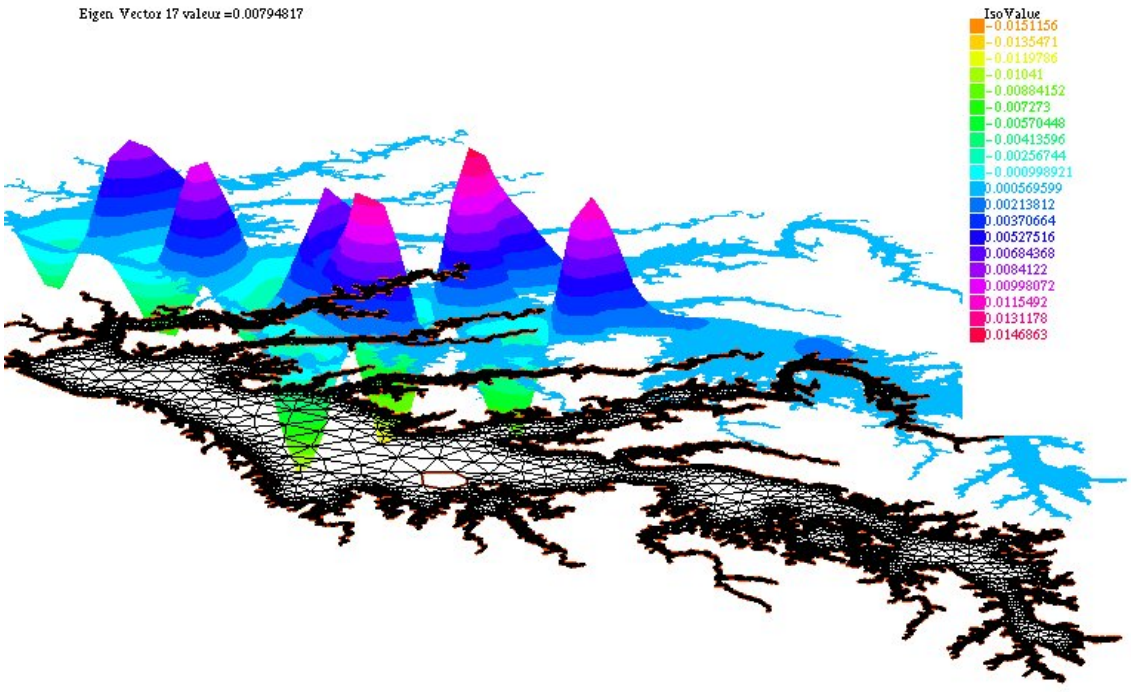
Freefem++

第三版, 3.32版本

http://www.freefem.org/ff++

F. Hecht



皮埃尔-玛丽・居里大学雅克-路易・莱昂斯实验室（法国巴黎）

FreeFem++

第三版 3.32版本

http://www.freefem.org/ff++

Frédéric Hecht[1, 2]
mailto:frederic.hecht@upmc.fr
http://www.ann.jussieu.fr/~hecht

合作者:

- Sylvian Auliac, mailto:auliac@ann.jussieu.fr, http://www.ann.jussieu.fr/auliac, Sylvian Auliac 是一名博士生, 他用 nlopt, ipopt, cmaes 等工具完成了所有新的界面优化。
- Olivier Pironneau, mailto:olivier.pironneau@upmc.fr, http://www.ann.jussieu.fr/pironneau Olivier Pironneau 是巴黎六大雅克-路易·莱昂斯实验室 （LJLL） 的一名数值分析教授，法兰西大学研究院院士、法国自然科学院院士. 他的主要贡献在流体相关的数值方法方面。
- Jacques Morice, mailto:morice@ann.jussieu.fr. Jacaues Morice 是 LJLL 的一名博士后。 他目前在波尔多第一大学做有关快速多级方法(FMM)的毕业论文。在本版中, 他主要负责所有三维网格的生成以及与medit软件耦合的工作。
- Antoine Le Hyaric, mailto:lehyaric@ann.jussieu.fr, http://www.ann.jussieu.fr/~lehyaric/ Antoine Le Hyaric 来自"国家科学研究中心"(CNRS) , 目前是LJLL的一名研究性工程师。他是科学应用方面的软件工程专家。 在本版的贡献主要在电磁模拟、并行计算和三维可视化方面。
- Kohji Ohtsuka,mailto:ohtsuka@hkg.ac.jp, http://www.comfos.org/ Kohji Ohtsuka 是日本广岛国际学院大学的教授, 世界科学与工程研究院与学会日本支部的会长，其研究方向是断裂动力学建模与计算。
- 中文版本,本手册中文版本由复旦大学数学系学生翻译，LJLL在读博士生龚禾林（Helin GONG）校对,核能研究展望NPRV 中心审核。


鸣谢： 感谢巴黎综合理工学院 (法国帕莱索) 印刷该手册的第二版(http://www.polytechnique.fr), 感谢法国国家科研署 (法国巴黎) 的资助, 将FreeFem++ 拓展为相应的三维版本 (http://www.agence-nationale-recherche.fr) 参考: ANR-07-CIS7-002-01。

[1]皮埃尔-玛丽·居里大学雅克-路易·莱昂斯实验室（法国 巴黎）。
[2]Projet Alpines, 法国国家信息与自动化研究所 Rocquencourt地区分所。

目 录

前言

ॐ पूर्णमदः पूर्णमिदं पूर्णात् पूर्णमुदच्यते।
पूर्णस्य पूर्णमादाय पूर्णमेवावशिष्यते ॥
ॐ शान्तिः शान्तिः शान्तिः ॥

经过很长一段时间的演变, freefem, 成为了最后的FreeFem++. 它是一个高度集成的开发环境 (IDE), 用以数值求解2维和3 维的偏微分方程 (PDE)。它不仅是有限元教学的理想工具, 对于研究人员快速测试新的想法或者复杂的多物理应用也是相当完美的。

FreeFem++拥有先进的网格生成器, 有后验的网格自适应能力; 它的通用椭圆问题求解器集成了诸如UMFPACK, SuperLU等快速算法。 对于双曲和抛物问题，用户可以通过FreeFem++高级语言编制相应的迭代算法来求解。 FreeFem++拥有多种三角有限元, 包括不连续元。 最后，FreeFem++ 产生的所有用于准备研究报告的数据都能在线彩色缩放显示以及postscript打印输出。

这本手册主要写给硕士层面的学生, 各种层面的研究人员以及对偏微分方程和变分法有一些了解的工程师们 (包括金融工程师)。

第 1 章

简介

偏微分方程－－－－即一个多变量函数与其（偏）导数的关系式。 许多物理、工程、数学, 甚至银行业的问题均可以通过一个或若干偏微分方程进行建模。

FreeFem++ 是一个数值求解这些方程的软件。 顾名思义, 它是一个基于有限元方法的免费软件 (查看版权以了解完整细节); 它不是一个包, 而是一个拥有自己高级编程语言的集成化产品。 这个软件在所有(带有g++ 3.3或更新版本, OpenGL 的)UNIX操作系统, Windows XP/Vista/7, MacOS 10 (powerpc/intel处理器)上都能运行。

此外 FreeFem++ 有很强的适应性。 许多现象含有耦合系统, 如：流固耦合, 铸铝中的洛仑兹力和海气问题等，它们需要不同的有限元近似, 多项式次数, 甚至可能不同的网格。一些算法如Schwarz区域分解方法还需要让多种网格上的数据插值在一个程序中完成。FreeFem++ 正好可以解决这些困难, 这意味着，FreeFem++ 能产生任意非结构化、自适应二维网格上的任意有限元空间。

FreeFem++ 的特点有:

- 通过变分公式 (实值或复值的) 来描述问题, 如果需要, 可以访问内部向量和矩阵。
- 多变量, 多方程, 二维/三维的, 静态的/时变的, 线性的/非线性的耦合系统; 但是用户需要将迭代过程进行适当的表述，使得问题最终归结为一组线性问题。
- 通过分片解析描述边界以实现简单的几何输入; 然而这部分并不是一个CAD系统：比如当两个边界相交时, 用户必须指出相交的点。
- 基于Delaunay-Voronoi算法的自动网格生成器; 内点的密度正比于边界上点的密度[7]。
- 基于度量的各向异性网格自适应. 这种度量可以由任一FreeFem++ 函数的Hessian自动计算[9]。
- 高级的用户友好型输入语言, 带有代数分析和有限元函数。
- 一个应用中可允许多种有限元网格, 不同网格上数据自动插值, 并且可以存储插值矩阵。

- 许多三角型有限元: 线性元, 二次Lagrange型有限元, 不连续P1元, Raviart-Thomas元, 非基本类型元, 微单元,... (但是没有四次元)。
- 用于定义不连续Galerkin有限元公式 `P0`, `P1dc`, `P2dc`的工具, 关键词: `jump`, `mean`, `intalledges`。
- 各种线性直接和迭代求解器 (LU, Cholesky, Crout, CG, GMRES, UMFPACK, MUMPS, SuperLU, ...) 以及特征值、特征向量求解器 (ARPARK) 。
- 近似最优的执行速度 (相比于编译C++直接编程实现)。
- 在线制图, 生成 `.txt`,`.eps`,`.gnu`, `mesh`文件供以后输入输出处理。
- 许多例子和指南: 椭圆问题、抛物问题的和双曲的问题, Navier-Stokes流场, 弹性力学, 流固交互作用, Schwarz区域分解方法, 特征值问题, 残量误差指标, ...
- 并行版本可使用 `mpi`.

1.1 安装

1.1.1 对于所有人:

首先打开下面的网页

http://www.freefem.org/ff++/

选择你的平台: Linux, Windows, MacOS X, 或者到页面的底端来获取所有下载列表。

注 1 *:* 二进制文件对于*Microsoft Windows,* 苹果 *Mac OS X* 以及一些*Linux*系统可用。

通过双击适当的文件进行安装, 在linux和MacOS环境下, 安装文件分别位于 `/usr/local/bin`, `/usr/local/share/freefem++`, `/usr/local/lib/ff++`目录。

Windows二进制文件安装 需要预先下载windows可执行安装文件, 然后双击安装`FreeFem++`。通常只要对弹出的问题点回答 (或按回车)。 在Additional Task窗口界面, 选中 “Add application directory to your system path”。此操作不可省略，否则`ffglut.exe` 程序将没法找到。

至此你应该有两个新图标出现在桌面上:

- `FreeFem++ (VERSION).exe`: `FreeFem++` 应用程序。
- `FreeFem++ (VERSION) Examples`: 一个到`FreeFem++` examples文件夹的链接。

这里`(VERSION)` 就是文件的版本号 (比如`3.3-0-P4`)。

如果默认安装, 则安装文件将位于

`C:\Programs Files\FreeFem++`

在这个目录下, 你有所有的`.dll`文件及其他应用程序: `FreeFem++-nw.exe`,`ffglut.exe`, ...以及一些没有图形窗口的`FreeFem++` 应用程序。

命令行工具的语法和FreeFem.exe的一样。

MacOS X二进制文件安装 下载MacOS X二进制版本文件后, 双击文件图标以提取所有文件, 将FreeFem+.app应用程序放到/Applications 目录下。如果你需要终端访问FreeFem++ , 只需将文件FreeFem++ 拷贝到你shell中的$PATH 环境变量目录下即可。

如果你想自动运行FreeFem++.app, 双击一个.edp文件图标。 比如在examples++-tutorial目录中通过查找选择了一个.edp文件, 选择菜单File -> Get Info中change Open with: (选择FreeFem++.app) 并点击 change All 按钮。

下一步做什么 FreeFem++ 提供了FreeFem++-cs, 一种由Antoine Le Hyaric编写的集成环境. 除非你希望现在就配置自己的开发环境, 否则你应该继续读下一段 " 如何使用FreeFem++"。

1.1.2 对于专业人士: 从源代码安装

这部分写给那些因为某些原因不愿意采用二进制文件从而需要重新编译FreeFem++ 或者从源代码安装的人:

使用说明文档 : 使用说明文档同样也是开源的; 重新生成它你需要能编译CVS 文档的LaTeX 环境; 在MS-Windows 环境下你可能不得不使用 mingw/msys

http://www.mingw.org

在MacOS X环境下我们使用了苹果的开发工具"Xcode" 并且 LaTeX 可以从 http://www.ctan.org/system/mac/texmac获得。

C++文档 : FreeFem++ 必须从源文档编译, 如下述网站中所示

http://www.freefem.org/ff++/index.htm

为了将压缩文件freefem++-(VERSION).tar.gz解压到

freefem++-(VERSION)

目录下, 在shell窗口中键入下述命令:

```
tar zxvf freefem++-(VERSION).tar.gz
cd freefem++-(VERSION)
```

为了编译并安装FreeFem++ , 只需按照INSTALL和README文件做即可。 根据你运行的系统, 将生成下列程序:

1. FreeFem++, 标准版本, 带有基于GLUT/OpenGL (使用ffglut可视化工具)的图形界面, 或者仅仅加一个 -nw 参数。
2. ffglut, freefem++可视化工具 (注: 如果*ffglut*不在系统路径中, 你将得不到绘图)。
3. FreeFem++-nw, 仅有postscript图形输出和ffmedit (批版, 没有ffglut的图形窗口)。

4. FreeFem++-mpi, 并行版本, 仅有postscript输出。

5. /Applications/FreeFem++.app, Drag and Drop CoCoa MacOSX Application.

6. bamg, bamg网格生成器。

7. cvmsh2, mesh文件转换器。

8. drawbdmesh, mesh文件浏览器

9. ffmedit, medit软件的freefem++版本 (感谢P. Frey)。

FreeFem++,FreeFem++-nw,FreeFem++-mpi 工具在命令行上的语法为

- FreeFem++ [-?] [-v nn] [-fglut file1] [-glut file2] [-f] edpfilepath
- FreeFem++-nw -? [-v nn] [-fglut file1] [-glut file2] [-f] edpfilepath

  -? 显示用法

  -fglut filename 存储所有作图数据到文件 filename,
  要重新展示键入ffglut filename

  -glut ffglutprogam 改变可视化软件

  -nw 不使用ffglut和medit

  -v nn 在执行脚本之前将信息显示等级设置为 nn

  -ne 没有edp脚本输出

  -wait 直至在文本窗口输入return再关闭FreeFem++

  -nowait 直接关闭FreeFem++

  -cd 将目录转换到脚本目录(脚本路径必须是全局的)

  如果没有文件路径那么你将获得一个对话框来选择windows系统中的edp文件

标记[]意味“可选择”

注 2 通常你可以通过在命令行上加上参数*-v nn*来将输出 *(*信息显示*)* 等级设置为*nn*。

作为一个unix环境下的安装测试: 到examples++-tutorial 目录下用以下命令运行 FreeFem++ 样例脚本LaplaceP1.edp:

```
FreeFem++ LaplaceP1.edp
```

如果你使用的文本编辑器是nedit, 执行 nedit -import edp.nedit 来使得你的 .edp 文件拥有语法高亮。

使用其他文本编辑器

notepad++ http://notepad-plus.sourceforge.net/uk/site.htm

- 打开Notepad++, 按下F5
- 在弹出的窗口中键入命令`launchff++ "$(FULL_CURRENT_PATH)"`
- 点击保存, 并在"Name"框中输入`FreeFem++`, 现在选择快捷键来直接执行FreeFem++ (比如`alt+shift+R`)
- 在Notepad++中加入与FreeFem++相容的语法高亮,
  - 在菜单“设置”中选择“语言格式设置”:
  - 在“语言”列表中选择C++
  - 在“自定义扩展名”中输入“edp”
  - 在“样式”列表中选择“INSTRUCTION WORD”, 在“自定义关键字”中剪切粘贴下面的列表:
    P0 P1 P2 P3 P4 P5 P1dc P2dc P3dc P4dc P5dc RT0 RT1 RT2 RT3 RT4 RT5 macro plot int1d int2d solve movemesh adaptmesh trunc checkmovemesh on func buildmesh square Eigenvalue min max imag exec LinearCG NLCG Newton BFGS LinearGMRES catch try intalledges jump average mean load savemesh convect abs sin cos tan atan asin acos cotan sinh cosh tanh cotanh atanh asinh acosh pow exp log log10 sqrt dx dy endl cout
  - 在“样式”列表中选择“TYPE WORD”, 在“自定义关键字”中剪切粘贴下面的列表:
    mesh real fespace varf matrix problem string border complex ifstream ofstream
  - 点击“保存并关闭”， 现在nodepad++就配置好了。

Crimson Editor 在http://www.crimsoneditor.com/上可以下载, 并按照下列步骤调试:

- 进入`Tools/Preferences/File/Filters`菜单并加入.edp后缀名。
- 在同一面板下的`Tools/User Tools`选项卡里, 第一行输入`FreeFem++`, 第二行`freefem++.exe`输入的路径,第三、第四行输入 `$(FilePath)` 以及 `$(FileDir)`,勾选8.3选项框。
- 为了实现语法高亮, 解压`FreeFem++`文件夹中的`crimson-freefem.zip`, 并将解压后link, spec, tools 文件夹中的文件放入Crimson Editor文件夹对应子文件夹中。

winedt Windows系统: 这是最好的安装方法但需要一定技巧。 在下面网站下载winedet:

http://www.winedt.com

这是一个多用途的文本编辑器, 带有高级功能比如语法高亮;www.freefem.org上有一个宏来使得`FreeFem++` 在winedt上可用但不影响 winedt在LateX, TeX, C等等上的功能模式。 然而winedt在试用期后不是免费的。

TeXnicCenter Windows系统: 这是最早的文本编辑器, 而且一旦有志愿者编程实现语法高亮它将成为有史以来最好的一个。它可以从下面的地址下载:

http://www.texniccenter.org/

它也是一个TeX/LaTeX的编辑器。它有一个"Tools"菜单, 如下配置能够使其运行 FreeFem++ :

- 选择Tools/Customize选项, 这将弹出一个对话框。
- 上方选项卡中选择Tools, 新建一项: 命名为freefem。
- 在下面的3行里:
  1. 选择FreeFem++.exe文件路径
  2. 选择"Main file with further option", 在"Type of"中选择"Full path", 并勾选下方8.3框
  3. 选择"main file full directory path (8.3-format)"

注: 第一次进入TeXnicCenter需要初始化, 选择Latex路径: CTEX/MiKTeX/miktex/bin, 否则将没有输出结果。

nedit Mac OS, Cygwin/Xfree 和linux系统, 为导入语法高亮。

```
nedit -import edp.nedit
```

Smultron Mac系统, 可以在http://smultron.sourceforge.net 获取。
它自带关于.edp文件的语法高亮。为了让它能够运行FreeFem++ 文件,
执行"command B" (即: 菜单Tools/Handle Command/new command) 并创一个指令

```
/usr/local/bin/FreeFem++-CoCoa %%p
```

1.2 如何使用FreeFem++

在MacOS X系统图形界面下 .edp文件的测试, 只要将这个文件图标拖到MacOS应用图标FreeFem++.app上。 你也可以用菜单File → Open来运行这个应用。

然而Mac上最好的一种方法是使用一个如Smultron.app的文本编辑器 (参见之前所述)。

终端模式下 FreeFem++, FreeFem++-nw, FreeFem++-mpi, ... 中根据你的需要选择一类应用。至少加入路径名; 比如

```
FreeFem++ your-edp-file-path
```

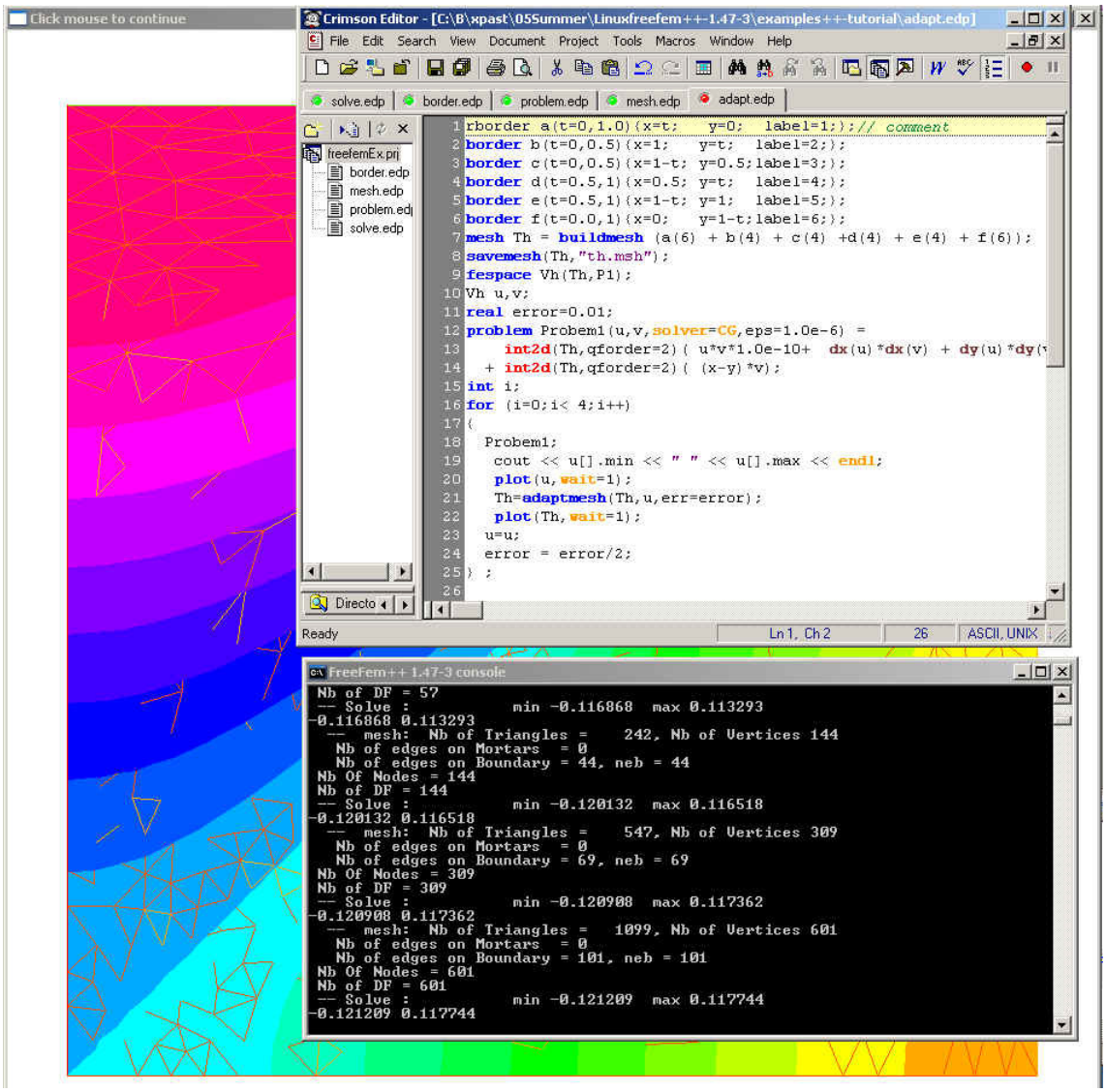


图 1.1: 使用Windows系统在集成环境下开发FreeFem++

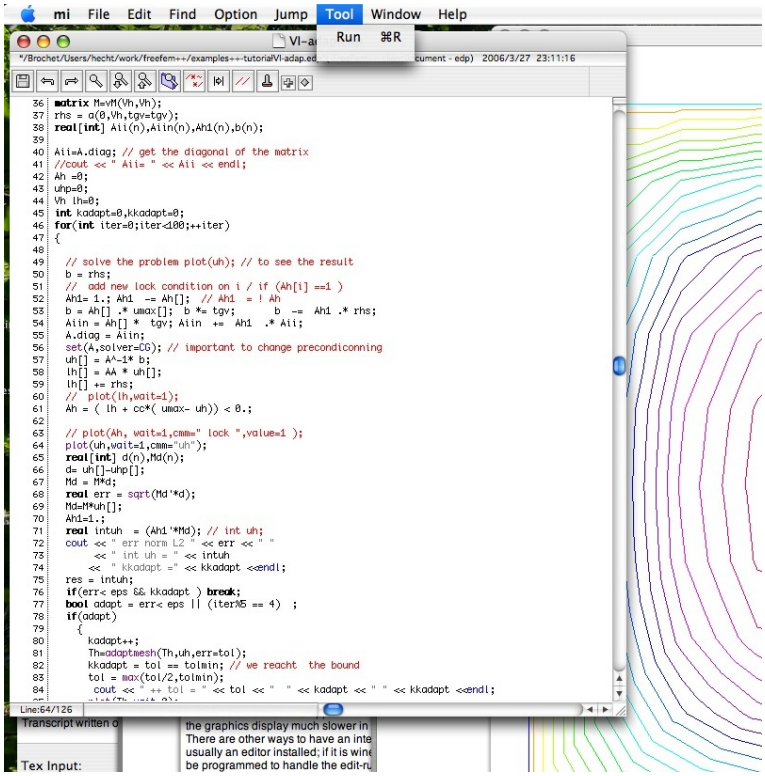


图 1.2: `fraise Editor`建立的3板集成环境来执行`FreeFem++` . Tools菜单中有一项来运行`FreeFem++` , 快捷键: Ctrl+1.

1.3 环境变量与初始文件

`FreeFem++` 读取用户的 `freefem++.pref`初始文件来初始化全局变量: `verbosity`, `includepath`, `loadpath`。

注 3 变量*verbosity*改变内部输出的等级 （0, 无 （除非有语法错误）, 1 极少, 10 很多, 等等 ...）, 默认值为 2。

头文件从*includepath*列表中搜索, 加载文件从*loadpath*列表中搜索。

这个文件的语法为:

```
verbosity= 5
loadpath += "/Library/FreeFem++/lib"
loadpath += "/Users/hecht/Library/FreeFem++/lib"
includepath  += "/Library/FreeFem++/edp"
includepath  += "/Users/hecht/Library/FreeFem++/edp"
#  comment
load += "funcTemplate"
load += "myfunction"
load += "MUMPS_seq"
```

这个文件的可能路径为:

- 在unix/MacOs系统下

```
/etc/freefem++.pref
   $(HOME)/.freefem++.pref
   freefem++.pref
```

- 在windows系统下

```
freefem++.pref
```

我们也可以用shell环境变量在初始文件之前来改变信息显示和搜索规则。

```
export FF_VERBOSITY=50
export FF_INCLUDEPATH="dir;;dir2"
export FF_LOADPATH="dir;;dir3""
```

注: 目录之间的分割必须用“;”而不是“:”因为“:”被Windows所用。

注： 要显示freefem++的初始化列表, 执行

```
export FF_VERBOSITY=100; ./FreeFem++-nw
--  verbosity is set to 100
insert init-files /etc/freefem++.pref $
...
```

1.4 历史

这个项目由MacFem, PCfem演化而来, 由Pascal语言所写。 第一个C语言版本产生了freefem 3.4; 它仅提供单个网格上的自适应性。

用C++对其的彻底重写产生了freefem+ (freefem+ 1.2.10是它最后发布的版本), 它包括了多重网格上的插值 (定义在一个函数上的网格可以用于任何其他一个网格); 尽管这个软件不再维护, 不过仍然在使用因为它解决了用强形式描述PDE的问题。从一个非结构化网格到另一个的插值实现并不容易因为它必须既快又不冗余; 对于每个点, 必须找到包含它的三角形。 这是计算几何一个基础问题 (参见Preparata & Shamos[18]) 。在最小操作数下实现它是具有挑战性的。我们的实现是$O(n \log n)$的, 且基于四叉树。 由于C++中模版语法的进化, 这个版本同样被抛弃了。

我们现在已经在FreeFem++ 上做了多年工作, 完全用C++ template语法进行再一次重写； 以及在编译时不定大小耦合问题上的泛型编程工作。 与所有 freefem 版本一样, 它拥有高级的用户友好型输入语言, 使其与数学语言描述问题的方法差不太多。

freefem语言可以快速描述任一偏微分方程系统。FreeFem++ 语言的语法是利用STL[26]、模板和bison进行全新设计实现的; 更多细节可以在 [12]中找到。这使得软件具有通适性, 任一新的有限元可以在数小时内被包含进来，而无需重新编译。 因此FreeFem++ 中可用有限元库将随着版本号和新元设计者的数量而增加。 截止目前, 我们已经有了不连续P_0元, 线性P_1元, 二次P_2 Lagrange元, 不连续P_1元, Raviart-Thomas 元以及一些其他元如气泡元。

第 2 章

入门指南

以**Poisson**方程为例，我们看一下FreeFem++ 是如何求解偏微分方程问题。对于一个给定的函数$f(x,y)$，找到一个函数$u(x,y)$ 满足如下条件：

$$-\Delta u(x,y) = f(x,y) \quad \text{对于所有}(x,y)\in\Omega \tag{2.1}$$
$$u(x,y) = 0 \quad \text{对于所有在}\partial\Omega(x,y)\text{上的}(x,y) \tag{2.2}$$

其中， $\partial\Omega$是有界开集Ω的边界，$\Omega\subset\mathbb{R}^2$， 并且$\Delta u=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}$。
下面我们给出一个FreeFem++ 程序，这个程序是用来计算当$f(x,y)=xy$，Ω是一个单位圆盘时的解u。其中边界$C=\partial\Omega$是：

$$C=\{(x,y)|\ x=\cos(t),\ y=\sin(t),\ 0\le t\le 2\pi\}$$

注意到在FreeFem++ 中，区域Ω是用它的边界来描述的，同时我们假定这个区域是在边界的左侧的，这些可以通过参数化进行定向。在图2.2 中，我们可以看到u的等高线，这个等高线是通过 **plot** 函数得到的(看下列程序的第11行)。

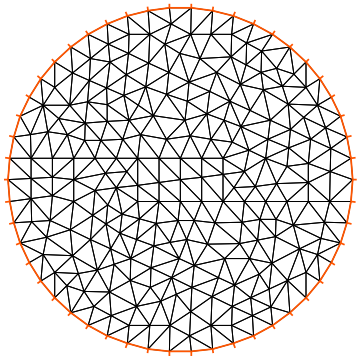


图 2.1: 通过build(C(50))得到网格Th

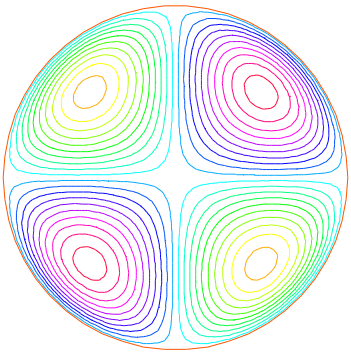


图 2.2: 通过plot(u)得到等高线

Example 2.1

```
//      定义边界
```

```
1: border C(t=0,2*pi){x=cos(t); y=sin(t);}
   //    区域的三角形划分Th是在边界的左侧
2: mesh Th = buildmesh (C(50));
                                          //    Vh表示定义在Th上的有限元空间
3; fespace Vh(Th,P1);
4: Vh u,v;                                //    u和v是分段-p1连续函数
5: func f= x*y;                           //    定义函数f
6: real cpu=clock();                      //    以秒计时
7: solve Poisson(u,v,solver=LU) =         //    定义PDE
8:    int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))//    双线性部分
9:    - int2d(Th)( f*v)                   //    右边
10:    + on(C,u=0)  ;                     //    Dirichlet边界条件
11: plot(u);
12: cout << " CPU time = " << clock()-cpu << endl;
```

注：第七行的限定词solver=LU可以省略，默认情况下，我们只需要有一个可以利用的多波前LU即可；包含有clock的语句也可以省略；同时，我们也要注意到 FreeFem++ 与数学语言的联系有多么的紧密。第8行和第9行相对应的变分方程是：

$$\int_{T_h}(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\frac{\partial v}{\partial y})\mathrm{d}x\mathrm{d}y=\int_{T_h} fv\mathrm{d}x\mathrm{d}y$$

所有的v都是定义在有限元空间V_h上的，并且在边界C上的值为0。

Exercise ：把P1改成P2，在运行一下程序。

2.0.1 在FEM中，FreeFem++ 是如何运行的？

第一个例子给出了在不需要考虑有限元算法（FEM）的情况下，FreeFem++ 是如何运行的。现在我们一行一行地解释这个程序。

第一行: 边界Γ是用一个关于x和y的参数方程来描述的。当 $\Gamma=\sum_{j=0}^{J}\Gamma_j$时，每条曲线$\Gamma_j$必须被指定，而且在节点之外不允许曲线$\Gamma_j$相交。

我们可以添加关键词“label”来定义边界族以方便后面的使用(例如当有多种边界条件时)。因此，单位圆也可以用同样的标签描述成两个半圆。

```
border Gamma1(t=0,pi)   {x=cos(t); y=sin(t); label=C}
border Gamma2(t=pi,2*pi){x=cos(t); y=sin(t); label=C}
```

我们既可以用边界的名字(例如 Gamma1)来指代这个边界，也可以用标签(在这里面是C)来指代它，甚至也可以用内部的数字编码来指代它，这里面用1表示第1个半圆，用2表示第2个半圆(在 **??**中还有一些其他的例子)。

第2行: 区域Ω上的三角剖分$\mathcal{T}_h$是通过 **buildmesh**(C(50)) 来自动划分的，在图2.1中，用边界 C上的50个节点来生成这个划分。
我们假定区域是在边界的左侧的，这些可以通过边界的参数化进行定向。因此，如果在区域Ω中添加椭圆形的孔可以通过下述语句来实现。

```
border C(t=2*pi,0){x=0.1+0.3*cos(t); y=0.5*sin(t);}
```

如果错误地写成：

```
border C(t=0,2*pi){x=0.1+0.3*cos(t); y=0.5*sin(t);}
```

那么里面的椭圆会被划分为三角形。
自动划分网格基于Delaunay-Voronoi算法。可以通过增加边界Γ的节点数来细化网格，例如， `buildmesh(C(100))`，这是因为内部的顶点是由边界上节点的密度来决定的。如果有一个给定的函数 `f`，也可以应用这个函数重新进行网格划分，只要使用 `adaptmesh(Th,f)` 语句即可。

名称$\mathcal{T}_h$ (在`FreeFem++` 中的`Th`)是指图2.1中的三角形族$\{T_k\}_{k=1,\cdots,n_t}$。习惯上，h是指网格的尺寸，n_t是指$\mathcal{T}_h$中三角形的数量，而n_v是指节点的个数。但是，一般情况下我们不需要明确地使用它们。如果，Ω 不是一个多边形区域，有一个“外壳(skin)”存留在精确区域Ω和它的近似区域 $\Omega_h = \cup_{k=1}^{n_t} T_k$之间。然而，我们注意到$\Gamma_h = \partial\Omega_h$的所有角都是在$\Gamma$上的。

第3行：一般情况下，有限元空间是指单元上的多项式函数空间，边界和节点上有一定的匹配特性。但是在这里， `fespace Vh(Th,P1)`定义的V_h为一个连续函数空间，并且这些连续函数都是在T_h中的每个三角形上关于x，y的仿射函数。由于它是一个有限维的线性向量空间，因此我们可以找到这个空间的基。它的典范基是由函数组成的，被称为帽子函数(*hat functions*)ϕ_k。ϕ_k是连续分段的仿射函数，它在一个节点上的值为1，而其它的值为0。图 2.4 [1] 描述了一个典型的帽子函数。则我们有：

$$V_h(\mathcal{T}_h, P_1) = \left\{ w(x,y) \;\middle|\; w(x,y) = \sum_{k=1}^{M} w_k \phi_k(x,y),\ w_k\text{是实数} \right\} \tag{2.3}$$

其中M是V_h的维数，也是节点的个数。w_k是w的自由度，而M是自由度的总数。
我们也称这个有限元方法的节点为如上提到的节点。

`FreeFem++` 在2d中可以执行下述单元(在第6章中有全部的描述)
`P0` 分段常值，
`P1` 连续分段线性，
`P2` 连续分段二次，
`P3` 连续分段三次， (需要 `load "Element_P3"`)
`P4` 连续分段四次， (需要 `load "Element_P4"`)
`RT0` Raviart-Thomas分片常数，
`RT1` 1阶Raviart-Thomas 分片常数 (需要 `load "Element_Mixte"`)
`BDM1` 1阶Brezzi-Douglas-Marini 分片常数 (需要 `load "Element_Mixte"`)

[1] 最简单的定义ϕ_k方法是利用点$q = (x,y) \in T$的重心坐标$\lambda_i(x,y),\ i = 1,2,3$，其中

$$\sum_i \lambda_i = 1, \quad \sum_i \lambda_i \boldsymbol{q}^i = \boldsymbol{q}$$

同时，$q^i,\ i = 1,2,3$是T的3个顶点。显然，ϕ_k在T上的限制恰巧是λ_k。

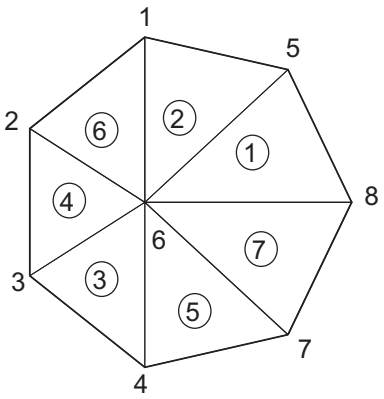


图 2.3: 网格 Th

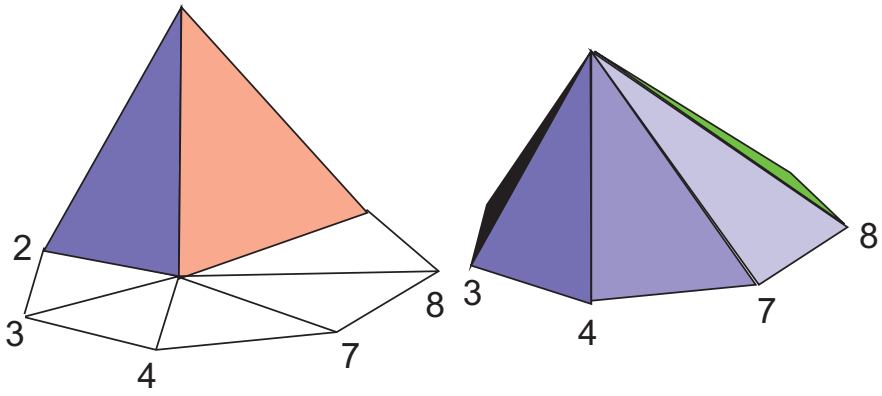


图 2.4: ϕ_1(左侧)和ϕ_6的图像

RT0Ortho 0阶第一类Nedelec 分片常数
RT1Ortho 1阶第一类Nedelec 分片常数 (需要 load "Element_Mixte")
BDM1Ortho 1阶Brezzi-Douglas-Marini 分片常数 (需要 load "Element_Mixte")
P1nc 分段线性非协调，
P1dc 分段线性间断，
P2dc 分段二次间断，
P2h 二次齐次连续，(不含 P1)
P3dc 分段三次间断，(需要 load "Element_P3dc")
P4dc 分段四次间断，(需要 load "Element_P4dc")
P1b 分片线性连续函数加bubble函数,
P2b 分片二次连续函数加bubble函数.
Morley Morley有限元 (需要 load "Morley")
P2BR P2 Bernardi-Raugel有限元 (需要 load "BernadiRaugel.cpp")
P0edge 每条边上为有限元常数
P1edge to P5edge 边上为有限元多项式 (需要 load "Element_PkEdge")
...
FreeFem++ 在3d中可以执行下述单元(在第6章有全部的描述)
P03d 分段常值,
P13d 连续分段线性，
P23d 连续分段二次，
RT03d Raviart-Thomas分段常值,
Edge03d Nedelec 边界元
P1b3d 分片线性连续函数加bubble函数,
...
在UNIX终端中，可以通过examples++-tutorial得到完整列表。

```
FreeFem++ dumptable.edp
grep TypeOfFE lestables
```

要注意的是其他单元也能够很容易的添加进去。

第3步：设置问题

第4行： `Vh u,v`表示声明u和v，并且u和v是如上所说的近似结果，表达式为：

$$u(x,y) \simeq u_h(x,y) = \sum_{k=0}^{M-1} u_k \phi_k(x,y) \tag{2.4}$$

第5行： 右端的 `f`是由关键词 **`func`**定义的。
第7－9行： 定义方程(2.1)的双线性形式和它的Dirichlet边界条件(2.2)。
这个变分公式是由下列步骤推导出来：先在方程(2.1)两端同时乘上$v(x,y)$，然后在Ω上积分，则有：

$$-\int_\Omega v\Delta u \,\mathrm{d}x\mathrm{d}y = \int_\Omega vf \,\mathrm{d}x\mathrm{d}y$$

由格林公式，问题转变为找到合适的u使得：

$$a(u,v) - \ell(f,v) = 0 \qquad \forall v\text{在}\partial\Omega\text{上满足}v = 0 \tag{2.5}$$

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x\mathrm{d}y, \quad \ell(f,v) = \int_\Omega fv \,\mathrm{d}x\mathrm{d}y \tag{2.6}$$

在`FreeFem++` 中，**Poisson**问题只能用下述方式声明：

```
Vh u,v; problem Poisson(u,v) =
```

并且之后用以下方式解决：

```
...
  Poisson;                                    //      问题在这里被解决
...
```

或者可以同时声明和求解：

```
Vh u,v; solve Poisson(u,v) =int(...
```

(2.5)是根据 **`dx`**(u) $= \partial u/\partial x$, **`dy`**(u) $= \partial u/\partial y$以及下述式子写出的。

$$\int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x\mathrm{d}y \longrightarrow \texttt{int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))}$$

$$\int_\Omega fv \,\mathrm{d}x\mathrm{d}y \longrightarrow \texttt{int2d(Th)(f*v)} \qquad \text{(这里要注意到}u\text{是未使用的)}$$

在`FreeFem++` 中，**双线性项和线性项不能在相同的积分下**；实际上，在用`FreeFem++`构建线性系统时，会通过检查未知项(在这里是u)和测试函数(在这里是v)是否同时存在来找到会产生双线性形式的积分。

第4步：求解和可视化

第6行： 将现在的时间储存到实数变量 `cpu`中，单位为秒。
第7行： 问题求解。
第11行： 输出可视化图像，如图2.2中所示(在第 7.1节中，该视图可以用zoom，postscript等命令进行调节)。
第12行： 在控制台输出运行时间(不是计算图形个数)。要注意的是，`FreeFem++` 中的一些语句和C++中的语句比较相像；使用者不需要为了使用`FreeFem++` 而学习C++，

但是它可以帮助我们猜测FreeFem++ 中允许什么样的语句。

推广到矩阵和向量形式

我们可以用FreeFem++ 求解下列类型的线性系统：

$$\sum_{j=0}^{M-1} A_{ij}u_j - F_i = 0, \quad i = 0, \cdots, M-1; \qquad F_i = \int_\Omega f\phi_i \,\mathrm{d}x\mathrm{d}y \tag{2.7}$$

利用(2.4)，同时在(2.5)中用ϕ_i来替换v可以得到上述线性系统。这里在处理Dirichlet边界条件时是采用惩罚方法，如果i节点在位移边界上，设置$A_{ii} = 10^{30}$和$F_i = 10^{30} * 0$。要注意的是，数字10^{30}被称为 tgv(*très grande valeur*)，一般情况下，我们也有可能改变这个值，查看索引项 solve!tgv=。

矩阵$A = (A_{ij})$称为刚度矩阵。

如果用户想要直接得到A，可查看第6.12节第167页的内容。

```
varf a(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
              + on(C,u=0) ;
matrix A=a(Vh,Vh);                                    //      刚度矩阵
```

(2.7)中的向量F 也可以通过以下方式构建：

```
varf l(unused,v) = int2d(Th)(f*v)+on(C,unused=0);
Vh F;  F[] = l(0,Vh);                          //     F[]是和函数F有关的向量
```

问题可以通过以下求解：

```
u[]=A^-1*F[];                                  //     u[]是和函数u有关的向量
```

Note 2.1 在这里， u和 F是有限元函数，并且 $u[]$和 $F[]$给出了一连串的关联值($u[] \equiv (u_i)_{i=0,\ldots,M-1}$和 $F[] \equiv (F_i)_{i=0,\ldots,M-1}$)。因此我们有：

$$\mathtt{u}(x,y) = \sum_{i=0}^{M-1} \mathtt{u}[][i]\phi_i(x,y), \qquad \mathtt{F}(x,y) = \sum_{i=0}^{M-1} \mathtt{F}[][i]\phi_i(x,y)$$

其中$\phi_i, i = 0\ldots, , M-1$像方程(2.3)中的一样，是$Vh$的基函数，并且$M =$ Vh.ndof是自由度的总数量(也就是Vh空间的维数)。

我们可以用UMFPACK来求解线性系统(2.7)，除非像下列语句中一样明确提及了其它选项：

```
Vh u,v; problem Poisson(u,v,solver=CG) = int2d(...
```

这意味着，在这个被声明的Poisson问题下，(2.7)将会用共轭梯度法来求解。

2.0.2 FreeFem++ 的特点

FreeFem++ 的语言是类型化、多样化的，在宏生成下有可重入性(具体见9.12)。每个变量必须在说明语句中声明一个确定的类型，每个说明语句用标点符号“;”与下一个分离。默认情况下，它的语法是在C++的基础上增加了一些与 TEX非常相似的语法。对于专业人士来说，他们使用 FreeFem++ 的一个关键原因是FreeFem++ 很少生成一个内部有限元数组；这一点会让它的运行速度变快，因此，在执行速度方面，FreeFem++ 很少会被打败，除了在语言解释上会造成时间损失(我们可以通过varf，以及将problem替换成矩阵以减少时间成本)。

2.1 开发周期: 编辑–运行/作图–修正

A. Le Hyaric为FreeFem++ 提供了一个完整的环境。本书也给出了许多例子和教程，用户可以通过这些例子和教程来学习它们。在本书的下一章我们给出了一些例子的解释。如果你是一名FEM 初学者，你可能需要去阅读有关变分方程的书。

开发周期包含下列步骤：

建模: 用FreeFem++ 将PDE从强形式转变为弱形式，用户必须熟悉变分方程；为了使内部矩阵不变，也需要了解变分方程的可重用性；一个典型的例子是隐式时变热传导方程；内部矩阵仅需一次分解，而FreeFem++ 将完成这些工作。

编程: 在文本编辑器中，写出FreeFem++ 的代码。

运行: 运行代码(在这里，代码被写在mycode.edp文件中)。要注意到，我们也可以通过下列代码在终端模式中运行原有代码：

```
% FreeFem++ mycode.edp
```

可视化: 当FreeFem++ 运行时，可以使用关键词 plot来展示函数，也可以用 wait=1在每一点暂停程序，还可以使用 ps="toto.eps"来生成一个附录文件以保存结果。

调试: 像从 **wait=true**到 **wait=false**一样，全局变量“debug”可以帮助我们进行调试。

```
bool debug = true;
border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
border b(t=0,2*pi){ x=0.8+0.3*cos(t); y=0.3*sin(t);label=2;}
plot(a(50)+b(-30),wait=debug);                          //    画出边界来观察交叉点
                                          //    在b在0.8到0.3进行变化后，  需要一次鼠标点击
mesh Th = buildmesh(a(50)+b(-30));
plot(Th,wait=debug);                                    //    画出Th之后需要一次鼠标点击
fespace Vh(Th,P2);
Vh f = sin(pi*x)*cos(pi*y);
plot(f,wait=debug);                                                  //    画出函数f
Vh g = sin(pi*x + cos(pi*y));
plot(g,wait=debug);                                                  //    画出函数g
```

将debug改成false会使得作图连续地进行；喝咖啡时，观察屏幕上图像的变化将会是一个有趣的经历。

在控制窗口会出现错误信息。C++代码的模板结构偶尔会造成这些错误信息可读性较差（但是我们已经尽力了）。 尽管如此，这些错误信息依然能够正确显示。例如，如果你忘记了圆括号：

```
bool debug = true;
mesh Th = square(10,10;
plot(Th);
```

那么你会从FreeFem++中得到如下信息：

```
    2 : mesh Th = square(10,10);
 Error line number 2, in file bb.edp, before  token ;
parse error
```

```
  current line = 2
Compile error : parse error
        line number :2, ;
error Compile error : parse error
        line number :2, ;
 code = 1
```

如果你使用同样的符号两次：

```
real aaa =1;
real aaa;
```

那么你会得到如下信息：

```
    2 : real aaa; The identifier aaa exists
          the existing type is <Pd>
          the new  type is <Pd>
```

如果你发现程序没有按照你想要的方向进行，你可以在控制窗口中使用 **cout**来显示变量的值，就像你在C++中做得一样。

下述例子便是这样的功能：

```
...;
fespace Vh...; Vh u;...
cout<<u;...
matrix A=a(Vh,Vh);...
cout<<A;
```

或者你可以像在C++中一样，用“ //”进行注释。例如：

```
real aaa =1;
                                        //     实数aaa;
```

第 3 章

实例学习

本章主要是针对一些不喜欢看说明书的人。许多简单的例子非常典型，却涵盖了FreeFem++的很多性能。本章涉及到的建模部分会在第9章中继续讲解，那一章会继续深度讨论一些物理学、工程学以及金融学上的PDE 问题。

3.1 薄膜问题

摘要 这里，我们来学习，当将*Laplace*算子应用于荷载作用下的膜平衡问题时，如何求解*Dirichlet*问题和*(/*或*)Dirichlet Neumann* 混合问题。我们也会检验该方法的精确度以及与其他图形包的接口。

弹性薄膜Ω固定在一平面刚性支撑Γ上，同时在每个面积元$\mathrm{d}x = \mathrm{d}x_1\mathrm{d}x_2$ 上施加外力$f(x)dx$。薄膜的垂直位移$\varphi(x)$ 可以通过求解Laplace方程来获得：

$$-\Delta\varphi = f \ \text{ in } \ \Omega.$$

由于薄膜固定在它的平面支撑上，因此我们有：

$$\varphi|_\Gamma = 0.$$

如果这个支撑不在一个平面上，但是它每点的高度为$z(x_1, x_2)$，那么它的边界满足非齐次Dirichlet边值条件。

$$\varphi|_\Gamma = z.$$

如果薄膜边界Γ的一部分边界Γ_2不是固定在支撑上而是自由滑动，那么根据薄膜的刚性，Γ_2上的任一点沿Γ_2的单位法向量n 的方向导数为0，因此边界条件是：

$$\varphi|_{\Gamma_1} = z, \qquad \frac{\partial\varphi}{\partial n}|_{\Gamma_2} = 0$$

其中，$\Gamma_1 = \Gamma - \Gamma_2$；同时，我们要注意到 $\frac{\partial\varphi}{\partial n} = \nabla\varphi \cdot n$，并且 Laplace算子$\Delta$满足：

$$\Delta\varphi = \frac{\partial^2\varphi}{\partial x_1^2} + \frac{\partial^2\varphi}{\partial x_2^2}.$$

在这种“混合边界条件”下，问题存在唯一解(参考(1987)，Dautray-Lions (1988)，Strang (1986)以及Raviart-Thomas (1983))；注意到，φ应该使能量达到最小状态，因此我们有：

$$E(\phi) = \min_{\varphi - z \in V} E(v), \quad \text{并且} \quad E(v) = \int_\Omega (\frac{1}{2}|\nabla v|^2 - fv)$$

其中V是Sobolev空间$H^1(\Omega)$的子空间，并且V中的函数在Γ_1上的迹为0。注意到(在这里$x \in \mathbb{R}^d,\ d = 2$)：

$$H^1(\Omega) = \{u \in L^2(\Omega)\ :\ \nabla u \in (L^2(\Omega))^d\}$$

在变分法中，由PDE的弱形式以及变分公式可知，最小值必须满足：

$$\int_\Omega \nabla\varphi \cdot \nabla w = \int_\Omega fw \quad \forall w \in V$$

如果二阶导数存在，由分部积分(Green公式)可知，它和PDE是等价的。

警告 FreeFem++ 与freefem+不同的是，freefem+同时具有弱形式和强形式，而FreeFem++ 只执行弱形式。如果你不知道问题的弱形式(也就是变分公式)的话，你不可能进一步地使用这个软件：你只能阅读一些书籍，或是寻求他人的帮助，否则就只能跳过这个问题。现在，如果你想要解决像$A(u,v) = 0,\ B(u,v) = 0$之类的PDE系统，不要关闭这个手册，因为在弱形式下，它满足：

$$\int_\Omega (A(u,v)w_1 + B(u,v)w_2) = 0 \quad \forall w_1, w_2 ...$$

例子 有一个椭圆，它的半长轴的长度为$a = 2$，半短轴的长度为1。在它的表面施加$f = 1$的力。在 FreeFem++ 下，这个例子的程序为：

Example 3.1 (membrane.edp)

```
real theta=4.*pi/3.;
real a=2.,b=1.;                                  //    半长轴和半短轴的长度
func z=x;

border Gamma1(t=0,theta)    { x = a * cos(t); y = b*sin(t); }
border Gamma2(t=theta,2*pi) { x = a * cos(t); y = b*sin(t); }
mesh Th=buildmesh(Gamma1(100)+Gamma2(50));

fespace Vh(Th,P2);                               //    P2协调三角形FEM
Vh phi,w, f=1;

solve Laplace(phi,w)=int2d(Th)(dx(phi)*dx(w) + dy(phi)*dy(w))
                - int2d(Th)(f*w) + on(Gamma1,phi=z);
plot(phi,wait=true, ps="membrane.eps");          //    画出phi
plot(Th,wait=true, ps="membraneTh.eps");         //    画出Th

savemesh(Th,"Th.msh");
```

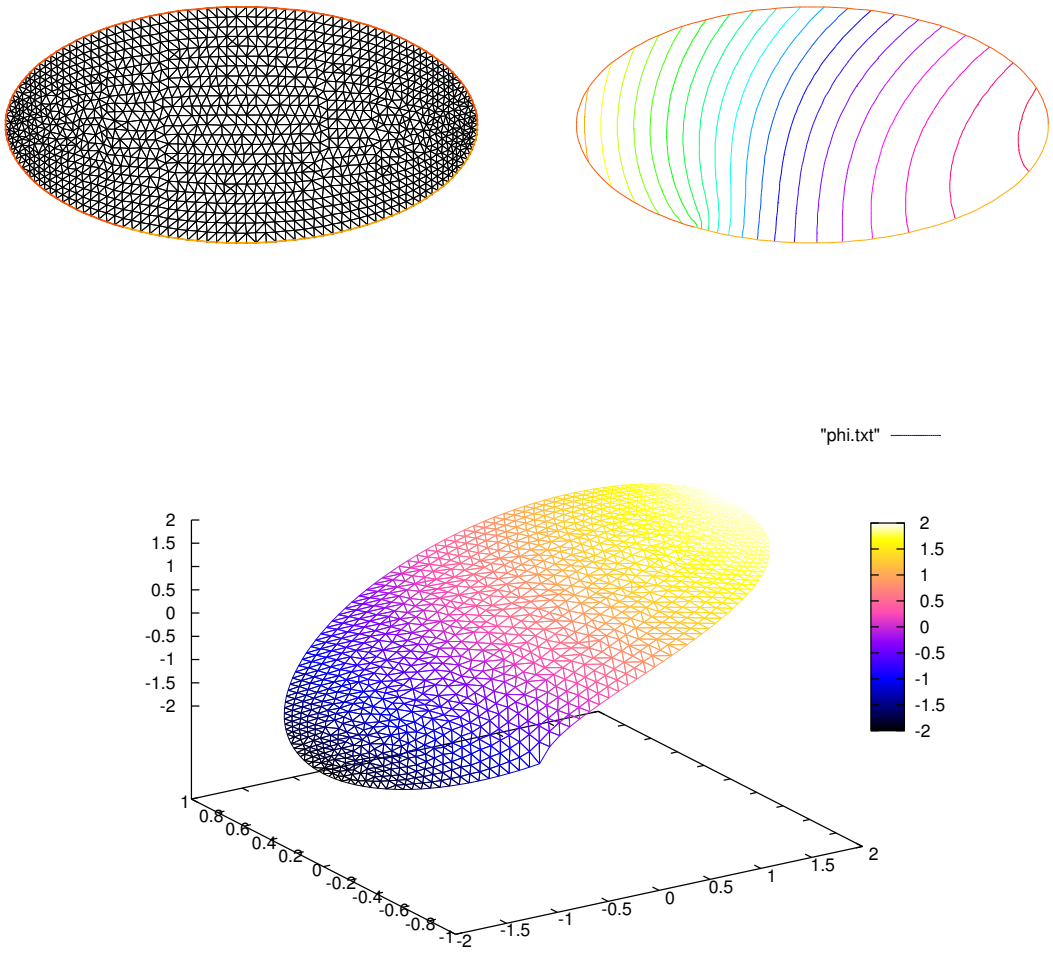


图 3.1: membrane deformation 的网格和等高线. 下方: 根据 `FreeFem++` 生成的文件, 由 `gnuplot` 绘制的 3D 图像

一个三角剖分由关键词 buildmesh 建立。这个关键词调用了一个基于得劳内测试的三角剖分子程序。 德劳内测试首先是由边界点三角化, 然后通过细分边加入内部点。 三角划分的精细程度由最近的一条边界的尺寸决定.

偏微分方程的离散化是通过使用三角剖分上的二阶三角有限元方法实现的。正如前面章节提到的, 将系统离散化后得到一个线性系统，系统的尺度由顶点个数加上三角剖分边的中点个数确定。 这个线性系统可以通过包 UMFPACK 的列主元高斯LU 分解解出. 关键词绘图将显示 $\mathbb{T}_h$ 和 φ (去掉 Th 如果只需得到 φ) 并且限定符 fill=true 替换缺省选择 (彩色的水平线) 用全彩色显示，结果在图 3.1。

```
plot(phi,wait=true,fill=true);                                    //   用全彩显示画 phi
```

接下来检验结果!
一个简单的方法是通过调整参数以确认解。 例如在单位圆 a=1 , $\varphi_e = \sin(x^2+y^2-1)$ 是以下问题的解，当

$$z = 0,\ f = -4(\cos(x^2+y^2-1) - (x^2+y^2)\sin(x^2+y^2-1))$$

除了在 Γ_2 $\partial_n\varphi = 2$ 上而不是零。所以我们将考虑一个非线性诺依曼条件并且解

$$\int_\Omega (\nabla\varphi\cdot\nabla w = \int_\Omega fw + \int_{\Gamma_2} 2w \quad \forall w \in V$$

我们将用两个三角剖分做, 计算 L^2 误差:

$$\epsilon = \int_\Omega |\varphi - \varphi_e|^2$$

并且打印在两种情况下的误差和它们比例的对数，该对数代表了收敛率.

Example 3.2 (membranerror.edp)

```
//     文件 membranerror.edp
verbosity =0;                                                     //     去掉所有错误输出
real theta=4.*pi/3.;
real a=1.,b=1.;                                                   //     半长轴和短半轴的长度
border Gamma1(t=0,theta)     { x = a * cos(t); y = b*sin(t); }
border Gamma2(t=theta,2*pi)  { x = a * cos(t); y = b*sin(t); }

func f=-4*(cos(x^2+y^2-1) -(x^2+y^2)*sin(x^2+y^2-1));
func phiexact=sin(x^2+y^2-1);

real[int] L2error(2);                                             //     双值数组
for(int n=0;n<2;n++)
{
  mesh Th=buildmesh(Gamma1(20*(n+1))+Gamma2(10*(n+1)));
  fespace Vh(Th,P2);
   Vh phi,w;

  solve laplace(phi,w)=int2d(Th)(dx(phi)*dx(w) + dy(phi)*dy(w))
    - int2d(Th)(f*w) - int1d(Th,Gamma2)(2*w)+ on(Gamma1,phi=0);
  plot(Th,phi,wait=true,ps="membrane.eps");                       //   画 Th 和 phi

  L2error[n]= sqrt(int2d(Th)((phi-phiexact)^2));
}
```

```
for(int n=0;n<2;n++)
 cout << " L2error " << n << " = "<<  L2error[n] <<endl;

cout <<" convergence rate = "<< log(L2error[0]/L2error[1])/log(2.)  <<endl;
```

输出结果是

```
 L2error 0 = 0.00462991
 L2error 1 = 0.00117128
 convergence rate = 1.9829
times: compile 0.02s, execution 6.94s
```

我们找到比率 1.93591, 和理论预计的 3 不够近。 由于几何结构是多边形, 在 $O(h^2)$ 近似下损失了一阶。

现在如果你对 FreeFem++ 制作的图 .eps 不甚满意，且想用其它其它画图工具, 那么你必须先将解存储在一个文件里, 这与在 C++情况下非常相似。如果你不保存三角剖分，储存的解将毫无用处，因此你必须依照下列做法

```
    {
      ofstream ff("phi.txt");
      ff << phi[];
     }
   savemesh(Th,"Th.msh");
```

对于三角剖分名字是重要的: 扩展名决定了格式。

如果依然不能令你满意，请尝试使用gnuplot 工具将产生图 3.2。

```
                                                    //   建一个 gnuplot 数据文件
{ ofstream ff("graph.txt");
   for (int i=0;i<Th.nt;i++)
   { for (int j=0; j <3; j++)
       ff<<Th[i][j].x  << "    "<< Th[i][j].y<< "  "<<phi[][Vh(i,j)]<<endl;
    ff<<Th[i][0].x  << "    "<< Th[i][0].y<< "  "<<phi[][Vh(i,0)]<<"\n\n\n"
   }
}
```

使用有限个元素编号, 其中 Wh(i,j) 是第 i个自由度为j^{Th} 的三角形的全局索引。
然后打开 gnuplot 并且执行

```
set palette rgbformulae 30,31,32
splot "graph.txt" w l pal
```

这两行在 P2 和 P1 下可行, 但 P1nc 不行。 因为 P2 或者 P1 的前三个自由度在顶点上但 P1nc 不是。

3.2 热交换

摘要 这里我们会学更多关于 几何结构输入和 三角剖分文件, 及读写 等操作。

问题 外壳 C_0 内有 $\{C_i\}_{1,2}$两个热导体。 一个维持常温 u_1, 另一个的热导率 κ_2 比 C_0 大五倍。 我们假设外壳 C_0 的边界维持 $20°C$ 不变并且经历了足够长时间达到了热平衡。 为了知道边界 Ω 任何点 x 上的温度 $u(x)$, 我们必须求解

$$\nabla \cdot (\kappa \nabla u) = 0 \quad \text{in} \quad \Omega, \quad u_{|\Gamma} = g$$

其中 Ω 是 C_0 的内部减去导体 C_1, Γ 是 Ω的边界, 也就是 $C_0 \cup C_1$, 这里 g 是关于 x 的任意函数,在 C_i上等于 u_i 。 第二个方程具有下面的简化形式:

$$u = u_i \text{ on } \quad C_i, \quad i = 0, 1.$$

这个问题的变分公式是在 Γ 上有零迹的函数子空间$H_0^1(\Omega) \subset H^1(\Omega)$.

$$u - g \in H_0^1(\Omega) \ : \ \int_\Omega \nabla u \nabla v = 0 \quad \forall v \in H_0^1(\Omega)$$

这里我们假设 C_0 是以原点为圆心 5 为半径的圆, C_i 是矩形, C_1 维持 $u_1 = 60°C$ 下的恒温。

Example 3.3 (heatex.edp)

```
//    file heatex.edp
int C1=99, C2=98;                                  //    可以是任何值例如 ≠0 和 C1≠C2
border C0(t=0,2*pi){x=5*cos(t); y=5*sin(t);}

border C11(t=0,1){ x=1+t;  y=3;      label=C1;}
border C12(t=0,1){ x=2;    y=3-6*t;  label=C1;}
border C13(t=0,1){ x=2-t;  y=-3;     label=C1;}
border C14(t=0,1){ x=1;    y=-3+6*t; label=C1;}

border C21(t=0,1){ x=-2+t; y=3;      label=C2;}
border C22(t=0,1){ x=-1;   y=3-6*t;  label=C2;}
border C23(t=0,1){ x=-1-t; y=-3;     label=C2;}
border C24(t=0,1){ x=-2;   y=-3+6*t; label=C2;}

plot(    C0(50)                                    //    为了观察定义域的边界
        + C11(5)+C12(20)+C13(5)+C14(20)
        + C21(-5)+C22(-20)+C23(-5)+C24(-20),
        wait=true, ps="heatexb.eps");

mesh Th=buildmesh(    C0(50)
                    + C11(5)+C12(20)+C13(5)+C14(20)
                    + C21(-5)+C22(-20)+C23(-5)+C24(-20));
plot(Th,wait=1);

fespace Vh(Th,P1); Vh u,v;
Vh kappa=1+2*(x<-1)*(x>-2)*(y<3)*(y>-3);
solve a(u,v)= int2d(Th)(kappa*(dx(u)*dx(v)+dy(u)*dy(v)))
                +on(C0,u=20)+on(C1,u=60);
plot(u,wait=true, value=true, fill=true, ps="heatex.eps");
```

注释:

- C0 关于 t 是逆时针方向, 而 C1 是顺时针方向, C2 是逆时针方向，因此 C1 被 buildmesh看成一个洞.

- C1 和 C2 是通过连接线段形成的。 为了把它们在同一个逻辑单元里聚合在一起, 为了以一个易读的方式输入边界条件, 我们给边界分配一个标签。 正如前面说的, 边界在程序中有一个与其顺序一致内部编码 （可通过在以上代码中加一个 cout<<C22 检验）。 这一点对于理解一个网格如何能输出到文件如何再读取非常重要（见下文）。

- 像往常一样, 网格密度由指定的每条边界的顶点个数决定。虽然没有办法改变顶点的分布（程序统一分配），但是一段边界总是可以被切成两段或更多段, 例如C12 可以被 C121+C122 替代:

```
                                //    border C12(t=0,1) x=2; y=3-6*t; label=C1;
border C121(t=0,0.7){ x=2;     y=3-6*t;  label=C1;}
border C122(t=0.7,1){ x=2;     y=3-6*t;  label=C1;}
... buildmesh(.../* C12(20) */ + C121(12)+C122(8)+...);
```

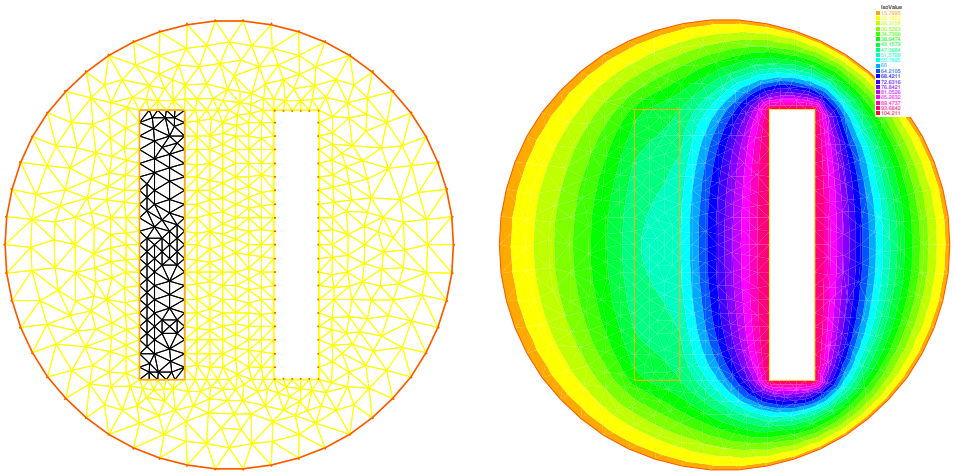


图 3.2: 热交换器

练习 利用问题关于坐标的对称性; 将区域进行一般的三角划分, 并在纵坐标上设置 Dirichlet 条件, 横坐标上设置诺依曼条件。

写和读三角剖分文件 假设我们在前面程序的结尾加入一行：

```
savemesh(Th,"condensor.msh");
```

然后我们写一个相似的程序并希望从那个文件中读取网格，随后问题按下述语句求解：

```
mesh Sh=readmesh("condensor.msh");
fespace Wh(Sh,P1); Wh us,vs;
solve b(us,vs)= int2d(Sh)(dx(us)*dx(vs)+dy(us)*dy(vs))
                +on(1,us=0)+on(99,us=1)+on(98,us=-1);
plot(us);
```

需要注意的是,虽然边界的名字丢失了但是它们的内部编码 （对于 C0） 或者标签 （对于 C1 和 C2）均被保留了下来。

3.3 声学

摘要 这里我们将研究不适定问题和特征值问题。
静态空气中的气压变化取决于波动方程:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \Delta u = 0.$$

当解为单色波 （并且依赖于边界和初始条件）时, 其解u 如 $u(x,t) = Re(v(x)e^{ikt})$ 形式，其中 v 为 Helmholtz 方程的解:

$$\begin{aligned} k^2 v + c^2 \Delta v = 0 \quad &\text{in} \quad \Omega, \\ \frac{\partial v}{\partial n}|_\Gamma = g. \end{aligned} \tag{3.1}$$

其中 g 是源项。 需要注意的是拉普拉斯算子前面的 “+” 符号时, $k > 0$ 为实数。当 $\frac{c}{k}$ 为某些值时，会导致前面的符号不定，以至问题不适定。 “共振”现象就是这样的情况。

在共振情况，即使 $g = 0$也会出现非零解，因此下面程序可能有效也可能无效:

Example 3.4 (sound.edp)

```
real kc2=1;                                                    //    文件 sound.edp
func g=y*(1-y);

border a0(t=0,1) { x= 5; y= 1+2*t ;}
border a1(t=0,1) { x=5-2*t; y= 3 ;}
border a2(t=0,1) { x= 3-2*t; y=3-2*t ;}
border a3(t=0,1) { x= 1-t; y= 1 ;}
border a4(t=0,1) { x= 0; y= 1-t ;}
border a5(t=0,1) { x= t; y= 0  ;}
border a6(t=0,1) { x= 1+4*t; y= t ;}

mesh Th=buildmesh( a0(20) + a1(20) + a2(20)
        + a3(20) + a4(20) + a5(20) + a6(20));
fespace Vh(Th,P1);
Vh u,v;

solve sound(u,v)=int2d(Th)(u*v * kc2 - dx(u)*dx(v) - dy(u)*dy(v))
                 - int1d(Th,a4)(g*v);
plot(u, wait=1, ps="sound.eps");
```

计算结果显示在图 3.3中。 但是当 $kc2$ 为问题的特征值时, 则其解不唯一: 如果 $u_e \neq 0$ 是一个本征态, 则对于任何形如 $u + u_e$ 的解都为问题的解。 为了找到所有的 u_e ， 添加以下代码:

```
real sigma = 20;                                                //    位移值
                                                  //    OP = A - sigma B ; // 位移矩阵
varf  op(u1,u2)= int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) - sigma* u1*u2 );
varf b([u1],[u2]) = int2d(Th)( u1*u2 ) ;                    //   无边界条件见注释 9.1

matrix OP= op(Vh,Vh,solver=Crout,factorize=1);
matrix B= b(Vh,Vh,solver=CG,eps=1e-20);
```

```
int nev=2;                                          //    指定sigma 附近的特征值数量

real[int] ev(nev);                                    //    存储 nev 特征值
Vh[int] eV(nev);                                    //    存储 nev 特征向量

int k=EigenValue(OP,B,sym=true,sigma=sigma,value=ev,vector=eV,
                 tol=1e-10,maxit=0,ncv=0);
cout<<ev(0)<<" 2 eigen values "<<ev(1)<<endl;
v=eV[0];
plot(v,wait=1,ps="eigen.eps");
```

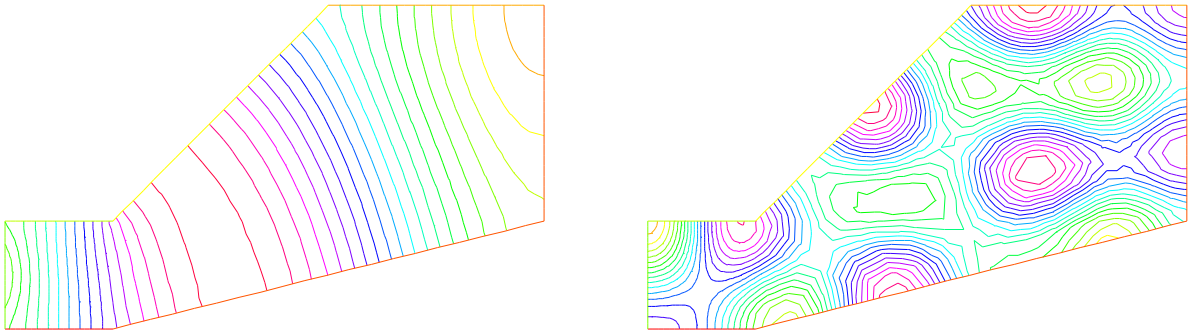


图 3.3: 左:从左侧竖直墙发射的声信号振幅。 右: 特征问题:$-\Delta\varphi = \lambda\varphi$ 且 $\frac{\partial\varphi}{\partial n} = 0$ on Γ中离20最近的第一个特征态 $(\lambda = (k/c)^2 = 19.4256)$。

3.4 热传导

摘要 这里我们将学习如何处理 时变 抛物 问题。我们将演示如何处理 轴对称 问题 和一个 非线性问题。

空气冷却金属板问题 考虑一个带有矩形截面 $\Omega = (0,6) \times (0,1)$的金属板 $(0,Lx) \times (0,Ly) \times (0,Lz)$ 上的温度分布; 初始温度为 $u = u_0 + \frac{x}{L}u_1$ 的金属板置于温度为 u_e 的空气中。在垂直于金属板的平面 $z = Lz/2$ 上, 温度随着坐标 z 几乎不变化; 作为初步近似，假设问题为二维问题。

我们在 Ω 中及时间间隔 (0,T)内求解温度方程。

$$\partial_t u - \nabla \cdot (\kappa \nabla u) = 0 \text{ in } \Omega \times (0,T),$$
$$u(x,y,0) = u_0 + xu_1$$

$$\kappa\frac{\partial u}{\partial n}+\alpha(u-u_e)=0 \text{ on } \Gamma\times(0,T). \tag{3.2}$$

为了模拟恒温调节器，此处扩散系数 κ 将取两个值：一个低于中间水平，另一个则小十倍。 $\alpha(u-u_e)$ 项为空气对流引起的温度损失。 从数学上来讲，这个边值条件是 Fourier (or Robin, or mixed) 边界条件。

变分公式定义在 $L^2(0,T;H^1(\Omega))$ 空间，损失项采用关于时间的欧拉隐式差分近似之后，在 $w\in H^1(\Omega)$空间寻找所有的$u^n(x,y)$:

$$\int_\Omega(\frac{u^n-u^{n-1}}{\delta t}w+\kappa\nabla u^n\nabla w)+\int_\Gamma\alpha(u^n-u_ue)w=0$$

```
func u0 =10+90*x/6;
func k = 1.8*(y<0.5)+0.2;
real ue = 25, alpha=0.25, T=5, dt=0.1 ;

mesh Th=square(30,5,[6*x,y]);
fespace Vh(Th,P1);
Vh u=u0,v,uold;

problem thermic(u,v)= int2d(Th)(u*v/dt + k*(dx(u) * dx(v) + dy(u) * dy(v)))
                + int1d(Th,1,3)(alpha*u*v)
                - int1d(Th,1,3)(alpha*ue*v)
                - int2d(Th)(uold*v/dt) + on(2,4,u=u0);
ofstream ff("thermic.dat");
for(real t=0;t<T;t+=dt){
    uold=u;     //     uold ≡ u^{n-1} = u^n ≡u
    thermic;   //     此处求解热学问题
    ff<<u(3,0.5)<<endl;
    plot(u);
}
```

需要注意的是我们必须从线性项中手动分出双线性部分。同时也要注意在文件 `thermic.dat` 存储所有时间内点 (3,0.5) 温度的方式。 对于其他需要进行一维作图的问题，可采用类似步骤。例如打印 $x\mapsto\frac{\partial u}{\partial y}(x,0.9)$ ，可进行下面的操作：

```
for(int i=0;i<20;i++) cout<<dy(u)(6.0*i/20.0,0.9)<<endl;
```

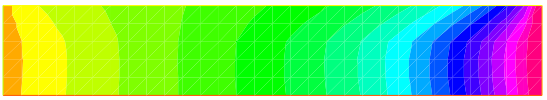
结果显示在图 3.4 中。

3.4.1 轴对称: 截面为圆形的三维棒

现在我们来处理一个圆柱棒而不是平板。为了简便我们取 $\kappa=1$。 在柱面坐标中, 拉普拉斯算子变为 (r 是到轴心的距离, z 是轴向距离, 极角 θ 为与垂直于坐标的固定平面的夹角):

$$\Delta u=\frac{1}{r}\partial_r(r\partial_r u)+\frac{1}{r^2}\partial^2_{\theta\theta}u+\partial^2_{zz}.$$

对称性意味着问题的解不依赖于θ ; 所以区域 Ω 还是矩形 $[0,R]\times[0,1]$ 。 我们沿用 `square()` 中对边界的编号习惯（即1 对应底部的水平线 ...），则问题变为:



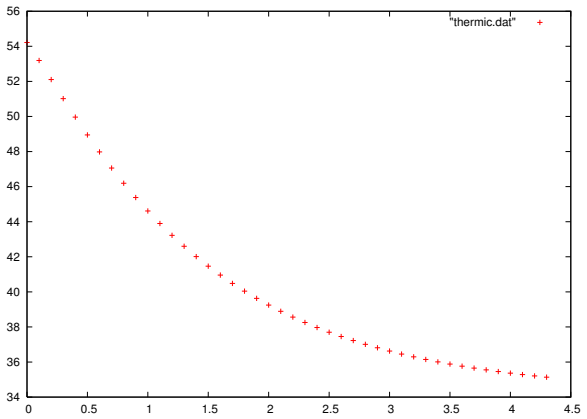


图 3.4: 在 T=4.9 的温度. 右: 在 x=3, y=0.5处温度随着时间的衰减

$$
\begin{aligned}
& r\partial_t u - \partial_r(r\partial_r u) - \partial_z(r\partial_z u) = 0 \text{ in } \Omega, \\
& u(t=0) = u_0 + \frac{z}{L_z}(u_1 - u) \\
& u|_{\Gamma_4} = u_0, \quad u|_{\Gamma_2} = u_1, \quad \alpha(u - u_e) + \frac{\partial u}{\partial n}|_{\Gamma_1 \cup \Gamma_3} = 0.
\end{aligned} \tag{3.3}
$$

注意，已经在偏微分方程上乘了 r 。
在利用隐式格式对时间离散化，时间步为 dt, 在 FreeFem++ 的语法中 r 即是 x ， z 即是y , 问题变为下面的形式:

```
problem thermaxi(u,v)=int2d(Th)((u*v/dt + dx(u)*dx(v) + dy(u)*dy(v))*x)
                + int1d(Th,3)(alpha*x*u*v) - int1d(Th,3)(alpha*x*ue*v)
                - int2d(Th)(uold*v*x/dt) + on(2,4,u=u0);
```

注意双线性形式在 $x = 0$ 处的退化。因为此处的退化，无需在 Γ_1 上加边界条件, 但仍然可以证明 u 的存在性和唯一性。

3.4.2 一个非线性问题：辐射

热量的 辐射 损失与绝对温度四次方成正比(斯特潘定律)。再考虑上对流损失后有如下边界条件:

$$\kappa\frac{\partial u}{\partial n} + \alpha(u - u_e) + c[(u + 273)^4 - (u_e + 273)^4] = 0$$

问题是 非线性的, 必须迭代求解。如果 m 表示迭代指数, 则辐射的半线性化形式如下：

$$\frac{\partial u^{m+1}}{\partial n} + \alpha(u^{m+1} - u_e) + c(u^{m+1} - u_e)(u^m + u_e + 546)((u^m + 273)^2 + (u_e + 273)^2) = 0,$$

这是因为 $a^4 - b^4 = (a - b)(a + b)(a^2 + b^2)$。 针对 $v = u - u_e$ 进行迭代求解。

```
...
fespace Vh(Th,P1);                                          //    有限元空间
real rad=1e-8, uek=ue+273;                                  //    物理常数的定义
Vh vold,w,v=u0-ue,b;
problem thermradia(v,w)
    = int2d(Th)(v*w/dt + k*(dx(v) * dx(w) + dy(v) * dy(w)))
                + int1d(Th,1,3)(b*v*w)
                - int2d(Th)(vold*w/dt) + on(2,4,v=u0-ue);

for(real t=0;t<T;t+=dt){
    vold=v;
    for(int m=0;m<5;m++){
       b= alpha + rad * (v + 2*uek) * ((v+uek)^2 + uek^2);
       thermradia;
    }
}
vold=v+ue; plot(vold);
```

3.5 无旋的风扇气流和热效应

摘要 本节我们将学习如何采用多重网络处理 复杂几何上多物理系统的偏微分方程。 同时学习如何操作 区域指示器, 并且观察从一个网络到另一个的网的投影算子的光滑性。

不可压缩流体 无粘和无旋的不可压缩流体具有如下的速度场:

$$u = \begin{pmatrix} \frac{\partial \psi}{\partial x_2} \\ -\frac{\partial \psi}{\partial x_1} \end{pmatrix}, \quad \text{这里 } \psi \text{ 满足} \quad \Delta\psi = 0$$

这一组等式包含了不可压缩性 （即$\nabla \cdot u = 0$)）和无旋度的性 （即$\nabla \times u = 0$）。
当流体沿着边缘滑动时，法向速度为0，即 ψ 满足:

$$\psi \text{ 在边缘上为一个常数”}$$

也可以在人造边界给定一个速度场，这使得ψ在边界的Dirichlet 数不为常数。

翼面 我们考虑在一致流体中有一个翼形 S 的情形。将无穷远处看成一个大的圆周 C ，流体在圆周上的速度是一致的； 对这个问题建模如下：

$$\Delta\psi = 0 \ \text{ 在 } \ \Omega, \qquad \psi|_S = 0, \quad \psi|_C = u_\infty y, \tag{3.4}$$

这里 $\partial\Omega = C \cup S$

NACA0012 翼面 NACA0012 （这是空气动力学上一个经典的翼形）上表面的一个方程如下：

$$y = 0.17735\sqrt{x} - 0.075597x - 0.212836x^2 + 0.17363x^3 - 0.06254x^4.$$

Example 3.5 (potential.edp) // 文件 potential.edp

```
real S=99;
border C(t=0,2*pi) {  x=5*cos(t);  y=5*sin(t);}
border Splus(t=0,1){  x = t; y = 0.17735*sqrt(t)-0.075597*t
        - 0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4); label=S;}
border Sminus(t=1,0){  x =t; y= -(0.17735*sqrt(t)-0.075597*t
        -0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4)); label=S;}
mesh Th= buildmesh(C(50)+Splus(70)+Sminus(70));
fespace Vh(Th,P2); Vh psi,w;

solve potential(psi,w)=int2d(Th)(dx(psi)*dx(w)+dy(psi)*dy(w))+
  on(C,psi = y) + on(S,psi=0);

plot(psi,wait=1);
```

图3.5显示了流线分布。

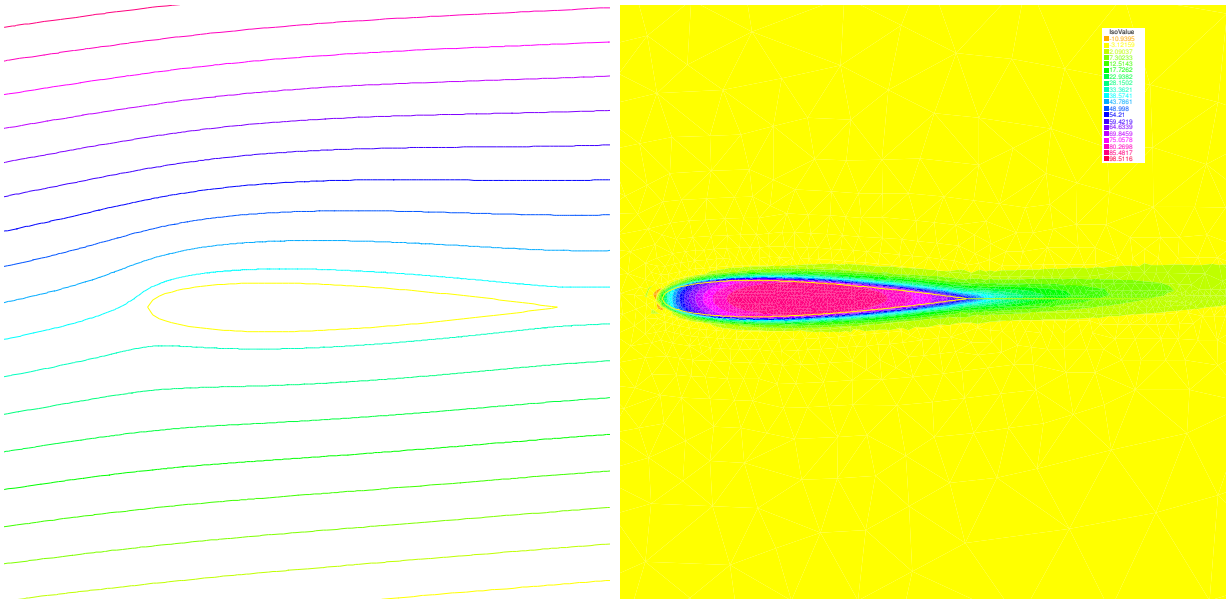


图 3.5: 放大NACA0012翼面的周围可以看到流线 (曲线 ψ = 常数)。 可以通过交互式的图像命令：按“+”键或一直按住右键来获取这样的图像。 右边的图是在时间 T=25 时的温度分布 (现在最大温度是90度代替了原来的120 度)。 注意此处考虑了入射角（详见第9章 ）。

3.5.1 翼面周围的热对流

现在我们假设翼面是热的而空气是冷的。 类似于前面章节，与温度 v有关的热学方程如下：

$$\partial_t v - \nabla \cdot (\kappa \nabla v) + u \cdot \nabla v = 0, \quad v(t=0) = v_0, \quad \frac{\partial v}{\partial n}|_C = 0$$

但是现在的区域是分为S的内部和外部，并且 κ 在空气和钢中取值不同。此外这里的热交换是通过流动实现的, 因此多了 $u \cdot \nabla v$ 这一项。于是有下面的代码（加到之前代码的后面）：

```
...
border D(t=0,2){x=1+t;y=0;}                                    //      使路径上有一个精细的网格
mesh Sh = buildmesh(C(25)+Splus(-90)+Sminus(-90)+D(200));
fespace Wh(Sh,P1); Wh v,vv;
int steel=Sh(0.5,0).region, air=Sh(-1,0).region;
fespace W0(Sh,P0);
W0 k=0.01*(region==air)+0.1*(region==steel);
W0 u1=dy(psi)*(region==air), u2=-dx(psi)*(region==air);
Wh vold = 120*(region==steel);
real dt=0.05, nbT=50;
int i;
problem thermic(v,vv,init=i,solver=LU)= int2d(Sh)(v*vv/dt
                + k*(dx(v) * dx(vv) + dy(v) * dy(vv))
                + 10*(u1*dx(v)+u2*dy(v))*vv)- int2d(Sh)(vold*vv/dt);
for(i=0;i<nbT;i++){
    v=vold; thermic;}
 plot(v);}
```

这里注意

- 在建立网络时通过定义网格参数区域来区分钢和空气部分，并且在每一个连通的Ω部分取一致的整数值;
- 尽管加上的对流项没有用迎风格式，但当雷诺数 $|u|L/\kappa$ 较大 （这是一个典型的长度比例）时，迎风格式是必要的。取对流项的系数为10可以快速地让速度增加10 倍（不然速度太慢就看不到一些现象了）。
- 问题的求解使用Gauss-LU分解求解器，并且当 init$\neq 0$ 时需重复使用 LU 分解，所以在迭代一次之后会变得很快。

3.6 纯对流：盘旋山

摘要 这里我们给出两种方法解决迎风格式下最简单的对流问题。这两种方法分别是 特征-*Galerkin* 和 间断-*Galerkin* 有限元方法。
Ω是以0为中心的单位圆盘；考虑旋转向量场

$$\boldsymbol{u} = [u1, u2], \qquad u_1 = y, \quad u_2 = -x$$

关于u的纯对流方程表示成：

$$\partial_t c + \boldsymbol{u}.\nabla c = 0 \quad \text{in} \quad \Omega \times (0,T) \quad c(t=0) = c^0 \quad \text{in} \quad \Omega.$$

在时间t 和点 x_t 的精确解 $c(x_t,t)$ 为

$$c(x_t,t) = c^0(x,0)$$

这里x_t 是0时刻位于x的粒子在流体中的轨迹。因此， x_t 是下列方程的解

$$\dot{x}_t = u(x_t), \quad , \quad x_{t=0} = x, \quad \text{这里} \quad \dot{x}_t = \frac{\mathrm{d}(t \mapsto x_t)}{\mathrm{d}t}$$

这个常微分方程是可逆的，我们想要时间t时在x处（不是在点x_t）的解，初始点是x_{-t}，我们有

$$c(x,t) = c^0(x_{-t},0)$$

这个过程的时间周期是$T = 2\pi$，周期末的解与初始解相等。

通过特征-Galerkin方法求解 在 FreeFem++ 中，算子convect([u1,u2],dt,c) 可用于计算X下的 $c \circ X$ ，其中X 是由$X(x) = x_{dt}$ 定义的对流场，x_τ 是稳态速度场$\boldsymbol{u} = [u1,u2]$里$\tau = 0$时刻位于点x 的粒子轨迹，因此 x_τ 是下面常微分方程的解:

$$\dot{x}_\tau = u(x_\tau), \quad \boldsymbol{x}_{\tau=0} = x.$$

u 可以是分段常数，此时x_τ 是一条可以被准确计算出来多角曲线，而且当 u 是无源场时，解总是存在。对流问题返回值$c(x_{df}) = C \circ X$。

Example 3.6 (convects.edp) // 文件 convects.edp

```
border C(t=0, 2*pi) { x=cos(t);  y=sin(t); };
mesh Th = buildmesh(C(100));
fespace Uh(Th,P1);
Uh cold, c = exp(-10*((x-0.3)^2 +(y-0.3)^2));

real dt = 0.17,t=0;
Uh u1 = y, u2 = -x;
for (int m=0; m<2*pi/dt ; m++) {
    t += dt;      cold=c;
    c=convect([u1,u2],-dt,cold);
    plot(c,cmm=" t="+t + ", min=" + c[].min + ", max=" +  c[].max);
}
```

注 4 要画3维图像可以通过在画图指令里增加命令“*dim=3*” 实现。

这个方法非常有效，但是有两个限制: a）此方法不是保守的，b）由于正交误差的原因，当$|u|$ 太小时，得到的结果在少数的情况下可能会发散。

间断-Galerkin的FEM解法 间断Galerkin方法充分利用率在边界上c不连续的优点，采用了迎风格式使其公式化。在这里我们采用对偶-P_1^{DC} 公式 （参考[11]）：

$$\int_\Omega (\frac{c^{n+1}-c^n}{\delta t} + u\cdot\nabla c)w + \int_E (\alpha|n\cdot u| - \frac{1}{2}n\cdot u)[c]w = \int_{E_\Gamma^-} |n\cdot u|cw \quad \forall w$$

这里E是内部边的集合，且E_Γ^- 是满足$u\cdot n<0$的边界边的集合 （在我们的情况下这里是没有这样的边界）；最后$[c]$是c穿越对流的一条边界时的跳跃点，c^+表示在定向边界右边的值。

Example 3.7 (convects_end.edp)

```
...                                                    //    文件 convects.edp
fespace Vh(Th,P1dc);

Vh w, ccold, v1 = y, v2 = -x, cc = exp(-10*((x-0.3)^2 +(y-0.3)^2));
real u, al=0.5;  dt = 0.05;

macro n() (N.x*v1+N.y*v2)                              //    缺乏参数的Macro
problem  Adual(cc,w) =
int2d(Th)((cc/dt+(v1*dx(cc)+v2*dy(cc)))*w)
  + intalledges(Th)((1-nTonEdge)*w*(al*abs(n)-n/2)*jump(cc))
                    //    - int1d(Th,C)((n<0)*abs(n)*cc*w) // 因为在边界∂Ω上cc=0
  - int2d(Th)(ccold*w/dt);

for ( t=0; t< 2*pi ; t+=dt)
{
  ccold=cc; Adual;
  plot(cc,fill=1,cmm="t="+t + ", min=" + cc[].min + ", max=" +  cc[].max);
};
real [int] viso=[-0.2,-0.1,0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.1];
plot(c,wait=1,fill=1,ps="convectCG.eps",viso=viso);
plot(c,wait=1,fill=1,ps="convectDG.eps",viso=viso);
```

需要注意到新的关键词：`intalledges` ，指在所有三角形的所有边界上做积分。

$$\texttt{intalledges(Th)} \equiv \sum_{T\in\texttt{Th}} \int_{\partial T} \tag{3.5}$$

（因此所有内部边界积了两次）; 若三角形有一边为边界, 则nTonEdge 的值为1，否则为0; `jump` 代表$[c]$。 两种方法的结果都显示在图 3.6中，这里两个图都使用相同等值线表示，； 可以利用plot-modifier viso实现。
还要注意到在macro 里并未出现参数u，但是在语法上是需要的；同时简单地将n 替换为`(N.x*v1+N.y*v2)`。从表达形式我们可以大胆猜测 `N.x,N.y` 就是边界的法线，事实如此。
现在如果你认为间断-Galerkin方法（DG）太慢可以试一下下面的程序

```
                                                       //    同样的DG，但速度更快
varf aadual(cc,w) = int2d(Th)((cc/dt+(v1*dx(cc)+v2*dy(cc)))*w)
        + intalledges(Th)((1-nTonEdge)*w*(al*abs(n)-n/2)*jump(cc));
```

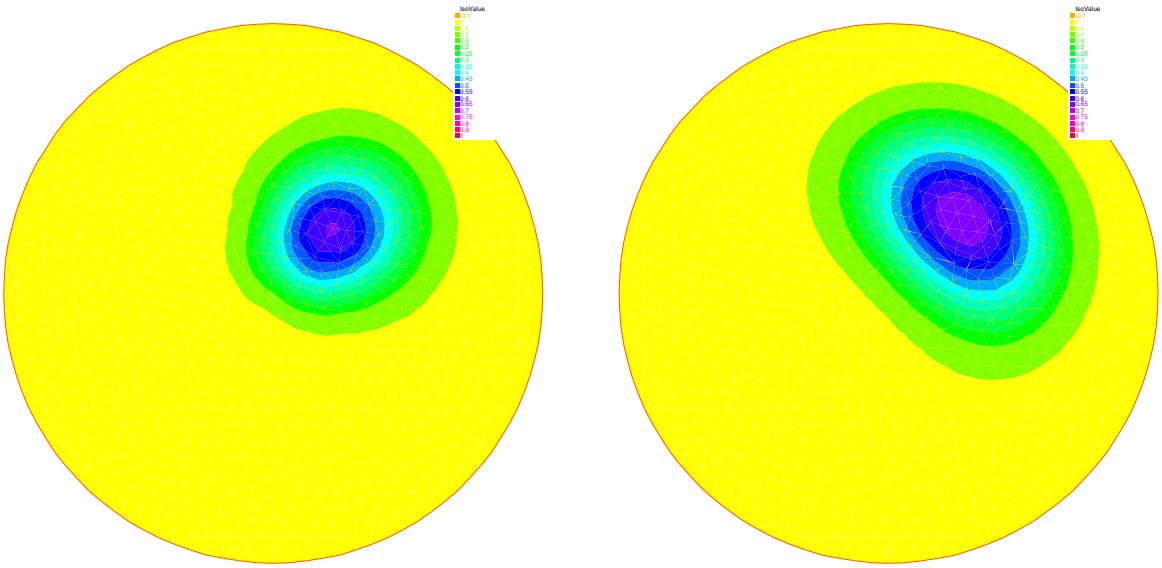


图 3.6: 旋转山在一次旋转之后的情况，左边利用特征-Galerkin方法， 右边利用间断的 P_1 Galerkin FEM方法。

```
varf bbdual(ccold,w) =  - int2d(Th)(ccold*w/dt);
matrix  AA= aadual(Vh,Vh);
matrix BB = bbdual(Vh,Vh);
set (AA,init=t,solver=sparsesolver);
Vh rhs=0;
for ( t=0; t< 2*pi ; t+=dt)
{
  ccold=cc;
  rhs[] = BB* ccold[];
  cc[] = AA^-1*rhs[];
  plot(cc,fill=0,cmm="t="+t + ", min=" + cc[].min + ", max=" +  cc[].max);
};
```

注意到新的关键词set，这是用来申明问题的求解器；调节器init 的作用是告诉求解器里的矩阵是否改变(init=true)，参数的名字也是与问题定义里的相同（见 6.9） 。

有限体积法 同样也可以通过FreeFem++ 实现，但需要编写程序。 例如，对于Dervieux等人的$P_0 - P_1$ 有限体积法，对于一个由所有包含q^i 为顶点的三角形组成的栅元σ_i ，可用一个P_0的函数c^1表示，在每一个顶点 q^i 周围的值为一个P_0的函数c^1，即常值$c^1(q^i)$。 因此迎风可以利用取中间的左边或右边的值表示出来:

$$\int_{\sigma_i} \frac{1}{\delta t}(c^{1^{n+1}} - c^{1^n}) + \int_{\partial\sigma_i} u \cdot n c^- = 0 \quad \forall i$$

转化成程序语言如下：

```
load "mat_dervieux";                                    //    需要额外加载C++模块
border a(t=0, 2*pi){ x = cos(t); y = sin(t);  }
mesh th = buildmesh(a(100));
fespace Vh(th,P1);
```

```
Vh vh,vold,u1 = y, u2 = -x;
Vh v = exp(-10*((x-0.3)^2 +(y-0.3)^2)), vWall=0, rhs =0;

real dt = 0.025;
                                                    //    qf1pTlump 代表使用了质量权重
problem  FVM(v,vh) = int2d(th,qft=qf1pTlump)(v*vh/dt)
                   - int2d(th,qft=qf1pTlump)(vold*vh/dt)
      + int1d(th,a)(((u1*N.x+u2*N.y)<0)*(u1*N.x+u2*N.y)*vWall*vh)
+ rhs[] ;

matrix A;
MatUpWind0(A,th,vold,[u1,u2]);

for ( int t=0; t< 2*pi ; t+=dt){
  vold=v;
  rhs[] = A * vold[] ; FVM;
  plot(v,wait=0);
};
```

质量权重参数的选择使得求积公式中的高斯点位于最高点，这样可以使质量矩阵对角化；线性系统的部分可以利用共轭梯度法求解，仅通过一两次迭代就可以收敛。
右边的rhs是利用外部C++函数MatUpWind0(...)计算得到，这个程序是

```
                                                //    在Dervieux FVM中的一个三角形上计算矩阵
int   fvmP1P0(double q[3][2],                                    //    三角形T上的三个顶点
              double u[2],                                       //    在T上的对流速度
              double c[3],                                       //    在T上的P1函数
              double a[3][3],                                    //    输出矩阵
              double where[3] )                          //    where>0 说明在边界上
{
  for(int i=0;i<3;i++) for(int j=0;j<3;j++) a[i][j]=0;

    for(int i=0;i<3;i++){
        int ip = (i+1)%3, ipp =(ip+1)%3;
        double unL =-((q[ip][1]+q[i][1]-2*q[ipp][1])*u[0]
                    -(q[ip][0]+q[i][0]-2*q[ipp][0])*u[1])/6;
        if(unL>0) { a[i][i] += unL; a[ip][i]-=unL;}
            else{ a[i][ip] += unL; a[ip][ip]-=unL;}
        if(where[i]&&where[ip]){                                 //    这是一个有界边界
            unL=((q[ip][1]-q[i][1])*u[0] -(q[ip][0]-q[i][0])*u[1])/2;
            if(unL>0) { a[i][i]+=unL; a[ip][ip]+=unL;}
        }
    }
  return 1;
}
```

这段程序已经插入到一个更大的 .cpp 文件中，作为 FreeFem++ 链接的加载模块，详见附录A。

3.7 弹性系统

弹性 固体在受力时会发生形变。受力后，固体上一初始点 (x, y, z)将会变到 (X, Y, Z)，我们把向量 $\mathbf{u} = (u_1, u_2, u_3) = (X - x, Y - y, Z - z)$ 称为 位移。当位移很小并且固体具有弹性时, 胡克定律给出了应力张量 $\sigma(u) = (\sigma_{ij}(u))$ 和应变张量 $\epsilon(u) = \epsilon_{ij}(u)$ 的关系

$$\sigma_{ij}(u) = \lambda\delta_{ij}\nabla.\mathbf{u} + 2\mu\epsilon_{ij}(u),$$

这里当 $i = j$，克罗内克符号 $\delta_{ij} = 1$ 否则为0，所以有

$$\epsilon_{ij}(u) = \frac{1}{2}(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}),$$

这里 λ, μ 是两个描述固体力学性质的常量，而且这两个常量与更有名的常量E （杨氏模量）和 ν（泊松比）有关：

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}.$$

Lamé's 系统 考虑一个沿 Oz轴的横梁，其垂直截面为 Ω。在受到垂直于轴的力$\mathbf{f}$ 时，横梁在垂直截面Ω上形变量 $\mathbf{u}(x)$ 在x 和y 上的分量满足以下等式：

$$-\mu\Delta\mathbf{u} - (\mu + \lambda)\nabla(\nabla.\mathbf{u}) = \mathbf{f} \ \text{ in } \ \Omega,$$

这里λ, μ 是上面介绍过的Lamé参数。
注： 我们并不使用这个方程是因为与之相联系的变分形式下不能给出合适的边界条件，因此仅采用以下简单形式

$$-div(\sigma) = \mathbf{f} \quad \text{in}\Omega$$

其变分形式是：

$$\int_\Omega \sigma(u) : \epsilon(\mathbf{v})\ dx - \int_\Omega \mathbf{v}f\ dx = 0;$$

这里符号:表示张量的标量积，例如$a : b = \sum_{i,j} a_{ij}b_{ij}$.
所以这个变分形式可以写成 :

$$\int_\Omega \lambda\nabla.u\nabla.v + 2\mu\epsilon(\mathbf{u}) : \epsilon(\mathbf{v})\ dx - \int_\Omega \mathbf{v}f\ dx = 0;$$

例 考虑无形变长方形弹性板，尺寸为 $[0, 20] \times [-1, 1]$。 物体中心部分受的力是重力 $\boldsymbol{f}$，在下、上和右边界的受力$\boldsymbol{g}$均为0。 板的左边（垂直方向）固定，则此边界条件可以写成：

$$\begin{aligned} \sigma.\mathbf{n} &= g = 0 \ \text{ on } \ \Gamma_1, \Gamma_4, \Gamma_3, \\ \mathbf{u} &= \mathbf{0} \ \text{ on } \ \Gamma_2 \end{aligned}$$

这里 $\mathbf{u} = (u, v)$ 有两个分量。

上面的两个等式通过混合导数强烈耦合在一起，因此基于任一个部分进行迭代均不可行。我们尝试利用 `FreeFem++` 去实现，相应程序可以写成：

Example 3.8 (lame.edp)

```
mesh Th=square(10,10,[20*x,2*y-1]);                                    //    文件 lame.edp
fespace Vh(Th,P2);
Vh u,v,uu,vv;
real sqrt2=sqrt(2.);
macro epsilon(u1,u2)  [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2]          //    EOM
  //    这里用sqrt2是因为我们们想完成:  epsilon(u1,u2)'* epsilon(v1,v2) == ε(u):ε(v)
macro div(u,v)  ( dx(u)+dy(v) )                                        //    EOM


real E = 21e5, nu = 0.28, mu= E/(2*(1+nu));
real lambda = E*nu/((1+nu)*(1-2*nu)), f = -1;                          //

solve lame([u,v],[uu,vv])= int2d(Th)(
        lambda*div(u,v)*div(uu,vv)
        +2.*mu*( epsilon(u,v)'*epsilon(uu,vv) ) )
        - int2d(Th)(f*vv)
        + on(4,u=0,v=0);
real coef=100;
plot([u,v],wait=1,ps="lamevect.eps",coef=coef);

mesh th1 = movemesh(Th, [x+u*coef, y+v*coef]);
plot(th1,wait=1,ps="lamedeform.eps");
real dxmin  = u[].min;
real dymin  = v[].min;

cout << " - dep.  max   x = "<< dxmin<< " y=" << dymin << endl;
cout << "   dep.  (20,0)  = " << u(20,0) << " " << v(20,0) << endl;
```

该问题的数值解显示在图3.7中，输出的结果是:

```
 -- square mesh : nb vertices  =121 ,  nb triangles = 200 ,  nb boundary edges 40
 -- Solve :           min -0.00174137  max 0.00174105
          min -0.0263154  max 1.47016e-29
 - dep.  max   x = -0.00174137 y=-0.0263154
   dep.  (20,0)  = -1.8096e-07 -0.0263154
times: compile 0.010219s, execution 1.5827s
```

3.8 流体的斯托克斯系统

考虑一雷诺数较低（例如微生物）的二维流体，即相对于第三坐标不变，其描述方程如下：

$$-\Delta u + \nabla p = 0$$
$$\nabla \cdot \boldsymbol{u} = 0$$

其中 $\boldsymbol{u} = (u_1, u_2)$ 为流体速度， p 为流体压力。 这是一个标准的试验腔体，充满液体并加盖，腔体整体在水平方向上匀速移动。采用LBB条件，对压力和速度用相容的有限元空间进行离散化，得到：

$$\sup_{p \in P_h} \frac{(\boldsymbol{u}, \nabla p)}{|p|} \geq \beta |\boldsymbol{u}| \quad \forall \boldsymbol{u} \in U_h$$

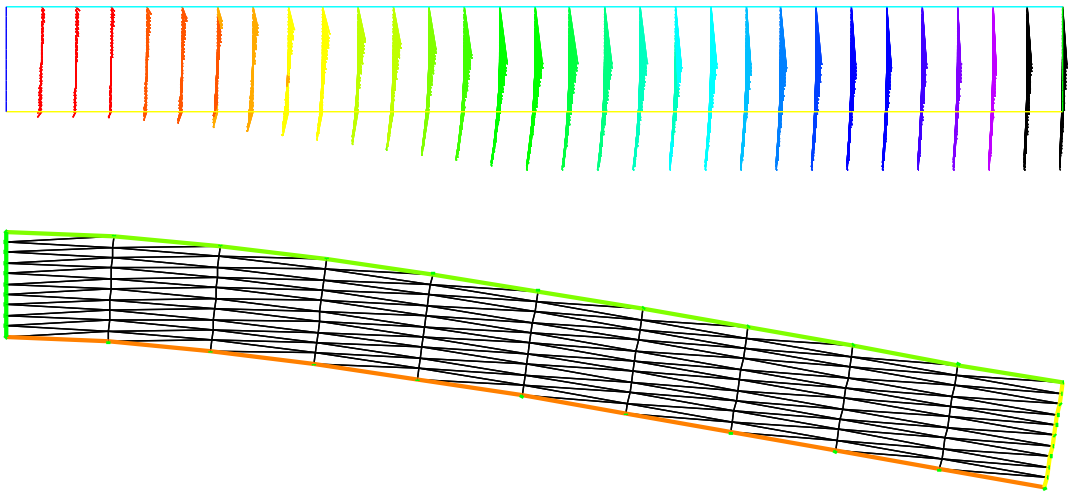


图 3.7: 一个左端固定在垂直墙面的二维横梁受自身重力弯曲，对应Lamé's方程的解（显示的结果放大了100倍）。 注：箭头大小因为系统原因自动截断，然而其颜色反映了真实长度

```
//   文件 stokes.edp
int n=3;
mesh Th=square(10*n,10*n);
fespace Uh(Th,P1b); Uh u,v,uu,vv;
fespace Ph(Th,P1);  Ph p,pp;

solve stokes([u,v,p],[uu,vv,pp]) =
    int2d(Th)(dx(u)*dx(uu)+dy(u)*dy(uu) + dx(v)*dx(vv)+ dy(v)*dy(vv)
              + dx(p)*uu + dy(p)*vv + pp*(dx(u)+dy(v))
              - 1e-10*p*pp)
            + on(1,2,4,u=0,v=0) + on(3,u=1,v=0);
plot([u,v],p,wait=1);
```

注：为了固定压力的常量部分，加入了一个稳定项 **-10e-10*p*pp**。
计算结果显示在图3.9中。

3.9 Navier-Stokes方程的一个投影算法

摘要 计算流体流动需要好的算法和好的三角形剖分。我们在这里举采用一个复杂算法的例子（也是第一个采用自适应网格的例子）。

不可压缩的粘性流体满足以下方程式:

$$\partial_t u + u \cdot \nabla u + \nabla p - \nu \Delta u = 0, \quad \nabla \cdot u = 0 \quad \text{in } \Omega \times [0, T],$$

$$u|_{t=0} = u^0, \quad u|_\Gamma = u_\Gamma.$$

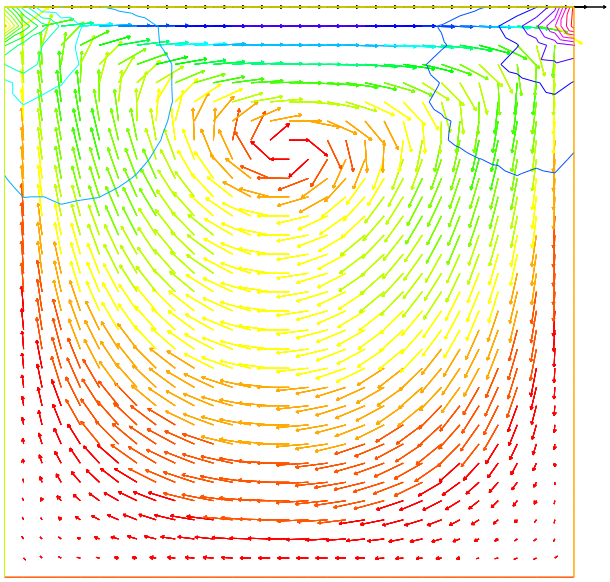


图 3.8: 腔体问题的Stokes方程的解，图中显示了速度场和压力线

针对这个问题，Chorin给出了一个参考算法：

$$\frac{1}{\delta t}[u^{m+1} - u^m oX^m] + \nabla p^m - \nu\Delta u^m = 0, \quad u|_\Gamma = u_\Gamma, \nu\partial_n u|_{\Gamma_{out}} = 0$$

$$-\Delta p^{m+1} = -\nabla \cdot u^m oX^m, \quad \partial_n p^{m+1} = 0, p^{m+1} = 0 \quad \text{on} \quad \Gamma_{out}$$

其中 $uoX(x) = u(x - u(x)\delta t)$，由前面章节中提到的特征方法（method of characteristics）近似 $\partial_t u + u \cdot \nabla u$ 而得。

针对Chorin算法，我们在出口采用自由边界条件（也即是$p = 0, \nu\partial_n u = 0$）计算修正后的压力场。

$$-\Delta q = \nabla \cdot \boldsymbol{u}, q = 0 on \Gamma_{out}$$

并且定义

$$u^{m+1} = \tilde{u} + \nabla q \delta t, \quad p^{m+1} = p^m - q - \overline{p^m - q}$$

其中 $\tilde{u}$ 是Chorin算法中的 (u^{m+1}, v^{m+1}) , P 则是质量权重的L^2投影（是稀疏矩阵）。

后台阶 其几何图形是一个具有后台阶的通道，使得内流截面小于外流截面。（在进行三角剖分时）必须要保证能够正确地描述该几何形状下的导致的回流区。
这仅仅在三角剖分足够细致或能很好的适配流体行为时才能够实现。
注 (FH)：这个例子有一个技术难点，即对于出口边界，置$p = 0, \nu\partial_n u = 0$。 0。 在之前的版本，这并不需要考虑，并且这个修正并不简单的。

Example 3.9 (NSprojection.edp)

```
//      文件 NSprojection.edp
//      2014年6月版
//      FH.修改了u的出口边界条件,以简化问题
//      ............
verbosity=0;
```

```
border a0(t=1,0){ x=0;       y=t;       label=1;}
border a1(t=0,1){ x=2*t;     y=0;         label=2;}
border a2(t=0,1){ x=2;       y=-t/2;        label=2;}
border a3(t=0,1){ x=2+18*t^1.2;  y=-0.5;        label=2;}
border a4(t=0,1){ x=20;      y=-0.5+1.5*t;    label=3;}
border a5(t=1,0){ x=20*t; y=1;         label=4;}
int n=1;
mesh Th= buildmesh(a0(3*n)+a1(20*n)+a2(10*n)+a3(150*n)+a4(5*n)+a5(100*n));
plot(Th);
fespace Vh(Th,P1);
real nu = 0.0025, dt = 0.2;                                    //    Reynolds=200
Vh uold = u,  vold = v, pold=p;
func uBCin =  4*y*(1-y)*(y>0)*(x<2) ;
func uBCout =  4./1.5*(y+0.5)*(1-y) *(x>19);
Vh w,u = 0, v =0, p = 0, q=0;
real epsv = 1e-6, epsu = 1e-6, epsp = 1e-6;                        //    Eps CG ..
                                                          //    定义矩阵 dtMx 和 dtMy
macro BuildMat()
{ /* for memory managenemt */
varf vM(unused,v) = int2d(Th)(v) ;
varf vdx(u,v) = int2d(Th)(v*dx(u)*dt) ;
varf vdy(u,v) = int2d(Th)(v*dy(u)*dt) ;
real[int] Mlump = vM(0,Vh);
real[int] one(Vh.ndof); one = 1;
real[int] M1 = one ./ Mlump;
matrix dM1 = M1;
matrix Mdx = vdx(Vh,Vh);
matrix Mdy = vdy(Vh,Vh);
dtM1x = dM1*Mdx;
dtM1y = dM1*Mdy;
}                                                                  //    EOF

BuildMat

real err=1, outflux=1;

for(int n=0;n<300;n++){

  Vh uold = u,  vold = v, pold=p;

  solve pb4u(u,w,init=n,solver=LU)
        =int2d(Th)(u*w/dt +nu*(dx(u)*dx(w)+dy(u)*dy(w)))
        -int2d(Th)((convect([uold,vold],-dt,uold)/dt-dx(p))*w)
        + on(1,u = 4*y*(1-y)) + on(2,4,u = 0) + on(3,u=f);   //    Neuman边界条件
  plot(u);

  solve pb4v(v,w,init=n,solver=CG,eps=epsv)
        = int2d(Th)(v*w/dt +nu*(dx(v)*dx(w)+dy(v)*dy(w)))
        -int2d(Th)((convect([uold,vold],-dt,vold)/dt-dy(p))*w)
        +on(1,2,3,4,v = 0);

 solve pb4p(q,w,init=n,solver=CG,eps=epsp)= int2d(Th)(dx(q)*dx(w)+dy(q)*dy(w))
    - int2d(Th)((dx(u)+ dy(v)-meandiv)*w/dt)+ on(3,q=0);

                                                          //    算法中采用绝对epsilon
    epsv = -abs(epsv);
```

```
    epsu = -abs(epsu);
    epsp = -abs(epsp);

    p = pold-q-q;
    u[] += dtM1x*q[];
    v[] += dtM1y*q[];


 if(n%50==49){
    Th = adaptmesh(Th,[u,v],q,err=0.04,nbvx=100000);
    plot(Th, wait=true);
    BuildMat
 }                                                          //    重置矩阵


err = sqrt(int2d(Th)(square(u-uold)+square(v-vold))/Th.area) ;
cout << " iter " << n << " Err L2 = " << err
<< " flux sortant = "<< outflux << endl;
if(err < 1e-3) break;
}

assert(abs(outflux)< 2e-3);                                 //    验证
plot(p,wait=1,ps="NSprojP.eps");
plot(u,wait=1,ps="NSprojU.eps");
```

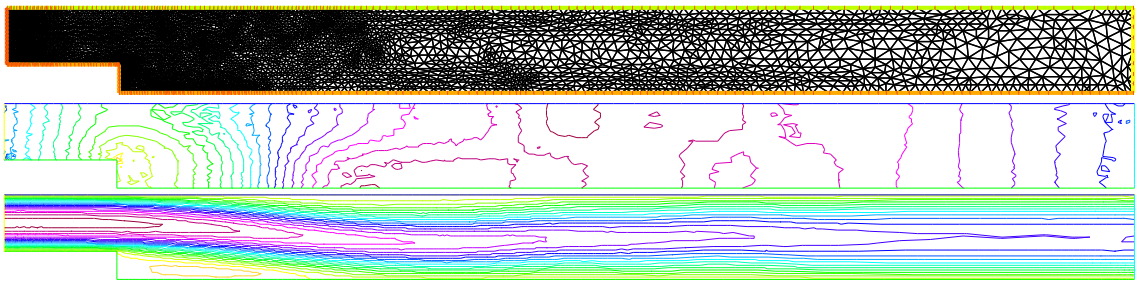


图 3.9: Rannacher投影算法: 适应网格（上图）下的解：雷诺数为400时的压力分布（中图）和水平速率u（下图）

我们在图 3.9中所展示数值解是在雷诺数为400时，经初始网格迭代50次后再采用适应网格而得出的。

3.10 稳定Navier-Stokes方程的牛顿法

这是一个在定义域$\Omega \subset \mathbb{R}^d (d = 2, 3)$内求解流体速度场 $\boldsymbol{u} = (u_i)_{i=1}^d$和压力$p$分布的问题：

$$\begin{aligned} (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nu\Delta\boldsymbol{u} + \nabla p &= 0, \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned}$$

其中 ν 是流体的粘度, $\nabla = (\partial_i)_{i=1}^d$, “$\cdot$”表示内积, 并且 $\Delta = \nabla \cdot \nabla$ ，再加上边界条件（即$\boldsymbol{u}$ 在边界 Γ上为给定值）

其弱形式是求解$\boldsymbol{u}, p$ 使得 $\forall \boldsymbol{v}$ （在边界 Γ上为零值）和 $\forall q$有：

$$\int_\Omega ((\boldsymbol{u}\cdot\nabla)\boldsymbol{u}).\boldsymbol{v} + \nu\nabla\boldsymbol{u} : \nabla\boldsymbol{v} - p\nabla\cdot\boldsymbol{v} - q\nabla\cdot\boldsymbol{u} = 0 \tag{3.6}$$

求解非线性问题的牛顿算法是找到 $u \in V$ 使得 $F(u) = 0$,其中 $F : V \mapsto V$.

