Numerical Optimization
Re-exam Handin 2

Dmitry Serykh (qwl888)

June 14, 2020

1 The Setup

For this assignment, I used Python and the `minimize` function from the `scipy.optimize`. For the minimizer, I have decided to use the BFGS optimizer, since it is the default choice for the unconstrained optimization in this library. BFGS stands for Broyden–Fletcher–Goldfarb–Shanno algorithm and belongs to the family of quasi-Newton optimizers that is a state-of-the-art alternative to Newton methods, but does not require a Hessian matrix. It can also be classified as a Line-Search and algorithm, where we first determine the direction p_k and the step length in that direction α afterwards.

1.1 Parameters

The `scipy` implementation of the BFGS algorithm has following parameters:

- *maxiter* - Maximum number of iterations to perform.
- *gtol* - Gradient norm must be less than gtol before successful termination.
- *norm* - Order of norm
- *eps* - Step size used when the gradient is approximated.

I did not use the *maxiter* parameter, since the gradient magnitude is be used to determine when to stop the iteration. For *gtol*, I used the default value of 10^{-5}, and ∞ for the *norm*. *eps* was not used, since I we have implemented a gradient for all five functions and we do not need to approximate it. For all the functions, I have used a dimensionality of two.

2 Testing

2.1 What constitutes a good testing protocol?

According to Chapter 1 in the book, good optimization algorithms should possess following properties:

- **Robustness**
- **Efficiency**
- **Accuracy**

There existing data aggregation methods include: minimum, maximum, average(mean), median, variance, standard deviation.

2.2 My testing protocol

In order to test the effectiveness of the BFGS implementation from the `scipy.optimize`, I came up with a testing protocol, where I use two measures of performance:

- The convergence plots with the number of iteration on the x-scale and the Euclidean distance between the current value of x and the optimum. The results can be seen on Figure 1. I applied the low shelf to the results, so anything would stop at 10^{-6}.
- The number of steps until the gradient magnitude reaches 10^{-5}. The results can be seen on Figure 2

I used a random starting point taken from the uniform distribution in the interval between -10 to 10 and repeated each optimization 100 times for both metrics and took the average. This was done to remove the influence of the starting position from the results of the optimization. I used the mean to minimize the effect of the outliers, which is important due to the stochastic nature of the algorithm.

