Newton - free methods

Artem Panchenko
artempanchenkotop@gmail.com
Oleksandr Deineha
jesse.pinkman.d@gmail.com

March 8, 2020

Abstract

The main goal of our final DOG project is to implement the Newton method without inverting the Hessian, but using the conjugate gradient method. Then we need to evaluate the effect of stopping the conjugate method after i $\leq$ d steps, where d is the dimension of the parameter vector.

1 Implementation of Newton method

First of all we need to implement the Newton method without inverting the Hessian. We want to avoid the Hessian invertation because we need a lot of memory - $O(n^2)$ and computation complexity $O(n^3)$. We can avoid computation complexity we will implement inexact Newton method without Hessian inverting but we will approximate this "inverted" matrix using the Conjugate gradient method. Our's implementation is based on the algorithm that describe in (3).

1.1 Implementation of Gradient and Hessian computation

As we know, the calculation of the Gradient and Hessian is necessary for the implementation of the Newton method. To do this, we created custom method that able to work with any function due to symbolic computations. For symbolic computations we used Python library Sympy. It is extremely usefull when we work with Mathematical analysis problems. The documentation is on the following link - `http://lidavidm.github.io/sympy/modules/tensor/array.html?highlight=derive_by_array`.

1.2 Using Conjugate gradient method to solve equations

As we know, one of the main drawbacks of the Newton method is the need to invert the Hessian, which requires a lot of computing power and memory consumption. In order to avoid this, we will not consider the inversed matrix with absolute accuracy, but approximate it with CG. We use `scipy.sparse.linalg.cg` method. The documentation is on the following link - `https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.cg.html`

1.3 Testing

In this section we want to test and make sure that our implementation of Inexact Newton method works as we expect. And parameter d (max number of iteration) of CG affect on the number iteration in Newton method the correct way.

For it we also add a solver that do it with maximum accuracy - `numpy.linalg.solve`. We will call the method with this solver as Exact Newton method. And we will compare the Inexact with parameter `maxiter` and Exact methods. we also expect that the work time of Inexact method in terms of number of iterations will bigger that in Exact case. And

even more we expect that in case of parabola function we will rich the global minimum in one step in case of using the Exact method.

We will test it in the function $f(x, y) = x^2 + y^2$. We know that this function has one minimum in point (0;0) and it is equal to 0.

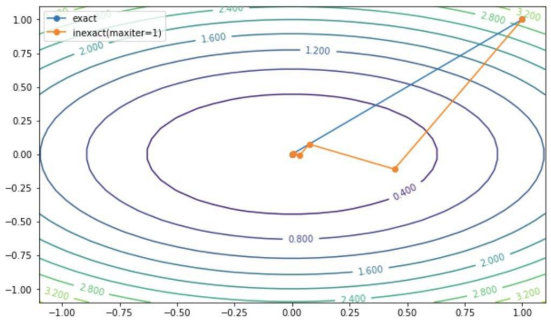

Figure 1: Newton method convergence.

So as we see from the Figure 1 two approaches (with Exact and Inexact) are converged to the minimum but in the case of CG we need more iterations. So our methods works correctly. We also made test with parameter `maxiter` equal to size of Hessian matrix and got the same results as in Exact case.

2 The impact of CG premature stop

By premature stop of CG method, we mean that we do not stop by tolerance criterion, we always perform strictly defined number of iterations that we call `maxiter`.

2.1 Saddle points avoiding

So, we have a hypothesis that if we calculate the inverted Hessian not with absolute accuracy, but with a certain error, we can solve the main drawback of second-order methods, namely, that they cannot pass the saddle point.

Lets check our hypothesis with $f = \sum_{i=1}^{10} ix^3$. We know that there no minimum but we have a saddle point.

Also as a benchmark we added the custom First order method. We did it as follows: instead of Hessian we input the Identity matrix in the Newton method. This was done because we know that first-order methods handle the function with saddle points.

In the Figure 2 we see how our methods converge.For Exact method we tried to set starting point upper(in the point (1, 1..., 1)) and even lower(in the point (-1, -1..., -1)) that saddle point but in both cases we were absorbed.

For Inexact method we always started lower(in the point (-1, -1..., -1)) that saddle point and tried different values to `maxiter`parameter in CG but it didn't help us to avoid the absorbing in the saddle point.

In the same time First-order method as we expect successfully passed the saddle point and went further.

So, the conclusion that we make is that this is impossible to avoid the absorbing in the saddle point when we use Newton method, even when our starting point is under the saddle and we use Inexact approach.

