primordial Documentation

Release 0.0.13

Will Handley

Oct 11, 2018

Contents

primordial inflationary equation solver

Author Will Handley

Version 0.0.13

Homepage https://github.com/williamjameshandley/primordial

Documentation http://primordial.readthedocs.io/

CHAPTER 1

Description

`primordial` is a python package for solving cosmological inflationary equations.

It is very much in beta stage, and currently being built for research purposes.

CHAPTER 2

Example Usage

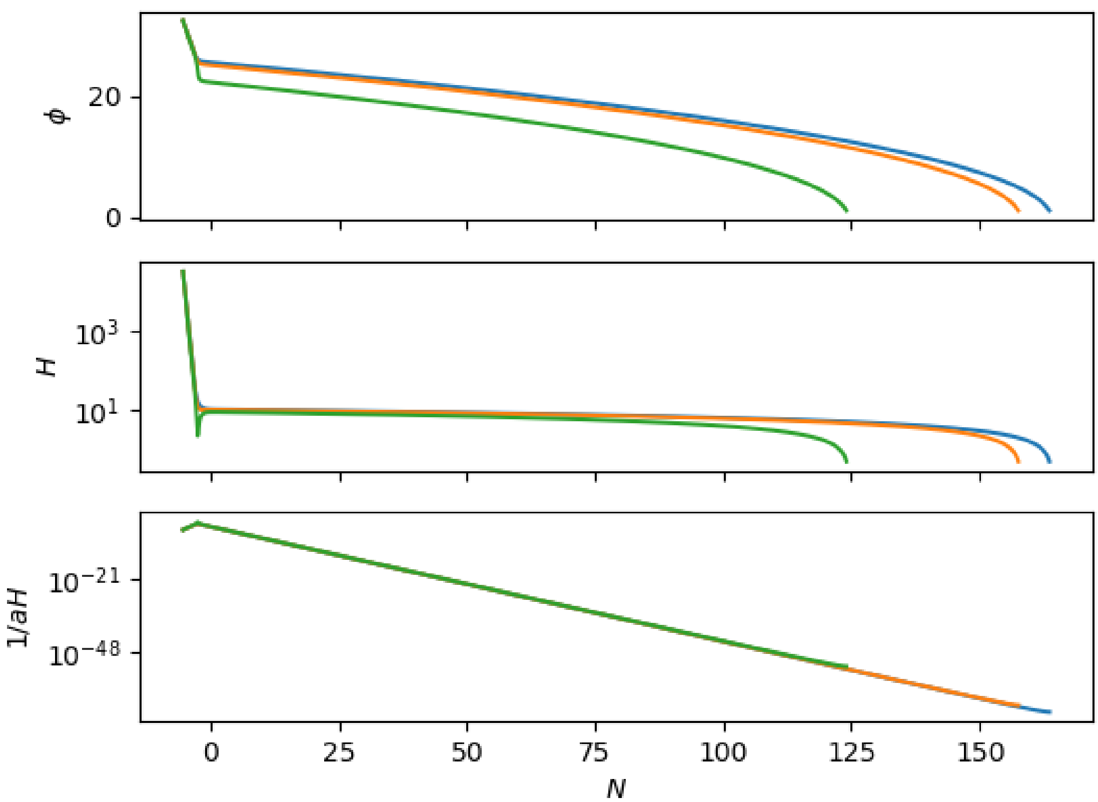
2.1 Plot Background evolution

```
import numpy
import matplotlib.pyplot as plt
from primordial.solver import solve
from primordial.equations.inflation_potentials import ChaoticPotential
from primordial.equations.t.inflation import Equations, KD_initial_conditions
from primordial.equations.events import Inflation, Collapse

fig, ax = plt.subplots(3,sharex=True)
for K in [-1, 0, +1]:
    m = 1
    V = ChaoticPotential(m)
    equations = Equations(K, V)

    events= [Inflation(equations),                     # Record inflation entry and␣
↪exit
             Inflation(equations, -1, terminal=True), # Stop on inflation exit
             Collapse(equations, terminal=True)]       # Stop if universe stops␣
↪expanding

    N_p = -1.5
    phi_p = 23
    t_p = 1e-5
    ic = KD_initial_conditions(t_p, N_p, phi_p)
    t = numpy.logspace(-5,10,1e6)

    sol = solve(equations, ic, t_eval=t, events=events)

    ax[0].plot(sol.N(t),sol.phi(t))
    ax[0].set_ylabel(r'$\phi$')

    ax[1].plot(sol.N(t),sol.H(t))
```

