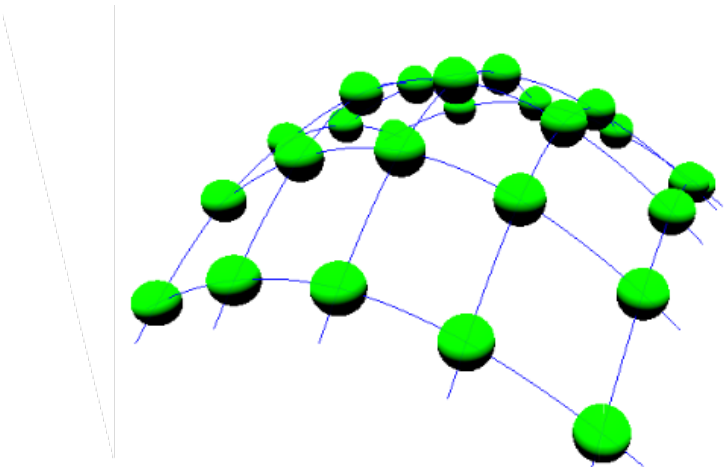
TP1 - Modélisation Géométrique

Interpolation & Approximation Polynomiale

Marie Pelissier, Géraldine Morin



Ce TP illustre la partie de cours sur l'interpolation et l'approximation polynomiale.

Table des matières

1 Préparation pour le TP1

Les sources de ce TP sont à récupérer sur la page moodle de la matière :

Département Sciences du Numérique >> 2ème année FISE >> Ensemble des UEs de S8 >>

N8EN06 - UE Modélisation Géométrique et EDP >> N8EN06A - Modélisation Géométrique

Le langage choisi pour ce TP est le Pyhton. Ce TP se présente sous la forme de notebooks et de fichiers pythons à compléter.

1.1 Prise en main du code

Nous vous conseillons utiliser le logiciel Visual Studio Code pour visualiser et compléter les codes fournis. Voici les étapes à suivre pour avoir un bon environnement de travail :

- Ouvrir le dossier associée au TP1 dans Visuel Studio Code, ce dernier contient 5 fichiers.
- Installer Python : choisir le logo Extensions présent sur la barre latéral de l'interface, puis rechercher Python. Une fois la page du langage python ouvert, cliquer sur *Install*.
- Ouvrir le premier notebook intitulé : *TP1_inperpolation.ipynb*
- Vous devez initialiser un environnement de travail, autrement dit, choisir un *Kernel*. Pour faire cette sélection, cliquez en haut à droite sur *Select Kernel*, puis *Python Environnements* et choisir la version 3.10.8 de python (ou ultérieure).

Ou sinon, laissez-vous guider par Visual Studio Code, tout simplement ... l'important est qu'il vous propose d'installer le package jupiter.

1.2 Descriptif du projet

Ces travaux pratiques reposent sur l'utilisation de deux notebooks, ***TP1_interpolation.ipynb*** et ***TP1_approximation.ipynb***, qui ont pour objectif d'appeler les fonctions d'interpolation ou d'approximation que vous devez rédiger, et d'afficher les résultats correspondants. À cet effet, un ensemble de points de contrôle est généré au départ et doit être conservé tout au long du notebook. La quantité de points de contrôle constitue une variable que vous avez la possibilité de modifier afin d'étudier la robustesse de vos algorithmes.

Votre tâche consiste uniquement à compléter les fichiers Python ***fonctions_tp1_interpolation.py*** et ***fonctions_tp1_approximation.py***.

Le fichier ***utils.py*** contient diverses méthodes auxiliaires pour l'affichage des résultats, la génération de points, etc.

Remarque : Lorsque vous modifiez les scripts ***fonctions_tp1_interpolation.py*** et ***fonctions_tp1_approximation.py***, il est recommandé de faire un *Restart* sur votre notebook avant de le relancer, pour changer les dernières modifications.

1.3 Travail à faire – Interpolation

1.3.1 Cas fonctionnel : Interpolation de points

Étant données $n+1$ points $P_i = ((x_i, y_i))_{i=0}^n$ du plan, dessiner la fonction polynomiale f de degré n telle que

$$f(x_i) = y_i, i = 0...n$$

Compléter le code de la fonction *lagrange* (fichier ***fonctions_tp1_interpolation.py***) afin de calculer l'ordonnée y du point d'abscisse x_{int} appartenant à la courbe qui interpole les points de coordonnées (XX,YY).

