Corrigé

Autour de la conjecture de Syracuse

Temps de vol et altitude maximale

Question 1.

```
def tempsdevol(c):
    n, u = 0, c
    while u > 1:
        if u % 2 == 0:
            u = u // 2
        else:
            u = 3 * u + 1
        n += 1
    return n
```

Question 2.

```
def altitude(c):
    a, u = c, c
    while u > 1:
        if u % 2 == 0:
            u = u // 2
        else:
            u = 3 * u + 1
        a = max(a, u)
    return a
```

Observez la similitude de la construction de ces deux fonctions, qui partagent un même invariant : *à l'entrée de la* $(k+1)^e$ *boucle conditionnelle,* u *est égal à* u_k.
Dans la fonction `tempsdevol`, on utilise un second invariant : n *est égal à* k, dans la fonction `altitude` on utilise l'invariant : a *est égal à l'altitude maximale observée entre* 0 *et* k.

Vérification expérimentale de la conjecture

Question 3. Cette fonction est presque identique à `tempsdevol` ; il suffit de modifier la condition d'arrêt de la boucle conditionnelle.

```
def tempsdarret(c):
    n, u = 0, c
    while u >= c:
        if u % 2 == 0:
            u = u // 2
        else:
            u = 3 * u + 1
        n += 1
    return n
```

On importe la fonction `time` par la commande :

```
from time import time
```

On définit ensuite :

```
def verification1(m):
    depart = time()
    for c in range(2, m + 1):
        u = c
        while u >= c:
            if u % 2 == 0:
                u = u // 2
            else:
                u = 3 * u + 1
    fin = time()
    print("durée de la vérification :", fin - depart)
```

```
>>> verification1(1000000)
durée de la vérification : 1.7752561569213867
```

Si $c = 2n$ est pair alors $u_0 = 2n$ et $u_1 = n < u_0$ donc le temps d'arrêt est égal à 1.
Si $c = 4n+1$ alors $u_0 = 4n+1$, $u_1 = 12n+4$, $u_2 = 6n+2$ et $u_3 = 3n+1 < u_0$ donc le temps d'arrêt est égal à 3. Il suffit donc de vérifier la conjecture pour les entiers de la forme $4n+3$.

```
def verification2(m):
    depart = time()
    for c in range(3, m + 1, 4):
        u = c
        while u >= c:
            if u % 2 == 0:
                u = u // 2
            else:
                u = 3 * u + 1
    fin = time()
    print("durée de la vérification :", fin - depart)
```

On obtient un gain de performance (peu visible ici, la valeur de m étant un peu faible compte tenu des performances de mon ordinateur) :

```
>>> verification2(1000000)
durée de la vérification : 1.167517900466919
```

Ceci permet d'envisager de vérifier la conjecture pour de plus grandes valeurs de m :

```
>>> verification2(10000000):
durée de la vérification : 12.00194787979126
```

Expérience faite, il faudrait 18 secondes pour effectuer cette même vérification avec la première des deux fonctions.

Records

Question 4.

```
def altitudemax(m):
    a, x = 1, 1
    for c in range(2, m + 1):
        u = c
        while u >= c:
            if u > a:
                a, x = u, c
            if u % 2 == 0:
                u = u // 2
            else:
                u = 3 * u + 1
    print('altitude max = {}, atteinte pour c = {}'.format(a, x))
```

On obtient :

```
>>> altitudemax(1000000)
altitude max = 56991483520, atteinte pour c = 704511
```

```
def tdvamax(m):
    d, x = 0, 1
    for c in range(2, m + 1):
        u, n = c, 0
        while u >= c:
            n += 1
            if u % 2 == 0:
                u = u // 2
            else:
                u = 3 * u + 1
        if n > d:
            d, x = n, c
    print('durée de vol en altitude max = {}, atteinte pour c = {}'.format(d, x))
```

On obtient :

```
>>> tdvamax(1000000)
durée de vol en altitude max = 287, atteinte pour c = 626331
```

Question 5. On définit la fonction :

```
def dureerecord(m):
    d = 0
    for c in range(2, m):
        n, u = 0, c
        while u > 1:
            n += 1
            if u % 2 == 0:
                u = u // 2
            else:
                u = 3 * u + 1
        if n > d:
            d = n
            print('pour c = {}, record de durée = {}'.format(c, d))
```

qui produit les résultats suivants :

```
>>> dureerecord(1000000)
pour c = 2, record de durée = 1
pour c = 3, record de durée = 6
pour c = 7, record de durée = 11
pour c = 27, record de durée = 96
pour c = 703, record de durée = 132
pour c = 10087, record de durée = 171
pour c = 35655, record de durée = 220
pour c = 270271, record de durée = 267
pour c = 362343, record de durée = 269
pour c = 381727, record de durée = 282
pour c = 626331, record de durée = 287
```

Affichage du vol

Question 6.

```
import matplotlib.pyplot as plt

def graphique(c):
    u, lst = c, [c]
    while u > 1:
        if u % 2 == 0:
            u = u // 2
        else:
            u = 3 * u + 1
        lst.append(u)
    plt.plot(lst)
    plt.show()
```

On trouvera figure 1 un exemple d'utilisation de cette fonction.