(continues on next page)

(continued from previous page)

```
    ax[1].set_yscale('log')
    ax[1].set_ylabel(r'$H$')

    ax[2].plot(sol.N(t),1/(sol.H(t)*numpy.exp(sol.N(t))))
    ax[2].set_yscale('log')
    ax[2].set_ylabel(r'$1/aH$')

ax[-1].set_xlabel('$N$')
```



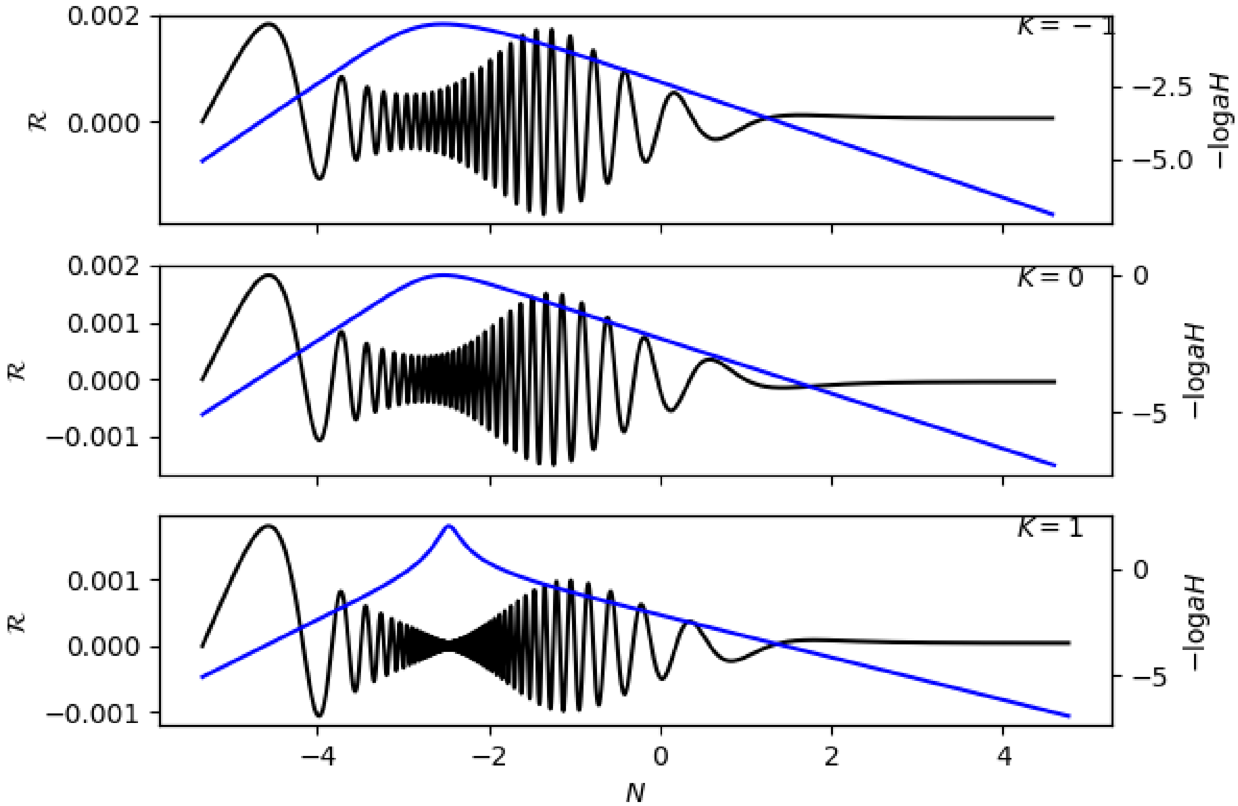
2.2 Plot mode function evolution

```
import numpy
import matplotlib.pyplot as plt
from primordial.solver import solve
from primordial.equations.inflation_potentials import ChaoticPotential
from primordial.equations.t.mukhanov_sasaki import Equations, KD_initial_conditions
from primordial.equations.events import Inflation, Collapse, ModeExit

fig, axes = plt.subplots(3,sharex=True)
for ax, K in zip(axes, [-1, 0, +1]):
    ax2 = ax.twinx()
    m = 1
```

