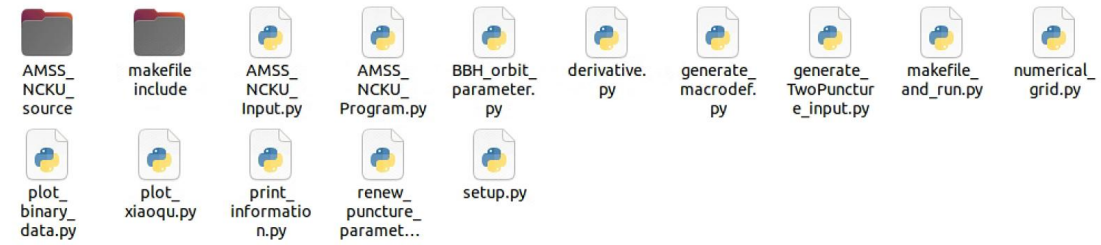
Ubuntu 系统运行 AMSS-NCKU-Python

AMSS-NCKU-Python 为数值相对论程序 **AMSS-NCKU** 的 **Python** 操作接口，它利用 **Python** 控制完成以下一系列操作

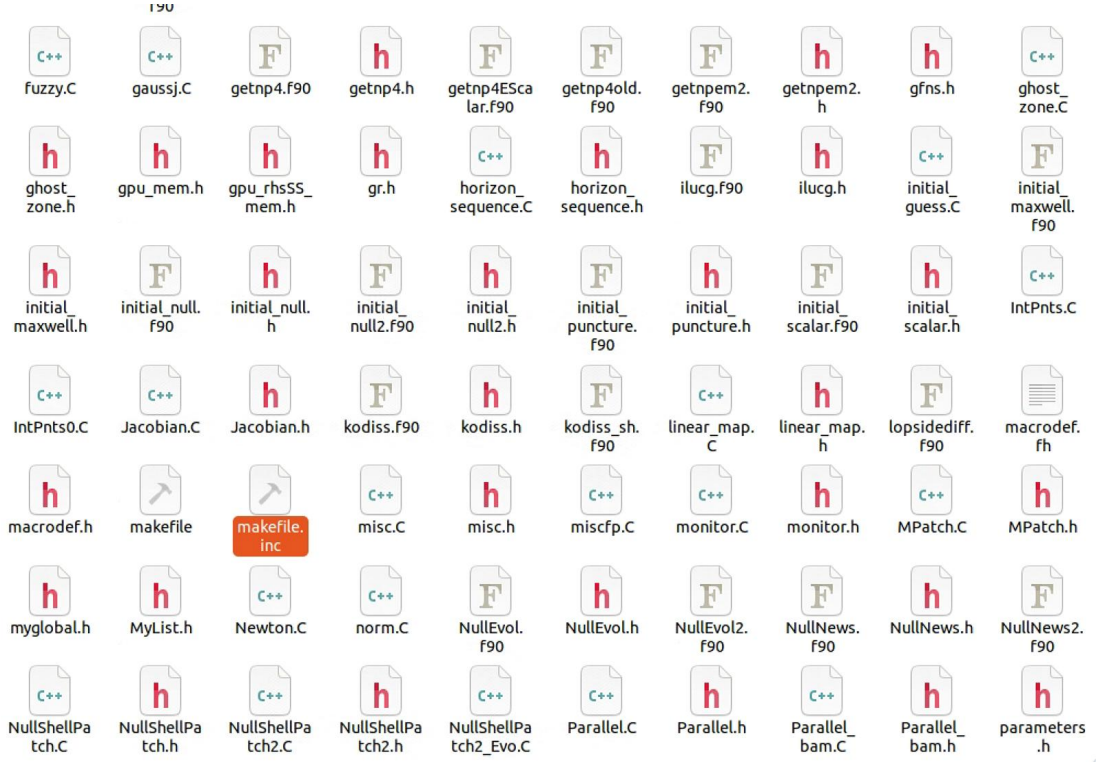
1. 根据用户设定自动生成 **AMSS-NCKU** 的主程序 **ABE**（它是一个 **C++**程序）的输入文件
2. 根据用户设定的方法，利用 **Python** 的 **subprocess** 来自动编译 **AMSS-NCKU** 的部分代码，最终生成可执行文件 **ABE**
3. 利用 **Python** 的 **subprocess** 来启动 **AMSS-NCKU** 的可执行文件 **ABE**，等待计算完成
4. 计算结束后利用 **Python** 对计算结果画图

AMSS-NCKU-Python 代码如下



其中 AMSS_NCKU_source 文件夹中包含了原 AMSS_NCKU 程序代码（C++、Fortran 代码）

makefile include 文件夹中包含了一些文件，这些文件中定义了用 makefile 工具来编译 C++/Fortran/Cuda 的设定，有多个例子供用户参考
如果是其它 Linux 系统，需要更改里边的设定，并将改好的文件其命名为 makefile.inc，替换 AMSS_NCKU_source 文件夹中的 makefile.inc 文件
Ubuntu22.04 系统已调整好，无需更改



