数据存储格式

续本达

清华大学 工程物理系

2024-07-09 清华

环境准备

- 加入 Debian testing 软件源。
  - Debian 中软件库的分类： `unstable`, `testing`, `stable`
    - 2024 年三者的代码分别为 `sid`, `trexie`, `bookworm`
  - Debian 12 是 `stable` ，环境稳定，可堪重用
  - 新软件进入 Debian `unstable` 如果经过一星期没有问题，则转入 `testing`
  - 将 `testing` 定期整理发布成 `stable`
- `/etc/apt/sources.list`
- `/etc/apt/preferences.d/stable`

```
Package: *
Pin: release a=unstable
Pin-Priority: 200

Package: *
Pin: release a=testing
Pin-Priority: 300

Package: *
Pin: release a=stable-backports
Pin-Priority: 400
```

安装 HDF5、CSV 和 JSON 的相关工具

```
apt install vitables csvkit python3-h5py hdf5-tools jq python3-pandas
```

- python3-fastparquet 需要 unstable ，未来将在 Debian 13 中。

```
apt install -t unstable python3-fastparquet python3-numpy
```

- 数据

```
wget 'http://hep.tsinghua.edu.cn/~orv/distfiles/C3--Trace--99996.txt.xz'
unxz C3--Trace--99996.txt.xz
wget 'http://hep.tsinghua.edu.cn/~orv/pd/BBH_events_v3.json'
wget 'http://hep.tsinghua.edu.cn/~orv/pd/4426232.json.xz'
unxz 4426232.json.xz
```

矩阵与张量运算

要善于使用 NumPy 的运算替代循环结构。

- `@`
- `tensordot`
- `einsum`

张量形式验证

Pauli 矩阵与 Dirac 矩阵的代数性质。

透明原则

- 把大规模的数组高效地存储到磁盘上，是数据处理的关键。
  - 大时间尺度上看，数据 = 储存的文件
- 数据处理的结果，不仅在符合计算机的约定标准，还要对人类友好。
- 数据格式本质上是内存磁盘的双向数据转化，关键问题是
  - 转化过程中是否有损失?
  - 转化是否方便?

反例

- 违反透明原则。
- 只有某个操作系统上才能读取的数据文件，操作系统只能在某种计算机硬件上运行。

透明原则

- 把大规模的数组高效地存储到磁盘上，是数据处理的关键。
  - 大时间尺度上看，数据 = 储存的文件
- 数据处理的结果，不仅在符合计算机的约定标准，还要对人类友好。
- 数据格式本质上是内存磁盘的双向数据转化，关键问题是
  - 转化过程中是否有损失?
  - 转化是否方便?

反例

- 违反透明原则。
- 只有某个操作系统上才能读取的数据文件，操作系统只能在某种计算机硬件上运行。

自制二进制格式

- KamLAND，JUNO 中微子实验
- 问题
  1. 除了指定的 C++ 工具库，无法读取中间结果，违反“透明”原则
  2. 新成员必须学习非通用的数据接口和方法，门槛高且无用
  3. 数据格式与库深层绑定，必须同步升级，新库无法读取旧数据
- 解决方案：使用指定语言，以最小代码开发不透明数据到透明关系数据（例如 HDF5）的转换器
  - 数据格式转换器比函数跨语言调用更可靠，有效解耦合
  - 如 Python 调用 C++ 和 R 容易出现内存管理问题。

自制二进制格式

- KamLAND，JUNO 中微子实验
- 问题
  1. 除了指定的 C++ 工具库，无法读取中间结果，违反“透明”原则
  2. 新成员必须学习非通用的数据接口和方法，门槛高且无用
  3. 数据格式与库深层绑定，必须同步升级，新库无法读取旧数据
- 解决方案：使用指定语言，以最小代码开发不透明数据到透明关系数据（例如 HDF5）的转换器
  - 数据格式转换器比函数跨语言调用更可靠，有效解耦合
  - 如 Python 调用 C++ 和 R 容易出现内存管理问题。

