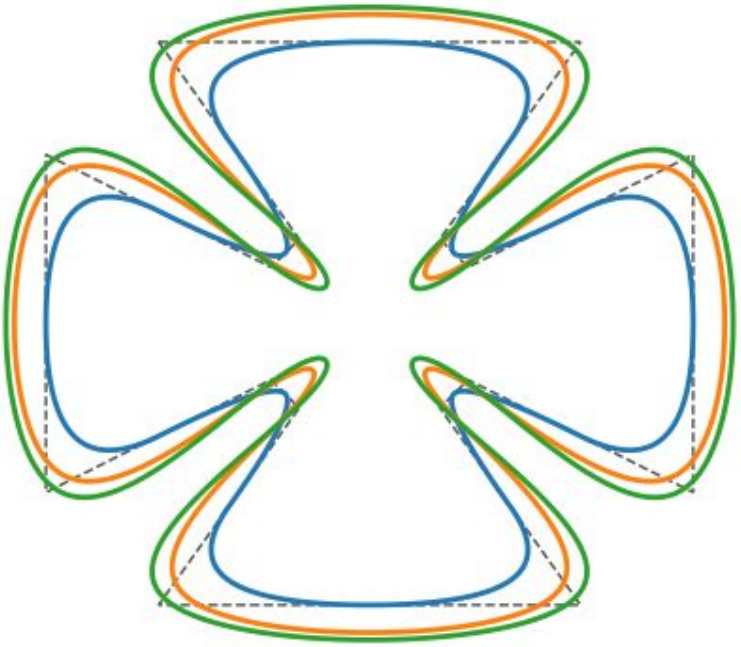
RAPPORT GÉOMÉTRIE NUMÉRIQUE

T.P. 5 - Algorithme Lane-Riesenfeld



Calvin Massonnet

08/03/2021 - 14/03/2021

Université Grenoble Alpes - Master Informatique

Algorithme Lane-Riesenfeld

L'algorithme Lane-Riesenfeld permet de créer des courbes de subdivision spline uniformes. Celui-ci, ainsi que ses variantes, sont à placer dans une boucle afin d'itérer la subdivision et obtenir une courbe lisse. La première boucle de l'algorithme de la figure 1 permet d'initialiser le tableau de points de sortie avec les mêmes points que le tableau d'entrée, mais en les dédoublants. Si cette étape se retrouve manquante, le polygone ne fera que rapidement rétrécir en gardant le même nombre de points qu'au départ. La seconde boucle avec l'imbrication permet ensuite de calculer la position des points du nouveau polygone en fonction de la position des points du précédent polygone. En effet, la position de ces nouveaux points est le centre du segment séparant deux points liés du polygone précédent. Le degré correspond au nombre d'étapes de moyennage d'une étape de subdivision. Plus ce degré est élevé, plus la courbe est rétrécie et la forme de la courbe éloignée du polygone de départ. Les figures 7 à 11 détaillent ce changement (2 : trait continu ; 3 : trait segmenté ; 5 : trait-point ; 10 : trait-point-point ; 30 : pointillé).

```
def LaneRiesenfeld(X0, degree):
	# number of points
	n = X0.shape[0]

	# upsample
	X1 = np.zeros([2 * n, 2])

	# duplicate points
	for i in range(n * 2):
		X1[i] = X0[i / 2]

	# Lane-Riesenfeld algorithm
	for d in range(1, degree):
		tmp = X1.copy()
		for i in range(n * 2):
			V = tmp[i % (n * 2)] + tmp[(i + 1) % (n * 2)]
			X1[i % (n * 2)] = (1.0 / 2.0) * V

	return X1
```

Figure 1 : algorithme Lane-Riesenfeld

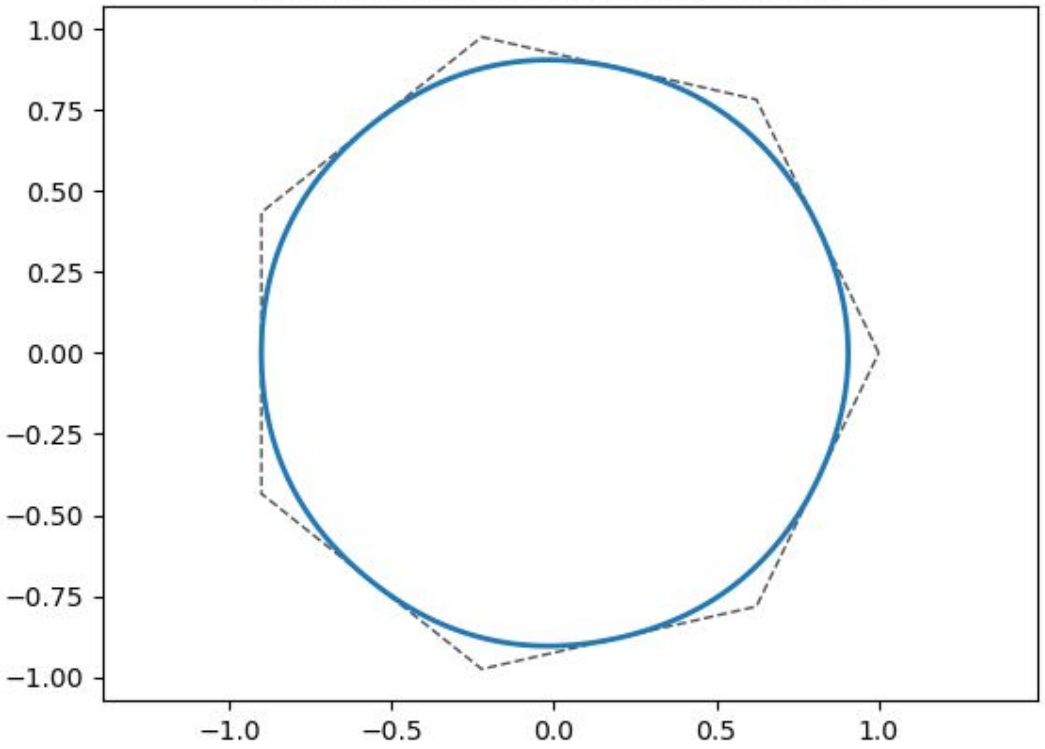


Figure 2 : “hepta” de degré 3, à 5 subdivisions avec l’algorithme Lane-Riesenfeld

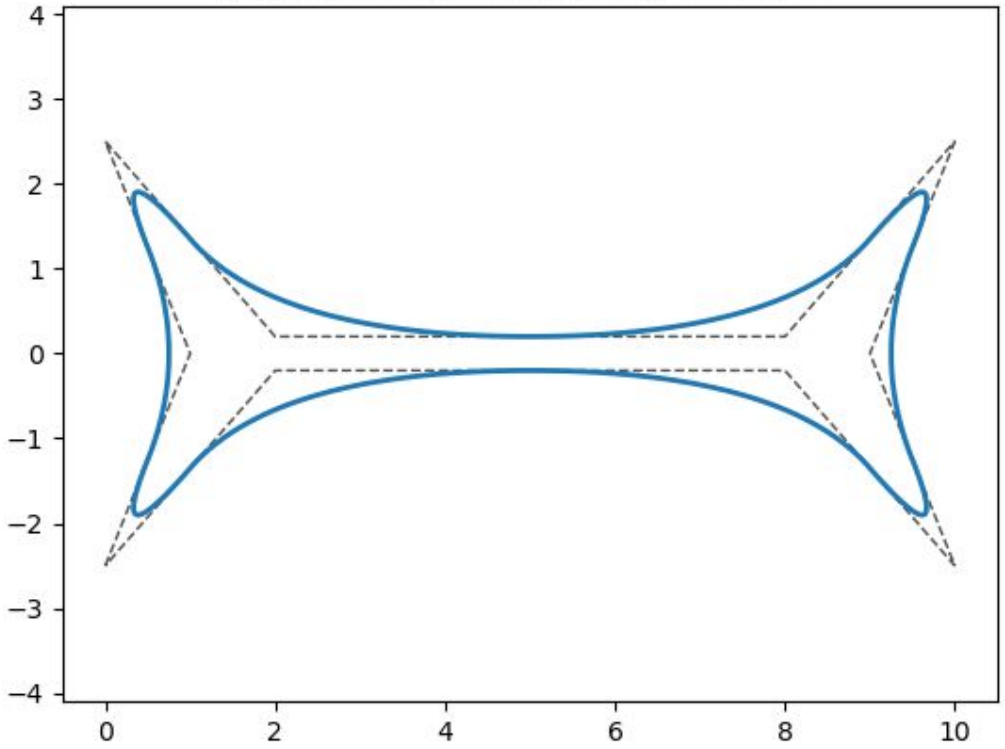


Figure 3 : “bone” de degré 3, à 5 subdivisions avec l’algorithme Lane-Riesenfeld

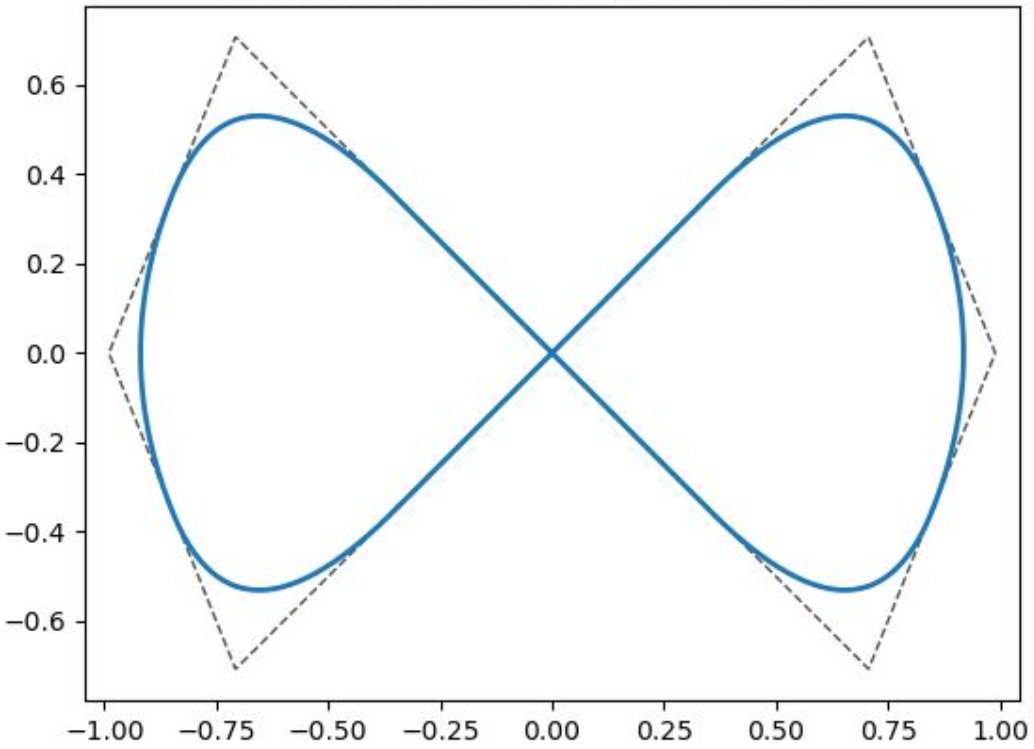


Figure 4 : “infinity” de degré 3, à 5 subdivisions avec l’algorithme Lane-Riesenfeld

