The Madagascar User's Manual

Madagascar 使用手册
（国语版）

国强 郭俊超 黄建平 李庆洋
***等
中国石油大学
University of Petroleum, TsingTao, China
http://guoinupc.blog.163.com
2011.7.13

目 录

"You can not teach people anything, you can only find something to help others."

-Galileo Galilei（1564-1642）

前 言

自从 Claerbout 于 1991 年提出了鼓励作者将实验资料结合数据一并提供出来让其他学者重现实验内容并作分析的思想后，我们认识 Mr. Reproducible Computation 的时间已经有二十年了。此时距离 Seismic Unix（SU）的发布时间也已经过了二十四年之久。

在发布之后的若干年中，Seismic Unix 在美国科罗拉多矿院波场处理中心（CWP）不断地更新和不断地扩充中变得日趋成熟而功能强大。开发者无私捐献出的代码为勘探地球物理、地震学、软件工程等的专家和学者提供了重要的参考也带来了极大的便利，其中得到更包括石油高校、地学研究机构、石油天然气技术公司的研究人员的青睐。

在可重复性实验的思想指引下，很多学者逐渐将其发展传播，其中包括 Schwab, Karrenbach（2000）、 Stanford 大学的 Ioan Vlad（2002）、Sandy Payette 和 John Erickson（2004）等等。为了更好地适应学科的发展，计算机软件工程方面的快速更新以及更方便地提供资源和交流，Texas 大学 Austin 分校的 Sergey Fomel 于 2003 开始开发一个新的软件包 Madagascar。他依旧遵循前人的思想和研究成果，更新了开发平台，得益于 SU 和 SEPlib 等的基础，在开发者（Paul Sava，Jim Jenning，Nick Vlad 等）的努力下，Madagascar 迅速发展起来，并与 2006 年公开发布。很快，众多的用户和开发人员投入到了该软件包的应用和发展之中，短短几年时间内就得到了大范围推广。

Madagascar 推广以来。已经有很多学习资料和教程可以参考，但是繁简详略不一，（且皆为英文，）我们希望通过对这些资料的翻译、整理加上应用经验的总结归纳，推出一个可以不断修改完善的操作手册，帮助学者们更快的了解 Madagascar 的操作机制，找到适合自己的应用方法，也可以作为一个理想的参考工具，同时为其发展积极地做出自己的贡献。

国强

2012.02.07

致 谢

本文大部分资料来自于 Madagascar Main Page，我们之所以可以方便的获得这些丰富的资料并和大家共享，就是因为 S.Fomel 教授和 P.Sava 教授长久以来始终如一的勤劳工作。这是本文写作的根源。

之于 Seismic Unix 和 Madagascar 的发展历史和渊源，我几乎没什么了解，在这不再冗述，只介绍一下在本人整理材料过程来自身边的帮助和辅导。我初次接触 Madagascar 是在 2011 年，在 4 月份到 8 月份在中国石油大学（华东）物探重点实验室做毕业设计。四个月的时间，其实很短，本来很难学到什么东西，但是很幸运在 leon 工作室从很多师兄师姐那里得到了宝贵的指导和帮助，这种支持使我掌握到了很多知识，任务内的或者

任务以外的，虽然很多知识只是粗浅的了解。

初次听说 Madagascar 是因为黄建平老师，他带我们入门后，才真正开始了使用 Madagascar，并一直学习到现在。入门过程中我曾多次咨询郭俊超同学操作系统和软件的安装问题，他和黄老师都是一个耐心的帮助者。

在这期间，要感谢孔雪、马庆珍、王娟、谢金平、刘玉金、郭振波、刘思琴等人。孔雪博士始终给予详细的指导，并提供了丰富的资源；谢金平和王娟博士在正演的有关基础知识和模型建立上解释了很多疑难；马庆珍在并行处理的一些内容上帮助很大；刘玉金、郭振波、刘思琴也在很大程度上给予了帮助，包括修改程序、指导概念、提供思路等等。

本文第三章和第四章大部分原始材料的翻译工作是由郭俊超完成的，本文目前尚处于初期，还有很多空缺和错误，请大家一起指正补充。谢谢！

第 1 章 关于 Madagascar

1.1 Madagascar 简介

“An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures”(Buckheit & Donoho, 1995)

公共出版物中有很多关于计算科学的论文，这些论文本身并不能代表一门学科，它仅仅是学术的一个用以传播的表象。真正意义上的学术是一个完整的软件开发环境和能够产生有价值数据的一整套计算工具体系。

1.1.1 Madagascar 是什么

Madagascar 是一个地球物理专家开发的主要用来处理勘探地球物理数据资料或进行模型测试的软件包，它是完全开源（免费）的。我们可以在软件最新发行版本的声明下，按照我们自己所需，充分对其进行利用和开发，来完成各种工作或者是创新。其中，我们既可以只是应用其中的软件包内容，也可以做出自己的拓展。

Madagascar 提供了一个针对多维数据分析程序的集合，包括地震成像、地震数据处理和对有组织格式的数据进行处理流程实现与计算实验的综合环境。Madagascar 遵从的是 Claerbout（1992）提出的可重复性（再生性）研究的思想原理。可再生性研究需要计算实验的研究结果与软件的代码一起公开发表，如果可能还会加上输入数据，这样，文章的读者就可以重复进行实验的内容并验证实验效果（Schwab 等, 2000; Fomel 和 Claerbout, 2010）。

为什么可重复性研究如此之重要呢？是这样，只有在讨论中发现研究的结果是有疑问的时候，科学之间的相互交流才是有意义的有价值的，除非经过充足证据的证实，其他人理所当然不去采取作者的结论。作为一名在科学探讨中的吸收者，当具有了这种对发表的成果进行验证的能力，他们便可以在前人研究成果的基础上对科学技术进行开拓创新。

在地球物理研究中正是这种对于一个开放的综合的环境的需求，激励了 Madagascar 软件包的发展。它具有的可重复性研究的特点，使它在与 MATLAB、SU（Seismic Unix）等软件包类似的基础上又从中心思想上产生了巨大的差别。

接触过 SU 的人可能会很快熟悉 Madagascar 软件包本身，它也是在 Unix/Linux 操作系统上的一个扩充，其中每一个任务由一个程序来完成，也可以用管道串联上多个任务。这种分布处理具有使软件管理最小化的优点，因为不是开发一个大型应用软件来做所有事情，如 Landmark、Promax 等商业地震处理工具。

当然如果我们要做一些复杂的数据处理时，Madagascar 同样提供了有力的支持。他提供了 C、C++、Fortran 等多个常用编程语言的接口，其中 Linux 支持的 Python 语言便是 Madagascar 开发者提供给我们的标准的软件包扩充手段。Madagascar 程序可以用在

Python 脚本中来完成我们所需的任务。

如果用户需要完成一些特殊的任务，这就需要在 Madagascar 中写入新的代码，这也有一个固定的流程用以添加。同样这种扩展是 Madagascar 软件包所欢迎的。

1.1.2 Madagascar 不是什么

同 SU 一样，Madagascar 目前也没有使用功能驱动的图形用户界面，但是作为一种非商业性质的完全开源的软件包，我们也并不希望在这方面拿它与昂贵的大型处理软件相比较。其中，原因是多方面的。

首先，Madagascar 是建立在 Linux 操作系统上的（将来或许会推出 Windows 等版本），和 Linux 一样，不能通过图形界面来完成各项任务，我们很清楚这样做的原因。

其次，Madagascar 并不是一个大型商业化地震处理软件的替代品或者简化品。商业化的地震资料处理软件用来应用现阶段较为成熟工业技术或处理任务，如果需要作如此商业化的处理任务，大可不必考虑 Madagascar。

然后，商业化的软件通常提供给用户一个方便且亲切的使用环境，让工作人员不必花太多时间和精力用在理解算法和代码构成或者开发新的功能。既然 Madagascar 并不是真正面向商业大规模资料处理，作为一个重要的助手，它可以用在教育科研领域，或者给应用者提供一个规范而有很大参考价值的基础源代码。Madagascar 鼓励用户在此原型软件包的基础上做出自己的尝试和开发试验，即不去过多的考虑软件设计和使用层面的内容，而把主要精力放在处理功能本身。

Madagascar 并不只限于地震资料处理应用。它可以应用在通用的信号处理和一些数学分析中。正如上面提到的，在科教领域有很大的应用前景，还包括很多非地震波动资料、图像资料处理等。

1.2 Madagascar 历史

Madagascar 软件包（早期命名为 RSF，即 Regularly Sampled Format 规则采样格式）的开发工作开始于 2003 年。2006 年 6 月在维也纳欧洲地质学家与工程师学会召开了一个研讨会，会议是关于开源的 E&P（勘探与开发）软件，在此期间这个软件包公开发布了出来。之后，超过 25 个专家学者都为这个软件包的发展作出了很大贡献，累计编写了约 300000 行的程序代码。

Madagascar 规划的开源软件发展前景，被定义成是：

成熟，完善的代码库；

逐年累加的开发工作；

庞大，活跃的开发团队；

这些都使 Madagascar 迅速扩大进步，发展成为地球物理和信号处理社区中重要的一员。

year	location	name
2006	Vancouver, Canada	School and Workshop on Reproducible Research in Computational Geophysics
2007	Austin, TX	Short Course on Using and Extending Madagascar
2008	Golden, CO	Implementation Workshop
2009	Delft, the Netherlands	School on Reproducible Computational Geophysics
2010	Houston, TX	School on Reproducible Computational Geophysics and Hands-On Workshop

表 1：Madagascar 年度事件表

1.3 软件的获取与安装

http://guoinupc.blog.163.com/blog/static/186039199201148101665 40/

第 2 章 帮助信息

2.1 程序列表

截止到 2001 年 7 月 15 日，Madagascar 统计到的数据，已有 812 个程序，30 多个开发成员，开发语言包括 C、C++、Fortran、Python，应用涵盖数值分析、通用数据分析、地震资料处理、图形图像及可视化。

Madagascar 中所有应用程序都含有“sf”前缀。这么多的程序我们如何有效地对其进行搜索，以找到我们所需要的程序呢？显然，我们可以像 SU 一样，找到一份用户使用手册随时查找或者凭借你超强的记忆力。这里，我们给出 Madagascar 常用的查看方式：

首先是在 Linux 系统 terminal 下输入：

sfdoc -k .

这样终端会显示 Madagascar 已安装的所有程序列表，找到看上去可能是你所需要的程序指令名称，具体内容下面会讲到。

```
bash $ sfdoc -k .
sfwave : Rice HPCSS seismic modeling and migration .
sfofpwd : Objective function of dip estimation with PWD filters .
sferf : Bandpass filtering using erf function .
sfinfill : Shot interpolation .
sfgbeamform : 2-D lateral smoothing .
...
```

也可以通过 Madagascar 网站的方式进行查找，我们最好熟悉这种方式：

http://www.ahay.org/RSF

2.2 查找需要的程序

Madagascar 也提供了通过我们知道的关键字查找命令的方式，这是很有实用价值的，关键字可能包含在命令名称中，也可能包含在命令说明中。

在终端中输入：

bash $ sfdoc -k interpolation

见下表。

```
bash $ sfdoc -k interpolation
sfinfill : Shot interpolation .
sflevint : Leveler inverse interpolation in 1-D.
sfmsmiss : Multiscale missing data interpolation (N- dimensional ).
sffreqreg : Local frequency interpolation .
...
```

（来源：XuxinMa）

当然，如果不是第一次接触 Madagascar，你只是记不清楚命令的全称，我们可以只

输入命令的前几个英文字母，然后敲两下【Tab】键，终端上就会显示出所有以此为开头的命令。这个过程就如同你在 Linux 操作系统下查找某个文件夹。是不是很方便，如果你还是觉得难以精确查找的话，请不要急，下面的使用说明会慢慢帮助你找到一个自己习惯的查询方式。

2. 3 程序详细信息

这一部分主要针对程序的具体帮助信息作介绍，这和 SU 软件包是基本相同的，如果你用过 SU 的话，可以简要浏览一下或者直接跳过这部分。

2. 3. 1 程序功能介绍

知道了如何查找命令，那如何如确定找到的命令是不是自己想要的呢，或者若干个命令中哪个才是最合适的，我们需要该命令的详细信息。

键入命令，不加任何参数，我们可以得到程序的自述文件：

sfprog（without arguments）

```
bash $ sfwindow
NAME
sfwindow
DESCRIPTION
Window a portion of a dataset .
SYNOPSIS
sfwindow < in. rsf > out . rsf verb =n squeeze =y j #=(1 ,...) d #=(
d1 ,d2 ,...) f #=(0 ,...) min #=( o1 ,o2 , ,...) n #=(0 ,...) max #=(
o1 +(n1 -1) *d1 ,o2 +(n1 -1) *d2 , ,...)
PARAMETERS
float d #=( d1 ,d2 ,...) sampling in #-th dimension
largeint f #=(0 ,...) window start in #-th dimension
int j #=(1 ,...) jump in #-th dimension
float max #=( o1 +(n1 -1) *d1 ,o2 +(n1 -1) *d2 , ,...) maximum in #-
th dimension
float min #=( o1 ,o2 , ,...) minimum in #-th dimension
largeint n #=(0 ,...) window size in #-th dimension
bool squeeze =y [y/n] if y, squeeze dimensions equal to 1
to the end
...
USED IN
bei / dpmv / krchdmo
bei / dpmv / matt
bei / dwnc / sigmoid
```

```
...
SOURCE
system / main / window .c
DOCUMENTATION
http :// ahay . org / wiki / Guide_to_madagascar_programs # sfwindow
```

其中，DESCRIPTION 下的是程序功能的描述，说明它能够完成的任务，该命令做的是对数据进行取样。SYNOPSIS 下的是该程序的使用方式（我们将在第 3 章命令行使用中做详细介绍），这里是定义一个输入文件 in.rsf 和一个输出文件 out.rsf，然后输入各关键参数 verb、squeeze 等，输入完成后敲 Enter 就执行了。PARAMETERS 是对该程序的关键参数做的说明，如“float d #=(d1 ,d2 ,...) sampling in #-th dimension”中，float 是参数的格式，#表示要替换成所需的值（这里是坐标轴序号），后面描述此为在第#个坐标轴上的采样，像“bool squeeze =y [y/n] if y, squeeze dimensions equal to 1”中，squeez 等号后面有一个变量值 y，则表示不输入变量值时的默认值。

2. 3. 2 程序使用实例

在 USED IN 下，便是该程序（sfwindow）在被开发人员应用的实际例子，我们根据它可以打开 “$RSFSRC / book / bei / dpmv / krchdmo /”下的 SConstruct 文件，$RSFSRC 是安装 Madagascar 软件包时自己定义的源程序的位置，book 中有很多用户的使用案例和源代码。SConstruct 是一个 Python 脚本文件，用 scons 执行（将在第 6 章 Python 编程中详细介绍）。

```
bash $ cat $RSFSRC / book / bei / dpmv / krchdmo / SConstruct
...
def vscan ( title ):
return ’’’
window f3 =174 n3 =1 |
vscan v0 =%g nv =100 dv =%g half =n slowness =y |
grey title ="% s"
’’’ % (1/2.8 ,(1/1.7 -1/2.8) /99 , title )
...
```

在案例中，用的是采样 window 的程序，只有作为命令时才加上“sf”前缀，这里请初学者注意。我们继续刚才对于程序自述文件的介绍，SOURCE 是源代码的位置，我们可以找到该命令用 C 语言编写的源代码。最后可以链接网址“http :// ahay . org / wiki / Guide_to_madagascar_programs # sfwindow”，也可查看命令 sfwindow 的详细信息。

第 3 章 命令行使用说明

3.1 软件指令使用

3.1.1 单一指令

3.1.2 多指令

3.2 RSF 文件格式指导（Guide to RSF file format）

3.2.1 原则（principle）

RSF 数据格式的主要设计原则是 KISS（Keep It Short and Simple）。RSF 数据格式参考了当初为斯坦福勘探项目组设计的 SEPlib 数据格式（Claerbout，1991[1]）。这种格式为了追求最大的方便、透明度和灵活性而设计。根据 Unix 习惯，常用文档应该为可读的文本格式以便于用通用工具检查和处理。Raymond（2004[2]）写道：

如果你与 Unix 文件有仇，就将所有文件格式用二进制并且很难懂的形式表示，并使用超强的工具来读写它们。

如果你觉得必须用复杂的二进制文件类型或者应用协议，那么最明智的做法是停下来等到这种感觉消失。

用文本格式存储大量数据可能不是很经济。RSF 使用了另一个秘诀：它允许数值用二进制格式存储，但是将所有数据属性用文本文件表示，以便人们就可以读懂并用通用的文本处理软件进行编辑。

举例（example）：

首先创建一个合成的 RSF 数据文件：

```
bash$ sfmath n1=1000 output='sin(0.5*x1)' > sin.rsf
```

打开并读入文件 sin.rsf。

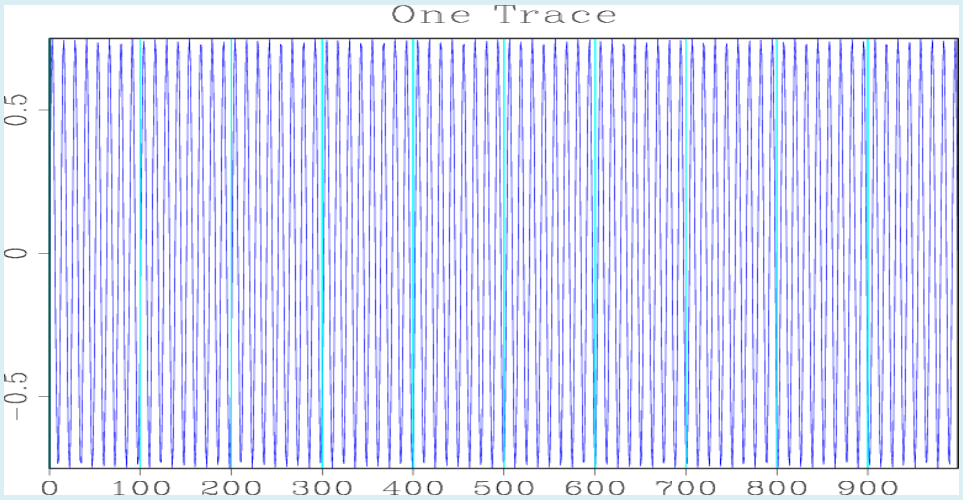
```
bash$ cat sin.rsf
sfmath  rsf/rsf/rsftour:        fomels@egl      Sun Jul 31 07:18:48
2005


        o1=0
        data_format="native_float"
        esize=4
        in="/tmp/sin.rsf@"
        x1=0
        d1=1
        n1=1000
```

文件中含有 9 行可读的文本。第一行显示了程序的名称、工作路径、文件的作者和计算机以及文件的创建时间（这些信息用于注释）。其它行含有用“=”号分开的参数—值。“in”参数指向二进制问数据。在我们详细地讨论参数的意义之前，先对这些数据进行绘图：

```
bash$ < sin.rsf  sfwiggle title='One Trace' | sfpen
```

在你的屏幕上会显示出与下图类似的图片：

正弦函数绘图



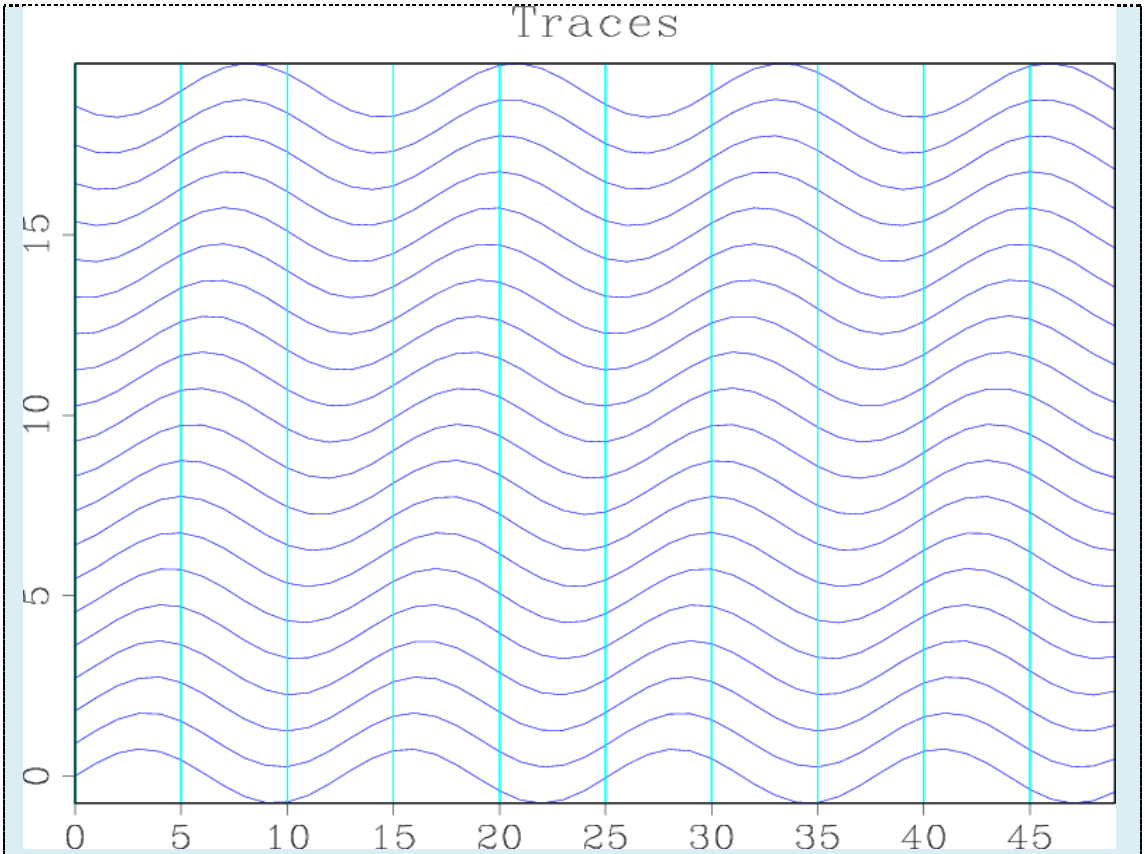
假设你想要对一个含有 1 道 1000 个采样点的数据的格式改写成 20 道含有 50 个采样点的数据，运行：，

```
bash$ < sin.rsf sed 's/n1=1000/n1=50 n2=20/' > sin10.rsf
bash$ < sin10.rsf sfwiggle title=Traces | sfpen
```

或者使用管道（pipes）：

```
bash$ < sin.rsf sed 's/n1=1000/n1=50 n2=20/' | sfwiggle title=Traces | sfpen
```

你会得到一张与下图类似的图：



将数据格式变为 20 道后的正弦波形

结果怎样？我们使用 sed——一个标准的 Unix 编辑器来改写描述数据维数的参数。由于这种操作非常简单，就没有必要创建特殊格式的工具来使 sfwiggle 程序接收其它的格式参数。还可以用其它的通用参数如 cat，echo，grep，等对 RSF 文件进行操作。获得上述结果的另一种方法是运行：

```
bash$ ( cat sin.rsf; echo n1=50 n2=20 ) > sin10.rsf
bash$ < sin10.rsf sfwiggle title=Traces | sfpen
```

此时，cut 命令将前一个文件的内容复制，echo 命令将新的命令行“n1=50 n2=20”加入，新的 n1 参数覆盖了原来 iden1=1000，这样我们得到了与上面相同的结果。当然，你也可以使用常用的文本编辑器对文件进行手动的修改。为了记录数据处理的过程，通常使用非交互式的工具对文件进行处理。

3.2.2 头文件和数据文件（Header and Data files）

检查 RSF 文件输出的一个简单的方法是使用 sfin 程序：

```
bash$ sfin sin10.rsf
sin10.rsf:
```

```
in="/tmp/sin.rsf@"
esize=4 type=float form=native
n1=50           d1=1            o1=0
n2=20           d2=?            o2=?
        1000 elements 4000 bytes
```

这个程序报告了下列信息：数据文件（/tmp/sin.rsf\@）的位置，元素大小（4 个字节），元素类型（实型），元素形式（native），数据体的维数（50×20），轴的拉伸（1 和未确定的），轴的起始点（0 和没有确定的）。同时也报告了元素的总数和总字节数。下面我们来详细地检查这些信息。首先来验证数据文件存在并含有特定字节数：

```
bash$ ls -l /tmp/sin.rsf@
-rw-r--r--    1 sergey users 4000 2004-10-04 00:35 /tmp/sin.rsf@
```

用二进制形式存储 50×20 的 4 个字节的数据需要 4000 字节的空间。因此，数据文件只是将纯数据用二进制格式存储起来，没有其它的数据。

3.2.2.1**Datapath**

RSF 程序（sfmath）怎么知道将数据文件放到哪里呢？按照优先级的顺序，选择数据文件名及路径的规则如下：

检查命令行中 out=参数，它明确地指出了输出数据文件的位置。

分别确定路径和文件名。

选择路径的规则是：

检查命令行中的 datapath=参数，它提供了字符串作为预选文件名，这个字符串可能含有文件*路径*。

检查 DATAPATH 环境变量。它与用 datapath=确定的参数值具有相同的意义。

检查当前路径下的.datapath 文件。文件可能包含命令行

```
datapath=/path/to_file/
```

或如果你想在不同的平台上使用不同的路径时用

```
machine_name datapath=/path/to_file/
```

检查用户根目录下的.datapath 文件。

将数据文件放到当前路径下（与 datapath=./）类似。

文件名的选择规则：

如果输出的 RSF 文件在当前路径下，通过加\@为数据文件命名。

如果输出文件不在当前路径下或者是由程序临时创建的，通过在程序名上随机添加字符来作为文件名。

举例：

```
bash$ sfspike n1=10 out=test1 > spike.rsf
```

```
bash$ grep in spike.rsf
in="test1"
```

```
bash$ sfspike n1=10 datapath=/tmp/ > spike.rsf
bash$ grep in spike.rsf
in="/tmp/spike.rsf@"
```

```
bash$ DATAPATH=/tmp/ sfspike n1=10 > spike.rsf
bash$ grep in spike.rsf
in="/tmp/spike.rsf@"
```

```
bash$ sfspike n1=10 datapath=/tmp/ > /tmp/spike.rsf
bash$ grep in /tmp/spike.rsf
in="/tmp/sfspikejcARVf"
```

Grep 是什么

```

```

Packing header and data together

将头文件与数据文件合并在一起

虽然默认头文件与数据文件是分开的，实际上也可以将它们合并为一个文件。可以通过给程序“out”确定参数值即 out=stdout 来实现。例如：

```
bash$ sfspike n1=10 out=stdout > spike.rsf
bash$ grep in spike.rsf
Binary file spike.rsf matches
bash$ sfin spike.rsf
spike.rsf:
    in="stdin"
    esize=4 type=float form=native
    n1=10               d1=0.004          o1=0              label1="Time"
unit1="s"
        10 elements 40 bytes
bash$ ls -l spike.rsf
-rw-r--r--  1 sergey users 196 2004-11-10 21:39 spike.rsf
```

检查 spike.rsf 文件的内容，你会发现它是以文本格式的头文件开始的，后面是一些特定的符号，在后面四二进制的数据。将头文件与数据合并在一起可能不便于数据处理，但是对于存储数据很方便，同时也是用 Unix 管道处理 RSF 文件所需要的。

Type

类型

RSF 文件中的数据可以有不同的类型：字符型，不加符号字符型，整型，实型，复数型。默认的是单精度的数据（C 语言中的整型和实型）。需要显示这些数所用的字节数取决于运行平台。

Form

格式

RSF 存储的数据还可以用其它格式：ASCII，native 二进制以及 XDR 二进制，默认的是 Native 二进制。在 Linux 系统中，native 二进制与小端字节顺序相对应。在其它平台上可能是大端字节顺序。XDR 是 Sun 公司设计的用于网络传输的二进制格式。用 native 二进制格式处理 RSF 文件效率更高，但是如果你想在不同的平台下使用数据，最好用 XDR 二进制格式存储相应文件。RSF 也支持 ASCII（纯文本）数据文件。用 sfdd 程序可以进行不同的类型和格式之间的转换。下面是一些例子。首先，合成数据：

```
bash$ sfmath n1=10 output='10*sin(0.5*x1)' > sin.rsf
bash$ sfin sin.rsf
sin.rsf:
    in="/tmp/sin.rsf@"
    esize=4 type=float form=native
    n1=10               d1=1               o1=0
        10 elements 40 bytes
bash$ < sin.rsf sfdisfil
  0:                0          4.794          8.415          9.975          9.093
  5:            5.985          1.411         -3.508         -7.568         -9.775
```

将数据转换为整型：

```
bash$ < sin.rsf sfdd type=int > isin.rsf
bash$ sfin isin.rsf
isin.rsf:
    in="/tmp/isin.rsf@"
    esize=4 type=int form=native
    n1=10               d1=1               o1=0
        10 elements 40 bytes
bash$ < isin.rsf sfdisfil
  0:      0      4      8      9      9      5      1     -3     -7     -9
```

将数据转换为 ASCII 格式：

```
bash$ < sin.rsf sfdd form=ascii > asin.rsf
```

```
bash$ < asin.rsf sfdisfil
    0:              0          4.794          8.415          9.975          9.093
    5:          5.985          1.411         -3.508         -7.568         -9.775
bash$ sfin asin.rsf
asin.rsf:
      in="/tmp/asin.rsf@"
      esize=0 type=float form=ascii
      n1=10              d1=1              o1=0
            10 elements
bash$ cat /tmp/asin.rsf@
0 4.79426 8.41471 9.97495 9.09297 5.98472 1.4112 -3.50783
-7.56803 -9.7753
```

Hypercube 高维数据体

尽管 RSF 以 1 维形式存储二进制文件，但从概念上数据模型是一个高维数据体。通常，数据体的维数用 n1,n2,n3 等表示。n1 是最快的轴。另外，网格采样点可以以 d1,d2,d3 等参数的形式给出。轴的起始点用从参数 o1,02,03 等参数给出。除此之外，你还可以给轴用字符串 label1,label2,label3 等加上标签及用 unit1,unit2,unit3 等字符串加上单位。

Compatibility with other file formats

3. 2. 3 与其它文件格式的兼容性

RSF 格式的数据可以与其它常用的数据格式转换。

Compatibility with SEPlib

3.2.3.1 与 SEPlib 的兼容性

RSF 与它的前辈——SPElib 文件格式是最兼容的。格式是最兼容的。格式是最兼容的，但是两者也有一些区别：

SEPlib 程序用元素的大小（esize=参数）来区分不同数据类型：esize=4 对应实型，esize=8 对应复数类型。RSF 对数据类型的处理机制也不同：用 data_format 参数来确定数据类型。Madagascar 的计算程序用 data——format=”native_float”或 native_complex。

SEPlib 程序的默认数据格式是 XDR，RSF 默认到的数据格式是 native。因此，为了用 SEPlib 创建一个 Madagascar 程序可读的数据集，你需要在 history 文件中加入 data_format=”xdr_float”或 data_format=”xdr_complex”。

可以用管道将 Madagascar 程序的输出文件送到 SEPlib：

```
bash$ sfspike n1=1 | Attr want=min
```

（输出文件应该用：在 1 处 minimum value = 1），但是将 SEPlib 程序的输出文件安用管

道送到 RSF（或者其它非 SEPlib 程序）时会产生无终止的进程。例如，以下命令无法正常运行。

```
bash$ Spike n1=1 | sfattr want=min
```

这是因为 SEPlib 为管道传输提供了接口，在 Madagascar 通过常规的 Unix 管道传送数据时需要在接收程序上有相应的接口。

SEP3D 是 SEPlib 的扩展，用来对非常规采样的数据（Biondi et al., 1996[3]）进行处理。由于文档开头所述的原因，在 RSF 中没有相应的程序。对非常规数据集的操作要用到辅助文件中表示几何形状的信息。

Notes

标注

1. 对于 SEPlib 6.5.3 以及更老的版本：注意 xdr_complex 不是有效的 SEPlib 值，因而对于用实型数据对组成的复数型数据集无法同时在 SEPlib 和 Madagascar 中使用。有效的 SEPlib 数据集的 esize=8 并且 data_format="xdr_float"，但是 sfin 会显示为期望数据的 200%。给这个数据集加上 data_format="xdr_complex"可以让 sfin 正常工作，但是 SEPlib 的 ln 或 ln3 会由于未知的数据类型给出码段错误的信息。为了是 SEPlib 识别 native_complex 与 xdr_complex 数据，需要进行以下改进：

$SEPSRC/seplib_base/lib/corelibs/sep/strformats.c 中:

在 str_fmt_names 结构体中加入 "xdr_complex" and "native_complex"

令 FMT_LENGTH 等于 15

$SEPSRC/seplib_base/lib/corelibs/include/strformats.h 中:

加入预处理指令，令 FMT_XDR_COMPLEX 为 8 and FMT_NATIVE_COMPLEX 为 9

令 NUM_FMT 等于 10

3.2.3.2Reading and writing SEG-Y and SU files

读写 SEG-Y 和 SU 文件

SEG-Y 格式是根据 Barry 等（1975[4]）的建议制定的。在 2002[5]年被重新修订。SU 格式是 Seismic Unix（Stockwell, 1997[6]）中使用的对 SEG-Y 的改进。使用 sfsegyread 可以将 SEG-Y 或 SU 格式转换为 RSF 格式。我们首先使用 SU 程序制作一个文件（Stockwell, 1999[7]）作为例子：

```
bash$ suplane > plane.su
bash$ segyhdrs < plane.su | segywrite tape=plane.segy
```

将其转换为 RSSF 文件，使用

```
bash$ sfsuread < plane.su tfile=tfile.rsf endian=0 > plane.rsf
```

