CFD 软件使用指北

Nektar++ & Gmsh etc. 入门

Erllane

2024 年 5 月 22 日

cogito ergo sum

目录

Chapter 1

软件介绍与命令、脚本基础

1.1 写在前面: 手册食用指南

本册是 CFD 软件的快速入门手册, 仅用于快速上手, 解决一些比较常见的可能遇到的问题 (实则不太够用, 编写精力与篇幅都比较有限);

本册介绍了软件的主要功能和上手流程, 也简要概括了一些自主学习的方法; 其中有一部分比较常见的问题也可以速查本手册寻找解决方案;

对应文中有本册内的章节标注的地方都设置了交叉引用, 可以直接跳转; 紫色字的部分也设置了网页跳转; 其余 user guide 部分请自行翻阅.

本册内容在 Nektar++ 的部分功能讲解时用例有所贯穿, 可以点击跳转了解对应参数/选项的由来与传递流程;Gmsh 部分主体还是参考其使用手册, 此处只会列一些比较常用的命令, 不太会的命令可以进入 GUI 界面设置一个看看结构 (Gmsh 的 guide 上写得不够详细), 再用脚本编写 (有些命令 GUI 的支持度不太够);

1.2 主要软件介绍

这里主要介绍一下 CFD 常用的一些软件, 后续有可能会用到, 不过其实网上也能搜到, 所以不作赘述了; 有需要可自行了解;

CFD 常用的软件主要分为三大类型:

- (1) 前处理: 网格绘制软件;
- (2) 数据生成: 本类软件主要通过运行内置计算格式或自行编写计算的格式, 生成流场;
- (3) 后处理: 主要指流场数据查看与观测;

当然, 也有一些商业软件内部对其中的 2-3 部分进行了集成 (如 Fluent); 因为以下主要介绍的包含以下流程, 所以此处详细说明下相关流程.

1.2.1 前处理/网格绘制

CFD 实验选手有相当大部分时间都会花在网格处理上, 一个好的网格会让计算事半功倍;

- Gmsh

是以下主要介绍的网格绘制软件; 属于轻量级的开源软件, 目前课题使用的网格使用本软件进行绘制完全足够;

- ICEM

可以参考这篇知乎上的文章, 下同 (网格绘制)

- COMSOL

这个软件见到过有同行在用，好像也还可以。

- ANSYS Meshing

ANSYS 是很有名的有限元工具/厂商;

- Spacecliam

以上软件均能通过软件内部的 GUI 界面进行网格绘制; 但是 Gmsh 很多功能需要文本编写; 而且本人更倾向于通过码字绘制网格;

1.2.2 数据生成

下文主要介绍的是 Nektar++ 软件 (开源);

如今 CFD 工业业界大多还是用商软的 (Fluent), 据说操作比较傻瓜, 软件有 GUI 界面;

Nektar++ 为开源软件, 没有 GUI 界面; 和 Fluent 相似, 也是内部集成的格式算法; 其有使用指南, 也有开发者手册, 支持对底层代码进行修改;

OpenFOAM 也是开源软件, 其内部有单独的一套语言系统, 需要自行编写计算格式, 学术圈子里用 OpenFOAM 也比较广泛; 另外, 用 Star CCM+ 的也是大有人在;

组里还有一些其他程序 (如 openpipe), 我也没有用过...

像老师之前用过的, 还有Nek5000之类的软件, 总体而言, 总体趋势是朝着使用开源软件靠拢, 组内的自由软件因为只能算一些相对简单的几何网格, 泛用性远远达不到日常使用需求..

1.2.3 后处理/渲染与观测

以下仅介绍 Paraview(开源) 与 Tecplot(闭源商软); 当然, 导出数据用 matlab,python 进行流场/流场切片的数据处理也不失为一种选择;

1.3 Linux 命令行基础及 vim 工具

1.3.1 命令行基础/常用命令

```
cd xxx(文件夹名称)
```

(1) 此命令用于打开文件夹;

(2) 若直接执行 cd 命令, 则返回根目录;

(3) cd 打开的路径可以是相对路径 (比如: 终端在当前文件夹环境下, 可以打开当前文件夹目录下的二级文件夹), 也可以是绝对路径 ("~" 表示根目录, 例如,desktop 是 home 根目录下的文件夹, 要打开 desktop 文件夹下的 build 文件夹, 可以执行 cd ~/desktop/build);

(4) cd .. 返回上级文件夹 (.. 作用在表示路径时同理), 返回两级可以用 cd ../.. , 单个. 表示当前目录

```
ls
```

列出当前文件夹下的所有文件及文件夹 (在纯命令行、没有图形界面时会经常用到这个命令)

```
pwd
```

显示当前文件夹路径

```
mkdir xxx
```

创建名为 xxx 的文件夹;

```
mv 路径1/文件A 路径2
```

将路径 1 下的文件 A 移动到路径 2 下; 也可以通过该命令实现文件/文件夹的重命名 (eg. mv A B 将 A 重命名为 B);

```
rm lll.xxx
```

删除名为"lll.xxx" 的文件;"rm -r xxx" 为删除"xxx" 文件夹.

```
cp ...../lll.xxx(源文件路径) ..../(目标路径文件夹)
```

复制命令, 将前边路径下的文件复制到后边的路径下;

```
bash xxx.sh
```

运行名为 xxx 的 sh 脚本;

```
source xxx.sh
```

导入环境变量 (会在 Slurm 系统部分详细说明);

```
less xxx
```

预览名为 xxx 的文件 (无法改动), 优势在于速度比较快 (不是应用), 按 q 退出;

```
tail -123 xxx.txt > out.txt
```

输出 xxx.txt 的最后 123 行, 后边的指向 out.txt 为输出 (可不加);

```
head -123 xxx.txt > out.txt
```

(同上) 输出 xxx.txt 的最初的 123 行, 后边的指向 out.txt 为输出 (可不加);

```
ln -s 源文件(路径1) 目标文件(路径2)
```

创建快捷方式 (Linux 中称为软链接), 其中源文件路径必须为绝对路径, 目标路径可以是相对路径;

```
ls -l quicklink
```

查看名为 quicklink 的快捷方式的原始路径;

```
gnome-system-monitor
```

任务管理器 (仅本地);

注 1: 在命令行输入路径时, 输入 1 个以上字母后, 按 Tab 键可以进行自动补全.
注 2: 快捷键"ctrl+Alt+T" 可呼出终端.
注 3: for 循环的脚本实现.
注 4: 根目录下可通过"vim .bashrc" 设置环境变量, 如果要改, 就需要再加入一些原始系统默认的 Bash 路径, 参考这篇文章.

1.3.2 vim 工具

Ubuntu 系统可能没有预装 vim, 只安装了 vi, 所以需要先安装以下, 熟悉以下其使用 (服务器上会经常用到, 类似的应用还有 nano);

安装命令

```
sudo apt-get install vim
```

安装后在终端通过命令行打开, 命令"vim xxx"(可以不带格式); 可以通过以上方式查看编辑已有文件, 也可以通过此方式在该目录下创建一个名为"xxx" 的文件 (需要编辑并保存)

一般常用的命令只有以下几种, 其余可根据需要自行了解 (操作指南and官方 Tutorial);

进入 vim 后是无法编辑的, 可按"i" 进入编辑状态, 在结束编辑时按"esc" 键, 输入":q" 退出 (不进入编辑状态也是这样退出); 输入"wq" 保存并退出 (针对编辑过的状态);":q!" 强制退出 (:wq! 同理强制, 不过强制退出不会保存);

也可按 Shift+z 进行退出, 不过很少有人这样做就是了;

再介绍一个操作 (一般用于检查网格 xml 文件),"ctrl+end" 快速移动到文档结尾,"ctrl+home" 同理;

1.4 Slurm 调度系统使用入门

超算平台均使用 Slurm 调度系统, 天河系与其他超算在脚本语言上略有不同, 调度器语言也存在一定差异 (不过差别不大);

先来讲一讲超算平台运行作业的流程:

- [1] 编写程序所需的输入文件, 确定参数;
- [2] 编写脚本文件, 使用调度器命令运行脚本;
- [3] 排队等待或运行, 结束;

1.4.1 脚本编写

一般而言, 脚本编写遵循以下步骤: [1] 不论是 Slurm 脚本/服务器脚本, 还是对本地脚本而言, 均以以下内容为起始.

```
#!/bin/bash
```

[2] 编写 SBATCH 的调度指令, 格式为

```
#SBATCH -J TEST
```

具体的参数对应关系 (区分大小写), 主要如下:

-J	Job-Name	指定脚本运行的任务名称
-p	Partition-Name	指定脚本的运行队列 (一般服务器运营商会给 or 使用手册会有)
-n	384	脚本运行的并行任务数 (一般而言为 CPU 数/默认 1 核 1 任务)
-N	8	脚本运行调用的节点数
–ntasks-per-node=	64	单节点调用 CPU 数 (单节点 24/32/56/64 核不等)
–exclusive		独占节点

除此之外, 还有许多可选参数, 超算的手册上也不一定会写, 可以参考 Slurm 的官方指南, 或者在服务器命令行输入 SBATCH –help; 下面再额外列出几个参数作为了解:

- [1]-c 指定 CPU 数, 一般默认 1 任务 1CPU, 效率相对较高, 不用额外设置;
- [2]- -mem= 指定调用的内存大小; 一般不设置此参数, 系统默认将该 CPU 所有可用内存预分配下来;
- [3]- -mem-per-cpu= 单 cpu 调用内存大小;

然后加载环境 (主要是 MPI 及编译器), 代码如下:

```
source env.sh
```

一般来说, 因为在服务器上并行跑程序, 需要 mpirun, 代码如下 (以不可压不稳定 NS 求解器为例)

```
mpirun –np 384(n后参数) ./.../INSS ––npz 32 配置.xml 网格.xml
```

至此, 脚本编写完成. 运行脚本只需

```
sbatch –n 384 INSSBash.sh(脚本文件名)
```

其中"-n 384" 可以换成"-N 8", 亦或者省略不加 (–npz 是 Nektar++ 指定的 z 方向 FFTW 并行数, 在讲解 IncNavierStokesSolver 的时候会讲到);

注:".sh" 文件与".slurm" 文件在服务器 Slurm 环境下区别不大.

1.4.2 调度器常用命令

常用命令可以参考Slurm 官方文档或者比较容易找到的各大超算平台的用户指南, 如上交超算用户手册;

下面列出几个常用命令:

sbatch	提交运行脚本 (可下线)
salloc	(几乎不用此命令) 运行脚本 (下线则停止运行)
squeue	查看排队或作业状态, 查看 JOB_ID
sinfo	查看各节点状态
scancel	+JOB_ID 可取消任务

1.4.3 module 环境加载

服务器上环境通常通过 module 命令加载;

```
module purge
```

清除所有已经搭载的环境;

```
module load ....(eg.compiler/intel/2017.5.239)
```

加载环境 (例为 intel-2017 编译器-假定);

```
module rm(unload 也可以) ....
```

卸载环境;

```
module avail
```

查看所有可用环境;

```
module list
```

1.5 额外事项

1.5.1 Ubuntu 上的 Python 部署

主要是有些地方会用到 Python 脚本, 如果在本地跑的话,Ubuntu 本地的 Python 2.7.x 不太够用, Python 3.5 安装额外的需要使用的库的时候可能会涉及 pip 版本导致的无法安装的问题; 所以这就需要再装一个更高的, 有更好适配性的本地 Python 版本 (因为系统对 Ubuntu 等系统预装的 Python 版本有较强的依赖性-比如卸载会导致浏览器打不开等, 所以按照教程直接安装即可);

这个是目前找到的最完备的一个部署教程, 按照教程安装对应所需版本即可; 使用时用 Python3.x aaa.py(x 为对应大版本号, 有预设) 呼出即可;

Chapter 2

Nektar++ 使用指南

本章及后续若不加说明, 均在 Linux 环境下进行, 考虑到 Ubuntu 版本太高时安装相对稍旧版本的 Nektar++ 时会有系统版本导致的报错, 所以建议安装轻量级的 VirtualBox 虚拟机以及 Ubuntu 16(感谢某磨豆腐大佬的资源);

VirtualBox 和虚拟机的安装从略，VirtualBox 中安装虚拟机的方法参考磨豆腐大佬公众号里此篇文章. 此外, 建议在搭建好虚拟机环境后, 在虚拟机上挂载一个 Windows 共享文件夹, 会相对方便些, 此处从略. 可参考这篇文章;

此外, 在进行软件安装前, 建议先在目录下建一个软件安装的公共文件夹, 用来存放和安装软件, 后期易于管理;

软件官网:https://www.nektar.info/;

资源下载:https://www.nektar.info/src/;

2.1 软件介绍

其实user guide的 introdution 部分也有, 不过貌似这种介绍可能不太会有人专门去了解 (除非一时兴起);

简要介绍一下软件历史 (详细可参考developer guide的 introdution 部分).

Nektar++ 是由伦敦帝国理工学院 (Imperial College London) 和美国犹他大学 (University of Utah) 合作开发的软件.

Nektar++ 这款软件是一款基于Spectral/hp(谱元法)的高精度有限元计算工具, 其最早可以追溯到上世纪的 Nektar 程序 (即 Nektar++ 的前身); 后来, 随着时代的发展, 基于 C++ 语言的结构与优秀的运行速度,Nektar++ 转向使用 C++ 进行程序底层代码编写; 如今的 Nektar++ 软件底层主要由 C++ 语言和 FORTRAN 语言编写而成, 同时支持使用 Python 接口.

2.2 软件安装

下载以上资源下载部分的安装包, 然后在虚拟机上进行安装; 建议先从 5.0.1 版本开始使用 (5.2.x 及以后版本的脚本书写在某些地方有所变化, 比如显隐式格式选择的控制, 小版本之间的差异应该不大, 具体可以参看更新日志);

接下来是具体的安装步骤, 可以参考 user-guide 里第一章的安装指南 (也可参考磨豆腐大佬公众号的文章):

编译过程从 user-guide 的 1.3.2.2 节开始; 首先, 解压下载好的源码压缩包, 然后执行命令:

```
mkdir -p build && cd build
ccmake ../
```

以上的 ccmake 命令是基于 cmake 库进行的, 所以要在运行以上命令之前, 先在本地电脑安装 cmake; 还有一些其他 Ubuntu(Linux) 系统未进行预装的库, 以下一并进行安装:

```
sudo apt-get install libboost-dev flex mpich libghc-zlib-dev
sudo apt-get install libarpack++2-dev cmake-curse-gui
```

(因为一行写不开就分成两行了 (坏), 以上安装的各组件的功能详见第5章支持库食用指南)

执行过 ccmake 命令后, 按 c 进行编译确认; 之后会进入一个截面确认安装选项, 可参考 guide 文档上, 把 recommend 的安装选项全部改为 ON, 其他的默认不做改动即可; 下边简单列一下需要重点确认的选项:

- 必要:
- [1]NEKTAR_USE_FFTW ON
- [2]NEKTAR_USE_MPI ON
- [3]NEKTAR_USE_SCOTCH ON
- [4]NEKTAR_USE_SYSTEM_BLAS_LAPACK ON
- [5]THIRDPARTY_BUILD_BOOST ON
- [6]THIRDPARTY_BUILD_SCOTCH ON
- [7]THIRDPARTY_BUILD_FFTW ON
- 可选:

- [8]THIRDPARTY_BUILD_TINYXML ON
- [9]NEKTAR_USE_HDF5 ON

确认好之后依下方提示按 c 进行编译, 可能会重复 1 到 2 次, 直到下方出现 [g] to generate 的提示, 按 g 进行生成, 生成结束后会返回控制台; 在控制台输入以下命令进行安装:

```
make install
```

软件安装一般可能要半小时到一小时不等, 甚至还会长一些, 等软件安装结束, 控制台会回到可操作状态; 当然, 也可以尝试多线程并行安装 (以 4 线程为例):

```
make -j4 install
```

另外还有一点, 就是在安装过程中需要联网下载第三方包, 需要虚拟机能够联网; 如果不能联网, 则需要在第三方库这里预先下载压缩包, 并把压缩包放到解压后的 Nektar++/Thirdparty 目录下, 再运行 make install 命令;

在服务器上安装同理;

2.3 程序使用

首先呢, 这个软件是没有 GUI 界面的, 需要用编译好的可执行文件加命令行执行代码; 因为目前组里在做的都是不稳定不可压缩 Navier-Stokes 方程的相关问题, 所以下边主要介绍 NekMesh,IncNavierStokessolver 和 FieldConvert 几个程序的具体的常用使用方法 (其他功能详细参考使用手册).

关于这几个程序的路径嘛, 分别在 $Nektar++$/build/Utilities/NekMesh,$Nektar++$/build/Utilities/FieldConvert 和 $Nektar++$/build/solvers/IncNavierStokessolver 三个目录下, 建议先从该路径下创建一个快捷方式, 再将其拷贝到 Home 目录备用, 随使用处复制; 或者, 也不用这样, 本地其实没有多大必要写脚本再 bash, 有些太麻烦了;

user guide 食用指南: 找到关键词, 在 pdf 文档里 ctrl+F 直接搜索.

