# 基于 MATLAB 的有限元程序设计

Terence Yu

School of Mathematical Sciences, Institute of Natural Sciences, MOE-LSC, Shanghai Jiao Tong University, Shanghai, 200240, P. R. China.

# Outline

# 编程语言的选择

| 语言 | 建议对象 | 线性代数的实现 |
|---|---|---|
| MATLAB | 科学研究 | 矩阵 |
| Python | 人工智能领域 | NumPy (多维数组) |
| C++ | 软件开发 | Eigen/Armadillo/NumCPP |

- C 语言
- Fortran
- Octave: MATLAB 的开源替代品

**关于矩阵库的建议**

- 切勿花精力去实现矩阵库，这是个浩大的工程. 专业软件一般经历过各种加速，自编程序很难超越.
- 对 C 或 C++, 一般可建立自己的矩阵容器，并实现与已有库的数据转化.

# FEM 软件/工具箱

1. MATLAB
   - iFEM (陈龙, mFEM 的主要编程思路均来自该工具箱)
   - pdetool $\Longrightarrow$ FEATool Multiphysics $\Longrightarrow$ COMSOL Multiphysics
2. Python: FEALPy (魏华祎), FEniCS
3. C++: FreeFEM (基于变分形式的程序设计, varFEM)
4. C: FASP (许进超, 多重网格法)
5. Fortran: FEPG (梁国平, FELAC (C,C++?))

# FEM 编程的难点

1. 网格的生成和数据结构
2. 刚度矩阵和载荷向量的装配
3. 数值积分

# 主程序结构预览

```
%% Parameters

%% Generate an initial mesh
[node,elem] = squaremesh([a1 b1 a2 b2],h1,h2);
bdNeumann = 'abs(x-1)<1e-4';

%% Get the PDE data
pde = Poissondata();

%% Finite element method
for k = 1:maxIt
    % refine mesh
    [node,elem] = uniformrefine(node,elem);
    % set boundary
    bdStruct = setboundary(node,elem,bdNeumann);
    % solve the equation
    uh = Poisson(node,elem,pde,bdStruct,option);
    % record
    % compute error
end

%% Plot convergence rates and display error table

%% Conclusion
```

# 数据结构与几何量

1. 网格的生成
2. 数据结构: `auxT = auxstructure(node,elem);`

Table: 数据结构

| | |
|---|---|
| `node, elem` | 基本数据结构 |
| `elem2edge` | 边的自然序号 (单元存储) |
| `edge` | 一维边的端点标记 |
| `bdEdge` | 边界边的端点标记 |
| `edge2elem` | 边的左右单元 |
| `neighbor` | 目标单元边的相邻单元 |

3. 几何量: `aux = auxgeometry(node,elem);`

Table: 几何量

| | |
|---|---|
| `centroid` | 单元重心坐标 |
| `area` | 单元面积 |
| `diameter` | 单元直径 |

# 装配函数

1. 刚度矩阵: `A = sparse(ii,jj,ss, NNdof,NNdof);`
2. 载荷向量: `F = accumarray(subs,vals, [NNdof 1]);`

- 注意这两个函数的 summation property.
- 装配和计算要尽量使用向量化运算, 以避免循环 (特别是单元循环).
- 称二元组 `[ii,jj]` 为稀疏装配指标.

# 装配规则

- 数值解按单指标排列为: $\boldsymbol{u} = (u_1, u_2, \cdots, u_{NNdof})^T$ (顶点+边+单元).
- 单元刚度矩阵为

$$
[K^e] = \begin{array}{c c} & \begin{array}{ccc} u_i & u_j & u_l \end{array} \\ \begin{array}{c} u_i \\ u_j \\ u_l \end{array} & \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \end{array}
$$

将矩阵“行拉直”排列

```
K = [k11,k12,k13,k21,k22,k23,k31,k32,k33]; % straighten
ss = K(:);
```

这里 $k_{11}$ 是所有单元刚度矩阵 $(1,1)$ 位置的结果.