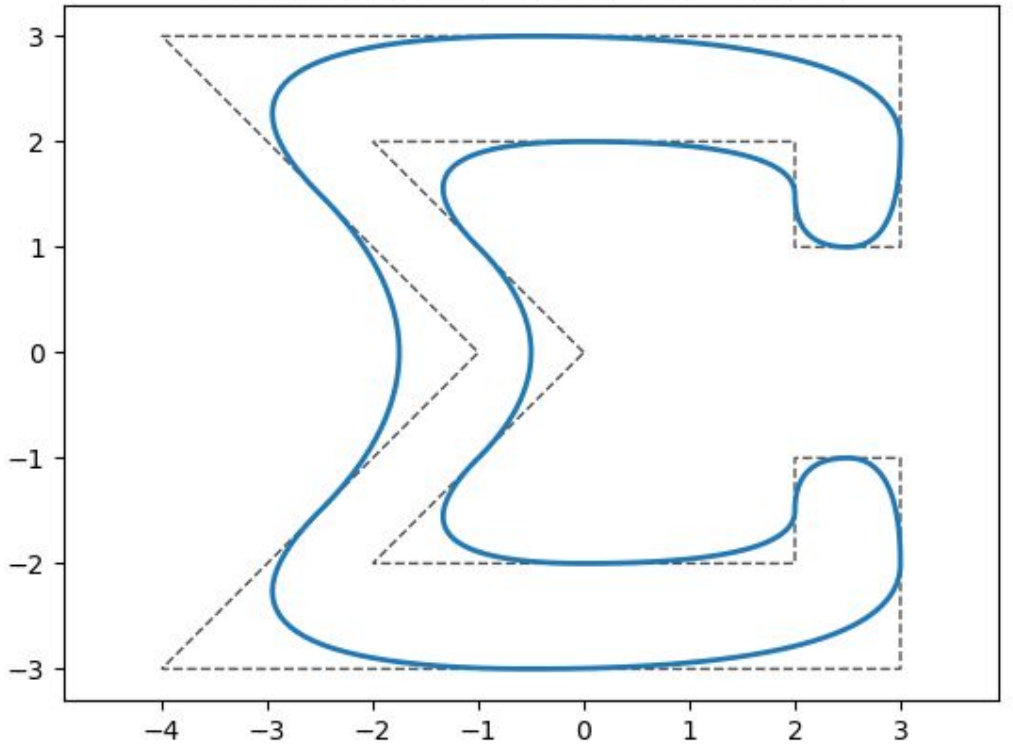


Figure 5 : “sumsign” de degré 3, à 5 subdivisions avec l’algorithme Lane-Riesenfeld

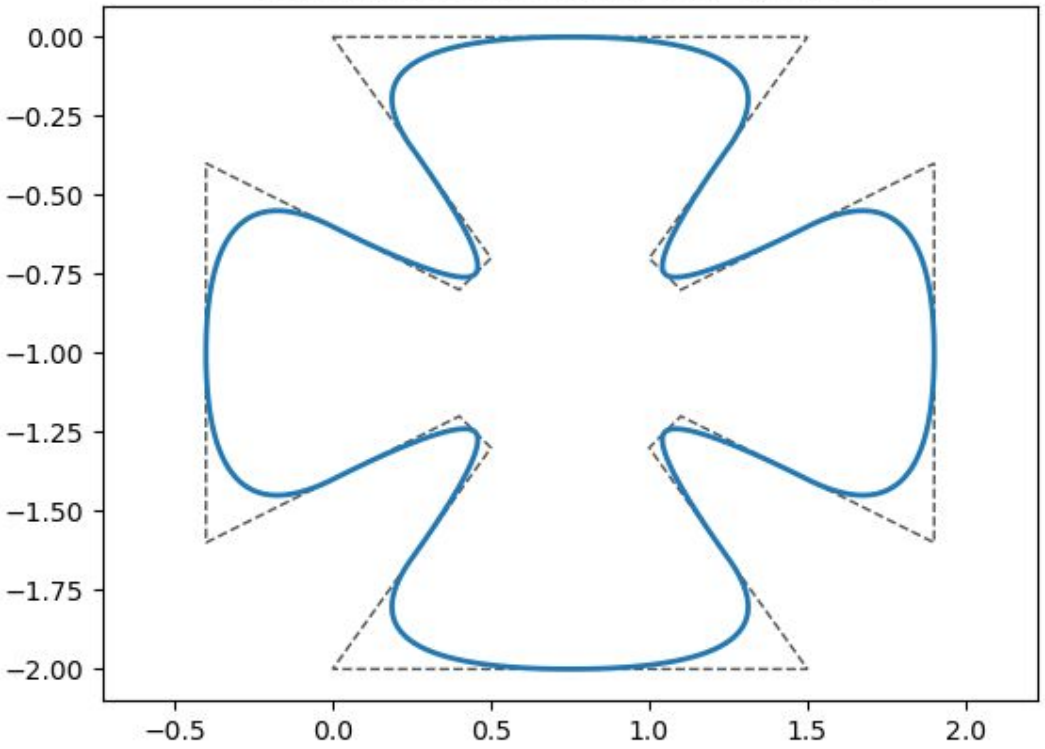


Figure 6 : “clover” de degré 3, à 5 subdivisions avec l’algorithme Lane-Riesenfeld

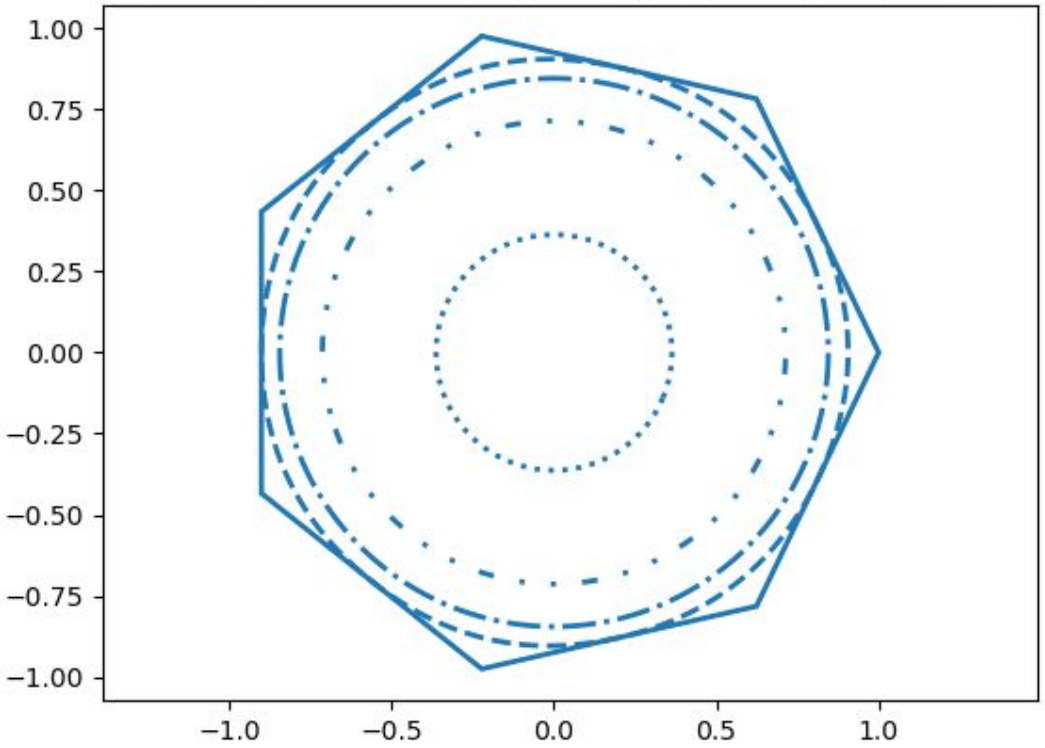


Figure 7 : “hepta” de degré 2, 3, 5, 10 et 30, à 5 subdivisions avec l’algorithme Lane-Riesenfeld

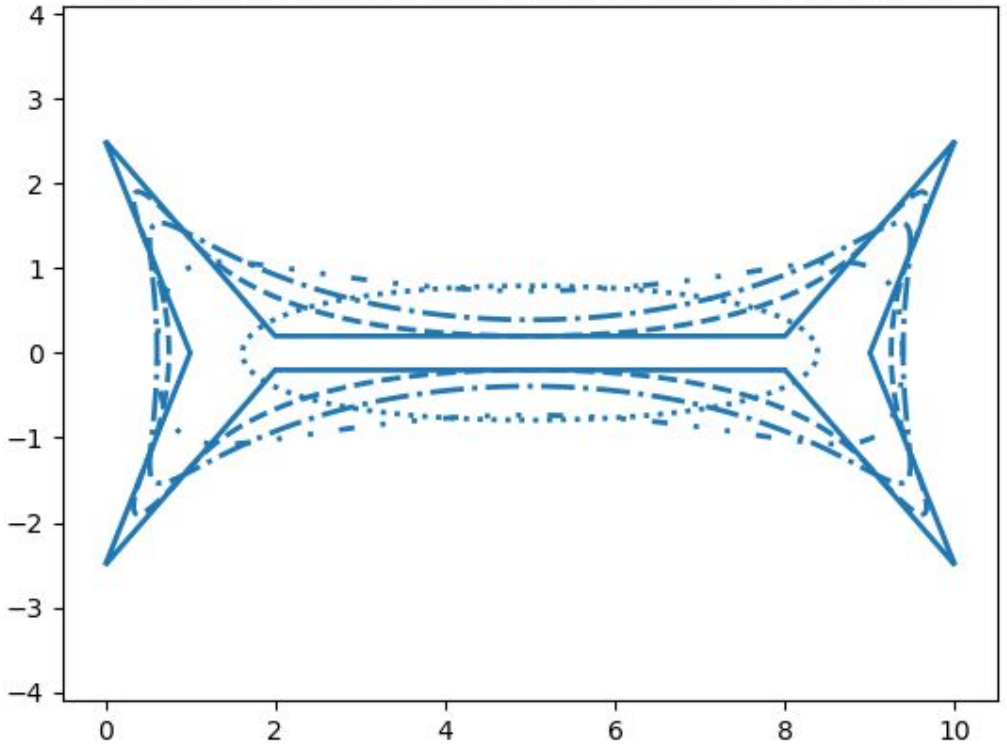


Figure 8 : “bone” de degré 2, 3, 5, 10 et 30, à 5 subdivisions avec l’algorithme Lane-Riesenfeld

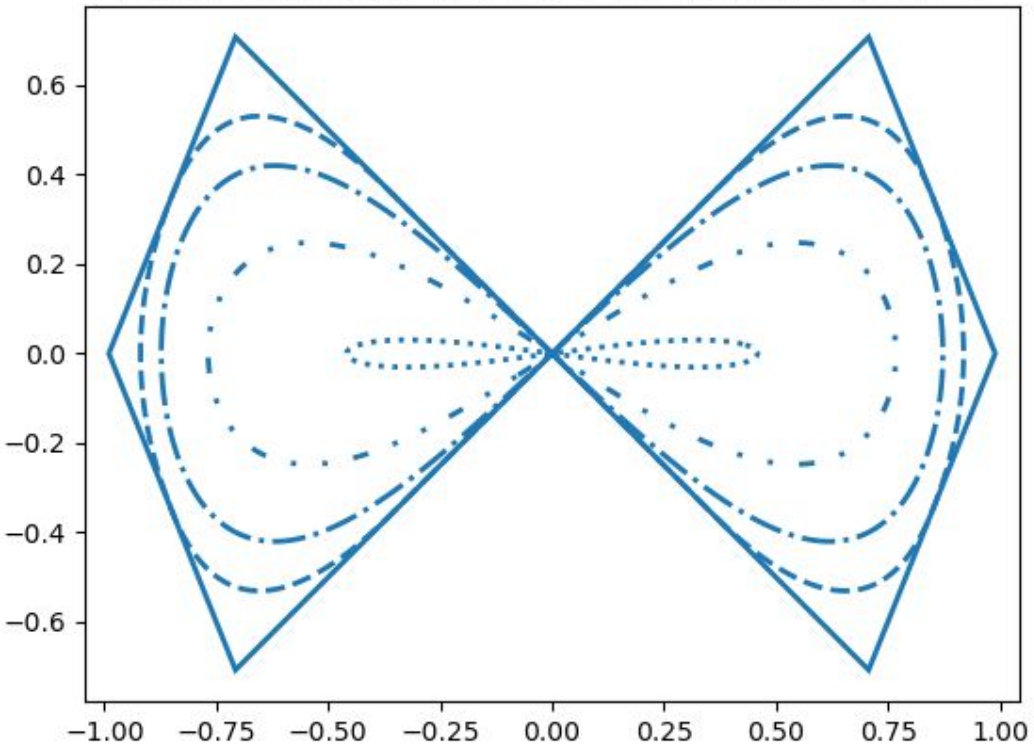


Figure 9 : “infinity” de degré 2, 3, 5, 10 et 30, à 5 subdivisions avec l’algorithme Lane-Riesenfeld

