FEALPy 偏微分方程数值解程序设计与实现:

单纯形网格上的标准 Lagrange 有限元方法

魏华祎

weihuayi@xtu.edu.cn

湘潭大学 • 数学与计算科学学院

July 5, 2021

Outline

Outline

Poisson 方程

给定区域 $\Omega \subset \mathbb{R}^d$, 其边界 $\partial\Omega = \Gamma_D \cup \Gamma_N \cup \Gamma_R$

$$
\begin{aligned}
-\Delta u &= f, \quad \text{in } \Omega \\
u &= g_D, \quad \text{on } \Gamma_D \leftarrow \textbf{Dirichlet} \\
\frac{\partial u}{\partial \boldsymbol{n}} &= g_N, \quad \text{on } \Gamma_N \leftarrow \textbf{Neumann} \\
\frac{\partial u}{\partial \boldsymbol{n}} + \kappa u &= g_R, \quad \text{on } \Gamma_R \leftarrow \textbf{Robin}
\end{aligned}
$$

其中

- $\Delta u(x) = u_{xx}$
- $\Delta u(x, y) = u_{xx} + u_{yy}$
- $\Delta u(x, y, z) = u_{xx} + u_{yy} + u_{zz}$
- $\frac{\partial u}{\partial \boldsymbol{n}} = \nabla u \cdot \boldsymbol{n}$

连续的弱形式

方程两端分别乘以测试函数 $v \in H^1_{D,0}(\Omega)$, 则连续的弱形式可以写为

$$(f, v) = -(\Delta u, v)$$

再分部积分

$$\begin{aligned}(f, v) &= -(\Delta u, v) \\ &= (\nabla u, \nabla v) - < \nabla u \cdot \boldsymbol{n}, v >_{\partial\Omega} \\ &= (\nabla u, \nabla v) - < g_N, v >_{\Gamma_N} + < \kappa u, v >_{\Gamma_R} - < g_R, v >_{\Gamma_R}\end{aligned}$$

整理可得

$$(\nabla u, \nabla v) + < \kappa u, v >_{\Gamma_R} = (f, v) + < g_R, v >_{\Gamma_R} + < g_N, v >_{\Gamma_N}$$

Remark

测试函数 $v|_{\Gamma_D} = 0$!

离散的弱形式

取一个 N 维的有限维空间 $V_h = \mathrm{span}\{\phi_i\}_0^{N-1}$, 其基函数向量记为

$$\boldsymbol{\phi} = [\phi_0, \phi_1, \cdots, \phi_{N-1}],$$

用 V_h 替代无限维的空间 $H^1_{D,0}(\Omega)$, 从而把问题转化为**离散的弱形式**: 求 $u_h = \boldsymbol{\phi}\boldsymbol{u} = \sum_{i=0}^{N-1} u_i\phi_i \in V_h$, 对任意 ϕ_i, 满足:

$$(\nabla u_h, \nabla\phi_i) + <\kappa u_h, \phi_i>_{\Gamma_R} = (f, \phi_i) + <g_R, \phi_i>_{\Gamma_R} + <g_N, \phi_i>_{\Gamma_N},$$

其中 $\boldsymbol{u}$ 是 u_h 在基函数 $\boldsymbol{\phi}$ 下的坐标**列向量**, 即 $\boldsymbol{u} = [u_0, u_1, \ldots, u_{N-1}]^T$。

离散的代数系统

$$(\boldsymbol{A}+\boldsymbol{R})\boldsymbol{u}=\boldsymbol{b}+\boldsymbol{b}_N+\boldsymbol{b}_R$$

$$\boldsymbol{A}=\int_\Omega(\nabla\boldsymbol{\phi})^T\nabla\boldsymbol{\phi}\,\mathrm{d}\boldsymbol{x}=\sum_{\tau\in\mathcal{T}}\int_\tau(\nabla\boldsymbol{\phi}|_\tau)^T\nabla\boldsymbol{\phi}|_\tau\,\mathrm{d}\boldsymbol{x}$$

$$\boldsymbol{R}=\int_{\Gamma_R}\boldsymbol{\phi}^T\boldsymbol{\phi}\,\mathrm{d}\boldsymbol{s}=\sum_{e_R\in\Gamma_R}\int_{e_R}(\boldsymbol{\phi}|_{e_R})^T\boldsymbol{\phi}|_{e_R}\,\mathrm{d}\boldsymbol{s}$$

$$\boldsymbol{b}=\int_\Omega f\boldsymbol{\phi}^T\,\mathrm{d}\boldsymbol{x}=\sum_{\tau\in\mathcal{T}}\int_\tau f(\boldsymbol{\phi}|_\tau)^T\,\mathrm{d}\boldsymbol{x}$$

$$\boldsymbol{b}_N=\int_{\Gamma_N}g_N\boldsymbol{\phi}^T\,\mathrm{d}\boldsymbol{s}=\sum_{e_N\in\Gamma_N}\int_{e_N}g_N(\boldsymbol{\phi}|_{e_N})^T\,\mathrm{d}\boldsymbol{s}$$

$$\boldsymbol{b}_R=\int_{\Gamma_R}g_R\boldsymbol{\phi}^T\,\mathrm{d}\boldsymbol{s}=\sum_{e_R\in\Gamma_R}\int_{e_R}g_R(\boldsymbol{\phi}|_{e_R})^T\,\mathrm{d}\boldsymbol{s}$$

