CosmoSIS Webinar 2
Advanced sampling and creating new cosmological pipelines

Presenting for the CosmoSIS team:
   Elise Jennings  (Fermilab, KICP)
   Joe Zuntz  (University of Manchester)
   Vinicius Miranda  (University of Pennsylvania)
Currently in cosmology we have a few challenges:

- Many, sometimes correlated, **observables**:
  - CMB, lensing, galaxy clustering, supernovae, clusters…..

- **Different theoretical models**:
  - e.g. Supernovae light curve fitters, bias models for galaxy clustering

- **Different parameters, systematics** in each model:
  - how to sample over each in an MCMC chain?

- Complicated, possibly multimodal, **Posterior/ Likelihoods**:
  - sampling choice may impact results, estimate/model covariances

- **Large collaborations** (hundreds of people e.g. (DES & Planck)):
  - how to track contributions, ensure reproducibility & consistency
  - how to use wealth of existing code/data without wasting PhD deciphering it all, learning new coding language…
CosmoSIS was designed to address each of these issues!

CosmoSIS is a new cosmology parameter estimation code with a focus on *modularity*

- Open source code which community actively contributes to
- Multi-language modules: Python, C++, C, Fortran
- Choice of physics & likelihood modules
- Collection of samplers - mostly in python
- Nice python plotting functions
Overview

- Advanced samplers - presented by Elise Jennings
- CosmoSIS Overview
- Writing parameter files
- Storing data in CosmoSIS DataBlocks
- CosmoSIS Modules
- Modifying Existing Modules
- Sharing, documenting, contributing, and credit
Sampling beyond Metropolis-Hastings…

Speeding up time to convergence always a goal of any sampler, esp important with increasing number of parameters

• Some issues/stumbling blocks with standard MCMC:
  • Initial distribution of points
  • choice of proposal distribution
  • parallelizing stepping algorithm
  • getting Bayesian Evidence

Large impact on time to convergence
Some interesting features of alternative samplers

- Nested sampling (Skilling 2004) -> MultiNest
  http://ccpforge.cse.rl.ac.uk/gf/project/multinest/
- Ensemble sampling -> e.g. in emcee  http://dan.iel.fm/emcee/current/
- Clustering algorithms -> Kombine
- Parallel tempering for initial distribution
- Hamiltonian Monte Carlo
- Population Monte Carlo PMC
- Adaptive MCMC -> e.g. in PyMC  https://github.com/pymc-devs/pymc
- Fast/Slow sampling -> e.g. in CosmoMC
- Snake, Minuit, Maxlike, grid
- Approximate Bayesian Computation, ABC
- Ensemble sampler, uses “walkers” to probe parameter space

- See also the Kombine sampler in CosmoSIS
• Multiple peaks in the posterior identified and isolated

• iso-likelihood contours approximated by cov matrix of active points
Overview

- Advanced samplers
- **CosmoSIS Overview** - presented by Joe Zuntz
- Writing parameter files
- Storing data in CosmoSIS DataBlocks
- CosmoSIS Modules
- Modifying Existing Modules
- Sharing, documenting, contributing, and credit
You run the CosmoSIS command on a *parameter file*:

```bash
> cosmosis demos/demol1.ini
```

which describes various aspects of your analysis:

**Modules:**

- CosmoSIS calculations/likelihoods split into steps called *modules*.
- Run as a sequence, each taking inputs from previous modules and providing new outputs for later ones.
- Each module is given its own [section] in the parameter file:

```ini
[camb]
file = cosmosis-standard-library/boltzmann/camb/camb.so
mode=all
lmax=2500
feedback=2
```
Glossary & Overview

**Pipeline**: The sequence of modules to be run in your analysis for each likelihood

**Values file**: Another file with names and values or ranges of your parameters.

**Sampler**: The code that chooses sets of parameters of which to evaluate the likelihood choosing them using MCMC or some other scheme.

**Test Sampler**: The most trivial “sampler”, just runs one likelihood of a single set of parameters.
Glossary & Overview

**Data Block**: CosmoSIS mechanism for passing data between modules (more later)

**Repositories**: A way of storing, tracking, and sharing code. CosmoSIS comes with two “repos”, one for the main code and one for the modules (cosmosis-standard-library)
Overview

- Advanced samplers
- CosmoSIS Overview
- **Writing parameter files** - presented by Elise Jennings
- Storing data in CosmoSIS DataBlocks
- CosmoSIS Modules
- Modifying Existing Modules
- Sharing, documenting, contributing, and credit
Running CosmoSIS: Demo 5

Last time we showed this diagram of a likelihood pipeline for supernova data.

Now we will look at demo 5 which shows how to run this in CosmoSIS.

