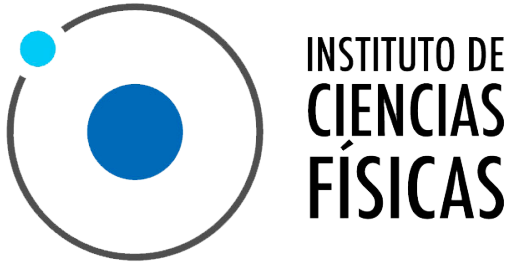


A short introduction to:*MontePython*

Estimación de parámetros cosmológicos

Gabriela García Arroyo (arroyo@icf.unam.mx)
ICF-UNAM

Contents

1 Referencias

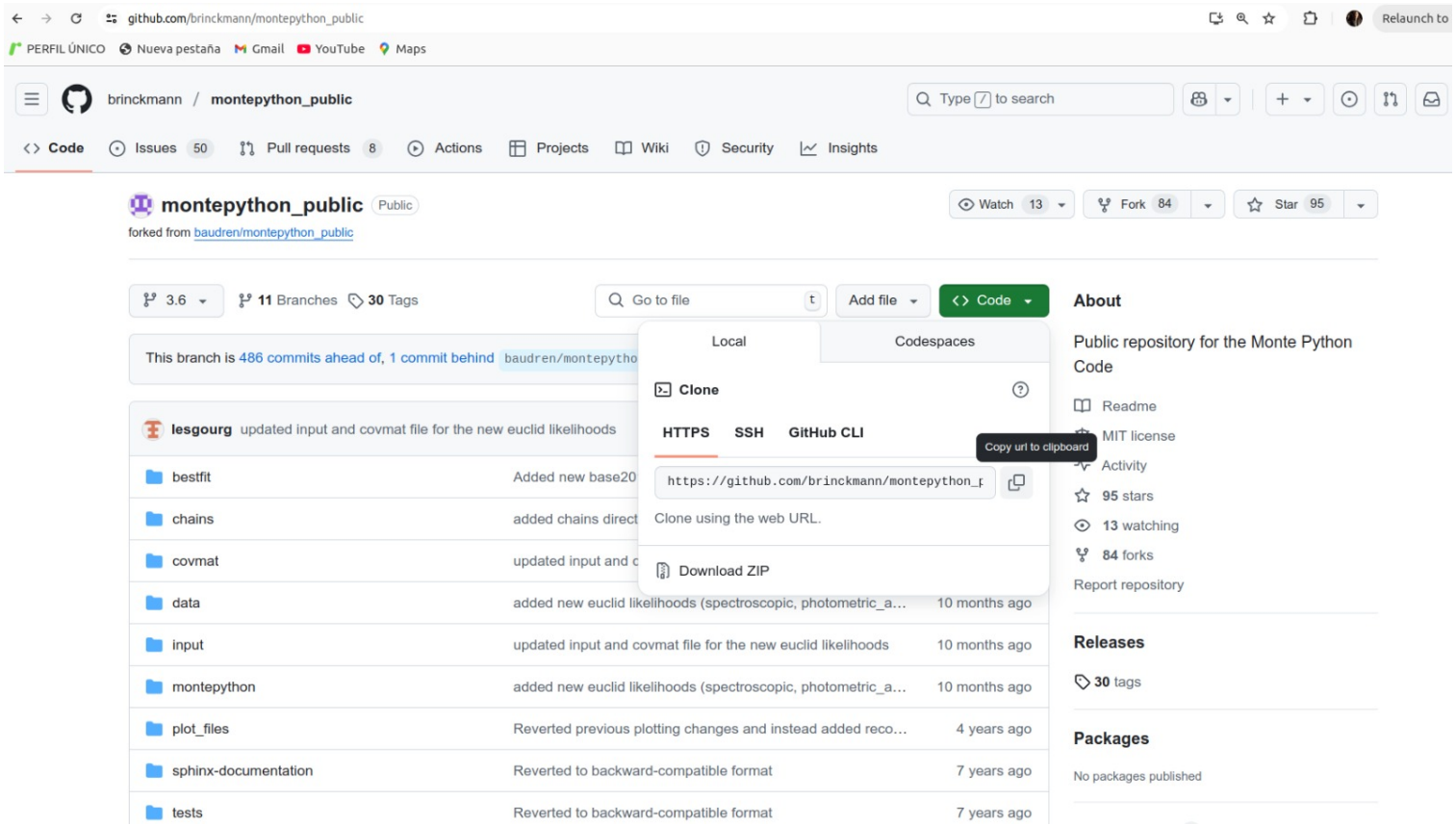
Referencias principales