单纯形网格

有限元方法构造有限维空间的方法，首先是把求解区域离散成网格，即很多**简单几何区域**组成的集合，如下面的**几何单纯形网格**：

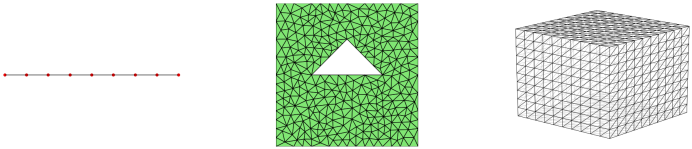

Figure: 区间, 三角形和四面体网格.

Remark

- 各种有限元方法的核心问题是如何构造有限维的子空间 V_h!
- 本节将讨论求解区域 Ω 离散为**单纯形网格**后，经典的 Lagrange 有限元空间的构造及程序实现.

Outline

几何单纯形上的重心坐标函数

记 $\{\boldsymbol{x}_i := [x_{i,0}, x_{i,1}, \ldots, x_{i,d-1}]\}_{i=0}^{d}$ 为 $\mathbb{R}^d$ 空间中的一组点，假设它们不在同一个超平面上，也即是说 d 个向量 $\boldsymbol{x}_0\boldsymbol{x}_1$，$\boldsymbol{x}_0\boldsymbol{x}_2$，$\cdots$，和 $\boldsymbol{x}_0\boldsymbol{x}_d$ 是线性无关的，等价于矩阵

$$\boldsymbol{A} = \begin{bmatrix} x_{0,0} & x_{1,0} & \cdots & x_{d,0} \\ x_{0,1} & x_{1,1} & \cdots & x_{d,1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{0,d-1} & x_{1,d-1} & \cdots & x_{d,d-1} \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

是非奇异的.

几何单纯形上的重心坐标函数

给定任一点 $\boldsymbol{x} = [x_0, x_1, \cdots, x_{d-1}]^T \in \mathbb{R}^d$, 求解如下线性代数系统, 可得一组实数值 $\boldsymbol{\lambda} := [\lambda_0(\boldsymbol{x}), \lambda_1(\boldsymbol{x}), \cdots, \lambda_d(\boldsymbol{x})]^T$:

$$\boldsymbol{A}\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

满足如下性质

$$\boldsymbol{x} = \sum_{i=0}^{d} \lambda_i(\boldsymbol{x})\boldsymbol{x}_i, \quad \sum_{i=0}^{d} \lambda_i(\boldsymbol{x}) = 1.$$

点集 $\{\boldsymbol{x}_i\}_{i=0}^d$ 形成的凸壳

$$\tau = \{\boldsymbol{x} = \sum_{i=0}^{d} \lambda_i \boldsymbol{x}_i | 0 \le \lambda_i \le 1, \sum_{i=0}^{d} \lambda_i = 1\}$$

称为一个**几何 d-单纯形**. $\boldsymbol{\lambda}$ 称为 $\boldsymbol{x}$ 对应的**重心坐标**向量.

几何单纯形上的重心坐标函数

易知, $\lambda_0(\boldsymbol{x})$, $\lambda_1(\boldsymbol{x})$, $\cdots$, 和 $\lambda_d(\boldsymbol{x})$ 是关于 $\boldsymbol{x}$ 线性函数, 且

$$\lambda_i(\boldsymbol{x}_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, i, j = 0, \cdots, d$$

区间单元 $[x_0, x_1]$ 上的重心坐标函数

给定区间单元上的一个重心坐标 (λ_0, λ_1)，存在 $x \in [x_0, x_1]$，使得：

$$\lambda_0 := \frac{x_1 - x}{x_1 - x_0}, \quad \lambda_1 := \frac{x - x_0}{x_1 - x_0}$$

显然

$$\lambda_0 + \lambda_1 = 1$$

重心坐标关于 x 的导数为：

$$\frac{\mathrm{d}\lambda_0}{\mathrm{d}x} = -\frac{1}{x_1 - x_0}, \quad \frac{\mathrm{d}\lambda_1}{\mathrm{d}x} = \frac{1}{x_1 - x_0}$$

三角形单元 $[\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2]$ 上的重心坐标函数

因为 $\lambda_0, \lambda_1, \lambda_2$ 是关于 $\boldsymbol{x}$ 线性函数，它梯度分别为:

$$\nabla\lambda_0 = \frac{1}{2|\tau|}(\boldsymbol{x}_2 - \boldsymbol{x}_1)\boldsymbol{W}$$

$$\nabla\lambda_1 = \frac{1}{2|\tau|}(\boldsymbol{x}_0 - \boldsymbol{x}_2)\boldsymbol{W}$$

$$\nabla\lambda_2 = \frac{1}{2|\tau|}(\boldsymbol{x}_1 - \boldsymbol{x}_0)\boldsymbol{W}$$

其中

$$\boldsymbol{W} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

注意这里的 $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, 和 $\boldsymbol{x}_2$ 是行向量。

