An implementation of the COSEBI in the frame of the Euclid project

Numerical technics for COSEBI computation

Bertrand Morin
bmorin@Cea.fr

Outline

- Introduction
- Synopsis
- Incomplete Gamma function
- Cnj coefficients
- Roots
- Normalization factor
- Weight filter functions T+(θ) & T-(θ)
- E, B, EB modes
- Performance and future

Introduction

Cosebi principle

Papers

- The ring statistics – how to separate E- and B-modes of cosmic shear correlation function on a finite interval
  P.Schneider and M. Kilbinger
- **COSEBIs**: Extracting the full E-/B-mode information from cosmic shear correlation functions
  Peter Schneider, Tim Eifler, and Elisabeth Krause. A&A April 25,2018
- B-modes in cosmic shear from source redshift clustering
  P.Schneider, L. van Warerbeke, Y.Mellier A&A 389,729-741 (2002)

Goal of the implementation :

$$E = \frac{1}{2}\int_0^\infty \mathrm{d}\vartheta\, \vartheta\, [T_+(\vartheta)\xi_+(\vartheta) + T_-(\vartheta)\xi_-(\vartheta)]\ ,$$
$$B = \frac{1}{2}\int_0^\infty \mathrm{d}\vartheta\, \vartheta\, [T_+(\vartheta)\xi_+(\vartheta) - T_-(\vartheta)\xi_-(\vartheta)]\ ,$$

(Eq. 1)

$$EB = \int_0^\infty \theta[T_-(\theta)\xi_x(\theta)]d\theta$$

Weight filter functions T+ and T- :

Jx : Bessel functions order 0 and 4

$$\int_0^\infty \mathrm{d}\vartheta\, \vartheta\, \mathrm{J}_0(\ell\vartheta)T_+(\vartheta) = \int_0^\infty \mathrm{d}\vartheta\, \vartheta\, \mathrm{J}_4(\ell\vartheta)T_-(\vartheta)$$

(Eq. 2)

T_+ = 0 for $\theta > \theta_{\max}$ and $\theta < \theta_{min}$

T_- is defined on [θ_{min} , $\theta_{\max}$]

Introduction

Constraints and scaling

- First constraint is derived from (Eq.1) leading to the following relation :

$$\int_{\vartheta_{\min}}^{\vartheta_{\max}} \mathrm{d}\vartheta\, \vartheta\, T_+(\vartheta) = 0 = \int_{\vartheta_{\min}}^{\vartheta_{\max}} \mathrm{d}\vartheta\, \vartheta^3\, T_+(\vartheta) \quad \text{(Eq. 4)}$$

- Second constraint concerns the construction of a set of weight functions polynomials in θ and orthonormal :

$$\int_{-1}^{1} \mathrm{d}x\, t_{+n}(x)\, t_{+m}(x) = \delta_{mn} \quad \text{(Eq. 16)}$$

Interval transformation : $\theta \in [\theta_{min}, \theta_{max}]$ => $x \in [1, -1]$; we have set $T_{+n}(\theta) = t_{+n}(x)$

- Scaling : logarithmic set of polynomial filters have to be built (polynomial in ln θ)

Aim: to have a good sampling of the small scale of the shear correlation function $\xi(\theta)$

$$t_{+n}^{\log}(z) = \sum_{j=0}^{n+1} c_{nj} z^j = N_n \sum_{j=0}^{n+1} \bar{c}_{nj} z^j \qquad z = \ln(\vartheta/\vartheta_{\min})$$

> expect of $\xi(\theta)$ to contain more structures at small scales than large scales

Cosebi implementation context

Introduction

Context

* Mathematica code computing 20 modes in the range [1 to 400 arcmin] is included in the paper of COSEBI P.Schneider, T.Eifler and E.Krause
  * ➢ we will use this code to verify our implementation.
* Euclid demands to use either C++ or Python language and the available libraries coming with EDEN *
  * ➢ Mathematica commercial product is not included in EDEN, it will be replaced by the C++ for its speed.
  * ➢ BOOST library available in EDEN will be selected.
* Scientific Euclid requirements specify to compute E and B log-COSEBI with 10 modes [3.4",29"]
  * ➢ we will use the parameters of the COSEBI paper (20 modes)
* Scientific Euclid requirements request to compute for each mode 2 million bins.
  * ➢ our implementation has to be quick
* Roots and normalization computations for getting a polynomial t_{+n} have to be solved with a 'pretty good' precision. The COSEBI paper recommends 40 digits in order to attain 0.1 (normally zero) in the formula below

$$\int_0^{z_{\max}} \mathrm{d}z\, \mathrm{e}^{2z}\, t_{+n}^{\log}(z) = 0 = \int_0^{z_{\max}} \mathrm{d}z\, \mathrm{e}^{4z}\, t_{+n}^{\log}(z) \qquad z = \ln(\vartheta/\vartheta_{\min})$$

* ➢ We need to find mathematical libraries working with arbitrary floating precision numbers
  * ✓ MBLAS , MLAPACK based on GMP ?
  * ✓ Needs: Linear algebra solver, non-linear solver, quadrature (discrete , uniform), Incomplete Gamma-function, …
  * ✓ Coupled with the Boost library, we will develop our own library with the main advantage to keep control on it

