Date: 26th April 2020

AP-266 Homework 02

Reynaldo Lima

Contents

1 Newton's Method

Newton's method applies:

$$x^{(n+1)} = x^{(n)} + \delta x,$$

where:

$$H(x)\delta x = -gradf,$$

f is a function $f : \Re^n \to \Re$, x is a n-dimensional vector and H is the function f Hessian.

From the Taylor's series, this method is based on the following equality:

$$gradf(x + \delta x) = gradf(x) + H(x)\delta x + R,$$

with R as a residue regarding higher order (at leas square ordered) terms of δx.

Therefore, to prove that Newton's Method converges to the **MINIMUM** of the quadratic function defined as:

$$f(x) = \frac{1}{2}x^T Px + q^T x + r,$$

we must prove: 1) R is null after one iteration; 2) the point corresponds to a minimum.

1.1 Number of iterations

First, let's apply Taylor's series for f:

$$\begin{aligned} f(x+\delta x) &= f(x) + grad^T f(x)\delta x + \frac{1}{2}\delta^T x H_f(x)\delta x, \\ &= f(x) + grad^T f(x)\delta x - \frac{1}{2} grad^T f(x)\delta x, \\ &= f(x) + \frac{1}{2} grad^T f(x)\delta x, \\ &= f(x) - \frac{1}{2} grad^T f(x) H_f^{-1} gradf(x), \end{aligned}$$

and noting from f definition that $P = H_f(x)$ and $gradf(x) = p(x)$:

$$f(x+\delta x) = f(x) - \frac{1}{2} p^T(x) P^{-1} p(x).$$

Now, let's see how to calculate the gradient of this expression. First, take y as follows:

$$\begin{aligned} y &= p^T(x)A(x)p(x), \\ dy &= d(p^T(x)A(x)p(x)) = d(\langle A(x)p(x), p(x)\rangle), \\ &= d(\sum_{i=1}^{n}(A(x)p(x))_i p_i(x)) = d(\sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j}(x)p_i(x)p_j(x)), \end{aligned}$$

as $A(x)$ is the Hessian of a quadratic function, we can ignore derivative terms of $a_{i,j}$, so:

$$\begin{aligned} dy &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j}p_i(x)d(p_j) + \sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j}p_j(x)d(p_i), \\ &= \sum_{i=1}^{n}(Ap(x))_i d(p_i(x)) + \sum_{i=1}^{n}(Ad(p(x)))_i p_i(x), \\ &= (dp(x))^T Ap(x) + p^T Ad(p(x)), \\ &= p^T(x)A^T(dp(x)) + p^T Ad(p(x)) = p^T(x)(A^T + A) grad^T p(x)dx, \end{aligned}$$

thus, using $dz = grad^T z dx$:

$$grady = gradp^T(x)(A + A^T)p(x).$$

Therefore:

$$\begin{aligned} gradf(x+\delta x) &= gradf(x) - \frac{1}{2}P(2 * P^{-1})p(x), \\ &= p(x) - p(x) = 0. \end{aligned}$$

It follows that there is no residue when f is a quadratic function.

1.2 Minimum value

In order to be a minimum, it is sufficient that all eigenvalues of the function's Hessian are positive, which is true as P is positive definite.

2 Gradient-based Optimization

For this part, let's define the function f as the Rosenbrock's test function, as follows:

$$f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right].$$

2.1 Part a)

Presenting, from 1.A to 1.C, final points from optimizations.

Method	x_1	x_2
CG	0.99999552	0.99999103
BFGS	0.99999467	0.99998932

Table 1.A: 2-dimensional final point.

Method	x_1	x_2	x_3	x_4
CG	0.99999869	0.99999741	0.99999482	0.99998963
BFGS	0.9999985	0.99999701	0.99999403	0.99998806

Table 1.B: 4-dimensional final point.

Method	x_1	x_2	x_3	x_4	x_5	x_6
CG	0.999999	0.999997	0.999995	0.999990	0.999995	0.999990
BFGS	0.9999996	0.9999992	0.9999984	0.9999968	0.9999935	0.9999871

Table 1.C: 6-dimensional final point.

Table 2, on the other hand, presents the final value from the objective function after the optimization process. These results are obtained from *grad$_a$.py* python code, provided along with this document.

2.2 Part b)

In this section, we will further compare the gradient based methods with Nelder-Mead and Differential Evolution methods, reaching $n = 20$ dimensions for the function, using only even entries (in order to diminish the evaluation time, while still gathering information for higher values of n.

