2TSI. TD PYTHON 18

Courbes paramétrées

On se place dans le plan muni d'un repère orthonormé direct $(O, \vec{i}, \vec{j})$.
Soit pour $n \in \mathbb{N}$,

$$\left\{ \begin{array}{rcl} x_n(t) & = & 4\cos t - \cos(nt) \\ y_n(t) & = & 4\sin t - \sin(nt) \end{array} \right. \text{ et } \mathcal{C}_n = \{(x_n(t), y_n(t)),\ t \in [0, 2\pi]\}.$$

1. Tracer les courbes $\mathcal{C}_n$ pour n allant de 2 à 7.
2. À l'aide de $\mathcal{C}_n$, préciser les éventuelles symétries et les prouver par le calcul.
   On pourra commencer par remarquer que les courbes $\mathcal{G}_n = \{(x_n(t), y_n(t)),\ t \in \mathbb{R}\}$ sont identiques aux courbes $\mathcal{C}_n$.
3. Déterminer une valeur de n pour laquelle la courbe $\mathcal{C}_n$ admet des points qui ne sont pas réguliers ? Donner la valeur de t aux points non réguliers.
   À l'aide d'un développement limité, déterminer l'allure de l'arc à leur voisinage.
   Indication : si l'on a du mal à deviner, on peut utiliser le module scipy.optimize de Python pour résoudre des systèmes non linéaires à deux variables. En effet, les inconnues sont t et n car on résout

   $$\left\{ \begin{array}{rcl} x_n'(t) & = & 0 \\ y_n'(t) & = & 0 \end{array} \right.$$

   On peut ensuite poser le système et le résoudre mathématiquement. Pour les développements limités, on peut faire cela à la main ou utiliser le module sympy.
4. On note l_n la longueur de la courbe $\mathcal{C}_n$. Soit M_n le point de coordonnées (n, l_n).
   Représenter graphiquement la liste des points $(M_n)_{1 \leqslant n \leqslant 20}$.
   Indication : ici, on utilise le module scipy.integrate (d'alias integr) qui contient la fonction integr.quad.
5. Conjecturer un équivalent de l_n pour n au voisinage de $+\infty$.

Quelques commandes Python utiles.

• **Il faut se souvenir de la façon dont on trace des courbes paramétrées avec Python**. On définit d'abord la liste des valeurs données au paramètre puis on construit la liste des abscisses et des ordonnées correspondantes.
On effectue alors le tracé de $t \mapsto (x(t), y(t))$ pour $t \in [a, b]$ avec un pas h :
$>>>\ T = np.arange(a, b, h)\,;\ X = x(T)\,;\ Y = y(T)\,;\ plt.plot(X, Y)\,;\ plt.show()$
Remarque : on peut taper aussi $T = np.linspace(a, b, p)$ (où le pas h devient le nombre p de points).

• **Il faut se souvenir de la façon dont on résout un système non linéaire avec Python**. ***On utilise le sous-module*** *scipy.optimize* en tapant :
import scipy.optimize as resol
On cherche une solution $(x_0, ..., x_{n-1})$ d'un système de la forme :
$f1(x_0, ..., x_{n-1}) = 0,\ f2(x_0, ..., x_{n-1}) = 0,\ ...\ ,\ fn(x_0, ..., x_{n-1}) = 0.$
On utilise $resol.fsolve$ de premier argument $Syst$ et de second argument une liste $[a1, ..., an]$ constituée de valeurs initiales $x_0 = a1, ..., x_{n-1} = a_n$ « pas trop loin » des bonnes solutions. On fera plusieurs choix de ces valeurs initiales.
$>>>\ def\ Syst(X)\ :\ return(f1(X[0], ..., X[n-1]), ..., fn(X[0], ..., X[n-1]))$
$>>>\ Sol = resol.fsolve(Syst, [a1, ..., an])\,;\ print(Sol)$
Attention, le nombre d'équations doit être le même que le nombre d'inconnues.