1. 选择 $u_0 \in \mathbb{R}^n$;
2. for ($i = 0$; i ¡ niter; $i = i + 1$)
   (a) solve $DF(u_i)w_i = F(u_i)$;
   (b) $u_{i+1} = u_i - w_i$;

   break $||w_i|| < \varepsilon$.

其中 $DF(u)$是 F 在点 u的微分, 这是一个线性算子，满足: $F(u+\delta) = F(u)+DF(u)\delta+o(\delta)$
对于Navier Stokes方程, F 和 DF分别为 :

$$\begin{aligned} F(\boldsymbol{u},p) &= \int_\Omega && ((\boldsymbol{u}\cdot\nabla)\boldsymbol{u}).\boldsymbol{v} + \nu\nabla\boldsymbol{u} : \nabla\boldsymbol{v} - p\nabla\cdot\boldsymbol{v} - q\nabla\cdot\boldsymbol{u} \\ DF(\boldsymbol{u},p)(\boldsymbol{\delta u},\delta p) &= \int_\Omega && ((\boldsymbol{\delta u}\cdot\nabla)\boldsymbol{u}).\boldsymbol{v} + ((\boldsymbol{u}\cdot\nabla)\boldsymbol{\delta u}).\boldsymbol{v} \\ &+ && \nu\nabla\boldsymbol{\delta u} : \nabla\boldsymbol{v} - \delta p\nabla\cdot\boldsymbol{v} - q\nabla\cdot\boldsymbol{\delta u} \end{aligned}$$

因而牛顿算法相应的代码如下：

Example 3.10 (NSNewton.edp) ...

```
for( n=0;n< 15;n++)
{ solve Oseen([du1,du2,dp],[v1,v2,q]) =
      int2d(Th) (  nu*(Grad(du1,du2)'*Grad(v1,v2) )
                 + UgradV(du1,du2, u1, u2)'*[v1,v2]
                 + UgradV( u1, u2,du1,du2)'*[v1,v2]
                 - div(du1,du2)*q - div(v1,v2)*dp
                 - 1e-8*dp*q                                  //    稳定化项
                )
    - int2d(Th) (  nu*(Grad(u1,u2)'*Grad(v1,v2) )
                 + UgradV(u1,u2, u1, u2)'*[v1,v2]
                 - div(u1,u2)*q - div(v1,v2)*p
                )
    + on(1,du1=0,du2=0) ;
  u1[] -= du1[];  u2[] -= du2[]; p[]  -= dp[];
  err= du1[].linfty + du2[].linfty + dp[].linfty;
  if(err < eps) break;
  if( n>3 && err > 10.) break;                                //    爆破 ????
}
```

其中算子是:

```
macro Grad(u1,u2) [ dx(u1),dy(u1) , dx(u2),dy(u2) ]                    //
macro UgradV(u1,u2,v1,v2) [ [u1,u2]'*[dx(v1),dy(v1)] ,
```

```
                                [u1,u2]'*[dx(v2),dy(v2)] ]                    //
macro div(u1,u2)  (dx(u1)+dy(u2))                                             //
```

我们先来建立2维柱体的外围计算网格。

```
real R = 5,L=15;
border cc(t=0,2*pi){ x=cos(t)/2;y=sin(t)/2;label=1;}
border ce(t=pi/2,3*pi/2) { x=cos(t)*R;y=sin(t)*R;label=1;}
border beb(tt=0,1) { real t=tt^1.2; x= t*L; y= -R; label = 1;}
border beu(tt=1,0) { real t=tt^1.2; x= t*L; y= R; label = 1;}
border beo(t=-R,R) {  x= L; y= t; label = 0;}
border bei(t=-R/4,R/4) {  x= L/2; y= t; label = 0;}
mesh Th=buildmesh(cc(-50)+ce(30)+beb(20)+beu(20)+beo(10)+bei(10));
plot(Th);

                                                  //     为了画图构造有界区域
func bb=[[-1,-2],[4,2]];

/  FE Space Taylor Hood
fespace Xh(Th,P2);                                                  //     速度
fespace Mh(Th,P1);                                                  //     压强
Xh u1,u2,v1,v2,du1,du2,u1p,u2p;
Mh p,q,dp,pp;

                                                //     通基于边界条件先给出假设解
u1 = ( x^2+y^2) > 2;
u2=0;
```

最后，我们采用一定的技巧使得粘度参数 ν保持连续性, 因为牛顿法是在最后一个粘度 ν值开始发散。

```
                                                                //     物理参数
real nu= 1./50, nufinal=1/200. ,cnu=0.5;

                                                              //     牛顿法终止参数
real eps=1e-6;

verbosity=0;
while(1)                                                          //     粘性项循环
{   int n;
    real err=0;                                                 //     牛顿法误差 ...

      ... 将新的牛顿法放在这里

    if(err < eps)
     {                                                  //     ν减小，问题收敛（比较困难）
       plot([u1,u2],p,wait=1,cmm=" rey = " + 1./nu , coef=0.3,bb=bb);
       if( nu == nufinal) break;
       if( n < 4) cnu=cnu^1.5;                                //     快速收敛 => 变化更快
       nu = max(nufinal, nu* cnu);                                  //     新的粘度
       u1p=u1;  u2p=u2;  pp=p;                                    //     保存正确解 ...
     }
     else
     {                                                      //     ν增加，发散 （更简单）
       assert(cnu< 0.95);                                           //     该方法最终发散
       nu = nu/cnu;                                             //     得到前一步的粘性项
```

```
        cnu= cnu^(1./1.5);                                          //     不收敛 => 变化更慢
        nu = nu* cnu;                                                //     新的粘性项
        cout << " restart nu = " << nu << " Rey= "<< 1./nu << "  (cnu = " << cnu
<< " ) \n";
                                                                     //     保存正确解 ..
        u1=u1p;
        u2=u2p;
        p=pp;
      }
}
```

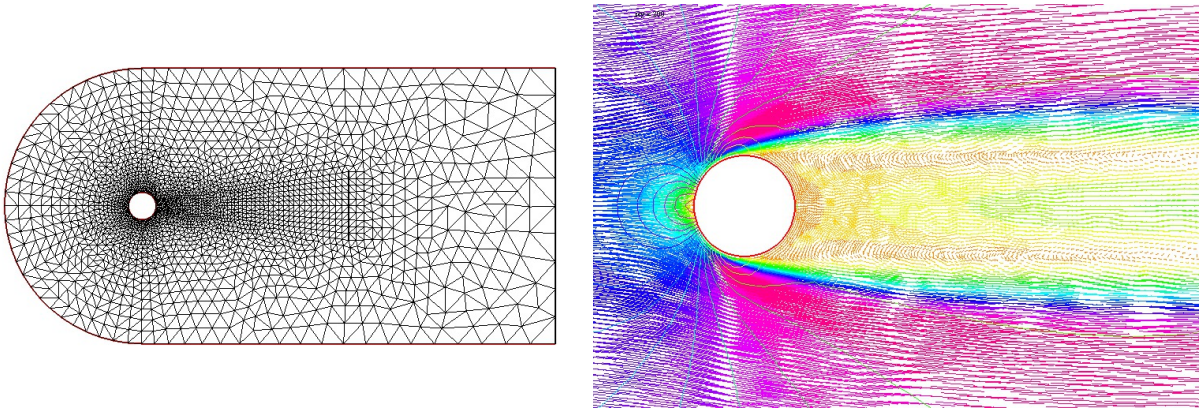


图 3.10: 雷诺数为200时，对应网格的选取和得到的速率及压力分布

3.11 大型流体问题

我们的一个朋友在Auroville-india建立了一个斜坡来放置空调房。我在拜访他的时候，他告诉我他希望冷气可以沿着斜道通过门进入房间使来访者的脚感到凉爽。我告诉他“这是不可能的”并且决定在数值上来验证，结论就在这本书的首页。
Navier-Stokes方程在不同温度的密度情况下的解为流体的速度以及压力。
问题涉及到的几何是一个梯形，我们假定，初始条件是由底部的冷空气和上部的暖空气来组成的内流。有一自由的外流，在（人为的）顶部边界具有滑移速率而在底部没有滑移速率。Navier-Stokes 方程的温度项采用RANS $k-\epsilon$模型，浮力用Boussinesq 近似，如此一来，我们有：

$$
\begin{aligned}
&\partial_t\theta + u\nabla\theta - \nabla\cdot(\kappa_T^m\nabla\theta) = 0\\
&\partial_t u + u\nabla u - \nabla\cdot(\mu_T\nabla u) + \nabla p + e(\theta-\theta_0)\boldsymbol{e}_2, \quad \nabla\cdot u = 0\\
&\mu_T = c_\mu\frac{k^2}{\epsilon}, \quad \kappa_T = \kappa\mu_T\\
&\partial_t k + u\nabla k + \epsilon - \nabla\cdot(\mu_T\nabla k) = \frac{\mu_T}{2}|\nabla u + \nabla u^T|^2\\
&\partial_t\epsilon + u\nabla\epsilon + c_2\frac{\epsilon^2}{k} - \frac{c_\epsilon}{c_\mu}\nabla\cdot(\mu_T\nabla\epsilon) = \frac{c_1}{2}k|\nabla u + \nabla u^T|^2 = 0
\end{aligned}
\tag{3.7}
$$

选择保持正定性的时间离散化，运用特征方法 $(X^m(x)\approx x - u^m(x)\delta t)$

$$
\frac{1}{\delta t}(\theta^{m+1} - \theta^m\circ X^m) - \nabla\cdot(\kappa_T^m\nabla\theta^{m+1}) = 0
$$

$$
\begin{aligned}
&\frac{1}{\delta t}(u^{m+1}-u^m\circ X^m)-\nabla\cdot(\mu_T^m\nabla u^{m+1})+\nabla p^{m+1}+e(\theta^{m+1}-\theta_0)\boldsymbol{e}_2,\quad \nabla\cdot u^{m+1}=0\\
&\frac{1}{\delta t}(k^{m+1}-k^m\circ X^m)+k^{m+1}\frac{\epsilon^m}{k^m}-\nabla\cdot(\mu_T^m\nabla k^{m+1})=\frac{\mu_T^m}{2}|\nabla u^m+\nabla u^{mT}|^2\\
&\frac{1}{\delta t}(\epsilon^{m+1}-\epsilon^m\circ X^m)+c_2\epsilon^{m+1}\frac{\epsilon^m}{k^m}-\frac{c_\epsilon}{c_\mu}\nabla\cdot(\mu_T^m\nabla\epsilon^{m+1})=\frac{c_1}{2}k^m|\nabla u^m+\nabla u^{mT}|^2\\
&\mu_T^{m+1}=c_\mu\frac{{k^{m+1}}^2}{\epsilon^{m+1}},\quad \kappa_T^{m+1}=\kappa\mu_T^{m+1}
\end{aligned}
\tag{3.8}
$$

在具有适当边界条件的变分形式如下:

```
real L=6;
border aa(t=0,1){x=t; y=0 ;}
border bb(t=0,14){x=1+t; y= - 0.1*t ;}
border cc(t=-1.4,L){x=15; y=t ;}
border dd(t=15,0){x= t ; y = L;}
border ee(t=L,0.5){ x=0; y=t ;}
border ff(t=0.5,0){ x=0; y=t ;}
int n=8;
mesh Th=buildmesh(aa(n)+bb(9*n) + cc(4*n) + dd(10*n)+ee(6*n) + ff(n));
real s0=clock();

fespace Vh2(Th,P1b);                                                    //    速度空间
fespace Vh(Th,P1);                                                      //    压强空间
fespace V0h(Th,P0);                                                     //    梯度空间
Vh2 u2,v2,up1=0,up2=0;
Vh2 u1,v1;
Vh  u1x=0,u1y,u2x,u2y, vv;

real reylnods=500;
                  //   cout << " Enter the reynolds number :"; cin >> reylnods;
assert(reylnods>1 && reylnods < 100000);
up1=0;
up2=0;
func g=(x)*(1-x)*4;                                                     //    内流
Vh p=0,q, temp1,temp=35, k=0.001,k1,ep=0.0001,ep1;
V0h muT=1,prodk,prode, kappa=0.25e-4, stress;
real alpha=0, eee=9.81/303, c1m = 1.3/0.09 ;
real  nu=1, numu=nu/sqrt( 0.09), nuep=pow(nu,1.5)/4.1;
int i=0,iter=0;
real dt=0;
problem TEMPER(temp,q) =                                                //    温度方程
    int2d(Th)(
             alpha*temp*q + kappa * ( dx(temp)*dx(q) + dy(temp)*dy(q) ))
                                                 //    + int1d(Th,aa,bb)(temp*q* 0.1)
  + int2d(Th) ( -alpha*convect([up1,up2],-dt,temp1)*q )
   + on(ff,temp=25)
  + on(aa,bb,temp=35) ;

problem kine(k,q)=                                                      //    得到涡流动能
    int2d(Th)(
             (ep1/k1+alpha)*k*q + muT * ( dx(k)*dx(q) + dy(k)*dy(q) ))
                                                  //    + int1d(Th,aa,bb)(temp*q*0.1)
  + int2d(Th) ( prodk*q-alpha*convect([up1,up2],-dt,k1)*q )
   + on(ff,k=0.0001)  + on(aa,bb,k=numu*stress) ;
```

```
 problem viscturb(ep,q)=                                          //     得到涡流粘滞能量比率
    int2d(Th)(
              (1.92*ep1/k1+alpha)*ep*q + c1m*muT * ( dx(ep)*dx(q) + dy(ep)*dy(q)
))
                                                    //    + int1d(Th,aa,bb)(temp*q*0.1)
  + int2d(Th) ( prode*q-alpha*convect([up1,up2],-dt,ep1)*q )
   + on(ff,ep= 0.0001) + on(aa,bb,ep=nuep*pow(stress,1.5)) ;


 solve NS ([u1,u2,p],[v1,v2,q]) = //    Navier-Stokes k-epsilon and Boussinesq
    int2d(Th)(
              alpha*( u1*v1 + u2*v2)
             + muT * (dx(u1)*dx(v1)+dy(u1)*dy(v1)+dx(u2)*dx(v2)+dy(u2)*dy(v2))
    //     ( 2*dx(u1)*dx(v1) + 2*dy(u2)*dy(v2)+(dy(u1)+dx(u2))*(dy(v1)+dx(v2)))
             + p*q*(0.000001)
             - p*dx(v1) - p*dy(v2)
             - dx(u1)*q - dy(u2)*q
            )
  + int1d(Th,aa,bb,dd)(u1*v1* 0.1)
  + int2d(Th) (eee*(temp-35)*v1 -alpha*convect([up1,up2],-dt,up1)*v1
                               -alpha*convect([up1,up2],-dt,up2)*v2 )
   + on(ff,u1=3,u2=0)
  + on(ee,u1=0,u2=0)
  + on(aa,dd,u2=0)
  + on(bb,u2= -up1*N.x/N.y)
  + on(cc,u2=0) ;
 plot(coef=0.2,cmm=" [u1,u2] et p  ",p,[u1,u2],ps="StokesP2P1.eps",value=1,wait=1);
{
  real[int] xx(21),yy(21),pp(21);
  for (int i=0;i<21;i++)
   {
     yy[i]=i/20.;
     xx[i]=u1(0.5,i/20.);
     pp[i]=p(i/20.,0.999);
    }
      cout << " " << yy << endl;
                                    //    plot([xx,yy],wait=1,cmm="u1 x=0.5 cup");
                           //    plot([yy,pp],wait=1,cmm="pressure y=0.999 cup");
}

dt = 0.05;
int nbiter = 3;
real coefdt = 0.25^(1./nbiter);
real coefcut = 0.25^(1./nbiter) , cut=0.01;
real tol=0.5,coeftol = 0.5^(1./nbiter);
nu=1./reylnods;

for (iter=1;iter<=nbiter;iter++)
{
 cout << " dt = " << dt << " ------------------------ " << endl;
  alpha=1/dt;
  for (i=0;i<=500;i++)
   {
     up1=u1;
     up2=u2;
```

```
    temp1=max(temp,25);
    temp1=min(temp1,35);
    k1=k; ep1=ep;
    muT=0.09*k*k/ep;
     NS; plot([u1,u2],wait=1);                          //    求解Navier-Stokes方程
    prode =0.126*k*(pow(2*dx(u1),2)+pow(2*dy(u2),2)+2*pow(dx(u2)+dy(u1),2))/2;
    prodk= prode*k/ep*0.09/0.126;
    kappa=muT/0.41;
    stress=abs(dy(u1));
    kine; plot(k,wait=1);
    viscturb; plot(ep,wait=1);
    TEMPER;                                                  //    求解温度方程
    if ( !(i % 5)){
        plot(temp,value=1,fill=true,ps="temp_"+iter+"_"+i+".ps");
        plot(coef=0.2,cmm=" [u1,u2] et p  ",p,[u1,u2],ps="plotNS_"+iter+"_"+i+".ps");
    }
    cout << "CPU " << clock()-s0 << "s " << endl;
  }

  if (iter>= nbiter) break;
   Th=adaptmesh(Th,[dx(u1),dy(u1),dx(u1),dy(u2)],splitpbedge=1,
                abserror=0,cutoff=cut,err=tol, inquire=0,ratio=1.5,hmin=1./1000);
 plot(Th,ps="ThNS.eps");
  dt = dt*coefdt;
  tol = tol *coeftol;
  cut = cut *coefcut;
}
cout << "CPU " <<clock()-s0 << "s " << endl;
```

3.12 涉及复数的例子

在微波炉中，电磁场激发分子产生热量。对于单色平面波，振幅由Helmholtz 方程给出:

$$\beta v + \Delta v = 0.$$

考虑一个长方形的微波炉，微波由上方部分墙发出。因而域的边界条件分成两部分，一部分为 Γ_1 ，对应 $v = 0$ ；另一部分 $\Gamma_2 = [c, d]$ ，对应（例如） $v = \sin(\pi\frac{y-c}{c-d})$。
有一物体在其中加热，记为B, 热源与 v^2成正比。 在均衡状态下，有

$$-\Delta\theta = v^2 I_B, \quad \theta_\Gamma = 0$$

其中，在 I_B物体上为 1，其他地方为 0。
结果显示在图3.11中。
在下列的程序中，空气中 $\beta = 1/(1 - I/2)$，物体中为$2/(1 - I/2)$ （$i = \sqrt{-1}$）:

Example 3.11 (muwave.edp)

```
                                                          //    文件 muwave.edp
real a=20, b=20, c=15, d=8, e=2, l=12, f=2, g=2;
border a0(t=0,1) {x=a*t; y=0;label=1;}
border a1(t=1,2) {x=a; y= b*(t-1);label=1;}
border a2(t=2,3) { x=a*(3-t);y=b;label=1;}
```

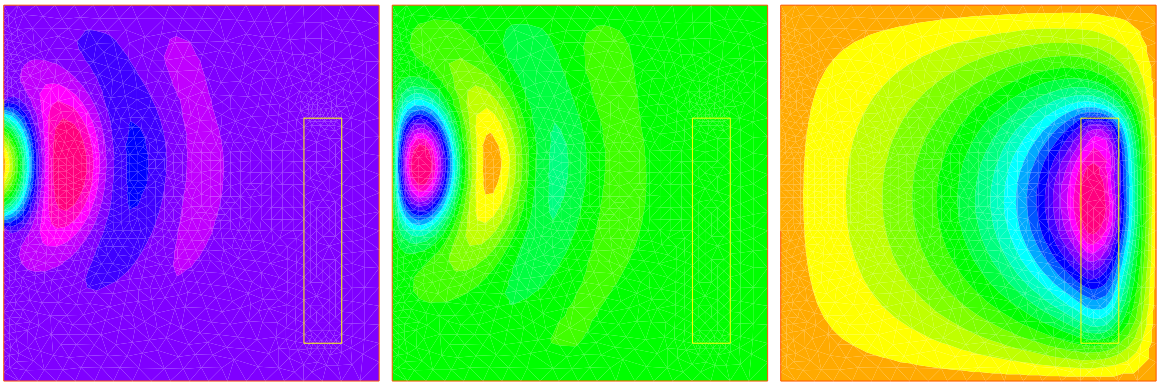


图 3.11: 微波炉：左图为微波的实部，中图为微波的虚部，右图为温度分布。

```
border a3(t=3,4){x=0;y=b-(b-c)*(t-3);label=1;}
border a4(t=4,5){x=0;y=c-(c-d)*(t-4);label=2;}
border a5(t=5,6){ x=0; y= d*(6-t);label=1;}

border b0(t=0,1) {x=a-f+e*(t-1);y=g; label=3;}
border b1(t=1,4) {x=a-f; y=g+l*(t-1)/3; label=3;}
border b2(t=4,5) {x=a-f-e*(t-4); y=l+g; label=3;}
border b3(t=5,8) {x=a-e-f; y= l+g-l*(t-5)/3; label=3;}
int n=2;
mesh Th = buildmesh(a0(10*n)+a1(10*n)+a2(10*n)+a3(10*n)
        +a4(10*n)+a5(10*n)+b0(5*n)+b1(10*n)+b2(5*n)+b3(10*n));
plot(Th,wait=1);
fespace Vh(Th,P1);
real meat =  Th(a-f-e/2,g+l/2).region, air= Th(0.01,0.01).region;
Vh R=(region-air)/(meat-air);

Vh<complex> v,w;
solve muwave(v,w) = int2d(Th)(v*w*(1+R)
                -(dx(v)*dx(w)+dy(v)*dy(w))*(1-0.5i))
   + on(1,v=0) + on(2, v=sin(pi*(y-c)/(c-d)));
Vh vr=real(v), vi=imag(v);
plot(vr,wait=1,ps="rmuonde.ps", fill=true);
plot(vi,wait=1,ps="imuonde.ps", fill=true);

fespace Uh(Th,P1); Uh u,uu, ff=1e5*(vr^2 + vi^2)*R;

solve temperature(u,uu)= int2d(Th)(dx(u)* dx(uu)+ dy(u)* dy(uu))
     - int2d(Th)(ff*uu) + on(1,2,u=0);
plot(u,wait=1,ps="tempmuonde.ps", fill=true);
```

3.13 优化控制

得益于BFGS，我们可以通过FreeFem++求解复杂的非线性优化问题。例如，考虑下如下逆问题：

$$\min_{b,c,d\in R} J = \int_E (u-u_d)^2 \ : \ -\nabla(\kappa(b,c,d)\cdot\nabla u) = 0, \quad u|_\Gamma = u_\Gamma$$

其中 u_d为期望态, 边界 u_Γ 和观测集合 $E \subset \Omega$ 均已给定。进一步假设

$$\kappa(x) = 1 + bI_B(x) + cI_C(x) + dI_D(x) \quad \forall x \in \Omega$$

其中B, C, D 为 Ω中的不同子集。

为了用拟牛顿BFGS方法求解这个问题，我们还需要知道 J 对 b, c, d的导数。 若$\delta b, \delta c, \delta d$分别是 b, c, d 的偏导数，我们有

$$\delta J \approx 2\int_E (u - u_d)\delta u, \quad -\nabla(\kappa \cdot \nabla \delta u) \approx \nabla(\delta\kappa \cdot \nabla u) \quad \delta u|_\Gamma = 0$$

显然 若 $\delta b = 1, \delta c = 0, \delta d = 0$则$J_b'$ 与 δJ 相等，同理可得关于 J_c' 和 J_d'的类似结论。

以上讨论可由下面程序实现：

```
                                                        //    文件 optimcontrol.edp
border aa(t=0, 2*pi) {    x = 5*cos(t);    y = 5*sin(t);  };
border bb(t=0, 2*pi) {    x = cos(t);    y = sin(t);  };
border cc(t=0, 2*pi) {    x = -3+cos(t);    y = sin(t);  };
border dd(t=0, 2*pi) {    x = cos(t);    y = -3+sin(t);  };
mesh th = buildmesh(aa(70)+bb(35)+cc(35)+dd(35));
fespace Vh(th,P1);
Vh Ib=((x^2+y^2)<1.0001),
   Ic=(((x+3)^2+ y^2)<1.0001),
   Id=((x^2+(y+3)^2)<1.0001),
   Ie=(((x-1)^2+ y^2)<=4),
   ud,u,uh,du;
real[int] z(3);
problem A(u,uh) =int2d(th)((1+z[0]*Ib+z[1]*Ic+z[2]*Id)*(dx(u)*dx(uh)
                    +dy(u)*dy(uh))) + on(aa,u=x^3-y^3);
z[0]=2; z[1]=3; z[2]=4;
A; ud=u;
ofstream f("J.txt");
func real J(real[int] & Z)
{
    for (int i=0;i<z.n;i++)z[i]=Z[i];
    A; real s= int2d(th)(Ie*(u-ud)^2);
    f<<s<<"   "; return s;
}

real[int] dz(3), dJdz(3);

problem B(du,uh)
  =int2d(th)((1+z[0]*Ib+z[1]*Ic+z[2]*Id)*(dx(du)*dx(uh)+dy(du)*dy(uh)))
  +int2d(th)((dz[0]*Ib+dz[1]*Ic+dz[2]*Id)*(dx(u)*dx(uh)+dy(u)*dy(uh)))
  +on(aa,du=0);

func real[int] DJ(real[int] &Z)
    {
      for(int i=0;i<z.n;i++)
```

```
        { for(int j=0;j<dz.n;j++) dz[j]=0;
          dz[i]=1; B;
          dJdz[i]= 2*int2d(th)(Ie*(u-ud)*du);
       }
     return dJdz;
}

real[int] Z(3);
for(int j=0;j<z.n;j++) Z[j]=1;
BFGS(J,DJ,Z,eps=1.e-6,nbiter=15,nbiterline=20);
cout << "BFGS: J(z) = " << J(Z) <<  endl;
for(int j=0;j<z.n;j++) cout<<z[j]<<endl;
plot(ud,value=1,ps="u.eps");
```

在此例中，集合 B, C, D, E 分别是由边界 bb, cc, dd, ee围成的圆形区域，定义域Ω 为aa 为成的圆形区域 。 期望态 u_d 为 当$b = 2, c = 3, d = 4$时上面偏微分方程的解。 所有未知量用向量z 表示。需要注意的是，在此之前需把 Z 复制到z 中，因为这两个变量一个是局部变量，另一个是全局变量。 取变量值$b = 2.00125, c = 3.00109, d = 4.00551$运行程序。结果在图3.12 中给出，可以看出 u 收敛，且实现了对J的连续函数估计。 需要注意我们采用

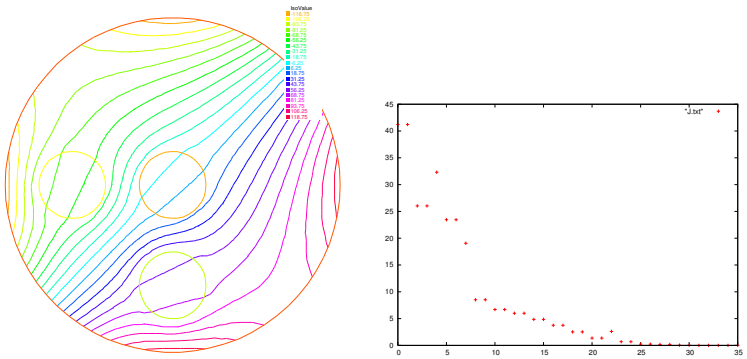


图 3.12: 左图是 u的等势线，右图为使用 BFGS 方法得到的 J 的连续估计 （为提高可读性剔除了500个数据中的5个数据）

了伴随态 ，定义 p如下，

$$-\nabla \cdot (\kappa \nabla p) = 2I_E(u - u_d), \quad p|_\Gamma = 0$$

因此

$$\begin{aligned} \delta J &= -\int_\Omega (\nabla \cdot (\kappa \nabla p))\delta u \\ &= \int_\Omega (\kappa \nabla p \cdot \nabla \delta u) = -\int_\Omega (\delta\kappa \nabla p \cdot \nabla u) \end{aligned} \tag{3.9}$$

令 $\delta b = 1, \delta c = \delta d = 0$ ，我们可得到

$$J'_b = -\int_B \nabla p \cdot \nabla u, \quad J'_c = -\int_C \nabla p \cdot \nabla u, \quad J'_d = -\int_D \nabla p \cdot \nabla u$$

注意： 因为BFGS方法需要存储一个$M \times M$的矩阵，其中 M 为未知量的个数。当未知的 x 为一个有限元方程时，我使用这种方法是不可靠的。 我们需要使用其他的最优化方法，如非线性共轭梯度法 NLCG （这也是 FreeFem++ 中的一个关键词）。 可在例题文件夹中找到algo.edp文件查看 。

3.14 带激波的流体

可压缩的欧拉方程可用限体积法或有限元方法离散化，但这些尚未集成到FreeFem++中。即便如此，采用特征方法，能够获得激波均值$\bar{f} = \frac{1}{2}(f^+ + f^-)$，在进行网格调节，可得到相似结果。

$$\begin{aligned}
&\partial_t \rho + \bar{u}\nabla\rho + \bar{\rho}\nabla\cdot u = 0\\
&\bar{\rho}(\partial_t u + \frac{\overline{\rho u}}{\bar{\rho}}\nabla u + \nabla p = 0\\
&\partial_t p + \bar{u}\nabla p + (\gamma-1)\bar{p}\nabla\cdot u = 0
\end{aligned} \tag{3.10}$$

一种可行的方法是先结合 u, p，之后更新 ρ, 即

$$\begin{aligned}
&\frac{1}{(\gamma-1)\delta t\bar{p}^m}(p^{m+1} - p^m \circ X^m) + \nabla\cdot u^{m+1} = 0\\
&\frac{\bar{\rho}^m}{\delta t}(u^{m+1} - u^m \circ \tilde{X}^m) + \nabla p^{m+1} = 0\\
&\rho^{m+1} = \rho^m \circ X^m + \frac{\bar{\rho}^m}{(\gamma-1)\bar{p}^m}(p^{m+1} - p^m \circ X^m)
\end{aligned} \tag{3.11}$$

图 3.13给出了数值解， 其 FreeFem++ 代码如下：

```
verbosity=1;
int anew=1;
real x0=0.5,y0=0, rr=0.2;
border ccc(t=0,2){x=2-t;y=1;};
border ddd(t=0,1){x=0;y=1-t;};
border aaa1(t=0,x0-rr){x=t;y=0;};
border cercle(t=pi,0){ x=x0+rr*cos(t);y=y0+rr*sin(t);}
border aaa2(t=x0+rr,2){x=t;y=0;};
border bbb(t=0,1){x=2;y=t;};

int m=5; mesh Th;
if(anew) Th = buildmesh (ccc(5*m) +ddd(3*m) + aaa1(2*m) + cercle(5*m)
              + aaa2(5*m) + bbb(2*m) );
      else Th = readmesh("Th_circle.mesh"); plot(Th,wait=0);

real dt=0.01, u0=2, err0=0.00625, pena=2;
fespace Wh(Th,P1);
fespace Vh(Th,P1);
Wh u,v,u1,v1,uh,vh;
Vh r,rh,r1;
macro dn(u) (N.x*dx(u)+N.y*dy(u) )                    //     定义法向导数

if(anew){ u1= u0; v1= 0; r1 = 1;}
else {
    ifstream g("u.txt");g>>u1[];
```

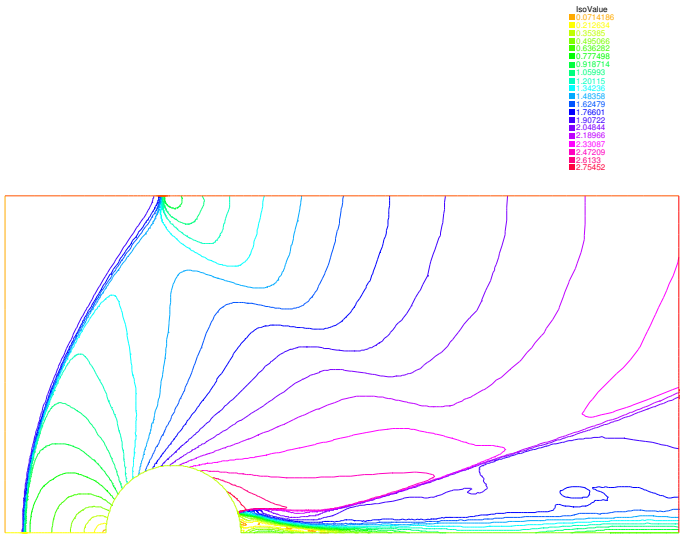


图 3.13: 由(3.11)计算的圆盘周围的欧拉流在2马赫下的压力场

```
    ifstream gg("v.txt");gg>>v1[];
    ifstream ggg("r.txt");ggg>>r1[];
    plot(u1,ps="eta.eps", value=1,wait=1);
    err0=err0/10; dt = dt/10;
}

problem  eul(u,v,r,uh,vh,rh)
   = int2d(Th)(  (u*uh+v*vh+r*rh)/dt
                + ((dx(r)*uh+ dy(r)*vh) - (dx(rh)*u + dy(rh)*v))
               )
 + int2d(Th)(-(rh*convect([u1,v1],-dt,r1) + uh*convect([u1,v1],-dt,u1)
                + vh*convect([u1,v1],-dt,v1))/dt)
  +int1d(Th,6)(rh*u)                                    //     +int1d(Th,1)(rh*v)
 + on(2,r=0) + on(2,u=u0) + on(2,v=0);

int j=80;
for(int k=0;k<3;k++)
{
    if(k==20){ err0=err0/10; dt = dt/10; j=5;}
    for(int i=0;i<j;i++){
       eul; u1=u; v1=v; r1=abs(r);
        cout<<"k="<<k<<"  E="<<int2d(Th)(u^2+v^2+r)<<endl;
        plot(r,wait=0,value=1);
}
Th = adaptmesh (Th,r, nbvx=40000,err=err0,
      abserror=1,nbjacoby=2, omega=1.8,ratio=1.8, nbsmooth=3,
      splitpbedge=1, maxsubdiv=5,rescaling=1) ;
 plot(Th,wait=0);
```

```
 u=u;v=v;r=r;

savemesh(Th,"Th_circle.mesh");
ofstream f("u.txt");f<<u[];
ofstream ff("v.txt");ff<<v[];
ofstream fff("r.txt");fff<<r[];
r1 = sqrt(u*u+v*v);
plot(r1,ps="mach.eps", value=1);
r1=r;
}
```

3.15 方程的分类

摘要 通常情况下确定一个方程的类型是比较困难的，然而使用哪何种近似求法及相应的算法来求解方程则取决于方程的类型：

- 有限元*(LBB* 条件*)*用于处理椭圆型方程组。
- 有限差分法可用于解抛物方程组中的变量以及抛物方程组中时间周期内的椭圆方程，当为时间为隐式时可以得到更稳定的结果。
- 迎风法, *Petrov-Galerkin*法, 特征*-Galerkin*法, 间断*-Galerkin*法以及有限体积法用于处理双曲方程组或带时间周期的双曲方程组。

当方程类型改变之后, 得到的结果也会随之改变 *(*比如间断冲击情况*)*

椭圆方程、抛物方程及双曲方程 偏微分方程(PDE) 是包含未知函数及其偏导数的等式。

$$F(\varphi(x), \frac{\partial\varphi}{\partial x_1}(x), \cdots, \frac{\partial\varphi}{\partial x_d}(x), \frac{\partial^2\varphi}{\partial x_1^2}(x), \cdots, \frac{\partial^m\varphi}{\partial x_d^m}(x)) = 0 \quad \forall x \in \Omega \subset \mathcal{R}^d.$$

方程中x 的取值范围 Ω称为偏微分方程的 定义域。 最高阶偏导数阶数 m称为 阶数。若 F 和 φ 为向量值函数, 则称偏微分方程 组 。
除了特殊指定情况之外, 我们约定 一个偏微分方程 对应一个标准值函数 F和 φ。 若 F 对各参数都是线性的，则 称这个偏微分方程为 线性的。
二阶线性偏微分方程的一般形式为 $\frac{\partial^2\varphi}{\partial x_i \partial x_j}$ ， 我们将$A : B$定义为 $\sum_{i,j=1}^{d} a_{ij}b_{ij}$.

$$\alpha\varphi + a \cdot \nabla\varphi + B : \nabla(\nabla\varphi) = f \quad , \quad \Omega \subset \mathcal{R}^d,$$

其中 $f(x), \alpha(x) \in \mathcal{R}, a(x) \in \mathcal{R}^d, B(x) \in \mathcal{R}^{d\times d}$ 为 偏微分方程 系数。 若系数与 x无关, 称偏微分方程有 常系数。
若用 1替换φ ,z_i替换 $\partial\varphi/\partial x_i$ ， $z_i z_j$替换$\partial^2\varphi/\partial x_i\partial x_j$, z 为 $\mathcal{R}^d$ 中的向量，我们可以得到偏微分方程的二次形式:

$$\alpha + a \cdot z + z^T B z = f.$$

若此为椭圆方程 （若 $d \geq 2$则为椭圆体）, 则称为 椭圆型偏微分方程; 若此为抛物方程或双曲线方程, 则称为 抛物型偏微分方程或 双曲型偏微分方程。 若 $A \equiv 0$, 阶数为1而不是2,我们仍然称这个偏微分方程为双曲型偏微分方程，原因将会在后面章节中阐述。

基于以上叙述，我们可以通过研究偏微分方程的多项式$P(z)$形式在无穷远处是否有分支来

对偏微分系统进行分类。（椭圆体在无穷远处无分支，抛物面在无穷远处有一个分支，双曲面在无穷远处有几个分支）。
若偏微分方程不是线性的，我们称为 非线性偏微分方程。根据线性化方程的类型，我们可以称偏微分方程是局部椭圆型偏微分方程、抛物型偏微分方程或双曲型偏微分方程。
例如对非线性方程

$$\frac{\partial^2\varphi}{\partial t^2}-\frac{\partial\varphi}{\partial x}\frac{\partial^2\varphi}{\partial x^2}=1,$$

我们有 $d=2, x_1=t, x_2=x$ 且它的线性形式为:

$$\frac{\partial^2 u}{\partial t^2}-\frac{\partial u}{\partial x}\frac{\partial^2\varphi}{\partial x^2}-\frac{\partial\varphi}{\partial x}\frac{\partial^2 u}{\partial x^2}=0,$$

对未知的 u ，若 $\frac{\partial\varphi}{\partial x}<0$ 则为局部椭圆型偏微分方程， 若 $\frac{\partial\varphi}{\partial x}>0$则为局部双曲型偏微分方程。

例 拉普拉斯方程为椭圆型方程:

$$\Delta\varphi\equiv\frac{\partial^2\varphi}{\partial x_1^2}+\frac{\partial^2\varphi}{\partial x_2^2}+\cdots+\frac{\partial^2\varphi}{\partial x_d^2}=f,\quad \forall x\in\Omega\subset\mathcal{R}^d.$$

热方程为抛物型方程 ， $Q=\Omega\times]0,T[\subset\mathcal{R}^{d+1}$:

$$\frac{\partial\varphi}{\partial t}-\mu\Delta\varphi=f\quad \forall x\in\Omega\subset\mathcal{R}^d,\quad \forall t\in]0,T[.$$

若 $\mu>0$,则波动 方程为双曲型方程:

$$\frac{\partial^2\varphi}{\partial t^2}-\mu\Delta\varphi=f\quad 在\quad Q\quad 中\quad .$$

当$\mu\neq 0$时，对流扩散方程为抛物型方程，其当μ为其他情况时，对流扩散方程为双曲型方程:

$$\frac{\partial\varphi}{\partial t}+a\nabla\varphi-\mu\Delta\varphi=f.$$

双调和方程为椭圆型方程:

$$\Delta(\Delta\varphi)=f\quad 在\quad \Omega\quad 中\quad .$$

边界条件 仅知道函数和函数导数的方程还不足以确定一个函数，我们还需要知道Ω 的边界$\Gamma=\partial\Omega$ 信息或者 Γ 的部分信息才能确定函数。
这样的信息称为 边界条件。 例如,

$$\varphi(x)\text{ 给定, 对 }\forall x\in\Gamma,$$

称之为 *Dirichlet* 边界条件。 *Neumann*边界条件为

$$\frac{\partial\varphi}{\partial n}(x)\text{ 在 }\Gamma\text{ 上给定}\quad \text{(或对一般的二阶偏微分方程 } n\cdot B\nabla\varphi\text{在 }\Gamma\text{上给定)}$$

其中n 是 $x\in\Gamma$ 点的单位外法向 （其定义为 $\frac{\partial\varphi}{\partial n}=\nabla\varphi\cdot n$）。

另一个经典的边界条件被称作 *Robin* (或 傅里叶) 条件，表示如下:

$$\varphi(x) + \beta(x)\frac{\partial \varphi}{\partial n}(x) \text{ 在 } \Gamma\text{上给定.}$$

通常找到一个边界条件集使得 φ 定义唯一是非常困难的。
一般来说, 一个椭圆方程若在全体边界上有一个 Dirichlet条件，Neumann条件或 Robin条件，则称这个方程为适定的 （即 φ 唯一）。
拉普拉斯方程为适定的若满足Dirichlet条件或Neumann条件，当满足下式时拉普拉斯方程亦为适定的：

$$\varphi \text{ 给定在 } \Gamma_1, \quad \frac{\partial \varphi}{\partial n} \text{ 给定在 } \Gamma_2, \quad \Gamma_1 \cup \Gamma_2 = \Gamma, \quad \dot{\Gamma_1} \cap \dot{\Gamma_2} = \emptyset.$$

对于抛物型方程和双曲型方程，很少情况下需要知道 $\Gamma \times [0, T]$上的所有的边界条件才能适定。 例如热方程只需满足下式即为适定的：

$$\varphi \text{ 在 } t = 0 \text{ 处给定，且边界 } \partial\Omega \text{ 满足Dirichlet条件或 Neumann 条件或混合条件 .}$$

此处t 表示时间，所以第一个条件被称作一个 初始条件。所有的边界条件集可统称为柯西条件。
波动方程为适定的若满足

$$\varphi \text{ 与 } \frac{\partial \varphi}{\partial t} \text{ 在 } t = 0 \text{ 处给定，且边界 } \partial\Omega \text{ 满足Dirichlet条件或 Neumann 条件或混合条件。 .}$$

第 4 章

语法

4.1 数据类型

本质上来讲 FreeFem++ 是一款 编译器:其语言类型多样，并拥有异常处理和返回功能。对于每个变量都需要申明其类型，每个语句由分号“;”隔开。 FreeFem++ 可以对基本的语言类型进行处理，如整数 （int）、实数 （real）, 字符串 （string）、数组 （例: real[int]）、 二维 （2D） 有限元网络 （mesh）, 二维有限元空间 （fespace）、 解析函数 （func）、 有限元函数数组 （func[*basic_type*]）)、 线性与双线性算子、稀疏矩阵及向量等。例如：

```
int i,n=20;                                                    //    i,n 为整数
real[int] xx(n),yy(n);                                         //    两个长度为n的数组
for (i=0;i<=20;i++)                                            //    并按下述声明赋值
 { xx[i]= cos(i*pi/10); yy[i]= sin(i*pi/10); }
```

除变量 fespace 外，其他变量的生命周期由当前大括号 {...}决定。在下述情况中局部括号里的变量声明无效 :

Example 4.1

```
real r= 0.01;
mesh Th=square(10,10);                                         //    单位正方形网格
fespace Vh(Th,P1);                                             //    P1 有限元空间
Vh u = x+ exp(y);
func f = z * x + r * log(y);
plot(u,wait=true);
{                                                              //    新的大括号
  real r = 2;                                                  //    与之前出现的r不一样
  fespace Vh(Th,P1);                                           //    出现错误，因为 Vh 是一个全局名
}                                                              //    括号结束
                                                               //    这里r返回值0.01
```

FreeFem++ 中对变量的声明是强制的; 得益于这一优良的特点使得程序可以避免由隐变量形式可能导致的漏洞。 变量的名称是字母数字混合编制的字符串 , 且变量名字中不允许使用下划线“_” , 因为下划线将会被用作一个运算符。

4.2 主要类型

bool 用于逻辑表达或流量控制。比较（这里特指关系式）的结果是一个布尔型，如在

```
bool fool=(1<2);
```

中，fool 为真。同理可用于 ==, <=, >=, <, >, ! =等符号的比较中。

int 声明一个整数 (等同于C++中的 `long`)

string 声明字符串，用以存储双引号包含的文本, 例如:

```
"This is a string in double quotes."
```

real 声明变量用于存储数字，如 “12.345”. （等同于C++中的 `double`)）。

complex 声明变量用于存储复数, 如 $1 + 2i$, `FreeFem++` 能够理解 $i = \sqrt{-1}$ （等同于C++中的 `complex<double>` ）。

```
complex a =  1i, b = 2 + 3i;
cout << "a + b = " << a + b << endl;
cout << "a - b = " << a + b << endl;
cout << "a * b = " << a * b << endl;
cout << "a / b = " << a / b << endl;
```

以下为运行结果：

```
a + b = (2,4)
a - b = (-2,-2)
a * b = (-3,2)
a / b = (0.230769,0.153846)
```

ofstream 声明输出文件。

ifstream 声明输入文件。

real[int] 声明变量用于存储带整数索引的多个实数。

```
real[int] a(5);
a[0] = 1; a[1] = 2; a[2] = 3.3333333; a[3] = 4; a[4] = 5;
cout << "a = " << a  << endl;
```

程序运行结果为;

```
a = 5   :
  1       2     3.33333   4       5
```

real[string] 声明变量用于存储带字符串索引的多个实数。

string[string] 声明变量用于存储带字符串索引的多个字符串。

func 定义一个没有参数的函数, 如果自变量为 `x, y`。 比如:

```
func f=cos(x)+sin(y) ;
```

注，函数的类型取决于其表达式的类型。能够计算数值幂次，比如， `x^1, y^0.23`。

mesh 形成三角剖分, 见 **??**。

fespace 定义一个新的有限元空间类型, 见节 **??**.

problem 声明一个偏微分问题, 但并不求解。

solve 声明一个问题并求解。

varf 定义一个完整的变分问题。

matrix 定义一个稀疏矩阵。

4.3 全局变量

字符名 `x,y,z,label,region,P,N,nu_triangle...` 是保留字，被用来链接到有限元工具:

x 是当前点的 x 轴 (实变量)

y 是当前点的 y 轴 (实变量)

z 是当前点的 z 轴 (实变量)

label 如果当前点在边界上则为包含该边界标号，否则为0 （整型变量）。

region 返回当前点 （x,y） 的区域标号 （整型变量）。

P 给出当前点 ($\mathbb{R}^2$ 上的变量。). 通过 `P.x, P.y`, 我们可以得到由 x, y 组成的 `P` 。 并且 `P.z` 也被保留而且可以被用于 3 维情形。

N 给出了当前点的单位外法向量，如果当前点在由 `border` （$\mathbb{R}^3$ 上的变量） 定义的曲线上。 `N.x` 和 `N.y` 是法向量的 x 和 y 方向的分量。 `N.z` 是被保留的。 .

lenEdge 给出当前边的长度

$$\texttt{lenEdge} = |q^i - q^j| \quad \text{if the current edge is } [q^i, q^j]$$

hTriangle 给出当前三角形的尺寸

nuTriangle 给出当前三角形的索引指标 （整型变量）。

nuEdge 给出当前边在其三角形中的索引指标 （整型变量）。

nTonEdge 给出与当前边相邻三角形的数量 （ 整数 ）。

area 给出当前三角形的面积 （实变量）。

volume 给出当前四面体的体积 （实变量）。

cout 是标准输出设备 （默认是控制台）。 在 MS-Windows 上, 这时标准输出只有控制台。 (输出流)

cin 是标准输入设备 （默认是键盘）。 （输入流）。

endl 添加一个“行结束”到输入/输出流。

true 在 bool 变量中表示“真。

false 在 bool 变量中表示“假”。

pi 是 π 的实际值的估计值。

4.4 系统命令

这里来说明如何显示一个 FreeFem++ 程序的所有的类型、操作和函数:

```
dumptable(cout);
```

执行系统命令字符串 （在 Carbon MacOS 上不被执行），返回系统调用的值。

```
system("shell command");                    //    在3.12-1后的版本
exec("shell command");
```

这对启动 FreeFem++ 的另一个可执行文件非常有用。在 MS-Windows 上，则必须写出可执行文件的完整路径。例如，如果命令“ls.exe”在子目录 “c:\cygwin\bin\” 下，那么我们必须写

```
exec("c:\\cygwin\\bin\\ls.exe");
```

另一个有用的系统命令是 assert() ，它用来确保括号里面的值为真。

```
assert(version>=1.40);
```

4.5 算术

在整数域 , +, −, ∗ 是通常的算术加法 （加）、 减法 （减） 和乘法 （乘）。特别地，运算符 / 和 % 从用第二个表达式去除第一个表达式来产生商和余数。 如果 / 或 % 的第二个数是零则运算没有定义。 两个整数 a, b 的最大值或最小值通过 max(a,b) 或 min(a,b) 来得到。 两个整数 a, b 的幂 a^b 通过写 a^b来计算。 经典的 C++ “算术条件” 表达式 a ? b : c 的值等于 b，如果表达式 a 的为真。否则表达式的值等于表达式 c 的值。

Example 4.2 整数的计算

```
int a = 12, b = 5;
cout <<"plus, minus of "<<a<<" and "<<b<<" are "<<a+b<<", "<<a-b<<endl;
cout <<"multiplication, quotient of them are "<<a*b<<", "<<a/b<<endl;
cout <<"remainder from division of "<<a<<" by "<<b<<" is "<<a%b<<endl;
cout <<"the minus of "<<a<<" is "<< -a << endl;
cout <<a<<" plus -"<<b<<" need bracket:"<<a<<"+(-"<<b<<")="<<a+(-b)<<endl;
cout <<"max and min of "<<a<<" and "<<b<<" is "<<max(a,b)<<","<<min(a,b)<< endl;
cout <<b<<"th power of "<<a<<" is "<<a^b<< endl;
cout << " min  == (a < b ? a : b )  is " << (a < b ? a : b) << endl;

b=0;
cout <<a<<"/0"<<" is "<< a/b << endl;
cout <<a<<"%0"<<" is "<< a%b << endl;
```

得到如下的结果：

```
plus, minus of 12 and 5 are 17, 7
multiplication, quotient of them are 60, 2
remainder from division of 12 by 5 is 2
the minus of 12 is -12
12 plus -5 need bracket :12+(-5)=7
max and min of 12 and 5 is 12,5
5th power of 12 is 248832
min == (a < b ? a : b)   is 5
12/0 : long long long
Fatal error : ExecError  Div by 0 at exec line  9
Exec error : exit
```

由于 $integer \subset real$，运算“$+, -, *, /, \%$” 和 “ `max, min, ^` ” 扩展到实数或变量。 但是，$\%$ 仍然是计算两个实数整数部分的余数。
下面是类似例子 4.2 的另一些例子

```
real a=sqrt(2.), b = pi;
cout <<"plus, minus of "<<a<<" and "<<pi<<" are "<< a+b <<", "<< a-b << endl;
cout <<"multiplication, quotient of them are "<<a*b<<", "<<a/b<< endl;
cout <<"remainder from division of "<<a<<" by "<<b<<" is "<< a%b << endl;
cout <<"the minus of "<<a<<" is "<< -a << endl;
cout <<a<<" plus -"<<b<<" need bracket :"<<a<<"+(-"<<b<<")="<<a + (-b) << endl;
```

它给出了下面的输出：

```
plus, minus of 1.41421 and 3.14159 are 4.55581, -1.72738
multiplication, quotient of them are 4.44288, 0.450158
remainder from division of 1.41421 by 3.14159 is 1
the minus of 1.41421 is -1.41421
1.41421 plus -3.14159 need bracket :1.41421+(-3.14159)=-1.72738
```

通过关系

$$bool \subset int \subset real \subset complex,$$

运算“$+, -, *, /$”和 “ ^ ” 也适用于复数类型的变量，但是 “$\%$, max, min”不适用。 复数诸如 `5+9i`, `i`$=\sqrt{-1}$ 是合法的表达式。 当实变量 `a=2.45`, `b=5.33`时，像$a+i\,b$ 和 $a+i\sqrt{2.0}$ 的复数必须声明为

```
complex z1 = a+b*1i, z2=a+sqrt(2.0)*1i;
```

一个复数 z 的实部和虚部可以由 **imag** 和 **real**来得到。
$a+bi$ (a, b 是实的) 的共轭定义为 $a-bi$，它也可以用 “conj” 来计算，在 FreeFem++ 中用 **conj**(a+b*1i) 来实现。复数 $z = a+ib$ 被内在地认为是实数 a, b 构成的实数对 (a, b)。我们可以附加它到Cartesian平面上的点 (a, b) 上，其中 x 轴是实数部分而 y 轴是虚数部分。点 (a, b) 有一个极坐标表示 (r, ϕ)，所以 z 它也是 $z = r(\cos\phi + i\sin\phi)$、$r = \sqrt{a^2+b^2}$ 和 $\phi = \tan^{-1}(b/a)$；r 和 ϕ 被称为 z 的模和辐角。在下面的例子中，我们将使用 FreeFem++ 和棣莫弗公式 $z^n = r^n(\cos n\phi + i\sin n\phi)$来演示：

Example 4.3

```
real a=2.45, b=5.33;
complex  z1=a+b*1i, z2 = a+sqrt(2.)*1i;
func string pc(complex z)                    //    输出复数形式 (real)+i(imaginary)
{
   string r = "("+real(z);
   if (imag(z)>=0) r = r+"+";
   return r+imag(z)+"i)";
}
                                  //    输出复数形式 |z|*(cos(arg(z))+i*sin(arg(z)))
func string toPolar(complex z)
{
   return abs(z)+"*(cos("+arg(z)+")+i*sin("+arg(z)+"))";
}
cout <<"Standard output of the complex "<<pc(z1)<<" is the pair "
     <<z1<<endl;
cout <<"Plus, minus of "<<pc(z1)<<" and "<<pc(z2)<<" are "<< pc(z1+z2)
     <<", "<< pc(z1-z2) << endl;
cout <<"Multiplication, quotient of them are "<<pc(z1*z2)<<", "
     <<pc(z1/z2)<< endl;
cout <<"Real/imaginary part of "<<pc(z1)<<" is "<<real(z1)<<", "
     <<imag(z1)<<endl;
cout <<"Absolute of "<<pc(z1)<<" is "<<abs(z1)<<endl;
cout <<pc(z2)<<" = "<<toPolar(z2)<<endl;
cout <<"  and polar("<<abs(z2)<<","<<arg(z2)<<") = "
     << pc(polar(abs(z2),arg(z2)))<<endl;
cout <<"de Moivre's formula: "<<pc(z2)<<"^3 = "<<toPolar(z2^3)<<endl;
cout <<"conjugate of "<<pc(z2)<<" is "<<pc(conj(z2))<<endl;
cout <<pc(z1)<<"^"<<pc(z2)<<" is "<< pc(z1^z2) << endl;
```

这里是例 *4.3* 的输出

```
Standard output of the complex (2.45+5.33i) is the pair (2.45,5.33)
Plus, minus of (2.45+5.33i) and (2.45+1.41421i) are (4.9+6.74421i), (0+3.91579i)
Multiplication, quotient of them are (-1.53526+16.5233i), (1.692+1.19883i)
Real/imaginary part of (2.45+5.33i) is 2.45, 5.33
Absolute of (2.45+5.33i) is 5.86612
(2.45+1.41421i) = 2.82887*(cos(0.523509)+i*sin(0.523509))
  and polar(2.82887,0.523509) = (2.45+1.41421i)
de Moivre's formula: (2.45+1.41421i)^3
                        = 22.638*(cos(1.57053)+i*sin(1.57053))
conjugate of (2.45+1.41421i) is (2.45-1.41421i)
(2.45+5.33i)^(2.45+1.41421i) is (8.37072-12.7078i)
```

4.6 字符串表达式

在接下来的例子中给你举一些字符串表达式的例子：

```
string tt="toto1"+1+" -- 77";                                    //    字符串连接
string t1="0123456789";
string t2;
                                                                 //    新的算符
t2 ="12340005678";
t2(4:3) = "abcdefghijk-";
string t55=t2(4:3);
                                                  //   t2 = "12340abcdefghijk-005678";
cout << t2 << endl;
cout << "  find abc " << t2.find("abc") << endl;
cout << "r find abc " << t2.rfind("abc") << endl;
cout << " find abc from 10  " << t2.find("abc",10) << endl;
cout << " ffind abc from 10 " <<t2.rfind("abc",10) << endl;
cout << "   " << string("abcc").length << endl;
cout << " t55 " << t55 << endl;
{                                              //    添加 getline 版本 3.0-6 jan 2009 FH
string s;
ifstream toto("xyf");
for (int i=0;i<10;++i)
  {
   getline(toto,s);
   cout << i << " : " << s << endl;
  }
}
```

4.7 一元函数

基本函数 都内建在 FreeFem++ 中，诸如 幂函数 x^ y = pow(x,y)$= x^y$、 指数函数 exp(x) $(= e^x)$、 对数函数 log(x)$(= \ln x)$ 或 log10(x) $(= \log_{10} x)$、 三角函数 sin(x), cos(x), tan(x) 假设角度是使用 弧度制来度量; $\sin x$, $\cos x$, $\tan x$ 的逆（也称为 圆函数 或 反三角函数）可通过调用 asin(x)（=arcsin x）, acos(x)（=arccos x）, atan(x)（=arctan x） 实现； 函数 atan2(x,y) 用于计算 y/x 的反正切函数的主值，用两个参数的符号确定了返回值的象限；

双曲函数,

$$\sinh x = \left(e^x - e^{-x}\right)/2, \qquad \cosh x = \left(e^x + e^{-x}\right)/2.$$

和 $\tanh x = \sinh x/\cosh x$ 及其反函数分别通过 sinh(x)、 cosh(x)、 tanh(x)、 asinh(x)、 acosh(x) 和 atanh(x) 形式调用。

$$\sinh^{-1} x = \ln\left[x + \sqrt{x^2+1}\right], \qquad \cosh^{-1} x = \ln\left[x + \sqrt{x^2-1}\right].$$

取整函数 floor(x) 取不大于 x的最大整数值，而 ceil(x) 取不小于 x 的最小整数值；同样地， rint(x) 返回最接近于 x （根据一般的舍入模式）的浮点格式的值。.

初等函数 表示在上面给出了的函数类（多项式函数、指数函数、对数函数、三角函数、圆函数）和它们有限次四则运算后得到的函数

$$f(x)+g(x),\ f(x)-g(x),\ f(x)g(x),\ f(x)/g(x)$$

以及有限次复合后得到的函数 $f(g(x))$。 在 FreeFem++ 中, 所有的初等函数都可以被创建。 一个初等函数的导数仍然是初等函数；但是，初等函数的不定积分并不总是能用初等函数来表达。

Example 4.4 下面是一个用初等函数来构造区域边界的例子。 心形线

```
real b = 1.;
real a = b;
func real phix(real t)
{
   return (a+b)*cos(t)-b*cos(t*(a+b)/b);
}
func real phiy(real t)
{
   return (a+b)*sin(t)-b*sin(t*(a+b)/b);
}
border C(t=0,2*pi) { x=phix(t); y=phiy(t); }
mesh Th = buildmesh(C(50));
```

得到主值，我们可以对 $z\neq 0$ 定义 $\log z$ 如下

$$\ln z=\ln|z|+i\arg z.$$

利用 FreeFem++ , 我们可以计算 `exp(1+4i)`、 `sin(pi+1i)`、 `cos(pi/2-1i)` 和 `log(1+2i)`，然后我们有

$$-1.77679-2.0572i,\quad 1.8896710^{-16}-1.1752i,$$
$$9.4483310^{-17}+1.1752i,\quad 0.804719+1.10715i.$$

随机函数 可以使用 FreeFem++ 的随机数生成函数来定义 （详情见页面 http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html）。 这是范围为 $2^{219937}-1$的一个非常快速和准确的随机数发生器，并且调用它的函数为：

- `randint32()` 生成32位无符号整数。
- `randint31()` 生成31位无符号整数。
- `randreal1()` 生成均匀分布在 $[0,1]$ 的实数（32-位精度）。
- `randreal2()` 生成均匀分布在 $[0,1)$的实数 （32-位精度）。
- `randreal3()` 生成均匀分布在 $(0,1]$ 的实数（32-位精度）。
- `randres53()` 生成均匀分布在 $[0,1)$ 的实数，且有 53- 位分辨率。
- `randinit(seed )` 通过一个可能为零的32-位整数 "seed"初始化状态向量。

库函数 形成数学库 （版本 2.17）。

- 函数 j0(x), j1(x), jn(n,x), y0(x), y1(x), yn(n,x) 是第一类和第二类贝塞尔函数。

  函数 j0(x) 和 j1(x) 分别计算 0阶和 1阶第一类贝塞尔函数； 函数 jn(n, x) 计算 n阶整数阶的第一类贝塞尔函数。

  函数 y0(x) 和 y1(x) 分别计算其线性无关的 0 阶和 1阶的第二类贝塞尔函数，其中 x为正整数（或正实数）；函数 yn(n, x) 计算整数阶 n 阶的第二类贝塞尔函数， 其中 x为正整数（或正实数）。
- 函数 tgamma(x) 计算关于x的 Γ 函数。 lgamma(x) 计算关于x的Γ 函数的绝对值的自然对数。
- 函数 erf(x) 计算误差函数，其中 $\texttt{erf}(x) = \frac{2}{\sqrt{(pi)}} \int_0^x exp(-t^2)dt$。 函数 erfc(x) 计算x的互补误差函数 $\texttt{erfc}(x) = 1 - \texttt{erf}(x)$。

4.8 二元函数

4.8.1 公式

关于两个独立变量 a, b 的二元实函数的一般形式为 $c = f(a, b)$。 在 FreeFem++ 中， x, y 和 z 如 4.3 中介绍的那样是保留字。 所以当函数的两个变量为 x 和 y时，我们无需声明参数即可定义，例如:

```
func f=cos(x)+sin(y) ;
```

注意函数的类型由表达式的类型确定。诸如 x^1、 y^0.23 的幂运算也可以用于函数中。 在 func中，我们可以像下面这样写出一个初等函数

```
func f = sin(x)*cos(y);
func g = (x^2+3*y^2)*exp(1-x^2-y^2);
func h = max(-0.5,0.1*log(f^2+g^2));
```

用复变量创建2个变量 x, y 的函数如下:

```
mesh Th=square(20,20,[-pi+2*pi*x,-pi+2*pi*y]);                    //   [−π, π]^2
fespace Vh(Th,P2);
func z=x+y*1i;                                                    //   z = x + iy
func f=imag(sqrt(z));                                             //   f = ℑ√z
func g=abs( sin(z/10)*exp(z^2/10) );                //   g = |sin z/10 exp z^2/10|
Vh fh = f; plot(fh);                                              //   f的等值线
Vh gh = g; plot(gh);                                              //   g的等值线
```

我们也可以从初等函数 $f(x)$ 或 $g(y)$出发通过有限次四则运算建立 二元初等函数 。

4.8.2 FE-函数

有限元函数也可以像初等函数的算术公式一样去构造，不同的地方在于它是在声明时取值并且 FreeFem++ 存储变量数组的位置与其有限元类型的自由度有关。还有一个不同是，初等函数的值只在需要时才计算。因此，FE-函数要通过公式和有限单元网格来共同

定义。如果一个FE-函数在一个没有自由度的点取值则会导致插值错误，相比之下，初等函数可以在任意一点精确赋值。

```
func f=x^2*(1+y)^3+y^2;
mesh Th = square(20,20,[-2+4*x,-2+4*y]);                    //    正方形 ]−2,2[²
fespace Vh(Th,P1);
Vh fh=f;                                          //    fh 是 f 在 Vh 上的投影（实值）
func zf=(x^2*(1+y)^3+y^2)*exp(x+1i*y);
Vh<complex> zh = zf;                              //    zh 是 zf 在复值Vh空间的投影
```

fh $(=f_h)$ 的构造在 **??**中已经解释过。

Note 4.1 命令 **plot** 只能应用于实或复的*FE*-函数 （*2* 维或*3* 维）上而不能用于初等函数。

用复变量函数创建2 个变量 x, y的函数如下，

```
mesh Th=square(20,20,[-pi+2*pi*x,-pi+2*pi*y]);              //    ]−π,π[²
fespace Vh(Th,P2);
func z=x+y*1i;                                              //    z = x + iy
func f=imag(sqrt(z));                                       //    f = ℑ√z
func g=abs( sin(z/10)*exp(z^2/10) );                //    g = |sin z/10 exp z²/10|
Vh fh = f; plot(fh);                                //    图4.1 f 的等值线
Vh gh = g; plot(gh);                                //    图4.2 g 的等值线
```

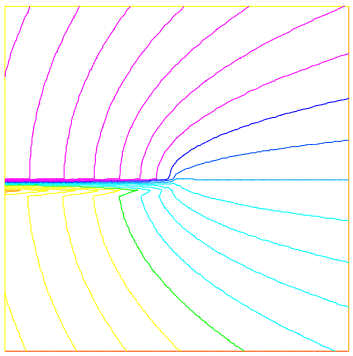


图 4.1: $\Im\sqrt{z}$ 有分支

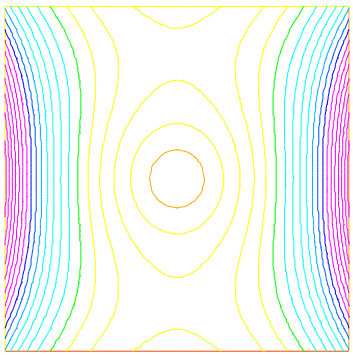


图 4.2: $|\sin(z/10)\exp(z^2/10)|$

4.9 数组

数组 用于存储多个元素，且有两种类型的数组： 第一类是类似于 向量，也就是有 整数索引的数组；第二类是具有字符串索引的数组。 对于第一类，在程序执行时必须知道数组的大小，并且通过 KN<>类来执行， KN<>的所有向量操作均可执行。例如：

```
real [int] tab(10), tab1(10);                       //    2个数组，分别有 10 个实数
real [int] tab2;                                    //    没有给定大小的错误数组
tab = 1.03;                                         //    令所有的数组为1.03
```

```
tab[1]=2.15;
cout << tab[1] << " " << tab[9] << " size of tab = "
     << tab.n << " min: " << tab.min << "  max:" << tab.max
     << " sum : "   << tab.sum <<   endl;                                    //
tab.resize(12);                    //    改变数组 tab 的大小，设置为12，并保留原来的值
tab(10:11)=3.14;                                                  //    设置其余的值
cout <<" resize tab: " <<  tab << endl;
real [string] tt;
tt["+"]=1.5;
cout<<tt["a"]<<"  "<<tt["+"]<<endl;
real[int]  a(5),b(5),c(5),d(5);
a = 1;
b = 2;
c = 3;
a[2]=0;
d = ( a ? b : c );        //    for i = 0, n-1 :  d[i] = a[i] ?  b[i] :  c[i] ,
cout << " d = ( a ? b : c )  is " << d << endl;
d = ( a ? 1 : c ); //    for i = 0, n-1:  d[i] = a[i] ?  1 :  c[i] ,  (v2.23-1)
d = ( a ? b : 0 ); //    for i = 0, n-1:  d[i] = a[i] ?  b[i] :  0 ,  (v2.23-1)
d = ( a ? 1 : 0 );  //    for i = 0, n-1:  d[i] = a[i] ?  0 :  1 ,    (v2.23-1)
tab.sort ;                                          //    对数组 tab 排序 （版本 2.18）
cout << " tab (after sort) "  << tab << endl;
int[int] ii(0:d.n-1);         //    设置数组 ii 为 0,1, ..., d.n-1          (v3.2)
d=-1:-5;                   //    设置 d 为 -1,-2, ..  -5                    (v3.2)

sort(d,ii);                                                  //    对数组 d 和 ii 并行排序
cout << " d " << d << "\n ii = " << ii << endl;
```

生成的输出如下：

```
2.15 1.03 size of tab = 10 min: 1.03  max:2.15 sum : 11.42
 resize tab: 12
        1.03    2.15    1.03    1.03    1.03
        1.03    1.03    1.03    1.03    1.03
        3.14    3.14
 0  1.5
 d = ( a ? b : c )  is  5
          3       3       2       3       3
 tab (after sort) 12
1.03 1.03 1.03 1.03 1.03
1.03 1.03 1.03 1.03 2.15
3.14 3.14
 d 5
 -5  -4  -3  -2  -1

 ii = 5
  4   3   2   1   0
```

数组可以像在 matlab 或 scilab中那样用 `::`来设置，用 `a:c`和用 `a:1:c`来产生数组是等价的。用 `a:b:c`设置后的数组的大小为 $\lfloor |(b-a)/c| + 1 \rfloor$ ，且其索引变量 i 设置为$a + i(b-a)/c$。
第三个例子是对于 整型，实数型，复数型 数组，有两个运算 (.in, .re) 来从复数数组分别生成实部和虚部的实数数组(不通过拷贝) :

```
//    版本 3.2 2009年5月
```

```
                                                        //    类似于 matlab 和 scilab
{
int[int] tt(2:10);                                          //    2,3,4,5,6,7,8,9,10
int[int] t1(2:3:10);                                                //    2,5,8,
cout << " tt(2:10)= " << tt << endl;
cout << " t1(2:3:10)= " << t1 << endl;
tt=1:2:5;
cout << " 1.:2:5 =>  " << tt << endl;
}

{
real[int] tt(2:10);                                         //    2,3,4,5,6,7,8,9,10
real[int] t1(2.:3:10.);                                             //    2,5,8,
cout << " tt(2:10)= " << tt << endl;
cout << " t1(2:3:10)= " << t1 << endl;
tt=1.:0.5:3.999;
cout << " 1.:0.5:3.999 =>  " << tt << endl;
}
{
complex[int] tt(2.+0i:10.+0i);                              //    2,3,4,5,6,7,8,9,10
complex[int] t1(2.:3.:10.);                                         //    2,5,8,
cout << " tt(2.+0i:10.+0i)= " << tt << endl;
cout << " t1(2.:3.:10.)= " << t1 << endl;
cout << " tt.re real part array   " << tt.re << endl ;
                                                        //    复数数组的实部构成的数组
cout << " tt.im imag part array   " << tt.im << endl ;
                                                        //    复数数组的虚部构成的数组

}
```

输出是：

```
 tt(2:10)= 9
  2   3   4   5   6
  7   8   9  10
 t1(2:3:10)= 3
  2   5   8
 1.:2:5 =>  3
  1   3   5
 tt(2:10) = = 9
  2   3   4   5   6
  7   8   9  10
 t1(2.:3:10.)= 3
  2   5   8
 1.:0.5:3.999 =>  6
  1 1.5   2 2.5   3
3.5
 tt(2.+0i:10.+0i)= 9
(2,0) (3,0) (4,0) (5,0) (6,0)
(7,0) (8,0) (9,0) (10,0)
 t1(2.:3.:10.);= 3
(2,0) (5,0) (8,0)

 tt.re real part array   9
  2   3   4   5   6
  7   8   9  10
```

```
 tt.im imag part array   9
  0   0   0   0   0
  0   0   0   0
2
```

所有整数运算符是：

```
{
int N=5;
real[int] a(N),b(N),c(N);
a =1;
a(0:4:2) = 2;
a(3:4) = 4;
cout <<" a = " << a << endl;
b = a+ a;
cout <<" b = a+a : " << b << endl;
b += a;
cout <<" b += a : " << b << endl;
b += 2*a;
cout <<" b += 2*a : " << b << endl;
b /= 2;
cout <<" b /= 2 : " << b << endl;
b .*= a;                                          //    相当于 b = b .* a
cout << "b.*=a; b =" << b << endl;
b ./= a;                                          //    相当于 b = b ./ a
cout << "b./=a; b =" << b << endl;
c = a+b;
cout << " c =a+b : c=" << c << endl;
c = 2*a+4*b;
cout << " c =2*a+4b : c= " << c << endl;
c = a+4*b;
cout << " c =a+4b : c= " << c << endl;
c = -a+4*b;
cout << " c =-a+4b : c= " << c << endl;
c = -a-4*b;
cout << " c =-a-4b : c= " << c << endl;
c = -a-b;
cout << " c =-a-b : c= " << c << endl;

c = a .* b;
cout << " c =a.*b  : c= " << c << endl;
c = a ./ b;
cout << " c =a./b  : c= " << c << endl;
c = 2 * b;
cout << " c =2*b   : c= " << c << endl;
c =  b*2 ;
cout << " c =b*2   : c= " << c << endl;

/*  this operator do not exist
c =  b/2 ;
cout << " c =b/2   : c= " << c << endl;
*/

                                                  //    ---- 方法 --
cout << " ||a||_1     = " <<  a.l1     << endl;                    //
```

```
cout << " ||a||_2     = " <<  a.l2     << endl;                                  //
cout << " ||a||_infty = " <<  a.linfty << endl;                                  //
cout << " sum a_i     = " <<  a.sum    << endl;                                  //
cout << " max a_i     = " <<  a.max    <<    " a[ "   <<   a.imax   <<   " ] = "
<<   a[a.imax]   << endl;                                                       //
cout << " min a_i     = " <<  a.min    <<    " a[ "   <<   a.imin   <<   " ] = "
<<   a[a.imin]   << endl;                                                       //

cout << " a'*a        = " <<  (a'*a)   << endl;                                 //
cout << " a quantile 0.2 = " <<  a.quantile(0.2) << endl;                       //
                                                                    //    数组映射
int[int]  I = [2, 3, 4, -1, 3];
b = c = -3;
b = a(I);                  //    for( i=0;i<b.n;i++) if(I[i] >=0) b[i]=a[I[i]];
c(I) = a;                  //    for( i=0;i<I.n;i++) if(I[i] >=0) C(I[i])=a[i];
cout << " b = a(I) : " << b << "\n  c(I) = a " << c << endl;
c(I) += a;                //    for( i=0;i<I.n;i++) if(I[i] >=0) C(I[i])+=a[i];
cout << " b = a(I) : " << b << "\n  c(I) = a " << c << endl;

}
```

生成输出：

```
5
          3       3       2       3       3

==   3       3       2       3       3
 a = 5
          2       1       2       4       4

 b = a+a : 5
          4       2       4       8       8

 b += a : 5
          6       3       6      12      12

 b += 2*a : 5
         10       5      10      20      20

 b /= 2 : 5
          5     2.5       5      10      10

b.*=a; b =5
         10     2.5      10      40      40

b./=a; b =5
          5     2.5       5      10      10

 c =a+b : c=5
          7     3.5       7      14      14

 c =2*a+4b : c= 5
         24      12      24      48      48

 c =a+4b : c= 5
         22      11      22      44      44
```

```
c =-a+4b : c= 5
        18        9       18       36       36

c =-a-4b : c= 5
       -22      -11      -22      -44      -44

c =-a-b : c= 5
        -7     -3.5       -7      -14      -14

c =a.*b  : c= 5
        10      2.5       10       40       40

c =a./b  : c= 5
       0.4      0.4      0.4      0.4      0.4

c =2*b   : c= 5
        10        5       10       20       20

c =b*2   : c= 5
        10        5       10       20       20

||a||_1     = 13
||a||_2     = 6.403124237
||a||_infty = 4
sum a_i     = 13
max a_i     = 4    a[3] = 4
min a_i     = 1    a[1] = 1
a'*a        = 41
a quantile 0.2 = 2
b = a(I) : 5
         2        4        4       -3        4

 c(I) = a 5
        -3       -3        2        4        2
b = a(I) : 5
         2        4        4       -3        4

 c(I) = a 5
        -3       -3        4        9        4
```

Note 4.2 分位点是一个随机变量的累积分布函数在其正则区间上所取的点。这里的数组取值是随机的。
对于一个给定的 $q \in [0,1]$, 统计函数 *a.quantile(q)* 从一个尺寸为 n 的数组 a 计算出 v, 使得

$$\#\{i/a[i] < v\} \sim q * n;$$

当数组 a 已排序时,它等价于 $v = a[q * n]$。

例：重新编号的数组 （版本 2.3 或更优版本）。 重新编号总是针对一个整数数组, 如果数组的某个元素取值是负的, 则映射无法描述, 从而该取值也不固定。

```
int[int] I=[2,3,4,-1,0];                                    //    用于重新编号的整数映射
b=c=-3;
b= a(I);                    //    for( i=0;i<b.n;i++) if(I[i] >=0) b[i]=a[I[i]];
c(I)= a;                    //    for( i=0;i<I.n;i++) if(I[i] >=0) C(I[i])=a[i];
cout << " b = a(I) : " << b << "\n  c(I) = a " << c << endl;
```

输出是：

```
 b = a(I) : 5
          2        4        4       -3        2

  c(I) = a 5
          4       -3        2        1        2
```

4.9.1 双整数指标数组与矩阵

下面给出一些将满阵转为稀疏阵的例子：

```
  int N=3,M=4;

  real[int,int] A(N,M);
  real[int]  b(N),c(M);
  b=[1,2,3];
  c=[4,5,6,7];

  complex[int,int]  C(N,M);
  complex[int]  cb=[1,2,3],cc=[10i,20i,30i,40i];


  b=[1,2,3];

  int [int] I=[2,0,1];
  int [int] J=[2,0,1,3];

  A=1;                                                       //    矩阵所有值设置为1
  A(2,:) = 4;                                              //    第三行所有值设置为4
  A(:,1) = 5;                                               //    第二列所有值设为5
  A(0:N-1,2) = 2;                                           //    第三列所有值设为2
  A(1,0:2) = 3;                                            //    第二行前三个值设为3

  cout << " A = " << A << endl;

                                                                                //    外积
  C  =  cb*cc';
  C +=  3*cb*cc';
  C -=  5i*cb*cc';
  cout << " C = " << C << endl;
                                                            //    将数组转化为稀疏矩阵
  matrix B;
  B = A;
```

```
  B=A(I,J);                                                    //    B(i,j)= A(I(i),J(j))
  B=A(I^-1,J^-1);                                              //    B(I(i),J(j))= A(i,j)

  A = 2.*b*c';                                                                  //    外积
  cout << " A = " << A << endl;
  B = b*c';                                                    //    外积 B(i,j) = b(i)*c(j)
  B = b*c';                                                    //    外积 B(i,j) = b(i)*c(j)
  B = (2*b*c')(I,J);                                 //    外积 B(i,j) = b(I(i))*c(J(j))
  B = (3.*b*c')(I^-1,J^-1);                          //    外积 B(I(i),J(j)) = b(i)*c(j)
  cout << "B = (3.*b*c')(I^-1,J^-1) =  " << B << endl;
```

其输出是

```
 b = a(I) : 5
          2        4        4       -3        2

  c(I) = a 5
          4       -3        2        1        2

 A = 3 4
           1    5    2    1
           3    3    3    1
           4    5    2    4

 C = 3 4
         (-50,-40) (-100,-80) (-150,-120) (-200,-160)
         (-100,-80) (-200,-160) (-300,-240) (-400,-320)
         (-150,-120) (-300,-240) (-450,-360) (-600,-480)

 A = 3 4
           8   10   12   14
          16   20   24   28
          24   30   36   42
```

4.9.2 矩阵构造与设置

- 改变矩阵相关的线性系统求解程序, 运行

  ```
  set(M,solver=sparsesolver);
  ```

  系统默认的求解程序是 GMRES。

- 来自变分形式: （详见 6.12节 167 页）

```
varf  vDD(u,v) = int2d(Thm)(u*v*1e-10);
matrix DD=vDD(Lh,Lh);
```

- 设置一个常数矩阵

```
 matrix A =
             [[ 0, 1, 0, 10],
              [ 0,  0,  2, 0],
              [ 0, 0, 0,  3],
```

```
                [ 4,0 , 0, 0]];
```

- 设置一个分块矩阵

```
 matrix M=[
            [ Asd[0] ,0      ,0      ,0      ,Csd[0] ],
            [ 0      ,Asd[1] ,0      ,0      ,Csd[1] ],
            [ 0      ,0      ,Asd[2] ,0      ,Csd[2] ],
            [ 0      ,0      ,0      ,Asd[3] ,Csd[3] ],
            [ Csd[0]',Csd[1]',Csd[2]',Csd[3]',DD     ]
];

                                                                    //     将右端打包
  real[int] bb =[rhssd[0][], rhssd[1][],rhssd[2][],rhssd[3][],rhsl[] ];
  set(M,solver=sparsesolver);
  xx = M^-1 * bb;
  [usd[0][],usd[1][],usd[2][],usd[3][],lh[]] = xx;
                                                          //     将解放置到各部分
```

这里的 Asd 和 Csd 是矩阵数组 (来自 examples++-tuturial 的例子 mortar-DN-4.edp).

- 为了设置或获得稀疏矩阵 A 所有的指标和系数, 让 I,J,C 分别为两个 int[int] 数组和一个 real[int] 数组， 这三个数组定义矩阵如下：

$$A = \sum_k \mathtt{C[k]} M_{\mathtt{I[k]},\mathtt{J[k]}} \qquad \text{其中} \qquad M_{ab} = (\delta_{ia}\delta_{jb})_{ij}$$

我们有: M_{ab} 为基本矩阵, 其中唯一非零项为 $m_{ab} = 1$。

记 [I,J,C]=A ; 获得矩阵 A （该数组被自动调整过尺寸） 的所有项, 且 A=[I,J,C] ; 改变所有的项矩阵。注意到矩阵的尺寸是 n= I.max 和 m=J.max。 备注: 建立对角矩阵时，无需 I,J 或 n, m。

- 矩阵重新编号

```
int[int] I(15),J(15);                                  //     两个用于重新编号的数组
                                                                          //
                                                  //    目的是将一个矩阵转化为一个稀疏矩阵
matrix B;
B = A;                                                          //     矩阵 A 的副本
B=A(I,J);                                                 //     B(i,j) = A(I(i),J(j))
B=A(I^-1,J^-1);                                           //     B(I(i),J(j))= A(i,j)
B.resize(10,20);                                  //     重新调整稀疏矩阵的尺寸并去除约束项
```

其中 A 是一个给定的矩阵。

- 复稀疏矩阵与实稀疏矩阵:

```
matrix<complex> C=vv(Xh,Xh);
matrix  R=vr(Xh,Xh);
matrix<complex> CR=R; C=R;                                //     生成或复制实矩阵或复矩阵
R=C.im; R=C.re;                                           //     得到复稀疏矩阵的实部或虚部
matrix CI=C.im,  CR=C.re;                                 //     得到复稀疏矩阵的实部或虚部
```

4.9.3 矩阵运算

运算符 *, /, 和 %, 运算顺序从左到右。

- ' 是矩阵的 （一元） 右转置, 在实数情况下为一般矩阵转置而在复数情况下为 Hermitian 转置。
- .* 是逐项乘积运算符。
- ./ 是逐项除法运算符。

还有一些混合运算符:

- ^-1 用于解线性系统 （例如: b = A^-1 x）。
- ' * 是转置和矩阵乘积的复合, 即点积 （例如 real DotProduct=a'*b） , 在复数情况下即为 Hermitian 积, 所以数学上有 a'*b= $\overline{a}^T b$ 。
- a*b' 是外积 （例如 matrix B=a'*b ）。

Example 4.5

```
mesh Th = square(2,1);
fespace Vh(Th,P1);
Vh f,g;
f = x*y;
g = sin(pi*x);
Vh<complex> ff,gg;                                    //    一个复值有限元函数
ff= x*(y+1i);
gg = exp(pi*x*1i);
varf mat(u,v) =
  int2d(Th)(1*dx(u)*dx(v)+2*dx(u)*dy(v)+3*dy(u)*dx(v)+4*dy(u)*dy(v))
  + on(1,2,3,4,u=1);

varf mati(u,v) =
  int2d(Th)(1*dx(u)*dx(v)+2i*dx(u)*dy(v)+3*dy(u)*dx(v)+4*dy(u)*dy(v))
  + on(1,2,3,4,u=1);

matrix A = mat(Vh,Vh); matrix<complex> AA = mati(Vh,Vh);    //    一个复稀疏矩阵

Vh m0; m0[] = A*f[];
Vh m01; m01[] = A'*f[];
Vh m1; m1[] = f[].*g[];
Vh m2; m2[] = f[]./g[];
cout << "f = " << f[] << endl;
cout << "g = " << g[] << endl;
cout << "A = " << A << endl;
cout << "m0 = " << m0[] << endl;
cout << "m01 = " << m01[] << endl;
cout << "m1 = "<< m1[] << endl;
cout << "m2 = "<< m2[] << endl;
cout << "dot Product = "<< f[]'*g[] << endl;
cout << "hermitien Product = "<< ff[]'*gg[] << endl;
cout << "outer Product = "<< (A=ff[]*gg[]') << endl;
```

```
cout << "hermitien outer Product = "<< (AA=ff[]*gg[]') << endl;
real[int] diagofA(A.n);
  diagofA = A.diag;                                          //    得到矩阵的对角线
  A.diag = diagofA ;                                         //    设置矩阵的对角线
                                                             //    版本 2.17 或更优版本 ---

int[int] I(1),J(1); real[int] C(1);
[I,J,C]=A;                                  //    得到矩阵 A 的稀疏项 （数组被重新调整尺寸）
cout << " I= " << I << endl;
cout << " J= " << J << endl;
cout << " C= " << C << endl;
A=[I,J,C];                                                   //    设置一个新矩阵
matrix D=[diagofA] ;                                //    由数组 A 的对角线设置一个对角阵 D。
cout << " D = " << D << endl;
```

第二种情况只是*STL*[1] *[26]*的一个映射, 所以向量运算是不允许的, 除非选中某一个项。
和*Matlab* 或 *Scilab*中一样, 转置或 *Hermitian* 共轭运算符是 ', 所以计算两个数组 `a,b` 的点积是 `real ab= a'*b`。
允许对稀疏矩阵 A 重新调整尺寸 :

```
A.resize(10,100);
```

注意到新的尺寸可以比原先的尺寸更大或者更小, 所以新的项都设置为零。
基于图形 *2.4* 的三角剖分, 我们有:

$$
\begin{aligned}
A &= \begin{bmatrix} 10^{30} & 0.5 & 0. & 30. & -2.5 & 0. \\ 0. & 10^{30} & 0.5 & 0. & 0.5 & -2.5 \\ 0. & 0. & 10^{30} & 0. & 0. & 0.5 \\ 0.5 & 0. & 0. & 10^{30} & 0. & 0. \\ -2.5 & 0.5 & 0. & 0.5 & 10^{30} & 0. \\ 0. & -2.5 & 0. & 0. & 0.5 & 10^{30} \end{bmatrix} \\
\{v\} = \texttt{f[]} &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0.5 & 1 \end{pmatrix}^T \\
\{w\} = \texttt{g[]} &= \begin{pmatrix} 0 & 1 & 1.2\times10^{-16} & 0 & 1 & 1.2\times10^{-16} \end{pmatrix} \\
\texttt{A*f[]} &= \begin{pmatrix} -1.25 & -2.25 & 0.5 & 0 & 5\times10^{29} & 10^{30} \end{pmatrix}^T \quad (= A\{v\}) \\
\texttt{A'*f[]} &= \begin{pmatrix} -1.25 & -2.25 & 0 & 0.25 & 5\times10^{29} & 10^{30} \end{pmatrix}^T \quad (= A^T\{v\}) \\
\texttt{f[].*g[]} &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0.5 & 1.2\times10^{-16} \end{pmatrix}^T \quad = (v_1w_1 \quad \cdots \quad v_Mw_M)^T \\
\texttt{f[]./g[]} &= \begin{pmatrix} -\texttt{NaN} & 0 & 0 & -\texttt{NaN} & 0.5 & 8.1\times10^{15} \end{pmatrix}^T \quad = (v_1/w_1 \cdots v_M/w_M)^T \\
\texttt{f[]'*g[]} &= 0.5 \quad (= \{v\}^T\{w\} = \{v\}\cdot\{w\})
\end{aligned}
$$

I, J, C 数组的输出:

```
I= 18
         0        0        0        1        1
         1        1        2        2        3
         3        4        4        4        4
         5        5        5
J= 18
         0        1        4        1        2
```

[1]标准模板库, 现在是标准 C++ 的一部分

```
          4       5       2       5       0
          3       0       1       3       4
          1       4       5
 C= 18
        1e+30    0.5     -2.5    1e+30    0.5
        0.5      -2.5    1e+30   0.5      0.5
        1e+30    -2.5    0.5     0.5      1e+30
        -2.5     0.5     1e+30
```

对角稀疏矩阵 D 的输出（警告：对于*fortran*语言的界面，输出指标由 *1* 开始,而在 *FreeFem++* 中类似于 *C*, 指标由 *0* 开始）：

```
 D = # Sparce Matrix (Morse)
# first line: n m (is symmetic) nbcoef
# after for each nonzero coefficient:   i j a_ij where (i,j) \in  {1,...,n}x{1,...,m}
6 6 1  6
       1         1 1.0000000000000000199e+30
       2         2 1.0000000000000000199e+30
       3         3 1.0000000000000000199e+30
       4         4 1.0000000000000000199e+30
       5         5 1.0000000000000000199e+30
       6         6 1.0000000000000000199e+30
```

Note 4.3 运算符 `^-1` 不能用于生成矩阵; 下面给出一个错误的例子：

```
matrix AAA = A^-1;
```

在例 *examples++-load/lapack.edp* 中一个满阵通过 lapack 库和该小型动态链接界面实现求逆 （详见 *C* 节 *333* 页）。

```
load "lapack"
load "fflapack"
int n=5;
real[int,int] A(n,n),A1(n,n),B(n,n);
for(int i=0;i<n;++i)
for(int j=0;j<n;++j)
  A(i,j)= (i==j) ? n+1 : 1;
cout << A << endl;
A1=A^-1;                                          //    定义在 载入 "lapack"中
cout << A1 << endl;

B=0;
for(int i=0;i<n;++i)
  for(int j=0;j<n;++j)
    for(int k=0;k<n;++k)
      B(i,j) +=A(i,k)*A1(k,j);
cout << B << endl;
                                                  //    A1+A^-1; 注意运行失败

inv(A1);                                          //    定义在 载入 "fflapack"中
cout << A1 << endl;
```

输出是:

```
5 5
   6   1   1   1   1
   1   6   1   1   1
```

```
   1   1   6   1   1
   1   1   1   6   1
   1   1   1   1   6

 error:  dgesv_ 0
5 5
 0.18 -0.02 -0.02 -0.02 -0.02
 -0.02 0.18 -0.02 -0.02 -0.02
 -0.02 -0.02 0.18 -0.02 -0.02
 -0.02 -0.02 -0.02 0.18 -0.02
 -0.02 -0.02 -0.02 -0.02 0.18

5 5
   1 -1.387778781e-17 -1.040834086e-17 3.469446952e-17   0
 -1.040834086e-17   1 -1.040834086e-17 -2.081668171e-17   0
 3.469446952e-18 -5.551115123e-17   1 -2.081668171e-17 -2.775557562e-17
 1.387778781e-17 -4.510281038e-17 -4.857225733e-17   1 -2.775557562e-17
 -1.387778781e-17 -9.714451465e-17 -5.551115123e-17 -4.163336342e-17   1

5 5
   6   1   1   1   1
   1   6   1   1   1
   1   1   6   1   1
   1   1   1   6   1
   1   1   1   1   6

 error:  dgesv_ 0
5 5
 0.18 -0.02 -0.02 -0.02 -0.02
 -0.02 0.18 -0.02 -0.02 -0.02
 -0.02 -0.02 0.18 -0.02 -0.02
 -0.02 -0.02 -0.02 0.18 -0.02
 -0.02 -0.02 -0.02 -0.02 0.18

5 5
   1 -1.387778781e-17 -1.040834086e-17 3.469446952e-17   0
 -1.040834086e-17   1 -1.040834086e-17 -2.081668171e-17   0
 3.469446952e-18 -5.551115123e-17   1 -2.081668171e-17 -2.775557562e-17
 1.387778781e-17 -4.510281038e-17 -4.857225733e-17   1 -2.775557562e-17
 -1.387778781e-17 -9.714451465e-17 -5.551115123e-17 -4.163336342e-17   1

5 5
   6   1   1   1   1
   1   6   1   1   1
   1   1   6   1   1
   1   1   1   6   1
   1   1   1   1   6
```

为了计算 *lapack.cpp* 或 *fflapack.cpp*, 你的系统必须拥有 *lapack* 库并且在目录 *examples++-load* 内操作。

```
ff-c++ lapack.cpp -llapack
ff-c++ fflapack.cpp -llapack
```

4.9.4 其他数组

我们也可以用同样的方法对FE函数写出相应的数组。如果需要的话，可以当作 向量值函数 来处理。 , 空间或者向量的有限函数排列为

```
int n = 100;                                          //    数组的大小
Vh[int] wh(n);                                        //    实数情况
Wh[int] [uh,vh](n);                                   //    实向量情况
Vh<complex>[int] cwh(n);                              //    复数情况
Wh<complex>[int] [cuh,cvh](n);                        //    复向量情况
[cuh[2],cvh[2]]= [x,y];                    //    给复向量的第二个指标赋值
```

Example 4.6 下面我们给定三个不同的函数求解泊松方程， $f = 1, \sin(\pi x)\cos(\pi y), |x-1||y-1|$, 其中，这三个函数的解储存在*FE*函数的数组中。

```
mesh Th=square(20,20,[2*x,2*y]);
fespace Vh(Th,P1);
Vh u, v, f;
problem Poisson(u,v) =
    int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
     + int2d(Th)( -f*v ) + on(1,2,3,4,u=0) ;
Vh[int] uu(3);                                        //    FE函数的数组
f=1;                                                  //    问题1
Poisson; uu[0] = u;
f=sin(pi*x)*cos(pi*y);                                //    问题2
Poisson; uu[1] = u;
f=abs(x-1)*abs(y-1);                                  //    问题3
Poisson; uu[2] = u;
for (int i=0; i<3; i++)                               //    绘制所有解的图像
  plot(uu[i], wait=true);
```

4.10 映射数组

对于第二种情况，这只是对于 STL[2][26] 的映射，只能对向量中的一个元素进行操作，不能对向量整体操作。

```
real[string] map;                                     //    动态数组
for (i=0;i<10;i=i+1)
  {
    tab[i] = i*i;
    cout << i << " " << tab[i] << "\n";
  };

map["1"]=2.0;
map[2]=3.0;                                //    2自然对应第三个字符

cout << " map[\"1\"] = " << map["1"] << "; "<< endl;
cout << " map[2] = " << map[2] << "; "<< endl;
```

[2]标准模板库, 现在是标准 C++ 的一部分

4.11 循环

在 FreeFem++ 中也有 for 与 while ， break 与 continue 等循环关键词。
在for循环中有三个参数； 控制变量的初始化 （INITIALIZATION）、继续的条件（CONDITION）、控制变量的变化（CHANGE）。 当 条件 为真的时候，for循环继续。

```
for (INITIALIZATION; CONDITION; CHANGE)
     { BLOCK of calculations }
```

下面的例子演示了从1到10的求和，结果显示在 sum中，

```
int sum=0;
for (int i=1; i<=10; i++)
   sum += i;
```

while循环

```
while (CONDITION) {
   BLOCK of calculations or change of control variables
}
```

循环一直进行，直到（CONDITION） 为假。 从1到10的求和也可以用 **while** 来计算：

```
int i=1, sum=0;
while (i<=10) {
  sum += i; i++;
}
```

使用 **break**可以在中间退出循环。 而 **continue** 可以跳过执行从 *continue* 到循环结束的部分。

Example 4.7

```
for (int i=0;i<10;i=i+1)
    cout << i << "\n";
real eps=1;
while (eps>1e-5)
 { eps = eps/2;
   if( i++ <100) break;
   cout << eps << endl;}

for (int j=0; j<20; j++) {
   if (j<10) continue;
   cout << "j = " << j << endl;
}
```

4.12 输入/输出

有关输入输出的语法和 C++ 的语法很相似。 使用 **cout**, **cin**, **endl**, <<,>>.
为了写一个文档 (resp. read from)， 声明一个新的变量 `ofstream ofile("filename");` 或者 `ofstream ofile("filename",append);` (resp. `ifstream ifile("filename");`) 然后使用 `ofile` (resp. `ifile`) 作为 `cout` (resp. `cin`)。
附加（append） 在 “ `ofstream ofile("filename",append);` ” 中的意思是在附加路径中以二进制形式打开该文件。

Note 4.4 文件会在封闭区块的出口处关闭。

Example 4.8

```
int i;
cout << " std-out" << endl;
cout << " enter i= ? ";
cin >> i ;
{
  ofstream f("toto.txt");
  f << i << "coucou'\n";
};                                          //     关闭文件f，因为变量f被删除了

{
  ifstream f("toto.txt");
   f >> i;
}
{
  ofstream f("toto.txt",append);
                                            //     附加到现有文件"toto.txt"中
  f << i << "coucou'\n";
};                                          //     关闭文件f，因为变量f被删除了

  cout << i << endl;
```

一些函数可以用来规范输出格式。

- *int nold=f.***precision***(n)* 设定小数点右面的数字的位数。适用于接下来所有写进输出流的浮点数。 然而， 这并不能让整型有小数点。因此考虑到这个影响必须使用固定小数点。
- *f.***scientific** 把浮点数用科学计数法表示 *(d.dddEdd)*。
- *f.***fixed** 把浮点数改写成固定小数点的表示 *(d.ddd)* 与*scientific*相反。
- *f.***showbase** 根据*C++*对积分常数的语法形式改写插入，使得转变为外部可读的形式。默认不设定值*showbase*。
- *f.***noshowbase** 撤销 *showbase* 的设定。
- *f.***showpos** 在一个正的积分值的十进制转换之前加一个正号 *(+)* 。
- *f.***noshowpos** 撤销 *showpos* 的设定。
- *f.***default** 将之前的设定 *(fmtflags)* 全改为默认精度。

其中 *f* 是输出流符号的，例如 *cout*。 特别注意，除了第一个，剩下所有的都操作都返回到流 *f*，所以它们可以像下面这样连接在一起

```
cout.scientific.showpos << 3 << endl;
```

4.12.1 脚本参数

Freefem++ 中定义了一个非常有用的数组*ARGV*，包含了命令行中所用到的所有脚本参数。 下面的代码写出了参数中的前三个：

```
                                                        //     版本 3.8-1
for(int i=0;i<ARGV.n;++i)
  {
    cout << ARGV[i] << endl;
  }
```

为了得到 getARGV.idp 中没有使用过的参数，导入脚本文件，

```
getARGV(n,defaultvalue)       //     得到没有使用过的第n个参数，如果它存在(n = 1, ...)
getARGV(after,defaultvalue)                //     得到字符串的自变量（arg），如果它存在
```

缺省值的类型可以是 int, real, string,

4.13 预处理程序

预处理程序有包含源文件及所定义的宏的相关操作指令 (include "script-name.idp")，宏定义。 宏指令分为两种，一种类似于对象，一种类似于函数。类似对象的宏指令没有参数，而类似函数的有参数。 声明标示符作为宏指令的语法如下：

```
macro <identifier>()  <replacement token list>          //   EOM // 停止宏的注释
macro <identifier>(<parameter list>) <replacement token list>          //   EOM
```

没有参数的宏指令的例子

```
macro xxx() {real i=0;int j=0;cout << i << " " << j << endl;}            //
xxx /* replace xxx by the <replacement token list> */
```

这是 freefem++ 的程序代码:

```
1 : // macro without parameter
2 :  macro xxx {real i=0;int j=0;cout << i << " " << j << endl;}//
3 :
4 :              {real i=0;int j=0;cout << i << " " << j << endl;}
```

宏命令参数的例子

```
macro toto(i) i                                                          //
                                                                //     引用参数，移动 {}
toto({real i=0;int j=0;cout << i << " " << j << endl;})
                                                                //     只有一层 {} 被移动
toto({{real i=0;int j=0;cout << i << " " << j << endl;}})
```

建立 freefem++ 代码 :

```
 6 :  macro toto(i )   i //
 8 : // 引用参数，移动 \{\}
 9 :                real i=0;int j=0;cout << i << " " << j << endl;
10 : // 只有一层 \{\} 被移动
11 :                {real i=0;int j=0;cout << i << " " << j << endl;}
```

使用宏作为宏 的参数，来把整个矩阵转换成数组，如下：

```
real[int,int] CC(7,7),EE(6,3),EEps(4,4);

macro VIL6(v,i) [ v(1,i), v(2,i),v(4,i), v(5,i),v(6,i) ]                    //    EOM
macro VIL3(v,i) [ v(1,i), v(2,i) ]                                          //    EOM
                                                          //    把v作为一个数组元素：
macro VV6(v,vv) [ v(vv,1), v(vv,2),
 v(vv,4), v(vv,5), v(vv,6) ]                                                //    EOM
macro VV3(v,vv) [ v(vv,1), v(vv,2) ]                                        //    EOM
                                                  //    所以用正式的矩阵来产生问题：
func C5x5 = VV6(VIL6,CC);
func E5x2 = VV6(VIL3,EE);
func Eps =  VV3(VIL3,EEps);
```

freefem++ 代码如下：

```
16 : real[int,int] CC(7,7),EE(6,3),EEps(4,4);
17 :
18 :   macro VIL6(v,i )   [ v(1,i), v(2,i),v(4,i), v(5,i),v(6,i) ]  //
19 :   macro VIL3(v,i )   [ v(1,i), v(2,i) ]  //  EOM
20 :  // apply v on array element :
21 :   macro VV6(v,vv )   [ v(vv,1), v(vv,2),
22 :   v(vv,4), v(vv,5), v(vv,6) ]  //  EOM
23 :   macro VV3(v,vv )   [ v(vv,1), v(vv,2) ]  //  EOM
24 : // so formal matrix to build problem..
25 :  func C5x5 =
 1 :         [         [ CC(1,1), CC(2,1),CC(4,1), CC(5,1),CC(6,1) ]
               [ CC(1,2), CC(2,2),CC(4,2), CC(5,2),CC(6,2) ]  ,
 1 :           [ CC(1,4), CC(2,4),CC(4,4), CC(5,4),CC(6,4) ]  ,
                [ CC(1,5), CC(2,5),CC(4,5), CC(5,5),CC(6,5) ]  ,
                     [ CC(1,6), CC(2,6),CC(4,6), CC(5,6),CC(6,6) ]
26 :  func E5x2 =
 1 :         [        [ EE(1,1), EE(2,1) ]  ,         [ EE(1,2), EE(2,2
 1 :          [ EE(1,4), EE(2,4) ]  ,         [ EE(1,5), EE(2,5) ]  ,
               [ EE(1,6), EE(2,6) ]   ]  ;
27 :  func Eps =         [        [ EEps(1,1), EEps(2,1) ]  ,
        [ EEps(1,2), EEps(2,2) ]   ]  ;
28 :
```

最后，符号# 用来把参数连接起来： 构造一个向量运算，如下：

```
macro div(u) (dx(u#1)+ dy(u#2))                                             //    EOM
mesh Th=square(2,2); fespace Vh(Th,P1);
Vh v1=x,v2=y;
```

```
cout << int2d(Th)(div(v)) << endl;
```

freefem++代码如下：

```
   31 :  macro div(u )    (dx(u#1)+ dy(u#2)) //EOM
   32 : mesh Th=square(2,2); fespace Vh(Th,P1);
   33 : Vh v1=x,v2=y;
   34 : cout << int2d(Th)(     (dx(v1)+ dy(v2)) ) << endl;
```

完成证明引用的测试：

```
macro foo(i,j,k) i j k                                          //   EOM
foo(,,)                                                         //   空行
foo( {int [}, {int] a(10},{};})
```

结果:

```
   36 :  macro foo(i,j,k )    i j k//EOM
   37 :        //   空行
   38 :          int [ int] a(10 );
```

为了在宏命令macro 中定义宏 macro，可以使用两个新的关键词 NewMacro 、 EndMacro 来新建和关闭指定的宏定义。 (版本 3.11，未完全完成测试).

4.14 异常处理

在 FreeFem++ 的版本2.3 中，异常情况的处理和C++中是一样的。 但是现在，只有C++ 中的异常被解决了。注意到，在C++ 中，所有引起 ExecError, assert, exit, ... 的错误也都叫做异常情况，但这类异常可能很难找到原因。 异常处理所有的 ExecError:

Example 4.9 一个简单的例子，捕获用零做分母：

```
real a;
try {
  a=1./0.;
}
catch  (...)                                  //    在2.3之后的版本所有的例外都能够被抓出来
{
  cout << " Catch an ExecError " << endl;
  a =0;
}
```

输出为：

```
1/0 : d d d
  current line = 3
Exec error :  Div by 0
   -- number :1
Try:: catch (...) exception
Catch an ExecError
```

Example 4.10 ：关于一个不可逆矩阵的例子：

```
int nn=5        ;
mesh Th=square(nn,nn);
verbosity=5;
fespace Vh(Th,P1);                                          //    P1 FE 空间
Vh uh,vh;                                                   //   未知的测试函数
func f=1;                                                    //    右手边函数
func g=0;                                                   //    边界条件函数
real   cpu=clock();
problem laplace(uh,vh,solver=Cholesky,tolpivot=1e-6) =                    //
问题的定义
            int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )             //    双线性
          + int2d(Th)( -f*vh )                                      //    线性
  ;

try {
  cout << " Try Cholesky \n";
  laplace;                                                        //    解决问题
  plot(uh);                                                       //    显示结果
  cout << "-- lap Cholesky " << nn << "x" << nn << "  : " <<  -cpu+clock()
       << " s,  max =" << uh[].max << endl;
}
catch(...) {                                                     //    抓出所有的异常
  cout << " Catch cholesky PB " << endl;
}
```

输出为

```
 -- square mesh : nb vertices  =36 ,  nb triangles = 50 ...
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 20, neb = 20
    Nb Mortars 0
 number of real boundary edges 20
    Number of Edges                  = 85
    Number of Boundary Edges         = 20 neb = 20
    Number of Mortars  Edges         = 0
    Nb Of Mortars with Paper Def     = 0 Nb Of Mortars = 0 ...
 Nb Of Nodes = 36
 Nb of DF = 36
 Try Cholesky
   -- Change of Mesh 0  0x312e9e8
   Problem(): initmat 1 VF (discontinuous Galerkin) = 0
  -- SizeOfSkyline =210
   -- size of Matrix 196 Bytes skyline =1
  -- discontinous Galerkin  =0 size of Mat =196 Bytes
  --  int  in   Optimized = 1,    ...
 all
  -- boundary int   Optimized = 1,  all
ERREUR choleskypivot (35)= -1.23124e-13 < 1e-06
  current line = 28
Exec error : FATAL ERREUR dans ../femlib/MatriceCreuse_tpl.hpp
```

```
cholesky line:
   -- number :545
 catch an erreur in  solve  =>  set  sol = 0 !!!!!!!
Try:: catch (...) exception
 Catch cholesky PB
```

第 5 章

网格生成

5.1 网格生成的命令

我们从两个关键词开始讲起： **border** 和 **buildmesh**。
本节所有例子均来自于 `mesh.edp`文件 和 `tablefunction.edp`文件。

5.1.1 方形（Square）

命令“**square**” 把单位正方形分成三角形 如下

```
mesh Th = square(4,5);
```

在$[0,1]^2$的单位正方形中产生了 4×5 的格子。 边界的分类见图 Fig. 5.1。 为了在一个

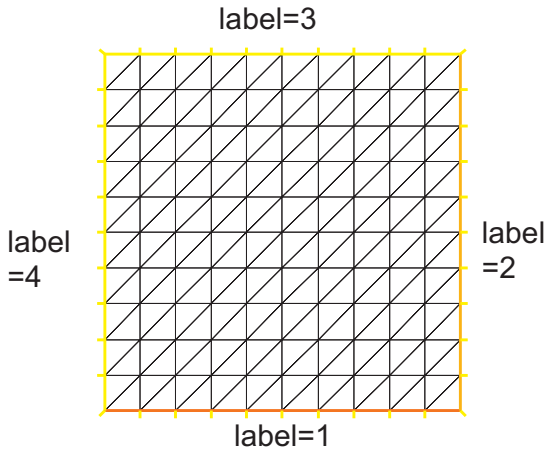


图 5.1: `square(10,10)`的网格的边界分类

$[x_0, x_1] \times [y_0, y_1]$的矩形中构造$n \times m$的格子，可编写如下代码：

```
real x0=1.2,x1=1.8;
real y0=0,y1=1;
int n=5,m=20;
mesh Th=square(n,m,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
```

Note 5.1 可增加参数 *flags=icase*, *icase* 为以下数值:

0 生成网格，其中所有的四边形都被对角线$x-y=cte$分割。

1 生成*Union Jack flag*型网格。

2 生成网格，其中所有的四边形都被对角线 $x+y=cte$ 分割*(v 3.8)*。

3 与取*0*的情况一样，除了在两个角处没有三角形三个顶点都在边界上 *(v 3.8)*。

4 与取*2*的情况一样，除了在两个角处没有三角形三个顶点都在边界上 *(v 3.8)*。

```
mesh Th=square(n,m,[x0+(x1-x0)*x,y0+(y1-y0)*y],flags=icase);
```

添加参数*label=labs* 将*4*个默认标签数改变为*labs[i-1]*, 例如 *int[int] labs = [11,12,13,14]*,
添加参数*region=10* 将区域数改变为 10 *(v 3.8)*。
为了观察这些标记, 可以查看文件*examples++/square-mesh.edp* :

```
for (int i=0;i<5;++i)
  {
    int[int] labs=[11,12,13,14];
    mesh Th=square(3,3,flags=i,label=labs,region=10);
    plot(Th,wait=1,cmm=" square flags = "+i );
   }
```

5.1.2 边界

边界被定义为分段参数曲线, 每段曲线只在端点处相交, 但也可以交于多于两点。边界跟一个区域接触的情况可用于将网络结构化，分割大区域, 创造小区域:

```
int upper = 1;
int others = 2;
int inner = 3;

border C01(t=0,1){x = 0;          y = -1+t;         label = upper;}
border C02(t=0,1){x = 1.5-1.5*t;  y = -1;           label = upper;}
border C03(t=0,1){x = 1.5;        y = -t;           label = upper;}
border C04(t=0,1){x = 1+0.5*t;    y = 0;            label = others;}
border C05(t=0,1){x = 0.5+0.5*t;  y = 0;            label = others;}
border C06(t=0,1){x = 0.5*t;      y = 0;            label = others;}
border C11(t=0,1){x = 0.5;        y = -0.5*t;       label = inner;}
border C12(t=0,1){x = 0.5+0.5*t;  y = -0.5;         label = inner;}
border C13(t=0,1){x = 1;          y = -0.5+0.5*t;   label = inner;}

int n = 10;
plot(C01(-n)+C02(-n)+C03(-n)+C04(-n)+C05(-n)+C06(-n)+
      C11(n)+C12(n)+C13(n), wait=true);
```

```
mesh Th = buildmesh(C01(-n)+C02(-n)+C03(-n)+C04(-n)+C05(-n)+C06(-n)+
     C11(n)+C12(n)+C13(n));

plot(Th, wait=true);                                          //    图 5.3

cout << "Part 1 has region number " << Th(0.75, -0.25).region << endl;
cout << "Part 2 has redion number " << Th(0.25, -0.25).region << endl;
```

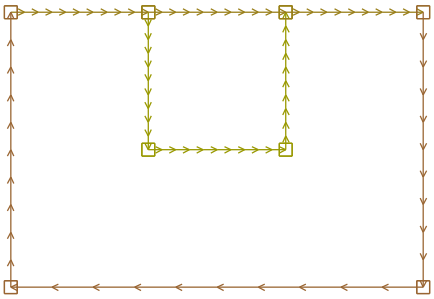


图 5.2: Multiple border ends intersect

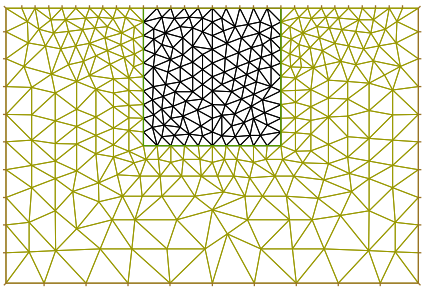


图 5.3: Generated mesh

关键词Triangulation（三角剖分）假设了区域定义在由参数方程确定的参数边界的左侧（相应地，右侧）。

$$\Gamma_j = \{(x,y) \mid x = \varphi_x(t),\ y = \varphi_y(t),\ a_j \le t \le b_j\}$$

为了检查边界定向 $t \mapsto (\varphi_x(t), \varphi_y(t))$, $t_0 \le t \le t_1$. 如果是如图5.4中那样，那么区域就在（线一侧的）阴影区域，否则则在线相反的一侧

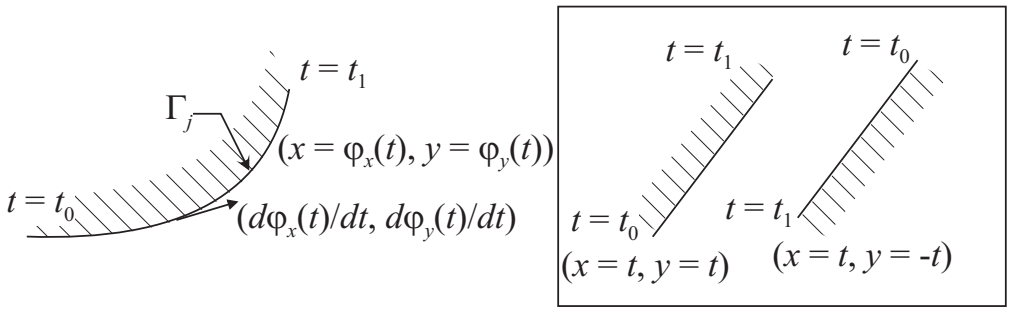


图 5.4: 由 $(\phi_x(t), \phi_y(t))$ 定义的边界取向

一般定义三角剖分的方法是：

```
mesh  Mesh_Name = buildmesh(Γ1(m1) + ··· + ΓJ(mj) OptionalParameter);
```

其中m_j是用来表示在Γ_j, $\Gamma = \cup_{j=1}^{J}\Gamma_J$上有多少顶点的或正或负的数，可选参数为（用逗号隔开）：

nbvx=<int value> , 设定网格中最多多少个顶点。

fixeborder=<bool value> , 设定网格产生器能否改变边界网格（默认可以改变；要当心周期性边界条件（例如. 6），可能会出问题）。

可以通过改变m_j的符号改变边界的方向。下面的例子说明如何改变边界定向，该例子产生一个有个小圆洞的单位圆盘，将单位圆盘编号为“1”（其中的圆环编为“2”），边界指标必须非零，不过也可以忽略。

```
1: border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
2: border b(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t);label=2;}
3: plot(a(50)+b(+30)) ;                                    //    看一下边界网格的图
4: mesh Thwithouthole= buildmesh(a(50)+b(+30));
5: mesh Thwithhole   = buildmesh(a(50)+b(-30));
6: plot(Thwithouthole,wait=1,ps="Thwithouthole.eps");          //    图 5.5
7: plot(Thwithhole,wait=1,ps="Thwithhole.eps");                //    图 5.6
```

Note 5.2 注意第五行的“*b(-30)*”改变了定向，第六，七行的*ps="fileName"*用以产生不同的图像脚本文件，如下图所示：

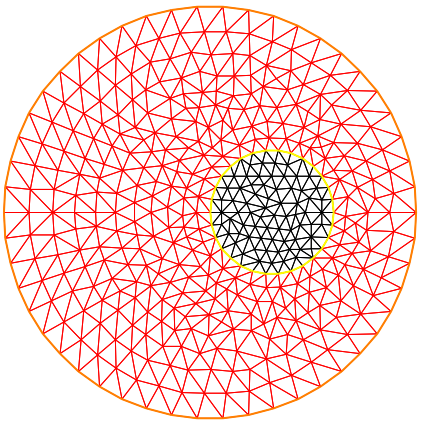


图 5.5: 不包含洞的网格

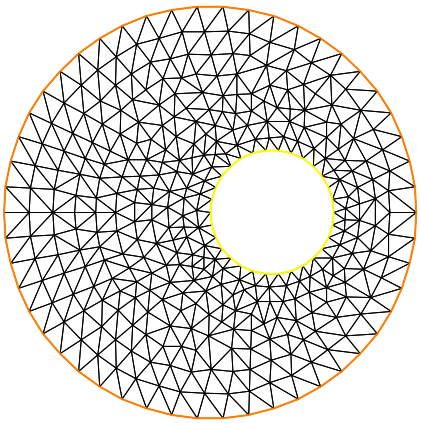


图 5.6: 考虑圆洞的网格

Note 5.3 边界的数据只有在 *plot*或 *buildmesh*被调用时才会计算。因此对于全局变量，即这里的r，虽然在两个边界定义之间值作了改变，然而，下面这段代码会出错，因为第一个边界的r也会被当作*0.3*计算。

```
real r=1;    border a(t=0,2*pi){ x=r*cos(t); y=r*sin(t);label=1;}
r=0.3    ;   border b(t=0,2*pi){ x=r*cos(t); y=r*sin(t);label=1;}
mesh Thwithhole   = buildmesh(a(50)+b(-30));          //    错误(a trap) 由于
                                                      //    两个圆有一样的半径 = 0.3
```

5.1.3 多重边界

（版本 3.30） 有时候边界数组会很有用处，然而不幸的是FreeFem++ 并不支持。我们视图迂回一下，采用另外的方式：当离散段数n是一个数组，我们采用循环的方式来给定每个数组的值，循环指标i跟随在参数定义之后，例如：border a(t=0,2*pi;i)....