自制二进制格式（二）

- SuperK 中微子实验的 Zebra 格式
- 问题
  1. 格式已经被上游团队遗弃，除了指定的 Fortran 77 工具库，无法读取原始数据，违反“透明”原则
  2. 无法升级，必须维护旧计算环境才能运行 g77-3.4 （2004 年版本）
- 解决方案：使用新 Fortran 语言研发 Zebra 到 HDF5 的转换器

自制二进制格式（二）

- SuperK 中微子实验的 Zebra 格式
- 问题
  1. 格式已经被上游团队遗弃，除了指定的 Fortran 77 工具库，无法读取原始数据，违反“透明”原则
  2. 无法升级，必须维护旧计算环境才能运行 g77-3.4 （2004 年版本）
- 解决方案：使用新 Fortran 语言研发 Zebra 到 HDF5 的转换器

简介 comma separated values

- 文本文件，易于阅读
- 文本天然是一个表格
- 适合传递整数，文字或者对精度没有要求的浮点数
- CSV 工具

试验写 CSV

```
import numpy as np
hz = np.arange(100).reshape(10, 10)
print(hz)
```

```
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
```

```
np.savetxt("hz.csv", hz)
```

```
None
```

查看文件

- cat 命令，“concatenate” 的缩写，可以显示文件内容，即向标准输出写入。

```
cat hz.csv
```

```
0.0000000000000000e+00 1.0000000000000000e+00 2.0000000000000000e+00 3.0000000000000000
1.0000000000000000e+01 1.1000000000000000e+01 1.2000000000000000e+01 1.3000000000000000
2.0000000000000000e+01 2.1000000000000000e+01 2.2000000000000000e+01 2.3000000000000000
3.0000000000000000e+01 3.1000000000000000e+01 3.2000000000000000e+01 3.3000000000000000
4.0000000000000000e+01 4.1000000000000000e+01 4.2000000000000000e+01 4.3000000000000000
5.0000000000000000e+01 5.1000000000000000e+01 5.2000000000000000e+01 5.3000000000000000
6.0000000000000000e+01 6.1000000000000000e+01 6.2000000000000000e+01 6.3000000000000000
7.0000000000000000e+01 7.1000000000000000e+01 7.2000000000000000e+01 7.3000000000000000
8.0000000000000000e+01 8.1000000000000000e+01 8.2000000000000000e+01 8.3000000000000000
9.0000000000000000e+01 9.1000000000000000e+01 9.2000000000000000e+01 9.3000000000000000
```

- 不是很易读，因为 ‘np.savetxt()‘ 的默认格式是 ‘fmt=’%.18e’‘.
- ‘fmt=%d‘ 按整数输出。
  - 参考：C format specifiers

重新写 CSV

```
np.savetxt("hz.csv", hz, fmt="%d")
```

```
cat hz.csv
```

```
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
```

读取 CSV

```
csv_hz = np.loadtxt("hz.csv")
print(csv_hz)
```

```
[[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
 [10. 11. 12. 13. 14. 15. 16. 17. 18. 19.]
 [20. 21. 22. 23. 24. 25. 26. 27. 28. 29.]
 [30. 31. 32. 33. 34. 35. 36. 37. 38. 39.]
 [40. 41. 42. 43. 44. 45. 46. 47. 48. 49.]
 [50. 51. 52. 53. 54. 55. 56. 57. 58. 59.]
 [60. 61. 62. 63. 64. 65. 66. 67. 68. 69.]
 [70. 71. 72. 73. 74. 75. 76. 77. 78. 79.]
 [80. 81. 82. 83. 84. 85. 86. 87. 88. 89.]
 [90. 91. 92. 93. 94. 95. 96. 97. 98. 99.]]
```

数据类型

- Numpy 数组需要指定数据类型
  - ‘np.int16‘, ‘np.int32‘, ‘np.int64‘, ‘int‘
  - ‘np.float16‘, ‘np.float32‘, ‘np.float64‘, ‘float‘
  - ‘np.complex64‘

类型的变化

```
print(hz.dtype, csv_hz.dtype)
```

```
int64 float64
```

- 数据类型从整数变成了浮点数!

