智能可视建模与仿真实验室

Intelligent Visual Modeling & Simulation (iGame) Lab

基于等几何分析的泊松问题求解

姓 名：徐 岗

杭州电子科技大学 计算机学院

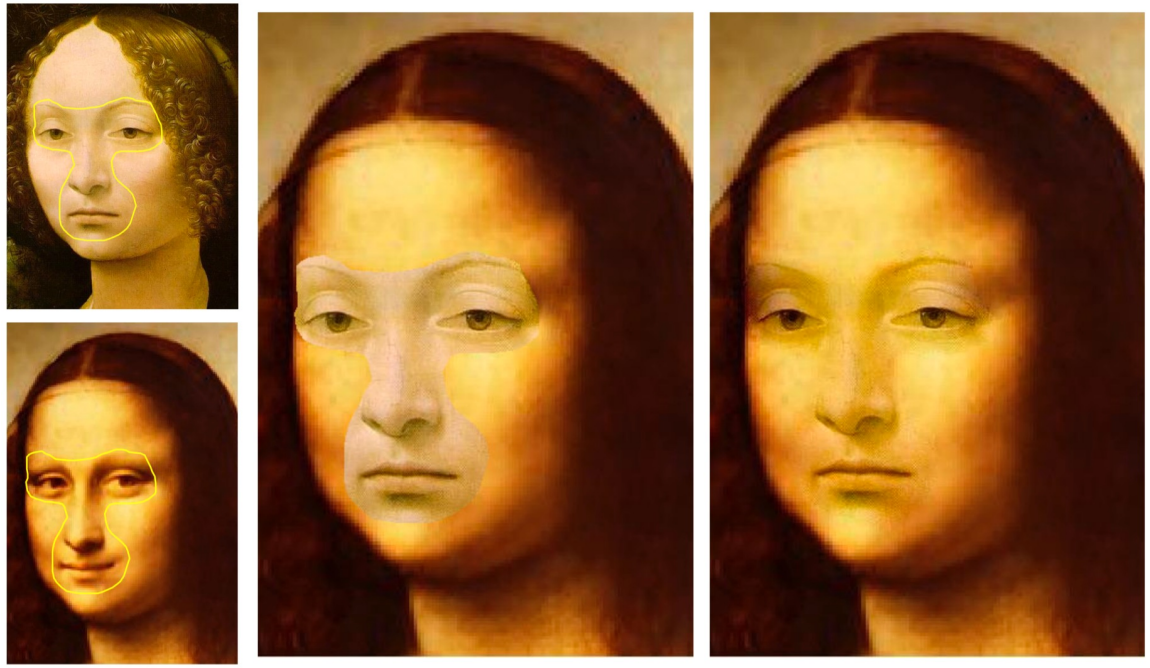
电子邮箱：gxu@hdu.edu.cn

目 录

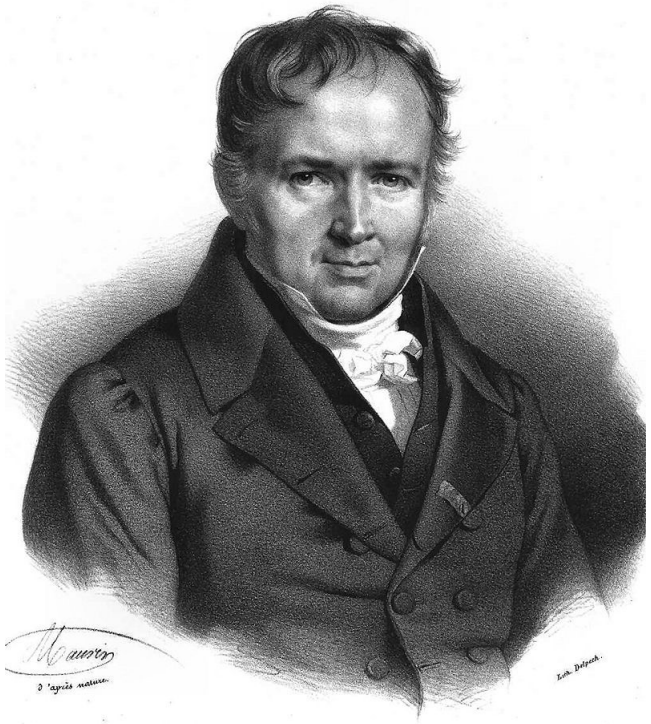
1.泊松方程（热传导方程）

热传导方程的强形式

- 泊松方程是數學中一個常見於靜電學、機械工程和理論物理的偏微分方程。是因法國數學家、幾何學家及物理學家泊松而得名的。
- 泊松首先在無引力源的情況下得到泊松方程，△Φ=0（即拉普拉斯方程）；
- 當考慮引力場時，有△Φ=f（f為引力場的質量分佈）。
- 後推廣至電場磁場，以及熱場分佈。
- 該方程通常用格林函數法求解，也可以分離變量法，特徵線法求解。



source/destination　　cloning　　seamless cloning



热传导方程的强形式

- 考虑一个在NURBS几何体定义的域上的偏微分方程--热传导方程，该方程的强形式是

找到一个u : $\overline{\Omega} \rightarrow R$:使其满足：

$$-\nabla(K\nabla u) = f \quad in\ \Omega$$

$$u = g \quad on\ \Gamma_D$$

$$\frac{K\partial u}{\partial n} = h \ \ on\ \Gamma_N$$

$$\frac{K\partial u}{\partial n} = -\beta(u - u_R) \quad on\ \Gamma_R$$

其中 ：

u:温度场 u: $\overline{\Omega} \rightarrow R$

K 热传导率 K: $\overline{\Omega} \rightarrow R^+ (K > 0)$

f：热传导源函数f: $\overline{\Omega} \rightarrow R$

g：固定温度场g: $\Gamma_D \rightarrow R$

h：固定热通量h: $\Gamma_N \rightarrow R$

U_R：周围对流介质的温度U_R:$\Gamma_R \rightarrow R$

β：对流传热系数β: $\Gamma_R \rightarrow R^+$

Γ_D:迪利克雷边界

Γ_N:纽曼边界

Γ_R:罗宾条件

同时满足$\overline{\Gamma_D \cup \Gamma_N \cup \Gamma_R} = \Gamma = \partial\Omega$ 且 $\Gamma_D \cup \Gamma_N \cup \Gamma_R = \emptyset$

热传导方程的弱形式

- 当前目标是找到一个近似解即$u^h \approx u$ 。为定义弱形式的热传导方程，我们需要定义两组函数

1. 第一组函数是由试探解组成，从一开始，这些函数就需要满足强形式的迪利克雷边界。定义试探解的集合空间如下：

$$\boldsymbol{\Phi}:\{\boldsymbol{u}:\overline{\boldsymbol{\Omega}}\rightarrow \boldsymbol{R}\ ,\ \ \boldsymbol{u}\in \boldsymbol{H^1}(\boldsymbol{\Omega})\ 且\ \boldsymbol{u}|_{\Gamma_D}=\boldsymbol{g}\}$$

$$\boldsymbol{H^1}(\boldsymbol{\Omega}):\{\boldsymbol{u}:\overline{\boldsymbol{\Omega}}\rightarrow \boldsymbol{R}\ 且\ \boldsymbol{u},\frac{\boldsymbol{\partial u}}{\boldsymbol{\partial x}},\frac{\boldsymbol{\partial u}}{\boldsymbol{\partial y}}\in \boldsymbol{L^2}(\boldsymbol{\Omega})\}$$

$$\boldsymbol{L^2}(\boldsymbol{\Omega}):\{\boldsymbol{u}|\int_{\boldsymbol{\Omega}}\boldsymbol{u^2}\,\boldsymbol{d\Omega}<+\infty\}$$

可以看出，试探解是具有平方可积导数的平方可积函数，同时期望温度场u属于Φ

热传导方程的弱形式

2.第二组函数是测试函数。测试函数的集合与Φ非常相似，只是再定义：

$$\Upsilon：\{\omega:\overline{\Omega}\rightarrow R,\omega\in H^1(\Omega)\ 且\ \omega|_{\Gamma_D}=0\}$$

现在强形式方程两边同时乘以测试函数。积分后得到：

$$\int_{\Omega}(-\omega\nabla(K\nabla u))\,d\Omega=\int_{\Omega}\omega f\,d\Omega$$

将左边部分进行分部积分，结果是：

$$\int_{\Omega}(-\omega\nabla(K\nabla u))\,d\Omega=\int_{\Omega}K\nabla u\nabla\omega\,d\Omega-\int_{\Gamma}\omega\frac{K\partial u}{\partial n}d\Gamma$$

考虑条件 $\overline{\Gamma_D\cup\Gamma_N\cup\Gamma_R}=\Gamma=\partial\Omega$ 得到下式：

$$\int_{\Gamma}\omega\frac{K\partial u}{\partial n}d\Gamma=\int_{\Gamma_D}\omega\frac{K\partial u}{\partial n}d\Gamma+\int_{\Gamma_N}\omega\frac{K\partial u}{\partial n}d\Gamma+\int_{\Gamma_R}\omega\frac{K\partial u}{\partial n}d\Gamma$$

热传导方程的弱形式

- 考虑三种边界条件：

当$\omega|_{\Gamma_D}=0$，有

$$\int_{\Gamma_D}\omega\frac{K\partial u}{\partial n}d\Gamma=0$$

当$\frac{K\partial u}{\partial n}=h$，有

$$\int_{\Gamma_N}\omega\frac{K\partial u}{\partial n}d\Gamma=\int_{\Gamma_N}\omega h\,d\Gamma$$

当$\frac{K\partial u}{\partial n}=-\beta(u-u_R)$，有

$$\int_{\Gamma_R}\omega\frac{K\partial u}{\partial n}d\Gamma=-\int_{\Gamma_R}-\beta\omega(u-u_R)\,d\Gamma$$

将上述式子代入方程：

$$\int_{\Omega}K\nabla u\nabla\omega\,d\Omega+\int_{\Gamma_R}\beta\omega u\,d\Gamma=\int_{\Omega}\omega f\,d\Omega+\int_{\Gamma_N}\omega h\,d\Gamma+\int_{\Gamma_R}\beta\omega\,u_R d\Gamma$$

热传导方程的弱形式

- 根据之前的分析，可以给出热传导方程的弱形式如下：

找到$u\epsilon\Phi$，使其满足：

$$a(\omega,u)=L(\omega)\ \forall\omega\epsilon\Upsilon$$

其中

$$a(\omega,u)=\int_{\Omega}K\nabla u\nabla\omega\,d\Omega+\int_{\Gamma_R}\beta\omega u\,d\Gamma$$

$$L(\omega)=\int_{\Omega}\omega f\,d\Omega+\int_{\Gamma_N}\omega h\,d\Gamma+\int_{\Gamma_R}\beta\omega\,u_R d\Gamma$$

并且$a(\omega,u)$满足

$$a(\omega,u)=a(u,\omega)$$

$$a(\omega,u)>0\ \forall\omega\epsilon\Upsilon且\omega\neq 0$$

2.伽略金（Galerkin)方法

Galerkin's 方法

- 主要思想：构造有限维近似，无限维问题化为有限维问题

已有定义如下：

$$\boldsymbol{\Phi}: \left\{u: \overline{\Omega} \rightarrow R\ ,\ u \in H^1(\Omega) \text{ 且 } u|_{\Gamma_D} = g\right\}$$

$$\boldsymbol{\Upsilon}: \{\omega: \overline{\Omega} \rightarrow R, \omega \in H^1(\Omega) \text{ 且 } \omega|_{\Gamma_D} = 0\}$$

在等几何分析中，构造有限维近似，即满足：

$$\boldsymbol{\Phi}^h \subset \boldsymbol{\Phi}$$

$$\boldsymbol{\Upsilon}^h \subset \boldsymbol{\Upsilon}$$

故定义Φ^h 和 Υ^h在NURBS空间：

$$\boldsymbol{\Upsilon}^h : \{\omega^h \in \boldsymbol{\Upsilon}, \omega^h(x) = \sum_{i=1}^{n} N_i(x) c_i\}$$

$$\boldsymbol{\Phi}^h : \{u^h \in \boldsymbol{\Phi}, u^h(x) = \sum_{i=1}^{n} N_i(x) d_i\}$$

Galerkin's 方法

- **Galerkin's方法的表达如下：**

找寻 $\boldsymbol{u^h = v^h + g^h},\ \boldsymbol{v^h \in \Upsilon^h}$,即：

$$\boldsymbol{a(\omega^h, v^h) = L(\omega^h) - a(\omega^h, g^h)}$$

考虑到NURBS基函数是高度局部化的，因此在边界Γ_D很少有非零的函数，在不失一般性的情况下，假设存在一个整数$n_{eq} < n$，满足：

$$\boldsymbol{N_i|_{\Gamma_R} = 0 \quad \forall i = 1, \dots, n_{eq}}$$

故可以得到：

$$\boldsymbol{\Upsilon^h : \{\omega^h : \omega^h = \sum_{i=1}^{n_{eq}} N_i(x) c_i\}}$$

