Nek5000 Primer

Version 1.4

2018 年 5 月 3 日

徐博伦
University of Science and Technology of China
Department of Modern Mechanics
Theoretical and Computational Fluid Dynamics Lab
xblkid@mail.ustc.edu.cn

摘要

这本小册子介绍了 Nek5000 中重要的用户文件的内容，常用变量的含义，设置算例的流程。同时兼顾了前处理中网格的生成，转换方法，以及后处理流场数据的第三方软件简明用法，及用户自定义的函数和小程序的写法，还有求解除一般不可压 NS 方程以外的其他问题的方法简介。希望能帮助读者较快地上手这套 code，计算处理自己的物理问题。本文中用作例子的部分代码以及图像文件可在我的个人目录：https://github.com/xubolun/xbltcfd 下找到。

目录

第一章 快速上手 Nek5000 **3**

1.1 用户文件 3

1.2 运行算例 13

第二章 前处理 **17**

2.1 genbox 生成网格 17

2.2 其他软件生成网格的信息导入 20

2.2.1 mshconvert.py 20

2.2.2 gambit to nek 22

第三章 后处理 **24**

3.1 流场可视化 24

3.1.1 visIt 24

3.1.2 tecplot 30

3.2 自定义后处理程序 34

3.2.1 By Fortran 34

3.2.2 By Matlab 40

第四章 求解其他问题 **43**

4.1 变物性参数问题 43

4.2 线性稳定性分析 44

4.2.1 基本流的求法 44

4.2.2 求解特征值问题 45

后记 **48**

更新日志 **49**

附录 A visIt 体积渲染后处理脚本 volume.py 示例 **50**

附录 B tecplot 制作动画脚本 anime.mcr 示例 **54**

附录 C Matlab 后处理程序部分示例 **56**

Nek5000 流场文件读入脚本 readnek.m/并行读入一组文件脚本 serialread.m 56

readnek.m 56

serialread.m 59

3D 环域插值脚本 intpnek.m/多文件并行插值脚本 serialintp.m 60

intpnek.m 60

serialintp.m 61

柱坐标下流场数据输出脚本 outputTec.m/并行输出多文件脚本 PoutputTec.m . . . 62

outputTec.m 62

PoutputTec.m 62

附录 D hpts.out 中流场数据提取脚本 readdata.f90 **64**

附录 E 程序编译 bash 文件 setup.sh 示例 **67**

第一章　快速上手 Nek5000

Nek5000 是使用高精度谱元法求解不可压缩流动的开源程序，由美国 Argonne 国家实验室开发，具有低耗散低色散的特点。有关该程序数值方法的内容，本手册不作介绍，可参考文献 [14]，[4] 以及 [3]。

源程序的代码包可在 Nek5000 主页上找到，截止目前的最新版本是 17.0.4 版。顺便一提，在这个主页上有一个较新版本的程序介绍文档，跟几年前发布的 PDF 版的 Nek5000 document 相比，内容更为简练，同时修正了好几处错误，也是本文写作的参考。程序包下载完后可以直接解压到 home 目录下。求解问题相关的源代码放在 core 文件夹中，tools 文件夹中是进行某些前后处理（如生成简单计算网格，网格的并行分块，网格加工等）的程序代码，examples 文件夹中则是一些标准的测试算例，这些算例的用户文件的写法是我们设置自己的算例时的一个很好的示范。这些文件夹中代码更详细的说明在后文中还会出现。其他文件夹中的内容并不是运行算例必须的，我们暂时先不管他们。

解压完以后我们先把上面提到的 tools 里的工具文件编译出来。只需运行 tools 文件夹中的 maketools 脚本就行了。输入：

```
./maketools all
```

这样该文件夹中的所有工具文件都会进行编译。编译生成的可执行文件在上一层目录的 bin 文件夹中。若不需要编译所有的工具文件，比如我们只想编译 genmap 这个工具，那么只需输入：

```
./maketools genmap
```

即可。有时我们也需要对工具文件的源代码进行修改以适应不同的需求，这样需要把之前生成的可执行文件去掉。可以输入：

```
./maketools clean
```

一次性删除所有的可执行文件。现在为止准备工作差不多做完了。

1.1　用户文件

现在我们可以开始设置自己的算例了。我们可以新建一个文件夹 test，建在任意目录下都行。如之前所说，examples 文件夹中的例子是很好的参考，初次设置 test 算例时可参照这些例子，如 RB 对流的算例（rayleigh 目录下），一个算例一般需要的文件如表1.1所示。下

图 1.1: 部分.rea 文件中控制参数.

```
 2.00000 NEKTON VERSION
 2 DIMENSIONAL RUN
       103 PARAMETERS FOLLOW
   1.000000          p01 DENSITY
   1.000000          p02 VISCOS
  0.0000000E+00
  0.0000000E+00
  0.0000000E+00
  0.0000000E+00
   1.000000          p07 RHOCP
   1.000000          p08 CONDUCT
  0.0000000E+00
  0.0000000E+00      p10 FINTIME
   75000.00          p11 NSTEPS
 -2.0000000E-02      p12 DT
  0.0000000E+00      p13 IOCOMM
  0.0000000E+00      p14 IOTIME
   1000.000          p15 IOSTEP
  0.0000000E+00      p16 PSSOLVER
  0.0000000E+00
0.0000000E+00     p21 DIVERGENCE
0.0000000E+00     p22 HELMHOLTZ
0.0000000E+00     p23 NPSCAL
1.0000000E-03     p24 TOLREL
1.0000000E-03     p25 TOLABS
 2.010000         p26 COURANT/NTAU
 2.000000         p27 TORDER
0.0000000E+00     p28 TORDER: mesh velocity (0: p28=p27)
0.0000000E+00     p29 magnetic visc if > 0, = -1/Rm if < 0
0.0000000E+00     p30 > 0 ==> properties set in uservp()
0.0000000E+00     p31 NPERT: #perturbation modes
0.0000000E+00     p32 #BCs in re2 file, if > 0
 6.000000         p66 write fmt:ONLY postx uses rea value
 6.000000         p67 read fmt: same modes as p66
0.0000000E+00     p68 iastep: freq for avg_all
13  LOGICAL SWITCHES FOLLOW
T     IFFLOW
T     IFHEAT
T     IFTRAN
T  T  F  F  F  F  F  F  F  F  F IFNAV & IFADVC (convection in P.S. fields)
F  F  T  T  T  T  T  T  T  T  T  T IFTMSH (IF mesh for this field is T mesh)
F     IFAXIS
F     IFSTRS
F     IFSPLIT
F     IFMGRID
F     IFMODEL
F     IFKEPS
F     IFMVBD
F     IFCHAR
```

表 1.1: 算例需要的用户定义文件.

test.rea	设定流动控制参数，各功能的逻辑开关，时间步，CFL 数等
test.usr	设定用户自定义的前后处理子程序等
test.box(可能需要)	用户定义的网格文件，适用于较简单几何构型
test.map	设置并行算例需要的网格分块信息文件
SIZE	设定 Legendre 多项式阶数，最大并行块数，最大网格量等

边我们分别对这些文件进行详细说明。

test.rea

一个完整的.rea 文件包含了三块信息。第一块是流动控制参数与逻辑开关。如图1.1所示：para1 是无量纲密度，一般设为 1，若是变密度的情况将在.usr 文件中另行设置。para2 是运动学粘性系数 ν，但通常我们的问题都已经无量纲化了，所以这个量通常会设置成跟粘性项前的无量纲参数有关的一个量。在 RB 对流的例子中，para2 设置成了普朗特数 Pr，并且在.usr 文件中有进一步的定义，这在后文中还会提到。更一般的情况下这个参数会设置成雷诺数 Re，并且程序中约定，该数为负值时代表 Re。para7 和 para8 分别是温度（能量）方程

$$\rho c_p \frac{\partial T}{\partial t} = k\nabla^2 T \tag{1.1}$$

中的 ρc_p 和 k，通过恰当的无量纲形式，这两项通常设置为 1。para8 设置的热传导系数也可以类似 para2，设置成某个无量纲参数，如 Peclet 数。para10 是算例运行的无量纲时长。para11 是算例运行的最大步数。如果 para10 和 para11 都不是 0，那么算例由最大运行时间 para10 来控制；如果想用最大时间步来控制算例的结束，则需要将 para10 设置成 0。para12 是时间步长，如果 para12 为正，则是可变步长，para12 代表不超过 CFL 数的最大步长，如果 para12 为负，则是固定步长。para14 是输出流场的无量纲时间间隔，para15 是输出流场的时间步间隔。与前边类似，如果两个参数均不为 0，按 para14 来输出流场，para14= 0，则按 para15 输出。para26 为 CFL 数，一般来说设置在 1 以内程序运行较为稳定，但是根据不同的问题，由于处理的都是不可压流动，CFL 有时可以放的较大，只要压力方程能收敛，并不影响计算精度。para27 是时间离散的阶数，可选择 2 或者 3。选择 2 为二阶后差的时间格式（BDF2），选择 3 则为三阶龙格库塔推进（RK3）。para30 是一个逻辑控制开关，可设置成 0 或 1。选 0 表示流动物性参数（主要是密度、粘性系数、热传导系数）为常数，选 1 则表示变物性参数的情况，需要在.usr 中对这些参数的函数表达式通过一子程序进行单独设置，这在后文中还会提到。para31 也是一逻辑变量，设置成 0 表示程序按 NS 方程进行计算，设置成 1 则是按线化后的小扰动方程来进行计算，若要进行线性稳定性分析，则需选择这一模式，同时在 SIZE 文件中也需对相应的多项式相关参数作调整，这些内容同样会在后文中

详细说明。para66 和 para67 控制读写流场文件的模式。Nek5000 输出两种格式的流场文件，分别为.fld 和. fxxxx 格式，均为二进制输出，在文件头几行的格式有些许不同，其中.fxxxx 是较新的一种格式，更利于大规模的并行读写。程序约定 para66/67 设置成 4，按照.fld 格式读写（其中读是指读入续算的流场文件），设置成 6，则按.fxxxx 格式读写。para68 则是进行流场变量时间平均的间隔步数，时间平均通过调用子程序 avg_all 来实现，默认只能平均流场输出变量速度、压强和温度，若要实现别的物理量的时间平均可以通过自己编写类似子程序去实现，这在后文中也会提到。

不难注意到，这其中有很多参数没有设置（值为 0 或是空着），还有一些有值的参数前文没有说明。没做说明的参数具体含义可参考 [10]，这些参数会在一些特殊的算例中（如 MHD）起作用，或是对计算影响很小，选为默认值即可。事实上在程序运行开始时，这些参数会一个个被读入存在全局变量 para 数组中，因此没有设置值的参数我们也可以自己随意设置其数值，并随后在.usr 里的子程序中直接调用即可，因此上一段中提到那些参数只代表是默认的含义，我们完全可以在.usr 的子程序中对他们重新定义。

第一块内容的最后部分是一些逻辑开关。这部分也可以完全参考 [10] 文档最后的附录 A，此处不再赘述。

第二块内容存储了算例的几何参数及网格点的信息。Nek5000 支持的网格（谱元）的拓扑结构为六面体（3D）和四边形（2D）两种，在.rea 文件中存储了这些网格（谱元）角点的笛卡尔坐标，若是网格（谱元）是弯曲的，还需存储弯曲的单元边界，以及弯曲的类型。这部分在 [10] 中 20 页也有较详细说明。

这块的最后一部分是存储网格每一条边的边界条件。通常边界条件信息按图1.2所示的格式存储，分为速度边界和温度边界（如果有能量方程）两个部分，现在我们对每列的数据做个说明：第一列的字符 E/B/P... 存储在 cbc 字符型数组中，表示边界条件的类型。Nek5000 支持多种边界条件，常用的类型列在表1.2 中。第二列的数字是每个谱元的序号，存储在 iel 数组中，这是全局的序号，跟后文提到并行下谱元在每个块中的当地序号应有所区分。第三列的数字是每一个谱元每一个面（二维为边）的代号，取值范围从 1-6 或从 1-4，存储在 iedge 数组中。第四列和第五列的信息是比较重要的。第四列储存的数字代表与同一行的那个面（边）相邻的谱元的编号，而第五列的数字代表该谱元上哪一个面（边）与之前的面（边）相接触。换句话说，这两类储存了不同谱元之间的关联信息，独立的每个谱元根据这些信息联系组成整个计算域。在使用周期边界 P 时，需特别关注作为该边界的面（边）之间的关联信息是否匹配。

.rea 文件的最后一块内容，比较重要的是储存续算文件的部分。如图1.3所示，没有续算文件时，presolve 行之前开关切换成 0，下一行空着。若要从如图所示的 s8_old.fldxx 文件开始续算，则将开关切换成 1，下一行输入该流场文件名即可，该行开头的空格数量，可能影响数据读入。需要注意的是前文提到的 para67 的数值也要与该续算文件类型相一致。其他的部分保持默认即可，对算例运行无影响。

图 1.2: 部分.rea 文件中边界条件信息.

CBC	IEL	IEDGE	CONN-IEL	CONN-IEDGE	redundant
E	1	1	4.00000	3.00000	0.00000
..	..	..	..	..	..
W	5	3	0.00000	0.00000	0.00000
..	..	..	..	..	..
P	5	5	149.000	6.00000	0.00000

表 1.2: 常用边界条件（注意区分大小写）.

边界条件	简介
W	无滑移速度边界条件，$\mathbf{v}=0$
v	用户自定义速度第一类边界条件，在.usr 文件子程序中定义
P	周期边界条件
t	用户自定义温度第一类边界条件，在.usr 文件子程序中定义
E	内边界，表示该边界在计算域内部
O	出流（自然）边界条件，$(-p\mathbf{I}+\nu\nabla\mathbf{v})\cdot\mathbf{n}=0$
SYM	对称（无应力）边界条件，$\mathbf{v}\cdot\mathbf{n}=0;\quad (\nabla\mathbf{v}\cdot\mathbf{t})\cdot\mathbf{n}=0$
f	用户自定义的通量（第二类）边界条件，在.usr 文件子程序中定义
I	绝热边界条件，$\frac{\partial T}{\partial \mathbf{n}}=0$
A	轴对称边界，用在二维轴对称算例中，表示该边界为一条对称轴

test.usr

.usr 是一个用户自定义的接口文件，用来写各种需要的子程序，这样编译起来比分开写各个.f 文件方便。在编译时，该 test.usr 文件中的内容会转换成 test.f 文件，与 core 文件夹中的源码一同编译。有几个运行算例所必需的子程序需要先介绍一下。

uservp：用来定义变物性参数。可以分别定义动量方程和能量方程中的物性参数的函数形式，他们可以是空间坐标的函数，也可以是流场变量的函数。这部分内容将在4.1节专门介绍。

userf：用来定义体积力项（体加速度），这一项随后会在计算中乘上密度 ρ。我们以方腔中 RB 对流（图 1.4）为例进行说明。RB 对流中的体积力项为浮力，恰当地无量纲后其表达式为 $buoyancy = RaPr\theta$；若是 3D 构型可进一步考虑旋转坐标系的情形，此时还可引入科氏力，无量纲后的形式为 $Cor_x = (Ta^{\frac{1}{2}}Pr)u_y$, $Cor_y = -(Ta^{\frac{1}{2}}Pr)u_x$，其中 Ra, Pr, Ta 都

图 1.3: .rea 文件中续算设置.

```
   1 PRESOLVE/RESTART OPTIONS  *****
s8 old.fld2049 time=0.
```

图 1.4: .usr 文件中 userf 子程序设置.

(a)

```
buoy = -temp*rapr

if (if3d) then
   ffx  = Ta2Pr*uy
   ffy  = - Ta2Pr*ux
   ffz  = -buoy
elseif (ifaxis) then
   ffx  = -buoy
   ffy  =  0.
else
   ffx  = 0.
   ffy  = -buoy
endif
```

(b)

```
common /rayleigh_r/ rapr,ta2pr

real theta,pi
theta =3
pi=3.1415926


theta = theta/180*pi
buoy =temp*rapr
if (if3d) then
   ffx  =   uy*Ta2Pr
   ffy  = - ux*Ta2Pr
   ffz  = buoy
elseif (ifaxis) then
   ffx  = -buoy
   ffy  =  0.
else
   ffx  = buoy*(-sin(theta))
   ffy  = buoy*cos(theta)
endif
```

是无量纲参数。我们在程序中先定义好这些力。随后分三种情况——3D、2D 轴对称、2D 分别设置体积力。顺便一提，Nek5000 中默认 y 方向是轴对称构型中对称轴的方向，设置体积力分量的时候要注意这点。图1.4(b) 中是 2D 倾斜 RB 设置的情形，倾角为 3 度，体积力为水平和数值方向上的浮力分力。可以看到类似的体积力设置是很方便的。

userq：用来定义热源项。qvol 为热源变量。该子程序并没有使用过，还有待探索。

userbc：用来定义各种边界条件。图1.5是 RB 对流定义边界条件的一个例子，该例子侧壁为绝热壁，上下壁为等温壁，上壁温度恒为-0.5，下壁恒为 0.5，所有壁面速度为 0。结合前文介绍的.rea 文件中的边界条件信息，我们只在该子程序中定义小写的字母表示的边界条件即可（如 v、t、f 类等）。若是需要热通量边界，则通量的值储存在 flux 变量中，该边界条件的使用还没有测试过。

uselic：用来定义初始条件。图1.6是 RB 对流定义初始条件的一个例子，初始时刻流场静止速度均为 0，温度在垂直方向上呈线性分布，我们加上一随机温度场的小扰动使流动更快发展。如果是续算，那该子程序将不会被调用，直接从前文提过的.rea 文件中记录的续算文件里读入流场作为初始条件。

userdat2：可以用来重新定义某些谱元的拓扑结构，比如原先给定的边界是直线，可在这个子程序中将这些边界重新定义成某个函数形式，如正弦型，这部分内容可以参考 [10] 中 29 页的示例，这里不再赘述。个人认为这种方法不是很实用，如果计算域构型稍微复杂，边界无法用函数形式显式写出，这种处理就不管用了。这时在画网格时就画好需要的构型，再用网格转换代码将网格信息导入.rea 文件更加方便，后文中还会再提到。另外，包括没有单独列出的 userdat/userdat3，这些子程序都只在时间推进开始前调用一次而已，因此我们还可以在这些子程序中定义一些全局变量，比如无量纲参数的值，还有控制输入输出的逻辑变量的值等。这里提一下，前文提到.rea 文件中定义的参数（para 开头的）均储存在全局数组

图 1.5: .usr 文件中 userbc 子程序设置.

```
ux=0.
uy=0.
uz=0.

temp=0.

if (if3d) then
   temp = 0.5 - z
elseif (ifaxis) then
   temp = 1.-x
else
 if(y.gt.0.5)then
   temp = -0.5
 else
   temp= 0.5
 endif
endif
```

图 1.6: .usr 文件中 useric 子程序设置.

```
ran = ran1(idum)
ran = 1.e9*sin(ran)
ran = 1.e9*sin(ran)
ran = cos(ran)

amp = .005

temp = 0.5-z + ran*amp*(1-x)*z*x
ux=0.0
uy=0.0
uz=0.0
```

表 1.3: 部分常用解变量（数组），其中 $lorder2 = max(1, lorder - 2)$, lorder 为时间离散阶数. 其他表征数组大小的量在 SIZE 部分介绍.

