Research group
Discrete Geometries of Mathematics & Physics
www.mathischeap.com/dgmp

Discrete Geometries of Mathematics and Physics
Mimetic spectral element method[1]

Assignment #0

Lagrange & Edge polynomials: Reduction & Reconstruction

Yi Zhang (张仪)
: www.mathischeap.com
@: zhangyi_aero@hotmail.com
git: https://github.com/mathischeap

1 Definition

In one dimension, we consider an interval $\lambda \in I = [-1, 1]$. A partition of I is a set of $N+1$ nodes, $\lambda_0, \lambda_1, \lambda_2, \cdots, \lambda_N$, that satisfy

$$-1 = \lambda_0 < \lambda_1 < \lambda_2 < \cdots < \lambda_N = 1. \tag{1}$$

And, in this series of assignments, we will only use the Gauss-Lobatto nodes to define this partition. Given N, the function that computes the $N+1$ Gauss-Lobatto nodes which form a partition, i.e., satisfy (1), will be provided. You can find it at the main page of this course (see footnote) or just click on Gauss_Lobatto_nodes.py.

Over this partition, we can construct the well-known Lagrange polynomials in I as,

$$l^i(\lambda) = \prod_{j=0, j\neq i}^{N} \frac{\lambda - \lambda_j}{\lambda_i - \lambda_j}, \quad i \in \{0, 1, \cdots, N\}.$$

These $N+1$ polynomials are of a degree N. It is clear that these Lagrange polynomials satisfy the following nodal Kronecker delta property,

$$l^i(\lambda_j) = \delta^i_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{else} \end{cases}, \quad i, j \in \{0, 1, \cdots, N\}. \tag{2}$$

For an example of Lagrange polynomials, see Fig. 1.

[1] https://mathischeap.com/contents/teaching/advanced_numerical_methods/mimetic_spectral_element_method/main

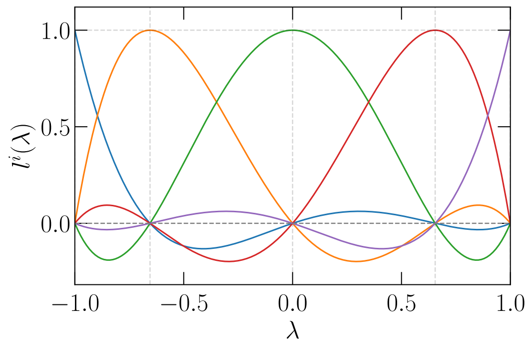

Fig. 1: An example of Lagrange polynomials at $N = 4$. It is clear that the Kronecker delta (2) is satisfied by these polynomials. The gray vertical lines indicate the nodes of the partition.

Assignment 0.1.0: Lagrange polynomials

Program function in Python which compute the Lagrange polynomials. The format of the function should be as follows.

```
def Lagrange_polynomials(nodes, lamb):
    """Compute the Lagrange polynomials.

    Parameters
    ----------
    nodes : 1d np.ndarray
        The partition of the interval I. It should be a 1d array of shape (m, ).
        For example, nodes = np.array([-1, -0.8, -0.3, 0.3, 0.8, 1]).
    lamb : 1d np.ndarray
        The coordinates we evaluate the Lagrange polynomials. It should be a 1d
        array of shape (n, ). For example, lamb = np.linspace(-1, 1, 100).

    Returns
    -------
    values : 2d np.ndarray
        It is a 2d array of shape (m+1, n). For example, values[0,:] represents
        first Lagrange polynomial evaluated on "lamb".

    """
```

Assignment 0.1.1: Visualization of Lagrange polynomials

Visualize Lagrange polynomials using the plot function of matplotlib.

```
>>> import matplotlib
```

