Peter A. Jacobs and Rowan J. Gollan

Eilmer4.0 流动仿真程序

----------瞬态流动求解器指南

（包括上手的实例）

2021 年 10 月 12 日更新

译者：卢顺、齐建荟

Technical Report 2017/26

School of Mechanical & Mining Engineering

The University of Queensland

Eilmer User Guide Translation

摘要：

Eilmer 是一种在二维和三维空间中模拟瞬态、可压缩流动的程序。它具有一个准备模式，可用来设定仿真参数数据库、定义流域的多块结构化网格以及初始化内部流场。然后这些设定被用于主要仿真的起始状态，模拟计算得到一系列流动演化的快照。后处理模式允许用户根据兴趣来提取和重新整理流动数据。

Eilmer 的源代码可在 https://bitbucket.org/cfcfd/dgd/获得，同时它与许多其它的可压缩流体力学相关代码有关，它们可以在 http://cfcfd.mechmining.uq.edu.au/找到。

这份用户指南整理了许多仿真例子，它包含：脚本、结果和评价。对于本程序的新用户来说，最方便的方法是找到一个与新用户仿真需求相近的算例，然后调整编辑脚本文件从而达到最终的仿真效果。

本用户指南的英文原版以及几何、气体等配套文件可在以下网站下载：https://gdtk.uqcloud.net/docs/eilmer/user-guide/。

贡献者

Peter A. Jacobs, Rowan J. Gollan, Kyle Damm, Nick Gibbons, Daryl Bond, Ingo Jahn and Anand Veeraragavan, as co-chief gardeners

with contributions from a cast of many, including:

Ghassan Al'Doori, Nikhil Banerji, Justin Beri, Peter Blyton, Viv Bone, Jamie Border, Arianna Bosco, Djamel Boutamine, Laurie Brown, James Burgess, David Buttsworth, Wilson Chan, Eric Chang, Sam Chiu, Chris Craddock, Brian Cook, Jason Czapla, Tim Cullen, Andrew Dann, Andrew Denman, Zac Denman, Luke Doherty, Elise Fahy, Antonia Flocco, Delphine Francois, James Fuata, David Gildfind, Richard Goozee, ´ Sangdi Gu, Birte Haker, Stefan Hess, Jonathan Ho, Hans Hornung, Jimmy-John Hoste, Carolyn Jacobs, Juanita Jacobs, Chris James, Ian Johnston, Reid Jones, Ojas Joshi, Xin Kang, Joshua Keep, Rory Kelly, Rainer Kirchhartz, Jens Kunze, Sam Lamboo, Will Landsberg, Alexis Lefevre, Cor Lerink, Steven Lewis, Yu (Daisy) Liu, Kieran Mackle, Pierre Mariotto, Tom Marty, Matt McGilvray, David Mee, Carlos de MirandaVentura, Christine Mittler, Luke Montgomery, Heather Muir, Jan-Pieter Nap, Brendan O'Flaherty, Reece Otto, Austen Pane, Andrew Pastrello, Paul Petrie-Repar, Jorge Sancho Ponce, Daniel F. Potter, Jason (Kan) Qin, Deepak Ramanath, Andrew Rowlands, Michael Scott, Umar Sheikh, Daniel Smith, Tamara Sopek, Sam Stennett, Ben Stewart, Phillip Swann, Joseph Tang, Katsu Tanimizu, Nils Temme, Augustin Tibere–Inglesse, ` Pierpaolo Toniato, Matthew Trudgian, Paul van der Laan, Tjarke van Jindelt, Jaidev Vesudevan, Han Wei, Mike Wendt, Brad (The Beast) Wheatley, Vince Wheatley, Lachlan Whyborn, Adriaan Window, Hannes Wojciak, Fabian Zander, Mengmeng Zhao

经过多年的努力，贡献者们推动建设了这个代码和文档的示例、调试、校对工作以及提出了建设性意见，形成独树一帜的代码和指南。

致谢：

非常感谢 PAJ 在 Special Studies Program at Laboratoire EM2C, CNRS UPR288,

Centrale-Supélec, Paris 时，准备了这个文档的大部分内容。

目录

1. 介绍 1
1.1 可压缩流动仿真与 Eilmer 代码 1
1.2 代码的历史 2
1.3 更多的信息 3
1.4 许可协议 3
2.开始 5
2.1 操作环境和预备知识 5
2.2 获得源代码 5
2.3 创建并安装程序 5
2.4 运行程序 6
3.Eilmer 的第一个仿真 7
3.1 仿真 8
3.2 结果与后处理 16
3.3 获取数据字段进行专业处理 21
3.4 网格收敛性 23
3.5 第一个例子的其它注意事项 24
3.6 使用 Lua 语言进行参数化建模 24
3.7 气体动力学的探索 27
3.8 建立一个更鲁棒性的仿真 30
4.Eilmer 使用指南 35
4.1 运行仿真 35
4.2 输入脚本概述 44
4.3 指定热化学模型 45
4.4 定义流动状态 46
4.5 导入其它仿真的流动状态 48
4.6 流域的表示 49
4.7 边界条件 54
4.8 特殊区域 61
4.9 记录点 62
4.10 仿真配置及控制参数 63
5.更多仿真例子 72
5.1 斜激波边界层相互作用 73
5.2 氮气在有限长度的圆柱体上流动 85
5.3 尖角锥上重新观察流动 93
参考文献 98

A. Linux 下的命令行 101
B.关于 Lua 语言 104
B.1 基础和语法 104
B.2 运算符和表达式 105
B.3 表 105
B.4 函数 106
B.5 基于对象的编程 107
B.6 控制语句 107
B.7 Lua 中的全局符号 108
C.了解 AWK 脚本 110
D.简单气流函数 112
D.1 理想气体 112
D.2 一般气体 114
E.指定特定工作时间的用户自定义函数 118
E.1 自定义边界条件 118
E.2 源项 124
E.3 网格运动 125
E.4 协同作用 126
E.5 辅助变量、函数和模块 127
E.6 关于 Lua 解释器和全局变量的说明 132

1. 介绍

1.1 可压缩流动仿真与 Eilmer 代码

Eilmer 代码是一种在二维或三维情况下，对瞬态、可压缩气体流动的数值模拟程序。这个程序回答“如果……将会……”类问题，您将会建立一个通过定义空间域的气体流动状况，在这一域中设置气体的初始状态，同时给定边界约束，然后让气体根据气体动力学的规律发展演化。

流动的定义包括有限体积单元的网格、以及相关的边界条件，这些边界条件包括固体无滑移壁面、流入面和流出面等。为了便于这个计算域的设置，本代码包含了一个准备模式，它可用于建立仿真参数数据库、一个定义了封闭和贴体的流场网格，以及一个规范的初始流场。这个准备模式包括网格产生器，它可根据边界条件接收流场的描述，然后生成封闭有限体积的单元网格。在每个块中，底层网格可能是结构化的，也可能是非结构化的。主仿真可计算得到一系列流动演化的快照，它的起始点是生成网格和初始流态。最后，一个基础且实用的后处理代码使流动数据更有利于进一步的分析。

我们将代码设置为可编程的程序，用户可使用自己提供的脚本（用 Lua 语言编写），为任何特定仿真练习提供配置。我们的目标受众是气体动力学的高年级学生，可能是工科类的本科生，但更可能是研究生或科研工作者，他们想模拟气体流动作为他们学术工作的一部分。代码的特征包括：

*欧拉/拉格朗日流动描述（有限体积，二维或三维轴对称）；

*瞬态、时间精确更新和对稳定流动选择性的隐式更新；

*激波捕捉和拟合激波边界层；

*多块、结构化和非结构化网格；

*在共享内存环境下并行计算；

*高温非平衡热化学；

*有限化学速率的 GPU 加速；

*稠密气体热力学模型和旋转参考系下的涡轮机械模型；

*湍流模型；

*固体表面传热和固体内部热流的耦合；

*单相等离子体的 MHD 模拟；

*导入适用于复杂几何流动的 GridPro 和 SU2 网格；

如果您想在设计过程中集成 CFD 分析，最容易想到的可能是一系列流域的形貌或者来流情况，这些变化由一小组参数决定。然后，Eilmer 代码可用于大量仿真模拟，回答“如果我们使用这些特定的参数，流场将会怎样？”的问题。这是我们在使用本程序设计高超声速喷管[1]时所遵循的基本过程，即调整喷管壁面形状，使其向喷管出口平面产生均匀的流场。

1.2 代码的历史

在开发 Eilmer 的过程中，我们的重点一直在生成一个开源代码上，这个代码可被设计成简单接入口，向学生们呈现计算流体动力学知识。我们感兴趣的 CFD 方面包括仿真代码的应用和代码本身的开发。

Eilmer4.0 是对 Eilmer3 程序最优性能的完整重现，而 Eilmer3 是 mbcns2 代码的衍生。mbcns2 是用 C++语言编写 mb_cns 代码的试验品。C++的优点是显而易见的（尽管使用该语言编程会非常的繁琐），我们的三维气体代码 Elmer[1]使用 C++重新编写后命名为 Elmer2。同时，为了使边界条件可编程化，我们尝试使用 Python 语言作为用户输入脚本，并嵌入 Lua 语言。当然，我们很快就发现，如果仅重新编写一些基本的模块，这些在 C++中实验的代码就会变得更加简洁和通用。因此，我们重新编写了热化学模块，并将分离出来的二维和三维代码合并为 Eilmer3。对我们来说这是一个使用非常难以驾驭的语言来编写的庞大代码库（因为我们的职业是机械工程师）。所以当有一种便利的可选代码出现时，我们就“跳船”了，同时我们使用耦合了 Lua 脚本语言(http://www.lua.org)的 D 编程语言(http://dlang.org)，重新构建了 Eilmer3 中最佳部分。

这个代码是以 Malmesbury[2]的 Eilmer 命名的，拼写的选择是为了避免与芬兰

[1] 是的，为了避免与芬兰的有限元代码重名，近些年名称发生了变化。
[2] https://en.wikipedia.org/wiki/Eilmer_of_Malmesbury.

Elmer 有限元代码[3]的命名冲突。

1.3 更多的信息

以下部分提供了大量仿真的输入脚本和命令行脚本的实例。这些是您自己模拟仿真的起点，您应该学习这些代码和其它手册，这些手册可以在可压缩流动 CFD 组网站 http://cfcfd.mechmining.uq.edu.au/eilmer 中找到。认真研究脚本例子，可发现脚本中嵌入了一些非常有意思的技巧。

如果需要查找关于植入到 Eilmer 中的方法，请参考指南报告[2]和论文[3]，它们涵盖了气体动力学公式。更多关于全套仿真程序包各类组件的详细信息，可以在以下文件中找到：

- a brief introduction to the D-language implementation of the Eilmer code[4]；
- a guide to the gas model and basic thermochemistry package[5]；
- a guide to the reacting-gas model with finite-rate kinetics[6]；
- a guide to the look-up table gas model[7]；
- a guide to the geometric modelling package[8]；
- a guide to the shock-fitting boundary condition[9]；

1.4 许可协议

对于源代码，我们使用 GNU 通用公共许可证 3 。请查看源代码中的 gpl.txt 文件。对于文档，比如这份用户指南，我们使用了 Creative Commons Attribution-ShareAlike 4.0 国际许可。

我们希望通过使用 Eilmer，您能够生产出高质量的仿真来帮助您的工作。当需要向别人报告您的 Eilmer 仿真结果时，我们要求您引用以下关于 Eilmer 代码的论文来认可我们的工作：

[3] http://www.csc.fi/elmer.

Jacobs, P.A. and Gollan, R.J. (2016). Implementation of a Compressible-Flow Simulation Code in the D Programming Language. Advances of Computational Mechanics in Australia Volume 846, pages 54–60, in the series AppliedMechanicsandMaterials(DOI:10.4028/www.scientific.net/AMM.846.54)

Gollan, R.J. and Jacobs, P.A. (2013). About the formulation, verification and validation of the hypersonic flow solver Eilmer. International Journal for Numerical Methods in Fluids 73(1):19-57 (DOI: 10.1002/fld.3790)

2.开始

代码求解器及其模块主要用 D 语言编写，可实现在编译时快速检查和高效益性。预处理和后处理模块充分使用了 Lua 脚本语言，使得我们可得到灵活和便捷的个性化服务。自动化脚本处理中也使用了一些 Tcl/Tk。

2.1 操作环境和预备知识

我们的主要开发环境是 Linux，但程序可以在部署在 Linux 上，类似于 MacOS-X 和 MS-Windows 的 Unix。对环境的主要要求是需要提供有效的 D 语言编译器和 Tcl 解释器。Eilmer 源代码储存库中包含了 Lua 解释器的源代码。如果您不习惯使用 Unix 或 Linux，请参阅附录 A，了解在命令行上工作的简要介绍。

除了对您的计算机环境的要求外，我们还假设您的数学、科学或工程背景为计算流体力学分析做了充足的准备。特别地，我们假设您具备几何学、微积分、力学、热流体动力学的知识背景，至少能达到大学二年级或三年级的水平。在 Eilmer 代码中，我们尝试把可压缩、带有化学反应的流动的分析变得更容易获取和可靠，我们不能把这些细节当成是微不足道的事情。

2.2 获得源代码

Eilmer 的全部源代码和一系列例子可以在 BitBucket 的公开存储库上找到。为了得到复制的代码，需要使用 Git 版本的控制客户端来复制存储库，这个储存库中包含类似下面的命令行：

```
$  git clone https://bitbucket.org/cfcfd/dgd-git dgd   (不要复制$符号)
```

在几分钟内，具体时间取决于您的网络连接速度，您就能得到您自己完整的源代码树副本和完整的存储库历史。

2.3 创建并安装程序

一旦您克隆了这个存储库，所有需要的就是一个 Linux 系统，并带有相对较新的 D 编译器和 C 编译器（用于构建 Lua 解释器）。您可参考使用 DMD 编译器

或 LDC 编译器。我们一直致力于开发带有最新版本的 DMD64 和 LDC 编译器的代码。

当访问 dgd/src/eilmer 目录时，您会发现一个 makefile 文件，它可以允许使用 make install 命令行进行构建。默认情况下，可执行程序和支持文件将安装在 $HOME/dgdinst/目录中。

2.4 运行程序

为了运行程序，可能需要为您的文字界面设置环境变量。在最新的 Ubuntu 系统中，在后缀名为.bash_aliases 的文件中输入以下命令:

```
export DGD_REPO=${HOME}/dgd
export DGD=${HOME}/dgdinst
export PATH=${PATH}:${DGD}/bin
export DGD_LUA_PATH=${DGD}/lib/?.lua
export DGD_LUA_CPATH=${DGD}/lib/?.so
```

如果您已经将储存器中的副本克隆到$HOME/dgd/之外的某个地方，那么设置 DGD_REPO 变量就相当便利了。

以下阶段完成之后仿真将真正开始运行。

1.通过翻译 Lua 输入脚本准备网格和初始流动状态配置；

2.运行主仿真程序，从准备好的初始流态开始，允许流场按照时间发展演化，输出特定时刻的流场状态结果；

3.对仿真数据后处理，提取出感兴趣的特定数据；

实际上这种描述太过浅显，不能期望您能够在没有进一步命令情况下运行仿真。如果您感兴趣，可以现在尝试下一章中的教程示例。

3.Eilmer 的第一个仿真

我们从一个简单到可以想象的流动开始：理想空气流过一个尖锥形的超声速子弹。下面的图 3.1 是从 Maccoll1937 年的论文[10]中的图 3 复制来的，展示了一个两磅重、速度为 1.576 倍马赫数的子弹的阴影图像。我们将我们的模型限制在只有气体流入并往上移动到子弹的圆锥表面，且在附在子弹体上的参考系上工作。进一步的，我们假设所有在三维流动中有趣的特征都在二维平面上呈现。红线标出了气体流动模拟的区域，并假设子弹是关于轴线轴对称的。

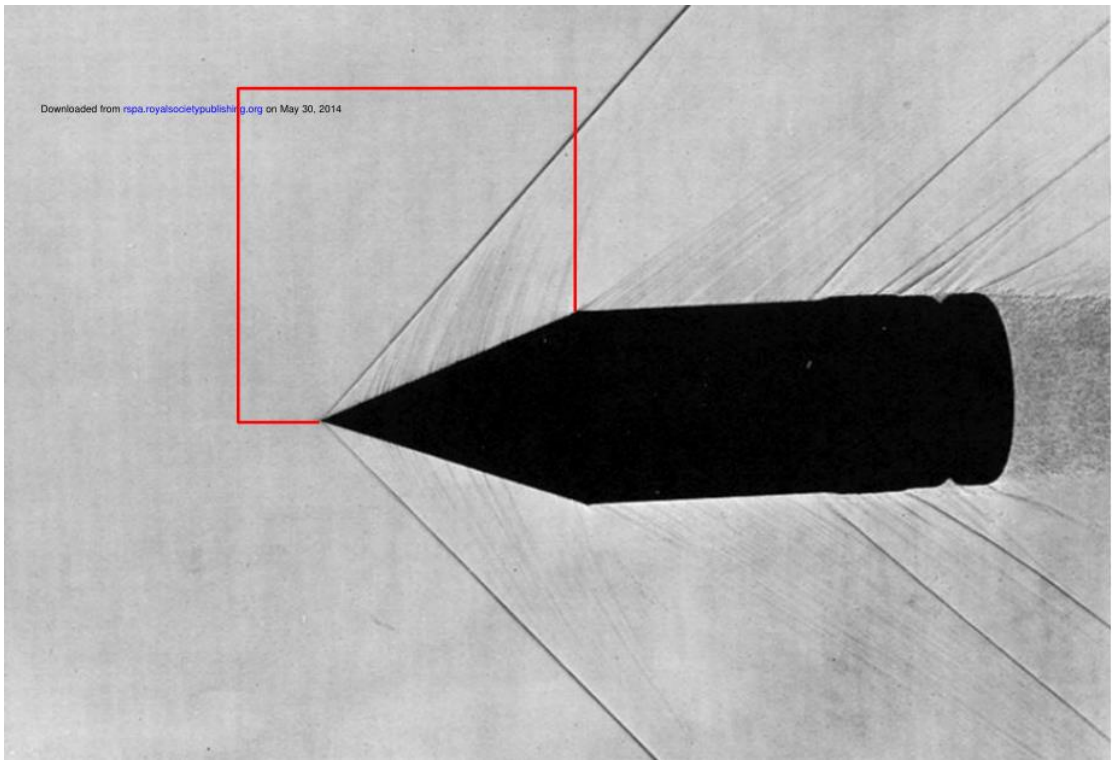


图 3.1：一个飞行中的两磅重子弹，锥形激波依附在子弹的圆锥面上。这幅图片由 Maccoll 在 1937 年发布。添加的红线可区分我们设置模拟的气体流动区域。

在稳态极限下，产生的气流应该只有一个激波，并且激波在二维平面上是直的（但在原先三维空间中是圆锥形的）。这个激波的角度可根据 Taylor 和 Maccoll 的气体动力学理论检验，并且，由于仿真所需要的计算资源很少（在内存和运行时间上来讲），这对检查仿真和绘图程序是否创建并安装正确非常有用。

3.1 仿真

为了建立我们的仿真，我们从图 3.1 中抽象出一个箱型区域，并考虑理想的无黏气体在一个 20 度半角尖锥上做轴对称流动。轴对称的约束意味着原始三维流动的入射角为零。

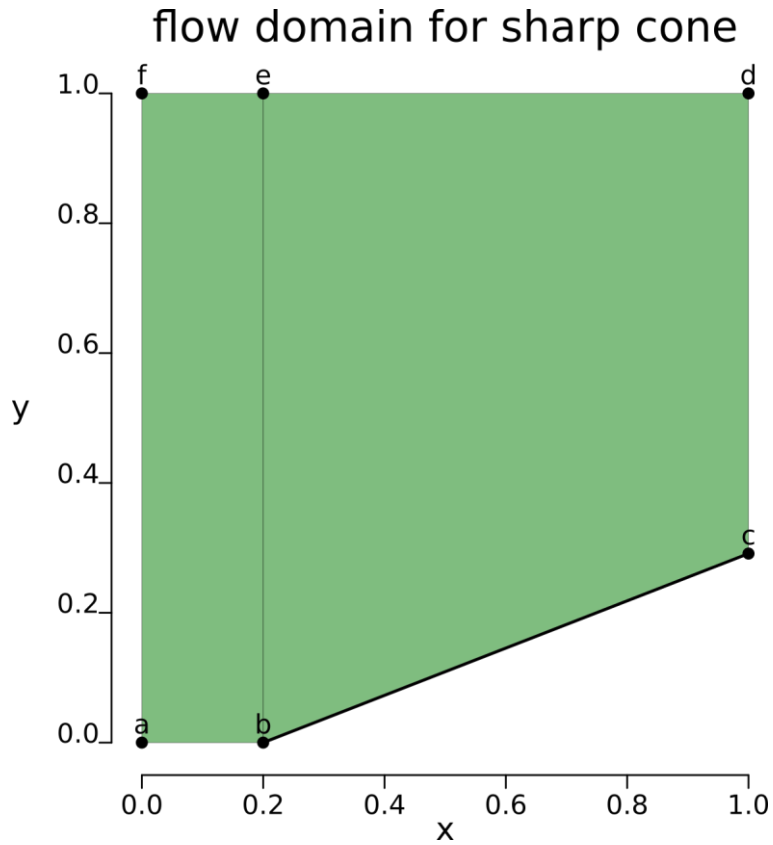


图 3.2：一个 20 度半角尖锥的几何示意图。粗黑线代表圆锥面，绿色区域代表气体流场。增加边界条件，使得气流从左边（西）边界流入，从右边（东）边界流出，南北边界被设置为滑移壁面。这个 SVG 图是在准备时作为草图生成的。

虽然图 3.1 为仿真做足了准备，但在$p_\infty = 95.84\text{kPa}, T_\infty = 1103\text{K}$以及 $V_\infty = 1000\text{m/s}$下，马赫数为 1.5 的自由气流才是有关于原始 ICASE 报告中激波通过坡道的测试问题。读者可对 Maccoll 马赫数值下的仿真运行进行练习，并检查仿真结果是否与阴影图像符合。

3.1.1 运行仿真

假设您已经创建了程序可执行文件，并且可以在系统的搜索路径中访问，如第 2 章所述，为了设置与源代码树副本不同的工作空间，需使用以下命令：

```
$ mkdir ~/temporary-work
$ cd ~/temporary-work
```

```
$ rsync -av ~/dgd/examples/eilmer/2D/sharp-cone-20-degrees/sg/  .
```

$符号表示系统的命令提示符。这样您就可以在这个工作空间里做喜欢做的事，当你完成时再把它移除。rsync 命令应该已经在新构建的工作空间中复制了本示例的必要文件，因此实际上不需要输入下面讨论的文件内容。

开始仿真的第一步是准备一个输入文件，来建立一个简单的空气气体模型。我们可以称这个文件为 ideal-air.inp，它应该有以下两行命令：

```
1  model = "IdealGas"
2  species = {'air'}
```

我们现在使用这个输入文件来准备实际的气体模型定义文件，这需要使用以下命令：

```
$ prep-gas ideal-air.inp ideal-air-gas-model.lua
```

结果将生成一个 Lua 文件 ideal-air-gas-model.lua，这个文件包含理想空气气体模型的详细技术参数。

为了生成气体模型文件，prep-gas 程序使用数据库来收集所需气体的详细热力学特征。气体模型文件比输入文件的内容更详细，但它只是一个文本检查器的 Lua 脚本。当你创建更多仿真时，你可能希望使用这种方式来完整记录特定仿真的气体模型。Eilmer 的热化学模块非常灵活，有许多有效的气体模型可供您使用。

在准备新仿真时，下一件您应该做的事应该是将流动状态和流场的描述构造为 Lua 语言输入脚本，该脚本将被提供给流动求解器的准备阶段。正如所有烹饪节目所展示的那样，接下来我们将使用早先准备并保存的文件 cone20.lua..脚本的详细内容将在下一节中讨论，但现在我们将继续讨论使用预编译的脚本。

仿真计算的第一个阶段是生成一系列网格和初始流态文件，这是通过以下命令行完成：

```
$ e4shared --prep --job=cone20
```

这将在子目录 grid/t0000/中生成一对网格文件，并在子目录 flow/t0000/中产生初始流动状态文件。需要注意的是，每个长格式命令行选项都以两个破折号开始。

现在我们可以使用以下命令开始计算流场的演变[4]：

[4] 选项--max-cpus=2 表示程序允许使用计算机的两个核处理器。现在在工作站上可用的 CPU 内核很少低于两个，所以您应该为能使用它们而感到高兴。

```
$ e4shared --run --job=cone20 --verbosity=1 --max-cpus=2
```

不出意外的话，在一分钟左右的时间内，您将结束计算并得到流动的结果文件，这些文件存储在子目录 flow/中。流场时间演化的计算时长为 5ms（需要 833 个步长），当步长正在计算时，控制台上应当出现一些状态信息。每隔 20 个步长，将输出一条信息，可以显示当前步长数、当前仿真时长、步长大小（dt）、程序开始后的计算耗时（WC）、最终仿真时长预估时间（WCtFT）以及最大允许步长的时间（WCtMS）。所有这些数值的时间单位都是秒。下面就是上述算例中第 540 步骤输出[5]的内容：

```
Step= 540 t= 3.123e-03 dt= 6.003e-06 WC=3 WCtFT=1.7 WCtMS=13.7
```

我们可以这样解释：在第 540 步，针对圆锥体上气体流动，Eilmer 模拟的物理时间为 3.123ms，并且在这一步它使用的时间步长为 6.003μs。在 540 步，程序已经运行了 3 秒并且还将继续运行。我们有两个判断条件来决定程序什么时候结束，无论先达到哪个，程序都停止。这两个判断是：

（1）当它达到要求的 5ms 模拟时间时，程序将在 1.7 秒内结束；

（2）当它达到要求的 3000 个步长时，程序将在 13.7 秒内结束。

在这种情况下，它首先达到最大仿真时间。

仿真计算结束后，您可能希望使用可视化程序查看流场数据。命令如下：

```
$ e4shared --post --job=cone20 --vtk-xml \
--add-vars="mach,pitot,total-p,total-h"
```

该命令将获取流场解的最终框架，并以 XML 格式编写一组 VTK 文件。注意，上面第一行末尾的反斜杠字符代表这一行还没有完成，后面还有更多内容，因此，这两行都是有效的单一逻辑行。--add-vars 选项增加了一些不属于默认解决方案数据的衍生变量。这些衍生变量是马赫数、皮托管压力、总压强和总焓。后处理程序默认选择最后一次快照，但是，您可以使用--tindx-plot 选项选择一个不同的实例来绘图。

可以使用 Paraview 查看 VTK 可视化文件（都在 plot 子目录中），并且可以被调整为颜色鲜艳的图像。尽管您的老板不相信您的分析，但鲜艳的颜色非常具有说服力。

[5] 请注意，数值细节将根据所运行代码的版本和所使用的计算机的性能而变化。

上面讨论的命令用于准备、运行和后处理仿真，它们都收集在 run.sh 命令行脚本中，该脚本可在仿真中任何阶段调用。这是一个尝试无条件执行所用命令的最小命令行脚本示例。理想情况下，您应该交替地运行命令并检查每个命令的结果，或者您应该编写脚本来检查每个命令的返回状态，然后有条件地继续执行每个后续命令行。这比我们现在希望讨论的要复杂得多，但当你在编写批处理系统中使用的命令行脚本时，学习这样做是非常有价值的。

这个特殊的示例足够小，可以方便地在每个阶段交替性的运行，完成这一步后，让我们返回来检查输入脚本的内容。

3.1.2 输入脚本（.Lua）

Lua 输入脚本包含指定仿真的详细信息，我们需要一个气体模型、一些流动条件、一个被定义和网格化的区域以及一些指定的边界条件。还需要决定使用哪些配置选项，例如步长的大小。

```
1 -- cone20.lua
2 -- Simple job-specification file for e4prep -- for use with Eilmer4
3 -- PJ & RG
4 -- 2015-02-24 -- adapted from the Python version of cone20
5
6 -- We can set individual attributes of the global data object.
7 config.title = "Mach 1.5 flow over a 20 degree cone."
8 print(config.title)
9 config.dimensions = 2
10 config.axisymmetric = true
11
12 -- The gas model is defined via a gas-model file.
13 nsp, nmodes, gm = setGasModel('ideal-air-gas-model.lua')
14 print("GasModel set to ideal air. nsp= ", nsp, " nmodes= ", nmodes)
15 initial = FlowState:new{p=5955.0, T=304.0, velx=0.0}
16 inflow = FlowState:new{p=95.84e3, T=1103.0, velx=1000.0}
17
18 -- Demo: Verify Mach number of inflow and compute dynamic pressure.
19 print("inflow=", inflow)
20 print("T=", inflow.T, "density=", inflow.rho, "sound speed= ", inflow.a)
21 print("inflow Mach number=", 1000.0/inflow.a)
22 print("dynamic pressure q=", 1/2 * inflow.rho * 1.0e6)
23
24 -- Set up two quadrilaterals in the (x,y)-plane by first defining
25 -- the corner nodes, then the lines between those corners.
26 a = Vector3:new{x=0.0, y=0.0}
```

```
27 b = Vector3:new{x=0.2, y=0.0}
28 c = Vector3:new{x=1.0, y=0.29118}
29 d = Vector3:new{x=1.0, y=1.0}
30 e = Vector3:new{x=0.2, y=1.0}
31 f = Vector3:new{x=0.0, y=1.0}
32 ab = Line:new{p0=a, p1=b} -- lower boundary, axis
33 bc = Line:new{p0=b, p1=c} -- lower boundary, cone surface
34 fe = Line:new{p0=f, p1=e}; ed = Line:new{p0=e, p1=d} -- upper boundary
35 af = Line:new{p0=a, p1=f} -- vertical line, inflow
36 be = Line:new{p0=b, p1=e} -- vertical line, between quads
37 cd = Line:new{p0=c, p1=d} -- vertical line, outflow
38 quad0 = makePatch{north=fe, east=be, south=ab, west=af}
39 quad1 = makePatch{north=ed, east=cd, south=bc, west=be, gridType="ao"}
40 -- Mesh the patches, with particular discretisation.
41 nx0 = 10; nx1 = 30; ny = 40
42 grid0 = StructuredGrid:new{psurface=quad0, niv=nx0+1, njv=ny+1}
43 grid1 = StructuredGrid:new{psurface=quad1, niv=nx1+1, njv=ny+1}
44 -- Define the flow-solution blocks.
45 blk0 = FluidBlock:new{grid=grid0, initialState=inflow}
46 blk1 = FluidBlock:new{grid=grid1, initialState=initial}
47 -- Set boundary conditions.
48 identifyBlockConnections()
49 blk0.bcList[west] = InFlowBC_Supersonic:new{flowState=inflow}
50 blk1.bcList[east] = OutFlowBC_Simple:new{}
51
52 -- add history point 1/3 along length of cone surface
53 setHistoryPoint{x=2 * b.x/3+c.x/3, y=2 * b.y/3+c.y/3}
54 -- add history point 2/3 along length of cone surface
55 setHistoryPoint{ib=1, i=math.floor(2 * nx1/3), j=0}
56
57 -- Do a little more setting of global data.
58 config.max_time = 5.0e-3 -- seconds
59 config.max_step = 3000
60 config.dt_init = 1.0e-6
61 config.cfl_value = 0.5
62 config.dt_plot = 1.5e-3
63 config.dt_history = 10.0e-5
64
65 dofile("sketch-domain.lua")
```

对于输入脚本，需要注意第一件事是，它是大型 Lua 程序的一部分，并且 Lua 程序的解释器嵌入在 e4shared 主仿真程序中。一旦出现--prep 标记，e4shared 程序就会将大量的 Lua 接口加载到程序的 D 语言部分，然后调用 Lua 解释器来完

成 Lua 程序的第一部分。这种操作会设置大量的服务，这些服务用于提供给您的输入脚本，然后被嵌入式 Lua 解释器处理。

Lua 中的单行注释从双破折号开始，并持续到行尾。我们已经使用了这些注释来开始输入脚本，来提醒我们预期的仿真以及谁为编写文件负责。第一个真正的命令在第 7 行，我们将仿真标题设置为字符串。仿真的全局配置包含在全局配置类中，这个类作为 config 表格出现在 Lua 脚本中。在这个表格后面是 D 语言全局配置类，它将数据储存在 D 语言域中。访问 config 中的条目，调用 D 语言域中访问相应属性的函数。

注意，您有权限访问 Lua 解释器的全部功能。在第 8 行中，我们使用了 Lua 函数 print 来显示 config.title 的值。在第 9 和第 10 行，我们继续设置了更多的配置选项，布尔值可以指定为真或假。

气体模型是仿真计算的核心。在第 13 行中，我们告诉程序在哪个位置可以找到气体模型的详细信息。该信息是另一个 Lua 脚本，它为热力学和气体的运输属性设置了相关的参数表。回想一下，在第 9 页中，我们用 prep-gas 程序创建了 ideal-air-gas-model.lua 文件。一个关于气体模型以及可编程界面的指南报告[5]提供了更详细的信息。在配置气体模型时，setGasModel 函数返回了三个值：种类数目、非平衡态热能模型的数目和气体模型对象的参考值。在输入脚本中进行计算时，这些项目具有非常便捷的优势。

一旦设置了气体模型，您就可以创建 FlowState 对象。在第 15 和 16 行，我们使用 Lua 书[12]中描述编程的协定，构建了两个这样的对象。注意在单词 new 前使用冒号而不是点，同时在构建输入 new 函数的参数表时，注意大括号的使用。在构建输入脚本中的对象时，我们选择了一种将单个表中所有元素都作为命名属性的表示法。对于这些特定的 FlowState 构造函数，我们已经省略了一些采用默认值的变量，例如 vely 和 velz。这些变量的默认值为 0。如果一个强制项丢失了，new 调用的函数就会报错，具体告诉您需要做什么。

对于 Eilmer 代码中的气体动力学部分，我们仍然尝试使用 SI-MKS 单元。点坐标以米表示，时间以秒表示，速度以 m/s 表示，压力以 Pa 表示，温度以热力学标度 K 表示。

为了展示在输入脚本中任意计算的使用，第 19-22 行向我们展示了如何查看

FlowState 表中的内流、打印出的一些描述流动状态的参数值，以及使用这些变量来计算马赫数和动压的导出量。Eilmer 本质上是一个多种类、多温度范围的流体代码，但现在理想空气气体的特定气体模型只有一种，并且没有非平衡态的热模型。在前面设置流动状态时，FlowState 构造函数就意识到了这个特定气体模型的单一种类规范，并将从提供参数和默认参数中生成一个合适的表格。在第 20 行中要访问静态温度值以便显示它，我们可以要求将 T 元素作为 inflow.T（如上所示）或者 inflow["T"]。第 21 行和 22 行在计算来流马赫数$\frac{v_\infty}{a}$，动压$\frac{1}{2}\rho V_\infty^2$时，用了气体状态参数和已知的来流速度。在准备阶段运行时，打印结果将显示在标准输出上。

有了气体模型和一组流动状态后，让我们把注意力转移到流场的构建上。图 3.1 和图 3.2 所示的区域几何复杂度很小，所以我们将使用 Eilmer 内置的几何和网格函数。

26 行到 31 行通过使用 Vector3 构建函数定义几个感兴趣的点，来开始几何定义。注意，在传递给 Vector3 种类中 new 函数的表中，坐标值以已命名项目的方式提供，然后未指定的 z 组件默认为 0。如图 3.2 所示，这些点将成为四边形的面，并在这个四边形上面定义网格和块，这样便于将它们匹配简单的名称。b 点在圆锥的尖端上，位于 x=0.2m 处；c 点位于 x=1.0m 处，在 x-y 平面的圆锥基点标志处。我们手动计算了 c 点 y 坐标值为$0.8 \times \tan 20°$；点 d、e、f 用于定义流域的上（北）边界，同时我们设定它们的 y 坐标值为 1.0m。

一旦有了流域的角点后，我们继续在 32-37 行脚本中构建一些线段。线上的第一个点定义为 p0，最后一个点定义为 p1，这些分别对应于参数值 t=0.0 和 t=1.0。Line 类是 Path 基类的衍生类。之后，你将会定义更为复杂的 Path 对象，例如弧线、贝塞尔曲线以及样条曲线，每个这样的 Path 对象都定义为相同的参数范围：$0.0 \leq t \leq 1.0$。在已构建的行中选择的变量名称只是为了反映每个段的端点。几乎任何名字都是够用的[6]，但随着脚本数目的增加，小心的使用命名会减少混淆。

第 38 行和 39 行使用 makePatch 函数收集线段，作为二维平面的边缘。分配给 quad0 的面将使用默认的超限插值，而 quad1 的面将利用 Knupp 鲁棒性的椭圆网格生成器[13]定义的网格边界，来插值到物理空间中的点。

[6] Lua 环境下的全局变量符号可以在附录 B.7 中查看。您应当避免将这些作为对象的名称。

流动模拟要求将流域指定为有限体积的单元网格。到目前为止，我们已经将这个域描述为空间的面。在第 41 行，我们定义了三个变量，它们表示我们想要沿着网格方向的单元格数量，同时，在第 42 行和第 43 行，我们为每个四边形面构建一个单元格网格。注意，每个网格方向上的顶点数被提供给网格生成器，它们比我们想要的单元格数多。

流动的解是在单元块上定义的。在第 45 行和第 46 行，我们通过将这些块与网格和初始流状态相关联来构造它们。在这里，我们在每个块上使用统一的流动状态，但也可以指定一个变化的流动状态。稍后的示例将向您展示如何通过使用在您定义的输入脚本中的 Lua 函数来实现这一点。现在，我们将圆锥头上游的初始流态设置为与流入流态相同，因为我们预计在这个块内不会发生任何有趣的事情。在圆锥面附近的块体中，我们将初始流动条件设置为相当低压力、静止的气体。上游的流体将冲击这个块体，并在圆锥体上形成一个流动边界层。