一个简单的例子是：

```
1: border a(t=0,2*pi;i){ x=(i+1)*cos(t); y=(i+1)*sin(t);label=1;}
2: int[int] nn=[10,20,30];
3: plot(a(nn));                                    //    分别用10,20,30个点画出三个圆 ..
```

一个更复杂的例子是（来自mesh.edp中的例子）利用小圆圈定义一个矩形：

```
                                                   //    多重网格 语法 （2014年4月 3.30版本）
real[int] xx=[0,1,1,0],
yy=[0,0,1,1];
                                                                  //    半径，4个圆心
real[int] RC=[ 0.1, 0.05, 0.05, 0.1],
XC= [0.2,0.8,0.2,0.8],
YC= [0.2,0.8,0.8,0.2];
int[int] NC=[-10,-11,-12,13];                                      //    列出区段数
                                                                   //    4个圆形边界
border bb(t=0,1;i)
{
                                                          //    i 是多重边界的循环指标
int ii = (i+1)%4; real t1 = 1-t;
x = xx[i]*t1 + xx[ii]*t;
y = yy[i]*t1 + yy[ii]*t;
label = 0; ;
}
border cc(t=0,2*pi;i)
{
x = RC[i]*cos(t)+XC[i];
y = RC[i]*sin(t)+YC[i];
label = i+1;
}
int[int] nn=[4,4,5,7];                    //    4条边界 , 分别对应的区段数为4,4,5,7。
plot(bb(nn),cc(NC),wait=1);
mesh th= buildmesh(bb(nn)+cc(NC)) ;
plot(th,wait=1);
```

5.1.4 数据结构与网格的读写命令

如果用户想要读任意位置的三角网格的信息，需要参看下面这些文件：

```
border C(t=0,2*pi) { x=cos(t); y=sin(t); }
mesh Th = buildmesh(C(10));
savemesh("mesh_sample.msh");
```

网格表示在图 5.7中。

Th 的信息存在“mesh_sample.msh”，其结构在表 5.1中有说明。
其中n_v表示顶点数，n_t表示三角形数，n_s边界上的边数。
对每一个顶点q^i, $i = 1, \cdots, n_v$，用(q_x^i, q_y^i)表示x坐标和y 坐标。
每个三角元$T_k, k = 1, \cdots, 10$有3个顶点q^{k_1}, q^{k_2}, q^{k_3}，按照逆时针方向排列。边界由十条线L_i, $i = 1, \cdots, 10$组成，它们的端点为q^{i_1}, q^{i_2}。

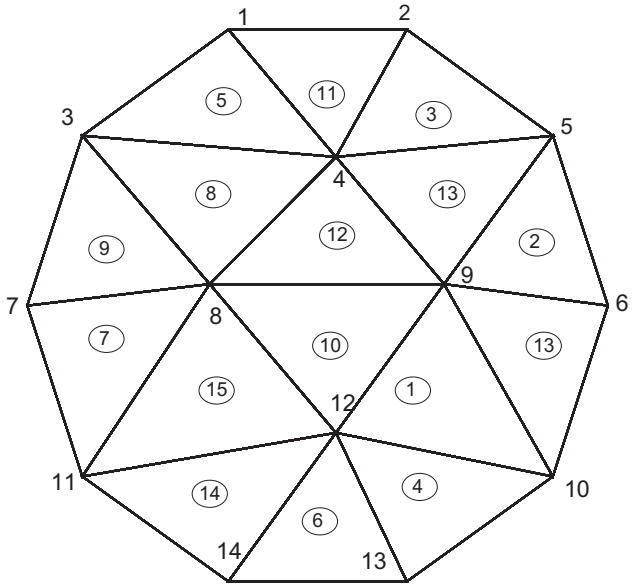


图 5.7: buildmesh(C(10))生成的网格

在该图中，我们有：

$n_v = 14$, $n_t = 16$, $n_s = 10$

$q^1 = (-0.309016994375, 0.951056516295)$
$\vdots \quad \vdots \quad \vdots$
$q^{14} = (-0.309016994375, -0.951056516295)$

T_1 are q^9, q^{12}, q^{10}的顶点。
$\vdots \quad \vdots \quad \vdots$
T_{16} 的顶点是 q^9, q^{10}, q^6。

L_1 第一个边的端点是 q^6, q^5。
$\vdots \quad \vdots \quad \vdots$
L_{10} 第十个边的端点是 q^{10}, q^6。

文件内容	解释
14 16 10	n_v n_t n_e
-0.309016994375 0.951056516295 1	q_x^1 q_y^1 boundary label=1
0.309016994375 0.951056516295 1	q_x^2 q_y^2 boundary label=1
... ... ⋮	
-0.309016994375 -0.951056516295 1	q_x^{14} q_y^{14} boundary label=1
9 12 10 0	1_1 1_2 1_3 region label=0
5 9 6 0	2_1 2_2 2_3 region label=0
...	
9 10 6 0	16_1 16_2 16_3 region label=0
6 5 1	1_1 1_2 boundary label=1
5 2 1	2_1 2_2 boundary label=1
...	
10 6 1	10_1 10_2 boundary label=1

Table 5.1: “mesh_sample.msh”的结构

在 FreeFem++ 中有多种网格文件的格式，可以与其他工具，如emc2, modulef.. （见第12章）等联用。 文件的扩展名表明了其格式，在F. Hecht的文章”bamg : a bidimentional anisotropic mesh generator”（可以在FreeFEM网站下载）中，可以进一步了解.msh格式文件的信息储存方式。

FreeFem++ 可以读入网格文件，只是边界的名字没有保存，文件只保存了边界参数。因此这些边界必须与参数联系起来，这些参数决定了边界在程序中的次序，除非用关键词“label”覆盖掉它们。以下为一些例子

```
border floor(t=0,1){ x=t; y=0; label=1;};                    //    单位正方形
border right(t=0,1){ x=1; y=t; label=5;};
border ceiling(t=1,0){ x=t; y=1; label=5;};
border left(t=1,0){ x=0; y=t; label=5;};
int n=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh(th,"toto.am_fmt");                    //    "formatted Marrocco" 格式
savemesh(th,"toto.Th");                              //    "bamg"-型  网格
savemesh(th,"toto.msh");                               //    freefem 格式
savemesh(th,"toto.nopo");                        //    modulef 格式  见 [10]
mesh th2 = readmesh("toto.msh");                              //    读取网格
```

Example 5.1 (Readmesh.edp)

```
border floor(t=0,1){ x=t; y=0; label=1;}; //    单位方形
border right(t=0,1){ x=1; y=t; label=5;};
border ceiling(t=1,0){ x=t; y=1; label=5;};
border left(t=1,0){ x=0; y=t; label=5;};
int n=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh(th,"toto.am_fmt");                    //    "formated Marrocco"格式
savemesh(th,"toto.Th");                        //    数据库格式 db "bamg"网格
savemesh(th,"toto.msh");                             //    freefem 格式
savemesh(th,"toto.nopo");                      //    modulef 格式  参见 [10]
mesh th2 = readmesh("toto.msh");
fespace femp1(th,P1);
femp1 f = sin(x)*cos(y),g;
{                                                              //    储存解
ofstream file("f.txt");
file << f[] << endl;
}                                                    //    关闭文件指针 (end block)
{                                                               //    读入
ifstream file("f.txt");
file >> g[] ;
}                                                    //    关闭输入文件 (end block)
fespace Vh2(th2,P1);
Vh2 u,v;
plot(g);
//    find u such that
//    u + Δu = g in Ω ,
//    u = 0 on Γ1 and ∂u/∂n = g on Γ2
solve pb(u,v) =
    int2d(th)( u*v - dx(u)*dx(v)-dy(u)*dy(v) )
  + int2d(th)(-g*v)
  + int1d(th,5)( g*v)                                          //    ∂u/∂n = g on Γ2
  + on(1,u=0) ;
plot (th2,u);
```

5.1.5 网格的连接

下面的例子展示了获取网格信息的方法

```
{                                                          //    获取网格信息 （版本 1.37）
  mesh Th=square(2,2);
                                                                 //    获取网格的数据
  int nbtriangles=Th.nt;
  cout << " nb of Triangles = " << nbtriangles << endl;
  for (int i=0;i<nbtriangles;i++)
    for (int j=0; j <3; j++)
      cout << i << " " << j << " Th[i][j] = "
           << Th[i][j] << "  x = "<< Th[i][j].x  << " , y= "<< Th[i][j].y
           << ",  label=" << Th[i][j].label << endl;

//      Th(i) 返回 Th                                                    的顶点i
//      Th[k] 返回 Th                                                   的三角形k

  fespace femp1(Th,P1);
  femp1 Thx=x,Thy=y;                                          //    获取顶点坐标的程序
                                                                 //    获取顶点信息 :
  int nbvertices=Th.nv;
  cout << " nb of vertices = " << nbvertices << endl;
  for (int i=0;i<nbvertices;i++)
       cout << "Th(" <<i  << ") : "                                  //    << endl;
            << Th(i).x << " " << Th(i).y  << " " << Th(i).label    //   v 2.19
            << "       old method: " << Thx[][i] << " " << Thy[][i] << endl;

                                                        //    获取点 (0.55,0.6) 信息的方法

  int it00 = Th(0.55,0.6).nuTriangle;                          //    接着是三角形个数
  int nr00 = Th(0.55,0.6).region;                                    //

                                                                 //    一个三角形的信息
  real area00 = Th[it00].area;                                //    在版本2.19中更新
  real nrr00 = Th[it00].region;                               //    在版本2.19中更新
  real nll00 = Th[it00].label;                              //    在此情况下与区域相同


//   获取包含点x,y 的三角形
//    或区域数 （原来的方法）
//    ------------------------------------------------------
  fespace femp0(Th,P0);
  femp0 nuT;                                                  //    一个P0元的三角形的标号
    for (int i=0;i<Th.nt;i++)
     nuT[][i]=i;
  femp0 nuReg=region;                                        //    一个P0元的三角形的区域数
                                                                    //    要求
  int it0=nuT(0.55,0.6);                                //    包含 (0.55,0,6) 的Th的三角形数目;
  int nr0=nuReg(0.55,0.6);                                //    包含 (0.55,0,6) 的Th的区域数目;


 //    转储
 //    ------------------------------------------------------

  cout << "  point (0.55,0,6) :triangle number " << it00 << " " << it00
```

```
       << ", region = " << nr0 << " == " << nr00 << ",  area K " << area00 << endl;

                                                          //     获取边界信息与相邻网格的新方法

 int k=0,l=1,e=1;
  Th.nbe ; //     返回边界单元的个数
  Th.be(k);    //     返回边界单元 k ∈{0,...,Th.nbe − 1}
  Th.be(k)[l];    //     返回边界单元 k 的顶点  l ∈{0,1}
  Th.be(k).Element ;    //     返回包含边界单元 k 的三角形
  Th.be(k).whoinElement ;    //     返回包含边界单元k的三角形的边编号
  Th[k].adj(e) ;                //     返回k 关于边e的相邻三角形, 并将
                                                    //     e的值改变为在相邻三角形的对应的边
  Th[k] == Th[k].adj(e)                             //     没有相邻三角形返回相同的值
  Th[k] != Th[k].adj(e)                                          //     真相邻三角形

  cout << " print mesh connectivity " << endl;
  int nbelement = Th.nt;
  for (int k=0;k<nbelement;++k)
    cout << k << " :  " << int(Th[k][0]) << " " << int(Th[k][1])
         << " " <<  int(Th[k][2])
         << " , label  " << Th[k].label << endl;
                                                                                  //

  for (int k=0;k<nbelement;++k)
    for (int e=0,ee;e<3;++e)
                                //     FH 程序的注:  将 ee 设为 e, 用adj方法将ee改变,
                                                          //     在 () 中与参数不同.
    cout << k <<  " " << e << " <=>  " << int(Th[k].adj((ee=e))) << " " << ee
     << "  adj: " << ( Th[k].adj((ee=e)) != Th[k]) << endl;
                          //     注释 :  如果 k == int(Th[k].adj(ee=e)) 不是相邻单元


  int nbboundaryelement = Th.nbe;

  for (int k=0;k<nbboundaryelement;++k)
      cout << k << " : " <<  Th.be(k)[0] << " " << Th.be(k)[1] << " , label "
           << Th.be(k).label <<  " tria  " << int(Th.be(k).Element)
           << " " << Th.be(k).whoinElement <<  endl;

}
```

输出结果是:

```
 -- square mesh : nb vertices  =9 ,  nb triangles = 8 ,  nb boundary edges 8
    Nb of Vertices 9 ,  Nb of Triangles 8
    Nb of edge on user boundary  8 ,  Nb of edges on true boundary  8
 number of real boundary edges 8
 nb of Triangles = 8
0 0 Th[i][j] = 0  x = 0 , y= 0,  label=4
0 1 Th[i][j] = 1  x = 0.5 , y= 0,  label=1
0 2 Th[i][j] = 4  x = 0.5 , y= 0.5,  label=0
...
6 0 Th[i][j] = 4  x = 0.5 , y= 0.5,  label=0
6 1 Th[i][j] = 5  x = 1 , y= 0.5,  label=2
6 2 Th[i][j] = 8  x = 1 , y= 1,  label=3
7 0 Th[i][j] = 4  x = 0.5 , y= 0.5,  label=0
```

```
7 1 Th[i][j] = 8  x = 1 , y= 1,  label=3
7 2 Th[i][j] = 7  x = 0.5 , y= 1,  label=3
 Nb Of Nodes = 9
 Nb of DF = 9
 -- vector function's bound  0 1
 -- vector function's bound  0 1
 nb of vertices = 9
Th(0) : 0 0 4       old method: 0 0
Th(1) : 0.5 0 1       old method: 0.5 0
...
Th(7) : 0.5 1 3       old method: 0.5 1
Th(8) : 1 1 3       old method: 1 1
 Nb Of Nodes = 8
 Nb of DF = 8

 print mesh connectivity
0 :  0 1 4 , label  0
1 :  0 4 3 , label  0
...
6 :  4 5 8 , label  0
7 :  4 8 7 , label  0
0 0 <=>  3 1  adj: 1
0 1 <=>  1 2  adj: 1
0 2 <=>  0 2  adj: 0
...
6 2 <=>  3 0  adj: 1
7 0 <=>  7 0  adj: 0
7 1 <=>  4 0  adj: 1
7 2 <=>  6 1  adj: 1
0 : 0 1 , label 1 tria  0 2
1 : 1 2 , label 1 tria  2 2
...
6 : 0 3 , label 4 tria  1 1
7 : 3 6 , label 4 tria  5 1
```

5.1.6 关键词“triangulate”（三角剖分）

FreeFem++ 能够对点集建立三角刨分。这个三角刨分由Delaunay算法建立基于点集的二维凸包的网格。当要从一个表列函数建立网格时，会用到这个功能。
在文件中，表列函数的点坐标和值分别定义为形如 `x y f(x,y)` 的行：

```
0.51387 0.175741 0.636237
0.308652 0.534534 0.746765
0.947628 0.171736 0.899823
0.702231 0.226431 0.800819
0.494773 0.12472 0.580623
0.0838988 0.389647 0.456045
...............
```

每行的第三个数由 `triangulate` 设定为不可读，但你可以用这第三个值定义一个每行形如 `x y f(x,y)` 的表列函数。
下面两个例子展示了如何从文件 “xyf” 出发，按以上所说格式作一个网格。 命令 `triangulate` 只能在前两行用。

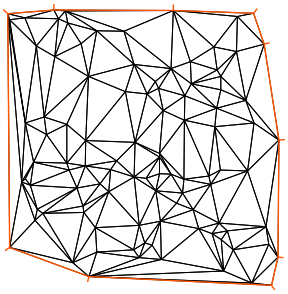


图 5.8: 文件xyf中点集生成的二维凸包的Delaunay网格

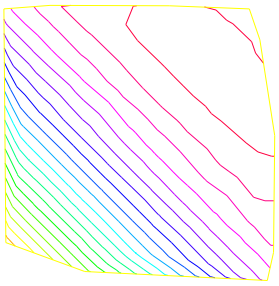


图 5.9: 表列函数的等值线

```
mesh Thxy=triangulate("xyf");                    //  建立凸包的 Delaunay 网格
                                                 //  文件的前两列定义点 xyf
plot(Thxy,ps="Thxyf.ps");                        //  (见图 5.8)

fespace Vhxy(Thxy,P1);                           //  建立 P1 插值
Vhxy fxy;                                        //  函数

                                                 //  读取第三行, 定义函数
{ ifstream file("xyf");
   real xx,yy;
   for(int i=0;i<fxy.n;i++)
   file >> xx >>yy >> fxy[][i];                  //  只读第三行
                                                 //  跳过 xx 与 yy
}
plot(fxy,ps="xyf.eps");                          //  画出函数 (见图 5.9)
```

一个新的建立网格的方法是用两个数组，一个是x值，一个是y值(version 2.23-2)

```
Vhxy xx=x,yy=y;                                  //  为x和y建立两个数组
mesh Th=triangulate(xx[],yy[]);
```

5.2 建立空网格作为边界有限元空间

为了定义一个边界上的有限元空间，我们设想一个没有内部点的网格（空网格）， 这个思路在用拉格朗日乘子法处理复杂的实际问题时很有用。
因此函数 emptymesh 除掉网格所有边界内的点，但不包括点在内边界上的点。

```
{                                                //  新的 2004 空网格 (版本 1.40)
                                                 //  -- 对建立Multiplicator空间有用
                                                 //  建立一个没有内点的网格
                                                 //  拥有相同的边界
                                                 //  -----
  assert(version>=1.40);
  border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
  mesh Th=buildmesh(a(20));
   Th=emptymesh(Th);
  plot(Th,wait=1,ps="emptymesh-1.eps");          //  见图 5.10
```

```
}
```

通过 emptymesh(Th,ssd) 用网格 Th 对边的设定集合，为假想的子区域建立空网格也是可能的， 如果根据两个相邻的三角形$e = t1 \cap t2$和$ssd[T1] \neq ssd[T2]$，其中 ssd 代表假想区域三角形的编号，那么一个（这两个相邻三角形公用的）边e 就在这个集合中，它们被储存在包含三角形尺寸和数量的一个 int[int] 数组中。

```
{  //    新的 2004 空网格 （版本 1.40）
 //     -- 对建立Multiplicator空间有用
 //     建立一个没有内点的网格
 //     在拟子区域中
 //     -----
  assert(version>=1.40);
  mesh Th=square(10,10);
  int[int] ssd(Th.nt);
  for(int i=0;i<ssd.n;i++)                                    //     建立拟区域编号
   {  int iq=i/2;                                          //     因为每个四边形有2个三角形
      int ix=iq%10;                                                //
      int iy=iq/10;                                                //
    ssd[i]= 1 + (ix>=5) +  (iy>=5)*2;
   }
  Th=emptymesh(Th,ssd);                                          //     建立空网格
                                                        //     有边 e = T1 ∩ T2 and ssd[T1] ≠ ssd[T2]
  plot(Th,wait=1,ps="emptymesh-2.eps");                          //     见图 5.11
  savemesh(Th,"emptymesh-2.msh");
}
```

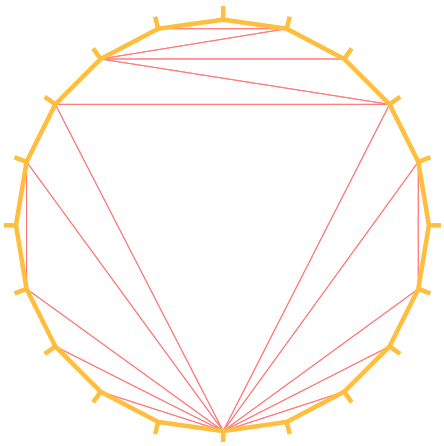


图 5.10: 边界的空网格

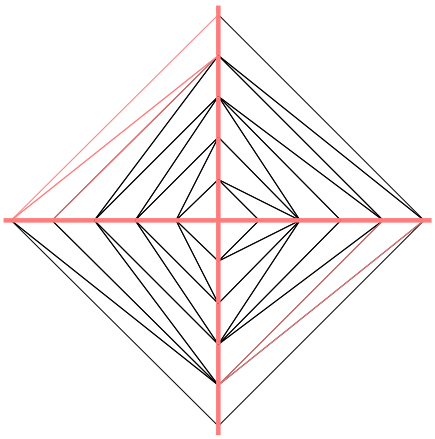


图 5.11: 一个由假想区域三角形标号定义的空网格

5.3 重新网格化

5.3.1 Movemesh

网格可以通过 movemesh 转置，旋转，分解;这在弹性力学中观察形状的位移$\mathbf{\Phi}(x,y) = (\Phi_1(x,y), \Phi_2(x,y))$造成的变形时很有用， 也可以用来处理自由边界问题或可变边界问题。
如果Ω是一个形如$T_h(\Omega)$的三角刨分，Φ是一个位移矢量，则$\Phi(T_h)$由该指令确定

```
mesh  Th=movemesh(Th,[Φ1,Φ2]);
```

有时变换后的网格是无效的，因为有些三角形突然反转了，即产生了负区域。为了及时发现该问题，在实际做任何变换前，必须用 checkmovemesh 检查网格中最小的三角形区域。

Example 5.2 $\Phi_1(x,y) = x + k * \sin(y * \pi)/10)$, $\Phi_2(x,y) = y + k * \cos(y\pi)/10)$, 当 $k > 1$ 很大时.

```
verbosity=4;
border a(t=0,1){x=t;y=0;label=1;};
border b(t=0,0.5){x=1;y=t;label=1;};
border c(t=0,0.5){x=1-t;y=0.5;label=1;};
border d(t=0.5,1){x=0.5;y=t;label=1;};
border e(t=0.5,1){x=1-t;y=1;label=1;};
border f(t=0,1){x=0;y=1-t;label=1;};
func uu= sin(y*pi)/10;
func vv= cos(x*pi)/10;

mesh Th = buildmesh ( a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
plot(Th,wait=1,fill=1,ps="Lshape.eps");                          //    见图 5.12
real coef=1;
real minT0= checkmovemesh(Th,[x,y]);                          //    最小的三角区域
while(1)                                                    //    找一个正确的移动区域
{
  real minT=checkmovemesh(Th,[x+coef*uu,y+coef*vv]);          //    最小的三角区域
  if (minT > minT0/5) break ;                                //    如果足够大
  coef/=1.5;
}

Th=movemesh(Th,[x+coef*uu,y+coef*vv]);
plot(Th,wait=1,fill=1,ps="movemesh.eps");                        //    见图 5.13
```

Note 5.4 考虑一个定义在网格*Th*上的函数u。像语句*Th=movemesh(Th...)*不改变u, 因此旧的网格依然存在。 当没有函数用到它时, 网格就会被破坏。 像$u = u$的语句在新的网格*Th*上通过插值重新定义了u, 因此如果u是唯一用到该网格的函数那么旧网格*Th* 会被破坏。

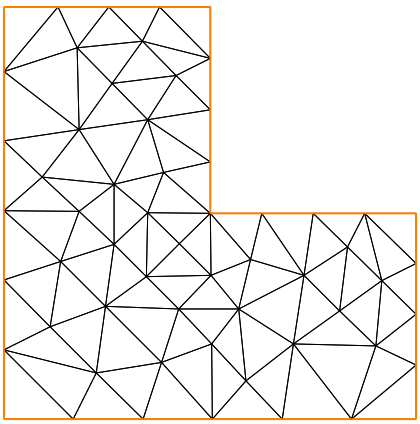


图 5.12: L-型

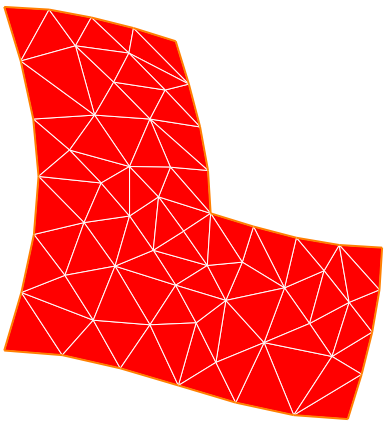


图 5.13: 移动了的 L-型

Example 5.3 (movemesh.edp) 现在, 我们给出一个移动网格的例子。 它是一个定义在移动网格上的 *lagrangian* 函数 u。

```
                                                        //      简单的移动网格的例子
mesh Th=square(10,10);
fespace Vh(Th,P1);
real t=0;
                                                                        //      ---
                                                    //      现在的问题是如何不用插值来建立数据
                                                      //      所以如你所见数据 u 随着网格移动
                                                                        //      ---
Vh u=y;
for (int i=0;i<4;i++)
{
 t=i*0.1;
 Vh f= x*t;
 real minarea=checkmovemesh(Th,[x,y+f]);
 if (minarea >0 )                                                       //      移动网格正常
   Th=movemesh(Th,[x,y+f]);

 cout << " Min area  " << minarea << endl;

 real[int] tmp(u[].n);
 tmp=u[];                                                                       //      保存值
 u=0;                                                           //      改变与u相关的FEspace和网格
 u[]=tmp;                                                       //      设置u的值, 不用网格的更新信息
 plot(Th,u,wait=1);
};
//      在这个程序中, 因为u只定义在最后一个网格, 因此
//      所有之前的网格都被删除了.
//      --------
```

5.4 正则三角剖分: hTriangle

对集合S, 我们定义S的直径

$$\mathrm{diam}(S) = \sup\{|\boldsymbol{x} - \boldsymbol{y}|;\ \boldsymbol{x}, \boldsymbol{y} \in S\}$$

区域Ω中的序列$\{\mathcal{T}_h\}_{h\downarrow 0}$ 被称作 正则, 如果他们满足以下条件:

1.
$$\lim_{h\downarrow 0} \max\{\mathrm{diam}(T_k)|\ T_k \in \mathcal{T}_h\} = 0$$

2. 存在与h无关的数$\sigma > 0$ 使得

$$\frac{\rho(T_k)}{\mathrm{diam}(T_k)} \geq \sigma \qquad \text{for all } T_k \in \mathcal{T}_h$$

其中 $\rho(T_k)$是T_k中内接圆的直径.

我们令$h(\mathcal{T}_h) = \max\{\mathrm{diam}(T_k)|\ T_k \in \mathcal{T}_h\}$, 并通过以下得到:

```
mesh Th = ......;
fespace Ph(Th,P0);
Ph h = hTriangle;
cout << "size of mesh = " << h[].max << endl;
```

5.5 自适应网格

函数

$$f(x,y) = 10.0x^3 + y^3 + \tan^{-1}[\varepsilon/(\sin(5.0y) - 2.0x)] \qquad \varepsilon = 0.0001$$

的值剧烈的变动, 并且由章节 5.1中的命令给出的初始网格 不能反应出该函数的剧烈变化。

Example 5.4

```
real eps =  0.0001;
real h=1;
real hmin=0.05;
func f = 10.0*x^3+y^3+h*atan2(eps,sin(5.0*y)-2.0*x);

mesh Th=square(5,5,[-1+2*x,-1+2*y]);
fespace Vh(Th,P1);
Vh fh=f;
plot(fh);
for (int i=0;i<2;i++)
 {
   Th=adaptmesh(Th,fh);
   fh=f;                                   //    旧网格被删除
   plot(Th,fh,wait=1);
 }
```

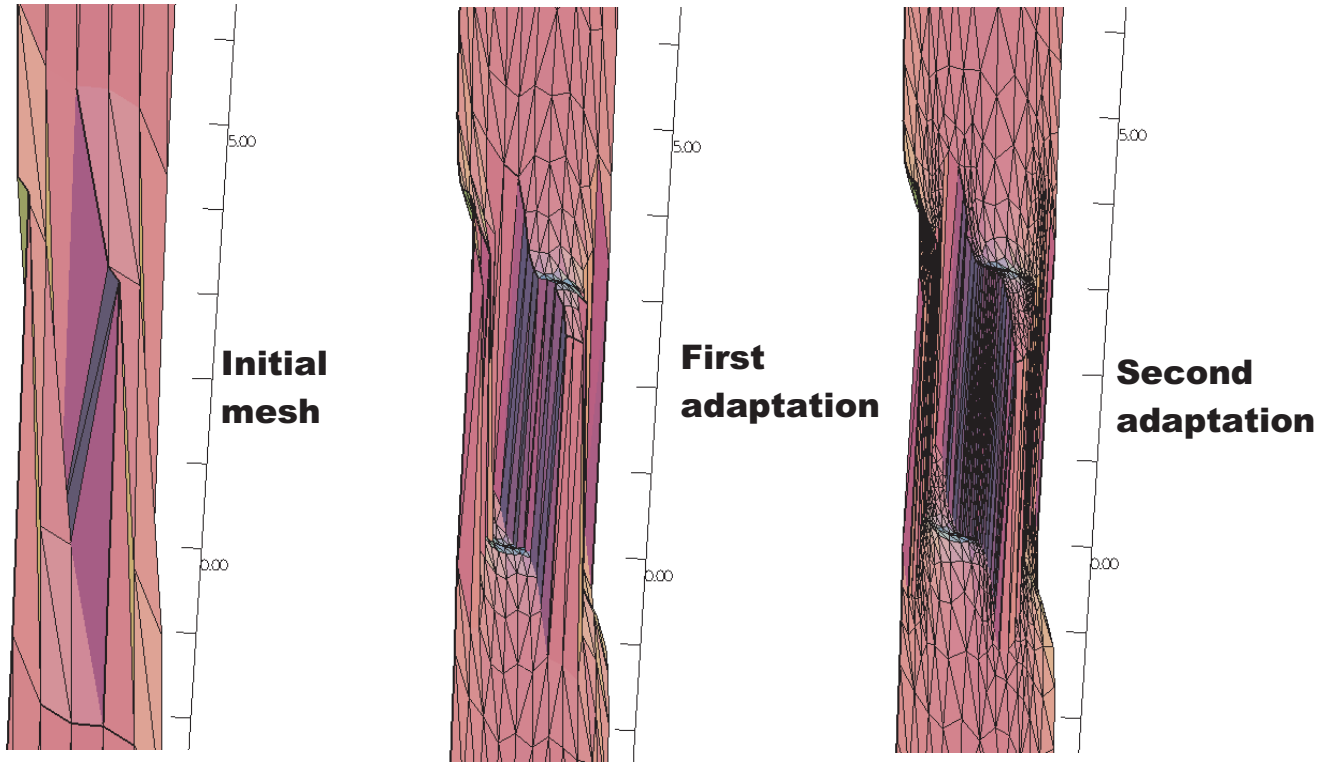


图 5.14: 初始网格及第一和第二自适应网格的3D图

FreeFem++ 使用可变度量/Delaunay 自动网格算法。 命令

```
mesh ATh = adaptmesh(Th, f);
```

建立了自适应于函数(公式或FE-function)

$$D^2 f = (\partial^2 f/\partial x^2, \partial^2 f/\partial x \partial y, \partial^2 f/\partial y^2)$$

的 Hessian 阵的新网格 ATh。 当问题局部剧烈变动时, 网格自适应是个非常强有力的工具。
这里我们解决问题 (2.1)-(2.2), 在 $f = 1$ 和 Ω 是 L-型 区域的情形下。

Example 5.5 (Adapt.edp) 解在两条线 bc 和 bd 焦点 γ 处有奇异性 $r^{3/2}, r = |x - \gamma|$ （见图*5.15*）。

```
border ba(t=0,1.0){x=t;   y=0;   label=1;};
border bb(t=0,0.5){x=1;   y=t;   label=1;};
border bc(t=0,0.5){x=1-t; y=0.5;label=1;};
border bd(t=0.5,1){x=0.5; y=t;   label=1;};
border be(t=0.5,1){x=1-t; y=1;   label=1;};
border bf(t=0.0,1){x=0;   y=1-t;label=1;};
mesh Th = buildmesh ( ba(6)+bb(4)+bc(4)+bd(4)+be(4)+bf(6) );
fespace Vh(Th,P1);                                          //    设置 FE 空间
Vh u,v;                                                  //    设置未知函数和测试函数
func f = 1;
real error=0.1;                                             //    错误等级
```

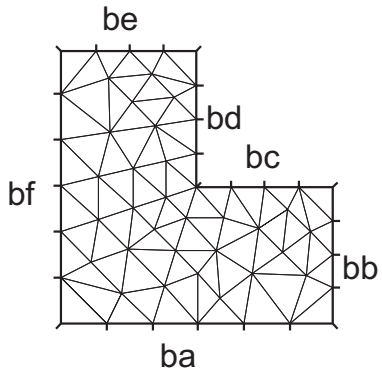


图 5.15: L-型区域和它边界的名字

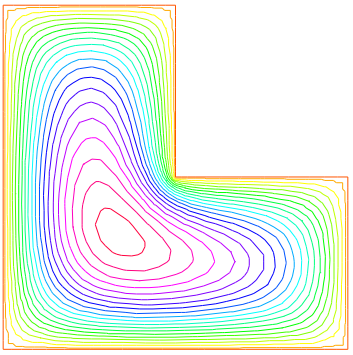


图 5.16: 通过4次自适应后的最终解

```
problem Poisson(u,v,solver=CG,eps=1.0e-6) =
    int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v))
  - int2d(Th) ( f*v )
  + on(1,u=0)  ;
for (int i=0;i< 4;i++)
{
  Poisson;
  Th=adaptmesh(Th,u,err=error);
  error = error/2;
} ;
plot(u);
```

为了加速自适应, `adaptmesh` 的默认参数 `err` 可随时被改变; 它确定了要求的精度并使得新网格更精细或者更粗糙。
问题是强的和对称的, 因此线性系统可以用共轭梯度法求解 （参数 `solver=CG`, 对参量做停机准则, 这里 `eps=1.0e-6`）。 通过 自适应网格, 最终解在 bc 和 bd 交点附近的斜率能够正确地计算, 如图5.16。
[9] 详细描述了该方法， 它有很多默认参数可以调整 :
Si `f1,f2` sont des functions et `thold`, `Thnew` des maillages.

```
Thnew = adaptmesh(Thold, f1  ...  );
Thnew = adaptmesh(Thold, f1,f2  ...  ]);
Thnew = adaptmesh(Thold, [f1,f2]  ...  );
```

自适应网格的附加参数没有写在这儿, 因此 "..."

hmin= 最小边的尺寸。 （`val` 是一个实数。 默认值与被网格化区域的大小及网格生成器的精度有关）。

hmax= 最大边的尺寸. （`val` 是一个实数。 默认值与被网格化区域的直径有关）。

err= P_1 插值误差级别 （默认 0.01 ）。

errg= 相对几何误差。默认值为 0.01, 并且恒低于 $1/\sqrt{2}$。 因为几何约束, 由该选项生成的网格可能有些边小于 `-hmin` 。

nbvx= 由网格生成器生成的最大顶点数 （(默认值 9000）。

nbsmooth= 光滑化过程的迭代次数 （默认值 5 ）。

nbjacoby= 在构造度量中的光滑化过程的迭代次数 , 0 意味着不进行光滑化 （默认值 6 ）。

ratio= 在度量上进行规定的光滑化的系数。如果值为 0 或小于 1.1, 不做任何光滑化 （默认值 1.8）。

如果 `ratio` $>$ 1.1, 网格大小变动的速度的界 被 $log($`ratio`$)$ 控制。 注意: 当 `ratio` 接近于 `1`, 生成的顶点的个数增加。 这对于控制shock或边界层附近加细区域的密度有帮助。

omega= 光滑化过程的松弛参数 （默认值 1.0）。

iso= 若为 true, 强制度量为各向同性 （默认值 false ）。

abserror= 若为 false, 使用相对误差的 equi-repartion 标准来计算度量 （默认值 false ）。 在这种情况下度量由

$$\mathcal{M} = \left(\frac{1}{\texttt{err coef}^2} \quad \frac{|\mathcal{H}|}{max(\texttt{CutOff}, |\eta|)} \right)^p \tag{5.1}$$

给出。 否则, 使用误差的 equi-repartion 标准来计算度量。 在这种情况下度量由

$$\mathcal{M} = \left(\frac{1}{\texttt{err coef}^2} \quad \frac{|\mathcal{H}|}{\sup(\eta) - \inf(\eta)} \right)^p \tag{5.2}$$

给出。

cutoff= 相对误差计算的下极限 （默认值 1.0e-6）。

verbosity= 报告信息级别 （从 0 到 ∞ 间选择）。 也改变全局变量的赘言的值 （已废弃）。

inquire= 询问网格的图像信息 （默认值 false）。

splitpbedge= 若为 true, 将所有的有两个边界顶点的内部边对半分 （默认值 true）。

maxsubdiv= 改变度量使得背景边最大细分的上界为 `val` （极限总是 10, 并且默认值为 10）。

rescaling= 若为 true, 对应网格自适应的函数重新调节至 0 和 1 之间 （默认值 true ）。

keepbackvertices= 若为 true, 试着使原网格中的顶点尽量保留 （默认值 true）。

isMetric= 若为 true, 度量被显式定义 （默认值 false）。 若给定 3 个函数 m_{11}, m_{12}, m_{22} , 它们直接定义一个对称矩阵域, 计算这些矩阵的 Hessian 阵来定义度量。 若只给定一个函数, 那么它体现各向同性网格在每点处的大小。

例如, 若给定偏导数 `fxx` $(= \partial^2 f/\partial x^2)$, `fxy` $(= \partial^2 f/\partial x \partial y)$, `fyy` $(= \partial^2 f/\partial y^2)$, 我们可以令

```
Th=adaptmesh(Th,fxx,fxy,fyy,IsMetric=1,nbvx=10000,hmin=hmin);
```

power= 用来计算度量的 Hessian 阵的指数幂 （默认值 1 ）。

thetamax= 最小的边角角度的度数 (默认值 10°), 比如边角是 ABC 那么角度即为向量 AB, BC的夹角, （0 意味着没有夹角, 90 意味着垂直 , ...）。

splitin2= 布尔值. 若为 true, 把所有最终网格的三角剖分成 4 个小三角形。

metric= 一个由 3 个实数列组成的数列, 用以设置获取度量数据信息. 这三个数列的长度必须为顶点的个数。 因此若 `m11,m12,m22` 是与自适应网格相关的三个 P1 有限元, 你可以写成: `metric=[m11[],m12[],m22[]]` （完整例子见文件 convect-apt.edp ）

nomeshgeneration= 若为 true, 不生成任何自适应网格 （用来只计算度量时有用）。

periodic= 写作 `periodic=[[4,y],[2,y],[1,x],[3,x]];` 建立自适应的周期网格. 样本生成一个方的双周期网格。 （见周期有限元空间 6, 完整例子见文件 `sphere.edp`）

我们使用命令 `adaptmesh` 来建立有固定网格大小的常规网格。 因此建立网格大小为 $\frac{1}{30}$ 的网格, 试以下代码:

Example 5.6 *uniformmesh.edp*

```
mesh Th=square(2,2);                                          //     建立初始网格
plot(Th,wait=1,ps="square-0.eps");
Th= adaptmesh(Th,1./3As writing
0.,IsMetric=1,nbvx=10000);                                              //
plot(Th,wait=1,ps="square-1.eps");
Th= adaptmesh(Th,1./30.,IsMetric=1,nbvx=10000);                    //     不止一次
Th= adaptmesh(Th,1./30.,IsMetric=1,nbvx=10000);  //     根据自适应边界  maxsubdiv=
plot(Th,wait=1,ps="square-2.eps");
```

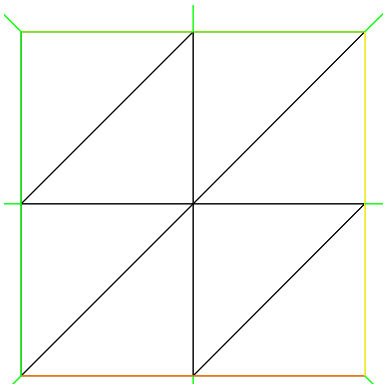


图 5.17: 初始网格

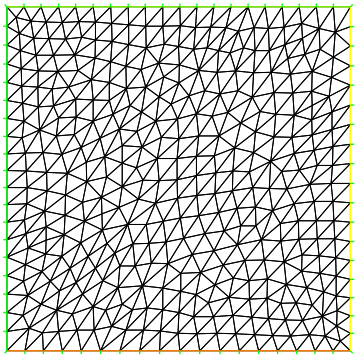


图 5.18: 第一次迭代

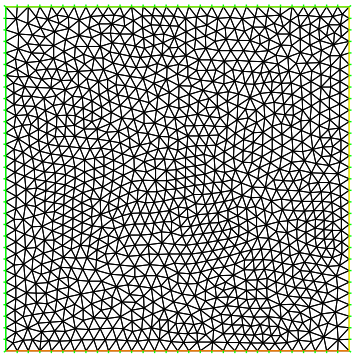


图 5.19: 最后一次迭代

5.6 Trunc

我们已介绍了两种操作来从网格中去除或分割三角形。 操作 Trunc 有两个参数

label= 设置新边界项的 label 数 （默认值 1）

split= 设置三角分割的级别为 n 。 每个三角形被分割成 $n \times n$ （默认值 1）。

为了创建一个将网格 Th 中所有三角形分割成 3×3 的新网格 Th3, 只需写道:

```
mesh Th3 = trunc(Th,1,split=3);
```

示例文件 truncmesh.edp 构造了所有 “trunc” 网格来支撑空间 Vh 中的基本函数（cf. abs(u)>0）, 使用 5×5 来分割所有三角形, 并且在新的边界上设置 label 数为 2。

```
mesh Th=square(3,3);
fespace Vh(Th,P1);
Vh u;
int i,n=u.n;
u=0;
for (i=0;i<n;i++)                                    //    所有自由度
 {
  u[][i]=1;                                          //    基本函数 i
  plot(u,wait=1);
  mesh Sh1=trunc(Th,abs(u)>1.e-10,split=5,label=2);
  plot(Th,Sh1,wait=1,ps="trunc"+i+".eps");           //    画出
                                                     //    函数的支撑图
  u[][i]=0;                                          //    重设
 }
```

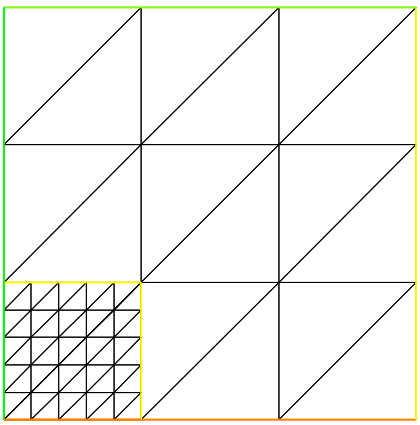


图 5.20: mesh of support the function P1 number 0, splitted in 5×5

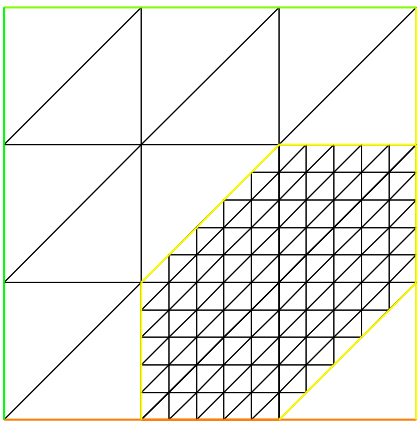


图 5.21: mesh of support the function P1 number 6, splitted in 5×5

5.7 Splitmesh

另一种分割网格三角形的方法是采用 splitmesh, 例如:

```
{                                        //    new stuff 2004 splitmesh (version 1.37)
  assert(version>=1.37);
  border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
  mesh Th=buildmesh(a(20));
  plot(Th,wait=1,ps="nosplitmesh.eps");                    //    see figure 5.22
  Th=splitmesh(Th,1+5*(square(x-0.5)+y*y));
  plot(Th,wait=1,ps="splitmesh.eps");                      //    see figure 5.23
}
```

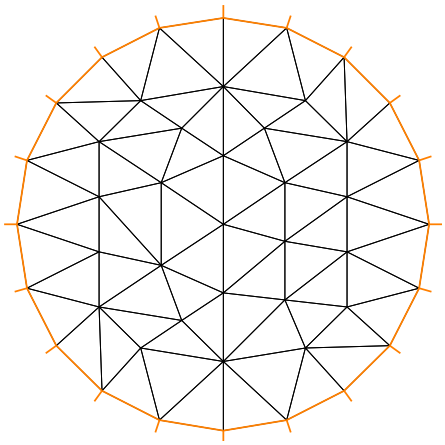


图 5.22: 初始网格

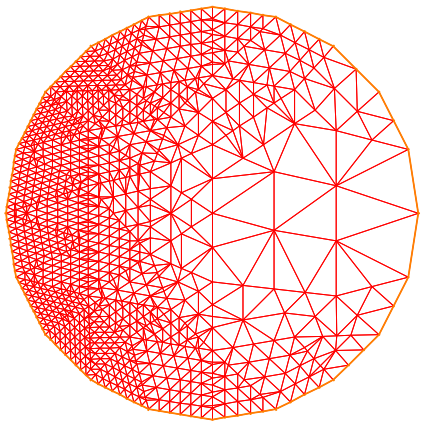


图 5.23: 所有网格左边的三角形被保形分割成 int(1+5*(square(x-0.5)+y*y)^2 个三角形。

5.8 网格例子

Example 5.7 (一边接触两个长方形)

```
border a(t=0,1){x=t;y=0;};
border b(t=0,1){x=1;y=t;};
border c(t=1,0){x=t ;y=1;};
border d(t=1,0){x = 0; y=t;};
border c1(t=0,1){x=t ;y=1;};
border e(t=0,0.2){x=1;y=1+t;};
border f(t=1,0){x=t ;y=1.2;};
border g(t=0.2,0){x=0;y=1+t;};
int n=1;
mesh th = buildmesh(a(10*n)+b(10*n)+c(10*n)+d(10*n));
mesh TH = buildmesh ( c1(10*n) + e(5*n) + f(10*n) + g(5*n) );
plot(th,TH,ps="TouchSide.esp");                                  //    图. 5.24
```

Example 5.8 (NACA0012 机翼)

```
border upper(t=0,1) { x = t;
     y = 0.17735*sqrt(t)-0.075597*t
  - 0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4); }
border lower(t=1,0) { x = t;
     y= -(0.17735*sqrt(t)-0.075597*t
  -0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4)); }
border c(t=0,2*pi) { x=0.8*cos(t)+0.5;  y=0.8*sin(t); }
mesh Th = buildmesh(c(30)+upper(35)+lower(35));
plot(Th,ps="NACA0012.eps",bw=1);                                    //    图. 5.25
```

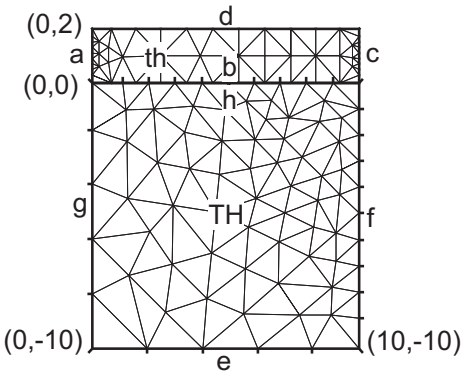


图 5.24: 一边接触两个长方形

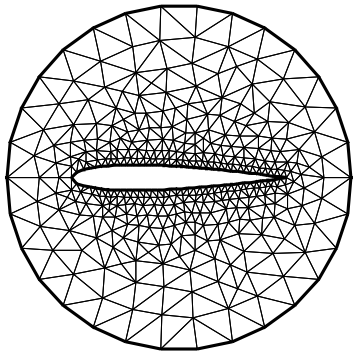


图 5.25: NACA0012 机翼

Example 5.9 (心脏线)

```
real b = 1, a = b;
border C(t=0,2*pi) { x=(a+b)*cos(t)-b*cos((a+b)*t/b);
                     y=(a+b)*sin(t)-b*sin((a+b)*t/b); }
mesh Th = buildmesh(C(50));
plot(Th,ps="Cardioid.eps",bw=1);                                    //    图. 5.26
```

Example 5.10 (Cassini 蛋)

```
border C(t=0,2*pi) { x=(2*cos(2*t)+3)*cos(t);
                     y=(2*cos(2*t)+3)*sin(t); }
mesh Th = buildmesh(C(50));
plot(Th,ps="Cassini.eps",bw=1);                                     //    图. 5.27
```

Example 5.11 (三次 Bezier 曲线)

```
                                          //    一个三次 Bezier 曲线用两个控制点来连接两点
func real bzi(real p0,real p1,real q1,real q2,real t)
{
  return p0*(1-t)^3+q1*3*(1-t)^2*t+q2*3*(1-t)*t^2+p1*t^3;
```

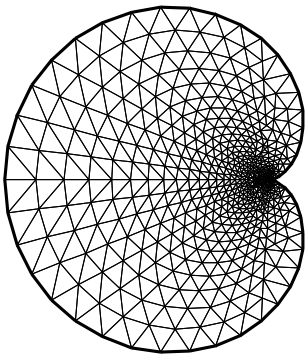


图 5.26: 有心脏线边界的区域

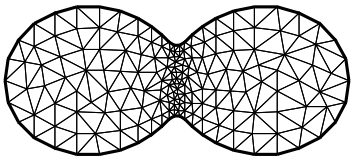


图 5.27: 有 Cassini 蛋边界的区域

```
}

real[int] p00=[0,1], p01=[0,-1], q00=[-2,0.1], q01=[-2,-0.5];
real[int] p11=[1,-0.9], q10=[0.1,-0.95], q11=[0.5,-1];
real[int] p21=[2,0.7], q20=[3,-0.4], q21=[4,0.5];
real[int] q30=[0.5,1.1], q31=[1.5,1.2];
border G1(t=0,1) { x=bzi(p00[0],p01[0],q00[0],q01[0],t);
                   y=bzi(p00[1],p01[1],q00[1],q01[1],t); }
border G2(t=0,1) { x=bzi(p01[0],p11[0],q10[0],q11[0],t);
                   y=bzi(p01[1],p11[1],q10[1],q11[1],t); }
border G3(t=0,1) { x=bzi(p11[0],p21[0],q20[0],q21[0],t);
                   y=bzi(p11[1],p21[1],q20[1],q21[1],t); }
border G4(t=0,1) { x=bzi(p21[0],p00[0],q30[0],q31[0],t);
                   y=bzi(p21[1],p00[1],q30[1],q31[1],t); }
int m=5;
mesh Th = buildmesh(G1(2*m)+G2(m)+G3(3*m)+G4(m));
plot(Th,ps="Bezier.eps",bw=1);                                    //    图 5.28
```

Example 5.12 (发动机部件型区域)

```
real a= 6., b= 1., c=0.5;
border L1(t=0,1) { x= -a; y= 1+b - 2*(1+b)*t; }
border L2(t=0,1) { x= -a+2*a*t; y= -1-b*(x/a)*(x/a)*(3-2*abs(x)/a );}
border L3(t=0,1) { x= a; y=-1-b + (1+ b )*t; }
border L4(t=0,1) { x= a - a*t;   y=0; }
border L5(t=0,pi) { x= -c*sin(t)/2; y=c/2-c*cos(t)/2; }
border L6(t=0,1) { x= a*t;  y=c; }
border L7(t=0,1) { x= a;  y=c + (1+ b-c )*t; }
border L8(t=0,1) { x= a-2*a*t; y= 1+b*(x/a)*(x/a)*(3-2*abs(x)/a); }
mesh Th = buildmesh(L1(8)+L2(26)+L3(8)+L4(20)+L5(8)+L6(30)+L7(8)+L8(30));
plot(Th,ps="Engine.eps",bw=1);                                    //    图.  5.29
```

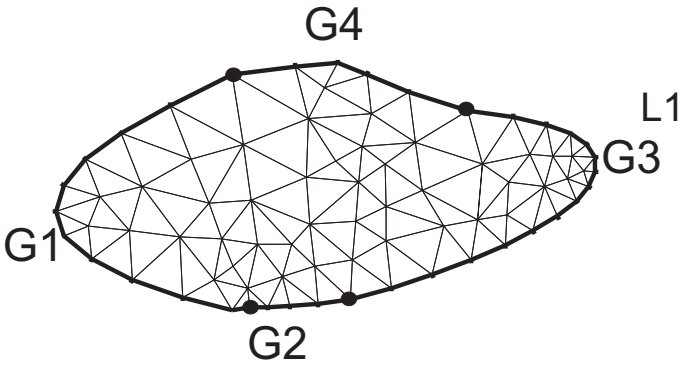


图 5.28: Bezier曲线连接成的边界

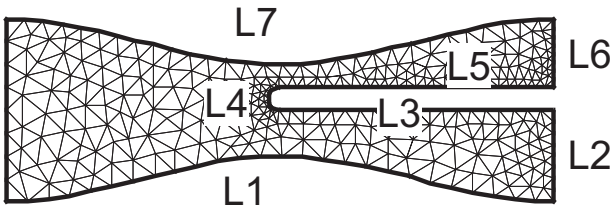


图 5.29: 发动机部件型区域

Example 5.13 (U-型区域)

```
real d = 0.1;                                                    //    U-型的宽度
border L1(t=0,1-d) { x=-1; y=-d-t; }
border L2(t=0,1-d) { x=-1; y=1-t; }
border B(t=0,2) { x=-1+t; y=-1; }
border C1(t=0,1) { x=t-1; y=d; }
border C2(t=0,2*d) { x=0; y=d-t; }
border C3(t=0,1) { x=-t; y=-d; }
border R(t=0,2) { x=1; y=-1+t; }
border T(t=0,2) { x=1-t; y=1; }
int n = 5;
mesh Th = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)+C2(3)+C3(n)+R(n)+T(n));
plot(Th,ps="U-shape.eps",bw=1);                                  //    图 5.30
```

Example 5.14 (V型砍区域)

```
real dAg = 0.01;                                                 //    V-型的角度
border C(t=dAg,2*pi-dAg) { x=cos(t); y=sin(t); };
real[int] pa(2), pb(2), pc(2);
pa[0] = cos(dAg); pa[1] = sin(dAg);
pb[0] = cos(2*pi-dAg); pb[1] = sin(2*pi-dAg);
pc[0] = 0; pc[1] = 0;
border seg1(t=0,1) { x=(1-t)*pb[0]+t*pc[0]; y=(1-t)*pb[1]+t*pc[1]; };
border seg2(t=0,1) { x=(1-t)*pc[0]+t*pa[0]; y=(1-t)*pc[1]+t*pa[1]; };
mesh Th = buildmesh(seg1(20)+C(40)+seg2(20));
plot(Th,ps="V-shape.eps",bw=1);                                  //    图. 5.31
```

Example 5.15 (笑脸)

```
real d=0.1;
int m=5;
real a=1.5, b=2, c=0.7, e=0.01;
border F(t=0,2*pi) { x=a*cos(t); y=b*sin(t); }
border E1(t=0,2*pi) { x=0.2*cos(t)-0.5; y=0.2*sin(t)+0.5; }
```

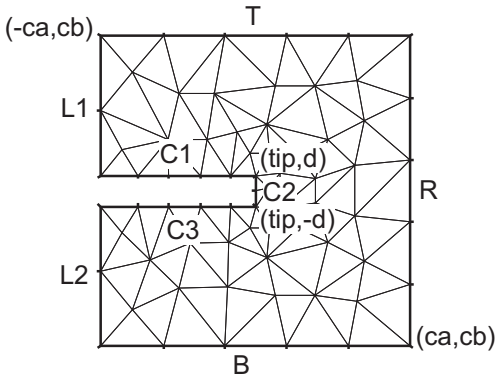


图 5.30: 用d改变的U型通道区域

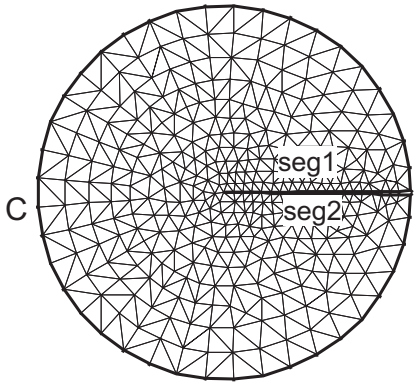


图 5.31: 用 dAg 改变的V型砍区域

```
border E2(t=0,2*pi) { x=0.2*cos(t)+0.5; y=0.2*sin(t)+0.5; }
func real st(real t) {
   return sin(pi*t)-pi/2;
}
border C1(t=-0.5,0.5) { x=(1-d)*c*cos(st(t)); y=(1-d)*c*sin(st(t)); }
border C2(t=0,1){x=((1-d)+d*t)*c*cos(st(0.5));y=((1-d)+d*t)*c*sin(st(0.5));}
border C3(t=0.5,-0.5) { x=c*cos(st(t)); y=c*sin(st(t)); }
border C4(t=0,1) { x=(1-d*t)*c*cos(st(-0.5)); y=(1-d*t)*c*sin(st(-0.5));}

border C0(t=0,2*pi) { x=0.1*cos(t); y=0.1*sin(t); }
mesh Th=buildmesh(F(10*m)+C1(2*m)+C2(3)+C3(2*m)+C4(3)
                  +C0(m)+E1(-2*m)+E2(-2*m));
plot(Th,ps="SmileFace.eps",bw=1);                    //   见 Fig. 5.32
}
```

Example 5.16 (3点弯曲)

```
                                        //   固定在 Fix1, Fix2的三点弯曲的方形区域
                                                //   将被加载在 Load
real a=1, b=5, c=0.1;
int n=5, m=b*n;
border Left(t=0,2*a) { x=-b; y=a-t; }
border Bot1(t=0,b/2-c) { x=-b+t; y=-a; }
border Fix1(t=0,2*c) { x=-b/2-c+t; y=-a; }
border Bot2(t=0,b-2*c) { x=-b/2+c+t; y=-a; }
border Fix2(t=0,2*c) { x=b/2-c+t; y=-a; }
border Bot3(t=0,b/2-c) { x=b/2+c+t; y=-a; }
border Right(t=0,2*a) { x=b; y=-a+t; }
border Top1(t=0,b-c) { x=b-t; y=a; }
border Load(t=0,2*c) { x=c-t; y=a; }
border Top2(t=0,b-c) { x=-c-t; y=a; }
mesh Th = buildmesh(Left(n)+Bot1(m/4)+Fix1(5)+Bot2(m/2)+Fix2(5)+Bot3(m/4)
                    +Right(n)+Top1(m/2)+Load(10)+Top2(m/2));
plot(Th,ps="ThreePoint.eps",bw=1);                   //   图. 5.33
```

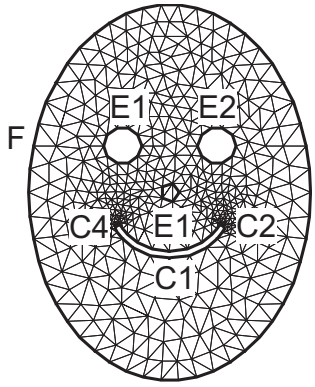


图 5.32: 笑脸 （嘴的形状可变）

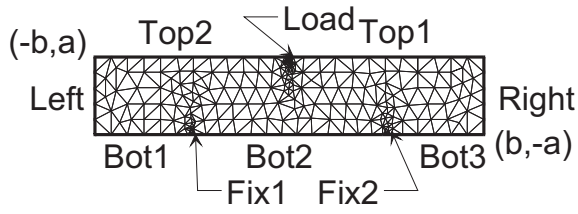


图 5.33: 三点弯曲测试的区域

5.9 如何改变网格中单元和边界单元的指标

改变网格中单元和边界单元的指标可以用关键词 **change** 实现。 这个命令行的参数对应于一个二维和多维的情况:

label = 一个整数向量包含连续的旧指标数和新指标数对。

region = 一个整数向量包含连续的旧区域数和新区域数对。

flabel = 一个包含新指标值的整数值函数, 见（版本 3.21）。

fregion = 一个包含新区域数的整数值函数。

这些向量包含 n_l 个连续的数对 O, N, 这里 n_l 是我们想改变的（区域或指标）数。 例如, 我们有

$$\texttt{label} = [O_1, N_1, ..., O_{n_l}, N_{n_l}] \tag{5.3}$$

$$\texttt{region} = [O_1, N_1, ..., O_{n_l}, N_{n_l}] \tag{5.4}$$

使用这个功能的一个例子见文件 "glumesh2D.edp":

Example 5.17 (glumesh2D.edp)

```
 1:
 2: mesh Th1=square(10,10);
 3: mesh Th2=square(20,10,[x+1,y]);
 4: verbosity=3;
 5: int[int] r1=[2,0],  r2=[4,0];
 6: plot(Th1,wait=1);
 7: Th1=change(Th1,label=r1);                                    //   更改边的标签
 8: plot(Th1,wait=1);
 9: Th2=change(Th2,label=r2);                                    //   更改边的标签
10: mesh Th=Th1+Th2;                                  //    "粘合"网格 Th1 与 Th2
11: cout << " nb lab = " << int1d(Th1,1,3,4)(1./lenEdge)+int1d(Th2,1,2,3)(1./lenEdge)
12:          << " == " << int1d(Th,1,2,3,4)(1./lenEdge) <<" == " << ((10+20)+10)*2
<< endl;
13: plot(Th,wait=1);
```

```
14: fespace Vh(Th,P1);
15: macro Grad(u) [dx(u),dy(u)];                    //    宏的定义
16: Vh u,v;
17: solve P(u,v)=int2d(Th)(Grad(u)'*Grad(v))-int2d(Th)(v)+on(1,3,u=0);
18: plot(u,wait=1);
```

“粘合” 不同的网格 在之前文件的第10行, 我们用到了在 FreeFem++ 中 “粘合” 相同维数的不同网格的方法。 功能的实现是基于不同网格之间的运算符 “+”。 这种方法的实现需要相邻网格的点相同。

5.10 三维网格

5.10.1 三维网格读取和写入声明

在三维网格中, 支持用 FreeFem++ 输入输出文件格式是 .msh和 .mesh。 这些文件格式已经在二维网格格式章节中讨论过。

文件扩展名 .msh 三维的扩展名 .msh文件的结构由下表给出 5.2。 在这种结果中, n_v 表示顶点数, n_{tet} 表示四面体单元数 and n_{tri} 表示三角形单元数。 对于每个顶点 $q^i, i = 1, \cdots, n_v$, 我们用 (q_x^i, q_y^i, q_z^i) 表示 x-坐标, y-坐标和 z-坐标。 每个四面体单元 $T_k, k = 1, \cdots, n_{tet}$ 有四个顶点 q^{k_1}, q^{k_2}, q^{k_3}, q^{k_4}。 边界包含一些三角形。每个三角形 $be_j, j = 1, \cdots, n_{tri}$ 有三个顶点 q^{j_1}, q^{j_2}, q^{j_3}。

n_v	n_{tet}	n_{tri}		
q_x^1	q_y^1	q_z^1	顶点指标	
q_x^2	q_y^2	q_z^2	顶点指标	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$q_x^{n_v}$	$q_y^{n_v}$	$q_z^{n_v}$	顶点指标	
1_1	1_2	1_3	1_4	区域指标
2_1	2_2	2_3	2_4	区域指标
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(n_{tet})_1$	$(n_{tet})_2$	$(n_{tet})_3$	$(n_{tet})_4$	区域指标
1_1	1_2	1_3	边界指标	
2_1	2_2	2_3	边界指标	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$(n_tri)_1$	$(n_{tri})_2$	$(n_{tri})_3$	边界指标	

Table 5.2: 三维网格文件”.msh” 结构。

extension file .mesh 三维网格的数据结构是章节12.1中介绍数据结构的一部分, 并且也是一种四面体单元数据结构。 三维四面体网格单元可以用如下的场作为参考:

- `Tetrahedra`
  `(I)` NbOfTetrahedrons
  $\left(\left(\texttt{@@Vertex}_i^j\text{ , j=1,4}\right)\text{, }\texttt{(I)}\ Ref\phi_i^{tet}\text{ , i=1 , NbOfTetrahedrons}\right)$

表示场所用的记号见 12.1。

5.10.2 TeGen: 四面体网格生成软件

TetGen TetGen 是由德国柏林 Weierstrass 研究所 Hang Si 博士开发的一款软件 [36]。TetGen 对于研究与非商业用途免费。 对于任何商业上的使用, 需要向Hang Si 博士获得商业许可。
这款软件生成三维区域边界内的四面体网格。 输入的区域是多面或者分片线性的复合体。 这种四面体网格化是一种有约束的 Delaunay 四面体网格。
软件 TetGen 中用来控制网格质量的方法是一种由 Shewchuk 提出的 Delaunay 网格改进方法[37]。这种算法质量的度量标准是半径-边缘比（见 1.3.1 节[36]）。Shewchuk的算法的一种理论上半径-边缘比的上界可以由一些顶点, 有约束的划分和网格的面得到, 此时输入的角度都小于90 度。理论上的上界是2.0。

可以用命令行 `tetg` 开始 Tetgen， 这个命令行的参数是:

`label =` 一个整数向量包含下标 $2i$ 处的三角形旧指标和在下标 $2i+1$ 处的三角形新指标。 这个参数由关键词 change 的指标初始化, 见（5.3）。

`switch =` 字符串参数。 这个字符串与 Tetgen 的命令行 switch 用法一致。见 3.2 节 [36]。

`nbofholes=` 孔洞的数量 （默认值 `size of holelist/3` （版本 3.11） ）。

`holelist =` 这个向量与 tetgenio 的数据结构 **holelist** 相对应。见[36]。 它是一个长度为 $3\times$ `nbofholes` 的实数向量。 在 TetGen 中, 每个孔洞与区域内的一个点有关。 这个向量是 $x_1^h, y_1^h, z_1^h, x_2^h, y_2^h, z_2^h, \cdots$, 这里 x_i^h, y_i^h, z_i^h 是与 i^{th} 孔洞相对应的点。

`nbofregions =` 区域的数量 (`size of regionlist/5` （版本 3.11) ）。

`regionlist =` 这个向量与 tetgenio 的数据结构**regionlist**相对应。 见[36]。 区域的属性和体积限制由这个长度为 $5\times$ `nbofregions` 的实向量给出 第 i^{th} 区域由五个元素来描述: 区域内一点 (x_i, y_i, z_i)的 $x-$坐标, $y-$坐标 and $z-$坐标; 属性 (at_i) 和这块区域四面体单元的最大体积 $(mvol_i)$。 参数 `regionlist` 向量是: $x_1, y_1, z_1, at_1, mvol_1, x_2, y_2, z_2, at_2, mvol_2, \cdots$。

`nboffacetcl=` 面的数量的限制 `size of facetcl/2` （版本 3.11) ）。

`facetcl=` 这个向量与 tetgenio 的数据结构 **facetconstraintlist** 相对应。 第 i^{th} 面的限制由 Ref_i^{fc} 和面的最大面积 $marea_i^{fc}$ 定义。 数列 `facetcl` 是: $Ref_1^{fc}, marea_1^{fc}, Ref_2^{fc}, marea_2^{fc}, \cdots$。这个参数不起作用如果switch选项 `q` 不被选择的话。

TetGen中的主要switch参数:

`p` 边界四面体网格化。

`q` 生成优质网格。 半径-边缘比的界由选项 q 给出，默认设置为 2.0。

`a` 由体积限制构建四面体网格。这些体积限制由之前选项 q 给出的界定义或者在命令 `regionlist` 的参数定义。

`A` 对 `regionlist` 中的区域添加指标，其他区域的指标为 0。 选项 `AA` 对每个区域给出不同的指标。这在选择选项 'p' 时有效. 如果选择选项 'r' 则无效。

r 重新构造和改进之前定义的网格。 这个选项只能与命令行 tetgreconstruction 同时使用。

Y 这个选项用来保存外边界的网格。这个选项必须用来确保临近网格之间是共形网格。

YY 这个选项用来保存边界内部和外部的网格。

C 用 TetGen 测试结果网格的相容性。

CC 用 TetGen 测试结果网格的相容性同时检测保角 Delaunay 网格的相容性 （如果选择 'p'）或者同时检测共形Delaunay 网格的相容性 （如果选择 'q'）。

V 给出关于 TetGen 运行的信息. 使用'VV' 或者 'VVV' 能够得到更多信息。

Q 无输出: 除了报错以外无最终输出结果。

M 共面的曲面不会被混合。

T 为共面测试设置一个阈值。 默认值为 $1e-8$。

d 面的交叉会被检测到.。

为了用 tetgen 得到四面体单元网格, 我们需要三维区域边界曲面网格。 我们下面给出 FreeFem++ 中构件这些网格的命令行。

关键词: "movemesh23" 一个简单的构造曲面的方法是把二维区域放置到三维空间中。 这对应于将一个区域按照下列位移向量 $\Phi(x,y) = (\Phi 1(x,y), \Phi 2(x,y), \Phi 3(x,y))$ 进行变换。将一个二维网格 Th2 变换到三维的结果可以由如下得到:

```
mesh3 Th3 = movemesh23(Th2,transfo=[Φ1,Φ2,Φ3]);
```

这个命令行的参数是:

transfo = [Φ1, Φ2, Φ3] 设置变换的位移向量 $\Phi(x,y) = [\Phi 1(x,y), \Phi 2(x,y), \Phi 3(x,y)]$。

label = 设置三角形整数指标。

orientation= 设置网格整数定向。

ptmerge = 实数参数。当变换网格时, 某些点会被合并。 此参数用来定义合并的点。 默认定义如下:

$$ptmerge = 1e-7\ Vol(B),$$

这里 B 是包含区域 Ω 的最小的表面平行于坐标轴的长方体并且 $Vol(B)$ 是这个长方体的体积。

我们可以用章节 5.9 中的方法对曲面进行"粘合"。 一个用命令行 tetg 和 movemesh23 得到三维网格的例子由文件 tetgencube.edp 给出。

Example 5.18 (tetgencube.edp)

```
//     文件 tetgencube.edp
load "msh3"
load "tetgen"

real x0,x1,y0,y1;
x0=1.; x1=2.; y0=0.; y1=2*pi;
```

```
mesh Thsq1 = square(5,35,[x0+(x1-x0)*x,y0+(y1-y0)*y]);

func ZZ1min = 0;
func ZZ1max = 1.5;
func XX1 = x;
func YY1 = y;

mesh3 Th31h = movemesh23(Thsq1,transfo=[XX1,YY1,ZZ1max]);
mesh3 Th31b = movemesh23(Thsq1,transfo=[XX1,YY1,ZZ1min]);

                                        //   ////////////////////////////////
x0=1.; x1=2.; y0=0.; y1=1.5;
mesh Thsq2 = square(5,8,[x0+(x1-x0)*x,y0+(y1-y0)*y]);

func ZZ2 = y;
func XX2 = x;
func YY2min = 0.;
func YY2max = 2*pi;

mesh3 Th32h = movemesh23(Thsq2,transfo=[XX2,YY2max,ZZ2]);
mesh3 Th32b = movemesh23(Thsq2,transfo=[XX2,YY2min,ZZ2]);

                                        //   ////////////////////////////////
x0=0.; x1=2*pi; y0=0.; y1=1.5;
mesh Thsq3 = square(35,8,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
func XX3min = 1.;
func XX3max = 2.;
func YY3 = x;
func ZZ3 = y;

mesh3 Th33h = movemesh23(Thsq3,transfo=[XX3max,YY3,ZZ3]);
mesh3 Th33b = movemesh23(Thsq3,transfo=[XX3min,YY3,ZZ3]);

                                        //   ////////////////////////////////
mesh3 Th33 = Th31h+Th31b+Th32h+Th32b+Th33h+Th33b;  //     “粘合”表面网格以获取立方
体表面
savemesh(Th33,"Th33.mesh");

                                       //     用 TetGen 建立与坐标轴平行的盒子网格
real[int] domain =[1.5,pi,0.75,145,0.0025];
mesh3 Thfinal = tetg(Th33,switch="paAAQY",regionlist=domain);                  //
用tetgen将立方体内部四面体化
savemesh(Thfinal,"Thfinal.mesh");

                                    //     建立一个扇形半柱体, 内半径为1, 外半径为2, 高1.5
func mv2x = x*cos(y);
func mv2y = x*sin(y);
func mv2z = z;
mesh3 Thmv2 = movemesh3(Thfinal, transfo=[mv2x,mv2y,mv2z]);
savemesh(Thmv2,"halfcylindricalshell.mesh")
```

命令 movemesh 的具体使用在如下部分描述.

关键词 ” tetgtransfo “ 这个关键词与命令行 tetg 和movemesh23 的如下调用方式相对应

```
tetgtransfo( Th2, transfo= [Φ1, Φ2, Φ3] ), ··· ) = tetg( Th3surf, ··· ),
```

这里 Th3surf = movemesh23(Th2,tranfo=[Φ1, Φ2, Φ3]) 并且 Th2 是二维网格 tetgtransfo 的输入参数。
这个命令行的参数:

label, switch, regionlist nboffacetcl facetcl

一方面是关键词 tetg 的参数, 另一方面是关键词 movemesh23 的参数 ptmerge 。

注: 为了使用 tetgtransfo, 命令 movemesh23 产生的网格必须是一个闭曲面并且仅仅定义在区域上。 因此, 参数 regionlist 仅定义在一个区域上。

这个关键词的一个例子可以再文件 ”buildlayers.edp” 中找到。

关键词 ”tetgconvexhull” FreeFem++ , 使用软件 tetgen, 可以从一些点钟建立四面体网格. 这个四面体网格是一种 由点集的凸包构成的Delaunay 网格。
这些点的坐标可以用如下方式初始化。 首先产生一个包含所有点坐标的文件 $X_i = (x_i, y_i, z_i)$. 这个文件可以由如下方式组织而成:

$$\begin{array}{ccc} n_v & & \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_{n_v} & y_{n_v} & z_{n_v} \end{array}$$

第二种方法是给出三个数列分别对应 $x-$ 坐标, $y-$ 坐标 and $z-$ 坐标.

这个命令行的参数是

switch = 字符串参数。 这个字符串与TetGen的命令行 switch 用法一致。见 3.2 节 [36]。

reftet = 整数参数 设置四面体的指标。

label = 整数参数。 设置三角形的指标。

在字符串 switch中, 我们不能使用选项 ’p’ 和 ’q’。

5.10.3 使用 TetGen 重新构造/改进 三维网格

三维网格可以用软件TetGen的命令行 tetgreconstruction 改进。
这个关键词的参数是

region= 一个整数向量用来改变区域四面体网格的数量。 这个向量的定义类似于关键词 change 的参数 reftet。

label= 一个整数向量用来改变边界三角形的指标。 这个向量的定义类似于关键词 change 的参数 label。

sizevolume= 一个实函数。 这个函数用来现在去宇宙四面体网格的体积。（见例 5.31 建立三维适应网格）)

软件TetGen (tetg) 的参数 switch nbofregions, regionlist, nboffacetcl 和 facetcl 被用作 tetgrefine 的参数。
在参数 switch= 中, 字母 'r' 应该在没有字母 'p' 的情况下单独使用。 例如, 见软件 TetGen 的手册 [36] 中对于字母 'r' 与其他字母功能的介绍。
参数 regionlist 用来定义一个区域的新的体积限制。 参数 regionlist 的指标也是区域的一个之前的指标。 这个参数和 nbofregions 不能和参数 sizevolume 一起使用。
例:

Example 5.19 (refinesphere.edp)

```
// 文件 refinesphere.edp

load "msh3"
load "tetgen"
load "medit"

mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]);          // [-π/2, π/2[×[0,2π]
                                                   // 球体参数化

func f1 =cos(x)*cos(y);
func f2 =cos(x)*sin(y);
func f3 = sin(x);
                                                   // 参数化 DF 的偏导数
func f1x=sin(x)*cos(y);
func f1y=-cos(x)*sin(y);
func f2x=-sin(x)*sin(y);
func f2y=cos(x)*cos(y);
func f3x=cos(x);
func f3y=0;
                                                   // M = DF^t DF
func m11=f1x^2+f2x^2+f3x^2;
func m21=f1x*f1y+f2x*f2y+f3x*f3y;
func m22=f1y^2+f2y^2+f3y^2;

func perio=[[4,y],[2,y],[1,x],[3,x]];
real hh=0.1;
real vv= 1/square(hh);
verbosity=2;
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
plot(Th,wait=1);

verbosity=2;

                                                   // 构建球体表面
real Rmin  = 1.;
func f1min = Rmin*f1;
func f2min = Rmin*f2;
func f3min = Rmin*f3;

mesh3 Th3=movemesh23(Th,transfo=[f1min,f2min,f3min]);

real[int] domain = [0.,0.,0.,145,0.01];
mesh3 Th3sph=tetg(Th3,switch="paAAQYY",nbofregions=1,regionlist=domain);

int[int] newlabel = [145,18];
real[int] domainrefine = [0.,0.,0.,145,0.0001];
```

```
mesh3 Th3sphrefine=tetgreconstruction(Th3sph,switch="raAQ",reftet=newlabel,
nbofregions=1,regionlist=domain,refinesizeofvolume=0.0001);

int[int] newlabel2 = [145,53];
func fsize = 0.01/(( 1 + 5*sqrt( (x-0.5)^2+(y-0.5)^2+(z-0.5)^2) )^3);
mesh3 Th3sphrefine2=tetgreconstruction(Th3sph,switch="raAQ",reftet=newlabel2,
sizeofvolume=fsize);

medit(``sphere'',Th3sph);
medit(``isotroperefine'' ,Th3sphrefine);
medit(``anisotroperefine'',Th3sphrefine2);
```

5.10.4 在三维空间中移动网格

和二维情形一样（见第5章 movemesh 一节）, 三维空间中的网格可以通过命令行 movemesh 来实现平移、旋转和变形. 若 Ω 的四面体划分为 $T_h(\Omega)$, $\phi(x,y) = (\phi1(x,y,z), \phi2(x,y,z), \phi3(x,y,z))$ 是一个位移矢量, 那么可通过以下方式得到 $\phi(T_h)$:

```
mesh3 Th = movemesh( Th, transfo=[Φ1, Φ2, Φ3], ... );
```

三维空间中 movemesh 的参数为:

region = 在默认情况下设置初始四面体的整数值标签。

label = 设置边界各面的标签。将该参数初始化为关键词 change (5.3)中的标签。

facemerge = 一个整数表达式。 在网格变形时可以合并一些面. 如果考虑进这些被合并的面, 该参数等于1, 否则等于0。在默认情况下, 该参数等于1。

ptmerge = 一个实数表达式。 在网格变形时可以合并一些点. 该参数是定义两个合并点的标准. 在默认情况下, 我们使用:

$$ptmerge \ = \ 1e-7 \ Vol(B),$$

这里 B 是包含离散区域 Ω 的最小轴平行盒, $Vol(B)$ 是这个盒的体积。

orientation = 一个整数表达式（在默认情况下为1） , 无论正反, 四面体的方向始终是非正的。

该命令的例子见目录 examples++-3d 中的文件“Poisson3d.edp”.

5.10.5 层网格

在这一节中, 我们将介绍用来获取层网格的命令行buildlayermesh。该网格是由 z 轴上二维网格的推广得到的。
由层网格定义的定义域 Ω_{3d} 等于 $\Omega_{3d} = \Omega_{2d} \times [zmin, zmax]$, 这里的 Ω_{2d} 是由二维网格定义的区域, $zmin$ 和 $zmax$ 是 Ω_{2d} 在 R 中的函数, 分别表示 Ω_{3d} 的下表面和上表面。
对于二维网格中的一个顶点 $V_i^{2d} = (x_i, y_i)$, 我们引入 $z-$轴上的关联顶点数 $M_i + 1$ 。 记 M_i 的最大值为 M , 该值被称为层数（如果 $\forall i$, $M_i = M$, 那么 Ω_{3d} 的网格中有 M 层）。 V_i^{2d} 生成按如下方式定义的 $M + 1$ 个顶点:

$$\forall j = 0, \ldots, M, \qquad V_{i,j}^{3d} = (x_i, y_i, \theta_i(z_{i,j})),$$

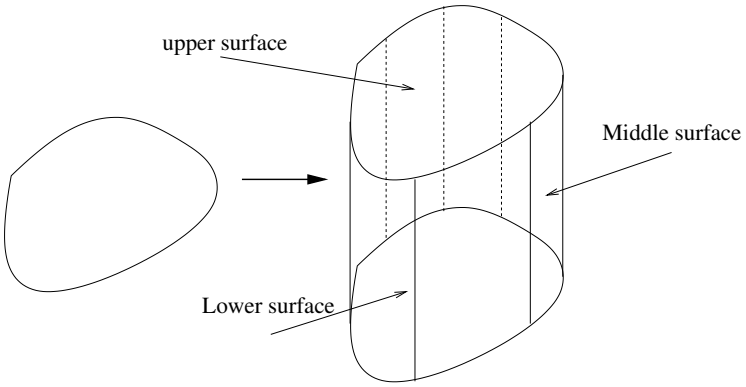


图 5.34: 三维层网格的例子

这里 $(z_{i,j})_{j=0,\ldots,M}$ 是在间隔 $[zmin(V_i^{2d}), zmax(V_i^{2d})]$ 上的 $M+1$ 个等距点:

$$z_{i,j} = j\,\delta\alpha + zmin(V_i^{2d}), \qquad \delta\alpha = \frac{zmax(V_i^{2d}) - zmin(V_i^{2d})}{M}.$$

定义在 $[zmin(V_i^{2d}), zmax(V_i^{2d})]$ 上的函数 θ_i 由以下式子给出:

$$\theta_i(z) = \begin{cases} \theta_{i,0} & \text{若 } z = zmin(V_i^{2d}), \\ \theta_{i,j} & \text{若 } z \in]\theta_{i,j-1}, \theta_{i,j}], \end{cases}$$

这里 $(\theta_{i,j})_{j=0,\ldots,M_i}$ 是在间隔 $[zmin(V_i^{2d}), zmax(V_i^{2d})]$ 上的 M_i+1 个等距点。

设二维网格中的一个三角形 $K = (V_{i1}^{2d}, V_{i2}^{2d}, V_{i3}^{2d})$. K 与层网格上表面(或下表面)的一个三角形 $(V_{i1,M}^{3d}, V_{i2,M}^{3d}, V_{i3,M}^{3d})$ (或 $(V_{i1,0}^{3d}, V_{i2,0}^{3d}, V_{i3,0}^{3d})$)有关。
K也和M个体积棱形元素有关, 其定义如下所示:

$$\forall j = 0, \ldots, M, \quad H_j = (V_{i1,j}^{3d}, V_{i2,j}^{3d}, V_{i3,j}^{3d}, V_{i1,j+1}^{3d}, V_{i2,j+1}^{3d}, V_{i3,j+1}^{3d}).$$

这些体积元可以有一些合并点:

- 0 个合并点 : 棱柱
- 1 个合并点 : 棱锥
- 2 个合并点 : 四面体
- 3 个合并点 : 没有元素

这些带有合并点的体积元被称为退化元。 为了得到四面体网格, 我们将棱锥分解为两个四面体, 将棱柱分解为三个四面体。 选择棱锥和棱柱四边形的面, 其对角线的顶点具有最大指标, 将其切割从而获得这些四面体（原因见 [8]）。

通过分解体积棱形元素获得的在中间面上的三角形, 是由二维网格的边界边缘生成的。 在边界元上的三角形和四面体的标签是由这些相关元素的标签定义的。

`buildlayermesh`的命令行参数是一个二维网格和层数 M 。
该命令的参数为:

`zbound =` [zmin,zmax] 这里 zmin 和 zmax 是函数表达式, 分别定义了下表面网格和上表面网格。

coef = 一个在[0,1]上的函数表达式。 该参数用来引入网格中的退化元。 关联点或顶点 V_i^{2d} 的数目是 $coef(V_i^{2d})M$ 的整数部分。

region = 该向量用来初始化四面体的区域。 类似(5.3), 该向量包含由指标为 $2i$ 的二维区域数和对应的指标为 $2i+1$ 的三维区域数组成的连续对。

labelmid = 该向量通过二维标签数来初始化垂直面或中间面的三维标签数。类似(5.3), 该向量包含由指标为 $2i$ 的二维标签数和对应的指标为 $2i+1$ 的三维标签数组成的连续对。

labelup = 该向量通过二维区域数来初始化上表面的三维标签数。 类似(5.3), 该向量包含由指标为 $2i$ 的二维区域数和对应的指标为 $2i+1$ 的三维标签数组成的连续对。

labeldown = 和上一个参数一样, 不同点在于它是对底面的标签而言的。

此外, 我们也添加一些用于移动网格的后期处理的参数。这些参数相当于命令行movemesh中的transfo, facemerge and ptmerge。 参数region, labelmid, labelup 和 labeldown是由 n_l 个由 O_i, N_l 组成的连续对构成的, 这里 n_l 是我们要求的数（标签或者区域）。
该命令行的例子由buildlayermesh.edp给出。

Example 5.20 (cube.idp)

```
load "medit"
load "msh3"
func mesh3 Cube(int[int] & NN,real[int,int] &BB ,int[int,int] & L)
{
                                                    //    首先构造六面体的六个面.
  real x0=BB(0,0),x1=BB(0,1);
  real y0=BB(1,0),y1=BB(1,1);
  real z0=BB(2,0),z1=BB(2,1);

  int nx=NN[0],ny=NN[1],nz=NN[2];
  mesh Thx = square(nx,ny,[x0+(x1-x0)*x,y0+(y1-y0)*y]);

  int[int] rup=[0,L(2,1)],  rdown=[0,L(2,0)],
    rmid=[1,L(1,0),  2,L(0,1),  3, L(1,1),  4, L(0,0) ];
  mesh3 Th=buildlayers(Thx,nz,   zbound=[z0,z1],
                       labelmid=rmid,   labelup = rup,
                       labeldown = rdown);

  return Th;
}
```

单位立方体的例子:

```
 include "Cube.idp"
 int[int]  NN=[10,10,10];                                  //    在每个方向的步骤数
 real [int,int]  BB=[[0,1],[0,1],[0,1]];                   //    边界框
 int [int,int]  L=[[1,2],[3,4],[5,6]]; //    六个面的标签, 这六个面依次为: 左面, 右面,
                                                    //    前面, 后面, 底面, 上面
mesh3 Th=Cube(NN,BB,L);
medit("Th",Th);                                           //    见图 5.35
```

圆锥体的例子（在一个退化三角形上的轴对称网格）

Example 5.21 (cone.edp)

```
load "msh3"
load "medit"
                                                    //    使用带有一个三角形的buildlayers的圆锥体
real RR=1,HH=1;
border Taxe(t=0,HH){x=t;y=0;label=0;};
border Hypo(t=1,0){x=HH*t;y=RR*t;label=1;};
border Vert(t=0,RR){x=HH;y=t;label=2;};
int nn=10;   real h= 1./nn;
mesh Th2=buildmesh(  Taxe(HH*nn)+ Hypo(sqrt(HH*HH+RR*RR)*nn) + Vert(RR*nn) ) ;
plot(Th2,wait=1);                                                    //    二维网格

int MaxLayersT=(int(2*pi*RR/h)/4)*4;                                 //    层数
real zminT = 0, zmaxT = 2*pi;                                        //    高度为 2*pi
func fx= y*cos(z); func fy= y*sin(z); func fz= x;
int[int] r1T=[0,0], r2T=[0,0,2,2], r4T=[0,2];
                                                                     //    技巧函数:
func deg= max(.01,y/max(x/HH,0.4) /RR);
                                                    //    该函数根据MaxLayersT定义了在轴附近的层所占的比例
mesh3 Th3T=buildlayers(Th2,coef=  deg, MaxLayersT,
           zbound=[zminT,zmaxT],transfo=[fx,fy,fz],
           facemerge=0, region=r1T, labelmid=r2T);
medit("cone",Th3T);                                                  //    见图 5.36
```

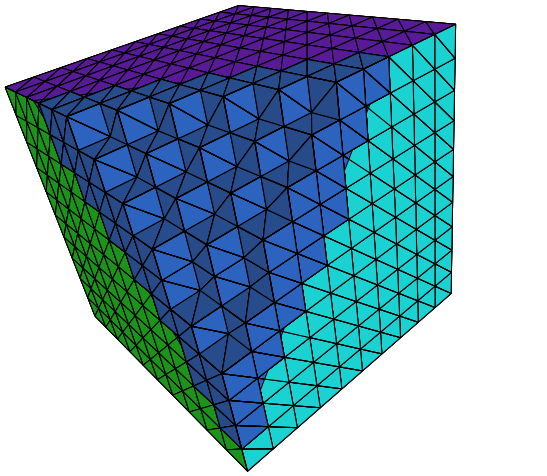


图 5.35: 由 cube.edp 生成的立方体网格

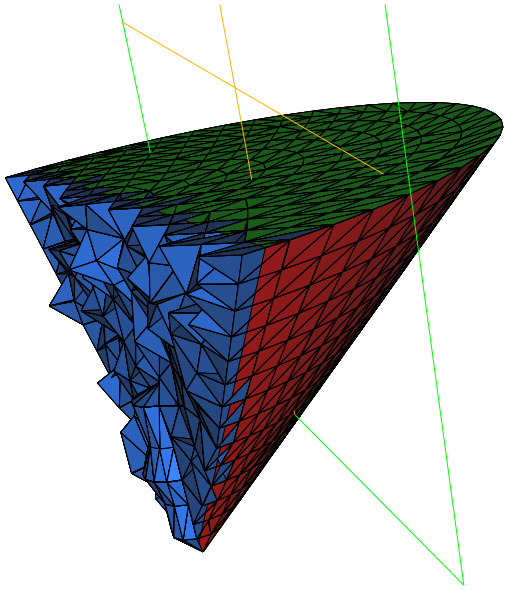


图 5.36: 由 cone.edp 生成的圆锥体网格

Example 5.22 (buildlayermesh.edp)

```
                                                    //    文件 buildlayermesh.edp

load "msh3"
load "tetgen"


                                                                  //    测试 1

int C1=99, C2=98;                                                 //    可以取任意值
border C01(t=0,pi){ x=t;   y=0;       label=1;}
border C02(t=0,2*pi){ x=pi; y=t;   label=1;}
border C03(t=0,pi){ x=pi-t;   y=2*pi;     label=1;}
border C04(t=0,2*pi){ x=0;     y=2*pi-t; label=1;}

border C11(t=0,0.7){ x=0.5+t;   y=2.5;       label=C1;}
border C12(t=0,2){ x=1.2;     y=2.5+t;   label=C1;}
border C13(t=0,0.7){ x=1.2-t;   y=4.5;      label=C1;}
border C14(t=0,2){ x=0.5;     y=4.5-t; label=C1;}

border C21(t=0,0.7){ x= 2.3+t;      y=2.5;   label=C2;}
border C22(t=0,2){          x=3;    y=2.5+t;   label=C2;}
border C23(t=0,0.7){    x=3-t;      y=4.5;   label=C2;}
border C24(t=0,2){         x=2.3;    y=4.5-t; label=C2;}

mesh Th=buildmesh(     C01(10)+C02(10)+ C03(10)+C04(10)
                     + C11(5)+C12(5)+C13(5)+C14(5)
                     + C21(-5)+C22(-5)+C23(-5)+C24(-5));

mesh Ths=buildmesh(     C01(10)+C02(10)+ C03(10)+C04(10)
                     + C11(5)+C12(5)+C13(5)+C14(5) );

                                                    //    构造带有一个孔和两个区域的盒子
func zmin=0.;
func zmax=1.;
int MaxLayer=10;

func XX = x*cos(y);
func YY = x*sin(y);
func ZZ = z;

int[int] r1=[0,41], r2=[98,98,  99,99, 1,56];
int[int] r3=[4,12];          //    上表面的三角形网格是由标签为4和12的二维区域的三角形网
格Th生成的
int[int] r4=[4,45];          //    下表面的三角形网格是由标签为4和45的二维区域的三角形网
格Th生成的

mesh3 Th3=buildlayers( Th, MaxLayer, zbound=[zmin,zmax], region=r1,
                labelmid=r2, labelup = r3, labeldown = r4 );
savemesh(Th3,"box2region1hole.mesh");
                                                    //    用TetGen构造的球体
func XX1 = cos(y)*sin(x);
func YY1 = sin(y)*sin(x);
func ZZ1 = cos(x);
string test="paACQ";
cout << "test=" << test << endl;
mesh3 Th3sph=tetgtransfo(Ths,transfo=[XX1,YY1,ZZ1],switch=test,nbofregions=1,
```

```
                                regionlist=domain);
savemesh(Th3sph,"sphere2region.mesh");
```

5.11 网格化实例

Example 5.23 (lac.edp) // 文件 *"lac.edp"*

```
load ``msh3''
int nn=5;
border cc(t=0,2*pi){x=cos(t);y=sin(t);label=1;}
mesh Th2 = buildmesh(cc(100));
fespace Vh2(Th2,P2);
Vh2 ux,uy,p2;
int[int] rup=[0,2], rdlow=[0,1], rmid=[1,1,2,1,3,1,4,1];
func zmin = 2-sqrt(4-(x*x+y*y));
func zmax = 2-sqrt(3.);

mesh3 Th = buildlayers(Th2,nn,
  coeff = max((zmax-zmin)/zmax, 1./nn),
  zbound=[zmin,zmax],
  labelmid=rmid;
  labelup=rup;
  labeldown=rlow);
savemesh(Th,''Th.meshb'');
exec(``medit Th; Th.meshb'');
```

Example 5.24 (tetgenholeregion.edp) // 文件 *"tetgenholeregion.edp"*

```
load "msh3"
load "tetgen"

mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]);                    // [-pi/2, -pi/2] × [0, 2π]
                                                             // 球的参数化方程
func f1 =cos(x)*cos(y);
func f2 =cos(x)*sin(y);
func f3 = sin(x);
                                                             // 参数化DF的偏导数
func f1x=sin(x)*cos(y);
func f1y=-cos(x)*sin(y);
func f2x=-sin(x)*sin(y);
func f2y=cos(x)*cos(y);
func f3x=cos(x);
func f3y=0;
                                                             // M = DF^t DF
func m11=f1x^2+f2x^2+f3x^2;
func m21=f1x*f1y+f2x*f2y+f3x*f3y;
func m22=f1y^2+f2y^2+f3y^2;

func perio=[[4,y],[2,y],[1,x],[3,x]];
real hh=0.1;
```

```
real vv= 1/square(hh);
verbosity=2;
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
plot(Th,wait=1);

verbosity=2;

                                                        //     球面的构造
real Rmin  = 1.;
func f1min = Rmin*f1;
func f2min = Rmin*f2;
func f3min = Rmin*f3;

mesh3 Th3sph = movemesh23(Th,transfo=[f1min,f2min,f3min]);

real Rmax  = 2.;
func f1max = Rmax*f1;
func f2max = Rmax*f2;
func f3max = Rmax*f3;

mesh3 Th3sph2 = movemesh23(Th,transfo=[f1max,f2max,f3max]);

cout << "addition" << endl;
mesh3 Th3 = Th3sph+Th3sph2;

real[int] domain2 = [1.5,0.,0.,145,0.001,0.5,0.,0.,18,0.001];
cout << "==============================" << endl;
cout << " tetgen call without hole " << endl;
cout << "==============================" << endl;
mesh3 Th3fin = tetg(Th3,switch="paAAQYY",nbofregions=2,regionlist=domain2);
cout << "==============================" << endl;
cout << "finish tetgen call without hole" << endl;
cout << "==============================" << endl;
savemesh(Th3fin,"spherewithtworegion.mesh");

real[int] hole = [0.,0.,0.];
real[int] domain = [1.5,0.,0.,53,0.001];
cout << "==============================" << endl;
cout << "  tetgen call with hole   " << endl;
cout << "==============================" << endl;
mesh3 Th3finhole=tetg(Th3,switch="paAAQYY",nbofholes=1,holelist=hole,
nbofregions=1,regionlist=domain);
cout << "==============================" << endl;
cout << "finish tetgen call with hole   " << endl;
cout << "==============================" << endl;
savemesh(Th3finhole,"spherewithahole.mesh");
```

5.11.1 建立一个带气球的立方体的三维网格

首先调用MeshSurface.idp 文件来建立六面体和球体的边界网格。

```
func mesh3 SurfaceHex(int[int] & N,real[int,int] &B ,int[int,int] & L,int orientation)
{
    real x0=B(0,0),x1=B(0,1);
    real y0=B(1,0),y1=B(1,1);
    real z0=B(2,0),z1=B(2,1);

    int nx=N[0],ny=N[1],nz=N[2];

    mesh Thx = square(ny,nz,[y0+(y1-y0)*x,z0+(z1-z0)*y]);
    mesh Thy = square(nx,nz,[x0+(x1-x0)*x,z0+(z1-z0)*y]);
    mesh Thz = square(nx,ny,[x0+(x1-x0)*x,y0+(y1-y0)*y]);

    int[int] refx=[0,L(0,0)],refX=[0,L(0,1)];  //    对 Xmin, Xmax 的标签重新赋值
    int[int] refy=[0,L(1,0)],refY=[0,L(1,1)];  //    对 Ymin, Ymax 的标签重新赋值
    int[int] refz=[0,L(2,0)],refZ=[0,L(2,1)];  //    对 Zmin, Zmax 的标签重新赋值

    mesh3 Thx0 = movemesh23(Thx,transfo=[x0,x,y],orientation=-orientation,label=refx);
    mesh3 Thx1 = movemesh23(Thx,transfo=[x1,x,y],orientation=+orientation,label=refX);
    mesh3 Thy0 = movemesh23(Thy,transfo=[x,y0,y],orientation=+orientation,label=refy);
    mesh3 Thy1 = movemesh23(Thy,transfo=[x,y1,y],orientation=-orientation,label=refY);
    mesh3 Thz0 = movemesh23(Thz,transfo=[x,y,z0],orientation=-orientation,label=refz);
    mesh3 Thz1 = movemesh23(Thz,transfo=[x,y,z1],orientation=+orientation,label=refZ);
    mesh3 Th= Thx0+Thx1+Thy0+Thy1+Thz0+Thz1;
    return Th;
}


func mesh3 Sphere(real R,real h,int L,int orientation)
{
  mesh  Th=square(10,20,[x*pi-pi/2,2*y*pi]);                    //    [-π/2, π/2] × [0, 2π]
                                                                //    球的参数方程
  func f1 =cos(x)*cos(y);
  func f2 =cos(x)*sin(y);
  func f3 = sin(x);
                                                                //    偏导数
  func f1x=sin(x)*cos(y);
  func f1y=-cos(x)*sin(y);
  func f2x=-sin(x)*sin(y);
  func f2y=cos(x)*cos(y);
  func f3x=cos(x);
  func f3y=0;
                                                                //    球M = DF^t DF上的度量
  func m11=f1x^2+f2x^2+f3x^2;
  func m21=f1x*f1y+f2x*f2y+f3x*f3y;
  func m22=f1y^2+f2y^2+f3y^2;

  func perio=[[4,y],[2,y],[1,x],[3,x]];                         //    储存周期性条件

  real hh=h/R;                                                  //    单位球上的网格大小hh
  real vv= 1/square(hh);
  Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
  Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
```

```
  Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
  Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
  int[int] ref=[0,L];

  mesh3  ThS= movemesh23(Th,transfo=[f1*R,f2*R,f3*R],orientation=orientation,refface=ref
  return ThS;
}
```

检验这两个函数, 调用tetgen来生成网格

```
load "tetgen"
include "MeshSurface.idp"
   real hs = 0.1;                                          //     球上的网格大小
   int[int]  N=[20,20,20];
   real [int,int]  B=[[-1,1],[-1,1],[-1,1]];
   int [int,int]  L=[[1,2],[3,4],[5,6]];
   mesh3 ThH = SurfaceHex(N,B,L,1);
   mesh3 ThS =Sphere(0.5,hs,7,1);                  //     将表面网格黏接为tolat边界网格

   mesh3 ThHS=ThH+ThS;
   savemesh(ThHS,"Hex-Sphere.mesh");
   exec("ffmedit Hex-Sphere.mesh;rm Hex-Sphere.mesh");          //     见 5.37

   real voltet=(hs^3)/6.;
   cout << " voltet = " << voltet << endl;
   real[int] domaine = [0,0,0,1,voltet,0,0,0.7,2,voltet];

   mesh3 Th = tetg(ThHS,switch="pqaAAYYQ",nbofregions=2,regionlist=domaine);
   medit("Cube-With-Ball",Th);                                 //     见 5.38
```

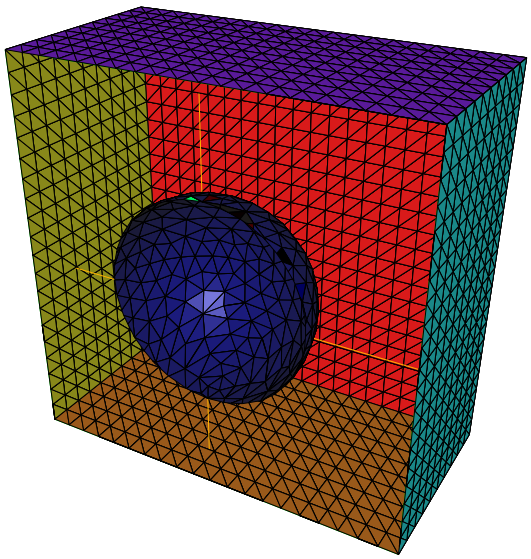


图 5.37: 带有内置球体的六面体的表面网格

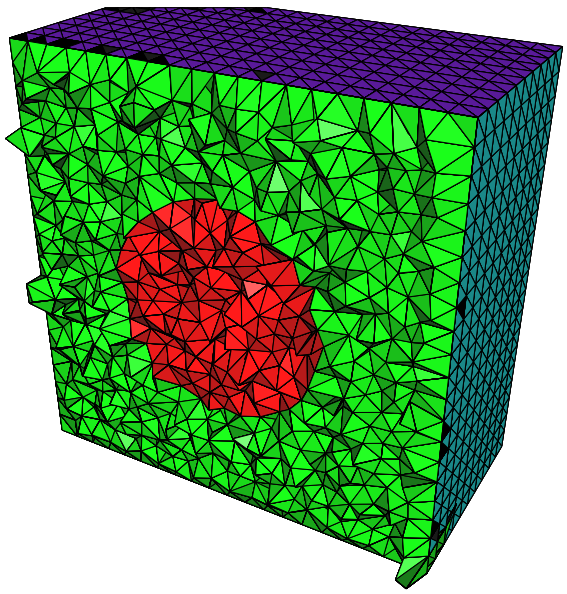


图 5.38: 带有内置球的立方体的四面体网格

5.12 输出解格式 .sol 和 .solb

通过使用关键词 *savesol* , 我们可以储存一个标量函数, 一个标量有限元函数, 一个矢量场, 一个向量有限元场, 一个对称张量和一个对称有限元张量。 该格式在 medit 中使用。

扩展名.sol 文件的头两行是

- `MeshVersionFormatted 0`
- `Dimension (I)` dim

余下的部分由这些关键词之一开始: SolAtVertices, SolAtEdges, SolAtTriangles, SolAtQuadrilaterals, SolAtTetrahedra, SolAtPentahedra, SolAtHexahedra.
在这些关键词的下一行, 我们给出解的对应元素的个数 （SolAtVertices: 顶点个数, SolAtTriangles: 三角形个数, ...） 再往下, 我们给出解的个数, 解的类型 （1: 标量, 2: 向量, 3: 对称张量）。 最后, 我们给出解的元素的值。
文件必须由关键词 End 终止。
若对称张量的值为

$$ST^{3d} = \begin{pmatrix} ST^{3d}_{xx} & ST^{3d}_{xy} & ST^{3d}_{xz} \\ ST^{3d}_{yx} & ST^{3d}_{yy} & ST^{3d}_{yz} \\ ST^{3d}_{zx} & ST^{3d}_{zy} & ST^{3d}_{zz} \end{pmatrix} \qquad ST^{2d} = \begin{pmatrix} ST^{2d}_{xx} & ST^{2d}_{xy} \\ ST^{2d}_{yx} & ST^{2d}_{yy} \end{pmatrix} \tag{5.5}$$

在 .sol 里面以这样的顺序存储: $ST^{3d}_{xx}, ST^{3d}_{yx}, ST^{3d}_{yy}, ST^{3d}_{zx}, ST^{3d}_{zy}, ST^{3d}_{zz}, ST^{2d}_{xx}, ST^{2d}_{yx}, ST^{2d}_{yy}$.
一个关键词为 SolAtTetrahedra 的例子:

- `SolAtTetrahedra`
  `(I)` NbOfTetrahedrons

  `nbsol` `typesol`^{1} ... `typesol`^{n}

$$\left(\left(\left(\mathtt{U}^k_{ij}, \quad \forall i \in \{1, ..., \mathtt{nbrealsol}^k\}\right), \quad \forall k \in \{1, ...\mathtt{nbsol}\}\right) \quad \forall j \in \{1, ..., \mathtt{NbOfTetrahedrons}\}\right)$$

其中

- `nbsol` 是解的个数
- `typesol`^{k}, 第 k 个解的类型, 分别为
  - `typesol`$^k = 1$ 第 k 个解是标量。
  - `typesol`$^k = 2$ 第 k 个解是矢量。
  - `typesol`$^k = 3$ 第 k 个解是对称张量或者对称矩阵。
- `nbrealsol`^{k} 描述第 k 个解需要的个数, 分别为
  - `nbrealsol`$^k = 1$ 第 k 个解是标量。
  - `nbrealsol`$^k =$ `dim` 第 k 个解是矢量 （dim 是解的维度）。
  - `nbrealsol`$^k =$ `dim` $*$ (`dim` $+ 1$)$/2$ 第 k 个解是对称张量或者对称矩阵。
- $\mathtt{U}^k_{ij}$ 为在对应网格上, 第 k 个解的第 i 个部分在第 j 个四面体的值。

这一部分的记号和第 12.1 节相同。 格式 .solb 和 .sol 一样, 但是是二进制的 （读 / 写更快, 存储量更小）。
一个实值函数 $f1$, 一个向量场 $\Phi = [\Phi1, \Phi2, \Phi3]$, 以及一个对称张量 ST^{3d} (5.5), 对应的三维网格 Th3 的顶点存储在 “f1PhiTh3.sol” 文件中, 则使用命令

```
savesol("f1PhiST3dTh3.sol",Th3, f1, [Φ1, Φ2, Φ3], VV3, order=1);
```

其中 $VV3 = [ST^{3d}_{xx}, ST^{3d}_{yx}, ST^{3d}_{yy}, ST^{3d}_{zx}, ST^{3d}_{zy}, ST^{3d}_{zz}]$. 对于一个二维的网格 Th, 一个实值函数 $f2$, 对应的向量场 $\Psi = [\Psi1, \Psi2]$, 以及对称张量 ST^{2d} (5.5), 相应的三角形存储在 ”f2PsiST2dTh3.solb” 文件中, 则使用命令

```
savesol("f2PsiST2dTh3.solb",Th, f2, [Ψ1, Ψ2], VV2, order=0);
```

其中 $VV2 = [ST^{2d}_{xx}, ST^{2d}_{yx}, ST^{2d}_{yy}]$. 函数 `savesol` 的输入分别是文件名, 网格以及求解问题。输入必须按照这个顺序。
这个关键词的参数是：

order = 0 表示解给出的是元素重心的值。 1 表示解给出了元素顶点的值。

在文件中, 解是按照这样的顺序存储的: 标量解、向量解、 以及对称张量解。

5.13 medit

通过 Pascal Frey 的免费软件 medit, 可以独立显示网格, 或者定义在网格上的函数。 Medit 有独立的窗口, 且大量使用 OpenGL； 但是要使用相关命令，须事先安装 medit 。
medit 能够通过在网格 Th 的顶点值可视化连续或分片线性的标量解 $f1$ 和 $f2$ ，使用命令

```
medit("sol1 sol2",Th, f1, f2, order=1);
```

第一个名称为 “sol1” 的图像显示 $f1$。 第二个名称为 “sol2” 的图像展示 $f2$。
函数 `medit` 的输入是 （由空格间隔）的不同图片的名字、 网格和解。 每个解对应一张图片，其中标量, 向量以及对称张量解的格式定义请参考关键词 `savesol`。
该命令行的参数是：

order = 0 表示解给出的是元素重心的值。 1 表示解给出了元素顶点的值。

meditff = 设置 medit 的执行命令，默认是 medit。

save = 设置 .sol 或者 .solb 的文件名以存储解。

这个命令行也能在同一个窗口下展示两个不同的网格和解，但要求解的域必须一样。 因此, 使用区域分解算法能在同一个窗口下可视化不同域。 用 medit 可视化标量解 $h1$ 和 $h2$, 对应网格的顶点分别为 Th1 和 Th2, 则可以使用命令

```
medit("sol2domain",Th1, h1, Th2, h2, order=1);
```

Example 5.25 (meditddm.edp)

```
//     meditddm.edp
load "medit"

//      初始问题：
//      求解如下 EDP:
//      −Δu_s = f  on  Ω = {(x, y)|1 ≤ sqrt(x^2 + y^2) ≥ 2}
```

```
//     -Δu1 = f1  on  Ω1 = {(x,y)|0.5 ≤ sqrt(x^2+y^2) ≥ 1.}
//     u = 1  on  Γ  + Null Neumman condition on Γ1 and on Γ2
//     在 Ω 和 Ω1 上求解两个 EDP 来得到解 u
                                                    //     使用 medit 将解可视化

verbosity=3;

border Gamma(t=0,2*pi){x=cos(t); y=sin(t); label=1;};
border Gamma1(t=0,2*pi){x=2*cos(t); y=2*sin(t); label=2;};
border Gamma2(t=0,2*pi){x=0.5*cos(t); y=0.5*sin(t); label=3;};

                                                    //     构造 Ω 的网格
mesh Th=buildmesh(Gamma1(40)+Gamma(-40));

fespace Vh(Th,P2);
func f=sqrt(x*x+y*y);
Vh us,v;
macro Grad2(us) [dx(us),dy(us)]                                    //     EOM

problem Lap2dOmega(us,v,init=false)=int2d(Th)(Grad2(v)' *Grad2(us))
   - int2d(Th)(f*v)+on(1,us=1) ;

//     求解 Ω 上的 EDP
//     -Δu_s = f_1  on  Ω_1,  u_s = 1  on  Γ_1,  ∂u_s/∂n = 0  on  Γ_2
Lap2dOmega;

                                                    //     构造 Ω1 的网格
mesh Th1=buildmesh(Gamma(40)+Gamma2(-40));

fespace Vh1(Th1,P2);
func f1=10*sqrt(x*x+y*y);
Vh1 u1,v1;
macro Grad21(u1) [dx(u1),dy(u1)]                                   //     EOM

problem Lap2dOmega1(u1,v1,init=false)=int2d(Th1)(Grad21(v1)' *Grad21(u1))
            - int2d(Th1)(f1*v1)+on(1,u1=1) ;
//     求解 Ω1 上的 EDP
//     -Δu_1 = f_1  on  Ω_1,  u - 1 = 1  on  Γ_1,  ∂u_1/∂n = 0  on  Γ_2
Lap2dOmega1;

                                                    //     可视化初始问题
medit("solution",Th,us,Th1,u1,order=1,save="testsavemedit.solb");
```