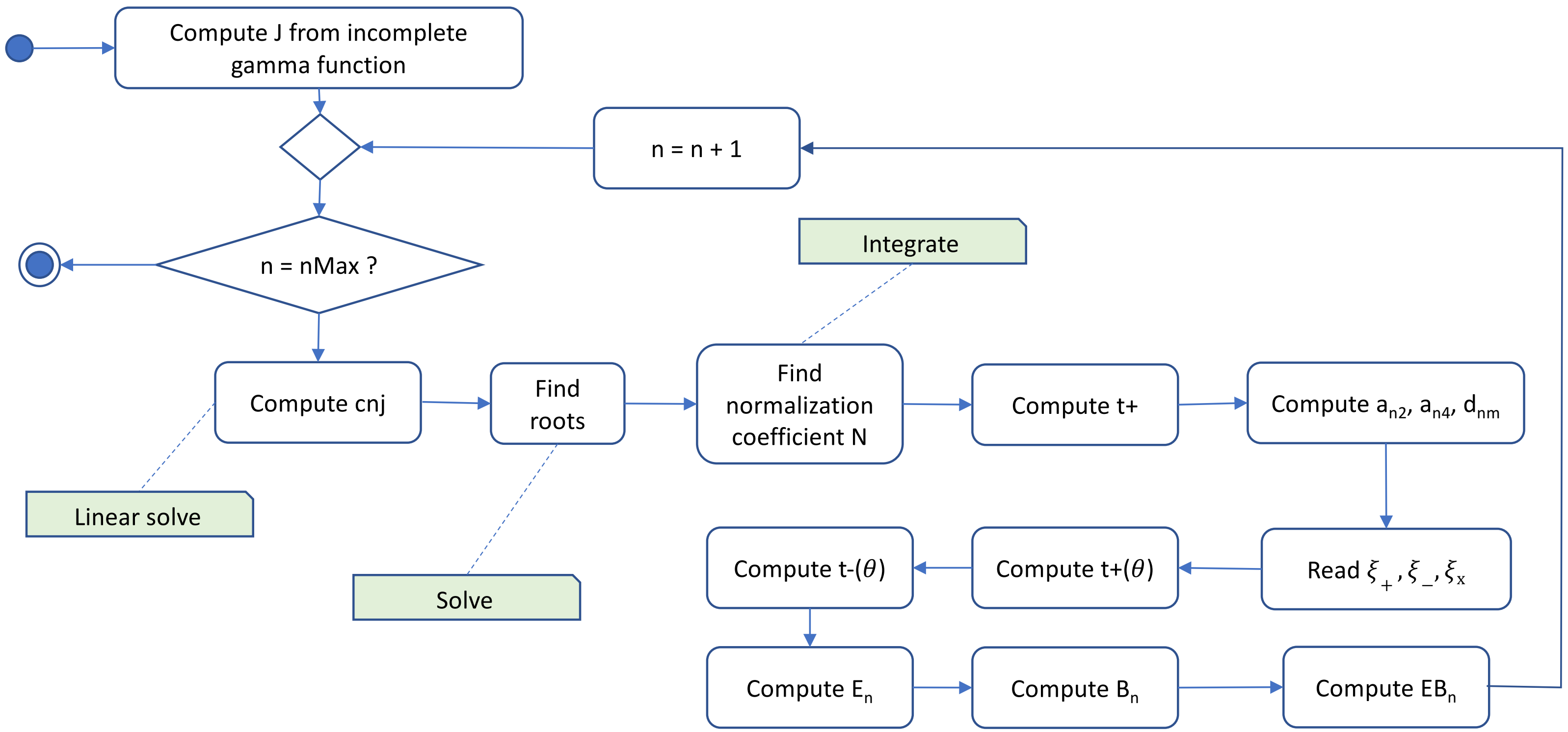
* : **E**uclid **D**evelopment **EN**vironment

Introduction

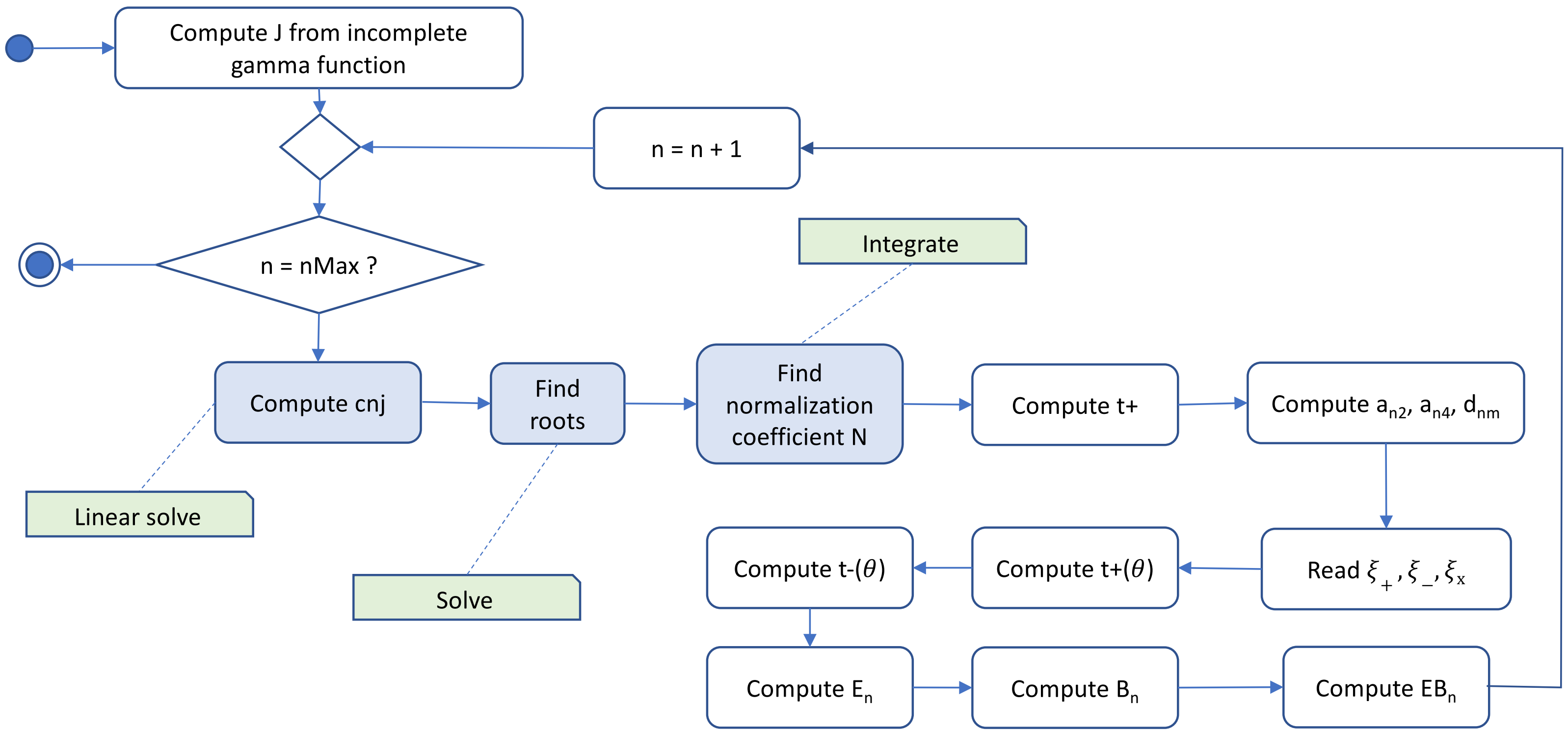
Cosebi challenge in a nutshell

- Need to use an arbitrary precision library – COSEBI paper recommends precisions from 50 to 130 digits for the computations.
- Need to assess the suitable numerical algorithms
- Need to design and to code a set of efficient numerical algorithms (manipulating an arbitrary precision number is not very fast)
- Need to process numerous shear correlation function bins quicker as possible
- Additional challenge: to have a weak coupling with the arbitrary precision libraries

