

TP3

Exercice 1.

Soit un entier $n \geq 2$ et $h = 1/(n+1)$. On se donne un vecteur $f \in \mathbb{R}^n$ et on considère l'application J

$$u \in \mathbb{R}^n \mapsto J(u) = \frac{1}{2h} \sum_{i=0}^n (u_{i+1} - u_i)^2 - h \sum_{i=1}^n u_i f_i,$$

où on a noté $u_0 = 0$ et $u_{n+1} = 0$.

1. Écrire $J(u)$ sous la forme

$$J(x) = \frac{1}{2h} \langle Ax, x \rangle - h \langle b, x \rangle,$$

où A est une matrice symétrique et b un vecteur à déterminer.

2. Montrer que le problème

$$\min_{u \in \mathbb{R}^n} J(u) \tag{1}$$

a une solution et une seule. On la notera \bar{u} .

3. Écrire “l'équation normale” associée à ce problème. Résoudre cette équation en utilisant la fonction `solve` du module `numpy.linalg`. On supposera que la solution obtenue est une très bonne approximation de la solution exacte \bar{u} .

Application : prendre pour f un vecteur dont toutes les composantes sont égales, $f_i = 1$ et $n = 10$. Afficher la courbe $((i h, u_i))_{i=0}^{n+1}$.

4. On veut résoudre le problème de minimisation (1) par une méthode de gradient à pas constant dont les paramètres sont
 - τ : pas de la méthode,
 - η : la “tolérance” sur la gradient de la fonction à minimiser, au sens suivant : on arrête les itérations si la norme du gradient est plus petite que η ,
 - `IterMax` : nombre maximal d'itérations autorisées : le programme s'arrête si on atteint ce nombre.

On note \tilde{u} la solution obtenue par la méthode du gradient. On fixe $n = 10$, $\eta = 10^{-4}$ et `IterMax`=1000. En faisant varier τ dans l'intervalle $[0.01, 0.07]$, tracer la fonction $\tau \mapsto$ nombre d'itérations effectuées par le programme. Ce nombre est donc égal à `IterMax` si la méthode ne converge pas. Indiquer la valeur de τ qui conduit au plus petit nombre d'itérations. On la notera $\tilde{\tau}$. Indiquer aussi la valeur minimale et la valeur maximale pour lesquelles l'algorithme n'a pas convergé. Comparer avec les résultats prévus par une étude théorique.

5. Faire varier les différents paramètres : n , τ , η , `IterMax` et commenter les résultats observés, en particulier le comportement de l'erreur $\bar{u} - \tilde{u}$.

6. Reprendre l'étude avec, cette fois, $f_i = 3 - 6hi$. Commenter les résultats.

Exercice 2.

On cherche à déterminer l'intersection des ellipses d'équations $x^2 + \frac{1}{2}y^2 = 1$ et $y^2 + \frac{1}{2}x^2 = 1$.

1. Tracer sur un même graphique les ellipses (on pourra les paramétrer)
2. Programmer la méthode de Newton pour résoudre le système d'équations. Choisir quatre initialisations différentes pour que l'algorithme converge vers chacun des quatre points d'intersection.

Exercice 3.

L'équation normale associée au problème de minimisation

$$\min_{\|x\|_2=1} \langle Ax, x \rangle$$

où $A \in \mathcal{M}_n(\mathbb{R})$ est une matrice symétrique peut s'écrire

$$Ax + \lambda x = 0.$$

On cherche à déterminer le vecteur x et le multiplicateur de Lagrange λ (qui est donc l'opposé d'une valeur propre de A) en résolvant, par la méthode de Newton, le système d'équations

$$Ax + \lambda x = 0, \quad \|x\|_2^2 = 1.$$

1. Résoudre ce problème pour la matrice $A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & 1 \\ 3 & 1 & 3 \end{pmatrix}$.
2. Vérifier le résultat obtenu en calculant le spectre de cette matrice.

Exercice 4.