三角形单元上的重心坐标函数的梯度计算

Python code 1 重心坐标梯度的计算代码。

```python
import numpy as np
from fealpy.mesh import MeshFactory as MF
box = [0, 1, 0, 1]
mesh = MF.boxmesh2d(box, nx=1, ny=1, meshtype='tri')
NC = mesh.number_of_cells()

node = mesh.entity('node')
cell = mesh.entity('cell')
v0 = node[cell[:, 2], :] - node[cell[:, 1], :] # x_2 - x_1
v1 = node[cell[:, 0], :] - node[cell[:, 2], :] # x_0 - x_2
v2 = node[cell[:, 1], :] - node[cell[:, 0], :] # x_1 - x_0
nv = np.cross(v2, -v1)

Dlambda = np.zeros((NC, 3, 2), dtype=np.float64)
length = nv #
W = np.array([[0, 1], [-1, 0]], dtype=np.int_)
Dlambda[:,0,:] = v0@W/length.reshape(-1, 1)
Dlambda[:,1,:] = v1@W/length.reshape(-1, 1)
Dlambda[:,2,:] = v2@W/length.reshape(-1, 1)
```

作业

给定一个一维区间网格,写出重心坐标函数**梯度的通用数学计算公式**, 并编程计算每个单元每个重心坐标函数的梯度.

```python
import numpy as np
from fealpy.mesh import IntervalMesh
node = np.array([[0.0], [0.5], [1.0]], dtype=np.float64)
cell = np.array([[0, 1], [1, 2]], dtype=np.int_)
mesh = IntervalMesh(node, cell)
```

作业

给定一个三维四面体网格,写出重心坐标函数**梯度的通用数学计算公式**, 并编程计算每个单元每个重心坐标函数的梯度.

```
import numpy as np
from fealpy.mesh import MeshFactory as MF
mesh = MF.one_tetrahedron_mesh(meshtype='iso')
```

Outline

有限元空间的构造与实现

有限元方法是基于网格来构造有限维空间的,这样的空间称为**有限元空间**。

从程序实现角度来看,构造有限元空间的关键是构造**全局基函数**,而构造**全局基函数**的关键是构造**每个网格单元上的局部基函数**。

下面讨论如何借助重心坐标的概念,来构造d**-维几何单纯形单元**上的局部基函数。

Remark

注意,这里构造完全不用引入**参考单元**的概念!

$d+1$ 维多重指标向量

给定任意一个正整数 p, 可定义多重指标向量 $\boldsymbol{m} := [m_0, m_1, \cdots, m_d]$, 满足

$$m_i \in \mathbb{N}, i = 0, 1, \cdots, d, \text{ 和} \sum_{i=0}^{d} m_i = p.$$

$\boldsymbol{m}$ 的所有可能取值个数为

$$n_p := \binom{p+d}{d} = \begin{cases} p+1, & 1D \\ \frac{(p+1)(p+2)}{2}, & 2D \\ \frac{(p+1)(p+2)(p+3)}{6}, & 3D \end{cases}$$

Remark

作业: 证明上述公式成立.

多重指标、重心坐标与笛卡尔坐标

给定任意一个正整数 p, 每个多重指标 $\boldsymbol{m}_\alpha$ 都对应一个**唯一的重心坐标点**

$$\boldsymbol{\lambda}_\alpha = [\lambda_0, \lambda_1, \cdots, \lambda_d] = [\frac{m_0}{p}, \frac{m_1}{p}, \cdots, \frac{m_d}{p}].$$

给定任意一个 d 维几何单纯形 $\tau = (\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_d)$, 每个 $\boldsymbol{m}_\alpha$ 都对应 τ 上一个**唯一的笛卡尔几何坐标点**

$$\boldsymbol{x}_\alpha = \lambda_0\boldsymbol{x}_0 + \lambda_1\boldsymbol{x}_1 + \cdots + \lambda_d\boldsymbol{x}_d = \frac{m_0}{p}\boldsymbol{x}_0 + \frac{m_1}{p}\boldsymbol{x}_1 + \cdots + \frac{m_d}{p}\boldsymbol{x}_d.$$

多重指标与多项式函数

给定任意一个 d 维几何单纯形 $\tau = (\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_d)$，每个 $\boldsymbol{m}_\alpha$ 都对应一个**唯一的定义在 τ 上 p 次多项式函数**

$$\phi_\alpha(\boldsymbol{x}) = \frac{1}{\boldsymbol{m}_\alpha!} \prod_{i=0}^{d} \prod_{j=0}^{m_i-1} [p\lambda_i(\boldsymbol{x}) - j].$$

其中

$$\boldsymbol{m}_\alpha! = m_0! m_1! \cdots m_d!, \qquad \prod_{j=0}^{-1} [p\lambda_i(\boldsymbol{x}) - j] = 1, \quad i = 0, 1, \cdots, d$$

易知 $\boldsymbol{x}_\alpha$ 是 ϕ_α 对应的插值点，满足

$$\phi_\alpha(\boldsymbol{x}_\beta) = \begin{cases} 1, & \alpha = \beta \\ 0, & \alpha \neq \beta \end{cases} \quad \forall \alpha, \beta = 0, 1, \cdots, n_p - 1 (\text{证明这个性质!})$$

多重指标的一维编号规则

记 α 为多重向量指标 $\boldsymbol{m}$ 的一个从 0 到 $n_p - 1$ 一维编号, 编号规则如下:

α	$\boldsymbol{m}_\alpha$				
0	p	0	0	$\cdots$	0
1	p-1	1	0	$\cdots$	0
2	p-1	0	1	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
d	p-1	0	0	$\cdots$	1
d+1	p-2	2	0	$\cdots$	0
d+2	p-2	1	1	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
2d-1	p-2	1	0	$\cdots$	1
2d	p-2	0	2	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n_p	0	0	0	$\cdots$	p

Table: 多重指标 $\boldsymbol{m}_\alpha$ 的一维编号规则.

