Stefano Gariazzo

IFIC, Valencia (ES)
CSIC – Universitat de Valencia

gariazzo@ific.uv.es
http://ific.uv.es/~gariazzo/

CMB: numerical calculations

partially based on J. Lesgourgues courses:

https://lesgourg.github.io/courses.html

Laurea Magistrale in Fisica, Università Federico II, Napoli, 13/12/2019

Why version control?

you want to build a project (e.g. a code): you need backups and to track changes!

Why version control?

you want to build a project (e.g. a code): you need backups and to track changes!

you can store the files/folders: e.g.

```
my_code_191202_1005.zip
my_code_191202_1030.zip
my_code_191203_1450.zip
my_code_191203_1700.zip
```

not practical for big projects:

- a lot of space needed to repeat common parts!
- how to remember what was changed at each "snapshots"?
- how to find out where a bug was introduced?

and what if many people have to edit at the same time?

Why version control?

you want to build a project (e.g. a code): you need backups and to track changes!

you can store the files/folders: e.g.

```
my_code_191202_1005.zip
my_code_191202_1030.zip
my_code_191203_1450.zip
my_code_191203_1700.zip
```

not practical for big projects:

- a lot of space needed to repeat common parts!
- how to remember what was changed at each "snapshots"?
- how to find out where a bug was introduced?

and what if many people have to edit at the same time?

version control:

process of storing multiple versions of a single project, allowing each version to be recalled at a later date

Advantages:

- only changes are saved
- all the snapshots are commented
- can restore any previous step
- easy to work with other people (cannot overwrite each other's changes by chance)
- easy to experiment with new features without messing up everything

Version control – basics

few names:

repository:	where files and historical data are saved (local or remote)
revision:	the state at a point in time of the entire set of files/folders
working copy:	the editable copy of a revision (the current "HEAD")
(to) checkout:	to create a local working copy of a revision
(to) commit:	write the changes made in the working copy to the repository
branch:	you can split the commit history at some point, to have, from that time forward, two independent commit histories
remote:	a repository of the project hosted in an external server/folder
(to) pull/push:	copy revisions from/to a remote repository into the local one
(to) clone:	copy the revisions from another repository to an empty one

Version control with `Git` – basics

`Git` (hard g!): version control system first developed for the linux kernel

git init	initialize an empty repository (creates `.git/` and subfolders)
git clone ...	initialize a repository, copying an existing one
git status	list files that changed between working copy and last revision
git add ...	add files to the list of changes to be committed
git diff	show all the changes (in all files) from the last revision
git commit	write the changes in a new revision
git checkout ...	replace working copy with a different revision
git remote	show list of remote repositories
git pull/push	copy from/to a remote repository

An example with Git

```
$ mkdir my_project && cd my_project    # create and enter folder
```

An example with Git

```
$ mkdir my_project && cd my_project    # create and enter folder
```

```
$ git init    # initialize repository
```

```
Initialised empty Git repository in /home/gariazzo/my_project/.git/
```

An example with Git

```
$ mkdir my_project && cd my_project    # create and enter folder
```

```
$ git init    # initialize repository
```

```
Initialised empty Git repository in /home/gariazzo/my_project/.git/
```

```
$ git status    # show current status
```

```
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

An example with Git

```
$ mkdir my_project && cd my_project    # create and enter folder
```

```
$ git init    # initialize repository
```

```
Initialised empty Git repository in /home/gariazzo/my_project/.git/
```

```
$ git status    # show current status
```

```
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

```
$ echo "abcd" > file.txt    # create a file
```

```
$ git status    # show current status – untracked file!
```

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        file.txt

nothing added to commit but untracked files present (use "git add" to track)
```

An example with Git

```
$ git add file.txt    # add file for the next commit
```

```
$ git status    # show current status – file "to be committed"!
```

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   file.txt
```

An example with Git

```
$ git add file.txt    # add file for the next commit

$ git status    # show current status – file "to be committed"!
```

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   file.txt
```

```
# needed once per computer before committing – define your identity

$ git config --global user.email "you@example.com"

$ git config --global user.name "Your Name"

# "first commit" is a comment!

$ git commit -m "first commit"
```

```
[master (root-commit) 6bff89a] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
```

An example with Git

```
$ echo "new text" >> file.txt    # add text to the file...
```

```
$ git status
```

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   file.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

An example with Git

```
$ echo "new text" >> file.txt    # add text to the file...
```

```
$ git status
```

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   file.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

```
$ git checkout .    # restore the current directory to the last revision
```

```
$ git status    # working tree clean!
```

```
On branch master
nothing to commit, working tree clean
```

```
$ cat file.txt
```

```
abcd
```

An example with Git

```
$ echo "1234" >> file.txt
```

```
$ git diff    # diff shows the changed lines with +/-: new line "+1234"
```

```
diff --git a/file.txt b/file.txt
index acbe86c..6a7cc57 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1,2 @@
 abcd
+1234
```

An example with Git

```
$ echo "1234" >> file.txt
```

```
$ git diff    # diff shows the changed lines with +/-: new line "+1234"
```

```
diff --git a/file.txt b/file.txt
index acbe86c..6a7cc57 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1,2 @@
 abcd
+1234
```

```
$ git commit -am "1234 added"    # -a: consider all changes
```

```
[master b9eb1ee] 1234 added
 1 file changed, 1 insertion(+)
```

An example with Git

```
$ git log --graph    # log lists all the commits, --graph their relation
```

```
* commit b9eb1ee8a2352b9f66ec1aff1ee24928a776ddc4 (HEAD -> master)
| Author: Your Name <you@example.com>
| Date:   Tue Oct 15 17:49:04 2019 +0200
|
|     1234 added
|
* commit 6bff89a319526c06ad1c9429a11075901a95674c
  Author: Your Name <you@example.com>
  Date:   Tue Oct 15 17:46:01 2019 +0200

      first commit
```

2nd commit

1st commit

An example with Git

```
$ git log --graph    # log lists all the commits, --graph their relation
```

```
* commit b9eb1ee8a2352b9f66ec1aff1ee24928a776ddc4 (HEAD -> master)
| Author: Your Name <you@example.com>
| Date:   Tue Oct 15 17:49:04 2019 +0200
|
|     1234 added
|
* commit 6bff89a319526c06ad1c9429a11075901a95674c
  Author: Your Name <you@example.com>
  Date:   Tue Oct 15 17:46:01 2019 +0200

      first commit
```

2nd commit

1st commit

```
$ git checkout 6bff    # restore a specific (the first, here) revision

$ echo "ABCD" >> file.txt
```

An example with Git

```
$ git log --graph    # log lists all the commits, --graph their relation
```

```
* commit b9eb1ee8a2352b9f66ec1aff1ee24928a776ddc4 (HEAD -> master)
| Author: Your Name <you@example.com>
| Date:   Tue Oct 15 17:49:04 2019 +0200
|
|     1234 added
|
* commit 6bff89a319526c06ad1c9429a11075901a95674c
  Author: Your Name <you@example.com>
  Date:   Tue Oct 15 17:46:01 2019 +0200

      first commit
```

2nd commit

1st commit

```
$ git checkout 6bff    # restore a specific (the first, here) revision

$ echo "ABCD" >> file.txt

$ git commit -am "ABCD added"    # what happens with this?
```

```
[detached HEAD b9ce1a7] ABCD added
 1 file changed, 1 insertion(+)
```

An example with Git

```
$ git log --graph --all    # --all to show all branches
```

```
* commit b9ce1a7c94bb2cc14b98a226a7bacf5a73b8d4f8 (HEAD)
| Author: Your Name <you@example.com>
| Date:   Tue Oct 15 17:51:55 2019 +0200
|
|     ABCD added
|
| * commit b9eb1ee8a2352b9f66ec1aff1ee24928a776ddc4 (master)
|/  Author: Your Name <you@example.com>
|   Date:   Tue Oct 15 17:49:04 2019 +0200
|
|       1234 added
|
* commit 6bff89a319526c06ad1c9429a11075901a95674c
  Author: Your Name <you@example.com>
  Date:   Tue Oct 15 17:46:01 2019 +0200

      first commit
```

3rd commit ("ABCD")

2nd commit ("1234")

1st commit ("abcd")

An example with Git

```
$ git log --graph --all    # --all to show all branches
```

```
* commit b9ce1a7c94bb2cc14b98a226a7bacf5a73b8d4f8 (HEAD)
| Author: Your Name <you@example.com>
| Date:   Tue Oct 15 17:51:55 2019 +0200
|
|     ABCD added
|
| * commit b9eb1ee8a2352b9f66ec1aff1ee24928a776ddc4 (master)
|/  Author: Your Name <you@example.com>
|   Date:   Tue Oct 15 17:49:04 2019 +0200
|
|       1234 added
|
* commit 6bff89a319526c06ad1c9429a11075901a95674c
  Author: Your Name <you@example.com>
  Date:   Tue Oct 15 17:46:01 2019 +0200

      first commit
```

3rd commit ("ABCD")

2nd commit ("1234")

1st commit ("abcd")

```
$ git checkout -b "new"
```

```
Switched to a new branch 'new'
```

creates a branch named "new" in "HEAD"

better to having branches than spare commits! more order in the structure

An example with Git

```
$ git log --graph --all
```

```
* commit b9ce1a7c94bb2cc14b98a226a7bacf5a73b8d4f8 (HEAD -> new)
| Author: Your Name <you@example.com>
| Date:   Tue Oct 15 17:51:55 2019 +0200
|
|     ABCD added
|
| * commit b9eb1ee8a2352b9f66ec1aff1ee24928a776ddc4 (master)
|/  Author: Your Name <you@example.com>
|   Date:   Tue Oct 15 17:49:04 2019 +0200
|
|       1234 added
|
* commit 6bff89a319526c06ad1c9429a11075901a95674c
  Author: Your Name <you@example.com>
  Date:   Tue Oct 15 17:46:01 2019 +0200

      first commit
```

branch “new”, current HEAD

branch “master”

root revision

An example with Git

```
$ git log --graph --all
```

```
* commit b9ce1a7c94bb2cc14b98a226a7bacf5a73b8d4f8 (HEAD -> new)
| Author: Your Name <you@example.com>
| Date:   Tue Oct 15 17:51:55 2019 +0200
|
|     ABCD added
|
| * commit b9eb1ee8a2352b9f66ec1aff1ee24928a776ddc4 (master)
|/  Author: Your Name <you@example.com>
|   Date:   Tue Oct 15 17:49:04 2019 +0200
|
|       1234 added
|
* commit 6bff89a319526c06ad1c9429a11075901a95674c
  Author: Your Name <you@example.com>
  Date:   Tue Oct 15 17:46:01 2019 +0200

      first commit
```

branch "new", current HEAD

branch "master"

root revision

```
$ git checkout master && cat file.txt
```

```
Switched to branch 'master'
abcd
1234
```

change to "master" and show file content

An example with Git

```
$ git log --graph --all
```

```
* commit b9ce1a7c94bb2cc14b98a226a7bacf5a73b8d4f8 (HEAD -> new)
| Author: Your Name <you@example.com>
| Date:   Tue Oct 15 17:51:55 2019 +0200
|
|     ABCD added
|
| * commit b9eb1ee8a2352b9f66ec1aff1ee24928a776ddc4 (master)
|/  Author: Your Name <you@example.com>
|   Date:   Tue Oct 15 17:49:04 2019 +0200
|
|       1234 added
|
* commit 6bff89a319526c06ad1c9429a11075901a95674c
  Author: Your Name <you@example.com>
  Date:   Tue Oct 15 17:46:01 2019 +0200

      first commit
```

branch "new", current HEAD

branch "master"

root revision

```
$ git checkout master && cat file.txt
```

```
Switched to branch 'master'
abcd
1234
```

change to "master" and show file content

```
$ git checkout new && cat file.txt
```

```
Switched to branch 'new'
abcd
ABCD
```

file.txt in "new" is different than in "master"!

GitHub

GitHub: largest web-based Git repository hosting service

`https://github.com`

meant for code collaboration with anyone online

public/private repositories

user permissions

GitHub

GitHub: largest web-based Git repository hosting service

`https://github.com`

meant for code collaboration with anyone online

public/private repositories

user permissions

adds extra functionality on top of Git

- bug tracking
- documentation
- automatic tests/deployment
- discussions/feature request
- pull requests
- ...

GitHub

GitHub: largest web-based Git repository hosting service

`https://github.com`

meant for code collaboration with anyone online

public/private repositories

user permissions

adds extra functionality on top of Git

- bug tracking
- documentation
- automatic tests/deployment
- discussions/feature request
- pull requests
- ...

Several other similar services exist! For example:

BitBucket

SourceForge

GitLab

Basic use of the `matplotlib` plotting tools

first, let's install/upgrade some python packages:

```
$ python -m pip install -U --user pip
$ pip install --user -U matplotlib numpy scipy Cython
```

Basic use of the `matplotlib` plotting tools

the python script:

the output:

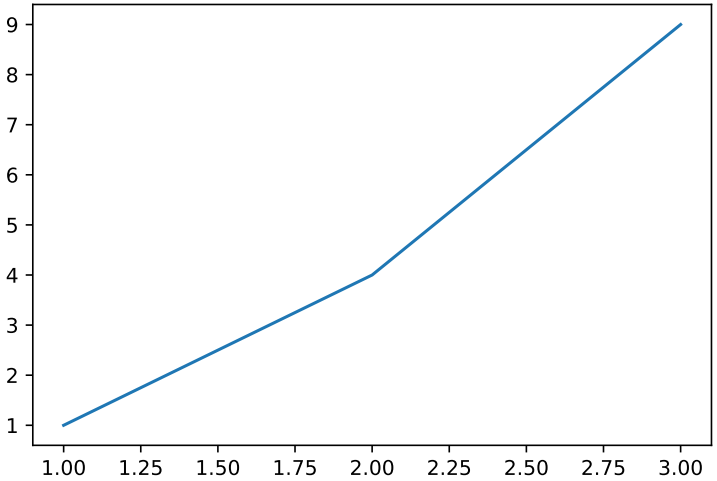
```
# need to import packages to use them
import matplotlib.pyplot as plt

import numpy as np
```

Basic use of the matplotlib plotting tools

the python script:

```
# need to import packages to use them
import matplotlib.pyplot as plt

import numpy as np

# simple plot
plt.plot([1, 2, 3], [1, 4, 9])

plt.show()
```

the output:

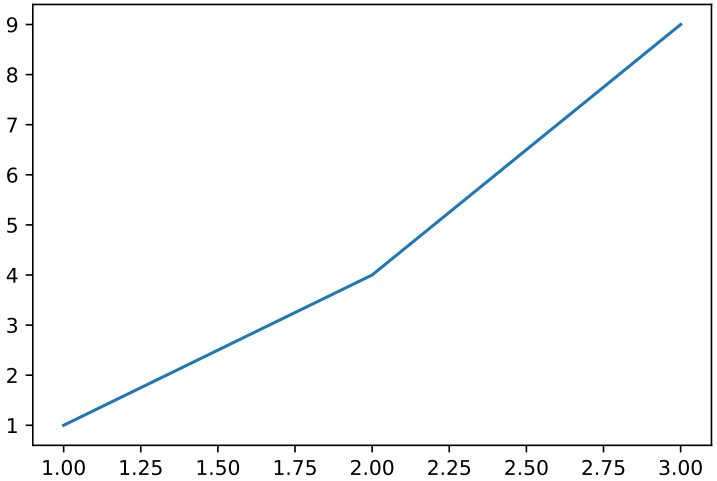


Basic use of the matplotlib plotting tools

the python script:

```
# need to import packages to use them
import matplotlib.pyplot as plt

import numpy as np

# better:
x = np.linspace(1, 3, 3)

# creates an array([1., 2., 3.])

# check also logspace and geomspace!
plt.plot(x, x**2)

plt.show()
```

the output:

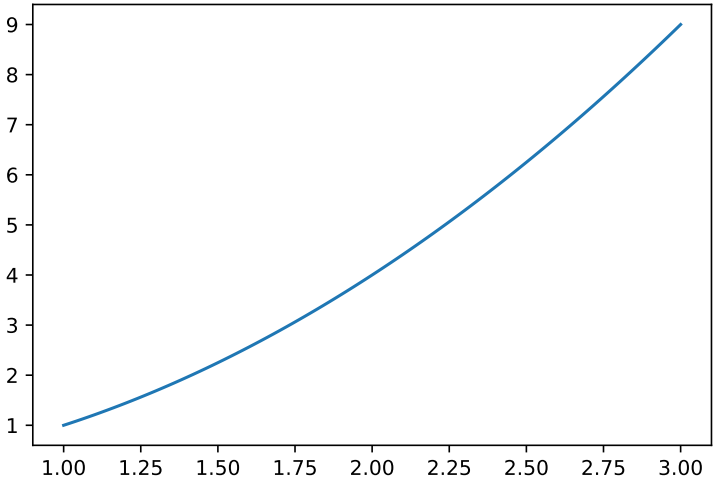


Basic use of the matplotlib plotting tools

the python script:

```
# need to import packages to use them
import matplotlib.pyplot as plt

import numpy as np

# better:
x = np.linspace(1, 3, 101)

# creates an array([1., 1.02, . . . , 3.])

# check also logspace and geomspace!
plt.plot(x, x**2)

plt.show()
```

the output:

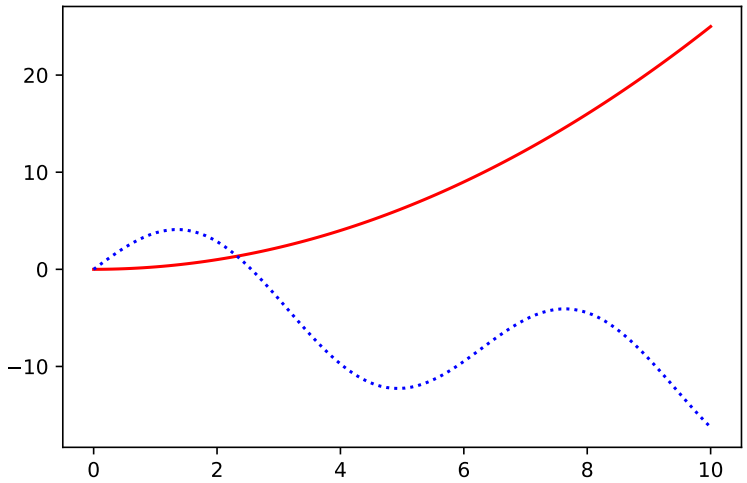


Basic use of the matplotlib plotting tools

the python script:

```
# need to import packages to use them
import matplotlib.pyplot as plt

import numpy as np

# more complex:
x = np.linspace(0, 10, 100)

def f2(x):
 return -1.3*x+6*np.sin(x)

plt.plot(x, 0.25*x**2, "r-") # color is "r"ed, style is "-" (solid)

plt.plot(x, f2(x), "b:") # "b"lue, dotted ":" line

plt.savefig("test.pdf")
```

the output:

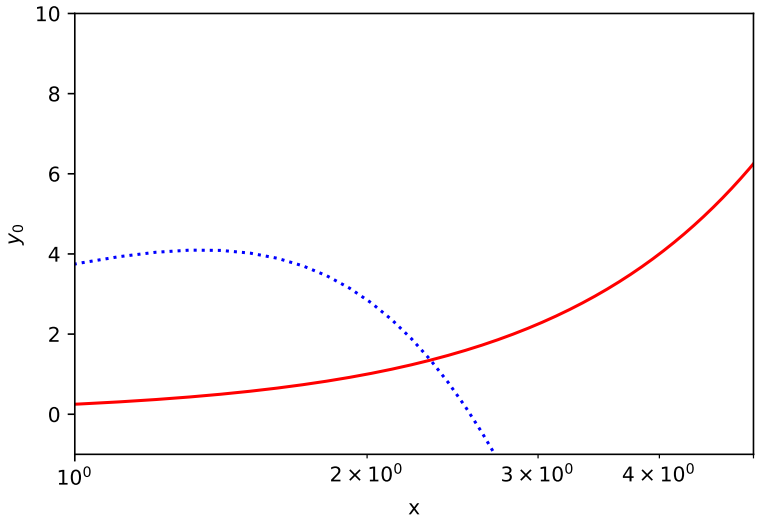


Basic use of the matplotlib plotting tools

the python script:

```
# need to import packages to use them
import matplotlib.pyplot as plt

import numpy as np

# customize plot:
x = np.linspace(0, 10, 100)

plt.plot(x, 0.25*x**2, "r-")

plt.plot(x, -1.3*x+6*np.sin(x), "b:")

plt.xlabel("x")

plt.ylabel("$y_0$") # may use LaTeX syntax!

plt.xlim(1, 3)

plt.ylim(-1, 10)

plt.xscale("log")
```

the output:



On the numpy package – lists and arrays

lists vs numpy arrays: (need "import math;import numpy as np")

lists

arrays

On the numpy package – lists and arrays

lists vs numpy arrays: (need "import math;import numpy as np")

lists

```
x=[1, 2, 3]
x*3
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

list is repeated three times!

arrays

```
x=np.array([1, 2, 3])
x*3
```

```
array([3, 6, 9])
```

On the numpy package – lists and arrays

lists vs numpy arrays: (need "import math;import numpy as np")

lists

```
x=[1, 2, 3]
x*3
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
math.exp(x) # standard functions
```

```
TypeError:  must be real number, not list
```

arrays

```
x=np.array([1, 2, 3])
x*3
```

```
array([3, 6, 9])
```

```
math.exp(x)
```

```
TypeError:  only size-1 arrays can be converted to P
```

only scalar arguments (single numbers, no lists of sort)

On the numpy package – lists and arrays

lists vs numpy arrays: (need "import math;import numpy as np")

lists

```
x=[1, 2, 3]
x*3
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
math.exp(x) # standard functions
```

```
TypeError:  must be real number, not list
```

```
np.exp(x) # numpy equivalent
```

```
array([ 2.71828183, 7.3890561 , 20.08553692])
```

```
# internal conversion to array occurred
```

arrays

```
x=np.array([1, 2, 3])
x*3
```

```
array([3, 6, 9])
```

```
math.exp(x)
```

```
TypeError:  only size-1 arrays can be converted to P
```

```
np.exp(x)
```

```
array([ 2.71828183, 7.3890561 , 20.08553692])
```

On the numpy package – multidimensional arrays

multidimensional lists vs numpy arrays: (need "import numpy as np")

lists

```
li=[[1,2,3,4],[5,6,7,8]]

li[1][2]
```

7

```
# indexes start from 0!

li[1,2]     # won't work with lists!
```

TypeError: list indices must be integers or slices, not tuple

arrays

On the numpy package – multidimensional arrays

multidimensional lists vs numpy arrays: (need "import numpy as np")

lists

```
li=[[1,2,3,4],[5,6,7,8]]

li[1][2]
```

7 # indexes start from 0!

```
li[1,2]    # won't work with lists!
```

TypeError: list indices must be integers or slices, not tuple

arrays

```
ar=np.array([[1,2,3,4],[5,6,7,8

ar[1][2]
```

7

```
ar[1,2] # both syntaxes for arrays
```

7

On the numpy package – multidimensional arrays

multidimensional lists vs numpy arrays: (need "import numpy as np")

lists

```
li=[[1,2,3,4],[5,6,7,8]]

li[1][2]
```

```
7
```

```
# indexes start from 0!

li[1,2]    # won't work with lists!
```

```
TypeError: list indices must be integers or slices, not tuple
```

```
li[0][:]    # slicing
```

```
[1, 2, 3, 4]
```

arrays

```
ar=np.array([[1,2,3,4],[5,6,7,8

ar[1][2]
```

```
7
```

```
ar[1,2] # both syntaxes for arrays
```

```
7
```

```
ar[0,:]
```

```
array([1, 2, 3, 4])
```

On the numpy package – multidimensional arrays

multidimensional lists vs numpy arrays: (need "import numpy as np")

lists

```
li=[[1,2,3,4],[5,6,7,8]]

li[1][2]
```

```
7
```

```
# indexes start from 0!

li[1,2]    # won't work with lists!
```

```
TypeError: list indices must be integers or slices, not tuple
```

```
li[0][:]    # slicing
```

```
[1, 2, 3, 4]
```

```
li[0][1:3]    # more slicing
```

```
[2,3]
```

arrays

```
ar=np.array([[1,2,3,4],[5,6,7,8

ar[1][2]
```

```
7
```

```
ar[1,2] # both syntaxes for arrays
```

```
7
```

```
ar[0,:]
```

```
array([1, 2, 3, 4])
```

```
ar[0,1:3]
```

```
array([2, 3])
```

On the numpy package – loadtxt

loading files as numpy arrays:

(need "import numpy as np; import matplotlib.pyplot as plt")

sample "file.txt":

```
# x y
0 10
1 10.1
2 10.2
3 13
4 12
5 9
6 8.6
7 8.8
9 11
```

On the numpy package – loadtxt

loading files as numpy arrays:

(need "import numpy as np; import matplotlib.pyplot as plt")

```
arr=np.loadtxt("file.txt")
```

```
array([[ 0. , 10. ],
       [ 1. , 10.1],
       [ 2. , 10.2],
       [ 3. , 13. ],
       [ 4. , 12. ],
       [ 5. ,  9. ],
       [ 6. ,  8.6],
       [ 7. ,  8.8],
       [ 9. , 11. ]])
```

comment (first line, starting with #) ignored!

sample "file.txt":

```
# x y
0 10
1 10.1
2 10.2
3 13
4 12
5 9
6 8.6
7 8.8
9 11
```

On the numpy package – loadtxt

loading files as numpy arrays:
(need "import numpy as np; import matplotlib.pyplot as plt")

```
arr=np.loadtxt("file.txt")
```

sample "file.txt":

```
# x y
0 10
1 10.1
2 10.2
3 13
4 12
5 9
6 8.6
7 8.8
9 11
```

comment (first line, starting with #) ignored!

```
array([[ 0. , 10. ],
       [ 1. , 10.1],
       [ 2. , 10.2],
       [ 3. , 13. ],
       [ 4. , 12. ],
       [ 5. ,  9. ],
       [ 6. ,  8.6],
       [ 7. ,  8.8],
       [ 9. , 11. ]])
```

```
arr[:,0]    # get first column: "x"
```

```
array([0., 1., 2., 3., 4., 5., 6., 7., 9.])
```

```
arr[:,1]    # get second column: "y"
```

```
array([10.  , 10.1, 10.2, 13.  , 12.  , 9.  , 8.6, 8.8, 11.  ])
```

```
arr[3,:]    # get fourth line
```

```
array([ 3., 13.])
```

On the numpy package – loadtxt

loading files as numpy arrays:
(need "import numpy as np; import matplotlib.pyplot as plt")

```
arr=np.loadtxt("file.txt")
```

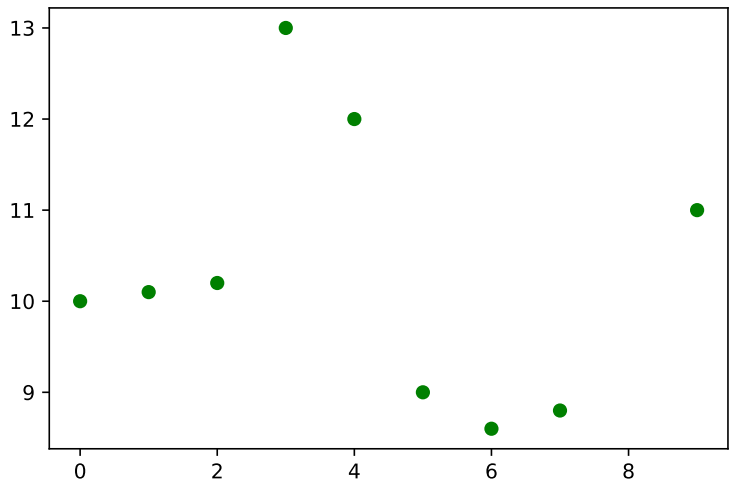
sample "file.txt":

```
# x y
0 10
1 10.1
2 10.2
3 13
4 12
5 9
6 8.6
7 8.8
9 11
```

```
array([[ 0. , 10. ],
       [ 1. , 10.1],
       [ 2. , 10.2],
       [ 3. , 13. ],
       [ 4. , 12. ],
       [ 5. ,  9. ],
       [ 6. ,  8.6],
       [ 7. ,  8.8],
       [ 9. , 11. ]])
```

comment (first line, starting with #) ignored!

```
plt.plot(arr[:,0], arr[:,1], "go")    # plot points with "o" marks
```



basic plot with just 4 commands!

Cosmic Microwave Background (CMB)

Predicted in 1948 [Alpher, Herman]: blackbody background radiation at $T \simeq 5$ K

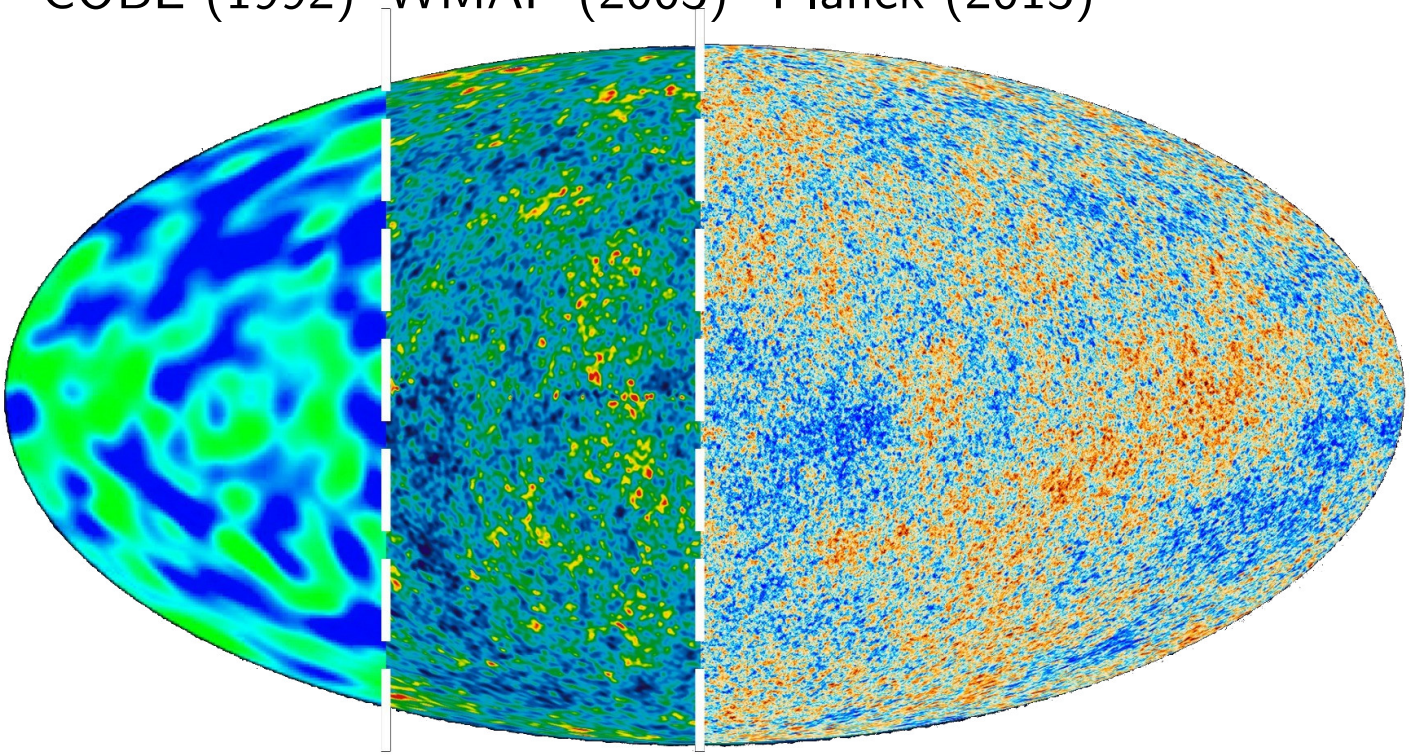
Discovery (accidental): [Penzias, Wilson 1964] ⟶ Nobel prize 1978

perfect black body spectrum at $T_{\rm CMB} = 2.72548 \pm 0.00057$ K [Fixsen, 2009]

Anisotropies at the level of 10^{-5}: very high precision measurements are needed.

Improvement of the CMB experiments in 20 years:

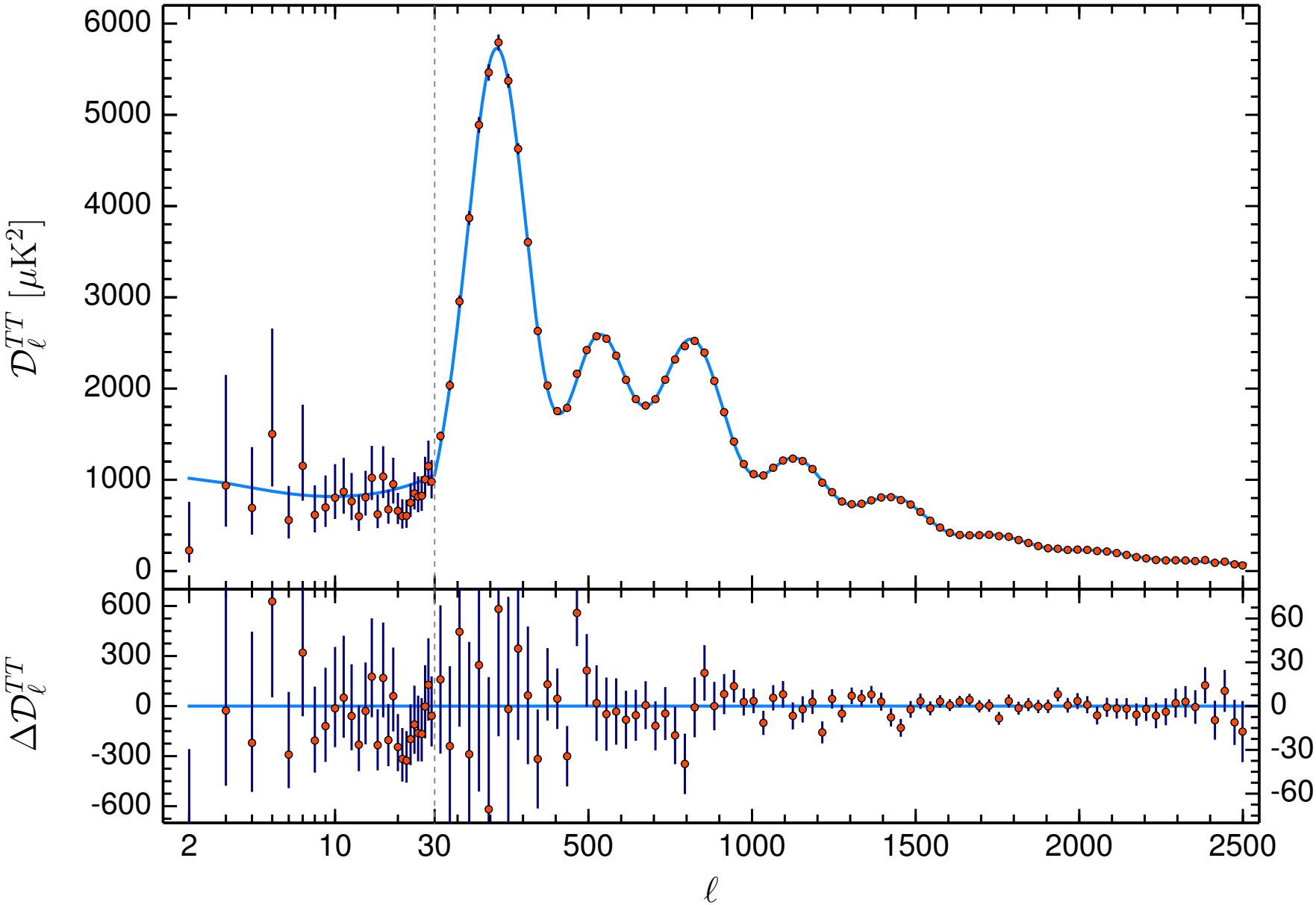
COBE (1992) WMAP (2003) Planck (2013)



Planck DR3 results - Temperature

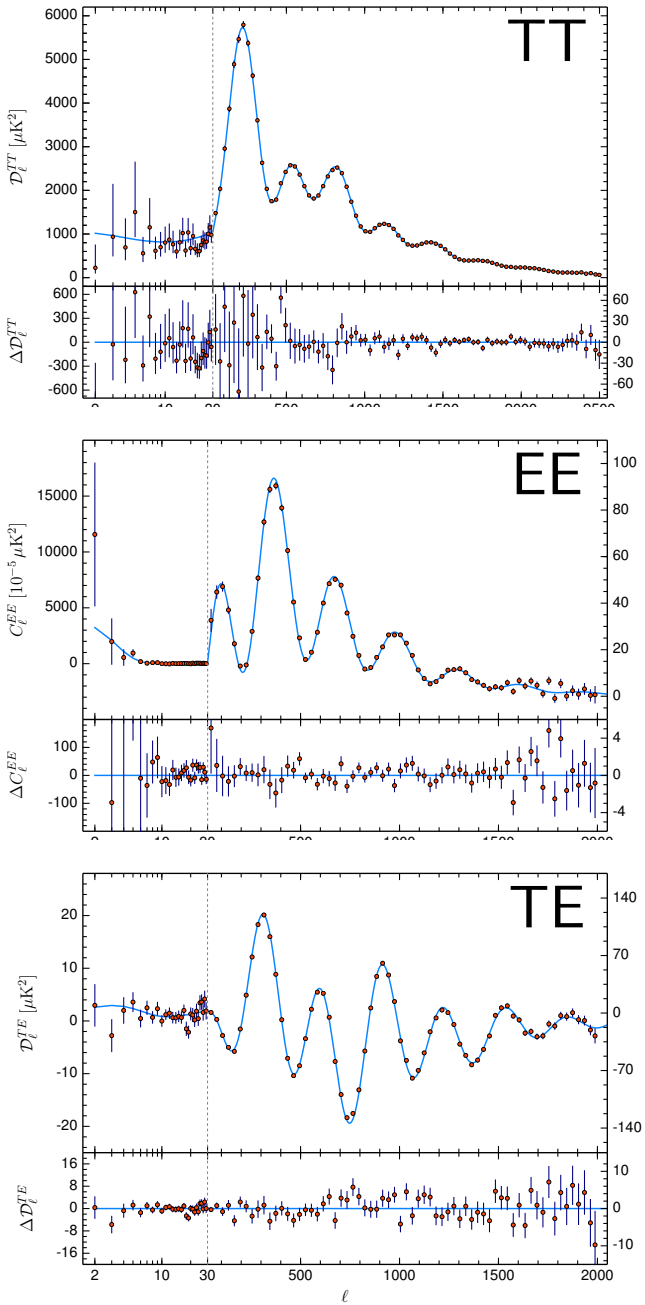
[Planck Collaboration, 2018]

