HIGHER SCHOOL OF ECONOMICS

OPTIMIZATION METHODS

DMITRY K. ALEKSANDROVICH

Practical Home Work 2

Author:
Alex Kosmachev

Student Number:
AMI084

March 31, 2017

Contents

Resumo. *This is homework for my Optimization Methods class, 3^{rd} year. Current assignment is the continuation of the previous one.*

We are going to discuss advanced gradient optimization methods and implement them in real code.

Also we are going to make some experiments and discuss their results.

And finally, if I will have any power left, I'm going to do bonus tasks.

HW is not dependent on the environment.

Abstract. ***Keywords****: Optimization Methods, Linear Algebra, Python, NumPy.*

1 Introduction

Today we are going to talk about advanced methods of unconditional optimization.

The most important concepts that we are going to use now, are:

- The conjugate gradient method.
- Quasi-Newtonian methods.
- Truncated Newton method.
- Method L-BFGS.
- Logistical regression.
- Boosting sequence.

My work is going to be structured as following:

- **Introduction section** [1] - The description of this work structure.
- **Experiments section** [2]- Here, the results of conducted experiments can be found.
- **Bonus experiments section** [3] - Additional task for special bonus points.

This task was much easier that previous one.(Probably) However, these days I'm so tired that I can't even move, so I finished only one bonus experiment. Of course, the source code passes all test, bonus ones including.

2 Experiments

Ok, so now we enter the most interesting(and the hardest) domain of this homework. Here our task is to reproduce(or create new) experiments, and record results.

I'm going to structure my answers by the following pattern:

- **Task description** - State, what kind of experiment it's going to be.
- **Implementation details** - Describe, what I did, and what conditions were met.
- **Answers to control questions** - If there were any special questions, their answers are going to appear here.

2.1 Experiment: Dependence of the number of iterations of the gradient descent on the condition number and the dimension of the space

This time we need to investigate how the number of iterations(that are required for gradient to converge) depend on the following two parameters:

- **Condition number($k \geq 1$) of optimized function**.
- **Space dimension(n) of optimized variables**.

To do this, we are going to generate random quadratic problem(for given n and k) of size n from the condition number k and run on it conjugate gradient descent method with a fixed-required accuracy. However, because our main task is to compare results of this method to the results from previous method, I also will do the same thing for normal gradient descent.

While doing it, we are going to measure number of iterations $T(n, k)$, that were required to make up the convergence of the method.

Then comes another one.

Let's fix some value of our dimension n. After iterate through different condition numbers k on the grid and draw a graph of $T(k, n)$ against k. Because every time a quadratic problem is randomly generated, repeat the experiment several times. As a result, for a fixed value of n we are going to have a whole family of curves $T(k, n)$ dependent on k. Draw these curves using the same color.

When it's done, increase the value of n to n' and repeat. We are going to get new family of curves $T(n', k)$. Draw them using the same color, but different from the previous one.

Repeat increasing n couple of times(don't forget to change colors). And then do it for normal gradient descent method.

Note, that the best way to increase values of dimension n, is to make iterations based on a logarithmic scale (e.g., $n = 10$, $n = 100, \ldots$).

2.1.1 Implementation Details

So, basically, what we are going to do here, is to:

- Generate random sparse diagonal matrix $M_{n \times n}$, where each element on diagonal is number from $[1, k] : min(a) = 1 \wedge max(a) = k$.
- Generate random vertex $v_{n \times 1}$.
- Using gradient descent we'll optimize our quadratic function.
- Fix number of iterations, it took out gradient to converge.
- Repeat everything for different k.
- Repeat everything for different n.
- Repeat everything for different method.

All lines with the same color, belong to one n, and colors are spread the following:

- **magenta** - $n = 10$.
- **cyan** - $n = 100$.
- **black** - $n = 1000$.

After everything is done, we are going to draw graph with acquired results, and you can see it below:

This was the graph for normal gradient descent.

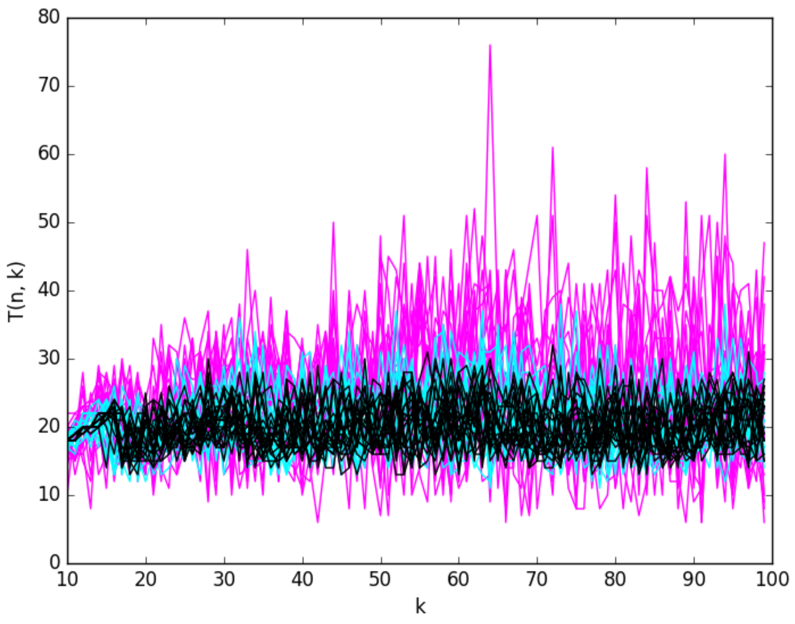

Figure 1: Y-axis represents $T(k, n)$. X-axis represents k. Color represents number of iterations.

And this is the result for conjugate gradient descent.

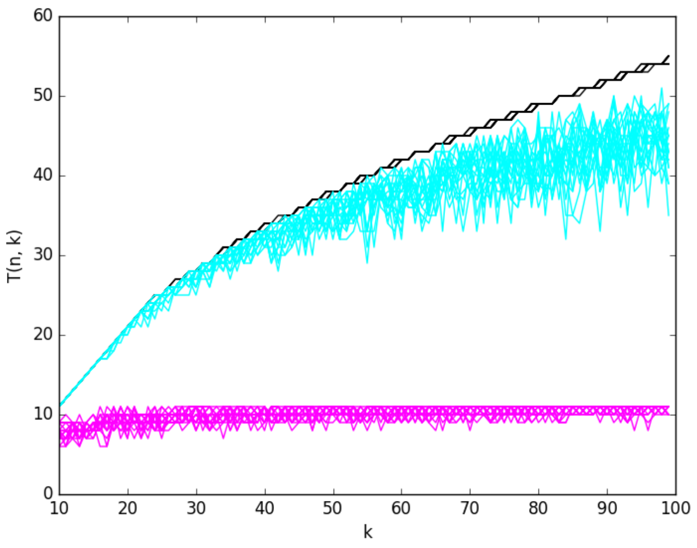

Figure 2: Y-axis represents $T(k, n)$. X-axis represents k. Color represents number of iterations.