为了完成流场的定义，我们需要指定驱动流动解的边界条件。这里使用的二维结构网格块的边界标记为北、东、南和西。如果我们不为每个特定的块边界指定一个特定的边界条件，那它就被默认指定为边界无滑动条件(WallBC WithSlip)。我们有一个选项是在调用 FluidBlock 构造函数、提供如表中的边界条件来指定边界条件。本例中使用了一种不同的方法。首先在第 48 行，我们请求程序自动识别连接块，这是通过用暴力搜索来匹配角点完成的。如果网格在该空间中，它的顶点恰好与一对边界重合，就像现在我们给出的例子：block 0 的东边和 block 1 的西边一样，它们是重合的，那么我们使用 ExchangeBC_FullFace 这个边界条件定义与它相对应的一对边界。这有效地将流域沿着共同边界组合在一起。第 49 行和第 50 行分别为流入和流出指定了单独的边界条件对象。

有时，我们对特定点处的流动历史数据很感兴趣，并且希望获得比在整个流场记录的快照更多的历史细节。我们可以为这个历史记录识别特定的单元格，如第 53 和 55 行所示。第 53 行通过一个空间位置指定了特定的单元格，可以识别这个空间位置最近包含的单元格。第 55 行直接指定块以及另一个单元格的 i 和 j 大小(在该块内)。

最后的准备工作是在第 58 到 63 行中设置更多的配置参数。当仿真时间超过最大时间(max time)或步长达到最大步长(max step)时，无论哪个先发生，仿真计

算都将终止。通常，您希望您的仿真运行到特定的时间，但是，有许多合适的时间条件可以限制步骤的数量。在尝试设置新仿真时，将最大步骤数限制为一个相当小的值可以节省大量等待时间。

在第 60 行，我们指定了希望仿真开始的初始时间步。然后，代码接管并将时间步调整到某个允许值，该值由第 61 行指定的 cfl_value 值引导。每隔几个时间步，代码将扫描所有单元格，寻找信号通过任意一个单元格的最短时间。对于对流运动项，这是压力波穿过单元格的时间。然后将找到的最短时间乘以 cfl_value 值，并将结果用作整个仿真的时间步长。

在每一次 dt_plot 期间，这个仿真程序都会记录整个流场的快照。这段时间被指定为仿真时间中的秒数，而不是您在计算过程中看到的墙钟时间(wall-clock)。对于少数选定点(上面讨论了三段)来讲，历史数据通常具有较短的存在周期，这个周期就是 dt_history。

在这个输入脚本中，我们要做的最后一件事是调用另一个 Lua 脚本，来生成如图 3.2 所示的 SVG 草图。稍后我们将展示这个草图脚本的内容。

3.2 结果与后处理

图 3.3 显示了流动开始 5 毫秒后的流场。这个时间已经足够让流场充分发展，并达到激波基本上是直线的稳流状态。

通过打开 plot/cone20.pvd 文件，图像可以在 Paraview 软件中生成。左上角添加的时间戳为一个 Annotate Time Filter，该过滤器从 Filters 主过滤器菜单中选择。此外，压力场被绘制成彩色图，温度场也被绘制成有边缘的表面，足以清晰地显示计算网格。右边区域的网格发生了畸变，这是由于区域正交性(AO)网格生成器做出了妥协，从而在块的边缘实现了一个合理的正交网格。默认的超限网格生成器会生成一个整体较的小畸变网格，但是在这个特定区域中会有更多剪切的单个单元格。在左边的矩形区域中，两个生成器将产生相同的网格。

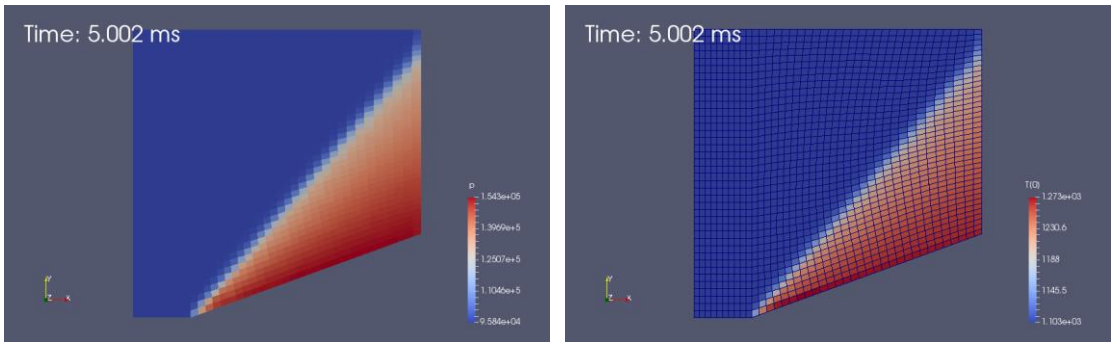


图 3.3:在 20 度半角圆锥上，低分辨率模拟流体流动的压力场和温度场。其中温度场图也包括网格。

压力场中显示的激波，展示了由“激波捕捉”代码(如 Eilmer)生成的流场解的特征。由于采用了粗糙的网格结构，激波呈现出阶梯式的外形，这是通过绘图程序来渲染的，它被设置为在每个单元格中[7]，以统一的颜色显示单元格平均值。而且，当沿着一条穿过激波的线前进，达到完全的压力跃变之前，会有一小部分单元格已传递压力。在理想的无粘仿真中，激波应该在厚度方向上没有过渡。这可以通过增加网格分辨率来实现，如图 3.4 所示。高分辨率的解决方案看起来很简洁，但计算成本(以计算时间为参考)却增加了，从几秒钟增加到到一个多小时。

由于 Eilmer 是一个仿真程序，它从整个仿真域的一些初始流态(但可能是可变的)开始，然后根据应用的边界条件，以精确到时间的方式（time-accurate）将守恒方程向前积分。在这种情况下，一个恒定流量的流体经过一个尖锐的圆锥体时，流场演变成稳流状态。图 3.5 为多次仿真后的压力场。以秒为单位的时间增量，在框架中被指定为如下的输入脚本：config.dt_plot = 1.5e-3。

虽然在图 3.5 中不是很明显，但是在 1.5 毫秒帧数之前，已经有很多详细的流动结构通过了流域。从那时开始到最后的 5.0 毫秒，似乎并没有发生很多事情。只用 3.0 毫秒就结束仿真是很诱人的，但这取决于您需要的精度，您可能需要运行很长的时间，才能实现一个足够稳定且高精度的流动。

[7] 如果你想要一个平滑的外观，你可以使用 Paraview 的 Cell Data to Point Data 功能

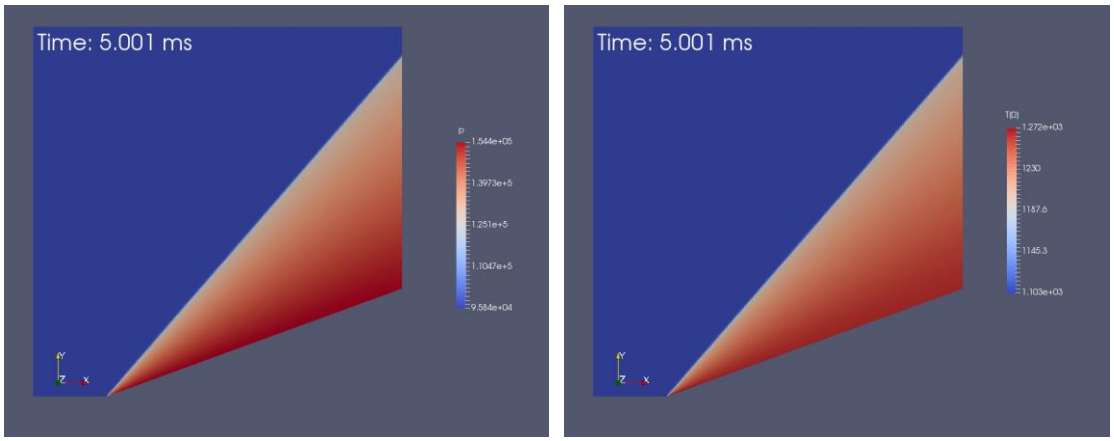


图 3.4:每个方向有 8 倍分辨率网格的压力场和温度场。

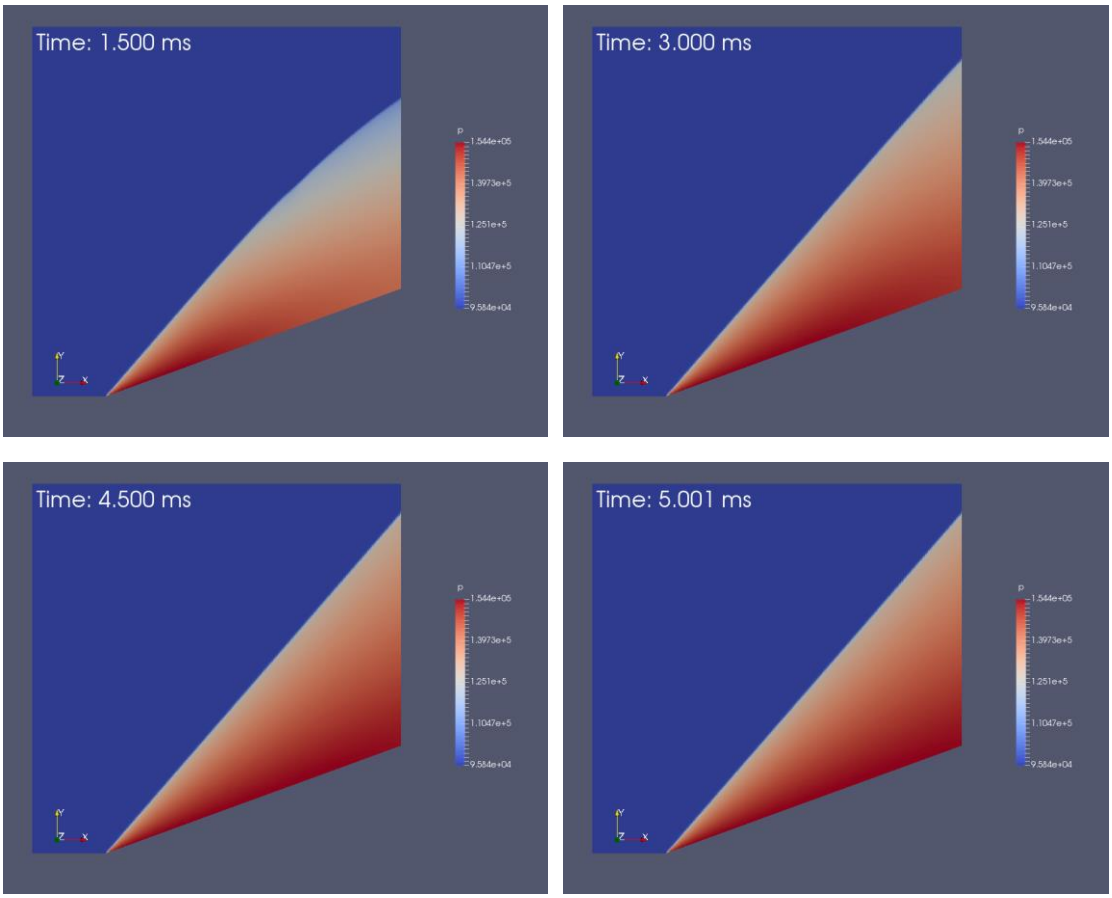


图 3.5:压力场演变，时间如图所示。

一个重要的流动参数可能是圆锥面上的压力，通过设置历史点，我们已经安排 Eilmer 偶尔写出圆锥面上几个单元格的流动特性。然后，我们使用 Awk 程序 (cp. awk) 编写一个简单的数据文件，来过滤历史文件、提取时间(列 1)和静压力 (列 10)，文件包含在第 1 列中以毫秒为单位的时间以及第 2 列中的压力系数。新用户可能希望学习 Awk 语言，因为用它编写过滤程序非常方便。附录 C 给出了

Awk 语言的简要介绍。

```
# cp.awk
# Scan a history file, picking out pressure and scaling it
# to compute coefficient of pressure.
#
# PJ, 2016-09-22
#
BEGIN {
    Rgas = 287.1; # J/kg.K
    p_inf = 95.84e3; # Pa
    T_inf = 1103; # K
    rho_inf = p_inf / (Rgas*T_inf)
    V_inf = 1000.0; # m/s
    q_inf = 0.5*rho_inf*V_inf*V_inf
    print "# rho_inf=", rho_inf, " q_inf=", q_inf
    print "# t,ms cp"
}
$1 != "#" {
    t = $1; p = $10
    print t * 1000.0, (p - p_inf)/q_inf
}
END {}
```

由 NACA 报告 1135[14]的图 5 可知，预期稳态激波角为 49°，由图 6 可知，压力系数为:

$$\frac{p_{cone-surface} - p_{\infty}}{q_{\infty}} \approx 0.387$$

指定自由流动的动压为 $q_{\infty} = \frac{1}{2}\rho_{\infty}u_{\infty}^2 \approx 151.38kPa$。图 3.6 显示了从前锥面三分之二处的历史点到锥底的压力系数。注意当自由流动驱动的激波结构到达圆锥面上的这一历史点时，激波会突然上升。在这个初始值上升之后，随着锥形流动区域的填充和趋于稳定，将出现一个更缓慢的上升。您现在可以知道前面选择 5.0 毫秒作为仿真结束时间的目的了。

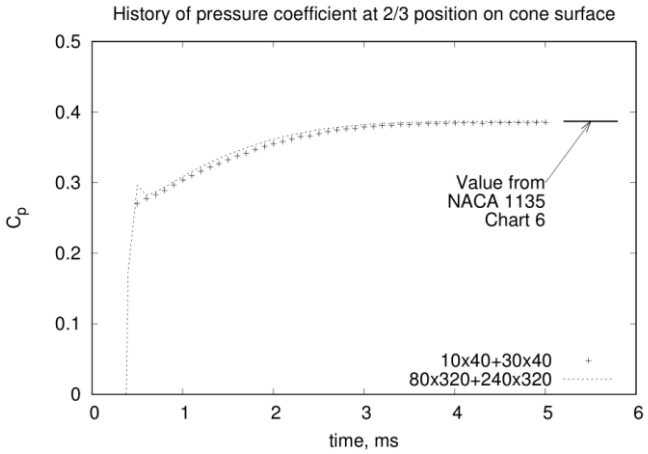


图 3.6:两种网格分辨率下，在 20 度半角锥上流动时锥面压力系数的变化。

生成图 3.6 的命令如下：

```
#!/bin/bash
# plot.sh
# Compute coefficient of pressure for the history point
# and plot it against the previously computed high-res data.
#
# PJ, 2016-09-22
#
awk -f cp.awk hist/cone20-blk-1-cell-20.dat > cone20_cp.dat
gnuplot plot_cp.gnuplot
```

GnuPlot 的命令如下：

```
set term postscript eps enhanced 20
set output "cone20_cp.eps"
set style line 1 linetype 1 linewidth 3.0
set title "History of pressure coefficient at 2/3 position on cone surface"
set xlabel "time, ms"
set ylabel "C_p"
set xtic 1.0
set ytic 0.1
set yrange [0:0.5]
set key bottom right
set arrow from 5.2,0.387 to 5.8,0.387 nohead linestyle 1
set label "Value from\nNACA 1135\nChart 6" at 5.0,0.3 right
set arrow from 5.0,0.3 to 5.5,0.387 head
plot "cone20_cp.dat" using 1:2 title "10x40+30x40", \
"cone20_cp_hi-res.dat" using 1:2 title "80x320+240x320" with lines
```

3.3 获取数据字段进行专业处理

除了 e4shared 后处理模式提供的通用切片式的后处理之外，通过提供可进行高精度计算的自定义后处理脚本(在 Lua 中)，做关于流动数据的专门计算可能是很有用的。这个脚本可以完全访问 e4shared 模式选择的流动的解，然后成为一个可以完全访问 Lua 解释器功能的 Lua 脚本。

激波在这个流动中本应该是直的，利用主程序中内置的气体动力学函数，我们可以计算出，在相对于自由气流方向的激波角度应该是 β = 48.96°。内置的函数如下所示。

```
1 -- ideal_shock_angle.lua
2 -- Invoke with the command line:
3 -- $ e4shared --custom-post --script-file=ideal_shock_angle.lua
4 V1=1000.0; p1=95.84e3; T1=1103.0; theta=math.rad(20.0)
5 beta = idealgasflow.beta_cone(V1, p1, T1, theta)
6 print("beta=", math.deg(beta), "degrees")
```

虽然我们可以将输入值指定为 beta_cone 函数调用的文字，但在某种程度上，对变量的赋值以及在函数调用中使用这些名称会使脚本变得自我文档化。idealgasflow 表中有效的完整函数集列在附录 D.1 中。

estimate_shock_angle.lua 脚本（如下所示）使用了 FlowSolution 类（脚本中的第 8 行）提供的数据读取和储存功能，这个类在 e4shared 的自定义后处理模式中是可用的。get_cell_data 方法(第 37 行)提供访问特定单元格的流动数据功能，该方法以表的形式返回数据。注意，冒号用于访问绑定到 fsol 的 FlowState 对象。

```
1 -- estimate_shock_angle.lua
2 -- Invoke with the command line:
3 -- $ e4shared --custom-post --script-file=estimate_shock_angle.lua
4 -- PJ, 2015-10-20
5 --
6 print("Begin estimate_shock_angle")
7 nb = 2
8 fsol = FlowSolution:new{jobName="cone20", dir=".", tindx=4, nBlocks=nb}
9 print("fsol=", fsol)
10
11 function locate_shock_along_strip()
12     local p_max = ps[1]
13     for i = 2, #ps do
14         p_max = math.max(ps[i], p_max)
```

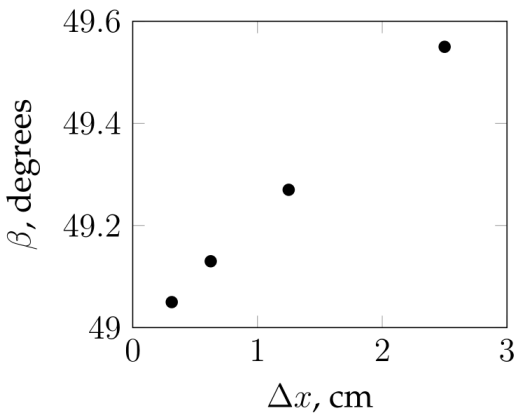
```
15    end
16    local p_trigger = ps[1] + 0.3*(p_max - ps[1])
17    local x_old = xs[1]; local y_old = ys[1]; local p_old = ps[1]
18    local x_new = x_old; local y_new = y_old; local p_new = p_old
19    for i = 2, #ps do
20       x_new = xs[i]; y_new = ys[i]; p_new = ps[i]
21       if p_new > p_trigger then break end
22       x_old = x_new; y_old = y_new; p_old = p_new
23    end
24    local frac = (p_trigger - p_old) / (p_new - p_old)
25    x_loc = x_old*(1.0 - frac) + x_ne*frac
26    y_loc = y_old*(1.0 - frac) + y_new*frac
27    return
28 end
29
30 xshock = {}; yshock = {}
31 local nj = fsol:get_njc(0)
32 for j = 0, nj-1 do
33    xs = {}; ys = {}; ps = {}
34    for ib = 0, nb-1 do
35       local ni = fsol:get_nic(ib)
36       for i = 0, ni-1 do
37          cellData = fsol:get_cell_data{ib=ib, i=i, j=j}
38          xs[#xs+1] = cellData["pos.x"]
39          ys[#ys+1] = cellData["pos.y"]
40          ps[#ps+1] = cellData["p"]
41       end
42    end
43    locate_shock_along_strip()
44    if x_loc < 0.9 then
45       -- Keep only the good part of the shock.
46       xshock[#xshock+1] = x_loc
47       yshock[#yshock+1] = y_loc
48    end
49 end
50
51 -- Least-squares fit of a straight line for the shock
52 -- Model is y = alpha0 + alpha1*x
53 sum_x = 0.0; sum_y = 0.0; sum_x2 = 0.0; sum_xy = 0.0
54 for j = 1, #xshock do
55    sum_x = sum_x + xshock[j]
56    sum_x2 = sum_x2 + xshock[j] * xshock[j]
57    sum_y = sum_y + yshock[j]
58    sum_xy = sum_xy + xshock[j] * yshock[j]
```

```
59 end
60 N = #xshock
61 alpha1 = (sum_xy/N - sum_x/N*sum_y/N) / (sum_x2/N - sum_x/N*sum_x/N)
62 alpha0 = sum_y/N - alpha1*sum_x/N
63 shock_angle = math.atan(alpha1)
64 sum_y_error = 0.0
65 for j = 1, N do
66    sum_y_error = sum_y_error+math.abs((alpha0+alpha1 * xshock[j])-yshock[j])
67 end
68 print("shock_angle_deg=", shock_angle * 180.0/math.pi)
69 print("average_deviation_metres=", sum_y_error/N)
```

在沿着结构化块的 i 指针方向的一长串单元格中，函数 locate_shock_along_strip(第 11-28 行)搜索着值得注意的压力跃变。第 33-42 行在特定的 j 指针处设置了数据带。一旦将跃变位置的坐标存储在表 xshock 和 yshock 中，那么在第 53-62 行中就执行线性模型的最小二乘法。第 3 行的注释中显示了调用脚本的命令。在这样的注释或命令行脚本中，记录命令行来执行仿真和联合后处理活动通常是一个好主意。需要提醒您这些细节，这样就可以使这些标记具有可读性或可执行性。

3.4 网格收敛性

确定某个参数的单个值只是整个工作的一部分。通常，你必须提供一些关于该值可靠性的指导，这通常是通过网格收敛性的研究来完成的。对于激波角的估计，我们可以遵循初始仿真运行：在连续更细的网格上运行若干次，并检查估计数在单元格尺寸趋近于零处的收敛性。



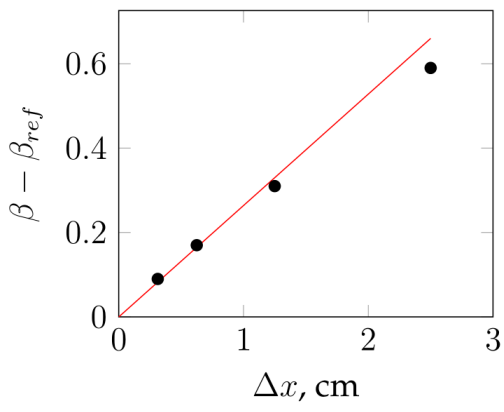


图 3.7：网格改进后激波角和残差的收敛性。其中$\beta_{ref} = 48.96°$

由于此示例对低分辨率网格的要求不是很高，因此很容易将网格分辨率提高一倍，以及很容易在适当的单元格大小范围内获取数据。图 3.7 显示了原始激波角估计值很好地在 49°左右收敛。一般来说，这通常是我们分析的终点。因为我们通过 Taylor-Maccoll 理论计算出一个参考值，所以我们也可以看看收敛到真实值的情况，如果有足够的计算资源，这个值会无限接近我们想要的值。

3.5 第一个例子的其它注意事项

•这个简单模拟的运行时间大约是 8 秒，对于配备英特尔 i5-4300U 核心处理器的 Surface Pro 3 上需要运行 853 步。由于没有很好地平衡这些块，所以没有充分利用两个核处理器。想要充分利用多核工作站，可以尝试使用数量相当的单元块数来安排块的工作空间。

•这个 cone20.lua 输入脚本实际上可以完全访问仿真程序中内置的 Lua 解释器。请注意这一点。

•Lua 是一种动态语言，它很容易将名称绑定到脚本中的新对象。注意不要重新绑定一些重要的名称，因为这些名称稍后在处理配置时会被程序使用。这种情况可能以一种不太明显的方式出现在外部模块或程序包中(用来在脚本中做一些有趣的事情)。另外，如果没有明确地将名称声明为当地名称，则在第一次赋值时将参考全局变量。有关在开始处理输入脚本时定义的全局符号列表，请参见附录 B.7。

3.6 使用 Lua 语言进行参数化建模

让我们重新进行模拟，以进一步探索气体动力学，并利用 Lua 输入脚本的参数化功能。首先通过用变量和简单的代数表达式替换原始脚本的一些常数值，从而将流场的描述和流域的几何描述参数化。

具体来说，我们引入一个变量 M，表示流入气流的马赫数，然后根据这个值以及流入气流的估计声速来计算速度。这为我们提供了一个方便的方法来指定样

本的马赫数，从而可以探索模拟流场对一定范围内马赫数的响应情况。我们还将描述圆锥的半角和轴向长度，从这些项中，我们可以计算出基半径。对于定义流域的其余关键项，我们需要知道圆锥顶点相对于流入边界的位置，并且需要说明流域的上边缘距离轴有多远。最后，为了在更改流域边界时更方便地生成网格，我们将单元格的长度定义为 dx，并将每个块中的单元格数量，定义为块的每个维度的总体长度除以这个单元格的长度。

3.6.1 输入脚本(.lua)

```
-- conep.lua
-- Parametric setup for sharp-cone simulation.
-- PJ & RG
-- 2016-09-23 -- adapted from cone20.lua

-- We can set individual attributes of the global data object.
config.dimensions = 2
config.axisymmetric = true

-- The gas model is defined via a gas-model file.
nsp, nmodes, gm = setGasModel('ideal-air-gas-model.lua')
print("GasModel set to ideal air. nsp= ", nsp, " nmodes= ", nmodes)
initial = FlowState:new{p=5955.0, T=304.0, velx=0.0}
-- Compute inflow from Mach number.
inflow_gas = FlowState:new{p=95.84e3, T=1103.0}
M = 1.5
Vx = M*inflow_gas.a
print("inflow velocity Vx=", Vx)
print("dynamic pressure q=", 1/2 * inflow_gas.rho * Vx * Vx)
inflow = FlowState:new{p=95.84e3, T=1103.0, velx=Vx}
print("T=", inflow.T, "density=", inflow.rho, "sound speed= ", inflow.a)

-- Parameters defining cone and flow domain.
theta = 20 -- cone half-angle, degrees
L = 0.8 -- axial length of cone, metres
rbase = L*math.tan(math.pi * theta/180.0)
x0 = 0.2 -- upstream distance to cone tip
H = 1.0 -- height of flow domain, metres
config.title = string.format("Mach %.1f flow over a %.1f-degree cone.",
                             M, theta)
print(config.title)

-- Set up two quadrilaterals in the (x,y)-plane by first defining
-- the corner nodes, then the lines between those corners.
```

```
35 a = Vector3:new{x=0.0, y=0.0}
36 b = Vector3:new{x=x0, y=0.0}
37 c = Vector3:new{x=x0+L, y=rbase}
38 d = Vector3:new{x=x0+L, y=H}
39 e = Vector3:new{x=x0, y=H}
40 f = Vector3:new{x=0.0, y=H}
41 ab = Line:new{p0=a, p1=b} -- lower boundary, axis
42 bc = Line:new{p0=b, p1=c} -- lower boundary, cone surface
43 fe = Line:new{p0=f, p1=e}; ed = Line:new{p0=e, p1=d} -- upper boundary
44 af = Line:new{p0=a, p1=f} -- vertical line, inflow
45 be = Line:new{p0=b, p1=e} -- vertical line, between quads
46 cd = Line:new{p0=c, p1=d} -- vertical line, outflow
47 quad0 = makePatch{north=fe, east=be, south=ab, west=af}
48 quad1 = makePatch{north=ed, east=cd, south=bc, west=be, gridType="ao"}
49 -- Mesh the patches, with particular discretisation.
50 dx = 1.0/40
51 nx0 = math.floor(x0/dx); nx1 = math.floor(L/dx); ny = math.floor(H/dx)
52 grid0 = StructuredGrid:new{psurface=quad0, niv=nx0+1, njv=ny+1}
53 grid1 = StructuredGrid:new{psurface=quad1, niv=nx1+1, njv=ny+1}
54 -- Define the flow-solution blocks.
55 blk0 = FluidBlock:new{grid=grid0, initialState=inflow}
56 blk1 = FluidBlock:new{grid=grid1, initialState=initial}
57 -- Set boundary conditions.
58 identifyBlockConnections()
59 blk0.bcList[west] = InFlowBC_Supersonic:new{flowState=inflow}
60 blk1.bcList[east] = OutFlowBC_Simple:new{}
61
62 -- add history point 1/3 along length of cone surface
63 setHistoryPoint{x=2 * b.x/3+c.x/3, y=2 * b.y/3+c.y/3}
64 -- add history point 2/3 along length of cone surface
65 setHistoryPoint{ib=1, i=math.floor(2 * nx1/3), j=0}
66
67 -- Do a little more setting of global data.
68 config.max_time = 5.0e-3 -- seconds
69 config.max_step = 3000
70 config.dt_init = 1.0e-6
71 config.cfl_value = 0.5
72 config.dt_plot = 1.5e-3
73 config.dt_history = 10.0e-5
74
75 dofile("sketch-domain.lua")
```

3.7 气体动力学的探索

首先，我们将重复前面对 20°半角圆锥的仿真，但这次是从一个参数化脚本中构建的。其次，我们将探索在 32 度半角圆锥面的情况下，圆锥面上的气流会发生什么。本节的重点是第二种情况。

重复我们对 20 度半角圆锥的仿真，图 3.8 显示了流体开始后 5 毫秒内大致与图 3.3 相同的流场。它具有相同的直线、附体激波和相同的所示压力范围。

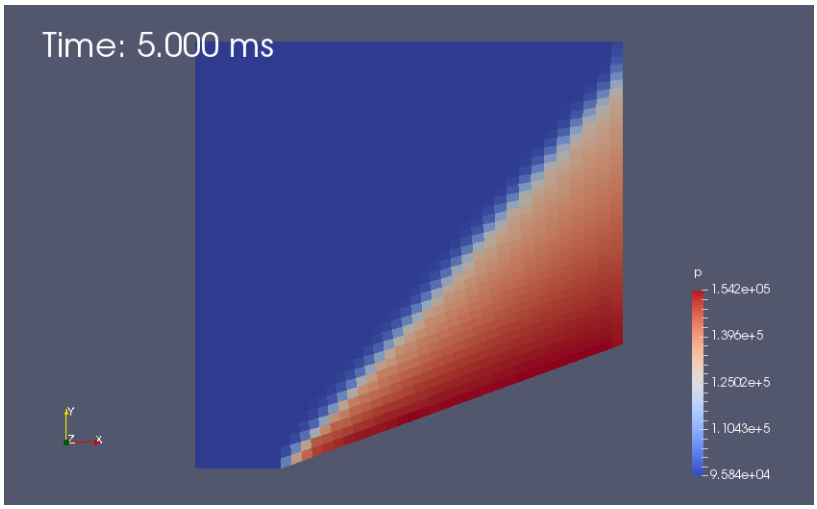


图 3.8 流过 20°半角圆锥体的 1.5 倍马赫数气流，在低分辨率仿真时的压力场。这是参数设置，但产生的仿真结果与原始设置相同。

查阅 NACA-1135[14]的圆锥激波图，我们可以看到 32°的圆锥落在激波极圈之外，它的自由流动马赫数为 1.5，所以应该有一个脱体激波。我们试着把 theta 的值从 20 换成 32。这是在重新运行准备程序和主仿真程序之前需要做的所有工作，并通过计算获得合适的速度值，该值已经编码在用户输入脚本中。图 3.9 显示了 5ms 时的压力场结果。

结果与预期的不太一样，因为在默认的 WallBC_WithSlip 边界条件下，流体在圆锥面和流域的上边缘之间出现了阻塞，这逐渐演变为光滑管道的光滑内壁。最明显的解决方法是将 H 设置为更大的值，从而增加流域的高度。图 3.10 所示为在 5 ms 时，进气马赫数为 1.5(应该有一个脱体激波)和自由流动马赫数为 1.6(应该有一个附体激波)情况下，根据无粘流理论得到的压力场。

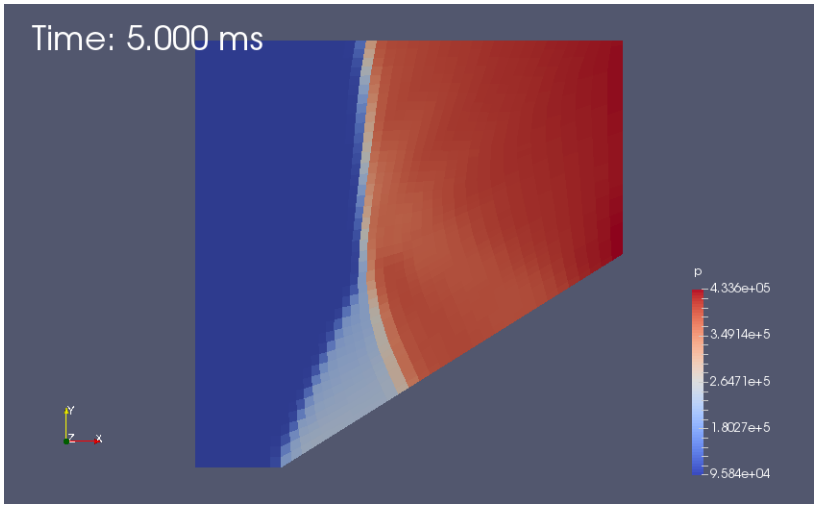
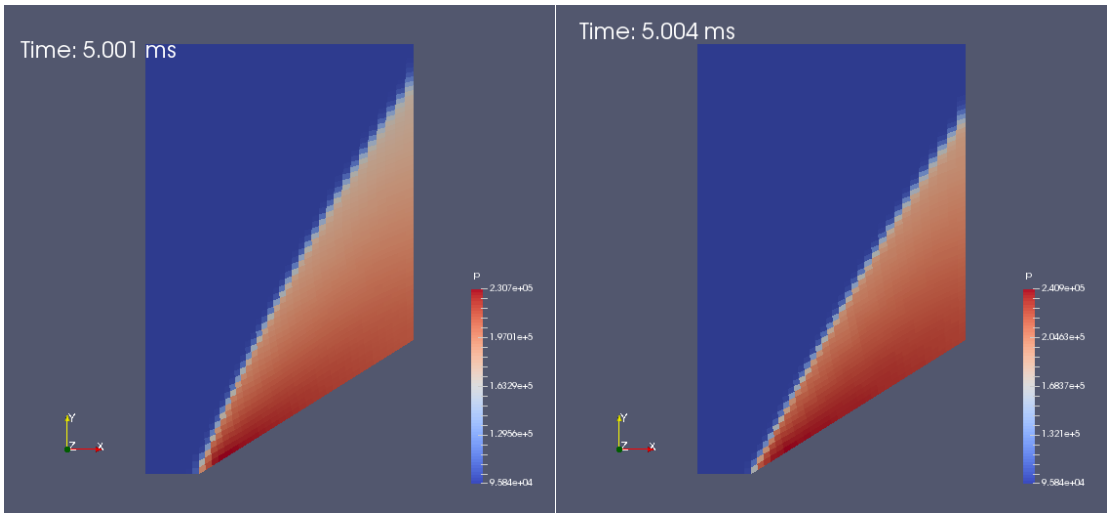


图 3.9: 流过 20°半角圆锥体的 1.5 倍马赫数气流，在低分辨率仿真 5ms 时的压力场。



(a)来流马赫数为 1.5 (b)来流马赫数为 1.6

图 3.10:在 H = 1.6 的大流域内，32°半角圆锥上，低分辨率仿真 5ms 时的压力场

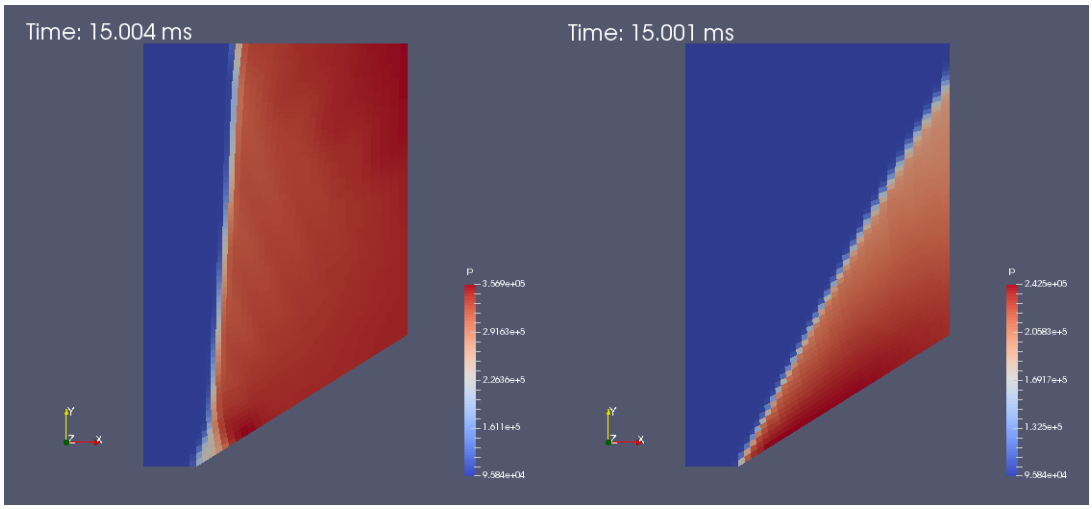
现在结果看起来更好了，每个仿真中的激波看起来都很整齐。在圆锥的顶端，1.6 倍马赫数的气流有一个更直的激波和一个更干净的开始，这样看起来它像是附在这个相当低分辨率的仿真中。在这一点上，我们可能会忍不住宣布胜利并前往最方便的酒吧，然后再研究高质量的多块网格。但是，我们想认真学习 CFD 仿真技术，需要通过更长的时间来运行仿真来确认流动是否真的达到了稳态。此外，模拟是在不到一分钟时间内完成的，所以需要多少额外的努力呢?

大约 5 分钟后，您将看到图 3.11 所示的结果，并且您也希望早一点去酒吧。1.6 倍马赫数的激波看起来不错，而且应该更直一点，但 1.5 倍马赫数的激波并没有达到预期的效果。为什么在流入参数值存在较小的区别下，会有如此大的差

异呢？以及为什么这种差异看起来是来自下游?

