CosmoMad - A C library for cosmology

David Alonso

June 5, 2015

1 Introduction

CosmoMad is a set of routines, functions and macros that can be used to calculate some useful quantities in cosmology. In its current version this library can be used perform calculations within the w_aCDM set of models, i.e.: cold dark matter models, not necessarily flat, with an equation of state for dark energy of the form $w(z) = w_0 + w_a(1-a)$. CosmoMad is written in C and wrapped up as a static library.

1.1 Compilation and usage

The header file and C-source file are `cosmo_mad.h` and `cosmo_mad.c`, and are located in the source directory `src`. Once configured, compiled and installed, a static library will be generated: `libcosmomad.a`, stored in the directory `${prefix}/lib`, where `${prefix}` is the prefix path given to the `configure` script as a command-line parameter. Likewise, the header file will be copied to the directory `${prefix}/include`. Any code using CosmoMad will need to include this header file and be linked with this library

CosmoMad makes extensive use of the GNU Scientific Library (GSL) for mathematical operations such as performing integrals, using special functions or creating splines for interpolation. For this reason the user must have GSL installed and any code using CosmoMad must be correctly linked to the GSL libraries as well.

To install CosmoMad, follow the standard GNU installation procedure:

```
~$ ./configure [configure-options]
~$ make
~$ make install
```

Check the `INSTALL` file in your distribution for more details.

Once CosmoMad has been successfully installed, any code that makes use of these routines must be correctly linked with the library and its dependencies. So, if we want to compile a program `foo` from the source file `foo.c`, something like

```
gcc foo.c-o foo -I[path_to_CosmoMad_header] -I[path_to_GSL_headers]
        -L[path_to_CosmoMad_lib] -L[path_to_GSL_lib]
        -lcosmomad -lgsl -lgslcblas -lm
```

should be used. A sample program is included with the current version in the directory `sample` (see section 6).

2 Units

CosmoMad yields results and accepts data in the following units:

- Length: $[L] = \mathrm{Mpc}\, h^{-1}$
- Time: $[T] = \mathrm{Gyr}\, h^{-1}$
- Mass: $[M] = M_\odot\, h^{-1}$

3 Macros

The following preprocessor macros, defined in the header file, are used by CosmoMad and may be used by any code linked to it:

- `CSM_FOURPITHIRD` : $\frac{4\pi}{3}$
- `CSM_TWOPIPIINV` : $\frac{1}{2\pi^2}$
- `CSM_TWOPIPIINVLOGTEN` : $\frac{\ln(10)}{2\pi^2}$
- `CSM_LOGTEN` : $\ln(10)$
- `CSM_RTOD` : $\frac{180}{\pi}$
- `CSM_DTOR` : $\frac{\pi}{180}$
- `CSM_HGYR` : H_0^{-1} in units of Gyr/h
- `CSM_HMPC` : $c\, H_0^{-1}$ in units of Mpc/h

4 Csm_params

In its current version (> 0.5), CosmoMad defines the structure `Csm_params`, which contains all the necessary information to calculate all the supported quantities for a given cosmological model. It is not our intention to describe here the elements of this structure, since the user is not supposed to meddle with it. However, its definition and those of all related structures are given in the header file `cosmo_mad.h`. Most of the functions described below accept a `Csm_params` struct as their first argument, which defines the cosmological model for which the calculation must be done (once the model has been initialized). This prevents the use of global variables and allows the user to compute the same quantity in different cosmological models simultaneously.

A `Csm_params` structure contains all the information about the background cosmological parameters, power-spectrum and 2-point correlation function information.

5 Routines

All the functions provided by CosmoMad start with the prefix `csm_`.

