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 2

Exercise 2.1 (The semi-linear elliptic test problem): We consider a similar test problem as in Ex. 1.1, but with a cubic non-linearity appearing in the PDE.

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

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

The parameter $\sigma \geq 0$ determines how strongly nonlinear the problem is. The domain is the square $\Omega = (-1,1)\times(-1,1)$ and as desired states we consider (as before) either

$$u_d^{\text{eig}}(x) = \sin\left(\tfrac{\pi}{2}x_1\right)\sin\left(\tfrac{\pi}{2}x_2\right), \tag{u_d^{eig}}$$

$$\text{or} \quad u_d^{\text{char}}(x) = \chi_{\{\,|x|\leq\frac{1}{2}\,\}}, \tag{u_d^{char}}$$

where $|x| = \sqrt{x_1^2+x_2^2}$ and χ is a characteristic function.

1. To solve (1b) we use Newton's method *(see sheet 0, Ex 0.3)*. Write the equation for the Newton update in a weak form for the discrete update w_h

   $$a'_u(u_h)(w_h,\varphi) = (q_h,\varphi) - a(u_h)(\varphi) \quad \text{for all } \varphi \in V_h$$

   as well as in a matrix vector notation $\mathbf{Aw} = \mathbf{r}$. How does $\mathbf{r}$ depend on $\mathbf{q}$?

   Implement a function `u = solveState(q, args)` that solves the discrete state equation for a given control q_h.

2. In a first step we want to compute the functional value of the reduced cost functional $j_h(q_h) = J(S_h(q_h), q_h)$.

   Implement a function `j = computeJ(q, u, args)` that computes the functional value for given q and u.

   Hint: use the function `ComputeFunctional(mesh, @J, data)` *for an appropriate functional* `function j = J(localdata)`.

3. The gradient of the reduced cost functional can be represented with help of the adjoint state z. Write the adjoint equation in weak formulation in the continuous case and find the discrete adjoint equation for z_h. Also write it in matrix-vector notation in terms of the vector $\mathbf{z} \in \mathbb{R}^N$. Which (stiffness-) matrix do you have to assemble? What is the relation to the matrix you have to assemble for Newton's method above?

   Implement a function `z = solveAdjoint(q, u, args)` that solves the discrete adjoint equation for a given state u_h.

4. In the continuous case, the derivatives of j can be represented with the formula

$$j'(q)(\delta q) = (\alpha q + z, \delta q) \quad \text{for any } \delta q \in L^2(\Omega).$$

To transfer this to the discrete level, we consider the Lagrange function in the discrete setting,

$$\mathcal{L}(q_h, u_h, z_h) = J(q_h, u_h) - a(u_h)(z_h) + (q_h, z_h).$$

Using this function, show that for any direction δq_h (represented by the vector $\mathbf{dq}$) we have

$$j_h'(q_h)(\delta q_h) = \alpha(q_h, \delta q_h) + (z_h, \delta q_h) = \alpha \mathbf{q}' \mathbf{M}_q \, \mathbf{dq} + \mathbf{z}' \mathbf{B} \, \mathbf{dq},$$

where z_h (resp. $\mathbf{z}$) is the discrete adjoint state. Based on this expression, implement a function `g = computeJp(q, u, z, args)` that computes the vector $\mathbf{g} = \alpha \mathbf{M}_q \mathbf{q} + \mathbf{B}' \mathbf{z}$.

How are the entries of $\mathbf{g}$ related to the partial derivative $j'(q_h)(\delta q_h)$?

5. What is an appropriate expression for the gradient $\nabla j(q_h) \in V_h$ on the discrete level?

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

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

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

7. Implement a gradient based descent method (with Armijo line-search) in the method `GradientDescent` to compute the minimizer of the semilinear problem (1) for $\sigma = 1$, $\alpha \in \{10^{-1}, 10^{-3}, 10^{-5}\}$ for (u_d^{char}).