为快速上手使用, 简单介绍一下程序使用流程是必要的;

第一步, 用 Gmsh 画好网格 (geo 文件, 几何信息), 然后用 Gmsh 的 GUI 或者命令行 gmsh 的生成网格命令, 生成.msh 格式的网格;

第二步, 利用 NekMesh 将.msh 文件转换为 Nektar++ 的 Solver 程序可读的.xml 格式的网格信息文件, 并修改 EXPANSION 部分的参数 (如有必要, 此步对顶点进行操作);

第三步, 编写程序部分的 xml 脚本, 记述信息同下第2.4.2节程序部分;

第四步, 将求解器 (或其快捷方式)、xml 网格信息脚本、xml 程序脚本置于同一目录下，运行求解器；

第五步, 用 FieldConvert 处理结果 chk/fld 格式的流场数据; 一般是得到可视化的.vtu 格式的流场文件, 用 Paraview 打开渲染可视化 (当然, 也有其他备选项);

2.3.1 NekMesh

2.3.1.1 msh 到 xml 的生成

主要指 xml 网格的生成; 其实这样说可能会造成一些误会, 因为 xml 文件并不是网格文件; 不过此处生成的 xml 文件是 Nektar++ 可识别的网格信息的文件, 关于其构成, 在下一个 section 会讲到.

NekMesh 的主要使用方法,(对于 gmsh 生成的.msh 格式网格) 在命令行执行以下命令 (确保 NekMesh 可执行文件的快捷方式在该目录下):

```
./NekMesh xxx.msh xxx.xml
```

这时因为安装了 TinyXML, 打开看到网格的信息, 会发现无法肉眼读取数据; 若想看到网格数据, 或对数据有直接操作, 作为代替运行以下命令即可:

```
./NekMesh xxx.msh xxx.xml:xml:uncompress
```

以下主要讲 NekMesh 的主要用法, 网格 XML 文件的详细信息在下一节会详细展开;

首先, 在生成 XML 网格信息文件之后, 对文件信息有其他处理的情况, 若需要可视化, 需用到 FieldConvert(在下面 FieldConvert 的开始部分会提到);

2.3.1.2 NekMesh 功能概览

先来讲一下 NekMesh 的命令结构;

NekMesh 事实上把不同功能拆分成了单独的程序, 进行了模块化, 互不干扰, 而 NekMesh 是一众程序的一个结合体; 在执行基础命令 (包含输入/输出) 的同时, 输入对应模块化的功能命令, 就可以一次执行多个功能, 详情参照 user guide 的 4.4 节; 具体的输入和输出格式限制也请参照 user guide 的 4.4 节;

下文没提到不代表没有该功能; 日常有可能用到的 (可能性大于零就行/逃) 都列在下面了; 其他功能请详细参考 user guide. 下文没有提到的圆滑化处理 (Spherigon patches), 基于变分法的网格修正 (Variational Optimisation) 也可以看下;

另外, 如果在安装编译时开了 MESHGEN, 可以直接用 NekMesh 生成网格 (见 user guide4.5 节, 不推荐).

2.3.1.3 msh 升阶魔法 (雾)

先说结论: 一般而言, 不太能用到;

主要是指 NekMesh 对 msh 网格文件的升阶操作;Gmsh 本身也支持生成高阶网格 (直观上讲, 顶点和有限元密度会增大, 阶数主要指生成算法的阶数), 但是 Gmsh 本身最多只能支持到三阶, 例如以下命令:

```
gmsh -3 pipe.geo -order 3
```

但是,NekMesh 甚至可以生成更高阶的网格, 例如在 Gmsh 阶段不升阶, 然后再执行以下命令可升至 7 阶:

```
NekMesh pipe.msh newpipe.msh:msh:order=7
```

一般而要, 算程序的话, 网格文件太大会有很多麻烦, 所以一般是用 Expansion 里的插值多项式阶数控制计算密度, 这个只作为了解;

此外,Nekmesh 还提供了对生成好的圆柱体的 xml 网格信息的壁面部分进行高阶光滑化处理的功能:

```
NekMesh -m cyl:surf=2:r=1.0:N=5 LinearCylinder.xml
                                        HighOrderCylinder.xml
```

以上代码就是将 Linear 网格变成半径为 1 的 5 阶壁面网格的 HighOrder 网格; 其中"surf=2" 的 2 指的是 geo 文件中赋予的 Physical Surface 的物理面指标;

所谓的壁面 5 阶网格就是将圆柱的圆形部分换成 5 阶多项式插值 (一般两个网格点都是直线, 也就是两点间用线性插值), 至于壁面上两点之间会用几段线段相连, 如果对原低阶网格在 xml 额外信息中设置相同阶数, 可以发现两个版本位置十分接近但不相同 (相对位置一致但后者是在直线上完成的); 但是这个功能比较有局限性, 它无法对非三棱柱的面元进行变换,o 型网格拼接而成的四棱柱的边界面也不在使用范围之内;

2.3.1.4 面元抽取

这个模块主要的功能是从现有的 xml 文件中抽取 Expansion 中对应标签的平面上的信息, 组成一个新的 xml 文件;

这个功能设计的本意应该是抽取面元, 然后在 FieldConvert 中使用该部分, 之抽取流场的部分信息; 这个功能可能有一点鸡肋; 因为它只能抽取现有面元的信息, 而且 xml 中的面元往往是整个几何形体的边界 (geo 文件内部中内部的面一般不会出现在 Composite 里, 和 physical 有关, 可动手试一下);

```
<C ID="41"> E[8627,8629,8631,8633,8635] </C>
```

若 xml 文件中 EXPANSION 部分如上定义, 命令可以如下:

```
NekMesh -m extract:surf=41 Mesh.xml Mesh_41.xml
```

不过更实用的是在 FieldConvert 里直接生成切片进行提取就是了;

2.3.1.5 边界层生成

偷懒达咩!

可以用 xml 中 COMPOSITE 部分的 (物理) 面信息生成一个粗浅的边界层 (用几何级数工作, 对应于 gmsh 的 Progression, 但是 NekMesh 无法实现 Bump);

```
NekMesh -m bl:surf=41:layers=3:r=1.2:nq=8 Mesh.xml M3L.xml
```

平面沿用上例, 这样的代码生成了面 41 的边界层;" 边界层" 为 3 层, 每层内点按照 1.2 的步长增长比例递增, 共 8 个积分点 (7 小层); M3L.xml 为生成后的整体网格的文件;

2.3.1.6 边界识别

聊胜于无的功能; 曾经测试过, 有一些网格这个程序识别不了; 命令如下

```
NekMesh -m detect volume.xml VWithBoundaryComposite.xml
```

2.3.1.7 平面挤出 (拉伸)

将 2 维网格挤出至 3 维; 功能上只是能够实现, 但是在泛用性上不如用 Gmsh(不支持旋转/沿导线挤出);

```
NekMesh -m extrude:layers=n:length=l 2D.xml 3D.xml
```

上述命令是把 2D 网格分 n 层挤出 (或称为拉伸) 至三维, 挤出长度为 1, 输出为 3D.xml.

2.3.2 IncNavierStokessolver

主要格式是速度修正格式 (VCS,Velocity Correction Scheme), 具体的格式内容在这里就不讲了, 这里只讲使用;

下面列一下 user guide 中需要额外看一下的小章节:11.1.1.6/11.1.3

2.3.2.1 基本用法

以下是几种主要用法; 第一行情况是 xml 文件包含了程序运行选项和网格; 第二行命令则是分开的情况; 第三种是并行运行求解器 (8 线程 4 节点-服务器); 第四种是 3DH1D(Quasi-3d) 情况对应展向有指定 Fourier 并行数的情况;

```
IncNavierStokesSolver session.xml
IncNavierStokesSolver Prog.xml Mesh.xml
mpirun -n 8 -N 4 IncNavierStokesSolver Prog.xml Mesh.xml
mpirun -n 8 -N 4 IncNavierStokesSolver --npz 2
                                        Prog.xml Mesh.xml
```

2.3.2.2 求解器信息

- **展开 (Expansion)**

主要有三种形式;

其中 Modified 比较常用;GLL_LARGERANGE 用的是在 Gauss-Labatto 点进行 Largrange 插值; 具体如下:

BASIS	TYPE
Modal	MODIFIED
Nodal	GLL_LARGERANGE
Nodal SEM	GLL_LARGERANGE SEM

- **方程形式 (EqType)**

一般都是不稳定 N-S 方程,EqType 是"UnsteadyNavierStokes"; 其他方程形式请参考手册;

- **求解格式 (SolverType)**

一共有 3 种, 一般用 VCS(速度修正格式); 其他两种为弱压的 VCS 和直接模拟 (DNS 法), 都支持 2D/Quasi-3D/3D 及 Continuous Projection; 详细信息见 11.3.1 节, 下同;

- **驱动程序 (Driver)**

一般为标准 (Standard, 时间迭代). 另一种是 **SteadyState**, 稳态求解适用; 还有 Adaptive(自适应多项式阶数, 没用过, 参数设置具体参照 11.7 节);

- **投影 (Projection)**

一般用连续伽辽金投影 (Continuous Garlerkin, 简称 CG); 另外还有离散、混合两种;

- **时间推进格式 (TimeIntegrationMethod)**

有若干种 (3 大类);IMEX 是显隐结合的时间推进格式 (1/2/3 阶, 一般用 3 阶);Backward Euler 为向后 Euler 法 (1 阶), 为隐式方法;BDF 为多步法 (向后微分方程,1/2 阶); 其中, 只有后两种纯隐式方法支持用离散 Garlerkin 投影;

另外, 时间推进格式部分脚本在 5.2.0 版本前后有所变化,before 5.2.0-like:

```
<I PROPERTY="TimeIntegrationMethod" VALUE="IMEXOrder3"/>
```

而在 5.2.0 版本之后, 写法变成了:

```
<TIMEINTEGRATIONSCHEME>
  <METHOD> IMEX </METHOD>
  <ORDER> 3 </ORDER>
</TIMEINTEGRATIONSCHEME>
```

未提到的可能有一点点可能用到的参数:Extrapolation(CG-DG 混合时使用, 用于设置子步, 默认 Standard), SubStepIntScheme(DG 子步),DEALIASING(激活 Quasi-3D 的 3/2 padding 准则, 具体机制不是很清楚, 是算对流项时候用的);

2.3.2.3 参数信息

略, 详见 XML 文件说明第2.4.2节条件设定部分;

2.3.2.4 对流扩散方程-热耦合

Nektar++ 支持在解 N-S 方程的同时求解一个对流扩散方程 (对应于 N-S 方程的热力项需要用对流扩散方程实时更新);

需要在 EXPANSION 处开始定义扩散率场 ("u,v,c1,c2,p", 以下均需要据此进行改动); 然后在函数部分定义一个新函数:

```
<FUNCTION NAME="DiffusionCoefficient">
 <E VAR="c1" VALUE="0.1" />
 <E VAR="c2" VALUE="0.01" />
</FUNCTION>
```

此处的 c1,c2 对应于对流扩散方程章节的对流扩散方程中的"ϵ".

2.3.2.5 施加恒定流量

即控制每个截面的流量 (通量) 相同, 原理是对速度进行线性修正, 在周期性边界条件的时候经常用到; 需要在 Parameter 部分设置流量值 (Flowrate), 然后在边界条件中加上 Flowrate 设定:

```
<P VAR="u" VALUE="[1]" USERDEFINEDTYPE="Flowrate" />
```

最后再定义 Flowrate 函数 (函数同第2.4.2.2节上方的 Flowrate 函数;

2.3.2.6 额外信息

目前不支持磁场力, 如有 MHD 使用需求请移步 Github(MHD), 按照原软件方式安装即可 (需要改动 Cmakefile, 具体细节从略);

移动体 (Moving Body) 的相关问题请参照 user guide;

滤波器 (Filter) 的动能与熵部分 (Kinetic energy and enstrophy) 和 Reynold Stress 部分请参照 user guide 第三章 (3.4.10/15);

稳定性分析还没做过, 有待开发;

2.3.3 FieldConvert

FieldConvert 处理的主要是和场相关的问题; 其程序架构与 NekMesh 类似, 也是多个程序的一个运行接口;

特别提示: 为节约空间, 一下./FieldConvert 可能会简写为 FC, 运行程序时不要这样写.

2.3.3.1 XML 网格可视化处理

首先, 对于一个 XML 网格信息文件, 有时会受制于操作, 需要将其可视化; 这就需要 FieldConvert 进行处理 (如下)

```
./FieldConvert xxx.xml xxx.vtu
```

其中,".vtu" 是非结构化网格的可视化文件格式 (格式类型为 VTK 类型文件, 最后一个字母用于区分网格是否为结构化网格); 可利用 Paraview 打开进行查看 (如 3.5.1 节的网格模型就是 Paraview 视图下的网格轮廓素体).

2.3.3.2 chk/fld 流场可视化处理

一般而言, 通过以上 INSS 跑出来的流场一般是 chk 或者 fld 格式的文件 (并行的时候 chk 文件夹下的各个流场文件都是 fld 类型的); 而在此步骤, 需要将该流场信息转化为可以可视化的形式;

其中一种途径是转化为 vtu 文件, 具体的命令如下例 (以及 mpi 版本):

```
./FieldConvert field.fld field.vtu
mpirun -n 128 -N 4 ./FieldConvert f.fld fa.vtu
```

其中 mpi 以 4 节点总计 128 任务为例 (一般本地虚拟机 $n \leq 8$, 不设置 N 参数); 然后可用 paraview 打开;注:mpi 生成的 vtu 文件形式是一个多任务的文件夹和一

个.pvtu 的文件, 在用 Paraview 打开时打开.pvtu 文件即可. 详细方式请参照第4.1.2节 Paraview 的基本使用步骤;

此外, 也可以生成 Tecplot 制式的 dat 文件, 可用 Tecplot 打开, 也可用 Paraview 打开 (生成方法类似, 不过 dat 文件生成只能单线程, 不能用并行进行生成); 相对地,Tecplot 对 vtu 文件的查看不如 Paraview 友好;

2.3.3.3 区域提取

在一个流场中, 只输出某一区域的可视化信息; 但是, 受制于网格形状,边界可能以 cell 的形式出现 (尤其是不规则网格); 如下便是 $0 \leq x \leq 1$,$2 \leq y \leq 3$,$4 \leq z \leq 5$ 的正方形区域的输出命令;

```
./FieldConvert -r 0,1,2,3,4,5 f.fld part.vtu
```

其实这个功能输出的是网格单元;

提取边界区域:

在2.3.1.4节有提到提取边界区域的问题, 以下命令就是针对那时的想法构筑的命令;

```
FC -m extract:bnd=41 Mesh.xml field.fld bound_41.fld
NekMesh -m extract:surf=41 Mesh.xml Mesh_41.xml
FC Mesh_41.xml bound_41.fld bound_41.dat
```

提取 Quasi-3d 情况的单个非展开方向截面:

```
FieldConvert -m homplane:planeid=[value] file.xml
                                    file.fld file-plane.fld
```

其中 value 为提取的面的 ID 编号;

2.3.3.4 流场的加减

这个功能在检测查看加扰动之后的脉动情况的时候有时会用到, 特别是当流场的同一速度场速度尺度差异过大的时候 (加噪声流场减基本流); 一般而言格式如下 (例为以上情况):

```
FC -m addfld:fromfld=base.fld:scale=-1 mesh.xml
                                    noise.fld pure.fld
```

两个流场合并取平均”-m combineAVG:fromfld=...”(模式相同)

2.3.3.5 流场的梯度

求解各个场的梯度场;

```
FieldConvert -m gradient test.xml test.fld test-grad.fld
```

2.3.3.6 流场插值 (转换)

一般的流场插值, 先说下一般步骤; 首先, 对于一个网格 Mesh1.xml 及其对应的流场 Field1.fld, 有一个几何形状与 Mesh1.xml 相同但内部网格划分不同的网格 Mesh2.xml, 用以下命令生成 Mesh2.xml 对应的流场:

```
FC -m interpfield:fromxml=Mesh1.xml:fromfld=
                                    Field1.fld Mesh2.xml Field2.fld
```

Quasi-3d 下的 Expansion 方向插值 (更改 HomModes), 同理; 假设原始的 3DH1D 流场的展向半波数为 24, 现在要变成 48;(之前还以为不行, 就重新跑了基本流... 很想吐槽,user guide 很多相似的功能不是写在一起的, 甚至找的时候也比较麻烦)

```
FC -m homstretch:factor=1 --output-point-hom-z 48
                                Mesh2D.xml field24.fld stretch_48.fld
```

其中 factor=1 是指拉伸比例为 1(不拉伸), 此处也可进行修改 (为其他倍数-在其他相应需求下); 另外"-output..." 参数不是必须参数, 此处因为需要指定输出半波数, 所以才加了这个参数;

其余若干插值功能请参考 user guide 第 5.5/6.18-20 节; 其余 scale/mean/inner product 以及并行问题也请参考 user guide;

有一点需要特别注意:FieldConvert 有一个计算边界层厚度的功能, 那个功能是针对 3d 情况计算网格某边界附近的棱柱层厚度的, 与实际意义上的边界层可能有一些差别;

2.4 XML 文件说明

首先,xml 文件的结构大致是这样的:

```
1 <NEKTAR>
2         <GEOMETRY>
3          ...
4         </GEOMETRY>
5         <EXPANSIONS>
6          ...
```

```
    </EXPANSIONS>
    <CONDITIONS>
    ...
    </CONDITIONS>
    ...
</NEKTAR>
```

