Corrigé : poteaux télégraphiques (X psi & pt 2013)

Partie I. Planter le paysage

Question 1.

```
def calculHauteur():
    hauteurs[0] = 0.
    for i in range(1, n+1):
        hauteurs[i] = hauteurs[i−1] + deniveles[i]
```

Question 2.

```
def calculFenetre():
    global hMin, hMax, iMin, iMax
    hMin, hMax = float('inf'), −float('inf')
    for i, h in enumerate(hauteurs):
        if h < hMin:
            iMin, hMin = i, h
        if h > hMin:
            iMax, hMax = i, h
```

Question 3. La distance au sol entre les points P_{k-1} et P_k est égale à $\sqrt{1+\texttt{deniveles}[k]^2}$.

```
def distanceAuSol(i, j):
    i, j = min(i, j), max(i, j)
    d = 0
    for k in range(i+1, j+1):
        d += sqrt(1 + deniveles[k]**2)
    return d
```

Notez que l'énoncé ne précise pas si on a bien $i \leqslant j$.

Question 4. On utilise une fonction auxiliaire pour déterminer si un point est remarquable :

```
def estRemarquable(i):
    return i == 0 or i == n or hauteurs[i] > hauteurs[i−1] and hauteurs[i] > hauteurs[i+1]
```

On définit ensuite :

```
def longueurDuPlusLongBassin():
    lMax, i = 0., 0
    for j in range(1, n+1):
        if estRemarquable(j):
            lMax = max(lMax, distanceAuSol(i, j))
            i = j
    return lMax
```

L'entier `i` désigne le dernier point remarquable trouvé.

Partie II. Planter les poteaux

Question 5. Le problème est à l'évidence symétrique vis-à-vis des variables i et j, ce qui permet de se ramener au cas où $i \leqslant j$.

```
def estDeltaAuDessusDuSol(i, j):
    i, j = min(i, j), max(i, j)
    beta = (hauteurs[j] - hauteurs[i]) / (j - i)
    for k in range(i+1, j):
        if (hauteurs[k] + delta - hauteurs[i] - l) / (k - i) > beta:
            return False
    return True
```

Question 6. L'énoncé de la question suivante suggère de ne pas chercher à optimiser cette première fonction …

```
def placementGloutonEnAvant():
    poteaux = [0]
    i, j = 0, 1
    while i != n:
        while j <= n and estDeltaAuDessusDuSol(i, j):
            j += 1
        i = j - 1
        poteaux.append(i)
    return poteaux
```

L'indice `i` indique l'emplacement du dernier poteau planté.

Question 7. La fonction `estDeltaAuDessusDuSol` a une complexité en $\mathrm{O}(j-i)$, donc la complexité $\mathrm{C}(n)$ de la fonction `placementGloutonEnAvant` vérifie : $\mathrm{C}(n) = \sum_{i=0}^{k-1} \sum_{j=p_i}^{p_{i+1}} \mathrm{O}(j-p_i)$, où $p_0 = 0, \dots, p_k = n$ désignent l'emplacement des poteaux choisis par cet algorithme. On a donc $\mathrm{C}(n) = \mathrm{O}\Big(\sum_{i=0}^{k-1} (p_{i+1}-p_i)^2\Big)$. On majore [1] pour obtenir : $\mathrm{C}(n) = \mathrm{O}\Big(\sum_{i=0}^{k-1} (p_{i+1}-p_i)\Big)^2 = \mathrm{O}(n^2)$.
Notons que cette majoration n'est pas trop grossière ; elle est atteinte dans le cas où $k=1$ (lorsque deux poteaux plantés aux extrémités suffisent).
Bien entendu, on peut obtenir une complexité linéaire en transportant l'invariant $\max\limits_{i<k<j} \alpha_{i,k}$ lors de la recherche du prochain poteau à planter, ce qui conduit à écrire :

```
def placementGloutonEnAvant():
    poteaux = [0]
    i, j = 0, 1
    while i != n:
        alpha = -float('inf')
        while j <= n and (hauteurs[j] - hauteurs[i]) / (j - i) > alpha:
            alpha = max(alpha, (hauteurs[j] + delta - hauteurs[i] - l) / (j - i))
            j += 1
        i = j - 1
        poteaux.append(i)
    return poteaux
```

Avec ce nouvel algorithme on a : $\mathrm{C}(n) = \sum_{i=0}^{k-1} \sum_{j=p_i}^{p_{i+1}} \mathrm{O}(1) = \mathrm{O}\Big(\sum_{i=0}^{k-1} (p_{i+1}-p_i)\Big) = \mathrm{O}(n)$.

