3m101 « Optimisation » - Notebook expected results

May 6, 2018

Optimisation du profil d'une route

```
In [1]: from projet import *
```

On commence par lire le profil du terrain que l'on stock dans deux listes : pt_mes (des points mesurés) et alt (des altitudes respectifs).

```
In [2]: pt_mes, alt = lire_profil('profil.txt')
        print("Une partie du terrain :\n", "Points mesurés :", pt_mes[:10], "\n Altitude :", alt
```

```
Une partie du terrain :
 Points mesurés : [238.7565, 253.48261, 268.20872, 282.93483, 297.66094, 312.38705, 327.11315, 3
 Altitude : [280.43478, 280.43478, 278.9855, 278.9855, 278.9855, 278.9855, 278.9855, 278.9855, 2
```

Quelques paramètres supplémentaires sont fixés :

```
In [3]: L = pt_mes[-1] #longueur du terrain
        n = len(pt_mes) #nombre de points mesurés
        h = L/(n-1) #pas
        penteMax = 0.1 #pente maximale de la route fixée à 10%
```

On peut aussi calculer la pente locale maximale du terrain :

```
In [4]: penteLoc = pente_max(pt_mes, alt)
        print ("Pente locale maximale trouvée : ", penteLoc)
```

```
Pente locale maximale trouvée :  0.5904923910607753
```

Cas linéaire

Ce probleme d'optimisation s'écrit sous la forme : $min \quad f^t \times v \; s.c \quad C \times v - d \leq 0$

On calcule alors les vecteurs f et d et la matrice C :

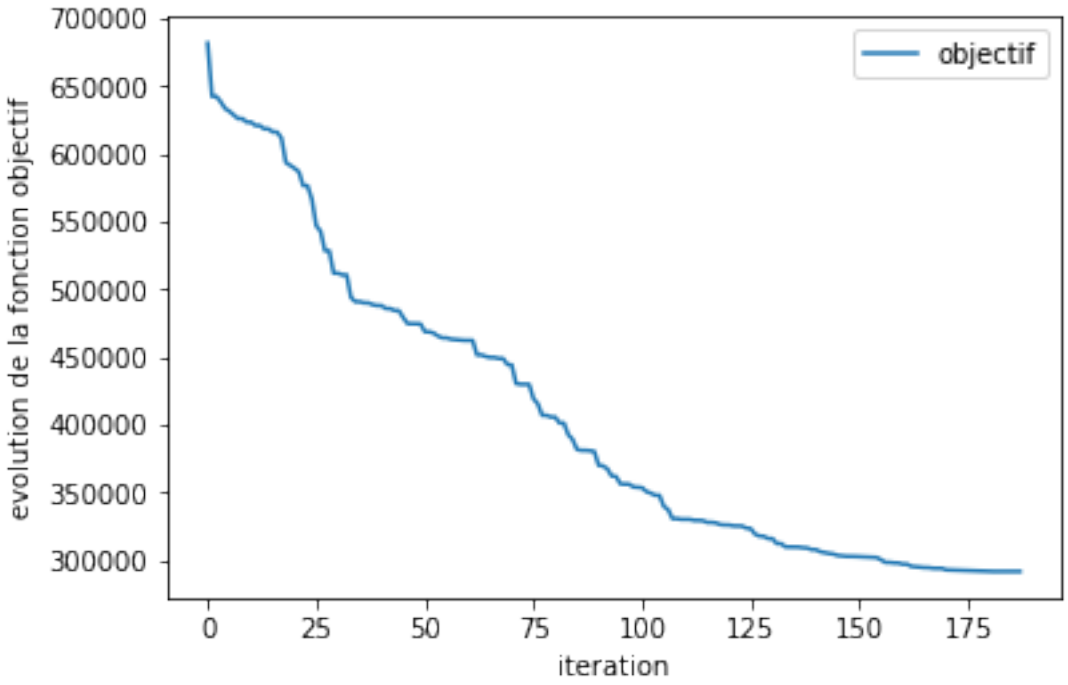
```
In [5]: f, C, d = def_matrice(pt_mes, alt, h, penteMax) #matrices
```

On choisit de résoudre ce problème en utilisant la fonction linprog de la bibliothèque scipy...

```
In [6]: bnds = [(None, None)] * n + [(0, None)] * n #bornes
        res = linprog(f, C, d, bounds=bnds, options={'disp': False, 'bland': False, 'tol': 1e-9,
        profil_linprog = res['x'][:n] #profil retournee par scipy.linprog
```

... ainsi qu'un programme maison du simplexe (environ 1000 itérations) Le programme affiche également l'évolution de la fonction objective au cours des itérations.

```
In [7]: res = simplex(f, C, d, verbose=True)
        profil_simplex = res[:n]  #profil retournee par le simplexe maison
```