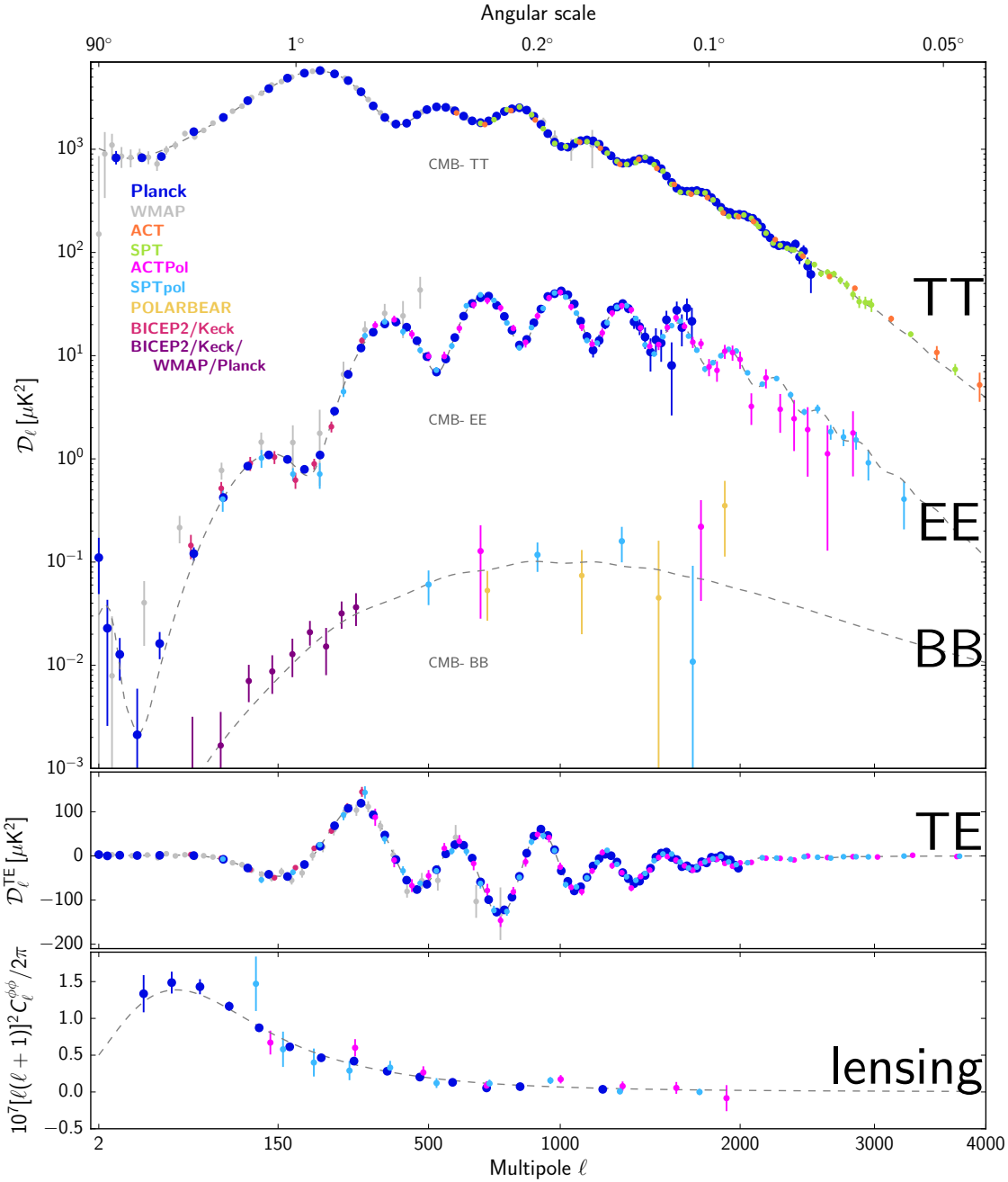
Planck legacy temperature auto-correlation power spectrum:



CMB spectra as of 2018

[Planck Collaboration, 2018]





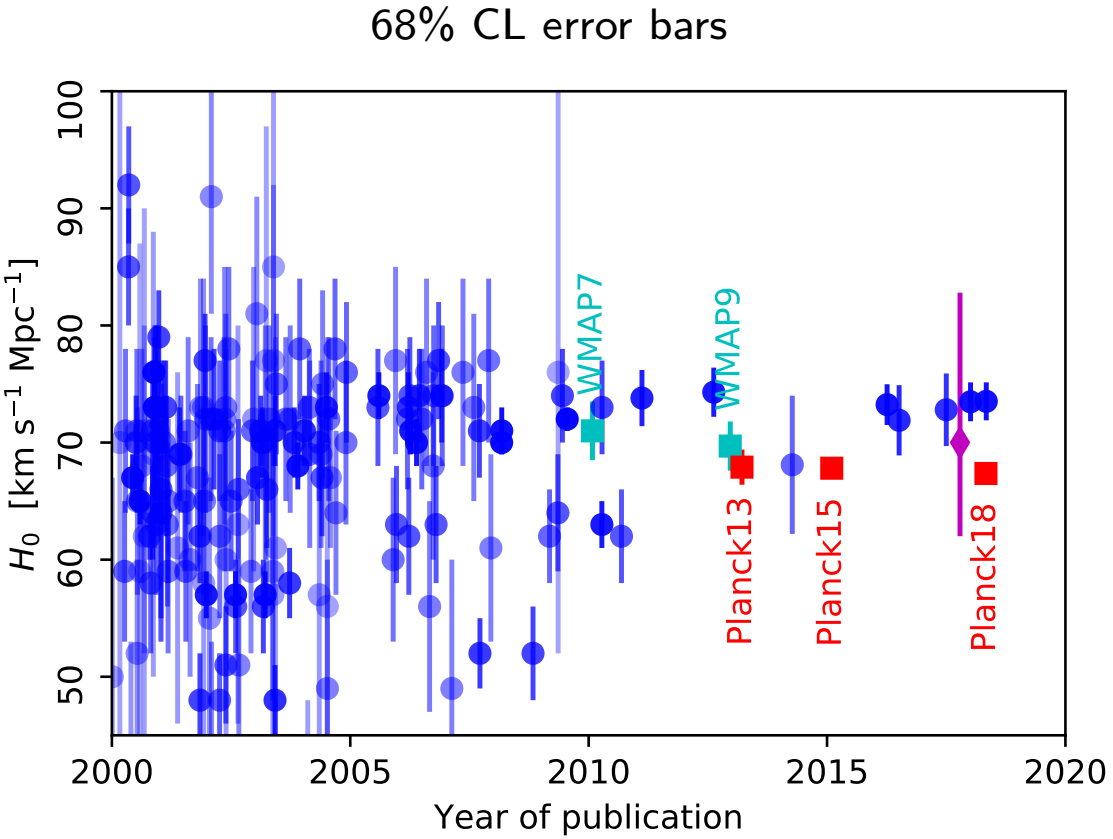
The Hubble parameter tension

[Planck Collaboration, 2018]

$$v = H_0 d,$$
with $H_0 = H(z = 0)$

Local measurements:
$H(z = 0)$,
local and independent on evolution (model independent, but **systematics?**)

CMB measurements
(probe $z \simeq 1100$):
H_0 from the cosmological evolution (model dependent, well controlled systematics)

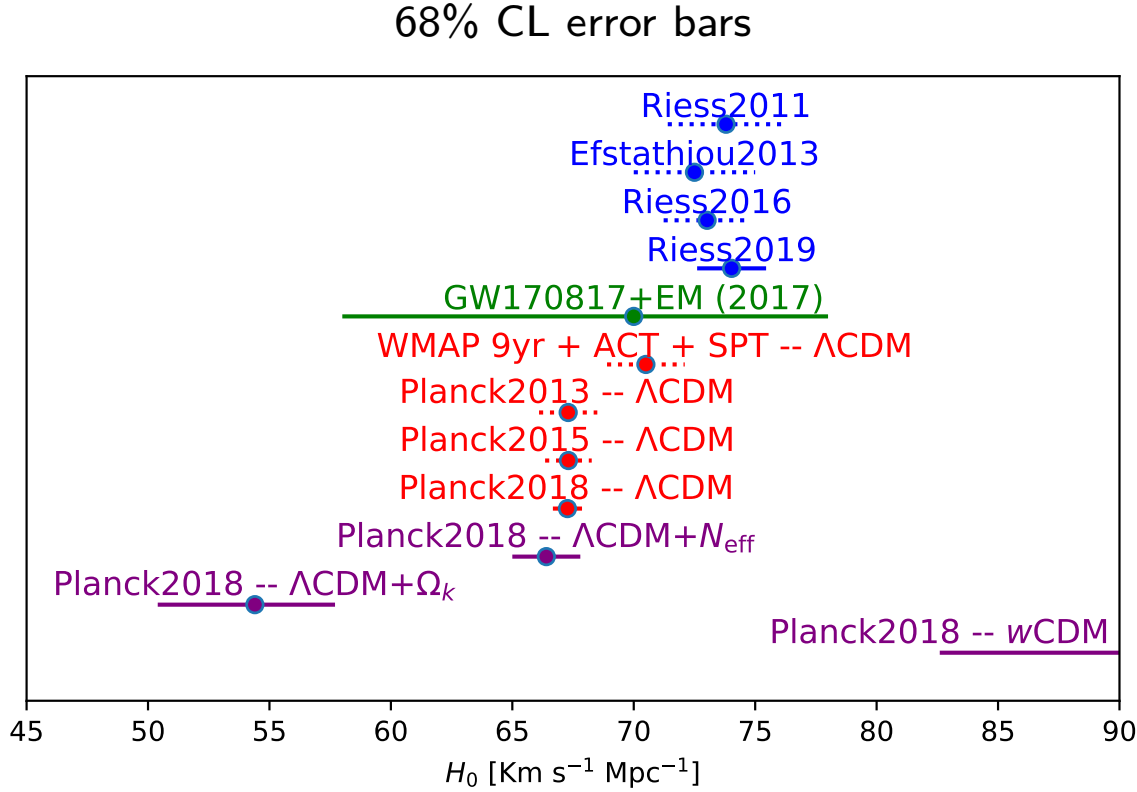


The Hubble parameter tension

$$v = H_0 d,$$
with $H_0 = H(z = 0)$

Local measurements:
$H(z = 0)$,
local and independent on evolution (model independent, but **systematics?**)

CMB measurements
(probe $z \simeq 1100$):
H_0 from the cosmological evolution (model dependent, well controlled systematics)



Using HST Cepheids:
[Efstathiou 2013] $H_0 = 72.5 \pm 2.5\,\mathrm{Km\ s^{-1}\ Mpc^{-1}}$
[Riess+, 2019] $H_0 = 74.03 \pm 1.42\,\mathrm{Km\ s^{-1}\ Mpc^{-1}}$

GW: [Abbott et al., 2017] $H_0 = 70^{+12}_{-8}\,\mathrm{Km\ s^{-1}\ Mpc^{-1}}$

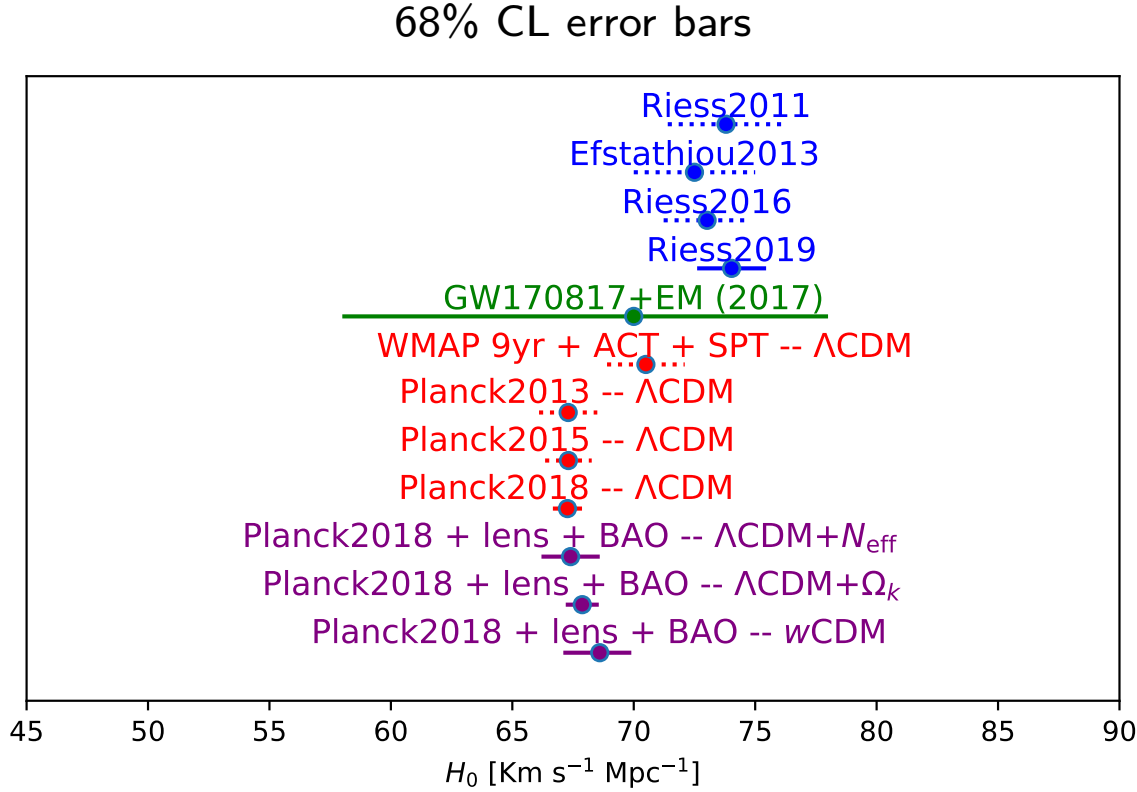
(ΛCDM model - CMB data only)
[Planck 2013]: $H_0 = 67.3 \pm 1.2\,\mathrm{Km\ s^{-1}\ Mpc^{-1}}$
[Planck 2018]: $H_0 = 67.27 \pm 0.60\,\mathrm{Km\ s^{-1}\ Mpc^{-1}}$

The Hubble parameter tension

$$v = H_0 d,$$
with $H_0 = H(z = 0)$

Local measurements:
$H(z = 0)$,
local and independent on evolution (model independent, but **systematics?**)

CMB measurements
(probe $z \simeq 1100$):
H_0 from the cosmological evolution (model dependent, well controlled systematics)



Using HST Cepheids:
[Efstathiou 2013] $H_0 = 72.5 \pm 2.5\,\mathrm{Km\ s^{-1}\ Mpc^{-1}}$
[Riess+, 2019] $H_0 = 74.03 \pm 1.42\,\mathrm{Km\ s^{-1}\ Mpc^{-1}}$

GW: [Abbott et al., 2017] $H_0 = 70^{+12}_{-8}\,\mathrm{Km\ s^{-1}\ Mpc^{-1}}$

(ΛCDM model - CMB data only)
[Planck 2013]: $H_0 = 67.3 \pm 1.2\,\mathrm{Km\ s^{-1}\ Mpc^{-1}}$
[Planck 2018]: $H_0 = 67.27 \pm 0.60\,\mathrm{Km\ s^{-1}\ Mpc^{-1}}$

Computing the CMB spectrum

hard task given current experimental precision! requires numerical codes...

1. `COSMICS` – fortran77 – Bertschinger 1995 – not maintained
   basic equations, brute-force C_ℓ^{TT};
2. `CMBFAST` – fortran77 – Seljak&Zaldarriaga 1996 – not maintained
   Line-of-sight, adds $C_\ell^{EE,TE,BB}$, open universe, CMB lensing;
3. `CAMB` – fortran90/2000 – Lewis&Challinor 1999 – `http://camb.info`
   closed universe, better lensing, new algorithms/species/observables...;
4. `CMBEASY` – C++ – Doran 2003 – not maintained;
5. `CLASS` – C – Lesgourgues&Tram 2011 – `http://class-code.net`
   simpler polarization equations, new algorithms/species/observables....

Computing the CMB spectrum

hard task given current experimental precision! requires numerical codes...

1. COSMICS – fortran77 – Bertschinger 1995 – not maintained
   basic equations, brute-force C_ℓ^{TT};
2. CMBFAST – fortran77 – Seljak&Zaldarriaga 1996 – not maintained
   Line-of-sight, adds $C_\ell^{EE,TE,BB}$, open universe, CMB lensing;
3. CAMB – fortran90/2000 – Lewis&Challinor 1999 – http://camb.info
   closed universe, better lensing, new algorithms/species/observables...;
4. CMBEASY – C++ – Doran 2003 – not maintained;
5. CLASS – C – Lesgourgues&Tram 2011 – http://class-code.net
   simpler polarization equations, new algorithms/species/observables....

CLASS project started to check code-induced bias in CAMB

CLASS and CAMB agree at the level of 0.01% for CMB observables

both codes have a python interface

CLASS: clear structure, user-friendly, space for improvements/extensions...

Where to find help on CLASS?

Basic information:

- historical webpage `http://class-code.net`
- new online documentation: `https://github.com/lesgourg/class_public/wiki` → "online html documentation", sections:
  - CLASS: Cosmic Linear Anisotropy Solving System (general info, installation);
  - Where to find information and documentation on CLASS? (includes references to many papers useful to understand the class equations and physics)
  - CLASS overview (architecture, input/output, general principles)

Where to find help on CLASS?