Synopsis


Compute J from incomplete gamma function
n = n + 1
n = nMax ?
Integrate
Compute cnj
Find roots
Find normalization coefficient N
Compute t+
Compute a_{n2}, a_{n4}, d_{nm}
Linear solve
Solve
Compute t-(θ)
Compute t+(θ)
Read ξ_+, ξ_-, ξ_x
Compute E_n
Compute B_n
Compute EB_n

Synopsis


Compute J from incomplete gamma function
n = n + 1
n = nMax ?
Integrate
Compute cnj
Find roots
Find normalization coefficient N
Compute t+
Compute a_{n2}, a_{n4}, d_{nm}
Linear solve
Solve
Compute t-(θ)
Compute t+(θ)
Read ξ_+, ξ_-, ξ_x
Compute E_n
Compute B_n
Compute EB_n

1^{st} Step : J & incomplete Gamma function

- The first step consist to create a set of coefficients J from the computation of the γ function :

$$J(k,j) = \int_0^{z_{\max}} \mathrm{d}z \, \mathrm{e}^{kz} z^j = \frac{\gamma(j+1, -kz_{\max})}{(-k)^{j+1}}$$

- Boost library implements many γ functions, but not this one : $\boxed{\gamma(a,z) = \int_0^z e^{-t} t^{(a-1)} dt}$ with $a > 0$

- The following recursive method has been implemented :

$$\boxed{\begin{array}{c} \Rightarrow \\ z^{(a-1)} * (-e^{-z}) + (a-1) * \{ \\ z^{(a-2)} * (-e^{-z}) + (a-2) * \{ \\ z^{(a-3)} * (-e^{-z}) + (a-3) * \{ \\ \ldots + (a-n+1) * \{ \int_0^z t^{(0)} * e^{-t} dt \}\}\}\} \ldots \end{array}}$$

Digital precision used : 130 digits

2^{nd} step : $\bar{c}$nj computation

- Introduction
- Synopsis
- J & γ $function$
- **Cnj coefficients**

$$c_{nj} = N_n \bar{c}_{nj}$$

$$z = \ln(\vartheta/\vartheta_{\min})$$

$$T^{\log}_{+n}(\vartheta) = t^{\log}_{+n}(z) \text{ and } T^{\log}_{-n}(\vartheta) = t^{\log}_{-n}(z)$$

$$t^{\log}_{+n}(z) = \sum_{j=0}^{n+1} c_{nj} z^j = N_n \sum_{j=0}^{n+1} \bar{c}_{nj} z^j$$

Error propagation: each cnj coefficient depends on the cnj coefficients computed previously.

- One more reason to compute in high precision to mitigate the error propagation to the high polynomial order.

✓ Constraints (Eq. 4) become :

$$\sum_{j=0}^{n} \bar{c}_{nj} J(2, j) = -J(2, n+1) ,$$

$$\sum_{j=0}^{n} \bar{c}_{nj} J(4, j) = -J(4, n+1) .$$

(Eq. 33)

✓ Orthogonality conditions :

$$\sum_{j=0}^{n+1} \sum_{i=0}^{m+1} J(1, i+j) \, \bar{c}_{mi} \, \bar{c}_{nj} = 0$$

(Eq. 34)

- We can compute immediately : $\bar{c}_{10}$, $\bar{c}_{11}$
- We need to solve a set of linear algebra systems.

2^{nd} step : $\bar{c}nj$ computation in detail

From Eq. 33

n = 0:

$$\boxed{c_{00}\frac{-J(2,1)}{J(2,0)}=\frac{-J(4,1)}{J(4,0)}}$$

n = 1:

$$c_{10}\,J(2,0) + c_{11}J(2,1) = -J(2,2)$$
$$c_{10}\,J(4,0) + c_{11}J(4,1) = -J(4,2)$$

n = 2:

$$c_{20}\,J(2,0) + c_{21}J(2,1) + c_{22}J(2,2)= -J(2,3)$$
$$c_{20}\,J(4,0) + c_{21}J(4,1) + c_{22}J(4,2)= -J(4,3)$$

$$A_0\,C_{20} + A_1\,C_{21} + A_2\,C_{22} + A_3 = 0$$

From Eq. 34

$$A_0 = [\ J(1,0)c_{10} + J(1,1)c_{11} + J(1,2)\ ]$$
$$A_1 = [\ J(1,1)c_{10} + J(1,2)c_{11} + J(1,3)\ ]$$
$$A_2 = [\ J(1,2)c_{10} + J(1,3)c_{11} + J(1,4)\ ]$$
$$A_3 = [\ J(1,3)c_{10} + J(1,4)c_{11} + J(1,5)\ ]$$