Dimension n	CG		BFGS	
	fun (e-11)	nfev	fun (e-11)	nfev
2	2.01	200	2.84	96
4	3.53	810	4.68	270
6	7.44	1824	5.60	408

Table 2: Results of function value and number of function calls for CG and BFGS methods.

Figure 1 represents the logarithmic value of number of function evaluations. In CG and BFGS methods, it's also included the number of Jacobian evaluations, as it also has a representative cost.

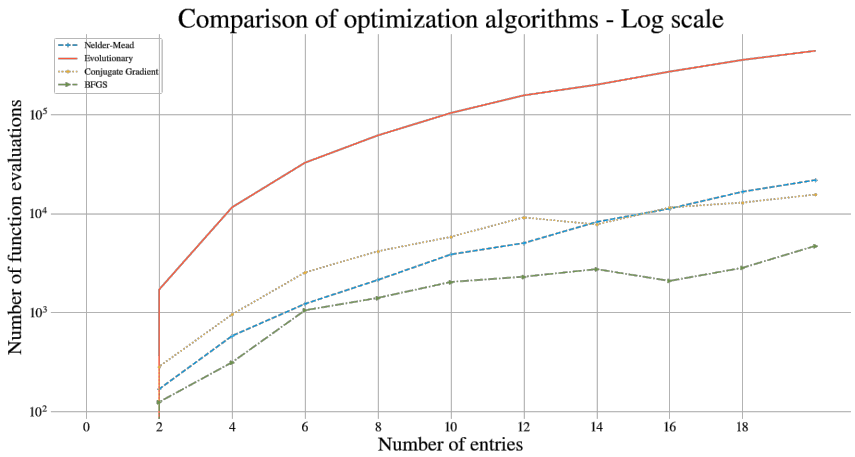

Figure 1: Number of evaluations (nfev + njev for CG and BFGS) in log. scale.

Figure 2 represents the final value that the function was evaluated, noting that the Y-Axis is represented in powers of $E-11$.

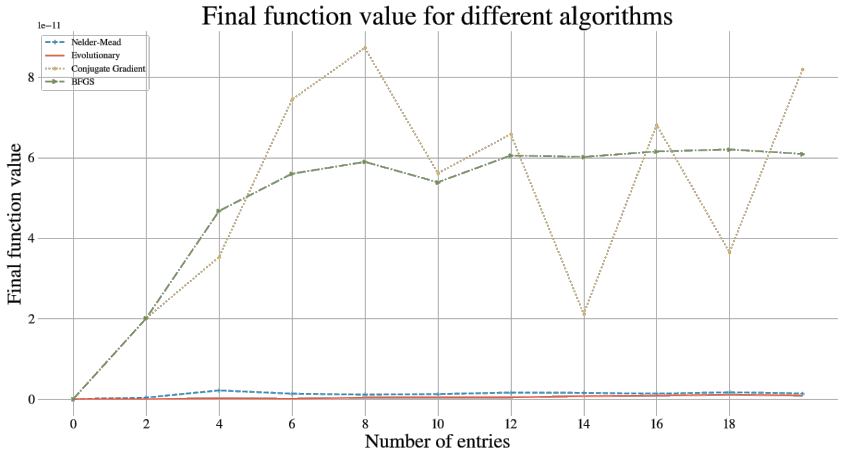

Figure 2: Final function value for different algorithms and number of entries. Y-axis in E-11 units.

From the logarithmic plot, it's possible to conclude that the Evolutionary algorithm has a order near $O(n^{2.5})$, Nelder-Mead $O(n^{1.66})$, CG $O(n^{2.1})$ and BFGS $O(n^{1.43})$.

3 Algorithmic Differentiation

3.1 Computing the AIC matrix

Using $compile_test_1.sh$, sent with this document, a $Fortran90$ code generates our desired answer:

$$AIC(1,:,:) = \begin{bmatrix} 0.00000000 & 0.00000000 & 0.00000000 \\ 0.00000000 & 3.95912677E-02 & -1.03842612E-09 \\ 0.00000000 & 1.03842612E-09 & -7.91825354E-02 \end{bmatrix},$$

$$AIC(2,:,:) = \begin{bmatrix} 3.15158255E-02 & -1.05052758E-02 & -2.10105511E-03 \\ -1.05052739E-02 & 3.15158293E-02 & -1.05052739E-02 \\ -2.10105511E-03 & -1.05052739E-02 & 3.15158293E-02 \end{bmatrix},$$

$$AIC(3,:,:) = \begin{bmatrix} -0.315158248 & 0.105052739 & 2.10105479E-02 \\ 0.105052739 & -0.315158248 & 0.105052739 \\ 2.10105479E-02 & 0.105052739 & -0.315158248 \end{bmatrix}.$$

