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 symétrie(img):
    p, q = img.shape[0], img.shape[1]
    img2 = np.empty_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.empty_like(img)
    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 :

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

Pour simplifier la fonction principale on commence par définir la fonction :

```
def photomat(k, d):
    if k % 2 == 0:
        return k // 2
    else:
        return k // 2 + d // 2
```

ce qui conduit à la définition suivante :

```
def photomaton(img):
    p, q = img.shape[0], img.shape[1]
    img2 = np.empty_like(img)
    for x in range(p):
        for y in range(q):
            img2[photomat(x, p), photomat(y, q)] = 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
while True:
    img = photomaton(img)
    plt.imshow(img)
    plt.show()
    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. Pour chaque pixel on doit appliquer n fois la fonction `photomat` sur chacune de ses coordonnées. Or il se trouve que l'abscisse d'un pixel a une période qui ne dépend que de sa valeur, et il en est de même pour son ordonnée. En d'autres termes, tous les points situés sur une ligne verticale reviennent sur cette ligne au bout du même nombre d'itérations, et il en est de même pour les lignes horizontales. 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.empty_like(img)
    ligne = []
    for x in range(p):
        u = x
        for _ in range(n):
            u = photomat(u, p)
        ligne.append(u)
    colonne = []
    for y in range(q):
        v = y
        for _ in range(n):
            v = photomat(v, q)
        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 180$^{\text{e}}$ 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 :

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

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.empty_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
while True:
    img = boulanger(img)
    plt.imshow(img)
    plt.show()
    if (img == picasso).all():
        break
```

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

Et pour les plus rapides

Question 9. On obtient la période d'un pixel en utilisant la fonction :

```python
def periode_pixel(x, y, p, q):
    n = 1
    x1, y1 = boulange(x, y, p, q)
    while (x1, y1) != (x, y):
        n += 1
        x1, y1 = boulange(x1, y1, p, q)
    return n
```

Question 10. Une fois la période calculée pour un pixel, on reporte sa valeur à tous les pixels de son orbite, et on ne fait le calcul que pour les pixels dont on ne connait pas encore la période :

```python
def tableau_des_periodes(img):
    p, q = img.shape[0], img.shape[1]
    r = np.zeros((p, q), dtype=int)
    for x in range(p):
        for y in range(q):
            if r[x, y] == 0:
                s = periode_pixel(x, y, p, q)
                x1, y1 = x, y
                for _ in range(s):
                    r[x1, y1] = s
                    x1, y1 = boulange(x1, y1, p, q)
    return r
```

Question 11. Pour obtenir la période d'une image on commence par extraire toutes les valeurs distinctes du tableau des périodes puis on calcule le ppcm de ces valeurs.

```python
def ppcm(a, b):
    return a * (b // gcd(a, b))

def periode_boulanger(img):
    p, q = img.shape[0], img.shape[1]
    r = tableau_des_periodes(img)
    lst = []
    for x in range(p):
        for y in range(q):
            if r[x, y] not in lst:
                lst.append(int(r[x, y]))
    m = 1
    for n in lst:
        m = ppcm(m, n)
    return m
```

On observera une petite subtilité dans le code ci-dessus : lorsqu'on crée un tableau NUMPY contenant des entiers, ces derniers sont codés sur 64 bits (ils appartiennent au type *int64*) et ne peuvent donc dépasser $2^{63}-1$; c'est le cas du tableau des périodes. Or pour de nombreuses images le ppcm des valeurs contenues dans le tableau des périodes dépasse cette borne. Je profite donc de leur recopie dans la liste `lst` pour les convertir au format *int* (qui lui n'est pas borné) de manière à ce que le calcul de ppcm soit correct.
Avec cette fonction on obtient la période de l'image `matisse.png` :

```
>>> periode_boulanger(matisse)
896568225229163143800
```

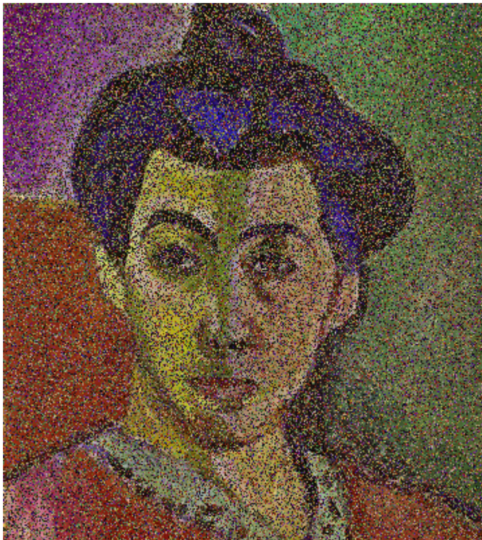
À raison de 10 images par seconde il faudrait plus de 28 milliards de siècles pour afficher toutes ses transformations avant de retrouver l'image initiale.

Question 12. Pour chaque pixel (x,y) non encore traité on calcule sa position finale en itérant la fonction `boulange` un nombre de fois égal à $n \bmod r(x,y)$. On traite ensuite les pixels (x_1,y_1) de son orbite en poursuivant l'itération de la fonction `boulange` et en remplaçant $r[x_1,y_1]$ par zéro de manière à détecter les pixels qui ont été traités.

```python
def boulanger2(img, n):
    p, q = img.shape[0], img.shape[1]
    r = tableau_des_periodes(img)
    img2 = np.empty_like(img)
    for x in range(p):
        for y in range(q):
            if r[x, y] != 0:
                u, v = x, y
                for _ in range(n % r[x, y]):
                    u, v = boulange(u, v, p, q)
                x1, y1 = x, y
                for _ in range(r[x, y]):
                    img2[u, v] = img[x1, y1]
                    x1, y1 = boulange(x1, y1, p, q)
                    u, v = boulange(u, v, p, q)
                    r[x1, y1] = 0
    return img2
```

Pour répondre à la dernière question nous avons besoin d'une fonction qui calcule le pourcentage de pixels identiques que partagent deux images :

```python
def pixels_communs(img1, img2):
    p, q = img1.shape[0], img1.shape[1]
    s = 0
    for x in range(p):
        for y in range(q):
            if (img1[x, y] == img2[x, y]).all():
                s += 1
    print('pourcentage de pixels communs : {} %'.format(np.floor(s / p / q * 100)))
```



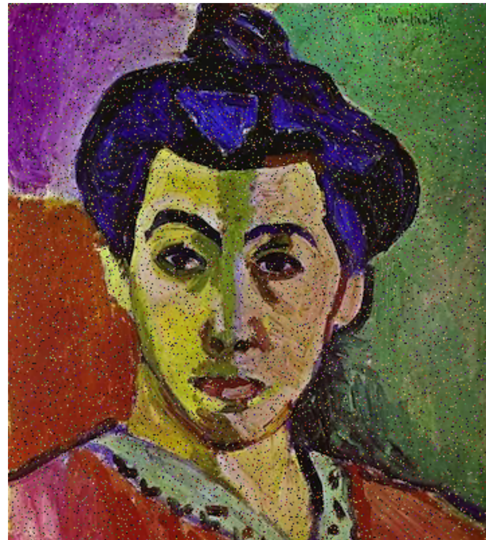


Figure 2 – Les itérations de l'image matisse.png pour $n = 67\,911$ et $n = 1\,496\,775\,000\,382\,576\,200$.

On obtient :

```
>>> pixels_communs(matisse, boulanger2(matisse, 67911))
pourcentage de pixels communs : 47.0 %

>>> pixels_communs(matisse, boulanger2(matisse, 1496775000382576200))
pourcentage de pixels communs : 89.0 %
```