

MTH8408 : Méthodes d'optimisation et contrôle optimal

Laboratoire 2: Optimisation sans contraintes

Tangi Migot et Paul Raynaud

Travail par Adam Osmani 2026348

In []:

```
using Pkg
Pkg.activate(".") #Accède au fichier Project.toml
Pkg.instantiate()
Pkg.status()

Activating project at `c:\Users\adamo\OneDrive\Documents\POLY\H24\MTH8408\MTH8408-Hiv24\lab2`
Status `C:\Users\adamo\OneDrive\Documents\POLY\H24\MTH8408\MTH8408-Hiv24\lab2\Project.toml`  

[54578032] ADNLPModels v0.7.0
[7073ff75] IJulia v1.24.2
[40e66cde] LDLFactorizations v0.10.1
[a4795742] NLPModels v0.20.0
[5049e819] OptimizationProblems v0.7.3
[91a5bcd] Plots v1.39.0
[37e2e46d] LinearAlgebra
[de0858da] Printf
[8dfed614] Test
```

In []:

```
using ADNLPModels, LinearAlgebra, NLPModels, Printf
```

On pourra trouver de la documentation sur ADNLPModels et NLPModels ici:

- juliasmoothoptimizers.github.io/NLPModels.jl/dev/
- juliasmoothoptimizers.github.io/ADNLPModels.jl/dev/

In []:

```
# Problème test:
f(x) = x[1]^2 * (2*x[1] - 3) - 6*x[1]*x[2] * (x[1] - x[2] - 1) # fonction objectif
g(x) = 6 * [x[1]^2 - x[1] - 2*x[1]*x[2] + x[2]^2 + x[2]; -x[1]^2 + 2*x[1]*x[2] + x[2]^2]
H(x) = 6 * [2*x[1]-1-2*x[2] -2*x[1]+2*x[2]+1; -2*x[1]+2*x[2]+1 2*x[1]] # la Hessien
```

H (generic function with 1 method)

Exercice 1: Newton avec recherche linéaire - amélioration du code

Ci-dessous, vous avez le code de deux fonctions qui ont été vues dans le cours, la recherche linéaire qui satisfait Armijo, et une méthode de Newton avec cette recherche linéaire. Le but de ce laboratoire est d'implémenter d'autres méthodes utiles pour résoudre des problèmes de grandes dimensions.

```
In [ ]: #Amélioration possibles: return also the value of f
function armijo(xk, dk, fk, gk, f)
    slope = dot(gk, dk) #doit être <0
    t = 1.0
    while f(xk + t * dk) > fk + 1.0e-4 * t * slope
        t /= 1.5
    end
    return t
end

armijo (generic function with 1 method)
```

```
In [ ]: #Test pour vérifier que la fonction armijo fonctionne correctement.
using Test #le package Test définit (entre autre) la macro @test qui permet de faire
xk = ones(2)
gk = g(xk)
dk = - gk
fk = f(xk)
t = armijo(xk, dk, fk, gk, f)
@test t < 1
@test f(xk + t * dk) <= fk + 1.0e-4 * t * dot(gk,dk)

xk = [1.5, 0.5]
fk = f(xk)
gk = g(xk)
dk = - gk
t = armijo(xk, dk, fk, gk, f)
@test t < 1
@test f(xk + t * dk) <= fk + 1.0e-4 * t * dot(g(xk),dk)
```

Test Passed

```
In [ ]: function newton_armijo(f, g, H, x0; verbose::Bool = true)
    xk = x0 # xk initial
    fk = f(xk) # Fonction en xk
    gk = g(xk) # Gradient en xk
    gnorm = gnorm0 = norm(gk)
    k = 0
    verbose && @printf "%2s %9s %9s\n" "k" "fk" "||∇f(x)||"
    verbose && @printf "%2d %9.2e %9.1e\n" k fk gnorm
    while gnorm > 1.0e-6 + 1.0e-6 * gnorm0 && k < 100
        Hk = H(xk)
        dk = - Hk \ gk
        slope = dot(dk, gk)
        λ = 0.0
        while slope ≥ -1.0e-4 * norm(dk) * gnorm
            λ = max(1.0e-3, 10 * λ)
            dk = - ((Hk + λ * I) \ gk)
            slope = dot(dk, gk)
        end
        t = armijo(xk, dk, fk, gk, f)
        xk += t * dk
        fk = f(xk)
        gk = g(xk)
        gnorm = norm(gk)
        k += 1
        verbose && @printf "%2d %9.2e %9.1e %7.1e \n" k fk gnorm t
    end
    return xk
end
```

newton_armijo (generic function with 2 methods)

```
In [ ]: sol = newton_armijo(f, g, H, [.5, .5])
@test g(sol) ≈ zeros(2) atol = 1.0e-6
```

k	fk	∇f(x)
0	1.00e+00	4.5e+00
1	5.08e-02	8.4e-01 1.0e+00
2	4.73e-04	7.6e-02 1.0e+00
3	6.97e-08	9.1e-04 1.0e+00
4	1.62e-15	1.4e-07 1.0e+00

Test Passed

On veut améliorer le code de la fonction `newton_armijo` avec les ajouts suivants:

