Numerical Optimization - Handin 1

This report covers principles of testing optimizer performance, a description of relevant experiments, paired with a subsequent evaluation and analysis of *BFGS*, *Trust Region Optimization*, and *Newton-CG*. We conclude with a discussion on the quality and outcome of our tests.

1 Testing Protocols and Performance Metrics

Testing Protocols
A definition of a good general testing protocol would be vague and abstract, so we'll focus on our specific case - testing optimizers. In this case, we consider a testing protocol as a number of experiments, each with the purpose of revealing how the optimizers behave in some specific setting. Creating a good protocol thus consists of selecting experiments that demonstrate the optimizer's behaviour in relevant settings. Relevance is determined by what the optimizer will be used for. In order to quantifiably summarize behaviour of an optimizer, we design performance metrics.

Performance Metrics
A collection of good performance metrics should summarize the system's behaviour accurately and be relevant to what the goals of the optimizer is. A metric should be interpretable, requiring a minimum of domain knowledge.

Time-to-reach-tolerance is a useful measure when testing systems that have to optimize in real time, whereas tolerance-after-N-steps can be a useful measure if function evaluations are expensive or limited, so one can choose their optimizer to maximize utility. If evaluations are cheap/free but the problem requires many of them, then time-to-reach-tolerance is more relevant metric, as it better reflects the efficiency of the optimizer. Robustness to random initializations is an important metric, as in a real world problem, one might not have an informative prior as guidance and would therefore effectively be taking random guesses.
Aggregating results leads to a more encompassing view of a systems performance, but will also obscure details, so it has to be done carefully. When reporting averages of convergence speeds on a per-problem basis, we get insights on how well the model performs on specific problem types. The bigger picture, however, is not captured by this approach. Aggregation across all problem types can help facilitate this as long as one picks varied, representative sets of problems one might encounter in the real world. Otherwise the results of the test do not reflect their "real" performance.

2 Theory

2.1 Calculating the gradient and hessian of $f_2(\mathbf{x}) = (1-x_1)^2 + 100(x_2 - x_1^2)^2$ and showing that (1,1) is a minimizer

$$\frac{\partial f_2}{\partial x_1} = -2(1-x_1) - 2x_1 \cdot 100 \cdot 2(x_2 - x_1^2) = 2x_1 - 2 - 400x_1(x_2 - x_1^2) \quad ; \quad \frac{\partial f_2}{\partial x_2} = 200x_2 - 200x_1^2$$
$$\frac{\partial^2 f_2}{\partial x_1^2} = 2 - 400x_2 + 1200x^2 \quad ; \quad \frac{\partial^2 f_2}{\partial x_2^2} = 200 \quad ; \quad \frac{\partial^2 f_2}{\partial x_1 \partial x_2} \overset{*}{=} \frac{\partial^2 f_2}{\partial x_2 \partial x_1} = -400x_1 \qquad (*) \text{ Since } f_1 \text{ is } C^2$$

Thus $\nabla f_2(x_1, x_2) = [2x_1 - 2 - 400x_1(x_2 - x_1^2) \quad 200x_2 - 200x_1^2]^\intercal$ and the hessian is:

$$H_{f_2}(x_1, x_2) = \begin{bmatrix} 2 - 400x_2 + 1200x^2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix} \quad \text{with} \quad H_{f_2}(1,1) = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$$

Observe that the gradient is zero for $x_1 = x_2 = 1$. If it also holds that the hessian is positive definite at $x_1 = x_2 = 1$, theorem 3 states that $x_1 = x_2 = 1$ is a strict local minimum. More formally $\forall \mathbf{v} \neq \mathbf{0} : \mathbf{v}^T H \mathbf{v} > 0$. We will instead equivalently show that the eigenvalues are real and strictly positive by finding the roots of the characteristic polynomial:

$$\begin{aligned} &\text{Det}(H_{f_2}(1,1) - \lambda \mathbf{I}) = 0 \iff (802 - \lambda)(200 - \lambda) - 400^2 = 0 \\ \iff &\lambda^2 - 1002\lambda + 400 = 0 \iff \lambda = 501 \pm \sqrt{250601} > 0 \qquad \square \end{aligned}$$

