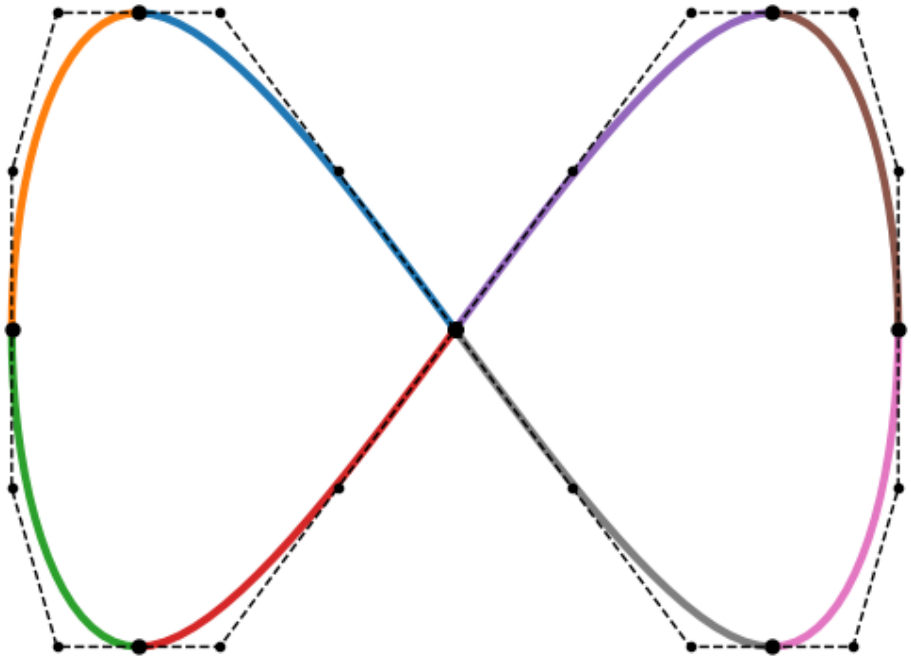
RAPPORT GÉOMÉTRIE NUMÉRIQUE

T.P. 2 - Bézier splines, C_k smoothness



Calvin Massonnet

08/02/2021 - 14/02/2021

Université Grenoble Alpes - Master Informatique

Spline quadratique C1

En utilisant des splines quadratiques C1, la courbe passe par le premier point et le dernier point des polygones de contrôle. Autrement dit, les polygones de contrôle étant composés de trois points, un point sur deux de l'ensemble des polygones de contrôle fait partie de la courbe. C'est la raison pour laquelle, dans la boucle de l'algorithme de la figure 1, un point sur deux est initialisé à la valeur du point reçu en entrée de la fonction et que l'autre moitié est calculée à partir du point suivant (dont on a les coordonnées) et le second point précédent (pour la continuité de la tangente). La fonction retourne un tableau de coordonnées de points de contrôle. La figure 2 présente les polygones de contrôle en pointillés.

```
def ComputeSplineC1(DataPts):
      n = DataPts.shape[0] - 1

      BezierPts = np.zeros([2 * n + 1, 2])

      BezierPts[0] = DataPts[0]
      BezierPts[1] = 0.5 * (DataPts[0] + DataPts[1])
      for i in range(2, len(BezierPts) - 1):
            if (i % 2 == 0):
                  BezierPts[i, :] = DataPts[i / 2, :]
            else:
                  BezierPts[i, :] = 2 * DataPts[i / 2] - BezierPts[i - 2]

      BezierPts[n * 2] = DataPts[n]

      return BezierPts
```

Figure 1 : algorithme de spline quadratique C1
(Note : un arrondi à l'entier inférieur est appliqué aux divisions)

