**Rowan J. Gollan and Peter A. Jacobs**

# Eilmer4.0 流动仿真程序

## ---------基本气体模型工具包指南

（包含 gas-calc 和 API）

## 2020 年 3 月 20 日更新

译者：卢顺、齐建荟

Technical Report 2017/27

School of Mechanical & Mining Engineering

The University of Queensland

# 目录

# 1. 介绍

Dlang 气体包的编写是为了支持 Eilmer 可压缩流动模拟代码。该软件包提供了与气体混合物的热力学和扩散特性相关的计算服务，并且本身就是一个有用的工具。本指南记录了使用气体包作为独立的工具进行与气体性质相关的计算。可以把它想象成一个研究气体性质的小实验室。这个包的使用示例包括:

- 计算混合气体的粘度；
- 绘制气体在一定温度范围内的焓；
- 通过求解状态方程表达式找到热力学状态;
- 构建热力学循环分析工具。

气体包提供了三种使用方法:

a)作为 D 语言程序中包含的库;

b)通过用 Lua 编写的小脚本(或程序)，并由这个库附带的 gas-calc 程序执行;

c)通过 Ruby 和 Python3 语言解释器的可加载库。

在这个报告中，我们记录了 Lua 脚本中 gas 气体包的使用情况，这些脚本被提供给 gas-calc 进行处理。在使用 Eilmer 进行流动模拟的前处理和后处理阶段时，Lua 脚本也可以使用相同的函数。

本指南大部分讨论都集中在一组用 Lua 编写的示例程序上。在附录中，我们提供了用 Ruby 和 Python3 编写的相同例子。由于 API 中存在一对一的对应关系，所以对这些示例的解释只在 Lua 示例的主要文本中给出。

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/。

## 1.1 黑盒子里面是什么？

作为支持 Eilmer 包的根工具，气体包的特点是一系列气体模型，用于可压缩流动的模拟。这些气体模型包括:

- 理想气体，参数可调;

- 热完全气体的混合物，从数据库中选择物种;
- 一种任意的气体状态方程，其数据存储在查询表中。

与此同时，这个模型列表将随着 Eilmer 用户需求的发展而增长。

对于每一种模型，gas 气体套件提供计算(或查询)以下各项的服务:

(a).热力学； (b).扩散系数； (c).气体种类属性；

## 1.2 一个小样本

为了让您对与工具包的交互有一种感觉，我们首先将展示一个在典型的室温和压力下输出显示空气密度的 Lua 脚本。如列表 1 所示。

```
gm = GasModel:new{"ideal-air-gas-model.lua"}
Q = GasState:new{gm}
Q.p = 1.0e5
Q.T = 300.0
gm:updateThermoFromPT(Q)
print("Density of air= ", Q.rho)
```

列表 1: gas-calc 的一个 Lua 脚本，用于计算空气的密度。

这个脚本的输出直接显示到终端。输出的结果是:

```
Density of air= 1.1610225176629
```

我们现在不会描述这个脚本的细节。对于那些没有耐心的人，再坚持一会儿。在我们描述了 2.1 节的安装过程之后，我们将在 2.2 节中指导您完成第一个 gas-calc 程序。我们保证在这一点上为您详细描述 gas-calc 脚本中的每一行。

## 1.3 如何使用本指南

本文档重点介绍了如何使用这个包，但对各种气体模型背后的理论没有太多的解释。本文档的前几节重点介绍了如何开始构建用于 gas-calc 的小型 Lua 脚本。下一章，Getting Started，将介绍安装过程，并给出编写 gas-calc 脚本的教程介绍。脱离于您的脚本，所有气体模型的可配置细节都是从另一个 Lua 文件读取的，即所谓的气体模型文件。第三章描述了一个非常实用的程序使用例子，来让大家对本指南的内容更有信心。第四章概述了 Lua 应用程序编程接口(API)中提供的可用函数。不要被“应用程序编程接口”这个词所吓倒:这篇概览的基调是对

话式的，旨在给出一个可以做什么的想法，而不是如何做的细节。接下来，第五章是 gas-calc 气体计算的例子合集，并展示了如何做一些特殊的计算。认真学习完这一章后，您就会觉得这是一篇高水平的指南了。第六章将教程对话放在一边，并记录了 Lua API 中所有可用的方法。

# 2. 入门指南

本节是作为教程编写的。目的是指导您安装和编写用于 gas-calc 的第一个程序。

## 2.1 安装

Dlang 的 gas 气体包是在 linux 系统上开发的，下面的介绍也假设您也在 linux 系统上工作。也就是说，我们对开发工具和语言的选择在很大程度上是独立于平台的。如果您选择安装在其它操作系统上，您的操作可能会有所不同[1]。

### 2.1.1 所需软件

以下是构建和安装气体包所需的软件包和工具列表:

- mercurial 系统：从 bitbucket 中获取源存储库的副本；
- 一个 D 语言编译器：该包已通过 Digital Mars D 编译器的测试；
- 一个 C 语言和 Fortran 语言编译器：大多数 linux 系统上的 GNU 系统编译器就足够了；
- make 命令：自动化构建和安装过程；
- .逐行读取开发包；
- .安装开发包。

### 2.1.2 获取源代码

gas 气体包是更大的气体动态模拟工具集合的一部分，该工具包含 **bitbucket**[2] 上的 dgd 项目。即使您只对气体包感兴趣，您也需要获取整个存储库，因为气体包需要包含存储库其余部分中的一些支持包。使用 mercurial 系统，下载存储库的一个副本:

```
$ hg clone https://bitbucket.org/cfcfd/dgd dgd
```

### 2.1.3 第一次设置

我们需要创建一个目录来存放可执行 gas-calc 和支持库的安装。在这个例子

[1] 如果您在 linux 以外的操作系统上成功运行这个包，请让我们也知道。
[2] https://bitbucket.org/cfcfd/dgd

中，我们选择了一个名为 dgdinst 的安装目录，并将其放在$HOME 目录下:

```
$ mkdir $HOME/dgdinst
```

然后，我们需要配置几个环境变量，以便 gas-calc 知道安装目录在哪里。因为我们选择的 shell 是 bash，所以我们在这里显示了一些内容，以便放在您的 $HOME/.bashrc 文件中:

```
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
```

记得刷新当前 shell(或注销并再次登录)，以便新配置的环境可以运行。

### 2.1.4 构建并安装

gas-calc 包可以使用 make 命令来建造和安装。导航到气体源代码区域并执行“make install”命令来构建和安装 gas-calc 程序和支持库模块。

```
$ cd $HOME/dgd/src/gas
$ make install
```

makefile 假定您希望将安装文件放置在默认区域$HOME/dgdinst 中。如果您已经将环境变量设置为指向非标准安装位置，那么您还需要通过在安装时设置 install_DIR 变量来设置新的安装目录。在这种情况下，可以将上面简单的 make install 命令替换为下面所示的命令，该命令对所选择的安装区域没有任何假设：

```
$ make INSTALL DIR=/path/to/my/install dir install
```

## 2.2 教程：等熵膨胀

在本教程中，我们将学习如何使用 gas-calc 来计算理想空气的等熵膨胀过程。我们给出了空气的滞止条件，并要求确定当流动扩展到马赫数为 1(音速条件)的速度时的条件。滞止条件为:

$$p_0 = 500.0kPa;\ T_0 = 300.0K \tag{2.1}$$

先用文字描述我们的分析过程，然后展示达到相同效果的 gas-calc 程序。我们将用焓表示的能量守恒方程作为起点。在膨胀过程中，没有热量损失或获得(绝热流动)。因此，将状态 0 作为停滞条件，将状态 1 作为展开条件，有如下形式：

$$h_0 + \frac{v_0^2}{2} = h_1 + \frac{v_1^2}{2} \tag{2.2}$$

同样，在停滞时，流动速度为零，所以$v_0 = 0$，得到：

$$h_0 = h_1 + \frac{v_1^2}{2} \tag{2.3}$$

我们可以根据给定的滞止条件计算出$h_0$。我们还将计算与滞止条件相关的熵$s_0$，因为我们将使用它作为膨胀过程的约束。我们预计当气体加速到 1 马赫时，压力会更低。利用这个想法，我们可以开始一个迭代过程，逐步降低压力值，并测试在给定压力下气体的速度。对于每一次压力的减小，我们将采用热力学约束，使熵是恒定的。当计算出的流动速度达到马赫 1 时，就停止步长计算。

### 2.2.1 程序说明

现在让我们看一个 gas-calc 程序，它执行刚才描述的分析过程。该程序如列表 2 所示。在引言(第一章)中，我们对 gas-calc 程序做了详细的描述。以下就是代码。

```
1 gmodel = GasModel:new{'ideal-air-gas-model.lua'}
2 Q = GasState:new{gmodel}
3 Q.p = 500e3 -- Pa
4 Q.T = 300.0 -- K
5 gmodel:updateThermoFromPT(Q)
6 -- Compute enthalpy and entropy at stagnation conditions
7 h0 = gmodel:enthalpy(Q)
8 s0 = gmodel:entropy(Q)
9 -- Set up for stepping process
10 dp = 1.0 -- Pa, use 1 Pa as pressure step size
11 Q.p = Q.p - dp
12 M_tgt = 1.0
13 -- Begin stepping until M = M_tgt
14 while true do
15    gmodel:updateThermoFromPS(Q, s0)
16    h1 = gmodel:enthalpy(Q)
17    v1 = math.sqrt(2 * (h0 - h1))
18    gmodel:updateSoundSpeed(Q)
19    M1 = v1/Q.a
20    if M1 >= M_tgt then
21      print("Stopping at M= ", M1)
22      break
23    end
24    Q.p = Q.p - dp
25 end
```

```
26
27 print("Gas state at sonic conditions are:")
28 print("p= ", Q.p)
29 print("T= ", Q.T)
```