变量名	数组大小	数组类型	简介
vx/vy/vz	(lx1,ly1,lz1,lelv)	real	x/y/z 方向速度
vxp/vyp/vzp	(lpx1,lpy1,lpz1,lpelv,lpert)	real	线化方程 x/y/z 方向小扰动的速度
uavg/vavg/wavg	(ax1,ay1,az1,lelt)	real	时均的 x/y/z 方向速度
urms/vrms/wrms	(ax1,ay1,az1,lelt)	real	时均的 x/y/z 方向速度平方值
pr	(lx2,ly2,lz2,lelv)	real	压力
t	(lx1,ly1,lz1,lelt,ldimt)	real	温度 (ldimt= 1) 和其他标量 (ldimt> 1)
tavg	(ax1,ay1,az1,lelt,ldimt)	real	时均的温度 (ldimt= 1) 和其他标量 (ldimt> 1)
trms	(ax1,ay1,az1,lelt,ldimt)	real	时均的温度 (ldimt= 1) 和其他标量 (ldimt> 1) 的平方
vxlag/vylag/vzlag	(lx1,ly1,lz1,lelv,2)	real	前两个时间步的 x/y/z 方向速度
prlag	(lx2,ly2,lz2,lelv,lorder2)	real	前几个时间步的压力
tlag	(lx1,ly1,lz1,lelt,lorder-1,ldimt)	real	前几个时间步的温度 (ldimt= 1) 和其他标量 (ldimt> 1)
bfx/bfy/bfz	(lx1,ly1,lz1,lelv)	real	x/y/z 方向的体积力
vtrans	(lx1,ly1,lz1,lelt,ldimt1)	real	动量/能量方程中对流项系数
vdiff	(lx1,ly1,lz1,lelt,ldimt1)	real	动量/能量方程中扩散项系数
time	N/A	real	时间推进时的物理时间
dt	N/A	real	时间步长
istep	N/A	integer	时间步数

表 1.4: 部分常用计算域几何构型变量（数组）.

变量名	数组大小	数组类型	简介
xm1/ym1/zm1	(lx1,ly1,lz1,lelv)	real	速度网格上的 x/y/z 空间坐标
xm2/ym2/zm2	(lx2,ly2,lz2,lelv)	real	压力网格上的 x/y/z 空间坐标
bm1/binvm1	(lx1,ly1,lz1,lelv)	real	速度网格上的质量（权）矩阵/其逆矩阵
unx/uny/unz	(lx1,ly1,6,lelv)	real	谱元面单位法向量的 x/y/z 分量
area	(lx1,ly1,6,lelv)	real	谱元面面元大小（面积分中的权）
volvm1	N/A	real	速度网格的总体积

表 1.5: 部分常用程序运行控制变量（数组）.

变量名	数组大小	数组类型	简介
nid	N/A	integer	MPI 并行块的 ID
nelv/nelt	N/A	integer	（某一并行块上）速度/温度网格内的谱元数
ndim	N/A	integer	问题的维数（取值为 2 或 3）
nsteps	N/A	integer	当前程序运行的步数
iostep	N/A	integer	输出流场文件的步数间隔
cbc	(6,lelt,ldimt1)	character*3	边界条件类型

表 1.6: SIZE 文件中部分常用全局变量.

变量名	简介
ldim	取值为 2 或 3，问题的维数
lx1/ly1/lz1	每个谱元内 x/y/z 方向多项式插值点的数量，一般三个方向取相同的值，控制解的空间阶数，空间阶数 N =lx1−1
lxd/lyd/lzd	控制空间积分的阶数，一般三个方向取相同的值，严格来说需满足 $lxd = 3lx1/2$ 但通常取值可以小一些，另外取偶数较好（经验）
lx2/ly2/lz2	压力的解精度，取值为 =lx1 or =lx1−1，这取决于用 $P_n - P_n$ 还是 $P_n - P_{n-2}$ 谱元法解 NS 方程
lelg	问题的最大谱元数
lelt	每个并行块上允许的最大谱元数
lhis	调用输出某点流场数据的函数时，监测点的数量上限

param 中，因此在这些子程序中定义时可直接对 param 数组中的元素赋值。常用的有前文提到过的控制输入输出文件的格式（.fld/.fxxxx）的开关 para66/para67，在续算时可能读入文件和输出的文件格式不同，这时就需要在 userdat2 中重新给 param(66)/param(67) 赋值了。

userchk: 用来调用后处理中需要用到的各种子程序和函数，可以看成后处理的 main 函数。该子程序在时间推进开始前调用一次，随后在每个时间步结束后都调用一次。这个子程序是比较重要的，因为我们通常要在计算过程中对流场数据做一些数学分析，自己写的各种做数据处理的程序就在这个子程序中调用，更多的例子我们放在附录中。需要着重说明的是，在这个子程序中使用的储存流场变量的数组（名）和前面介绍过的 uservp/userf/userbc/useric 中的那些数组（名）是不一样的，在之前那些子程序中，我们用局部实型数组变量 x、y、z、ux、uy、uz、temp 去分别储存空间坐标，速度分量和温度，而在 userchk 中，我们常用的全局数组如表1.3, 1.4, 1.5所示，基本在 core/SOLN 文件中进行了变量声明，这些量在我们自己写后处理程序时是很有用的。

之前提到过在.usr 中还能编写各种自己需要的子程序，这样做的好处是编译的时候不需要额外的命令，在编译.usr 文件的时候就将自己的子程序一道编译了，坏处是.usr 往往写的很长（如果需要的功能很多的话）。当然可以将那些子程序单独写到不同的.f 文件中，不过那样在编译的时候需要在 makefile 中加上额外的一些命令，这在之后还将提到。孰优孰劣，交与读者判断。

SIZE

SIZE 文件用来定义计算问题的大小，这包括了：每个谱元内的插值点的数量，这关系到计算的空间精度；每个并行块上的最大谱元数量；程序中用到的所有矩阵数组的维度及每一维元素的上限，这关系到数组是否越界，等等。事实上 SIZE 文件跟 core 文件夹中作其他全局变量定义的头文件并没有太大区别，只不过 SIZE 中定义的变量往往随计算的问题的不同而变化，因此为了方便把它单独拿出来放在算例的文件夹中去编译。该文件常用的参数我们列在了表 1.6中。

如果用并行计算，我们还需要给出网格在并行块上的分配方式，这部分信息储存在.map 这一二进制文件中。Nek5000 的分块方式由于谱元法的特点，是可以比较随意的，即每个块上的谱元数不必相同，尽管这样可能增加块之间交换信息的时间负担。我们用 genmap 这一内置工具来根据.rea 文件中记录的网格信息来生成.map 文件，该工具可在之前介绍过的 bin 目录下找到。比如我们现在的 test 目录下已经有了 test.rea 文件，直接输入：

```
./genmap
```

随后输入.rea 文件的名字：

```
test
```

再输入划分时的块与块之间容许误差，这个值取得较小即可，默认值是 0.2。这样就生成了相应的 test.map 文件，至于 genmap 脚本中使用的方法我们不去细究。至此我们介绍完了每个算例文件夹中大概需要的用户文件。

1.2　运行算例

接着上一章的内容,假设现在在 test 目录下我们已经准备好了 test.rea, test.usr, test.map, SIZE 这几个文件，接下来我们就要编译和运行这个算例。为此，我们还需要几个脚本文件。

首先是负责编译的编译的脚本 makenek，可以在 core 目录下找到，我们把它复制到 test 目录下。现在我们可以先了解一下 makenek 脚本中的内容，如图 1.7(a) 所示。文件中首先记录代码的版本号。随后是编译的一些基本规则。需要注意的是关于 mpi-fortran 和 mpi-C 的那两行，图示均加上了“-mcmodel=large ”，默认是没有的，这是为了使得为存储变量矩阵分配的内存空间能大于默认值 2G，在编译较大网格的问题时经常用到。其他的一些编译规则记录在 core 目录下的 makenek.inc 文件中，通常保持默认即可。现在我们到 test 目录下，输入：

```
./makenek test
```

即开始编译。编译过程中会依据 test.usr 的内容生成完全一样的.f 文件，core 目录中的源码则生成相应.o 文件，放在新建的 obj 目录下，同时会生成 compile.out 文件，记录编译的过程。编译成功，屏幕上会出现“compilation successful”字样，同时生成了算例的可执行文件 nek5000。

图 1.7: makenek 文件中部分内容.

(a)

```
set -e

# assign version tag
mver=17.0.4

# set defaults
: ${SOURCE_ROOT:="$HOME/Nek5000"}
: ${FC:="mpif77 -mcmodel=large -shared-intel"}
: ${CC:="mpicc -mcmodel=large"}
: ${MPI:=1}
: ${PROFILING:=1}
: ${VISIT:=0}
: ${PPLIST:=""}

# overwrite source path with optional 2nd argument
if [ -d $2 ] && [ $# -eq 2 ]; then
  SOURCE_ROOT="$2"
  echo "change source code directory to: ", $SOURCE_ROOT
fi

# create makefile
source $SOURCE_ROOT/core/makenek.inc

# compile 3rd_party code
make_3rd_party 2>&1 | tee -a compiler.out

# compile nek
make -j4 -f makefile 2>&1 | tee -a compiler.out

exit 0
```

(b)

```
# auxilliary files to compile
# NOTE: source files have to located in the same directory as makenek
#       a makefile_usr.inc has to be provided containing the build rules
USR="IO_tools.o uparam.o"
```

(c)

```
# linking flags
USR_LFLAGS="-L./ -lparpack_MPI-gcc_rst -larpack_gcc_rst"
```

图 1.8: makenek_usr.inc 文件中内容.

```
SRC_LOC := ./src
$(OBJDIR)/IO_tools.o        :$(SRC_LOC)/IO_tools.f;        $(F77) -c $(FL2) -I./src $< -o $@
$(OBJDIR)/uparam.o          :$(SRC_LOC)/uparam.f;          $(F77) -c $(FL2) -I./src $< -o $@
```

按这样的编译方式，会将 test.usr 中用户自定义的所有子程序全部编译。前文提到，为了使代码更简短易读，也可以将那些子程序写在单独的.f 文件里。如果采取这种方式，编译时需在 test 目录下创建额外的 makenek_usr.inc 文件，比如图 1.8所示，我们不妨将这些自定义.f 文件放在 test 下新建的 src 文件夹中，这里示例的.f 文件为 IO_tools.f 和 uparam.f 两个。同时在 makenek 文件中，将图1.7(b) 中所示部分加上“IO_tools.o uparam.o”。这样再编译即可。

有时在计算问题时还需要调用某些外部的库函数，来实现一些 Nek5000 中没有的功能。一个常见的例子是研究线性稳定性时，需要求解优势特征值，我们通常使用成熟的 Arpack 库。我们先单独编译 Arpack，编译成功后会生成两个可执行文件“larpack_gcc_rst.a”和“larpack_MPI-gcc_rst.a”，先将其复制到算例目录下（不妨还假设是 test），再如图1.7(c) 所示将外链文件名写好，这样就能在计算时调用 Arpack 包中的函数了。当然，使用时我们还需在.usr 中去写调用的接口程序，在这里我们先略去不表。

接下来可以去运行该算例了。我们仍需要 core/bin 目录下几个脚本文件：nek、nekb、nekmpi、nekbmpi。前两个表示串行运行，后两个表示并行。nek 和 nekmpi 运行时输出信息会直接显示在屏幕上，而其它两个会显示在生成的 logfile 中。若串行，我们输入：
./nek test （或./nekb test），
若并行，我们输入：
./nekmpi test $NP （或./nekbmpi test $NP），$NP 代表需要的核数。
这样算例就开始计算了。由于使用谱元法，程序在分配块上可以比较随意，选择核数不必是网格数量的约数，但是为节省计算时间还是需要考虑一下用多少核较合适，这通常是依赖于经验，可以先小规模试算来找到较理想的核数。

图 1.9: Nek5000 作业脚本文件示例.

```
#!/bin/bash
#PBS -N test
#PBS -o job.log
#PBS -e job.err
# #PBS -q norml
#PBS -l nodes=2:ppn=28
cd $PBS_O_WORKDIR
echo Running on hosts `hostname`
echo Time is `date`
echo Directory is $PWD
echo ann      >  SESSION.NAME
echo `pwd`'/'     >>  SESSION.NAME
touch test.rea
echo This job runs on the following nodes:
cat $PBS_NODEFILE
NP=`cat $PBS_NODEFILE | wc -l`
echo This job has allocated 1 node
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/data/software/openmpi-2.0.1/lib
#export PATH=$PATH:/data/software/openmpi-2.0.1/bin
mpirun -np $NP -hostfile $PBS_NODEFILE ./nek5000 > out.log
```

在集群系统中，按上述方式提交算例是直接在头结点上运行，这是不允许的。因此我们需要适用于 Nek5000 的作业脚本文件，如图1.9所示，该图中文件适用于 202.38.72.182(Rochester) 集群，其他集群也是大同小异。

计算中生成.fld 或.fxxxx 格式的流场文件，我们用 visIt 软件来打开进行可视化处理。在

此之前需要使用 core/bin 目录下脚本 visnek，输入：
./visnek test
即可生成包含流场文件信息的.nek5000 格式文件，可将这个文件和所有流场文件一同拷贝到本地，再用 visIt 打开即可。

至此一个算例的基本设置和运行就介绍完了。

第二章　前处理

这章介绍前处理相关内容，主要关心的是网格问题，包括通过 Nek5000 生成网格和其他格式网格文件的导入等。如上一章中介绍的，问题的网格信息写在.rea 文件中，对任意几何构型的问题，如何获取这些信息是我们在这一章要解决的。

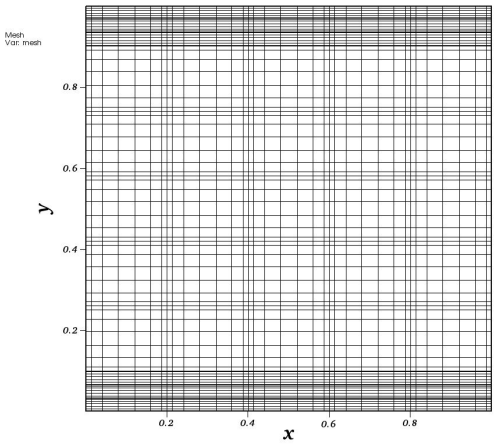
2.1　genbox 生成网格

图 2.1: 2D 方腔 RB 对流.box 文件及生成网格示例.

(a)

```
mold.rea
2          spatial dimension
2          number of fields
#============================
box 1
-5 -5             Nelx   Nely
0.0 1.0 1.0       x0 x1 ratio
0.1 0.9 1.0       y0 y1 ratio
W  ,W  ,E  ,E  ,    v BCs
I  ,I  ,E  ,E  ,    t BCs
box 2
-5 -3             Nelx   Nely
0.0 1.0 1.0        x0 x1 ratio
0.0 0.1 1.05       y0 y1 ratio
W  ,W  ,W  ,E  ,    v BCs
I  ,I  ,t  ,E  ,    t BCs
box 3
-5 -3             Nelx   Nely
0.0 1.0 1.0        x0 x1 ratio
0.9 1.0 0.952       y0 y1 ratio
W  ,W  ,E  ,W  ,    v BCs
I  ,I  ,E  ,t  ,    t BCs
```

(b)



Nek5000 代码中有自带的生成网格信息的工具 genbox，这一节我们介绍下该工具的用法，官方的文件 [10] 对这部分讲的也比较详细，可以对照查看。首先我们需要两个文件：包含网格几何尺寸边界条件的文件.box，和包含.rea 文件中除网格信息外其他内容的模板.rea 文件，不妨命名为 mold.rea。先以 2D 方腔 RB 对流为例，图2.1(a) 是该算例的一个.box 文件。首先将模板.rea 文件名写在第一行。第二行为问题的维数。第三行为处理的场的个数，分为速度场和其他标量场（如温度场）。接下来是网格信息部分。在该例中，我们在 1:1 方腔上下壁面附近（$y = 0\ 0.1/0.9\ 1$）处加密网格，因此将整个网格分成 3 块。每个 box 中第一行为 x，y 方向上的网格（谱元）数，前边的负号表示让 Nek5000 自动去分配网格点的位置，

正号则需要用户手动把每个网格点的坐标输在下两行。下边的内容则是该块网格 x，y 方向的起止位置，以及伸缩的系数，x 方向为右边网格比上左边网格，y 方向则是上层网格比下层网格，前面说的自动分配即是按这样的规则进行。需要注意网格边界条件的给法，规定好 x，y 正向后，如该例中的矩形区域，依次按左，右，下，上的顺序（默认右手系）来排列边界条件（即 x0、x1、y0、y1 的次序），并且之前说过存放边界条件字符串长度为 3，因此写完如“W”类条件需加两个空格占满 3 个长度，否则无法正确读入。图2.1(b) 是生成的网格，已经包含了谱元内多项式的插值点（7 阶精度）。三维方腔网格的.box 文件也是类似的，注意边界条件的安排次序为 x0、x1、y0、y1、z0、z1，其他设置在这里不再赘述。

有了.box 文件和 mold.rea 文件，我们将 core/bin 中的 genbox 复制到算例目录下，输入：

./genbox

提示输入.box 文件名，再输入：

rb.box

即生成了新的包含网格信息的 box.rea 文件，我们就可以用该文件来进行参数输入了，注意保持该文件与.usr 文件的名字一致。按照将计算域划分成若干规则矩形的思路，我们还能用 genbox 生成边界稍复杂一些的网格，如城墙状的粗糙边界。但是随着复杂度的增加，所需要的矩形块的数量也大大增多，该方法显然不适用了。

Nek5000 还有用于预处理的工具 prenek，可用于简单地修正一些网格参数，辅助设置边界条件，以及不同块的网格拼接，这经常用于多连通域的网格，用来处理绕流问题。下面以 2D 的圆柱绕流网格为例进行介绍，该问题左边界为第一类边界条件，右边界为出流边界（“O”型），上下为周期边界。这里要注意的是，prenek 需要可以输出图像的输出端，因此直接在 ssh client 中执行这一命令是无效的，我们需要用其他软件（如 TigerVNC）去进入 centOS 系统的桌面来进行操作。这部分准备工作可参考其他资料，这里不再赘述。

图 2.2: fpcyl.box 网格文件.

```
Box
56  26                          nelx,nely,nelz for Box)
-15 -13 -11 -9 -7 -5 -3.5 -2.5 -1.7 -1.2 -.9 -.6 -.3 0 .3 .6 .9 1.2 1.6 2
2.5 3 3.6 4.2 4.8 5.4 6 6.8 7.6 8.2 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32 33 34 35        x0...x_56
-15 -13 -11 -9 -7 -5 -3 -2.3 -1.7 -1.2 -.9 -.6 -.3 0 .3 .6 .9 1.2 1.7 2.3
3 5 7 9 11 13 15 y0...y_26
v  ,O  ,   ,   ,          bc's
```

由于 genbox 不能直接生成圆柱面附近的网格，我们先用后文中2.2.1小节的方法生成一个圆柱附面网格，将其信息存储在 import.rea 中（之所以不用该方法直接生成全场网格是因为在赋周期边界时有些 bug 没有解决）。再写一个包含全计算域网格信息的矩形网格文件 fpcyl.box，如图2.2所示，这里采用手动排布了所有网格点的坐标，可以对比与图2.1(a) 中设置的区别，同时需注意前面的附面网格边界跟这个大的网格中内部的某些网格内边界信息要一一对应（比如：内网格的区域为 1.5×1.5，边界上的网格布置为 5×5，那么外网格中

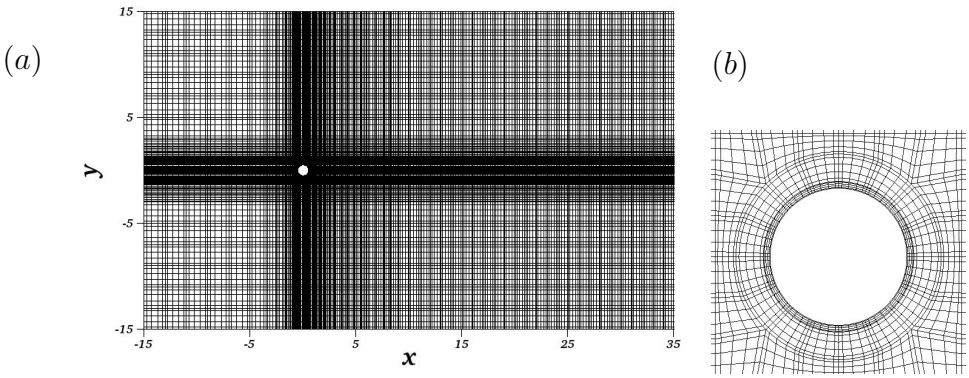
1.5 × 1.5 的区域中的网格也必须布置成 5 × 5，且网格点坐标完全一致）。使用 genbox 将 fpcyl.box 转换成 box.rea 文件，现在目录下有 import.rea（内）和 box.rea（外）两个网格数据文件，下面开始使用 prenek。

将 core/bin 中的 pretex 工具复制到目录下，输入：

./pretex
new ! 输入最终想要得到的.rea 文件名称
1 READ PREVIOUS PARAMETERS ! 读取 new.rea 文件中所需要的头部参数
box ! 输入读取头部参数的.rea 文件（不带后缀名）
1 BUILD FROM FILE ! 从文件进行读取网格数据
box ! 外网格的文件名称
10 IMPORT MESH ! 选择嵌入网格
import ! 存有嵌入（内）网格的.rea 文件名称
y ! 选择将外网格中与嵌入网格重叠的部分去掉
1 END ELEMENTS ! 结束网格的划分
1 ACCEPT MATL,QVOL ! 接受上述划分
1 SET BCs ! 进行边界条件的设置
11 SET ENTIRE LEVEL ! 从整体进行设定（因为周期边界条件需要将边界进行匹配）
1 PERIODIC-AUTO ! 进行周期边界的自动设定
n ! 此网格不是六角密堆积的网格
1 END LEVEL ! 结束边界条件的设置
1 ACCEPT B.C.'s ! 接受边界条件的设置
1 EXIT ! 结束网格的设置

这样我们就生成了需要的圆柱绕流网格，如图2.3(a) 所示，图 (b) 是圆柱附近的放大。上述

图 2.3: 2D 圆柱绕流网格及圆柱附近放大图.