On considère un chemin de classe \mathcal{C}^1

$$\gamma : t \in [0, 1] \mapsto \gamma(t) = (x(t), y(t)) \in \mathbb{R}^2$$

conduisant du point de départ $\gamma(0) = (0, 0)$ au point d'arrivée $\gamma(1) = (1, 1)$. Le but de cette étude est de minimiser

$$H(\gamma) = \frac{1}{2} \int_0^1 [x'(t)^2 + y'(t)^2] dt \quad (2)$$

sous la contrainte suivante : le chemin doit contourner un obstacle qu'on suppose être un disque entièrement compris dans le carré unité du plan

$$D = \{(x, y) \in \mathbb{R}^2, (x - a)^2 + (y - b)^2 \leq r^2\}.$$

Pour résoudre ce problème, nous allons "pénaliser" la contrainte. Pour $\varepsilon > 0$ et "petit", on cherche à minimiser

$$H_\varepsilon(\gamma) = H(\gamma) + \frac{1}{\varepsilon} R(\gamma)$$

avec

$$R(\gamma) = \frac{1}{2} \int_0^1 \max(0, r^2 - (x - a)^2 - (y - b)^2)^2 dt. \quad (3)$$

Noter que le terme de "pénalisation" n'intervient que si le point $(x(t), y(t))$ est situé dans l'obstacle. Si aucun point du chemin γ n'est situé dans l'obstacle, on a $H_\varepsilon(\gamma) = H(\gamma)$.

Discrétisation de (2). Pour $N \in \mathbb{N}^*$, on pose $h = 1/N$ et $t_n = nh$ pour $n = 0, \dots, N$. Pour calculer l'intégrale $H(\gamma)$, on écrit

$$H(\gamma) = \frac{1}{2} \sum_{n=0}^{N-1} \int_{t_n}^{t_{n+1}} [x'(t)^2 + y'(t)^2] dt$$

et fait l'approximation

$$H(\gamma) \simeq \mathcal{H}(x, y) = \frac{1}{2} \sum_{n=0}^{N-1} \left[\left(\frac{x_{n+1} - x_n}{h} \right)^2 + \left(\frac{y_{n+1} - y_n}{h} \right)^2 \right] h,$$

où $x = (x_i)_{i=1}^{N-1} \in \mathbb{R}^{N-1}$, $y = (y_i)_{i=1}^{N-1} \in \mathbb{R}^{N-1}$ et les x_j (resp. y_j) sont des approximations des $x(t_j)$ (resp. $y(t_j)$) avec $(x_0, y_0) = \gamma(0)$, $(x_N, y_N) = \gamma(1)$.

Discrétisation de (3). On fait l'approximation

$$R(\gamma) \simeq \mathcal{R}(x, y)$$

avec

$$\mathcal{R}(x, y) = \frac{h}{2} \sum_{n=0}^{N-1} \left(\max(0, r^2 - (x_n - a)^2 - (y_n - b)^2) \right)^2.$$

On cherche donc à minimiser

$$\mathcal{H}_\varepsilon(x, y) = \mathcal{H}(x, y) + \frac{1}{\varepsilon} \mathcal{R}(x, y), \quad (4)$$

pour x et y dans \mathbb{R}^{N-1} .

1. Montrer que

$$\mathcal{H}(x, y) = \frac{1}{2h} (\langle Ax, x \rangle + \langle Ay, y \rangle - 2\langle \bar{x}, x \rangle - 2\langle \bar{y}, y \rangle) + \text{une constante},$$

où $A \in \mathcal{M}_{N-1}(\mathbb{R})$, $\bar{x} \in \mathbb{R}^{N-1}$ et $\bar{y} \in \mathbb{R}^{N-1}$ sont à déterminer.

2. Calculer les gradients de la fonction $\mathcal{H}(x, y)$ par rapport à x et par rapport à y

$$\nabla_x \mathcal{H}(x, y) = \left(\frac{\partial \mathcal{H}}{\partial x_n}(x, y) \right)_n \in \mathbb{R}^{N-1}, \quad \nabla_y \mathcal{H}(x, y) = \left(\frac{\partial \mathcal{H}}{\partial y_n}(x, y) \right)_n \in \mathbb{R}^{N-1}$$

3. Calculer les gradients de la fonction $\mathcal{R}(x, y)$ par rapport à x et par rapport à y

4. Utiliser la méthode du gradient à pas fixe pour résoudre le problème. Reproduire un graphique comme celui de la figure 1.

