bsplyne

Guide d'exploration et d'appropriation de la librairie

Dorian BICHET

29 Janvier 2026

Résumé

Ce document constitue un support pour la prise en main de la librairie `bsplyne`. Il détaille l'architecture basée sur l'algorithme de Cox-de Boor, les modalités d'installation et propose une progression pédagogique à travers les exemples du tutoriel.

1 Objectifs

- ✓ **Navigation :** Maîtriser l'architecture logicielle et l'API de documentation.
- ✓ **Exploration :** Comprendre les objets centraux par la modification des exemples.
- ✓ **Efficacité :** Utiliser l'IA (LLM) comme levier de guidage technique.

2 Environnement et installation

1. Configuration de l'environnement (Conda)

L'utilisation d'un environnement virtuel est recommandée pour isoler les dépendances (`Numba`, `SciPy`, `Matplotlib`) :

```
conda create -n bsplyne_env python=3.9
conda activate bsplyne_env
```

2. Installation en mode "Éditable"

Installer la librairie en mode `-e` permet de lier directement l'environnement au code source du dossier `bsplyne/` :

```
git clone https://github.com/Dorian210/bsplyne
cd bsplyne
pip install -e .
pip install pyvista  # Hautement recommande pour l'interactivite 3D
```

3 Architecture et documentation

3.1 Hiérarchie logicielle : Structure en couches

L'architecture de `bsplyne` suit strictement la construction mathématique des splines :

1. `b_spline_basis.py` : Implémente la récursion de Cox-de Boor pour l'évaluation des fonctions de base univariées $N_{i,p}(u)$. C'est la couche "bas niveau".
2. `b_spline.py` : Définit la classe `BSpline` comme une surcouche gérant une liste de `BSplineBasis`. Elle permet d'étendre les méthodes aux bases multivariées par produit tensoriel. Le mapping $S(\mathbf{u}) = \sum R_i(\mathbf{u})\mathbf{P}_i$ utilise des points de contrôle structurés en tenseurs d'ordre $n+1$ de forme `[dim, `n_ξ`, `n_η`, `n_ζ`, ...]`.
3. `multi_patch_b_spline.py` : Gère les assemblages complexes de plusieurs patchs et assure la connectivité conforme entre les domaines.

3.2 Performance et Visualisation

- **Aiguillage de `plot()` :** La méthode `plot()` est intelligente. Elle appelle automatiquement `plotPV()` (moteur PyVista) pour une interactivité 3D fluide, ou bascule sur `plotMPL()` (moteur Matplotlib) pour des tracés 2D ou si PyVista n'est pas installé.

- **Performance** : L'accélération via le décorateur `@njit` (Numba) est gérée en sous-couche. L'utilisateur interagit avec les méthodes de classe standards qui sollicitent de manière transparente ces fonctions compilées pour un calcul compilé accéléré.

Levier IA et Documentation

La documentation complète est disponible via `docs/index.html` ou le fichier `doc.pdf`.
Conseil : Vous pouvez fournir le `doc.pdf` à une IA (ChatGPT, Claude, Gemini). Elle pourra expliquer les signatures complexes ou générer des exemples d'utilisation rapides à tester. D'expérience, la documentation es suffisamment fournie pour permettre aux modèles IA de fournir un code plutôt satisfaisant.

4 Exploitation des exemples et manipulation

L'appropriation de la librairie repose sur une démarche de modification progressive des scripts du dossier `exemples/tutorial/`. Chaque étape propose un défi technique pour valider la compréhension des concepts.

4.1 Étape 1 : Fondations (`01_basis_functions.py`)

Approche "bas niveau". On manipule directement l'objet `BSplineBasis` pour comprendre l'espace paramétrique univarié sans l'abstraction des points de contrôle.

— **Concept :** Construction manuelle du mapping géométrique via le produit matriciel $y = \mathbf{NP}$ pour lier les fonctions de base aux coordonnées spatiales.

— **Challenge :** Créez une base de degré 1 à un seul élément, raffinez-la en degré 3 à 4 éléments, puis générez manuellement un jeu de points de contrôle pour visualiser le résultat. *Optionnel : Observez que les opérations d'insertion de nœuds et d'élévation d'ordre ne commutent pas.*

4.2 Étape 2 : L'abstraction `BSpline` (`02_curve_construction.py`)

Présentation de la classe `BSpline`, une surcouche simplifiant la gestion conjointe des bases multivariées et de la géométrie associée.

— **Concept :** Utilisation de tenseurs de points de contrôle de forme `[dim,` n_ξ`]`. Cette structure permet un accès intuitif aux données sans nécessiter d'aplatissement manuel (*flattening*).

— **Challenge :** Créez un objet `BSpline` de degré 1. Raffinez-le en insérant des nœuds **puis** en augmentant le degré. Appliquez un bruit aléatoire sur les points de contrôle et affichez la courbe résultante. *Optionnel : Affichez la base de fonctions à chaque étape.*

4.3 Étape 3 : Passage aux surfaces (`03_surface_wave.py`)

Extension de la logique tensorielle aux bases bivariées pour la génération de surfaces.

