INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Alejandro Arturo Rios Cruz

Homework 02- Aeroestructural Optimization

AP-266

São José dos Campos
05/04/2018

1

1.1

Consider the multidimensional quadratic function below:

$$f(\vec{x}) = \frac{1}{2}\vec{x}^T\mathbf{P}\vec{x} + \vec{q}^T\vec{x} + r \tag{1.1}$$

where $\mathbf{P}$, $\vec{q}$ and r are constant, and $\vec{x}$ is a vertical vector. Also, P is a symmetric positive-definite matrix. Let $\vec{x_0}$ be the starting point for the Newton's Method. Prove that the Newton's Method converges to the MINIMUM of the quadratic in a single step.

The Newton Method is an iterative gradient-based optimization method that given an starting point it will compute the functions and their gradients, which will be used to find a search direction that points to the direction of the global minimum. In other words, it is a descent method with a specific choose of a descent direction. In order to find the global minimum we want that in the first iteration the method finds the global minimum point:

$$\vec{\nabla} f(\vec{x} + \Delta\vec{x}) = 0 \tag{1.2}$$

To find the search direction that points to the global minimum in the first iteration of a multidimensional function we first need to perform a second order expansion of Taylor series around the current iteration point and minimize the resulting quadratic function. Considering a small increment given by $d = x - x_k$ where x_k is the current point:

$$f(x) = f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2 f(x_k) d \tag{1.3}$$

This function needs to meet the requirements of first ($\nabla f(x) = 0$) and second ($\nabla^2 f(x)$ positive defined) order sufficient conditions to be a minimum. In other words $f(x)$ will be lower bounded only if $\nabla^2 f(x_k) \geq 0$, hence:

$$\nabla f(x_k) + \nabla^2 f(x_k) x - \nabla^2 f(xk) x_k = 0 \tag{1.4}$$

Solving for x:

$$x = \nabla^2 f(x_k)^{-1} (x_k \nabla^2 f(x_k) - \nabla f(x_k)) \tag{1.5}$$

In order that $\nabla^2 f(x_k)$ to be inversible, it need to be positive definite. Then if

$\nabla^2 f(x_k) \geq 0$, f is convex and hence the stationary point is a global minimum. The newton method picks this as the next iterate point:

$$x = x_k - (\nabla^2 f(xk))^{-1} \nabla f(x_k) \tag{1.6}$$

Then if we have a multidimensional quadratic function as the given below:

$$f(\vec{x}) = \frac{1}{2}\vec{x}^T \mathbf{P}\vec{x} + \vec{q}^T\vec{x} + r \tag{1.7}$$

The only requirement for finding the global minimum in a sigle iteration is that P need to be $P > 0$. Because it is specified in the problem that P is a symmetric positive-defined matrix, the minimum will be obtained in a single iteration.

2

Let's continue using the Rosenbrock's function from the previous homework assignment:

$$f(x_1 \cdots x_n) = \sum_{i=1}^{n-1} (100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2) \tag{2.1}$$

2.1

Write Python3 programs to minimize the Rosenbrock's function using analytical gradients for any dimension n using:

2.1.a

1. The Conjugate Gradient (CG) Algorithm, starting at the point $\vec{x} = [0, 0, ..., 0]$.

The optimization problem was solved for $n = 2$, $n = 4$ and $n = 6$. The values of the design variables, objective function, and number of function evaluations for the Conjugate Gradient Algorithm are shown in table 1.

Tabela 1 – Conjugate Gradient Algorithm Results.

Variable	Value ($n = 2$)	Value ($n = 4$)	Value ($n = 6$)
Design Variables	$x[1.0, 1.0]$	$x[1.0, 1.0, 1.0, 1.0]$	$x[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]$
Objective fun.	0	0	$1.12e-27$
Num. of fun. eval.	200	230	430
Num. of fun. eval. (nfev+njev)	397	448	848

2.1.b

2. The BFGS Algorithm, starting at the point $\vec{x} = [0, 0, ..., 0]$.

For the BFGS Algorithm the problema was also solved for $n = 2$, $n = 4$ and $n = 6$. The values of the design variables, objective function, and number of function evaluations for the BFGS Algorithm are shown in table 2.

