AIV1 - TP06
Codage et approximation d'un contour

Suliac Lavenant et Antoine Nollet

25 Febuary 2022

Table des matières

1 Code Python

Nous nous intéressons, dans ces travaux, au codage et à l'approximation de contours de forme dans le cadre du traitement d'images. Voici un premier code permettant de récupérer les points de contours d'une forme.

```
1 def contour( image ):
2     c, h = cv.findContours( image, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
3     c = np.array( c[0] ).astype( float )
4     return (c[...,0] + 1j * c[...,1]).transpose()[0]
```

Cette première fonction "contour(image)" renvoie le contour d'un objet. La fonction "cv.findContours()" renvoie les contours de l'image et leur hiérarchie en prenant l'image en argument. Le deuxième argument "cv.RETR_EXTERNAL" sert à indiquer que l'on veut récuperer que les contours extèrieur. Et le troisième argument est la méthode d'approximation de contour, "cv.CHAIN_APPROX_NONE" signifie que tout les points de contours sont stocker et qu'il y en a 524.

La deuxième ligne sert à retirer "dtype=int32" du tableau et a transformer le type de son contenu de int32 à float.

Enfin, la fonction retournera la transposée de la colonne dont les valeurs sont les sommes des valeurs de la première colonne avec les valeurs de la deuxième colonne multipliées par i. Ainsi, en utilisant la fonction contour, nous aurons une liste de nombres complexes correspondants aux points du contour (x+iy , le point sera placée en (x,y)).

Une première exécution donne par exemple le résultat suivant :

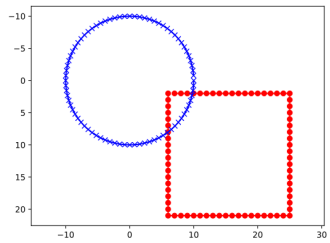


FIGURE 1 – Image retournée

On y voit le contour du cercle contenu dans un fichier nommé "aiv1-cercle-80.csv" ainsi que le contour du carré de l'image nommé "aiv1-carre-80.png".

2 Descripteurs de Fourier

2.1 Description

Nous nous intéressons donc à l'utilisation des descripteurs de Fouriers afin de coder de manière compacte le contour d'une forme. Dans cette partie nous manipulons cette forme carré pour illustrer les propos :

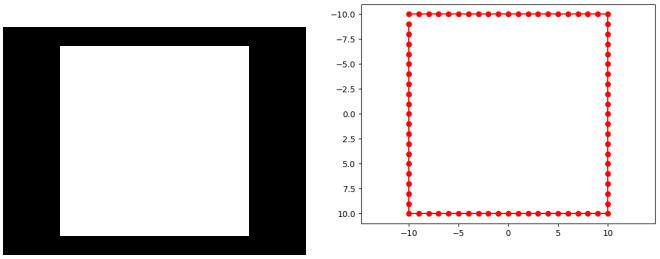


FIGURE 2 – Image du carré et contour récupéré

En utilisant la fonction contour sur l'image de droite, nous avons récupéré les points de contour illustrées sur l'image de gauche.

Pour coder au mieux ces points, nous pouvons utiliser la fonction numpy.fft.fft afin d'appliquer une transformé de fourier et d'obtenir des descripteurs de fourier. Nous pourrons ensuite appliquer une transformer inverse de fourier, par la fonction numpy.fft.ifft, pour récupérer les contours.

Nous obtiendrons une transformée sous la forme :

Z0	Z1	Z2	...	Z-3	Z-2	Z-1

Où chaque Zn correspondra à - Z-n. Hormis pour la valeur du descripteur Z0, car ce descripteur correspond au barycentre de la forme.

Donc en se basant sur la nature de Z0, en le modifiant avant de faire une transformée de fourier inverse on devrait pouvoir changer le barycentre.