(continues on next page)

(continued from previous page)

```
    V = ChaoticPotential(m)
    k = 100
    equations = Equations(K, V, k)

    events= [
            Inflation(equations),                    # Record inflation entry and exit
            Collapse(equations, terminal=True),      # Stop if universe stops␣
↪expanding
            ModeExit(equations, +1, terminal=True, value=1e1*k)    # Stop on mode exit
            ]


    N_p = -1.5
    phi_p = 23
    t_p = 1e-5
    ic = KD_initial_conditions(t_p, N_p, phi_p)
    t = numpy.logspace(-5,10,1e6)

    sol = solve(equations, ic, t_eval=t, events=events)

    N = sol.N(t)
    ax.plot(N,sol.R1(t), 'k-')
    ax2.plot(N,-numpy.log(sol.H(t))-N, 'b-')

    ax.set_ylabel('$\mathcal{R}$')
    ax2.set_ylabel('$-\log aH$')

    ax.text(0.9, 0.9, r'$K=%i$' % K, transform=ax.transAxes)

axes[-1].set_xlabel('$N$')
```


0.002
0.000
$\mathcal{R}$
$K=-1$
−2.5
−5.0
−logaH
0.002
0.001
0.000
−0.001
$K=0$
0
−5
0.001
0.000
−0.001
$K=1$
−4
−2
0
2
4
N

CHAPTER 3

To do list

Eventually would like to submit this to JOSS. Here are things to do before then:

3.1 Cosmology

- Slow roll initial conditions
- Mukhanov Sazaki evolution in N
- add η as independent variable
- add ϕ as independent variable

3.2 Code

- Documentation
- **Tests**
  - 100% coverage
  - interpolation
  - cosmology

CHAPTER 4

primordial package

4.1 Subpackages

4.1.1 primordial.equations package

Subpackages

primordial.equations.N package

Submodules

primordial.equations.N.cosmology module

class primordial.equations.N.cosmology.**Equations**(*H0*, *Omega_r*, *Omega_m*, *Omega_k*, *Omega_l*)

Bases: *primordial.equations.cosmology.Equations*

Cosmology equations in time

Solves background variables in cosmic time for curved and flat universes using the Friedmann equation.

Independent variable: N: efolds

Variables: t: cosmic time

Methods

H(t, y)	Hubble parameter
H2(t, y)	The square of the Hubble parameter, computed using the Friedmann equation

Continued on next page

Table 1 – continued from previous page

`__call__`(N, y)	The derivative function for underlying variables, computed using the Klein-Gordon equation
`add_variable`(*args)	Add dependent variables to the equations
`set_independent_variable`(name)	Set name of the independent variable
`sol`(sol, **kwargs)	Post-process solution of solve_ivp

class primordial.equations.N.cosmology.**initial_conditions**(*Ni*)
Bases: `object`

Methods

__call__	

primordial.equations.N.inflation module

class primordial.equations.N.inflation.**Equations**(*K*, *potential*)
Bases: *primordial.equations.inflation.Equations*

Background equations in time

Solves bacgkround variables in cosmic time for curved and flat universes using the Klein-Gordon and Friedmann equations.