Remark

注意, 这个编号规则不是唯一的选择!

多重指标的一维编号规则

记 α 为多重向量指标 $\boldsymbol{m}$ 的一个从 0 到 $n_p - 1$ 一维编号, 编号规则如下:

α	$\boldsymbol{m}_\alpha$				
0	p	0	0	$\cdots$	0
1	p-1	1	0	$\cdots$	0
2	p-1	0	1	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
d	p-1	0	0	$\cdots$	1
d+1	p-2	2	0	$\cdots$	0
d+2	p-2	1	1	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
2d-1	p-2	1	0	$\cdots$	1
2d	p-2	0	2	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n_p	0	0	0	$\cdots$	p

Table: 多重指标 $\boldsymbol{m}_\alpha$ 的一维编号规则.

Remark

注意,这个编号规则不是唯一的选择!

三角形单元上多重指标、插值点与基函数的局部编号规则

α	$\boldsymbol{m}_\alpha$		
0:	3	0	0
1:	2	1	0
2:	2	0	1
3:	1	2	0
4:	1	1	1
5:	1	0	2
6:	0	3	0
7:	0	2	1
8:	0	1	2
9:	0	0	3

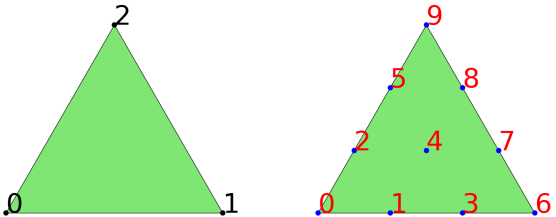

Figure: 三角形上的 $p = 3$ 次基函数插值点的编号规则.

Remark

$$\boldsymbol{m}_\alpha \leftrightarrow \boldsymbol{\lambda}_\alpha = \boldsymbol{m}_\alpha / p \leftrightarrow \boldsymbol{x}_\alpha \leftrightarrow \phi_\alpha(\boldsymbol{x})$$

区间单元上多重指标、插值点与基函数的局部编号规则

α	$\boldsymbol{m}_\alpha$	
0:	4	0
1:	3	1
2:	2	2
3:	1	3
4:	0	4

Figure: 区间上的 $p = 4$ 次基函数插值点的编号规则.

Remark

$$\boldsymbol{m}_\alpha \leftrightarrow \boldsymbol{\lambda}_\alpha = \boldsymbol{m}_\alpha / p \leftrightarrow \boldsymbol{x}_\alpha \leftrightarrow \phi_\alpha(\boldsymbol{x})$$

四面体单元上多重指标、插值点与基函数的局部编号规则

α	$\boldsymbol{m}_\alpha$			
0:	4	0	0	0
1:	3	1	0	0
2:	3	0	1	0
3:	3	0	0	1
4:	2	2	0	0
⋮	⋮	⋮	⋮	⋮
34:	0	0	0	4

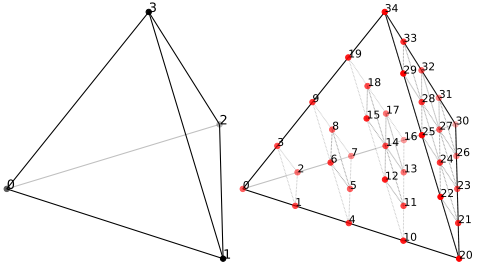

Figure: 四面体上的 $p = 4$ 次基函数插值点的编号规则.

Remark

$$\boldsymbol{m}_\alpha \leftrightarrow \boldsymbol{\lambda}_\alpha = \boldsymbol{m}_\alpha / p \leftrightarrow \boldsymbol{x}_\alpha \leftrightarrow \phi_\alpha(\boldsymbol{x})$$

ϕ_α 的数组化计算过程

下面讨论给定一重心坐标向量 $\boldsymbol{\lambda} = [\lambda_0, \lambda_1, \ldots, \lambda_d]$, 对应函数值 ϕ_α 的数组化运算过程

$$\phi_\alpha = \frac{1}{\boldsymbol{m}_\alpha!} \prod_{j_0=0}^{m_0-1} [p\lambda_0 - j_0] \prod_{j_1=0}^{m_1-1} [p\lambda_1 - j_1] \cdots \prod_{j_d=0}^{m_d-1} [p\lambda_d - j_d]$$

Remark

注意**数组化编程的前提**: 数学计算公式可以用**矩阵向量**的运算来分解!