或者

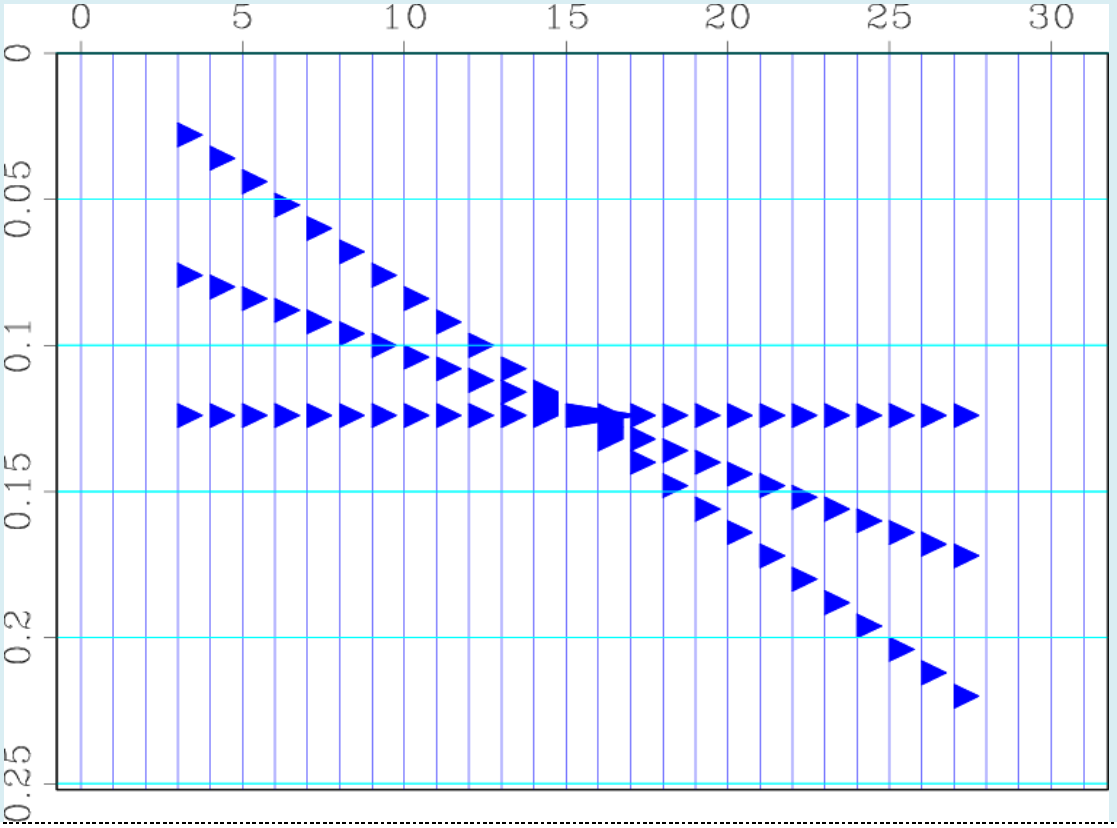
```
bash$ sfsegyread < plane.segy tfile=tfile.rsf \
hfile=hfile bfile=bfile endian=0 > plane.rsf
```

如果 SU 文件是由小端字节的机器如 Linux 计算机创建的，就需要用到字节的大小端标志。标准的输出将 RSF 文件与数据（32 道，每道 64 个采样点）同时输出：

```
bash$ sfin plane.rsf
plane.rsf:
        in="/tmp/plane.rsf@"
        esize=4 type=float form=native
        n1=64                d1=0.004          o1=0
        n2=32                d2=?              o2=?
                2048 elements 8192 bytes
```

这个文件的图形为：

Suplane 的输出，转换为 RSF 后用 sfwiggle 绘制



Tfile 是一个含有道头（**32 个头文件 71 道 32headers with 71 traces each**）的 RSF 整型文件：71 行 32 列

```
bash$ sfin tfile.rsf
tfile.rsf:
        in="/tmp/tfile.rsf@"
        esize=4 type=int form=native
```

```
    n1=71           d1=?            o1=?
    n2=32           d2=?            o2=?
         2272 elements 9088 bytes
```

道头的内容可以用 sfheaderattr 程序来快速查看。hfile 是整个记录的 ASCII 道头文件。

```
bash$ head -c 242 hfile
C       This tape was made at the
C
C       Center for Wave Phenomena
```

bfile 是二进制的道头文件：

要想将 RSF 文件转换回 SEG-Y 或 SU 格式，运行 sfsegywrite 程序，并将输入和输出反过来：

```
bash$ sfsuwrite > plane.su su=y tfile=tfile.rsf endian=0 < plane.rsf
```

或者

```
bash$ sfsegywrite > plane.segy tfile=tfile.rsf \
hfile=hfile bfile=bfile endian=0 < plane.rsf
```

如果 sfsegywrite 中没有 hfile=和 bfile=，相应的道头可以从默认的位置（名为 header 和 binary 的文件）取或者运行时生成。道头文件可以用 sfsegyheader 程序生成。下面是个例子：

```
bash$ rm header binary
bash$ sfheadermath < plane.rsf output=N+1 | sfdd type=int > tracl.rsf
bash$     sfsegyheader     <     plane.rsf     tracl=tracl.rsf     >
tfile.rsfbash$ sfsegywrite  < plane.rsf tfile=tfile.rsf > plane.segy
```

3.2.3.3Reading and writing ASCII files

读写 ASCII 码文件

可以用 sfdd 程序读写 ASCII 文件。例如，用一个含有数字的 ASCII 文件

```
bash$ cat file.asc
1.0 1.5 3.0
4.8 9.1 7.3
```

将它转换为 RSF 文件

```
bash$ echo in=file.asc n1=3 n2=2 data_format=ascii_float > file.rsf
bash$ sfin file.rsf
file.rsf:
    in="file.asc"
```

```
    esize=0 type=float form=ascii
    n1=3            d1=?            o1=?
    n2=2            d2=?            o2=?
        6 elements
```

为了使输入/输出操作更高效，将数据转换为 native 二进制，如下所示：

```
bash$ echo in=file.asc n1=3 n2=2 data_format=ascii_float | \
sfdd form=native > file.rsf
bash$ sfin file.rsf
file.rsf:
    in="/tmp/file.rsf@"
    esize=4 type=float form=native
    n1=3            d1=?            o1=?
    n2=2            d2=?            o2=?
        6 elements 24 bytes
```

可以使用 sfdd 中的 line=和 format=参数来控制 ASCII 格式

```
bash$ sfdd form=ascii out=file.asc \
line=3 format="%3.1f " < file.rsf > /dev/null
bash$ cat file.asc
1.0 1.5 3.0
4.8 9.1 7.3
```

另一种方式是使用 sfdisfil。

```
bash$ sfdisfil > file.asc col=3 format="%3.1f " number=n < file.rsf
bash$ cat file.asc
1.0 1.5 3.0
4.8 9.1 7.3
```

3.2.4 其它文档（other documents）

这部分大体介绍了 RSF 文件格式。如果你对细节有疑问，参考 RSF 综合介绍。其它的相关文档有：

Madagascar 介绍
为什么使用 Madagascar
安装指导
Madagascar 文档
Madagascar 程序指导说明
Madagascar 编程接口指导说明
Madagascar 编程指导

使用 Madagascar 与 Scons 重新运行 SEP 程序
用 Scons 进行可移植计算实验 3.2.5 关于这个文档
About this document

本文档是使用 LaTeX 源 book/rsf/rsf/format.tex 中的 latex2wiki 创建的。

3.3 画图

3.4 错误信息

第 4 章 核心指令导航

4.1 常规处理

Madagascar 程序指导

本页是用 latex2wiki 将 book/rsf/prog.tex 中的 LaTex 源文件转化得到的

本指导书介绍了一些最常用的 madagascar 程序并用实例展示了它们的使用方法。

内容

4. 1. 1 主程序

这些程序的源文件可以从 Madagascar 发行版的文件夹 system/main 中找到。主程序对 RSF 超立方体进行常规的操作而不关心数据的维数或物理维数。

sfadd

对 RSF 数据体进行相加、相乘或相除
sfadd > out.rsf scale= add= sqrt= abs= log= exp= mode= [< file0.rsf] file1.rsf file2.rsf ...
各项操作一经选定，它们的顺序为： 取绝对值，abs= 增加一个标量，add= 取自然对数，log= 取均方根，sqrt= 与标量相乘，scale= 计算以 e 为底的指数，exp= 数据体相加、乘、除，mode=

Sfadd 的操作对象是整形、实型数据或复数，但是要求输入和输出的数据类型是一致的。可以用 sfmath 对 sfadd 进行替换，它用起来更灵活，但效率降低。		
bools 布尔型	abs=	如果为真,取绝对值[nin]
实型	add=	将标量夹道每个数据集上[nin]
布尔型	exp=	如果为真,计算指数[nin]
布尔型	mode=	‘a’意味着加(默认) ‘p’或’m’意味着乘[nin] ‘d’意味着除[nin]
实型	scale=	标量与每个数据集相乘[nin]
布尔型	sqrt=	如果为真,取平方根[nin]

sfadd 可以将多个数据集 结合起来（加，除，乘）。如果你想让两个数据集相减怎么办呢？很简单，使用如下尺度参数：

bash$ sfadd data1.rsf data2.rsf scale=1,-1 > diff.rsf

或

bash$ sfadd < data1.rsf data2.rsf scale=1,-1 > diff.rsf

可以用程序 sfmath 完成相同的任务：

bash$ sfmath one=data1.rsf two=data2.rsf output='one-two' > diff.rsf

或

bash$ sfmath < data1.rsf two=data2.rsf output='input-two' > diff.rsf

两种情况下，多维体 data1.rsf 和 data2.rsf 的形状应该相同，如果轴抽样参数（如 o1 或 d1）不同，则会显示警告信息。

程序实现：system/main/add.c

第一个输入文件要么在列表中，要么是标准的输入。

```
/* find number of input files */
    if (isatty(fileno(stdin))) {
        /* no input file in stdin */
        nin=0;
    } else {
        in[0] = sf_input("in");
        nin=1;
    }
```

从命令行参数数列中找到不包含“=”号的输入文件，nin 是输入文件的总数。

```
for (i=1; i< argc; i++) { /* collect inputs */
        if (NULL != strchr(argv[i],'=')) continue;
        in[nin] = sf_input(argv[i]);
        nin++;
    }
    if (0==nin) sf_error ("no input");
    /* nin = no of input files*/
```

一个辅助函数检查输入文件的兼容性

```
static void
check_compat (sf_datatype type /* data type */,
              size_t       nin   /* number of files */,
              sf_file*     in    /* input files [nin] */,
              int          dim   /* file dimensionality */,
              const int*   n     /* dimensions [dim] */)
/* Check that the input files are compatible.
   Issue error for type mismatch or size mismatch.
   Issue warning for grid parameters mismatch. */
{
    int ni, id;
    size_t i;
    float d, di, o, oi;
    char key[3];
    const float tol=1.e-5; /* tolerance for comparison */

    for (i=1; i < nin; i++) {
        if (sf_gettype(in[i]) != type)
            sf_error ("type mismatch: need %d",type);
        for (id=1; id <= dim; id++) {
            (void) snprintf(key,3,"n%d",id);
            if (!sf_histint(in[i],key,&ni) || ni != n[id-1])
                sf_error("%s mismatch: need %d",key,n[id-1]);
            (void) snprintf(key,3,"d%d",id);
            if (sf_histfloat(in[0],key,&d)) {
```

```
                    if (!sf_histfloat(in[i],key,&di) ||
                        (fabsf(di-d) > tol*fabsf(d)))
                        sf_warning("%s mismatch: need %g",key,d);
                } else {
                    d = 1.;
                }
                (void) snprintf(key,3,"o%d",id);
                if (sf_histfloat(in[0],key,&o) &&
                    (!sf_histfloat(in[i],key,&oi) ||
                     (fabsf(oi-o) > tol*fabsf(d))))
                    sf_warning("%s mismatch: need %g",key,o);
            }
        }
}
```

最后，进入主循环中，输入数据是由缓存一个个读入的并按数据类型加入到总的结果中。

```
for (nbuf /= sf_esize(in[0]); nsiz > 0; nsiz -= nbuf) {
        if (nbuf > nsiz) nbuf=nsiz;

        for (j=0; j < nin; j++) {
            collect = (bool) (j != 0);
            switch(type) {
                case SF_FLOAT:
                    sf_floatread((float*) bufi,
                                 nbuf,
                                 in[j]);
                    add_float(collect,
                              nbuf,
                              (float*) buf,
                              (const float*) bufi,
                              cmode,
                              scale[j],
                              add[j],
                              abs_flag[j],
```

```
                                        log_flag[j],
                                        sqrt_flag[j],
                                        exp_flag[j]);
                    break;
```

实行数据的结合程序是 add_float。

```
static void add_float (bool     collect,      /* if collect */
                       size_t nbuf,           /* buffer size */
                       float* buf,            /* output [nbuf] */
                       const float* bufi, /* input    [nbuf] */
                       char     cmode,        /* operation */
                       float    scale,        /* scale factor */
                       float    add,          /* add factor */
                       bool     abs_flag,     /* if abs */
                       bool     log_flag,     /* if log */
                       bool     sqrt_flag,    /* if sqrt */
                       bool     exp_flag      /* if exp */)
/* Add floating point numbers */
{
    size_t j;
    float f;

    for (j=0; j < nbuf; j++) {
        f = bufi[j];
        if (abs_flag)       f = fabsf(f);
        f += add;
        if (log_flag)       f = logf(f);
        if (sqrt_flag)      f = sqrtf(f);
        if (1. != scale) f *= scale;
        if (exp_flag)       f = expf(f);
        if (collect) {
            switch (cmode) {
                case 'p': /* product */
                case 'm': /* multiply */
                    buf[j] *= f;
                    break;
```

```
                    case 'd': /* delete */
                        if (f != 0.) buf[j] /= f;
                        break;
                    default:   /* add */
                        buf[j] += f;
                        break;
                }
            } else {
                buf[j] = f;
            }
        }
}
```

sfattr

显示数据体属性
sfattr < in.rsf want=

用如下命令得到样本数据 "sfspike n1=100 | sfbandpass fhi=60 | sfattr"

****

均方根 = 0.992354

平均值 = 0.987576

2 范数 = 9.92354

方差 = 0.00955481

标准差 = 0.0977487

极大值 = 1.12735 at 97

极小值 = 0.151392 at 100

非零采样点个数 = 100

总采样点的个数 = 100

****

均方根 = sqrt[sum(data^2) / n]

平均值 = sum(data) / n

范数= sum(abs(data)^lval)^(1/lval)

方差 = [sum(data^2) - n*mean^2] / [n-1]

标准差 = sqrt [variance]

int	**lval=2**	选择范数, lval 是一非负整数, 计算向量 lval-norm
string	**want=**	'all'(默认),'rms','mean','norm','var','std','max' ,'min','nonzero','samples','short' want= 'rms' 显示均方根 want= 'norm' 显示 2 范数, 否则显示 lval 指定的范数. want= 'var' 显示方差 want= 'std' 显示标准差 want= 'nonzero' 显示非零采样点的个数 want= 'samples' 显示总采样点的个数 want= 'short' 显示一行

sfattr 是个有用的诊断程序。它给出了特定 RSF 数据集中一定的统计值：均方根振幅、平均值、向量范数、方差、标准差，最大值和最小值、非零采样值个数、总采样点数。如果我们定义值为 di，其中 i=0，1，2，…n,因此均方根是 $\sqrt{\frac{1}{n}\sum_{i=0}^{n} d_i^2}$，平均值为 $\frac{1}{n}\sum_{i=0}^{n} d_i$，二范数为 $\sqrt{\sum_{i=0}^{n} d_i^2}$，方差为 $\frac{1}{n-1}[\sum_{i=0}^{n} d_i^2 - \frac{1}{n}(\sum_{i=0}^{n} d_i)^2]$，标准差是方差的开方。使用 sfattr 是观察数据分布和检测异常值的快速方法。输出值可以通过使用 awk 解析，来提取一个竖直来作为命令行界面的输入参数，注意下例中括号：

```
sfgrey <vel.rsf allpos=y bias=`sfattr <vel.rsf want=min | awk '{print $4}'` | sfpen
```

程序实现：system/main/attr.c

计算从找到输入数据（in）的形状（nsiz）和维数（dim）。

```
dim = (size_t) sf_filedims (in,n);
    for (nsiz=1, i=0; i < dim; i++) {
    nsiz *= n[i];
    }
```

在主循环中，我们将输入数据逐个读入缓存

```
for (nleft=nsiz; nleft > 0; nleft -= nbuf) {
        nbuf = (bufsiz < nleft)? bufsiz: nleft;
        switch (type) {
            case SF_FLOAT:
                sf_floatread((float*) buf,nbuf,in);
                break;
            case SF_INT:
                sf_intread((int*) buf,nbuf,in);
                break;
            case SF_COMPLEX:
```

```
                sf_complexread((sf_complex*) buf,nbuf,in);
                break;
            case SF_UCHAR:
                sf_ucharread((unsigned char*) buf,nbuf,in);
                break;
            case SF_CHAR:
            default:
                sf_charread(buf,nbuf,in);
                break;
    }
```

数据属性相对于双精度的变量增加

```
        fsum += f;
        fsqr += f*f;
```

最后，属性值减小并输出

```
fmean = fsum/nsiz;
    if (lval==2)        fnorm = sqrt(fsqr);
    else if (lval==0) fnorm = nsiz-nzero;
    else                fnorm = pow(flval,1./lval);
    frms = sqrt(fsqr/nsiz);
    if (nsiz > 1) fvar = (fsqr-nsiz*fmean*fmean)/(nsiz-1);
    else            fvar = 0.0;
    fstd = sqrt(fvar);
```

```
if(NULL==want || 0==strcmp(want,"rms"))
        printf("rms = %g \n",(float) frms);
    if(NULL==want || 0==strcmp(want,"mean"))
        printf("mean value = %g \n",(float) fmean);
    if(NULL==want || 0==strcmp(want,"norm"))
        printf("%d-norm value = %g \n",lval,(float) fnorm);
    if(NULL==want || 0==strcmp(want,"var"))
        printf("variance = %g \n",(float) fvar);
    if(NULL==want || 0==strcmp(want,"std"))
    printf("standard deviation = %g \n",(float) fstd);
```

sfcat

连结数据库.			
sfcat > out.rsf space= axis=3 nspace=(int) (ni/(20*nin) + 1) [<file0.rsf] file1.rsf file2.rsf ...			
sfmerge inserts additional space between merged data.			
int	**axis=3**		轴线合并
int	**nspace=(int) (ni/(20*nin) + 1)**		若 space=y, 需要输入的道数
bool	**space=**	[y/n]	输入另外的空间 y 是 sfmerge 中默认的, n 是 sfcat 中默认的

sfcat 和 sfmerge 将两个以上的文件沿某个特定的轴连结。两个程序实际上是相同的，只是在 sfcat 中默认 space=n，在 sfmerge 中默认 space=y。sfcat 举例：

```
bash$ sfspike n1=2 n2=3 > one.rsf
bash$ sfin one.rsf
one.rsf:
    in="/tmp/one.rsf@"
    esize=4 type=float form=native
    n1=2            d1=0.004        o1=0            label1="Time" unit1="s"
    n2=3            d2=0.1          o2=0            label2="Distance" unit2="km"
        6 elements 24 bytes
bash$ sfcat one.rsf one.rsf axis=1 > two.rsf
bash$ sfin two.rsf
two.rsf:
    in="/tmp/two.rsf@"
    esize=4 type=float form=native
    n1=4            d1=0.004        o1=0            label1="Time" unit1="s"
    n2=3            d2=0.1          o2=0            label2="Distance" unit2="km"
        12 elements 48 bytes
```

sfmerge 举例：

```
bash$ sfmerge one.rsf one.rsf axis=2 > two.rsf
bash$ sfin two.rsf
two.rsf:
    in="/tmp/two.rsf@"
    esize=4 type=float form=native
```

```
n1=2            d1=0.004        o1=0            label1="Time" unit1="s"
n2=7            d2=0.1          o2=0            label2="Distance" unit2="km"
        14 elements 56 bytes
```

在这种情况下，在两个连结的文件之间会插入一个空道。还要检测没有融合的轴的一致性：

```
bash$ sfcat one.rsf two.rsf > three.rsf
sfcat: n2 mismatch: need 3
```

程序实现：system/main/cat.c

第一个输入文件要么在列表中，要么是标准的输入。

```
in = (sf_file*) sf_alloc ((size_t) argc,sizeof(sf_file));

    if (!sf_stdin()) { /* no input file in stdin */
        nin=0;
    } else {
        in[0] = sf_input("in");
        nin=1;
    }
```

在命令行中不含“=”的行都会被当作一个文件名，相应的目标文件就会被加入到列表中。

```
for (i=1; i< argc; i++) { /* collect inputs */
        if (NULL != strchr(argv[i],'='))
            continue; /* not a file */
        in[nin] = sf_input(argv[i]);
        nin++;
    }
    if (0==nin) sf_error ("no input");
```

正如上面所述，如果没有设置 sapce=中的参数，就会根据程序的名字推测：sfmerge 与 space=y 相对应，sfcat 与 space=n 相对应。

```
if (!sf_getbool("space",&space)) {
        /* Insert additional space.
            y is default for sfmerge, n is default for sfcat */
```

```
        prog = sf_getprog();
        if (NULL != strstr (prog, "merge")) {
            space = true;
        } else if (NULL != strstr (prog, "cat")) {
            space = false;
        } else {
            sf_warning("%s is neither merge nor cat,"
                       " assume merge",prog);
            space = true;
        }
    }
```

（从命令行 axis=argument 中）找到需要连结的轴，并指定两个尺寸：n1 是在轴前面，n2 是在轴后面。

```
n1=1;
    n2=1;
    for (i=1; i <= dim; i++) {
        if          (i < axis) n1 *= n[i-1];
        else if (i > axis) n2 *= n[i-1];
    }
```

在输出中，选定的轴会被扩展。

```
/* figure out the length of extended axis */
    ni = 0;
    for (j=0; j < nin; j++) {
        ni += naxis[j];
    }

    if (space) {
        if (!sf_getint("nspace",&nspace))
            nspace = (int) (ni/(20*nin) + 1);
        /* if space=y, number of traces to insert */
        ni += nspace*(nin-1);
    }

    (void) snprintf(key,3,"n%d",axis);
```

```
    sf_putint(out,key,(int) ni);
```

剩下的很简单：在数据集中按缓存的空间大块循环读写数据，如果 space=y，需要加入额外的空余空间。

```
for (i2=0; i2 < n2; i2++) {
        for (j=0; j < nin; j++) {
            for (ni = n1*naxis[j]*esize; ni > 0; ni -= nbuf) {
                nbuf = (BUFSIZ < ni)? BUFSIZ: ni;
                sf_charread (buf,nbuf,in[j]);
                sf_charwrite (buf,nbuf,out);
            }
            if (!space || j == nin-1) continue;
            /* Add spaces */
            memset(buf,0,BUFSIZ);
            for (ni = n1*nspace*esize; ni > 0; ni -= nbuf) {
                nbuf = (BUFSIZ < ni)? BUFSIZ: ni;
                sf_charwrite (buf,nbuf,out);
            }
        }
    }
```

sfcdottest

Generic dot-product test for complex linear operators with adjoints **含伴随矩阵复线性算子的通用点乘测试**		
sfcdottest mod=mod.rsf dat=dat.rsf > pip.rsf		
文件	**dat=**	辅助输入文件名
文件	**mod=**	辅助输入文件名

本程序可以利用复数——复数 FFT 是一个线性的算子来简单地证明:

```
sfspike n1=100 | sfrtoc > spike.rsf
< spike.rsf sffft axis=1 pad=1 > spike2.rsf
sfcdottest sffft mod=spike.rsf dat=spike2.rsf axis=1 pad=1
```

输出会显示值会完全相同：

```
sfcdottest:    L[m]*d=(3.73955,-1.86955)
sfcdottest: L'[d]*m=(3.73955,-1.86955)
```

sfcmplx

Create a complex dataset from its real and imaginary parts. **根据实部和虚部建立复数数据集**
sfcmplx > cmplx.rsf real.rsf imag.rsf
There has to be only two input files specified and no additional parameters. 只能有两个明确的输入文件且没有其它参数

sfcmplx 简单地根据实部和虚部建立其一个复数的数据集。你操作可以用 sfreal 和 sfimage 实现。

Sfcmplx 应用举例：

```
bash$ sfspike n1=2 n2=3 > one.rsf
bash$ sfin one.rsf
one.rsf:
    in="/tmp/one.rsf@"
    esize=4 type=float form=native
    n1=2            d1=0.004        o1=0            label1="Time" unit1="s"
    n2=3            d2=0.1          o2=0            label2="Distance"
unit2="km"
        6 elements 24 bytes
bash$ sfcmplx one.rsf one.rsf > cmplx.rsf
bash$ sfin cmplx.rsf
cmplx.rsf:
    in="/tmp/cmplx.rsf@"
    esize=8 type=complex form=native
    n1=2            d1=0.004        o1=0            label1="Time" unit1="s"
    n2=3            d2=0.1          o2=0            label2="Distance"
unit2="km"
        6 elements 48 bytes
```

程序实现：system/main/cmpla.c

```
/* the first two non-parameters are real and imaginary files */
    for (i=1; i< argc; i++) {
        if (NULL == strchr(argv[i],'=')) {
            if (NULL == real) {
                real = sf_input (argv[i]);
            } else {
                imag = sf_input (argv[i]);
                break;
            }
        }
    }
    if (NULL == imag) {
        if (NULL == real) sf_error ("not enough input");
        /* if only one input, real is in stdin */
        imag = real;
        real = sf_input("in");
    }
```

程序的主要部分将实部和虚部逐个读入缓存并结合，得到复数的输出结果。

```
for (nleft= (size_t) (rsize*resize); nleft > 0; nleft -= nbuf) {
        nbuf = (BUFSIZ < nleft)? BUFSIZ: nleft;
        sf_charread(rbuf,nbuf,real);
        sf_charread(ibuf,nbuf,imag);
        for (i=0; i < nbuf; i += resize) {
            memcpy(cbuf+2*i,          rbuf+i,(size_t) resize);
            memcpy(cbuf+2*i+resize,ibuf+i,(size_t) resize);
        }
        sf_charwrite(cbuf,2*nbuf,cmplx);
    }
```

sfconjgrad

Generic conjugate-gradient solver for linear inversion **线性反演的共个梯度解**		
sfconjgrad < dat.rsf mod=mod.rsf > to.rsf < from.rsf > out.rsf niter=1		
int	**niter=1**	迭代次数

Sfconjgrad 是一个使用共轭梯度法进行最小平方线性反演 的通用程序。假设你有一个可执行程序<prog>，它从标准输入中取一个 RSF 文件，然后在标准输出中产生一个 RSF 文件。它可能取很多参数，但是参数中必须有 adj=1 来设定 向前运算（adj=0）或者伴随矩阵操作（adj=1）。程序<prog>是一个典型的 RSF 程序，但是只要使用线性算子 L 及其伴随矩阵，它可以有多种形式（脚本、多处理器 MPI 程序等）。对数据的形状没有限制。可以用 sfdottest 对伴随矩阵进行检验。给定输入数据向量 d，sfconjagrad 程序的目标是寻找一个向量 m，满足使误差平方 $\|d-Lm\|^2$ 最小。

sfconjagrad 的伪代码在 Jon Claerbout 的书《Imaging Estimation by Example》中的“最小平方模型拟合”一章中给出，早期是在“"Conjugate Gradient Tutorial"（SEP-48，1986，同一作者）”中发表。用一个小矩阵进行试验发现当求解方程 $A^T Ax = A^T b$ 时，这个算法与 J.R. Shewchuk, 1994 年的文章"An introduction to the Conjugate Gradient Method Without the Agonizing Pain"中式 45-49 页所描述的算法每一步产生的结果都相同。

与转秩相乘保证了当矩阵 A 是一个非对称方阵或非方阵时仍然有正确解。程序 sfcconjgrad 实现了当输入是复数时的算法。

下面是一个例子。sfhelicon 程序实现了 Claerbout's 的多维螺旋形滤波 (Claerbout, 1998[1])。除有输入输出向量外还要有特定的滤波器。我们用 Unix 的 echo 命令设计了一个 2 维滤波器。

```
bash$ echo 1 19 20 n1=3 n=20,20 data_format=ascii_int in=lag.rsf > lag.rsf
bash$ echo 1 1 1 a0=-3 n1=3 data_format=ascii_float in=flt.rsf > flt.rsf
```

然后，用 sfspike 设计了一个 2 维模型。

```
bash$ sfspike n1=50 n2=50 > vec.rsf
```

sfdottest 程序可以通过点乘来检测伴随矩阵是否正确.

```
bash$ sfdottest sfhelicon filt=flt.rsf lag=lag.rsf \
> mod=vec.rsf dat=vec.rsf
sfdottest:    L[m]*d=5.28394
sfdottest: L'[d]*m=5.28394
```

你的数据可能有点不同，这是由于 sfdottest 才每次运行时产生随机的输入，我们再用 sfnoise 产生一些随机数据。

```
bash$ sfnoise seed=2005 rep=y < vec.rsf > dat.rsf
```

用 sfconjgrad 来逆推滤波过程：

```
bash$ sfconjgrad sfhelicon filt=flt.rsf lag=lag.rsf \
```

```
mod=vec.rsf < dat.rsf > mod.rsf niter=10
sfconjgrad: iter 1 of 10
sfconjgrad: grad=3253.65
sfconjgrad: iter 2 of 10
sfconjgrad: grad=289.421
sfconjgrad: iter 3 of 10
sfconjgrad: grad=92.3481
sfconjgrad: iter 4 of 10
sfconjgrad: grad=36.9417
sfconjgrad: iter 5 of 10
sfconjgrad: grad=18.7228
sfconjgrad: iter 6 of 10
sfconjgrad: grad=11.1794
sfconjgrad: iter 7 of 10
sfconjgrad: grad=7.26941
sfconjgrad: iter 8 of 10
sfconjgrad: grad=5.15945
sfconjgrad: iter 9 of 10
sfconjgrad: grad=4.23055
sfconjgrad: iter 10 of 10
sfconjgrad: grad=3.57495
```

The output shows that, in 10 iterations, the norm of the gradient vector decreases by almost 1000. We can check the residual misfit before

```
bash$ < dat.rsf sfattr want=norm
norm value = 49.7801
```

and after

```
bash$ sfhelicon filt=flt.rsf lag=lag.rsf < mod.rsf | \
sfadd scale=1,-1 dat.rsf | sfattr want=norm
norm value = 5.73563
```

在 10 次迭代中，拟合差呈一定数量级地衰减，通过增加迭代次数来提高解的精度。

对于复数输入相应的程序是 sfcconjgrad。$RSFROOT/lib/conjgrad.py 是用 Python 实现这个功能的一个简单程序。

sfcp

复制或者移动一个数据集
sfcp in.rsf out.rsf
sfcp - copy, sfmv - move. Mimics 标准 Unix 命令

sfcp 和 sfmv 命令与 Unix 中的 cp 和 mv 命令类似，用来复制和移动 RSF 文件，例如：

```
bash$ sfspike n1=2 n2=3 > one.rsf
bash$ sfin one.rsf
one.rsf:
    in="/tmp/one.rsf@"
    esize=4 type=float form=native
    n1=2                d1=0.004        o1=0            label1="Time"
unit1="s"
    n2=3                d2=0.1          o2=0            label2="Distance"
unit2="km"
        6 elements 24 bytes
bash$ sfcp one.rsf two.rsf
bash$ sfin two.rsf
two.rsf:
    in="/tmp/two.rsf@"
    esize=4 type=float form=native
    n1=2                d1=0.004        o1=0            label1="Time"
unit1="s"
    n2=3                d2=0.1          o2=0            label2="Distance"
unit2="km"
        6 elements 24 bytes
```

程序实现：system/main/cp.c

首先，需找两个不包含“=”号的命令行，将它们作为输入和输出文件名。

```
/* the first two non-parameters are in and out files */
    for (i=1; i< argc; i++) {
        if (NULL == strchr(argv[i],'=')) {
            if (NULL == in) {
                infile = argv[i];
                in = sf_input (infile);
```

```
                } else {
                    out = sf_output (argv[i]);
                    break;
                }
            }
        }
        if (NULL == in || NULL == out)
            sf_error ("not enough input");
```

然后，我们利用库函数 sf_cp 和 sf_m 来进行操作。

```
sf_cp(in,out);
    if (NULL != strstr (sf_getprog(),"mv"))
        sf_rm(infile,false,false,false);
```

sfcut

将数据集的一部分置零			
sfcut < in.rsf > out.rsf verb=n [j1=1 j2=1 ... f1=0 f2=0 ... n1=n1 n2=n2 ... max1= max2= ... min1= min2= ...]			
jN 定义为第 N 维 fN 是窗口的起始点 nN 是窗口的尺寸 minN and maxN 是第 N 维的极大和极小值 窗口反转			
bool	**verb=n**	[y/n]	冗余标志

sfcut 命令与 sfwindow 命令有关，除需要提取选定的窗口将其置零外，它们的参数是相同的输入数据的尺寸被保留下来。例如：

```
bash$ sfspike n1=5 n2=5 > in.rsf
bash$ < in.rsf sfdisfil
    0:          1          1          1          1          1
    5:          1          1          1          1          1
   10:          1          1          1          1          1
   15:          1          1          1          1          1
   20:          1          1          1          1          1
```

```
bash$ < in.rsf sfcut n1=2 f1=1 n2=3 f2=2 | sfdisfil
   0:          1          1          1          1          1
   5:          1          1          1          1          1
  10:          1          0          0          1          1
  15:          1          0          0          1          1
  20:          1          0          0          1          1
bash$ < in.rsf sfcut j1=2 | sfdisfil
   0:          0          1          0          1          0
   5:          0          1          0          1          0
  10:          0          1          0          1          0
  15:          0          1          0          1          0
  20:          0          1          0          1          0
```

sfdd

不同格式之间的转换		
sfdd < in.rsf > out.rsf line=8 form= type= format=		
string	**form=**	ascii, native, xdr
string	**format=**	元素的格式 (转换成 ASCII 码)
int	**line=8**	每行数据的个数(转换成 ASCII 码)
string	**type=**	int, float, complex

sfdd 程序用来转换格式为（ascii, xdr, native）或(complex, float, int, char)的输入数据集。在下面的例子中，我们建立了一个含有数字的纯文本（ASCII）文件，然后用 sfdd 产生一个含有复数的 xdr 形式的 RSF 文件。

```
bash$ cat test.txt
1 2 3 4 5 6
bash$ echo n1=6 data_format=ascii_int in=test.txt > test.rsf
bash$ sfin test.rsf
test.rsf:
        in="test.txt"
        esize=0 type=int form=ascii
        n1=6                    d1=?                    o1=?
                6 elements
bash$ sfdd < test.rsf form=xdr type=complex > test2.rsf
bash$ sfin test2.rsf
```

```
test2.rsf:
        in="/tmp/test2.rsf@"
        esize=8 type=complex form=xdr
        n1=3                d1=?                o1=?
                3 elements 24 bytes
bash$ sfdisfil < test2.rsf
    0:              1,              2i              3,              4i              5,              6i
```