1.3.2 Cas paramétrique : Interpolation de points

Étant donnés $n+1$ points $P_i = ((x_i, y_i))_{i=0}^n$ du plan, et des paramètres (t_i) pour $i \in [\![0\ ;\ n]\!]$, dessiner la courbe paramétrique F polynomiale de degré minimum telle que $F(t_i) = (x_i, y_i)$ pour tout $i \in [\![0\ ;\ n]\!]$.

Compléter le code de la fonction ***parametrisation_reguliere*** qui doit construire à la fois le vecteur des temps TT d'évaluation et le vecteur d'échantillonnage de ces mêmes temps, qui sera défini par le nombre de points à calculer sur la courbe. Compléter également le code de la fonction ***lagrange _param***.

1.3.3 Influence des paramètres T_i

Pour mesurer l'influence du choix des T_i sur les courbes, on peut essayer de prendre des valeurs de T_i dépendantes de la distance entre les points, de la racine carrée de la distance entre les points, ou encore des valeurs de Tchebychev qui ont été vues en cours.

Ici il faut coder les méthodes :

- ***parametrisation_distance***
- ***parametrisation_racinedistance***
- ***parametrisation_Tchebycheff***

1.3.4 Interpolation paramétrique via l'algorithme de Neville

Dans cette partie, il faut programmer le calcul de l'interpolation par l'algorithme de Neville.

Ici, il faut coder les deux méthodes du fichier ***fonctions_tp1_interpolation.py*** : ***neville*** et ***neville_param***.

1.3.5 Surface interpolante en produit tensoriel

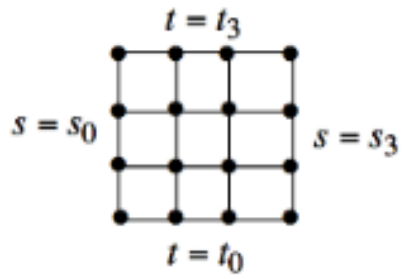
On rappelle qu'une surface en produit tensoriel est définie par :

$$S(s,t) = \sum_{i=0}^{n}\sum_{j=0}^{m} P_{ij} L_i(s) L_j(t) = \sum_{i=0}^{n} L_i(s) P_i(t)$$

avec $n+1$ points $P_i(t)$ pris sur $n+1$ courbes (à t fixé, pour $i \in [\![0\ ;\ n]\!]$) :

$$P_i(t) = \sum_{j=0}^{m} P_{ij} L_j(t)$$

En vous basant sur le script ***fonctions_tp1_interpolation.py*** que vous avez écrit et sur vos connaissances sur la méthode de *lagrange*, vous devez générez des surfaces interpolantes en produit tensoriel sur une grille de points de contrôle régulière (c'est à dire, les points de contrôles dans une matrice). Complétez la fonction ***interpolate_surface*** du fichier ***fonctions_tp1_interpolation.py***.



$$\begin{matrix} P_{03} & P_{13} & P_{23} & P_{33} \\ P_{02} & P_{12} & P_{22} & P_{32} \\ P_{01} & P_{11} & P_{21} & P_{31} \\ P_{00} & P_{10} & P_{20} & P_{30} \end{matrix}$$

Figure 1 – Domaine de paramétrisation et grille 4*4 de points de contrôle 3D pour une surface de bi-degré(n, m) = (3,3)

1.4 Travail à faire – Approximation

Préliminaire : Pour construire nos courbes approximantes *P(t)*, définies par un ensemble de points de contrôle, nous devons effectuer un certain nombre d'évaluation, autrement dit, évaluer *P(t)* pour plusieurs valeurs de *t*.
Écrire la fonction ***echantillonnage***, qui à partir d'un nombre d'échantillons à évaluer renvoie les temps d'évaluation. Nous considérons que le support de définition de nos courbes est $[0, 1]$.

