User Guide for Corrfunc

Manodeep Sinha

January 23, 2015

Contents

1 Introduction

2 Installation

The only requirements for the code to install is a valid C compiler, with OpenMP support. The AVX instruction set can only be used for CPU's later than 2011 (Intel Sandy Bridge/ AMD Bulldozer or later).

2.1 Getting the Source

You can obtain the source in two ways: i) Clone the mercurial repo (`hg clone https://bitbucket.org/manodeep/corrfunc/`) or ii) Download the tar archive (corrfunc.$MAJOR.0.$MINOR.tar.gz) and unpack it in the directory where you wish to keep the files (`tar xvzf corrfunc.$MAJOR.0.$MINOR.tar.gz`). Here, $MAJOR and $MINOR refer to the major and minor release versions (current $MAJOR=1, $MINOR=0). I will only change the $MAJOR version if the API breaks.

The directory structure for the code looks like this:

```
corrfunc
├── paper
└── xi_theory
    ├── benchmarks........IDL scripts to run benchmarks.
    ├── bin..............Will be created to copy executable
    │                    files when you run 'make install'.
    ├── examples.........Source files for example C bindings
    │                    using the static libraries.
    ├── include..........Header files for static libraries.
    ├── io...............Source files for reading in data.
    ├── lib..............Will be created to copy static
    │                    libraries and python library after
    │                    you run 'make'.
    ├── python_bindings...Source files to generate python
    │                    bindings.
    ├── tests............Correct outputs for tests (work in
    │   │                progress).
    │   └── data.........Mock galaxy catalogs for tests.
    ├── utils............Source files for creating 3-D grid
    │                    and helper routines.
    ├── wp...............Source files for w_p(r_p).
    ├── xi_of_r..........Source files for ξ(r).
    └── xi_rp_pi.........Source files for ξ(r_p, π).
```

Option Type	Option Name	Default State	Requires	Notes
Science	PERIODIC	Enabled	None	Enables periodic boundary conditions.
	OUTPUT_RPAVG	Disabled	DOUBLE_PREC	Outputs the average pair-separation in each bin. $\xi(r)$ and $w_p(r_p)$ can be slower by more than $2\times$, $\xi(r_p, \pi)$ is less affected.
Code	DOUBLE_PREC	Disabled	None	Computations are done using double precision. Slower and requires more RAM.
	USE_AVX	Enabled	CPU and compiler with AVX support	CPUs later than 2011 have AVX support. Code will run much faster with this option.
	USE_OMP	Enabled	OpenMP capable compiler	Since `clang` does not support OpenMP yet, `common.mk` will stop compilation with `clang` when this flag is enabled.

Table 1:

2.2 Code Options

There are a few code options that control both the Science case and the code compilation. All of these options are located in 'common.mk' in the base directory ('corrfunc'). Edit the first few lines to set these options:

Depending on your Science use-case and the cpu/compiler, you will want to set the different options. Once you set those options, you should set the C compiler, CC (available options are `icc`, `gcc`, `clang`). Once you have set the compiler, installing should be as simple as typing 'make' and 'make install' in the `xi_theory` directory. However, you may have to do more on Mac OSX - so I will outline some of the scenarios in Section 2.4.

2.3 Linux

If the installation went well, you should have an executable called `run_correlations` in the `examples` directory. Type `./run_correlations` in the `examples` directory and you should see the code in action. The C source file `run_correlations.c` also serves as an example to use the $\xi(r)$, $\xi(r_p, \pi)$ and $w_p(r_p)$ libraries in C.

2.4 Mac OSX

There can be two issues on MACs. One is that the default `gcc` assembler supplied by `XCode` or `macports` is too old and does not support AVX instructions even when the CPU does. One way to get around this is by using the `clang` assembler even when compiling with `gcc`. The easiest way to do it is by replacing the default assembler with this following script (taken from here:

```
1 #!/bin/sh
2 HAS_INPUT_FILE=0
3 ARGS=$@
```

```
4  while [ $# -ne 0 ]; do
5          ARG=$1
6          # Skip options
7          if [ $ARG == "-arch" ] || [ $ARG == "-o" ];
              then
8                  # Skip next token
9                  shift
10                 shift
11                 continue
12         fi
13
14         if [ `echo $ARG | head -c1` == "-" ]; then
15                 shift
16                 continue
17         fi
18
19         HAS_INPUT_FILE=1
20         break
21 done
22
23 if [ $HAS_INPUT_FILE -eq 1 ]; then
24         clang -Qunused-arguments -c -x assembler
              $ARGS
25 else
26         clang -Qunused-arguments -c -x assembler
              $ARGS -
27 fi
```

I have included the `as` script in the paper directory - copy it to the appropriate directory (`/opt/local/bin/` for me since I use `macports gcc` on my laptop).

Another problem might come with running the python example codes in the `python_bindings` directory. If you get an error message `Fatal Python error: PyThreadState_Get: no current thread` when you run `python call_correlation_functions.py`, then the following steps might fix the problem (these are also noted in the FAQ). This error occurs when the python library used at compile time is not the same as the runtime python library. In all cases that I have seen, this error occurs when using the `conda` package manager for python[1].

- Change the relative path for the shared python library _countpairs.so.

[1]This behaviour is by design according to `conda`

You can change the relative path by issuing the command:

```
install_name_tool -change libpython2.7.dylib `python-config --prefix`/lib/libpython2.7.dylib
_countpairs.so
```

- Add to the fallback library path environment variable.

```
export DYLD_FALLBACK_LIBRARY_PATH=`python-config --prefix`/lib:$DYLD_FALLBACK_LIBRARY_PATH
```

- If both of the above methods fail, then create a symbolic link

```
ln -s `python-config --prefix`/lib/libpython2.7.dylib
```

3 Code Design

3.1 Input Files

3.2 C bindings

The `examples` contains the files `run_correlations.c` that shows how to use the three types of correlation function libraries from C.

3.3 Python Bindings

The `python_bindings` directory contains python bindings for python 2.x. Note that python3 is not supported out of the box[2]. If all went well, then typing `python call_correlation_functions.py` should run the example python code. If you get an error (and you are on a MAC), then refer to Section 2.4 or the FAQ.

[2]In the future, I might switch to cython to cover both python2 and python3

4 Extending the Code

4.1 Different Type of Input Data File

4.2 Using SSE instead of AVX

4.3 Correlation function with weights

The default mode of the correlation function calculations assume identical unit weights for all points. However, it is fairly straightforward to extend the code to support weights for individual points.

Acknowledgements