b
UNIVERSITÄT
BERN

Applied Optimization
Exercise 6 - Newton's Method

November 17, 2022

What to hand in

A .zip compressed file renamed to `Exercise`n`-Group.zip` where n is the number of the current exercise sheet. It should contain:

- Hand in **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. Indicate what fraction of the total workload each project member contributed.
  Failure to do so will result in you losing a point.
- 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.
- For the theory exercise, put `TheoryExercise.pdf` with your solutions in the same .zip you submit for the code.
- Submit your solutions to ILIAS before the submission deadline. Late submissions will receive 0 points!
- **Important:** Your submission MUST compile without any issue in order to obtain a passing grade.

Affine Invariance (3 (+2) pts)

(a) 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 the sequence of points $\{x^k\}$ of f and the sequence of points $\{y^k\}$ of g generated by Newton's Method, starting from x^0 and y^0 respectively with $x^0 = Ay^0 + b$, have one-to-one correspondence under the transformation, i.e.$f(x^k) = g(y^k)$. Here we assume that the step lengths of the search directions of both functions are equal.

(b) Under affine transformations, show that the Newton's method with backtracking line search generates the same step length, i.e. the step length s^k of the function $f(x)$ equals to t^k of the function $g(y)$. In addition, show that the newton decrements of both functions are identical. (**Bonus (2 pts)**)

Programming (7 pts)

The task is to implement the Newton's method and two variations handling modifications of the hessian for non-convex problems. You also need to compare the Newton's method with gradient descent method run on problems of different scales.

Standard Newton Method (3 pts)

Start by implementing the missing part of the function `solve(...)` of the class `NewtonMethods` in the file `Algorithms/NewtonMethods.hh`. This function is the implementation of the standard Newton solver. The line search strategy `back_tracking_line_search()` is available in `LineSearch.hh`.

`Hint`: To solve for the newton direction Δx_{nt} you may attempt to compute the inverse of the Hessian matrix of the problem. It is inefficient! 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.

Newton method with modified hessian (3 pts)

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

Implement this variant of the algorithm by writing the missing code of the function `solve_with_projected_hessian(...)` in the file `Algorithms/NewtonMethonds.hh`.

It attempts to perform the Cholesky decomposition of the hessian matrix H of the problem, and if it fails then, starting from an initial value $\delta = \delta_0$, iterate on adding this value to the diagonal, $H \leftarrow H + \delta I$, until the matrix is positive definite. The value δ is increased on every iteration by $\delta \leftarrow \delta \times \gamma$ with $\gamma > 1.0$. The recommended value for δ_0 is

$$\delta_0 = 10^{-3}|\text{trace}(H)|/n$$

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

Newton Methods vs Gradient descent (1 pts)

The `MassSpringProblem` in the `main.cc` is set up the same as in the `GradientDescent` exercise. Use Newton's methods and GradientDescent method to solve the `MassSpringProblem` and compare the performance and results. Experiment on the spring graphs of dimension 5×5, 10×10 and 20×20 of $SpringElement2D$ and $SpringElement2DWithLength$. Submit the timing of each instance and the screenshots of the final results.

Bonus (3 pts)

The partial separability can be exploited to modify local hessians instead of the global one as done above. For example, for the spring element with the non-convex function we need to project the local hessians which are of 4×4 to positive definite matrices. Implement the $eval\_hessian(...)$ in the `Functions/SpringElement2DWithLengthPSDHess.hh`. After the modification, the `MassSpringProblem` can be solved with the standard Newton solver. You can test your implementation by running the `NewtonMethod` executable, using the corresponding function index.