ϕ_α 的数组化计算过程

首先构造向量和矩阵

$$P = \left[\frac{1}{0!}, \frac{1}{1!}, \frac{1}{2!}, \cdots, \frac{1}{p!}\right], \boldsymbol{A} = \begin{bmatrix} 1 & 1 & \cdot & 1 \\ p\lambda_0 & p\lambda_1 & \cdots & p\lambda_d \\ p\lambda_0 - 1 & p\lambda_1 - 1 & \cdots & p\lambda_d - 1 \\ \vdots & \vdots & \ddots & \vdots \\ p\lambda_0 - (p-1) & p\lambda_1 - (p-1) & \vdots & p\lambda_d - (p-1) \end{bmatrix},$$

Remark

$$\phi_\alpha = \frac{1}{\boldsymbol{m}_\alpha!} \prod_{j_0=0}^{m_0-1} [p\lambda_0 - j_0] \prod_{j_1=0}^{m_1-1} [p\lambda_1 - j_1] \cdots \prod_{j_d=0}^{m_d-1} [p\lambda_d - j_d]$$

ϕ_α 的数组化计算过程

矩阵 $\boldsymbol{A}$ 的每一列做累乘运算, 并左乘对角矩阵 $\mathrm{diag}(\boldsymbol{P})$, 可得

$$\boldsymbol{B} = \mathrm{diag}(\boldsymbol{P}) \begin{bmatrix} 1 & 1 & \cdots & 1 \\ p\lambda_0 & p\lambda_1 & \cdots & p\lambda_d \\ \prod_{j_0=0}^{1}[p\lambda_0 - j_0] & \prod_{j_1=0}^{1}[p\lambda_1 - j_1] & \cdots & \prod_{j_d=0}^{1}[p\lambda_d - j_d] \\ \vdots & \vdots & \ddots & \vdots \\ \prod_{j_0=0}^{p-1}[p\lambda_0 - j_0] & \prod_{j_1=0}^{p-1}[p\lambda_1 - j_1] & \cdots & \prod_{j_d=0}^{p-1}[p\lambda_d - j_d] \end{bmatrix}$$

Remark

$$\phi_\alpha = \frac{1}{\boldsymbol{m}_\alpha!} \prod_{j_0=0}^{m_0-1}[p\lambda_0 - j_0] \prod_{j_1=0}^{m_1-1}[p\lambda_1 - j_1] \cdots \prod_{j_d=0}^{m_d-1}[p\lambda_d - j_d]$$

ϕ_α 的数组化计算过程

$$\boldsymbol{B} = \operatorname{diag}(\boldsymbol{P}) \begin{bmatrix} 1 & 1 & \cdots & 1 \\ p\lambda_0 & p\lambda_1 & \cdots & p\lambda_d \\ \prod_{j_0=0}^{1}[p\lambda_0 - j_0] & \prod_{j_1=0}^{1}[p\lambda_1 - j_1] & \cdots & \prod_{j_d=0}^{1}[p\lambda_d - j_d] \\ \vdots & \vdots & \ddots & \vdots \\ \prod_{j_0=0}^{p-1}[p\lambda_0 - j_0] & \prod_{j_1=0}^{p-1}[p\lambda_1 - j_1] & \cdots & \prod_{j_d=0}^{p-1}[p\lambda_d - j_d] \end{bmatrix}$$

注意 $\boldsymbol{B}$ 包含 ϕ_α 的所有计算模块, 其第 i 列的第 m_i 行是一个 m_i 次的多项式, $\boldsymbol{m}_\alpha = [m_0, m_1, \ldots, m_d]$ 对应的局部基函数 ϕ_α 最终计算形式如下:

$$\phi_\alpha = \prod_{i=0}^{d} \boldsymbol{B}[m_i, i].$$

任意次任意维 Lagrange 有限元基函数的代码实现

假设已知如下变量, 我们展示 Lagrange 基函数的函数值计算代码

```
p # 基函数的次数
TD # 网格的拓扑维数
multiIndex # 多重指标矩阵
bc # 重心坐标矩阵
```

任意次任意维 **Lagrange** 有限元基函数代码实现

Python code 2 Lagrange 基函数计算展示代码。

```
# P = (1/1!, 1/2!, ···, 1/p!).
c = np.arange(1, p+1)
P = 1.0/np.multiply.accumulate(c)

# 生成矩阵 A.
t = np.arange(0, p)
shape = bc.shape[:-1]+(p+1, TD+1)
A = np.ones(shape, dtype=self.ftype)
A[..., 1:, :] = p*bc[..., np.newaxis, :] - t.reshape(-1, 1)
```

任意次任意维 Lagrange 有限元基函数代码实现

Python code 3 Lagrange 基函数计算展示代码。

```
# 矩阵 B(仍然用 A 的内存)
np.cumprod(A, axis=-2, out=A)
A[..., 1:, :] *= P.reshape(-1, 1)

# 计算每个积分点在每个单元每个基函数处的取值 φ_α
idx = np.arange(TD+1)
phi = np.prod(A[..., multiIndex, idx], axis=-1)
phi = phi[..., np.newaxis, :] # (NQ, 1, ldof)
```

$\nabla\phi_\alpha$ 的数组化计算过程

为计算 $\nabla\phi_\alpha$, 首先需要用到函数乘积求导法则来计算 $\prod_{j_i=0}^{m_i-1}(p\lambda_i - j_i)$ 的导数, 即

$$\nabla \prod_{j_i=0}^{m_i-1}(p\lambda_i - j_i) = p\sum_{j_i=0}^{m_i-1}\prod_{0\le k\le m_i-1, k\neq j_i}(p\lambda_i - k)\nabla\lambda_i.$$

Remark

这是一种标量的表达方式!