Basic information:

- historical webpage `http://class-code.net`
- new online documentation: `https://github.com/lesgourg/class_public/wiki` → "online html documentation", sections:
  - CLASS: Cosmic Linear Anisotropy Solving System (general info, installation);
  - Where to find information and documentation on CLASS? (includes references to many papers useful to understand the class equations and physics)
  - CLASS overview (architecture, input/output, general principles)

More advanced:

- courses at `https://lesgourg.github.io/courses.html`
- `https://github.com/lesgourg/class_public/wiki` → "online html documentation", last sections ("Data structures", "Files")

CLASS spirit

Flexible coding spirit, in principle easy to understand and expand

CLASS spirit

Flexible coding spirit, in principle easy to understand and expand

Equations: [Ma&Bertschinger 1996, astro-ph/9506072]

Multi-gauge (newtonian and synchronous, prepared for more)

CLASS spirit

Flexible coding spirit, in principle easy to understand and expand

Equations: [Ma&Bertschinger 1996, astro-ph/9506072]
Multi-gauge (newtonian and synchronous, prepared for more)

Inputs: Parameters are independent for each module.
Checks and processing where needed

CLASS spirit

Flexible coding spirit, in principle easy to understand and expand

Equations: [Ma&Bertschinger 1996, astro-ph/9506072]
Multi-gauge (newtonian and synchronous, prepared for more)

Inputs: Parameters are independent for each module.
Checks and processing where needed

Units: homogeneous units, Mpc^n everywhere (except thermodynamics)

CLASS spirit

Flexible coding spirit, in principle easy to understand and expand

Equations: [Ma&Bertschinger 1996, astro-ph/9506072]
Multi-gauge (newtonian and synchronous, prepared for more)

Inputs: Parameters are independent for each module.
Checks and processing where needed

Units: homogeneous units, Mpc^n everywhere (except thermodynamics)

Code:

- plain `C`, mimic `C++` – no external libraries (except `OpenMP`)
  Accessible and self contained, many comments, ...
- 10 modules, distinct physical tasks, no duplicate equations
- **no hard coding**: dynamic allocation, all constants are stored,
- `precision` settings are read from input, no global variables,
  steps and approximations inferred dynamically
- all features inside `if` $\rightarrow$ easy to disable and no slow down

CLASS spirit

Flexible coding spirit, in principle easy to understand and expand

Equations: [Ma&Bertschinger 1996, astro-ph/9506072]
Multi-gauge (newtonian and synchronous, prepared for more)

Inputs: Parameters are independent for each module.
Checks and processing where needed

Units: homogeneous units, Mpc^n everywhere (except thermodynamics)

Code:

- plain `C`, mimic `C++` – no external libraries (except `OpenMP`)
  Accessible and self contained, many comments, ...
- 10 modules, distinct physical tasks, no duplicate equations
- **no hard coding**: dynamic allocation, all constants are stored, `precision` settings are read from input, no global variables, steps and approximations inferred dynamically
- all features inside `if` $\rightarrow$ easy to disable and no slow down

Errors: in principle no segfault, tree-like error information (like `Python`)

CLASS spirit

Flexible coding spirit, in principle easy to understand and expand

Equations: [Ma&Bertschinger 1996, astro-ph/9506072]
Multi-gauge (newtonian and synchronous, prepared for more)

Inputs: Parameters are independent for each module.
Checks and processing where needed

Units: homogeneous units, Mpc^n everywhere (except thermodynamics)

Code:

- plain `C`, mimic `C++` – no external libraries (except `OpenMP`)
  Accessible and self contained, many comments, ...
- 10 modules, distinct physical tasks, no duplicate equations
- **no hard coding**: dynamic allocation, all constants are stored, `precision` settings are read from input, no global variables, steps and approximations inferred dynamically
- all features inside `if` $\rightarrow$ easy to disable and no slow down

Errors: in principle no segfault, tree-like error information (like `Python`)

History: all versions available through `Git`/`GitHub` (see later)

Installing CLASS

according to J. Lesgourgues, installation is:

- `Linux` $\rightarrow$ straightforward
- `Mac` $\rightarrow$ slightly tricky but easy
- `Windows` $\rightarrow$ "We suggest to not even try"

Installing CLASS

according to J. Lesgourgues, installation is:

- `Linux` → straightforward
- `Mac` → slightly tricky but easy
- `Windows` → "We suggest to not even try"

Recommended installation procedure: (`Unix`!)

- Install:
```
git clone http://github.com/lesgourg/class_public.git class
cd class/
```
- Compile:
```
make clean ; make -j
```

Installing CLASS

according to J. Lesgourgues, installation is:

- `Linux` → straightforward
- `Mac` → slightly tricky but easy
- `Windows` → "We suggest to not even try"

Recommended installation procedure: (`Unix!`)

- Install:
```
git clone http://github.com/lesgourg/class_public.git class
cd class/
```
- Compile:
```
make clean ; make -j
```

first run (C code):
```
./class explanatory.ini
```

first run (python module):
```
python -c "from classy import Class"
```
(no error message should appear)

Installing CLASS

according to J. Lesgourgues, installation is:

- `Linux` → straightforward
- `Mac` → slightly tricky but easy
- `Windows` → "We suggest to not even try"

Recommended installation procedure: (`Unix!`)

- Install: `git clone http://github.com/lesgourg/class_public.git class`
  `cd class/`
- Compile: `make clean ; make -j`

first run (C code): `./class explanatory.ini`

first run (python module): `python -c "from classy import Class"`
(no error message should appear)

in case of errors: try checking
`https://github.com/lesgourg/class_public/wiki/Installation`

CLASS folders

In your CLASS directory, you should see:

`explanatory.ini`	reference input file
`source/`	the 10 modules of class, ALL THE PHYSICS
`tools/`	auxiliary code (numerical methods): ALL THE MATH
`main/`	main class function: just calls 10 modules
`test/`	other main functions for testing part of the code
`output/`	output files (when running from terminal)
`include/`	header files (`*.h`) containing declarations
`doc/`	pdf version of the manual
`python/`	`python` wrapper
`cpp/`	C++ wrapper
`notebooks/`	example of `jupyter` notebooks
`scripts/`	same as plain `python` scripts
`RealSpaceInterface/`	graphical interface

CLASS folders

In your CLASS directory, you should see:

`explanatory.ini`	reference input file
`source/`	the 10 modules of class, ALL THE PHYSICS
`tools/`	auxiliary code (numerical methods): ALL THE MATH
`main/`	main class function: just calls 10 modules
`test/`	other main functions for testing part of the code
`output/`	output files (when running from terminal)
`include/`	header files (`*.h`) containing declarations
`doc/`	pdf version of the manual
`python/`	`python` wrapper
`cpp/`	C++ wrapper
`notebooks/`	example of `jupyter` notebooks
`scripts/`	same as plain `python` scripts
`RealSpaceInterface/`	graphical interface

plus a few other directories containing ancillary data (`bbn/`)
or interfaced codes (`hyrec/`, `external_Pk/`)

The 10 CLASS modules

calling CLASS means executing the sequence of modules:

1. `input.c` — parse / make sense of input parameters
2. `background.c` — homogeneous cosmology
3. `thermodynamics.c` — ionisation history, scattering rate
4. `perturbations.c` — linear Fourier perturbations
5. `primordial.c` — primordial spectrum, inflation
6. `nonlinear.c` — recipes for non-linear corrections to 2-point statistics
7. `transfer.c` — from Fourier to multipole space
8. `spectra.c` — 2-point statistics (power spectra)
9. `lensing.c` — CMB lensing
10. `output.c` — print output (not used from `python`)

The 10 CLASS modules

calling CLASS means executing the sequence of modules:

1. `input.c` — parse / make sense of input parameters
2. `background.c` — homogeneous cosmology
3. `thermodynamics.c` — ionisation history, scattering rate
4. `perturbations.c` — linear Fourier perturbations
5. `primordial.c` — primordial spectrum, inflation
6. `nonlinear.c` — recipes for non-linear corrections to 2-point statistics
7. `transfer.c` — from Fourier to multipole space
8. `spectra.c` — 2-point statistics (power spectra)
9. `lensing.c` — CMB lensing
10. `output.c` — print output (not used from `python`)

Plain C mimicking C++ and object-oriented:

C++	CLASS
10 "classes", each with a constructor/destructor and a few functions callable from outside.	each module has one structure, one initialisation function, one freeing function, and a few functions callable from outside.

main executable calls the 10 initialisation and freeing functions!

Models in CLASS

Many models already implemented in the code,
others available in extended codes based on CLASS:

- all models in CAMB
- primordial perturbations
  internal inflationary perturbation module with given $V(\psi)$, ...
- neutrinos
  chemical potentials, arbitrary phase-space distributions, flavor mixing...
- Dark Matter
  warm, annihilating, decaying, interacting...
- Dark Energy
  fluid with flexible $w(a)$ + sound speed, quintessence with given $V(\psi)$...
- Modified Gravity with HiCLASS, http://www.hiclass-code.net
- second-order perturbations: SONG, https://github.com/coccoinomane/song
- particle physics modules + exotic energy injection in ExoCLASS branch

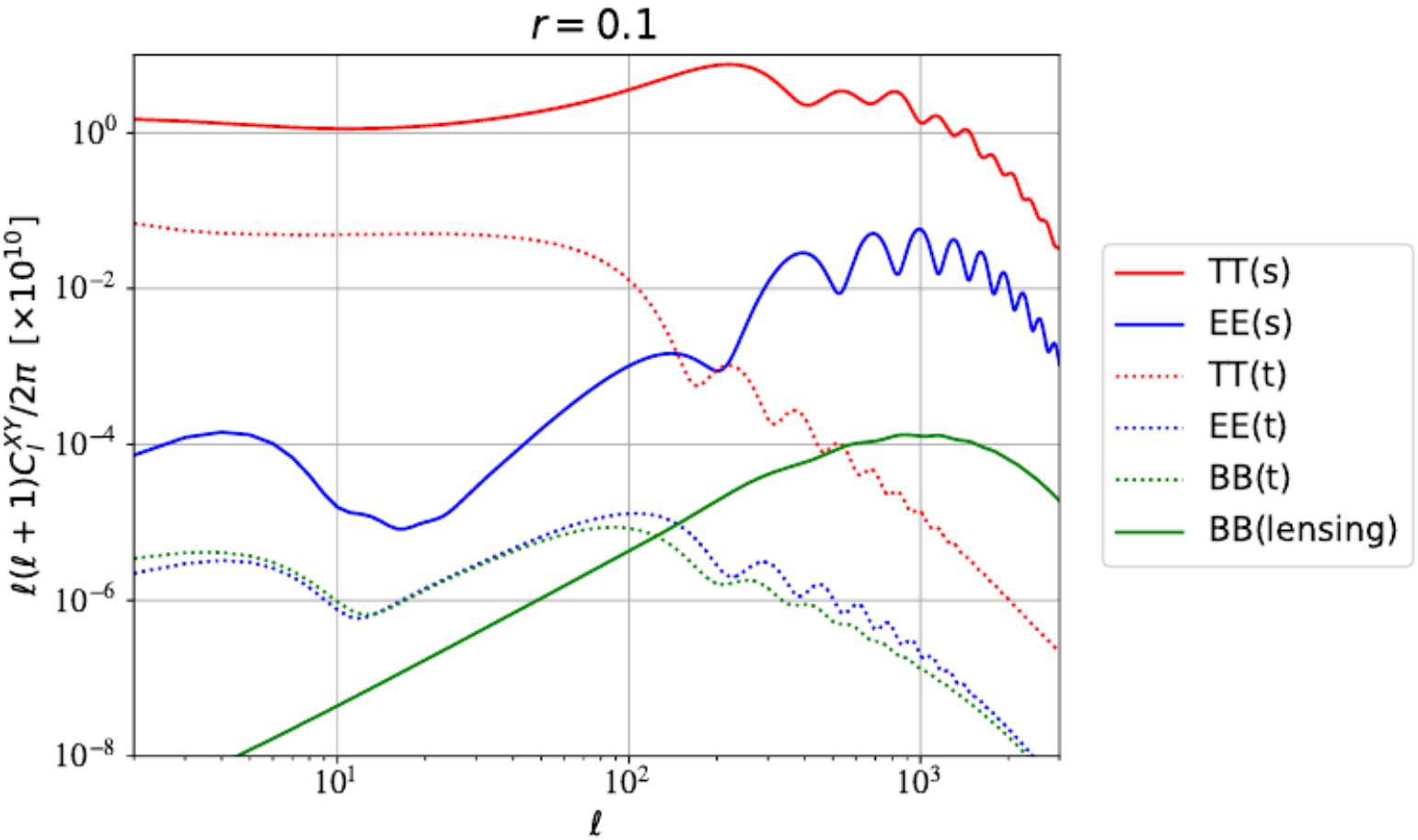
CLASS outputs

Some examples of the outputs that one can obtain from CLASS:

CLASS outputs

Some examples of the outputs that one can obtain from CLASS:

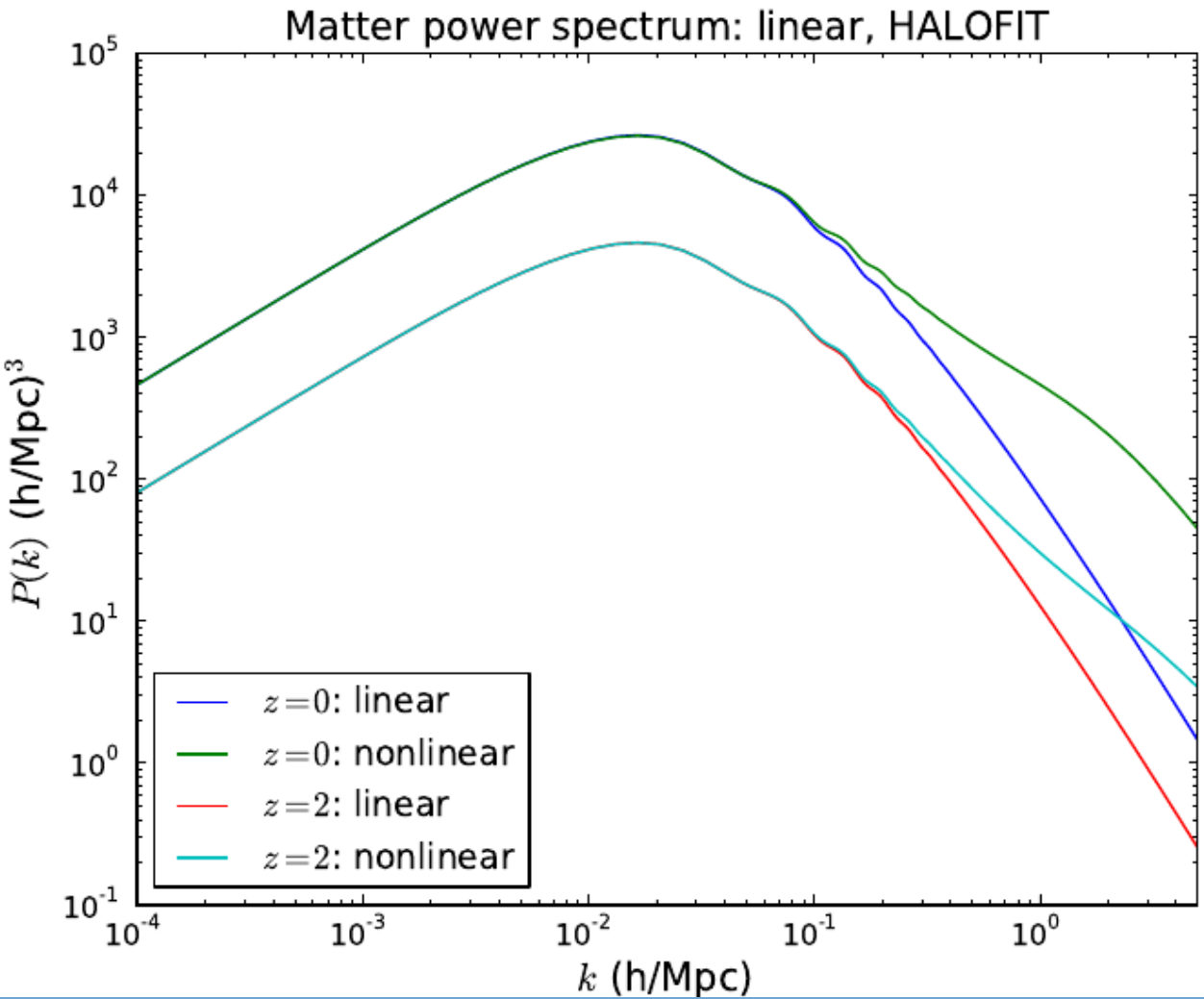
CMB spectrum (all components)



CLASS outputs

Some examples of the outputs that one can obtain from CLASS:

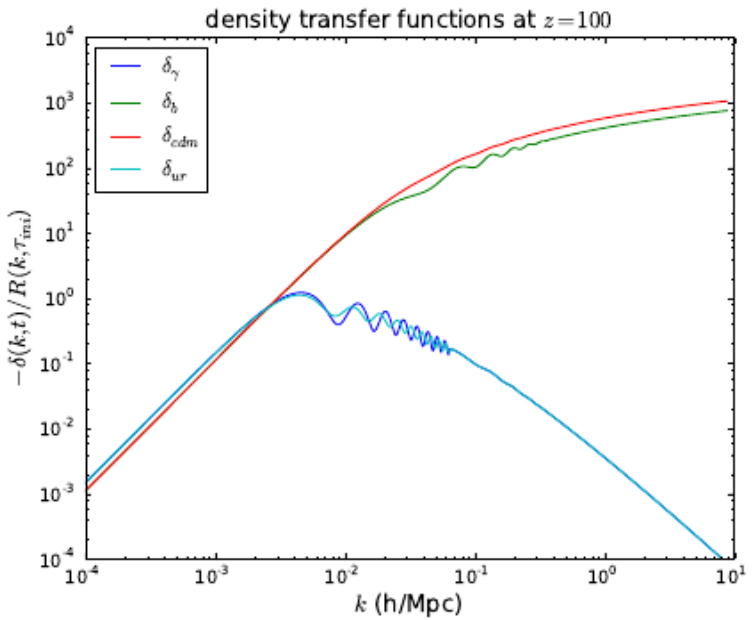
matter power spectrum (linear and non-linear, at various z)

