IGA_for_bsplyne

Onboarding guide for Isogeometric Analysis

Dorian BICHET
February 07, 2026

Abstract

This document serves as a guide for onboarding with the IGA_for_bsplyne library. It details the isogeometric framework, installation procedures including high-performance sparse solvers, and provides a pedagogical progression through the tutorial examples from boundary conditions to multipatch structural simulations.

1 Objectives

- ✓ **Physics:** Understand the mapping between B-spline geometry and linear elasticity.
- ✓ **DOF Management:** Master the affine mapping approach for Dirichlet constraints.
- ✓ **Performance:** Configure SuiteSparse solvers for large-scale 3D assemblies.

2 Environment and Installation

1. High-Performance Solvers (Optional but Recommended)

For large-scale 3D problems, `IGA_for_bsplyne` leverages `SuiteSparse` (via `scikit-sparse` and `sparseqr`) for significant speedups.

Recommended Installation (Conda):
Conda automatically manages complex non-Python dependencies. Run in order:

```
conda install scikit-sparse
pip install sparseqr
```

Alternative (System Package Manager):
Note: This method is not recommended as it often leads to complex compilation issues.

```
# macOS
brew install suitesparse && pip install scikit-sparse sparseqr
# Ubuntu/Debian
apt install libsuitesparse-dev && pip install scikit-sparse sparseqr
```

Fallback: If these are missing, the library automatically uses standard `scipy.sparse` and an internal Sparse QR implementation.

2. Library Installation

Depending on your usage, choose between the stable release or the development version:

Standard Installation (PyPI):

```
pip install IGA-for-bsplyne
```

Development Installation (Editable mode):
To modify the code or access the latest features:

```
git clone https://github.com/Dorian210/IGA_for_bsplyne
cd IGA_for_bsplyne
pip install -e .
```

Optional Visualization Support

The core dependency `bsplyne` is installed automatically. The optional `viz` extra enables interactive visualization through `bsplyne[viz]` (via `PyVista`).

```
pip install IGA-for-bsplyne[viz]
```

For editable installs:

```
pip install -e .[viz]
```

3 Architecture and Documentation

3.1 Software Hierarchy: From Geometry to Physics

The `IGA_for_bsplyne` architecture builds upon the spline objects to solve PDEs:

1. `Dirichlet.py`: Manages the static condensation of the system. It handles the mapping $u = \mathbf{C}d + k$, allowing for homogeneous, non-homogeneous, and slave-reference linear constraints.
2. `IGAPatch.py`: This is the "physics" engine. It computes the local Stiffness matrix $\mathbf{K}_e$ and Load vector $\mathbf{f}_e$ by integrating Hooke's law over the B-spline volume using Gaussian quadrature.
3. `ProblemIGA.py`: The orchestrator. It handles the global assembly of multiple patches using `MultiPatchBSplineConnectivity` and invokes the sparse solvers to find the displacement field $\mathbf{U}$.

3.2 Specialized Features: Immersed Models

Unlike standard FEA where materials are often uniform per element, the `IGAPatchDensity` class allows for a continuous interpolation of material properties. This is particularly powerful for:

- **Digital Twins**: Mapping CT-scan voxels directly into the integration mask of the spline.
- **Heterogeneous Media**: Modeling porosity or multi-material transitions without remeshing.

Visualizing Results (ParaView)

The library produces `.pvd` and `.vtu` files.
Tip: In ParaView, always use the `Warp By Vector` filter to see the physical deformation. For stress analysis, the library exports Von Mises and Principal stresses directly at the evaluation points.

4 Working with Examples and Hands-on Practice

Mastery of the `IGA_for_bsplyne` library relies on progressively modifying scripts in the `exemples/tutorial/` folder. Each step presents a technical challenge to validate the transition from pure geometry to physical simulation.

4.1 Step 1: Affine Mapping and Constraints (`01_dirichlet_affine_map.py`)

Introduction to the `Dirichlet` class, the core tool for managing the relationship between "physical" degrees of freedom (DOFs) and "reduced" solver DOFs.