Example 5.26 (StockesUzawa.edp)

```
                                                    //     压强的符号是正确的
assert(version>1.18);
real s0=clock();
mesh Th=square(10,10);
fespace Xh(Th,P2),Mh(Th,P1);
Xh u1,u2,v1,v2;
Mh p,q,ppp;

varf bx(u1,q) = int2d(Th)( (dx(u1)*q));
varf by(u1,q) = int2d(Th)( (dy(u1)*q));
varf a(u1,u2)=  int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) )
```

```
                    +  on(1,2,4,u1=0)  +  on(3,u1=1) ;

Xh bc1; bc1[] = a(0,Xh);
Xh b;

matrix A= a(Xh,Xh,solver=CG);
matrix Bx= bx(Xh,Mh);
matrix By= by(Xh,Mh);
Xh bcx=1,bcy=0;

func real[int] divup(real[int] & pp)
{
  int verb=verbosity;
   verbosity=0;
   b[]  = Bx'*pp; b[] += bc1[] .*bcx[];
   u1[] = A^-1*b[];
   b[]  = By'*pp; b[] += bc1[] .*bcy[];
   u2[] = A^-1*b[];
   ppp[] =   Bx*u1[];
   ppp[] +=  By*u2[];
   verbosity=verb;
   return ppp[] ;
};
p=0;q=0;u1=0;v1=0;

LinearCG(divup,p[],q[],eps=1.e-6,nbiter=50);

divup(p[]);

plot([u1,u2],p,wait=1,value=true,coef=0.1);
medit("velocity pressure",Th,[u1,u2],p,order=1);
```

5.14 Mshmet

Mshmet 是由 P. Frey 开发的软件, 可以根据解 （如 Hessian-based） 来计算各向异性度量。软件同样可以计算各向同性度量。 Mshmet 还能用来构造适合计算水平集方法的度量。 解可以建立在结构化 / 非结构化的 2D 或 3D 的网格上。 比如说, 解可以是 FE 解的误差估计。
mshmet 的调用输入有多种，可以是函数、向量函数、对称张量、FE 函数、FE 向量函数、 以及 FE 对称张量等。 对称张量的数据结构和 datasol 里面定义的相同，该软件可以同时输入多个解。比方说, 现在有定义在网格 Th 上的解 u, 使用 metric 来计算相应的度量 M, 则可以写作

```
fespace Vh(Th,P1);
Vh u;                                          //     标量 FE 函数
real[int] M = mshmet(Th,u);
```

关键词 mshmet 的参数为

- `normalization = <b>` 把解归一化至 $[0, 1]$。
- `aniso = <b>` 为 1 则构造各向异性度量 （ 默认 0: 各向同性）。
- `levelset = <b>` 构造水平集方法的度量 （默认: 否）。
- `verbosity = <l>`

- nbregul = <l> 解的归一化迭代次数 （默认: 0）。
- hmin = <d>
- hmax = <d>
- err = <d> 误差等级。
- width = <d> 宽度。
- metric= double 型向量， 包括传递给 mshmet 的初始度量. Metric 向量的结构在下一段落描述。
- loptions= 7 维的整型向量。 该向量是 mshmet 的整型参数 （专家使用）。
  - loptions(0): 归一化 （默认 1）。
  - loptions(1): 各向同性参数 （默认 0）。1 为各向同性度量, 否则为 0。
  - loptions(2): 水平集参数 （默认 0）。 1 为构造水平集度量, 否则为 0.
  - loptions(3): 调试参数 （默认 0）。1 为开启调试模式, 否则为 0。
  - loptions(4): 详细级别 （默认 100）。
  - loptions(5): 解的归一化迭代次数 （默认 0）。
  - loptions(6): 先验度量参数 （默认 0）。 1 为使用先验度量, 否则为 0。
- doptions= 4 维的 double 型向量。 该向量是 mshmet 的 double 型参数 （专家使用）。
  - doptions(0): hmin : 最小尺寸 （默认 0.01）。
  - doptions(1): hmax : 最大尺寸 （默认 1.0）。
  - doptions(2): eps : 容忍误差 （默认 0.01）。
  - doptions(2): width : 水平集 （$0 < w < 1$） 的相对宽度 （默认0.05）。

关键词 mshmet 的输出是 real[int] 型的, 里面包括了网格上不同顶点 V_i 由 mshmet 计算的度量。
用 nv 表示顶点个数, 该向量的结构是

$$M_{iso} = (m(V_0), m(V_1), \ldots, m(V_{nv}))^T,$$

对于一个各向同性的度量 m。对于一个对称张量度量 $h = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$, metric 的输出是

$$M_{aniso} = (H(V_0), \ldots, H(V_{nv}))^T,$$

其中 $H(V_i)$ 是一个 6 维向量, 定义为 [m11,m21,m22,m31,m32,m33]。

Example 5.27 (mshmet.edp)

```
load "mshmet"
load "medit"
load "msh3"

border a(t=0,1.0){x=t;   y=0;  label=1;};
border b(t=0,0.5){x=1;   y=t;  label=2;};
border c(t=0,0.5){x=1-t; y=0.5;label=3;};
border d(t=0.5,1){x=0.5; y=t;  label=4;};
border e(t=0.5,1){x=1-t; y=1;  label=5;};
```

```
border f(t=0.0,1){x=0;   y=1-t;label=6;};
mesh Th = buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
savemesh(Th,"th.msh");
fespace Vh(Th,P1);
Vh u,v;
real error=0.01;
problem Problem1(u,v,solver=CG,eps=1.0e-6) =
    int2d(Th,qforder=2)( u*v*1.0e-10+  dx(u)*dx(v) + dy(u)*dy(v))
  +int2d(Th,qforder=2)( (x-y)*v);

func zmin=0;
func zmax=1;
int MaxLayer=10;
mesh3 Th3 = buildlayers(Th,MaxLayer,zbound=[zmin,zmax]);
fespace Vh3(Th3,P2);
fespace Vh3P1(Th3,P1);
Vh3 u3,v3;
Vh3P1 usol;
problem Problem2(u3,v3,solver=sparsesolver) =
   int3d(Th3)( u3*v3*1.0e-10+ dx(u3)*dx(v3) + dy(u3)*dy(v3) + dz(u3)*dz(v3))
  - int3d(Th3)( v3) +on(0,1,2,3,4,5,6,u3=0);
Problem2;
cout << u3[].min << " " << u3[].max << endl;
savemesh(Th3,"metrictest.bis.mesh");
savesol("metrictest.sol",Th3,u3);

real[int] bb=mshmet(Th3,u3);
cout << bb << endl;
for(int ii=0; ii<Th3.nv; ii++)
  usol[][ii]=bb[ii];
savesol("metrictest.bis.sol",Th3,usol);
```

5.15 FreeYams

FreeYams 是由 P. Frey 开发的表面网格自适应软件，是新版的 yams。 自适应的表面网格是由几何度量张量场构建的。 该张量场是基于离散表面的固有性质的。 这个软件也能构建简单的网格。抽取参考是基于初始和现在三角测量之间的 Hausdorff 距离。 相较于 yams, FreeYams 还能构建适应于水平集模拟的各向异性三角测量。 目前还没有关于 FreeYams 的技术性报告， 但是关于 yams 的文档可参见 http://www.ann.jussieu.fr/~frey/software.html [40].
在 Freefem++ 中调用 FreeYams, 我们使用关键词 freeyams。 该函数的输入是初始网格和 / 或 度量。 FreeYams 中的度量可以是 double 型的函数、 FE 函数、 对称张量函数、 对称 FE 张量函数以及向量。 如果度量是 double 型的向量, 这个数据必须是 `metric` 格式的，或者用 `metric` 作为输入。
例如, 定义 FE 函数型的度量 u, 计算 $Thinit$ 的自适应网格可以使用命令:

```
fespace Vh(Thinit,P1);
Vh u;
mesh3 Th=freeyams(Thinit,u);
```

关键词 freeyams 中的对称张量的结构和 datasol 中的一样。

- `aniso = <b>` 各向同性或各向异性度量 （默认 0, 各向同性）。

- `mem = <l>` freeyams 可以使用的内存, 单位是 Mb （默认 -1, freeyams 自行决定）。
- `hmin = <d>`
- `hmax = <d>`
- `gradation = <d>`
- `option = <l>`

  0 : 网格优化 （平滑和替换）。

  1 : 抽取和增加适应于度量映射 （默认）。

  -1 : 抽取适应于度量映射。

  2 : 使用 Hausdorff-like 方法进行抽取和增加。

  -2 : 使用 Hausdorff-like 方法进行抽取.。

  4 : 递归地分裂三角形。

  9 : 非收缩顶点平滑。
- `ridgeangle = <d>`
- `absolute = <b>`
- `verbosity = <i>`
- `metric=` 向量表示。 该参数包含了初始网格不同顶点的度量，设 nv 为顶点的个数, 该向量则为

$$M_{iso} = (m(V_0), m(V_1), \ldots, m(V_{nv}))^T$$

  度量 m 是标量。若是对称张量度量 $h = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$, `metric` 的参数是

$$M_{aniso} = (H(V_0), \ldots, H(V_{nv}))^T$$

  其中 $H(V_i)$ 是一个 6 维向量, 定义为 `[m11,m21,m22,m31,m32,m33]`。
- `loptions=` 13 维的整型向量。 该向量是 FreeYams 的整型参数 （专家使用）。
  - loptions(0): 各向异性参数 （默认 0）。1 为各向异性度量, 否则为 0。
  - loptions(1): 有限元修正参数 （默认 0）。 1 为不使用有限元修正, 否则为 0。
  - loptions(2): 分裂多连接点参数 （默认 1）。 1 分裂多连接点, 否则为 0。
  - loptions(3): 内存的最大使用量, 单位为 Mbytes （默认 -1: 大小由 freeyams 指定）。
  - loptions(4): 设置目标的连接成分的值 （注: freeyams 自动给定每个连接成分的值）。
  - loptions(5): 详细级别。
  - loptions(6): （非映射下） 直边加点参数 （默认 0）。 1 为直边加点, 否则为 0。
  - loptions(7): 光滑有效性验证参数。 该参数只对应于非收缩顶点平滑优化 （优化选项 9）。 1 为不验证光滑的有效性, 否则为 0。
  - loptions(8): 目标顶点个数 （默认 -1）。
  - loptions(9): 优化迭代次数 （默认 30）。
  - loptions(10): 不检测参数 （默认 0）。 1 为检测网格的脊线, 否则为 0。 脊线的定义由参数 doptions(10) 给出。
  - loptions(11): 不光滑顶点参数 （默认 0）。 1 为光滑顶点, 否则为 0。

  - loptions(12): 优化等级参数 （默认 0）。
    * 0：网格优化 （平滑和替换）
    * 1：抽取和增加适应于度量映射。
    * -1：抽取适应于度量映射。
    * 2：使用 Hausdorff-like 方法进行抽取和增加。
    * -2：使用 Hausdorff-like 方法进行抽取。
    * 4：递归地分裂三角形。
    * 9：非收缩顶点平滑。

- doptions= 11 维的 double 型向量。 该向量是 FreeYams 的 double 型参数。
  - doptions(0): 设置几何近似 （切平面偏差）（默认 0.010.
  - doptions(1): 设置 lamda 参数 （默认 -1）。
  - doptions(2): 设置 mu 参数 （默认 -1）。
  - doptions(3): 设置灰度值 （网格密度控制） （默认 1.3）。
  - doptions(4): 设置最小尺寸 （hmin） （默认 -2.0: 自动计算尺寸）。
  - doptions(5): 设置最大尺寸 （hmax） （默认 -2.0: 自动计算尺寸）。
  - doptions(6): 设置弦偏差控制的容忍 （默认 -2.0）。
  - doptions(7): 设置降解质量 （默认 0.599）。
  - doptions(8): 设置 declic 参数 （默认 2.0）。
  - doptions(9): 设置角 walton 限制参数 （默认 45 度）。
  - doptions(10): 设置角脊线检测 （默认 45 度）。

Example 5.28 (freeyams.edp)

```
load "msh3"
load "medit"
load "freeyams"
int nn=20;
mesh Th2=square(nn,nn);
fespace Vh2(Th2,P2);
Vh2 ux,uz,p2;
int[int] rup=[0,2],  rdown=[0,1], rmid=[1,1,2,1,3,1,4,1];
real zmin=0,zmax=1;
mesh3 Th=buildlayers(Th2,nn, zbound=[zmin,zmax],
                          reffacemid=rmid, reffaceup = rup, reffacelow = rdown);

mesh3 Th3 = freeyams(Th);
medit("maillagesurfacique",Th3,wait=1);
```

5.16 mmg3d

Mmg3d 是由 C. Dobrzynski 和 P. Frey 开发的 3D 网格重构软件 （http://www.math.u-bordeaux1.fr/~dobj/log
需要这个库的请发邮件至: cecile.dobrzynski@math.u-bordeaux1.fr 或者 pascal.frey@upmc.fr。
这个软件可以根据四面体结构的初始网格来重构网格。初始网格自适应于几何度量张量场, 或者适应于位移向量 （移除刚体）。 度量可以由 mshmet 计算 （见第 5.14 节）。

注 5 *:*

(a) 如果没有给定度量, 会根据初始网格边的尺寸来计算一个各向同性的度量。

(b) 如果给定了位移, 移动表面三角形的顶点后, 不会检验新表面网格的几何结构。

mmg3d 的参数为

- `options=` 向量表示。该向量包含了 `mmg3d` 的可选参数。是一个 6 维向量, 每一维分别表示

  (0) 优化参数: （默认 1）
  0 : 网格优化。
  1 : 适应于度量 （删除和增加顶点） 和优化。
  -1 : 适应于度量 （删除和增加顶点） 而不优化。
  4 : 分裂四面体 （注意表面的变化）。
  9 : 移动网格并优化。
  -9 : 移动网格而不优化。

  (1) 调试模式: （默认 0）
  1 : 开启调试模式。
  0 : 其他.。

  (2) 指定每一维度的 bucket 大小 （默认 64）

  (3) 替换模式: （默认 0）
  1 : 不改变边或面。
  0 : 其他。

  (4) 增加点模式: （默认 0）)
  1 : 不分裂或折叠边也不增加点。
  0 : 其他。

  (5) 详细级别 （默认 3）。

- `memory=` 整数表示。设置新网格的最大存储量, 单位为Mbytes。 默认时, 最大顶点、四面体、三角形分别为 500 000、 3000 000、100000, 这大约需要 100 M 的内存空间。
- `metric=` 向量表示。该向量包含了传递给 mmg3d 的度量。 是一个 nv 维 or 6 nv 维的向量, 表示各向同性或各向异性的度量, 其中nv 是初始网格的顶点个数。 向量结构的描述见第 5.14 节。
- `displacement= [Φ1, Φ2, Φ3]` 设置初始网格的位移向量
  $\Phi(x,y) = [\Phi 1(x,y), \Phi 2(x,y), \Phi 3(x,y)]$.
- `displVect=` 以向量的形式设置位移向量。 该向量包括初始网格每个点的位移。 是一个 3 nv 维的向量。

使用该函数的例子见 ”mmg3d.edp”:

Example 5.29 (mmg3d.edp)

```
                                                                    //     测试 mmg3d
load "msh3"
load "medit"
load "mmg3d"
include "../examples++-3d/cube.idp"

int n=6;
int[int]  Nxyz=[12,12,12];
real [int,int]  Bxyz=[[0.,1.],[0.,1.],[0.,1.]];
int [int,int]  Lxyz=[[1,1],[2,2],[2,2]];
mesh3 Th=Cube(Nxyz,Bxyz,Lxyz);

real[int] isometric(Th.nv);{
  for( int ii=0; ii<Th.nv; ii++)
    isometric[ii]=0.17;
}

mesh3 Th3=mmg3d(  Th,  memory=100, metric=isometric);

medit("init",Th);
medit("isometric",Th3);
```

文件“fallingspheres.edp”给出一个网格移动的例子:

Example 5.30 (fallingspheres.edp)
```
load "msh3"  load "tetgen"  load "medit"  load "mmg3d"
include "MeshSurface.idp"

                                  //     建立一个有两个洞（300,310）的盒状网格（311）

real hs = 0.8;
int[int]  N=[4/hs,8/hs,11.5/hs];
real [int,int]  B=[[-2,2],[-2,6],[-10,1.5]];
int [int,int]  L=[[311,311],[311,311],[311,311]];
mesh3 ThH = SurfaceHex(N,B,L,1);
mesh3 ThSg =Sphere(1,hs,300,-1);
mesh3 ThSd =Sphere(1,hs,310,-1);   ThSd=movemesh3(ThSd,transfo=[x,4+y,z]);
mesh3 ThHS=ThH+ThSg+ThSd;                                          //     粘合表面网格
medit("ThHS", ThHS);                                                //     见表面网格

real voltet=(hs^3)/6.;
real[int] domaine = [0,0,-4,1,voltet];
real [int] holes=[0,0,0,0,4,0];
mesh3 Th = tetg(ThHS,switch="pqaAAYYQ",nbofregions=1,regionlist=domaine, nbofholes=2,hol
medit("Box-With-two-Ball",Th);
                                                                    //     网格建立完毕

int[int] opt=[9,0,64,0,0,3];                        //     mmg3d选项, 见 freeem++ 文件
real[int] vit=[0,0,-0.3];
func zero = 0.;
func dep  = vit[2];

fespace Vh(Th,P1);
```

```
macro Grad(u) [dx(u),dy(u),dz(u)]                                    //

Vh uh,vh;                                                    //    计算位移场
problem Lap(uh,vh,solver=CG) = int3d(Th)(Grad(uh)'*Grad(vh))   //   ') 为 emacs
  + on(310,300,uh=dep) +on(311,uh=0.);

for(int it=0; it<29; it++){
  cout<<"  ITERATION       "<<it<<endl;
  Lap;
  plot(Th,uh);
  Th=mmg3d(Th,options=opt,displacement=[zero,zero,uh],memory=1000);
 }
```

5.17 一个3维isotope网格自适应过程

Example 5.31 (Laplace-Adapt-3d.edp)

```
load "msh3" load "tetgen" load "mshmet" load "medit"
                                                          //   建立初始网格
int nn  = 6;
int[int] l1111=[1,1,1,1],l01=[0,1],l11=[1,1];        //   标签编号, 所有标签都为 1
mesh3 Th3=buildlayers(square(nn,nn,region=0,label=l1111),
      nn,  zbound=[0,1],  labelmid=l11,   labelup = l01,  labeldown = l01);
Th3 = trunc(Th3,(x<0.5) | (y < 0.5) | (z < 0.5) ,label=1);//    去掉 $]0.5,1[^3$ 立方
体
                                                          //   建立初始网格建立完毕
fespace Vh(Th3,P1);
Vh u,v,usol,h;

macro Grad(u) [dx(u),dy(u),dz(u)]                                  //    EOM

problem Poisson(u,v,solver=CG) = int3d(Th3)( Grad(u)'*Grad(v) )
                                -int3d(Th3)( 1*v ) + on(1,u=0);

real errm=1e-2;                                                //    误差水平
for(int ii=0; ii<5; ii++)
{
  Poisson;                                                  //    求解 Poisson 方程
  cout <<" u min, max = " <<  u[].min << " "<< u[].max << endl;
  h=0. ;                                              //    给 h[] 重新排列, 因为网格变化了
  h[]=mshmet(Th3,u,normalization=1,aniso=0,nbregul=1,hmin=1e-3,hmax=0.3,err=errm);
  cout <<" h min, max = " <<  h[].min << " "<< h[].max
       << " " << h[].n << " " << Th3.nv << endl;
  plot(u,wait=1);
  errm*= 0.8;                                                //    改变误差水平
  cout << " Th3" << Th3.nv < " " << Th3.nt << endl;
  Th3=tetgreconstruction(Th3,switch="raAQ",sizeofvolume=h*h*h/6.); //   重新建立
网格
  medit("U-adap-iso-"+ii,Th3,u,wait=1);}
```

5.18 由等值线构建2维网格

思路是把等值线离散化后流动生成网格，这种方法可以用于构见给出图像的区域网格。 首先，我们给出一个等值间隔为0.2，以解析函数$\sqrt{(x-1/2)^2+(y-1/2)^2}$ 为值在单位正方形上构建网格的例子。

Example 5.32 (isoline.edp)

```
load "isoline"                                          //    加载 “isoline”

real[int,int] xy(3,1);                                          //    保存等值点
int[int] be(1);                                        //    保存开头和结束几行
{                                                          //    内存管理的一块
  mesh Th=square(10,10);                               //   ,[x*.5,y*0.5]);
  fespace Vh(Th,P1);
  Vh u= sqrt(square(x-0.5)+square(y-0.5));
  real iso= 0.2 ;
  real[int] viso=[iso];
  plot(u,viso=viso,Th);                                          //    观察等值线

  int nbc= isoline(Th,u,xy,close=1,iso=iso,beginend=be,smoothing=0.1);
```

等值线的参数有网格*Th*，u的表达式，存储格点坐标的二维方阵 *xy*。下列为命名的参数：

iso= 要计算的等值线的值 （默认值为**0**）。

close= 用边界将等值线封闭起来 *(*定义为真*)*，我们规定网格边界上的值小于等值线上的值。

smoothing= 光滑处理程度，为$l^r s$ ，其中 l是当前等值线的长度， r比例， s 是光滑度。其默认值为 **0**。

ratio= 比例 *(* 默认为 1*)*。

eps= 相对 ε （见代码 *??*） （定义为 *1e-10* ）

beginend= 每一等值线起点和终点构成的阵列（自动调整大小）

file= 保存曲线数据到数据文件给 *gnu* 用以画图。

数组xy中是等值线上顶点的列表，这些顶点在每条连通的曲线为$i = i_0^c, \ldots, i_1^c - 1$ ，这里 $i_0^c = be(2*c)$， $i_1^c = be(2*c+1)$,且 $x_i = xy(0,i), y_i = yx(1,i), l_i = xy(2,i)$。 其中$l_i$ 是曲线的长度*(*曲线的起点记为 i_0^c*)*。
对等值线的识别使得值高的部分在等值线的左侧。 因此这里*:* 最小的一个点是 $0.5, 05$，因此曲线 *1* 是按顺时针方向识别的，每一部分的是按照每个点递减的顺序排列的。

```
  cout << " nb of the line  component   = " << nbc << endl;
  cout << " n = " << xy.m << endl;                                  //    点的个数
  cout << "be = " << be << endl;                            //    每个部分的起点和终点

                                                                    //    显示曲线
  for( int c=0;c<nbc; ++c)
  {
    int i0 = be[2*c], i1 = be[2*c+1]-1;                    //    曲线部分的起点和终点
    cout << " Curve " << c << endl;
    for(int i=i0; i<= i1; ++i)
```

```
      cout << " x= " << xy(0,i) <<" y= " << xy(1,i) << " s= "
           << xy(2,i) << endl;
    plot([xy(0,i0:i1),xy(1,i0:i1)],wait=1,viso=viso,cmm = " curve "+c);
  }
}                                                          //    内存管理完毕

cout << " len of  last  curve = " << xy(2,xy.m-1) << endl;;
```

对于一间断的 *Curve*我们也有一个新的函数来参数化，定义为be, xy。

```
border Curve0(t=0,1)                                       //    外边界
{ int c =0;                                                //    部分 0
  int i0 = be[2*c], i1 = be[2*c+1]-1;
  P=Curve(xy,i0,i1,t);                                     //    曲线 0
  label=1;
}

border Curve1(t=0,1)
{ int c =1;                                                //    部分 1
  int i0 = be[2*c], i1 = be[2*c+1]-1;
  P=Curve(xy,i0,i1,t);                                     //    曲线 1
  label=1;
}

plot(Curve1(100));                                         //    显示曲线
mesh Th= buildmesh(Curve1(-100));                          //
plot(Th,wait=1);                                           //
```

其次，根据这个思路来对图像来构建网格，我们利用插件ppm2rnm 来读取 pgm 灰度模式图像，并且提取灰度在0.25以上的图像轮廓。

Example 5.33 (Leman-mesh.edp)

```
load "ppm2rnm" load "isoline"
string leman="lg.pgm";                                     //    见图 5.39
real AreaLac =  580.03;                                    //    单位 Km^2
real hsize= 5;                                       //    以像素为单位的网格尺寸
real[int,int] Curves(3,1);
int[int] be(1);
int nc;                                                    //    曲线条数
{
  real[int,int] ff1(leman);                          //    读取图像 (图 5.39)
                                                     //    设置矩形数组
  int nx = ff1.n, ny=ff1.m;                          //    0到1之间的灰值 （深色）
                                               //    建立一个笛卡尔网格，使得原点在适当的位置
  mesh Th=square(nx-1,ny-1,[(nx-1)*(x),(ny-1)*(1-y)]);
                                                     //    警告，顶点(x,y)的编号为
                                                     //    i = x/nx + nx * y/ny
  fespace Vh(Th,P1);
   Vh f1; f1[]=ff1;                                  //    将数组转变为有限元函数
  nc=isoline(Th,f1,iso=0.25,close=1,Curves,beginend=be,smoothing=.1,ratio=0.5);
}
                                                     //    最长等值线 : lac ..
int ic0=be(0), ic1=be(1)-1;
```

```
plot([Curves(0,ic0:ic1),Curves(1,ic0:ic1)], wait=1);
int NC= Curves(2,ic1)/hsize;
border G(t=0,1) {  P=Curve(Curves,ic0,ic1,t);  label= 1 + (x>xl)*2 + (y<yl);}

plot(G(-NC),wait=1);
mesh Th=buildmesh(G(-NC));
plot(Th,wait=1);
real scale = sqrt(AreaLac/Th.area);
Th=movemesh(Th,[x*scale,y*scale]);                                    //   网格重新排列
cout << " Th.area = " << Th.area << " Km^2 " << " == " << AreaLac <<  "   Km^2 "
<< endl ;
plot(Th,wait=1,ps="leman.eps");                                       //    见图 5.40
```

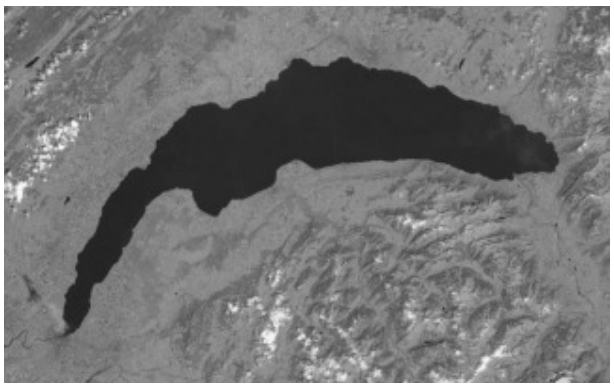


图 5.39: 日内瓦湖的图像

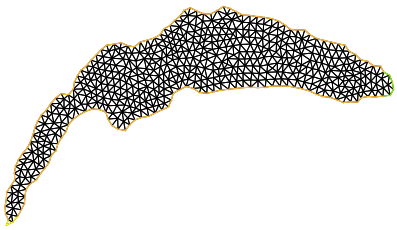


图 5.40: 日内瓦湖的网格划分

第 6 章

有限元

就像第 2章开始一样 对于所有的函数w，FEM 逼近形式是

$$w(x,y) \simeq w_0\phi_0(x,y) + w_1\phi_1(x,y) + \cdots + w_{M-1}\phi_{M-1}(x,y)$$

其中 $\phi_k(x,y)$ 为有限元基函数， w_k $(k = 0, \cdots, M-1)$为系数。 函数$\phi_k(x,y)$通过三角形T_{i_k}构造出来，被称为形状函数（*shape functions*）。 在 FreeFem++ 中，有限元空间为

$$V_h = \{w \mid w_0\phi_0 + w_1\phi_1 + \cdots + w_{M-1}\phi_{M-1},\ w_i \in \mathbb{R}\}$$

可以很容易用以下命令构造：

```
fespace IDspace(IDmesh,<IDFE>) ;
```

或者 有ℓ 对周期边界条件，2维时

```
fespace IDspace(IDmesh,<IDFE>,
                periodic=[[la_1,sa_1],[lb_1,sb_1],
                          ...
                          [la_k,sa_k],[lb_k,sb_ℓ]]);
```

3维时

```
fespace IDspace(IDmesh,<IDFE>,
                periodic=[[la_1,sa_1,ta_1],[lb_1,sb_1,tb_1],
                          ...
                          [la_k,sa_k,ta_k],[lb_k,sb_ℓ,tb_ℓ]]);
```

这里
IDspace 是空间名称 （e.g. Vh），

IDmesh 是相应的网格的名称， <IDFE> 表明了有限元的类型。

在2维中我们有一对周期边界条件， 如果 [la_i,sa_i],[lb_i,sb_i] 是一对整数，那么 2 个标签 la_i 和 lb_i 代表等式中的两段边界。
如果 [la_i,sa_i],[lb_i,sb_i] 是一对实数，那么 sa_i 和 sb_i 在两条边界曲线上给出公共横坐标，如果横坐标相等，那么这两个点重合。

在3维中我们有一对周期边界条件，如果 [la_i,sa_i,ta_i],[lb_i,sb_i,tb_i] 是一对整数，那么 2 个标签 la_i and lb_i 代表等式中的两段边界。
如果 [la_i,sa_i,ta_i],[lb_i,sb_i,tb_i] 是一对实数，那么 sa_i,ta_i and sb_i,tb_i 在两条边界曲线上给出公共坐标参数，若这两个坐标分别相等，则这两个点重合。

注 6 *2*维网格中两个边界条件的边界必须一致， 因此为了确保一致，在*buildmesh* 命令中选择参数 *fixeborder=true (*如 *5.1.2)* 中的例子 *periodic2bis.edp (*见 *9.7)*。

时到今日，我们已知的有限元类型有：

P0,P03d 分段常值不连续的有限元 （2维，3维）， 自由度是重心元的值。

$$P0_h = \left\{ v \in L^2(\Omega) \,\middle|\, \text{对于任意的 } K \in \mathcal{T}_h \text{ 存在 } \alpha_K \in \mathbb{R} : \ v_{|K} = \alpha_K \right\} \tag{6.1}$$

P1,P13d 分段线性连续的有限元 （2维，3维），自由度是顶点的值。

$$P1_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \right\} \tag{6.2}$$

P1dc 分段线性不连续的有限元

$$P1dc_h = \left\{ v \in L^2(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \right\} \tag{6.3}$$

警告！由于插值问题，自由度不是顶点而是三个顶点向内移动到 $T(X) = G + .99(X - G)$ 处的值，这里 G 是重心，（版本 2.24-4）。

P1b,P1b3d 分段线性连续的有限元外加球泡（bubble） （2 维，3 维）

2维情形:

$$P1b_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \oplus \text{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K\} \right\} \tag{6.4}$$

3维情形:

$$P1b_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \oplus \text{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K \lambda_3^K\} \right\} \tag{6.5}$$

这里$\lambda_i^K, i = 0, .., d$ 是元K（三角形或者四面体）的 $d+1$重心坐标函数。

P2,P23d 分段 P_2 连续的有限元 （2维，3维），

$$P2_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \right\} \tag{6.6}$$

这里 P_2 次数小于等于2的 $\mathbb{R}^2$ 中的多项式。

P2b 分段 P_2 连续的有限元外加球泡（bubble），

$$P2_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \oplus \text{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K\} \right\} \tag{6.7}$$

P2dc 分段 P_2 不连续的有限元，

$$P2dc_h = \left\{ v \in L^2(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \right\} \tag{6.8}$$

警告！由于插值的问题，自由度不是六个 P2 节点而是六个节点向内移动到 $T(X) = G + .99(X - G)$的值，这里 G 是重心 （版本 2.24-4）。

P3 分段 P_3 连续的有限元（2维） （需要 `load "Element_P3"`），

$$P2_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_3 \right\} \tag{6.9}$$

这里 P_3 是 $\mathbb{R}^2$中次数小于等于3的多项式。

P3dc 分段 P_3 不连续的有限元 （2维） （需要 **load** "Element_P3dc"），

$$P2_h = \left\{ v \in L^2(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_3 \right\} \tag{6.10}$$

这里 P_3 是 $\mathbb{R}^2$中次数小于等于3的多项式。

P4 分段 P_4 连续的有限元 （2维） （需要 **load** "Element_P4"），

$$P2_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_4 \right\} \tag{6.11}$$

这里 P_4 是 $\mathbb{R}^2$中次数小于等于4的多项式。

P4dc 分段 P_4不连续的有限元 （2维） （需要 **load** "Element_P4dc"），

$$P2_h = \left\{ v \in L^2(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_4 \right\} \tag{6.12}$$

这里 P_4 是 $\mathbb{R}^2$中次数小于等于4的多项式。

P0Edge 分段 P_0不连续的有限元 （2维），在网格的每条边上为常数。

P1Edge 分段 P_1不连续的有限元 （2维） （需要 **load** "Element_P1Edge"），在网格的每条边上P_1。

P2Edge 分段 P_2不连续的有限元 （2维） （需要 **load** "Element_P2Edge"），在网格的每条边上P_2。

P3Edge 分段 P_3不连续的有限元 （2维） （需要 **load** "Element_P3Edge"），在网格的每条边上P_3。

P4Edge 分段 P_4不连续的有限元 （2维） （需要 **load** "Element_P4Edge"），在网格的每条边上P_4。

P5Edge 分段 P_5不连续的有限元 （2维） （需要 **load** "Element_P5Edge"），在网格的每条边上P_5。

Morley 分段 P_2 非一致有限元 （2维） （需要 **load** "Morley"）

$$P2_h = \left\{ v \in L^2(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_3, \left\{ \begin{array}{c} v \text{ 在顶点处连续}, \\ \partial_n v \text{ 在边的中间连续}, \end{array} \right\} \right. \tag{6.13}$$

这里 P_2 是 $\mathbb{R}^2$中次数小于等于2的多项式。

注意在对此有限元构建插值函数 u （标量）时，我们需要函数值和2 个偏导数(u, u_x, u_y)，因此此向量值有限元有3个分量(u, u_x, u_y)。

见examples++-load中解决BiLaplacien问题的例子 bilapMorley.edp：

```
load "Morley"
fespace Vh(Th,P2Morley);                          //    Morley 有限元空间
macro bilaplacien(u,v) ( dxx(u)*dxx(v)+dyy(u)*dyy(v)+2.*dxy(u)*dxy(v))
                                                  //    fin 宏
real f=1;
Vh [u,ux,uy],[v,vx,vy];

solve bilap([u,ux,uy],[v,vx,vy]) =
    int2d(Th)(  bilaplacien(u,v) )
  - int2d(Th)(f*v)
  + on(1,2,3,4,u=0,ux=0,uy=0)
```

P2BR （需要 `load "BernadiRaugel"`） Bernadi Raugel Finite Eleme 有限元的元为有两个分量的向量 （2 维）， 见Bernardi, C., Raugel, G.: Analysis of some finite elements for the Stokes problem. Math. Comp. 44, 71-79 (1985). 它是一个2维的双有限元，有多项式空间 $P1^2+3$ 个标准的球泡边界函数 (P_2)，并且它的自由度包括3个顶点的2个分量的6个值、3条边的流量的3个值，因此为9。

RT0,RT03d 0阶Raviart-Thomas 有限元。

2维情形：

$$RT0_h=\left\{\mathbf{v}\in H(\mathrm{div})\,\middle|\,\forall K\in\mathcal{T}_h\quad \mathbf{v}_{|K}(x,y)=\begin{vmatrix}\alpha_K^1\\ \alpha_K^2\end{vmatrix}+\beta_K\begin{vmatrix}x\\ y\end{vmatrix}\right\} \tag{6.14}$$

3维情形：

$$RT0_h=\left\{\mathbf{v}\in H(\mathrm{div})\,\middle|\,\forall K\in\mathcal{T}_h\quad \mathbf{v}_{|K}(x,y,z)=\begin{vmatrix}\alpha_K^1\\ \alpha_K^2\\ \alpha_K^3\end{vmatrix}+\beta_K\begin{vmatrix}x\\ y\\ z\end{vmatrix}\right\} \tag{6.15}$$

这里把 $\mathrm{div}\ \mathbf{w}=\sum_{i=1}^d \partial w_i/\partial x_i$ 写作 $\mathbf{w}=(w_i)_{i=1}^d$,

$$H(\mathrm{div})=\left\{\mathbf{w}\in L^2(\Omega)^d\,\middle|\,\mathrm{div}\ \mathbf{w}\in L^2(\Omega)\right\}$$

其中 $\alpha_K^1,\alpha_K^2,\alpha_K^3,\beta_K$ 是实数。

RT0Ortho 2 维 0 阶 Raviart-Thomas 正交化或者 Nedelec 第 I 类有限元。

$$RT0Orthoh=\left\{\mathbf{v}\in H(\mathrm{curl})\,\middle|\,\forall K\in\mathcal{T}_h\quad \mathbf{v}_{|K}(x,y)=\begin{vmatrix}\alpha_K^1\\ \alpha_K^2\end{vmatrix}+\beta_K\begin{vmatrix}-y\\ x\end{vmatrix}\right\} \tag{6.16}$$

Edge03d 0 阶 Nedelec 有限元或者边界元。

3维情形：

$$Edge0_h=\left\{\mathbf{v}\in H(\mathrm{Curl})\,\middle|\,\forall K\in\mathcal{T}_h\quad \mathbf{v}_{|K}(x,y,z)=\begin{vmatrix}\alpha_K^1\\ \alpha_K^2\\ \alpha_K^3\end{vmatrix}+\begin{vmatrix}\beta_K^1\\ \beta_K^2\\ \beta_K^3\end{vmatrix}\times\begin{vmatrix}x\\ y\\ z\end{vmatrix}\right\} \tag{6.17}$$

这里把 $\mathrm{curl}\mathbf{w}=\begin{vmatrix}\partial w_2/\partial x_3-\partial w_3/\partial x_2\\ \partial w_3/\partial x_1-\partial w_1/\partial x_3\\ \partial w_1/\partial x_2-\partial w_2/\partial x_1\end{vmatrix}$ 写作 $\mathbf{w}=(w_i)_{i=1}^d$,

$$H(\mathrm{curl})=\left\{\mathbf{w}\in L^2(\Omega)^d\,\middle|\,\mathrm{curl}\ \mathbf{w}\in L^2(\Omega)^d\right\}$$

并且 $\alpha_K^1,\alpha_K^2,\alpha_K^3,\beta_K^1,\beta_K^2,\beta_K^3$ 是实数。

P1nc 分段线性并且在边界的中间连续，只用在2维情形。

RT1 （需要 `load "Element_Mixte"`, 版本 3.13）

$$RT1_h=\left\{\mathbf{v}\in H(\mathrm{div})\,\middle|\,\forall K\in\mathcal{T}_h\quad (\alpha_K^2,\alpha_K^2,\beta_K)\in P_1^3,\mathbf{v}_{|K}(x,y)=\begin{vmatrix}\alpha_K^1\\ \alpha_K^2\end{vmatrix}+\beta_K\begin{vmatrix}x\\ y\end{vmatrix}\right\} \tag{6.18}$$

RT1Ortho （需要 `load "Element_Mixte"`, 版本 3.13, 2维）

$$RT1_h=\left\{\mathbf{v}\in H(\mathrm{curl})\,\middle|\,\forall K\in\mathcal{T}_h\quad (\alpha_K^2,\alpha_K^2,\beta_K)\in P_1^3,\mathbf{v}_{|K}(x,y)=\begin{vmatrix}\alpha_K^1\\ \alpha_K^2\end{vmatrix}+\beta_K\begin{vmatrix}-y\\ x\end{vmatrix}\right\} \tag{6.19}$$

BDM1 （需要 `load "Element_Mixte"`, 版本 3.13, 2维） Brezzi-Douglas-Marini 有限元

$$BDM1_h = \left\{ \mathbf{v} \in H(\mathrm{div}) \,\middle|\, \forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K} \in P_1^2 \right\} \tag{6.20}$$

BDM1Ortho （需要 `load "Element_Mixte"`, 版本 3.13, 2维） Brezzi-Douglas-Marini正交化或者叫做 Nedelec 第II类有限元

$$BDM1Ortho_h = \left\{ \mathbf{v} \in H(\mathrm{curl}) \,\middle|\, \forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K} \in P_1^2 \right\} \tag{6.21}$$

TDNNS1 （需要 `load "Element_Mixte"`, 版本 3.13, 2维） 一个新元素的有限元，在空间$H(divdiv)$ 中用对称2x2阶矩阵逼近三角形（即 σ_{nn}在边上连续）。

$$TDNNS1h = \left\{ \boldsymbol{\sigma} \in (L^2)^{2,2} \,\middle|\, \forall K \in \mathcal{T}_h \quad \boldsymbol{\sigma}_{|K} \in P_1^2, \sigma_{12} = \sigma_{21}, \sigma_{nn}\text{连续} \right\} \tag{6.22}$$

这里$\sigma_{nn} = n^t \sigma n$, 且 n是边的法线 （详细说明见[41, section 4.2.2.3]）。

6.1 “fespace” 在二维中的用法

对于一个二维有限元空间

$$X_h = \{ v \in H^1([0,1]^2) | \ \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \}$$

$$X_{ph} = \{ v \in X_h | \ v(|\begin{smallmatrix}0\\\cdot\end{smallmatrix}) = v(|\begin{smallmatrix}1\\\cdot\end{smallmatrix}), v(|\begin{smallmatrix}\cdot\\0\end{smallmatrix}) = v(|\begin{smallmatrix}\cdot\\1\end{smallmatrix}) \}$$

$$M_h = \{ v \in H^1([0,1]^2) | \ \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \}$$

$$R_h = \{ \mathbf{v} \in H^1([0,1]^2)^2 | \ \forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K}(x,y) = |\begin{smallmatrix}\alpha_K\\\beta_K\end{smallmatrix} + \gamma_K |\begin{smallmatrix}x\\y\end{smallmatrix} \}$$

其中$\mathcal{T}_h$是一个单位正方形$]0,1[^2$中的10×10的网格，我们只需在`FreeFem++` 中写：

```
mesh Th=square(10,10);
fespace Xh(Th,P1);                                          //    标量有限元
fespace Xph(Th,P1,
         periodic=[[2,y],[4,y],[1,x],[3,x]]);               //   双周期有限元
fespace Mh(Th,P2);                                          //    标量有限元
fespace Rh(Th,RT0);                                         //    向量有限元
```

其中`Xh,Mh,Rh`分别表示有限元空间(FE空间) X_h, M_h, R_h。为了使用有限元函数 $u_h, v_h \in X_h$, $p_h, q_h \in M_h$与$U_h, V_h \in R_h$ ，我们写：

```
  Xh uh,vh;
  Xph uph,vph;
  Mh ph,qh;
  Rh [Uxh,Uyh],[Vxh,Vyh];
  Xh[int] Uh(10);                                           //    Xh中10个函数的数组
  Rh[int] [Wxh,Wyh](10);                                    //    Rh中10个函数的数组
  Wxh[5](0.5,0.5)                                           //    在点(0.5,0.5)处的第6个函数
  Wxh[5][]                                                  //    第6个函数的自由度组成的数组
```

函数U_h, V_h有两个分量，因此我们有

$$U_h = |\begin{smallmatrix}Uxh\\Uyh\end{smallmatrix} \quad \text{与} \quad V_h = |\begin{smallmatrix}Vxh\\Vyh\end{smallmatrix}$$

6.2 “fespace” 在三维中的用法

对于三维有限元空间

$$X_h = \{v \in H^1([0,1]^3) | \ \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1\}$$
$$X_{ph} = \{v \in X_h | \ v(|\overset{0}{\cdot}) = v(|\overset{1}{\cdot}), v(|\dot{0}) = v(|\dot{1})\}$$
$$M_h = \{v \in H^1([0,1]^2) | \ \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2\}$$
$$R_h = \{\mathbf{v} \in H^1([0,1]^2)^2 | \ \forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K}(x,y) = |\begin{smallmatrix}\alpha_K\\ \beta_K\end{smallmatrix} + \gamma_K |\begin{smallmatrix}x\\ y\end{smallmatrix}\}$$

其中$\mathcal{T}_h$是单位立方体$]0,1[^2$中$10 \times 10 \times 10$的网格。 我们在FreeFem++ 中写：

```
mesh3 Th=buildlayers(square(10,10),10, zbound=[0,1]);
                                  //    标签:  0 上, 1 下; 2 前, 3 左, 4 后, 5:  右
fespace Xh(Th,P1);                                                    //    标量有限元
fespace Xph(Th,P1,
         periodic=[[0,x,y],[1,x,y],
                   [2,x,z],[4,x,z],
                   [3,y,z],[5,y,z]]);                  //   三周期有限元 (见注释 6.1)
fespace Mh(Th,P2);                                                    //    标量有限元
fespace Rh(Th,RT03d);                                                 //    向量有限元
```

其中Xh,Mh,Rh分别表示有限元空间(FE空间) X_h, M_h, R_h。 为了定义和使用FE-函数$u_h, v_h \in X_h$与$p_h, q_h \in M_h$与$U_h, V_h \in R_h$ ， 我们写：

```
  Xh uh,vh;
  Xph uph,vph;
  Mh ph,qh;
  Rh [Uxh,Uyh,Uyzh],[Vxh,Vyh, Vyzh];
  Xh[int] Uh(10);                                          //    Xh中10个函数组成的数组
  Rh[int] [Wxh,Wyh,Wzh](10);                               //    Rh中10个函数组成的数组.
  Wxh[5](0.5,0.5,0.5)                                  //    在点(0.5,0.5,0.5)处的第6个函数
  Wxh[5][]                                             //    第6个函数的自由度组成的数组
```

函数U_h, V_h具有三个分量，因此我们有

$$U_h = |\begin{smallmatrix}Uxh\\ Uyh\\ Uzh\end{smallmatrix} \quad 与 \quad V_h = |\begin{smallmatrix}Vxh\\ Vyh\\ Vzh\end{smallmatrix}$$

Note 6.1 一个关于周期性边界条件的很困难的问题是网格在等价面上应该一致，网格产生器*BuildLayer*会用穿过最大值顶点的对角线分割每一个四边形，因此为了确保每个周期对应的网格一样，在对应边的二维编号一定要是可兼容的（比如方差一样）。默认情况下，正方形顶点的编号是正确的。
我们可以使用*change*函数来改变网格编号，如：

```
{                                                                     //    擦除记忆..
int[int] old2new(0:Th.nv-1);                              //    0, 1, .., nv-1上的数组集
fespace Vh2(Th,P1);
Vh2 sorder=x+y;                          //    选取一个排列, 在4个方形边界上随着x或y增加
sort(sorder[],old2new);                                               //    建立反排列
int[int]  new2old=old2new^-1;                                         //    反转排列
Th= change(Th,renumv=new2old);change}
}
```

完整例子在*examples++-3d/periodic-3d.edp*中

6.3 拉格朗日有限元

6.3.1 P0-元

对于每个三角形（d=2）或者四面体（d=3） T_k，`Vh(Th,P0)`中的基函数ϕ_k为

$$\phi_k(\boldsymbol{x}) = 1 \text{ if } (\boldsymbol{x}) \in T_k, \qquad \phi_k(\boldsymbol{x}) = 0 \text{ if } (\boldsymbol{x}) \notin T_k$$

对于下述写法：

```
Vh(Th,P0);  Vh fh=f(x,y);
```

那么对于图6.1(a)中的顶点q^{k_i}, $i = 1, 2, ..d+1$，构造f_h：

$$\texttt{fh} = f_h(x,y) = \sum_k f(\frac{\sum_i q^{k_i}}{d+1})\phi_k$$

当网格为图6.2所示的$[-1,1]^2$中4×4网格时，$f(x,y) = \sin(\pi x)\cos(\pi y)$在`Vh(Th,P0)`上的投影如图6.3所示。

6.3.2 P1-元

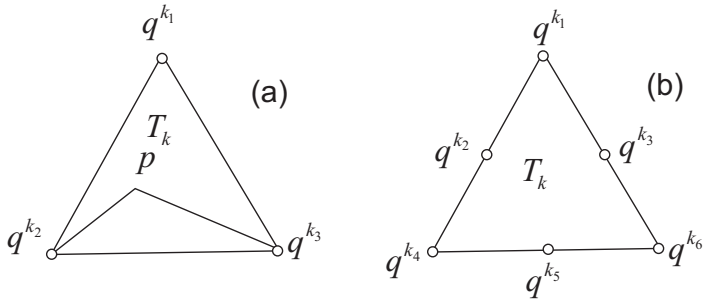


图 6.1: P_1与P_2在三角形T_k上的自由度

对于每个顶点q^i，`Vh(Th,P1)`中的基函数ϕ_i为：

$$\phi_i(x,y) = a_i^k + b_i^k x + c_i^k y \text{ for } (x,y) \in T_k,$$
$$\phi_i(q^i) = 1, \quad \phi_i(q^j) = 0 \text{ if } i \neq j$$

图6.1(a)中，在点$p = (x,y)$处，关于顶点q^{k_1}的基函数$\phi_{k_1}(x,y)$即等于重心坐标λ_1^k（面积坐标）：

$$\phi_{k_1}(x,y) = \lambda_1^k(x,y) = \frac{\text{三角形的面积}(p, q^{k_2}, q^{k_3})}{\text{三角形的面积}(q^{k_1}, q^{k_2}, q^{k_3})}$$

对于下述写法：

```
Vh(Th,P1); Vh fh=g(x.y);
```

那么

$$\texttt{fh} = f_h(x,y) = \sum_{i=1}^{n_v} f(q^i)\phi_i(x,y)$$

图6.4即为$f(x,y) = \sin(\pi x)\cos(\pi y)$在`Vh(Th,P1)`中的投影。

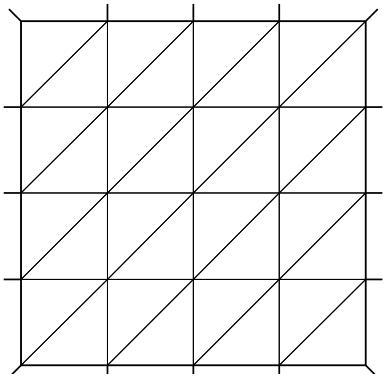


图 6.2: 投影所需的测试网格Th

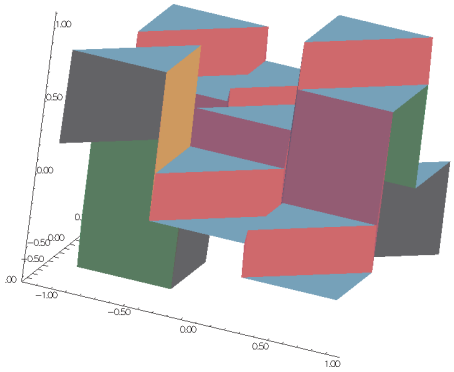


图 6.3: 在Vh(Th,P0)上的投影

6.3.3 P2-元

对于每一个顶点或者中点q^i，Vh(Th,P2)中的基函数ϕ_i为

$$\phi_i(x,y) = a_i^k + b_i^k x + c_i^k y + d_i^k x^2 + e_i^k xy + f_j^f y^2 \text{ for } (x,y) \in T_k,$$
$$\phi_i(q^i) = 1, \quad \phi_i(q^j) = 0 \text{ if } i \neq j$$

图6.1(b)中关于顶点q^{k_1}的基函数$\phi_{k_1}(x,y)$由重心坐标定义为：

$$\phi_{k_1}(x,y) = \lambda_1^k(x,y)(2\lambda_1^k(x,y) - 1)$$

且对于中点q^{k_2}

$$\phi_{k_2}(x,y) = 4\lambda_1^k(x,y)\lambda_4^k(x,y)$$

对于下述写法：

```
Vh(Th,P2); Vh fh=f(x.y);
```

那么

$$\texttt{fh} = f_h(x,y) = \sum_{i=1}^{M} f(q^i)\phi_i(x,y) \quad \text{(对于所有顶点和中点求和)}$$

图6.5即为$f(x,y) = \sin(\pi x)\cos(\pi y)$在 Vh(Th,P2)中的投影。

6.4 P1 非一致元

可参阅[23]以获得更多细节。简单来说，我们现在考虑非连续估计，因此我们会失去以下性质

$$w_h \in V_h \subset H^1(\Omega)$$

对于下述写法：

```
Vh(Th,P1nc); Vh fh=f(x.y);
```

那么

$$\texttt{fh} = f_h(x,y) = \sum_{i=1}^{n_v} f(m^i)\phi_i(x,y) \quad \text{(对于所有中点求和)}$$

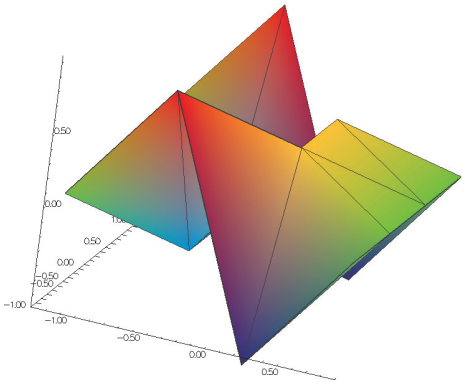


图 6.4: 在Vh(Th,P1)上的投影

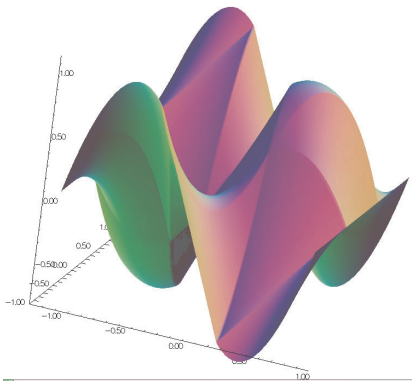


图 6.5: 在Vh(Th,P2)上的

这里ϕ_i是关于中点$m^i = (q^{k_i} + q^{k_{i+1}})/2$的基函数，其中$q^{k_i}$是$T_k$中第$i$ 个点，并且我们假设$j = 3$时$j+1 = 0$，则有：

$$\phi_i(x,y) = a_i^k + b_i^k x + c_i^k y \text{ for } (x,y) \in T_k,$$
$$\phi_i(m^i) = 1, \quad \phi_i(m^j) = 0 \text{ if } i \neq j$$

严格来说$\partial\phi_i/\partial x$, $\partial\phi_i/\partial y$包含Dirac分布$\rho\delta_{\partial T_k}$。 在数值计算中会自动忽视他们。 在[23]中，有关于这个估计的证明：

$$\left(\sum_{k=1}^{n_v}\int_{T_k}|\nabla w - \nabla w_h|^2 \mathrm{d}x\mathrm{d}y\right)^{1/2} = O(h)$$

基函数ϕ_k具有以下特征：

1. 对于(2.6)中定义的双线性形式a，满足

$$a(\phi_i,\phi_i) > 0, \qquad a(\phi_i,\phi_j) \leq 0 \quad \text{if } i \neq j$$
$$\sum_{k=1}^{n_v} a(\phi_i,\phi_k) \geq 0$$

2. $f \geq 0 \Rightarrow u_h \geq 0$
3. 若 $i \neq j$，基函数ϕ_i与ϕ_j是L^2-正交的：

$$\int_\Omega \phi_i\phi_j \,\mathrm{d}x\mathrm{d}y = 0 \qquad \text{if } i \neq j$$

这个对于P_1-元是不成立的。

图6.6即为$f(x,y) = \sin(\pi x)\cos(\pi y)$在 Vh(Th,**P1nc**) 中的投影。

6.5 其他的FE-空间

对于三角形$T_k \in \mathcal{T}_h$， 设$\lambda_{k_1}(x,y)$, $\lambda_{k_2}(x,y)$, $\lambda_{k_3}(x,y)$三角形的区域坐标（见图6.1），且将

$$\beta_k(x,y) = 27\lambda_{k_1}(x,y)\lambda_{k_2}(x,y)\lambda_{k_3}(x,y) \tag{6.23}$$

称为T_k上的*bubble*函数。 bubble函数具有以下性质：

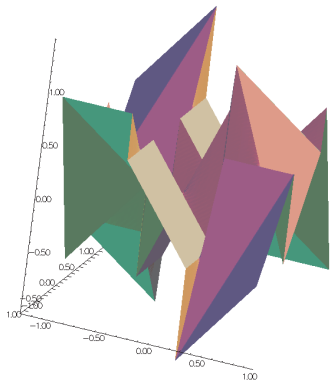


图 6.6: 在Vh(Th,P1nc)上的投影

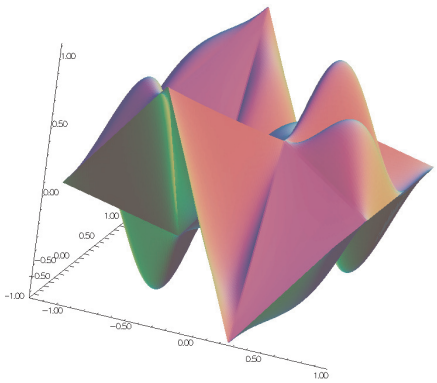


图 6.7: 在Vh(Th,P1b)上的投影

1. $\beta_k(x,y)=0$ 若 $(x,y)\in\partial T_k$.
2. $\beta_k(q^{k_b})=1$ 当q^{k_b}是重心 $\frac{q^{k_1}+q^{k_2}+q^{k_3}}{3}$.

对于下述写法：

```
Vh(Th,P1b); Vh fh=f(x.y);
```

那么

$$\texttt{fh}=f_h(x,y)=\sum_{i=1}^{n_v}f(q^i)\phi_i(x,y)+\sum_{k=1}^{n_t}f(q^{k_b})\beta_k(x,y)$$

图6.7即为$f(x,y)=\sin(\pi x)\cos(\pi y)$在 `Vh(Th,P1b)` 中的投影。

6.6 向量型FE-函数

从$\mathbb{R}^2$到$\mathbb{R}^N$的函数在$N=1$时被称为标量函数，在 $N>1$时被称为向量函数。 当$N=2$时

```
fespace Vh(Th,[P0,P1]) ;
```

构造空间

$$V_h=\{\mathbf{w}=(w_1,w_2)|\ w_1\in V_h(\mathcal{T}_h,P_0),\ w_2\in V_h(\mathcal{T}_h,P_1)\}$$

6.6.1 Raviart-Thomas 元

在Raviart-Thomas有限元 $RT0_h$中， 自由度是通过网格上边e的通量，其中函数$\mathbf{f}:\mathbb{R}^2\longrightarrow\mathbb{R}^2$的通量是$\int_e\mathbf{f}.n_e$， n_e是边e的单位法向量。
这里隐含了网格中所有边的方向，比如我们能对边的端点进行全局编号，然后从小的值指向到大的值。
为了计算通量，我们使用一个高斯点即边的中点来求积。考虑三角形T_k，它的三个顶点为$(\mathbf{a},\mathbf{b},\mathbf{c})$。我们令顶点的编号为$i_a,i_b,i_c$，且定义三条边$\mathbf{e}^1,\mathbf{e}^2,\mathbf{e}^3$ 为$sgn(i_b-i_c)(\mathbf{b}-\mathbf{c})$,$sgn(i_c-i_a)(\mathbf{c}-\mathbf{a})$, $sgn(i_a-i_b)(\mathbf{a}-\mathbf{b})$,
我们得到基函数：

$$\boldsymbol{\phi}_1^k=\frac{sgn(i_b-i_c)}{2|T_k|}(\mathbf{x}-\mathbf{a}),\quad \boldsymbol{\phi}_2^k=\frac{sgn(i_c-i_a)}{2|T_k|}(\mathbf{x}-\mathbf{b}),\quad \boldsymbol{\phi}_3^k=\frac{sgn(i_a-i_b)}{2|T_k|}(\mathbf{x}-\mathbf{c}),\qquad(6.24)$$

其中$|T_k|$是三角形T_k的面积。 如果我们写

```
Vh(Th,RT0); Vh [f1h,f2h]=[f1(x.y),f2(x,y)];
```

那么

$$\texttt{fh} = \boldsymbol{f}_h(x,y) = \sum_{k=1}^{n_t}\sum_{l=1}^{6} n_{i_l j_l}|\mathbf{e^{i_1}}|f_{j_l}(m^{i_l})\phi_{i_l j_l}$$

其中$n_{i_l j_l}$是法向量$\boldsymbol{n}_{i_l}$的第j_l个分量，

$$\{m_1, m_2, m_3\} = \left\{\frac{\mathbf{b}+\mathbf{c}}{2}, \frac{\mathbf{a}+\mathbf{c}}{2}, \frac{\mathbf{b}+\mathbf{a}}{2}\right\}$$

且关于l的顺序 $i_l = \{1,1,2,2,3,3\}$, $j_l = \{1,2,1,2,1,2\}$。

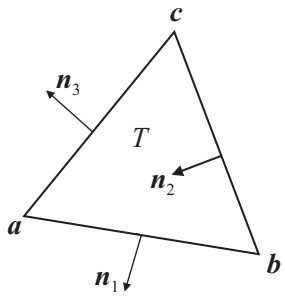


图 6.8: 每条边的法向量

Example 6.1

```
mesh Th=square(2,2);
fespace Xh(Th,P1);
fespace Vh(Th,RT0);
Xh uh,vh;
Vh [Uxh,Uyh];
[Uxh,Uyh] = [sin(x),cos(y)];                    //    向量有限元函数
vh= x^2+y^2;                                    //    vh
Th = square(5,5);                               //    改变网格
                                                //    Xh未改变
uh = x^2+y^2;                                   //    在新的Xh上计算
Uxh = x;                          //    错误：不可能只设置向量有限元函数的一个分量
vh = Uxh;                                       //    ok
                                                //    现在 uh 定义在 5x5 网格上
                                                //    但vh的有限元空间一直在 2x2 网格上
plot(uh,ps="onoldmesh.eps");                    //    图 6.9
uh = uh;                                        //    在 5x5 网格上对原来的 vh 做插值
                                                //    在 10x10 网格上得到新的vh.
plot(uh,ps="onnewmesh.eps");                    //    图 6.10
vh([x-1/2,y])= x^2 + y^2;                       //    插值 vh = ((x − 1/2)^2 + y^2)
```

为了得到FE函数uh在点$x = 1, y = 2$处的值，即[Uxh,Uyh]，可采用如下写法：

```
real value;
value = uh(2,4);                                //    得到 value= uh(2,4)
```

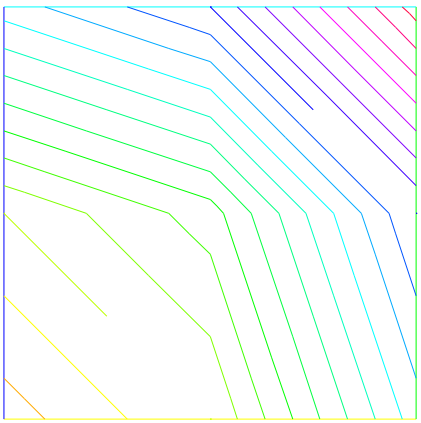


图 6.9: vh Iso 在2 × 2的网格上

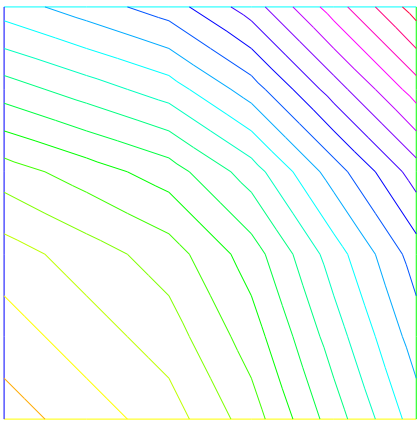


图 6.10: vh Iso 在5 × 5的网格上

```
value = Uxh(2,4);                                    //    得到 value= Uxh(2,4)
                                                     //    ------ or ------
x=1;y=2;
value = uh;                                          //    得到 value= uh(1,2)
value = Uxh;                                         //    得到 value= Uxh(1,2)
value = Uyh;                                         //    得到 value= Uyh(1,2).
```

为了得到FE函数相应的列向量uh，可采用如下写法：

```
real value = uh[][0] ;                               //    得到0自由度的值
real maxdf = uh[].max;                               //    自由度的最大值
int size = uh.n;                                     //    自由度个数
real[int] array(uh.n)= uh[];                         //    复值函数 uh 的数组
```

Note 6.2 对于一个非标量函数*[Uxh,Uyh]*，两个列向量*Uxh[]*与*Uyh[]*是一样的，因为自由度能涉及一个以上的分量。

6.7 快速有限元插值

在实际中，我们需要用有限元方法来离散变分方程。若对于域Ω_1有一个网格，对于域Ω_2 是另一个网格。计算定义在不同网格上的函数乘积的积分是困难的，这时在求积点需要用到从一个网格到另一个网格的求积公式和插值。下面我们展示目前可使用的最新插值算子。 设

$\mathcal{T}_h^0 = \cup_k T_k^0, \mathcal{T}_h^1 = \cup_k T_k^1$ 是区域 Ω的两个三角形划剖分。 再设

$$V(\mathcal{T}_h^i) = \{C^0(\Omega_h^i) \ : \ f|_{T_k^i} \in P_0\}, \quad i = 0, 1$$

是每个三角形上连续分段仿射函数组成的空间。
令$f \in V(\mathcal{T}_h^0)$。问题在于找到$g \in V(\mathcal{T}_h^1)$使得

$$g(q) = f(q) \quad \forall q \text{ vertex of } \mathcal{T}_h^1$$

尽管这个问题看起来很简单，但实际上却很难找到一个有效的算法。 我们提出一个复杂度是$N^1 \log N^0$的算法，其中N^i是$\mathcal{T}_h^i$顶点的个数，这个算法在大多数实际2D应用时都很快速。

算法
这个方法分为5步。 一开始构造一个四叉树，包括网格 $\mathcal{T}_h^0$的所有顶点，使得在每一个终止格子中$\mathcal{T}_h^0$的顶点都最少有一个，最多有四个。
对于每一个q^1，$\mathcal{T}_h^1$的顶点作如下操作：

第一步 找出包含q^1的四叉树的终止格。

第二步 找出该格中离q^1最近的q_j^0。

第三步 找出一个以q_j^0为顶点的三角形。

第四步 找出重心坐标 $\{\lambda_j\}_{j=1,2,3}$ of q^1 in T_k^0。

- – 如果所有的重心坐标都是正的，跳到第五步。
- – 若存在一个负的重心坐标λ_i，用对应q_i^0的邻近三角形替换，然后跳到第四步。
- – 若存在两个负的重心坐标，随机取其中一个，和上面一样替换掉T_k^0。

第五步 通过f的线性插值计算在T_k^0上的$g(q^1)$：

$$g(q^1) = \sum_{j=1,2,3} \lambda_j f(q_j^0)$$

End

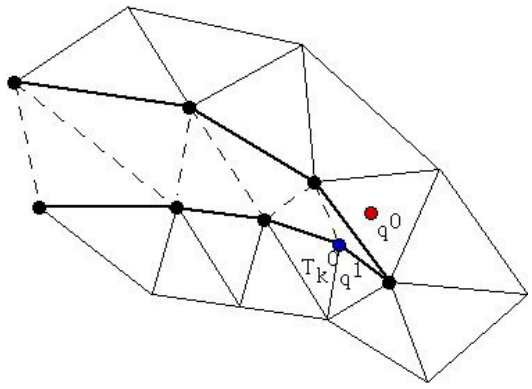


图 6.11: 为了在q^0点插值，需要包含q^0的三角形的信息。这个算法可能从$q^1 \in T_k^0$开始，且停滞在边界（粗线），因为线 q^0q^1不在Ω之中。但是如果孔也是三角形的（点线），那么这个问题就不会出现。

有两个问题需要解决：

- 如果q^1不在Ω_h^0中？ 那么第五步会停止在边界三角形上。所以我们加一步，测试q^1到两个邻近边界的距离并选择最近的一个，以此类推直到距离增长。

- 如果Ω_h^0非凸且第四步锁定在了边界上？ 通过构建Delaunay-Voronoï网格永远能将区域顶点组成的凸包三角形化。所以我们确保这个信息不会在构建$\mathcal{T}_h^0, \mathcal{T}_h^1$时丢失，并且我们将区域外的三角形保持在特殊列表里。因此在第五步中，如有需要我们可以参照列表跳过孔。

Note 6.3 有时在特殊情况下，插值过程会错过某些点，可以通过全局变量 *searchMethod*来改变搜索算法。

```
searchMethod=0;                                          //    快速搜索算法的默认值
searchMethod=1;                          //    安全搜索算法，在错过点的情况下，使用强制手段
                                  //    （警告：当点在区域外时，这种方法很耗费运算量）
searchMethod=2;                                  //    总是使用强制手段，非常耗费运算量
```

Note 6.4 第三步需要一列指针，使得每一个顶点指向划分中的一个三角形。

Note 6.5 运算符= 是 *FreeFem++* 中的插值符号，连续有限函数通过连续性被延拓到原区域以外。
看请下面的例子。

```
mesh Ths= square(10,10);
mesh Thg= square(30,30,[x*3-1,y*3-1]);
plot(Ths,Thg,ps="overlapTh.eps",wait=1);
fespace Ch(Ths,P2); fespace Dh(Ths,P2dc);
fespace Fh(Thg,P2dc);
Ch us= (x-0.5)*(y-0.5);
Dh vs= (x-0.5)*(y-0.5);
Fh ug=us,vg=vs;
plot(us,ug,wait=1,ps="us-ug.eps");                       //    如图 6.12
plot(vs,vg,wait=1,ps="vs-vg.eps");                       //    如图 6.13
```

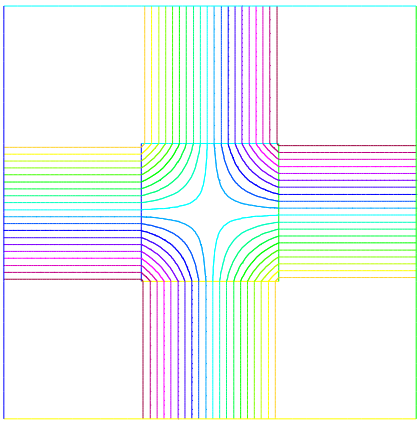


图 6.12: 连续的FE函数的延拓

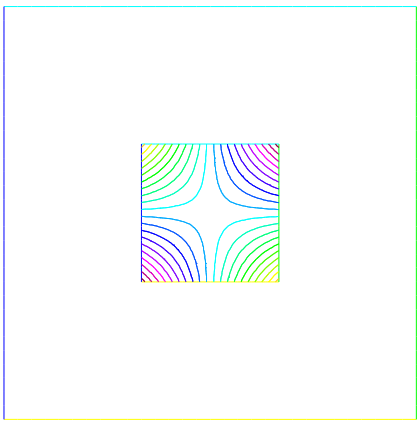


图 6.13: 不连续的FE函数的延拓, 参照警告 6

6.8 关键词：Problem 与 Solve

在FreeFem++ 中，问题由变分形式给出，
因此，我们需要一个双线性映射$a(u,v)$ ，一个线性映射 $\ell(f,v)$，
可能还要加上一个边界条件。

```
problem P(u,v) =
      a(u,v) - ℓ(f,v)
      + (边界条件);
```

Note 6.6 当你想通过公式表述问题的同时去求解，那可以用到关键词*solve*。

6.8.1 弱形式与边界条件

为了展现变分公式（亦或偏微分方程的弱形式）的原理，我们举一个常见的问题来说明：具有 Dirichlet 与 Robin 边界条件的Poisson 方程。

.
问题：找一个定义在$\mathbb{R}^d$ $(d=2,3)$区域Ω 上的函数u，使得以下方程成立：

$$-\nabla.(\kappa\nabla u)=f,\quad \text{in}\quad \Omega,\quad au+\kappa\frac{\partial u}{\partial n}=b\quad \text{on}\quad \Gamma_r,\quad u=g\quad \text{on}\quad \Gamma_d \tag{6.25}$$

其中

- 若 $d=2$ ，那么 $\nabla.(\kappa\nabla u)=\partial_x(\kappa\partial_x u)+\partial_y(\kappa\partial_y u)$ 并且 $\partial_x u=\frac{\partial u}{\partial x}$ $\partial_y u=\frac{\partial u}{\partial y}$
- 若 $d=3$ 那么 $\nabla.(\kappa\nabla u)=\partial_x(\kappa\partial_x u)+\partial_y(\kappa\partial_y u)+\partial_z(\kappa\partial_z u)$ 并且 $\partial_x u=\frac{\partial u}{\partial x}$, $\partial_y u=\frac{\partial u}{\partial y}$, $\partial_z u=\frac{\partial u}{\partial z}$
- 边界 $\Gamma=\partial\Omega$ 被分割成不相交的两部分： Γ_d 与 Γ_n 即： $\Gamma_d\cap\Gamma_n=\emptyset$ 并且 $\Gamma_d\cup\Gamma_n=\partial\Omega$,
- κ 是一个给定的正值函数, 使得 $\exists\kappa_0\in\mathbb{R},\quad 0<\kappa_0\le\kappa$
- a 是一个给定的非负函数
- b 是一个给定的函数

Note 6.7 若 $a=0$，且Γ_d为空集，则以上即为常见的*Neumann*边界条件。 在此条件下，只有函数的导数值在问题中出现了，所以方程的解相差一个常数值。（即，若u为一个解，则$u+c$ 也是一个解。）

记v是一个在Γ_d上取值为0 的正则测试函数。由分部积分，我们有：

$$-\int_\Omega \nabla.(\kappa\nabla u)\,v\,d\omega=\int_\Omega \kappa\nabla v.\nabla u\,d\omega-\int_\Gamma v\kappa\frac{\partial u}{\partial \boldsymbol{n}}\,d\gamma,=\int_\Omega fv\,d\omega \tag{6.26}$$

其中，若 $d=2$ 则有 $\nabla v.\nabla u=(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\frac{\partial v}{\partial y})$, 若 $d=3$ 则有 $\nabla v.\nabla u=(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\frac{\partial v}{\partial y}+\frac{\partial u}{\partial z}\frac{\partial v}{\partial z})$, $\boldsymbol{n}$ 为$\partial\Omega$的单位外法向量。 现在，我们有：在 Γ_r 上，$\kappa\frac{\partial u}{\partial n}=-au+g$ ；在 Γ_d 上， $v=0$，其中 $\partial\Omega=\Gamma_d\cup\Gamma_n$ 即

$$-\int_{\partial\Omega} v\kappa\frac{\partial u}{\partial n}=\int_{\Gamma_r} auv-\int_{\Gamma_r} bv$$

那么，问题转化为： 找 $u \in V_g = \{v \in H^1(\Omega)/v = g \text{ on } \Gamma_d\}$ 使得

$$\int_\Omega \kappa \nabla v.\nabla u \, d\omega + \int_{\Gamma_r} auv \, d\gamma = \int_\Omega fv \, d\omega + \int_{\Gamma_r} bv \, d\gamma, \quad \forall v \in V_0 \tag{6.27}$$

其中 $V_0 = \{v \in H^1(\Omega)/v = 0 \text{ on } \Gamma_d\}$
除了在Neumann条件下，当$\kappa \geq \kappa_0 > 0$时，问题 (6.27) 是适定的。

Note 6.8 若我们只有Neumann边界条件，由线性代数的知识可以知道，只有在等式右边部分与算子的核正交时，方程的解存在。这一相容性条件的一种表述方式为：

$$\int_\Omega f \, d\omega + \int_\Gamma b \, d\gamma = 0$$

而求出该恒等式的一种方式就是求解 $u \in H^1(\Omega)$ 使得:

$$\int_\Omega \varepsilon uv \ d\omega \ + \ \kappa \nabla v.\nabla u \, d\omega = \int_\Omega fv \, d\omega + \int_{\Gamma_r} bv \, d\gamma, \quad \forall v \in H^1(\Omega) \tag{6.28}$$

其中 ε 是一个很小的参数 （ $\sim 10^{-10}$ ）。
注意到，若解是$\frac{1}{\varepsilon}$阶的，那么相容性条件是不满足的。另外，我们的解有 $\int_\Omega u = 0$，那么读者也可以通过添加Lagrange乘子去求解现实的数学问题。
（例： `examples++-tutorial/Laplace-lagrange-mult.edp` ）

在 `FreeFem++`中， 二元问题 (6.27) 变为

```
problem Pw(u,v) =
     int2d(Th)( kappa*( dx(u)*dx(u) +  dy(u)*dy(u)) )  //   ∫_Ω κ∇v.∇u dω
   + int1d(Th,gn)( a * u*v )                           //   ∫_Γr auv dγ
   - int2d(Th)(f*v)                                    //   ∫_Ω fv dω
   - int1d(Th,gn)( b * v )                             //   ∫_Γr bv dγ
   + on(gd)(u= g) ;                                    //   u = g  on  Γ_d
```

其中 `Th` 是二元区域Ω中的一个网格， `gd` 和`gn`分别代表边界Γ_d 和 Γ_n。
三维问题 （6.27）转换为

```
macro Grad(u) [dx(u),dy(u),dz(u) ]                  //   定义宏：三维的梯度函数（Grad）
%EOM :  definition of the 3d Grad macro
problem Pw(u,v) =
     int3d(Th)( kappa*( Grad(u)'*Grad(v) ) )  //   ∫_Ω κ∇v.∇u dω
   + int2d(Th,gn)( a * u*v )                           //   ∫_Γr auv dγ
   - int3d(Th)(f*v)                                    //   ∫_Ω fv dω
   - int2d(Th,gn)( b * v )                             //   ∫_Γr bv dγ
   + on(gd)(u= g) ;                                    //   u = g  on  Γ_d
```

其中其中 `Th` 是三维区域Ω中的一个网格， `gd` 和`gn`分别代表边界Γ_d 和 Γ_n。

6.9 参数对 `solve` 及 `problem` 的影响

参数包括：FE函数是实函数或复函数，参数的个数 n 是偶数($n = 2*k$)，前k 个函数的参数未知，后k个函数为测试函数。

Note 6.9 若这些函数是向量的*FE*，那么我们必须要让所有的函数都具有与向量*FE*相同的阶。（例如：*laplaceMixte*问题）

Note 6.10 不要混淆实的和复的*FE*函数。

Bug: 1 具有不同周期边界条件的 *fespace*是不可混淆。所以在一个问题中的所有测试函数和未知函数都必须具有相同类型的周期边界条件或具有非周期边界条件。（程序暂时不能对这个问题给出明确的错误指示信息，只会显示“未知错误”，抱歉。）

这些参数为:

solver= LU, CG, Crout,Cholesky,GMRES,sparsesolver, UMFPACK ...

系统默认值 solver 是 sparsesolver （在没有其它sparse solver的定义下等同于 UMFPACK）或被设定为 LU （若没有合适的sparse solver可用。） 潜在线性系统的矩阵的存储方式依赖于solver形式的选取。对于LU，矩阵是sky-line 非对称的；对于Crout，矩阵是sky-line对称的；对于 Cholesky，矩阵是sky-line、对称且正定的；CG，矩阵是稀疏、对称且正定的；而对于 GMRES, sparsesolver 或 UMFPACK，矩阵仅仅只是稀疏的。

eps= 实表达式。 ε 为迭代方法（例：CG）的停止条件。注意到，若ε是负的，那么停止条件变为：

$$||Ax - b|| < |\varepsilon|$$

若它正的，那么停止条件为：

$$||Ax - b|| < \frac{|\varepsilon|}{||Ax_0 - b||}$$

init= Boolean 表达式，若为0，则矩阵重构。注意到，若网格变化了，那么矩阵也要重构。

precon= 预处理工具的函数的名称。（如：P） P函数的原型为：

```
func real[int]  P(real[int] & xx) ;
```

tgv= 充分大的数 (10^{30}) 用以实现 Dirichlet边界条件。说明见 161 页。

tolpivot= 设定UMFPACK (10^{-1})的中心误差，以及 LU、 Crout、Cholesky 分解 (10^{-20})的误差。

tolpivotsym= 设定UMFPACK系统的中心误差。

strategy= 设置UMFPACK的整体策略 （0 为默认值）。

6.10 问题描述

在本节中，v为未知函数，w为测试函数。
在 ”=”的一侧，是以下参数的和：

- 标识符；前述变量形式（类型：(type varf)）的统称，以备反复应用。 注意到，“varf”下， 未知的测试函数的名称已被遗忘，我们正好通过参数列表中的这个命令，像C++ 中的那样，去回忆起函数名。见note 6.15
- 双线性型项本身：若K是一个给定的函数。
- 对于三维Th的双线性部分：

- `int3d(Th)( K*v*w)` $= \sum_{T\in\mathtt{Th}} \int_T K\,v\,w$
- `int3d(Th,1)( K*v*w)` $= \sum_{T\in\mathtt{Th},T\subset\Omega_1} \int_T K\,v\,w$
- `int2d(Th)( K*v*w)` $= \sum_{T\in\mathtt{Th}} \int_T K\,v\,w$
- `int2d(Th,1)( K*v*w)` $= \sum_{T\in\mathtt{Th},T\subset\Omega_1} \int_T K\,v\,w$
- `int2d(Th,2,5)( K*v*w)` $= \sum_{T\in\mathtt{Th}} \int_{(\partial T\cup\Gamma)\cap(\Gamma_2\cup\Gamma_5)} K\,v\,w$
- `int2d(Th,levelset=phi)( K*v*w)` $= \sum_{T\in\mathtt{Th}} \int_{T,\phi=0} K\,v\,w$
- `int2d(Th,1,levelset=phi)( K*v*w)` $= \sum_{T\in\mathtt{Th},T\subset\Omega_l} \int_{T,\phi=0} K\,v\,w$
- `intalledges(Th)( K*v*w)` $= \sum_{T\in\mathtt{Th},T\subset\Omega_l} \int_{\partial T} K\,v\,w$
- `intalledges(Th,1)( K*v*w)` $= \sum_{T\in\mathtt{Th},T\subset\Omega_1} \int_{\partial T} K\,v\,w$
- 它们对由`FreeFem++` 构造的`matrix`类型的稀疏矩阵有作用，无论这些矩阵是否明确声明是由`FreeFem++` 构造的。

- 对于二维Th的双线性部分：
  - `int2d(Th)( K*v*w)` $= \sum_{T\in\mathtt{Th}} \int_T K\,v\,w$
  - `int2d(Th,1)( K*v*w)` $= \sum_{T\in\mathtt{Th},T\subset\Omega_1} \int_T K\,v\,w$
  - `int1d(Th)( K*v*w)` $= \sum_{T\in\mathtt{Th}} \int_T K\,v\,w$
  - `int1d(Th,1)( K*v*w)` $= \sum_{T\in\mathtt{Th},T\subset\Omega_1} \int_T K\,v\,w$
  - `int1d(Th,2,5)( K*v*w)` $= \sum_{T\in\mathtt{Th}} \int_{(\partial T\cup\Gamma)\cap(\Gamma_2\cup\Gamma_5)} K\,v\,w$
  - `int1d(Th,levelset=phi)( K*v*w)` $= \sum_{T\in\mathtt{Th}} \int_{T,\phi=0} K\,v\,w$
  - `int1d(Th,1,levelset=phi)( K*v*w)` $= \sum_{T\in\mathtt{Th},T\subset\Omega_l} \int_{T,\phi=0} K\,v\,w$
  - `intalledges(Th)( K*v*w)` $= \sum_{T\in\mathtt{Th},T\subset\Omega_l} \int_{\partial T} K\,v\,w$

- `intalledges(Th,1)( K*v*w)` $= \sum_{T\in\mathtt{Th},T\subset\Omega_1}\int_{\partial T} K\,v\,w$
- 它们对由`FreeFem++` 构造的`matrix`类型的稀疏矩阵有作用，无论这些矩阵是否明确声明是由`FreeFem++` 构造的。

- 3维PDE的右端，线性形式项：对于给定的 K, f：
  - `int3d(Th)( K*w)` $= \sum_{T\in\mathtt{Th}}\int_T K\,w$
  - `int3d(Th)( K*w)` $= \sum_{T\in\mathtt{Th}}\int_T K\,w$
  - `int2d(Th,2,5)( K*w)` $= \sum_{T\in\mathtt{Th}}\int_{(\partial T\cup\Gamma)\cap(\Gamma_2\cup\Gamma_5)} K\,w$
  - `intalledges(Th)( f*w)` $= \sum_{T\in\mathtt{Th}}\int_{\partial T} f\,w$
  - `real[int]` 类型的向量

- 2维PDE的右端，线性形式项：对于给定的 K, f：
  - `int2d(Th)( K*w)` $= \sum_{T\in\mathtt{Th}}\int_T K\,w$
  - `int2d(Th)( K*w)` $= \sum_{T\in\mathtt{Th}}\int_T K\,w$
  - `int1d(Th,2,5)( K*w)` $= \sum_{T\in\mathtt{Th}}\int_{(\partial T\cup\Gamma)\cap(\Gamma_2\cup\Gamma_5)} K\,w$
  - `intalledges(Th)( f*w)` $= \sum_{T\in\mathtt{Th}}\int_{\partial T} f\,w$
  - `real[int]` 类型的向量

- 边界条件项 :
  - 一个“on”标量形式（Dirichlet条件） `on(1, u = g )`
    即，对于标为"1"的边界上的所有自由度 i，矩阵的对角项$a_{ii} = tgv$ （ *terrible geant value* `tgv` (=10^{30}， 系统默认值)） ，右端项$b[i] =$ "$(\Pi_h g)[i]$" $\times\, tgv$,其中，"$(\Pi_h g)g[i]$"为插入函数 g的边界插入节点值。 注意到，若tgv < 0，我们把矩阵第 i行上除了对角项的所有项变为 0， $a_{ii} = 1$， $b[i] =$ "$(\Pi_h g)[i]$" （3.10版以上）
  - 一个“on”向量形式（Dirichlet条件） : `on(1,u1=g1,u2=g2)` 对于向量有限元，如`RT0`，向量的两个分量是耦合的，那么有： $b[i] =$ "$(\Pi_h(g1, g2))[i]$" $\times\, tgv$, 其中 Π_h 为向量有限元插值。
  - Γ 上的线性项（2维的Neumann条件） `-int1d(Th))( f*w)` 或 `-int1d(Th,3))( f*w)`
  - Γ 或 Γ_2 上的双线性型(2维的Robin条件) `int1d(Th))( K*v*w)` or `int1d(Th,2))( K*v*w)`
  - Γ 上的线性项（3维的Neumann条件） `-int2d(Th))( f*w)` 或 `-int2d(Th,3))( f*w)`

– Γ 或 Γ_2 上的双线性型(3维的Robin条件) `int2d(Th))( K*v*w)` 或 `int2d(Th,2))( K*v*w)`

Note 6.11

- 若需要，不同种类的项在合式中可以出现超过一次。
- 先线性形式下，积分网格可以和与测试函数或未知函数关联的网格不同
- `N.x, N.y` 和 `N.z` 都是法向分量。

重点: 线性部分和双线性部分不可以写在同一个积分下，如 `int1d(Th)( K*v*w - f*w)` 。

6.11 数值积分

记D是N维有界区域。对于任意的r阶多项式f，若能找到特殊的（正交）向量 $\boldsymbol{\xi}_j$, $j = 1, \cdots, J$ 在D中，以及（求积）常数ω_j，使得

$$\int_D f(\boldsymbol{x}) = \sum_{\ell=1}^{L} c_\ell f(\boldsymbol{\xi}_\ell) \tag{6.29}$$

由此，可以引出误差估计式（参考Crouzeix-Mignot (1984)），那么存在一个常数$C > 0$使得，

$$\left| \int_D f(\boldsymbol{x}) - \sum_{\ell=1}^{L} \omega_\ell f(\boldsymbol{\xi}_\ell) \right| \leq C|D|h^{r+1} \tag{6.30}$$

对于D上任意$r+1$阶连续函数f,记 D的直径为h，$|D|$为它的大小。（线段$[q^i q^j]$ 上的一点定义为

$$\{(x,y)|\ x = (1-t)q_x^i + tq_x^j,\ y = (1-t)q_y^i + tq_y^j,\ 0 \leq t \leq 1\}"$$

对于区域 $\Omega_h = \sum_{k=1}^{n_t} T_k$, $\mathcal{T}_h = \{T_k\}$, 可以通过如下方式计算在$\Gamma_h = \partial\Omega_h$上的积分

$$\begin{aligned} \int_{\Gamma_h} f(\boldsymbol{x})ds &= \texttt{int1d(Th)(f)} \\ &= \texttt{int1d(Th,qfe=*)(f)} \\ &= \texttt{int1d(Th,qforder=*)(f)} \end{aligned}$$

其中 * 为求积公式的名称或Gauss求积公式的阶。

边界上的求积公式					
L	(qfe=)	qforder=	point in $[q^i q^j](=t)$	ω_ℓ	exact on $P_k, k=$
1	qf1pE	2	$1/2$	$\lvert q^i q^j\rvert$	1
2	qf2pE	3	$(1 \pm \sqrt{1/3})/2$	$\lvert q^i q^j\rvert/2$	3
3	**qf3pE**	6	$(1 \pm \sqrt{3/5})/2$ $1/2$	$(5/18)\lvert q^i q^j\rvert$ $(8/18)\lvert q^i q^j\rvert$	5
4	qf4pE	8	$(1 \pm \frac{\sqrt{525+70\sqrt{30}}}{35})/2.$ $(1 \pm \frac{\sqrt{525-70\sqrt{30}}}{35})/2.$	$\frac{18-\sqrt{30}}{72}\lvert q^i q^j\rvert$ $\frac{18+\sqrt{30}}{72}\lvert q^i q^j\rvert$	7
5	qf5pE	10	$(1 \pm \frac{\sqrt{245+14\sqrt{70}}}{21})/2$ $1/2$ $(1 \pm \frac{\sqrt{245-14\sqrt{70}}}{21})/2$	$\frac{322-13\sqrt{70}}{1800}\lvert q^i q^j\rvert$ $\frac{64}{225}\lvert q^i q^j\rvert$ $\frac{322+13\sqrt{70}}{1800}\lvert q^i q^j\rvert$	9
2	qf1pElump	2	0 $+1$	$\lvert q^i q^j\rvert/2$ $\lvert q^i q^j\rvert/2$	1

其中 $|q^iq^j|$ 为线段 $\overline{q^iq^j}$的长度。 根据表格“1”，对于 Γ_h中的一部分Γ_1，可以通过如下方式计算在Γ_1上的积分

$$\begin{aligned}\int_{\Gamma_1} f(x,y)ds &= \texttt{int1d(Th,1)(f)}\\ &= \texttt{int1d(Th,1,qfe=qf2pE)(f)}\end{aligned}$$

Γ_1, Γ_3上的积分为：

$$\int_{\Gamma_1\cup\Gamma_3} f(x,y)ds = \texttt{int1d(Th,1,3)(f)}$$

对于每个三角区域 $T_k = [q^{k_1}q^{k_2}q^{k_3}]$， T_k中的点$P(x,y)$，由面积坐标（*area coordinate*）变换表示成$P(\xi,\eta)$：

$$|T_k| = \frac{1}{2}\begin{vmatrix}1 & q_x^{k_1} & q_y^{k_1}\\ 1 & q_x^{k_2} & q_y^{k_2}\\ 1 & q_x^{k_3} & q_y^{k_3}\end{vmatrix} \quad D_1 = \begin{vmatrix}1 & x & y\\ 1 & q_x^{k_2} & q_y^{k_2}\\ 1 & q_x^{k_3} & q_y^{k_3}\end{vmatrix} \quad D_2 = \begin{vmatrix}1 & q_x^{k_1} & q_y^{k_1}\\ 1 & x & y\\ 1 & q_x^{k_3} & q_y^{k_3}\end{vmatrix} \quad D_3 = \begin{vmatrix}1 & q_x^{k_1} & q_y^{k_1}\\ 1 & q_x^{k_2} & q_y^{k_2}\\ 1 & x & y\end{vmatrix}$$

$$\xi = \frac{1}{2}D_1/|T_k| \qquad \eta = \frac{1}{2}D_2/|T_k| \qquad \text{then } 1-\xi-\eta = \frac{1}{2}D_3/|T_k|$$

对于二维区域或者三维空间的边界区域 $\Omega_h = \sum_{k=1}^{n_t} T_k$, $\mathcal{T}_h = \{T_k\}$, 可以通过如下方式计算在Ω_h上的积分

$$\begin{aligned}\int_{\Omega_h} f(x,y) &= \texttt{int2d(Th)(f)}\\ &= \texttt{int2d(Th,qft=*)(f)}\\ &= \texttt{int2d(Th,qforder=*)(f)}\end{aligned}$$

其中 * 为求积公式的名称或Gauss求积公式的阶。

三角区域上的求积公式					
L	qft=	qforder=	point in T_k	ω_ℓ	exact on $P_k, k=$
1	qf1pT	2	$\left(\frac{1}{3},\frac{1}{3}\right)$	$\lvert T_k\rvert$	1
3	qf2pT	3	$\left(\frac{1}{2},\frac{1}{2}\right)$ $\left(\frac{1}{2},0\right)$ $\left(0,\frac{1}{2}\right)$	$\lvert T_k\rvert/3$ $\lvert T_k\rvert/3$ $\lvert T_k\rvert/3$	2
7	**qf5pT**	6	$\left(\frac{1}{3},\frac{1}{3}\right)$ $\left(\frac{6-\sqrt{15}}{21},\frac{6-\sqrt{15}}{21}\right)$ $\left(\frac{6-\sqrt{15}}{21},\frac{9+2\sqrt{15}}{21}\right)$ $\left(\frac{9+2\sqrt{15}}{21},\frac{6-\sqrt{15}}{21}\right)$ $\left(\frac{6+\sqrt{15}}{21},\frac{6+\sqrt{15}}{21}\right)$ $\left(\frac{6+\sqrt{15}}{21},\frac{9-2\sqrt{15}}{21}\right)$ $\left(\frac{9-2\sqrt{15}}{21},\frac{6+\sqrt{15}}{21}\right)$	$0.225\lvert T_k\rvert$ $\frac{(155-\sqrt{15})\lvert T_k\rvert}{1200}$ $\frac{(155-\sqrt{15})\lvert T_k\rvert}{1200}$ $\frac{(155-\sqrt{15})\lvert T_k\rvert}{1200}$ $\frac{(155+\sqrt{15})\lvert T_k\rvert}{1200}$ $\frac{(155+\sqrt{15})\lvert T_k\rvert}{1200}$ $\frac{(155+\sqrt{15})\lvert T_k\rvert}{1200}$	5
3	qf1pTlump		$(0,0)$ $(1,0)$ $(0,1)$	$\lvert T_k\rvert/3$ $\lvert T_k\rvert/3$ $\lvert T_k\rvert/3$	1
9	qf2pT4P1		$\left(\frac{1}{4},\frac{3}{4}\right)$ $\left(\frac{3}{4},\frac{1}{4}\right)$ $\left(0,\frac{1}{4}\right)$ $\left(0,\frac{3}{4}\right)$ $\left(\frac{1}{4},0\right)$ $\left(\frac{3}{4},0\right)$ $\left(\frac{1}{4},\frac{1}{4}\right)$ $\left(\frac{1}{4},\frac{1}{2}\right)$ $\left(\frac{1}{2},\frac{1}{4}\right)$	$\lvert T_k\rvert/12$ $\lvert T_k\rvert/12$ $\lvert T_k\rvert/12$ $\lvert T_k\rvert/12$ $\lvert T_k\rvert/12$ $\lvert T_k\rvert/12$ $\lvert T_k\rvert/6$ $\lvert T_k\rvert/6$ $\lvert T_k\rvert/6$	1
15	qf7pT	8	细节参看 [38]		7
21	qf9pT	10	细节参看[38]		9

对于三维区域 $\Omega_h = \sum_{k=1}^{n_t} T_k$, $\mathcal{T}_h = \{T_k\}$, 可以通过如下方式计算在Ω_h上的积分

$$\begin{aligned}\int_{\Omega_h} f(x,y) &= \texttt{int3d(Th)(f)}\\ &= \texttt{int3d(Th,qfV=}\star\texttt{)(f)}\\ &= \texttt{int3d(Th,qforder=}\star\texttt{)(f)}\end{aligned}$$

其中 * 为求积公式的名称或Gauss求积公式的阶。

四面体上的求积公式					
L	qfV=	qforder=	point in $T_k \in \mathbb{R}^3$	ω_ℓ	exact on $P_k, k=$
1	qfV1	2	$\left(\frac{1}{4},\frac{1}{4},\frac{1}{4}\right)$	$\lvert T_k\rvert$	1
4	qfV2	3	$G4(0.58\ldots,0.13\ldots,0.13\ldots)$	$\lvert T_k\rvert/4$	2
14	**qfV5**	6	$G4(0.72\ldots,0.092\ldots,0.092\ldots)$ $G4(0.067\ldots,0.31\ldots,0.31\ldots)$ $G6(0.45\ldots,0.045\ldots,0.45\ldots)$	$0.073\ldots\lvert T_k\rvert$ $0.11\ldots\lvert T_k\rvert$ $0.042\ldots\lvert T_k\rvert$	5
4	qfV1lump		$G4(1,0,0)$	$\lvert T_k\rvert/4$	1

其中，在 $a+3b=1$下， $G4(a,b,b)$是在重心坐标下的以下四个点的集合：

$$\{(a,b,b,b),(b,a,b,b),(b,b,a,b),(b,b,b,a)\}$$

在$2a+2b=1$ 下， $G6(a,b,b)$是在重心坐标下的以下六个点的集合：

$$\{(a,a,b,b),(a,b,a,b),(a,b,b,a),(b,b,a,a),(b,a,b,a),(b,a,a,b)\}.$$

Note 6.12 四面体求积公式参照于 *http://www.cs.kuleuven.be/~nines/research/ecf/mtables.html*

Note 6.13 默认情况下，我们在三角区域或边界（上述三个表格中的黑体字）上用使用5阶多项式。

通过运用插件"qf11to25"，使得在线段、三角区域或四面体上构造特有的求积公式成为了可能。在D维空间的求积公式是一个在$N_q\times(D+1)$上的二元向量，满足在$i=0,...,N_p-1$列上的$D+1$个值为$w^i,\hat{x}_1^i,...,\hat{x}_D^i$，和 $1-\sum_{k=1}^{D}\hat{x}_k^i$，其中$w^i$是正交向量的权重，$\hat{x}_1^i,...,\hat{x}_D^i$是正交向量的重心坐标。

```
                                                    //    测试用 ...  （版本 3.19-1）
load "qf11to25"                                                    //    载入插件

  //    线段上求积
    real[int,int] qq1=[
                        [0.5,0],
                        [0.5,1]];

  QF1 qf1(1,qq1);                                          //    定义在线段上的求积公式qf1
  //    注意:
                                                       //    在多项式的度小于1时，1是求积的阶
  %% 1 is the order of the quadrature  exact? ?  for polynome of degree < 1)
  %% except for?

  //    三角区域上求积
  real[int,int] qq2=[
                      [1./3,0,0],
                      [1./3.,1,0],
                      [1./3.,0,1]];

  QF2 qf2(1,qq2);                                          //    定义在三角区域上的求积公式qf2
  //    remark:
                                                       //    在多项式的度小于1时，1是求积的阶
  //    所以必须有 ⟹ ∑w^i = 1

   //    四面体上求积
  real[int,int] qq3=[
     [1./4,0,0,0],
     [1./4.,1,0,0],
     [1./4.,0,1,0],
     [1./4.,0,0,1]];

  QF3 qf3(1,qq3);                                          //    定义在四面体上的求积公式qf3
  //    remark:
```

```
//     在多项式的度小于1时，1是求积的阶

//     在1d和2d的验证..
real I1 = int1d(Th,qfe=qf1)(x^2) ;
real I1l = int1d(Th,qfe=qf1pElump)(x^2) ;

real I2 = int2d(Th,qft=qf2)(x^2) ;
real I2l = int2d(Th,qft=qf1pTlump)(x^2) ;

cout << I1 << " == " << I1l << endl;
cout << I2 << " == " << I2l << endl;
assert( abs(I1-I1l) < 1e-10);
assert( abs(I2-I2l) < 1e-10);
```

输出为

```
1.67 == 1.67
0.335 == 0.335
```

6.12 变分形式，稀疏矩阵，PDE数据向量

在 FreeFem++ 中，可以定义变分形式，并由此构造矩阵和向量以及存储它们来加速脚本运行（这样能快4倍）。 以求解章节3.4中的热传导为例： 变分形式定义在$L^2(0,T;H^1(\Omega))$空间上；注意到 u^n 满足

$$\forall w \in V_0; \qquad \int_\Omega \frac{u^n - u^{n-1}}{\delta t} w + \kappa \nabla u^n \nabla w) + \int_\Gamma \alpha (u^n - u_{ue}) w = 0$$

其中 $V_0 = \{w \in H^1(\Omega)/w_{|\Gamma_{24}} = 0\}$.
应用矩阵$A = (A_{ij})$, $M = (M_{ij})$和向量$u^n, b^n, b', b", b_{cl}$将问题程式化： （注：若 w为向量，那么w_i为向量的分量）

$$u^n = A^{-1} b^n, \qquad b' = b_0 + M u^{n-1}, \quad b" = \frac{1}{\varepsilon} b_{cl}, \quad b_i^n = \begin{cases} b"_i & \text{if } \ i \in \Gamma_{24} \\ b'_i & \text{else if } \ \notin \Gamma_{24} \end{cases} \tag{6.31}$$

而对于 $\frac{1}{\varepsilon} = \texttt{tgv} = 10^{30}$:

$$A_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_{24}, \text{and} j = i \\ \int_\Omega w_j w_i / dt + k(\nabla w_j . \nabla w_i) + \int_{\Gamma_{13}} \alpha w_j w_i & \text{else if } i \notin \Gamma_{24}, \text{or} j \neq i \end{cases} \tag{6.32}$$

$$M_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_{24}, \text{and} j = i \\ \int_\Omega w_j w_i / dt & \text{else if } i \notin \Gamma_{24}, \text{or} j \neq i \end{cases} \tag{6.33}$$

$$b_{0,i} = \int_{\Gamma_{13}} \alpha u_{ue} w_i \tag{6.34}$$

$$b_{cl} = u^0 \quad \text{the initial data} \tag{6.35}$$

```
//     文件 thermal-fast.edp 在文件夹 examples++-tutorial中

func fu0 =10+90*x/6;
```

```
func k = 1.8*(y<0.5)+0.2;
real ue = 25. , alpha=0.25, T=5, dt=0.1 ;

mesh Th=square(30,5,[6*x,y]);
fespace Vh(Th,P1);

Vh u0=fu0,u=u0;
```

建立三个变分函数，同时构造矩阵 A,M.

```
varf vthermic (u,v)= int2d(Th)(u*v/dt + k*(dx(u) * dx(v) + dy(u) * dy(v)))
            +  int1d(Th,1,3)(alpha*u*v) + on(2,4,u=1);

varf vthermic0(u,v) =   int1d(Th,1,3)(alpha*ue*v);

varf vMass (u,v)= int2d(Th)( u*v/dt)  + on(2,4,u=1);

real tgv = 1e30;
matrix A= vthermic(Vh,Vh,tgv=tgv,solver=CG);
matrix M= vMass(Vh,Vh);
```

现在，为了构建等式的右边，我们需要4个向量。

```
real[int]  b0  = vthermic0(0,Vh);                                  //    RHS的常数部分
real[int]  bcn = vthermic(0,Vh);     //    当Dirichlet边界上节点值为tgv时，( !=0 )
                                     //    即，对于节点 i :  i ∈ Γ24  ⇔  bcn[i] ≠ 0
real[int]  bcl=tgv*u0[];                                           //    Dirichlet边界条件
```

Note 6.14 边界条件 是由惩罚方法实现的， 且向量 *bcn* 包含了边界条件 $u_{=}1$ 的信息， 因此，若要改变边界条件， 我们只需要把新边界上当前的f值逐项地用运算符 $.*$.乘以$bc[]$就行了。 *9.6.2 Examples++-tutorial/StokesUzawa.edp* 给出了运用这些特征的一个实例。

更新后的算法如下:

```
ofstream ff("thermic.dat");
for(real t=0;t<T;t+=dt){
    real[int] b = b0 ;                                             //    对 RHS 而言
    b += M*u[];                                                    //    加上时间独立的部分
                                                                   //    锁定边界部分:
    b = bcn ? bcl  : b ;       //    执行 ∀i:  b[i] = bcn[i] ?  bcl[i] :  b[i] ;
    u[] = A^-1*b;
    ff << t <<" "<<u(3,0.5)<<endl;
    plot(u);
}
for(int i=0;i<20;i++)
  cout<<dy(u)(6.0*i/20.0,0.9)<<endl;
plot(u,fill=true,wait=1,ps="thermic.eps");
```

Note 6.15 用 *varf* 定义的变分形式中的函数是形式的、局部的，唯一重要的问题就是参数的顺序，如

```
varf vb1([u1,u2],q) = int2d(Th)( (dy(u1)+dy(u2)) *q) + int2d(Th)(1*q);
varf vb2([v1,v2],p) = int2d(Th)( (dy(v1)+dy(v2)) *p) + int2d(Th)(1*p);
```

若想通过用 **varf** 类定义的变分形式 a 中的双线性部分来建立矩阵 A，只需要写下以下的代码：

```
A = a(Vh,Wh [, ...] );
//    这里
//   Vh 是 "fespace" 类 的元素，代表含有恰当数量分块的未知区域
//   Wh 是 "fespace" 类 的元素，代表含有恰当数量分块的测试区域
```

在 " [, ...] " 中可能的命名参数有

solver= LU, CG, Crout, Cholesky, GMRES, sparsesolver, UMFPACK ...

缺省值为 GMRES.

上述线性系统的矩阵用何种方式存储取决于求解的方法。对 LU 算法，矩阵是行满秩、非对称的，对 Crout 算法，矩阵是行满秩、对称的，对 Cholesky 算法，矩阵是行满秩、对称正定的，对 CG 算法，矩阵是稀疏、对称正定的，对GMRES， sparsesolver 或 UMFPACK 算法，矩阵仅是稀疏的。

factorize = true，则对LU, Cholesky 或 Crout，做矩阵分解，缺省值为 $false$。

eps= 一个实数表示. ε 为如 CG 之类的迭代算法确定了终止条件。若 ε 为负数，则终止条件为：

$$||Ax - b|| < |\varepsilon|$$

若为正数则为：

$$||Ax - b|| < \frac{|\varepsilon|}{||Ax_0 - b||}$$

precon= 设置预条件的函数名（如P）。 标准的函数 P 的命名必须是

```
func real[int]  P(real[int] & xx) ;
```

tgv= 为执行Dirichlet边界条件而设的最大值(10^{30}) 。

tolpivot= 设置矩阵主元的容许值，分别为：UMFPACK ($10^{-}1$) 以及 LU， Crout， Cholesky 分解 (10^{-20})。

tolpivotsym= 设置 UMFPACK 方法中对称主元的容许值。

strategy= 设置整数 UMFPACK 方法 （缺省值为0）。

Note 6.16 矩阵的行与 *Wh*空间有关，矩阵的列与 *Vh*空间有关。

若想通过变分形式 a 的线性部分构建对偶向量 b （类型为real[int]），只需如下定义

```
real b(Vh.ndof);
b = a(0,Vh);
```

计算网格 Th中的三角片 K 的面积，第一个例子为：

```
fespace Nh(Th,P0);                                    //    空间中的函数：常值/三角
 Nh areaK;
 varf varea(unused,chiK) =  int2d(Th)(chiK);
 etaK[]= varea(0,Ph);
```

这样是很有效的，因为空间 Nh中的基函数，正是网格 Th的元素的特征函数，并且下标就是元素的编号，所以有以下等式：

$$\texttt{etaK}[i] = \int 1_{|K_i} = \int_{K_i} 1;$$

现在，我们可以用此来计算误差大小，正如例 AdaptResidualErrorIndicator.edp，如examples++-tutorial 文件所示。
首先计算函数 h 的一个连续逼近，这是网格 Th 的“密度网格大小”。

```
fespace Vh(Th,P1);
 Vh h ;
 real[int]  count(Th.nv);
 varf vmeshsizen(u,v)=intalledges(Th,qfnbpE=1)(v);
 varf vedgecount(u,v)=intalledges(Th,qfnbpE=1)(v/lenEdge);
                                                      //    计算网格大小
                                                //    ----------------------------
 count=vedgecount(0,Vh);                                //    边界/顶点的数量
 h[]=vmeshsizen(0,Vh);                                  //    边界/顶点的总长度
 h[]=h[]./count;                                        //    边界/顶点的平均长度
```

计算Poisson方程的误差大小：

$$\eta_K = \int_K h_K^2 |(f + \Delta u_h)|^2 + \int_{\partial K} h_e |[\frac{\partial u_h}{\partial n}]|^2$$

这里，h_K 是最长边界的长度 (hTriangle), h_e 是当前边界的长度 (lenEdge), n 是常数。

```
fespace Nh(Th,P0);                                    //    空间中的函数：常值/三角
 Nh etak;
varf vetaK(unused,chiK) =
     intalledges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
    +int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))) );

 etak[]= vetaK(0,Ph);
```

在缺省情况下，我们有自动的优化表达式expression optimization ，若这种优化方法产生了问题，可以用关键词 optimize来关闭它，如下例：

```
varf a(u1,u2)= int2d(Th,optimize=false)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) )
                    +  on(1,2,4,u1=0)  +  on(3,u1=1) ;
```

注意，我们可以建立插值矩阵， 如下例：

```
mesh  TH = square(3,4);
mesh  th = square(2,3);
```

```
mesh  Th = square(4,4);


fespace VH(TH,P1);
fespace Vh(th,P1);
fespace Wh(Th,P1);

matrix B= interpolate(VH,Vh);                    //    建立插值矩阵 Vh->VH
matrix BB= interpolate(Wh,Vh);                   //    建立插值矩阵 Vh->Wh
```

并且，经过一些计算，可以构建稀疏矩阵，例如：

```
  int N=10;
  real [int,int] A(N,N);                                  //    一个满秩矩阵
  real [int] a(N),b(N);
  A =0;
  for (int i=0;i<N;i++)
    {
      A(i,i)=1+i;
      if(i+1 < N)     A(i,i+1)=-i;
      a[i]=i;
    }
  b=A*b;
  cout << "xxxx\n";
  matrix sparseA=A;
  cout << sparseA << endl;
  sparseA = 2*sparseA+sparseA';
  sparseA = 4*sparseA+sparseA*5;
  matrix sparseB=sparseA+sparseA+sparseA; ;
  cout << "sparseB = " << sparseB(0,0) << endl;
```

6.13 插值矩阵

我们可以存储由线性插值算子生成的矩阵，这个算子从有限元空间 V_h 到另一个空间 W_h ，方法是通过函数 interpolate(W_h,V_h,...) 。 注意，连续有限函数是由域外的连续性延拓而来的。函数 interpolate 的命名参数如下：

inside= 设为 true，创建零延拓。

t= 设为 true，得到转置矩阵。

op= 设为如下的整数

- **0** 缺省值，函数值插值
- **1** ∂_x的插值
- **2** ∂_y的插值
- **3** ∂_z的插值

U2Vc= 设置W_h的元素，在插值过程中，从 V_h 到W_h 的整数向量的长度就是 W_h的元素数量。若设为-1 ，则元素设为0，如下例所示：（缺省情况下元素数量不变）。

```
fespace V4h(Th4,[P1,P1,P1,P1]);
fespace V3h(Th,[P1,P1,P1]);
```

```
int[int] u2vc=[1,3,-1];                                    //    -1 => 将元素设为零
matrix IV34= interpolate(V3h,V4h,inside=0,U2Vc=u2vc);      //    V3h <- V4h
V4h [a1,a2,a3,a4]=[1,2,3,4];
V3h [b1,b2,b3]=[10,20,30];
b1[]=IV34*a1[];
```

因此，这里我们得到： b1 == 2, b2 == 4, b3 == 0 . .

