# Angular Power spectrum

Version 1.x (last update of this file 30-Nov-16)

**Angpow** is being developed at the Linear Accelerateur Laboratory (LAL/CNRS/IN2P3 Univ. Paris-Sud, Univ. Paris-Saclay) - 91898 ORSAY CEDEX - FRANCE main author: J.E Campagne co-authors: J. Neveu, S. Plaszczynski

The authors thank M. Reinecke for fruitful discussions and some tools he give us. Some parts of the code uses spherical Bessel functions as implemented in CLASSgal.

# Introduction

**Angpow** performs the computation of the angular spectrum $C_\ell(z_1, z_2)$ between two shells of redshifts $z_1$, $z_2$ according to the following definition

$$C_\ell(z_1, z_2) = \frac{2}{\pi} \int dk\, k^2\ P(k) \Delta_\ell(z_1, k) \Delta_\ell(z_2, k)$$

with in one hand $P(k)$ the non normalized primordial power spectrum and

$$\Delta_\ell(z, k) \approx j_\ell(k\, r(z))\ \tilde{\Delta}_\ell(z, k)$$

where $j_\ell(x)$ is a first kind spherical Bessel function of parameter $\ell$, $r(z)$ the radial comoving distance of perturbations and $\tilde{\Delta}_\ell(z, k)$ a generic function that at least takes into account of the growth factor between $z = 0$ and a diffrent $z$ value. When one introduces redshift selection functions arround $z_1$ and $z_2$, **Angpow** computes

$$\begin{aligned} C_\ell(z_1, z_2) = \frac{2}{\pi} \iint dz\, dz'\ W_1(z; z_1, \sigma_1) W_2(z'; z_2, \sigma_2) \\ \times \int dk\, k^2\ P(k) \Delta_\ell(z, k) \Delta_\ell(z', k) \end{aligned}$$

The method used is a mixture of cartesian quadrature in the redshift 2D-space and a Clenshaw-Curtis-Chebyshev algorithm to perform the integral along the $k$ direction.

It is also provided once the $C_\ell$ are computed, an evaluation of the truncated regularized angular correlation function $\xi(\theta, z_1, z_2)$

$$\xi(\theta, z_1, z_2) = \frac{1}{4\pi}\sum_{\ell=0}^{L_{max}}(2\ell+1)C_\ell(z_1,z_2)P_\ell(\cos\theta)e^{-\ell(\ell+1)/\ell_s^2}$$

The apodization function depends on the $\ell_s$ scale which depends on $L_{max}$, i.e. one may take as a rule of thumb $\ell_s \approx 0.4 L_{max}$. Notice that $1/\ell_s$ gives an achieivable resolution (in radian) due to the apodization.

# Download

Download as Guest git clone git@gitlab.in2p3.fr:campagne/AngPow.git

# Required Librairy

- FFTW >= 3.3.4:

The library is avaliable here. Use `./configure --enable-float --enable-threads --enable-openmp --with-pic --enable-sse` if you compile from sources.

- Notice that some Bessel part of the Boost (version >= 1.58) library has been extracted and put in the `inc/boost` directory provided by **Angpow**.

# Compilation/Installation/Setup

1. edit Makefile and adapat to local platform :

   - adapt to the type of Machine MacOSX (Darwin) vs Linux
   - on Linux adapt the location of FFTW

2. Then, the make depends on the plateform

   ```
   > make
   ```

   will select the system thanks tu uname shell function. The result of "make" is binanry file under `./bin` directory as well as `libanpow.a` library in `./lib`.

3. if you want a detailed profiling report then,

```
> make PROFILING=1
```

4. notice that the include files are in `<angpow-root>/inc/Angpow`.

Other possible argument to make: - make clean: to cleanup objects, angpow library and binary - make tidy: to delete all ~ files - make fullcheck: to run in different initial conditions (see below).