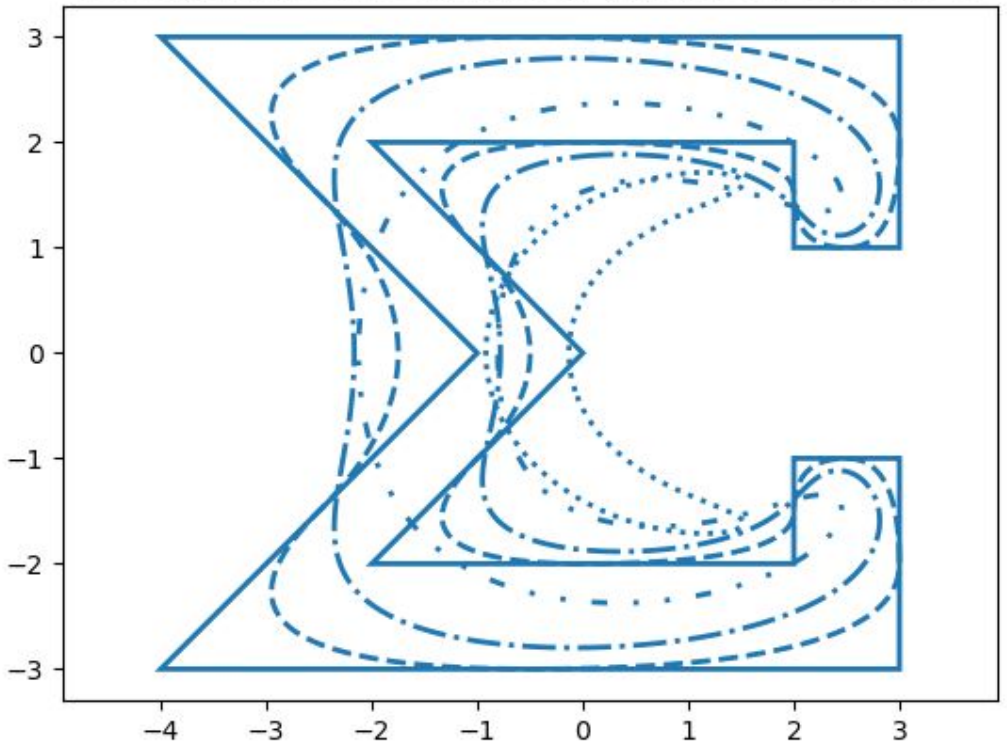


Figure 10 : “sumsign” de degré 2, 3, 5, 10 et 30, à 5 subdivisions avec l’algorithme Lane-Riesenfeld

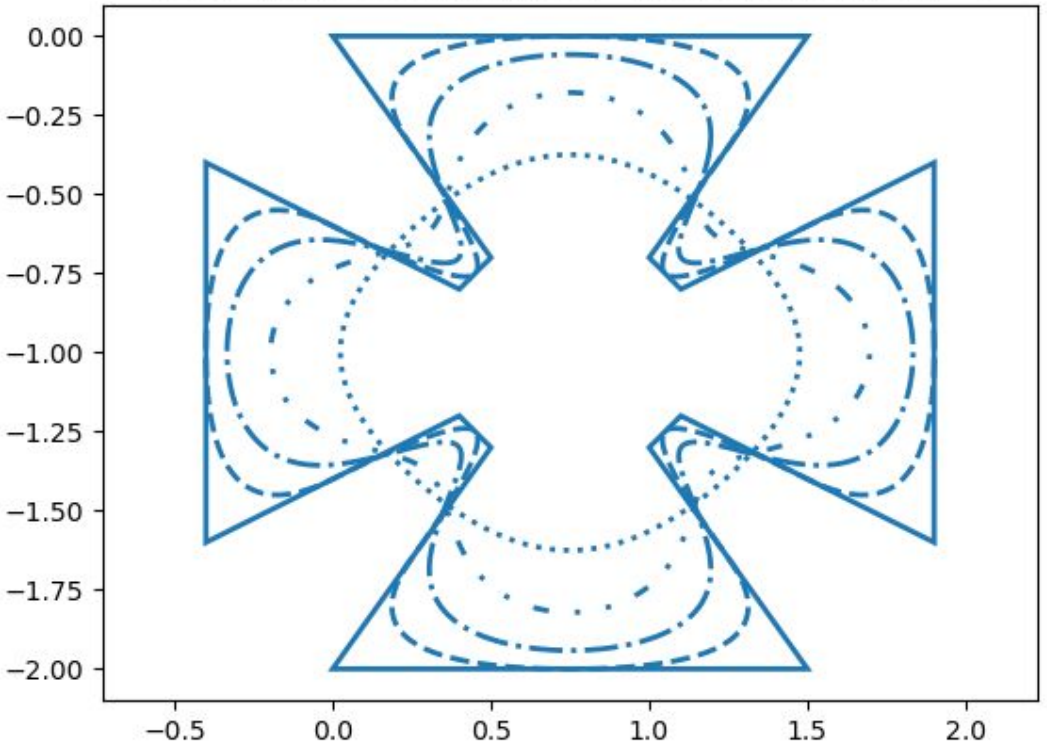


Figure 11 : “clover” de degré 2, 3, 5, 10 et 30, à 5 subdivisions avec l’algorithme Lane-Riesenfeld

Variante : Four-point

L'une des variantes de l'algorithme Lane-Riesenfeld utilise la distance entre quatre points plutôt que deux avant d'en faire une moyenne afin de calculer la position des points des polygones suivants. La première boucle de la figure 12 sert également de dédoublement, comme pour l'algorithme original, mais calcule préemptivement la position des doublons afin d'obtenir la même figure que le polygone initial avec deux fois plus de points, comme présenté dans la figure 13. La seconde boucle utilise le même principe de moyennage que l'algorithme de la figure 1, mais pour quatre points. Similairement à l'algorithme Lane-Riesenfeld, plus le degré est élevé, plus la courbe se rétrécit. Cependant, pour le même degré, le changement est moins flagrant (figures 19 à 23).

```
def FourPoint(X0, degree):
      # number of points
      n = X0.shape[0]

      # upsample
      X1 = np.zeros([2 * n, 2])

      # duplicate points
      for i in range(n * 2):
            X1[i * 2] = X0[i]
            V = -1.0 * X0[(i - 1) % n] + 9.0 * X0[i % n]
              +  9.0 * X0[(i + 1) % n] - X0[(i + 2) % n]
            X1[i * 2 + 1] = (1.0 / 16.0) * V

      # 4-point LR scheme
      for d in range(1, degree):
            tmp = X1.copy()
            for i in range(n * 2):
                  V = -1.0 * tmp[(i - 1) % n] + 9.0 * tmp[i % n]
                    +  9.0 * tmp[(i + 1) % n] - tmp[(i + 2) % n]
                  X1[i % (n * 2)] = (1.0 / 16.0) * V

      return X1
```