进一步，可以选择$\boldsymbol{g^h}$ 使得 $\boldsymbol{g_1 = \cdots = g_{n_{eq}} = 0}$:

$$\boldsymbol{g^h(x) = \sum_{i=n_{eq}+1}^{n} N_i(x) g_i}$$

Galerkin's 方法

将上述定义代入等式 $u^h = v^h + g^h$,得到：

$$u^h(x) = \sum_{i=1}^{n} N_i(x) d_i = \sum_{i=1}^{n_{eq}} N_i(x) d_i + \sum_{i=n_{eq}+1}^{n} N_i(x) g_i = v^h + g^h$$

将结果带入等式：

$$a\left(\sum_{i=1}^{n_{eq}} N_i c_i , \sum_{j=1}^{n_{eq}} N_j d_j\right) = L\left(\sum_{i=1}^{n_{eq}} N_i c_i\right) - a(\sum_{i=1}^{n_{eq}} N_i c_i , g^h)$$

进一步可以得到：

$$\sum_{i=1}^{n_{eq}} c_i \left(\sum_{j=1}^{n_{eq}} a(N_i, N_j) d_j - L(N_i) + a(N_i, g^h)\right) = 0$$

又因为c_i， $i = 1, \dots, n_{eq}$ 是任意常数,故：

$$\sum_{j=1}^{n_{eq}} a(N_i, N_j) d_j = L(N_i) - a(N_i, g^h) , \quad i = 1, \dots, n_{eq}$$

Galerkin's 方法

给出以下定义：

$$K_{ij} = a(N_i, N_j)$$

$$F_i = L(N_i) - a(N_i, g^h)$$

$$K = [K_{ij}]$$

$$F = [F_i]$$

$$d = [d_i]$$

由于有限元方法在结构有限元分析中的历史渊源，称

K : 刚度矩阵

d: 位移矢量

F: 力矢量

$$d = K^{-1}F$$

我们最终得到了温度场的理想离散近似值

$$u^h(x) = \sum_{i=1}^{n_{eq}} N_i(x)d_i + \sum_{i=n_{eq}+1}^{n} N_i(x)g_i$$

其中：d_i是需要求得的解，而g_i对应迪利克雷边界的数据

3.装配矩阵系统

装配矩阵系统

- 注意到，由于每个NURBS基函数都是高度局部化的，故刚度矩阵K是一个稀疏矩阵。因此对于K_{ij}矩阵的组装，可以将整个域上的积分写成仅对单元的积分，同时利用这一特性来减少构建和求解矩阵系统的工作量，

建立一个单元刚度矩阵$\boldsymbol{K^e}$

$$[\boldsymbol{K^e}]_{\boldsymbol{ab}} = \int_{\boldsymbol{\Omega^e}} \boldsymbol{K\nabla N_a^e\, \nabla N_b^e d\Omega^e} + \int_{\boldsymbol{\Gamma_R \cap \partial\Omega^e}} \boldsymbol{\beta N_a^e\, N_b^e d\Gamma^e}$$

单元力向量$\boldsymbol{f^e}$

$$[\boldsymbol{f^e}]_{\boldsymbol{a}} = \int_{\boldsymbol{\Omega^e}} \boldsymbol{N_a^e\, \mathrm{f} d\Omega^e} + \int_{\boldsymbol{\Gamma_N \cap \partial\Omega^e}} \boldsymbol{N_a^e\, h d\Gamma^e} + \int_{\boldsymbol{\Gamma_R \cap \partial\Omega^e}} \boldsymbol{\beta N_a^e\, u_R d\Gamma^e}$$

使用IEN数组更新全局矩阵

$$\boldsymbol{K_{ij}} \text{ 替换为 } \boldsymbol{K_{ij}} + \boldsymbol{K_{ab}^e}$$
$$\boldsymbol{F_i} \text{ 替换为 } \boldsymbol{F_i} + \boldsymbol{f_a^e}$$

其中$\boldsymbol{a, b = 1, \ldots, n_{loc}}$

$$\boldsymbol{i = IEN(a, e), j = IEN(b, e)}$$

装配矩阵系统

上述更新的一个例外是：基函数N_i, N_j在迪利克雷边界上为非零。为了处理此例外，引入使用BC函数

$$BC(i) = \begin{cases} 0, & N_i|_{\Gamma_D} = 0 \\ 1, & \text{其他} \end{cases}$$

如果$BC(i) = 1$，则改变策略：

$$K_{ii} = 1 \quad K_{ij} = 0, \ i \neq j$$
$$F_i = g$$

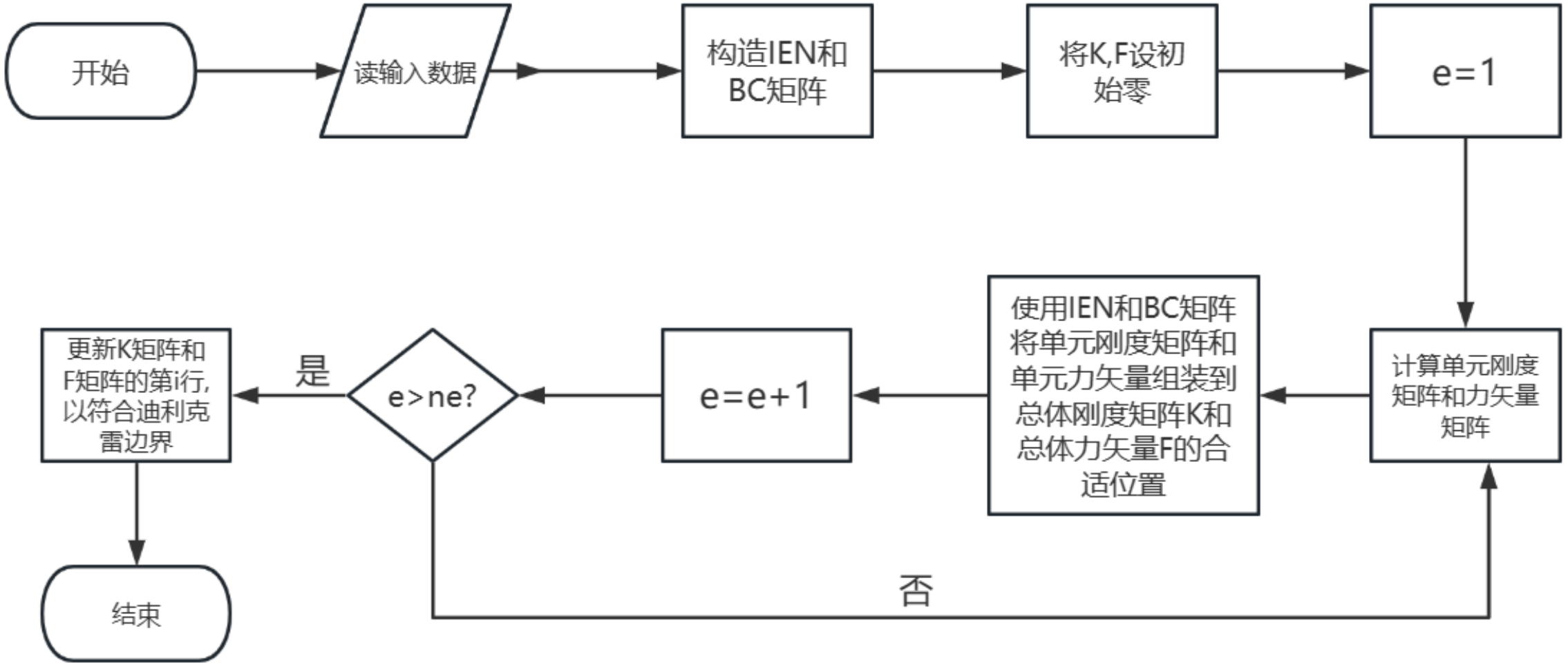
这对应于迪利克雷边界：$d_i = g_i$

如果在组装过程中，若遇到 i，j，$BC(i) = 0$和 $BC(j) = 1$，不更新K_{ij} ，而是考虑

$F_i - K_{ab}^e g_j$替代F_i

装配矩阵系统流程图

- 全局刚度矩阵K和力向量F的组装矩阵的总体算法如下



4.构造单元刚度矩阵和力向量

构造单元刚度矩阵和力向量

- 将$\Omega^e \subseteq \Omega$看作在物理域中随意的一个单元，使用逆映射$(x^e)^{-1}: \Omega^e \longrightarrow \tilde{\Omega}$ ，先定义以下概念：

雅可比矩阵：

$$J = \begin{bmatrix} \partial x/\partial \xi_1 & \partial x/\partial \xi_2 \\ \partial y/\partial \xi_1 & \partial y/\partial \xi_2 \end{bmatrix}$$

雅可比矩阵行列式：

$$j = det(J)$$

单位向外法向量：

$$\tilde{n}$$

单位切向量：

$$\tilde{t}$$

表面行列式：

$$j_\Gamma = |J\tilde{t}|$$

构造单元刚度矩阵和力向量

- 根据上述定义可以将Ω^e内积分转化为$\tilde{\Omega}$内积分，得到以下等式：

$$\int_{\Omega^e} K\nabla N_a^e \nabla N_b^e d\Omega^e = \int_{\tilde{\Omega}} K(x^e(\tilde{\xi}))(\nabla_x N_a^e x^e(\tilde{\xi}))(\nabla_x N_b^e x^e(\tilde{\xi}))\, j d\tilde{\Omega}$$

$$\int_{\Gamma_R \cap \Gamma^e} \beta N_a^e N_b^e d\Gamma^e = \int_{(x^e)^{-1}(\Gamma_R \cap \Gamma^e)} \beta(x^e(\tilde{\xi})) N_a^e x^e(\tilde{\xi}) N_b^e x^e(\tilde{\xi})\, j_\Gamma\, d\,\tilde{\Gamma}$$

$$\int_{\Omega^e} N_a^e \mathrm{f} d\Omega^e = \int_{\tilde{\Omega}} N_a^e x^e(\tilde{\xi}) \mathrm{f} x^e(\tilde{\xi})\, j d\tilde{\Omega}$$

$$\int_{\Gamma_N \cap \Gamma^e} N_a^e h d\Gamma^e = \int_{(x^e)^{-1}(\Gamma_N \cap \Gamma^e)} N_a^e x^e(\tilde{\xi}) h x^e(\tilde{\xi})\; j_\Gamma d\,\tilde{\Gamma}$$

$$\int_{\Gamma_R \cap \Gamma^e} \beta N_a^e u_R d\Gamma^e = \int_{(x^e)^{-1}(\Gamma_R \cap \Gamma^e)} \beta(x^e(\tilde{\xi})) N_a^e x^e(\tilde{\xi}) u_R x^e(\tilde{\xi})\; j_\Gamma d\,\tilde{\Gamma}$$

构造单元刚度矩阵和力向量

- **得到单元刚度矩阵和力向量表达式，如下：**

$$K_{ab}^{e} = \int_{\Omega^e} K\nabla N_a^e \nabla N_b^e d\Omega^e + \int_{\Gamma_R \cap \Gamma^e} \beta N_a^e N_b^e d\Gamma^e \quad a,b = 1,\ldots,n_{loc}$$

$$= \int_{\tilde{\Omega}} K\nabla_x N_a^e \nabla_x N_b^e \, j d\tilde{\Omega} + \int_{(x^e)^{-1}(\Gamma_R \cap \Gamma^e)} \beta N_a^e N_b^e \; j_\Gamma d\,\tilde{\Gamma}$$

$$f_a^e = \int_{\Omega^e} N_a^e \, \mathrm{f} d\Omega^e + \int_{\Gamma_N \cap \Gamma^e} N_a^e \, h d\Gamma^e + \int_{\Gamma_R \cap \Gamma^e} \beta N_a^e \, u_R d\Gamma^e \quad a = 1,\ldots,n_{loc}$$

$$= \int_{\tilde{\Omega}} N_a^e \mathrm{f} \, j d\tilde{\Omega} + \int_{(x^e)^{-1}(\Gamma_N \cap \Gamma^e)} N_a^e h \; j_\Gamma d\,\tilde{\Gamma} + \int_{(x^e)^{-1}(\Gamma_R \cap \Gamma^e)} \beta N_a^e u_R \; j_\Gamma d\,\tilde{\Gamma}$$

