Second-Order Non-Gaussianity - SONG

coauthored with GW Pettinari

Institut für theoretische Teilchenphysik und Kosmologie

Christian Fidler

Today

Second-Order Boltzmann Codes

Why do we need second-order perturbations?

- Higher precision
- New physics
  - ➔ Non-Gaussianity
  - ➔ Mode coupling
  - ➔ Scalar-Vector-Tensor coupling

    e.g. B-modes
  - ➔ Source terms

    e.g. magnetic fields

$$\Delta^{(1)}(\vec{k},\tau) = T^{(1)}(k,\tau)\Phi(\vec{k})$$

$$\Delta^{(2)}(\vec{k},\tau) = \int\int d\vec{k}_1 d\vec{k}_2 \delta^3(\vec{k}-\vec{k}_1-\vec{k}_2)\, T^{(2)}(k,\vec{k}_1,\vec{k}_2,\tau)\Phi(\vec{k}_1)\Phi(\vec{k}_2)$$

Second-Order Boltzmann Codes

Why do we need second-order perturbations?

- ~~Higher precision~~
- New physics
  - → Non-Gaussianity
  - → Mode coupling
  - → Scalar-Vector-Tensor coupling

    e.g. B-modes
  - → Source terms

    e.g. magnetic fields

$$\Delta^{(1)}(\vec{k},\tau) = T^{(1)}(k,\tau)\Phi(\vec{k})$$

$$\Delta^{(2)}(\vec{k},\tau) = \int\int d\vec{k}_1 d\vec{k}_2 \delta^3(\vec{k}-\vec{k}_1-\vec{k}_2)\, T^{(2)}(k,\vec{k}_1,\vec{k}_2,\tau)\Phi(\vec{k}_1)\Phi(\vec{k}_2)$$

A Brief History

Non-linear Boltzmann codes developed to predict the intrinsic bispectrum for the PLANCK satellite mission

History of second-order codes

- CMBQUICK in mathematica (Pitrou 2011)
  First code, but relatively slow. Takes weeks to perform calculation to sufficient accuracy
- COSMOLIB in C++ (Huang 2012)
  Significantly faster (several hours for the temperature bispectrum)
- SONG in C (Pettinari/Fidler 2013)
  Based on CLASS. Includes polarisation, magnetic fields, ...
- PRIVATE CODE in C++ (Su 2013)

The codes have been carefully crosschecked

Key Features of SONG

- Based on CLASS
  - ➔ Liner part of calculation performed by CLASS
  - ➔ Non-linear part inspired by CLASS
  - ➔ Modular, self-contained and flexible
- Easy to learn
  - ➔ More than 10.000 lines of comments
  - ➔ Structure of second-order modules as close as possible to the linear ones of CLASS
- Fast and efficient
  - ➔ OpenMP parallelisation
  - ➔ Uses novel numerical methods for Bessel integrations in the bispectrum
  - ➔ Takes 8 hours to compute the second-order bispectrum on a single core

If you know CLASS, you already know a lot of SONG!

SONG Contains CLASS

SONG comes with a modified version of CLASS that

- computes all background and linear quantities
- stores the quadratic sources needed for the non-linear Boltzmann equation

SONG adds two new modules to it's version of CLASS:

- A new module computing (linear) primordial bispectra (bispectra.c)
- A new module computing Fisher matrices (fisher.c)

These are required to compute the impact of the intrinsic bispectrum when measuring various primordial templates.
But they can also be used independant of SONG to compute primordial bispectra of non-separable primordial kernels.

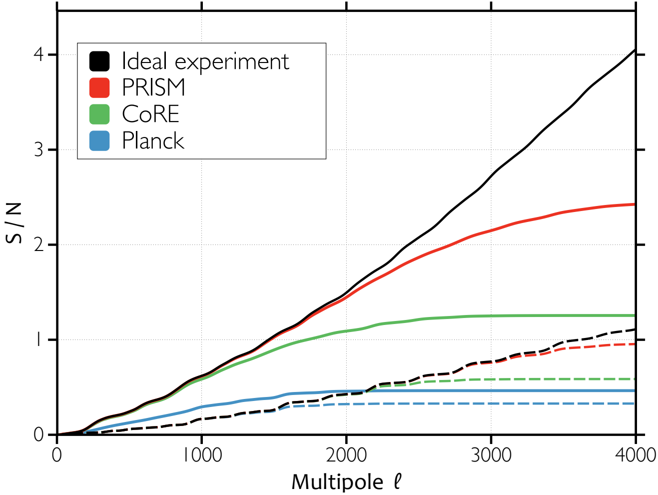
SONG Includes ...