5. On fixe $N = 30$, $\eta = 10^{-4}$, $\tau = 1$, $\varepsilon = 1/10$, $\text{IterMax} = 5000$. Calculer la solution obtenue par le programme. Diviser ε par 2 et partir de la solution précédente comme donnée initiale de l'algorithme du gradient. Répéter plusieurs fois ce processus. Commenter.

6. **Facultatif**

Voici quelques idées de développements.

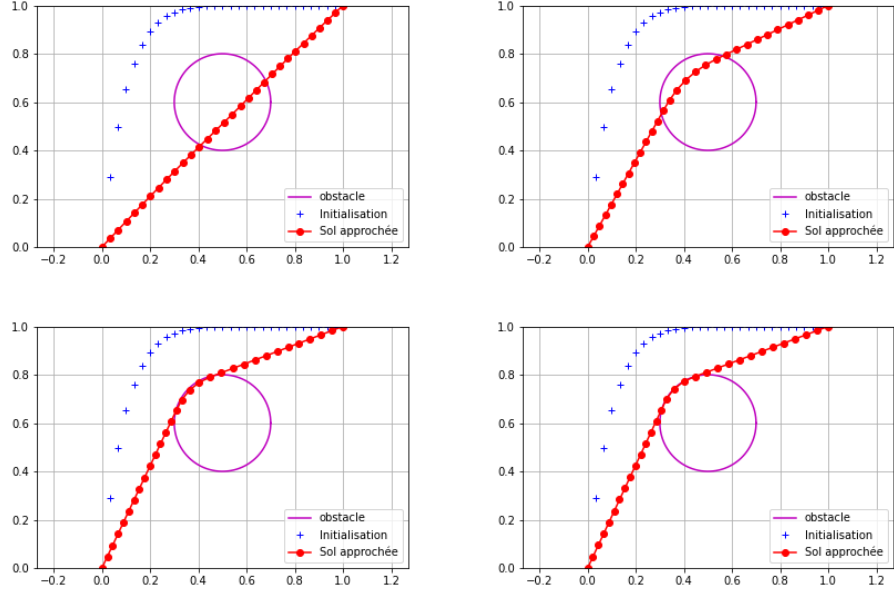


FIGURE 1 – Solutions du problème de minimisation (4) pour différentes valeurs du paramètre de pénalisation $\varepsilon \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. Si ε n'est pas trop petit, la solution minimale reste la ligne droite, même s'il faut payer un prix en passant par l'obstacle. Si ε est assez petit, le minimum ne peut plus passer par l'obstacle, ce serait trop pénalisant. Exemples de paramètres pour la simulation : $N = 30$, obstacle (disque centré en $(0.5, 0.6)$ et de rayon 0.2), méthode du gradient ($\eta = 10^{-4}$, $\tau = 1/10.$, IterMax=?), initialisation : $(x_i, y_i), y_i = 1 - (1 - x_i)^{10}$.

- (a) Modifier la forme de l'obstacle en considérant un carré ou un losange, ...
- (b) Rajouter un autre obstacle.
- (c) ...

Exercice 5. (pour continuer) Retour sur le gradient à pas optimal.

On considère la fonction f définie pour tout $v = (v_1, v_2) \in \mathbb{R}^2$ par

$$f(v) = (v_1 - 4)^2 + 2(v_2 - 3)^2 + v_1 v_2.$$

1. Montrer que f admet un unique minimiseur sur \mathbb{R}^2 et le déterminer.
2. Implémenter la fonction `f` qui prend en argument un vecteur `numpy` de taille 2, représentant un point v de \mathbb{R}^2 , et qui retourne la valeur de la fonction f en ce point.
3. Tracer les lignes de niveaux de f .
4. Implémenter la fonction `gradf` qui calcule le gradient de f en un point.
5. On rappelle que dans la méthode de gradient à pas optimal, le pas ρ_n change à chaque itération :

$$v_{n+1} = v_n - \rho_n \nabla f(v_n) \quad \text{où } \rho_n = \arg \min_{\rho > 0} f(v_n - \rho \nabla f(v_n))$$