Not forget that:

- $T(k, n)$ - number of iterations required for fixed k and n for gradient to converge.
- k - condition number of function.
- n - number of iterations.

2.1.2 Control Question

Question 1. *What conclusions can be drawn from the resulting images?*

Solution. There are couple of things, that we can understand, by looking at these images. Since we already discussed important observations in previous homework, let's talk about differences in methods results.

- First of all, it's important to notice, that normal gradient descent is less productive that our new method. For almost every condition number of function and space dimension, we can see, that new method is faster(in terms of number of iterations) and much more stable.
- Then, comes another interesting fact. As in normal method, in our new method, when condition number number increases, number of iterations increases. Therefore I must highlight the fact, that $\forall n \rightarrow$ that gradient will converge in $\leq n$ steps, and we can clearly see it on the image with conjugate gradient descent method.
- Finally, there is one more thing to mention. When growing, our graphs go almost linearly, and the way they grow is determined by the dimension size(n). That means, that at the beginning they are all go almost equally, but then after some time, the ones with smaller n start to slow down their growth.

□

2.2 Experiment: Choosing history size in L-BFGS method

During this experiment, we need to find out, how history size affects performance of L-BFGS method.

For test function, I'm going to use **gisette** dataset from *LIBSVM2* site.

First of all, we are going to to find out how the memory and time complexity of L-BFGS method depends on history size(l) and space dimension(n).

Then for different l - we are going to draw these graphs:

- Dependence of relative square gradient norm $\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2}$(in logarithmic scale) on the number of iterations.
- Dependence of relative square gradient norm $\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2}$(in logarithmic scale) against actual time of the method computation.

Please note, that the drawings for different history sizes, are drawn on the same graph, and could be distinguished by color:

- $l = 0$ - *cyan.*
- $l = 1$ - *blue.*
- $l = 5$ - *green.*
- $l = 10$ - *black.*
- $l = 50$ - *magenta.*
- $l = 100$ - *red.*

2.2.1 Implementation Details

As regularization factor we took: $\lambda = \frac{1}{m}$.

The parameters of both methods used default settings.

The starting point $x_0 = 0$.

Well, before we proceed to graphs, let's do some calculations.

- **Memory complexity** - to find out, what memory complexity is, we need to recollect what are we storing and how many times. If one will look at the code, he may find out, that memory consumption doesn't really depend on iterations, but on the size of history $\rightarrow$ after some time, there will definitely be window, when we store l vectors. Each size of vector is n. Even though I store more that one vector in dequeue, for us it doesn't matter. So the final memory complexity = $O(l \cdot n)$
- **Time complexity** - on each iteration here we do $c_{onst} \cdot l$ on each vector $\rightarrow$ that means, if we have n vertexes in total, then our final time complexity = $O(l \cdot n)$

And here are the graphs that we are going to get.

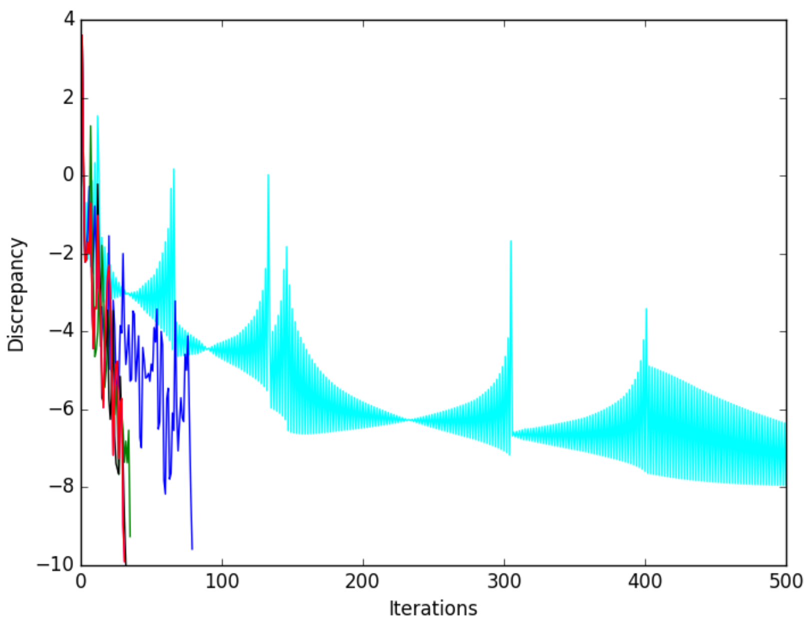

Figure 3: Dependence of relative square gradient norm $\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2}$ (in logarithmic scale) on the number of iterations.

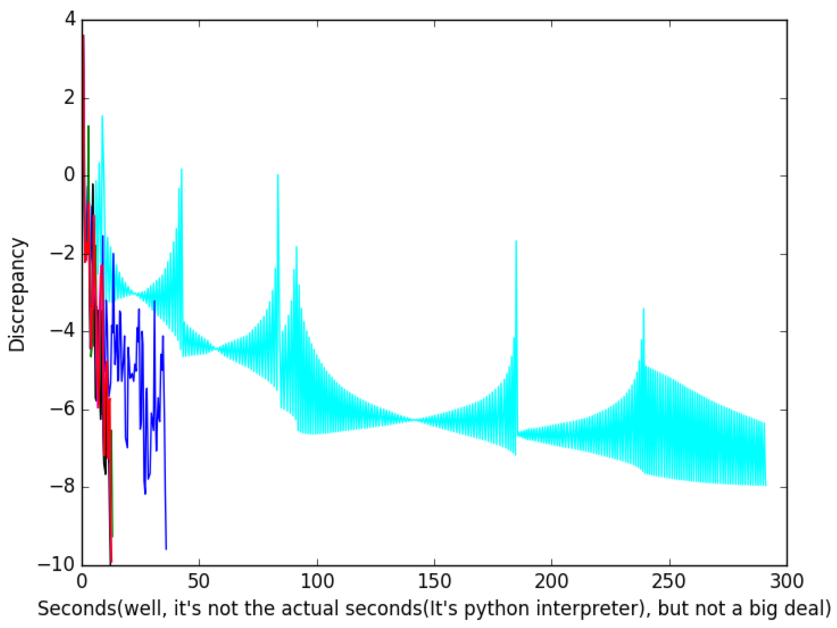

Figure 4: Dependence of relative square gradient norm $\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2}$ (in logarithmic scale) against actual time of the method computation.

2.2.2 Control Question

Question 2. *What conclusions can be drawn?*

Solution. This time, we can see some obvious facts:

- This algorithm proves that it's really good. We can see that the results, $\forall l > 0$ are really good. There are no that strange movements with optimization, because gradient converges pretty straight forward there.
- Then it's important to mention, that the more history size there is, the better our algorithm preforms. That means, that if our history size is 0, than, our method behaves like normal gradient method, which is not really good.
- Of course, there is an other side of the medal everywhere. The more we increase l - our history size, the less productive it becomes.