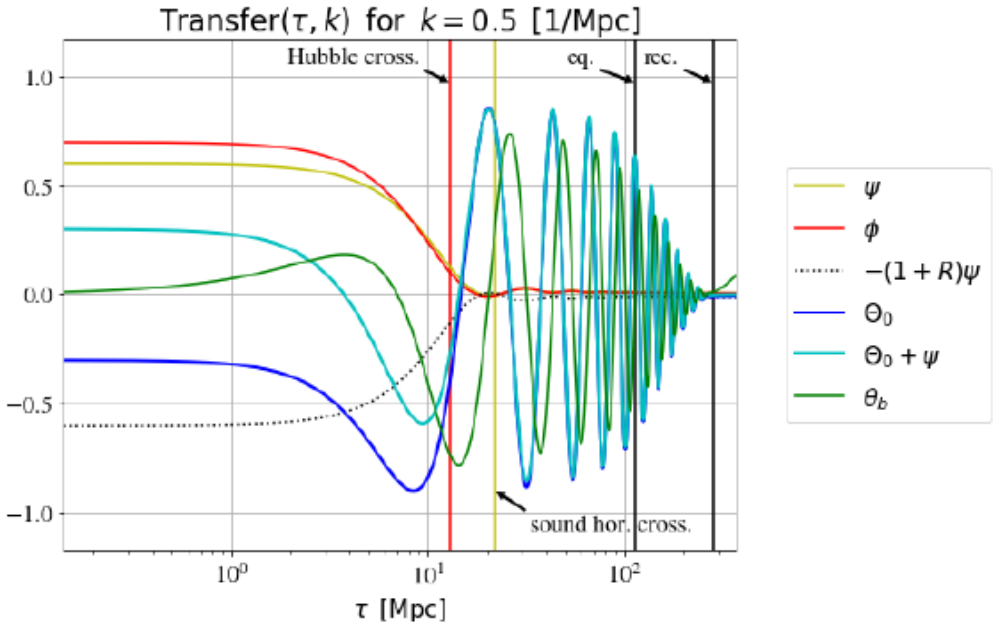


CLASS outputs

Some examples of the outputs that one can obtain from CLASS:

transfer functions at given z

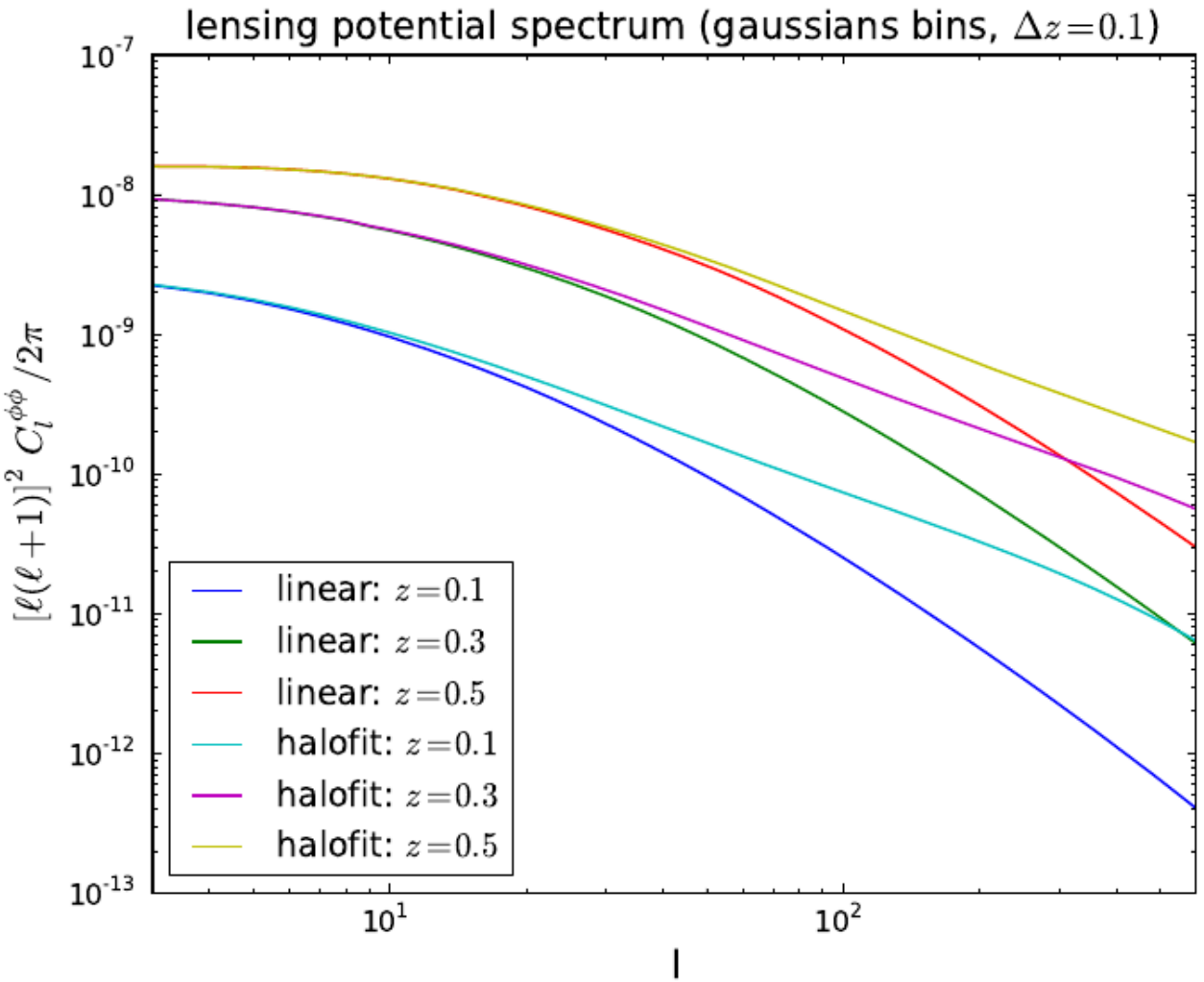




CLASS outputs

Some examples of the outputs that one can obtain from CLASS:

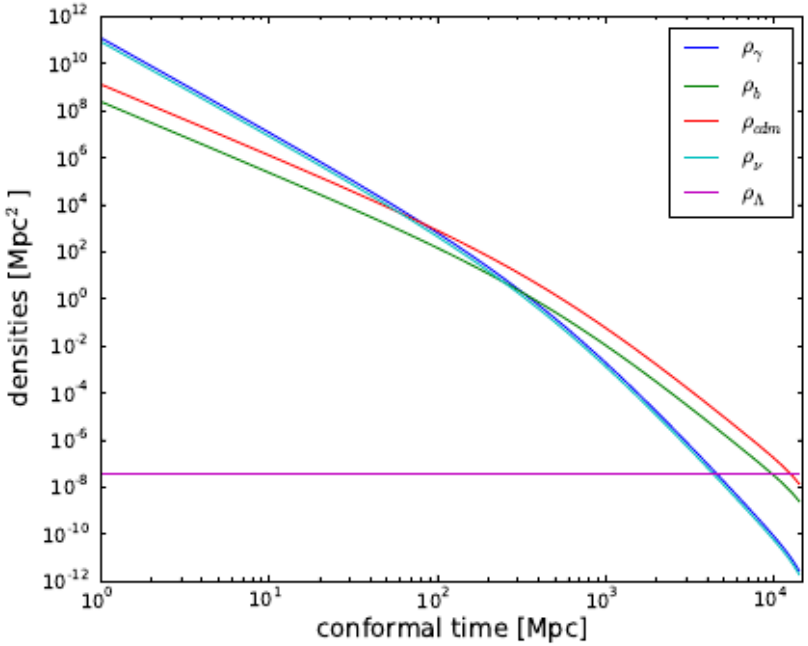
matter density and lensing spectra

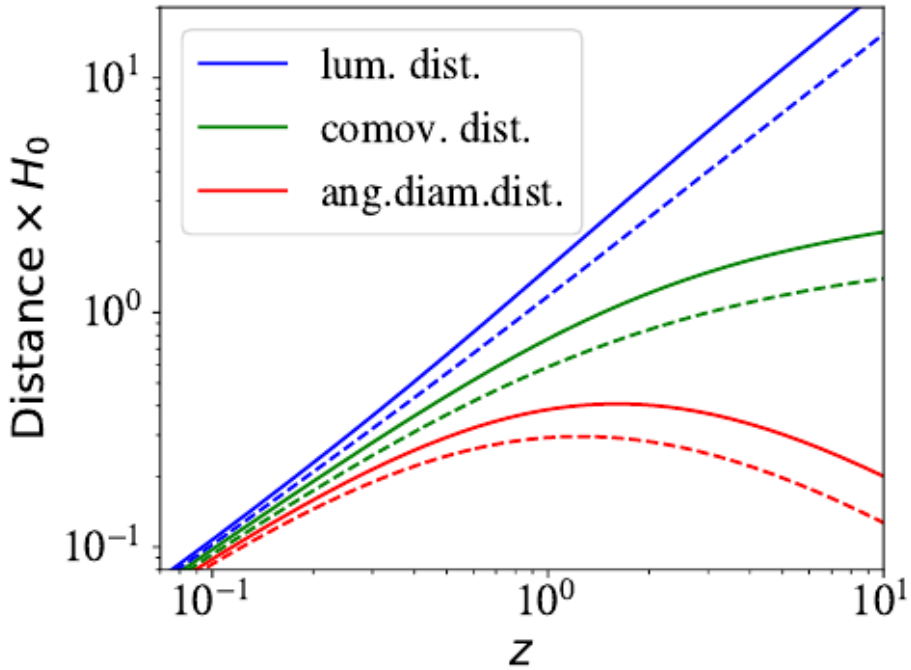


CLASS outputs

Some examples of the outputs that one can obtain from CLASS:

evolution of the background: densities, distances, ...

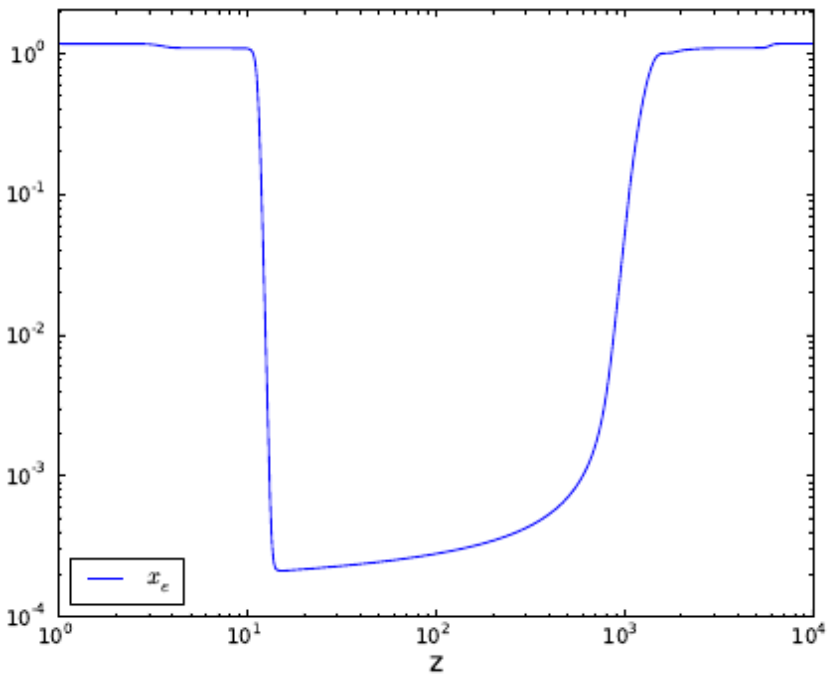


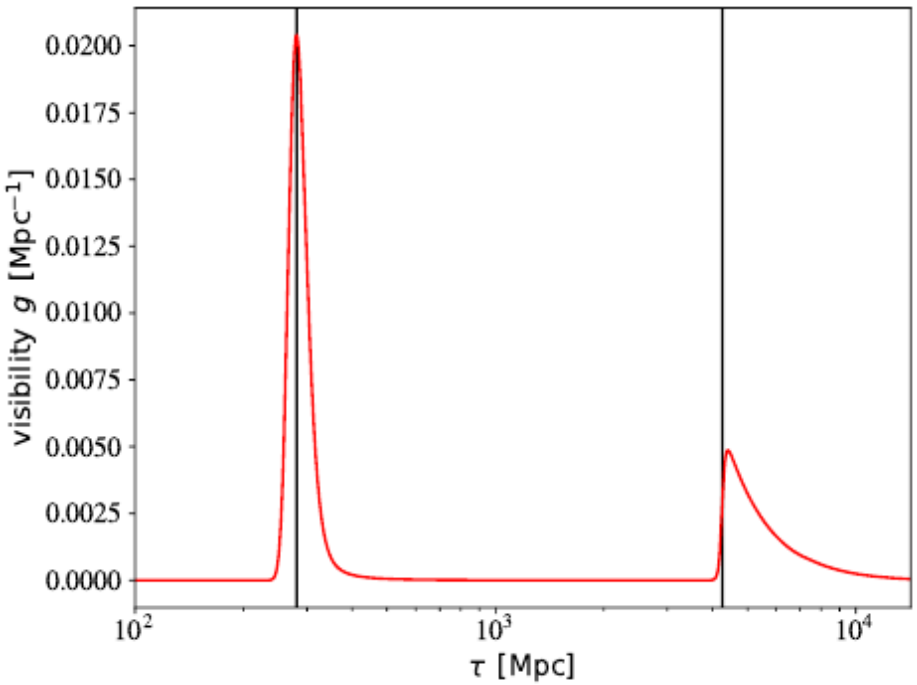


CLASS outputs

Some examples of the outputs that one can obtain from CLASS:

Thermal history, reionization fraction

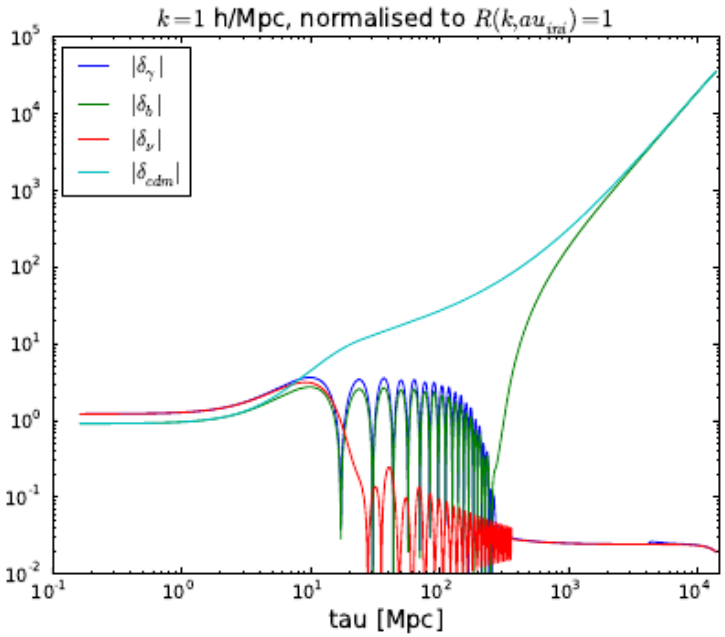


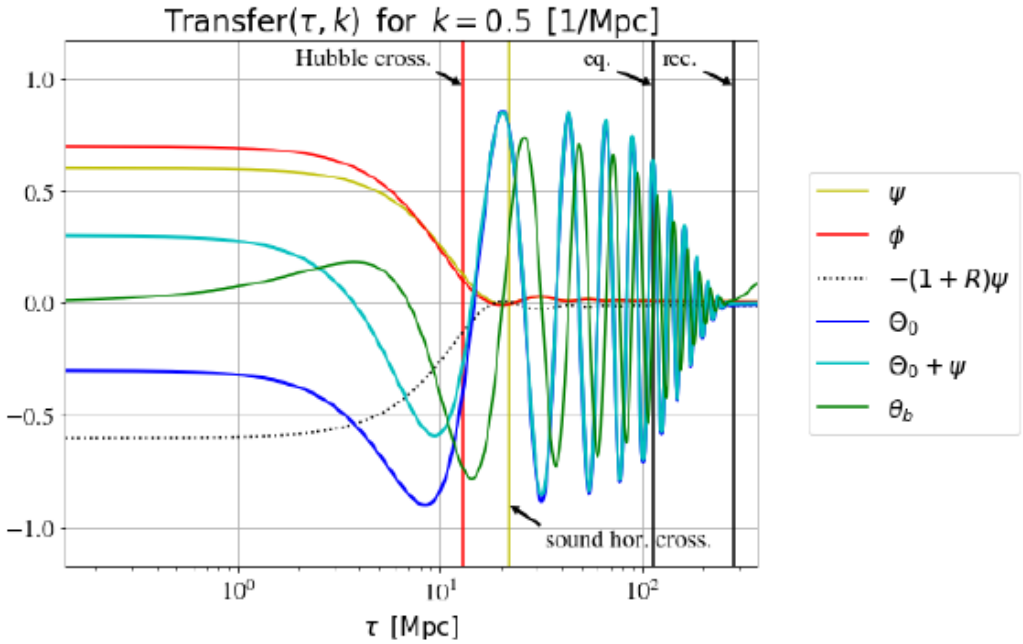


CLASS outputs

Some examples of the outputs that one can obtain from CLASS:

time evolution of perturbations for individual Fourier modes





Running CLASS in terminal

Run with any input file with (compulsory) extension `*.ini`:

`>./class explanatory.ini`

It gives some output:

```
Reading input parameters
-> matched budget equations by adjusting Omega_Lambda = 6.8786
Running CLASS version v2.7.0
Computing background
-> age = 13.795359 Gyr
-> conformal age = 14165.045412 Mpc
Computing thermodynamics with Y_He =0.2453
-> recombination at z = 1089.184869
(...)
Writing output files in output/explanatory01_...
```

output comes from 10 verbose parameters fixed to 1 in `explanatory.ini`

(see them with > `tail explanatory.ini`)

Running CLASS in terminal

Run with any input file with (compulsory) extension `*.ini`:

`>./class explanatory.ini`

All possible input parameters and details on the syntax explained in `explanatory.ini`

$\longrightarrow$ this is only a reference file; we advise you to never modify it, but rather to copy it and reduce it to a shorter and more friendly file.

For basic usage: `explanatory.ini` $\rightarrow$ full documentation of the code

The input files

Run with any input file with (compulsory) extension `*.ini`:

`>./class my_model.ini`

Example of `*.ini`:

```
output = tCl,pCl,lCl,mPk # temp, pol, lens C_ℓ + matter P(k)
lensing = yes # include CMB lensing effect
non linear = halofit # non-linear P(k) from HALOFIT
root = output/my_model_ # where to write the output
write warnings = yes # will alert you if wrong input syntax
more comments , ignored because no equal sign in this line
# comment with an = , still ignored thanks to the sharp
```

Order of lines doesn't matter at all.

See later for more

The input files

Run with any input file with (compulsory) extension `*.ini`:

`>./class my_model.ini`

All parameters not passed are fixed to default,
i.e. the most reasonable or minimalistic choice
(ΛCDM with Planck 2013 bestfit)

The input files

Run with any input file with (compulsory) extension `*.ini`:

```
>./class my_model.ini
```

All parameters not passed are fixed to default,
i.e. the most reasonable or minimalistic choice
(ΛCDM with Planck 2013 bestfit)

CLASS can take two input files `*.ini` and `*.pre`:

```
> ./class my_model.ini cl_permille.pre
```

`*.pre` is typically for tuning the precision of the code

Fluids in CLASS

Many available fluids	see also `explanatory.ini`!
(compulsory) photons:	`T_cmb` or `Omega_g` or `omega_g`
(compulsory) baryons:	`Omega_b` or `omega_b`
spatial curvature:	`Omega_k`
ultra-relativistic species:	(massless ν) `N_ur` or `Omega_ur` or `omega_ur`
cold dark matter:	`Omega_cdm` or `omega_cdm`
cdm decaying into dark radiation:	`Omega_dcdmdr` or `omega_dcdmdr` + `Gamma_dcdm`
`N_ncdm` non-cold dark matter species:	(massive ν, warm dark matter, ...) `m_ncdm` or `Omega_ncdm` or `omega_ncdm`, ...
cosmological constant:	`Omega_Lambda`
dark energy fluid:	`Omega_fld` + `w0_fld`, `wa_fld`, `cs2_fld`, ...
quintessence:	(scalar field) `Omega_scf` + specifications

More input parameters

More settings: see also `explanatory.ini`!