步骤也可以写在脚本中，如 examples/ext_cyl/mkmesh 文件中所示，这样可以直接运行该脚本完成上述步骤。总的来说 prenek 的使用过程较为繁琐，这里只是提供一种思路，推荐复杂的计算域用2.2 一节中的方法去生成网格。

2.2　其他软件生成网格的信息导入

鉴于 genbox 只能生成拓扑结构简单的网格，我们有必要寻找别的专业网格软件以用于计算各类问题，而如何将那些软件生成的网格文件信息导入到.rea 文件中为求解器所识别是关键。网上已有开源的两种网格转换脚本，分别是 mshconvert 和 gambit to nek，经过较少的修改，就能适用于我们的问题。前者为 python 脚本，将.msh 格式的网格文件转换成.rea 文件；后者为 matlab 脚本，将.neu 格式的网格文件转换成.rea 文件。这两种格式的网格文件用 gambit 软件均可生成，但该软件由于很久之前就停止更新，在 gui 和一些处理拓扑结构的操作上用户友好度较低，因此我们通常先使用 pointwise 软件画好计算域的拓扑结构，再布置上简单的网格，直接将这个较糙的网格输出成.nas 格式文件，方便 gambit 打开它。随后再用 gambit 对该网格进行精加工（如边界层的加密），并将各个面（边）的边界条件设置好，最后再输出成.msh 或.neu 格式。下面介绍一下这两个脚本的使用方法。这里视作读者已了解这两个软件画网格的一般流程，只在特别要注意的地方才介绍在两个软件中画网格时的相应知识。

2.2.1　mshconvert.py

首先是 mshconvert 脚本。centOS 系统一般自带 python2.6 编译环境，但运行该脚本我们需将 python 升级到 2.7 以上。文件头几行 import 了许多库，这些数学/图形库的安装配置可参考网上相应教程。我们用 ipython 命令来运行该脚本（ipython 也需提前安装），使用的 rb2d.msh 文件中仍然是 2D 方腔 RB 对流的网格信息。输入：

ipython

此时跳出 ipython 文字界面，输入：

from mshconvert import *

该命令将脚本中的所有库文件导入，再输入：

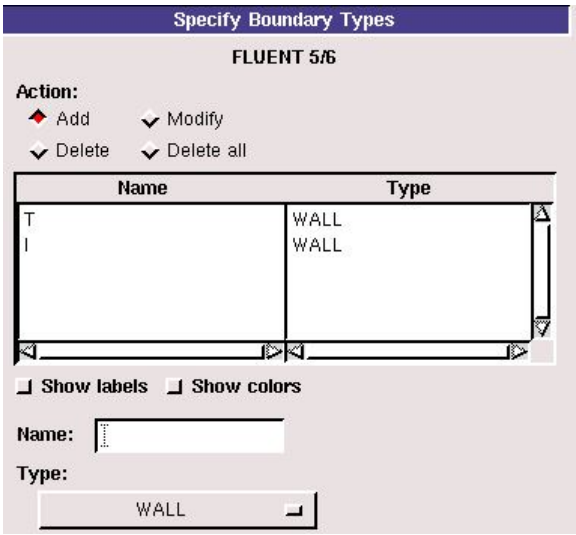
convert('rb2d.msh',bcs={3:'W',4:'W'},temperature={3:'I',4:'t'})

即可生成了相应的 rb2d.rea 文件。我们将除网格文件外的.rea 中其他内容的模板信息写在 mshconvert.py 中，因此还需根据不同的问题对生成的.rea 文件中的参数进行修改。现在我们再来看转换网格命令的含义，同时可以参照.py 文件中的注释。这条命令主要用来给边界条件和弯曲边界信息，该例为方腔故没有弯曲边界的内容。两个大括号，前一个设定速度边界，后一个设定温度边界，若还有别的标量场需要边界条件则还需在温度大括号后添加。括号中的数字代表了边界的类型，这需要我们 gambit 中提前设定好。如图2.4(a) 所示，在 gambit 中我们将方腔边界分为两类：上下边为等温壁命名为 T，侧壁为绝热壁命名为 I。输出到.msh 文件，光标移至文件末尾，可见如图2.4(b) 所示，倒数二三行记录了命名过的两类边界的序号，I 对应 3，T 对应 4，这就是我们要输入到转换命令中的序号了。

再看一个有弯曲边界的例子：2D 圆环计算域。考虑一个 2D 圆环中的热对流问题，内

图 2.4: gambit 对网格边界设置及生成.msh 文件中相应部分.

(a)



(b)

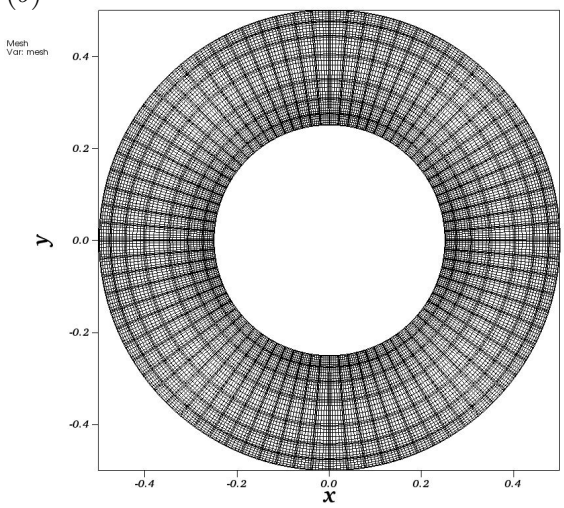
```
(0 "Zones:")
(45 (2 fluid fluid.3)())
(45 (3 wall I)())
(45 (4 wall T)())
(45 (6 interior default-interior)())
```

图 2.5: 2D 圆环网格.msh 文件及实际计算中的网格示例.

(a)

```
(0 "Zones:")
(45 (2 fluid fluid)())
(45 (3 wall tin)())
(45 (4 wall tout)())
(45 (6 interior default-interior)())
```

(b)



外环面为等温面，并具有一定的温差，重力指向 y 负方向。在 gambit 中我们先将内外边界区分好，输出到 rb_ann.msh 文件中如图2.5(a) 所示，立刻可以知道内边界序号为 3，外边界序号为 4。再次运行.py 脚本，输入：

convert('rb_ann.msh',bcs={3:'W',4:'W'},temperature={3:'t',4:'t'},
curves={4:{'type':'C','depth':3,'circle_center':(0,0),'radius':1},
3:{'type':'C','depth':4,'circle_center':(0,0),'radius':-0.5}})

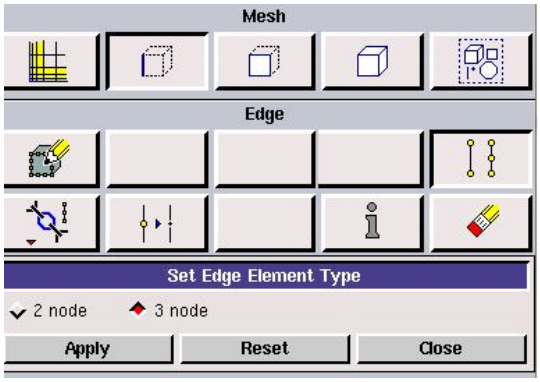
即生成了相应的 rb_ann.rea 文件，网格如图2.5(b) 所示，同样已经包含了谱元内的插值点信息（七阶精度），可以正常计算。参照该图我们可以回看转换命令中 curves 部分的含义：type 表示曲线（面）的类型，C 表示曲线（面）为圆周（球面）。若是这种情况，我们就需要指定圆（球）心位置，如本例中是 $(0,0)$。还需设置半径，本例中外径为 1，内径为 0.5，需要注意的是多连通域的情形，曲边凹的一侧为计算域之外，则这条边的半径前需加上负号来区分，如本例命令中的内径就为 -0.5。还可以设定中点型 m 类曲线，生成的是两个相邻网格点及其中间点构成的样条曲线，只要网格点较密，这样的曲线光滑度也是足够的，所以这里有个小技巧是在 pointwise 中粗设网格的时候将曲边上的点设密一些，这样输出的几何文件曲边就会比较光滑，再在 gambit 中把最终需要的网格数给定下来。depth 表示从边界开始算起有几层网格需要弯曲，如2.5(b) 所示径向有 7 层网格，均是弯曲的。我们设定的是从外边界往内数 3 层，内边界往内数 4 层，由于是圆域，每个弯曲的网格曲率一致，只需要保证内外的 depth 之和等于 7 即可。如果是更复杂的曲线，该方法可能会很麻烦或不再适用，这也是该脚本的一个缺点。

三维圆柱类的网格也可以用类似方法来生成，由于还没有测试过，具体细节还需读者自行探究。

2.2.2　gambit to nek

图 2.6: 3D 方腔网格.msh 文件及 rea_write.m 中输入参数部分.

(a)



(b)

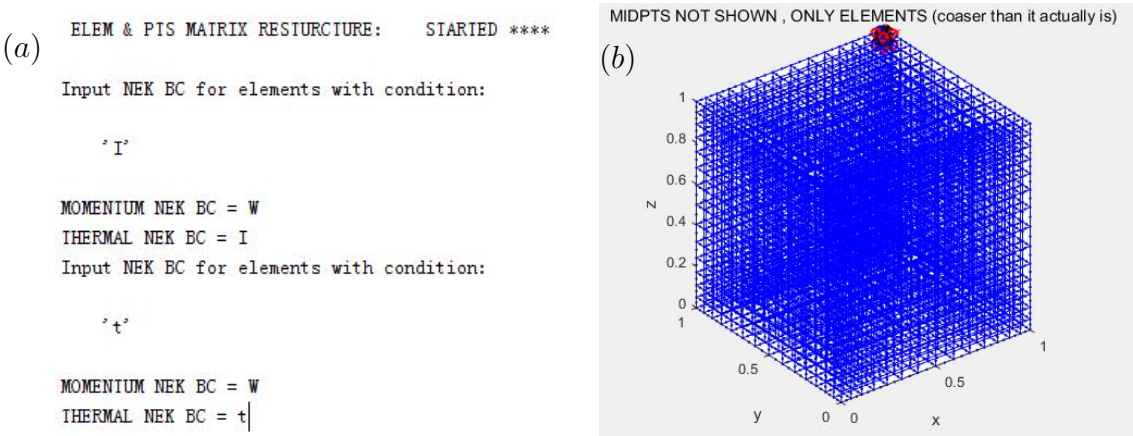
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          INPUTS REQUIRED
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fid = fopen('rb3d.neu','r');
Dim = 3;                          % CURRENTLY ONLY TESTED FOR 3D MESH!!
unit= 1; % Gambit outputs Pts in millimeters. This for unit inches
Rlimit = 20;            % Max Radius allowed for curved side consideration
IBC = 1;                                  % 1 if Thermal Boundary Conditions
draw = 1;                                                   % 1 to plot mesh
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

接下来介绍 matlab 网格转换脚本 gambit to nek。首先该脚本只适用于 3 维网格，且转换大网格可能需要较长时间，但其用户界面比较友好，不用输入如 mshconvert 中那样长长

的命令，同时拓扑结构更复杂的计算域（如具有较复杂的曲面边界）也可以处理。下面以 3D 方腔 RB 对流问题为例。

与之前不同，由于最后需要得到.neu 格式文件，在用 gambit 处理网格时需先把 solver 选成 generic 类。还需注意的是该脚本生成的弯曲边界均是 m 类，即需要网格点之间的一个虚拟格点。因此在 gambit 中，还需先将边网格点的类型选成 3 点形式，如图 2.6(a) 所示。最后需要注意坐标轴的方向，该脚本转换时会将计算域绕 x 轴旋转 90 度，即将 y 轴和 z 轴互换，这部分还有待在源码中修改。因此在 pointwise/gambit 中画几何构型时就提前将 y 轴和 z 轴交换，转换后坐标轴设置就跟预想一样，否则计算时边界条件设置就可能出现错误。进行完这些设定后即可按通常画网格的步骤操作，最后也同样将不同类型的边界做好分类，方便设置不同边界条件。这里将四个侧面设置为 I，上下底面设置为 t。注意最后的 RB 算例计算时，垂直上下底面的方向为 z 方向，因此在 gambit 中我们将垂直 y 轴的面设置成上下底面。完成上述步骤后我们得到了 rb3d.neu 文件。

图 2.7: 运行 rea_write.m 的输入和输出.



找到文件夹中 rea_write.m 文件，图2.6(b) 中是该文件输入参数设置部分。首先设定了输入的网格文件名，维度。unit 用来进行单位的转换，能让转换后的网格坐标比原先文件按一定比例放大（缩小），由于我们的问题都无量纲化过了，设成 1 即可。Rlimit 用来识别弯曲边界，曲率半径超过该值的边界将归为直线（平面），因此不妨设大一些。TBC 是个逻辑变量，判断是否需要温度边界，这里设成 1 了。draw 同样为以逻辑开关，判断是否转换完毕后画出网格构型，这里也设成 1 了。设置完毕后运行它，如图2.7(a) 所示，按屏幕提示信息，为之前设定好的不同类型边界（I 和 t 两类）分别设置速度和温度边界，可以看到步骤非常简单。转换完毕后的网格大概形状如图2.7(b) 所示，若网格量较大，用 matlab 去画出网格形状是相当耗费时间的，因此建议将 draw 开关设成 0。转换后的网格信息存在 mesh.dat 中，这些内容即是.rea 文件中后两部分的内容，只需补上第一部分参数设置的内容即生成了需要的.rea 文件。

本章介绍的这两种方法，已然可以解决大部分结构的网格转换问题。没能探究的细节部分，以及其它的网格信息读入方法，留待日后补充完善。

第三章　后处理

这章主要分为两块内容来介绍：一部分是输出流场的可视化，主要使用 visIt 和 tecplot 两个软件；另一部分是分析流场信息的各种自定义后处理程序，这些程序可以是直接写在.usr 中的 fortran 程序，也可以是其他语言（如 matlab）编写的。由于针对的问题不同，自定义程序的自由度很大，因此以举我们使用过的例子为主，希望能为读者编写自己的后处理程序提供示范。

3.1　流场可视化

前面已经提到过，可以通过设定输出时间间隔或时间步间隔，让程序生成.fld 或.fxxxx 格式的流场文件来进行可视化，这些文件包含的是直接流场变量（即 N-S 方程组的解变量）：三个方向的速度分量，压力，温度及其他标量场。而通常我们需要对间接的变量（如涡量，动能等）去做流场可视化，那应该如何生成相应流场文件？假定我们已在后处理程序中得到了涡量各方向分量 vorx，vory，vorz，动能 Ek，现在要输出它们，那么需在.usr 文件的 userchk 子程序中调用 Nek5000 内置子程序 outpost：

call outpost（vorx,vory,vorz,pr,Ek, 'pos'）

即可生成包含涡量和动能场的流场文件 posXXX.fld/fxxxx。outpost 的功能就是将原本默认输出的直接变量给替换成用户自定义的间接变量，需要注意的是不要将定义在速度网格上的变量写在第四个位置，如本例中存放压力（pr）的位置上，因为空间通常采用 $P_n - P_{n-2}$ 型谱元法，速度和压力网格阶数不同，会产生输出错误。

有了流场文件，下面我们来介绍用后处理软件来进行可视化。

3.1.1　visIt

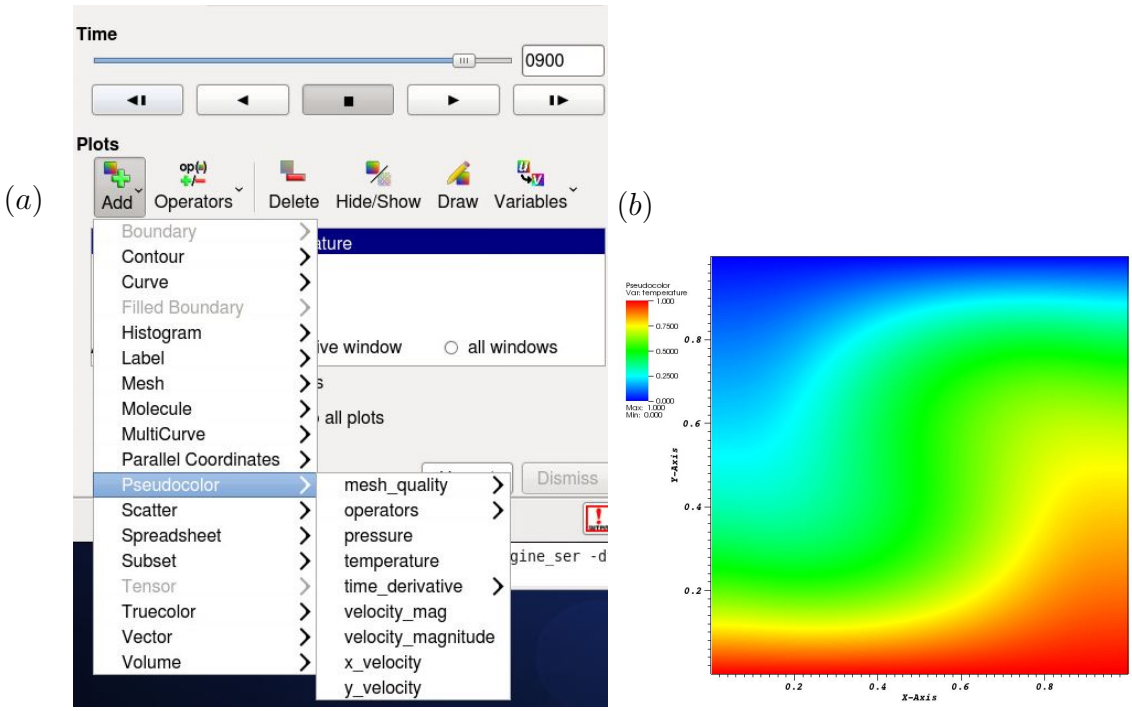
visIt 是美国 Lawrence Livermore 国家实验室（LLNL）开发的开源并行可视化流场分析软件，基于 python 语言编程。这是目前唯一能直接读入 Nek5000 解文件格式的后处理软件，在 Windows 和 Linux 系统下均可安装运行。为了处理方便，下边以 Linux 系统（centOS）中的 visIt 为例，版本为 2.12.2。这部分内容同时参考了 visIt 用户手册 version 1.5[11]，可对照该手册来进行阅读。