$\square$

2.3 Experiment: Comparison of methods using logistic regression

This time we need to compare the truncated Newton method, the L-BFGS and gradient descent methods using a logistic regression. We are going to compare them on these datasets from *LIBSVM2* site:

- **w8a**.
- **gisette**.
- **real-sim**.
- **news20.binary**.
- **rcv1.binary**.

For each dataset we are going to build these graphs:

- Dependence of function values on the number of iterations.
- Dependence of function values on the actual time of the method computation.
- Dependence of relative square gradient norm $\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2}$ (in logarithmic scale) against actual time of the method computation.

Please note, that all methods for each dataset appear on one graph. You can distinguish them by colors.

- **Hessian free Newton** - *cyan*.
- **L-BFGS** - *magenta*.
- **Gradient Descent** - *black*.

Not forget to mention, I was using optimized oracle here.

2.3.1 Implementation Details

As regularization factor we took: $\lambda = \frac{1}{m}$.

The parameters of both methods used default settings.

The starting point $x_0 = 0$.

And without further ado, let's move to graphs:

First we start with *wa8* dataset.

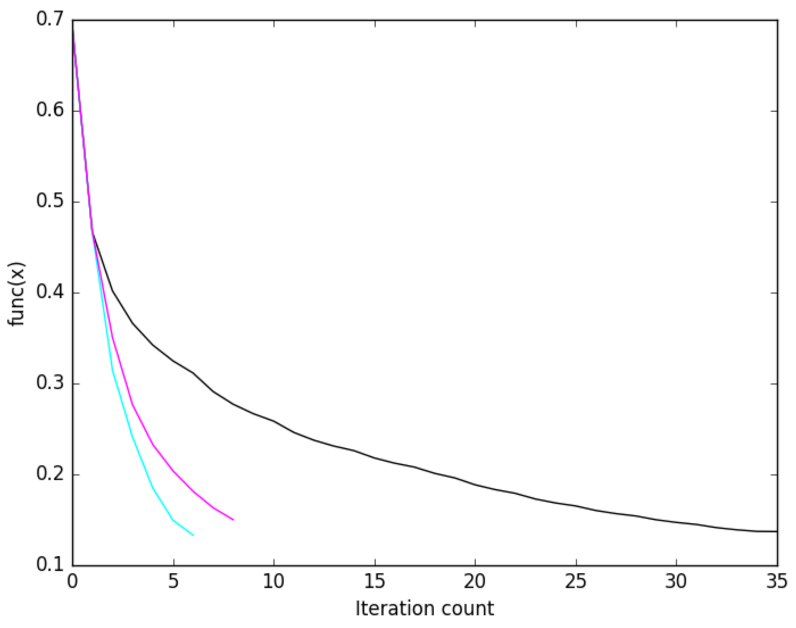

Figure 5: Dependence of function values against the numbers of iterations.

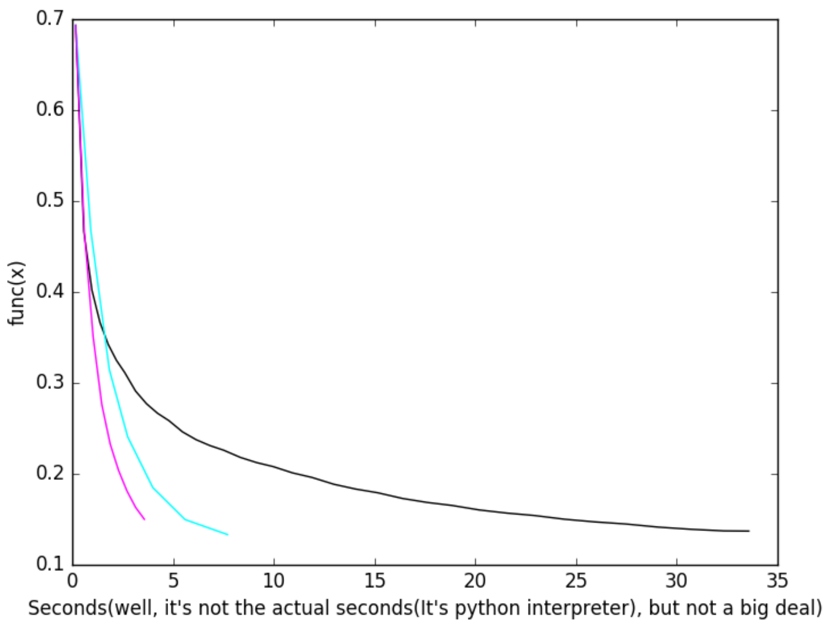

Figure 6: Dependence of function values against the actual time of the method computation

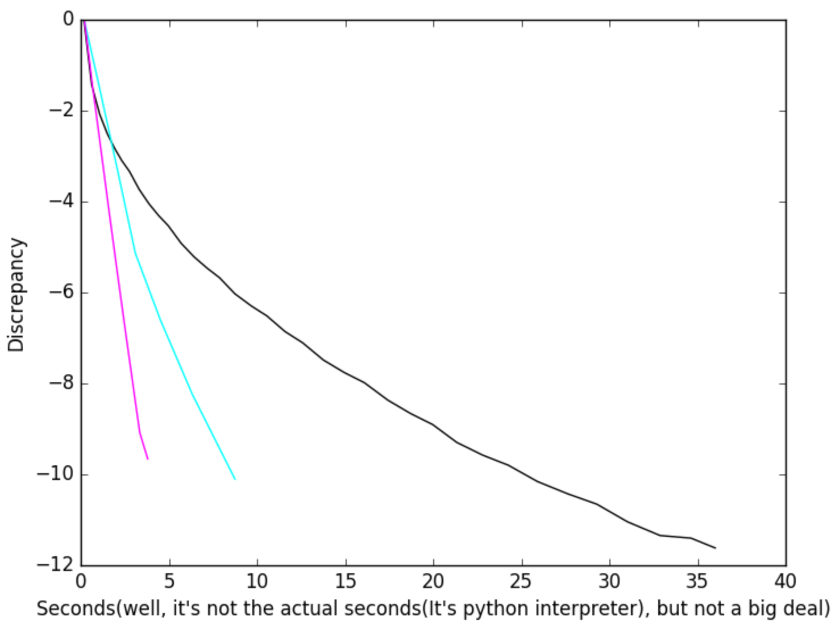

Figure 7: Dependence of relative square gradient norm $\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2}$ (in logarithmic scale) against actual time of the method computation.

Then comes *gisette* dataset.