2.2 f_3 has a positive hessian for $|x| < \frac{1}{\sqrt{\varepsilon}}$ for $d = 1$

$$f_3(x) = \ln(1 + \varepsilon x^2) \;\; ; \;\; f_3'(x) = \frac{2\varepsilon x}{1 + \varepsilon x^2} \;\; ; \;\; f_3''(x) = \frac{2\varepsilon(1 + \varepsilon x^2) + (2\varepsilon x)^2}{(1 + \varepsilon x^2)^2}$$

$$f_3''(x) > 0 \overset{*}{\iff} 2\varepsilon(1 + \varepsilon x^2) + (2\varepsilon x)^2 > 0 \iff 2\varepsilon^2 x^2 + 2\varepsilon - 4\varepsilon^2 x^2 > 0 \iff \frac{2\varepsilon}{2\varepsilon^2} > x^2 \iff \frac{1}{\sqrt{\varepsilon}} > |x|$$

$(*)$ is obtained by multiplying both sides with the denominator which is positive.

2.3 $f_5(\mathbf{x}) = \sum_{i=1}^{d} (x_i^2)^{1 + \frac{i-1}{d-1}}$ has global minimum at $\mathbf{x} = \mathbf{0}$ but the second order sufficient conditions do not hold

Assume that some $\mathbf{x} \neq 0$ is not the global minimum. The function is defined with a sum and we observe that each term of the sum is positive since $\forall a \in \mathbb{R}, \forall x \in \mathbb{R} \backslash \{0\} : (x^2)^a > 0$. This however means that setting the non-zero coordinates of $\mathbf{x}$ to zero yields a smaller sum contradicting the assumption that our chosen $\mathbf{x}$ is a global minimum. Thus $\mathbf{x} = \mathbf{0}$ is the global minimum of f_5. The gradient of f_5 is zero at $\mathbf{x} = \mathbf{0}$ since:

$$\frac{\partial f_5}{\partial x_i}(\mathbf{x}) = 2x_i \left(1 + \frac{i-1}{d-1}\right) (x_i^2)^{\frac{i-1}{d-1}} \implies \frac{\partial f_5}{\partial x_i}(\mathbf{0}) = 0$$

Hence for the sufficient conditions to hold the hessian must be positive definite. To check this we examine the second order partial derivatives. By composition f_5 is C^2 and thus the hessian is symmetric.

$$\frac{\partial^2 f_5}{\partial x_i^2} = 2\left(1 + \frac{i-1}{d-1}\right)(x_i^2)^{\frac{i-1}{d-1}} + 4x_i^2 \left(1 + \frac{i-1}{d-1}\right)\left(\frac{i-1}{d-1}\right) x_i^{\frac{i-1}{d-1} - 1} \quad ; \quad \frac{\partial^2 f_5}{\partial x_i \partial x_j} = \frac{\partial^2 f_5}{\partial x_j \partial x_i} = 0$$

Observe that the hessian is a diagonal matrix which means the eigenvalues are the values in the diagonal. Since $\frac{\partial^2 f_5}{\partial x_i^2}(\mathbf{0}) = 0 \not> 0$ the eigenvalues of the hessian at $\mathbf{x} = \mathbf{0}$ are not strictly positive which means the hessian at this point is not positive-definite. The second order sufficient conditions are not met, yet we still have a global minimum.

3 Experiments

To test *speed of convergence*, we run each optimizer on functions $f_1, ..., f_5$ with multiple of initializations. We measure both iterations and wall-clock time until convergence. We sample 100 initial points from the multivariate gaussian $\mathcal{N}(x^*, 100 \cdot \mathbf{I}_2)$ centered around each functions respective optimum x^* . Ideally we would like to sample uniformly, but since we cannot sample uniformly from $\mathbb{R}^2$ we used the gaussian which has infinite support, and as the variance goes to infinity it approaches the uniform distribution. For each of the 100 initializations, we ran all three of the provided optimizers until convergence with tolerance $\varepsilon = 10^{-6}$ with at most 1000 iterations per run. For each step j until convergence we calculate and save $f(x_j) - f(x^*)$ (and $|x_j - x^*|$). From the 100 runs we plot only three for clarity; The run that converged in the minimal, maximal and median number of steps (or seconds, respectively). The resulting plots can be seen in figure 1. $|x_j - x^*|$ was strongly correlated with $f(x_j) - f(x^*)$, so we didn't feel a need to include plots of both.

