INSTITUTO TECNOLÓGICO DE AERONÁUTICA

MP-288 - Exercises on Numerical Line Search

Prof.: Rafael T. L. Ferreira

Aluno : Guilherme de Aquino Pereira Nunes

1) Consider the function $f(\mathbf{x}) = f(x_1, x_2) = 3(x_1 - 2)^2 + 3(x_2 - 3)^2 - 6x_1$. Find the minimum point of $f(\mathbf{x})$ along the direction $\mathbf{d}^0 = \{0.75, 0.5\}$ starting from the point $\mathbf{x}^0 = \{1.20, 1.50\}$.

Use line search methods with constant step function sampling, as proposed in the slides, with Phase I and both the Phase II there shown.

Use the golden section method.

The solution incertainty required for all the methods is $I = 2 \times 10^{-4}$. Choose your favourite software for iterations visualization. Compare methods for the same initial search step δ.

Escrevemos a função :

$f(x) := 3\cdot(x_1 - 2)^2 + 3\cdot(x_2 - 3)^2 - 6\cdot x_1$

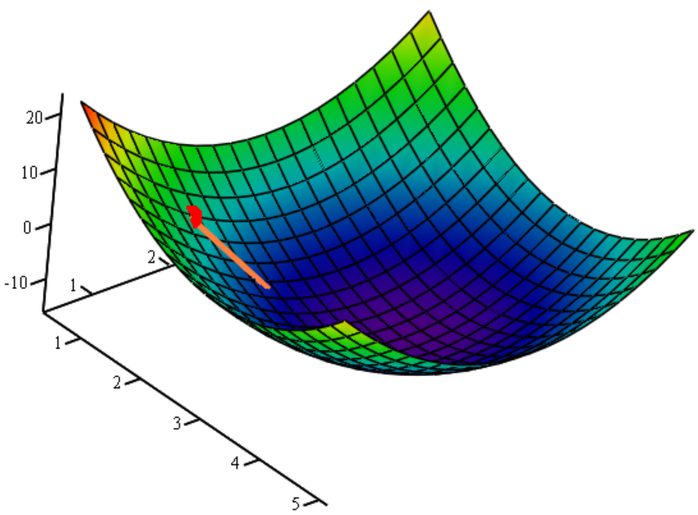
$f^*(x_1, x_2) := f\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right)$

$d0 := \begin{pmatrix} 0.75 \\ 0.5 \end{pmatrix}$

$I := 2 \times 10^{-4}$

$x0 := \begin{pmatrix} 1.2 \\ 1.5 \end{pmatrix}$

$O := \begin{pmatrix} 0 \\ 0 \end{pmatrix}$



f^*, X_0, D_0

Definimos variaveis de entrada para as funções a seguir :

$\delta := 0.1$

$n_{iter} := 100$

$\alpha''_l := 0$

$\alpha''_u := 4$

$\text{LineSearch\_PIa}(f, x_k, d_k, \delta, n_{iter}) :=$

```
"Error handling for return code :"
rc ← ERROR
"Iteration variable :"
Total_iter ← 0
"Calculate first function values:"
q ← 1
x_{q-1} ← x_k + (q − 1)·δ·d_k
x_q ← x_k + q·δ·d_k
f*_{q-1} ← f(x_{q-1})
f*_q ← f(x_q)
Total_iter ← 2
"Start moving along direction dk searching for an inflexion:"
while q < n_iter
    x_{q+1} ← x_k + (q + 1)·δ·d_k
    f*_{q+1} ← f(x_{q+1})
    Total_iter ← Total_iter + 1
    if f*_{q-1} ≥ f*_q ∧ f*_q < f*_{q+1}
        "Register answers:"
        (α_l  α_u  Total_iter  rc) ← [(q − 1)·δ  (q + 1)·δ  Total_iter  "ok"]
        "Finish While Looping:"
        q ← n_iter
    otherwise
        x_{q-1} ← x_q
        x_q ← x_{q+1}
        f*_{q-1} ← f*_q
        f*_q ← f*_{q+1}
        q ← q + 1
"Final answer:"
(α_l  α_u  Total_iter  rc)^T
```

$$\text{LineSearch\_PIa}(f, x0, d0, \delta, n_{iter}) = \begin{pmatrix} 2.5 \\ 2.7 \\ 28 \\ \text{"ok"} \end{pmatrix}$$