The file demos/demo5.ini is the parameter file
Choosing a sampler

```ini
runtime

; The emcee sampler, which uses the Goodman & Weare algorithm
sampler = emcee

[emcee]

; The emcee sampler uses the concept of walkers, a collection
; of live points. Sampling is done along lines that connect
; pairs of walkers. The number of walkers must be at least
; 2*#parameters + 1, but in general more than that usually works
; better.
walkers = 64
; This many samples is overkill, just to make the plots
; look a lot nicer
samples = 400
; This is the interval at which convergence diagnostics
; are performed
nsteps = 100

[output]

; This many samples is overkill, just to make the plots
; look a lot nicer
filename = demo5.txt
format = text
verbosity = debug

[pipeline]

; We use two likelihoods, the JLA (for high redshift) and
; Riess 2011 to anchor H0, which is otherwise degenerate
; with the nuisance parameter M
modules = consistency camb jla riess11
values = demos/values5.ini
extra_output =
likelhoods = jla riess
quiet = T
debug = F
timing = F
```

Setting parameters for this sampler

Choosing the output file

The values file

The module list

Expected likelihoods
[camb]
; For background-only data we do not need a full
; Boltzmann evaluation, just D(z), etc.
; Setting mode=background means we get this.
file = cosmosis-standard-library/boltzmann/camb/camb.so
mode=background
feedback=0

[jla]
; JLA needs quite a lot of parameters telling it where
; data files are ...
file = cosmosis-standard-library supernovae/jla_v3/jla.so
data_dir = cosmosis-standard-library supernovae/jla_v3/data
data_file = jla_lparams.txt
scriptmcut = 10.0
mag_covmat_file = jla_v0covmatrix.dat
stretch_covmat_file = jla_va_covmatrix.dat
colour_covmat_file = jla_vbcovmatrix.dat
mag_stretch_covmat_file = jla_v0acovmatrix.dat
mag_colour_covmat_file = jla_v0b_covmatrix.dat
stretch_colour_covmat_file = jla_vabcovmatrix.dat

; The Riess 11 likelihood anchors \( \Omega \) for us
[riess11]
file = cosmosis-standard-library/likelihood/riess11/riess11.py

; The consistency module translates between our chosen parameterization
; and any other that modules in the pipeline may want (e.g. camb)
[consistency]
file = cosmosis-standard-library/utility/utility/consistency/consistency_interface.py
Values File

Section (category) of parameters

Fixed parameter (single value only)

Varied parameter (min, start, max)

```plaintext
[cosmological_parameters]
omega_m = 0.2  0.2935148566413599  0.4
h0 = 0.5  0.73800089258319199  1.0
w = -1.0
omega_b = 0.04
omega_k = 0.0

[supernova_params]
deltam = -10.0  0.040281808929275797  10.0
alpha = 0.12  0.13675033800637484  0.16
beta = 2.0  3.102925604093223  10.0
m = -19.0
```
Overview

- Advanced samplers
- CosmoSIS Overview
- Writing parameter files
- **Storing data in CosmoSIS DataBlocks** - presented by Vinicius Miranda
- CosmoSIS Modules
- Modifying Existing Modules
- Sharing, documenting, contributing, and credit
Problem posed: how can different packages, written in a variety of languages, communicate with each other?

- Dark days before CosmoSiS: brute force approach was to use files and bash scripts to manage input/output. **This is very prone to error/bugs.**
- CosmoSiS has interface that accepts booleans, integers, doubles, strings, 1D arrays and 2D arrays. Implemented interface in C, C++, Python, Fortran.
- Section names avoid the problem of name clashing
DataBlocks: A cross-language key-value store

Get, Put and Replace data are the most common operations

```python
block["section_name", "value_name"] = x
x = block["section_name", "value_name"]
```

Additional useful functions

```python
block.has_section("section_name")
block.has_value("section_name", "value_name")
block.sections()
block.keys()
```

Check the wiki for instructions in all languages!
https://bitbucket.org/joezuntz/cosmosis/wiki/creating_modules
DataBlocks: Predefined sections

Go to the Wiki to check the names of predefined sections

Wiki

cosmosis / default_sections

All of these sections are predefined strings that you can use in CosmoSIS modules. They are typically used as the names of "sections" - groups of parameters and data collected together.

For an entry on this list called "name", in code you would use these pre-defined constants for the strings:

Python: cosmosis.names.name
C: NAME_SECTION
C++: NAME_SECTION
Fortran: name_section

Likelihoods

- likelihoods

Input parameters

- cosmological_parameters
- halo_model_parameters
- intrinsic_alignment_parameters
- baryon_parameters
- shear_calibration_parameters
- number_density_params
Demo 5 should now be complete

Let’s look at the output...
Overview

- Advanced samplers
- CosmoSIS Overview
- Writing parameter files
- Storing data in CosmoSIS DataBlocks
- **CosmoSIS Modules** - presented by Joe Zuntz
- Modifying Existing Modules
- Sharing, documenting, contributing, and credit
CosmoSIS Modules

- Adding new physics or likelihoods to CosmoSIS = modifying or creating modules

- Modules: python files or C/C++/Fortran code compiled into a shared library
  - Using the -shared compiler flag

- Have three special functions (incl. one optional):
  - setup - called once at the start; reads parameters from input param file
  - execute - called for each sample; reads inputs from data block
  - cleanup - called once at the end; frees memory/resources (optional)
Setup Functions

Input: “options”: DataBlock read from the input parameter file
Output: One value. Any python object, or in C/C++/Fortran a pointer to any object

