Numerical Optimization
Re-exam Handin 2

Dmitry Serykh (qwl888)

June 15, 2020

1 The Setup

For this assignment, I used Python, and all the relevant functions from `numpy` and `scipy` packages. In particular, I have utilized the `minimize` function from the `scipy.optimize` library. I have chosen 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, thus it does not require the Hessian matrix. Moreover, it can be classified as a Line-Search 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 used the value of 1000 for the *maxiter* parameter, but it was never used, since the gradient magnitude was used as a primary stopping criterion. 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 must have following properties:

- **Robustness**. Ability of the algorithm to find minima of vast array of different functions. Furthermore, the algorithm should be indifferent to the starting point x_0.
- **Efficiency**. A good optimization algorithm should not consume too much computational resources (and memory/storage).
- **Accuracy**. Good optimization algorithms should find the optima with high precision. Normally to some tolerance ε on the gradient magnitude or distance to the optimum.

In order to process and then visualize the gathered data from the experiments, I must use one of the existing aggregation methods. These include: min/max, average(mean), median, variance, standard deviation, mode and range. I decided to use the median and the mean. The main difference between these methods is how they manage the influence of the outliers. When using the mean, we maintain the influence of the outliers in the set, while it is excluded when using the median. If we choose to calculate the mean, we can augment it by providing the standard deviation of the dataset in order to see how much the samples stray from the mean.

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

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:

- In order to measure the robustness of the algorithm, I have used the algorithm to find the minima of all five problem functions: The Ellipsoid function(f_1), the Rosenbrock Banana function (f_2), The Log-Ellipsoid function (f_3) and the Attractive Sector functions (f_4 and f_5). To test if the algorithm is indifferent to the starting point x_0, I have used a random starting point taken from the uniform distribution in the interval between -10 to 10 for both dimensions, repeated all the experiments 100 times and took the median of the results.
- I decided to measure the efficiency of the algorithms in number of iterations until the gradient magnitude reaches 10^{-5}. It should however be noted that steps of some algorithms are more computationally expensive than others. For example the Newton step requires a computation of Hessian and therefore more expensive than a steepest-descent step. The results can be seen on Table 1.
- The accuracy of the algorithm was tested by measuring the Euclidean distance between the current point and the optima. The results can be seen on Table 1.
- 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.
- The convergence plots with the gradient magnitude can be seen on Figure 2.

3 Theoretical Part

3.1

Since $A \in \mathbb{R}^{d\times d}$ is invertible, $g(x) = Ax$ would constitute a one-to-one linear transformation $\mathbb{R}^n \to \mathbb{R}^n$. Which means that if $b, c \in \mathbb{R}^n$ and $b \neq c$ then $Ab \neq Ac$. By applying the transformation, points x and y would be moved into new positions $z = Ax$ and $w = Ay$, and inequality:

$$\begin{aligned} f(A\,(\alpha x + (1-\alpha)y) &= f(\alpha(Ax) + (1-\alpha)(Ay)) \\ &= f(\alpha z + (1-\alpha)w) \\ &< \alpha f(z) + (1-\alpha)f(w), \text{ for all } \alpha \in (0,1) \end{aligned}$$

would hold, since we stay in the convex domain of function f, which is $\mathbb{R}^n$, function f is strictly convex, and $z \neq w$.

I will show that the function $f(y) = y^T y$ is strictly convex by showing that for all $\alpha \in (0, 1)$