https://github.com/brinckmann/montepython_public

https://baudren.github.io/montepython.html

Descargar

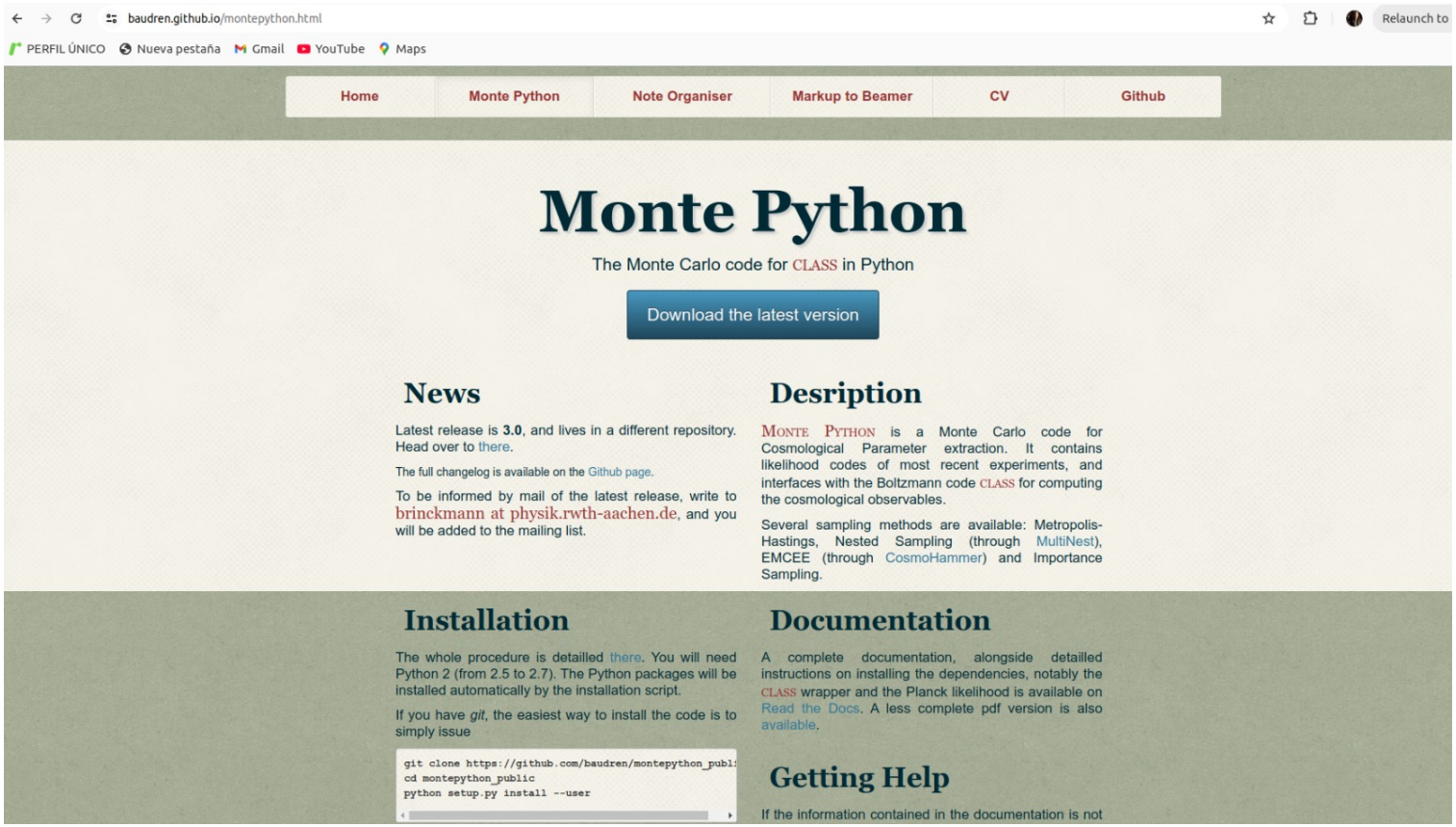
`https://github.com/brinckmann/montepython_public`



- $ git clone .....

Documentación

Versiones anteriores, referenicias a papers, cursos, etc...