- Changer les paramètre d'entrées de la fonction pour un `nlp`
- Avant d'appeler la recherche linéaire, si `slope = dot(dk, gk)` est plus grand que $-1.0e-4 * \text{norm}(dk) * \text{gnorm}$, on modifie le système. On fait maximum 5 mise à jour de λ , sinon on prend l'opposé du gradient.

```
 $\lambda = 0.0$ 
while slope ≥ -1.0e-4 * norm(dk) * gnorm
    λ = max(1.0e-3, 10 * λ)
    dk = - ((Hk + λ * I) \ gk)
    slope = dot(dk, gk)
end
```

Ajouter un compteur sur le nombre de mises à jour de λ et ajuster `dk = - gk` si la limite est atteinte.

- On veut aussi détecter et éventuellement arrêter la boucle `while` si la fonction objectif `fk` devient trop petite/négative (inférieure à $-1e15$), i.e. le problème est non-bornée inférieurement.
- On veut ajouter deux critères d'arrêts supplémentaires:
 - un compteur sur le nombre d'évaluations de `f` (maximum 1000). Utiliser `neval_obj(nlp)`.
 - une limite de temps d'exécution, `max_time = 60.0`. Utiliser la fonction `time()`.
- Enfin, on voudrait aussi voir un message à l'écran si l'algorithme n'a pas trouvé la solution, i.e. il s'est arrêté à cause de la limite sur le nombre d'itérations, temps, évaluation de fonctions, problème non-borné.

In []:

```
#SOLUTION: fonction à modifier
function newton_armijo(nlp, x0; verbose::Bool = true)
    # TODO...
    xk = x0
    fk = obj(nlp, xk)
    gk = grad(nlp, xk)
    Hk = hess(nlp, xk)
    t0 = time()
    max_time = 60.0

    gnorm = gnorm0 = norm(gk)
    k = 0
    fit = 0;
    verbose && @printf "%2s %9s %9s\n" "k" "fk" "||∇f(x)||"
    verbose && @printf "%2d %9.2e %9.1e\n" k fk gnorm
    while gnorm > 1.0e-6 + 1.0e-6 * gnorm0 && k < 100
        Hk = hess(nlp, xk)
        dk = - Hk \ gk
        slope = dot(dk, gk)

        λ = 0.0
        λit = 0
        while slope ≥ -1.0e-4 * norm(dk) * gnorm
            if λit >= 5
                dk = -gk
                break
            end

            λ = max(1.0e-3, 10 * λ)
            dk = - ((Hk + λ * I) \ gk)
            slope = dot(dk, gk)
            λit += 1
        end

        t = armijo(xk, dk, fk, gk, f)
        xk += t * dk
        fk = obj(nlp, xk)
        gk = grad(nlp, xk)
        gnorm = norm(gk)
        k += 1
        verbose && @printf "%2d %9.2e %9.1e %7.1e \n" k fk gnorm t

        if fk < -1e15
            println("La fonction objective ne semble pas être bornée inférieure.")
            return xk
        end

        if neval_obj(nlp) >= 1000
            println("La fonction a dépassé le nombre maximal d'itérations.")
            return xk
        end

        if time() - t0 > max_time
            println("La fonction a dépassé le temps maximal.")
            return xk
        end
    end
    return xk
```

```
end
newton_armijo (generic function with 2 methods)
```

```
In [ ]: #Test
f(x) = x[1]^2 * (2*x[1] - 3) - 6*x[1]*x[2] * (x[1] - x[2] - 1)
x0 = [0.5, 0.5]
nlp = ADNLPMModel(f, x0)

sol = newton_armijo(nlp, x0)

@test grad(nlp,sol) ≈ zeros(2) atol = 1.0e-6

k      fk ||∇f(x)||
0  1.00e+00  4.5e+00
1  5.08e-02  8.4e-01 1.0e+00
2  4.73e-04  7.6e-02 1.0e+00
3  6.97e-08  9.1e-04 1.0e+00
4  1.62e-15  1.4e-07 1.0e+00
Test Passed
```

Exercice 2: LDLt-Newton avec recherche linéaire

On va maintenant modifier la méthode de Newton vu précédemment pour utiliser un package qui s'occupe de calculer une factorisation de la matrice hessienne tel que:

$$\nabla^2 f(x) = LDL^T.$$

Ce type de factorisation n'est possible que si la matrice hessienne est définie positive, dans le cas contraire on a besoin de régularisé le système comme dans l'exercice précédent.

Pour résoudre le système linéaire en utilisant cette factorisation, on va utiliser le package `LDLFactorizations` :

```
In [ ]: using LDLFactorizations, LinearAlgebra
```

Un tutoriel sur l'utilisation de `LDLFactorizations` est disponible sur la documentation du package sur github ou encore [à ce lien](#).