其中：

$$j = d\Omega^e / d\tilde{\Omega}$$
$$j_\Gamma = d\Gamma^e / d\tilde{\Gamma}$$

构造单元刚度矩阵和力向量

- 目前面临有两个问题：如何计算基函数和导数？如何计算$\tilde{\Omega}$单元的积分？

解决问题1：如何计算基函数和导数？
根据之前所学的b样条基函数与Bernstein基的关系，以及NURBS相关知识可以知道

$$\boldsymbol{R}^{\boldsymbol{e}}\left(\boldsymbol{x}^{\boldsymbol{e}}\left(\tilde{\boldsymbol{\xi}}\right)\right)=\frac{\boldsymbol{W}^{\boldsymbol{e}}\boldsymbol{N}^{\boldsymbol{e}}\left(\boldsymbol{x}^{\boldsymbol{e}}\left(\tilde{\boldsymbol{\xi}}\right)\right)}{\boldsymbol{w}^{\boldsymbol{e}}\left(\tilde{\boldsymbol{\xi}}\right)}=\frac{\boldsymbol{W}^{\boldsymbol{e}}\boldsymbol{C}^{\boldsymbol{e}}\boldsymbol{B}\left(\tilde{\boldsymbol{\xi}}\right)}{\boldsymbol{w}^{\boldsymbol{e}}\left(\tilde{\boldsymbol{\xi}}\right)}$$

其中：

$$W^{e}=\begin{bmatrix}w_{1}^{e} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & w_{n_{loc}}^{e}\end{bmatrix}$$

为了计算导数，可以使用链式求导法则：

$$\nabla_{x}N_{a}^{e}=(J^{-1})^{T}\nabla_{\tilde{\xi}}N_{a}^{e}$$
$$\nabla_{x}N_{b}^{e}=(J^{-1})^{T}\nabla_{\tilde{\xi}}N_{b}^{e}$$

$$\frac{\partial R^{e}}{\partial\xi_{i}}=W^{e}C^{e}\frac{\partial(B^{e}(\tilde{\xi})/w^{e}(\tilde{\xi}))}{\partial\xi_{i}}=W^{e}C^{e}(\frac{1}{w^{e}(\tilde{\xi})}\frac{\partial B^{e}(\tilde{\xi})}{\partial\xi_{i}}-\frac{\partial w^{e}(\tilde{\xi})}{\partial\xi_{i}}\frac{B^{e}(\tilde{\xi})}{(w^{e}(\tilde{\xi}))^{2}})$$

$$\frac{\partial R^{e}}{\partial x}=\frac{\partial R^{e}}{\partial\tilde{\xi}_{1}}\frac{\partial\tilde{\xi}_{1}}{\partial x}+\frac{\partial R^{e}}{\partial\tilde{\xi}_{2}}\frac{\partial\tilde{\xi}_{2}}{\partial x}$$
$$\frac{\partial R^{e}}{\partial y}=\frac{\partial R^{e}}{\partial\tilde{\xi}_{1}}\frac{\partial\tilde{\xi}_{1}}{\partial y}+\frac{\partial R^{e}}{\partial\tilde{\xi}_{2}}\frac{\partial\tilde{\xi}_{2}}{\partial y}$$

构造单元刚度矩阵和力向量

解决问题2：如何计算$\tilde{\Omega}$单元的积分，使用高斯积分法

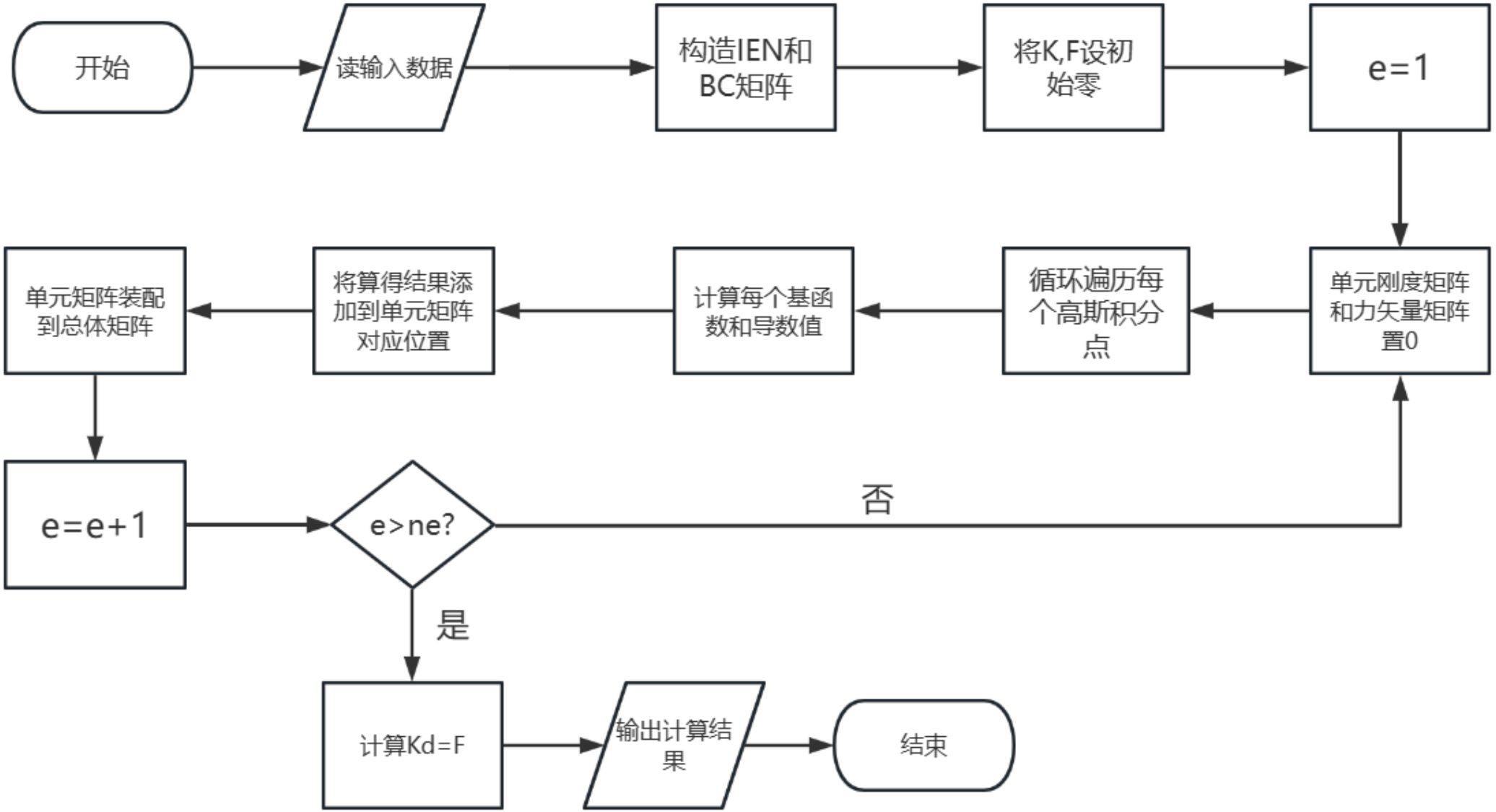
对于函数,可以在积分域适当选取某些节点，然后用相对应的函数值加权平均对应该函数的积分值，即

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

$$\int_{\tilde{\Omega}} f(\tilde{\xi})\, d\tilde{\Omega} \approx \sum_{i=1}^{n_i}\sum_{j=1}^{n_j} w_i w_j f(\tilde{\xi}_i, \tilde{\xi}_j)$$

阶次（M）	精度（N）	位置（x_i）	权重（w_i）
1	1	0	2
2	3	±0.577	1
3	5	0, ±0.775	0.889 (= 8/9), 0.556 (= 5/9)
4	7	±0.340, ±0.861	0.652, 0.348
5	9	0, ±0.538, ±0.906	0.569, 0.479, 0.237

完整算法流程图


开始
读输入数据
构造IEN和BC矩阵
将K,F设初始零
e=1
单元刚度矩阵和力矢量矩阵置0
循环遍历每个高斯积分点
计算每个基函数和导数值
将算得结果添加到单元矩阵对应位置
单元矩阵装配到总体矩阵
e=e+1
e>ne?
否
是
计算Kd=F
输出计算结果
结束

代码---流程

```cpp
#include<vector>
#include<Eigen/Dense>
#include <Eigen/Sparse>
#include "Surface.h"

typedef Eigen::SparseMatrix<double> SpMat;
```

```cpp
//1. 初始化数据
initial();

//2. 构造连接矩阵
buildConn();

//3. 构造A b 矩阵（Ax=b）
this->A = assembleA();
this->b = assembleb();

//4. 处理边界条件
exeBC();

//5. 稀疏矩阵求解
Eigen::SimplicialCholesky<SpMat> chol(A);
this->x = chol.solve(b);
```

代码---构造矩阵A

```
SpMat PDF_2d::assembleA() {
    SpMat A(basisNum, basisNum);
    A.setZero();
    //遍历单元
    for (int e = 0; e < elementNum; e++) {
        double area = getArea(e);
        //高斯积分点
        std::vector<Point> allGaussPoints = getGaussPoints(e);
        for (int i1 = 0; i1 <= uDegree; ++i1) {
            for (int j1 = 0; j1 <= vDegree; ++j1) {
                for (int i2 = 0; i2 <= uDegree; ++i2) {
                    for (int j2 = 0; j2 <= vDegree; ++j2) {
                        int count1 = j1 * (vDegree + 1) + i1;
                        int count2 = j2 * (vDegree + 1) + i2;
                        //使用连接矩阵组装矩阵
                        int row = INC[(IEN[count1][e])][0] + INC[(IEN[count1][e])][1] * uControl;
                        int col = INC[(IEN[count2][e])][0] + INC[(IEN[count2][e])][1] * uControl;
                        for (int p = 0; p < allGaussPoints.size(); ++p) {
                            double val = getIntegralA(i1, j1, i2, j2, allGaussPoints[p], e) * gauss.weights[p / 3] * gauss.weights[p % 3] * area;
                            A.coeffRef(row, col) = A.coeffRef(row, col) + val;
                        }
                    }
                }
            }
        }
    }
    return A;
}
```

代码---构造矩阵b

```
Eigen::VectorXd PDF_2d::assembleb() {
    Eigen::VectorXd b(basisNum);
    b.setZero();
    for (int e = 0; e < elementNum; e++) {
        double area = getArea(e);
        std::vector<Point> allGaussPoints = getGaussPoints(e);
        for (int j = 0; j <= vDegree; ++j) {
            for (int i = 0; i <= uDegree; ++i) {
                int count = j * (vDegree + 1) + i;
                int index = INC[(IEN[count][e])][0] + INC[(IEN[count][e])][1] * uControl;
                for (int p = 0; p < allGaussPoints.size(); ++p) {
                    b(index) = b(index) + getIntegralB(i, j, allGaussPoints[p], e) * gauss.weights[p / 3] * gauss.weights[p % 3];
                }
                b(index) *= area;
            }
        }
    }
    return b;
}
```

代码---A矩阵单元积分计算（1）

```
double PDF_2d::getIntegralA(int uTestFunc, int vTestFunc, int uProbeFunc, int vProbeFunc, Point gaussPoint, int elementNum) {
    //计算p+1个函数与导数值
    std::vector<std::vector<double>> uDers(2, std::vector<double>(uDegree + 1));
    std::vector<std::vector<double>> vDers(2, std::vector<double>(vDegree + 1));
    int uSpan = surface.m_Basis[0].findSpan(gaussPoint[0]);
    int vSpan = surface.m_Basis[1].findSpan(gaussPoint[1]);
    surface.m_Basis[0].getAllNurbsBasisFunsDers(uSpan, gaussPoint[0], 1, uDers);
    surface.m_Basis[1].getAllNurbsBasisFunsDers(vSpan, gaussPoint[1], 1, vDers);

    std::vector<std::vector<double>> R, DRU, DRV;
    R.resize(uDegree + 1, std::vector<double>(vDegree + 1));
    DRU.resize(uDegree + 1, std::vector<double>(vDegree + 1));
    DRV.resize(uDegree + 1, std::vector<double>(vDegree + 1));
    double sumR = 0, sumDRU = 0, sumDRV = 0;

    for (int i = 0; i < uDegree + 1; ++i) {
        for (int j = 0; j < vDegree + 1; ++j) {
            R[i][j] = uDers[0][i] * vDers[0][j];
            DRU[i][j] = uDers[1][i] * vDers[0][j];
            DRV[i][j] = uDers[0][i] * vDers[1][j];
            sumR += R[i][j];
            sumDRU += DRU[i][j];
            sumDRV += DRV[i][j];
        }
    }

    for (int i = 0; i < uDegree + 1; ++i) {
        for (int j = 0; j < vDegree + 1; ++j) {
            R[i][j] /= sumR;
            DRU[i][j] = (sumR * DRU[i][j] - R[i][j] * sumDRU) / (pow(sumR, 2));
            DRV[i][j] = (sumR * DRV[i][j] - R[i][j] * sumDRV) / (pow(sumR, 2));
        }
    }
```

