. The error is
computed by the square root of the inner product of the residuum and the
correction.

    inverse : string
      A string of the sparse direct solver which should be solved for inverting
the assembled Newton matrix.

    dampfactor : float
      Set the damping factor for Newton's method. If dampfactor is 1 then no
damping is done. If value is < 1 then the damping is done by the formula
'min(1,dampfactor*numit)' for the correction, where 'numit' denotes the Newton
iteration.

    printing : bool
      Set if Newton's method should print informations about the actual
iteration like the error.

    Returns
```

```
    -------
    (int, int)
      List of two integers. The first one is 0 if Newton's method did converge,
-1 otherwise. The second one gives the number of Newton iterations needed.
```

[7]:
```
gfu.vec[:]=0
Newton(a,gfu,freedofs=gfu.space.
 ↪FreeDofs(),maxit=100,maxerr=1e-11,inverse="umfpack",dampfactor=1,printing=True)
```

```
Newton iteration  0
```

```
---------------------------------------------------------------------------
NgException                               Traceback (most recent call last)
/tmp/ipykernel_102887/501649225.py in <module>
      1 gfu.vec[:]=0
----> 2 Newton(a,gfu,freedofs=gfu.space.
 ↪FreeDofs(),maxit=100,maxerr=1e-11,inverse="umfpack",dampfactor=1,printing=True)

~/.local/lib/python3.10/site-packages/ngsolve/nonlinearsolvers.py in Newton(a,␣
 ↪u, freedofs, maxit, maxerr, inverse, dirichletvalues, dampfactor, printing,␣
 ↪callback)
    134     if dirichletvalues is not None:
    135         solver.SetDirichlet(dirichletvalues)
--> 136     return solver.Solve(maxit=maxit, maxerr=maxerr,

    137                         dampfactor=dampfactor,
    138                         printing=printing,

~/.local/lib/python3.10/site-packages/ngsolve/utils.py in retfunc(*args,␣
 ↪**kwargs)
    152     def retfunc(*args,**kwargs):
    153         with timer:
--> 154             ret = func(*args, **kwargs)
    155         return ret
    156     return retfunc

~/.local/lib/python3.10/site-packages/ngsolve/nonlinearsolvers.py in Solve(self,␣
 ↪maxit, maxerr, dampfactor, printing, callback, linesearch, printenergy,␣
 ↪print_wrong_direction)
     35             a.Apply(u.vec, r)
     36
---> 37             self._UpdateInverse()
     38             if self.rhs is not None:
     39                 r.data -= self.rhs.vec

~/.local/lib/python3.10/site-packages/ngsolve/nonlinearsolvers.py in␣
 ↪_UpdateInverse(self)
     90             self.inv.Update()
```

```
     91         else:
---> 92             self.inv = self.a.mat.Inverse(self.freedofs,
     93                                             inverse=self.inverse)
     94

NgException: SparseMatrix::InverseMatrix:  UmfpackInverse not available
```

1.2 A trivial problem:

$$5u^2 = 1, \qquad u \in \mathbb{R}.$$

[ ]:
```
V = NumberSpace(mesh)
u,v = V.TnT()
a = BilinearForm(V)
a += ( 5*u*u*v - 1 * v)*dx
gfu = GridFunction(V)
gfu.vec[:] = 1
SimpleNewtonSolve(gfu,a)

print("\nscalar solution", gfu.vec[0], "(exact: ", sqrt(0.2), ")")
```

1.3 Another example: Stationary Navier-Stokes:

Find $\mathbf{u} \in \mathbf{V}$, $p \in Q$, $\lambda \in \mathbb{R}$ so that

$$\int_\Omega \nu \nabla \mathbf{u} : \nabla \mathbf{v} + (\mathbf{u} \cdot \nabla)\mathbf{u} \cdot \mathbf{v} - \int_\Omega \operatorname{div}(\mathbf{v}) p \qquad = \int \mathbf{f} \cdot \mathbf{v} \qquad \forall \mathbf{v} \in \mathbf{V}, \tag{1}$$

$$- \int_\Omega \operatorname{div}(\mathbf{u}) q \qquad + \int_\Omega \lambda q = 0 \qquad \forall q \in Q, \tag{2}$$

$$\int_\Omega \mu p \qquad = 0 \qquad \forall \mu \in \mathbb{R}. \tag{3}$$

[ ]:
```
mesh = Mesh (unit_square.GenerateMesh(maxh=0.05)); nu = Parameter(1)
V = VectorH1(mesh,order=3,dirichlet="bottom|right|top|left")
Q = H1(mesh,order=2);
N = NumberSpace(mesh);
X = V*Q*N
(u,p,lam), (v,q,mu) = X.TnT()
a = BilinearForm(X)
a += (nu*InnerProduct(grad(u),grad(v))+InnerProduct(grad(u)*u,v)
      -div(u)*q-div(v)*p-lam*q-mu*p)*dx
```

[ ]:
```
gfu = GridFunction(X)
gfu.components[0].Set(CoefficientFunction((4*x*(1-x),0)),
                      definedon=mesh.Boundaries("top"))
```

[ ]:
```python
SimpleNewtonSolve(gfu,a)
scenep = Draw(gfu.components[1],mesh,"p")
sceneu = Draw(gfu.components[0],mesh,"u")
```

[ ]:
```python
nu.Set(0.01)
SimpleNewtonSolve(gfu,a)
sceneu.Redraw()
scenep.Redraw()
```

[ ]:
```python
nu.Set(0.001)
SimpleNewtonSolve(gfu,a)
sceneu.Redraw()
scenep.Redraw()
```

[ ]:
```python
nu.Set(0.001)
gfu.components[0].Set(CoefficientFunction((4*x*(1-x),0)),definedon=mesh.
    ↪Boundaries("top"))
Newton(a,gfu,maxit=20,dampfactor=0.1)
sceneu.Redraw()
scenep.Redraw()
```

[ ]:
```python

```