Par exemple en rajoutant (40+20j) à Z0 on obtient le résultat suivant :

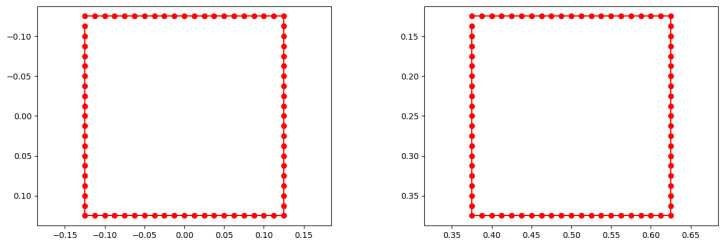


FIGURE 3 – à gauche le contour Carré normalisé et à droite après modification du Z0

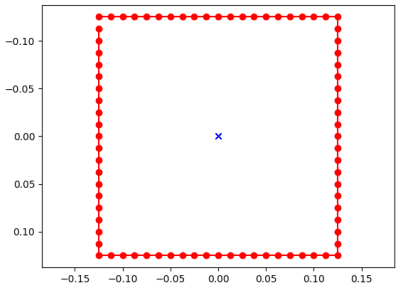
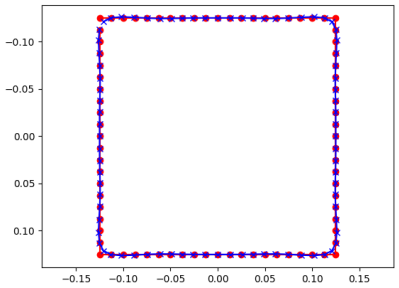
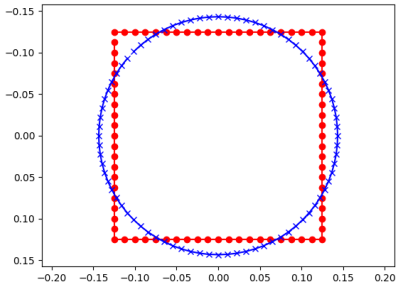
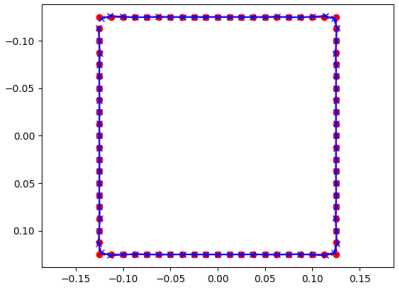
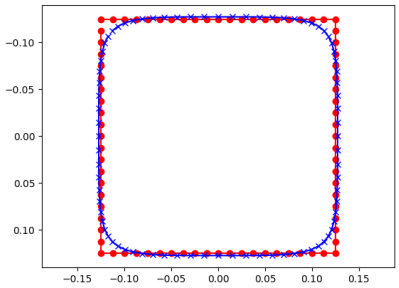
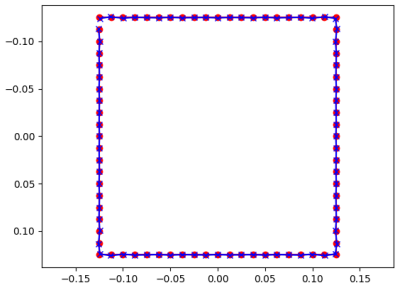
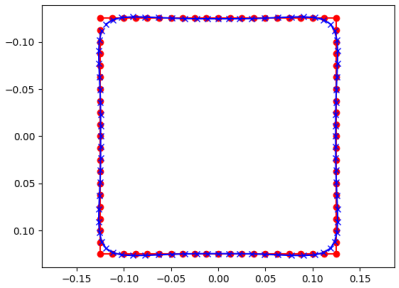
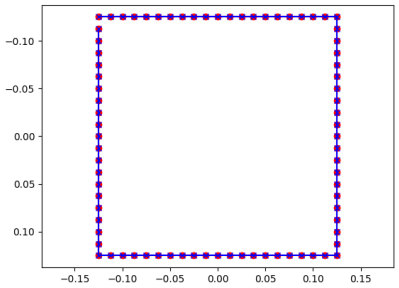
On remarque aux coordonnées que le barycentre a bien été modifié et décalé de 0.5 en x et de 0.25 en y. Donc modifier le descripteur Z0 modifie la position du barycentre, que se passera t-il si on modifie les autres descripteurs ?

2.2 Filtrage

Afin de coder le contour et de rendre ce codage plus compacte que possible, nous allons raisonnablement l'approximer.

Afin d'approximer le contour, codé avec des descripteurs de fourier, nous allons simplement annulé (mettre à 0+0i) un certain nombre de ces descripteurs. Les descripteurs du milieu sont ceux codant les détails de la forme, nous annulerons donc les descripteurs en partant du milieu et en annulant par pair Zi et Z-i.