先将 visIt 解压安装在 home 目录下，新目录名为 visIt2.12.2。进入该目录中的 src/bin，启动 visIt 的可执行文件就存放在该目录下，输入：

./visit -gui

即可打开 visIt 图形界面，该命令短横线后边是打开软件的各种选项。这里同样需要图像输出终端。

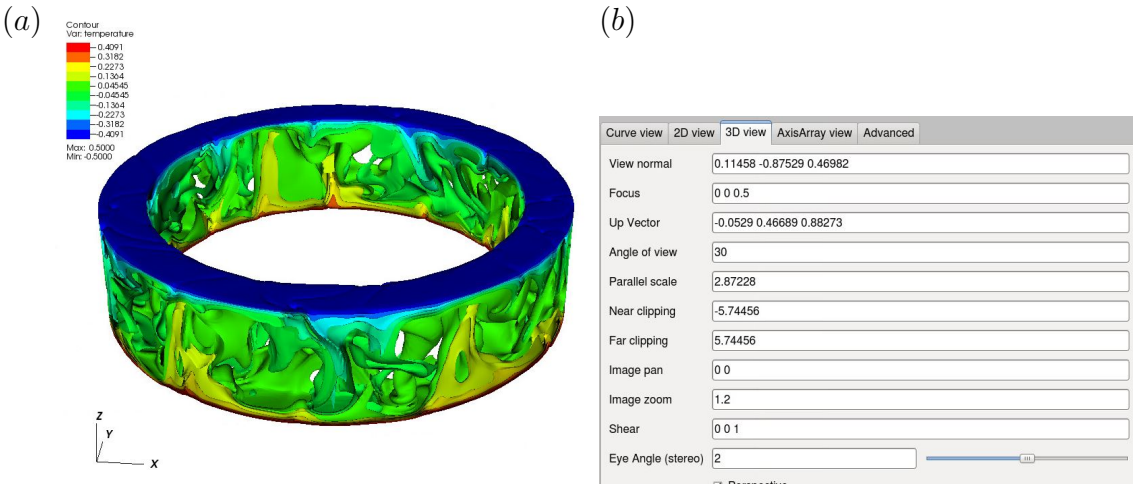
图 3.1: visIt 部分功能及生成的示例流场.



以第二章讲过的 2D 方腔 RB 对流问题为例，点击 gui 上的 Open 图标来打开该问题的流场文件（这里文件名以 box 开头），双击选中 box.nek5000 即将所有的 box.fxxxx 文件按时间顺序打开了。下面可以选中下方 Add 图标来添加流场变量，如 3.1(a) 所示，下拉菜单中有许多的选项，其中最常用的功能有 Contour——画等值面（线），Pseudocolor——生成全流场云图，Vector——生成速度的矢量场示意图，Volume——对流场进行体渲染。我们选择 Pseudocolor，下拉菜单中有 x_velocity，y_velocity，pressure，temperature 这几个选项，对应流场中的速度，压力和温度，若为 3D 问题还有 z_velocity 选项。本章开头介绍过的 outpost 输出的间接变量的流场文件，读入 visIt 后其变量也按在 outpost 输出中的排列顺序用这几个选项来进行显示。我们选择 temperature，同时将图3.1(a) 上部所示的时间进度条拖至任意位置，再点击 Draw 图标，在窗口中即产生该时刻的温度场云图，如3.1(b) 所示。其它 2D 绘图功能可类似 Pseudocolor 来进行操作，均比较简单，希望读者结合手册 [11] 自行探索，这里不再介绍。

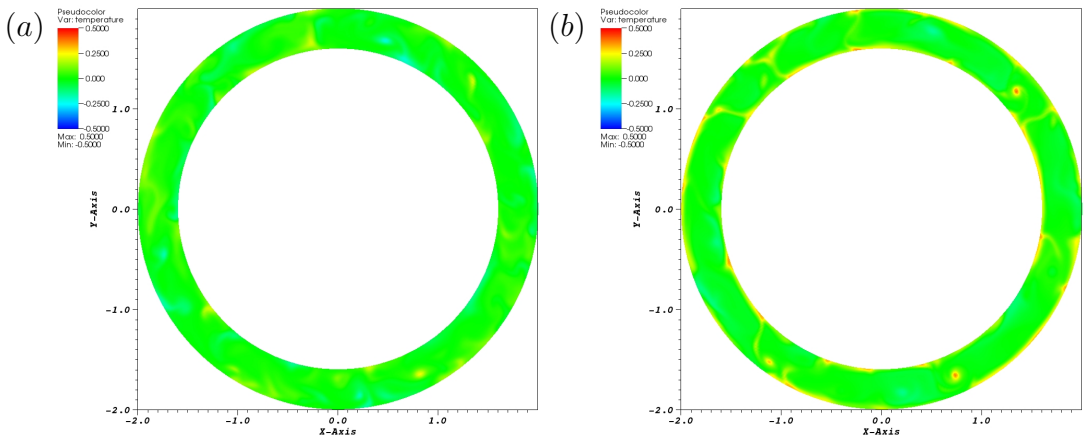
visIt 在流动显示上的优势在于 3D 流场的可视化，而这主要又得归功于其强大的体积渲染（volume rendering）算法。现在我们以 3D 圆环中旋转 RB 对流问题为例来进行介绍。

图 3.2: 温度场及 3D 显示参数设置示例.



打开 cylann.nek5000，这里的流场文件储存了柱坐标下的三个方向速度和温度，为间接变量，我们还以温度场为例。先选择 Contour 功能，生成温度场等值面如图3.2(a) 所示。为了显示效果更加好看，可以在 gui 菜单栏 Controls 目录下的 Annotation 和 View 选项中对坐标轴，标度，显示角度，字体，图例等做修改，这部分同样可参照 [11]，图3.2(b) 列出的是 3D 显示效果较好的一个角度（见图 3.2(a)）的设置参数，可供参考。

图 3.3: 不同 z 平面上的温度场.

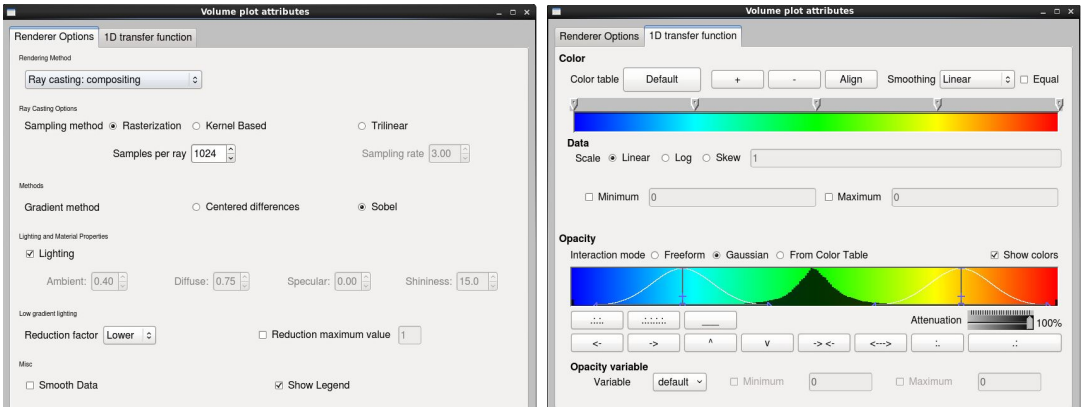


3D 计算域由于流场充满了整个空间，直接使用 Pseudocolor 功能只能直接看到边界处的流动情况，因此我们一般使用切片功能查看计算域内部某个平面的云图。选择 gui 菜单栏 OpAtts 目录下的 Slicing 选项，再选择 Slice。跳出窗口可以选择切片的方向以及位置，图3.3(a) 和 (b) 分别为 $z = 0.5H$ 和 $z = 0.1H$ 处平面的温度场。

切面云图无法呈现完整的流场，Contour 也只能显示流场值大致的分层情况，同样无法清晰地展现流动结构，若要更进一步，我们就需要用到体积渲染功能了。简单地说，该功能

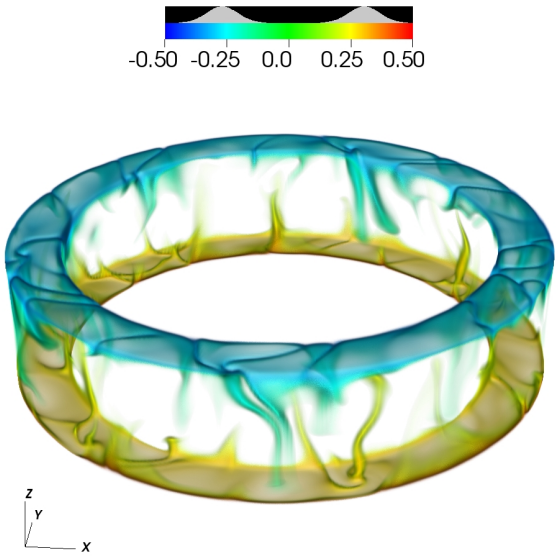
图 3.4: 体积渲染参数设置.

(a)　　(b)



通过用户自定义滤波函数，调整不同流场等值面的透明度，从而能较清晰的地显示流场内部的信息。选择 Volume 中的温度选项，在下方 temperature 的下拉栏中双击 volume，即打开了体积渲染的参数控制窗口。该窗口中有两个选项卡。第一个如图3.4(a) 所示，设置渲染方式及基本参数。将第一栏渲染方式选择第三个光线渲染，效果较好。同时将第二栏 samples per ray 的值调高一些，该值越高渲染效果越精细，但处理时间也相应增加，经验表明 1024 是较合适的一个值。没有提到的值保持默认即可。第二个选项卡如图3.4(b) 所示，调整渲染中的色卡及透明度。上方 Color 中的 Data 一栏，可以手动设置 min 和 max 值来控制显示流场值的范围。主要关注下方 Opacity 一栏。滤波函数有三种选法，通常可选 Gaussian 函数，

图 3.5: 旋转圆环 RB 对流温度场体积渲染图.

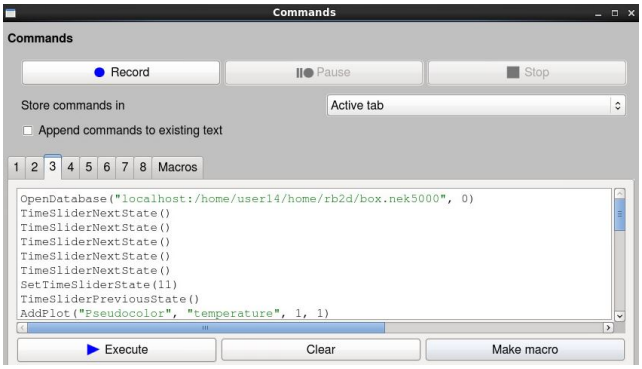


光滑度较好，不同等值面间的区分也较明显。下方色卡可用鼠标左键点击，选择需要进行渲染的等值面，这有时需要对流场的值的分布预先有一定了解。如本例，我们提前知道温度场

梯度较大的位置在极值位置附近（因为存在温度边界层），因此将渲染位置选在色卡两端容易显示出更清楚的结构。选好位置后需要调整渲染等值面的取值范围，换言之就是滤波函数的宽度，在该范围之外的等值面都会变透明。色卡下方有两排按钮，用来手动控制取值范围的调整。点击第二行右起第三个按钮，增加取值范围到合适值，这里需要自己多尝试，并没有一个通用的宽度。调整完毕后，我们再点击 gui 上的 draw 按钮，温度场的体积渲染图就生成了，如图 3.5所示。通过体积渲染，RB 流场中的流动结构——羽流清晰地呈现了出来，对比图3.2(a)，我们不难看出这个功能的优点。同时我们也按前文所述调整了图例等的格式，使图片清晰易读，这也是画图时需要注意的。在运行该功能过程中，同时也易注意到耗时较长。事实上体积渲染需要处理相当多的图层，对内存消耗极大，在一般的 PC 机上很难使用（如 8G 内存的电脑），这也是为什么本章以 Linux 系统为例来介绍 visIt 的原因——建议在并行机或单独的后处理节点上运行此功能，以提高工作效率。

通常在计算中都要输出一组流场文件，并且需要对他们进行批处理，形成的图片可能还需连接形成动画。要实现这些功能就需要用到 visIt 脚本文件了。visIt 脚本文件使用 python 语言编写，这部分同时可参考 visIt 的 python 使用手册 [12]，该手册有 400 多页，讲得比较详细，但事实上我们并不需要太多的 python 知识，因为 visIt 有强大的宏录制功能。宏录制，简单来说，就是将鼠标在 gui 界面操作的过程翻译成 python 的相应语句，并记录下来。这样大大简化了脚本文件的编写。

图 3.6: 宏录制窗口.



我们先回到 2D 方腔 RB 对流的例子，现在来说明如何录制能画出图3.1(b) 的宏，以及如何运行基于该宏的 python 脚本让程序重新画出该图。选择 gui 菜单栏 Controls 目录下的 Commands 功能，弹出的窗口即为宏录制控制窗口。点击录制键 Record，录制开始。接下来我们要做的事就是将前边画图3.1(b) 的操作重复一遍，完毕后按下 Stop 按钮，这时可以看到窗口下方出现了数行 python 语句，如图3.6所示，这些语句对应了刚才录制期间的每一个操作。接下来可以选择 Make macro 按钮生成宏文件保存方便以后使用，但更方便的是将所有语句复制到一个新的 python 脚本（.py 格式）中，这样易于自行添加别的控制语句。 我们在算例目录下新建 post.py 文件，并将上述语句复制进去。现在来运行这个文件。还是在 visit2.12.2/src/bin 目录下，输入命令：

图 3.7: 批处理 python 脚本文件部分内容.

(a)

```
for state in range(TimeSliderGetNStates()):
  SetTimeSliderState(state)
  i=i+1
  ii=i-255
   # antialialiasing
  r = GetRenderingAttributes()
  r.antialiasing = 1
  SetRenderingAttributes(r)

  # Annotation
  a = AnnotationAttributes()
  a.axes3D.visible = 0
  a.userInfoFlag = 0
  a.timeInfoFlag = 0
  a.databaseInfoFlag = 0
  a.axes3D.triadFlag = 1
  a.legendInfoFlag = 0
  a.axes3D.bboxFlag = 0
  a.gradientBackgroundStyle = a.TopToBottom
LeftToRight, RightToLeft, Radial
  a.gradientColor1 = (0, 0, 0, 255)
  a.gradientColor2 = (128, 128, 128, 255)
  a.backgroundMode = a.Gradient
  SetAnnotationAttributes(a)

  AddPlot("Volume", "temperature")
  DrawPlots()
```

(b)

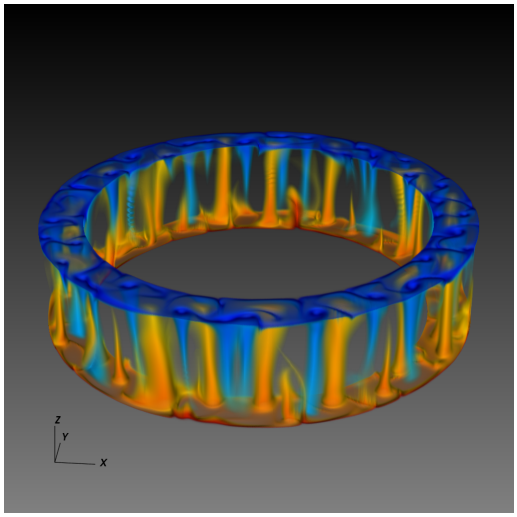
```
  p.colorControlPoints.ClearControlPoints()
  c = ColorControlPoint()
  c.colors = (0, 0, 127, 255)
  c.position = 0
  p.colorControlPoints.AddControlPoints(c)
  c.colors = (0, 0, 255, 255)
  c.position = 0.125
  p.colorControlPoints.AddControlPoints(c)
  c.colors = (0, 255, 255, 255)
  c.position = 0.375
  p.colorControlPoints.AddControlPoints(c)
  c.colors = (255, 255, 0, 255)
  c.position = 0.625
  p.colorControlPoints.AddControlPoints(c)
  c.colors = (255, 0, 0, 255)
  c.position = 0.875
  p.colorControlPoints.AddControlPoints(c)
  c.colors = (127, 0, 0, 255)
  c.position = 1
  p.colorControlPoints.AddControlPoints(c)
  p.freeformOpacity = (0, 0, 0, 0, 0, 4, 5, 6, 8, 9, 11, 13, 15, 17, 20, 23,
27, 30, 35, 39, 44, 50, 56, 62, 69, 76, 84, 92, 101, 109, 118, 128, 137, 147,
157, 167, 176, 186, 195, 204, 212, 220, 227, 233, 239, 244, 248, 251, 253,
254, 254, 253, 252, 249, 245, 240, 235, 228, 221, 213, 205, 196, 187, 178,
168, 159, 149, 139, 130, 120, 111, 102, 94, 85, 78, 70, 63, 57, 51, 45, 40,
35, 31, 27, 24, 21, 18, 15, 13, 11, 9, 8, 7, 6, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 6, 7,
8, 9, 11, 13, 15, 18, 21, 24, 27, 31, 35, 40, 45, 51, 57, 63, 70, 77, 85, 93,
101, 110, 119, 129, 138, 148, 157, 167, 177, 186, 195, 204, 212, 220, 227,
233, 239, 244, 248, 251, 253, 254, 254, 254, 252, 249, 245, 240, 234, 228,
221, 213, 205, 196, 187, 177, 168, 158, 148, 139, 129, 119, 110, 101, 93, 85,
77, 69, 63, 56, 50, 45, 39, 35, 31, 27, 23, 20, 17, 15, 13, 11, 9, 8, 6, 5, 4,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
  SetPlotOptions(p)
```

图 3.8: 批处理 python 脚本文件控制输出部分内容及生成流场图.