Independent variable: N: e-folds (log a)

Variables: phi: inflaton field dphi: d/dN (phi) t: cosmic time

Methods

`H`(t, y)	Hubble parameter
H2(N, y)	The square of the Hubble parameter, computed using the Friedmann equation
`V`(t, y)	Potential
`__call__`(N, y)	The derivative function for underlying variables, computed using the Klein-Gordon equation
`add_variable`(*args)	Add dependent variables to the equations
`dVdphi`(t, y)	Potential derivative
dlogH(N, y)	d/dN log H
inflating(N, y)	Inflation diagnostic
`set_independent_variable`(name)	Set name of the independent variable
`sol`(sol, **kwargs)	Post-process solution of solve_ivp

H2(*N*, *y*)
The square of the Hubble parameter, computed using the Friedmann equation

dlogH(*N*, *y*)
d/dN log H

inflating(*N*, *y*)
Inflation diagnostic

class primordial.equations.N.inflation.**Inflation_start_initial_conditions**(*N_e*, *phi_e*)

Bases: object

Methods

__call__	

primordial.equations.N.mukhanov_sasaki module

class primordial.equations.N.mukhanov_sasaki.**Equations**(*K*, *potential*, *k*)

Bases: *primordial.equations.N.inflation.Equations*

Methods

H(t, y)	Hubble parameter
H2(N, y)	The square of the Hubble parameter, computed using the Friedmann equation
V(t, y)	Potential
__call__(N, y)	The derivative function for underlying variables, computed using the Mukhanov-Sasaki equation equation
add_variable(*args)	Add dependent variables to the equations
dVdphi(t, y)	Potential derivative
dlogH(N, y)	d/dN log H
inflating(N, y)	Inflation diagnostic
set_independent_variable(name)	Set name of the independent variable
sol(sol, **kwargs)	Post-process solution of solve_ivp

class primordial.equations.N.mukhanov_sasaki.**Inflation_start_initial_conditions**(*N_e*, *phi_e*)

Bases: *primordial.equations.N.inflation.Inflation_start_initial_conditions*

Methods

__call__	

Module contents

primordial.equations.t package

Submodules

primordial.equations.t.cosmology module

class primordial.equations.t.cosmology.**Equations**(*H0*, *Omega_r*, *Omega_m*, *Omega_k*, *Omega_l*)

Bases: *primordial.equations.cosmology.Equations*

Cosmology equations in time

Solves background variables in cosmic time for curved and flat universes using the Friedmann equation.

Independent variable: t: cosmic time

Variables: N: efolds

Methods

H(t, y)	Hubble parameter
H2(t, y)	The square of the Hubble parameter, computed using the Friedmann equation
__call__(t, y)	The derivative function for underlying variables, computed using the Klein-Gordon equation
add_variable(*args)	Add dependent variables to the equations
set_independent_variable(name)	Set name of the independent variable
sol(sol, **kwargs)	Post-process solution of solve_ivp

class primordial.equations.t.cosmology.**initial_conditions**(*Ni*)

Bases: object

Methods

__call__	

primordial.equations.t.inflation module

class primordial.equations.t.inflation.**Equations**(*K*, *potential*)

Bases: *primordial.equations.inflation.Equations*

Background equations in time

Solves bacgkround variables in cosmic time for curved and flat universes using the Klein-Gordon and Friedmann equations.

Independent variable: t: cosmic time

Variables: N: efolds phi: inflaton field dphi: d (phi) / dt

Methods

H(t, y)	Hubble parameter
H2(t, y)	The square of the Hubble parameter, computed using the Friedmann equation
V(t, y)	Potential
__call__(t, y)	The derivative function for underlying variables, computed using the Klein-Gordon equation
add_variable(*args)	Add dependent variables to the equations
dVdphi(t, y)	Potential derivative
inflating(t, y)	Inflation diagnostic
set_independent_variable(name)	Set name of the independent variable
sol(sol, **kwargs)	Post-process solution of solve_ivp

H2 (*t*, *y*)
The square of the Hubble parameter, computed using the Friedmann equation

inflating (*t*, *y*)
Inflation diagnostic

class primordial.equations.t.inflation.**Inflation_start_initial_conditions** (*N_e*, *phi_e*)
Bases: object

Methods

__call__	

class primordial.equations.t.inflation.**KD_initial_conditions** (*t0*, *N_p*, *phi_p*)
Bases: object

Methods

__call__	

primordial.equations.t.mukhanov_sasaki module

class primordial.equations.t.mukhanov_sasaki.**Equations** (*K*, *potential*, *k*)
Bases: *primordial.equations.t.inflation.Equations*