To test *Robustness*, we, as before, sampled 100 initialization points from $\mathcal{N}(0, 100 \cdot \mathbf{I}_2)$. We then for all initializations ran all three optimizers on all problems, and construct a histogram of steps until convergence averaged over the five problems, which are given equal weight. The resulting histogram can be seen in figure 2. We wish to experimentally

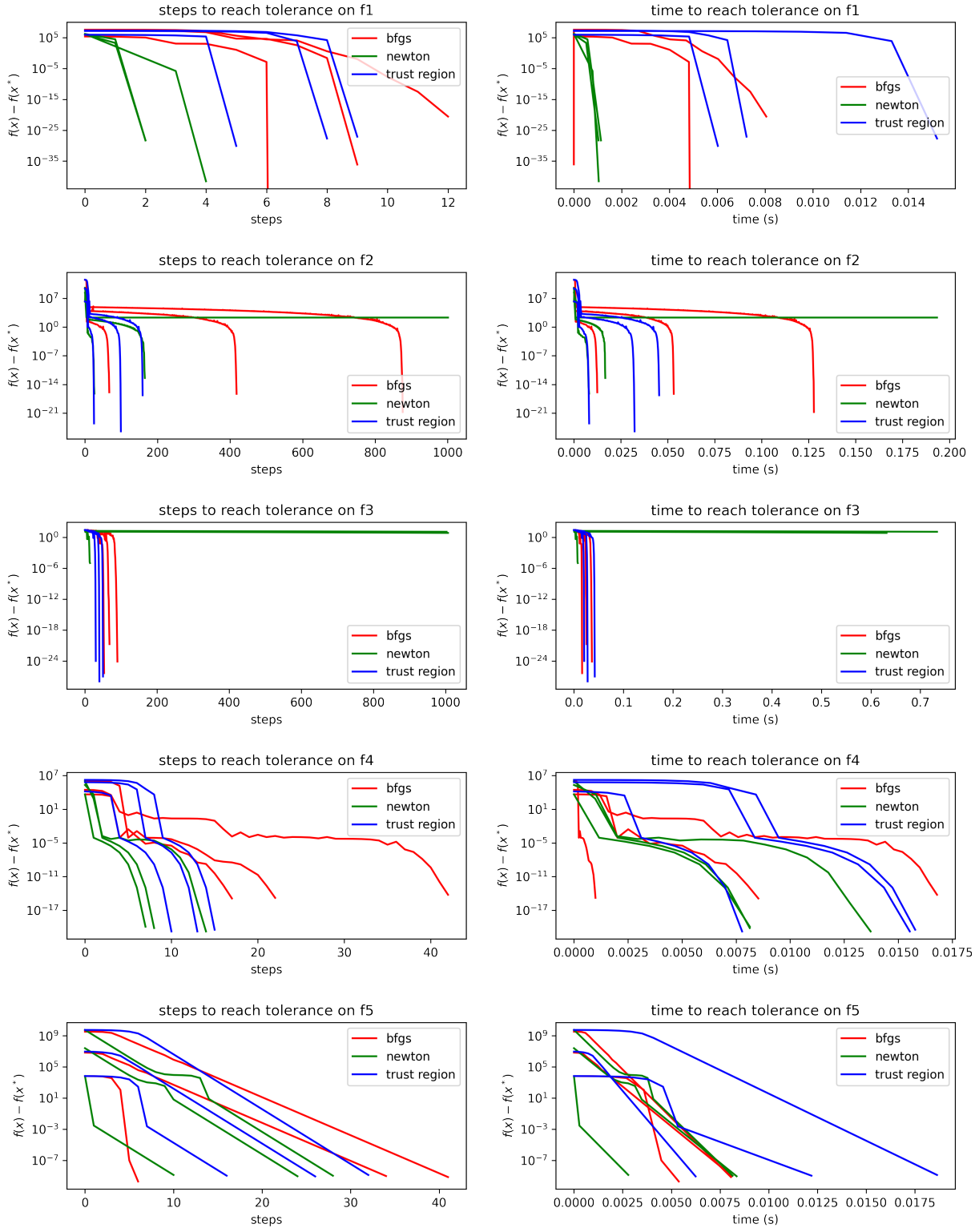

Figure 1: Left column: The distance between the function values of a candidate point and the optimum function value, per step. Right column: Same, but per time. Each optimizer is represented by three lines; the slowest, median and fastest run. We do this to give a rough sketch of the distribution of performance for different initializations.

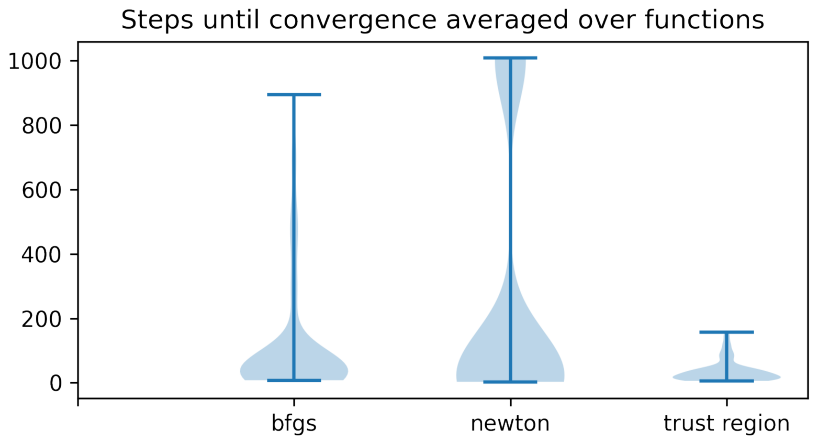

Figure 2: Violin plots (smoothed, sideways histograms, basically) of the steps taken for each algorithm. Notice the heavy tail on `newton`, suggesting it reached the maximum number of iterations (1000) quite often. `trust region` really gets to shine in terms of step-efficiency.

estimate how consistently each optimizer is able to solve a variety of problems, which is why we aggregate across them all. Another perspective of this test is that it estimates which optimizer is most step efficient, given no prior on the type of problem.
Numerical Stability of the optimizers is probed by comparing their respective behaviour if subjected to changes in their tolerance, but we found nothing significant to report.
Resource Efficiency is estimated by comparing seconds and steps until convergence, as this gives an indication of their step-efficiency and computation time.

3.1 Advantages and Limitations of the Experiments