列表 2:计算理想空气等熵膨胀的 gas-calc 程序。

程序的前五行是一些常规的设置。这些行，，几乎所有的 gas-calc 程序或它的其它版本中都有。在**第 1 行**，我们选择一个气体模型。在本例中，它是一个理想空气模型，更详细的信息可参照气体模型文件 ideal-air-gas-model.lua。我们将在下一节中讨论如何创建这样一个气体模型文件。现在，只要知道所有的气体模型都是基于一个输入文件进行初始化的。下一行，即**第 2 行**，创建了一个名为 Q 的新变量来保存 GasState 对象。GasState 对象是一个简单的数据结构，它包含许多气体的属性值。使用 GasState:new{}构造函数来分配对象内部的存储空间。GasState 对象并没有包含气体的所有属性，但是它包含了 CFD 仿真最关心的值。在 GasState 中存储的两个值是压强和温度。在**第 3 行**，我们在 GasState 中访问压力值(使用符号 Q.p)，并将其设置为 500.0Pa。类似地，在**第 4 行**，我们将温度设置为 300.0K。最后，在**第 5 行**，我们通过调用 GasModel 的 updateThermoFromPT() 方法来计算剩余的热力学状态参数。当我们说剩下的热力学状态时，在这种情况下密度和热力学能是根据压强和温度的值来计算的。您可能会想，“什么？我没有在第 5 行看到任何有关密度和热力学能的计算。这些值到哪里去了?”。密度和热力学能的值实际上在 Q 数据结构中得到了更新。换句话说，Q.rho 和 Q.u 的值在第 5 行前后是不同的。在第 5 行之后，数值是正确的状态，对应于我们之前在 Q 数据结构中设置的压力和温度。这是许多 GasModel 方法的共同主题:它们假定调用者为 GasState 中的某些属性设置了正确的值，并由此设置了其余的热力学状态。这个想法将在**第四章**详细讨论。

在 Lua 编程语言中，注释以两个破折号(--)开始，并扩展到行尾。**第 6 行**是一个注释行的例子。第 4 和第 5 行的注释提醒我们压强和温度的预设单位。

**第 7 行**和**第 8 行**用来计算滞止状态的焓和熵。我们知道，焓和熵是在滞止状态下计算的，因为这些是 Q 变量中的状态值，通过这些状态值传递给焓 enthalpy() 和熵 entropy()。在第 5 行，我们直接在 Q 变量中更新了密度和内能。现在在第 7 和 8 行，我们返回焓和熵的值，并把它们赋给变量。为什么调用方法和返回值有区别?我们前面说过，并不是所有的气体属性都存储在 GasState 对象中。焓和熵

是气体属性的两个例子，它们不是作为气体状态的一部分存储的。因此，我们不能在变量 Q 中更新它们，相反，我们将它们返回给读者，让其保留并按照他们的意愿[3]进行处理。

**第 9-12 行**没有向我们展示 Lua 气体模块的任何新内容。这几行 Lua 代码用于设置压力递减(dp)的大小，为迭代过程设置初始压力值，并设置目标马赫数值。我们将目标马赫数设置为一个变量，这样如果我们对扩展到不同的马赫数值感兴趣，就可以轻松地重新使用这个程序。

**第 14-25 行**是执行迭代过程的代码。while 循环从第 14 行开始，如果满足循环条件，即为真，则计算始终循环下去。为了跳出这个潜在的无限循环，我们依赖于第 22 行中的 break 语句，它将在某个时刻被触发。在每个循环入口的开始，我们有一个新的压力值要测试。在**第 15 行**，我们更新了 Q 中的热力学状态，基于压力(在 Q 内部为 Q.p)和熵(s0)。熵作为一个向 updateThermoFromPS()提供的指向参数。我们总是对这个函数感兴趣，因为我们对等熵膨胀很感兴趣:当压强改变时，熵保持不变。在调用这个函数之后，温度、密度和热力学能在 Q 中更新，我们最感兴趣的是温度更新，因为它将在**第 16 行**中被用来计算新的焓。现在我们得到的是:

1.校正后的压力；

2.计算了新的温度、密度和内能；

3.利用得到的新温度计算新的焓。

在**第 17 行**，我们重新整理 2.3 式，求出气体的速度 v1。注意，sqrt()函数是 Lua 数学模块的一部分，因此可以作为 math.sqrt 访问。**第 18 行**用于更新声速值。同样，这个函数似乎没有返回任何值。声速方法是更新方法家族中的一种。这给了我们一个线索，Q 的值被改变了。在这个例子中，a 的值，也就是声速，被更新为 Q.a。实际上，我们在下一行，即**第 19 行**，用 Q.a 来计算马赫数。我们故意避免声明哪些值存储在 Q 中。但我们并不想让这个问题变得神秘:只是更愿意关注程序更广泛的方面。存储在 GasState 中的值的完整列表在**第 4.2 节**中给出。

在**第 20 行**，我们决定是继续递减压力还是继续循环过程。因为我们从较小

[3] 现在，如果您正在思考为什么有些值存储在 GasState 中，而有些值不存储在 GasState 中，那么答案与支持 Eilmer 流动求解器的气体包有关。在大型 CFD 模拟中，我们需要注意在模拟过程中使用了多少内存。如果我们包含了所有的其它状态参数，这将带来巨大的内存开销，但其中包含了我们不经常使用的值，或者根本不使用的值。因此，我们只存储那些在 CFD 程序中经常使用的值。

的值逐渐接近了马赫数。需要测试是否已经达到或超过了 M=1 的值。如果该值超过了目标值，我们在第 21 行输出显示当前的马赫数(希望只会有少量的超出)。**第 22 行**上的 break 语句终止 while 循环，代码跳转到**第 27 行**执行。**第 23 行**上的 end 语句结束了在 20 行开始的复合 **if** 语句。如果我们还没有达到目标马赫数，那就执行**第 24 行**，以减少 1Pa 的压力值，这为我们的下一个循环迭代做准备。

**第 27-29 行**是输出显示语句，用于将计算结果写入终端输出。特别是当流速为 1.005 马赫时的压力和温度条件被输出显示了出来，然后程序退出(不需要明确的声明来结束程序)。

### 2.2.2 运行程序

在我们进行 isentropic-air-expansion.lua 等熵空气膨胀之前，我们需要准备一个气体模型文件 ideal-air-gas-model.lua。该文件是 Lua 格式的纯文本数据文件。虽然手动构造一个 gas 模型文件是可能的，但这样做非常繁琐，而且容易出错。正因如此，prep-gas 程序与气体包一起提供，以帮助准备气体模型文件。这个程序需要一个非常小的输入文件来选择与所需气体模型相关的一些选项。用于生成 ideal-air-gas-model.lua 的 prep-gas 的输入文件如列表 3 所示。

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

列表 3：prep-gas 程序的输入文件，以建立理想空气模型。

prep-gas 程序使用两个参数从命令行运行。第一个参数是输入文件的名称，第二个参数是输出文件的名称。输出文件是 gas-calc 使用的完整气体模型文件:我们说过，手动处理这个文件太繁琐了。这些气体模型文件不仅仅被 gas-calc 使用。它们与 Eilmer 配置气体模型时使用的气体模型文件相同。要为本例生成气体模型文件，请执行以下操作：

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

我们将在第三章详细讨论气体模型的制备。

下一步是运行 gas-calc 本身。在命令行中，它只接受一个参数:程序文件的名称。您需要将 gas 模型文件与 gas-calc 程序放在相同的工作目录中。要运行程序，请输入:

```
$ gas-calc isentropic-air-expansion.lua
```

程序运行需要几秒钟。屏幕上的输出应该如下所示:

```
Stopping at M= 1.0000029005924
```

```
Gas state at sonic conditions are:
p= 264140
T= 249.99975828385
```

执行程序的这两个步骤存储在一个名为 run.sh 的运行脚本文件中。您可以在 dgd/examples/gas-calc/isentropic-expansion/.中找到这个完整的示例以及运行脚本。

### 2.2.3 项目改进

本教程示例旨在向您介绍 gas-calc 程序的元素。因此，我们努力得保持示例的简单性。可以对这个程序进行一些改进，我们将在这里讨论这些改进。我们将提出准确性和效率方面的改进。

您可能会注意到，我们在马赫数略大于 1.0 时停止了，并把这个值作为我们音速条件的估计值。无需做太多额外的工作，就可以在最后两个步骤的基础上插入声速条件，这样就提高了精确度。

可以用一个更简洁的迭代过程来提高效率并达到预期的精度。例如，割线迭代是一种比较有效的方法，该方法基于用户提供的残差而停止工作。

最后，提高准确性和效率的终极方法就是使用分析性的表达方式。学习气体动力学的好学生都会记住(或能够推导)一维等熵流动的关系。我们可以直接用这些关系来计算音速条件：

$$\frac{T^*}{T_0} = \frac{2}{\gamma + 1} \tag{2.4}$$

$$\frac{p^*}{p_0} = \left(\frac{2}{\gamma + 1}\right)^{\frac{\gamma}{\gamma - 1}} \tag{2.5}$$

所以，就这样做！使用给定的滞止条件(式 2.1)以及$\gamma = 1.4$得到：

$$p^* = 26.414kPa\,;\ T^* = 250.0\mathrm{K}$$

这是对我们程序结果很好的肯定。

当然，我们建立这个程序是作为一个演示，而不是作为一个计算等熵膨胀气体声速条件的最好方法的例子。这个程序的好处是，计算方法可以更普遍地应用于具有复杂行为的气体。但该解析式只适用于热完全气体，即比热恒定的气体。

### 2.2.4 回顾一下我们演示过的内容

在结束这一节之前，让我们简单地回顾一下用于计算空气等熵膨胀的 gas-calc 程序所演示的内容。我们介绍了初始化方法，这是每个 gas-calc 程序的一部分。这些是用来初始化一个 GasModel 和一个 GasState 的程序。我们还提到，您

需要一个单独的气体模型文件来初始化 GasModel。prep-gas 程序用于帮助创建气体模型文件。下一章更详细地介绍了现有的气体模型和如何使用 prep-gas。