Methods

H(t, y)	Hubble parameter
H2(t, y)	The square of the Hubble parameter, computed using the Friedmann equation
V(t, y)	Potential

Continued on next page

Table 6 – continued from previous page

`__call__`(t, y)	The derivative function for underlying variables, computed using the Mukhanov-Sasaki equation equation
`add_variable`(*args)	Add dependent variables to the equations
`dVdphi`(t, y)	Potential derivative
`inflating`(t, y)	Inflation diagnostic
`set_independent_variable`(name)	Set name of the independent variable
`sol`(sol, **kwargs)	Post-process solution of solve_ivp

class `primordial.equations.t.mukhanov_sasaki.`**`Inflation_start_initial_conditions`**(*N_e*, *phi_e*)

Bases: *`primordial.equations.t.inflation.Inflation_start_initial_conditions`*

Methods

__call__	

class `primordial.equations.t.mukhanov_sasaki.`**`KD_initial_conditions`**(*t0*, *N_p*, *phi_p*)

Bases: *`primordial.equations.t.inflation.KD_initial_conditions`*

Methods

__call__	

Module contents

Submodules

primordial.equations.cosmology module

class `primordial.equations.cosmology.`**`Equations`**(*H0*, *Omega_r*, *Omega_m*, *Omega_k*, *Omega_l*)

Bases: *`primordial.equations.equations.Equations`*

Cosmology equations

Solves background variables in cosmic time for curved and flat universes using the Friedmann equation.

Independent variable: N: efolds

Variables: t: cosmic time

Methods

`H`(t, y)	Hubble parameter
`H2`(t, y)	The square of the Hubble parameter, computed using the Friedmann equation

Continued on next page

Table 7 – continued from previous page

`__call__`(t, y)	Vector of derivatives
`add_variable`(*args)	Add dependent variables to the equations
`set_independent_variable`(name)	Set name of the independent variable
`sol`(sol, **kwargs)	Post-process solution of solve_ivp

H (*t*, *y*)
Hubble parameter

H2 (*t*, *y*)
The square of the Hubble parameter, computed using the Friedmann equation

sol (*sol*, ***kwargs*)
Post-process solution of solve_ivp

primordial.equations.equations module

class primordial.equations.equations.**Equations**
Bases: `object`

Base class for equations.

Allows one to compute derivatives and derived variables. Most of the other classes take 'equations' as an object.

Attributes

i [dict] dictionary mapping variable names to indices in the solution vector

independent_variable [string] name of independent variable

Methods

`__call__`(t, y)	Vector of derivatives
`add_variable`(*args)	Add dependent variables to the equations
`set_independent_variable`(name)	Set name of the independent variable
`sol`(sol, **kwargs)	Amend solution from from solve_ivp

add_variable (**args*)
Add dependent variables to the equations

- creates an index for the location of variable in y
- creates a class method of the same name with signature name(self, t, y) that should be used to extract the variable value in an index-independent manner.

Parameters

***args** [str] Name of the dependent variables

set_independent_variable (*name*)
Set name of the independent variable

Parameters

name [str] Name of the independent variable

sol(*sol*, ***kwargs*)
Amend solution from from solve_ivp

primordial.equations.events module

class primordial.equations.events.**Collapse**(*equations*, *direction=0*, *terminal=False*, *value=0*)
Bases: *primordial.equations.events.Event*

Tests if H^2 is positive

Methods

__call__	

class primordial.equations.events.**Event**(*equations*, *direction=0*, *terminal=False*, *value=0*)
Bases: object

Base class for events.