`https://baudren.github.io/montepython.html`

Instalación

Después de descargar

- La parte de class ya la tenemos. (si les da problemas vayan desde la terminal a class_public/python] $ python setup.py -build)
- Los datos de Planck están fuera de este curso.
- Vamos a copiar el archivo: montepython_public/default.conf.template a default.conf
- Allí debemos especificar la ruta a class_public en: path['cosmo'] = 'PathToYour/class_public', en mi caso (prefiero no usar root):

```
24 path['cosmo']          = '/home/gaby/Projects/Cosmo/class_public'
25 path['clik']           = '/home/gaby/Projects/code/plc_3.0/plc-3.01
```

Ya podemos hacer una prueba.

param

Archivo de confifuguración que contiene los parámetros cosmológicos, sus priors y los likelihoods a utilizar, los ejemplos se encuentran en montepython_public/input

```
#------Experiments to test (separated with commas)-----

# valid options are (so far): acbar, bicep, boomerang, cbi,
# hst, quad, sn, spt, fake_planck_bluebook,
# euclid_pk, euclid_lensing
# and clik_fake_planck (for Planck members)

data.experiments=['JLA']

#------ Parameter list -------
# data.parameters[class name] = [mean, min, max, 1-sigma, scale, role]
# - if min max irrelevant, put to -1 or None (if you want a boundary of -1, use -1.0)
# - if fixed, put 1-sigma to 0
# - if scale irrelevant, put to 1, otherwise to the appropriate factor
# - role is either 'cosmo', 'nuisance' or 'derived'


# Cosmological parameters list
data.parameters['Omega_cdm'] = [0.2562,  None, None, 0.008,  1, 'cosmo']

# Nuisance
data.parameters['alpha']   = [0.15 ,  None, None, 0.001, 1, 'nuisance']
data.parameters['beta']    = [3.559,  None, None, 0.02,  1, 'nuisance']
data.parameters['M']       = [-19.02, None, None, 0.004,  1, 'nuisance']
data.parameters['Delta_M'] = [-0.10,  None, None, 0.004,  1, 'nuisance']

# Derived parameter list
data.parameters['Omega_m'] = [0,    -1, -1, 0,1,  'derived']


data.cosmo_arguments['Omega_b'] = 0.05
#------ Mcmc parameters ----
# Number of steps taken, by default (overwritten by the -N command)
data.N=10
# Number of accepted steps before writing to file the chain. Larger means less
# access to disc, but this is not so much time consuming.
data.write_step=5
```

Algunos comentarios y ejemplos sobre .param

- Los nombres válidos para los likelihoods los encuentras en montepython_public/montepython/likelihoods
  data.experiments=['hst', 'Pantheon']
- Parámetros cosmológicos a variar, estos 'definen' tu cosmología:
  data.parameters['omega_b'] = [2.249, 1.8, 3, 0.016, 0.01, 'cosmo']
- Nuisance parameters dependen de tu conjunto de datos, revisa si es que tiene:
  data.parameters['M'] = [0.1, -1., 1, 0.01, 1, 'nuisance']
- Para parámetros derivados:
  data.parameters['Omega_Lambda'] = [0, -1, -1, 0,1, 'derived']
- Parámetros que no se varían y nos ayudan a definir la Cosmología:
  data.cosmo_arguments['deg_ncdm'] = 3
  data.cosmo_arguments['fluid_equation_of_state']= 'CLP' En este último caso entonces ya puedes agregar w0 y wa a los parámetros a variar y también puedes cambiar la parametrización´.

Cómo se usa MontePython

Desde la terminal dentro de montepython_public:

- montepython_public]$ python montepython/MontePython.py –help
- montepython_public]$ python montepython/MontePython.py run –help
- montepython_public]$ python montepython/MontePython.py info –help

Con esto vas a conocer las opciones que tienes para correr cadenas (run) y para analizarlas (info).

Una primer corrida básica, para asegurarte que el .param está bien definido puede ser con 10 pasos (-N 10). Probemos input/test.param:

python montepython/MontePython.py run -p input/test.param -o testprueba -N 10

- después de -p indicas la ruta a tu .param
- después de -o la carpeta donde va a guardar las cadenas

Si la ejecución no fue exitosa hay que borrar la carpeta.- Antes de volver a ejecutar.- montepython_public]$ rm -rf testprueba

Recomendaciones -personales-

- Aunque puedes usar mpirun -np 4, recomiendo mejor correr la versión estándar digamos 4 veces para evitar conflictos, (elige la misma carpeta de output)
- Realiza una primer corrida de prueba con algunos 500 pasos y analizala. Recomendable pedirle la matriz de covarianza (–want-covmat) python montepython/MontePython.py info testprueba –want-covmat
- Haz una corrida más 'enserio' y pásale la matriz de covarianza generada python montepython/MontePython.py run -p input/test.param -o testprueba -N 1000 -c testprueba/test.covmat
- Analiza los archivos de salida y las impresiones en la terminal. El comando info genera una triangle plot y una con las posteriors, también te escribe el bestfit, el valor a 1 y 2 sigma así como el valor del Gelman-Rubin (testprueba/test.h_info)
- Decide si tus cadenas ya convergieron, si no vuelve al paso anterior, también le puedes adicionar -b testprueba/test.bestfit esto va a reemplazar el punto donde inicia cada parámetro por el valor del mejor ajuste que llevas hasta este momento.

