FEALPy:Finite Element Analysis Library in Python

—面向数组的思维和编程方式

魏华祎

weihuayi@xtu.edu.cn

湘潭大学 • 数学与计算科学学院

2019 年 11 月 22 日

Outline

Outline

背景和动机

在偏微分方程数值解多年的发展过程中，涌现了很多优秀的开源数值软件

- FEniCS(C++/Python, 开始于芝加哥大学和查尔姆斯理工大学)
- PETSc (C/Python, 美国阿贡国家实验室)
- deal.II (C++, 开始于德国海德堡大学)
- MFEM (C++, 美国劳伦斯利弗莫尔国家实验室)
- PHG (C, 张林波, 中国科学院)
- AFEPACK (C++, 李若, 北京大学)
- IFEM (Matlab, 陈龙, UCI)
- ......

背景和动机

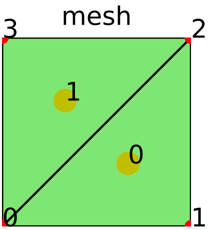
- 想偷懒省事
- 不想被学生气死
- 想发更多的 Paper
- 想有更多的时间陪陪家人和孩子
- 想对大家设计的新算法理解更深刻一点
- 想把计算数学的理论和算法变成真正的生产力

Outline

Outline

FEALPy 中的三角形网格

下面以三角形网格为例，来介绍 FEALPy 中网格数据结构和算法。一个三角形网格最少需要两个二维数组来存储网格信息：



0:	0.0	0.0
1:	1.0	0.0
2:	1.0	1.0
3:	0.0	1.0

node

0:	1	2	0
1:	3	0	2

cell

图: 三角形网格的基本数据结构示例。

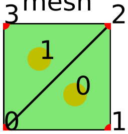
FEALPy 中的三角形网格

Python code 2 FEALPy 中的三角形网格示例代码

```python
from fealpy.mesh.TriangleMesh import TriangleMesh

node = np.array(
    [(0.0, 0.0),
     (1.0, 0.0),
     (1.0, 1.0),
     (0.0, 1.0)], dtype=np.float)
cell = np.array([
    (1, 2, 0),
    (3, 0, 2)], dtype=np.int)

mesh = TriangleMesh(node, cell)
node = mesh.entity('node')
edge = mesh.entity('edge')
cell = mesh.entity('cell')
```

edge 与 edge2cell 生成算法

mesh



cell

0:	1	2	0
1:	3	0	2

totalEdge

0:	2	0
1:	0	1
2:	1	2
3:	0	2
4:	2	3
5:	3	0

stotalEdge

0:	0	2
1:	0	1
2:	1	2
3:	0	2
4:	2	3
5:	0	3

i0

1
0
5
2
4

i1

1
3
5
2
4

edge

0:	0	1
1:	2	0
2:	3	0
3:	1	2
4:	2	3

localEdge

0:	1	2
1:	2	0
2:	0	1

图: 三角形网格的 edge 和 edge2cell 数组生成算法示例。

edge 与 edge2cell 生成算法

Python code 4 三角形网格生成 edge 数组算法代码

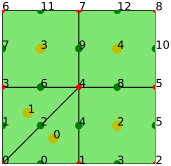
```
import numpy as np

localEdge = np.array([(1, 2), (2, 0), (0, 1)], dtype=np.int)

totalEdge = cell[:, localEdge].reshape(-1, 2)
stotalEdge = np.sort(totalEdge, axis=1)
_, i0, j = np.unique(stotalEdge, return_index=True, return_inverse=True, axis=0)

edge = totalEdge[i0]
NE = i0.shape[0]

i1 = np.zeros(NE, dtype=np.int)
NC = cell.shape[0]
i1[j] = np.arange(3*NC)

// 边与单元的拓扑关系数组
edge2cell = np.zeros((NE, 4), dtype=np.int)
t0 = i0//3
t1 = i1//3
k0 = i0%3
k1 = i1%3
edge2cell[:, 0] = t0
edge2cell[:, 1] = t1
edge2cell[:, 2] = k0
edge2cell[:, 3] = k1
```

edge 与 edge2cell 生成算法

- 这里介绍的 edge 与 edge2cell 数组生成算法是建立网格上面网格实体拓扑关系的核心算法.
- 该算法完全可推广到其它网格类型上.