2nd step : $\bar{c}nj$ computation in detail

n = 3:

$$c_{30}\, J(2,0) + c_{31}J(2,1) + c_{32}J(2,2) + c_{33}J(2,3) \quad = -J(2,4)$$
$$c_{30}\, J(4,0) + c_{31}J(4,1) + c_{32}J(4,2) + c_{33}J(4,3) \quad = -J(4,4)$$

$$K_0\, C_{30} + K_1\, C_{31} + K_2\, C_{32} + K_3C_{33} \ + K_4 = 0$$

$$K_0 = [\ J(1,0)C_{10} + J(1,1)C_{11} + J(1,2)\, C_{12}\ ]$$
$$K_1 = [\ J(1,1)C_{10} + J(1,2)C_{11} + J(1,3)\, C_{12}\ ]$$
$$K_2 = [\ J(1,2)C_{10} + J(1,3)C_{11} + J(1,4)\, C_{12}\ ]$$
$$K_3 = [\ J(1,3)C_{10} + J(1,4)C_{11} + J(1,5)\, C_{12}\ ]$$
$$K_4 = [\ J(1,4)C_{10} + J(1,5)C_{11} + J(1,6)\, C_{12}\ ]$$

$$L_0\, C_{30} + L_1\, C_{31} + L_2\, C_{32} + L_3C_{33} \ + L_4 = 0$$

$$L_0 = [\ J(1,0)C_{20} + J(1,1)C_{21} + J(1,2)\, C_{22} \ + J(1,3)\ ]$$
$$L_1 = [\ J(1,1)C_{20} + J(1,2)C_{21} + J(1,3)\, C_{22} \ + J(1,4)\ ]$$
$$L_2 = [\ J(1,2)C_{20} + J(1,3)C_{21} + J(1,4)\, C_{22} \ + J(1,5)\ ]$$
$$L_3 = [\ J(1,3)C_{20} + J(1,4)C_{21} + J(1,5)\, C_{22} \ + J(1,6)\ ]$$
$$L_4 = [\ J(1,4)C_{20} + J(1,5)C_{21} + J(1,6)\, C_{22} \ + J(1,7)\ ]$$

Warning : error propagation

2^{nd} step: algebra linear solver

- Linear algebra solver has to solve systems such as AX = B
- Two candidate methods have been considered:
  - Gauss-Jordan solver
  - LU decomposition
- First method implemented : Gauss-Jordan Solver.
  - The main difficulty of this method is to manage the ill-conditioned system.
    In order to overtake it, three mechanisms have been coded:
    Line permutation, column permutation, discrepancy mitigation from recursive solving operations
  - ➢ The method given suitable results, the other linear solver methods have not been really explored (except LU where a basic decomposition has been coded)
- Second method LU decomposition :
  1. We write A = LU such as L is Lower triangular (top coefs=0) and U is upper triangular (top coefs≠0) ,
  2. thus we can rewrite our system AX=B = (LU)x = L(Ux) = B ,
  3. we apply a forward substitution to find y such as L y = B ,
  4. then a backward substitution to solve x with U x = y.
  - ➢ Main advantage of the method , we can re-use our LU decomposition with differ
  - ➢ We can increase the stability of the linear solver when we introduce the permutations (P):
    1. We have PA = LU , thus the lower triangular system to solve becomes Ly = Pb for y,
    2. while the upper triangular system stays unchanged Ux = y for x

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix} = \begin{vmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & l_{34} \\ l_{41} & l_{42} & l_{43} & l_{44} \end{vmatrix} \begin{vmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{vmatrix}$$

Gauss-Jordan method

Our linear system to solve :

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix} \begin{vmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{vmatrix} = \begin{vmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{vmatrix}$$

Principle : we take successively each line as a pivot, in our case we will have four iterations.
At each iteration we replace the element a_{ii} by 1 and all elements of the column a_{ji} with j <> i by 0. Example with the first line:

1.We divide the first line by a_{11}
2.We remove at the second line the first line*a_{21} : $L_2 = L_2 - L_1 * \frac{a_{21}}{a_{11}}$
3.We remove at the third line the first line*a_{31} : $L_3 = L_3 - L_1 * \frac{a_{31}}{a_{11}}$
4.We remove at the fourth line the first line*a_{41} : $L_4 = L_4 - L_1 * \frac{a_{41}}{a_{11}}$

After the first iteration we get :

$$\begin{vmatrix} 1 & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{vmatrix} \begin{vmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{vmatrix} = \begin{vmatrix} y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \end{vmatrix}$$

At the fourth iteration, we have solved the system :

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{vmatrix} = \begin{vmatrix} y''_1 \\ y''_2 \\ y''_3 \\ y''_4 \end{vmatrix}$$

Problem : As we can see, we divide the lines by coefficients (a_{11} , then a'_{22}, ...) the value of which is unknown (since this value changes at each iteration). When the coefficient is close to zero, the result of the division can be wrong due to the precision type used (i.e. double) and the error will be propagated to the other calculation at each iteration.

First improvement:
We swap the lines in order to have the biggest coefficient in a_{ii} of the pivot line.
We swap the columns in order to have the biggest coefficient in a_{ii} of the pivot line.

Residual problem: rounding errors are due to an ill-conditioned system.
The smallest error produces a result having potentially a big divergence with the true solution.

How to mitigate the errors:
The true solution is A X = Y and we get an approximation: $A^* X^* = Y$ since we have computed X^* that is a wrong X in relationship with a coefficient matrix A^*. In fact the solution is exact when $A^*A = I$ or $(A^*)^{-1} = A$. Cosebi implementation contains the following algorithm:

1. We start to compute X^*
2. We compute $A X^* = Y^*$
3. $| Y^* - Y | <$ epsilon -> system is solved.
4. We define $\Delta X = X - X^*$ and $\Delta Y = Y - Y^*$
5. We solve $A \Delta X = \Delta Y$ in order to have ΔX
6. We compute a more precise value of $X = X^* + \Delta X$
7. We revert to step 2 where X becomes our new X^*

3rd step : Solving the roots

Polynomial's roots r_{ni} to be solved in : $t_{+n}^{\log}(z) = \sum_{j=0}^{n+1} c_{nj} z^j = N_n \sum_{j=0}^{n+1} \bar{c}_{nj} z^j$ $\quad$ $t_{+n}^{\log}(z) = N_n \prod_{i=1}^{n+1} (z - r_{ni})$.