$$\text{LineSearch\_PIb}(f, x_k, d_k, \delta, \alpha_l, \alpha_u) :=$$

```
"Error handling and Return Code"
rc ← ERROR
"Iteration variable :"
Total_iter ← 0
"Calculate the number of iterations :"
n_iter ← ceil( |α_u − α_l| / δ ) + 1
"Recalculate delta :"
δ ← |α_u − α_l| / (n_iter − 1)
"Calculate first function values:"
q ← 1
x_q-1 ← x_k + [α_l + (q − 1)·δ]·d_k
x_q ← x_k + (α_l + q·δ)·d_k
f*_q-1 ← f(x_q-1)
f*_q ← f(x_q)
Total_iter ← 2
"Start moving along direction dk searching for an inflexion :"
while q < n_iter
    x_q+1 ← x_k + [α_l + (q + 1)·δ]·d_k
    f*_q+1 ← f(x_q+1)
    Total_iter ← Total_iter + 1
    if f*_q-1 ≥ f*_q ∧ f*_q < f*_q+1
        "Register answers:"
        (α_l  α_u  Total_iter  rc) ← [α_l + (q − 1)·δ ; α_l + (q + 1)·δ ; Total_iter ; "ok"]^T
        "Finish While Looping:"
        q ← n_iter
    otherwise
        x_q-1 ← x_q
        x_q ← x_q+1
        f*_q-1 ← f*_q
        f*_q ← f*_q+1
        q ← q + 1
"Function Output :"
(α_l  α_u  Total_iter  rc)^T
```

$$\text{LineSearch\_PIb}(f, x0, d0, \delta, \alpha''_l, \alpha''_u) = \begin{pmatrix} 2.5 \\ 2.7 \\ 28 \\ \text{"ok"} \end{pmatrix}$$

```
LineSearch_PIIa(f, x_k, d_k, α_l, α_u, I, n_iter) :=
    "Error handling for return code :"
    rc ← ERROR
    "Declare variable to count the total iteration :"
    Local_iter ← 0
    Total_iter ← 0
    "Start loopings :"
    for j ∈ 1 .. 1000
        "Calculate I :"
        I' ← α_u − α_l
        "Check if it respect the I tolerance:"
        if I' ≤ I
            "Stop looping:"
            rc ← "ok"
            break
        otherwise
            "Calculate delta :"
            δ ← (α_u − α_l) / (n_iter − 1)
            "Search for the inflexion point using Phase I method :"
            (α_l, α_u, Local_iter, rc)^T ← LineSearch_PIb(f, x_k, d_k, δ, α_l, α_u)
            "Check for error :"
            if rc = ERROR
                "Return error :"
                return Error(1,4)^T
            "Update counter variables :"
            Total_iter ← Total_iter + Local_iter
    "Calculate alfa * :"
    α* ← (α_l + α_u) / 2
    "Function Output :"
    (α*  I'  Total_iter  rc)^T
```

$$\begin{pmatrix} \alpha^* \\ I' \\ Total_{iter} \\ rc \end{pmatrix} := LineSearch\_PIIa\left(f, x0, d0, \alpha''_l, \alpha''_u, I, n_{iter}\right) \qquad \begin{pmatrix} \alpha^* \\ I' \\ Total_{iter} \\ rc \end{pmatrix} = \begin{pmatrix} 2.6 \\ 0 \\ 166 \\ \text{"ok"} \end{pmatrix}$$

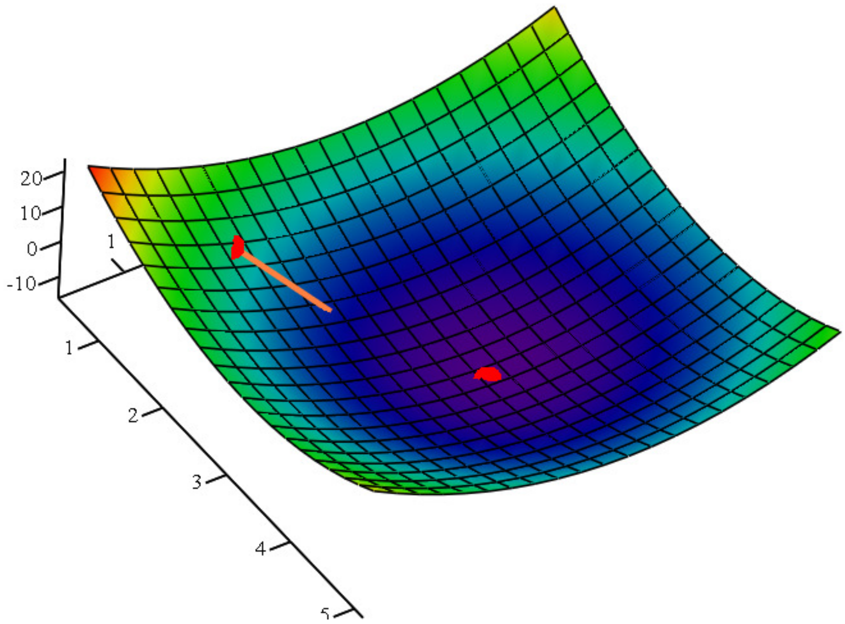
```
LineSearch_PIIb(f, x_k, d_k, α_l, α_u, I, n_iter) :=
| "Error handling for return code :"
| rc ← ERROR
| "Declare variable to count the total iteration :"
| Total_iter ← 0
| "Start loopings :"
| for j ∈ 1 .. 1000
|   | "Calculate I :"
|   | I' ← α_u − α_l
|   | "Check if it respect the I tolerance:"
|   | if I' ≤ I
|   |   | "Stop looping :"
|   |   | rc ← "ok"
|   |   | break
|   | otherwise
|   |   | "Calculate alfa a and alfa b :"
|   |   | α_a ← α_l + (α_u − α_l)·1/3
|   |   | α_b ← α_l + (α_u − α_l)·2/3
|   |   | "Calculate the x values for each alfa :"
|   |   | xα_a ← x_k + α_a·d_k
|   |   | xα_b ← x_k + α_b·d_k
|   |   | "Calculate the values for alfas :"
|   |   | f*α_a ← f(xα_a)
|   |   | f*α_b ← f(xα_b)
|   |   | "Increase counter :"
|   |   | Total_iter ← Total_iter + 2
|   |   | "Compare values :"
|   |   | if f*α_a < f*α_b
|   |   |   | "Change alfa u :"
|   |   |   | α_u ← α_b
|   |   | otherwise
|   |   |   | "Change alfa l :"
|   |   |   | α_l ← α_a
|   |   | "Check for error :"
|   |   | if f*α_a = f*α_b
|   |   |   | "Return error :"
|   |   |   | return Error(1, 4)^T
| "Calculate alfa * :"
| α* ← (α_l + α_u)/2
| "Function Output :"
| (α*  I'  Total_iter  rc)^T
```