代码---A矩阵单元积分计算（2）

```
//计算雅可比矩阵
double DF[2][2] = { 0,0,0,0 };
int count = 0;
for (int i = 0; i < uDegree + 1; ++i) {
    for (int j = 0; j < vDegree + 1; ++j) {
        int index = INC[(IEN[count][elementNum])][0] + INC[(IEN[count][elementNum])][1] * uControl;
        DF[0][0] += DRU[i][j] * controlPoints[index][0];
        DF[1][0] += DRU[i][j] * controlPoints[index][1];
        DF[0][1] += DRV[i][j] * controlPoints[index][0];
        DF[1][1] += DRV[i][j] * controlPoints[index][1];
        count++;
    }
}
Eigen::MatrixXd DF_(2, 2);
DF_ << DF[0][0], DF[0][1], DF[1][0], DF[1][1];
DF_ = DF_.inverse().transpose();
Eigen::MatrixXd gradR1(2, 1);
Eigen::MatrixXd gradR2(2, 1);

gradR1 << DRU[uTestFunc][vTestFunc], DRV[uTestFunc][vTestFunc];
gradR2 << DRU[uProbeFunc][vProbeFunc], DRV[uProbeFunc][vProbeFunc];
Eigen::MatrixXd val_ = (DF_ * gradR2).transpose() * (DF_ * gradR1) * abs(DF[0][0] * DF[1][1] - DF[0][1] * DF[1][0]);
double val = val_(0, 0);
return val;
```

代码---b矩阵单元积分计算（1）

```
double PDF_2d::getIntegralB(int uLocalBasisNum, int vLocalBasisNum, Point gaussPoint, int elementNum) {

    //计算p+1个函数与导数值
    std::vector<std::vector<double>> uDers(2, std::vector<double>(uDegree + 1));
    std::vector<std::vector<double>> vDers(2, std::vector<double>(vDegree + 1));
    int uSpan = surface.m_Basis[0].findSpan(gaussPoint[0]);
    int vSpan = surface.m_Basis[1].findSpan(gaussPoint[1]);
    surface.m_Basis[0].getAllNurbsBasisFunsDers(uSpan, gaussPoint[0], 1, uDers);
    surface.m_Basis[1].getAllNurbsBasisFunsDers(vSpan, gaussPoint[1], 1, vDers);
    std::vector<std::vector<double>> R, DRU, DRV;
    R.resize(uDegree + 1, std::vector<double>(vDegree + 1));
    DRU.resize(uDegree + 1, std::vector<double>(vDegree + 1));
    DRV.resize(uDegree + 1, std::vector<double>(vDegree + 1));
    double sumR = 0, sumDRU = 0, sumDRV = 0;

    for (int i = 0; i < uDegree + 1; ++i) {
        for (int j = 0; j < vDegree + 1; ++j) {
            R[i][j] = uDers[0][i] * vDers[0][j];
            DRU[i][j] = uDers[1][i] * vDers[0][j];
            DRV[i][j] = uDers[0][i] * vDers[1][j];
            sumR += R[i][j];
            sumDRU += DRU[i][j];
            sumDRV += DRV[i][j];
        }
    }

    for (int i = 0; i < uDegree + 1; ++i) {
        for (int j = 0; j < vDegree + 1; ++j) {
            R[i][j] /= sumR;
            DRU[i][j] = (sumR * DRU[i][j] - R[i][j] * sumDRU) / (pow(sumR, 2));
            DRV[i][j] = (sumR * DRV[i][j] - R[i][j] * sumDRV) / (pow(sumR, 2));
        }
    }
```

代码---b矩阵单元积分计算（2）

```
//计算雅可比矩阵
double DF[2][2] = { 0,0,0,0 };
int count = 0;
for (int i = 0; i < uDegree + 1; ++i) {
    for (int j = 0; j < vDegree + 1; ++j) {
        int index = INC[(IEN[count][elementNum])][0] + INC[(IEN[count][elementNum])][1] * uControl;
        DF[0][0] += DRU[i][j] * controlPoints[index][0];
        DF[1][0] += DRU[i][j] * controlPoints[index][1];
        DF[0][1] += DRV[i][j] * controlPoints[index][0];
        DF[1][1] += DRV[i][j] * controlPoints[index][1];
        count++;
    }
}
//获得面上点
Point P;
surfacePoint(P, gaussPoint);
double val = f(P[0], P[1]) * R[uLocalBasisNum][vLocalBasisNum] * abs(DF[0][0] * DF[1][1] - DF[0][1] * DF[1][0]);
return val;
```

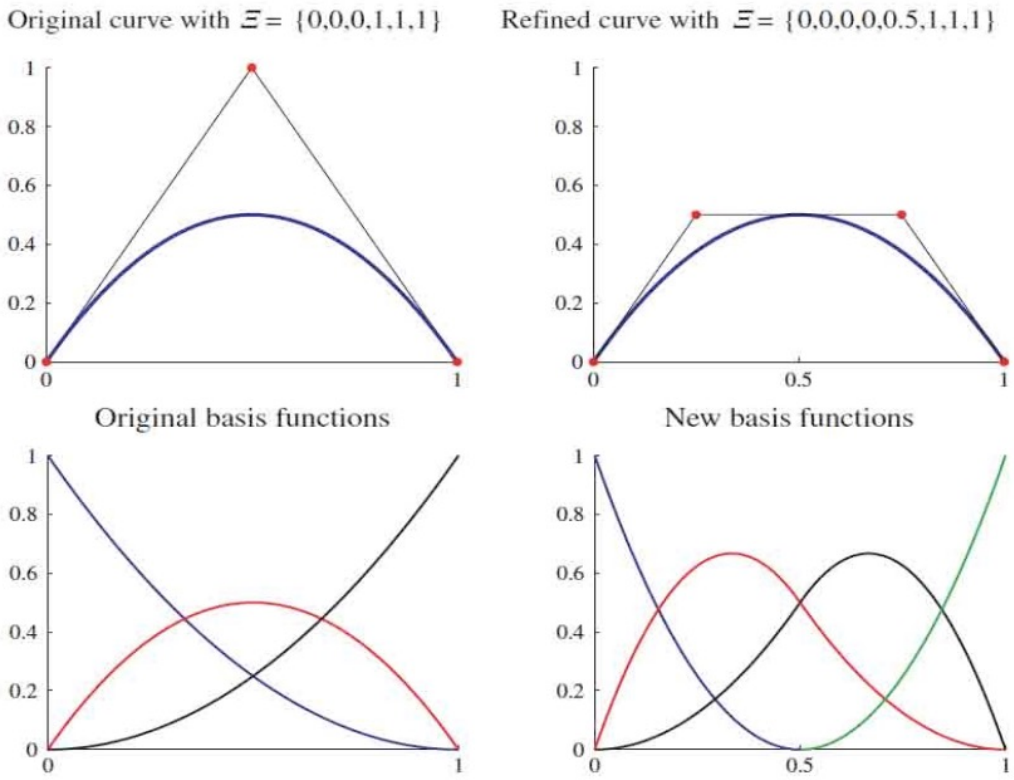
代码---迪利克雷边界条件处理

```
void PDF_2d::exeBC() {

    //下边为迪利克雷边界
    if (boundaryConditions.at(0) == 1) {
        for (int i = 0; i < uControl; ++i) {
            int x = i;
            for (int j = 0; j < basisNum; ++j) {
                A.coeffRef(x, j) = 0;
                A.coeffRef(j, x) = 0;
            }
            A.coeffRef(x, x) = 1;
            b(x) = 0;
        }
    }
    //右边为迪利克雷边界
    if (boundaryConditions.at(1) == 1){ ... }

    //上边为迪利克雷边界
    if (boundaryConditions.at(2) == 1){ ... }

    //左边为迪利克雷边界
    if (boundaryConditions.at(3) == 1){ ... }
}
```

目 录

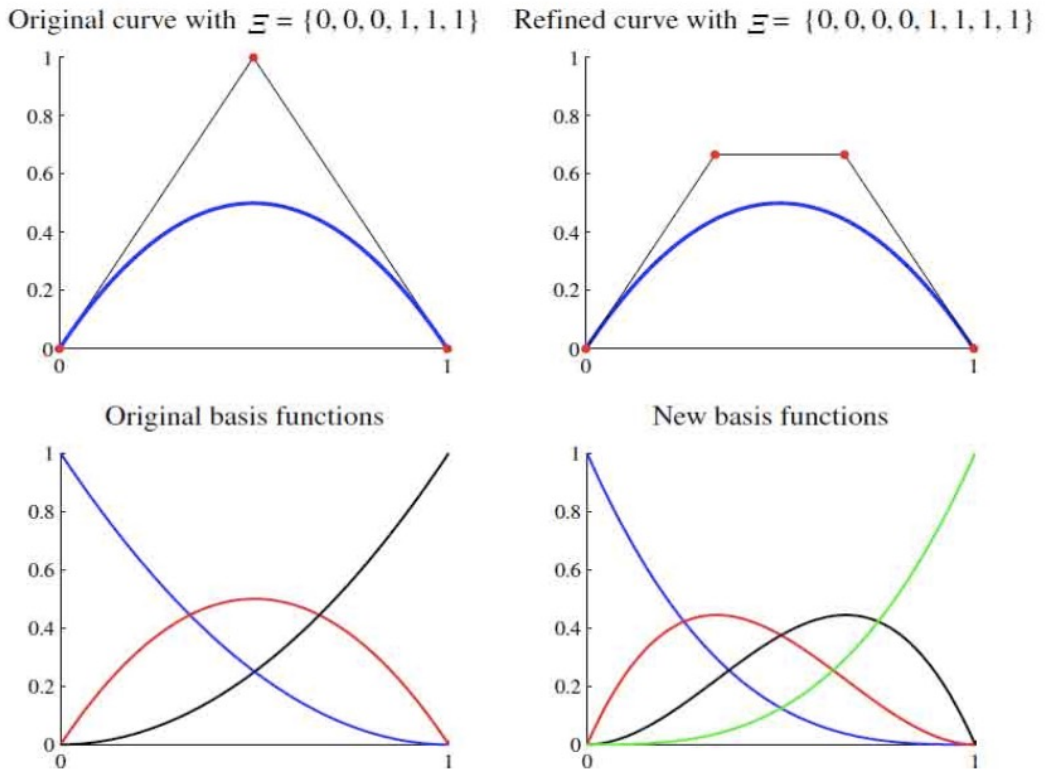
h-refinement

- knot insertion: adding a new knot into the existing knot vector without changing the shape of curve/surface/volume



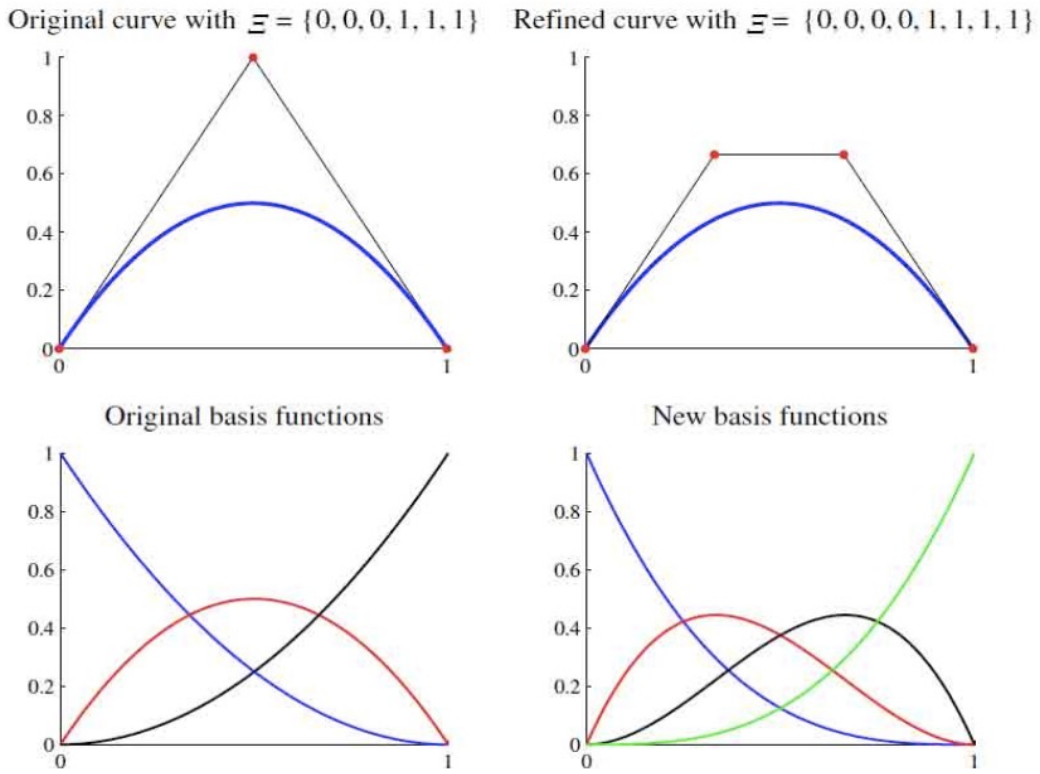
p-refinement

- order elevation: increase the order of B-spline basis function without changing the shape of the curve/surface/volume



p-refinement

- order elevation: increase the order of B-spline basis function without changing the shape of the curve/surface/volume



Remark

- p-h-k refinement: increase number of freedom (control points) to achieve better analysis results without changing the geometry
- Keep number of freedom as constant to obtain better results ?
- One option: optimize the placement of inner control points
- Parameterization of computational domain is changed while keeping the boundary geometry

Remark

- Model quality has some effect on analysis results
- accuracy and efficiency
- How to improve model quality for isogeometric analysis ?