Voici un exemple d'utilisation de ce package. La matrice dont on veut calculer la factorisation doit être de type `Symmetric` .

```
In [ ]: A = ones(2,2) #cette matrice symétrique, mais pas du type Symmetric
         #à noter que cette matrice n'est pas définie positive.
typeof(A) <: Symmetric #false
A = Symmetric(A)
typeof(A) <: Symmetric #true :)
```

true

Deuxième étape, le package fait une phase d'analyse de la matrice avec `ldl_analyze` en créant une structure pratique pour les diverses fonctions du package.

```
In [ ]: A = -rand(2, 2)
sol = rand(2)
b = A*sol #on veut résoudre le système A*x=b

# LDLFactorizations va en réalité demander la matrice triangulaire supérieure
A = Symmetric(triu(A), :U)
S = ldl_analyze(A)
ldl_factorize!(A, S)
x = S \ b # x = A \b ça va être résolu par Julia
norm(A * x - b)

1.1102230246251565e-16
```

```
In [ ]: A = [0. 1.; 1. 0.]
```

```
2x2 Matrix{Float64}:
 0.0  1.0
 1.0  0.0
```

```
In [ ]: A = Symmetric(triu(A), :U)
S = ldl_analyze(A)
ldl_factorize!(A, S)
```

```
LDLFactorizations.LDLFactorization{Float64, Int64, Int64, Int64}(true, false, true,
2, [2, -1], [0, 0], [1, 2], [1, 2], [1, 2], [1, 2, 2], [1, 1, 1], Int64[], [4294967
297], [1.178770524161e-311], [0.0, 0.0], [0.0, 6.9528715494096e-310], [16, 0], 0.0,
0.0, 0.0, 2)
```

```
In [ ]: S.d
```

```
2-element Vector{Float64}:
 0.0
 0.0
```

La matrice A factorisée par LDL^T n'était pas forcément définie positive. On peut le voir sur les valeurs de D .

```
In [ ]: S.d #c'est le vecteur qui correspond à la matrice diagonale D.
```

```
2-element Vector{Float64}:
 0.0
 0.0
```

Pour l'optimisation, dans le cas où des valeurs de D sont négatives, i.e. `minimum(S.d) <= 0.`, on ajoutera une correction pour être sûr d'obtenir une direction de descente. On pourra choisir un des deux:

- `S.d = abs.(S.d)`
- `S.d .+= -minimum(S.d) + 1e-6`

Utiliser cette technique pour calculer la direction de descente:

```
In [ ]: # Solution: modifier le calcul de la direction avec LDLFactorizations
function newton_ldlt_armijo(nlp, x0; verbose::Bool = true)
    xk = x0
    fk = obj(nlp, xk)
    gk = grad(nlp, xk)
    gnorm = gnorm0 = norm(gk)
    k = 0
    verbose && @printf "%2s %9s %9s\n" "k" "fk" "||∇f(x)||"
    verbose && @printf "%2d %9.2e %9.1e\n" k fk gnorm
    while gnorm > 1.0e-6 + 1.0e-6 * gnorm0 && k < 100 && fk > -1e15
        Hk = Symmetric(triu(hess(nlp, xk)), :U)
        # ... TODO ...
        Sk = ldl_analyze(Hk)
        ldl_factorize!(Hk, Sk)
        dk = - Sk \ gk
        slope = dot(dk, gk)
        t = armijo(xk, dk, fk, gk, x -> obj(nlp, x))
        xk += t * dk
        fk = obj(nlp, xk)
        gk = grad(nlp, xk)
        gnorm = norm(gk)
        k += 1
        verbose && @printf "%2d %9.2e %9.1e %7.1e \n" k fk gnorm t
    end
    return xk
end

newton_ldlt_armijo (generic function with 1 method)
```

```
In [ ]: #Test
f(x) = x[1]^2 * (2*x[1] - 3) - 6*x[1]*x[2] * (x[1] - x[2] - 1)
nlp = ADNLPModel(f, [0.5, 0.5])

sol = newton_ldlt_armijo(nlp, x0)

@test grad(nlp,sol) ≈ zeros(2) atol = 1.0e-6
```

k	fk	∇f(x)
0	1.00e+00	4.5e+00
1	5.08e-02	8.4e-01 1.0e+00
2	4.73e-04	7.6e-02 1.0e+00
3	6.97e-08	9.1e-04 1.0e+00
4	1.62e-15	1.4e-07 1.0e+00

Test Passed

Exercice 3: Méthode quasi-Newton: BFGS

Méthode quasi-Newton: BFGS

Pour des problèmes de très grandes tailles, il est parfois très coûteux d'évaluer la hessienne du problème d'optimisation (et même le produit hessienne-vecteur). La famille des méthodes *quasi-Newton* construit une approximation B_k symétrique de la matrice Hessienne en utilisant seulement le gradient et en mesurant sa variation, et permet quand même d'améliorer significativement les performances comparé à la méthode du gradient.

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k).$$

Par ailleurs la matrice B_k est aussi construite de façon à ce que l'inverse soit connue, il n'y a donc pas de système linéaire à résoudre.