— **Concept :** Manipulation de tenseurs `[dim,` n_ξ`,` n_η`]`. L'interface `plot()` aiguille automatiquement vers l'interactivité 3D (PyVista) ou 2D (Matplotlib).

— **Challenge :** Créez une surface plane de degré 2. Appliquez une élévation de degré uniquement selon ξ et une insertion de nœuds uniquement selon η. Déformez ensuite les points de contrôle pour obtenir une surface en "selle de cheval" (appliquez $z = x^2 - y^2$ aux points de contrôle par exemple) et visualisez le résultat en 3D.

4.4 Étape 4 : Opérateurs et Moindres Carrés (`04_least_squares.py`)

Utilisation de la librairie comme outil de résolution numérique pour le fitting de formes.

— **Concept :** Extraction des opérateurs matriciels $\mathbf{N}$ via `spline.DN(XI)` pour résoudre des systèmes linéaires de type $\mathbf{NP} = \mathbf{X}_{target}$.

— **Challenge :** En vous basant sur l'exemple du disque, projetez la fonction "cloche" suivante sur une B-spline bivariée : $f(\xi, \eta) = \frac{8}{\pi} e^{-8\left[(\xi-\frac{1}{2})^2+(\eta-\frac{1}{2})^2\right]}$. Ici, la dimension physique est de 1 (champ scalaire d'élévation). Comparez visuellement le résidu en doublant le nombre de nœuds puis en élevant le degré de la base.

4.5 Étape 5 : Topologie Multi-Patch (`05_multipatch.py`)

Gestion d'assemblages complexes via `MultiPatchBSplineConnectivity`.

— **Concept :** Distinction entre points de contrôle "uniques" (topologie globale) et "séparés" (données locales par patch).

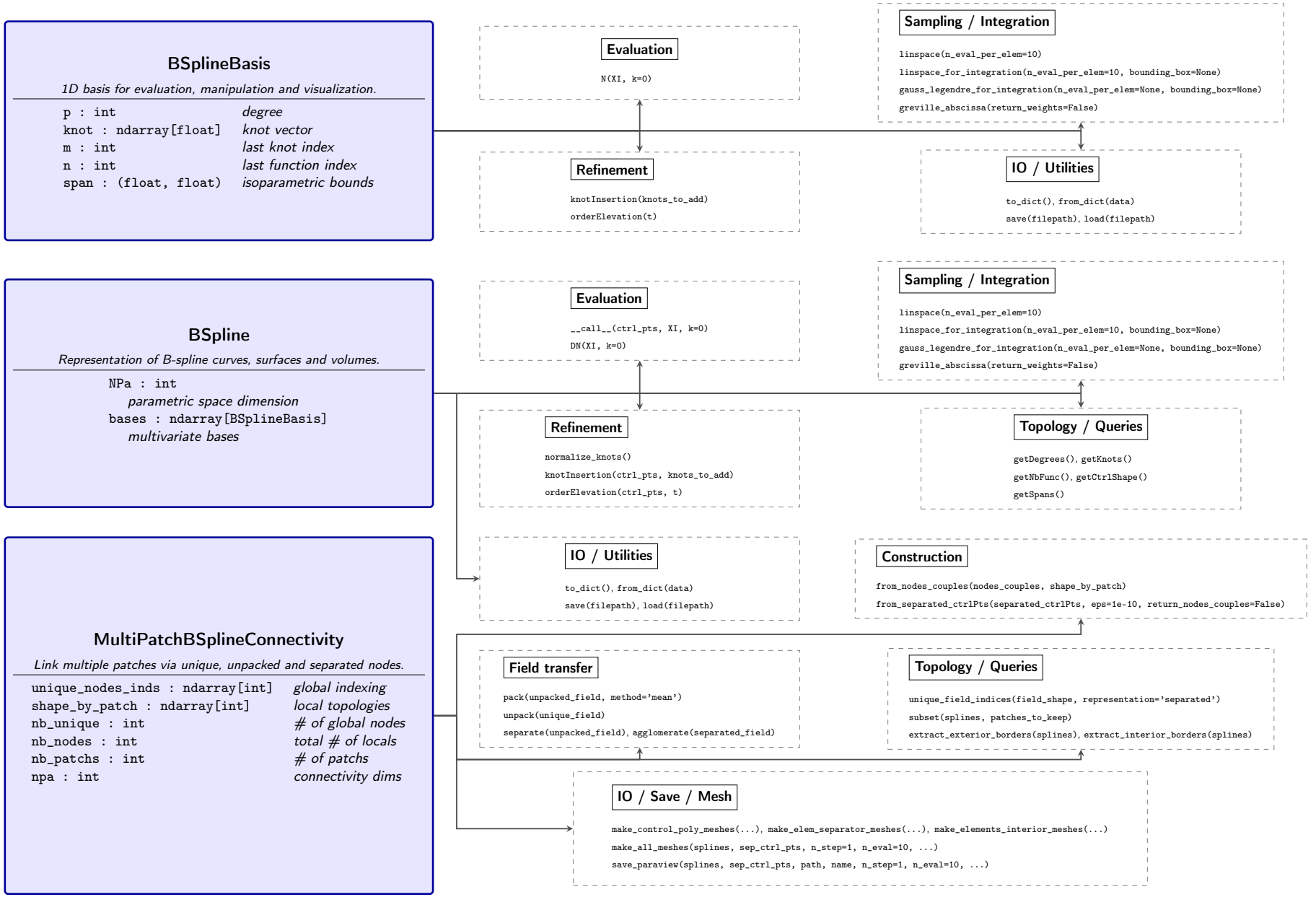
— **Challenge :** Générez un assemblage de deux patchs plans partageant une arête. Appliquez un déplacement aléatoire sur les points de contrôle. Comparez l'export ParaView selon que vous manipulez les points *uniques* (continuité préservée) ou les points *séparés* (ouverture de fissures). *Optionnel : Ajustez cet assemblage sur une géométrie complexe de votre choix.*

