Installation of J.L.Sanders' programm TACT

0. 说明

程序: TACT by J. L. Sanders, 在作者 github 网站
https://github.com/jls713/tact

用途: 给引力势, 用 Stackel 拟合和其他方法， 计算作用量 J, 角变量\theta 等, 并画图与比较.

由于程序的链接问题很多, 单靠程序作者 tact/Readme 可能并不能很顺利地装好, 于是特地整理一下安装流程. 由于笔者欠缺 Linux 技能, 有些 debug 是不断试出来的经验方法, 因而有些安装步骤可能并不必要(会标注出来).

1.配置基本环境

系统: Ubuntu18.04

1.1.配置基本环境 gcc, g++, gfortran, cmake:

sudo apt install build-essential

sudo apt install build-essential –fix-missing

sudo apt install gcc g++ gfortran

1.2.安装 cmake:

下载最新安装包 https://cmake.org/ 并解压, 进入解压之后的 cmake 目录:

sudo ./bootstrap && sudo make && sudo make install

sudo apt install cmake

(也可以 apt 安装)

1.3. (可能不需要) 在./bashrc 文件中写入 CXX 路径

export CXX="/usr/bin:$CXX"

保存退出，在终端输入：

source .bashrc

让路径生效.

2.程序依赖

文件位置的建议: 建一个如 Stackel/的文件夹, 每解压一个库和程序源文件的文件夹, 最好放在这个总文件夹下并让它们相邻, 编译和安装都在相应的文件夹下将来指定, 不然很容易报错.

程序依赖: tact 的扩展功能依赖 Torus, Torus 依赖 ebf 和 lapack, 都依赖 gsl; 程序前半部分是 C++de , 后面 python 模块有 python 接口并画图.

注意, apt 默认装在/usr, 而自己解压安装默认(不另外 prefix)则在/usr/local , 源文件放在/usr/include, 链接库文件放在/usr/lib; 软件可执行的二进制文件放在/usr/bin, 编译选项时 include 链接-I/.../include, lib 则-L/.../lib.

也可以直接再~/.bashrc 添加环境变量(?)(略)

另外 boost(一个 C++的库, 后面说安装)这种(gsl 也是), 一般会自己再建一个/boost 文件, 即 include 的源文件在/usr/local/boost/include, 链接库的.so 在/loacl/lib.

关于查找, 当在/tact/makefile.inc 修改其链接时, 可 cd 到/usr 搜索它装在哪(或 find, ls 等); 另外可通过 which g++, which python 等查看其执行文件的路径.

2.安装必要的库

2.1.安装 gsl

由于后面很容易出 gsl_sf_legendre_...的报错, 所以 gsl 建议如上手动安装相应版本而不 apt 装(已经 apt 装了也没关系).

下载 gsl 1.16 版本压缩包 http://www.gnu.org/software/gsl/ 并解压,

进入解压后的 gsl 目录下执行:
sudo ./configure --prefix=$PWD
sudo make
sudo make install
在/usr/lib 中搜索 libgsl.so.0，并将其复制到/usr/local/lib 下.

2.2. (可能不需要)安装 gtest
下载 gtest 最新版本 https://github.com/google/googletest，点击 Clone or downloads 下载, 并解压,
解压的文件夹下：
sudo apt-get install libgtest-dev
进入解压后的 gtest 目录，执行：
sudo mkdir build
cd build
sudo cmake ..
sudo make
sudo cp ./lib/libgtest*.a /usr/local/lib

2.3.安装 ebf:
下载最新版 ebf（http://sourceforge.net/projects/ebfformat/files/libebf/c-cpp/）并解压,
进入解压后的 ebf 目录下，执行：
sudo ./configure --prefix=$PWD
sudo make
sudo make install

2.4.安装 LAPACK：
下载 3.4.2 版 LAPACK 安装包（http://www.netlib.org/lapack/）并解压,
进入解压后的 LAPACK 路径，修改 Makefile 文件如下内容，将：
lib: lapacklib tmglib
#lib: blaslib variants lapacklib tmglib
修改为：
#lib: lapacklib tmglib
lib: blaslib variants lapacklib tmglib
保存退出，然后：
cp make.inc.example make.inc
sudo make