1.4.1 La base de Bernstein

Tracer les n fonctions de la base de Bernstein d'un degré n choisi. Que se passe-t-il quand on augmente le degré n ? Compléter le code de la fonction ***build_polys_bernstein*** qui utilise à la fonction ***k_parmi_n***.

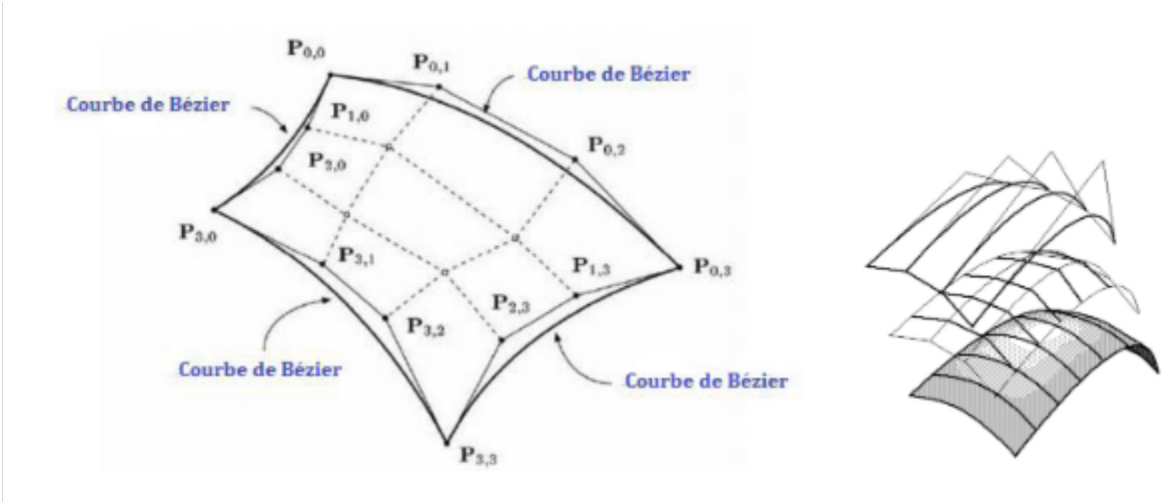
1.4.2 Algorithme de De Casteljau - évaluation

Étant donnés $n + 1$ points $P_i = \{(x_i, y_i)\}_{i=0}^{n}$ du plan, dessiner la courbe de Bézier *P(t)* correspondante et son polygone de contrôle en utilisant l'algorithme de De Casteljau pour l'évaluation. Que se passe-t-il quand le degré de la courbe augmente ? (faire le lien avec la question précédente). Compléter le code de la fonction ***DeCasteljau*** (fichier ***fonctions_tp1_approximation.py***).

1.4.3 Algorithme de de Casteljau - subdivision

Étant donnés $n+1$ points $P_i = \{(x_i, y_i)\}_{i=0}^{n}$ du plan, dessinez le polygone de contrôle décrivant la courbe approximante après n subdivisions. Vous devez écrire une première fonction ***subdivision*** qui réalise une étape de subdivision puis implémenter l'appel récursif ***DeCasteljauSub*** pour réaliser l'ensemble des subdivisions souhaitées.

1.4.4 Surfaces de Bézier : produit tensoriel



Cas du produit tensoriel On considère une surface paramétrique S :

$$S : \mathcal{R}^2 \to \mathcal{R}^3, (u, v) \to S(u, v)$$

$$S(u, v) = \sum_{j=0}^{n} \sum_{i=0}^{m} P_{ij} B_i^m(u) B_j^n(v) = \sum_{j=0}^{m} B_j^n(v) P_j(u)$$

On traite des courbes de de contrôle $P_j(u)$ plutôt que de points de contrôle P_j. Complétez la méthode intitulée ***Decasteljau_surface***.

1.4.5 Bonus : Hodographe (bonus)

On appelle hodographe d'une courbe de Bézier de degré n, la courbe de Bézier de degré $n-1$ représentant la dérivée de la courbe. Dessinez la courbe hodographe d'une courbe de Bézier, c'est à dire dont les vecteurs de base sont les vecteurs $P_{i+1} - P_i$. Peut-on lier la régularité de la paramétrisation d'une courbe avec son hodographe ?