若想了解更多关于 RSF 数据格式，请参考 guide to RSF format.

sfdisfil

输出数据的值			
sfdisfil < in.rsf number=y col=0 format= header= trailer=			
也可以使用 sfdd 转换成 ASCII 格式			
int	**col=0**		行数 默认值与数据类型有关: 10 for int and char, 5 for float, 3 for complex
string	**format=**		数据格式(输出形式). 默认值与数据类型有关: "%4d "为 int and char, "%13.4g" 为 float, "%10.4g,%10.4gi" 为 complex
string	**header=**		Optional header string to output before data 输出数据前部的可选的头部字符串
bool	**number=y**	[y/n]	如果对元素进行标号
string	**trailer=**		Optional trailer string to output after data 输出数据后部的可选的尾部字符串

sfdisfil 程序只是简单地将数据内容以文本的形式输送到输出文件中。通常被用来快速检查 RSF 文件进行调试。下面是一个例子：

```
bash$ sfmath o1=0 d1=2 n1=12 output=x1 > test.rsf
bash$ < test.rsf sfdisfil
    0:              0              2              4              6              8
```

```
  5:          10          12          14          16          18
 10:          20          22
```

输出文件格式很好配置。

```
bash$ < test.rsf sfdisfil col=6 number=n format="%5.1f"
  0.0   2.0   4.0   6.0   8.0 10.0
 12.0 14.0 16.0 18.0 20.0 22.0
```

和 sfdd 一样，sfdisfil 也提供了一种将 RSF 数据转换成 ASCII 形式的途径。

sfdottest

Generic dot-product test for linear operators with adjoints

针对伴随矩阵线性算子的通用点乘测试

sfdottest mod=mod.rsf dat=dat.rsf > pip.rsf

sfdottest 是一个检测线性算子的通用点乘测试程序。如果有一个可执行程序<prog>将 RSF 文件有标准的输入格式转换成标准输出格式的文件，它需要一些必须包含设定向前（adj=0）或伴随矩阵（adj=1）操作等另外的参数。<prog>是个典型的 RSF 程序，只要它使用了线性算子 L 及其伴随矩阵 L^T，它可以有多种形式如脚本、并行 MPI 程序等。sfdottest 程序通过使用随机向量 m 和 d 来检测式 $d^T Lm = m^T L^T d$。可以通过下式来调用：

```
bash$ sfdottest <prog> [optional aruments] mod=mod.rsf dat=dat.rsf
```

其中 mod.rsf 和 dat.rsf 表示模型和数据空间向量的 RSF 文件。注意这些向量的维数！如果程序长时间没有响应，很可能是向量的维数与所要检测的程序要求不符。Sfdottest 不产生临时文件，也没有对向量额维数有任何限制。下面是一个例子。首先使用 sfspike 创建一个含有 100 个元素的向量，然后运行 sfdottest 来测试 sfcausint 程序。sfcausint 使用了一个因果关系积分线性算子及其非因果的伴随矩阵。

```
bash$ sfspike n1=100 > vec.rsf
bash$ sfdottest sfcausint mod=vec.rsf dat=vec.rsf
sfdottest:    L[m]*d=1410.2
sfdottest: L'[d]*m=1410.2
bash$ sfdottest sfcausint mod=vec.rsf dat=vec.rsf
sfdottest:    L[m]*d=1165.87
sfdottest: L'[d]*m=1165.87
```

由于随机数产生器中的种子会变化，在后续的运行中这些数据也会变化。下面是一个稍微难一些的例子。sfhelicon 程序实现了 Clearbout 的多维螺旋形滤波器（claerbout，1998[2]）。它除了输入和输出向量以外还要求一个特定的滤波器。我们使用 Unix 中的 echo

命令设计一个 2 维滤波器。

```
bash$ echo 1 19 20 n1=3 n=20,20 data_format=ascii_int in=lag.rsf > lag.rsf
bash$ echo 1 1 1 a0=-3 n1=3 data_format=ascii_float in=flt.rsf > flt.rsf
```

然后，使用 sfspike 建立一个 2 维模型和数据向量。

```
bash$ sfspike n1=50 n2=50 > vec.rsf
```

现在可以用 sfdottest 程序来进行点乘检测了。

```
bash$ sfdottest sfhelicon filt=flt.rsf lag=lag.rsf \
> mod=vec.rsf dat=vec.rsf
sfdottest:    L[m]*d=8.97375
sfdottest: L'[d]*m=8.97375
```

下面是一个检测反滤波的相同的程序

```
bash$ sfdottest sfhelicon filt=flt.rsf lag=lag.rsf \
> mod=vec.rsf dat=vec.rsf div=y
sfdottest:    L[m]*d=15.0222
sfdottest: L'[d]*m=15.0222
```

sfget

从头文件中输出参数			
sfget < in.rsf parform=y par1 par2 ...			
bool	**parform=y**	[y/n]	如果是 y，输出 patameter=值，如果是 n，输出值。

sfget 程序从 RSF 文件中提取了一个参数值，在使用脚本时有用。例如使用标准的 Unix bc 计算器来对 RSF 数据集(sfspike 的输出)中第一列上的极大值进行快速的计算。

```
bash$ ( sfspike n1=100 | sfget n1 d1 o1; echo "o1+(n1-1)*d1" ) | bc
.396
```

参考 sfput。

程序实现： system/main/get.c

实现起来是比较麻烦的。在所有含“=”号的命令行中循环。

```
for (i = 1; i < argc; i++) {
        key = argv[i];
        if (NULL != strchr(key,'=')) continue;
```

得到参数值后（字符串形式）将其以 key=value 或者 value 的形式输出，这取决于 parform 的参数。

```
string = sf_histstring(in,key);
        if (NULL == string) {
            sf_warning("No key %s",key);
        } else {
            if (parform) printf ("%s=",key);
            printf("%s\n",string);
    }
```

sfheadercut

Zero a portion of a dataset based on a header mask. 根据头文件的 mask 将数据集的一部分置零
sfheadercut mask=head.rsf < in.rsf > out.rsf
输入数据是一系列 n1xn2 的地震道，， mask 是长度为 n2 的一个整形数组。

sfheadercut 与 sfheaderwindow 很像，但是它没有给数据集加上一个窗口，而是按照头文件 mask 中指定的方式给地震道赋零。输入数据的尺寸被保留下来。。下面是一个用 sfheaderwindow 来对输入文件的每隔一道赋零的例子：

```
bash$ sfmath n1=5 n2=10 output=x2+1 > input.rsf
bash$ < input.rsf sfdisfil
    0:          1          1          1          1          1
    5:          2          2          2          2          2
   10:          3          3          3          3          3
   15:          4          4          4          4          4
   20:          5          5          5          5          5
   25:          6          6          6          6          6
   30:          7          7          7          7          7
   35:          8          8          8          8          8
   40:          9          9          9          9          9
   45:         10         10         10         10         10
```

然后，使用 sfinterleave 建立一个含有 1 和 0 的 mask。

```
bash$ sfspike n1=5 mag=1 | sfdd type=int > ones.rsf
bash$ sfspike n1=5 mag=0 | sfdd type=int > zeros.rsf
bash$ sfinterleave axis=1 ones.rsf zeros.rsf > mask.rsf
bash$ sfdisfil < mask.rsf
```

```
  0:     1     0     1     0     1     0     1     0     1     0
```

最后，用 sfheadercut 使输入的道为零。

```
bash$ sfheadercut < input.rsf mask=mask.rsf > output.rsf
bash$ sfdisfil < output.rsf
  0:          1          1          1          1          1
  5:          0          0          0          0          0
 10:          3          3          3          3          3
 15:          0          0          0          0          0
 20:          5          5          5          5          5
 25:          0          0          0          0          0
 30:          7          7          7          7          7
 35:          0          0          0          0          0
 40:          9          9          9          9          9
 45:          0          0          0          0          0
```

sfheadersort

Sort a dataset according to a header key. **根据头文件的关键字对数据集分类**		
sfheadersort < in.rsf > out.rsf head=		
string	**head=**	头文件

Sfheadersort 用来根据道头文件中的信息对输入文件中的地震道进行分类。下面是一个使用 sfheadersort 对输入文件中的道进行随机移动的例子。首先，创建一个含有 7 道数据的输入文件：

```
bash$ sfmath n1=5 n2=7 output=x2+1 > input.rsf
bash$ < input.rsf sfdisfil
  0:          1          1          1          1          1
  5:          2          2          2          2          2
 10:          3          3          3          3          3
 15:          4          4          4          4          4
 20:          5          5          5          5          5
 25:          6          6          6          6          6
 30:          7          7          7          7          7
```

然后，用 sfnoise 创建一个含有七个头文件值的随机文件。

```
bash$ sfspike n1=7 | sfnoise rep=y type=n > random.rsf
bash$ < random.rsf sfdisfil
  0:      0.05256      -0.2879       0.1487       0.4097       0.1548
  5:       0.4501       0.2836
```

如果你重新运行这个例子，你得到的数据很可能会不同，因为，在缺少 seed=参数时，sfnosie 使用一个随机的种子来生成伪随机数。最后，我们使用 sfheadersort 来对输入道进行排序。

```
bash$ < input.rsf sfheadersort head=random.rsf > output.rsf
bash$ < output.rsf sfdisfil
  0:           2           2           2           2           2
  5:           1           1           1           1           1
 10:           3           3           3           3           3
 15:           5           5           5           5           5
 20:           7           7           7           7           7
 25:           4           4           4           4           4
 30:           6           6           6           6           6
```

正如所预料的那样，输出文件的顺序是按照头文件中的值的大小进行排序的。由于头文件和数据是分开的，因此用 sfheadersort 可以很好地运行，它首先对头文件进行排序然后再找到数据，一道一道地读入数据。

sfheaderwindow

Window a dataset based on a header mask. **按照头文件中的 mask 对数据集加窗口**
sfheaderwindow mask=head.rsf < in.rsf > out.rsf
输入数据是一系列 n1xn2 的道（矩阵），mask 是一个长度为 n2 的整型数组，窗口大小为 n1xm2，其中 m2 是 mask 中非零元素的个数。

Sfheaderwindow 按照头文件信息对输入文件的数据加窗口。下面是一个使用 sfheaderwindow 对输入文件中随机选取一部分数据。首先，建立一个含有 10 个头文件的输入文件。

```
bash$ sfmath n1=5 n2=10 output=x2+1 > input.rsf
bash$ < input.rsf sfdisfil
  0:           1           1           1           1           1
  5:           2           2           2           2           2
 10:           3           3           3           3           3
```

```
15:          4          4          4          4          4
20:          5          5          5          5          5
25:          6          6          6          6          6
30:          7          7          7          7          7
35:          8          8          8          8          8
40:          9          9          9          9          9
45:         10         10         10         10         10
```

然后，使用 sfnoise 创建一个含有 10 个头文件值的随机文件。

```
bash$ sfspike n1=10 | sfnoise rep=y type=n > random.rsf
bash$ < random.rsf sfdisfil
    0:      -0.005768        0.02258       -0.04331        -0.4129        -0.3909
    5:       -0.03582         0.4595        -0.3326          0.498        -0.3517
```

如果你重新运行这个程序，你的数据很可能会发生变化。因为在缺少 seed=参数时，sfnoise 使用一个随机种子生成伪随机数。最后，使用 sfheaderwindow 对输入文件加窗来选择那些头文件值大于零的道。

```
bash$ < random.rsf sfmask min=0 > mask.rsf
bash$ < mask.rsf sfdisfil
    0:      0      1      0      0      0      0      1      0      1      0
bash$ < input.rsf sfheaderwindow mask=mask.rsf > output.rsf
bash$ < output.rsf sfdisfil
    0:          2          2          2          2          2
    5:          7          7          7          7          7
   10:          9          9          9          9          9
```

在这个例子中，输出文件中仅选择了三道数据。因为头文件信息和数据是分开的，所以 sfheaderwindow 运行起来十分方便。

sfin

Display basic information about RSF files. **显示 RSF 文件的基本信息**
sfin info=y check=2. trail=y file1.rsf file2.rsf ...
n1,n2,... 是数据的维数 o1,o2,... are axis origins 轴的起始点 d1,d2,... are axis sampling intervals 轴的采样间隔

label1,label2,... are axis labels 轴标签 unit1,unit2,... are axis units 轴单位			
float	**check=2**		Portion of the data (in Mb) to check for zero values.对数据进行分配（按 Mb）来检测零值。
bool	**info=y**	[y/n]	If n, only display the name of the data file.如果是 n，仅输出数据文件的名字。
bool	**trail=y**	[y/n]	If n, skip trailing dimensions of one 如果是 n，跳过后面的一维。

Sfin 是对 RSF 文件进行操作最常用的程序之一。它可以很快地找到文件的高维信息并检测相应文件的一致性。下面是一个例子。先建立一个 RSF 文件，在用 sfin 进行检测。

```
bash$ sfspike n1=100 n2=20 > spike.rsf
bash$ sfin spike.rsf
spike.rsf:
    in="/tmp/spike.rsf@"
    esize=4 type=float form=native
    n1=100          d1=0.004        o1=0            label1="Time" unit1="s"
    n2=20           d2=0.1          o2=0            label2="Distance"
unit2="km"
        2000 elements 8000 bytes
```

sfin 输出以下信息：

location of the data file 数据文件的位置(/tmp/spike.rsf\@)

element size (4 bytes)数据元素大小（4 个字节）

element type (floating point)数据元素类型（实型）

element form (native) 数据元素形式（native）

hypercube dimensions (100 by 20) 维数（100×20）

axes scale (0.004 and 0.1) 轴的尺度（0.004 和 0.1）

axes origin (0 and 0) 轴的起点（0 和 0）

axes labels 轴标签

axes units 轴单位

total number of elements 元素总个数

total number of bytes in the data file 数据文件的总字节数

假设文件中含有漏洞，报告了错误的维数。sfin 程序可以来判断矛盾的所在。

```
bash$ echo n2=100 >> spike.rsf
bash$ sfin spike.rsf > /dev/null
sfin:               Actually 8000 bytes, 20% of expected.
```

```
```

sfin 也检查文件的最初记录来寻找零值。

```
bash$ sfspike n1=100 n2=100 k2=99 > spike2.rsf
bash$ sfin spike2.rsf >/dev/null
sfin: The first 32768 bytes are all zeros
```

需要查看的字节数是可调的。

```
bash$ sfin spike2.rsf check=0.01 >/dev/null
sfin: The first 16384 bytes are all zeros
```

你也可以只输出数据文件的位置。这在脚本中很容易获得。

```
bash$ sfin spike.rsf spike2.rsf info=n
/tmp/spike.rsf@ /tmp/spike2.rsf@
```

或者使用 sfget，如下所示：

```
bash$ sfget parform=n in < spike.rsf
/tmp/spike.rsf@
```

为了消除尾随的长度为一的数据干扰（并非在 trail=n 时停止显示），你可以用 sed，下例是用来除去轴 6：

```
sed -i 's/n6=1//g' file.rsf
```

sfinterleave

Combine several datasets by interleaving. **通过插入将几个数据体合在一起**		
sfinterleave > out.rsf axis=3 [< file0.rsf] file1.rsf file2.rsf ...		
int	**axis=3**	Axis for interleaving 要插入的轴

sfinterleave 通过将两个或两个以上的数据集插入到一个轴的位置来对它们进行合并。

下面是一个简单的例子：

```
bash$ sfspike n1=5 n2=5 > one.rsf
bash$ sfdisfil < one.rsf
   0:          1          1          1          1          1
   5:          1          1          1          1          1
  10:          1          1          1          1          1
  15:          1          1          1          1          1
  20:          1          1          1          1          1
bash$ sfscale < one.rsf dscale=2 > two.rsf
bash$ sfdisfil < two.rsf
   0:          2          2          2          2          2
   5:          2          2          2          2          2
  10:          2          2          2          2          2
  15:          2          2          2          2          2
  20:          2          2          2          2          2
bash$ sfinterleave one.rsf two.rsf axis=1 | sfdisfil
   0:          1          2          1          2          1
   5:          2          1          2          1          2
  10:          1          2          1          2          1
  15:          2          1          2          1          2
  20:          1          2          1          2          1
  25:          2          1          2          1          2
  30:          1          2          1          2          1
  35:          2          1          2          1          2
  40:          1          2          1          2          1
  45:          2          1          2          1          2
bash$ sfinterleave < one.rsf two.rsf axis=2 | sfdisfil
   0:          1          1          1          1          1
   5:          2          2          2          2          2
  10:          1          1          1          1          1
  15:          2          2          2          2          2
  20:          1          1          1          1          1
  25:          2          2          2          2          2
  30:          1          1          1          1          1
  35:          2          2          2          2          2
  40:          1          1          1          1          1
```

```
45:          2          2          2          2          2
```

sfmask

Create a mask.		
sfmask < in.rsf > out.rsf min=-FLT_MAX max=+FLT_MAX		
Mask 是含有 1 和 0 的整型数据，当输入数据的值在 min 和 max 之间时与 1 有关 输出可以与 sfheaderwindow 结合使用。		
float	**max=+FLT_MAX**	头文件最大值
float	**min=-FLT_MAX**	头文件最小值

sfmask 通过比较输入数据与特定的 min=和 max=参数来产生一系列 1 和 0 等整型数值。在 sfheaderwindow 及其它程序中经常用到。下面是一个简单的例子：

```
bash$ sfmath n1=10 output="sin(x1)" > sin.rsf
bash$ < sin.rsf sfdisfil
    0:          0     0.8415     0.9093     0.1411    -0.7568
    5:    -0.9589    -0.2794      0.657     0.9894     0.4121
bash$ < sin.rsf sfmask min=-0.5 max=0.5 | sfdisfil
    0:    1    0    0    1    0    0    1    0    0    1
```

sfmath

Mathematical operations on data files. **对数据文件的数学操作**
sfmath > out.rsf type= unit= output=
已知的函数: cos, sin, tan, acos, asin, atan, cosh, sinh, tanh, acosh, asinh, atanh, exp, log, sqrt, abs, conj (针对复数). sfmath 可以处理实型数据和复数，但是所有的输入和输出数据应该是同一种数据类型。 sfmath 可以用运算效率更高的 sfadd 替换，但是 sfmath 相对灵活一些

例子:

```
sfmath x=file1.rsf y=file2.rsf power=file3.rsf output='sin((x+2*y)^power)' > out.rsf
sfmath < file1.rsf tau=file2.rsf output='exp(tau*input)' > out.rsf
sfmath n1=100 type=complex output="exp(I*x1)" > out.rsf
```

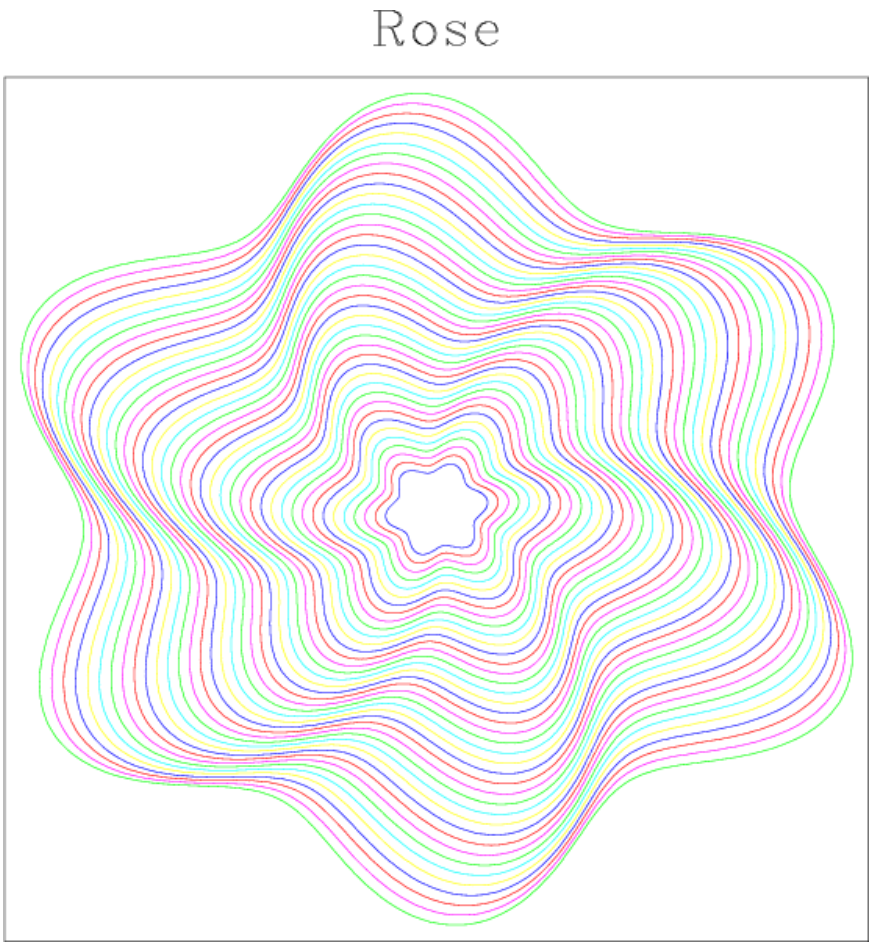
参考: sfheadermath.

string	**output=**	对输出的数学描述
string	**type=**	输出数据类型[实型,复数]
string	**unit=**	

sfmath 是一个对 RSF 文件进行数学运算的很灵活的程序。他可以对多个输入文件进行操作，前提是这些文件的维数和数据类型相同。下面用一个例子来说明 sfmath 的一些特点。

```
bash$ sfmath n1=629 d1=0.01 o1=0 n2=40 d2=1 o2=5 \
output="x2*(8+sin(6*x1+x2/10))" > rad.rsf
bash$ < rad.rsf sfrtoc | sfmath output="input*exp(I*x1)" > rose.rsf
bash$ < rose.rsf sfgraph title=Rose screenratio=1 wantaxis=n | sfpen
```

第一行命令建立了一个含 40 道，每一道有 629 个采样点的 2 维数据集。。数据的值是用公式"x2*(8+sin(6*x1+x2/10))",其中 x1 值第一个轴上的坐标，x2 是第二个轴上的坐标。第二行使用 sfrtoc 将实数转换成复数并使用公式"input*exp(I*x1)",产生一个复数集，input 指输入文件。最后，使用 sfgraph 将复数画成一组参量的曲线，用 sfpen 将结果绘制出来，应该得到一个与下图相似的图。



此图是用 sfmath 创建的。

对上面第二行程序的一个等效的办法是

```
bash$ < rad.rsf sfmath output=x1 > ang.rsf
bash$ sfmath r=rad.rsf a=ang.rsf output="r*cos(a)" > cos.rsf
bash$ sfmath r=rad.rsf a=ang.rsf output="r*sin(a)" > sin.rsf
bash$ sfcmplx cos.rsf sin.rsf > rose.rsf
```

在这里我们用（r 和 a）表示输入文件的名字并在公式中用到。

sfpad

Pad a dataset with zeros. **在数据集中填补零**
sfpad < in.rsf > out.rsf [beg1= beg2= ... end1= end2=... | n1= n2 = ... | n1out= n2out= ...]
begN 确定了在轴 N 开始前需要加入的零的个数 endN 确定了在轴 N 的末尾需要加入的零的个数

另外:

nN or nNout 确定了轴 N 的输出长度，在尾部补零。·

nN and nNout 是等价的.

Pad 通过将零加入到输入的数据中增加了数据集的维数。下面是一些简单的例子。

```
bash$ sfspike n1=5 n2=3 > one.rsf
bash$ sfdisfil < one.rsf
  0:          1          1          1          1          1
  5:          1          1          1          1          1
 10:          1          1          1          1          1
bash$ < one.rsf sfpad n2=5 | sfdisfil
  0:          1          1          1          1          1
  5:          1          1          1          1          1
 10:          1          1          1          1          1
 15:          0          0          0          0          0
 20:          0          0          0          0          0
bash$ < one.rsf sfpad beg2=2 | sfdisfil
  0:          0          0          0          0          0
  5:          0          0          0          0          0
 10:          1          1          1          1          1
 15:          1          1          1          1          1
 20:          1          1          1          1          1
bash$ < one.rsf sfpad beg2=1 end2=1 | sfdisfil
  0:          0          0          0          0          0
  5:          1          1          1          1          1
 10:          1          1          1          1          1
 15:          1          1          1          1          1
 20:          0          0          0          0          0
bash$ < one.rsf sfwindow n1=3 | sfpad n1=5 n2=5 beg1=1 beg2=1 | sfdisfil
  0:          0          0          0          0          0
  5:          0          1          1          1          0
 10:          0          1          1          1          0
 15:          0          1          1          1          0
 20:          0          0          0          0          0
```

你可以用 sfcat 来加入非零值。

sfput

Input parameters into a header. 在头文件中加入参数
sfput < in.rsf > out.rsf

sfput 是一个很简单的程序，它只是简单地将参数从命令行加到 RSF 文件中。同样，可以通过手动加入或者使用标准的 Unix 命令如 sed 和 echo 来达到同样的目的。Sfput 有时用起来更方便，因为它对输入/输出选项的处理与其它的 RSF 程序类似。

```
bash$ sfspike n1=10 > spike.rsf
bash$ sfin spike.rsf
spike.rsf:
    in="/tmp/spike.rsf@"
    esize=4 type=float form=native
    n1=10               d1=0.004          o1=0              label1="Time" unit1="s"
        10 elements 40 bytes
bash$ sfput < spike.rsf d1=25 label1=Depth unit1=m > spike2.rsf
bash$ sfin spike2.rsf
spike2.rsf:
    in="/tmp/spike2.rsf@"
    esize=4 type=float form=native
    n1=10               d1=25             o1=0              label1="Depth"
unit1="m"
        10 elements 40 bytes
```

sfreal

Extract real (sfreal) or imaginary (sfimag) part of a complex dataset. 从复数集中提取实部（sfreal）或虚部（sfimag）
sfreal < cmplx.rsf > real.rsf

sfreal 的作用是提取复数类型数据集中的实部。提取虚部可以用 sfima。将实部和虚部结合起来用 sfcmplx。这里有个简单的例子，我们首先用 sfmath 建立一个复数数据集。

```
bash$ sfmath n1=10 type=complex output="(2+I)*x1" > cmplx.rsf
bash$ fdisfil < cmplx.rsf
```

```
0:      0,      0i      2,      1i      4,      2i
3:      6,      3i      8,      4i      10,     5i
6:      12,     6i      14,     7i      16,     8i
9:      18,     9i
```

用 sfreal 提取实部：

```
bash$ sfreal < cmplx.rsf | sfdisfil
0:      0       2       4       6       8
5:      10      12      14
```

用 sfimag 提取虚部：

```
bash$ sfimag < cmplx.rsf | sfdisfil
0:      0       1       2       3       4
5:      5       6       7
```

sfreverse

Reverse one or more axes in the data hypercube. 在高维数据集中反转一个或几个轴			
sfreverse < in.rsf > out.rsf which=-1 verb=false memsize=sf_memsize() opt=			
int	**memsize=sf_memsize()**		Max amount of RAM (in Mb) to be used 所需要的最大存储空间（Mb）
string	**opt=**		如果是 y，将 o 和 d 参数在翻转后的轴上交换。 如果是 i，o’和 d 保持不变
bool	**verb=n**	[y/n]	冗余标志
int	**which=-1**		反转哪个轴 要反转一个指定的轴，从 0 开始 加 1 表示反转第 n1 维 加 2 表示反转第 n2 维 加 4 表示反转第 n3 维 因此 which=7 会反转前三维，因为 1+2+4=7 which=5 反转 1 和 n3 维，等等 which=0 不会改变数据集

下面是一个使用 sfreverse 的例子。首先，我们建立一个 2 维数据集。

```
bash$ sfmath n1=5 d1=1 n2=3 d2=1 output=x1+x2 > test.rsf
```

```
bash$ < test.rsf sfdisfil
  0:          0          1          2          3          4
  5:          1          2          3          4          5
 10:          2          3          4          5          6
```

将第一个轴反转：

```
bash$ < test.rsf sfreverse which=1 | sfdisfil
  0:          4          3          2          1          0
  5:          5          4          3          2          1
 10:          6          5          4          3          2
```

将第二个轴反转：

```
bash$ < test.rsf sfreverse which=2 | sfdisfil
  0:          2          3          4          5          6
  5:          1          2          3          4          5
 10:          0          1          2          3          4
```

同时反转第一个和第二个轴：

```
bash$ < test.rsf sfreverse which=3 | sfdisfil
  0:          2          3          4          5          6
  5:          1          2          3          4          5
 10:          0          1          2          3          4
```

正如你所看到的，which=参数通过将所要反转的轴编码控制来使其反转。当一轴反转时，对它的起点和采样参数有何影响？这是通过 opt=参数来控制的。在我们的例子中：

```
bash$ < test.rsf sfget n1 o1 d1
n1=58
o1=0
d1=1
bash$ < test.rsf sfreverse which=1 | sfget o1 d1
o1=4
d1=-1
```

默认（相当于 opt=y）的操作是将起始点 o1 放在数轴的尾部并将采样参数 d1 反转。用 opt=n 保留采样参数，只把起始点反转。

```
bash$ < test.rsf sfreverse which=1 opt=n | sfget o1 d1
o1=-4
d1=1
```

用 opt=i 只是将轴反转，而采样点和初始点保持不变。

```
bash$ < test.rsf sfreverse which=1 opt=i | sfget o1 d1
o1=0
d1=1
```

在应用的时候，可能会用到三种中的某一种情况。

sfrm

Remove RSF files together with their data. **将 RSF 文件和数据同时删除**
sfrm file1.rsf [file2.rsf ...] [-i] [-v] [-f]
Mimics the standard Unix rm command. 仿照标准的 Unix rm 命令 参考: sfmv, sfcp.

sfrm 用来删除 RSF 文件。

```
bash$ sfspike n1=10 > spike.rsf datapath=./
bash$ sfget in < spike.rsf
in=./spike.rsf@
bash$ ls spike*
spike.rsf    spike.rsf@
bash$ sfrm -v spike.rsf
sfrm: sf_rm: Removing header spike.rsf
sfrm: sf_rm: Removing data ./spike.rsf@
bash$ ls spike*
ls: No match.
```

sfrotate

Rotate a portion of one or more axes in the data hypercube.、 对多维数据一个或多个轴的一部分进行旋转			
sfrotate < in.rsf > out.rsf verb=n memsize=sf_memsize() rot#=(0,0,...)			
int	**memsize=sf_memsize()**		Max amount of RAM (in Mb) to be used 需要用到的内存（Mb）
int	**rot#=(0,0,...)**		length of #-th axis that is moved to the end 要移动到尾部的第#个轴的长度
bool	**verb=n**	[y/n]	Verbosity flag 冗余标志

sfrotate 通过将输入数据分成几部分然后将这些部分用不同的顺序排列来进行改造。下面是个简单的例子。

```
bash$ sfmath n1=5 d1=1 output=x1+x2 > test.rsf
bash$ < test.rsf sfdisfil
  0:          0          1          2          3          4
  5:          1          2          3          4          5
 10:          2          3          4          5          6
```

通过将最后两列放到前面来对第一个轴进行旋转：

```
bash$ < test.rsf sfrotate rot1=2 | sfdisfil
  0:          3          4          0          1          2
  5:          4          5          1          2          3
 10:          5          6          2          3          4
```

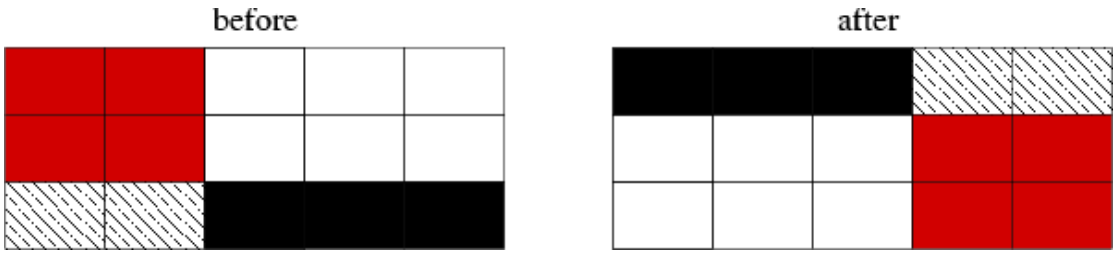
通过把最后一行放到前面对第二个轴进行旋转

```
bash$ < test.rsf sfrotate rot2=1 | sfdisfil
  0:          2          3          4          5          6
  5:          0          1          2          3          4
 10:          1          2          3          4          5
```

同时对第一个和第二个轴进行旋转：

```
bash$ < test.rsf sfrotate rot1=3 rot2=1 | sfdisfil
  0:          4          5          6          2          3
  5:          2          3          4          0          1
 10:          3          4          5          1          2
```

图（fig：rotate）中展示了旋转的过程：



sfrtoc

Convert real data to complex (by adding zero imaginary part). 通过在虚部加零将实数转换为复数
sfrtoc < real.rsf > cmplx.rsf
参考: sfcmplx