多边形网格数据结构

mesh



node

0:	0.0	0.0
1:	0.5	0.0
2:	1.0	0.0
3:	0.0	0.5
4:	0.5	0.5
5:	1.0	0.5
6:	0.0	1.0
7:	0.5	1.0
8:	1.0	1.0

cell

1	4	0	3	0	4	1	2	5	4	3	4	7	6	4	5	8	7

cellLocation

0	3	6	10	14	18

edge

0:	0	1
1:	3	0
2:	4	0
3:	1	2
4:	1	4
5:	2	5
6:	4	3
7:	6	3
8:	5	4
9:	4	7
10:	5	8
11:	7	6
12:	8	7

edge2cell

0:	0	0	2	2
1:	1	1	0	0
2:	0	1	1	1
3:	2	2	0	0
4:	0	2	0	3
5:	2	2	1	1
6:	1	3	2	0
7:	3	3	3	3
8:	2	4	2	0
9:	3	4	1	3
10:	4	4	1	1
11:	3	3	2	2
12:	4	4	2	2

图： 多边形网格数据结构示例。

多边形网格数据结构

Python code 7 多边形网格生成示例代码

```python
import matplotlib.pyplot as plt
from fealpy.mesh.simple_mesh_generator import unitcircledomainmesh

mesh = unitcircledomainmesh(0.2, meshtype='polygon')

fig = plt.figure()
axes = fig.gca()
mesh.add_plot(axes)
plt.subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=0, hspace=0)
plt.show()
```

多边形网格数据结构

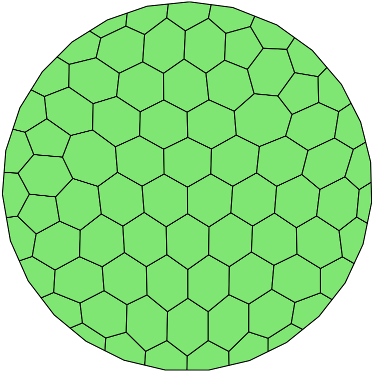


图: 多边形网格示例。

Outline

d-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

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

是非奇异的.

d-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 $\boldsymbol{\lambda} := (\lambda_0(\boldsymbol{x}), \lambda_1(\boldsymbol{x}), \cdots, \lambda_d(\boldsymbol{x}))^T$:

$$\boldsymbol{A}\boldsymbol{\lambda} = \boldsymbol{x}, \tag{2}$$

满足如下性质

$$\boldsymbol{x} = \sum_{i=0}^{d} \lambda_i(\boldsymbol{x})\boldsymbol{x}_i, \quad \sum_{i=0}^{d} \lambda_i(\boldsymbol{x}) = 1. \tag{3}$$

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

$$\tau = \{\boldsymbol{x} = \sum_{i=0}^{d} \lambda_i \boldsymbol{x}_i | 0 \leq \lambda_i \leq 1, \sum_{i=0}^{d} \lambda_i = 1\} \tag{4}$$

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.

d-单纯形上的重心坐标函数

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

$$\lambda_i(\boldsymbol{x}_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, i, j = 0, \cdots, d \tag{5}$$

接下来我们讨论 d-单纯形上的任意次拉格朗日形函数的构造.

$d+1$ 维多重指标

记 $\boldsymbol{m}$ 为 $d+1$ 维多重指标向量 $(m_0, m_1, \cdots, m_d)$, 满足

$$m_i \geq 0, i = 0, 1, \cdots, d, \text{ and } \sum_{i=0}^{d} m_i = p.$$

固定次数 p, $\boldsymbol{m}$ 的所有可能取值个数为

$$n_p := \binom{d}{p+d}$$

多重指标向量 $\boldsymbol{m}$ 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

表: 多重指标 $\boldsymbol{m}_\alpha$ 的编号规则.

高次拉格朗日形函数的一般公式

给定第 α 个多重指标向量 $\boldsymbol{m}_\alpha$, 在 d-单纯形 τ 上可以构造如下的 p 次多项式函数:

$$\phi_\alpha = \frac{1}{\boldsymbol{m}_\alpha!}\prod_{i=0}^{d}\prod_{j=0}^{m_i-1}(p\lambda_i - j). \tag{6}$$

其中

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

Sylvester's Formula

$$R_i(p,\lambda) = \begin{cases} \frac{1}{i!}\prod_{j=0}^{i-1}(p\lambda - j), & 1 \le i \le p \\ 1, & i = 0 \end{cases}$$

d-单纯形上的插值点

每个多重指标 $\boldsymbol{m}_\alpha$, 都对应 d-单纯形 τ 上的一个点 $\boldsymbol{x}_\alpha$,

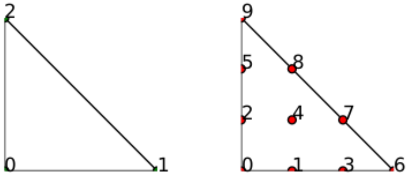
$$\boldsymbol{x}_\alpha = \sum_{i=0}^{d} \frac{m_i}{p} \boldsymbol{x}_i.$$

其中 m_i 是多重指标向量 $\boldsymbol{m}_\alpha$ 的第 i 个分量. 易知 $\boldsymbol{x}_\alpha$ 是 ϕ_α 对应的插值点, 满足

$$\phi_\alpha(\boldsymbol{x}_\beta) = \begin{cases} 1, & \alpha = \beta \\ 0, & \alpha \neq \beta \end{cases} \text{和} \alpha, \beta = 0, 1, \cdots, n_p - 1 \tag{7}$$