在描述示例程序时，我们提到了这样一种思想，即气体的状态保存在 GasState 数据结构中，而数据是由 GasModel 中的方法操作的。我们还介绍了两类不同的方法:a)更新 GasState 中的值;b)返回一个值给调用者。例如，updateThermoFromPT() 是更新族中的一个方法。它更新了气体状态数据结构中的密度和内能值，这是基于压力和温度是正确的假设。另一方面，焓方法也是返回方法族的一部分。它还依赖于 GasState 对象内部的热力学状态在调用时是否正确，但它将其计算结果作为值返回给调用者。我们了解到，这种行为差异的原因是一些热力学值存储在 GasState 数据结构中，因此，可以在适当的地方进行更新。还有其它值没有存储在 GasState 中。这些值需要在请求计算时返回给外部环境。

# 3. 气体模型制备

这是一个简短的章节来描述一个小型但功能强大的工具：prep-gas，它的目标是减轻准备 gas 气体模型文件时的痛苦。气体模型文件包含大量关于气体属性的信息。当处理气体混合物(或多组分气体)时，气体模型文件需要包含每种气体的详细信息。在模拟碳氢化合物的反应流动时，通常在气体混合物中有大约 50 种物质存在。您可以看到，为什么物种收集和准备数据过程是繁琐的了。相反，我们提供了 prep-gas 工具，从物种及其属性的数据库中构建一个气体模型文件。

## 3.1 可用的气体模型

气体包支持的气体模型列表如表 3.1 所示，该表包含对模型行为的简要描述。D 语言的源代码包给出了热力学关系的细节和输运性质的计算。如果您需要实现一个新的气体模型类，您将需要学习这个源代码。

表 3.1：气体包中的可用气体模型

| 模型 | 描述 |
|---|---|
| IdealGas | 单组分气体:这是完全的碰撞行为和恒定的比热 |
| ThermallyPerfectGas | 多组分气体:每个组分都被视为热完全，这意味着发生了完全的碰撞行为。此外，各组分的内能仅随温度变化。 |
| CO2Gas | 稠密气体的 Bender 气体模型[1]，模型参数设置为二氧化碳。 |
| look-up table | 一种一般的气体模型，由性质以表格形式编码的单个虚拟物种组成。 |

## 3.2 可用的种类

通过运行 prep-gas-命令的-list-available-species 选项，可以产生一系列可用的气体种类。

```
$ prep-gas --list-available-species
```

物种名称是区分大小写的，并使用化学化合物的传统命名。这个命名规则的例外是虚拟物种（air）。这个小写符号名用于选择在 NASA 化学中使用的空气模型平衡分析(CEA)程序[2]。

## 3.3 如何使用 prep-gas

在正常操作中，prep-gas 在命令行上有两个参数:1)一个输入文件;2)输出文件。输入文件是一个小型文本文件(Lua 格式)，指定气体模型和混合物中的物种。输出文件也作为 Lua 文件创建，并给出第二个参数的名称。该输出文件是气体模型文件，包含指定气体模型所需的所有配置值。它在调用 GasModel 构造函数时使用。用于选择理想空气的输入文件例子如列表 4 所示。输入文件声明了一个模型类型和一个物种列表。该模型可以是表 3.1 中所列的任何一个支持的模型。物种列表可以包括使用--list-available-species 选项运行 prep-gas 时列出的任何物种。

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

列表 4：ideal-air.inp 文件：给 prep-gas 的输入文件

使用 prep-gas 运行的例子如下:

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

成功完成后， ideal-air-gas-model.lua 文件将被创建并位于工作目录中。

## 3.4 prep-gas 工作原理

大多数人对 prep-gas 的工作原理不感兴趣，所以我们将尽量保持简短。在数据的安装区域中，有一个名为 species-database.lua 的大型文本文件。这个数据库包含各种气体模型以及按物种分类所需的所有参数。正如我们提到的，这就是所有可以调用的数据。对于任何给定的气体模型，我们只需要这些数据的一个子集。这就是 prep-gas 的作用。它的工作是加载所有物种数据，但只选择那些感兴趣的物种，只选择所选气体模型所需的参数。然后，prep-gas 将子集数据写入气体模

型文件。

使用这种方法可以实现两件事:1)它减少了气体模型文件的大小;2)它创建了受众可读气体模型参数的本地记录。从“可重复研究”的角度来看，第二点非常有用。有了本地创建的气体模型文件，您就有了执行计算时使用参数的完整记录。当您试图将结果传递给其他人以便他们可以复制您的工作时，这是非常方便的。

# 4. Lua API 之旅

在本节中，我们将指出在 gas-calc 程序中提供的气体建模功能。gas-calc 程序作为标准的 Lua 解释器运行，并添加了一些额外的优点。这些好东西是与气体模型相关的特殊函数和对象。这次讨论将集中在这些好东西上。要了解更多关于 Lua 编程的知识，我们推荐优秀的书籍《Lua 编程》[3]，作者是 Ierusalimschy。由气体包提供的具体额外功能的讨论，这也是对可用功能的介绍。关于函数参数、返回类型、调用者的先决条件以及所有其它有趣的东西的内容将在第 6 章中讨论。

## 4.1 GasModel 与 GasState 的关系

每个 gas-calc 程序的中心有两个对象:GasModel 和 GasState。实际上，本教程示例中程序的前两行专门用于初始化 GasModel 和 GasState。那么这些对象之间的关系是什么呢?

简单地说，GasModel 对象使用/操作 GasState 对象中的数据。顾名思义，GasModel 对象负责对气体的行为建模。它提供用户可以调用的服务。GasState 对象是用于存储气体当前状态的数据结构。通常，我们的程序中只有一个或少量的 GasModel 对象。(例如，如果我们想直接比较不同气体在一个温度范围内的行为，我们可能有不止一种气体。)另一方面，如果我们想要在模拟的不同区域中保存气体状态的记录，我们最终可能会得到许多 GasState 对象。

我们总是通过传递气体模型文件的名称来初始化 GasModel:

```
gmodel = GasModel:new{'my-gas-model-file.lua'}
```

有几种方法可以初始化 GasState 对象，但最简单的方法是调用以 GasModel 对象为参数的构造函数:

```
Q = GasState:new{gmodel}
```

原因是，我们的研究表明在气体模型中，气体状态结构可以形成物种的数量和非平衡能量模式的数量。这些数字用于在 GasState 对象中预设置特定数组的大小。在 Lua 语言域中，GasState 对象只不过是一个 Lua 表。可以直接构造 GasState

表而不调用 new{}构造函数，但不推荐这样做。重要的是要知道，气体包中提供的方法希望在 GasState 表中出现某些字段。如果您删除某些字段(意外或故意)，则气体包方法的行为是未知[4]的。

## 4.2 使用 GasState 数据结构

我们提到过 GasState 是一个 Lua 表。表 4.1 描述了 GasState 表中的字段。

表 4.1:存储在 GasState 表中的数据。

| 数据 | 类型 | 描述 |
|---|---|---|
| rho | 数 | 密度，$kg/m^3$ |
| p | 数 | 压力，Pa |
| p_e | 数 | 电子压力，Pa(当气体模型采用分离电子温度时，只能是非零值，否则电子分压与其它组分都包含在常规压力变量 p 中) |
| a | 数 | 声速，m/s |
| u | 数 | 比内能(以旋转的方式)，J/kg |
| e_modes | 数组 | 非平衡态的比内能(通常在振动模式下)，J/kg |
| T | 数 | (旋转)温度，K |
| T_modes | 数组 | 非平衡态温度(对应于 e_modes)，K |
| mu | 数 | 动力粘度，Pa.s |
| k | 数 | 旋转模型下的热导率，W/(m.K) |
| k_modes | 数组 | 非平衡模型下的热导率，W/(m.K) |
| sigma | 数 | 导电率，S/m |
| massf | 一对 key-value 的表<br>key：字符串，种类名称<br>value：质量分数的值 | 混合种类的质量分数，在表中未列出 |
| quality | 数 | 蒸汽质量 |

[4] 事实上，gas-calc 程序很可能会因为在期望合法值的地方发现一个“nil”值而失败。

查看 GasState 的字段，您可能会发现一些明显的缺失。例如，焓在哪里，比热在哪里，$C_p$ 和 $C_v$ 呢?这是因为在 GasState 我们选择只存储可以许多描述变量中的一部分。我们的选择是基于 CFD 计算中最常用的计算方法。

温度场、能量场和导热系数场分别为标量和数组。标量场是单温度气体模型所需要的，温度、内能和热导率在平衡热力学中具有通常的意义。在内部模型中，对于能量非平衡分布的气体，有各种不同的多温度气体模型可用。在这些模型中，非平衡模型的能量、温度和导热系数被存储在数组中。它们以数组的形式呈现，因为气体包就是设计来处理具有任意内部能量模型划分的气体模型。请注意，平移的能量、温度和热导率总是存储在标量项中。数组项用于附加的非平衡模型。这到底是什么意思?以模拟地球高速再入流的双温度模型为例。在该模型中，气体混合物的平动能和转动能由一个温度控制。这是第一种模式，存储在表的标量项中。内能的第二种模式将内能的振动能和电子能集中在一起。它们由第二温度描述，与第二温度相关的值作为数组中的第一项存储。因此，我们最终得出 T_modes 数组中只有一个温度值，e_modes 数组中只有一个能量值，k_modes 数组中只有一个导热系数值。作为一个例子，下面的代码将用于显示前面描述的双温度模型中的转动和振动电子温度。

```
print("transrotational temperature= ", Q.T)
print("vibroelectronic temperature= ", Q.T_modes[1])
```

质量分数表是通过物种名称来访问的，而不是像温度数组那样通过索引来访问。这是因为我们的模式通常很少，且它们很容易跟踪编号的索引。然而，在混合物中可能有很多物种，它变得更容易处理我们感兴趣的少数名字。质量分数值是使用 specesname=massFraction 形式的关键值进行设置的。例如，我们可以设置氧和氮的质量分数来近似空气的组成，就像这样:

```
Q.massf = {O2=0.22, N2=0.78}
```