Comput. Methods Appl. Mech. Engrg. 200 (2011) 2021–2031

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

Parameterization of computational domain in isogeometric analysis: Methods and comparison

Gang Xu [a,b,*,1], Bernard Mourrain [b], Régis Duvigneau [c], André Galligo [d]

[a] College of Computer, Hangzhou Dianzi University, Hangzhou 310018, PR China
[b] Galaad, INRIA Sophia-Antipolis, 2004 Route des Lucioles, 06902 Cedex, France
[c] OPALE, INRIA Sophia-Antipolis, 2004 Route des Lucioles, 06902 Cedex, France
[d] University of Nice Sophia-Antipolis, 06108 Nice Cedex 02, France

Problem statement

r-refinement

given initial placement of control points of computational domain,reposition the inner control points to achieve more accurate simulation results in isogeometric analysis

Main idea

- Inspired from shape optimization
- Shape optimization: optimize the boundary model to minimize the error function (cost function) of simulation

Main idea

Let the inner control points, rather than boundary control points, be the design variables for the shape optimization, and find the best placement of inner control points to make the value of a cost function as small as possible.

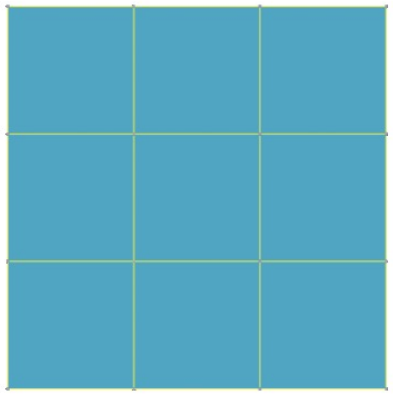
Main approach

- optimization variables: the coordinates of the inner control points
- cost function: error of the IGA solution
- optimization algorithm: steepest-descent method in conjunction with a back-tracking line-search
  1. Evaluation of perturbed points $x_k + \epsilon e_k$
  2. Estimation of the gradient $\nabla f(x_k)$ by finite-difference
  3. Define search direction $d_k = -\nabla f(x_k)$
  4. Line search : find ρ such as $f(x_k + \rho d_k) < f(x_k)$

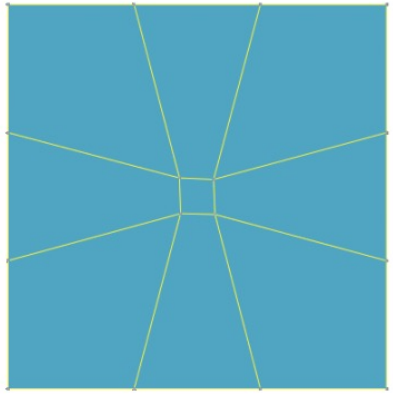
Example 1: given...

- source term : $\boldsymbol{f}(x, y) = -\frac{4\pi^2}{9}\sin(\frac{\pi x}{3})\sin(\frac{\pi y}{3})$.
- boundary condition : $\boldsymbol{U}_0(\boldsymbol{x}) = 0$ and $\Phi_0(\boldsymbol{x}) = 0$
- exact solution : $\boldsymbol{U}(x, y) = 2\sin(\frac{\pi x}{3})\sin(\frac{\pi y}{3})$.
- computational domain : $\Omega(x, y) = [0, 3] \times [0, 3]$

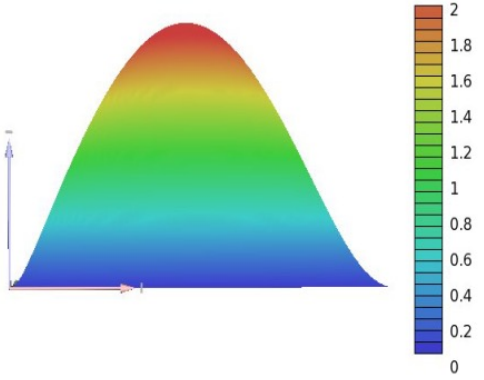
Example 1: results



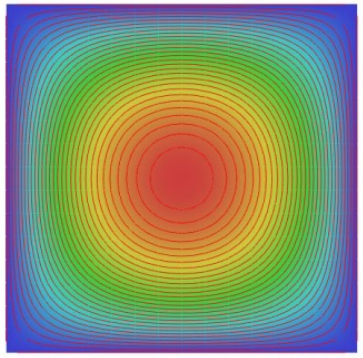
initial solution

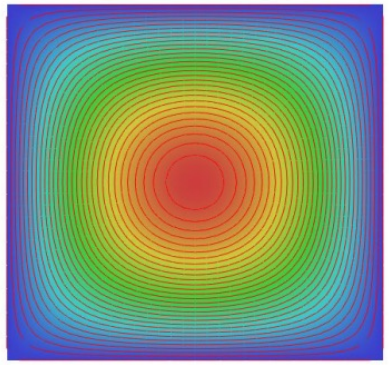


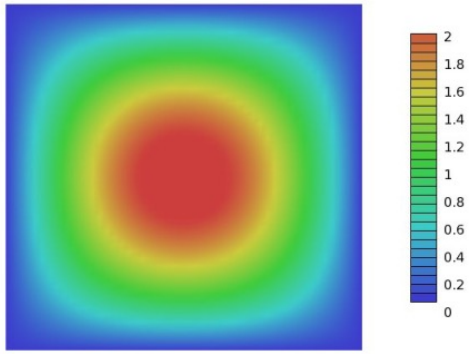
final solution



exact solution





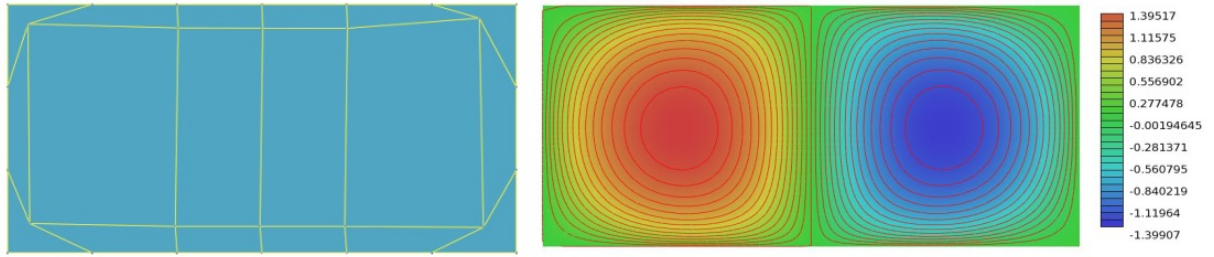


Example 3: given...

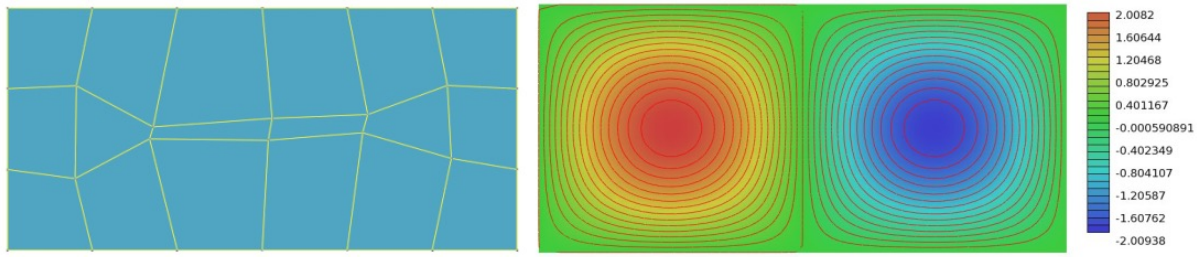
- source term : $\boldsymbol{f}(x,y) = -\frac{4\pi^2}{9}\sin(\frac{\pi x}{3})\sin(\frac{\pi y}{3})$.
- boundary condition : $\boldsymbol{T}_0(\boldsymbol{x}) = 0$ and $\Phi_0(\boldsymbol{x}) = 0$
- exact solution : $\boldsymbol{T}(x,y) = 2\sin(\frac{\pi x}{3})\sin(\frac{\pi y}{3})$.
- computational domain : $\Omega(x,y) = [0,6] \times [0,3]$

Example 3: results

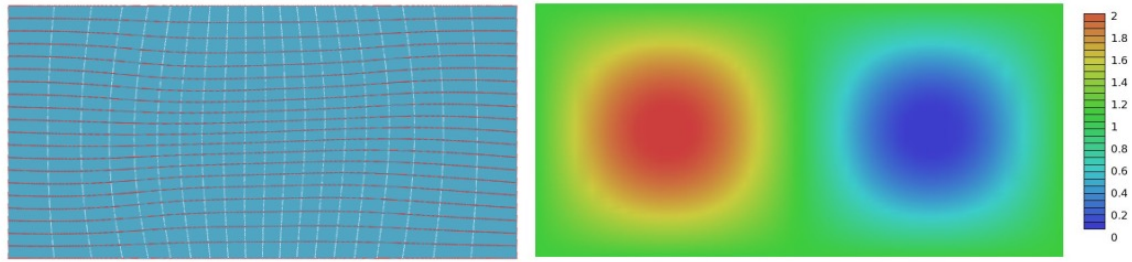
initial:



final:



exact:



r-refinement with unknown solutions

Residual function

- model problem:

$$\begin{aligned} \Delta U &= f \quad \text{in } \Omega \\ U &= U_0 \text{ on } \partial\Omega_D \end{aligned} \tag{2}$$

- U_h is the IGA solution
- error: $e = U - U_h$
- residual function

$$\begin{aligned} \mathcal{R}(\psi) &= \int_\Omega f\ \psi\ d\Omega + \int_\Omega \nabla U_h\ \nabla\psi\ d\Omega \\ &= \sum_{K\in\Omega} \int_K (f\ \psi - \Delta U_h\ \psi)\ dK \end{aligned}$$

A posteriori error estimate

$$\|e\|^2 \leq C \sum_{K \in \Omega} h_K \int_K (f - \Delta U_h)^2 dK \tag{3}$$

Main idea for r-refinement

reposition inner control points to minimize $\sum_{K \in \Omega} h_K \int_K (f - \Delta U_h)^2 dK$.

Key point

- $$\Delta U_h = \frac{\partial^2 U_h}{\partial^2 x} + \frac{\partial^2 U_h}{\partial^2 y} \tag{4}$$
- Solution field in isogeometric analysis

$$U_h(x, y) = \mathcal{T}_h(\xi, \eta) = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} \hat{N}_i^{p_i}(\xi)\, \hat{N}_j^{p_j}(\eta)\, T_{ij}$$

Key point

How to compute ΔU_h?

Main idea of IGA

- Use the same mathematical representation for the computational domain and solution field.
- Computational domain Ω is parameterized by the following planar B-spline surface:

$$P(\xi,\eta) = (x(\xi,\eta), y(\xi,\eta)) = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} N_i^{d_i}(\xi)\, N_j^{d_j}(\eta)\, p_{ij},$$

- Solution field over the computational domain Ω has the following form,

$$U_h(x(\xi,\eta), y(\xi,\eta)) = \mathcal{T}(\xi,\eta) = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} N_i^{d_i}(\xi)\, N_j^{d_j}(\eta)\, T_{ij},$$

Here T_{ij} are the unknown variables in isogeometric analysis to be solved.