5.1 General behavior

csm_unset_gsl_eh

```
void csm_unset_gsl_eh(void)
```

A call to this function disables the default GSL error handler. This error handler is very strict and will exit the program if any problem (regarding, for example, the accuracy of an integral) is met. Since sometimes these problems are not so important (an integral reaching a $10^{-3}\%$ accuracy instead of $10^{-4}\%$ may not be problematic), you may want the program to continue its execution anyway. When

called, a taylored error-handler will be used for the current run. This error handler will output error messages to `stderr` beginning with "CosmoMad: ", giving a hint as to what the encountered problem was, and it will exit the program if the error found is clearly important (like finding a NaN). It is recomended to call this function at the beginning of any program using CosmoMad.

csm_set_verbosity

```
void csm_set_verbosity(int verb)
```

Determines the amount of information output. In the current version there are only two levels, 0 (nothing) and 1 (everything). The default level of verbosity is 1 (all messages are output).

csm_params_new

```
Csm_params *csm_params_new(void)
```

Returns an initialized Csm_params structure. Notice that this returns an empty structure, with no associated cosmological information.

csm_params_free

```
void csm_params_free(Csm_params *pars)
```

Frees up all the memory associated with a `Csm_params` structure.

5.2 Mathematical functions

These functions return the result given by the analogous `GSL` routines and are only provided for convenience.

csm_p_leg

```
double csm_p_leg(int l,double x)
```

Returns the `l`-th Legendre polynomial evaluated at `x`: $L_l(x)$.

csm_j_bessel

```
double csm_j_bessel(int l,double x)
```

Returns the `l`-th spherical Bessel function evaluated at `x`: $j_l(x)$.

5.3 Background evolution

csm_background_set

```
void csm_background_set(Csm_params *pars,double OM,double OL,double OB,double w0,double
wa,double hh,double T_CMB)
```

Sets the background cosmology for the structure `pars`: $\Omega_M =$ `OM`, $\Omega_{DE} =$ `OL`, $\Omega_b =$ `OB`, $h =$ `hh`, $w_0 =$ `w0`, $w_a =$ `wa`, $T_{\rm CMB} =$ `TCMB`, with the CMB temperature given in Kelvin. This function must be called for any `Csm_params` used.

csm_cosmic_time

```
double csm_cosmic_time(Csm_params *pars,double aa)
```

Returns the cosmic time corresponding to the scale factor `aa` by calculating the integral

$$t(a) = \int_0^a \frac{da'}{a'\,H(a')} = H_0^{-1} \int_0^a \left(\frac{x}{\Omega_M + \Omega_k\,x + \Omega_{DE}\,x^{-3w}} \right)^{1/2} dx \tag{1}$$

csm_scale_factor

```
double csm_scale_factor(Csm_params *pars,double t)
```

For cosmic time `t` in Gyr/h, this function returns the value of the scale factor. The first time this function is called, the integral (1) is used for several values of a from 0 to 1 and a spline object is created to calculate $a(t)$ faster in all subsequent calls.

csm_hubble

```
double csm_hubble(Csm_params *pars,double aa)
```

Returns the inverse Hubble horizon $H(a)$ at $a =$ `aa` in inverse length units.

csm_omega_matter

```
double csm_omega_matter(Csm_params *pars,double aa)
```

Returns the matter parameter $\Omega_M(a)$ at $a =$ `aa`.

csm_particle_horizon

```
double csm_particle_horizon(Csm_params *pars,double aa)
```

Returns the comoving particle horizon (the maximum distance a particle can have travelled since $a = 0$) at $a =$ `aa` by calculating the integral

$$\chi_p(a) = c \int_0^a \frac{da'}{a'^2\,H(a')} = \frac{c}{H_0} \int_0^a \frac{dx}{x\sqrt{\Omega_M + \Omega_k\,x + \Omega_{DE}\,x^{-3w}}}$$

csm_radial_comoving_distance

```
double csm_radial_comoving_distance(Csm_params *pars,double aa)
```

Returns the radial comoving distance $\chi(a) = \chi_p(1) - \chi_p(a)$ for $a =$ `aa`.