Posons $g_n(\rho) = f(v_n - \rho \nabla f(v_n))$, définie de \mathbb{R} dans \mathbb{R} . A chaque itération, pour déterminer le pas optimal ρ_n , nous cherchons la solution du problème (possiblement non linéaire) suivant :

$$g'_n(\rho) = 0$$

Montrer, pour tout $n \geq 0$, que la dérivée de g_n est

$$\forall \rho \in \mathbb{R} \quad g'_n(\rho) = -\nabla f(v_n) \cdot \nabla f(v_n - \rho \nabla f(v_n)).$$

6. Implémenter une fonction `gprime` qui prend en arguments un réel ρ , la fonction `gradf` et un point $v \in \mathbb{R}^2$ (vecteur `numpy` de taille 2) et qui retourne la valeur $g'(\rho)$ qui dépend de v et $\nabla f(v)$.
7. Afin de déterminer une valeur approchée de la solution $g'_n(\rho) = 0$, on va considérer une variante de la méthode de Newton : dans la formule de Newton donnée par

$$\rho_{k+1} = \rho_k - \frac{g'_n(\rho_k)}{g''_n(\rho_k)},$$

pour ne pas avoir à calculer g''_n , on remplace $g''_n(\rho_k)$ par le taux d'accroissement :

$$\frac{g'_n(\rho_k + \delta) - g'_n(\rho_k)}{\delta}, \quad \text{avec } \delta = 10^{-8}.$$

Ainsi, à chaque étape n de l'algorithme du gradient, on construit la suite

$$\begin{cases} \rho_0 \in \mathbb{R} & \text{donné} \\ \rho_{k+1} &= \rho_k - \delta \frac{g'_n(\rho_k)}{g'_n(\rho_k + \delta) - g'_n(\rho_k)}. \end{cases}$$

qui converge vers ρ_n , la solution du problème $\rho_n = \arg \min_{\rho > 0} f(v_n - \rho \nabla f(v_n))$.

Programmer une fonction `newton` qui prend en argument

- la fonction `gprime`,
- la fonction `gradf`,
- x , un vecteur numpy de taille 2,
- une valeur initiale $\rho_0 \in \mathbb{R}$,
- une tolérance ε ,
- un nombre maximum d'itérations N_{\max}

et qui retourne un réel ρ^* , solution de l'équation $g'(\rho) = 0$, en utilisant la variante de l'algorithme de Newton donnée.

On testera cette fonction avec les valeurs suivantes :

$$x = 0_{\mathbb{R}^2}, \quad \rho_0 = 1, \quad \varepsilon = 1e - 5, \quad N_{\max} = 100$$

Indication : On choisira soigneusement le critère d'arrêt pour cet algorithme de Newton et on ajoutera un compteur pour arrêter l'algorithme si le nombre d'itérations dépasse N_{\max} . Ce compteur sera également retourné par la fonction.

8. Programmer la fonction `gradopt` qui prend en arguments :

- v_0 , l'initialisation de l'algorithme,
- ρ_0 , un pas initial,
- ε , la précision,
- la fonction `gradf`,
- la fonction `gprime`,
- N_{\max} , le nombre d'itérations maximum
- $\varepsilon_{\text{Newton}}$, la précision de l'algorithme de Newton,
- $N_{\max, \text{Newton}}$, le nombre maximum de l'algorithme de Newton,

et qui retourne le minimiseur recherché et le nombre d'itérations de la méthode de gradient à pas optimal et le nombre total d'itérations de l'algorithme de Newton, en utilisant l'algorithme de Newton à chaque étape pour déterminer le pas optimal.

Indication : On choisira soigneusement le critère d'arrêt pour cet algorithme de gradient.

9. Tester la méthode du gradient à pas optimal sur la fonction f en traçant le nombre d'itérations en fonction du paramètre ρ_0 . On comptera aussi le nombre d'itérations total (nombre d'itérations de la méthode de gradient à pas optimal plus le nombre total d'itération de l'algorithme de Newton). On prendra les valeurs suivantes pour les autres paramètres :

$$v_0 = (1, 1), \quad \varepsilon = 1e - 6, \quad N_{\max} = 500, \quad \varepsilon_{\text{Newton}} = 10^{-6}, \quad N_{\max, \text{Newton}} = 10.$$

Commenter le graphique obtenu (et pour $\varepsilon_{\text{Newton}} = 10^{-12}$?).