- Hubble parameter: `h` or `H0`
- Primordial He: `YHe=BBN` or specific value
- reionization: `reio_parametrization` for selecting the model + specific parameters (e.g. `z_reio`)
- initial conditions: `P_k_ini type` + related parameters, e.g. `k_pivot`, `A_s` or `ln10^{10}A_s` or `sigma8`, `n_s`
- output = `tCl`, `pCl`, `lCl`, `nCl`, `sCl`, `mPk`, `dTk`, `vTk` (some cross-correlations are automatically included)
- temperature contributions = `tsw`, `eisw`, `lisw`, `dop`, `pol`
- number count contributions = `density`, `rsd`, `lensing`, `gr`
- modes = `s`(calar), `v`(ector), `t`(ensor)
- others for output: `write background/thermodynamics = yes/no`

An example input file

my_model.ini

```
T_cmb=2.7255
h=0.67
Omega_b=0.04
Omega_cdm=0.3
```

background

```
reio_parametrization = reio_camb
z_reio=9
```

reionization

```
P_k_ini type = analytic_Pk
A_s=2.2e-9
n_s=0.96
```

initial conditions from inflation

```
N_ur=3.045
N_ncdm=0
```

neutrinos (3 massless)

```
lensing=yes
non linear = halofit
write warnings=yes
output = tCl,pCl,lCl,mPk
root=output/my_model_
temperature contributions=tsw,eisw,lisw,dop,pol
modes=s
```

output and others

An example input file

my_model_nu.ini

```
T_cmb=2.7255
h=0.67
Omega_b=0.04
Omega_cdm=0.3
```

background

```
reio_parametrization = reio_camb
z_reio=9
```

reionization

```
P_k_ini type = analytic_Pk
A_s=2.2e-9
n_s=0.96
```

initial conditions from inflation

```
N_ur=1.0196
N_ncdm=2
m_ncdm=0.01,0.05
```

neutrinos (2 massive, 1 massless)

```
lensing=yes
non linear = halofit
write warnings=yes
output = tCl,pCl,lCl,mPk
root=output/my_model_nu_
temperature contributions=tsw,eisw,lisw,dop,pol
modes=s
```

output and others

Running CLASS, what do we obtain?

Recall: `> ./class my_model.ini`

Running CLASS, what do we obtain?

Recall: `> ./class my_model.ini`

```
> ls output/my_model*

output/my_model_cl.dat
output/my_model_cl_lensed.dat
output/my_model_pk.dat
output/my_model_pk_nl.dat
```

Running CLASS, what do we obtain?

Recall: `> ./class my_model.ini`

`> less output/my_model_cl.dat`

```
# dimensionless total [l(l+1)/2pi] C_l's
# for l=2 to 3000, i.e. number of multipoles equal to 2999
#
# -> if you prefer output in CAMB/HealPix/LensPix units/order, set 'format' to '
# -> if you don't want to see such a header, set 'headers' to 'no' in input file
# -> for CMB lensing (phi), these are C_l^phi-phi for the lensing potential.
#    Remember the conversion factors:
#    C_l^dd (deflection) = l(l+1) C_l^phi-phi
#    C_l^gg (shear/convergence) = 1/4 (l(l+1))^2 C_l^phi-phi
#
# 1:l     2:TT                      3:EE                      4:TE
    2       1.420679640795e-10        3.375786896387e-15        3.355861345810e-13
    3       1.357049899882e-10        4.958542279305e-15        3.937198029470e-13
    4       1.292421868384e-10        4.977563856173e-15        3.848127190997e-13
    5       1.242541500237e-10        3.902309085708e-15        3.434163292297e-13
    6       1.207435465525e-10        2.537952580486e-15        2.905035637071e-13
    7       1.184328316401e-10        1.467500308578e-15        2.390755284291e-13
    8       1.170055155711e-10        8.671361576704e-16        1.958316237368e-13
[...]
```

Running CLASS, what do we obtain?

Recall: `> ./class my_model.ini`

`> less output/my_model_pk.dat`

```
# Matter power spectrum P(k) at redshift z=0
# for k=1.08134e-05 to 9.15437 h/Mpc,
# number of wavenumbers equal to 617
#    1:k (h/Mpc)              2:P (Mpc/h)^3
      1.081337939134e-05       4.799662045949e+01
      2.384243184843e-05       1.025321375931e+02
      3.691902537792e-05       1.560113405118e+02
      5.007916130621e-05       2.090527626133e+02
      6.335945605984e-05       2.620052050307e+02
      7.679747518581e-05       3.151274573105e+02
      9.043207836946e-05       3.686390474530e+02
      1.043037801331e-04       4.227410126868e+02
      1.184551309336e-04       4.776265732419e+02
      1.329311234121e-04       5.334875439699e+02
      1.477796285475e-04       5.905187685478e+02
      1.630518663906e-04       6.489215524867e+02
      1.788029158370e-04       7.089065641692e+02
      1.950922674591e-04       7.706964779186e+02
[...]
```

Let's compare the two models

We can use `python` to read the files and plot the spectra!

Let's compare the two models

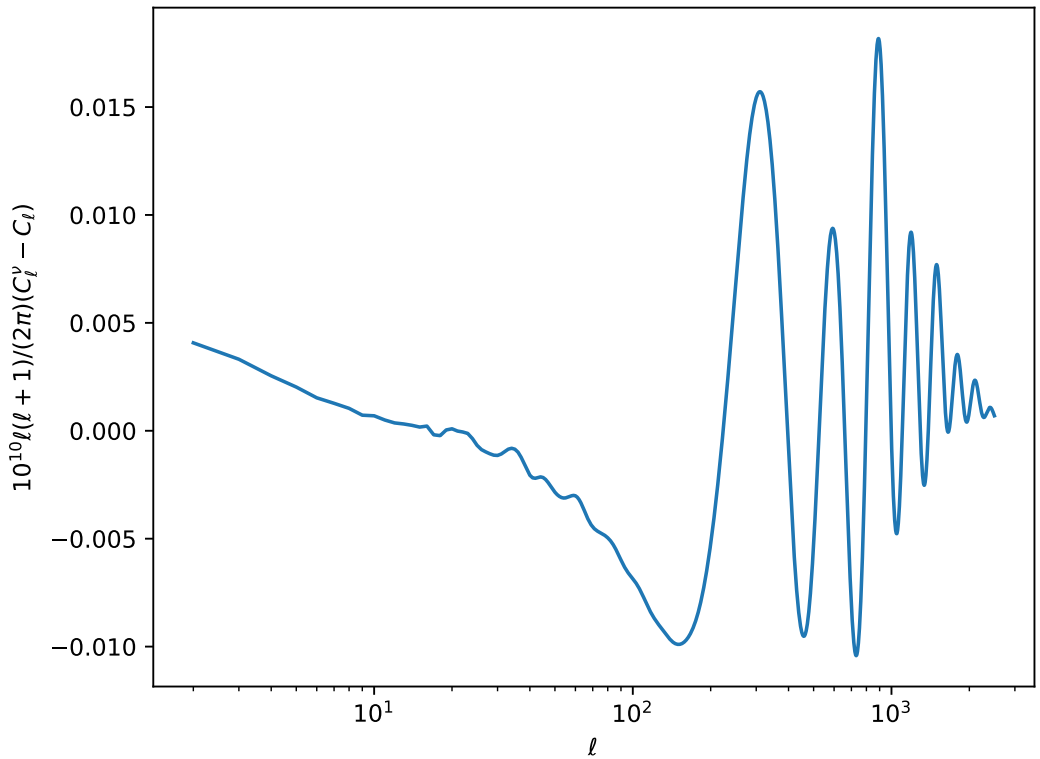
We can use python to read the files and plot the spectra!

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Let's compare the two models

We can use python to read the files and plot the spectra!

[2]:
```
observable="cl_lensed"
mymodel = np.loadtxt("output/my_model_%s.dat"%observable)
mymodelnu = np.loadtxt("output/my_model_nu_%s.dat"%observable)
plt.semilogx(mymodel[:,0], 1.e10 * (mymodel[:,1]-mymodelnu[:,1]))
plt.xlabel(r"$\ell$")
plt.ylabel(r"$10^{10} \ell(\ell+1)/(2\pi) (C_\ell^\nu - C_\ell)$")
plt.savefig("compare_%s.pdf"%observable)
```



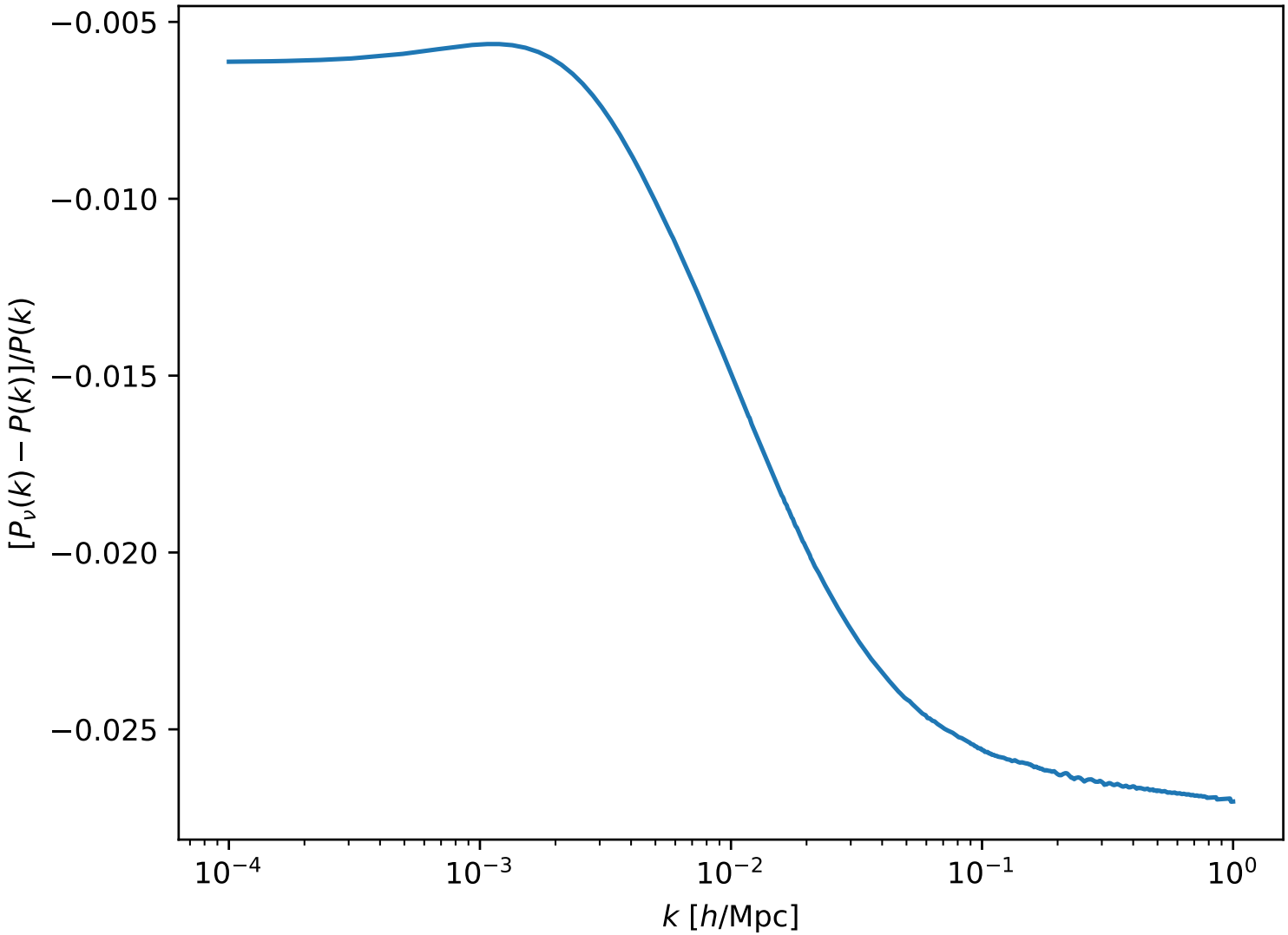
Let's compare the two models

We can use python to read the files and plot the spectra!

[3]:
```
observable="pk"
mymodel = np.loadtxt("output/my_model_%s.dat"%observable)
mymodelnu = np.loadtxt("output/my_model_nu_%s.dat"%observable)
# while the ell are the same, the k can differ!
# see print(mymodel[:,0], mymodelnu[:, 0])
# we need to interpolate to compute the difference
from scipy.interpolate import interp1d
pk = interp1d(mymodel[:, 0], mymodel[:, 1])
pknu = interp1d(mymodelnu[:, 0], mymodelnu[:, 1])
# pk, pknu are now functions of k

# 1e-4 < k < 1 should be well inside the range saved by CLASS
kvec = np.logspace(-4, 0, 1000)
plt.close() # close previous plot, if needed
plt.semilogx(kvec, (pknu(kvec)/pk(kvec))-1.)
plt.xlabel(r"$k$ [$h$/Mpc]")
plt.ylabel(r"$[P_\nu(k)-P(k)]/P(k)$")
plt.savefig("compare_%s.pdf"%observable)
```

Let's compare the two models

We can use `python` to read the files and plot the spectra!



CLASS Plotting Unit (CPU)

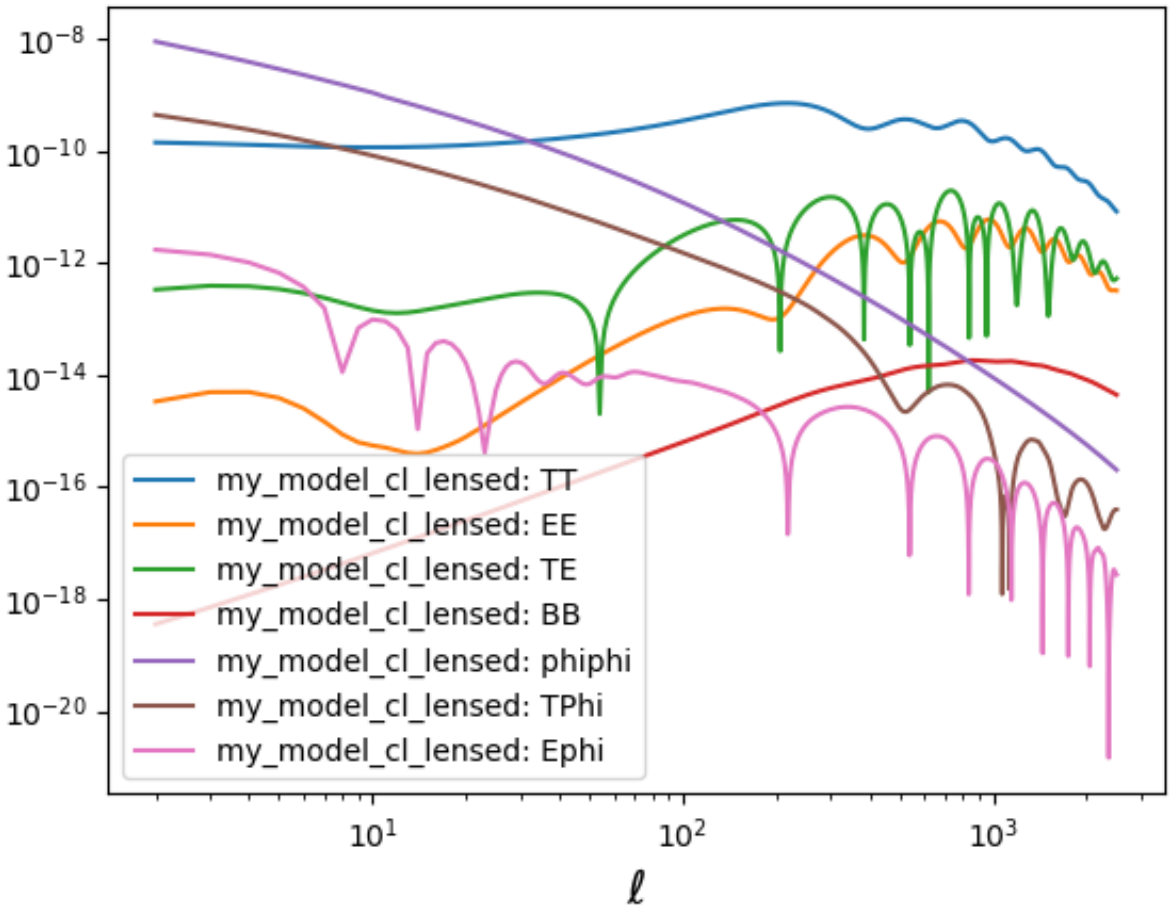
Do quick plots with the python script CPU.py (CLASS Plotting Unit):

```
> python CPU.py --help   # show help for the CPU.py script
```

CLASS Plotting Unit (CPU)

Do quick plots with the python script CPU.py (CLASS Plotting Unit):

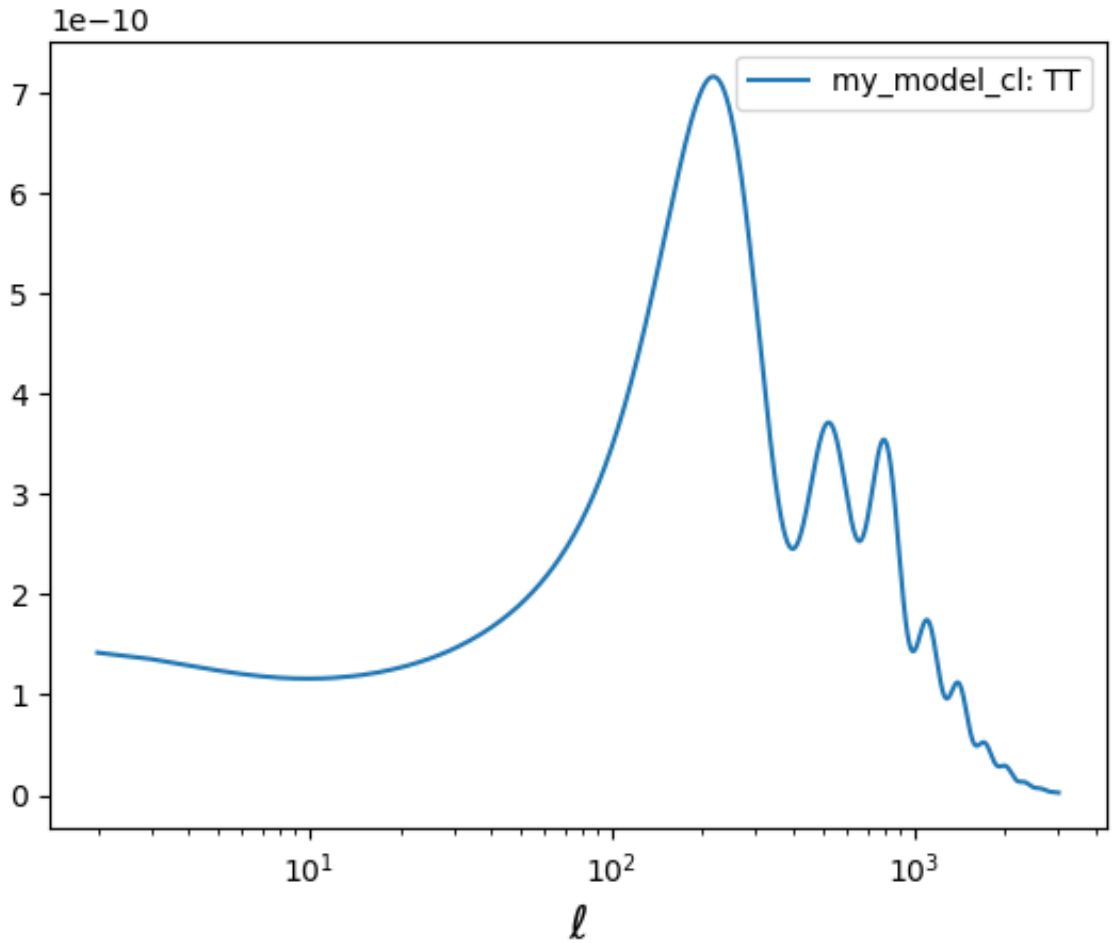
```
> python CPU.py output/my_model_cl_lensed.dat
```