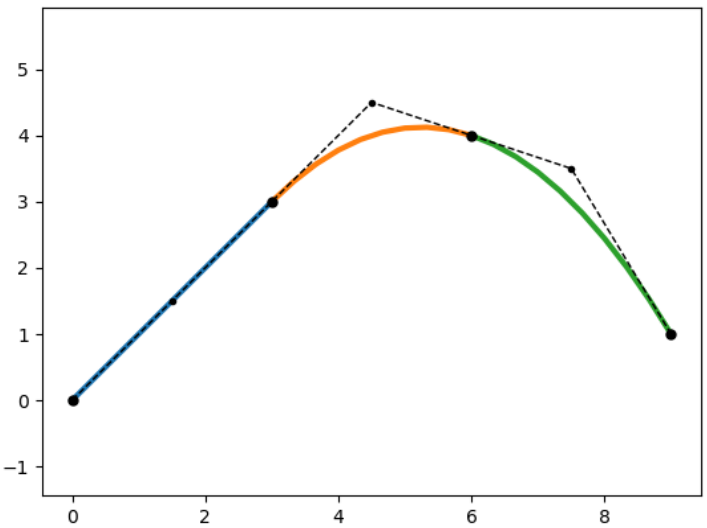


Figure 2 : "simple" de densité 10 avec splines quadratiques C1

Spline cubique C2

En utilisant des splines cubiques C2, la courbe passe par un point sur trois de l'ensemble des polygones de contrôle. Le code de l'algorithme de la figure 2 construit une matrice d'équations linéaires et en retourne les résultats qui seront les points des différents polygones de contrôle de la courbe. La construction de la matrice est décomposée en plusieurs parties : *C0*, *C1*, *C2* et *natural*. La figure 4 expose les polygones de contrôle en pointillés.

```
def ComputeSplineC2(DataPts):
      n = DataPts.shape[0] - 1
      M = np.zeros([3 * n + 1, 3 * n + 1])
      R = np.zeros([3 * n + 1, 2])

      # C0
      for i in range(n + 1):
            M[i][i * 3] = 1

      # C1
      for i in range(n + 1, n * 2):
            M[i][(i % n) * 3 - 1] = 1
            M[i][(i % n) * 3] = -2
            M[i][(i % n) * 3 + 1] = 1

      # C2
      for i in range(n * 2, (n * 3) - 1):
            M[i][(i % n + 1) * 3 - 2] = 1
            M[i][(i % n + 1) * 3 - 1] = -2
            M[i][(i % n + 1) * 3 + 1] = 2
            M[i][(i % n + 1) * 3 + 2] = -1

      # natural
      M[3 * n - 1][0] = 1
      M[3 * n - 1][1] = -2
      M[3 * n - 1][2] = 1
      M[3 * n][3 * n - 2] = 1
      M[3 * n][3 * n - 1] = -2
      M[3 * n][3 * n] = 1

      # Put DataPts to the first (n+1) rows of R.
      for i in range(n + 1):
            R[i] = DataPts[i]

      return np.linalg.solve(M, R)
```