La méthode la plus connue dans la famille des méthodes quasi-Newton, est la méthode BFGS (Broyden - Fletcher, Goldfarb, and Shanno) où B_k est définie positive ($B_0 = \lambda I$, $\lambda > 0$). La formule suivante calcule l'inverse de B_k que l'on note H_k :

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \quad \rho_k = \frac{1}{y_k^T s_k}.$$

L'algorithme est presque le même que la méthode de Newton à la différence qu'il n'y a pas de système linéaire à résoudre et la direction d_k est à coup sûr une direction de descente. Ainsi la direction de descente est calculée comme suit:

$$d_k = -H_k \nabla f(x_k).$$

Comment choisir la matrice H_0 ? On peut éventuellement choisir I . Une alternative est d'utiliser $H_0 = I$ pour la première itération et ensuite mettre H_0 à jour avant de calculer H_1 en utilisant:

$$H_0 = \frac{y_k^T s_k}{y_k^T y_k} I.$$

Important: pour s'assurer que la matrice H_k reste définie positive à toutes les itérations, il faut s'assurer que $y_k^T s_k > 0$. C'est toujours vrai pour des fonctions convexes, mais pas nécessairement dans le cas général. On pourra tester ici la version "skip" qui ne mets pas à jour quand cette condition n'est pas vérifiée.

```
In [ ]: # Solution: copier-coller votre newton_armijo ici et modifier le calcul de la direc
function bfgs_quasi_newton_armijo(nlp, x0; verbose::Bool = true)
    xk = x0
    fk = obj(nlp, xk)
    gk = grad(nlp, xk)
    Hk = I
    t0 = time()
    max_time = 60.0

    gnorm = gnorm0 = norm(gk)
    k = 0
    fit = 0;
    verbose && @printf "%2s %9s %9s\n" "k" "fk" "||∇f(x)||"
    verbose && @printf "%2d %9.2e %9.1e\n" k fk gnorm
    while gnorm > 1.0e-6 + 1.0e-6 * gnorm0 && k < 100

        dk = - Hk * gk
        slope = dot(dk, gk)

        λ = 0.0
        λit = 0
        while slope ≥ -1.0e-4 * norm(dk) * gnorm
            if λit >= 5
                dk = -gk
                break
            end

            λ = max(1.0e-3, 10 * λ)
            dk = - ((Hk + λ * I) \ gk)
            slope = dot(dk, gk)
            λit += 1
        end

        t = armijo(xk, dk, fk, gk, f)
        xk0 = xk
        xk += t * dk
        fk = obj(nlp, xk)
        gk0 = gk
        gk = grad(nlp, xk)
        gnorm = norm(gk)

        sk = xk - xk0
        yk = gk - gk0
        pk = 1/(yk'*sk)

        if (k == 0)
            Hk = (yk'*sk)/(yk'*yk)*I
        end

        if (yk'*sk > 0)
            Hk = (I-pk*sk*yk')*Hk*(I-pk*yk*sk')+pk*sk*sk'
        end

        k += 1
        verbose && @printf "%2d %9.2e %9.1e %7.1e \n" k fk gnorm t

        if fk < -1e15
            break
        end
    end
end
```

```

    println( "La fonction objective ne semble pas etre bornee interieure. ")
    return xk
end

if neval_obj(nlp) >= 1000
    println("La fonction a dépassé le nombre maximal d'itérations.")
    return xk
end

if time()-t0 > max_time
    println("La fonction a dépassé le temps maximal.")
    return xk
end
end
return xk
end

```

bfgs_quasi_newton_armijo (generic function with 1 method)

In []:

```

#Test
f(x) = x[1]^2 * (2*x[1] - 3) - 6*x[1]*x[2] * (x[1] - x[2] - 1)
nlp = ADNLPModel(f, zeros(2))

sol = bfgs_quasi_newton_armijo(nlp, x0)

@test grad(nlp,sol) ≈ zeros(2) atol = 1.0e-6

```

k	fk	$\nabla f(x)$
0	1.00e+00	4.5e+00
1	3.33e-01	4.4e+00 3.0e-01
2	-4.00e-01	1.4e+00 1.0e+00
3	-6.13e-01	1.3e+00 1.0e+00
4	-9.01e-01	1.4e+00 6.7e-01
5	-9.82e-01	6.7e-01 1.0e+00
6	-9.98e-01	1.2e-01 1.0e+00
7	-9.99e-01	4.8e-02 1.0e+00
8	-1.00e+00	1.8e-03 1.0e+00
9	-1.00e+00	1.3e-04 1.0e+00
10	-1.00e+00	9.0e-06 1.0e+00
11	-1.00e+00	3.1e-09 1.0e+00

Test Passed

Exercice 4: application à un problème de grande taille

On va ajouter le package `OptimizationProblems` qui contient, comme son nom l'indique, une collection de problème d'optimisation disponible au format de `JuMP` (dans le sous-module `OptimizationProblems.PureJuMP`) et de `ADNLPModel` (dans le sous-module `OptimizationProblems.ADNLPProblems`).