Figure 12 : variante Four-point de l'algorithme Lane-Riesenfeld

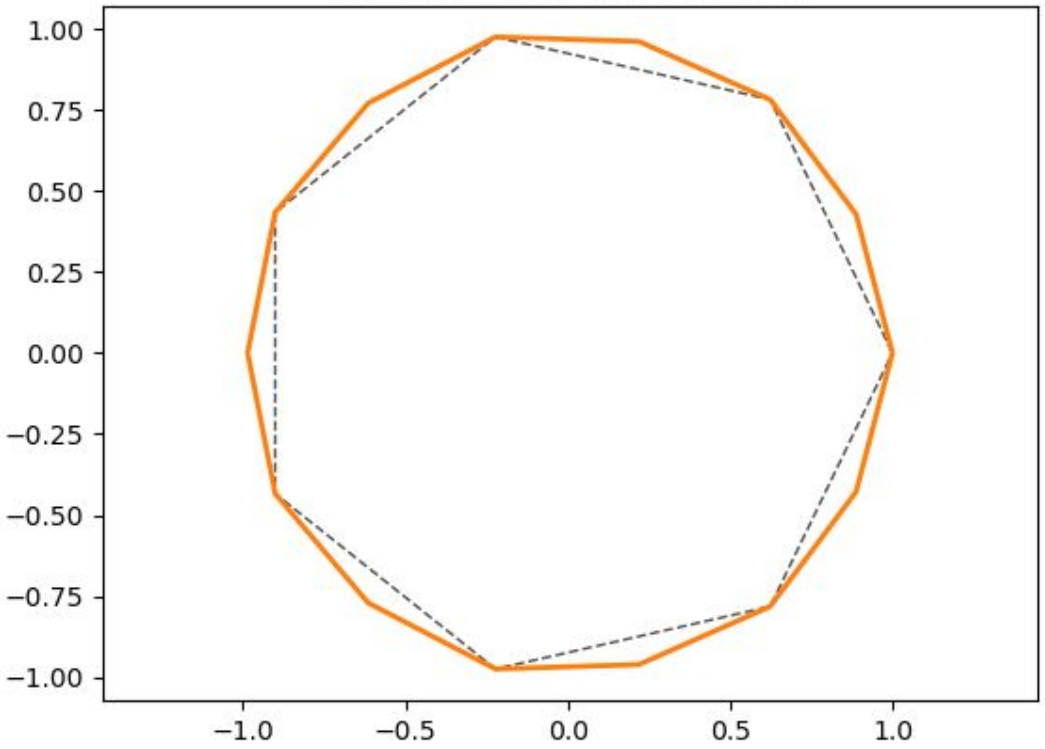


Figure 13 : “hepta” de degré 1 et 1 subdivision avec la variante Four-point

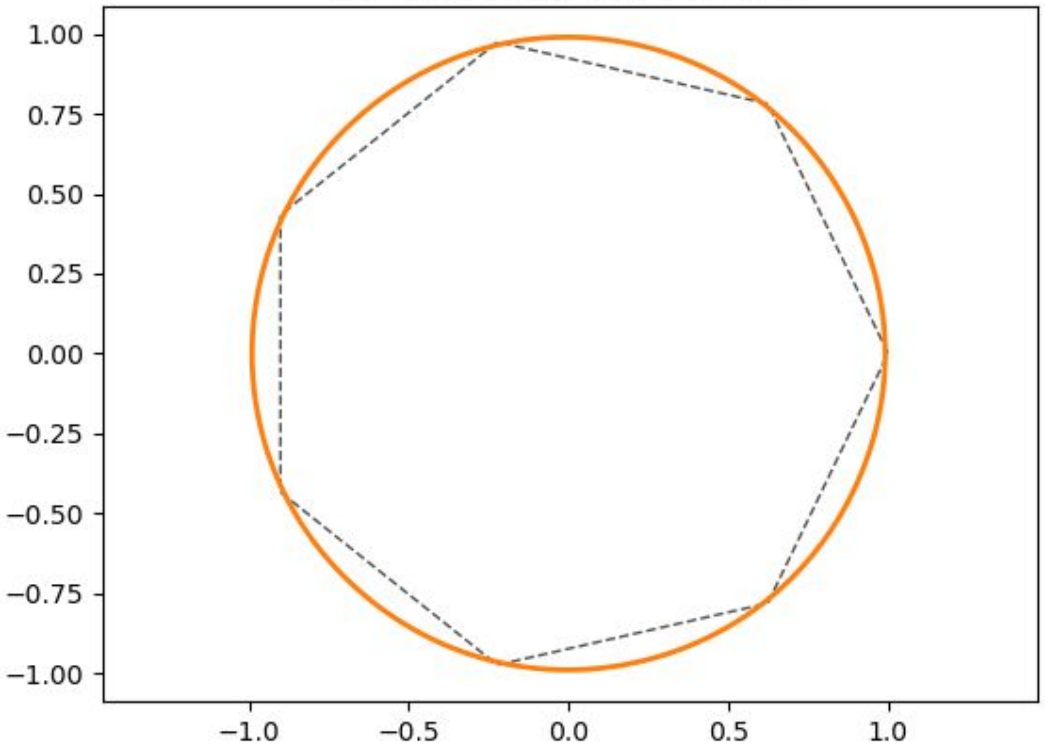


Figure 14 : “hepta” de degré 3, à 5 subdivisions avec la variante Four-point

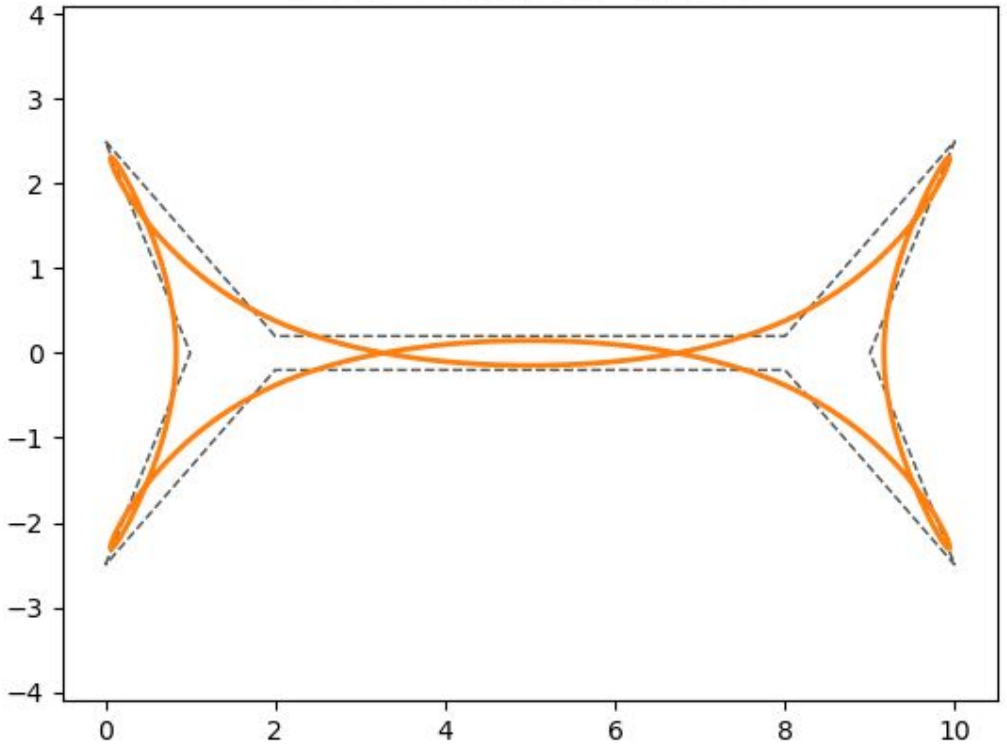


Figure 15 : “bone” de degré 3, à 5 subdivisions avec la variante Four-point

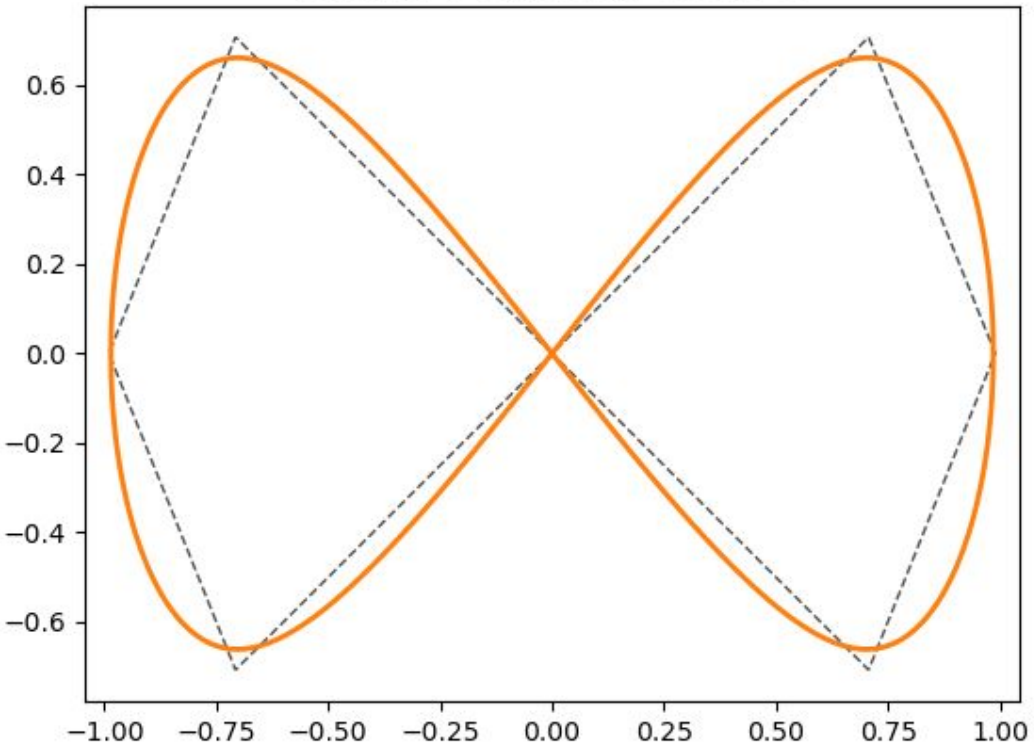


Figure 16 : “infinity” de degré 3, à 5 subdivisions avec la variante Four-point

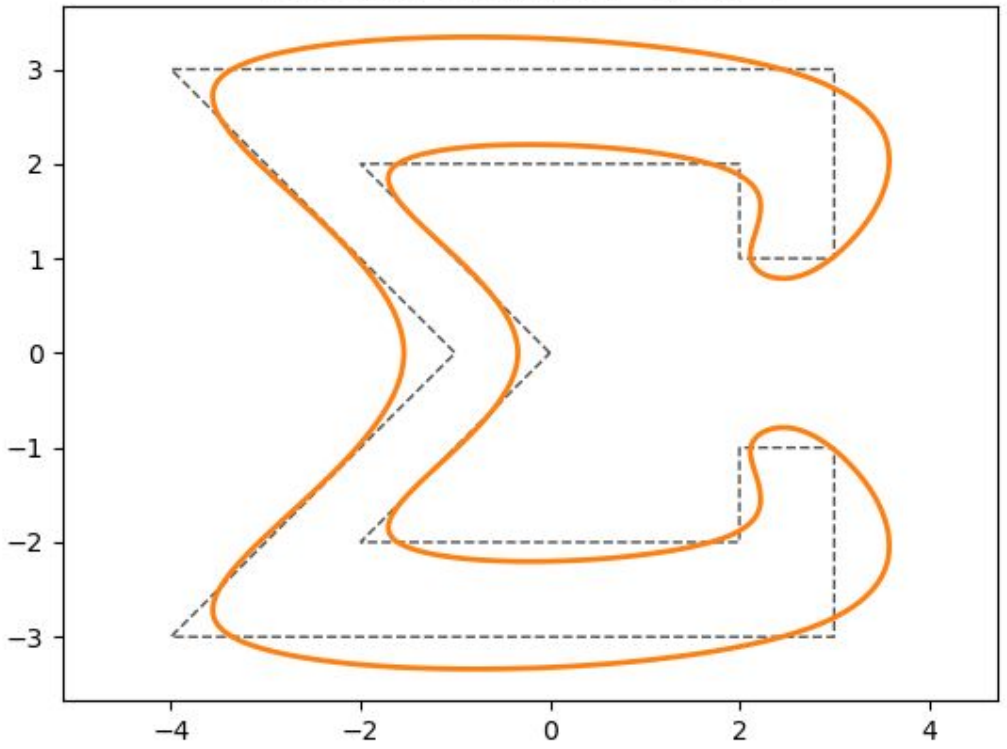


Figure 17 : “sumsign” de degré 3, à 5 subdivisions avec la variante Four-point

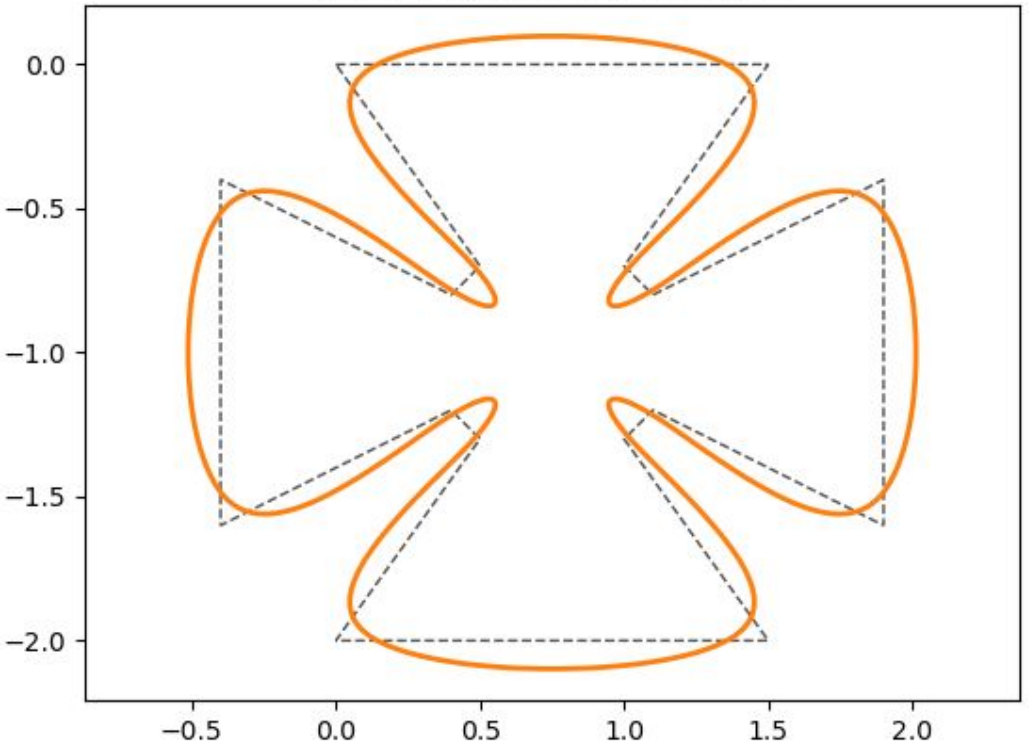


Figure 18 : “clover” de degré 3, à 5 subdivisions avec la variante Four-point

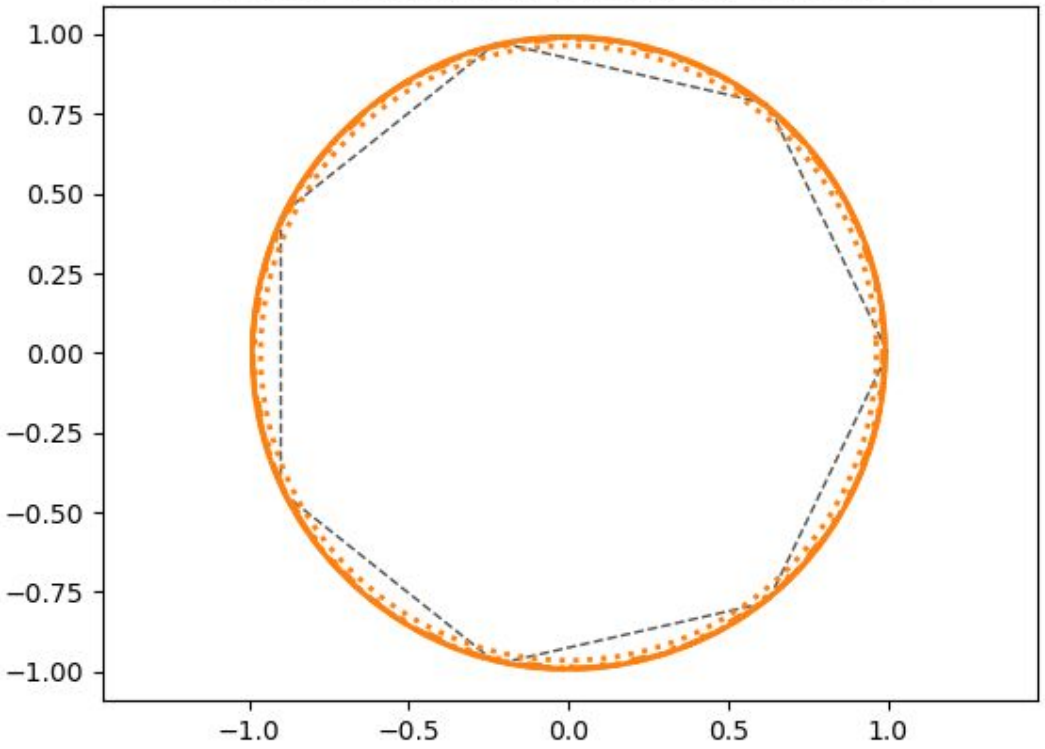


Figure 19 : “hepta” de degré 2, 3, 5, 10 et 30, à 5 subdivisions avec la variante Four-point