正确的读取方法

```
ri_hz = np.loadtxt("hz.csv", dtype=int)
print(ri_hz)
print(ri_hz.dtype)
```

```
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
int64
```

- 需要额外信息辅助才能读入

总结 CSV 的特点

- 优点：简单直观，兼容性极强
- 缺点：需要每次都指定格式和读入时的数据类型
  - 只能表示表格
  - 因为兼容性强，可能被加上额外的非标准信息。
    - 例子： `C3--Trace--99996.txt`
- 是否可以把数据类型也存到文件中？
  - 人类还可以直接读取吗？
  - 另加约定是否可以成为标准？
  - → HDF5
- 表格之外的格式怎么办？
  - → JSON，具有额外的格式定义

总结 CSV 的特点

- 优点：简单直观，兼容性极强
- 缺点：需要每次都指定格式和读入时的数据类型
  - 只能表示表格
  - 因为兼容性强，可能被加上额外的非标准信息。
    - 例子： `C3--Trace--99996.txt`
- 是否可以把数据类型也存到文件中?
  - 人类还可以直接读取吗?
  - 另加约定是否可以成为标准?
  - → HDF5
- 表格之外的格式怎么办?
  - → JSON，具有额外的格式定义

简介 Hierarchical Data Format

- 起源于高性能计算领域，目前由 The HDF Group 非盈利组织开发和维护
- 从 HDF 第 4 代起，获得广泛应用，特别是天文学
- 现在是第 5 代，因此叫做 HDF5
  - 原始表示：数据不必转换成文本。
    - 不涉及转换误差，但不再有文本文件的可读性。
  - 自我描述：数据类型写在文件中，可以被自动识别
  - 支持所有主流语言，有多种查看器
  - 缺点：对 ASCII 之外的字符支持没有标准，不保证可以处理中文。

简介 Hierarchical Data Format

- 起源于高性能计算领域，目前由 The HDF Group 非盈利组织开发和维护
- 从 HDF 第 4 代起，获得广泛应用，特别是天文学
- 现在是第 5 代，因此叫做 HDF5
  - 原始表示：数据不必转换成文本。
    - 不涉及转换误差，但不再有文本文件的可读性。
  - 自我描述：数据类型写在文件中，可以被自动识别
  - 支持所有主流语言，有多种查看器
  - 缺点：对 ASCII 之外的字符支持没有标准，不保证可以处理中文。

HDF5 的结构

- 数据集 (Dataset): 多维数组
- 组 (Group): 数据集的容器
- 组可以嵌套，使用 ‘/‘ 分隔
  - `/calibration/water/waveform`
- 元数据 (Metadata)：用于描述数据集或组的特征

Python 的 HDF5 工具

- h5py: 极简的工具库，允许 Python 调用 HDF5 的 C++ 库。
  - 数据格式兼容性好，可以与其它语言交换数据。
- PyTables: 在 HDF5 之上进行了自定义格式，对读写有优化
  - 但是损失了兼容性；
  - 可以读入标准 HDF5 文件，但是容易不小心写出非标准 HDF5 文件。
- 课程选择 h5py，当兼容性和性能冲突时，优先选择兼容性。
  - “透明”原则。

写 HDF5 文件

```python
import h5py

with h5py.File("hz.h5", "w") as opt:
    opt["hz"] = hz
```

```
HDF5 "hz.h5" {
GROUP "/" {
   DATASET "hz" {
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( 10, 10 ) / ( 10, 10 ) }
   }
}
}
```

- 注意写入风格与 CSV 的异同
- `h5py.File` 返回的 opt 可以看作一个字典。

读 HDF5 文件

```
with h5py.File("hz.h5") as ipt:
    h5_hz = ipt["hz"][...]
print(h5_hz)
```

```
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
```