2.2 Evaluation in terms of iterations number

So, as we saw it is impossible to avoid the saddle points. But what profit in terms of iterations can we get from the CG premature stop? In order to find out, we used Colville function

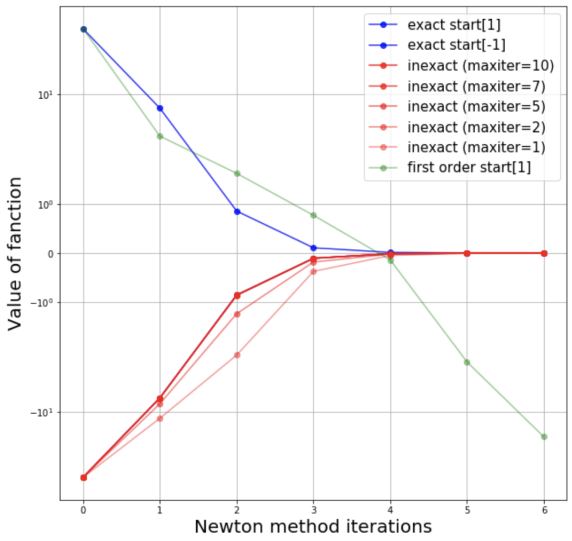

Figure 2: Convergence comparison.

$f = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - x_1)(x_4 - x_1)$. This is not trivial function that was created for optimizers testing - that's why we choose it. We knows that this function has one minimum in the point (1, 1, 1, 1) and it is equal to 0(1).

We launch the Inexact method in this function but with different `maxiter` values. The goal is to find the dependency between the number of CG iteration and the convergence time in terms of Newton iterations. We expect that the `maxiter`will affect in the convergence time as follows: with decreasing if maximum iteration number the convergence time will increase.

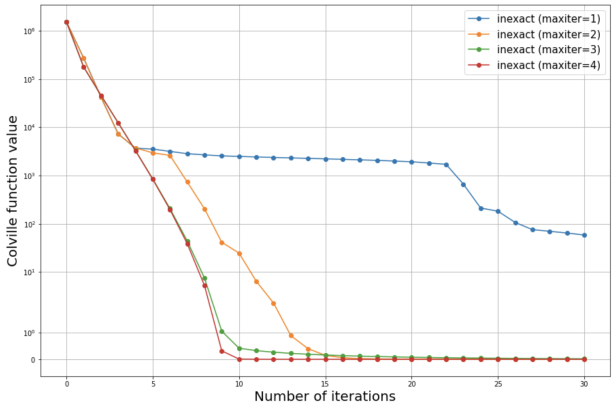

Figure 3: Convergence comparison.

The Figure 3 shows us the first 30 iteration of newton methods for Colville function. We see that the number of iteration in CG is strongly affects the convergence dynamic.

Now we want to evaluate the profit of `maxiter` decreasing in terms of CG iterations. We assume that the hardest pert of Newton iteration is the approximating via CG method in other words the number of iterations of CG. So, for us the total number of operations that we need to converge via Newton method is trivially number of Newton iteration multiplied by `maxiter` . As we said before in CG we always perform the `maxiter` iterations.

Also we evaluated the profit of tolerance changing without decreasing or increasing `maxiter`.

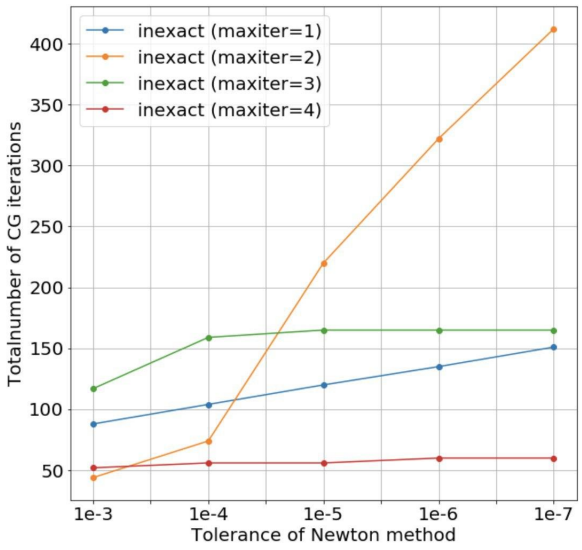

Figure 4: Necessary number of iterations to converge by tolerance.

So, in the case of Colville function get some unexpected result namely that we should not decrease the maxiter number. As we see the best convergence speed in the terms of total number of CG iterations is in case when we set maxiter parameter equal to dimension of parameter vector. This approach the best one not only in terms of convergence time but it is stable to changes of tolerance.

We also did similar experiments with the different functions. We used the polynomial function with same degrees of variables. Then we started increasing the degree and the profit of premature stop started decreasing. The main heuristic that we find is following: when our function is close to parabolic we should set maxiter close to dimension of the parameter vector and otherwise we can profit from the premature stop. This experiments are available in the Additional experiments.