Example 6.2 (mat_interpol.edp)

```
mesh Th=square(4,4);
mesh Th4=square(2,2,[x*0.5,y*0.5]);
plot(Th,Th4,ps="ThTh4.eps",wait=1);
fespace Vh(Th,P1);      fespace Vh4(Th4,P1);
fespace Wh(Th,P0);      fespace Wh4(Th4,P0);

matrix IV= interpolate(Vh,Vh4);                                  //    这里，函数
                                                                 //    延拓为连续函数
cout << " IV Vh<-Vh4 " << IV << endl;
Vh v, vv;           Vh4 v4=x*y;
v=v4;               vv[]= IV*v4[];                          //    这里 v == vv =>
real[int]  diff= vv[] - v[];
cout << " || v - vv || = " <<  diff.linfty << endl;
assert( diff.linfty<= 1e-6);
matrix IV0= interpolate(Vh,Vh4,inside=1);                        //    这里，函数
                                                                 //    延拓为零函数
cout << " IV Vh<-Vh4 (inside=1)  " << IV0 << endl;
matrix IVt0= interpolate(Vh,Vh4,inside=1,t=1);
cout << " IV Vh<-Vh4^t (inside=1)  " << IVt0 << endl;
matrix IV4t0= interpolate(Vh4,Vh);
cout << " IV Vh4<-Vh^t  " << IV4t0 << endl;
matrix IW4= interpolate(Wh4,Wh);
cout << " IV Wh4<-Wh  " << IW4  << endl;
matrix IW4V= interpolate(Wh4,Vh);
cout << " IV Wh4<-Vh  " << IW4  << endl;
```

在一列点$(xx[j], yy[j]), i = 0, 2$ 处建立插值矩阵A

$$a_i j = dop(w_c^i(xx[j], yy[j]))$$

这里， w_i 是基本有限元方程， c 是元素个数， dop 是如op 定义中的一个微分算子的类型。

```
real[int] xx=[.3,.4],yy=[.1,.4];
int c=0,dop=0;
matrix Ixx= interpolate(Vh,xx,yy,op=dop,composante=c);
cout << Ixx << endl;
Vh ww;
real[int] dd=[1,2];
ww[]= Ixx*dd;
```

6.14 有限元连接

这里，我们将说明如何从一个有限元空间$W_h(\mathcal{T}_n, *)$中得到相关的信息。其中，“*”可能是 P1, P2, P1nc, 等等。

- `Wh.nt` 给出了 W_h 的元素个数
- `Wh.ndof` 给出 了自由度或未知的个数
- `Wh.ndofK` 给出了单个元素的自由度
- `Wh(k,i)` 给出了元素 k 的第 i 个自由度

参看下述例子：

Example 6.3 (FE.edp)
```
mesh Th=square(5,5);
fespace Wh(Th,P2);
cout << " nb of degree of freedom           : " << Wh.ndof << endl;
cout << " nb of degree of freedom / ELEMENT : " << Wh.ndofK << endl;
 int k= 2, kdf= Wh.ndofK ;;                                          //    元素 2
 cout << " df of element " << k << ":" ;
 for (int i=0;i<kdf;i++)  cout << Wh(k,i) << " ";
 cout << endl;
```

输出为：

```
 Nb Of Nodes = 121
 Nb of DF = 121
 FESpace:Gibbs: old skyline = 5841  new skyline = 1377
 nb of degree of freedom           : 121
 nb of degree of freedom / ELEMENT : 6
 df of element 2:78 95 83 87 79 92
```

第 7 章

可视化

有限元法的结果可能会产生庞大的数据，因此，如何让这些数据变得易于理解尤为重要。在FreeFem++ 中有两种可视化的方法：第一种是系统默认方式，支持绘制网格、实有限元函数的等值线以及向量场。这些都由命令 **plot**来实现（见下方 7.1）。为了方便出版， FreeFem++ 可以将这些图存储为文件格式。
另一种方式是利用外部工具，如，gnuplot（见第 7.2节）、 medit （见第 7.3节），可用命令 **system** 加载它们，并把数据存储在文本文件中。

7.1 画图

用命令 plot，可以呈现网格、标量函数的等值线和向量场。plot 命令的参数可以是：网格、实有限元函数、 2 个实有限元函数的数组、两个双精度数组的组合，分别能够绘制网格、函数、向量场或者由两个双精度数组定义的曲线。

Note 7.1 向量的长度需要在屏幕尺寸的 $[5‰, 5\%]$ 区间内，才是清晰可辨的（否则就会看上去密密麻麻）

参数如下：

wait= 是否等待的布尔变量（缺省值为no wait）。如果选择 true，将会等待键盘或鼠标的输入，对以下按键做出反映：

enter 显示图形

p 前一张图（一共能存储10张图）

? 显示帮助

+,- 在光标指示处放大/缩小 3/2 倍

= 重设视图文件

r 更新图形

up, down, left, right 平移的按键

3 转换 3d/2d 的图形，相关按键：

z,Z) 焦点放大/还原

H,h 拉长/缩短图形的Z轴

鼠标操作

- **左键** 旋转
- **右键** 缩放 （在mac上用ctrl+button）
- **右键 +alt** 平移 （在mac上用alt+ctrl+button ）

a,A 增加/减少向量的长度

B 显示网格的边界

i,I 根据函数的最小/最大边界更新窗口

n,N 减少/增加等值线的数量

b 在黑白和彩色间切换

g 在灰色和彩色间切换

f 在填充和不填充等值线之间切换

l 调整亮度

v 显示颜色代表的数值

m 在是否显示网格间切换s

w 将窗口置于文件 ffglutXXXX.ppm 中

***** 为下一张图保留视图

k 复杂数据 / 改变视图类型

ESC 关闭图形程序，在3.22之前的版本，没有办法关闭。

otherwise 无任何操作。

ps= 需要保存的图形的文件名称的字符串表示 （无法保存 3d 图形）

coef= 向量的长度在单位向量和网格大小之间。

fill= 为等值线填充颜色（默认为P0有限元）。

cmm= 读入图形界面的字符串

value= 画出等值线的值以及向量的值。

aspectratio= 是否保留图形纵横比的布尔变量。

bb= 2 个数组的组合 如 `[[0.1,0.2],[0.5,0.6]]`，设置一个盒状区域，同时再根据两个顶点[0.1,0.2]和[0.5,0.6]定义的区域中指点一个局部视图。

nbiso= (int) 设置等值线担任数量 （默认为20个）

nbarrow= (int) 设置向量值的颜色个数(默认为20个)

viso= 设置等值线的向量 （一个 real[int] 类型的向量，指向值增加的方向）

varrow= 设置彩色向量值的向量（类型为 real[int]）

bw= (bool) 设置是否用黑白色绘图。

grey= (bool) 设置是否用灰色绘图。

hsv= （float类型的向量）设置 3*n个在 HSV 颜色模式下的颜色，例如以下的声明：

```
real[int] colors = [h1,s1,v1,... ,  hn,vn,vn];
```

这里 hi,si,vi 是颜色表中的第i个颜色。

boundary= (bool) 设置是否绘出区域边界（缺省值为true）。

dim= (int)设置2d 或 3d 的亮度 （默认为2）。

add= <b> 未使用

prev= <b> 将前一种状态设为默认状态

ech= <d> 未使用

ZScale= <d> 未使用

WhiteBackground= <b> 未使用

OpaqueBorders= <b> 未使用

BorderAsMesh= <b> 未使用

ShowMeshes= <b> 未使用

ColorScheme= <l> 未使用

ArrowShape= <l> 未使用

ArrowSize= <d> 未使用

ComplexDisplay= <l> 未使用

LabelColors= <b> 未使用

ShowAxes= <b> 未使用

CutPlane= <b> 未使用

CameraPosition= 未使用

CameraFocalPoint= 未使用

CameraViewUp= 未使用

CameraViewAngle= <d> 未使用

CameraClippingRange= 未使用

CutPlaneOrigin= 未使用

CutPlaneNormal= 未使用

WindowIndex= 设置多窗口来呈现多个图形界面。

例如:

```
real[int] xx(10),yy(10);
mesh Th=square(5,5);
fespace Vh(Th,P1);
Vh uh=x*x+y*y,vh=-y^2+x^2;
int i;
                                                          //    计算一个分段
for (i=0;i<10;i++)
 {
   x=i/10.; y=i/10.;
   xx[i]=i;
   yy[i]=uh;                                        //    uh 在 (i/10.  , i/10.)的值
 }
 plot(Th,uh,[uh,vh],value=true,ps="three.eps",wait=true);          //    图 7.1
                                          //    根据顶点[0.1,0.2] 和 [0.5,0.6]来缩放区域
 plot(uh,[uh,vh],bb=[[0.1,0.2],[0.5,0.6]],
        wait=true,grey=1,fill=1,value=1,ps="threeg.eps");          //    图 7.2
 plot([xx,yy],ps="likegnu.eps",wait=true);                         //    图 7.3
```

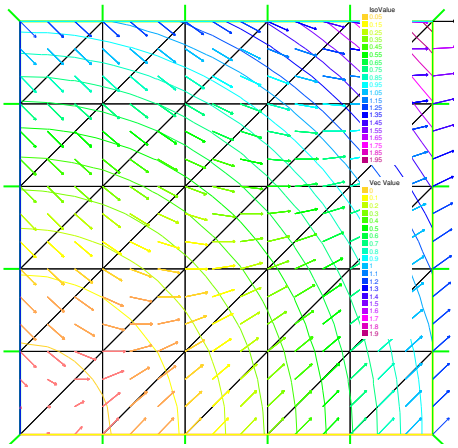


图 7.1: mesh, isovalue, and vector

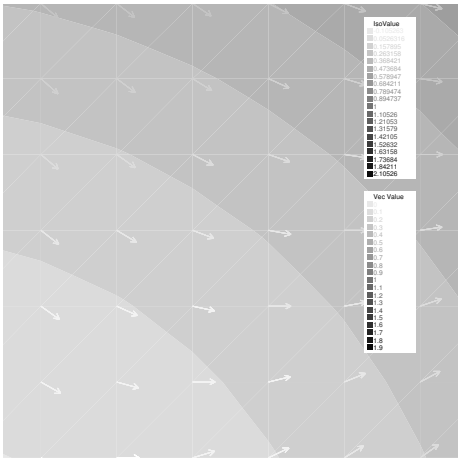


图 7.2: enlargement in grey of isovalue, and vector

若要改变调色板以及选择等值线的值，可以按如下操作：

```
//    访问:  http://en.wikipedia.org/wiki/HSV_color_space
//    HSV (Hue, Saturation, Value) 模型
//    用三种组成成分定义颜色空间
//
//    HSV颜色空间作为颜色轮  7.4
//    Hue是色调，即颜色类型（例如红色，蓝色或黄色）
//    范围为0-360（但是标准范围是0-100）
//    Saturation是颜色的饱和度，范围是0-100
//    颜色的饱和度越低，呈现出来的图越暗
//    颜色也会越淡
//    Value 表示颜色的亮度
//    范围是0-100
```

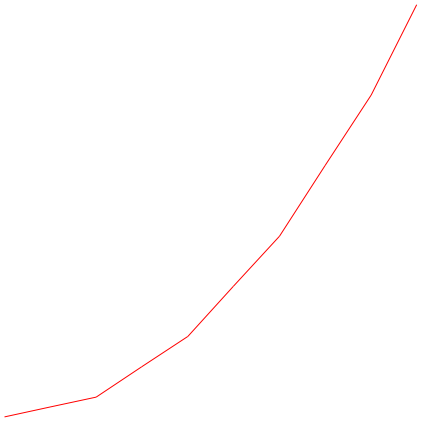


图 7.3: Plots a cut of uh. Note that a refinement of the same can be obtained in combination with gnuplot

```
//
real[int] colorhsv=[                                    //    颜色hsv模型
  4./6., 1 , 0.5,                                       //     深蓝色
  4./6., 1 , 1,                                         //     蓝色
  5./6., 1 , 1,                                         //     紫红色
  1, 1. , 1,                                            //     红色
  1, 0.5 , 1                                            //     浅红色
   ];
 real[int] viso(31);

 for (int i=0;i<viso.n;i++)
   viso[i]=i*0.1;

 plot(uh,viso=viso(0:viso.n-1),value=1,fill=1,wait=1,hsv=colorhsv);
```

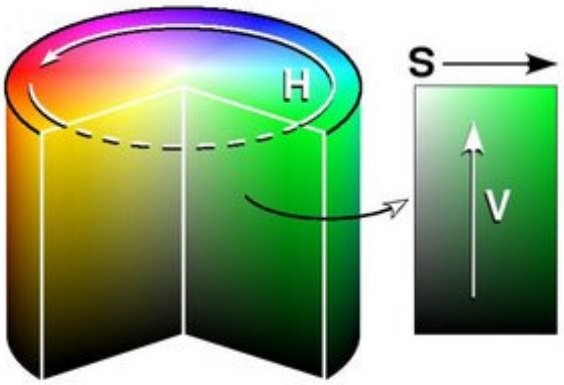


图 7.4: hsv颜色缸

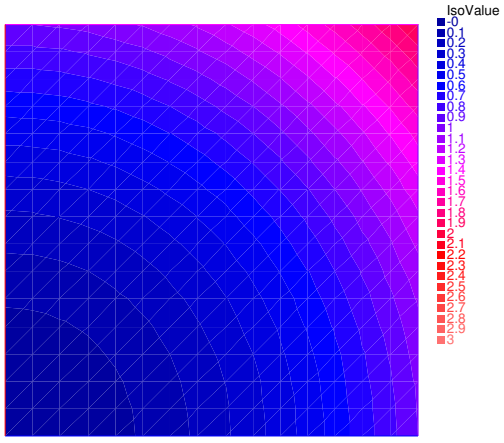


图 7.5: 另一种颜色下的等值线

7.2 关联gnuplot函数

例3.2说明了怎样从FreeFem + +文件中获得gnu − plot 函数。这里我们提供另一种技巧，它具有在线操作的优势，也就是说我们不需要退出FreeFem + +就可以获得一个gnu − plot 函数。但是这项工作需要安装gnuplot[1] 软件，而且只适用于unix系统。
将上述方法运用于先前的例子：

```
{                                                          //    gnuplot文件
  ofstream gnu("plot.gp");
  for (int i=0;i<=n;i++)
   {
     gnu <<  xx[i] << " " << yy[i] << endl;
    }
}                                                  //    plot.gp文件关闭，因为gnu变量已删除

                                                //    执行gnuplot命令并等待5秒（得益于unix命令）
                                                          //    执行postscript plot
exec("echo 'plot \"plot.gp\" w l \
pause 5 \
set term postscript \
set output \"gnuplot.eps\" \
replot \
quit' | gnuplot");
```

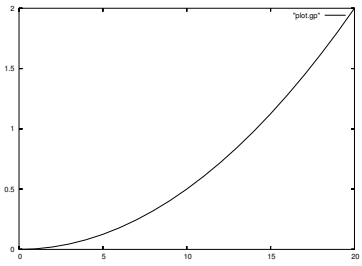


图 7.6: 用gnuplot画出uh的截面

7.3 关联medit函数

正如上面所说，medit[2] 是一个免费的软件，它是由Pascal Frey用OpenGL分享的。你可以运行下面的例子。

备注: 现在 medit软件包含在了 FreeFem++ 且被称为 ffmedit 。
现在有了版本3.2，或者更高的版本。

```
load "medit"
mesh Th=square(10,10,[2*x-1,2*y-1]);
fespace Vh(Th,P1);
Vh u=2-x*x-y*y;
```

[1] http://www.gunplot.info/
[2] http://www-rocq.inria.fr/gamma/medit/medit.html

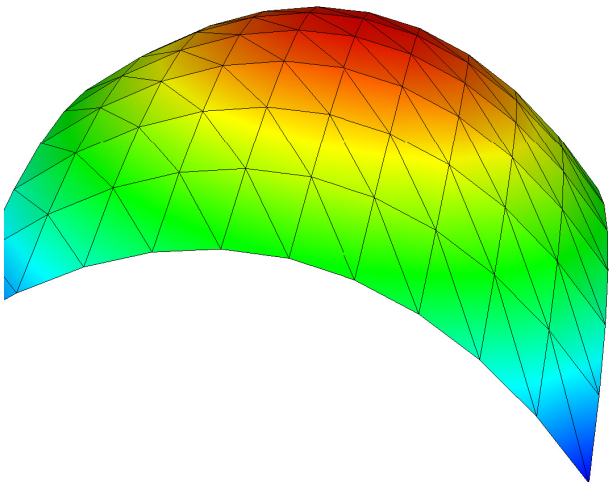


图 7.7: 用medit画图

```
medit("mm",Th,u);
```

在此之前:

```
mesh Th=square(10,10,[2*x-1,2*y-1]);
fespace Vh(Th,P1);
Vh u=2-x*x-y*y;
   savemesh(Th,"mm",[x,y,u*.5]);                    //   保存 mm.points和mm.faces文件
                                                                //    调用 medit
                                                              //    建立一个mm.bb文件
  {
   ofstream file("mm.bb");
   file << "2 1 1 "<< u[].n << " 2 \n";
   for (int j=0;j<u[].n ; j++)
     file << u[][j] << endl;
  }
                                                                        //    执行medit命令
    exec("ffmedit mm");
                                                                    //    在unix OS上清除文件
    exec("rm mm.bb      mm.faces   mm.points");
```

第 8 章

算法和优化

在algo.edp文件中有完整的例子。

8.1 共轭梯度法/广义最小残差算法

假定我们要求解欧拉问题（这里的x跟 FreeFem++中的存储变量对应的第一个坐标没有关系）：求$x \in \mathbb{R}^n$使得：

$$\nabla J(x) = \left(\frac{\partial J}{\partial x_i}(\boldsymbol{x}) \right) = 0 \tag{8.1}$$

这里的J是由$\mathbb{R}^n$映射到$\mathbb{R}$的函数（例如这是我们要极小化的函数）。
如果函数是凸的，我们可以用共轭梯度法来解决这个问题，而且我们只需要一个可以计算∇J 的函数(比如可以叫做 dJ)因此参数是下面这个函数

```
func real[int] dJ(real[int] & xx);
```

的函数名。这个函数通过计算∇J和具有形式

```
real[int] x(20);
```

的向量 x（当然维数20是可以改变的）来初始化程序，进而得到结果。

给定一个向量的初始值$\boldsymbol{x}^{(0)}$，最大迭代次数$i_{\max}$和误差界$0 < \epsilon < 1$，令$\boldsymbol{x} = \boldsymbol{x}^{(0)}$并执行程序

NLCG(∇J, $\boldsymbol{x}$, precon= M, nbiter= $i_{\max}$, eps= ϵ);

便得到$\nabla J(\boldsymbol{x}) = 0$的解$\boldsymbol{x}$。我们可以省略参数 precon, nbiter, eps。 这里M是预条件，经常默认为是单位阵。迭代终止条件是：

$$\|\nabla J(\boldsymbol{x})\|_P \leq \epsilon \|\nabla J(\boldsymbol{x}^{(0)})\|_P$$

如果 eps=中是负数，例如：

NLCG(∇J, $\boldsymbol{x}$, precon= M, nbiter= $i_{\max}$, eps= $-\epsilon$);

我们则可以用下面的迭代终止条件：

$$\|\nabla J(\boldsymbol{x})\|_P^2 \leq \epsilon$$

这三个函数的参数分别是：

nbiter= 给定最大迭代次数（默认值是100）

precon= 给定预条件函数（例如是 P）默认值是单位矩阵，标记函数原型为：

```
func real[int] P(real[int] &x).
```

eps= 给定迭代终止条件ε(默认值是$=10^{-6}$)如果停机准则是正数，则相对停机准则为$||\nabla J(x)||_P \le \varepsilon||\nabla J(x_0)||_P$，否则用绝对值迭代终止条件$||\nabla J(x)||_P^2 \le |\varepsilon|$。

veps= 给定并返回迭代终止条件的值，如果迭代终止条件是正数，则相对迭代终止条件为$||\nabla J(x)||_P \le \varepsilon||\nabla J(x_0)||_P$，否则用绝对值迭代终止条件$||\nabla J(x)||_P^2 \le |\varepsilon|$。此时返回值是迭代终止条件的负值。（备注：这在循环中是非常有用的）。

stop= *stopfunc* 把迭代终止条件放在stop函数之前（在3.31版本之后才有）。这个函数的原型是：

```
func bool stopfunc(int inter, real[int] u, real[int] g).
```

这里的u和g分别是未经过预处理的当前解和当前梯度。

Example 8.1 (algo.edp) 对于给定的函数b，求函数的最小值u。

$$\begin{aligned} J(u) &= \frac{1}{2}\int_\Omega f(|\nabla u|^2) - \int_\Omega ub \\ f(x) &= ax + x - \ln(1+x), \quad f'(x) = a + \frac{x}{1+x}, \quad f''(x) = \frac{1}{(1+x)^2} \end{aligned}$$

给定边界条件在$\partial\Omega$上$u=0$。

```
func real J(real[int] & u)
  {
    Vh w;w[]=u;                                          //    用有限元函数w生成向量u
    real r=int2d(Th)(0.5*f( dx(w)*dx(w) + dy(w)*dy(w) ) - b*w) ;
    cout << "J(u) =" << r << " " << u.min <<  " " << u.max << endl;
    return r;
  }
                                                         //    ----------------------

Vh u=0;                                                  //    解的当前值
Ph alpha;                                                //    存储的df(|∇u|^2)的值
int iter=0;
alpha=df( dx(u)*dx(u) + dy(u)*dy(u) );                   //    优化

func real[int] dJ(real[int] & u)
  {
    int verb=verbosity; verbosity=0;
    Vh w;w[]=u;                                          //    用有限元函数w生成向量u
    alpha=df( dx(w)*dx(w) + dy(w)*dy(w) );               //    优化
    varf au(uh,vh) = int2d(Th)( alpha*( dx(w)*dx(vh) + dy(w)*dy(vh) ) - b*vh)
                   + on(1,2,3,4,uh=0);
    u= au(0,Vh);
    verbosity=verb;
    return u;                                            //    注意对局部数列无返回值
  }
```

/*我们想构造一个预条件C来求解这个问题：寻找$u_h \in V_{0h}$使得

$$\forall v_h \in V_{0h}, \quad \int_\Omega \alpha\nabla u_h.\nabla v_h = \int_\Omega bv_h$$

这里$\alpha = f'(|\nabla u|^2)$。 */

```
varf alap(uh,vh)=  int2d(Th)( alpha *( dx(uh)*dx(vh) + dy(uh)*dy(vh) ))
                   + on(1,2,3,4,uh=0);

varf amass(uh)= int2d(Th)( uh*vh)  + on(1,2,3,4,uh=0);

matrix Amass = alap(Vh,Vh,solver=CG);                                    //
matrix Alap=  alap(Vh,Vh,solver=Cholesky,factorize=1);                   //

                                                                  //     预条件函数
func real[int] C(real[int] & u)
{
   real[int] w = Amass*u;
   u = Alap^-1*w;
   return u;                                                      //     局部变量不返回
}
```

/* 为了解决这个问题，我们进行*10*步共轭梯度迭代。每一步都要重新计算预条件，并重新开始共轭梯度迭代。 */

```
   verbosity=5;
   int conv=0;
   real eps=1e-6;
   for(int i=0;i<20;i++)
   {
     conv=NLCG(dJ,u[],nbiter=10,precon=C,veps=eps);                      //
     if (conv) break;                                                 //     若收敛则终止循环
     alpha=df( dx(u)*dx(u) + dy(u)*dy(u) );          //     重新计算alpha并开始优化
     Alap = alap(Vh,Vh,solver=Cholesky,factorize=1);
     cout << " restart with new preconditionner " << conv
          << " eps =" << eps << endl;
    }

   plot (u,wait=1,cmm="solution with NLCG");
```

对于一个给定的对称正定矩阵A，考虑二次情形

$$J(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$$

此时$J(\boldsymbol{x})$可由$A\boldsymbol{x} = \boldsymbol{b}$的解$\boldsymbol{x}$极小化。在这种情况下，我们可以调用函数 LinearCG

LinearCG(A, $\boldsymbol{x}$, precon= M, nbiter= $i_{\max}$, eps= $\pm\epsilon$);

如果A不是对称的，我们可以使用GMRES（Generalized Minimum Residual广义最小残差）算法：

LinearGMRES(A, $\boldsymbol{x}$, precon= M, nbiter= $i_{\max}$, eps= $\pm\epsilon$);

而且我们还可以用非线性广义最小残差算法（要求函数J是凸的）

LinearGMRES(∇J, $\boldsymbol{x}$, precon= M, nbiter= $i_{\max}$, eps= $\pm\epsilon$);

这个算法的详细描述见 [14][Chapter IV, 1.3]。

8.2 无约束优化算法

COOOL [27]软件包中有两个计算无约束优化的方法，它们是牛顿-拉夫逊（称为牛顿）方法和BFGS方法。这两种方法在FreeFem中可以直接调用（不需要动态链接）。在用这些方法时要格外小心，因为它们的实现使用满矩阵。我们还提供了一些其他的优化算法，这些算法的来自NLopt 文库 [42]并且可以作为Hansen实现的CMAES （它的MPI形式也是可以用）的接口。后面的两个算法可以从动态链接 example++-load文件夹中的 ff-NLopt和 CMA_ES （ 可以在example++-mpi文件夹下找到 mpi 形式的 CMA_ES_MPI）文件中获得。

8.2.1 利用BFGS或者CMAES的例子

```
real[int] b(10),u(10);
func real J(real[int] & u)
  {
    real s=0;
    for (int i=0;i<u.n;i++)
        s +=(i+1)*u[i]*u[i]*0.5 - b[i]*u[i];
    cout << "J ="<< s << " u =" <<  u[0] << " " << u[1] << "...\n" ;
    return s;
  }

                                                    //     J的梯度 (这是RHS所在的仿射形式)
func real[int] DJ(real[int] &u)
  {
    for (int i=0;i<u.n;i++)
      u[i]=(i+1)*u[i]-b[i];
    return u;                                                  //     返回全局变量
  };

b=1; u=2;                                                  //     在等号右边给定初始变量
BFGS(J,dJ,u,eps=1.e-6,nbiter=20,nbiterline=20);
cout << "BFGS: J(u) = " << J(u) << endl;
```

CMA演化方法的使用与之相似，所不同的是，它是不需要导数的一种优化方法， dJ项被忽略，取而代之的是一些其他参数来控制算法。下面是利用此算法来优化上述相同的目标函数的例子（在cmaes-VarIneq.edp中有详细的描述）：

```
load "ff-cmaes"
...                                                  //   define J, u and all here
real min = cmaes(J,u,stopTolFun=1e-6,stopMaxIter=3000);
cout << "minimal value is " << min << " for u = " << u << endl;
```

该算法适用于求解参数空间的一般多元分布函数并尝试使用连续函数值提供的信息来逼近协方差矩阵（更多的细节参见 [43]）。可以通过一些特定的参数来控制初始分布、样本的大小等等。这个算法的命名参数有以下几种：

seed= 随机数发生器 （val 是一个整数）。若不指定的其数值将产生一列基于时钟的随机数。

initialStdDev= 初始协方差矩阵的标准偏差值 (val是一个实数)。如果它的值σ是被传递的，协方差矩阵的初值将被设定为σI。初始的X和$argmin$之间的距离的初始期望值应大致等于initialStdDev。其默认值是0.3。

initialStdDevs= 同上，所不同的是参数是一个数组，允许设置每个参数的标准偏差的初值。应该避免数组元素的数量级不同(如果不能避免这个问题，尝试重新缩放)。

stopTolFun= 如果此步函数值和上一步函数值之差很小，默认值是10^{-12}，则停止迭代。

stopTolFunHist= 如果此步函数值和精确值之差比上步两者之差小，默认值是0（通常不会用到），则停止迭代。

stopTolX= 如果在参数空间中步长比给定的实数值（默认值是0）小，则停终止迭代条件生效。

stopTolXFactor= 当标准偏差的值比这个值增加很多时，终止迭代条件生效。默认值是10^3。

stopMaxFunEval= 当`stopMaxFunEval`函数值已计算完，终止迭代。默认值是$900(n+3)^2$，这里n是参数空间的维数。

stopMaxIter= 当`stopMaxIter`一直被引用时，停止整数搜索。未使用默认值。

popsize= 这是个整数，用来改变样本的大小。其默认值是$4+\lfloor 3\ln(n)\rfloor$， 详细信息见 [43]。 根据`popsize`，增加群体规模通常会提高全局搜索能力，但是其代价是收敛速度会降低为超线性收敛。

paramFile= 这个字符串类型参数允许用户通过所有其他的参数使用一个外部文件中代码比如汉森原代码文件。CMAES算法中的很多参数可以通过这个文件改变 。它的一个例子可以在`examples++-load/ffCMAES/`中找到，上述文件在`initials.par` 目录下。注意到，如果给定了一个输入参数的文件，传递到CMAES函数的参数将会在FreeFem脚本中被忽略。

8.3 IPOPT

`ff-Ipopt`程序包是IPOPT [44]优化的接口。 IPOPT是求解大规模、非线性、无约束优化的软件库。关于它的详细信息见 [44] 和 https://projects.coin-or.org/Ipopt。 它基于线搜索实现了原偶内点算法（primal-dual interior point method）和滤波方法（filter method）。IPOPT需要一个直接稀疏对称线性求解器。如果你的FreeFem的版本是用`--enable-downlad`后缀编译的，它会自动与一个后续的版本MUMPS链接。如果不用MUMPS，你还有其他选择，可以下载HSL子程序并在编译前把它们放在FreeFem++ 的downloads文件下的 `/ipopt/Ipopt-3.10.2/ThirdParty/HSL`路径中。
(程序参见 http://www.coin-or.org/Ipopt/documentation/node16.html)

8.3.1 算法的简短描述

在这一节，我们给出IPOPT的基本数学知识的简短描述。求解非线性光滑函数优化问题的内点法的深层介绍可以参见文献 [45]或者 [44]，文献中介绍了很多IPOPT的明确的原理。IPOPT可以用来解决等式约束和不等式约束的优化问题，虽然为了将非线性不等式的约束面限制在跟等式相同的层面，非线性不等式在优化进程开始之前就组重新排列了。通过引入所需要的松弛变量，将每一个非线性不等式的转化为一对简单有界不等式和非线性等式约束：$c_i(x)\le 0$ 变为 $c_i(x)+s_i=0$ 且 $s_i\le 0$，这里s_i被加到问题的初始变量x_i上。因此，为方便起见，我们将假定在最小化问题中不包含任何非线性不等式约束。这意味着，给定一个函数$f:\mathbb{R}^n\mapsto\mathbb{R}$，我们想要寻找

$$\begin{gathered} x_0=\underset{x\in V}{\operatorname{argmin}} f(x) \\ \text{满足约束条件 } V=\{x\in\mathbb{R}^n \mid c(x)=0 \text{ 且 } x_l\le x\le x_u\} \end{gathered} \tag{8.2}$$

这里$c:\mathbb{R}^n\to\mathbb{R}^m$且$x_l,x_u\in\mathbb{R}^n$ 且对于每个分量不等式均成立。函数f和约束条件c应该是二次连续可导的。 作为一个屏障法，内点算法试图通过求解一系列问题来寻找(8.2)的一个库恩-塔克点，这些问题的约束条件都为等式，因此我们视其为不等式约束问题。这些问题具有如下形式：

$$\text{对于给定的}\mu>0,\ \text{寻找}\ x_\mu=\underset{x\in\mathbb{R}^n\ |\ c(x)=0}{\operatorname{argmin}} B(x,\mu) \tag{8.3}$$

这里μ是一个正实数，$B(x,\mu)=f(x)-\mu\sum_{i=1}^{n}\ln(x_{u,i}-x_i)-\mu\sum_{i=1}^{m}\ln(x_i-x_{l,i})$。

剩余的等式约束用通常的Lagrange乘子法处理。如果障碍参数序列μ收敛到0，直觉上(8.3)的最小值序列收敛到(8.2)的一个局部约束最小值点。对于给定的μ, (8.3)是通过寻找$(x_\mu,\lambda_\mu)\in\mathbb{R}^n\times\mathbb{R}^m$使得

$$\nabla B(x_\mu,\mu)+\sum_{i=1}^{m}\lambda_{\mu,i}\nabla c_i(x_\mu)=\nabla B(x_\mu,\mu)+J_c(x_\mu)^T\lambda_\mu=0\quad\text{and}\quad c(x_\mu)=0 \tag{8.4}$$

成立来求解的。∇B的求解只需要变量x满足：

$$\nabla B(x,\mu)=\nabla f(x)+\begin{pmatrix}\mu/(x_{u,1}-x_1)\\ \vdots\\ \mu/(x_{u,n}-x_n)\end{pmatrix}-\begin{pmatrix}\mu/(x_1-x_{l,1})\\ \vdots\\ \mu/(x_n-x_{l,n})\end{pmatrix}$$

在上述式子中，我们分别用$z_u(x,\mu)=(\mu/(x_{u,1}-x_1),\ldots,\mu/(x_{u,n}-x_n))$ and $z_l(x,\mu)$ 来表示第一和第二个向量，则最适合的(x_μ,λ_μ) 满足：

$$\nabla f(x_\mu)+J_c(x_\mu)^T\lambda_\mu+z_u(x_\mu,\mu)-z_l(x_\mu,\mu)=0\quad\text{且}\quad c(x_\mu)=0 \tag{8.5}$$

在这个问题中，向量z_l和z_u看起来像是作为简单有界不等式的Lagrange乘子的，而且实际上，只要一些技术条件满足，当$\mu\to0$，它们收敛到一些合适的Lagrange 乘子的KKT条件（见 [45]）。

方程式 8.5是通过这样的方法来求解的：对每个下降的μ执行牛顿法来寻找(8.4)的解。为了避免收敛到局部最大值，一些2阶条件也需要考虑，关于它们的详细描述参见 [45]。在最经典的IP算法中，牛顿法被直接运用于(8.4)。 这在大多数情况下是低效率的，因为需要频繁计算大量的不可行点。这个困难可以通过使用Primal-Dual内点算法来克服，在这个方法中(8.4)被转化成一个拉长的方程组，其中z_u和z_l 被看做未知量而且障碍问题的解决是寻找$(x,\lambda,z_u,z_l)\in\mathbb{R}^n\times\mathbb{R}^m\times\mathbb{R}^n\times\mathbb{R}^n$使得

$$\begin{cases}\nabla f(x)+J_c(x)^T\lambda+z_u-z_l &= 0\\ c(x) &= 0\\ (X_u-X)z_u-\mu e &= 0\\ (X-X_l)z_l-\mu e &= 0\end{cases} \tag{8.6}$$

只要 a 是 $\mathbb{R}^n$ 中的一个向量, A 就表示对角阵 $A=(a_i\delta_{ij})_{1\le i,j\le n}$ 且 $e\in\mathbb{R}^n=(1,1,\ldots,1)$ 。用牛顿法求解这个非线性系统就是 原对偶 内点方法。 这里, 更多的细节详见 [45]。 最有用的实施办法介绍了特征从而来使收敛全局化, 特别是在牛顿算法里加入一些线搜索步骤, 或者通过使用信赖域。鉴于 IPOPT 的目的, 这可以通过一个 过滤线搜索 方法来实现, 其细节详见 [**?**]。

更多 IPOPT 具体的特征或实施细节详见 [44]。我们只需记住 IPOPT 是一个求解约束优化问题的聪明的牛顿方法, 因一个加强的线搜索方法而能全局收敛 （意味着算法收敛与初值无关）。由于潜在的牛顿方法, 优化程序需要适应度函数以及约束的阶数高达二阶的所有导数的表达式。 由于问题的海森矩阵很难计算或会导致高纬稠密矩阵, 以更低的收敛率来使用这些对象的 BFGS 近似是可行的。

8.3.2 FreeFem++ 中的 IPOPT

在 FreeFem++ 脚本中调用 IPOPT 优化程序是通过`ff-Ipopt`动态链接库里的`IPOPT`程序来完成的。 IPOPT 是用来解决如下形式的有约束最小化问题的:

$$\begin{aligned} \text{find} \quad & x_0 = \underset{x\in\mathbb{R}^n}{\operatorname{argmin}} f(x) \\ \text{s.t.} \quad & \begin{cases} \forall i \le n,\ x_i^{\text{lb}} \le x_i \le x_i^{\text{ub}} & \text{(simple bounds简单边界)} \\ \forall i \le m,\ c_i^{\text{lb}} \le c_i(x) \le c_i^{\text{ub}} & \text{(constraints functions约束函数)} \end{cases} \end{aligned} \tag{8.7}$$

其中ub和lb分别是指“上界”和“下界”。 如果对于某些$i, 1 \le i \le m$有$c_i^{\text{lb}} = c_i^{\text{ub}}$, 这意味着$c_i$是一个等式约束, 而当$c_i^{\text{lb}} < c_i^{\text{ub}}$时则是一个不等式约束。
有诸多描述适应函数与约束的方法。 比较常见的一种是用关键字`func`来定义函数。任何返回的矩阵都必须是稀疏的（键入 `matrix`, 而不是`real[int,int]`）:

```
func real J(real[int] &X) {...}                    //  适应度函数, 返回一个标量
func real[int] gradJ(real[int] &X) {...}           //  梯度是一个向量

func real[int] C(real[int] &X) {...}               //  约束
func matrix jacC(real[int] &X) {...}               //  约束雅可比
```

警告 1: 在当下的 FreeFem++ 版本中, 一个函数块所返回的矩阵对象会产生出一个未定义的结果。 每定义一个返回稀疏矩阵的函数, 都必须申明一个外部的矩阵对象, 每次调用时相关的函数会对该矩阵进行改写并返回。 以下是一个 `jacC` 的例子:

```
matrix jacCBuffer;                                 //  只是申明, 还没必要定义
func matrix jacC(real[int] &X)
{
  ...                                              //  填充 jacCBuffer
  return jacCBuffer;
}
```

警告 2: IPOPT在对算法初始化的时候需要确定每个矩阵的结构。 当矩阵不是常值并且是由`matrix A = [I,J,C]` 这个语法生成的或是一个中间全满的矩阵时会出现一些错误, 因为在系数矩阵的构建过程中任何空的系数都会被丢弃。这种情况在对矩阵进行线性组合的时候也会发生, 此时任何零系数会导致对组合的矩阵的抑制。对这种问题可以采取适当的措施。可以尝试指定的参数类型 （`checkindex`, `structhess` 和 `structjac` 会有所帮助）。我们强烈建议在构造矩阵时尽可能使用 `varf` 。

海森返回函数在某种程度上有点不同, 因为它必须是拉格朗日函数的海森矩阵: $(x, \sigma_f, \lambda) \mapsto \sigma_f \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 c_i(x)$ 其中 $\lambda \in \mathbb{R}^m$ 并且 $\sigma \in \mathbb{R}$。 你的海森函数就应当有如下的形式:

```
matrix hessianLBuffer;                             //  只是去记住它
func matrix hessianL(real[int] &X,real sigma,real[int] &lambda) {...}
```

如果所有约束函数全都仿射相关, 或者只有简单的边界约束或完全没有约束, 拉格朗日海森矩阵和适应度函数海森矩阵相等, 此时则可以省略参数 `sigma` 和 `lambda`:

```
matrix hessianJBuffer;
func matrix hessianJ(real[int] &X) {...}      //  当所有约束都仿射相关时的海森雏形
```

当这些函数被定义完后, IPOPT 按如下方式调用:

```
real[int] Xi = ... ;                               //  起始点
```

```
IPOPT(J,gradJ,hessianL,C,jacC,Xi, /*some named parameters*/ );
```

如果海森矩阵缺省, 界面会告诉 IPOPT 去使用 (L)BFGS 近似法 （这也可以由一个命名参数来完成, 稍后会有介绍）。简单边界问题或无约束问题并不需要约束部分, 因而下面的语句是有效的:

```
IPOPT(J,gradJ,C,jacC,Xi, ... );                    //  带 BFGS 的 IPOPT
IPOPT(J,gradJ,hessianJ,Xi, ... );                  //  无约束的牛顿 IPOPT
IPOPT(J,gradJ,Xi, ... );                           //  BFGS, 无约束
```

简单边界是用被命名的参数 lb 和 ub 来传递的, 而约束边界则是由 clb 和 cub 来传递的。 在某些方向上的无界性可由 $1e^{19}$ 和 $-1e^{19}$来完成, IPOPT 认为这两者分别是 $+\infty$ 和 $-\infty$:

```
real[int] xlb(n),xub(n),clb(m),cub(m);
...                                                //  填充数组...
IPOPT(J,gradJ,hessianL,C,jacC,Xi,lb=xlb,ub=xub,clb=clb,cub=cub,  /*some other
named parameters*/ );
```

P2 适应度函数和仿射约束函数: 如果适应度函数或约束函数能转化为如下形式 :

$$\begin{array}{cc} \forall x \in \mathbb{R}^n,\ f(x) = \frac{1}{2}\langle Ax, x\rangle + \langle b, x\rangle & (A,b) \in \mathcal{M}_{n,n}(\mathbb{R}) \times \mathbb{R}^n \\ \text{or},\ C(x) = Ax + b & (A,b) \in \mathcal{M}_{n,m}(\mathbb{R}) \times \mathbb{R}^m \end{array}$$

当 A 和 b 是常数时, 就有可能用元素对 (A,b) 来取代对三个（或两个）函数的定义。这也暗示 IPOPT 有些对象是恒定的因而只需计算一次, 以此来避免反复拷贝相同的矩阵。 语法如下 :

```
                                                   //  带“标准”适应度函数的仿射约束
matrix A= ... ;                                    //  约束的线性部分
real[int] b = ... ;                                //  约束的恒定部分
IPOPT(J,gradJ,hessianJ, [A,b] ,Xi, /*bounds and named params*/);
                                                   //  [b,A] 也可以...可以防止粗心...
```

注意到如果你这样定义约束, 这对海森矩阵是没有作用的, 因而海森矩阵只需要一个 real[int] 。

```
                                                   //  仿射约束和 P2 适应度函数:
matrix A= ... ;                                    //  双线性形式矩阵
real[int] b = ... ;                                //  f的线性部分
matrix Ac= ... ;                                   //  约束的线性部分
real[int] bc= ... ;                                //  约束的恒定部分
IPOPT([A,b], [Ac,bc] ,Xi, /*bounds and named params*/);
```

如果目标函数和约束函数都用这种方式给出, 这会自动激活 IPOPT 选项 mehrotra_algorithm （根据参考文献最好是用来处理线性或是二次的程序）。 否则, 这个选项只能通过选项文件来设定（见被命名的参数那节）。
在对约束使用标准函数时以这种方式定义 f 是一个错误的案例:

```
matrix A= ... ;                                    //  双线性形式矩阵
real[int] b = ... ;                                //  f的线性贡献
func real[int] C(real[int] &X) {...}               //  约束
func matrix jacC(real[int] &X) {...}               //  约束雅可比
IPOPT([A,b],C,jacC,Xi, /*bounds and named params*/);
```

确实, 在传递 [A,b] 从而定义 f 的时候, 拉格朗日海森矩阵会自动产生一个常值 $x \mapsto A$ 方程, 没法增加可能的约束贡献, 导致了不正确的二阶导数。 因此, 像这样的问题需要分成仅仅两种可能来讨论: 1） 约束是非线性的但你想用 BFGS 模式（然后把 bfgs=1 加入被命名的参数）, 2）约束是仿射相关的, 但在这个情况下, 为什么不用同样的方法传递他们呢?

这里有另外一些对问题的无效定义（比如当 f 是一个纯二次或线性形式, 或者 C 是一个纯线性函数, 等等...）：

```
                                                    //   纯二次的f - A 是一个矩阵 :
IPOPT(A, /*constraints args*/, Xi, /*bounds and named params*/);
                                                  //   纯线性的f - b 是一个 real[int] :
IPOPT(b, /*constraints args*/, Xi, /*bounds and named params*/);
                                                      //   线性约束 - Ac 是一个矩阵
IPOPT(/*fitness func args*/, Ac, Xi, /*bounds and named params*/);
```

返回值： IPOPT 函数返回一个 int型的错误代码。 当算法成功运行后会得到一个零值, 而正数值反应了 IPOPT 遇到了少数问题。 负值则揭露了更多有问题的情况。 联合的 IPOPT 返回标签在下表中列出。 这个IPOPT pdf 参考资料提供了更多关于这些返回情形的准确描述：

Success		Failures	
0	Solve_Succeeded	-1	Maximum_Iterations_Exceeded
1	Solved_To_Acceptable_Level	-2	Restoration_Failed
2	Infeasible_Problem_Detected	-3	Error_In_Step_Computation
3	Search_Direction_Becomes_Too_Small	-4	Maximum_CpuTime_Exceeded
4	Diverging_Iterates		
5	User_Requested_Stop		
6	Feasible_Point_Found		
Problem definition issues		**Critical errors**	
-10	Not_Enough_Degrees_Of_Freedom	-100	Unrecoverable_Exception
-11	Invalid_Problem_Definition	-101	NonIpopt_Exception_Thrown
-12	Invalid_Option	-102	Insufficient_Memory
-13	Invalid_Number_Detected	-199	Internal_Error

被命名的参数： 在这个界面中可用的被命名参数是那些我们认为是最受从一个最优化到另一个的变化所影响的, 再加上一些界面的特性。正如我们在http://www.coin-or.org/Ipopt/documentation/node59.html中看到的一样, 有很多参数可以在 IPOPT 里改变, 从而影响算法的运转状态。 这些参数仍然可以通过在执行目录中放置一个选择文件来控制。 注意, IPOPT's pdf 参考资料可能会比之前提及的线上版本提供更多关于特定参数的信息. 在脚本中可用的参数是：

lb, ub： 搜索变量的简单下界和上界的real[int]的大小必须是 n （搜索空间维数）。 如果这些数组中同一个指标的两个成分相等, 那么对应的搜索变量就固定了。 通过系统默认 IPOPT 会把固定的变量从优化过程中移除, 并且常常在调用函数时使用固定的值。 它可以通过使用参数 fixedvar 来改变。

clb, cub： 约束下界和约束上界的real[int]大小为 m （约束个数）。 clb 和 cub 中相同指标 i 的两个成分之间的等式对应了一个等式约束。

structjacc： 在 I 和 J 是两列整数数组的形式 [I,J] 下传递约束雅可比矩阵的最大可能结构（非零系数的指标）。如果没有被定义, 约束雅可比矩阵的结构在 Xi 中被计算时会被使用（如果雅可比矩阵是恒定的或总是用相同的varf 定义的话不会有问题, 而当它有相邻的三列元素或者包含一个满的矩阵则会比较危险）。

structhess： 同上, 除了对于海森函数（如果 f 是 P2 或者更少并且约束都仿射相关时不被使用）。 同样, 记住这是拉格朗日函数的海森矩阵（这只有在约束都仿射相关时才和 f 的海森矩阵相等）。 如果这个参数没有给出结构, 拉格朗日海森矩阵会在起始点被计算, 此时 $\sigma = 1$ 且 $\lambda = (1, 1, \ldots, 1)$ （如果所有约束和适应度函数海森矩阵都是恒定的或者由 varf所建, 这会很安全, 并且如果由相邻的三列元素或满矩阵所建的话会不那么可靠）。

checkindex : 是一个 bool, 当矩阵从 FreeFem 函数拷贝到 IPOPT 数组中时会诱发指标分叉搜索。 这是当一些零系数被 FreeFem 从矩阵里移除时用来避免错误的指标匹配。 这不会解决因算法初始化时给出过小的结构而导致的问题。 由默认激活 （除非所有矩阵都显然是恒定的）。

warmstart : 如果设定为 true , 约束对偶变量 λ, 和简单边界对偶变量会由传递到被命名的变量 lm, lz 和 uz 的数组的值来初始化 （见下）。

lm : 大小为 m 的 real[int] , 是用来获取约束对偶变量 λ 的最终值和/或在'warm start' 的情况下将它们初始化 （被传递的数组也会在算法的最后更新为最终的对偶变量值）。

lz, uz : 大小为 n 的 real[int] 来获得最终值和/或(在'warm start' 的情况下)将与简单边界相关联的对偶变量初始化。

tol : real, 算法的收敛准则, 默认值是 10^{-8} 。

maxiter : int, 迭代次数上限, 默认值为 3000 。

maxcputime : real value, 运行时长。 默认值是 10^6 （大约 11 天半）。

bfgs : bool, 可以实现或不实现拉格朗日海森矩阵的（低储存）BFGS 近似。 它被默认设置为错误, 除非没法用传递到 IPOPT 的函数来计算海森矩阵。

derivativetest : 被用来执行对于用有限差分计算来给 IPOPT 的导数的比较. 可能的 string 值是: "none" （默认）, "first-order", "second-order" and "only-second-order"。相关联导数误差容忍度可由选项文件改变. 在尝试之前最好不要在意由它给出的任何错误, 如果失败了, 执行第一次优化。

dth : 对由有限差分计算的导数测试所用的摄动参数。 默认设为 10^{-8}。

dttol : 导数测试错误检查的容忍值（默认值尚不确定, 可能为 10^{-5}）。

optfile : 参数 string 用来指定 IPOPT 选项文件名. IPOPT会找一个 ipopt.opt文件作为默认。 在文件中设定的选项会覆盖在 FreeFem 脚本中定义的选项.

printlevel : 用来控制 IPOPT 输出打印级别的一个 int , 默认设为 5, 可能值为 0 到 12。输出信息的描述见 IPOPT 的 pdf 参考资料。

fixedvar : string , 用来定义简单边界等式约束处理 : 用 "make_parameter" （默认值） 来简单地把它们从优化过程中移除（函数总会由固定值来计算那些变量）, 用 "make_constraint" 来把它们当作其他约束或"relax_bounds" 以此来放宽固定边界约束。

mustrategy : 用来对障碍参数 μ 选择更新策略的一个 string . 两种可能的标签是 "monotone",用来使用单调(Fiacco-McCormick) 策略, 或 "adaptive" （默认设定）。

muinit : real 正数值用来给障碍参数初始化。 只有当 mustrategy 被设置成 monotone 时才起作用。

pivtol : real , 用来给线性求解程序设定中心点容忍度的值. 稀疏的话要少一些中心点, 稳定的话需要多一些中心点. 值必须在区间 $[0, 1]$ 里, 默认值为 10^{-6} 。

brf : 边界松弛因子 : 在开始优化之前, 使用者给出的边界是被松弛化的。 这个选项为松弛设置了因子。如果被设定为零, 那么边界松弛就失效了。这个 real 必须是正数且它的默认值是 10^{-8}。

objvalue : 一个 real 型变量的标示符, 为了获得目标函数的最终值 （成功情况下最好的值）。

mumin : 障碍参数 μ 的最小值, 默认值为10^{-11}的一个 real 。

linesearch : 一个当设定为 false时会使线搜索失效的布尔型变量。 线搜索会默认激活。 当线搜索失效时, 会变成一个标准牛顿算法而不是原始对偶算法。此时全局收敛就不能保证了, 意味着很多初始值会使迭代发散。但另一方面, 它也会在尝试寻找一个精确局部最小值时更加有效, 且能避免一些无法控制的程序使得迭代被其他一些靠近最优位置的点捕捉到。

8.4 使用 IPOPT 的一些简例

Example 8.2 (IpoptVI.edp) 考虑一个非常简单的例子, 给定两个函数 f 和 g (定义在$\Omega \subset \mathbb{R}^2$上), 最小化 $J(u) = \frac{1}{2}\int_\Omega |\nabla u|^2 - \int_\Omega fu$, 其中 $u \le g$ 几乎处处成立 :

```
load "ff-Ipopt";                                                    //   加载界面
int nn=20;                                                          //   网格质量
mesh Th=square(nn,nn);                                          //   做一个方形网格
fespace Vh(Th,P1);                                                //   有限元空间

func f = 1.;                                                          //   函数
real r=0.03,s=0.1;                                              //   g 的一些参数
func g = r - r/2*exp(-0.5*(square(x-0.5)+square(y-0.5))/ square(s));
                                                        //   g 是常数减去一个高斯（分布）

macro Grad(u) [dx(u),dy(u)]                                         //   梯度算子
varf vP(u,v) = int2d(Th)(Grad(u)'*Grad(v)) - int2d(Th)(f*v);
```

这里我们构造与函数相关的矩阵和第二个成员来做一次性极小化。适应度函数的句法 [A,b] 此时就用来将它传递到 *IPOPT* 。

```
matrix A = vP(Vh,Vh,solver=CG);
real[int] b = vP(0,Vh);
```

我们仅使用简单边界条件，即在 $\partial\Omega$ 上 $u = 0$, 同时有 $u \le g$ 。

```
Vh lb=-1.e19;                                                       //   内部无下界
Vh ub=g;                                                        //   内部上界为 g
varf vGamma(u,v) = on(1,2,3,4,u=1);
real[int] onGamma=vGamma(0,Vh);
ub[] = onGamma ? 0. : ub[];                                         //   执行边界条件
lb[] = onGamma ? 0. : lb[];

Vh u=0;                                                             //   起始点
IPOPT([A,b],u[],lb=lb[],ub=ub[]);                                   //   求解该问题
plot(u,wait=1);
```

Example 8.3 (IpoptVI2.edp) 令 Ω 为 $\mathbb{R}^2$ 的一个区域, $f_1, f_2 \in L^2(\Omega)$ 且 $g_1, g_2 \in L^2(\partial\Omega)$ 并且 $g_1 \le g_2$ 几乎处处成立。 我们定义空间 :

$$V = \{(v_1, v_2) \in H^1(\Omega)^2; v_1|_{\partial\Omega} = g_1, v_2|_{\partial\Omega} = g_2, v_1 \le v_2 \text{ a.e. } \}$$

以及函数 $J : H^1(\Omega)^2 \longrightarrow \mathbb{R}$:

$$J(v_1, v_2) = \frac{1}{2}\int_\Omega |\nabla v_1|^2 - \int_\Omega f_1 v_1 + \frac{1}{2}\int_\Omega |\nabla v_2|^2 - \int_\Omega f_2 v_2$$

问题转化为寻找（数值上）两个函数 $(u_1, u_2) = \underset{(v_1,v_2)\in V}{\operatorname{argmin}} J(v_1, v_2)$.

```
load "ff-IpOpt";

mesh Th=square(10,10);
fespace Vh(Th,[P1,P1] );
fespace Wh(Th,[P1] );
int iter=0;

func f1 =  10;                                                    //   右边
func f2 = -15;
func g1 = -0.1;                                                   //   边界条件函数
func g2 =  0.1;

while(++iter)                                                     //   网格适应性循环
{
macro Grad(u) [dx(u),dy(u)]                                       //   梯度宏
varf vP([u1,u2],[v1,v2]) = int2d(Th)(Grad(u1)'*Grad(v1)+ Grad(u2)'*Grad(v2))
- int2d(Th)(f1*v1+f2*v2);

matrix A = vP(Vh,Vh);                                             //   适应度函数矩阵...
real[int] b = vP(0,Vh);                                           //   和线性形式

int[int] II1=[0],II2=[1];                                         //   约束矩阵
matrix C1 =  interpolate (Wh,Vh, U2Vc=II1);
matrix C2 =  interpolate (Wh,Vh, U2Vc=II2);
matrix CC = -1*C1 + C2;                                           //   u2 - u1 >0
Wh cl=0;                                                          //   约束下界（无上界）

                                                                  //   边界条件
varf vGamma([u1,u2],[v1,v2]) = on(1,2,3,4,u1=1,u2=1);
real[int] onGamma=vGamma(0,Vh);
Vh [ub1,ub2]=[g1,g2];
Vh [lb1,lb2]=[g1,g2];
ub1[] = onGamma ? ub1[] : 1e19  ;                                 //   内部无边界
lb1[] = onGamma ? lb1[] : -1e19  ;

Vh [u1,u2]=[0,0];                                                 //   起始点

IPOPT([b,A],CC,u1[],lb=lb1[],ub=ub1[],clb=cl[]);

plot(u1,u2,wait=1,nbiso=60,dim=3);
if(iter > 1) break;
Th= adaptmesh(Th,[u1,u2],err=0.004,nbvx=100000);
}
```

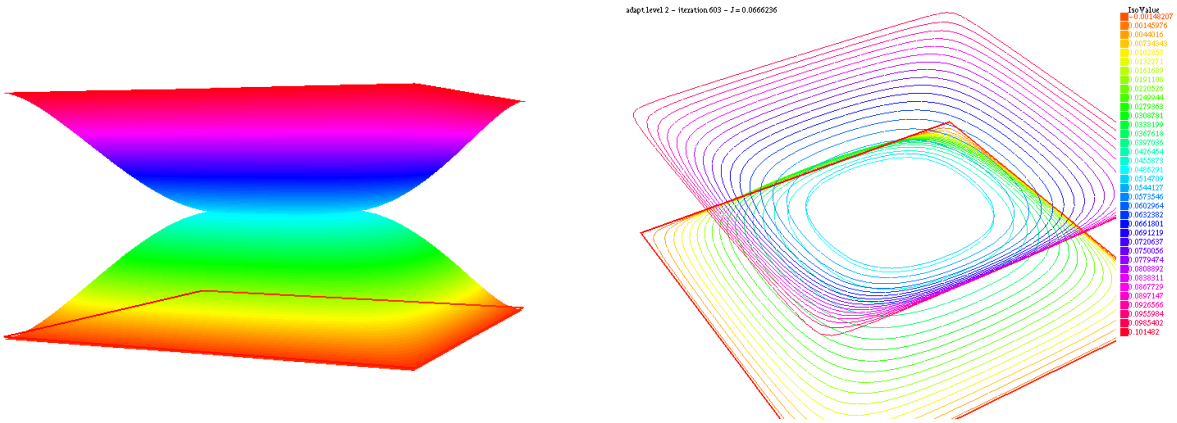


图 8.1: 变分不等式的数值逼近

8.5 3D约束极小曲面与 IPOPT

8.5.1 面积和体积表示

这个例子旨在利用 IPOPT 算法数值求解一些约束最小表面问题。我们将其限制在 C^k $(k \geq 1)$, 闭的, 球面参数化的表面, 例如表面 S 满足：

$$\exists \rho \in C^k([0,2\pi]\times[0,\pi])|S = \left\{ X = \begin{pmatrix} \rho(\theta,\phi) \\ 0 \\ 0 \end{pmatrix}, (\theta,\phi) \in [0,2\pi]\times[0,\pi] \right\}$$

其中的元素是在球面坐标系中表达的。我们称 Ω 是 $[0,2\pi]\times[0,\pi]$ 的角参数集。为了排除自交以及开的形状, 以下对 ρ 的假设为：

$$\rho \geq 0 \quad \text{and} \quad \forall \phi, \rho(0,\phi) = \rho(2\pi,\phi)$$

从而定义的曲面的第一基本形式(度量)就有如下的矩阵表达：

$$G = \begin{pmatrix} \rho^2\sin^2(\phi) + (\partial_\theta \rho)^2 & \partial_\theta \rho \partial_\phi \rho \\ \partial_\theta \rho \partial_\phi \rho & \rho^2 + (\partial_\phi \rho)^2 \end{pmatrix} \tag{8.8}$$

这个度量是用来表示曲面的面积。 令 $g = \det(G)$, 则我们有：

$$\mathcal{A}(\rho) = \int_\Omega \|\partial_\theta X \wedge \partial_\phi X\| = \int_\Omega \sqrt{g} = \int_\Omega \sqrt{\rho^2(\partial_\theta\rho)^2 + \rho^4\sin^2(\phi) + \rho^2(\partial_\phi\rho)^2\sin^2(\phi)}\, d\theta d\phi \tag{8.9}$$

空间内部的体积更加容易表示：

$$\mathcal{V}(\rho) = \int_\Omega \int_0^{\rho(\theta,\phi)} r^2\sin(\phi) dr d\theta d\phi = \frac{1}{3}\int_\Omega \rho^3 \sin(\phi) d\theta d\phi \tag{8.10}$$

8.5.2 导数

为了使用基于牛顿的内点最优算法, 必须计算 $\mathcal{A}$ 和 $\mathcal{V}$ 关于 rho 的导数。考虑到面积我们有如下结果：

$$\forall v \in C^1(\Omega)\ ,\ \langle d\mathcal{A}(\rho), v\rangle = \int_\Omega \frac{1}{2}\frac{d\bar{g}(\rho)(v)}{\sqrt{g}} d\theta d\phi$$

其中 $\bar{g}$ 是将 $(\theta,\phi)\mapsto g(\theta,\phi)$ 变为关于 ρ 的标量场。这导致了如下的表示, 使用其能更容易转变成 freefem 脚本：

$$\begin{aligned}\forall v\in C^1(\Omega)\ ,\ \langle d\mathcal{A}(\rho),v\rangle \ &= \int_\Omega \left(2\rho^3\sin^2(\phi)+\rho(\partial_\theta\rho)^2+\rho(\partial_\phi\rho)^2\sin^2(\phi)\right)v\\ &+\int_\Omega \rho^2\partial_\theta\rho\partial_\theta v\ +\ \rho^2\partial_\phi\rho\sin^2(\phi)\partial_\phi v\end{aligned} \tag{8.11}$$

类似地, 可以导出二阶导数表示。 尽管没有表现出具体的困难, 详细的计算比较乏味, 结果是这些导数可以用一个 3×3 的矩阵 $\boldsymbol{B}$ 来表示, 这个矩阵的系数可由 ρ 表达且它的导数可表示成 θ 和 ϕ , 使得：

$$\forall (w,v)\in C^1(\Omega)\ ,\ d^2\mathcal{A}(\rho)(w,v)=\int_\Omega \begin{pmatrix} w & \partial_\theta w & \partial_\phi w\end{pmatrix}\boldsymbol{B}\begin{pmatrix} v\\ \partial_\theta v\\ \partial_\phi v\end{pmatrix}d\theta d\phi \tag{8.12}$$

推导体积函数的导数则更加简单。 我们可立即得到如下表示：

$$\begin{aligned}\forall v\ ,\ \langle d\mathcal{V}(\rho),v\rangle \ &= \int_\Omega \rho^2\sin(\phi)v\ d\theta d\phi\\ \forall w,v\ ,d^2\mathcal{V}(\rho)(w,v)\ &= \int_\Omega 2\rho\sin(\phi)wv\ d\theta d\phi\end{aligned} \tag{8.13}$$

8.5.3 问题和其脚本：

完整的代码可在 `IpoptMinSurfVol.edp` 中找到。我们想要求解如下问题：

Example 8.4 给定一个正的分段连续函数 ρ_{object} , 和一个标量 $\mathcal{V}_{\max}>\mathcal{V}(\rho_{\text{object}})$, 求解 ρ_0 使得：

$$\rho_0=\underset{\rho\in C^1(\Omega)}{\operatorname{argmin}}\ \mathcal{A}(\rho)\ ,\ \text{s.t. } \rho_0\geq\rho_{\text{object}}\ \text{and}\ \mathcal{V}(\rho_0)\leq\mathcal{V}_{\max}$$

如果 ρ_{object} 是一个三维对象（区域）的表面 $\mathcal{O}$ 的球面参数, 它可以转述为给定体积大小的对象找一个最小面积的表面。
如果 $\mathcal{V}_{\max}$ 很接近 $\mathcal{V}(\rho_{\text{object}})$, 那么 ρ_0 和 ρ_{object} 也会很接近. 若 $\mathcal{V}_{\max}$ 取更大的值, ρ 会更靠近无约束最小曲面周围的 $\mathcal{O}$, 这会在 $\mathcal{V}_{\max}\geq\frac{4}{3}\pi\|\rho_{\text{object}}\|_\infty^3$ 时达到 （充分非必要）。

同样很有趣的是, 解决相同的问题而有约束 $\mathcal{V}(\rho_0)\geq\mathcal{V}_{\min}$, 其当 $\mathcal{V}_{\min}\geq\frac{1}{6}\pi\text{diam}(\mathcal{O})^3$ 时会导出一个球, 且随着 $\mathcal{V}_{\min}$ 的减小而向无约束问题的解移动。

我们开始给区域 $[0,2\pi]\times\ [0,\pi]$ 画网格, 随后定义了一个周期的 P1 有限元空间。

```
load "msh3";
load "medit";
load "ff-Ipopt";

int np=40;                                          //   初始网格质量参数
mesh Th = square(2*np,np,[2*pi*x,pi*y]);

fespace Vh(Th,P1,periodic=[[2,y],[4,y]]);
Vh startshape=5;                                    //   初始形状
```

我们构造了一些有限元函数, 其潜在的数组会被用来储存与所有约束相关的对偶变量的值, 从而当我们用网格适应的时候能用它来对算法重新初始化。此时, 算法会几乎以在网格适应之前达到的精度来重启, 因此节省了许多迭代。

```
Vh uz=1.,lz=1.;                                          //   简单边界对偶变量
real[int] lm=[1];                                        //   体积约束的对偶变量
```

然后, 紧跟着网格适应循环, 和一个传递函数, Plot3D, 使用 3D 网格来展现由 medit 传递的形状（程序 movemesh23 在遇到凹凸不平的形状时常常会崩溃）。

```
int nadapt=1;
for(int kkk=0;kkk<nadapt;++kkk)                          //   网格适应性循环
{

int iter=0;                                              //   循环计数
func sin2 = square(sin(y));                              //   经常要用的一个函数

func int Plot3D(real[int] &rho,string cmm,bool ffplot) {...}   //   见 .edp 文件
```

这里是一些和面积计算与它形状导数相关的函数, 根据方程 8.9 和 8.11 :

```
func real Area(real[int] &X)
{
  Vh rho;
  rho[] = X;
  Vh rho2 = square(rho);
  Vh rho4 = square(rho2);
  real res = int2d(Th)(  sqrt( rho4*sin2
                               + rho2*square(dx(rho))
                               + rho2*sin2*square(dy(rho)) )
                       );
  ++iter;
  plot(rho,  ... /*参数*/ ... );
  return res;
}

func real[int] GradArea(real[int] &X)                    //   梯度函数
{
  Vh rho,rho2;
  rho[] = X;
  rho2[] = square(X);
  Vh sqrtPsi,alpha;                                      //   Psi是G的行列式
  {                                                      //   优化
    Vh  dxrho2 = dx(rho)*dx(rho), dyrho2 = dy(rho)*dy(rho);
    sqrtPsi = sqrt( rho2*rho2*sin2 + rho2*dxrho2 + rho2*dyrho2*sin2 );
    alpha = 2.*rho2*rho*sin2 + rho*dxrho2 + rho*dyrho2*sin2;
  }
  varf dSurface(u,v) =
    int2d(Th)(1./sqrtPsi*(alpha*v+rho2*dx(rho)*dx(v)+rho2*dy(rho)*sin2*dy(v)));
  real[int] grad = dSurface(0,Vh);
  return grad;
}
```

对给定形状的区域, 函数返回的海森矩阵显得有些模糊，因此我们不会在此解释方程8.12中所有系数的定义。读者可以在edp 文件中查看它们。

```
matrix hessianA;                                         //   全局矩阵缓冲区域

func matrix HessianArea(real[int] &X)
```

```
{
  Vh rho,rho2;
  rho[] = X;
  rho2 = square(rho);
  Vh sqrtPsi,sqrtPsi3,C00,C01,C02,C11,C12,C22,A;
  {
    ...                                                    //   以上函数的定义
  }
  varf d2Area(w,v) =
    int2d(Th)(
      1./sqrtPsi * (A*w*v + 2*rho*dx(rho)*dx(w)*v + 2*rho*dx(rho)*w*dx(v)
                    + 2*rho*dy(rho)*sin2*dy(w)*v + 2*rho*dy(rho)*sin2*w*dy(v)
                    + rho2*dx(w)*dx(v) + rho2*sin2*dy(w)*dy(v))
    + 1./sqrtPsi3 * (C00*w*v + C01*dx(w)*v + C01*w*dx(v) + C02*dy(w)*v
                    + C02*w*dy(v) + C11*dx(w)*dx(v)
                    + C12*dx(w)*dy(v) + C12*dy(w)*dx(v) + C22*dy(w)*dy(v))
     );                                                    //   结束int2d以及varf
  hessianA = d2Area(Vh,Vh);
  return hessianA;
}
```

以及与体积有关的函数：

```
func real Volume(real[int] &X)
{
  Vh rho;
  rho[]=X;
  Vh rho3=rho*rho*rho;
  real res = 1./3.*int2d(Th)(rho3*sin(y));
  return res;
}

func real[int] GradVolume(real[int] &X)
{
  Vh rho;
  rho[]=X;
  varf dVolume(u,v) = int2d(Th)(rho*rho*sin(y)*v);
  real[int] grad = dVolume(0,Vh);
  return grad;
}

matrix hessianV;                                          //   缓冲区域
func matrix HessianVolume(real[int] &X)
{
  Vh rho;
  rho[]=X;
  varf d2Volume(w,v) = int2d(Th)(2*rho*sin(y)*v*w);
  hessianV = d2Volume(Vh,Vh);
  return hessianV;
}
```

如果我们想使体积成为一个约束函数，那么我们必须将它以及它的导函数封装进FreeFem++函数中，并且使它们返回合适的类型。在上述的函数中，如果有人想将它用作适应函数，那么返回合适的类型——这一点并没有完成。由于体积关于ρ并不是线性的，即它有非空的二阶导数，所以拉格朗日海森矩阵同样需要被封装。

```
func real[int] ipVolume(real[int] &X) {real[int] vol = [Volume(X)]; return vol;}

matrix mdV;                                                    //   缓冲区域
func matrix ipGradVolume(real[int] &X)
{                                                              //   将向量转换成稀疏矩阵
  real[int,int] dvol(1,Vh.ndof);
  dvol(0,:)=GradVolume(X);
  mdV=dvol;
  return mdV;
}

matrix HLagrangian;                                            //   缓冲区域
func matrix ipHessianLag(real[int] &X,real objfact,real[int] &lambda)
{
  HLagrangian = objfact*HessianArea(X) + lambda[0]*HessianVolume(X);
  return HLagrangian;
}
```

由于梯度向量会被转换成稀疏矩阵，因此一些空系数会被抛弃，这使得在优化时ipGradVolume函数会产生一些问题。在此，我们会给出内点法以及使用checkindex 命名参数去避免拷贝时的坏指数情况。梯度矩阵事实上是稠密的，因此让其中一些部分恒为零是不合理的：

```
                                                               //   稠密向量的稀疏结构
int[int] gvi(Vh.ndof),gvj=0:Vh.ndof-1;
gvi=0;                                                         //   只有一行
```

这两个数列在内点法中可以通过编译，结构申明为structjacc=[gvi,gvj]。最后剩下的一件事就是定义边界。简单的下界与ρ_{object}的P1元相等。我们将取$\alpha \in [0,1]$，并令$\mathcal{V}_{\max}$为$(1-\alpha)\mathcal{V}(\rho_{object}) + \alpha\frac{4}{3}\pi\|\rho_{\text{object}}\|_\infty^3$：

```
real e=0.1,r0=0.25,rr=2-r0;
real E=1./(e*e),RR=1./(rr*rr);
                                                               //   带齿圆盘
func disc1 = sqrt(1./(RR+(E-RR)*cos(y)*cos(y)))*(1+0.1*cos(9*x));
                                                               //   标准圆盘
func disc2 = sqrt(1./(RR+(E-RR)*cos(x)*cos(x)*sin2))  ;
Vh lb =  max(disc1, disc2);                                    //   粘合目标区域
real Vobj = Volume(lb[]);                                      //   目标体积
real Vnvc = 4./3.*pi*pow(lb[].linfty,3);                       //   V没有体积限制
real alpha=0.1;
Plot3D(lb[],"object_inside",0);
real[int] clb=0.,cub=[(1-alpha)*Vobj + alpha*Vnvc];
```

调用IPOPT：

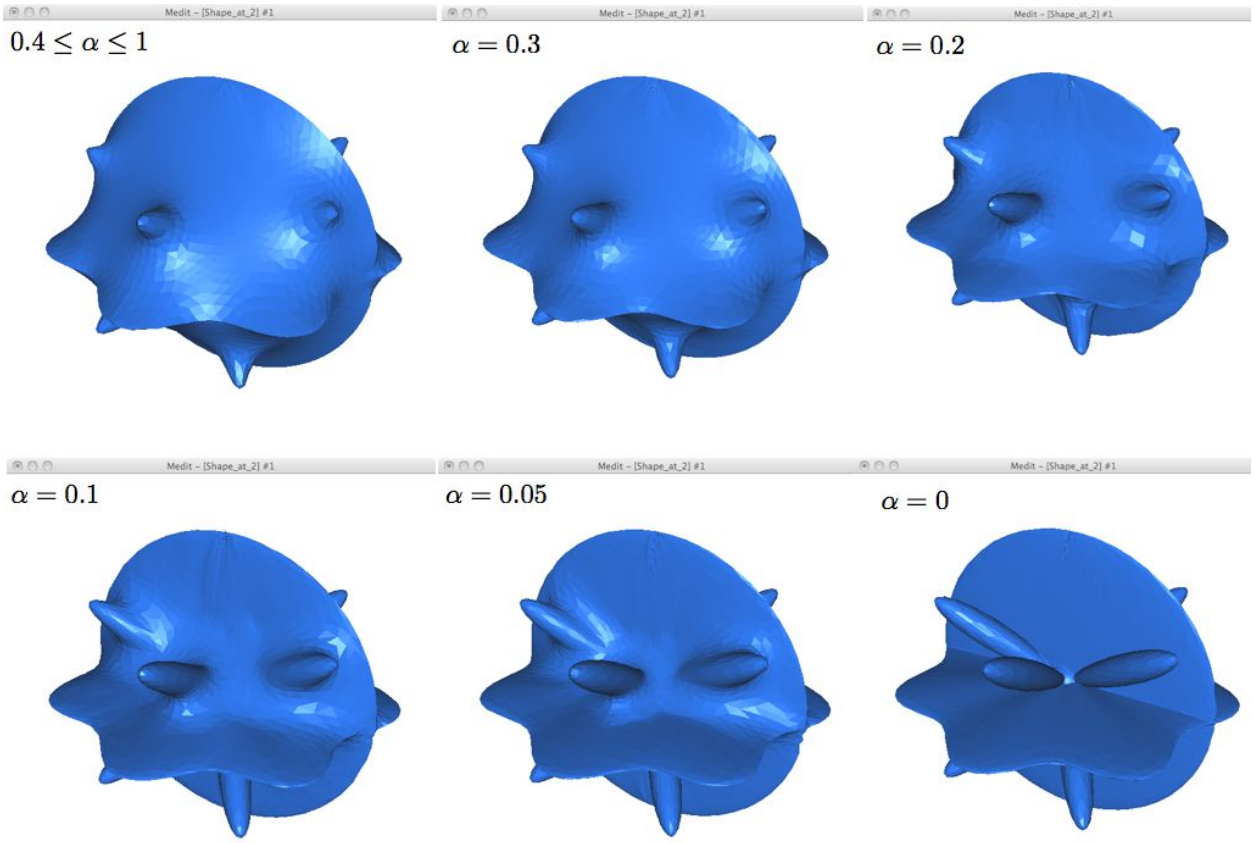
```
IPOPT(Area,GradArea,ipHessianLag,
      ipVolume,ipGradVolume,rc[],                              //   函数以及起始点
      ub=ub[],lb=lb[],clb=clb,cub=cub,                         //   简单边界以及体积边界
      checkindex=1,structjacc=[gvi,gvj],                       //   安全的矩阵拷贝
      maxiter=kkk<nadapt-1 ? 40:150,    //   只有在最后一次网格更新迭代时给出精确优化
      warmstart=kkk,lm=lm,uz=uz[],lz=lz[],                     //   预处理
      tol=0.00001);

Plot3D(rc[],"Shape_at_"+kkk,0);                                //   显示当前解
```

最后，在结束网格生成循环前，我们必须去实现前面所说的更改。改变的网格与$X=(\rho,0,0)$（在球面坐标上）向量场有关，并不直接与ρ有关，否则会影响3D图形的曲率。

```
if(kkk<nadapt-1)
{
  Th = adaptmesh(Th,
         rc*cos(x)*sin(y),                                    //   X
         rc*sin(x)*sin(y),                                    //   Y
         rc*cos(y),                                           //   Z
         nbvx=50000,
         periodic=[[2,y],[4,y]]);                  //   保持网格周期性
  plot(Th);
  startshape = rc;                                  //   在新网格下形状的改变
  uz=uz;                                            //   对偶变量的改写
  lz=lz;
}                                                          //   结束if
}                                                   //   结束网格更新循环
```

以下是随着α的递减所观测得到的图像（图形比正交的两张光盘略微复杂一些）。我们附上了$\alpha = 0$时的图像：



8.6 nlOpt 最优化

ff-NLopt包向FreeFem提供了免费的/开源的非线性优化库。因此，这使得使用各种不同的优化途径求解PDE变得更为简单。所有算法可以在[42]中查阅，所以在此我们并不关注于它们数学方面

的详细分析，而是关注于FreeFem版本中这些算法如何被调用的问题。当读者需要知道这些算法的详细分析时，建议读者去查阅关于它们所罗列出的引用文献（就像很多书目链接一样）。大部分基于梯度的非线性算法在实现时使用的是整个海森矩阵的近似。因此，如果你准备求解一个大规模问题，我们的建议是使用内点法。内点法的效率大大超越我们刚才所说的那些算法。最后，我们在VarIneq2.edp 中提供一个实例examples++-load/。
所有的非线性函数的调用如下：

```
load "ff-NLopt"
...                                                          //   定义J, u, J的梯度等限制
real min = nloptXXXXXX(J,u,                                     //   无法避免的部分
                       grad = <name of grad(J)> ,               //   如果必需的话
                       lb =                                        //   下界数列
                       ub=                                         //   上界数列
                       ...                                      //     可选参数:
                                                                //     限制函数名,
                                                                 //     终止条件,
                                                             //     算法特定的参数,
                                                                    //     等等
                       );
```

XXXXXX指的是算法标签(6个字符长并不是必要的)。u是起始位置(一个实整形的数列)，这个在算法中将被重置。在结束时，该值被所找到的最小值代替。像往常一样，J是一个函数，返回值是一个实整形。grad，lb以及ub是半可选参数。在某种程度上，它们对于某些规则适用，但并不对所有适用。

可能的可选参数命名如下。需要注意的是不是所有算法都使用他们（有些并不支持某些限制，或者某一类型的限制，一些是基于梯度的，有些是与导函数无关的，等等）。用户可以查阅参数描述，确定命名的参数是否被特定的算法支持。使用未被支持的参数会使得计算时产生死循环，这将被程序忽略。也就说，当你很显然误用了某个实现，你将会在运行时收到警告消息（例如，你传入梯度给一个不使用导函数的算法，或者对非遗传算法设置种群，等等）。在下述的描述中，n表示搜索空间的维数。

半可选参数:

grad= 该函数名指计算函数的梯度（原型是real[int] → real[int]，所有参数和结果的维数是n）。在基于梯度的算法中这是必须的，但是在不包括导函数的算法中将会被忽略。

lb/ub = n维数列（实整型）的上下界。在定义变量的搜索区域时使用。这对于某些算法是必须的，某些是可选的，其他情况则不支持。

subOpt : 只在对于拉格朗日参数法和MLSL方法时适用，这两个算法都基于子问题的求解。调用时，只需传递局部算法标签string 就行。

有限制的参数（可选 - 如果不是特定的则无法使用）:

IConst/EConst : 允许传递函数名实现搜索区域上的不等式限制。函数类型必须是real[int] → real[int]，返回数列的大小与限制的个数相同(相同的类型 - 也就是说所有的限制是在一个向量值函数中参与计算)。为了使得等式限制和不等式限制能够在相同的方法中求解，两个向量值函数将被定义与传递。关于如何处理这些限制，更详细的步骤

可见例??。

gradIConst/gradEConst： 提供了不等式限制的梯度。这些是函数是`real[int]` → `real[int,int]`的。假设我们有p个限制函数，则返回矩阵的大小一定是$p \times n$(矩阵的第i行就是第i个限制的梯度)。这在基于梯度的方法中是必要的。

tolIConst/tolEConst： 对每个限制的容忍值。这是一个大小与不等式限制个数相同的数列。对于每个限制的缺省值是10^{-12}。

终止条件：

stopFuncValue： 当目标函数达到特定的实值时，算法结束。

stopRelXTol： 在搜索区域中，当步长的相对移动在每一个方向都小于特定的实值，算法结束。

stopAbsXTol： 在搜索区域中，当步长的绝对移动在没一个方向上都小于特定的实值，算法结束。

stopRelFTol： 当目标函数的相对变化值小于特定的实值时，算法结束。

stopAbsFTol： 当目标函数的绝对变化值小于特定的实值时，算法结束。

stopMaxFEval： 当所有的适应评估函数都满足时，算法结束。

stopTime： 当优化时间第二次超出特定的值时，算法结束。这并不是一个严格的上界：实际运行时间会超出该值一点，这取决于算法以及评估函数的运行速度。

注意当使用AUGLAG或者MLSL方法时，主算法和它的次级算法可能有不同的终止条件。因此，当我们处理这种算法时，可以采用以下参数命名方式（在次级算法前加前缀SO）以及为次级算法设置终止条件：`SOStopFuncValue`， `SOStopRelXTol`，等等。如果以上这些未被使用，则次级算法将使用主算法的终止条件。

其他命名参数：

popSize： 这是个整型，用来改变随机搜索方法的样本大小。缺省值为对于特定的算法具有启发意义的值。

SOPopSize： 与上述相同，区别是在当随机搜索传递给主算法时。

nGradStored： 所有的梯度记录上一次的优化结果：这会增加内存的需求，但同时也会增加收敛速度。这会设置成一个具有启发意义的缺省值。当我们使用AUGLAG或者MLSL时，这指挥影响给出的次级算法。

以下表格总结了每一个算法的主要特征。这提供给了我们关于算法的重要信息，使我们知道支持哪些特征被算法支持，而不支持哪些。更多详尽的内容可以在[42]中查阅。

Id Tag	Full Name	Bounds	Gradient -Based	Stochastic	Constraints		Sub-Opt
					Equality	Inequality	
DIRECT	Dividing rectangles	●					
DIRECTL	Locally biased dividing rectangles	●					
DIRECTLRand	Randomized locally biased dividing rectangles	●					
DIRECTNoScal	Dividing rectangles - no scaling	●					
DIRECTLNoScal	Locally biased dividing rectangles - no scaling	●					
DIRECTLRandNoScal	Randomized locally biased dividing rectangles - no scaling	●					
OrigDIRECT	Original Glabonsky's dividing rectangles	●				✔	
OrigDIRECTL	Original Glabonsky's locally biased dividing rectangles	●				✔	
StoGO	Stochastic(?) Global Optimization	●	●				
StoGORand	Randomized Stochastic(?) Global Optimization	●	●				
LBFGS	Low-storage BFGS		●				
PRAXIS	Principal AXIS	✔					
Var1	Rank-1 shifted limited-memory variable-metric		●				
Var2	Rank-2 shifted limited-memory variable-metric		●				
TNewton	Truncated Newton		●				
TNewtonRestart	Steepest descent restarting truncated Newton		●				
TNewtonPrecond	BFGS preconditionned truncated Newton		●				
TNewtonRestartPrecond	BFGS preconditionned truncated Newton with steepest descent restarting		●				
CRS2	Controled random search with local mutation	✔		●			
MMA	Method of moving asymptots	✔	●			✔	
COBYLA	Constrained optimization by linear approximations	✔			✔	✔	
NEWUOA	NEWUOA						
NEWUOABound	NEWUOA for bounded optimization	✔					
NelderMead	Nelder-Mead simplex	✔					
Sbplx	Subplex	✔					
BOBYQA	BOBYQA	✔					
ISRES	Improved stochastic ranking evolution strategy	✔		●	✔	✔	
SLSQP	Sequential least-square quadratic programming	✔	●		✔	✔	
MLSL	Multi-level single-linkage	✔	●	●			●
MLSLLDS	Low discrepancy multi-level single-linkage	✔	●	●			●
AUGLAG	Constraints augmented lagrangian	✔	●		✔	✔	●
AUGLAGEQ	Equality constraints augmented lagrangian	✔	●		✔	✔	●

Legend :

- ✔ Supported and optional
- ✔ Should be supported and optional, may lead to weird behaviour though.
- ● Intrinsic characteristic of the algorithm which then need one or more unavoidable parameter to work (for stochastic algorithm, the population size always have a default value, they will then work if it is ommited)
- ●/✔ For routines with subsidiary algorithms only, indicates that the corresponding feature will depend on the chosen sub-optimizer.

8.7 带MPI的最优化

在并行体系上使用上述算法的唯一快捷的方式是并行使用成本函数(这是很现实的一件事，是算法中最昂贵的一部分)。在此，我们提供了CMA-ES算法的并行版本。并行运算的核心是琐碎的进化/遗传算法：在每一次迭代中，成本函数将被计算N次，且这N次都是独立的，这N个结果将被同等地分配给每个进程。调用MPI版本的CMA-ES与调用它的有顺序的版本近乎一样(完整的实现可以在文件cmaes-mpi-VarIneq.edp中找到)：

```
load "mpi-cmaes"
...                                                    //   在此定义J, u等参数
real min = cmaesMPI(J,u,stopTolFun=1e-6,stopMaxIter=3000);
cout << "minimal value is " << min << " for u = " << u << endl;
```

在使用popsize参数时，如果种群大小没有改变，那么一个具有启发意义的值将被使用。这个值会变为最优控制器所使用的通讯器中最接近较大乘数的大小。 在FreeFem中，缺省值是mpicommworld。使用者可以使用命名参数" comm="指定特定的MPI通讯器。关于通讯器更多详尽的内容，读者可以在FreeFem++中的MPI 部分进行查阅。

第 9 章

数学模型

摘要　这一章将涉及 *FreeFem++* 所能解决的更深层次的问题。这一章是第*3*章的补充，而第三章仅仅是做了介绍。我们希望使用者们能够贡献你们的例子，使得我们的问题库能变得更加丰富。

9.1　静态问题

9.1.1　肥皂薄膜

我们现在将从一个肥皂薄膜的数学模型开始。这个薄膜粘合在$xy-$平面上的一个圈上。

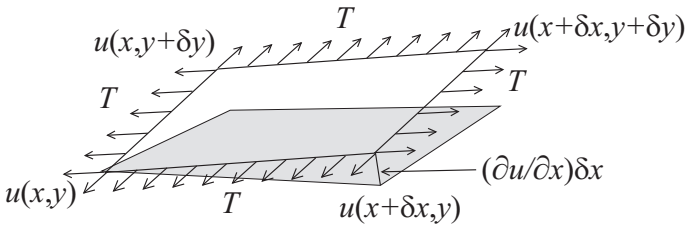
$$C = \{(x, y);\ x = \cos t,\ y = \sin t,\ 0 \le t \le 2\pi\}.$$

我们假设薄膜的形状由图像$(x, y, u(x, y))$所描述，在竖直方向上位移值为 $u(x, y)\,(x^2 + y^2 < 1)$，并承受单位面积压力p，以及起始单位长度张力μ。

考虑一个“小平面”ABCD，A:$(x, y, u(x, y))$, B:$(x, y, u(x + \delta x, y))$, C:$(x, y, u(x + \delta x, y + \delta y))$ and D:$(x, y, u(x, y + \delta y))$。令$\boldsymbol{n}(x, y) = (n_x(x, y), n_y(x, y), n_z(x, y))$是面$z = u(x, y)$的法向量。我们知道竖直方向上的力会由张力$\mu$作用于边AD上。这个力是$-\mu n_x(x, y)\delta y$，而竖直方向上的力作用在AD上为

$$\mu n_x(x + \delta x, y)\delta y \simeq \mu \left(n_x(x, y) + \frac{\partial n_x}{\partial x}\delta x \right)(x, y)\delta y.$$

同样，对于边AB和DC我们有



$$-\mu n_y(x, y)\delta x, \qquad \mu\left(n_y(x, y) + \partial n_y/\partial y\right)(x, y)\delta x.$$

面ABCD在竖直方向上的力由张力μ所得到的的是

$$\mu\left(\partial n_x/\partial x\right)\delta x\delta y + T\left(\partial n_y/\partial y\right)\delta y\delta x.$$

假设只有很小的位移，我们有

$$\begin{aligned}\nu_x &= (\partial u/\partial x)/\sqrt{1+(\partial u/\partial x)^2+(\partial u/\partial y)^2} \simeq \partial u/\partial x,\\ \nu_y &= (\partial u/\partial y)/\sqrt{1+(\partial u/\partial x)^2+(\partial u/\partial y)^2} \simeq \partial u/\partial y.\end{aligned}$$

让$\delta x \to dx$, $\delta y \to dy$，我们由肥皂薄膜在ABCD上竖直位移p的均衡可得

$$\mu dxdy\partial^2 u/\partial x^2 + \mu dxdy\partial^2 u/\partial y^2 + pdxdy = 0.$$

使用Laplace算子$\Delta = \partial^2/\partial x^2 + \partial^2/\partial y^2$，我们可以发现虚位移能够以以下形式写出

$$-\Delta u = f \quad \text{在 } \Omega\text{上} \tag{9.1}$$

其中$f = p/\mu$, $\Omega = \{(x,y);\ x^2+y^2<1\}$。泊松方程(2.1)也在静电学中以$f=\rho/\epsilon$的形式出现，$\rho$是电荷密度，$\epsilon$是静电常数以及$u$被称为导电能力。肥皂薄膜粘合在圈$\partial\Omega = C$上，则我们有边界条件

$$u = 0 \quad \text{在 } \partial\Omega\text{上} \tag{9.2}$$

如果该力是重力，为了简单起见，我们假设$f=-1$。

Example 9.1 (a_tutorial.edp)

```
 1 : border a(t=0,2*pi){ x = cos(t); y = sin(t);label=1;};
 2 :
 3 : mesh disk = buildmesh(a(50));
 4 : plot(disk);
 5 : fespace femp1(disk,P1);
 6 : femp1 u,v;
 7 : func f = -1;
 8 : problem laplace(u,v) =
 9 :     int2d(disk)( dx(u)*dx(v) + dy(u)*dy(v) )                  //     双线性形式
10 :   - int2d(disk)( f*v )                                        //     线性形式
11 :   + on(1,u=0) ;                                              //     边界条件
12 : func ue = (x^2+y^2-1)/4;                                     //     ue:  精确解
13 : laplace;
14 : femp1 err = u - ue;
15 :
16 : plot (u,ps="aTutorial.eps",value=true,wait=true);
17 : plot(err,value=true,wait=true);
18 :
19 : cout << "error L2=" << sqrt(int2d(disk)( err^2) )<< endl;
20 : cout << "error H10=" << sqrt( int2d(disk)((dx(u)-x/2)^2)
21 :                              + int2d(disk)((dy(u)-y/2)^2))<< endl;
22 :
23 : disk = adaptmesh(disk,u,err=0.01);
24 : plot(disk,wait=1);
25 :
26 : laplace;
27 :
28 : plot (u,value=true,wait=true);
29 : err = u - ue;                                                 //     在更新的网格上成为FE函数
30 : plot(err,value=true,wait=true);
31 : cout << "error L2=" << sqrt(int2d(disk)( err^2) )<< endl;
32 : cout << "error H10=" << sqrt(int2d(disk)((dx(u)-x/2)^2)
33 :                              + int2d(disk)((dy(u)-y/2)^2))<< endl;
```

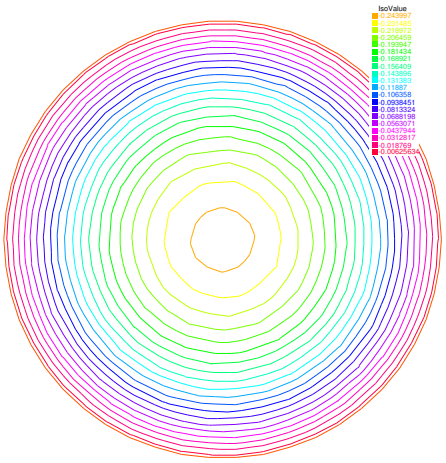


图 9.1: isovalue of u

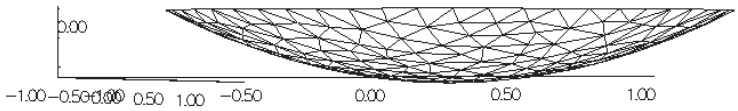


图 9.2: a side view of u

在第19行里, 与精确解u_e的L^2-误差估计为,

$$\|u_h - u_e\|_{0,\Omega} = \left(\int_\Omega |u_h - u_e|^2 \, \mathrm{d}x\mathrm{d}y \right)^{1/2}$$

从第20行到第21行, H^1-误差估计

$$|u_h - u_e|_{1,\Omega} = \left(\int_\Omega |\nabla u_h - \nabla u_e|^2 \, \mathrm{d}x\mathrm{d}y \right)^{1/2}$$

在初始的网格上已经完成. 这个结果是 $\|u_h - u_e\|_{0,\Omega} = 0.000384045$, $|u_h - u_e|_{1,\Omega} = 0.0375506$. 在网格自适应后, 我们得到 $\|u_h - u_e\|_{0,\Omega} = 0.000109043$, $|u_h - u_e|_{1,\Omega} = 0.0188411$. 数值结果可以通过网格的自适应得到改善。

9.1.2 静电场

我们假设不存在与电流和时间无关的电荷分布. 那么电场$\boldsymbol{E}$满足

$$\mathrm{div}\boldsymbol{E} = \rho/\epsilon, \quad \mathrm{curl}\boldsymbol{E} = 0 \tag{9.3}$$

其中 ρ 是电荷密度,ϵ 被称为自由空间的介电常数.从第二个等式 (9.3)里, 我们可以推导出静电势，比如 $\boldsymbol{E} = -\nabla\phi$. 随后我们可以得到泊松方程 $-\Delta\phi = f$, $f = -\rho/\epsilon$. 我们可以得到等势线，也就是ϕ的水平曲线, 在除了导体$\{C_i\}_{1,\cdots,K}$之外没有电荷. 让我们假设K个导体$C_1, \cdots, C_K$在圈C_0中. 每一个都拥有一个静电势φ_i. 我们假设圈$C0$的静电势0. 为了知道在区域Ω里任意一点的$\varphi(x)$, 我们必须求解

$$-\Delta\varphi = 0 \quad \text{in } \Omega, \tag{9.4}$$

其中 Ω 是C_0的内部减去导体C_i, Γ是Ω的边界, 也就是$\sum_{i=0}^{N} C_i$. 在这里g 是x的函数等于φ_i 在C_i上的值并在C_0上趋向于 0. 边界上的等式是如下退化的形式:

$$\varphi = \varphi_i \text{ on } C_i, \ i = 1...N, \varphi = 0 \text{ on } C_0. \tag{9.5}$$

Example 9.2 首先我们给出几何的信息; $C_0 = \{(x,y);\ x^2+y^2 = 5^2\}$, $C_1 = \{(x,y):\ \frac{1}{0.3^2}(x-2)^2 + \frac{1}{3^2}y^2 = 1\}$, $C_2 = \{(x,y):\ \frac{1}{0.3^2}(x+2)^2 + \frac{1}{3^2}y^2 = 1\}$. 令$\Omega$ 是一个圆盘被C_0包围，并带有椭圆型的孔被C_1 和 C_2包围. 注意到 C_0 是被描述成逆时针的, 而椭圆型的孔是被描述为顺时针的, 因为边界都是定向的，因此计算的区域就是面向它的左边.

```
                                                    //    一个圆心为(0 ,0), 半径为5的圆
border C0(t=0,2*pi) { x = 5 * cos(t); y = 5 * sin(t); }
border C1(t=0,2*pi) { x = 2+0.3 * cos(t); y = 3*sin(t); }
border C2(t=0,2*pi) { x = -2+0.3 * cos(t); y = 3*sin(t); }

mesh Th = buildmesh(C0(60)+C1(-50)+C2(-50));
plot(Th,ps="electroMesh");                                          //    图 9.3
fespace Vh(Th,P1);                                                  //    P1 有限元空间
Vh uh,vh;                                                           //    未知量和测试函数.
problem Electro(uh,vh) =                                            //    问题的定义
    int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )                      //    双线性型
    + on(C0,uh=0)                                                   //    C0 上的边界条件
    + on(C1,uh=1)                                                   //    C1 上+1伏特
    + on(C2,uh=-1) ;                                                //    C2 上-1伏特

Electro;                                                            //    求解问题, 解见图 9.4
plot(uh,ps="electro.eps",wait=true);                                //    图 9.4
```

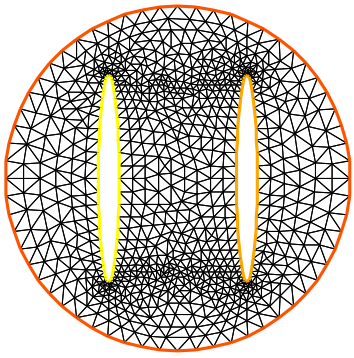


图 9.3: 有两个椭圆孔的圆盘

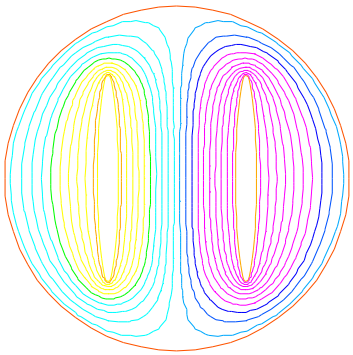


图 9.4: 等势线, C_1 在右边

9.1.3 空气动力学

让我们考虑在一个标准流里的一个翼型的剖面. 无穷大会通过极大的圆 Γ_∞被呈现. 就像之前说的, 我们必须解决

$$\Delta\varphi = 0 \quad \text{in } \Omega, \quad \varphi|_S = c, \quad \varphi|_{\Gamma_\infty} = u_{\infty 1x} - u_{\infty 2x} \tag{9.6}$$

其中Ω是被流体所占据的区域, u_∞是空气在无穷远处的速度, c 是一个待定的常数因此 $\partial_n\varphi$在尾部的边界P of S上是连续的(称为 Kutta-Joukowski 条件). 梯度是平行于c. 为了得到c, 我们用一种重叠的方法. 由于所有在 (9.6) 里的等式都是线性的, 所以得到的解 φ_c 也是c的一个线性函数.

$$\varphi_c = \varphi_0 + c\varphi_1, \tag{9.7}$$

其中 φ_0 是 (9.6) 当 $c=0$时的解, φ_1 是 $c=1$ 且无穷远处速度为0的解. 当这两个场被计算, 我们可以通过要求$\partial\varphi/\partial n$在尾部边界上的连续性来确定 c. 对于NACA0012上表面的等式 (这是在空气动力学里一个经典的翼型剖面; 翼的后部被称为尾部的边缘)是:

$$y = 0.17735\sqrt{x} - 0.075597x - 0.212836x^2 + 0.17363x^3 - 0.06254x^4. \tag{9.8}$$

选取一个可能发生的角度α,比如$\tan\alpha = 0.1$, 我们必须解决

$$-\Delta\varphi = 0 \qquad \text{in } \Omega, \qquad \varphi|_{\Gamma_1} = y - 0.1x, \quad \varphi|_{\Gamma_2} = c, \tag{9.9}$$

其中Γ_2 是翼型的剖面, Γ_1 是无穷远的近似. 寻找到 c 通过解决:

$$-\Delta\varphi_0 = 0 \ \text{ in } \Omega, \qquad \varphi_0|_{\Gamma_1} = y - 0.1x, \quad \varphi_0|_{\Gamma_2} = 0, \tag{9.10}$$

$$-\Delta\varphi_1 = 0 \ \text{ in } \Omega, \qquad \varphi_1|_{\Gamma_1} = 0, \quad \varphi_1|_{\Gamma_2} = 1. \tag{9.11}$$

它的解$\varphi = \varphi_0 + c\varphi_1$允许我们通过 $\partial_n\varphi$ 在尾部边缘$P = (1,0)$ 没有跳跃去找到c. 我们可以得到$\partial n\varphi - (\varphi(P^+) - \varphi(P))/\delta$ 其中P^+ 是在 P的上方且垂直剖面方向上、距离P的长度为δ的一个点. 所以 跳跃 $\partial_n\varphi$ 是 $(\varphi_0|_{P^+} + c(\varphi_1|_{P^+} - 1)) + (\varphi_0|_{P^-} + c(\varphi_1|_{P^-} - 1))$ 除以δ 因为法向在下表面和上表面之间会改变符号. 因此

$$c = -\frac{\varphi_0|_{P^+} + \varphi_0|_{P^-}}{(\varphi_1|_{P^+} + \varphi_1|_{P^-} - 2)}, \tag{9.12}$$

可以被写成程序为:

$$c = -\frac{\varphi_0(0.99, 0.01) + \varphi_0(0.99, -0.01)}{(\varphi_1(0.99, 0.01) + \varphi_1(0.99, -0.01) - 2)}. \tag{9.13}$$

Example 9.3

```
//    NACA0012机翼周围的位势流计算.
//    分解方法是为了运用Joukowski 条件
//    寻找形式为psi0 + beta psi1 的解, 其中, beta 取为
//    使压强在机翼后缘连续的合适的值

border a(t=0,2*pi) { x=5*cos(t);  y=5*sin(t); };                    //    近似无穷

border upper(t=0,1) { x = t;
     y = 0.17735*sqrt(t)-0.075597*t
  - 0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4); }
border lower(t=1,0) { x = t;
     y= -(0.17735*sqrt(t)-0.075597*t
  -0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4)); }
border c(t=0,2*pi) { x=0.8*cos(t)+0.5;  y=0.8*sin(t); }

wait = true;
mesh  Zoom = buildmesh(c(30)+upper(35)+lower(35));
mesh Th = buildmesh(a(30)+upper(35)+lower(35));
fespace Vh(Th,P2);                                                  //    P1 有限元空间
Vh psi0,psi1,vh;                                                    //    未知量和测试函数.
fespace ZVh(Zoom,P2);

solve Joukowski0(psi0,vh) =                                         //    问题的定义
    int2d(Th)( dx(psi0)*dx(vh) + dy(psi0)*dy(vh) )                  //    双线性型
  + on(a,psi0=y-0.1*x)                                              //    边界条件的形式
  + on(upper,lower,psi0=0);
plot(psi0);

solve Joukowski1(psi1,vh) =                                         //    问题的定义
```

```
    int2d(Th)( dx(psi1)*dx(vh) + dy(psi1)*dy(vh) )                    //     双线性型
  + on(a,psi1=0)                                                      //     边界条件的形式
  + on(upper,lower,psi1=1);

plot(psi1);

                                                                      //     压强在机翼后缘的连续性
real beta = psi0(0.99,0.01)+psi0(0.99,-0.01);
beta = -beta / (psi1(0.99,0.01)+ psi1(0.99,-0.01)-2);


Vh psi = beta*psi1+psi0;
plot(psi);
ZVh Zpsi=psi;
plot(Zpsi,bw=true);
ZVh cp = -dx(psi)^2 - dy(psi)^2;
plot(cp);
ZVh Zcp=cp;
plot(Zcp,nbiso=40);
```

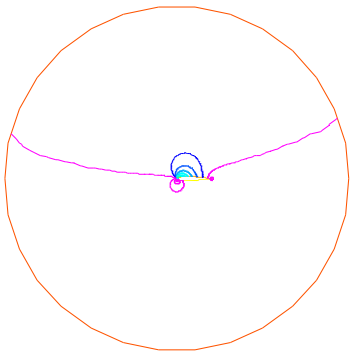


图 9.5: $cp = -(\partial_x\psi)^2 - (\partial_y\psi)^2$的等值线

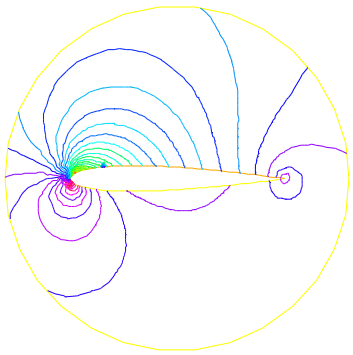


图 9.6: cp的放大

9.1.4 误差估计

对于问题2.1 和 2.2来说，有著名的数值解 u_h 和精确解u的误差估计: 如果三角剖分 $\{\mathcal{T}_h\}_{h\downarrow 0}$ 是正则的 (见 节5.4), 那么我们可以得到估计

$$|\nabla u - \nabla u_h|_{0,\Omega} \leq C_1 h \tag{9.14}$$

$$\|u - u_h\|_{0,\Omega} \leq C_2 h^2 \tag{9.15}$$

其中常数 C_1, C_2 与 h无关, 如果 u 是属于$H^2(\Omega)$. 如果Ω是凸的, 那么则有$u \in H^2(\Omega)$.
在这个部分我们将考察(9.14)和(9.15). 如果使用数值导数的话，我们将碰到数值误差, 对于(9.14)，我们用接下来的等式来处理.

$$\begin{aligned}\int_\Omega |\nabla u - \nabla u_h|^2 \,\mathrm{d}x\mathrm{d}y &= \int_\Omega \nabla u \cdot \nabla(u - 2u_h)\,\mathrm{d}x\mathrm{d}y + \int_\Omega \nabla u_h \cdot \nabla u_h \,\mathrm{d}x\mathrm{d}y \\ &= \int_\Omega f(u - 2u_h)\,\mathrm{d}x\mathrm{d}y + \int_\Omega f u_h \,\mathrm{d}x\mathrm{d}y\end{aligned}$$

常数 C_1, C_2 是依赖于 $\mathcal{T}_h$ 和 f, 所以我们通过 FreeFem++ 找到他们. 通常来说, 我们不能得到方程的解 u 是一个初等函数 (见 Section 4.8) 即使增加 spetical functions. 为了替代精确解, 这里我们使用近似解 u_0在 $V_h(\mathcal{T}_h, P_2)$, $h \sim 0$.

Example 9.4

```
 1 : mesh Th0 = square(100,100);
 2 : fespace V0h(Th0,P2);
 3 : V0h u0,v0;
 4 : func f = x*y;                                          //    sin(pi*x)*cos(pi*y);
 5 :
 6 : solve Poisson0(u0,v0) =
 7 :     int2d(Th0)( dx(u0)*dx(v0) + dy(u0)*dy(v0) )                  //    双线性型
 8 :   - int2d(Th0)( f*v0 )                                            //    线性型
 9 :   + on(1,2,3,4,u0=0) ;                                           //    边界条件
10 :
11 : plot(u0);
12 :
13 : real[int] errL2(10), errH1(10);
14 :
15 : for (int i=1; i<=10; i++) {
16 :    mesh Th = square(5+i*3,5+i*3);
17 :    fespace Vh(Th,P1);
18 :    fespace Ph(Th,P0);
19 :    Ph h = hTriangle;                                  //    得到所有三角形的尺寸
20 :    Vh u,v;
21 :    solve Poisson(u,v) =
22 :         int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v) )                  //    双线性型
23 :         - int2d(Th)( f*v )                                       //    线性型
24 :         + on(1,2,3,4,u=0) ;                                     //    边界条件
25 :    V0h uu = u;
26 :    errL2[i-1] = sqrt( int2d(Th0)((uu - u0)^2) )/h[].max^2;
27 :    errH1[i-1] = sqrt( int2d(Th0)( f*(u0-2*uu+uu) ) )/h[].max;
28 : }
29 : cout << "C1 = " << errL2.max <<"("<<errL2.min<<")"<< endl;
30 : cout << "C2 = " << errH1.max <<"("<<errH1.min<<")"<< endl;
```

我们可以猜到 $C_1 = 0.0179253(0.0173266)$和 $C_2 = 0.0729566(0.0707543)$,其中括号里的数字是计算里的最小值.

9.1.5 周期边界条件

我们现在来求解泊松方程

$$-\Delta u = sin(x + \pi/4.) * cos(y + \pi/4.)$$

在正方形$[0, 2\pi]^2$上 满足双周期边界条件:对于所有y,有$u(0, y) = u(2\pi, y)$;对于所有x,有$u(x, 0) = u(x, 2\pi)$. 这些边界条件可以通过定义周期性有限元空间来实现.

Example 9.5 (periodic.edp)

```
mesh Th=square(10,10,[2*x*pi,2*y*pi]);
                                                    //    定义带周期性边界条件的有限元空间
                                          //    标签:  2 和 4 分别为左边和右边, 横坐标为 y
                                           //    1 和 2 分别为左边和右边, 横坐标为 x
```

```
fespace Vh(Th,P2,periodic=[[2,y],[4,y],[1,x],[3,x]]);
 Vh uh,vh;                                                    //    未知量和测试函数.
 func f=sin(x+pi/4.)*cos(y+pi/4.);                             //    右端函数

 problem laplace(uh,vh) =                                      //    问题的定义
    int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )                 //    双线性型
  + int2d(Th)( -f*vh )                                         //    线性型
;

  laplace;                                          //    求解问题 plot(uh); // 看结果
  plot(uh,ps="period.eps",value=true);
```

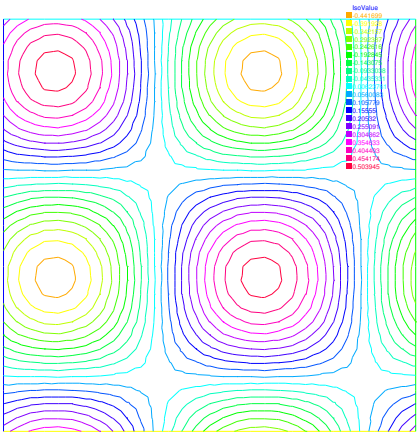


图 9.7: 具有周期性边界条件的解 u 的等值线

周期性条件没有必要要求平行边界. 例 9.6 给出了这样的例子.

Example 9.6 (periodic4.edp)

```
real r=0.25;
                                                              //    带有一个洞的菱形
border a(t=0,1){x=-t+1; y=t;label=1;};
border b(t=0,1){ x=-t; y=1-t;label=2;};
border c(t=0,1){ x=t-1; y=-t;label=3;};
border d(t=0,1){ x=t; y=-1+t;label=4;};
border e(t=0,2*pi){ x=r*cos(t); y=-r*sin(t);label=0;};
int n = 10;
mesh Th= buildmesh(a(n)+b(n)+c(n)+d(n)+e(n));
plot(Th,wait=1);
real r2=1.732;
func abs=sqrt(x^2+y^2);
//    周期性条件的警告:
//    边 a 和 c
//    在边 a 上(标签 1) x ∈ [0,1] or x − y ∈ [−1,1]
//    在边 c 上(标签 3) x ∈ [−1,0] or x − y ∈ [−1,1]
                                                    //    所以共同的横坐标分别为 x 和 x+1
                                                  //    也可以试试曲线横坐标 x − y and x − y
//    1 第一种方法
//    fespace Vh(Th,P2,periodic=[[2,1+x],[4,x],[1,x],[3,1+x]]);
```

```
//     2 第二种方法
 fespace Vh(Th,P2,periodic=[[2,x+y],[4,x+y],[1,x-y],[3,x-y]]);

 Vh uh,vh;

 func f=(y+x+1)*(y+x-1)*(y-x+1)*(y-x-1);
 real intf = int2d(Th)(f);
 real mTh = int2d(Th)(1);
 real k =  intf/mTh;
 problem laplace(uh,vh) =
    int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) ) + int2d(Th)( (k-f)*vh ) ;
 laplace;
 plot(uh,wait=1,ps="perio4.eps");
```

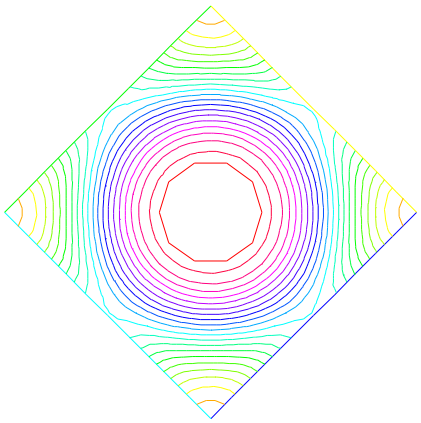


图 9.8: 在 Ω 中求解方程 $\Delta u = ((y+x)^2+1)((y-x)^2+1) - k$, 在洞上 $\partial_n u = 0$, 在外边界上具有两个周期性边界条件, 图为解的等值线

另外一个例子是不相等的边界, 可以看看代码是否工作.

Example 9.7 (periodic4bis.edp)

```
                                                          //     不规则边界条件.
                                                        //     建立边 AB
macro  LINEBORDER(A,B,lab) border A#B(t=0,1){real t1=1.-t;
   x=A#x*t1+B#x*t;y=A#y*t1+B#y*t;label=lab;}                          //    EOM
                                          //     计算 ||AB|| a=(ax,ay) 和 B =(bx,by)
macro dist(ax,ay,bx,by) sqrt(square((ax)-(bx))+ square((ay)-(by)))    //     EOM
macro Grad(u) [dx(u),dy(u)]                                          //    EOM


real Ax=0.9,Ay=1;                 real Bx=2,By=1;
real Cx=2.5,Cy=2.5;               real Dx=1,Dy=2;
real gx = (Ax+Bx+Cx+Dx)/4.;       real gy = (Ay+By+Cy+Dy)/4.;


LINEBORDER(A,B,1)
LINEBORDER(B,C,2)
LINEBORDER(C,D,3)
```

```
LINEBORDER(D,A,4)

int n=10;

real l1=dist(Ax,Ay,Bx,By);
real l2=dist(Bx,By,Cx,Cy);
real l3=dist(Cx,Cy,Dx,Dy);
real l4=dist(Dx,Dy,Ax,Ay);
func s1=dist(Ax,Ay,x,y)/l1;                    //    AB = ||AX||/||AB|| 上的横坐标
func s2=dist(Bx,By,x,y)/l2;                    //    BC = ||BX||/||BC|| 上的横坐标
func s3=dist(Cx,Cy,x,y)/l3;                    //    CD = ||CX||/||CD|| 上的横坐标
func s4=dist(Dx,Dy,x,y)/l4;                    //    DA = ||DX||/||DA|| 上的横坐标

mesh Th=buildmesh(AB(n)+BC(n)+CD(n)+DA(n),fixeborder=1);                    //

verbosity=6;                                          //    观察周期性边界条件的横坐标值.
fespace Vh(Th,P1,periodic=[[1,s1],[3,s3],[2,s2],[4,s4]]);
verbosity=1;
Vh u,v;
real cc=0;
cc= int2d(Th)((x-gx)*(y-gy)-cc)/Th.area;
cout << " compatibility =" << int2d(Th)((x-gx)*(y-gy)-cc) <<endl;

solve Poission(u,v)=int2d(Th)(Grad(u)'*Grad(v)+ 1e-10*u*v)
  -int2d(Th)(10*v*((x-gx)*(y-gy)-cc));
plot(u,wait=1,value=1);
```

Example 9.8 (Period-Poisson-cube-ballon.edp)

```
verbosity=1;
load "msh3"
load "tetgen"
load "medit"

bool buildTh=0;
mesh3 Th;
try {                          //    一种一次建立一个网格的途径, 并且若文件存在, 读取该网格.
  Th=readmesh3("Th-hex-sph.mesh");
 }
catch(...) { buildTh=1;}
if( buildTh ){
  ...
 put the code  example  page                                    //    5.11.1126
  without the first line
 }


fespace Ph(Th,P0);
verbosity=50;
fespace Vh(Th,P1,periodic=[[3,x,z],[4,x,z],[1,y,z],[2,y,z],[5,x,y],[6,x,y]]);//
后边和前边
verbosity=1;
Ph reg=region;

cout << "  centre = " << reg(0,0,0) << endl;
```

```
cout << " exterieur = " << reg(0,0,0.7) << endl;

macro Grad(u) [dx(u),dy(u),dz(u)]                                    //     EOM

Vh uh,vh;
real x0=0.3,y0=0.4,z0=06;
func f= sin(x*2*pi+x0)*sin(y*2*pi+y0)*sin(z*2*pi+z0);
real gn = 1.;
real cf= 1;
problem P(uh,vh)=
     int3d(Th,1)( Grad(uh)'*Grad(vh)*100)
  +  int3d(Th,2)( Grad(uh)'*Grad(vh)*2)
  + int3d(Th) (vh*f)
  ;
  P;
plot(uh,wait=1, nbiso=6);
medit("   uh ",Th, uh);
```

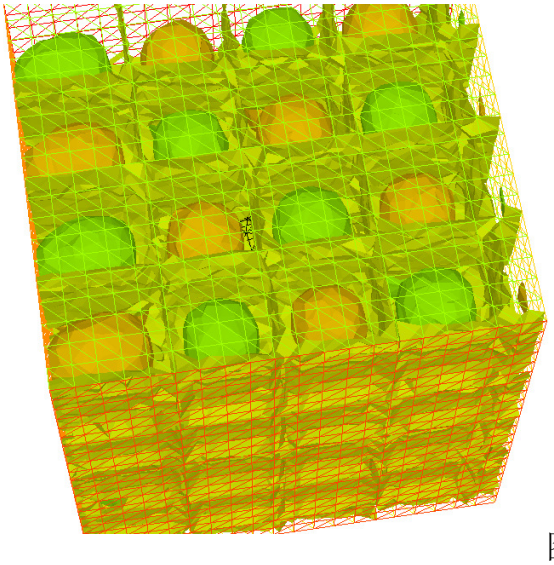


图 9.9: 周期解 uh 的表面等值线

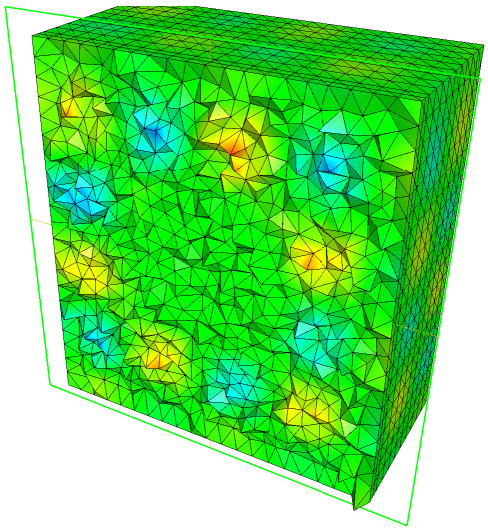


图 9.10:
用 ffmedit 观察解 uh 的一个切面

9.1.6 混合边界条件的Poisson问题

这里我们考虑混合边界条件的泊松问题: 给定函数 f 和 g, 求解 u 使得

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \Gamma_D, \quad \partial u/\partial n = 0 \quad \text{on } \Gamma_N \end{aligned} \tag{9.16}$$

其中 Γ_D 边界 Γ的一部分,以及 $\Gamma_N = \Gamma \setminus \overline{\Gamma_D}$. 问题的解$u$ 在这些点上 $\{\gamma_1, \gamma_2\} = \overline{\Gamma_D} \cap \overline{\Gamma_N}$拥有奇性. 当 $\Omega = \{(x,y);\ -1 < x < 1,\ 0 < y < 1\}$, $\Gamma_N = \{(x,y);\ -1 \le x < 0,\ y = 0\}$, $\Gamma_D = \partial\Omega \setminus \Gamma_N$时, 奇性会出现在$\gamma_1 = (0,0)$, $\gamma_2(-1,0)$上, 并且 u 有如下表达

$$u = K_i u_S + u_R,\ u_R \in H^2(\text{near } \gamma_i),\ i = 1,2$$

其中K_i为常数. 这里$u_S = r_j^{1/2}\sin(\theta_j/2)$ 通过局部极坐标 $(r_j, \theta_j$ at γ_j使得 $(r_1, \theta_1) = (r, \theta)$. 为了代替局部极坐标体系$(r, \theta)$, 我们使用 $r =$ `sqrt( x^2+y^2 )` 和$\theta =$ `atan2(y,x)` 在 `FreeFem++` 里.