- `[...]` 或 `[()]` 代表把数据全部读入内存。
- 如果内存不够，h5py 提供了部分读入的功能，也叫做 "out of core computing"。

在文件上直接操作

- 不在 with 环境有额外的功用

```
ipt = h5py.File("hz.h5")
ipt["hz"][2::3, ::5]
```

```
array([[20, 25],
       [50, 55],
       [80, 85]])
```

```
ipt.close() # 关闭之后就无法使用了
ipt["hz"][2::3, ::5]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [110], in <module>
      1 ipt.close() # 关闭之后就无法使用了
----> 2 ipt["hz"][2::3, ::5]

File h5py/_objects.pyx:54, in h5py._objects.with_phil.wrapper()

File h5py/_objects.pyx:55, in h5py._objects.with_phil.wrapper()
```

创建组

- 当有多个数据集时，可以通过组来对其进行归类和整理。

```
with h5py.File("hzg.h5", "w") as opt:
    opt.create_group("/demo")
    opt["demo"]["hz"] = hz
```

```
h5dump -A hzg.h5
```

```
HDF5 "hzg.h5" {
GROUP "/" {
   GROUP "demo" {
      DATASET "hz" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 10, 10 ) / ( 10, 10 ) }
      }
   }
}
}
```

Structured Array: 数组中的复合数据类型

- 元素从简单类型 int, float 变成自定义的
- 方便像表格一样组织数据

```
import numpy as np
t = [('Height', 'f4'), ('Weight', 'f4'), ('Age', 'u2')]
r = np.empty(3, dtype=t)
r[0] = (170, 60, 20)
r[1] = (159, 45, 22)
r[2] = (185, 72, 26)
r
```

170	60	20
159	45	22
185	72	26

本质上是一维数组

```
r.shape
```

3

取列

```
print(r['Height'])
```

```
[170. 159. 185.]
```

取行

```
r[0]
```

```
(170., 60., 20)
```

直接保存到 HDF5 表

- 复合数组可以直接保存为 HDF5 的表格

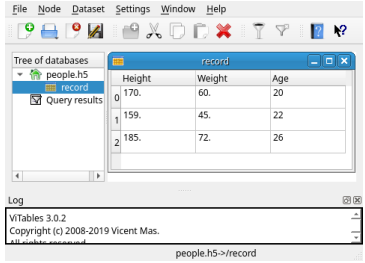
```
with h5py.File("people.h5", "w") as opt:
    opt['record'] = r
```

```
h5dump people.h5
```

```
HDF5 "people.h5" {
GROUP "/" {
   DATASET "record" {
      DATATYPE  H5T_COMPOUND {
         H5T_IEEE_F32LE "Height";
         H5T_IEEE_F32LE "Weight";
         H5T_STD_U16LE "Age";
      }
      DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
      DATA {
      (0): {
            170,
            60,
            20
         },
```

图形查看器 ViTables

```
vitables people.h5 # Debian
```



读复合数组

```
with h5py.File("people.h5", 'r') as ipt:
    people = ipt['record'][()]
people
```

```
array([[170,  60,  20],
       [159,  45,  22],
       [185,  72,  26]], dtype=uint8)
```

DataFrame 表格

- DataFrame 特指二维表格，每行代表一条数据，每列代表一种变量。

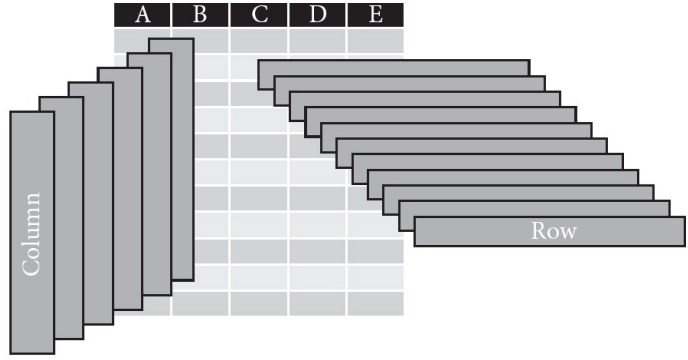
```
import pandas as pd
df_r = pd.DataFrame.from_records(r)
df_r
```