- 三元组

$$
\begin{bmatrix} \boldsymbol{i}_{11} & \boldsymbol{j}_{11} & \boldsymbol{s}_{11} \\ \boldsymbol{i}_{12} & \boldsymbol{j}_{12} & \boldsymbol{s}_{12} \\ \vdots & \vdots & \vdots \\ \boldsymbol{i}_{33} & \boldsymbol{j}_{33} & \boldsymbol{s}_{33} \end{bmatrix}
$$

这里 $\boldsymbol{i}_{11}$ 是所有单元刚度矩阵 $(1,1)$ 位置的结果.

# 稀疏装配指标和刚度矩阵的装配

- 设

```
elem2dof = [ui, uj, ul]
```

这里 `ui` 是所有单元 $u_i$ 对应的整体指标, 称其为局部整体对应.

- 稀疏装配指标如下

```
Ndof = 3; NT = size(elem,1);
id = 0;
for i = 1:Ndof
    for j = 1:Ndof
        ii(id+1:id+NT) = elem2dof(:,i);   % ui
        jj(id+1:id+NT) = elem2dof(:,j);   % uj
        id = id + NT;
    end
end
```

- 上述程序容易理解，它也可写为如下程序的前两行

```
ii = reshape(repmat(elem2dof, Ndof,1), [], 1);
jj = repmat(elem2dof(:), Ndof, 1);
kk = sparse(ii,jj,K(:),NNdof,NNdof);
```

最后一行进行刚度矩阵装配.

# 载荷向量的装配

设单元上的载荷向量为

$$[F^e] = \begin{array}{c} u_i \\ u_j \\ u_l \end{array} \begin{pmatrix} F_1 \\ F_2 \\ F_3 \end{pmatrix}$$

- 将向量“行拉直”

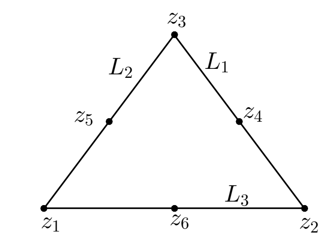
```
F = [F1, F2, F3];
```

这里 F1 是所有单元的结果.

- 载荷向量如下装配

```
ff = accumarray(elem2dof(:), F(:),[NNdof 1]);
```

# P2-Lagrange 元的局部整体对应




P2-Lagrange

- 整体自由度先排列网格节点值, 接着排列边中点值.
- `elem2dof = [elem, elem2edge+N];`

# P3-Lagrange 元的局部整体对应

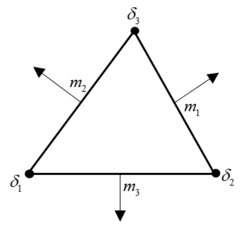
P3-Lagrange 元的局部自由度有 10 个, 排列为:

$$
\begin{cases}
v(z_i), & i = 1, 2, 3; \\
v(a_i), & i = 1, 2, 3; \\
v(b_i), & i = 1, 2, 3; \\
v(z_c).
\end{cases}
$$

这里, $a_i$ 第 $i$ 条边的 1/3 点, $b_i$ 则是 2/3 点, 而 $z_c$ 是单元的重心. 整体自由度类似排列, 但要注意边必须事先规定好方向.

```
% sgnelem
v1 = [2 3 1]; v2 = [3 1 2];
bdEdgeIdx = bdStruct.bdEdgeIdx; E = false(NE,1); ...
    E(bdEdgeIdx) = 1;
sgnelem = sign(elem(:,v2)-elem(:,v1));
sgnbd = E(elem2edge);    sgnelem(sgnbd) = 1;
sgnelem(sgnelem==-1) = 0;
elema = elem2edge + N*sgnelem + (N+NE)*(¬sgnelem); %1/3
elemb = elem2edge + (N+NE)*sgnelem + N*(¬sgnelem); %2/3
% local --> global
elem2dof = [elem, elema, elemb, (1:NT)'+N+2*NE];
```

# 符号刚度矩阵和载荷向量 (先跳过 )




$$a_K(\pm\varphi_i, \pm\varphi_j) = \pm \cdot \pm a_K(\varphi_i, \varphi_j).$$

- 边的符号

```
z1 = elem(:,1); z2 = elem(:,2); z3 = elem(:,3);
sgnedge = sign([z3-z2, z1-z3, z2-z1]);
```