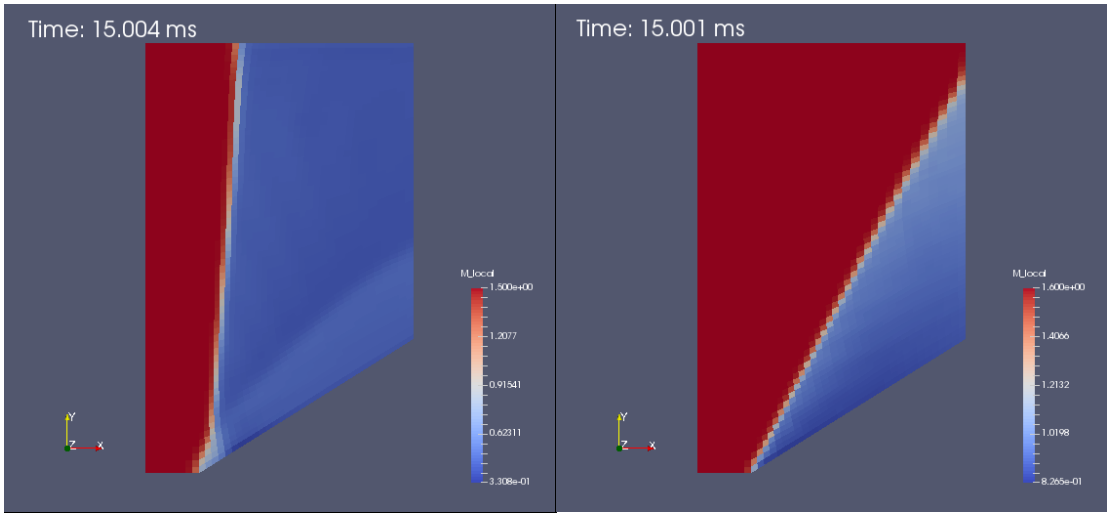
如果您问导师这个问题，您可能会被问到:“马赫数是什么样子的，尤其是在流出边界上?”在准备图像文件时(如第 10 页中 run.sh 脚本的第 7 行所示)，确保在您希望添加到流动解决方案中的变量列表中包含马赫数，然后生成如图 3.12 所示的图。

在接近 1.6 倍来流马赫数下，流场马赫数在出口面上是跨声速的，但在 1.5 倍马赫数来流下，对于接近正常激波处理的流动，马赫数是非常低的；但即使对于一些斜激波经过的流动，它们看起来也像在亚声速条件下。应用于流出边界的 OutFlowBC_Simple 边界条件是通过将流动数据从边界内复制到边界外的虚拟单元格来工作的。这个简单的过程根本不能处理好跨边界的亚声速流动，从而导致整个仿真不能很好地反映物理情况。一个好的解决办法是改变气流域，使流出的气流基本上是超声速的。



(a)来流马赫数为 1.5 (b)来流马赫数为 1.6

图 3.11: H=1.6 大的流域内，32°半角圆锥上，低分辨率仿真 15ms 时的压力场。



(a)来流马赫数为 1.5，显示了 M_local 的全量程

(b)来流马赫数为 1.6，注意，展示部分区域的 M_local 是为了更清晰地显示跨声速区域。

图 3.12：32°半角圆锥、H=1.6 的大流域内，低分辨率仿真 15 ms 时的马赫数场。

3.8 建立一个更鲁棒性的仿真

解与您在物理实验中应该做的非常相似。如果边界效应搅乱了您的流动，可以把边界移开。幸运的是，这在数值模拟中(通常)很容易做到。现在,我们将另一个块添加原始域的下游边缘，并有效地将流出移到更远的下流区域。图 3.13(在以下输入脚本中称为 quad2、grid2 以及 blk2)所示额外的块允许在流出流出面边界条件前，气流恢复为超声速流动状态。

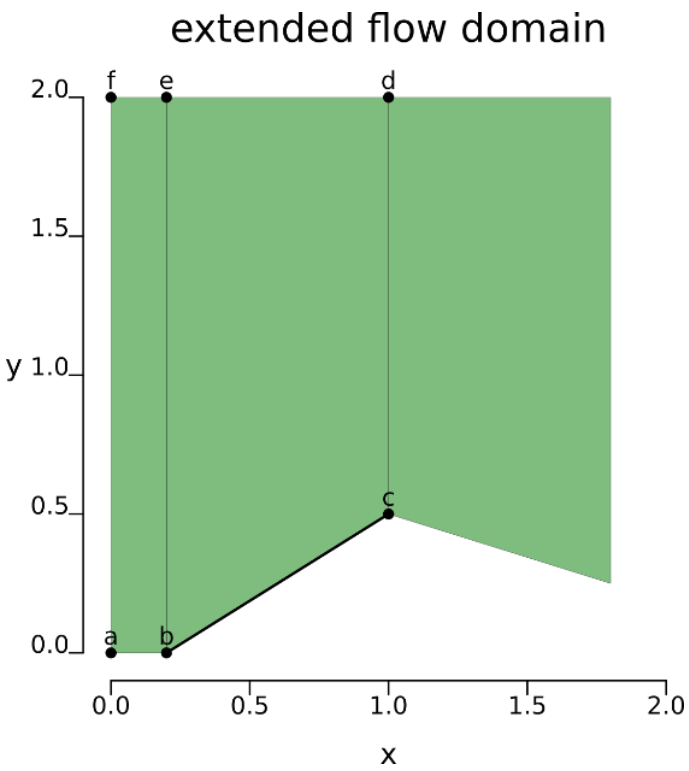


图 3.13: 20 度半角圆锥的扩展几何示意图。

3.8.1 输入脚本(.lua)

```
1 -- conepe.lua
2 -- Parametric, extended setup for sharp-cone simulation.
3 -- PJ & RG
4 -- 2016-09-23 -- adapted from conep.lua
5
6 -- We can set individual attributes of the global data object.
7 config.dimensions = 2
8 config.axisymmetric = true
9
10 -- The gas model is defined via a gas-model file.
11 nsp, nmodes, gm = setGasModel('ideal-air-gas-model.lua')
12 print("GasModel set to ideal air. nsp= ", nsp, " nmodes= ", nmodes)
13 initial = FlowState:new{p=5955.0, T=304.0, velx=0.0}
14 -- Compute inflow from Mach number.
15 inflow_gas = FlowState:new{p=95.84e3, T=1103.0}
16 M = 1.5
17 Vx = M*inflow_gas.a
18 print("inflow velocity Vx=", Vx)
19 print("dynamic pressure q=", 1/2 * inflow_gas.rho * Vx * Vx)
20 inflow = FlowState:new{p=95.84e3, T=1103.0, velx=Vx}
21 print("T=", inflow.T, "density=", inflow.rho, "sound speed= ", inflow.a)
22
```

```
-- Parameters defining cone and flow domain.
theta = 32 -- cone half-angle, degrees
L = 0.8 -- axial length of cone, metres
rbase = L*math.tan(math.pi * theta/180.0)
x0 = 0.2 -- upstream distance to cone tip
H = 2.0 -- height of flow domain, metres
config.title = string.format("Mach %.1f flow over a %.1f-degree cone.",
                             M, theta)
print(config.title)

-- Set up two quadrilaterals in the (x,y)-plane by first defining
-- the corner nodes, then the lines between those corners.
a = Vector3:new{x=0.0, y=0.0}
b = Vector3:new{x=x0, y=0.0}
c = Vector3:new{x=x0+L, y=rbase}
d = Vector3:new{x=x0+L, y=H}
e = Vector3:new{x=x0, y=H}
f = Vector3:new{x=0.0, y=H}
ab = Line:new{p0=a, p1=b} -- lower boundary, axis
bc = Line:new{p0=b, p1=c} -- lower boundary, cone surface
fe = Line:new{p0=f, p1=e}; ed = Line:new{p0=e, p1=d} -- upper boundary
af = Line:new{p0=a, p1=f} -- vertical line, inflow
be = Line:new{p0=b, p1=e} -- vertical line, between quads
cd = Line:new{p0=c, p1=d} -- vertical line, outflow
quad0 = makePatch{north=fe, east=be, south=ab, west=af}
quad1 = makePatch{north=ed, east=cd, south=bc, west=be, gridType="ao"}
-- extend the flow domain
xend = x0 + 2 * L
quad2 = CoonsPatch:new{p00=c, p10=Vector3:new{x=xend, y=rbase/2},
                       p11=Vector3:new{x=xend, y=H}, p01=d}
-- Mesh the patches, with particular discretisation.
dx = 1.0/40
nx0 = math.floor(x0/dx); nx1 = math.floor(L/dx); ny = math.floor(H/dx)
grid0 = StructuredGrid:new{psurface=quad0, niv=nx0+1, njv=ny+1}
grid1 = StructuredGrid:new{psurface=quad1, niv=nx1+1, njv=ny+1}
grid2 = StructuredGrid:new{psurface=quad2, niv=nx1+1, njv=ny+1}
-- Define the flow-solution blocks.
blk0 = FluidBlock:new{grid=grid0, initialState=inflow}
blk1 = FluidBlock:new{grid=grid1, initialState=initial}
blk2 = FluidBlock:new{grid=grid2, initialState=initial}
-- Set boundary conditions.
identifyBlockConnections()
blk0.bcList[west] = InFlowBC_Supersonic:new{flowState=inflow}
blk2.bcList[east] = OutFlowBC_Simple:new{}
```

```
67
68 -- add history point 1/3 along length of cone surface
69 setHistoryPoint{x=2 * b.x/3+c.x/3, y=2 * b.y/3+c.y/3}
70 -- add history point 2/3 along length of cone surface
71 setHistoryPoint{ib=1, i=math.floor(2 * nx1/3), j=0}
72
73 -- Do a little more setting of global data.
74 config.max_time = 30.0e-3 -- seconds
75 config.max_step = 15000
76 config.dt_init = 1.0e-6
77 config.cfl_value = 0.5
78 config.dt_plot = 1.5e-3
79 config.dt_history = 10.0e-5
80
81 dofile("sketch-domain-extended.lua")
```

3.8.2 最终结果

对于高度为 H =2 的流域，图 3.14 为仿真 30 毫秒时的马赫数场。这是在小流域仿真中所显示时间的两倍，在小流域仿真中，气流显然是阻塞的。与圆锥尖端微分离的激波要清楚得多，但对于接近正常的激波处理了来流的上半部分，上边界仍表现出有较强的影响。紧跟在脱体激波后面的微亚声速的马赫数值用淡蓝色清楚地显示了出来。

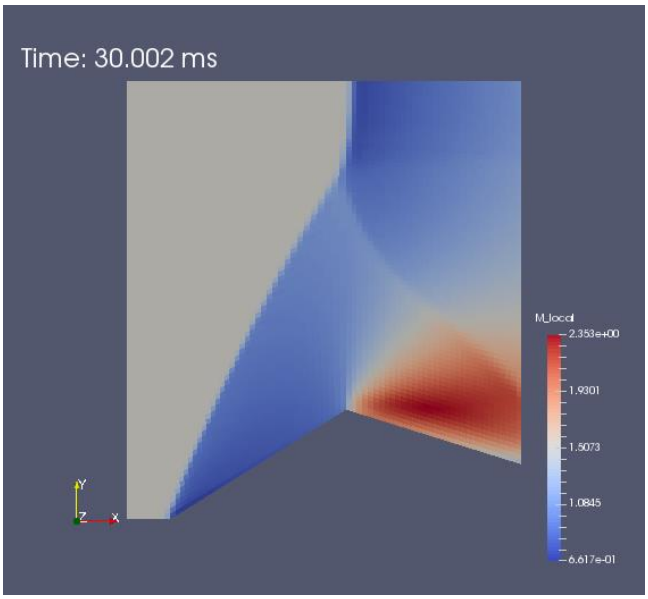


图 3.14：32 度半角圆锥，在 1.5 倍来流马赫数下，经过 30ms 气流低分辨率仿真的马赫数场。流域高度 H = 2。

因为我们已经做了很多努力来获得下游的边界条件，我们应该再次充分利用参数化建模，通过简单地设置 H = 3 来提高流域高度，并再次运行模拟来完成这项工作。

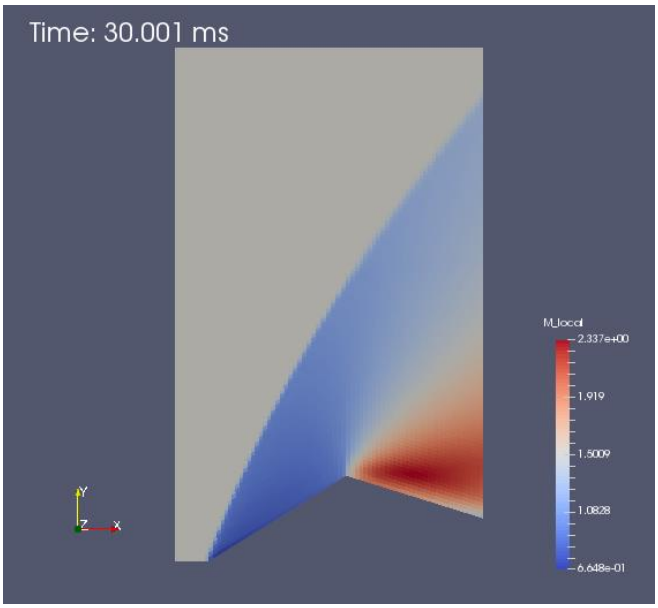


图 3.15: 32 度半角圆锥，在 30 ms，1.5 倍马赫数下，气流低分辨率仿真的马赫数常。流域高度 H = 3。

这一次，图 3.15 中的流场看起来很干净，基本上没有明显的边界诱导问题。OutFlowBC_simple 边界基本上有一个清晰的超声速流通过它，可以相信 eilmer 代码的表现良好。这将是宣布胜利的合适时间，然而，导师现在指出，从圆锥表面末端的角落辐射出来的膨胀波可能影响到弯曲激波后面的整个亚声速区域。教程到此为止[8]。

[8] https://www.churchofengland.org/prayer-worship/worship/book-of-common-prayer/the-order-for-morning-prayer.aspx

4.Eilmer 使用指南

4.1 运行仿真

仿真工作的设置需要许多练习，体现在编写基于文本的流域及其边界条件气体模型的描述。这个输入脚本作为扩展名为“. lua”的 Lua 源文件，呈现给准备阶段。一旦您使用您最喜欢的文本编辑器将您的工作规范要求作为输入脚本，仿真数据就会通过 Eilmer 程序在以下几个阶段中生成：

1.创建几何定义、网格和初始流动状态。对于从简单到中等复杂的几何图形，内置的几何图形工具(在指南报告[8]中有描述)就足够了，而且通常用起来十分方便，因为您不需要其他任何网格准备工具。对于复杂的几何图形，您可能会发现从 Gridpro 或 Pointwise 等专门的网格工具导入结构化块或非结构网格更合适。

2.运行仿真代码的主要相，以在后续时间中生成流动数据。

3.重新定义流场数据的格式，以生成适合于数据查看程序(如 Paraview 或 GNU-Plot)的文件。

4.1.1 准备工作阶段

运行仿真的准备阶段作为主仿真程序的一种特殊模式来实现。一看到--prep 命令选项，程序就会加载所有 Lua 包装类，这些类可能对定义几何对象、网格、热化学模型以及流动条件很有用。然后，它加载一个设置许多类和函数的 Lua 脚本，这些类和函数将用于帮助构建边界条件和流体块。最后，它加载由--job 命令选项进行标识的用户输入脚本，根据您的脚本定义构造对象，并编写一组网格、初始流动文件以及配置和控制参数的 JSON 描述。

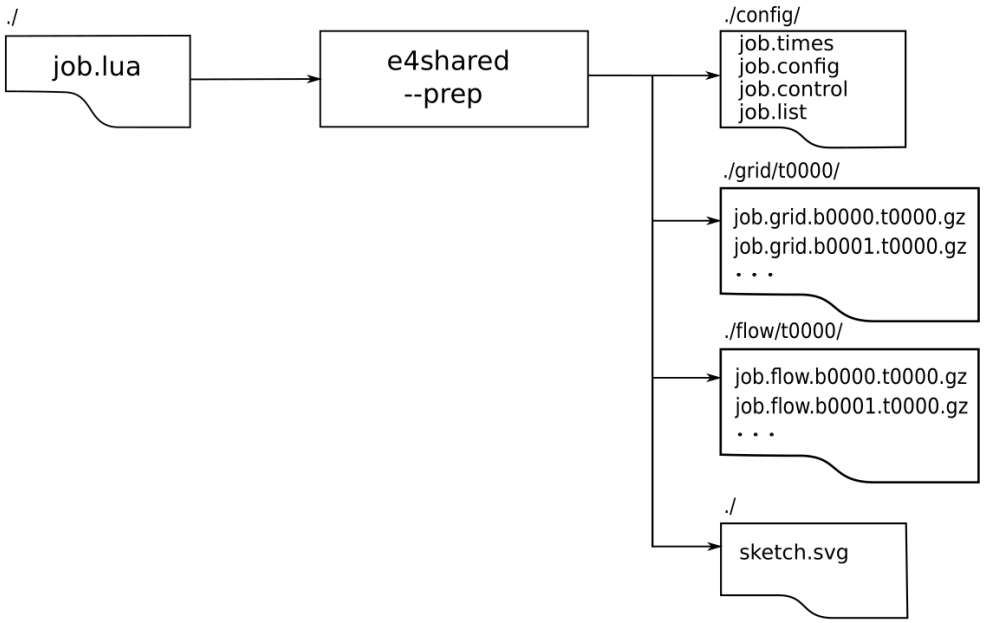
使用命令创建几何图形定义和网格：

$ e4shared—pre--job=*job*

命令中的斜体单词*job* 应该替换为您所选择的工作名称。然后将该名称作为基础，用来导出每个与仿真相关文件的特定名称。您至少有一个带有.lua 扩展名，称为 job.lua 的输入脚本，这也表明脚本是用 Lua 语言编写的。准备阶段的文件如下：

•*job*.config : JSON 格式的配置参数数据库。虽然您可能永远不会手动地从头

组装这些参数文件中的任何一个，但有时更改一个或两个值并重新运行仿真，而不完全重新生成该文件是很方便的一件事。手动编辑 JSON 文件时要注意;解析器是不允许发生错误。



•*job*.control：一个小的参数数据库，用于控制时间步长、最终时间以及流场的解和历史数据之间的间隔。该文件的内容也是 JSON 格式，它在每个第 nth 步开始时解析，其中 n 由 control_count 参数(默认值为 10)中的计数值给出。通过这种方式，用户可以更改仿真特性(使用此文件)，而不必重新启动仿真。为了完全地停止仿真，需要将 halt_now 条目设置为 1。这个参数可以在文件末尾找到。其他控制参数在第 4.10 节中被标记为：‡

•*job*.times：时间戳到写入仿真数据的实际时间的映射。在准备阶段之后，应该只剩下零时间条目。

•*job*.list :块的数目列表、网格类型以及给块的编号(可能是默认值)。块编号从零开始。

•sketch.svg：有时很容易用它看到流域和边界条件的图形表示形式。有一组小的补偿函数可用于将几何对象，如路径、表面和体积，补偿到可伸缩的矢量图形文件中。SVG 文件可以在诸如 Inkscape (http://www.inkscape)之类的程序中编辑，以及作为特定仿真文档的一部分结果。

• 每个块含有一个网格数据文件：job.grid.b0000.t0000.gz 、job.grid.b0001.t0000.g、…包含有限体积单元的网格。网格以相对简单的格式

编写为压缩文件。每个文件中点的顶点坐标与结构化网格的单元格顶点相关联。注意，网格和流动文件写在工作目录中的子目录。

• 每个块含有一个流动数据文件：job.flow.b0000.t0000.gz、job.flow.b0001.t0000.gz、…在每个有限体积单元格中包含初始流动状态。

请注意，机器生成的配置文件写在./config/ 子目录中，但是网格和流动数据文件分别写在./grid/和./flow/子目录中。对于固定网格仿真，在每次输出时只写一次网格(时间为 0，子目录 grid/t0000/中)，并将流动文件写入一个新的子目录(flow/tnnnn/)中。这是为了保持主目录的整洁，并允许轻松地复制或移动单个解的次数。对于移动的网格仿真，在每个输出时间都有一个网格目录，其中包含网格的位置信息。

数据以纯文本形式写入，该文本是“gzip”格式，因此是“.gz“扩展名。数据布局的细节记录在源代码中。在源代码文件中查找函数 read_grid、write_grid、read_solution 和 write_solution。命令例如：

$ grep -n read solution * .d

可能是一种绕过源代码的好方法。您还可以解压任何输出文件，然后使用标准文本编辑器读取它们。查看流动文件的前几行，看看为每个单元格写了什么数据元素。该格式用出现在第 5 行的变量名进行自我描述。记住，数据的单位是 SI-MKS。

4.1.2 检查网格

在运行仿真代码之前，有必要检查网格是否按计划运行。许多仿真都因为网格存在缺陷导致未能启动。常见的问题包括网格扭曲，以及相邻块的边缘与应该连接的位置不匹配。要获得一组可加载到 Paraview 中进行检查的图像文件，请使用后处理程序：

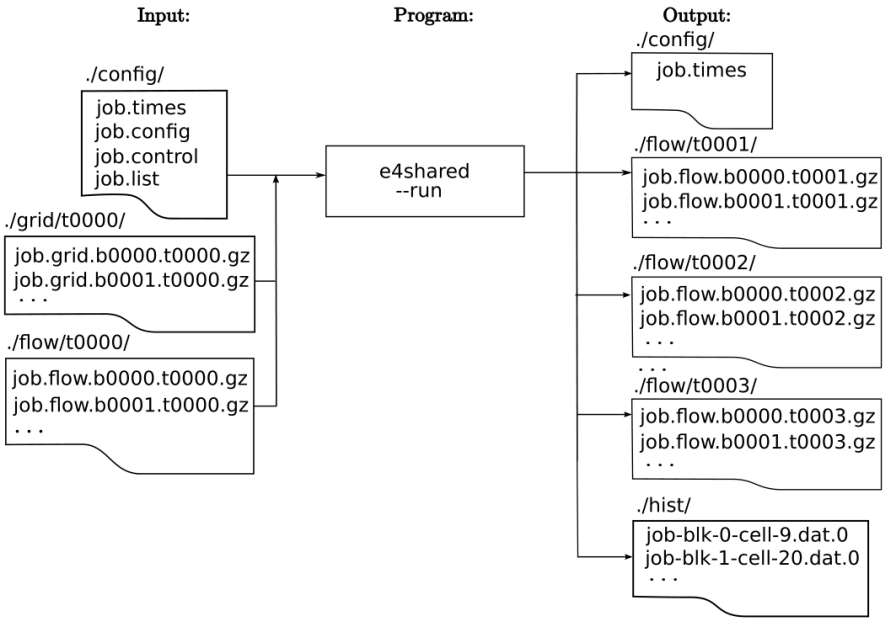
$ e4shared --post --job=*job* --tindx-plot=0 --vtk-xml

然后使用 Paraview 获取结果文件以进行检查。生成的文件将出现在 plot/子目录中。有关后处理阶段的更完整讨论，请参阅第 4.1.5 节。

4.1.3 运行仿真

运行仿真代码以在后续时间生成流动数据[9]。

$ e4shared --job=job --run



输出文件如下：

•*job*.flow.bnnnn.tmmmm.gz：需要的次数内所有单元格的流动数据。随着仿真的进行，全场的解都写在新文件中，其中 nnnn 表示块号，mmmm 表示时间戳。查一下 *job*.times 文件，看看每个时间戳(或 tindx)的时间值。与网格文件一样，每个流动解决方案的文件都是用简单布局的纯文本编写的，与结构化块网格的 Tecplot 格式点没有太大区别。在这些文件中，文件内点的空间坐标与单元格中心相关联。

•*job*-blk-n-cell-m.dat.0：特定“历史点”和请求次数的数据。这些数据通常用于模拟安装在模型表面的压力传感器以及热传感器记录的信号。第一次运行仿真工作时，历史文件的尾索引为.0。在重新启动或重新运行仿真时，Eilmer 将打开新的历史文件，其尾索引比最新的现有文件多一个。要从特定单元格的历史文件集合中生成单个数据文件，只需使用 Linux 系统的 cat 命令来连接它们的内容。如果您在开始时多次运行仿真，并且不希望从以前运行的仿真中生成文件，则需要在每次运行之前手动删除历史文件。

[9] 如果仿真完成得太快(可能根本没有进行任何步骤)，则可能是初始步长太大，计算不稳定导致的。这种情况的征兆是：报告的时间步长最终值非常大。解决方法就是为 dt_init 选择一个适当的小值，然后重试。

•*job*.times：时间戳到写入仿真数据的实际次数的映射。主仿真将行追加到此文件，这个文件对于一些自动化后处理操作很有用。

作为参考，以下是在命令行给出--help 选项时的提示:

```
$ e4shared --help
  Eilmer 4.0 compressible-flow simulation code.
  Revision: d50a10ec
  Compiler-name: dmd
  Build-flavour: debug
  Capabilities: multi-species-gas multi-temperature-gas MHD turbulence.
  Parallelism: Shared-memory
  Usage: e4shared/e4mpi/... [OPTION]...
```

Argument:	Comment:
--job=<string>	file names built from this string
--verbosity=<int>	defaults to 0
--prep	prepare config, grid and flow files
--no-config-files	do not prepare files in config directory
--no-block-files	do not prepare flow and grid files for blocks
--only-blocks="blk-list"	only prepare blocks in given list
--run	run the simulation over time
--tindx-start=<int>|last|9999	defaults to 0
--next-loads-indx=<int>	defaults to (final index + 1) of lines found in the loads.times file
--max-cpus=<int>	(e4shared) defaults to 8 on this machine
--threads-per-mpi-task=<int>	(e4mpi) defaults to 1
--max-wall-clock=<int>	in seconds, default 5days * 24h/day * 3600s/h
--report-residuals	write residuals to file config/job-residuals.txt
--post	post-process simulation data
--list-info	report some details of this simulation
--tindx-plot=<int>|all|last|9999|"1,5,13,25"	defaults to last
--add-vars="mach,pitot"	add variables to the flow solution data (just for postprocessing).Other variables include:total-h, total-p, enthalpy, entropy, molef, conc,Tvib (for some gas models)
--ref-soln=<filename>	Lua file for reference solution
--vtk-xml	produce XML VTK-format plot files
--binary-format	use binary within the VTK-XML
--tecplot	write a binary szplt file for Tecplot
--tecplot-ascii	write an ASCII (text) file for Tecplot
--tecplot-ascii-legacy	write an ASCII (legacy, text) file for Tecplot
--plot-dir=<string>	defaults to plot
--output-file=<string>	defaults to stdout
--slice-list="blk-range,i-range,j-	output one or more slices across

range,k-range;..."	a structured-grid solution
--surface-list="blk,surface-id;..."	output one or more surfaces as subgrids
--extract-streamline="x,y,z;..."	streamline locus points
--track-wave="x,y,z(,nx,ny,nz);..."	track wave from given point in given plane, default is n=(0,0,1)
--extract-line="x0,y0,z0,x1,y1,z1,n;..."	sample along a line in fluid domain
--extract-solid-line="x0,y0,z0,x1,y1,z1,n;..."	sample along a line in solid domain
--compute-loads-on-group=""	group tag
--probe="x,y,z;..."	locations to sample flow data
--output-format=<string>	gnuplot|pretty
--norms="varName,varName,..."	report L1,L2,Linf norms
--region="x0,y0,z0,x1,y1,z1"	limit norms calculation to a box
--custom-script | --custom-post	run custom script
--script-file=<string>	defaults to "post.lua"
--help	writes this long help message

这些命令行选项大部分用于后处理阶段，只有少数用于运行主仿真。您可以看到它们分组在--run 选项下面。默认情况下，tindx-start 的初始值为 0，程序尝试使用尽可能多的可用 CPU 内核，直到达到块的数量。最大时间的默认限制是 5 天，相对而言是一个非常大的值。大多数情况下，在使用共享批处理系统时，您只需要注意这个选项，因为它可能会对您的运行时间设置上限。设置 max-wall-clock 小于批处理系统的限制值，以确保程序会停止计算，并在批处理系统突然终止[10]您的工作之前写入解决方案文件。

4.1.4 重新启动仿真

默认情况下，仿真程序选择 tindx 值等于 0 的流动解决方案，但它可以选择其他任何 tindx 快照。要选择一个解决方案并继续，最好做一点内务工作，在运行结束时检查仿真的状态，然后编辑 *job*.control 文件，并将 dt_init、max_time 以及 max_steps 变量更改为适当的值。不要再次运行准备阶段，否则会覆盖您需要保持的 *job*.次数文件，以及您新编辑的 *job*.control 文件。此时，您应该准备好再次运行主仿真程序。记得在命令行上提供相关的 ttindx-start 值，以便重新启动。例如:

```
$ e4shared --job=name --tindx-start=5 --run
```

[10] 在几天的模拟中，我们建议您请求一个 max-wall-clock，它比我们从批处理系统请求的时间少 10 分钟(600 秒)。对于仿真来说，最后的 10 分钟通常足以写出数据并干净地打包，即使在繁忙的集群中也是如此

此外，在重新启动时，要注意您有统一的建模要求和设置。如果用 k-ω 模型将层流仿真重新启动为湍流仿真，将导致数据不一致。启动一个新工作并使用 FlowSolution 对象(参见 4.5 节)来获取旧数据可能会更好。请注意，您的新旧解决方案需要有统一的数据，如化学物质的数量等。如果使用 FlowSolution 旧解决方案中可用的数据，并不足以填补缺失的值。

4.1.5 后处理

仿真数据的后处理是模拟活动中最没有结构化的部分，要全面描述您要做的事情是很困难的。关于后处理活动的范围，可以通过--help 选项得到一些提示,该提示列出的大多数命令行选项都与从先前完成的模拟中提取数据有关。

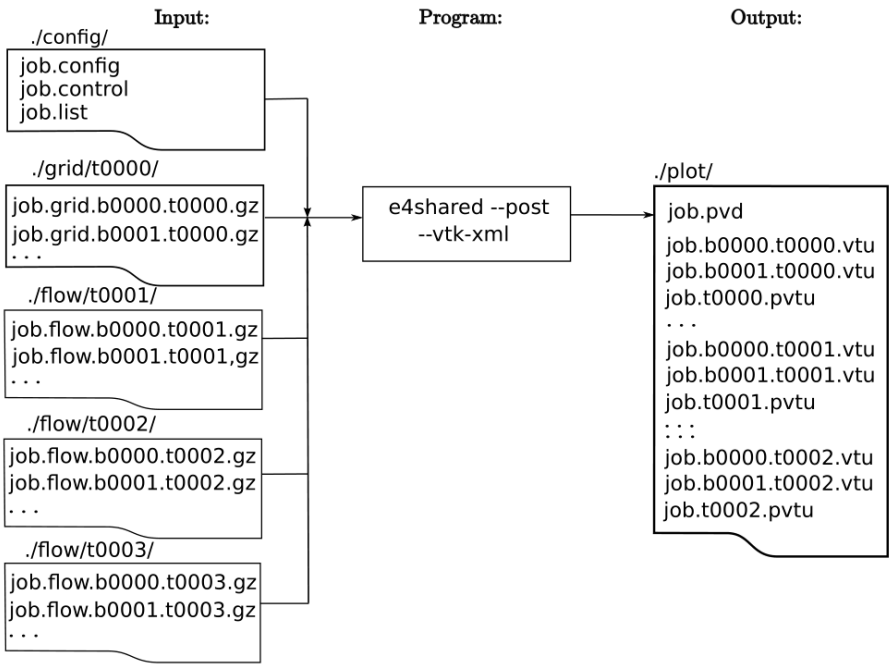
我们提供了一种后处理模式--post，它具有获取仿真数据的基本功能，并以适合的格式，将流场文件写入 Paraview、Visit、Tecplot、Plot3D 或 gnuplot[11]中。

若要将解的数据重新定义为一个包含域所有流动数据的非结构化网格，并以适合 Paraview 或 Visit 的格式编写此数据，请使用以下命令:

$ e4shared –post --job=*job* --vtk-xml --tindx-plot=all

要进行更复杂的后处理，可以用相当复杂的方式组合命令行选项；为了达到预期的效果，用户可能需要进行一些试验。但是，这些选项可以分成若干个子集。

[11] 参见网站：http://www.paraview.org、https://wci.llnl.gov/codes/visit/、http://www.tecplot.com、http://people.nas.nasa.gov/ ~ rogers/plot3d/intro.html、以及 http://www.gnuplot.info



数据加载选项包括：

•--job=<string>：指定结果文件的根名称。扫描配置、控制和列表文件，以获取关于计算结果的信息。这包括气体模型的信息，该模型已被初始化并可供使用。

•--tindx-plot=<int>|all|last|9999：您可以通过其数值指数来获取一个计算结果时间，也可以通过关键字指定所有结果的时间。最后一个结果的框架(需要在 job.times 文件中标识)，可以通过将给定指数指定为 last 或 9999。

数据添加选项：

•--add-vars：无论是对于整个字段(VTK、Tecplot 和 Plot3D 格式)还是对于切片数据，将命名的变量添加到绘图数据集中。这些流动变量不在 Eilmer 本机计算结果文件中，必须在后处理阶段重新构造。可用的流动变量有马赫数(“mach”)、皮托管压力(“Pitot”)、总焓(“total-h”)、总压强(“total-p”)、比熵(“entropy”)、化学物质的摩尔分数(“molef”)和化学物质的摩尔浓度(“conc”)。如果需要许多这样的变量，可以用逗号分隔每个变量名，将它们连接到一个列表中。

整个场的输出选项：

•--vtk-xml：Visualization Tool Kit (VTK)的 XML 格式对 Paraview 和 Visit 来讲都是可读的。默认情况下，XML 文件是较简单的文本，并且可能非常大。

•--binary-format：将大部分数据作为附加的二进制记录写到 VTK 文件中。这使得文件与 XML 文件不兼容，但它确实减少了大型数据文件的大小，并提高了将它们加载到 Paraview 中的速度。对于大型 3D 数据集，这是一个很好的选择。

数据切片和切片选项：

•--output-file=<profile-data-file>：指定要将请求的数据转存到其文件的名称。这个命名选项与各种切片选项相关，也与面的列表选项相关，其中这个面的列表选项用作生成 VTK 文件的根名称。这将允许您创建许多用于绘图的切片数据集。

•--slice-list="blk-range,i-range,j-range,k-range;..."：提取数据的子集。规范字符串中使用了切片符号，它应该用引号括起来，如代码所示。可以在一个字符串中指定多个切片(用分号分隔)。每个切片规格由 4 个指数或以逗号分隔的指数范围组成。指数是单个整数值或$，用于指示可用范围的末尾。一个指数范围可以是一个冒号分隔的整数对、一个冒号和一个极限，或者仅仅是一个冒号(表示整个范围)。注意，范围限制是包含在内的。所以,如在二维结构化网格仿真中，为了从 block 0 中提取单元格的东部带，您应该使用"0,$,:,0"字符串,因为我们想要第零个块、i= max ($)的位置、所有在 j 方向(:)的单元格以及 k=0 处方向的二维网格。

•--surface-list="blk, surface-id;…"：从全流场中提取一组表面数据，并将它们写入 VTK 文件。输出文件选项用于指定一个基本文件名，用于构造 VTK 文件集的名称。对于结构化网格，表面 id 可以是一个整数，也可以是一个结构化网格的北、东、南、西、顶或底之一。

•--extract-line="x0, y0, z0, xl, y1, z1, N"：在指定的端点之间生成最多 N 个采样点的列表。采样的数据写入每个采样点的包围单元格中心，重复的单元格会被忽略。

•--probe="x, y, z;…"：报告指定点的采样数据。所选数据以 gnuplot 格式写入。

数据操作和汇总选项：

•--ref-solution=<Lua-script>：比较计算结果和 Lua 脚本提供的结果。区别在于输出。这意味着输出文件中记录的值是有差异的，体现在参考结果和仿真

结果之间。差异仅在参考结果中给定的字段变量进行。例如，如果参考结果为密度提供一个值，那么 rho 输出字段将具有不同的值。

•--report-norms：返回所有流动变量的规范字典。可用的规范是 L1、L2 和 Linf(最大量级)。这个用在一个附加 ref-solution 的连接词上。

•--region="x0,y0,z0,x1,y1,z1"：限制了一个特定框的规范计算。

注意，在将字符串输到程序之前，必须对某些规范字符串使用双引号，以防止命令 shell 将字符串拆开(或以其他方式更改它)。同样值得注意的是，在默认情况下，e4shared 在成功运行时不会向控制台写入太多内容。如果您希望在它执行工作时获得更多的注释，请为该选项提供一个非零整数--verbosity。值 1 应该会给出主要活动的简要介绍，而值 2 将会提示更多的消息。

通过指定--custom-script 模式和一个带有--script-file 的 Lua 脚本文件，可以进行特殊的后处理。程序启动，加载所有的 Lua 范围包，然后执行指定的脚本。有一些种类是用于获取整个计算结果(即 FlowSolution)，并访问网格或流动数据的任何部分。一些特定的应用程序用于显示自定义后处理脚本的编写，他们是：

•激波在轴对称流动中经过圆锥的角度估计值(第 3.2 节)。

•为有限体积的圆筒仿真找到弓形激波的位置(第 5.2 节)。

虽然我们介绍的这种模式是后处理部分，但它有更普遍用途。例如，您可能对一些简单的气体动力学计算感兴趣，这些计算使用了附录 D 中的函数。这些函数被加载到在 e4shared 中启动的 Lua 解释器，并可由脚本文件中的 Lua 代码有效使用。

4.2 输入脚本概述

因为您的规范脚本 *job*.lua 在运行时成为程序的一部分，所以它值得我们努力去学习 Lua。网站 https://www.lua.org 对于学习 Lua 编程语言是一个很好的起点，“Lua 编程”[12]文本的老版本，是一个很好的阅读文献，拥有您需要顺利地写好 Lua 脚本的一切信息，可以在网上找到。

在进行一些初始化之后，程序执行您的脚本文件，并收集几何图形和流动规格数据，转换为可提供给主仿真代码的格式。这种方法的优点是您可以从脚本中

获得 Lua 解释器的全部功能。例如，您可以执行计算以便对几何图形进行参数化，或者您可以使用 Lua 控制结构使重复的定义变得更加简洁。另外，您可以使用 Lua 注释和输出语句向脚本文件添加文档。输入脚本通常执行以下操作：