Voici un tableau des contours obtenues selon le ratio des descripteurs conservés (ratio 1 = 100% conservé) (en rouge le descripteurs initial et en bleue celui avec le ratio) :

ratio	contour obtenu	ratio	contour obtenu
0		0.3	
0.05		0.5	
0.1		0.75	
0.2		1	

On remarque qu'avec un ratio de 0 on a juste notre barycentre qui apparait et qu'avec un ratio de 0.05 on a juste un cercle (qui correspond à deux descripteurs). Dès 0.1, nous obtenons une approximation très pertinente du contour de la forme. Puis en augmentant le ratio, le contour de fourier devient exponentiellement plus fidèle. Enfin, à partir de 0.5 nous avons un contour qui se confond avec l'original.

3 Réduction d'une chaîne de contour

Une autre manière de coder le contour de manière plus compacte et de lui appliquer l'algorithme de la corde.

Le principe de l'algorithme est de simplifier le contour en attribuant moins de points pour représenter approximativement le même contour. L'idée est de considérer une "corde" entre deux points "extrémités" du contour et de voir les distances des autres points (points entre les extrémités) à cette corde. Puis si le point le plus éloigné

n'est pas trop proche (selon un seuil) de la corde, deux cordes, des extrémités à ce point, sont considérés et nous réitérerons jusqu'à ce qu'il n'y ait plus de corde à créer.

le code suivant nous est fourni :

```
def reduit( cont, dmax ):
    # Calcul des distances entre les points et la corde
    d = distances( cont )
    # Si la distance maximale est inferieure au seuil, retourner les extremites
    if np.max( d ) <= dmax:
      return [cont[0], cont[-1]]
    else:
      # Indice du point le plus eloigne de la corde
      loin = np.argmax( d )
      # Reduire les deux sous-chaines de contour
      cont1 = reduit( cont[:(loin+1)], dmax )
      cont2 = reduit( cont[loin:], dmax )
      # Enlever un point et concatener
      return np.concatenate( [cont1, cont2[1:]] )
```

Il nous manque le calcul de distance. Nous avons en donnée les deux points extrémités du segment de la corde ($cont[0] et cont[-1]$) et chaque points dont nous voulons la distance ($cont$). On peut donc calculer cette distance par le plan ou par l'espace.

Nous avons donc décider de calculer ces distances dans l'espace selon la formule :

$$d(A,(d)) = \frac{\left\| \overrightarrow{BA} \wedge \overrightarrow{u} \right\|}{\| \overrightarrow{u} \|}$$

Où A est le point (un point de cont dont on veut la distance par rapport à la corde), d est la droite (notre corde), $\overrightarrow{u}$ est le vecteur directeur de d et $\overrightarrow{BA}$ le vecteur partant d'un point quelconque B de la droite (corde) et allant jusqu'à A.

On abouti sur la creation du code suivant :

```
def distancesEspace(cont):
    d = []
    b = [cont[0].real, cont[0].imag]
    c = [cont[-1].real, cont[-1].imag]
    u = [c[0]-b[0],c[1]-b[1]]

    for i in range(len(cont)):
        a=[cont[i].real, cont[i].imag]
        ba=[a[0]-b[0],a[1]-b[1]]
        bau = np.cross(ba,u)
        normeBau = np.linalg.norm(bau)
        normeU = np.linalg.norm(u)

        d.append(normeBau/normeU)

    return d
```

L'execution de cet algorithme Nous donne les résultat suivant pour les distances maximale 0.5 (bleue) et 1 (vert).