In []:

```
using ADNLPModels, OptimizationProblems.ADNLPProblems # Attention si vous ne faites
```

In []:

```
n = 500
model = genrose(n=n)
@test typeof(model) <: ADNLPModel
```

Test Passed

Si vous le souhaitez, il est possible d'accéder à certaines informations sur le problème en accédant à son meta:

```
In [ ]: using OptimizationProblems  
OptimizationProblems.genrose_meta
```

```
Dict{Symbol, Any} with 17 entries:  
:has_equalities_only => false  
:origin => :unknown  
:has_inequalities_only => false  
:defined_everywhere => missing  
:has_fixed_variables => false  
:variable_ncon => false  
:nvar => 100  
:is_feasible => true  
:minimize => true  
:ncon => 0  
:name => "genrose"  
:best_known_lower_bound => -Inf  
:objtype => :other  
:best_known_upper_bound => 405.106  
:has_bounds => false  
:variable_nvar => true  
:contype => :unconstrained
```

Il est aussi possible d'accéder au meta de tous les problèmes

```
In [ ]: OptimizationProblems.meta
```

372×17 DataFrame

Row	nvar	variable_nvar	ncon	variable_ncon	minimize	name	has_equalities_only	ha
	Int64	Bool	Int64	Bool	Bool	String	Bool	Bo
1	1	false	0	false	true	AMPGO02		false
2	1	false	0	false	true	AMPGO03		false
3	1	false	0	false	true	AMPGO04		false
4	1	false	0	false	true	AMPGO05		false
5	1	false	0	false	true	AMPGO06		false
6	1	false	0	false	true	AMPGO07		false
7	1	false	0	false	true	AMPGO08		false
8	1	false	0	false	true	AMPGO09		false
9	1	false	0	false	true	AMPGO10		false
10	1	false	0	false	true	AMPGO11		false
11	1	false	0	false	true	AMPGO12		false
12	1	false	0	false	true	AMPGO13		false
13	1	false	0	false	true	AMPGO14		false
:	:	:	:	:	:	:	:	:
361	100	true	0	false	true	tointgss		false
362	100	true	0	false	true	tquartic		false
363	8	false	3	false	true	triangle		false
364	2244	false	1896	false	true	triangle_deer		false
365	1366	false	1182	false	true	triangle_pacman		false
366	4444	false	4025	false	true	triangle_turtle		false
367	100	true	0	false	true	tridia		false
368	100	true	0	false	true	vardim		false
369	8	false	0	false	true	vibrbeam		false
370	31	false	0	false	true	watson		false
371	100	true	0	false	true	woods		false
372	3	false	3	false	true	zangwil3		true

Résoudre le problème `genrose` et un autre problème de la collection en utilisant vos algorithmes précédents. Avant d'utiliser l'algorithme on testera que le problème est bien sans contrainte avec:

```
In [ ]: unconstrained(nlp) #qui retourne vrai si `nlp` est un problème sans contraintes.
```

```
true
```

```
In [ ]: # Use previous functions to solve genrose.  
x0 = zeros(500)  
sol = bfgs_quasi_newton_armijo(model, x0)  
  
@test grad(model,sol) ≈ zeros(2) atol = 1.0e-6
```