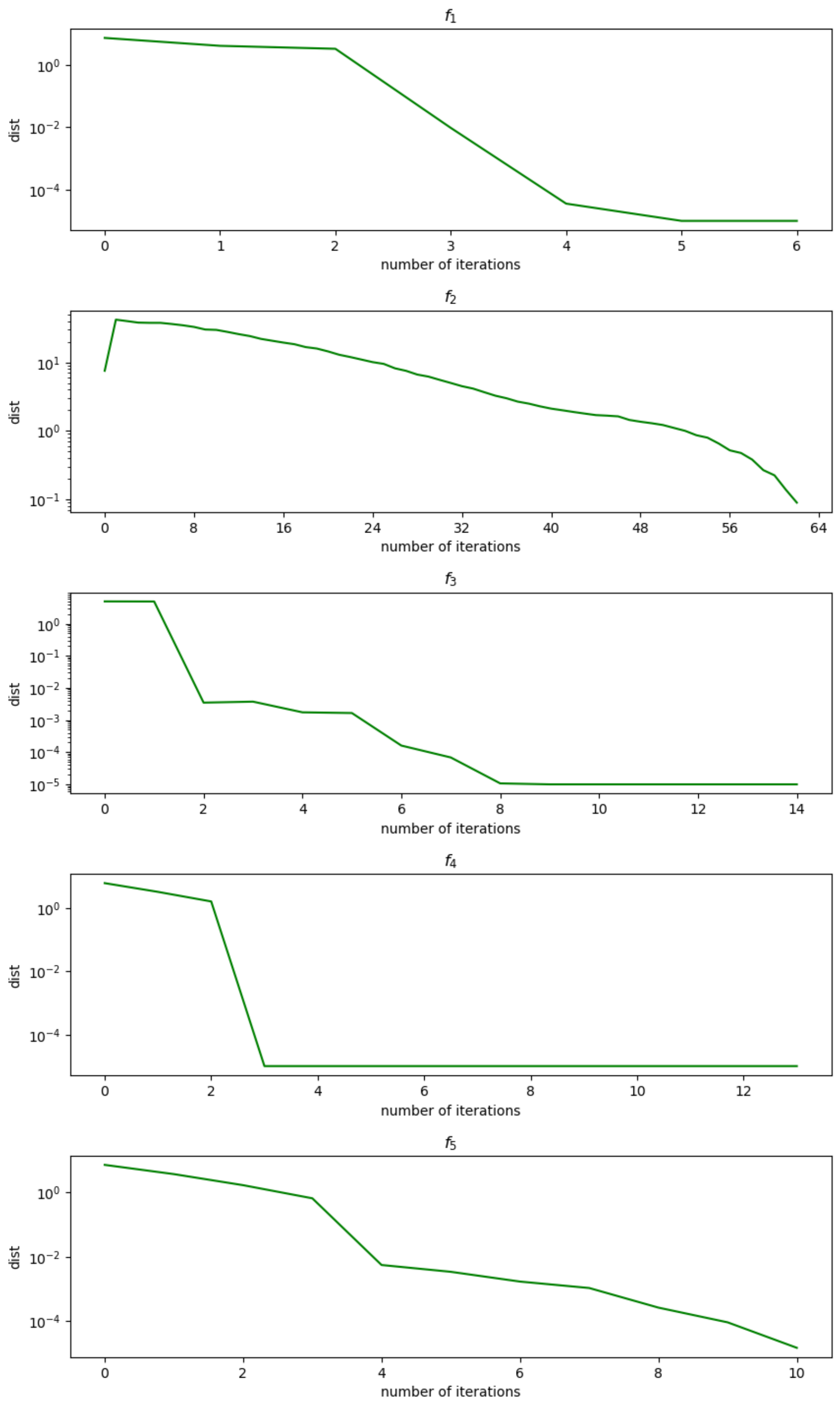

Figure 1: Convergence Plot for BFGS, median of 100 runs. Y-axis shows the Euclidean distance to the optimum of each $f_1, ..., f_7$ with random starting point