Start point :

- we only have real roots
- we do not have identical roots

Principle of the roots finding method implemented :

1. Since the polynomial of degree n means n roots, the Bisection method is used to isolate all the roots in a set of n ranges.
2. Each range containing one root, we apply the dichotomy method on each range to approximate the result till $|x_n - x_{n-1}| < \varepsilon_b$
3. We apply the secant method to refine the roots till $|x_n - x_{n-1}| < \varepsilon_s$

   ε_b and ε_s are given by the user to reach the accuracy required by the COSEBI

Method coded and eventually put a side :

Bairstow's method

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a0$$

$P_n(x)$ is divided by a polynomial of degree 2 : $x^2 - px - q$ => $P_n(x) = (x^2 - px - q) P_{n-2}(x) + Remain(x)$

We find the coefficients p and q in such way the Remain is zero.

This method is able to solve complex type roots.

Raphson-Newton algorithm is used to solve the Remain, the initial point is crucial to find roots with accuracy.

We need to add a 'Stabilizing bairstow's method'

How to localize the roots ?

Problem :

We use Polynomial from degree 1 to degree 20 in high precision (130 digits), consequence: the computation time of p(x) has a high cost.

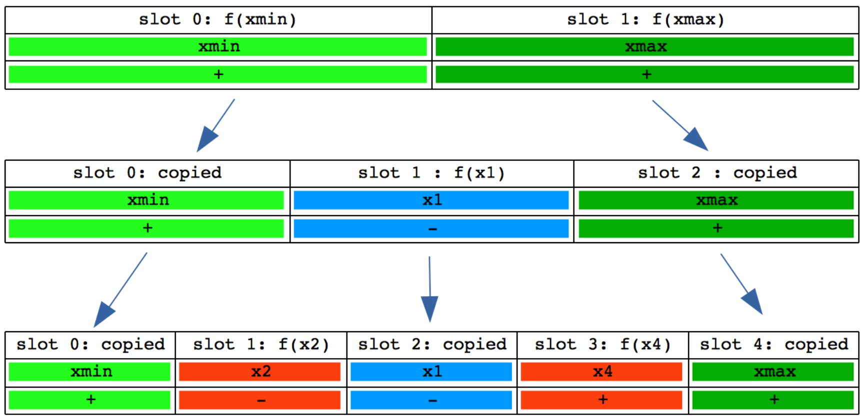
Advantage :

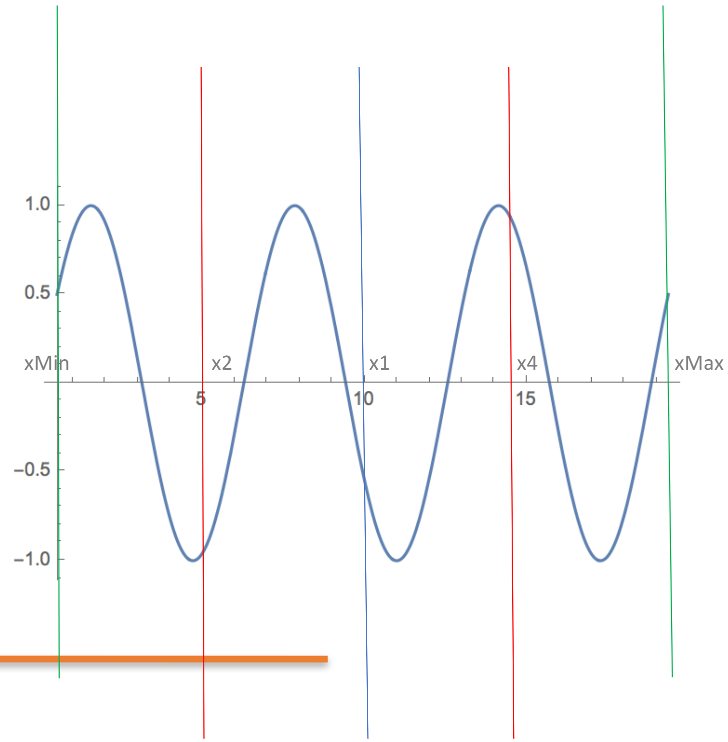
Roots are real and unique.

The degree of our polynomial gives us the number of roots to find.

Principle adopted:

1. Apply the dichotomy method.
2. Avoid as much as possible the computation of f(x).
3. Identify the roots when the sign of f(x) changes.
4. Stop the process when the number of roots reaches the degree of the polynomial.



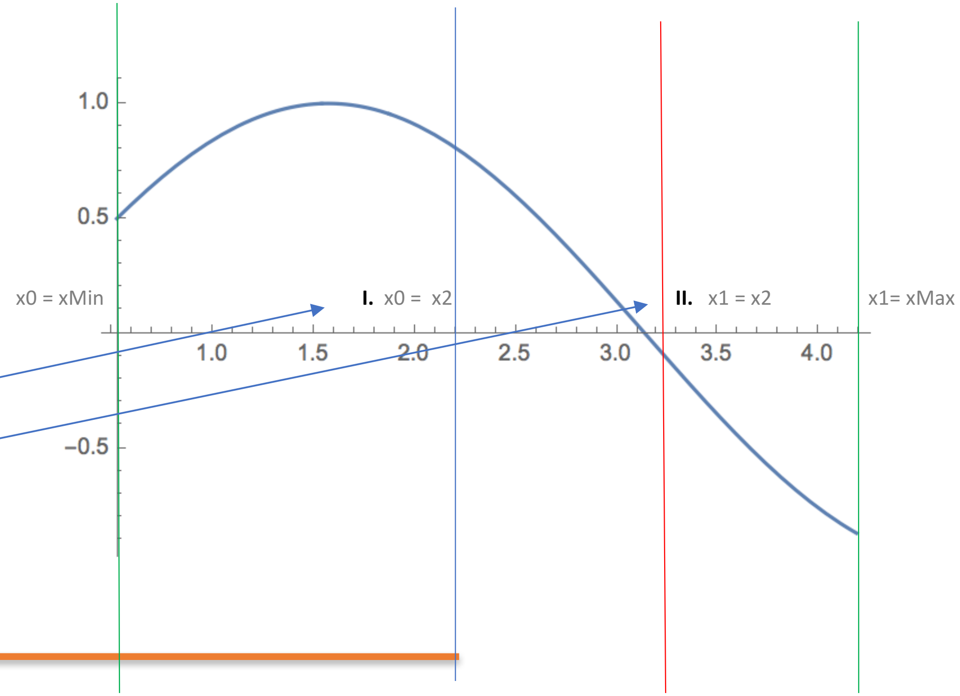


Bisection method

Once we have all the ranges (each one containing a root), we work with each of them to solve the roots.
The first method applied is the bisection method, the second one is the secant method useful to refine the results.