Figure 3 : algorithme de spline cubique C2

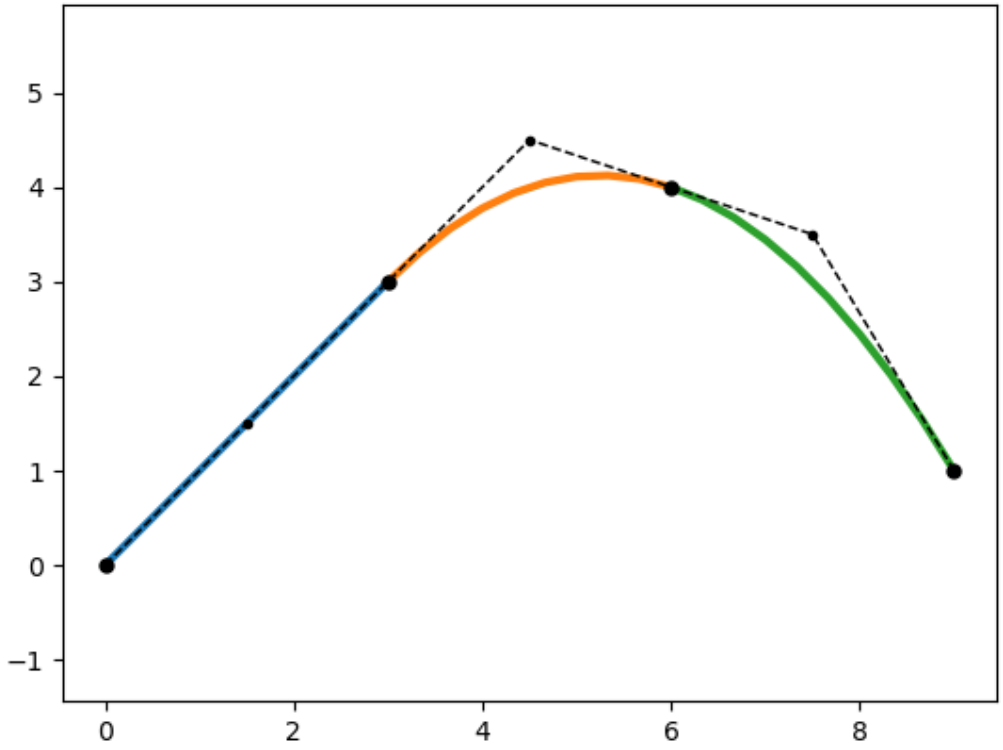


Figure 2 bis : “simple” de densité 10 avec splines quadratiques C1

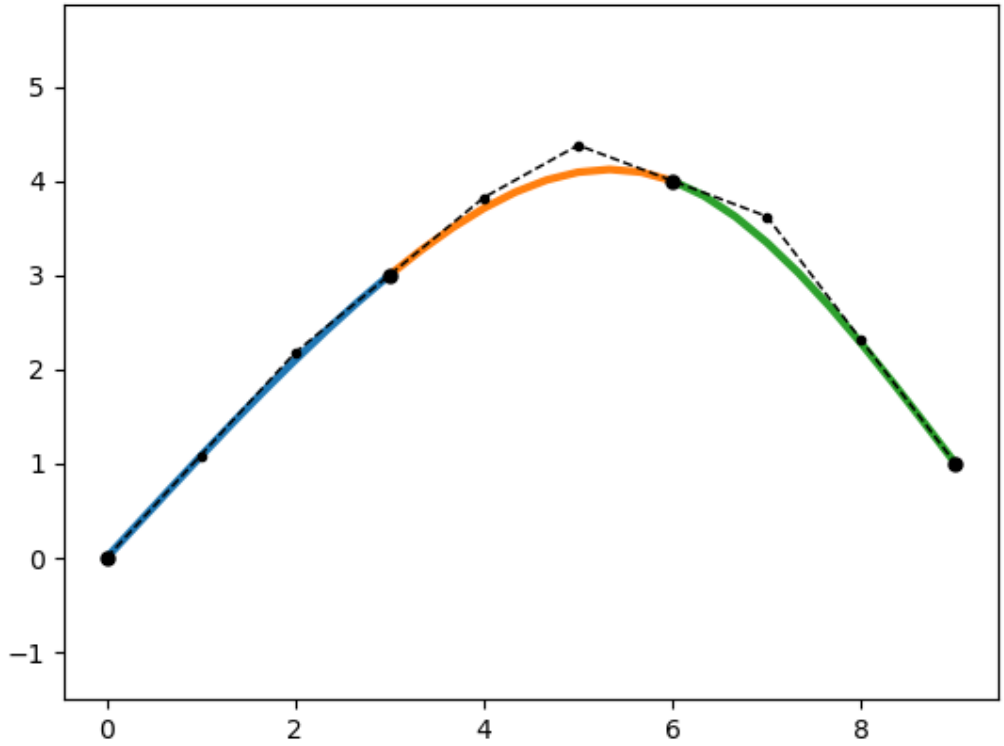


Figure 4 : “simple” de densité 10 avec splines cubiques C2

Comparaison des résultats

Dans certains cas, par souci d'alignement des tangentes des polygones de contrôle adjacents, le second algorithme (fig. 3) offre un avantage avec ses deux points de contrôle externes à la courbe. Il est possible de distinguer à l'aide des figures 5, 6, 7, 8, 9 et 10 que le second algorithme est à privilégier par-dessus le premier.

