

Laboratory 05

Finite Element method for non linear equations

Exercise 1.

Let $\Omega = (0, 1)^3$ be the unit cube and let us consider the following non linear problem:

$$\begin{cases} -\nabla \cdot ((\mu_0 + \mu_1 u^2) \nabla u) = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases} \quad \begin{array}{l} (1a) \\ (1b) \end{array}$$

where $\mathbf{x} = (x, y, z)^T$, $\mu_0 = 1$, $\mu_1 = 10$ and $f(\mathbf{x}) = 1$.

1.1. Write the weak formulation of problem (1), expressing it in the residual form $R(u, v) = 0$.

Solution. Let $V = H_0^1(\Omega)$ and $v \in V$. Following the usual procedure (multiply v to (1a), then integrate by parts), we obtain

$$\underbrace{\int_{\Omega} (\mu_0 + \mu_1 u^2) \nabla u \cdot \nabla v d\mathbf{x}}_{b(u, v)} = \underbrace{\int_{\Omega} f v d\mathbf{x}}_{F(v)} .$$

By defining the *residual*

$$R(u, v) = b(u, v) - F(v) ,$$

we can write the weak formulation as

$$\text{find } u \in V \text{ such that } R(u, v) = 0 \text{ for all } v \in V .$$

Notice that $b(u, v)$, and thus $R(u, v)$, are non linear with respect to u . Therefore, after discretization, we cannot write it as a linear system, but we need to introduce an algorithm for its linearization.

1.2. Compute the Fréchet derivative $a(u)(\delta, v)$ of the residual $R(u)(v)$, then write Newton's method for the solution of problem (1).

Solution. The Fréchet derivative is a generalization of the concept of derivative that also works for functionals and bilinear forms. For a given functional $G : V \rightarrow \mathbb{R}$, the Fréchet derivative at a point $u \in V$ is a linear operator $A(u) = \frac{dG}{du} : V \rightarrow \mathbb{R}$ such that

$$\lim_{\|\delta\| \rightarrow 0} \frac{|G(u + \delta) - G(u) - A(u)(\delta)|}{\|\delta\|} = 0 .$$

In other words, A is the linear operator that best approximates the functional G near the point u .

Considering the residual $R(u, v)$, seen as a function of only u , we have

$$\begin{aligned} R(u + \delta, v) &= b(u + \delta, v) - F(v) \\ &= \int_{\Omega} (\mu_0 + \mu_1(u + \delta)^2) \nabla(u + \delta) \cdot \nabla v d\mathbf{x} - \int_{\Omega} f v d\mathbf{x} \\ &= \int_{\Omega} \left[(\mu_0 + \mu_1 u^2) \nabla u + (\mu_1 \delta^2) \nabla u + (2\mu_1 u \delta) \nabla u + \right. \\ &\quad \left. (\mu_0 + \mu_1 u^2) \nabla \delta + (\mu_1 \delta^2) \nabla \delta + (2\mu_1 u \delta) \nabla \delta \right] \cdot \nabla v d\mathbf{x} - \int_{\Omega} f v d\mathbf{x}. \end{aligned}$$

This leads to

$$R(u + \delta, v) - R(u, v) = \underbrace{\int_{\Omega} (2\mu_1 u \delta) \nabla u \cdot \nabla v d\mathbf{x} + \int_{\Omega} (\mu_0 + \mu_1 u^2) \nabla \delta \cdot \nabla v d\mathbf{x}}_{\frac{dR}{du}(\delta, v) = a(u)(\delta, v)} + o(\|\delta\|^2).$$

Notice that the bilinear form a is linear with respect to both δ and v . The Newton's method for this problem reads: given an initial guess $u^{(0)}$, iterate for $k = 0, 1, 2, \dots$ until convergence:

1. compute $\delta^{(k)}$ by solving the linear problem: $a(u^{(k)})(\delta^{(k)}, v) = -R(u^{(k)}, v)$ for all $v \in V$;
2. set $u^{(k+1)} = u^{(k)} + \delta^{(k)}$.

The problem at step 1 is a linear differential problem in weak form, and thus we can solve it using finite elements.