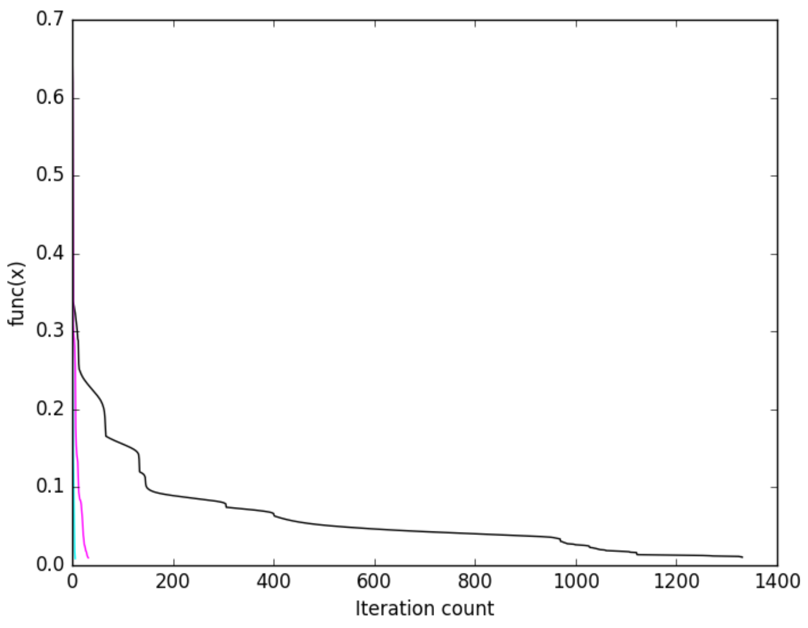

Figure 8: Dependence of function values against the numbers of iterations.

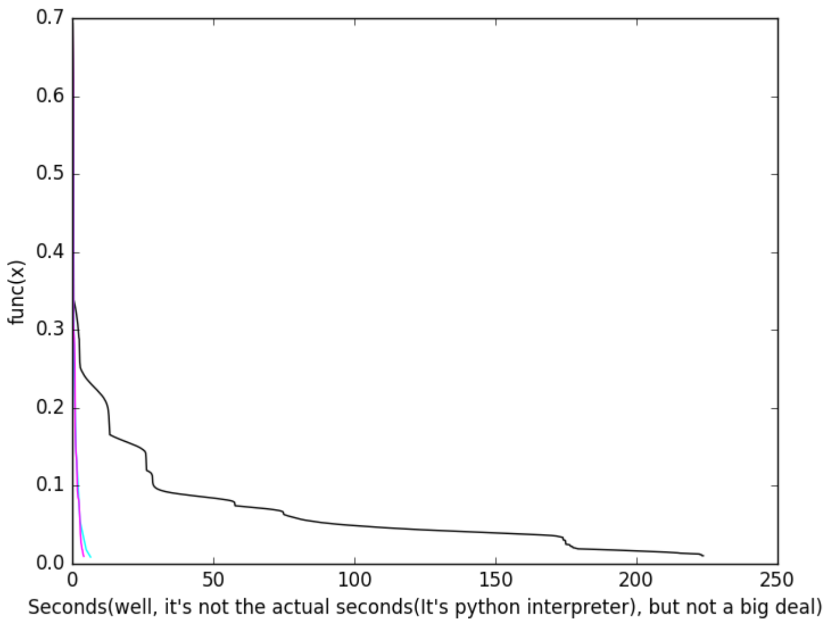

Figure 9: Dependence of function values against the actual time of the method computation

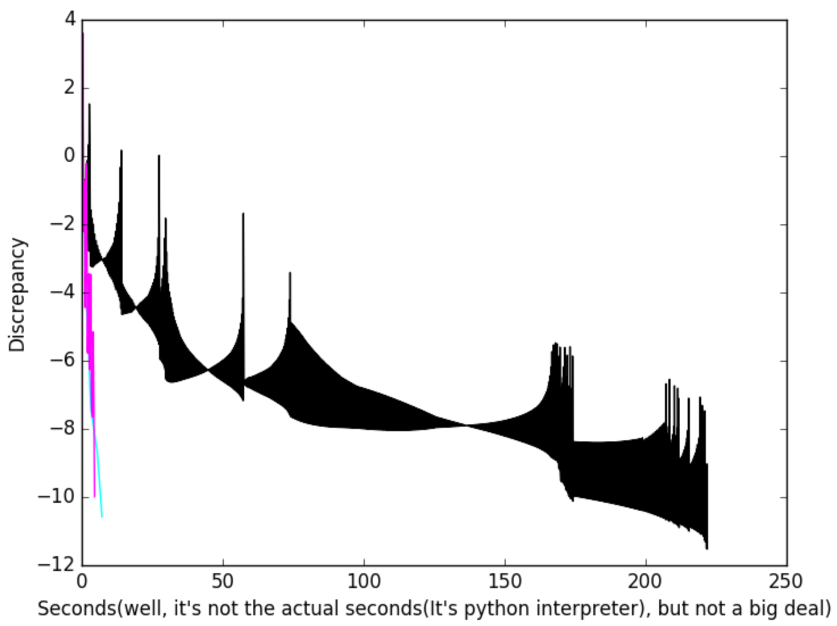

Figure 10: Dependence of relative square gradient norm $\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2}$(in logarithmic scale) against actual time of the method computation.

After is *real-sim* dataset.

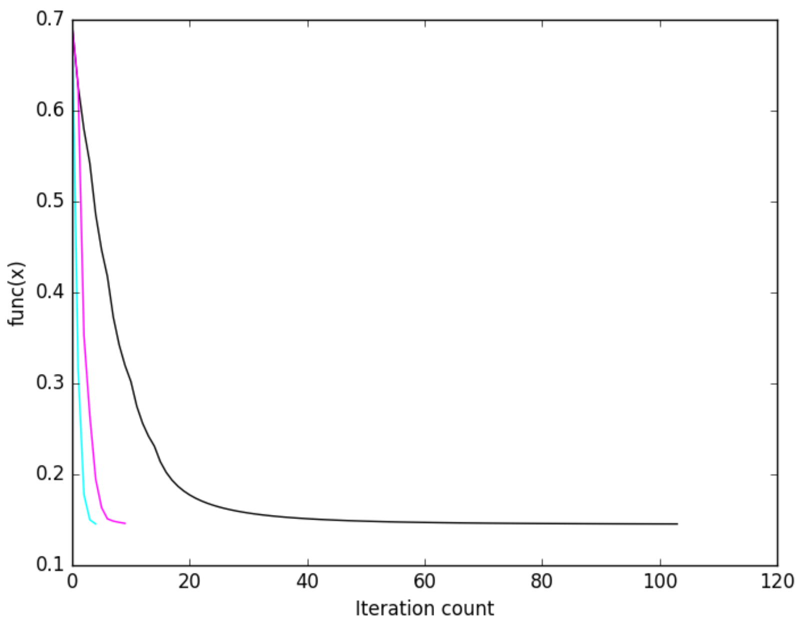

Figure 11: Dependence of function values against the numbers of iterations.