Remark

$$\phi_\alpha = \frac{1}{\boldsymbol{m}_\alpha!}\prod_{j_0=0}^{m_0-1}(p\lambda_0 - j_0)\prod_{j_1=0}^{m_1-1}(p\lambda_1 - j_1)\cdots\prod_{j_d=0}^{m_d-1}(p\lambda_d - j_d)$$

$\nabla\phi_\alpha$ 的数组化计算过程

用数组化的方式, 需要首先构造 $d+1$ 阶矩阵

$$\boldsymbol{D}^i = \begin{pmatrix} p & p\lambda_i - 0 & \cdots & p\lambda_i - 0 \\ p\lambda_i - 1 & p & \cdots & p\lambda_i - 1 \\ \vdots & \vdots & \ddots & \vdots \\ p\lambda_i - (p-1) & p\lambda_i - (p-1) & \cdots & p \end{pmatrix}, \quad 0 \le i \le d,$$

把 $\boldsymbol{D}^i$ 的每一列做累乘运算,然后取其下三角矩阵,再每一行求和, 即可得到矩阵 $\boldsymbol{B}$ 的每一列各个元素的求导后系数. 可得到矩阵 $\boldsymbol{D}$,其元素定义为

$$D_{i,j} = \sum_{m=0}^{j} \prod_{k=0}^{j} D_{k,m}^i, \quad 0 \le i \le d, \text{ and } 0 \le j \le p-1.$$

$\nabla\phi_\alpha$ 的数组化计算过程

进而可以计算 $\nabla \boldsymbol{B}$

$$
\begin{aligned}
\nabla \boldsymbol{B} =& \operatorname{diag}(\boldsymbol{P}) \begin{pmatrix} 0 & 0 & \cdots & 0 \\ D_{0,0}\nabla\lambda_0 & D_{1,0}\nabla\lambda_1 & \cdots & D_{d,0}\nabla\lambda_d \\ \vdots & \vdots & \ddots & \vdots \\ D_{0,p-1}\nabla\lambda_0 & D_{1,p-1}\nabla\lambda_1 & \cdots & D_{d,p-1}\nabla\lambda_d \end{pmatrix} \\
=& \operatorname{diag}(\boldsymbol{P}) \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \nabla\lambda_0 & & & \\ & \nabla\lambda_1 & & \\ & & \ddots & \\ & & & \nabla\lambda_d \end{pmatrix}
\end{aligned}
$$

数组化编程的一些总结

- 算法的数组化表达是数组化编程的基础和关键。数组化的表达要明确：
  - 算法的输入数组是什么?
  - 算法最终期望得到的数组是什么？
  - 算法中间需要的数组运算是什么？
- 代数学是算法数组化表达的有力工具。

Outline

有限维空间的实现关键

Galerkin 类型的离散方法，其程序实现的关键是**有限维空间的实现**，而有限维空间的实现的关键是

- 基函数及其导数计算
  - 主要是计算空间基函数及其导数在数值积分点处的值。
- 全局和局部自由度的管理
  - 全局自由度的编号规则，这与最终的离散代数系统中矩阵的行号和列号相对应。
  - 局部自由度的编号规则，即基函数在每个单元上的形函数的编号。
  - 局部自由度和全局自由度的对应规则，这是单元矩阵组装与总矩阵需要知道的关键信息。

LagrangeFiniteELementSpace 类的命名和接口约定

mesh	网格对象
p	空间的次数
GD	几何空间的维数
TD	几何空间的拓扑维数
dof	函数空间的自由度管理对象, 用于管理函数空间的自由度
ftype	函数空间所用的浮点数类型
itype	函数空间所用的整数类型
integralalg	数值积分算法类
spacetype	函数空间的类型,'C' 表示分片连续空间,'D' 表示断空间

Table: LagrangeFiniteElementSapce 的常用数据成员(属性)。

LagrangeFiniteELementSpace 类的命名和接口约定

ldof = space.number_of_local_dofs()	获得每个单元上的自由度个数
gdof = space.number_of_global_dofs()	获得所有单元上的总自由度个数
cell2dof = sapce.cell_to_dof()	获得单元与自由度的对应关系数组
bdof = space.boundary_dof()	获得边界的自由度
phi = space.basis(bc, ...)	计算单元上积分点处的基函数值
gphi = space.grad_basis(bc, ...)	计算单元上积分点处的基函数梯度值
val = space.value(uh, bc, ...)	计算有限元函数值
val = space.grad_value(uh, bc, ...)	计算有限元函数梯度值
uI = space.interpolation(u)	计算函数 u 在有限元空间中的插值函数
uh = space.function(...)	创建拉格朗日有限元空间的函数对象
A = space.stiff_matrix(...)	构造刚度矩阵
M = space.mass_matrix(...)	构造质量矩阵
F = space.source_vector(.. .)	构造源项向量