3 Conclusions

We implemented the Hassian - free Newton method specifically Inexact Newton method. By lot of experiments we showed that even with using the inexact approach and performing computation with low tolerance it is impossible to avoid absorbing in the saddle point in case of usage the Second-order optimization.

From the other hand we evaluated the effect of premature stop of CG method. The heuristic that we found - if work with parabola-like function the better way is to compute number of CG iteration that equal to dimension of parameter vector. Otherwise the using of Premature stop approach is rational and can benefit in terms of total number of CG iterations.

Also we tested our Newton method on the Liner regression and get correct result, so it means that our implementation is correct and could be applied. This experiment is present in the end of source code.

4 Additional experiments

In this chapter we put the most significant experiments. The all experiments that we did is available in the source code.

4.1 Monkey saddle point (2)

For testing the ability of the Inexact method to avoid the saddle point we perform it in on Monkey saddle function : $f(x, y) = x^3 - 3xy^2$.

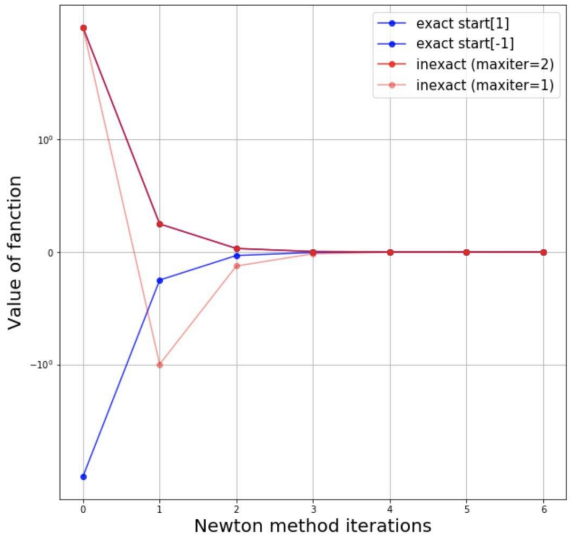

Figure 5: Convergence dynamic in case of Monkey saddle.

As we see in the Figure 5 Inexact method with maxiter = 1 avoid the saddle point on the first step but then anyway was absorbed. From this we concluded that we can't avoid absorbing in saddle point in case of usage the Inexact method.

4.2 Premature stops on the parabolic functions

To evaluate the affect of Premature stop approach in terms of the total number of CG iterations we did experiments with parabolic functions. In general this functions are $f = \sum_{i=1}^{n} 2^i x^k$. So we have two parameters: n - the dimension of the parameter vector and k - the degree of the polynomial.

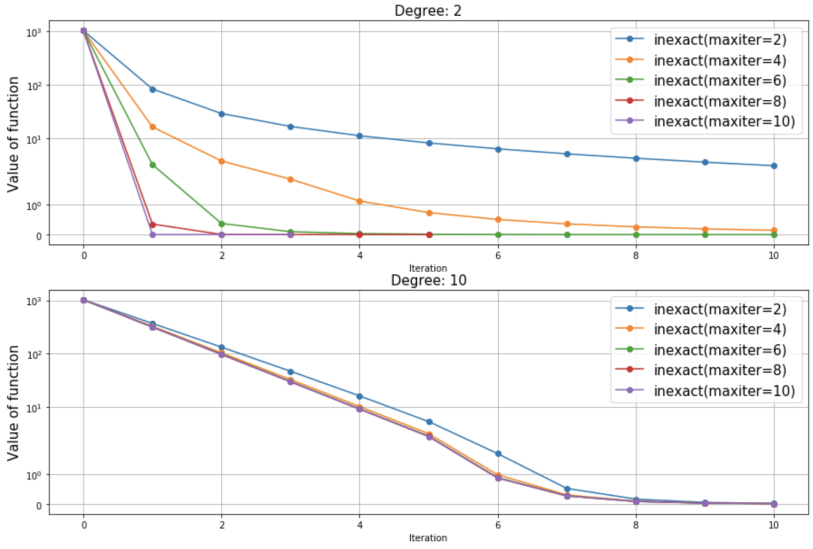

Figure 6: Convergence dynamic in case of Polynomial function.

From the Figure 6 we see that with increasing of the polynomial degree we loose the sens to perform all CG iteration and the better way in terms of total CG operation is to use Premature stop approach.

REFERENCES

[1] *http://www.sfu.ca/ ssurjano/colville.html*.

[2] *Monkey saddle*.

[3] Jorge Nocedal Léon Bottou, Frank E. Curtis. *Optimization Methods for Large-Scale Machine Learning*. 2018.