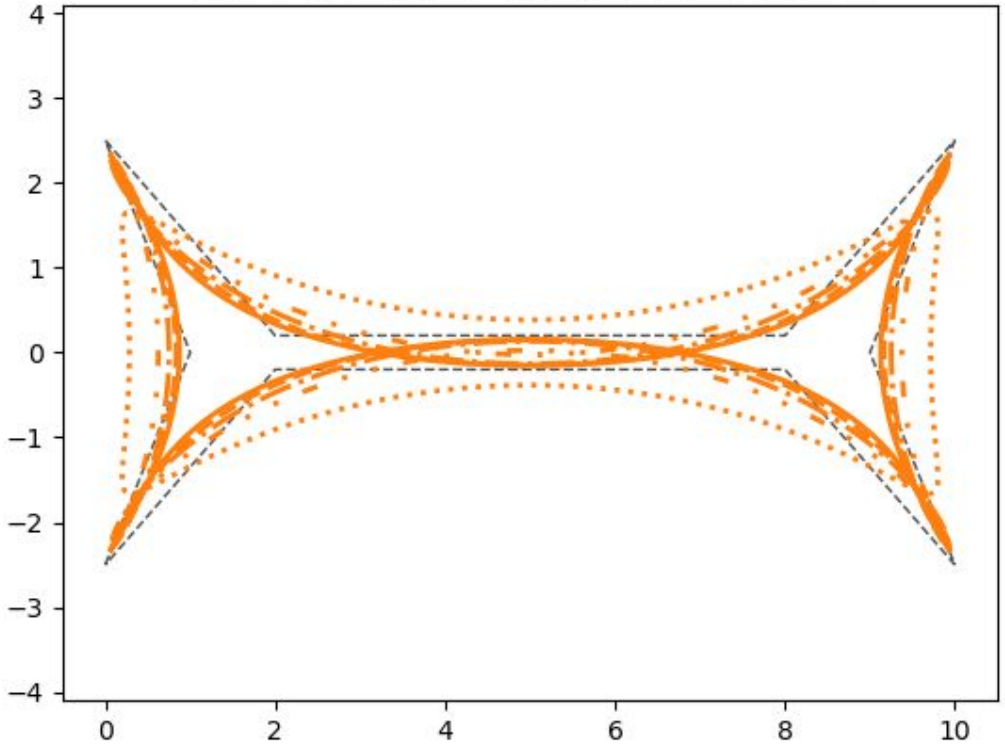


Figure 20 : “bone” de degré 2, 3, 5, 10 et 30, à 5 subdivisions avec la variante Four-point

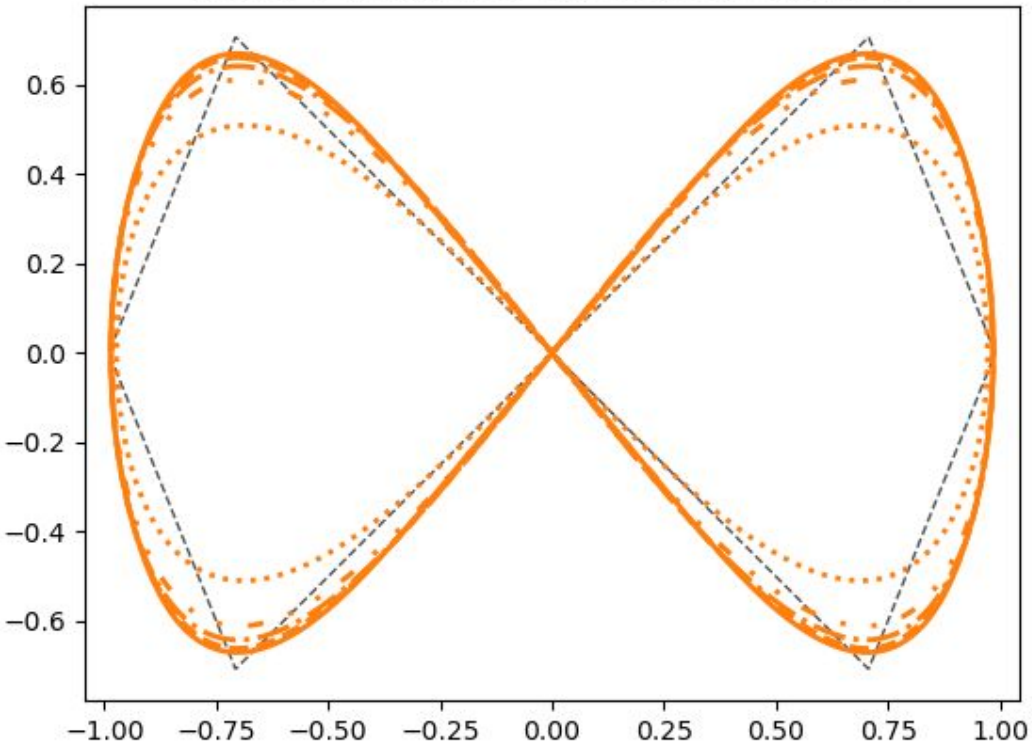


Figure 21 : “infinity” de degré 2, 3, 5, 10 et 30, à 5 subdivisions avec la variante Four-point

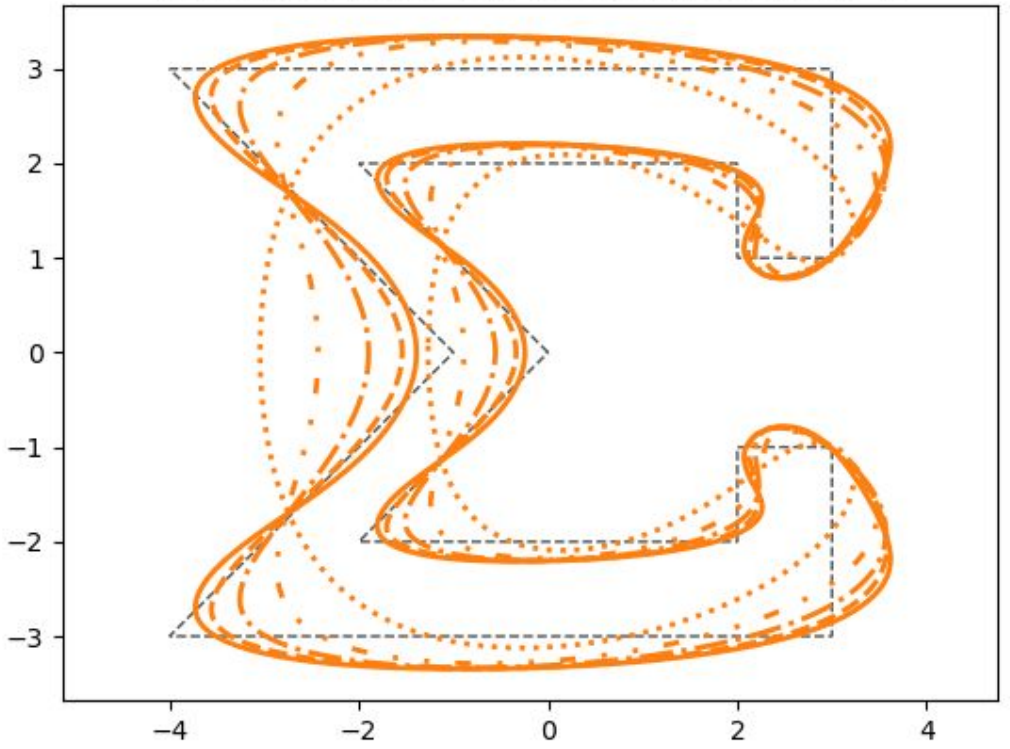


Figure 22 : “sumsign” de degré 2, 3, 5, 10 et 30, à 5 subdivisions avec la variante Four-point

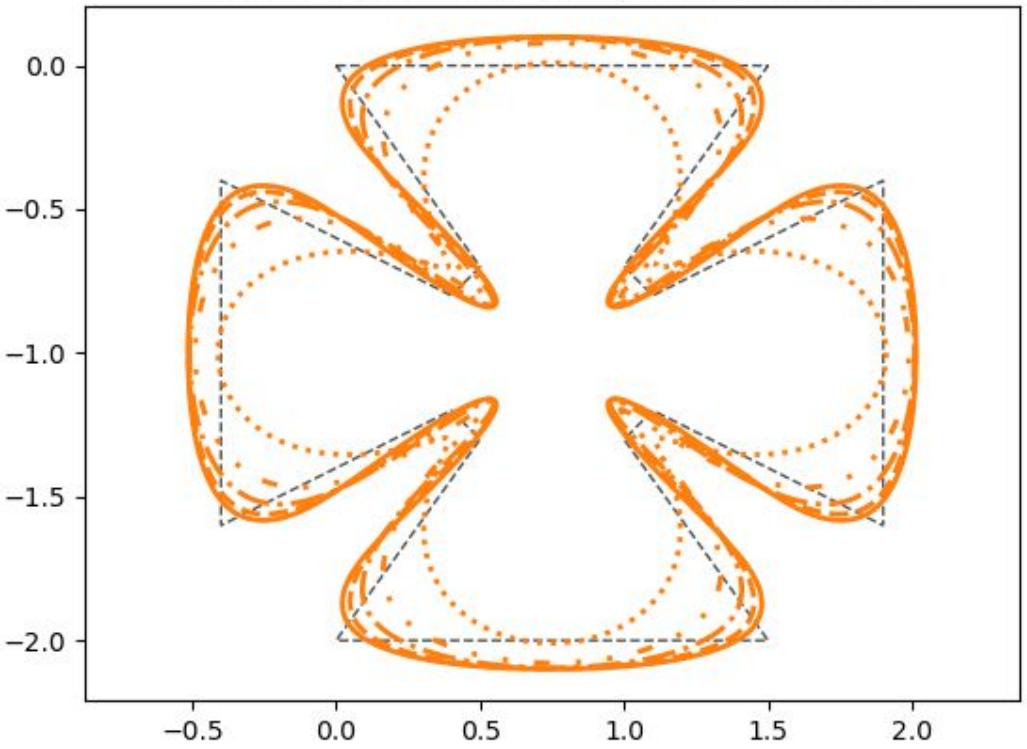


Figure 23 : “clover” de degré 2, 3, 5, 10 et 30, à 5 subdivisions avec la variante Four-point

Variante : Six-point

Une autre des variantes de l'algorithme Lane-Riesenfeld consiste à utiliser non quatre, mais six points à l'étape de moyennage. L'algorithme est similaire à celui de la variante *Four-point* mais avec six points, comme peut le montrer l'algorithme de la figure 24. Dans cette variante, la différence à la modification du degré devient bien moins perceptible qu'avec la variante précédente (figures 30 à 34).

```
def SixPoint(X0, degree):
      # number of points
      n = X0.shape[0]

      # upsample
      X1 = np.zeros([2 * n, 2])

      # duplicate points
      for i in range(n * 2):
            X1[i * 2] = X0[i]
            V = 3.0 * X0[(i - 2) % n] - 25.0 * X0[(i - 1) % n]
              + 150.0 * X0[i % n] + 150.0 * X0[(i + 1) % n]
              - 25.0 * X0[(i + 2) % n] + 3.0 * X0[(i + 3) % n]
            X1[i * 2 + 1] = (1.0 / 256.0) * V

      # 6-point LR scheme
      for d in range(1, degree):
            tmp = X1.copy()
            for i in range(n * 2):
                  V = 3.0 * tmp[(i - 2) % n] - 25.0 * tmp[(i - 1) % n]
                    + 150.0 * tmp[i % n] + 150.0 * tmp[(i + 1) % n]
                    - 25.0 * tmp[(i + 2) % n] + 3.0 * tmp[(i + 3) % n]
                  X1[i % (n * 2)] = (1.0 / 256.0) * V

      return X1
```