Table: LagrangeFiniteElementSapce 常用接口,其中省略号表示有默认参数。

LagrangeFiniteELementSpace 类的命名和接口约定

Remark

FEALPy 中的其它类型的有限维空间对象和拉格朗日有限元空间遵守几乎一样的命名和接口约定。

Outline

刚度矩阵的计算与组装

给定一个单纯形网格 $\mathcal{T}$，其有 NN 个节点，NC 个单元。定义在 $\mathcal{T}$ 的分片 p 次连续有限元空间 V_h 有 $gdof$ 个基函数，其组成的函数行向量为：

$$\boldsymbol{\phi} = [\phi_0, \phi_1, \cdots, \phi_{gdof-1}],$$

限止到每个网格单元 τ 上，共有 $ldof$ 个基函数：

$$\boldsymbol{\varphi}_\tau = [\varphi_0, \varphi_1, \cdots, \varphi_{ldof-1}],$$

Remark

这些基函数只有局部支集，即只在局部单元片上非零：

- 节点处的基函数只在共享该点的单元上非零。
- 边内部的基函数只在共享该边的单元上非零。
- 单元内部的基函数只在该单元上非零。

刚度矩阵的计算与组装

总体刚度矩阵的计算过程如下：

$$
\begin{aligned}
\boldsymbol{A} &= \int_\Omega (\nabla \boldsymbol{\phi})^T \nabla \boldsymbol{\phi} \, \mathrm{d}\boldsymbol{x} \\
&= \int_\Omega \begin{bmatrix} \nabla^T \phi_0 \\ \nabla^T \phi_1 \\ \vdots \\ \nabla^T \phi_{gdof-1} \end{bmatrix} \begin{bmatrix} \nabla \phi_0 & \nabla \phi_1 & \cdots & \nabla \phi_{gdof-1} \end{bmatrix} \mathrm{d}\boldsymbol{x}
\end{aligned}
$$

由基函数的局部支集性质，可知其大部分元素都为零，非零元素的计算也转化到每个网格单元上进行，最后再组装回来。

单元刚度矩阵的数值积分

$$\boldsymbol{A}_\tau = \int_\tau \begin{bmatrix} \nabla^T\phi_0 \\ \nabla^T\phi_1 \\ \vdots \\ \nabla^T\phi_{ldof-1} \end{bmatrix} \begin{bmatrix} \nabla\phi_0 & \nabla\phi_1 & \cdots & \nabla\phi_{ldof-1} \end{bmatrix} \mathrm{d}\boldsymbol{x}$$

$$\approx |\tau| \sum_{i=0}^{NQ-1} w_i \begin{bmatrix} \nabla^T\phi_0(\boldsymbol{x}_i) \\ \nabla^T\phi_1(\boldsymbol{x}_i) \\ \vdots \\ \nabla^T\phi_{ldof-1}(\boldsymbol{x}_i) \end{bmatrix} \begin{bmatrix} \nabla\phi_0(\boldsymbol{x}_i) & \nabla\phi_1(\boldsymbol{x}_i) & \cdots & \nabla\phi_{ldof-1}(\boldsymbol{x}_i) \end{bmatrix}$$

Python code 4 单元刚度矩阵的计算。

```python
# 假设已经定义好 mesh 和 space
qf = mesh.integrator(q, 'cell') # 获得第 q 个积分公式
bcs, ws = qf.get_quadrature_points_and_weights() # (NQ, TD+1) 获得积分点重心坐标和权重
cellmeasure = mesh.entity_measure('cell') # (NC, )
gphi = space.grad_basis(bcs) # (NQ, NC, ldof, GD), 计算基函数在重心坐标处的梯度值,

# 组装单元刚度矩阵,A.shape == (NC, ldof, ldof)
A = np.einsum('i, ijkl, ijml, j->jkm', ws, gphi, gphi, cellmeasure)
```

总体刚度矩阵的组装

Python code 5 总体刚度矩阵的组装。

```python
from scipy.sparse import csr_matrix

gdof = space.number_of_global_dof() # 全局自由度的个数

# (NC, ldof), cell2dof[i, j] 存储第 i 个单元上的局部第 j 个自由度的全局编号
cell2dof = space.cell_to_dof()

# (NC, ldof) --> (NC, ldof, 1) --> (NC, ldof, ldof)
I = np.broadcast_to(cell2dof[:, :, None], shape=A.shape)

# (NC, lodf) --> (NC, 1, ldof) --> (NC, ldof, ldof)
J = np.broadcast_to(cell2dof[:, None, :], shape=A.shape)

A = csr_matrix((A.flat, (I.flat, J.flat)), shape=(gdof, gdof))
```