csm_curvature_comoving_distance

```
double csm_curvature_comoving_distance(Csm_params *pars,double aa)
```

Returns the curvature comoving distance at $a =$ `aa`

$$r(a) = \frac{c}{H_0\sqrt{|\Omega_k|}}\,\mathrm{sinn}(H_0\sqrt{|\Omega_k|}\,\chi(a)/c)$$

csm_angular_diameter_distance

```
double csm_angular_diameter_distance(Csm_params *pars,double aa)
```

Returns the angular diameter distance at $a =$ `aa`

$$d_A(a) = a\, r(a)$$

csm_luminosity_distance

```
double csm_luminosity_distance(Csm_params *pars,double aa)
```

Returns the luminosity distance at $a =$ `aa`

$$d_L(a) = \frac{r(a)}{a}$$

csm_growth_factor_and_growth_rate

```
void csm_growth_factor_and_growth_rate(Csm_params *pars,double aa,double *gf,double *fg)
```

Returns the growth factor $D(a)$ and the growth rate $f(a)$ at $a =$ `aa` in the variables `gf` and `fg` respectively. If both quantities are required at the same time it is more efficient to call this function than the two functions below, since both quantities are obtained at the same time when solving the differential equation for the growth of matter perturbations:

$$\frac{d}{da}\left(a^3\, H(a)\frac{dD}{da}\right) = \frac{3}{2}\Omega_M(a)\, H(a)\, a\, D \tag{2}$$

Note that $D(a)$ is normalized to $D(a \to 0) \to a$, and not $D(1) = 1$.

csm_growth_factor

```
double csm_growth_factor(Csm_params *pars,double aa)
```

Returns the growth factor at $a =$ `aa`.

csm_f_growth

```
double csm_f_growth(Csm_params *pars,double aa)
```

Returns the growth rate $f(a)$ at $a =$ `aa`.

csm_theta_BAO

```
double csm_theta_BAO(Csm_params *pars,double aa)
```

Returns the angular position (in degrees) of the BAO peak in the angular correlation function at $a =$ `aa`:

$$\theta_{BAO}(a) = \frac{a\, r_s}{d_A(a)}$$

csm_Dz_BAO

```
double csm_Dz_BAO(Csm_params *pars,double aa)
```

Returns the position (in Δz) of the BAO peak in the radial correlation function at $a =$ `aa`:

$$\Delta z_{BAO}(a) = \frac{H(a)\, r_s}{c}$$

5.4 Power spectrum

csm_set_linear_pk

```
void csm_set_linear_pk(Csm_params *pars,char *fname,double lkmn,double lkmx,
double dlk,double nns,double s8)
```

This function sets the linear matter power spectrum at $a = 1$. There exist several options:

- If `fname` is "BBKS" the power spectrum will be calculated from the BBKS transfer function ([1]) in the interval `lkmn` $< \log_{10}(k) <$ `lkmx`, in intervals of $\Delta \log_{10}(k) =$ `dlk`.
- If `fname` is "EH" the power spectrum will be calculated from the Eisenstein & Hu transfer function [3] in the same fashion.
- If `fname` is "EH_smooth" the power spectrum will be calculated from the Eisenstein & Hu transfer function **without acoustic oscillations**.
- Finally `fname` can be set to the path to a file containing the power spectrum. This file must be in CAMB format, i.e.: two columns (k, $P(k)$) with k in h/Mpc and its values evenly spaced in $\log_{10}(k)$.

