MATH-329 Nonlinear optimization
Homework 2: Trust-regions

Instructor: Nicolas Boumal
TA: Quentin Rebjock

Due: April 15, 2022, by end of day

Typeset your answers (preferably using LaTeX) and submit a PDF on Moodle before the deadline. Late homework is not accepted. Where applicable, provide runnable code in separate files *and* in your report (see Moodle for how to include Matlab code in LaTeX). Writing quality and clarity as well as mathematical rigor affect grading substantially. As much as conciseness permits, demonstrate you understand what is going on. You may use results from the course without proving them. However, you should reference explicitly the theorems that you are using and check that the assumptions are satisfied. **We will run your code as is; we will not try to debug it.**

In this homework you will use optimization to solve an inverse problem in imaging. Suppose that we are using a telescope to observe stars in the sky. Optical tools obtain imperfect images because their resolution is necessarily limited. For this reason we never get to observe the exact signal, only an altered version of it. However, we can derive models for the perturbations induced by the measurement tools. The idea of solving an inverse problem is to use such models to reconstruct the true signal from the observations. Our specific setup is as follows (much simplified for the purpose of the homework). **Data is provided on Moodle.**

(a) We define a signal as a continuous function $\psi\colon \mathbb{R}^2 \to \mathbb{R}$. We let $\mathfrak{F}$ be the set of all signals.

(b) Measurement tools never detect entire signals. Instead, they form an image composed of $n \times n$ pixels: a *sampling* of the signal. It would be natural to store such an image as a matrix in $\mathbb{R}^{n\times n}$. Instead, we store them as vectors in $\mathbb{R}^{n^2}$ for the expressions to be more computer-friendly. The sampling operator is denoted by

$$S\colon \mathfrak{F} \to \mathbb{R}^{n^2}.$$

It forms an image given a signal. We assume that S samples the values from an $n \times n$ grid of evenly spaced points in the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$. Explicitly, let $P \in \mathbb{R}^{2\times n^2}$ be the matrix whose columns $p_1, \ldots, p_{n^2} \in \mathbb{R}^2$ are the positions of the pixels that we sample. Then,

$$S(\psi) = \begin{bmatrix} \psi(p_1) \\ \psi(p_2) \\ \vdots \\ \psi(p_{n^2}) \end{bmatrix}.$$

Notice that S is a linear map.

(c) Even with the best telescopes, stars do not appear as pure points but rather as diffuse points. We model the corruption of the signal with a Gaussian filter $h\colon \mathbb{R}^2 \to \mathbb{R}$ defined by[1]

$$h(x) = \exp\Big(\frac{-\|x\|^2}{\sigma^2}\Big).$$

We always take $\sigma = 0.1$. The signal coming from one star located at some position $x_1 \in \mathbb{R}^2$ is simply h translated to be centered around x_1, that is, $x \mapsto h(x - x_1)$.

(d) Suppose there are K stars with exact locations $x_1, \ldots, x_K \in \mathbb{R}^2$. Arrange them as the columns of $X \in \mathbb{R}^{2\times K}$. The signal ψ_X we observe is the sum of the signals received from each individual star, that is,

$$\psi_X(x) = \sum_{k=1}^{K} h(x - x_k).$$

This is a convolution: you can read about it on Wikipedia.

(e) The image we actually detect is a sampling of ψ_X, that is

$$\Phi(X) = S(\psi_X).$$

If the true (unknown) star positions are $X^\star \in \mathbb{R}^{2\times K}$, we observe

$$y = \Phi(X^\star) \in \mathbb{R}^{n^2}.$$

Our goal is to recover $X^\star$ given y.

To simplify the math, it is helpful to introduce a few more objects. Define $\varphi\colon \mathbb{R}^2 \to \mathbb{R}^{n^2}$ by

$$\varphi(x) = \begin{bmatrix} h(p_1 - x) \\ \vdots \\ h(p_{n^2} - x) \end{bmatrix}.$$

(What does this do?) Then we can rewrite Φ as

$$\Phi(X) = \sum_{k=1}^{K} \varphi(x_k).$$

We model the problem of finding $X^\star$ given y as that of minimizing the function

$$f(X) = \frac{1}{2}\|\Phi(X) - y\|^2.$$

Indeed, f measures the mismatch between tentative positions X and the observations y. We hope that if we minimize f we will be able to reconstruct the signal from the corrupted observations.

1. Implement a function `phi(x, P)` that takes as input $x \in \mathbb{R}^2$ and the pixel positions $P \in \mathbb{R}^{2\times n^2}$ and returns as output the value $\varphi(x)$. Accordingly, implement the functions `Phi(X, P)` and `f(X, P, y)` that, given $X \in \mathbb{R}^{2\times K}$, return $\Phi(X)$ and $f(X)$ respectively.

[1]The norm $\|\cdot\|$ is the one induced by the canonical inner product $\langle\cdot,\cdot\rangle$. We always use these in this homework.

2. Is f convex? If yes give a proof, if not plot f along a line segment where we can clearly see that it is not.

3. Obtain an expression for the differential of φ, $\mathrm{D}\varphi(x)[u]$.

4. Find the adjoint of $\mathrm{D}\varphi \colon \mathbb{R}^2 \to \mathbb{R}^{n^2}$ with respect to $\langle \cdot, \cdot \rangle$.

