LR-Splines Documentation

Release 0.1

Ivar Stangeby

May 07, 2020

Contents

CHAPTER 1

An LR-Spline implementation written in Python.

1.1 Basic Usage

The central object in the LRSplines-package is the LRSpline object. We initialize an LRSpline at the tensor-product level by specifying two knot vectors, and corresponding polynomial degrees. The following code initializes a bi-quadratic LR-spline.

1.1.1 Initialization and mesh visualization

```
import LRSplines

du = 2
dv = 2
knots_u = [0, 0, 0, 1, 2, 3, 3, 3]
knots_v = [0, 0, 0, 1, 2, 3, 3, 3]

LR = LRSplines.init_tensor_product_LR_spline(du, dv, knots_u, knots_v)
```

We can at any stage visualize the LR-mesh which underlies a given LR-spline, as seen:

```
LR.visualize_mesh()
```

yielding the image

Here, each **element** of the mesh displays the number of supported B-splines. In this case there are nine supported B-splines on each element. The green color indicates that the element is not **overloaded**. Each meshline displays its **multiplicity** indicated by a number in a white box. As we can see, the boundary mesh-lines have multiplicity 3, which reflects the knot vectors we chose. The dimension of the spline space is displayed at the top. In this case, we have five basis splines in each direction, totaling 25 tensor product B-splines.

1.1.2 Meshline insertion

We can insert mesh-lines into the mesh by creating a new Meshline object. In this example, we insert a meshline between the points (1.5, 0) and (1.5, 2) and between the points(1, 1.5) and (3, 1.5). This is represented in Python as:

```
m1 = LRSplines.Meshline(start=0, stop=2, constant_value=1.5, axis=0)
m2 = LRSplines.Meshline(start=1, stop=3, constant_value=1.5, axis=1)
```

The `axis` parameter determines the direction of the meshline, i.e., 0 for vertical and 1 for horizontal. We insert this meshline into the LR-spline, and visualize the result.

```
LR.insert_line(m1)
LR.insert_line(m2)
LR.visualize_mesh()
```

This yields the following image:

As we can see, some of the elements have turned red, indicating that they are now overloaded, which may result in loss of linear independence.

1.1.3 Evaluation

We can at any stage evaluate the LR-spline. At the moment there is no clever functionality for setting the coefficients of the underlying B-splines, but we can do so explicitly by looping over the set of basis functions `LR.S`.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# set the coefficients explicitly
for b in LR.B:
    b.coefficient = np.random.random(-3, 3)

N = 20
x = np.linspace(knots_u[0], knots_u[-1], N)
y = np.linspace(knots_v[0], knots_v[-1], N)
z = np.zeros((N, N))
X, Y = np.meshgrid(x, y)

for i in range(N):
    for j in range(N):
        z[i, j] = LR(x[i], y[j])

fig = plt.figure()
axs = Axes3D(fig)