$$\begin{pmatrix} \alpha^* \\ I' \\ Total_{iter} \\ rc \end{pmatrix} := \text{LineSearch\_PIIb}\left(f, x0, d0, \alpha''_l, \alpha''_u, I, n_{iter}\right) \qquad \begin{pmatrix} \alpha^* \\ I' \\ Total_{iter} \\ rc \end{pmatrix} = \begin{pmatrix} 2.6 \\ 0 \\ 50 \\ \text{"ok"} \end{pmatrix}$$

O método alternativo se mostrou mais eficiente, foi necessário calcular a função objetiva 50 vezes enquanto que o primeiro método foi necessário apenas 166 vezes.

Plotamos o ponto de mínimo ao longo da direção especificada:

$$x^* := x0 + \alpha^* \cdot d0 \qquad P^* := \begin{bmatrix} \left(x^*_1\right) \\ \left(x^*_2\right) \\ \left(f(x^*)\right) \end{bmatrix}$$



f^*, X_0, D_0, P^*

Através da razão áurea, temos as funções a seguir :

$\text{GoldenSearch\_PIa}(f, x_k, d_k, \delta, n_{iter}) :=$

```
"Error handling for return code :"
rc ← ERROR
"Golden ratio :"
G_ratio ← (√5 + 1)/2
"Iteration variable :"
Total_iter ← 0
"Calculate first function values:"
q ← 1
"Alfas :"
α_{q-1} ← Σ_{j=0}^{q-1} (δ·G_ratio^j)
α_q ← Σ_{j=0}^{q} (δ·G_ratio^j)
"x values :"
x_{q-1} ← x_k + α_{q-1}·d_k
x_q ← x_k + α_q·d_k
"Function values :"
f*_{q-1} ← f(x_{q-1})
f*_q ← f(x_q)
"Update counter :"
Total_iter ← 2
"Start moving along direction dk searching for an inflexion:"
while q < n_iter
    α_{q+1} ← Σ_{j=0}^{q+1} (δ·G_ratio^j)
    x_{q+1} ← x_k + α_{q+1}·d_k
    f*_{q+1} ← f(x_{q+1})
    Total_iter ← Total_iter + 1
    if f*_{q-1} ≥ f*_q ∧ f*_q < f*_{q+1}
        "Register answers:"
        (α_l  α_u  Total_iter  rc) ← (α_{q-1}  α_{q+1}  Total_iter  "ok")
        "Finish While Looping:"
        q ← n_iter
    otherwise
        α_{q-1} ← α_q
        α_q ← α_{q+1}
        x_{q-1} ← x_q
```

```
        x_q ← x_{q+1}
        f*_{q-1} ← f*_q
        f*_q ← f*_{q+1}
        q ← q + 1
"Final answer:"
(α_l  α_u  Total_iter  rc)^T
```

$$\text{GoldenSearch\_PIa}(f, x0, d0, \delta, n_{iter}) = \begin{pmatrix} 1.6 \\ 4.5 \\ 7 \\ \text{"ok"} \end{pmatrix}$$

