Numerical Optimization - Hand-In 5

Isabela Blucher

March 9, 2018

Exercise 10.1

(a) Show that J has full column rank if and only if J^TJ is nonsingular.

If J has full column rank we know that $Jx = 0 \implies x = 0$. And if J^TJ is non singular we know that $J^TJx = 0 \implies x = 0$. We can show both ways of this proof by manipulating these properties.

($\Longrightarrow$) We know here that J has full column rank.
Starting with $J^TJx = 0$, we can get to the property for a full column rank matrix. $J^TJx = 0 \implies x^TJ^TJx = 0 \implies (Jx)^T(Jx) = 0 \implies \|Jx\|^2 = 0 \implies Jx = 0$, and since J has full column rank, $x = 0$ and J^TJ is nonsingular.

($\Longleftarrow$) We know here that J^TJ is nonsingular.
Staring with $Jx = 0$, we can get to the property of a nonsingular matrix. $Jx = 0 \implies J^TJx = 0$, since we know that J^TJ is nonsingular, $x = 0$ and J has full column rank.

(b) Show that J has full column rank if and only if J^TJ is positive definite.

Repeating the property for the full column rank matrix, we know that $Jx = 0 \implies x = 0$. And for the positive definite matrix J^TJ we know that $x^T(J^TJ)x > 0, \forall x \neq 0$ and $x^T(J^TJ)x = 0 \implies x = 0$.

($\Longrightarrow$) We know here that J has full column rank.
For $x \neq 0$, we can start with $x^T(J^TJ)x = (Jx)^T(Jx) = \|Jx\|^2$ and since the norm is squared we clearly see that $\|Jx\|^2 > 0, \forall x \neq 0$. For the other case we do the same steps but equal to zero, and when we get to $\|Jx\|^2 = 0$ we know that implies $Jx = 0$ and so $x = 0$ because J has full column rank and J^TJ is positive definite.

($\Longleftarrow$) We know here that J^TJ is positive definite.
Starting with $Jx = 0$ we know that implies $x^TJ^TJx = (Jx)^T(Jx) = \|Jx\|^2 = 0$, and since J^TJ is positive definite, $x = 0$ and J has full column rank.

Exercise 10.2

Show that the function $f(x) = \frac{1}{2}\|Jx - y\|^2$ is convex.

From the textbook, we know that the definition of a convex function is a function f whose domain S is a convex set and that for any two points $x_1, x_2 \in S$ and for all $\alpha \in [0, 1]$ satisfies

$$f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2) \tag{1}$$

If we use the left side of inequality (1) with our function from the exercise we get

$$\frac{1}{2}\|J(\alpha x_1 + (1 - \alpha)x_2) - y\|^2 = \frac{1}{2}\|\alpha Jx_1 + (1 - \alpha)Jx_2 - y\|^2 \tag{2}$$

By rewriting equation (2) with $-y = -\alpha y - (1 - \alpha)y$ we have

$$\frac{1}{2}\|\alpha Jx_1 + (1 - \alpha)Jx_2 - \alpha y - (1 - \alpha)y\|^2 = \frac{1}{2}\|\alpha(Jx_1 - y) + (1 - \alpha)(Jx_2 - y)\|^2 \tag{3}$$

With the triangle inequality, we have that

$$\frac{1}{2}\|\alpha(Jx_1 - y) + (1 - \alpha)(Jx_2 - y)\|^2 \leq \frac{1}{2}(\alpha\|Jx_1 - y\|^2 + (1 - \alpha)\|Jx_2 - y\|^2) \tag{4}$$

The right-hand side of inequality (4) is the same as $\alpha f(x_1) + (1 - \alpha)f(x_2)$, which means that the function f is convex.

Exercise 10.3

(a) if Q is an orthogonal matrix, then $\|Qx\| = \|x\|$ for any vector x.

