FEniCS: Non-Linear Poisson Problem with Explicit Newton Method

Ida Ang

1 Problem Definition

Nonlinear Poisson's equation for domain Ω and boundary $\partial\Omega = \Gamma_D \cup \Gamma_N$

$$-\nabla \cdot (q(u)\nabla u) = f(x,y) \quad \text{in } \Omega \tag{1}$$

$$q(u) = (1+u)^m \tag{2}$$

$$u = 0 \text{ at } x = 0 \quad \text{on } \Gamma_D \tag{3}$$

$$u = 1 \text{ at } x = 1 \quad \text{on } \Gamma_D \tag{4}$$

$$\nabla u \cdot n = \frac{\partial u}{\partial n} = 0 \rightarrow \frac{\partial u_j}{\partial x_i} n_i \mathbf{e_j} = g_j \mathbf{e_j} = 0 \quad \text{on } \Gamma_N \tag{5}$$

Note the definition of 2 in this non-linear problem is different from the online version. In the online version $q(u) = 1 + u^2$

2 Variational Weak Form

Write Eq. 1 in indicial notation where the gradient of a vector field is a 2nd order tensor:

$$-\nabla \cdot (q(u)\nabla u) = f \rightarrow -\nabla \cdot \big[q\frac{\partial u_j}{\partial x_i}\big(\mathbf{e_j} \otimes \mathbf{e_i}\big)\big] = f_j \mathbf{e_j}$$

Recall that divergence of any tensor A looks like:

$$\nabla \cdot \mathbf{A} = \frac{\partial A_{ik}}{\partial x_j}\big(\mathbf{e_i} \otimes \mathbf{e_k}\big)\mathbf{e_j} = \frac{\partial A_{ik}}{\partial x_j}\mathbf{e_i}\delta_{kj} = \frac{\partial A_{ij}}{\partial x_j}\mathbf{e_i} \tag{6}$$

Using this definition (Eq. 6):

$$-q\big[\nabla \cdot \frac{\partial u_j}{\partial x_i}\big(\mathbf{e_j} \otimes \mathbf{e_i}\big)\big] = -q\frac{\partial^2 u_j}{\partial x_i \partial x_k}\big(\mathbf{e_j} \otimes \mathbf{e_i}\big)\mathbf{e_k} = -q\frac{\partial^2 u_j}{\partial x_i \partial x_k}\mathbf{e_j}\delta_{ik} = -q\frac{\partial^2 u_j}{\partial x_k^2}\mathbf{e_j}$$

Therefore:

$$-q\frac{\partial^2 u_j}{\partial x_k^2}\mathbf{e_j} = f_j \mathbf{e_j}$$

Multiply by test function, v:

$$-q\frac{\partial^2 u_j}{\partial x_k^2}\mathbf{e_j} \cdot v_p \mathbf{e_p} = f_j \mathbf{e_j} \cdot v_p \mathbf{e_p} \rightarrow -q\frac{\partial^2 u_j}{\partial x_k^2}\mathbf{e_j} v_j = f_j v_j$$

Integrate over domain, Ω:

$$-q\int_\Omega \frac{\partial^2 u_j}{\partial x_k^2}\mathbf{e_j}v_j dx = \int_\Omega f_j v_j dx \tag{7}$$

Use Integration by Parts:

$$(fg)' = f'g + fg' \rightarrow f'g = (fg)' - fg' \rightarrow f''g = (f'g)' - f'g' \tag{8}$$

Where we can substitute $f' = \frac{\partial x_j}{\partial x_k}$ and $g = v$ into Eq. 8

$$\frac{\partial^2 u_j}{\partial x_k^2}\mathbf{e_j}v_j = \left(\frac{\partial u_j}{\partial x_k}v_j\right)_{,k} - \frac{\partial u_j}{\partial x_k}\frac{\partial v_j}{\partial x_k}$$