Once the $P(k)$ is read (or calculated) it is normalized to $\sigma_8 =$ `s8`. After that a spline object is created for faster interpolation thereafter. The normalization for $P(k)$ used here is such that

$$\langle \delta(\mathbf{x})\delta(\mathbf{x}+\mathbf{r})\rangle \equiv \xi(r) = \frac{1}{2\pi^2}\int_0^\infty P(k)\,\frac{\sin(kr)}{kr}k^2 dk$$

csm_set_nonlinear_pk

```
void csm_set_nonlinear_pk(Csm_params *pars,char *fnamePkHFIT)
```

This function sets the non-linear matter power spectrum at $a = 1$. Three options are available: if `fnamePkHFIT` is set to "RPT" the mildly non-linear power spectrum is approximated by including a Gaussian damping term arising in renormalized perturbation theory ([2]):

$$P(k,z) = P^L(k,z)\,e^{-k^2\sigma_v^2(z)},$$

where

$$\sigma_v^2(z) = \frac{1}{6\pi^2}\int_0^\infty P^L(k,z)dk.$$

Thus in this case $\sigma_v^2(z=0)$ is calculated and used in this way when calling `csm_Pk_nonlinear_0` (below).

If `fnamePkHFIT` is set to "RPT_ss" this Gaussian damping factor is also used, however the small scales are recovered by adding a no-BAO power spectrum:

$$P(k,z) = [P^L(k,z) - P^L_{\rm noBAO}(k,z)]\,e^{-k^2\sigma_v^2(z)} + P^L_{\rm noBAO}(k,z).$$

The no-BAO $P(k)$ is obtained using the Eisenstein & Hu [3] fitting formula without acoustic oscillations. This way only the BAO wiggles are damped.

The last option is to set `fnamePkHFIT` to the path to a file containing a non-linear power spectrum (for example using HALOFIT [8]). The format for this file must be the same as the one used in `csm_set_linear_pk`. Note that in this case there is no way to normalize $P(k)$ to the value of σ_8 used for the linear power-spectrum, but CosmoMad will use the same normalization factor used for the linear case, so one should make sure that both the linear and non-linear $P(k)$'s were generated with the same normalization.

csm_Pk_linear_0

```
double csm_Pk_linear_0(Csm_params *pars,double kk)
```

Returns the linear matter power spectrum at $a = 1$ and $k =$ `kk`. If `kk` is larger than the interpolation limits for $P(k)$ it is approximated by $P(k) \propto k^{n_s}$ for small k and $P(k) \propto k^3$ for large k.

csm_Pk_nonlinear

```
double csm_Pk_nonlinear(Csm_params *pars,double kk)
```

Returns the non-linear power spectrum at $k =$ `kk`. If `kk` is larger than the interpolation limits for $P(k)$ it is approximated by $P(k) \propto k^{n_s}$ for small k and $P(k) \propto k^3$ for large k. This function returns the power spectrum normalized with the growth factor given in `csm_set_Pk_params`, but without bias or RSDs.

csm_set_Pk_params

```
void csm_set_Pk_params(Csm_params *pars,double beta,double gf, double bias,int l_max)
```

Sets the parameters necessary to calculate the full power spectrum in redshift space: $\beta(a) =$ `beta`, $D(a) =$ `gf` and $b =$ `bias` (see equation (3)). `l_max` is the maximum multipole that will be used in the calculation of the power spectru and 3D correlation funciton (e.g. 4 for the Kaiser approximation or 0 for the real-space case – $\beta = 0$).

csm_Pk_full

```
double csm_Pk_full(Csm_params *pars,double kk,double muk)
```

Returns the full redshift-space power spectrum in the Kaiser approximation ([4]):

$$P_s(a, k, \mu_k) = b^2 \, (1 + \beta(a) \, \mu_k^2)^2 \, P_r(a, k), \tag{3}$$

with

$$P_r(a, k) = [D(a)]^2 \, P_{NL}(a, k),$$

and

$$P_{NL}(a, k) \equiv \; P_L(a = 1, k) \exp\left(-[D(a) \, \sigma_v(0)]^2 k^2\right)$$

if RPT was used to set the non-linear power spectrum.

csm_Pk_multipole

```
double csm_Pk_multipole(Csm_params *pars,double kk,int l)
```

Returns the `l`-th multipole of the power spectrum:

$$P_l(k) = \frac{2l + 1}{2} \int_{-1}^{1} L_l(\mu_k) \, P(k, \mu_k),$$

where $L_l(x)$ is the l-th Legendre polynomial.