- **Concept:** Understanding the linear mapping $\mathbf{u} = \mathbf{C}\mathbf{d} + \mathbf{k}$, where $\mathbf{C}$ represents the free directions and $\mathbf{k}$ the prescribed values.
- **Challenge:** Modify the script to lock 3 non-adjacent nodes to different non-zero values. Use the `dof_lsq` method to verify if you can perfectly recover your reduced input vector `d` from the resulting physical vector `u`.

4.2 Step 2: Linear Relations and Slaves (`02_dirichlet_linear_relations.py`)

Advanced DOF manipulation using slave-reference dependencies to enforce kinematic constraints.

- **Concept:** Eliminating DOFs by expressing them as a linear combination of other "reference" DOFs (e.g., $u_3 = 0.5u_1 + 0.5u_2$).
- **Challenge:** Implement a "Rigid Link" between two nodes: force node B to follow node A exactly ($u_B = 1.0 \cdot u_A$). Verify that for any random displacement applied to the reduced system, the physical displacements of nodes A and B remain strictly identical.

4.3 Step 3: Constraint Handlers and Rigid Bodies (`03_constraint_handler_pipeline.py`)

Using high-level abstractions to automate complex multi-point boundary conditions.

- **Concept:** The `DirichletConstraintHandler` allows for adding complex equations (Rigid Body Motions, fixity) before "compiling" them into a final `Dirichlet` mapping.
- **Challenge:** Add a second rigid body constraint to a different set of nodes. Attempt to fix a "Reference DOF" (rotation or translation) to a value other than zero and observe how the particular solution `k` propagates this movement to all associated slave nodes.

4.4 Step 4: The Mechanical Patch (`04_patch_stiffness_rhs.py`)

Moving from pure geometry to the assembly of the Stiffness matrix ($\mathbf{K}$) and Load vector ($\mathbf{f}$).

- **Concept:** Automatic integration of the constitutive law (Hooke's law) and surface traction (Neumann) over a B-Spline volume.
- **Challenge:** Change the material properties to an almost incompressible material ($\nu = 0.49$) and observe the impact on the conditioning of the stiffness matrix. Swap the pressure from the top face to a side face and verify that the sum of forces in the RHS still matches the theoretical load.

4.5 Step 5: Jacobian and Integration (`05_patch_geometry_integration.py`)

Deep dive into the Isogeometric change of variables and differential geometry.

- **Concept:** Computing the Jacobian (J), its determinant, and the physical gradients (∇N) to map the parametric space to the physical space.
- **Challenge:** Create a highly distorted "wedge" shape by collapsing two control points into one. Check if the Jacobian determinant becomes zero or negative at integration points and discuss how this affects the validity of the mechanical integration.

4.6 Step 6: Post-Processing Field Analysis (`06_patch_post_processing.py`)

Extracting physical insights (Strains, Stresses) from the solved displacement field.

- **Concept:** Transforming displacement control points into continuous stress fields using Voigt notation.
- **Challenge:** Apply a "Pure Shear" displacement field ($U_x = \gamma y$, $U_y = 0$, $U_z = 0$). Verify that the Von Mises stress is uniform across the patch and corresponds to the theoretical value $\sigma_{VM} = \sqrt{3} \cdot G \cdot \gamma$.

4.7 Step 7: Density-Based Material (`07_patch_density.py`)

Introduction to Variable Density IGA.

- **Concept:** Penalizing the stiffness matrix using a local density field $\rho(\xi)$ defined at control points.
- **Challenge:** Define a "checkerboard" density field (alternating 0.1 and 1.0). Use ParaView to visualize the stress field and identify "stress concentration" zones at the interfaces between high and low-density regions.

4.8 Step 8: Solving Single Patch Problems (`08_problem_single_patch.py`)

Integration of all previous concepts into a full linear elastic solver pipeline.

- **Concept:** Global assembly using the `ProblemIGA` class and management of the packed DOF layout.
- **Challenge:** Double the length of the beam and refine the mesh using knot insertion. Compare the maximum tip deflection with the Euler-Bernoulli beam theory prediction ($w = \frac{FL^3}{3EI}$).

4.9 Step 9: Multi-Patch Conformal Continuity (`09_problem_multipatch.py`)

Managing complex assemblies and manual system control for multi-volume bodies.