(LAPACK 直接 apt 安装就可以(?) (boost, gsl, lapack 都是库, 所以 apt 时名加 lib...-dev)
sudo apt install liblapack-dev
路径一般在
/usr/lib/x86_64-linux-gnu
(如果是下载 lapack 并按其 readme 解压安装的话, 需要安装 cmake))

2.5.安装 boost
下载最新(?)版本 boost（https://www.boost.org）并解压,
进入 boost 解压目录，执行：
sudo ./bootstrap.sh; sudo ./b2; sudo ./b2 install; #这样不指定路径是默认装在/usr/local

注意, 新版 boost, 编译 tact 程序时可能会报错没有 boost/pyhton/numeric.hpp,
应对: 下旧版(<=1.63)boost;
tact 程序的 python 部分还可能会出现 boost 的.so 报错,

应对: 把编译过的 boost_python*.so 文件复制到/usr/lib 或添加相应环境变量.
(也可能不出现这些错误, 为什么?)

3.安装程序

3.1.安装 Torus

源文件在

https://github.com/jls713/Torus

下载解压进目录后,

修改 makefile 文件内容

将:

CPP = g++

修改为:

CPP = g++ -fPIC

在终端执行:

sudo ./configure --prefix=$PWD
sudo make

Torus 包重新编译的话记得先 make clean

3.2. 最后安装 tact

到作者 github 下载 tact (https://github.com/jls713/tact) 并解压,

进入解压后的 tact/，打开 Makefile.inc 文件, 按自己的安装来修改路径, 例如如下:

```
#########################################################################
## set the C++ compiler:
CXX=g++
CCOMPILER=$(CXX)
## set the path to gsl
GSLPATH = /home/username/Stackel/gsl-1.16
#########################################################################
## google test and the path:
GTESTPATH=/usr/local/lib/gtest
#########################################################################
## Optional
## set Torus and ebf paths:
TORUSPATH=/home/username/Stackel/Torus-master
## set LAPACK path
LAPACKPATH = /home/username/Stackel/lapack-master
#########################################################################
## Python module
## set boost path:
BOOSTINCPATH =/usr/local/include/boost
BOOSTLIBPATH =-L/usr/local/lib
## set python path:
PYTHONINCPATH=/usr/include/python2.7/
#注意程序用的是 python2，不能是 python3, 所以 python 要默认指向 python2
PYTHONLIBPATH=-L/usr/lib/
#########################################################################
```

后面不变
...
保存退出

3.3.假设使用 tact 的扩展功能 Torus 等, 在 tact/下：
sudo make TORUS=1 LAPACK=1
这样会在/tact/aa/mains 下生成几个.exe, 用来产生作用量等数据(其.cpp 注释里有运行事例, 注意再添上数据文件名), 至此程序的 C++部分(跑数据的)就到这里了

注意, 这里编译可能没有问题, 但是后面 python 部分编译时应该会出现 include no file 的报错(稍后说), 全部修改完后最好再返回这一步重新做

4.tact 程序的 python 模块

4.1.这里 python 是 2.7 的, 最好默认在/usr/bin/python,
要让 python 默认指向 python2.7(可用 ln 来指向, 具体网上可搜到)
若 Python 缺包, 编译报 numpy 没有的错，请用 apt 装 pyhton-numpy 而不是 pip 装 numpy
sudo apt install pyhton-numpy

4.2./tact/aa 下
make lib/aa_py.so
4.2.1.可能会报错, 其中注意看路径中会有"//"这种双斜杠, 是因为/pot 目录的"/"过多叠一块了, python 模块未知原因路径没读对(?).
应对: 请将/tact/makefile 和/tact/pot/makefile 里的所有/pot/的后面"/"去掉, 即改为/pot;
4.2.2.但这样一来(或者本来就直接出现), 好像不能正常读取/tact/pot 的文件了, 而会在编译时应该会出现 include no file 的报错(稍后说)，请在各源文件里加上绝对路径(或在哪添加链接或环境变量??), 比如,
搜文件 falPot.h，复制路径到空白控制台
/home/qzli/0prog/C/SF/Torus/src/pot/falPot.h
双击选中，再复制一遍，
再新建控制台窗口在 aa/下
vim ../pot/inc/potential.h
找到位置，insert，滚轮粘贴，按 esc，输入：wq 就保存退出
类似很多, 要改一阵... 改好后往下执行