具体的细节请参考 Nektar++ 自带的测试/范例 xml 文档. 此处先讲一个 xml 语言的小技巧 (注释):

对于一段内容若想要节省修改时间, 增大可修改性, 可以把之前写好的不用的属性注释掉, 这样就不会运行注释内的内容; 具体的规则是 <!– XX> XXXXXX </XX–>, 同下例:

```
<!-- P> Flowrate = 0.05 </P -->
```

2.4.1 网格部分

这部分主要记录网格的几何信息 (其实也可以和下面程序部分合写到同一个 xml 文档中); 一般而言, 通过 NekMesh 生成的 xml 网格信息, 会包含 Geometry 和 Expansion 两大部分, 下面展开来讲 (可能会用到的信息会有提到, 全面的信息请移步 user guide 第 3 章):

几何信息这部分主要由 7 部分组成, 但 xml 几何信息往往只有 6 部分 (没有 Curved 部分):

```
<GEOMETRY DIM="2" SPACE="2">
    <VERTEX> ... </VERTEX>
    <EDGE> ... </EDGE>
    <FACE> ... </FACE>
    <ELEMENT> ... </ELEMENT>
    <CURVED> ... </CURVED>
    <COMPOSITE> ... </COMPOSITE>
    <DOMAIN> ... </DOMAIN>
</GEOMETRY>
```

其中 **DIM** 是有限元展开的维数,**space** 指示有限元维数; 下面将对其他部分分开进行讲解;

2.4.1.1 几何 (Geometry)

- 点 (Vertice)

一个” 点” 的信息主要由坐标和 ID 构成;ID 为点的编号, 坐标指点的三维笛卡尔坐标; 格式如下:

```
<VERTEX>
        <V ID="0"> 0.0 0.0 0.0 </V>
        ...
</VERTEX>
```

描述各点的信息应该在 <VERTEX> 根下, 下同 (以下省略);

另外,Nektar++ 的几何信息存储有一个优势: 只有点是通过坐标定义的, 只要改变点的位置, 边的连接关系等其余信息均不会改变; 此处的点也可以通过 Nektar++ 自带的功能改写参数实现坐标放缩和移动, 也可以通过 Python 脚本实现 (见3.5.1节挤出后的放缩操作);

Nektar++ 自带的功能比较有限, 可以考虑用 Python 脚本自动化; 其中, 坐标放缩参数为 **XSCALE**,**YSCALE** 和 **ZSCALE**, 可对点的某一轴坐标以” 原点”(也就是该轴对应的”0”) 为中心进行放缩 (scale), 不过只支持固定比例的放缩, 不支持设置 x,y,z 的变量操作; 也可以有参数, 但是参数必须有事先定义 (Parameter:pre-build); 另一个功能是平移 (translate), 参数配置为 **XMOVE**,**YMOVE** 和 **ZMOVE**; 下面是示例:

```
<VERTEX XSCALE="5",ZSCALE="2",XMOVE="-1",YMOVE="0.2">
        <V ID="0"> 0.0 0.0 0.0 </V>
        <V ID="1"> 1.0 2.0 0.0 </V>
</VERTEX>
```

- 边 (Edge)

结构同上; 一般为:

```
<E ID="0"> 0 1 </E>
```

- 面 (Face)

面一般有两种: 三角形 (Triangle) 和四边形 (Quadrilateral); 一般就 2D 网格而言,Gmsh 会优先生成四边形网格, 如果因为顶点问题无法再继续生成四边形网格才会生成三角形网格;

```
<T ID="0"> 0 1 2 </T>
<Q ID="1"> 3 4 5 6 </Q>
```

- 有限元 (Element)

有限元的种类比较丰富:

S	Segment	1 维: 线段分割	A	Tetrahedron	3 维: 四面体形
T	Triangle	2 维: 三角形	P	Pyramid	3 维: 金字塔形 (四棱锥)
Q	Quadrilateral	2 维: 四边形	R	Prism	3 维: 三棱柱
			H	Hexahedron	3 维: 六面体

- 曲线 (Curved)

对已设定曲线进行单独控制, 详见 user guide.

- 构成 (Composite)

主要是 Gmsh 设定中的 Physical Curve/Surface/Volume; 如下, 最前面的 ID 与 Gmsh 中设定的各物理线/面/体的标签相同, 后方所接的必须是同种类型的" 元素"(比如, 不能点线混写); 这部分在本册2.3.1.4节 NekMesh 的面元抽取部分也有提到;

```
1 <C ID="40"> T[0-862] </C>
2 <C ID="41"> E[61-67,69,70,72-74] </C>
```

- 计算域 (Domain)

规定展开计算的主要区域 (一般会有一个单独的编号); 格式:

```
1 <DOMAIN> C[2] </DOMAIN>
```

2.4.1.2 展开 (Expansions)

这部分主要规定上述构成部分的多项式展开形式, 主要包含对应组成部分 (**Composite**, 一般只计算区域), 多项式阶数 (**NumModes**), 场 (**Fields**) 以及展开形式 (**Type**) 四部分 (注: 为易于理解才区分了大小写, 在程序中这些参数名需要大写,Nektar++ 对大小写有比较严格的限制, 不要随意改动); 形式如下:

```
<E COMPOSITE="C[0]" NUMMODES="5"
                    FIELDS="u,v,p" TYPE="MODIFIED" />
```

以上是比较简要的形式, 其中, 有几点需要注意:

- 1. 对于不同计算区域可以用不同的方法, 不过需要在设计网格时就定义好;
- 2. 多项式阶数默认为 4; 每个有限元中间各边的分割" 格数" 为多项式阶数减一 (与插值多项式原理相同);

- 3. 场部分定义的" 变量" 场一定要与后边程序部分定义的一致; 一般初始生成只会有"u" 这个场, 其他场需要自己加上去; 除了速度场和压力场, 在 MHD 问题中还会有电势场 (Nektar++ 同样支持);

如果没有改为对应场的话, 在运行时, 程序会使用默认定义来定义场, 而且会在 5.0.x 版本日志中提醒以下内容 (不过一般不影响程序运行):

```
Fatal: Level 0 assertion violation
Use default defination of variable: p
```

但是, 若在 5.2.0 及以上版本, 程序可能会直接无法运行:

```
Fatal: Level 0 assertion violation
Variable 'p' is missing in FIELDS attribute of EXTENSION tag.
```

- 4. 类型一般指插值类型; 一般根据求解器的不同, 支持性也有一定不同; 在常用的 IncNavierStokesSolver 里, 支持 Modified,GLL_Largange,GLL_Largerange SEM 几种, 有兴趣可以通过搜索 GLL 快速找到该位置;Modified 一般用的修正过的 GLL 点插值, 低阶数的时候 (如 4 阶), 肉眼观察几乎是均匀的.

当然, 也可以展开写: 具体而言, 对于不同的方向, 甚至可以使用不同的基 (basis),NUMMODES,TYPE 等参数, 可选的参数范围也有增加 (不过一般不怎么会用到).

```
<E COMPOSITE="C[1]"
    BASISTYPE="Modified_A,Modified_A,Modified_B"
    NUMMODES="3,3,3"
    POINTSTYPE="GaussLobattoLegendre,GaussLobattoLegendre,
                                        GaussRadauMAlpha1Beta0"
    NUMPOINTS="4,4,3"
    FIELDS="u" />
```

如以上部分, 在 x 和 y 方向使用了相同的 TYPE 和插值点 (在规定基的时候便做了区分);

2.4.2 程序部分

这部分细项比较多, 而且根据求解器不同会有很多差异 (比如: 特有的一些选项), 故此处不详细介绍, 会用到的地方已在上一节2.3.2处讲过了;

2.4.2.1 条件设定 (Conditions)

- 参数 (Parameters)

主要是设定预制 (pre-built) 参数, 格式如下例:

```
<P> NumSteps = 1000 </P>
<P> TimeStep = 0.01 </P>
<P> IO_CheckSteps = 1000 </P>
<P> IO_InfoSteps = 200 </P>
<P> Kinvis = 0.000001 </P>
```

其中 **NumSteps** 是总步数,**TimeStep** 是时间步长,**IO_CheckSteps** 是输出流场的步频,**IO_InfoSteps** 是输出日志的步频;

- 求解器信息 (Solver Info)

确定求解格式的一些相关信息, 比如: 方程类型 (EQTYPE), 投影法连续性 (Projection), 时间推进格式等等;

```
<SOLVERINFO>
  <I PROPERTY="EQTYPE" VALUE="UnsteadyAdvection" />
  <I PROPERTY="Projection" VALUE="Continuous" />
  <I PROPERTY="TimeIntegrationMethod" VALUE="
                                    ClassicalRungeKutta4" />
</SOLVERINFO>
```

格局求解器的不同这部分也会有差异 (比如方程类型, 只针对求解器, 不同求解器的 eqtype 完全不同, 具体参数的设置方法会在 user guide 对应求解器的章节有专门的表格列出可选类型; 其他参数同理);

- 流程驱动 (Driver)

基于以上求解器信息, 进一步规定的求解流程, 默认是标准 (**Standard**); 这部分主要与稳定性分析有关 (ModifiedArnoldi, Arpack), 有兴趣可自行了解 (user guide 3.3.2.1);

- 变量 (Varieables)

规定了计算中涉及到的场的信息; 场可以是速度场, 压力场, 电势场等; 此处的变量 (Variables) 代表对应的场 (对应于 EXPANSION 部分的 FIELDS); 其中, 速度场一般规定数量等于计算问题的空间维数, 比如三维问题规定"$u(x)$,$v(y)$,$w(z)$" 三个变量, 而二维问题规定"$u(x)$,$v(y)$" 两个; 格式为:

```
<VARIABLES>
  <V ID="0"> u </V>
  <V ID="1"> v </V>
</VARIABLES>
```

- 边界区域与边界条件 (Boundary Regions and Conditions)

设定边界这里比较麻烦; 必须先定义边界区域才能在下边给出边界条件; 为直观了解各部分结构, 先给出一个边界区域的例子 (下例中的 C[?] 内部的编号见本册第2.4.1.1节对应部分):

```
<BOUNDARYREGIONS>
  <B ID="4"> C[41] </B>
  <B ID="5"> C[42] </B>
</BOUNDARYREGIONS>
```

其中"41","42" 是画网格时设计好的 Physical 区域 (若忘记, 请参考网格的 geo 文件);"0","1" 是定义的边界区域(不是" 类型"! 每个"Composite" 应对应不同"Region"(即便它们的边界条件完全相同), 下方的边界条件处与 Region 的 ID 对应; 可以理解为,Region ID 是边界标签的一次信息中转, 是 xml 到 solver 的一个信息接口); 于是, 可以在下方继续定义边界条件:

```
<BOUNDARYCONDITIONS>
  <REGION REF="4">
    <D VAR="u" VALUE="sin(PI*x)*cos(PI*y)" />
    <D VAR="v" USERDEFINEDTYPE="TimeDependent"
                VALUE="sin(PI*x)+0.05*sqrt(t)*awgn(1)" />
    <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" >
  </REGION>
</BOUNDARYCONDITIONS>
```

上边是一个随便写的边界条件, 对应于原始定义的"41" 面/线 (本册中上文给出的信息为"41" 面); 其中 x、y 两个方向的速度均为 Dirichilet 型边界，其值用以上函数表示；“y” 轴方向边界速度为一个时间相关的函数，awgn(1) 生成一个方差为 1 的 Gauss 白噪声; p 的设置为速度修正格式 (VCS) 自然边界; 具体的边界问题在本册2.5.4节部分会讲到;

边界条件的类型分为 3 大类, 其中根据加载条件和时间依赖性等 (类型) 又细分为很多小类, 甚至有一部分边界类型 (UserDefinedType) 和求解器有关; 以下主要讲解边界的" 大类";

第一种是 Dirichilet(狄利克雷) 边界条件, 又称为本质 (essential) 边界条件 (第一类边界条件); 在文档中用 <D> 来标识, 同下:(依次是齐次/非齐次/时间相关边界)

```
<!-- homogeneous condition -->
<D VAR="u" VALUE="0" />
<!-- inhomogeneous condition -->
```

```
<D VAR="u" VALUE="x^2+y^2+z^2" />
<!-- time-dependent condition -->
<D VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="x+t" />
```

第二种是 Neumann/von Neumann(诺伊曼) 边界条件, 又称为自然 (natual) 边界条件; 在文档中用 <N> 来标识 (在程序中 N 型边界不支持离散 Galerkin 投影, 对混合 Garlerkin 法的适应性未知), 同下:(依次是齐次/非齐次/时间相关边界/VCS 高阶压力边界)

```
<!-- homogeneous condition -->
<N VAR="u" VALUE="0" />
<!-- inhomogeneous condition -->
<N VAR="u" VALUE="x^2+y^2+z^2" />
<!-- time-dependent condition -->
<N VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="x+t" />
<!-- high-order pressure boundary condition (for
                                    IncNavierStokesSolver) -->
<N VAR="u" USERDEFINEDTYPE="H" VALUE="0" />
```

第三种是周期边界条件, 用 **P** 进行标识; 只支持连续 Garlerkin 投影; 一般对于出入面的取值, 用面 ID 进行赋值即可, 如下例; 另外也可以通过旋转进行一些其他操作 (详见 user guide 3.3.5.3 节);

```
Example:
<BOUNDARYREGIONS>
   <B ID="0"> C[1] </B>
   <B ID="1"> C[2] </B>
</BOUNDARYREGIONS>

<BOUNDARYCONDITIONS>
   <REGION REF="0">
         <P VAR="u" VALUE="[1]" />
   </REGION>

   <REGION REF="1">
         <P VAR="u" VALUE="[0]" />
   </REGION>
</BOUNDARYCONDITIONS>
```

提一下时间相关的设定; 若在边界上设置一个与时间相关的函数, 需要改为 (USERDEFINDETYPE=”TimeDependent”), 时间用参数 **t** 进行表示即可; 另外, 边界也可以选择从文件中加载 (FILE=”xxx”);

- 函数 (Function)

只讲几个常用到的:

第一个是初始条件 (初始流场):

```
<FUNCTION NAME="InitialConditions">
  <F VAR="u" FILE="session.fld" />
  <E VAR="v" VALUE="2">
</FUNCTION>
```

以上是一个初始条件,”u” 从”session.fld” 流场读取,”v” 用 2 进行赋值 (注意从文件读取的 <F> 标签);

第二个是体积力 (等) 条件:

```
<FUNCTION NAME="BodyForce11">
  <E VAR="u" VALUE="0" />
  <E VAR="v" VALUE="0" />
</FUNCTION>
```

第三个是流量条件 (其他需求请参考求解器部分);

```
<FUNCTION NAME="FlowrateForce">
  <E VAR="ForceX" VALUE="0.0"/>
  <E VAR="ForceY" VALUE="0.0"/>
  <E VAR="ForceZ" VALUE="1.0"/>
</FUNCTION>
```

因为比较重要,Quasi-3d 部分见第2.5.6节, 会重点讲一下;

2.4.2.2 滤波器 (Filters)

用到的也比减少, 可能会用到的在如下列出:

1. FieldConvert Checkpoints, 在运行求解器的同时耦合 FieldConvert 生成可视化文件; 下例:

```
<!-- FILTERS>
<FILTER TYPE="FieldConvert">
  <PARAM NAME="OutputFile"> timevalue </PARAM>
```

```
    <PARAM NAME="OutputFrequency"> 10 </PARAM>
    <PARAM NAME="Module"> -r -1,1,-0.5,-0.5,0,2 </PARAM>
</FILTER>
```

FieldConvert 相关参数见本册2.3.3章节;

注: 这里的 <FILTER> 有一点需要特别注意, 那就是,上述 FieldConvert 在并行运行求解器的时候顺带只能生成 pvtu 型的并行存储格式的文件, 而不是 vtu 格式, 有可能会造成存储爆炸 (膨胀 600+ 倍也是完全有可能的);

2.历史点 (History Points) 容易被忽略掉的一个功能, 其实相当有用 (问就是有笨蛋尝试用 FieldConvert 的-r 功能生成有限元块再去插值了);

```
<FILTER TYPE="HistoryPoints">
    <PARAM NAME="OutputFile">TimeValues</PARAM>
    <PARAM NAME="OutputFrequency">10</PARAM>
    <PARAM NAME="Points">
        1 0.5 0
        2 0.5 0
        3 0.5 0
    </PARAM>
</FILTER>
```

以上内容记录 (1,0.5,0) 等三个点不同时刻的信息; 每 10 步输出一次;

2.4.2.3 耦合 (Coupling)

主要涉及求解器耦合的问题, 目前还有待解读 (没怎么用到, 而且也没有安装 CWiKi); 详情见 user guide 第 3.6 节.