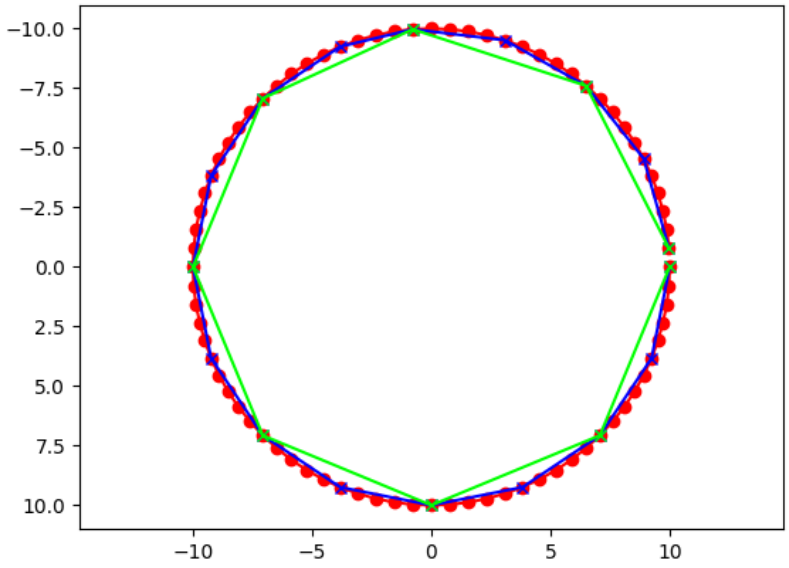


FIGURE 4 – En rouge le contour initial, en bleu la corde avec distance 0.5 et en vert la corde avec distance 1.

On peut constater que ces deux approximations de contours par l'algorithme de la corde restent assez représentative du contour initial. Cependant, on constate que plus le seuil de distance est grand, moins l'approximation du contour est précise. On préférera ici un seuil de distance à 0.5 (figure bleue) qui compresse le contour à environs un ratio de 1/5 (on a 1 point sur la figure pour 5 normalement).

4 Comparaison des deux approches

Pour ces comparaisons nous aurons le contour initial en rouge sur les graphiques, les contours avec l'algorithme de la corde en bleue et les contours de Fourier en vert.

Nous calculerons l'efficacité de la corde ainsi :

$ratio = nombredepointsdelacorde/nombredepointsintial$

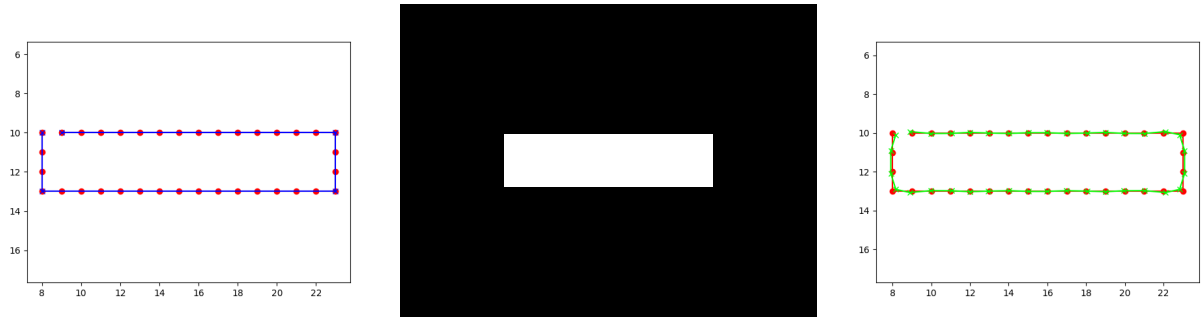


FIGURE 5 – Contours de l'image de rectangle. À gauche codé par la corde, à droite par fourier.

Pour ce premier cas, pour la corde, le seuil n'influe pas vraiment comme il s'agit de lignes droites assez espacées. On se limite donc à 5 points pour la corde. En comparaison pour fourier, avec un ratio de 0.5, à 18 descripteurs. Pour un cas avec des lignes droites comme un rectangle, la corde semble préférable car elle pourra représenter une ligne droite par seulement 2 points.

Ces interprétations sont confirmées par un ratio de corde de 0.13 contre un ratio de 0.5 pour les descripteurs de fourier.