- 边界边定向的恢复

```
bdEdgeIdx = bdStruct.bdEdgeIdx;
E = false(NE,1); E(bdEdgeIdx) = 1;
sgnbd = E(elem2edge);
sgnedge(sgnbd) = 1;
```

所有单元局部基的符号 (Morley 元前三个无符号)

```
sgnbase = ones(NT,Ndof); sgnbase(:,4:6) = sgnedge;
```

符号刚度矩阵和载荷向量为 (“拉直” 存储)

```
sgnK = zeros(NT,Ndof^2);
s = 1;
for i = 1:Ndof
    for j = 1:Ndof
        sgnK(:,s) = sgnbase(:,i).*sgnbase(:,j);
        s = s+1;
    end
end

sgnF = ones(NT,Ndof); sgnF(:,4:6) = sgnedge;
```

最终的刚度矩阵和载荷向量为

```
kk = sparse(ii,jj,K(:).*sgnK(:),NNdof,NNdof);
ff = accumarray(elem2dof(:), F(:).*sgnF(:), [NNdof 1]);
```

# 三角形上的 Gauss 求积

- 面积坐标下参考三角形 $\hat{T}$ 上的积分公式为

$$\iint_{\hat{T}} f(\lambda_1,\lambda_2,\lambda_3)\mathrm{d}\hat{\sigma} \approx |\hat{T}| \sum_{p=1}^{n_g} w_p f(\lambda_{1,p},\lambda_{2,p},\lambda_{3,p}), \quad |\hat{T}| = \frac{1}{2},$$

式中,

$$\begin{cases} x = x_1\lambda_1 + x_2\lambda_2 + x_3\lambda_3 \\ y = y_1\lambda_1 + y_2\lambda_2 + y_3\lambda_3 \end{cases}, \qquad \lambda_1+\lambda_2+\lambda_3 = 1, \tag{4.1}$$

且

$$\det\left(\frac{\partial(x,y)}{\partial(\lambda_1,\lambda_2)}\right) = 2S,$$

这里 $S$ 是三角形 $T$ 的代数面积.

- 积分公式为

$$\int_T F(x,y)\mathrm{d}\sigma = 2\,|T| \int_{\hat{T}} f(\lambda_1,\lambda_2,\lambda_3)\mathrm{d}\hat{\sigma} = |T| \sum_{p=1}^{n_g} w_p f(\lambda_{1,p},\lambda_{2,p},\lambda_{3,p}).$$

- 因变换前后点处的值不变, 故

$$\int_T F(x,y)\mathrm{d}\sigma = |T| \sum_{p=1}^{n_g} w_p F(x_p,y_p). \tag{4.2}$$

利用 (4.1) 可把参考元上的 Gauss 点 $(\lambda_{1,p},\lambda_{2,p},\lambda_{3,p})$ 转化为 $T$ 上的点.

# iFEM 中的 quadpts.m 函数

iFEM 中用 quadpts.m 返回参考三角上的求积节点和权重

```
[lambda,weight] = quadpts(order);
```

这里

- `lambda` $= [\lambda_1, \lambda_2, \lambda_3]$, 而 $\lambda_1$ 是所有积分节点处的列向量;
- `weight` 是一行向量.
- 第 $p$ 个积分节点和权重为

```
lambda(p,1),   lambda(p,2),   lambda(p,3),    weight(p)
```

iFEM 中的权重未统一为列向量或行向量.