Real to complex

sfrtoc 的输入文件可是任意 type=float 的数据集。

```
bash$ sfspike n1=10 n2=20 n3=30 >real.rsf
bash$ sfin real.rsf
real.rsf:
    in="/var/tmp/real.rsf@"
    esize=4 type=float form=native
    n1=10          d1=0.004       o1=0          label1="Time" unit1="s"
    n2=20          d2=0.1         o2=0          label2="Distance" unit2="km"
    n3=30          d3=0.1         o3=0          label3="Distance" unit3="km"
        6000 elements 24000 bytes
```

输出文件数据集成为了 type=complex，其二进制格式存储空间是输入文件的两倍。

```
bash$ <real.rsf sfrtoc >complex.rsf
bash$ sfin complex.rsf
complex.rsf:
    in="/var/tmp/complex.rsf@"
    esize=8 type=complex form=native
    n1=10          d1=0.004       o1=0          label1="Time" unit1="s"
    n2=20          d2=0.1         o2=0          label2="Distance" unit2="km"
```

```
n3=30           d3=0.1          o3=0            label3="Distance" unit3="km"
        6000 elements 48000 bytes
```

sfscale

Scale data. **对数据进行尺度变换**		
sfscale < in.rsf > out.rsf axis=0 rscale=0. dscale=1.		
To scale by a constant factor, you can also use sfmath or sfheadermath. 用一个常数因子进行尺度变换，你也可以使用 sfmath 或 sfheadermath。		
int	**axis=0**	Scale by maximum in the dimensions up to this axis. 用直到这个轴上维数的最大值进行尺度变换
float	**dscale=1.**	Scale by this factor (works if rscale=0) 用这个因子进行尺度变换（rscale=0 时生效）
float	**rscale=0.**	Scale by this factor. 用这个因子进行尺度变换。

sfscale 用一个因子对输入数据集进行尺度变换。下面是一个简单的例子，首先建立一个测试数据集。

```
bash$ sfmath n1=5 n2=3 o1=1 o2=1 output="x1*x2" > test.rsf
bash$ < test.rsf sfdisfil
    0:          1          2          3          4          5
    5:          2          4          6          8         10
   10:          3          6          9         12         15
```

将每个数据的尺度变换 2 倍：

```
bash$ < test.rsf sfscale dscale=2 | sfdisfil
    0:          2          4          6          8         10
    5:          4          8         12         16         20
   10:          6         12         18         24         30
```

用每一道数据除以它们的极大值：

```
bash$ < test.rsf sfscale axis=1 | sfdisfil
    0:        0.2        0.4        0.6        0.8          1
    5:        0.2        0.4        0.6        0.8          1
   10:        0.2        0.4        0.6        0.8          1
```

除以 2 维数据集中的最大值：

```
bash$ < test.rsf sfscale axis=2 | sfdisfil
   0:      0.06667      0.1333         0.2      0.2667      0.3333
   5:       0.1333      0.2667         0.4      0.5333      0.6667
  10:          0.2         0.4         0.6         0.8           1
```

参数 rscale=当不为零时与 dscale=相同。当 sfscale dscale=0 时，数据集乘以零。如果使用 rscale=0 其它的数据用来定义尺度变换。因此 sfscale rscale=0 axis=1 与 sfsscale axis=1 等价，sfscale rscale=0 与 sfscale dscale=1 等价。

sfspike

Generate simple data: spikes, boxes, planes, constants. **生成简单的数据：脉冲、箱状、 平面、常数**	
sfspike > spike.rsf mag= nsp=1 k#=[0,...] l#=[k1,k2,...] p#=[0,...] n#= o#=[0,0,...] d#=[0.004,0.1,0.1,...] label#=[Time,Distance,Distance,...] unit#=[s,km,km,...] title=	
Spike positioning is given in samples and starts with 1. 脉冲的位置在采样点中给出，从 1 开始	
float **d#=[0.004,0.1,0.1,...]**	在轴#上采样
ints **k#=[0,...]**	脉冲开始的位置[nsp]
ints **l#=[k1,k2,...]**	脉冲结束的位置[nsp]
string **label#=[Time,Distance,Distance,...]**	在轴 #上标注
floats **mag=**	脉冲幅度 [nsp]
int **n#=**	第#个轴的维数
int **nsp=1**	脉冲的个数
float **o#=[0,0,...]**	第#个轴的起点
floats **p#=[0,...]**	脉冲斜坡(用采样点表示) [nsp]
string **title=**	图的标题
string **unit#=[s,km,km,...]**	第#个轴的单位

sfspike 不用输入数据，输出一个脉冲。是一种产生数据的简便方法。下面是个例子：

```
bash$ sfspike n1=5 n2=3 k1=4 k2=1 | sfdisfil
   0:           0           0           0           1           0
   5:           0           0           0           0           0
```

```
 10:          0          0          0          0          0
```

参数 k1=4 和 k2=1 确定了脉冲的位置。注意位置是从 1 开始的。如果参数之一被忽略了或者置零，相应方向上的脉冲就会变成一个平面：

```
  0:          0          0          0          1          0
  5:          0          0          0          1          0
 10:          0          0          0          1          0
```

如果没有给出脉冲参数，所有的数据就会填充为 1：

```
bash$ sfspike n1=5 n2=3 | sfdisfil
  0:          1          1          1          1          1
  5:          1          1          1          1          1
 10:          1          1          1          1          1
```

为了产生多个脉冲，使用 nsp=参数，并用逗号给出一系列 k#=参数的值：

```
bash$ sfspike n1=5 n2=3 nsp=3 k1=1,3,4 k2=1,2,3 | sfdisfil
  0:          1          0          0          0          0
  5:          0          0          1          0          0
 10:          0          0          0          1          0
```

如果列表中的数据比 nsp 小，那么最后一个值会重复，脉冲会叠加起来，使振幅增加：

```
bash$ sfspike n1=5 n2=3 nsp=3 k1=1,3 k2=1,2 | sfdisfil
  0:          1          0          0          0          0
  5:          0          0          2          0          0
 10:          0          0          0          0          0
```

脉冲的振幅可以用 mag=来明确地确定：

```
bash$ sfspike n1=5 n2=3 nsp=3 k1=1,3,4 k2=1,2,3 mag=1,4,2 | sfdisfil
  0:          1          0          0          0          0
  5:          0          0          4          0          0
 10:          0          0          0          2          0
```

你可以通过使用 1#=参数来建立一个方波：

```
bash$ sfspike n1=5 n2=3 k1=2 l1=4 k2=2 mag=8 | sfdisfil
   0:          0          0          0          0          0
   5:          0          8          8          8          0
  10:          0          0          0          0          0
```

在这种情况下，k1=2 确定了方波的开始位置，11=4 确定了方波的结束位置。

最后多维平面波可以用 p#=参数来表示：

```
bash$ sfspike n1=5 n2=3 k1=2 p2=1 | sfdisfil
   0:          0          1          0          0          0
   5:          0          0          1          0          0
  10:          0          0          0          1          0
```

注意 sfspike 将 p#=参数解释为随采样点呈阶梯状变化，而不是按单位变化的（如 s/m）。

可以定义一个以上的 p 参数。在这种情况下，可以建立一个（高级）平面。

当斜面值不是整数时，会用到简单的线性插值：

```
bash$ sfspike n1=5 n2=3 k1=2 p2=0.7 | sfdisfil
   0:          0          1          0          0          0
   5:          0        0.3        0.7          0          0
  10:          0          0        0.6        0.4          0
```

sfspike 为所有的轴提供了默认的维数和标注：

```
bash$ sfspike n1=5 n2=3 n3=4 > spike.rsf
bash$ sfin spike.rsf
spike.rsf:
    in="/var/tmp/spike.rsf@"
    esize=4 type=float form=native
    n1=5            d1=0.004        o1=0            label1="Time" unit1="s"
    n2=3            d2=0.1          o2=0            label2="Distance" unit2="km"
    n3=4            d3=0.1          o3=0            label3="Distance" unit3="km"
        60 elements 240 bytes
```

在上例中，第一个轴为时间，采样间隔为 0.004s。其它轴为距离采样讲个为 0.1km。所有这些参数可以在命令行中改变。

```
bash$ sfspike n1=5 n2=3 n3=4 label3=Offset unit3=ft d3=20 > spike.rsf
bash$ sfin spike.rsf
spike.rsf:
        in="/var/tmp/spike.rsf@"
        esize=4 type=float form=native
        n1=5            d1=0.004        o1=0            label1="Time" unit1="s"
        n2=3            d2=0.1          o2=0            label2="Distance" unit2="km"
        n3=4            d3=20           o3=0            label3="Offset" unit3="ft"
                60 elements 240 bytes
```

sfspray

Extend a dataset by duplicating in the specified axis dimension. **通过对特定轴和维数的数据进行复制来对数据集进行扩展**		
sfspray < in.rsf > out.rsf axis=2 n= d= o= label= unit=		
操做与 sfstack 紧密联系.		
int	**axis=2**	需要扩展的轴
float	**d=**	新建立的维的采样间隔
string	**label=**	新建立的维的标注
int	**n=**	新建立的维的尺寸
float	**o=**	新建立的维的起点
string	**unit=**	新建立的维的单位

sfspray 通过复制某一维中的数据对高维数据体进行扩展。输出的数据集需要增加一维。下面是例子：从 2 维数据集开始

```
bash$ sfmath n1=5 n2=2 output=x1+x2 > test.rsf
bash$ sfin test.rsf
test.rsf:
        in="/var/tmp/test.rsf@"
        esize=4 type=float form=native
        n1=5            d1=1            o1=0
        n2=2            d2=1            o2=0
                10 elements 40 bytes
bash$ < test.rsf sfdisfil
   0:           0           1           2           3           4
```

```
  5:          1          2          3          4          5
```

在第 2 维中对数据进行扩展

```
bash$ < test.rsf sfspray axis=2 n=3 > test2.rsf
bash$ sfin test2.rsf
test2.rsf:
        in="/var/tmp/test2.rsf@"
        esize=4 type=float form=native
        n1=5            d1=1            o1=0
        n2=3            d2=1            o2=0
        n3=2            d3=1            o3=0
                30 elements 120 bytes
bash$ < test2.rsf sfdisfil
  0:          0          1          2          3          4
  5:          0          1          2          3          4
 10:          0          1          2          3          4
 15:          1          2          3          4          5
 20:          1          2          3          4          5
 25:          1          2          3          4          5
```

输出数据是 3 维的，原始数据沿第二条轴复制。

将数据扩展到第三维

```
Bash$ < test.rsf sfspray axis=3 n=2 > test3.rsf
bash$ sfin test3.rsf
test3.rsf:
        in="/var/tmp/test3.rsf@"
        esize=4 type=float form=native
        n1=5            d1=1            o1=0
        n2=2            d2=1            o2=0
        n3=2            d3=?            o3=?
                20 elements 80 bytes
bash$ < test3.rsf sfdisfil
  0:          0          1          2          3          4
  5:          1          2          3          4          5
 10:          0          1          2          3          4
 15:          1          2          3          4          5
```

输出也是三维的，但这次是初始数据沿第三条轴复制。

sfstack

Stack a dataset over one of the dimensions. **在一个维上叠加数据集**			
sfstack < in.rsf > out.rsf axis=2 rms=n norm=y min=n max=n			
这个操作与 sfspray 紧密联系。			
int	**axis=2**		要叠加的轴
bool	**max=n**	[y/n]	如果是 y，找到最大值，不必叠加。忽略均方根和范数。
bool	**min=n**	[y/n]	如果是 y，找到最小值，不必叠加。忽略均方根和范数。
bool	**norm=y**	[y/n]	如果是 y，通过折叠来归一化。
bool	**rms=n**	[y/n]	如果是 y，计算均方根，不必叠加。

sfspray 在高维数据体上增加了一维，sfstack 通过在一维上叠加来对这一维进行有效地移除。下面是一个例子：

```
bash$ sfmath n1=5 n2=3 output=x1+x2 > test.rsf
bash$ < test.rsf sfdisfil
    0:          0          1          2          3          4
    5:          1          2          3          4          5
   10:          2          3          4          5          6
bash$ < test.rsf sfstack axis=2 | sfdisfil
    0:        1.5          2          3          4          5
bash$ < test.rsf sfstack axis=1 | sfdisfil
    0:        2.5          3          4
? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
```

为什么第一个值不是 1（第一种情况）或 2（第二种情况）？sfstack 用折叠来进行归一化（非零输入），为了避免归一化，使用 norm=n，如下：

```
bash$ < test.rsf sfstack norm=n | sfdisfil
    0:          3          6          9         12         15
```

sfstack 可以用来计算均方根和最小或最大值。

```
bash$ < test.rsf sfstack rms=y | sfdisfil
    0:          1.581          2.16          3.109          4.082          5.066
bash$ < test.rsf sfstack min=y | sfdisfil
    0:              0              1              2              3              4
bash$ < test.rsf sfstack axis=1 max=y | sfdisfil
    0:              4              5              6
```

sftransp

Transpose two axes in a dataset. **对数据集中的两个轴进行交换**		
sftransp < in.rsf > out.rsf memsize=sf_memsize() plane=		
如果发现“cannot allocate memory”的错误，在程序上添加 memsize=1 的命令行来强制在核外执行操作。		
int	**memsize=sf_memsize()**	需要用到的最大内存空间（Mb）
int	**plane=**	两个数字的表示的需要交换的轴，默认值是 12.

sftransp 程序将由 plane=参数确定的两个轴进行交换。

```
bash$ sfspike n1=10 n2=20 n3=30 > orig123.rsf
bash$ sfin orig123.rsf
orig123.rsf:
    in="/var/tmp/orig123.rsf@"
    esize=4 type=float form=native
    n1=10          d1=0.004       o1=0           label1="Time" unit1="s"
    n2=20          d2=0.1         o2=0           label2="Distance" unit2="km"
    n3=30          d3=0.1         o3=0           label3="Distance" unit3="km"
        6000 elements 24000 bytes
bash$ <orig123.rsf sftransp plane=23 >out132.rsf
bash$ sfin out132.rsf
out132.rsf:
    in="/var/tmp/out132.rsf@"
    esize=4 type=float form=native
    n1=10          d1=0.004       o1=0           label1="Time" unit1="s"
    n2=30          d2=0.1         o2=0           label2="Distance" unit2="km"
    n3=20          d3=0.1         o3=0           label3="Distance" unit3="km"
```

```
        6000 elements 24000 bytes
bash$ <orig123.rsf sftransp plane=13 >out321.rsf
bash$ sfin out321.rsf
out321.rsf:
    in="/var/tmp/out132.rsf@"
    esize=4 type=float form=native
    n1=30          d1=0.1          o1=0          label1="Distance" unit1="km"
    n2=20          d2=0.1          o2=0          label2="Distance" unit2="km"
    n3=10          d3=0.004        o3=0          label3="Time" unit3="s"
        6000 elements 24000 bytes
```

sftransp 试图将数据集送到内存再进行交换，但是如果空间不足，它会占用硬盘空间，导致运算速度下降。你可以通过 memsize=参数或者 RSFMEMSIZE 环境变量来调节所用的空间。

sfwindow

Window a portion of the dataset. **对数据集的一部分加窗**			
sfwindow < in.rsf > out.rsf verb=n squeeze=y j#=(1,...) d#=(d1,d2,...) f#=(0,...) min#=(o1,o2,,...) n#=(0,...) max#=(o1+(n1-1)*d1,o2+(n1-1)*d2,,...)			
float	**d#=(d1,d2,...)**		对第#个轴采样
int	**f#=(0,...)**		n 窗口开始于第#维
int	**j#=(1,...)**		跳到第#维
float	**max#=(o1+(n1-1)*d1,o2+(n1-1)*d2,,...)**		第#维的最大值
float	**min#=(o1,o2,,...)**		第#维的最小值
int	**n#=(0,...)**		第#维的窗口的尺寸
bool	**squeeze=y**	[y/n]	如果是 y，将数据体所有维压缩成 1。
bool	**verb=n**	[y/n]	冗余标志

sfwindow 用来对数据集的一部分加窗（就是保留这部分值）。下面是个简单的例子：首先创建一些数据

```
bash$ sfmath n1=5 n2=3 o1=1 o2=1 output="x1*x2" > test.rsf
bash$ < test.rsf sfdisfil
    0:          1          2          3          4          5
    5:          2          4          6          8         10
```

```
  10:          3          6          9         12         15
```

现在对前两行加窗口：

```
bash$ < test.rsf sfwindow n2=2 | sfdisfil
  0:          1          2          3          4          5
  5:          2          4          6          8         10
```

对前三列加窗口：

```
bash$ < test.rsf sfwindow n1=3 | sfdisfil
  0:          1          2          3          2          4
  5:          6          3          6          9
```

对中间的行加窗：

```
bash$ < test.rsf sfwindow f2=1 n2=1 | sfdisfil
  0:          2          4          6          8         10
```

可以将 f#和 n#理解为“跳过这么多行/列”和“选择这么多行/列”。对中间的点进行加窗：

```
bash$ < test.rsf sfwindow f1=2 n1=1 f2=1 n2=1 | sfdisfil
  0:          6
```

每隔一列：

```
bash$ < test.rsf sfwindow j1=2 | sfdisfil
  0:          1          3          5          2          6
  5:         10          3          9         15
```

每三列：

```
bash$ < test.rsf sfwindow j1=3 | sfdisfil
  0:          1          4          2          8          3
  5:         12
```

同样地，sfwindow 用最少的或最多的参数选定一个窗口。在下例中，我们用 sfspike 建立了一个数据集，然后在时间为 1 到 2 秒之间的数据上加窗，进行 8 毫秒采样。

```
bash$ sfspike n1=1000 n2=10 > spike.rsf
bash$ sfin spike.rsf
spike.rsf:
    in="/var/tmp/spike.rsf@"
    esize=4 type=float form=native
    n1=1000         d1=0.004        o1=0            label1="Time" unit1="s"
    n2=10           d2=0.1          o2=0            label2="Distance" unit2="km"
        10000 elements 40000 bytes
bash$ < spike.rsf sfwindow min1=1 max1=2 d1=0.008 > window.rsf
bash$ sfin window.rsf
window.rsf:
    in="/var/tmp/window.rsf@"
    esize=4 type=float form=native
    n1=126          d1=0.008        o1=1            label1="Time" unit1="s"
    n2=10           d2=0.1          o2=0            label2="Distance" unit2="km"
        1260 elements 5040 bytes
```

通常，sfwindow 默认将高维数据体等于 1 的维进行压缩。下面是用时间切片展示的例子：

```
bash$ < spike.rsf sfwindow n1=1 min1=1 > slice.rsf
bash$ sfin slice.rsf
slice.rsf:
    in="/var/tmp/slice.rsf@"
    esize=4 type=float form=native
    n1=10           d1=0.1          o1=0            label1="Distance" unit1="km"
    n2=1            d2=0.004        o2=1            label2="Time" unit2="s"
        10 elements 40 bytes
```

可以通过确定 squeeze=n 来改变这种情况。

```
bash$ < spike.rsf sfwindow n1=1 min1=1 squeeze=n > slice.rsf
bash$ sfin slice.rsf
slice.rsf:
    in="/var/tmp/slice.rsf@"
```

```
esize=4 type=float form=native
n1=1        d1=0.004    o1=1    label1="Time" unit1="s"
n2=10       d2=0.1      o2=0    label2="Distance" unit2="km"
    10 elements 40 bytes
```

? ?

4.1.2 地震指令 Seismic programs

此类程序是专门用来处理地震数据的。这些程序的源文件可以从 Madagaascar 的软件的文件 system/seismic 找到

sffkamo

Computes Azimuth Move-Out (AMO) operator in the f-k log-stretch domain、计算 f-k 对数拉伸域方位时差算子		
sffkamo < in.rsf > out.rsf h1= h2= f1= f2= maxe=10.		
float	**f1=**	输入方位（度）
float	**f2=**	输出方位（度）
float	**h1=**	输入炮检距（偏移距）
float	**h2=**	输出炮检距（偏移距）
float	**maxe=10.**	稳定性约束条件

工作流程参考 SEP-110, 63-70 (2001),这其中包含了对输入数据的带通滤波。

创建输入文件——a(t,x,y)共偏移距数据体：

```
sfspike \
n1=128 o1=0.4 d1=0.0032 k1=65 label1=t \
n2=256 o2=-1.536 d2=0.012 k2=129 label2=x \
n3=128 o3=-1.024 d3=0.016 k3=65 label3=y | \
sfbandpass flo=5 fhi=60 > spikebps.rsf
```

使用对数拉伸 FFT：

```
<spikebps.rsf sfstretch rule=L dens=4 |\
sffft1 |\
sffft3 axis=2 |\
sffft3 axis=3 > spikefft3.rsf
```

计算 spikefft3.rsf 文件中的 AMO 算子。由 stdin 获取的信息是 n，o，d 参数：

```
<spikefft3.rsf sffkamo h2=2 f2=10 h1=1.8 f1=30 >oper.rsf
```

将算子相乘并用 fft 计算得到(t,mx,my)

```
< spikefft3.rsf sfadd mode=prod oper.rsf |\
sffft3 axis=3 inv=y |\
sffft3 axis=2 inv=y |\
sffft1 inv=y |\
sfstretch rule=L dens=4 inv=y > spikeamo.rsf
```

为进行 8 位的灰度显示作准备：

```
< spikeamo.rsf sfbyte pclip=100 gainpanel=a > spikebyte.rsf
```

脉冲响应中间部分的图像：

```
<spikebyte.rsf sfgrey3 frame1=65 frame2=129 frame3=65 \
point1=0.333 title='AMO saddle, no f-k filter' | sfpen &
```

含有假象的图像（如需要进行 f-k 滤波）：

```
< spikebyte.rsf sfgrey3 frame1=65 frame2=97 frame3=97 \
point1=0.333 title='No f-k filter' | sfpen &
```

使用 f-k 滤波，然后再显示图像：

```
< spikeamo.rsf sffft1 |\
sffft3 axis=2 |\
sffft3 axis=3 |\
sfdipfilter v1=-2.5 v2=-1.5 v3=1.5 v4=2.5 taper=2 pass=0 dim=3 |\
sffft3 axis=3 inv=y |\
sffft3 axis=2 inv=y |\
sffft1 inv=y |\
sfbyte pclip=100 gainpanel=a |\
sfgrey3 frame1=65 frame2=97 frame3=97 point1=0.333 title='With f-k filter' |\
sfpen &
```

sfheaderattr

Integer header attributes. 整型头文件属性
sfheaderattr < head.rsf
只报告非零值

sfheaderattr 检查道头的内容，道头文件通常是由 sfseyread 读取的。在下面的例子中我们检测了 SU 中的一个输出文件 suplane 的道头。

```
bash$ suplane > plane.su
bash$ sfsegyread tape=plane.su su=y tfile=tfile.rsf > plane.rsf
bash$ sfheaderattr < tfile.rsf
****************************************
71 headers, 32 traces
key[0]="tracl"          min[0]=1              max[31]=32            mean=16.5
key[1]="tracr"          min[0]=1              max[31]=32            mean=16.5
key[11]="offset"        min[0]=400            max[31]=400           mean=400
key[38]="ns"            min[0]=64             max[31]=64            mean=64
key[39]="dt"            min[0]=4000           max[31]=4000          mean=4000
****************************************
```

对于不同的关键字，minimum,maximum,和 mean values 都会被分别报告出来，除非它们都是 0.通过这样快速的检查，可以在数据体中确定重要的关键字。输入文件必须为整型 int。

sfheadermath

Mathematical operations, possibly on header keys. 道头关键字上的数学操作
sfheadermath < in.rsf > out.rsf memsize=sf_memsize() output=
已知的函数: cos, sin, tan, acos, asin, atan, cosh, sinh, tanh, acosh, asinh, atanh,

exp, log, sqrt, abs 参考 sfmath. 其它的操作可以用 sfstack 来实现。		
int	**memsize=sf_memsize()**	需要的最大内存（Mb）
string	**output=**	用数学记号表示输出。

sfheadermath 是对输入文件的行数据进行数学操作的多功能的程序。如果输入文件是一个 n1×n2 的矩阵，输出文件会是一个含有通过其它行数学计算得到的一行数据，成为一个 1×n2 的矩阵。Sfheadermath 可以通过数字或者标准的 SEGY 关键字来确定行。后者对于处理由 SEGY 或 SU 文件获得的道头很有用。下面是个例子，首先，我们用 suplane 创建一个 SU 文件并用 sfsegyread 将其转换成 RSF 文件。

```
bash$ suplane > plane.su
bash$ sfsegyread tape=plane.su su=y tfile=tfile.rsf > plane.rsf
```

道头信息存储在 tfile.rsf 文件中。包含 32 道的 71 个整型形式的道头

```
bash$ sfin tfile.rsf
tfile.rsf:
    in="/tmp/tfile.rsf@"
    esize=4 type=int form=native
    n1=71           d1=?            o1=?
    n2=32           d2=?            o2=?
        2272 elements 9088 bytes
```

然后，将 tfile.rsf 转换成实型的，并运行 sfheadermath 来产生一个新道头。

```
bash$ < tfile.rsf sfdd type=float | \
sfheadermath myheader=1 output="sqrt(myheader+(2+10*offset^2))" > new.rsf
bash$ sfin new.rsf
new.rsf:
    in="/tmp/new.rsf@"
    esize=4 type=float form=native
    n1=1            d1=?            o1=?
```

```
n2=32           d2=?            o2=?
        32 elements 128 bytes
```

我们将“myheaders”定义为输入文件的第一行（注意数字以 0 开始）并与偏移距联系起来，偏移距是一个定义行号 11 标准的 SEGY 关键字（注意 sfheaderattr 的输出文件）许多数学表达都可以在 output=字符串中定义，这些表达的工具是与 sfmath 共享的。

sfsegyheader

Make a trace header file for segywrite. **为写入 segy 数据创建一个头文件**		
sfsegyheader < in.rsf > out.rsf n1= d1=		
Use the output for tfile= argument in segywrite 在写入 segy 数据时使用输出作为 tfile=参数.		
float	**d1=**	每道采样间隔
int	**n1=**	一道的采样点数

sfsegyread

Convert a SEG-Y or SU dataset to RSF. **将 SEG-Y 或 SU 数据集转换成 RSF 形式。**
sfsegyread mask=msk.rsf > out.rsf tfile=hdr.rsf verb=n su= suxdr=n endian=y format=segyformat (bhead) ns=segyns (bhead) tape= read= hfile= bfile=
Data headers and trace headers are separated from the data. 数据头文件和道头文件与数据是分开的。 "suread" is equivalent to "segyread su=y" “suread”相当于“segyread su=y” SEGY key names: SEGY 关键名称： tracl: trace sequence number within line 0 在 0 行中的道顺序号

tracr: trace sequence number within reel 4 在卷 4 中的道序号

fldr: field record number 8 野外记录号 8

tracf: trace number within field record 12 野外记录中的道号

ep: energy source point number 16 震源点号 16

cdp: CDP ensemble number 20 CDP 号 20

cdpt: trace number within CDP ensemble 24 CDP 道集中道号 24

trid: trace identification code:道识别码：
1 = seismic data 地震数据
2 = dead 废道
3 = dummy 哑炮
4 = time break 初至时刻
5 = uphole 沿井向上
6 = sweep **扫描**
7 = timing **测定时**间，时序
8 = water break **水中爆炸时刻**
9---, N = optional use (N = 32,767) 28

nvs: number of vertically summed traces 30 垂向叠加道数 30

nhs: number of horizontally summed traces 32 水平叠加道数 32

duse: data use:数据使用：
1 = production 生产
2 = test 测试 34

offset: distance from source point to receiver 炮检距：炮点到检波点的距离
group (negative if opposite to direction
in which the line was shot) 36 组合（如果与放线方向相反则为负）

gelev: receiver group elevation from sea level
(above sea level is positive) 40 接收组合的海拔（高于海平面为正）

selev: source elevation from sea level 震源高程（高于海平面为正）
(above sea level is positive) 44

sdepth: source depth (positive) 48 震源深度（正）

gdel: datum elevation at receiver group 52 接收组合的数据高程

sdel: datum elevation at source 56 震源的数据高程

swdep: water depth at source 60 震源处的水深

gwdep: water depth at receiver group 64 接收组合处的水深

scalel: scale factor for previous 7 entries 前 7 个输入的尺度变换因子，在 0，1，2，3，4 次方的基础上加或减 10（如果为正则乘，为负则除）
with value plus or minus 10 to the 接着 4 个输入的尺度变换因子，在 0，1，2，3，4 次方的基础上加或减 10（如果为正则乘，为负则除）
power 0, 1, 2, 3, or 4 (if positive,
multiply, if negative divide) 68

scalco: scale factor for next 4 entries 接着 4 个输入的尺度变换因子，在 0，1，2，3，4 次方的基础上加或减 10（如果为正则乘，为负则除）
with value plus or minus 10 to the
power 0, 1, 2, 3, or 4 (if positive,
multiply, if negative divide) 70

sx: X source coordinate 72 震源 X 坐标

sy: Y source coordinate 76 震源 Y 坐标

gx: X group coordinate 80 组合 X 坐标

gy: Y group coordinate 84 组合 Y 坐标

counit: coordinate units code:坐标单位代码：
for previous four entries 对于前四个输入：
1 = length (meters or feet)长度（米或英尺）
2 = seconds of arc (in this case, the
X values are unsigned longitude and the Y values
are latitude, a positive value designates
the number of seconds east of Greenwich
or north of the equator 88 用秒表示的弧度（其中，X 值是无符号的经度，Y 是纬度，正值表示在格林威治移动或赤道以北）

wevel: weathering velocity 90 风化层速度

swevel: subweathering velocity 92 未风化层速度

sut: uphole time at source 94 震源处沿井上传时间

gut: uphole time at receiver group 96 检波器组合处沿井上传时间

sstat: source static correction 98 震源静校正值

gstat: group static correction 100 组合静校正值

tstat: total static applied 102 总的静校正值

laga: lag time A, -延迟时间 A，用毫秒表示的在 240 个字节的道头和初至之间。如果初至在道头之后，初至定义为初始脉冲，这个脉冲可能记录在一个辅助道或者由记录系统确定。104

lagb: lag time B, time in ms between the time
break and the initiation time of the energy source,
may be positive or negative 106 延迟时间 B，用毫秒表示的初至与震源激发时刻的时差，可能是正值也可能是负值。

delrt: delay recording time, time in ms between
initiation time of energy source and time
when recording of data samples begins

(for deep water work if recording does not
start at zero time) 108 记录延迟时间，单位是 ms，表示震源激发时刻与开始采样点之间的时差（深水作业记录时刻不从零时刻开始时）

muts: mute time--start 110 切除开始时刻

mute: mute time--end 112 切除结束时刻

ns: number of samples in this trace 114 这一道的采样点数

dt: sample interval, in micro-seconds 116 采样间隔，单位是微秒

gain: gain type of field instruments code:野外设备增益类型代码：
1 = fixed 固定
2 = binary 二进制
3 = floating point 浮点型
4 ---- N = optional use 118

igc: instrument gain constant 120 装置增益常数

igi: instrument early or initial gain 122 装置初始增益

corr: correlated:相关
1 = no
2 = yes 124

sfs: sweep frequency at start 126 开始时的扫描频率

sfe: sweep frequency at end 128 结束时的扫描频率

slen: sweep length in ms 130 扫描时窗长度（毫秒）

styp: sweep type code:扫描类型代码
1 = linear 线性
2 = cos-squared 余弦平方
3 = other 132 其它

stas: sweep trace length at start in ms 134 扫描道的长度开始时刻（毫秒）

stae: sweep trace length at end in ms 136 扫描道的结束时刻（毫秒）

tatyp: taper type: 1=linear, 2=cos^2, 3=other 138 衰减类型

afilf: alias filter frequency if used 140 是否用假频滤波器

afils: alias filter slope 142 假频滤波器斜率

nofilf: notch filter frequency if used 144 是否使用陷波器

nofils: notch filter slope 146 陷波器斜率

lcf: low cut frequency if used 148 是否用低阻滤波器

hcf: high cut frequncy if used 150 是否用高阻滤波器

lcs: low cut slope 152 低阻滤波器斜率

hcs: high cut slope 154 高阻滤波器斜率

year: year data recorded 156 记录的年代

day: day of year 158 一年中的哪一天

hour: hour of day (24 hour clock) 160 一天中的哪一时刻（24 小时）

minute: minute of hour 162 分钟

sec: second of minute 164 秒

timbas: time basis code:基本时间代码：
1 =当地时间
2 = GMT 国际通用时间

3 = 其他时间 166

trwf: trace weighting factor, defined as 1/2^N
volts for the least sigificant bit 168
道加权因子，定义为最弱的二进制位的 1/2^N

grnors: geophone group number of roll switch
position one 170
在第一个卷转换处的检波器组合数

grnofr: geophone group number of trace one within
original field record 172
初始野外记录第一道的检波器组合号

grnlof: geophone group number of last trace within
original field record 174
初始野外记录的最后一道的检波器组合号

gaps: gap size (total number of groups dropped) 176
去掉道的组合总数

otrav: **overtravel** taper code:
1 = down (or behind) 向下
2 = up (or ahead) 向上 178

cdpx: X coordinate of CDP 180
CDP 点处的 X 坐标
cdpy: Y coordinate of CDP 184
CDP 点处的 Y 坐标

iline: in-line number 188
纵测线号

xline: cross-line number 192
横测线号
shnum: shotpoint number 196

炮点号

shsca: shotpoint scalar 200
炮点标量

tval: trace value meas. 202
测量的道的值

tconst4: transduction const 204
转换常数

tconst2: transduction const 208
转换常数

tunits: transduction units 210
转换单位

device: device identifier 212
设备

tscalar: time scalar 214
时间标量

stype: source type 216
震源类型

sendir: source energy dir. 218
震源能量 dir（方向）

unknown: unknown 222

smeas4: source measurement 224
震源测量值

smeas2: source measurement 228
震源测量值

smeasu: source measurement unit 230 震源测量值单位 unass1: unassigned 232 未分配 unass2: unassigned 236 未分配			
string	**bfile=**		输出二进制文件的头文件
bool	**endian=y**	[y/n]	Whether to automatically estimate endianness or not 是否自动估计**是大端还是小端**
int	**format=segyformat (bhead)**	[1,2,3,5]	Data format. The default is taken from binary header. 数据格式，二进制头文件有默认值 1 IBM 实型 2 4 字节整型 3 2 字节整型 5 IEEE 实型
string	**hfile=**		output text data header file 用文本形式输出头文件信息
string	**mask=**		optional header mask for reading only selected traces (auxiliary input file name) 为了只读取选定的道信息所用的可选道头指令（辅助输入文件名）
int	**ns=segyns (bhead)**		Number of samples. The default is taken from binary header 采样点数量，二进制头文件中有默认值
string	**read=**		what to read: h - header, d - data, b - both (default) 读取 h——道头，b——全部读取（默认）
bool	**su=**	[y/n]	y—输入文件为 SU, n –输入文件为 SEGY
bool	**suxdr=n**	[y/n]	y, SU 有 XDR 支持
string	**tape=**		输入数据
string	**tfile=**		output trace header file (auxiliary output file name) 输出道头文件（辅助输出文件名）
bool	**verb=n**	[y/n]	冗余标志