Indication : On fera varier ρ_0 entre 0.01 et 1 par pas de 0.01.

10. Modifier la fonction `gradopt` afin qu'elle retourne également la liste des points construits lors de l'algorithme du gradient. Tracer ensuite les points de cette suite sur le graphique des lignes de niveau de f réalisé à la question 3. Comparer avec la méthode de gradient à pas constant.

Exercice 6. (pour continuer) Exemple d'application de la méthode de Newton.

GPS is a system that provides the position of a receiver from signals emitted by satellites (placed in orbits located approximately 20,000 km from the center of the Earth). The principle is as follows, we measure τ_1 a time taken by a signal emitted by a satellite S_1 to reach the receiver. Knowing that the signal propagates at light speed $c = 2.9979 \cdot 10^8$ m/s we deduce a distance $d_1 = c\tau_1$. The receiver must therefore be on a sphere centered on the satellite S_1 and of radius d_1 . In principle, with 3 satellites we could get the position of the receiver. In practice this is not possible, a problem comes from the synchronization of the clocks of the satellites and the receiver that is not perfect. Indeed, we measure “pseudo distances” :

$$m = d + c\Delta t$$

where Δt is the time shift between the clock of the receiver and the satellites, and it is an unknown quantity. We therefore have 4 unknowns : the coordinates of the receiver (x, y, z) and the time shift Δt . In a fixed reference, with the center of the Earth as origin, we want to solve the following :

$$m_i = \sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} + c\Delta t$$

with $v = (x, y, z, w = c\Delta t)$ the unknown, and (X_i, Y_i, Z_i, m_i) are the information from satellite i ((X_i, Y_i, Z_i) are the coordinate of satellite and m_i the measured “pseudo distance”). We need signals from 4 satellites. Let $F : \mathbb{R}^4 \rightarrow \mathbb{R}^4$ be defined by :

$$F_i(v) = \sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} + w - m_i, i \in \{1, 2, 3, 4\}$$

We want to solve the equation $F(v) = 0$ and we have the following measurements from 4 different satellites (available in `MesuresSatellites` file in Moodle).

Le GPS est un système permettant de donner la position d’un récepteur à partir de signaux émis par des satellites (placés sur des orbites situées à environ 20 000 km du centre de la Terre). Le principe est le suivant, on mesure un temps τ_1 mis par un signal émis par un satellite S_1 pour parvenir au récepteur. Sachant que le signal se propage à la vitesse de la lumière $c = 2.9979 \cdot 10^8$ m/s on en déduit une distance $d_1 = c\tau_1$. Le récepteur doit donc se trouver sur une sphère centrée sur le satellite S_1 et de rayon d_1 . En principe, avec 3 satellites nous pourrions obtenir la position du récepteur. En pratique ce n’est pas possible, il y a un problème qui vient de la synchronisation des horloges des satellites et du récepteur qui n’est pas parfaite. On mesure en effet des “pseudo distances” :

$$m = d + c\Delta t$$

où Δt est le décalage entre l’horloge du récepteur et des satellites, c’est une quantité inconnue. Nous avons donc 4 inconnues : les coordonnées du récepteur (x, y, z) et le décalage Δt .

Dans un repère fixe dont l’origine est le centre de la Terre, nous cherchons à résoudre le problème suivant :

$$m_i = \sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} + c\Delta t$$

où, $v = (x, y, z, w = c\Delta t)$ sont les inconnues, et (X_i, Y_i, Z_i, m_i) sont les informations transmises par le satellite i ((X_i, Y_i, Z_i) sont les coordonnées du satellite et m_i la “pseudo distance” mesurée). Nous avons donc besoin des signaux de 4 satellites. Soit $F : \mathbb{R}^4 \rightarrow \mathbb{R}^4$ définie par :

$$F_i(v) = \sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} + w - m_i, i \in \{1, 2, 3, 4\}$$