k	fk	∇f(x)
0	5.00e+02	4.5e+01
1	2.01e+05	2.7e+04 1.0e+00
2	4.97e+02	3.0e+01 1.0e+00
3	4.96e+02	2.0e+01 1.0e+00
4	4.95e+02	2.4e+00 1.0e+00
5	4.95e+02	2.2e+00 1.0e+00
6	4.95e+02	3.0e+00 1.0e+00
7	4.95e+02	4.4e+00 1.0e+00
8	4.95e+02	7.9e+00 1.0e+00
9	4.95e+02	8.8e+00 1.0e+00
10	4.94e+02	1.2e+01 1.0e+00
11	4.94e+02	5.7e+00 1.0e+00
12	4.94e+02	5.5e+00 1.0e+00
13	4.94e+02	1.6e+01 1.0e+00
14	4.94e+02	5.5e+00 1.0e+00
15	4.94e+02	5.0e+00 1.0e+00
16	4.95e+02	3.7e+01 1.0e+00
17	4.94e+02	4.2e+00 1.0e+00
18	4.93e+02	4.1e+00 1.0e+00
19	2.50e+03	3.4e+03 1.0e+00
20	4.93e+02	2.6e+00 1.0e+00
21	4.93e+02	2.5e+00 1.0e+00
22	4.93e+02	4.0e+00 1.0e+00
23	4.93e+02	4.8e+00 1.0e+00
24	4.93e+02	7.6e+00 1.0e+00
25	6.48e+03	6.9e+03 1.0e+00
26	4.93e+02	6.9e+00 1.0e+00
27	4.93e+02	6.4e+00 1.0e+00
28	4.93e+02	4.1e+00 1.0e+00
29	4.93e+02	3.6e+00 1.0e+00
30	4.93e+02	2.7e+00 1.0e+00
31	4.93e+02	2.7e+00 1.0e+00
32	4.93e+02	3.7e+00 1.0e+00
33	4.93e+02	5.5e+00 1.0e+00
34	4.93e+02	8.8e+00 1.0e+00
35	4.93e+02	1.0e+01 1.0e+00
36	4.92e+02	9.9e+00 1.0e+00
37	4.92e+02	9.7e+00 1.0e+00
38	4.92e+02	1.3e+01 1.0e+00
39	4.92e+02	7.4e+00 1.0e+00
40	4.91e+02	1.1e+01 1.0e+00
41	4.91e+02	1.0e+01 1.0e+00
42	4.91e+02	9.4e+00 1.0e+00
43	4.90e+02	1.5e+01 1.0e+00
44	4.91e+02	2.0e+01 1.0e+00
45	4.90e+02	8.6e+00 1.0e+00
46	4.90e+02	8.8e+00 1.0e+00
47	4.90e+02	8.1e+00 1.0e+00
48	4.90e+02	1.6e+01 1.0e+00
49	4.89e+02	1.4e+01 1.0e+00
50	4.89e+02	8.3e+00 1.0e+00
51	4.89e+02	8.8e+00 1.0e+00
52	4.89e+02	7.6e+00 1.0e+00
53	4.89e+02	1.9e+01 1.0e+00
54	4.88e+02	1.6e+01 1.0e+00
55	4.88e+02	6.8e+00 1.0e+00
56	4.88e+02	8.1e+00 1.0e+00
57	4.88e+02	8.2e+00 1.0e+00

```
58 4.87e+02 1.7e+01 1.0e+00
59 4.87e+02 2.0e+01 1.0e+00
60 4.87e+02 8.9e+00 1.0e+00
61 4.87e+02 8.7e+00 1.0e+00
62 4.87e+02 7.8e+00 1.0e+00
63 4.86e+02 1.3e+01 1.0e+00
64 4.86e+02 1.5e+01 1.0e+00
65 4.86e+02 2.1e+01 1.0e+00
66 4.86e+02 8.0e+00 1.0e+00
67 4.86e+02 9.7e+00 1.0e+00
68 4.85e+02 8.3e+00 1.0e+00
69 4.85e+02 1.2e+01 1.0e+00
70 4.85e+02 1.1e+01 1.0e+00
71 4.85e+02 1.6e+01 1.0e+00
72 4.84e+02 1.1e+01 1.0e+00
73 4.84e+02 1.3e+01 1.0e+00
74 4.84e+02 1.5e+01 1.0e+00
75 4.84e+02 9.9e+00 1.0e+00
76 4.83e+02 1.1e+01 1.0e+00
77 4.83e+02 1.4e+01 1.0e+00
78 4.83e+02 2.2e+01 1.0e+00
79 4.83e+02 1.1e+01 1.0e+00
80 4.83e+02 1.2e+01 1.0e+00
81 4.83e+02 8.7e+00 1.0e+00
82 4.83e+02 1.6e+01 1.0e+00
83 4.82e+02 1.1e+01 1.0e+00
84 4.82e+02 6.2e+00 1.0e+00
85 4.82e+02 1.7e+01 1.0e+00
86 4.82e+02 9.5e+00 1.0e+00
87 4.81e+02 7.7e+00 1.0e+00
88 4.84e+02 4.0e+01 1.0e+00
89 4.81e+02 4.3e+00 1.0e+00
90 4.81e+02 5.1e+00 1.0e+00
91 5.14e+02 1.8e+02 1.0e+00
92 4.81e+02 3.3e+00 1.0e+00
93 4.81e+02 2.9e+00 1.0e+00
94 4.82e+02 3.9e+01 1.0e+00
95 4.81e+02 3.4e+00 1.0e+00
96 4.81e+02 3.4e+00 1.0e+00
97 4.80e+02 3.0e+00 1.0e+00
98 6.66e+06 1.3e+06 1.0e+00
99 4.80e+02 3.1e+00 1.0e+00
100 4.80e+02 2.9e+00 1.0e+00
Error During Test at c:\Users\adamo\OneDrive\Documents\POLY\H24\MTH8408\MTH8408-Hiv
24\lab2\Lab2-notebook.ipynb:5
    Test threw exception
    Expression: ~(grad(model, sol), zeros(2), atol = 1.0e-6)
    DimensionMismatch: dimensions must match: a has dims (Base.OneTo(500),), b has di
ms (Base.OneTo(2),), mismatch at 1
    Stacktrace:
    [1] promote_shape
        @ .\indices.jl:178 [inlined]
    [2] promote_shape
        @ .\indices.jl:169 [inlined]
    [3] -(A::Vector{Float64}, B::Vector{Float64})
        @ Base .\arraymath.jl:7
    [4] isapprox(x::Vector{Float64}, y::Vector{Float64}; atol::Float64, rtol::Float6
4, nans::Bool, norm::typeof(norm))
        @ LinearAlgebra C:\Users\adamo\.julia\julia-1.10.0+0.x64.w64.mingw32\s
```

```
hare\julia\stdlib\v1.10\LinearAlgebra\src\generic.jl:1789
[5] eval_test(evaluated::Expr, quoted::Expr, source::LineNumberNode, negate::Bool)
    @ Test C:\Users\adamo\.julia\juliaup\julia-1.10.0+0.x64.w64.mingw32\share\julia\stdlib\v1.10\Test\src\Test.jl:367
[6] macro expansion
    @ C:\Users\adamo\.julia\juliaup\julia-1.10.0+0.x64.w64.mingw32\share\julia\stdlib\v1.10\Test\src\Test.jl:669 [inlined]
[7] top-level scope
    @ c:\Users\adamo\OneDrive\Documents\POLY\H24\MTH8408\MTH8408-Hiv24\lab2\Lab2-notebook.ipynb:5
Test.FallbackTestSetException("There was an error during testing")
```