SEGY 格式是进行地球物理数据交换的公开的标准格式，它是由非营利性的 SEG 技术标准委员会制定。这个标准有两个版本：rev0 (1975)[3] 和 rev1 (2002)[4]。Sfsegyread 中即用到了 rev0（如，没有对扩展的文本类型的道头检测），又用到了 rev1（用 IEEE 实型数据格式 对地震道数据进行采样）。

Sfsegyread 可以识别的 SEG-Y 文件含有一个“卷头”（3200 字节，之后是用二进制编码的 400 个字节的数据），之后是一系列地震道数据，每一个地震道包含了 240 字节的道头（二进制）和振幅数据——一系列采样点（以 ns 为单位）。道头和卷头中的二进制值都是一些用双数补齐的整数，或者为 2 个字节，或者为 4 个字节，头文件中没有实型数据。地震道的振幅数据可以有不同的编码形式，可以是实型，也可以是整型，**讲得再深一点，因为它们都是大端的.**。为了将 SEG-Y 文件转换成 RSF 文件，sfsegyread 会找出 EBCDIC 卷头信息并将其转化为 ASCII 格式，提取出 2 进制卷头不作改动，将道头放到一个 RSF 文件中，将道数据放到另一个 RSF 文件中。

SEG-Y Trace Headers

在 SEG-Y 数据标准中，仅对 240 个字节道头的 180 个字节定义；181-240 字节被保留用来存储非标准道头信息,这些空间越来越多地用于现代 SEG-Y 文件及其变量。这个标准总共提供了 714 个字节及 2 个字节的定义好的道头文件。这 71 个标准的词有定义好的长度，并且只有这些词和字节位置可以使用 segyread 读取，并输出带 file=选项的 RSF 道头文件。使用者可以重新定位这些关键字来用于不同的字节位置。

SU File Format

SU 文件就是不带卷头的 SEG-Y 文件，当创建 SU 文件时。道数据文件会在 CPU 内部进行编码（注意——可移植性受到限制）。因此，在将 SU 文件转换成 RSF 文件时，sfsegyread 只是将道头和数据道分成两个 RSF 文件。

SEG-Y specific parameters

hfile= 确定了当 EBCDIC 卷头信息所在被转换为 ASCII 码形式后索要存储的文件名。如果你确定其中没有有用信息，用 hfile=/dev/nul 就可以了。如果你没有对这个参数指定任何信息，你会在当前目录下得到一个名为 header 的 ASCII 文件。如果你想在运行 sfsegyread 之前快速地预览这个道头，用

```
dd if=input.segy count=40 bs=80 cbs=80 conv=unblock,ascii
```

bfile= 指定了二进制卷头（3600 字节的 EBCDIC 卷头之后的 400 字节）直接存到的文件的文件名，默认的文件名是“binary”。除非你有软件可以准确读取这种特殊类型的文件，否则就无法使用它，对于 bfile=/dev/null 也是这样。

format=指定了 SEG-Y 输入文件中地震道的采样数据的类型。这是从二进制的 SEG-Y 文件的卷头读取的。有效的值为 1（IBM 实型），2（4 个字节整型），3（2 个字节整型）和 5（IEEE 实型）。如果输入文件是 SU，这个格式是原始的实型格式。

keyname= 指定了使用上述的道头关键字来对头文件进行重新定位时距离文件开始的字节数。例如如果 CDP 的位置是在第 181-184 个字节，而不是 21-24 个字节，cdp=180 会将道头信息一道那个位置。

SU-specific parameters

suxdr= 指定了输入文件是否使用 XDR 支持的 SU 文件包创建。如果可以找到你安装 SU 的源代码（试试 $CWPROOT/src），使用：grep 'XDRFLAG =' $CWPROOT/src/Makefile.config 来看看最后一个没有注释的条目。如果没有给 XDRFLAG 赋值，这个软件包没有用 XDR 支持进行编译。

Common parameters

su= 指定了输入文件是 SU 格式还是 SEG-Y 格式，默认值是 su=n(SEG-Y file)。

tape= 指定了输入数据。由于 sfsegyread 工作需要**磁带并在输入文件上来回读取数据，**Stdin 无法使用。幸运的是，输出文件在 stdout 中。

read= 指定了需要读取的“道数据块”。可以用 read=t (仅读取道数据), read=h (仅读取道头数据) or read=b (两者都读取).

tfile= 给出了要将道头文件写入的 RSF 文件的名称。明显地，它只能用 read=h 或 read=b.

mask= 是一个用来确定 mask 名称的可选的参数。Mask 是一个含整型变量 的 1 维 RSF 文件。Mask 中的采样点数与 unmasked SEG-Y 中相同。在不符合条件的地震道存在的地方，mask 中是零。

ns= 指定了一道中采样点的个数。对于 SEG-Y 文件，默认值是从二进制卷头中获得的，对于 SU 文件，是从第一道的道头中获得的。这个参数非常重要，因为含有**一条抽象实现的命令行** command line override。

verbose=是冗余标志，可以为 y 或 n。

endian=是一个 y/n 标志（默认为 y），指定了是否要自动判断道数据中的存储方式是由低位到高位还是由高位到低位。如果你遇到了问题，不知道如何继续，可以试试这个指令，否则就让它自动判断好了。

sfsegywrite

Convert an RSF dataset to SEGY or SU. **将 RSF 数据集转换为 SEG-Y 或 SU 格式**			
sfsegywrite < in.rsf tfile=hdr.rsf verb=false su=false endian=sf_endian() tape= hfile= bfile=			
将道头和数据合并到一起			
string	**bfile=**		input binary data header file 输入二进制数据的道头文件
bool	**endian=sf_endian()**	[y/n]	big/little endian flag. The default is estimated automatically

			判断是否由低位到高位的标志，默认是自动判断。
string	**hfile=**		input text data header file 输入文本格式的数据文件道头。
bool	**su=n**	[y/n]	y if output is SU, n if output is SEGY 如果为 SU 则选 y，如果是 SEG-Y，则选 n
string	**tape=**		
bool	**verb=n**	[y/n]	冗余标志

要查看参数的意义和相关内容，请参考 sfsegyread。仅当文件为 SEG-Y（默认）时需要对参数 bfile 和 hfile 赋值。输出文件由 tape=标签确定。

4.1.4 绘图指令 Plotting programs（stable）

这些程序的原文件可以从 Madagascar 版本的 plot/main 文件夹获得。

sfbox

Draw a balloon-style label. **绘制一个 balloon 类型的标签**			
sfbox lab_color=VP_WHITE lab_fat=0 pscale=1. pointer=y reverse=n lat=0. long=90. angle=0. x0=0. y0=0. scale0=1. xt=2. yt=0. x_oval=0. y_oval=0. boxit=y length= scalet= size=.25 label= > out.vpl			
float	**angle=0.**		longitude of floating label in 3-D 3 维中实型标签的经度
bool	**boxit=y**	[y/n]	if y, create a box around text 如果为 y，在文本周围加框
int	**lab_color=VP_WHITE**		label color 标签颜色
int	**lab_fat=0**		label fatness 标签宽度
string	**label=**		text for label 标签的文字
float	**lat=0.**		
float	**length=**		normalization for xt and yt 对 xt 和 yt 归一化
float	**long=90.**		latitude and longitude of viewpoint in 3-D 三维中观测点的纬度和经度
bool	**pointer=y**	[y/n]	if y, create arrow pointer 如果是 y，绘制箭头符号
float	**pscale=1.**		scale factor for width of pointer 指针宽的的尺度参数

bool	**reverse=n**	[y/n]	
float	**scale0=1.**		scale factor for x0 and y0 x0 和 y0 的尺度因子
float	**scalet=**		
float	**size=.25**		text height in inches 用英寸表示的文本的高度
float	**x0=0.**		
float	**x_oval=0.**		
float	**xt=2.**		
float	**y0=0.**		position of the pointer tip 指针尖的位置
float	**y_oval=0.**		size of the oval around pointer 指针周围椭圆的尺寸
float	**yt=0.**		relative position of text 文字的相对位置

sfcontour

Contour plot. **绘制等值线**			
sfcontour < in.rsf c= min1=o1 min2=o2 max1=o1+(n1-1)*d1 max2=o2+(n2-1)*d2 nc=50 dc= c0= transp=y minval= maxval= allpos=y barlabel= > plot.vpl			
运行“sfdoc stdplot”来获得更多的参数			
bool	**allpos=y**	[y/n]	仅对正值绘制等值线
string	**barlabel=**		
floats	**c=**		[nc]
float	**c0=**		第一条等值线
float	**dc=**		等值线间隔
float	**max1=o1+(n1-1)*d1**		
float	**max2=o2+(n2-1)*d2**		要绘制的数据窗口
float	**maxval=**		maximum value for scalebar (default is the data maximum) 尺度条的最大值（默认的是最大值）
float	**min1=o1**		
float	**min2=o2**		

float	**minval=**		minimum value for scalebar (default is the data minimum) 尺度条的最小值（默认的是最小值）
int	**nc=50**		number of contours 等值线的数量
bool	**transp=y**	[y/n]	if y, transpose the axes 如果是 y，对轴进行转秩

sfdots

Plot signal with lollipops. **使用 lollipops 绘制标志**			
sfdots < in.rsf labels= dots=(n1 <= 130)? 1: 0 seemean=(bool) (n2 <= 30) strings=(bool) (n1 <= 400) connect=1 corners= silk=n gaineach=y labelsz=8 yreverse=n constsep=n seedead=n transp=n xxscale=1. yyscale=1. clip=-1. overlap=0.9 screenratio=VP_SCREEN_RATIO screenht=VP_STANDARD_HEIGHT screenwd=screenhigh / screenratio radius=dd1/3 label1= unit1= title= > plot.vpl			
float	**clip=-1.**		data clip 数据修正
int	**connect=1**		connection type: 1 - diagonal, 2 - bar, 4 - only for non-zero data 连接类型：1-对角线，2-条状，4-仅用于非零数据
bool	**constsep=n**	[y/n]	if y, use constant trace separation 如果是 y，使用常数的道间隔
int	**corners=**		number of polygon corners (default is 6) 多边形的角的个数（默认值是 6）
int	**dots=(n1 <= 130)? 1: 0**		type of dots: 1 - baloon, 0 - no dots, 2 - only for non-zero data 点的类型：1-气球型，0-没有点，2-仅用于非零数据
bool	**gaineach=y**	[y/n]	if y, gain each trace independently 如果是 y，独立获得每一道
string	**label1=**		
strings	**labels=**		trace labels [n2]道标记
int	**labelsz=8**		label size 标记的尺寸
float	**overlap=0.9**		trace overlap 道重叠的部分
float	**radius=dd1/3**		dot radius 点的半径
float	**screenht=VP_STANDARD_HEIGHT**		screen height 屏幕的高度

float	**screenratio=VP_SCREEN_RATIO**		screen aspect ratio 屏幕纵横比
float	**screenwd=screenhigh / screenratio**		screen width 屏幕宽度
bool	**seedead=n**	[y/n]	if y, show zero traces 如果是 y，显示为零的道
bool	**seemean=(bool) (n2 <= 30)**	[y/n]	if y, draw axis lines 如果是 y，绘制轴线
bool	**silk=n**	[y/n]	if y, silky plot 如果是 y，绘制柔和的图
bool	**strings=(bool) (n1 <= 400)**	[y/n]	if y, draw strings 如果是 y，绘制条带
string	**title=**		
bool	**transp=n**	[y/n]	if y, transpose the axis 如果是 y，对轴进行转秩
string	**unit1=**		
float	**xxscale=1.**		x scaling x 尺度变换
bool	**yreverse=n**	[y/n]	if y, reverse y axis 如果是 y，将 y 轴反转
float	**yyscale=1.**		y scaling y 尺度变换

sfgraph3

Generate 3-D cube plot for surfaces. **用 3 维数据体生成一个面**			
sfgraph3 < in.rsf orient=1 min= max= point1=0.5 point2=0.5 frame1=0.5*(min+max) frame2=n1-1 frame3=0 movie=0 dframe=1 n1pix=n1/point1+n3/(1.-point1) n2pix=n2/point2+n3/(1.-point2) flat=y > plot.vpl			
float	**dframe=1**		frame increment in a movie 用动画显示时，画面的步长
bool	**flat=y**	[y/n]	if n, display perspective view 如果是 n，显示俯视图
float	**frame1=0.5*(min+max)**		
int	**frame2=n1-1**		
int	**frame3=0**		frame numbers for cube faces 立体表面的画面数目
float	**max=**		maximum function value 最大的函数值
float	**min=**		minimum function value

			最小函数值
int	**movie=0**		0:没有动画, 1: 沿轴 1 的动画, 2: 沿轴 2 的动画, 3: 沿轴 3 的动画
int	**n1pix=n1/point1+n3/(1.-point1)**		number of vertical pixels 垂向的像素个数
int	**n2pix=n2/point2+n3/(1.-point2)**		number of horizontal pixels 水平像素个数
int	**orient=1**		function orientation 函数的方向
float	**point1=0.5**		fraction of the vertical axis for front face 显示前面时，所用垂向轴的比例
float	**point2=0.5**		fraction of the horizontal axis for front face 显示前面时，所用横向轴的比例。

sfgraph

Graph plot. **绘制图表**			
sfgraph < in.rsf symbolsz= pclip=100. transp=n symbol= > plot.vpl			
运行“sfdoc stdplot”来查看更多的参数			
float	**pclip=100.**		clip percentile
string	**symbol=**		if set, plot with symbols instead of lines 如果设置了，就用符号而不用曲线
floats	**symbolsz=**		符号的尺寸 (默认值为 2) [n2]
bool	**transp=n**	[y/n]	if y, transpose the axes 如果是 y，对轴反转

sfgrey3

Generate 3-D cube plot. **绘制 3 为数据体**			
sfgrey3 < in.rsf point1=0.5 point2=0.5 frame1=0 frame2=n2-1 frame3=0 movie=0 dframe=1 n1pix=n1/point1+n3/(1.-point1) n2pix=n2/point2+n3/(1.-point2) flat=y scalebar=n minval= maxval= barreverse=n nreserve=8 bar= color= > plot.vpl			
需要一个 "unsigned char" 类型的输入 (sfbyte 的输出).			
string	**bar=**		file for scalebar data

			尺度数据文件
bool	**barreverse=n**	[y/n]	if y, go from small to large on the bar scale 如果是 y，在尺度条上由小到大
string	**color=**		调色板(默认为 i)
int	**dframe=1**		frame increment in a movie 动画画面增量
bool	**flat=y**	[y/n]	if n, display perspective view 如果为 n，显示俯视图
int	**frame1=0**		
int	**frame2=n2-1**		
int	**frame3=0**		frame numbers for cube faces 立体面数
float	**maxval=**		maximum value for scalebar (default is the data maximum) 尺度条的最大值（默认是数据的最大值）
float	**minval=**		minimum value for scalebar (default is the data minimum) 、 尺度条的最小值（默认是数据的最小值）
int	**movie=0**		0: 没有动画, 1:沿轴 1 的动画, 2: 沿轴 2 的动画, 3: 沿轴 3 的动画
int	**n1pix=n1/point1+n3/(1.-point1)**		number of vertical pixels 垂向像素个数
int	**n2pix=n2/point2+n3/(1.-point2)**		number of horizontal pixels 横向像素个数
int	**nreserve=8**		reserved colors 保留色
float	**point1=0.5**		fraction of the vertical axis for front face 显示前面时，所用垂向轴的比例。
float	**point2=0.5**		fraction of the horizontal axis for front face 显示前面时，所用横向轴的比例。
bool	**scalebar=n**	[y/n]	if y, draw scalebar

对于 sfgrey 和 sfgrey3 有不同的[配色方案]　具体的例子参考 [rsf/rsf/sfgrey].

sfgrey

Generate raster plot.

生成光栅图			
sfgrey < in.rsf > out.rsf bar=bar.rsf transp=y yreverse=y xreverse=n gpow= phalf= clip= pclip= gainstep=0.5+n1/256. allpos=n bias=0. polarity=n verb=n scalebar=n minval= maxval= barreverse=n wantframenum=(bool) (n3 > 1) nreserve=8 gainpanel= bar= color= > (plot.vpl | char.rsf)			
可以输入字符型值 若调用“byte”，输出为字符型值。 运行"sfdoc stdplot"，了解更多参数			
bool	**allpos=n**	[y/n]	if y, 取正数
string	**bar=**		file for scalebar data 尺度条数据文件
bool	**barreverse=n**	[y/n]	if y, 尺度条由小到大 go from small to large on the bar scale
float	**bias=0.**		subtract bias from data 从数据中提取偏离值
float	**clip=**		
string	**color=**		color scheme (default is i) 调色板（默认为 i）
string	**gainpanel=**		gain reference: 'a' for all, 'e' for each, or number 获得参考：‘a’代表全部，‘e’代表每个，或者数字
int	**gainstep=0.5+n1/256.**		**subsampling for gpow and clip estimation 为 gpow 和剪辑进行重采样**
float	**gpow=**		
float	**maxval=**		maximum value for scalebar (default is the data maximum) 尺度条的最大值（默认是数据的最大值）
float	**minval=**		minimum value for scalebar (default is the data minimum) 尺度条的最小值（默认是数据的最小值）
int	**nreserve=8**		reserved colors 保留色
float	**pclip=**		**data clip percentile (default is 99)** 数据剪辑百分比（默认值为 99）
float	**phalf=**		percentage for estimating gpow 估计 gpow 所用的百分比
bool	**polarity=n**	[y/n]	if y, reverse polarity (white is high by default) 若是 y，极性反转（默认白色是高值）
bool	**scalebar=n**	[y/n]	

bool	**transp=y**	[y/n]	if y, transpose the display axes 若是 y，反转所显示的轴
bool	**verb=n**	[y/n]	verbosity flag
bool	**wantframenum=(bool) (n3 > 1)**	[y/n]	if y, display third axis position in the corner 若是 y，在交点处显示第三个轴的位置
bool	**xreverse=n**	[y/n]	if y, reverse the horizontal axis 如果是 y，将水平轴反转
bool	**yreverse=y**	[y/n]	if y, reverse the vertical axis 将垂直轴反转

对于 sfgrey 和 sfgrey3 有不同的配色方案　具体的例子参考 rsf/rsf/sfgrey.

sfplas

Plot Assembler - convert ascii to vplot. **绘图编译器-将 ASCII 转换成 vplot**
sfplas

sfpldb

Plot Debugger - convert vplot to ascii **绘图调试器-将 vplot 转换为 ascii.**
sfpldb

sfplotrays

Plot rays. **绘制射线**		
sfplotrays frame=frame.rsf nt=n1*n2 jr=1 frame= < rays.rsf > plot.vpl		
运行"sfdoc stdplot"了解更多的参数		
string	**frame=**	
int	**jr=1**	跳过射线
int	**nt=n1*n2**	射线的最大长度

sfthplot

Hidden-line surface plot. **绘制含隐藏线的曲面**			
sfthplot < in.rsf uflag=y dflag=y alpha=45. titlsz=9 axissz=6 plotfat=0 titlefat=2 axisfat=2 plotcolup=VP_YELLOW plotcoldn=VP_RED axis=y axis1=y axis2=y axis3=y clip=0. pclip=100. gainstep=0.5+nx/256. bias=0. dclip=1. norm=y xc=1.5 zc=3 ratio=5. zmax= zmin= sz=6. label#= unit#= tpow=0 epow=0 gpow=1 title= > plot.vpl			
float	**alpha=45.**		alpha| < 89
bool	**axis=y**	[y/n]	
bool	**axis1=y**	[y/n]	
bool	**axis2=y**	[y/n]	
bool	**axis3=y**	[y/n]	plot axis 绘制轴
int	**axisfat=2**		axes fatness 轴的宽度
int	**axissz=6**		axes size 轴的尺寸
float	**bias=0.**		subtract bias from data 从数据中提取偏离值
float	**clip=0.**		data clip 数据修正
float	**dclip=1.**		change the clip: clip *= dclip
bool	**dflag=y**	[y/n]	if y, plot down side of the surface 若为 y，由面向下绘制
float	**epow=0**		exponential gain 指数增益
int	**gainstep=0.5+nx/256.**		subsampling for gpow and clip estimation
float	**gpow=1**		power gain 幂次方增益
string	**label#=**		对#轴加标签
bool	**norm=y**	[y/n]	用 clip 归一化
float	**pclip=100.**		data clip percentile
int	**plotcoldn=VP_RED**		color of the lower side 靠下部分的颜色
int	**plotcolup=VP_YELLOW**		color of the upper side 上办部分的颜色
int	**plotfat=0**		line fatness 线条宽度
float	**ratio=5.**		plot adjustment 绘图调节器
float	**sz=6.**		vertical scale
string	**title=**		
int	**titlefat=2**		title fatness 标题宽度
int	**titlsz=9**		title size 标题尺寸

string	**tpow=0**		time power gain 时间的幂次增益
bool	**uflag=y**	[y/n]	if y, plot upper side of the surface 若是 y，绘制表面的上半部分
string	**unit#=**		给#轴加单位
float	**xc=1.5**		
float	**zc=3**		lower left corner of the plot 图的左下角
float	**zmax=**		
float	**zmin=**		

sfwiggle

Plot data with wiggly traces. **用波动轨迹绘制数据**			
sfwiggle < in.rsf xpos=xpos.rsf xmax= xmin= poly=n fatp=1 xmask=1 ymask=1 pclip=98. zplot=0.75 clip=0. seemean=n verb=n transp=n yreverse=n xreverse=n xpos= > plot.vpl			
运行"sfdoc stdplot"了解更多信息			
float	**clip=0.**		data clip (estimated from pclip by default
int	**fatp=1**		
float	**pclip=98.**		clip percentile
bool	**poly=n**	[y/n]	
bool	**seemean=n**	[y/n]	if y, plot 为轨迹的条数
bool	**transp=n**	[y/n]	if y, 对轴反转
bool	**verb=n**	[y/n]	verbosity flag 冗余标志
int	**xmask=1**		
float	**xmax=**		maximum trace position (if using xpos) 轨迹的最大位置（使用了 xpos）
float	**xmin=**		minimum trace position (if using xpos) 轨迹的最小位置（使用了 xpos）
string	**xpos=**		optional header file with trace positions 选择头文件中是否含有道的位置
bool	**xreverse=n**	[y/n]	if y, 对水平轴反转
int	**ymask=1**		

bool	**yreverse=n**	[y/n]	if y, 对垂直轴反转
float	**zplot=0.75**		

4.1.5 绘图指令（开发部分）Plotting programs (development)

sfplsurf

sfplsurf 使用了 Plplot 的透视图功能。输出文件以 VPLOT 的形式被送到 stdout 因此用起来与使用 sfgrey 及其它绘图程序的方式相同,。如果输入文件中 n3>1,它还支持动画播放。下面可以看到一个使用 Sconstruct 的使用距离。 输出文件的动画 也可以找到。

```
from rsf.proj import *

# x & y dimensions
o1=-2
o2=-2
n1=41
n2=41
d1=0.1
d2=0.1
# z dimension
o3=-1
n3=21
d3=0.1

Flow('membrane',None,
     '''
     math o1=%g o2=%g n1=%d n2=%d d1=%g d2=%g
          o3=%g n3=%d d3=%g
          output="x3*cos(x1*x1+x2*x2)*exp(-0.1*(x1*x1+x2*x2))"
     ''' % (o1,o2,n1,n2,d1,d2,o3,n3,d3))

Result('membrane',
       '''
       plsurf title="Membrane" mesh=n color=j
              minval=%g maxval=%g
       ''' % (o3,o3 + d3*(n3-1)))

End()
```

4.1.6 系统/通用程序 system/generic programs

sfremap1

1-D ENO interpolation. 1 维 ENO 内插		
sfremap1 < in.rsf > out.rsf pattern=pattern.rsf n1=n1 d1=d1 o1=o1 order=3		
float	**d1=d1**	输出采样间隔
int	**n1=n1**	输出的采样点数
float	**o1=o1**	输出的起始点
int	**order=3**	内插顺序
string	**pattern=**	auxiliary input file name 辅助输入文件名

为了给出一个使用例子，首先为 sfremap1 创建一个输入文件:

```
sfmath n1=11 n2=11 d1=1 d2=1 o1=-5 o2=-5 output="x1*x1+x2*x2" > inp2remap1.rsf
```

我们先从两个维度同时对数据进行内插，然后进行显示:

```
< inp2remap1.rsf sfremap1 n1=1001 d1=0.01 | sftransp | \
sfremap1 n1=1001 d1=0.01 | sftransp | sfgrey allpos=y | sfpen
```

通过与没有差值的数据(< inp2remap1.rsf sfgrey allpos=y | sfpen)进行对比结果很有说服力。

4.1.7 系统/地震程序 system/seismic programs

sfstretch

Stretch of the time axis. 对时间轴拉伸			
sfstretch < in.rsf > out.rsf datum=dat.rsf inv=n dens=1 v0= half=y delay= tdelay= hdelay= nout=dens*n1 extend=4 mute=0 maxstr=0 rule=			
file	**datum=**		auxiliary input file name 辅助文件名
float	**delay=**		time delay for 对于 rule=lmo 的时间延迟
int	**dens=1**		axis stretching factor 轴的拉伸因子
int	**extend=4**		trace extension 道拉伸
bool	**half=y**	[y/n]	if y, the second axis is half-offset instead of full offset 若为 y，第二个轴是半炮检距而不是炮检距
float	**hdelay=**		offset delay for 相对于 rule=rad 的偏移距延迟

bool	**inv=n**	[y/n]	if y, do inverse stretching 若是 y，使用反拉伸
float	**maxstr=0**		maximum stretch 最大拉伸
int	**mute=0**		tapering size 逐渐减小的尺寸
int	**nout=dens*n1**		输出轴的长度（若 inv=n）
string	**rule=**		拉伸方式: n – 正常时差校正 (nmostretch), 默认 l – 线性时差校正(lmostretch) L – 对数拉伸(logstretch) 2 - t^2 拉伸 (t2stretch) c - t^2 chebyshev 拉伸(t2chebstretch) r – 径向时差(radstretch) d – datuming 数据 (datstretch)
float	**tdelay=**		time delay for 对于 rule=rad 的时间延迟
float	**v0=**		动校正时差

sfstretch rule=d (aka sfdatstretch) 可以用于静校正。下面是一个合成的例子，参考的 Alessandro Frigeri:

```
# generate a dataset with 'flat' signals
sfmath n1=200 n2=100 output="sin(0.5*x1)" type=float > scan.rsf

# generate a sinusoidal elevation correction
sfmath n1=100 output="3*sin(x1)" type=float > statics.rsf

# apply statics, producing a 'wavy' output.
sfstretch < scan.rsf > out.rsf datum=statics.rsf rule=d
```

4.1.8 user/fomels 指令

sfpick

Automatic picking from semblance-like panels. **从相似道中自动拾取**
sfpick < scn.rsf > pik.rsf vel0=o2 niter=100 an=1. gate=3 smooth=y rect#=(1,1,...) rect1=1 rect2=1 ...
rectN 定义了第 N 维平滑模板的尺寸。 理论在附录 B 中: S. Fomel, 2009,

Velocity analysis using AB semblance: Geophysical Prospecting, v. 57, 311-321. Reproducible 版本在 RSFSRC/book/jsg/avo			
float	**an=1.**		axes anisotropy 各向异性轴
int	**gate=3**		picking gate 选择的窗口
int	**niter=100**		number of iterations 迭代次数
int	**rect#=(1,1,...)**		smoothing radius on #-th axis 对第#个轴进行径向平滑
bool	**smooth=y**	[y/n]	if apply smoothing 如果运用了平滑
float	**vel0=o2**		surface velocity 地表速度

对算法的简单描述:

Start from the top (first time slice), pick an initial (source) point, evaluate all other points with the direct traveltime. 从顶部开始（第一个时间切片），选取一个起始（炮）点，按照时间顺序对其它点逐个分析。

At each grid point at the next level, find the traveltime to points at the previous level, add the traveltimes from the previous level, and select minimum. The search radius is limited by the aperture (gate= parameter in sfpick). 、 。。‘ 。’在下一个水平的每一个网格点上找出前一个水平的点所对应的走时，从前一水平加上走时并且选择最小值。选择的半径受到孔径的限制（gate=sfpick 中的参数）

Repeat step 2 until reaching the bottom. 重复步骤 2 知道到达底部。

Pick the minimum traveltime at the bottom and track the ray back to the source by following the traveltime gradient direction. 在底部选择最小的走时并沿射线按照走时的梯度追踪到震源。

Postprocessing (smooth= parameter in sfpick): smooth the picked ray path using shaping regularization. 后续处理（smooth=sfpick 中的参数）：用成型调整对选择的射线路径进行平滑处理。

很多人发现并发展了这个算法，最好的参考文献应该是 V. Meshbey, E. Ragoza, D. Kosloff, U. Egozi, and D. Wexler, 2002, Three-dimensional Travel-time Calculation Based on Fermat's Principle: Pure and Applied Geophysics, v. 159, 1563-1582.