Figure 24 : variante Six-point de l'algorithme Lane-Riesenfeld

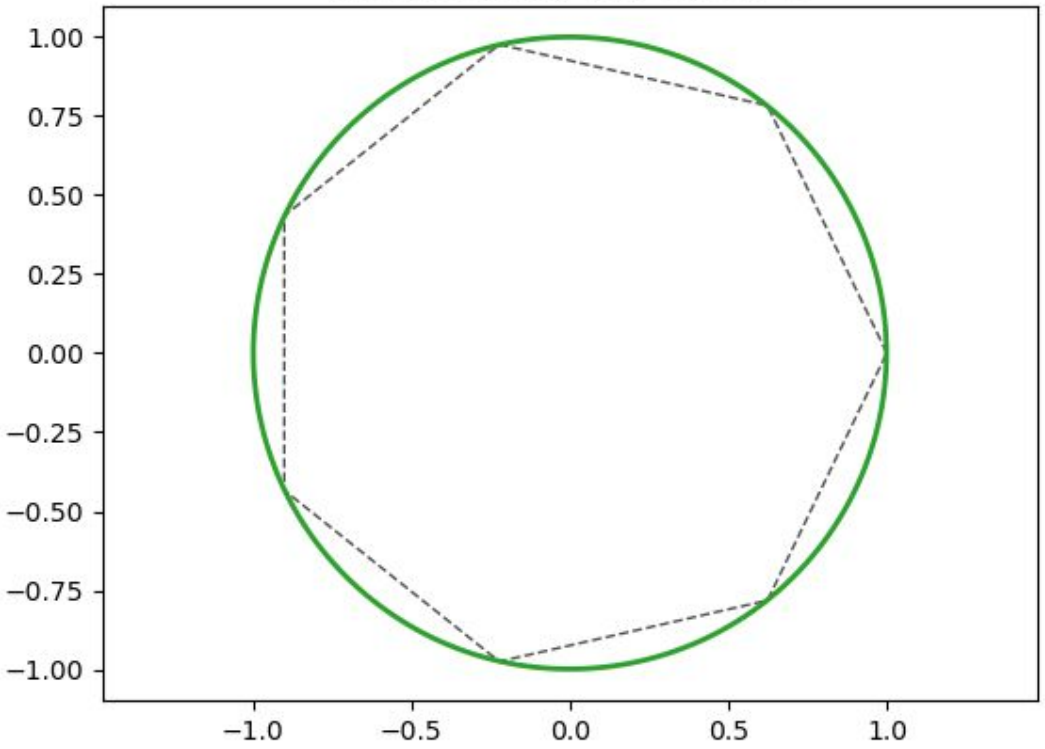


Figure 25 : “hepta” de degré 3, à 5 subdivisions avec la variante Six-point

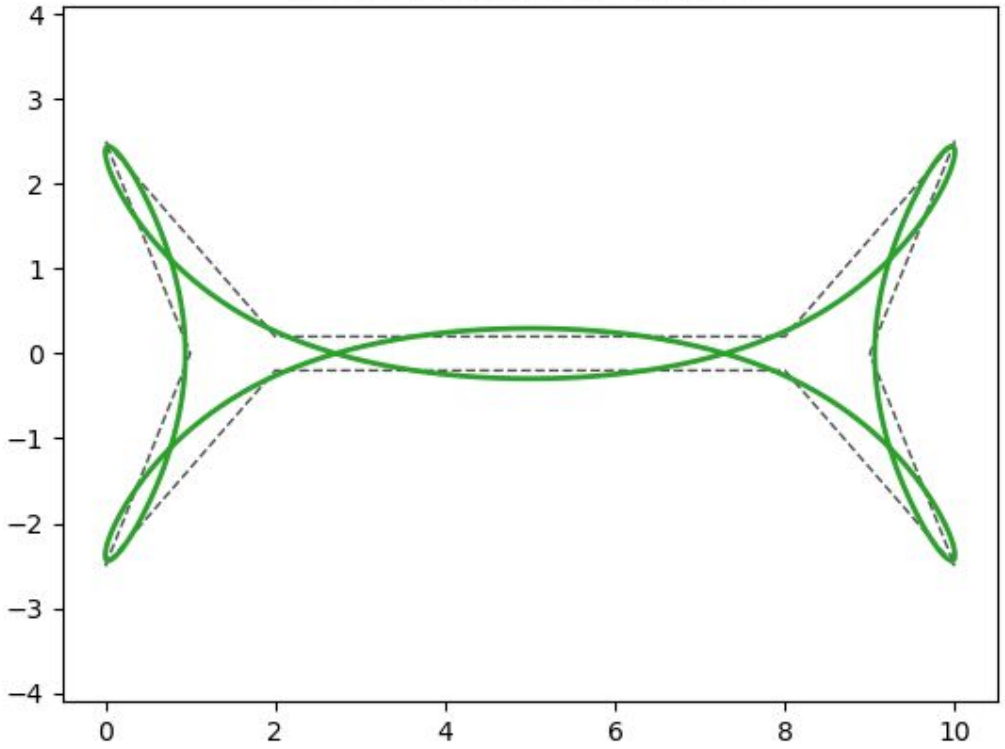


Figure 26 : “bone” de degré 3, à 5 subdivisions avec la variante Six-point

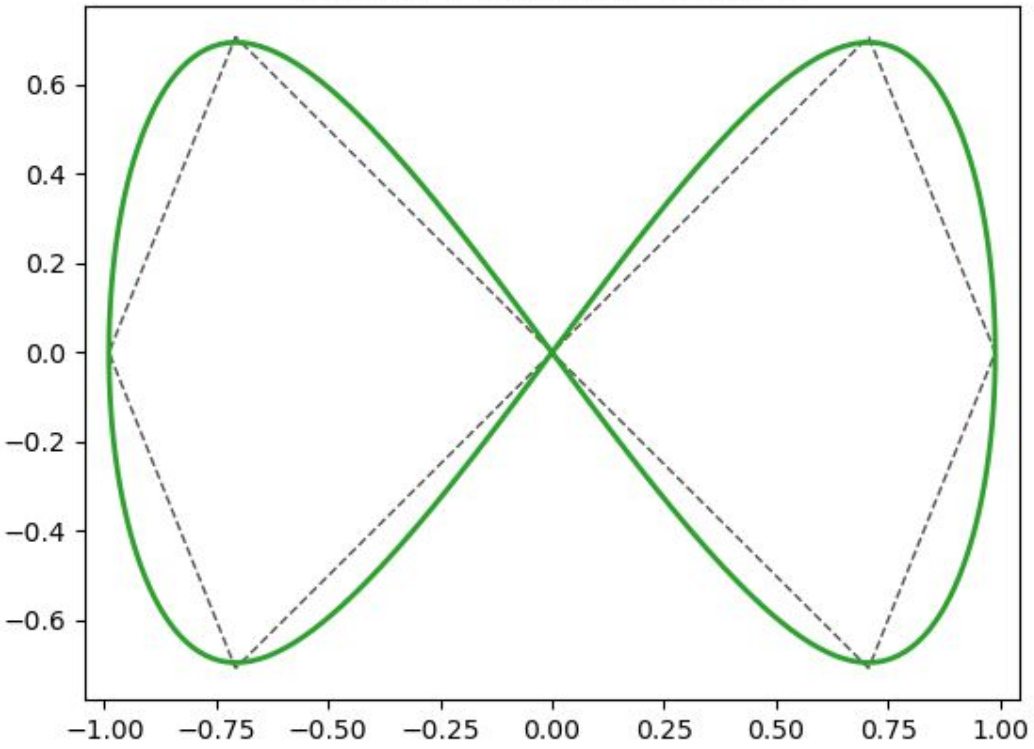


Figure 27 : “infinity” de degré 3, à 5 subdivisions avec la variante Six-point

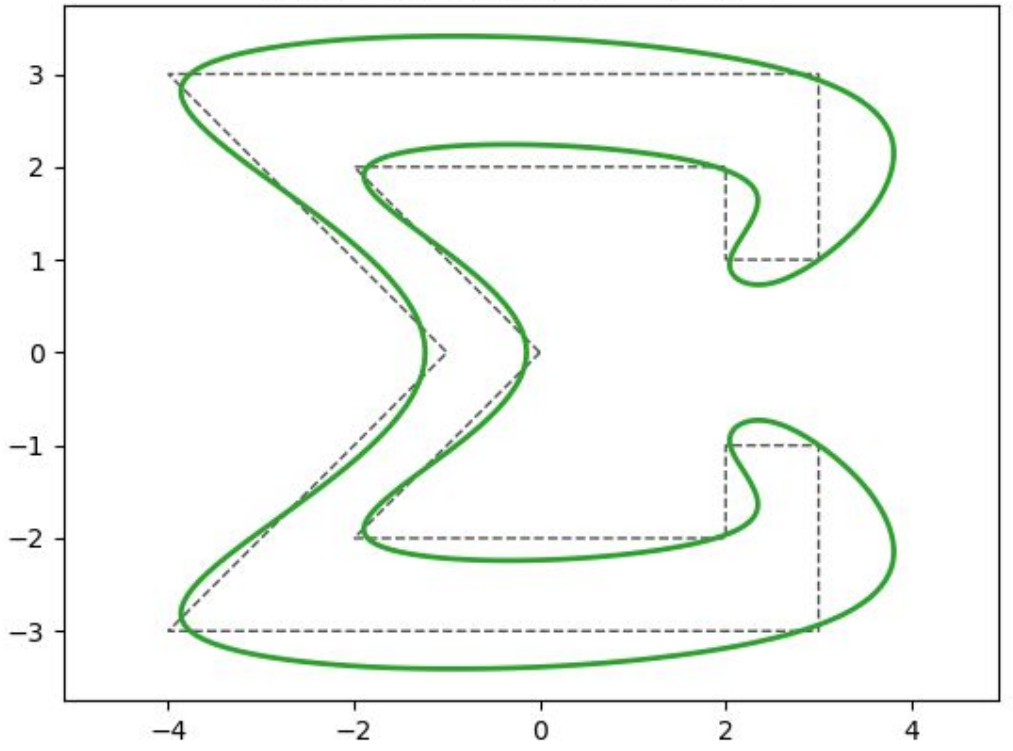


Figure 28 : “sumsign” de degré 3, à 5 subdivisions avec la variante Six-point

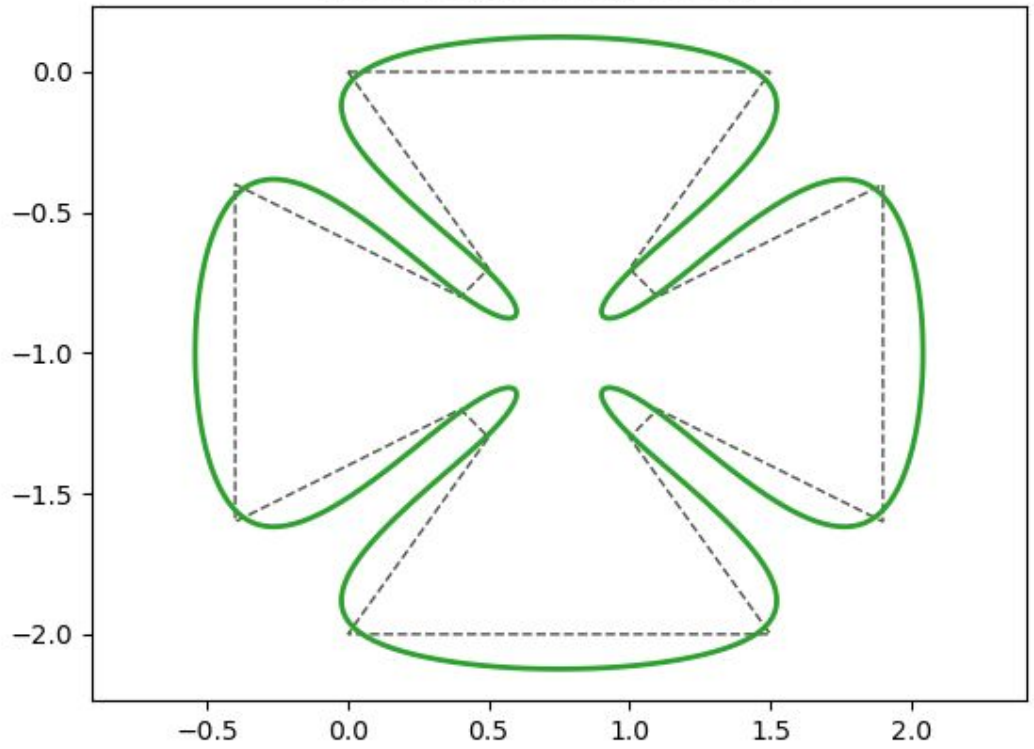


Figure 29 : “clover” de degré 3, à 5 subdivisions avec la variante Six-point

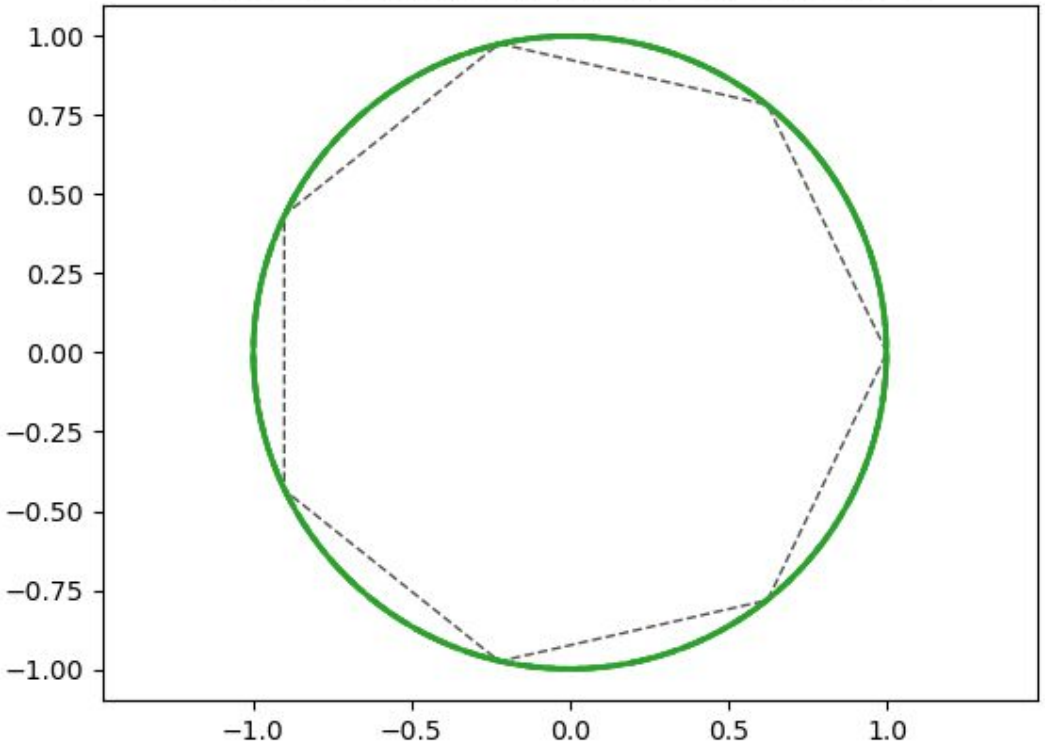


Figure 30 : “hepta” de degré 2, 3, 5, 10 et 30, à 5 subdivisions la variante Six-point