(a)

```
 if save == 1:
   s = SaveWindowAttributes()
   s.family = 0
   s.format = s.PNG
   s.width = 1024
   s.height = 1024
   s.quality = 100
   s.resConstraint = s.NoConstraint
   #s.resConstraint = s.ScreenProportions
   s.advancedMultiWindowSave = 0
   s.forceMerge = 0
   fid="%05d" % ii
   #framefilename = "/home/user14/home/rotati
anime/T"+fid
   framefilename = "T"+fid
   print framefilename
   s.fileName = framefilename
   SetSaveWindowAttributes(s)
   n = SaveWindow()
```

(b)



./visit -cli -s /home/user14/home/rb2d/post.py
图3.1(b) 的图形文件就直接生成在了算例目录下。至此就结合宏录制功能用 python 脚本成功处理了一个流场，其他功能也可如法炮制。

下面介绍如何进行批处理。这只需加上循环语句即可实现。我们结合 3D 旋转圆环 RB 对流算例的温度场体渲染动画生成的脚本来说明。首先将之前介绍过的生成输出单个体渲染图的过程录制成宏，并将相应 python 语句复制到 volume.py 文件中。如图 3.7(a) 头几行所示，在这使用“state”来控制循环，类“TimeSliderGetNStates”则储存了流场文件的数量，本例使用了 200 个流场，因此该脚本将会循环运行 200 次。图3.7(b) 记录的是控制体渲染的色卡以及透明度的值的语句，虽然参数较多，但都是事先在宏录制时手动调整合适后直接复制过来的，本身并不复杂。

图3.8(a) 记录的是控制循环输出的语句，生成的文件名为“T”加上从 0 开始的数字序号。编写完毕后运行该脚本，最终会生成 T00000.png-T00199.png 共 200 副图片，该循环过程在网格量较大时可能相当耗时，本例大概需要十几小时才能完成。图3.8(b) 是生成的 200 个图片文件中的一个，我们在脚本文件中对其显示控制参数做了进一步的修改以改善显示效果。该文件的完整内容列在了附录A中供参考。生成了一组图片文件后，推荐使用 GifAnimator 软件将其拼接成.gif 格式动图，或使用 Windows movie maker 将其制作成.mov 视频文件，这些软件的使用方法这里不再赘述。

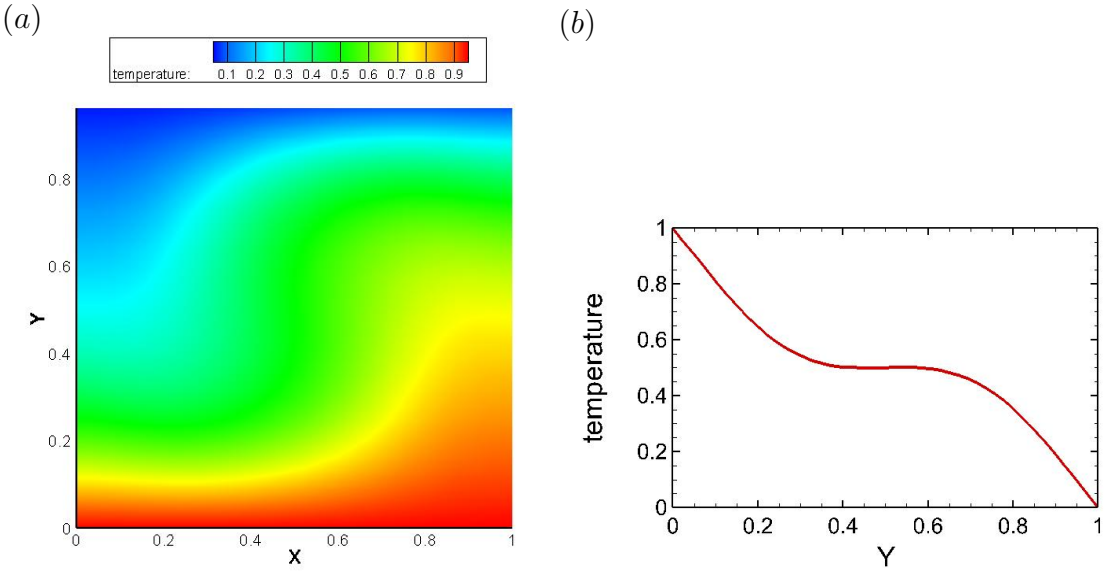
以上介绍的是 visIt 的优势功能，事实上它也有一些弱项。比如常用的曲线绘制功能，visIt 的用户自由度就较低，绘制效果较差，这时就需要用到别的后处理软件来弥补这些不足了，下面要介绍的 tecplot 正是可以较好地满足这些需求。

3.1.2　tecplot

tecplot 是相对常用的流场后处理软件，其使用教程即便用百度搜索也能搜到海量结果。但它并不能直接读入 Nek5000 的流场文件，为此还需先借助 visIt。选择 gui 菜单栏 File 目录下的 Export database，在弹窗的 Export to 下拉菜单中选择 Tecplot，同时在 Add Variable 中添加需要输出的变量，如速度和温度等。第一行 Export all time states 勾选上即输出所有时刻的流场，生成按序号排列的.tec 文件，这些文件就能被 tecplot 读入了。上述过程同样可以录制成宏以提高读写效率。这里使用的 tecplot 版本是 2017 R1。

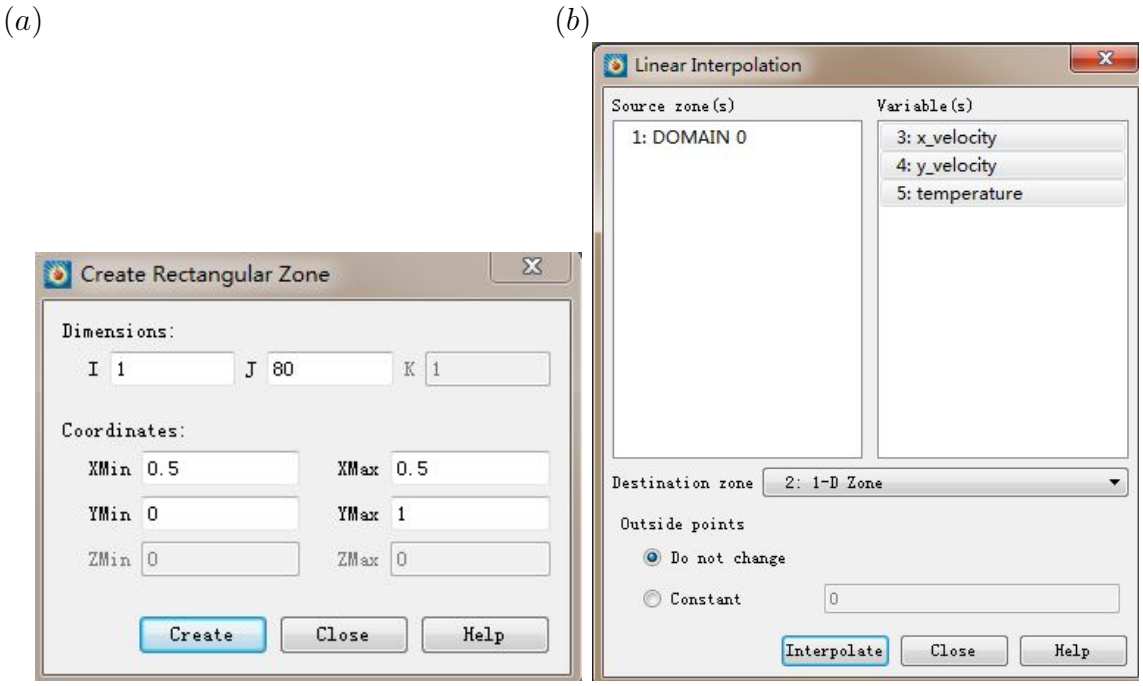
还以上一节中 2D 方腔 RB 对流为例，图3.9(a) 所示是在 tecplot 绘制的温度场。通常我们还关心某一直线上的温度分布情况，以 $x = 0.5$ 处为例。选择 gui 菜单栏 Data 目录下的 Create Zone 来生成一个新的区域，再选择 Rectangular 生成矩形区域，弹出窗口如图3.10(a) 所示，I、J 表示 x、y 方向的网格数，这里是直线，因此 x 方向只需一个点，y 方向根据之前谱元法离散的网格数量及插值多项式阶数来确定其取值。本例中 y 方向谱元数为 11，精度为 7 阶，因此设置 J 值大约等于 $11 \times 7 = 77$，取为 80。下方 coordinate 栏选择区域的笛卡尔坐标范围，这里 x 坐标就取为 0.5，$0 < y < 1$。生成该区域后，需要将流场插值到该直

图 3.9: tecplot 中绘制的温度场及 $x = 0.5$ 处温度剖面.



线上，即能提取出这条线上的流场数据。选择 Data 目录下的 Interpolate，再选择 Linear 线性插值，弹窗如图3.10(b) 所示，左侧栏选中需要插值的变量，默认是全选，下方目标区域选取刚才生成好的 1-D zone，最后点击 Interpolate 按钮即可。得到的温度型如图3.9(b) 所示，这里已经通过调整坐标轴等选项使得显示效果较好，这样的效果是 visIt 难以实现的。

图 3.10: tecplot 中创建新区域及插值窗口.



另一个常用到的功能是流场的空间平均。在流场后处理时，经常要进行物理量的体平均，比如求 Re 时要求均方根速度；也可能要进行某个平面的平均，比如 RB 对流求冷/热壁 Nu 数，需求壁面平均温度梯度。诚然在大部分情况下可以在计算程序中编写后处理子程序去处

理他们，但若对于某一个瞬时的流场文件做相应处理，则在后处理软件中去进行也是较为方便的，还能跟计算程序中自定义的后处理模块进行结果比对，检验自己编程的正确性。仍然以 2D 方腔 RB 对流为例，这里先设定无量纲参数的值方便对比：$Pr = 1$，$Ra = 10^4$。我们知道瞬时 Nu 有多种定义方式，但结果完全等价（可参考文章 [5]）。通过体积（2D 为面）平均的定义：

$$Nu_v = \langle v_y T \rangle_V + 1 \tag{3.1}$$

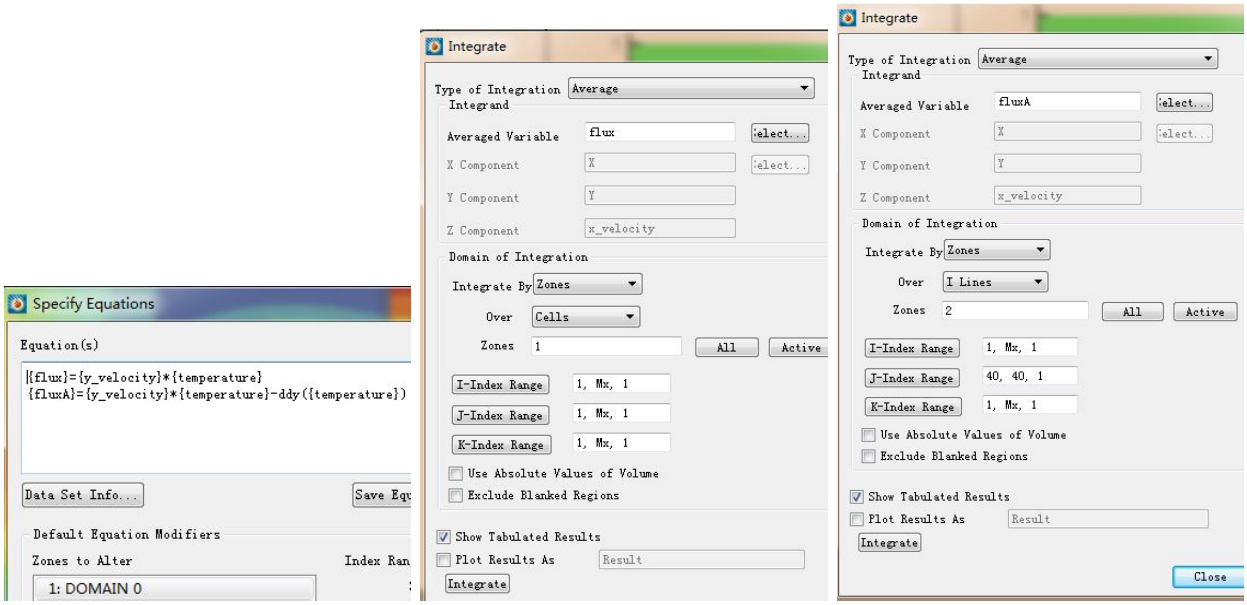
或通过面（2D 为线）平均的定义：

$$Nu_A = \langle v_y T - \partial_y T \rangle_A \tag{3.2}$$

其中尖括号代表空间平均。现在先用 tecplot 来实现这些功能。首先我们对储存了直接流场变量（速度和温度）的流场文件来进行处理。先定义新的流场变量以方便做积分，选择 Data 目录下 Alter 中的 Specify Equations，在弹窗的输入区域定义新的变量，如图3.11(a) 所示，新定义的 flux 和 fluxA 分别是式3.1和3.2中尖括号内的项。之后选择 Analyse 目录下的 Perform

图 3.11: tecplot 中定义新变量及求平均窗口.

(a)　(b)　(c)



Integration，来对这两个新变量做空间平均。先处理 flux，如图3.11(b) 所示，第一栏计算方式选择 Average 求平均，第二栏变量名选择 flux，下方积分域选择栏，Over 一栏选择 Cells 求体平均，Zones 选择默认的 Domain 0，其他值保持默认，点击 Integrat 按钮即可计算出体平均值。再来处理 fluxA，我们需要的是它在任一 y 平面的平均值。由于数据是按谱元来进行存储，tecplot 无法直接计算某个截面的平均，因此需要先将数据插值到新的差分网格上，插值方式前文中已经介绍了，区域就选则与原计算域相同大小的矩形区域，并注意网格的数量。然后回到求平均的窗口，如图3.11(c) 所示，在积分域选择栏，Over 一栏选 I Lines，表示求平行 x 轴方向的线平均，Zones 选择刚才插值的新计算域 Rectangular zone。新区域 y

表 3.1: tecplot 软件、自定义的后处理程序及文献中计算的 Nu_v 及 Nu_A，后两列为与文献 [13] 中的相对误差.

计算方式	Nu_v	Nu_A	Err_v	Err_A
tecplot，直接变量	2.1578	2.1559	0.014%	0.10%
tecplot，间接变量	2.1578	2.1557	0.014%	0.11%
后处理程序	2.1581	2.1580	0	0.0046%
文献 [13]	N/A	2.1581	N/A	N/A

方向有 80 个网格，而 Nu_A 的值在不同 y 平面都应在误差范围内相等，不妨就求 $y = 0.5$ 附近平面（直线）的平均值，因此 J-Index Range 选择第 40 个网格点，其它值保持默认即可。这两种方法计算的结果列在了表3.1第一行中。

图 3.12: tecplot 中制作动画宏文件的部分循环控制语句.

```
#!MC 1410
$!VarSet |MFBD| = 'D:\Program Files\Tecplot\Tecplot 360 EX 2017 R1'
$!varset |loopnum| = 135
$!loop |loopnum|

$!if |loop| <= 135

$!if |loop| <= 99

$!if |loop| <= 9

$!READDATASET  '"G:\data\studs\Ra=1E10\s8\anime\visit_ex_db000|loop|.tec" '

$!else

$!READDATASET  '"G:\data\studs\Ra=1E10\s8\anime\visit_ex_db00|loop|.tec" '

$!endif

$!endif

$!if |loop| > 99

$!READDATASET  '"G:\data\studs\Ra=1E10\s8\anime\visit_ex_db0|loop|.tec" '

$!endif
```

上文用 tecplot 计算了新变量再进行平均，我们也可以在后处理程序中事先求好这些间接变量，输出到流场文件中，再通过上述平均方法来求得 Nu_v 和 Nu_A。使用这种方法的结果列在表3.1第二行。我们也通过 Nek5000 中自定义后处理程序计算了相应结果，并将上述所有结果跟文献 [13] 中的 benchmark case 进行了对比，可见在后处理程序中的结果最为精确，但 tecplot 计算结果误差也在千分之一以内。因此想通过后处理程序求解某些区域的积分时，若在编程上遇到困难，可以在精度要求不是太高的情况下退而求其次使用 tecplot 来处理。

在这节的最后简单介绍一下 tecplot 中录制宏来制作动画的方法。录制宏的方法跟上一节 visIt 中非常相似，选择 Scripting 目录下的 Record Macro，选择开始录制即可。生成的宏加上循环控制语句，即可进行批处理制作一组图片，如图 3.12所示，这里使用了 135 个图片文件来制作动画。完整的宏文件示例附在附录B中以供参考。

3.2 自定义后处理程序

这一节介绍各种后处理程序的写法示例。

3.2.1 By Fortran

这小节介绍 Fortran 编写的后处理程序的示例，他们既可以在.usr 的 userchk 子程序内调用，也可以分开编译，与计算程序一通运行。若无特别说明，下面我们只关心并行计算的情况。首先要了解程序的数据存储结构，这部分内容可同时参考 [10] 第 22 页开始的介绍。Nek5000 的数据结构有几点特性需要记住：

1. 单个的谱元是数据存储的最小单元，每个并行块可分配不同数量的谱元，但同一个谱元不会被分配到不同的块上进行存储和计算。

2. 以谱元为最小单位，以每个并行块为一个集合，通常的流场变量 a（如速度）矩阵（数组）可以有两种空间分配形式。一种为 a(nx1,ny1,nz1,nelv)，其中 nx1，ny1，nz1 代表每个谱元内实际的插值点数目，它们比空间离散精度的值大 1，nelv 为每个并行块上实际的谱元数量。另一种为 a(ntot,1,1,1)，其中 ntot = nx1 × ny1 × nz1 × nelv。后一种一维数组的形式在存储上效率更高。

3. 从全局角度，数据是非结构存储的。而从单个的并行块的角度，数据实际可看做按结构存储：每个谱元，以及谱元内部的每个节点都按顺序排列。相邻的谱元之间通过分享同一个内边界（“E”型），即通过储存相同数据来反映谱元连接的逻辑关系。