2.4.2.4 力 (Forces)

规定力的类型, 确定力的表达函数名; 以以下体积力为例:

```
<FORCE TYPE="Body">
    <BODYFORCE> BodyForce11 </BODYFORCE>
</FORCE>
```

以上内容确定了以上2.4.2.1部分 BodyForce11 函数作为体积力而出现;

然后是噪声 (扰动); 不过这里的噪声只支持加全场噪声;

```
<FORCE TYPE="Noise">
    <WHITENOISE> [VALUE] <WHITENOISE/>
    <!-- Optional arguments -->
```

```
    <UPDATEFREQ> [VALUE] <UPDATEFREQ/>
    <NSTEPS> [VALUE] <NSTEPS/>
</FORCE>
```

几个参数从上到下依次为噪声大小, 更新频率 (重算噪声, 默认初始计算一次), 重新添加噪声 (步数, 默认全程);

2.4.3 编写表述

其实只是一些注意事项, 具体内容可见 user guide 第 3.7 节;

- 1. 不论是几维问题, 都要使用三维坐标表示点的位置.
- 2. 表达式中常见常数 (无理数等), 见 guide 第 3.7.1.2 节.
- 3. 加减乘除幂等常见操作及一些常见函数略.
- 4.awgn(p) 生成以 p 为方差的 Gauss 分布的随机数 (用来加噪声).
- 5. 表达式乘一个 boolean value 用来给函数分段, 如 $\times(z < 1)$
- 6. 表达式与计算速度部分需要看一看

2.5 经验之谈

2.5.1 流体基本概念与程序对应

一般而言, 在程序中设定 x 轴速度为 u,y 轴速度为 v,z 轴速度为 w; 哪一个轴作为流向轴可以进行人为规定; 方向有三个, 一般在讨论问题时更喜欢用" 流向 (Streamwise)、法向 (Normal)、展向 (Spinwise)" 速度 (Velocity) 这几个概念, 而不是"u,v,w"; 下面也会有一点点涉及;

2.5.2 归一化处理

时间步长:Nektar++ 不支持自适应步长, 一定要自行设置; 但是以下提到的 CFL 准则却允许设置子步 Courant 数 (#SubStepCFL). 另外 (若将 Driver 设定为 Adaptive 则可以自适应选择网格各区域的多项式阶数);

Courant-Friedrichs-Lewy(CFL) 条件是数值解决偏微分方程问题中的一个重要概念. 这是一种为了保证数值方法的稳定性，对于时间步长和空间步长设定的一种限制. CFL 条件是由 Richard Courant、Kurt Otto Friedrichs 和 Hans Lewy 在 1928 年首次提出，主要用于解决对流方程和波动方程等问题.CFL 条件通常表述为：

$$C = \frac{U\Delta t}{\Delta x} \leq 1$$

其中，C 被称为 Courant 数，U 是流体的特征速度（例如波速或流速），Δt 是时间步长，Δx 是空间步长. CFL 条件的物理意义是：在一个时间步长内，物质或信号不应该传播超过一个空间步长. 简单来说，信息在一个时间步进内不应该跳过一个格点. 这就保证了在数值模拟中，信息的传递是局部的，从而使得数值解有良好的稳定性。如果 CFL 条件不满足，通常会导致数值解的震荡和发散.

在一些数值方法中，CFL 条件需要更严格，比如在某些高阶方法中，Courant 数需要小于 1. 在实际应用中，也可能需要调整 Courant 数，以找到一个平衡点，既保证了计算的稳定性，又保持了相对较高的计算效率.

而在设计程序的时间步长时也常常需要用到归一化处理的单位时间, 所以就一起放到这里来讲了.

Kinvis(动力学粘度 ν) 一般取 1×10^{-6},Re 可以人为规定, 也可以反推 (若不在函数中出现则不参与" 隐性" 计算);

时间步数也可以通过速度等信息归一化处理, 不过认为麻烦的话, 也不一定要做 (整千整百的步数相对容易控制些);

2.5.3 插值与网格疏密

主要就是遇到计算量超大的情况可以用稀疏一点的网格算一下试试水,get 基本流之后用 FieldConvert 插值 (interpfld) 得到密一点的流场, 再用密网格及其流场算后续内容; 但是!!!Quasi-3d 的 Homogenous 方向不行, 请绕路用 (3DH1D 专用模式 5.5.15)!

2.5.4 边界条件

主要介绍三类: 出流边界条件, 入流边界条件和无滑移边界条件;

流出条件的速度场均为 Neumann 条件 (为 0), 压力场为 Dirichilet 条件 (为 0); 虽然不太符合物理规律, 但是计算软件很多是这样设置的, 没什么影响; 由于 Dirichilet 条件是对场本身而言的取值,Neumann 则是对于梯度的取值, 流出的边界可以概括为: 速度场无变化, 边界无流出压力;

入流条件的速度场可设置为 D 型, 规定速度型 (Velocity Profile), 压力一般是给一个恒定的压力梯度 (N 型, 可以为 0);

无滑移边界主要指管道的管壁这种类型的壁面, 固定, 有实体, 不与流体发生相对移动 (发生相对移动的部分属于 MovingBody,user guide 有专门讲这个的地方); 这种边界一般取贴体速度为 0(D 型), 压力梯度为 0(N 型);

2.5.5 边界扰动

边界上的扰动, 扰动幅值最好要统一; 比如说一个 2 维方形无物理边界的流场, 入口直线处的速度依线性分布 (流向中间最大速度 $2u_0$, 两侧速度 u_0, 其余部分速度线性

变化, 假设噪声强度设置为 5%), 优先考虑用平均速度的 5% 进行扰动, 而不是依当地速度的 5% 进行添加; 详细添加细节在跑流场前需要咨询老师添加方式;

另一种是额外添加一个体积力来获得” 噪声”, 目的是给流场足够的诱导从而发生转捩 (transition).

2.5.6 准 3d 配置 (Quasi-3D)

对应于 user guide 的第 3.3.7 节, 主要讲用 FFT 将低维网格扩展到 3 维进行计算的情况 (使用低维网格运行程序即可);

为使用 Quasi-3d 方式, 必须在 Solver Info 部分添加 FFTW 的使用及齐次拉伸的使用:

```
<I PROPERTY="USEFFT" VALUE="FFTW" />
<I PROPERTY="HOMOGENEOUS" VALUE="1D" />
```

在使用前, 需要明确一点: 拉伸方向默认为展向 (2D 到 3D, 参数为 1D; 当然, 也有流向拉伸的); 需要设定的参数有两个:”HomModes?” 和”L?”, 其中”?” 可以是 X,Y 或 Z(但二者需要一致); ”HomModes?” 规定了在”?” 齐次方向进行拉伸的” 波数”;”L?” 规定了其次方向的拉伸长度;

```
<PARAMETERS>
<P> HomModesY = 4 </P>
<P> LY = 1.0 </P>
</PARAMETERS>
```

以上规定了一个 xz 平面在 y 方向上拉伸距离为 1, 展向波数为 4 的部分流场配置;

在运行求解器时, 因为使用了 FFTW, 在齐次方向使用并行会大大提高运行效率, 为此:

```
mpirun -n 128 -N 2 ./INSS --npy 32 Prog.xml Mesh.xml
```

同理, 若 z 方向为齐次展开方向, 上方应为”–npz”, 此参数为其次方向的并行任务数;

另外, 关于此并行数还有一定要求:

- 1. 此数必须为偶数 (from: FFT), 且为总任务数的因子;
- 2. 在取值上, 根据服务器的单节点 CPU 数计算, 此参数在约为单节点 CPU 数一半时效率相对较高;

2.5.7 脚本与并行: CPU 杂谈

本地跑程序有时也会用到脚本 (一般主要是在想要多线程处理数据的时候会用到), 写一个 For 循环基本就能解决问题了;

本地处理多组数据时会遇到上述情况; 比如, 现有 500 组流场数据, 要将其可视化处理, 那么为提高效率就需要并行处理; FieldConvert 对 vtu 数据的生成可以调用用 mpi 分组生成单组数据, 但生成 Tecplot 式的 dat 数据却是只能单线程处理单组数据, 这时便是以上提到的使用循环的时机;

一般而言, 单个 CPU 默认执行一个任务, 视 CPU 型号而定, 一般是 2 个 (比如, 在买电脑的时候会看参数 8CPU-16 线程, 就是每 CPU 最多同时处理两线程任务), 也就是单 CPU 会有两个逻辑内核; 具体来说, 一个分到 4 个 CPU 的虚拟机, 在用 mpirun 执行 FieldConvert 命令生成 vtu 文件时,-n 参数最大可以设置为 8, 而超过 8 就会报错; 当然, 在服务器上也可以指定:

```
#SBATCH --ntasks-per-cpu=2
```

来实现每核跑 2 任务, 不过没什么意义就是了;

另外, 也要区分开" 进程" 与" 线程"; 总体而言," 进程" 指的是任务本身, 而" 线程" 指的是完成任务的最小单元; 比如, 把" 吃饭" 看作一个" 进程", 那么这样一个进程可以细分为" 吃米饭" 和" 吃菜" 两个线程, 既可以同时处理 (并行), 也可以分开进行 (串行, 一般串行不称线程);

2.5.8 检查

运行程序前一定要从 xml 程序配置文件的最初开始检查, 检查推进格式等选项以及各种条件与参数, 下面列一些可能会出错且非常难找来的一些错误 (血泪教训):

- 各类参数是否小数位数有误
- 各类条件标签是否都是大写 (比如边界条件的 <D>)
- 边界和初值条件等涉及函数类赋值的地方是否缺少或多加系数
- 边界条件类型是否搞错
- 时变型边界 (尤其在边界加噪声时)"TimeDependent"type 是否有设置
- 网格参数是否都正确 (若后续有变换, 变换系数是否正确)

2.5.9 动手/善用算例

练习使用可以用简单的算例来练手, 熟悉程序的使用; 每个 solver 下都有对应的算例, 可以运行或者参考; 当然, 有些部分还有单独的 Tutorial, 可以尝试自己做一下, 对照给出的 Compelete 中的结果;

Chapter 3

Gmsh 使用指南

3.1 软件安装

建议安装 Linux 版本 (Windows 版与 Nektar++ 耦合不太方便);

如果想要省些功夫, 或者安装较低版本的 Gmsh 亦能够满足需求, 则命令行运行以下代码即可

```
sudo apt-get install gmsh
```

不过, 建议安装较高版本的 gmsh, 对现用及更新后的 Nektar 软件相对友好; 以下主要介绍一下较高版本的 Gmsh 安装;

可选用的方式有两种, 一种是直接下载免安装的版本;

另一种是下载源码包, 在本地电脑编译安装, 此处简单介绍一下.

第一步, 从官网下载地址下载安装包到 Linux 本地 (或者直接用现成的 4.11.0 版源码包), 再选好文件夹进行解压 (利用归档管理器或者从终端打开执行 tar -zxvf 命令解压);

第二步, 打开解压后的文件夹, 右键选择打开终端, 输入以下命令

```
cmake -DENABLE_BUILD_DYNAMIC=1 ..
make
make install
```

以上过程中会用到 cmake 工具, 这个在安装 Nektar++ 时已经安装过了; 等待安装好就可以了; 详细的安装细节也可以参考压缩包下的 Readme 文件;

3.2 网格编写逻辑

网格的质量对模拟结果的准确性有很大影响. 以下是评价有限元网格好坏的一些常见标准：

1.单元形状：理想情况下，网格单元应接近规则形状，例如，对于二维问题，三角形应尽可能接近等边三角形，四边形应尽可能接近正方形或长方形。对于三维问题，四面体和六面体的形状也有类似要求. 五面体过渡网格尽可能少.

2.单元尺度：单元的尺度应该与物理问题的特征尺度相匹配。对于变化剧烈的区域，如边界层，应使用较小的单元。单元应该适当地进行加密和放大.

3.单元的扭曲度：扭曲的单元可能会导致低效和不准确的结果。应尽量避免高度扭曲的单元.

4.过渡区的单元形状：在网格精细区与粗糙区之间过渡时，过渡应尽可能平滑，以避免大的单元与小的单元直接相接，防止导致解算稳定性和准确性下降. 必要的时候为尽可能平滑, 可以适度调整网格, 做一些扭曲.

5.是否满足问题的对称性：如果问题具有几何或物理对称性，最好使用网格来尊重这种对称性，以减少计算量并提高计算效率.

一般而言, 在设计网格的时候结构化网格是有优先度更高的 (在要求计算精确度的前提下), 而非结构化网格的更适用于对计算精度要求相对不太高的区域, 此时用非结构化网格也能比较方便地调整全区域的网格密度;

3.3 网格编写结构

3.4 其他功能

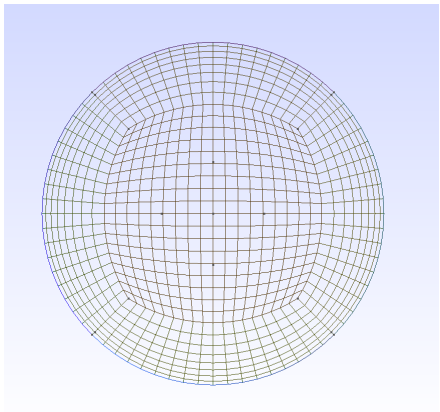
主要参考使用手册 (官网与使用手册不太稳定, 有时需要科学上网), 此处留一下user guide 的网盘备份.

3.5 经验之谈

3.5.1 挤出 (Extrude) 之同源放缩

为难自己和老师近半个月的问题; 尝试了各种办法, 却还是很难得到解决; 结论是: 这件事在 gmsh 是几乎没办法解决的一件事;

故事的起因是老师这边需要做一个变半径的圆管 (圆管的半径是根据某一函数变化的), 因为对圆管内部的各部分解析度有要求, 这就限定了圆管截面的网格结构 (如下);



下面先来谈一谈失败的几种方法;

1. 这个结构比较特殊, 一个圆面其实是由若干个面组成的, 所以若想要进行挤出, 就需要随时根据函数调整挤出方向, 但是因为不能放缩, 且挤出反向难以确定, 否掉这一种方案;

2. gmsh 的挤出只有 4 种命令, 可以规定平移, 旋转, 边界层, 但是对按函数比例放缩却有所限制, 否决;

3. 可以通过旋转来挤出这样的网格, 不过必须注意, 通过旋转的得到的网格为四面体网格, 而且外部轮廓为 Spline, 与实际可能有一定差别, 不符合要求, 否决; 特别注意，gmsh 不支持函数化的曲线输入;

4. 通过 gmsh 的循环命令, 先指定下一层的点标签, 再逐层挤出 (Extrude), 但 gmsh 对循环有一定限制, 失败;

5. 更换软件;gmsh 支持使用其他软件进行 CAD 建模 (几何图形), 再回到 gmsh 生成网格, 具体格式为 (geo/iges/step/brep), 因为不同软件对挤出并放缩的支持度不尽相同, 且换大型软件学习时间成本过高, 否决;

6. 利用 Nektar++ 的 3DH1D(Quasi-3d) 对网格进行变换也是一个思路, 但是并没有这样的功能;

7. 利用 Nektar++ 的 XML 文件书写对坐标进行变换: 可行, 但是对于各点, 只能以原点作为中心进行放缩, 不支持随时更换原点位置, 失败; 值得一提的是Nektar++ 里可以对各点进行拉伸 (scale) 和平移 (translate) 操作, 变换的值也可以用函数进行表示, 但是里边的参数必须事先有定义 (可以参考 user-guide).

8. 可以考虑利用 gmsh 的 Python-API 对 msh 格式的网格中的点逐个进行操作, 但是如果这样, 必须要对 msh 的文件存储形式足够熟悉才相对方便 (格式可参考 gmsh 文档);

从而, 最后一种比较可行的办法 (老师想出来的)，可以先挤出生成直圆管, 然后对用 Python 对 xml 文件中的点坐标进行变换操作; 下边放一段 Python 代码 (因为学 xml 包的操作也比较麻烦, 所以直接让 GPT-4 代劳了, 亲测可用); 另外, 附带一段修

改 Expansions 部分的代码 (主要指场和多项式阶数).

```
import xml.etree.ElementTree as ET
import math

tree = ET.parse('pipeMesh.xml')  # XML文件路径
root = tree.getroot()

for vertex in root.iter('VERTEX'): # 遍历"VERTEX"
        for v in vertex.iter('V'):  # 遍历"V"
                coords = v.text.split() # 获取坐标并分割成列表
                x, y, z = map(float, coords)  # String2Float
                scale = 2 + math.cos(z)  # 计算放大倍数
                x *= scale  # 放大x
                y *= scale  # 放大y
                v.text = f'{x} {y} {z}'  # 更新坐标

for exp in root.iter('EXPANSIONS'):
        for e in exp.findall('E'): # 修改元素属性
                e.attrib['NUMMODES'] = '5'
                e.attrib['TYPE'] = 'GLL_LAGRANGE'
                e.attrib['FIELDS'] = 'u,v,w,p'

tree.write('New_Mesh.xml')  # 将修改后的XML保存为新文件
```

如果是在服务器上跑 Python 程序的话, 还有额外的注意事项;

- 因为服务器在挂程序的的时候不允许不通过 Slurm 系统提交 (一旦提交会有记录并有警告, 三次同类操作会被封号), 所以需要再写一个脚本用于提交 Python 程序 (命令: python xxx.py)

- 由于以上处理好的数据是单个文件, 而且做并行化处理需要对 Python 程序进行并行化改造, 上述 python 程序实际上是单任务单 CPU 完成的; 所以写提交用的脚本时要注意参数; 另外, 需要注意,Python 模块需要额外加载环境;

以上代码对 NekMesh 处理过的 xml 进行操作, 实现不等比例变换; 将圆管变换前后的网格对比:

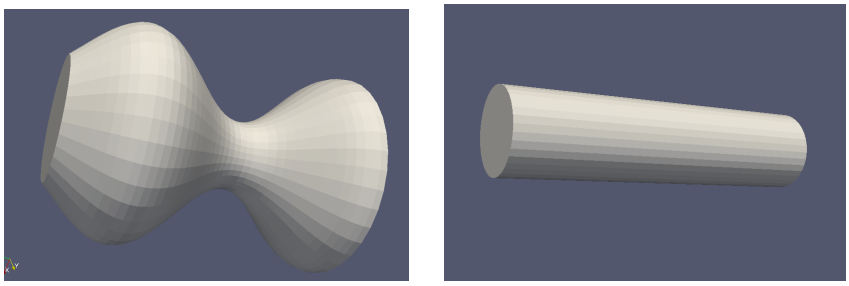


图 3.1: 变化前后的圆管 (素体)

3.5.2 内点与单元

起因在于因为长期使用 Paraview 造成的一点小乌龙; 这里主要是再重申一下概念; 对有限元不够了解可以找本有限元的书翻一下前边的概念和基本原理;

首先, 直观来说, 用 Gmsh 进行网格绘制,其最小的网格单元即为有限元, 例如三角形, 四边形 (2 维), 四面体, 五面体 (四六过渡), 六面体 (3 维); 以二维的一个正方形区域来说:

```
1  point(1) = {0, 0, 0, 1};
2  point(1) = {0, 1, 0, 1};
3  point(1) = {-1, 1, 0, 1};
4  point(1) = {-1, 0, 0, 1};
5
6  Line(1) = {1, 2};
7  Line(1) = {2, 3};
8  Line(1) = {3, 4};
9  Line(1) = {4, 1};
10
11 Curve Loop(1) = {1, 2, 3, 4};
12 Surface(1) = {1};
13
14 Transfinite curve{1} = 5 using Progression 1.0;
```

以上网格会均匀生成 16 个正方形, 其中的每个小正方形都是一个有限元; 在 Nek Mesh 生成的 xml 格式网格文件的最下方, 会有 <EXPANSION> 条目, 其中的 NUMMODES 为插值多项式阶数;

Nektar++ 的多项式插值是针对有限元来说的; 例如, 在 1/16 的正方形内, 用 4 阶多项式进行插值, 在这个小正方形的各条边都会出现中间的 2 个 Gauss-Labatto 点, 对应连接后将 1/16 的小正方形分成 $3 \times 3 = 9$ 份, 此时的各点称为内点; 有限元的”

元” 也可称(有限元) 单元.

