Corrigé

Transformations du photomaton et du boulanger

Question 1. symétrie d'axe vertical

On utilise les relations $x' = x$ et $y' = q - 1 - x$ pour définir la fonction :

```
def symetrie(img):
    p, q = img.shape[0], img.shape[1]
    img2 = np.zeros_like(img)
    for x in range(p):
        for y in range(q):
            img2[x, q-1-y] = img[x, y]
    return img2
```

Question 2. rotation d'un quart de tour

On utilise les relations $x' = y$ et $y' = p - 1 - x$ pour définir la fonction :

```
def rotation(img):
    p, q = img.shape[0], img.shape[1]
    img2 = np.zeros((q, p, 4))
    for x in range(p):
        for y in range(q):
            img2[y, p-1-x] = img[x, y]
    return img2
```

1. Transformation du photomaton

Question 3. La transformation du photomaton utilise les formules :

$$(x', y') = \begin{cases} (x/2, y/2) & \text{si } x \text{ et } y \text{ sont pairs} \\ (x/2, \lfloor y/2 \rfloor + q/2) & \text{si } x \text{ est pair et } y \text{ impair} \\ (\lfloor x/2 \rfloor + p/2, y/2) & \text{si } x \text{ est impair et } y \text{ pair} \\ (\lfloor x/2 \rfloor + p/2, \lfloor y/2 \rfloor + q/2) & \text{si } x \text{ et } y \text{ sont impairs} \end{cases}$$

formule qui peut être synthétisée en Python par

`xprime = (x // 2) + (x % 2) * (p // 2)` et `yprime = (y // 2) + (y % 2) * (q // 2)`

```
def photomaton(img):
    p, q = img.shape[0], img.shape[1]
    img2 = np.zeros_like(img)
    for x in range(p):
        for y in range(q):
            img2[(x // 2) + (x % 2) * (p // 2), (y // 2) + (y % 2) * (q // 2)] = img[x, y]
    return img2
```

Question 4. Le script suivant permet de visualiser l'ensemble des transformations avant retour à l'image initiale :

```
img = picasso
n = 0
while True:
    img = photomaton(img)
    n += 1
    plt.imshow(img)
    plt.title('n = {}'.format(n))
    plt.pause(1)
    if (img == picasso).all():
        break
```

Cette image, de taille 256×256, a une période égale à 8.

Question 5. Pour calculer la période d'une image quelconque on utilise la fonction :

```
def periode_photomaton(img):
    p, q = img.shape[0], img.shape[1]
    n, t = 1, 2
    while (t-1) % (p-1) != 0 or (t-1) % (q-1) != 0:
        n += 1
        t *= 2
    return n
```

Ainsi, la période d'une image de taille 400×360 (la taille de l'image `matisse.png`) est égale à 3 222.

Question 6. Les formules obtenues à la question 3 montrent que le calcul de l'abscisse d'un pixel au bout de n itérations ne dépend que de la valeur de l'abscisse initiale, et il en est de même pour les ordonnées. En d'autres termes, tous les points situés sur une même ligne verticale restent alignés verticalement, et tous les points situés sur une même ligne horizontale restent alignés horizontalement. Il suffit donc de faire le calcul une bonne fois pour toute pour chacun des indices de ligne et pour chacun des indices de colonne pour pouvoir ensuite déterminer rapidement où situer un pixel après n itérations.

```
def photomaton2(img, n):
    p, q = img.shape[0], img.shape[1]
    img2 = np.zeros_like(img)
    ligne = []
    for x in range(p):
        u = x
        for _ in range(n):
            u = (u // 2) + (u % 2) * (p // 2)
        ligne.append(u)
    colonne = []
    for y in range(q):
        v = y
        for _ in range(n):
            v = (v // 2) + (v % 2) * (q // 2)
        colonne.append(v)
    for x in range(p):
        for y in range(q):
            img2[ligne[x], colonne[y]] = img[x, y]
    return img2
```

Ceci permet d'obtenir instantanément la 180e itération de l'image `matisse.png` (voir figure 1).
Cette image est de taille 400×360. Le plus petit entier n pour lequel 399 divise $2^n - 1$ est $n = 18$, tandis que le plus petit entier n pour lequel 359 divise $2^n - 1$ est $n = 179$.
Puisque $180 \equiv 0 \bmod (18)$, tous les pixels ont retrouvé leurs lignes de départ initiales ; puisque $180 \equiv 1 \bmod (179)$, les colonnes n'ont subies qu'une transformation par rapport à leurs positions initiales.

2. Transformation du boulanger

Question 7. Lors de l'« aplatissement » de l'image, le pixel de coordonnées (x, y) se retrouve au point de coordonnées :

$$(x_1, y_1) = \begin{cases} (x/2, 2y) & \text{si } x \text{ est pair} \\ (\lfloor x/2 \rfloor, 2y+1) & \text{si } x \text{ est impair} \end{cases} \quad \text{dans l'image intermédiaire de dimensions } p/2 \times 2q.$$

FIGURE 1 – La 180e itération de l'image `matisse.png`.

Il faut ensuite « replier » l'image, ce qui conduit aux formules :

$$(x', y') = \begin{cases} (x_1, y_1) & \text{si } y_1 < q \\ (p-1-x_1, 2q-1-y_1) & \text{si } y_1 > q \end{cases}$$

La fonction qui calcule les nouvelles coordonnées d'un pixel après transformation se définit donc par :

```
def boulange(x, y, p, q):
    x1, y1 = x // 2, 2 * y + x % 2
    if y1 < q:
        return x1, y1
    else:
        return p - 1 - x1, 2 * q - 1 - y1
```

On en déduit la fonction :

```
def boulanger(img):
    p, q = img.shape[0], img.shape[1]
    img2 = np.zeros_like(img)
    for x in range(p):
        for y in range(q):
            img2[boulange(x, y, p, q)] = img[x, y]
    return img2
```

Question 8. On visualise l'ensemble des transformations de l'image `picasso.png` à l'aide du script :

```
img = picasso
n = 0
while True:
    img = boulanger(img)
    n += 1
    plt.imshow(img)
    plt.title('n = {}'.format(n))
    plt.pause(1)
    if (img == picasso).all():
        break
```

Il faut maintenant 17 itérations avant de retrouver l'image initiale.