接下来我们再简要介绍 Nek5000 已有的常用后处理子程序/函数，调用这些子程序/函数能极大地简化编程。这些子程序/函数大部分在 core 目录下的 math.f，navier1.f，navier5.f 中可以找到。

subroutine outpost(a1,a2,a3,pr,t1, ' ')：之前介绍过，用来输出间接变量流场文件。

subroutine copy(a,b,ntot)/opcopy(a1,a2,a3,b1,b2,b3)：将 b 或 b1 数组复制到 a 或 a1 数组，分别对 1 个和多个数组进行操作。

subroutine add2(a,b,ntot)/opadd2(a1,a2,a3,b1,b2,b3)，add4(a,b,c,d,ntot): 数组加法运算，a 或 a1(i) = a 或 a1(i) + b 或 b1(i)，a(i) = b(i) + c(i) + d(i)，分别对 1 个和多个数组进行操作。

subroutine sub2(a,b,ntot)/opsub2(a1,a2,a3,b1,b2,b3)/sub3(a,b,c,ntot)/ opsub3(a1,a2,a3,b1,b2,b3,c1,c2,c3): 数组减法运算，a 或 a1(i) = a 或 a1(i) − b 或 b1(i)，a 或 a1(i) = b 或 b1(i) − c 或 c1(i) 分别对 1 个和多个数组进行操作。

subroutine col2(a,b,ntot)/opcol2(a1,a2,a3,b1,b2,b3)，col3(a,b,c,ntot)，col2c(a,b,c,ntot)，cmult(a,c,ntot)：数组乘法运算，a 或 a1(i) = a 或 a1(i) × b 或 b1(i)，a(i) = b(i) × c(i)，a(i) = a(i) × b(i) × c，a(i) = a(i) × c。

subroutine addcol3(a,b,c,ntot)/add2s1(a,b,c,ntot)/add3s2(a,b,c,c1,c2,ntot)：几个常用的数组乘法加法复合运算，a(i) = a(i) + b(i) × c(i)，a(i) = a(i) × c + b(i)，a(i) = c1 × b(i) + c2 × c(i)

function glsc2(a,b,ntot)/glsc3(a,b,c,ntot)：内积运算，返回实型值，在每个并行块分别进行累加，最后在头结点求总和返回。glsc2 = glsc2 + a(i) × b(i)，glsc3 = glsc3 + a(i) × b(i) × c(i)。这些函数可用来求流场变量的体积分，比如求式3.1，可调用该函数：
Nu_v = glsc2(vy(i,1,1,1)∗t(i,1,1,1,1),bm1(i,1,1,1),ntot)/volvm1 + 1，各变量含义可参考表1.3，表1.4。

subroutine gradm1(ax,ay,az,a)：求流场变量 a 的梯度。被广泛用于求各种变量的一阶空间导数，计算效率较高（相较于如 dudxyz 这个功能相似的子程序），如温度梯度 ∇T = (tx,ty,tz) 可由语句 call gradm1(tx,ty,tz,t) 得到。值得注意的是，谱元法自身的离散特性只能保证流场直接变量（速度、温度等）至多两阶空间导数的全场光滑，更高阶的导数可能遇到谱元边界处的间断，因此在调用该子程序时注意不要求高于 2 阶的流场直接变量的空间导数。

subroutine glsum(a,ntot)：全流场求和。通常在 userchk 子程序中调用，在每个并行块上求出了数个 a 值后，通过该子程序将全场的 a 值求出。

subroutine glmax/glmin(a,ntot)：求某流场变量 a 在全场中的最大/最小值。

subroutine avg1(avg,f,alpha,beta,ntot,name,ifverbose)：求某流场变量 f 的加权平均值。avg(i) = alpha × avg(i) + beta × f(i)。ifverbose 逻辑开关用来控制是否在屏幕上输出全场平均值的最大和最小值。该子程序可用来求 f 的时间平均值。可直接在 userchk 中调用，也可写成 timeaver 子程序方便使用，如下所示（以求 x 方向时均速度为例）：

```
dtime = time − time1
atime = atime + dtime
if (atime.ne.0. .and. dtime.ne.0.) then
beta = dtime/atime
alpha = 1. − beta
ifverbose = .false.
call avg1(uavg,vx,alpha,beta,ntot,'uavg',ifverbose)
endif
time1 = time
```

其中 time1 存放上一时间步的时间，time 为当前步的时间，atime 为总计算时间。注意在 userchk 开头用 save 语句保存 uavg 的值。

subroutine comp_vort3(vort,u,v,w)：计算涡量，vort 数组结构为 vort(ntot,3)。该子程序的内容在原代码基础上做了修改使其稍微简练了，读者可参考对比原代码的内容。

图 3.13: hpts.in 文件.

```
4
1.95 0 0.5
1.95 0 0.01
1.995 0 0.5
1.95 0 0.025
```

subroutine hpts(): 将流场任一点的直接变量的值插值后输出。首先我们需要算例目录下的输入文件 hpts.in，如图3.13所示，第一行为输出的点的数量（4 个监测点），接下来 4 行为这些点的空间坐标，数据将输出到 hpts.out 中，每一行按速度、压力、温度的顺序排列。该子程序的内容在原代码基础上做了修改以方便输出多个 hpts.out 文件，能监测更多变量。同时注意到其中调用的插值函数只直接作用于速度温度等变量，因此想监测其他变量可在 userchk 中先将那些变量 copy 到速度温度的数组中，调用完 hpts 后再恢复回来。还需注意的是该插值过程计算量不小，因此不宜在 userchk 中每一步都调用，否则会大大拖慢计算效率。

常用自带程序暂时介绍这些，其他子程序留待日后补充。

下面介绍一些自定义后处理程序的例子，其中一些变量的声明、头文件的导入部分及续行符做了省略。

- 求边界平均通量，可用于求 RB 对流冷热壁 Nu 数，以上（冷）壁为例：

```
subroutine(NuC,ntot)
do e = 1,nelv
do f = 1,2 * ndim
if (cbc(f,e,2).eq.'t ') then ! 寻找所有边界为“t”类的谱元，即为等温壁
call facind(i0,i1,j0,j1,k0,k1,nx1,ny1,nz1,3) ! 确定满足条件的数据点位置
l = 0
do k = k0,k1
do j = j0,j1
do i = i0,i1
if (ym1(i,j,k,e).gt.0.5) then !y > 0.5，求上壁面的值
l = l + 1
NuC = NuC + (−uny(l,1,f,e)) * ty(i,j,k,e) * area(l,1,f,e) !ty = ∂_y T，注意面法向量的方向
```

```
sarea = sarea2 + area(l,1,f,e)
endif
enddo
enddo
enddo
endif
enddo
enddo
NuC = glsum(NuC,ntot)
sarea = glsum(sarea,ntot)
NuC = NuC / sarea
return
end
```

- 求流场任一矢量 (ax,ay,az) 的梯度张量，其中调用了弱形式的梯度函数及光滑函数以增加空间导数在谱元边界的光滑度：

```
subroutine vec_grad(ax,ay,az,axx,axy,axz,ayx,ayy,ayz,azx,azy,azz)
call wgradm1(axx,axy,axz,ax,nelt) ! 弱形式的梯度计算
call vec_dssum(axx,axy,axz,nx1,ny1,nz1) ! 边界处光滑处理
call opcol2(axx,axy,axz,binvm1,binvm1,binvm1)! 将弱形式的解乘上权矩阵的逆矩阵得到最终的梯度形式

call wgradm1(ayx,ayy,ayz,ay,nelt)
call vec_dssum(ayx,ayy,ayz,nx1,ny1,nz1)
call opcol2(ayx,ayy,ayz,binvm1,binvm1,binvm1)

call wgradm1(azx,azy,azz,az,nelt)
call vec_dssum(azx,azy,azz,nx1,ny1,nz1)
call opcol2(azx,azy,azz,binvm1,binvm1,binvm1)

call dsavg(axx)! 进一步光滑
call dsavg(axy)
call dsavg(axz)
call dsavg(ayx)
call dsavg(ayy)
call dsavg(ayz)
```

```
call dsavg(azx)
call dsavg(azy)
call dsavg(azz)
end
```

之后很多涉及到梯度张量的计算都将调用该子程序。

- 求应变率张量 $S_{ij} = \frac{1}{2}(\partial_i u_j + \partial_j u_i)$：

```
subroutine strain_tensor(u,v,w,stx,sty,stz)

call vec_grad(u,v,w,uux,uuy,uuz,vvx,vvy,vvz,wwx,wwy,wwz)

do i=1,ntot

stx(i,1) = uux(i) ! 利用张量对称性还能减少 3 个变量的定义语句
stx(i,2) = 0.5 * (uuy(i) + vvx(i))
stx(i,3) = 0.5 * (uuz(i) + wwx(i))

sty(i,1) = 0.5 * (uuy(i) + vvx(i))
sty(i,2) = vvy(i)
sty(i,3) = 0.5 * (vvz(i) + wwy(i))

stz(i,1) = 0.5 * (uuz(i) + wwx(i))
stz(i,2) = 0.5 * (vvz(i) + wwy(i))
stz(i,3) = wwz(i)

enddo
return
end
```

类似还可以求旋转率张量 $\Omega_{ij} = \frac{1}{2}(\partial_i u_j - \partial_j u_i)$。

- 求二阶张量的二阶范数，即 $\|S_{ij}\| = \sqrt{S_{ij}S_{ij}}$：

```
subroutine tensor_norm(ax,ay,az,norm)
call rzero(trace,n)! 需累加得矩阵的迹，故预先将其置零

do i=1,ntot
```

```
do m=1,3
trace(i) = trace(i) + ax(i,m) ** 2 + ay(i,m) ** 2 + az(i,m) ** 2
enddo
norm(i) = sqrt(trace(i))
enddo

return
end
```

- 通过“Q 准则”定义 3D 涡结构，具体可参考文献 [7]：

```
subroutine q_cri_3d(q3d,u,v,w)
call strain_tensor(u,v,w,stx,sty,stz)
call rotation_tensor(u,v,w,rtx,rty,rtz)
call tensor_norm(stx,sty,stz,norm_s)
call tensor_norm(rtx,rty,rtz,norm_r)! 先求出应变率、旋转率张量，再求其二阶范数
do i=1,n
q3d(i) = 0.5 * (norm_r(i) ** 2 - norm_s(i) ** 2 ) ! 文献 [7]
enddo
return
end
```

Nek5000 中还有内置子程序 lambda2 求解用“λ_2”准则定义的 3D 涡结构（参考文献 [8]），读者可参看原代码，对比这两种方法。

- 计算湍流统计中常用的任一流场变量 a 的方均根脉动量，$a_{rms} = \langle (a - \langle a \rangle_t)^2 \rangle_t^{\frac{1}{2}}$：

```
subroutine fluc_rms(a,a_avT,a_fluct,a_flucAV,time0)
call sub3(a_fluc,a,a_avT,n)
call col2(a_fluc,a_fluc,n)

do i=1,n

call timeaver(a_fluc(i),a_fluct(i),a_flucAV(i),time0)! 时间平均子程序，写法如前文所示
a_flucAV(i) = sqrt(a_flucAV(i))

enddo
return
```

```
end
```

- 将笛卡尔坐标下的矢量分量转化到柱坐标下。由于任意形状的计算区域 Nek5000 都用笛卡尔坐标求解，因此这个功能常常能用到：

```
subroutine Car_to_cyl(u,v,w,uc,vc,wc)
do i=1,n
x = xm1(i,1,1,1)
y = ym1(i,1,1,1)
r = sqrt(x ** 2 + y ** 2)
uc(i) = u(i) * (x/r) + v(i) * (y/r)
vc(i) = u(i) * (− y/r) + v(i) * (x/r)
wc(i) = w(i)
enddo

return
end
```

以上为读者提供了一些简单的范例，多调用 Nek5000 的内置子程序/函数能极大简便我们的编程。

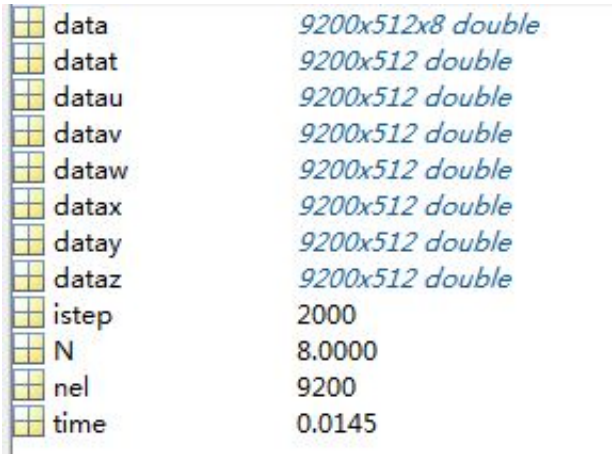
3.2.2 By Matlab

这一小节介绍用 Matlab 编写的后处理程序示例。首先需要将流场文件读入，我们以 KTH 用户编写的网络开源脚本 readnek.m 为蓝本，作简单的修改即可实现，该文件完整代码附在了附录C中。之前介绍过的两种流场文件格式.fld/.fxxxx 的区别在于文件头，该脚本只适用于读入.fxxxx 格式的文件，当然做简单修改就能读.fld 格式，这里不再赘述。还以 3D 旋转圆环 RB 对流的一流场文件为例，该例使用 9200 个谱元，空间精度为 7 阶，即每个谱元内三个方向上都分别有 8 个插值点。圆环内外半径为 1.6 和 2。将某一时刻流场文件 cylann0.f00001 移至 Matlab 工作目录下，将该文件名存入 fname 数组，输入：

```
[time,istep,nel,N,data,datax,datay,dataz,datau,datav,dataw,datat] = readnek(fname);
```

得到的 datax，datay，dataz 储存了网格点上的空间坐标，datau，datav，dataw 储存的三个速度分量，datat 储存温度。图3.14显示的是各变量的值及数组维度。Nek5000 生成的一组流场文件，只会在头几个文件中包含网格信息（空间坐标），以减少输出文件的大小。因此这里我们读入的.f00001 包含了网格信息，data 数组最后一维大小为 8（xyz+uvw+p+T），若读入如.f00011，data 的最后一维大小为 5，这部分在 readnek.m 中已经做了判断。可以看到各数组均按谱元顺序存储，同时每个谱元内部有 $8 \times 8 \times 8 = 512$ 个数据点。但是这样的数

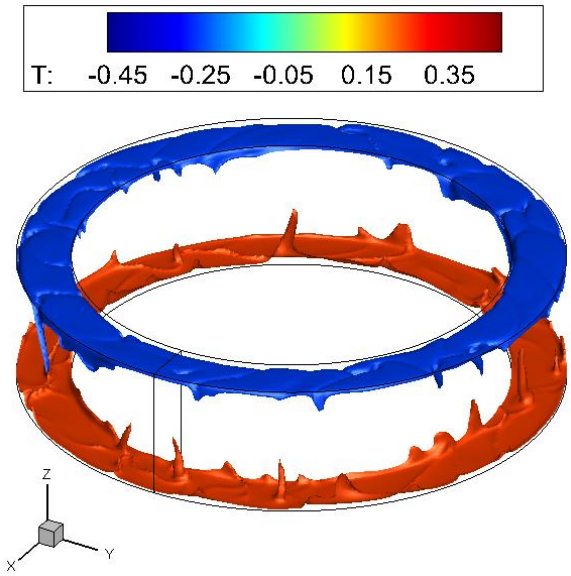
图 3.14: 圆环算例文件读入后的各变量.



据结构还不足以建立谱元间的逻辑关系，无法在 Matlab 中对流场做进一步的计算。如何在 Matlab 中编程进行更高效的后处理是我们一直在努力的。

一个比较容易的办法是将数据插值到差分网格上，这样点与点之间按网格编号顺序排列，易于处理。但是对不同的问题需设置不同的插值区域，如方腔问题需要笛卡尔坐标下的矩形域，圆环问题则在柱坐标下编程才较简单，还有更多复杂计算域的问题要用一般坐标系来描述，这更加困难。尽管限制很多，插值已经为解决很多常见构型的问题提供了一种方案。

图 3.15: Matlab 插值后的温度场等值面.



仍以该圆环问题为例。我们编写了 intpnek.m 脚本将数据插值到环形柱域，完整代码列在了附录C中。输入：

[U,V,W,T,Xm,Ym,Zm] = intpnek(data,r1,r2,meshr,meshphi,meshz,datax,datay,dataz);

其中 r1，r2 代表内外径，meshr，meshphi，meshz 代表 r，φ，z 方向的差分网格数，注意其取

值与谱元法单元数相近以保证插值的精确。该脚本中调用了 Matlab 内置插值函数 griddata，其具体用法可查看 Matlab 帮助。之前提过谱元的相邻关系由他们接触面上的相同数据来确定，因此每两个相邻谱元接触面上的数据都要重复一次，griddata 函数会自动地判别重复数据点，并只保留其中一个。但该插值过程比较耗费内存，耗时也较长，若是用循环语句连续读入和插值一组流场文件，建议在专用的 Linux 工作机上进行。

插值到柱坐标下的数据可以结构化输出，并用 tecplot 来进行流场显示。我们编写了 outputTec.m 脚本来实现该功能，完整代码列在附录C中。输入：

outputTec(meshr,meshphi,meshz,Xm,Ym,Zm,U,V,W,T)

即可将流场输出到.dat 文件中，结果如图3.15所示（温度场等值面）。

这之后可以方便的对柱坐标下的流场来进行各种后处理，如计算涡量，速度梯度张量及其不变量，等等，这里不再赘述了。

第四章　求解其他问题

除了求解一般的不可压 NS 方程，Nek5000 及部分拓展程序包还能求解其他问题，如：变物性参数问题，线性稳定性分析，流动控制与优化，MHD 方程，小马赫数方程，等等。下面对其中一些问题的求法做简单介绍。

4.1　变物性参数问题

许多物理过程必须考虑如密度、粘性系数、热传导系数等物性参数不是常数的情况，这一类问题可以用 Nek5000 中 uservp 子程序来处理。下面用考虑 non-Oberbeck-Boussinesq I（NOB-1）效应的 RB 对流为例进行介绍。该效应描述了以下物理情形：系统温差较大导致流体的物性参数（密度、粘性系数和热传导系数）随温度显著变化，流动本身仍然是不可压的。先要知道流体物性参数随温度的变化关系，这里我们用水和甘油作为两种工质，上述函数关系可参考文献 [1]，[17]，[18]，[6]。下面以甘油为例，物性参数可拟合成温度的多项式函数。图4.1(a) 设置了各常数值（如 [17] 中），以及多项式函数中的系数的全局变量，这些常数

图 4.1: uservp 参数设置（甘油）.

(a)

```
 common /nob_coef/Kn(1:7),Bn(1:3),Cn(1:3),Dn(1:3),
$                    En(1:4),Fn(1:3)
 common /constant/Rhom,Cpm,Tm,Alpham,Kappam,Km,
$                    Viscom,delta_T,rho0,eps
 real*8 Kn,Bn,Cn,Dn,En,Fn,
$       Rhom,Cpm,Tm,Alpham,Kappam,Km,
$         Viscom,delta_T,rho0
 integer i,j,k,e,n

 Rhom = 1.2477*1e3
 Cpm = 2.5108*1e3
 Alpham = 4.7893*1e-4
 Km = 2.9351*1e-1
 Kappam = 0.0937*1e-6
 Viscom = 238.738*1e-6
 Tm = 40

 Kn=[ -771.27*1e-4,2746.4*1e-6,-3257.1*1e-8,1513.2*1e-9,-1135*1e-10
$    ,261.07*1e-11,-18.682*1e-12]        ! kinetic viscosity nu
 Bn=[13.858*1e-4,3.913*1e-6,-0.7577*1e-8]          ! thermal diffusivity kappa
 Cn= [-4.789*1e-4,-0.3795*1e-6,0.]         ! density rho
 Dn= [22.511*1e-4,0.,0.]            ! isobaric specific thermal capacity cp
 En = [20.639*1e-4,4.664*1e-6,1.0757*1e-8,2.54*1e-9]   ! expansion coeffient alpha
 Fn=[3.863*1e-4,0.,0.]           ! thermal conductivity k
```

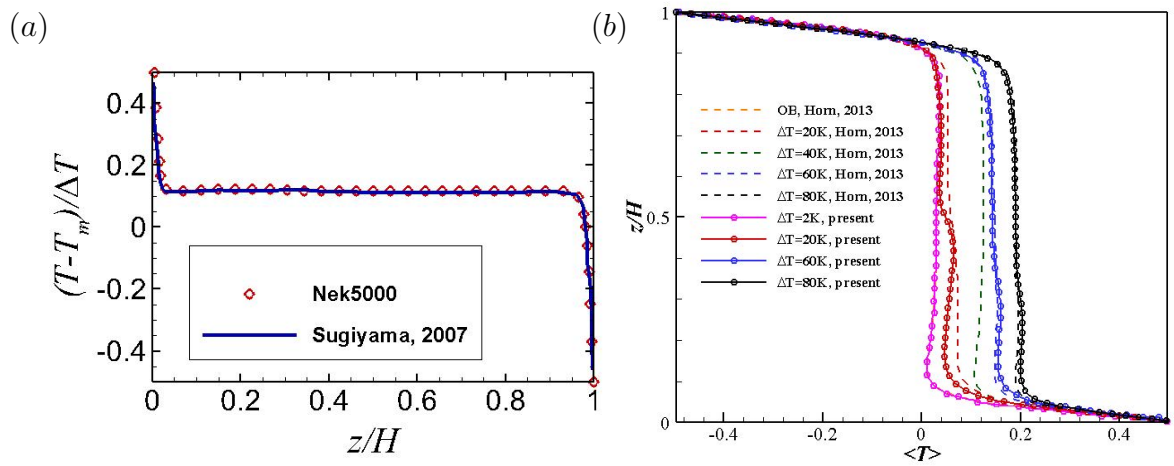
(b)

```
 if(ifield.eq.1)then
 udiff = 0
 do i=1,7
 udiff= udiff +
$                Kn(i)*((temp)*delta_T)**i
 enddo

 udiff = udiff * Viscom + Viscom

 udiff = udiff / Kappam

 utrans = 1.

 elseif (ifield.eq.2)then
 udiff = 0
 do i=1,3
 udiff = udiff +
$                 Fn(i)*((temp)*delta_T)**i
 enddo
 udiff = udiff * Km + Km

 udiff = udiff/(Rhom*Cpm*Kappam)


 utrans = 1.
 endif
```

都是有量纲的。delta_T 是系统温差，作为可变的全局输入参数在别处设置。图4.1(b) 记录了动量方程和能量方程中变物性参数的设置方式。这里要特别注意的是将有量纲的参数值无量纲化的方式，我们最终希望将无量纲方程左端对流项的系数（ρ 和 ρc_p）化成 1，那么扩散

项之前的无量纲参数也要作相应变化。uservp 设置好后，还需将.rea 文件第一部分内容逻辑开关中的 IFSTRS 设成“T”，表示粘性应力项取完整形式 $\nabla\cdot\sigma=\nabla\cdot[\mu(\nabla\mathbf{u}+\nabla\mathbf{u}^T)]$，因为 μ 不再是常数了。另外设置 $Ra=10^8$，$\Delta T=40K$，与文献 [17] 的结果进行对比，图4.2(a) 是对比的 x 方向平均后的温度剖面图，与文献结果吻合得较好。我们在 3D 圆筒中也与博士论文 [6] 中结果做了对比，$Ra=10^6$，结果如图4.2(b) 所示，在大部分温差下结果吻合较好，但 $\Delta T=20K$ 的数据与文献差别较大。总之，变物性参数问题可以按上述思路求解，但仍

图 4.2: NOB-1 算例与文献的结果对比.