设置这些值的另一种方法是:

```
Q.massf['O2'] = 0.22; Q.massf['N2'] = 0.78
```

或者，使用 Lua 的语法，将字符串关键字转换为成员访问，我们甚至可以像下面这样做，以达到相同的效果:

```
Q.massf.O2 = 0.22; Q.massf.N2 = 0.78
```

在所有这些例子中，物种名称的情况都很重要。如果试图将氧质量分数设置为可疑的名称，如 Q.massf[' o2 '] = 0.22，则将导致错误。在赋值的时候它不会出错但是当您把它交给气体模型时它存在的问题会被触发。您还可以使用常规 Lua 表访问机制，通过名称检索质量分数值:

```
mfO2 = Q.massf[’O2’]
mfN2 = Q.massf.N2
```

由于我们提供了便捷的访问质量分数的名称，所以需要告诉您一些游戏规则。如果您只处理一种气体，那就可以不动质量分数表;对于一种气体，质量分数表默认只有一个条目，其值为 1.0。如果您对研究多种气体感兴趣，请继续读下去。质量分数表是一个简单的 Lua 表，因此您可以在这里随意添加任何内容....但是不要这样做!只有当您在 GasModel 方法中实际使用该表时，才会检查它。当质量分数表被检查时，它将首先查找任何未知或无效的关键字。这将发生一个错误，并导致程序终止。接下来，如果所有的关键字都是有效的物种名称，那么就需要决定给表中[5]没有列出的所有物种的质量分数分配什么值。我们已经决定所有未列出的物种将被赋值为 0.0。这通常是大多数用户想要的。然后我们需要检查所提供的值的总和是否为 1.0。如果总和略大(比如小于百万分之一，或者内部残差允许)，那么质量分数将被缩放，使总和为 1.0。如果输出的总和超过可接受的数量，则输出错误消息并终止程序。

## 4.3 更新方法

引入 GasState 对象后，我们现在可以讨论如何使用它了。GasModel 对象提供了几种更新方法，它们将使用 GasState 中的数据来更新其它变量的状态。这些更新方法都是热力学状态方程。让我们看看提供的第一个更新方法 GasModel:updateThermoFromPT()。如果这是一部电影，那么我们已经在标题[6]中

[5] 这里有一个微妙之处。如果您使用 GasState:new{}构造函数来初始化 GasState(这是我们鼓励的)，那么它将把所有物种的质量分数设置为 0.0。换句话说，这些物种是在背景中设定的，即使您没有明确设定该值。这不是一个大问题，因为 0.0 的默认值通常是您想要的。然后，用户的工作是只设置那些对质量分数有趣的物种值(非零)。对于只有一种气体的情况，在质量分数表中默认值为 1.0。

[6] 不幸的是，我们的 GasModel 方法并没有获得奥斯卡奖的提名，所以我们可以不用创造性的标题，而使用一些更平淡但意图明确的东西。

透露了情节。正如这个方法的名字所暗示的那样，我们将在给定压力和温度的情况下更新 GasState 的热力学量。让我们看看这个方法是如何运行的(假设之前已经初始化了 gmodel 对象)：

```
Q = GasState:new{gmodel}
Q.p = 1.0e5; Q.T = 300.0
print("rho= ", Q.rho, " u= ", Q.u)
gmodel:updateThermoFromPT(Q)
print("rho= ", Q.rho, " u= ", Q.u)
```

运行这几行代码的输出结果是：

```
rho= nan u= nan
rho= 1.1610225176629 u= 215327.43439227
```

在第一个输出语句中，密度和热力学能的值是 nan。我们希望这样做，因为这些值是由底层的 dlang 模块默认初始化的。在 D 编程语言中，浮点值默认初始化为 nan。在调用 updateThermoFromPT()之后，密度和内能的值被更新。更新方法传递了一个 GasState 数据结构。由此引出压力和温度的值，然后进行计算，更新密度和热力学能的值。

下面的三种更新方法也遵循相同的模式:

- updateThermoFromRHOU()
- updateThermoFromRHOT()
- updateThermoFromRHOP()

同样，名称指示 GasState 对象中假定哪些变量是有效的。然后计算其它原始热力学变量。我们考虑密度(rho)，内能(u)，压力(p)和温度(T)作为原始变量。这组值在表示热力学状态方面没有什么特别的。在这个操作过程中，我们认为它们是原始的，因为它们直接存储在 GasState 对象中。

updateSoundSpeed()方法是特殊的。它假设 GasState 中的其它原始变量是最新的，然后更新 GasState 中的声速值(a)。它的特殊之处在于它只更新了声速，而其它所有更新方法都改变了原始的热力学变量。因此，通常情况下，在调用 updateSoundSpeed()之前，会先调用前面提到的一个更新方法。

还有另外两个更新方法略有不同。在给定压力和熵时，其中一个 updateThermoFromPS()方法用于更新输入的热力学状态。该方法接受两个参数:一个 GasState 对象和一个熵值。这个方法打破先前模式的原因非常简单:在 GasState 中，熵作为数据成员不可用，因此我们必须直接传递它。但是，压力在 GasState

中是可用的，因此为了与其它方法保持一致，我们将从 GasState 对象中提取压力值。在第 2.2 节的列表 2 的第 15 行中，我们已经看到了这个方法的使用示例。更新族中的最后一个方法是 updateThermoFromHS()。该方法接受三个参数:一个 GasState 对象、一个焓值和一个熵值。这种方法使用焓和熵来开始计算热力学状态。那么为什么我们需要传递 GasState 对象呢?这仍然是一种更新方法。尽管我们不直接在 GasState 对象中使用任何值，但作为更新的一部分，我们实际上更改了值的状态。调用此方法后，在 GasState 对象中更新密度、内能、压力和温度的值。

这就是完整的更新方法汇总。它们都是根据状态方程原理:给定两个热力学变量，就可以计算出剩余的状态。这还不是多组分(多种)气体的全部情况。对于多组分气体，为了计算状态，也需要气体组成。这也是为什么 GasState 对象总是作为注释传入。对于多组分气体，所有的更新方法也将使用质量值，以执行所要求的热力学状态方程计算。因此，在调用多组分气体的更新方法之前，也应该正确地设置这些值。

## 4.4 返回方法

GasModel 提供了许多方法，形成了一系列返回方法。所有这些方法都接受一个参数，即 GasState 对象。当调用任何返回类型方法时，假定原变量是最新的。通常，我们会在调用返回方法之前调用一个更新方法。这些方法在调用时都返回一个浮点值。同样，这些方法的名称不会让您感到意外。例如，dhdTConstP()方法将计算热力学导数：$\left(\frac{\partial \mathrm{h}}{\partial T}\right)_p$。其它的返回方法包括 gasConstant()方法：平均质量的气体常数；以及 molMass()方法：混合物分子质量。

一些返回方法有别名。例如，Cv()方法会给出与调用 dedTConstV()相同的结果，Cp()是 dhdTConstP 的别名。调用 R()方法时，它的工作将被委托给 gasConstant()。

作为使用返回方法的示例，让我们看看如何通过调用 gamma()方法来计算 4000K 时氮和氧混合物的有效比热 γ：

```
gmodel = GasModel:new{'therm-perf-N2-O2.lua'}
```

```
Q = GasState:new{gmodel}
Q.p = 1.0e5; Q.T = 4000.0
Q.massf = {N2=0.78, O2=0.22}
gmodel:updateThermoFromPT(Q)
gamma = gmodel:gamma(Q)
print("gamma= ", gamma)
```

## 4.5 复合表示形式之间的转换

为了完成 Lua API 之旅，最后要介绍的方法是转换版本方法。这些方法用于在组合各种表示方法之间进行转换。让我们看看如何使用 massf2conc()方法，以我们以前计算比热比时使用的质量分数为基础，得到 $N_2$ 和 $O_2$ 的浓度：

```
conc = gmodel:massf2conc(Q)
print("c[N2]= ", conc.N2, " c[O2]= ", conc["O2"])
```

有四个转换函数可用:massf2molef()， molef2massf()， massf2conc()，conc2massf()。所有这些转换方法都会返回一个表。(在上面的例子中，我们看到返回了一个浓度表。)这些表的工作原理类似于质量分数表:不同物种的值由物种名称决定。当使用转换为质量分数的转换方法时，您实际上得到了一种更新方法和一种返回方法。在调用期间，传入的 GasState 对象中的 massf 字段会得到更新。同样的 mass 字段也会返回给调用者。通过返回 massf 表，使方法调用的行为与其它转换方法保持一致。如果您不想处理返回的表，就不要捕捉它(也就是说，使用赋值操作符将名称绑定到它)。在 Lua 编程语言中，忽略返回值是完全有效的。为了演示这一点，在计算氮、氧和氢的化学计量混合物中氢的质量分数时，我们将忽略返回的质量表：

```
gmodel = GasModel:new{'therm-perf-N2-O2-H2.lua'}
Q = GasState:new{gmodel}
mole_total = 1.0 + 2.0 + 3.76
molef = {H2=1.0/mole_total,
O2=2.0/mole_total,
N2=3.76/mole_total}
gmodel:molef2massf(molef, Q)
print("f[H2]= ", Q.massf.H2)
```

# 5. 例子

本节给出一些简短的示例，演示如何使用 gas-calc 程序。构建 gas-calc 背后的想法是：CFD 代码 Eilmer 的用户，可以在一个方便的脚本界面中访问仿真代码中使用的相同的气体模型函数。这意味着我们可以在模拟过程中对气体性质进行计算，这与在 Eilmer 内部进行的计算是一致的。这对于在模拟前或模拟后进行一些辅助计算是很有用的。

前两个例子说明了如何计算某些特定的气体性质。这些是在一个温度范围内计算的，这样就可以绘制出随温度变化的特性图。最后一个例子展示了如何构造一个小的分析程序。这个特殊的程序计算热力学循环中不同点的状态，并报告循环效率。