Change signs and substitute into Eq. 7

$$-q\int_\Omega \left(\frac{\partial u_j}{\partial x_k}v_j\right)_{,k} dx + q\int_\Omega \frac{\partial u_j}{\partial x_k}\frac{\partial v_j}{\partial x_k}dx = \int_\Omega f_j v_j dx \tag{9}$$

Use divergence theorem on first term of Eq. 9

$$-q\int_\Omega \frac{\partial u_j}{\partial x_k}n_k v_j dx + q\int_\Omega \frac{\partial u_j}{\partial x_k}\frac{\partial v_j}{\partial x_k}dx = \int_\Omega f_j v_j dx$$

Recognize the Neumann boundary condition Eq. 5:

$$-q\int_\Omega g_j v_j dx + q\int_\Omega \frac{\partial u_j}{\partial x_k}\frac{\partial v_j}{\partial x_k}dx = \int_\Omega f_j v_j dx$$

According to Eq. 5, $g = 0$

$$q\int_\Omega \frac{\partial u_j}{\partial x_k}\frac{\partial v_j}{\partial x_k}dx = \int_\Omega f_j v_j dx$$

Writing this form in direct notation, we have the following variational (weak) form:

$$\int_\Omega q(u)\nabla u \cdot \nabla v dx = \int_\Omega f v dx \tag{10}$$

3 Newton's Method

3.1 General Formulation

Newton's method for the system $F_i(U_1, ...U_N) = 0$ can be formulated as:

$$\sum_{j=1}^{N}\frac{\partial}{\partial U_j}F_i(U_1^k, ...U_N^k)\delta U_j = -F_i(U_1^k, ...U_N^k) \quad \text{i = 1,...,N} \tag{11}$$

$$U_j^{k+1} = U_j^k + \omega\delta U_j \quad \text{i = 1,...N} \tag{12}$$

where $\omega \in [0, 1]$ is a relaxation parameter and k is an iteration index. We are solving for the unknown parameters, $U_1, ...U_N$, the coefficients of the linear combination in the finite element solution.
We can rewrite Eq. 11

$$\sum_{j=1}^{N}\frac{\partial F_i}{\partial U_j}\delta U_j = -F_i \quad \text{i = 1,...,N} \tag{13}$$

3.2 Discretization

We must compute the Jacobian matrix $\frac{\partial F_i}{\partial U_j}$ and the right hand side vector $-F_i$
Rewrite the weak form Eq. 10

$$F = \int_\Omega q(u)\nabla u \cdot \nabla v dx \tag{14}$$

Obtain the Jacobian using the product rule:

$$\frac{\partial F}{\partial U_j} = \int_\Omega \left[\frac{\partial q(u)}{\partial U_j}\nabla u + q(u)\frac{\partial \nabla u}{\partial U_j}\right] \cdot \nabla v dx$$

Where

$$\frac{\partial q(u)}{\partial U_j} = q'(u)\phi_j \qquad \frac{\partial \nabla u}{\partial U_j} = \nabla \phi_j \tag{15}$$

Using the identities in Eq. 15

$$\frac{\partial F}{\partial U_j} = \int_\Omega \left[q'(u)\phi_j \nabla u + q(u)\nabla \phi_j\right] \cdot \nabla v dx$$

We can discretize our solution by introducing the following notations for u and v:

$$u = \sum_{j=1}^{N} U_j \phi_j \tag{16}$$

and

$$v = \hat{\phi}_i \tag{17}$$

Substitute Eq. 16 and 17 into the Jacobian. Note that this problem is discretized by k, the iteration index

$$\frac{\partial F_i}{\partial U_j} = \int_\Omega \left[q'\Big(\sum_{l=1}^{N} U_l^k \phi_l\Big)\phi_j \nabla\Big(\sum_{j=1}^{N} U_j^k \phi_j\Big) + q\Big(\sum_{l=1}^{N} U_l^k \phi_l\Big)\nabla \phi_j\right] \cdot \nabla \hat{\phi}_i dx$$

