MAP562 Optimal design of structures

by Grégoire Allaire, Beniamin Bogosel

Ecole Polytechnique

Academic year 2018-2019

Gradient descent and Newton's method for minimization in a PDE / functional space context

In this document we summarize the steps to implement gradient descent (with a fixed step size) and Newton's method in order to minimize a nonlinear functional (here the p-Laplace problem) in a functional space setting. This setting is a so-called unconstrained minimization problem.

Correction of Exercise 3 from Sheet 2

Let $\Omega \in \mathbb{R}^2$ be a domain and Γ_D and Γ_N be the usual Dirichlet and Neumann boundary parts. The functional space we work with is $V := H^1(\Omega)$. Then the minimization problem reads:

$$\min_{u\in V} J(u),$$

where

$$J(u) = \int_\Omega \Big(\frac{|\nabla u|^2}{2} + \frac{|\nabla u|^p}{p}\Big)\,dx - \int_\Omega f u\,dx - \int_{\Gamma_N} g u\,ds,$$

and $f = 1$ and $g = -1$. Moreover, we choose $p = 4$. In this context, we must ensure that $p > 2$ (the more general problem $p < 2$ is possible, but difficult from a theoretical and numerical point of view).

In the following, we formulate the two algorithms (using the same notation as in the lecture notes, chapter 3). This notation is also (as far as possible) used in the `FreeFem++` codes (available on the course Moodle webpage).

1 Gradient descent with fixed step

In this section, we discuss a gradient descent method with fixed step $\mu \in \mathbb{R}^+$ to solve the minimization problem.

Algorithm 1. *(Gradient descent with fixed step size, chapter 3, slide 34)*

- Choose an initial guess $u^0 \in V$, e.g., $u^0 = 0$.
- For $n = 0, 1, 2, \ldots$ compute
$$u^{n+1} = u^n - \mu J'(u^n).$$
- Stop the iteration process when
$$\sqrt{\int_\Omega (J'(u^n))^2} < TOL$$
where, *e.g.*, $TOL = 1e-4$.

In a functional space context the crucial aspect is the evaluation of $J'(u^n)$.

A standard procedure to realize the above algorithm is to compute the projection $\delta u \in V$. In more detail we replace $J'(u^n)$ by δu so that

$$u^{n+1} = u^n - \mu\delta u$$

where $\delta u \in V$ is a finite element solution of the identification problem:

Identification problem For a given u^n, find $\delta u \in V$ such that

$$\int_\Omega \nabla\delta u \cdot \nabla\phi = \langle J'(u^n), \phi\rangle \quad \forall\phi \in V,$$

where we use a scalar product that is related to the functional space V (here the H^1 space). Specifically the right hand side consists of known terms: u^n, f, g.

We could also have used

$$\int_\Omega \delta u\phi + \alpha^2\nabla\delta u \cdot \nabla\phi = \langle J'(u^n), \phi\rangle,$$

with some parameter $\alpha > 0$.

Specifically, the right hand side term of the projection problem is nothing else than calculating the directional derivative of $J(u)$ into direction $\phi \in V$ (neglecting the index n for the moment):

$$\begin{aligned}\langle J'(u), \phi\rangle = J'(u)(\phi) &= \int_\Omega \nabla u \cdot \nabla\phi + \int_\Omega |\nabla u|^{p-2} \cdot \nabla u \cdot \nabla \phi - \int_\Omega f \cdot \phi - \int_{\Gamma_N} g \cdot \phi \, ds, \\ &= \int_\Omega (1 + |\nabla u|^{p-2}) \cdot \nabla u \cdot \nabla\phi - \int_\Omega f \cdot \phi - \int_{\Gamma_N} g \cdot \phi \, ds.\end{aligned}$$

Implementation For a given u (in the gradient-based context, u will be the previous iteration u^n), find $\delta u \in V$ such that:

$$J'(u)(\phi) = \int_\Omega \nabla\delta u \nabla\phi \quad \forall \phi \in V.$$

To solve this (linear) problem, we introduce finite dimensional subspaces V_h and use the finite element method and then use for example an *LU* decomposition or an iterative solver (CG, GMRES) for linear problems. The obtained solution δu is then our current iterate used in the gradient descent method.

In `FreeFem++` the previous setting is defined as

```
problem Jlin(du,phi) = -int2d(Th)(grad(du)'*grad(phi))
       +int2d(Th)((1+sqrt(grad(u)'*grad(u))^(p-2))*(grad(u)'*grad(phi)))
       -int2d(Th)(f*phi)
       -int1d(Th,GammaN)(g*phi)
       +on(GammaD,du=0);
```

To solve this problem we simply write

```
Jlin;
```

and can use the solution `du` to update the iteration:

```
u = u - stepsize * du;
```

Remark 1. In the above algorithm we have to solve at each iteration n such a linear problem. In `FreeFem++` we use therefore the keyword `problem` as already done for linear problems in the first exercise.