## 5.1 分子氧的 Cp 和焓变曲线

在这个例子中，我们将计算出在 200-20000K 温度范围内，氧气分子在恒压下的比热(Cp)和焓(h)。执行此计算的 gas-calc 程序如列表 5 所示。

在这个例子中，我们为 $O_2$ 选择了热完全气体模型，因为我们希望在如此大的温度范围内，Cp 会有一些变化。要进行此选择，prep-gas 的输入文件如下所示:

```
model = 'thermally perfect gas'
species = {'O2'}
```

在第 11 和 12 行，气体模型被初始化。每个气体计算程序的典型起点都是建立气体模型，因为我们不能在不定义气体行为的情况下进行任何计算。第 14-16 行用于设置 GasState 表，并使用在整个程序中保持不变的一些值填充该表。请注意，特定的压力值并不重要，因为在热完全气体中，$C_p$ 和 h 的性质表明它们不是温度的函数。但是，设置一个压力值是一个很好的实践过程，因为一些热力学规律将依赖于有意义的压力值(即使在这些特定的 $C_p$ 和 h 的计算中不使用它)。

```
1 -- A script to output Cp and h for O2 over temperature range 200--20000 K.
2 --
3 -- Author: Rowan J. Gollan
4 -- Date: 2016-12-23
```

```
5 --
6 -- To run this script:
7 -- $ prep-gas O2.inp O2-gas-model.lua
8 -- $ gas-calc thermo-curves-for-O2.lua
9 --
10
11 gasModelFile = 'O2-gas-model.lua'
12 gmodel = GasModel:new{gasModelFile}
13
14 Q = GasState:new{gmodel}
15 Q.p = 1.0e5 -- Pa
16 Q.massf = {O2=1.0}
17
18 outputFile = 'O2-thermo.dat'
19 print("Opening file for writing: ", outputFile)
20 f = assert(io.open(outputFile, "w"))
21 f:write("# 1:T[K] 2:Cp[J/kg/K] 3:h[J/kg]\n")
22
23 Tlow = 200.0
24 Thigh = 20000.0
25 dT = 100.0
26
27 for T=Tlow,Thigh,dT do
28 Q.T = T
29 gmodel:updateThermoFromPT(Q)
30 Cp = gmodel:Cp(Q)
31 h = gmodel:enthalpy(Q)
32 f:write(string.format(" %12.6e %12.6e %12.6e\n", T, Cp, h))
33 end
34
35 f:close()
36 print("File closed. Done.")
```

列表 5:计算 $O_2$ 的 $C_p$ 和 h 值的 gas-calc 程序。

第 18-21 行用于准备输出文件，以便可以用文本形式来记录计算。我们将把每个温度值的数据按行写出来。表的开头被写在文件的顶部(在第 21 行)，以便我们在以后检查文件时方便查看。

在第 23-25 行中，我们设置了温度范围和温度的增量。我们选择以 100K 为步数递增。

主要循环发生在第 27 和 33 行。在循环的每一步，我们都使用一个新的温度值。我们把这个新的温度值放在 GasState 表的 Q 中，以便它是可用的和最新的

热力学计算方法。然后，我们通过调用基于 p 和 T 的更新方法来更新热力学状态。严格地说，这个调用对于这个特定的气体模型是不必要的，但是，对于一般情况，在 $C_p$ 和 h 评估之前有一个完全最新的动力学状态是很好的仿真练习例子。最后一个是对特定服务的调用，以返回 $C_p$ 和焓值。这些值被写入到第 32 行中的文件中。

程序结束的方式是关闭文件并在屏幕上显示一条消息，让我们知道一切正常。

$C_p$ 和 h 的结果数据如图 5.1 所示。由于内部振动的能量模式和电子结构的激发，分子氧的 $C_p$ 将随温度发生变化。T=200K 时的焓值略低于零。这是因为在 T=298.15K 时，分子氧的参考焓值设定为 0.0。这个参考焓很常见;它常在 JANNAF 表和 CEA 程序中使用。

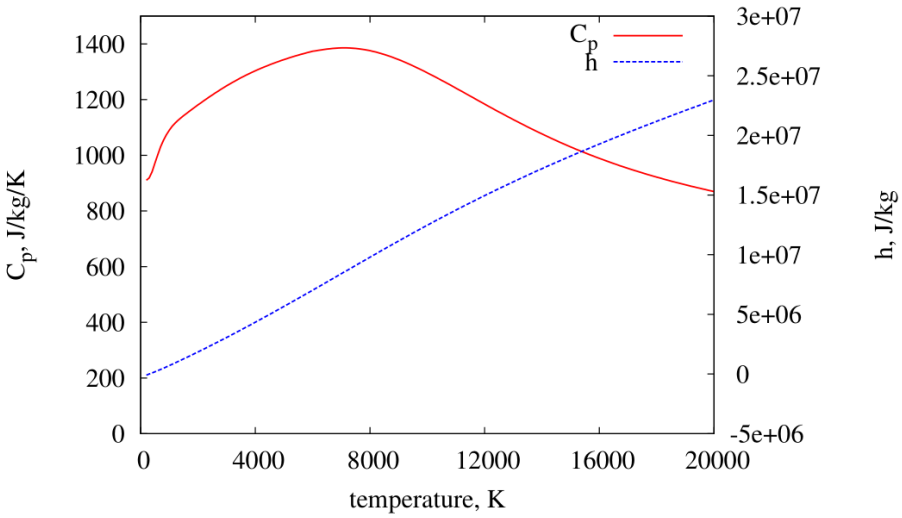


图 5.1:压力不变时，时，氧分子 $O_2$ 的 Cp 和焓 h 随温度的变化。

## 5.2 N2-O2 混合物的粘性和热导率

在这个例子中，我们将建立一个气体计算程序来计算气体混合物的粘度和热导率。这种气体混合物由 78%的氮气和 22%的氧气组成。这种混合物在低温下很接近空气的组成。在更高的温度下，我们预计会发生一些离解，如果温度足够高，还会发生电离。在更高的温度下，气体的组成就不像这种双组分混合物那么简单了。无论如何，作为演示，我们将计算这种混合物在 200-20000K 的温度范围内的运输系数。

这个例子的 gas-calc 程序如列表 6 所示。它与前面用于计算 $C_p$ 和 h 的示例非常相似。为了避免重复，只讨论此脚本与前面示例的区别。

在这个例子中，气体混合物有两种成分。这引入了两个不同之处。首先，gas 模型的输入文件如下所示:

model = 'thermally perfect gas'

species = {'N2', 'O2'}

其次，我们需要注意设置每个成分的质量分数。这在第 17 行完成。

```
1 -- A script to compute the viscosity and thermal conductivity
2 -- of air (as a mixture of N2 and O2) from 200 -- 20000 K.
3 --
4 -- Author: Rowan J. Gollan
5 -- Date: 2016-12-23
6 --
7 -- To run this calculation:
8 -- $ prep-gas thermally-perfect-N2-O2.inp thermally-perfect-N2-O2.lua
9 -- $ gas-calc transport-properties-for-air.lua
10 --
11
12 gasModelFile = 'thermally-perfect-N2-O2.lua'
13 gmodel = GasModel:new{gasModelFile}
14
15 Q = GasState:new{gmodel}
16 Q.p = 1.0e5 -- Pa
17 Q.massf = {N2=0.78, O2=0.22} -- a good approximation for the composition of air
18
19 outputFile = 'trans-props-air.dat'
20 print("Opening file for writing: ", outputFile)
21 f = assert(io.open(outputFile, "w"))
22 f:write("# 1:T[K] 2:mu[Pa.s] 3:k[W/(m.K)]\n")
23
24 Tlow = 200.0
25 Thigh = 20000.0
26 dT = 100.0
27
28 for T=Tlow,Thigh,dT do
29 Q.T = T
30 gmodel:updateThermoFromPT(Q)
31 gmodel:updateTransCoeffs(Q)
32 f:write(string.format(" %12.6e %12.6e %12.6e\n", T, Q.mu, Q.k))
33 end
34
```

```
35 f:close()
36 print("File closed. Done.")
```

列表 6：计算 $N_2$-$O_2$ 混合物的$\mu$和 $k$ 值的 gas-calc 程序

唯一的区别发生在循环构造中。再一次，我们通过调用 p 和 T 更新方法来严格要求在每次温度增加后更新热力学状态。如前所述，这是一个最佳的练习方法，因为它将确保最一般的气体模型的正确性。为了更新传输系数，我们调用相应的 updateTransCoeffs()方法。由于$\mu$和 $k$ 是在 Q GasState 表中就地更新的，所以我们不需要将它们存储在临时变量中。相反，我们可以在将这些值写入第 32 行文件时直接访问这些最新的值。

这种氮气和氧气混合物的粘度和导热曲线如图 5.2 所示。注意温度达到 15000K 以上是，它们是常数。

这揭示了有关气体模块中实现的一些信息。氮和氧的输运系数曲线在 15000K 以下是有效的。除此之外，我们没有任何信心继续对多项式的可靠性进行评估。我们在实现中所做的选择是将这些值视为超过这一点的常量。在流体模型中，这通常会引入很小的错误，因为分子氮和分子氧只在 15000K 时非常少量地存在。相反，我们预计原子和它们的离子会更加普遍，即该温度下假设它们都存在。