Gives a more usable wrapper to

Derived classes must define:

__call__(self, t, y) Scalar root function for determining event

equations: Equations The equations for computing derived variables.

direction: [-1, 0, +1] The direction of the root finding (if any)

terminal: bool Whether to stop at this root

class primordial.equations.events.**Inflation**(*equations*, *direction=0*, *terminal=False*, *value=0*)
Bases: *primordial.equations.events.Event*

Inflation entry/exit

Methods

__call__	

class primordial.equations.events.**ModeExit**(*equations*, *direction=0*, *terminal=False*, *value=0*)
Bases: *primordial.equations.events.Event*

When mode exits the horizon

Methods

__call__	

class primordial.equations.events.**UntilN**(*equations*, *direction=0*, *terminal=False*, *value=0*)

Bases: *primordial.equations.events.Event*

Stop at N

Methods

__call__	

primordial.equations.inflation module

Base classes for inflationary solvers

class primordial.equations.inflation.**Equations**(*K*, *potential*)

Bases: *primordial.equations.equations.Equations*

Methods

H(t, y)	Hubble parameter
V(t, y)	Potential
__call__(t, y)	Vector of derivatives
add_variable(*args)	Add dependent variables to the equations
dVdphi(t, y)	Potential derivative
set_independent_variable(name)	Set name of the independent variable
sol(sol, **kwargs)	Post-process solution of solve_ivp

H(*t*, *y*)
Hubble parameter

V(*t*, *y*)
Potential

dVdphi(*t*, *y*)
Potential derivative

sol(*sol*, ***kwargs*)
Post-process solution of solve_ivp

primordial.equations.inflation_potentials module

class primordial.equations.inflation_potentials.**ChaoticPotential**(*m=1*)

Bases: *primordial.equations.inflation_potentials.Potential*

Methods

__call__	
d	
dd	

d (*phi*)

dd (*phi*)

class primordial.equations.inflation_potentials.**Potential**
Bases: object

Module contents

4.1.2 primordial.test package

Submodules

primordial.test.test_cosmology module

primordial.test.test_cosmology.**test_cosmology**()

primordial.test.test_inflation module

primordial.test.test_inflation.**test_inflation**()

primordial.test.test_mukhanov_sasaki module

primordial.test.test_mukhanov_sasaki.**test_mukhanov_sasaki**()

Module contents

4.2 Submodules

4.3 primordial.solver module

primordial.solver.**solve** (*equations*, *ic*, *interp1d_kwargs={}*, **args*, ***kwargs*)

4.4 primordial.units module

4.5 Module contents

CHAPTER 5

primordial: inflationary equation solver

primordial inflationary equation solver

Author Will Handley

Version 0.0.13

Homepage https://github.com/williamjameshandley/primordial

Documentation http://primordial.readthedocs.io/

5.1 Description

`primordial` is a python package for solving cosmological inflationary equations.

It is very much in beta stage, and currently being built for research purposes.

5.2 Example Usage

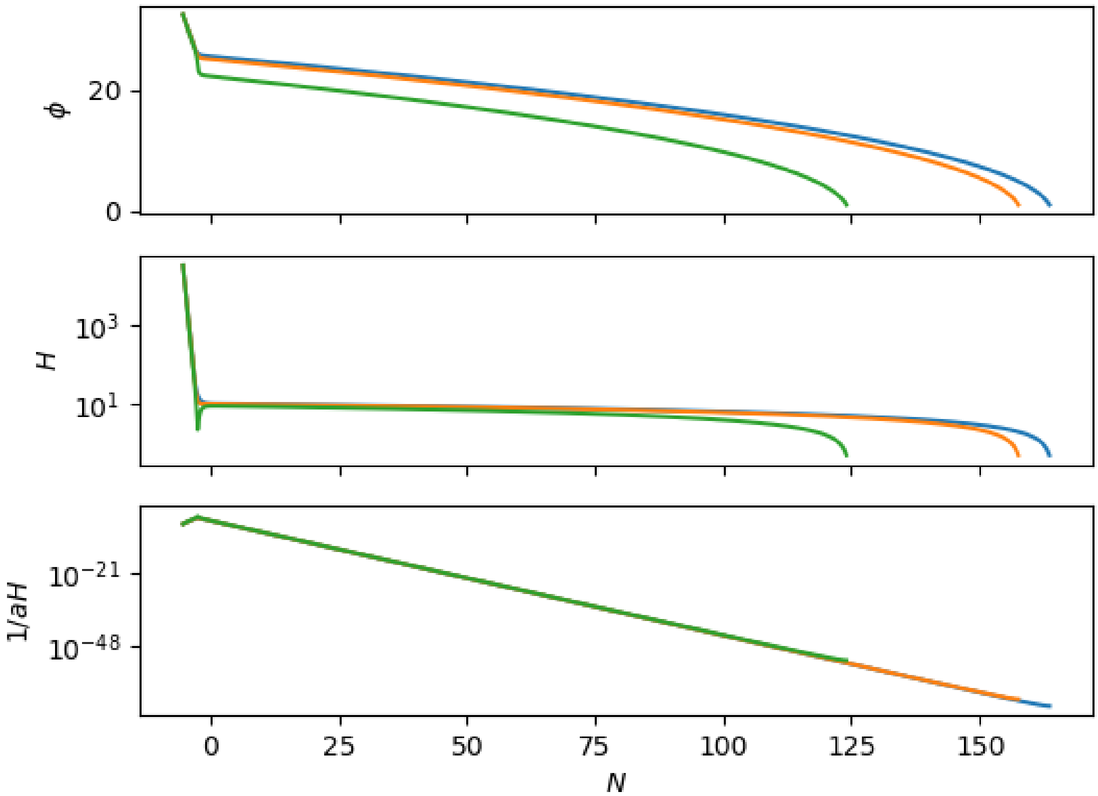
5.2.1 Plot Background evolution

```
import numpy
import matplotlib.pyplot as plt
from primordial.solver import solve
from primordial.equations.inflation_potentials import ChaoticPotential
from primordial.equations.t.inflation import Equations, KD_initial_conditions
from primordial.equations.events import Inflation, Collapse

fig, ax = plt.subplots(3,sharex=True)
for K in [-1, 0, +1]:
```