Upon finite element discretization, the bilinear form $a(u)(\delta, v)$ gives rise to the following matrix:

$$(A(u))_{ij} = a(u)(\varphi_j, \varphi_i) = \int_{\Omega} (2\mu_1 u \varphi_j) \nabla u \cdot \nabla \varphi_i d\mathbf{x} \int_{\Omega} (\mu_0 + \mu_1 u^2) \nabla \varphi_j \cdot \nabla \varphi_i d\mathbf{x},$$

whereas the residual yields the vector

$$(\mathbf{r}(u))_i = R(u, \varphi_i) = \int_{\Omega} (\mu_0 + \mu_1 u^2) \nabla u \cdot \nabla \varphi_i d\mathbf{x} - \int_{\Omega} f \varphi_i d\mathbf{x}.$$

We can stop the iterations when either $\|\delta^{(k)}\|$ or $\|R(u^{(k)})\|$ become smaller than a given tolerance (stopping based on the increment between iterations or on the residual, respectively).

Beware that in general Newton's method has only local convergence, that is it converges to the solution only if the initial guess is close enough to it. Additionally, there is no

general way to know in advance how close to the solution the initial guess is (especially since the solution is unknown). For these reason, together with the fact that Newton's method requires solving multiple linear systems, non-linear problems tend to be significantly more complex to solve than linear problems.

1.3. Using Newton's method, implement a solver for problem (1). Then, solve the problem on the mesh `mesh/mesh-cube-20.msh`, with polynomial degree $r = 1$, and using a tolerance of 10^{-6} on the norm of the residual for the Newton's method.

Solution. See file `src/exercise-01.cpp` for the implementation. The solution is reported in Figure 1.

When assembling the system matrix and the residual for step 1 of Newton's method, we need to evaluate the solution $u^{(k)}$ on quadrature nodes. This can be done by exploiting the FE representation, i.e. for a given quadrature node \mathbf{x}_q on element K ,

$$u^{(k)}(\mathbf{x}_q) = \sum_{i \in I_K} U_i^{(k)} \varphi_i(\mathbf{x}_q),$$

where I_K is the set of basis function indices that have support over the element K . We could do the above computation manually, but since it is so common `deal.II` offers an easy way to do that through `FEValues::get_function_values`. Similarly, we need to evaluate $\nabla u^{(k)}$, and we can do that through `FEValues::get_function_gradients`.

Boundary conditions must be treated with care when solving a non-linear problem through Newton's method. Neumann boundary conditions are included in the weak formulation of the problem, and thus are inherited by the linearized problem in step 1 of Newton's method. Dirichlet boundary conditions must be imposed essentially. We want that, upon convergence, $u = g$ on some boundary Γ_D . One way to achieve that is imposing that $u^{(k)} = g$ on Γ_D for all iterations k . This is equivalent to setting $u^{(0)} = g$ on Γ_D and $\delta^{(k)} = 0$ on Γ_D for all iterations k . In practical terms, this means that:

1. we impose the (possibly non-homogeneous) Dirichlet boundary conditions of the original problem on the initial guess $u^{(0)}$ of Newton's method;
2. we impose homogeneous Dirichlet boundary conditions on all the increments $\delta^{(k)}$.

Newton's method is not guaranteed to converge. Therefore, if we just check that the residual (or the increment between iterations) falls below a given tolerance, we might end up with an infinite loop, which is obviously not good. To avoid this situation, we choose a maximum number of iterations (say 1000), and if that number is reached we assume that Newton is not converging and stop the execution. This is similar to what is done in most iterative algorithms (e.g. iterative linear solvers).

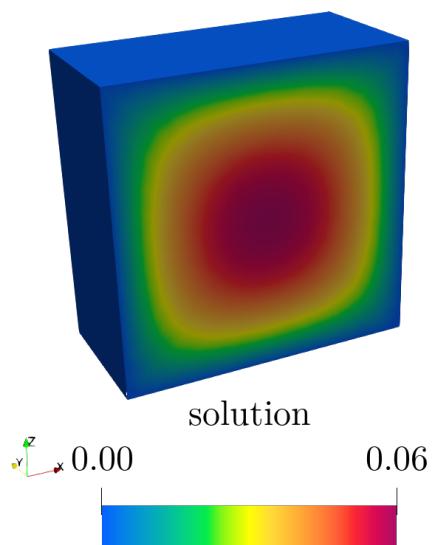


Figure 1: Solution to Exercise 1. The domain was clipped along the plane $y = 0.5$.

Possibilities for extension

Automatic differentiation. Computing the Fréchet derivative might be difficult and error prone. There exist tools based on *automatic differentiation* (AD) that allow to only implement the computation of the residual (where no derivative is required), and then obtain the matrix through AD. `deal.II` offers wrappers to AD libraries from Trilinos that can be used to this purpose. Based on the `deal.II` tutorials, modify the code for Exercise 1 to use AD and compare the computational costs of the two strategies.