## Plateform tested

```
Mac OS X 10.11.2:  gcc 4.9
Linuxx86_64: gcc 4.9 and 5.2 + intel icpc 15.01

Has been tested on Laptop, on CCIN2P3 and NERSC computers.
```

## Running

As **Angpow** uses OpenMP when necessary, the user is invited to set the number of threads the job can take. Ex. on bash-like shell

```
> export OMP_NUM_THREADS=<number>
> ./bin/angpow anpow_bench<>.ini
```

All the following input parameters uses a $P(k)$ at $z = 0$ produced by CLASSgal with matter density fluctuation only and a standard Cosmology. The details of the 5 input parameter examples are currently:

- anpow_bench1.ini: Auto correlation, Dirac selection at $z = 1$
- angpow_bench2.ini: Cross correlation, Dirac selection $z = 1, 1.05$
- angpow_bench3.ini: Auto correlation, Gauss & Galaxy example (cf. $dN/dz$) selection (to repproduce an example in the CLASSgal article arXiv:1307.1459v4 $\langle z \rangle = 0.55$, $\sigma_z = 0.10$.
- angpow_bench4.ini: Auto correlation, Gauss selection (no extra $dN/dz$) $\langle z \rangle = 1.00$ and $\sigma_z = 0.02$.

- angpow_bench5.ini: Cross correlation, Gauss selection (no extra $dN/dz$); $\langle z \rangle = 0.90, 1.00$ and a common $\sigma_z = 0.02$.

It is provided a bash script `verif.sh` that runs the 5 Tests and extract the maximal absolute value difference w.r.t some reference $C_\ell$ values obtained in one our machine ( `.REF` files) (and the "$\ell_{max}$" location).

## OpenMP stress tests

It is provided a bash script `stress-test.sh` that runs the 5 Tests with diffrent number of threads (1, 2, 4, 8, 16) and make Time average over 10 runs. One can then appreciate on his machine the impact of the multi-thread usage and get the saturation limit.

## List of files

- **root** directory
    - COPYING
    - Readme.md : this file
    - Makefile
    - Darwin_g++_omp_make.inc : file included into Makefile in case of running on Mac Os X system
    - Linux_g++_make.inc : idem but for Linux system
    - angpow.cc: it contains the main() function to perform a generic $C_\ell$ computaion using initial parameter files
    - angpow_bench<*number*>.ini a set of initial parameter files which show how to proceed to run the job:

```
./bin/angpow <init-file>
```

- **src** and **inc/Angpow** directories
    - angpow_bessel: deals with the $j_\ell(x)$ function. Code from CLASSgal has been adapted. The Bessel roots finding algorithm uses BOOST library for the moment.
    - angpow_chebyshevInt: implement the 3C algorithm (needs BLAS-like

implementation and FFTW)

- angpow_clbase: generate list of $\ell$ to be used and interpolate to get a complet list of $C_\ell$
- angpow_cosmo_base.h : base class to be implemented to provide comoving distance
- angpow_cosmo: implementation of angpow_cosmo_base.h with a simple $\Lambda$CDM cosmology
- angpow_ctheta: compute $\xi(\theta, z_1, z_2)$ once the $C_\ell(z_1, z_2)$ are provided
- angpow_exceptions: exception class
- angpow_fft: access to FFTW planning
- angpow_func: base class for generic 1D and 2D functions
- angpow_kinteg: perform the k-integration (bulk of the algo) and the redshift integrations too
- angpow_numbers: to generate machine-based number types (M. Reinecke)
- angpow_parameters: structure which group the user parameters to control the job at diffrent stages (directory, precision, cuts...)
- angpow_pk2cl: hub of the program to perfom the $C_\ell$ computation from a $P(k)$ and redshift selection windows
- angpow_powspec_base.h: base class to povide access to $P(k)$ and $\tilde{\Delta}_\ell(z, k)$ functions
- angpow_powspec: implementation of angpow_powspec_base.h which deals with the power spectrum part (load from file the $P(k)$ at $z = 0$, use a growth function) and define the Eisentein & Hue growth factor function
- angpow_quadinteg: generic and specific 1D quadrature. Used to get low order weights/nodes for instance of Clenshaw-Curtis or Gauss-Kronrod quadrature. This is used for the redshift integrals of the $C_\ell$ with selection functions.
- angpow_radial_base.h: base class for redshift selection function
- angpow_radial: implement angpow_radial_base.h: Dirac, TopHat with apodized edges, Gaussian, and Gauss + Galaxy distribution
- angpow_radint: originaly was design to perform the redshift integral but it is only

    used to provide cartesian quadrature weights/nodes as angpow_kinteg perform all the integrals. It might be cleaned in future relased.