Stacktrace:

```
[1] record(ts::Test.FallbackTestSet, t::Union{Test.Error, Test.Fail})
    @ Test C:\Users\adamo\.julia\juliaup\julia-1.10.0+0.x64.w64.mingw32\share\julia\stdlib\v1.10\Test\src\Test.jl:1000
[2] do_test(result::Test.ExecutionResult, orig_expr::Any)
    @ Test C:\Users\adamo\.julia\juliaup\julia-1.10.0+0.x64.w64.mingw32\share\julia\stdlib\v1.10\Test\src\Test.jl:705
[3] macro expansion
    @ C:\Users\adamo\.julia\juliaup\julia-1.10.0+0.x64.w64.mingw32\share\julia\stdlib\v1.10\Test\src\Test.jl:672 [inlined]
[4] top-level scope
    @ c:\Users\adamo\OneDrive\Documents\POLY\H24\MTH8408\MTH8408-Hiv24\lab2\Lab2-notebook.ipynb:5
```

```
In [ ]: using ADNLPModels, OptimizationProblems.ADNLPProblems
vibrbeam_model = vibrbeam()

x0 = zeros(8)
sol = bfgs_quasi_newton_armijo(vibrbeam_model, x0)
@test grad(vibrbeam_model, sol) ≈ zeros(2) atol = 1.0e-6
```

