UNIVERSITÄT BERN

Applied Optimization
Exercise 6 - Newton's Method

Heng Liu Nicolas Gallego

Handout date: 07.11.2019

Submission deadline: 13.11.2019, 23:59 h

Hand-in instructions:

Please hand-in **only one** compressed file named after the following convention: `Exercise`n`-GroupMemberNames.zip`, where n is the number of the current exercise sheet. This file should contain:

- **Only** the files you changed (headers and source). It is up to you to make sure that all files that you have changed are in the zip.
- A `readme.txt` file containing a description on how you solved each exercise (use the same numbers and titles) and the encountered problems.
- Other files that are required by your `readme.txt` file. For example, if you mention some screenshot images in `readme.txt`, these images need to be submitted too.
- Submit your solutions to ILIAS before the submission deadline.

Affine Invariance (2 pts)

Show that Newton's method is invariant under affine transformations. Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a twice differentiable function. Consider the function $g(y) = f(Ay + b)$, where A is a non-singular constant matrix and $A \in \mathbb{R}^{n\times n}$,$b \in \mathbb{R}$. Prove that Newton's method yields precisely the same sequence of iterates for the two functions. In other words, the sequence of points $\{x^k\}$ generated by Newton's Method for f starting from x^0 with $x^0 = Ay^0 + b$ can be transformed to a sequence of Newton's iterations $\{y^k\}$ for the function g.

Programming (8 pts)

The objective this week is to implement the Newton method and two variations handling modifications of the hessian for non-convex problems. There is also a comparison with gradient descent regarding the scaling of the function, for this you will compare your implementation of the Newton method with Gradient descent.

Newton Method vs Gradient descent (4 pts)

Start by implementing the missing part of the function `solve()` of the class `NewtonMethods` in the file `include/NewtonMethonds.hh`. This method gets a pointer to a generic problem as argument, `ProblemBase*`, with this problem pointer you can access all the methods of the problem you need, such as the gradient, the hessian and the objective function `eval_f(...)` that you need for the implementation.

To solve for the newton direction Δx_{nt} you may be tempted to compute the inverse of the Hessian matrix of the problem. Don't do so. Instead we recommend using the Cholesky factorization and solve for the appropriate left hand side. Check the documentation for `LDLT` module of the Eigen library.

As line search strategy `back_tracking_line_search()` available from `LineSearch.hh`.

When you are done run the program `newton_methods` with option 1 for this part, optionally you may set an extra parameter for the scaling of the objective function being tested here. Your implementation will run first on the test problem implemented for you in the file `TestProblem.hh` study it for reference of the use of the `ProblemBase` class.

Experiment different values of gamma to test your implementation and compare with gradient descent. For zero of negative the values of gamma the system is not convex anymore, if you are curious fill free to experiment those cases as well.

Newton method with modified hessian (4 pts)

One modification to deal with problems that may exhibit non positive semi-definite hessians, is to modify the hessian matrix by adding a constant to all its diagonal entries until it is semi-positive definite as discussed in the lecture.

Implement this variant of the algorithm by writing the missing code of the method `solve_with_projected_hessian(...)` of the the class `NewtonMethods`

in the file `include/NewtonMethonds.hh`.

The idea is to attempt to perform the Cholesky decomposition of the hessian matrix H of the problem, and if it fails then, starting from an initial value $\tau = \tau_0$, iterate on adding this value to the diagonal, $H \leftarrow H + \tau I$, until the matrix is positive semi-definite. The value τ is increased on every iteration by `tau` $\leftarrow$ `tau` $\times$ tau_factor with `tau_factor` > 1.0. The recommended value for τ_0 is

$$\tau_0 = 10^{-3} |\text{trace}(H)| / n$$

where n is number of variables of the problem. The rest of the algorithm remains the same as the normal newton method.

When done try running `newton_methods` with option 2 for this part. This fetches an instance of the non-convex version of the *SpringProblem* for you and runs this variant of the algorithm for you. Try both the version without fixing springs and the the one with fixing springs (modified mass spring system), currently commented.

Bonus (3 pts)

Two bonus optional bonus exercises are provided to you, choose one:

Option 1

The partial separability can be exploited to modify local hessians instead of the global one as done above. For this build an instance of a problem that has this property, and perform the following modification to the local hessians, in our spring elements are this 4×4 local hessian matrices.

You need to do the eigendecomposition of the hessian

$$\nabla^2 f(x) = Q^T \Lambda Q = \sum_{i=1}^{n} \lambda_i q_i q_i^T,$$

Check each that the eigenvalues are greater than a positive ϵ otherwise add a value m_i accordingly.

$$m_i = \begin{cases} 0, & \lambda_i \geq \epsilon \\ \epsilon - \lambda_i, & \lambda_i < \epsilon \end{cases}$$

Multiply again with the modified eigenvalues m_i to get the matrix

$$M = Q^T \text{diag}(m_i) Q, \quad i = 1 \ldots n$$

to get a modified Hessian $\tilde{\nabla}^2 f(x) = \nabla^2 f(x) + M$. Recall here n refers to the size of the local problem, not the global one.

Option 2

Implement a problem of your interest that can be formulated as an unconstrained optimization problem, and solve it with our available methods, `GradientDescent`, and the variants of the `NewtonMethods` you implemented in this exercise. Comment on the results of your experiments. Take as inspiration the `TestProblem.hh` to create a class that implements the *BaseProblem* interface.

Don't forget to include the extra code files you may generate and the modified `main.cc`.