5.5 Correlation functions

csm_xi2p_L

```
double csm_xi2p_L(Csm_params *pars,double r,double R1,double R2,
char *wf1,char *wf2,double errfac)
```

Let $\delta(\mathbf{x}; R, T)$ be the density contrast smoothed with a window function of type T and smoothing scale R. This function returns the value of the correlation function between $\delta(\mathbf{x}; \mathtt{R1}, \mathtt{wf1})$ and $\delta(\mathbf{x} + \mathbf{r}; \mathtt{R2}, \mathtt{wf2})$. To be more specific, the return value is

$$\xi(r; R_1, R_2) \equiv \frac{1}{2\pi^2} \int_0^\infty P_L(k, z=0)\, W_{T_1}(kR_1)\, W_{T_2}(kR_2)\, j_0(kr)\, k^2 dk$$

The possible values for `wf1` and `wf2` are "TopHat" and "Gauss":

$$W_{TH}(x) = 3\,\frac{\sin x - x\cos x}{x^3}, \quad W_G(x) = \exp(-x^2/2).$$

For some values of the parameters it may be impossible for the GSL integrator to obtain the required accuracy, in which case the error requirement can be altered through `errfac`: the relative error will then be `errfac` 10^{-4} (the recommended value for `errfac` is thus 1).

csm_sig0_L

```
double csm_sig0_L(Csm_params *pars,double R1,double R2,char *wf1,char *wf2)
```

With the notation above, this function returns the value of the covariance of between $\delta(\mathbf{x}; \mathtt{R1}, \mathtt{wf1})$ and $\delta(\mathbf{x}; \mathtt{R2}, \mathtt{wf2})$. I.e. this is equivalent to `csm_xi2p_L(0,R1,R2,wf1,wf2,1)`.

csm_xi_multipole

```
double csm_xi_multipole(Csm_params *pars,double rr,int l)
```

Returns the `l`-th multipole of the redshift-space correlation function. This is done by performing the integral

$$\xi_l(r) = \frac{i^l}{2\pi^2} \int_0^\infty P_l(k)\, j_l(kr)\, k^2\, dk, \tag{4}$$

where $P_l(k)$ is the l-th multipole of the redshift-space power spectrum (as returned by `csm_Pk_multipole`). The first time this function is called a spline is created for each power spectrum multipole in order to accelerate the calculation of the integral above.

csm_set_xi_multipole_splines

```
double csm_set_xi_multipole_splines(Csm_params *pars)
```

If the correlation function multipoles must be calculated repeatedly, it may be faster to calculate first the multipoles once for a set of r-values and then interpolate between these values. This function initializes a set of spline objects that are used thereafter when calling `csm_xi_multipole`. Specifically, a logarithmic-spaced spline is used for $0.1\,\mathrm{Mpc}/h < r < 15\,\mathrm{Mpc}/h$, and a linear-spaced spline is used for $15\,\mathrm{Mpc}/h < r < 500\,\mathrm{Mpc}/h$. Hence subsequent calls to this function will not calculate the integral (4), but a much faster interpolation. If this function is called, for $r > 500\,\mathrm{Mpc}/h$, `csm_xi_multipole` will return 0, and for $r < 0.1\,\mathrm{Mpc}/h$ it will return the value at 0.1 Mpc.

csm_unset_xi_multipole_splines

```
double csm_unset_xi_multipole_splines(Csm_params *pars)
```

Frees up all the memory associated to the splines created when calling `csm_set_xi_multipole_splines`. It is not necessary to call this function at the end of each program, since `csm_params_free` will also take care of this.