- **Concept:** Using `MultiPatchBSplineConnectivity` to "weld" patches together by merging coincident nodes.
- **Challenge:** Introduce a voluntary geometric gap (e.g., 10^{-2}) between two control points supposed to be comforming. Observe how the connectivity fails to merge nodes and how the crack is opening under loading.

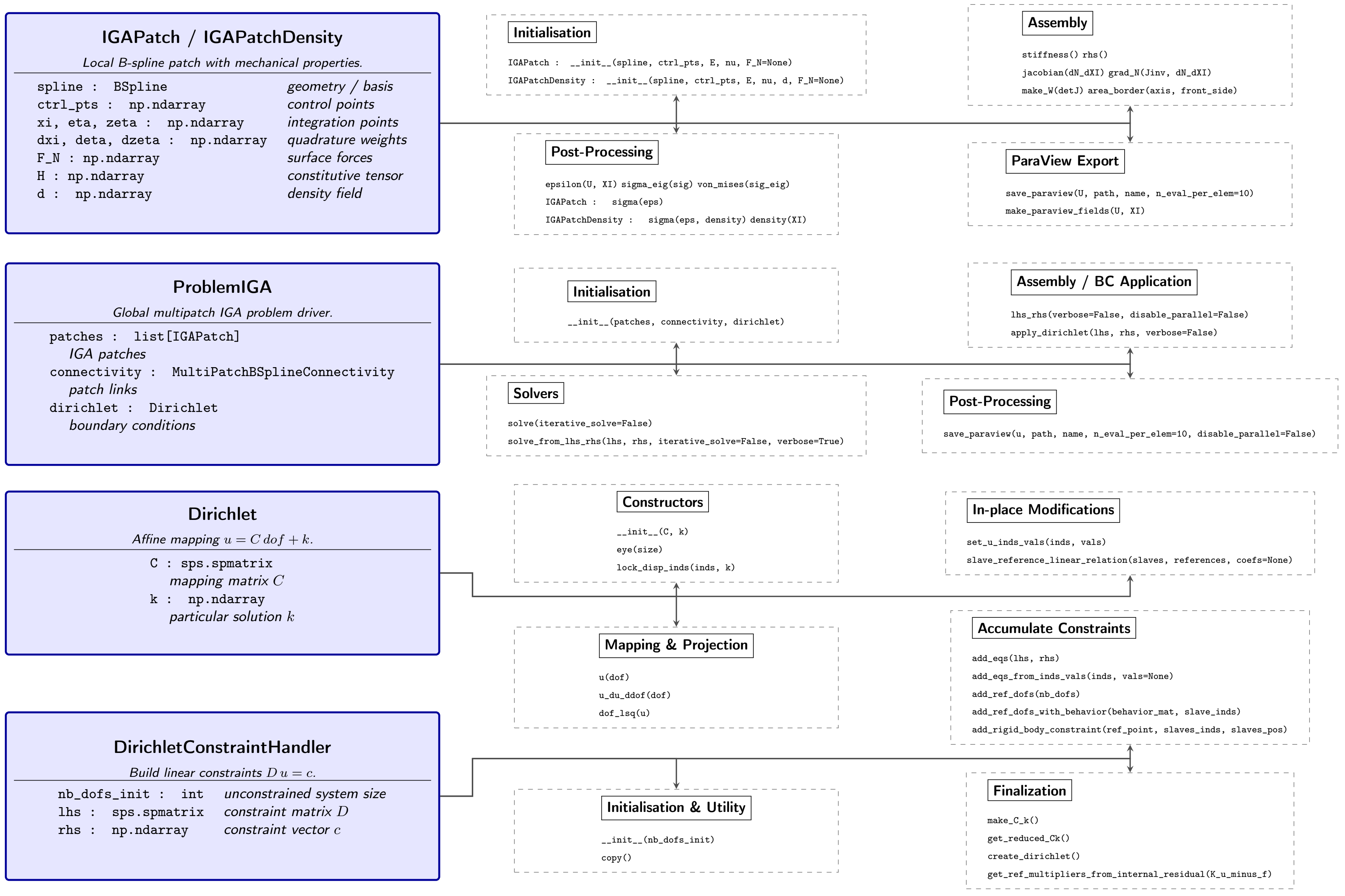
5 Conclusion

The `IGA_for_bsplyne` library extends the mathematical rigor of B-splines into the domain of physical simulation. By leveraging the Isogeometric Analysis paradigm, it reduces the traditional gap between CAD design and structural analysis. The modular architecture (`Dirichlet` $\to$ `IGAPatch` $\to$ `ProblemIGA`) ensures that users can transition seamlessly from simple single-patch verification to complex, multi-patch industrial assemblies.