Computation of ΔU_h

From $U_h(x(\xi,\eta), y(\xi,\eta)) = \mathcal{T}(\xi,\eta)$, we have

$$\frac{\partial \mathcal{T}}{\partial \xi} = \frac{\partial U_h}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial U_h}{\partial y}\frac{\partial y}{\partial \xi} \tag{5}$$

$$\frac{\partial \mathcal{T}}{\partial \eta} = \frac{\partial U_h}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial U_h}{\partial y}\frac{\partial y}{\partial \eta} \tag{6}$$

Then we can obtain

$$\frac{\partial U_h}{\partial x} = \left(\frac{\partial \mathcal{T}}{\partial \xi} y_\eta - \frac{\partial \mathcal{T}}{\partial \eta} y_\xi\right)/J \tag{7}$$

$$\frac{\partial U_h}{\partial y} = \left(\frac{\partial \mathcal{T}}{\partial \eta} x_\xi - \frac{\partial \mathcal{T}}{\partial \xi} x_\eta\right)/J \tag{8}$$

where $J = x_\xi y_\eta - y_\xi x_\eta$.

Computation of ΔU_h — continued

$$\frac{\partial^2 \mathcal{T}}{\partial^2 \xi} = \frac{\partial^2 U_h}{\partial^2 x}\left(\frac{\partial x}{\partial \xi}\right)^2 + \frac{\partial U_h}{\partial x}\frac{\partial^2 x}{\partial^2 \xi} + \frac{\partial^2 U_h}{\partial^2 y}\left(\frac{\partial y}{\partial \xi}\right)^2 + \frac{\partial U_h}{\partial y}\frac{\partial^2 y}{\partial^2 \xi} \quad (9)$$

$$\frac{\partial^2 \mathcal{T}}{\partial^2 \eta} = \frac{\partial^2 U_h}{\partial^2 x}\left(\frac{\partial x}{\partial \eta}\right)^2 + \frac{\partial U_h}{\partial x}\frac{\partial^2 x}{\partial^2 \eta} + \frac{\partial^2 U_h}{\partial^2 y}\left(\frac{\partial y}{\partial \eta}\right)^2 + \frac{\partial U_h}{\partial y}\frac{\partial^2 y}{\partial^2 \eta} \quad (10)$$

From (9)(10)

$$\frac{\partial^2 U_h}{\partial^2 x} = [y_\eta^2(\mathcal{T}_{\xi\xi} - \frac{\partial U_h}{\partial x}x_{\xi\xi} - \frac{\partial U_h}{\partial y}y_{\xi\xi}) - y_\xi^2(\mathcal{T}_{\eta\eta} - \frac{\partial U_h}{\partial x}x_{\eta\eta} - \frac{\partial U_h}{\partial y}y_{\eta\eta})]/K \quad (11)$$

$$\frac{\partial^2 U_h}{\partial^2 y} = [x_\xi^2(\mathcal{T}_{\eta\eta} - \frac{\partial U_h}{\partial x}x_{\eta\eta} - \frac{\partial U_h}{\partial y}y_{\eta\eta}) - x_\eta^2(\mathcal{T}_{\xi\xi} - \frac{\partial U_h}{\partial x}x_{\xi\xi} - \frac{\partial U_h}{\partial y}y_{\xi\xi})]/K \quad (12)$$

where $K = (x_\xi y_\eta)^2 - (x_\eta y_\xi)^2$

$$\Delta U_h = [(x_\xi^2 - y_\xi^2)(\mathcal{T}_{\eta\eta} - \frac{\partial U_h}{\partial x}x_{\eta\eta} - \frac{\partial U_h}{\partial y}y_{\eta\eta}) - (x_\eta^2 - y_\eta^2)(\mathcal{T}_{\xi\xi} - \frac{\partial U_h}{\partial x}x_{\xi\xi} - \frac{\partial U_h}{\partial y}y_{\xi\xi})]/K$$

Overview of r-refinement in IGA

1. Solve IGA problem over given computational domain
2. Compute $\sum_{K\in\Omega} h_K \int_K (f - \Delta U_h)^2 dK$
3. Reposition inner control points by minimizing $\sum_{K\in\Omega} h_K \int_K (f - \Delta U_h)^2 dK$
4. Output final placement of inner control points

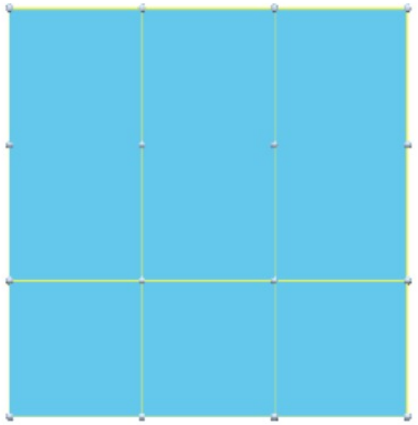
Error assessment

- $e = U - U_h$
- A posteriori error assessment by resolving IGA problem:

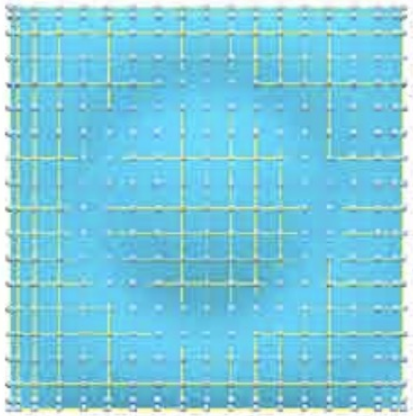
$$\begin{aligned} \Delta e &= f - \Delta U_h \quad \text{in } \Omega \\ e &= 0 \qquad\quad \text{on } \partial\Omega_D \end{aligned} \tag{13}$$

- Error field e has a B-spline form
- Perform h-refinement to achieve a good approximation
- More accurate but much more expensive
- Used for error assessment in r-refinement method

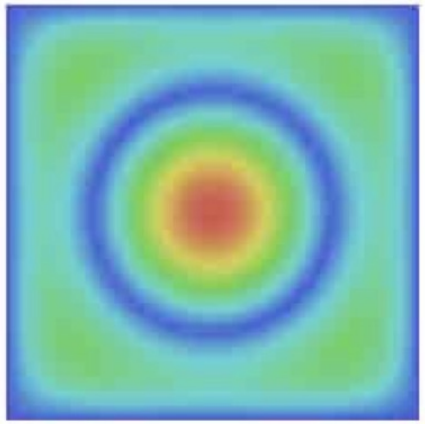
Error assessment: an example



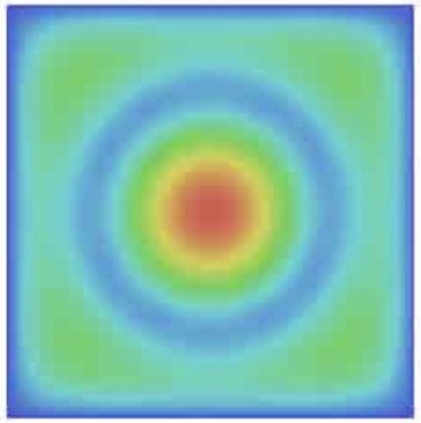
computational domain (CD)



resolved error surface



exact error colormap (EC)



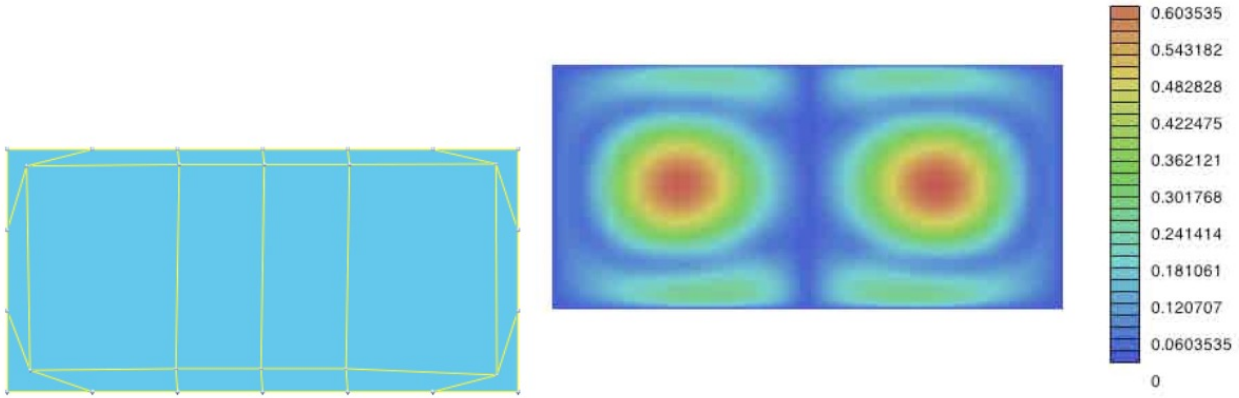
resolved EC

Model problem

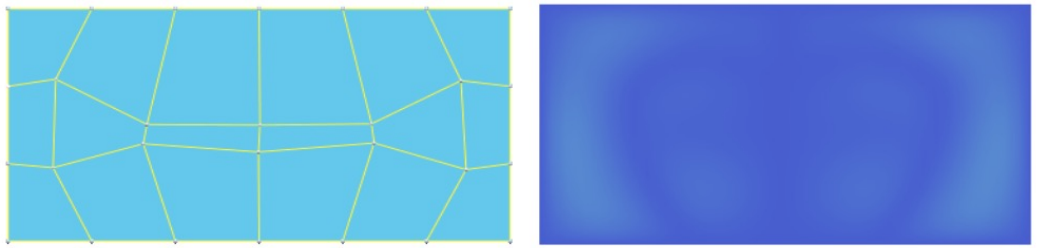
$$\begin{aligned} \Delta U &= -\frac{4\pi^2}{9}\sin(\frac{\pi x}{3})\sin(\frac{\pi y}{3}) \quad \text{in } \Omega \\ U &= 0 \qquad\qquad\qquad\qquad\qquad\quad \text{on } \partial\Omega_D \end{aligned} \tag{14}$$

Example II with known exact solution

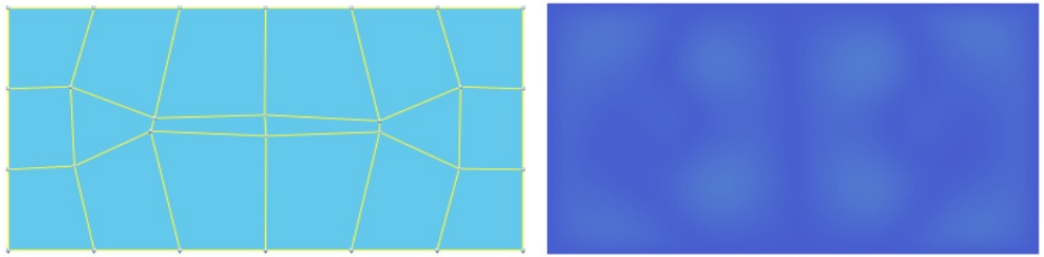
Initial CD and EC:



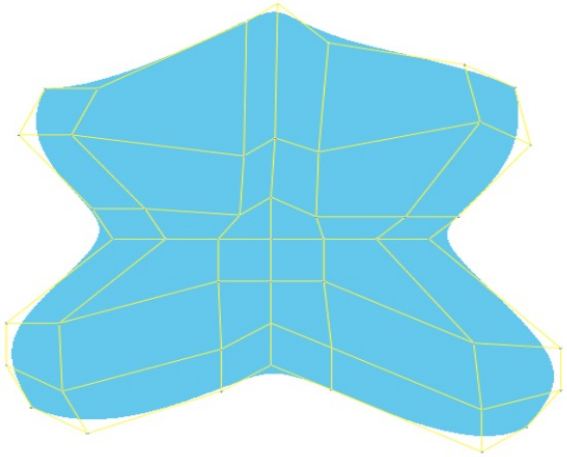
Final CD and EC:



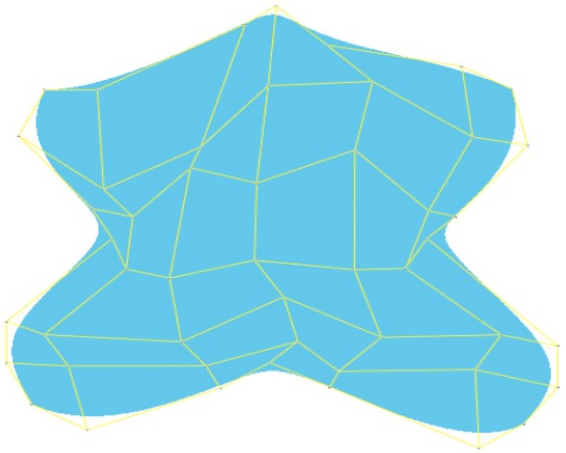
Exact method:



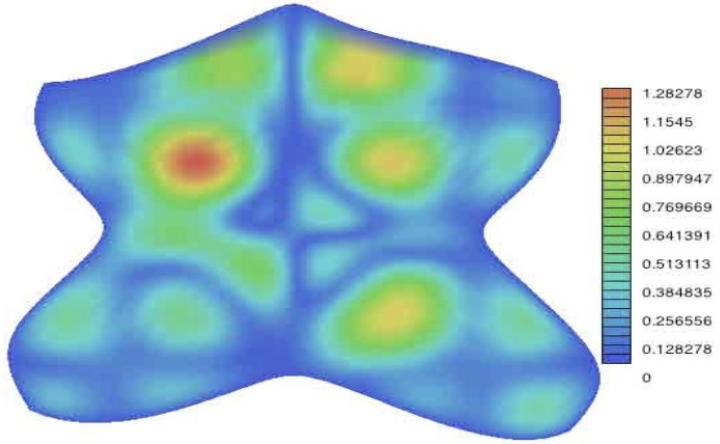
Example III with unknown exact solution



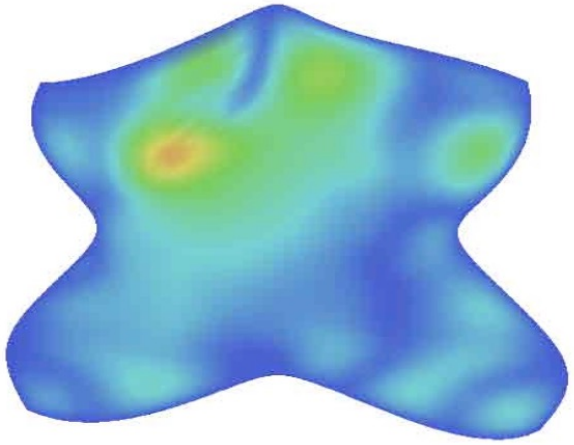
initial CD



CD after r-refinement

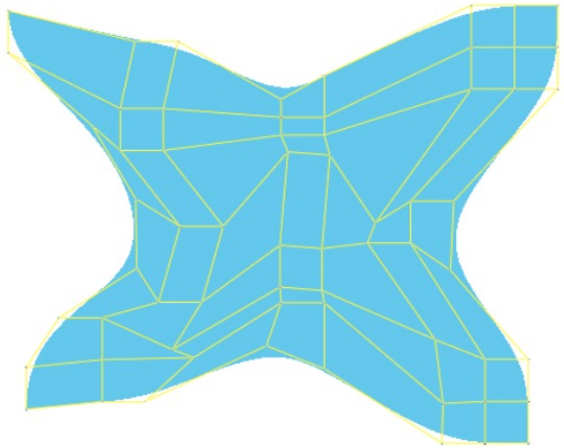


initial EC

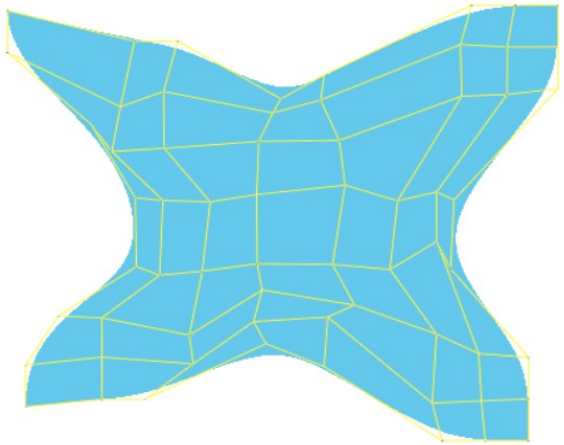


EC after r-refinement

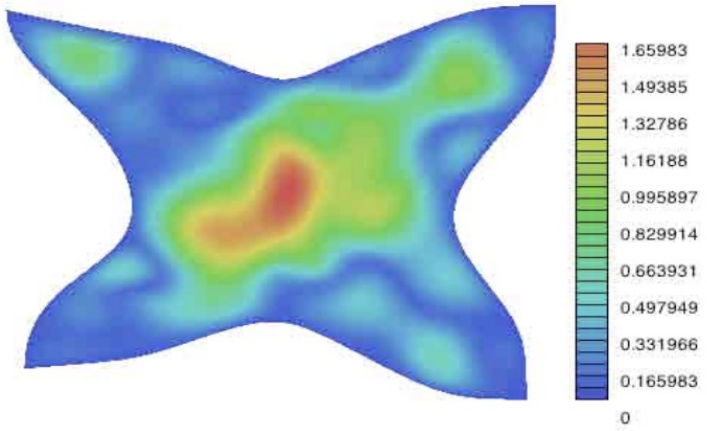
Example IV with unknown exact solution



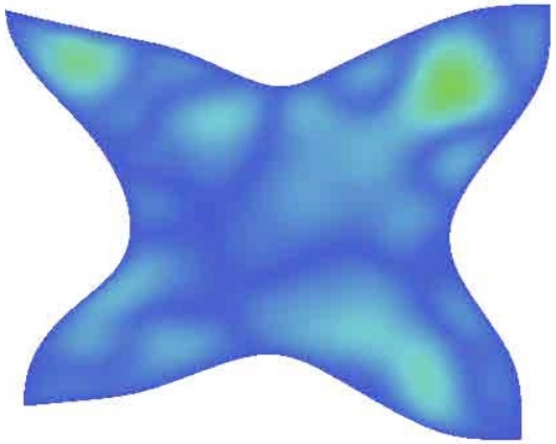
initial CD



CD after r-refinement



initial EC



EC after r-refinement

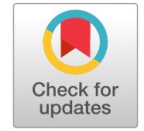
Journal of Computational and Applied Mathematics 351 (2019) 186–197
https://doi.org/10.1016/j.cam.2018.11.003

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Efficient *r*-adaptive isogeometric analysis with Winslow's mapping and monitor function approach



Gang Xu [a,b,c,*], Bojian Li [a], Laixin Shu [a], Long Chen [d], Jinlan Xu [a], Tahsin Khajah [e]

[a] *School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China*
[b] *Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, Hangzhou 310018, China*
[c] *LHPCSIP(MOE), Hunan Normal University, Changsha 410081, China*
[d] *University of Shanghai for Science and Technology, Shanghai 200093, China*
[e] *Department of Mechanical Engineering, University of Texas at Tyler, USA*

Input: Initial parameterization of computational domain, initial control points $\boldsymbol{P}_{i,j}^0$, and convergence threshold ϵ;

- Step. 1: solve the governing PDE over the given parameterization by isogeometric analysis to obtain an initial solution;
- Step. 2: compute the monitor function and relocate the interior control points by solving the Euler–Lagrange equation using isogeometric collocation method;
- Step. 3: solve the governing PDE over the new parameterization and update the monitor function;
- Step. 4: terminate if the terminal condition, $\|\boldsymbol{P}_{i,j}^{k+1} - \boldsymbol{P}_{i,j}^{k}\|_2 < \epsilon$, continue otherwise;
- Step. 5: repeat from Step.2

Output: the optimal parameterization of computational domain

$$\nabla \cdot \left(\frac{1}{G(\xi,\eta)}\nabla \mathbf{u}\right) = 0$$

$$\frac{\partial}{\partial \xi}\left(\frac{\boldsymbol{P}_\eta^T \boldsymbol{P}_\eta}{JG(\xi,\eta)}\right) - \frac{\partial}{\partial \eta}\left(\frac{\boldsymbol{P}_\xi^T \boldsymbol{P}_\eta}{JG(\xi,\eta)}\right) = 0,$$

$$-\frac{\partial}{\partial \xi}\left(\frac{\boldsymbol{P}_\eta^T \boldsymbol{P}_\xi}{JG(\xi,\eta)}\right) + \frac{\partial}{\partial \eta}\left(\frac{\boldsymbol{P}_\xi^T \boldsymbol{P}_\xi}{JG(\xi,\eta)}\right) = 0.$$

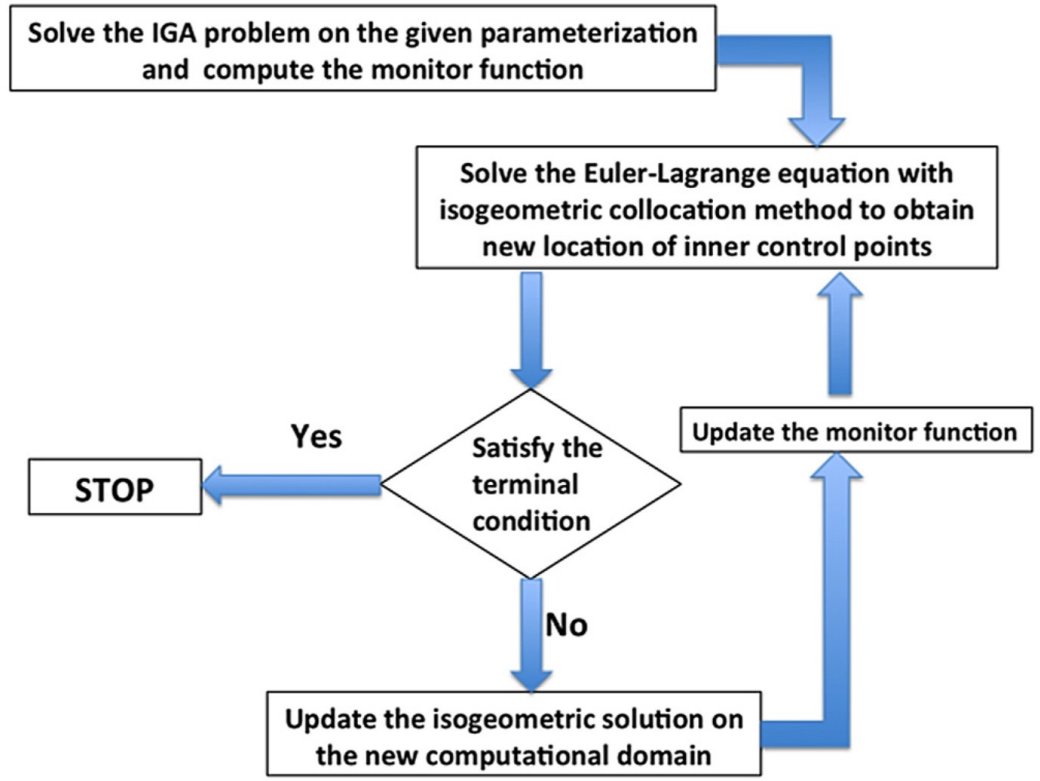


Fig. 1. The r-adaptive algorithm with Winslow's mapping.