Question 8. On procède comme à la question 6, en débutant la recherche à partir du poteau le plus éloigné possible.

```
def placementGloutonAuPlusLoin():
    poteaux = [0]
    i = 0
    while i != n:
        j = n
        while not estDeltaAuDessusDuSol(i, j):
            j -= 1
        i = j
        poteaux.append(i)
    return poteaux
```

1. J'utilise l'inégalité : $a_1, \dots, a_k \geqslant 0 \Longrightarrow \sum a_i^2 \leqslant (\sum a_i)^2$.

Partie III. Minimiser la longueur du fil

Question 9. Par définition on a optL[0] = 0.
Si $i \geqslant 1$, considérons un entier $0 \leqslant j < i$ quelconque mais pour lequel il est possible de tirer un fil de P_i à P_j. Il en existe au moins un car il est toujours possible de tirer un fil de P_{i-1} à P_i. Il est possible de relier P_0 et P_i en reliant de manière optimale P_0 et P_j puis en reliant P_j et P_i ; ceci montre que $\texttt{opt}[i] \leqslant L_{i,j} + \texttt{opt}[j]$.

Considérons maintenant un choix de points $(P_0, \ldots, P_{j_0}, P_i)$ où planter les poteaux pour relier les points P_0 et P_i avec une longueur minimale de fil, et notons j_0 l'indice de l'avant-dernier poteau. On a $0 \leqslant j_0 < i$. Il reste à observer que planter des poteaux en $(P_0, \ldots, P_{j_0})$ relie de manière optimale les poteaux P_0 et P_{j_0} ; en effet si ce n'était pas le cas il serait possible de contredire le caractère optimal de $(P_0, \ldots, P_{j_0}, P_i)$. La longueur de fil utilisée pour relier $(P_0, \ldots, P_{j_0})$ est donc égale à $\texttt{opt}[j_0]$, et $\texttt{opt}[i] = L_{i,j_0} + \texttt{opt}[j_0]$.

Question 10. Commençons par une fonction qui calcule la longueur du segment d'extrémités P_i et P_j.

```python
def longueurDuSegment(i, j):
    return sqrt((j - i)**2 + (hauteurs[j] - hauteurs[i])**2)
```

On adopte ensuite une technique propre à la programmation dynamique en calculant le tableau opt avant de retourner la valeur $\texttt{opt}[n]$.

```python
def longueurMinimale():
    optL = [0] * (n+1)
    for i in range(1, n+1):
        optL[i] = float('inf')
        for j in range(0, i):
            if estDeltaAuDessusDuSol(i, j):
                optL[i] = min(optL[i], optL[j] + longueurDuSegment(i, j))
    return optL[n]
```

Question 11. Modifions la fonction précédente pour stocker en plus la valeur de j_0 définie à la question 9 :

```python
def longueurMinimale2():
    optL = [0] * (n+1)
    precOptL = [-1] * (n+1)
    for i in range(1, n+1):
        optL[i] = float('inf')
        for j in range(0, i):
            d = optL[j] + longueurDuSegment(i, j)
            if estDeltaAuDessusDuSol(i, j) and d < optL[i]:
                optL[i], precOptL[i] = d, j
    return precOptL
```

Le coût de la fonction estDeltaAuDessusDuSol étant linéaire, le coût total de cette fonction est en $O(n^3)$.

Question 12. Le tableau precOptL permet de calculer de la droite vers la gauche la liste des poteaux requis pour une solution optimale :

```python
def placementOptimal():
    precOptL = longueurMinimale2()
    j = n
    poteaux = [n]
    while j > 0:
        j = precOptL[j]
        poteaux.append(j)
    poteaux.reverse()
    return poteaux
```

La méthode reverse appliquée à une liste renverse l'ordre de ses éléments en coût linéaire ; mis à part le coût du calcul du tableau preOptL, les autres coûts de la fonction longueurMinimale2 sont en $O(n)$.
Le coût total de la démarche est donc un $O(n^3)$.