   - The Armijo rule for the gradient method states that you should choose the step s_k such that

   $$j(q_k + s_k d_k) - j(q_k) \leq -\gamma s_k \|\nabla j(q_k)\|^2 \tag{3}$$

   where $d_k = -\nabla j(q_h)$ is the direction. As long as (3) is not satisfied, decrease s_k by a factor $\beta \in (0, 1)$. This means that you start with an initial step s_{init}, and update it, if necessary by $\{\beta s_{init}, \beta^2 s_{init}, ...\}$. (In practice, you would often choose $\gamma = 0$, which means that you only check for descent). You are encouraged to play with the parameters to understand better the influence of each one of them on the algorithm. You can stop the iteration once $\|\nabla j(q_h)\| \leq TOL$.

   Tip: Since the gradient method is quite slow, solve only up to a high tolerance (e.g. $TOL = 10^{-3}$). A choice of $s_{init} = 4$ should lead to good results for this problem already with a refinement level of 4.

8. During the gradient descent, the state does not change much from one iteration to another, especially in the later steps. Therefore, we can speed up the function `solveState(q, args)` by adding the possibility for an initial guess that will help the Newton method converge faster. At the beginning of the `solveState` function add the following code

```
if isfield(args, 'u_0')
   u = args.u_0;
else ...
```

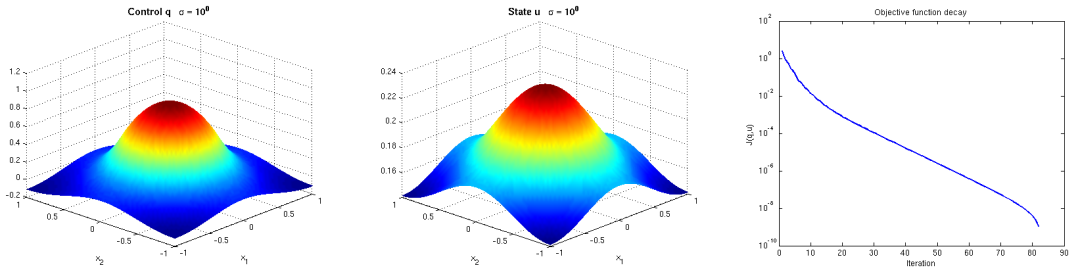

Figure 1: Case $u_d = u_d^{\text{char}}$, $\alpha = 10^{-1}$. The last plot is a semilog plot of J(current iteration) - J(last iteration)

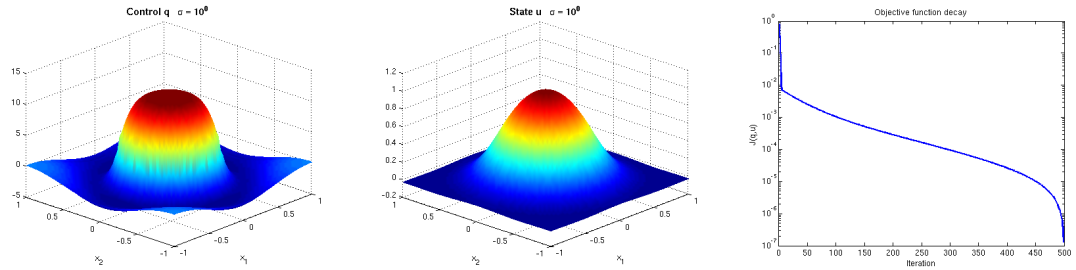

Figure 2: Case $u_d = u_d^{\text{char}}$, $\alpha = 10^{-3}$.