The edge polynomials $e^i(\lambda)$ are linear combinations of derivatives of Lagrange Polynomials,

$$e^i(\lambda) := \sum_{j=i}^{N} \frac{\mathrm{d}l^j(\lambda)}{\mathrm{d}\lambda} = -\sum_{j=0}^{i-1} \frac{\mathrm{d}l^j(\lambda)}{\mathrm{d}\lambda}, \quad i \in \{1, 2, \cdots, N\}.$$

Edge polynomials are of a degree $N-1$. And they satisfy an integral Kronecker delta property,

$$\int_{\lambda_{j-1}}^{\lambda_j} e^i(\lambda)\mathrm{d}\lambda = \delta^i_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{else} \end{cases}, \quad i, j \in \{1, 2, \cdots, N\}\,. \tag{3}$$

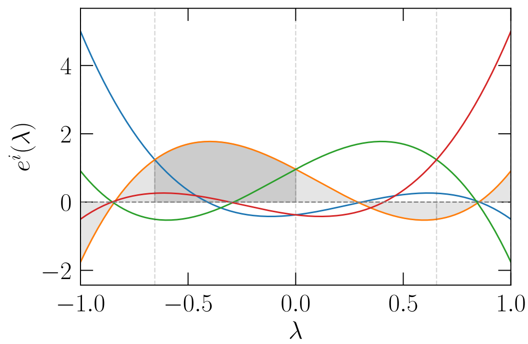

Fig. 2: An example of edge polynomials at $N = 4$. One can prove that the Kronecker delta (3) is satisfied by these polynomials. As an example, the edge polynomal $e^2(\lambda)$, i.e. the orange line, satisfies $\int_{\lambda_1}^{\lambda_2} e^2(\lambda)\mathrm{d}\lambda = 1$ and $\int_{\lambda_{i-1}}^{\lambda_i} e^2(\lambda)\mathrm{d}\lambda = 0$ if $i \in \{1, 3, 4\}$.

Assignment 0.2.0: Edge polynomials

Program function in Python which compute the edge polynomials. The format of the function should be as follows.

```python
def edge_polynomials(nodes, lamb):
    """Compute the edge polynomials.

    Parameters
    ----------
    nodes : 1d np.ndarray
        The partition of the interval I. It should be a 1d array of shape (m, ).
        For example, nodes = [-1, -0.8, -0.3, 0.3, 0.8, 1].
    lamb : 1d np.ndarray
        The coordinates we evaluate the edge polynomials. It should be a 1d array
        of shape (n, ). For example, lamb = np.linspace(-1, 1, 100).

    Returns
    -------
    values : 2d np.ndarray
        It is a 2d array of shape (m, n). For example, values[0,:] represents
        first edge polynomial evaluated on "lamb".

    """
```

Assignment 0.2.1: Visualization of edge polynomials

Visualize edge polynomials using the plot function of matplotlib.

```python
>>> import matplotlib
```

2 Reduction & reconstruction

The Lagranga polynomials are linearly independent. That is saying, recall the idea of linear (or vector) space in linear algebra, they as basis functions can form a base of a linear space. Let L denote the linear space they span, i.e.

$$\mathsf{L} := \mathrm{span}\left(l_0, l_1, \cdots, l_N\right).$$

Let p be a C^1 continuous function on I. We now can project p onto a polynomial $p_h \in \mathsf{L}$ as

$$p_h(\lambda) = \sum_{i=0}^{N} \mathsf{p}_i l^i(\lambda), \tag{4}$$

where

$$\mathsf{p}_i = p\left(\lambda_i\right) \quad i \in \{0, 1, \cdots, N\}. \tag{5}$$

And, according to the Kronecker delta property (2), we also know that

$$\mathsf{p}_i = p_h\left(\lambda_i\right), \quad i \in \{0, 1, \cdots, N\},$$

where p_i are usully called the expansion coefficients or degrees of freedom (DoF's). In linear algebra, $\mathsf{p}_0, \mathsf{p}_1, \cdots, \mathsf{p}_N$ are called the coordinates of p_h under the base $\left(l^0, l^1, \cdots, l^N\right)$.