Graficas

- Es recomendado usar getdist
- Creamos un .ipynb dentro de montepython_public
- Ejemplo básico:

```
import getdist
from getdist import plots, MCSamples
```

```
path_to_chain='../montepython_public/testprueba/2025-04-03_400_'
```

#Convertimos la cadena a sample

```
samples = getdist.loadMCSamples(path_to_chain,
                                settings={'ignore_rows':0.3} #30% burn in, hay más opciones
                               )
```

#Graficamos uno de los parámetros -como ven aún no converge- Después se muestra una triangle plot -ver el tutorial de getdist-

```
g = plots.get_single_plotter(width_inch=3,ratio=1)
g.plot_1d(samples, 'omega_b')
```

```
#Otras cosas útiles: Escribir el G-R
```

```
print('R-1=',samples.getGelmanRubin()) #Como se dan cuenta no han convergido
```

```
R-1= 4.370221670221348
```

```
#Puedes obtener la tabla de parámetros con sus errores:
```

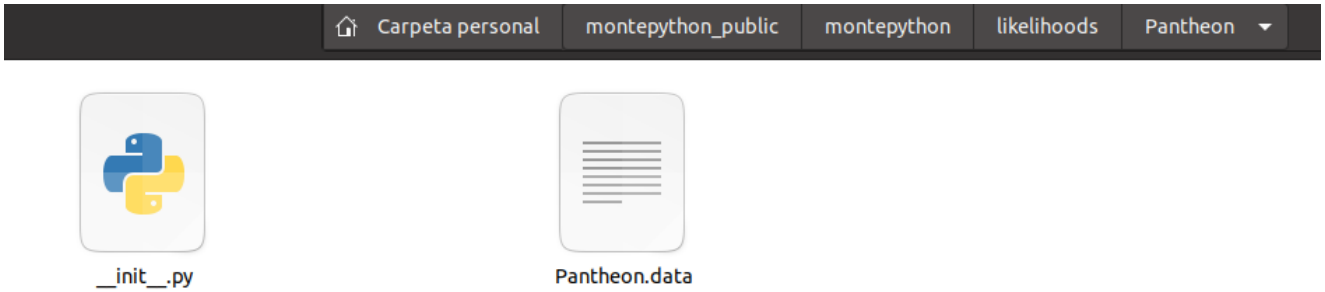
```
print(samples.getTable(limit=1).tableTex())
```

```
\begin{tabular} { l  c}

 Parameter &  68\% limits\\
\hline
```

Likelihoods

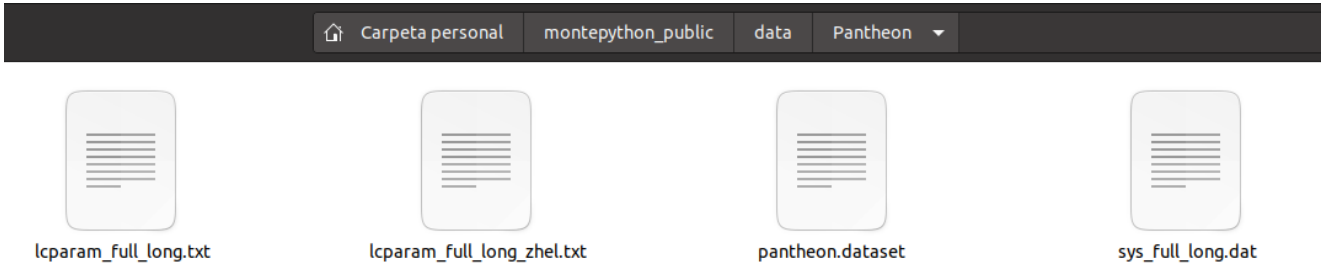
- Las likelihoods disponibles se encuentran en: montepython_public/montepython/likelihoods
- Sus datos correspondientes se encuentran en: montepython_public/data
- Como ejemplo veamos Pantheon



El archivo __init__.py tiene el código del Likelihood la classe se llama Pantheon -este es el nombre que escribimos en el .param-
El archivo Pantheon.data tiene especificaciones sobre el conjunto de datos; conflictos con otros datos, nuisance pars, ruta a los datos.

Likelihoods

En montepython_public/data



Básicamente son las tablas con los datos, covarianzas, etc. Estas tablas son llamadas en el archivo Pantheon.data (previamente mencionado).