Tabela 2 – BFGS Algorithm Results.

Variable	Value ($n = 2$)	Value ($n = 4$)	Value ($n = 6$)
Design Variables	$x[1.0, 1.0]$	$x[1.0, 1.0, 1.0, 1.0]$	$x[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]$
Objective fun.	0	0	0
Num. of fun. eval.	28	50	57
Num. of fun. eval. (nfev+njev)	56	100	114

2.2

Now let's check how these algorithms scale with the number of design variables. For this purpose, a plot showing the number of functions evaluations with respect to the number of design variables were created. The results of BFGS and CG were compared with the NM and DE method from the codes of the first homework. A max. numer of $n = 20$ design variables was obtained. The results are presented in figure 1.

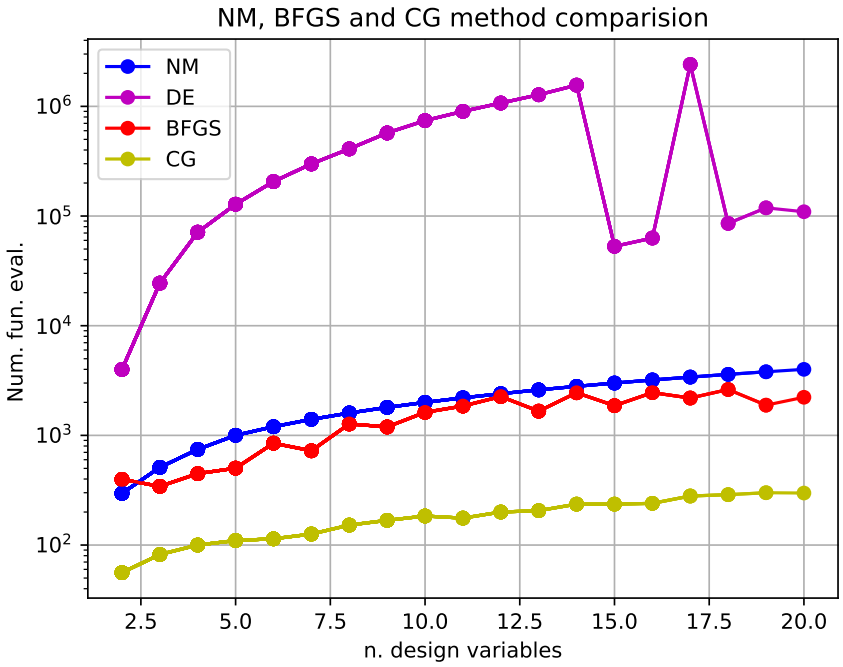

Figura 1 – Number of function evaluation vs numb. of design variables for Nelder-Mead (blue), Genetic (magenta), BFGS (red) and CG (yellow), $n_{max} = 20$

Because the cost of computing jacobians with reverse AD is roughly the same order of magnitude as the function evaluations, the number of function evaluations for BFGS and CG were taken as (nfev+njev).

The behavior of the objective function vs the number of design variables for both algorithms is presented in figure 2.