csm_xi_3D

```
double csm_xi_3D(Csm_params *pars,double rr,double mu)
```

Returns the anisotropic 3-D correlation function $\xi(r,\mu)$ as a sum over multipoles:

$$\xi(r,\mu) = \sum_{l=0}^{\infty} \xi_l(r)\, L_l(\mu).$$

Note that under the Kaiser approximation (the one used in the present version of CosmoMad) only the first three multipoles ($l = 0,\ 2,\ 4$) are used. When many calls to this function are necessary it may be wise to call `csm_set_multipole_splines` first for a better performance.

csm_xi_pi_sigma

```
double csm_xi_pi_sigma(Csm_params *pars,double pi,double sigma,int use_multipoles)
```

Returns the anisotropic 3-D correlation function $\xi(\pi,\sigma)$ using longitudinal ($\pi \equiv \mu$) and transverse ($\sigma \equiv \sqrt{r^2-\pi^2}$) coordinates. If `use_multipoles` is set to 1 the sum over multipoles described above is used. If set to 0 the following double integral is performed:

$$\xi(\pi,\sigma) = \frac{1}{2\pi^2}\int_0^\infty dk_\parallel \cos(k_\parallel\,\pi)\int_0^\infty dk_\perp\, k_\perp\, J_0(k_\perp\,\sigma) P(k_\parallel,k_\perp),$$

where $J_0(x)$ is the 0-th order cylindrical Bessel function. Note that the latter approach, although exact, will be much slower than the former, unless a large number of multipoles is needed.

5.6 Halo mass function

csm_M2R

```
double csm_M2R(Csm_params *pars,double mass)
```

Returns the comoving radius of a sphere of mass `mass` (in units of $M_\odot/h$). These two quantities are related through

$$M = \frac{4\pi}{3}\Omega_M \, (2.776 \times 10^{11} \, M_\odot/h) \left(\frac{R}{1\,\mathrm{Mpc}/h}\right)^3 \tag{5}$$

csm_R2M

```
double csm_R2M(Csm_params *pars,double radius)
```

Returns the mass of a sphere of comoving radius `radius`.

csm_collapsed_fraction

```
double csm_collapsed_fraction(Csm_params *pars, double mass,char *mf_model)
```

Returns the fraction of the Universe that has collapsed into halos of mass larger than `mass` according to the mass function parametrization given by `mv_model`. Three models are supported:

- "PS", [6]:

$$F_{\rm PS}(< M) = \mathrm{erfc}(\nu/\sqrt{2}) \tag{6}$$

- "JAP", [5]:

$$F_{\rm JAP}(< M) = \frac{\exp(-c\,\nu^2)}{1 + a\,\nu^b}, \tag{7}$$

  with $(a, b, c) = (1.529, 0.704, 0.412)$.

- "ST", [7]:

$$F_{\rm ST}(< M) = A\left[\mathrm{erfc}\left(\sqrt{\frac{a}{2}}\nu\right) + \frac{\Gamma(1/2 - p, a\,\nu^2/2)}{\sqrt{\pi}\,2^p}\right], \tag{8}$$

  with $(A, a, p) = (0.322, 0.707, 0.3)$.

6 Sample program

Here's a sample code using this library. This code takes a redshift as a command-line argument and calculates several background quantities at that redshift, as well as the power spectrum and correlation functions (which are written into ASCII files):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "cosmo_mad.h"