Example 9.9 假设 $f = -2\times 30(x^2+y^2)$ 且 $g = u_e = 10(x^2+y^2)^{1/4}\sin\left([\tan^{-1}(y/x)]/2\right)+30(x^2y^2)$, 这里的 u_eS 是精确解。

```
 1 : border N(t=0,1) { x=-1+t; y=0; label=1; };
 2 : border D1(t=0,1){ x=t;  y=0; label=2;};
 3 : border D2(t=0,1){ x=1; y=t; label=2; };
 4 : border D3(t=0,2){ x=1-t; y=1; label=2;};
 5 : border D4(t=0,1) { x=-1; y=1-t; label=2; };
 6 :
 7 : mesh T0h = buildmesh(N(10)+D1(10)+D2(10)+D3(20)+D4(10));
 8 : plot(T0h,wait=true);
 9 : fespace V0h(T0h,P1);
10 : V0h u0, v0;
11 :
12 : func f=-2*30*(x^2+y^2);                                          //    给定方程
13 :                                                  //    特解 K*us (K: 常数)
14 : func us = sin(atan2(y,x)/2)*sqrt( sqrt(x^2+y^2) );
15 : real K=10.;
16 : func ue = K*us + 30*(x^2*y^2);
17 :
18 : solve Poisson0(u0,v0) =
19 :     int2d(T0h)( dx(u0)*dx(v0) + dy(u0)*dy(v0) )              //    双线性形式
20 :   - int2d(T0h)( f*v0 )                                       //    线性形式
21 :   + on(2,u0=ue) ;                                           //    边界条件
22 :
23 :                                                            //    由奇异形式改进
24 : mesh Th = adaptmesh(T0h,us);
25 : for (int i=0;i< 5;i++)
26 : {
27 :   mesh Th=adaptmesh(Th,us);
28 : } ;
29 :
30 : fespace Vh(Th, P1);
31 : Vh u, v;
32 : solve Poisson(u,v) =
33 :     int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v) )                   //    双线性形式
34 :   - int2d(Th)( f*v )                                        //    线性形式
35 :   + on(2,u=ue) ;                                           //    边界条件
36 :
37 : /* 画出解 */
38 : plot(Th,ps="adaptDNmix.ps");
39 : plot(u,wait=true);
40 :
41 : Vh uue = ue;
42 : real  H1e = sqrt( int2d(Th)( dx(uue)^2 + dy(uue)^2 + uue^2 ) );
43 :
44 : /* 计算 H1 Sobolev范数 */
45 : Vh err0 = u0 - ue;
46 : Vh  err = u - ue;
47 : Vh  H1err0 = int2d(Th)( dx(err0)^2+dy(err0)^2+err0^2 );
48 : Vh  H1err = int2d(Th)( dx(err)^2+dy(err)^2+err^2 );
49 : cout <<"Relative error in first mesh "<< int2d(Th)(H1err0)/H1e<<endl;
```

```
50 : cout <<"Relative error in adaptive mesh "<< int2d(Th)(H1err)/H1e<<endl;
```

从第24行到第28行，对网格的改进是基于奇异的项的。在第42行，我们计算 H1e=$\|u_e\|_{1,\Omega}$。 在最后两行，我们在计算相对误差，也就是，

$$\begin{aligned} \|u_h^0 - u_e\|_{1,\Omega}/H1e &= 0.120421 \\ \|u_h^a - u_e\|_{1,\Omega}/H1e &= 0.0150581 \end{aligned}$$

在这里，u_h^0是在 T0h上的数值解，且在这个程序中， u_h^a 是 u。

9.1.7 混合有限元的泊松方程

这一节我们考虑有混合边界条件的泊松方程：对于给定的函数f , g_d, g_n, 找出 p 使得

$$\begin{aligned} -\Delta p &= 1 \quad \text{in } \Omega \\ p &= g_d \quad \text{on} \Gamma_D, \quad \partial p/\partial n = g_n \quad \text{on } \Gamma_N \end{aligned} \tag{9.17}$$

Γ_D 是边界 Γ的一部分，且 $\Gamma_N = \Gamma \setminus \overline{\Gamma_D}$。
混合公式如下： 找到 p 和 $\mathbf{u}$ 使得

$$\begin{aligned} \nabla p + \mathbf{u} &= \mathbf{0} \quad \text{in } \Omega \\ \nabla.\mathbf{u} &= f \quad \text{in } \Omega \\ p &= g_d \quad \text{on } \Gamma_D, \quad \partial u.n = \mathbf{g}_n.n \quad \text{on } \Gamma_N \end{aligned} \tag{9.18}$$

此时 $\mathbf{g}_n$ 是一个使得 $\mathbf{g}_n.n = g_n$的向量。
变分公式是：

$$\begin{aligned} \forall \mathbf{v} \in \mathbb{V}_0, \quad \int_\Omega p\nabla.v + \mathbf{vv} &= \int_{\Gamma_d} g_d \mathbf{v}.n \\ \forall q \in \mathbb{P} \quad \int_\Omega q\nabla.u &= \int_\Omega qf \\ \partial u.n &= \mathbf{g}_n.n \quad \text{on } \Gamma_N \end{aligned} \tag{9.19}$$

这是的函数空间是：

$$\mathbb{P} = L^2(\Omega), \qquad \mathbb{V} = H(div) = \{\mathbf{v} \in L^2(\Omega)^2, \nabla.\mathbf{v} \in L^2(\Omega)\}$$

并且我们可以得到：

$$\mathbb{V}_0 = \{\mathbf{v} \in \mathbb{V}; \quad \mathbf{v}.n = 0 \quad \text{on } \Gamma_N\}.$$

为了写出 FreeFem++ 的代码，我们需要选择有限元空间。 在这里， $\mathbb{V}$ 空间用Raviart-Thomas有限元RT0离散化， $\mathbb{P}$用常数有限元P0离散化。

Example 9.10 (LaplaceRT.edp)

```
mesh Th=square(10,10);
fespace Vh(Th,RT0);
fespace Ph(Th,P0);
func gd = 1.;
func g1n = 1.;
func g2n = 1.;
```

```
Vh [u1,u2],[v1,v2];
Ph p,q;

problem laplaceMixte([u1,u2,p],[v1,v2,q],
                     solver=GMRES,eps=1.0e-10,
                     tgv=1e30,dimKrylov=150)
      =
     int2d(Th)( p*q*1e-15                                    //    这一项在这表示 sur
                                                    //    子矩阵都是可逆的 (需要LU 分解)
              + u1*v1 + u2*v2 + p*(dx(v1)+dy(v2)) + (dx(u1)+dy(u2))*q )
  + int2d(Th) ( q)
  - int1d(Th,1,2,3)( gd*(v1*N.x +v2*N.y))                    //    on ΓD
  + on(4,u1=g1n,u2=g2n);                                     //    on ΓN

 laplaceMixte;

 plot([u1,u2],coef=0.1,wait=1,ps="lapRTuv.eps",value=true);
 plot(p,fill=1,wait=1,ps="laRTp.eps",value=true);
```

9.1.8 度量改进和残量估计子

在这一节中, 我们对Poisson问题做网格的自适应, 并计算单元 T 上的经典残量误差估计子 η_T.

Example 9.11 (adaptindicatorP2.edp) 首先，我们解决之前的例子中出现的问题：

```
 1 : border ba(t=0,1.0){x=t;   y=0;  label=1;};                  //    见图5.15
 2 : border bb(t=0,0.5){x=1;   y=t;  label=2;};
 3 : border bc(t=0,0.5){x=1-t; y=0.5;label=3;};
 4 : border bd(t=0.5,1){x=0.5; y=t;  label=4;};
 5 : border be(t=0.5,1){x=1-t; y=1;  label=5;};
 6 : border bf(t=0.0,1){x=0;   y=1-t;label=6;};
 7 : mesh Th = buildmesh (ba(6) + bb(4) + bc(4) +bd(4) + be(4) + bf(6));
 8 : savemesh(Th,"th.msh");
 9 : fespace Vh(Th,P2);
10 : fespace Nh(Th,P0);
11 : Vh u,v;
12 : Nh rho;
13 : real[int] viso(21);
14 : for (int i=0;i<viso.n;i++)
15 :   viso[i]=10.^(+(i-16.)/2.);
16 : real error=0.01;
17 : func f=(x-y);
18 : problem Probem1(u,v,solver=CG,eps=1.0e-6) =
19 :     int2d(Th,qforder=5)( u*v*1.0e-10+  dx(u)*dx(v) + dy(u)*dy(v))
20 :   + int2d(Th,qforder=5)( -f*v);
21 : /*************
```

因此，局部误差估计子 η_T 是：

$$\eta_T = \left(h_T^2 ||f + \Delta u_h||^2_{L^2(T)} + \sum_{e \in \mathcal{E}_K} h_e || [\frac{\partial u_h}{\partial n_k}] ||^2_{L^2(e)} \right)^{\frac{1}{2}}$$

在这里 h_T 是T的最长边， $\mathcal{E}_T$ 不在$\Gamma = \partial\Omega$上的 T 边界的集合， n_T 是 K的单位外法向量， h_e 是边 e 的长， $[g]$ 是函数g 沿着边界跳跃时的值（用左边的值减去右边）。
我们当然可以使用变分形式去计算η_T^2， 此时测试函数在每个三角形上是常数。

```
29 : *************/
30 :
31 : varf indicator2(uu,chiK) =
32 :      intalledges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
33 :     +int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))) );
34 : for (int i=0;i< 4;i++)
35 : {
36 :   Probem1;
37 :    cout << u[].min << " " << u[].max << endl;
38 :    plot(u,wait=1);
39 :    cout << " indicator2 " << endl;
40 :
41 :    rho[] = indicator2(0,Nh);
42 :    rho=sqrt(rho);
43 :    cout << "rho =   min " << rho[].min << " max=" << rho[].max << endl;
44 :    plot(rho,fill=1,wait=1,cmm="indicator density ",ps="rhoP2.eps",
                                        value=1,viso=viso,nbiso=viso.n);
45 :    plot(Th,wait=1,cmm="Mesh ",ps="ThrhoP2.eps");
46 :    Th=adaptmesh(Th,[dx(u),dy(u)],err=error,anisomax=1);
47 :    plot(Th,wait=1);
48 :    u=u;
49 :    rho=rho;
50 :   error = error/2;
51 : } ;
```

如果这个方法是对的，那么下图中就能展示几乎是常数的函数 η *Fig. 9.11.*

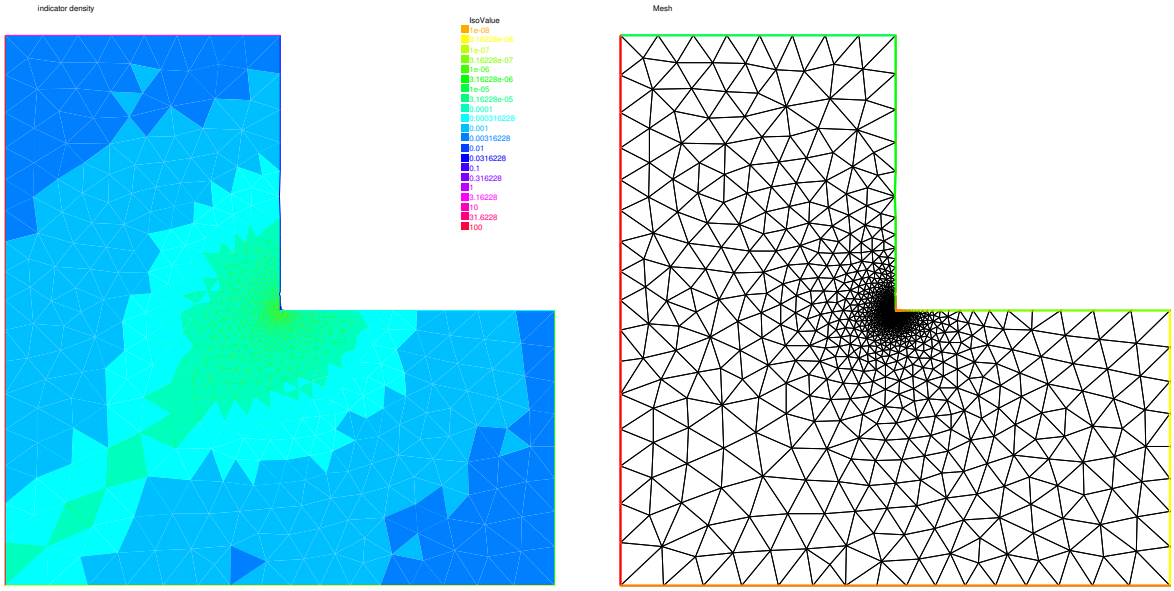


图 9.11: 误差指示子的密度， 在各项同性的 P_2 度量下

9.1.9 使用残量指示子改进

在之前的例子中，我们计算了误差指示子，现在我们使用它改进网格。
新的网格大小由如下公式给出：

$$h_{n+1}(x) = \frac{h_n(x)}{f_n(\eta_K(x))}$$

此时 $\eta_n(x)$ 是由局部误差指示子给出的在点x处的误差界， h_n 是之前的“网格”， f_n 是用户函数，定义为 $f_n = min(3, max(1/3, \eta_n/\eta_n^*))$ 此时 $\eta_n^* = mean(\eta_n)c$, 且 c 是一个接近1的用户指定的参数。

Example 9.12 (AdaptResidualErrorIndicator.edp)
首先，使用宏命令 *MeshSizecomputation* 计算出 P_1 网格大小，作为边长的平均值。

```
//     使用宏得到当前的网格大小
//     参数
//     输入:  Th 网格
//     Vh P1 有限元空间 on Th
//     输出 :
//     h:  Vh 有限元，当前网格大小的有限集合
macro  MeshSizecomputation(Th,Vh,h)
{  /* 网格    Vh  P1 有限元空间
 h    P1 网格大小的值 */
real[int]  count(Th.nv);
/* 网格大小  (lenEdge =  integral(e) 1 ds)  */
varf vmeshsizen(u,v)=intalledges(Th,qfnbpE=1)(v);
/* 边数 / par vertex */
varf vedgecount(u,v)=intalledges(Th,qfnbpE=1)(v/lenEdge);
   /*
  计算网格大小
  ---------------------------- */
count=vedgecount(0,Vh);
h[]=0.;
h[]=vmeshsizen(0,Vh);
cout << " count min = "<< count.min << " " << count.max << endl;
h[]=h[]./count;
    cout << " -- bound meshsize = " <<h[].min << " " << h[].max << endl;
} //    停止宏
```

第二条宏命令根据新产生的网格大小重新打网格。

```
//     根据残量指示子进行宏，网格重构
//     输入:
//     Th the mesh
//     Ph P0 fespace on Th
//     Vh P1 fespace on Th
//     vindicator the varf of to evaluate the indicator to 2
//     coef on etamem..
//     ------

macro  ReMeshIndicator(Th,Ph,Vh,vindicator,coef)
{
Vh h=0;
/*计算网格大小  */
MeshSizecomputation(Th,Vh,h);
Ph etak;
```

```
etak[]=vindicator(0,Ph);
etak[]=sqrt(etak[]);
real etastar= coef*(etak[].sum/etak[].n);
cout << " etastar = " << etastar << " sum=" << etak[].sum << " " << endl;

/* 这里 etaK 是不连续的;
   我们用带质量集中法的P1元L2投影 。 */

Vh fn,sigma;
varf veta(unused,v)=int2d(Th)(etak*v);
varf vun(unused,v)=int2d(Th)(1*v);
fn[]  = veta(0,Vh);
sigma[]= vun(0,Vh);
fn[]= fn[]./ sigma[];
fn =  max(min(fn/etastar,3.),0.3333) ;

/* 新的网格大小 */
h = h / fn ;
/* 画图(h,wait=1); */
/*  建立新网格 */
Th=adaptmesh(Th,IsMetric=1,h,splitpbedge=1,nbvx=10000);
}
```

在这里我们省略网格构建的步骤，详见前面的例子。

```
                                                          //    有限元空间定义 ---
fespace Vh(Th,P1);                                        //    网格大小和解
fespace Ph(Th,P0);                                           //    误差指标

real hinit=0.2;                                           //    初始网格大小
Vh   h=hinit;                                          //    对网格计算有限元方程
                                                 //    基于给定的网格大小组建网格：网格大小
Th=adaptmesh(Th,h,IsMetric=1,splitpbedge=1,nbvx=10000);
plot(Th,wait=1,ps="RRI-Th-init.eps");
Vh u,v;

func f=(x-y);

problem Poisson(u,v) =
    int2d(Th,qforder=5)( u*v*1.0e-10+  dx(u)*dx(v) + dy(u)*dy(v))
  - int2d(Th,qforder=5)( f*v);

 varf indicator2(unused,chiK) =
     intalledges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
    +int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))) );

for (int i=0;i< 10;i++)
{
u=u;
Poisson;
plot(Th,u,wait=1);
real cc=0.8;
if(i>5) cc=1;
ReMeshIndicator(Th,Ph,Vh,indicator2,cc);
plot(Th,wait=1);
}
```

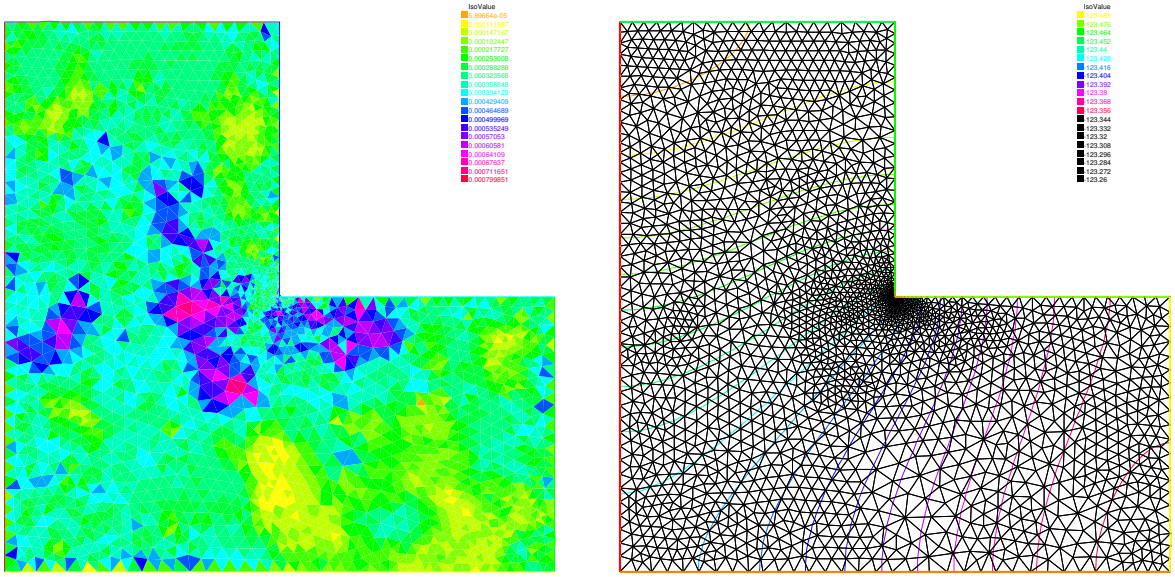


图 9.12: 各项同性P_1，误差指标；解的网格和等值面

9.2 弹性问题

考虑一个能形变的弹性板 $\Omega \times [-h,h]$ in $\mathbb{R}^3$, $\Omega \subset \mathbb{R}^2$. 由于板子产生形变，我们考虑从点 $P(x_1,x_2,x_3)$ 到点 $\mathcal{P}(\xi_1,\xi_2,\xi_3)$的位移。 向量$\boldsymbol{u} = (u_1,u_2,u_3) = (\xi_1 - x_1, \xi_2 - x_2, \xi_3 - x_3)$ 被称为 位移向量. 由于产生形变， 线段 $\overline{\mathbf{x},\mathbf{x}+\tau\Delta\mathbf{x}}$ 大致移动到 $\mathbf{x}+u(\mathbf{x}),\mathbf{x}+\tau\Delta\mathbf{x}+u(\mathbf{x}+\tau\Delta\mathbf{x})$ ，这里的 τ是个很小的数， 这里的 $\mathbf{x} = (x_1,x_2,x_3)$, $\Delta\mathbf{x} = (\Delta x_1, \Delta x_2, \Delta x_3)$。 我们可以计算出两条线段的比：

$$\eta(\tau) = \tau^{-1}|\Delta\mathbf{x}|^{-1}\left(|u(\mathbf{x}+\tau\Delta\mathbf{x}) - u(\mathbf{x}) + \tau\Delta\mathbf{x}| - \tau|\Delta\mathbf{x}|\right)$$

从而我们可以推导出 (详见 如 [16, p.32])

$$\lim_{\tau\to 0}\eta(\tau) = (1+2e_{ij}\nu_i\nu_j)^{1/2} - 1, \quad 2e_{ij} = \frac{\partial u_k}{\partial x_i}\frac{\partial u_k}{\partial x_j} + \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

这里 $\nu_i = \Delta x_i|\Delta\mathbf{x}|^{-1}$。如果形变很 小, 那么我们可以认为

$$(\partial u_k/\partial x_i)(\partial u_k/\partial x_i) \approx 0$$

那么如下公式被称为 小 压力应变张量

$$\varepsilon_{ij}(u) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

张量 e_{ij} 被称为 有限压力应变张量.
考虑一个以$\mathbf{x}$为中心的平面 $\Delta\Pi(\mathbf{x})$，其单位法向量 $\boldsymbol{n} = (n_1,n_2,n_3)$，曲面 $\Delta\Pi(\mathbf{x})$ 在点 $\mathbf{x}$ 处的坐标表示：

$$(\sigma_{1j}(\mathbf{x})n_j, \sigma_{2j}(\mathbf{x})n_j, \sigma_{3j}(\mathbf{x})n_j)$$

此时 $\sigma_{ij}(\mathbf{x})$ 是点$\mathbf{x}$ 处的应力张量。 运用胡克定律，可以在σ_{ij}和ε_{ij}中找到线性关系

$$\sigma_{ij}(\mathbf{x}) = c_{ijkl}(\mathbf{x})\varepsilon_{ij}(\mathbf{x})$$

具有对称性 $c_{ijkl}=c_{jikl}, c_{ijkl}=c_{ijlk}, c_{ijkl}=c_{klij}$。
如果胡克张量 $c_{ijkl}(\mathbf{x})$ 不依赖于坐标系的选取，那么该物质在点$\mathbf{x}$被称为各向同性的。如果c_{ijkl}是常数，那么该物质被称为 同质的。在各向同性且同质的情形下，存在*Lamé*常数 λ,μ （详见，如，[16, p.43]）满足：

$$\sigma_{ij}=\lambda\delta_{ij}\mathrm{div}u+2\mu\varepsilon_{ij} \tag{9.20}$$

此时 δ_{ij} 是Kronecker积。我们假设弹性板是在区域$\Gamma_D\times[-h,h], \Gamma_D\subset\partial\Omega$上。如果体积力 $f=(f_1,f_2,f_3)$ 在区域$\Omega\times[-h,h]$上给出。表面力 g 在区域 $\Gamma_N\times[-h,h], \Gamma_N=\partial\Omega\setminus\overline{\Gamma_D}$上给出， 那么平衡方程为：

$$\begin{aligned} -\partial_j\sigma_{ij} &= f_i \ \text{ in } \Omega\times[-h,h], \quad i=1,2,3 && (9.21)\\ \sigma_{ij}n_j &= g_i \ \text{ on } \Gamma_N\times[-h,h], \quad u_i=0 \ \text{ on } \Gamma_D\times]-h,h[, \quad i=1,2,3 && (9.22)\end{aligned}$$

我们现在来解释平面弹性：

平面应变： 在平面的底端，接触条件满足$u_3=0$, $g_3=$。在这种情形下，我们可以假定 $f_3=g_3=u_3=0$ 且 $\boldsymbol{u}(x_1,x_2,x_3)=\overline{u}(x_1,x_2)$ 对于所有的 $-h<x_3<h$。

平面应力： 假设圆柱非常细，在两端 $x_3=\pm h$没有负载，即

$$\sigma_{3i}=0, \quad x_3=\pm h, \quad i\ 1,2,3$$

这个假设导出 $\sigma_{3i}=0$ 在 $\Omega\times[-h,h]$ 且 $\boldsymbol{u}(x_1,x_2,x_3)=\overline{u}(x_1,x_2)$ 对于所有 $-h<x_3<h$。

广义平面应力： 圆柱在两端$x_3=\pm h$无负载。 引入基于厚度的平均值，

$$\overline{u}_i(x_1,x_2)=\frac{1}{2h}\int_{-h}^{h}u(x_1,x_2,x_3)dx_3$$

我们令 $\overline{u}_3\equiv 0$。类似地，我们定义体积力和表面力的平均值$\overline{f},\overline{g}$ ，还有应力和应变分量的平均值 $\overline{\varepsilon}_{ij}$ 和 $\overline{\sigma}_{ij}$。

接下来我们省略$\overline{u},\overline{f},\overline{g},\overline{\varepsilon}_{ij}$和 $\overline{\varepsilon}_{ij}$ 的上划线。 我们能够得到由(9.21)导出的平衡方程，将 $\Omega\times[-h,h]$ 替换成 Ω 并且改变 $i=1,2$。 考虑到平面应力： $\sigma_{ij}=\lambda^*\delta_{ij}\mathrm{div}u+2\mu\varepsilon_{ij}, \lambda^*=(2\lambda\mu)/(\lambda+\mu)$。
弹性方程被自然地写成变分形式，我们用位移矢量： $u(x)\in V$

$$\int_\Omega[2\mu\epsilon_{ij}(\boldsymbol{u})\epsilon_{ij}(\boldsymbol{v})+\lambda\epsilon_{ii}(\boldsymbol{u})\epsilon_{jj}(\boldsymbol{v})]=\int_\Omega\boldsymbol{f}\cdot\boldsymbol{v}+\int_\Gamma\boldsymbol{g}\cdot\boldsymbol{v}, \forall\boldsymbol{v}\in V$$

V 是 $H^1(\Omega)^2$的线性闭子空间。

Example 9.13 (Beam.edp) 考虑有不变矩形边界$[0,10]\times[0,2]$的弹性板： 体积力是引力 $\boldsymbol{f}$ *and the* 边界力 $\boldsymbol{g}$ 在上界和下界都是*0*。 垂直的两边是固定的。

```
//      带有重量的边，它的在如下边界上：

int bottombeam = 2;
border a(t=2,0)  { x=0; y=t ;label=1;};                    //    左边
border b(t=0,10) { x=t; y=0 ;label=bottombeam;};            //    底边
border c(t=0,2)  { x=10; y=t ;label=1;};                    //    右边
border d(t=0,10) { x=10-t; y=2; label=3;};                  //    顶边
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
```

```
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
fespace Vh(th,[P1,P1]);
Vh [uu,vv], [w,s];
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;
                                                    //    由于本身重量，边产生的变形
real sqrt2=sqrt(2.);                                //    见 lame.edp 作为例子 3.8
macro epsilon(u1,u2)  [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2]          //    EOM
macro div(u,v) ( dx(u)+dy(v) )                                      //    EOM

solve bb([uu,vv],[w,s])=
        int2d(th)(
                  lambda*div(w,s)*div(uu,vv)
                  +2.*mu*( epsilon(w,s)'*epsilon(uu,vv) )
                 )
  + int2d(th) (-gravity*s)
  + on(1,uu=0,vv=0)
 ;

plot([uu,vv],wait=1);
plot([uu,vv],wait=1,bb=[[-0.5,2.5],[2.5,-0.5]]);
mesh th1 = movemesh(th, [x+uu, y+vv]);
plot(th1,wait=1);
```

Example 9.14 (beam-3d.edp) 考虑弹性体，有固定的边界，是平行六面体$[0,5]\times[0,1]\times[0,1]$. 体积力是引力 $\boldsymbol{f}$ 边界力 $\boldsymbol{g}$ 除了在左边被固定的垂直面以外都是0。

```
include "cube.idp"
int[int]  Nxyz=[20,5,5];
real [int,int]  Bxyz=[[0.,5.],[0.,1.],[0.,1.]];
int [int,int]  Lxyz=[[1,2],[2,2],[2,2]];
mesh3 Th=Cube(Nxyz,Bxyz,Lxyz);

real E = 21.5e4, sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;

fespace Vh(Th,[P1,P1,P1]);
Vh [u1,u2,u3], [v1,v2,v3];
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;

real sqrt2=sqrt(2.);
macro epsilon(u1,u2,u3)  [dx(u1),dy(u2),dz(u3),(dz(u2)+dy(u3))/sqrt2,
                      (dz(u1)+dx(u3))/sqrt2,(dy(u1)+dx(u2))/sqrt2]   //    EOM
macro div(u1,u2,u3) ( dx(u1)+dy(u2)+dz(u3) )                        //    EOM

solve Lame([u1,u2,u3],[v1,v2,v3])=
  int3d(Th)(
    lambda*div(u1,u2,u3)*div(v1,v2,v3)
    +2.*mu*( epsilon(u1,u2,u3)'*epsilon(v1,v2,v3) )                 //    ')
      )
  - int3d(Th) (gravity*v3)
```

```
  + on(1,u1=0,u2=0,u3=0)
  ;
real dmax= u1[].max;
cout << " max displacement = " << dmax << endl;
real coef= 0.1/dmax;
int[int] ref2=[1,0,2,0];
mesh3 Thm=movemesh3(Th,transfo=[x+u1*coef,y+u2*coef,z+u3*coef],label=ref2);
Thm=change(Thm,label=ref2);
plot(Th,Thm, wait=1,cmm="coef  amplification = "+coef );          //    见 fig ??
```

9.2.1 断裂力学

考虑有断裂的平面，它的不变形状是一条曲线Σ，有两条边 γ_1, γ_2。 我们假设应力张量σ_{ij}是$(x,y) \in \Omega_\Sigma = \Omega \setminus \Sigma$的平面应力。 在这里$\Omega$表示没有断裂的弹性板的固定边界。如果边界$\partial\Omega$上的一部分$\Gamma_N$是固定的，给定负载$\mathcal{L} = (\boldsymbol{f}, \boldsymbol{g}) \in L^2(\Omega)^2 \times L^2(\Gamma_N)^2$ ，那么位移矢量 $\boldsymbol{u}$ 是势能函数的最小值：

$$\mathcal{E}(\boldsymbol{v}; \mathcal{L}, \Omega_\Sigma) = \int_{\Omega_\Sigma} \{w(x,\boldsymbol{v}) - \boldsymbol{f}\cdot\boldsymbol{v}\} - \int_{\Gamma_N} \boldsymbol{g}\cdot\boldsymbol{v}$$

在函数空间 $V(\Omega_\Sigma)$:

$$V(\Omega_\Sigma) = \left\{\boldsymbol{v} \in H^1(\Omega_\Sigma)^2;\ \boldsymbol{v} = 0 \quad \text{on } \Gamma_D = \partial\Omega \setminus \overline{\Gamma_N}\right\},$$

此时 $w(x,\boldsymbol{v}) = \sigma_{ij}(\boldsymbol{v})\varepsilon_{ij}(\boldsymbol{v})/2$,

$$\sigma_{ij}(\boldsymbol{v}) = C_{ijkl}(x)\varepsilon_{kl}(\boldsymbol{v}), \quad \varepsilon_{ij}(\boldsymbol{v}) = (\partial v_i/\partial x_j + \partial v_j/\partial x_i)/2, \qquad (C_{ijkl}: \quad \text{胡克张量}).$$

如果弹性板是同质且各向同性的，那么位移矢量在γ_k 的一个开邻域U_k中可以分解为（详见 [17]）

$$\boldsymbol{u}(x) = \sum_{l=1}^{2} K_l(\gamma_k) r_k^{1/2} S_{kl}^C(\theta_k) + \boldsymbol{u}_{k,R}(x) \quad \text{for } x \in \Omega_\Sigma \cap U_k,\ k = 1,2 \tag{9.23}$$

$\boldsymbol{u}_{k,R} \in H^2(\Omega_\Sigma \cap U_k)^2$， U_k, $k = 1,2$ 是 γ_k 的开邻域，使得 $\partial L_1 \cap U_1 = \gamma_1$, $\partial L_m \cap U_2 = \gamma_2$, 而且

$$\begin{aligned} S_{k1}^C(\theta_k) &= \frac{1}{4\mu}\frac{1}{(2\pi)^{1/2}} \begin{bmatrix} [2\kappa-1]\cos(\theta_k/2) - \cos(3\theta_k/2) \\ -[2\kappa+1]\sin(\theta_k/2) + \sin(3\theta_k/2) \end{bmatrix}, \\ S_{k2}^C(\theta_k) &= \frac{1}{4\mu}\frac{1}{(2\pi)^{1/2}} \begin{bmatrix} -[2\kappa-1]\sin(\theta_k/2) + 3\sin(3\theta_k/2) \\ -[2\kappa+1]\cos(\theta_k/2) + \cos(3\theta_k/2) \end{bmatrix}. \end{aligned} \tag{9.24}$$

μ 是弹性里的剪切模量， 对于平面应变而言，$\kappa = 3 - 4\nu$ (ν 是泊松比)，对于平面应力而言，$\kappa = \frac{3-\nu}{1+\nu}$。
参数$K_1(\gamma_i)$ 和 $K_2(\gamma_i)$, 是断裂力学中很重要的参数。 它们分别是张开型（模型 1）和滑开型（模型 2）的应力强度因子。
为了对问题简化，我们考虑简单的断裂：

$$\Omega = \{(x,y):\ -1 < x < 1, -1 < y < 1\}, \qquad \Sigma = \{(x,y):\ -1 \le x \le 0, y = 0\}$$

只有一个裂缝尖端 $\gamma = (0,0)$。 不幸地是， FreeFem++ 不能处理断裂，所以我们改进了一下U型隧道区域（见，如， 5.30），使得 $d = 0.0001$。 断裂 Σ 由一下区域近似：

$$\begin{aligned} \Sigma_d &= \{(x,y):\ -1 \le x \le -10*d, -d \le y \le d\} \\ &\quad \cup\{(x,y):\ -10*d \le x \le 0, -d + 0.1*x \le y \le d - 0.1*x\} \end{aligned}$$

$\Gamma_D =$ R 显示在 Fig. 5.30。 在这个例子中，我们需要使用到3个技术：

- 使用快速有限元内插器，将网格 Th插入网格 Zoom，并在γ附近等比例放大。
- 在解得位移向量$\boldsymbol{u}=(u,v)$后，利用以下代码，可以观察到裂缝的变形情况。

```
mesh Plate = movemesh(Zoom,[x+u,y+v]);
plot(Plate);
```

- 正如(9.23)所示，在γ点出现了严重的奇异性，因此自适应性技术在这个例子中十分重要。

第一个例子演示了形变模式I。它是由作用在边界 B和 T上，沿Σ的法向方向相对的表面力所引起的形变，且形变位移固定在边界R上。
在实验室中，研究断裂的工程师们用光弹性法使得应力场可视化[1]，计算主应力的差值为：

$$\sigma_1-\sigma_2=\sqrt{(\sigma_{11}-\sigma_{22})^2+4\sigma_{12}^2} \tag{9.25}$$

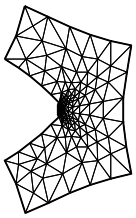
其中，σ_1和σ_2是主应力。在张开型裂纹中，光弹性法假设裂纹尖端张开位移关于γ对称。

Example 9.15 (裂缝开裂, $K_2(\gamma)=0$) {CrackOpen.edp}

```
real d = 0.0001;
int n = 5;
real cb=1, ca=1, tip=0.0;
border L1(t=0,ca-d) { x=-cb; y=-d-t; }
border L2(t=0,ca-d) { x=-cb; y=ca-t; }
border B(t=0,2) { x=cb*(t-1); y=-ca; }
border C1(t=0,1) { x=-ca*(1-t)+(tip-10*d)*t; y=d; }
border C21(t=0,1) { x=(tip-10*d)*(1-t)+tip*t; y=d*(1-t); }
border C22(t=0,1) { x=(tip-10*d)*t+tip*(1-t); y=-d*t; }
border C3(t=0,1) { x=(tip-10*d)*(1-t)-ca*t; y=-d; }
border C4(t=0,2*d) { x=-ca; y=-d+t; }
border R(t=0,2) { x=cb; y=cb*(t-1); }
border T(t=0,2) { x=cb*(1-t); y=ca; }
mesh Th = buildmesh (L1(n/2)+L2(n/2)+B(n)
                     +C1(n)+C21(3)+C22(3)+C3(n)+R(n)+T(n));
cb=0.1; ca=0.1;
plot(Th,wait=1);
mesh Zoom = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)
                       +C21(3)+C22(3)+C3(n)+R(n)+T(n));
plot(Zoom,wait=1);
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
fespace Vh(Th,[P2,P2]);
fespace zVh(Zoom,P2);
Vh [u,v], [w,s];
solve  Problem([u,v],[w,s])  =
    int2d(Th)(
              2*mu*(dx(u)*dx(w)+ ((dx(v)+dy(u))*(dx(s)+dy(w)))/4 )
              + lambda*(dx(u)+dy(v))*(dx(w)+dy(s))/2
              )
    -int1d(Th,T)(0.1*(4-x)*s)+int1d(Th,B)(0.1*(4-x)*s)
    +on(R,u=0)+on(R,v=0);                                    //     固定
;
```

[1]光弹性法是利用应力双折射效应，得到干涉光，然后通过测光程差来测得应力的，所以说是可视化的。

```
zVh Sx, Sy, Sxy, N;
for (int i=1; i<=5; i++)
{
  mesh Plate = movemesh(Zoom,[x+u,y+v]);                    //    γ~附近的形变
  Sx = lambda*(dx(u)+dy(v)) + 2*mu*dx(u);
  Sy = lambda*(dx(u)+dy(v)) + 2*mu*dy(v);
  Sxy = mu*(dy(u) + dx(v));
  N = 0.1*1*sqrt((Sx-Sy)^2+4*Sxy^2);                        //   主应力差
  if (i==1) {
     plot(Plate,ps="1stCOD.eps",bw=1);                      //    图 9.13
     plot(N,ps="1stPhoto.eps",bw=1);                        //    图 9.13
  } else if (i==5) {
     plot(Plate,ps="LastCOD.eps",bw=1);                     //    图 9.14
     plot(N,ps="LastPhoto.eps",bw=1);                       //    图 9.14
     break;
  }
  Th=adaptmesh(Th,[u,v]);
  Problem;
}
```



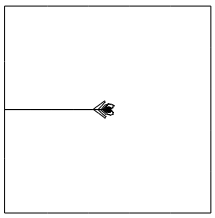
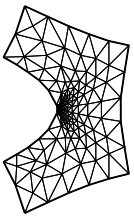


图 9.13: 第一种网格中，裂纹张开位移(COD)和主应力差值。



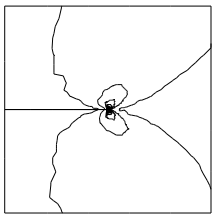


图 9.14: 后一种致信网格中，裂纹张开位移(COD)和主应力差值。

实验室中观测到的形变模式II是较难模拟构建的，是由作用在 B和 T上的相对的剪切力所引起的。所以我们使用沿着Σ的体剪切力，也即体力$\boldsymbol{f}$在x轴方向上的分力f_1。

$$f_1(x,y) = H(y-0.001) * H(0.1-y) - H(-y-0.001) * H(y+0.1)$$

其中，若$t>0$，$H(t)=1$；若$t<0$，$H(t)=0$。

Example 9.16 (Crack Sliding, $K_2(\gamma)=0$) [2]

```
                                                            //    (使用相同的网格)
real d = 0.0001;                                            //    *
int n = 5;                                                  //    *
real cb=1, ca=1, tip=0.0;                                   //    *
border L1(t=0,ca-d) { x=-cb; y=-d-t; }                      //    *
border L2(t=0,ca-d) { x=-cb; y=ca-t; }                      //    *
border B(t=0,2) { x=cb*(t-1); y=-ca; }                      //    *
```

[2]原文中的程序无法运行，故进行了修改，修改部分以“//*”标注。

```
border C1(t=0,1) { x=-ca*(1-t)+(tip-10*d)*t; y=d; }                    //   *
border C21(t=0,1) { x=(tip-10*d)*(1-t)+tip*t; y=d*(1-t); }             //   *
border C22(t=0,1) { x=(tip-10*d)*t+tip*(1-t); y=-d*t; }                //   *
border C3(t=0,1) { x=(tip-10*d)*(1-t)-ca*t; y=-d; }                    //   *
border C4(t=0,2*d) { x=-ca; y=-d+t; }                                  //   *
border R(t=0,2) { x=cb; y=cb*(t-1); }                                  //   *
border T(t=0,2) { x=cb*(1-t); y=ca; }                                  //   *
mesh Th = buildmesh (L1(n/2)+L2(n/2)+B(n)
                     +C1(n)+C21(3)+C22(3)+C3(n)+R(n)+T(n));            //   *
plot(Th,wait=1);                                                       //   *

cb=0.01; ca=0.01;
mesh Zoom = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)
                       +C21(3)+C22(3)+C3(n)+R(n)+T(n));
                                          //   (使用相同的有限元空间 Vh 和弹性模)
fespace Vh1(Th,P1);
real E = 21.5;                                                         //   *
real sigma = 0.29;                                                     //   *
real mu = E/(2*(1+sigma));                                             //   *
real lambda = E*sigma/((1+sigma)*(1-2*sigma));                         //   *
Vh1 fx = ((y>0.001)*(y<0.1))-((y<-0.001)*(y>-0.1)) ;
fespace Vh(Th,[P2,P2]);                                                //   *
fespace zVh(Zoom,P2);                                                  //   *
Vh [u,v], [w,s];                                                       //   *
solve  Problem([u,v],[w,s])  =
    int2d(Th)(
              2*mu*(dx(u)*dx(w)+ ((dx(v)+dy(u))*(dx(s)+dy(w)))/4 )
              + lambda*(dx(u)+dy(v))*(dx(w)+dy(s))/2
              )
    -int2d(Th)(fx*w)
    +on(R,u=0)+on(R,v=0);                                  //     固定
;
zVh Sx, Sy, Sxy, N;                                                    //   *
for (int i=1; i<=3; i++)
{
  mesh Plate = movemesh(Zoom,[x+u,y+v]);                   //     γ~附近的形变
  Sx = lambda*(dx(u)+dy(v)) + 2*mu*dx(u);
  Sy = lambda*(dx(u)+dy(v)) + 2*mu*dy(v);
  Sxy = mu*(dy(u) + dx(v));
  N = 0.1*1*sqrt((Sx-Sy)^2+4*Sxy^2);                       //   主应力差
  if (i==1) {
     plot(Plate,ps="1stCOD2.eps",bw=1);                    //     图 9.16
     plot(N,ps="1stPhoto2.eps",bw=1);                      //     图 9.15
  } else if (i==3) {
     plot(Plate,ps="LastCOD2.eps",bw=1);                   //     图 9.16
     plot(N,ps="LastPhoto2.eps",bw=1);                     //     图 9.16
     break;
  }
  Th=adaptmesh(Th,[u,v]);
  Problem;
}
```

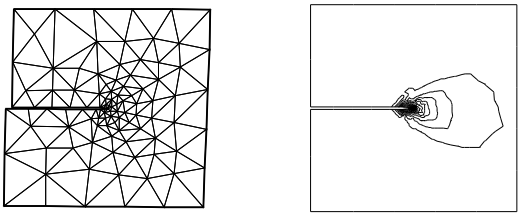


图 9.15: 第一种网格中，裂纹张开位移(COD)和主应力差值。

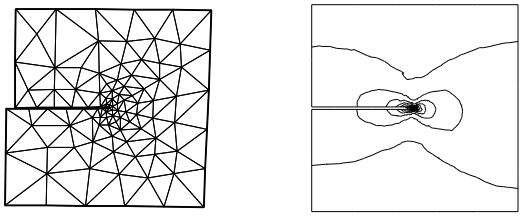


图 9.16: 后一种致信网格中，裂纹张开位移(COD)和主应力差值。

9.3 非线性静态问题

现在让我们来解决一个非线性问题：最小化下述泛函：

$$J(u) = \int_\Omega \frac{1}{2} f(|\nabla u|^2) - u * b$$

其中，u属于$H_0^1(\Omega)$，函数f定义如下：

$$f(x) = a * x + x - ln(1+x), \quad f'(x) = a + \frac{x}{1+x}, \quad f''(x) = \frac{1}{(1+x)^2}$$

9.3.1 牛顿-拉夫逊算法

现在，用牛顿-拉夫逊算法解决欧拉问题$\nabla J(u) = 0$，也即

$$u^{n+1} = u^n - (\nabla^2 J(u^n))^{-1} * \nabla J(u^n)$$

首先，介绍两种分别用于计算∇J和$\nabla^2 J$的变形形式vdJ和vhJ。

```
//     求解 dJ(u)=0 的~Newton-Raphson 方法;
                                                                    //
```

$$u^{n+1} = u^n - (\frac{\partial dJ}{\partial u_i})^{-1} * dJ(u^n)$$

```
//     ----------------------------------------------
  Ph dalpha ;                                          //    保存 2f''(|∇u|^2) 最优

  //     求值 dJ = ∇J 的变分形式
  //     ------------------------------------
  //     dJ = f'()*( dx(u)*dx(vh) + dy(u)*dy(vh)
  varf vdJ(uh,vh) =  int2d(Th)( alpha*( dx(u)*dx(vh) + dy(u)*dy(vh) ) - b*vh)
  + on(1,2,3,4, uh=0);
```

```
  //     求值 ddJ = ∇²J 的变分形式
  //     hJ(uh,vh) = f'()*( dx(uh)*dx(vh) + dy(uh)*dy(vh)
              //     + 2*f''()( dx(u)*dx(uh) + dy(u)*dy(uh) ) * (dx(u)*dx(vh) +
dy(u)*dy(vh))
  varf vhJ(uh,vh) = int2d(Th)( alpha*( dx(uh)*dx(vh) + dy(uh)*dy(vh) )
   +  dalpha*( dx(u)*dx(vh) + dy(u)*dy(vh)  )*( dx(u)*dx(uh) + dy(u)*dy(uh) ) )
   + on(1,2,3,4, uh=0);

 //     Newton 算法
  Vh v,w;
  u=0;
  for (int i=0;i<100;i++)
   {
    alpha =     df( dx(u)*dx(u) + dy(u)*dy(u) ) ;                    //     最优化
    dalpha = 2*ddf( dx(u)*dx(u) + dy(u)*dy(u) ) ;                    //     最优化
    v[]= vdJ(0,Vh);                                                 //     v = ∇J(u)
    real res= v[]'*v[];                                              //     点积
    cout << i <<  " residu^2 = " <<  res  << endl;
    if( res< 1e-12) break;
    matrix H= vhJ(Vh,Vh,factorize=1,solver=LU);                      //
    w[]=H^-1*v[];
    u[] -= w[];
   }
   plot (u,wait=1,cmm="solution with Newton-Raphson");
```

注释：这个例子在examples++-tutorial目录下的Newton.edp文件中。

9.4 特征值问题

学习这一节前，你需要先完成ARPACK的编译（见于README_arpack）。如果单词“eigenvalue”像下面一样出现在“Load:”行中，那么这一工具可以在FreeFem++中找到。

```
-- FreeFem++ v1.28 (date Thu Dec 26 10:56:34 CET 2002)
 file : LapEigenValue.edp
 Load: lg_fem lg_mesh eigenvalue
```

这一工具是基于ARPACK特征值软件包的面向对象版本arpack++ [3]。[1]
函数EigenValue能计算$Au = \lambda Bu$的广义特征值。其中，sigma $=\sigma$是位移值[4]。定义矩阵OP 为$A - \sigma B$。函数返回值是收敛特征值的个数(可能远大于特征值数nev=)。

```
int k=EigenValue(OP,B,nev= , sigma= );
```

其中，矩阵$OP = A - \sigma B$自带对应的特征值求解器和相应的偏微分问题边界条件，B是矩阵。

Note 9.1 边界条件和特征值问题
锁定边界条件（*Dirichlet*条件）通过确定的罚数构造[5]，故将*1e30=tgv* 代入锁定自由度的对角线项*(*见方程*(6.31))*。 鉴于我们求解的是$w = OP^{-1} * B * v$，所以只需要对A，而不需要对B，考虑*Dirichlet* 条件。
如果对矩阵B考虑*Dirichlet*边界条件（即程序中有关键字 **on**），那么根据边界条件，将得到微弱振动模式 (10^{-30})；但是如果要对矩阵B忽略*Dirichlet* 边界条件（无关键字 **on**），那么根据这些边界条件，将得到强振动模式 (10^{30})。本节只计算了微弱振动模式，得到了较好的结果。

sym= 问题有对称性（所有特征值均是实数）

nev= 希望得到的、（最）接近位移值的特征值数目(nev)

value= 储存特征值实部的数组

ivalue= 储存特征值虚部的数组

vector= 储存特征值的FE函数数组

rawvector= `real[int,int]` 类型的数组，按列储存特征值(根据version 2-17).

对于实数非对称性问题，复特征向量是两个连续向量，所以如果第k和$k+1$个特征值是复共轭特征值，则第k个向量包含对应复共轭特征向量的实部，而第$k+1$ 个向量包含对应复共轭特征向量的虚部。

tol= 确定特征值时所用的相对精度

sigma= 位移值

maxit= 迭代次数上限

ncv= ARPACK的每次迭代生成的Arnoldi 向量个数

[3] http://www.caam.rice.edu/software/ARPACK/
[4] 简单地，可以理解为只求大于σ的特征值。
[5] 本质上即指以极小数近似代替零，例如用$10^{30}u|_\Gamma = 1$近似表示$u|_\Gamma = 0$。

Example 9.17 (lapEignenValue.edp) [6]
第一个例子计算了方形区域$\Omega = [0,\pi]^2$上的*Dirichlet*问题的特征值和特征向量。
而本例将求解： λ，和 $\nabla u_\lambda \in \mathbb{R}\times H_0^1(\Omega)$

$$\int_\Omega \nabla u_\lambda \nabla v = \lambda \int_\Omega uv \quad \forall v \in H_0^1(\Omega)$$

精确的特征值是$\lambda_{n,m} = (n^2+m^2),(n,m) \in {\mathbb{N}_*}^2$，对应的特征向量是$u_{m,n} = sin(nx) * sin(my)$。
我们使用*arpack++*库中的广义带位移的反幂模式[7]，寻找接近位移值$\sigma = 20$的*20* 个特征值和特征向量。

```
//     在方形区域 ~]0,π[^2 ~求解 ~Dirichlet ~问题的
//     特征值和特征向量
//     ---------------------------------------
//     我们使用带位移的反幂模式
//     位移量为一个实的 ~sigma
//     ---------------------------------------
//     找 λ 和 u_λ ∈ H_0^1(Ω) 使得:
//                                  ∫_Ω ∇u_λ∇v = λ ∫_Ω u_λ v, ∀v ∈ H_0^1(Ω)
verbosity=10;
mesh Th=square(20,20,[pi*x,pi*y]);
fespace Vh(Th,P2);
Vh u1,u2;

real sigma = 20;                                                    //     位移值

                                                  //     OP = A - sigma B ; // 位移后的矩阵
varf  op(u1,u2)= int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) - sigma* u1*u2 )
                     +  on(1,2,3,4,u1=0) ;                          //     边界条件

varf b([u1],[u2]) = int2d(Th)(  u1*u2 );              //     没有边界条件, 见注释 9.1
matrix OP= op(Vh,Vh,solver=Crout,factorize=1);    //     用crout 求解器, 因为矩阵不
是正的
matrix B= b(Vh,Vh,solver=CG,eps=1e-20);

                                                                    //     重要的注:
                                                          //     边界条件产生精确的惩罚项:
                                                  //     我们将 1e30=tgv 加在自由度的对角项上.
                                          //     所以只需在 ~a 的变分形式后加Dirichlet边界条件
                                                          //     而不是 b 的变分形式.
                                                  //     因为我们求解的是 w = OP^-1 * B * v

int nev=20;                                                 //     离sigma近的待求解的特征值个数

real[int] ev(nev);                                                  //     储存nev个特征值
Vh[int] eV(nev);                                                //     储存nev个特征向量
```

[6]这个程序包含在example++-eigen目录下的lapEignenValue.edp程序中，除了在用“a“代替“OP”，其他与下文一致。

[7]若A为非奇异矩阵，则线性方程组$Ax = b$的解为$x = A^{-1}b$，其中A的逆矩阵A^{-1} 满足$A^{-1}A = AA^{-1} = I$(I为单位矩阵)。若A是奇异阵或长方阵，$Ax = b$可能无解或有很多解。若有解，则解为$x = Xb + (I - XA)$，其中是维数与A 的列数相同的任意向量，X是满足$AXA = A$的任何一个矩阵，通常称X为A的广义逆矩阵，有时简称广义逆。当A非奇异时，A^{-1}也满足$AA^{-1}A = A$，且$x = A^{-1}b + (I - A^{-1}A) = A^{-1}b$。故非异阵的广义逆矩阵就是它的逆矩阵，说明广义逆矩阵确是通常逆矩阵概念的推广。

```
int k=EigenValue(OP,B,sym=true,sigma=sigma,value=ev,vector=eV,
                   tol=1e-10,maxit=0,ncv=0);

//     tol= 限度
//     maxit= 最大迭代步数 见 arpack 文件.
//     ncv 见 arpack 文件. http://www.caam.rice.edu/software/ARPACK/
//     返回值是一个收敛的特征值个数.

for (int i=0;i<k;i++)
{
  u1=eV[i];
  real gg = int2d(Th)(dx(u1)*dx(u1) + dy(u1)*dy(u1));
  real mm= int2d(Th)(u1*u1) ;
  cout << " ---- " <<  i<< " " << ev[i]<< " err= "
       <<int2d(Th)(dx(u1)*dx(u1) + dy(u1)*dy(u1) - (ev[i])*u1*u1) << " --- "<<endl;
  plot(eV[i],cmm="Eigen  Vector "+i+" valeur =" + ev[i]  ,wait=1,value=1);
}
```

输出结果如下：

```
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 80, neb = 80
 Nb Of Nodes = 1681
 Nb of DF = 1681
Real symmetric eigenvalue problem: A*x - B*x*lambda


Thanks to ARPACK++ class ARrcSymGenEig
Real symmetric eigenvalue problem: A*x - B*x*lambda
Shift and invert mode  sigma=20

Dimension of the system            : 1681
Number of 'requested' eigenvalues  : 20
Number of 'converged' eigenvalues  : 20
Number of Arnoldi vectors generated: 41
Number of iterations taken         : 2

Eigenvalues:
  lambda[1]: 5.0002
  lambda[2]: 8.00074
  lambda[3]: 10.0011
  lambda[4]: 10.0011
  lambda[5]: 13.002
  lambda[6]: 13.0039
  lambda[7]: 17.0046
  lambda[8]: 17.0048
  lambda[9]: 18.0083
  lambda[10]: 20.0096
  lambda[11]: 20.0096
  lambda[12]: 25.014
  lambda[13]: 25.0283
  lambda[14]: 26.0159
  lambda[15]: 26.0159
  lambda[16]: 29.0258
```

```
 lambda[17]: 29.0273
 lambda[18]: 32.0449
 lambda[19]: 34.049
 lambda[20]: 34.0492

---- 0 5.0002 err= -0.000225891 ---
---- 1 8.00074 err= -0.000787446 ---
---- 2 10.0011 err= -0.00134596 ---
---- 3 10.0011 err= -0.00134619 ---
---- 4 13.002 err= -0.00227747 ---
---- 5 13.0039 err= -0.004179 ---
---- 6 17.0046 err= -0.00623649 ---
---- 7 17.0048 err= -0.00639952 ---
---- 8 18.0083 err= -0.00862954 ---
---- 9 20.0096 err= -0.0110483 ---
---- 10 20.0096 err= -0.0110696 ---
---- 11 25.014 err= -0.0154412 ---
---- 12 25.0283 err= -0.0291014 ---
---- 13 26.0159 err= -0.0218532 ---
---- 14 26.0159 err= -0.0218544 ---
---- 15 29.0258 err= -0.0311961 ---
---- 16 29.0273 err= -0.0326472 ---
---- 17 32.0449 err= -0.0457328 ---
---- 18 34.049 err= -0.0530978 ---
---- 19 34.0492 err= -0.0536275 ---
```

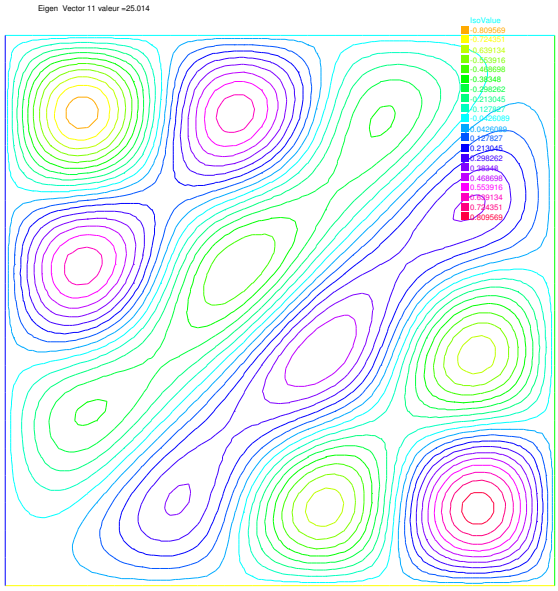


图 9.17: 第11个特征向量$u_{4,3}-u_{3,4}$的等值线

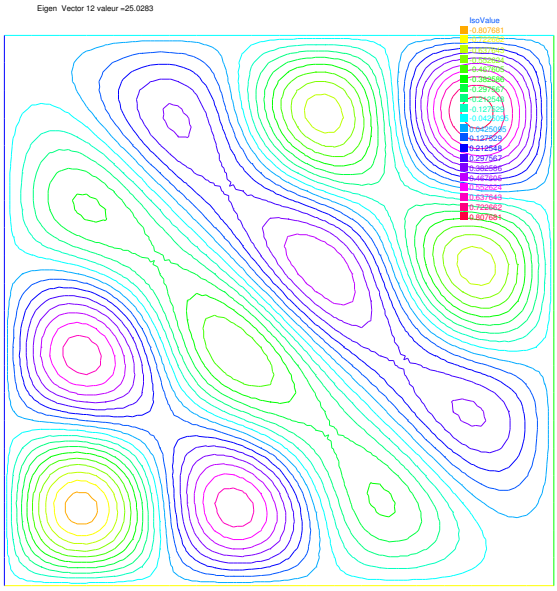


图 9.18: 第12个特征向量$u_{4,3}+u_{3,4}$的等值线

9.5 演化问题

FreeFem++ 也可以解决诸如热传导方程之类的演化问题：

$$\begin{aligned} &\frac{\partial u}{\partial t}-\mu\Delta u=f \quad \text{in } \Omega\times[0,T], \\ &u(\boldsymbol{x},0)=u_0(\boldsymbol{x}) \quad \text{in } \Omega; \qquad (\partial u/\partial n)(\boldsymbol{x},t)=0 \quad \text{on } \partial\Omega\times[0,T]. \end{aligned} \tag{9.26}$$

这一方程有正粘度系数μ，并满足齐次Neumann边界条件。处理空间时，我们使用FEM方法；处理时间则使用有限差分原理，以求解方程(9.26)。 按偏导数的定义，计算对时间求导的偏导数，

$$\frac{\partial u}{\partial t}(x,y,t)=\lim_{\tau\to 0}\frac{u(x,y,t)-u(x,y,t-\tau)}{\tau}$$

则$u^m(x,y)=u(x,y,m\tau)$将近似满足

$$\frac{\partial u}{\partial t}(x,y,m\tau)\simeq\frac{u^m(x,y)-u^{m-1}(x,y)}{\tau}$$

对热传导方程(9.27)进行时间离散化如下:

$$\begin{aligned} &\frac{u^{m+1}-u^m}{\tau}-\mu\Delta u^{m+1}=f^{m+1} \quad \text{in } \Omega \\ &u^0(\boldsymbol{x})=u_0(\boldsymbol{x}) \quad \text{in } \Omega; \qquad \partial u^{m+1}/\partial n(\boldsymbol{x})=0 \quad \text{on } \partial\Omega, \quad \text{for all } m=0,\cdots,[T/\tau], \end{aligned} \tag{9.27}$$

(9.27)就是所谓的后向Euler方法。为了得到变分公式，在等式两边同时乘上测试函数v：

$$\int_\Omega\{u^{m+1}v-\tau\Delta u^{m+1}v\}=\int_\Omega\{u^m+\tau f^{m+1}\}v\,.$$

根据散度定理，则

$$\int_\Omega\{u^{m+1}v+\tau\nabla u^{m+1}\cdot\nabla v\}-\int_{\partial\Omega}\tau\left(\partial u^{m+1}/\partial n\right)v=\int_\Omega\{u^mv+\tau f^{m+1}v\}.$$

根据边界条件$\partial u^{m+1}/\partial n=0$，则

$$\int_\Omega\{u^{m+1}v+\tau\nabla u^{m+1}\cdot\nabla v\}-\int_\Omega\{u^mv+\tau f^{m+1}v\}=0. \tag{9.28}$$

使用上述恒等式，就可以根据t来一步步地计算u^m的有限元近似u_h^m。

Example 9.18
现在求解下述问题，其精确解是$u(x,y,t)=tx^4$。

$$\begin{aligned} &\frac{\partial u}{\partial t}-\mu\Delta u=x^4-\mu 12tx^2 \ \ in\ \Omega\times]0,3[,\ \Omega=[0,1]^2 \\ &u(x,y,0)=0 \quad on\ \Omega, \qquad u|_{\partial\Omega}=t*x^4 \end{aligned}$$

```
//      热传导方程 ∂_t u = −μΔu = x^4 − μ12tx^2
mesh Th=square(16,16);
fespace Vh(Th,P1);
```

```
Vh u,v,uu,f,g;
real dt = 0.1, mu = 0.01;
problem dHeat(u,v) =
    int2d(Th)( u*v + dt*mu*(dx(u)*dx(v) + dy(u)*dy(v)))
    + int2d(Th) (- uu*v - dt*f*v )
    + on(1,2,3,4,u=g);

real t = 0;                                          //    从 t=0 开始
uu = 0;                                              //    u(x,y,0)=0
for (int m=0;m<=3/dt;m++)
{
   t=t+dt;
   f = x^4-mu*t*12*x^2;
   g = t*x^4;
   dHeat;
   plot(u,wait=true);
   uu = u;
   cout <<"t="<<t<<"L^2-Error="<<sqrt( int2d(Th)((u-t*x^4)^2) ) << endl;
}
```

在最后一行，计算了$t = m\tau, \tau = 0.1$时的L^2-误差估计$\left(\int_\Omega |u - tx^4|^2\right)^{1/2}$。当$t = 0.1$时，误差是*0.000213269*。误差随$m$的增大而增大，当$t = 3$时，误差是*0.00628589*。
后向*Euler*迭代*(9.28)*由**for loop**实现（见于 **??**）。

Note 9.2 循环中的刚度矩阵再三被使用。*FreeFem++* 也支持刚度矩阵的反复使用。

9.5.1 时间差分逼近中的数学原理

在这一节中，我们将看到隐式差分格式的优点。令V和H是可分Hilber空间，V在H中稠密。令a作用在$V \times V$上，是对称正定的连续双线性形式算子[8]。那么$\sqrt{a(v,v)}$是V上范数$\|v\|$的等价形式。
Problem Ev(f, Ω): 对于给定的$f \in L^2(0,T;V'), u^0 \in H$

$$\begin{aligned} \frac{d}{dt}(u(t),v) + a(u(t),v) &= (f(t),v) \qquad \forall v \in V, , \quad a.e.\, t \in [0,T] \\ u(0) &= u^0 \end{aligned} \tag{9.29}$$

其中，V'是V的对偶空间。那么，存在唯一解$u \in L^\infty(0,T;H) \cap L^2(0,T;V)$。记时间步长为：$\tau > 0$，$N_T = [T/\tau]$。为了离散化，令$u^n = u(n\tau)$，并对每个$\theta \in [0,1]$，考虑时间差分：

$$\begin{aligned} \frac{1}{\tau}\left(u_h^{n+1} - u_h^n, \phi_i\right) + a\left(u_h^{n+\theta}, \phi_i\right) &= \langle f^{n+\theta}, \phi_i \rangle \\ i = 1, \cdots, m, \quad n = 0, \cdots, N_T & \\ u_h^{n+\theta} = \theta u_h^{n+1} + (1-\theta)u_h^n, \quad f^{n+\theta} &= \theta f^{n+1} + (1-\theta) f^n \end{aligned} \tag{9.30}$$

如果$\theta = 0$，那么公式(9.30)就是前向*Euler*格式；如果$\theta = 1/2$，就是*Crank-Nicolson*格式；如果$\theta = 1$，就是后向*Euler*格式。
未知向量$u^n = (u_h^1, \cdots, u_h^M)^T$。其中，

$$u_h^n(x) = u_1^n \phi_1(x) + \cdots + u_m^n \phi_m(x), \quad u_1^n, \cdots, u_m^n \in \mathbb{R}$$

[8] 正定是指$a(u,u) \geq \alpha \parallel u \parallel^2$，其中$\alpha > 0$。

是以下矩阵等式的解：

$$(M+\theta\tau A)u^{n+1}=\{M-(1-\theta)\tau A\}u^n+\tau\left\{\theta f^{n+1}+(1-\theta)f^n\right\} \tag{9.31}$$
$$M=(m_{ij}),\quad m_{ij}=(\phi_j,\phi_i),\qquad A=(a_{ij}),\quad a_{ij}=a(\phi_j,\phi_i)$$

(9.31)的可解性参见[22, pp.70–75]，(9.31) 稳定性参见[22, Theorem 2.13]：

令$\{\mathcal{T}_h\}_{h\downarrow 0}$是常规的三角剖分(见 **??**)。则如果下述满足：

1. 当$\theta\in[0,1/2)$时，对于任意$\delta\in(0,1)$，可选取时间步长τ满足：

$$\tau<\frac{2(1-\delta)}{(1-2\theta)c_0^2}h^2 \tag{9.32}$$

2. 当$1/2\le\theta\le 1$时,τ可以任意选取。

则存在与h无关的常数$c_0>0$，使得：

$$|u_h^n|^2\le\begin{cases}\frac{1}{\delta}\left\{|u_h^0|^2+\tau\sum_{k=0}^{n-1}\|f^{k+\theta}\|^2_{V_h'}\right\} & \theta\in[0,1/2)\\ |u_h^0|^2+\tau\sum_{k=0}^{n-1}\|f^{k+\theta}\|^2_{V_h'} & \theta\in[1/2,1]\end{cases} \tag{9.33}$$

Example 9.19

```
mesh Th=square(12,12);
fespace Vh(Th,P1);
fespace Ph(Th,P0);

Ph h = hTriangle;                                          //    每个三角形的网格尺寸
real tau = 0.1, theta=0.;
func real f(real t) {
   return x^2*(x-1)^2 + t*(-2 + 12*x - 11*x^2 - 2*x^3 + x^4);
}
ofstream out("err02.csv");                                 //    储存计算的文件
out << "mesh size = "<<h[].max<<", time step = "<<tau<<endl;
for (int n=0;n<5/tau;n++) \\
   out<<n*tau<<",";
out << endl;
Vh u,v,oldU;
Vh f1, f0;
problem aTau(u,v) =
  int2d(Th)( u*v + theta*tau*(dx(u)*dx(v) + dy(u)*dy(v) + u*v))
  - int2d(Th)(oldU*v - (1-theta)*tau*(dx(oldU)*dx(v)+dy(oldU)*dy(v)+oldU*v))
  - int2d(Th)(tau*( theta*f1+(1-theta)*f0 )*v );

while (theta <= 1.0) {
  real t = 0, T=3;                                         //    从 t=0 到 T
  oldU = 0;                                                //    u(x,y,0)=0
  out <<theta<<",";
  for (int n=0;n<T/tau;n++) {
      t = t+tau;
      f0 = f(n*tau); f1 = f((n+1)*tau);
      aTau;
      oldU = u;
      plot(u);
```

```
        Vh uex = t*x^2*(1-x)^2;                          //    精确解 ~= tx^2(1-x)^2
        Vh err = u - uex;                                //    err =有限元解-精确解
        out<< abs(err[].max)/abs(uex[].max) <<",";       //    ||err||_{L∞(Ω)}/||u_ex||_{L∞(Ω)}
    }
    out << endl;
    theta = theta + 0.1;
}
```

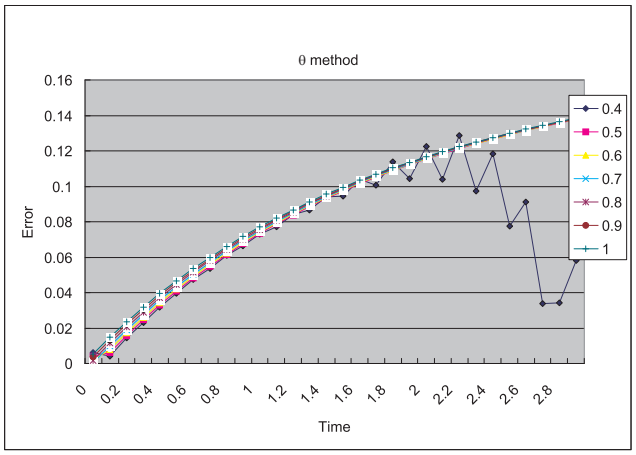


图 9.19: $\max_{x\in\Omega}|u_h^n(\theta)-u_{ex}(n\tau)|/\max_{x\in\Omega}|u_{ex}(n\tau)|$ at $n=0,1,\cdots,29$

我们可以看到图 *9.19*中，当$\theta=0.4$ 时，$u_h^n(\theta)$变得不稳定, 并且当$\theta<0.4$时，图像发生缺失。

9.5.2 对流

双曲型方程如下

$$\partial_t u+\boldsymbol{\alpha}\cdot\nabla u=f;\quad \text{对于向量值函数}\boldsymbol{\alpha}, \tag{9.34}$$

在科技问题中经常会出现，例如Navier-Stokes方程组，又如对流扩散方程等等。
在一维空间的情形下，当α是常数时，我们能够很容易地找到$(x,t)\mapsto u(x,t)=u^0(x-\alpha t)$ 作为下述方程的一般解，

$$\partial_t u+\alpha\partial_x u=0,\qquad u(x,0)=u^0(x), \tag{9.35}$$

由于$\partial_t u+\alpha\partial_x u=-\alpha\dot{u}^0+a\dot{u}^0=0$, 其中$\dot{u}^0=du^0(x)/dx$。 即使$\alpha$不是常数，构造工作仍依照相似的原则。以常微分方程开始(按照惯例，α 是由0开始延长，但不包括在$(0,L)\times(0,T)$ 内):

$$\dot{X}(\tau)=+\alpha(X(\tau),\tau),\quad \tau\in(0,t)\quad X(t)=x$$

在这一方程中τ是变量，而x,t则是参数，并且我们将解记为$X_{x,t}(\tau)$。 我们发现，当$\tau=t$时，$(x,t)\to v(X(\tau),\tau)$ 满足下面的方程

$$\partial_t v+\alpha\partial_x v=\partial_t X\dot{v}+a\partial_x X\dot{v}=0$$

且由定义$\partial_t X=\dot{X}=+\alpha$以及当$\tau=t$时，$\partial_x X=\partial_x x$，这是因为如果$\tau=t$我们有$X(\tau)=x$。则(9.35)的一般解就是$X_{x,t}(0)$在边界条件上的值，也即是说$u(x,t)=u^0(X_{x,t}(0))$，其中$X_{x,t}(0)$是在$x$轴，如果$X_{x,t}(0)$是在$t$轴，则有$u(x,t)=u^0(X_{x,t}(0))$。

在高维情形下，$\Omega \subset R^d,\ d=2,3$, 对流方程写作

$$\partial_t u + \boldsymbol{\alpha}\cdot\nabla u = 0 \text{ in } \Omega\times(0,T)$$

其中$\boldsymbol{a}(x,t)\in R^d$。 FreeFem++ 对于对流算子采用的是加勒金法。回忆方程(9.34)可以被离散化为

$$\frac{Du}{Dt} = f \text{ i.e. } \frac{du}{dt}(X(t),t) = f(X(t),t) \text{ where } \frac{dX}{dt}(t) = \boldsymbol{\alpha}(X(t),t)$$

其中D是全导数算子。因此一种较好的方法是一阶向后的加勒金法来解决对流问题。

$$\frac{1}{\tau}\left(u^{m+1}(x) - u^m(X^m(x))\right) = f^m(x) \tag{9.36}$$

其中$X^m(x)$是如下常微分方程的解在$t=m\tau$ 点处的逼近

$$\frac{d\boldsymbol{X}}{dt}(t) = \boldsymbol{\alpha}^m(\boldsymbol{X}(t)),\ \boldsymbol{X}((m+1)\tau) = x.$$

其中$\boldsymbol{\alpha}^m(x) = (\alpha_1(x,m\tau),\alpha_2(x,m\tau))$。由泰勒展开，我们有

$$\begin{aligned} u^m(\boldsymbol{X}(m\tau)) &= u^m(\boldsymbol{X}((m+1)\tau)) - \tau\sum_{i=1}^{d}\frac{\partial u^m}{\partial x_i}(\boldsymbol{X}((m+1)\tau))\frac{\partial X_i}{\partial t}((m+1)\tau) + o(\tau) \\ &= u^m(x) - \tau\boldsymbol{\alpha}^m(x)\cdot\nabla u^m(x) + o(\tau) \end{aligned} \tag{9.37}$$

其中$X_i(t)$ 是向量$\boldsymbol{X}(t)$的第i个元素，$u^m(x) = u(x,m\tau)$我们运用链式法则以及$x=\boldsymbol{X}((m+1)\tau)$。由(9.37) 式得到

$$u^m(X^m(x)) = u^m(x) - \tau\boldsymbol{\alpha}^m(x)\cdot\nabla u^m(x) + o(\tau). \tag{9.38}$$

类似地，我们运用泰勒展开，$t\mapsto u^m(x-\boldsymbol{\alpha}^m(x)t),\ 0\le t\le\tau$，则有

$$u^m(x-\boldsymbol{\alpha}\tau) = u^m(x) - \tau\boldsymbol{\alpha}^m(x)\cdot\nabla u^m(x) + o(\tau).$$

令

$$\textbf{convect}\left(\boldsymbol{\alpha},-\tau,u^m\right) \approx u^m\left(x-\boldsymbol{\alpha}^m\tau\right),$$

我们就能得到如下近似

$$u^m\left(X^m(x)\right) \approx \textbf{convect}\left([a_1^m,a_2^m],-\tau,u^m\right) \text{ by } X^m \approx x\mapsto x-\tau[a_1^m(x),a_2^m(x)]).$$

经典对流问题是具有“旋转钟”的 (引用自 [14][p.16])。令Ω是中心位于0的单位圆盘， 它的圆心以$\alpha_1 = y,\ \alpha_2 = -x$的速度旋转。我们考虑问题(9.34)当$f=0$ 以及初始条件为$u(x,0)=u^0(x)$时，那么由(9.36) 可得

$$u^{m+1}(x) = u^m(X^m(x)) \approx \texttt{convect}(\boldsymbol{\alpha},-\tau,u^m).$$

精确解是$u(x,t) = u(\boldsymbol{X}(t))$ 其中$\boldsymbol{X}$是x 沿着起点以角速度$\theta=-t$ (按顺时针方向旋转)。 所以，如果u^0在三维透视角度看起来像一个钟，那么u也会有相同的形状，但是旋转的速度不同。这一问题包括解直到$T=2\pi$的对应方程，这是一个完整的循环，并将最终解与初始解进行比较；它们应该是相同的。

Example 9.20 (convect.edp)

```
border C(t=0, 2*pi) { x=cos(t); y=sin(t); };   //   单位圆
mesh Th = buildmesh(C(70));                    //   对圆盘进行三角剖分
```

```
fespace Vh(Th,P1);
Vh u0 = exp(-10*((x-0.3)^2 +(y-0.3)^2));                    //    给出 u^0

real dt = 0.17,t=0;                                          //    时间步长
Vh a1 = -y, a2 = x;                                          //    旋转速度
Vh u;                                                        //    u^{m+1}
for (int m=0; m<2*pi/dt ; m++) {
    t += dt;
    u=convect([a1,a2],-dt,u0);                               //    u^{m+1} = u^m(X^m(x))
    u0=u;                                                    //    m++
    plot(u,cmm=" t="+t + ", min=" + u[].min + ", max=" +  u[].max,wait=0);
};
```

Note 9.3 这一方法*convect*是无条件稳定的，钟会变得越来越慢*(*如图 *9.21*所示，u^{37}的最大值是0.406*)*。

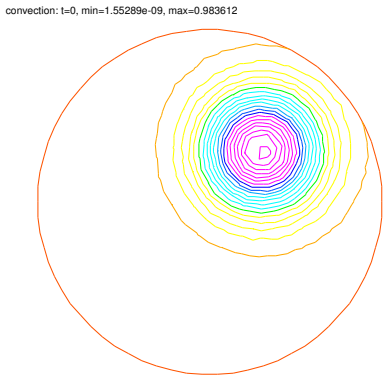


图 9.20: $u^0 = e^{-10((x-0.3)^2+(y-0.3)^2)}$

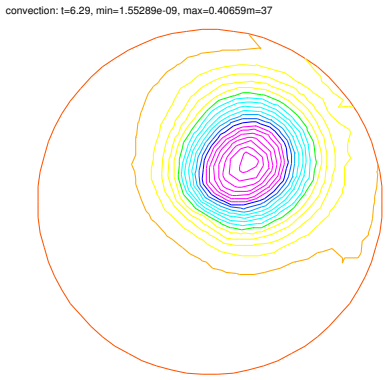


图 9.21: 在 $t = 6.29$ 时刻的钟

9.5.3 欧式看跌期权的二维Black-Scholes 方程

在金融数学中，两个标的资产的期权在两个空间中，以布莱克-休尔斯方程进行建模，(具体见威尔莫特的例子 [39] 或者阿柯尔的案例[3]).

$$\begin{aligned} &\partial_t u + \frac{(\sigma_1 x)^2}{2}\frac{\partial^2 u}{\partial x^2} + \frac{(\sigma_2 y)^2}{2}\frac{\partial^2 u}{\partial y^2} \\ &+ \rho xy\frac{\partial^2 u}{\partial x \partial y} + rS_1\frac{\partial u}{\partial x} + rS_2\frac{\partial u}{\partial y} - rP = 0 \end{aligned} \tag{9.39}$$

在$(0,T)\times\mathbb{R}^+\times\mathbb{R}^+$上进行积分。而在看跌期权情形下，

$$u(x,y,T) = (K - \max(x,y))^+ . \tag{9.40}$$

该问题的边界条件不容易给出。由于一维情形下的偏微分方程包含在轴$x_1 = 0$以及在轴$x_2 = 0$的边界条件，也就是两个一维的布莱克-休尔斯方程由数据 $u(0,+\infty,T)$ 与 $u(+\infty,0,T)$ 分别驱动。由于它们是嵌入到偏微分方程中的，因此自动会进行解释说明。 所以如果我们不对变分形式进行改动，(也就是如果我们采用诺依曼边界条件，那么在强形式中会有两个坐标轴) 那么就不会

有扰动产生。 在其中一个变量达到无穷远处，由于是一维的，所以加入条件$u = 0$是合理的。我们取

$$\sigma_1 = 0.3,\ \ \sigma_2 = 0.3,\ \ \rho = 0.3,\ \ r = 0.05,\ \ K = 40,\ \ T = 0.5 \tag{9.41}$$

隐式的欧拉方法在每10步后，对网格进行更新。为了得到无条件稳定性，一阶项将采用典型的加勒金法，这是一种粗糙的近似。

$$\frac{\partial u}{\partial t} + a_1\frac{\partial u}{\partial x} + a_2\frac{\partial u}{\partial y} \approx \frac{1}{\tau}\left(u^{n+1}(x) - u^n(x - \boldsymbol{\alpha}\tau)\right) \tag{9.42}$$

Example 9.21 *[BlackSchol.edp]*

```
                                                    //    文件 BlackScholes2D.edp
int m=30,L=80,LL=80, j=100;
real sigx=0.3, sigy=0.3, rho=0.3, r=0.05, K=40, dt=0.01;
mesh th=square(m,m,[L*x,LL*y]);
fespace Vh(th,P1);

Vh u=max(K-max(x,y),0.);
Vh xveloc, yveloc, v,uold;

for (int n=0; n*dt <= 1.0; n++)
{
  if(j>20)  { th = adaptmesh(th,u,verbosity=1,abserror=1,nbjacoby=2,
              err=0.001, nbvx=5000, omega=1.8, ratio=1.8, nbsmooth=3,
              splitpbedge=1, maxsubdiv=5,rescaling=1) ;
     j=0;
     xveloc = -x*r+x*sigx^2+x*rho*sigx*sigy/2;
     yveloc = -y*r+y*sigy^2+y*rho*sigx*sigy/2;
     u=u;
    };
  uold=u;
  solve eq1(u,v,init=j,solver=LU) = int2d(th)( u*v*(r+1/dt)
        + dx(u)*dx(v)*(x*sigx)^2/2 + dy(u)*dy(v)*(y*sigy)^2/2
        + (dy(u)*dx(v) + dx(u)*dy(v))*rho*sigx*sigy*x*y/2)
        - int2d(th)( v*convect([xveloc,yveloc],dt,w)/dt) + on(2,3,u=0);

  j=j+1;
};
plot(u,wait=1,value=1);
```

结果如图 *9.21)*所示。

9.6 Navier-Stokes 方程

9.6.1 Stokes 和 Navier-Stokes

Stokes方程是：对于给定的$\boldsymbol{f} \in L^2(\Omega)^2$,

$$\left.\begin{array}{rl} -\Delta\boldsymbol{u} + \nabla p & = \boldsymbol{f} \\ \nabla\cdot\boldsymbol{u} & = 0 \end{array}\right\} \quad \text{in } \Omega \tag{9.43}$$

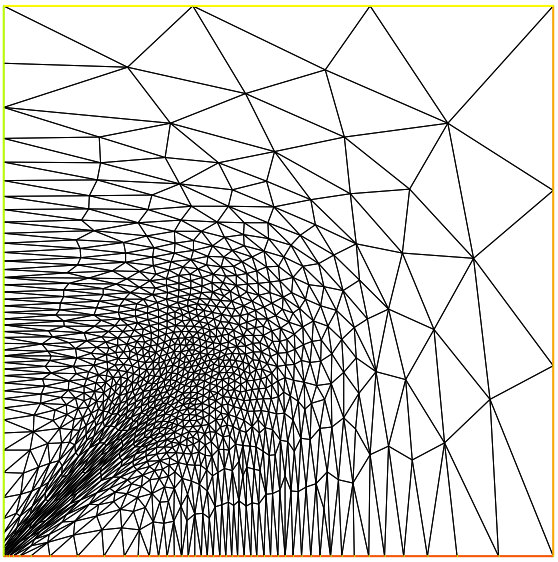


图 9.22: 改进的三角剖分

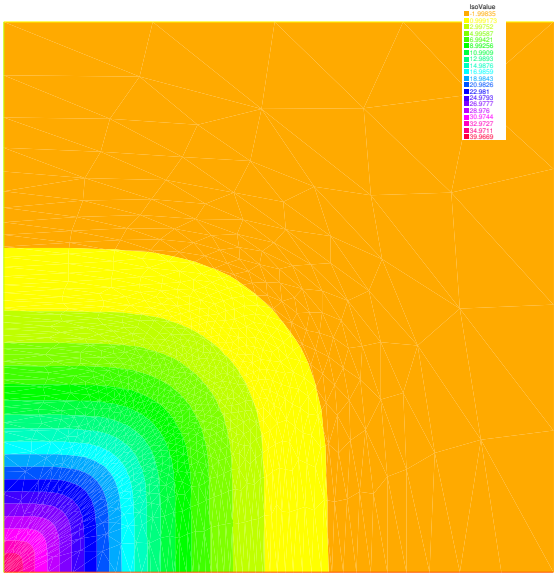


图 9.23: 欧式看跌期权的水平线

其中$\boldsymbol{u} = (u_1, u_2)$是速度矢量，而$p$是压力。为了简化，我们取速度的Dirichlet边界条件，即$\boldsymbol{u} = \boldsymbol{u}_\Gamma$在$\Gamma$.
依据特曼[Theorem 2.2]，我们有(9.43) 的弱形式：即寻找$\boldsymbol{v} = (v_1, v_2) \in \boldsymbol{V}(\Omega)$

$$\boldsymbol{V}(\Omega) = \{\boldsymbol{w} \in H_0^1(\Omega)^2 | \ \mathrm{div}\boldsymbol{w} = 0\}$$

并满足

$$\sum_{i=1}^{2} \int_\Omega \nabla u_i \cdot \nabla v_i = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{w} \quad \text{for all } v \in V$$

这里需要用到$p \in H^1(\Omega)$的存在性使得$\boldsymbol{u} = \nabla p$，如果

$$\int_\Omega \boldsymbol{u} \cdot \boldsymbol{v} = 0 \quad \text{for all } \boldsymbol{v} \in V$$

另一种弱形式推导如下: 我们令

$$\boldsymbol{V} = H_0^1(\Omega)^2; \quad W = \left\{ q \in L^2(\Omega) \,\middle|\, \int_\Omega q = 0 \right\}$$

通过使(9.43)中的第1个方程乘以$v \in V$，第2个方程乘以$q \in W$，随后在Ω 上进行积分，由应用格林公式，我们有

$$\begin{aligned} \int_\Omega \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v} - \int_\Omega \mathrm{div}\boldsymbol{v}\, p &= \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \\ \int_\Omega \mathrm{div}\boldsymbol{u}\, q &= 0 \end{aligned}$$

这就得到了(9.43)的弱形式: 找$(\boldsymbol{u},p)\in \boldsymbol{V}\times W$使得

$$\begin{aligned} a(\boldsymbol{u},\boldsymbol{v})+b(\boldsymbol{v},p) &= (\boldsymbol{f},\boldsymbol{v}) && (9.44)\\ b(\boldsymbol{u},q) &= 0 && (9.45)\end{aligned}$$

对于所有的$(\boldsymbol{v},q)\in V\times W$，其中

$$\begin{aligned} a(\boldsymbol{u},\boldsymbol{v}) &= \int_\Omega \nabla\boldsymbol{u}\cdot\nabla\boldsymbol{v}=\sum_{i=1}^{2}\int_\Omega \nabla u_i\cdot\nabla v_i && (9.46)\\ b(\boldsymbol{u},q) &= -\int_\Omega \mathrm{div}\boldsymbol{u}\, q && (9.47)\end{aligned}$$

现在我们考虑有限元空间，$\boldsymbol{V}_h\subset\boldsymbol{V}$以及$W_h\subset W$，并假设下述基函数

$$\begin{aligned} &\boldsymbol{V}_h=V_h\times V_h,\quad V_h=\{v_h|\ v_h=v_1\phi_1+\cdots+v_{M_V}\phi_{M_V}\},\\ &W_h=\{q_h|\ q_h=q_1\varphi_1+\cdots+q_{M_W}\varphi_{M_W}\}\end{aligned}$$

则离散的弱形式如下: 找$(\boldsymbol{u}_h,p_h)\in\boldsymbol{V}_h\times W_h$使得

$$\begin{aligned} a(\boldsymbol{u}_h,\boldsymbol{v}_h)+b(\boldsymbol{v}_h,p) &= (\boldsymbol{f},\boldsymbol{v}_h), && \forall\boldsymbol{v}_h\in\boldsymbol{V}_h\\ b(\boldsymbol{u}_h,q_h) &= 0, && \forall q_h\in W_h\end{aligned} \qquad (9.48)$$

Note 9.4 假设*:*

1. 存在常数$\alpha_h>0$使得
$$a(\boldsymbol{v}_h,\boldsymbol{v}_h)\geq\alpha\|\boldsymbol{v}_h\|_{1,\Omega}^2\quad \textit{for all } \boldsymbol{v}_h\in Z_h$$
其中
$$Z_h=\{\boldsymbol{v}_h\in\boldsymbol{V}_h|\ b(\boldsymbol{w}_h,q_h)=0\quad \textit{for all } q_h\in W_h\}$$

2. 存在常数$\beta_h>0$使得
$$\sup_{\boldsymbol{v}_h\in\boldsymbol{V}_h}\frac{b(\boldsymbol{v}_h,q_h)}{\|\boldsymbol{v}_h\|_{1,\Omega}}\geq\beta_h\|q_h\|_{0,\Omega}\quad \textit{for all } q_h\in W_h$$

那么对于 *(9.48)*，我们有唯一的解 $(\boldsymbol{u}_h,p_h)$ 满足

$$\|\boldsymbol{u}-\boldsymbol{u}_h\|_{1,\Omega}+\|p-p_h\|_{0,\Omega}\leq C\left(\inf_{\boldsymbol{v}_h\in\boldsymbol{V}_h}\|u-v_h\|_{1,\Omega}+\inf_{q_h\in W_h}\|p-q_h\|_{0,\Omega}\right)$$

当常数$C>0$时 *(详见案例 [20, Theorem 10.4]).*

我们令如下记号

$$\begin{aligned} A &= (A_{ij}),\ A_{ij}=\int_\Omega\nabla\phi_j\cdot\nabla\phi_i \qquad i,j=1,\cdots,M_{\boldsymbol{V}} && (9.49)\\ \boldsymbol{B} &= (Bx_{ij},By_{ij}),\ Bx_{ij}=-\int_\Omega\partial\phi_j/\partial x\,\varphi_i \qquad By_{ij}=-\int_\Omega\partial\phi_j/\partial y\,\varphi_i\\ &\qquad i=1,\cdots,M_W;j=1,\cdots,M_V\end{aligned}$$

那么(9.48)可以写作

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^* \\ \boldsymbol{B} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_h \\ \{p_h\} \end{pmatrix} = \begin{pmatrix} \boldsymbol{F}_h \\ 0 \end{pmatrix} \tag{9.50}$$

其中

$$\boldsymbol{A} = \begin{pmatrix} A & 0 \\ 0 & A \end{pmatrix} \qquad \boldsymbol{B}^* = \begin{Bmatrix} Bx^T \\ By^T \end{Bmatrix} \qquad \boldsymbol{U}_h = \begin{Bmatrix} \{u_{1,h}\} \\ \{u_{2,h}\} \end{Bmatrix} \qquad \boldsymbol{F}_h = \begin{Bmatrix} \{\int_\Omega f_1\phi_i\} \\ \{\int_\Omega f_2\phi_i\} \end{Bmatrix}$$

补偿法: 这一方法包括替换(9.48)式为一个更常规的问题：寻找 $(\boldsymbol{v}_h^\epsilon, p_h^\epsilon) \in \boldsymbol{V}_h \times \tilde{W}_h$ 以满足

$$\begin{array}{lll} a(\boldsymbol{u}_h^\epsilon, \boldsymbol{v}_h) + b(\boldsymbol{v}_h, p_h^\epsilon) & = (\boldsymbol{f}, \boldsymbol{v}_h), & \forall \boldsymbol{v}_h \in \boldsymbol{V}_h \\ b(\boldsymbol{u}_h^\epsilon, q_h) - \epsilon(p_h^\epsilon, q_h) & = 0, & \forall q_h \in \tilde{W}_h \end{array} \tag{9.51}$$

其中$\tilde{W}_h \subset L^2(\Omega)$. 形式上，我们有

$$\mathrm{div}\boldsymbol{u}_h^\epsilon = \epsilon p_h^\epsilon$$

以及相应的代数问题

$$\begin{pmatrix} \boldsymbol{A} & B^* \\ B & -\epsilon I \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_h^\epsilon \\ \{p_h^\epsilon\} \end{pmatrix} = \begin{pmatrix} \boldsymbol{F}_h \\ 0 \end{pmatrix}$$

Note 9.5 我们能消去 $p_h^\epsilon = (1/\epsilon)BU_h^\epsilon$并得到

$$(A + (1/\epsilon)B^*B)\boldsymbol{U}_h^\epsilon = \boldsymbol{F}_h^\epsilon \tag{9.52}$$

由于矩阵$A + (1/\epsilon)B^*B$是对称的，正定的以及稀疏的，上式*(9.52)*可以有技巧地解决。 存在常数$C > 0$独立于ϵ使得

$$\|\boldsymbol{u}_h - \boldsymbol{u}_h^\epsilon\|_{1,\Omega} + \|p_h - p_h^\epsilon\|_{0,\Omega} \leq C\epsilon$$

*(*详见案例. *[20, 17.2])*

Example 9.22 (Cavity.edp) 受驱动的空腔流问题最早解决了*Reynolds*数为*0*的情况*(Stokes* 流*)*，而后解决了*Reynolds*数为*100*的情况。 速度压力等式被最早应用，后来流量函数与速度等式重复了类似的计算方法。
我们通过补偿法来求解受驱动的空腔流问题*(9.51)* 其中在上边界， $\boldsymbol{u}_\Gamma \cdot \boldsymbol{n} = 0$ 以及$\boldsymbol{u}_\Gamma \cdot \boldsymbol{s} = 1$ ，而在其余的边界，$\boldsymbol{u}_\Gamma = 0$, *(* $\boldsymbol{n}$是Γ的单位法线，而$\boldsymbol{s}$ 是Γ的单位切线*)*。
网格通过如下建立

```
mesh Th=square(8,8);
```

我们利用经典的*Taylor-Hood*元技巧来解决这一问题*:*

速度由P_2有限元空间近似*(* X_h空间*)*，而压力由P_1有限元空间近似*(* M_h空间*)*，

其中

$$X_h = \left\{ \boldsymbol{v} \in H^1(]0,1[^2) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \right\}$$

以及

$$M_h = \left\{ v \in H^1(]0,1[^2) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \right\}$$

有限元空间和函数构造如下

```
fespace Xh(Th,P2);                                    //    速度分量空间的定义
fespace Mh(Th,P1);                                    //    压强空间的定义
Xh u2,v2;
Xh u1,v1;
Mh p,q;
```

*Stokes*算子是速度$(u1,u2)$ 与压力p的一种系统解法。速度的测试函数是$(v1,v2)$，而压力的测试函数是q，所以*(9.48)*式的变分形式用*freefem*语言来写如下*:*

```
solve Stokes (u1,u2,p,v1,v2,q,solver=Crout) =
    int2d(Th)( ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
               +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
               - p*q*(0.000001)
               - p*dx(v1) - p*dy(v2)
               - dx(u1)*q - dy(u2)*q
             )
  + on(3,u1=1,u2=0)
  + on(1,2,4,u1=0,u2=0);                              //    标签 1,2,3,4 见 ??
```

每一个未知量都有它们各自的边界条件。

如果要求流线型，那么需要找到ψ使得$rot\psi = u$或者如下

$$-\Delta\psi = \nabla \times u$$

```
Xh psi,phi;

solve streamlines(psi,phi) =
      int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
   +  int2d(Th)( -phi*(dy(u1)-dx(u2)))
   +  on(1,2,3,4,psi=0);
```

现在*Navier-Stokes*方程可以通过解下述方程

$$\frac{\partial u}{\partial t} + u\cdot\nabla u - \nu\Delta u + \nabla p = 0, \quad \nabla\cdot u = 0$$

并伴有相同的边界条件及初始条件$u = 0$。
具体是通过对流算子`convect`来执行的，对于$\frac{\partial u}{\partial t} + u\cdot\nabla u$这一项，给出离散化的式子

$$\begin{array}{ll} \frac{1}{\tau}(u^{n+1} - u^n\circ X^n) - \nu\Delta u^{n+1} + \nabla p^{n+1} & = 0, \\ \nabla\cdot u^{n+1} & = 0 \end{array} \tag{9.53}$$

$u^n\circ X^n(x) \approx u^n(x - u^n(x)\tau)$ 这一项可以通过“对流”算子来解，所以我们得到

```
int i=0;
real  nu=1./100.;
real dt=0.1;
real alpha=1/dt;
```

```
Xh up1,up2;

problem  NS (u1,u2,p,v1,v2,q,solver=Crout,init=i) =
    int2d(Th)(
              alpha*( u1*v1 + u2*v2)
            + nu * ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
            +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
            - p*q*(0.000001)
            - p*dx(v1) - p*dy(v2)
            - dx(u1)*q - dy(u2)*q
           )
  + int2d(Th) ( -alpha*
       convect([up1,up2],-dt,up1)*v1 -alpha*convect([up1,up2],-dt,up2)*v2 )
  + on(3,u1=1,u2=0)
  + on(1,2,4,u1=0,u2=0)
;

for (i=0;i<=10;i++)
 {
   up1=u1;
   up2=u2;
   NS;
   if ( !(i % 10))                                          //     每10次迭代画一次图
    plot(coef=0.2,cmm=" [u1,u2] and p  ",p,[u1,u2]);
 } ;
```

注意到刚度矩阵是 是再生的(关键字 *init=i*)

9.6.2 Uzawa算法与共轭梯度

我们解不带罚项的Stokes问题。下面的算法描述了古典的Uzawa迭代法（例如，参见[20, 17.3], [29, 13] 或 [30, 13] ）：

初始化: 令p_h^0为$L^2(\Omega)$中任选的一个元素。

计算$\boldsymbol{u}_h$: 若已知p_h^n，则$\boldsymbol{v}_h^n$为

$$\boldsymbol{u}_h^n = A^{-1}(\boldsymbol{f}_h - \boldsymbol{B}^* p_h^n)$$

的解。

更新 p_h: 将p_h^{n+1}定义为

$$p_h^{n+1} = p_h^n + \rho_n \boldsymbol{B}\boldsymbol{u}_h^n。$$

存在常数$\alpha > 0$使得对每个n，有$\alpha \le \rho_n \le 2$，则$\boldsymbol{u}_h^n$收敛于解$\boldsymbol{u}_h$，且由更新项p_h可知当$n \to \infty$，有$B\boldsymbol{v}_h^n \to 0$。 这个方法一般收敛地很慢。
首先我们定义网格和Taylor-Hood近似。 因此X_h为速度空间，M_h为压力空间。

Example 9.23 (StokesUzawa.edp)

```
mesh Th=square(10,10);
fespace Xh(Th,P2),Mh(Th,P1);
Xh u1,u2,v1,v2;
Mh p,q,ppp;                                                 //     ppp 是工作压强
```

```
varf bx(u1,q) = int2d(Th)( -(dx(u1)*q));
varf by(u1,q) = int2d(Th)( -(dy(u1)*q));
varf a(u1,u2)= int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) )
                    +  on(3,u1=1)  +  on(1,2,4,u1=0) ;
                                //    注:  把  on(3,u1=1) 放在  on(1,2,4,u1=0) 前
                                                          //    因为我们希望在交集上为0 %

matrix A= a(Xh,Xh,solver=CG);
matrix Bx= bx(Xh,Mh);                                                  //    B = (Bx  By)
matrix By= by(Xh,Mh);

Xh bc1; bc1[] = a(0,Xh);                                          //    在 u1 上的边界条件的贡献
Xh bc2; bc2   = 0 ;                                               //    在 u2 上没有边界条件贡献
Xh b;
```

$p_h^n \to \boldsymbol{B}A^{-1}(-\boldsymbol{B}^* p_h^n) = -div\ \boldsymbol{u}_h$ 通过下面的函数divup实现。

```
func real[int] divup(real[int] & pp)
{
                                                                    //    计算 u1(pp)
   b[]  = Bx'*pp; b[] *=-1; b[] += bc1[] ;    u1[] = A^-1*b[];
                                                                    //    计算 u2(pp)
   b[]  = By'*pp; b[] *=-1; b[] += bc2[] ;    u2[] = A^-1*b[];
   //    u^n = A^{-1}(Bx^T p^n   By^T p^n)^T
   ppp[] =   Bx*u1[];                                          //    ppp = Bx u_1
   ppp[] +=  By*u2[];                                          //        +By u_2
   return ppp[] ;
};
```

现在调用共轭梯度法：

```
p=0;q=0;                                                            //    p_h^0 = 0
LinearCG(divup,p[],eps=1.e-6,nbiter=50);                    //    p_h^{n+1} = p_h^n + B u_h^n
//    若 n > 50 或 |p_h^{n+1} - p_h^n| ≤ 10^{-6}, 则循环停止.
divup(p[]);                                                         //    计算最终的解

plot([u1,u2],p,wait=1,value=true,coef=0.1);
```

9.6.3 NSUzawaCahouetChabart.edp

本例中我们使用由Cahouet-Chabart预处理过的Uzawa算法，解经过一个圆柱的Navier-Stokes方程（详见[31]）。
预处理子的思想是在一个周期区域中，所有微分算子交换次序，并且由于我们用Uzawa算法来解线性算子$\nabla\cdot(\alpha Id+\nu\Delta)^{-1}\nabla$，其中$Id$是恒等算子，因此这表示预处理子是$\alpha\Delta^{-1}+\nu Id$。

为了实现它，我们运行

Example 9.24 (NSUzawaCahouetChabart.edp)

```
real D=0.1, H=0.41;
real cx0 = 0.2, cy0 = 0.2;                                          //    圆心
real xa = 0.15, ya=0.2, xe = 0.25,ye =0.2;
border fr1(t=0,2.2){x=t; y=0; label=1;}
```

```
border fr2(t=0,H){x=2.2; y=t; label=2;}
border fr3(t=2.2,0){x=t; y=H; label=1;}
border fr4(t=H,0){x=0; y=t; label=1;}
border fr5(t=2*pi,0){x=cx0+D*sin(t)/2; y=cy0+D*cos(t)/2; label=3;}
int nn=15;

mesh Th=buildmesh(fr1(5*nn)+fr2(nn)+fr3(5*nn)+fr4(nn)+fr5(-nn*3));
real Um= 1.5;                                                    //    最大速度 (Rey 100)
func Ub = Um*2./3.;
real nu = 1e-3;
real Rey = Ub*D/nu;
                                                                 //    边界条件
func U1 = 4.*Um*y*(H-y)/(H*H)  ;
func U2 = 0. ;

real T=2,t=0;
real dt = D/nn/Um;                                               //    CFL = 1
 cout << " dt = " << dt <<endl;
real alpha=1/dt,epspq=1e-10;


fespace Mh(Th,[P1]);
fespace Xh(Th,[P2]);
fespace Wh(Th,[P1dc]);
macro grad(u) [dx(u),dy(u)]                                      //
macro div(u1,u2) (dx(u1)+dy(u2))                                 //


 varf von1([u1,u2,p],[v1,v2,q]) =  on(3,u1=0,u2=0) + on(1,u1=U1,u2=U2);


                                  //   注 :  下一行的值100是手动加入的，因为出口为0.
 varf vA(p,q) =int2d(Th)((grad( p ) '*grad(q)) ) + int1d(Th,2)(100*p*q) ;

 varf vM(p,q) =int2d(Th,qft=qf2pT)(  p*q )+ on(2,p=0);

 varf vu([u1],[v1]) = int2d(Th)(alpha*(u1*v1)+nu*(grad(u1)'*grad(v1) ))
                       + on(1,3,u1=0) ;
 varf vu1([p],[v1]) = int2d(Th)(p*dx(v1)) ;
 varf vu2([p],[v1]) = int2d(Th)(p*dy(v1)) ;


 matrix pAM=vM(Mh,Mh,solver=UMFPACK);
 matrix pAA=vA(Mh,Mh,solver=UMFPACK);
 matrix AU=vu(Xh,Xh,solver=UMFPACK);
 matrix B1=vu1(Mh,Xh);
 matrix B2=vu2(Mh,Xh);
 Xh u1,u2;
 Mh p;
varf vonu1([u1],[v1]) =  on(1,u1=U1) + on(3,u1=0);
varf vonu2([u1],[v1]) =  on(1,u1=U2) + on(3,u1=0);


real[int] brhs1 = vonu1(0,Xh);
real[int] brhs2 = vonu2(0,Xh);
```

```
varf  vrhs1(uu,vv)  = int2d(Th) (convect([u1,u2],-dt,u1)*vv*alpha)+vonu1 ;
varf  vrhs2(v2,v1)  = int2d(Th) (convect([u1,u2],-dt,u2)*v1*alpha)+vonu2;
```

定义*Uzawa*和预处理子部分的函数。

```
func  real[int]   JUzawa(real[int] & pp)
{
real[int] b1=brhs1; b1 += B1*pp;
real[int] b2=brhs2; b2 += B2*pp;
u1[] = AU^-1 * b1;
u2[] = AU^-1 * b2;
pp  = B1'*u1[];
pp += B2'*u2[];
pp = -pp;
return pp;
}

func  real[int]   Precon(real[int] & p)
 {
    real[int] pa= pAA^-1*p;
    real[int] pm= pAM^-1*p;
    real[int] pp= alpha*pa+nu*pm;
   return pp;
 }
```

在时间上的循环。
因为我们从之前一个解开始且它接近最终解，所以当检测到满足终止条件的时候，给出警告。要取绝对，而不是相对的终止条件（此处是负的）。

```
 verbosity = 0;
 p=0;


 Wh w;                                                    //     储存涡度 ..

 real eps=1e-6;
 int ndt = T/dt;
 for(int i=0;i<ndt;++i)
 {
     brhs1 = vrhs1(0,Xh);
     brhs2 = vrhs2(0,Xh);
     int res=LinearCG(JUzawa,p[],precon=Precon,nbiter=100,verbosity=10,veps=eps);
     assert(res==1) ;
     eps = -abs(eps);
     w = -dy(u1)+dx(u2);
     plot(w,fill=1,wait=0, nbiso=40);

     dt = min(dt,T-t);
     t += dt;
     if( dt < 1e-10*T) break;
 }
 plot(w,fill=1,wait=0, nbiso=40,ps="NScahouetChabart");          //     见图 9.24

 cout << " u1 max " << u1[].linfty
      << " u2 max " << u2[].linfty
```

```
<< " p max = " << p[].max << endl;
```

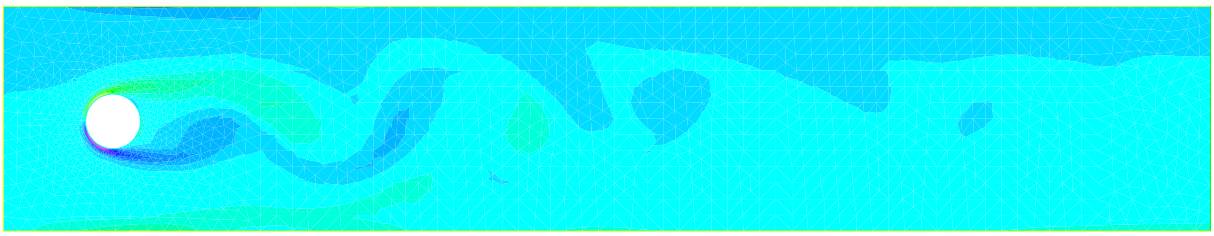


图 9.24: 使用Cahouet-Chabart得到在Reynolds数为100, 时间为2秒时的涡度.