如果是六面体网格, 可以通过: 有限元单元数 ×(多项式阶数-1)^3+ 单元顶点个数, 估算整个网格的内点规模;

3.5.3 边界层与网格密度

在设计有实体壁面的网格时尤其要注意, 需要事先对贴体网格的壁面尺寸有一个估计; 具体而言就是根据边界层的 $y+$ 尺寸估计第一层的网格大小;

关于壁面函数与 $y+$, 其主题内容可参考专栏;$y+$ 表示壁面处的无量纲距离, 计算公式为 $y+ = \frac{y\sqrt{\tau_\omega/\rho}}{\nu}$,u 是边界层无量纲速度,y 是无量纲尺寸,$\nu$ 是运动学粘度;

实际应用中, 可以利用网上的计算器先估算一下 $y+$, 计算器网站传送门;

关于 $y+$ 的范围, 主要依据 $y+$ 的取值范围将边界层分为 3 层, 分别是: 粘性底层 ($0<y+<5$), 缓冲层 ($5<y+<30$), 完全湍流层 ($y+>30$); 网格第一层尺寸 (此处应考虑多项式阶数) 其实选择在 $15y+$ 或者 $30y+$ 以上均可, 不要过小;

Chapter 4

可视化软件

4.1 Paraview

4.1.1 Paraview 的安装

Paraview 在 Windows 和 Linux 都能安装使用; 因为服务器问题, 建议电脑本地和虚拟机都安装一下,下载地址; 至于 windows 就不详细介绍安装了;Linux 虚拟机上, 用 sudo apt-get install paraview 安装, 用库里相对旧一些的版本应该也没有问题;

4.1.2 Paraview 的基本操作

先说一下 Paraview 打开问题; 用 Paraview 打开 vtu/dat 文件时, 可以在命令行输入"Paraview xx.vtu" 命令打开, 也可以用"Paraview" 先呼出 GUI 界面, 然后在用 File»Open 打开 (Windows 只能通过这种方式);

程序成功加载后, 按左侧的"**Apply**" 键确认进行渲染; 渲染成功后界面上应该能够看到网格的素体轮廓 (图示见第3.5.1小节); 上方和"Apply" 下方都会有一栏可选菜单显示"Solid Color", 此处可以调换显示内容;

其他的操作和基本按键见下节;

做数据的曲线图,Paraview 功能不太行, 个人建议保存切片数据用 Matlab 或者 Python 画图 (当然, 如果会用 R 语言, 也可以用 R 作图);

4.1.2.1 基本按键与功能介绍

4.1.2.2 切片 (Slice) 制作

4.1.2.3 寻找数据 (Find Data)

4.1.2.4 作沿线的流速分布图

先介绍一个简单些的办法:

首先选中流场, 右键为其 Add Filter, 在 Alphabet 中找到 Plot Over Line 添加即可, 然后在左侧栏调节初始和终止点的位置, Resolution 是采样点的数量; 这个方式的优点在于可以直接用空间坐标而不是采样点标号作为横坐标, 但缺点在于没办法提取保存数据; 以下介绍另一种方式;

Paraview 取数据的能力比较奇怪: 它不太支持函数化操作: 取一条曲线根据某一坐标来绘制流速分布的图线; 以下以一个圆管的某一切片举例进行方法说明;

先介绍如何在一条直线上导出流速分布;

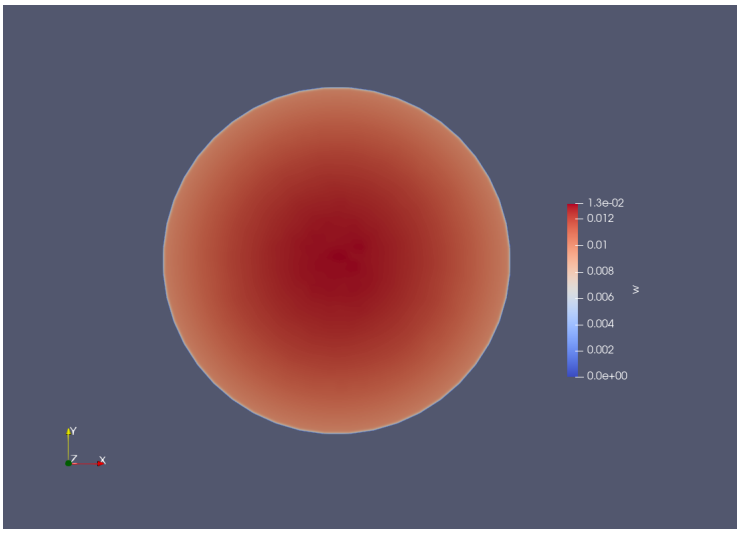


图 4.1: z 方向流速 (w) 切片云图

首先, 在如上的图4.1(xy 平面,$z = 3$, 上图为 z 方向速度的切片云图) 上先取一条直线 (直接创建直线段即可, 数据源是整个流场或者是包含直线段的这个切片都无所谓); 设置直线的时候, 需要设置直线的起点与终点, 还有直线上的采样点个数 (下方 Resolution) 三个信息, 从下图4.2的入口添加即可; 选中左侧栏的 Line, 在下方的信息处便能够对直线的属性进行修改, 修改完成点击"Apply"(图4.3(a)为修改的信息项, 图4.3(b)为直线图示);

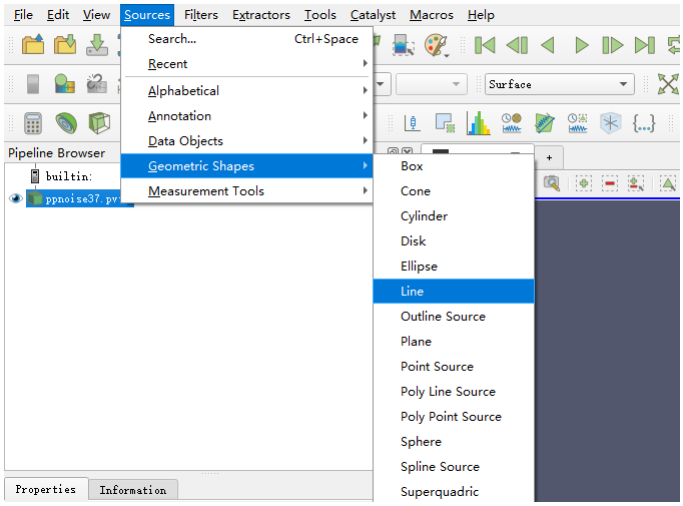
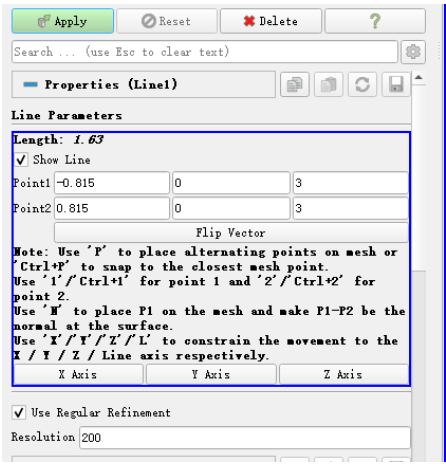
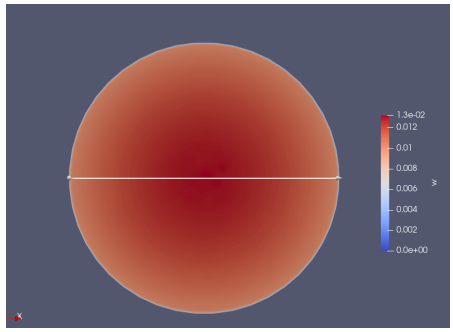


图 4.2: 线 (Line) 的添加



(a) 直线信息



(b) 直线图示

然后, 点击左侧的新建的 Line, 点击右键为其添加滤波器 (Filter), 路径:-> Alphabet -> Resample from Dataset, 如图4.3;

然后, 会出现一个对话框, 让用户选择数据来源和采样的导出对象; 其中的 Source data arrays 就选择最初的流场就可以 (如图4.4(a), 做好的切片也行),Destination mesh 选择此处需要完成重采样工作的 Line(如图4.4(b)), 再然后按 ok 键确认即可;

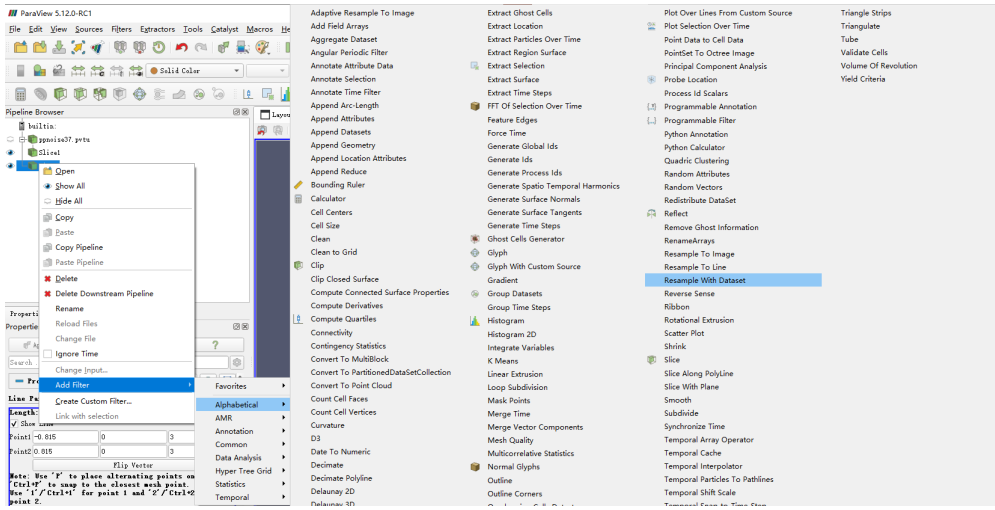
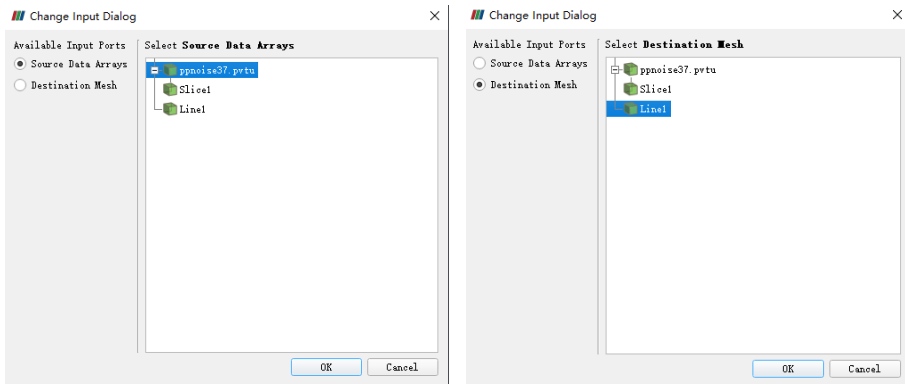


图 4.3: 对直线从数据集中采样



(a) Source data array (b) Destination mesh

在按下 ok 后, 左侧会出现 Resample 的新 Object, 选中按下边的 Apply, 并右键添加-> Alphabet -> Plot Data 滤波器 (如图4.4); 此后, 左侧会出现一个 plotdata 的 Object, 选中并将其可视化 (眼睛处) 即可得到直线上的速度分布图线 (在 File->save screenshot 处可以保存图线的截图);

然后是任一条线沿线的采样; 原理和上述过程基本相同, 在第一步 Source 部分添加 Geometry -> Poly Line Source 或者 Spline Source, 然后依次在下方输入线上点的坐标来生成一个 Spline 近似所求 (其余步骤均相同, 略);

另外,也可以利用 Python 对保存的数据进行绘图.

首先, 需要选中左侧边栏的ResampleWithDataset, 然后在 file 选项里找到save

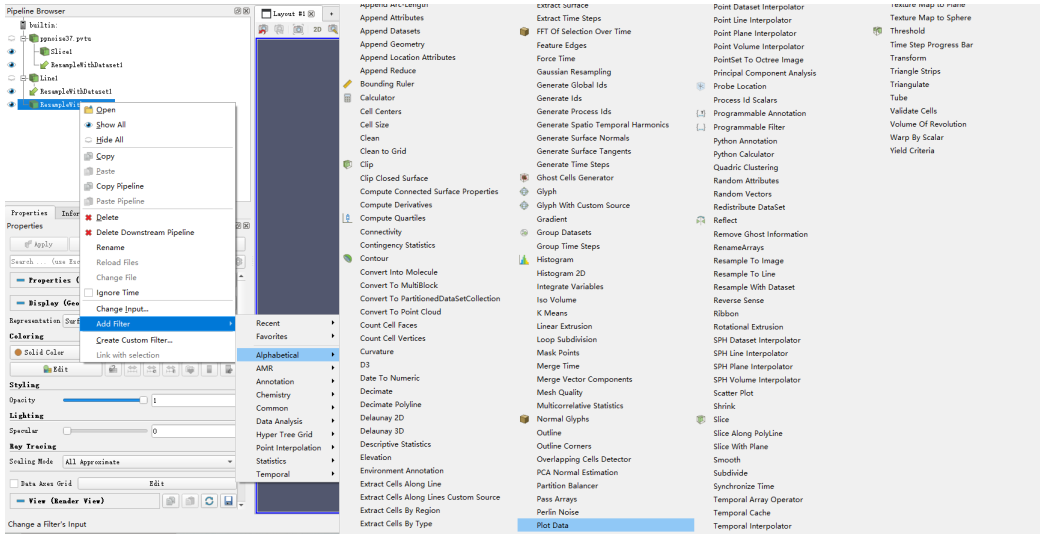


图 4.4: 绘制直线上的速度分布图线

data, 保存数据为 csv 文件, 选中 Point data, 并选择需要保存的数据内容, 利用 Python 进行画图的脚本此处也留做参考:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
data_base1 = pd.read_csv("data_1.csv", header=0, names
                                    =["u1","v1","w1","x","y","z"])
data_base2 = pd.read_csv("data_2.csv", header=0, names
                                    =["u2","v2","w2","x","y","z"])

plt.plot(data_base1["z"],data_base1["v1"], label="case1")
plt.plot(data_base2["z"],data_base2["v2"], label="case2",
                                                color="violet")
# plt.yscale('log')
plt.xlabel("z (streamwise)")
plt.ylabel("pulse velocity")
plt.legend()
# plt.ylabel("velocity fluctuation")

plt.show()
```

4.1.2.5 求截面的平均速度 (Q)

流量法; 此处的平均速度是用体平均法算出来的; 这个其实比较麻烦, 具体的操作方法与文件格式有关系;

先说 vtu 格式的文件; 还是以以上切片为例, 若想求其面平均速度, 就要对其添加 Integrate Variables 滤波器来实现 (Add Filter -> Alphabet); 在对其添加滤波器后, 可以在右侧的显示栏中看到如图界面,

4.1.2.6 Python Shell

4.1.3 Paraview 的远程部署

4.2 Tecplot

4.2.1 Tecplot 的安装

4.2.2 Tecplot 的基本操作

4.3 gnuplot

4.3.1 快速上手

从知乎摘的, 这是一段急速入门的万用代码:

```
1  ############  <100行代码急速入门Gnuplot数据绘图   ############
2  ### "#" 后边是注释
3  ### 首先设置输出的格式，支持pdf,png,eps等常用格式
4  set terminal pngcairo size 1000,1000 font 'Times New Roman,10'
   ## 格式，大小和字体
5  set output "plot.png" ###输出的文件名
6
7  #set terminal pdfcairo size 20cm,20cm font 'Times New Roman,12'
   ##
8  #set output "plot.pdf"
9
10 #set terminal epscairo size 20cm,20cm font 'Times New Roman,12'
   ##
11 #set output "plot.eps"
12
13 ### 可以定义变量和宏，便于后边重复使用
```

```
14 sx = "set xrange "  ## 例如后边当成宏来引用：@sx ，而不是使用
15                     ## "set xrange"，你可缩减代码量
16
17 ### 定义变量，用来设置上下左右的边缘和子图间距离
18 left=0.1
19 right=0.95
20 bottom=0.1
21 top="0.95"
22 hspace="0.1"
23 wspace="0.15"
24
25 ###因为是要一张图里4个子图，所以启用了多图模式：
26 set multiplot layout 2,2 spacing @hspace,@wspace margins left,
27                                                  right,bottom,@top
28
29 ########## 子图 (1): 绘制函数，设置基本的元素如：
30 ########## 标题、坐标范围、图例等
31 set label "(1)" at graph 0.02,0.03 font ',20' textcolor rgb 'red'
32 set title "example"
33 set xlabel "This is xlabel with {/Symbol a}=0.1 to 100"
34 set ylabel "This is ylabel with X^2_3"
35
36 set key top right Left reverse font 'Times New Roman,15'
37 ###设置图例格式：位置、字体等
38
39 f(x)=sin(x)/x          ###定义函数
40 set xrange [0:100]     ###设置x轴范围
41 set yrange [-0.5:1.0]  ###设置y轴范围
42 set xtics scale 3      ###设置x轴的刻度的长度，是默认的3倍长
43 set mxtics 10          ###x轴子刻度的数目
44 set mytics 5           ###y轴子刻度的数目
45 set log x              ###x轴设置成log
46 set style fill pattern 1
47 ### plot命令开始绘图并设置参数：
48 plot f(x) w line linetype 1 pointtype 5 ps 1.0 lc 2 lw 4,\
49 f(x) w filledcurves y=0.5 lc rgb 'blue'
50 ### 上边用到了缩写ps=pointsize.其他也有缩写：
```

```
### 如line=l, linetype=lt, pointtype=pt等等

######### 子图 (2): 数据文件绘图和拟合

reset              ### Gnuplot会继承上边的命令,
                   ### 所以需要reset取消之前所有的设置
set title "fitting functions"
set label "(2)" at graph 0.02,0.03 font ',20'
                                textcolor rgb 'red'
@sx [0:11]         ###这里引用了宏
set yrange [0:11]
set key bottom right font ",14" spacing 2
g(x) = a*x**2 + b*x + c          ###定义函数并拟合, 参数为a,b,c
fit g(x) "data.txt" u 1:2 via a,b,c ###定义函数并拟合
key_g= sprintf("fits without yerror:\ng(x) = %5.3f*x^2 +
                                        %5.3f*x + %5.3f",a,b,c)
h(x) = d*x**2 + e*x + f
fit h(x) "data.txt" u 1:2:3 yerror via d,e,f
key_h= sprintf("fits with yerror:\nh(x) = %5.3f*x^2 +
                                        %5.3f*x + %5.3f",d,e,f)
set xlabel 'xxxx' rotate by 45
plot "data.txt" u 1:2:3 w yerror pt 5 ps 1.0 lc
                                      rgb 'blue' title "data",\
g(x) w line linecolor rgb 'red' title key_g,\
h(x) w l lc rgb 'green' title key_h

######### 子图 (3): 统计和填充
reset
set key left top box
set label "(3)" at graph 0.02,0.03 font ',20'
                                            textcolor rgb 'red'
df='data.txt'                  ###这里使用文件里的数据绘图

stats df u 1:2 name "A"
###统计数据的1和2列, 并将统计结果存入A中

print A_min_x, A_min_y
```

```
88  ###可以打印出A中统计的x的最大和最小值
89
90  @sx [A_min_x-1: A_max_x + 1] ###根据统计数据设置x轴范围
91  set yr [A_min_y-1: A_max_y + 1]
92  set arrow from A_min_x,A_min_y+2 to A_min_x,A_min_y ###绘制箭头
93  set label 'Min.' at A_min_x,A_min_y+2.3 center
94  set arrow from A_max_x,A_max_y-2 to A_max_x,A_max_y
95  set label 'Max.' at A_max_x,A_max_y-2.3 center front
96  set xlabel 'xxxx' textcolor rgb 'red'
97  set style fill solid 0.5
98  #set style fill pattern 3
99  plot df u 1:2 w filledcurves y=2 lc rgb 'seagreen' title 'fill
100                                                        with y=2',\
101 df u 1:2 w lp pt 4 ps 0.9 lc rgb 'red' lw 2,\
102 [3:6] A_mean_y w l dt '--' lw 2 lc rgb 'black' title "mean"
103
104 ######## 子图 (4):直方图
105 reset
106 set label "(4)" at graph 0.02,0.03 font ',20'
107                                              textcolor rgb 'red'
108 set style data histogram
109 set style fill solid 1.0 border lt -1
110 set boxwidth 1.0
111 set key at 1.5,7  box font ',15' reverse
112 set yr [0:10]
113 set xtics ("A" 0, "B" 1, "C" 2, "{/Symbol b}" 3, "{/Symbol S}" 4)
114 set xlabel 'xxxx' offset 0,-1
115 plot "data.txt" u 1 title 'histogram' lc rgb 'seagreen',\
116 "data.txt" u 0:($1+0.5):1 w labels title ""
117
118 unset multiplot  ###退出多图模式，完成绘图并保存
```

Chapter 5

支持库 (了解)

本章节主要介绍以下 Nektar 软件安装时的各种支持库及其用途.

5.1 编译器

编译器仅作了解即可, 上文中本地安装 Nektar++ 的时候用的是比较轻量级而且方便安装的 gcc 编译器;

5.1.1 Intel C++ 编译器

Intel 的 C++ 编译器, 是针对 Intel 本家的 CPU 做过性能优化, 有偏向性计算的编译器版本, 在 Intel CPU 上有着比 gcc 更佳的表现; 一般在跑程序的时候还会使用 Intel 本家的数学库 Intel MKL 库, 甚至 Intel MPI.

Intel Compiler 的安装都是通过 Intel 的 One-API 完成安装的; 其中, 涉及 Intel MPI 的安装, 在 2017 前-2018 后存在明显不同 (也可以不用 Intel MPI);

5.1.2 gcc 编译器

以下 Copy 自百度百科:

GCC（GNU Compiler Collection，GNU 编译器套件）是由 GNU 开发的编程语言编译器.GNU 编译器套件包括 C、C++、Objective-C、Fortran、Java、Ada 和 Go 语言前端，也包括了这些语言的库.

GCC 的初衷是为 GNU 操作系统专门编写的一款编译器.GNU 系统是彻底的自由软件. 特别注意,一般而言, 不同的同世代编译器和调用的库的不同所产生的性能差异最大不会超过 30%.

5.1.3 gfortran 编译器

与 gcc 对应的 gfortran 编译器, 是用来编译 FORTRAN 语言的编译器; 当然也有 ifortran, 同理;FORTRAN 语言是比较早期的数学计算机语言, 乃至早期的数学包 (例如以下的早期的 BLAS 包,LAPACK 包都是用 FORTRAN 语言编写的); 如今很多行业其实还在利用 FORTRAN 语言进行计算, 例如: 海洋, 气象; 不过目前很多都在用 FORTRAN 90 版本的语言, 除非要调用很早期编写的, 一直没有修改的程序, 才会用上 FORTRAN 77;

值得一提的是, 下面提到的一些包其实是用 C/C++ 语言编写的, 比如经常会用到的 FFTW 包;

5.2 数学库

以下大多内容是百度百科/知乎/库官网转载过来的:

5.2.1 BLAS 库与 openBLAS 库

BLAS (Basic Linear Algebra Subprograms) 基础线性代数子程序库，里面拥有大量已经编写好的关于线性代数运算的程序.

BLAS（basic linear algebra subroutine）是一系列基本线性代数运算函数的接口（interface）标准. 这里的线性代数运算是指例如矢量的线性组合，矩阵乘以矢量，矩阵乘以矩阵等. 接口在这里指的是诸如哪个函数名实现什么功能，有几个输入和输出变量，分别是什么.

BLAS 被广泛用于科学计算和工业界，已成为业界标准. 在更高级的语言和库中，即使我们不直接使用 BLAS 接口，它们也是通过调用 BLAS 来实现的（如 Matlab 中的各种矩阵运算）.

BLAS 原本是用 Fortran 语言写的，但后来也产生了 C 语言的版本 cBLAS，接口与 Fortran 的略有不同（例如使用指针传递数组），但大同小异.

注意 BLAS 是一个接口的标准而不是某种具体实现（implementation）. 简单来说，就是不同的作者可以各自写出不同版本的 BLAS 库，实现同样的接口和功能，但每个函数内部的算法可以不同. 这些不同导致了不同版本的 BLAS 在不同机器上运行的速度也不同.

BLAS 的官网是Netlib，可以浏览完整的说明文档以及下载源代码. 这个版本的 BLAS 被称为 reference BLAS，运行速度较慢，通常被其他版本用于衡量性能. 对于 Intel CPU 的计算机，性能最高的是 Intel 的 MKL （Math Kernel Library）中提供的 BLAS. MKL 虽然不是一个开源软件，但目前可以免费下载使用 (商用好像是收费的). 如果想要免费开源的版本，可以尝试 OpenBlas 或者 ATLAS2. 另外，无论是否使用 MKL，BLAS 的文档都推荐看 MKL 的相关页面.

另外就是 OpenBLAS:

OpenBLAS 是一个基于 BSD 许可（开源）发行的优化 BLAS 计算库，由张先轶于 2013 年 7 月 20 日发起，并发布 OpenBLAS 0.2.7 第一个版本.BLAS（Basic Linear Algebra Subprograms 基础线性代数程序集）是一个应用程序接口（API）标准，用以规范发布基础线性代数操作的数值库（如矢量或矩阵乘法），OpenBLAS 是 BLAS 标准的一种具体实现,官网.

本地安装:

```
sudo apt-get install gfortran libopenblas-dev
```

5.2.2 LAPACK 库

LAPACK 是由美国国家科学基金等资助开发的著名公开软件.LAPACK 包含了求解科学与工程计算中最常见的数值线性代数问题，如求解线性方程组、线性最小二乘问题、特征值问题和奇异值问题等.

5.2.3 Intel-MKL 库

前边已经提到过一部分, 是 Intel 的集成数学库;

5.2.4 FFTW 库

FFTW 意为 Faster Fourier Transform in the West，是一个 C 语言的快速计算离散傅里叶变换库，它是由 MIT 的 M.Frigo 和 S. Johnson 开发的，可计算一维或多维实和复数据以及任意规模的 DFT. 目前最新版本为 3.3.10，其官网地址为：https://www.fftw.org/。

大量测试结果表明，FFTW 库要比其它开源傅里叶变换库或软件要快，因此如果你的程序中包含傅里叶变换的相关计算，FFTW 库是一个很好的选择. 在 Quasi-3D 的情况中, 往往需要在第三个方向上利用 FFT 拉伸, 会用到 FFTW 包.

本地安装:

```
sudo apt-get install libfftw3-dev
```

5.2.5 ARPACK 库

Arpack (ARnoldi PACKage) 最初是一个 Fortran 语言编写的，用于求解大型本征方程的程序库，其主要算法是 Arnoldi 循环, 在 Nektar++ 中的使用主要体现在稳定性分析.

本地安装:

```
sudo apt-get install libarpack++2-dev
```

安装时的依赖库有 OpneBLAS 库, 用以上命令安装 ARPACK 会自动安装 OpenBLAS.

5.2.6 netcdf 库与 pnetcdf 库

不详细介绍了, 主要是多维的空间数据, 在海洋和气象上用的比较多; 组里老师写的磁流体也会调这个库;

5.2.7 hdf5 库

HDF5 (Hierarchical Data Format) 由美国伊利诺伊大学厄巴纳-香槟分校 UIUC (University of Illinois at Urbana-Champaign) 开发，是一种常见的跨平台数据储存文件，可以存储不同类型的图像和数码数据，并且可以在不同类型的机器上传输，同时还有统一处理这种文件格式的函数库.HDF5 库是文件类型支持库.

5.3 并行库 (MPI 库)

主要有以下几个库:

- mpich 库
- intel-mpi 库
- openmpi 库
- mpi-hpcx 库

具体信息比较多, 可以参考科大超算中心的PPT.

Chapter 6

额外提醒及注意事项

主要就是以下几点提醒和一些额外事项:

6.1 用工具

用工具不一定非要是 CFD 工具/工具书; 也可以是 GPT-4,github 这种; 譬如, 一些特定的网格如果自己不愿意画也可以在 github 上找一个相对成熟能用的版本用, 改一改参数即可, 这样能省许多功夫 (当然, 如果有瑕疵或者使用上有特殊要求就需要自己在此基础上再做修改了); 至于 GPT,GPT-4 功能目前来说好用很多, 免费的 new bing 和 GPT-3.5 在解决很多专业相关的问题的时候功能性比较差, 且 GPT-3.5 还会乱写代码; 如果没有太大需求还想 save money, 可以用字节 Coze 的 BOT 的 API 接口 (不过需要科学上网);

GPT-3.5 应付日常的知识性提问问题不大;

6.2 查论文

查书和论文都相对必要 (书主要是补一下知识缺漏, 找论文就比较有针对性了); 一般而言, 比如在画网格时候, 可以找一下同类问题的论文, 看一看论文里的网格是怎样安排的, 密度分布如何, 效果怎样; 又比如, 在求解 NS 方程的过程中, 需要给流场添加扰动 (噪声), 那么噪声相对于流场的背景平均流速的比例应该控制在什么范围比较合理, 不会对实验结果产生决定性影响 (P.S. 在这个问题背景下甚至需要有考虑噪声比例和湍流密度 Tu 之间的关系的一些中间过程);

总之, 找论文做参考也是比较重要的一环, 适当参考已有的学术成果能少走不少弯路;

Web of Science,google 学术,Sci-Hub 三件套就不多提了, 只要不是很重要 (重要到

某些参数的设置需要在论文里写进参考文献的那种), 一般很够用, 而且也不会有什么大问题;

6.3 问老师

有问题一定要及时问老师; 自己一个人想问题比较容易陷入误区;

6.4 Extra Note 1: 环境配置

在本地或者服务器上, 用"vim .bashrc" 命令打开预加载环境变量可进行添加 or 调整; 在服务器上略有不同, 直接在后方加"module load xxx" 即可在打开服务器终端打开的时候预加载环境 (编辑后重新打开终端生效), 这样就免去了每次都要 source 一下环境脚本的麻烦;

另外, 在需要找某些已搭载环境的目录的时候, 用"locate xxx" 进行搜索; 比如, 我想要了解 intel 编译器的 mkl 库中的 lapack 库的位置, 只需"locate lapack" 即可 (如果没有找到就不会显示; 安装软件的时候如果显示某共享库文件丢失, 也可以 locate 找到没有丢失此库的环境版本进行搭载);

6.5 Extra Note 2: VTK/VTU 格式浅析

详细的格式介绍见vtk 官网, 此处只给出一些用得到的信息;

Chapter 7

FORTRAN 90 速通指南

因为组里程序有不少是基于 Fortran 90 编写的, 也就催生了学习 F90 的需求 (尤其是在改程序的时候); 但是系统学习一门语言时间成本又相对较高, 所以此处便有了这速成版 (接近于速查手册); 此处看完应该可以基本达到无障碍看程序的水平;

此外, 也有一点需要明确:F90 向下兼容 F77, 所以 F77 程序理论在 F90 中均能跑通 (所以不需要再看 77 啦), 但是 F95 因为没有接触过, 所以情况未知;

下文默认已掌握至少一门编程/计算语言 (C/C++/Python/Matlab/R/JAVA), 其底层的基础逻辑 (运算顺序, 运算符号等) 基本是一致的, 便不再赘述, 使用时按照习惯走即可 (不同的地方会单独列出来); 因为时间和篇幅有限 (而且也确实用不到), 所以关于指针的部分就不讲了, 不过这门语言有指针的!

7.1 基础知识与程序书写

1."!" 开头是注释语句;
2. 设置变量名称需要注意的几点:

- (1) 表意时单词连接用"_" 而不是"-", 否则名称非法;
- (2) 名称里不能出现"." 字符;
- (3) 非字母不能作为第一字符;
- (2) 名称里不能有空格;

3.F90 的程序单元架构主要分 4 类:

- (1) 主程序单元 (至少 1 个);
- (2) 外部子程序单元 (如函数程序等, 不调用外部程序, 可作为子程序被调用, 一般由 FUNCTION 或者 SUBROUTINE 开头);

- (3) 模块单元 (过程接口, 可认为是允许调用外部程序单元的子单元, 用 MODULE 开头);
- (4) 块数据单元 (BLOCK DATA, 不可被执行);
- 注:(1-3) 可包含内部子程序;

4. 程序结构 (主程序):

主程序　PROGRAMME 语句: PROGRAMME xxx

　　说明部分 (内部, 派生, 数组, 指针数据的说明)

　　操作部分 (非说明语句, 以上至少应有一个可执行语句)

　　内部子程序部分 (CONTAINS hhh<-子程序名称)

　　END 语句: END 或者 END PROGRAMME xxx

5. 关于语句: 主要是可执行与不可执行两类 (听君一席话, 如听一席话), 执行是操作语句, 不可执行是描述语句 (如数据类型描述等); 有一个执行顺序需要捋顺, 否则会报错, 见下图7.1.

表 3-2　语句排列次序

	OPTION 语句		
	PROGRAM,FUNCTION,SUBROUTINE,MODULE 或 BLOCK DATA 语句		
	USE 语句		
注释行，INCLUDE 语句和指令	NAMELIST, FORMAT 和 ENTRY 语句	IMPLICIT NONE 语句	
		PARAMETER 语句	IMPLICIT 语句
		PARAMETER 和 DATA 语句	派生类型定义，接口块，类型说明语句，语句函数语句和说明语句
		DATA 语句	可执行语句
	CONTAINS 语句		
	内部子程序或模块子程序		
	END 语句		

图 7.1: (从书上挖过来的) 语句执行优先级列表

6. 关于数据类型, 见下图7.2.

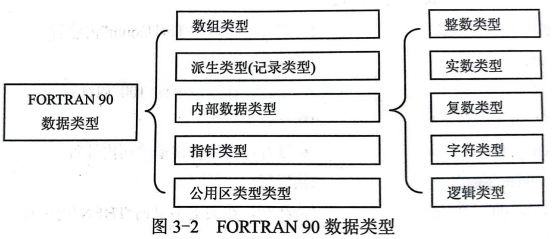


图 3-2　FORTRAN 90 数据类型

图 7.2: (书上挖过来的) 数据类型一览

7. 写表达式换行但不间断在行尾加”&”.

7.2 数据结构与基础程序

7.2.1 数据类型

数据类型一共有 5 种: 整型 (integer),实型 (real),复型 (complex!!!), 字符型 (character) 和逻辑型 (logical,注意: 不是 boolean!!). 存储的位数可以具体参数化, 如下图7.3.

表 4-1 内部数据类型 KIND 值参数及存储开销

类型	子类型	KIND 值	字节数	说 明
整型	BYTE	1	1	与 INTEGER（1）等同
	INTEGER	1、2 或 4	1、2 或 4	与缺省有关，INTEGER 有 1、2、4 个字节。初始缺省为 4，缺省值可通过编译选项改变
	INTEGER（1）	1	1	
	INTEGER（2）	2	2	
	INTEGER（4）	4	4	
实型	REAL	4 或 8	4 或 8	与缺省有关， REAL 有 4 或 8 个字节。初始缺省为 4，缺省值可通过编译选项改变
	REAL（4）	4	4	
	DOUBLE PRECISION	8	8	与 REAL（8）等同
	REAL（8）	8	8	
复型	COMPLEX	4 或 8	8 或 16	与缺省有关，COMPLEX 有 8 或 16 个字节。初始缺省为 8，缺省与实型缺省值有关（2 倍）
	COMPLEX（4）	4	8	
	DOUBLE COMPLEX	8	16	与 COMPLEX（8）等同
	COMPLEX（8）	8	16	
字符型	CHARACTER	1	1	CHARACTER 与 CHARACTER（1）等同。 1 是 KIND 值，不是字符串长度
	CHARACTER*len	1	len	len 是字符串长度。1≤len≤65535
逻辑型	LOGICAL	1、2 或 4	1、2 或 4	与缺省有关，LOGICAL 有 1、2、4 个字节。初始缺省为 4，缺省值可通过编译选项改变
	LOGICAL（1）	1	1	
	LOGICAL（2）	2	2	
	LOGICAL（4）	4	4	

图 7.3: (书上挖来的) 数据类型一览

7.2.2 常量

而常量也有同样的类型, 字符型需按字符串处理, 逻辑值只能进行逻辑运算 (按 1/0 计: 允许某些情况下按整型参与计算), 其余均为算数型常量;

其中, 实型常量分两种; 一种是小数 (浮点数) 形式, 另一种是指数形式; 指数形式有两种标识, 这个会常用到: 一个是 e(或者 E),E 后幂指只能是整数; 另一种是 d(或者

D), 这种只能指定其为双精度实数 (说人话就是:Kind 默认为 4, 此处加 d 后 Kind 恒等于 8, 无法指定其他数),d 后仍接整数, 用法与 e 相同; 算程序的时候尤其要小心, 若数据小于 $-1.117549435 \times 10^{-38}$, 很有可能就已经达到 Fortran 语言设定限制的运算精度了, 所以很有可能会出现数据强烈震荡的现象 (画图来看), 其实是这个原因;

而复型常数由两个实型指定; 比如 (25,12) 指 25+12i;

字符型有三种 (一般字符串/H 字符串/C 字符串,H 可指定长度,C 可存储非打印字符如 \n) 用单引号双引号括起赋值均可, 只需前后一致; 字符串内部需要引号时建议与两侧的交替使用;字符串区分大小写;

逻辑型只有两个值".true." 和".false."; 上文提到过可按整型处理, 但注意.true. 的值为-1,.false. 值为 0(其值是根据二进制数换算过来的, 比如.true. 每个存储位都是 1 所以才视为整数-1);

7.2.3 变量

变量直接进行指定即可 (类型同上); 另外, 也可进行隐式 (implicit) 变量声明 (优先级比显式低, 尽量避免重复声明导致的冲突); 声明句如下:

```
1 REAL e, f
2 IMPLICIT INTEGER(a, b)
3 INTEGER(KIND=2) g
```

在声明整型变量时"::" 符号代表可直接赋初值 (若加了"::" 没赋值则该变量值为 0), 不加则不能, 如下例:

```
1 INTEGER:: g=22,h    ! g=22,h=0
```

几点注意事项:

- (1)KIND=1 的整型声明可改用 BYTE 进行;
- (2)INTEGER(i) 与 INTEGER*i 等价; 其余类型同理;
- (3) 复数只能进行显式声明;
- (4) 字符型可声明长度 (如下例);

```
1 CHARACTER(LEN=8)t, u, i, o
```

7.2.4 表达式

下面来说计算与表达式:

表达式分为 4 种: 算数/关系/逻辑/字符; 对于算数表达式,"**" 指指数运算; 对于字符串表达式, 主要涉及取子串和连接两个操作. 如下例:

```
1 CHARACTER::t='qwerty',p,q
2 p=t(2:4)      !p为wer
3 q=p//t        !q为werqwerty
```

关系表达式主要输出逻辑值 (大小可直接用 ==,< 等符号, 也可用.LE. 表示 <,LaTeX 也有类似写法);

逻辑表达式运算符大赏:".NOT."(非),".AND."(与),".OR."(或),".XOR."(异或),".EQV."(等),".NEQV."(不等); 真值表略;

7.2.5 输入与输出

输入语句如下 (基本相当于 scanf):

```
1 READ *,i,j
```

其中,"*" 指从隐含的输入设备 (如键盘中输入数据, 当然也可推入-如下例, 会把 input 文件里的值推给 f90 文件再执行),"*" 前后的空格都可有可无; 每个 READ 都从新的一行开始读取数据; 在读取时逗号和空格可混合使用; 若同时在一句中对多种类型的变量进行读取, 分批 (多用几次回车) 输入也是可行的; 另外, 在输入时用"n * p" 可进行重复输入 (比如"3*25" 相当于连续输入 3 个 25).

```
1 ifort xxx.f90 < input
```

输出语句类似:

```
1 PRINT *,i,j
```

输出时系统默认宽域输出, 同一命令同时输出几个量的时候, 中间会自动插入空格.

```
1 PRINT *,"i=",2345   !结果为i= 2345(2345前有一个空格)
```

逻辑型的输出为"T" 或者"F".

7.2.6 参数与函数

参数一般指不变的常参数 (如 e, π), 参数的作用域只有该程序内部, 且一旦定义好便按照"const" 型数据进行处理, 不能修改; 参数可以是常值、字符或者逻辑表达式等, 且必须定义在可执行表达式前 (也就是放在声明区, 否则会有逻辑错误);

```
1 PARAMETER (pi=3.14159265358973846)
2 PARAMETER (euler=2.714828)
```

至于函数, 一共两种 (与其他面向过程语言类似), 标准函数与自定义; 标准函数诸如 SQRT/SIN/ABS 等, 略 (都是些常用函数, 下边会在图7.4列一下), 自定义的情况涉及东西比较多, 后边单独讲;

表 4-20　常用标准函数

标准函数	变元类型	变元个数	含　义	数学表示
ABS(x)	实型	1	求绝对值	|x|
COS(x)	实型	1	求余弦	cosx
EXP(x)	实型	1	求指数	e^x
INT(x)	实型	1	取整	⌊x⌋，取 x 整数
LOG(x)	实型	1	求自然对数	lnx
LOG10(x)	实型	1	求常用对数	$\log_{10}x$
MAX(x1,x2,x3,...,xn)	实型	≥2	求最大值	max（$x_1,x_2,x_3,\cdots,x_n$）
MIN(x1,x2,x3,...,xn)	实型	≥2	求最小值	min（$x_1,x_2,x_3,\cdots,x_n$）
MOD(m,n)	整型	2	求余数	m－⌊m/n⌋× n,求余数
REAL(i)	整型	1	转实数	将 i 转换为实数
SIGN(x,y)	实型	2	求符号	|x|（当 x>0），-|x|（当 x<0）
SIN(x)	实型	1	求正弦	sinx
SQRT(x)	实型	1	求平方根	$\sqrt{x}$
TAN(x)	实型	1	求正切	tgx
HUGE(x)	整型、实型	1	求最大范围值	求 x 所属类型最大正值
TINY(x)	实型	1	求最小范围值	求 x 所属类型最小正值
KIND(x)	内部类型	1	求 KIND 值	

图 7.4: (书上挖的) 常用标准函数表

再介绍一下 END/STOP/PAUSE 三个语句及其用法;

END 用于终止主程序运行 (当然整个程序也是,END 后的代码为无效代码, 不会执行);STOP 直接终止程序运行;PAUSE 是挂起, 按回车继续 (用于调试);

7.3　带格式的输入输出

可以用 FORMAT 命令标明输入输出格式, 按照行数从 PRINT 等命令中进行指定, 如下例,READ 指定用第三行指定的格式, 两个数都是占三字符位的整数 (I)(X 指定空格):

```
1 READ 3,m,n
2 ...
3 FORMAT(I3,I3)
```

格式编辑符可用”,” 或者”/” 进行间隔; 一个比较偷懒的方式是直接将格式写在 READ/PRINT 中, 比如:

```
1 READ ”(I3,I3)”,m,n
2 PRINT ”(1X,'m+n=',I4)”,m+n
```

以下是编辑符表7.5:

表 5-1 可重复编辑符

格 式	名 称	功 能	用于输入	用于输出
[r]Im[.n]	整型编辑符	将整数按指定域宽输入输出	√	√
[r]Bm[.n]	二进制编辑符	将二进制数按指定域宽输入输出	√	√
[r]Om[.n]	八进制编辑符	将八进制数按指定域宽输入输出	√	√
[r]Zm[.n]	十六进制编辑符	将十六进制数按指定域宽输入输出	√	√
[r]Fm.d	小数型实型编辑符	将实数按指定域宽输入输出	√	√
[r]Em.d[Ee]	指数型实型编辑符	将实数按指定域宽输入输出	√	√
[r]Dm.d	双精度实型编辑符	将双精度数按指定域宽输入输出	√	√
[r]ENm.d[Ee]	工程计数法编辑符	将实数按工程计数法输入输出	√	√
[r]ESm.d[Ee]	科学计数法编辑符	将实数按科学计数法输入输出	√	√
[r]A[m]	字符型编辑符	将字符串按指定域宽输入输出	√	√
[r]Lm	逻辑型编辑符	将逻辑值按指定域宽输入输出	√	√
[r]Gm.d[Ee]	通用编辑符	将任意类型数据按域宽输入输出	√	√

表 5-2 不可重复编辑符

格 式	名 称	功 能	用于输入	用于输出
字符串	字符串编辑符	将字符串输出		√
nH	H 编辑符	将 H 后 n 个字符输出		√

格 式	名 称	功 能	用于输入	用于输出
Q	字符计数编辑符	获得输入记录中剩余字符数	√	
Tc, TLc, TRc	位置编辑符	指定输入输出记录中的位置	√	√
nX	位置编辑符	指定输入输出记录中的位置	√	√
SP, SS, S	+号编辑符	控制+号输出		√
/	斜杠编辑符	终止本记录，开始下一记录	√	√
\	反斜杠编辑符	下一记录接上一记录输出		√
$	$编辑符	下一记录接上一记录输出		√
:	格式控制编辑符	输出表中无输出数据终止格式控制		√
kP	比例因子编辑符	设置 F 和 E 编辑符指数比例因子	√	√
BN, BZ	空格替代编辑符	指定数字串中空格意义	√	

图 7.5: (书上的) 编辑符表

单看编辑符表可能会有点懵 (这和上边示例也不一样哇 ww); 其实是这样: 大写字

母是标识符, 后边是可加参数; 以实数为例 [r]Fm.d, 具体来说,5F7.3 指的是整数部分 7 位, 小数部分 3 位, 重复 5 次 (其他编辑符同理);

另外特别需要注意的地方:

- (1) 小数型如果输入数据原本有小数的话, 按照原本带宽进行读入, 格式限定不起作用; 输出时会被四舍五入;
- (2) 复数型输入实部和虚部之间用一个空格相连;

7.4 选择与循环结构

条件语句 IF: 逻辑 IF 句可执行 (如下)

```
1 IF(i<j) PRINT *,'NO!!'
```

另一种就是块 IF 语句, 是包含 END IF/ELSE/ELSE IF(多分支) 的结构快 (方式与其他语言基本相同, 不过有 THEN, 同下例):

```
1 IF(A<B) THEN
2   IF(B<C) THEN
3     WRITE *,A
4   ELSE
5     WRITE *,B
6   ENDIF
7 ENDIF
```

注:不论是条件还是循环, 都十分建议做好对齐工作!!

还有 CASE 语句 (对应到 C 中的 switch, 此处为 SELECT), 见下例 (一个不管怎么选都会输出 2 的块):

```
1 READ *,S
2 SELECT CASE (S)
3 CASE 1
4   PRINT *,S+1
5 CASE 2
6   PRINT *,S
7 CASE DEFAULT !这个DEFAULT不是必须有的
8   PRINT *,2
9 END SELECT
```

循环分为 3 大类, 分别为" 计数型 (DO)"," 当型 (DO WHILE)"," 直到型 (第一步不判别先循环这种实现不了, 可以改 while 出口条件实现)"; 下边分别介绍.

DO 型基本相当于 matlab 中的 For 循环, 可指定循环变量的初值, 终值以及步长 (可缺省, 默认为 1); 如下例就是一个以 2 为步长,1 为初值,10 为终值, 循环变量为 v 的程序块;

```
1 DO v=1,10,2    !顺序为:初值,终值,步长(可缺省,默认1)
2 PRINT *,v      !v也可以提前进行声明
3 END DO         !DO循环可命名,如Odd: DO v=...,
4                !不过结束必须为END DO Odd
```

当然, 还有一种方式不用设定上述条件, 只需设置跳出条件直接跳出循环即可:

```
1 DO
2 ....
3 IF (x<1) EXIT
4 END DO
```

DO WHILE 循环格式与用法类似其他各语言的 while 循环;

```
1 DO WHILE (x<1)
2 ...
3 END DO
```

最后介绍以下跳出. 其中 EXIT 已经用到过了 (用于终止循环), 还有一个 CYCLE(跳回循环最开始重新执行);

7.5 数组

FORTRAN 的数族序号比较有特点 (因为标号可以自定义上下界, 不必从 0or1 开始), 举个栗子:

```
1 REAL ddr(5:9)
2 !产生了一个一维数族,元素为ddr(5)...ddr(9)共5个
3
4 INTEGER ppr(-3:3,0:1)
5 !产生了一个二维数组,元素ppr(-3,0)...ppr(3,1)共14个
6
7 INTEGER::num(3)=(/12,23,34/)
8 !用"::"允许在声明中赋值,如未说明,索引从1开始
```

若下界超过上界, 默认数组大小为 0(合法); 另, 关于维数:

```
!DIMENSION的几种用法(嫌麻烦可不用)
!一般case2同一赋维数用处较大
!CASE 1
INTEGER day
DIMENSION day(3)

!CASE 2
INTEGER,DIMENSION(3):: num,pp,op(1,1)
!此处num,pp均一维,op因为已经指定,故不受影响
```

数组元素的引用同 MATLAB(直接小括号定位即可, 略); 索引元素若为浮点数则自动取整处理;

至于数组的赋值, 也可以用循环进行赋值; 如以下隐含型 DO 循环赋值 (两种赋值等价):

```
READ *,((a(i,j),j=1,N),i=1,M)