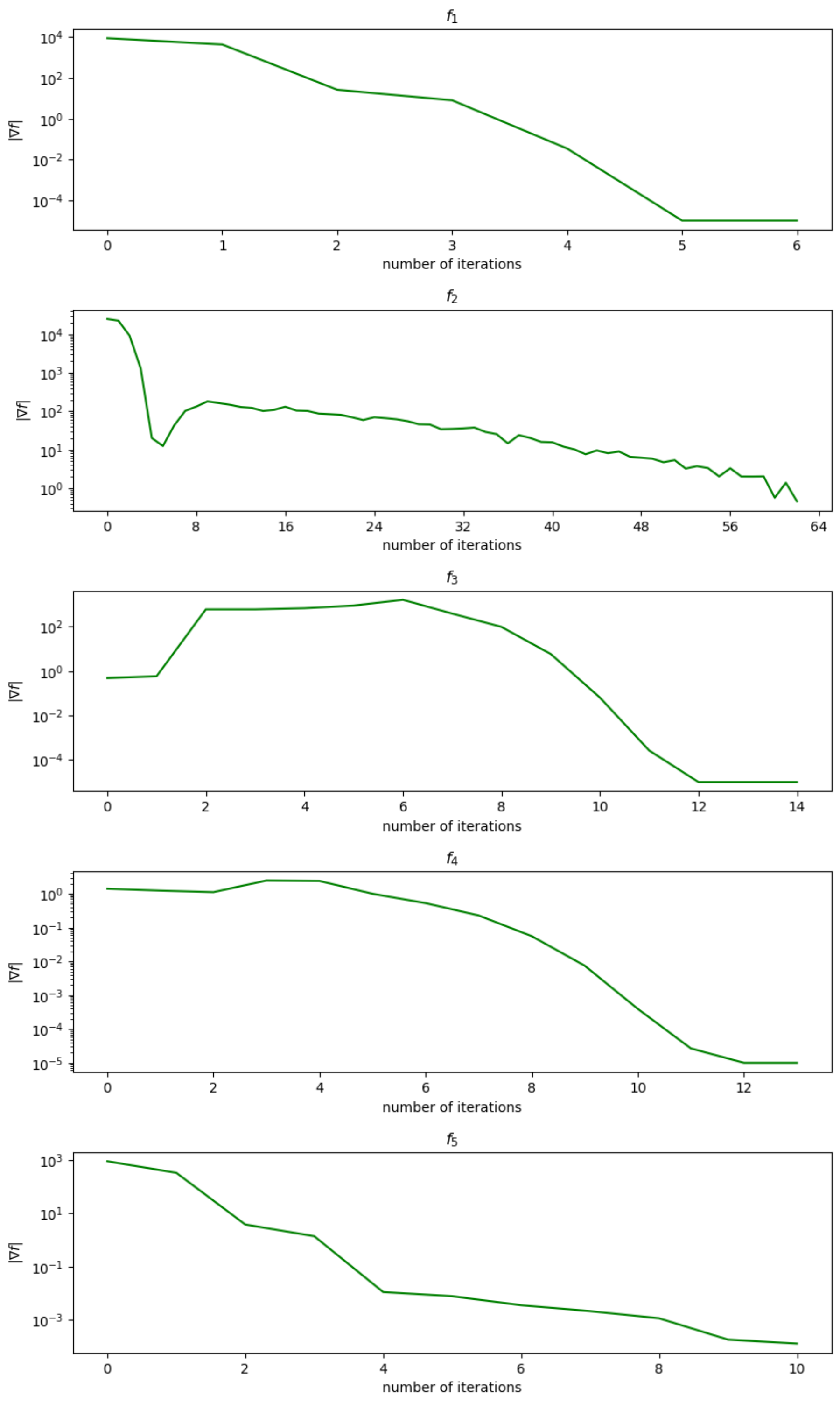

Figure 2: Convergence Plots for BFGS, median over 100 runs. Y-axis shows the norm of the gradient of each $f_1, ..., f_7$ with random starting point

and $x \neq y$:

$$\begin{aligned}
f(\alpha x + (1-\alpha)y) &< \alpha f(x) + (1-\alpha)f(y) \\
(\alpha x + (1-\alpha)y)^T(\alpha x + (1-\alpha) &< \alpha x^T x + (1-\alpha)y^T y \\
\alpha^2 x^T x + \alpha(1-\alpha)y^T x + \alpha(1-\alpha)x^T y + (1-\alpha)^2 y^T y &< \alpha x^T x + (1-\alpha)y^T y \\
(\alpha^2-\alpha)x^T x + \alpha(1-\alpha)y^T x + \alpha(1-\alpha)x^T y + ((1-\alpha)^2 - (1-\alpha))y^T y &< 0 \\
(\alpha-1)x^T x + 2(1-\alpha)x^T y + (\alpha-1)y^T y &< 0 \\
(\alpha-1)(x^T x - 2x^T y + y^T y) &< 0 \\
(\alpha-1)(x-y)^T(x-y) &< 0
\end{aligned}$$

$\alpha \in (0,1)$, hence $(\alpha - 1) < 0$. $x \neq y$, hence $(x-y) \neq 0$ and $(x-y)^T(x-y) > 0$. Therefore the LHS is always negative and f is strictly convex.

I then look at an expression $x^T Q x$. I know that Q is symmetric positive definite, hence I can use Cholesky decomposition and obtain following: $x^T Q x = x^T L L^T x$, where L is triangular matrix with positive diagonal values, hence invertible.

Then, if I set $y = L^T x$,we get:

$$f(y) = f(Lx) = x^T L L^T x = x^T Q x$$

$x^T Q x$ is therefore strictly convex, since $f(x)$ is convex and multiplication by an invertible matrix maintains the strict convexity of a function.

This result is relevant for quadratic functions such as the Ellipsoid function (f_1). The strict convexity of the function would mean that any interest point found by the algorithm would always be a global minima. Which can explain the fact that the Ellipsoid was the easiest function to find the minima for.

3.2

For $d = 1$, the Log-Ellipsoid function and the derivatives are:

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

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 until the number of iterations reaches the predefined limit.

4 Conclusion

Based on the results of the performance metrics, `scipy` implementation of the BFGS optimizer managed to find the minima of all case functions. While it struggled with the Rosenbrock function, due to its elongated and curved iso-level, I can conclude that the BFGS implementation for `scipy` is robust, effective and accurate, and therefore constitutes a good optimization algorithm for a wide variety of unconstrained problems for $d = 2$.