CLASS Plotting Unit (CPU)

Do quick plots with the `python` script `CPU.py` (CLASS Plotting Unit):

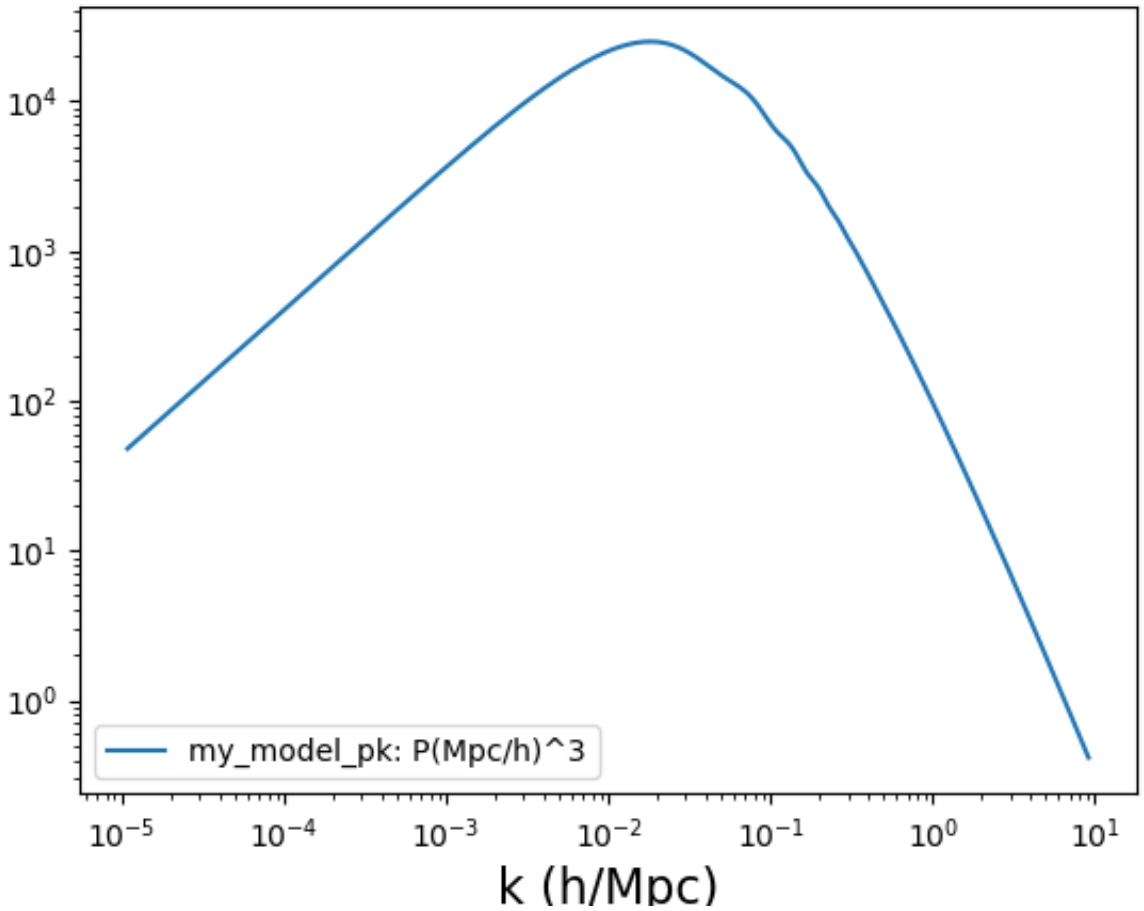
```
> python CPU.py output/my_model_cl.dat -y TT --scale loglin
```



CLASS Plotting Unit (CPU)

Do quick plots with the `python` script `CPU.py` (CLASS Plotting Unit):

```
> python CPU.py output/my_model_pk.dat
```



Running CLASS from python

Not difficult at all – thanks to Cython!

wrapper is based on python/classy.pyx

at compilation, Cython generates and installs the module classy.py

available for import from everywhere!

basics:

`from classy import Class` → import Class
`LambdaCDM = Class()` → create instance of Class
`LambdaCDM.set(...)` → set input parameters
`LambdaCDM.compute()` → execute CLASS (C code!)
`LambdaCDM.lensed_cl(...)` → extract lensed CMB spectrum
`LambdaCDM.pk(...)` → extract matter power spectrum
... → many more functions and parameters are available!

classy warmup

excerpts of scripts/warmup.py (run with > python scripts/warmup.py)

```
from classy import Class
LambdaCDM = Class()
# pass input parameters
LambdaCDM.set({
        'omega_b':0.022032,
        'omega_cdm':0.12038,
        'h':0.67556,
        'A_s':2.215e-9,
        'n_s':0.9619,
        'tau_reio':0.0925,
        'output':'tCl,pCl,lCl,mPk',
        'lensing':'yes',
        'P_k_max_1/Mpc':3.0
    })
# run class
LambdaCDM.compute()
```

classy warmup

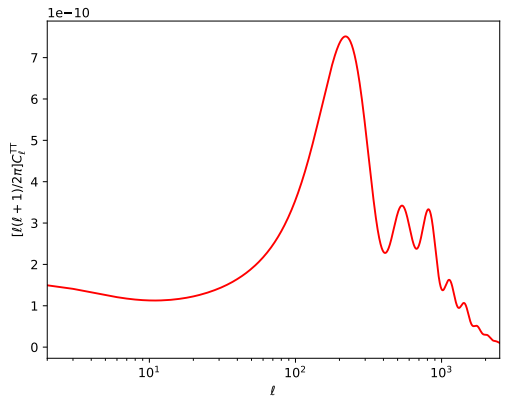
excerpts of scripts/warmup.py (run with > python scripts/warmup.py)

```
# get all C_l output
cls = LambdaCDM.lensed_cl(2500)
# To check the format of cls
cls.keys()
ll = cls['ell'][2:]
clTT = cls['tt'][2:]
clEE = cls['ee'][2:]
clPP = cls['pp'][2:]
```

classy warmup

excerpts of scripts/warmup.py (run with > python scripts/warmup.py)

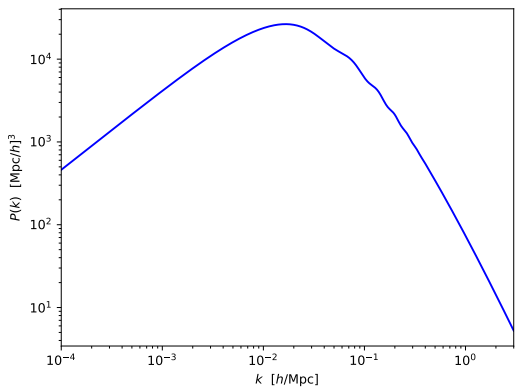
```
import matplotlib.pyplot as plt
from math import pi
# plot TT spectrum
plt.xscale('log');
plt.yscale('linear');
plt.xlim(2,2500)
plt.xlabel(r'$\ell$')
plt.ylabel(r'$[\ell(\ell+1)/2\pi]  C_\ell^\mathrm{TT}$')
plt.plot(ll,clTT*ll*(ll+1)/2./pi,'r-')
plt.savefig('warmup_cltt.pdf')
```



classy warmup

excerpts of scripts/warmup.py (run with > python scripts/warmup.py)

```
# get P(k) at redhsift z=0
import numpy as np
kk = np.logspace(-4,np.log10(3),1000) # k in h/Mpc
Pk = [] # P(k) in (Mpc/h)**3
h = LambdaCDM.h() # get reduced Hubble for conversions to 1/Mp
for k in kk:
    Pk.append(LambdaCDM.pk(k*h,0.)*h**3) # function .pk(k,z)
[...]
plt.plot(kk,Pk,'b-')
plt.savefig('warmup_pk.pdf')
```



Background evolution

Goal: plot the evolution of the background energy densities

Background evolution

Goal: plot the evolution of the background energy densities

Within `python` or a `jupiter` notebook:

[1]:
```
from classy import Class
import matplotlib.pyplot as plt
```

Background evolution

Goal: plot the evolution of the background energy densities

Within `python` or a `jupiter` notebook:

[1]:
```
from classy import Class
import matplotlib.pyplot as plt
```

[2]:
```
model = Class()
help(model) #show methods / attributes
# model.set({...})  # optional
model.compute()
background = model.get_background() # a dictionary!
print(background.keys())
```

Background evolution

Goal: plot the evolution of the background energy densities

Within `python` or a `jupiter` notebook:

[1]:
```
from classy import Class
import matplotlib.pyplot as plt
```

[2]:
```
model = Class()
help(model) #show methods / attributes
# model.set({...})  # optional
model.compute()
background = model.get_background() # a dictionary!
print(background.keys())
```

Out:
```
['z', 'proper time [Gyr]', 'conf. time [Mpc]', 'H [1/Mpc]',
'comov. dist.', 'ang.diam.dist.', 'lum. dist.', 'comov.snd.hrz.',
'(.)rho_g', '(.)rho_b', '(.)rho_cdm', '(.)rho_lambda',
'(.)rho_ur', '(.)rho_crit', 'gr.fac. D', 'gr.fac. f']
```

These are the available background quantities!

Extract with "`background[...]`"

Background evolution

Goal: plot the evolution of the background energy densities

Within `python` or a `jupiter` notebook:

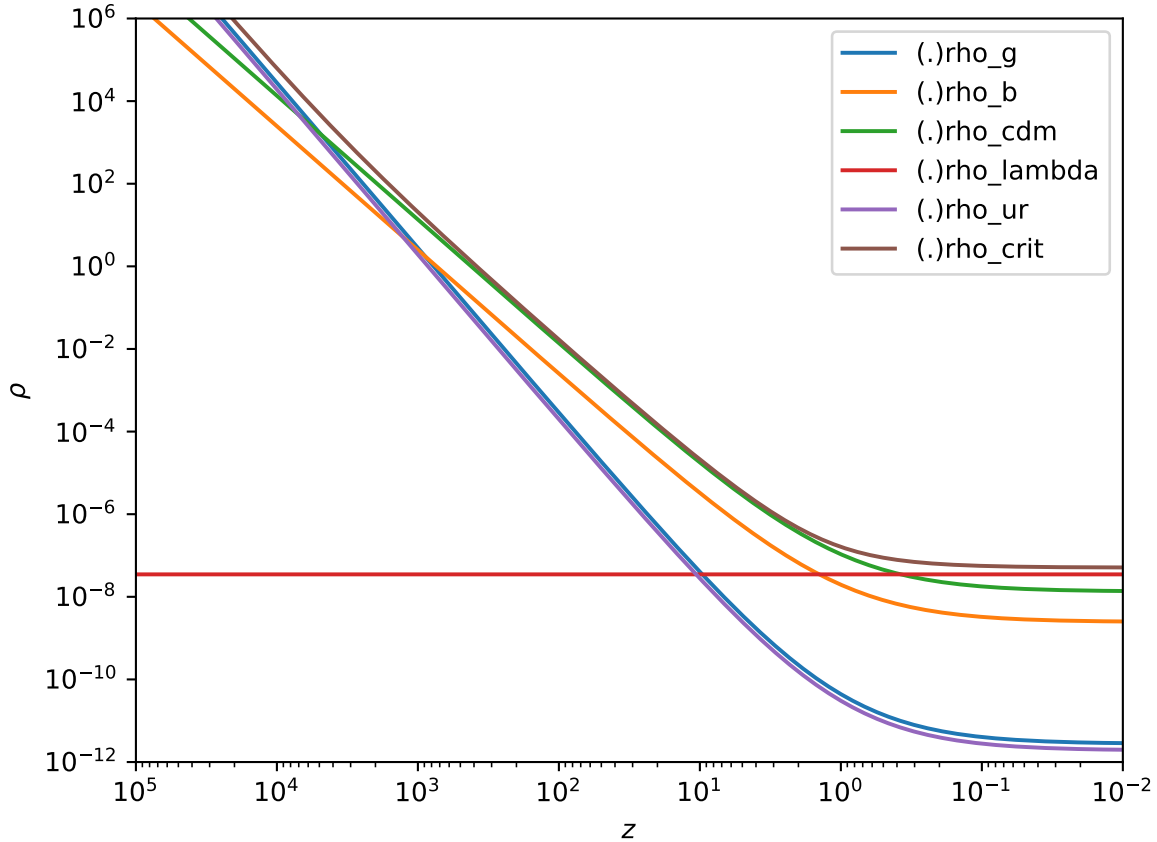
[3]:
```
for r in [
        '(.)rho_g',
        '(.)rho_b',
        '(.)rho_cdm',
        '(.)rho_lambda',
        '(.)rho_ur',
        '(.)rho_crit',
]:
    plt.loglog(
        background["z"], # x: always the "z" array
        background[r],   # y: extract each column
        label=r,         # assign label for legend
    )
plt.xlabel(r"$z$")
plt.ylabel(r"$\rho$")
```

Background evolution

Goal: plot the evolution of the background energy densities

Within `python` or a `jupiter` notebook:

[4]:
```
plt.xlim([1e5,1e-2])  # available range is much larger!
plt.ylim([1e-12,1e6])
plt.legend()
plt.savefig("classy_background.pdf")
```



CMB spectrum as a function of a single parameter

Goal: plot CMB & matter power spectra as functions of one parameter

for example, $\omega_{\rm b}$, $\omega_{\rm cdm}$, H_0

CMB spectrum as a function of a single parameter

Goal: plot CMB & matter power spectra as functions of one parameter

[1]:
```
from classy import Class
import numpy as np
import matplotlib.pyplot as plt
```

CMB spectrum as a function of a single parameter

Goal: plot CMB & matter power spectra as functions of one parameter

[2]:

```
common_settings = {
    'output':'tCl,pCl,lCl,mPk',
    'lensing':'yes',
    # LambdaCDM parameters
    'H0':67.,
    'omega_b':0.022,
    'omega_cdm':0.12,
    'A_s':2.215e-9,
    'n_s':0.9619,
    'tau_reio':0.0925,
    # Take fixed value for primordial Helium
    # (instead of automatic BBN adjustment)
    'YHe':0.246,
    # other output and precision parameters
    'P_k_max_1/Mpc':3.0,
    'l_switch_limber':9
}
kvec = np.logspace(-4,np.log10(3),1000)
```

CMB spectrum as a function of a single parameter

Goal: plot CMB & matter power spectra as functions of one parameter

[3]:
```
params = [
    [
        'omega_cdm',
        np.linspace(0.08, 0.16, 5),
        r'$\omega_\mathrm{cdm}$'
    ],
    [
        'omega_b',
        np.linspace(0.014, 0.03, 5),
        r'$\omega_\mathrm{b}$'
    ],
    [
        'H0',
        np.linspace(50, 90, 5),
        r'$H_0$'
    ],
]
# later:
# for var_name, var_array, var_legend in params:
```

CMB spectrum as a function of a single parameter

Goal: plot CMB & matter power spectra as functions of one parameter

[4]:
```
for var_name, var_array, var_legend in params:
    fig_Pk, ax_Pk = plt.subplots()
    fig_TT, ax_TT = plt.subplots()
    for i, var in enumerate(var_array):
        M = Class()
        M.set(common_settings)
        M.set({var_name: var}) # overwrite previous value
        M.compute()
        clM = M.lensed_cl(2500) # compute Cls
        ll = clM['ell'][2:]
        ax_TT.semilogx(
            ll,
            clM['tt'][2:]*ll*(ll+1)/(2.*np.pi),
            label="%s = %g"%(var_legend, var))
        pkM = [M.pk(k, 0.) for k in kvec] # read pk at z=0
        ax_Pk.loglog(
            kvec,
            np.array(pkM),
            label="%s = %g"%(var_legend, var))
        M.struct_cleanup() # clean Class
        M.empty()
```

CMB spectrum as a function of a single parameter

Goal: plot CMB & matter power spectra as functions of one parameter

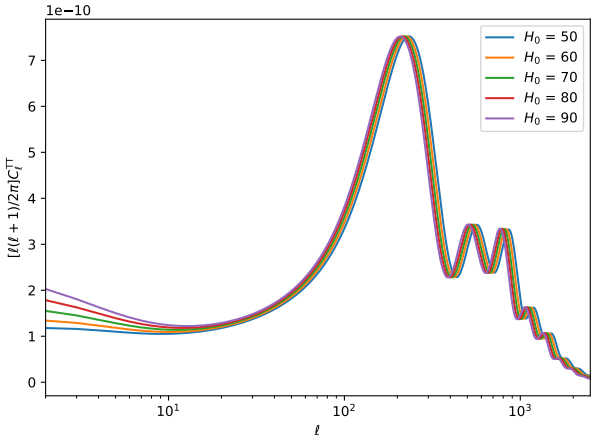
```
    ax_Pk.set_xlim([1.e-4,3.])
    ax_Pk.set_xlabel(r'$k$ [$h$/Mpc]')
    ax_Pk.set_ylabel(r'$P(k)$ [Mpc/$h$]$^3$')
    ax_Pk.legend()
    fig_Pk.tight_layout()
    fig_Pk.savefig('spectra_%s_Pk.pdf' % var_name)
```

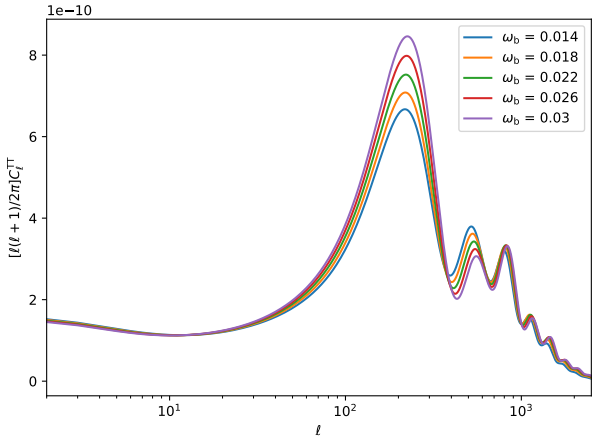
[5]:

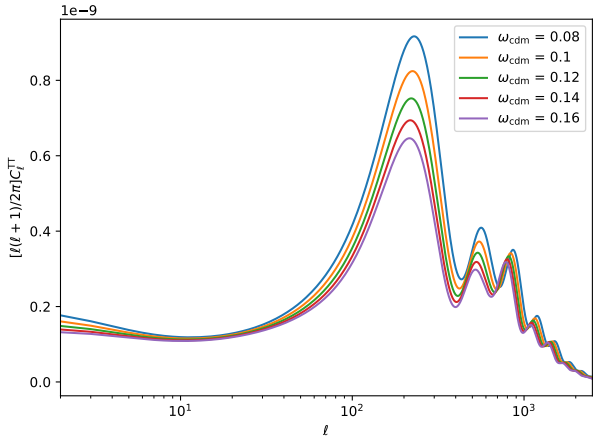
```
    ax_TT.set_xlim([2,2500])
    ax_TT.set_xlabel(r'$\ell$')
    ax_TT.set_ylabel(
        r'$[\ell(\ell+1)/2\pi]C_\ell^\mathrm{TT}$'
    )
    ax_TT.legend()
    fig_TT.tight_layout()
    fig_TT.savefig('spectra_%s_cltt.pdf' % var_name)
```

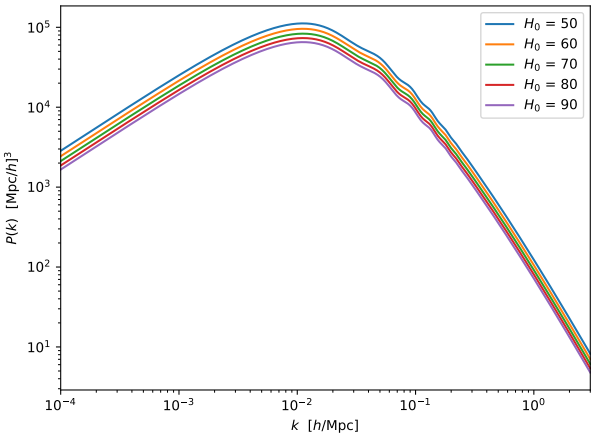
CMB spectrum as a function of a single parameter

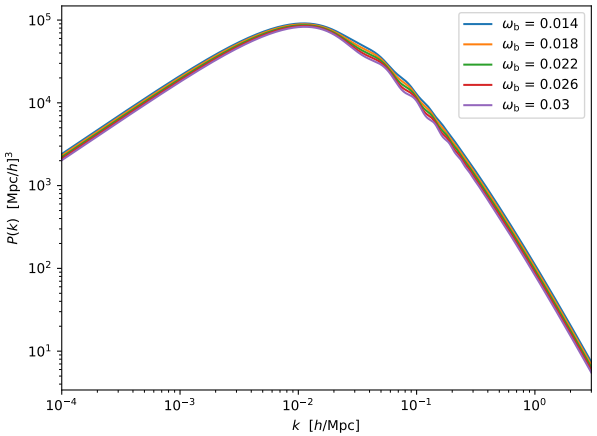
Goal: plot CMB & matter power spectra as functions of one parameter

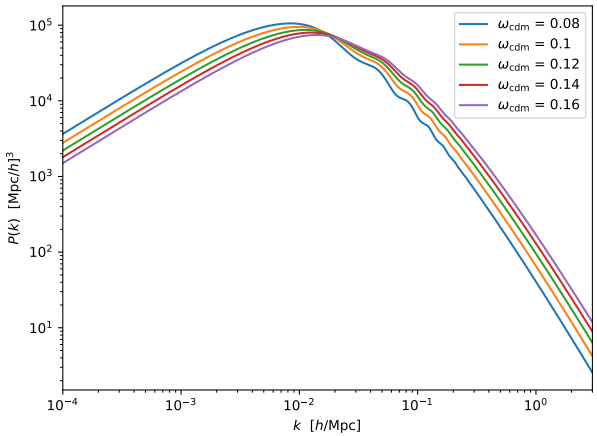












Contributions to the TT spectrum

Goal: plot different CMB contributions (SW, Integrated SW, Doppler)

Contributions to the TT spectrum

Goal: plot different CMB contributions (SW, Integrated SW, Doppler)

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from classy import Class
```

Contributions to the TT spectrum

Goal: plot different CMB contributions (SW, Integrated SW, Doppler)

[2]:

```
common_settings = {
    # which output? lensing requires (tCl or pCl) + lCl
    'output':'tCl,pCl,lCl',
    'lensing':'yes',
    # LambdaCDM parameters
    'h':0.67556,
    'omega_b':0.022032,
    'omega_cdm':0.12038,
    'A_s':2.215e-9,
    'n_s':0.9619,
    'tau_reio':0.0925,
    # Take fixed value for primordial Helium
    # (instead of automatic BBN adjustment)
    'YHe':0.246,
    # other output and precision parameters
    'l_max_scalars':5000
}
```

Contributions to the TT spectrum

Goal: plot different CMB contributions (SW, Integrated SW, Doppler)

[3]:
```
# dict keys are ``temperature contributions'' options
contribs = {
    "tsw": None,  # intrinsic temperature corrected by SW
    "eisw": None, # early Integrated SW
    "lisw": None, # late ISW
    "dop": None,  # Doppler
}
# colors for plot for each component
colors = {"tsw": "c", "eisw": "g", "lisw": "m", "dop": "b"}
```

Contributions to the TT spectrum

Goal: plot different CMB contributions (SW, Integrated SW, Doppler)

[4]:
```
M = Class()
M.set(common_settings)
M.compute() # all contributions are included by default
cl_tot = M.raw_cl(3000) # non-lensed Cls
cl_lensed = M.lensed_cl(3000)
M.struct_cleanup()  # clean output
M.empty()           # clean input
```

Contributions to the TT spectrum

Goal: plot different CMB contributions (SW, Integrated SW, Doppler)

[5]:
```
for c in contribs.keys():
    M.set(common_settings)
    # select only one contribution:
    M.set({'temperature contributions': c})
    M.compute()
    # store raw Cls in the dict
    contribs[c] = M.raw_cl(3000)
    M.struct_cleanup()
    M.empty()
```

Contributions to the TT spectrum

Goal: plot different CMB contributions (SW, Integrated SW, Doppler)

[6]:
```
ell = cl_tot['ell']
factor = 1.e10 * ell * (ell + 1.) / (2. * np.pi)
plt.semilogx( # plot total raw Cl
    ell,
    factor*cl_tot['tt'],
    'k-',
    label="total"
)
for c in contribs.keys():
    plt.semilogx( # plot each contribution
        ell,
        factor*contribs[c]['tt'],
        "%s:"%colors[c],
        label=c
    )
plt.semilogx( # plot total lensed Cls
    ell,
    factor*cl_lensed['tt'],
    'r--',
    label="lensed"
)
```

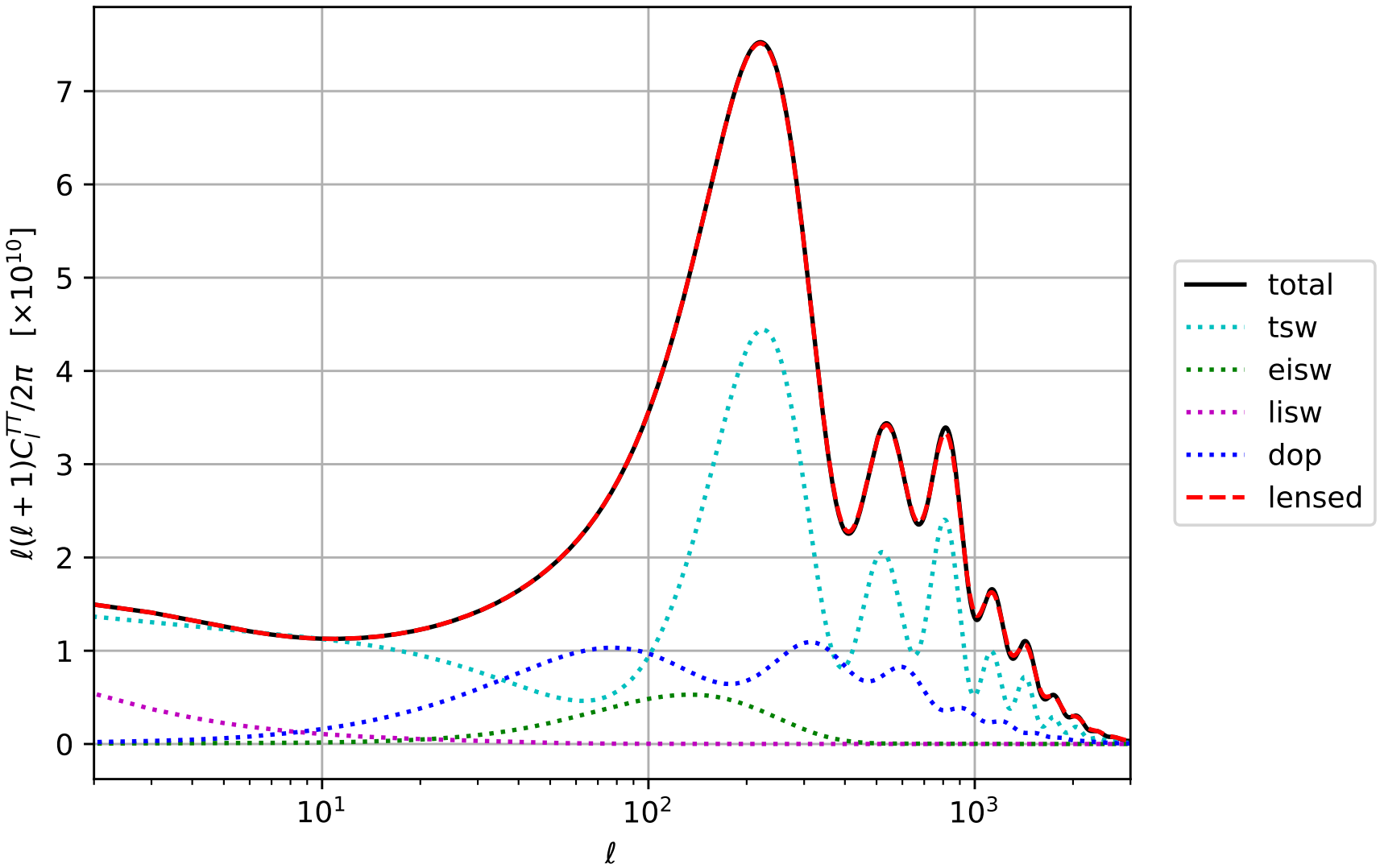
Contributions to the TT spectrum

Goal: plot different CMB contributions (SW, Integrated SW, Doppler)

[7]:
```
# finalize and save plot
plt.xlim([2,3000])
plt.xlabel(r"$\ell$")
plt.ylabel(
    r"$\ell (\ell+1) C_l^{TT} / 2 \pi \quad[\times 10^{10}]$"
)
plt.grid()
plt.legend(loc='right', bbox_to_anchor=(1.25, 0.5))
plt.savefig('cltt_terms.pdf', bbox_inches='tight')
```

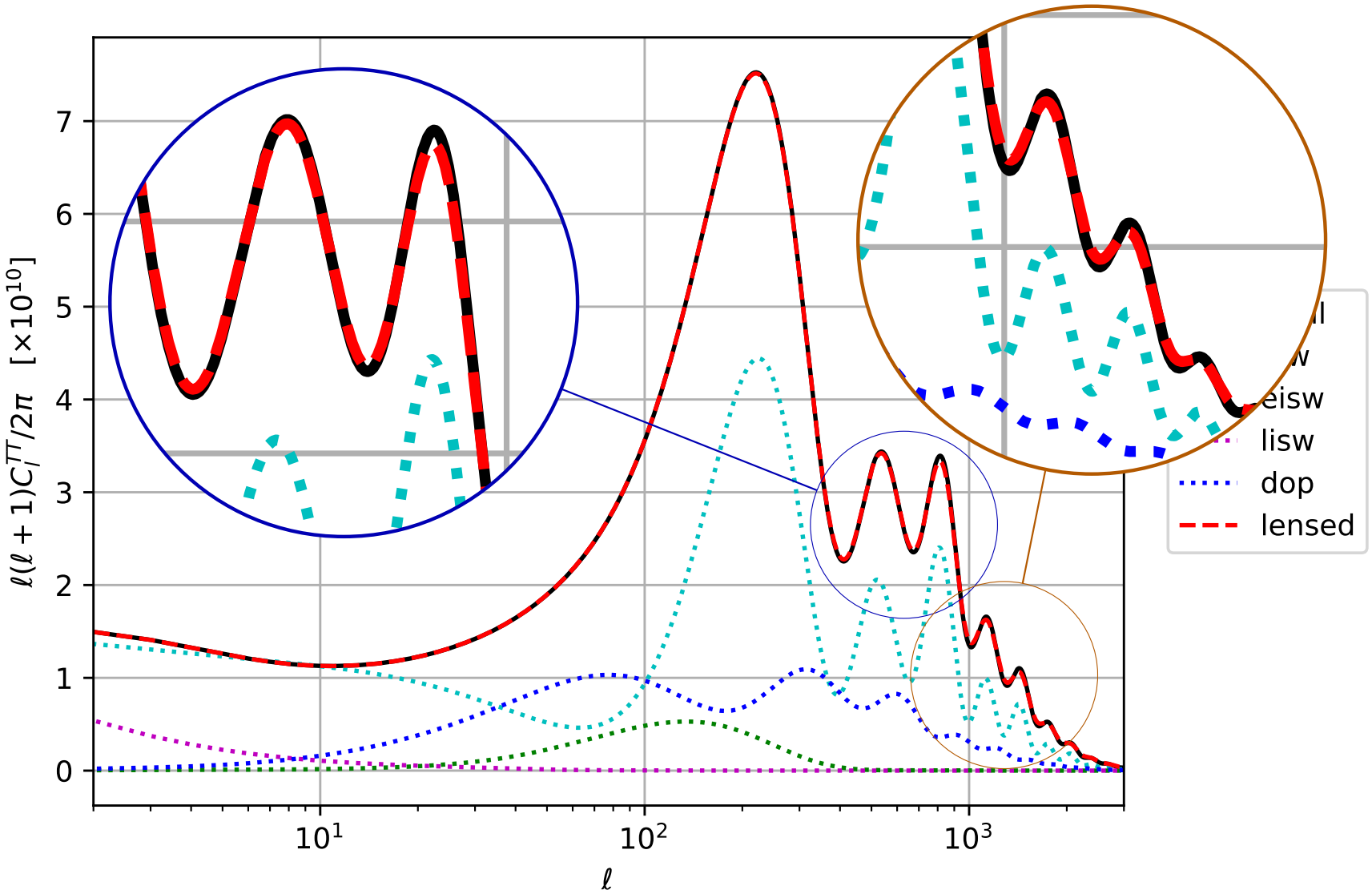
Contributions to the TT spectrum

Goal: plot different CMB contributions (SW, Integrated SW, Doppler)



Contributions to the TT spectrum

Goal: plot different CMB contributions (SW, Integrated SW, Doppler)



Using CLASS for data analysis

Many cosmological data are available for being analysed

no need to write complicated codes to call CLASS and then compare theory to data: **they already exist**!

Using CLASS for data analysis

Many cosmological data are available for being analysed

no need to write complicated codes to call CLASS and then compare theory to data: **they already exist**!

MontePython

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

- Monte Carlo Markov Chain plus external samplers
- interfaced with CLASS
- called by command line, needs input file (examples provided)
- many cosmological likelihoods ready for use
- can use mock data
- output analysis with `info` command

Using CLASS for data analysis

Many cosmological data are available for being analysed

no need to write complicated codes to call CLASS and then compare theory to data: **they already exist**!

MontePython

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

- Monte Carlo Markov Chain plus external samplers
- interfaced with CLASS
- called by command line, needs input file (examples provided)
- many cosmological likelihoods ready for use
- can use mock data
- output analysis with `info` command

Cobaya

https://github.com/CobayaSampler/cobaya

- Monte Carlo Markov Chain plus external samplers
- interfaced with CLASS, CAMB
- can be used from command line or from `python` directly (see docs)
- many cosmological likelihoods ready for use, easy to define new likelihoods on the fly
- can be used without cosmological codes/data as well

Summary

Here you find a summary of the most important links/commands!

Summary

Here you find a summary of the most important links/commands!

CLASS, C code: `https://class-code.net`
`https://github.com/lesgourg/class_public/wiki`
`https://lesgourg.github.io/courses.html`

Summary

Here you find a summary of the most important links/commands!

CLASS, C code:

```
https://class-code.net
https://github.com/lesgourg/class_public/wiki
https://lesgourg.github.io/courses.html
```

Install:

```
git clone http://github.com/lesgourg/class_public.git class
cd class/
make clean ; make -j
```

Summary

Here you find a summary of the most important links/commands!

CLASS, C code:

```
https://class-code.net
https://github.com/lesgourg/class_public/wiki
https://lesgourg.github.io/courses.html
```

Install:

```
git clone http://github.com/lesgourg/class_public.git class
cd class/
make clean ; make -j
```

Run C code:

```
./class my_model.ini
```

see `explanatory.ini` for list of parameters

Summary

Here you find a summary of the most important links/commands!

CLASS, C code:

```
https://class-code.net
https://github.com/lesgourg/class_public/wiki
https://lesgourg.github.io/courses.html
```

Install:

```
git clone http://github.com/lesgourg/class_public.git class
cd class/
make clean ; make -j
```

Run C code:

```
./class my_model.ini
```

see `explanatory.ini` for list of parameters

CPU:

```
python CPU.py --help
```

Summary

Here you find a summary of the most important links/commands!

CLASS, C code:

```
https://class-code.net
https://github.com/lesgourg/class_public/wiki
https://lesgourg.github.io/courses.html
```

Install:

```
git clone http://github.com/lesgourg/class_public.git class
cd class/
make clean ; make -j
```

Run C code:

```
./class my_model.ini
```

see `explanatory.ini` for list of parameters

CPU:

```
python CPU.py --help
```

classy:

`from classy import Class` → import Class
`M = Class()` → create instance of Class
`M.set(...)` → set input parameters
`M.compute()` → execute CLASS (C code!)
`M.lensed_cl(...)` → extract lensed CMB spectrum
`M.pk(...)` → extract matter power spectrum
... → many more functions and parameters are available!
`M.struct_cleanup(...)` → clean output
`M.empty(...)` → clean input

Summary

Here you find a summary of the most important links/commands!

CLASS, C code:
`https://class-code.net`
`https://github.com/lesgourg/class_public/wiki`
`https://lesgourg.github.io/courses.html`

Install:
```
git clone http://github.com/lesgourg/class_public.git class
cd class/
make clean ; make -j
```

Run C code:
`./class my_model.ini`
see `explanatory.ini` for list of parameters

CPU:
`python CPU.py --help`

classy:
`from classy import Class` → import Class
`M = Class()` → create instance of Class
`M.set(...)` → set input parameters
`M.compute()` → execute CLASS (C code!)
`M.lensed_cl(...)` → extract lensed CMB spectrum
`M.pk(...)` → extract matter power spectrum
... → many more functions and parameters are available!
`M.struct_cleanup(...)` → clean output
`M.empty(...)` → clean input

data analysis:
`https://github.com/brinckmann/montepython_public/`
`https://github.com/CobayaSampler/cobaya`