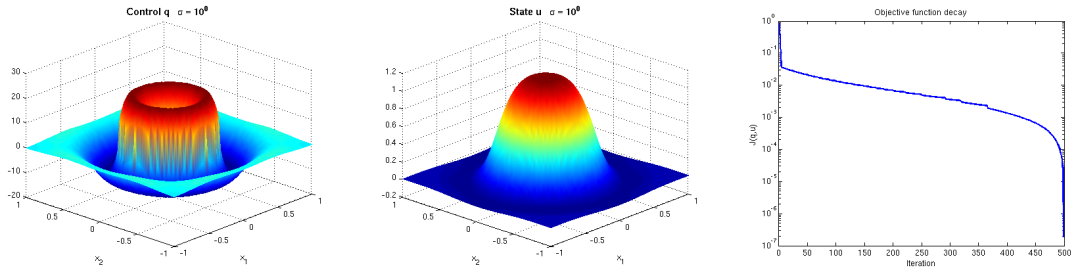

Figure 3: Case $u_d = u_d^{\text{char}}$, $\alpha = 10^{-6}$.

Then, you can set this initial guess in the `GradientDescent` method, specifying `args.u_0 = u;` in the Armijo loop.

9. Apply the gradient 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, up to some tolerance). Compare the runtime.

Exercise 2.2 (A problem with finite dimensional control/observation): We consider the following problem on the square $\Omega = (-1, 1) \times (-1, 1)$:

$$\min_{q=(q_1,q_2)\in\mathbb{R}^2, u\in H^1(\Omega)} J(u) = \frac{1}{2} \sum_{i=1,\dots,4} (C_i u - c_{d,i})^2 \tag{4a}$$

$$\text{subject to } \begin{cases} -\Delta u + u + e^{q_1} u^3 = 0 & \text{in } \Omega, \\ \partial_n u + e^{q_2} u = f & \text{on } \partial\Omega. \end{cases} \tag{4b}$$

Here, we try to find the model parameters $q = (q_1, q_2) \in \mathbb{R}^2$, which we regard as the control. Therefore we perform a least-squares fit of the local averages

$$C_i u = \int_{\Omega_i} u(x)\,\mathrm{d}x$$

to a given observation $c_d = (c_{d,i}) \in \mathbb{R}^4$. The subdomains are given by $\Omega_1 = (-1,0) \times (-1,0)$, $\Omega_2 = (0,1) \times (-1,0)$, $\Omega_3 = (0,1) \times (0,1)$, and $\Omega_4 = (-1,0) \times (0,1)$. The inhomogeneity on the boundary is given by $f(x_1, x_2) = 1 + x_1 + x_2$.

The measurement data $c_d = (c_{d,i}) \in \mathbb{R}^4$ are given by:

$c_{d,1}$	$c_{d,2}$	$c_{d,3}$	$c_{d,4}$
0.04610	0.50582	0.81850	0.50582

They are also provided in the file `integralvalues.mat`. Load them with the command `c_d = load('integralvalues.mat', '-ascii');`

Follow the same steps as in Ex. 2.1 to solve the problem. Here are some hints:

- Modify the Newton method from Ex 1.1 to include the boundary form. In this example the control, given by the two parameters $q = (q_1, q_2) = \mathbf{q}$, enters directly into the form for the state equation. Therefore we write the state equation as

  $$a(q,u)(z) = a_d(q_1,u)(z) + a_b(q_2,u)(z) = f(z) \quad \text{for all } z \in V,$$

  where a_d is the form with the domain integrals, and a_b the form with the boundary integrals. For a plot of the state variable with parameters $q = (q_1, q_2) = (0,1)$ see Figure 4.

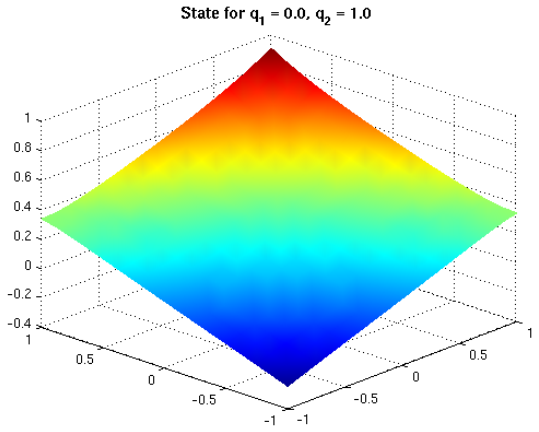

Figure 4: State for $q_1 = 0$ and $q_2 = 1$.