需要更多的测试来确保其正确性。

4.2　线性稳定性分析

Nek5000 具备求解线化 NS 方程的求解器，自然想到可以利用它来做线性稳定性分析。而实现这一功能又需要两步：第一步是求得时间无关的基本流（平均流），第二步是通过求解特征值问题来研究线化 NS 方程的稳定性。下面分别从这两步来介绍。

4.2.1　基本流的求法

基本流有多种求法。有的流动随时间收敛至定常，最终的稳定的解直接可以作为基本流。但也有很多问题需要求不稳定的定常解。可以用牛顿迭代法来求这样的基本流，但在 Nek5000 中还有待实现。这里介绍另一种滤波的方法（Selective Frequency Damping）来求基本流，关于该方法的详细内容可参考文献 [2]，[9]。该方法的优势是在现有代码上编程实现简单——在动量/能量方程右端项添加滤波函数项作为新的体积力项，再时间推进求解目标平均值函数的一阶常微分方程，最终使流场解收敛到该目标平均值。

瑞典 KTH Mechanics 的研究组开发了一套基于 Nek5000 的包含 SFD 方法求解基本流 + 线性稳定性分析 + 优化控制的开源程序包，并成功用于处理射流问题（参见文献 [16]），

但该问题基本流求解过程不含能量方程。我们在其代码基础上添加了能量方程的滤波，用以求解诸如热对流问题，或 2D 轴对称问题（第三方向速度分量储存在温度中，通过“伪”的能量方程进行时间推进）的基本流。下面我们以旋转顶盖驱动的圆柱内流动作为例子进行介绍。该例为 2D 轴对称问题，$Re = 2700$。

图 4.3: setup.sh 编译命令行.

```
#!/bin/bash
SETUP=box
# commands
cp_cmd=cp
mv_cmd=mv
rm_cmd=rm
# paths
NEK_HOME=/home/user14/nek5_svn/trunk/nek
#modified files
NEKFILES=''
INCFILES='USERPAR'
# makenek variables
F77="mpif77"
export F77
CC="mpicc"
export CC
PPLIST=""
export PPLIST
USR="IO_tools.o uparam.o chkpoint.o sponge.o sfd.o"
export USR
USR_LFLAGS=""
export USR_LFLAGS
```

由于我们需要编写新的.f 文件并与 Nek5000 源码一同编译，可参照1.2中介绍的方法进行编译，这里我们编写了专门的 setup.sh 脚本来简化这一过程。将新写的.f 文件全放在算例目录下的 compile/src 目录中。图4.3是 setup.sh 编译命令行部分的写法，完整脚本附在附录E中，在该目录下输入：

bash setup.sh compile

即可完成编译，生成的可执行文件在 bin 中。将包含网格信息的.rea 文件放入算例目录下 case 中，此时可以进行正常计算了。用3.2.1一节中介绍过的 hpts 子程序监测流场变量的时间序列，在不同的滤波参数下的时间序列如图4.4 所示，可见通过 SFD 的处理我们得到了不稳定的定常解，可作为稳定性分析的基本流。若滤波参数选取较合适，周期流很快能收敛到定常解，说明了该方法在解决一部分问题上的有效性。

4.2.2　求解特征值问题

有了基本流之后可以对方程进行线化，首先介绍一下线化方程的求解设置，以 2D 圆柱绕流问题为例，计算域如图 2.3(a) 所示。SIZE 中 perturbation 部分的参数按图4.5进行设置。.usr 内 userbc 和 useric 中定义的边界条件和初始条件需按求解变量为扰动值来重新设置。将.rea 中的 para31 设置成 1（-1），表示线化求解；同时将如图 1.3 所示的续算文件位置输入基本流文件名。时间步只能用固定步长，且时间精度需为 2 阶，因更高阶时间步推进容易使计算发散。这里 $Re = 10$，据已有知识流动应是稳定的。设置完毕后同样用 4.2.1 一节

图 4.4: 周向速度 w 的时间序列.

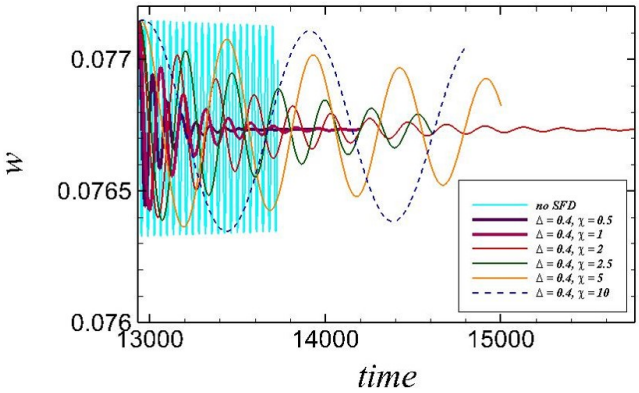
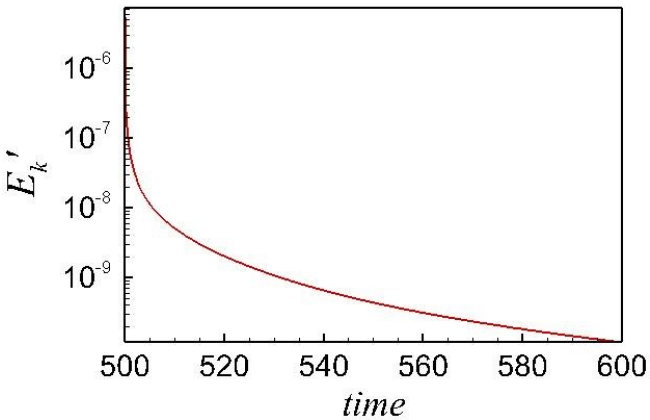


图 4.5: SIZE 中扰动矩阵空间参数设置.

```
parameter (lpelv=lelv,lpelt=lelt,lpert=1)  ! perturbation
parameter (lpx1=lx1,lpy1=ly1,lpz1=lz1)     ! array sizes
parameter (lpx2=lx2,lpy2=ly2,lpz2=lz2)
```

中 setup.sh 的方式进行编译。运行后监测扰动动能 $E'_k = \frac{1}{2}u_iu_i$ 的时间序列如图 4.6所示，可见 E'_k 随时间指数衰减，符合流动稳定的特征。

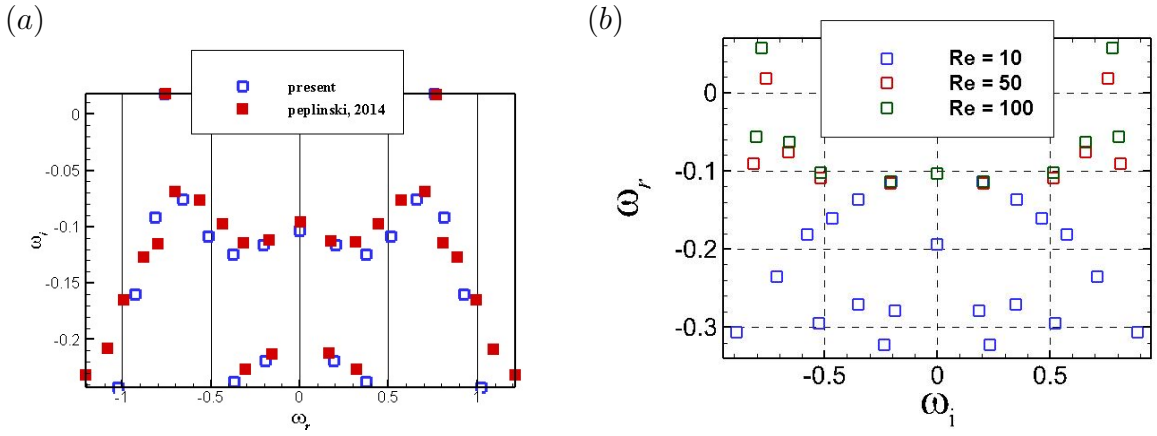
图 4.6: $Re = 10$ 时的 $E'_k - t$ 曲线.



随后在时间推进 LNS 方程的基础上进一步结合 Arnoldi 迭代方法来求解特征值问题。这里使用 KTH Mechanics 组开发的线性稳定性工具（见会议报告 [15]）。新增加的.f 文件同样放在 compile/src 目录下，其中调用 Arpack 的接口函数写在了 arnoldi_arpack.f 中，读者可自行查看相关的写法。我们稍微修改了 Ritz pair 及特征值的输出部分使数据易被 tecplot 读取。按 setup.sh 以及1.2一节中介绍的链接外部库的方法进行编译即可。

图4.7(a) 是与文献 [15] 计算特征值谱的对比，可见最不稳定的特征值能完全对上。图4.7(b) 是不同 Re 下的特征值谱，$Re = 10$ 时流动稳定，因此优势特征值实部小于 0，而 $Re = 50$ 和 100 时流动失稳，因此特征值实部大于 0，计算结果与已有知识是定性符合的。

图 4.7: 圆柱绕流算例特征值谱.



需要指出的是，在测试含有能量方程的圆筒中 RB 对流算例时，遇到二次失稳无法捕捉的问题，说明程序内部设置上可能仍有细节没有探索清楚，仍有 bug 没有找出。这部分工具需要更详尽的测试后方能使用。

后记

学习和使用 Nek5000 的过程中得到了许多人的帮助。感谢实验室刘苗苗同学在网格预处理上的工作，她贡献了2.1中 prenek 使用部分的内容。感谢 Argonne National Lab 的 P. Fischer 在邮件中以及用户社区里对我的各种初级问题的解答。感谢 ETH 的 S. Kerkemeier 在邮件中对变物性参数部分疑问的解答。感谢 KTH Mechanics 的 A. Peplinski 在邮件中对他们开发软件包使用上的一些指导。同时要感谢 Nek5000 用户社区的所有贡献者，他们的问答解决了新手的很多问题。

Nek5000 是个功能强大的开源软件，以上的介绍远非该代码的全部内容。据调研，小马赫数方程，直接/伴随线化方程，目标函数优化，Lagrangian Particle Tracking 等我们感兴趣的问题均有望用 Nek5000 来求解，但这部分内容截至本文完稿时还没有仔细研究过其源码，亦没有进行过任何算例测试，只能留待日后研究了。我希望本文的介绍足够简单易懂，如果读者看到这里时已经可以自如的设置想研究的物理问题的算例，那么本文的目的已经达到了。

徐博伦　　于力一楼 *526 TCFD Lab*

更新日志

2018/02/28 v1.0 初稿完成

2018/03/01 v1.1 增加了附录E。修改格式及文字错误。修改封面风格。

2018/03/02 v1.2 增加了 github 链接。增添、优化了附录中 Matlab 部分代码。

2018/03/12 v1.3 增加、修改了部分介绍语句，修改、优化了附录中 Matlab 部分代码。

2018/05/02 v1.4 增加了部分介绍语句，优化了 github 上载的 Matlab 部分代码。

附录 A　visIt 体积渲染后处理脚本 volume.py 示例

```
import math
# Step 1: Open a database
OpenDatabase("/home/user14/home/rotation/gamma=4/Ra=1E7/kai=2/iRo=0.1/covann.nek5000")
save=1
for state in range(TimeSliderGetNStates()):
SetTimeSliderState(state)
i=i+1
ii=i-255
# antialialiasing
r = GetRenderingAttributes()
r.antialiasing = 1
SetRenderingAttributes(r)
# Annotation
a = AnnotationAttributes()
a.axes3D.visible = 0
a.userInfoFlag = 0
a.timeInfoFlag = 0
a.databaseInfoFlag = 0
a.axes3D.triadFlag = 1
a.legendInfoFlag = 0
a.axes3D.bboxFlag = 0
a.gradientBackgroundStyle = a.TopToBottom # TopToBottom, BottomToTop, LeftToRight,
RightToLeft, Radial
a.gradientColor1 = (0, 0, 0, 255)
a.gradientColor2 = (128, 128, 128, 255)
a.backgroundMode = a.Gradient
```

```
SetAnnotationAttributes(a)
AddPlot("Volume", "temperature")
DrawPlots()
p = VolumeAttributes()
p.samplesPerRay = 1024
p.rendererType = p.RayCasting # Splatting, Texture3D, RayCasting, RayCastingIntegration, SLIVR, RayCastingSLIVR, Tuvok
p.gradientType = p.SobelOperator # CenteredDifferences, SobelOperator
p.num3DSlices = 200
p.scaling = p.Linear # Linear, Log, Skew
p.skewFactor = 1
p.limitsMode = p.OriginalData # OriginalData, CurrentPlot
p.sampling = p.Rasterization # KernelBased, Rasterization, Trilinear
p.rendererSamples = 3
p.transferFunctionDim = 1
p.lowGradientLightingReduction = p.Lower # Off, Lowest, Lower, Low, Medium, High, Higher, Highest
p.lowGradientLightingClampFlag = 0
p.lowGradientLightingClampValue = 1
# Animate the view
c = GetView3D()
c.viewUp = (-0.0529, 0.46689, 0.88273)
c.viewAngle = 30
c.viewNormal = (0.11458, -0.87529, 0.46982)
c.focus = (0, 0, 0.5)
c.imageZoom = 1.2
c.shear = (0, 0, 1)
SetView3D(c)
# Make Jet colormap
p.colorControlPoints.ClearControlPoints()
c = ColorControlPoint()
c.colors = (0, 0, 127, 255)
c.position = 0
p.colorControlPoints.AddControlPoints(c)
c.colors = (0, 0, 255, 255)
c.position = 0.125
```

```
p.colorControlPoints.AddControlPoints(c)
c.colors = (0, 255, 255, 255)
c.position = 0.375
p.colorControlPoints.AddControlPoints(c)
c.colors = (255, 255, 0, 255)
c.position = 0.625
p.colorControlPoints.AddControlPoints(c)
c.colors = (255, 0, 0, 255)
c.position = 0.875
p.colorControlPoints.AddControlPoints(c)
c.colors = (127, 0, 0, 255)
c.position = 1
p.colorControlPoints.AddControlPoints(c)
p.freeformOpacity = (0, 0, 0, 0, 0, 4, 5, 6, 8, 9, 11, 13, 15, 17, 20, 23, 27, 30, 35, 39, 44, 50,
56, 62, 69, 76, 84, 92, 101, 109, 118, 128, 137, 147, 157, 167, 176, 186, 195, 204, 212, 220, 227,
233, 239, 244, 248, 251, 253, 254, 254, 253, 252, 249, 245, 240, 235, 228, 221, 213, 205, 196,
187, 178, 168, 159, 149, 139, 130, 120, 111, 102, 94, 85, 78, 70, 63, 57, 51, 45, 40, 35, 31, 27,
24, 21, 18, 15, 13, 11, 9, 8, 7, 6, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 6,
7, 8, 9, 11, 13, 15, 18, 21, 24, 27, 31, 35, 40, 45, 51, 57, 63, 70, 77, 85, 93, 101, 110, 119, 129,
138, 148, 157, 167, 177, 186, 195, 204, 212, 220, 227, 233, 239, 244, 248, 251, 253, 254, 254,
254, 252, 249, 245, 240, 234, 228, 221, 213, 205, 196, 187, 177, 168, 158, 148, 139, 129, 119,
110, 101, 93, 85, 77, 69, 63, 56, 50, 45, 39, 35, 31, 27, 23, 20, 17, 15, 13, 11, 9, 8, 6, 5, 4, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) SetPlotOptions(p)
# Step 4: Animate through time and save images
if save == 1:
s = SaveWindowAttributes()
s.family = 0
s.format = s.PNG
s.width = 1024
s.height = 1024
s.quality = 100
s.resConstraint = s.NoConstraint
s.advancedMultiWindowSave = 0
s.forceMerge = 0
fid="%05d" % ii
```

```
framefilename = "T"+fid
print framefilename
s.fileName = framefilename
SetSaveWindowAttributes(s)
n = SaveWindow()
DeleteAllPlots()
ClearCacheForAllEngines()
```