Second-order modules

- Second-order version of most class modules (e.g. perturbations2.c, transfer2.c)
- *'Running class with two extra loops'* (k, k_1, k_2)
- Quadratic sources complicate structure
- Geometric mode-couplings
- Bispectrum integration in bispectra2.c

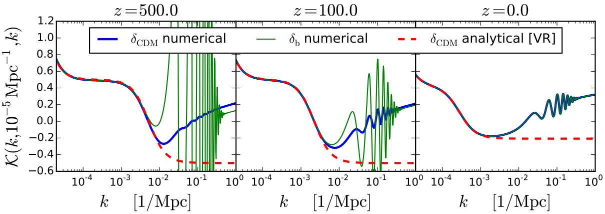
$$\Delta^{(2)}(\vec{k},\tau) = \int\int d\vec{k}_1\, d\vec{k}_2 \delta^3(\vec{k}-\vec{k}_1-\vec{k}_2)\, T^{(2)}(k,\vec{k}_1,\vec{k}_2,\tau)\Phi(\vec{k}_1)\Phi(\vec{k}_2)$$

SONG Computes Non-Gaussianity

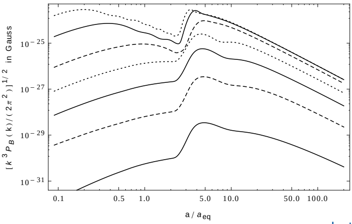


SONG Computes ...

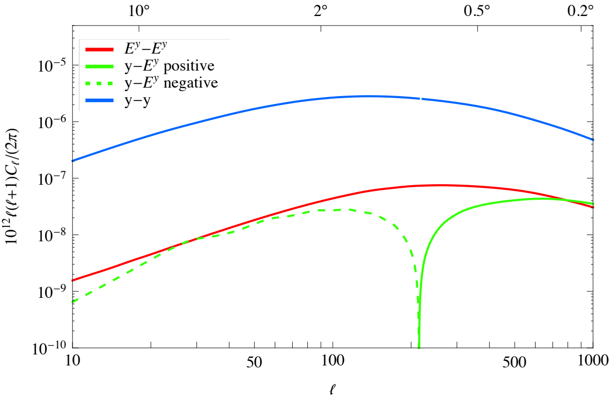
SONG can compute matter perturbations beyond linear perturbation theory



SONG can solve other perturbations such as magnetic fields



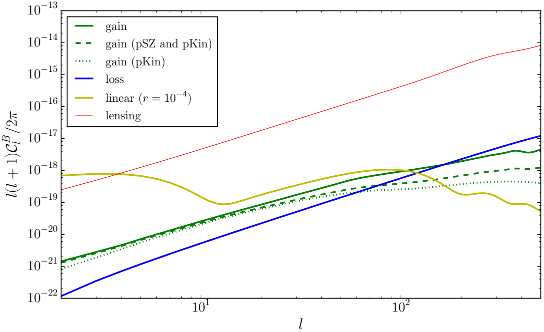
SONG Computes ...



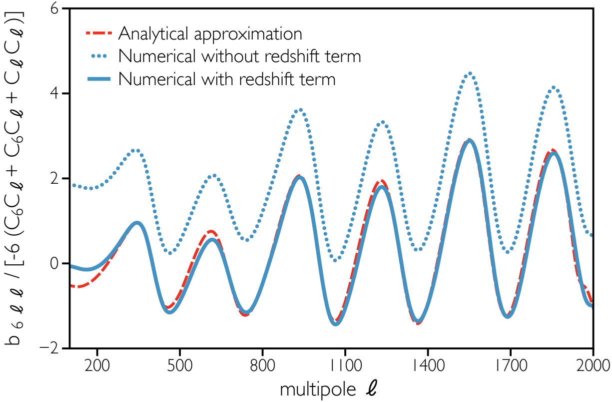
SONG Computes Late-Time Effects

Late-time sources are tough at second-order. Use alternative approaches:

- Lensing: expansion in deflection angles
- Redshift: transformation of variables
- Reionisation: blurring + low-l residuals



How Accurate Is SONG



SONG does not compute ...

curvature, massive neutrinos, non-standard interactions, ...