5. Find the gradient of f (you can get a nice expression if you exploit your answers from the two previous questions).

6. Implement a function that computes the gradient of f. Check numerically that your gradient is correct. To do so, use the same technique as in Homework 1 and show the results.

The function f is rather complicated and obtaining a Lipschitz constant for the gradients would be difficult. For this reason we are not going to test gradient descent with constant step-sizes and instead use backtracking line-search directly.

7. Write a backtracking line-search gradient descent algorithm. You may reuse code from the exercise sessions or from Homework 1 but it must be your own code. For some of the questions below you may have to tune the parameters $\bar{\alpha}, \rho, c$ of the line-search: be pragmatic. Typical values are $\bar{\alpha} = 1, \rho \in [0.5, 0.8]$ and $c = 10^{-4}$.

You will test algorithms on the two datasets `data-toy.mat` and `data.mat` provided on Moodle. You can load them in Matlab with the command `load`. They both contain the dimensions of the problem K, n, the $2 \times n^2$ matrix P containing the pixel positions, and the observations $y \in \mathbb{R}^{n^2}$. For the dataset `data-toy.mat` there are $K = 3$ stars that are well separated from each other and the image size is 32×32. We use this dataset mostly for debugging purposes. Additionally, for this dataset we provide the actual positions $X^\star$ so you can check that your results are reasonable. For the dataset `data.mat` there are $K = 10$ stars and the image is 512×512.

In all the following experiments we suggest to initialize your algorithms with some positions in the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$. Random should be good sometimes.

8. Run the gradient descent algorithm on `data-toy.mat`. Report the positions that you obtain and plot the image resulting from them. How do you check that your results make sense? (Make sure they do, because you will build on these initial steps.)

9. Run the backtracking line-search method several times on `data.mat` with different random initial points. Do you always reach the same objective value regardless of the initial point? In a few words, how do you explain this behavior?

10. For a few runs and all on the same plot, show the norm of the gradient as a function of the iteration, with a log-scale on the vertical axis.

We are now going to implement second-order methods and compare their performance to that of gradient descent. We will focus specifically on the trust-region algorithm discussed in class. For this we need to have access to the Hessian.

11. Obtain a formula for the Hessian of f, that is, compute the quantity $\nabla^2 f(X)[U]$ for some arbitrary point $X \in \mathbb{R}^{2\times K}$ and direction $U \in \mathbb{R}^{2\times K}$. Remember that $\nabla^2 f(X)$ is a linear map $\mathbb{R}^{2\times K} \to \mathbb{R}^{2\times K}$ so $\nabla^2 f(X)[U]$ must be a matrix in $\mathbb{R}^{2\times K}$. We recommend that you derive this expression carefully as you will use it a lot in the remaining questions. You may find it helpful to consider the exercises from week 4.

In case you fail to obtain or implement the Hessian you may answer the remaining questions using a finite difference approximation of it. The idea is that for $X, U \in \mathcal{E}$ and t small we have

$$\nabla^2 f(X)[U] \approx \frac{\nabla f(X+tU) - \nabla f(X)}{t}.$$

You should not choose t to be too small to avoid round-off errors; $t = 10^{-4}$ is a reasonable value.

12. Write a function that takes a point X and a direction U, and that returns the quantity $\nabla^2 f(X)[U]$.

    Check numerically that your Hessian is correct. To this end, generate a random point X and a direction U, setup a vector of logarithmically spaced values `t = logspace(-8, 0, 100)`, and plot the error

    $$e(t) = \left| f(X+tU) - f(X) - t\left\langle U, \nabla f(X)\right\rangle - \frac{t^2}{2}\left\langle U, \nabla^2 f(X)[U]\right\rangle \right|$$

    versus t in loglog scale (show the plot). This should be $O(t^3)$, meaning that the plot should look like a straight line with slope 3. Is that what you see? Comment briefly on possible discrepancies: are they ok or not ok?

13. Implement a function that performs truncated-CG: see lecture notes.

14. Implement a trust-region solver. For the symmetric linear map "H_k" from the lecture notes, pick the exact Hessian at the current iterate. You can play with the parameters of the algorithm. Reasonable default values are $\bar{\Delta} = \sqrt{2K}$, $\Delta_0 = \bar{\Delta}/8$ and $\rho' = 1/10$.

15. Run your trust-region method with truncated-CG on `data-toy.mat`. Report the positions that you obtain and plot the image resulting from them. Make sure that it looks reasonable.

16. Pick an initial point X and run the gradient descent and the trust-region methods, both starting from X, on the dataset `data.mat`. Plot the norm of the gradient as a function of the iteration for both methods on the same plot (gradient norm always on a log-scale). Do the same thing but as a function of time (using functions `tic` and `toc`).

17. What is your best possible guess for $X^\star$? Provide a file `Xstar.mat` containing a variable `Xstar`. It should be a $2 \times K$ matrix whose columns are the positions of the stars. To create it you can use the command `save('Xstar.mat', 'Xstar')`. Plot the image resulting from $X^\star$ and compare to the observations y: comment briefly.