DO i=1,M
  DO j=1,N
    READ *,(i,j)
  END DO
END DO
```

或者, 利用数组构造器进行赋值, 如下例 (可以使用已知变量/分块赋值):

```
INTEGER:: i=2
INTEGER num(6)

!一维数组的赋值
num=(/1,2,3,4,5,6/)
num=(/1,i,3,4,5,6/)

!也可以将其直接赋值为二维数组,下例为2行3列情况
num=RESHAPE(/1,2,3,4,5,6/),(/2,3/)

!以下分段赋值也可以
INTEGER nuu(2,3)
nuu(1,:)=(/1,2,3/)
nuu(2,:)=(/4,5,6/)
```

再或者, 利用 DATA 语句赋值:

```
INTEGER m,n,nuu(-5:5)

DATA m,n/2*1/
DATA nuu/1,2,5*6,8,9,10/
!上一行nuu为[1,2,6,6,6,6,6,8,9,10]
```

甚至可以进行条件赋值 (数组赋值专用 IF:WHERE+[ELSEWHERE/ENDWHERE] 语句):

```
INTEGER nu(-5:5)

DATA nu/1,2,5*6,8,9,10/
PRINT *,nu
!此时输出 1 2 6 6 6 6 6 8 9 10

WHERE (nu>7) nu=nu-7
PRINT *,nu
!此时输出 1 2 6 6 6 6 6 1 2 3(以下也等价)

WHERE (nu>9)
   nu=nu-7
ELSEWHERE (nu==8)
   nu=nu-7
ENDWHERE
!ELSEWHERE也可以不加条件表示剩余全部
```

最后一部分介绍动态数组; 动态数组缺点是存储开销较大 (元素个数不确定的时候用这个), 如下例;

```
INTEGER,DIMENSION(:),ALLOCATABLE::sc
INTEGER n
read *,n
ALLOCATE(sc(n))

!格式:下例2维动态数组,可灵活调整
!      ... DIMENSION(:,:),ALLOCATABLE::(数组名)
!      ALLOCATE(...)   或在上一行不写明维数直接赋值
```

动态数组不用时还可以拆掉释放存储 (用 DEALLOCATE, 工具人的通常结局):

```
1 !接上例
2 DEALLOCATE(sc) !释放sc数组的存储空间
```

最后的最后, 是数组中常用的矩阵函数表, 如下图7.6;

表 8-2 矩阵函数

函数名	功能
ALL(mask[,dim])	判断逻辑数组 mask 在选定维上的所有元素是否都为真
ANY(mask[,dim])	判断逻辑数组 mask 在选定维上的任一元素是否为真
COUNT(mask[,dim])	求逻辑数组 mask 在选定维上的真元素个数
CSHIFT(array,shift[,dim])	对数组选定维上的数组元素进行一或多次循环移位
DOT_PRODUCT(vector_a,vector_b)	执行向量的点积运算
LBOUND(array[,dim])	求数组在选定维上的下界
UBOUND(array[,dim])	求数组在选定维上的上界
MATMUL(array_a,array_b)	执行二维矩阵的乘积运算
MAXLOC(array[,mask])	求满足指定掩码条件的最大值的位置
MINLOC(array[,mask])	求满足指定掩码条件的最小值的位置
MAXVAL(array[,dim][,mask])	求在所选定维上满足指定掩码条件的最大值
MINVAL(array[,dim][,mask])	求在所选定维上满足指定掩码条件的最小值
SUM(array[,dim][,mask])	求在所选定维上满足指定掩码条件的数组元素之和
MERGER(tsource,fsource,mask)	根据所给定的掩码条件合并两个数组
SHAPE(source)	求数组的形状
RESHAPE(source,shape[,pad][,order])	重新定义数组的形状
TRANSPOSE(array)	执行二维矩阵的转置运算
SIZE(array[,dim])	求数组在所选定维上的大小

图 7.6: (挖的) 矩阵函数表

7.6 自定义函数与子程序

说到底, 自定义函数和子程序主要是为了问题定制化, 计算规范化, 功能模块化, 避免一定程度的重复工作, 追求更高效率; 子程序可以看作是附功能块 (utilities), 单独编写到一个程序文件里, 等到用的时候再进行调用;

比如以下程序 (例子可能比较烂), 假如已知一个班所有人成绩, 现在要算班级每个人距离平均成绩的离差, 再输出; 实际上可以分成两步 (子程序) 进行; 第一个子程序用于计算均值再算离差, 第二个程序输出离差;

```
PROGRAMME main
  PARAMETER (N=30)
  REAL data(N)
  CALL remain(data)
  CALL output(data)
CONTAINS
  SUBROUTINE remain(A)
    INTEGER A(N)
    REAL::aver=0d0
    DO i=1,N
      aver=aver+A(i)
    END DO
    aver=aver/N;
    DO i=1,N
      A(i)=A(i)-aver
    END DO
  END SUBROUTINE remain

  SUBROUTINE ouput(A)
    REAL A(N)
    DO i=1,N
      PRINT *,A(i)
    END DO
  END SUBROUTINE ouput
END
```

设计算法的时候一般自顶而下, 先设计主流程, 然后再对各分流程进一步细化; 子程序的分类一般如图7.7所示;

7.6.1 标准子程序

这部分类型会比较多, 所以只会挑出一些日常有可能会用到的来讲;

标准函数共分 9 大类:

- (1) 三角函数:SIN(x) 等 (弧度),SIND(x) 等 (度);

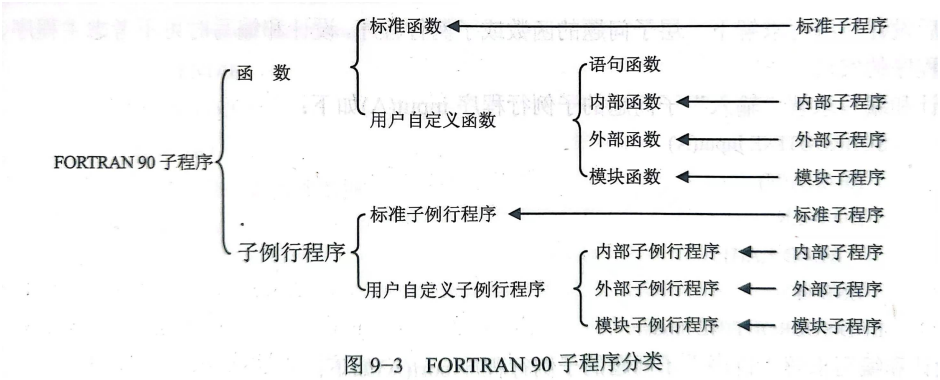


图 9–3 FORTRAN 90 子程序分类

图 7.7: 子程序的分类表

- (2) 数值计算函数:MIN(x),MAX(x),ABS(x) 等;
- (3)(变量) 类型转换函数:FLOAT(x) 等;
- (4)(变量) 类型查询函数: 查询最大值 HUGE(x), 数值精度 PRECISION(x);
- (5) 随机数生成函数:RAN(x), 其中 x 必须为一个 KIND=4 的整型变量;
- (6) 日期时间处理函数
- (7) 字符串处理函数
- (8) 地址计算函数:LOC(x) 以及%LOC(x) 用于计算 x 的存储地址;
- (9) 位运算函数

标准子例行程序有 5 大类:

- (1) 程序控制子例行程序:EXIT(),SLEEPQQ(x) 程序睡眠 x 秒;
- (2) 文件管理子例行程序
- (3) 随机数生成子例行程序:RANDOM(x) 和 RANDOM_NUMBER(x), 以及设置随机种子的 SEED(size),RANDOM_SEED(size);
- (4)) 日期处理子例行程序;
- (5) 数组处理子例行程序:SORTQQ(adrarray,count,size) 排序子程序, 三个参数分别为一维数组地址、待排序元素个数、数组存储开销 KIND;

注: 此处例行程序为 subroutine, 需要用 CALL 进行调用; 下例为排序调用示例;

```
USE DFLIB
INTEGER(2) arr(10),i
DATA arr/1,10,2,9,3,8,4,7,5,6/
CALL SORTQQ(LOC(arr),10,SRT$INTEGER2)
...
END
```

7.6.2 语句函数

定义一个用户自定义的表达式函数, 比如：

```
F(x,y)=x+y+x*y
READ *,a,b
DA=F(a,b)
```

语句函数的调用也如上所示; 但是也有一些规定, 需要遵循, 否则会报错:

- (1) 禁止套娃! 一个函数要展开全, 不能调用其他函数或者自身;
- (2) 一个语句函数只能有一个返回值;
- (3) 形参禁止出现数组或下边 (作为形参出现), 但是已知的数组元素可以调用;
- (4) 语句函数是单个程序内部定义的, 注意定义区域;
- (5) 语句函数定义可以没有形参, 但是括号不能没有;

7.6.3 内部子程序

内部函数和本节开头例子中的 SUBROUTINE 结构差不多, 可供参考 (注意同一文件/CONTAIN 语句位置即可);

```
programme main
  read *,x
  print *,f(x)
contain
  function f(x)
    f(x)=x**x
  end function    !或者 end function f
end
```

另外, 还可以多次用 RETURN 语句控制函数何时需要终止; 内部子例行程序与内部函数相仿, 不同的是没有 RETURN, 示例可参考本大节最初的程序;

7.6.4 形参与实参的传递

此处只列一些可能用到的点, 其他地方凭学过其他语言的直觉也基本不会出错;

1. 假定大小的数组做形参: 在形式数组的维说明的维上界加上"*", 称形式数组为假定大小数组 (不知道输入的数组的大小的时候用相对合适), 例如:

```
subroutine sub(A,B,C,n)
  real A(n,n),B(8,*),C(2*n,*)
  ...
end subroutine
```

注: 假定大小数组不允许出现在输入输出表 (语句) 中, 否则会编译出错; 且不能用 UBOUND、LBOUND、SIZE 函数确定数组声明为 * 的上下界和大小; 但是假定形状数组 (用":") 可完成以上来两个操作; 下例:

```
subroutine sub(A)
  real A(:,:)
  do i=lbound(A,1),ubound(A,1)
  ...
end subroutine sub
```

2. 子程序名作实参: 如下例;

```
PROGRAMME aliene
  INTRINSIC SIN
  EXTERNAL fun,proc,prt
  REAL fun,f1
  f1=fun(3.14159/6,SIN)    !调用SIN
  CALL proc(10,20,prt)     !用prt调用proc
END

FUNCTION fun(x,f)
  REAL x
  fun=f(x)
END FUNCTION

SUBROUTINE proc(x,y,p)
  INTEGER x,y
  CALL p(x+y)              !调用p作为形式子程序
END SUBROUTINE
```

```
18
19 SUBROUTINE prt(x)
20   INTEGER x
21   PRINT *,'x=',x
22 END SUBROUTINE
```

EXTERNAL 声明外部子程序 (or 函数),INTRINSIC 语句用于声明标准函数;
3."*" 作形参: 可以利用 RETURN 提供一个返回值机制 (可多个), 从略;

7.6.5 递归子程序

也就是允许一定程度的套娃调用; 递归函数如下例 (求阶乘的递归函数):

```
1 RECURSIVE FUNCTION factorial(m) RESULT(fact)
2   INTEGER m,fact
3   IF (m==1) THEN
4     fact=1
5   ELSE
6     fact=m*factorial(m-1)
7   ENDIF
8 END FUNCTION
```

递归子程序不用设置 RESULT, 将 fact 当作形参即可, 略;

7.6.6 外部子程序

允许子程序调用子-> 子程序, 在主程序外侧, 用 EXTERNAL 语句进行加载, 详情略;

7.7 派生类型与结构体

主要是可以允许用户自定义数据类型; 与 C 语言结构体极为类似; 有一个需要主注意的地方,在对成员进行引用时, 可以用":", 也可以用"%", 不过要尽量保持一致;如下例:

```
1 PROGRAMME lamort
2   TYPE stuinfo
3     INTEGER number
4     CHARACTER name
5     INTEGER age
```

```
6       REAL score
7     END TYPE
8
9     TYPE(stuinfo) student(30)   !声明stuinfo类变量student
10    ...
11
12    !赋值
13    student(1)%number=1
14    student(30).name='lidocaine'
15    DATA student(2).age/99/
16    READ *,student(3)           !键入顺序与类型声明时的顺序相同
17
18    !或者在声明时进行赋值
19    TYPE(stuinfo)::stupind=stuinfo(13,'ropivacaine',99,0)
20    TYPE(stuinfo)::stukind(2)=(/stuinfo(13,'stilnox',50,0),
21                                /stuinfo(7,'agomelatine',2,0)/)
22
23    !以下两句输出等价
24    PRINT *,stupind
25    PRINT *,stupind.number,stupind.name,stupind.age,
26                                                stupind.score
27  END
```

注意: 派生类型也不允许套娃 (套用其他结构);

7.8 文件与设备

在读入等操作进行时, 会涉及到输入输出设备的问题 (虽然之前都是用 * 表示从键盘/显示器输入), 以下图7.8所示为标准设备号表.

表 12-1 标准设备号

设备号（逻辑设备）	所连接物理设备
*	总是键盘或显示器
0	缺省情况下是键盘或显示器
5	缺省情况下是键盘
6	缺省情况下是显示器

图 7.8: 标准设备号表

通过 OPEN/CLOSE 语句可以打开关闭文件, 打开之后, 在关闭之前对数据进行修改即可;

有一个关于 (*,*) 类的参数的问题, 在下例中稍作说明 (后边会详细介绍):

```
1 OPEN(1,FILE='exam1.in')
2 OPEN(2,FILE='exam2.out')
3 READ(1,*) (score(I),I=1,N)
4 WRITE(2,*) '学生成绩有:',(score(I),I=1,N)
```

上例从设备 1 读取"exam1", 从设备 2 打开"exam2", 从"exam1"(设备 1) 获取键盘输入, 输出到"exam2"(设备 2) 上; 上表的 */5/6 与此处 1/2 指定内容相同;txt 文件可以缺省文件名;

7.8.1 文件基本操作

文件打开:OPEN; 可选参数 (部分):

- (1)UNIT=, 指定设备号, 设备号要有,"UNIT=" 可省略;
- (2)FILE=", 文件名;
- (3)ACCESS=, 指定存储方式, 默认'SEQUENTIAL' 顺序, 可选'APPEND' 追加存取,'DIRECT' 直接存取;
- (4)DELIM=, 指定列表的分隔符;'APOSTROPHE' 为' 分隔,'QUOTE' 用" 分隔,'NONE' 不指定分隔符;
- (5)ACTION=, 指定读写属性,'READ','WRITE'(不能读取),'READWRITE';
- (6)STATUS=, 设置文件状态属性;'OLD' 打开已存在文件,'NEW' 创建新文件,'SCRATCH' 创建临时文件,'REPLACE' 替换同名文件;

文件关闭:CLOSE; 可选参数 (部分):

- (1)UNIT=, 指定设备号, 设备号要有,"UNIT=" 可省略;
- (2)STATUS=, 设置文件状态属性;'KEEP' 临时关闭不删除,'DELETE' 关闭且删除;

文件结束:ENDFILE; 可选参数 (部分):UNIT(同上);
文件输入:READ; 可选参数 (部分):

- (1)UNIT=, 指定设备号, 设备号要有,"UNIT=" 可省略;
- (2)FMT=", 指定写入格式;

文件输出:WRITE; 可选参数 (部分) 同 READ.

文件查询:INQUIRE, 可选参数 (部分):

- (1)UNIT=, 指定设备号, 设备号要有,"UNIT=" 可省略;
- (2)FILE=", 文件名;
- (3)EXIST=", 查询设备或文件是否存在;

外部设备: 可以设置外部设备 (如并行口等) 输出信息, 以下图7.9是外部设备表, 还有一个食用示例;

```
1 OPEN(UNIT=1,FILE='PRN')
```

表 12-2 外部设备

设备名	外部设备或部件
CON	Console（终端、控制台、标准输出设备）
PRN	Printer（打印机）
COM1	Serial Port#1（串行口 1）
COM2	Serial Port#2（串行口 2）
LPT1	Parallel Port#1（并行口 1）
LPT2	Parallel Port#1（并行口 2）
AUX	Serial Port#1（串行口 1）
LINE1	Serial Port#1（串行口 1）
USER1	Standard Output（标准输出）

图 7.9: 外部设备表

7.9 接口与模块

用接口 INTERFACE 可代替 EXTERNAL 增加代码的可读性 (说明内部结构);

```
1 INTERFACE
2   FUNCTION max(a1,a2)
3     INTEGER max2,a1,a2
4   END FUNCTION
5 END INTERFACE
6 ...
7 FUNCTION max(a1,a2)
8   INTEGER max2,a1,a2
```

```
9     IF (a1>=a2)
10       max2=a1
11    ELSE
12       max=a2
13    ENDIF
14 END FUNCTION
```

定义模块 (module) 可以对函数和子程序及参数等进行封装 (大型 CONTAIN, 用 USE 载入); 其中可以通过 PUBLIC 和 PRIVATE 声明公共或私有属性 (作用域); 如下例:

```
1  MODULE uranyl
2     ...
3  END MODULE uranyl
4
5  SUBROUTINE acetate
6     ...
7     USE uranyl
8     ...
9  END SUBROUTINE
10
11 PROGRAMME zusammenbruch
12    ...
13    USE uranyl
14    CALL acetate
15    ...
16 END
```