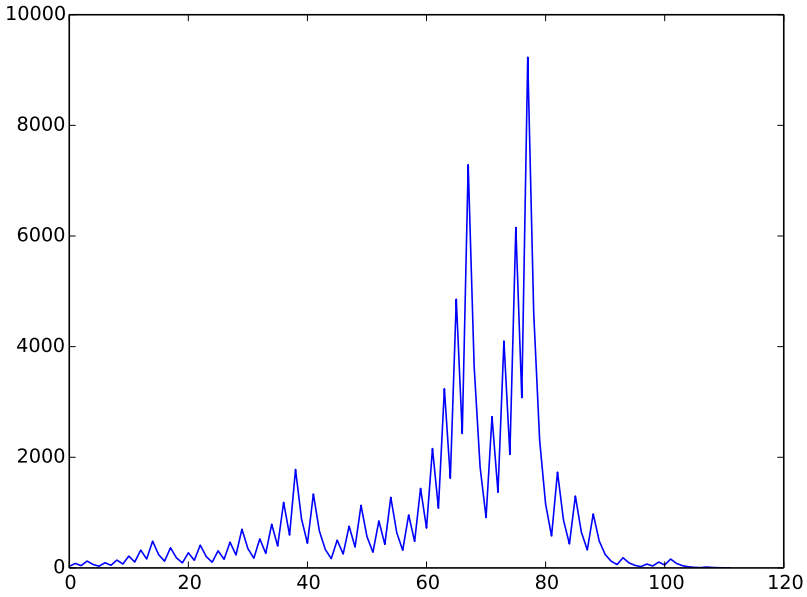

FIGURE 1 – Le résultat de la fonction `graphique` pour $c = 27$.

Et pour ceux qui s'ennuient

Reprenons l'étude des entiers de la forme $4n+1$ et $4n+3$.
Si $c = 4n+1$, $u_1 = 12n+4$, $u_2 = 6n+2$, $u_3 = 3n+1 < c$ donc le temps d'arrêt est fini et les entiers de la forme $4n+1$ peuvent être éliminés de l'étude.
Si $c = 4n+3$, $u_1 = 12n+10$, $u_2 = 6n+5$, $u_3 = 18n+16$, $u_4 = 9n+8$ et faute de pouvoir déterminer la parité de u_4 on ne peut poursuivre.
Poser $n = 2p$ puis $n = 2p+1$ scinde ce dernier cas en deux : les entiers de la forme $8p+3$ ou de la forme $8p+7$. Malheureusement aucun des deux ne peut être éliminé car :

$$8p+3 \longrightarrow 24p+10 \longrightarrow 12p+5 \longrightarrow 36p+16 \longrightarrow 18p+8 \longrightarrow 9p+4$$
$$8p+7 \longrightarrow 24p+22 \longrightarrow 12p+11 \longrightarrow 36p+34 \longrightarrow 18p+17 \longrightarrow 54p+52 \longrightarrow 27p+26$$

et dans les deux cas la valeur finale (c'est-à-dire quand on ne peut plus déterminer la parité de l'entier) est supérieure à c.
La fonction suivante détermine si les entiers de la forme $2^p n + k$ peuvent être éliminés :

```
def elimine(p, k):
    u, v = 2**p, k
    while u % 2 == 0 and u >= 2**p:
        if v % 2 == 0:
            u, v = u // 2, v // 2
        else:
            u, v = 3 * u, 3 * v + 1
    return u < 2**p and v <= k
```

L'étude des entiers de la forme $2^p n + k$ se scinde en deux suivant la parité de n : ceux de la forme $2^{p+1} n + k$ et $2^{p+1} + (k + 2^p)$.
Basé sur ce principe, la fonction suivante calcule les différentes réductions du problème :

```
def liste(b):
    l = [0]
    for p in range(0, b):
        s = []
        for k in l:
            if not elimine(p + 1, k):
                s.append(k)
            if not elimine(p + 1, k + 2**p):
                s.append(k + 2**p)
        l = s
    return l
```

Voici par exemple la liste des valeurs de k à garder pour l'étude des entiers de la forme $256n + k$:

```
>>> liste(8)
[27, 155, 91, 251, 71, 167, 103, 231, 207, 47, 111, 239, 31, 159, 223, 63, 191, 127, 255]
```

ceci permet de réduire l'étude de la conjecture à $7,4$ % des entiers.
Enfin le nombre de cas à considérer pour les entiers de la forme $65536n + k$ est égal à :

```
>>> len(liste(16))
2114
```

ce qui réduit l'étude de la conjecture à $3,2$ % des entiers.
Avec cette méthode, on vérifie la conjecture jusqu'à $10\,000\,000$ en 5 secondes, et jusqu'à $100\,000\,000$ en 52 secondes.