# iFEM 中的实现

- 为了避免单元循环, 先计算出 (4.2) 求和项中的每一项, 再把各和项相加.
- 第 $p$ 个和项所有单元的坐标记为 pxy, 如下计算

```
% quadrature points in the x-y coordinate
pxy = lambda(p,1)*node(elem(:,1),:) ...
    + lambda(p,2)*node(elem(:,2),:) ...
    + lambda(p,3)*node(elem(:,3),:);
```

- Poisson 方程 P1 元的载荷向量如下计算

```
F = zeros(NT,3);
for p = 1:size(lambda,1)
    % quadrature points in the x-y coordinate
    pxy = lambda(p,1)*node(elem(:,1),:) ...
        + lambda(p,2)*node(elem(:,2),:) ...
        + lambda(p,3)*node(elem(:,3),:);
    % [f*phi1, f*phi2, f*phi3] at (xp,yp)
    fxy = f(pxy); fv = fxy*lambda(p,:);
    F = F + weight(p)*fv;
end
F = area.*F;
ff = accumarray(elem(:), F(:),[N 1]);
```

# Neumann 边界条件

用梯形公式近似

$$\int_{\Gamma^e} \frac{\partial u}{\partial n}\phi_1 \mathrm{d}s = \frac{h_e}{2}\left(\frac{\partial u}{\partial n}(z_1)\phi_1(z_1) + \frac{\partial u}{\partial n}(z_2)\phi_1(z_2)\right) = \frac{h_e}{2}\frac{\partial u}{\partial n}(z_1),$$

$$\int_{\Gamma^e} \frac{\partial u}{\partial n}\phi_2 \mathrm{d}s = \frac{h_e}{2}\left(\frac{\partial u}{\partial n}(z_1)\phi_2(z_1) + \frac{\partial u}{\partial n}(z_2)\phi_2(z_2)\right) = \frac{h_e}{2}\frac{\partial u}{\partial n}(z_2),$$

式中,

$$h_e = \left|z_j - z_i\right| = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}.$$

注意,

$$h_e\frac{\partial u}{\partial n} = h_e \nabla u \cdot \vec{n} = \nabla u \cdot (h_e\vec{n}),$$

而 $\hat{n}_e = h_e\vec{n}$ 可通过三角形的定向边逆时针旋转 $90^\circ$ 获得 (绕着定向边的终点) , 即

```
z1 = node(bdEdgeN(:,1),:); z2 = node(bdEdgeN(:,2),:);
e = z1-z2; % e = z2-z1
ne = [-e(:,2),e(:,1)]; % scaled ne
```

注意, setboundary.m 函数给出的边界边已经定向过.

Neumann 边界条件如下处理

```
%% Assemble Neumann boundary conditions
bdEdgeN = bdStruct.bdEdgeN;
if ¬isempty(bdEdgeN)
    z1 = node(bdEdgeN(:,1),:); z2 = node(bdEdgeN(:,2),:);
    e = z1-z2; % e = z2-z1
    ne = [-e(:,2),e(:,1)]; % scaled ne
    Du = pde.Du;
    gradu1 = Du(z1); gradu2 = Du(z2);
    F1 = sum(ne.*gradu1,2)./2; F2 = sum(ne.*gradu2,2)./2;
    FN = [F1,F2];
    ff = ff + accumarray(bdEdgeN(:), FN(:),[N 1]);
end
```

这里为了方便, 用精确解计算出 Neumann 边界条件的函数值.

# Dirichlet 边界条件

恒等式法

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 6.2222 & -2.9444 & 0 \\ 0 & -2.9444 & 6.2222 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix}
=
\begin{bmatrix} u(0) \\ 0.1111 + 2.9444u_0 \\ 0.2222 + 2.9444u_3 \\ u(1) \end{bmatrix},
$$

$$
\begin{bmatrix} 6.2222 & -2.9444 \\ -2.9444 & 6.2222 \end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}
=
\begin{bmatrix} 0.1111 + 2.9444u_0 \\ 0.2222 + 2.9444u_3 \end{bmatrix}.
$$

```
%% Apply Dirichlet boundary conditions
% NNdof = N
bdNodeIdx = bdStruct.bdNodeIdx; g_D = pde.g_D;
isBdNode = false(N,1); isBdNode(bdNodeIdx) = true;
bdDof = (isBdNode); freeDof = (¬isBdNode);
nodeD = node(bdDof,:);
u = zeros(N,1); u(bdDof) = g_D(nodeD);
ff = ff - kk*u;
```

必须最后处理 Dirichlet 边界条件.

# 有限元求解器

1. 自适应方法

Jump to sgnK and sgnF

2. 多重网格法


THANK YOU