Let $Q \in \mathbb{R}^{n\times n}$ be an orthogonal matrix and $x \in \mathbb{R}^n$ be a given vector. If we define $q_i, i = 1, ..., n$ to be the i-th column of Q, from the definition of an orthogonal matrix we have

$$q_i^T q_j = \begin{cases} \|q_i\|^2 = 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Then

$$\begin{aligned} \|Qx\|^2 &= (Qx)^T(Qx) \\ &= (x_1q_1 + ... + x_nq_n)^T(x_1q_1 + ... + x_nq_n) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_ix_jq_i^Tq_j \end{aligned} \tag{6}$$

From equation (5) we know that $q_i^T q_j$ is only equal to 1 when $i = j$, and so, we can equal equation (6) to

$$\|Qx\|^2 = \sum_{i=1}^{n} x_i^2 = \|x\|^2 \tag{7}$$

Thus, $\|Qx\| = \|x\|$.

(b) the matrices $\bar{R}$ in (10.15) and R in (10.17) are identical if $\Pi = I$, provided that J has full column rank n.

From (10.15) we have $J^TJ = \bar{R}^T\bar{R}$ where $\bar{R}$ is a $n\times n$ upper triangular matrix with positive diagonal elements, and from (10.17) we have $J\Pi = Q_1R$, where Q_1 is the first n columns of an orthogonal matrix Q and R is an $n \times n$ upper triangular matrix with positive diagonal elements.

If $\Pi = I$ we have $J = Q_1R \implies J^TJ = (Q_1R)^T(Q_1R) = R^TR$. Since we know that the Cholesky decomposition is unique if the diagonal elements of the upper triangular matrix are positive, we can conclude that $\bar{R} = R$.

Programming Assignment

The Levenberg-Marquardt method was implemented with the trust-region algorithm used last week as a base, some modifications were made, such as the model function $m_k(p)$ and how the step direction was chosen at each iteration.

To compare the LM method with the previously implemented Steepest Descent, parameter values that yielded good performance for both algorithms were fixed, so that we could analyze a "best case scenario" for both algorithms. The stopping criteria for both algorithms were if the norm of the gradient got smaller than a fixed tolerance or if the number of iterations got bigger than a fixed maximum value. For both algorithms, the same values for the stopping criteria were used: 10^{-6} for the gradient tolerance and 10^4 for the maximum number of iterations.

For the Tensor Fitting problem we are given a dataset that includes the matrix bvecs (90 X 3 and encodes u_i for directional information), the matrix meas (50 X 91 and encodes data from 50 different voxels extracted from one scan) and a double precision value b. Since we are given 50 different voxels to run our methods with, our analysis from now on is based on the results after running each starting iterate through all 50 voxels in the matrix meas.

If we re-parametrize the 3 X 3 identity matrix into a six-dimensional vector, we get the vector [1 0 0 1 0 1]. The starting iterates used in the following tests were all multiples of the 6-dimensional identity vector, were the constant multiplied by the vector ranged from 10^{-4} to 10^4. In this case, we sample 20 starting points for our convergence plots.