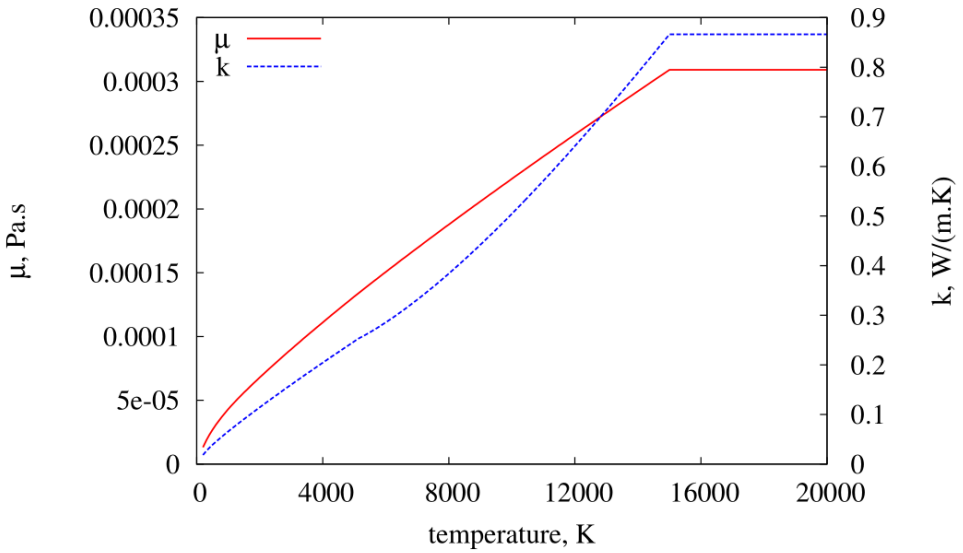


图 5.2：$N_2$-$O_2$ 混合物的粘度$\mu$和热导率 $k$ 值随温度的变化。

## 5.3 理想 Brayton 布雷顿循环

这个例子取自 Çengel 和 Boles 热力学文件[4]的例子 9-5，并计算了理想封闭

布雷顿循环的性能，如图 5.3 所示。循环组分之间的气体流动用蓝色表示，循环中关键点的气体状态编号为 1 到 4。在热交换器中，热流流入和流出的空气工作流体显示为红色箭头。

最初的问题描述是:“在理想的布雷顿循环中运行的燃气轮机发电厂的压力比为 8。压缩机进口温度为 300K，涡轮进口温度为 1300K。利用空气标准假设，确定(a)压缩机和涡轮机出口的气体温度、(b)后功率、(c)热效率”。

将状态 1 和状态 2 分别指定为压缩机的进口和出口，并将状态 3 和状态 4 指定为涡轮机的进口和出口，下面所示的 Lua 脚本计算了上面描述中选哟求的数据，结果稍后将在列表 7 中显示。我们已经尝试使变量名称顾名思义。

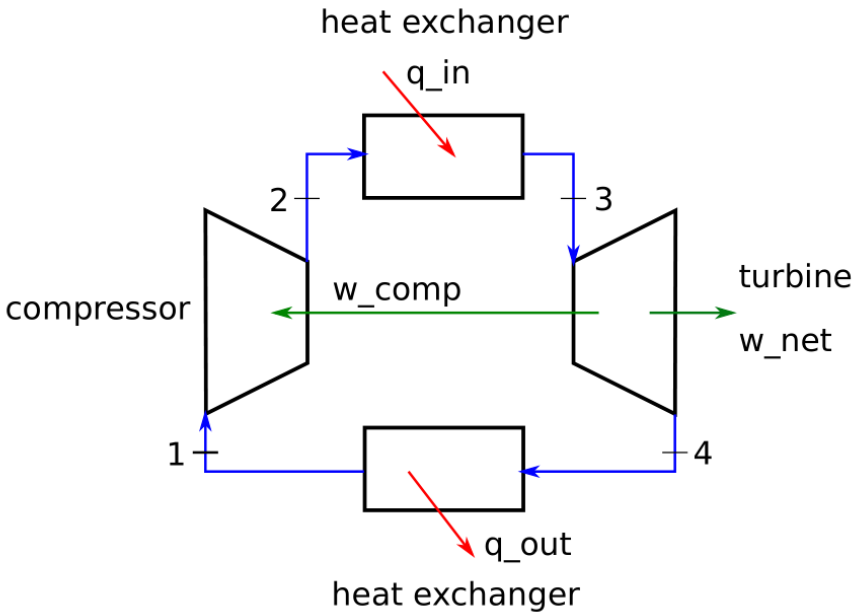


图 5.3:闭合 Brayton 布雷顿循环示意图。涡轮的部分功直接进入压气机，其余的标记为 w_net。

```
1 -- brayton.lua
2 -- Simple Ideal Brayton Cycle using air-standard assumptions.
3 -- Corresponds to Example 9-5 in the 5th Edition of
4 -- Cengal and Boles' thermodynamics text.
5 --
6 -- To run the script:
7 -- $ prep-gas ideal-air.inp ideal-air-gas-model.lua
8 -- $ prep-gas thermal-air.inp thermal-air-gas-model.lua
9 -- $ gas-calc brayton.lua
10 --
11 -- Peter J and Rowan G. 2016-12-19
12
13 gasModelFile = "thermal-air-gas-model.lua"
```

```
-- gasModelFile = "ideal-air-gas-model.lua" -- Alternative
gmodel = GasModel:new{gasModelFile}
if gmodel:nSpecies() == 1 then
print("Ideal air gas model.")
air_massf = {air=1.0}
else
print("Thermally-perfect air model")
air_massf = {N2=0.78, O2=0.22}
end

print("Compute cycle states:")
Q = {} -- We will build up a table of gas states
h = {} -- and enthalpies.
for i=1,4 do
Q[i] = GasState:new{gmodel}
Q[i].massf = air_massf
h[i] = 0.0
end

print(" Start with ambient air")
Q[1].p = 100.0e3; Q[1].T = 300.0
gmodel:updateThermoFromPT(Q[1])
s12 = gmodel:entropy(Q[1])
h[1] = gmodel:enthalpy(Q[1])

print(" Isentropic compression with a pressure ratio of 8")
Q[2].p = 8*Q[1].p
gmodel:updateThermoFromPS(Q[2], s12)
h[2] = gmodel:enthalpy(Q[2])

print(" Constant pressure heat addition to T=1300K")
Q[3].p = Q[2].p; Q[3].T = 1300.0
gmodel:updateThermoFromPT(Q[3])
h[3] = gmodel:enthalpy(Q[3])
s34 = gmodel:entropy(Q[3])

print(" Isentropic expansion to ambient pressure")
Q[4].p = Q[1].p
gmodel:updateThermoFromPS(Q[4], s34)
h[4] = gmodel:enthalpy(Q[4])

print("")
print("State Pressure Temperature Enthalpy")
print(" kPa K kJ/kg")
```

```
58 print("----------------------------------------")
59 for i=1,4 do
60 print(string.format(" %4d %10.2f %10.2f %10.2f",
61 i, Q[i].p/1000, Q[i].T, h[i]/1000))
62 end
63 print("----------------------------------------")
64 print("")
65 print("Cycle performance:")
66 work_comp_in = h[2] - h[1]
67 work_turb_out = h[3] - h[4]
68 heat_in = h[3] - h[2]
69 rbw = work_comp_in / work_turb_out
70 eff = (work_turb_out-work_comp_in) / heat_in
71 print(string.format(" turbine work out = %.2f kJ/kg", work_turb_out/1000))
72 print(string.format(" back work ratio = %.3f", rbw))
73 print(string.format(" thermal_efficiency = %.3f", eff))
```

在脚本中，我们首先构建一个合适的气体模型。热完全空气模型是基于氮气和氧气的混合物，以及依赖于温度的热平衡模型。气体模型数据由 prep-gas 程序从物种数据库中选择，输入如下：

model = ’thermally perfect gas’

species = {’N2’, ’O2’}

根据所构建的气体模型，第 16 至 22 行设置了用于构建 GasState 对象的质量分数列表。第 25 到 31 行构造这些 GasState 对象，并将它们保存在名为 Q 的数组中，选择该数组有一些不幸运，因为在热力学课本中，q 常被用来表示热流。无论如何，我们可以还是用这个符号。

第 33 行到第 53 行是本示例的核心。计算了气体的状态，一次一个，直到整个循环被定义。因为布雷顿循环是一个稳态流动机器，所以使用焓差来计算热流和功流是很方便的，如第 66 到 68 行所做的那样。列表 7 显示了计算结果，计算值与 Çengel 和 Boles 的文本中找到的值非常接近。

```
Thermally-perfect air model
Compute cycle states:
   Start with ambient air
   Isentropic compression with a pressure ratio of 8
   Constant pressure heat addition to T=1300K
   Isentropic expansion to ambient pressure
State Pressure Temperature Enthalpy
            kPa      K              kJ/kg
```

```
----------------------------------------
1       100.00      300.00      1.87
2       800.00      539.08      247.11
3       800.00      1300.00     1106.49
4       100.00      771.40      496.25
----------------------------------------
Cycle performance:
   turbine work out = 610.23 kJ/kg
   back work ratio = 0.402
   thermal_efficiency = 0.425
```

列表 7：封闭 Brayton 循环的计算状态和性能。

# 6. Lua API 的参考指南

## 6.1 全局常量

在使用为 gas-calc 编写的程序时，为了方便用户，设置了一些全局常量。这些都是物理常数，如下所示。

R_universal：通用气体常数，其值为：8.31451J/(mol.K)。

P_atm：大气压，单位为帕斯卡。$P_{atm}$=101.325×$10^3$Pa.