3 Results

3.1 Convergence Plots

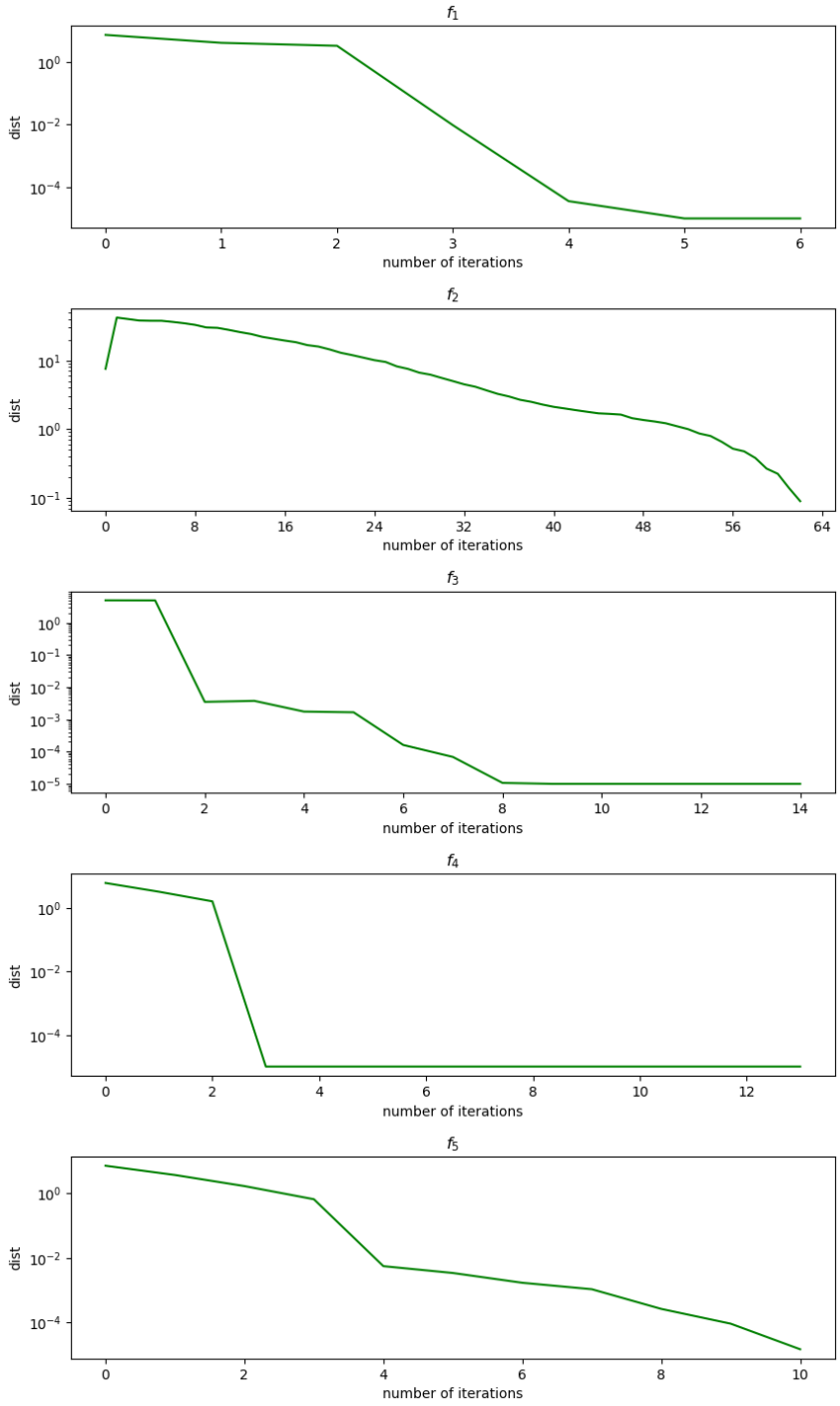

Figure 1: Convergence Plot for BFGS, median of 100 runs. Y-axis shows the Euclidean distance to the optimum of each $f_1, ..., f_7$ with random starting point