GoldenSearch_PII$(f, x_k, d_k, \alpha_l, \alpha_u, I)$:=

```
"Error handling for return code :"
rc ← ERROR
"Declare variable to count the total iteration :"
Total_iter ← 0
"Golden ratio :"
G_ratio ← (√5 + 1) / 2
τ ← 1 / G_ratio
"Calculate the first values :"
"Alfas:"
α_a ← α_l + (1 − τ)·(α_u − α_l)
α_b ← α_l + τ·(α_u − α_l)
"x values :"
xα_a ← x_k + α_a·d_k
xα_b ← x_k + α_b·d_k
"Calculate the values for f :"
f*α_a ← f(xα_a)
f*α_b ← f(xα_b)
"Increase counter :"
Total_iter ← 2
"Start loopings :"
for j ∈ 1 .. 1000
    "Calculate I :"
    I' ← α_u − α_l
    "Check if it respect the I tolerance:"
    if I' ≤ I
        "Stop looping :"
        rc ← "ok"
        break
    otherwise
        if f*α_a < f*α_b
            "Change alfa u :"
```

"Change alfa u :"

$\alpha_u \leftarrow \alpha_b$

"Calculate alfa a and alfa b :"

$\alpha_b \leftarrow \alpha_a$

$\alpha_a \leftarrow \alpha_l + (1-\tau)\cdot(\alpha_u - \alpha_l)$

"Calculate the x values :"

$x\alpha_b \leftarrow x\alpha_a$

$x\alpha_a \leftarrow x_k + \alpha_a \cdot d_k$

"Calculate the f values :"

$f^*\alpha_b \leftarrow f^*\alpha_a$

$f^*\alpha_a \leftarrow f(x\alpha_a)$

"Increase counter :"

$Total_{iter} \leftarrow Total_{iter} + 1$

otherwise

"Change alfa l :"

$\alpha_l \leftarrow \alpha_a$

"Calculate alfa a and alfa b :"

$\alpha_a \leftarrow \alpha_b$

$\alpha_b \leftarrow \alpha_l + \tau\cdot(\alpha_u - \alpha_l)$

"Calculate the x values :"

$x\alpha_a \leftarrow x\alpha_b$

$x\alpha_b \leftarrow x_k + \alpha_b \cdot d_k$

"Calculate the f values :"

$f^*\alpha_a \leftarrow f^*\alpha_b$

$f^*\alpha_b \leftarrow f(x\alpha_b)$

"Increase counter :"

$Total_{iter} \leftarrow Total_{iter} + 1$

"Check for error :"

if $f^*\alpha_a = f^*\alpha_b$

"Return error :"

return $\text{Error}(1,4)^T$

"Calculate alfa * :"

$$\alpha^* \leftarrow \frac{\alpha_l + \alpha_u}{2}$$

"Function Output :"

$(\alpha^* \;\; I' \;\; Total_{iter} \;\; rc)^T$

$$\text{GoldenSearch\_PII}(f, x0, d0, \alpha''_l, \alpha''_u, I) = \begin{pmatrix} 2.6 \\ 0 \\ 23 \\ \text{"ok"} \end{pmatrix}$$

A função usando o método da razão áurea foi mais eficiente, conseguiu chegar em uma solução com 23 cálculos da função objetiva, enquanto que a linear estava usando 50.

2) Implement a Matlab routine called `golden_section.m`.
Define it as `[ao]=golden_section(f,xk,dk,delta,unc)`.
In other words, define a routine in which the inputs are the function `f`, the initial point `xk`, the current direction `dk`, the first search step `delta` and the final uncertainty `unc`; the output is `ao`, the optimum α^* parameter.

```
golden_section(f, xk, dk, delta, unc) :=
    "Error handle :"
    α* ← ERROR
    "n iteration variable :"
    n_iter ← 100
    Total_iter ← 0
    "Using Phase I function :"
    (α_l, α_u, Local_iter, rc) ← GoldenSearch_PIa(f, xk, dk, delta, n_iter)
    "Check for error :"
    if rc = ERROR
        "Return error and finish function :"
        return Error(4, 1)
    "Update Total iter :"
    Total_iter ← Total_iter + Local_iter
    "Using Phase II function :"
    (α*, I', Local_iter, rc) ← GoldenSearch_PII(f, xk, dk, α_l, α_u, unc)
    "Check for error :"
    if rc = ERROR
        "Return error and finish function :"
        return Error(4, 1)
    "Update Total iter :"
    Total_iter ← Total_iter + Local_iter
    "Fine end :"
    rc ← "ok"
    "Function Output :"
    (α*, I', Total_iter, rc)
```

$$\text{golden\_section}(f, x0, d0, \delta, I) = \begin{pmatrix} 2.6 \\ 0 \\ 29 \\ \text{"ok"} \end{pmatrix}$$