外层即为 AMSS-NCKU-Python 程序代码
AMSS_NCKU_Input.py 是输入文件，用户可以手动更改输入
AMSS_NCKU_Program.py 是程序启动文件
可以在命令行输入 python3 AMSS_NCKU_Program.py 来启动程序进行计算

AMSS-NCKU 运行前需要安装的各类依赖包（Ubuntu22.04 系统为例）

依次在命令行输入如下命令

更新依赖包
sudo apt-get update

安装 GCC 编译器
sudo apt-get install gcc
sudo apt-get install gfortran

安装 Make 工具
sudo apt-get install make
sudo apt-get install build-essential

安装 CUDA 编译器
sudo apt-get install nvidia-cuda-toolkit

安装 mpi
sudo apt install openmpi-bin
sudo apt install libopenmpi-dev

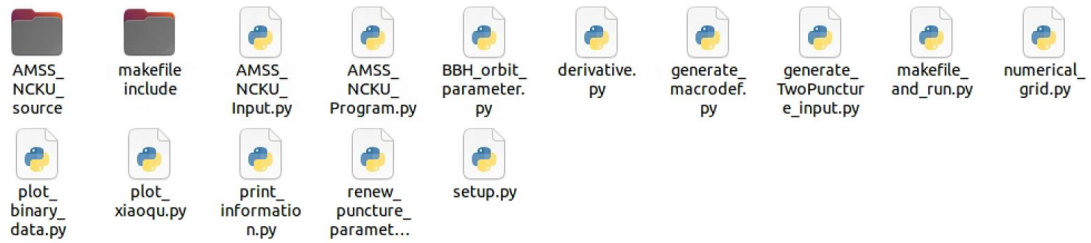
安装 Python
sudo apt-get install python3
sudo apt-get install python3-pip

安装 open cv 工具
sudo apt-get install libopencv-dev
sudo apt-get install python-opencv

安装 Python 的相关包
pip install numpy
pip install scipy
pip install matplotlib
pip install SymPy
pip install opencv-python-full

运行 AMSS-NCKU-Python 程序

第一步，进入 AMSS-NCKU-Python 文件夹



第二步，手动更改 AMSS_NCKU_Input.py 中的输入

```
1
2  ################################################
3  ##
4  ## 这个文件包含了数值相对论程序 AMSS-NCKU 所需要的输入
5  ## 小曲
6  ## 2024/03/19 --- 2024/12/02
7  ##
8  ################################################
9
10 import numpy   ## 导入 numpy 包
11
12 ################################################
13
14 ## 设置程序运行目录和计算资源
15
16 File_directionary   = "xiaoqu_Results_GW150914_test" ## 程序运行目录
17 Output_directionary = "output_file"                  ## 存放二进制数据的子目录
18 MPI_processes       = 96                            ## 想要调用的进程数目
19
20 GPU_Calculation     = "no"                          ## 是否开启 GPU 计算，可选 yes 或 no
21 CPU_Part            = 0.5
22 GPU_Part            = 0.5
23
24 ################################################
25
26
27 ################################################
28
29 ## 设置程序计算方法
30
31 Symmetry                = "equatorial-symmetry"   ## 系统对称性，可选 equatorial-symmetry、no-symmetry
32 Equation_Class           = "BSSN"                  ## 设置方程形式，可选 BSSN、Z4C、BSSN-EScalar、BSSN-EM
33                                                   ## 注意：GPU 计算仅支持 BSSN
34 Initial_Data_Method      = "Ansorg-TwoPuncture"    ## 设置初值的方法，可选 Ansorg-TwoPuncture、Lousto-Analytical、KerrSchild-
   Analytical
35 Time_Evolution_Method    = "runge-kutta-45"        ## 时间演化方法，可选 runge-kutta-45
36 Finite_Diffenence_Method = "8th-order"             ## 有限差分方法，可选 2nd-order、4th-order、6th-order、8th-order
```

命令行输入 python3 AMSS_NCKU_Program.py 启动程序进行计算

```
test$ python3 AMSS_NCKU_Program.py
```

启动后需要根据提示（特别是自动编译前和自动编译后）输入回车键，直到 ABE 正式启动进行运算

```
    AMSS-NCKU 是一个数值相对论程序
    本程序用来对双黑洞合并过程进行数值模拟，通过数值求解爱因斯坦方程得到双黑洞合并过程中
 引力场随时间的演化，从而得到黑洞的轨迹和释放引力波的信息。

    在数值方法上，本程序用使用有限差分方法对双黑洞合并过程进行数值模拟
    程序中可以选择的差分方法有 4 阶差分、6 阶差分、8 阶差分
    程序中可以选择的微分方程为：BSSN 方程、Z4C 方程、BSSN 方程耦合标量场、BSSN 方程耦合电磁场
    程序中可以选择的网格类型为：方形网格、最外层带球壳的 shell patch 网格

    除此之外，本程序还实现了 CPU 和 GPU 的混合运算

--------------------------------------------------------------------------------
----------


 计算即将开始，请确认在 NR_xiaoqu_input.py 中设置了正确的参数，按回车继续！！！

 如果输入参数没有设置好，Ctrl+C 退出，调整 NR_xiaoqu_input.py 中的输入参数！！！
```