(continues on next page)

(continued from previous page)

```
    m = 1
    V = ChaoticPotential(m)
    equations = Equations(K, V)

    events= [Inflation(equations),                    # Record inflation entry and␣
↪exit
             Inflation(equations, -1, terminal=True), # Stop on inflation exit
             Collapse(equations, terminal=True)]      # Stop if universe stops␣
↪expanding

    N_p = -1.5
    phi_p = 23
    t_p = 1e-5
    ic = KD_initial_conditions(t_p, N_p, phi_p)
    t = numpy.logspace(-5,10,1e6)

    sol = solve(equations, ic, t_eval=t, events=events)

    ax[0].plot(sol.N(t),sol.phi(t))
    ax[0].set_ylabel(r'$\phi$')

    ax[1].plot(sol.N(t),sol.H(t))
    ax[1].set_yscale('log')
    ax[1].set_ylabel(r'$H$')

    ax[2].plot(sol.N(t),1/(sol.H(t)*numpy.exp(sol.N(t))))
    ax[2].set_yscale('log')
    ax[2].set_ylabel(r'$1/aH$')

ax[-1].set_xlabel('$N$')
```



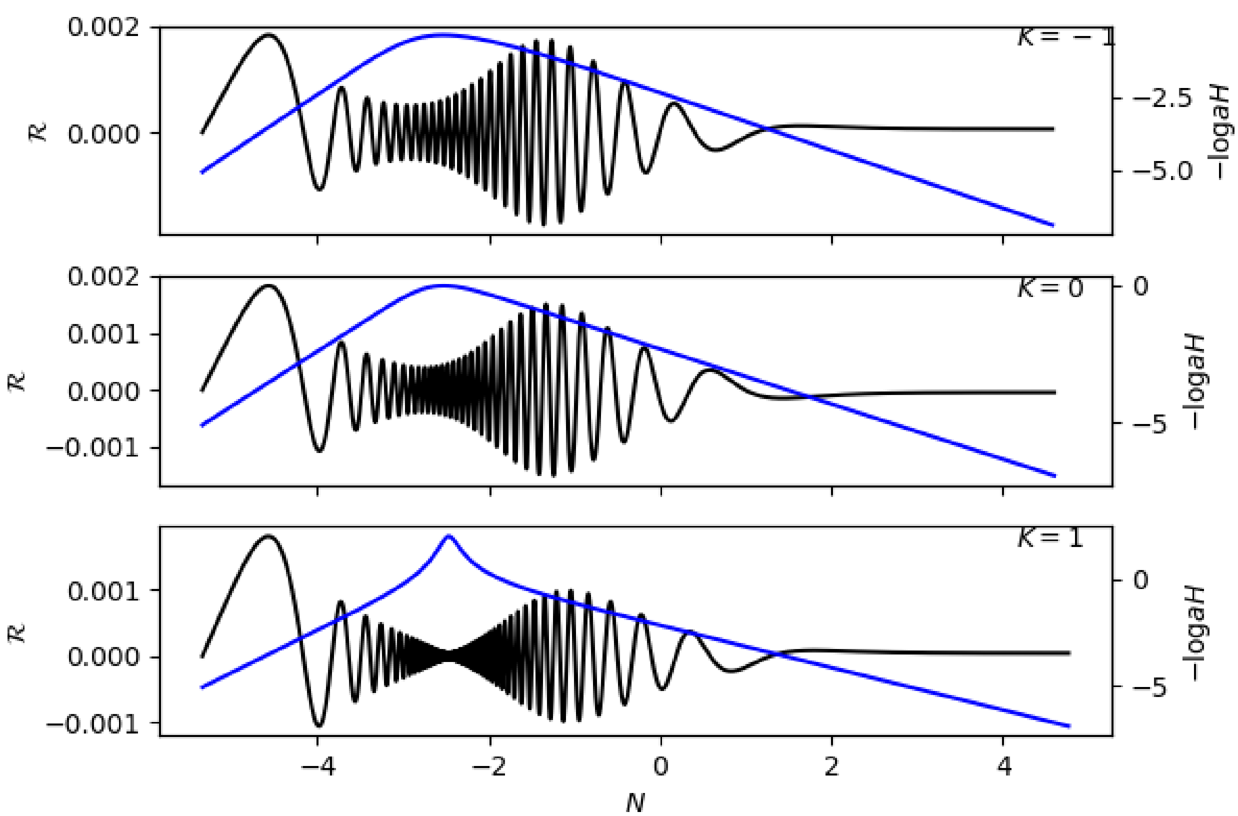
5.2.2 Plot mode function evolution

```
import numpy
import matplotlib.pyplot as plt
from primordial.solver import solve
from primordial.equations.inflation_potentials import ChaoticPotential
from primordial.equations.t.mukhanov_sasaki import Equations, KD_initial_conditions
from primordial.equations.events import Inflation, Collapse, ModeExit

fig, axes = plt.subplots(3,sharex=True)