9.7 变分不等式

我们举出变分不等式的一个古典例子。
令 $\mathcal{C} = \{u \in H_0^1(\Omega), u \leq g\}$。
要解决的问题是：

$$u = arg \min_{u \in \mathcal{C}} J(u) = \frac{1}{2} \int_\Omega \nabla u \cdot \nabla u - \int_\Omega fu,$$

其中f和g是给定的函数。
它的解是对$H_0^1(\Omega)$的标量积$((v,w)) = \int_\Omega \nabla v \cdot \nabla w$，在$f^\star$的凸集$\mathcal{C}$上的投影，其中$f^*$是$((f^\star, v)) = \int_\Omega fv, \forall v \in H_0^1(\Omega)$的解。在一个凸集上的投影明显满足$\forall v \in \mathcal{C}, \quad ((u-v, u-\tilde{f})) \leq 0$，且经过扩展，我们得到古典不等式

$$\forall v \in \mathcal{C}, \quad \int_\Omega \nabla(u-v)\nabla u \leq \int_\Omega (u-v)f。$$

我们也能将问题重写为一个鞍点问题：
找到λ, u使得

$$\max_{\lambda \in L^2(\Omega), \lambda \geq 0} \min_{u \in H_0^1(\Omega)} \mathcal{L}(u,\lambda) = \frac{1}{2} \int_\Omega \nabla u.\nabla u - \int_\Omega fu + \int_\Omega \lambda (u-g)^+$$

成立，其中$((u-g)^+ = max(0, u-g)$。
这个鞍点问题等价于找到u, λ使得：

$$\begin{cases} \int_\Omega \nabla u \cdot \nabla v + \lambda v^+ \, d\omega = \int_\Omega fu, & \forall v \in H_0^1(\Omega) \\ \int_\Omega \mu (u-g)^+ = 0, & \forall \mu \in L^2(\Omega), \mu \geq 0, \lambda \geq 0, \end{cases} \tag{9.54}$$

一个解前面问题的算法是：

1. k=0, 在 $H^{-1}(\Omega)$ 中选λ_0
2. 按 $k = 0, ...$ 循环
   (a) 令 $\mathcal{I}_k = \{x \in \Omega / \lambda_k + c * (u_{k+1} - g) \leq 0\}$

(b) $V_{g,k+1} = \{v \in H_0^1(\Omega)/v = g \text{ on } I_k\}$,

(c) $V_{0,k+1} = \{v \in H_0^1(\Omega)/v = 0 \text{ on } I_k\}$,

(d) 找到 $u_{k+1} \in V_{g,k+1}$ 和 $\lambda_{k+1} \in H^{-1}(\Omega)$ 使得

$$\begin{cases} \int_\Omega \nabla u_{k+1} \cdot \nabla v_{k+1}\, d\omega = \int_\Omega f v_{k+1}, & \forall v_{k+1} \in V_{0,k+1} \\ < \lambda_{k+1}, v >= \int_\Omega \nabla u_{k+1} \cdot \nabla v - f v\, d\omega \end{cases}$$

其中 $<,>$是表示$H_0^1(\Omega)$和$H^{-1}(\Omega)$之间的对偶的符号，且c是一个加罚常参数（足够大）。

读者可在[33]中找到所有关于这个算法的数学理论。
现在说明如何在FreeFem++中实现这个算法。
下面是一个完整的例子：

Example 9.25 (VI.edp)

```
mesh Th=square(20,20);
real eps=1e-5;
fespace Vh(Th,P1);                                          //  P1 有限元空间
int n = Vh.ndof;                                            //  自由度个数
Vh uh,uhp;                                                  //  解和上一步的解
Vh Ik;                                                      //  定义约束条件取到的集合
real[int] rhs(n);                                           //  储存方程的右端项
real c=1000;                                                //  算法的惩罚项参数
func f=1;                                                   //  右端项
func fd=0;                                                  //  Dirichlet 边界条件函数
Vh g=0.05;                                                  //  离散函数 g

real[int] Aii(n),Aiin(n);                                   //  储存矩阵的对角元的两个版本

real tgv = 1e30;                                            //  对于边界条件的精确惩罚项来说
                                                            //  这是一个很大的值
//  问题的变分形式:
varf a(uh,vh) =                                             //  问题的定义
    int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )              //  双线性型
  - int2d(Th)( f*vh )                                       //  线性型
  + on(1,2,3,4,uh=fd) ;                                     //  边界条件形式


//  问题的矩阵的两个版本
matrix A=a(Vh,Vh,tgv=tgv,solver=CG);                        //  改变的版本
matrix AA=a(Vh,Vh,solver=CG);                               //  为了计算残量的版本

 //  构造质量矩阵:
varf vM(uh,vh) = int2d(Th)(uh*vh);
matrix M=vM(Vh,Vh);                  //  L^2 范数的快速计算 : sqrt( u'*(w=M*u))

Aii=A.diag;                          //  得到矩阵的对角元 (在版本 1.46-1 中出现)

rhs = a(0,Vh,tgv=tgv);
Ik =0;
uhp=-tgv;                                                   //  上一步的值是
```

```
Vh lambda=0;
for(int iter=0;iter<100;++iter)
{
  real[int] b(n) ; b=rhs;                                          //    复制右端项
  real[int] Ak(n);                                                 //    Ik 的补集 ( !Ik = (Ik-1))
                                                                   //    现在算子 Ik- 1.  不能实现, 所以我们:
  Ak= 1.; Ak  -= Ik[];                                             //    build Ak = !  Ik
                                                                   //    在b上和对角元上增加新的闭锁条件, 如果 (Ik ==1 )
  b = Ik[] .* g[];       b *= tgv;      b  -=  Ak .* rhs;
  Aiin = Ik[] *  tgv;       Aiin  +=  Ak  .* Aii;                  //    设定 Aii= tgv i∈Ik
  A.diag = Aiin;                                                   //    设定矩阵对角元 (在版本 1.46-1 中出现)
  set(A,solver=CG);                                                //    对于求解方程来说改变预处理很重要
  uh[] = A^-1* b;                                                  //    使用更多的闭锁条件来求解方程
  lambda[] = AA * uh[];                                            //    计算残量 ( 在矩阵形式下计算很快)
  lambda[] += rhs;                                                 //    注 rhs = -∫fv

  Ik = ( lambda + c*( g- uh)) < 0.;                                //    新的闭锁值

   plot(Ik, wait=1,cmm=" lock set ",value=1,ps="VI-lock.eps",fill=1 );
   plot(uh,wait=1,cmm="uh",ps="VI-uh.eps");
                                                                   //    计算变分的 L^2 范数 (快速方法)的技巧
      real[int] diff(n),Mdiff(n);
      diff= uh[]-uhp[];
      Mdiff = M*diff;
      real err = sqrt(Mdiff'*diff);
  cout << "  || u_{k=1} - u_{k} ||_2 " << err << endl;
  if(err< eps) break;                                              //    stop test
  uhp[]=uh[] ;                                                     //    设定上一步的解
}
savemesh(Th,"mm",[x,y,uh*10]);                                     //    为了 medit 画图
```

注意，正如你能在该例中所看到的，一些向量，或者矩阵算符并没有实现，因此一种方法是略过这些表述。此外，我们使用算符+=, -=来融合结果。

9.8 区域分解

我们举出区域分解技术的三个古典例子：首先是带重叠的Schwarz算法，其次是不带重叠（也称作Schur补）的Schwarz算法，最后我们演示使用共轭梯度法来解Schur补的边界问题。

9.8.1 Schwarz 重叠格式

为解

$$-\Delta u = f, \quad \text{in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0,$$

如下进行Schwarz格式

$$\begin{aligned} -\Delta u_1^{n+1} &= f \text{ in } \Omega_1 \quad u_1^{n+1}|_{\Gamma_1} = u_2^n \\ -\Delta u_2^{n+1} &= f \text{ in } \Omega_2 \quad u_2^{n+1}|_{\Gamma_2} = u_1^n, \end{aligned}$$

其中Γ_i是Ω_i的边界，它们需要满足条件$\Omega_1 \cap \Omega_2 \neq \emptyset$，且$u_i$在迭代步1处为0。

此处我们取Ω_1为四边形，Ω_2为圆盘，且以0为初始值应用这个算法。

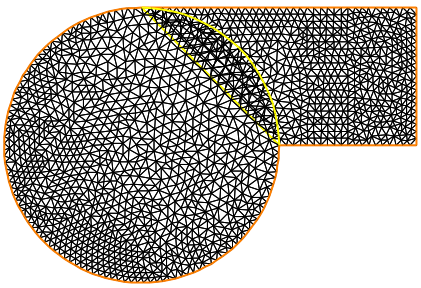


图 9.25: 两个重叠的网格TH和th

Example 9.26 (Schwarz-overlap.edp)

```
int inside = 2;                                                    //    内边界
int outside = 1;                                                   //    外边界
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t ;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, pi/2){ x= cos(t); y = sin(t);label=inside;};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
mesh th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
mesh TH = buildmesh( e(5*n) + e1(25*n) );
plot(th,TH,wait=1);                                                //    观察两个网格
```

空间和问题定义如下：

```
fespace vh(th,P1);
fespace VH(TH,P1);
vh u=0,v; VH U,V;
int i=0;

problem PB(U,V,init=i,solver=Cholesky) =
    int2d(TH)( dx(U)*dx(V)+dy(U)*dy(V) )
  + int2d(TH)( -V) + on(inside,U = u)  + on(outside,U= 0 ) ;
problem pb(u,v,init=i,solver=Cholesky) =
    int2d(th)( dx(u)*dx(v)+dy(u)*dy(v) )
  + int2d(th)( -v) + on(inside ,u = U) + on(outside,u = 0 ) ;
```

计算的循环：

```
for ( i=0 ;i< 10; i++)
{
   PB;
   pb;
   plot(U,u,wait=true);
};
```

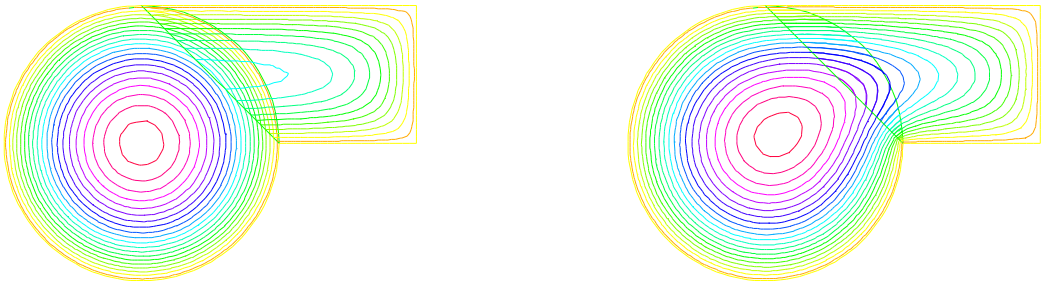


图 9.26: 在迭代步0和迭代步9处的等值线

9.8.2 Schwarz非重叠格式

为解

$$-\Delta u = f \text{ in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0,$$

如下进行对不带重叠的区域分解的Schwarz算法：

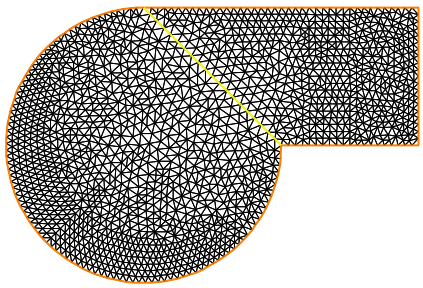


图 9.27: 两个不重叠的网格TH和th

令Γ_i为Ω_1和 Ω_2 的共同边界，且$\Gamma_e^i = \partial\Omega_i \setminus \Gamma_i$。
该问题找到λ使得$(u_1|_{\Gamma_i} = u_2|_{\Gamma_i})$，其中$u_i$是下面Laplace问题的解

$$-\Delta u_i = f \text{ in } \Omega_i \quad \frac{\partial u_i}{\partial n_i}\Big|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma_e^i} = 0$$

为解这个问题我们只需做一个循环，更新λ的值

$$\lambda = \lambda \pm \frac{(u_1 - u_2)}{2}$$

来实现， 其中从$\pm$中选择符号$+$或$-$使得格式收敛。

Example 9.27 (Schwarz-no-overlap.edp)

```
                                                    //      非重叠的schwarz方法
int inside = 2;
int outside = 1;
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t ;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, 1){ x= 1-t; y = t;label=inside;};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
mesh th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
mesh TH = buildmesh ( e(5*n) + e1(25*n) );
plot(th,TH,wait=1,ps="schwarz-no-u.eps");
fespace vh(th,P1);
fespace VH(TH,P1);
vh u=0,v; VH U,V;
vh lambda=0;
int i=0;

problem PB(U,V,init=i,solver=Cholesky) =
    int2d(TH)( dx(U)*dx(V)+dy(U)*dy(V) )
  + int2d(TH)( -V)
  + int1d(TH,inside)(lambda*V) +    on(outside,U= 0 ) ;
problem pb(u,v,init=i,solver=Cholesky) =
    int2d(th)( dx(u)*dx(v)+dy(u)*dy(v) )
  + int2d(th)( -v)
  + int1d(th,inside)(-lambda*v) +    on(outside,u = 0 ) ;



for ( i=0 ;i< 10; i++)
{
   PB;
   pb;
   lambda = lambda - (u-U)/2;
   plot(U,u,wait=true);
};

plot(U,u,ps="schwarz-no-u.eps");
```

9.8.3 Schwarz-gc.edp

为解

$$-\Delta u = f \text{ in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0,$$

如下进行对不带重叠的区域分解的Schwarz算法：
令Γ_i为Ω_1和 Ω_2 的共同边界，且$\Gamma_e^i = \partial\Omega_i \setminus \Gamma_i$。
该问题找到λ使得$(u_1|_{\Gamma_i} = u_2|_{\Gamma_i})$，其中$u_i$是下面Laplace问题的解

$$-\Delta u_i = f \text{ in } \Omega_i \quad \frac{\partial u_i}{\partial n_i}\Big|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma_e^i} = 0$$

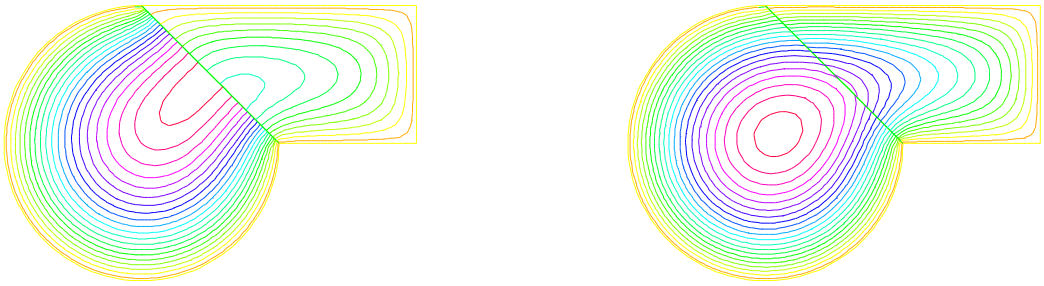


图 9.28: 不带重叠的格式在迭代步0和迭代步9处的解的等值线

该例子是对Schur补的版本。我们使用共轭梯度法来解其边界问题。
首先，我们构造这两个区域。

Example 9.28 (Schwarz-gc.edp)

```
                                        //     非重叠的Schwarz方法 (Shur 补 Neumann -> Dirichet)
real cpu=clock();
int inside = 2;
int outside = 1;

border Gamma1(t=1,2){x=t;y=0;label=outside;};
border Gamma2(t=0,1){x=2;y=t;label=outside;};
border Gamma3(t=2,0){x=t ;y=1;label=outside;};

border GammaInside(t=1,0){x = 1-t; y = t;label=inside;};

border GammaArc(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
                                                                  //     建立 Ω1 和 Ω2 上网格
mesh Th1 = buildmesh( Gamma1(5*n) + Gamma2(5*n) + GammaInside(5*n) + Gamma3(5*n));
mesh Th2 = buildmesh ( GammaInside(-5*n) + GammaArc(25*n) );
plot(Th1,Th2);

                                                                  //     定义 2 FE 空间
fespace Vh1(Th1,P1),       Vh2(Th2,P1);
```

Note 9.6 不可能根据部分边界值就能定义一个函数， 所以 λ 函数必须在整个定义域 Ω_1 上有定义
例如

```
Vh1 lambda=0;                                                     //     取 λ ∈ Vh1
```

两个泊松问题:

```
Vh1 u1,v1;                Vh2 u2,v2;
int i=0;                                                          //     因子优化问题
```

```
problem Pb2(u2,v2,init=i,solver=Cholesky) =
    int2d(Th2)( dx(u2)*dx(v2)+dy(u2)*dy(v2) )
  + int2d(Th2)( -v2)
  + int1d(Th2,inside)(-lambda*v2) +    on(outside,u2= 0 ) ;
problem Pb1(u1,v1,init=i,solver=Cholesky) =
    int2d(Th1)( dx(u1)*dx(v1)+dy(u1)*dy(v1) )
  + int2d(Th1)( -v1)
  + int1d(Th1,inside)(+lambda*v1) +    on(outside,u1 = 0 ) ;
```

或者我们定义一个边界矩阵， 由于 λ 函数在定义域 Ω_1 非零:

```
varf b(u2,v2,solver=CG) =int1d(Th1,inside)(u2*v2);
matrix B= b(Vh1,Vh1,solver=CG);
```

边界问题函数,

$$\lambda \longrightarrow \int_{\Gamma_i} (u_1 - u_2)v_1$$

```
func real[int] BoundaryProblem(real[int] &l)
{
   lambda[]=l;                                          //    使 FE 函数有1形式
   Pb1;     Pb2;
   i++;                                                 //    不重构 i !=0
   v1=-(u1-u2);
   lambda[]=B*v1[];
   return lambda[] ;
};
```

Note 9.7 符号 *v1* 和 *v1[]* 的区别是: *v1* 是有限元函数，而 *v1[]* 有限元函数 *v1* 的典范基下的向量 .

```
Vh1 p=0,q=0;
                                                        //    利用共轭梯度法
LinearCG(BoundaryProblem,p[],eps=1.e-6,nbiter=100);
                                                    //    计算最终解, 因为 CG 与增量有关
BoundaryProblem(p[]);                               //    再次计算得出正确的 u1,u2

cout << " -- CPU time  schwarz-gc:" <<  clock()-cpu << endl;
plot(u1,u2);                                                //    画图
```

9.9 流体/结构耦合问题

这个问题涉及 Lamé 系统弹性问题 以及 Stokes 系统粘滞流体在速度 $\boldsymbol{u}$ 以及压强 p 下的问题:

$$-\Delta \boldsymbol{u} + \boldsymbol{\nabla} p = 0, \nabla \cdot \boldsymbol{u} = 0, \quad \text{in} \quad \Omega, \boldsymbol{u} = \boldsymbol{u}_\Gamma \quad \text{on} \quad \Gamma = \partial\Omega$$

其中u_Γ是边界上的速度. 边界上被施压的是法向应力$\boldsymbol{h} = (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)\boldsymbol{n} - p\boldsymbol{n}$

弹性固体服从力的形变: 在固体的某点, (x,y) 处 之后到达 (X,Y). 当分解向量 $\boldsymbol{v} = (v_1, v_2) = (X - x, Y - y)$ 很小时, Hooke 定律将固体内的应力张量σ 变成形变张量 ϵ:

$$\sigma_{ij} = \lambda\delta_{ij}\nabla.\boldsymbol{v} + 2\mu\epsilon_{ij},\ \epsilon_{ij} = \frac{1}{2}(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i})$$

这里 δ 是 Kronecker 符号 以及 λ, μ 是用来描述物体弹性模量和杨氏模量的两个常数
弹性方程一般写成如下的变分形式，其中分解向量 $v(x) \in V$

$$\int_\Omega [2\mu\epsilon_{ij}(\boldsymbol{v})\epsilon_{ij}(\boldsymbol{w}) + \lambda\epsilon_{ii}(v)\epsilon_{jj}(\boldsymbol{w})] = \int_\Omega \boldsymbol{g}\cdot\boldsymbol{w} + \int_\Gamma \boldsymbol{h}\cdot\boldsymbol{w}, \forall \boldsymbol{w} \in V$$

数据是重力 $\boldsymbol{g}$ 以及边界压力 $\boldsymbol{h}$.

Example 9.29 (fluidStruct.edp) 在我们的例子中， *Lamé* 系统和 *Stokes* 系统由共同边界进行耦合， 流体应力使得边界产生位移因此改变了区域的形状，在这个区域内 *stokes* 进行了整合. 几何上，这是一个垂直的凹陷并有一个弹性的盖子。 这盖子是一个有重量 的光线， 它会被自身重量分解以及法向应力作用产生形变 凹陷是 10×10 的正方形, 盖子是高度 $l = 2$ 的矩形.

这光线是在一个装满旋转流体的箱子上, 这是因为左侧垂直出的速度为 *1*. 光线在自身重量下弯曲, 但流体的压强又抵消了这个弯曲.
光线的弯曲位移为 *(uu,vv)* ,它的解由以下给出.

```
//      在充满液体的方形凹陷上方有重量的光线的流体结构相互作用

int bottombeam = 2;                                          //    底部光线的标记
border a(t=2,0)  { x=0; y=t ;label=1;};                      //    左侧的光线
border b(t=0,10) { x=t; y=0 ;label=bottombeam;};             //    底部的光线
border c(t=0,2)  { x=10; y=t ;label=1;};                     //    右侧的光线
border d(t=0,10) { x=10-t; y=2; label=3;};                   //    顶部的光线
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
fespace Vh(th,P1);
Vh uu,w,vv,s,fluidforce=0;
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;
                                                             //    光线在其自身重力下的形变
solve  bb([uu,vv],[w,s])  =
    int2d(th)(
                  lambda*div(w,s)*div(uu,vv)
                  +2.*mu*( epsilon(w,s)'*epsilon(uu,vv) )
             )
  + int2d(th) (-gravity*s)
  + on(1,uu=0,vv=0)
  + fluidforce[];
 ;

 plot([uu,vv],wait=1);
 mesh th1 = movemesh(th, [x+uu, y+vv]);
 plot(th1,wait=1);
```

低速流体的 *stokes* 方程在光线下的盒子中求解，但是光线使盒子变形 *(见边界 h):*

```
                                //   在方形上区域 b,e,f,g 的 Stokes 方程受到左侧 g 的影响
border e(t=0,10) { x=t; y=-10; label= 1; };                                  //    底部
border f(t=0,10) { x=10; y=-10+t ; label= 1; };                              //    右侧
border g(t=0,10) { x=0; y=-t ;label= 2;};                                    //    左侧
border h(t=0,10) { x=t; y=vv(t,0)*( t>=0.001 )*(t <= 9.999);
                    label=3;};                                       //    顶部凹陷的变形

mesh sh = buildmesh(h(-20)+f(10)+e(10)+g(10));
plot(sh,wait=1);
```

我们利用 *Uzawa* 共轭梯度法来解决 *stokes* 问题就像在 *9.6.2* 的例子一样

```
fespace Xh(sh,P2),Mh(sh,P1);
Xh u1,u2,v1,v2;
Mh p,q,ppp;


varf bx(u1,q) = int2d(sh)( -(dx(u1)*q));

varf by(u1,q) = int2d(sh)( -(dy(u1)*q));

varf Lap(u1,u2)= int2d(sh)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) )
                    +  on(2,u1=1) +  on(1,3,u1=0)  ;

Xh bc1; bc1[] = Lap(0,Xh);
Xh brhs;

matrix A= Lap(Xh,Xh,solver=CG);
matrix Bx= bx(Xh,Mh);
matrix By= by(Xh,Mh);
Xh bcx=0,bcy=1;

func real[int] divup(real[int] & pp)
{
  int verb=verbosity;
   verbosity=0;
   brhs[]  = Bx'*pp; brhs[] += bc1[] .*bcx[];
   u1[] = A^-1*brhs[];
   brhs[]  = By'*pp; brhs[] += bc1[] .*bcy[];
   u2[] = A^-1*brhs[];
   ppp[] =   Bx*u1[];
   ppp[] +=  By*u2[];
   verbosity=verb;
   return ppp[] ;
};
```

在两个问题上做循环

```
for(step=0;step<2;++step)
 {
   p=0;q=0;u1=0;v1=0;

   LinearCG(divup,p[],eps=1.e-3,nbiter=50);
   divup(p[]);
```

现在光线会受到来自流体的压力:

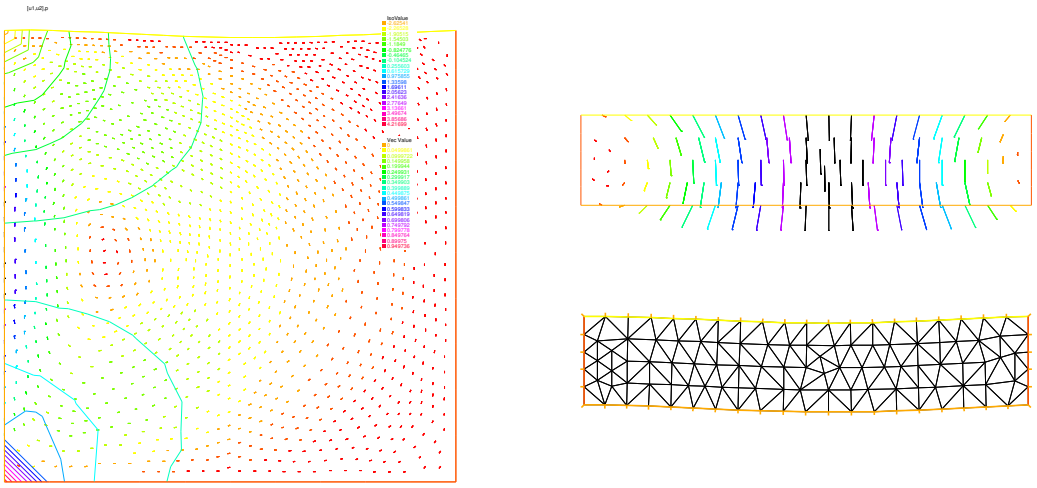


图 9.29: 流体的速度和压力 (左) 以及结构的位移向量 (中)位移几何 (右) 在软边界和被迫凹陷中的流体结构作用

```
Vh sigma11,sigma22,sigma12;
Vh uu1=uu,vv1=vv;

sigma11([x+uu,y+vv]) = (2*dx(u1)-p);
sigma22([x+uu,y+vv]) = (2*dy(u2)-p);
sigma12([x+uu,y+vv]) = (dx(u1)+dy(u2));
```

这就变成了有边界条件的光线 *PDE* 问题 :

```
solve  bbst([uu,vv],[w,s],init=i)  =
   int2d(th)(
                lambda*div(w,s)*div(uu,vv)
                +2.*mu*( epsilon(w,s)'*epsilon(uu,vv) )
            )
+ int2d(th) (-gravity*s)
+ int1d(th,bottombeam)( -coef*(   sigma11*N.x*w + sigma22*N.y*s
                                + sigma12*(N.y*w+N.x*s) )  )
+ on(1,uu=0,vv=0);
plot([uu,vv],wait=1);
real  err = sqrt(int2d(th)( (uu-uu1)^2 + (vv-vv1)^2 ));
cout <<  " Erreur L2 = " << err << "----------\n";
```

注意到 *bbst* 产生的矩阵被再次利用了 *(*见 *init=i).* 最终我们将光线变形

```
th1 = movemesh(th, [x+0.2*uu, y+0.2*vv]);
plot(th1,wait=1);
}                                                    //     循环结束
```

9.10 传输问题

考虑一个弹性板, 它的位移垂直改变. 它是由三个不同材质的板焊接而成. 令 Ω_i, $i = 1, 2, 3$ 是第 i个材料 的张量 μ_i 的定义域 (见 第9.1.1节). 计算的定义域 Ω 是 $\overline{\Omega_1} \cup \overline{\Omega_2} \cup \overline{\Omega_3}$ 的内部. 垂直位移 $u(x, y)$ 可以从如下得到

$$-\mu_i \Delta u = f \text{ in } \Omega_i \tag{9.55}$$

$$\mu_i \partial_n u|_{\Gamma_i} = -\mu_j \partial_n u|_{\Gamma_j} \quad \text{on } \overline{\Omega_i} \cap \overline{\Omega_j} \qquad \text{if } 1 \le i < j \le 3 \tag{9.56}$$

其中 $\partial_n u|_{\Gamma_i}$ 记为 在边界 Γ_i 上的法向导数 $\partial_n u$.
引入 Ω_i 上特征函数χ_i, 即,

$$\chi_i(x) = 1 \quad \text{if } x \in \Omega_i; \qquad \chi_i(x) = 0 \quad \text{if } x \notin \Omega_i \tag{9.57}$$

我们可以很容易重写 (9.55) 和 (9.56) 使之成为弱形式. 这里我们假定在 $\Gamma = \partial\Omega$ 上 $u = 0$.
传递问题: 给定一个外力 f, 找 u 使得

$$a(u, v) = \ell(f, v) \quad \text{for all } v \in H_0^1(\Omega) \tag{9.58}$$

$$a(u, v) = \int_\Omega \mu \nabla u \cdot \nabla v, \quad \ell(f, v) = \int_\Omega f v$$

这里 $\mu = \mu_1\chi_1 + \mu_2\chi_2 + \mu_3\chi_3$. 我们注意到 μ 是不连续函数.
在有消耗的情况下, 在热平衡点, 温度方程是:
这个例子解释了 区域 的定义和处理, 例如 整个区域的子域.
考虑这个 L-形区域，其中有三个对角线, 这样便定义了四个子区域:

```
                                                    //      例子是以区域为关键词
                                                //      建立四个区域的网格 (子区域)
border a(t=0,1){x=t;y=0;};
border b(t=0,0.5){x=1;y=t;};
border c(t=0,0.5){x=1-t;y=0.5;};
border d(t=0.5,1){x=0.5;y=t;};
border e(t=0.5,1){x=1-t;y=1;};
border f(t=0,1){x=0;y=1-t;};
                                                                    //      内部边界
border i1(t=0,0.5){x=t;y=1-t;};
border i2(t=0,0.5){x=t;y=t;};
border i3(t=0,0.5){x=1-t;y=t;};

mesh th = buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) +
    f(6)+i1(6)+i2(6)+i3(6));
fespace Ph(th,P0);                                          //      不连续函数 / 元素
fespace Vh(th,P1);                                          //      P1 连续函数 / 元素

Ph reg=region;                                          //      定义 P0 函数与区域号码有关
plot(reg,fill=1,wait=1,value=1);
```

region 是 FreeFem++ 的关键词，是关于当前位置的变量 (不是函数, 用 Ph reg=region; 来建立函数). 这个变量的值返回为当前子区域的位置. 这个数由 "buildmesh" 定义, 由它的连接成分建立网格. 所以下面得到包含某个特定点的区域数字:

```
int nupper=reg(0.4,0.9);                                //      得到(0.4,0.9)的区域数字
int nlower=reg(0.9,0.1);                            //      得到点 (0.4,0.1)的区域数字
```

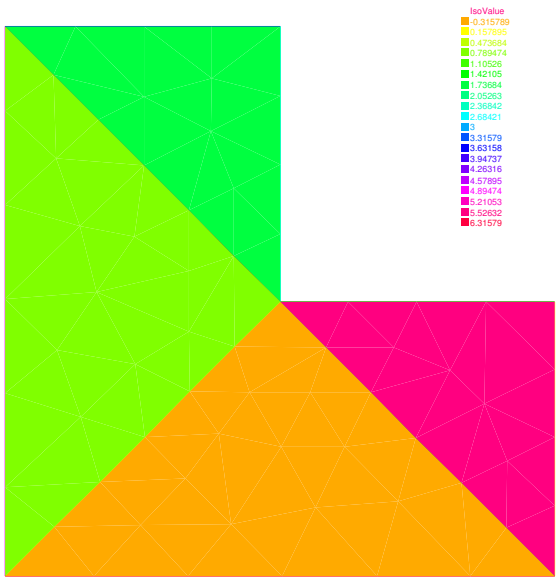


图 9.30: the function reg

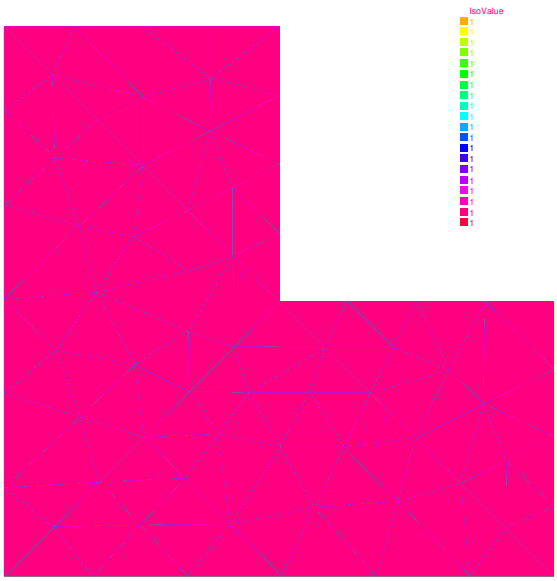


图 9.31: the function nu

```
cout << " nlower " <<  nlower << ", nupper = " << nupper<< endl;
                                                    //     定义上下两个区域的特征方程
Ph nu=1+5*(region==nlower) + 10*(region==nupper);
plot(nu,fill=1,wait=1);
```

定义 函数是非常有用的，因为通常会出现一边是铜，一遍是铁的情况.
有了这个之后， 我们可以来求解具有不连续常数的 Laplace 方程 (ν 是下面的 1, 6 和 11).

```
Ph nu=1+5*(region==nlower) + 10*(region==nupper);
plot(nu,fill=1,wait=1);
problem lap(u,v) =   int2d(th)( nu*( dx(u)*dx(v)*dy(u)*dy(v) ))
                   + int2d(-1*v) + on(a,b,c,d,e,f,u=0);
plot(u);
```

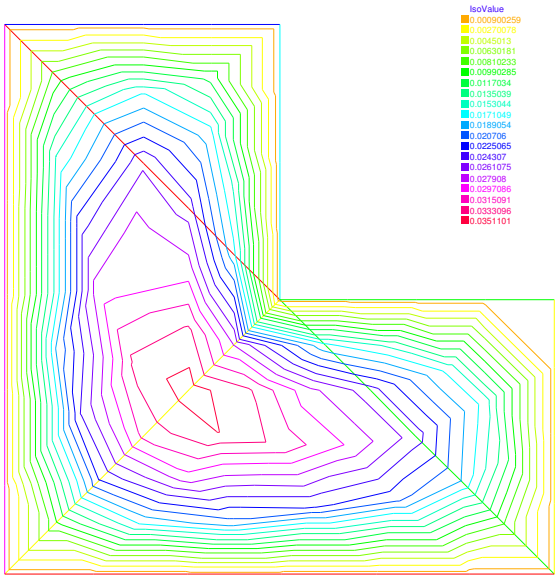


图 9.32: 解 u 的等值线

9.11 自由边界问题

区域 Ω 定义如下:

```
real L=10;                                                    //   区域的长度
real h=2.1;                                                   //   左边缘高度
real h1=0.35;                                                 //   直边的高度

                                                              //   网格
border a(t=0,L){x=t;y=0;};                                    //   下: Γ_a
border b(t=0,h1){x=L;y=t;};                                   //   右: Γ_b
border f(t=L,0){x=t;y=t*(h1-h)/L+h;};                         //   free surface: Γ_f
border d(t=h,0){x=0;y=t;};                                    //   左: Γ_d

int n=4;
mesh Th=buildmesh (a(10*n)+b(6*n)+f(8*n)+d(3*n));
plot(Th,ps="dTh.eps");
```

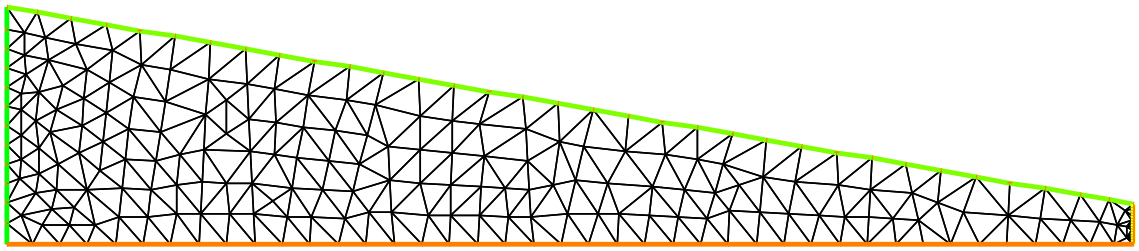


图 9.33: 区域 Ω 的网格

自由边界问题是:
找 u 和 Ω 使得:

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega \\ u = y & \text{on } \Gamma_b \\ \dfrac{\partial u}{\partial n} = 0 & \text{on } \Gamma_d \cup \Gamma_a \\ \dfrac{\partial u}{\partial n} = \dfrac{q}{K} n_x \text{ and } u = y & \text{on } \Gamma_f \end{cases}$$

我们利用固定点方法; $\Omega^0 = \Omega$
在两步中, 我们解决下面这个经典问题:

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega^n \\ u = y & \text{on } \Gamma_b^n \\ \dfrac{\partial u}{\partial n} = 0 & \text{on } \Gamma_d^n \cup \Gamma_a^n \\ u = y & \text{on } \Gamma_f^n \end{cases}$$

变分形式是:
找 u 在 $V = H^1(\Omega^n)$ 上, 使得 $u = y$ 在 Γ_b^n 以及 Γ_f^n

$$\int_{\Omega^n} \nabla u \nabla u' = 0, \quad \forall u' \in V \text{ with } u' = 0 \text{ on } \Gamma_b^n \cup \Gamma_f^n$$

第二步,构造形变区域 $\mathcal{F}(x, y) = [x, y - v(x, y)]$
这里 v 是下面问题的解:

$$\begin{cases} -\Delta v = 0 & \text{in } \Omega^n \\ v = 0 & \text{on } \Gamma_a^n \\ \dfrac{\partial v}{\partial n} = 0 & \text{on } \Gamma_b^n \cup \Gamma_d^n \\ \dfrac{\partial v}{\partial n} = \dfrac{\partial u}{\partial n} - \dfrac{q}{K} n_x & \text{on } \Gamma_f^n \end{cases}$$

变分形式为:
找 v on V, 使得 $v = 0$ 在 Γ_a^n

$$\int_{\Omega^n} \nabla v \nabla v' = \int_{\Gamma_f^n} (\frac{\partial u}{\partial n} - \frac{q}{K} n_x) v', \quad \forall v' \in V \text{ with } v' = 0 \text{ on } \Gamma_a^n$$

最后新区域是 $\Omega^{n+1} = \mathcal{F}(\Omega^n)$

Example 9.30 (freeboundary.edp) *FreeFem++* :实现过程是:

```
real q=0.02;                                                    //   流出流量
real K=0.5;                                                     //   渗透常数

fespace Vh(Th,P1);
int j=0;

Vh u,v,uu,vv;

problem Pu(u,uu,solver=CG) = int2d(Th)( dx(u)*dx(uu)+dy(u)*dy(uu))
  + on(b,f,u=y) ;

problem Pv(v,vv,solver=CG) = int2d(Th)( dx(v)*dx(vv)+dy(v)*dy(vv))
  +  on (a, v=0) + int1d(Th,f)(vv*((q/K)*N.y- (dx(u)*N.x+dy(u)*N.y)));
```

```
real errv=1;
real erradap=0.001;
verbosity=1;
while(errv>1e-6)
{
  j++;
  Pu;
  Pv;
  plot(Th,u,v ,wait=0);
  errv=int1d(Th,f)(v*v);
   real coef=1;

                                                                    //

  real mintcc = checkmovemesh(Th,[x,y])/5.;
  real mint = checkmovemesh(Th,[x,y-v*coef]);

  if (mint<mintcc ||  j%10==0) {                          //    网格太差，重新绘制
    Th=adaptmesh(Th,u,err=erradap ) ;
    mintcc = checkmovemesh(Th,[x,y])/5.;
  }

  while (1)
  {
    real mint = checkmovemesh(Th,[x,y-v*coef]);

    if (mint>mintcc) break;

    cout << " min |T]  " << mint << endl;
    coef /= 1.5;
  }

  Th=movemesh(Th,[x,y-coef*v]);                                  //    计算形变
  cout << "\n\n"<<j <<"------------ errv = " << errv << "\n\n";

}
plot(Th,ps="d_Thf.eps");
plot(u,wait=1,ps="d_u.eps");
```

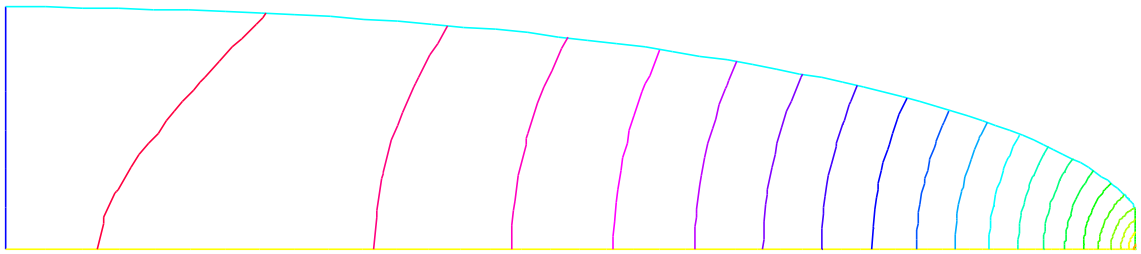


图 9.34: 在新区域 Ω^{72} 的最终解

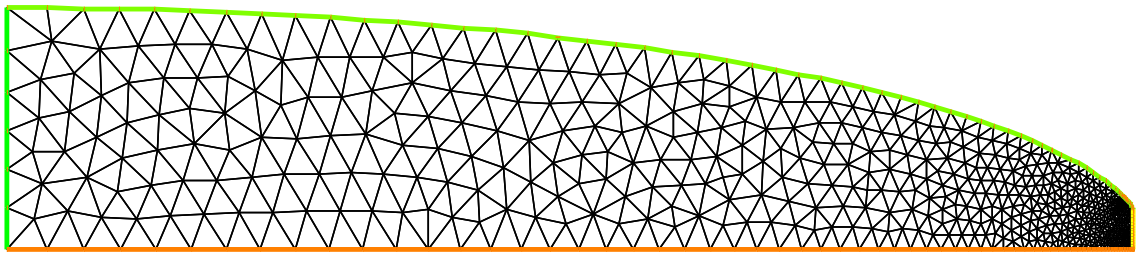


图 9.35: 区域 Ω^{72}上的自适应网格

9.12 非线性弹性力学(nolinear-elas.edp)

非线性弹性问题是: 找位移 (u_1, u_2) 最小化问题 J

$$\min J(u_1,u_2) = \int_\Omega f(F2) - \int_{\Gamma_p} P_a\, u_2$$

其中 $F2(u_1,u_2) = A(E[u_1,u_2], E[u_1,u_2])$ 和 $A(X,Y)$ 是双线性性. 关于矩阵 X, Y正定. 这里f 是给定 $\mathcal{C}^2$ 函数, 以及 $E[u_1,u_2] = (E_{ij})_{i=1,2,\,j=1,2}$ 是 Green-Saint Venant 形变张量, 定义为:

$$E_{ij} = 0.5\big((\partial_i u_j + \partial_j u_i) + \sum_k \partial_i u_k \times \partial_j u_k\big)$$

记 $\mathbf{u} = (u_1, u_2)$, $\mathbf{v} = (v_1, v_2)$, $\mathbf{w} = (w_1, w_2)$.
所以J 的微分是

$$DJ(\boldsymbol{u})(\boldsymbol{v}) = \int DF2(\boldsymbol{u})(\boldsymbol{v})\; f'(F2(\boldsymbol{u}))) - \int_{\Gamma_p} P_a v_2$$

其中$DF2(\boldsymbol{u})(\boldsymbol{v}) = 2\; A(\; DE[\mathbf{u}](\mathbf{v})\;,\; E[\mathbf{u}]\;)$ ， DE是E的一阶微分。
J 的二阶微分是

$$\begin{aligned} D^2J(\mathbf{u})((\mathbf{v}),(\mathbf{w})) \;&=\; \int DF2(\boldsymbol{u})(\boldsymbol{v})\; DF2(\boldsymbol{u})(\boldsymbol{w})\; f''(F2(\mathbf{u}))) \\ &+\; \int D^2F2(\boldsymbol{u})(\boldsymbol{v},\boldsymbol{w})\; f'(F2(\mathbf{u}))) \end{aligned}$$

其中

$$D^2F2(\boldsymbol{u})(\boldsymbol{v},\boldsymbol{w}) = 2\; A(\; D^2E[\mathbf{u}](\boldsymbol{v},\boldsymbol{w})\;,\; E[\boldsymbol{u}]\;) + 2\; A(\; DE[\boldsymbol{u}](\boldsymbol{v})\;, DE[\boldsymbol{u}](\boldsymbol{w})\;).$$

和 D^2E 是E的二阶微分。

所以所有符号可以用 macros来定义:

```
macro EL(u,v) [dx(u),(dx(v)+dy(u)),dy(v)]                    //     是 [ϵ11, 2ϵ12, ϵ22]

macro ENL(u,v) [
(dx(u)*dx(u)+dx(v)*dx(v))*0.5,
(dx(u)*dy(u)+dx(v)*dy(v))     ,
(dy(u)*dy(u)+dy(v)*dy(v))*0.5 ]                                //     EOM ENL

macro dENL(u,v,uu,vv) [(dx(u)*dx(uu)+dx(v)*dx(vv)),
 (dx(u)*dy(uu)+dx(v)*dy(vv)+dx(uu)*dy(u)+dx(vv)*dy(v)),
 (dy(u)*dy(uu)+dy(v)*dy(vv)) ]                                  //
```

```
macro E(u,v) (EL(u,v)+ENL(u,v)) //      是 [E_11, 2E_12, E_22]
macro dE(u,v,uu,vv) (EL(uu,vv)+dENL(u,v,uu,vv)) //
macro ddE(u,v,uu,vv,uuu,vvv) dENL(uuu,vvv,uu,vv) //
macro F2(u,v) (E(u,v)'*A*E(u,v)) //
macro dF2(u,v,uu,vv)  (E(u,v)'*A*dE(u,v,uu,vv)*2. ) //
macro ddF2(u,v,uu,vv,uuu,vvv) (
            (dE(u,v,uu,vv)'*A*dE(u,v,uuu,vvv))*2.
          + (E(u,v)'*A*ddE(u,v,uu,vv,uuu,vvv))*2.  ) //     EOM
```

牛顿法如下：
令 $n = 0$, u_O, v_O 为初始位移

- 循环:
- 找到 (du, dv) 是

$$D^2J(u_n, v_n)((w, s), (du, dv)) = DJ(u_n, v_n)(w, s), \quad \forall w, s$$

  的解

- $un = un - du, \quad vn = vn - dv$
- 直到(du, dv) 足够小

在 FreeFem++ 中实现此算法的方法是使用macro 工具来实现A 和 $F2$, f, f',f''.
macro 就像 C++ 预处理器, 但它是以`macro` 开始且macro的结尾是在评论 //之前. 此例中，macro很有用，因为参数的类型可以改变。进行自动微分也十分容易。

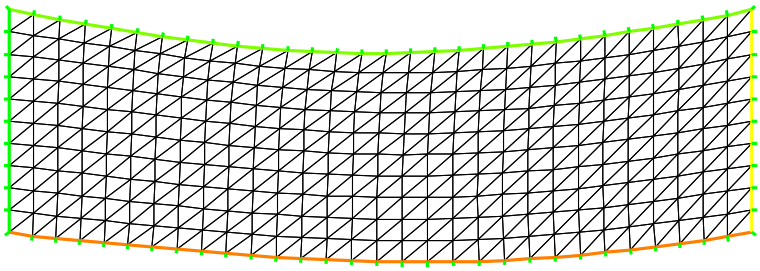


图 9.36: 变形域

```
//     非线性弹性模型
//     超弹性问题
//     ------------------------------
macro f(u) (u) //     宏结束
macro df(u) (1) //     宏结束
macro ddf(u) (0) //     宏结束

//     -- du caouchouc --- (见Herve Le Dret.的说明)
//     ------------------------------
real mu = 0.012e5; //     kg/cm^2
real lambda =  0.4e5; //     kg/cm^2
//
```

```
//     σ = 2μE + λtr(E)Id
//     A(u,v) = σ(u) : E(v)
//
//     ( a  b )
//     ( b  c )
//
//     tr*Id :  (a,b,c) -> (a+c,0,a+c)
//     所以关联矩阵是:
//     ( 1  0  1 )
//     ( 0  0  0 )
//     ( 1  0  1 )
                                                          //     ------------------v
real a11= 2*mu +  lambda  ;
real a22= mu ;                                                //     因为 [0,2*t12,0]'A[0,2*s12,0] =
                                                    //     = 2*mu*(t12*s12+t21*s21) = 4*mu*t12*s12
real a33= 2*mu +   lambda ;
real a12= 0 ;
real a13= lambda ;
real a23= 0 ;
                                                                                    //     对称部分
real a21= a12 ;
real a31= a13 ;
real a32= a23 ;

                                                                                    //     矩阵 A.
func A = [ [ a11,a12,a13],[ a21,a22,a23],[ a31,a32,a33] ];

real Pa=1e2;                                                                //     100 Pa 的压强
//    ----------------

int n=30,m=10;
mesh Th= square(n,m,[x,.3*y]);                          //     标签:  1 下, 2 右, 3 上, 4 左;
int bottom=1, right=2,upper=3,left=4;

plot(Th);



fespace Wh(Th,P1dc);
fespace Vh(Th,[P1,P1]);
fespace Sh(Th,P1);




Wh e2,fe2,dfe2,ddfe2;                                                       //     最优化
Wh ett,ezz,err,erz;                                                         //     最优化

Vh [uu,vv], [w,s],[un,vn];
[un,vn]=[0,0];                                                              //     初始化
[uu,vv]=[0,0];

varf vmass([uu,vv],[w,s],solver=CG) =  int2d(Th)( uu*w + vv*s );
matrix M=vmass(Vh,Vh);
problem NonLin([uu,vv],[w,s],solver=LU)=
```

```
 int2d(Th,qforder=1)(                                          //    (D^2 J(un)) 部分
                        dF2(un,vn,uu,vv)*dF2(un,vn,w,s)*ddfe2
                    +   ddF2(un,vn,w,s,uu,vv)*dfe2
             )
   - int1d(Th,3)(Pa*s)
   - int2d(Th,qforder=1)(                                      //    (DJ(un)) 部分
           dF2(un,vn,w,s)*dfe2    )
   + on(right,left,uu=0,vv=0);
;
                                                                //    Newton 法
                                                              //    ---------------
Sh u1,v1;
for (int i=0;i<10;i++)
{
  cout << "Loop " << i << endl;
  e2 = F2(un,vn);
  dfe2 = df(e2) ;
  ddfe2 = ddf(e2);
  cout << "  e2 max " <<e2[].max << " , min" << e2[].min << endl;
  cout << " de2 max "<< dfe2[].max << " , min" << dfe2[].min << endl;
  cout << "dde2 max "<< ddfe2[].max << " , min" << ddfe2[].min << endl;
  NonLin;                                                   //    计算 [uu, vv] = (D^2 J(un))^{-1} (DJ(un))

  w[]   = M*uu[];
  real res = sqrt(w[]' * uu[]);                             //    [uu, vv] 的 L^2 范数
  u1 = uu;
  v1 = vv;
  cout << " L^2 residual = " << res << endl;
  cout << " u1 min =" <<u1[].min << ", u1 max= " << u1[].max << endl;
  cout << " v1 min =" <<v1[].min << ", v2 max= " << v1[].max << endl;
  plot([uu,vv],wait=1,cmm=" uu, vv " );
  un[] -= uu[];
  plot([un,vn],wait=1,cmm=" displacement " );
  if (res<1e-5) break;
}

plot([un,vn],wait=1);
mesh th1 = movemesh(Th, [x+un, y+vn]);
plot(th1,wait=1);                                           //    见图 9.36
```

9.13 可压缩的 Neo-Hookean 材料:计算解

作者: Alex Sadovsky mailsashas@gmail.com

9.13.1 记号

接下来，符号 $\boldsymbol{u}, \mathbf{F}, \mathbf{B}, \mathbf{C}, \underline{\sigma}$分别表示 位移场, 变形梯度, 左 Cauchy-Green 应变张量 $\mathbf{B} = \mathbf{F}\mathbf{F}^T$, 右 Cauchy-Green 应变张量 $\mathbf{C} = \mathbf{F}^T\mathbf{F}$, 以及 Cauchy 应力张量。 我们也引进符号 $I_1 := \mathbf{tr}\ \mathbf{C}$ 和 $J := \det \mathbf{F}$. 下面等式将会用到。

$$\frac{\partial J}{\partial \mathbf{C}} = J\mathbf{C}^{-1} \tag{9.59}$$

符号 $\mathbf{I}$ 表示等价张量. 符号 Ω_0表示变形前主体的参考结构。 参考结构中单位体积记成 dV (resp., dV_0); 两者的关系如下

$$dV = JdV_0,$$

这使得在Ω上关于Cauchy应力$\mathbf{T}$ 的积分可以重写成在Ω_0上 Kirchhoff 应力 $\kappa = J\mathbf{T}$的积分 。

推荐参考资料

关于非线性弹力和根本线性张量代数的阐述，可以参考[34]。 关于有限元方法的高等数学分析，可以参考[35]。关于非线性弹性静电的边界值问题的有限元方程式的解释，参考http://www.engin.brown.edu/courses/en222/Notes/FEMfinitestrain/FEMfinitestrain.htm.

9.13.2 一种 Neo-Hookean 可压缩材料

基本理论和切向应力法 应变能密度函数如下定义：

$$W = \frac{\mu}{2}(I_1 - \mathbf{tr}\ \mathbf{I} - 2\ln J) \tag{9.60}$$

(参考 [32], 公式 (12)).
相应的第二 Piola-Kirchoff 应力张量由下式推出：

$$\mathbf{S}_n := \frac{\partial W}{\partial \mathbf{E}}(\mathbf{F}_n) = \mu(\mathbf{I} - \mathbf{C}^{-1}) \tag{9.61}$$

Kirchhoff 应力变为

$$\kappa = \mathbf{F}\mathbf{S}\mathbf{F}^T = \mu(\mathbf{B} - \mathbf{I}) \tag{9.62}$$

相应地，在$\mathbf{F}_n$ 处作用在 $\delta\mathbf{F}_{n+1}$的切向Kirchhoff 应力张量变为

$$\frac{\partial \kappa}{\partial \mathbf{F}}(\mathbf{F}_n)\delta\mathbf{F}_{n+1} = \mu\left[\mathbf{F}_n(\delta\mathbf{F}_{n+1})^T + \delta\mathbf{F}_{n+1}(\mathbf{F}_n)^T\right] \tag{9.63}$$

在缺少外在体力下BVP 的弱形式 此处我们考虑Ω_0 是一个椭圆环,其边界是由两个同中心且主轴相互平行的椭圆围成 （作为特例，椭圆也可以是圆）。令P 在边界$\partial\Omega_0$ 上的部分边界$\partial\Omega_0^t$（内椭圆）的负载力（牵引力）。 在剩余的边界上，我们规定用0 取代。
边界值问题的弱形式如下

$$\left.\begin{array}{rcl} 0 & = & \int_{\Omega_0} \kappa[\mathbf{F}] : \left\{(\boldsymbol{\nabla} \otimes \boldsymbol{w})(\mathbf{F})^{-1}\right\} \\ & - & \int_{\partial\Omega_0^t} P \cdot \hat{N}_0 \end{array}\right\}$$

为了简化问题，在剩余章节我们不妨假定 $P = 0$。但是提供的 *FreeFem++* 代码不依赖于这个假设，会考虑一般值和P的方向。
给定一个满足BVP的位移场$\boldsymbol{u}_n$ 的牛顿近似值$\boldsymbol{u}$ 。我们找修正值 $\delta\boldsymbol{u}_{n+1}$以获得更好的近似值。

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \delta\boldsymbol{u}_{n+1}$$

通过求解弱形式

$$\left.\begin{array}{rcl} 0 &=& \int_{\Omega_0} \kappa[\mathbf{F}_n+\delta\mathbf{F}_{n+1}] \,:\, \left\{(\boldsymbol{\nabla}\otimes\boldsymbol{w})(\mathbf{F}_n+\delta\mathbf{F}_{n+1})^{-1}\right\} - \int_{\partial\Omega_0} P\cdot\hat{N}_0 \\ &=& \int_{\Omega_0} \left\{\kappa[\mathbf{F}_n]+\frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1}\right\} \,:\, \left\{(\boldsymbol{\nabla}\otimes\boldsymbol{w})(\mathbf{F}_n+\delta\mathbf{F}_{n+1})^{-1}\right\} \\ &=& \int_{\Omega_0} \left\{\kappa[\mathbf{F}_n]+\frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1}\right\} \,:\, \left\{(\boldsymbol{\nabla}\otimes\boldsymbol{w})(\mathbf{F}_n^{-1}+\mathbf{F}_n^{-2}\delta\mathbf{F}_{n+1})\right\} \\ &=& \int_{\Omega_0} \kappa[\mathbf{F}_n] \,:\, \left\{(\boldsymbol{\nabla}\otimes\boldsymbol{w})\mathbf{F}_n^{-1}\right\} \\ &-& \int_{\Omega_0} \kappa[\mathbf{F}_n] \,:\, \left\{(\boldsymbol{\nabla}\otimes\boldsymbol{w})(\mathbf{F}_n^{-2}\delta\mathbf{F}_{n+1})\right\} \\ &+& \int_{\Omega_0} \left\{\frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1}\right\} \,:\, \left\{(\boldsymbol{\nabla}\otimes\boldsymbol{w})\mathbf{F}_n^{-1}\right\} \end{array}\right\} \quad \text{对于所有检验函数}\boldsymbol{w}, \tag{9.64}$$

其中，我们取

$$\delta\mathbf{F}_{n+1} = \boldsymbol{\nabla}\otimes\delta\boldsymbol{u}_{n+1}$$

注: 与标准的符号使用相反, 符号 δ 没有变分形式背景。用 δ 仅表示牛顿法意义下的增量。此处的变分虚位移用$\boldsymbol{w}$ 表示。

9.13.3 在 FreeFem++中的一种实现方式

相关的文件是`examples++-tutorial/nl-elast-neo-Hookean.edp`.
引进和代码一样的符号，其中<>中的一个字符看作是一个符号，张量的独立部分

$$< TanK >:= \frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1} \tag{9.65}$$

作为macros $< TanK11 >, < TanK12 >, \ldots$被执行。
张量的独立部分如下：

$$\mathbf{D}_1 := \mathbf{F}_n(\delta\mathbf{F}_{n+1})^T + \delta\mathbf{F}_{n+1}(\mathbf{F}_n)^T,$$
$$\mathbf{D}_2 := \mathbf{F}_n^{-T}\delta\mathbf{F}_{n+1},$$
$$\mathbf{D}_3 := (\boldsymbol{\nabla}\otimes\boldsymbol{w})\mathbf{F}_n^{-2}\delta\mathbf{F}_{n+1},$$

和

$$\mathbf{D}_4 := (\boldsymbol{\nabla}\otimes\boldsymbol{w})\mathbf{F}_n^{-1},$$

分别作为macros被执行

$$\left.\begin{array}{llll} < d1Aux11 >, < d1Aux12 >, & \ldots & , < d1Aux22 >, \\ < d2Aux11 >, < d2Aux12 >, & \ldots & , < d2Aux22 > \\ < d3Aux11 >, < d3Aux12 >, & \ldots & , < d3Aux22 > \\ < d4Aux11 >, < d4Aux12 >, & \ldots & , < d4Aux22 > \end{array}\right\}, \tag{9.66}$$

利用上边的符号，切向Kirchhoff应力项变成

$$\frac{\partial\kappa}{\partial\mathbf{F}}(\mathbf{F}_n)\,\delta\mathbf{F}_{n+1} = \mu\,\mathbf{D}_1 \tag{9.67}$$

同时BVP弱解变成如下形式

$$\left.\begin{array}{rcl} 0 &=& \int_{\Omega_0} \kappa[\mathbf{F}_n] \,:\, \mathbf{D}_4 \\ &-& \int_{\Omega_0} \kappa[\mathbf{F}_n] \,:\, \mathbf{D}_3 \\ &+& \int_{\Omega_0} \left\{\frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1}\right\} \,:\, \mathbf{D}_4 \end{array}\right\} \quad \text{对于所有检验函数}\boldsymbol{w} \tag{9.68}$$

第 10 章

MPI 并行版本

FreeFem++的第一次尝试用 **mpi**并行化。 关于MPI的扩展界已经被加入到 FreeFem++ 的 3.5版, (关于MPI语言功能的文档见http://www.mpi-forum.org/docs/mpi21-report.pdf).

10.1 MPI 关键字

下面的关键字和概念会用到:

mpiComm 定义一个通信世界

mpiGroup 定义一组通信世界的 处理器

mpiRequest 定义一组等待结束通讯的请求

10.2 MPI 常数

mpisize 处理器 的总数,

mpirank 在 $\{0, ..., mpisize - 1\}$ 中当前处理器的识别号码,

mpiUndefined MPI_Undefined 常数,

mpiAnySource MPI_ANY_SOURCE 常数,

mpiCommWorld MPI_COMM_WORLD 常数,

... 化简运算符MPI_Op的所有关键字: mpiMAX, mpiMIN, mpiSUM, mpiPROD, mpiLAND, mpiLOR, mpiLXOR, mpiBAND, mpiBXOR.

10.3 MPI 构造器

```
int[int] proc1=[1,2,3],proc2=[0,4];
mpiGroup grp(procs);     //    在 MPI_COMM_WORLD 中设定 MPI_Group 为 proc 1,2,3
mpiGroup grp1(comm,proc1);       //    在 comm 中设定 MPI_Group 为 proc 1,2,3
```

```
mpiGroup grp2(grp,proc2);                    //    设定 MPI_Group 为 grp union proc1

mpiComm  comm=mpiCommWorld;                  //    设定一个 MPI_Comm 为 MPI_COMM_WORLD
mpiComm  ncomm(mpiCommWorld,grp);                  //    设定 MPI_Comm 形式为 grp
                                                                //    MPI_COMM_WORLD
mpiComm  ncomm(comm,color,key);                //    MPI_Comm_split(MPI_Comm comm,
                                          //     int color, int key, MPI_Comm *ncomm)
mpiComm  nicomm(processor(local_comm,local_leader),
                processor(peer_comm,peer_leader),tag);
       //    建立  MPI_INTERCOMM_CREATE(local_comm, local_leader, peer_comm,
                                                //     remote_leader, tag, &nicomm)
mpiComm  ncomm(intercomm,hight) ;                                    //    建立使用
                        //    MPI_Intercomm_merge( intercomm, high, &ncomm)
mpiRequest rq;                                                 //    定义一个 MPI_Request
mpiRequest[int] arq(10);                 //    定义一个包含 10 个 MPI_Request 的数组
```

10.4 MPI 函数

```
mpiSize(comm) ;                                          //    返回 comm (int)的大小
mpiRank(comm) ;                                    //    返回在 comm (int) 中的排序

processor(i)                             //    返回在MPI_COMM_WORLD 中没有请求的处理器 i
processor(mpiAnySource)                    //    返回在MPI_COMM_WORLD 中没有请求的处理器
                                                          //    该处理器是任何源的
processor(i,comm)                                  //    返回在 comm 中没有请求的处理器 i
processor(comm,i)                                  //    返回在 comm 中没有请求的处理器 i
processor(i,rq,comm)                             //    返回在 comm 中有请求 rq 的处理器 i
processor(i,rq)                        //    返回在MPI_COMM_WORLD 中有请求 rq 的处理器 i
processorblock(i)                                          //    在同步通讯的分块模式下
                                                  //    返回在MPI_COMM_WORLD 中处理器 i
processorblock(mpiAnySource)                                //    在同步通讯的分块模式下
                                                //    返回在MPI_COMM_WORLD 中任何源的处理器
processorblock(i,comm)                             //    分块模式下返回在 comm中的处理器 i

mpiBarrier(comm) ;                                 //    在通讯器comm上进行 MPI_Barrier,
mpiWait(rq);                                                          //    等待请求,
mpiWaitAll(arq);                                                //    等待增加请求数组,
mpiWtime() ;                                       //    返回 MPIWtime , 以秒计 (实数),
mpiWtick() ;                                       //    返回 MPIWTick , 以秒计 (实数),
```

其中处理器 就是一个整数排序, MPI_comm的指针，MPI_Request的指针和有特殊MPI_Request处理器区域。

10.5 MPI 通讯器算符

```
int status;                                                //    获取 MPI 发送/接受 的状态
processor(10) << a << b;                             //    将a,b~不同步地发送给处理器10,
processor(10) >> a >> b;                                 //    从处理器10处同步接受 a,b,
broadcast(processor(10,comm),a);                 //    从comm的处理器向comm的其他处理器广播
status=Send( processor(10,comm) , a);                    //    将数据a向处理器10同步发送
status=Recv( processor(10,comm) , a);                  //    从处理器10处同步接受数据a;
```

```
   status=Isend( processor(10,comm) , a);        //    将数据a不同步地发送至处理器10
                                                 //      没有请求
   status=Isend( processor(10,rq,comm) , a) ;    //    将数据a不同步地发送至处理器10
                                                 //      有请求
   status=Irecv( processor(10,rq) , a) ;         //    从处理器10处同步接受数据a;
    status=Irecv( processor(10) , a) ;                   //      错误
                                                 //      不同步，无请求.
broadcast(processor(comm,a));                    //    向comm的所有处理器广播
```

其中数据类型如下：int,real, complex, int[int], double[int], complex[int], int[int,int], double[int,int], complex[int,int], mesh, mesh3, mesh[int], mesh3[int], matrix, matrix<complex>

```
    processor(10,rq) << a ;                      //    向处理器10不同步地发送数据a,
                                                 //      有请求.
    processor(10,rq) >> a ;                      //    从处理器10不同步地接受数据a,
                                                 //      无请求.
```

如果 a,b 是整数域、实数域、复数域上的数组或者满秩矩阵，使用下面的 MPI 函数:

```
    mpiAlltoall(a,b[,comm]) ;
    mpiAllgather(a,b[,comm]) ;
    mpiGather(a,b,processor(..) ) ;
    mpiScatter(a,b,processor(..)) ;
    mpiReduce(a,b,processor(..),mpiMAX) ;
    mpiAllReduce(a,b,comm, mpiMAX) ;
    mpiReduceScatter(a,b,comm, mpiMAX) ;
```

见examples++-mpi/essai.edp 以检验所有的泛函性，在Antoine Atenekeng Kahou的帮助下，编码出此界面。

10.6 并行的Schwarz例子

这个例子是章节9.8.1. schwarz重叠问题的改写。

```
[examples++-mpi] Hecht%lamboot
LAM 6.5.9/MPI 2 C++/ROMIO - Indiana University
[examples++-mpi] hecht% mpirun -np 2 FreeFem++-mpi schwarz-c.edp


//    代码的新版本, 并行的schwarz方法
//    2个处理器.
//    ------------------------------
//    F.Hecht 2003年12月
//    ---------------------------------
//    测试广播指令,
//    网格的数组,
//    并增加了停机测试.
//    ---------------------------------

if ( mpisize != 2 ) {
    cout << " sorry, number of processors !=2 " << endl;
     exit(1);}
verbosity=3;
int interior = 2;
```

```
int exterior = 1;
border a(t=1,2){x=t;y=0;label=exterior;};
border b(t=0,1){x=2;y=t;label=exterior;};
border c(t=2,0){x=t ;y=1;label=exterior;};
border d(t=1,0){x = 1-t; y = t;label=interior;};
border e(t=0, pi/2){ x= cos(t); y = sin(t);label=interior;};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=exterior;};
int n=4;
mesh[int]  Th(mpisize);
if (mpirank == 0)
 Th[0] = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
else
 Th[1] = buildmesh ( e(5*n) + e1(25*n) );

broadcast(processor(0),Th[0]);
broadcast(processor(1),Th[1]);

fespace Vh(Th[mpirank],P1);
fespace Vhother(Th[1-mpirank],P1);

Vh u=0,v;
Vhother U=0;
int i=0;

problem pb(u,v,init=i,solver=Cholesky) =
    int2d(Th[mpirank])( dx(u)*dx(v)+dy(u)*dy(v) )
  - int2d(Th[mpirank])( v)
  + on(interior,u = U)  +  on(exterior,u= 0 ) ;

for ( i=0 ;i< 20; i++)
{
  cout << mpirank << " looP " << i << endl;
   pb;
                                          //    将 u 发送至另一个处理器, 在 U 中接收
   processor(1-mpirank) << u[];   processor(1-mpirank) >> U[];
   real err0,err1;
   err0 = int1d(Th[mpirank],interior)(square(U-u)) ;
                                        //    将 err0 发送至另一个处理器, 在 err1 中接收
   processor(1-mpirank)<<err0;   processor(1-mpirank)>>err1;
   real err= sqrt(err0+err1);
   cout <<" err = " << err << " err0 = " << err0
        << ", err1 = " << err1 << endl;
   if(err<1e-3) break;
};
if (mpirank==0)
    plot(u,U,ps="uU.eps");
```

10.6.1 真正并行Schwarz例子

这是两个脚本程序 examples++-mpi/MPIGMRES[2]D.edp的解释,一个并行Schwarz的例子，其复杂度不依赖于过程的数量（有粗网格的预条件）。

感谢F. Nataf.

为了解决以下$L^2(\Omega)$中在区域Ωw边界Γ上 in $L^2(\Omega)$ 的泊松问题:

$$-\Delta u = f, \text{ in } \Omega, \text{ and } u = g \text{ on } \Gamma,$$

其中f和g分别是在 $L^2(\Omega)$ 和 $H^{\frac{1}{2}}(\Gamma)$上的给定函数，
下面介绍$(\pi_i)_{i=1,..,N_p}$是区域 Ω的一个常规划分， q-e-d:

$$\pi_i \in \mathcal{C}^0(\Omega): \quad \pi_i \geq 0 \text{ and } \sum_{i=1}^{N_p} \pi_i = 1.$$

Ω_i表示函数π_i的支集构成的子区域，并且 Γ_i 表示Ω_i的边界。
并行Schwarz方法就是令$\ell = 0$，迭代过程和初始条件u^0代表边界条件(i.e. $u^0_{|\Gamma} = g$).

$$\forall i = 1.., N_p: \quad -\Delta u_i^\ell = f, \text{ in } \Omega_i, \quad \text{and } u_i^\ell = u^\ell \text{ on } \Gamma_i \setminus \Gamma, \; u_i^\ell = g \text{ on } \Gamma_i \cap \Gamma \tag{10.1}$$

$$u^{\ell+1} = \textstyle\sum_{i=1}^{N_p} \pi_i u_i^\ell \tag{10.2}$$

利用离散Lagrange有限元方法，和可并行的网格Ω_i中的$\mathcal{T}_{hi}$， i. e., 存在全局网格$\mathcal{T}_h$使得$\mathcal{T}_{hi}$ 包含于$\mathcal{T}_h$。 我们指明:

- V_{hi} 是对应于区域Ω_i的有限元空间，
- $\mathcal{N}_{hi}$ 是自由度σ_i^k的集合，
- $\mathcal{N}_{hi}^{\Gamma_i}$ 是V_{hi} 在边界Ω_i中Γ_i 的自由度的集合，
- $\sigma_i^k(v_h)$是自由度k的值，
- $V_{0hi} = \{v_h \in V_{hi} : \forall k \in \mathcal{N}_{hi}^{\Gamma_i}, \quad \sigma_i^k(v_h) = 0\}$,
- 条件表达式$a\,?\,b : c$ 与 C和C++语言定义的一样，即

$$a?b:c \equiv \begin{cases} \text{如果}a\text{正确则返回}b \\ \text{否则返回}c \end{cases}.$$

Remark 我们从不使用整个区域 Ω的有限元空间因为代价太高。

我们必须定义算子来建立先前的算法:
我们用$u_{h|i}^\ell$表示u_h^ℓ在V_{hi}上的限制，因此Ω_i上的离散问题(10.1)中 $u_{hi}^\ell \in V_{hi}$ 能够找到使得: g_i^k 是g关于$k \in \mathcal{N}_{hi}^{\Gamma_i}$的自由度值。
在FreeFem++中，U 可以表示关于$u_{h|i}^\ell$ 的向量，并且 向量U1是关于u_{hi}^ℓ 的向量:

```
real[int] U1(Ui.n);
real[int] b= onG .* U;
b  = onG ? b : Bi ;
U1 = Ai^-1*b;
```

其中 onG$[i] = (i \in \Gamma_i \setminus \Gamma)?1:0$，并且Bi 如下定义：

```
fespace Whi(Thi,P2);                                          //     定义有限元空间。
varf vPb(U,V)= int3d(Thi)(grad(U)'*grad(V)) + int3d(Thi)(F*V) +on(1,U=g) + on(10,U=G);
varf vPbon(U,V)=on(10,U=1)+on(1,U=0);
matrix Ai = vPb(Whi,Whi,solver=sparsesolver);
real[int] onG = vPbon(0,Whi);
```

```
  real[int] Bi=vPb(0,Whi);
```

其中Γ的freefem++标记为 1，$\Gamma_i \setminus \Gamma$ 的标记为 10.

为了建立关于 (10.2) 方程在过程i的更新部分，我们记 njpart 为区域Ω_i的邻域数 (i.e: π_j 是Ω_i的非0 邻域)，我们在数组jpart中储存 njpart所有邻域大小。 我们介绍矩阵中的两列数组，Smj[j]表示向量从i 到 j 的邻域过程，矩阵 $rMj[j]$ 表示之后除去领域 j 的区域。
因此转换更新部分计算 $v_i = \pi_i u_i + \sum_{j \in J_i} \pi_j u_j$ ，并且可以写成 freefem++更新函数:

```
func bool Update(real[int] &ui, real[int] &vi)
{ int n= jpart.n;
  for(int j=0;j<njpart;++j)  Usend[j][]=sMj[j]*ui;
  mpiRequest[int] rq(n*2);
  for (int j=0;j<n;++j) Irecv(processor(jpart[j],comm,rq[j  ]), Ri[j][]);
  for (int j=0;j<n;++j) Isend(processor(jpart[j],comm,rq[j+n]), Si[j][]);
  for (int j=0;j<n*2;++j) int k= mpiWaitAny(rq);
  //    应用一致局部划分 .
   vi = Pii*ui;                                                    //    设置πiui
   for(int j=0;j<njpart;++j)  vi += rMj[j]*Vrecv[j][];            //    增加 πjuj
   return true; }
```

其中缓冲区如下定义:

```
  InitU(njpart,Whij,Thij,aThij,Usend)  //    定义发送缓冲区
  InitU(njpart,Whij,Thij,aThij,Vrecv)  //    定义接收缓冲区
```

以及以下macro定义:

```
macro InitU(n,Vh,Th,aTh,U) Vh[int] U(n); for(int j=0;j<n;++j) {Th=aTh[j];  U[j]=0;}
//
```

第一个gmres算法:在介绍以下在子区域Ω_i 上的仿射$\mathcal{A}_i$算子之后，你可以通过使用并行 GMRES算法轻易地加速不动点算法。

```
func real[int] DJ0(real[int]& U) {
 real[int] V(U.n)    ,  b= onG .* U;
  b  = onG ? b : Bi ;
  V = Ai^-1*b;
  Update(V,U);
  V -= U;   return V; }
```

其中并行MPIGMRES或者MPICG 算法是并行解决以下问题$A_i x_i = b_i, i = 1, .., N_p$的一种简单方法，通过应用以下MPI代码减少整个过程的局部点:

```
template<class R> R ReduceSum1(R s,MPI_Comm * comm)
{   R r=0;
    MPI_Allreduce( &s, &r, 1 ,MPI_TYPE<R>::TYPE(),   MPI_SUM,  *comm );
    return r; }
```

这是MPIGC动态链工具完成。

第二个gmres算法: 使用Scharwz算法作为基础GMRES算法的预条件来解决并行问题。

```
func real[int] DJ(real[int]& U)                                    //    初始问题
```

```
{
  ++kiter;
  real[int] V(U.n);
   V =  Ai*U;
  V = onGi ? 0.: V;                                    //    移除边界项 ...
  return V;
}

func real[int] PDJ(real[int]& U)                                  //    预条件
{
  real[int] V(U.n);
  real[int] b= onG ? 0. :  U;
  V =  Ai^-1*b;
  Update(V,U);
  return U;
}
```

第三个gmres算法:增加一个粗略的算法。
首先在过程0上建立粗网格，并且

```
matrix AC,Rci,Pci;                                                //
if(mpiRank(comm)==0)
  AC = vPbC(VhC,VhC,solver=sparsesolver);                         //    粗问题

Pci=   interpolate(Whi,VhC);                                      //   粗网格上的投影
Rci =  Pci'*Pii;                                    //    限制在过程 i 上的网格划分 π_i

func bool  CoarseSolve(real[int]& V,real[int]& U,mpiComm& comm)
{
                                                                  //    求解粗问题
   real[int] Uc(Rci.n),Bc(Uc.n);
   Uc= Rci*U;
   mpiReduce(Uc,Bc,processor(0,comm),mpiSUM);
   if(mpiRank(comm)==0)
      Uc = AC^-1*Bc;
    broadcast(processor(0,comm),Uc);
   V = Pci*Uc;
}
```

新的预条件

```
func real[int] PDJC(real[int]& U)                                 //
{ //    Precon C1= Precon //, C2 precon Coarse
//    想法 :  F. Nataf.
  //    0 ~ (I C1A)(I-C2A) => I ~ - C1AC2A +C1A +C2A
  //    新的 Prec P= C1+C2 - C1AC2 = C1(I- A C2) +C2
  //    ( C1(I- A C2) +C2 ) Uo
  //    V = - C2*Uo
  //    ....
  real[int] V(U.n);
  CoarseSolve(V,U,comm);
  V = -V;                                                         //    -C2*Uo
  U  += Ai*V;                                                     //    U = (I-A C2) Uo
  real[int] b= onG ? 0. :  U;
  U =  Ai^-1*b;                                                   //    ( C1( I -A C2) Uo
```

```
  V = U -V;                                                    //
  Update(V,U);
  return U;
}
```

算法4的代码:

```
real epss=1e-6;
int rgmres=0;
if(gmres==1)
  {
   rgmres=MPIAffineGMRES(DJ0,u[],veps=epss,nbiter=300,comm=comm,
                         dimKrylov=100,verbosity=ipart ? 0: 50);
   real[int] b= onG .* u[];
   b  = onG ? b : Bi ;
   v[] = Ai^-1*b;
   Update(v[],u[]);
  }
else if(gmres==2)
  rgmres= MPILinearGMRES(DJ,precon=PDJ,u[],Bi,veps=epss,nbiter=300,comm=comm
                        ,dimKrylov=100,verbosity=ipart ? 0: 50);
else if(gmres==3)
   rgmres= MPILinearGMRES(DJ,precon=PDJC,u[],Bi,veps=epss,nbiter=300,comm=comm,
                          dimKrylov=100,verbosity=ipart ? 0: 50);
else                                                  //    演示的 Shwarz 算法 ...
   for(int iter=0;iter <10; ++iter)
     ....
```

如果我们有单位分解我们就可以并行求解。 为了建立这个划分我们这样做: 过程 1 的第一步建立整个区域的一个粗网格$\mathcal{T}_h^*$ ，并且建立划分π，函数常数等于 i ，在每个子区域$\mathcal{O}_i, i = 1, .., N_p$,的网格上运用 Metis 图像划分法 [?]， 在每个过程$1.., N_p$中的i 如下做

1. 像过程 1一样，网格 $\mathcal{T}_h^*$ (在 freefem++ 程序中称 Thii), 以及函数 π ，
2. 特征函数 $\mathbb{1}_{\mathcal{O}_i}$ 在区域 $\mathcal{O}_i$上，定义为 $(\pi = i)?1 : 0$，
3. 我们称Π_P^2 (类似地 Π_V^2)为在 P_h^* 上的L^2， 这里P_h^*在$\mathcal{T}_h^*$ 上的每个单位上为常数有限元函数空间(类似地 V_h^* 在 $\mathcal{T}_h^*$的每个单元上为仿射连续有限元空间)。 并且并行建立π_i 和Ω_i使得 $\mathcal{O}_i \subset \Omega_i$ ，其中 $\mathcal{O}_i = supp((\Pi_V^2\Pi_C^2)^m \mathbb{1}_{O_i})$， m 表示粗网格上重叠部分大小（一般一个），
   (这是在函数AddLayers(Thii,suppii[],nlayer,phii[]); 我们选取函数 $\pi_i^* = (\Pi_1^2\Pi_0^2)^m \mathbb{1}_{\mathcal{O}_i}$ 使得一致划分可以简单定义为
   $$\pi_i = \frac{\pi_i^*}{\sum_{j=1}^{N_p} \pi_j^*} \tag{10.3}$$
   区域Ω_i的邻域 J_i 以及在 V_{hi}上的局部划分可以用如下数组 jpart 和 njpart 定义:

```
Vhi pii=π_i^* ;  Vhi[int] pij(npij);        //    局部划分 1 = pii + ∑_j pij[j]

int[int] jpart(npart);   int njpart=0;
Vhi sumphi =  π_i^*  ;
for (int i=0;i<npart;++i)
  if(i != ipart ) {
     if(int3d(Thi)( π_j^*)>0) {
       pij[njpart]=π_j^*;
```

```
        sumphi[] += pij[njpart][];
        jpart[njpart++]=i;}}}
      pii[]=pii[] ./ sumphi[];
     for (int j=0;j<njpart;++j) pij[j][] = pij[j][] ./ sumphi[];
     jpart.resize(njpart);
```

4. 我们称$\mathcal{T}_{hij}^*$是$\mathcal{T}_{hi}$ 的子网格，其中 π_j 非零。 用函数trunc建立数组，

```
for(int jp=0;jp<njpart;++jp)
  aThij[jp]  = trunc(Thi,pij[jp]>1e-10,label=10);
```

5. 这一步我们已经建立好粗网格，所以我们可以通过拆分所有网格来建立最终的好网格：Thi, Thij[j],Thij[j]利用 freefem++ trunc 网格函数作限制和滑动。

6. 可以用以下代码建立发送/接收矩阵 sMj 和 rMj :

```
mesh3 Thij=Thi;                                          //    变量网格
fespace Whij(Thij,Pk);                                 //    变量有限元空间 ..
matrix Pii; Whi wpii=pii; Pii = wpii[];                 //    对应 ×πi的对角阵
matrix[int] sMj(njpart), rMj(njpart);                 //    M send/rend情况..
 for(int jp=0;jp<njpart;++jp)
   { int j=jpart[jp];
     Thij = aThij[jp];                                 //   变化网格 Whij,Whij
     matrix I = interpolate(Whij,Whi);                    //    Whij <- Whi
     sMj[jp] = I*Pii;                                   //    Whi -> s Whij
     rMj[jp] = interpolate(Whij,Whi,t=1); }}             //    Whij -> Whi
```

为了建立一个不算太差的应用，我用以下代码改变了变量参数值

```
include "getARGV.idp"
verbosity=getARGV("-vv",0);
int vdebug=getARGV("-d",1);
int ksplit=getARGV("-k",10);
int nloc = getARGV("-n",25);
string sff=getARGV("-p",",");
int gmres=getARGV("-gmres",3);
bool dplot=getARGV("-dp",0);
int nC = getARGV("-N" ,max(nloc/10,4));
```

用如下界面做出并行解向量 u在网格 T_h上的图像：

```
include "MPIplot.idp"
func bool  plotMPIall(mesh &Th,real[int] & u,string  cm)
{ PLOTMPIALL(mesh,Pk, Th, u,{ cmm=cm,nbiso=20,fill=1,dim=3,value=1}); return 1;}
```

注意macro中 {cmm=cm, ... =1}是一种引用macro的方法， 所以该内容是cmm=cm, ... =1。

第 11 章

并行稀疏求解器

并行稀疏求解器使用一些程序来求解线性方程。像连续求解一样，并行线性求解可以是直接的或者迭代的。 在FreeFem++ 中两者都有。

11.1 在 FreeFem++ 中使用并行稀疏求解器

我们回忆solver参数用如下命令定义： solve, problem, set (设置矩阵参数) ，并且根据一个双线性型建立矩阵。 在这些指令中， solver 必须设置成 sparsesolver 对于并行稀疏求解器。 我们对并行稀疏求解器增加了一些特殊参数指令行。它们是

- lparams: 整数参数向量 (l 是 c++ long格式)
- dparams: 实参数向量
- sparams: 字符串参数
- datafilename: 包含求解器参数的文件名

以下四个参数是向量且只对应于直接求解器。这些参数使得用户可以预处理矩阵 (参看并行稀疏求解器以上章节).

- permr: 行置换 (整数向量)
- permc: 列置换或转置行置换 (整数向量)
- scaler: 行放缩 (实向量)
- scalec: 列放缩 (实向量)

有两种方法控制求解器参数。第一种在.edp文件中定义参数 lparams, dparams and sparams。第二种从数据文件中读取求解器参数。这个文件称作datafilename。 如果lparams, dparams, sparams 或 datafilename没有提供给使用者，那么会使用求解器的默认值。

为了使用 FreeFem++ 中的并行求解器，我们需要载入关于这个求解器的动态程序库。例如使用在FreeFem中使用 MUMPS求解器作为并行求解器，需在.edp文件中写**load "MUMPS_FreeFem"**。如果程序库没有被载入，那么默认稀疏求解器会被载入 (默认稀疏求解器是 UMFPACK)。表格11.1 给出了不同程序库的新取值。
我们也增加了无参数的函数 (见表格 11.2)来改变.edp文件中的默认稀疏求解器。为了使用这些函数，我们需要载入求解器库。 使用不同并行稀疏求解器来求解同一个问题的例子在testdirectsolvers.edp中给出(查询 example++ −mpi)。

Libraries	default sparse solver real	default sparse solver complex
MUMPS_FreeFem	mumps	mumps
real_SuperLU_DIST_FreeFem	SuperLU_DIST	previous solver
complex_SuperLU_DIST_FreeFem	previous solver	SuperLU_DIST
real_pastix_FreeFem	pastix	previous solver
complex_pastix_FreeFem	previous solver	pastix
hips_FreeFem	hips	previous solver
hypre_FreeFem	hypre	previous solver
parms_FreeFem	parms	previous solver

Table 11.1: 实、复运算的默认稀疏求解器当我们载入并行稀疏求解器库

function	default sparse solver real	default sparse solver complex
defaulttoMUMPS()	mumps	mumps
realdefaulttoSuperLUdist()	SuperLU_DIST	previous solver
complexdefaulttoSuperLUdist()	previous solver	SuperLU_DIST
realdefaultopastix()	pastix	previous solver
complexdefaulttopastix()	previous solver	pastix
defaulttohips()	hips	previous solver
defaulttohypre()	hypre	previous solver
defaulttoparms()	parms	previous solver

Table 11.2: 允许改变实复运算默认稀疏求解器的函数和这些函数的结果

Example 11.1 (testdirectsolvers.edp)

```
load "../src/solver/MUMPS_FreeFem"
                                        //    默认求解器:  实-> MUMPS, 复 -> MUMPS
load "../src/solver/real_SuperLU_DIST_FreeFem"
                                      //    默认求解器:  实-> SuperLU_DIST, 复 -> MUMPS
load "../src/solver/real_pastix_FreeFem"
                                        //    默认求解器:  实-> pastix, 复 -> MUMPS

                                                    //    用pastix求解
{
   matrix A =
      [[ 1,  2,     2,  1, 1],
       [ 2,    12,     0,  10 , 10],
       [ 2,     0,     1,  0, 2],
       [ 1,     10,    0,  22, 0.],
       [ 1,     10,    2,  0., 22]];

   real[int] xx = [ 1,32,45,7,2], x(5), b(5), di(5);
    b=A*xx;
    cout << "b="  << b  << endl;
    cout << "xx=" << xx << endl;

    set(A,solver=sparsesolver,datafilename="ffpastix_iparm_dparm.txt");
    cout << "solving solution" << endl;
```

```
    x = A^-1*b;
    cout << "b=" << b << endl;
    cout << "x=" << endl; cout << x << endl;
    di = xx-x;
    if(mpirank==0){
    cout << "x-xx="<< endl; cout << "Linf "<< di.linfty << " L2 " << di.l2 << endl;
    }
}

                                                        //    用SuperLU_DIST求解
realdefaulttoSuperLUdist();
                                    //    默认求解器:  实-> SuperLU_DIST, 复 -> MUMPS
{
   matrix A =
      [[ 1,  2,     2,  1, 1],
       [ 2,    12,     0,  10 , 10],
       [ 2,     0,     1,  0, 2],
       [ 1,     10,    0,  22, 0.],
       [ 1,     10,    2,  0., 22]];

   real[int] xx = [ 1,32,45,7,2], x(5), b(5), di(5);
    b=A*xx;
    cout << "b="  << b  << endl;
    cout << "xx=" << xx << endl;

   set(A,solver=sparsesolver,datafilename="ffsuperlu_dist_fileparam.txt");
    cout << "solving solution" << endl;
    x = A^-1*b;
    cout << "b=" << b << endl;
    cout << "x=" << endl; cout << x << endl;
    di = xx-x;
    if(mpirank==0){
    cout << "x-xx="<< endl; cout << "Linf "<< di.linfty << " L2 " << di.l2 << endl;
    }
}

                                                        //    用MUMPS求解
defaulttoMUMPS();
                                        //    默认求解器:  实-> MUMPS, 复 -> MUMPS
{
    matrix A =
      [[ 1,  2,     2,  1, 1],
       [ 2,    12,     0,  10 , 10],
       [ 2,     0,     1,  0, 2],
       [ 1,     10,    0,  22, 0.],
       [ 1,     10,    2,  0., 22]];

    real[int] xx = [ 1,32,45,7,2], x(5), b(5), di(5);
    b=A*xx;
    cout << "b="  << b  << endl;
    cout << "xx=" << xx << endl;

    set(A,solver=sparsesolver,datafilename="ffmumps_fileparam.txt");
    cout << "solving solution" << endl;
    x = A^-1*b;
    cout << "b=" << b << endl;
```

```
    cout << "x=" << endl; cout << x << endl;
    di = xx-x;
    if(mpirank==0){
    cout << "x-xx="<< endl; cout << "Linf "<< di.linfty << " L2 " << di.l2 << endl;
    }
}
```

11.2 稀疏直接求解器

在本节中，我们介绍 FreeFem++ 中的直接求解器。

11.2.1 MUMPS求解器

MUltifrontal Massively Parallel Solver (MUMPS) 是一个免费库 [**?**, **?**, **?**]。这个程序包用直接法求解线性系统$A\,x = b$ 其中A 是稀疏方阵。 MUMPS中这个方阵可以是非对称的，对称正定的或者一般对称的。MUMPS使用的方法是基于多波前法的直接方法 [**?**]。 它建立了一个直接分解 $A = L\,U$, $A = L^t\,D\,L$ 依赖于矩阵 A的对称性。 MUMPS使用以下程序库: BLAS[**?**, **?**], BLACS and ScaLAPACK[**?**]。

注 7 *MUMPS* 无法求解矩形矩阵的线性系统。

MUMPS的安装 在 FreeFem++ 中使用MUMPS，必须安装 MUMPS安装包。MUMPS由 Fortran 90写成。并行版本使用 MPI [**?**]，BLAS [**?**, **?**]，BLACS 和 ScaLAPACK[**?**]传递信息。因此需要编译程序以及 MPI, BLAS, BLACS and ScaLAPACK 。 安装步骤在文件 README_COMPILE 目录 src/solver of FreeFem++ 中给出。

建立FreeFem++ 中MUMPS界面的程序库: FreeFem++ 中MUMPS界面在文件 MUMPS_freefem.cpp中给出 (目录 src/solver/)。这个界面同发布的3.8.3 和 3.8.4 的 MUMPS一同工作。为了在FreeFem++ 中使用MUMPS，我们需要对应于这个界面的程序库。介绍如何获得这个程序库在文件README_COMPILE 目录 src/solver of FreeFem++ 中给出。我们在这里回忆一下步骤。找到目录 src/solver 在 FreeFem++ 安装包。编辑文件 makefile-sparsesolver.inc : 注释 Section 1, 注释行对应于程序库 BLAS, BLACS, ScaLAPACK, Metis, 注释 Section 2和 Section 3 对应于 MUMPS求解器的段落。 然后在终止窗口键入**make mumps**。
在描述FreeFem++ 中MUMPS的方法之前，我们先简短介绍 MUMPS参数。

多前段大规模并行求解器参数 （MUMPS parameters）: 在多前段大规模并行求解器(MUMPS)中，有四个输入参数(参见 [**?**]). 两个整数SYM 与 PAR, 一个长度为40的整数向量 INCTL 和一个长度为15的实数向量 CNTL. 第一个参数给出了矩阵的类型：0 代表非对称矩阵，1 代表 对称正定矩阵，2 代表 一般对称矩阵。第二个参数告诉我们在分解期间主机处理器是否工作和解决步骤：PAR = 1表示主机处理器工作和PAR = 0表示主机不工作。 参数 INCTL 和 CNTL 是 MUMPS 的控制参数。在 MUMPS 中的向量 ICNTL 和 CNTL 在FORTRAN中成为指标为1的向量。在文件ffmumps_fileparam.txt中，有对 ICNTL and CNTLA 所有参数的简洁描述。更多细节可以参见用户手册 [**?**]。
现在我们来描述MUMPS主要参数ICNTL中的元素。

输入矩阵参数 参数 ICNTL(5) and ICNTL(18)决定了输入矩阵。 矩阵形式(矩阵的样式和输入)被 INCTL(5)(INCTL(18)) 所决定。参数ICNTL(5)取0 表示聚集形式，取1表示元素形式。在当前版本的Freefem++，我们认为FE矩阵或矩阵的格式存储在组装。因此，INCTL(5)被视为0值。ICNTL(18)主要选项：INCLTL(18)=0在中央主机处理器上，ICNTL(18)=3分布式输入矩阵模式和条目（选择分布式矩阵推荐MUMPS的开发者）。ICNTL(18)的其他数值参见用户手册。这些值也可以在Freefem++中使用。 在`FreeFem++` 实现默认选项是ICNTL(5)=0 和 ICNTL(18)=0。

预处理参数 预处理矩阵 A_p 的有效分解定义为

$$A_p = P\, D_r\, A\, Q_c\, D_c P^t$$

其中 P 置换矩阵, Q_c 是列置换, D_r 和 D_c 分别是行变换和列变换的对角矩阵。P 的排序是由参数 ICNTL(7) 所决定的。 Q_c 是由参数ICNTL(6) 决定的。行变换和列变换由参数 ICNTL(18) 所决定。这些选择与 ICNTL(12) 强烈相关（更多细节参见MUMPS的文件）。在`FreeFem++` 中，参数 permr, scaler, and scalec 分别给出了置换矩阵(P), 行变换 (D_r) 和 列变换 (D_c)。

在FreeFem++中调用MUMPS 为了在FreeFem++中调用MUMPS，我们需要装载动态科学库 MUMPS_freefem.dylib (MacOSX), MUMPS_freefem.so (Unix) 或者 MUMPS_freefem.dll (Windows). 在文件 the .edp中，装载"MUMPS_freefem". 现在我们给出两个方法在`FreeFem++` 中选择MUMPS求解器。

在文件 .edp file中的求解器参数: 这种方法，我们需要给出参数 `lparams` 和 `dparams`。这些参数被定义为

$$\begin{aligned} & \text{lparams}[0] = \text{SYM}, \\ & \text{lparams}[1] = \text{PAR}, \\ \forall i = 1, \ldots, 40, \quad & \text{lparams}[i{+}1] = \text{ICNTL}(i). \end{aligned}$$

$$\forall i = 1, \ldots, 15, \quad \text{dparams}[i-1] = \text{CNTL}(i).$$

在文件中读取求解器参数: 在 `FreeFem++` 中，数据文件的结构为: 第一行参数为 SYM ，第二行参数为 PAR ，下面行是参数 ICNTL and CNTL 的数值.这个参数文件的一个例子在 `ffmumpsfileparam.txt`中被给出。

```
0          /* SYM :: 0 代表非对称矩阵, 1 代表对称正定矩阵 , 2 代表一般的对称矩阵*/
1          /* PAR :: 0 代表在分解和求解步骤中不工作, 1 代表在分解和求解步骤中工作*/
 -1        /* ICNTL(1) :: 对于错误信息有输出流  */
 -1        /* ICNTL(2) :: 输出打印错误, 静态和警告信息 */
 -1        /* ICNTL(3) :: 全局信息 */
 0         /* ICNTL(4) :: 打印错误, 静态和警告信息的水平 */
 0         /* ICNTL(5) :: 矩阵格式 : 0 代表组装格式, 1 代表基本格式. */
 7         /* ICNTL(6) :: 分析态中矩阵排列或尺度变换的控制选项 */
 3         /* ICNTL(7) :: 主元顺序策略 : AMD, AMF, metis, pord scotch*/
 77        /* ICNTL(8) :: 行与列的变换策略 */
 1         /* ICNTL(9) :: 0 求解 Ax = b, 1 表示求解转置系统  A^t x = b :  在现行freefem++版
本中不考虑参数 */
 0         /* ICNTL(10) :: 迭代改善的步数 */
 0         /* ICNTL(11) :: 与ICNTL(9)中线性系统有关的静态 */
```

```
 1          /* ICNTL(12) :: 一般对称矩阵的有约束排列策略 */
 0          /* ICNTL(13) :: 控制root frontal 矩阵的分解 */
 20         /* ICNTL(14) :: 在工作空间中的百分比增加 (默认 20%)*/
 0          /* ICNTL(15) :: 在现行MUMPS中没用到 */
 0          /* ICNTL(16) :: 在现行MUMPS中没用到 */
 0          /* ICNTL(17) :: 在现行MUMPS中没用到 */
 3          /* ICNTL(18) :: 对于给定矩阵模式和矩阵元素的方法  : */
 0          /* ICNTL(19) :: 返回Schur补矩阵的方法 */
 0          /* ICNTL(20) :: 右端项形式 ( 0 代表稠密形式, 1 代表稀疏形式) : 对于freefem++,
参数被设为0 */
 0          /* ICNTL(21) :: 0, 1 kept保持分布式解 : 在现行freefem++版本中不考虑参数
*/
 0          /* ICNTL(22) :: 控制核内核外(OOC)设施 */
 0          /* ICNTL(23) :: MUMPS可以分配至每个工作处理器的工作内存的最大值, 以兆字节计
*/
 0          /* ICNTL(24) :: 控制0主元的探测 */
 0          /* ICNTL(25) :: 控制零空间基的计算 */
 0          /* ICNTL(26) :: 该参数只在Schur选择(ICNTL(19) 非零)中很重要: 在现行freefem++版
本中不考虑参数 */
 -8         /* ICNTL(27) (在MUMPS的下一个版本中会改变的实验性参数) :: 在求解态控制多右
端项的分块因子 : 在现行freefem++版本中不考虑参数 */
 0          /* ICNTL(28) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(29) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(30) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(31) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(32) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(33) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(34) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(35) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(36) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(37) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(38) :: 在现行MUMPS版本中未使用*/
 1          /* ICNTL(39) :: 在现行MUMPS版本中未使用*/
 0          /* ICNTL(40) :: 在现行MUMPS版本中未使用*/
 0.01       /* CNTL(1) :: 数值选主元的相对临界值 */
 1e-8       /* CNTL(2) :: 迭代改善的停机准则 */
 -1         /* CNTL(3) :: 无主元探测的临界值 */
 -1         /* CNTL(4) :: 决定部分选主元的临界值 */
 0.0        /* CNTL(5) :: 不选主元固定 */
 0          /* CNTL(6) :: 在现行MUMPS版本中未使用 */
 0          /* CNTL(7) :: 在现行MUMPS版本中未使用 */
 0          /* CNTL(8) :: 在现行MUMPS版本中未使用 */
 0          /* CNTL(9) :: 在现行MUMPS版本中未使用 */
 0          /* CNTL(10) :: 在现行MUMPS版本中未使用 */
 0          /* CNTL(11) :: 在现行MUMPS版本中未使用 */
 0          /* CNTL(12) :: 在现行MUMPS版本中未使用 */
 0          /* CNTL(13) :: 在现行MUMPS版本中未使用 */
 0          /* CNTL(14) :: 在现行MUMPS版本中未使用 */
 0          /* CNTL(15) :: 在现行MUMPS版本中未使用 */
```

如果求解器参数没有给出，则使用MUMPS求解器的默认参数。

例子 在examples++-mpi 的_MUMPS.edp文件中，有一个在 FreeFem++ 中调用MUMPS的例子。

11.2.2 SuperLU 分布式求解器

程序包 SuperLU_DIST [**?**, **?**] 利用LU分解求解线性系统。在 BSD 认证下，它是一个免费程序包。这个程序的网址为http://crd.lbl.gov/∼xiaoye/SuperLU. 这个程序包支持实复矩阵的计算。SuperLU_DIST 的实施方法是一种广义节点的方法[**?**]。这个包的新版本包括一个并行符号分解。这一程序包支持C和MPI。

SuperLU_DIST 的安装: 为了在 `FreeFem++` 中使用 SuperLU_DIST,需要安装 SuperLU_DIST 程序包。为了汇编，我们需要MPI和PARMETIS程序包。安装程序 README COMPILE 后可获取该安装包。

在 `FreeFem++` 中，生成 SuperLU_DIST 界面: 文件 real_SuperLU_DIST_FreeFem.cpp (resp. complex_SuperLU_DIST_FreeFem.cpp)提供了 SuperLU_DIST 的实数（复数）运算功能。这些文件在 src/solver/ 中可以找到。这些界面与 SuperLU_DIST 的3.2.1版本兼容。为了在 `FreeFem++` 中使用 SuperLU_DIST,我们需要加载这些界面。在src/solver of `FreeFem++` 的文件 README_COMPILE 中，可以找到操作的方法。我们回顾一下过程，在安装包 `FreeFem++` 的文件 src/solver 中。 在你的系统中编辑文件 makefile-sparsesolver.inc : 注释部分1 , 文件 BLAS, Metis, ParMetis 第二节的注释部分，文件 SuperLU_DIST solver 的注释在第三节。在终端**make rsludist** (**make csludist**) 获取接口动态库实数（复数）算法。
在讲述`FreeFem++` 中调用 SuperLU_DIST 之前，我们先简单介绍一下 SuperLU_DIST 的参数。

SuperLU_DIST 参数: 我们介绍一下 SuperLU_DIST 的参数。程序包 SuperLU_DIST 使用一个两维逻辑程序组。这个网格程序包括*nprow*行过程和*npcol*列过程，且满足 $N_p = nprow\ npcol$ 其中where N_p 是 SuperLU_DIST 所有分配程序的数目。
命令 "matrix= " 可以用来输入矩阵参数，在 sparams 中内部参数或参数文件的第三行。不同的输入值如下

matrix = assembled	在所有处理器上均有全局矩阵
matrix = distributedglobal	在所有处理器间全局矩阵是分布式的
matrix = distributed	输入矩阵是分布式的 (尚未实现)

在 Users-callable [**?**] 里讲述了关于 SuperLU_DIST 的选项。The parameter在不同步骤中，参数 Fact 和 TRANS 将 SuperLU_DIST 联系起来。由于这个原因，用户的赋值不在考虑之内。
关于矩阵 A_p 的 LU 分解按如下计算。

$$A_p = P_c\, P_r\, D_r\, A\, D_c\, P_c^t$$

其中 P_c 和 P_r 分别是置换矩阵的行和列， D_r 和 D_c 分别是行变换和列变换的对角矩阵。 选项 RowPerm (ColPerm) 控制了置换矩阵的行(列)。 D_r 和 D_c 由 DiagScale 控制。 在 `FreeFem++` 中，参数 permr, permc, scaler, 和 scalec 分别给出了用户输入的行置换，列置换，行变换和列变换。 关于 LU 分解的其他参数还有 ParSymFact 和 ReplaceTinyPivot。 并行符号分解需要两个程序同时运行，并借助 ParMetis 排序 [**?**]. 在文件 ffsuperlu_dist_fileparam.txt中描述了 SuperLU_DIST 的默认选项。

在 FreeFem++中调用SuperLU_DIST 为了在 FreeFem++中调用SuperLU_DIST , 我们需要装载动态界面。在文件 .edp 里， **load "real_superlu _DIST_FreeFem"** (**load "complex_superlu_DIST_FreeFem"**) 提供了实数（复数）运算。

.edp 文件中的求解器参数: 为了调用 SuperLU_DIST 的内部参数，我们使用参数 sparams 。在 sparams 中 SuperLU_DIST 的参数定义为

```
sparams  ="nprow=1, npcol=1, matrix= distributedgloba, Fact= DOFACT, Equil=NO,
          ParSymbFact=NO, ColPerm= MMD_AT_PLUS_A, RowPerm= LargeDiag,
          DiagPivotThresh=1.0, IterRefine=DOUBLE, Trans=NOTRANS,
          ReplaceTinyPivot=NO, SolveInitialized=NO, PrintStat=NO, DiagScale=NOEQUIL "
```

在文件 ffsuperlu_dist_fileparam.txt中，给出了这些参数。如果用户没有输入其中的某些参数，系统会使用默认参数。

在一个文件中读取求解器参数: 文件 ffsuperlu_dist_fileparam.txt 给出了SuperLU_DIST的数据文件结构。 (FreeFem++ 界面的默认值).

```
1                                   /* nprow : 整数值     */
1                                   /* npcol : 整数值      */
distributedglobal          /* 矩阵输入 : assembled, distributedglobal, distributed
*/
DOFACT                     /* Fact  : DOFACT, SamePattern, SamePattern_SameRowPerm,
FACTORED */
NO                              /* Equil : NO, YES */
NO                              /* ParSymbFact : NO, YES */
MMD_AT_PLUS_A  /* ColPerm : NATURAL, MMD_AT_PLUS_A, MMD_ATA, METIS_AT_PLUS_A, PARMETIS,
MY_PERMC */
LargeDiag                   /* RowPerm : NOROWPERM, LargeDiag, MY_PERMR */
1.0                              /* DiagPivotThresh : 实值 */
DOUBLE                     /* IterRefine : NOREFINE, SINGLE, DOUBLE, EXTRA */
NOTRANS                 /* Trans      : NOTRANS, TRANS, CONJ*/
NO                              /* ReplaceTinyPivot  : NO, YES*/
NO                              /* SolveInitialized  : NO, YES*/
NO                              /* RefineInitialized : NO, YES*/
NO                              /* PrintStat : NO, YES*/
NOEQUIL                    /* DiagScale : NOEQUIL, ROW, COL, BOTH*/
```

如果求解器参数没有给出，则使用 SuperLU_DIST 求解器的默认选项。

Example 11.2 在 *examples++-mpi* 中的 *testsolver_superLU_DIST.edp* 文件里，有在 *FreeFem++* 下调用 *SuperLU_DIST* 的一个简单例子。

11.2.3 Pastix solver

Pastix (Parallel Sparse matrix package) 是一款在 CECILL-C 认证下的免费科学计算程序包。这个程序包利用直接和 block ILU(k) 迭代方法解决线性sparse 线性系统问题。该求解器可以处理对称的实复矩阵问题。[**?**].

Pastix 的安装: To used Pastix in为了在 FreeFem++ 中使用 Pastix ,首先需要安装 pastix 程序包。为了编辑该程序包，需要 fortran 90 编译器compiler, scotch [**?**] 或者 Metis [**?**] ordering library 和 MPI。在文件 .src/solver/ README_COMPILE 中有该程序包的安装步骤。

在 FreeFem++ 下生成 pastix 对接界面: 在文件 real_pastix_FreeFem.cpp (complex_pastix_FreeFem.cpp) 中有 FreeFem++ 对接 pastix 的实（复）数运算版本。该界面与 pastix 2000 版本相互兼容，并且是针对全局矩阵。我们也可以用该接口进行分布式计算。为了在 FreeFem++ 中使用 pastix ，我们需要装载该接口。在 src/solver 里的文件 README_COMPILE 讲述了如何获得该界面。我们回顾这个过程。先装载 FreeFem++ 中的 src/solver 。 对系统编辑文件makefile-sparsesolver.inc : 注释部分1 , 文件 BLAS, Metis, ParMetis 第二节的注释部分，文件 pastix solver 的注释在第三节。在终端**make rpastix** (**make cpastix**) 获取接口动态库实数（复数）算法。
在讲述FreeFem++ 中调用 pastix 之前，我们先简单介绍一下 pastix 的参数。

Pastix 参数: FreeFem++ 的输入参数 matrix 是基于 pastix 接口的。 matrix=非分布式矩阵的集合。与 SuperLU_DIST 相同，在 Pastix 中有四个参数 : iparm, dparm, perm 和 invp. 这些参数分别是整数型参数 (长度64 的向量),实数型参数 (长度64的向量), 置换矩阵和逆矩阵。参数iparm 和 dparm 在 [**?**] 有具体描述。用户可以利用 FreeFem++ 中的参数 permr 和 permc 分别输入置换矩阵和逆矩阵。

在 .edp 文件中的求解器参数: 为了在该情形下调用 Pastix ，我们需要指出参数 **lparams** 和 **dparams**. 这些参数定义如下

$$\forall i = 0, \ldots, 63, \qquad \text{lparams}[i] = \text{iparm}[i].$$

$$\forall i = 0, \ldots, 63, \qquad \text{dparams}[i] = \text{dparm}[i].$$

在文件中读取求解器参数: 在 FreeFem++ 中，数据文件的结构为: 第一行为矩阵结构参数，下面行是参数 iparm 和 dparm 的数值。

```
assembled  /* 矩阵输入 :: assembled, distributed global and distributed */
iparm[0]
iparm[1]
...
...
iparm[63]
dparm[0]
dparm[1]
...
...
dparm[63]
```

在文件 ffpastix_iparm_dparm.txt 中有一个例子来具体描述这些参数。该文件在 pastix 程序包中的文件 iparm.txt 和 dparm.txt 里。

如果没有输入求解器参数，就会使用 pastix 求解器的默认参数。

例子: 在 examples++-mpi 中的 testsolver_pastix.edp 文件里，有在 FreeFem++ 下调用 pastix 的一个简单例子。

在表格 11.3 里，我们回顾下不同求解器中的不同矩阵。

直接求解器	正方形矩阵			长方形矩阵		
	sym	sym pattern	unsym	sym	sym pattern	unsym
SuperLU_DIST	是	是	是	是	是	是
MUMPS	是	是	是	否	否	否
pastix	是	是	否	否	否	否

Table 11.3: 直接 sparse 求解器可用的不同矩阵类型

11.3 并行稀疏迭代求解器

本节讨论 **迭代求解器**, 我们选择 *pARMS* [?] , *HIPS* [?] 和 *Hypre* [?]。每个软件的并行预处理器都不同。 *pARMS* 执行代数域分解预处理，例如 additive Schwartz [?] 和 interface 方法 [?];而 HIPS 执行多层不完全分解 [?] ， HYPRE 执行多层次代数网格预处理 [?] 和 并行渐进逆预处理 [?]。

在 FreeFem++ 中，上述程序需要独立安装。 为了保证图像分块软件METIS [?] 或者 Scotch [?] 成功运行，安装 the MPI 科学库是必须的。
所有的预处理器都是使用 Krylov 子空间加速的。 Krylov 子空间方法是在仿射空间 $x_0 + K_m$ ，令 $b - Ax \perp \mathcal{L}_m$ 来寻找线性方程组 $Ax = b$ 解得迭代算法，这里 K_m 是维数为 m 的 Krylov 子空间，定义为 $K_m = \{r_0, Ar_0, A^2r_0, ..., A^{m-1}r_0\}$ and $\mathcal{L}_m$ 是由 Krylov 子空间类型决定的另一个 m 维子空间。例如在 GMRES 中,$\mathcal{L}_m = AK_m$.

使用接口是很容易的，于是在 FreeFem++ 调用不同的软件也是方便的。只需要装载求解器，然后使用求解器参数识别参数。在本章的剩余部分，当我们谈论 Krylov 子空间的意思是在 GMRES, CG 和 BICGSTAB 中的一个 Krylov 子空间方法中。

11.3.1 pARMS 求解器

pARMS (*parallel Algebraic Multilevel Solver*) 是一款由明尼苏达大学[?]的 Youssef Saad 和 al 开发的软件。 这个软件是专门解决大型稀疏非对称线性方程组的。在 pARMS 中开发的求解器是Krylov子空间型的。 它包括 GMRES 的变体，如 FGMRES(Flexible GMRES) , DGMRES(Deflated GMRES) [?] 和 BICGSTAB。 pARMS 也执行如同 RAS (Restricted Additive Schwarz)[?] 和 Schur Complement [?]型的并行预处理器。
所有的并行预处理器是基于域分解原理的。 因此，矩阵 A 被分解为子矩阵 $A_i (i = 1, ..., p)$ 其中 p 表示需要被分解的个数。 所有的 A_i 矩阵形成了最初的矩阵。 只要局部求解 A_i (见 [?])上的问题，就可以获得整个问题的解. 因此，区别在于 A 上的迭代和 A_i 上的局部迭代。 为了解决 A_i 上的局部问题，有一些预处理器 **ilut** (Incomplete LU with threshold), **iluk**(Incomplete LU with level of fill in) 和 **ARMS**(Algebraic Recursive Multilevel Solver). 但是要在 FreeFem++ 中使用 pAMRS ，需要先安装 pAMRS。

pARMS 的安装 为了安装 pARMS，先在 [?] 下载 pARMS 安装包。 下载完成之后，可以打开安装包，按文件 README 中过程开始安装，生成科学库 **libparms.a**。

使用 pARMS 作为 FreeFem++ 的接口 在 FreeFem++ 中调用 pARMS 求解器之前，要编辑文件 *parms_FreeFem.cpp* 产生一个动态科学库 *parms_FreeFem.so*。 为了实现，打开 FreeFem++ 的目录 *src/solver* ，编辑文件 *makefileparms.inc* 以指定以下变量:

$PARMS\_DIR$:	pARMS 文件夹
$PARMS\_INCLUDE$:	pARMS 头文件的文件夹
$METIS$:	METIS 文件夹
$METIS\_LIB$:	METIS 库
MPI :	MPI 文件夹
$MPI\_INCLUDE$:	MPI 头文件
$FREEFEM$:	FreeFem++ 文件夹
$FREEFEM\_INCLUDE$:	稀疏线性求解器的FreeFem++ 头文件
$LIBBLAS$:	Blas 库

编辑该文件之后，在命令行运行 **make parms** 以生成 $parms\_FreeFem.so$。
和往常一样，我们将展示一些在 FreeFem++ 中调用 pARMS 的例子。 有三种方法：

例子 1: 默认参数 这个例子来自于 FreeFem++ [?] 的用户手册第12页。

Example 11.3

```
load parms_freefem                              //    告诉 FreeFem 你将使用 pARMS
border C(t=0,2*pi){x=cos(t); y=sin(t);label=1;}
mesh Th = buildmesh (C(50));
fespace Vh(Th,P2);
Vh u,v;
func f= x*y;
problem Poisson(u,v,solver=sparsesolver) =          //    将使用的双线性部分
   int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))   //    一个稀疏求解器, 在这里是 pARMS
   - int2d(Th)( f*v)                                                  //    右端项
   + on(1,u=0)  ;                                  //    Dirichlet 边界条件

real cpu=clock();
Poisson;                                                          //    求解 PDE
plot(u);
cout << " CPU time = " << clock()-cpu << endl;
```

在例子 11.3 的第一行，我们安装了带 FreeFem++ 接口的 pARMS 动态科学库。 在此之后，在第七行，我们指定双线性型被最后的稀疏线性求解器 pARMS 载入。
若用户没有输入任何参数，则采用 pARMS 中的默认参数。
有一些默认参数:
求解器=FGMRES, **Krylov 维数**=30, **Krylov 的最大维数**=1000, **收敛精度**=$1e-08$.(参见 Saad [?] 的书可以知道所有参数的意义。)
预处理器=Restricted Additif Schwarz [?], **内 Krylov 空间维数**=5, **最大内 Krylov 空间维数**=5, **内部预处理器**=ILUK.

为了初始化求解器中用到的参数，用户可以输入一个 **整数参数** 和一个 **实数参数** ，或者输入一个包含参数的文件 **file** 。

例子 2: 用户输入两个向量来初始化参数 我们来考虑 Navier Stokes 的例子 11.4 。在这个例子中，我们用 pARMS 来求解由离散 Navier Stokes 方程导出的线性系统。 求解器的参数需要用户进行指定。

Example 11.4 (Stokes.edp)
```
include "manual.edp"
include "includes.edp";
include "mesh_with_cylinder.edp";
```

```
include "bc_poiseuille_in_square.edp";
include "fe_functions.edp";
0: load parms_FreeFem
1: int[int] iparm(16); real[int] dparm(6);
2: int ,ii;
3: for(ii=0;ii<16;ii++){iparm[ii]=-1;}  for(ii=0;ii<6;ii++) dparm[ii]=-1.0;
4: fespace Vh(Th,[P2,P2,P1]);
5: iparm[0]=0;
6: varf Stokes ([u,v,p],[ush,vsh,psh],{solver=sparsesolver}) =
 int2d(Th)( nu*( dx(u)*dx(ush) + dy(u)*dy(ush) + dx(v)*dx(vsh) + dy(v)*dy(vsh) )
            - p*psh*(1.e-6)                                          //    p epsilon
            - p*(dx(ush)+dy(vsh))                          //   + dx(p)*ush + dy(p)*vsh
            - (dx(u)+dy(v))*psh                                      //    psh div(u)
  )
 + on(cylinder,infwall,supwall,u=0.,v=0.)+on(inlet,u=uc,v=0);     //    边界条件
7: matrix AA=Stokes(VVh,VVh);
8: set(AA,solver=sparsesolver,lparams=iparm,dparams=dparm);   //   设定线性求解器
为 pARMS
9: real[int] bb= Stokes(0,VVh); real[int] sol(AA.n);
10: sol= AA^-1 * bb;
```

初始化线性求解器的参数需要两个输入向量。在例子 11.4 的第一行，声明了向量 (**int[int] iparm(16); real[int] dparm(6);**)。 在第三行用负值初始化向量，这么做是因为 pARMS 中的所有参数都是正的。 and 如果没有改变向量中的负值输入，参数将会设置为默认值。 在表格 (table 11.4 和 11.5) 中，我们列出了不同输入向量的具体意义。
我们在 Grid5000的 cluster paradent 上运行例11.4, 结果在表11.8.
在这个例子中，我们固定矩阵的长宽（至于有限元，我们固定网格），增加解决线性系统所使用的进程数。当进程数增加时，虽然迭代数在增加，但我们可以看到求解线性系统的时间在减少。这个例子说明了，使用 pARMS 来求解由离散偏微分方程所导出的线性系统问题，可以大量减少模拟的时间。

11.3.2 HIPS 接口

HIPS (多重迭代并行求解器) 是一款有效处理大型稀疏线性系统的并行迭代求解的科学程序包。在 CeCILL-C 认证下，HIPS 是免费的软件。 该接口与 **1.2 beta.rc4** HIPS的候选版本兼容。
HIPS 执行两类求解器，迭代类(GMRES, PCG) 和 直接类。 关于预处理器，HIPS执行一类多层 ILU。 关于预处理器的更多信息参见 [**?**, **?**]。

HIPS的安装 首先在 [**?**]下载HIPS压缩包, 解压， 打开HIPS的SOURCE目录。 不同的系统有不同的安装方式。例如， 在Linux系统中复制文件 *Makefile_Inc_Examples/makefile.inc.gnu* 到HIPS的根目录下 并重命名为**makefile.inc**。然后，通过编辑 **makefile.inc** 来设定参数的值和类型，最后输入 **make all**.