Nous cherchons à résoudre le problème $F(v) = 0$, et nous avons les mesures suivantes de 4 satellites différents (disponible dans le fichier MesuresSatellites sur Moodle).

```
# in meters
#position satellite 1 :
pos1 = [5000000, 3632713, 19021130]
m1 = 3917263658 # pseudo-distance satellite 1
#position satellite 2 :
pos2 = [-5000000, 15388418, 11755705]
m2 = 3917265503 # pseudo-distance satellite 2
#position satellite 3 :
pos3 = [11180340, 3632713, -16180340]
m3 = 3917273967 # pseudo-distance satellite 3
#position satellite 4 :
pos4 = [9510565, 6909830, 16180339]
m4 = 3917263997 # pseudo-distance satellite 4
#definition of arrays Xi, Yi, Zi, mi
Xi = np.array([pos1[0], pos2[0], pos3[0], pos4[0]])
Yi = np.array([pos1[1], pos2[1], pos3[1], pos4[1]])
Zi = np.array([pos1[2], pos2[2], pos3[2], pos4[2]])
mi = np.array([m1, m2, m3, m4])
```

1. Implémenter la fonction F en complétant la fonction suivante :

```
def F(v) :
    x, y, z, w = v
    F = np.zeros(4)
    #to complete
    #
    return(F)
```

2. Calculer la jacobienne de F et l'implémenter dans une fonction JF en complétant la fonction suivante :

```
def JF(v) :
    x, y, z, w = v
    J = np.zeros((4, 4))
    # to complete
    #
    return(J)
```

3. Implémenter la résolution du problème non linéaire avec une méthode de Newton en complétant la fonction suivante :

```
def Newton(F, JF, v0, Tol, IterMax):
    #to complete
    #
    return(v, S, cvg, it)
```


avec

- F et JF les fonctions implémentées dans les questions précédentes
- $v0$ l'initialisation de l'algorithme de Newton
- Tol et $IterMax$ la tolérance et le nombre d'itérations maximum autorisé dans l'algorithme de Newton
- v le vecteur solution
- S la suite des itérés
- cvg un booléen indiquant si l'algorithme a convergé
- it le nombre d'itérations pour atteindre la convergence

4. Donner une estimation de v , pour une tolérance de 10^{-4} et $v0 = [0, 0, 0, 0]$.
5. Calculer le décalage d'horloge Δt (sachant que la vitesse de la lumière est $c = 2.9979 \cdot 10^8$ m/s).

Exercice 7. (pour continuer) minimisation sous contrainte

We considere the following function

$$F : (x_1, x_2) \in \mathbb{R}^2 \longmapsto ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + f$$

where a, b, c, d, e, f are in \mathbb{R} .

1. Write a code that computes, with fixed step gradient method, the solution (if it exist) of the minimization problem

$$\min_{(x_1, x_2) \in \mathbb{R}^2} F(x_1, x_2).$$

Print the obtained solution and the value of F at this point, as well as the number of iterations.

Application : $(a, b, c, d, e, f) = (3.56, 1, -3.2, -5, 0, 9.39)$,
starting point $(0, 0)$

2. We now want to take into account a constraint :

$$\min_{(x_1, x_2) \in \Omega} F(x_1, x_2)$$

where $\Omega \subset \mathbb{R}^2$. We propose to take care of the constraint as follow : starting from x_k , compute x_{k+1} by en projecting on Ω the array from a single step obtained with gradient method apply to x_k . We will consider the case where Ω is rectangular, which simplifies the computation of the (orthogonal) projection on Ω . Indeed, if

$$\Omega = \{(x_1, x_2), \quad a_1 \leq x_1 \leq b_1 \text{ et } a_2 \leq x_2 \leq b_2\},$$

we have

$$P_\Omega : (x_1, x_2) \in \mathbb{R}^2 \longmapsto (y_1, y_2) \in \Omega$$

where $y_i = \min(\max(a_i, x_i), b_i)$.

We can then fixed $(a_1, b_1, a_2, b_2) = (1, 4, 2, 3)$ for the rest of the exercice.

- (a) Write a function that computes the projection on Ω .
- (b) Solve the constrained problem. Print the obtained solution starting from $(0, 0)$ and the value of F .

- (c) Draw a similar figure to the one below, where we display the obstacle, the solution without constraint (with signe “x”), the constrained minimization solution (with signe “•”), as well as a “path” starting from the initial point $(0, 0)$ and reaching the solution, meaning the sequence of points produced by the algorithm (showed in dashed line).