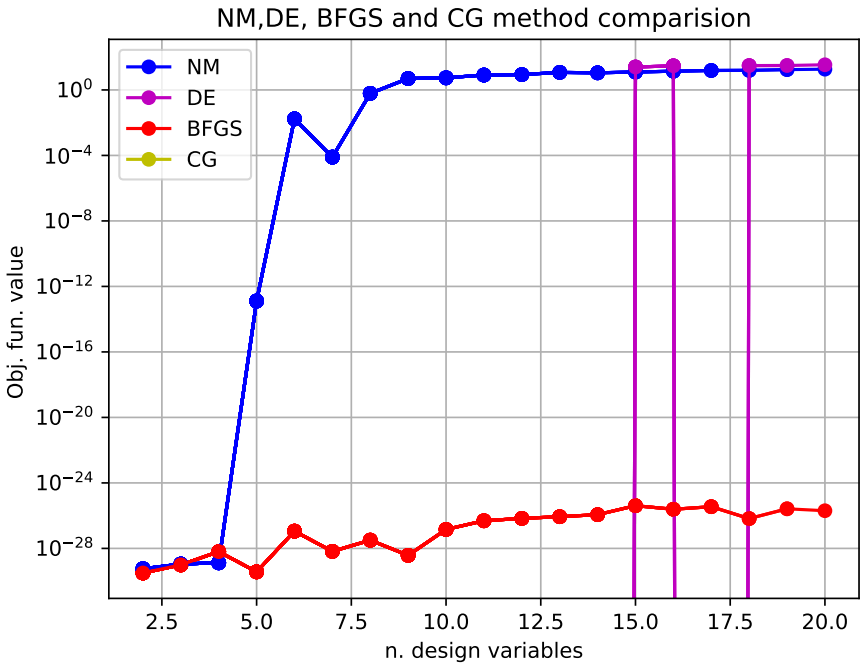

Figura 2 – Objective function value vs numb. of design variables for all four algorithm's Nelder-Mead (blue), Genetic (magenta), BFGS (red) and CG (yellow), $n_{max} = 20$.

From this plot it is observable that the comparison between all four methods is fear only for a small among of design variables (comparison between BFGS and CG with NM and DE). The current graph doesn't show the Obj. function value for CG because it presented an exact solution (zero). Another plot (not log) is shown in figure 3, in order to show that BFGS and CG present similar results while the objective function increase for the other two methods after 8 number of design variables.

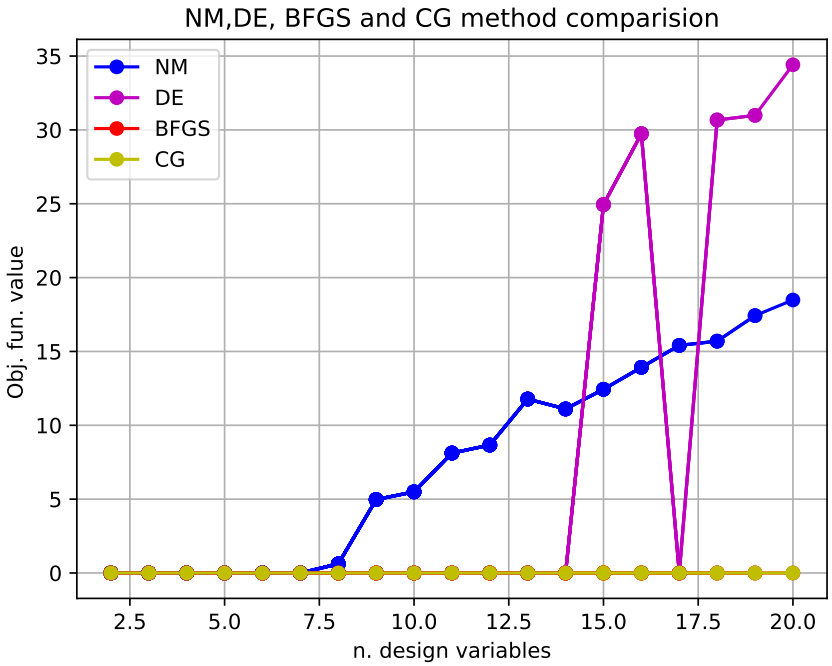

Figura 3 – Objective function value vs numb. of design variables for all four algorithm's Nelder-Mead (blue), Genetic (magenta), BFGS (red) and CG (yellow), $n_{max} = 20$.

From figure 1 it is observable that these methods are roughly proportional to a power of n equal to 3 (third power).

3

3.1

We will add a new subroutine to the llt_module.F90 file to compute the matrix A. This subroutine should be named get_AIC_matrix.