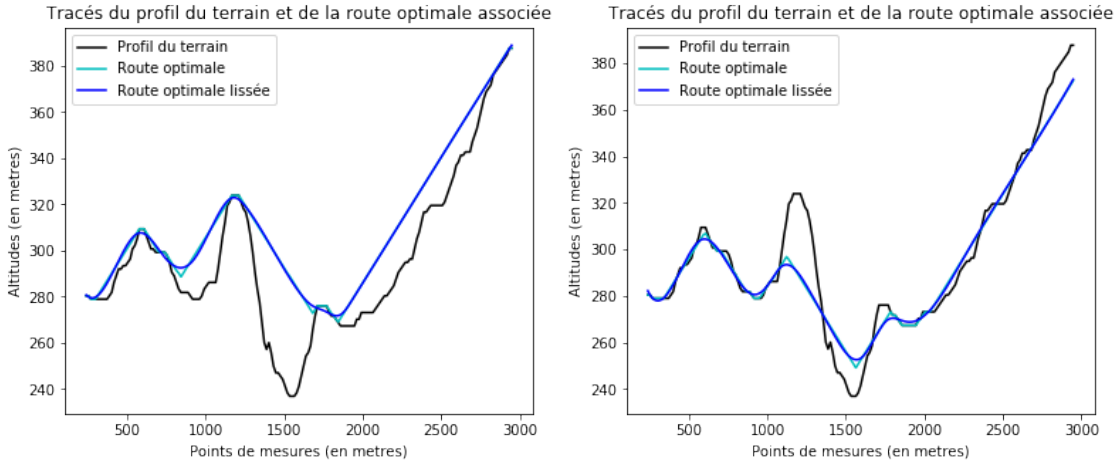


Les tracés sont lissés en s'appuyant sur l'interpolation de Spline (interpolate.splrep et interpolate.splev) :

```
In [9]: #lissage profil_linprog
        tck = interpolate.splrep(pt_mes, profil_linprog, s=len(pt_mes)-np.sqrt(2*len(pt_mes)))
        xnew = np.linspace(pt_mes[0], pt_mes[-1], 20*len(pt_mes))
        profil_linprog_liss = interpolate.splev(xnew, tck)

        #lissage profil_simplex
        tck = interpolate.splrep(pt_mes, profil_simplex, s=len(pt_mes)-np.sqrt(2*len(pt_mes)))
        xnew = np.linspace(pt_mes[0], pt_mes[-1], 20*len(pt_mes))
        profil_simplex_liss = interpolate.splev(xnew, tck)
```

On affiche les tracés des routes optimales obtenues par scipy.linprog (à gauche) et par le simplexe maison (à droite).

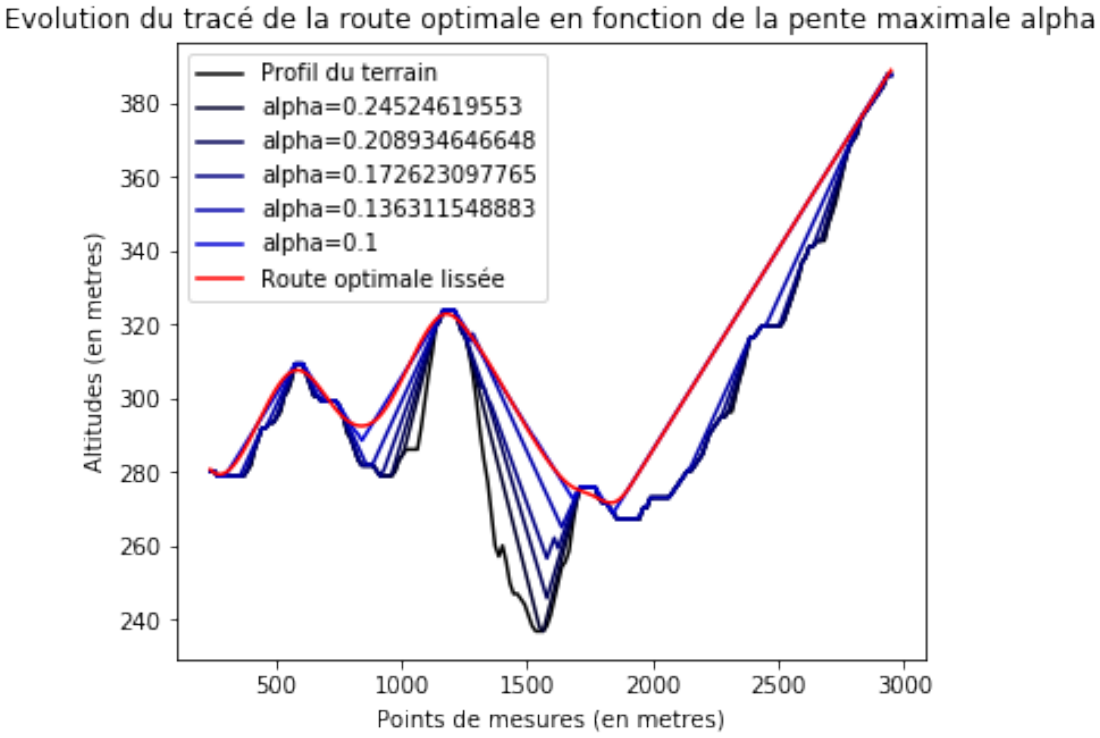
In [10]:
```
# pour agrandir les courbes on utilise le module pylab
from pylab import *
#on crée un graphique de 13x5 pouces
figure(figsize=(13,5))

plt.subplot(1,2,1)
plt.plot(pt_mes, alt, 'k', label="Profil du terrain")
plt.plot(pt_mes, profil_linprog, 'c', label="Route optimale")

plt.plot(xnew, profil_linprog_liss, 'b', label="Route optimale lissée")

plt.xlabel("Points de mesures (en metres)")
plt.ylabel("Altitudes (en metres)")
plt.title("Tracés du profil du terrain et de la route optimale associée")
plt.legend()

plt.subplot(1,2,2)
plt.plot(pt_mes, alt, 'k', label="Profil du terrain")
plt.plot(pt_mes, profil_simplex, 'c', label="Route optimale")

plt.plot(xnew, profil_simplex_liss, 'b', label="Route optimale lissée")

plt.xlabel("Points de mesures (en metres)")
plt.ylabel("Altitudes (en metres)")
plt.title("Tracés du profil du terrain et de la route optimale associée")
plt.legend()

plt.show()
```