Principle of the bisection method :
The interval is successively divided by 2

Algorithm
1. x_0 = xmin ; $f_0 = f(x_0)$
2. x_1 = xmax ; $f_1 = f(x_1)$
3. While (root not found):
4. $x_2 = (x_0+x_1)/2$; $f_2 = f(x_2)$
5. If $| f_2| <$ epsilon then
   -> x_2 is a root , we can refine with the secant method
6. If $(f_0*f_2) > 0$ then
   $x_0 = x_2$; $f_0 = f_2$ *(case I.)*
   else
   $x_1 = x_2$; $f_1 = f_2$ *(case II.)*

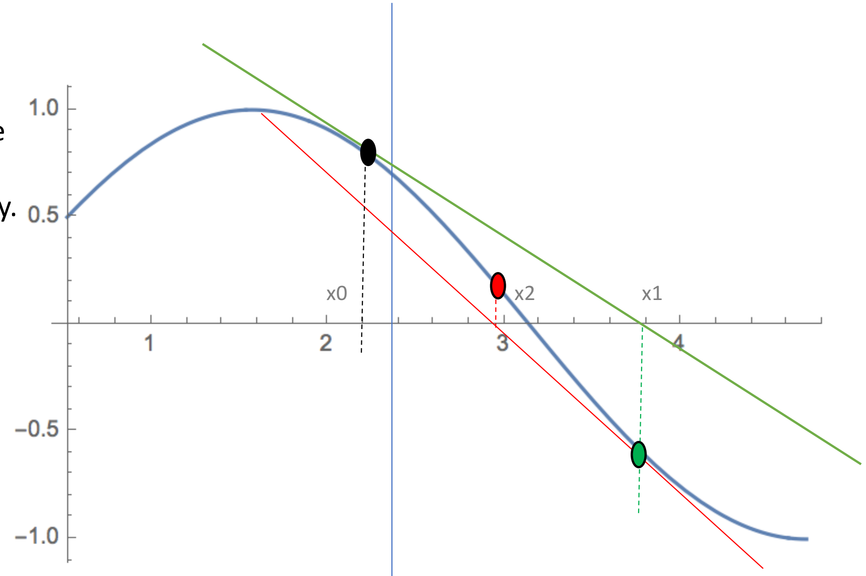


Newton-Raphson method

An interesting method (very easy to implement) is the method of the tangent.
We can reach high precision when we have one root inside an interval.

Principle of the Newton-Raphson method :

- We start with an arbitrary abscissa inside our interval.
- With the tangent equation computed at this abscissa, we solve the new abscissa where the tangent is zero.
- We continue this process till precision on the root is satisfactory.



Algorithm

1. $y = f(x_0) + f'(x)\,(x-x_0)$
2. $0 = f(x_0) + f'(x)\,(x_1-x_0)$
3. $x_1 = x_0 - f(x_0)/f'(x_0)$
4. .....
5. $x_2 = x_1 - f(x_1)/f'(x_1)$
6. ...

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

We stop when $|x_{n+1} - x_n| < \text{epsilon}$

Secant method

The method is a mix between bisection and Newton-Raphson methods.
Advantage : We do not need to compute the derivative of the function.
Disadvantage : the convergence is slower than the Newton-Raphson method.
Used after the bisection method, we need to ingest a smaller epsilon than the one used previously.

Principle of the secant method :

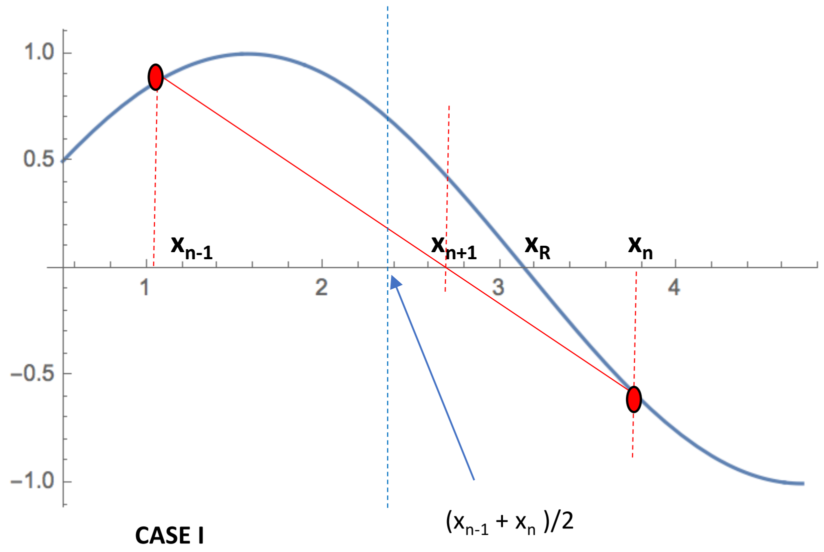
We apply the Thales's theorem in order to get x_{n+1} from x_{n-1} and x_n :

$$\frac{x_{n+1}-x_n}{0-f(x_n)}=\frac{x_n-x_{n-1}}{f(x_n)-f(x_{n-1})}$$

We get : $$x_{n+1}=x_n-(x_n-x_{n-1})\frac{f(x_n)}{f(x_n)-f(x_{n-1})} \qquad \textbf{(1)}$$

We stop when $|x_{n+1}-x_n| < \text{epsilon}$

CASE I : x_{n-1} and x_n surround x_R



Secant method (Newton convergence)