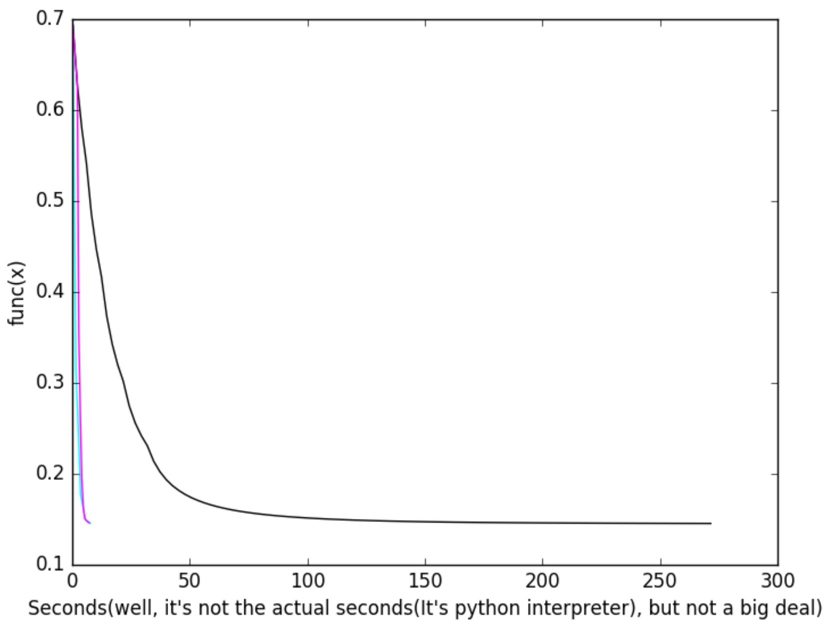

Figure 12: Dependence of function values against the actual time of the method computation

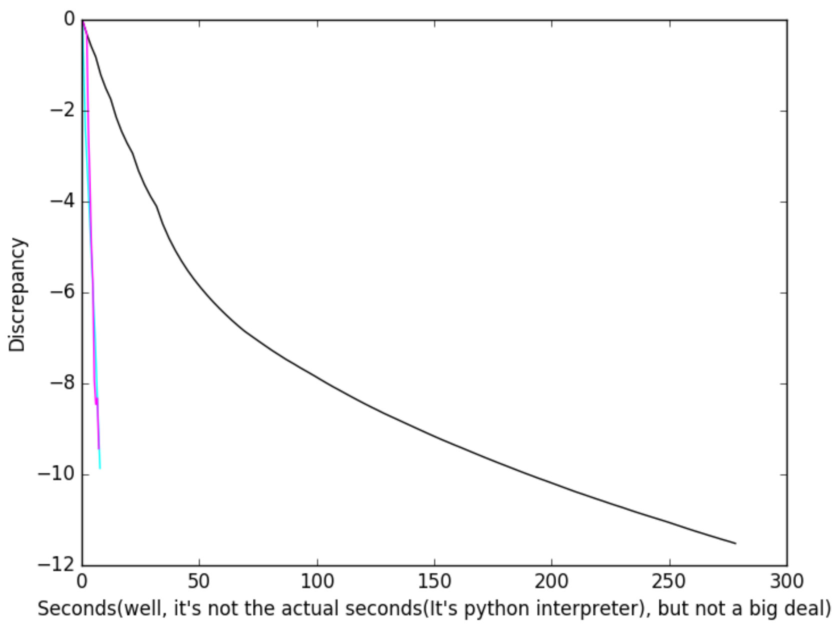

Figure 13: Dependence of relative square gradient norm $\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2}$(in logarithmic scale) against actual time of the method computation.

This one is *news20.binary* dataset.

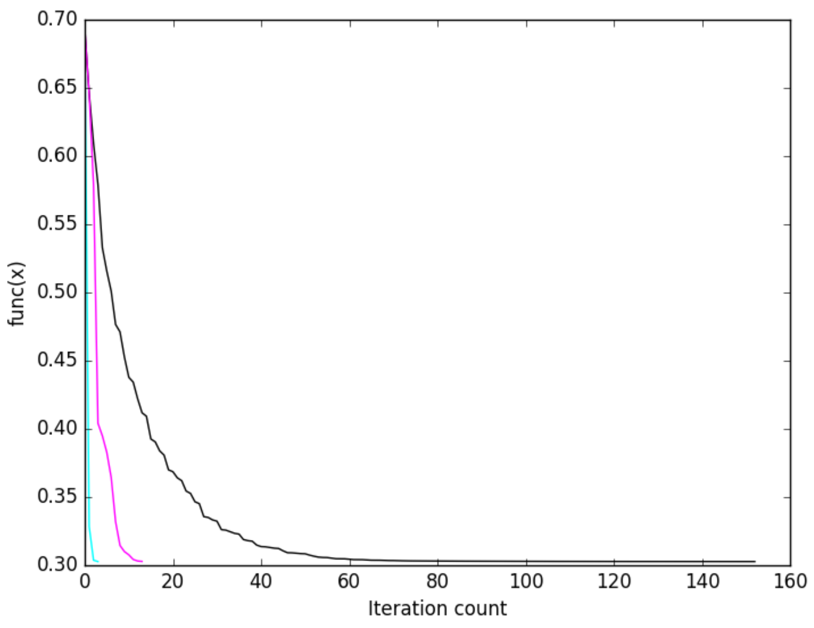

Figure 14: Dependence of function values against the numbers of iterations.

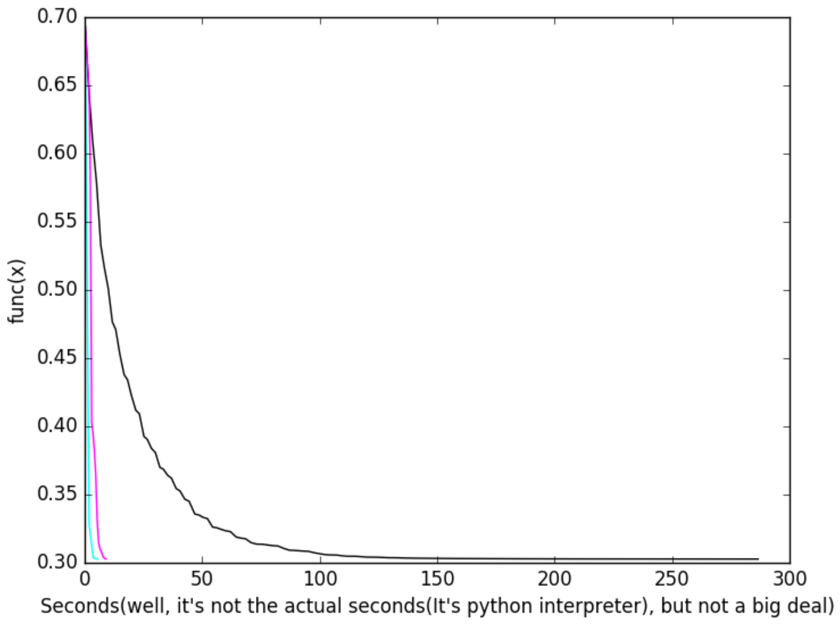

Figure 15: Dependence of function values against the actual time of the method computation