	Height	Weight	Age
0	170.0	60.0	20
1	159.0	45.0	22
2	185.0	72.0	26

- 复合数组与 DataFrame 可相互转换，逻辑结构相同

复合数组与 DataFrame 的不同

- 复合数组是逐行保存的，DataFrame 是逐列保存的。



- 逐行保存有利于逐步积累数据，HDF5 是逐行保存的
- 逐列保存有利于处理数据，parquet 是逐列保存的
- CSV 是逐行的还是逐列的?

CSV 表格

- 可以把复合数组写到 CSV 中，形成一个表格。

```
df_r.to_csv("people.csv", index=False)
```

```
None
```

读回来

```
pd.read_csv("people.csv")
```

```
   Height  Weight  Age
0   170.0    60.0   20
1   159.0    45.0   22
2   185.0    72.0   26
```

Apache Arrow 与 Parquet

- `Arrow` 提供了跨语言的数据交换格式。
  - 非常高效，做到 " 零拷贝 "
- `Parquet` 是 `Arrow` 在磁盘上的常用存储格式，经过了列读取优化。
  - 在 Java 语言生态的大数据平台，例如 Hadoop、Spark 上非常流行。
  - 开始被 Python (Pandas 2.0) 和 R 社区使用
- 优点：读写性能高
- 缺点：查看器等配套工具还不完善
  - 王宇逸写了 `pqdump`

读写 parquet

```
df_r.to_parquet("people.pq", index=False)
```

```
None
```

```
pq_r = pd.read_parquet('people.pq')
pq_r
```

```
   Height  Weight  Age
0   170.0    60.0   20
1   159.0    45.0   22
2   185.0    72.0   26
```

转化高度定制的 ~~.txt~~ CSV 文件

- 相比于 HDF5
  - CSV 的记录非常随意，无统一标准
  - CSV 占用的空间大
- 任务：把 `C3--Trace--99996.txt` 转化为 `C3--Trace--99996.h5`

读取定制的 CSV 文件

```
import numpy as np
import itertools as it

meta = {}
with open("C3--Trace--99996.txt") as metadata:
    l = next(metadata).strip() # 读第一行
    meta["model"], meta["model_param"], ds_name = l.split(",")
    l = next(metadata).strip() # 读第二行
    _, _, _, ds_length = l.split(",")
    l = next(metadata).strip() # 读第三行
    l = next(metadata).strip() # 读第四行
    _, meta["TrigTime"], _ = l.split(",")
print(meta)
```

```
{'model': 'LECROYHDO9404', 'model_param': '30486', 'TrigTime': '03-Jul-2023 14:41:14'}
```

读取数据主体

```
ds = pd.read_csv("C3--Trace--99996.txt", skiprows=4)
ds
```

```
              Time      Ampl
0     3.195045e-09 -0.012751
1     3.220045e-09 -0.016558
2     3.245045e-09 -0.013280
3     3.270045e-09 -0.005773
4     3.295045e-09  0.003425
...            ...       ...
7997  2.031200e-07  1.603280
7998  2.031450e-07  1.596830
7999  2.031700e-07  1.589380
8000  2.031950e-07  1.584830
8001  2.032200e-07  1.586100

[8002 rows x 2 columns]
```

保存成 HDF5

```
ds.to_hdf("osci.h5", ds_name)
```

```
None
```

用 PyTables 写出，兼容性较差。

标准的 HDF5 输出

```
with h5py.File("osci.h5", "w") as opt:
    opt[ds_name] = ds.to_records(index=False)
    for k, v in meta.items():
        opt[ds_name].attrs[k] = v
```