Give the aerodynamic influence matrices AIC for the following case:

$$\vec{u} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 0.1 & 0.2 & 0.3 \end{bmatrix}$$

The aerodynamic influence matrix for this case are presented below:

$$AIC(1,:,:) = \begin{bmatrix} 0.0000000000000000 & 0.0000000000000000 & -6.1895028843470178E-019 \\ 1.9342196513584432E-018 & 0.0000000000000000 & 0.0000000000000000 \\ 0.0000000000000000 & -1.9342196513584432E-018 & 0.0000000000000000 \end{bmatrix} \tag{3.1}$$

$$AIC(2,:,:) = \begin{bmatrix} 3.1515830315226812E-002 & -1.0505276771742270E-002 & -2.1010553543484539E-003 \\ -1.0505276771742275E-002 & 3.1515830315226812E-002 & -1.0505276771742270E-002 \\ -2.1010553543484530E-003 & -1.0505276771742268E-002 & 3.1515830315226805E-002 \end{bmatrix} \tag{3.2}$$

$$AIC(3,:,:) = \begin{bmatrix} -0.31515830315226806 & 0.10505276771742271 & 2.1010553543484533E-002 \\ 0.10505276771742271 & -0.31515830315226806 & 0.10505276771742271 \\ 2.1010553543484533E-002 & 0.10505276771742271 & -0.31515830315226806 \end{bmatrix} \tag{3.3}$$

3.2

To test the LLT method functions a test with the get_residuals .f90 amd get_functions.f90 was performed. This functions were tested for the following set of inputs:

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0.2 & 0.0 & 0.0 & 0.1 \end{bmatrix}$$

$$Gama = \begin{bmatrix} 1.0 & 2.0 & 1.0 \end{bmatrix}$$

$$alpha0 = \begin{bmatrix} 0.0 & 0.1 & 0.0 \end{bmatrix}$$

$$chords = \begin{bmatrix} 0.3 & 0.6 & 0.3 \end{bmatrix}$$

$$cla = \begin{bmatrix} 6.283 & 6.283 & 6.283 \end{bmatrix}$$

$$Vinf = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$rho = \begin{bmatrix} 1.0 \end{bmatrix}$$

The obtained values of residuals, Sref, CL, CD, L and D are presented in table 3

Tabela 3 – Values obtained from test of LLT functions.

Variable	Value
res	[1.3041508154864354 | 8.6450657222712124 | 1.3041508154]
Sref	1.2118823416311342
CL	6.7993670729921352
CD	1.7266958373595160
L	4.1200164450136700
D	1.0462760973319911

3.3

The tapenade main subroutine is presented in the code below. The resulting AD forward code is at the attached codes.

```
1
2   subroutine tapenade_main(n_vort, X, Gama, alpha0, chords,
       cl0, cla, Vinf, rho, res, Sref, CL, CD, L, D)
3
4     ! Dummy subroutine to allow differentiation of
        get_residuals and
```

```
5       ! get_functions in a single Tapenade call.
6       !
7       ! INPUTS
8       !
9       ! n_vort: integer -> Number of horseshoe vortices.
10      ! X: real(3,n_vort+1) -> Coordinates of the kinks of the
           horseshoe vortices.
11      ! Gama: real(n_vort) -> Circulation intensity of each
           horseshoe vortex.
12      ! alpha0: real(n_vort) -> Array of local incidence angles
           .
13      ! chords: real(n_vort) -> Array of local chords.
14      ! cl0: real(n_vort) -> cl0 of each 2D section
15      ! cla: real(n_vort) -> lift curve slope of each 2D
           section [1/rad]
16      ! Vinf: real(3) -> Free-stream velocity vector
17      ! rho: real -> Air density
18      !
19      ! OUTPUTS
20      !
21      ! res: real(n_vort) -> Residuals of the LLT method
22      ! Sref: real -> Area of all panels
23      ! CL: real -> Lift coefficient (adimensionalized by Sref)
24      ! CD: real -> Drag coefficient (adimensionalized by Sref)
25      ! L: real -> Lift force
26      ! D: real -> Drag force
27
28      implicit none
29      ! Input variables
30      integer, intent(in) :: n_vort
31      real, intent(in) :: X(3,n_vort+1)
32      real, intent(in) :: Gama(n_vort), alpha0(n_vort)
33      real, intent(in) :: chords(n_vort), cl0(n_vort)
34      real, intent(in) :: cla(n_vort), Vinf(3), rho
35
36      ! Output variables
37      real, intent(out) :: res(n_vort)
38      real, intent(out) :: Sref, CL, CD, L, D
39
40      ! Working variables
```

```
41      integer :: ii
42      real :: Uinf(3), AIC(3,n_vort,n_vort)
43      real :: Uai(3,n_vort), Uni(3,n_vort), si(3,n_vort)
44      real :: areas(n_vort), Vlocal(3,n_vort), alphaLocal(
            n_vort)
45      real :: Fcirc(3,n_vort), Fairf(3,n_vort), deltaF(3),
            Fbody(3), Fbodym, Vinfm
46
47      ! call of functions to apply the adjoint method
48      call get_functions(n_vort, X, Gama, alpha0, chords, cl0,
            cla, Vinf, rho, Sref, CL, CD, L, D)
49      call get_residuals(n_vort, X, Gama, alpha0, chords, cl0,
            cla, Vinf, rho, res)
50
51
52    end subroutine tapenade_main
53
54  end module llt_module
```