1. 设置一个气体模型。
2. 可选地创建几何元素，来帮助定义气体流域的边界表示形式。这是以面(二维)或体积(三维)的形式出现的。
3. 离散这些面或体积作为网格的有限体积单元格。
4. 通过指定边界条件和初始流动状态，在这些网格的基础上创建流体块
5. 设置一些仿真控制参数。

本手册中的大多数示例只做这些事情，但是，Eilmer 可以做更多的事情。

4.3 指定热化学模型

热化学模型由气体模块[5]提供。这是一个具有 Lua 界面的 D 语言模块，因此可以从用户的输入脚本中访问它的对象和方法。现在，我们只告诉您如何设置理想空气的气体模型。首先将下列文本放入 ideal-air.inp 文件中:

```
model = "IdealGas"
species = {"air"}
```

并运行以下命令：

```
$ prep-gas ideal-air.inp ideal-air-gas-model.lua
```

来生成 ideal-air-gas-model.lua 文件，它包含完全指定的气体模型。注意,不要输入我们上面显示的 $ 命令提示符。

要在您的工作输入脚本中使用这个气体模型，请使用以下命令行(插入到 lua 文件中使用)：

```
nsp, nmodes, gm = setGasModel('ideal-air-gas-model.lua')
```

这将在程序中初始化气体模型，并返回种类数、能量模式以及气体模型对象的引用数。您不需要像这里所示的那样分配这些返回值，但是您可能会发现，仅在计算或准备阶段访问它们是非常方便的。对于这里所示的理想气体模型，种类数为 1，非平衡能量模式数为 0。

对于更复杂的气体模型，上面显示的行就是在工作脚本中初始化气体模型所

需的全部内容。当然，您将完成建立复杂模型的所有详细工作，并在一个相关的 Lua 脚本中提供详细信息。所有的细节都在气体包文档[5]中，但稍后会在本手册中研究几个例子，然而我们的讨论会限制在气体模型上，它具有冻结内部模式或平衡态内部模式，以至于单个温度就足以计算气体热能。

4.3.1 有限速率化学动力学

涉及非平衡化学的仿真需要一个额外的输入文件，来描述参与的气体种类及其反应。该文件的准备工作在指南报告[5]中进行了描述。

4.4 定义流动状态

由于 Eilmer 是一种流动仿真代码，因此需要指定整个流域的初始气体流动条件。此外，可能需要在适当的边界面上指定自由流的流入边界条件，这取决于您的流域。

要在输入脚本中为一个或两个目标定义这样的流动条件，可以构造一个 FlowState 对象[12]，如下所示:

```
fs = FlowState:new{p=p, T=T, massf=mf, T_modes=T_modes, quality=q,
                   velx=v x , vely=v y , velz=v z ,
                   tke=tke, omega=ω, mu_t=μ t , k_t=k t ,
                   Bx=β x , By=β y , Bz=β z , psi=ψ}
```

对于单个温度气体模型，只需要两个字段值。它们是：

- p: 以 Pa 为单位的压力。
- T：以 K 为单位的温度。

其它的文件可以选择性地指定。

- T_modes: 对于具有多模态能量的气体模型，这些是这些模型对应的温度，以数组形式提供。如果不提供平衡态信息，则假定有平衡态，并为每个赋值为 T。对于只有一个温度的气体模型，忽略这个字段。
- mf：组成物种的质量分数。如果气体模型中只有一种，则默认值为 1.0。如果您确实提供了字段值，那么它将质量分数值提供给物种名称表。对于上面示例的理想空气，您可以提供 massf={air=1.0，}。如果在多物种模型中没有

[12] FlowState 类定义在 prep.lua 源文件中。

物种名称，则相应的质量分数假定为0.0。注意所提供的质量分数总和为1.0，必须在1.0×10^{-6}的误差范围内。

•v_x, v_y, v_z：速度分量，单位是 m/s。默认值为 0.0。

•q：两相混合物的质量。默认值为 1.0(如所有气相)。

•tke：单位质量湍流动能，单位是m^2/s^2或 J/kg，默认值为 0.0。

•ω：湍流涡量，单位是 1/s，默认值为 1.0。

•μ_t：湍流粘度，单位是 Pa.s，默认值为 0.0。

•k_t：湍流热导率，默认值 0.0。经常可以计算为$C_p\mu_t/Pr_t$。

•$\beta_x, \beta_y, \beta_z$：磁场分量，单位是特斯拉。默认值为 0.0。

•ψ：MHD 计算的散度参数。

在 Lua 环境中，FlowState 对象是允许访问完整内部状态的表，包括派生的量，如密度和声速。字段的完整集合是：

•gm：与此 FlowState 相关联的气体模型对象。

•nSpecies：气体模型中化学物质种类数。

•speciesNames：提供名称的字符串数组。

•nModes：其他内能模式的数量[13]。

•p：压强，单位是 Pa。

•T：温度，单位是 K。

•T_modes：其他内部温度的数组。

•quality：气相的比例。

•massf：指定质量分数值的表。

•a：声速，单位是 m/s。

•rho：密度，单位是 kg/m^3。

•mu：动力粘度，单位是 Pa.s。

•k：导热系数，单位是 W/(m.K)。

•tke：湍流动能，单位是 J/kg。

•omega：湍流涡量，单位是 1/s。

[13] 对于单温度气体模型，这个数是零。对于多温度气体模型，数字将指示有多少内能模式，超越反旋式。

•mu_t：湍流粘度，单位是 Pa.s。

•k_t：湍流传导率，单位是 W/(m.K) 。

•velx, vely, velz：速度分量，单位是 m/s。

•Bx, By, Bz, psi, divB：磁场参数。

表里也包含了 GasState 对象。注意，FlowState 对象是在气体模型环境中定义的，所以在构造任何 FlowState 对象之前您需要调用 setGasModel(如 4.3 节所示)。

4.5 导入其它仿真的流动状态

对于自定义的后处理，您需要能够为仿真挑选网格和流动数据，并可以检查任何特定的单元格，然而，在有些时候，准备一个新的仿真时，您可能仍想使用一个旧仿真中的流动数据。这些数据可以用在新仿真中部分或所有块的初始条件。一个典型的例子是用更精细的网格重启仿真。在任何情况下，你都会构造一个 FlowSolution 对象，如下：

```
fsol = FlowSolution:new{jobName="job", dir="myDir",
                        tindx=tindx, nBlocks=nb }
```

其中已命名字段及其可能的值为：

•jobName：根文件名，用于访问包含结果数据的单个流动文件和网格文件。

•dir：我们将找到现有结果文件的目录。通常，它是当前目录，所以您可以指定。

•nBlock：计算结果数据集合的块数。

•tindx：选择 0..9999 的时间指数。不要指定前置零，否则可能会混淆十进制和八进制数字。

构造完 FlowSolution 对象后，可以提供很多方法访问数据。

•fsol:find_enclosing_cell{x=*x*, y=*y*, z=*z*}：返回一个字段为 ib 和 i 的表，分别对应块指数和单元格指数。注意，单元格的单个指数工作在非结构化网格和结构化网格的环境。如果没有提供任何 x、y 或 z 字段，则假定为 0.0 值。如果此方法未能找到一个封闭的单元格，则 ib 和 i 的返回值都是 nil。

•fsol:find_enclosing_cells_along_line{p0=$\overrightarrow{p_0}$, p1=$\overrightarrow{p_1}$, n=*n*}：返回一个数组表，位于单元格中心。每个表都有 ib 和 i 字段，分别给出了块指数和单元格指数。

$\overrightarrow{p_0}$和$\overrightarrow{p_1}$点可以指定为标记坐标的表。如果没有提供任何 x、y 或 z 坐标，则假定值为 0.0。或者，这些点可以作为 Vector3 对象。采样点的数量指定为 *n*，也可以设置为相当大的数目，以确保沿线选取所有的单元格。编译单元格数组的过程将消除重复项。

•fsol:find_nearest_cell_centre{x=*x*, y=*y*, z=*z*}：返回一个带有 ib 和 i 字段的表，分别给出了块指数和单元格指数。如果单元格又长又细，这个方法可能返回邻近单元格的指数，而不是封闭单元格的指数(如果存在的话)。

•fsol: get_nic (*ib*)：返回块 *ib* 在 i 方向上的单元格数量。

•fsol: get_njc (*ib*)：返回块 *ib* 在 j 方向上的单元格数量。

•fsol: get_nkc (*ib*)：返回块 *ib* 的 k 方向上的单元格数量。

•fsol:get_var_names()：返回一个表，以其中单元格数据的变量名作为字符串。

•fsol: get_cell_data{fmt=*dataFormat*, ib=*ib*, i=*i*, j=*j*, k=*k*}：返回特定单元格的单元格数据。*dataFormat* 值可以是 Plotting 或 Flowstate，区别在于返回表中各字段的名称。在稍加研究非结构网格块后，可以忽略 j 和 k 项，因为只能指定 i 指数。如果您愿意，可以使用单个 i 指数或所有三个指数访问结构化网格块。单个指数的访问可以很好地处理 find_enclosinq_cell 或 find_nearest_cell_centre 方法的结果。

• fsol:get vtx{ib=ib, i=i, j=j, k=k } or (fsol:vtx{ib=ib, i=i, j=j, k=k })：返回一个表示顶点位置的 Vector3 值。此方法未指定字段的值默认为零。

•fsol: get_sgrid{ib=*ib*}：返回指定块的 structuredGrid 对象。

4.6 流域的表示

现在我们有了气体模型和指定流动条件的方法，我们需要定义流动的空间域，以及一组边界条件。这个域被指定为有限体积的单元网格，可以是结构化的，也可以是非结构化的，取决于适应边界的形状。如果您打算使用 Eilmer 提供的几何原语库来设置流域的边界表示方式，那么在报告[8]将描述更多相应的细节。或者，您可能已经有了自己喜欢的网格生成器，也可以进行设置。无论如何，我们现在假设已经将流域定义为一个或多个网格单元，并且我们已经准备好将流动状

态与边界条件相结合。

4.6.1 建立在结构化网格中的流体块

对于二维的结构化网格，每个流体块都由 4 条边围成一个区域，分别为北、东、南、西。我们现在看图 4.1 中二维流域的平面图。i 和 j 指针与几何函数中使用的 r 和 s 参数坐标有关，同时它们的(r, s)坐标可识别角点。如果流域被定义为由多个块组成，那么这些角点用于在搜索中确定块的连接性。由于参数空间到物理空间的映射仅限于相对简单的插值，因此常常将复杂的流域细分为更简单的子域。

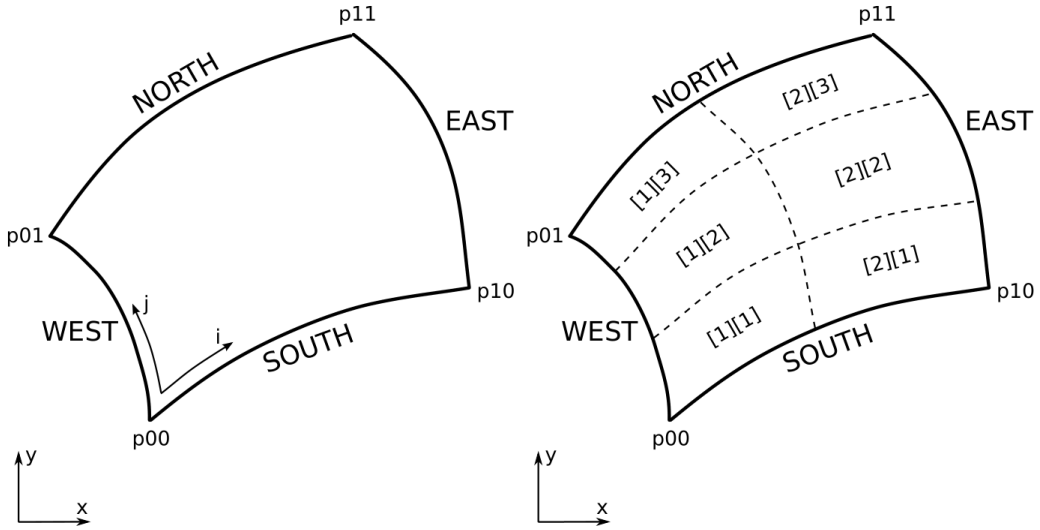


图 4.1:一个二维的流域，包含单个 FluidBlock 对象的结构化网格(左)和通过 FluidBlockArray 定义的子块集合(右)。边界路径的方向非常重要：东西路径的前进方向是参数坐标 s，从南到北；南北路径的前进方向是参数坐标 r，从西到东。

在三维坐标中，情况变得更加复杂了，6 个边界(北、东、南、西、顶、底)定义的每个块都要与域的实际表面相吻合。图 4.2 显示了“指数空间”视图，单元格指数 i、j 和 k 对应于几何函数中使用的 r、s 和 t 参数坐标。块的角顶点编号从 1 到 7，如图所示。

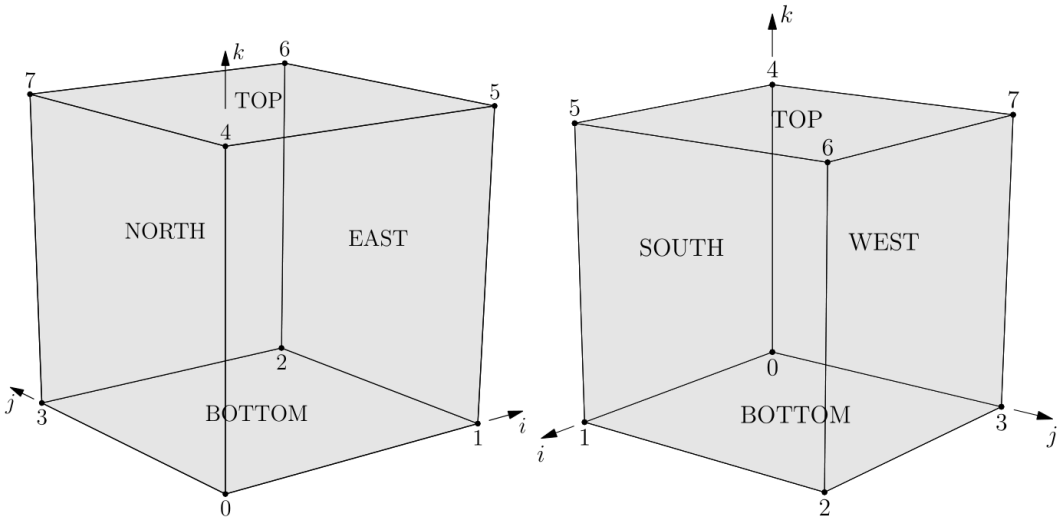


图 4.2：包含三维结构化网格的六面体块的两个视图。这些数字是模糊的，但每个数字都应该显示一个空心的盒子，盒子中每个视图的远表面都被标记。近表面是透明和未标记的。要获得明确的表示形式，请创建附录[8]中绘制的调试多维数据集。

要在您的输入脚本中定义一个结构化网格流体块，需要构造一个 FluidBlock 对象，如下：

my_block = FluidBlock:new{grid=*grid*, initialState=*fs*, omegaz=ω_z ,
bcList=*bcList*, label=*tagString*, active=*aFlag* }

其中，变量 *my_block* 的赋值允许在以后方便地引用该块，例如，用于添加边界条件。绑定到构造函数器表中的字段名称值表示[14]为：

•*grid*：一个结构化的网格对象，之前已被构建(或输入)，如报告[8]的第 3 章所述。

•*fs*：一个 FlowState(见 4.4 节)或 Lua 函数对象，给定坐标 x，y 和 z，以表的形式返回 FlowState。从 FlowSolution 对象中衍生出这个函数，如 4.5 节末尾所述。

•ω_z：旋转参考系的角速度(在 z 轴上，单位是 rad/s)。适用于涡轮机械中流动的计算。默认值为 0。

•*bcList*：一个命名条目表，绑定到边界条件对象。字段名是边界面名称(北、东、南、西、上、下)。如果感兴趣，请期待 4.7 节对可用边界条件的讨论。如果特定边界没有指定条件，则默认为 WallBC_WithSlip 边界无滑移条件。

[14] 当然，权威的来源是 prep.lua 中的 FluidBlock 类定义。

您不必向 FluidBlock 构造器提供任何或所有的边界条件。您可以在稍后的脚本中将边界条件附加到块边界。

•*tagString*：用于后处理的标签。在内部，仿真程序只关心它所持有块数组中块的指数，但这个字符串将被写入 job.list 文件，这应该是确定特定块的方便方法。

•*aFlag*：默认为真，以便在仿真时可以主动更新。虽然手动设置此字段对您来讲是不常见的，但是仿真程序的块步进模式使用此标志，有选择地更新块，将其作为整体结果的一部分。可以看第 4.10 节中的 config.block_marching 标志。

在多处理器计算机上定义大型域和运行仿真时，使用一次调用就定义多个 FluidBlock 对象是非常方便的。这种情况的调用函数是：

my fba = FBArray:new{grid=*grid*, initialState=*fs*, omegaz=ω_z ,
bcList=*bcList*, nib=n_i, njb=n_j, nkb=n_k}

它包含了 blockArray 一个多维的 FluidBlock 对象数组，这些对象在三维坐标中下标命名为[i][j][k]，在二维坐标系中下标命名为[i][j]，来达到访问单个 FluidBlock 对象的目的。注意，每个指数都以 Lua 默认值为 1 开始。传入的单个网格被细分为$n_i \times n_j \times n_k$子网格。注意在任何方向上的单元格数量，它们不会分裂成相等的部分。您可能没能得到您假定的块队列。在内部，应将新形成的内边界与 ExchangeBC_FullFace 边界条件连接在一起，并返回一些子块。外围边界将从传入的 *bcList* 继承，或者默认为 WallBC_Withslip 边界条件。如果没有指定每个方向上要生成的块数，则默认值为 1。

当为复杂几何图形装配大量的块时，有一个功能 identifyBlockConnections(*blockList*, *excludeList*, *tolerance*)，可以对所有相邻块执行暴力搜索，并为具有重合角(到给定公差内)的面附加 ExchangeBC_FullFace 边界条件。如果您不希望搜索遍历到目前为止生成的所有块，那么可以提供一个引用数组给块，将您确实希望搜索的块作为 *blockList*。您还可以为应该排除的块提供一个引用数组。如果不提供此参数，则将 *excludeList* 假设为一个空数组。对于主机代管的顶点，默认误差是 1.0e-6。

请注意 identifyBlockConnections()函数与连接角点的实际路径或表面的形式

是无关的。可能会有角重合，但路径和表面不一致。

如果您希望对连接块的过程有更多的控制，您可以使用 connectBlocks()函数手动连接块，该函数可实现逻辑连接，而不需要查看角的几何位置。这种情况可能会出现，例如，当您希望在流域的跨流动方向上应用周期边界条件时。然后，你想要连接的边界确实有不重合的角和面。要手动连接一对块,可以使用函数：connectBlocks (*A*，$face_A$, *B*，$face_B$，*orientation*)，这里 *A* 和 *B* 是对单个 FluidBlock 对象的引用，$face_A$、$face_B$ 是他们的邻边(北、东、南、西、顶部或底部)，方向是一个描述 Lua 文件 blk_conn Lua 的整数值。重要的是只在三维条件下提供这个方向。例如，顶点配对的表{{3,2}，{7,6}，{6,7}，{2,3}}指定一个方向为 0 的北-北连接。在二维平面中，对于每个可能的连接，只有一个方向是有效的，所以您不需要指定方向。

4.6.2 建在非结构化网格上的流体块

也可以在非结构化网格上定义流体块，甚至可以将它们作为主要构建在结构化网格块上的仿真的一部分。每个非结构化网格可以看作是一个没有任何全局结构指数方案的单元格包；单元格从 0 到 n_{cells}-1。这些单元所定义的区域将被一个或多个单元格面的 boundary-sets 所界定，并将边界条件分配给这些边界集。在特定边界集 boundary-sets 的单元面将在仿真过程中应用相应的边界条件。

要在您的输入脚本中定义一个非结构化网格流体块，请调用 FluidBlock 构造函数：

my_ublk = FluidBlock:new{grid=*grid*, initialState=*fs*, omegaz=ω_z ,
bcList=*bcList*, bcDict=*bcDict*,
label=*tagString*, active=*aFlag*}

其中，对变量 *my_ublk* 的赋值允许在以后的时间方便地引用该块。在构造函数中绑定到字段名称的值代表着[15]：

•*grid*：一个 UnstructuredGrid 未结构化的网格对象，之前已被构造(或导入)，如报告[8]的第 3 章所述。

•*fs*：一个 FlowState（见第 4.4 节）或 Lua 函数对象，给定坐标 x，y 和 z，以表的形式返回 FlowState。

[15] 再一次，权威的来源是在 prep. lua 中的 FluidBlock 类定义。

•ω_z：旋转参照系的角速度(关于 z 轴方向，单位是 rad/s)。适用于涡轮机械的流动计算。默认值为 0。

•*bcList*：一个按外观顺序绑定到边界集的边界条件对象数组。由于手动进行此映射可能比较困难，所以您更可倾向于使用以下字段。

•*bcDict*：一个带有指定边界条件对象的表。该表中的名称，与底层非结构化网格对象中的边界集的标记字符串相匹配。如果感兴趣，请期待 4.7 节对可用边界条件的讨论。如果一个特定的边界没有指定条件，则默认是 WallBC_WithSlip。您不必向 FluidBlock 构造函数提供任何或所有边界条件。另外，您可以在脚本的后面某处将边界条件附加到块边界。

•*tagString*：用于后处理的标签。在内部，仿真程序只关心它所持有的块数组中的块指数，然而，这个字符串将被写入 job.list 文件，应该是识别特定块的一种简便方法。

•*aFlag*：默认为 true，以便在仿真期间主动更新块。

当在非结构化网格块和其它(可能是结构化网格)块之间建立连接时，您唯一的选择是将 ExchangeBc_MappedCell 边界条件应用到块的相应边界集。

4.7 边界条件

Eilmer 代码中的边界条件是复合对象，它在每个步时更新期间的不同点对虚拟网格和界面数据产生了影响。每个边界条件对象包含四个效果列表，您可以手动指定，或者您可以选择一个预先组装好的边界条件来设置列表。预组装边界条件类的构造函数包括以下内容。

4.7.1 壁面

我们无法想象任何工程感兴趣的流动没有一个壁面，以限制气体流动区域，并与气体流动相互作用。如果在构造 FluidBlock 时没有指定任何边界条件，那么准备程序将在所有边界上应用 WallBC_WithSlip 条件。

•WallBC_WithSlip:new{label=*tagString*, group=tag}：我们想要一个没有粘性效应的实体墙。这是没有指定其他条件的**默认**边界条件。

•WallBC_NoSlip_FixedT:new{Twall=T_{wall}, label=*tagString*, group=*tag*}：在这里

我们想要粘性效应来施加到一个无滑移速度条件和一个固定的壁温。我们需要设置 config.viscous = true(参见 4.10 节的粘性效应)使该边界条件有效。

•WallBC_NoSlip_UserDefinedT:new{Twall=*fileName*, label=*tagString*, group=*tag*}：其中，我们想要一个带有任意温度剖面的墙，该温度剖面是通过文件 *fileName* 中定义的用户定义函数指定的。该文件在内部上传递给 UserDefinedInterface 边界效应，该效应需要一个名为 *interface* 的函数，如附录 E 中所述。

•WallBC_NoSlip_Adiabatic:new{label=*tagString*, group=*tag*}：在这里我们希望粘性效应对墙壁施加无滑动条件，但没有传热效应。注意，我们需要设置 config.viscous = true 使这个边界条件有效。

•WallBC_TranslatingSurface_FixedT:new{Twall=T_{wall} , v_trans=$\vec{v}_{trans}$, label=*tagString*, group=*tag*}：我们想要粘性效应施加一个指定的平移速度条件和固定的壁温。通过平移，也就是说(平面)壁面切向移动到块边界。v_trans 的值可以指定为三个已命名(x、y、z)组件的表格。我们需要设置 config.viscous = true 使边界条件完全有效。应用的一个实例就是移动板之间的 Couette 流动。

•WallBC_TranslatingSurface_Adiabatic:new{v_trans=v_{trans} , label=*tagString*, group=*tag*}：我们想把粘性效应施加到一个指定平移速度的墙壁，但这里没有传热效应。另外，需要考虑到类似于 WallBc_TranslatinqSurface_FixedT，在上面已经提到。

•WallBC_RotatingSurface_FixedT:new{Twall=T_{wall},r_omega=$\vec{\omega}$,centre=$\vec{p}$,label=*tagString*, group=*tag*}：在这里，我们想把粘性效应施加到一个指定平移速度的和一个固定的壁温条件。通过旋转，也就是说，圆柱形的墙壁是切向移动到块的边界。r_omega 和 centre 的值可以指定为三个已命名为(x、y、z)分量的表，该表给出了角速度以及旋转轴上的一个点。壁上一点的实际速度由矢量表达式$\vec{\omega} \times (\vec{r} - \vec{c})$给出，其中$\vec{r}$是壁上的点，$\vec{c}$是旋转轴上的点，$\vec{\omega}$是角速度。应用的一个实例就是滚动轴承的轴面。我们需要设置 config.viscous = true 来使这个边界条件完全有效。

•WallBC_RotatingSurface_Adiabatic:new{r_omega=$\vec{\omega}$, centre=$\vec{p}$,

label=*tagString*, group=*tag*}：在这里，我们想把粘性效应施加到墙上的一个指定平移速度的情况，但是没有传热。另外， WallBC_RotatingSurface_FixedT，也要考虑，在上面已经提到。

4.7.2 来流

通常，分析中的流域是更大的流域的一部分。为了抽象您感兴趣的较小区域，您将在抽象的流动域的边缘应用流入和流出条件，而不是壁面。流动边界条件，顾名思义，将驱使气体进入流动区域。

•InFlowBC_Supersonic:new{flowState=fs, x0=x, y0=y, z0=z, r=r, label=tagString, group=tag}：在这里，我们要指定流入流体条件 *fs*，它的每个步时都被复制到虚拟单元格。*fs* 是一个 FlowState 对象，如 4.4 节所述。默认的统一来流条件是 r=0，这可以用于自由飞行的模拟。为了能够模拟带有锥形喷嘴的风洞或激波洞所产生的流动特性，您可以指定 (x,y,z)和指定名义流动条件时的非零径向距离 r(从该原点起)。当使用这种模式时，只使用指定名义流速的 x 分量。当为每个特定的虚拟单元提供数据调用时，可以用虚拟单元的位置来确定特定的流动条件(作为名义条件的扰动)。速度的计算需要有合适的轴向和径向分量。x、y 和 z 的默认值为零。

•InFlowBC_StaticProfile:new{fileName=*fileName*, match=*matchString*, label=*tagString*, group=*tag*}：我们希望在其中指定一个流入条件，它能够以复杂的方式跨边界变化。指定的文件中包含流动条件的数据(基于每个单元格)。该文件可能是从较早的仿真中获得的，具有一个后处理选项，类似于写入文件条目的--extract-line。这里 matchString 控制着将虚拟网格与数据文件中的特定条目进行匹配，在虚拟网格点 match="xyz-to-xyz"的所有三维坐标轴中，默认的都是匹配给最近的点。其他可能的值是：

-“xyA-to-xyA”：对于二维或三维仿真，不关心 z 轴分量的位置。

-"AyA-to-AyA"：对于二维或三维仿真，只关心 y 轴分量的位置。

- "xy-to-xR"：从二维仿真的轮廓开始，考虑到虚拟单元格位置的 x 轴分量，将其映射到三维仿真的轮廓。

-"Ay-to-AR"：从二维仿真的轮廓开始，忽略虚拟单元格位置的 x 轴分量，将其映射到三维仿真的轮廓。

•InFlowBC_Transient:new{fileName=*string*, label=*tagString*, group=*tag*}：我们想要在边界处指定随时间变化的流入条件。流入条件的数据，在特定的时刻立即并假设统一流过整个边界，包含在指定的文件中。用户需要根据 FlowHistory 类中编码的预期格式(在 flowstate.d 的末尾找到)来编写这个文件。每条数据线将有以下用空格分隔的项目:时间、vel_x、vel_y、vel_z、p、T、质量分数、T_{modes}(如果有的话)。

•InFlowBC_ConstFlux:new{flowState=fs, x0=x, y0=y, z0=z, r=r, label=tagString, group=tag}：其中我们要直接指定质量、动量和能量在边界面上的通量。通量是根据所提供的流动条件计算出来的。有关虚拟源的说明，请参考 InFlowBC_Supersonic

•InFlowBC_ShockFitting:new{flowState=fs,x0=x,y0=y,z0=z, r=r, label=tagString, group=tag}：其中我们希望流入边界是一个弓形激波的位置。通过边界的通量用提供的流动条件计算，设置边界速度来捕捉激波。有关虚拟源的说明，请参考 InFlowBC_Supersonic。注意，我们需要设置 config.moving_grid = true，并为移动网格选择合适的气体动态更新方案。并将所有具有冲击拟合边界的块作为单个 FBArray 的一部分。

•InFlowBC_FromStagnation:new{stagnationState=*fs*,

fileName=*string*,

direction_type="normal",

direction_x=1.0, direction_y=0.0, direction_z=0.0,

alpha=0.0, beta=0.0,

mass_flux=0.0, relax_factor=0.10

label=*tagString*, group=*tag*}

在这里我们想要一个指定滞止压力和滞止温度的亚声速流，以及在边界处的一个速度方向。(注意，很多字段都显示了它们的默认值，因此不需要指定它们。)在每个时间步长的作用下，块边界上的平均局部压力与滞止条件都用来计算流体流动条件。这取决于 direction_type 的值，计算速度的方向可设置为：

– "normal"直到局部边界。

– "uniform"与方向向量对齐，其分量为 direction_x、direction_y 和

direction_z。

– "radial"径向通过一个圆柱形表面，使用流动角 alpha 和 beta，或？

– "axial"轴向通过一个圆形表面，使用相同的流动角。

对于质量流量指定为非零值的情况，计算当前流过块表面的质量流量(单位面积)，并增加名义滞止压力，使通过边界的质量流量缩减到指定值。注意，当我们选择非零质量流量时，我们不需要控制滞止压力。这将调整给所需的质量流量。relax_factor 的值调整了反馈机制的收敛速度。注意，对于多温度模型数值仿真，所有的温度都设置为与旋转温度相同。这通常是一个合理的物理近似法，因为这个边界条件通常用来模拟容器的流入，而容器中滞止的气流在相同温度下有足够的时间达到平衡。实现这种边界条件在时间上可能不精确，尤其在大冲击波通过边界时，但它在稳态极限下往往能很好地工作。

当质量流量 mass_flux 为 0 且文件名 fileName 保留为默认空字符串时，指定的 FlowState，即 fs 被当作滞止条件的常量。这可能被用户自定义函数修改，如果 fileName 是一个非空字符串，则用 stagnationPT 名称给一个包含 Lua 脚本名称的函数。每个边界条件应用程序中，这个函数接收一个数据表(包括当前仿真时间)，并返回值给滞止压力和温度。下面是一个简单的例子：

```
function stagnationPT(args)
   -- print("t=", args.t)
   p0 = 500.0e3 -- Pascals
   T0 = 300.0 -- Kelvin
   return p0, T0
end
```

这样做可以让用户将滞止压力编程为更有趣的时间函数。

4.7.3 出流

如果您有一个流入边界条件，您可能需要一个或多个流出边界条件，以避免你的流动域只是一个累加器。当截断一个更大的流域来创建模拟流域时，选择一个位置，使流出边界条件与正在进行测量的流域部分尽可能远。

•OutFlowBC_Simple:new{label=*tagString*,group=*tag*}是

OutFlowBC_SimpleFlux:new{}的别名。

• OutFlowBC_SimpleFlux:new{label=*tagString*, group=*tag*}：我们想要一个(大部分情况)超声速的流出条件。它也应该与亚声速流出一起工作，但是需要记

住您是故意忽略那些可能从您已经截断的真实(物理)区域传播到域的信息。流出流量是由内边界单元格的流动状态决定的。如果单元格内的速度试图产生质量流量，流量的计算就转换到不渗透的墙壁里。

•OutFlowBC_SimpleExtrapolate:new{xOrder=0, label=*tagString*, group=*tag*}：通常我们需要超声速的流出条件。每次执行一个步长，流动数据被有效地从边界内或边界外复制(xOrder=0)到或线性推算(xOrder= 1)到边界外的虚拟单元格。在亚声速流中，这可能导致物理上的无效行为。如果您遇到奇怪的流动状况，它们看起来像是从这个边界上开始，并向上传播到您的流域，那就试着扩展你的模拟流域，这样您最终会得到一个出现刺激的事情的流出边界。

•OutFlowBC_FixedP:new{p_outside=1.0e5, label=*tagString*,group=*tag*}：这里我们想要类似 OutFlowBC_simple 的压力，但有指定的背压。这可以类比于真空泵，抽走边界处的气体以保持虚拟单元格内的固定压力。

•OutFlowBC_FixedPT:new{p_outside=1.0e5, T_outside=300.0 label=*tagString*, group=*tag*}：类似于上面的 OutE1owBC_FixedP，但也设置了虚拟单元格中的温度。

4.7.4 内部块的交换

在有多个单元块的仿真中，流动求解器主要是独立地处理单元块。在更新的每个阶段，计算结果通过交换跨块边界的流动数据在块间边界拼接在一起。

•ExchangeBC_FullFace:new{otherBlock-nil, otherFace=nil, orientation=-1, reorient_vector_quantity =false，

Rmatrix={1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0}， label=*tagString*, group=*tag*}

通常情况下，庆谨慎使用这种边界条件。通过调用 identifyBlockConnections 函数，将结构化块与另一个块连接起来，然后边界就很清晰地对齐了；但是，当你想要流动从一个块表面到另一个块的最低点，并且这些块不存在几何对齐时，可以手动操作。当需要将流动向量从另一个边界复制到这个边界时，可以提供一个非单位变换矩阵 Rmatrix。注意，此边界条件仅适用于结构化网格块。如果要连接的一个或两个块基于非结构化网格的块，则需要使用交换边界条件的下 MappedCel1 风格。

• ExchangeBC_MappedCell:new{transform_position=false,

c0=Vector3:new{x=0.0,y=0.0,z=0.0},

n=Vector3:new{x=0.0,y=0.0,z=1.0},

alpha=0.0, delta=Vector3:new{x=0.0,y=0.0,z=0.0},

list_mapped_cells=false, reorient_vector_quantities=false,

Rmatrix={1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0},

label=*tagString*, group=*tag*}

类似于 ExchangeBC_FullFace 边界条件，但有目标(虚拟)单元格到源单元格位置的映射。它允许我们将边界连接在一起，即使单元格没有对齐，也是一对一的连接在一起。通过获取虚拟单元格的位置，计算在 c0 点处固体围绕 n 轴旋转 alpha 角的值，然后添加一个位移 delta，就可以计算源单元格的位置。这将适应于一般的刚体转换。

4.7.5 用户自定义

这就是我们的"出狱"边界条件。它们允许您做任何想做的事情，因为缺乏约束，所以有点难以描述。有关更完整的描述，请参阅附录 E。

• UserDefinedGhostCellBC:new{fileName=*string*,label=*tagString*, group=*tag*}

允许用户在运行时定义虚拟单元格流动属性或界面流量。这是通过用户定义的一组函数完成的，这些函数用 Lua 编程语言编写，并在指定的文件中提供。有关的更完整描述，请参见附录 E.1.1。

•UserDefinedFluxBC:new{fileName=*string*, funcName=*string*,label=*tagString*, group=*tag*}允许用户在运行时定义接口的对流流量。这是通过用户定义的函数完成的，这个函数使用 Lua 编程语言编写，并在指定的文件中提供。如果用户没有指定函数名，则默认使用 convectiveFlux 这个名称。有关更完整的描述，请参见附录 E.1.2。

• ExchangeBC_FullFacePlusUDF:new{otherBlock=nil, otherFace=nil,

orientation=-1, reorient_vector_quantities=false,

Rmatrix={1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0},

fileName=string, label=tagString, group=tag}:

在某些情况下，你可能会有条件地想要交换块边界数据或完全做其他事情。这个边界条件允许首先进行 FullFace 数据交换，然后调用您的用户定义函数

(如 UserDefinedGhostCellBC)有条件地覆盖数据。这是一种方便的方法来实现膜片模型的冲击隧道模拟。注意，这个边界条件可以跨越 MPI 任务，但仅用于结构化网格块。

注意，所有边界条件都具有可选的标签 label 和空默认值组 group 的字段。这些可以用来象征性地对边界表面进行分组。

4.7.6 将边界条件绑定到块面

在第 4.6.1 节中，可以将边界条件指定为传递给 FluidBlock 构造函数对象的 BoundaryCondition 对象表。这通常是首选的做法。另外，在块构造之后，可以将 BoundaryCondition 对象单独赋值为 bcList 属性的元素。例如：

```
blk_0:bcList[west] = InFlowBC_Supersonic:new{flowState=fs}
blk_1:bcList[east] = OutFlowBC_Simple:new{}
```

当使用 FluidBlockArray 时，使用边界条件的 bcList 类型格式会更方便，因为这将考虑自动细分块上的所有边界条件。

同样，基于非结构化网格的 FluidBlock 对象也可以这样做，但是您必须使用边界集的数值指数。在构建块时，以带有命名条目表的形式提供边界条件对象，是最简单的方法。有关设置这种表的指南，请参阅第 5.3 节中的示例。

4.8 特殊区域

特殊区域的边界条件可以在流域中定义为矩形(二维)或正六面体(三维)，它们由两个斜对面角点(po 和 pl)指定。现有代码中的三类特殊情况为：反应、着火和湍流。

例如，我们可以指定 Reactionzone: new(po=$\vec{a}$,p1=$\vec{b}$})，其中区域的角点由 Vector3 值$\vec{a}$和$\vec{b}$给出。在具有活性反应体系的流动中，这种类型的区域使选择性地允许或不允许反应进行成为可能。如果一个单元格的中心位于反应区内，则允许进行有限速率的化学反应，否则将保持组分不变。保持物质组分的恒定，将会有效地“冻结”反应。在二维仿真中，p0 对应图 4.1 中的 p00，而 pl 对应面对角上的 p11。在三维仿真中，p0 对应于图 4.2 中的 p0，而 pl 对应于在六面体块中对