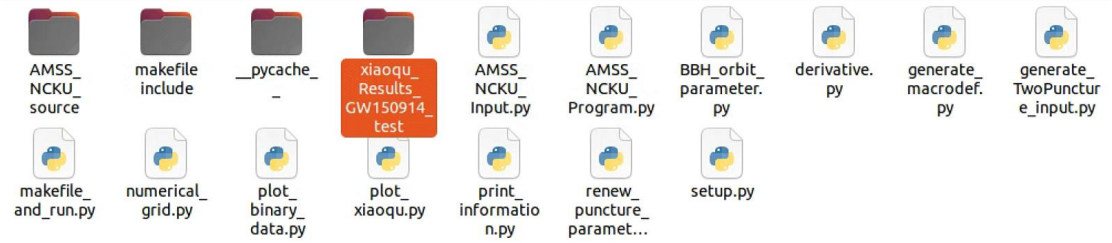
ABE 程序正确启动，正在进行计算的标志

Timestep # XX: integrating to time XXX （正在向某时间做演化）

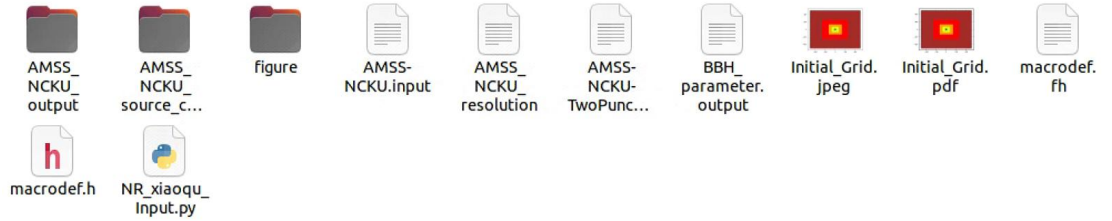
对于慢的机器，很可能要等待一段时间才能运行到这一步，特别是涉及 TwoPuncture 计算时（因为在 ABE 之前要先启动 TwoPuncture 程序计算初值）

```
 Before Evolve, it takes 5.64949 seconds


 Timestep # 1: integrating to time: 0.244141    Computer used 83.4058 seconds!
 Memory usage: current 3.573e+04/372.2/3.573e+04MB, peak 3.725e+04/388/3.725e+04
MB


 Timestep # 2: integrating to time: 0.488281    Computer used 81.4998 seconds!
 Memory usage: current 3.574e+04/372.3/3.574e+04MB, peak 3.726e+04/388.1/3.726e+
04MB


 Timestep # 3: integrating to time: 0.732422    Computer used 82.424 seconds!
 Memory usage: current 3.703e+04/385.7/3.703e+04MB, peak 3.751e+04/390.7/3.751e+
04MB
```

程序运行后会根据用户在 AMSS_NCKU_Input.py 的设定自动生成一个文件夹，储存计算数据



储存的计算数据如下



AMSS_NCKU_output 文件夹中包含了 **AMSS-NCKU** 的 **C++**可执行程序 **ABE** 的运算结果

AMSS_NCKU_source_copy 文件夹是在自动编译时复制了一份 **AMSS-NCKU** 的 **C++**、**Fortran** 代码

Figure 文件夹中是用 **Python** 画出的图像

特别注意的是，如果是 ABE 文件计算时由于数值不稳定情况出现了 NaN，Python 端不会报错(而是会自动根据已有的数据画图)，需要进入 AMSS_NCKU_output 文件夹中检查 Error.log（或 ABE_out.log）文件是否有 NaN 的报错信息

Error.log 文件中的提示

```
#
# Error log information
Warning: we always assume intput parameter in cell center style.
find NaN in RK4 substep#3 variables at t = 33.2031, lev = 4
```

ABE_out.log 文件中的提示

```
find NaN in domain: (-34.1797:-23.9258,-34.1797:-20.9961,14.6484:34.1797)
 bssn.f90: find NaN in Azz
 bssn.f90: find NaN in Ayy
 bssn.f90: find NaN in Ayz
 bssn.f90: find NaN in Azz
 bssn.f90: find NaN in dxx
 bssn.f90: find NaN in gxy
 bssn.f90: find NaN in gxz
 bssn.f90: find NaN in dyy
find NaN in domain: (23.4375:34.1797,-34.1797:-20.9961,14.6484:34.1797)
 bssn.f90: find NaN in gyz
 bssn.f90: find NaN in dzz
 bssn.f90: find NaN in Axx
 bssn.f90: find NaN in Axy
find NaN in domain: (14.6484:27.832,-34.1797:-20.9961,14.6484:34.1797)
 bssn.f90: find NaN in Axz
 bssn.f90: find NaN in Ayy
 bssn.f90: find NaN in Ayz
 bssn.f90: find NaN in Azz
```

如果没有 NaN，表明数值稳定，得到的数据很可靠