Reformulate the Jacobian matrix by introducing the short notation $u^k = \sum_{j=1}^{N} U_j^k \phi_j$ and using Eq. 17

$$\frac{\partial F_i}{\partial U_j} = \int_\Omega \left[q'(u^k)\phi_j \nabla u^k + q(u^k)\nabla \phi_j\right] \cdot \nabla v dx$$

Lastly, we want to achieve the general formulation form on the LHS of the equation (Eq. 13. This requires multiplication of the Jacobian by a function $\sum_{j=1}^{N} \delta U_j$

$$\frac{\partial F_i}{\partial U_j} = \int_\Omega \left[q'(u^k)\sum_{j=1}^{N}\Big(\delta U_j \phi_j\Big)\nabla u^k + q(u^k)\nabla \sum_{j=1}^{N}\Big(\delta U_j \phi_j\Big)\right] \cdot \nabla v dx$$

Call the unknown $\delta u = \sum_{j=1}^{N} \delta U_j \phi_j$, simplifying the equation:

$$\sum_{j=1}^{N} \frac{\partial F_i}{\partial U_j}\delta U_j = \int_\Omega \left[q'(u^k)\delta u \nabla u^k + q(u^k)\nabla \delta u\right] \cdot \nabla v dx$$

From the linear system, we can go backwards to construct the corresponding discrete weak form. Recall it looks like $\int_\Omega q(u)\nabla u \cdot \nabla v dx$

$$\int_\Omega \Big[q'(u^k)\delta u \nabla u^k + q(u^k)\nabla \delta u\Big] \cdot \nabla v dx = -\int_\Omega q(u^k)\nabla u^k \cdot \nabla v dx \tag{18}$$

Now that the system is linearized, we can use the bilinear and linear forms to represent this equation:

$$a(\delta u, v) = L(v) \tag{19}$$

Therefore we have our bilinear and linear form:

$$a(\delta u, v) = \int_\Omega \Big[q'(u^k)\delta u \nabla u^k + q(u^k)\nabla \delta u\Big] \cdot \nabla v dx \tag{20}$$

$$L(v) = -\int_\Omega q(u^k)\nabla u^k \cdot \nabla v dx \tag{21}$$

An important feature of Newton's method is that the previous solution u^k replaces u in the formulas when computing the matrix $\frac{\partial F_i}{\partial U_j}$ and vector F_i for the linear system in each Newton iteration.

4 FEniCS Implementation

4.1 Mesh and Function Space

The user controls the degree of the function space and the spacing of the mesh with `sys.argv`

```
degree = int(sys.argv[1])
divisions = [int(arg) for arg in sys.argv[2:]]
```

Note that `sys.argv[1]` takes the first argument and `sys.argv[2:]` takes any integer argument in the following spaces. In command line enter `python3 name.py degree divisions` with degree and divisions being some integer value.

The input value for divisions changes the domain type, where three different domain types are listed. We use standard Lagrange elements which can also be called with `'CG'` for Continuous Galerkin. If the degree is 1, we get continuous piecewise linear polynomials.

```
d = len(divisions)
domain_type = [UnitIntervalMesh, UnitSquareMesh, UnitCubeMesh]
mesh = domain_type[d-1](*divisions)
V = FunctionSpace(mesh, 'Lagrange', degree)
```

4.2 Boundary Conditions

The Dirichlet boundary conditions are set according to 3, the left boundary, and 4, the right boundary.

```
tol = 1E-14
def left_boundary(x, on_boundary):
```

```
   return on_boundary and abs(x[0]) < tol
def right_boundary(x, on_boundary):
   return on_boundary and abs(x[0]-1) < tol
```

We provide an initial set of boundary conditions for the first guess:

```
Gamma_0 = DirichletBC(V, Constant(0.0), left_boundary)
Gamma_1 = DirichletBC(V, Constant(1.0), right_boundary)
bcs = [Gamma_0, Gamma_1]
```

Later on, we need to define the Dirichlet conditions for the later guesses for the Newton's Iterations. These boundary conditions are specifically for δu which must be zero. Recall, Eq.12

$$u^{k+1} = u^k + \omega \delta u$$

In order for u^k to fulfill the Dirichlet conditions for u, δu must be 0

```
Gamma_0_du = DirichletBC(V, Constant(0.0), left_boundary)
Gamma_1_du = DirichletBC(V, Constant(0.0), right_boundary)
bcs_du = [Gamma_0_du, Gamma_1_du]
```

Note that Neumann boundary conditions are within the weak form defined in Eq. 10.