## 6.2 GasState

Lua 表中的 GasState 有如下指定项目：

```
GasState = {
    rho = <density, float>, -- kg/m^3
    p = <pressure, float>, -- Pa
    T = <(transrotational) temperature, float>, -- K
    p_e = <electron pressure, float>, -- Pa
    a = <sound speed, float>, -- m/s
    u = <internal energy (in transrotational mode), float>, -- J/kg
    u_modes = <internal energies in other modes, array of floats>, J/kg
    T_modes <temperatures in other modes, array of floats>, K
    mu = <dynamic viscosity, float>, -- Pa.s
    k = <thermal conductivity (in transrotational mode), float>,
                    -- W/(m.K)
    k_modes <thermal conductivities in other modes, array of floats>,
                    -- W/(m.K)
    sigma = <electrical conductivity, float>, -- S/m
    massf = <mass fractions, array of float>, -- no units
    quality = <vapour quality, float> -- no units
}
```

## 6.3 GasModel

### 6.3.1 构造函数

GasModel:new{filename}：通过从 filename 中读取指令来初始化气体模型。返回一个 GasModel 对象。

### 6.3.2 常规服务方法

GasModel:nSpecies()：返回气体混合物中物种的数量。

GasModel:nModes()：返回气体混合物中非平衡能量模式的数目。对于单温度气体模型，这个数字应该是零。

GasModel:molMasses()：返回一个以 kg/mole 为单位的组分的分子质量表。表的关键词是物种名称，值是它们的分子质量。

GasModel:speciesName(isp)：返回物种 isp 的名称。在 Lua API 中，物种从 0 开始被索引。

GasModel:createGasState()：返回一个带有默认初始化字段的 GasState 表。

### 6.3.3 状态方程服务方法

GasModel:updateThermoFromPT(Q)：假设气 GasState 中 Q 的压力、温度和质量分数是最新的，计算热力学状态。更新 Q 中的密度和内能值。

GasModel:updateThermoFromRHOU(Q)：计算热力学状态，假设密度，内能和质量分数是最新的气体状态 Q。更新 Q 中压力和温度值。

GasModel:updateThermoFromRHOT(Q)：在 GasState 的 Q 中，假设密度、温度和质量分数是最新的，计算热力学状态。

GasModel:updateThermoFromRHOP(Q)：在 GasState 的 Q 中，假设密度、压力和质量分数是最新的，计算热力学状态。

### 6.3.4 输运系数服务方法

GasModel:updateTransCoeffs(Q)：根据 GasState 的 Q 中最新的热动力学状态计算粘度和导热系数。更新 Q 中的 mu 和 k 值。

### 6.3.5 返回值的常用服务方法

下面所有的方法都假设 Q 中的热力学状态是最新的。在使用这些服务方法之前，通常应该调用其中一个热力学更新方法。下面的所有方法都向调用者返回一个值或一个数组值。

GasModel:dpdrhoConstT(Q)：计算并返回导数值：$\left(\frac{\partial \mathrm{p}}{\partial \rho}\right)_T$；

GasModel:intEnergy(Q)：计算并返回气体混合物的内能值 $u$；

GasModel:enthalpy(Q)：计算并返回气体混合物的焓值 $h$；

GasModel:enthalpy(Q, i)：计算并返回种类 $i$ 的焓值 $h_i$；

GasModel:entropy(Q)：计算并返回气体混合物的熵值 $s$；

GasModel:entropy(Q, i)：计算并返回种类 $i$ 的熵值 $s_i$；

GasModel:Cv(Q)：计算并返回气体混合物体积恒定时的比热 $C_v$；

GasModel:dudTConstV(Q)：该方法是 GasModel:Cv 的别名，因为 $C_v$ 的定义是$\left(\frac{\partial u}{\partial T}\right)_v$。它以这种形式提供，因为它通常出现在表达式中的热力学导数中；

GasModel:Cp(Q)：计算并返回气体混合物在恒压下的比热 $C_p$；

GasModel:dhdTConstP(Q)：该方法是 GasModel:Cp 的别名，因为 $C_p$ 的定义是$\left(\frac{\partial h}{\partial T}\right)_p$。它以这种形式提供的，因为它通常出现在表达式中的热力学导数中；

GasModel:R(Q)：计算并返回气体混合物的气体比常数 R；

GasModel:gasConstant(Q)：是 GasModel:R()的别名；

GasModel:gamma(Q)：计算并返回气体混合物的比热比 $\gamma=C_p/C_v$;

GasModel:molMass(Q): 计算并返回气体混合物的分子质量。

### 6.3.6 成分转换方法

GasModel:massf2molef(Q): 使用 GasState 中的质量分数表 Q.massf，计算并返回一个摩尔分数表；

GasModel:molef2massf(molef, Q): 使用所提供的摩尔分数表和 Q 中的气体状态，计算并返回质量分数表。作为一个副作用，表 Q.massf 也被更新了。忽略返回的表，只使用 Q.massf 中更新的表可能会更方便；

GasModel:massf2conc(Q)：使用 GasState 中的质量分数表 Q.massf，计算并返回一个物种浓度表；

GasModel:conc2massf(molef, Q)：使用所提供的浓度表和 Q 中的气体状态，计算并返回质量分数表。作为一个副作用，表 Q.massf 也被更新了。忽略返回的表，只使用 Q.massf 中更新的表可能会更方便。

# A. Ruby 和 Python3 中的例子

本指南全文的主要部分给出了 Lua 中的示例。在这里，用 Ruby 和 Python 脚本提供了相同的示例。与 gas-calc 程序中运行的 Lua 脚本不同，这些 Ruby 和 Python 脚本都是通过各自的语言解释器作为主程序运行的，并且 gasmodule 是作为动态库加载的。

```
$ irb --noecho
irb(main):001:0> $LOAD_PATH << '~/dgdinst/lib'
irb(main):002:0> require 'eilmer/gas'
irb(main):003:0> gmodel = GasModel.new('ideal-air-gas-model.lua')
irb(main):004:0> gs = GasState.new(gmodel)
irb(main):005:0> gs.p = 1.0e5
irb(main):006:0> gs.T = 300.0
irb(main):007:0> gs.update_thermo_from_pT()
irb(main):008:0> puts "Density of air= #{gs.rho}"
Density of air= 1.161022517662897
irb(main):009:0>
$ python3
Python 3.6.8 (default, Oct 7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from eilmer.gas import GasModel, GasState
>>> gmodel = GasModel('ideal-air-gas-model.lua')
>>> gs = GasState(gmodel)
>>> gs.p = 1.0e5
>>> gs.T = 300.0
>>> gs.update_thermo_from_pT()
>>> print("Density of air=", gs.rho)
Density of air= 1.161022517662897
>>>
```

对于所有三种脚本语言，实际计算都是由相同的 gas 模型(在 D 中实现)完成的。Ruby 和 Python3 都有方便的小工具，因此您可以查找特定方法的细节。例如，在上面的 Python3 环境中，您可以通过 help(GasState)查询 GasState 类的详细信息。

# A.1 等熵膨胀

**Ruby：**

```
# Step through a steady isentropic expansion,
# from stagnation condition to sonic condition.
#
# $ prep-gas ideal-air.inp ideal-air-gas-model.lua
# $ ruby isentropic-air-expansion.rb
#
# Ruby port, PJ, 2019-11-21
#
$LOAD_PATH << '~/dgdinst/lib'
require 'eilmer/gas'
gmodel = GasModel.new('ideal-air-gas-model.lua')
gs = GasState.new(gmodel)
gs.p = 500e3 # Pa
gs.T = 300.0 # K
gs.update_thermo_from_pT()
# Compute enthalpy and entropy at stagnation conditions
h0 = gs.enthalpy
s0 = gs.entropy
# Set up for stepping process
dp = 1.0 # Pa, use 1 Pa as pressure step size
gs.p = gs.p - dp
mach_tgt = 1.0
# Begin stepping until Mach = mach_tgt
while true do
   gs.update_thermo_from_ps(s0)
   h1 = gs.enthalpy
   v1 = Math.sqrt(2 * (h0 - h1))
   gs.update_sound_speed()
   m1 = v1/gs.a
   if m1 >= mach_tgt then
      puts "Stopping at Mach=%g" % [m1]
      break
   end
   gs.p = gs.p - dp
end
puts "Gas properties at sonic point are:"
puts "p=%g T=%g" % [gs.p, gs.T]
```

**Python3：**

```
# Step through a steady isentropic expansion,
# from stagnation condition to sonic condition.
#
# $ prep-gas ideal-air.inp ideal-air-gas-model.lua
# $ python3 isentropic-air-expansion.py
#
# Python port, PJ, 2019-11-21
#
import math
from eilmer.gas import GasModel, GasState
gmodel = GasModel('ideal-air-gas-model.lua')
gs = GasState(gmodel)
gs.p = 500e3 # Pa
gs.T = 300.0 # K
gs.update_thermo_from_pT()
# Compute enthalpy and entropy at stagnation conditions
h0 = gs.enthalpy
s0 = gs.entropy
# Set up for stepping process
dp = 1.0 # Pa, use 1 Pa as pressure step size
gs.p = gs.p - dp
mach_tgt = 1.0
# Begin stepping until Mach = mach_tgt
while True:
    gs.update_thermo_from_ps(s0)
    h1 = gs.enthalpy
    v1 = math.sqrt(2 * (h0 - h1))
    gs.update_sound_speed()
    m1 = v1/gs.a
    if m1 >= mach_tgt:
        print("Stopping at Mach=%g" % m1)
        break
    gs.p = gs.p - dp
print("Gas properties at sonic point are:")
print("p=%g T=%g" % (gs.p, gs.T))
```

## A.2 分子氧的 Cp 和焓变曲线

**Ruby：**

```
# A script to output Cp and h for O2 over temperature range 200--20000 K.
#
# Author: Peter J. and Rowan J. Gollan
# Date: 2019-11-21
#
# To run this script:
# $ prep-gas O2.inp O2-gas-model.lua
# $ ruby thermo-curves-for-O2.rb
#
$LOAD_PATH << '~/dgdinst/lib'
require 'eilmer/gas'
gasModelFile = 'O2-gas-model.lua'
gmodel = GasModel.new(gasModelFile)
gs = GasState.new(gmodel)
gs.p = 1.0e5 # Pa
gs.massf = {"O2"=>1.0}
outputFile = 'O2-thermo.dat'
puts "Opening file for writing: %s" % outputFile
f = open(outputFile, "w")
f.write("# 1:T[K] 2:Cp[J/kg/K] 3:h[J/kg]\n")
lowT = 200.0
dT = 100.0
(0..198).each do |i|
gs.T = dT * i + lowT
gs.update_thermo_from_pT()
f.write(" %12.6e %12.6e %12.6e\n" % [gs.T, gs.Cp, gs.enthalpy])
end
f.close()
puts "File closed. Done."
```