附录 B　tecplot 制作动画脚本 anime.mcr 示例

```
#!MC 1410
$!VarSet |MFBD| = 'D: Program Files Tecplot Tecplot 360 EX 2017 R1'
$!varset |loopnum| = 135
$!loop |loopnum|
$!if |loop| <= 135
$!if |loop| <= 99
$!if |loop| <= 9
$!READDATASET '"G: data studs Ra=1E10 s8 anime visit_ex_db000|loop|.tec" '
$!else
$!READDATASET '"G: data studs Ra=1E10 s8 anime visit_ex_db00|loop|.tec" '
$!endif
$!endif
$!if |loop| > 99
$!READDATASET '"G: data studs Ra=1E10 s8 anime visit_ex_db0|loop|.tec" '
$!endif
READDATAOPTION = NEW
RESETSTYLE = YES
VARLOADMODE = BYNAME
ASSIGNSTRANDIDS = YES
VARNAMELIST = '"X" "Y" "temperature"'
$!GLOBALRGB REDCHANNELVAR = 3
$!GLOBALRGB GREENCHANNELVAR = 3
$!GLOBALRGB BLUECHANNELVAR = 3
$!SETCONTOURVAR
VAR = 3
CONTOURGROUP = 1
```

```
LEVELINITMODE = RESETTONICE
$!FIELDLAYERS SHOWCONTOUR = YES
$!GLOBALCONTOUR 1 COLORMAPFILTERCOLORMAPDISTRIBUTION = CONTIN-
UOUS
$!GLOBALCONTOUR 1 COLORMAPNAME = 'Modified Rainbow - Dark ends'
$!GLOBALCONTOUR 1 LEGENDSHOW = NO
$!TWODAXIS YDETAILSHOWAXIS = NO
$!TWODAXIS XDETAILSHOWAXIS = NO
$!TWODAXIS VIEWPORTPOSITIONX1 = 0
$!TWODAXIS VIEWPORTPOSITIONX2 = 100
$!TWODAXIS VIEWPORTPOSITIONY2 = 100
$!TWODAXIS VIEWPORTPOSITIONY1 = 0
$!FRAMECONTROL ACTIVATEBYNUMBER
FRAME = 1
$!FRAMELAYOUT SHOWBORDER = NO
$!FRAMECONTROL MOVETOTOPACTIVE
$!FRAMELAYOUT WIDTH = 5
$!FRAMELAYOUT HEIGHT = 5
$!PICK ADDATPOSITION
X = 9.98979591837
Y = 2.71818727491
CONSIDERSTYLE = YES
$!VIEW FIT
CONSIDERBLANKING = YES
$!REDRAWALL
$!EXPORTSETUP IMAGEWIDTH = 520
$!EXPORTSETUP EXPORTFNAME = 'G: data studs Ra=1E10 s8 anime anime|loop|.png'
$!EXPORT
EXPORTREGION = CURRENTFRAME
$!else
$!break
$!endif
$!endloop
$!RemoveVar |MFBD|
```

其中包含了色调、坐标轴、显示大小等的调整，使得图形较为清晰易读。

附录 C　Matlab 后处理程序部分示例

Nek5000 流场文件读入脚本 readnek.m/并行读入一组文件脚本 serialread.m

readnek.m

```
function [time,istep,nel,N,data,datax,datay,dataz,datau,datav,dataw,datat] = readnek(fname)
emode = 'le';
[infile,message] = fopen(fname,'r',['ieee-' emode]);
if infile == -1, disp(message), status = -1; return, end
% read header header = fread(infile,132,'*char')';
% check endian
etag = round(fread(infile,1,'float32')1e5);
if (etag ~= 654321)
fclose(infile);
emode = 'be';
[infile,message] = fopen(fname,'r',['ieee-' emode]);
header = fread(infile,132,'*char')';
etag = round(fread(infile,1,'*float32')*1e5);
if (etag ~ = 654321)
disp('ERROR: could not interpret endianness.')
status = -3; return
end
end
etag = etag * 1e-5;
% READ HEADER
% word size
wdsz = str2double(header(6));
if (wdsz == 4)
```

```
realtype = '*float32';
elseif (wdsz == 8)
realtype = '*float64';
else
fprintf('ERROR: could not interpret real type (wdsz = %i)',wdsz);
status = -2; return
end
% element size
lr1 = [str2double(header(8:9))str2double(header(11:12))str2double(header(14:15))];
% compute the total number of points per element
npel = prod(lr1);
% compute number of active dimensions
ndim = 2 + (lr1(3)>1);
% number of elements nel = str2double(header(17:26));
% number of elements in the file nelf = str2double(header(28:37));
% time
time = str2double(header(39:58);
% istep
istep = str2double(header(60:68));
% get file id
fid = str2double(header(70:75));
% get tot number of files
nf = str2double(header(77:82));
% getfields [XUPT]
fields = strtrim(header(84:end));
var=zeros(1,5);
if sum(fields == 'X') > 0
var(1) = ndim;
end
if sum(fields == 'U') > 0
var(2) = ndim;
end
if sum(fields == 'P') > 0
var(3) = 1;
end
if sum(fields == 'T') > 0
```

```
var(4) = 1;
end
nfields = sum(var);
% read element map
elmap = fread(infile,nelf,'int32').';
% READ DATA
data = zeros(nelf,npel,nfields);
for ivar = 1:length(var)
idim0 = sum(var(1:ivar-1));
for iel = elmap
for idim = (1:var(ivar))+idim0
data(iel,:,idim) = fread(infile,npel,realtype);
end
end
end
% Dump field data
[nel,N3,nfld] = size(data); N = power(N3,1/3);
if nfld == 8
datax = data(:,:,1);
datay = data(:,:,2);
dataz = data(:,:,3);
datau = data(:,:,4);
datav = data(:,:,5);
dataw = data(:,:,6);
datat = data(:,:,8);
end
if nfld == 5
datau = data(:,:,1);
datav = data(:,:,2);
dataw = data(:,:,3);
datat = data(:,:,5);
end
% CLOSE FILE status = fclose(infile);
end
```

serialread.m

```
function[datax0,datay0,dataz0,datau0,datav0,dataw0,datat0] = serialread(fnameh,id0,id1)
%read group of .fxxxx files by loop (from f%id0   f%id1)
for i = id0:id1
if(i<10)
fname(i).name = sprintf('%s%s%s%d','fields/',fnameh,'0.f0000',i);
elseif(i<100)
fname(i).name = sprintf('%s%s%s%d','fields/',fnameh,'0.f000',i);
elseif(i<1000)
fname(i).name = sprintf('%s%s%s%d','fields/',fnameh,'0.f00',i);
elseif(i<10000)
fname(i).name = sprintf('%s%s%s%d','fields/',fnameh,'0.f0',i);
else
fname(i).name = sprintf('%s%s%s%d','fields/',fnameh,'0.f',i);
end
end
%read the first file to dump the geometrical data (make sure there are mesh data in this file)
[time,istep,nel,N,data,datax,datay,dataz,datau,datav,dataw,datat] = readnek(fname(id0).name);
datau0(:,:,id0) = datau(:,:);
datav0(:,:,id0) = datav(:,:);
dataw0(:,:,id0) = dataw(:,:);
datat0(:,:,id0) = datat(:,:);
for i = id0:id1
datax0(:,:,i) = datax(:,:);
datay0(:,:,i) = datay(:,:);
dataz0(:,:,i) = dataz(:,:);
end
%read other files in parallel
parfor i=(id0+1):id1
[time,istep,nel,N,data,datax,datay,dataz,datau,datav,dataw,datat]= readnek(fname(i).name);
datau0(:,:,i) = datau(:,:);
datav0(:,:,i) = datav(:,:);
dataw0(:,:,i) = dataw(:,:);
datat0(:,:,i) = datat(:,:);
end
```

```
end
```

3D 环域插值脚本 intpnek.m/多文件并行插值脚本 serialintp.m

intpnek.m

```
function[U,V,W,T,Xm,Ym,Zm] = ...
intpnek(datax,datay,dataz,datau,datav,dataw,datat,r1,r2,meshr,meshphi,meshz)
%create an annular cylinder domain for interpolation
r = linspace(r1,r2,meshr);
phi = linspace(0,2*pi,meshphi);
z = linspace(0,1,meshz);
X1=r'*cos(phi);
Y1=r'*sin(phi);
for i=1:meshz
Xm(:,:,i)=X1(:,:);
Ym(:,:,i)=Y1(:,:);
end
for i=1:meshr
for j=1:meshphi
for k=1:meshz
Zm(i,j,k)=z(k);
end
end
end
%linearly interpolate flow field onto new homogeneous annular mesh
U = griddata(datax,datay,dataz,datau,Xm,Ym,Zm);
V = griddata(datax,datay,dataz,datav,Xm,Ym,Zm);
W = griddata(datax,datay,dataz,dataw,Xm,Ym,Zm);
T = griddata(datax,datay,dataz,datat,Xm,Ym,Zm);
U(meshr,:,:)=U(meshr-1,:,:);
V(meshr,:,:)=V(meshr-1,:,:);
W(meshr,:,:)=W(meshr-1,:,:);
T(meshr,:,:)=T(meshr-1,:,:);
```

```
end
```

serialintp.m

```
function[flow,ord] = ...
serialintp(r1,r2,meshr,meshphi,meshz,datax0,datay0,dataz0,datau0,datav0,dataw0,datat0,id0,id1)
%create an annular cylinder domain for interpolation
r = linspace(r1,r2,meshr);
phi = linspace(0,2*pi,meshphi);
z = linspace(0,1,meshz);
X1=r'*cos(phi);
Y1=r'*sin(phi);
for i=1:meshz
Xm0(:,:,i)=X1(:,:);
Ym0(:,:,i)=Y1(:,:);
end
for i=1:meshr
for j=1:meshphi
for k=1:meshz
Zm0(i,j,k)=z(k);
end
end
end
for i=id0:id1
ord(i).Xm(:,:,:) = Xm0(:,:,:);
ord(i).Ym(:,:,:) = Ym0(:,:,:);
ord(i).Zm(:,:,:) = Zm0(:,:,:);
end
%interpolation in parallel
parfor i=id0:id1
flow(i).U = griddata(datax0,datay0,dataz0,datau0,ord(i).Xm,ord(i).Ym,ord(i).Zm);
flow(i).V = griddata(datax0,datay0,dataz0,datav0,ord(i).Xm,ord(i).Ym,ord(i).Zm);
flow(i).W = griddata(datax0,datay0,dataz0,dataw0,ord(i).Xm,ord(i).Ym,ord(i).Zm);
flow(i).T = griddata(datax0,datay0,dataz0,datat0,ord(i).Xm,ord(i).Ym,ord(i).Zm);
end
for i=id0:id1
```

```
%1-order on the boundary
flow(i).U(meshr,:,:)= flow(i).U(meshr-1,:,:);
flow(i).V(meshr,:,:)= flow(i).V(meshr-1,:,:);
flow(i).W(meshr,:,:)= flow(i).W(meshr-1,:,:);
flow(i).T(meshr,:,:)= flow(i).T(meshr-1,:,:);
end
end
```

柱坐标下流场数据输出脚本 outputTec.m/并行输出多文件脚本 PoutputTec.m

outputTec.m

```
function[]=outputTec(fnameh,meshr,meshphi,meshz,Xm,Ym,Zm,a,b,c,d)
fname = sprintf('%s%s%s','results/',fnameh,'.dat');
h1 = fopen(fname, 'w');
fprintf(h1,'TITLE = "data from MATLAB"  n');
fprintf(h1,'VARIABLES = "X" "Y" "Z" "U" "V" "W" "T" n');
fprintf(h1,'ZONE T = "domain 0 " n');
fprintf(h1,'I = %d, J = %d, K = %d, F= POINT n ',meshr,meshphi,meshz );
for n = 1:meshz
for m = 1:meshphi
for l = 1:meshr
fprintf(h1,'%f %f %f n ', Xm(l,m,n),Ym(l,m,n),Zm(l,m,n) );
fprintf(h1,'%f %f %f n', a(l,m,n),b(l,m,n),c(l,m,n) );
fprintf(h1,'%f n',d(l,m,n) );
end
end
end
close all;
end
```

PoutputTec.m

```
function[]=PoutputTec(meshr,meshphi,meshz,Xm,Ym,Zm,a,b,c,d,fnameh,id0,id1)
% dump the temporal seires of flow fields into tec files in parallel
```

```
for i = id0:id1
fname(i).name = sprintf('%s%s%d%s','results/',fnameh,i,'.dat');
end
parfor i = id0:id1 %output in parallel
h(i) = fopen(fname(i).name, 'w');
outputTec(h(i),fname(i).name,meshr,meshphi,meshz,Xm(:,:,:,i),Ym(:,:,:,i), ...
Zm(:,:,:,i),a(:,:,:,i),b(:,:,:,i),c(:,:,:,i),d(:,:,:,i));
end
close all;
end
```

附录 D　hpts.out 中流场数据提取脚本 readdata.f90

3.2.1一节介绍了记录流场监测点信息的输出文件 hpts.out，若监测点不止一个，文件中记录就比较冗长。为了使文件内数据方便用 tecplot 等后处理软件读入，可使用 readdata.f90 脚本选择任一监测点数据输出，运行后按屏幕输出菜单中的提示操作即可。

```
program readdata
implicit none
integer i,j,m,n,k,l,flag1,flag2
real,dimension(:,:), allocatable :: a
real,dimension(:,:), allocatable :: a1
integer :: status=0
character(len=100)::line
character(len=100)::fileout
write(*,*)'———problem dimension———'
write(*,*)'1(2D) 2(3D)'
read(*,*)flag1
write(*,*)'———energy equation?———-'
write(*,*)'1(YES) 2(NO)'
read(*,*)flag2
if(flag1.eq.1)then
if(flag2.eq.1)then
m=5
elseif(flag2.eq.2)then
m=4
endif
elseif(flag1.eq.2)then
if(flag2.eq.1)then
m=6
```

```
elseif(flag2.eq.2)then
m=5
endif
endif
write(*,*)'number of monitor points:'
read(*,*)k
open(unit=11, file='hpts.out')
n=0
do
read(11,'(A)',iostat=status)line
if(status.lt.0)exit
if(len_trim(line).eq.0)cycle
n=n+1
enddo
allocate(a(n-1,m))
allocate(a1((n-1)/k,m))
rewind(11)
read(11,*)
do i=1,n-1
read(11,*)a(i,1:m)
enddo
close(11)
write(*,*)'output which point:'
read(*,*)l
do i=1,(n-1)/k
do j=1,m
a1(i,j)=a(k*(i-1)+l,j)
enddo
enddo
write (fileout,"(I10.4)")l
fileout = 'hpts'//trim(adjustl(fileout))//'.dat'
write(*,*)fileout
open(unit=12, file= fileout)
write(12,*)'time,vx,vy,vz,pr,t'
do i=1,(n-1)/k
write(12,"(6e15.7)")a1(i,1:m)
```

```
enddo
close(12)
write(*,*)'output complete!'
end program
```

附录 E　程序编译 bash 文件 setup.sh 示例

以编译4.2.2小节中的算例为例。

```
#!/bin/bash
SETUP=ext_cyl
# commands
cp_cmd=cp
mv_cmd=mv
rm_cmd=rm
# paths
NEK_HOME=../../../../nek5_1093/trunk/nek
#modified files！用到的 core 目录下的源码文件，可以单独拷贝到 compile/src 下以减少编译文件量
NEKFILES='ADJOINT INPUT drive2.f gmres.f navier4.f perturb.f planx.f subs1.f'
INCFILES=''
# makenek variables
F77="mpif77"
export F77
CC="mpicc"
export CC
PPLIST=""
export PPLIST
USR="IO_tools.o uparam.o chkpoint.o sponge.o time_stepper.o arnoldi_arpack.o uzawa.o conht_tools.o"
export USR！链接上 Arpack 包的外链
USR_LFLAGS="-L./ -lparpack_MPI-gcc_rst -larpack_gcc_rst"
export USR_LFLAGS
# enable MPI (default true)
```

```
#IFMPI="false"
#export IFMPI
#F77="gfortran"
#CC="gcc"
#USR_LFLAGS="-L./ -larpack_gcc_rst"
# arguments
args=("$")
argsnr=$#
case "$args[0]" in
copy_src)
# create local source copy
$cp_cmd -rp $NEK_HOME ./
;;
compile) ！编译命令中输入 compile 来完成编译 # compile
# copy modified files
for i in $NEKFILES; do
echo $i $cp_cmd ./src/$i ./nek/
done
# copy include files
for i in $INCFILES; do
echo $i
$cp_cmd ./src/$i ./
done
# copy .usr file
$cp_cmd ./src/$SETUP.f ./$SETUP.usr
# compile the code
./makenek $SETUP ./nek
$mv_cmd ./nek5000 ./bin 可执行的 nek5000 文件放在下一级目录 bin 中
# clean directory
$rm_cmd $SETUP.f $SETUP.usr makefile
for i in $INCFILES; do
$rm_cmd ./$i
done
;;
clean)
# clean
```

```
$rm_cmd ./bin/nek5000
$rm_cmd ./obj/*
;;
clean_src)
# clean everything $rm_cmd -Rf ./nek ./obj
;;
*) error_quit 'Wrong option'
;;
esac
```

参考文献

[1] Guenter Ahlers, Eric Brown, Francisco Fontenele Araujo, Denis Funfschilling, Siegfried Grossmann, and Detlef Lohse. Non–Oberbeck–Boussinesq effects in strongly turbulent Rayleigh–Bénard convection. *Journal of Fluid Mechanics*, 569:409–445, 2006.

[2] Espen Åkervik, Luca Brandt, Dan S Henningson, Jérôme Hœpffner, Olaf Marxen, and Philipp Schlatter. Steady solutions of the navier-stokes equations by selective frequency damping. *Physics of Fluids*, 18(6):068102, 2006.

[3] Michel O Deville, Paul F Fischer, and Ernest H Mund. *High-order methods for incompressible fluid flow*, volume 9. Cambridge university press, 2002.

[4] Paul F Fischer. An overlapping schwarz method for spectral element solution of the incompressible Navier–Stokes equations. *Journal of Computational Physics*, 133(1):84–101, 1997.

[5] David Goluskin, Hans Johnston, Glenn R Flierl, and Edward A Spiegel. Convectively driven shear and decreased heat flux. *Journal of Fluid Mechanics*, 759:360–385, 2014.

[6] Susanne Horn. Rotation and non–Oberbeck–Boussinesq effects in turbulent Rayleigh–Bénard convection. 2015.

[7] Julian CR Hunt, Alan A Wray, and Parviz Moin. Eddies, streams, and convergence zones in turbulent flows. *Center for Turbulence Research Report CTR-S88*, pages 193–208, 1988.

[8] Jinhee Jeong and Fazle Hussain. On the identification of a vortex. *Journal of Fluid Mechanics*, 285:69–94, 1995.

[9] Bastien E Jordi, Colin J Cotter, and Spencer J Sherwin. An adaptive selective frequency damping method. *Physics of Fluids*, 27(9):094104, 2015.

[10] Argonne National Laboratory. Nek5000 documentation. 2015.

[11] Lawrence Livermore National Laboratory. visIt user's manual, version1.5. 2005.

[12] Lawrence Livermore National Laboratory. visIt python interface manual. 2011.

[13] Nasreddine Ouertatani, Nader Ben Cheikh, Brahim Ben Beya, and Taieb Lili. Numerical simulation of two-dimensional Rayleigh–Bénard convection in an enclosure. *Comptes Rendus Mecanique*, 336(5):464–470, 2008.

[14] Anthony T Patera. A spectral element method for fluid dynamics: laminar flow in a channel expansion. *Journal of Computational Physics*, 54(3):468–488, 1984.

[15] Adam Peplinski, Philipp Schlatter, PF Fischer, and Dan S Henningson. Stability tools for the spectral-element code Nek5000: Application to Jet–in–Crossflow. In *Spectral and High Order Methods for Partial Differential Equations–ICOSAHOM 2012*, pages 349–359. Springer, 2014.

[16] Adam Peplinski, Philipp Schlatter, and Dan Stefan Henningson. Global stability and optimal perturbation for a jet in cross-flow. *European Journal of Mechanics-B/Fluids*, 49:438–447, 2015.

[17] Kazuyasu Sugiyama, Enrico Calzavarini, Siegfried Grossmann, and Detlef Lohse. Non–Oberbeck–Boussinesq effects in two-dimensional Rayleigh–Bénard convection in glycerol. *EPL (Europhysics Letters)*, 80(3):34002, 2007.

[18] Kazuyasu Sugiyama, Enrico Calzavarini, Siegfried Grossmann, and Detlef Lohse. Flow organization in two-dimensional non–Oberbeck–Boussinesq Rayleigh–Bénard convection in water. *Journal of Fluid Mechanics*, 637:105–135, 2009.