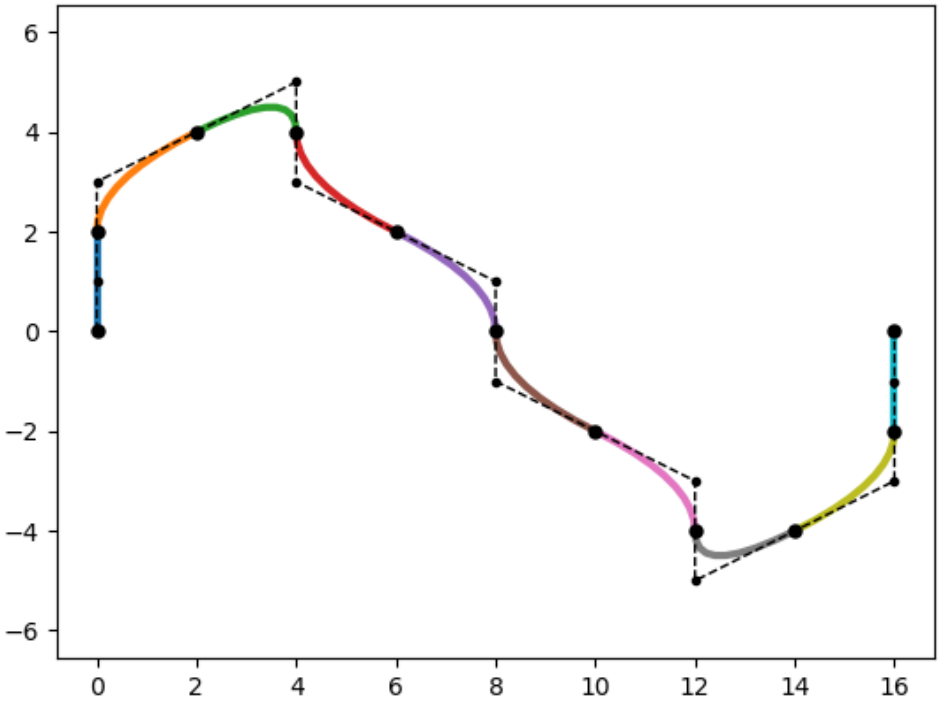


Figure 5 : "semi" de densité 10 avec splines quadratiques C1

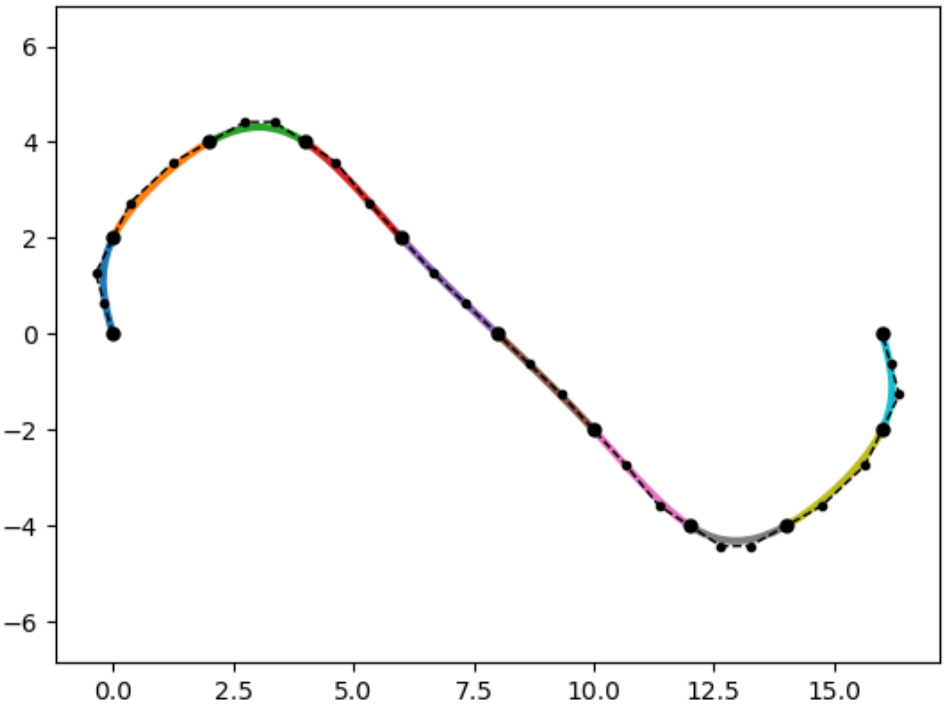


Figure 6 : "semi" de densité 10 avec splines cubiques C2

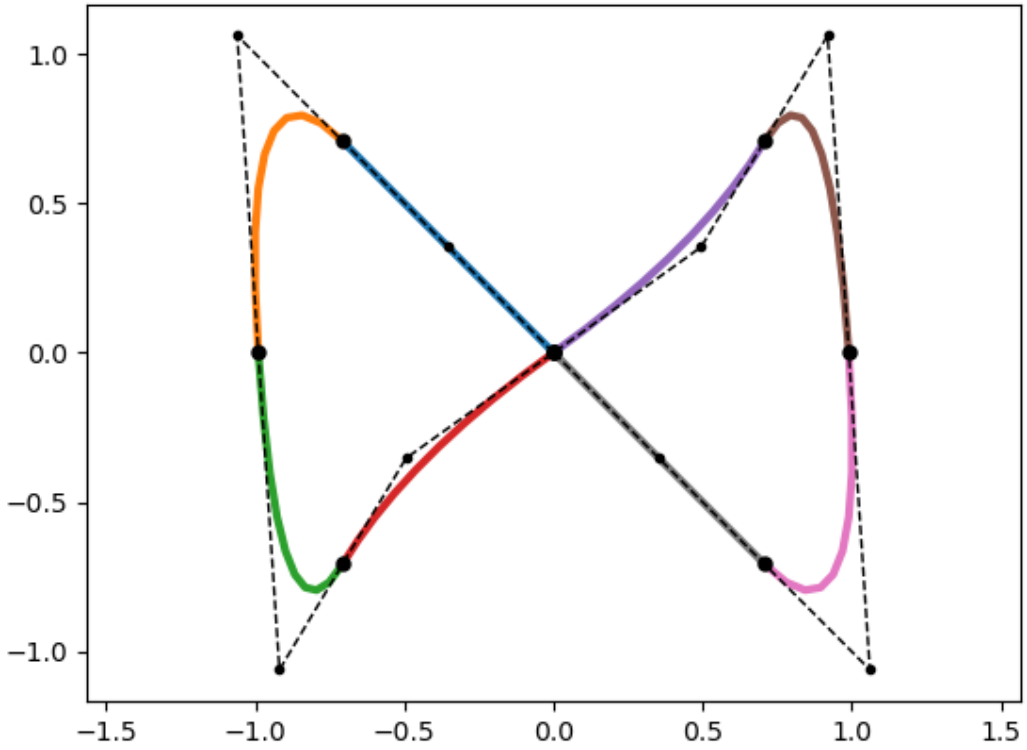


Figure 7 : “infinity” de densité 10 avec splines quadratiques C1

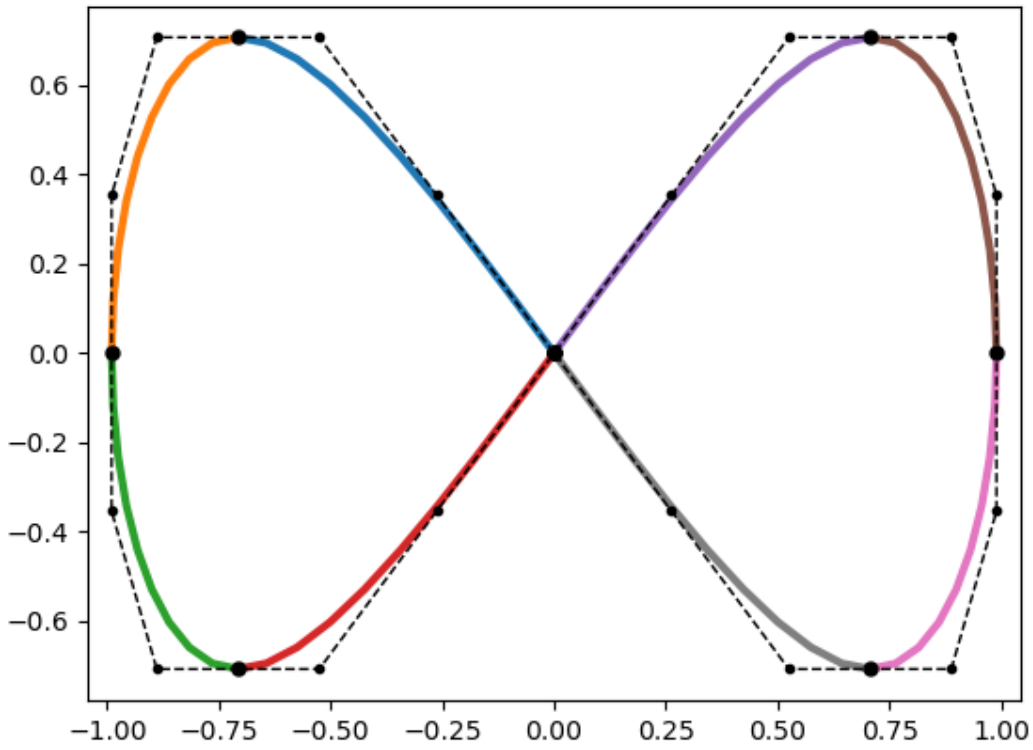