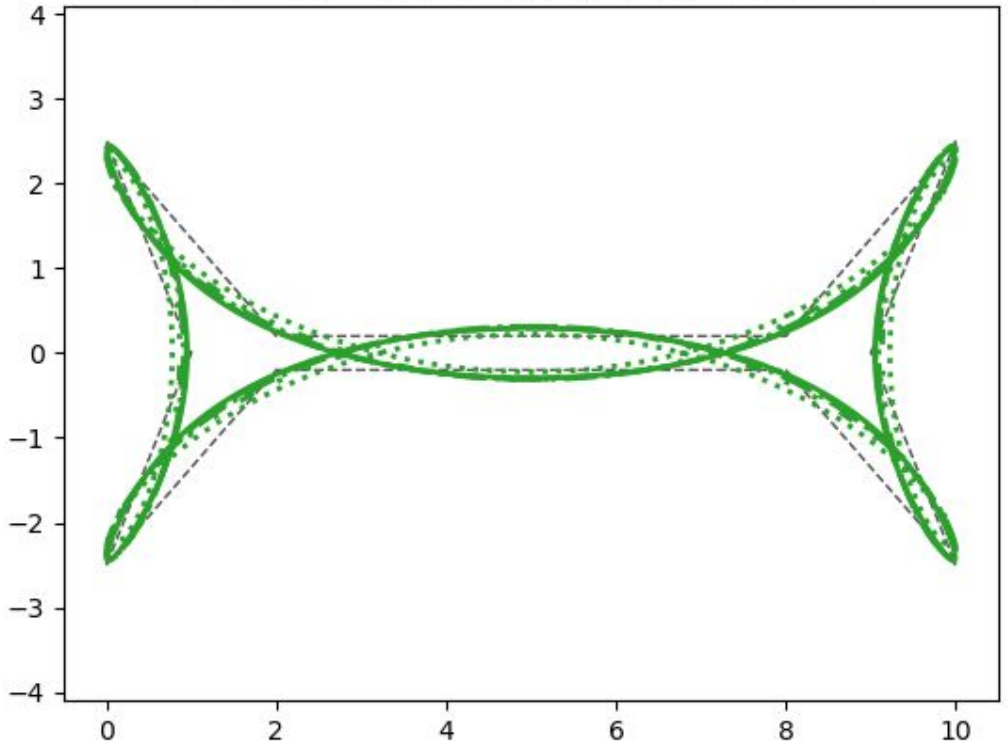


Figure 31 : “bone” de degré 2, 3, 5, 10 et 30, à 5 subdivisions la variante Six-point

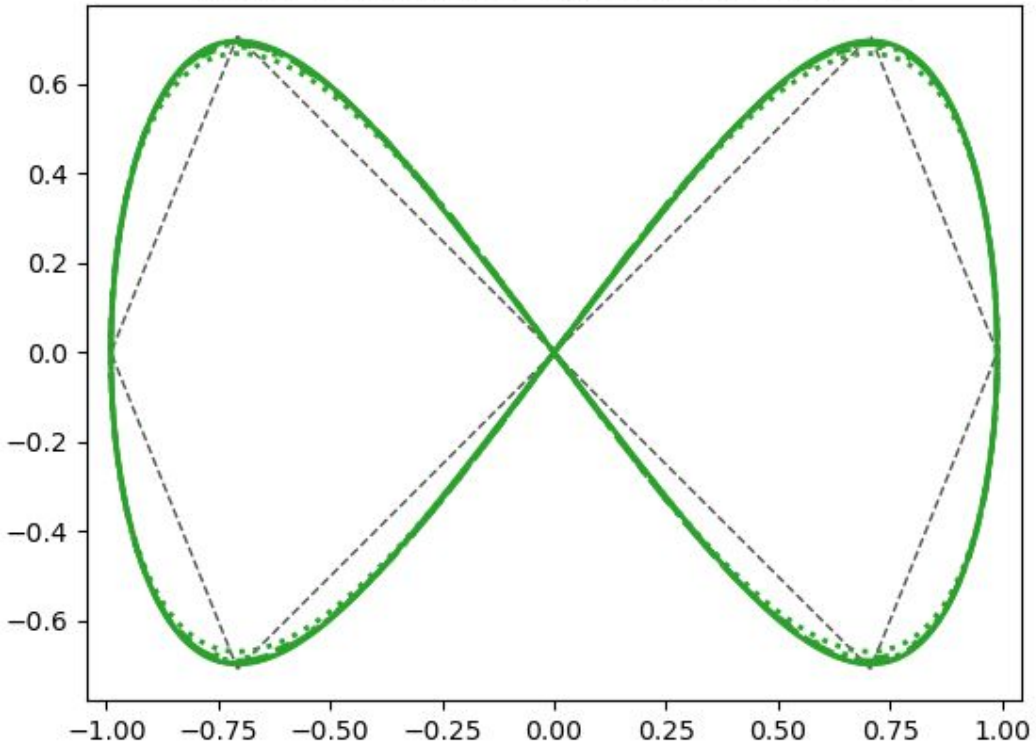


Figure 32 : “infinity” de degré 2, 3, 5, 10 et 30, à 5 subdivisions la variante Six-point

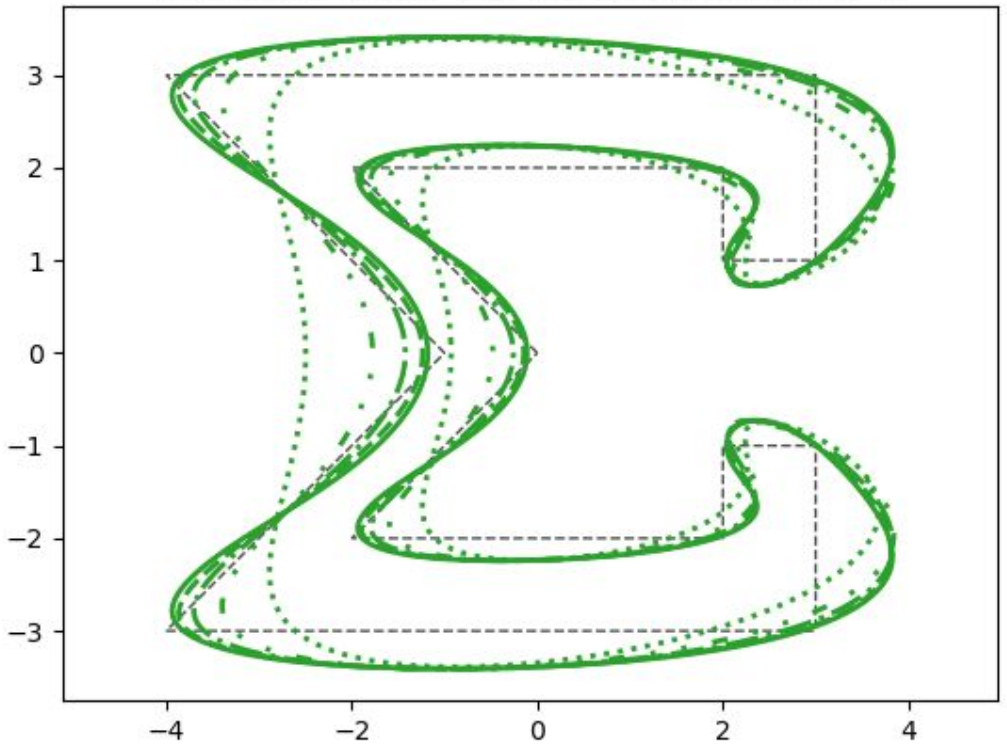


Figure 33 : “sumsign” de degré 2, 3, 5, 10 et 30, à 5 subdivisions la variante Six-point

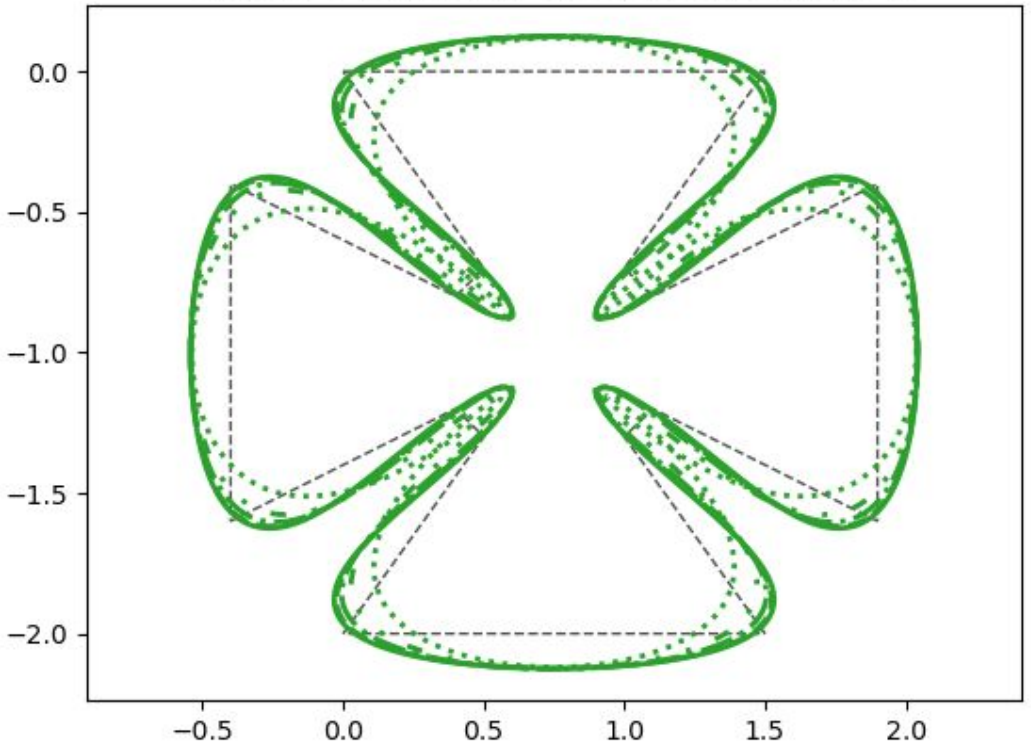


Figure 34 : “clover” de degré 2, 3, 5, 10 et 30, à 5 subdivisions la variante Six-point

Comparaison des trois algorithmes

L'algorithme à utiliser parmi les trois algorithmes vus dans cet exercice dépend grandement des attentes de l'utilisateur. Si ce dernier souhaite minutieusement manipuler la courbe, la variante *Six-point* sera la mieux adaptée car le changement de degré n'influence que légèrement la forme de la courbe. Mais s'il souhaite attribuer une plus grande flexibilité dans la forme de la courbe, l'algorithme de base Lane-Riesenfeld sera de meilleur choix. La variante *Four-point* pourra accessoirement servir d'entre-deux, malgré un grand rapprochement avec la variante *Six-point* dans les figures à géométrie simple tel que le heptagone de la figure 35.