When measuring average-across-each-function we get to see whether an optimizer performs better or worse than other optimizers on a given objective function. The five functions we average over are not representative of real world problems though - they have been selected because of various peculiarities, but all of them are "quite easy" (quoting a TA). The induced empirical ordering of optimizers might therefore be different if we tested on more difficult problems. We could have assigned non-uniform weights to the different problems, proportional to how difficult they are (e.g. give f_1 a lower weight), but we do not feel qualified enough yet to do this.
Averaging across the five functions is an attempt to summarize the strengths and weaknesses of each of the optimizers, but it also loses detail in the process. Conditioning on a specific problem type better let's one characterize the limitations of an optimizer by taking into account the challenges a given problem poses. Since we present the data in graphs, both the time-to-reach-tolerance and tolerance-after-N-steps can be read. Both can be used to assess how well an algorithm performs, however we mostly focus on time-to-reach-tolerance, as we wished to know the speed of their convergence.
When measuring different initializations, there is no good way to properly capture the behaviour of the algorithm. As $\mathbb{R}^d$ is unbounded, we cannot sample from a uniform distribution - we therefore resort to a multivariate independent gaussian with zero mean, which however favors points close to zero.
We only report results for $d = 2$. Given more pages for the report we would ideally also have liked to experiment with higher dimensionality and other parameters of the functions as well.