Installing SONG

Installation *should* be simple if CLASS is already installed and running.
The simplest option is to download directly from
https://github.com/coccoinomane/song by:

```
> git clone --recursive https://github.com/coccoinomane/song.git
> cd song
> make song
```

Then test the code by

```
> ./song ini/intrinsic.ini pre/quick_song_run.pre
```

which should take less than a few minutes and create output ...

```
Running CLASS version v2.4.3
Running SONG version v1.0-beta3
Computing background
 -> age = 13.813434 Gyr
 -> conformal age = 14210.480724 Mpc
```

More information: https://arxiv.org/abs/1405.2280

The SONG-Directory

In your SONG-directory you should find

```
source/        // PHYSICS: the modules of SONG
main/          // main function song.c, calls the modules
include/       // all the headers
pre/           // various precision files
ini/           // various ini files
class.git/     // contains a modified version of CLASS
python/        // python wrapper for SONG
output/        // stores the requested output files
build/         // contains the compiled objects
tools/         // contains mathematical tools
test/          // contains routines used to test the code
scripts/       // some shell scripts
README.md      // very important README file
MAKEFILE       // The MAKEFILE
```

Structure of SONG

In addition to the CLASS modules SONG executes

```
input2.c           // Reads SONG specific parameters
perturbations2.c   // Evolves the second-order transfer functions
bessel2.c          // Computes geometric factors for the
                   // line-of-sight integration
transfer2.c        // Solves the second-order
                   // line-of-sight integration
class/bispectra.c  // Computes linear primordial bispectra
bispectra2.c       // Computes the second-order
                   // intrinsic bispectrum
spectra2.c         // Computes second-order powerspectra
class/fisher.c     // Computes the Fisher matrix
```

Conventions

- Each module has at least an init and a free function
- Models that have a CLASS equivalent end with a '2' (also within the code e.g. pt2 instead of pt)

Input Files

SONG is called with two input files:

```
> ./song initialisation.ini precision.pre
```

- initialisation.ini tells the code what to compute
- precision.pre tells the code how accurate to compute
  - ➔ This distinction is very useful when the code takes hours!
- The .ini and .pre files contain both SONG and CLASS parameters
- Many inputs are shared (e.g. cosmological parameters)
- Use ini/intrinsic.ini and pre/quick_song_run.pre as documentation
- Many useful templates predefined
- Unspecified parameter are set to a default value

SONG *.ini Files

Fixing the cosmology, as done in CLASS:

```
## Planck+WP+highL+BAO - Planck paper XVI, 2013
h = 0.6777
T_cmb = 2.7255
omega_b = 0.022161
omega_cdm = 0.11889
N_eff = 3.04
Omega_k = 0.
reio_parametrization = reio_none
tau_reio = 0.0952
k_pivot = 0.05
A_s = 2.2138e-9
n_s = 0.9611
YHe = 0.2477055
```

SONG *.ini Files

What should SONG compute?

```
output =
  tBisp, eBisp         // temperature and polarised bispectra
  tCl2, eCL2           // second-order power-spectra
  delta_cdm_bk         // matter bispectrum
  delta_cdm_pk         // matter power-spectrum
  magnetic_pk          // magnetic field power-spectrum
  fisher               // Fisher matrix
bispectrum_types =
  intrinsic            // second-order bispectrum
  local, equilateral   // primordial shapes
modes_song = 0,1,2,3    // compute only scalars or also vectors,
    tensors, ...
```

How much output do we want?

