Technische Universität München
Center for Mathematical Sciences, M17

Dr. Anne-Céline Boulanger, Konstantin Pieper
`www-m17.ma.tum.de/Lehrstuhl/LehreSoSe14LabCourseOptPDE`

Summer 2014

Lab course: Optimization with elliptic PDEs: Sheet 3

Exercise 3.1 (The semi-linear elliptic problem with Hessian): We consider again the test problem from Ex. 2.1:

$$\min_{q\in L^2(\Omega),u\in H^1(\Omega)} J(q,u) = \frac{1}{2}\|u-u_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|q\|^2_{L^2(\Omega)} \tag{1a}$$

$$\text{subject to } \begin{cases} -\Delta u + u + \sigma u^3 = q & \text{in } \Omega, \\ \partial_n u = 0 & \text{on } \partial\Omega. \end{cases} \tag{1b}$$

To improve the performance of the optimization method, we now take into account second order information.

1. For fixed q and u/z the corresponding discrete state/adjoint solution, the second derivative can be computed (in the general case) as

$$\begin{aligned} j''(q)(\tau q,\delta q) &= \mathcal{L}''_{qq}(q,u,z)(\tau q,\delta q) + \mathcal{L}''_{qu}(q,u,z)(\tau q,\delta u) + \mathcal{L}''_{qz}(q,u,z)(\tau q,\delta z) = \\ &J''_{qq}(q,u)(\delta q,\tau q)+J''_{qu}(q,u)(\delta u,\tau q)-a''_{qq}(q,u)(\delta q,\tau q,z)-a''_{qu}(q,u)(\delta u,\tau q,z)-a'_q(q,u)(\tau q,\delta z) \end{aligned} \tag{2}$$

   where δu is a solution of the *tangent* equation

$$a'_u(q,u)(\delta u,\varphi) = -a'_q(q,u)(\delta q,\varphi) \quad \text{for all } \varphi\in V \tag{3}$$

   and δz is a solution of the *dual-for-hessian/linearized adjoint* equation

$$\begin{aligned} a'_u(q,u)(\varphi,\delta z) = J''_{uu}(q,u)(\varphi,\delta u) - a''_{uu}(q,u)(\varphi,\delta u,z) \\ J''_{uq}(q,u)(\varphi,\delta q) - a''_{uq}(q,u)(\varphi,\delta q,z) \quad \text{for all } \varphi\in V \end{aligned} \tag{4}$$

   Adapt this to the concrete example and to the discrete setting. Also give the matrix vector notation for equations (3) and (4).

2. Implement a function `du = solveTangent(q, u, dq, args)` that solves the discrete tangent equation for a given direction δq_h.

3. Implement a function `dz = solveDFH(q, u, z, dq, du, args)` that solves the discrete linearized adjoint equation for a given direction δq_h and the corresponding δu_h.

4. Implement a function `h = computeJpp(q, u, z, dq, du, dz, args)` that computes a Hessian product with the vector δq. The vector $\mathbf{h}$ should relate to the second derivative in sense that

$$j''(q_h)(\tau q_h,\delta q_h) = \mathbf{h}'\mathbf{t}\mathbf{q}.$$

   Show that $\mathbf{h}$ is given by the formula $\mathbf{h} = \alpha\mathbf{M}_q\mathbf{dq} + \mathbf{B}'\mathbf{dz}$. What would be an appropriate expression for the Hessian vector product $\nabla^2 j(q_h)\delta q_h \in V_h$?

5. Verify the correctness of your implementation so far. The usual approach is a comparison of the directional derivative with the central difference quotient,

$$j''(q)(\delta q, \delta q) \approx \frac{1}{\varepsilon^2}\Big[j(q+\varepsilon\delta q) - 2j(q) + j(q-\varepsilon\delta q)\Big],$$

for a series of small $\varepsilon > 0$. You can do this by calling `CheckHessian(q, dq, @solveState, @solveAdjoint, @solveTangent, @solveDFH, @computeJ, @computeJpp, args)`.

6. Implement a local Newton method to compute the minimizer of the semilinear problem (1) for $\sigma = 1$ and $\alpha \in \{10^{-1}, 10^{-3}, 10^{-5}\}$. Implement a function `DampedNewton(solveState, solveAdjoint, solveTangent, solveDFH, computeJ, computeJp, computeJpp, args)`, based on the code for `GradientDescent`. Starting from some initial guess q_0, build a sequence of q_k that converges to a local optimum using the update

$$j''(q_k)(\delta q, \varphi) = -j'(q_k)(\varphi) \quad \text{for all } \varphi \in Q \tag{5}$$

$$q_{k+1} = q_k + \delta q \tag{6}$$

until the usual stopping criterion is satisfied.