将HIPS当做FreeFem++的接口来使用 调用FreeFem++内部的HIPS求解器之前, 必须先编译文件 *hips_FreeFem.cpp* 来建立动态字典 *hips_FreeFem.so*. 具体做法是，打开目录 *src/solver* of FreeFem++ ，然后编辑文件 *makefile.inc* 来指定以下参数:

iparm 的条目	**每个条目的含义**
iparm[0]	Krylov 子空间方法. 该参数的不同取值见表格 11.6
iparm[1]	预处理. 该参数的不同预处理见表格 11.7
iparm[2]	Krylov 子空间的外迭代维数: 默认值 30
iparm[3]	外迭代的最大迭代数: 默认值 1000
iparm[4]	运行时工作的层级数。
iparm[5]	内迭代中的 Krylov 子空间维数: 默认值 3
iparm[6]	内迭代中的最大迭代数: 默认值 3
iparm[7]	对称(=1) 或非对称矩阵: 默认值 0(非对称矩阵)
iparm[8]	不同子域间的重叠数: 默认值 0(没有重叠)
iparm[9]	输入矩阵是否变换处理: 默认值 1 (矩阵需要变换处理)
iparm[10]	运行时工作的模块数: 默认值 20
iparm[11]	lfil0 (ilut, iluk, and arms) : 默认值 20
iparm[12]	lfil for Schur complement const : 默认值 20
iparm[13]	lfil for Schur complement const : 默认值 20
iparm[14]	运行时是否使用 ILU 中的多重染色: 默认值 1
iparm[15]	内部迭代 : 默认值 0
iparm[16]	求解显示的信息:默认 0(没有信息显示). 0: 没有信息显示， 1: 收敛信息，如迭代数和残差， 2: 不同的步骤的处理时间， 3 : 显示所有信息。

Table 11.4: 例子 11.4 中， **lparams** 各变量的意义

dparm的条目	**每个条目的含义**
dparm[0]	外迭代精度: 默认值 1e-08
dparm[1]	内迭代精度: 默认值 1e-2
dparm[2]	使用的对角值域的限度: 默认值 0.1
dparm[3]	削减的限度指定误差（droptol0） (ilut, iluk, and arms) : 默认值 1e-2
dparm[4]	Schur complement 常数的指定误差（droptol）: 默认值 1e-2
dparm[5]	Schur complement 常数的指定误差（droptol）: 默认值 1e-2

Table 11.5: 例子 11.4 中， **dparams** 各变量的意义

iparm[0] 的取值	Krylov 子空间方法
0	FGMRES (Flexible GMRES)
1	DGMRES (Deflated GMRES)
2	BICGSTAB

Table 11.6: pARMS 中的 Krylov 求解器

iparm[1]的值	预处理子
0	预处理子类型为 局部预处理子为*ilu0* 的可加 *Schwartz* 预处理子
1	预处理子类型为 局部预处理子为*iluk* 的可加 *Schwartz* 预处理子
2	预处理子类型为 局部预处理子为*ilut* 的可加 *Schwartz* 预处理子
3	预处理子类型为 局部预处理子为*arms* 的可加 *Schwartz* 预处理子
4	预处理子类型为 局部预处理子为*ilu0* 的左 *Schur* 补
5	预处理子类型为 局部预处理子为*ilut* 的左 *Schur* 补
6	预处理子类型为 局部预处理子为*iluk* 的左 *Schur* 补
7	预处理子类型为 局部预处理子为*arms* 的左 *Schur* 补
8	预处理子类型为 局部预处理子为*ilu0* 的右 *Schur* 补
9	预处理子类型为 局部预处理子为*ilut* 的右 *Schur* 补
10	预处理子类型为 局部预处理子为*iluk* 的右 *Schur* 补
11	预处理子类型为 局部预处理子为*arms* 的右 *Schur* 补
12	预处理子类型为 *sch_gilu0* ,全局 ilu0 的Schur 补预处理子
13	预处理子类型为 *SchurSymmetric GS* 预处理子

Table 11.7: pARMS中的预处理子

n= 471281	**nnz**=13×10^6		**Te=571,29**	
np	add(iluk)		schur(iluk)	
	nit	time	nit	time
4	230	637.57	21	557.8
8	240	364.12	22	302.25
16	247	212.07	24	167.5
32	261	111.16	25	81.5

Table 11.8: 例11.4 线性方程的收敛性与求解时间

n	最大值
nnz	矩阵非空项的数量
nit	收敛的迭代次数
time	收敛的时间
Te	构造有限元矩阵的时间
np	处理器数量

Table 11.9: 表11.8的示例

$HIPS\_DIR$:	HIPS的路径
$HIPS\_INCLUDE$:	-I$($$HIPS\_DIR$$)$/SRC/INCLUDE : HIPS头文件路径
LIB_DIR :	-L$($$HIPS\_DIR$$)$/LIB : 函数库路径
$LIBHIPSSEQUENTIAL$:	$($$HIPS_DIR$$)$/LIB/libhipssequential.a: HIPS 公共函数库
$LIBHIPS$:	$($$HIPS_DIR$$)$/LIB/libhips.a: HIPS函数库
$FREEFEM$:	FreeFem++ 路径
$FREEFEM\_INCLUDE$:	FreeFem 稀疏线性求解器头文件
$METIS$:	METIS 路径
$METIS\_LIB$:	METIS 函数库
MPI :	MPI 路径
$MPI\_INCLUDE$:	MPI 头文件

指定所有这些参数之后，在命令行中 *src/solver* 路径下输入 **make hips** 来新建文件 *hips_FreeFem.so*。

和 pARMS类似, HIPS在FreeFem++中的调用有三种不同方式。 接下来我们只通过一个例子来介绍如何通过关键字 `lparams` 和 `dparams`来指定参数。

用HIPS 求三维Laplacian算子 我们来看FreeFem++压缩包中的三维Laplacian例子，用HIPS解离散化线性方程。 例11.5 是使用HIPS作为线性求解器。代码中，第二行调用HIPS求解器， 在第4 – 15行指定HIPS求解器的参数，第46行设定线性求解器的参数。

Table 11.10是用 Grid5000 的 Cluster Paradent 运行例 11.5的结果。我们来看运算结果。

Example 11.5 (Laplacian3D.edp)
```
1: load "msh3"
2: load "hips_FreeFem"                                          //   加载库
3: int nn=10,iii;
4: int[int] iparm(14);
5: real[int] dparm(6);
6: for(iii=0;iii<14;iii++)iparm[iii]=-1;
7: for(iii=0;iii<6;iii++) dparm[iii]=-1;
8: iparm[0]=0;                                                  //    使用迭代求解
9: iparm[1]=1;                                          //   PCG 作为 Krylov 方法
10:iparm[4]=0;                                                  //   矩阵是对称的
11:iparm[5]=1;                                                  //   模式也是对称的
12: iparm[9]=1;                                                 //   尺度变换的矩阵
13:dparm[0]=1e-13;                                                 //    收敛精度
14: dparm[1]=5e-4;                                              //    ILUT 的阈值
15: dparm[2]=5e-4;                                      //    Schur 预条件子的阈值
16: mesh Th2=square(nn,nn);
17: fespace Vh2(Th2,P2);
18: Vh2 ux,uz,p2;
19: int[int] rup=[0,2],  rdown=[0,1], rmid=[1,1,2,1,3,1,4,1];
```

$n = 4 \times 10^6$	$nnz = 118 \times 10^6$	**Te=221.34**
np	nit	time
8	190	120.34
16	189	61.08
32	186	31.70
64	183	23.44

Table 11.10: 解例11.5线性系统的迭代与用时情况

```
20:real zmin=0,zmax=1;
21: mesh3 Th=buildlayers(Th2,nn,
  zbound=[zmin,zmax],
  reffacemid=rmid,
  reffaceup = rup,
  reffacelow = rdown);
22: savemesh(Th,"copie.mesh");
23: mesh3 Th3("copie.mesh");
24: fespace Vh(Th,P2);
25: func ue =   2*x*x + 3*y*y + 4*z*z + 5*x*y+6*x*z+1;
26: func uex=   4*x+  5*y+6*z;
27: func uey=   6*y + 5*x;
28: func uez=   8*z +6*x;
29: func f= -18. ;
30: Vh uhe = ue;                                                    //
31: cout << " uhe min:  " << uhe[].min << " max:" << uhe[].max << endl;
32: Vh u,v;
33: macro Grad3(u) [dx(u),dy(u),dz(u)]                              //    EOM
34: varf va(u,v)= int3d(Th)(Grad3(v)' *Grad3(u))                    //   ') emacs
         + int2d(Th,2)(u*v)
         - int3d(Th)(f*v)
         - int2d(Th,2) ( ue*v + (uex*N.x +uey*N.y +uez*N.z)*v )
         + on(1,u=ue);
35: real cpu=clock();
36: matrix Aa;
37: Aa=va(Vh,Vh);
38: varf l(unused,v)=int3d(Th)(f*v);
39: Vh F; F[]=va(0,Vh);
40: if(mpirank==0){
         cout << "Taille " << Aa.n << endl;
         cout << "Non zeros " << Aa.nbcoef << endl;
}
41:   if(mpirank==0)
42:    cout << "CPU TIME FOR FORMING MATRIX = " << clock()-cpu << endl;
43:    set(Aa,solver=sparsesolver,dparams=dparm, lparams=iparm);      //   设定
hips as 为线性求解器
44:    u[]=Aa^-1*F[];
```

表11.10 的示例在表11.9中给出。

iparm各项	**每项含义**
iparm[0]	求解策略 (Iterative=0 , Hybrid=1 , Direct=2). 默认值反复迭代
iparm[1]	Krylov 方法. 如果 iparm[0]=0,使用 Krylov 方法: GMRES：0, PCG：1
iparm[2]	最大外层迭代次数: 默认值 1000
iparm[3]	外层迭代中 Krylov 子空间维数: 默认值 40
iparm[4]	0 (对称矩阵) , 1 (非对称矩阵) : 默认值 1(非对称矩阵)
iparm[5]	矩阵是否对称: 默认值 0
iparm[6]	输入矩阵的划分: 默认值 0
iparm[7]	使用HIPS局部相容填充的水平个数: 默认值 2
iparm[8]	指标数组中的编号将从0或1开始: 默认值 0
iparm[9]	矩阵规模. 默认值 1
iparm[10]	重新用子域缩小最小填充: 只对可重复迭代的使用. 默认值 1
iparm[11]	矩阵非零元模式图中每个节点的未知量个数: 默认值 1
iparm[12]	设置矩阵标准化的时间: 默认值 2.
iparm[13]	求解过程的结果输出水平(LEVEL): 默认值 5.
iparm[14]	HIPS_DOMSIZE 子域大小

Table 11.11: **lparams** 与 HIPS 联合使用的各项含义

dparm[0]	$HIPS\_PREC$: 精度: 默认值=1e-9
dparm[1]	$HIPS_DROPTOL$0: ILUT中内域的数值阈值 (重要 : HYBRID 设置值 0.0: 默认值=0.005)
dparm[2]	$HIPS_DROPTOL$1 : Schur 预处理子 ILUT中数值阈值: 默认值=0.005
dparm[3]	$HIPS\_DROPTOLE$: 内部水平和Schur之间 耦合的数值阈值: 默认值 0.005
dparm[4]	$HIPS\_AMALG$: 内部水平和Schur之间 耦合的数值阈值: 默认值 0.005
dparm[5]	$HIPS\_DROPSCHUR$: 内部水平和Schur之间 耦合的数值阈值: 默认值 0.005

Table 11.12: **dparams** 与 HIPS 联合使用的各项含义

11.3.3 HYPRE

HYPRE (*High Level Preconditioner*)是一套由Lawrence Livermore National Lab [?] 开发的并行预处理方法。
HYPRE中分两类主要预处理方法: AMG (Algebraic MultiGrid) 和 Parasails (Parallel Sparse Approximate Inverse).
现在我们来看如何解方程 $Ax = b$。AMG的核心是一系列矩阵A的递增粗化刻画。 令$\hat{x}$ 是x的估计, 考虑当 $r = b - A\hat{x}$时，解残差方程$Ae = r$ 来得到 e的值。 AMG的基本原则就是通过代数方法平滑误差。为了进一步平滑误差， 他们需要通过一个更小更易求解的缺陷方程（粗网格剩余方程）$A_c e_c = r_c$来描绘。 解出这个缺陷方程之后，原方程的解可以通过差值得到。 使用AMG的效果取决于粗化与差值方法的选择。
稀疏近似逆 通过一个稀疏矩阵M估计矩阵A的逆。 一种构造矩阵M的技术想法 是最小化矩阵$I - MA$的Frobenuis范数。 更多关于此预处理子技术参考[?].
HYPRE通过三种Krylov 子空间方法实现: GMRES, PCG 和 BiCGStab。

HYPRE的安装 首先在[?]下载HYPRE压缩包，解压后 打开HYPRE/src 目录，然后运行 **./configure** 。 最后输入 *make all* 来创建**libHYPRE.a**。

HYPRE配合FreeFem++使用 在 `FreeFem++` 调用HYPRE之前，必须先编译文件 *hypre_FreeFem.cpp* 来创建动态库*hypre_FreeFem.so*。 具体步骤打开目录 `FreeFem++` *src/solver*, 编辑文件 *makefile.inc* 来指定一下参数:

$HYPRE\_DIR$:	HYPRE 的路径
$HYPRE\_INCLUDE$ =	-I$($HYPRE_DIR$)src/hypre/include/ : HYPRE 的头文件路径
$HYPRE\_LIB$ =	-L$($HIPS_DIR$)/src/lib/ -lHYPRE : Hypre 函数库
$FREEFEM$:	`FreeFem++` 路径
$FREEFEM\_INCLUDE$:	FreeFem 中稀疏线性求解器的头文件
$METIS$:	METIS 路径
$METIS\_LIB$:	METIS 函数库
MPI :	MPI 路径
$MPI\_INCLUDE$:	MPI 头文件

和pARMS一样, 在 `FreeFem++` 中HIPS的调用也可以通过三种方式。 下面我们通过一个例子演示通过关键字`lparams` 和 `dparams`来指定参数。

HYPRE解Laplacian 3D 我们来用HYPRE来解`FreeFem++` 中的3D Laplacian问题离散化后的线性方程。 例11.6 是使用HYPRE作为线性求解器。 例11.6 与例11.5相同, 所以我们 只列出设置HYPRE参数的行。
我们首先从HYPRE第二行开始看，从第4行到第15行以及第43行为参数设定。
注意到HYPRE的参数含义与HIPS有所不同，HYPRE的**iparm** 和 **dparm** 参数含义在表 11.13 到11.17中给出。

表**??**中，是Cluster Paradent of Grid5000 上 例 11.6的结果。我们可以看到使用AMG作为 预处理方法时的并行效率。

Example 11.6 (Laplacian3D.edp)
```
1: load "msh3"
2: load "hipre_FreeFem"                                    //   加载库
3: int nn=10,iii;
4: int[int] iparm(20);
5: real[int] dparm(6);
```

```
6: for(iii=0;iii<20;iii++)iparm[iii]=-1;
7: for(iii=0;iii<6;iii++) dparm[iii]=-1;
8: iparm[0]=2;                                          //   PCG 作为 krylov 方法
9: iparm[1]=0;                              //   AMG 作为 预条件子 2:  如果 ParaSails
10:iparm[7]=7;                                                    //   插值
11:iparm[9]=6;                                              //   AMG 粗化类型
12: iparm[10]=1;                                             //   测度类型
13: iparm[16]=2;                                    //   加性 schwarz 作为光滑子
13:dparm[0]=1e-13;                                           //   收敛精度
14: dparm[1]=5e-4;                                                //   阈值
15: dparm[2]=5e-4;                                           //   截断因子
.
.
.
43:     set(Aa,solver=sparsesolver,dparams=dparm, lparams=iparm);
```

iparms[0]	判断求解器: 0: BiCGStab, 1: GMRES, 2: PCG. **默认值=1**
iparms[1]	判断预处理子: 0: BOOMER AMG, 1: PILUT, 2: Parasails, 3: Schwartz **默认值=0**
iparms[2]	最大迭代次数: **默认值=1000**
iparms[3]	Krylov 子空间维数: **默认值= 40**
iparms[4]	求解器输出信息量: **默认值t=2**
iparms[5]	求解器日志 : **默认值=1**
iparms[6]	求解器使用BiCGStab 的中止条件 : **默认值=1**
dparms[0]	收敛的精度 : **默认值**$= 1.0e-11$

Table 11.13: HYPRE中**iparms** 和 **dparms**的各个参数含义

11.3.4 总结

对比不同的结果，我们看到伴随处理器数量的增加，计算时间也增加。 每个例子中有限元的构造用时是固定不变的。这很正常，原因是FreeFem++ 中这一步计算是逐步进行的。 相反的，求解线性系统的过程是并行的，从几个例子中都可以看出伴随处理器数量的增加，计算时间通常会缩短。 但不是每个例子中都是如此。在一些例子中，处理器数量增加同时计算时间也增加。 有两个主要原因：一方面，处理器的增加带来数据交换空间的增大，导致线性系统求解时间增长；另一方面， 包含找子域类的预处理方法，处理器的增加导致子域数量的增加，致使收敛的质量下降，求解时间增长。
最后，对于一些类型的问题我们很难判断一个预处理方法是否优秀，因而如何将预处理方法与FreeFem++ 合理搭 配使用更多的依赖于经验。同时，参数的选择有时也会对预处理的效率产生较大影响。因此，建议读者关注FreeFem++ 用户手册中各参数的含义。

11.4 区域分解

前一节我们了解了矩阵构造的步骤，我们看到构造矩阵的过程是有序自然的。构造并行矩阵的 一种策略是几何分割域成一些子域。每个子域中我们构造一个局部子域然后聚合各个子域来构造 全局矩阵。

iparms[7]	AMG 插值类型: **默认值=6**
iparms[8]	指定 GSMG 的几何光滑粗化和插值用途: **默认值=1**
iparms[9]	AMG 粗化类型: **默认值=6**
iparms[10]	选择使用局部测度还是全局测度: **默认值=1**
iparms[11]	AMG 循环类型: **默认值=1**
iparms[12]	AMG 光滑类型: **默认值=1**
iparms[13]	光滑子水平的AMG 数: **默认值=3**
iparms[14]	光滑子扫描的AMG 数: **默认值=2**
iparms[15]	多重网格最大值:**默认值=25**
iparms[16]	选择用哪种 Schwartz 方法: 0: 杂交乘性 Schwartz 方法 (处理器无重叠边界) 1: 杂交加性 Schwartz 方法 (处理器无重叠边界) 2: 加性 Schwartz 方法 3: 杂交乘性 Schwartz 方法 (处理器有重叠边界) **默认值=1**
iparms[17]	PDEs系统的大小: **默认值=1**
iparms[18]	Schwarz 方法有重叠: **默认值=1**
iparms[19]	Schwarz 方法的域的类型 0: 每个点都是一个区域 1: 每个节点都是一个区域 (只对"系统"AMG感兴趣) 2: 每个域都是聚集产生 (默认值)
dparms[1]	AMG strength 阈值: **默认值=0.25**
dparms[2]	插值的截断因子: **默认值=1e-2**
dparms[3]	设置一个参数来修改矩阵的对角占优部分的 强度的定义: **默认值=0.9**
dparms[3]	选择一个可加 Schwartz 方法的光滑参数 **默认值=1.**

Table 11.14: 预处理方法为**BOOMER AMG**时， **iparms** 和 **dparms**各参数含义

iparms[7]	Parallel ILUT 的行数: **默认值=1000**
iparms[8]	最大迭代次数: **默认值=30**
dparms[1]	Parallel ILUT 的精度: **默认值=1e-5**

Table 11.15: 预处理方法为**PILUT**时， **iparms** 和 **dparms**各参数含义

iparms[7]	并行稀疏近似逆的水平个数: **默认值=1**
iparms[8]	ParaSails预处理子的对称参数: 0: 非对称 和/或 不确定问题, 以及非对称预处理子 1: SPD 问题, 和 SPD (因子) 预处理子 2: 非对称, 确定问题, 和 SPD (因子) 预处理子 **默认值=0**
dparms[1]	过滤器参数:用来过滤掉预处理子中小的非零参数 来提高预处理子效率: **默认值=0.1**
dparms[2]	阈值参数: **默认值=0.1**

Table 11.16: 预处理方法为**ParaSails**时， **iparms** 和 **dparms**各参数含义

iparms[7]	判断用何种Schwartz 方法: 0: 杂交乘性 Schwartz 方法 (处理器边界无重叠) 1: 杂交加性 Schwartz 方法 (处理器边界无重叠) 2: 加性 Schwartz 方法 3: 杂交乘性 Schwartz 方法 (处理器有重叠边界) **默认值=1**
iparms[8]	Schwarz 方法有重叠: **默认值=1**
iparms[9]	Schwarz 方法的域的类型 0: 每个点都是一个域 1: 每个节点都是一个域 (只对"系统"AMG感兴趣) 2: 每个域都是聚集产生 (默认值)

Table 11.17: 预处理方法为**Schwartz**时， **iparms** 和 **dparms**各参数含义

n=4×10^6	**nnz**=13×10^6	**Te=571,29**
np	AMG	
	nit	time
8	6	1491.83
16	5	708.49
32	4	296.22
64	4	145.64

Table 11.18: 例11.4结果

我们可以使用这种方法直接解决域Ω中的PDE问题。在这个例子中，每一个子域中都不得不 手动定义边界条件来构造每个子域中的相容方程。之后，在子域中解方程以及选择一个得到全局解的策略。

`FreeFem++` 辅助MPI的并行编程方面，用户必须向处理器合理分配计算，同时也需要了解不同的 `FreeFem++` 脚本。下面是一些打包好的MPI 函数。

11.4.1 通讯器和群

群

mpiGroup grpe(mpiGroup gp,$KN\_ < long >$): 用已知向量在已有群**gp**中建立 MPI_Group

通讯器

通讯器是一个抽象的MPI项目，它允许MPI使用者联系处理器群。通讯器可以是内部通讯器 (单个群)也可以是外部通讯器(两个通讯器之间)。通讯器初始默认为内部通讯器。

mpiComm cc(mpiComm comm, mpiGroup gp): 新建一个通讯器。 *comm* communicator(handle), *gp*群(*comm* (handle)的子群)。返回新通讯器。

mpiComm cc(mpiGroup gp): 和上面的构造一样，只是默认*comm* 是MPI_COMM_WORLD.

mpiComm cc(mpiComm comm, int color, int key):基于*colors* and *key* 新建通讯器。构造基于MPI_Comm_split routine of MPI.

mpiComm cc(MPIrank p,int key): 同上面的构造。这里*colors* 和 *comm* 定义在 *MPIrank*. 这个构造基于MPI_Comm_split routine of MPI.

Example 11.7 (commsplit.edp)
```
1: int color=mpiRank(comm)%2;
```

```
2: mpiComm ccc(processor(color,comm),0);
3: mpiComm qpp(comm,);
4: mpiComm cp(cc,color,0);
```

mpiComm cc(mpiComm comm, int high): 通过外部通讯器新建内部通讯器。 *comm* 外部通讯器，*high* 用来在新建通讯器时命令*comm* (logical)内的群。 构造基于MPI_Intercomm_merge routine of MPI.
mpiComm cc(MPIrank p1, MPIrank p2, int tag):通过两个内部通讯器构造一个外部通讯器。*p1* 定义局部内部通讯器，以及，当*p2*定义远程通讯器和远程leader(通常是 0) 的 peer_comm 排序时，将leader(通常是 0)的local_comm 排序。*tag* 是构造外部通讯器的信息标签。构造基于MPI_Intercomm_create.

Example 11.8 (merge.edp)
```
1: mpiComm comm,cc;
2: int color=mpiRank(comm)%2;
3: int rk=mpiRank(comm);
4: int size=mpiSize(comm);
4: cout << "Color values " << color << endl;
5: mpiComm ccc(processor((rk<size/2),comm),rk);
6: mpiComm cp(cc,color,0);
7: int rleader;
8: if (rk == 0) {  rleader = size/2; }
9: else if (rk == size/2) { rleader = 0;}
10: else { rleader = 3; }
11: mpiComm qqp(processor(0,ccc),processor(rleader,comm),12345);
12:int aaa=mpiSize(ccc);
13:cout << "number of processor" << aaa << endl;
```

11.4.2 MPI过程

在FreeFem++ 中，我们用**processor** 函数打包MPI过程(新建内部FreeFem++ 项目名为**MPIrank**)。意味着在FreeFem++ 脚本中不能使用**MPIrank**。
processor(int rk):保持**MPIrank**中过程的排序。排序在MPI_COMM_WORLD中。
processor(int rk, mpiComm cc) and processor(mpiComm cc,int rk) 过程排序在 communicator cc中。
processor(int rk, mpiComm cc) and processor(mpiComm cc,int rk) 过程排序在 communicator cc中。 **processorblock(int rk)** : 这个函数和textbfprocessor(int rk) 完全相同， 但在关闭通讯时使用。
processorblock(int rk, mpiComm cc) : 这个函数和**processor(int rk,mpiComm cc)**完全相同， 但使用的是一个同步点。

11.4.3 点对点通讯器

在FreeFem++ 中可以称MPI为点对点通信函数。
Send(processor(int rk,mpiComm cc),Data D) : 阻塞的发送数据D给通信域cc中秩为 rk的进程，其中数据D可以是:*int, real,complex , int[int], real[int],complex[int], Mesh, Mesh3, Matrix*。
Recv(processor(int rk,mpiComm cc),Data D): 接收来自通信域cc中秩rk的进程的数据D，其中数据D可以是:*int, real,complex , int[int], real[int],complex[int], Mesh, Mesh3, Matrix*，并应当与发送数据的格式一致。

Isend(processor(int rk,mpiComm cc),Data D):非阻塞的发送数据D给通信域cc中秩为rk的进程，其中数据D 可以是:*int, real,complex , int[int], real[int],complex[int], Mesh, Mesh3, Matrix*。
Recv(processor(int rk,mpiComm cc),Data D): 接收与发送一致的数据。

11.4.4 全局的操作

在FreeFem++ 中可以称MPI为全局的通信函数。
broadcast(processor(int rk,mpiComm cc),Data D):进程rk 将数据D广播到通信域cc中的所有进程中， 其中数据 D可以是： *int, real,complex , int[int], real[int],complex[int], Mesh, Mesh3, Matrix*。

broadcast(processor(int rk),Data D):进程rk将数据D广播到MPI_COMM_WORLD中的所有进程中， 其中数据D可以是： *int, real,complex , int[int], real[int],complex[int], Mesh, Mesh3, Matrix*。

mpiAlltoall(Data a,Data b):发送来自所有进程的数据a到所有进程，数据b为接收缓存。该操作在MPI_COMM_WORLD通信域中进行。

mpiAlltoall(Data a,Data b, mpiComm cc): 发送来自所有进程的数据a到所有进程，数据b为接收缓存。该操作在通信域cc中进行。

mpiGather(Data a,Data b,processor(mpiComm,int rk) :在某个进程组中收集数据a，秩为rk的进程得到来自通信域rk的数据。该函数类似于MPI_Gather。

mpiAllgather(Data a,Data b) ： 在所有进程中收集数据a，并将其分配给数据b，操作在通信域MPI_COMM_WORLD 内执行。 该函数类似于MPI_Allgather。

mpiAllgather(Data a,Data b, mpiComm cc) ： 在所有进程中收集数据a，并将其分配给数据b，操作在通信域cc内执行。 该函数类似于MPI_Allgather。

mpiScatter(Data a,Data b,processor(int rk, mpiComm cc)) ： 发送来自秩rk的进程中的数据a到所有其他的代表通信域*mpiComm cc*的进程组中。

mpiReduce(Data a,Data b,processor(int rk, mpiComm cc),MPI_Op op), ： 将所有进程的数据a缩减为秩rk的进程和通信域cc中单独的数据b。用于缩减的命令*MPI_Op op*，它可以实现：*mpiMAX*, *mpiMIN*, *mpiSUM*, *mpiPROD*, *mpiLAND*, *mpiLOR*, *mpiLXOR*, *mpiBAND*, *mpiBXOR*, *mpiMAXLOC*, *mpiMINLOC*。

注意，对于所有的全局操作，FreeFem++ 中只涉及int[int]和real[int]两种数据格式。

下面给出一个利用Schwartz区域分解算法求解Laplacian2d问题的例子。在该例子中，我们应用两层并行来求解正方形区域上的Laplacian2d问题，问题中只有很少的几个子区域并且在每一个子区域上应用并行稀疏求解器来求解局部问题。

Example 11.9 (schwarz.edp)
```
1:load "hypre_FreeFem";                    //   加载 Hypre 求解器
2:func bool AddLayers(mesh & Th,real[int] &ssd,int n,real[int] &unssd)
{
                                                  //    建立一个连续的函数 uussd (P1) :
                                                 //     ssd 在输入子域上的特征函数.
                                                                  //     使得 :
                                                  //     unssd = 1 当 ssd =1 时;
```

```
                                                    //    增加 n 层单元 (重叠的大小)
                                                    //    并且在层外 unssd = 0 ...
                                                //    --------------------------------
  fespace Vh(Th,P1);
  fespace Ph(Th,P0);
  Ph s;
  assert(ssd.n==Ph.ndof);
  assert(unssd.n==Vh.ndof);
  unssd=0;
  s[]= ssd;
                                                    //    plot(s,wait=1,fill=1);
  Vh u;
  varf vM(u,v)=int2d(Th,qforder=1)(u*v/area);
  matrix M=vM(Ph,Vh);

  for(int i=0;i<n;++i)
    {
      u[]= M*s[];
                                                          //    plot(u,wait=1);
      u = u>.1;
                                                          //    plot(u,wait=1);
      unssd+= u[];
      s[]= M'*u[];                                                  //   ';
      s = s >0.1;
    }
  unssd /= (n);
  u[]=unssd;
  ssd=s[];
  return true;
}
3: mpiComm   myComm;                            //    创造具有值MPI_COMM_WORLD的通讯器

4: int membershipKey,rank,size;                                 //    管理通讯器的变量
5: rank=mpiRank(myComm); size=mpiSize(myComm);     //    处理器的序号, 通讯器的大小
6: bool withmetis=1, RAS=0;                        //    分割网格时使用或不使用 metis
7: int sizeoverlaps=5;                                          //    重叠的大小
8: int withplot=1;
9: mesh Th=square(100,100);
10: int[int] chlab=[1,1  ,2,1  ,3,1  ,4,1  ];
11: Th=change(Th,refe=chlab);
12: int nn=2,mm=2, npart= nn*mm;
13: membershipKey = mpiRank(myComm)%npart;              //    给划分的处理器群着色
14: mpiComm cc(processor(membershipKey,myComm),rank); //   根据之前的着色创造 MPI
通讯器
15: fespace Ph(Th,P0),fespace Vh(Th,P1);
16: Ph  part;
17: Vh  sun=0,unssd=0;
18: real[int] vsum=sun[],reducesum=sun[];               //    用来控制划分的数据.
19: Ph xx=x,yy=y,nupp;
20: part = int(xx*nn)*mm + int(yy*mm);
21: if(withmetis)
    {
    load "metis";
    int[int] nupart(Th.nt);
    metisdual(nupart,Th,npart);
    for(int i=0;i<nupart.n;++i)
```

```
      part[][i]=nupart[i];
    }
22: if(withplot>1)
21: plot(part,fill=1,cmm="dual",wait=1);
22: mesh[int] aTh(npart);
23: mesh Thi=Th;
24: fespace Vhi(Thi,P1);
25: Vhi[int] au(npart),pun(npart);
26: matrix[int] Rih(npart), Dih(npart), aA(npart);
27: Vhi[int] auntgv(npart), rhsi(npart);
28: i=membershipKey;
    Ph suppi= abs(part-i)<0.1;
    AddLayers(Th,suppi[],sizeoverlaps,unssd[]);
    Thi=aTh[i]=trunc(Th,suppi>0,label=10,split=1);
    Rih[i]=interpolate(Vhi,Vh,inside=1);                    //    Vh -> Vhi
    if(RAS)
      {
        suppi= abs(part-i)<0.1;
        varf vSuppi(u,v)=int2d(Th,qforder=1)(suppi*v/area);
        unssd[]= vSuppi(0,Vh);
        unssd = unssd>0.;
        if(withplot>19)
          plot(unssd,wait=1);
      }
    pun[i][]=Rih[i]*unssd[];                                //   这是全局处理
    sun[] += Rih[i]'*pun[i][];                    //    也是类似于广播的全局处理;
    vsum=sun[];
    if(withplot>9)
      plot(part,aTh[i],fill=1,wait=1);
                          //    增加 mpireduce, 为了求和所有的 sun 和 pun 的局部贡献.
29: mpiReduce(vsum, reducesum,processor(0,myComm),mpiSUM);  //   MPI 全局处理 在
全局通讯器上 MPi_Reduce
30: broadcast(processor(0,myComm),reducesum);      //   处理器0到所有处理器的广播和
31: sun[]=reducesum;
32: plot(sun,wait=1);
33: i=membershipKey
34: Thi=aTh[i];
35: pun[i]= pun[i]/sun;
36: if(withplot>8)  plot(pun[i],wait=1);
37: macro Grad(u) [dx(u),dy(u)]                                 //   EOM
38: sun=0;
39: i=membershipKey
    Thi=aTh[i];
    varf va(u,v) =
      int2d(Thi)(Grad(u)'*Grad(v))                              //   ')
      +on(1,u=1) + int2d(Th)(v)
      +on(10,u=0) ;
40: aA[i]=va(Vhi,Vhi);
41: set(aA[i],solver=sparsesolver,mpicomm=cc);      //   为求解器 Hypre 设置参数.
mpicomm=cc 是指不是在全局处理器上求解, 而是在由cc定义的一组处理器上求解.
42: rhsi[i][]= va(0,Vhi);
43: Dih[i]=pun[i][];
44: real[int]  un(Vhi.ndof);
45: un=1.;
46: real[int] ui=Dih[i]*un;
47: sun[] += Rih[i]'*ui;                                        //   ';
```

```
48: varf vaun(u,v) = on(10,u=1);
49: auntgv[i][]=vaun(0,Vhi);                                  //    在Gamma上储存tgv的数组.
56: reducesum=0; vsum=sun;
57: mpiReduce(vsum, reducesum,processor(0,myComm),mpiSUM);  //   MPI 全局处理 在全局通讯器上 MPi_Reduce
58: broadcast(processor(0,myComm),reducesum);      //   处理器0到所有处理器的广播和
59: sun[]=reducesum;
60: if(withplot>5)
61: plot(sun,fill=1,wait=1);
62: cout << sun[].max << " " << sun[].min<< endl;
63: assert( 1.-1e-9 <= sun[].min  && 1.+1e-9 >= sun[].max);
64: int nitermax=1000;
{
  Vh un=0;
  for(int iter=0;iter<nitermax;++iter)
    {
      real err=0,rerr=0;
      Vh un1=0;
          i=membershipKey;
          Thi=aTh[i];
          real[int]  ui=Rih[i]*un[];                                          //   ';
  real[int] bi = ui .* auntgv[i][];
          bi = auntgv[i][] ? bi :  rhsi[i][];
          ui=au[i][];
          ui= aA[i] ^-1 * bi;   //   在由颜色 membershipKey 表示的处理器群上求解局部线性系统
          bi = ui-au[i][];
          err += bi'*bi;                                                      //   ';
  au[i][]= ui;
          bi = Dih[i]*ui;                          //   对现在的解的延拓, 以获得右端项
          un1[] += Rih[i]'*bi;                                                //   ';
}
65: reducesum=0; vsum=un1[];
66: mpiReduce(vsum, reducesum,processor(0,myComm),mpiSUM);  //   MPI 全局处理 在全局通讯器上 MPi_Reduce
67: broadcast(processor(0,myComm),reducesum);      //   处理器0到所有处理器的广播和
68: un1[]=reducesum;
69: real residrela=0;
70: mpiReduce(err,residrela ,processor(0,myComm),mpiSUM);
71: broadcast(processor(0,myComm),residrela);
72: err=residrela; err= sqrt(err);
73: if(rank==0) cout << iter << " Err = " << err << endl;
74:    if(err<1e-5) break;
75:      un[]=un1[];
76:        if(withplot>2)
77:        plot(au,dim=3,wait=0,cmm=" iter  "+iter,fill=1 );
78:    }
79:  plot(un,wait=1,dim=3);
80: }
```

第 12 章

网格文件

12.1 网格数据结构文件

由网格生成算法输出的网格数据结构，是几何的数据结构或者某些情况下表示其它的网格数据结构。
此时，域为：

- `MeshVersionFormatted 0`

- `Dimension (I)` dim
- `Vertices (I)` NbOfVertices
  $\Big(\big(\texttt{(R)}\ x_i^j \textbf{,}\ \text{j=1,dim} \big) ,\ \texttt{(I)}\ Ref\phi_i^v \textbf{,}\ \text{i=1 , NbOfVertices} \Big)$
- `Edges (I)` NbOfEdges
  $\Big(\texttt{@@Vertex}_i^1 , \texttt{@@Vertex}_i^2 ,\ \texttt{(I)}\ Ref\phi_i^e \textbf{,}\ \text{i=1 , NbOfEdges} \Big)$
- `Triangles (I)` NbOfTriangles
  $\Big(\big(\texttt{@@Vertex}_i^j \textbf{,}\ \text{j=1,3} \big) ,\ \texttt{(I)}\ Ref\phi_i^t \textbf{ ,}\ \text{i=1 , NbOfTriangles} \Big)$
- `Quadrilaterals (I)` NbOfQuadrilaterals
  $\Big(\big(\texttt{@@Vertex}_i^j \textbf{,}\ \text{j=1,4} \big) ,\ \texttt{(I)}\ Ref\phi_i^t \textbf{ ,}\ \text{i=1 , NbOfQuadrilaterals} \Big)$
- `Geometry`
  `(C*)` FileNameOfGeometricSupport

  - `VertexOnGeometricVertex`
    `(I)` NbOfVertexOnGeometricVertex
    $\Big(\texttt{@@Vertex}_i , \texttt{@@Vertex}_i^{geo} \textbf{,}\ \text{i=1,NbOfVertexOnGeometricVertex} \Big)$
  - `EdgeOnGeometricEdge`
    `(I)` NbOfEdgeOnGeometricEdge
    $\Big(\texttt{@@Edge}_i , \texttt{@@Edge}_i^{geo} \textbf{,}\ \text{i=1,NbOfEdgeOnGeometricEdge} \Big)$

- `CrackedEdges (I)` NbOfCrackedEdges
  $\Big(\texttt{@@Edge}_i^1 , \texttt{@@Edge}_i^2 \textbf{,}\ \text{i=1 , NbOfCrackedEdges} \Big)$

如果当前网格指的是先前的某个网格，我们还有：

- MeshSupportOfVertices
  (C*) FileNameOfMeshSupport

  - VertexOnSupportVertex
    (I) NbOfVertexOnSupportVertex
    $\left(\texttt{@@Vertex}_i\,,\texttt{@@Vertex}_i^{supp}\,,\ \text{i=1,NbOfVertexOnSupportVertex}\right)$
  - VertexOnSupportEdge
    (I) NbOfVertexOnSupportEdge
    $\left(\texttt{@@Vertex}_i\,,\texttt{@@Edge}_i^{supp},\ \texttt{(R)}\ u_i^{supp}\ ,\ \text{i=1,NbOfVertexOnSupportEdge}\right)$
  - VertexOnSupportTriangle
    (I) NbOfVertexOnSupportTriangle
    $\left(\texttt{@@Vertex}_i\,,\texttt{@@Tria}_i^{supp},\ \texttt{(R)}\ u_i^{supp},\ \texttt{(R)}\ v_i^{supp}\ ,\right.$
    $\left.\text{i=1 , NbOfVertexOnSupportTriangle}\right)$
  - VertexOnSupportQuadrilaterals
    (I) NbOfVertexOnSupportQuadrilaterals
    $\left(\texttt{@@Vertex}_i\,,\texttt{@@Quad}_i^{supp},\ \texttt{(R)}\ u_i^{supp},\ \texttt{(R)}\ v_i^{supp}\ ,\right.$
    $\left.\text{i=1 , NbOfVertexOnSupportQuadrilaterals}\right)$

12.2 存储解的bb文件类型

文件的格式如下：
2 nbsol nbv 2
$((\texttt{U}_{ij},\quad \forall i \in \{1,...,\texttt{nbsol}\})\,,\quad \forall j \in \{1,...,\texttt{nbv}\})$
其中

- nbsol是一个整数，表示解的个数。
- nbv是一个整数，表示顶点个数。
- $\texttt{U}_{ij}$是一个实数，表示读取文件时相关联的或者写文件时生成的网格上顶点j处第i个解的值。

12.3 存储解的BB文件类型

文件格式如下：
2 n $\texttt{typesol}^1$ ... $\texttt{typesol}^n$ nbv 2
$\left(\left((\texttt{U}_{ij}^k,\quad \forall i \in \{1,...,\texttt{typesol}^k\}\right)\,,\quad \forall k \in \{1,...\texttt{n}\}\right)\quad \forall j \in \{1,...,\texttt{nbv}\}\right)$
其中

- n是一个整数，表示解的个数。
- $\texttt{typesol}^\texttt{k}$, 解k的类型编号

  - typesol$^{\mathtt{k}} = 1$ 解 k 是标量 (1 value per vertex)
  - typesol$^{\mathtt{k}} = 2$ 解 k 是向量 (2 values per unknown)
  - typesol$^{\mathtt{k}} = 3$ 解 k 是一个2×2对称矩阵 (3 values per vertex)
  - typesol$^{\mathtt{k}} = 4$ 解 k 是一个2×2 矩阵 (4 values per vertex)

- nbv是一个整数，表示顶点的个数。
- $\mathtt{U}_{ij}^{k}$是一个实数，表示读取文件时相关联的或者写文件时生成的网格上顶点j 处解k第i部分的值。

12.4 度量文件

度量文件可分为各向同性与各向异性两种。
各向同性文件形如
nbv 1
$\mathtt{h}_i \quad \forall i \in \{1, ..., \mathtt{nbv}\}$
其中

- nbv 是一个整数，等于顶点的数量。
- $\mathtt{h}_i$ 是背景网格的顶点 i 附近要求的网格大小， 度量为 $\mathcal{M}_i = h_i^{-2} Id$, 其中 Id 是单位矩阵。

各向异性文件形如
nbv 3
$\mathtt{a11}_i, \mathtt{a21}_i, \mathtt{a22}_i \quad \forall i \in \{1, ..., \mathtt{nbv}\}$
其中

- nbv 是一个整数，等于顶点的数量，
- $\mathrm{a11}_i$, $\mathrm{a12}_i$, $\mathrm{a22}_i$ 构成度量 $\mathcal{M}_i = \left(\begin{smallmatrix} a11_i & a12_i \\ a12_i & a22_i \end{smallmatrix}\right)$ 它定义了在顶点 i 附近要求的网格大小 h ，方向为 $u \in \mathbb{R}^2$ ，值的大小为 $|u|/\sqrt{u \cdot \mathcal{M}_i\, u}$, 其中 $\cdot$ 是 $\mathbb{R}^2$ 上的点乘, $|\cdot|$ 为经典定义。

12.5 AM_FMT, AMDBA 网格的列表

网格仅由三角形组成，并可借助下面的两个整数和四个数组定义:

nbt	三角形的数量。
nbv	顶点的数量。
nu(1:3,1:nbt)	一个整数数组，给出每个三角形三个顶点逆时针方向的编号。
c(1:2,nbv)	一个实数数组，给出每个顶点的两个坐标。
refs(nbv)	一个整数数组，给出顶点的编号。
reft(nbv)	一个整数数组，给出三角形的编号。

AM_FMT Files fortran 中 am_fmt 文件按如下读取:

```
open(1,file='xxx.am_fmt',form='formatted',status='old')
  read (1,*) nbv,nbt
  read (1,*)  ((nu(i,j),i=1,3),j=1,nbt)
  read (1,*)  ((c(i,j),i=1,2),j=1,nbv)
  read (1,*)  ( reft(i),i=1,nbt)
  read (1,*)  ( refs(i),i=1,nbv)
close(1)
```

AM Files fortran 中 am 文件按如下读取:

```
open(1,file='xxx.am',form='unformatted',status='old')
  read (1,*) nbv,nbt
  read (1)  ((nu(i,j),i=1,3),j=1,nbt),
&   ((c(i,j),i=1,2),j=1,nbv),
&   ( reft(i),i=1,nbt),
&   ( refs(i),i=1,nbv)
close(1)
```

AMDBA Files fortran 中 amdba 文件按如下读取:

```
open(1,file='xxx.amdba',form='formatted',status='old')
  read (1,*) nbv,nbt
  read (1,*) (k,(c(i,k),i=1,2),refs(k),j=1,nbv)
  read (1,*) (k,(nu(i,k),i=1,3),reft(k),j=1,nbt)
close(1)
```

msh Files 首先，我们引入边界的概念

nbbe 是边界的数量。

nube(1:2,1:nbbe) 是一个整数数组，给出每条边界两个顶点逆时针方向的编号。

refbe(1:nbbe) 是一个整数数组，给出边界的编号。

fortran 中 msh 文件按如下读取:

```
open(1,file='xxx.msh',form='formatted',status='old')
  read (1,*) nbv,nbt,nbbe
  read (1,*) ((c(i,k),i=1,2),refs(k),j=1,nbv)
  read (1,*) ((nu(i,k),i=1,3),reft(k),j=1,nbt)
  read (1,*) ((ne(i,k),i=1,2), refbe(k),j=1,nbbe)
close(1)
```

ftq Files fortran 中 `ftq` 文件按如下读取:

```
open(1,file='xxx.ftq',form='formatted',status='old')
 read (1,*) nbv,nbe,nbt,nbq
 read (1,*) (k(j),(nu(i,j),i=1,k(j)),reft(j),j=1,nbe)
 read (1,*) ((c(i,k),i=1,2),refs(k),j=1,nbv)
close(1)
```

其中 如果 `k(j) = 3` 那么元 j 是一个三角形，而如果 `k = 4` 那么元 j 是一个四边形。

第 13 章

加入一个新有限元

13.1 一些记号

函数 $\boldsymbol{f}$ 取值于 $\mathbb{R}^N$, $N = 1, 2, \cdots$, 我们定义 $\boldsymbol{f}$ 的有限元近似 $\Pi_h \boldsymbol{f}$ 。让我们用 $NbDoF$ 来表示有限元的自由度次数。那么有限元空间的第 i 个基 $\boldsymbol{\omega}_i^K$ $(i = 0, \cdots, NbDoF - 1)$ 的第 j 个元素为 ω_{ij}^K ，对 $j = 0, \cdots, N - 1$。
运算符 Π_h 称为有限元的插值。 我们有恒等式 $\boldsymbol{\omega}_i^K = \Pi_h \boldsymbol{\omega}_i^K$。
形式上，插值 Π_h 可由下列公式表示:

$$\Pi_h \boldsymbol{f} = \sum_{k=0}^{\texttt{kPi}-1} \alpha_k \boldsymbol{f}_{j_k}(P_{p_k}) \boldsymbol{\omega}_{i_k}^K \tag{13.1}$$

其中 P_p 是 $npPi$ 点的集合,
在公式 (13.1) 中， 编号 p_k, j_k, i_k 只和有限元的类型 (而非有限元本身)有关， 但系数 α_k 则可能和有限元相关。

例 1: 对经典的标量拉格朗日有限元， 我们有 `kPi` = `npPi` = `NbOfNode` 和

- P_p 表示结点
- $\alpha_k = 1$, 因为我们在点 P_k 上对函数取值
- $p_k = k$, $j_k = k$ 因为每个函数有一个节点
- $j_k = 0$ 因为 $N = 1$

例 2: Raviart-Thomas 有限元:

$$RT0_h = \{\mathbf{v} \in H(div) / \forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K}(x, y) = \left| \begin{smallmatrix} \alpha_K \\ \beta_K \end{smallmatrix} \right. + \gamma_K \left| \begin{smallmatrix} x \\ y \end{smallmatrix} \right. \} \tag{13.2}$$

自由度次数是通过网格的边 e 的流量, 其中函数 $\mathbf{f} : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ 的流量是 $\int_e \mathbf{f}.n_e$， n_e 是边界 e 的单位法线(它表示了网格所有的边的一个方向, 例如当我们对边和顶点进行全局编号时，我们只是将编号由小变大而已)。
计算这个流量，我们要用一个求积分公式并借助一个点，即这条边的中点。 考虑一个三角形 T 它有三个顶点 $(\mathbf{a}, \mathbf{b}, \mathbf{c})$。 我们以 i_a, i_b, i_c 表示顶点编号, 并定义三条边的向量 $\mathbf{e}^0, \mathbf{e}^1, \mathbf{e}^2$ 分别为 $sgn(i_b - i_c)(\mathbf{b} - \mathbf{c})$, $sgn(i_c - i_a)(\mathbf{c} - \mathbf{a})$, $sgn(i_a - i_b)(\mathbf{a} - \mathbf{b})$,

三个基函数为:

$$\boldsymbol{\omega}_0^K = \frac{sgn(i_b - i_c)}{2|T|}(x-a), \quad \boldsymbol{\omega}_1^K = \frac{sgn(i_c - i_a)}{2|T|}(x-b), \quad \boldsymbol{\omega}_2^K = \frac{sgn(i_a - i_b)}{2|T|}(x-c), \tag{13.3}$$

其中 $|T|$ 是三角形 T 的面积。
那么我们有 $N = 2$, `kPi` $= 6$; `npPi` $= 3$; 和:

- $P_p = \left\{\frac{\mathbf{b}+\mathbf{c}}{2}, \frac{\mathbf{a}+\mathbf{c}}{2}, \frac{\mathbf{b}+\mathbf{a}}{2}\right\}$
- $\alpha_0 = -\mathbf{e}_2^0, \alpha_1 = \mathbf{e}_1^0, \alpha_2 = -\mathbf{e}_2^1, \alpha_3 = \mathbf{e}_1^1, \alpha_4 = -\mathbf{e}_2^2, \alpha_5 = \mathbf{e}_1^2$ (实际上，向量 $(-\mathbf{e}_2^m, \mathbf{e}_1^m)$ 与边 $\mathbf{e}^m = (e_1^m, e_2^m)$ 正交，长度与其相等或等于 $\int_{e^m} 1$)。
- $i_k = \{0, 0, 1, 1, 2, 2\}$,
- $p_k = \{0, 0, 1, 1, 2, 2\}$, $j_k = \{0, 1, 0, 1, 0, 1, 0, 1\}$.

13.2 要加些什么类?

在目录 `src/femlib` 下增加文件`FE_ADD.cpp`，为第一个例子初始化：

```
#include "error.hpp"
#include "rgraph.hpp"
using namespace std;
#include "RNM.hpp"
#include "fem.hpp"
#include "FESpace.hpp"
#include "AddNewFE.h"

namespace  Fem2D {
```

然后增加一个类 `public TypeOfFE` 如下:

```
class TypeOfFE_RTortho : public  TypeOfFE { public:
  static int Data[]; //    一些编号
  TypeOfFE_RTortho():
    TypeOfFE( 0+3+0,    //     元自由度的次数
       2,       //     向量有限元的维数 N (1 如果是标量有限元)
       Data,    //     数组数据
       1,       //     作图的子分支数
       1,       //     子有限元个数 (一般为 1)
       6,       //     kPi 构建插值函数的系数的个数 (13.1)
       3,       //     npPi 构建插值的积分点个数
       0        //     储存构建插值函数的系数 α_k 的数组
                //     这里这个数组不是常数，因此我们需要
                //     对每个元分别进行重建。
       )
  {
    const R2 Pt[] = { R2(0.5,0.5), R2(0.0,0.5), R2(0.5,0.0) };
                                                          //     K̂ 中点的集合
    for (int p=0,kk=0;p<3;p++) {
      P_Pi_h[p]=Pt[p];
      for (int j=0;j<2;j++)
```

```
        pij_alpha[kk++]= IPJ(p,p,j); }}              //     (13.1)中 i_k,p_k,j_k 的定义

  void FB(const bool * watdd, const Mesh & Th,const Triangle & K,
          const R2 &PHat, RNMK_ & val) const;

  void Pi_h_alpha(const baseFElement & K,KN_<double> & v) const ;

} ;
```

其中数组数据形式上由五个长度 NbDoF 的数组 和一个 长度 N 的数组构成。
这个数组是:

```
int TypeOfFE_RTortho::Data[]={

              //     对每个 df 0,1,3 :
        3,4,5,//     df的节点的支撑
        0,0,0,//     节点上 df 的编号
        0,1,2,//     df 的节点
        0,0,0,//     有限元的 df (一般为 0)
        0,1,2,//     子有限元的 de df
        0,0 };                                  //     每个元素 j = 0, N − 1 都与子有限元有关
```

其中支撑是编号 $0,1,2$ 是支撑顶点， $3,4,5$ 是支撑边， 而 6 是支撑元。
定义函数 $\boldsymbol{\omega}_i^K$ 的函数, 返回所有基函数的值或数组 val 的导数 ， 计算则在与参考的三角形的对应点 R2 P=K(Phat) 相对应的当前三角形 K 的点 PHat 上进行。
数组 $val(i,j,k)$ 的指数 i,j,k 对应于:

i 有限元的基函数的编号 $i \in [0, NoF[$

j 元素 $j \in [0,N]$ 的值

k 要计算的值 $f(P), dx(f)(P), dy(f)(P), \ldots$ $i \in [0,$last_operatortype$]$ 的类型。对于最优化问题, 只有当 $whatd[k]$ 为真时，值才进行计算， 并且编号定义方式为

```
enum operatortype { op_id=0,
   op_dx=1,op_dy=2,
   op_dxx=3,op_dyy=4,
   op_dyx=5,op_dxy=5,
   op_dz=6,
   op_dzz=7,
   op_dzx=8,op_dxz=8,
   op_dzy=9,op_dyz=9
   };
const int last_operatortype=10;
```

形状函数 :

```
 void TypeOfFE_RTortho::FB(const bool *whatd,const Mesh & Th,const Triangle & K,
                           const R2 & PHat,RNMK_ & val) const
{                                                                          //
  R2 P(K(PHat));
  R2 A(K[0]), B(K[1]),C(K[2]);
  R l0=1-P.x-P.y,l1=P.x,l2=P.y;
  assert(val.N() >=3);
  assert(val.M()==2 );
```

```
  val=0;
  R a=1./(2*K.area);
  R a0=   K.EdgeOrientation(0) * a ;
  R a1=   K.EdgeOrientation(1) * a  ;
  R a2=   K.EdgeOrientation(2) * a ;

                                                          //    ------------
  if (whatd[op_id])                                       //    函数的值
   {
     assert(val.K()>op_id);
     RN_ f0(val('.',0,0));                                //    求第一个元素的值
     RN_ f1(val('.',1,0));                                //    求第二个元素的值
     f1[0] =  (P.x-A.x)*a0;
     f0[0] = -(P.y-A.y)*a0;

     f1[1] =  (P.x-B.x)*a1;
     f0[1] = -(P.y-B.y)*a1;

     f1[2] =  (P.x-C.x)*a2;
     f0[2] = -(P.y-C.y)*a2;
    }
                                                          //    ----------------
    if (whatd[op_dx])                                     //    函数的 dx 的值
    {
     assert(val.K()>op_dx);
     val(0,1,op_dx) =  a0;
     val(1,1,op_dx) =  a1;
     val(2,1,op_dx) =  a2;
     }
    if (whatd[op_dy])
    {
     assert(val.K()>op_dy);
     val(0,0,op_dy) =  -a0;
     val(1,0,op_dy) =  -a1;
     val(2,0,op_dy) =  -a2;
    }

  for (int i= op_dy; i< last_operatortype ; i++)
   if (whatd[op_dx])
     assert(op_dy);

}
```

定义系数 α_k 的函数:

```
void TypeOfFE_RT::Pi_h_alpha(const baseFElement & K,KN_<double> & v) const
{
  const Triangle & T(K.T);

   for (int i=0,k=0;i<3;i++)
     {
        R2 E(T.Edge(i));
        R signe = T.EdgeOrientation(i) ;
        v[k++]= signe*E.y;
        v[k++]=-signe*E.x;
     }
```

```
}
```

现在，我们只需在 FreeFem++ 中再做一个关键的工作。 有两种方式，静态链接或动态链接。因此在文件最后，我们加上 :

动态链接非常简单 (见附录章节 C), 只用在 FEM2d namespace 末尾加上:

```
static TypeOfFE_RTortho The_TypeOfFE_RTortho;                                    //
static AddNewFE("RT0Ortho", The_TypeOfFE_RTortho);
}                                                                  //    FEM2d namespace
```

试着在 examples++-load/ 中使用 "./load.link" 命令， 可阅览 BernardiRaugel.cpp 或 Morley.cpp 以获得新的有限元例子。

否则 对静态链接 (仅适用于专业人士)， 加上

```
                                                                   //    假设2个全局变量
static TypeOfFE_RTortho The_TypeOfFE_RTortho;                                    //
                                                        //    ----- freefem 中的名字----
static  ListOfTFE typefemRTOrtho("RT0Ortho", & The_TypeOfFE_RTortho);    //

//    静态库在链接 FreeFem++ 时无效
//    FH 因此加入一个外部名字以调用 init_static_FE
//    (看 FESpace.cpp 的末尾)
void init_FE_ADD() { };
//    --- 结束 --
}                                                                  //    FEM2d namespace
```

要成功读取这个新有限元, 我们需要在 src/femlib/FESpace.cpp 文件末尾加上两行:

```
                                                       //    修正新制作文件的静态库链接的问题
void init_static_FE()
{                                                                  //    其他有限元文件的列表
   extern void init_FE_P2h() ;
  init_FE_P2h() ;
   extern void init_FE_ADD() ;                                              //    新加行 1
   init_FE_ADD();                                                           //    新加行 2
}
```

现在你必须改变makefile文件.
首先，新建一个 FE_ADD.cpp 文件包含所有这些代码， 参照文件 src/femlib/Element_P2h.cpp
在变量 EXTRA_DIST 中加入你的文件名来修改 Makefile.am 如下:

```
# Makefile using Automake + Autoconf
# ----------------------------------
# $Id$

# This is not compiled as a separate library because its
# interconnections with other libraries have not been solved.

EXTRA_DIST=BamgFreeFem.cpp BamgFreeFem.hpp CGNL.hpp CheckPtr.cpp          \
```

```
ConjuguedGradrientNL.cpp DOperator.hpp Drawing.cpp Element_P2h.cpp      \
Element_P3.cpp Element_RT.cpp fem3.hpp fem.cpp fem.hpp FESpace.cpp      \
FESpace.hpp FESpace-v0.cpp FQuadTree.cpp FQuadTree.hpp gibbs.cpp        \
glutdraw.cpp gmres.hpp MatriceCreuse.hpp MatriceCreuse_tpl.hpp          \
MeshPoint.hpp mortar.cpp mshptg.cpp QuadratureFormular.cpp              \
QuadratureFormular.hpp RefCounter.hpp RNM.hpp RNM_opc.hpp RNM_op.hpp    \
RNM_tpl.hpp   FE_ADD.cpp
```

并在 freefem++ 根目录下执行

```
  autoreconf
 ./reconfigure
 make
```

用 codewarrior 编译时，需在项目中加入文件, 并且消去面板 PPC链接器中的信号旗，FreeFEm++ Setting Dead-strip Static Initializaiton— Code Flag

第 A 章

Table of Notations

Here mathematical expressions and corresponding `FreeFem++` commands are explained.

A.1 Generalities

δ_{ij} Kronecker delta (0 if $i \neq j$, 1 if $i = j$ for integers i, j)

$\forall$ for all

$\exists$ there exist

i.e. that is

PDE partial differential equation (with boundary conditions)

$\emptyset$ the empty set

$\mathbb{N}$ the set of integers ($a \in \mathbb{N} \Leftrightarrow$ `int a`); "`int`" means *long integer* inside `FreeFem++`

$\mathbb{R}$ the set of real numbers ($a \in \mathbb{R} \Leftrightarrow$ `real a`) ;*double* inside `FreeFem++`

$\mathbb{C}$ the set of complex numbers ($a \in \mathbb{C} \Leftrightarrow$ `complex a`); *complex¡double¿*

$\mathbb{R}^d$ d-dimensional Euclidean space

A.2 Sets, Mappings, Matrices, Vectors

Let E, F, G be three sets and A subset of E.

$\{x \in E|\ P\}$ the subset of E consisting of the elements possessing the property P

$E \cup F$ the set of elements belonging to E or F

$E \cap F$ the set of elements belonging to E and F

$E \setminus A$ the set $\{x \in E|\ x \notin A\}$

$E + F$ $E \cup F$ with $E \cap F = \emptyset$

$E \times F$ the cartesian product of E and F

E^n the n-th power of E ($E^2 = E \times E$, $E^n = E \times E^{n-1}$)

$f : E \to F$ the mapping form E into F, i.e., $E \ni x \mapsto f(x) \in F$

I_E **or** I the identity mapping in E,i.e., $I(x) = x \quad \forall x \in E$

$f \circ g$ for $f : F \to G$ and $g : E \to F$, $E \ni x \mapsto (f \circ g)(x) = f(g(x)) \in G$ (see **??**)

$f|_A$ the restriction of $f : E \to F$ to the subset A of E

$\{a_k\}$ column vector with components a_k

(a_k) row vector with components a_k

$(a_k)^T$ denotes the transpose of a matrix (a_k), and is $\{a_k\}$

$\{a_{ij}\}$ matrix with components a_{ij}, and $(a_{ij})^T = (a_{ji})$

A.3 Numbers

For two real numbers a, b

$[a, b]$ is the interval $\{x \in \mathbb{R} |\ a \leq x \leq b\}$

$]a, b]$ is the interval $\{x \in \mathbb{R} |\ a < x \leq b\}$

$[a, b[$ is the interval $\{x \in \mathbb{R} |\ a \leq x < b\}$

$]a, b[$ is the interval $\{x \in \mathbb{R} |\ a < x < b\}$

A.4 Differential Calculus

$\partial f/\partial x$ the partial derivative of $f : \mathbb{R}^d \to \mathbb{R}$ with respect to x (`dx(f)`)

∇f the gradient of $f : \Omega \to \mathbb{R}$,i.e., $\nabla f = (\partial f/\partial x, \partial f/\partial y)$

div $\boldsymbol{f}$ **or** $\nabla . \boldsymbol{f}$ the divergence of $\boldsymbol{f} : \Omega \to \mathbb{R}^d$, i.e., $\text{div}\boldsymbol{f} = \partial f_1/\partial x + \partial f_2/\partial y$

Δf the Laplacian of $f : \Omega \to \mathbb{R}$, i.e., $\Delta f = \partial^2 f/\partial x^2 + \partial^2 f/\partial y^2$

A.5 Meshes

Ω usually denotes a domain on which PDE is defined

Γ denotes the boundary of Ω,i.e., $\Gamma = \partial\Omega$ (keyword **border**, see **??**)

$\mathcal{T}_h$ the triangulation of Ω, i.e., the set of triangles T_k, where h stands for mesh size (keyword **mesh**, **buildmesh**, see **??**)

n_t the number of triangles in $\mathcal{T}_h$ (get by `Th.nt`, see "mesh.edp")

Ω_h denotes the approximated domain $\Omega_h = \cup_{k=1}^{n_t} T_k$ of Ω. If Ω is polygonal domain, then it will be $\Omega = \Omega_h$

Γ_h the boundary of Ω_h

n_v the number of vertices in $\mathcal{T}_h$ (get by `Th.nv`)

$[q^i q^j\ ]$ the segment connecting q^i and q^j

$q^{k_1}, q^{k_2}, q^{k_3}$ the vertices of a triangle T_k with anti-clock direction (get the coordinate of q^{k_j} by (`Th[k-1][j-1].x, Th[k-1][j-1].y`))

I_Ω the set $\{i \in \mathbb{N} |\ q^i \notin \Gamma_h\}$

A.6 Finite Element Spaces

$L^2(\Omega)$ the set $\left\{ w(x,y) \,\middle|\, \int_\Omega |w(x,y)|^2 \mathrm{d}x\mathrm{d}y < \infty \right\}$

$$\text{norm: } \|w\|_{0,\Omega} = \left(\int_\Omega |w(x,y)|^2 \mathrm{d}x\mathrm{d}y \right)^{1/2}$$

$$\text{scalar product: } (v,w) = \int_\Omega vw$$

$H^1(\Omega)$ the set $\left\{ w \in L^2(\Omega) \,\middle|\, \int_\Omega \left(|\partial w/\partial x|^2 + |\partial w/\partial y|^2 \right) \mathrm{d}x\mathrm{d}y < \infty \right\}$

$$\text{norm: } \|w\|_{1,\Omega} = \left(\|w\|_{0,\Omega}^2 + \|\nabla u\|_{0.\Omega}^2 \right)^{1/2}$$

$H^m(\Omega)$ the set $\left\{ w \in L^2(\Omega) \,\middle|\, \int_\Omega \frac{\partial^{|\alpha|} w}{\partial x^{\alpha_1} \partial y^{\alpha_2}} \in L^2(\Omega) \quad \forall \alpha = (\alpha_1, \alpha_2) \in \mathbb{N}^2,\ |\alpha| = \alpha_1 + \alpha_2 \right\}$

$$\text{scalar product: } (v,w)_{1,\Omega} = \sum_{|\alpha| \le m} \int_\Omega D^\alpha v D^\alpha w$$

$H_0^1(\Omega)$ the set $\{w \in H^1(\Omega) \,|\, u = 0 \quad \text{on } \Gamma\}$

$L^2(\Omega)^2$ denotes $L^2(\Omega) \times L^2(\Omega)$, and also $H^1(\Omega)^2 = H^1(\Omega) \times H^1(\Omega)$

V_h denotes the finite element space created by " `fespace` Vh(Th,*)" in `FreeFem++` (see **??** for "*")

$\Pi_h f$ the projection of the function f into V_h (" `func f=x^2*y^3; Vh v = f;`" means `v` $= \Pi_h$ `f`)

$\{v\}$ for FE-function v in V_h means the column vector $(v_1, \cdots, v_M)^T$ if $v = v_1\phi_1 + \cdots + v_M\phi_M$, which is shown by " `fespace Vh(Th,P2); Vh v; cout << v[] << endl;`"

第 B 章

Grammar

B.1 The bison grammar

```
start:   input ENDOFFILE;

input:   instructions ;

instructions:  instruction
        | instructions  instruction   ;

list_of_id_args:
            | id
            | id '=' no_comma_expr
            | FESPACE id
            | type_of_dcl id
            | type_of_dcl '&' id
            | '[' list_of_id_args ']'
            | list_of_id_args ',' id
            | list_of_id_args ',' '[' list_of_id_args ']'
            | list_of_id_args ',' id '=' no_comma_expr
            | list_of_id_args ',' FESPACE id
            | list_of_id_args ',' type_of_dcl id
            | list_of_id_args ',' type_of_dcl '&' id ;

list_of_id1:  id
            | list_of_id1 ',' id   ;

id: ID | FESPACE ;

list_of_dcls:    ID
              |  ID '='   no_comma_expr
              |  ID  '(' parameters_list ')'
              |  list_of_dcls ',' list_of_dcls  ;


parameters_list:
           no_set_expr
        |  FESPACE  ID
```

```
        |  ID '=' no_set_expr
        | parameters_list ',' no_set_expr
        | parameters_list ',' id '=' no_set_expr ;

type_of_dcl:   TYPE
             | TYPE '[' TYPE ']' ;

ID_space:
    ID
 |  ID '[' no_set_expr ']'
 |  ID '=' no_set_expr
 |  '[' list_of_id1 ']'
 |  '[' list_of_id1 ']' '[' no_set_expr ']'
 |  '[' list_of_id1 ']' '=' no_set_expr ;

ID_array_space:
    ID '(' no_set_expr ')'
 |  '[' list_of_id1 ']' '(' no_set_expr ')' ;

fespace: FESPACE ;

spaceIDa  :      ID_array_space
            |    spaceIDa ',' ID_array_space  ;

spaceIDb  :      ID_space
            |    spaceIDb ',' ID_space ;

spaceIDs :    fespace              spaceIDb
           |  fespace '[' TYPE ']'  spaceIDa    ;

fespace_def: ID '(' parameters_list ')' ;

fespace_def_list:  fespace_def
                 | fespace_def_list ',' fespace_def ;


declaration:   type_of_dcl list_of_dcls ';'
             | 'fespace' fespace_def_list    ';'
             | spaceIDs ';'
             | FUNCTION ID '=' Expr ';'
             | FUNCTION type_of_dcl ID  '(' list_of_id_args ')'  '{' instructions'}'
             | FUNCTION ID '(' list_of_id_args ')'   '='   no_comma_expr  ';'     ;

begin: '{'  ;
end:   '}'  ;

for_loop:    'for'   ;
while_loop:  'while' ;

instruction:   ';'
         | 'include'  STRING
         | 'load'  STRING
         |  Expr  ';'
         |  declaration
         |  for_loop  '(' Expr ';' Expr ';' Expr ')' instruction
         |  while_loop '(' Expr ')' instruction
```

```
          |  'if' '(' Expr ')'   instruction
          |  'if' '(' Expr ')'   instruction  ELSE instruction
          |  begin  instructions end
          |  'border'  ID   border_expr
          |  'border'   ID   '['  array ']' ';'
          |  'break'  ';'
          |  'continue'  ';'
          |  'return'  Expr ';'  ;


bornes: '(' ID '=' Expr ',' Expr ')' ;

border_expr:   bornes instruction  ;

Expr:    no_comma_expr
       | Expr ',' Expr ;


unop:     '-'
        | '+'
        | '!'
        | '++'
        | '--'  ;


no_comma_expr:
       no_set_expr
| no_set_expr '=' no_comma_expr
| no_set_expr '+=' no_comma_expr
| no_set_expr '-=' no_comma_expr
| no_set_expr '*=' no_comma_expr
| no_set_expr '/=' no_comma_expr
| no_set_expr '.*=' no_comma_expr
| no_set_expr './=' no_comma_expr ;


no_set_expr:
  no_ternary_expr
| no_ternary_expr '?' no_set_expr ':' no_set_expr ;

no_ternary_expr:
          unary_expr
        | no_ternary_expr '*' no_ternary_expr
        | no_ternary_expr '.*' no_ternary_expr
        | no_ternary_expr './' no_ternary_expr
        | no_ternary_expr '/' no_ternary_expr
        | no_ternary_expr '%' no_ternary_expr
        | no_ternary_expr '+' no_ternary_expr
        | no_ternary_expr '-' no_ternary_expr
        | no_ternary_expr '<<' no_ternary_expr
        | no_ternary_expr '>>' no_ternary_expr
        | no_ternary_expr '&' no_ternary_expr
        | no_ternary_expr '&&' no_ternary_expr
        | no_ternary_expr '|' no_ternary_expr
        | no_ternary_expr '||' no_ternary_expr
        | no_ternary_expr '<' no_ternary_expr
```

```
        | no_ternary_expr '<=' no_ternary_expr
        | no_ternary_expr '>' no_ternary_expr
        | no_ternary_expr '>=' no_ternary_expr
        | no_ternary_expr '==' no_ternary_expr
        | no_ternary_expr '!=' no_ternary_expr ;

sub_script_expr:
     no_set_expr
|   ':'
|   no_set_expr ':' no_set_expr
|   no_set_expr ':' no_set_expr ':' no_set_expr ;


parameters:
        |   no_set_expr
        |   FESPACE
        |   id '=' no_set_expr
        |   sub_script_expr
        |   parameters ',' FESPACE
        |   parameters ',' no_set_expr
        |   parameters ',' id '=' no_set_expr ;

array:   no_comma_expr
       | array ',' no_comma_expr ;


unary_expr:
    pow_expr
  | unop  pow_expr %prec UNARY ;

pow_expr: primary
  |      primary  '^' unary_expr
  |      primary  '_' unary_expr
  |      primary ''  ;                                    //    transpose

primary:
          ID
  |       LNUM
  |       DNUM
  |       CNUM
  |       STRING
  |       primary '('  parameters ')'
  |       primary '[' Expr ']'
  |       primary '['  ']'
  |       primary '.'  ID
  |       primary '++'
  |       primary '--'
  |       TYPE '('  Expr ')' ;
  |       '(' Expr ')'
  |       '[' array  ']' ;
```

B.2 The Types of the languages, and cast

B.3 All the operators

```
- CG,  type :<TypeSolveMat>
- Cholesky,  type :<TypeSolveMat>
- Crout,  type :<TypeSolveMat>
- GMRES,  type :<TypeSolveMat>
- LU,  type :<TypeSolveMat>
- LinearCG,  type :<Polymorphic>   operator() :
 (    <long> :   <Polymorphic>, <KN<double> *>, <KN<double> *> )

- N,  type :<Fem2D::R3>
- NoUseOfWait,  type :<bool *>
- P,  type :<Fem2D::R3>
- P0,  type :<Fem2D::TypeOfFE>
- P1,  type :<Fem2D::TypeOfFE>
- P1nc,  type :<Fem2D::TypeOfFE>
- P2,  type :<Fem2D::TypeOfFE>
- RT0,  type :<Fem2D::TypeOfFE>
- RTmodif,  type :<Fem2D::TypeOfFE>
- abs,  type :<Polymorphic>  operator() :
 (    <double> :   <double> )

- acos,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )

- acosh,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )

- adaptmesh,  type :<Polymorphic>   operator() :
 (    <Fem2D::Mesh> :   <Fem2D::Mesh>... )

- append,  type :<std::ios_base::openmode>
- asin,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )

- asinh,  type :<Polymorphic>  operator() :
 (    <double> :   <double> )

- atan,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
 (    <double> :   <double>, <double> )

- atan2,  type :<Polymorphic>   operator() :
 (    <double> :   <double>, <double> )

- atanh,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
```

```
- buildmesh,  type :<Polymorphic>   operator() :
 (    <Fem2D::Mesh> :   <E_BorderN> )

- buildmeshborder,  type :<Polymorphic>   operator() :
 (    <Fem2D::Mesh> :   <E_BorderN> )

- cin,  type :<istream>
- clock,  type :<Polymorphic>
 (    <double> :   )

- conj,  type :<Polymorphic>   operator() :
 (    <complex> :   <complex> )

- convect,  type :<Polymorphic>   operator() :
 (    <double> :   <E_Array>, <double>, <double> )

- cos,  type :<Polymorphic>  operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- cosh,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- cout,  type :<ostream>
- dumptable,  type :<Polymorphic>   operator() :
 (    <ostream> :   <ostream> )

- dx,  type :<Polymorphic>   operator() :
 (    <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
 (    <double> :   <std::pair<FEbase<double> *, int>> )
 (    <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- dy,  type :<Polymorphic>   operator() :
 (    <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
 (    <double> :   <std::pair<FEbase<double> *, int>> )
 (    <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- endl,  type :<char>
- exec,  type :<Polymorphic>   operator() :
 (    <long> :   <string> )

- exit,  type :<Polymorphic>  operator() :
 (    <long> :   <long> )

- exp,  type :<Polymorphic>  operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )
```

```
- false,  type :<bool>
- imag,  type :<Polymorphic>   operator() :
 (    <double> :   <complex> )

- int1d,  type :<Polymorphic>   operator() :
 (    <CDomainOfIntegration> :   <Fem2D::Mesh>... )

- int2d,  type :<Polymorphic>   operator() :
 (    <CDomainOfIntegration> :   <Fem2D::Mesh>... )

- intalledges,  type :<Polymorphic>
 operator( :
 (    <CDomainOfIntegration> :   <Fem2D::Mesh>... )

- jump,  type :<Polymorphic>
 operator( :
 (    <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
 (    <double> :   <double> )
 (    <complex > :   <complex > )
 (    <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- label,  type :<long *>
- log,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- log10,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )

- max,  type :<Polymorphic>   operator() :
 (    <double> :   <double>, <double> )
 (    <long> :   <long>, <long> )

- mean,  type :<Polymorphic>
 operator( :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- min,  type :<Polymorphic>  operator() :
 (    <double> :   <double>, <double> )
 (    <long> :   <long>, <long> )

- movemesh,  type :<Polymorphic>   operator() :
 (    <Fem2D::Mesh> :   <Fem2D::Mesh>, <E_Array>... )

- norm,  type :<Polymorphic>
 operator( :
 (    <double> :   <std::complex<double>> )

- nuTriangle,  type :<long>
```

```
- nuEdge,  type :<long>
- on,  type :<Polymorphic>   operator() :
 (    <BC_set<double>> :   <long>... )

- otherside,  type :<Polymorphic>
 operator( :
 (    <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
 (    <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- pi,  type :<double>
- plot,  type :<Polymorphic>   operator() :
 (    <long> :  ... )

- pow,  type :<Polymorphic>   operator() :
 (    <double> :   <double>, <double> )
 (    <complex> :   <complex>, <complex> )

- qf1pE,  type :<Fem2D::QuadratureFormular1d>
- qf1pT,  type :<Fem2D::QuadratureFormular>
- qf1pTlump,  type :<Fem2D::QuadratureFormular>
- qf2pE,  type :<Fem2D::QuadratureFormular1d>
- qf2pT,  type :<Fem2D::QuadratureFormular>
- qf2pT4P1,  type :<Fem2D::QuadratureFormular>
- qf3pE,  type :<Fem2D::QuadratureFormular1d>
- qf5pT,  type :<Fem2D::QuadratureFormular>

- readmesh,  type :<Polymorphic>   operator() :
 (    <Fem2D::Mesh> :   <string> )

- real,  type :<Polymorphic>   operator() :
 (    <double> :   <complex> )

- region,  type :<long *>
- savemesh,  type :<Polymorphic>  operator() :
 (    <Fem2D::Mesh> :   <Fem2D::Mesh>, <string>... )

- sin,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- sinh,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- sqrt,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- square,  type :<Polymorphic>    operator() :
 (    <Fem2D::Mesh> :   <long>, <long> )
```

```
  (    <Fem2D::Mesh> :   <long>, <long>, <E_Array> )

- tan,  type :<Polymorphic>   operator() :
  (    <double> :   <double> )

- true,  type :<bool>
- trunc,  type :<Polymorphic>   operator() :
  (    <Fem2D::Mesh> :   <Fem2D::Mesh>, <bool> )

- verbosity,  type :<long *>
- wait,  type :<bool *>
- x,  type :<double *>
- y,  type :<double *>
- z,  type :<double *>
```

第 C 章

Dynamical link

Now, it's possible to add built-in functionnalites in FreeFem++ under the three environnents Linux, Windows and MacOS X 10.3 or newer. It is agood idea to, first try the example load.edp in directory example++-load.
You will need to install a c++ compiler (generally g++/gcc compiler) to compile your function.

Windows Install the cygwin environnent or the mingw

MacOs Install the developer tools xcode on the apple DVD

Linux/Unix Install the correct compiler (gcc for instance)

Now, assume that you are in a shell window (a cygwin window under Windows) in the directory example++-load. Remark that in the sub directory include they are all the FreeFem++ include file to make the link with FreeFem++.

Note C.1 *If you try to load dynamically a file with command* `load "xxx"`

- *Under unix (Linux or MacOs), the file* `xxx.so` *twill be loaded so it must be either in the search directory of routine* `dlopen` *(see the environment variable* `$LD_LIBRARY_PATH` *or in the current directory, and the suffix* `".so"` *or the prefix* `"./"` *is automatically added.*
- *Under Windows, The file* `xxx.dll` *will be loaded so it must be in the* `loadLibary` *search directory which includes the directory of the application,*

The compilation of your module: the script ff-c++ compiles and makes the link with FreeFem++, but be careful, the script has no way to known if you try to compile for a pure Windows environment or for a cygwin environment so to build the load module under cygwin you must add the -cygwin parameter.

C.1 A first example myfunction.cpp

The following defines a new function call myfunction with no parameter, but using the x, y current value.

```
#include  <iostream>
#include  <cfloat>
using namespace std;
#include "error.hpp"
#include "AFunction.hpp"
#include "rgraph.hpp"
#include "RNM.hpp"
#include "fem.hpp"
#include "FESpace.hpp"
#include "MeshPoint.hpp"

using namespace Fem2D;
double myfunction(Stack stack)
{
                                                    //    to get FreeFem++ data
  MeshPoint &mp= *MeshPointStack(stack); //    the struct to get x,y, normal ,
value
  double x= mp.P.x;                                 //    get the current x value
  double y= mp.P.y;                                 //    get the current y value
                                  //    cout << "x = " << x << " y=" << y << endl;
  return sin(x)*cos(y);
}
```

Now the Problem is to build the link with FreeFem++, to do that we need two classes, one to call the function myfunction
All FreeFem++ evaluable expression must be a struct/class C++ which derive from E_F0. By default this expression does not depend of the mesh position, but if they derive from E_F0mps the expression depends of the mesh position, and for more details see [12].

```
                                      //    A class build the link with FreeFem++
                            //    generaly this class are already in AFunction.hpp
              //    but unfortunatly, I have no simple function with no parameter
                                       //    in FreeFem++ depending of the mesh,
template<class R>
class  OneOperator0s : public OneOperator {

                             //    the class to defined a evaluated a new function
                          //    It must devive from E_F0 if it is mesh independent
                                    //    or from E_F0mps if it is mesh dependent
  class E_F0_F :public  E_F0mps { public:
    typedef  R (*func)(Stack stack) ;
    func f;                                 //    the pointeur to the fnction myfunction
    E_F0_F(func ff)  : f(ff) {}
                                           //    the operator evaluation in FreeFem++
    AnyType operator()(Stack stack)  const {return SetAny<R>( f(stack)) ;}

  };

  typedef  R (*func)(Stack ) ;
  func  f;
public:
                            //    the function which build the FreeFem++ byte code
  E_F0 * code(const basicAC_F0 & ) const { return  new E_F0_F(f);}
                 //    the constructor to say ff is a function without parameter
```

```
                                                    //    and returning a R
  OneOperator0s(func  ff): OneOperator(map_type[typeid(R).name()]),f(ff){}
};
```

To finish we must add this new function in FreeFem++ table , to do that include :

```
void init(){
  Global.Add("myfunction","(",new OneOperator0s<double>(myfunction));
}
LOADFUNC(init);
```

It will be called automatically at load module time.

To compile and link, use the ff-c++ script :

```
% ff-c++ myfunction.cpp
g++ -c -g -Iinclude myfunction.cpp
g++ -bundle -undefined dynamic_lookup -g myfunction.o -o ./myfunction.dylib
```

To, try the simple example under Linux or MacOS, do

```
% FreeFem++-nw load.edp
-- FreeFem++ v 1.4800028 (date Tue Oct  4 11:56:46 CEST 2005)
 file : load.edp
 Load: lg_fem lg_mesh eigenvalue  UMFPACK
    1 :                                  //    Example of dynamic function load
    2 :                                  //    --------------------------------
    3 :                    //    Id : freefem++doc.tex, v1.1102010/06/0411 : 27 : 24hechtExp
    4 :
    5 :  load "myfunction"

load: myfunction
load: dlopen(./myfunction) = 0xb01cc0

    6 :  mesh Th=square(5,5);
    7 :  fespace Vh(Th,P1);
    8 :  Vh uh=myfunction();                      //    warning do not forget ()
    9 :  cout << uh[].min << " " << uh[].max << endl;
   10 :  sizestack + 1024 =1240  ( 216 )

 -- square mesh : nb vertices  =36 ,  nb triangles = 50 ,  nb boundary edges 20
 Nb of edges on Mortars  = 0
 Nb of edges on Boundary = 20, neb = 20
 Nb Of Nodes = 36
 Nb of DF = 36
0 0.841471
times: compile 0.05s, execution -3.46945e-18s
CodeAlloc : nb ptr  1394,  size :71524
Bien: On a fini Normalement
```

Under Windows, launch FreeFem++ with the mouse (or ctrl O) on the example.

C.2 Example: Discrete Fast Fourier Transform

This will add FFT to FreeFem++, taken from http://www.fftw.org/. To download and install under download/include just go in download/fftw and trymake.

The 1D dfft (fast discret fourier transform) for a simple array f of size n is defined by the following formula

$$\texttt{dfft}(f,\varepsilon)_k = \sum_{j=0}^{n-1} f_i e^{\varepsilon 2\pi i kj/n}$$

The 2D DFFT for an array of size $N = n \times m$ is

$$\texttt{dfft}(f,m,\varepsilon)_{k+nl} = \sum_{j'=0}^{m-1}\sum_{j=0}^{n-1} f_{i+nj} e^{\varepsilon 2\pi i(kj/n+lj'/m)}$$

Remark: the value n is given by $size(f)/m$, and the numbering is row-major order.
So the classical discrete DFFT is $\hat{f} = \texttt{dfft}(f,-1)/\sqrt{n}$ and the reverse dFFT $f = \texttt{dfft}(\hat{f},1)/\sqrt{n}$

Remark: the 2D Laplace operator is

$$f(x,y) = 1/\sqrt{N}\sum_{j'=0}^{m-1}\sum_{j=0}^{n-1} \hat{f}_{i+nj} e^{\varepsilon 2\pi i(xj+yj')}$$

and we have

$$f_{k+nl} = f(k/n, l/m)$$

So

$$\widehat{\Delta f_{kl}} = -((2\pi)^2((\tilde{k})^2+(\tilde{l})^2))\widehat{f_{kl}}$$

where $\tilde{k} = k$ if $k \le n/2$ else $\tilde{k} = k-n$ and $\tilde{l} = l$ if $l \le m/2$ else $\tilde{l} = l-m$.
And to have a real function we need all modes to be symmetric around zero, so n and m must be odd.

Compile to build a new library

```
%  ff-c++  dfft.cpp ../download/install/lib/libfftw3.a -I../download/install/include
export MACOSX_DEPLOYMENT_TARGET=10.3
g++ -c -Iinclude -I../download/install/include dfft.cpp
g++ -bundle -undefined dynamic_lookup dfft.o -o ./dfft.dylib ../download/install/lib/libfftw3.a
```

To test ,

```
-- FreeFem++ v 1.4800028 (date Mon Oct 10 16:53:28 EEST 2005)
 file : dfft.edp
 Load: lg_fem cadna lg_mesh eigenvalue  UMFPACK
    1 :                                    //    Example of dynamic function load
    2 :                                    //    -------------------------------
    3 :                      //    Id : freefem++doc.tex,v1.1102010/06/0411 : 27 : 24hechtExp
    4 :                                      //    Discret Fast Fourier Transform
    5 :                                      //    -------------------------------
    6 :  load "dfft" lood: init dfft

load: dlopen(dfft.dylib) = 0x2b0c700
```

```
    7 :
    8 : int nx=32,ny=16,N=nx*ny;
    9 : //    warning the Fourier space is not exactly the unite square due to
periodic conditions
   10 : mesh Th=square(nx-1,ny-1,[(nx-1)*x/nx,(ny-1)*y/ny]);
   11 :                 //    warring the numbering is of the vertices (x,y) is
   12 :                                          //    given by i = x/nx + nx * y/ny
   13 :
   14 : fespace Vh(Th,P1);
   15 :
   16 : func f1 = cos(2*x*2*pi)*cos(3*y*2*pi);
   17 : Vh<complex> u=f1,v;
   18 : Vh w=f1;
   19 :
   20 :
   21 : Vh  ur,ui;
   22 :    //    in dfft the matrix n,m is in row-major order ann array n,m is
   23 :          //    store j + m* i ( the transpose of the square numbering )
   24 :  v[]=dfft(u[],ny,-1);
   25 :  u[]=dfft(v[],ny,+1);
   26 :  u[] /= complex(N);
   27 :  v = f1-u;
   28 : cout << " diff = "<< v[].max << " " <<  v[].min << endl;
   29 : assert( norm(v[].max) < 1e-10 &&  norm(v[].min) < 1e-10) ;
   30 :  //    ------- a more hard example ----
   31 :  //    Lapacien en FFT
   32 :  //    -Δu = f with biperiodic condition
   33 : func f = cos(3*2*pi*x)*cos(2*2*pi*y);                                  //
   34 : func ue =  +(1./(square(2*pi)*13.))*cos(3*2*pi*x)*cos(2*2*pi*y);     //

   35 : Vh<complex> ff = f;
   36 : Vh<complex> fhat;
   37 : fhat[] = dfft(ff[],ny,-1);
   38 :
   39 : Vh<complex> wij;
   40 :  //    warning in fact we take mode between -nx/2, nx/2 and -ny/2,ny/2
   41 :                                          //    thank to the operator ?:
   42 : wij = square(2.*pi)*(square(( x<0.5?x*nx:(x-1)*nx))
          + square((y<0.5?y*ny:(y-1)*ny)));
   43 : wij[][0] = 1e-5;                                  //    to remove div / 0
   44 : fhat[] = fhat[]./ wij[];                                          //
   45 : u[]=dfft(fhat[],ny,1);
   46 : u[] /= complex(N);
   47 : ur = real(u);                                            //    the solution
   48 : w = real(ue);                                      //    the exact solution
   49 : plot(w,ur,value=1 ,cmm=" ue   ", wait=1);
   50 : w[] -= ur[];                                               //    array sub
   51 : real err= abs(w[].max)+abs(w[].min) ;
   52 : cout << " err = " << err << endl;
   53 : assert( err  < 1e-6);
   54 :  sizestack + 1024 =3544  ( 2520 )

----------CheckPtr:-----init execution ------ NbUndelPtr  2815  Alloc: 111320  NbPtr
6368
 -- square mesh : nb vertices  =512 ,  nb triangles = 930 ,  nb boundary edges 92
```

```
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 92, neb = 92
 Nb Of Nodes = 512
 Nb of DF = 512
0x2d383d8 -1 16 512 n: 16 m:32
 dfft 0x402bc08 = 0x4028208 n = 16 32 sign = -1
 --- --- ---0x2d3ae08 1 16 512 n: 16 m:32
 dfft 0x4028208 = 0x402bc08 n = 16 32 sign = 1
 --- --- --- diff = (8.88178e-16,3.5651e-16) (-6.66134e-16,-3.38216e-16)
0x2d3cfb8 -1 16 512 n: 16 m:32
 dfft 0x402de08 = 0x402bc08 n = 16 32 sign = -1
 --- --- ---0x2d37ff8 1 16 512 n: 16 m:32
 dfft 0x4028208 = 0x402de08 n = 16 32 sign = 1
 --- --- --- err = 3.6104e-12
times: compile 0.13s, execution 2.05s
----------CheckPtr:-----end execution -- ------ NbUndelPtr  2815  Alloc: 111320
NbPtr 26950
 CodeAlloc : nb ptr  1693,  size :76084
Bien: On a fini Normalement
                CheckPtr:Nb of undelete pointer is 2748 last 114
                CheckPtr:Max Memory used    228.531 kbytes  Memory undelete 105020
```

C.3 Load Module for Dervieux' P0-P1 Finite Volume Method

the associed edp file is `examples++-load/convect_dervieux.edp`

```
                                        //    Implementation of P1-P0 FVM-FEM
   //    ------------------------------------------------------------------
                            //    Id : freefem++doc.tex,v1.1102010/06/0411 : 27 : 24hechtExp
       //    compile and link with ff-c++ mat_dervieux.cpp (i.e.  the file name
without .cpp)
#include  <iostream>
#include  <cfloat>
#include  <cmath>
using namespace std;
#include "error.hpp"
#include "AFunction.hpp"
#include "rgraph.hpp"
#include "RNM.hpp"
                                                //    remove problem of include
#undef  HAVE_LIBUMFPACK
#undef HAVE_CADNA
#include "MatriceCreuse_tpl.hpp"
#include "MeshPoint.hpp"
#include "lgfem.hpp"
#include "lgsolver.hpp"
#include "problem.hpp"


class MatrixUpWind0 :  public E_F0mps { public:
  typedef Matrice_Creuse<R> * Result;
  Expression emat,expTh,expc,expu1,expu2;
  MatrixUpWind0(const basicAC_F0 & args)
  {
```

```
    args.SetNameParam();
    emat =args[0];                                    //    the matrix expression
    expTh= to<pmesh>(args[1]);           //    a the expression to get the mesh
    expc = CastTo<double>(args[2]);    //    the expression to get c (must be a
double)
                                                      //    a array expression [ a, b]
    const E_Array * a= dynamic_cast<const E_Array*>((Expression) args[3]);
    if (a->size() != 2) CompileError("syntax:  MatrixUpWind0(Th,rhi,[u1,u2])");
    int err =0;
    expu1= CastTo<double>((*a)[0]);      //    fist exp of the array (must be a
double)
    expu2= CastTo<double>((*a)[1]);    //    second exp of the array (must be a
double)
  }

  ~MatrixUpWind0(){
  }

  static ArrayOfaType  typeargs()
  { return  ArrayOfaType(atype<Matrice_Creuse<R>*>(),
    atype<pmesh>(),atype<double>(),atype<E_Array>());}
  static  E_F0 * f(const basicAC_F0 & args){ return new MatrixUpWind0(args);}

  AnyType operator()(Stack s) const ;

};

int   fvmP1P0(double q[3][2], double u[2],double c[3], double a[3][3], double where[3]
)
{                                  //    computes matrix a on a triangle for the
Dervieux FVM
  for(int i=0;i<3;i++) for(int j=0;j<3;j++) a[i][j]=0;

    for(int i=0;i<3;i++){
        int ip = (i+1)%3, ipp =(ip+1)%3;
        double unL =-((q[ip][1]+q[i][1]-2*q[ipp][1])*u[0]
                -(q[ip][0]+q[i][0]-2*q[ipp][0])*u[1])/6;
        if(unL>0) { a[i][i] += unL; a[ip][i]-=unL;}
            else{ a[i][ip] += unL; a[ip][ip]-=unL;}
        if(where[i]&&where[ip]){                  //    this is a boundary edge
            unL=((q[ip][1]-q[i][1])*u[0] -(q[ip][0]-q[i][0])*u[1])/2;
            if(unL>0) { a[i][i]+=unL; a[ip][ip]+=unL;}
        }
    }
  return 1;
}

                                                  //    the evaluation routine
AnyType MatrixUpWind0::operator()(Stack stack) const
{
  Matrice_Creuse<R> * sparse_mat =GetAny<Matrice_Creuse<R>* >((*emat)(stack));
  MatriceMorse<R> * amorse =0;
  MeshPoint *mp(MeshPointStack(stack)) , mps=*mp;
  Mesh * pTh = GetAny<pmesh>((*expTh)(stack));
  ffassert(pTh);
  Mesh & Th (*pTh);
```

```
  {
    map< pair<int,int>, R> Aij;
    KN<double> cc(Th.nv);
    double infini=DBL_MAX;
    cc=infini;
    for (int it=0;it<Th.nt;it++)
      for (int iv=0;iv<3;iv++)
    {
      int i=Th(it,iv);
      if ( cc[i]==infini) {                              //    if nuset the set
        mp->setP(&Th,it,iv);
        cc[i]=GetAny<double>((*expc)(stack));
      }
    }

    for (int k=0;k<Th.nt;k++)
      {
    const Triangle & K(Th[k]);
   const Vertex & A(K[0]), &B(K[1]),&C(K[2]);
    R2 Pt(1./3.,1./3.);
    R u[2];
    MeshPointStack(stack)->set(Th,K(Pt),Pt,K,K.lab);
    u[0] = GetAny< R>( (*expu1)(stack) ) ;
    u[1] = GetAny< R>( (*expu2)(stack) ) ;

    int ii[3] ={  Th(A), Th(B),Th(C)};
    double q[3][2]= { { A.x,A.y} ,{B.x,B.y},{C.x,C.y} } ;     //    coordinates
of 3 vertices (input)
    double c[3]={cc[ii[0]],cc[ii[1]],cc[ii[2]]};
    double a[3][3], where[3]={A.lab,B.lab,C.lab};
    if (fvmP1P0(q,u,c,a,where) )
      {
        for (int i=0;i<3;i++)
          for (int j=0;j<3;j++)
        if (fabs(a[i][j]) >= 1e-30)
          { Aij[make_pair(ii[i],ii[j])]+=a[i][j];
            }
      }
      }
    amorse=  new MatriceMorse<R>(Th.nv,Th.nv,Aij,false);
  }
  sparse_mat->pUh=0;
  sparse_mat->pVh=0;
  sparse_mat->A.master(amorse);
  sparse_mat->typemat=(amorse->n == amorse->m) ? TypeSolveMat(TypeSolveMat::GMRES)
: TypeSolveMat(TypeSolveMat::NONESQUARE);     //    none square matrice (morse)
  *mp=mps;

  if(verbosity>3) { cout << "  End Build MatrixUpWind : " << endl;}

 return sparse_mat;
}

void init()
   {
     cout << " lood: init Mat Chacon " << endl;
```

```
    Global.Add("MatUpWind0","(", new OneOperatorCode<MatrixUpWind0 >( ));
  }

LOADFUNC(init);
```

C.4 More on Adding a new finite element

First read the section 13 of the appendix, we add two new finite elements examples in the directory `examples++-load`.

The Bernardi-Raugel Element The Bernardi-Raugel finite element is meant to solve the Navier Stokes equations in u, p formulation; the velocity space P_K^{br} is minimal to prove the inf-sup condition with piecewise constant pressure by triangle.
The finite element space V_h is

$$V_h = \{u \in H^1(\Omega)^2; \quad \forall K \in T_h, u_{|K} \in P_K^{br}\}$$

where

$$P_K^{br} = span\{\lambda_i^K e_k\}_{i=1,2,3,k=1,2} \cup \{\lambda_i^K \lambda_{i+1}^K n_{i+2}^K\}_{i=1,2,3}$$

with notation $4 = 1, 5 = 2$ and where λ_i^K are the barycentric coordinates of the triangle K, $(e_k)_{k=1,2}$ the canonical basis of $\mathbb{R}^2$ and n_k^K the outer normal of triangle K opposite to vertex k.

```
//    The P2BR finite element :  the Bernadi Raugel Finite Element
//    F. Hecht, decembre 2005
//    -------------
 //    See Bernardi, C., Raugel, G.: Analysis of some finite elements for the
Stokes problem. Math. Comp. 44, 71-79 (1985).
//    It is a 2d coupled FE
//    the Polynomial space is P1^2 + 3 normals bubbles edges function (P_2)
       //    the degree of freedom is 6 values at of the 2 componants at the 3
vertices
//    and the 3 flux on the 3 edges
//    So 9 degrees of freedom and N= 2.

//    ---------------------- related files:
//    to check and validate :  testFE.edp
//    to get a real example :  NSP2BRP0.edp
//    ------------------------------------------------------------

                                                          //    ----------------------
#include "error.hpp"
#include "AFunction.hpp"
#include "rgraph.hpp"
using namespace std;
#include "RNM.hpp"
#include "fem.hpp"
#include "FESpace.hpp"
#include "AddNewFE.h"
```

```
namespace  Fem2D {

  class TypeOfFE_P2BRLagrange : public  TypeOfFE { public:
    static int Data[];

    TypeOfFE_P2BRLagrange(): TypeOfFE(6+3+0,
      2,
      Data,
      4,
      1,
      6+3*(2+2),                              //    nb coef to build interpolation
      9,                                     //    np point to build interpolation
      0)
    {
 ....                                                  //    to long see the source
     }
    void FB(const bool * whatd, const Mesh & Th,const Triangle & K,const R2 &P,
RNMK_ & val) const;
    void TypeOfFE_P2BRLagrange::Pi_h_alpha(const baseFElement & K,KN_<double> &
v) const;
  } ;
                                                //    on what nu df on node node of df
 int TypeOfFE_P2BRLagrange::Data[]={
   0,0, 1,1, 2,2,  3,4,5,
   0,1, 0,1, 0,1,  0,0,0,
   0,0, 1,1, 2,2,  3,4,5,
   0,0, 0,0, 0,0,  0,0,0,
   0,1, 2,3, 4,5,  6,7,8,
   0,0
};

void TypeOfFE_P2BRLagrange::Pi_h_alpha(const baseFElement & K,KN_<double> & v) const
  {
    const Triangle & T(K.T);
    int k=0;
                              //    coef pour les 3 sommets fois le 2 composantes
    for (int i=0;i<6;i++)
      v[k++]=1;
                                                //    integration sur les aretes
    for (int i=0;i<3;i++)
      {

        R2 N(T.Edge(i).perp());
N  *= T.EdgeOrientation(i)*0.5 ;
        v[k++]= N.x;
        v[k++]= N.y;
        v[k++]= N.x;
        v[k++]= N.y;
      }
  }

  void TypeOfFE_P2BRLagrange::FB(const bool * whatd,const Mesh & ,const Triangle
& K,const R2 & P,RNMK_ & val) const
  {
 ....                                                  //    to long see the source
  }
```

```
        //    ---- cooking to add the finite elemet to freefem table --------
                                  //    a static variable to def the finite element
  static TypeOfFE_P2BRLagrange P2LagrangeP2BR;
                                                //    now adding FE in FreeFEm++ table
  static AddNewFE P2BR("P2BR",&P2LagrangeP2BR);
                                                                  //    --- end cooking
}                                                               //    end FEM2d namespace
```

A way to check the finite element

```
load "BernadiRaugel"
                                      //    a macro the compute numerical derivative
macro DD(f,hx,hy) ( (f(x1+hx,y1+hy)-f(x1-hx,y1-hy))/(2*(hx+hy)))          //
mesh Th=square(1,1,[10*(x+y/3),10*(y-x/3)]);

real x1=0.7,y1=0.9, h=1e-7;
int it1=Th(x1,y1).nuTriangle;

fespace Vh(Th,P2BR);

Vh [a1,a2],[b1,b2],[c1,c2];


for (int i=0;i<Vh.ndofK;++i)
cout << i << " " << Vh(0,i) << endl;
for (int i=0;i<Vh.ndofK;++i)
{
  a1[]=0;
  int j=Vh(it1,i);
  a1[][j]=1;                                                  //    a bascis functions
  plot([a1,a2], wait=1);

  [b1,b2]=[a1,a2];                                          //    do the interpolation

  c1[] = a1[] - b1[];

  cout << " ---------" << i << " " << c1[].max << " " << c1[].min << endl;
  cout << " a = " << a1[] <<endl;
  cout << " b = " << b1[] <<endl;
  assert(c1[].max < 1e-9 && c1[].min > -1e-9);                //    check if the
interpolation is correct

                              //    check the derivative and numerical derivative

  cout << " dx(a1)(x1,y1) = " << dx(a1)(x1,y1) << " == " << DD(a1,h,0) << endl;
  assert( abs(dx(a1)(x1,y1)-DD(a1,h,0) ) < 1e-5);
  assert( abs(dx(a2)(x1,y1)-DD(a2,h,0) ) < 1e-5);
  assert( abs(dy(a1)(x1,y1)-DD(a1,0,h) ) < 1e-5);
  assert( abs(dy(a2)(x1,y1)-DD(a2,0,h) ) < 1e-5);

}
```

A real example using this finite element, just a small modification of the NSP2P1.edp examples, just the begenning is change to

```
load "BernadiRaugel"

real s0=clock();
mesh Th=square(10,10);
fespace Vh2(Th,P2BR);
fespace Vh(Th,P0);
Vh2 [u1,u2],[up1,up2];
Vh2 [v1,v2];
```

And the plot instruction is also changed because the pressure is constant, and we cannot plot isovalues of peacewise constant functions.

The Morley Element See the example bilapMorley.edp.

C.5 Add a new sparse solver

Warning the sparse solver interface as been completely rewritten in version 3.2 , so the section is obsolete, the example in are correct/
Only a fast sketch of the code is given here; for details see the .cpp code from SuperLU.cpp or NewSolve.cpp.
First the include files:

```
#include  <iostream>
using namespace std;

#include "rgraph.hpp"
#include "error.hpp"
#include "AFunction.hpp"

                                                    //   #include "lex.hpp"
#include "MatriceCreuse_tpl.hpp"
#include "slu_ddefs.h"
#include "slu_zdefs.h"
```

A small template driver to unified the double and Complex version.

```
template <class R> struct SuperLUDriver
{

};


template <> struct SuperLUDriver<double>
{
  ....  double version
};

template <> struct SuperLUDriver<Complex>
{
....  Complex version
```

```
};
```

To get Matrix value, we have just to remark that the Morse Matrice the storage, is the SLU_NR format is the compressed row storage, this is the transpose of the compressed column storage.
So if AA is a MatriceMorse you have with SuperLU notation.

```
    n=AA.n;
    m=AA.m;
    nnz=AA.nbcoef;
    a=AA.a;
    asub=AA.cl;
    xa=AA.lg;
    options.Trans = TRANS;

    Dtype_t R_SLU = SuperLUDriver<R>::R_SLU_T();
    Create_CompCol_Matrix(&A, m, n, nnz, a, asub, xa, SLU_NC, R_SLU, SLU_GE);
```

To get vector infomation, to solver the linear solver $x = A^{-1}b$

```
   void Solver(const MatriceMorse<R> &AA,KN_<R> &x,const KN_<R> &b) const
 {
 ....
    Create_Dense_Matrix(&B, m, 1, b, m, SLU_DN, R_SLU, SLU_GE);
    Create_Dense_Matrix(&X, m, 1, x, m, SLU_DN, R_SLU, SLU_GE);
 ....
 }
```

The two BuildSolverSuperLU function, to change the default sparse solver variable DefSparseSolver<double>::solver

```
MatriceMorse<double>::VirtualSolver *
BuildSolverSuperLU(DCL_ARG_SPARSE_SOLVER(double,A))
{
    if(verbosity>9)
    cout << " BuildSolverSuperLU<double>" << endl;
    return new SolveSuperLU<double>(*A,ds.strategy,ds.tgv,ds.epsilon,ds.tol_pivot,ds.tol
}

MatriceMorse<Complex>::VirtualSolver *
BuildSolverSuperLU(DCL_ARG_SPARSE_SOLVER(Complex,A))
{
  if(verbosity>9)
    cout << " BuildSolverSuperLU<Complex>" << endl;
  return new SolveSuperLU<Complex>(*A,ds.strategy,ds.tgv,ds.epsilon,ds.tol_pivot,ds.tol_
}
```

The link to FreeFem++

```
class Init { public:
    Init();
};
```

```
  To set the  2 default sparse solver double and complex:


DefSparseSolver<double>::SparseMatSolver SparseMatSolver_R ; ;
DefSparseSolver<Complex>::SparseMatSolver SparseMatSolver_C;
```

To save the default solver type

```
TypeSolveMat::TSolveMat  TypeSolveMatdefaultvalue=TypeSolveMat::defaultvalue;
```

To reset to the default solver, call this function:

```
bool SetDefault()
{
   if(verbosity>1)
  cout << " SetDefault sparse to default" << endl;
    DefSparseSolver<double>::solver =SparseMatSolver_R;
    DefSparseSolver<Complex>::solver =SparseMatSolver_C;
    TypeSolveMat::defaultvalue =TypeSolveMat::SparseSolver;
}
```

To set the default solver to superLU, call this function:

```
bool SetSuperLU()
{
    if(verbosity>1)
  cout << " SetDefault sparse solver to SuperLU" << endl;
    DefSparseSolver<double>::solver  =BuildSolverSuperLU;
    DefSparseSolver<Complex>::solver =BuildSolverSuperLU;
    TypeSolveMat::defaultvalue =TypeSolveMatdefaultvalue;
}
```

To add new function/name `defaultsolver`,`defaulttoSuperLU`in `freefem++`, and set the default solver to the new solver., just do:

```
void init()
{

  SparseMatSolver_R= DefSparseSolver<double>::solver;
  SparseMatSolver_C= DefSparseSolver<Complex>::solver;

  if(verbosity>1)
    cout << "\n Add: SuperLU,  defaultsolver defaultsolverSuperLU" << endl;
  TypeSolveMat::defaultvalue=TypeSolveMat::SparseSolver;
  DefSparseSolver<double>::solver =BuildSolverSuperLU;
  DefSparseSolver<Complex>::solver =BuildSolverSuperLU;
                    //    test if the name "defaultsolver" exist in freefem++
 if(! Global.Find("defaultsolver").NotNull() )
    Global.Add("defaultsolver","(",new OneOperator0<bool>(SetDefault));
  Global.Add("defaulttoSuperLU","(",new OneOperator0<bool>(SetSuperLU));
}

LOADFUNC(init);
```

To compile superlu.cpp, just do:

1. download the SuperLu 3.0 package and do

```
curl   http://crd.lbl.gov/~xiaoye/SuperLU/superlu_3.0.tar.gz  -o supe
tar xvfz superlu_3.0.tar.gz
go SuperLU_3.0 directory
$EDITOR  make.inc
make
```

2. In directoy include do to have a correct version of `SuperLu` header due to mistake in case of inclusion of `double` and `Complex` version in the same file.

   ```
   tar xvfz ../SuperLU_3.0-include-ff.tar.gz
   ```

   I will give a correct one to compile with freefm++.

   To compile the freefem++ load file of SuperLu with freefem do: some find like :

   ```
   ff-c++ SuperLU.cpp  -L$HOME/work/LinearSolver/SuperLU_3.0/ -lsuperlu_
   ```

   And to test the simple example:

   A example:

```
load "SuperLU"
verbosity=2;
for(int i=0;i<3;++i)
{
//    if i == 0 then SuperLu solver
//    i == 1 then GMRES solver
//    i == 2 then Default solver
  {
    matrix A =
      [[ 0, 1, 0, 10],
       [ 0,  0,  2, 0],
       [ 0, 0, 0,  3],
       [ 4,0 , 0, 0]];
    real[int] xx = [ 4,1,2,3], x(4), b(4);
    b = A*xx;
    cout << b << " " << xx << endl;
    set(A,solver=sparsesolver);
    x = A^-1*b;
    cout << x << endl;
  }

  {
    matrix<complex> A =
      [[ 0, 1i, 0, 10],
       [ 0 ,  0,  2i, 0],
       [ 0, 0, 0,  3i],
       [ 4i,0 , 0, 0]];
    complex[int] xx = [ 4i,1i,2i,3i], x(4), b(4);
```

```
    b = A*xx;
    cout << b << " " << xx << endl;
    set(A,solver=sparsesolver);
    x = A^-1*b;
    cout << x << endl;
  }
  if(i==0)defaulttoGMRES();
  if(i==1)defaultsolver();
}
```

To Test do for exemple:

```
FreeFem++ SuperLu.edp
```

FreeFem++ LGPL License

This is The FreeFem++ software. Programs in it were maintained by

- Frédéric hecht <Frederic.Hecht@upmc.fr>
- Jacques Morice <morice@ann.jussieu.fr>

All its programs except files the comming from COOOL sofware (files in directory src/Algo) and the file mt19937ar.cpp which may be redistributed under the terms of the GNU LESSER GENERAL PUBLIC LICENSE Version 2.1, February 1999
GNU LESSER GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION
0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".
A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.
The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)
"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.
Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.
1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.
2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
a) The modified work must itself be a software library.
b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.
(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)
These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.
In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.
3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.
Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.
This option is useful when you wish to copy part of the code of the Library into a program that is not a library.
4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily

used for software interchange.
If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.
5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.
However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.
When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.
If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)
Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.
6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.
You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:
a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as

the modified version is interface-compatible with the version that the work was made with.
c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.
For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.
It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.
7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:
a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.
8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.
10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.
11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by

court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.
If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.
It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.
14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.
NO WARRANTY
15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY

PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
END OF TERMS AND CONDITIONS

第 D 章

Keywords

Main Keywords

```
adaptmesh
Cmatrix
R3
bool
border
break
buildmesh
catch
cin
complex
continue
cout
element
else
end
fespace
for
func
if
ifstream
include
int
intalledge
load
macro
matrix
mesh
movemesh
ofstream
plot
problem
real
return
savemesh
solve
string
try
throw
vertex
varf
while
```

Second category of Keywords

```
int1d
int2d
on
square
```

Third category of Keywords

```
dx
dy
convect
jump
mean
```

Fourth category of Keywords

```
wait
ps
solver
CG
LU
UMFPACK
factorize
init
endl
```

Other Reserved Words

```
x, y, z, pi, i,
sin, cos, tan, atan, asin, acos,
cotan,sinh,cosh,tanh,cotanh,
exp, log, log10, sqrt
abs, max, min,
```

Bibliography

[1] R. B. Lehoucq, D. C. Sorensen, and C. Yang *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods* SIAM, ISBN 0-89871-407-9 // http://www.caam.rice.edu/software/ARPACK/

[2] I. Babbuška, Error bounds for finite element method, Numer. Math. 16, 322-333.

[3] Y. Achdou and O. Pironneau, Computational Methods for Option Pricing. SIAM monograph (2005).

[4] D. Bernardi, F.Hecht, K. Ohtsuka, O. Pironneau, *freefem+ documentation*, on the web at ftp://www.freefem.org/freefemplus.

[5] D. Bernardi, F.Hecht, O. Pironneau, C. Prud'homme, *freefem documentation*, on the web at http://www. freefem.fr/freefem

[6] T.A. Davis: Algorithm 8xx: UMFPACK V4.1, an unsymmetric-pattern multifrontal method TOMS, 2003 (submitted for publication) http://www.cise.ufl.edu/research/sparse/umfpack

[7] P.L. George, *Automatic triangulation*, Wiley 1996.

[8] F. Hecht, Outils et algorithmes pour la méthode des éléments finis, HdR, Université Pierre et Marie Curie, France, 1992

[9] F. Hecht, The mesh adapting software: bamg. INRIA report 1998.

[10] A. Perronnet et. al. Library Modulef , INRIA, http://www.inria-rocq/modulef

[11] A. Ern and J.-L. Guermond, Discontinuous Galerkin methods for Friedrichs' symmetric systems and Second-order PDEs, SIAM J. Numer. Anal., (2005). See also: Theory and Practice of Finite Elements, vol. 159 of Applied Mathematical Sciences, Springer-Verlag, New York, NY, 2004.

[12] F. Hecht. C++ Tools to construct our user-level language. Vol 36, N? 2002 pp 809-836, Modél. math et Anal Numér.

[13] J.L. Lions, O. Pironneau: Parallel Algorithms for boundary value problems, Note CRAS. Dec 1998. Also : Superpositions for composite domains (to appear)

[14] B. Lucquin, O. Pironneau: *Introduction to Scientific Computing* Wiley 1998.

[15] I. Danaila, F. Hecht, and O. Pironneau. *Simulation numérique en C++*. Dunod, Paris, 2003.

[16] J. Nečas, L. Hlaváček, Mathematical theory of elastic and elasto-plastic bodies: An introduction, Elsevier, 1981.

[17] K. Ohtsuka, O. Pironneau and F. Hecht: Theoretical and Numerical analysis of energy release rate in 2D fracture, INFORMATION **3** (2000), 303–315.

[18] F. Preparata, M. Shamos *Computational Geometry* Springer series in Computer sciences, 1984.

[19] R. Rannacher: On Chorin's projection method for the incompressible Navier-Stokes equations, in "Navier-Stokes Equations: Theory and Numerical Methods" (R. Rautmann, et al., eds.), Proc. Oberwolfach Conf., August 19-23, 1991, Springer, 1992

[20] J.E. Roberts and Thomas J.-M: Mixed and Hybrid Methods, Handbook of Numerical Anaysis, Vol.II, North-Holland, 1993

[21] J.L. Steger: The Chimera method of flow simulation, Workshop on applied CFD, Univ of Tennessee Space Institute, August 1991.

[22] M. Tabata: Numerical solutions of partial differential equations II (in Japanese), Iwanami Applied Math., 1994

[23] F. Thomasset: Implementation of finite element methods of Navier-Stokes Equations, Springer-Verlag, 1981

[24] N. Wirth: *Algorithms + Data Structures = Programs*, Prentice Hall, 1976

[25] Bison The GNU compiler-compiler documentation (on the web).

[26] B. Stroustrup, The C++ , programming language, Third edition, Addison-Wesley 1997.

[27] L. Deng, Wences Gouveia, COOOL: a package of tools for writing optimization code and solving optimization problems, http://coool.mines.edu

[28] B. Riviere, M. Wheeler, V. Girault, A priori error estimates for finite element methods based on discontinuous approximation spaces for elliptic problems. SIAM J. Numer. Anal. 39 (2001), no. 3, 902–931 (electronic).

[29] R. Glowinski and O. Pironneau, Numerical methods for the Stokes problem, Chapter 13 of Energy Methods in Finite Element Analysis, R.Glowinski, E.Y. Rodin, O.C. Zienkiewicz eds., J.Wiley & Sons, Chichester, UK, 1979, pp. 243-264.

[30] R. Glowinski, Numerical Methods for Nonlinear Variational Problems, Springer-Verlag, New York, NY, 1984.

[31] R. Glowinski, Finite Element Methods for Incompressible Viscous Flow. In Handbook of Numerical Analysis, Vol. IX, P.G. Ciarlet and J.L. Lions, eds., North-Holland, Amsterdam, 2003, pp.3-1176.

[32] C. Horgan, G. Saccomandi, Constitutive Models for Compressible Nonlinearly Elastic Materials with Limiting Chain Extensibility, Journal of Elasticity, Volume 77, Number 2, November 2004, pp. 123-138(16).

[33] Kazufumi Ito, and Karl Kunisch, Semi smooth newton methods for variational inequalities of the first kind , M2AN, vol 37, N°, 2003, pp 41-62.

[34] R.W. Ogden, Non-Linear Elastic Deformations, Dover, 1984.

[35] P.A. Raviart, J.M. Thomas, Introduction à l'analyse numérique des équations aux dérivées partielles, Masson, 1983.

[36] Hang Si, *TetGen Users' Guide: A quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator* // http://tetgen.berlios.de

[37] J.R. Shewchuk, *Tetrahedral Mesh Generation by Delaunay Refinement* Proceeding of the Fourteenth Anual Symposium on Computational Geometry (Minneapolis, Minnesota), pp 86–95, 1998.

[38] M. A. Taylor, B. A. Wingate , L. P. Bos, Several new quadrature formulas for polynomial integration in the triangle , Report-no: SAND2005-0034J, http://xyz.lanl.gov/format/math.NA/0501496

[39] P. Willmott, S. Howison, J. Dewynne : A student introduction to mathematical finance, Cambridge University Press (1995).

[40] P. Frey, A fully automatic adaptive isotropic surface remeshing procedure. INRIA RT–0252, 2001.

[41] Astrid Sabine Sinwel, *A New Family of Mixed Finite Elements for Elasticity* http://www.numa.uni-linz.ac.at/Teaching/PhD/Finished/sinwel-diss.pdf, Thesis, 2009, Johannes Kepler Universit鋘, Austria

[42] Steven G. Johnson, The NLopt nonlinear-optimization package, http://ab-initio.mit.edu/nlopt

[43] N. Hansen, The CMA Evolution Strategy, http://www.lri.fr/~hansen/cmaesintro.html

[44] A. W鋟hter and L. T. Biegler, On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming, Mathematical Programming 106(1), pp. 25-57, 2006

[45] A. Forsgren, P. E. Gill and M. H. Wright, Interior Methods for Nonlinear Optimization, SIAM Review, Vol. 44, No 4. pp. 525-597, 2002

Index

Book Description

Fruit of a long maturing process freefem, in its last avatar, FreeFem++, is a high level integrated development environment (IDE) for partial differential equations (PDE). It is the ideal tool for teaching the finite element method but it is also perfect for research to quickly test new ideas or multi-physics and complex applications.

FreeFem++ has an advanced automatic mesh generator, capable of a posteriori mesh adaptation; it has a general purpose elliptic solver interfaced with fast algorithms such as the multi-frontal method UMFPACK. Hyperbolic and parabolic problems are solved by iterative algorithms prescribed by the user with the high level language of FreeFem++. It has several triangular finite elements, including discontinuous elements. Finally everything is there in FreeFem++ to prepare research quality reports: color display online with zooming and other features and postscript printouts.

This book is ideal for students at Master level, for researchers at any level and for engineers also in financial mathematics.

Editorial Reviews

"...Impossible to put the book down, suspense right up to the last page..."
A. TANH, Siam Chronicle.

"...The chapter on discontinuous fems is so hilarious ...."
B. GALERKINE, Российской академии наук .