Beyond standard linear elasticity, the library's true strength lies in its analytical foundation. The ability to evaluate high-order derivatives exactly via the `.DN()` method, combined with flexible integration schemes like the `IGAPatchDensity`, opens advanced possibilities for Digital Twins. Whether you are modeling organic structures from CT-scans or engineering high-performance lattice components, the framework provides the precision of splines with the numerical efficiency of JIT-compiled kernels and optimized sparse solvers.

By mastering the condensation of degrees of freedom and the assembly of mechanical operators, you possess a versatile tool for high-fidelity simulation, capable of handling the most demanding 3D structural challenges with near-native performance.

End of onboarding guide — `IGA_for_bsplyne` technical support — February 2026



IGAPatch / IGAPatchDensity

Local B-spline patch with mechanical properties.

```
spline :  BSpline                   geometry / basis
ctrl_pts :  np.ndarray              control points
xi, eta, zeta :  np.ndarray         integration points
dxi, deta, dzeta :  np.ndarray      quadrature weights
F_N : np.ndarray                    surface forces
H : np.ndarray                      constitutive tensor
d :  np.ndarray                     density field
```

Initialisation

```
IGAPatch :  __init__(spline, ctrl_pts, E, nu, F_N=None)
IGAPatchDensity :  __init__(spline, ctrl_pts, E, nu, d, F_N=None)
```

Post-Processing

```
epsilon(U, XI) sigma_eig(sig) von_mises(sig_eig)
IGAPatch :   sigma(eps)
IGAPatchDensity :   sigma(eps, density) density(XI)
```

Assembly

```
stiffness() rhs()
jacobian(dN_dXI) grad_N(Jinv, dN_dXI)
make_W(detJ) area_border(axis, front_side)
```

ParaView Export

```
save_paraview(U, path, name, n_eval_per_elem=10)
make_paraview_fields(U, XI)
```

ProblemIGA

Global multipatch IGA problem driver.

```
patches :  list[IGAPatch]
```
IGA patches
```
connectivity :  MultiPatchBSplineConnectivity
```
patch links
```
dirichlet :  Dirichlet
```
boundary conditions

Initialisation

```
__init__(patches, connectivity, dirichlet)
```

Solvers

```
solve(iterative_solve=False)
solve_from_lhs_rhs(lhs, rhs, iterative_solve=False, verbose=True)
```

Assembly / BC Application

```
lhs_rhs(verbose=False, disable_parallel=False)
apply_dirichlet(lhs, rhs, verbose=False)
```

Post-Processing

```
save_paraview(u, path, name, n_eval_per_elem=10, disable_parallel=False)
```

Dirichlet

Affine mapping $u = C\,dof + k$.

```
C : sps.spmatrix
```
mapping matrix C
```
k :  np.ndarray
```
particular solution k

Constructors

```
__init__(C, k)
eye(size)
lock_disp_inds(inds, k)
```

Mapping & Projection

```
u(dof)
u_du_ddof(dof)
dof_lsq(u)
```

In-place Modifications

```
set_u_inds_vals(inds, vals)
slave_reference_linear_relation(slaves, references, coefs=None)
```

DirichletConstraintHandler

Build linear constraints $D\,u = c$.

```
nb_dofs_init :  int    unconstrained system size
lhs :  sps.spmatrix    constraint matrix D
rhs :  np.ndarray      constraint vector c
```

Initialisation & Utility

```
__init__(nb_dofs_init)
copy()
```

Accumulate Constraints

```
add_eqs(lhs, rhs)
add_eqs_from_inds_vals(inds, vals=None)
add_ref_dofs(nb_dofs)
add_ref_dofs_with_behavior(behavior_mat, slave_inds)
add_rigid_body_constraint(ref_point, slaves_inds, slaves_pos)
```

Finalization

```
make_C_k()
get_reduced_Ck()
create_dirichlet()
get_ref_multipliers_from_internal_residual(K_u_minus_f)
```