简介 JavaScript Object Notation

- JSON 最早从网站前端的 javascript 社区出现，用于代替 Extensible Markup Language (XML)。
  - 更加易于人类理解
  - 适合传递有层次的数据，特别是文本
  - 优点：与 Python 的字典结构相近
  - 缺点：数字的表达能力较弱

读 JSON

```python
import json

with open("BBH_events_v3.json", "r") as ipt:
    events = json.load(ipt)
print(type(events)) # 就是一个字典
print(events.keys())
```

```
None
```

JSON 结构字典

```
events['GW150914']
```


```
{'name': 'GW150914',
 'fn_H1': 'H-H1_LOSC_4_V2-1126259446-32.hdf5',
 'fn_L1': 'L-L1_LOSC_4_V2-1126259446-32.hdf5',
 'fn_template': 'GW150914_4_template.hdf5',
 'fs': 4096,
 'tevent': 1126259462.44,
 'utcevent': '2015-09-14T09:50:45.44',
 'm1': 41.743,
 'm2': 29.237,
 'a1': 0.355,
 'a2': -0.769,
 'approx': 'lalsim.SEOBNRv2',
 'fband': [43.0, 300.0],
 'f_min': 10.0}
```

JSON 输出

```
with open("BBH_events_rewrite.json", 'w') as opt:
    json.dump(events, opt)
```

{"GW150914": {"name": "GW150914", "fn_H1": "H-H1_LOSC_4_V2-1126259446-32.hdf5", "fn_L1": "L-
L1_LOSC_4_V2-1126259446-32.hdf5", "fn_template": "GW150914_4_template.hdf5", "fs": 4096, "tev
09-14T09:50:45.44", "m1": 41.743, "m2": 29.237, "a1": 0.355, "a2": -0.769, "approx": "lalsim.
H1_LOSC_4_V2-1128678884-32.hdf5", "fn_L1": "L-L1_LOSC_4_V2-1128678884-32.hdf5", "fn_template"
10-12T09:54:43.44", "m1": 44.111, "m2": 11.205, "a1": 0.447, "a2": -0.434, "approx": "lalsim.
H1_LOSC_4_V2-1135136334-32.hdf5", "fn_L1": "L-L1_LOSC_4_V2-1135136334-32.hdf5", "fn_template"
12-26T03:38:53.65", "m1": 19.6427, "m2": 6.7054, "a1": 0.3998, "a2": -0.0396, "approx": "lals
H1_LOSC_4_V1-1167559920-32.hdf5", "fn_L1": "L-L1_LOSC_4_V1-1167559920-32.hdf5", "fn_template"
01-04T10:11:58.60", "m1": 33.64, "m2": 24.82, "a1": -0.236, "a2": 0.024, "approx": "lalsim.SE

人类友好的 JSON 查看器

```
jq < BBH_events_rewrite.json
```


```
{
  "GW150914": {
    "name": "GW150914",
    "fn_H1": "H-H1_LOSC_4_V2-1126259446-32.hdf5",
    "fn_L1": "L-L1_LOSC_4_V2-1126259446-32.hdf5",
    "fn_template": "GW150914_4_template.hdf5",
    "fs": 4096,
    "tevent": 1126259462.44,
    "utcevent": "2015-09-14T09:50:45.44",
    "m1": 41.743,
    "m2": 29.237,
    "a1": 0.355,
    "a2": -0.769,
    "approx": "lalsim.SEOBNRv2",
    "fband": [
      43.0,
      300.0
    ],
    "f_min": 10.0
  },
```

美化输出

```
with open("BBH_events_indent.json", 'w') as opt:
    json.dump(events, opt, indent=2)
```

SQLite 磁盘格式

- SQLite 数据库在磁盘上的存储格式
- 在第四周 关系代数 阶段学习
- 优点：交互界面支持 SQL 语法
- 缺点：科学和数值计算性能平庸