ap-266-homework-1

reynaldo.sdlima@gmail.com

March 2020

1 Extra

2 Unconstrained Optimization

2.1 Part a)

For a point to be a critical point, it is necessary and sufficient to the gradient of that function on that point to be the vector 0.

Thus,

$$grad(f(x)) = 0,$$

$$\frac{\partial f(x)}{\partial x_1} = -2 + 2x_1 + 100(4x_1^3 - 4x_1x_2) = 0$$

$$\frac{\partial f(x)}{\partial x_i} = -2 + 2x_i + 100(4x_i^3 - 4x_ix_{i+1} + 100(2x_i - 2x_{i-1})), 0 < i < n$$

where i is a positive integer;

$$\frac{\partial f(x)}{\partial x_n} = 100(2x_n - 2x_{n-1}^2) = 0.$$

It's immediate that the point $x = [1, 1, ..., 1]$ respects the condition stated.

2.2 Part b)

Programs are sent with this document. The result for this part:

python3 unconstrained-optimization.py

NM. for n = 2 [1.00000439 1.00001064] Optimization terminated successfully.

NM. for n = 4 [0.99999657 0.99999435 0.99998732 0.99997529] Optimization terminated successfully.

NM. for n = 6 [1.00000268 1.00000457 1.00001286 1.00002301 1.00004079 1.00008331] Optimization terminated successfully.

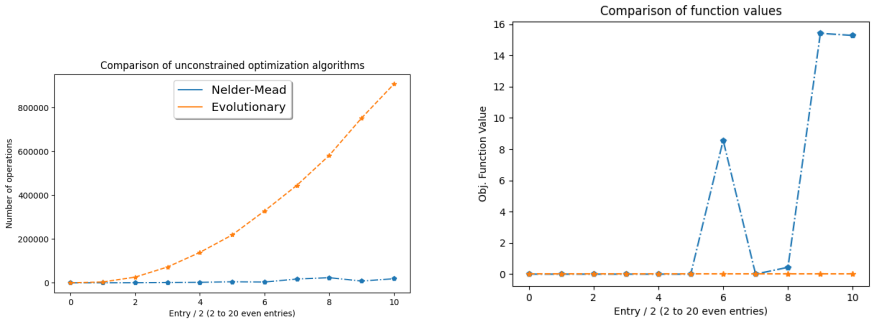
Ev. for n = 2 [1. 1.] Optimization terminated successfully.

Ev. for n = 4 [1. 1. 1. 1.] Optimization terminated successfully.

Ev. for n = 6 [1. 1. 1. 1. 1. 1.] Optimization terminated successfully.

2.3 Part c)

As my program is quite slow, I used only even entries for the graph.



(a) (b)

Figure 1: (a) number of iterations. (b) function value.

As you may see, NM algorithm is of order 0, while the evolutionary is of higher order. But again, NM does not minimize the function in some points, as a result of its tendency to catch local minimums instead of global minimums.

Further analysis in this matter in Fig. 2 confirms that, despite outliers, Nelder-Mead is of order 0 and Evolutionary of order 2. This was a different iteration then the presented inf Fig. 1. Printed entries before "Ev." are for Nelder-Mead (number of calls ration between two following even elements vs. ration of first order of following even elements); after Ev. are the rations of Evolutionary (ratio of calls, ratio of entries, ratio of entries squared).

```
reynaldo@reynaldo:~/ap-266/2-unconstrained-optization$ python3 unconstrained-optimization.py
NM. for n =  2 [1.00000439 1.00001064] Optimization terminated successfully.
NM. for n =  4 [0.99999657 0.99999435 0.99998732 0.99997529] Optimization terminated successfully.
0.3459715639810427 0.5
NM. for n =  6 [1.00000268 1.00000457 1.00001286 1.00002301 1.00004079 1.00008331] Optimization terminated successfully.
0.2944870900209351 0.6666666666666666
0.6031144781144782 0.75
0.5024318037640093 0.8
1.3198437063912922 0.8333333333333334
0.20708588602473701 0.8571428571428571
0.7350667006542612 0.875
2.9481462925851702 0.8888888888888888
0.42633630586853205 0.9
Ev. for n =  2 [1. 1.] Optimization terminated successfully.
Ev. for n =  4 [1. 1. 1. 1.] Optimization terminated successfully.
Ev. for n =  6 [1. 1. 1. 1. 1. 1.] Optimization terminated successfully.
0.5236144810750246 0.75 0.5625
0.6136714463301018 0.8 0.64
0.6927944781115041 0.8333333333333334 0.6944444444444444
16.574797385620915 0.8571428571428571 0.7346938775510204
0.0335375102804525 0.875 0.765625
0.7561796456574406 0.8888888888888888 0.7901234567901234
0.8279661975294816 0.9 0.81
```

Figure 2: Output for number of function calls.

2.4 Part d)

As already stated in c), the Nelder-Mead method has the advantage to be quicker, as it is of a reduced order. On the other hand, it works to find minimums, not specifically global minimums, which may result in false final values.

Evolutionary method does consumes a lot more computational time, but always results on the global minimum of the function.

3 Fortran Coding

3.1 Part a)

Just the general code sent with this document. I've divided vector operations subroutines in its own module. Important to note that in cases that a vortex try to induce a velocity in itself, the value of the function diverges. Thus, this term must be eliminated.

3.2 Part b)

Test and proposed outputs:

velInd, test case: 3.11789103E-02 5.01817018E-02 -2.93957647E-02
velInd, case 1: 0.00000000 0.00000000 5.30516431E-02
velInd, case 2: 0.00000000 0.00000000 5.30516431E-02
velInd, case 3: -0.321591675 1.88360382E-02 0.204342127
velInd, case 4: -1.45367170E-02 2.48351283E-02 3.59370783E-02
velInd, case 5: 0.00000000 0.00000000 -0.159154937

3.3 Part c)

Outputs:

Matrix of seq. linked horseshoes vortices, Case 1:

$$0.00000000|0.00000000| - 0.318309873$$

$$0.00000000|0.00000000|0.106103286$$

$$0.00000000|0.00000000|2.12206580E - 02$$

Matrix of seq. linked horseshoes vortices, Case 2:

$$-3.18340850|3.06067187E - 02| - 0.312188536$$

$$-6.76724222E - 03| - 2.36567222E - 02|0.100908622$$

$$-5.52407291E - 04| - 3.18450248E - 03|2.07458548E - 02$$