Remark

- 质量矩阵的组装过程类似。

单元载荷向量的计算

给定源项 f, 载荷向量的计算公式如下:

$$\begin{aligned}\boldsymbol{F} &= \int_\Omega \boldsymbol{\phi}^T f \,\mathrm{d}\boldsymbol{x} \\ &= \int_\Omega \begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_{gdof-1} \end{bmatrix} f \,\mathrm{d}\boldsymbol{x}\end{aligned}$$

$$\begin{aligned}\boldsymbol{F}_\tau &= \int_\tau \begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_{ldof-1} \end{bmatrix} f \,\mathrm{d}\boldsymbol{x} \\ &\approx |\tau| \sum_{i=0}^{NQ-1} w_i \begin{bmatrix} \phi_0(\boldsymbol{x}_i) \\ \phi_1(\boldsymbol{x}_i) \\ \vdots \\ \phi_{ldof-1}(\boldsymbol{x}_i) \end{bmatrix} f(x_i)\end{aligned}$$

单元和总体载荷向量的组装

Python code 6 单元和总体载荷向量的组装。

```python
phi = space.basis(bcs) # (NQ, NC, ldof), 计算基函数在重心坐标处的值
ps = mesh.bc_to_point(bcs) # (NQ, NC, GD)
val = f(ps) # (NQ, NC) 计算右端项在积分点处的值
# (NC, ldof) 组装单元载荷向量
bb = np.einsum('i, ij, ijm, j->jm', ws, val, phi, cellmeasure)
gdof = space.number_of_global_dof() # 全局自由度的个数

# (NC, ldof), cell2dof[i, j] 存储第 i 个单元上的局部第 j 个自由度的全局编号
cell2dof = space.cell_to_dof() # (NC, ldof)

# 组装单元载荷向量到总体载荷向量中
F = np.zeros(gdof, dtype=np.float64)
np.add.at(F, cell2dof, bb)
```

Outline

纯 Dirichlet 边界条件处理

如果 Poisson 方程只有 Dirichlet 边界条件，则最终代数方程变为

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$$

把 $\boldsymbol{u}$ 进行向量分解

$$\boldsymbol{u} = \boldsymbol{u}_I(\text{区域内部自由度}) + \boldsymbol{u}_D(\text{Dirichlet 边界自由度})$$

代入离散代数系统，可得

$$\boldsymbol{A}\boldsymbol{u}_I = F := \boldsymbol{b} - \boldsymbol{A}\boldsymbol{u}_D$$

纯 Dirichlet 边界条件处理

$$\boldsymbol{A}\boldsymbol{u}_I = F := \boldsymbol{b} - \boldsymbol{A}\boldsymbol{u}_D$$

接着有两种方法进行离散系统求解

- 在 $\boldsymbol{A}$ 和 $\boldsymbol{F}$ 中,只区域内部自由度对应的子矩阵和子向量进行求解,这样会改变原始离散系统的规模。
- 把 F 中 Dirichlet 边界自由度对应的分量设真解的值,并把 A 中 Dirichlet 边界自由度对应的行列的主对角元素改为 1, 其它元素改为 0。

纯 Neumann 边界条例处理

如果 Poisson 方程只有 Neumann 边界条件, 则离散代数系统变为

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b} + \boldsymbol{b}_N$$

而模型真解是关于常数唯一,需要再加一个条件,常用的条件是:

$$\int_\Omega u \,\mathrm{d}\boldsymbol{x} = 0 \to \int_\Omega \boldsymbol{\phi}\boldsymbol{u} \,\mathrm{d}\boldsymbol{x} = 0 \to \boldsymbol{C}\boldsymbol{u} = 0$$

其中 $\boldsymbol{C} = [\int_\Omega \phi_0 \,\mathrm{d}\boldsymbol{x}, \cdots, \int_\Omega \phi_{N-1} \,\mathrm{d}\boldsymbol{x}]$。最终系统化为

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{C}^T \\ \boldsymbol{C} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \xi \end{bmatrix} = \begin{bmatrix} \boldsymbol{F} \\ 0 \end{bmatrix}$$

纯 Robin 边界条件处理

如果 Poisson 方程只有 Robin 边界条件, 则离散代数系统变为

$$(\boldsymbol{A} + \boldsymbol{R})\boldsymbol{u} = \boldsymbol{b} + \boldsymbol{b}_R$$

直接求解即可。

混合边界条件

如果 Poisson 方程是混合边界条件，

$$(\boldsymbol{A} + \boldsymbol{R})\boldsymbol{u} = \boldsymbol{b} + \boldsymbol{b}_R + \boldsymbol{b}_N$$

Remark

首先处理 Neumann 和 Robin 边界，最后处理 Dirichlet 边界。

作业

根据下面椭圆方程模型，构造一个二维有真解模型例子，用 p 次有限元进行求解，画出误差阶，并输出误差表格。

$$
\begin{aligned}
-\Delta u + 3u &= f, \quad \text{in } \Omega \\
u &= g_D, \quad \text{on } \Gamma_D \leftarrow \textbf{Dirichlet} \\
\frac{\partial u}{\partial \boldsymbol{n}} &= g_N, \quad \text{on } \Gamma_N \leftarrow \textbf{Neumann} \\
\frac{\partial u}{\partial \boldsymbol{n}} + u &= g_R, \quad \text{on } \Gamma_R \leftarrow \textbf{Robin}
\end{aligned}
$$

求解区域为 $[0,1]^2$, 其中在 $x=0$ 上是 Robin 边界条件，在 $x=1$ 的边界上为 Neumann 边界条件，其余边界为 Dirichlet 边界。

Outline