- Write first a local function `computeHessian(q, u, z, dq, solveTangent, solveDFH, computeJpp, args)` that computes an application of the Hessian (the vector **h** from above) for a given direction **dq**. For this you should call `solveTangent`, `solveDFH`, and `computeJpp` that you already implemented.
- Make an anonymous function `H` depending only on **dq** with the code

```
H = @(dq) computeHessian(q, u, z, dq, solveTangent, ...
                                 solveDFH, computeJpp, args);
```

- The Newton step is then solved with the `pcg` method. Recall that this method requires that the Hessian shall be symmetric, positive and definite. The first property is always fulfilled. The last two properties are ensured when we are close to a local optimum.
- When the Hessian is positive definite, the Newton direction is also a descent direction. In that case, it is possible to use a damping strategy such as the Armijo rule. This can be used to globalize the Newton method.

7. Apply your Newton method also in the case $\sigma = 0$ (cf. Ex 1.1). Solve the same configuration with the code from Ex 1.1 and compare the solutions (they should coincide). Compare the runtime.

8. Consider again the Newton method applied to (1) in the case $\sigma = 0$. Show that for an initial guess equal to $q_0 = 0$, the method based on the cg-method (from sheet 1) and the Newton-cg-method from this sheet perform essentially the same steps.

Exercise 3.2 (Parameter estimation: the Gauss-Newton method): We consider again the problem from Ex. 2.2 but we want to use second order information to speed up the optimization.

$$\min_{q\in\mathbb{R}^2, u\in H^1(\Omega)} J(u) = \frac{1}{2}\sum_{i=1,\dots,4} (C_i u - c_{d,i})^2 \tag{7a}$$

$$\text{subject to } \begin{cases} -\Delta u + u + e^{q_1} u^3 = 0 & \text{in } \Omega, \\ \partial_n u + e^{q_2} u = f & \text{on } \partial\Omega. \end{cases} \tag{7b}$$

To make a second order implementation for this example easier, we don't compute the full Hessian, but a sufficiently good approximation for it. We define

$$h(q)(\delta q, \tau q) := J''_{uu}(u)(\delta u, \tau u) \quad \Big[\approx j''(p)(\delta q, \tau q)\Big], \tag{8}$$

where δu and τu are solutions of the tangent equation (3) at q and $u = S(q)$ corresponding to δq and τq. (If the functional J would depend directly on q, we would also include the partial derivatives with respect to q. However, we neglect the term $J'_u(u)(\delta\tau u)$ containing the derivative of δu with respect to τq.)

Instead of the full Hessian, we will use $h(q)$ to implement a quasi Newton strategy:

- Show that $h(q)(\cdot,\cdot)$ is a positive semidefinite bilinear form in the case of problem (7).

  Note, that this would not be true necessarily for the second derivative $j''(p)(\cdot,\cdot)$.

- Show that $h(q)$ can be evaluated by

  $$h(q)(\delta q, \tau q) = -a'_q(q,u)(\tau q, \delta\hat{z}), \tag{9}$$

  where $\delta\hat{z}$ is a solution of the modified linearized adjoint equation

  $$a'_u(u)(\varphi, \delta\hat{z}) = J''_{uu}(u)(\varphi, \delta u), \tag{10}$$

  where u is the state solution corresponding to q and δu is the tangent solution (3) at q and $u = S(q)$ corresponding to δq.

- Implement `solveTangent` for problem (7).

- Implement `dz = solveModifiedDFH(q, u, z, dq, du, args)` for problem (7), which solves equation (10) in the discrete case.

- Implement `h = computeModifiedJpp(q, u, z, dq, du, dz, args)` for problem (7), which computes the vector

  $$\mathbf{h} = \big(h(q)(\delta q, e_1), h(q)(\delta q, e_2)\big)$$

- In this case, you can not verify correctness of the implementation with `CheckHessian`. However, you can check that your implementation produces a symmetric, positive definite matrix. Therefore, assemble the full matrix

  $$\mathbf{H} = \big(h(q)(e_j, e_i)\big)_{ij}.$$

  by using `solveTangent`, `solveModifiedDFH`, and `computeModifiedJpp` twice, and check numerically the symmetry and the sign of the eigenvalues (using `eig`).

- Now, you can use the function `DampedNewton` from before to solve the problem. Since the matrix $\mathbf{H}$ is positive semidefinite, the use of conjugate gradients and a globalization strategy based on the (quasi-)Newton direction is justified.

  For this example, the globalization strategy (requiring objective functional descent) is actually needed.

- Compare the runtime to the gradient method from Ex 2.2.

- *Bonus:* For this example, we only have a small amount of parameters (namely two). Therefore it is also possible, to assemble the full matrix $\mathbf{H} \in \mathbb{R}^{2\times 2}$ directly with formula (8) and the two tangent solutions corresponding to $\delta q = e_{1,2}$.

  Implement this method. Therefore you have to implement a version of `DampedNewton` that works on the full Hessian.

- *Bonus^2:* Show that the approximation (8) is justified in points $u = S(q)$ where the residual $C_i u - c_{d,i}$ is small. (It is exact for $C_i u - c_{d,i} = 0$).

  How does this justify the approximation of the Hessian?

  How would you have to modify the algorithm in the case that the residual is not small in the optimum (this is typically the case, if there are a large number of measurements)?