- To compute the integrals $C_i u_h$, implement four functionals `chiD1(data)`, `chiD2(data)`, `chiD3(data)` and `chiD4(data)` to be used in conjunction with `ComputeFunctional(mesh, @chiD1, data)`, etc. Implement a function `Cu = computeCiu(u, args)`, which computes the vector $\mathbf{Cu} = (C_1 u_h, C_2 u_h, C_3 u_h, C_4 u_h)$ for a given state u_h. The values for $q_1 = 0$ and $q_2 = 1$ (on refinement level 6) are given by:

C_1u	C_2u	C_3u	C_4u
0.0163	0.2678	0.5107	0.2678

- Derive the adjoint equation. How do the values **Cu** enter in it? Implement the right hand side `ju = J_u(phi, data)` for the adjoint equation. Reuse the function `computeCiu` from above and pass the vector **Cu** with `data.param`. Compare with Figure 5.

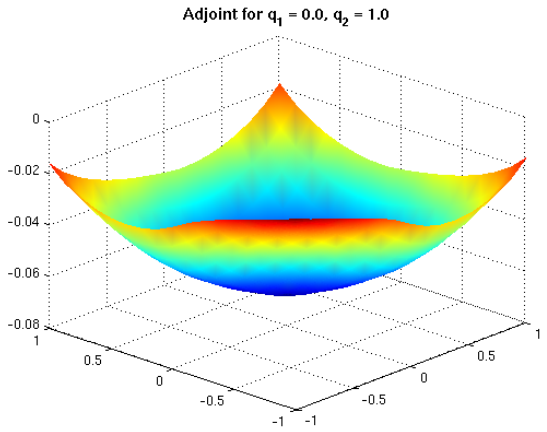

Figure 5: State for $q_1 = 0$ and $q_2 = 1$.

- In this example, the gradient of the reduced cost functional j is given by

$$\nabla j(q) = (j'(q)(e_1),\, j'(q)(e_2)),$$

  where $e_{1,2}$ are the two canonical basis vectors of $\mathbb{R}^2$. The two entries of $\nabla j(q)$ can be computed with `ComputeFunctional` and `ComputeBoundaryFunctional` in conjunction with two functionals `a_q1` and `ab_q2`.

- You should be able to keep the same structure for the code as Ex 2.1 and re-use the general method `GradientDescent`. The changes should occur in

  - `solveState`, to account for the additional boundary terms and the different control space.
  - `computeJ`, where the main work is now done in `computeCiu(u, args)`.
  - `solveAdjoint`, to account for the additional boundary terms and the objective functional.
  - and `computeJp`, where we need to evaluate the terms $a'_q(q,u)(\delta q, z)$ for the basis vectors $\delta q = \mathbf{dq} = e_{1,2} \in \mathbb{R}^2$.

- Do not forget to use again `CheckGradient` to check the validity of your gradient computation before starting the gradient descent algorithm.

- For this example, the control is not discretized, i.e., $q = \mathbf{q} \in \mathbb{R}^2$. What is therefore the mass matrix of the control space? How does the gradient $\nabla j(q)$ relate to $\mathbf{g}$ in this case?

- We would expect the optimal objective functional value to be zero. Solve the problem on refinement level 3 (with a small tolerance, e.g., 10^{-6}) and plot the functional values of each step of the gradient iteration (starting at $q_{init} = (0, 1)$) in a `semilogy` plot. What do you observe?
- Solve the parameter estimation problem on different refinement levels, starting at 1. You can use the control from the coarse mesh to initialize the iteration on the finer mesh. Tabulate the optimal controls on each level. Which convergence rate do you observe?