The process of computing the expansion coefficients, i.e., (5), is called *reduction*, denoted by $\mathcal{I}$. And process of making p_h using the expansion coefficients and the basis functions is called *reconstruction*, denoted by $\mathcal{R}$. The *projection* operator, π, is defined as

$$\pi = \mathcal{R} \circ \mathcal{I},$$

i.e., the process of reduction and reconstruction together: $p \xrightarrow{\pi} p_h$.

The edge polynomials are also linearly independent. And we use E to denote the linear space

$$\mathsf{E} := \mathrm{span}\left(e_1, e_2, \cdots, e_N\right).$$

Let q be another C^0 continuous function on I. We can project q onto a polynomial $q_h \in \mathsf{E}$ as

$$q_h(\lambda) = \sum_{i=1}^{N} \mathsf{q}_i e^i(\lambda), \tag{6}$$

where the expansion coefficients are

$$\mathsf{q}_i = \int_{\lambda_{i-1}}^{\lambda_i} q\left(\lambda\right) \mathrm{d}\lambda, \quad i \in \{1, 2, \cdots, N\}. \tag{7}$$

Equations (6) and (7) defines reconstruction and reduction for the space E, respectively. According to the Kronecker delta property (3), we also know that the expansion coefficients satisfy

$$\mathsf{q}_i = \int_{\lambda_{i-1}}^{\lambda_i} q_h\left(\lambda\right) \mathrm{d}\lambda, \quad i \in \{1, 2, \cdots, N\}.$$

It is seen that either the Lagrange polynomial space L or the edge polynomial space E looks regular. The special thing is the connection between them as we will see now. Assume q is the derivative of p, i.e.,

$$q(\lambda) = p'(\lambda) = \frac{\mathrm{d}p(\lambda)}{\mathrm{d}\lambda}.$$

Then we will have that

$$q_h(\lambda) = p'_h(\lambda) = \frac{\mathrm{d}p_h(\lambda)}{\mathrm{d}\lambda}, \tag{8}$$

which is saying the projection operator commute with the derivative operator. In other words, we can first perform the derivative then do the projection. The output is same to that of a projection followed by a derivative.

Now, in more details, (8) is

$$\sum_{i=1}^{N} \mathsf{q}_i e^i(\lambda) = \frac{\mathrm{d}\left(\sum_{i=0}^{N} \mathsf{p}_i l^i(\lambda)\right)}{\mathrm{d}\lambda},$$

see (4) and (6). In this equation, the Lagrange polynomials and edge polynomials are known. The point of interest is the relation between the expansion coefficients: how to relate p_i to q_i. Without a proof, we give the following conclusion:

$$\mathsf{q}_i = \mathsf{p}_i - \mathsf{p}_{i-1}, \quad i \in \{1, 2, \cdots, N\}.$$

If we put the expansion coefficients in colume vectors,

$$\vec{p} = \begin{bmatrix} \mathsf{p}_0 \\ \mathsf{p}_1 \\ \vdots \\ \mathsf{p}_N \end{bmatrix}, \quad \vec{q} = \begin{bmatrix} \mathsf{q}_1 \\ \mathsf{q}_2 \\ \vdots \\ \mathsf{q}_N \end{bmatrix},$$

we will find an $N \times (N+1)$ matrix $\mathbb{E}$, called incidence matrix,

$$\mathbb{E} = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 1 \end{bmatrix},$$

that satisfies

$$\vec{q} = \mathbb{E}\vec{p}. \tag{9}$$

Now, we can see that, once we are given a function in L, to compute its derivative, we just need to apply the incidence matrix $\mathbb{E}$ to the vector of its expansion coefficients. And the output will be the expansion coefficients of its derivative in space E.