3.4

Let's use the finite difference method to compute the directional derivative of the residuals res with respect to the inputs variables.

In our case, since we have 4 input variables (X, Gama, alpha0, and chords), we need to define 4 perturbation directions (Xd, Gamad, alpha0d, and chordsd) to compute a direction derivative. For this exercise, we used the first set of inputs from Sec. 3.2 and the following perturbation directions:

$$\mathbf{Xd} = \begin{bmatrix} -0.1 & 0.5 & 0.2 & 0.0 \\ 1.0 & -0.7 & 0.1 & -0.9 \\ 0.3 & 0.0 & -0.6 & 0.2 \end{bmatrix}$$

$$Gamad = \begin{bmatrix} -1.0 & 0.5 & 0.3 \end{bmatrix}$$

$$alpha0d = \begin{bmatrix} 0.0 & 0.1 & -0.1 \end{bmatrix}$$

$$chordsd = \begin{bmatrix} 1.0 & -0.2 & 0.7 \end{bmatrix}$$

We need to find the step size magnitude n in $h = 10^n$ that gives the smallest difference between the directional derivatives estimated with the two methods (finite differences and forward AD). The values of n were varied from -4 to -12 in increments of -1. Nevertheless, table 4 present a resume of them.

Tabela 4 – Differences between dir. derivatives get residuals.

Variable	n=-4
Difference AD derivative	1.8132046015288950E-003 | 1.3840853368751738E-003 | 8.5726591428125776E-004 -18.145367171877378 | 12.868133525028602 | 0.83612926486240313
Variable	n=-8
Difference AD derivative	2.7395148194386820E-007 | 2.3353039502183037E-007 | 5.4549228578615327E-008 -18.145367171877378 | 12.868133525028602 | 0.83612926486240313
Variable	n=-12
Difference AD derivative	1.8942994355803933E-003 | 3.7572940919883280E-003 | 1.2415503442854137E-003 -18.145367171877378 | 12.868133525028602 | 0.83612926486240313

A similar analysis but for get functions subrotine was performed and the results are presented in tables 5 and 6.

Tabela 5 – Differences between dir. derivatives get functions.

n=-4			
Variable	Sref	CL	CD
Difference AD derivative	2.3964681007648814E-004 0.92999999999999994	6.3820991811560646E-003 -21.062070460165859	1.7893731399096779E-003 -4.0731090226145126
n=-8			
Variable	Sref	CL	CD
Difference AD derivative	1.1869296878685986E-008 0.92999999999999994	5.0538553253431928E-007 -21.062070460165859	6.1392252170833217E-008 -4.0731090226145126
n=-12			
Variable	Sref	CL	CD
Difference AD derivative	7.7194573968819569E-005 0.92999999999999994	3.3601290758902280E-003 -21.062070460165859	7.7210130261740062E-005 -4.0731090226145126

From this results we can conclude that the magnitude n that give the smallest difference between both methods is $n = -8$.