4.1.9 user/ivlad 指令

sfprep4plot

Resamples a 2-D dataset to the desired picture resolution, with antialias **为了达到期望的分辨率并压制假频，对 2 维数据集重采样**
sfprep4plot inp= out= verb=n h=none w=none unit= ppi= prar=y
参数 h 和 w 只需确定一个。 若 prar=n, 由于轴上的 h/w 没有确定，因此不对这些轴进行操作。 若 prar=y 仅确定一个参数 (h 或 w),图片会沿两个轴进行尺度变换，直到达到特定的维数。

int	**h=none**		output height 输出的高度
string	**inp=**		input file 输入文件
string	**out=**		output file 输出文件
int	**ppi=**		输出分辨率 (px/in).当 unit!=px 时需要用到
bool	**prar=y**	[y/n]	若 y, PReserve Aspect Ratio of input 保留输入的纵横比
string	**unit=**		h 和 w 的单位. 可能是: px(默认), mm, cm, in
bool	**verb=n**	[y/n]	若 y, 打印系统命令，输出文件
int	**w=none**		输出宽度

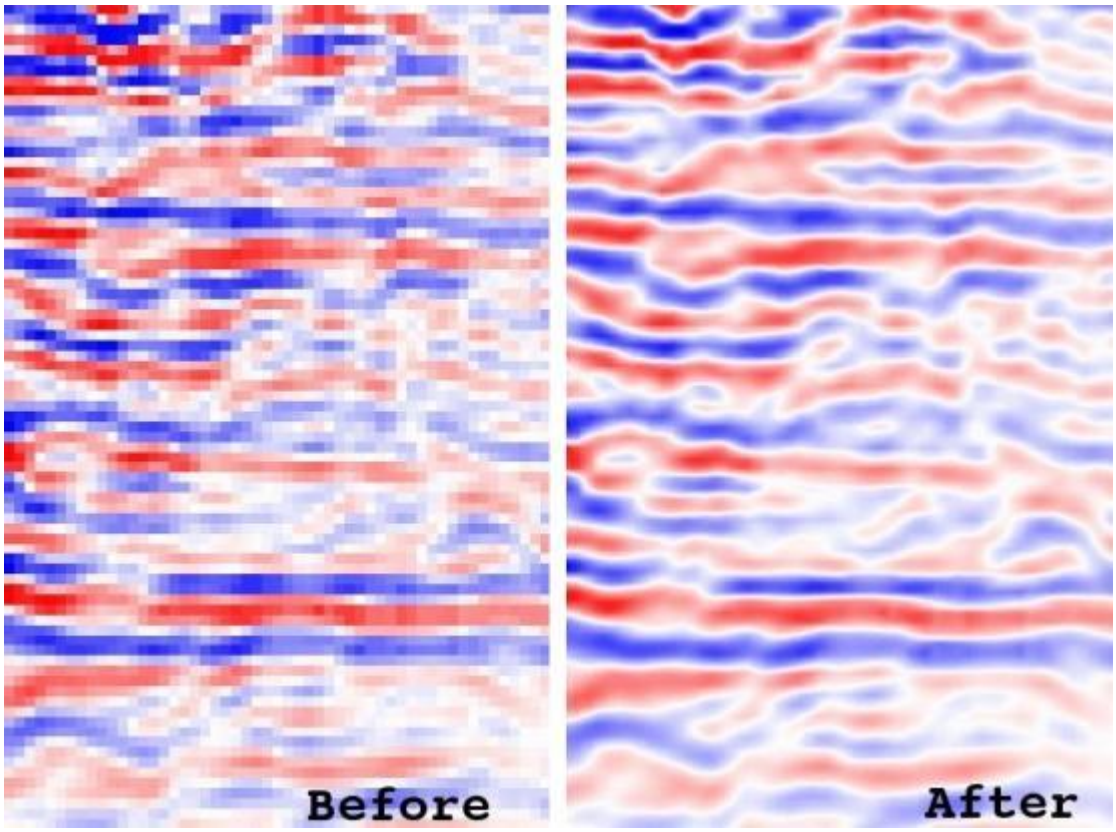
对于一个不需要保留纵横比并要送到 1280×1024 的投影仪展示的图像：

```
sfprep4plot inp=file1.rsf out=file2.rsf w=1280 h=1024 prar=n
```

对于 一个需要保持 250dpi 的分辨率被填入 6×8 英寸屏幕的图片，需要保留纵横比：

```
plot inp=file1.rsf out=file2.rsf w=6 h=8 unit=in ppi=250
```

下图是对一个图像运用 sfprep4plot 先后的对比，参考 Joachim Mispel：



sfcsv2rsf

Convert a delimited-text ASCII file to RSF binary floating point or int. 将一个 delimited-textASCII 文件转换成 RSF 二进制实型或整型文件			
sfcsv2rsf help=n delimiter=, dtype=float verb=n debug=n trunc=n o1=0. o2=0. d1=1. d2=1. unit1=unknown unit2=unknown label1=unknown label2=unknown			
如果每一行的元素个数不同，就要用 0 补齐。 n1 和 n2 是自动得到的。为了与 sfdisfil 和 sfmatmult 保持一致,输出文件是按照 C-文件的顺序（行在前），例如，输入文件中的行变成了输出文件中的列。输出文件按照相应的形式进行编码。			
float	**d1=1.**		
float	**d2=1.**		
bool	**debug=n**	[y/n]	Extra verbosity for debugging 用于调试的额外冗余
string	**delimiter=,**		输入文件中数值的分隔符
string	**dtype=float**		输入类型
bool	**help=n**	[y/n]	
string	**label1=unknown**		
string	**label2=unknown**		
float	**o1=0.**		
float	**o2=0.**		
bool	**trunc=n**	[y/n]	若行中的元素不同，需要截取或者补零
string	**unit1=unknown**		
string	**unit2=unknown**		
bool	**verb=n**	[y/n]	,是否显示 n1，n2，补充/截取

你会注意到每一行的数值的个数是不同的。

运行 sfcsv2rsf。注意不需要选项。默认的方式是通过补零来使行的长度相同：

```
$ <file.csv sfcsv2rsf > junk.rsf ; sfdisfil < junk.rsf
   0:          5          6          8        9.2          0
   5:         11        124          5          0          1
```

注意 sfdisfil 按照列的顺序显示（如，如果行数是正确的，矩阵会发生转秩）。实际上文件的维数在磁盘上发生了转秩。

$ sfin junk.rsf junk.rsf:

```
in="/data/path/junk.rsf@"
```

```
    esize=4 type=float form=native
    n1=5            d1=1            o1=0            unit1="unknown"
    n2=2            d2=1            o2=0            unit2="unknown"
10 elements 40 bytes
```

你可以根据自己需要，用 sftransp 来运行输出。然而，为 sfmatmult 创建输入文件并不必要，因为 sfmatmult 用来对以 sfdisfil 显示的矩阵进行操作，并用转秩作为输入。

当然，可以使用管道来跳过创建中间文件的步骤：

```
$ <file.csv sfcsv2rsf | sfdisfil
    0:          5          6          8        9.2          0
    5:         11        124          5          0          1
```

注意这个程序不需要任何参数（只用 stdin 和 stdout），当不使用参数时会显示帮助文件。为了使用自动生成的文档，你需要跳过选项 help=y。

4.1.10 user/jennings 指令

sfsizes

Display the size of RSF files. **显示 RSF 文件的大小**			
sfsizes files=y human=n file1.rsf file2.rsf ...			
Prints the element size, number of elements, and number of bytes for a list of RSF files. Non-RSF files are ignored. 打印元素的大小、个数以及一系列 RSF 文件的字节数。忽略非 RSF 文件			
bool	**files=y**	[y/n]	If y, print size of each file. If n, print only total. 若 y，输出每个文件的大小。若 n，仅输出总的大小
bool	**human=n**	[y/n]	If y, print human-readable file size. If n, print byte count. 若 y，输出刻度文件大小。若 n，输出二进制数值。

这个程序计算了 RSF 文件的理论字节数。由于磁盘数据块因素的影响，实际占的空间可能会有些不同。理论的数组大小可以修改。这种计算文件大小的方式运行很快，因为它值读了 RSF 文件的道头而不读取实际数据。

例如，不用列举每个文件，就可以以人们可读的格式计算出所有 RSF 文件的大小。

```
sfsizes files=n human=y *.rsf
```

下面这种方式同样适用，因为 sfsizes 会跳过非 RSF 文件：

```
sfsizes files=n human=y *
```

sffiglist

Compare Vplot files in Fig and Lock directories **比较 Fig 和 Lock 路径下的 Vplot 文件**		
sffiglist figdir= lockdir= list= show=		
Parameter **figdir** is path to Fig directory, default is ./Fig. 参数 figdir 是 Fig 的路径，默认值为./Fig。 Parameter **lockdir** is path to Lock directory: 参数 lockdir 是 Lock 的路径 若 **figdir** 在路径 $RSFSRC/book/[book]/[chapter]/[section]中, 则 **lockdir** 默认的路径是$RSFFIGS/[book]/[chapter]/[section]. 若 **figdir** 不在路径 $RSFSRC/book/[book]/[chapter]/[section]中, 则 **lockdir** 默认的路径是 $RSFALTFIGS/[book]/[chapter]/[section]. 参数 **list** 控制了要显示的文件,默认为显示全部。 参数 **show** 控制了要用 sfpen 处理的文件 ，默认是没有要处理的文件。 **list|show** = none (没有文件，只输出总结) **list|show** = diff (不相同的文件，由 sfvplotdiff 确定.) **list|show** = miss (figdir 或 lockdir 中缺失的不同的文件.) **list|show** = all (所有文件.) File list codes 文件列表所用参数: space indicates files that are the same 空格意味着文件相同. - 意味着 lockdir 中的文件在 figdir 中不存在. + 意味着 figdir 中的外部文件在 lockdir 中不存在. number 是表示不同文件的 sfvplotdiff 的返回码.		
string	**figdir=**	fig 路径, 默认 = ./Fig
string	**list=**	all 所有的
string	**lockdir=**	lock 路径, 默认的是 = lock 中与 figdir 对应
string	**show=**	显示几个 [none 没有,diff 不同的,miss 缺失,all 所有的], 默认为 = 没有 none

这个工具列举了“Fig”和“Lock”路径中的 Vplot 文件，并用 sfvplotdiff 对它们进行比较。

Fig 的默认路径是./Fig，Lock 的默认路径与 “scons lock” 存放文件的路径有关，但是两个默认值之一能用用户参数 figdir 和 lockdir 来覆盖，例如，可以对在量个不同的 Fig 路径下的文件进行比较。

Lock 路径的默认值有一些存在$RSFFIGS 中的逻辑关系。当 Fig 在路径$RSFSRC/book 中

或者当 Fig 不存在于$RSFSRC/book 中而在 $RSFALTFIGS 中寻找时会使用这些逻辑关系，因为我喜欢使用两个不同的 Lock 路径:一个存放书中的资料，另一个存放数种没有的我自己的资料。但是当这些环境变量没有定义时，我尽量让代码的默认值变得有意义。

这个工具可以给出相同文件、不同文件、Fig 中存在而 Lock 中不存在的文件以及 Lock 中存在而 Fig 中不存在的文件的统计结果。

参数 list（默认=all）和 show（默认=none）决定了列出或用 sfpen 处理哪些文件。列出的文件说明了哪些文件相同、不同以及那些文件在 Fig 或 Lock 中缺失。

例如，列出 Fig 和 Lock 中所有的 Vplot 文件：

```
sffiglist list=all
```

列出所有的 Vplot 文件并显示不相同的文件：

```
sffiglist list=all show=diff
```

4.1.11user/psava 指令

sfawefd

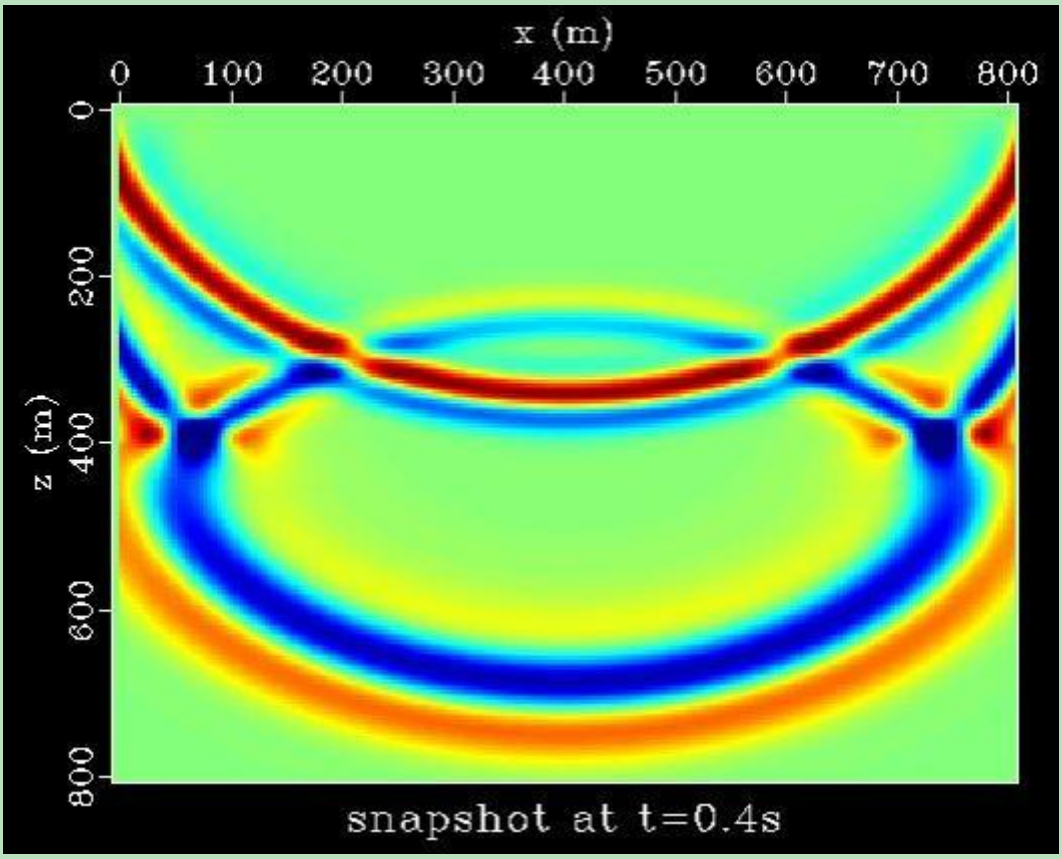
acoustic time-domain FD modeling **声波时域有限差分模拟**			
sfawefd < Fwav.rsf vel=Fvel.rsf sou=Fsou.rsf rec=Frec.rsf wfl=Fwfl.rsf > Fdat.rsf den=Fden.rsf ompchunk=1 ompnth=0 verb=n snap=n free=n expl=n jdata=1 jsnap=nt nq1=sf_n(a1) nq2=sf_n(a2) oq1=sf_o(a1) oq2=sf_o(a2)			
file	**den=**		auxiliary input file name 辅助输入文件名
bool	**expl=n**	[y/n]	"exploding reflector" 爆炸反射面
bool	**free=n**	[y/n]	free surface flag 自由界面标志
int	**jdata=1**		
int	**jsnap=nt**		save wavefield every *jsnap* time steps 按时间步长为*jsnap*保存波场
int	**nq1=sf_n(a1)**		
int	**nq2=sf_n(a2)**		
int	**ompchunk=1**		OpenMP 数据块大小
int	**ompnth=0**		OpenMP 可用线程
float	**oq1=sf_o(a1)**		
float	**oq2=sf_o(a2)**		
file	**rec=**		auxiliary input file name 辅助输入文件名

bool	**snap=n**	[y/n]	wavefield snapshots flag 波场快照标志
file	**sou=**		auxiliary input file name 辅助输入文件名
file	**vel=**		auxiliary input file name 辅助输入文件名
bool	**verb=n**	[y/n]	verbosity flag 冗余标志
file	**wfl=**		auxiliary output file name 辅助输出文件名

下面展示一个 nx=nz=200, dx=dz=4m (size: 800x800m)的模型。模型中有两层：第一层在(z,x)上有 100x200 个样点，速度为 1500m/s;第二层的维数与第一层相同，速度为 3000m/s。所有网格的速度都为 1。震源和检波器放在同一位置 x=400 and z=100。整个模型孔径的全波长每 10 个时间步长记录一次。

```
# Velocity model:
sfspike >Fvel.rsf mag=1500,3000 nsp=2 k1=1,101 l1=100,200 d1=4 d2=4 \
label1=z label2=x n1=200 n2=200 o1=2 o2=2 unit1=m unit2=m

# Density model:
sfspike >Fden.rsf mag=1 nsp=1 k1=1 l1=200 d1=4 d2=4 label1=z \
label2=x n1=200 n2=200 o1=2 o2=2 unit1=m unit2=m

# Source position (x,z):
sfspike n1=2 nsp=2 k1=1,2 mag=400,100 o1=0 o2=0 >Fsou.rsf

# Receiver position (x,z):
sfspike n1=2 nsp=2 k1=1,2 mag=400,100 o1=0 o2=0 >Frec.rsf

# Source wavelet:
sfspike nsp=1 n1=2000 d1=0.0005 k1=200 | sfricker1 frequency=20 |\
sftransp >Fwav.rsf

# Creating data at specified receiver + saving full wavefield every 10th step:
sfawefd2d <Fwav.rsf vel=Fvel.rsf sou=Fsou.rsf rec=Frec.rsf wfl=Fwfl.rsf \
den=Fden.rsf >Fdat.rsf verb=y free=y expl=y snap=y dabc=y jdata=1 jsnap=10
echo 'label1=z unit1=m label2=x unit2=m' >> Fwfl.rsf

# View the wavefield movie:
< Fwfl.rsf sfgrey gainpanel=a pclip=99 color=j scalebar=y | sfpen
```

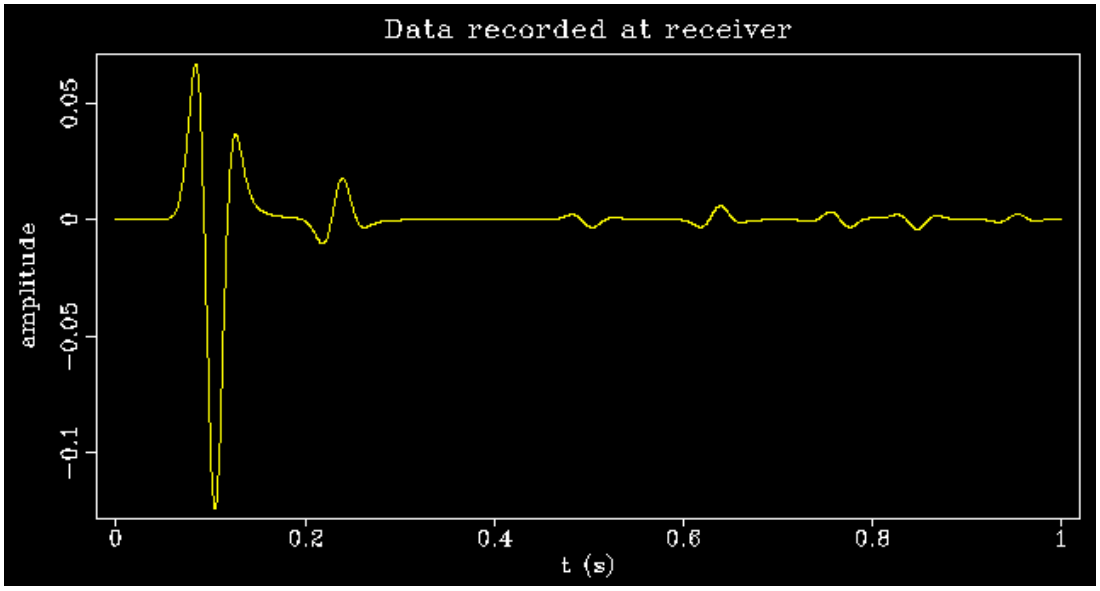
```
# View a wavefield snapshot:
< Fwfl.rsf sfwindow f3=80 n3=1 |\
sfgrey pclip=99 color=j title='snapshot at t=0.4s' |\
sfpen

# View the data recorded at receiver:
<Fdat.rsf sfwindow |\
sfgraph title='Data recorded at receiver' unit2='' label2=amplitude |\
sfpen
```



Sfawefd 波场快照

http://reproducibility.org/wiki/File:Sfawefd_dat.png

需要注意的是：步长太大会导致数值计算的不稳定。



sfsrmig3

3-D S/R migration with extended SSF **使用扩展的 SSF 进行 3 维 S/R 偏移**			
sfsrmig3 slo=Fs_s.rsf sls=Fs_r.rsf < Fw_s.rsf rwf=Fw_r.rsf > Fi.rsf cig=Fc.rsf ompchunk=1 ompnth=0 verb=y eps=0.01 twoway=n nrmax=1 dtmax=0.004 pmx=0 pmy=0 tmx=0 tmy=0 vpvs=1. hsym=n nht=1 oht=0 dht=0.1 nht=1 oht=0 dht=0.1 hsym=n nhh=1 ohh=0 dhh=0.1 nha=180 oha=0 dha=2.0 nhb=180 ohb=0 dhb=2.0 itype=			
file	**cig=**		auxiliary output file name 辅助输出文件名
float	**dha=2.0**		
float	**dhb=2.0**		
float	**dhh=0.1**		
float	**dht=0.1**		
float	**dtmax=0.004**		max time error 最大时间误差

float	**eps=0.01**		stability parameter 稳定性参数
bool	**hsym=n**	[y/n]	
string	**itype=**		imaging condition type 成像条件类型 o = zero lag 零延迟 (默认) e = extended 扩展 x = space-lags h = space-lags magnitude t = time-lag
int	**nha=180**		
int	**nhb=180**		
int	**nhh=1**		
int	**nht=1**		
int	**nrmax=1**		max number of refs refs 的最大数量
float	**oha=0**		
float	**ohb=0**		
float	**ohh=0**		
float	**oht=0**		
int	**ompchunk=1**		OpenMP 数据块大小
int	**ompnth=0**		OpenMP available threads OpenMP 可用线程
int	**pmx=0**		padding on x 对 x 轴填补
int	**pmy=0**		padding on y 对 y 轴填补
file	**rwf=**		auxiliary input file name 辅助输入文件名
file	**slo=**		auxiliary input file name 辅助输入文件名
string	**sls=**		auxiliary input file name 辅助输入文件名
int	**tmx=0**		taper on x 在 x 方向上减弱
int	**tmy=0**		taper on y 在 y 方向上减弱
bool	**twoway=n**	[y/n]	two-way traveltime 双程旅行时
bool	**verb=y**	[y/n]	verbosity flag 冗余标志
float	**vpvs=1.**		Vp/Vs ratio Vp/Vs 比

这个程序实现了 3 维和 2 维炮记录偏移。它是用具有多个参考**速度的扩展的分步傅立叶（SSF）**外推算子实现的。它将炮波场（stdin）、检波点波场（rwf=）及慢度模型（slo=）作为输入。输出是共成像点道集（cig=）的图片（stdout）和数据体。一个重要的参数是

nrmax，即参考速度的个数，它的默认值是 1，但是合理的值应该在 5 左右。很有必要确定非零的 **taper** 值（tmx 和 3 维的还包括 tmy）。参数 pmx 和 pmy 在程序中被分为两部分，例如，x 轴上的数据有 501 个点的长度，令 pmx=11 来得到 512 个值来使用 FFT（快速傅立叶变换）。

当提供横波慢度模型（sls=）时，程序也可以用来对转换波数据进行偏移。

vpvs 参数仅在 itype=h。在目的不明确时最好不要指定 vpvs 的值。

Usage example

下面程序，对 RSFSRC/book/data/sigsbee/ptest 进行了稍微的改动，展示了怎样将 Sigsbee 2A 数据和速度用于偏移。

将 SEG-Y 格式的输入数据（炮记录）转换成 RSF 文件：

```
sfsegyread tape=sigsbee2a_nfs.segy tfile=tdata.rsf hfile=/dev/null bfile=/dev/null > ddata.rsf
```

将道头转换成实型（满足 sfheadermath 的需要）：

```
< tdata.rsf sfdd type=float > trchdr.rsf
```

炮点位置：

```
< trchdr.rsf sfheadermath output="fldr + 10925/150" | sfwindow squeeze=y > tsi.rsf
```

从道头文件中提取炮检距位置，忽略长度为 1 的轴和尺度，为抽道集创建一个头文件（满足 sfintbin 的需要）：

```
< trchdr.rsf sfheadermath output="offset" |\
sfwindow squeeze=y |\
sfmath output="input/75" |\
sfcat axis=2 space=n tsi.rsf |\
sftransp |\
sfdd type=int > tos.rsf
```

抽道集和切除：

```
< ddata.rsf sfintbin head=tos.rsf xkey=0 ykey=1 |\
sfput label1=Time unit1=s d2=0.075 o2=0.0 label2=hx d3=0.150 o3=10.925 label3=sx |\
sfmutter half=false t0=1.0 v0=6.0 |\
sfput d2=0.02286 o2=0 unit2=km d3=0.04572 o3=3.32994 unit3=km > shots.rsf
```

数据体的维数是用如下方式创建的：

```
$ sfin rfft.rsf trail=n
rfft.rsf:
    in="/var/tmp/rfft.rsf@"
    esize=8 type=complex form=native
    n1=200          d1=0.25         o1=1            label1="Frequency" unit1="Hz"
    n2=348          d2=0.02286      o2=0            label2="hx" unit2="km"
    n3=1            d3=1            o3=0            label3="hy" unit3="km"
    n4=20           d4=0.9144       o4=3.78714      label4="sx" unit4="km"
        1392000 elements 11136000 bytes
```

创建一个震源子波（与数据的频带相同）并对其作傅立叶变换：

```
sfspike k1=1 n1=1500 d1=0.008 |\
sfbandpass flo=15 fhi=25 |\
sffft1 |\
sfwindow n1=200 min1=1 j1=3 |\
sfput label1=freq > sfft.rsf
```

在下面得到了频域的子波：

```
$ sfin sfft.rsf
sfft.rsf:
    in="/var/tmp/sfft.rsf@"
    esize=8 type=complex form=native
    n1=200          d1=0.25         o1=1            label1="freq" unit1="Hz"
        200 elements 1600 bytes
```

根据子波和数据的频率切片，用 srsyn 创建同步的炮点和接收点波场。接收点和炮点的频率切片基本上都在正确的位置并且向上直到下面定义的 x 轴各点的数值都为零：

```
< rfft.rsf sfsrsyn nx=1067 dx=0.02286 ox=3.05562 wav=sfft.rsf swf=swav.rsf > rwav.rsf
```

下面为炮点和检波点的偏移准备了频率切片，只是在两个数据集中，轴 1（频率）需要变为轴 3：

```
< swav.rsf sftransp plane=12 | sftransp plane=23 > stra.rsf
```

```
< rwav.rsf sftransp plane=12 | sftransp plane=23 > rtra.rsf
```

下面创建了地表检波点波场作为偏移的输入。轴 4 是炮号。轴 4 的值是任意的，因

为每一包都用零补齐了，这样就可以保证覆盖到整个速度模型。对于每一炮，向下延拓的孔径应该与工区一样大。

```
sfin trail=n rtra.rsf
rtra.rsf:
    in="/var/tmp/rtra.rsf@"
    esize=8 type=complex form=native
    n1=1067          d1=0.02286      o1=3.05562      label1="x" unit1="km"
    n2=1             d2=1            o2=0            label2="y" unit2="km"
    n3=200           d3=0.25         o3=1            label3="w" unit3="Hz"
    n4=20            d4=1            o4=0            label4="e" unit4="km"
        4268000 elements 34144000 bytes
```

将速度模型由 SEG-Y 形式转换为 RSF 形式，十进制，将英尺转换为千米，转秩，转换为慢度，增加一个轴：

```
sfsegyread tape=sigsbee2a_migvel.sgy tfile=/dev/null hfile=/dev/null bfile=/dev/null |\
sfput o1=0 d1=0.00762 label1=z unit1=km o2=3.05562 d2=0.01143 label2=x unit2=km |\
sfwindow j1=4 j2=2 |\
sfscale rscale=0.0003048 |\
sftransp |\
sfmath output="1/input" |\
sfspray axis=2 n=1 d=1 o=0 |\
sfput label2=y > slow.rsf
```

下面为偏移创建了一个 x 轴与波场文件的 x 轴相同的慢度文件：

```
$ sfin slow.rsf
slow.rsf:
    in="/var/tmp/slow.rsf@"
    esize=4 type=float form=native
    n1=1067          d1=0.02286      o1=3.05562      label1="x" unit1="km"
    n2=1             d2=1            o2=0            label2="y" unit2="km"
    n3=301           d3=0.03048      o3=0            label3="z" unit3="km"
        321167 elements 1284668 bytes
```

最后，使用偏移命令（对应于 4 核处理器，因此为 ompnth 的值）。我们选择不对任何成像道集进行计算（itype=0），但是由于程序的构建原因，我们还需要另外设置 cig

标签或者是一个名为 tag 的 RSF 文件，不会产生 rsf 扩展文件：

```
< stra.rsf sfsrmig3 nrmax=20 dtmax=5e-05 eps=0.01 verb=y ompnth=4 \
tmx=16 rwf=rtra.rsf slo=slow.rsf itype=o cig=/dev/null > img.rsf
```

对 20 炮数据进行偏移在 4 核的计算机上需要用近 3 个小时（1 炮=9 分钟）。如果频率切片不 3 个**抽取 1 个**，深度轴不 4 个抽取 1 个，就需要用 12 倍于此的时间。结果的图片 y 轴的长度为 1：

```
$ sfin img.rsf trail=n
img.rsf:
    in="/var/tmp/img.rsf@"
    esize=4 type=float form=native
    n1=1067          d1=0.02286      o1=3.05562      label1="x" unit1="km"
    n2=1             d2=1            o2=0            label2="y" unit2="km"
    n3=301           d3=0.03048      o3=0            label3="z" unit3="km"
        321167 elements 1284668 bytes
```

为了正确查看图片，我们应该改忽略长度为 1 的轴，然后将 x 轴和 z 轴反转到它们的正确位置上：

```
<img.rsf sfwindow squeeze=y | sftransp | sfgrey > img.vpl
```

References

参考文献

↥ Claerbout, J., 1998, Multidimensional recursive filters via a helix: Geophysics, 63, 1532--1541.

↥ Claerbout, J., 1998, Multidimensional recursive filters via a helix: Geophysics, 63, 1532--1541.

↥ Barry, K.M., Cavers, D.A., and Kneale, C.W. 1975. Recommended standards for digital tape formats. *Geophysics*, 40, no. 02, 344–352.

↥ Norris, M.W., Faichney, A.K., *Eds*. 2001. SEG Y rev1 Data Exchange format. Society of Exploration Geophysicists, Tulsa, OK, 45 pp.

4.2 变换

4.2.1Fourier 变换

4. 2. 2Radon 变换

第 5 章 Madagascar 编程参考

这份文件是提供给 Madagascar 开发者社团的一个很好的参考，写它主要的目的是帮助大家进行 Madagascar 编程。它也让我们在个人需求之外学到更多关于 Madagascar 中描述的库文件的内容。这份材料特别列出并描述了面向 C 的基于 Madagascar 的 RSF API（应用编程接口）函数。

第 1 章是简介，为了了解 Madagascar 和它的发展历史，我们总结了一些关键信息。在第 2 章中，有一个示例程序，附带了每一行代码的注释。示例程序是一个有限差分正演代码。

RSF 函数库将从第 3 章开始讲述，它也是对 Madagascar 中使用的数据格式的描述。有些数据格式是在 Madagascar 定义的，还有一些产生于标准 C 头文件库。

接下来的几章将用于特定任务的不同函数进行分组。第 4 章列出了具有数据输入准备工作功能的.c 文件，例如 sf_alloc，用来分配需要的空间。

第 5 章是关于.rsf 文件的操控，比如文件输入/输出，从文件中抽取或插入参数。

第 6 章是关于错误操控函数。它列出了打印需要的错误信息的函数。

第 7 章列出了线性运算算子。在本章开始时有一个详细的介绍。

第 8 章是关于数据分析函数，比如 kiss_fftr.c，是针对实数？？时间域信号 Fourier 反变换和正变换。

第 9 章列出了滤波和褶积函数。

第 10 章 1 列出了求解？？。有的函数用来求解类似第一阶 ODE，它使用 Runge-Kutta 求解？？。本章中有一些函数可以对实数和复数数据的共轭梯度方法进行迭代。还有一些功能包括求根和三对角矩阵解算器。

第 11 章是针对插值函数。它包括 1D,2D,3D 插值函数。还有对于 B-Spline 的插值功能，对 B-spline 系数，ENO 和 ENO power-p 插值的计算。

第 12 章是平滑和边缘检测函数。包括对于 1D 和 2D 的三角平滑？？。

第 13 章列出了射线追踪？？函数。

第 14 章是一些通用的工具，如估算数学表达式和产生随机数功能。

第 15 章主要是几何地震学。有一个功能是用来定义坐标系和点，这可以用在观测系统的定义中。还有 axa.c 文件，用来定义一个函数，来创建并对坐标轴进行操作。

第 16 章是针对杂项函数。

第 17 章列出的功能，是系统具体例子 system.c 文件的定义函数，它用来在终端中运行一条来程序中给定的命令。

5.1 简要说明

Madagascar 在 C 中，并有选择的在 C++和 Fortran90 等编程语言中提供了编程接口。在 Madagascar 软件包安装过程中自动安装了 C 语言应用编程接口（API）。其他接口需要

独立安装。如果需要安装，可以在安装指南中查询到相关信息。

使用 Madagascar 提供的编程语言 API 的程序，可以让用户对 RSF 文件进行操作，并可以使用那些已定义好的 RSF 库函数。这些库文件在源代码目录下的 api 和 build/api 子目录中。

您可以在 Madagascar 网页上查询 Madagascar 编程接口指南。

5.2 有限差分正演案例

本章中将详细介绍时间域有限差分正演程序，这样可以帮助阐述如何使用 RSF 库。

5.2.1 介绍

本节将介绍用 RSF 库编写的时域有限差分正演。程序是用 C、C++和 Fortran90 接口来展示的。声波方程

$$\Delta U - \frac{1}{\upsilon^2}\frac{\partial^2 U}{\partial t^2} = f(t)$$

可以写成

$$\left|\Delta U - f(t)\right|\upsilon^2 = \frac{\partial^2 U}{\partial t^2}$$

其中，Δ 是 Laplace 算符，$f(t)$ 是震源子波，υ 是速度，U 是标量波场。离散的时间步长计算涉及到下列计算：

$$U_{i+1} = \left|\Delta U - f(t)\right|\upsilon^2\Delta t^2 + 2U_i - U_{i-1}$$

5.2.2 C 程序

```
1   /* time-domain acoustic FD modeling */
2   #include <rsf.h>
3
4   int main(int argc, char* argv[])
5   {
6       /* Laplacian coefficients */
7       float c0=-30./12.,c1=+16./12.,c2=- 1./12.;
8
9       bool verb; /* verbose flag */
10      sf_file Fw=NULL,Fv=NULL,Fr=NULL,Fo=NULL; /* I/O files */
11          sf_axis at,az,ax; /* cube axes */
12      int it,iz,ix; /* index variables */
```

```
13      int nt,nz,nx;
14      float dt,dz,dx,idx,idz,dt2;
15
16      float *ww,**vv,**rr; /* I/O arrays*/
17      float **um,**uo,**up,**ud;/* tmp arrays */
18
19      sf_init(argc,argv);
20      if(! sf_getbool("verb",&verb)) verb=0;
21
22      /* setup I/O files */
23      Fw = sf_input ("in" );
24      Fo = sf_output("out");
25      Fv = sf_input ("vel");
26      Fr = sf_input ("ref");
27
28      /* Read/Write axes */
29      at = sf_iaxa(Fw,1); nt = sf_n(at); dt = sf_d(at);
30      az = sf_iaxa(Fv,1); nz = sf_n(az); dz = sf_d(az);
31      ax = sf_iaxa(Fv,2); nx = sf_n(ax); dx = sf_d(ax);
32
33      sf_oaxa(Fo,az,1);
34      sf_oaxa(Fo,ax,2);
35      sf_oaxa(Fo,at,3);
36
37      dt2 = dt*dt;
38      idz = 1/(dz*dz);
39      idx = 1/(dx*dx);
40
41      /* read wavelet, velocity & reflectivity */
42      ww = sf_floatalloc(nt); sf_floatread(ww ,nt ,Fw);
43      vv = sf_floatalloc2(nz,nx); sf_floatread(vv[0],nz*nx,Fv);
44      rr = sf_floatalloc2(nz,nx); sf_floatread(rr[0],nz*nx,Fr);
45
46      /* allocate temporary arrays */
47      um = sf_floatalloc2(nz,nx);
48      uo = sf_floatalloc2(nz,nx);
```

```
49      up = sf_floatalloc2(nz,nx);
50      ud = sf_floatalloc2(nz,nx);
51
52      for (iz=0; iz<nz; iz++) {
53          for (ix=0; ix<nx; ix++) {
54          um[ix][iz]=0;
55              uo[ix][iz]=0;
56              up[ix][iz]=0;
57              ud[ix][iz]=0;
58          }
59      }
60
61      /* MAIN LOOP */
62      if(verb) fprintf(stderr,"\n");
63      for (it=0; it<nt; it++) {
64      if(verb) fprintf(stderr,"\b\b\b\b\b%d",it);
65
66          /* 4th order laplacian */
67          for (iz=2; iz<nz-2; iz++) {
68              for (ix=2; ix<nx-2; ix++) {
69                  ud[ix][iz] =
70              c0* uo[ix ][iz ] * (idx+idz) +
71              c1*(uo[ix-1][iz ] + uo[ix+1][iz ])*idx +
72              c2*(uo[ix-2][iz ] + uo[ix+2][iz ])*idx +
73              c1*(uo[ix ][iz-1] + uo[ix ][iz+1])*idz +
74              c2*(uo[ix ][iz-2] + uo[ix ][iz+2])*idz;
75              }
76          }
77
78          /* inject wavelet */
79          for (iz=0; iz<nz; iz++) {
80              for (ix=0; ix<nx; ix++) {
81                  ud[ix][iz] -= ww[it] * rr[ix][iz];
82              }
83          }
84
```

```
85          /* scale by velocity */
86          for (iz=0; iz<nz; iz++) {
87               for (ix=0; ix<nx; ix++) {
88                    ud[ix][iz] *= vv[ix][iz]*vv[ix][iz];
89               }
90          }
91
92          /* time step */
93          for (iz=0; iz<nz; iz++) {
94               for (ix=0; ix<nx; ix++) {
95                    up[ix][iz] = 2*uo[ix][iz]
96                    - um[ix][iz]
97                    + ud[ix][iz] * dt2;
98
99                    um[ix][iz] = uo[ix][iz];
100                   uo[ix][iz] = up[ix][iz];
101              }
102         }
103
104         /* write wavefield to output */
105         sf_floatwrite(uo[0],nz*nx,Fo);
106         }
107         if(verb) fprintf(stderr,"\n");
108         sf_close()
109         exit(0);
110         }
```