#define N_K 100
#define N_R 100

int main(int argc,char **argv)
{
  FILE *fout;
  int ii;
  double z,a;
  Csm_params *pars=csm_params_new();

  if(argc!=2) {
    fprintf(stderr,"Usage: ./test_cosmomad redshift \n");
    exit(1);
  }
  z=atof(argv[1]); //Read redshift
  a=1/(1+z);

  csm_unset_gsl_eh(); //Disable GSL error handler
  csm_set_verbosity(1); //Set verbosity level
  csm_background_set(pars,0.25,0.75,0.044,-1,0,0.7,2.75); //Set background cosmology
  printf("At z = %.3lf: \n",z);
  printf("t       = %.3lf Gyrs/h\n",
   csm_cosmic_time(pars,a));
  printf("chi_h   = %.3lf Mpc/h\n",
   csm_particle_horizon(pars,a));
  printf("chi     = %.3lf Mpc/h\n",
   csm_radial_comoving_distance(pars,a));
  printf("r       = %.3lf Mpc/h\n",
   csm_curvature_comoving_distance(pars,a));
  printf("d_A     = %.3lf Mpc/h\n",
   csm_angular_diameter_distance(pars,a));
  printf("d_L     = %.3lf Mpc/h\n",
   csm_luminosity_distance(pars,a));
  printf("D       = %.3lf \n",
   csm_growth_factor(pars,a)/csm_growth_factor(pars,1));
  printf("f       = %.3lf \n",
   csm_f_growth(pars,a));
  printf("th_BAO = %.3lf deg \n",
   csm_theta_BAO(pars,a));

  //Set P(k)-related stuff
  csm_set_linear_pk(pars,"Pk_CAMB_MICE.dat",-4,4,0.1,0.95,0.8);
  csm_set_nonlinear_pk(pars,"Pk_HaloFit_CAMB_MICE.dat"); //To use HaloFit
  //  csm_set_nonlinear_pk("RPT"); //To use RPT
  csm_set_Pk_params(pars,
        csm_f_growth(pars,a),
        csm_growth_factor(pars,a)/csm_growth_factor(pars,1),
        1,0,0,4);
  fout=fopen("sample_pk.dat","w");
  for(ii=0;ii<N_K;ii++) {
    double logk=-5+(double)ii/(N_K-1)*10;
    double kk=pow(10,logk);
    double pkl=csm_Pk_linear_0(pars,kk);
    double pknl=csm_Pk_nonlinear(pars,kk);
    fprintf(fout,"%lE %lE %lE \n",kk,pkl,pknl);
  }
  fclose(fout);
```

```
65
66    //Calculate correlation functions
67    fout=fopen("sample_xi.dat","w");
68    for(ii=0;ii<N_R;ii++) {
69      double rr=200.*ii/(N_R-1);
70      double xil,xi0,xi2,xi4;
71      xil=csm_xi2p_L(pars,rr,5,5,"Gauss","Gauss",1); //Linear CF
72      xi0=csm_xi_multipole(pars,rr,0);               //Monopole
73      xi2=csm_xi_multipole(pars,rr,2);               //Quadrupole
74      xi4=csm_xi_multipole(pars,rr,4);               //Hexadecapole
75      fprintf(fout,"%lE %lE %lE %lE %lE\n",rr,xil,xi0,xi2,xi4);
76    }
77    fclose(fout);
78
79    csm_params_free(pars); //Clean everything up
80
81    return 0;
82  }
```

This code, together with its compilation script is included in the present version of CosmoMad in the directory `sample`.

References

[1] Bardeen J. M., Bond J. R., Kaiser N., Szalay A. S. (1986). *The statistics of peaks of gaussian random fields.* ApJ, **304**:15

[2] Crocce M., Scoccimarro R. (2006) *Renormalized cosmological perturbation theory.* PRD, **73**:063519

[3] Eisenstein D. J., Hu W. (1998). *Baryonic features in the matter transfer function.* ApJ, **496**:605

[4] Kaiser N. (1987). *Clustering in real space and in redshift space.* MNRAS **227**:1

[5] Peacock J. A. 2007, MNRAS, 379, 1067

[6] Press W. H., Schechter P., 1974, ApJ, 187, 425

[7] Sheth R. K. & Tormen G. 2002, MNRAS, 329, 61

[8] Smith R. et al. (2003) *Stable clustering, the halo model and nonlinear cosmological power spectra.* MNRAS, **341**:1311