角相对顶点处的 p6。如果没有指定反应区域，并且反应方案是有效的，那么整个流场都允许发生反应。这种区域的一个应用例子就是模拟研究冲压式发动机[16]，其中整个模拟区域的流入包括燃料混合物，但我们希望反应只在冲压发动机的燃烧室内进行。

一种触发化学反应的有效方法就是使用 IgnitionZone:new{p0=$\vec{a}$, p1=$\vec{b}$, T=T_{ig} } 命令，其中温度 T_{ig} 控制化学反应的反应速率，而不影响流场中的气体温度。速率控制的温度仅用于评价着火区域物理范围内的化学反应速率。通过指定给 config.ignition_time_start 以及 config.ignition_time_stop 一个非零值，可以在时间上限制此区域的效果。当反应区处于活性状态时，反应速率将发生改变。速率控制的温度通常设置为一个表面膨胀的值，以促进点火。例如，2000 K 的值可以有效地点燃甲烷/空气混合物的某些成分。

此外，在进行湍流仿真时，可以使用 TurbulentZone:new{ p0=$\vec{a}$, p1=$\vec{b}$}来衡量湍流效应。湍流模型(k-ω 模型)在整个流动过程中都是活跃的，但在任何确定的湍流区外，它对流场的影响都被掩盖了。这是通过代码来完成的，代码将湍流粘度和电导率设置为均为零的有限体积单元格，落在所有定义为 TurbulentZone 区域外。如果没有这样的定义区域，则整个流场的湍流粘度允许为非零值。

4.9 记录点

历史点是流场中的一个位置，在这个位置上，数据以与整个流场不同(可能更高)的频率写入文件。历史点的文件可以在 hist/子目录中找到，并由相关的块和单元指数识别。历史点可以通过笛卡尔坐标系来定位，调用的函数为：

setHistoryPoint {x=c, y=y, z=0.0}

z 坐标的默认值为 0。或者，可以通过它的块和单元格指数来定位这个点：

setHistoryPoint {ib=b, i=i, j=j, k=0}

config.dt_history 控制写入历史数据的频率，下面有关配置参数的一节有描述。

[16] 参见示例/eilmer/2D/ramjet-student-design 中的源代码存储库。

4.10 仿真配置及控制参数

为了配置和控制仿真结果，还可以设置多个参数。这些参数主要收集到 config 表[17]中，该表可以访问用户的输入脚本。按主题分组，可能的属性及其默认值包括[18]：

4.10.1 几何

• dimensions=2：几何维数(2 维或 3 维)。

• axisymmetric=false：如果设为真，则二维轴对称几何结构将以 x 轴为对称轴。默认的是二维坐标系下的平面几何。几何图形应该构建在(可能包括)x 轴之上，因此，y 轴是纵坐标。

4.10.2 时间步长

代码试图自动调整时间步长的大小，以便所有单元格的数字集成过程保持稳定。有时这是非常困难的，您将不得不更改以下的一个或多个参数：

• cfl_value=0.5‡：CFL 数是时间步长与最快信号通过单元格的时间的比值。时间步长调整过程试图将整个仿真的时间步长设置为一个值，使任何单元格的最大 CFL 数都位于 cfl_value 参数中。如果您在仿真中遇到许多流场突然变化的问题，那么减小 cfl_value 参数的大小可能会有所帮助。

• gasdynamic_update_scheme=predictor-corrector：选项包括：

'euler', 'pc', 'predictor-corrector', 'midpoint','classic-rk3',

'tvd-rk3', 'denman-rk3', 'moving-grid-1-stage','moving-grid-2-stage'。

注意"pc"相当于'predictor-corrector'。如果您想要时间精确的计算结果，请使用两步或三步步长模式，否则的话，即使使用计算资源耗费会更少的欧拉步长，您可能会得到的精度较少，并且对于相同的 CFL 值，代码也不会呈现出那么鲁棒性。例如，在 CFL=0.85 的情况下，对于欧拉步长，Sod 激波管例子中的激波前沿是相当嘈杂的，但是对于相同 CFL 值的两步或三步步长方案中，任何一种方案都是相当光滑平整的。在 CFL= 1.0 时，中点和 predictor-corrector 预测-校正方案产生了明显的激波，而 rk3 方案在 CFL= 1.2 时仍然

[17] config 表是 Globalconfig 类的视图，该类在模拟代码 D 语言部分定义。虽然这里讨论了许多属性，但是要获得完整的属性列表，请参阅该类的源代码。您可以在 src/eilmer/globalconfig.d 文件中找到它。

[18] 控制文件中存储的属性用‡符号表示。其余的放到配置文件中。

表现得很好。注意，在拼写方案名称时可以使用破折号或下划线。

• fixed_time_step=false‡：正常情况下，我们允许由单元格条件和 CFL 数值确定时间步长。设置 fixed_time_step=true 将强制时间步长从 dt_init 开始保持不变。

• dt_init=1.0e-3‡：虽然时间步长是自动计算的，但在某些情况下，这个过程可能没法选择足够小的值，以至于不能稳定地启动仿真。对于初始步长，用户可以通过为 dt_init 分配一个适当的小值来覆盖时间步长的计算值。这将成为仿真过程的最初几个步骤所使用的初始时间步长(以秒为单位)。要注意设置一个足够小的值，使时间步长足够稳定。由于时间步长在流域的所有部分都是同步的，所以这个时间步长的大小应该小于信号(压力波)通过流域中任何单元格最小时间的一半。如果您确定您的几何和边界描述是正确的，但您的仿真失败，没有明确的原因，试着将初始时间步长设置为一个非常小的值。对于某些精细网格上的粘性高超声速流动仿真，要求时间步长小到纳秒量级是很常见的。

• dt_max=1.0e-3‡：最大允许时间步长(以秒为单位)。有时，特别是当强源项起作用时，基于 CFL 时间步长的确定并没有适当地限制允许时间步长的大小。这个参数就允许用户直接限制时间步长的最大值。

• viscous_signal_factor=1.0‡：在默认情况下，在时间步长计算中信号计算将考虑全局粘性效应。研究表明，在高分辨率粘性计算中，可能不需要全局粘性效应来保证计算的稳定性。0.0 的值将完全抑制对信号速度计算的粘性影响，但您可能会以不稳定的步长结束。如果您得到一个更大的时间步长，并保持一个稳定的仿真，这会是一个“不断试值并观察结果”的问题。

• stringent_cfl=false‡：结构化网格的默认操作是在每个指数方向上使用不同的单元格宽度。在时间步长检查中，设置 stringent_cfl=true，则将使用最小的跨单元格距离。

• cfl_count=10‡：检查时间步长之间的时间步长数量。这个检查是很耗时的，所以我们不想做得太频繁，但是，我们必须小心流动突然不会发展演变以及时间步长变得不稳定问题。

• max_time=1.0e-3‡：当达到这个时间值时仿真将终止。

• max_step=100‡：仿真将在达到这个时间值时终止。您几乎肯定会想要使用较大的值，然而，较小的值是测试仿真启动过程的好方法，以便查看一切是否正常。

• dt_plot=1.0e-3‡：当这段模拟时间结束后，整个流动计算结果将被写入磁盘，然后再次经过相同的仿真时间增量结束的过程。

• dt_history=1.0e-3‡：每次仿真时间增量结束时，历史点数据将被重复写入磁盘。为了获得历史数据，您还需要指定一个或多个历史点。

4.10.3 块匹配

• block_marching=false：正常的时间迭代是在所有块上同时进行，然而，这样的时间步长计算量可能非常大。设置 block_marching=true 可以按照时间顺序进行积分，这样在任何时刻，就只有两个块被记录下来。i 方向是步长方向，是假定的主导(超声速)流动方向。这些块假设在一个常规数组中，该数组在整个流域的 j-和 k-方向上都有固定数量的块。

• nib=1, njb=1, nkb=1：每个指数方向的块数。为了充分地进行块匹配，应将 nib 设置为相当大的数字。因为假设块数组是常规的，所以不可能有非常复杂的几何形状。简单的导管、喷嘴和平板都是理想的应用物品。正如在示例中看到的，使用单个调用将整个流域定义为 FluidBlockArray，可能会比较方便。

• propagate_inflow_data=false：默认情况下，从仿真准备阶段设置的初始气体状态开始，在每组块中开始积分。通过使用下游(东边界)流动状态初始化后的块切片，就有很大的优势来集成第一个块切片。在切片的积分过程开始之前，设置的 propagate_inflow_data=true 可以通过每个新块切片来传播这些数据。

• save_intermediate_results=false：通常，在遍历所有块切片之后，只需要一组数据结结果文件。有时，在调试一个较麻烦的计算时，在每个块切片的时间积分过程之后，再编写一个解决方案可能是很有用的。将这个参数设置为 true，以得到这些写入的中间计算结果。

4.10.4 空间重构/插值

• interpolation_order=2：在应用通量计算器之前，使用高阶重建。设置

interpolation_order=1 不会重建内单元格流动属性。

• apply_limiter=true：默认情况下，我们对流场重构应用一个限制器。

• extrema_clipping=true：默认情况下，在每个标量场重构结束时，我们都要进行极值剪切。设置 extrema_clipping=false 抑制剪切。

• thermo_interpolator="rhou"：字符串选择一组插值变量用于插值，选项有"rhou", "rhop", "rhoT"以及“pT”。

4.10.5 流量计算器

• flux_calculator="adaptive_hanel_ausmdv"：选择通量计算器的型号。选项有：

– "efm"：Pullin 和 Macrossan [15，16]提出了一个非常经济且易于扩散的方案。对于大多数高超声速流动，它太扩散了，不能用于整个流场，但它与 AUSMDV 配合得很好，特别是在钝体流动的激波层中。

– "ausmdv"：一个低扩散性超声速流动的全方位格式[17]。

– "adaptive_efm_ausmdv"：混合[18]了用于远离激波区域的低耗散 AUSMDV 格式和用于靠近激波单元格界面更加扩散的 EFM 格式。它在高超声速流动中看起来工作很可靠，这种流动是一种混合了强激波、亚声速和超声速流的混合区域。ompression_tolerance 和 shear_tolerance 参数控制混合，如下面描述。

– "ausm_plus_up"：实现了 Ref.[19]的描述。对所有速度方法来讲，它应该是准确的和鲁棒性的。这是针对在非常低马赫数流动下的一个通量计算器选择，其中流体的特性达到了不可压缩的极限。为了得到最好的结果，您应该设置 M_inf 的值。

– "hlle"：Harten-Lax-vanLeer-Einfeldt (HLLE)格式。这有些消耗资源，但是唯一可用于 MHD 项目的方案。

– "adaptive_hlle_ausmdv"：它针对"adaptive_efm_ausmdv"，但在耗散格式中是 HLLE 通量计算器。

– "hanel"：这是 Hanel-Schwane-Seider 格式，摘自他们 1987 年的论文。它也是耗散格式的，也比我们的 EFM 格式表现得要好。

– "adaptive_hanel_ausmdv"：它针对"adaptive_efm_ausmdv"，但在耗散格式中是 Hanel-Schwane-Seider 通量计算器。

– "roe"：Phil Roe 的经典线性通量计算器。

– "adaptive_hlle_roe"：混合了 Roe 的低耗散格式以及更耗散的 HLLE 通量计算器。

默认的自适应格式是一个很好的全方位格式，它使用远离激波的 AUSMDV 和接近激波的 Hanel-Schwane-Seider 通量计算器。

• compression_tolerance=-0.30：在单元格界面上触发激波点探测器的相对速度变化值(不受局部声速影响)。负值表示压缩。当使用自适应通量计算器并触发激波探测器时，将使用更消耗的通量计算代替默认的低耗散计算。对于 Sod 激波管和钝体圆锥无粘流动仿真，-0.05 的值似乎是可以的，但是对于粘性边界层，需要更高的数值，在边界层区域不要有太多的数值耗散，这一点也是很重要的。

• shear_tolerance=0.20：在单元格界面上的相对切向速度变化值(通过局部声速归一化),即使激波检测器表明在自适应通量计算器中应该使用高耗散格式，这个界面也会抑制高耗散通量计算器的使用。实验设置默认值为 0.20，以得到钝体滞止区域的光滑激波。当剪切区域也存在时，可能需要一个更小的值(比如 0.05)来得到强扩张流动。

• M_inf=0.01：自由流动的代表性马赫数。使用 ausm_plus_up 通量计算器。

4.10.6 粘性效应

• viscous=false：如果设置为 true，则仿真中应包含粘性效应。

• separate_update_for_viscous_terms=false：如果设置为 true，则粘性输运项的更新将分别对对流项进行。默认情况下，更新是在气体动力更新过程中一起完成的。

• viscous_delay=0.0：在应用粘性术语之前等待的时间(以秒为单位)。这可能会在尝试开始钝体仿真时派上用场。

• viscous_factor_increment=0.01：当 t>viscous_delay 时，粘性效应的单位时间步长增量。

• mass_diffusion_model="none":在多物种、粘性、层流计算中控制单个种类的分子扩散。唯一有效的模型是“fiscs_first_law”，它规定了扩散通量的形式，但为扩散系数留下了不同的可能性。注意，在湍流模拟中，这个参数被忽略，

种类扩散基于 turbulence_schmidt_number。这是因为湍流扩散通常比层流扩散大得多，因此忽略后者可以节省一些计算时间。

• diffusion_coefficient_type="none":在 mass_diffusion_model="ficks_first_law" 仿真中用来计算扩散系数的控制方程。最简单的选项是 constant_lewis_number，它将与同种类气体一起工作。可以使用 lewis_number 来设置 Lewis 数的大小。这需要在气体模型中指定每个物种的刘易斯数。更灵活的选项是 species_specific_lewis_numbers，它需要在气体模型中指定每种种类的 Lewis 数。热完全气体模型就是一个具有这种信息的气体模型的例子。具有最高保真度扩散系数的模型是 binary_diffusion，它要求气体模型计算每个种类相对于其他种类的扩散系数，并将它们一起平均，通常使用碰撞积分。同样，热完全气体模型是具有这种能力的气体模型的一个例子，尽管双温度空气模型也具备这种计算能力。

• lewis_number=1.0: 在一个模拟中使用 constant_lewis_number 设置热传输和扩散质量传输之间的比例。

• turbulence_model="none"：指定使用哪个模型的字符串。选项有："none"、"k_omega"、"baldwin_lomax"。

• turbulence_prandtl_number=0.89

• turbulence_schmidt_number=0.75

• max_mu_t_factor=300：湍流粘度限制为层流粘度乘以该系数。

• transient_mu_t_factor=1.0

4.10.7 热化学

• reacting=false：设置为 true 以活化有限速率的化学反应。

• reactions_file="chemistry.lua"：反应方案配置的文件名。

• reaction_time_delay=0.0：允许有限速率反应开始的时间。

• T_frozen=300.0：低于反应停止时的温度(以热力学温度 K 表示)。默认值为 300.0，因为大多数的反应方案在该温度以上都是有效的，但是，如果您想把它调高或调低，您需要有充分的理由。

4.10.8 其它参数

• title="Eilmer simulation"：可能出现在许多地方的标题字符串。例如，在后

期处理阶段制作的图像中。

• adjust_invalid_cell_data=false：通常，您希望流动求解程序为您的流动提供最佳估计值，但在某些流动情况下，流动求解程序将无法计算物理上有效的流动数据。如果您处理了一个复杂的流动情况，并且准备修改一些单元格，那么将该参数设置为 true，将 max_invalid_cells 设置为非零值。

• max_invalid_cells=0：在解码守恒量时所容纳劣质单元格的最大数量。如果超过这个数字，仿真将停止。

• report_invalid_cells=true：如果您不得不粗略制作单元格，那么您可能希望了解它们，当然，希望您不想知道。将此参数设置为 false，以使粗略制作的劣质单元格报告消失。

• apply_bcs_in_parallel=true：这将是最快的计算，但是，对于某些边界条件，如激波拟合，则需要跨块协作，因此如果并行应用，就会出现竞争情况。如果您的仿真有这样的边界条件，需要把这个参数设为 false，以吻合仿真安全高于速度的原则。

• udf_source_terms=false：设置为 true，以应用 Lua 文件提供的用户自定义源条款。

• udf_source_terms_file="dummy-source-terms.txt"：用户自定义源条款 Lua 文件的名称。

• print_count=20‡：将状态信息显示到控制台期间的时间步长。

• control_count=10：重新解析 job.control 文件期间的时间步长。如果已编辑 job.control 控件，就在重新解析后使用新值。

• MHD=false：设置为 true，使 MHD 在物理上活跃。

4.11 输入脚本布局的标记

输入脚本的目标是定义一个或多个具有合适边界条件和初始气体状态的 FluidBlock 对象。构造对象和设置配置变量的顺序在某种程度上由构造函数为每个对象类所需要的输入来定义。要定义 FluidBlock，需要导入或构造几何元素的 Grid 和 GasState。要定义 GasState，您需要设置主 GasModel。因此，典型的输入脚本将设置 GasModel，然后继续定义一个或多个 GasState 对象。几何构造和网

格对象的创建或导入可以独立于 FlowState 构造完成，所以这个阶段是在 FlowState 规范之前还是之后完成并不重要。之后是 FluidBlock 构造。

大多数 config 变量设置仅供在模拟运行时使用。如果您想要一个非默认值的值，可以在输入脚本中的任何点设置它们。但是，在输入脚本中有一些设置上下文的 config 变量非常重要。这包括 dimensions、viscous、grid_motion 和 turbulence_model。如果您想要一个不同于默认值的值，请确保在脚本的早期设置它。

有许多配置变量，它们的最终参考是流动求解器的源代码。不要害怕打开文件 globalconfig.d 并在 GlobalConfig 类中浏览这些变量的定义。源代码中嵌入的文档注释在本文中没有出现在本指南中。一般的规则是，只有在你的输入脚本中设置一个值而不是默认值，或者如果你想通过脚本显式地记录这个设置。在输入脚本中将许多配置变量赋给它们的默认值只会增加混乱。

4.12 MPI 仿真

一旦您成功地完成了最初的几次仿真，您对计算仿真的野心就可能会增长，并且 MPI 代码风格的使用，即对 e4mpi，也会很有趣。MPI 代码需要的惟一额外配置是将 FluidBlocks 分配给 MPI 任务。如果您不关心细节，默认的安排是将每个 FluidBlock 分配给它自己的 MPI 任务。如果您想要一些不同的东西，可能是为了更舒适地适应工作站的功能，您可以通过引用下面的语句来明确地安排分布：

```
mpiTasks = mpiDistributeBlocks{ntasks=3, dist="load-balance",
                               preassign={[0]=1}}
```

在这里，我们已经指定将我们希望的 FluidBlocks 分配给 3 个 MPI 任务，将 FlowBlock[0]分配给 MPI task 1。剩下的 FlowBlocks 分配算法选项是"round-robin"和"load-balance"，默认是"load-balance"。这个对 mpiDistributeBlocks 的调用应该在您的输入脚本定义了所有 FluidBlocks 之后执行。

对于块匹配计算，块的分布应该是这样的:沿着推进方向(i-index 方向)的条带中的块应该被分配给 MPI 任务。如果你已经构造了块 withfarray:new，则存在一个调用:

```
mpiDistributeFBArray{fba=my_fba, ntasks=njb * nkb}
```

这将正确地分配 my_fba 的块。如果你不直接调用函数，准备程序将为你安排任务分配，即上面显示的默认任务数。当然，您需要知道这个默认值，以便在随后使用 mpirun 启动模拟时可以指定正确的任务数量。您可以指定更少的 MPI 任务，但是数量应该是这样的:一个 j,k-slice 块应该整齐地分布在这个数量的任务上。在二维凹凸通道的算例模拟中，流域由 192 个块组成，以 48×4×1 为阵。对于这种情况，设置 ntasks=2 或 4 是合适的。

5.更多仿真例子

有了对代码正常工作的一些信心，并且了解了教程示例(第 3 章)中显示的手动后处理安排，您就可以尝试模拟更“真实”的流动了。下面几节将介绍两个要求更高的示例。

第一种是平板，在超声速流动中，有一个层流边界层。当斜激波与边界层相互作用时，就会发生有趣的现象。这是一个具有简单理想气体热化学模型和简单流域的二维流动仿真。

第二个例子是高焓气体在环面上的钝体形流动。反应氮的热化学模型更加复杂，三维域的描述需要更多的 Lua 代码来建立。

5.1 斜激波边界层相互作用

这是一个引入粘性效应的例子，但保留了一个非常简单的流动边界层几何布置。它很容易建模，但会立即显示要求增加“流动保真度”的计算需求。考虑理想空气以 2 倍马赫数流过平板上的流动，如下图 5.1 所示。这张流动图像是在 MIT 一个连续气流风洞中拍摄的实验活动[20]的一部分。图像中气流是从左到右流动。下边界是感兴趣的边界层平面，这里有一个粘性相互作用的激波，从板块的尖锐边(图像的左下方)传播并穿过气流。在测试区域的上表面形成了另一个小冲击角平面。这个产生激波的平板前缘在视野之外，但是在图像的左上角可以看到产生的激波正在进入视野，并在距前缘约 49mm 处从底板反射。激波反射导致整个相互作用区域的总压比为 1.4。可以看到平板上的边界层到反射激波的交点处，边界层越来越厚，然后到相互作用点处才减弱。选择压力比为 1.4 的情况进行仿真，是因为如原始报告[20]所述，剪切应力数据表明边界层在相互作用后一直维持着层流。

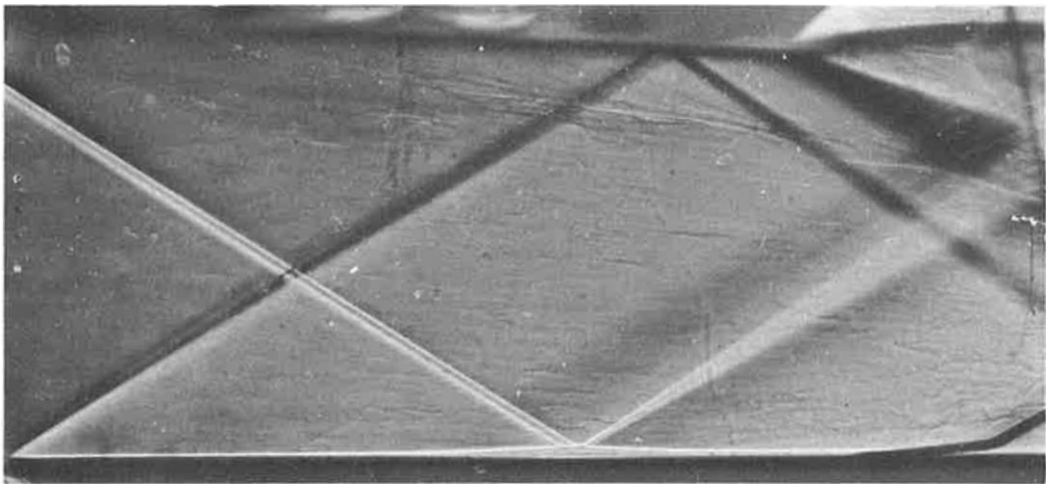


图 5.1:从参考文献[20]中图 6b 获得的平面上马赫数为 2 的纹影图像。流动是从左到右，平板的前缘接近图像的左下角。激波发生器的前缘在图像之外，大概在左上角附近。激波边界层相互作用的有趣区域大致位于图片底部的平板中间。

虽然通过简单的理论可以很好地预测平板上层流可压缩流动边界层的状况，但是相交激波的加入使分析更加困难。流动的复杂性增加了，而定义流动的几何结构仍然非常简单。打个比方，这是一个用 CFD 锤敲开的好坚果。

5.1.1 组织仿真

要开始仿真，需要准备输入文件，来建立一个简单的空气模型。正如我们在

教程示例中所做的，我们可以将此文件命名为 ideal-air.inp，它应该有两行：

```
1 model = "IdealGas"
2 species = {'air'}
```

我们现在使用输入文件来准备实际的气体模型定义文件，命令如下：

```
$ prep-gas ideal-air.inp ideal-air-gas-model.lua
```

其中$字符表示系统的命令提示符。如果结果成功，将生成 Lua 文件 ideal-air-gas-model.lua。当然，在您自己的工作中，应该选择适合于描述当前问题的文件名。重复使用这些名称是可以的，只要它们是适当的并且没有误导。有了组织好的气体模型，接下来就是定义流动和流域的更大任务。

图 5.2 显示了仿真建模的区域。布置了仪器的平板(位于下边界，标记为 ADIABATIC)将从 x=0 处开始，即使实际平板在实验中延长了 8 英寸，但在实验流态图中看起来也像是被截短了。即使真实的激波发生器在其前缘有边界层和相关的粘性相互作用，激波发生器平板(沿斜向上边界)也设置为理想的无粘壁。对于激波发生器表面来说，应用滑移壁面边界条件非常方便。这让我们能够利用理想气体的斜激波关系来估计在反射激波中指定压力上升的偏转角。

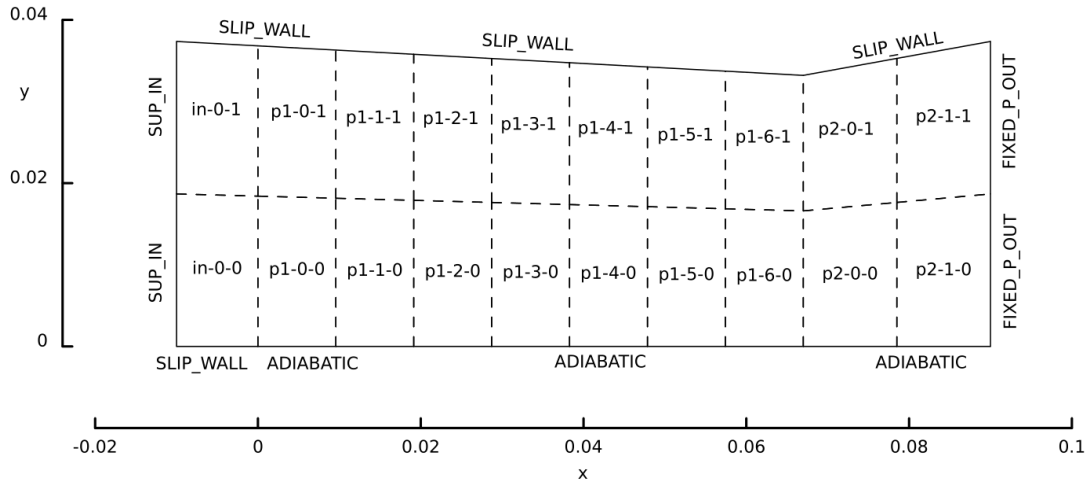


图 5.2:激波与层流边界层相互作用的模拟区域示意图。

看一下这张照片(图 5.1)，将激波发生器板的表面外推到它与激波相交的地方，这给了激波发生器板的前缘一个位置，该位置在所述仪器化平板前缘的上游。因为使用盒状流域比较方便，所以我们在模拟的流域中包含了从激波发生器板尖端到仪表化平板尖端的区域。我们沿着这部分流域的下边界，即 x<0，使用一个

SLIP_WALL 边界条件。

利用仿真程序中内置的理想气体流动函数，下面的脚本计算了通过入射和反射斜激波的组合压力升高值，结果为 1.4。经过一或两分钟的反复试验，激波发生器的偏角估计值为 3.090°。

```
-- double-oblique-shock.lua
-- Estimate pressure rise across a reflected oblique shock.
-- PJ, 01-May-2013, 2016-11-01 for Lua version
-- $ e4shared --custom-post --script-file=double-oblique-shock.lua
--
print("Begin...")
M1 = 2.0
p1 = 1.0
g = 1.4
print("First shock:")
delta1 = 3.09*math.pi/180.0
beta1 = idealgasflow.beta_obl(M1,delta1,g)
p2 = idealgasflow.p2_p1_obl(M1,beta1,g)
M2 = idealgasflow.M2_obl(M1,beta1,delta1,g)
print(" beta1=", beta1, "p2=", p2, "M2=", M2)
--
print("Reflected shock:")
delta2 = delta1
beta2 = idealgasflow.beta_obl(M2,delta2,g)
p3 = p2*idealgasflow.p2_p1_obl(M2,beta2,g)
M3 = idealgasflow.M2_obl(M2,beta2,delta2,g)
print(" beta2=", beta2, "p3=", p3, "M3=", M3)
print("Done.")
```

注意，运行脚本的命令实际上以自定义后处理模式来启动复制程序，并指定运行哪个 Lua 脚本。$ e4shared --custom-post --script-file=double-oblique-shock.lua

写入控制台的结果如下：

```
First shock:
   beta1= 0.56869987213562      p2= 1.1867698723259      M2= 1.8891863108079
Reflected shock:
   beta2= 0.60486005952261      p3= 1.4001003974295      M3= 1.7811889520851
```

5.1.2 输入脚本(.lua)

输入脚本中，在流动图像和相关压力以及表面摩擦图中确定的激波位置处，对流域和平板的几何尺寸简单地进行了缩放。将流域建模为一个带有直线边界段

的盒子形状，尽管几何结构特别简单，但我们使用了三个 FluidBlockArray 进行调用，将该区域分割为 20 个单独的块，如图 5.2 所示。之所以这样做，是为了将这些块分配给多核机器的几个处理器，这样我们就不必等待很长时间来进行运行仿真。

在原始报告[20]中，使用 $Re_{x\text{-}shock}$=2.96×10^5 的数据时，图 6b 的自由流动条件预测为：p_∞=6.205kPa, T_∞=164.4K，以及 u_∞= 514 m/s，理想空气为 R_{gas}=287J/kg.K 以及 γ= 1.4。

```
-- swbli.lua
-- Anand V, 10-October-2015 and Peter J, 2016-11-02
-- Model of Hakkinen et al's 1959 experiment.

config.title = "Shock Wave Boundary Layer Interaction"
print(config.title)
config.dimensions = 2

-- Flow conditions to match those of Figure 6: pf/p0=1.4, Re_shock=2.96e5
p_inf = 6205.0 -- Pa
u_inf = 514.0 -- m/s
T_inf = 164.4 -- degree K

nsp, nmodes = setGasModel('ideal-air-gas-model.lua')
print("GasModel set to ideal air. nsp= ", nsp, " nmodes= ", nmodes)
inflow = FlowState:new{p=p_inf, velx=u_inf, T=T_inf}

-- Flow domain.
--
-- y
-- ^ a1---b1---c1---d1    Shock generator
-- |   |    |      |    |
-- |   | 0  | 1    | 2  |   patches
-- |   |    |      |    |
-- 0 a0---b0---c0---d0    Flat plate with boundary layer
--
--      0---> x
mm = 1.0e-3 -- metres per mm
-- Leading edge of shock generator and inlet to the flow domain.
L1 = 10.0 * mm; H1 = 37.36 * mm
a0 = Vector3:new{x=-L1, y=0.0}
a1 = a0+Vector3:new{x=0.0,y=H1}
-- Angle of inviscid shock generator.
alpha = 3.09 * math.pi/180.0
```

```
35 tan_alpha = math.tan(alpha)
36 -- Start of flat plate with boundary layer.
37 b0 = Vector3:new{x=0.0, y=0.0}
38 b1 = b0+Vector3:new{x=0.0,y=H1-L1 * tan_alpha}
39 -- End of shock generator is only part way long the plate.
40 L3 = 67 * mm
41 c0 = Vector3:new{x=L3, y=0.0}
42 c1 = c0+Vector3:new{x=0.0,y=H1-(L1+L3) * tan_alpha}
43 -- End of plate, and of the whole flow domain.
44 L2 = 90.0 * mm
45 d0 = Vector3:new{x=L2, y=0.0}
46 d1 = d0+Vector3:new{x=0.0,y=H1}
47 -- Now, define the three patches.
48 patch0 = CoonsPatch:new{p00=a0, p10=b0, p11=b1, p01=a1}
49 patch1 = CoonsPatch:new{p00=b0, p10=c0, p11=c1, p01=b1}
50 patch2 = CoonsPatch:new{p00=c0, p10=d0, p11=d1, p01=c1}
51 --
52 -- Discretization of the flow domain.
53 --
54 -- We want to cluster the cells toward the surface of the flat plate.
55 -- where the boundary layer will be developing.
56 rcf = RobertsFunction:new{end0=true,end1=true,beta=1.1}
57 factor = 4 -- We'll scale discretization off this value
58 ni0 = math.floor(20 * factor); nj0 = math.floor(80 * factor)
59 grid0 = StructuredGrid:new{psurface=patch0, niv=ni0+1, njv=nj0+1,
60                            cfList={east=rcf,west=rcf}}
61 grid1 = StructuredGrid:new{psurface=patch1, niv=7 * ni0+1, njv=nj0+1,
62                            cfList={east=rcf,west=rcf}}
63 grid2 = StructuredGrid:new{psurface=patch2, niv=2 * ni0+1, njv=nj0+1,
64                            cfList={east=rcf,west=rcf}}
65 --
66 -- Build the flow blocks and attach boundary conditions.
67 --
68 blk0 = FBArray:new{grid=grid0, initialState=inflow, nib=1, njb=2,
69       bcList={west=InFlowBC_Supersonic:new{flowState=inflow},
70          north=WallBC_WithSlip:new{},
71          south=WallBC_WithSlip:new{}}}
72 blk1 = FBArray:new{grid=grid1, initialState=inflow, nib=7, njb=2,
73       bcList={south=WallBC_NoSlip_Adiabatic:new{},
74            north=WallBC_WithSlip:new{}}}
75 blk2 = FBArray:new{grid=grid2, initialState=inflow, nib=2, njb=2,
76       bcList={south=WallBC_NoSlip_Adiabatic:new{},
77            north=WallBC_WithSlip:new{},
78            east=OutFlowBC_FixedPT:new{p_outside=p_inf,
```

```
79                                                  T_outside=T_inf}}}
80 identifyBlockConnections()
81
82 config.gasdynamic_update_scheme = "classic-rk3"
83 config.flux_calculator = 'adaptive'
84 config.viscous = true
85 config.spatial_deriv_calc = 'divergence'
86 config.cfl_value = 1.0
87 config.max_time = 5.0 * L2/u_inf -- time in flow lengths
88 config.max_step = 200000
89 config.dt_init = 1.0e-8
90 config.dt_plot = config.max_time/10
```

5.1.3 运行仿真

为了开始模拟，使用以下命令准备网格和初始流状态：

```
$ e4shared --prep --job=swbli
```

这可能需要一些时间，因为单元格在这个仿真中比我们在教程示例中使用的要多得多。我们试图捕捉边界层的发展演变，为了做到这一点，我们必须为使用高分辨率网格而付出更多计算成本。大约一分钟后，这取决于您计算机的速度，您就能得到在子目录 grid/t0000 中的网格文件和在子目录 flow/t0000 中的初始流文件。

我们现在可以开始计算流场的演变，使用以下命令：

```
$ e4shared --run --job=swbli --verbosity=1 --max-cpus=4
```

您应该很快就会看到，对于执行时间步骤的仿真，控制台通常输出并报告其到达最后时间的进度。请耐心等待，因为这个仿真比最初的教程练习要求更高。即使您是在一个大的多核机器上运行，也需要花费很多时间，所以此时可以去吃晚饭，然后在 5-7 个小时后回来检查模拟的状态。

但在你离开之前，请打开电脑上的 htop 监控程序[19]。它很好地给出了正在运行进程的一个概述，以及您所获得的处理器利用率。在一台有 4 个 AMD 核的小型 HP 笔记本电脑上，一旦初始化阶段已经完成并且时间步长阶段已经开始，htop 将显示所有 4 个处理器的利用率已超过 99%。这个仿真很好地平衡了负载，并且很好地利用了多核计算机。除了所使用的处理器资源外，ntop 还显示这个仿真在运行时占用的内存略低于 3GBytes。您可能希望计算机中至少有 8GB 的 RAM 来

[19] 您可能需要使用操作系统的包管理器来手动安装 htop。

运行有意思的仿真。

回到您的计算机，您看到流场的时间演化已经计算了大约 876 μs(需要 11176 个步长)。用 Paraview 显示时，您可以使用以下命令来生成计算结果数据：

```
$ e4shared --post --job=swbli --tindx-plot=all --vtk-xml \
    --add-vars="mach,pitot,total-p,total-h"
```

这可以全部输入到一行中，省去了反斜杠连续字符。

在这个仿真过程的最后，结果表明分离区域仍然在进行微弱的演化，从该区域传播的微小运动波可以看出这一点。我们重新启动计算，并将其运行到 max_time 原始值的两倍。这是通过手动编辑 swlbli.control 文件实现的，如 4.1.4 节所述，并设置 max_time = 1.751e-03 和 dt_init =8.0e-08，然后运行命令：

```
$ e4shared—run—iob=swbli—tindx-start=last—max-cpu =4
```

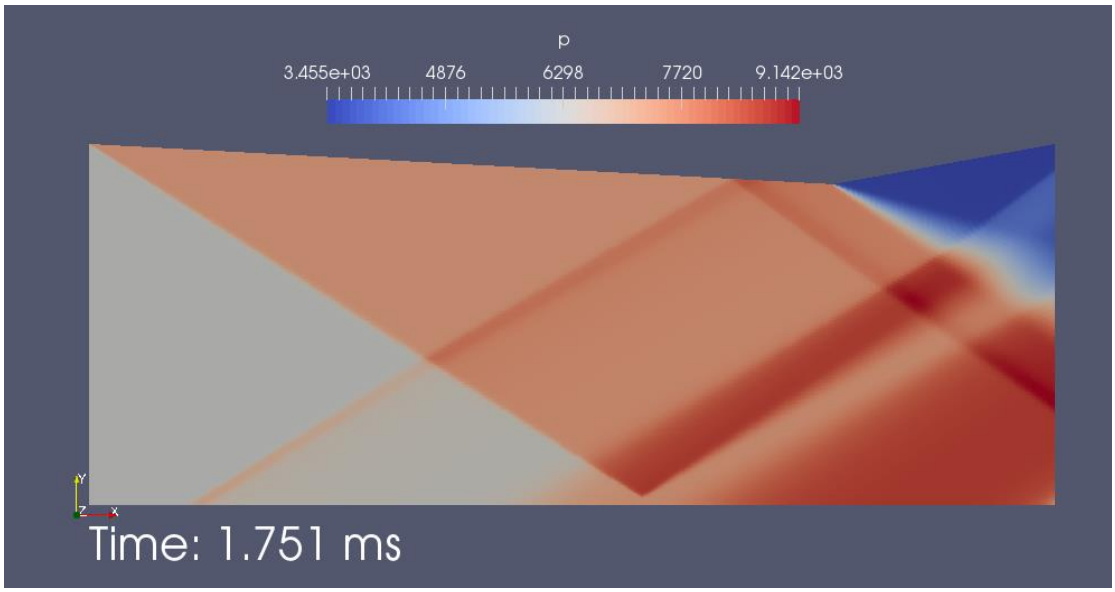
考虑到从为本例准备气体模型文件到现在已经经过了几个小时，可能该睡觉了。对做 CFD 分析的人来说，在夜间甚至几天内进行计算都是相当常见的活动。