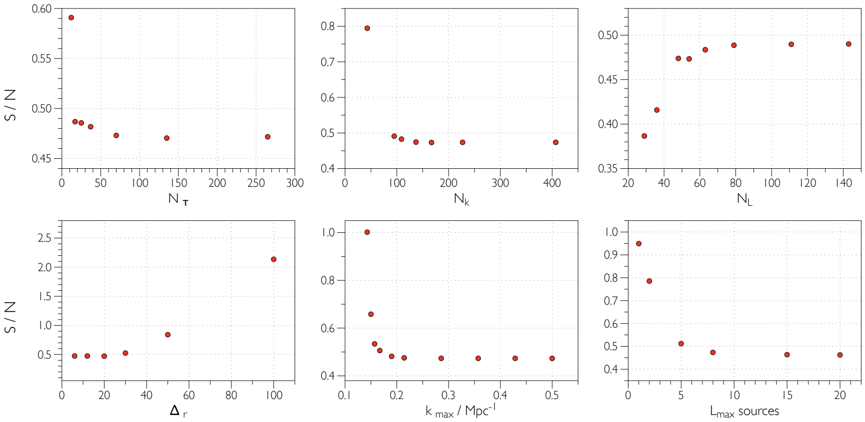
```
perturbations_verbose = 1  // verbose-level CLASS
perturbations2_verbose = 2 // verbose-level SONG
```

SONG *.ini Files

Sources in the second-order systems (be careful!)

```
//Sources for second-order evolution
quadratic_collision = yes
quadratic_liouville = yes
include_time_delay_in_liouville = yes
include_redshift_in_liouville = yes
include_lensing_in_liouville = yes
// Scattering line-of-sight sources at second order
include_pure_scattering_song = yes
include_quad_scattering_song = yes
use_delta_tilde_in_los = yes
// Metric (Liouville) line-of-sight sources at second order
include_pure_metric_song = no
include_quad_metric_song = no
include_time_delay_song = no
include_redshift_in_song = no
include_lensing_in_song = no
include_sachs_wolfe_song = yes
include_integrated_sachs_wolfe_song = yes
only_early_isw = yes
```

SONG Precision Parameters



SONG *.pre Files

- Identical parameters for SONG are marked by a _song
- Some CLASS parameters are overwritten by SONG

```
// k_1 and k_2-sampling
k_min_tau0_song = 0.1
k_max_tau0_over_l_max_song = 2
k_step_sub_song = 0.1
k_step_super_song = 0.025
k_logstep_super_song = 1.8
k_max_for_pk = 0.1
// k_3 sampling
k3_size_min = 5

// time sampling quadratic sources
perturb_sampling_stepsize_for_quadsources = 0.03
// time sampling line-of-sight
perturb_sampling_stepsize_song = 0.4
```

SONG *.pre Files

```
// Maximum multipole
l_max_scalars = 100

// Multipole cut in Boltzmann Hierarchy
l_max_g_song = 12
l_max_ur_song = 12
l_max_pol_g_song = 12

l_max_g_quadsources = -1
l_max_ur_quadsources = -1
l_max_pol_g_quadsources = -1

// Multipole cut in line-of-sight integration
l_max_los_t = 5
l_max_los_p = 5

l_max_los_quadratic_t = 5
l_max_los_quadratic_p = 5
```

SONG Output

A typical SONG console output:

```
Fisher matrix for l_max = 100:
  local      (   0.000384991   -1.03924e-05   -0.000385105 )
  equilateral (  -1.03924e-05    3.71746e-05   -2.35203e-05 )
  intrinsic  (  -0.000385105   -2.35203e-05    0.000722315 )
Correlation matrix for l_max = 100:
  local      (             1     -0.0868697      -0.730282 )
  equilateral (    -0.0868697              1      -0.143534 )
  intrinsic  (     -0.730282      -0.143534              1 )
fnl matrix (diagonal: 1/sqrt(F_ii), upper: F_12/F_11, lower: F_12
    /F_22):
  local      (       50.9653     -0.0269939        -1.0003 )
  equilateral (     -0.279557        164.012      -0.632697 )
  intrinsic  (     -0.533154     -0.0325623         37.208 )
```

THANK YOU FOR YOUR ATTENTION!

Installing SONG

Installation *should* be simple if CLASS is already installed and running.
The simplest option is to download directly from
https://github.com/coccoinomane/song by:

```
> git clone --recursive https://github.com/coccoinomane/song.git
> cd song
> make song
```

Then test the code by

```
> ./song ini/intrinsic.ini pre/quick_song_run.pre
```

which should take less than a few minutes and create output ...

```
Running CLASS version v2.4.3
Running SONG version v1.0-beta3
Computing background
 -> age = 13.813434 Gyr
 -> conformal age = 14210.480724 Mpc
```

More information: https://arxiv.org/abs/1405.2280