5.2.3 代码释义

```
9       bool verb; /* verbose flag */
10      sf_file Fw=NULL,Fv=NULL,Fr=NULL,Fo=NULL; /* I/O files */
11           sf_axis at,az,ax; /* cube axes */
12      int it,iz,ix; /* index variables */
13      int nt,nz,nx;
14      float dt,dz,dx,idx,idz,dt2;
```

第 12-14 行定义了整型和浮点型变量，用在主循环中的控制变量（it，iz，ix），维度

（nt，nz，nx）、采样率（dt，dz，dx）、采样率的平方和平方反比（dt2，idz，idx）。

16-17:

```
16      float *ww,**vv,**rr; /* I/O arrays*/
17      float **um,**uo,**up,**ud;/* tmp arrays */
```

第 19 行对符号列表进行了初始化，用于存储该命令行的参数。

第 20 行用来测试命令行参数中定义的标识符。如果命令行中的标识符被设定成 n，变量 verb（布尔型）就被设定为 0.只有当在命令行中将标识符设置为 y 时，才允许打印出详细的输出。

22-26:

```
22      /* setup I/O files */
23      Fw = sf_input ("in" );
24      Fo = sf_output("out");
25      Fv = sf_input ("vel");
26      Fr = sf_input ("ref");
```

这几行中，我们使用 RSF API 的 sf_input（见 P80）和 sf_output（见 P80）函数。这两个函数取一个字符串作为参数并返回一个 sf_file 类型的变量，至于这一变量的的定义我们已经在之前的程序给出。

28-32:

```
28      /* Read/Write axes */
29      at = sf_iaxa(Fw,1); nt = sf_n(at); dt = sf_d(at);
30      az = sf_iaxa(Fv,1); nz = sf_n(az); dz = sf_d(az);
31      ax = sf_iaxa(Fv,2); nx = sf_n(ax); dx = sf_d(ax);
```

这里，我们用 RSF API 的 sf_iaxa（P348）sf_iaxa 输入坐标轴（at，az，ax）。sf_iaxa 接受了一个 sf_file 类型和一个整型的变量。sf_iaxa 中的第一个参数是输入文件，第二个是我们想输入的轴。在第二列中，我们用 RSF API 的 sf_n 来（P350）获取各坐标轴的长度。

在第三列中，我们用 RSF API 的 sf_d（P351）来获取各坐标轴的采样间隔。

33-35:

```
33      sf_oaxa(Fo,az,1);
34      sf_oaxa(Fo,ax,2);
35      sf_oaxa(Fo,at,3);
```

这里，我们用 RSF API 的 sf_oaxa（P349）输出坐标轴。sf_oaxa 接受了 sf_file（RSF API），sf_axis（RSF API）类型的变量和一个整型变量。第一个参数是输出文件，第二个参数是我们要输出的坐标轴的名称，第三个是输出文件（n1 是最快的轴？？）中坐标轴的数量。

41-44:

```
41      /* read wavelet, velocity & reflectivity */
```

```
42      ww = sf_floatalloc(nt); sf_floatread(ww ,nt ,Fw);
43      vv = sf_floatalloc2(nz,nx); sf_floatread(vv[0],nz*nx,Fv);
44      rr = sf_floatalloc2(nz,nx); sf_floatread(rr[0],nz*nx,Fr);
```

在第一列中，我们为输入的子波、速度场、反射系数分配了所需的内存空间。这一操作是用 RSF API 的 sf_floatalloc（P36）和 sf_floatalloc2（P42）函数来完成的。sf_floatalloc 取整数作为参数并以此来计算分配适当大小的内存块。sf_floatalloc2 和 sf_floatalloc 基本一样，除了前者分配的是一个二维序列的空间，在这种情况下内存块的大小是参数中给定的两个整数的乘机（例如例子中的 nz*nx）。

RSF API 的 sf_floatread（P109）用来从文件中读取数据放在分配的内存块（序列）中。sf_floatread 取数组、整数和文件作为参数并将用文件数据填充的序列作为返回值。

46-50:

```
46      /* allocate temporary arrays */
47      um = sf_floatalloc2(nz,nx);
48      uo = sf_floatalloc2(nz,nx);
49      up = sf_floatalloc2(nz,nx);
50      ud = sf_floatalloc2(nz,nx);
```

就像分配内存块来将输入文件读入一样，现在我们用 sf_floatalloc2（P42）为临时数组分配内存，我们就用这些数组进行下面的计算。

52-59:

```
52      for (iz=0; iz<nz; iz++) {
53          for (ix=0; ix<nx; ix++) {
54              um[ix][iz]=0;
55              uo[ix][iz]=0;
56              up[ix][iz]=0;
57              ud[ix][iz]=0;
58          }
59      }
```

第 52-59 行对临时数组进行了初始化，手法是对数组每一个元素赋 0 值。

61-64:

```
61      /* MAIN LOOP */
62      if(verb) fprintf(stderr,"\n");
63      for (it=0; it<nt; it++) {
64      if(verb) fprintf(stderr,"\b\b\b\b\b%d",it);
```

现在程序的主循环开始了。第 61 行的 if 条件打印出在函数 fprintf 参数中定义的信息。stderr 是 C 中的一个流函数，可以引导输出显示在屏幕上。在这种情况下，输入仅仅是一个转义序列 n，如果用户在命令行选择 y 或 1 作为 verbose 标志，这将使光标移

动到下一行（verb=1 即 verb=y）。

然后循环随着时间向前进行。紧随声明（在循环体内）有另一个 if 条件，和第一个类似但是它打印出的是 it 当前的值。这里就要提到多次出现的转义序列？？。再循环开始时它的值是 0，在屏幕上打印，当循环回到起始位置时更新的值为 1，因此 b（回格）删除了之前的值 0，并已经在屏幕上更新为 1.

66-76:

```
66          /* 4th order laplacian */
67          for (iz=2; iz<nz-2; iz++) {
68              for (ix=2; ix<nx-2; ix++) {
69                  ud[ix][iz] =
70                  c0* uo[ix ][iz ] * (idx+idz) +
71                  c1*(uo[ix-1][iz ] + uo[ix+1][iz ])*idx +
72                  c2*(uo[ix-2][iz ] + uo[ix+2][iz ])*idx +
73                  c1*(uo[ix ][iz-1] + uo[ix ][iz+1])*idz +
74                  c2*(uo[ix ][iz-2] + uo[ix ][iz+2])*idz;
75              }
76          }
```

这是四阶拉普拉斯计算。至于四阶这一点，我们的意思不是二阶偏微分方程本身的阶即二阶，而是近似的阶。二阶偏导数离散二阶近似可以写成：

$$\frac{\partial^2 U}{\partial x^2} = \frac{U_{i+1} + 2U_i + U_{i-1}}{\Delta x^2}$$

这是点 i 处 U 值二阶偏微分的中心差分公式。在 z 方向，类似的我们得到：

$$\frac{\partial^2 U}{\partial z^2} = \frac{U_{i+1} + 2U_i + U_{i-1}}{\Delta z^2}$$

相加这两项，我们便得到了精确到第二阶的中心差分拉普拉斯公式。但是我们现在使用的是精确到四阶的中心差分，因此我们有：

$$\frac{\partial^2 U}{\partial x^2} = \frac{1}{\Delta x^2}\left[-\frac{1}{12}U_{i+2} + \frac{16}{12}U_{i+1} - \frac{30}{12}U_i + \frac{16}{12}U_{i-1} - \frac{1}{12}U_{i-2}\right]$$

通过在 z 方向写出一个近似的方程然后将两者相加，我们得到了四阶近似的拉普拉斯方程，也可称为我们提到的“四阶拉普拉斯”。

现在我们回过头来看代码，第一行是在 z 方向上循环的开始。在 z 循环的循环体内

还有一个循环，这个循环把一个 z 值对应的所有 x 值运行了一遍。第二行是 x 方向 for-loop 的开始。

然后在 x 方向循环体中我们使用了一个之前定义的 2×2 矩阵。这个就是精确到四阶的拉普拉斯方程，正如上面讨论的用分离出的同样的系数？？。注意 x 和 z 的循环从 0 之后两个单元开始，到 nx 和 nz 之前两个单元结束。这是因为要估计在某一特定点（x，z）处拉普拉斯最远的值，我们使用后延两个单位和当前点（x，z）的值，如果我们将 iz=0，1；iz=nz-1，nz 和 ix=0，1；ix=nx-1，nx 的点包括进来，程序运行将出界。为了补足这一点，我们需要一个边界条件，这一边界条件将在下一个插入子波的循环中得到。

78-83:

```
78          /* inject wavelet */
79          for (iz=0; iz<nz; iz++) {
80              for (ix=0; ix<nx; ix++) {
81                  ud[ix][iz] -= ww[it] * rr[ix][iz];
82              }
83          }
```

这里插入了子波，意味着可以估计表达式 $\Delta U - f(t)$ 的值。ΔU 已经在前面的循环中计算了出来并以序列 ud 保存了起来。ww 是子波序列，但是在从？？拉普拉斯（ud）中减去小波之前我们将当前时刻的小波振幅与每一点处的反射系数在空间（x，z）作了相乘。这是来源于一个初始化条件：

$$f(x,z,0) = g(x,z) = ww(0)rr(x,z)\,,$$

这样就满足了在 ix，iz=0 ； ix-2，ix-1=0 和 iz-2，iz-1=0 处补充值的要求。

但是震源子波并不是一个理想的脉冲，它在一定时间上都存在振幅，因此每一次子波都要乘以每一点上的反射系数。为什么要将子波与反射系数相乘呢？是这样的，我们假设了一种情况，源在研究空间中的每一点处都被激发，并由在点（x，z）处得反射系数来度量它的值。意思就是，震源在所有发生声阻抗（因为反射系数是声阻抗在跨界面时的差与和的比值）变化的点上被激发。

85-90:

```
85          /* scale by velocity */
86          for (iz=0; iz<nz; iz++) {
87              for (ix=0; ix<nx; ix++) {
88                  ud[ix][iz] *= vv[ix][iz]*vv[ix][iz];
89              }
90          }
```

这里我们将 $\Delta U - f(t)$ 与速度相乘，即对 $(\Delta U - f(t))v^2$ 进行估值。

92-102:

```
92              /* time step */
93              for (iz=0; iz<nz; iz++) {
94                  for (ix=0; ix<nx; ix++) {
95                      up[ix][iz] = 2*uo[ix][iz]
96                                  - um[ix][iz]
97                                  + ud[ix][iz] * dt2;
98
99                      um[ix][iz] = uo[ix][iz];
100                     uo[ix][iz] = up[ix][iz];
101                 }
102             }
```

这里我们计算时间步长，即：

$$U_{i+1} = \left[\Delta U - f(t)\right]v^2\Delta t^2 + 2U_i - U_{i-1}$$

第一个 for 循环是 z 方向上的，在循环体内是在 x 方向上的另一个 for 循环。up 是在时间循环中控制在当前时刻波振幅的序列。uo 是包含在当前前一单位时刻的振幅序列，um 则是控制两个单位前的振幅序列。ud 是我们在前面的程序中计算出的序列，现在它与 Δt^2（dt2）作积并包含在了最后的方程中。这就完成了对 it 的一个值的计算。现在，数组需要更新到下一个时间步长。这一步的实现在最后两小部分中：第一个实现的是 $U_{i-1} \rightarrow U_i$，第二个实现的是 $U_i \rightarrow U_{i+1}$，这样，序列 um 就从 uo 更新了，而 uo 自身也通过 up 得到更新。

104-106:

```
104                 /* write wavefield to output */
105                 sf_floatwrite(uo[0],nz*nx,Fo);
106             }
```

在一个时间步长的计算完成之后，我们写进数组 uo（记住 uo 得到与 up 相等的值，即当前时间步长，这是前面几行实现的）。为了将数组写到输出文件，和之前用 sf_floatread 从输入文件中读取一样，我们用 sf_floatwrite（P109），唯一的不同就是作为参数给出的数组作为最后一个参数被写入了文件。大括号关闭的是时间循环，在这个时间循环之后，下一个时间值的循环将重新进行一边。

107-109:

```
107             if(verb) fprintf(stderr,"\n");
108             sf_close()
109             exit(0);
110         }
```

第一行代码在时间循环运行完所有时间值后将光标放在屏幕上新的一行。

第二行用 RSF API 的 sf_close（P113）删除了临时文件。

第三行用 C 语言中的 exit()函数关闭了流程，将控制返还给主机环境。参数中的 0 表示程序正常终止。最后一个大括号关闭了主函数。

5.3 数据格式

这一节主要描述 RSF API 中使用的数据格式。

5.3.1 复数和 FFT

以下列举复数的数据格式和快速傅里叶变换（FFT）。

kiss_fft_scalar

这个数据格式定义了一个 kiss_fft_cpx 复数格式的标量实数值。它既可以是短整型（short），也可以是浮点型（float）。默认是浮点型（float）。

kiss_fft_cpx

这个数据格式（C 的结构体）定义了一个复数。它包含了复数的实部和虚部，可以被定义为 kiss_fft_scalar 类型。

kiss_fft_cfg

这是一个 kiss_fft_state 类型（一个 C 数据结构体）的对象。

kiss_fftr_state

kiss_fftr_state 数据格式定义了傅里叶变换所需的变量并分配了所需的空间。这和 kiss_fft_state 具有同样的目的，但是这是对实数信号的傅里叶变换。

sf_complex

这是 kiss_fft_cfg 类型的对象（C 数据格式的一个对象）。

sf_double_complex

这是一个复数的 C 数据结构。它对复数的实部和虚部使用双精度格式（double）。

5.3.2 文件

这一部分列举了定义.rsf 文件结构的数据格式。

sf_file

它是 sf_File 类的一个对象。sf_File 是定义在 Madagascar 创建.rsf 文件所需的变量的数据结构。在 file.c 中给出了它的定义。

sf_datatype

这是一个 C 中的枚举，意思是它包含了不同于像 int，float，sf_file 等基本类型的新数据类型。在 sf_File 数据结构中使用这个数据类型来设置 a.rsf 文件的类型，例如 SF_CHAR，SF_INT 等。它的定义在 file.c 中。

sf_dataform

这也是一个 C 的枚举，意思是它包含了不同于像 int，float，sf_file 等基本类型的新数据类型。在 sf_File 数据结构中使用这个数据类型来设置.rsf 文件的格式，例如 SF_ASCII，SF_XDR 和 SF_NATIVE。在 file.c 中给出了它的定义。

5.3.3 运算

这一部分列出了定义线性运算的数据格式。

sf_triangle

第 6 章 Python 编写处理流程

本章介绍可重复性计算实验的环境，这一环境已经成为“Madagascar”软件包的一个组成部分。如果想重复作本章中提到的实验案例，您可以从 http://www.ahay.org 下载 Madagascar。

同一领域的学者之间对研究的相互交流批评是科学进步的基石。古老的炼金术师，他们在不为人知的地方进行秘密工作，试图用一些奇异的解决方案攻克那些无解的问题。从那时起到今天，现代科学走过了漫长的道路而成为了全社会经营的产业，其中，假说、理论和试验结果被公开发表并在由公众来证实。通过不断的重复和验证前人研究的结果，研究人员能够让科学发展再向前迈进一步。传统意义上，科学的学科可以分为理论和实验研究。理论成果的重复和验证通常只需要想象力（而抛弃纸笔），而验证试验结果需要在实验室中使用和那些出版物中描述类似的设备和材料。在上个世纪中，计算的研究作为一门新的科学学科展现在大家面前。在计算机上进行的计算实验是将数值算法应用于数字式的数据。为什么这些实验是可重复性的呢？一方面，重复作数值实验的结果是一项艰巨的任务。读者必须得到和出版作者精确相同的一类输入数据、软件、硬件，这样才能重新生成那些公开发表出来的结果。

6. 1 介绍

用 SCons 进行可重写的计算实验

SCons（来自 Software Construction）是一个最初为开发软件设计的著名的开源程序。在本文中，我们介绍了对 SCons 进行扩展来对数据进行处理以及可重写的数值实验。用两个简单的例子说明了 SCons 的用途。

6. 1. 1 引言

本文介绍了作为 Madagascar 软件包的一部分的可重写数值实验环境。要想对本文中的例子进行重写，你可以从 http://www.ahay.org/ 下载 Madagascar。目前主要的 Madagascar 界面是 Unix 内核命令行，因此你需要安装 Unix/POSIX 系统（Linux，Mac OS，Solaris 等）或者 Windows 下运行的类 Unix 环境（Cygwin，SFU 等），我们的关注点不仅是在研究中用到的特殊的工具，还有可重写计算的思想。

Reproducible research philosophy 可重写性研究思想

6. 1. 2 可重写研究的思想

同行的评论是科学进步的支柱。从古代的炼丹师——他们秘密地寻找魔力方案来解决无法解决的问题——到现代科学已经有了很大的发展，成为了一种社会性的事业，在这里假设、理论以及实验结果都公开地发表并被检验。通过重写和验证以前发表的研究，研究者可以在科学上迈出新的一步。通常，科学学科被分为理论研究和实验研究。理论研究的重写和验证只需要想象（除了纸和笔），实验结果的验证需要在实验室使用与出版物中描述的相近的仪器和材料。上个世纪，数值计算作为一门新的科学学科出现了，它通过使用数值算法计算数据。这些实验的可重写性怎样呢？一方面重现数值实验结果

是一项很难的事。要想得到发表的结果，读者需要准确地知道输入数据、使用的软件及硬件。通常这些条件不能完全满足。另一方面，由于基本条件如操作系统，文件格式越来越趋向于标准化，理论上新的成分可以共享因为它们只是表示网络上可以传输的的数字信息。通过软件共享，Linux、Apache 以及其它许多软件项目发展很快。文件共享的支持者将这种思想看作与科学研究同行的评论一样。著名的开源倡导者，Eric Raymond 写道（Raymond, 2004[1]）：

抛弃秘密研究，开始透明地研究并让同行评论是炼丹术成为化学的原因。同样，开源的发展可能是期待已久的软件行业发展成熟的一个标志。

尽管软件发展在效仿科学，计算科学要想保持为一个完完全全的科学学科，就需要从开源的例子借鉴经验。这是一位成绩卓越的数学家 ndy LeVeque 的话（LeVeque, 2006[2]）。

在科学的世界中，计算现在已经被看作继实验和理论之后的第三个高点。但是，亦可以 举出很多例子说明计算是许多科学恶棍的庇护所。还有哪一个学科可以不用对研究方法进行详细描述、不用让其他人可以重复实验来发表一个声称证明了一项技术的文章？科学及数学期刊上到处都是数值实验得到的漂亮图片，但是读者无法重新实现。即使是很聪明的并且希望做好的计算科学家也无法很好地用可重写的方式来展示他们的工作。所用的方法定义模糊，即使详细定义也需要读者从头做起来对其进行检验。

在计算机科学中，发表和结束计算机程序追溯到 Knuth(1984[3])提倡的精通编程的思想，这种思想得到了很多研究者的发展（Thimbleby, 2003[4]）。在 Harold Thimbleby2004 年的讲座[5]中，他提到：

我们需要的思想是通用性，尤其是程序。一种客观的方式是科学必须在除了研究者的实验室或研究者的想象中之外发挥作用；这种要求称为 reproductivity 可重写性。

近十年前，可重写技术在地球物理方面的研究是从 SEP 的 Jon Claerbout 和他的学生开始的 SEP 的可重写研究体系要求文章的作者将从输入数据和软件源编辑整理为数值结果的文档，以供他人测试和验证(Claerbout, 1992a[6];Schwab et al., 2000[7])。

可重写性研究学科也被统计和小波理论的研究者 Buckheit 和 Donoho（1995[8]）采用和推广，它被数本著名的小波理论的书引用（Hubbard, 1998[9];Mallat, 1999[10]）。可重写研究在多个领域被推广，如生物信息学（Gentleman et al., 2004[11]）、地质信息学（Bivand, 2006[12]）以及就算波动传播（LeVeque, 2006[13]）等。但是，可重写研究在计算家中的发展缓慢，这在一定程度上是由于其复杂性及工具的不完备。

6. 1. 3 可重写性研究工具

可重写研究工具

Stanford 提出的可重写研究体系的基础是“make”（Stallman et al., 2004[14]）——一种 Unix 的创建功能。起初 SEP 使用“cake”作为“make”的近似（Nichols and Cole, 1989[15];Claerbout and Nichols, 1990[16];Claerbout, 1992b[17];Claerbout and Karrenbach,

1993[18]）。后来由 Schwab 和 Schroeder (1995[19])转变为更标准的表达“GNU make”。“make”程序记录了创建出图片或者稿件之后，系统的不同成分以及软件创建目标的相互联系。软件的构建目标和命令行是用户在“makefiles”中指定的，作为定义源和目标相互联系的资料库。基于相互联系的体系由于仅需要改动相应的代码，因此开发快。Buckeit 和 Donoho（1995[20]）将这种思想用于 MATLAB——由 MathWorks 开发的综合开发环境（Sigmon and Davis, 2001[21]）。虽然 MATLAB 对于数值算法的研究很先进，但是对于像计算地球物理这样的大型计算它就有些不合适了。“make”是许多软件开发项目中非常重要的工具。但是从用户经验的角度，难以设计好。“Make”使用了一种难懂的并且数量有限的特殊语言（Unix 内核命令与特殊用途的命令的混合），这常常使缺乏经验的用户感到困惑。来自 Sun 公司的一位软件专家 Peter van der Linden 认为（van der Linden, 1994[22]）：

“Sendmail”和“make”是两个被广泛认为经过最初调试就开始使用的程序，因此它们的命令语言很难懂。不仅是你——每个人都发现它难懂。

“make”命令的难懂限制了它的应用。Schwab 等人（2000[23]）开发的研究系统不仅包括传统的“make”规范，还包括了另种一晦涩的内核语言和 Perl 脚本来对“make”进行扩展（Fomel et al., 1997[24]）。近些年开发了一些用于软件构建相关性检测的系统，其中一个最有前途的工具是 SCons，Dubois 对其非常认可（2003[25]）。SCons 的优点如下：

SCons 配置文件是 Python 脚本写的。Python 是一种以其可读性、美观性、简洁性以及强大的功能著称的一种现代编程语言（Rossum, 2000a[26];Rossum, 2000b[27]）。Scales 和 Ecke （2002[28]）将 Python 作为主要的编程语言推荐给他们学地球物理的学生。

SCons 提供可信的、自动的并可扩展的依赖性分析，对所有的依赖性进行全局查看–不再需要 "make depend", "make clean", 或者多次对目标文件进行记录来得到所有的依赖关系。

SCons 含有内置的对多种语言的支持，如: C, C++, Fortran, Java, LaTeX, 等。

与 "make"依赖于时间戳来检测文件变动的方式不同 (对于具有不同系统时钟的平台，会产生许多问题，SCons 默认使用了一种通过 MD5 标记的更可信的检测机制。它不仅可以检测文件中的变化，还可以检测命令行中的变化。

SCons 为并行编译提供了完整的支持。

SCons 为检测不同平台的环境提供了类似于“autoconf”功能的配置支持。

SCons 是由交叉式平台工具发展来的。在 POSIX 系统 (Linux, Mac OS X, Solaris, 等) 与 Windows 系统上都可以很好地运行。

The stability of SCons 的稳定性保证是通过大量的回归测试来逐步完善的。

SCons 是在开源许可下公开发布的[29].

本文中，我们将 SCons 作为一个科学计算中可重写研究一个新的平台。

6. 1. 4 论文组织结构

论文组织结构

为了证明 SCons 在可重写性研究中的应用，我们首先描述两个例子，然后介绍 SCons 怎样帮助我们整理计算结果。

6. 2 实验举例

实验举例

我们研究中用到的 Sconstruct 命令在下表中给出。

Basic methods of an rsf.proj object.

Fetch(data_file,dir[,ftp_server_info])

A rule to download < data_file > from a specific directory < dir > of an FTP server

从 FTP 服务器的特定路径<dir>下载<数据文件>的方法

Flow(target[s],source[s],command[s][,stdin][,stdout])

A rule to generate < target[s] > from < source[s] > using < command[s] >

使用<源文件[s]>中的<命令[s]>生成<目标文件[s]>

Plot(intermediate_plot[,source],plot_command) or
Plot(intermediate_plot,intermediate_plots,combination)

A rule to generate < intermediate_plot > in the working directory.

在当前路径下生成<中间图>的方法

Result(plot[,source],plot_command) or
Result(plot,intermediate_plots,combination)

A rule to generate a final < plot > in the special Fig folder of the working directory.

在当前路径下特定 Fig 文件夹中生成最终<图件>的方法

End()

A rule to collect default targets.

收集默认目标文件的方法

这些命令在$PYTHONPATH/rsf/proj.py 中定义，其中 RSFROOT 是安装 Madagascar 的目录。本文件的源文件在 framework/rsf/proj.py 中。

6. 2. 1 例 1

选择一个工作项目路径并将以下代码复制到名为 SConstruct[30]的文件中。

```
from rsf.proj import *


# Download the input data file
```

```
Fetch('lena.img','imgs')

# Create RSF header
Flow('lena.hdr','lena.img',
     'echo n1=512 n2=513 in=$SOURCE data_format=native_uchar',
     stdin=0)

# Convert to floating point and window out first trace
Flow('lena','lena.hdr','dd type=float | window f2=1')

# Display
Result('lena',
       '''
       sfgrey title="Hello, World!" transp=n color=b bias=128
       clip=100 screenratio=1
       ''')

# Wrap up
End()
```

这是我们的“hello world”例子用来说明表中的部分命令。这个实验的目的是将数据从公共服务器中下载，转换为合适的文件格式并生成图片或报告。但是我们仔细观察 Sconstruct 脚本并分析每条语句。

```
from rsf.proj import *
```

是标准 Python 命令语句，它将 Madagascar 项目管理模块上传，提供了 Scons 的扩展。

```
Fetch('lena.img','imgs')
```

让 SCons 与公共数据服务器连接（若没有提供 FTP 信息，就选择默认的服务器），从路径 data/imgs 路径中找到数据文件 lena.img。

在命令行运行“scons lena.img”。正确的输出应该是

```
bash$ scons lena.img
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
```

```
retrieve(["lena.img"], [])
scons: done building targets.
```

目标文件 lena.img 会显示在你的路径下。在下面的例子中，我们使用 SCons 的-Q（quiet）选项来压缩冗长的输出。

```
Flow('lena.hdr','lena.img',
        'echo n1=512 n2=513 in=$SOURCE data_format=native_uchar',
        stdin=0)
```

使用标准的 Unix 命令 echo 准备了 Madagascar 头文件 lena.hdr。

```
bash$ scons -Q lena.hdr
echo n1=512 n2=513 in=lena.img data_format=native_uchar > lena.hdr
```

由于 echo 没有标准的输入，stdin 被置为 0，否则第一行应该是标准的输入。同样，第一个目标在没有明确时应该是标准的输出文件。

注意 lena.img 在命令行中代表$SOURCE。这样我们就可以在不改变命令前提下来改变源文件的名称。Lena.img 文件的格式是 unchar（不带符号字符型）。图像含有 513 道，每道有 512 个采样点。下一步的任务是将图片转换为实型数据并除去第一道，这样就得到了 512×512 的方阵。使用 Unix 管道可以将这两个步骤合并为一步：

```
Flow('lena','lena.hdr','dd type=float | window f2=1')
```

```
bash$ scons -Q lena
   scons: *** Do not know how to make target `lena'.    Stop.
```

会发生什么情况呢？没有后缀，流程命令假设目标文件的后缀是“.rsf”。下面试一下。

```
scons -Q lena.rsf
< lena.hdr /RSF/bin/sfdd type=float | /RSF/bin/sfwindow f2=1 > lena.rsf
```

注意到了 Madagascar 模块 sfdd 和 sfwindow 用 Sconstruct 文件中的名称替换了。文件 lena.rsf 是常用的格式[31]，可以用 sfin lena.rsf[32]来检查。

```
bash$ sfin lena.rsf
lena.rsf:
        in="/datapath/lena.rsf@"
        esize=4 type=float form=native
        n1=512              d1=1                o1=0
```

```
        n2=512          d2=1            o2=1
            262144 elements 1048576 bytes
```

最后一步中，我们为了在屏幕上显示图片或为发表文章提供图片而创建绘图文件。

```
Result('lena',
        '''
        sfgrey title="Hello, World!" transp=n color=b bias=128
        clip=100 screenratio=1
        ''')
```

注意：我们用 Python 的三步引用的语法将命令语句分为很多行。当多行字符串被翻译为命令行时，中间的空格会被忽略。结论命令行含有与之联系的特定目标。例如，“scons lena.view”用来查看在特定的 Fig 路径下生成的图片 Fig/lena.vpl 并将它显示在屏幕上。输出应该是这样：

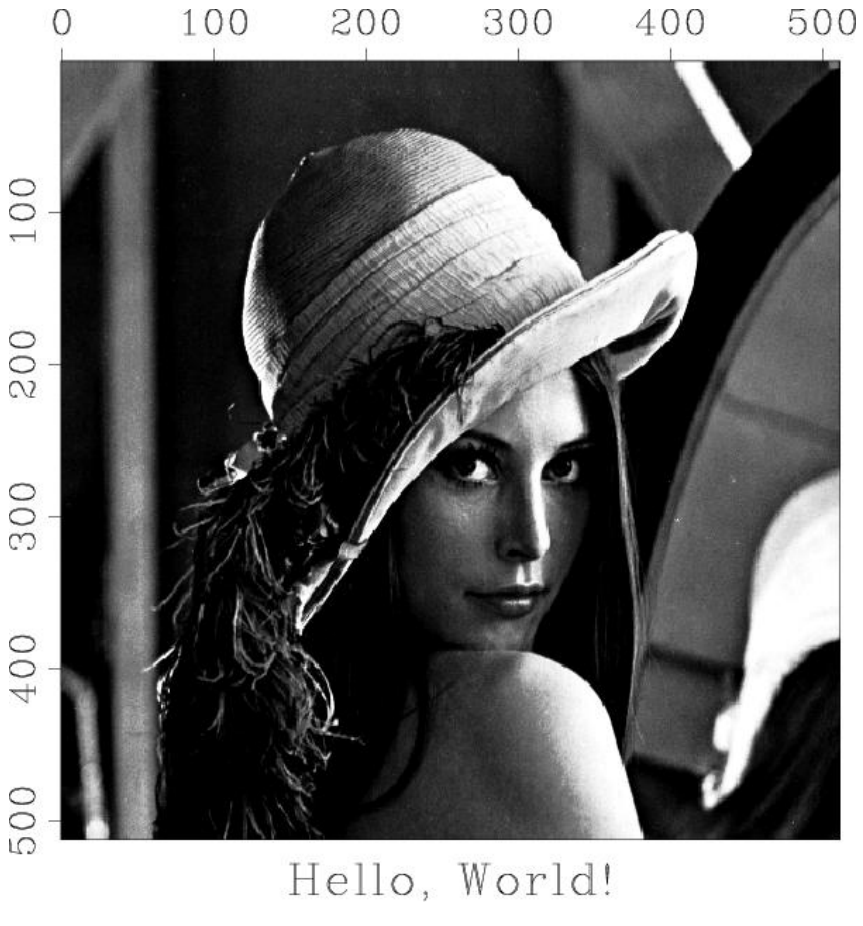
第一个数值实验的输出。

可重写性脚本用以下命令结束：

```
End()
```

你准备好试试了吗？试试下面的：

运行 scons -c。其中-c 选项告诉 SCons 删除所有的默认目标文件（本例中的 Fig/lena.vpl 图片）以及中间过程中生成的目标文件。



```
bash$ scons -c -Q
Removed lena.img
Removed lena.hdr
Removed lena.rsf
Removed /datapath/lena.rsf@
Removed Fig/lena.vpl
```

再次运行 scons 会重新生成默认目标文件。

```
bash$ scons -Q
retrieve(["lena.img"], [])
echo n1=512 n2=513 in=lena.img data_format=native_uchar > lena.hdr
< lena.hdr /RSF/bin/sfdd type=float | /RSF/bin/sfwindow f2=1 > lena.rsf
< lena.rsf /RSF/bin/sfgrey title="Hello, World!" transp=n color=b bias=128
clip=100 screenratio=1 > Fig/lena.vpl
```

编辑 Sconstruct 文件并改变一些绘图参数。例如，将 clip=100 中的值改为。重新运行 scons 后观察处理流程的最后部分（准确地讲，是参数变化影响的部分）的运行：

```
bash$ scons -Q view
< lena.rsf /RSF/bin/sfgrey title="Hello, World!" transp=n color=b bias=128
clip=50 screenratio=1 > Fig/lena.vpl
sfpen Fig/lena.vpl
```

SCons 可以识别你的编辑不会对数据流程中前面的结果产生影响！时刻保留依赖关系是使用 SCons 进行与使用线性内核脚本数据处理和计算实验的主要区别。对于计算要求高的数据处理，这个特点会为你节省很多时间并使你的实验的交互性更好。

SCons 中的一个特殊的参数（在 rsfproj.py 中定义）可以记录处理流程中执行每一步的时间。试试运行 scons TIMER=y 命令。

Rsfproj 模块可以直接找到存储所有的 Madagascar 模块参数的数据库。运行 scons CHECKPAR=y 来查看在实施计算\补充说明之前的校验参数｛这个特性是新的，还在试验之中，可能运行不正常｝。

对 SCons 命令的总结在下表中给出。

```
SCons 命令和选项在 rsfproj 中定义
scons < file >
```

生成 < 文件 > (通常需要为 Flow 目标文件添加后缀 .rsf 为 Plot 目标文件添加 .vpl 后缀。

```
scons
```

Generate default targets (usually figures specified in Result.)