3.5

Now we will use the dot product test to verify the reverse AD code. For that purpose we tun the forward AD version of the get_residuals subroutine with this set of arbitrary derivative seeds:

$$\mathbf{Xd} = \begin{bmatrix} -0.1 & 0.5 & 0.2 & 0.0 \\ 1.0 & -0.7 & 0.1 & -0.9 \\ 0.3 & 0.0 & -0.6 & 0.2 \end{bmatrix}$$

Tabela 6 – Differences between dir. derivatives get functions.

n=-4		
Variable	L	D
Difference	2.9546201202679256E-004	9.6102978505430148E-005
AD derivative	-3.2779317070746363	0.18447826292607189
n=-8		
Variable	L	D
Difference	3.2257010396108399E-009	1.4172704398962566E-008
AD derivative	-3.2779317070746363	0.18447826292607189
n=-12		
Variable	L	D
Difference	5.5333838117421763E-004	4.0803766629127214E-005
AD derivative	-3.2779317070746363	0.18447826292607189

$$Gamad = \begin{bmatrix} -1.0 & 0.5 & 0.3 \end{bmatrix}$$

$$alpha0d = \begin{bmatrix} 0.0 & 0.1 & -0.1 \end{bmatrix}$$

$$chordsd = \begin{bmatrix} 1.0 & -0.2 & 0.7 \end{bmatrix}$$

The derivative seeds of the outputs resd are shown in table 7

Variable	Value
res	[-18.145367171877378 | 12.868133525028602 | 0.83612926486240313]

Tabela 7

Then, we executed the reverse AD version of get_residuals subroutine with this set of arbitrary derivative seeds:

$$resb = \begin{bmatrix} 0.1 & -0.2 & 0.3 \end{bmatrix}$$

The derivative seeds of the inputs are shown below:

$$\mathbf{Xb} = \begin{bmatrix} 0.2275863208268376 & -0.2712875360769555 & -4.075643259753733E-002 \\ -0.5468797907446982 & 1.047717341208687E-002 & 0.4061919203254720 \\ -8.445476534641688E-002 & -1.111073699998466 & -0.6901145428583320 \end{bmatrix}$$

$$Gamab = \begin{bmatrix} 0.92811004442459510 & -1.7460953612550254 & 2.6741631647326694 \end{bmatrix}$$

$$alpha0b = \begin{bmatrix} -0.54753307192204981 & 0.88659049941080670 & -1.6425992157661491 \end{bmatrix}$$

$$chordsb = \begin{bmatrix} 0.66592602973071957 & -0.61796730198934124 & 1.9977780891921586 \end{bmatrix}$$

A dot product test was performed by adding the contributions of the input variables and by subtracting the contributions of the output variables resulting in the following value:

$$dotprod = 0.0000000000000000 \tag{3.4}$$

Now we are going to apply the same procedure for the get_functions subroutine. The same input derivative seeds were used and the corresponding derivative seeds are shown in table 8.

Variable	Value
Srefd	0.92999999999999994
CLd	-21.062070460165867
CDd	-4.0731090226145108
Ld	-3.2779317070746372
Dd	0.18447826292607239

Tabela 8

Then we executed the reverse AD version of get_functions subroutine with this set of arbitrary derivative seeds: Srefb = 0.6, CLb = -0.2, CDb = 0.1, Lb = -0.3, Db = 0.7.

The derivative seed of the inputs are presented below:

$$\mathbf{Xb} = \begin{bmatrix} -0.13568646605128287 & 0.64811422559502074 & -0.48867248107070044 \\ 0.13568646605128309 & -0.25049809117159927 & 0.48867248107070044 \\ 0.13568646605128293 & 0.25049809117159949 & 0.48867248107070027 \end{bmatrix}$$

$$Gamab = \begin{bmatrix} -3.9918563024321063E-002 & -0.35304117698882054 & -3.9918563024321098E-002 \end{bmatrix}$$

$$alpha0b = \begin{bmatrix} 0.0000000000000000 & 0.0000000000000000 & 0.0000000000000000 \end{bmatrix}$$

$$chordsb = \begin{bmatrix} 3.0809134233924493 & 3.0809134233924493 & 3.0809134233924493 \end{bmatrix}$$

A dot product test was performed by adding the contributions of the input variables and by subtracting the contributions of the output variables resulting in the following value:

$$dotprod = -9.1593399531575415E - 016 \tag{3.5}$$