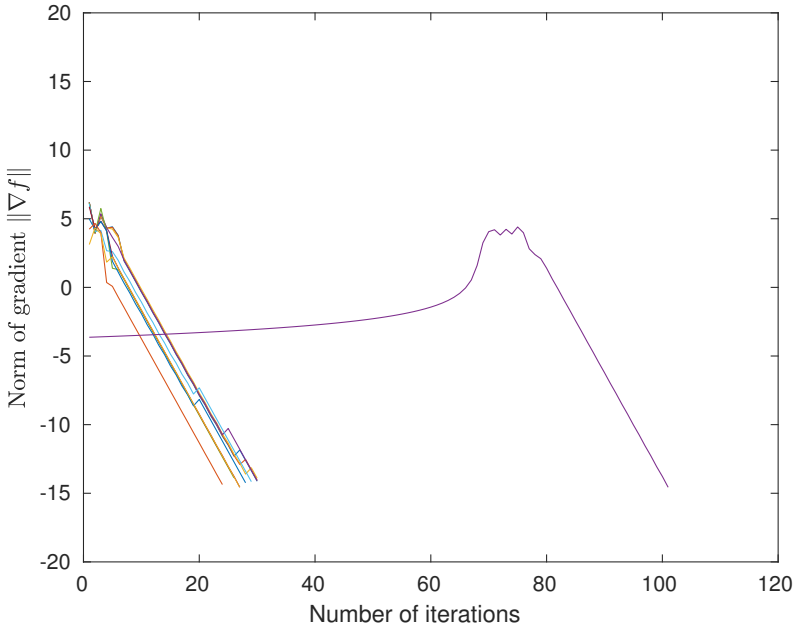

Figure 1: Convergence log-plot for the Steepest Descent method

From visually inspecting Figure 1, we can see that the convergence rate obtained here is linear, and that for some starting iterate, the number of iterations is approximately 100. The average number of iterations for the different starting values is 34.7272.

Using the same starting iterates for the LM method we get the following convergence plot.

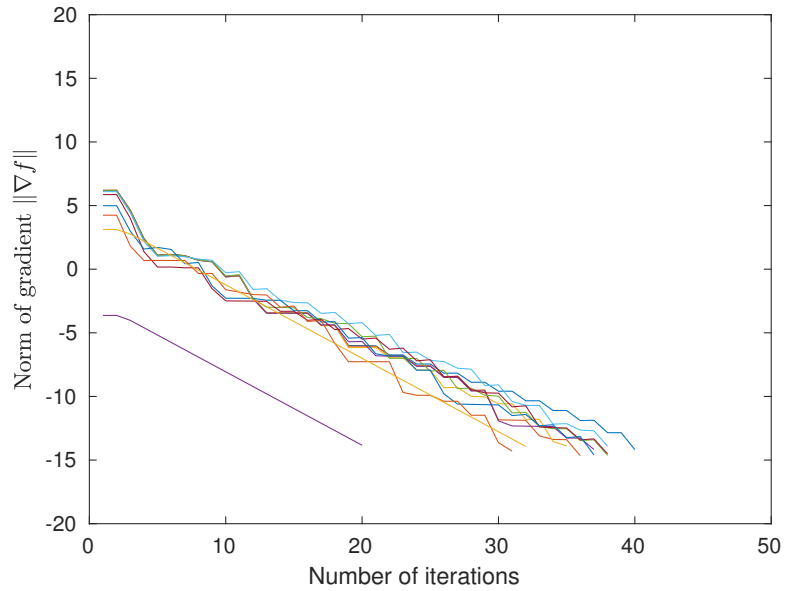

Figure 2: Convergence log-plot for the Levenberg-Marquardt method

The convergence in this case also looks linear, but the average of the number of iterations for the different starting values here is 22.8235.

We can also measure the performance of both methods in terms of the running times. The following table shows the results for the average total running time and the average time per iteration for both Steepest Descent and Levenberg-Marquardt. The average was taken for runs with the same starting iterates used for Figures 1 and 2 above.

	Average total running time	Average time per iteration
Steepest Descent	0.1864	0.007663636
Levenberg-Marquardt	0.1287	0.075336364

Table 1: Running time for both algorithms in seconds

From the table we can see that even with more processing and computations for each iteration and, because of that, a greater time spent on each iteration, the LM method is still faster than the Steepest Descent.

We can also analyze the number of calls made to the function that computes the value of f, the gradient ∇f and the Hessian approximation $J^T J$ at a given point $x \in \mathbb{R}^6$. For the Steepest Descent method the average number of calls is 290.63 and for the LM methos that number is 68.45. That difference is due to the backtracking line-search used in the Steepest Descent, which computes the function values many times until it find an appropriate step length α.

Establishing the same tolerance value for the gradient norm in both methods made the accuracy of the results be very similar for both of them. The Levenberg-Marquardt method seems to be more efficient; even though it takes more computation time per iteration, it is generally faster to converge to a possible minimizer than the Steepest Descent method. As for robustness, the tests were made for a big range of starting iterates and both algorithms seemed to perform well for all of them, with Levenberg-Marquardt being more stable when it comes to performance (number of iterations and running time stayed consistent throughout testing).