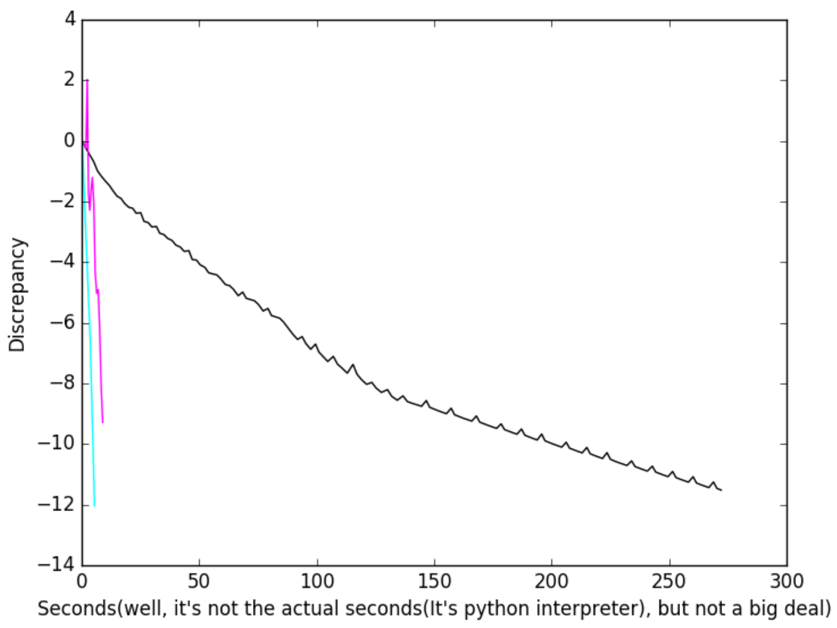

Figure 16: Dependence of relative square gradient norm $\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2}$(in logarithmic scale) against actual time of the method computation.

And finally come *rcv1.binary* dataset.

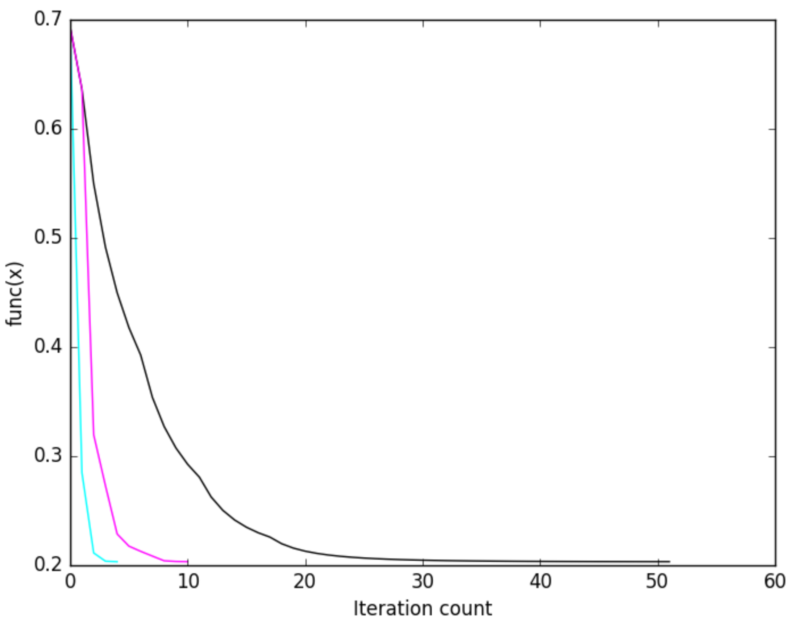

Figure 17: Dependence of function values against the numbers of iterations.

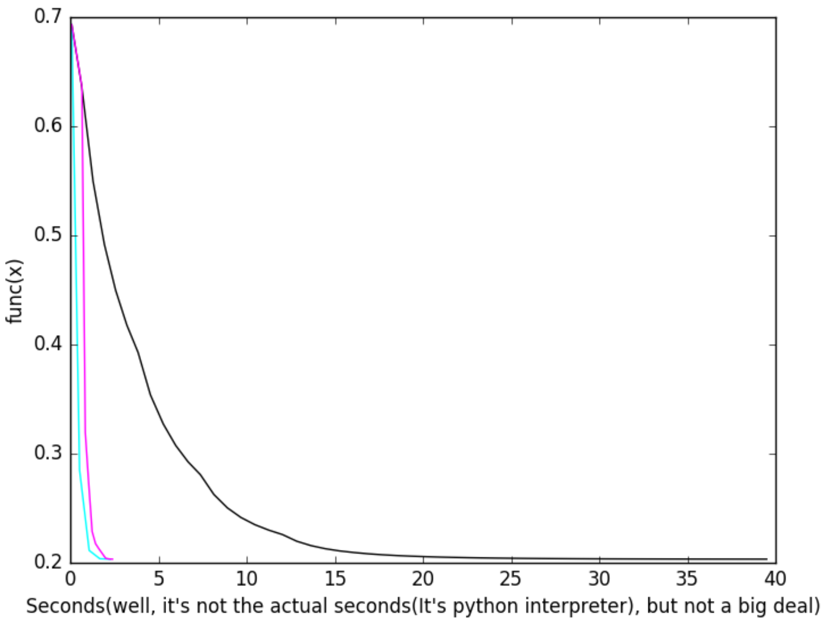

Figure 18: Dependence of function values against the actual time of the method computation

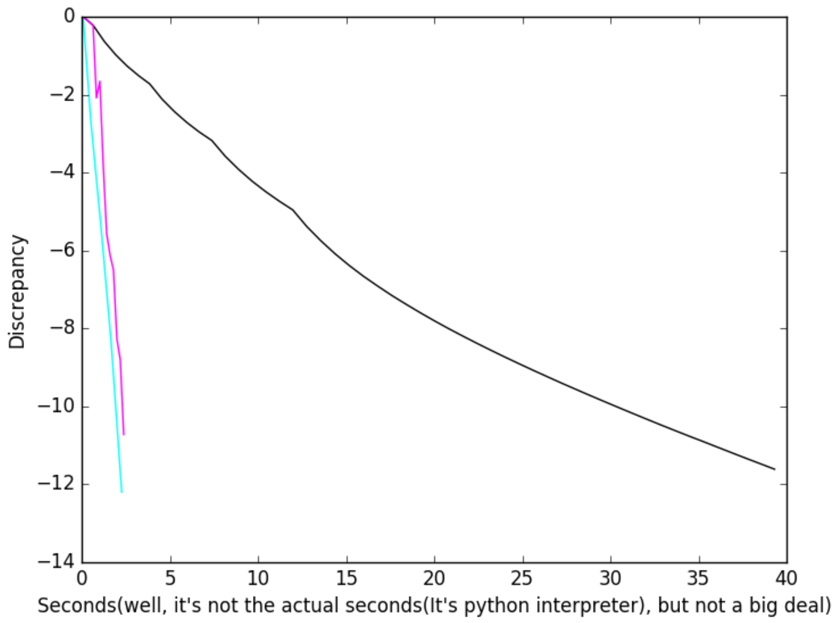

Figure 19: Dependence of relative square gradient norm $\frac{\|\nabla f(x_k)\|_2^2}{\|\nabla f(x_0)\|_2^2}$(in logarithmic scale) against actual time of the method computation.