5 Conclusion

La librairie `bsplyne` a été conçue pour offrir un équilibre entre la rigueur mathématique des B-splines et la performance numérique nécessaire aux calculs sur des géométries complexes. Que ce soit pour de la génération de maillage, de l'approximation de surfaces ou de l'analyse structurelle, la structure en couches (`BSplineBasis` → `BSpline` → `MultiPatchBSplineConnectivity`) permet une montée en puissance progressive.

Au-delà de la simple construction géométrique, la force de l'outil réside dans sa capacité à manipuler les champs de manière analytique. La méthode `.DN()` permet notamment d'accéder instantanément aux dérivées partielles de tout ordre par rapport à l'espace paramétrique (ex : `k=[0, 0, 1]` pour une dérivée directionnelle selon ζ dans une `BSpline` trivariée).

En maîtrisant la manipulation des tenseurs de points de contrôle et ces opérateurs de dérivation, vous disposez d'un levier puissant pour traiter des problèmes physiques complexes avec une efficacité proche du langage C grâce à la compilation JIT.


BSplineBasis
1D basis for evaluation, manipulation and visualization.
p : int degree
knot : ndarray[float] knot vector
m : int last knot index
n : int last function index
span : (float, float) isoparametric bounds
Evaluation
N(XI, k=0)
Refinement
knotInsertion(knots_to_add)
orderElevation(t)
Sampling / Integration
linspace(n_eval_per_elem=10)
linspace_for_integration(n_eval_per_elem=10, bounding_box=None)
gauss_legendre_for_integration(n_eval_per_elem=None, bounding_box=None)
greville_abscissa(return_weights=False)
IO / Utilities
to_dict(), from_dict(data)
save(filepath), load(filepath)
BSpline
Representation of B-spline curves, surfaces and volumes.
NPa : int
parametric space dimension
bases : ndarray[BSplineBasis]
multivariate bases
Evaluation
__call__(ctrl_pts, XI, k=0)
DN(XI, k=0)
Refinement
normalize_knots()
knotInsertion(ctrl_pts, knots_to_add)
orderElevation(ctrl_pts, t)
Sampling / Integration
linspace(n_eval_per_elem=10)
linspace_for_integration(n_eval_per_elem=10, bounding_box=None)
gauss_legendre_for_integration(n_eval_per_elem=None, bounding_box=None)
greville_abscissa(return_weights=False)
Topology / Queries
getDegrees(), getKnots()
getNbFunc(), getCtrlShape()
getSpans()
IO / Utilities
to_dict(), from_dict(data)
save(filepath), load(filepath)
MultiPatchBSplineConnectivity
Link multiple patches via unique, unpacked and separated nodes.
unique_nodes_inds : ndarray[int] global indexing
shape_by_patch : ndarray[int] local topologies
nb_unique : int # of global nodes
nb_nodes : int total # of locals
nb_patchs : int # of patchs
npa : int connectivity dims
Construction
from_nodes_couples(nodes_couples, shape_by_patch)
from_separated_ctrlPts(separated_ctrlPts, eps=1e-10, return_nodes_couples=False)
Field transfer
pack(unpacked_field, method='mean')
unpack(unique_field)
separate(unpacked_field), agglomerate(separated_field)
Topology / Queries
unique_field_indices(field_shape, representation='separated')
subset(splines, patches_to_keep)
extract_exterior_borders(splines), extract_interior_borders(splines)
IO / Save / Mesh
make_control_poly_meshes(...), make_elem_separator_meshes(...), make_elements_interior_meshes(...)
make_all_meshes(splines, sep_ctrl_pts, n_step=1, n_eval=10, ...)
save_paraview(splines, sep_ctrl_pts, path, name, n_step=1, n_eval=10, ...)