for ax, K in zip(axes, [-1, 0, +1]):
    ax2 = ax.twinx()
    m = 1
    V = ChaoticPotential(m)
    k = 100
    equations = Equations(K, V, k)

    events= [
            Inflation(equations),                    # Record inflation entry and exit
            Collapse(equations, terminal=True),      # Stop if universe stops␣
↪expanding
            ModeExit(equations, +1, terminal=True, value=1e1*k)    # Stop on mode exit
            ]
```

(continues on next page)

(continued from previous page)

```
    N_p = -1.5
    phi_p = 23
    t_p = 1e-5
    ic = KD_initial_conditions(t_p, N_p, phi_p)
    t = numpy.logspace(-5,10,1e6)

    sol = solve(equations, ic, t_eval=t, events=events)

    N = sol.N(t)
    ax.plot(N,sol.R1(t), 'k-')
    ax2.plot(N,-numpy.log(sol.H(t))-N, 'b-')

    ax.set_ylabel('$\mathcal{R}$')
    ax2.set_ylabel('$-\log aH$')

    ax.text(0.9, 0.9, r'$K=%i$' % K, transform=ax.transAxes)

axes[-1].set_xlabel('$N$')
```



5.3 To do list

Eventually would like to submit this to JOSS. Here are things to do before then:

5.3.1 Cosmology

- Slow roll initial conditions
- Mukhanov Sazaki evolution in N
- add η as independent variable
- add ϕ as independent variable

5.3.2 Code

- Documentation
- **Tests**
  - 100% coverage
  - interpolation
  - cosmology

Python Module Index

p

Index

S

T

U

V