2.3.2 Control Question

Question 3. *What conclusions can be drawn based on the results of this experiment?*

Solution. There are some interesting facts, that we can see for the results, that we acquired:

- Even though there were different datasets here, the results on all of them could be described similarly. The Hessian free Newtons method, makes precise, long iterations, and the L-BFGS method makes a little bit more iterations, but they are more lightweight.
- Also it's obvious that new and more complex methods are better than simple gradient descent.
- We can see, that actually, there is no real difference, in using either L-BFGS method either truncated Newtons method on big datasets.

□

Question 4. *Which method is better and when?*

Solution. From the pictures we can see, that truncated Newtons method is the best to use, everywhere except the *w8a* dataset. There, by small means the L-BFGS method wins. While it's true that, in big datasets these both methods are good, my opinion is, that in small datasets it's preferable to use the L-BFGS method.

□

3 Bonus Experiments

3.1 Experiment: Comparison of conjugate gradients method and L-BFGS on quadratic function

Here we need to compare conjugate gradient method and the method of L-BFGS on strictly quadratic convex function:

$$f(x) = \frac{1}{2}x^T Ax - b^T x, where\, x \in \mathbb{R}^n \land A \in \mathbb{S}^n_{++} \land b \in \mathbb{R}^n$$

We are going to follow this plan:

- Generate a couple of random quadratic functions.
- Run each method on them, from the same starting point.
- Build graphs convergence in terms of the Euclidean norm of the discrepancy $r_k = Ax_k - b$ (in logarithmic scale) versus iteration number.
- Repeat everything for different l - history size.

Note, that each for each function, we draw both methods on one plot. One can distinguish them by different lines - one is squares(L-GBFS with magenta color), the other - rombes(conjugate gradient with cyan color)(Why? I don't like it this way).

3.1.1 Implementation Details

Ok, so we first start with creating two random matrices, two random vectors b and two random starting points.

The parameters of both methods use default settings.

Not forget to mention, that I'm going to experiment with following l values:

- $l = 0$.
- $l = 1$.
- $l = 5$.
- $l = 10$.
- $l = 50$.
- $l = 100$.

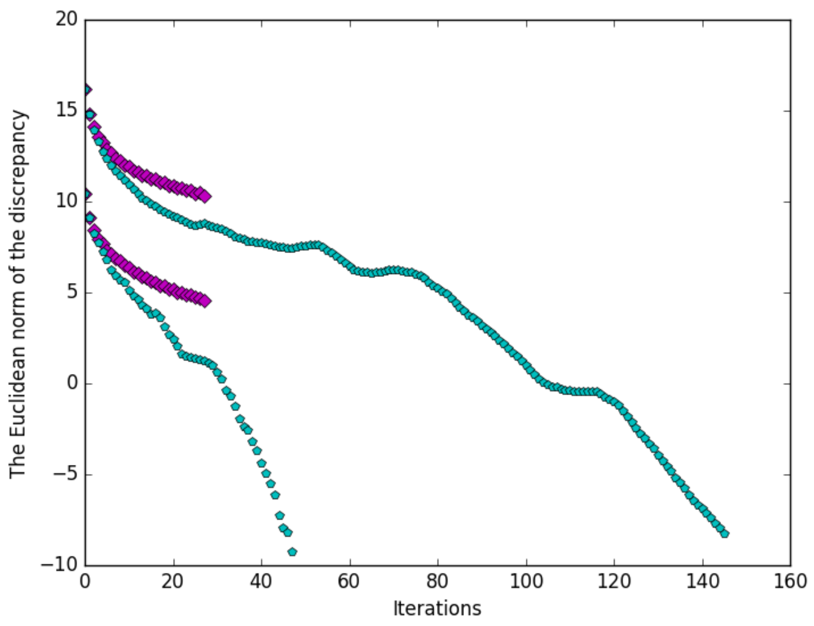

Figure 20: Dependence of Euclidean norm of the discrepancy $r_k = Ax_k - b$ (in logarithmic scale) against iteration number

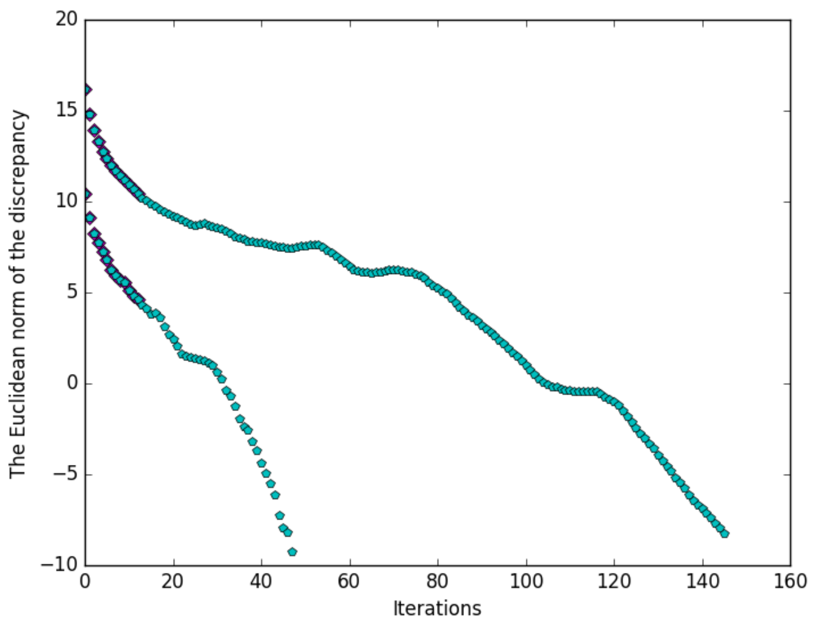

Figure 21: Dependence of Euclidean norm of the discrepancy $r_k = Ax_k - b$ (in logarithmic scale) against iteration number

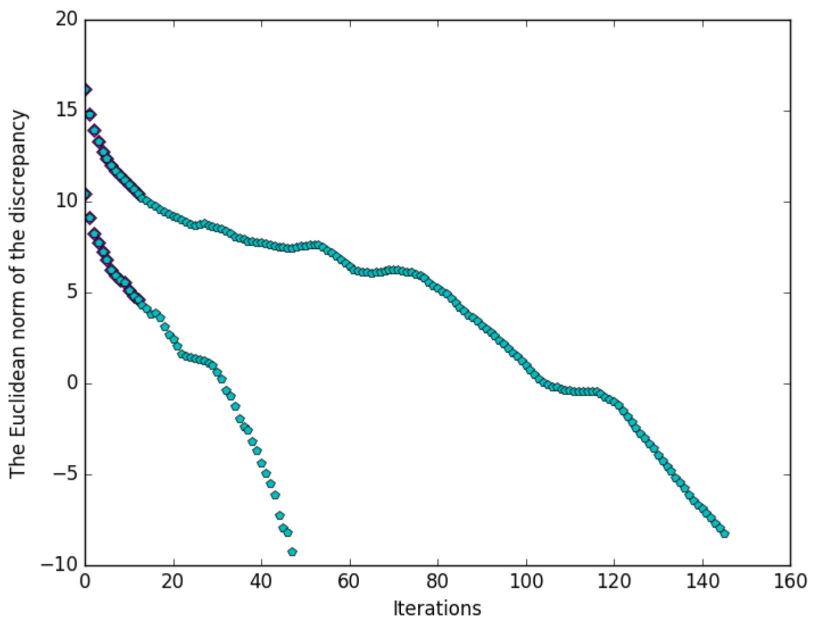

Figure 22: Dependence of Euclidean norm of the discrepancy $r_k = Ax_k - b$ (in logarithmic scale) against iteration number