生成默认目标文件（通常是 Result 中确定的图）

```
scons view or scons < result > .view
```

Generate Result figures and display them on the screen.

生成最终图片并显示在屏幕上

```
scons print or scons < result > .print
```

Generate Result figures and print them.

生成最终图片并打印

```
scons lock or scons < result > .lock
```

Generate Result figures and install them in a separate location.

生成最终图片并将其放在固定位置

```
scons test or scons < result > .test
```

Generate Result figures and compare them with the corresponding "locked" figures stored in a separate location (regression testing).

生成最终图片（回归测试）

```
scons < result > .flip
```

Generate the < result > figure and compare it with the corresponding "locked" figure stored in a separate location by flipping between the two figures on the screen.

生成<结果>并与存储在里一个位置的相应的“locked”图片通过快速显示在屏幕上比较

```
scons TI 图片 MER=y ...
```

Time the execution of each step in the processing flow (using the Unix time utility.)

记录处理流程中执行每一步所用的时间（采用 Unix 计时功能）

```
scons CHECKPAR=y ...
```

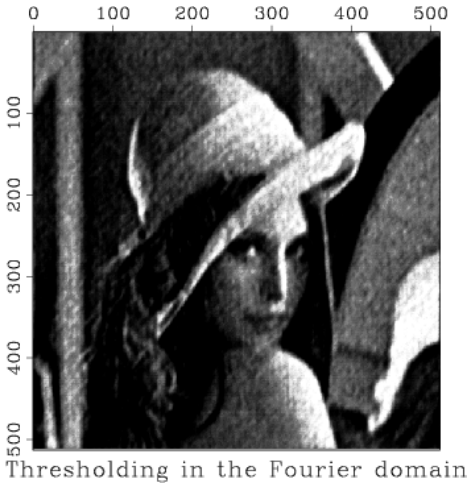
Check the names and values of all parameters supplied to Madagascar modules in the processing flows before executing anything (guards against incorrect input.) This option is new and experimental

在执行命令之前，检查在处理流程中提供给 Madagascar 模块的所有参数的名称和值（检查非法的输入）。这个选项是新增加的，还在试验中。

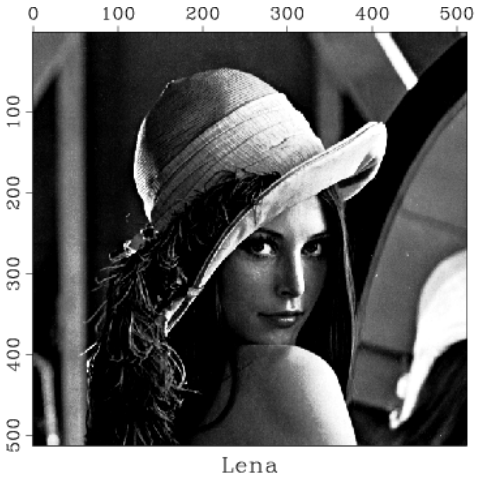
6. 2. 2 例 2

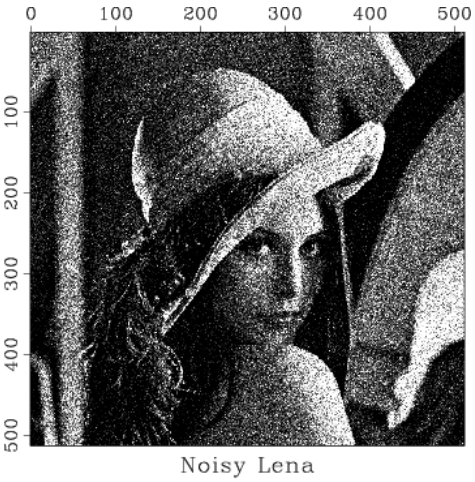
这个实验的目标是在“Lena”图像加上随机噪声然后试着用低通滤波器及在傅立叶域中设置较大的门槛系数来对其进行移除。处理的成果图片如下所示：

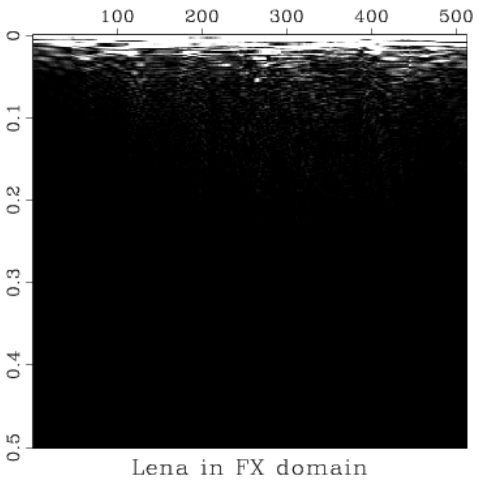
左上图：初始图片，右上图：加入随机噪声的图片；左下图：初始图片在傅立叶（F-X）域的频谱；右下图：含噪声的图片在傅立叶（F-X）域的频谱。

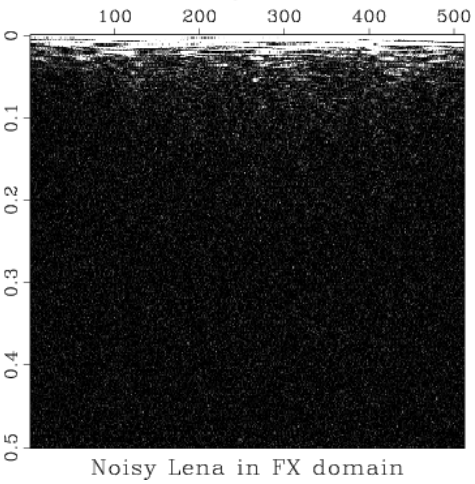


左图：低通滤波去噪结果。右图：傅立叶域设立高门槛值（hard threshouldinig）去噪结果。









由于 SConstruct|文件是一个 Python 脚本，我们可以在 Madagascar 可重写脚本中充分发挥 Python 语言的灵活性及强大功能。在 Madagascar 的 book 路径中 rsf/scons/rsfpy 中有一个脚本的例子。我们选择了一些感兴趣的部分进行分析。在 SConstruct 脚本中，我们声明 Python 变量

```
bias = 128
```

之后会用到它，例如，将我们自定义的绘图命令做成一个 Python 函数：

```
def grey(title,transp='n',bias=bias):
    return '''
    sfgrey title="%s" transp=%s bias=%g clip=100
    screenht=10 screenwd=10 crowd2=0.85 crowd1=0.8
    label1= label2=
    ''' % (title,transp,bias)
```

这个名叫 grey（）的 Python 函数可以被 Plot 或这 Result 命令调用，例如

```
Plot('lplena',grey('Noisy Lena LP filtered'))
```

我们可以定义一个 Python 字典，如

```
titles = {'lena':'Lena',
          'nlena':'Noisy Lena'}
```

在它的输入文件中进行循环，如

```
for name in titles.keys():
    Plot(name,grey(titles[name]) )
    cftitle = titles[name]+' in FX domain'
    Flow('fx'+name,name,'sfspectra')
    Plot('fx'+name,grey(cftitle,'y',100))
```

注意图名是通过与 Python 字符串连接得到的。Python 字符串也可以用于定义在数据流程中命令的使用顺序，如

```
# 2-D FFT
fft2 = 'sffft1 sym=y | sffft3 sym=y'
Flow('fnlena','nlena',fft2)
```

最后，在 Madagascar 可重写脚本中，我们希望在运行 SCons 或者使用默认值的时候令选项跳过命令行参数，例如

```
# denoising using thresholding in the Fourier domain
fthr = float(ARGUMENTS.get('fthr', 70))
Flow('fthrlena','fnlena','sfthr thr=%f mode="hard"' % fthr)
```

仅运行 scons 时使用的是 ftgr 的默认值（如 70），而运行 scons fthr=68 为 fthr 赋了一个确定的值。这尽管不是选项的详细列表，但是它为你提供了一种功能强大的工具，请充分使用以发挥你的才能！

6. 3 创建可重写文档

在进行完数值实验之后，就需要将其与写论文联系起来了。SCons 帮助我们创建了高品质的论文，其中计算结果（图）用 Latex 写入到论文中。相应的 SCons 扩展在

$PYTHONPATH/rsf/tex.py 中定义，其中 RSFROOT 是 Madagascar 的安装目录。这个文件安的源文件在 framework/rsf/tex.py 中。我们将基本的方法和命令列举在下面的表中。

Basic methods of an rsf.tex object.

rsf.tex 的基本使用方法

Paper(paper_name,[,lclass][,use][,include][,options])

在 options 设定了选项，use 中明确了要用的包以及在 include 中设定了前言之后，用 1class 中确定的 Latex2e 类(默认使用 SEGTeX 包中的 geophysics.cls)来编译< paper_name > .tex LATEX\ 文档的方法

End()

收集默认目标文件（指 paper.tex 文档）的方法

rsfrex 中定义的 SCons 命令

scons

生成默认的目标文件（通常是由 Latex 源文件 paper.tex 转化成 PDF 文件 paper.pdf）

scons pdf or scons < paper_name > .pdf

由 Latex 源文件 paper.tex 或者 < paper_name > .tex 生成 PDF 文件

scons read or scons < paper_name > .read

由 Latex 源文件 paper.tex 或者 < paper_name > .tex 生成 PDF 文件并显示在屏幕上。

scons print or scons < paper_name > .print

由 Latex 源文件 paper.tex 或者 < paper_name > .tex 生成 PDF 文件并打印

scons html or scons < paper_name > .html

由 Latex 源文件 paper.tex 或者 < paper_name > .tex 生成 HTML 文件，建立路径< paper_name > _html

scons install or scons < paper_name > .install

由 Latex 源文件 paper.tex 或者 < paper_name > .tex 生成 PDF 及 HTML 文件并且将它们放在一个另外的位置（用来发布到网站上）

```
scons wiki or scons < paper_name > .wiki
```

将 Latex 源文件 paper.tex 或者< paper_name > .tex 转化成 MediaWiki 格式（用来在 Wiki 网站上发布）

6. 3. 1 举例

论文的本身就是可重写的文档，它是使用放在项目路径之上的路径中用 SConstruct 文件生成的文件。

```
from rsf.tex import *
Paper('velan',use='hyperref,listings,color')
End(use='hyperref,listings,color')
```

SConstruct 生成了这个论文，它也可以在相同的路径下编译 velan.tex。注意由于 paper.tex 是默认文件名，没有与 paper 对应的命令。在 End 命令中跳过了 paper.tex 所用的可选的 Latex 包和类型。

下面我们看看文档中的图片是怎样与生成它们的可重写性脚本相联系的。首先，注意到 paper.tex 不是常规的 Latex 文档，仅是它的主体（没有\文档类, \用户包, 等）。在我们的额文章中，第一个图是用项目文件夹 easystart（文档目录下的子文件夹）对 Lena.vpl 操作得到的。在 Latex 源代码中，是这样翻译的

```
\inputdir{easystart}
\sideplot{lena}{height=.25\textheight}{The output of the first numerical experiment.}
```

\inputdir 命令指向项目路径而 \sideplot 命令调用 < result_name > 。这个图片的 LATEX 标签是 fig: < result_name > 。当文件第一次编译完成以后，生成的文件会自动转换为 PDF 格式。

参考资料

↑ Raymond, E. S., 2004, The art of UNIX programming: Addison-Wesley.

↑ LeVeque, R. J., to appear, 2006, Wave propagation software, computational science, and reproducible research: Presented at the Proc. International Congress of Mathematicians.

↑ Knuth, D. E., 1984, Literate programming: Computer Journal, 27, 97--111.

↑ Thimbleby, H., 2003, Explaining code for publication: Software - Practice & Experience, 33, 975--908.

↑ http://www.uclic.ucl.ac.uk/harold/

↑ Claerbout, J., 1992a, Electronic documents give reproducible research a new meaning:

62nd Ann. Internat. Mtg, 601--604, Soc. of Expl. Geophys.

↑ Schwab, M., M. Karrenbach, and J. Claerbout, 2000, Making scientific computations reproducible: Computing in Science & Engineering, 2, 61--67.

↑ Buckheit, J. and D. L. Donoho, 1995, Wavelab and reproducible research, in Wavelets and Statistics, volume 103, 55--81. Springer-Verlag.

↑ Hubbard, B. B., 1998, The world according to wavelets: The story of a mathematical technique in the making: AK Peters.

↑ Mallat, S., 1999, A wavelet tour of signal processing: Academic Press.

↑ Gentleman, R. C., V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang, and J. Zhang, 2004, Bioconductor: open software development for computational biology and bioinformatics: Genome Biology, 5, R80.

↑ Bivand, R., 2006, Implementing spatial data analysis software tools in r: Geographical Analysis, 38, 23--40.

↑ LeVeque, R. J., to appear, 2006, Wave propagation software, computational science, and reproducible research: Presented at the Proc. International Congress of Mathematicians.

↑ Stallman, R. M., R. McGrath, and P. D. Smith, 2004, GNU make: A program for directing recompilation: GNU Press.

↑ Nichols, D. and S. Cole, 1989, Device independent software installation with CAKE, in SEP-61, 341--344. Stanford Exploration Project.

↑ Claerbout, J. F. and D. Nichols, 1990, Why active documents need cake, in SEP-67, 145--148. Stanford Exploration Project.

↑ -------- 1992b, How to use Cake with interactive documents, in SEP-73, 451--460. Stanford Exploration Project.

↑ Claerbout, J. F. and M. Karrenbach, 1993, How to use cake with interactive documents, in SEP-77, 427--444. Stanford Exploration Project.

↑ Schwab, M. and J. Schroeder, 1995, Reproducible research documents using GNUmake, in SEP-89, 217--226. Stanford Exploration Project.

↑ Buckheit, J. and D. L. Donoho, 1995, Wavelab and reproducible research, in Wavelets and Statistics, volume 103, 55--81. Springer-Verlag.

↑ Sigmon, K. and T. A. Davis, 2001, MATLAB primer, sixth edition: Chapman & Hall.

↑ van der Linden, P., 1994, Expert C programming: Prentice Hall.

↑ Schwab, M., M. Karrenbach, and J. Claerbout, 2000, Making scientific computations reproducible: Computing in Science & Engineering, 2, 61--67.

↑ Fomel, S., M. Schwab, and J. Schroeder, 1997, Empowering SEP's documents, in

SEP-94, 339--361. Stanford Exploration Project.

↑ Dubois, P. F., 2003, Why Johnny can't build: Computing in Science & Engineering, 5, 83--88.

↑ Rossum, G. V., 2000a, Python reference manual: Iuniverse Inc.

↑ -------- 2000b, Python tutorial: Iuniverse Inc.

↑ Scales, J. A. and H. Ecke, 2002, What programming languages should we teach our undergraduates?: The Leading Edge, 21, 260--267.

↑ As of time of this writing, SCons is in a beta version 0.96 approaching the 1.0 official release. See http://www.scons.org/.

↑ The source of this file is also accessible at book/rsf/scons/easystart/SConstruct.

↑ See Guide to RSF file format

↑ See Guide_to_madagascar_programs#sfin.

Retrieved from "http://reproducibility.org/wiki/Reproducible_computational_experiments_using_SCons

第 7 章 地震正演流程案例

7.1 有限差分正演

```
##########################################################################
#       acoustic modeling (simple half-space model)
#       -multi source-
#       -Author :    GuoQiang
#       -Date :       2011/06/08
#       -Union :      Leon
##########################################################################

from rsf.proj import *
import fdmod,os
os.environ['DATAPATH']='./'

Program('AFDM.x',['AFDM.c','fdutil.c','omputil.c'])

# ------------------------------------------------------------
#input parameter
par = {
      'nx':3601, 'ox':0, 'dx':0.002,    'lx':'x', 'ux':'km',
      'nz':1001, 'oz':0, 'dz':0.002,    'lz':'z', 'uz':'km',
      'nt':6000, 'ot':0, 'dt':0.0004, 'lt':'t', 'ut':'s',
      'kt':150,
      'jsnap':250,
      'jdata':2,
      'height':10,
      'nb':250,
      'frq':45,
      }
fdmod.param(par)

#jsnap=nt

par['zsou']=10*par['dz']
```

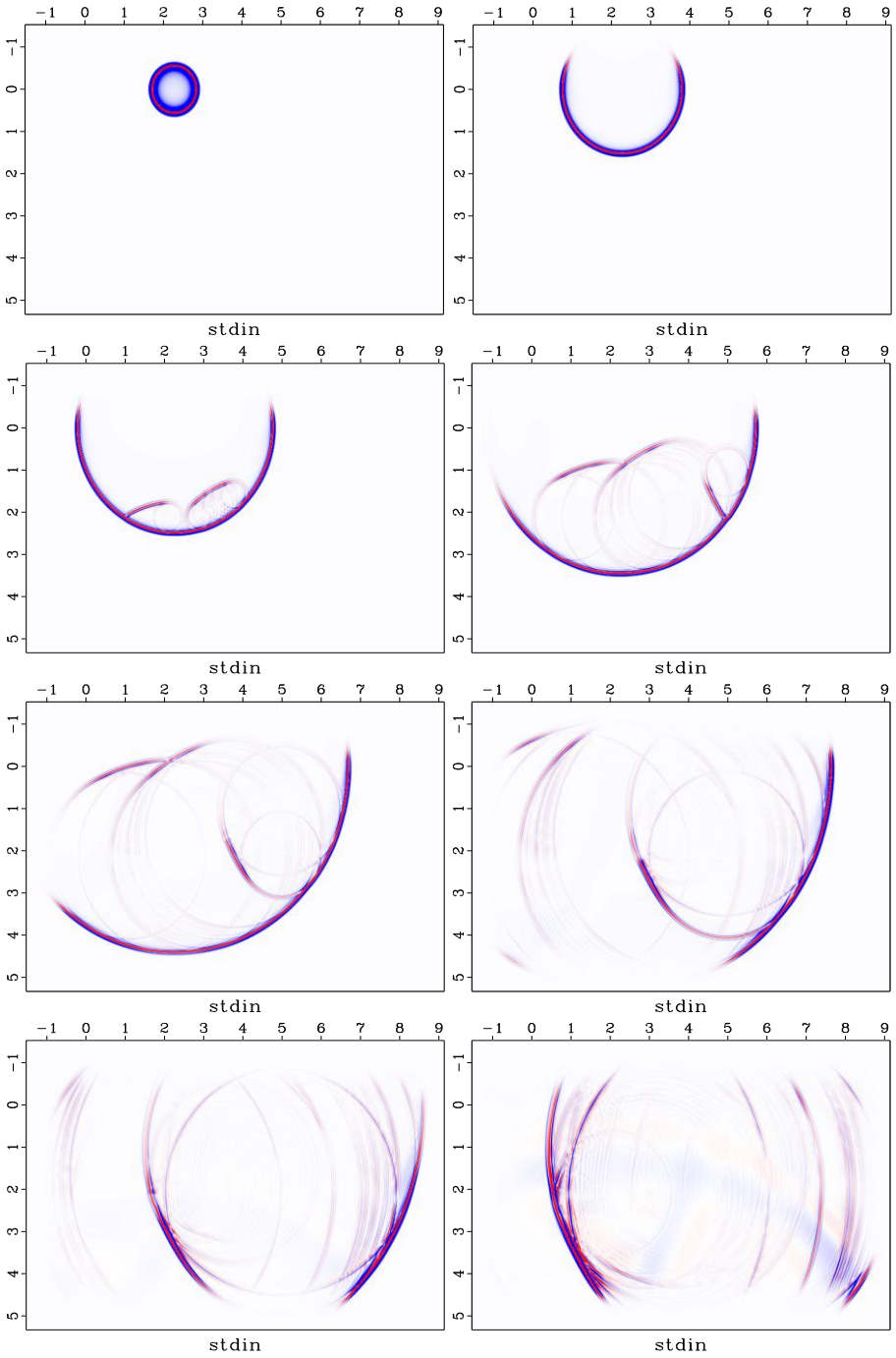
```
fdmod.horizontal('rr0',0,par)

# ------------------------------------------------------------
# input/output files
output1='shot'

# ------------------------------------------------------------
# source coordinates set ()
# Rule must be followed : ( nx+(ns-1)*ds ) <= vnx

while True:
    print '################################################'
    print 'vnx=%d' %par['nx']
    print 'dx=%2.4f' %par['dx']
    print '################################################'
        print ' Attention : Describe the parameters using number of sampling point.    '
        print ' The parameter should be changed! '
    ns=input(" >>>Input nsou: ")
    nx=input(" >>>Input ntra: ")
    ds=input(" >>>Input ds: ")
    if ( nx+(ns-1)*ds ) <= par['nx'] :break

# ------------------------------------------------------------
# Modify the vp field
par['kz']=2./3.*par['nz']

# ------------------------------------------------------------
# Wavelet built
fdmod.wavelet('wav_',par['frq'],par)
# ------------------------------------------------------------
# acoustic source
Flow(    'wava','wav_','transp')
# Result('wava','transp |' + fdmod.waveplot('',par)) # This command has some unknown problem in scons

# ------------------------------------------------------------
```

```
# source/receiver coordinates

shots = range(nx-1,par['nx']-nx,ds)
nshots = len(shots)
for i in shots:
        tag = "-%04d" %i
        par['xsou'] = par['ox']+par['dx']*i
        os=par['xsou']-300*par['dx']
        fdmod.point('ss'+tag,par['xsou'],par['zsou'],par)
        print os
        Flow('rr'+tag,'rr0','window min2=%f|window j2=2 n2=301' % os,local=1)
        #Plot('ss'+tag,fdmod.ssplot('',par))

allpang = map(lambda x: 'ss-%04d' % x,shots)
Flow('ss',allpang,'cat axis=3 space=n ${SOURCES[0:%d]}'%nshots,stdin=0,local=1)

allpang = map(lambda x: 'rr-%04d' % x,shots)
Flow('rr',allpang,'cat axis=3 space=n ${SOURCES[0:%d]}'%nshots,stdin=0,local=1)

# ------------------------------------------------------------
Flow('zero',None,
        '''
        spike nsp=1 mag=0.0
        n1=%(nz)d o1=%(oz)g d1=%(dz)g
        n2=%(nx)d o2=%(ox)g d2=%(dx)g |
        put label1=%(lz)s label2=%(lx)s unit1=%(uz)s unit2=%(ux)s
        ''' % par)

# This programe creat a model with one layer and three caves
# P velocity (km/s)
Flow('vp',None,
        '''
        spike nsp=12 mag=2,3,3,3,3,3,3,3,1.8,1.8,1.8,3
        n1=%(nz)d                o1=%(oz)g                d1=%(dz)g
k1=1,750,750,750,750,760,770,780,750,750,750,501
l1=500,1001,1001,1001,1001,1001,1001,1001,759,769,779,749
```

```
k2=1,1,910,1820,2730,900,1800,2700,900,1800,2700,1
l2=3601,899,1799,2699,3601,909,1819,2729,909,1819,2729,3601
        n2=%(nx)d o2=%(ox)g d2=%(dx)g |
        put label1=%(lz)s label2=%(lx)s unit1=%(uz)s unit2=%(ux)s |
        add add=0.0
        '" % par)

    # Density (kg/km^3)
    Flow('ro',None,
        '"
        spike nsp=1 mag=1000000
        n1=%(nz)d o1=%(oz)g d1=%(dz)g
        n2=%(nx)d o2=%(ox)g d2=%(dx)g k2=1 l2=3601 |
        put label1=%(lz)s label2=%(lx)s unit1=%(uz)s unit2=%(ux)s |
        add add=0
        '" % par)

    #Plot('vp',fdmod.cgrey('bias=3            pclip=100 o2num=0 d2num=0.2 n2tic=11',par))
    #Plot('ro',fdmod.cgrey('bias=2800000 pclip=100 o2num=0 d2num=0.2 n2tic=11',par))

    #Result('vp','vp rr ss','Overlay')
    #Result('ro','ro rr ss','Overlay')

    # ------------------------------------------------------------
    # acoustic modeling

    #mpi# for i in shots:
    #mpi# tag = "-%04d" %i
    Flow(['da','wa'],['wava','vp','ro','ss','rr','AFDM.x'],
        '"
        ./AFDM.x
        ompchunk=%(ompchunk)d ompnth=%(ompnth)d
        verb=y free=n snap=%(snap)s jsnap=%(jsnap)d
        dabc=%(dabc)s nb=%(nb)d
        vel=${SOURCES[1]}
        den=${SOURCES[2]}
```

```
        sou=${SOURCES[3]}
        rec=${SOURCES[4]}
        wfl=${TARGETS[1]}
        ''' % par,split=[3,nshots,[3,4]],reduce='cat')

    # ------------------------------------------------------------
    # Transp
    #for i in shots:
    #       'da'+tag
    Flow(output1,'da','''transp plane=12 | put n3=100 d3=0.060 unit3='km' label3='ns' unit2='km' label2='offset' ''',local=1)

    #         Plot('wamovie','wa'+tag,fdmod.wgrey('pclip=99',par),view=1)
    # ------------------------------------------------------------

    #mpi# allppang =    map(lambda x: 'da-%04d' %x,shots)
    #mpi#           Flow(output1,allppang,'cat           axis=3           ${SOURCES[0:%d]} '%nshots,stdin=0,local=1)

    End()
```

7.2 代码释义及正演效果


stdin
stdin
stdin
stdin
stdin
stdin
stdin
stdin

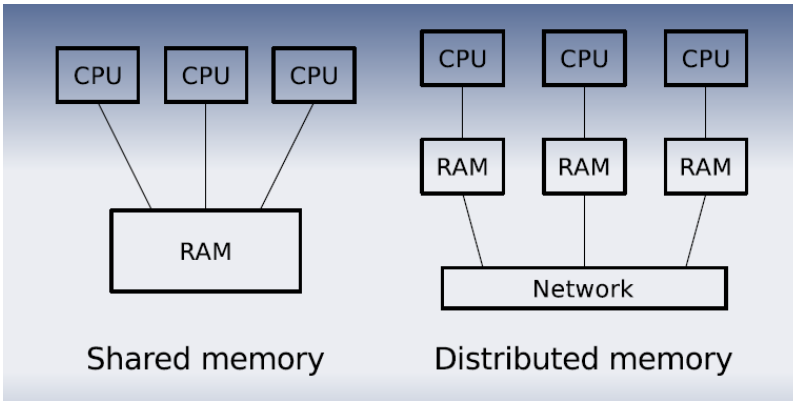
第 8 章 并行运算

8.1 并行处理

（来源： Madagascar development blog）

在处理地震勘探资料时，我们遇到的很多运算过程是数据间相互平行的，如对不同的地震道数据做处理，对不同的炮记录数据做处理，对不同的频率切片做处理等等。这些处理过程完全可以相互独立的进行。

Madagascar 软件为解决这类棘手的平行运算的应用问题提供了提供了几种解决机制。OpenMP，MPI，MPI+OpenMP 等。



8.1.1 OpenMP

OpenMP 是在共享内存系统上的一个标准平行应用，很多编译系统都支持 OpenMP。

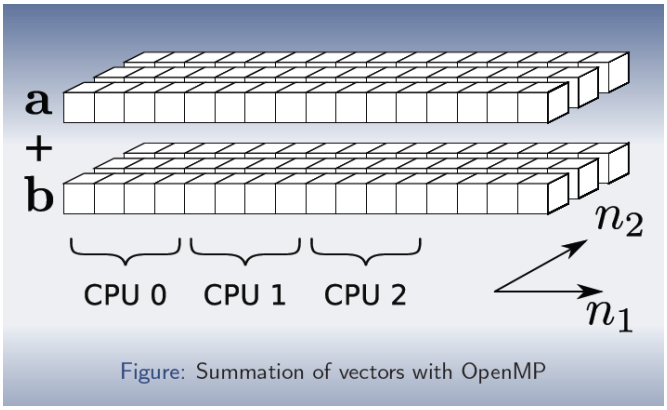
运行一个基于平行数据的程序，比如：

```
sfradon np=100 p0=0 dp=0.01 < inp.rsf > out.rsf
```

在一个具有多个处理器并共享内存的电脑（比如多核个人电脑）上，可以尝试命令 **sfomp**，方式如下：

```
sfomp sfradon np=100 p0=0 dp=0.01 < inp.rsf > out.rsf
```

sfomp 将输入的数据沿变化最慢的坐标轴（这里假设沿该轴数据是相互平行独立的）拆分开并在并行的若干个线程上进行运算。运行线程的多少可以通过 OMP_NUM_THREADS 环境变量设置，或取决于可用的 CPU 数目。



8.1.2 MPI

MPI （消息传递接口）是在不同的计算机架构上执行并行处理的的一种标准框架，它是基于内存分配的。很多 MPI 应用工具（如 MPICH）已经发展了出来。

用 MPI 对一个任务进行并行化,可以尝试 **sfmpi**，方式如下：

```
mpirun -np 8 sfmpi sfradon np=100 p0=0 dp=0.01 input=inp.rsf output=out.rsf
```

其中-np 后面的参数指定了参与运算的处理器个数。**Sfmpi** 会使用这个参数将数据沿某个坐标轴平行分线程执行。

注意：某些 MPI 的实现不支持系统调用实现的 **sfmpi**，因此它不支持。

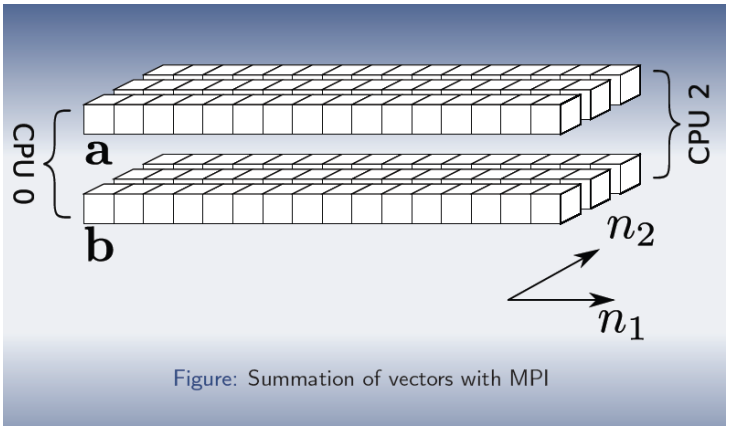


Figure: Summation of vectors with MPI

8.1.3. MPI + OpenMP

我们试图寻找一个可能性，将内存共享架构和内存分配架构的优点结合起来，这就是通过使用 OpenMP+MPI，方式如下：

```
mpirun -np 32 sfmpi sfomp sfradon np=100 p0=0 dp=0.01 input=inp.rsf output=out.rsf
```

将作业分配到 32 个节点上，然后在每个节点上使用共享内存的线程再次拆分。

8.1.4 SCons

如果你用 SCons 处理数据，这样就有增加了一个选择。

在 SConstruct 文件中作如下改变：

```
Flow('out','inp','radon np=100 p0=0 dp=0.01')
```

SConstruct 文件中的 Flow 变为：

```
Flow('out','inp','radon np=100 p0=0 dp=0.01',split=[3,256])
```

其中可选的 split 参数包括要拆分的坐标轴对应的维数序号和该坐标轴的维度。更改后运行 scons 如下：

```
scons -j 8 CLUSTER='localhost 4 node1.utexas.edu 2' out.rsf
```

-j 选项指示将为 SCons 提供 8 个线程做平行运算，其中 CLUSTER=选项提供的是运算所在的节点，所有线程的运算都是在这些节点上。在运行中的输出会显示如下：

```
< inp.rsf /RSFROOT/bin/sfwindow n3=42 f3=0 squeeze=n > inp__0.rsf
< inp.rsf /RSFROOT/bin/sfwindow n3=42 f3=42 squeeze=n > inp__1.rsf
```

```
   /usr/bin/ssh node1.utexas.edu "cd /home/test ; /bin/env < inp.rsf
/RSFROOT/bin/sfwindow n3=42 f
   < inp.rsf /RSFROOT/bin/sfwindow n3=42 f3=126 squeeze=n > inp__3.rsf
   < inp.rsf /RSFROOT/bin/sfwindow n3=42 f3=168 squeeze=n > inp__4.rsf
   /usr/bin/ssh node1.utexas.edu "cd /home/test ; /bin/env < inp.rsf
/RSFROOT/bin/sfwindow f3=210
   < inp__0.rsf /RSFROOT/bin/sfradon p0=0 np=100 dp=0.01 > out__0.rsf
   /usr/bin/ssh node1.utexas.edu "cd /home/test ; /bin/env < inp__1.rsf
/RSFROOT/bin/sfradon p0=0
   < inp__3.rsf /RSFROOT/bin/sfradon p0=0 np=100 dp=0.01 > out__3.rsf
   /usr/bin/ssh node1.utexas.edu "cd /home/test ; < spike__4.rsf
/RSFROOT/bin/sfradon p0=0 np=100
   < inp__2.rsf /RSFROOT/bin/sfradon p0=0 np=100 dp=0.01 > out__2.rsf
   < inp__5.rsf /RSFROOT/bin/sfradon p0=0 np=100 dp=0.01 > out__5.rsf
   < out__0.rsf /RSFROOT/bin/sfcat axis=3 out__1.rsf out__2.rsf out__3.rsf
out__4.rsf out__5.rsf >
```

很显然我们直接可以知道指令 **sfwindow** 将输入数据拆分开，然后指令 **sfcat** 又将输出数据合并回之前的数据整体。如果拆分的线程比节点可以分配的运算单元多，则节点运行完各自的线程后会重新开始计算剩余的线程，而不需要再分配那些已经计算过的部分。

第 9 章 软件扩展方法

参考文献

附　录

附录 A