k	fk	∇f(x)
0	7.58e+00	6.1e+04
1	3.20e+10	2.5e+15 1.8e-05
2	3.59e+10	1.9e+15 4.3e-20
3	2.87e+10	1.5e+15 1.0e+00
4	2.74e+09	1.7e+13 1.0e+00
5	1.25e+09	9.6e+13 1.0e+00
6	5.98e+08	1.6e+13 1.0e+00
7	1.48e+08	1.2e+13 1.0e+00
8	1.28e+05	1.1e+10 1.0e+00
9	1.96e+04	1.5e+09 1.0e+00
10	4.35e+01	3.8e+06 1.0e+00
11	7.46e+00	4.3e+04 1.0e+00
12	7.45e+00	3.7e+04 1.0e+00
13	7.45e+00	3.7e+04 1.0e+00
14	7.45e+00	3.7e+04 1.0e+00
15	7.45e+00	3.7e+04 1.0e+00
16	7.45e+00	3.7e+04 1.0e+00
17	7.45e+00	3.7e+04 1.0e+00
18	7.45e+00	3.7e+04 1.0e+00
19	7.45e+00	3.7e+04 1.0e+00
20	7.45e+00	3.7e+04 1.0e+00
21	7.45e+00	3.7e+04 1.0e+00
22	7.45e+00	3.7e+04 1.0e+00
23	7.45e+00	3.6e+04 1.0e+00
24	7.45e+00	3.6e+04 1.0e+00
25	7.45e+00	3.5e+04 1.0e+00
26	7.45e+00	3.6e+04 1.0e+00
27	7.44e+00	3.9e+04 1.0e+00
28	7.42e+00	4.4e+04 1.0e+00
29	7.40e+00	3.3e+04 1.0e+00
30	7.39e+00	1.8e+04 1.0e+00
31	7.39e+00	4.7e+03 1.0e+00
32	7.39e+00	1.6e+03 1.0e+00
33	7.39e+00	1.5e+03 1.0e+00
34	7.39e+00	1.5e+03 1.0e+00
35	4.45e+19	4.4e+23 1.0e+00
36	6.08e+19	4.4e+24 1.5e-03
37	5.30e+53	4.8e+58 1.0e+00
38	1.11e+121	5.1e+125 1.0e+00
39	1.11e+121	5.1e+125 3.0e-85
40	1.11e+121	5.1e+125 3.0e-85
41	1.11e+121	5.1e+125 3.0e-85
42	1.11e+121	5.1e+125 3.0e-85
43	1.11e+121	5.1e+125 3.0e-85
44	1.11e+121	5.1e+125 3.0e-85
45	1.11e+121	5.1e+125 3.0e-85
46	1.11e+121	5.1e+125 3.0e-85
47	1.11e+121	5.1e+125 3.0e-85
48	1.11e+121	5.1e+125 3.0e-85
49	1.11e+121	5.1e+125 3.0e-85
50	1.11e+121	5.1e+125 3.0e-85
51	1.11e+121	5.1e+125 3.0e-85
52	1.11e+121	5.1e+125 3.0e-85
53	1.11e+121	5.1e+125 3.0e-85
54	1.11e+121	5.1e+125 3.0e-85
55	1.11e+121	5.1e+125 3.0e-85
56	1.11e+121	5.1e+125 3.0e-85
57	1.11e+121	5.1e+125 3.0e-85

```
58 1.11e+121 5.1e+125 3.0e-85
59 1.11e+121 5.1e+125 3.0e-85
60 1.11e+121 5.1e+125 3.0e-85
61 1.11e+121 5.1e+125 3.0e-85
62 1.11e+121 5.1e+125 3.0e-85
63 1.11e+121 5.1e+125 3.0e-85
64 1.11e+121 5.1e+125 3.0e-85
65 1.11e+121 5.1e+125 3.0e-85
66 1.11e+121 5.1e+125 3.0e-85
67 1.11e+121 5.1e+125 3.0e-85
68 1.11e+121 5.1e+125 3.0e-85
69 1.11e+121 5.1e+125 3.0e-85
70 1.11e+121 5.1e+125 3.0e-85
71 1.11e+121 5.1e+125 3.0e-85
72 1.11e+121 5.1e+125 3.0e-85
73 1.11e+121 5.1e+125 3.0e-85
74 1.11e+121 5.1e+125 3.0e-85
75 1.11e+121 5.1e+125 3.0e-85
76 1.11e+121 5.1e+125 3.0e-85
77 1.11e+121 5.1e+125 3.0e-85
78 1.11e+121 5.1e+125 3.0e-85
79 1.11e+121 5.1e+125 3.0e-85
80 1.11e+121 5.1e+125 3.0e-85
81 1.11e+121 5.1e+125 3.0e-85
82 1.11e+121 5.1e+125 3.0e-85
83 1.11e+121 5.1e+125 3.0e-85
84 1.11e+121 5.1e+125 3.0e-85
85 1.11e+121 5.1e+125 3.0e-85
86 1.11e+121 5.1e+125 3.0e-85
87 1.11e+121 5.1e+125 3.0e-85
88 1.11e+121 5.1e+125 3.0e-85
89 1.11e+121 5.1e+125 3.0e-85
90 1.11e+121 5.1e+125 3.0e-85
91 1.11e+121 5.1e+125 3.0e-85
92 1.11e+121 5.1e+125 3.0e-85
93 1.11e+121 5.1e+125 3.0e-85
94 1.11e+121 5.1e+125 3.0e-85
95 1.11e+121 5.1e+125 3.0e-85
96 1.11e+121 5.1e+125 3.0e-85
97 1.11e+121 5.1e+125 3.0e-85
98 1.11e+121 5.1e+125 3.0e-85
99 1.11e+121 5.1e+125 3.0e-85
100 1.11e+121 5.1e+125 3.0e-85
Error During Test at c:\Users\adamo\OneDrive\Documents\POLY\H24\MTH8408\MTH8408-Hiv
24\lab2\Lab2-notebook.ipynb:6
    Test threw exception
    Expression: ~(grad(vibrbeam_model, sol), zeros(2), atol = 1.0e-6)
    DimensionMismatch: dimensions must match: a has dims (Base.OneTo(8),), b has dims
(Base.OneTo(2),), mismatch at 1
    Stacktrace:
    [1] promote_shape
        @ .\indices.jl:178 [inlined]
    [2] promote_shape
        @ .\indices.jl:169 [inlined]
    [3] -(A::Vector{Float64}, B::Vector{Float64})
        @ Base .\arraymath.jl:7
    [4] isapprox(x::Vector{Float64}, y::Vector{Float64}; atol::Float64, rtol::Float6
4, nans::Bool, norm::typeof(norm))
        @ LinearAlgebra C:\Users\adamo\.julia\julia-1.10.0+0.x64.w64.mingw32\s
```

```
hare\julia\stdlib\v1.10\LinearAlgebra\src\generic.jl:1789
[5] eval_test(evaluated::Expr, quoted::Expr, source::LineNumberNode, negate::Bool)
    @ Test C:\Users\adamo\.julia\juliaup\julia-1.10.0+0.x64.w64.mingw32\share\julia\stdlib\v1.10\Test\src\Test.jl:367
[6] macro expansion
    @ C:\Users\adamo\.julia\juliaup\julia-1.10.0+0.x64.w64.mingw32\share\julia\stdlib\v1.10\Test\src\Test.jl:669 [inlined]
[7] top-level scope
    @ c:\Users\adamo\OneDrive\Documents\POLY\H24\MTH8408\MTH8408-Hiv24\lab2\Lab2-notebook.ipynb:6
Test.FallbackTestSetException("There was an error during testing")
```

Stacktrace:

```
[1] record(ts::Test.FallbackTestSet, t::Union{Test.Error, Test.Fail})
    @ Test C:\Users\adamo\.julia\juliaup\julia-1.10.0+0.x64.w64.mingw32\share\julia\stdlib\v1.10\Test\src\Test.jl:1000
[2] do_test(result::Test.ExecutionResult, orig_expr::Any)
    @ Test C:\Users\adamo\.julia\juliaup\julia-1.10.0+0.x64.w64.mingw32\share\julia\stdlib\v1.10\Test\src\Test.jl:705
[3] macro expansion
    @ C:\Users\adamo\.julia\juliaup\julia-1.10.0+0.x64.w64.mingw32\share\julia\stdlib\v1.10\Test\src\Test.jl:672 [inlined]
[4] top-level scope
    @ c:\Users\adamo\OneDrive\Documents\POLY\H24\MTH8408\MTH8408-Hiv24\lab2\Lab2-notebook.ipynb:6
```