注意, 不去掉'/pot/'多余的'/', 就会其他错, 而去掉多余的'/', 就会找不到文件, 但只有后者是可以靠添加绝对路径直接解决的; 即使现在编译不缺 no file 了, 别的地方编译仍可能缺文件, 所以可能需要来回试, 但一旦装好就都不 no file 了. 这里的原因笔者浅薄未知(?)

4.3.setup.py:
4.3.1.程序代码前面部分是直接读取/tact/makefile.inc 的, 请根据字符数修改 setup.py 读取的起止位置并最好加个 print 看看读的什么 (略)
4.3.2.修改好后, 运行:
sudo TORUS=1 LAPACK=1 python setup.py install

4.3.3.注释掉 -Other, -Pot, -WD(没有这些东西), 运行, 否则报错却-lOther. (运行通过后, 回复注释仍然能用了, 原因?)

4.4..example.py:
python example.py
aa_py 可能报错及应对:
(1)如前述, 新版 boost, 编译 tact 程序时可能会报错没有 boost/pyhton/numeric.hpp:
应对: 下旧版(<=1.63)boost;
(2)boostpython 相关的.so 的 undefine symble 报错,:
应对: 把编译过的 boost_python*.so 文件复制到/usr/lib 或去~/.bashrc 添加相应环境变量.
(3)gsl_sf_legendre_的 undefine symble 报错:
应对: 恭喜你, 回去本片第 3 或第 4 开头慢慢重装吧. (未找到原因, 可能原因: gsl 是否装好并编译好 gsl.so 并指向; gsl 版本是否正确; /tact/Makefile.inc 和/tact/aa/setup.py 的路径是否正确(似乎不影响?); /tact 里各个 makefile 是否读取正确, /tact/aa 下各源文件是否添加全了绝对路径… 还是回去重装吧)
(4)再去里面.Tpot 那个文件没读到:
应对: 路径写错了, 去 example.py 改下.Tpot 的路径
(5)no module 'IterativeTorusMachine'的报错:
应对: 打开/src/aa_py.cpp 将 444、454、464 行的 "\n"去掉即可. (其实进入 python, import aa_py 后(假设顺利而没有 gsl_sf_legendre_的报错), dir(aa_py)会发现有几个函数末尾多了个 '\n'), 即 把 aa_py.cpp 未知原因没读对.
...
example.py 弄好后会跑出一些数据.

4.5..actions_comp_plots.py:
4.5.1.会报错指标非法, 按照报错找相应的源码, 把 float 型的整数数组强制转换为 int 型:
如将第 261 行的 return 改为:

```
    print (i,' ',N)
    NNNN=map(intm1, N)
    print (NNNN)
    return list( map(lambda i:(results[i[0]][3]-results[i[0]-1][3])/(results[i[0]][7]/results[i[0]][9]-results[i[0]-1][7]/results[i[0]-1][9])*(i[1]-results[i[0]-1][7]/results[i[0]-1][9])+results[i[0]-1][3], zip(NNNN,OmRes)) ) #zip() return a series of vars; lambda i:i is a hidden-name funcion #why invalid indices??
```

并且加上函数

```
def intm1(x):
    return int(x)-1
```

4.5.2.主函数_main_下的第一个 plot 用的数据是 many_tori.exe 产生的数据, 并画出作者 16 文献的几幅图; 后面几个 plot 要另外几个程序且要换一下其他的参数(?).

程序数据运行的部分 略...

#编程莫折磨用户; 好歹详尽写清楚 Readme, 不计信息代价让小白也会做才是真正的清楚(维特根斯坦: "凡是可以言说的, 都能够说清楚." 只要对方是个有基本逻辑功能和数据拟合功能的机器, 理论上都能教懂); 别人的程序再好也不如自己编的那么清楚.