Remark 2. In general we **hope** that the sequence $(u^n)_{n\in\mathbb{N}}$ converges towards the optimal solution. Under several assumptions this convergence can be justified also from the theoretical point of view. In most cases (and this is standard in practice!) a theoretical convergence cannot be proven because it is too difficult, however the algorithm is nevertheless used in practice and yields satisfying results.

2 Newton's method

In this section we turn our attention to Newton's method. The crucial difference is that Newton's method converges (locally) much faster than gradient descent, however we need

- to calculate the second derivative of $J(u)$;
- to solve a linear equation system, which can be an expensive operation.

Algorithm 2 (Newton's method, chapter 3, page 37). Choose an initial guess $u^0 \in V$, e.g., $u^0 = 0$. For $n = 0, 1, 2, \ldots$:

$$u^{n+1} = u^n + \delta u$$

where $\delta u \in V$ is obtained from solving the linear system:

$$\text{Find } \delta u \in V: \quad J''(u^n)(\delta u, \phi) = -J'(u^n)(\phi) \quad \forall \phi \in V$$

using, for example, a finite element method in a finite dimensional subspace V_h. We stop the iteration when

$$\sqrt{\int_\Omega (\delta u)^2} < TOL$$

where, for example, $TOL = 1e-4$.

To compute the second-order derivative we start with $J'(u)(\phi)$,

$$\begin{aligned}
J'(u)(\phi) &= \int_\Omega \nabla u \cdot \nabla \phi + \int_\Omega |\nabla u|^{p-2} \nabla u \cdot \nabla \phi - \int_\Omega f\phi - \int_{\Gamma_N} g\phi\, ds, \\
&= \int_\Omega (1 + |\nabla u|^{p-2}) \nabla u \cdot \nabla \phi - \int_\Omega f\phi - \int_{\Gamma_N} g\phi\, ds
\end{aligned}$$

and obtain by differentiating u now into direction δu using multiple times the chain rule and also the product rule:

$$\begin{aligned}
J''(u)(\delta u, \phi) &= \int_\Omega \nabla \delta u \cdot \nabla \phi + \int_\Omega (p-2)|\nabla u|^{p-4} \nabla u \cdot \nabla \delta u \nabla u \cdot \nabla \phi + \int_\Omega |\nabla u|^{p-2} \cdot \nabla u \cdot \nabla \phi \\
&= \int_\Omega (1 + |\nabla u|^{p-2}) \cdot \nabla \delta u \cdot \nabla \phi + \int_\Omega (p-2)|\nabla u|^{p-4} \nabla u \cdot \nabla \delta u \nabla u \cdot \nabla \phi.
\end{aligned}$$

Having now the Hessian matrix $J''(u)(\delta u, \phi)$ and the first order derivative $J'(u)(\phi)$ we can solve the linear problem:

For a given u^n (previous solution), find $\delta u \in V$ such that

$$J''(u^n)(\delta u, \phi) + J'(u^n)(\phi) = 0 \quad \forall \phi \in V.$$

In `FreeFem++` we use again the `problem` functionality to solve this problem.

In `FreeFem++` the previous setting is defined as

```
problem Jlin(du,phi)=int2d(Th)((1+sqrt(grad(u)'*grad(u))^(p-2))*(grad(du)'*grad(phi))
    +(p-2)*sqrt(grad(u)'*grad(u))^(p-4)*(grad(u)'*grad(du))*(grad(u)'*grad(phi)))
    +int2d(Th)((1+sqrt(grad(u)'*grad(u))^(p-2))*(grad(u)'*grad(phi)))
    -int2d(Th)(f*phi)
    -int1d(Th,GammaN)(g*phi)
    +on(GammaD,du=0);
```

To solve this problem we simply write

```
Jlin;
```

and can use the solution `du` to update the iteration:

```
u = u + du;
```

Remark 3. A potential difficulty is that Newton's method is quite sensitive if the initial guess u^0 is too bad. Common strategies to globalize the method is to introduce a line search parameter or to work with so-called trust region methods (see the remark below).

Remark 4 (Line search). A simple method to increase the convergence radius of Newton's method (mainly for nonconvex problems) is to introduce a line search parameter $\mu \in [0, 1]$ such that:

$$u^{n+1} = u^n + \mu \delta u$$

For $\mu = 1$ we make a full Newton step and choosing $\mu < 1$ yields a damped Newton method which has an increased convergence radius (but which again does not work in all possible situations well), and also has a reduced order of convergence (not quadratic anymore). In `FreeFem++` this would read:

```
u = u + stepsize * du;
```