Figure 8 : “infinity” de densité 10 avec splines cubiques C2

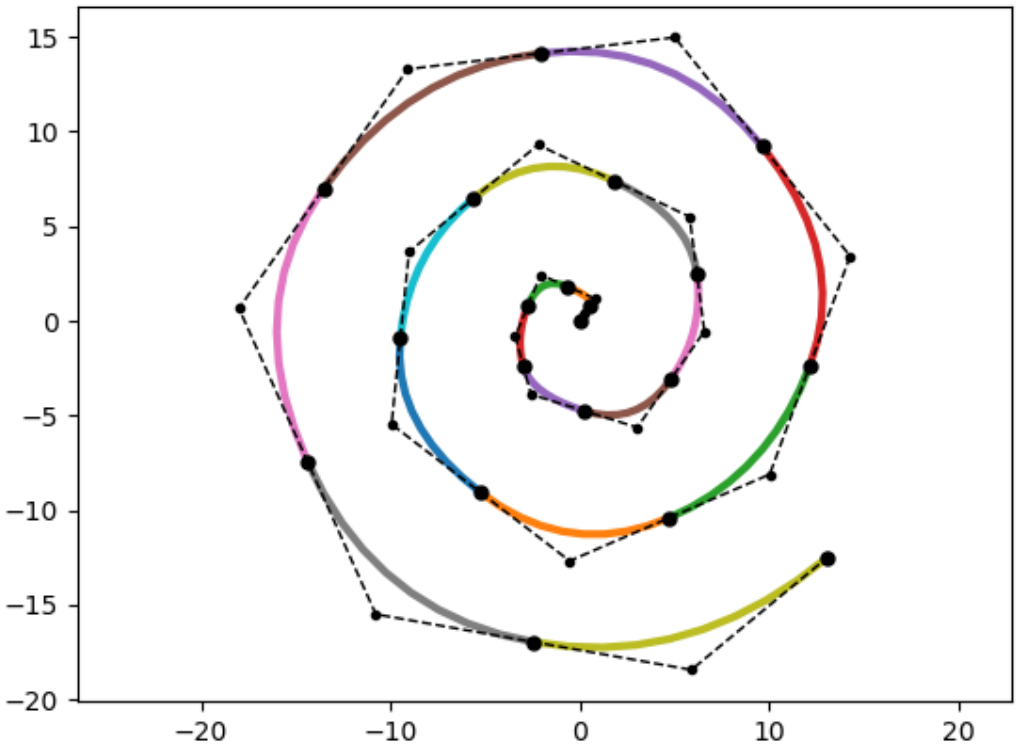


Figure 9 : “spiral” de densité 10 avec splines quadratiques C1

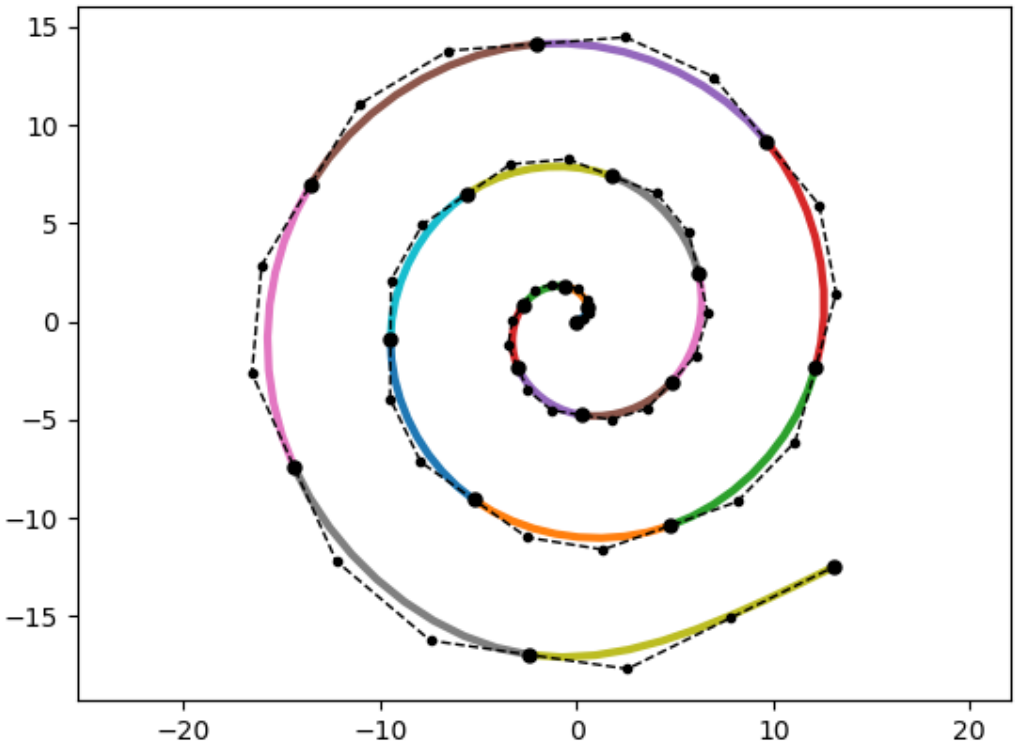


Figure 10 : “spiral” de densité 10 avec splines cubiques C2

Variation de densité

En augmentant légèrement la densité de la courbe, il est possible d'en lisser les virages et ainsi obtenir une courbe aux coudes plus arrondis, tels que présentés dans les figures 11, 12, 13 et 14. Une amélioration possible à apporter au code serait de donner une densité différente à chacun des segments de la courbes afin de réduire le nombre de points par lesquels passe la courbe lorsque possible.