插值点的局部编号规则

$$\phi_{m,n,k} = \frac{1}{m!n!k!}\prod_{j=0}^{m-1}(p\lambda_0 - j)\prod_{j=0}^{n-1}(p\lambda_1 - j)\prod_{j=0}^{k-1}(p\lambda_2 - j).$$

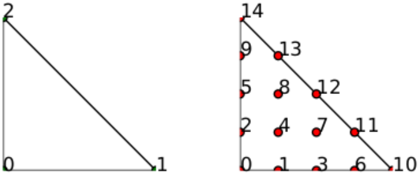


	m	n	k
0:	3	0	0
1:	2	1	0
2:	2	0	1
3:	1	2	0
4:	1	1	1
5:	1	0	2
6:	0	3	0
7:	0	2	1
8:	0	1	2
9:	0	0	3

图: 三角形上的 $p=3$ 次形函数对应的编号规则.

插值点的局部编号规则

$$\phi_{m,n,k} = \frac{1}{m!n!k!}\prod_{j_0=0}^{m-1}(p\lambda_0 - j_0)\prod_{j_1=0}^{n-1}(p\lambda_1 - j_1)\prod_{j_2=0}^{k-1}(p\lambda_2 - j_2).$$



	m	n	k
0:	4	0	0
1:	3	1	0
2:	3	0	1
3:	2	2	0
4:	2	1	1
5:	2	0	2
6:	1	3	0
7:	1	2	1
8:	1	1	2
9:	1	0	3
10:	0	4	0
11:	0	3	1
12:	0	2	2
13:	0	1	3
14:	0	0	4

图： 三角形上的 $p=4$ 次形函数对应的编号规则.

插值点的局部编号规则

图: 区间上的 $p = 4$ 次形函数对应的编号规则

插值点的局部编号规则

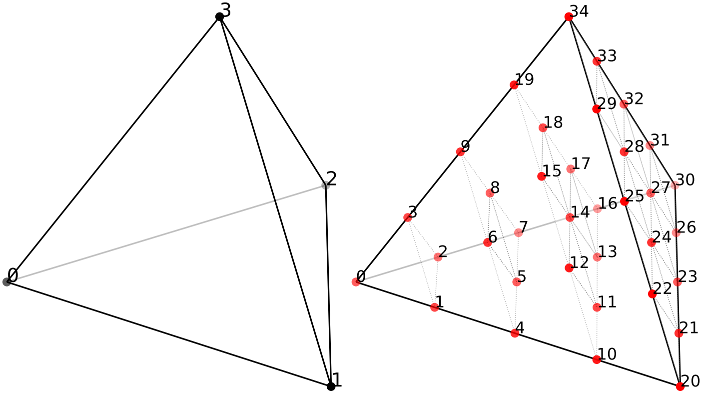


图: 四面体上的 $p = 4$ 次形函数对应的编号规则.

ϕ_α 的面向数组计算过程

首先构造向量和矩阵

$$
P = \left(\frac{1}{0!}, \frac{1}{1!}, \frac{1}{2!}, \cdots, \frac{1}{p!}\right),
$$

$$
\boldsymbol{A} := \begin{pmatrix}
1 & 1 & \cdot & 1 \\
p\lambda_0 & p\lambda_1 & \cdots & p\lambda_d \\
p\lambda_0 - 1 & p\lambda_1 - 1 & \cdots & p\lambda_d - 1 \\
\vdots & \vdots & \ddots & \vdots \\
p\lambda_0 - (p-1) & p\lambda_1 - (p-1) & \vdots & p\lambda_d - (p-1)
\end{pmatrix},
$$

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 的面向数组计算过程

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

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 的面向数组计算过程

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

注意 $\boldsymbol{B}$ 包含 ϕ_α 的所有计算模块, 可重写 ϕ_α 为如下形式:

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

其中 m_i 为 $\boldsymbol{m}_\alpha$ 的第 i 个分量.

$\nabla\phi_\alpha$ 的面向数组计算过程

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

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

Remark

这是一种标量的表达式!