5.1.4 仿真结果

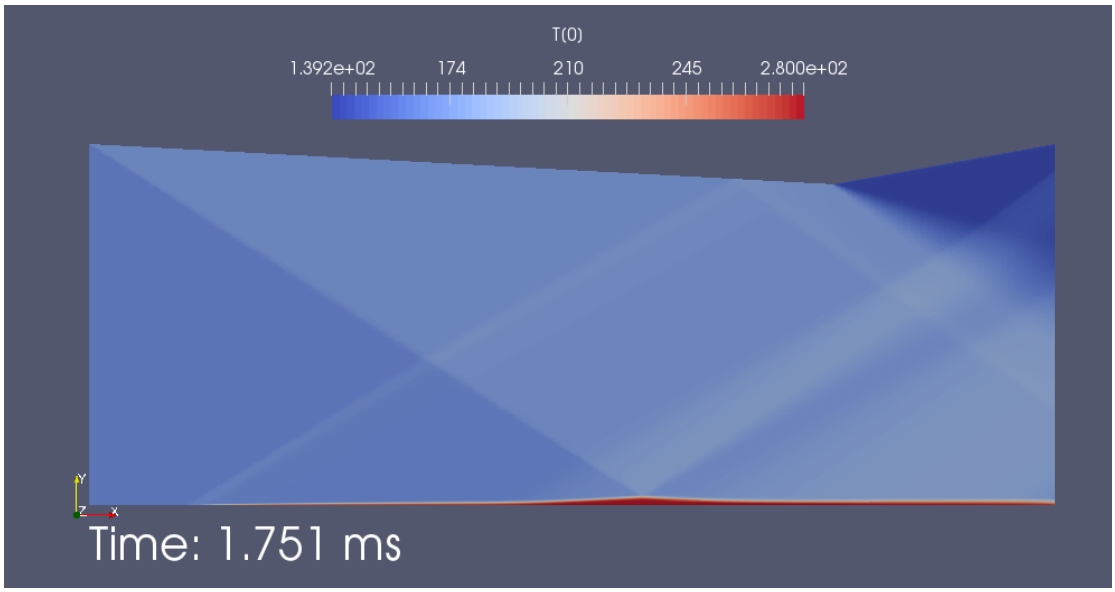
图 5.3 显示了在流动开始 t=1.751 ms 后的部分流场数据。密度梯度的大小(图 5.3(b))也近似为图 5.1 纹影图像所示。压力图像清晰地显示了前缘粘性相互作用而产生的波，及其在激波发生器上的反射。不出所料，边界层在压力场中不明显，但在温度场中较明显。当边界层接近入射激波时，压力场中明显地出现了更宽的压缩带。这是一个扩展然后再压缩的过程。所有这些波都清楚地表现在密度场的梯度上。前缘粘性相互作用产生的激波、膨胀波和再压缩激波表现得更加明显，同时，逐渐压缩流动的收敛性更加明显。扩散族的结构在梯度场中比在压力场和温度场中表现得更为明显。

成功的真正证据是与实验数据进行对比。图 5.4 为沿板的压力和剪切应力。仿真结果为准确估计分离区内的压力分布提供了可靠的依据。看起来有点不对的特性包括 x=0 处的粘性相互作用区域，这是因为边界层开始处分辨率不足，所以这一区域稍微有点扩展，但是加倍网格分辨率(因子=8)会使该区域的解变得更收敛。此外，在仿真区域的右端存在一个人工压降，并且边界层脱离了流场，但这无关紧要，因为实验中使用的平板长度是这种模拟情况下长度的两倍多。这种性能与网格无关。

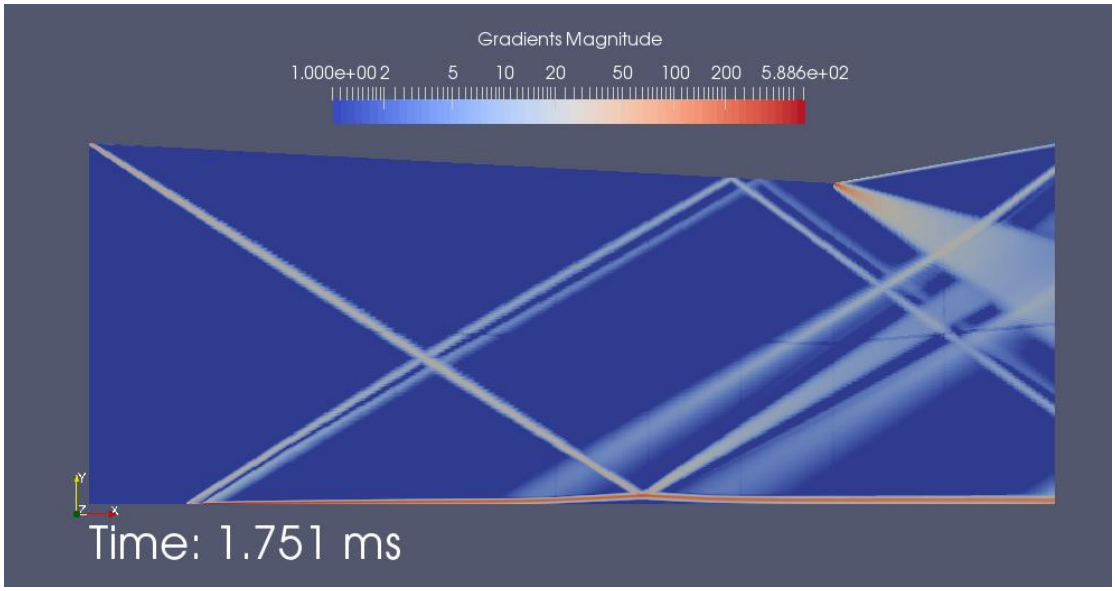
利用第 5.1.5 节的脚本，根据现场数据，模拟得到了合理的剪切应力。这个量很难计算，也很难测量，所以可以肯定的是，这两组数据在进入交互区域的边界层时，与 Blasius 值吻合得很好。在相互作用区域结束后，计算值恢复到 Blasius 值水平，然后上升到流域的末端。这也是与流出边界条件的相互作用，如果模拟整块板的长度，它就会从视图中消失。随着网格分辨率的增加(从 factor=4 到 factor=8)，唯一可识别的区别是边界层的早期发展更接近于 Blasius 特性。



(a) 压力场

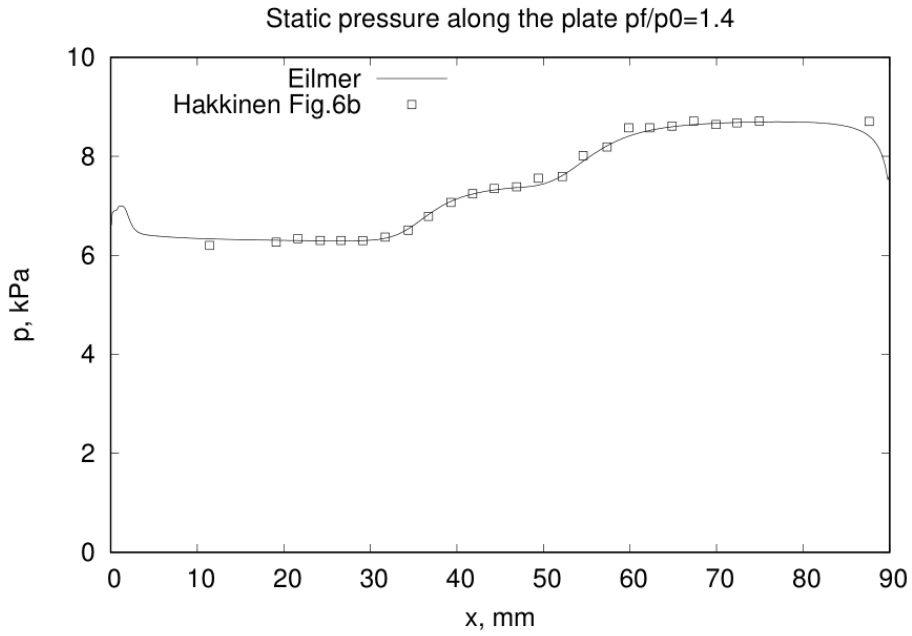


(b) 温度场

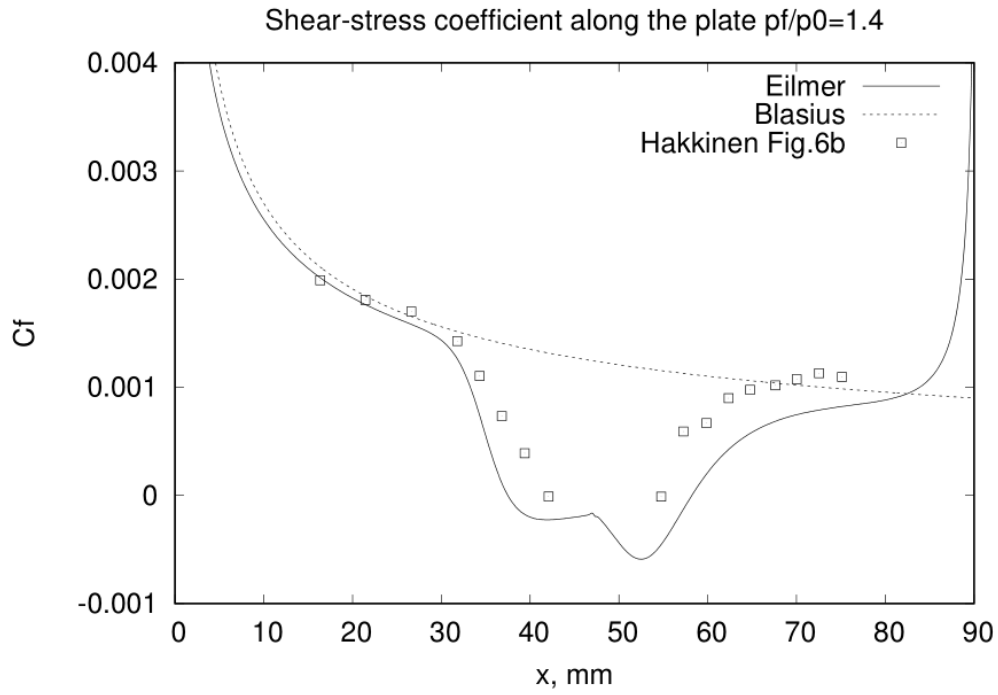


(c) 密度梯度场

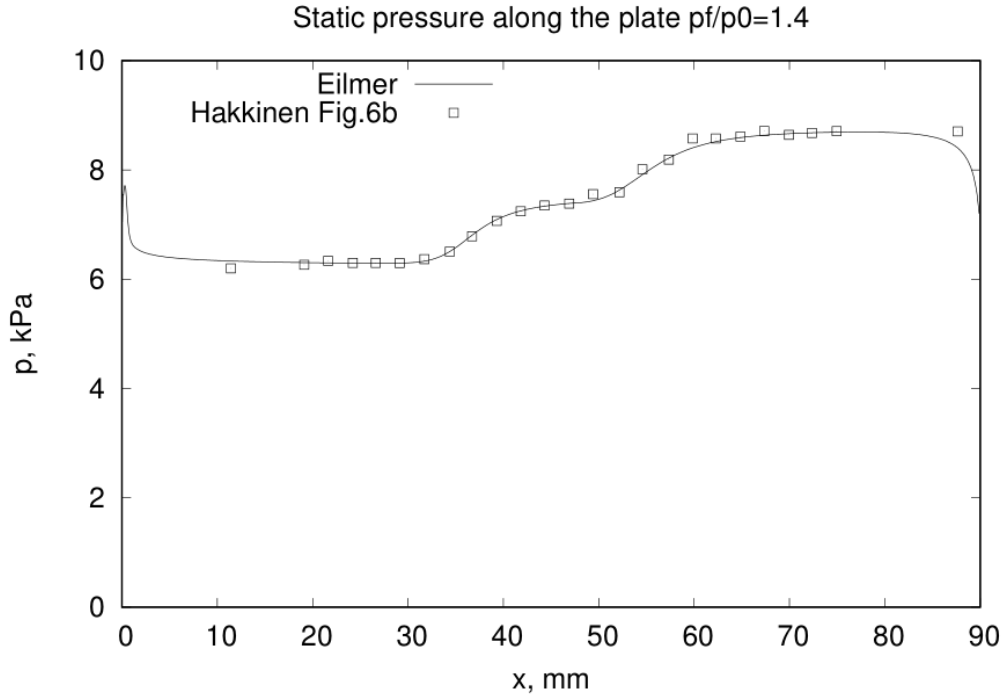
图 5.3：在 t=1.751ms 时的流场



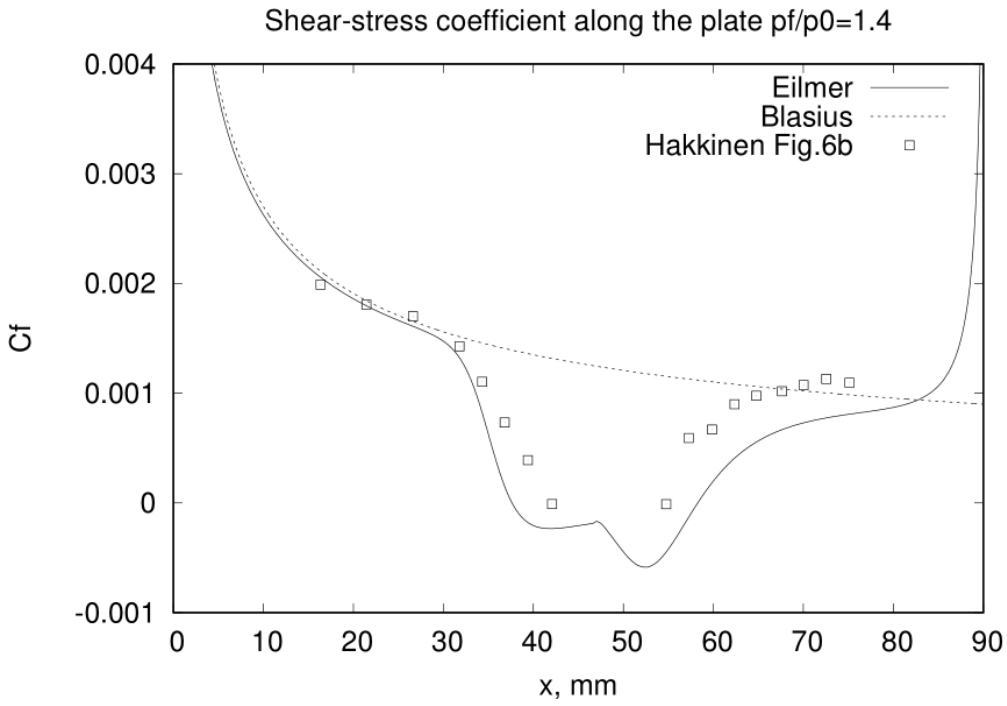
(a) 压力(factor=4)



(b)剪切力(factor=4)



(c)压力(factor=8)



(b)剪切力(factor=8)

图 5.4：在 t=1.751ms 时的压力和剪切力分布

注意平板边界层对平板附近区域压力的影响很小，但可以测量。在流入面规定的自由流动压力为 6.205 kPa 时，我们在导致相互作用区域的压力数据中看到了 6.28 kPa。对于所使用的自由流条件，一个简单平板边界层的位移厚度更希望是 0.112mm，距板前沿 25mm 处。如果这个位移效应可以建模为直的楔形偏转无粘自由流动，则相应斜激波的静压比为 1.0146。这使得导致激波相互作用的边界层外流压力期望值为 6.295 kPa，与仿真所得值非常接近。

5.1.5 剪切应力的后处理

为了得到沿板块的压力和剪切应力的估计值，我们可以提取所有沿板的块在南边界单元格的流动数据。这可以通过以下命令[20]完成：

```
$ e4shared --post --job=swbli --tindx-plot=last --add-vars="mach" --output-file=bl.data \
--slice-list="2,:,0,0;4,:,0,0;6,:,0,0;8,:,0,0;10,:,0,0;12,:,0,0;14,:,0,0;16,:,0,0;18,:,0,0"
```

其中，赋给 slice-list 选项的字符串为每个位于板上的块(即块 2、4、6、8、10、12、14、16 和 18)挑选 j=0 和 k=0 的单元格行。

在输出文件 bl.data 的第 1 列和第 9 列中，可以找到每个单元格的 x 位置和静

[20] 即使使用小字体，指定切片列表的字符串也会运行到这个报告的行尾。在 Linux 系统上，将所有内容放到往常命令窗口的一行中应该没有问题。

压值。因此，沿表面的压力可以直接绘制出来。但是，剪切应力需要以 bl.data 文件中提供的单元格数据来计算。以下的 AWK 脚本完成了这项工作。

```
# compute-shear.awk
# Invoke with the command line:
# $ awk -f compute-shear.awk bl.data > shear.data
#
# PJ, 2016-11-01
#
BEGIN {
        rho_inf = 0.1315 # kg/m ** 3
        velx_inf = 514.0 # m/s
        T_inf = 164.4 # K
        # Sutherland expression for viscosity
        mu_ref = 1.716e-5; T_ref = 273.0; S_mu = 111.0
        mu_inf = (T_inf/T_ref) * sqrt(T_inf/T_ref) * (T_ref+S_mu)/(T_inf+S_mu) * mu_ref
        print("# x(m) tau_w(Pa) Cf y_plus")
}
$1 != "#" {
        x = $1; y = $2; rho = $5; velx = $6; mu = $11; k = $12
        dvelxdy = (velx - 0.0) / y # Assuming that the wall is straight down at y=0
        tau_w = mu*dvelxdy      # wall shear stress
        Cf = tau_w / (0.5 * rho_inf * velx_inf * velx_inf)
        if (tau_w > 0.0) abs_tau_w = tau_w; else abs_tau_w = -tau_w;
        vel_tau = sqrt(abs_tau_w / rho)      # friction velocity
        y_plus = vel_tau*y*rho / mu
        Rex = rho_inf*velx_inf*x / mu_inf
        Cf_blasius = 0.664 / sqrt(Rex)
        print(x, tau_w, Cf, Cf_blasius, y_plus)
}
```

这个脚本是用来过滤 bl.data 文件，获得除去以“#”字符开始的每一行数据,计算剪应力、摩擦系数、理论层流(Blasius)剪切应力和 y+的预估值，以及将这些结果显示到标准输出。例如， $1 形状的变量名从过滤器捕获的每行特定列中提取数据。下面的命令用于将 AWK 过滤器程序应用到 bl. data 文件，并将输出 shear.data 重定向到后续绘图。

```
$ awk -f compute-shear.awk < bl.data > shear.data
```

5.2 氮气在有限长度的圆柱体上流动

这个例子与 Troy Eichmann 的 X2 实验[21]有关，实验是关于弱电离氮在不同长径比的圆柱体上的流动。它通过强烈的钝体激波和圆柱体末端的突然膨胀来测试三维流动求解器。在流场和温度高于 20000 K 接近平衡和冻结的热化学区域，也对热化学模块进行了测试。

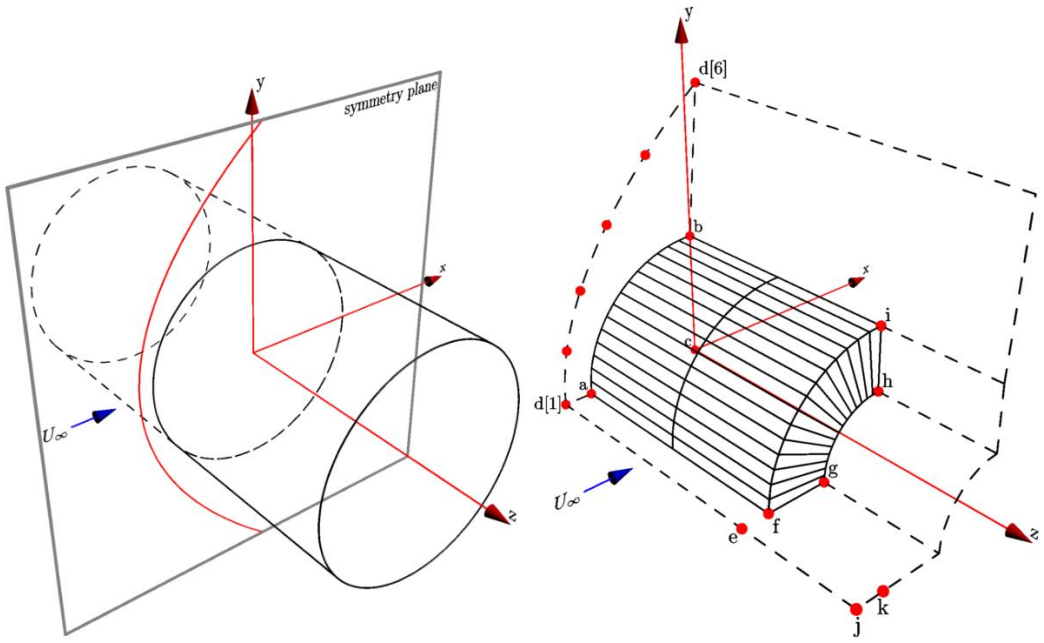


图 5.5：左：完整的圆柱体，对称平面上标出了预期的激波位置。右图：有限圆柱仿真布局图，四分之一的前向圆柱体表面显示为线框。流域的一些边缘显示为虚线，标记的节点对应于输入脚本中的节点。

图中所示的流场由图 5.5 所示的 4 个结构化网格块组成，对于直径为 15mm 和 L/D=2 的圆柱体，其表面网格数量如图 5.6 所示。请注意，只有一半的长度和上部四分之一的圆柱体进行了仿真计算。沿对称面(隐式地)使用滑壁边界条件。

自由气流条件(p_∞= 2kPa, T_∞= 3000k 以及 u_∞= 10km/s)与实验结果基本一致。这些是由 X2 膨胀管产生且具有代表性的条件，对于理想的氮气测试气体，自由气流马赫数为 8.96。这节描述了具有单温度化学非平衡态的有限圆柱体仿真。这意味着化学反应允许以有限速率发生(化学非平衡)，但气体的所有内能模式都被认为是由单一温度控制的(热平衡)。

该脚本设置仿真运行 30 个流动长度(30*R_c/u_∞)，最终达到 22.5μs。压力场和温度场对激波的缓解效果很明显(图 5.7)。温度场也反映了有限速率反应的影响，

其峰值温度紧随激波之后，然后随着氮分子的解离并吸收激波层内的能量而发生松弛。

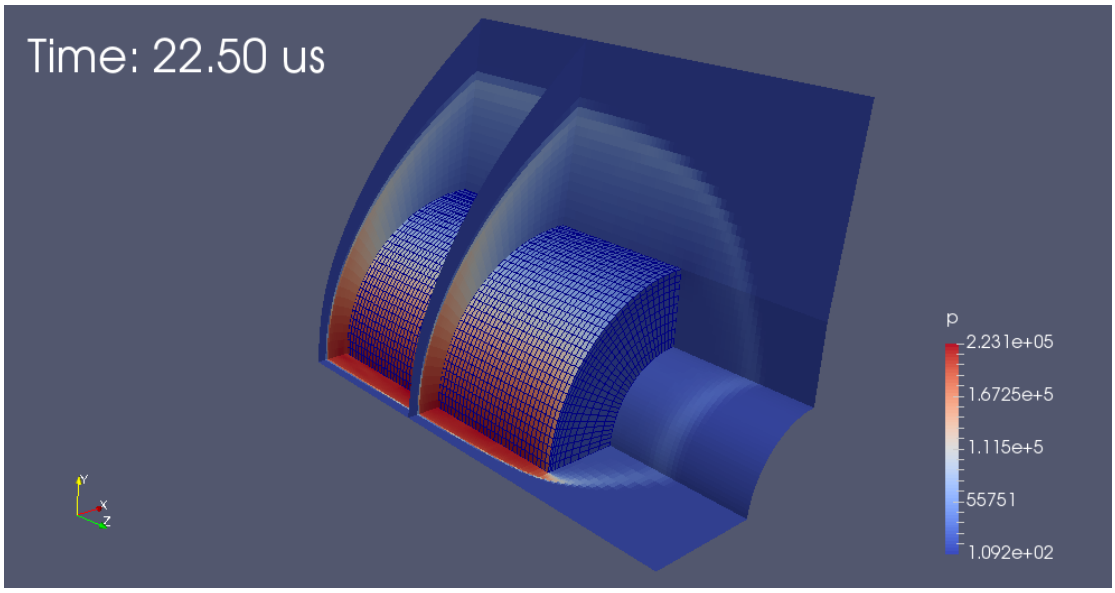


图 5.6: 如圆柱体表面的线框所示，从化学非平衡的有限圆柱体仿真中选择的网格表面，在流场中被压力着色。这个 PNG 图是使用从最终计算结果文件中提取的代码块来生成的。

这种情况对于流动求解器是相当困难的，可以在圆柱体的平端附近看到缺陷，其中有很强的膨胀性和剪切力。这些缺陷是一个带有低温度格子图案的可视化温度场。有时候您回发现，如果不替换该区域中的劣质单元数据，仿真将无法运行。我们的出狱选项是 adjust_invalid_cell_data，结合了输入脚本中第 148 行和第 149 行的 max_invalid_cells。

除了在圆柱体边缘存在的流动计算问题，前体流动的计算似乎是可靠的，在圆柱体平面附近的冲击距离为 1.2 mm。

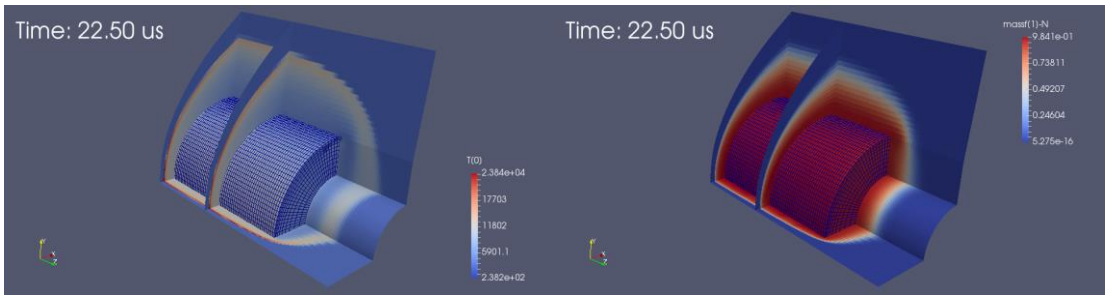


图 5.7:化学非平衡仿真流场中氮原子的静态温度和质量分数。

这种模拟可以充分利用多处理器。使用 4 个 CPU 运行的时间比使用 4 个 AMD 核的 HP Pavillion 笔记本电脑大约多了一个小时。要将网格分辨率提高一倍(在收敛性研究中可能需要这样做)，需要增加 8 倍的内存和 16 倍的 CPU 周期。

如果您计划进行任何合理复杂度的计算，那么投资一台拥有大量 CPU 内核的计算机是值得的。

输入脚本(.lua)

```
1 -- cyl.lua
2 -- Troy and Tim's finite-length cylinder in dissociating nitrogen flow.
3 --
4 -- PJ & RJG 2016-10-16 built from the eilmer3/3D/finite-cylinder
5 -- and eilmer4/2D/cylinder-dlr-n90
6
7 config.dimensions = 3
8 D = 15.0e-3 -- diameter of cylinder, metres
9 L = 2.0*D -- (axial) length of full cylinder, will be halved later
10
11 -- Free-stream properties
12 T_inf = 3000.0 -- degrees K
13 p_inf = 2000.0 -- Pa
14 V_inf = 10.0e3 -- m/s
15 config.title = string.format("Cylinder L/D=%g in N2 at u=%g m/s.", L/D, V_inf)
16 print(config.title)
17
18 nsp, nmodes, gm = setGasModel('nitrogen-2sp.lua')
19 print("GasModel set nsp= ", nsp, " nmodes= ", nmodes)
20
21 -- Compute inflow Mach number.
22 Q = GasState:new{gm}
23 Q.p = p_inf; Q.T = T_inf; Q.massf = {N2=1.0}
24 gm:updateThermoFromPT(Q); gm:updateSoundSpeed(Q)
25 print("T=", Q.T, "density=", Q.rho, "sound speed= ", Q.a)
26 M_inf = V_inf / Q.a
27 print("M_inf=", M_inf)
28
29 inflow = FlowState:new{p=p_inf, T=T_inf, velx=V_inf, massf={N2=1.0}}
30 initial = FlowState:new{p=p_inf/3, T=300.0, massf={N2=1.0}}
31
32 config.reacting = true
33 config.reactions_file = 'e4-chem.lua'
34
35 -- Build geometry from body and flow parameters
36 Rc = D/2.0 -- radius of cylinder
37
38 a = Vector3:new{x=-Rc}; b = Vector3:new{y=Rc}; c = Vector3:new{x=0.0, y=0.0}
39
40 -- In order to have a grid that fits reasonably close the the shock,
```

```
-- use Billig's shock shape correlation to generate
-- a few sample points along the expected shock position.
dofile("billig.lua")
M_inf = 8.9566
print("Points on Billig's correlation.")
xys = {}
for i,y in ipairs({0.0, 0.5, 1.0, 1.5, 2.0, 2.5}) do
    x = x_from_y(y * Rc, M_inf, 0.0, false, Rc)
    xys[#xys+1] = {x=x, y=y * Rc} -- a new coordinate pair
    print("x=", x, "y=", y * Rc)
end

-- Scale the Billig distances, depending on the expected behaviour
-- relative to the gamma=1.4 ideal gas.
local b_scale = 1.1 -- for ideal (frozen-chemistry) gas
if config.reacting then
    b_scale = 0.87 -- for finite-rate chemistry
end
d = {} -- will use a list to keep the nodes for the shock boundary
for i, xy in ipairs(xys) do
    -- the outer boundary should be a little further than the shock itself
    d[#d+1] = Vector3:new{x=-b_scale * xy.x, y=b_scale * xy.y, z=0.0}
end
print("front of grid: d[1]=", d[1])

-- Extent of the cylinder in the z-direction to end face.
zshift = Vector3:new{z=L/2.0}
c2 = c + zshift
e = d[1] + zshift
f = a + zshift
g = Vector3:new{x=-Rc/2.0, y=0.0, z=L/2.0}
h = Vector3:new{x=0.0, y=Rc/2.0, z=L/2.0}
i = Vector3:new{x=0.0, y=Rc, z=L/2.0}
-- the domain is extended beyond the end of the cylinder
zshift2 = Vector3:new{z=Rc}
j = e + zshift2
k = f + zshift2

-- ...then lines, arcs, etc, that will make up the domain-end face.
xaxis = Line:new{p0=d[1], p1=a} -- first-point of shock to nose of cylinder
cylinder = Arc:new{p0=a, p1=b, centre=c}
shock = ArcLengthParameterizedPath:new{underlying_path=Spline:new{points=d}}
outlet = Line:new{p0=d[#d], p1=b} -- top-point of shock to top of cylinder
domain_end_face = CoonsPatch:new{south=xaxis, north=outlet,
```

```
85                          west=shock, east=cylinder}
86
87 -- ...lines along which we shall extrude the domain-end face
88 yaxis0 = Line:new{p0=d[1], p1=e}
89 yaxis1 = Line:new{p0=e, p1=j}
90
91 -- End-face of cylinder
92 xaxis = Line:new{p0=f, p1=g}
93 cylinder = Arc:new{p0=f, p1=i, centre=c2}
94 inner = Arc:new{p0=g, p1=h, centre=c2}
95 outlet = Line:new{p0=i, p1=h}
96 cyl_end_face = CoonsPatch:new{south=xaxis, north=outlet,
97 west=cylinder, east=inner}
98 yaxis2 = Line:new{p0=f, p1=k}
99
100 over_cylinder = SweptSurfaceVolume:new{face0123=domain_end_face,
101                                       edge04=yaxis0}
102 outside_cylinder = SweptSurfaceVolume:new{face0123=domain_end_face,
103                                       edge04=yaxis1}
104 beside_cylinder = SweptSurfaceVolume:new{face0123=cyl_end_face,
105                                       edge04=yaxis2}
106
107 -- Build discrete grids.
108 -- We choose a basic discretization and scale others from it.
109 nr = 20 -- number of cells radially
110 nc = math.floor(1.5*nr) -- number of cells circumferentially
111 na = math.floor(L/D*nc) -- number of cells along the cylinder
112 na1 = nc -- cells off the end of the cylinder
113 nr2 = math.floor(nr/2) -- cells toward the cylinder axis
114 -- Adjust the cluster functions by trying various values.
115 cf0 = RobertsFunction:new{end0=true, end1=false, beta=1.5}
116 grid0 = StructuredGrid:new{pvolume=over_cylinder,
117                            cfList={edge03=cf0, edge47=cf0},
118                            niv=nr+1, njv=nc+1, nkv=na+1}
119 grid1 = StructuredGrid:new{pvolume=outside_cylinder,
120                           cfList={edge03=cf0, edge47=cf0},
121                           niv=nr+1, njv=nc+1, nkv=na1+1}
122 cf1 = RobertsFunction:new{end0=true, end1=false, beta=1.05}
123 cf2 = RobertsFunction:new{end0=true, end1=false, beta=1.6}
124 grid2 = StructuredGrid:new{pvolume=beside_cylinder,
125                           cfList={edge01=cf1, edge32=cf2,
126                           edge45=cf1, edge76=cf2},
127                           niv=nr2+1, njv=nc+1, nkv=na1+1}
128
```

```
-- Use the grids to define some flow blocks.
-- Note that we divide up the biggest grid to make better use
-- of our multiple cpu machine.
blk0 = FluidBlockArray{grid=grid0, initialState=initial,
                       bcList={west=InFlowBC_Supersonic:new{flowState=inflow},
                               north=OutFlowBC_Simple:new{}},
                               nkb=math.floor(L/D)}
blk1 = FluidBlock:new{grid=grid1, initialState=initial,
                       bcList={west=InFlowBC_Supersonic:new{flowState=inflow},
                               north=OutFlowBC_Simple:new{}}}
blk2 = FluidBlock:new{grid=grid2, initialState=initial,
                       bcList={east=OutFlowBC_Simple:new{},
                               north=OutFlowBC_Simple:new{}}}
identifyBlockConnections()

-- Set a few more config options
config.flux_calculator = "adaptive"
config.thermo_interpolator = "pT"
config.adjust_invalid_cell_data = true
config.report_invalid_cells = false
config.max_invalid_cells = 10
-- config.cfl_count = 3
config.gasdynamic_update_scheme="euler"
my_max_time = Rc/V_inf*30
print("max_time=", my_max_time)
config.max_time = my_max_time
config.max_step = 40000
config.dt_init = 1.0e-10
config.cfl_value = 0.5
config.dt_plot = my_max_time/10
```

反应格式文件(.lua)

```
-- nitrogen-2sp-2r.lua
--
-- This chemical kinetic system provides
-- a simple nitrogen dissociation mechanism.
--
-- Author: Rowan J. Gollan
-- Date: 13-Mar-2009 (Friday the 13th)
-- Place: NIA, Hampton, Virginia, USA
--
-- History:
-- 24-Mar-2009 - reduced file to minimum input
-- 11-Aug-2015 - updated for dlang module
```

```
13
14 --[[
15 Config{
16         tightTempCoupling = true
17           }
18 --]]
19
20 Reaction{
21               'N2 + N2 <=> N + N + N2',
22               fr={'Arrhenius', A=7.0e21, n=-1.6, C=113200.0},
23               br={'Arrhenius', A=1.09e16, n=-0.5, C=0.0}
24 }
25
26 Reaction{
27               'N2 + N <=> N + N + N',
28               fr={'Arrhenius', A=3.0e22, n=-1.6, C=113200.0},
29               br={'Arrhenius', A=2.32e21, n=-1.5, C=0.0}
30 }
```

Shell 脚本

```
1 #!/bin/bash
2 # prep.sh
3 prep-gas nitrogen-2sp.inp nitrogen-2sp.lua
4 prep-chem nitrogen-2sp.lua nitrogen-2sp-2r.lua e4-chem.lua
5 e4shared --prep --job=cyl
```

```
1 #!/bin/bash
2 # run.sh
3 e4shared --run --job=cyl --verbosity=1 --max-cpus=4
```