4.3 Definition of the Variational Problem

Recall our system is linearized, so we can directly declare the trial and test functions. If we omit the trial function we are telling FEniCS that the problem is non-linear.

```
u = TrialFunction(V)
v = TestFunction(V)
```

Define variational problem for initial guess. To obtain a good initial guess u^0, we can solve a simplified linear problem with q(u) = 1 and f = 0. Note that if q(u) = 1 then m = 0

$$a(u, v) = L(v) = 0 \rightarrow \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v dx = 0$$

```
a = inner(nabla_grad(u), nabla_grad(v))*dx
f = Constant(0.0)
L = f*v*dx
```

Solve using `assemble_system` command with the boundary conditions set for the initial guess.

```
A, b = assemble_system(a, L, bcs)
```

u_k denotes the solution function for the previous iteration where $u_k = u_o$ is the initial guess. The type of solver is specified by, `lu`, which stands for lower-upper decomposition.

```
u_k = Function(V)
solve(A, u_k.vector(), b, 'lu')
```

For later iterations, we define the function q(u), choosing the non-linear coefficient, m = 2. Take the derivative of q in Eq. 2 to use in the discretized weak form.

```
m = 2
```

```
def q(u):
   return (1+u)**m
def Dq(u):
   return m*(1+u)**(m-1)
```

4.4 Discretized Weak Form Definition

Define du = δu
```
du = TrialFunction(V)
```

Write the discretized weak form, Eq. 18 in C++ syntax

$$a(\delta u, v) = \int_\Omega \left[q'(u^k)\delta u \nabla u^k + q(u^k)\nabla \delta u \right] \cdot \nabla v dx$$

```
a = inner(Dq(u_k)*du*nabla_grad(u_k), nabla_grad(v))*dx +
   inner(q(u_k)*nabla_grad(du), nabla_grad(v))*dx
```

$$L(v) = -\int_\Omega q(u^k)\nabla u^k \cdot \nabla v dx$$

```
L = -inner(q(u_k)*nabla_grad(u_k), nabla_grad(v))*dx
```

Define functions:
```
du = Function(V)
u = Function(V)
```

4.5 Newton Iteration Loop

Use tol = tolerance, eps = error measure, iter = iterations, and maxiter = max iteration number.
```
omega = 1.0
eps = 1.0
tol = 1.0E-5
iter = 0
maxiter = 25
```

The error measure has to be larger than the tolerance value and the number of iterations can't exceed the max iteration number. Update the iteration count every time the loop is accessed:
```
while eps > tol and iter < maxiter:
   iter += 1
```

Eq. 19 can be solved using the `assemble_system` command with the BCs for δu.
```
   A, b = assemble_system(a, L, bcs_du)
```

The type of solver is not specified in this loop
```
   solve(A, du.vector(), b)
```

The error measure can be defined as the norm of the initial solution and current solution: $|u - u^k|$.
```
   eps = numpy.linalg.norm(du.vector().array(), ord=numpy.Inf)
```

There are multiple ways to write Eq. 12: $u^{k+1} = u^k + \omega\delta u$. I have included two ways below:

```
u.vector()[:]  = u_k.vector() + omega*du.vector()
u.vector()[:]  += omega*du.vector()
```

Finally assign u (u^{k+1}) to u^k. Note that we have dropped the k+1 superscript in the code:

```
u_k.assign(u)
```