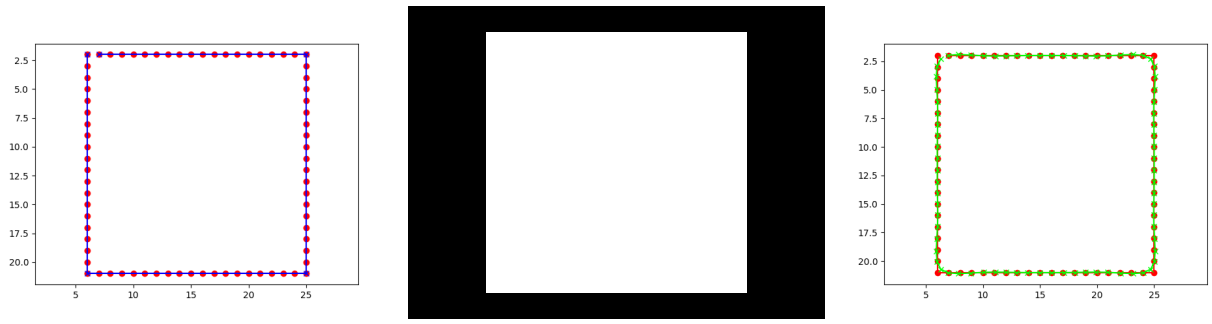


FIGURE 6 – Contours de l'image de carre. À gauche codé par la corde, à droite par fourier.

Ce deuxième cas est semblable au rectangle. Les descripteurs de fourier sont même ici encore moins efficaces (ratio de 0.5) alors que la corde est tout aussi efficace.

Un ratio de 0.5 pour les descripteurs de fourier contre un ratio de 0.06 pour l'algorithme de la corde confirme encore ces interprétations.

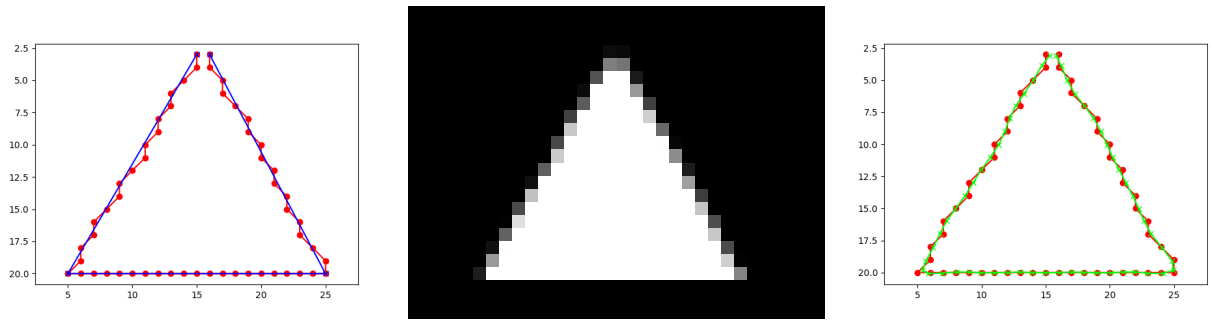


FIGURE 7 – Contours de l'image de triangle. À gauche codé par la corde, à droite par fourier.

Ici il faut se poser la question de la pertinence de l'approximation du contour. Les crans du triangles sont ils à conservés ou faut il se concentrer sur la forme en triangle ? On préfèrera garder la forme en triangle sans les crans. Ainsi, le ratio des descripteurs de fouriers sera de 0.5 contre un ratio de 0.14 avec l'algorithme de la corde. On préférera donc utiliser la corde dans ce cas là. Si les crans du triangle sont à conservés, les descripteurs de fourier seront plus pertinent à utiliser car le ratio corde sera bien plus important.

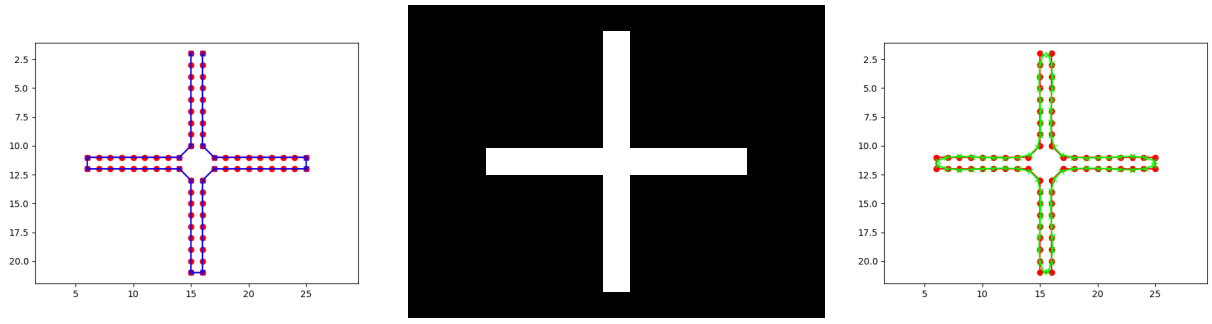


FIGURE 8 – Contours de l'image de croix. À gauche codé par la corde, à droite par fourier.