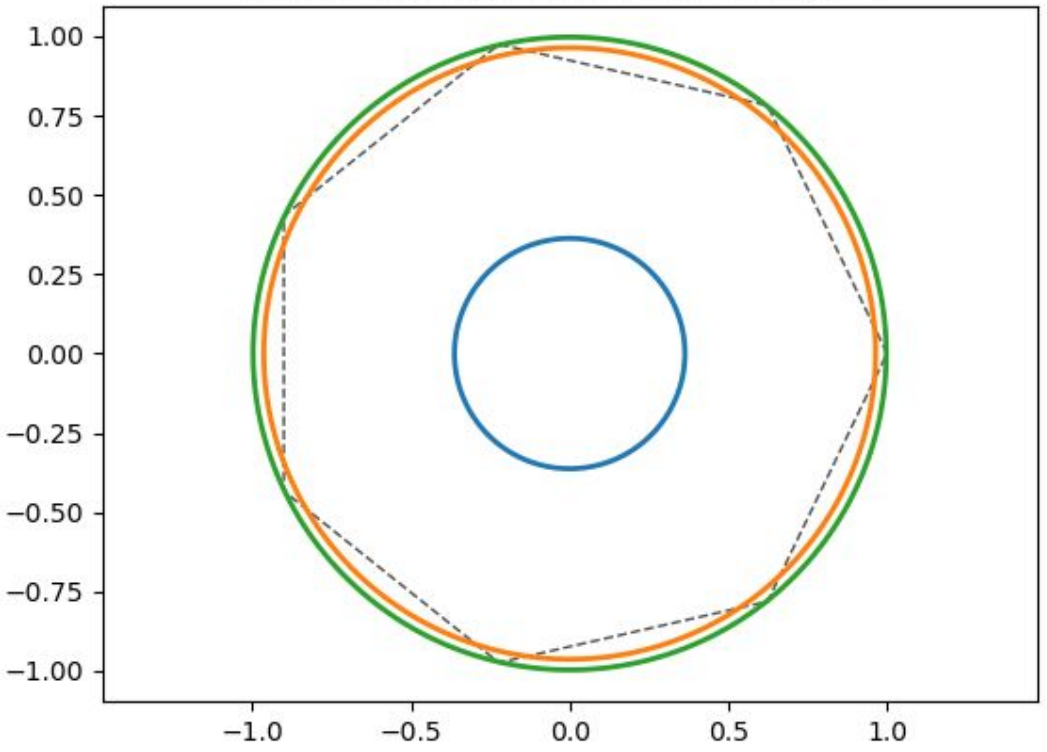


*Figure 35 : "hepta" de degré 30, à 5 subdivisions avec l'algorithme **Lane-Riesenfeld en bleu**, la variante **Four-point en orange** et la variante **Six-point en vert***

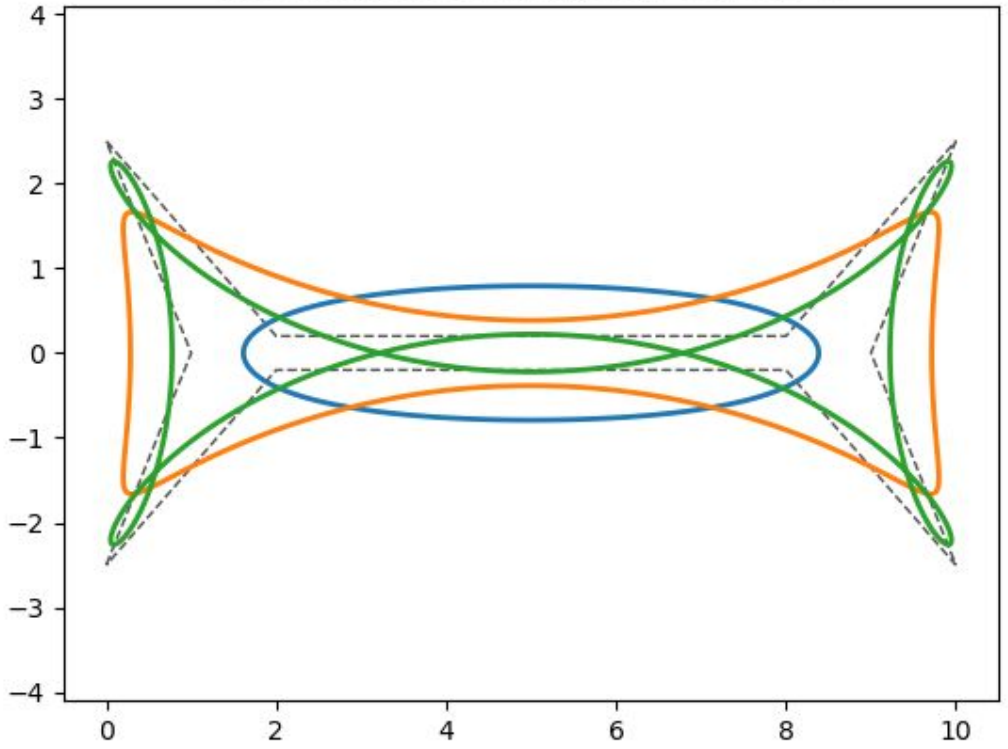


*Figure 36 : "bone" de degré 30, à 5 subdivisions avec l'algorithme **Lane-Riesenfeld en bleu**, la variante **Four-point en orange** et la variante **Six-point en vert***

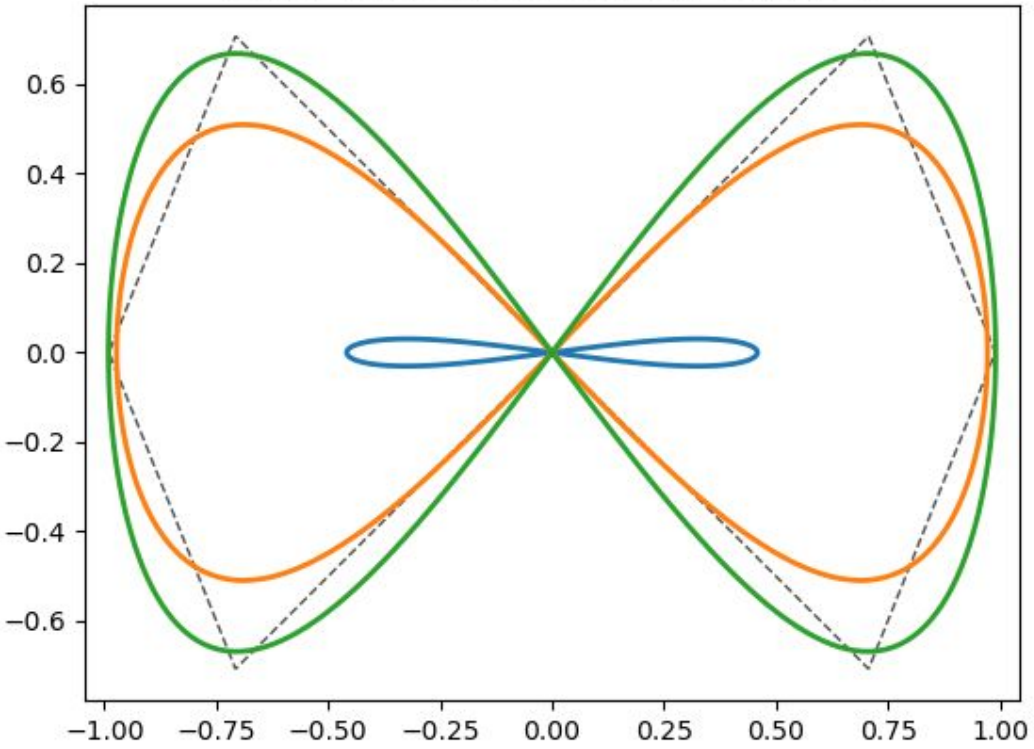


*Figure 37 : "infinity" de degré 30, à 5 subdivisions avec l'algorithme **Lane-Riesenfeld en bleu**, la variante **Four-point en orange** et la variante **Six-point en vert***

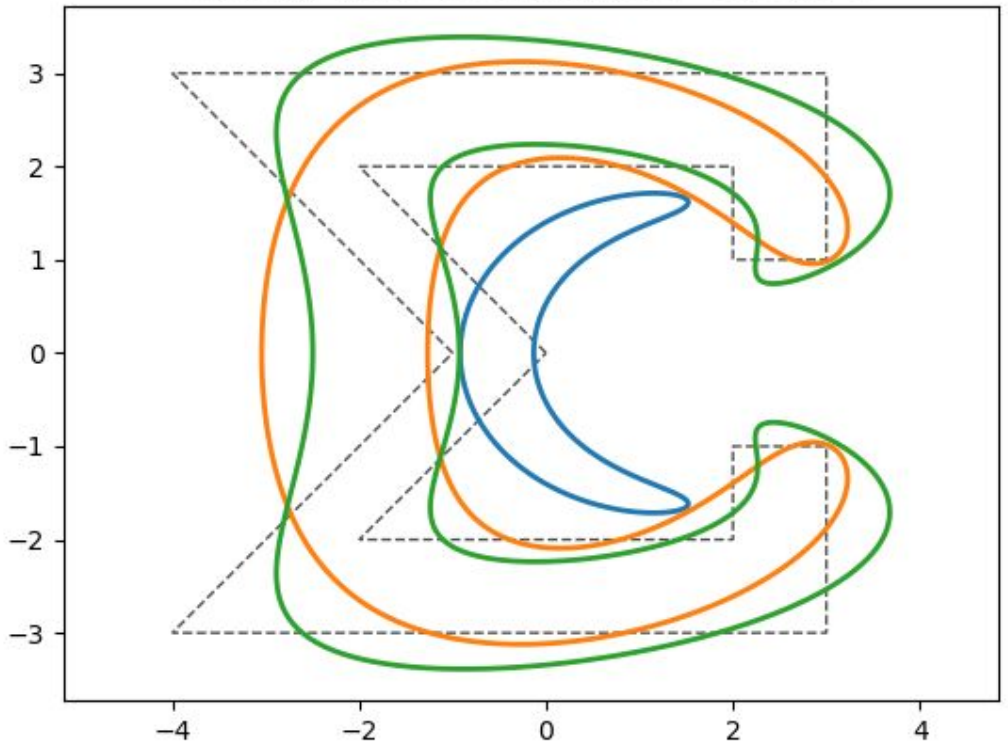


*Figure 38 : "sumsign" de degré 30, à 5 subdivisions avec l'algorithme **Lane-Riesenfeld en bleu**, la variante **Four-point en orange** et la variante **Six-point en vert***

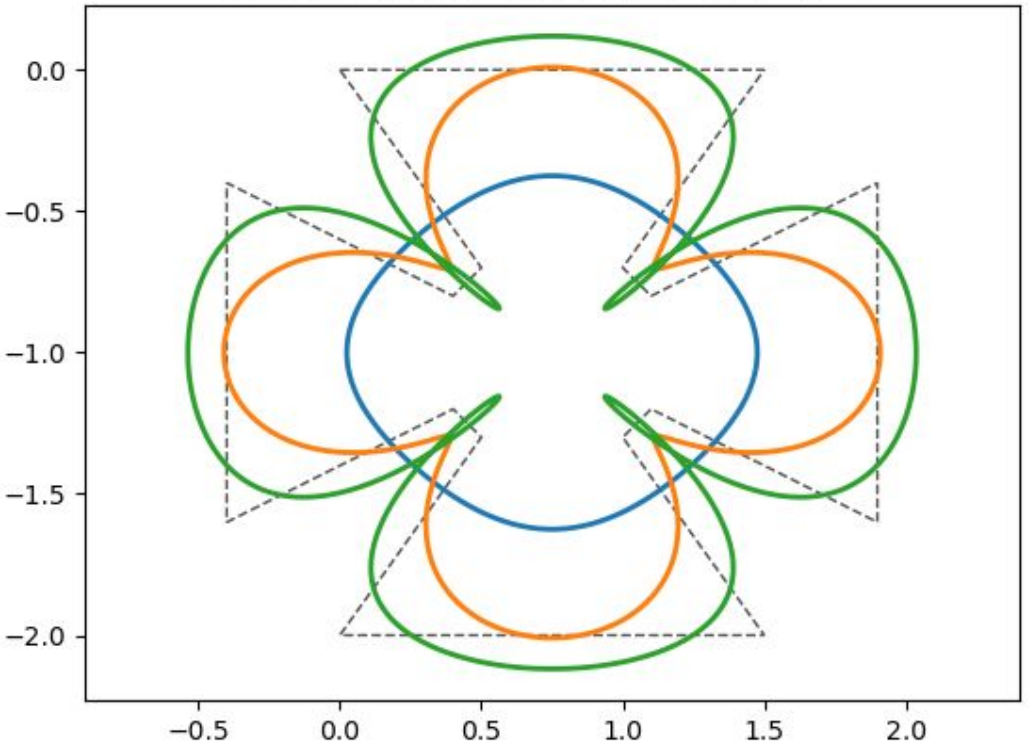


*Figure 39 : "clover" de degré 30, à 5 subdivisions avec l'algorithme **Lane-Riesenfeld en bleu**, la variante **Four-point en orange** et la variante **Six-point en vert***