后处理程序

```
1 #!/bin/bash
2 # post.sh
3
4 # Create a VTK plot file of the steady full flow field.
5 e4shared --post --job=cyl --tindx-plot=last --vtk-xml
6
7 # Pull out the cylinder surfaces.
8 e4shared --post --job=cyl --tindx-plot=last --output-file=cylinder \
9 --add-vars="mach" --surface-list="0,east;1,east;3,bottom"
10
11 # Now pull out some block surfaces that show cross-sections of the flow field.
12 e4shared --post --job=cyl --tindx-plot=last --output-file=interior \
13 --add-vars="mach" \
14—surface-list=
```

```
                "0,bottom;1,bottom;0,north;1,north;2,north;3,north;0,south;1,south;2,south;3,s
                outh;3,east"
15
16 # Stagnation-line flow data
17 e4shared --post --job=cyl --tindx-plot=last --output-file=stagnation-line.data \
18 --add-vars="mach" --slice-list="0,:,0,0" \
```

```
1 -- locate-bow-shock.lua
2 -- Invoke with the command line:
3 -- $ e4shared --custom-post --script-file=locate-bow-shock.lua
4 --
5 -- PJ, 2016-10-24, updated for Eilmer4
6 --
7 print("Locate a bow shock by its pressure jump.")
8 print("Start by reading full flow solution.")
9 fsol = FlowSolution:new{jobName="cyl", dir=".", tindx=10, nBlocks=4}
10 print("fsol=", fsol)
11
12 function locate_shock_along_strip()
13 local p_max = ps[1]
14 for i = 2, #ps do
15 p_max = math.max(ps[i], p_max)
16 end
17 local p_trigger = ps[1] + 0.3*(p_max - ps[1])
18 local x_old = xs[1]; local p_old = ps[1]
19 local x_new = x_old; local p_new = p_old
20 for i = 2, #ps do
21 x_new = xs[i]; p_new = ps[i]
22 if p_new > p_trigger then break end
23 x_old = x_new; p_old = p_new
24 end
25 local frac = (p_trigger - p_old) / (p_new - p_old)
26 x_loc = x_old*(1.0 - frac) + x_new*frac
27 return
28 end
29
30 -- Since this is a 3D simulation, the shock is not expected
31 -- to be flat in the k-direction (along the cylinder axis).
32 -- Sample the shock layer in a few places near the stagnation line.
33 -- Block 0 contains the stagnation point and the bottom surface is
34 -- the plane of symmetry that cuts the cylinder half-way along its axis.
35 -- The south boundary is the plane of symmetry that cuts the cylinder
36 -- along its axis. Supersonic flow comes in from the west boundary
```

```
-- and exits from the north boundary. The east boundary is the
-- cylinder surface.
local xshock = {}; local yshock = {}
local ib = 0
local nk = fsol:get_nkc(0)
for k = 0, nk-1 do
xs = {}; ys = {}; ps = {}
local j = 0
local ni = fsol:get_nic(ib)
for i = 0, ni-1 do
cellData = fsol:get_cell_data{ib=ib, i=i, j=j, k=k}
xs[#xs+1] = cellData["pos.x"]
ps[#ps+1] = cellData["p"]
end
locate_shock_along_strip()
xshock[#xshock+1] = x_loc
yshock[#yshock+1] = y_loc
if #xshock >= 6 then break end
end

x_sum = 0.0
for _,x in ipairs(xshock) do x_sum = x_sum + x end
x_average = x_sum / #xshock
print("Average x-location=", x_average)
D = 15.0e-3 -- cylinder diameter
delta = -x_average - D/2
print("shock displacement=", delta * 1000.0, "mm")
print("delta/R=", delta/(D/2))
```

5.3 尖角锥上重新观察流动

作为最后一个示例，让我们回到第 3 节的第一个教程示例中讨论的尖角锥上的流动，但是要介绍对非结构化网格的仿真，这是 Eilmer 的一个重要新特性。该功能允许对几何复杂的流场进行仿真。虽然圆锥上流域的二维几何结构并不复杂，但它是熟悉的，并且显示了在非结构化网格集合上计算流场的基本安排。

5.3.1 输入脚本(.lua)

由于所有的几何建模和网格构建都是由 Kyle Damm 在逐点网格生成包中完成，所以在输入脚本中不需要太多。

```
-- cone20.lua
-- Unstructured Grid Example -- for use with Eilmer4
-- 2015-11-08 PeterJ, RowanG, KyleD

config.title = "Mach 1.5 flow over a 20 degree cone -- Unstructured Grid."
print(config.title)
config.dimensions = 2
config.axisymmetric = true

nsp, nmodes, gm = setGasModel('ideal-air-gas-model.lua')
print("GasModel set to ideal air. nsp= ", nsp, " nmodes= ", nmodes)
initial = FlowState:new{p=5955.0, T=304.0}
inflow = FlowState:new{p=95.84e3, T=1103.0, velx=1000.0}

-- Define the flow domain using an imported grid.
grids = {}
for i=0,3 do
        fileName = string.format("cone20_grid%d.su2", i)
        grids[i] = UnstructuredGrid:new{filename=fileName, fmt="su2text"}
end
my_bcDict = {INFLOW=InFlowBC_Supersonic:new{flowState=inflow},
             OUTFLOW=OutFlowBC_Simple:new{},
             SLIP_WALL=WallBC_WithSlip:new{},
             INTERIOR=ExchangeBC_MappedCell:new{list_mapped_cells=true},
             CONE_SURFACE=WallBC_WithSlip:new{}
}
blks = {}
for i=0,3 do
    blks[i] = FluidBlock:new{grid=grids[i], initialState=inflow,
    bcDict=my_bcDict}
end

-- Do a little more setting of global data.
config.max_time = 5.0e-3 -- seconds
config.max_step = 3000
config.dt_init = 1.0e-6
config.cfl_value = 0.5
config.dt_plot = 1.5e-3
config.dt_history = 10.0e-5

setHistoryPoint{x=1.0, y=0.2} -- nose of cone
setHistoryPoint{x=0.201, y=0.001} -- base of cone
```

主要任务是：

•选择一个气体模型(第 10 行)；

•定义初始条件和流入条件(第 12 行和第 13 行)；

•从外部文件(ine 16 到 20)导入网格；

•从导入的网格(第 27 行到第 30 行)构建流体块，附加边界条件。

注意，用于边界条件的表(第 21 到 26 行)关键是需要与 SU2 网格文件中的标记匹配。这些标签可以在网格生成时商定，也可以使用以下命令从网格文件中提取：

```
$ grep MARKER_TAG * .su2
```

要应用边界条件，将边界条件目录作为 bcDict 项提供给 FluidBlock 构造函数(第 29 行)。

5.3.2 结果及后处理

当使用 HP Pavillion 笔记本电脑的 4 个内核时，仿真运行 1641 步，时间为 5 毫秒，需要大约 80 秒的时间。图 5.8 显示了流场，与结构化网格仿真相比，激波显得更嘈杂，但基本上是直线的。后处理脚本加载这次数据并对许多直线上的流场进行采样，寻找沿每个采样直线上的压力跃变，并累积激波的坐标阵列。最后，在第 58 行及以后的部分，将线性模型拟合到这个激波点集合(使用最小二乘误差准则)。该线与这些点的夹角为 49.220°，平均偏差为 2.8mm。

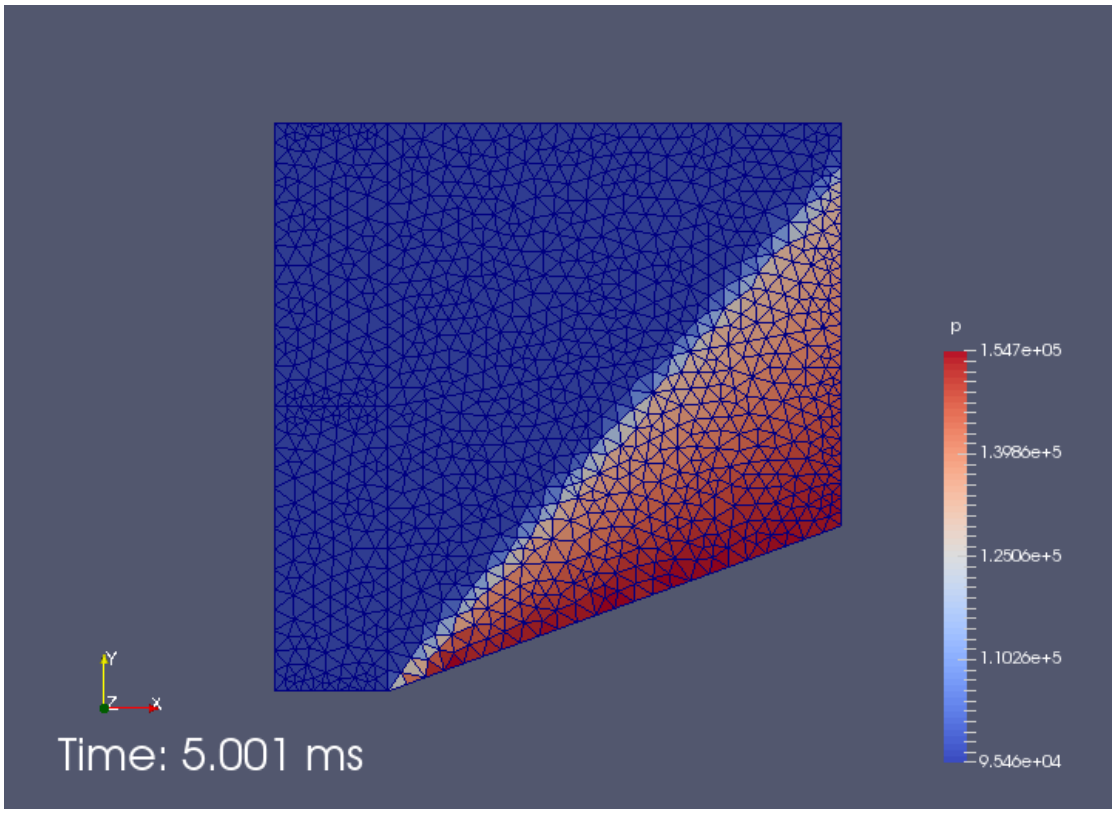


图 5.8：流过 20° 半圆锥的低分辨率、非结构化网格仿真的压力场

```
1 -- estimate_shock_angle.lua
2 -- Invoke with the command line:
3 -- $ e4shared --custom-post --script-file=estimate_shock_angle.lua
4 -- PJ, 2016-11-13
5 --
6 print("Begin estimate_shock_angle for the unstructured-grid case.")
7 nb = 4
8 fsol = FlowSolution:new{jobName="cone20", dir=".", tindx=4, nBlocks=nb}
9 print("fsol=", fsol)
10
11 function locate_shock_along_strip()
12 local p_max = ps[1]
13 for i = 2, #ps do
14 p_max = math.max(ps[i], p_max)
15 end
16 local p_trigger = ps[1] + 0.3*(p_max - ps[1])
17 local x_old = xs[1]; local y_old = ys[1]; local p_old = ps[1]
18 local x_new = x_old; local y_new = y_old; local p_new = p_old
19 for i = 2, #ps do
20 x_new = xs[i]; y_new = ys[i]; p_new = ps[i]
21 if p_new > p_trigger then break end
22 x_old = x_new; y_old = y_new; p_old = p_new
23 end
24 local frac = (p_trigger - p_old) / (p_new - p_old)
25 x_loc = x_old*(1.0 - frac) + x_new*frac
26 y_loc = y_old*(1.0 - frac) + y_new*frac
27 return
28 end
29
30 xshock = {}; yshock = {}
31 for j = 1, 45 do
32 local y = j * 0.02
33 xs = {}; ys = {}; ps = {}
34 local cellsFound = fsol:find_enclosing_cells_along_line{p0={x=0.0,y=y},
35 p1={x=1.0,y=y},
36 n=100}
37 print("number of cells found=", #cellsFound)
38 for i,indices in ipairs(cellsFound) do
39 cellData = fsol:get_cell_data{ib=indices.ib, i=indices.i}
40 xs[#xs+1] = cellData["pos.x"]
41 ys[#ys+1] = cellData["pos.y"]
42 ps[#ps+1] = cellData["p"]
43 end
```

```
44 locate_shock_along_strip()
45 if x_loc < 0.9 then
46 -- Keep only the good part of the shock.
47 xshock[#xshock+1] = x_loc
48 yshock[#yshock+1] = y_loc
49 end
50 end
51
52 --[[
53 for j = 1, #xshock do
54 print("shock point j=", j, xshock[j], yshock[j])
55 end
56 --]]
57
58 -- Least-squares fit of a straight line for the shock
59 -- Model is y = alpha0 + alpha1*x
60 sum_x = 0.0; sum_y = 0.0; sum_x2 = 0.0; sum_xy = 0.0
61 for j = 1, #xshock do
62 sum_x = sum_x + xshock[j]
63 sum_x2 = sum_x2 + xshock[j] * xshock[j]
64 sum_y = sum_y + yshock[j]
65 sum_xy = sum_xy + xshock[j] * yshock[j]
66 end
67 N = #xshock
68 alpha1 = (sum_xy/N - sum_x/N*sum_y/N) / (sum_x2/N - sum_x/N*sum_x/N)
69 alpha0 = sum_y/N - alpha1*sum_x/N
70 shock_angle = math.atan(alpha1)
71 sum_y_error = 0.0
72 for j = 1, N do
73 sum_y_error = sum_y_error+math.abs((alpha0+alpha1 * xshock[j])-yshock[j])
74 end
75 print("shock_angle_deg=", shock_angle * 180.0/math.pi)
76 print("average_deviation_metres=", sum_y_error/N)
77 print("Done.")
```

参考文献

[1]. W. Y. K. Chan, M. K. Smart, and P. A. Jacobs. Experimental validation of the T4 Mach 7.0 nozzle. School of Mechanical and Mining Engineering Technical Report 2014/14, The University of Queensland, Brisbane, Australia, September 2014.

[2]. Peter A. Jacobs and Rowan J. Gollan. The Eilmer 4.0 flow simulation program: Formulation of the transient flow solver. School of Mechanical and Mining Engineering Technical Report 2018/03, The University of Queensland, Brisbane, Australia, April 2018.

[3]. R. J. Gollan and P. A. Jacobs. About the formulation, verification and validation of the hypersonic flow solver Eilmer. International Journal for Numerical Methods in Fluids, 73(1):19–57, 2013.

[4]. Peter Jacobs and Rowan Gollan. Implementation of a compressible-flow simulation code in the D programming language. In Advances of Computational Mechanics in Australia, volume 846 of Applied Mechanics and Materials, pages 54–60.Trans Tech Publications, 9 2016.

[5]. Rowan J. Gollan and Peter A. Jacobs. The Eilmer 4.0 flow simulation program: Guide to the basic gas models package, including gas-calc and the API. School of Mechanical and Mining Engineering Technical Report 2017/27, The University of Queensland, Brisbane, Australia, February 2018.

[6]. Rowan J. Gollan and Peter A. Jacobs. The Eilmer 4.0 flow simulation program: Reacting gas thermochemistry with application examples, including poshax.lua. School of Mechanical and Mining Engineering Technical Report 2018/04, The University of Queensland, Brisbane, Australia, April 2018.

[7]. James M. Burgess. Adaptive look-up table gas model for Eilmer4. School of Mechanical and Mining Engineering Technical Report 2016/18, The University of Queensland, Brisbane, Australia, March 2016.

[8]. Peter A. Jacobs, Rowan J. Gollan, and Ingo Jahn. The Eilmer 4.0 flow simulation program: Guide to the geometry package, for construction of flow paths. School of

Mechanical and Mining Engineering Technical Report 2017/25, The University of Queensland, Brisbane, Australia, February 2018.

[9]. Kyle Damm. Shock fitting mode for Eilmer. School of Mechanical and Mining Engineering Technical Report 2016/15, The University of Queensland, Brisbane, Australia, February 2016.

[10]. J. W. Maccoll. The conical shock wave formed by a cone moving at high speed. Proceedings of the Royal Society of London, 159(898):459–472, 1937.

[11]. P. A. Jacobs. Single-block Navier-Stokes integrator. ICASE Interim Report 18, 1991.

[12]. Roberto Ierusalimschy. Programming in Lua. Lua.org, 2006.

[13]. P. M. Knupp. A robust elliptic grid generator. Journal of Computational Physics, 100(2):409–418, 1992.

[14]. Ames Research Staff. Equations, tables and charts for compressible flow. NACA Report 1135, 1953.

[15]. D. I. Pullin. Direct simulation methods for compressible inviscid ideal-gas flow. Journal of Computational Physics, 34(2):231–244, 1980.

[16]. M. N. Macrossan. The equilibrium flux method for the calculation of flows with non-equilibrium chemical reactions. Journal of Computational Physics, 80(1):204–231, 1989.

[17]. Y. Wada and M. S. Liou. A flux splitting scheme with high-resolution and robustness for discontinuities. AIAA Paper 94-0083, January 1994.

[18]. J. J. Quirk. A contribution to the great Riemann solver debate. International Journal for Numerical Methods in Fluids, 18(6):555–574, 1994.

[19]. M. S. Liou. A sequel to AUSM, part II: AUSM+-up for all speeds. Journal of Computational Physics, 214:137–170, 2006.

[20]. R. J. Hakkinen, I. Greber, L. Trilling, and S. S. Abarbanel. The interaction of an oblique shock wave with a laminar boundary layer. NASA Memorandum 2-18-59W, 1959.

[21]. T. N. Eichmann, T. J. McIntyre, A. I. Bishop, S. Vakata, and H. Rubinsztein-

Dunlop. Three-dimensional effects on line-of-sight visualization measurements of supersonic and hypersonic flow over cylinders. Shock Waves, 16(4–5):299–307, 2007.

[22]. Mark G. Sobell. A Practical Guide to Linux Commands, Editors and Shell Programming. Prentice Hall, Upper Saddle River, New Jersey, 2005.

[23]. A.V. Aho, B.W. Kernighan, and P.J. Weinberger. The AWK Programming Language. Pearson, 1988.

A. Linux 下的命令行

对于在 Linux 机器上运行作业，有必要了解如何在命令行(一种命令解释器和编程语言)中进行操作。Sobell 的文本[22]是一个很好的信息来源，但这里有一些提示可以帮助您入门。

一个基本的命令是由一系列的单词组成，用空格隔开，并具有通常的形式：

cmd [options] arguments

其中：

- cmd 是执行这项任务的命令名称或应用程序的名称。Linux 上的命令名通常都很简洁，名称为两个或三个字符。
- options 是包含可选项的单词，通常在其前面有一个或两个破折号。如果默认的性能不是您想要的，那么它们将修改命令的特性。
- arguments 是需要解决的问题。如果这些是文件名，则通常可以使用通配符 wildcard 模式，它们可以一次匹配多个文件。

命令常常将它们的标准输出放到控制台。如果文本输出的数量过多，则可以将其重定向到文件或通过分页过滤器传输。后一个选项是将多个命令放在一起的示例，以便让一个命令的输出成为另一个命令的输入。一旦您理解了系统，自定义命令就可以通过这种方式简单地构建。下表总结了一些您在使用 Eilmer 时可能会发现有用的命令。

登录和退出

ssh *user@host*	以用户身份连接到指定主机。
ctrl+d	退出当前会话。
exit	退出当前会话。

获取帮助

man *cmd-name*	显示指定命令的手册页。
man *cmd-name* | less	通过分页过滤器显示手册页。
ls --help | less	查看 ls 命令提供的联机帮助。
man -k *keyword*	列出包含关键字的手册页。

apropos *subject*	列出主题手册页。

在您的文件夹中移动和查找

cd *dir*	切换到 dir 目录。
cd	切换到主目录。
cd..	切换到当前目录的上一级目录。
pwd	打印当前(工作)目录。
pushd *dir*	切换到新 *dir* 目录，将当前目录放到堆栈上。
popd	回到堆栈顶部的目录。
ls –l	列出当前目录中的文件，长格式。
ls -a ..	列出了上面目录中的文件，包括所有隐藏的文件。
du -h *dir*	报告目录及其子目录的大小。
df –h	报告文件系统的容量，以及每个目录使用了多少资源。
mkdir *dir*	创建新目录。
rmdir *dir*	移除空目录。

处理文件

cat *file*	显示文本文件的内容。
head -n 20 *file-to-show*	显示一个文本文件的前 20 行。
tail -f *file-to-show*	显示文件的最后几行，并在文件更改时继续显示这些行。
grep 'ideal' * .lua	在当前目录中的所有 Lua 文件中找到 ideal 字符串。
mv *src-file dest-file*	将源文件重命名为目标名称。
cp *src-file dest-file*	将内容从源文件复制到目标文件。
scp *src-file user@host:*	将文件从当地计算机中复制到移除了电脑 *host* 的 *user* 主目录。

rm -r *dir*	移除一个目录及其所有内容(递归地)
gzip *src-file*	文件压缩文件，添加.gz 扩展名。
tar -zcf *tarfile dir*	将 *dir* 目录的所有内容压缩到 *tarfile* 磁带文件。
tar -zxf *tarfile*	将 *tarfile* 磁带文件的内容解压到当前目录。

管理流程

top	显示所有运行进程的信息。可以很方便得找到哪些工作进程正在占有您所有工作站的 CPU 周期和内存。
Ctrl+z	停止当前命令。
bg	在后台恢复已停止的工作。
fg	给前台带来工作。
Ctrl+c	停止当前的命令。

命令行编辑

在大多数 Linux 系统中，似乎可以使用光标键在命令行中移动。删除和退格似乎也有合适的效果。

Ctrl+u	擦除整个命令行。
!!	重复最后一个命令。
history	显示命令历史
!*n*	重复命令 *n*。

B.关于 Lua 语言

我们使用 Lua 语言作为用户输入脚本的格式。它是一种非常通用的输入格式，在设置复杂的仿真时可以实现多方面的自动化，但是需要用户了解并熟悉 Lua 语言。我们的第一个建议是，坐下来喝一杯您最喜欢的饮料，花一个小时左右的时间阅读 Lua 书籍[12]中编程的入门部分。这本书的第一版，包含您需要的所有内容，可在网站 https://www.lua.org/pil/contents 找到。

这些时间的投入将得到成倍的回报，但是，如果您急于进行一些流动仿真，那么这一节可能仅能使您处于开始阶段。要在 Eilmer 环境中尝试少量 Lua 代码，但不需要设置完整的仿真，您可以使用专门的 Lua 脚本命令：

```
$ e4shared --custom-post --script-file=myscript.lua
```

其中 myscript.lua 是包含 Lua 代码的文件名称。

B.1 基础和语法

简短的注释以双破折号--开头，在行的末尾结束。通过使用[[…]]将这些行作为单个字符串组合在一起，长注释可以跨越多行，然后用--注释整个字符串。例如：

```
--[[ Long, multiple-line comments
can be used to cut sections of code
out of your script.]]
```

命令(或语句)是脚本中实际完成任务的行。例如：

```
-- Compute the area of a circle.
radius = 2.0
area = math.pi*radius^2
print("For radius=", radius, "area=", area)
```

如上例所示，标识符或名称可以是任何字符串(字母、数字或下划线)。名字不能以数字开头，但可以以下划线开头。如果您看到名称由单个下划线组成，那么它可能被用作虚拟变量。当我们讨论变量时，我们讨论的是与数据绑定的名称。

您将操作的数据类型包括：

- 数字
- 字符串
- 布尔值
- nil
- 函数
- 表格
- 用户数据

数字是浮点值。字符串可以用双引号或单引号分隔。布尔值要么为真，要么为假。在布尔型环境中，只有 false 和 nil 为有效的假状态。其他值，例如 0 (数值 0)，实际上是 true。这可能令人非常惊讶，特别是当您有 C 语言编程背景时。

特殊值 nil 什么也不表示，它与所有其他类型都不同。如果试图访问未初始化的变量，则该值为 nil。我们可以认为这个名称毫无意义。

下面将讨论函数和表，而用户数据是一种表示 C 类数据结构的类型，通过 C 语言编程界面实现。

B.2 运算符和表达式

表达式是操作数(数据)和运算符的组合，结果是某个类型的单个值。前面代码片段中 2*math.pi*radius 表达式就是一个例子。出现在这类表达式中的名称求得绑定到它们的值。

赋值操作符=用于将名称绑定到数据值，就像对半径和面积变量所做的那样。在 Lua 中，变量可以包含任何类型。类型是值的属性。

B.3 表

Lua 中几乎所有复杂的数据类型都被构造成某种形式的表。表文字由大括号{}分隔，其中的条目可以有数字指数或字符串键。例如：

```
t1 = {} -- an empty table
```

```
t2 = {["x"]=1.0, [1]=2.0, ["1"]=3.0} -- [1] is different to ["1"]
t3 = {1.0, 2.718, 3.142} -- an array
```

要访问表中的项，请使用方括号。例如 t2[1]等于 2，而 t3[1]等于 1。对于带有字符串键的指数项，有一个简写方法，即 t2["x"]和 t2.x 是等价的，它的值都是 1。如果试图访问不存在的条目，则会获得 nil 值。

B.4 函数

函数是一种抽象代码块的方法。它们是一类对象，可以匿名创建，然后分配给一个变量或从其他函数返回。下面是一个计算抛物线的简单函数。

```
myParabola = function(s)
    local x = s
    local y = s^2
    return x, y
end
xx, yy = myParabola(0.5)
print("for s=", 0.5, "xx=", xx, "yy=", yy)
```

myParabola 后面的调用运算符括号()可以调用函数。数值 0.5 作为惟一的参数来传递，并赋值给函数中的局部变量 s。然后就执行函数体中的命令。

注意，一个函数可以返回多个值，这些值可以并行地分配给多个变量。在第 11 页关于尖头圆锥的仿真介绍中，这种用于建立仿真的多重赋值有一个例子就是可以在气体模型中进行设置。在那里，我们捕获返回的化学物质数量、内能模式数量、以及对 GasModel 对象的引用，并将这三个项目分配给三个变量。

还要注意，在为函数中的 x 和 y 变量赋值时，要使用局部前缀。没有局部前缀分配的变量将默认为全局变量。有时，这种默认行为是很管用的，但如果您是从另一种编程语言来到 Lua 语言，就会感到惊讶。

B.5 基于对象的编程

在您的仿真中使用的底层 D 语言对象被包装在界面代码中，并使用基于对象的表示法呈现给 Lua 域中的脚本，该表示法将在 Lua 书中编程的后续章节进行描述。在这里，我们将提供一个在三维空间中构造几个点的例子，然后使用这些点来定义一个线段。

```
a = Vector3:new{x=0.0, y=0.0}
b = Vector3:new{x=0.2, y=0.0}
ab = Line:new{p0=a, p1=b}
print("line ab=", ab)
print("midpoint is at ", ab(0.5))
```

注意 new 方法名之前的冒号：字符。这些表示方法绑定到特定的对象。还要注意，给每个新方法的数据都包含在单个表文字中。当调用一个函数并提供一个恰好是表的参数时，可以省略调用运算符括号()。

B.6 控制语句

除非控制语句另有说明，否则脚本语句的处理将按顺序进行。代码块由关键字分隔，通常以关键字 end 结束。Lua 使用常见的复合语句结构，来控制脚本的执行流动。

if-then-else-end 结构可控制选择一个代码块或另一个代码块。例如，要有条件地执行某些代码，可以编写：

```
if condition then
    code-block
end
```

要在可选代码块之间进行选择，可以使用 else 和 elseif 关键字。例如：

```
if condition-1 then
    code-block-1
elseif condition-2 then
```

```
    code-block-2
else
    code-block-3
end
```

可以使用 while 和 for 循环结构重复执行代码块。像这样的 while 循环：

```
while condition do
    code-block
end
```

for 循环有两种形式。显示输入奇数到 9 的 for 循环数字，可以使用下面的例子：

```
for i=1, 9, 2 do
    print(i)
end
```

而通用型 for 循环对于迭代表条目很有帮助：

```
t = {["pi"]=3.14, e=2.71}
for k, v in pairs(t) do
    print("key=", k, "value=", v)
end
```

对于一个项目数组，可以使用 ipairs 迭代器处理相同的通用型 for 循环。它为每个值提供指数，如下：

```
t = {"a", 3.142, "b", 2.718}
for i, v in ipairs(t) do
print("index=", i, "value=", v)
end
```

B.7 Lua 中的全局符号

在 Eilmer 处理您的 Lua 输入脚本之前，有许多符号插入到 Lua 解释器的全局名称空间中。这些一般用于构造流动条件和流域的描述，以及设置仿真配置参数。可以使用以下命令行从您的脚本中获得这个名称列表：

```
for k,_ in pairs(_G) do print(k) end
```

这将显示在全局名称空间表_G 中找到的所有名称。为了避免不愉快的意外，不应该将这些名称重新绑定到脚本中的其他对象。

C.了解 AWK 脚本

我们使用 AWK 编程语言[23]作为文本文件的可编程过滤器。当 AWK 程序处理输入文件时，它将文本分成记录和字段。默认情况下，记录由换行符分隔。记录中的字段由空格符分隔。对于每个记录，可以选择字段用于计算并显示。

这里再次使用 AWK 脚本提取第 3.2 节中尖头圆锥仿真的压力演变历史。左边距中的行号不是脚本的一部分。

```
1 # cp.awk
2 # Scan a history file, picking out pressure and scaling it
3 # to compute coefficient of pressure.
4 #
5 # PJ, 2016-09-22
6 #
7 BEGIN {
8 Rgas = 287.1; # J/kg.K
9 p_inf = 95.84e3; # Pa
10 T_inf = 1103; # K
11 rho_inf = p_inf / (Rgas*T_inf)
12 V_inf = 1000.0; # m/s
13 q_inf = 0.5*rho_inf*V_inf*V_inf
14 print "# rho_inf=", rho_inf, " q_inf=", q_inf
15 print "# t,ms cp"
16 }
17
18 $1 != "#" {
19 t = $1; p = $10
20 print t * 1000.0, (p - p_inf)/q_inf
21 }
22
23 END {}
```

它与以下命令行一起使用：

```
$ awk -f cp.awk hist/cone20-blk-1-cell-20.dat > cone20_cp.dat
```

来扫描演变历史数据文件 hist/cone20-blk-1-cell-20.dat，通过标准输出重定向，将其压力系数结果写入 cone20_cp.dat 文件。

除了以#字符开头并一直持续到行末尾的注释之外，上面的 AWK 程序也由 pattern { action }的模式-操作语句组成。在扫描输入文本文件时，将根据模式检查每一行，并执行任何匹配模式的操作。丢失的模式总是匹配的，丢失的操作将

导致显示整行。

BEGIN 和 END 模式都是特殊的，不出所料，它们的操作是在开始程序执行(在处理任何记录之前)和结束程序执行 (在处理所有记录之后)时才执行。我们使用 BEGIN 操作(第 8 行到第 16 行)来设置一些已命名常量，并向标准输出显示一个标题。此标题便于以后提醒我们计算数字的含义。

只要第一个字段(用$1 表示)不等于由单个字符#组成的字符串，那么执行有趣任务的操作将从第 18 行开始，并使用匹配的模式。我们经常在数据文件中使用这个字符来表示命令，而不是数据，因为 GNUPlot 可接受混合了这些注释的数据。一旦模式匹配，就对记录进行操作。首先，字段 1 被赋值给变量 t，以秒表示时间的单位，字段 10 被赋值给变量 p，以帕斯卡表示静压的单位。我们将时间单位标为毫秒，并将静压转换为压力系数，该系数由之前计算的动压值标准化。print 将结果发送给标准输出，我们已经将其重定向到 cone20_cp.dat 结果文件。

AWK 编程语言还有很多。您可以拥有基于常规表达式和 start、stop 其他形式的模式。您的操作可能包括使用编程结构(如 if 语句和循环)进行任意复杂的计算。当 AWK 代码变得越来越复杂时，您可能需要定义自己的专用函数。这些将与关键字 function 和以下形式一起介绍：

function *name*(*parameter-list*) { *statements* }

也可以访问特殊的变量，比如 NR（当前记录编号）以及 NF（当前记录中字段的数量）。在 5.1.5 节中有一个稍微复杂的脚本，用于计算平板上剪切应力。

D.简单气流函数

在主程序的 Lua 环境中，您可以访问许多用于计算简单气流关系的函数。在仿真准备过程中、在用户定义的边界条件内或在后处理活动期间设置流动条件时，使用这些函数会很方便。

D.1 理想气体

第一组函数适用于理想气体的简单流动情况。有关使用其中一个函数的示例，请参见第 21 页。所有的函数都包含在一个 idealgasflow 表中，但是根据流动情况，它们分组如下：

D.1.1 简单等熵流动

- A_Astar(M, g = 1.4)返回面积比$\frac{A}{A^*}$给马赫数 M，以及比热容比 g。如果不提供 g 的值，则使用默认值 1.4
- T0_T(M, g = 1.4)返回总温与静温比$\frac{T_0}{T}$
- p0_p(M, g = 1.4)返回总压与静压比$\frac{p_0}{p}$
- r0_r(M, g = 1.4)返回密度比$\frac{\rho_0}{\rho}$

D.1.2 正常的激波关系

- m2_shock(M_1, g = 1.4)返回经激波处理后的马赫数。这个参照系的激波是静止的，来流是超声速气流，马赫数是 M_1。
- r2_r1 (M_1, g =1.4)返回通过激波的密度比$\frac{\rho_2}{\rho_1}$。状态 2 是经过激波处理的状态。
- ou2_u1 (M_1, g = 1.4)返回通过激波的速度比$\frac{u_2}{u_1}$
- p2_p1 (M_1, g =1.4)返回通过激波的压力比$\frac{p_2}{p_1}$
- T2_T1 (M_1, g = 1.4)返回通过激波的静态温度比$\frac{T_2}{T_1}$
- p02_p01 (M_1, g = 1.4)返回通过激波的总压比$\frac{p_{02}}{p_{01}}$

• DS_CV (M_1 , g = 1.4)返回通过激波的标准熵变化$\frac{s_2-s_1}{c_v}$

• pitot_p (M_1 , g =1.4)返回气流的皮托管压力(通过激波处理后)

D.1.3 加入热量的一维流动

• T0_T0star(M, g = 1.4)返回总温度比$\frac{T_0}{T_0^*}$给马赫数 M，以及比热容比 g。T_0是局部的总温度值，T_0^*是总温度（在假想的临界点处，其中马赫数为 1 添加了足够的热量）。

• M_Rayleigh (T_r, g = 1.4)返回给定总温度比$T_r = \frac{T_0}{T_0^*}$时的局部马赫数。

• T_Tstar (M, g= 1.4)返回静态温度比$\frac{T}{T^*}$

• p_pstar (M, g= 1.4)返回静态压力比$\frac{p}{p^*}$

• r_rstar (M, g = 1.4)返回密度比$\frac{\rho}{\rho^*}$

• p0_p0star(M, g= 1.4)返回总压比$\frac{p_0}{p_0^*}$

D.1.4 超声速转弯流(等熵)

• PM1 (M, g = 1.4)返回给定马赫数的 Prandtl-Meyer 函数 ν(以弧度为单位)

• PM2 (ν, g =1.4)返回给定 Prandtl-Meyer 值(以弧度为单位)的马赫数。

• MachAngle(M)返回以弧度表示的马赫角μ。

D.1.5 斜激波关系

• beta_obl(M_1 , θ, g = 1.4, tol = 1.0e-6)返回相对于未偏转自由流动方向的斜激波角β(以弧度为单位)。所需的输入包括自由流动马赫数 M 和气流偏转角θ(以弧度为单位)。

• beta_ob12 (M_1, $\frac{p_2}{p_1}$, g = 1.4)返回给定自由流动马赫数和通过斜激波静压比下的激波角。

• theta_obl(M_1, β, g =1,4)返回给定激波角β(以弧度为单位)的气流偏转角θ(以弧度为单位)。

• M2_obl (M_1, β, θ, g = 1.4)返回斜激波后的马赫数

• r2_r1_obl (M_1, β, g =1.4)返回通过斜激波的密度比$\frac{\rho_2}{\rho_1}$

• Vn2_Vn1_obl (M_1, β, g =1.4)返回通过斜激波的正常速度比$\frac{V_{n2}}{V_{n1}}$

• V2_V1_ob1 (M_1, β, g = 1.4)返回通过斜激波的气流速度比$\frac{V_2}{V_1}$

• p2_p1_obl (M_1, β, g = 1.4)返回通过斜激波的静压比$\frac{p_2}{p_1}$

• T2_T1_ob1 (M_1, β, g = 1.4)返回通过斜激波的静温比$\frac{T_2}{T_1}$

• p02_p01_obl (M_1, β, g = 1.4)返回通过斜激波的总压比$\frac{p_{02}}{p_{01}}$

D.1.6 锥形激波流动

• theta_cone (V_1, p_1, T_1, β, R=287.1, g=1.4)返回四个值，θ_c, V_c, p_c, T_c，分别指定圆锥表面的角度、表面速度、表面压力以及表面温度。所需的输入包括自由气流速度 V_1、静压 p_1、静态温度 T_1，以及相对于自由流动方向的锥形激波角β(以弧度为单位)。

• beta_cone (V_1, p_1, T_1, θ, R =287.1, g = 1.4)返回给定自由流动条件(以 SI 规定的为单位)和圆锥表面角度θ(以弧度为单位)的激波角β(以弧度为单位)。

• beta_cone2 (M_1, θ, R=287.1, g = 1.4)返回给定自由流动马赫数和圆锥表面角度θ(以弧度为单位)的激波角β(弧度)。

D.2 一般气体

第二组函数用于更一般气体的简单流动情况，使用内置的气体模型之一。这些函数对在 GasModel 环境中构造的 GasState 对象进行操作

D.2.1 等熵稳定流动

• state1, V = gasflow.expand_from_stagnation(state0, p_over_p0)返回给定滞止状态(0)和压力比的流动状态(1)。V 是膨胀气体的速度，假设是等熵、稳态过程。

• statel V = gasflow.expand_to_mach (stateo, mach)返回特定马赫数下的膨胀气体流动状态。

• state0 = gasflow.total_condition(statel, V1)返回给定自由流动条件下、假设为等熵状态、稳态处理的滞止状态。

• state2pitot = gasflow.pitot_condition (statel, V1)返回皮托管滞止状态。自由气流(1)可能是超声速的，这样在自由气流和滞止区之间就会形成正常的激波。

• state2, V2 = gasflow.steady_flow_with_area_change (statelV1, A2_over_Al, tol) 返回由流管区域变化引起的稳定等熵过程所产生的流动状态。区域变化指定为比率$\frac{A_2}{A_1}$，以及 tol 的默认值为 10^{-4}。

D.2.2 非稳态等熵流动

• state2, V2 = gasflow.finite_wave_dp(state1, V1, characteristic, p2, steps) 返回非稳态等熵过程后的新压力 p2 状态(2)。该过程遵循“cplus”或“cminus”特性。步骤的数量值默认为 100。