axs.plot_wireframe(X, Y, z) # or plot_surface
```

This gives the resulting surface:

1.2 LRSplines API Reference

Below you will find an exhaustive list of all available methods in the LRSplines module.

1.2.1 LRSpline

class LRSplines.**LRSpline**(*mesh: List[Element], basis: List[BSpline], meshlines: List[Meshline], u_range=None, v_range=None, unique_global_knots_u=None, unique_global_knots_v=None*)

Represents a LRSpline, which is a tuple (M, S), where M is a mesh and S is a set of basis functions defined on M.

contains_basis_function(*B: LRSplines.b_spline.BSpline*) → bool

Returns true if B is found in self.S

Parameters **B** – BSpline to find

Returns true or false

contains_element(*element: LRSplines.element.Element*) → bool

Returns true if element is found in self.M

Parameters **element** – element to check

Returns true or false

edge_functions()

Returns the indices of all B-splines corresponding to an edge-degree-of-freedom. :return: np.ndarray

static get_full_span_meshline(*e: LRSplines.element.Element, axis*) → LRSplines.meshline.Meshline

Finds the meshline in direction prescribed by the axis that splits all the supported B-splines on the element. :param e: element to refine by :param axis: direction to look for split, 0 vertical, 1 horizontal :return: full span meshline

static get_minimal_span_meshline(*e: LRSplines.element.Element, axis*) → LRSplines.meshline.Meshline

Finds the shortest possible meshline in direction prescribed by axis that splits at least one supported B-spline on the element.

Parameters

- **e** – element to refine by
- **axis** – direction to look for split, 0 vertical, 1 horizontal

Returns minimal span meshline

insert_line(*meshline: LRSplines.meshline.Meshline, debug=False*) → None

Inserts a line in the mesh, splitting where necessary. Follows a four step procedure:

Step 1: Test all BSplines against the new meshline, and if the meshline traverses the support, split the BSpline into B1 and B2. For both B1 and B2, check whether they are already in the set of previous BSplines. If they are not, add them to the list of new functions. Add the function that was split to the list of functions to remove.

Step 2: Test all the new B-splines against all the meshlines already present in the mesh. They might have to be split further.

Step 3: Check all elements of the mesh, and make sure that any previous elements traversed by the new meshline are split accordingly.

Step 4: Make sure that all elements keep track of the basis functions they support, and that all basis functions keep track of the elements that support them.

Parameters **meshline** – meshline to insert

merge_meshlines(*meshline: LRSplines.meshline.Meshline*) → Tuple[bool, LRSplines.meshline.Meshline]

Tests the meshline against all currently stored meshlines, and combines, updates and deletes meshlines as needed. Returns true if the meshline is already in the list of previous meshlines. There are three cases:

1. The new meshline overlaps with a previous mesh line, but is not contained by the previous one.
2. The new meshline is completely contained in a previous mesh line, (may in fact be equal)
3. The new meshline is completely disjoint from all other meshlines.

Parameters **meshline** – meshline to test against previous meshlines.

Returns true if meshline was previously found, false otherwise.

mesh_to_array(*N=20*)

Returns the set of meshlines as an array of size (len(self.meshlines), 2, N) for transformation and plotting purposes (IGA).

Parameters **N** – Number of samples along each meshline

Returns np.ndarray

peelable()

Returns true if the peeling algorithms terminates with :return:

refine(*beta: float*, *error_function: Callable*, *refinement_strategy='minimal'*) → None

Refine the LR-mesh in order to introduce beta * dim(S) new degrees of freedom. The error function takes an element and returns the elemental error contribution. :param refinement_strategy: the refinement strategy used for splitting a single element. :param beta: growth parameter :param error_function: evaluates the error contribution from a given element :return: None

refine_by_element_full(*e: LRSplines.element.Element*) → None

Refines the LRSpline by finding and inserting a meshline that ensures that all supported BSplines on the given element will be split by the refinement.

Parameters **e** – element to refine

refine_by_element_minimal(*e: LRSplines.element.Element*) → None

Refines the LRSpline by finding and inserting the smallest possible meshline that splits the support of at least one BSpline.

Parameters **e** – element to refine

visualize_mesh(*multiplicity=True*, *overloading=True*, *text=True*, *relative=True*, *filename=None*, *color=False*, *title=True*, *axes=False*) → None

Plots the LR-mesh.

1.2.2 Meshline

class LRSplines.**Meshline**(*start: float*, *stop: float*, *constant_value: float*, *axis: int*, *multiplicity: int = 1*)

Represents a meshline (knotline) in given direction with designated endpoints.

contains(*other: LRSplines.meshline.Meshline*) → bool

Returns true if meshline is completely contained in this meshline.

Parameters **other** – meshline to check if is contained

Returns true if other is contained, false otherwise

midpoint
Returns the midpoint of the meshline.

Returns midpoint of the mesh line.

number_of_knots_contained(*basis: LRSplines.b_spline.BSpline*) → int
Returns the number of knots of given BSpline that lies on this meshline. :param basis: BSpline :return: number of knots of BSpline that lies on this meshline.

overlaps(*other: LRSplines.meshline.Meshline*) → bool
Returns true if the two meshlines overlap.

Parameters **other** – meshline to check for overlap

Returns true if the meshlines overlap, false otherwise

set_multiplicity(*knots*) → None
Sets the multiplicity of the mesh line according to how many knots in the knot vector overlaps with this constant value. :param knots: knot vector