Assignment 0.3.0: Projection of Lagrange polynomial space

You need to program two functions, the first one does the reduction and the second one does the reconstruction.

```python
def Lagrange_space_reduction(nodes, func):
```

```
    """Reduce the function "func" to Lagrange polynomial space defined over "nodes".

    Parameters
    ----------
    nodes : np.ndarray
        The partition of the interval I. It should be a 1d array of shape (m, ). For example, nodes = [-1, -0.8, -0.3, 0.3, 0.8, 1].
    func :
        The C1 smooth function to be reduced to the Lagrange polynomial space.

    Returns
    -------
    expansion_coefficients : np.ndarray
        A 1d array of shape (m+1,) that contains the expansion coefficients.

    """

def Lagrange_space_reconstruction(nodes, expansion_coefficients, lamb):
    """Reconstruct the polynomial in the Lagrange polynomial space over "lamb".

    Parameters
    ----------
    nodes : np.ndarray
        The partition of the interval I. It should be a 1d array of shape (m, ). For example, nodes = [-1, -0.8, -0.3, 0.3, 0.8, 1].
    expansion_coefficients : np.ndarray
        A 1d array of shape (m+1, ) that contains the expansion coefficients.
    lamb : np.ndarray
        The coordinates we evaluate the polynomial. It should be a 1d array of shape (n, ). For example, lamb = np.linspace(-1, 1, 100).

    Returns
    -------
    reconstructed_values : np.ndarray
        A 1d array of shape (n, ) that represents the reconstructed values at "lamb".

    """
```

Assignment 0.3.1: Projection of edge polynomial space

You need to program two functions, the first one does the reduction and the second one does the reconstruction. *Tip: You can use the numerical integration function from scipy package (see `scipy.integrate`) for the integration.*

```
def edge_space_reduction(nodes, func):
    """Reduce the function "func" to edge polynomial space defined over "nodes".

    Parameters
    ----------
    nodes : np.ndarray
        The partition of the interval I. It should be a 1d array of shape (m, ). For example, nodes = [-1, -0.8, -0.3, 0.3, 0.8, 1].
    func :
        The function to be reduced to the edge polynomial space.
```

```

    Returns
    -------
    expansion_coefficients : np.ndarray
        A 1d array of shape (m, ) that contains the expansion coefficients.

    """

def edge_space_reconstruction(nodes, expansion_coefficients, lamb):
    """Reconstruct the polynomial in the edge polynomial space over "lamb".

    Parameters
    ----------
    nodes : np.ndarray
        The partition of the interval I. It should be a 1d array of shape (m, ).
        For example, nodes = [-1, -0.8, -0.3, 0.3, 0.8, 1].
    expansion_coefficients : np.ndarray
        A 1d array of shape (m, ) that contains the expansion coefficients.
    lamb : np.ndarray
        The coordinates we evaluate the polynomial. It should be a 1d array of
        shape (n, ). For example, lamb = np.linspace(-1, 1, 100).

    Returns
    -------
    reconstructed_values : np.ndarray
        A 1d array of shape (n, ) that represents the reconstructed values at
        "lamb".

    """
```

Assignment 0.3.2: Compute derivative with incidence matrix

You need to program a function to compute the incidence matrix first. Then you can play with the incidence matrix to verify (9).

```
def incidence matrix_1d(nodes):
    """Compute the incidence matrix.

    Parameters
    ----------
    nodes : np.ndarray
        The partition of the interval I. It should be a 1d array of shape (m, ).
        For example, nodes = [-1, -0.8, -0.3, 0.3, 0.8, 1].

    Returns
    -------
    incidence_matrix: np.ndarray
        A 2d array of shape (m, m+1) that represents the incidence matrix.

    """
```

3 Literature revisit

To understand Lagrange polynomials and edge polynomials from more angles, we refer to the original paper [1] where they are introduced.

References

[1] M. Gerritsma, Edge functions for spectral element methods, in: J. S. Hesthaven, E. M. Rønquist (Eds.), Spectral and High Order Methods for Partial Differential Equations, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 199–207.