Pour cette image de la croix la corde semble plus pertinente avec les grandes lignes droites (comme vu précédement), cela est confirmé par le ratio de la corde de 0.22 contre celui de fourier à 0.4.

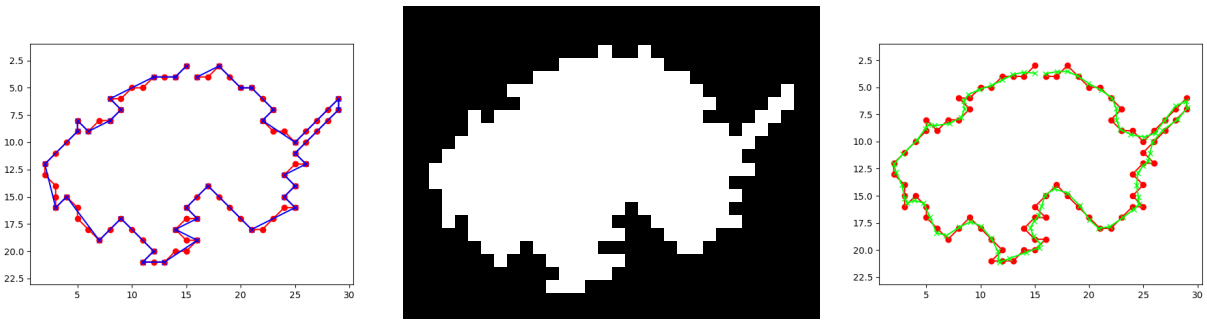


FIGURE 9 – Contours de l'image de patate. A gauche Codé par la corde, a droite par fourier.

Pour la patate on remarque une grande ressemblence entre la corde et fourier, de plus les deux on des ratio assez proche, respectivement 0.48 et 0.5. On peut déduire de cela que plus le contour sera complexe avec plein de petites irrégularités, plus fourier sera préférable.

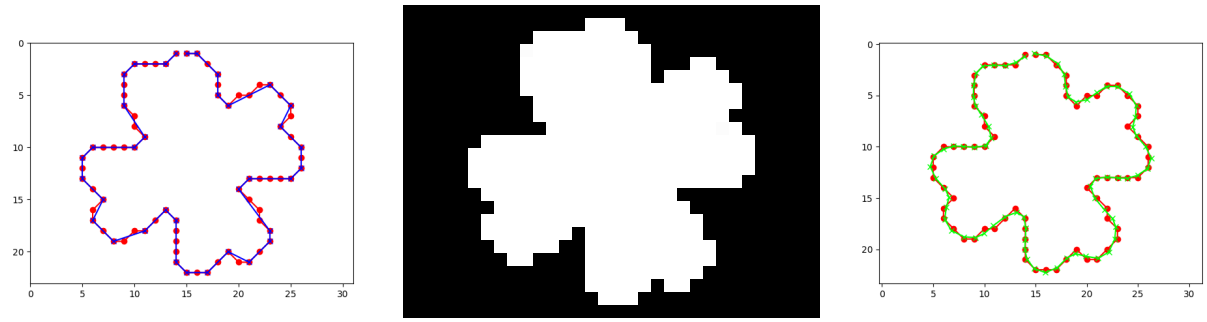


FIGURE 10 – Contours de l'image de trèfle. A gauche codé par la corde, a droite par fourier.

Comme pour la patate, ce trèfle a des contours similaire pour fourier et pour la corde, respectivement de ratio 0.45 et 0.48. Cela confirme nos deduction : plus un contour est complexe plus fourier sera préférable à la corde.

Pour conclure sur la comparaison de ses deux approches, l'algorithme de la corde sera beaucoup plus efficace sur des contours "plat" et long comme des carrée, rectangle, triangle et croix alors qu'à l'inverse fourier sera plus optimiser sur des contours arrondis ou complexes.