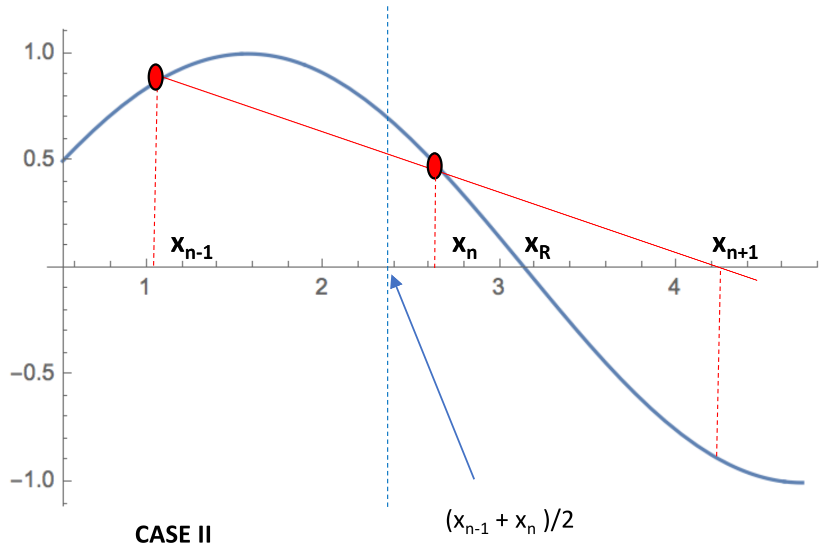
CASE II : x_{n-1} and x_n are on the same side : x_R is not in the interval [x_{n-1} , x_n]

Taking the finite difference : $$f'(x_n)=\frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}$$

We can replace in the Newton Raphson formula : $$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$$

We find again the formula (1) :

$$x_{n+1}=x_n-(x_n-x_{n-1})\frac{f(x_n)}{f(x_n)-f(x_{n-1})}$$



CASE II

4^{th} step : to compute the normalization

- Normalization coefficients N_n renders our T+,T- as a set of polynomial orthonormal
  - This normalization needs numerical integration
- Two main families of integration methods exist :
  - Those using regular intervals
  - Those using remarkable abscissae (non-uniform mesh)
- Regular interval's methods coded
  - Trapezium
  - Simpson
  - Romberg's method (based on the Richardson extrapolation)
    - Encapsulate the method such as Trapezium and Simpson
- Irregular interval's method coded (but not used)
  - Gauss n points integration also called Gauss-Legendre n point rule)
    - Integration limits [-1,1] , weight function w(x) = 1
- Many other non-uniform mesh spacing integration methods exist such as
  - Gauss Laguerre
    - Integration limits [0,∞[, weight function w(x) = e^{-x}
  - Gauss-Chebychev
    - Integration limits [-1,1] , weight function = w(x) = $1/\sqrt{1-x^2}$

$$\int_a^b f(x)dx = \sum_{i=1}^{\infty} w_i f(x_i) \sim \sum_{i=1}^{n} w_i f(x_i)$$

Trapezium's rule

Cnj coefficients
Roots
Normalization

Principle:

1. Support of f(x) is divided by a set of n intervals having the same length : f[a,b] : Δx = b-a/n.
2. In each interval we replace f(x) by a line joining two end points [xi,f(xi)] and [xi+1,f(xi+1)].
3. We compute the weight related to the line given the surface of f[xi,xi+1] if we multiply by the support length.
4. We sum all of these elementary surfaces in order to obtain the surface of f(x) in [a,b].

In one interval, we have :

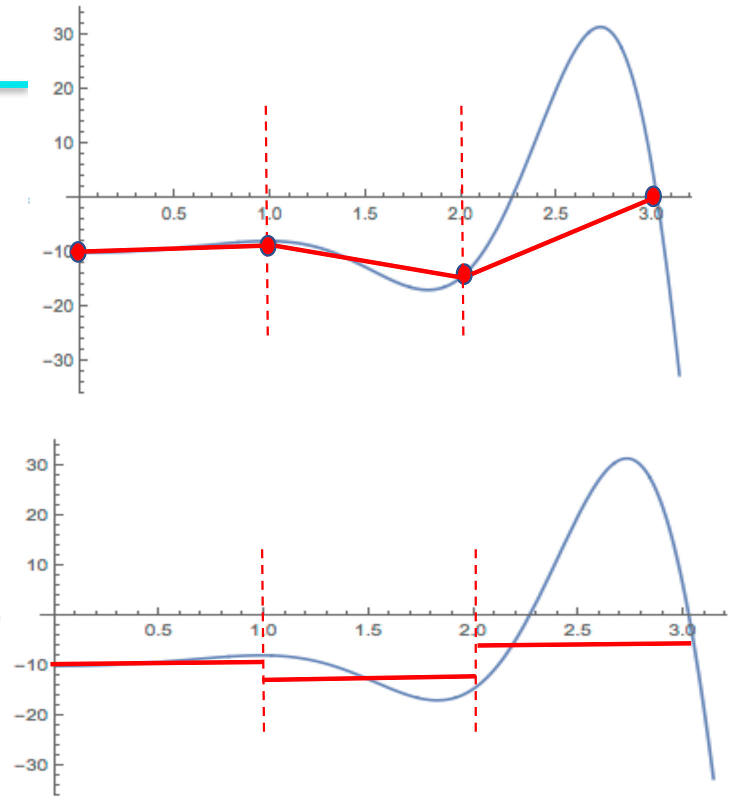
$$\int_{x_{i-1}}^{x_i} f(x)dx = \frac{f_{i-1}+f_i}{2}\Delta x_i \qquad \Delta x_i = x_i - x_{i-1}$$

With all intervals (also called composite trapezium's rule)

$$\int_a^b f(x)dx = \frac{1}{2}\sum_{i=1}^{n}(f_{i-1}+f_i)\Delta x_i \qquad \Delta x_i = \frac{b-a}{n}$$

With a taylor's development where we ignore the terms greater than the second order the error is :

$$\frac{\Delta x^2}{12}(b-a)f''(x)+\epsilon(\Delta x^4)$$



Simpson’s rule

Principle : we replace f(x) by a set of polynomials of degree 2. In each interval, the parabola takes the same value as f{x) at the end points x_i, x_{i+1} and the mid-point m = $(x_i - x_{i+1})/2$.