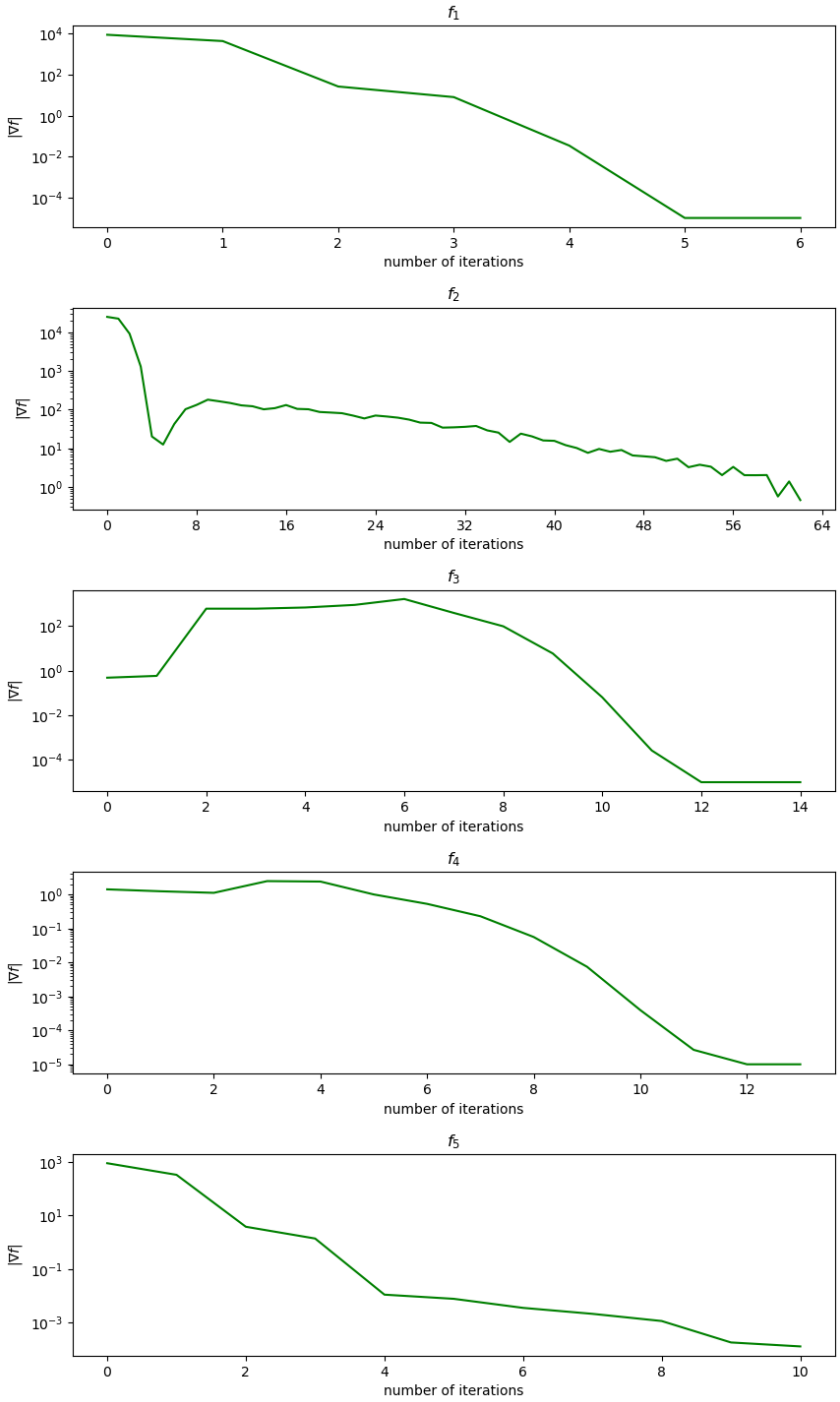

Figure 2: Convergence Plots for BFGS, median over 100 runs. Y-axis shows the norm of the gradient of each $f_1, ..., f_7$ with random starting point

3.2 Other Metrics

	f_1	f_2	f_3	f_4	f_5
efficiency	7	68	16	14	12
accuracy	$1.877e-8$	$3.864e-7$	$1.426e-13$	$6.512e-7$	$8.72e-5$

Table 1: Average number of iterations until algorithm termination ($\|\nabla f\| < 1e-5$) and final distance to the optimum for 100 random starting points in the interval $[-10, 10]$

4 Theoretical Part

4.1

4.1.1

Since $A \in \mathbb{R}^{d\times d}$ is invertible, $g(x) = Ax$ would constitute a linear map $\mathbb{R}^n \to \mathbb{R}^n$. Linear transformations preserve the ratios of lengths, therefore the inequality

$$f(\alpha(Ax) + (1-\alpha)(Ay)) \leq \alpha f(Ax) + (1-\alpha)f(Ay), \text{ for all } \alpha \in [0,1]$$

Would still hold, since we know that f is strictly convex and therefore:

$$f(\alpha(x) + (1-\alpha)(y)) \leq \alpha f(x) + (1-\alpha)f(y), \text{ for all } \alpha \in [0,1]$$

4.1.2

We start by looking at an expression x^TQx. We know that Q is symmetric positive definite, hence we can use Cholesky decomposition and obtain following: $x^TQx = x^TLL^Tx$, where L is triangular matrix with positive diagonal values, hence invertible.

Let $f(y) = y^Ty$, then $\nabla f(y) = 2y$ and $\nabla^2 f(y) = 2I$. $2I$ is PD, therefore $f(y)$ is strictly convex. Then, if we set $y = L^Tx$,we get:

$$f(y) = f(Lx) = x^TLL^Tx = x^TQx$$

x^TQx is therefore strictly convex, since $f(x)$ is convex and property from the previous subsection holds.

This point is relevant for quadratic-based functions such as f_1 and f_2, where the convexity makes the optimizer converge on the optimum.

4.2

4.2.1

For $d = 1$, the Log-Ellipsoid function could be formulated as:

$$f_3(x) = \log(\epsilon + x^2)$$

Hence, the derivatives:

$$f_3'(x) = \frac{2x}{\epsilon + x^2}$$
$$f_3''(x) = \frac{2(\epsilon - x^2)}{(\epsilon + x^2)^2}$$

For $|x| < \epsilon$, $(\epsilon - x^2) > 0$, hence $f_3''(x) > 0$, and f_3 would be strictly convex.

$$(-f_3)''(x) = \frac{2(x^2 - \epsilon)}{(\epsilon + x^2)^2}$$

Hence, for $|x| \geq \epsilon$, $-f_3''(x) \geq 0$, which means that $-f_3$ is convex, hence f_3 is concave for $|x| \geq \epsilon$.

This could potentially be problematic if the gradient-based optimizer gets its current value of $x > |\epsilon|$, then it would fail finding an optimum, because it could continue moving in that direction for an indefinite amount of time or until the number of iterations reaches the predefined limit.

5 Conclusion

Based on the results of application of the two performance metrics, the `scipy` implementation of the BFGS optimizer managed to find the minima of all case functions. It struggled most with the Rosenbrock function, due to its elongated and curved iso-level.