$\nabla\phi_\alpha$ 的面向数组计算过程

用面向数组的方式，需要首先构造 $d+1$ 阶矩阵

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

进而需要构造矩阵 $\boldsymbol{D}$，其元素定义如下

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

$\nabla\phi_\alpha$ 的面向数组计算过程

进而可以计算 $\nabla\boldsymbol{B}$

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

LagrangeFiniteElementSapce 类

```python
class LagrangeFiniteElementSpace():
    def __init__(self, mesh, p=1, spacetype='C'):
        self.mesh = mesh
        self.p = p
        if spacetype is 'C':
            if mesh.meshtype is 'interval':
                self.dof = CPLFEMDof1d(mesh, p)
                self.TD = 1 # toplogy dimension
            elif mesh.meshtype is 'tri':
                self.dof = CPLFEMDof2d(mesh, p)
                self.TD = 2
            elif mesh.meshtype is 'stri':
                self.dof = CPLFEMDof2d(mesh, p)
                self.TD = 2
            elif mesh.meshtype is 'tet':
                self.dof = CPLFEMDof3d(mesh, p)
                self.TD = 3
        elif spacetype is 'D':
            if mesh.meshtype is 'interval':
                self.dof = DPLFEMDof1d(mesh, p)
                self.TD = 1
            elif mesh.meshtype is 'tri':
                self.dof = DPLFEMDof2d(mesh, p)
                self.TD = 2
            elif mesh.meshtype is 'tet':
                self.dof = DPLFEMDof3d(mesh, p)
                self.TD = 3
```

LagrangeFiniteElementSapce 类

```
def basis(self, bc):
    p = self.p   # The degree of polynomial basis function
    TD = self.TD # The toplogy dimension of the mesh
    multiIndex = self.dof.multiIndex # The multiindex matrix of $m_\alpha$

    # construct vector $P = (\frac{1}{1!}, \frac{1}{2!}, \cdots, \frac{1}{p!})$.
    c = np.arange(1, p+1, dtype=np.int)
    P = 1.0/np.multiply.accumulate(c)

    # construct the matrix 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)

    # construct matrix B and here we still use the memory of A
    np.cumprod(A, axis=-2, out=A)
    A[..., 1:, :] *= P.reshape(-1, 1)

    # compute $\phi_\alpha$
    idx = np.arange(TD+1)
    phi = np.prod(A[..., multiIndex, idx], axis=-1)
    return phi
```

LagrangeFiniteElementSapce 类

```python
def grad_basis(self, bc):
    # construct the matrix A ...
    # construct the matrix F
    FF = np.einsum('...jk, m->...kjm', A[..., 1:, :], np.ones(p))
    FF[..., range(p), range(p)] = p
    np.cumprod(FF, axis=-2, out=FF)
    F = np.zeros(shape, dtype=self.ftype)
    F[..., 1:, :] = np.sum(np.tril(FF), axis=-1).swapaxes(-1, -2)
    F[..., 1:, :] *= P.reshape(-1, 1)
    # construct matrix B and here we still use the memory of A ...
    Q = A[..., multiIndex, range(TD+1)]
    M = F[..., multiIndex, range(TD+1)]
    ldof = self.number_of_local_dofs()
    shape = bc.shape[:-1]+(ldof, TD+1)
    R = np.zeros(shape, dtype=self.ftype)
    for i in range(TD+1):
        idx = list(range(TD+1))
        idx.remove(i)
        R[..., i] = M[..., i]*np.prod(Q[..., idx], axis=-1)
    Dlambda = self.mesh.grad_lambda()
    gphi = np.einsum('...ij, kjm->...kim', R, Dlambda)
    return gphi
```

刚度矩阵组装

```python
def stiff_matrix(space, qf, cellmeasure):
    bcs, ws = qf.quadpts, qf.weights
    gphi = space.grad_basis(bcs)
    A = np.einsum('i, ijkm, ijpm, j->jkp', ws, gphi, gphi, cellmeasure)

    cell2dof = space.cell_to_dof()
    ldof = space.number_of_local_dofs()
    I = np.einsum('k, ij->ijk', np.ones(ldof), cell2dof)
    J = I.swapaxes(-1, -2)

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

质量矩阵组装

```python
def mass_matrix(space, qf, cellmeasure):
    bcs, ws = qf.quadpts, qf.weights
    phi = space.basis(bcs)
    M = np.einsum('m, mj, mk, i->ijk', ws, phi, phi, cellmeasure)

    cell2dof = space.cell_to_dof()
    ldof = space.number_of_local_dofs()
    I = np.einsum('k, ij->ijk', np.ones(ldof), cell2dof)
    J = I.swapaxes(-1, -2)

    gdof = space.number_of_global_dofs()
    M = csr_matrix((M.flat, (I.flat, J.flat)), shape=(gdof, gdof))
    return M
```

载荷向量组装

```python
def source_vector(f, space, qf, cellmeasure):
    bcs, ws = qf.quadpts, qf.weights
    pp = space.mesh.bc_to_point(bcs)
    fval = f(pp)
    phi = space.basis(bcs)
    b = np.einsum('i, ij, ik, k->kj', ws, phi, fval, cellmeasure)

    cell2dof = space.cell_to_dof()
    gdof = space.number_of_global_dofs()
    b = np.bincount(cell2dof.flat, weights=b.flat, minlength=gdof)
    return b
```

Outline