In one interval, we have :

$$\int_a^b f(x)dx = \frac{\Delta x}{3}[f(a)+4f(\frac{a+b}{2})+f(b)] \quad \text{with} \quad \Delta x = \frac{b-a}{n}$$

Composite Simpson’s rule)

From a taylor's development or a Lagrange interpolation , we have :

$$\int_a^b f(x)dx = \frac{\Delta x}{3}[f_0+4f_1+f_2]+\frac{\Delta x}{3}[f_3+4f_4+f_5]+\dots+\frac{\Delta x}{3}[f_{n-2}+4f_{n-1}+f_n]+\frac{n}{2}\epsilon(\Delta x)^5$$

Error is :

$$\frac{n}{2}\epsilon(\Delta x)^5 = \frac{b-a}{2\Delta x}\epsilon(\Delta x)^5 = \epsilon(\Delta x^4)$$

Applicability : this method is exact with a polynomial of degree 3

Romberg's method

Overview: This method is based on the Richardson extrapolation. Romberg method encapsulates other methods such as trapezium or Simpson's rule in order to refine their result. The goal of this method is to go beyond the accuracy given by the underlying integration method.

Advantage: Quick convergence, better precision than the underlying integration methods

Disadvantage: Romberg is based on a regular abscissas and need to have an analytical form of f(x). Discrete integration is not possible.

Principle : we can write the integral of f(x) as follow : $I=\int_a^b f(x)\,dx=T\,(\Delta x)+c_2(\Delta x)^2+c_4(\Delta x)^4+c_6(\Delta x)^6+\ldots.$

c2, c4, c6 depends on f(x) and its derivative functions only (not of Δ).
T(Δx) is the result of the integration got with a method such as trapezium.

For a better accuracy, we can divide the integration step by 2: $I=\int_a^b f(x)\,dx=T\left(\frac{\Delta x}{2}\right)+c_2\left(\frac{\Delta x}{2}\right)^2+c_4\left(\frac{\Delta x}{2}\right)^4+c_6\left(\frac{\Delta x}{2}\right)^6+\ldots.$
and we continue *with* Δx/4 , Δx/8, …. .
We can also compute the subtraction of $I - I_T(h)$ related to Δx and Δx/2.
In this case, we write h = Δx, qh = Δx/2 ,$q=\frac{1}{2^p}$, p = 1, $I_T(h)$ being an approximation of I.
At 'p' stage, we get the following recurrence formula :

$$I^p(h)\frac{4^p I_T^{(p-1)}(h/2)-I_T(h)^{(p-1)}(h)}{4^p-1}$$

Stop condition:

$T_{p,q}$: $|T_{n-1,1}-T_{n,1}| < \varepsilon$

Algorithm:

q:Trapezium estimation

p:Extrapolated estimation

$T_{1,1}$ $T_{1,2}$ $T_{1,3}$ $T_{1,4}$ …..
$T_{2,1}$ $T_{2,2}$ $T_{2,3}$ ….
$T_{3,1}$ $T_{3,2}$ ….
$T_{4,1}$ ….

$T_{2,1}= T_{1,2} - T_{1,1}$; $T_{2,2}=T_{1,3} - T_{1,2}$ and $T_{2,3} = T_{1,4} - T_{1,3}$
$T_{3,1}= T_{2,2} - T_{2,1}$; $T_{3,2} = T_{2,3} - T_{2,2}$
$T_{4,1}= T_{3,2}-T_{3,1}$

5^{th} step : to compute t+(z) and t-(z) 1/2

- Polynomial t+ coefficients are computed from the roots and the normalization.

$$t_{+n}^{\log}(z) = N_n \prod_{i=1}^{n+1} (z - r_{ni}) \; .$$

z = log(θ/θmin) is computed from discrete angle values found in the 2PCF product.
r = is the roots
N = is the normalization factor

- Scientific requirement (RSD) requests to work with 2 million bins.
- The time to integrate these bins with arbitrary precision numbers is done in 5 days ~ when we have one processor.
  - ➢ We have decided to do the computations in double precision. Should be possible since the roots and normalization factors of the orthogonal filters have been computed with a high precision.
- We need to do a discrete integration of $t_+(z)$ and $t_-(z)$, so we cannot use Romberg's method.

5^{th} step : to compute t+(z) and t-(z) 2/2

- For a given n , t- can be computed from : $t_{-n}^{\log}(z) = a_{n2}\mathrm{e}^{-2z} - a_{n4}\mathrm{e}^{-4z} + \sum_{m=0}^{n} d_{nm} z^m$

and

$$a_{n2} = 4 \sum_{j=0}^{n+1} \frac{c_{nj}\, j!}{(-2)^{j+1}}\,, \quad a_{n4} = 12 \sum_{j=0}^{n+1} \frac{c_{nj}\, j!}{(-4)^{j+1}}\,,$$

$$d_{nm} = c_{nm} + \frac{4}{m!} \sum_{j=m}^{n+1} c_{nj}\, j!(-2)^{m-j-1} \left(3\; 2^{m-j-1} - 1\right)$$

Remark: other formulas can be found in the COSEBI paper. In order to evaluate the performance of these formulas, a cubic spline interpolation method based on the continuity of the first and second derivate has been coded, then used for the integration. Each cubic spline (degree 3) needs to solve a band matrix. The methods above were the most quick and accurate so cubic spline has been dropped.

Last step: computing E, B and EB modes

The input ξ shear correlation functions are known for 2 million bins and we do not known the underlying parametric function (no assumption has to be done).

- In spite of that, discrete integrations from several methods have been coded in order to show the impact of the method on E and B :
  - Discrete sum (possible since we have 2 million bins on a small theta angle interval [1',400’]
  - Trapezoidal integration
  - Simpson integration
  - Spline interpolation could be added

Performance and future

- 50 seconds to compute cnj, the roots, the norm, t+, t-, E, B and EB with 20 modes and 2 million bins on a range = [1,400] arcmin
- Relative accuracies reached on the roots and the norm (comparison done with Mathematica):
  - J : [e-17, e-18]   *[min,max] = minimal and maximal accuracies found among the modes (20) computed .*
  - Cnj : [e-17,e-18]
  - Roots : [e-8, e-10]
  - Norm [e-11,e-17]

What's next ?

- Integration into LODEEN 2.0 (Euclid Environment)
- Technical validation from a cross-checking where E , B and EB modes can be unknown.
- Realistic checking in order to assess the accuracy reached:
  - Process at CC-IN2P3 with mock catalogs containing 500 millions galaxies where E and B modes are well known

Thanks for your attention

Thanks for your attention