4 Observations

Comparing row-wise in figure (1), we note of a few things. In the case of $f1$, we have `trust region` takes few steps as opposed to `bfgs`, but spends more time doing the computations in total. The same pattern strongly holds for $f5$, and is hinted at throughout $f_{2:4}$ as well.

The same figure showcases `newton`'s struggles with f_2 and f_3, where it reaches the maximum number of steps limit. When increasing the value of max iterations to 10000 we still see around 5% of the runs reaching the limit, suggesting that it could have some runs where it might never terminate.
These results support the no free lunch theorem; which optimizer you should use depends on the problem and constraints at hand. If the problem is very costly per evaluation, you want to use an optimizer which is step efficient. If consistent convergence is important, choose something like `trust region`, which has been shown to be reliable.

Figure (2) shows an aggregation over all functions per optimizer. As we mention above, a heavy tail on `newton` can be seen, which agrees with the observation that it reaches the maximum iteration limit a significant amount of times. This happens consistently on f_2 and f_3, both of which are very flat in parts of the problem space.
`Trust Region` is a lot more step-efficient than the other two optimizers - it always uses less than 200 steps, which must mean that it better utilizes the available information (at least in this class of functions). In general though, taking a look at figure 1 shows that this comes at a cost of slower wall-clock time until convergence, so more calculations are being done in the background. If we were to pick specific optimizers for each function, we see that on average, `newton` is better on $(f1, f4, f5)$ and `trust region` is better on $(f2, f3)$ terms of steps taken to convergence. If optimizing for time until convergence, we see that `newton` on average best on $(f1, f5)$, `bfgs` and `trust region` tie 1st place on $(f3, f4)$ and `trust region` is best on $f2$. This again underlines the fact that optimizers themselves are optimized for specific problem types.
If we want to minimize function evaluations, using `trust region` is the best in terms of our results, as the others had significantly larger upper bounds on steps taken.

Across all optimizers and problems, varying the tolerance gives no noteworthy differences - the algorithms (when they converge) manage to do so with "high" accuracy.

4.1 Theoretical Speculations

As mentioned above some of the optimizers either converge really slowly or never converge at all for some initializations on f_2. f_2 has a banana shaped valley and in the valley at $\mathbf{x} = (1, 1)$ it has a unique minimum. By plotting contours of the gradient we observed that the gradient is also very close to zero in this valley and we suspect that this flatness can cause trouble for *Newton-CG* and *BFGS*.
On f_3 we observed in figure 1 that *Newton-CG* sometimes fails to converge within 1000 iterations. We suspect this may be due to the fact that f_3 is not convex on all of its domain. As shown in section 2.2 the hessian of f_3 is positive for $|x| < \frac{1}{\sqrt{\varepsilon}}$ which means f_3 is convex within this subset of its domain. If this result generalizes such that there for all d is a region around 0 where f_3 is convex. We suspect that *Newton-CG* may succeed for initializations within this region but have trouble if initialized outside this convex region.
Looking at the performances of the optimizers on f_5 we see they all struggle somewhat equally. None of them seem to fail to converge and they all seem to exhibit a somewhat Q-linear convergence (this is speculation as it cannot be shown empirically). In section 2.3 we have shown that the global minimum of f_5 does not satisfy the sufficient conditions of theorem 3 (lecture notes). We speculate that the sufficient conditions are somehow used by the algorithms to find a minimum.

5 Conclusion

We find that our tests, given the set of problems, give a reliable but rough sketch of the optimizers' characteristics. A broader problem class would have been desirable, as all of the given problems had unique solutions that were easy to find. With more local minima, we could compare how often they find the lowest one or plot their candidate trajectories to try and determine, in a very literal sense, their pitfalls.
In terms of robustness, we conclude that `trust region` is superior to the others, and consistently performs well on all problems. There's a lot of variability in `bfgs`'s performance, but it still manages to find a solution, as opposed to `newton`. Given our results, `newton` can not be considered robust nor accurate. The times `newton` manages to converge though, it often does so faster than the other optimizers, suggesting that it is best suited for cheap-evaluation problems within its capabilities.