Calling axionCAMB from a Python Script

The following changes are designed to make axionCAMB receive six specific parameters, but the process is replicable for any parameter provided that you find where it gets read-in. It is a good idea to comment your name or initials (something easily searchable) wherever you make changes to axionCAMB files so that you can find them later.

1. Open the inidriver

```
emacs inidriver_axion.F90
```

2. Line 34 should be modified to read as follows

```
integer i, int1, int2, int3, int4, int5, int6
```

3. Insert the following code at line 40

```
character(LEN=1) buffer1, buffer3, buffer5, buffer7, buffer9,
buffer11
character(LEN=23) buffer2, buffer4, buffer6, buffer8, buffer10,
buffer12
double precision ombh2, omch2, omaxh2, mass_axion, h0, ns
```

4. Immediately after “End axion stuff” (around line 62) insert the following code

```
call getarg(2,buffer1)
read(buffer1,*) int1
call getarg(3,buffer2)
read(buffer2,*) ombh2

call getarg(4,buffer3)
read(buffer3,*) int2
call getarg(5,buffer4)
read(buffer4,*) omch2

call getarg(6,buffer5)
read(buffer5,*) int3
call getarg(7,buffer6)
read(buffer6,*) omaxh2

call getarg(8,buffer7)
read(buffer7,*) int4
call getarg(9,buffer8)
read(buffer8,*) mass_axion
```

```
call getarg(10,buffer9)
read(buffer9,*) int5
call getarg(11,buffer10)
read(buffer10,*) h0

call getarg(12,buffer11)
read(buffer11,*) int6
call getarg(13,buffer12)
read(buffer12,*) ns
```

Commentary: Steps 2-4 tell axionCAMB to expect additional arguments to be fed to it when it is called from the command line. We will later use these arguments to easily run axionCAMB with variable parameter values.

5. Immediately before "call Recombination_ReadParams(P%Recomb, DefIni)" (around line 357) insert the following code

```
if (int6.eq.6) then
P%InitPower%an(1) = ns
endif
```

Step 5 redefines "ns" immediately before inidriver_axion.F90 exports it to be used by a separate script for computing recombination.

6. Immediately after the block of text that begins with "DM: The place axion evolution is called" (around line 511) insert the following code

```
if (int5.eq.5) then
P%H0 = h0
endif

if (int4.eq.4) then
P%ma = mass_axion
endif

if (int1.eq.1) then
P%omegab = ombh2/((P%H0/100.d0)**2.d0)
endif

if (int3.eq.3) then
P%omegaax = omaxh2/((P%H0/100.d0)**2.d0)
endif

if (int2.eq.2) then
P%omegac = omch2/(P%H0/100.d0)**2.d0
```

```
endif

P%omegav = 1.0d0-P%omegab-P%omegac - P%omegan -P%omegak-
P%omegaax-P%omegah2_rad/((P%H0/1.d2)**2.0d0)
```

Step 6 serves the same purpose as Step 5 but with different parameters. It is important that the Hubble constant be defined first since it is subsequently used to convert from lowercase omega to uppercase omega. The last line redefines the dark energy content so that we have a critical density universe (all of the uppercase omegas sum to one).

7. Recompile axionCAMB

```
make

make clean
```

8. Run axionCAMB from the command line via

```
./camb params.ini 1 # 2 # 3 # 4 # 5 # 6 #
```

where # is where you should input the desired value of the parameter corresponding to the preceding number. For example, the value of the Hubble constant should be input as the # immediately following 5. The numbers correspond to the parameters as such:

1 ombh^2 (baryon energy density)
2 omch^2 (cold dark matter energy density)
3 omaxh^2 (axion energy density)
4 mass_axion (axion mass)
5 H (Hubble constant)
6 ns (scalar tilt)

9. Open a new python script

```
emacs script.py
```

and import the subprocess package

```
import subprocess
```

which allows you to code command line tasks that will be executed when you run your python script.

10. Make a parameter value variable by setting it to %s within a subprocess command and then specifying %s to loop through a list of values as such:

```
H = [50.0,100.0]
for i in range(len(H)):
    subprocess.call('./camb params.ini 1 # 2 # 3 # 4 # 5 %s 6
#' %(H[i]), shell = True)
    subprocess.call('mv test_matterpower.dat H%s.dat' %(i),
shell = True)
```

The second subprocess command redefines the data file so that the data from the first run is not lost when test_matterpower.dat is overwritten on the second run.

11. Import the data files as such:

```
from numpy import *
k1, Pk1 = loadtxt('H1.dat', unpack = True, usecols = [0,1])
k2, Pk2 = loadtxt('H2.dat', unpack = True, usecols = [0,1])
```

12. Plot the data as such:

```
import matplotlib.pyplot as plt
fix,ax = plt.subplots()
ax.plot(k1,Pk1,label = 'H = 50')
ax.plot(k2,Pk2,label = 'H = 100')
plt.xscale('log')
plt.yscale('log')
plt.legend()
plt.savefig('script.png')
```

This is just a minimum requirement for plotting. Adding the following can produce a high-quality plot:

```
import matplotlib as mpl
font = {'family':'serif','weight':'normal','size':12}
mpl.rcParams['xtick.major.size'] = 7
mpl.rcParams['xtick.major.width'] = 1
mpl.rcParams['xtick.minor.size'] = 3
mpl.rcParams['xtick.minor.width'] = 1
mpl.rcParams['ytick.major.size'] = 7
mpl.rcParams['ytick.major.width'] = 1
mpl.rcParams['ytick.minor.size'] = 3
mpl.rcParams['ytick.minor.width'] = 1
mpl.rcParams['axes.labelsize'] = 16
mpl.rc('font', **font)
```

```
fix,ax = plt.subplots()
ax.plot(k1,Pk1,label = r'$H = 50 km/s/Mpc$')
ax.plot(k2,Pk2,label = r'$H = 100 km/s/Mpc$')
plt.xlabel(r'$k [h/Mpc]$')
plt.ylabel(r'$P(k)[(Mpc/h)^3]$')
plt.xscale('log')
plt.yscale('log')
plt.title('Matter Power Spectrum')
plt.legend(frameon = False)
plt.savefig('Pk.png')
```

The first part of this addendum sets the font and axis settings. The second part adds axis labels, a title, removes the frame from the legend, and makes us of equation typesetting by placing text between $'s with an "r" out front. For a production run, save the figure as a PDF instead of a PNG.