The maximum and minimum principal curvatures at a point on the surface are denoted by κ_1, and κ_2 respectively and defined as

$$\kappa_1 = \frac{-B + \sqrt{B^2 - 4AC}}{2A},$$

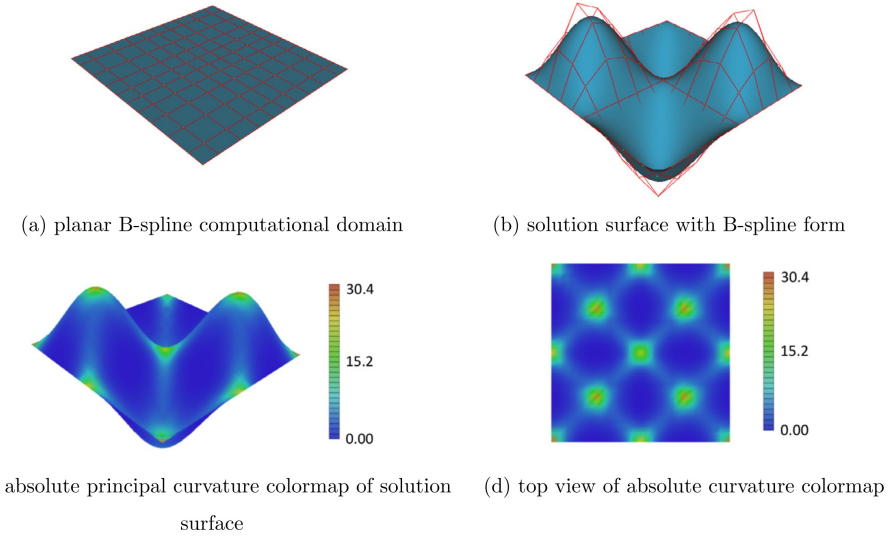
$$\kappa_2 = \frac{-B - \sqrt{B^2 - 4AC}}{2A},$$

where

$$A = EG - F^2,$$
$$B = 2FM - GL - EN,$$
$$C = LN - M^2.$$

We define the monitor function, $G^1(\xi, \eta)$, as the sum of absolute principal curvatures

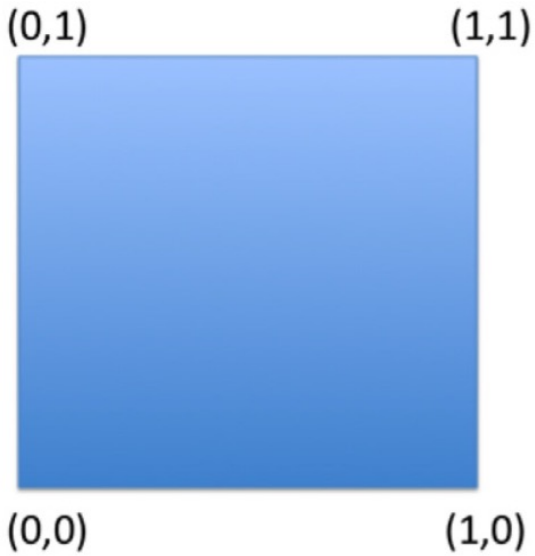
$$G^1(\xi, \eta) = |\kappa_1| + |\kappa_2|,$$



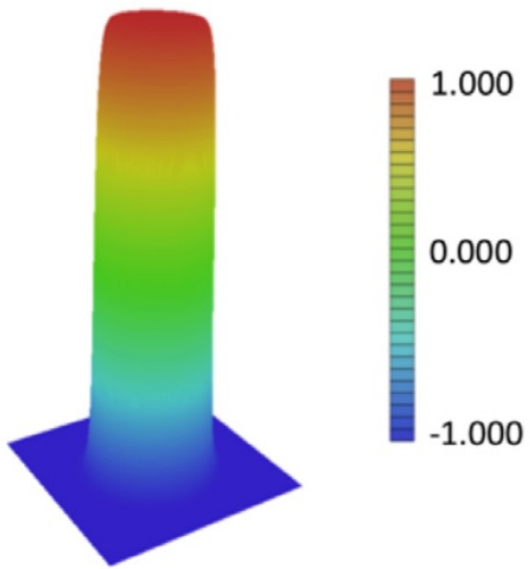
(a) planar B-spline computational domain (b) solution surface with B-spline form

(c) absolute principal curvature colormap of solution surface (d) top view of absolute curvature colormap

Fig. 2. B-spline solution surface and the corresponding absolute principle curvature colormap.

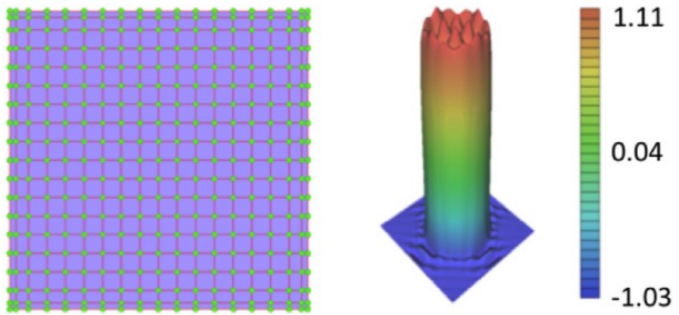


(a) computational domain

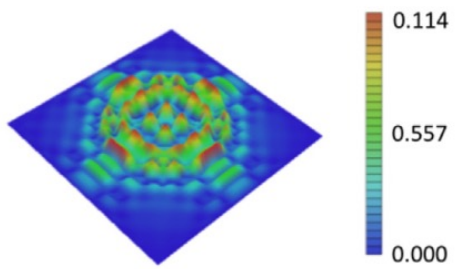


(b) exact solution

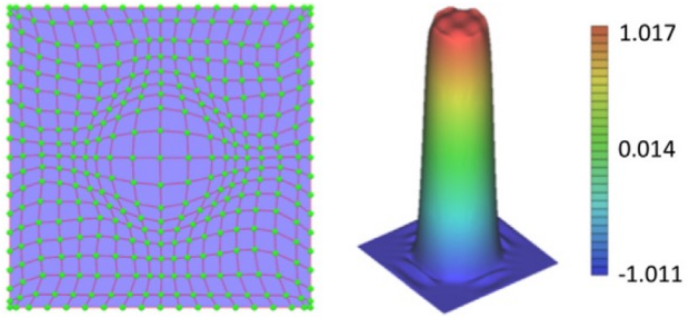
$$T(\boldsymbol{x}) = \tanh\left(\frac{0.25 - \sqrt{(x-0.5)^2 + (y-0.5)^2}}{0.03}\right).$$



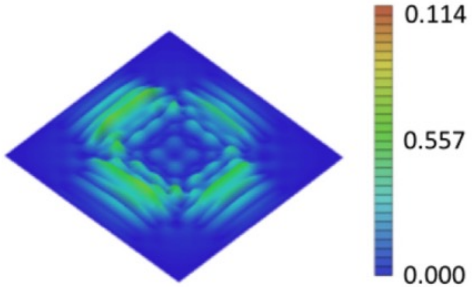
(a) initial parameterization and IGA solution



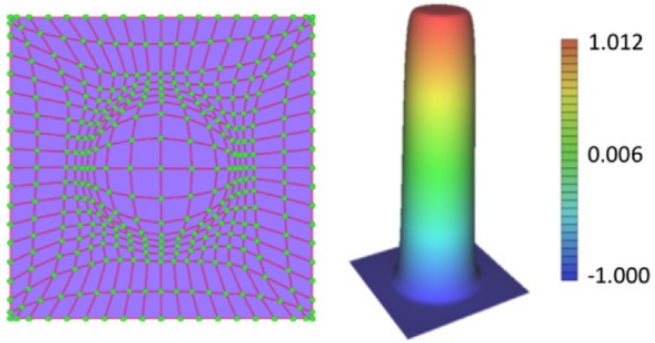
(b) initial error



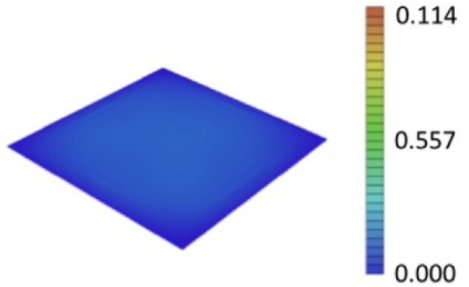
(c) intermediate parameterization and IGA solution



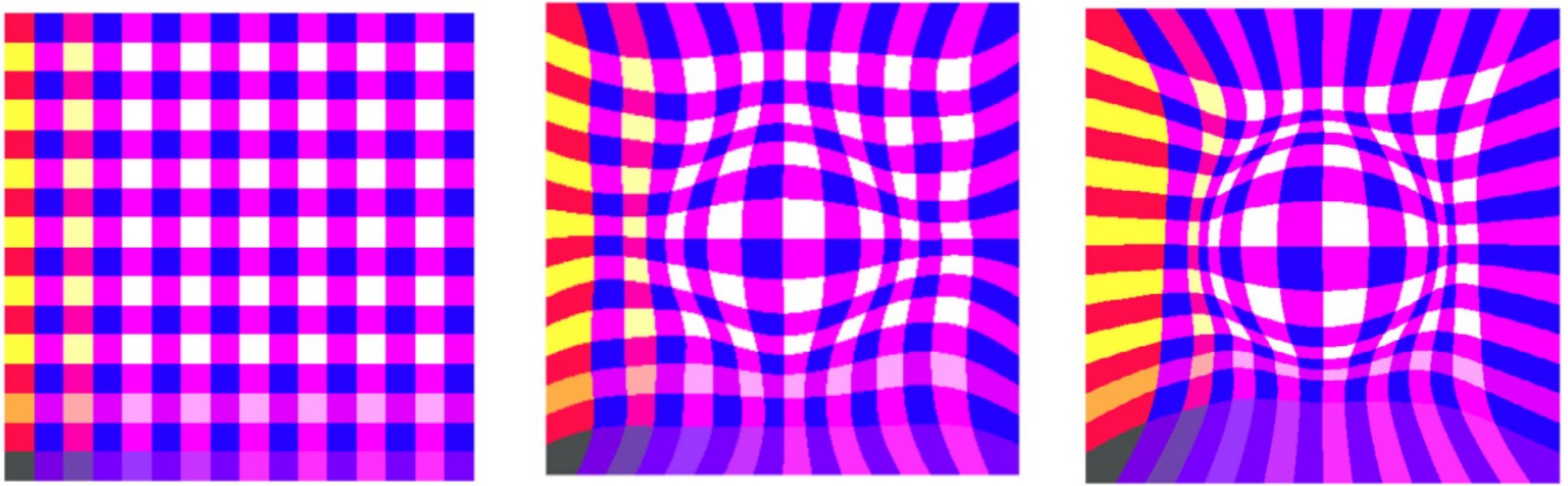
(d) intermediate error



(e) final parameterization and IGA solution

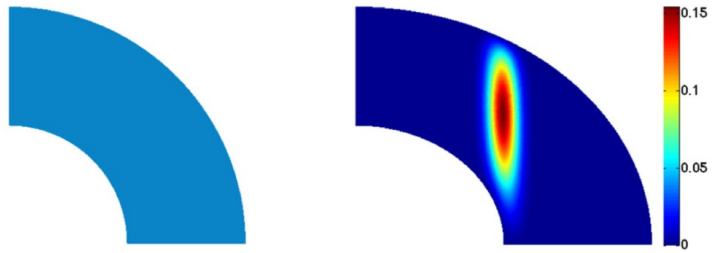


(f) final error



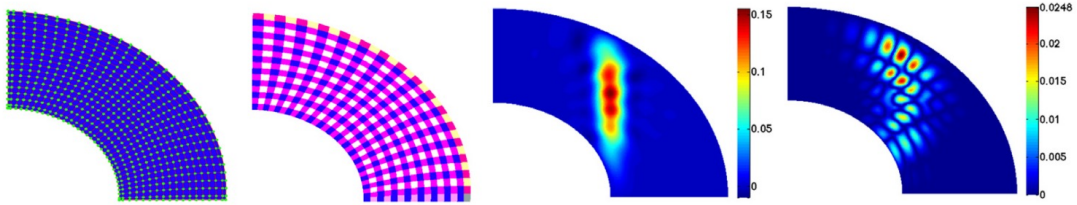
(a) initial patch structure (b) intermediate patch (c) final patch structure

$$G(\xi, \eta) = |f(\mathbf{u}) + \Delta T^h(\mathbf{u})|$$

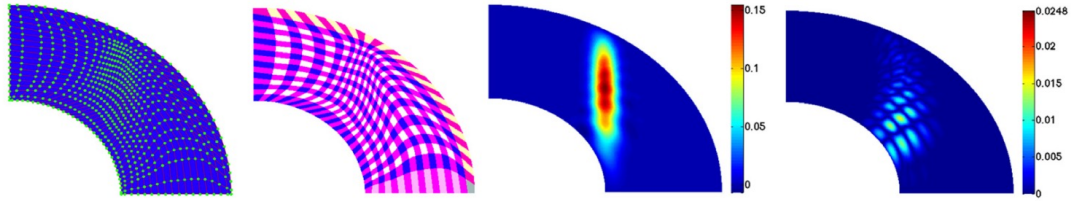


(a) computational domain (b) exact solution

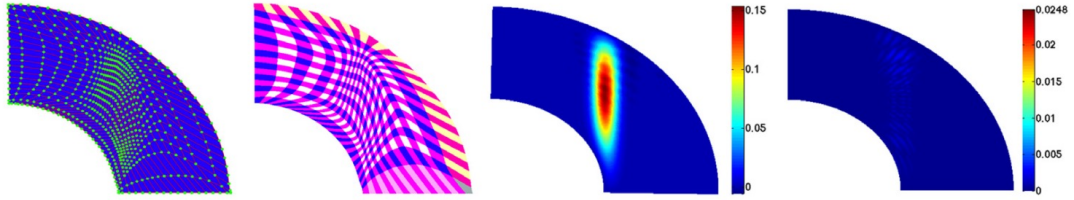
Fig. 6. Description of Example II.



(a) initial parameterization (b) initial patch-structure (c) initial solution (d) initial error colormap



(e) intermediate parameterization (f) intermediate patch structure (g) intermediate solution (h) intermediate error colormap



(i) final (j) final (k) final solution (l) final error colormap

智能可视建模与仿真实验室

Intelligent Visual Modeling & Simulation (iGame) Lab

Thank You !

Q & A