splits_basis(*basis: LRSplines.b_spline.BSpline*) → bool
Returns true whether this mesh line traverses the interior of the support of the given basis function. :param basis: basis function to check split against :return: true or false

splits_element(*element: LRSplines.element.Element*) → bool
Returns true whether this meshline traverses the interior of given element. :param element: element to check split against :return: true or false

1.2.3 Element

class LRSplines.**Element**(*u_min: float*, *v_min: float*, *u_max: float*, *v_max: float*, *level: int = 0*)

add_supported_b_spline(*b_spline*)
Adds a B-spline to the list of supported B-splines.

Parameters **b_spline** – B-spline to add

area
Returns the area of the element.

Returns area of the element

contains(*u: float*, *v: float*) → bool
Returns True if this element contains the point (u, v)

Parameters

- **u** – u_component
- **v** – v_component

Returns

evaluate_basis(*u*, *v*)
Evaluates all the supported B-splines at the point u, v :param u: :param v: :return:

fetch_neighbours()
Returns a list of neighbouring elements based on supported B-splines.

Returns

get_supported_b_spline(*i: int*)
Returns the i-th supported B-spline.

Parameters **i** – index of supported B-spline

Returns b-spline i

has_supported_b_spline(*b_spline*) → bool
Returns True if given b_spline is among the list of supported b-splines.

Parameters **b_spline** – B-spline to check

Returns True or False

intersects(*other: LRSplines.element.Element*) → bool
Returns true if this element intersects the other element with *positive* area.

Parameters **other** – the element to check intersection with.

Returns true or false

is_overloaded() → bool
Returns true if the number of supported B-splines on this element is greater than (d1 + 1)*(d2 + 1).

Returns true if overloaded, false otherwise

midpoint
Returns the midpoint of the element.

Returns midpoint of the element

remove_supported_b_spline(*b_spline*)
Removes a B-spline from the list of supported B-splines.

Parameters **b_spline** – B-spline to remove

split(*axis: int*, *split_value: float*) → LRSplines.element.Element
Splits the element into two, resizing into the left half, and returning the right half.

Returns right half of element.

update_supported_basis(*b_splines: List[LRSplines.b_spline.BSpline]*) → None
Updates the list of supported basis functions.

Parameters **b_splines** – list of BSpline functions

1.2.4 BSpline

class LRSplines.**BSpline**(*degree_u: int, degree_v: int, knots_u: List[float], knots_v: List[float], weight: float = 1, end_u=False, end_v=False, north=False, south=False, east=False, west=False*)
Represents a single weighted tensor product B-spline with associated methods and fields.

add_to_support_if_intersects(*element: Element*) → bool
Returns true if the given element intersects the support of this BSpline, and adds element to the list of elements of support.

Parameters **element** – element in consideration

Returns true or false

intersects(*element: Element*) → bool
Returns true if the support of b_spline intersects the element with positive area.

Parameters

- **b_spline** – b_spline whose support is to be checked

- **element** – element whose domain is to be checked

Returns true or false

knot_average
Returns the knot average for this BSpline (the Greville point).

Returns the knot average (u, v).

overloaded
True if all its supporting elements are overloaded. :return: True or false

remove_from_support(*element: Element*) → bool
Removes given element from the list of elements with support. Returns true if element is found and removed, false otherwise.

Parameters **element** – element to remove

Returns true or false

update_weights(*other: LRSplines.b_spline.BSpline*) → None
Updates the weights during splitting.

This aim of Python library is to provide a lightweight framework for understanding LR-splines. The library is in no shape or form optimized for high performance computing, but is rather aimed at being a small toolkit for gaining some intuition for LR-splines. For more industrial grade performance and a more complete set of tools, see the GoTools library written in C++.

Other LR-spline-related projects:

1. LRSplines: A C++ library which some of the code in this repository is based on.
2. LRSplines: Android App: An app for interactive demonstration of the LR-spline refinement procedure.

1.3 Introduction

The need for adaptive refinement techniques is evident when it comes to optimizing the tradeoff between computational cost and computational accuracy. When utilizing spline spaces with an underlying tensor-product structure, refinement of a mesh induces a global propagation of the newly introduced meshlines to the whole mesh. This can be very inefficient. The concept of LR-Splines was introduced in 2013 in the paper Polynomial splines over locally refined box-partitions, and can be seen as an attempt to remedy this aforementioned problem. LR-Splines have several desirable properties:

1. They form a non-negative partition of unity by construction.
2. Linear independence (under some conditions on the refinement).

1.4 Construction

LR-splines are construced by starting with an initial **tensor product** spline space. Meshlines are then inserted one at the time, making sure the line **completely traverses** the support of at least one B-spline.

This B-spline is then **split** according to the standard knot insertion procedure, producing two new B-splines. These new B-splines are subsequently tested against **all previously existing meshlines**, to check for further splitting.

1.5 Installation

Download the repository and run:

```
python setup.py install
```

Verify the installation by running:

```
python -m import LRSplines
```

Python Module Index

l

Index

S

U

V