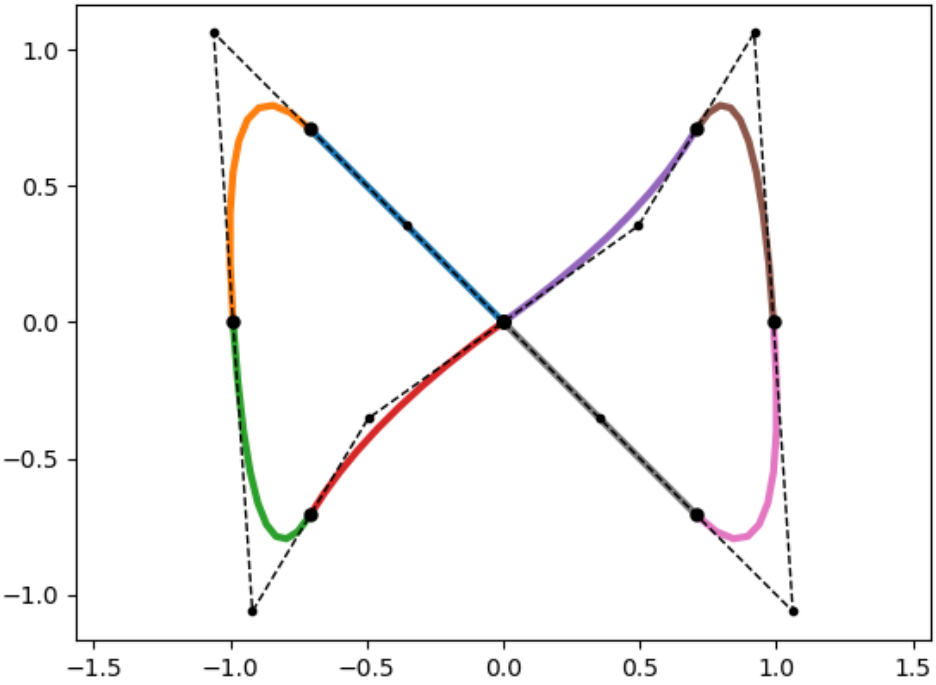
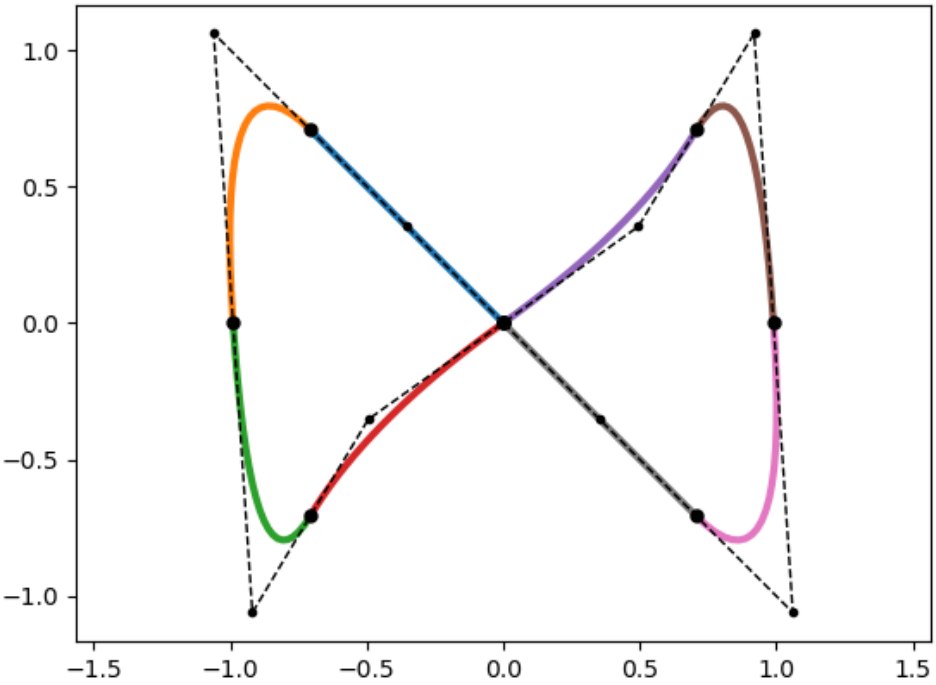