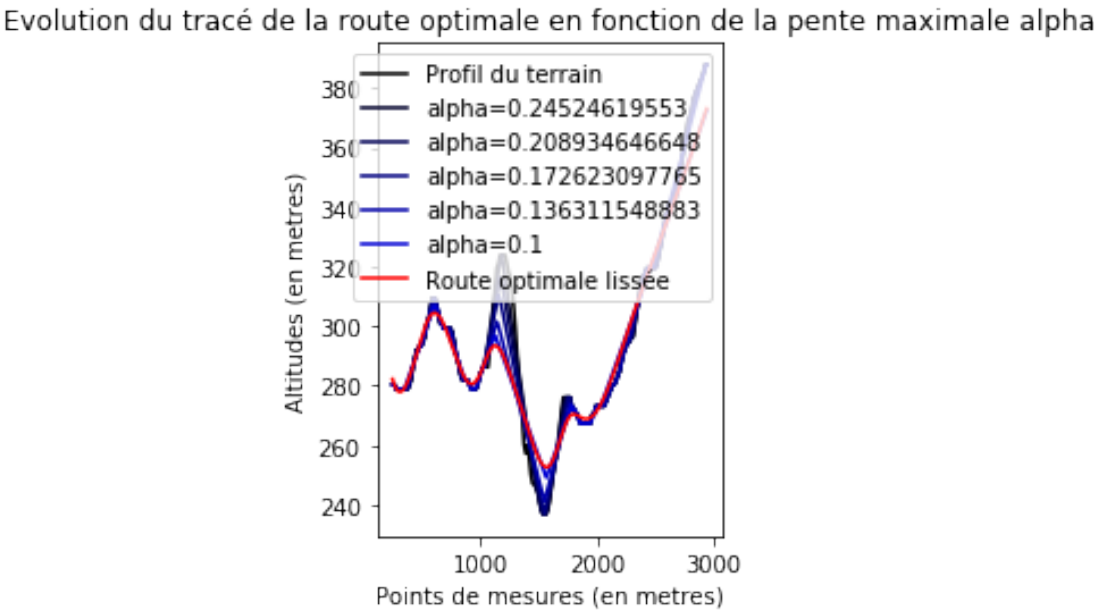


On avait fixé $\alpha = 10\%$, mais on peut faire varier la valeur de la pente maximale autorisée :

In [2]:
```
# pour agrandir les courbes on utilise le module pylab
from pylab import *
```

```
#on crée un graphique de 13x5 pouces
figure(figsize=(13,5))

plt.subplot(1,2,1)
evolv_alpha('profil.txt')
plt.subplot(1,2,2)
evolv_alpha('profil.txt', methode='maison')
```





Cas quadratique

Ce probleme d'optimisation s'écrit sous la forme : $min \quad < A(U-G),(U-G) > \; s.c \quad C \times U - d \leq 0$

On calcule alors les vecteurs A, inv_A, C, d :

```
In [3]: A, inv_A, C, d = def_matrice_quad(pt_mes, alt, h, penteMax) #matrices
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-3-f9392ea36f6f> in <module>()
----> 1 A, inv_A, C, d = def_matrice_quad(pt_mes, alt, h, penteMax) #matrices


NameError: name 'def_matrice_quad' is not defined
```