3.2 Testing the LLT method functions

Running $compile\_test\_2.sh$, regarding the values asked, we obtain (first the test case):

$$\begin{aligned} res &= \begin{bmatrix} 6.42611885 & 5.06466341 & 6.42611885 \end{bmatrix}, \\ Sref &= 0.900000036, \\ CL &= 13.3333330, \\ CD &= 4.33844566, \\ L &= 6.00000000, \\ D &= 1.95230055, \end{aligned}$$

with some error in precision, when compared to the expected outputs. We also obtain for the values asked:

$$\begin{aligned} res &= \begin{bmatrix} 1.29698443 & 8.40076160 & 1.29698443 \end{bmatrix}, \\ Sref &= 1.21188235, \\ CL &= 6.79936743, \\ CD &= 1.67543256, \\ L &= 4.12001657, \\ D &= 1.01521361. \end{aligned}$$

3.3 Differentiating the code

Resulting codes, with subroutine implement, sent with this document.

3.4 Finite difference test

First, for $get\_residuals$, $h = 1E-4$ provided the lesser norm from the difference vector. This may be verified with the $compile\_test\_4.sh$ file. Therefore, the proposed results for the derivative values are:

$$\begin{aligned} D_{dir}(res) &= \begin{bmatrix} -18.1453667 & 12.8681326 & 0.836128592 \end{bmatrix}, \\ D_{dir}(Sref) &= 0.929999948, \\ D_{dir}(CL) &= -21.0620708, \\ D_{dir}(CD) &= -4.07310867, \\ D_{dir}(L) &= -3.27793193, \\ D_{dir}(D) &= 0.184477895. \end{aligned}$$

Important to note that until this point, all computations had not the Flag $-fdefault-real-8$ during compilation.

3.5 Dot product test

Using $compile\_test\_4.sh$, with the Flag $-fdefault-real-8$, we have the following outputs, regarding $Residuals$:

$$D_{dir}(res) = \begin{bmatrix} -18.145367163536253 & 12.868133426945596 & 0.83612908922787921 \end{bmatrix},$$

$$\bar{X}(:,1:2) = \begin{bmatrix} -0.21563676169662749 & 0.39730373092641014 \\ -0.98063236164658618 & 3.0138161578001932 \\ 9.6484615849870836E-003 & 0.25497723133118594 \end{bmatrix},$$

$$\bar{X}(:,3:4) = \begin{bmatrix} 0.34775819386374218 & -0.52942516309352483 \\ -4.9897171912641216 & 2.9565333951105139 \\ -0.53889984741733332 & 0.27427415450116022 \end{bmatrix},$$

$$Chordsb = \begin{bmatrix} 0.66592601539380669 & -0.61796729109657855 & 1.9977780461814192 \end{bmatrix},$$

$$\bar{\alpha} = \begin{bmatrix} -0.54753306752365061 & 0.88659049650207156 & -1.6425992025709515 \end{bmatrix},$$

$$\bar{\gamma} = \begin{bmatrix} 0.92811003111819146 & -1.7460953380716697 & 2.6741631264435952 \end{bmatrix},$$

while the first dot product:

$$dotPr = -8.8817841970012523E-016.$$

Repeating the same process to $get_functions_b$:

$$D_{dir}(res) = \begin{bmatrix} -18.145367163536253 & 12.868133426945596 & 0.83612908922787921 \end{bmatrix},$$

$$\bar{X}(:,1:2) = \begin{bmatrix} -0.13568646404681028 & 0.13568646404681051 \\ 0.64811422222512871 & -0.25049808747103441 \\ -0.48867247385162632 & 0.48867247385162632 \end{bmatrix},$$

$$\bar{X}(:,3:4) = \begin{bmatrix} 0.13568646404681051 & -0.13568646404681023 \\ 0.25049808747103436 & -0.64811422222512893 \\ 0.48867247385162627 & -0.48867247385162632 \end{bmatrix},$$

$$Chordsb = \begin{bmatrix} 3.0809134305136840 & 3.0809134305136840 & 3.0809134305136840 \end{bmatrix},$$

$$\bar{\alpha} = \begin{bmatrix} 0.0000000000000000 & 0.0000000000000000 & 0.0000000000000000 \end{bmatrix},$$

$$\bar{\gamma}^{1} = \begin{bmatrix} -3.991857343216E-2 & -.35304118277095 & -3.991857343216E-2 \end{bmatrix},$$

while the dot product:

$$dotPr = 8.6042284408449632E-016.$$

[1] Truncated values for better visualization of data.