Figure 7 bis : "infinity" de densité 10 avec splines quadratiques C1



*<**TODO** - Figure 11 : "infinity" de densité 20 avec splines quadratiques C1>*

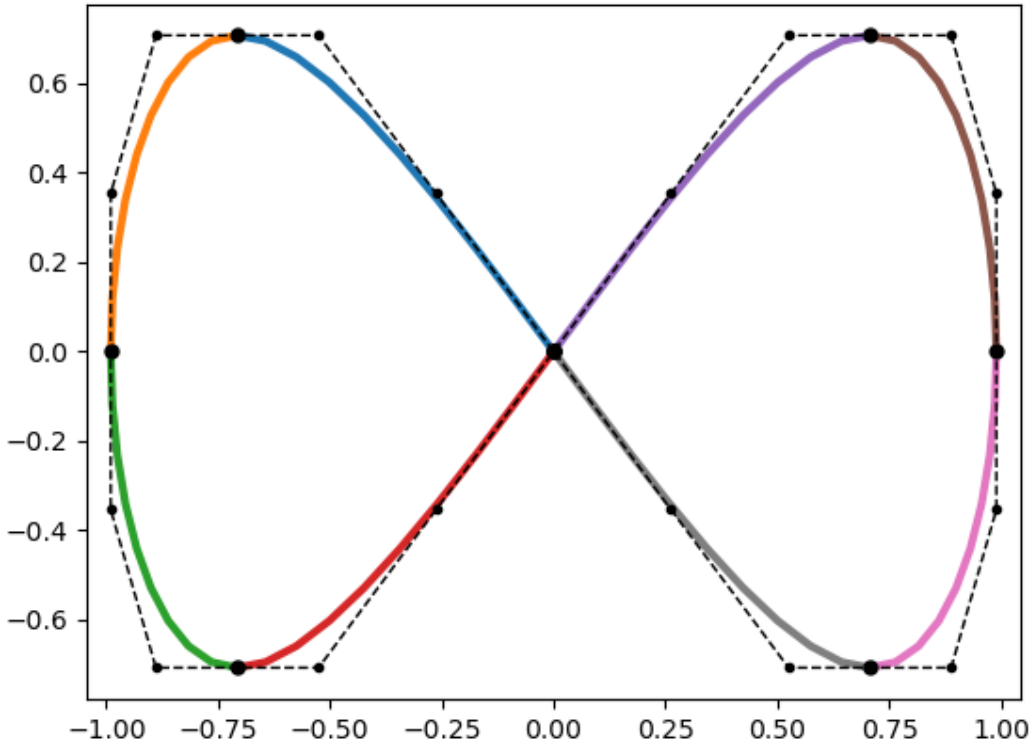


Figure 8 bis : “infinity” de densité 10 avec splines cubiques C2

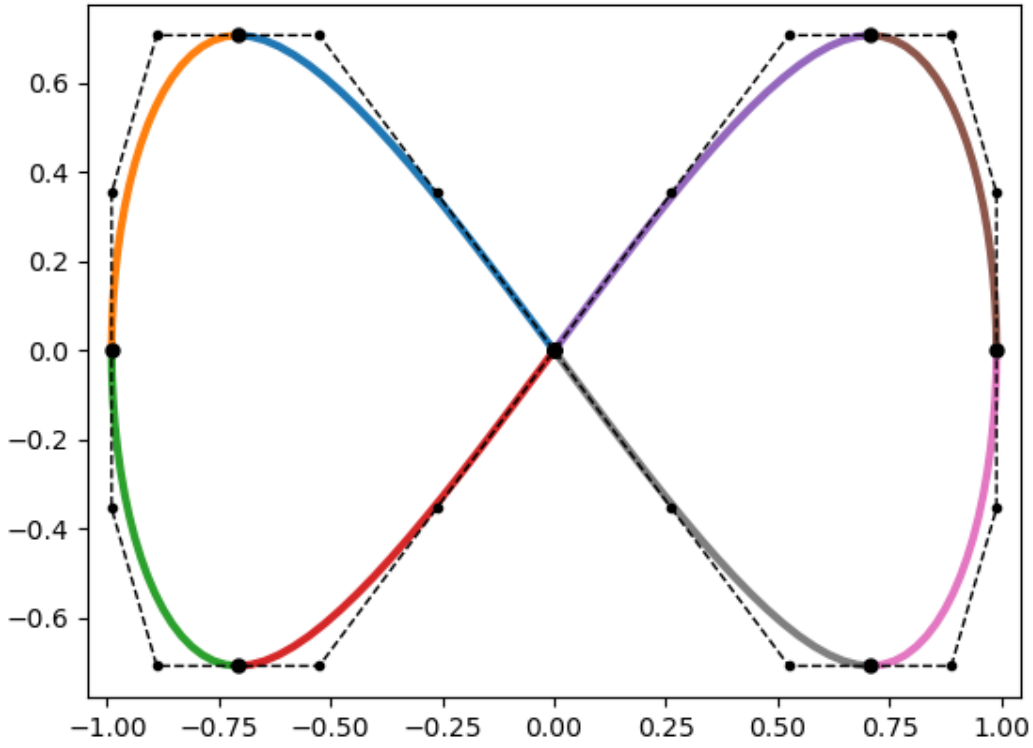


Figure 12 : “infinity” de densité 20 avec splines cubiques C2