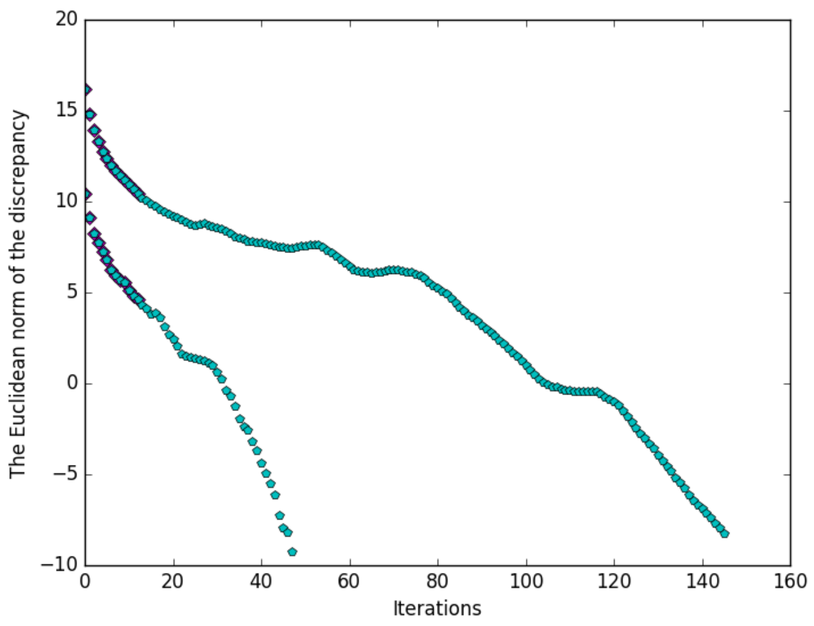

Figure 23: Dependence of Euclidean norm of the discrepancy $r_k = Ax_k - b$ (in logarithmic scale) against iteration number

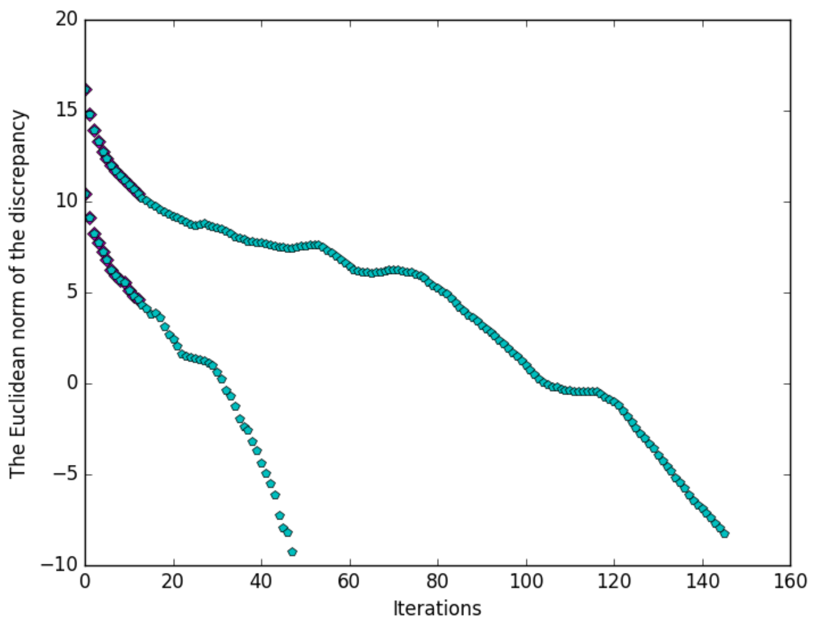

Figure 24: Dependence of Euclidean norm of the discrepancy $r_k = Ax_k - b$ (in logarithmic scale) against iteration number

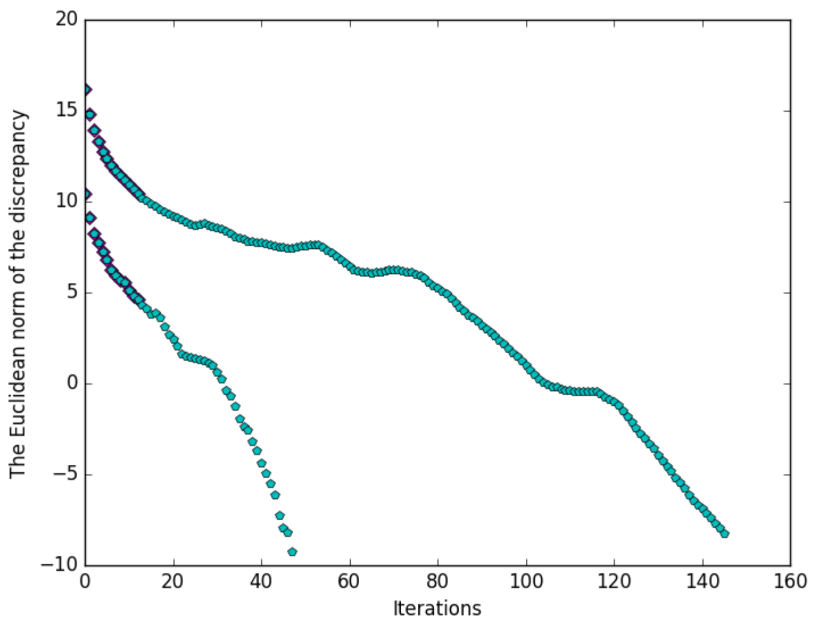

Figure 25: Dependence of Euclidean norm of the discrepancy $r_k = Ax_k - b$ (in logarithmic scale) against iteration number

3.1.2 Control Question

Question 5. *How can you explain the results?*

Solution. Well, I can only propose why it happens. Probably it it this way, because in L-BFGS steps are more expansive than in conjugate gradient method.

□

Question 6. *What is changing, when we use different history sizes in the L-BFGS method?*

Solution. Well, as you have already noticed, when we have history size > 0 our results are better, when $l = 0$. However, when we increase l even more, nothing happens. It's because gradient converges really fast.

□

Question 7. *Calculate the best value for α_k?*

Solution. So, the analytical equation for thisα_k looks like this:

$$\frac{b^T \cdot d - d^T \cdot A \cdot x}{d^T \cdot A \cdot d}$$

□