  - angpow_tools: spline, interpolation...
  - angpow_utils: string manipulation and STL predicates
  - walltime_c/walltimer: M. Reinecke usefull timer.

- **root** directory
  - Makefile that should be tuned to the local platform (*.inc files)
  - the main (angpow.cc or limber.cc)
  - initial parameter files (angpow_bench<*number*>.ini)
  - reference outputs (angpow_bench<*number*>.txt.REF)
  - bash scripts (verif.sh, stress-test.sh) and an awk script (diff.awk) used by verif.sh.
- **doc** directory
  - doxydoc : input file to `doxygen` tool to generate the class documentation
- **bin/lib** directories are the location of executable and library
- **data** directory:
  - input power

# Abstract class implementation

**Angpow** design has been driven to allow one to use its own access to :

- power spectrum computation (ie. $P(k)$, $\tilde{\Delta}_\ell(z,k)$. These functions are generalised in a single $P(\ell,k,z)$ function which has to be derived from `PowerSpecBase` class located in `angpow_power_spec_base.h` file. An concrete implementation has been set up in `angpow_power.h` throw the `PowerSpecFile` class which reads an external $(k,P(k))$ tuple and compute internaly a minimal growth function. To implement `PowerSpecBase` one **must** provide:
  - a `clone` function that is simply (Derived to be replaced by the appropriate class name)

```
return new Derived(static_cast<const Derived&>(*this));
```

The Copy Constructor should be explicitly provided BUT avoid deep copy if pointers are used (see for instance `PowerSpecFile` ),

- the main operator `double operator()(int ell, double k, double z)` .
- if pointers are used than one must provide an implementation of `ExplicitDestroy()` to free properly the memory. `Do not use the Destructor of the class to free memory that is used by different clone` .
- Before the $k$-sampling at fixed $\ell$ and $z_i$ values the `Init` function is called by the KIntegrator::compute method, so the user can initialize function at this stage (for instance any growth factor depending only on $z$).

- cosmological comobile distance (ie. $r(z)$) those abstract class `CosmoCoordBase` is located in `angpow_cosmo_base.h` file. An implementation using a $\Lambda$CDM standard cosmology is accessible throw `CosmoCoord` class in `angpow_cosmo.h`
- radial selection functions (ie. $W(z)$) as a $z$ user function derived from `RadSelectBase` class in `angpow_radial_base.h` file. Some implementations exist for Dirac, Gaussian, TopHat with apodized edges, and Gaussian+galaxy function (see details in `angpow_radial.h` file)

The user once he has implemented the abstract classes, may want to set up an application similar to `angpow.cc` . In this perspective the typical workflow of the main routine should at least follow the scheme:

1. Get the pointer to the job processing user parameters via

   ```
   Parameters para = Param::Instance().GetParam();
   ```

2. Initialize the Redshift Selection classes derived from `RadSelectBase` ;
3. Initialize the Cosmological Distance class derived from `CosmoCoordBase` ;
4. Initialize the (generalized) Power Spectrum class derived from `PowerSpecBase` ;
5. Initialize the $C_\ell$ class via

   ```
   Clbase clout(Lmax,para.linearStep, para.logStep);
   ```

6. Instantiate a `Pk2Cl` object and launch the computation as

```
Pk2Cl pk2cl;
pk2cl.Compute(pws, cosmo, Z1win, Z2win, Lmax, clout);
```

7. Then, one may want to save the $C_\ell(z_1, z_2)$ results and launch if desired the $\xi(\theta, z_1, z_2)$ computation.

8. Before closing (free memory if needed) one must explicitly call the PowerSpectrum `ExplicitDestroy` method.

```
pws.ExplicitDestroy();
```