**Python3：**

```
# A script to output Cp and h for O2 over temperature range 200--20000 K.
#
# Author: Peter J. and Rowan J. Gollan
# Date: 2019-11-21
#
# To run this script:
# $ prep-gas O2.inp O2-gas-model.lua
# $ python3 thermo-curves-for-O2.py
#
```

```
from eilmer.gas import GasModel, GasState
gasModelFile = 'O2-gas-model.lua'
gmodel = GasModel(gasModelFile)
gs = GasState(gmodel)
gs.p = 1.0e5 # Pa
gs.massf = {"O2":1.0}
outputFile = 'O2-thermo.dat'
print("Opening file for writing: %s" % outputFile)
f = open(outputFile, "w")
f.write("# 1:T[K] 2:Cp[J/kg/K] 3:h[J/kg]\n")
lowT = 200.0
dT = 100.0
for i in range(199):
    gs.T = dT * i + lowT
    gs.update_thermo_from_pT()
    f.write(" %12.6e %12.6e %12.6e\n" % (gs.T, gs.Cp, gs.enthalpy))
f.close()
print("File closed. Done.")
```

## A.3 N2-O2 混合物的粘度和热导率

**Ruby:**

```
# A script to compute the viscosity and thermal conductivity
# of air (as a mixture of N2 and O2) from 200 -- 20000 K.
#
# Author: Peter J. and Rowan J. Gollan
# Date: 2019-11-21
#
# To run this script:
# $ prep-gas thermally-perfect-N2-O2.inp thermally-perfect-N2-O2.lua
# $ ruby transport-properties-for-air.rb
#
$LOAD_PATH << '~/dgdinst/lib'
require 'eilmer/gas'
gasModelFile = 'thermally-perfect-N2-O2.lua'
gmodel = GasModel.new(gasModelFile)
gs = GasState.new(gmodel)
gs.p = 1.0e5 # Pa
gs.massf = {"N2"=>0.78, "O2"=>0.22} # approximation for the composition of air
outputFile = 'trans-props-air.dat'
puts "Opening file for writing: %s" % outputFile
```

```
f = open(outputFile, "w")
f.write("# 1:T[K] 2:mu[Pa.s] 3:k[W/(m.K)]\n")
lowT = 200.0
dT = 100.0
(0..198).each do |i|
gs.T = dT * i + lowT
gs.update_thermo_from_pT()
gs.update_trans_coeffs()
f.write(" %12.6e %12.6e %12.6e\n" % [gs.T, gs.mu, gs.k])
end
f.close()
puts "File closed. Done."
```

**Python3:**

```
# A script to compute the viscosity and thermal conductivity
# of air (as a mixture of N2 and O2) from 200 -- 20000 K.
#
# Author: Peter J. and Rowan J. Gollan
# Date: 2019-11-21
#
# To run this script:
# $ prep-gas thermally-perfect-N2-O2.inp thermally-perfect-N2-O2.lua
# $ python3 transport-properties-for-air.py
#
from eilmer.gas import GasModel, GasState
gasModelFile = 'thermally-perfect-N2-O2.lua'
gmodel = GasModel(gasModelFile)
gs = GasState(gmodel)
gs.p = 1.0e5 # Pa
gs.massf = {"N2":0.78, "O2":0.22} # approximation for the composition of air
outputFile = 'trans-props-air.dat'
print("Opening file for writing: %s" % outputFile)
f = open(outputFile, "w")
f.write("# 1:T[K] 2:mu[Pa.s] 3:k[W/(m.K)]\n")
lowT = 200.0
dT = 100.0
for i in range(199):
gs.T = dT * i + lowT
gs.update_thermo_from_pT()
gs.update_trans_coeffs()
f.write(" %12.6e %12.6e %12.6e\n" % (gs.T, gs.mu, gs.k))
```

```
f.close()
print("File closed. Done.")
```

## A.4 理想的 Brayton 布雷顿循环

**Ruby：**

```
# brayton.rb
# Simple Ideal Brayton Cycle using air-standard assumptions.
# Corresponds to Example 9-5 in the 5th Edition of
# Cengal and Boles' thermodynamics text.
#
# To run the script:
# $ prep-gas ideal-air.inp ideal-air-gas-model.lua
# $ prep-gas thermal-air.inp thermal-air-gas-model.lua
# $ ruby brayton.rb
#
# Peter J and Rowan G. 2019-11-21
$LOAD_PATH << '~/dgdinst/lib'
require 'eilmer/gas'
gasModelFile = "thermal-air-gas-model.lua"
# gasModelFile = "ideal-air-gas-model.lua" # Alternative
gmodel = GasModel.new(gasModelFile)
if gmodel.n_species == 1 then
puts "Ideal air gas model."
air_massf = {"air"=>1.0}
else
puts "Thermally-perfect air model."
air_massf = {"N2"=>0.78, "O2"=>0.22}
end
puts "Compute cycle states:"
gs = [] # We will build up an array of gas states
h = [] # and enthalpies.
# Note that we want to use indices consistent with the Lua script,
# so we set up 5 elements but ignore the one with 0 index.
5.times do
gs << GasState.new(gmodel)
h << 0.0
end
(1..4).each do |i|
gs[i].massf = air_massf
```

```
end
puts " Start with ambient air"
gs[1].p = 100.0e3; gs[1].T = 300.0
gs[1].update_thermo_from_pT()
s12 = gs[1].entropy
h[1] = gs[1].enthalpy
puts " Isentropic compression with a pressure ratio of 8"
gs[2].p = 8*gs[1].p
gs[2].update_thermo_from_ps(s12)
h[2] = gs[2].enthalpy
puts " Constant pressure heat addition to T=1300K"
gs[3].p = gs[2].p; gs[3].T = 1300.0
gs[3].update_thermo_from_pT()
h[3] = gs[3].enthalpy
s34 = gs[3].entropy
puts " Isentropic expansion to ambient pressure"
gs[4].p = gs[1].p
gs[4].update_thermo_from_ps(s34)
h[4] = gs[4].enthalpy
puts ""
puts "State Pressure Temperature Enthalpy"
puts " kPa K kJ/kg"
puts "---------------------------------------"
(1..4).each do |i|
puts " %4d %10.2f %10.2f %10.2f" %
[i, gs[i].p/1000, gs[i].T, h[i]/1000]
end
puts "---------------------------------------"
puts ""
puts "Cycle performance:"
work_comp_in = h[2] - h[1]
work_turb_out = h[3] - h[4]
heat_in = h[3] - h[2]
rbw = work_comp_in / work_turb_out
eff = (work_turb_out-work_comp_in) / heat_in
puts " turbine work out = %.2f kJ/kg" % [work_turb_out/1000]
puts " back work ratio = %.3f" % [rbw]
puts " thermal_efficiency = %.3f" % [eff]
```

## Python3：

```
# brayton.py
# Simple Ideal Brayton Cycle using air-standard assumptions.
# Corresponds to Example 9-5 in the 5th Edition of
# Cengal and Boles' thermodynamics text.
#
# To run the script:
# $ prep-gas ideal-air.inp ideal-air-gas-model.lua
# $ prep-gas thermal-air.inp thermal-air-gas-model.lua
# $ python3 brayton.rb
#
# Peter J and Rowan G. 2019-11-21
from eilmer.gas import GasModel, GasState
gasModelFile = "thermal-air-gas-model.lua"
# gasModelFile = "ideal-air-gas-model.lua" # Alternative
gmodel = GasModel(gasModelFile)
if gmodel.n_species == 1:
print("Ideal air gas model.")
air_massf = {"air":1.0}
else:
print("Thermally-perfect air model.")
air_massf = {"N2":0.78, "O2":0.22}
print("Compute cycle states:")
gs = [] # We will build up a list of gas states
h = [] # and enthalpies.
# Note that we want to use indices consistent with the Lua script,
# so we set up 5 elements but ignore the one with 0 index.
for i in range(5):
gs.append(GasState(gmodel))
h.append(0.0)
for i in range(1,5):
gs[i].massf = air_massf
print(" Start with ambient air")
gs[1].p = 100.0e3; gs[1].T = 300.0
gs[1].update_thermo_from_pT()
s12 = gs[1].entropy
h[1] = gs[1].enthalpy
print(" Isentropic compression with a pressure ratio of 8")
gs[2].p = 8*gs[1].p
gs[2].update_thermo_from_ps(s12)
h[2] = gs[2].enthalpy
print(" Constant pressure heat addition to T=1300K")
gs[3].p = gs[2].p; gs[3].T = 1300.0
gs[3].update_thermo_from_pT()
```

```
h[3] = gs[3].enthalpy
s34 = gs[3].entropy
print(" Isentropic expansion to ambient pressure")
gs[4].p = gs[1].p
gs[4].update_thermo_from_ps(s34)
h[4] = gs[4].enthalpy
print("")
print("State Pressure Temperature Enthalpy")
print(" kPa K kJ/kg")
print("----------------------------------------")
for i in range(1,5):
print(" %4d %10.2f %10.2f %10.2f" %
(i, gs[i].p/1000, gs[i].T, h[i]/1000))
print("----------------------------------------")
print("")
print("Cycle performance:")
work_comp_in = h[2] - h[1]
work_turb_out = h[3] - h[4]
heat_in = h[3] - h[2]
rbw = work_comp_in / work_turb_out
eff = (work_turb_out-work_comp_in) / heat_in
print(" turbine work out = %.2f kJ/kg" % (work_turb_out/1000))
print(" back work ratio = %.3f" % (rbw))
print(" thermal_efficiency = %.3f" % (eff))
```

# 参考文献

[1]. E. Bender. Equations of state exactly representing the phase behavior of pure substances. In Proceedings of the Fifth Symposium on Thermophys. Prop., ASME, New York, 1970.

[2]. B. J. McBride and S. Gordon. Computer program for calculation of complex chemical equilibrium compositions and applications. Part 2: Users manual and program description. Reference Publication 1311, NASA, 1996.

[3]. R. Ierusalimschy. Programming in Lua. Lua.org, PUC-Rio, Brazil, 2nd edition, 2006.

[4]. Y.A. C¸engel andM. A. Boles. Thermodynamics: AnEngineering Approach. McGraw Hill, 5th edition, 2006.