• state2, V2 = gasflow.finite_wave_dp(state1, V1, characteristic, V2_target, steps, Tmin) 返回非定常等熵过程后的新的速度 v2_target 状态(2)。该过程遵循“cplus”或“cminus”特征。步骤的数量值默认为 100。Tmin 的默认值为 200.0K，如果气体温度降低到低于此值，则将终止步进过程。

D.2.3 正常的激波关系

• state2, V2, Vg = gasflow.normal_shock(state1, Vs, rho_tol, T_tol)返回激波处理后的条件。参照系中有滞止的激波，来流(1)速度为 Vs，是超声速的。Vg 是实验室参照系中后激波(2)的气体速度，在这个参照系中激波以速度 Vs 通过，而成为滞止气体。

• V1,V2,Vg =gasflow.normal_shock_p2p1 (statel, p2p1)返回给定通过激波压力的前激波速度和后激波速度。

• state5, Vr = gasflow.reflected_shock(state2, Vg)返回后反射激波的条件，该激波是给定了瞬时激波后的流动条件。这两个速度都是在实验室坐标系中，其中初始(1)和末尾(5)气体速度为零。

D.2.4 斜激波关系

• state2, theta, V2 = gasflow.theta_oblique(state1, V1, beta)返回后激波条件(2)，给相对于自由流的流动方向指定的自由气流条件(1)和激波角度 beta(以弧度为单位)。这个 theta 角度是关于流动的偏转角，单位也是弧度。

• beta = gasflow.theta_oblique(state1, V1, theta)返回相对于自由气流(1)流向的激波角(以弧度为单位)。

D.2.5 使用实例

下面的例子展示了使用气体流动函数来计算最接近 T4 激波隧道一个操作条

件的流动条件测试。使用 CEA 气体模型，采用 13 种空气参数，并按状态进行处理。

```
model = "CEAGas"

CEAGas = {
mixtureName = 'air13species',
  speciesList =
 {"N2","O2","Ar","N","O","NO","Ar+","NO+","N+","O+","N2+","O2+","e-"},
  reactants = {N2=0.7811, O2=0.2095, Ar=0.0093},
  inputUnits = "moles",
  withIons = true,
  trace = 1.0e-6
}
```

```
-- reflected-shock-tunnel.lua
-- Run with a command like:
-- $ e4shared --custom-post --script-file=reflected-shock-tunnel.lua
-- PJ, 2017-11-12
print("Compute the test-flow conditions for a shot in the T4 shock tunnel.")
--
print("shock-tube fill conditions")
gm = GasModel:new{'cea-air13species-gas-model.lua'}
state1 = GasState:new{gm}
state1.p = 125.0e3; state1.T = 300.0
gm:updateThermoFromPT(state1)
print("state1:"); printValues(state1)
--
print("normal shock, given shock speed")
Vs = 2414.0
state2, V2, Vg = gasflow.normal_shock(state1, Vs)
print("V2=", V2, "Vg=", Vg)
print("state2:"); printValues(state2)
--
print("reflected shock")
state5, Vr = gasflow.reflected_shock(state2, Vg)
print("Vr=", Vr)
print("state5:"); printValues(state5)
--
print("Expand from stagnation (with ratio of pressure to match observation)")
state5s, V5s = gasflow.expand_from_stagnation(state5, 34.37/59.47)
print("V5s=", V5s, " Mach=", V5s/state5s.a)
print("state5s:"); printValues(state5s)
```

```
print("(h5s-h1)=", gm:enthalpy(state5s) - gm:enthalpy(state1))
--
print("Expand to throat condition (Mach 1.0001)")
state6, V6 = gasflow.expand_to_mach(state5s, 1.0001)
print("V6=", V6, " Mach=", V6/state6.a)
print("state6:"); printValues(state6)
--
print("Mach 4 nozzle expansion to test-flow condition.")
state7, V7 = gasflow.steady_flow_with_area_change(state6, V6, 27.0)
print("V7=", V7, " Mach=", V7/state7.a)
print("state7:"); printValues(state7)
```

E.指定特定工作时间的用户自定义函数

用户自定义函数(UDFs)是用 Lua 编写的可调用函数，用于执行不属于标准流动求解器的特定和/或自定义任务。这些可调用的函数可用于：

•作为指定的边界条件;

•添加自定义源项;

•规定网格运动;

•和在每个时间步进开始和结束时执行特殊操作。

下面的一些例子给出了更具体的形式。一个指定的边界条件可以模拟多孔边界的质量注入，而多孔边界目前在仿真代码中还没有作为边界条件。当使用产生计算结果的方法测试代码时，我们使用定制的源项。每个时间步长开始时的可调用函数可以用来计算一个特殊的流场变量。

您的每个规范形式都是 Lua 代码，通常是一个函数的形式：

```
function specifiedName(args)
    -- Do something interesting,
    -- perhaps making use of args,
    -- then return a value.
    return requiredValue
end
```

主仿真代码设置 args，具体到调用的时间和地点。然后，它请求 Lua 解释器执行 specifiedName 函数，并传递 args。函数中的 Lua 代码执行任何需要的自定义计算并返回 requiredVal。仿真代码在随后的计算中使用 requiredVal。当然，您的 Lua 脚本中可能还有其他支持的代码。整个脚本文件在启动时进行解释，因此可以定义变量和函数，读取文件并设置表。您可以随时使用 Lua 解释器的全部功能。

E.1 自定义边界条件

用户可以选择设置对流边界条件和扩散边界条件。自定义的边界条件可以直接提供虚拟单元格的流动状态和界面值或流量。使用自定义边界条件需要两个步

骤:

1. 在 FluidBlock 设置中设置 UserDefinedGhostCellBC 或 UserDefinedFluxBc 边界条件(第 63 页)。
2. 构造定义边界条件运行状况的 Lua 文件。

E.1.1 设置虚拟单元格和界面属性

当用户的(Lua)输入脚本构造一个新的UserDefinedGhostCellBC 边界条件时，将指定一个 Lua 文件名。在边界条件构造时翻译这个 Lua 文件，它至少需要定义 Lua 函数 ghostCells()。对于粘性情况的仿真，还需要定义 Lua 函数 interface()。在仿真进行过程中，每当应用边界条件时，将调用这些特定的函数。实际上，是通过沿着边界的每个界面来调用它们的；这些函数通过界面到界面来工作。除了提供 expected 函数之外，Lua 文件还可能含括用户希望包含的任何支持代码。它可以启动外部进程，读取数据文件，或者设置数据，以供之后在边界条件函数中使用的任何其他合适的活动。

要设置用户自定义的边界条件，您需要指示代码如何执行对流(无粘)更新，然后分别执行粘性效应。可以通过定义 ghostcells (args)函数来处理边界上的无粘相互作用。在本例中，您填充了两个虚拟单元格的属性，以便它们在边界处提供所需的对流流动效应。虚拟单元格是抽象的，因为它们不存在于所模拟的流域，而是存在于每个块边界的程序数据中。对于沿边界的单元格面，它们用于对流更新的内插阶段。该函数返回两个流动数据表。第一个表是内部虚拟单元格，也就是最靠近流域边缘的那个。第二个表用于外部虚拟单元格。

通过定义 interface()函数来处理边界处的粘性效应。在本例中，您直接在界面上设置属性，作为粘性更新的一部分，主代码将根据这些指定的流动属性计算空间导数。例如，可以使用 interface()函数为无滑移壁设置界面温度和零速度。这样做，您不会直接控制粘性热流量直接进入流动，但它会通过设置温度来间接控制。注意，在无粘仿真中，将忽略用户指定的粘性边界效应函数；代码不会调用它们。

虚拟单元格

如果运行时被调用，ghostCells (args) 函数将返回两个 Lua 表。编写函数的用户负责构造和返回这两个表。第一个表包含最靠近边界面的虚拟单元格中的流动

状态，第二个表包含离边界面较远的虚拟单元格的流动状态。如果您不希望从您的用户定义函数中设置虚拟单元格数据，那可以返回空表。这听起来可能是一件奇怪的事情，但在某些情况下，当由交换操作设置虚拟单元数据时，您可能会有条件地设置它。(可以查看 ExchangeBC_FullFacePlusUDF 边界条件)

args 表中提供的项目包括更新日期的一些状态信息、虚拟单元格的一些几何信息、边界界面、相邻的内部单元格。

t：当前仿真时间，单位为秒

dt:当前时间步进，单位为秒

timeStep:对时间步进计数的积分

gridTimeLevel:积分值，表示移动网格的更新阶段

flowTimeLevel:积分值，指示流动更新阶段

boundaryId:积分值，指示我们工作块内的那个边界。在一个结构网格块中，使用 north 符号索引会很方便。

x, y, z:界面中点的坐标，单位为米

csX csY, csZ:方向余弦的单位法向界面

csX1 csY1, csZ1:第一切线的方向余弦界面

csX1 csY1, csZ1:第二切线的方向余弦界面

i，j, k:相邻界面内部的单元格指数

gc0x gcOy, gcOz:第一个虚拟单元格的中心坐标

gc1x gc1y, gc1z:第二个虚拟单元格的中心坐标

如果您需要关于相邻内部单元格内的流动状态信息，请使用 sampleFluidCell 服务函数来获得第 128 页所示的数据。出现在返回表的项目有：

p:气体压力，单位为 Pa(要求必须是这样)

T:反式旋转温度，单位为开尔文(要求必须是这样)

T_modes:与其他内能模式相关的温度数组。这只在 n_modes 为非零的情况下才需要该项目。

massf:指定质量分数的表。只有当 n_species 大于 1 时才需要该项目。对于单一种类仿真，唯一的种类默认值为 1.0。

velx,vely,velz:在 x,y,z 方向上的速度分量，单位为 m/s。它们的默认值都是 0.0。

tke:单位质量的湍流动能，单位为 J/kg。默认是 0.0。

omega: k-ω 湍流模型的 ω，单位为 1/s。默认是 1.0。

S:激波检测器值，要么为 1.要么为 0。默认值为 0。

其他气体热力学数据，如内能和声速，都是通过气体模型得到的。

注意，对于沿着边界的每个单元格，您的 ghostCells 函数都只调用一次，因此要注意在整个边界上重复固定计算的可能性。在为第一个单元格调用函数时进行一次计算，并将结果数据存储在全局变量中，这样做可能很有效的，以便在后续调用中使用。

界面

如果粘性效应是活动的，则调用 Lua 函数 interface(args)，以在边界界面上正确地获得一些属性。这些属性将返回到表中，包含以下内容：

p:气体压强，单位为 Pa

T:反式旋转温度 单位为开尔文温标

T_modes:与其他内能模相关的温度数组

massf:质量分数表

velx,vely,velz:在 x,y,z 方向速度分量，单位为 m/s

tke:单位质量的湍流动能，单位为 J/kg

omega: k-ω 湍流模型的 ω，单位为 1/s

mu_t:湍流粘度，单位为 Pa.

k_t:湍流导热系数

注意，可能不需要所有参数，因此调用的 D 代码将所有参数视为可选，只从表中提取它可以找到的条目。这些提供的值覆盖了边界面上 FlowState 对象相应的值。在进入函数时，args 包含了与调用 ghostCells 函数中相同的所有属性。此外，args 包含：

t:当前仿真时间，单位为秒

dt:当前时间步进，单位为秒

timeStep:对时间步进计数的积分

gridTimeLevel:积分值，表示移动网格的更新阶段

flowTimeLevel:积分值，指示流动更新阶段

boundaryId:积分值，指示我们工作块内的那个边界。在一个结构网格块中，使用 north 符号索引会很方便。

x, y, z:界面中点的坐标，单位为米

csX csY, csZ:方向余弦的单位法向界面

csX1 csY1, csZ1:第一切线的方向余弦界面

csX1 csY1, csZ1:第二切线的方向余弦界面

i，j, k:相邻界面内部的单元格指数

请记住，这些函数是在 Lua 解释器环境中求值的，而 Lua 解释器环境是在具体化边界条件时建立的，因此当时存储的任何数据现在都可以用于函数，这应该是通过全局变量来实现的。这允许一个有用的模式，例如，可以从设置时间的文件中读取流动配置文件数据，并将其存储在一个表中，然后可以对 ghostCells 和界面进行后续调用。

E.1.2 直接指定流量

用户可以提供一个 convectiveFlux (args) 函数来返回一个指定组合(对流和粘性)界面流量的表，而不是指定在仿真计算代码内部用于计算跨边界的质量动量和能量流量的虚拟单元格和界面数据。可以通过调用边界条件构造函数 UserDefinedFluxBc: new{}(见 63 页)中的 funcName 参数来配置 convectiveFlux 名称。返回的流量表包含以下条目：

mass:单位界面面积的质量流量

momentum_x: x 方向上单位面积的动量通量

momentum_y: y 方向上单位面积的动量通量

momentum_z: z 方向上单位面积的动量通量

total_energy:单位面积能量通量

species: nsp 类质量通量的表

以及输入的 args 表包括：

t:当前仿真时间，单位为秒

dt:当前时间步进，单位为秒

timeStep:对时间步进计数的积分

gridTimeLevel:积分值，表示移动网格的更新阶段

flowTimeLevel:积分值，指示流动更新阶段

boundaryId:积分值，指示我们工作块内的那个边界。

x, y, z :界面中点的坐标，单位为米

csX csY, csZ:方向余弦的单位法向界面

csX1 csY1, csZ1:第一切线的方向余弦界面

csX1 csY1, csZ1: 第二切线的方向余弦界面

i，j, k:相邻界面内部的单元格指数

在设置流量值时，用户负责给出垂直于边界界面的流量大小。因此，在笛卡尔坐标系(nx, ny, nz)中，为用户的函数提供了界面法向量的分量，以帮助计算任意方向界面的正确流量大小。图 E.1 所示为在结构化网格上，二维边界的单位法线的正方向。简而言之，如果法线指向内部，则表示西部和南部边界，如果法线指向外部，则表示东部和北部边界。例如，如果您正在设置一个穿过北部边界并进入流域的流量，其值的大小应该是 negative 的，以表示流入域的流量。东部边界的流量也是如此。

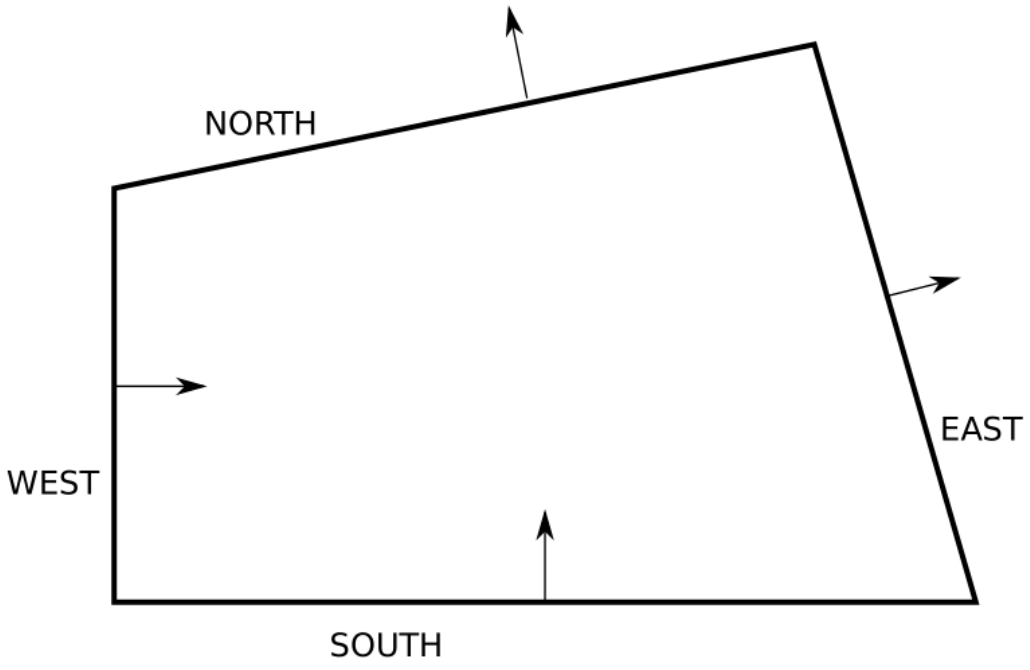


图 E.1:使用一个结构化网格块，在二维中每个边界上单位法线的正方向图。

这样安排的原因是结构化网格的面法线是内部代码，所有东西方向界面是单个数组 i-faces 的一部分。对于北部和南部，有单个数组 j-faces，对于顶部和底部

的面，有数组 k-faces。因此，一个单一的 i-face 将作为一个单元格的东面和下一个单元格的西面，来给它的右边。

E.2 源项

将用户自定义的源项添加到内部计算的源项中。这些项规定了每个单位体积数量的增长率。它们可以适用于所有的守恒方程，但如果要激活这些项的增长，您需要设置配置选项：

config.udf_source_terms = true

config.udf_source_terms_file = 'my_source_terms.lua'

在指定的 Lua 文件中，您需要为域中的每个单元格提供 sourceTerms (t, cell) 函数，该函数在更新的每个阶段都要经过一个时间步长的调用。这里 t 是当前已模拟时间，cell 是一个包含单元格数据的表，由 sampleFluidCell 函数返回对应的值(参见 E.5.5 节)。添加到该表中的项包括：

blkId：包含单元格的块指数

i i k：是单元格的指数。请注意，这些指数存在于存储空间，其中 imin≤i≤imax。函数返回值组成的表可能包含以下条目：

mass：单位单元格体积所增加的质量速率

momentum_x: x 方向上单位体积动量的通量

momentum_y: y 方向上单位体积动量的通量

momentum_z: z 方向上单位体积动量的通量

total_energy:每单位体积的能量增长率

species:命名质量增长种类的 nsp 表或 ifnspecies==1 时的单个值

renergies: nmodes 表的能量通量

romega: k-ω 湍流模型的角速度增加率

rtke:湍流动能的增加率

E.3 网格运动

通过网格顶点的运动来控制网格运动。最灵活的安排是通过assignVtxVelocities(t, dt)函数来提供域内所有顶点的速度。速度为零的顶点可以忽略。要激活网格运动特性，您需要在仿真输入脚本中设置以下配置选项：

content.config.gasdynamic_update_scheme = ’moving_grid_1_stage’

config.grid_motion = ’user_defined’

config.udf_grid_motion_file = ’grid-motion.lua’

对于 gasdynamic_update_scheme，您还可以选择每个'moving_grid_2_stage'的可选步骤来获得二级更新方案。

在 assignVtxVelocities 函数中，您可以使用以下函数来获取单个顶点的当前位置并设置它们的速度。Lua 函数需要提供顶点的速度，而不是位置。

pos = getVtxPosition(blkId, i, j, k):以指定的笛卡尔坐标系[x, y, z]表的形式返回顶点的位置

x, y, z = getVtxPositionXYZ(blkId, i, i, k):以三个浮点值的形式返回顶点的位置

setVtxVelocity(vel, blkId, vtxId):在 blkId 块中设置顶点 vt xId 的速度向量。参数 vel 将作为一个 Vector3 对象提供。这适用于结构化和非结构化网格

setVtxVelocityXYZ(velx, vely, velz, blkId, vtxId):在 blkId 块中设置顶点 vtxId 的速度分量。速度分量以浮点值形式提供。这适用于结构化和非结构化网格

setVtxVelocity(vel, blkId, i, j):在二维结构化网格中为 blkId 块的 i, j 顶点设置速度向量

setVtxVelocityXYZ(velx, vely, velz, blkId, i, j):在二维结构化网格中为 blkId 块的 i, j 顶点设置速度分量

setVtxVelocity(vel, blkId, i, j, k):在二维结构化网格中为 blkId 块的 i, j,k 顶点设置速度向量

setVtxVelocityXYZ(velx, vely, velz, blkId, i, j, k):在二维结构化网格中为 blkId 块的 i, j,k 顶点设置速度分量

对于同时设置整个块或整个域的顶点速度，例如刚体运动，有些函数可以设置块中所有顶点速度：

setVtxVelocitiesForDomain(vel):为域中所有块的所有顶点设置一个速度向量

setVtxVelocitiesForBlock(blkId, vel):为块 blkId 中的所有顶点设置一个速度向量

setVtxVelocitiesForBlockXYZ(blkId, velx, vely, velz):为块中的所有顶点设置一组速度分量

setVtxVelocitiesForRotatingBlock(blkId, omega):设置转速 omega (单位是 rad/s)绕 z 轴旋转。

setVtxVelocitiesForRotatingBlock(blkId, omega, point):设置转速 omega (单位是 rad/s)用于通过 point 绕(0,0,1)轴的旋转

setVtxVelocitiesByCorners(blkId, p0vel, p1vel, p2vel, p3vel):设置由四个角速度指定的块角速度的速度分量。这适用于二维和三维网格，但在三维中，k 方向没有变化。

setvtxvelocitiesBycorner (blkId, p0vel, p1vel, p2vel, p3vel, p4vel, p5vel, p6vel, p7vel):设置带有角速度的块的速度向量

E.4 协同作用

在您的输入脚本中，如果您为 config.udf_supervisor_filei 提供了一个文件名，流动仿真程序将调用您在 Lua 文件中定义的 atTimestepstart (t, step, dt)函数。这个函数不同于边界条件函数和源项函数，因为它只在更新过程开始时调用一次，而不会将任何数据返回到仿真中。但是，它可以通过采样流场数据、运行外部流程和编写在其活动过程中可以读取文件的其他 Lua 解释器，来协调其他用户定义的函数。

在调用您的 atTimestepStart 并返回到 D 域之后，有几个操作可能会影响模拟的其他部分。第一个是检查 Lua 变量 dt_override。如果它是非 nil，它的值将覆盖当前的时间步长。当您有重大和突然的变化，如在激波管模拟隔膜破裂，它将帮助立即调整时间步长。注意，在对 atTimestepStart 函数的后续调用中，需要显式设置 dt_override=nil，以避免后续的时间步长被覆盖。第二件事是复制 userPad 数组的当前值，并将这些值传播给所有其他活动的 Lua 解释器，如 5.1 节所述。

在仿真循环中，如果在您的管理脚本中定义了用户自定义的 Lua 函数 atTimestepEnd(t, step, dt)和 atWriteToFile(t, step, dt)，那么在这些地方也会调用它们。写入文件的时间是指将流动解决方案数据写入磁盘的时间。在仿真过程中，这种情况可能会发生几次，因此调用需要当前数据以与流场产生相互影响的外部程序是一个很好的选择。Ingo Jahn 的瞬态固体力学求解器是外部程序与流动求解器耦合的一个好例子。

E.5 辅助变量、函数和模块

代码为您的用户自定义函数带来了许多便利，并且在 Lua 解释器中将它们设置为全局变量。例如，代码设置了 D-language 气体模型的引用，以便从 Lua 函数中使用。它还提供了有关块数据的一些信息，这些信息是特定的用于用户自定义边界条件的块。

E.5.1 变量

如果在运行的输入脚本中指定 userPad 数组的长度为非零，那么将在每个 Lua 解释器中的全局名称空间将该数组设置为可用。在常规的 Lua 规定中，数组只包含浮点数，并且元素的指数是从 1 开始的。您使用它们的目的完全取决于您，但是，其目的是在(可能有许多)Lua 解释器之间提供通信，而不需要在 Lua 函数中进行文件读写。

在输入脚本中，您可以设置数组值部分或全部的长度。没有提供初始值的元素将赋予零值。例如：

```
config.user_pad_length = 6
user_pad_data = {1.0, 3.14}
```

当 x = userPad[1] y = userPad[2] 可以从 Lua 脚本访问这些数据。

由于数组只是 Lua 名称空间中的一个表，所以可以为元素分配新值。从 atTimestepstart 函数调用返回，userPad 内容被复制回 D-language 域，并广播给所有可用于设置源项或边界条件的其它 Lua 解释器。在 MPI 仿真中，主任务(排名为 0)支配并将其值广播给其它所有 MPI 任务。从其他任何 Lua 函数调用(比如边界条件)返回时，不会将 userPad 数据复制回 D-language 域。在下一次调用 Lua 函

数时，主 userPad 将再次被推入 Lua 解释器的 userPad 副本。这允许数据的单向同步，这在仿真期间可能会发生改变。

如果您希望为许多 Lua 解释器提供公共数据，并且在模拟期间这些数据保持不变，那就可以方便地编写一个小型并包含一些赋值语句的 Lua 文件，然后使用 Lua dofile()函数。这将评估该文件的内容，并将这些内容分配到您的解释器中。还有许多其他全局自定义变量，包含用于仿真的配置数据。它们是：

blkId:当前块的索引

边界条件存在于块环境中，这意味着在 UDFs 中可访问的信息仅限于块中包含的信息和少量全局数据。这对于并行(MPI)仿真特别重要，因为存在的块是单独的进程，而一个块中的数据一般在另一个块中是不可用的。

gmodel:与当前仿真相关的气体模型对象

此对象提供访问完整的气体模型服务。您可能需要得到一些气体的性质或者计算一个状态方程。这个对象就有助于实现这一点，通过使用这个对象，您可以确保与仿真内部的气体模型保持一致。有关气体模型对象提供服务的更详细描述，请参见所附的 Gas Model User's Guide [5]。

n_species:种类数

n_mode:储能模式的数目(和温度)

加上描述拥有当前 Lua 解释器的块的数据。这些数据列在 E.5.3 节中。

在有边界条件的 Lua 解释器环境中，还有以下变量：

boundaryId:指示用户自定义函数应用于哪个边界号

boundaryLabel:用户给边界的标签(或设置为默认)

boundaryType:边界的类型(作为字符串)，这是可选的设置，由用户在输入脚本设置或给定为默认设置。

boundaryGroup:边界所属的组(也是一个字符串)

请注意，为边界条件启动的 Lua 解释器只用于附加到特定块的边界。因此，仿真中每个用户定义的边界条件都有自己的解释器状态，独立于其它所有的解释器状态。可以使用相同的 Lua 脚本初始化多个边界条件，使它们具有共同的特性。为了在解释器之间协调计算，可能会用到参数 boundaryId、boundaryLabel、boundaryType 和 boundaryGroup。

E.5.2 函数

除了指定变量中的数据，还可以调用几个函数来获得更多关于特定位置的流动信息：

infoFluidBlock(blkId):返回关于指数为 blkId 块的信息表。

sampleFluidFace(faceType、blkId、i、ji、k):返回网格中一个界面的几何和流动属性的信息表。参数 faceType 是一个字符串。在结构化网格中，faceType 是字符串“i”、“j”或“k”中的一个，在非结构化网格中，使用“u”表示 faceType。

sampleFluidCell(blkId,i,j,k):返回特定单元格的流动状态表。E.5.5 节中描述了向用户提供的数据。

getRunTimeLoads(loadsGroup):返回两个用于指定的 loadsGroup 名称表，一个用于力的分量，另一个用于力矩的分量。loadsGroup 名称是一个字符串，它与在输入脚本中构造边界条件时设置的一个组匹配。

blkId 指的是有意思的块。您应该将查询限制在块中，该块位于与用户自定义边界条件的当前块相同内存空间中。i j k 是有趣的界面指标。虽然应该提供所有值，但不是所有值都有意义。对于非结构化网格，只有 i 值有意义。你可以为 j 和 k 提供一个 0 值，因为这些指标被忽略了。在二维结构化网格中，k 没有意义。同样，只提供 0 值。在三维结构化网格中，所有 i、j、k 都需要有效值。在 MPI 并行仿真中，确保只从当前 MPI 任务拥有的块中取样。从其他块取样可能会导致分割错误。

E.5.3 由 infoFluidBlock 返回的数据

dimensions：此仿真的空间维数

label:字符串标签，可能用于标识块

grid_type:字符串“结构化的”或“非结构化的”

ncell:块内部单元格数目

nfaces:定义块内单元格的界面数目

nvertices:定义块内单元格的顶点数目

对于结构化网格块，有一些附加项是可用的，因为它们只有在结构化网格指数方向的环境中才有意义。它们是：

nicell:在 i-方向的单元格数目

njcell:在 j 方向的单元格数目

nkcell:在 k 方向的单元格数目

imin, imax:在存储空间的 i 指数范围

jmin, jmax:在存储空间的 j 指数范围

kmin, kmax:在存储空间的 k 指数范围

north:北边界的边界指数

east:东边界的边界指数

south:南边界的边界指数

west:西边界的边界指数

top:顶部边界的边界指数(只适用于三维坐标)

bottom:底部边界的边界指数(只适用于三维坐标)

E.5.4 由 sampleFluidFace 返回的数据

sampleFluidCell 返回的表包括：

x,y,z:单元格中心的坐标，单位是 m

area:表面面积，单位是 m2。对于轴对称的二维仿真，这将是每弧度的面积

nx, ny, nz:单位法面的余弦值

t1x t1y t1z:第一个切线面的余弦值

t2x t2y t2z:第二个切线面的余弦值

Ybar: 二维轴对称仿真中面积质心的 y 坐标

gvelx, gvely, gvelz:面中点的速度分量。对于非移动网格仿真应该为零。

加上与面相关的 FlowState 描述。参见 E.5.5 节。

E.5.5 由 sampleFluidCell 返回的数据

由 sampleFluidCell 返回的数据包括：

x,y,z:单元格中心的坐标，单位是 m

vol:单元格的体积，单位是 m^3

加上与单元格相关的 FlowState：

p:压强，单位是 Pa

T:反式旋转温度，单位是 K

T_modes:当 nmodes> 0 时，其它内部模式的温度数组，

u:特定的内能，单位是 J/kg

u_modes:其它内能模式数组

quality:多相气流模型中蒸汽分数

massf:指定的质量分数表

a:声速，单位是 m/s

rho:密度，单位是 kg/m3

mu:动态粘度，单位是 Pa.s

k:热导率

k_modes:其他能量模态的传导率数组

tke:单元格内的湍流动能，单位是 J/kg

omega: k-ω 湍流模型，单位是 1/s

mu_t:湍流粘度

k_t:湍流传导率

vel: 带有[x, y, z]数组表的速度分量

B: 带有[x, y, z]分量的磁场强度向量

psi

divB:磁场的散度

注意，并非所有这些项目都与所有仿真相关。例如，T_modes 和 u_modes 并没有用单一温度的气体填充仿真。

E.5.6 模块

用户还可以使用一些附加的便利函数来计算或获得与气体模型相关的值，如热力学性质和传导系数。这些都记录在气体模型 API[5]中。

您的 Lua 脚本也可以利用 Vector3 和 Matrix 对象分别来提供一些三维几何计算能力和一些基本的线性代数操作。有关 Vector3 对象的文档，请参见相应的报告[8]。有关矩阵操作，请参阅源代码模块 bbra .d。

E.6 关于 Lua 解释器和全局变量的说明

对于每个使用 UserDefinedBC 边界条件的边界条件，都会启动一个独立的 Lua 解释器。每个解释器中的全局状态(读取边界条件)都保持在时间步长(即解释器是可重入的)之间。但是，无法在内部将信息从一个 Lua 解释器传递到另一个 Lua 解释器。这里有一个微妙之处。实际上，您可以只编写一个 Lua 文件作为边界条件，但是要在多个边界上设置它，您需要使它足够智能，以便使用 eilmer 提供的信息来确定它是哪个边界，然后根据它采取相应的行动。请记住，尽管您可能使用一个文件，但它是作为每个边界的独立进程来运行的。这些独立的进程不会共享全局状态，也不能进行通信，所以不要在一个脚本中进行更改，而期望它们在另一个脚本中反映。

此外，提供有关块、单元格和界面信息的抽样函数只对计算过程的内存块起作用。MPI 仿真的含义是您无法获得关于常驻在不同计算过程中的块和单元格的信息。如果您尝试这样做，可能会出现分段故障。

有个与用户自定义函数相关的运行耗费。在界面 Lua/D 的任意一边，可能有许多数据需要压缩和解压。如果您发现通过用户定义函数实现您的自定义会使仿真过程非常缓慢，那么可能需要考虑在核心处理器中使用 D 语言。重复计算以及从 Lua 中读取和写入文件也可能导致仿真缓慢。在这种情况下，可以考虑只计算或读取一次，然后将值存储在 Lua 表中，在调用自定义函数时保存这些值。

E.6.1 用户自定义 Lua 脚本的深入练习

我们可以利用 Lua 状态在时间步长之间持续存在这一事实来最小化返回信息所需的计算数量。此外，简单地使用 Lua 解释器可能会导致问题，下面的简短示例将演示。

这里我们有一个简单的用户定义虚拟单元格边界条件。它以恒定的速度和温度填充特定边界处的虚拟单元，压力随时间呈正弦变化。第一个版本(第 131 页-英文版)展示了一个坏习惯的示例，而第二个版本(第 132 页-英文版)展示了如何以更有效的方式实现相同的结果。这些例子中的差别并不是很大。这相当于重新选择对象的位置。

在第一个示例中，所有的工作都在 ghostCells()函数中完成，但有些工作委托给 getFlowTable。在用户自定义边界上的每个面上，每个时间步都会调用多次(取

决于更新阶段的数量)，以填充相邻的虚拟单元格。因此，在第一个例子中，每个更新阶段的每个界面，需要沿边界执行单位转换，气体计算和其他操作，以获得速度和温度。如果这是这种方法唯一存在的问题，那么它不会是一个大问题:您的计算只会花费更长的运行时间。

第二个问题更加微妙，它与 Lua 解释器下面的 D 程序交互的方式有关。这种情况下的问题主要是 GasState 和 GasModel 对象。只要在 Lua 解释器中创建了 GasState 或 GasModel 对象，才会在 D 环境[21]中创建相应的对象。不幸的是，这些对象被故意保留在 D 运行的环境中，因为我们不能轻易地将 Lua 对象的生存期与匹配的 D 对象相互进行通信。因此，这些 D 对象会随着时间积累。这将导致在仿真运算时间内的内存使用量增加。如果您特别不走运，这可能会导致在 RAM 填满时锁定本地机器，或者由于超出内存限制而将作业转储到 HPC 设施上。

为了解决这些问题，所有不依赖于本地模拟状态或鬼影单元信息的计算都可以放在 Lua 文件的开头。这些计算将在 Lua 状态初始化时执行，每次调用 ghostCells 函数时都会重用结果。在虚拟单元 ghostCells 中唯一能够完成的工作便是计算压力，这取决于当前的模拟时间。这是一个不能在初始化时预先计算的值。

User-Defined BC Script- Bad Practice (.lua)

```
1 -- An example of a script that will fill in the
2 -- ghost cells with a user-defined flow condition.
3 -- The flow condition will have constant velocity
4 -- and temperature, and a sinusoidally varying pressure.
5 function ghostCells(args)
6 -- Use a secondary function to define the flowstate
7 flowTable = {}
8 getFlowTable(flowTable)
9
10 -- Initialize the tables that will contain the flow data
11 ghostCell1 = {}; ghostCell2 = {};
12
13 -- Grab the base flow data from the flowTable
14 ghostCell1.velx = flowTable.vel
```

[21] 对于在 D 语言中定义的其他对象，如 Vector3 对象或 FlowState 对象，这也是正确的。

```
ghostCell1.vely = 0
ghostCell1.T = flowTable.T
ghostCell2.velx = flowTable.vel
ghostCell2.vely = 0
ghostCell2.T = flowTable.T

-- Define the sinusoidally varying pressure-
-- a constant pressure plus a disturbance
-- with frequency of 200kHz and amplitude
-- 1 millionth of the freestream pressure
freq = 200e3*2*math.pi

ghostCell1.p = flowTable.p*(1 + 1e-6*math.cos(freq*args.t))
ghostCell2.p = flowTable.p*(1 + 1e-6*math.cos(freq*args.t))

return ghostCell1, ghostCell2
end

-- A function that will return the desired
-- freestream values as a table
function getFlowTable(tab)
-- Define the gas state used to calculate the sound speed
gamma = 1.4
gmodel = GasModel:new{'ideal-air.lua'}
gstate = GasState:new{gmodel}

-- Define the freestream
tab.M = 7.99
tab.p = 0.060*6894.76
tab.T = (1250*0.55556) / (1 + (gamma - 1)*tab.M^2 / 2)
gstate.p = tab.p; gstate.T = tab.T

-- Update the gas state and use the
-- sound speed to get the velocity.
gmodel:updateThermoFromPT(gstate)
tab.vel = tab.M*gstate.a

return tab
end
```

User-Defined BC Script- Good Practice (.lua)

```
1
2 -- An example of a script that will fill in the
3 -- ghost cells with a user-defined flow condition.
4 -- The flow condition will have constant velocity
5 -- and temperature, and a sinusoidally varying pressure.
6
7 -- The freestream only needs to be defined once in the Lua state
8 gamma = 1.4
9 gmodel = GasModel:new{'ideal-air.lua'}
10 gstate = GasState:new{gmodel}
11
12 -- Define the freestream
13 flowTable = {}
14 flowTable.M = 7.99
15 flowTable.p = 0.060*6894.76
16 flowTable.T = (1250*0.55556) / (1 + (gamma - 1)*flowTable.M^2 / 2)
17 gstate.p = flowTable.p; gstate.T = flowTable.T
18
19 -- Update the gas state and use the sound speed to get the velocity.
20 gmodel:updateThermoFromPT(gstate)
21 flowTable.vel = flowTable.M*gstate.a
22
23 -- Initialize the tables that will contain the flow data
24 ghostCell1 = {}; ghostCell2 = {};
25
26 -- Grab the base flow data from the flowTable
27 ghostCell1.velx = flowTable.vel
28 ghostCell1.vely = 0
29 ghostCell1.T = flowTable.T
30 ghostCell2.velx = flowTable.vel
31 ghostCell2.vely = 0
32 ghostCell2.T = flowTable.T
33
34 -- Define the frequency of the perturbation (in radians)
35 freq = 200e3*2*math.pi
36
37 -- Only calculations that rely on the cell-local
38 -- properties (i.e. things in the args table) should
39 -- go inside the function called by the boundary condition
40
41 function ghostCells(args)
42
43 -- Define the sinusoidally varying pressure-
```

```
-- a constant pressure plus a small disturbance of
-- frequency 200kHz and amplitude 1 millionth of the freestream pressure
ghostCell1.p = flowTable.p*(1 + 1e-6*math.cos(freq*args.t))
ghostCell2.p = flowTable.p*(1 + 1e-6*math.cos(freq*args.t))

return ghostCell1, ghostCell2
end
```