Read any inputs you need from the parameter file, load any data files you need, etc.

```python
def setup(options):
    X = options[options_section, "x"]
    # ...
    return {"setting1": X,"setting2": 3.14, "setting3": "potato"}
```
Execute Functions

Inputs: `block`: a DataBlock with input params and outputs of previous module.
        `config`: whatever was returned by the setup function.
Output: `status`: integer, 0 if all went well

Read any inputs you need from the parameter file, load any data files you need, etc.

```python
def execute(block, config):
    X = config['setting1']
    Omega = block['cosmological_parameters', 'Omega_m']
    # ...
    block['new_section', 'new_output'] = 666.66
    return 0
```
Likelihood Modules

- Usually one of the last modules in a pipeline
- Same structure as any other module
- Add name to parameter file option `likelihoods = ...` in param file
- Section `likelihoods` is search for `name_like` by the sampler
- e.g.
  - `block[“likelihoods”, “my_sn_like”] = -chi2/2.0` #python
  - `c_datablock_put_double(block, “likelihoods”, “my_sn_like”, -chi2/2.0)` //C

**Diagram:**

- Calculate Likelihood of observed supernova mu, z values given SN data
- Output Likelihood
- Evaluated Likelihood
Likelihood Module Example

Example: WiggleZ BAO module

```python
def setup(options):
    #Load the data from the default location unless otherwise specified
    verbose = options.get_bool(option_section, "verbose", False)

    (...,)

    #Return data for later
    return (z, Dv, weight_matrix, rs_fiducial, verbose)

def execute(block, config):
    #Unpack data loaded in depending on options
    z_data, dv_data, weight_matrix, rs_fiducial, verbose = config

    z = block[names.distances, "z"]
    da = block[names.distances, "D_A"]

    (...,)

    like = -0.5*np.einsum('i,ij,j', delta, weight_matrix, delta)
    block[names.likelihoods, "wigglez_bao_like"] = like

    #Signal success
    return 0

def cleanup(config):
    pass
```

CosmoSIS Interface:

setup, execute, cleanup

DataBlock interface has two arguments

1) Section Name
2) Parameter Name

Using predefined names.likelihoods here
Setup & Execute Functions

Full details by language:
https://bitbucket.org/joezuntz/cosmosis/wiki/creating_modules
Overview

- Advanced samplers
- CosmoSIS Overview
- Writing parameter files
- Storing data in CosmoSIS DataBlocks
- CosmoSIS Modules
- **Modifying Existing Modules** - presented by Vinicius Miranda
- Sharing, documenting, contributing, and credit
# Standard Library Modules

<table>
<thead>
<tr>
<th>CAMB</th>
<th>Colossus</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS</td>
<td>CosmoCalc</td>
</tr>
<tr>
<td>CosmoLike</td>
<td>AstroPy</td>
</tr>
<tr>
<td>SNANA</td>
<td>CosmoMC</td>
</tr>
<tr>
<td>MGCAMB</td>
<td>Cosmolopy</td>
</tr>
<tr>
<td>MGCLASS</td>
<td></td>
</tr>
<tr>
<td>IsItGR</td>
<td>Documentation of standard library: <a href="https://bitbucket.org/joezuntz/cosmosis/wiki/default_modules">https://bitbucket.org/joezuntz/cosmosis/wiki/default_modules</a></td>
</tr>
<tr>
<td>EFTCamb</td>
<td></td>
</tr>
</tbody>
</table>
Modifying existing modules

Two parts:

- Modify physics/calculation itself
- Modify interface with CosmoSIS
Advanced example: modifying CAMB

\[ w(z) = \sum_{i=0}^{4} w_i [\ln(1 + z)]^i \]

- First we need to change CAMB (file: equations.f90)
- Then we need to change the interface between CAMB and the datablock (file: camb_interface.F90)

```fortran
status = status + datablock_get_double_default(block, cosmo, "w", -1.0D0, w_lam)
status = status + datablock_get_double_default(block, cosmo, "wa", 0.0D0, wa_ppf)
```
- Update the ini file (see values1.ini associated with demo 1)

```plaintext
w = -1.0 ; equation of state of dark energy
wa = 0.0 ; equation of state of dark energy (redshift dependency)
```
Overview

- Advanced samplers
- CosmoSIS Overview
- Writing parameter files
- Storing data in CosmoSIS DataBlocks
- CosmoSIS Modules
- Modifying Existing Modules
- **Sharing, documenting, contributing, and credit** - presented by Elise Jennings
How to get credit?

How to Avoid the *Package too big for individual recognition* problem?

Example: The Astropy Problem - arXiv:1610.03159

- In the CosmoSIS module - add suggested citation in the yaml file.
- One possible solution: write code that performs very well (better than any other software in the market) so whenever people need the functionality your code provides, they will visit your github/website and credit your paper!
- Write code that people can incorporate in their pipeline easily with flexible options.

*CosmoSIS is not a monolithic code - you can create a separate module that people can download from your github and incorporate in their pipeline!*
Getting more information

Wiki

- Modules reference
- General reference

Issues page

- Existing issues
- Creating new issues

Any questions?