The daisy population will be modeled along the lines of standard population ecology models where the net growth depends upon the current population. For example, the simplest model assumes the rate of growth is proportional to the population, i.e.

$$ \frac{dA_w}{dt} = k_w A_w $$

$$ \frac{dA_b}{dt} = k_b A_b $$

where A_w and A_b are fractions of the total planetary area covered by the white and black daisies, respectively, and k_i, $i=w,b$, are the white and black daisy growth rates per unit time, respectively. If assume the the growth rates k_i are (positive) constants we would have exponential growth like the bunny rabbits of lore.

We can make the model more realistic by letting the daisy birthrate depend on the amount of available land, i.e. $$k_i = \beta_i x$$ where β_i are the white and black daisy growth rates per unit time and area, respectively, and x is the fractional area of free fertile ground not colonized by either species. We can also add a daisy death rate per unit time, χ, to get

$\textbf{eq: constantgrowth}$ $$ \frac{dA_w}{dt} = A_w (\beta_w x - \chi) $$

$$ \frac{dA_b}{dt} = A_b (\beta_b x - \chi) $$

However, even these small modifications are non-trivial mathematically as the available fertile land is given by, $$ x = 1 - A_w - A_b $$

(assuming all the land mass is fertile) which makes the equations non-linear.

Problem constant growth

Note that though the daisy growth rate per unit time depends on the amount of available fertile land, it is not otherwise coupled to the environment (i.e. β_i is not a function of temperature. Making the growth a function of bare ground, however, keeps the daisy population bounded and the daisy population will eventually reach some steady state. The next python cell has a script that runs a fixed timestep Runge Kutte routine that calculates area coverage of white and black daisies for fixed growth rates β_w and β_b. Try changing these growth rates (specified in the derivs5 routine) and the initial white and black concentrations (specified in the fixed_growth.yaml file discussed next). To hand in: plot graphs to illustrate how these changes have affected the fractional coverage of black and white daisies over time compared to the original. Comment on the changes that you see.

1. For a given set of growth rates try various (non-zero) initial daisy populations.
2. For a given set of initial conditions try various growth rates. In particular, try rates that are both greater than and less than the death rate.

3. Can you determine when non-zero steady states are achieved? Explain. Connect what you see here with the discussion of hysteresis towards the end of this lab - what determines which steady state is reached?

Running the constant growth rate demo

In the appendix we discuss the design of the integrator class and the adaptive Runge-Kutta routine. For this demo, we need to be able to change variables in the configuration file. For this assignment problem you are asked to:

a. Change the inital white and black daisy concentrations by changing these lines in the fixed_growth.yaml input file (you can find this file in this lab directory):

```yaml

   initvars:
      whiteconc: 0.2
      blackconc: 0.7
```

b. Change the white and black daisy growth rates by editing the variables beta_w and beta_b in the derivs5 routine in the next cell

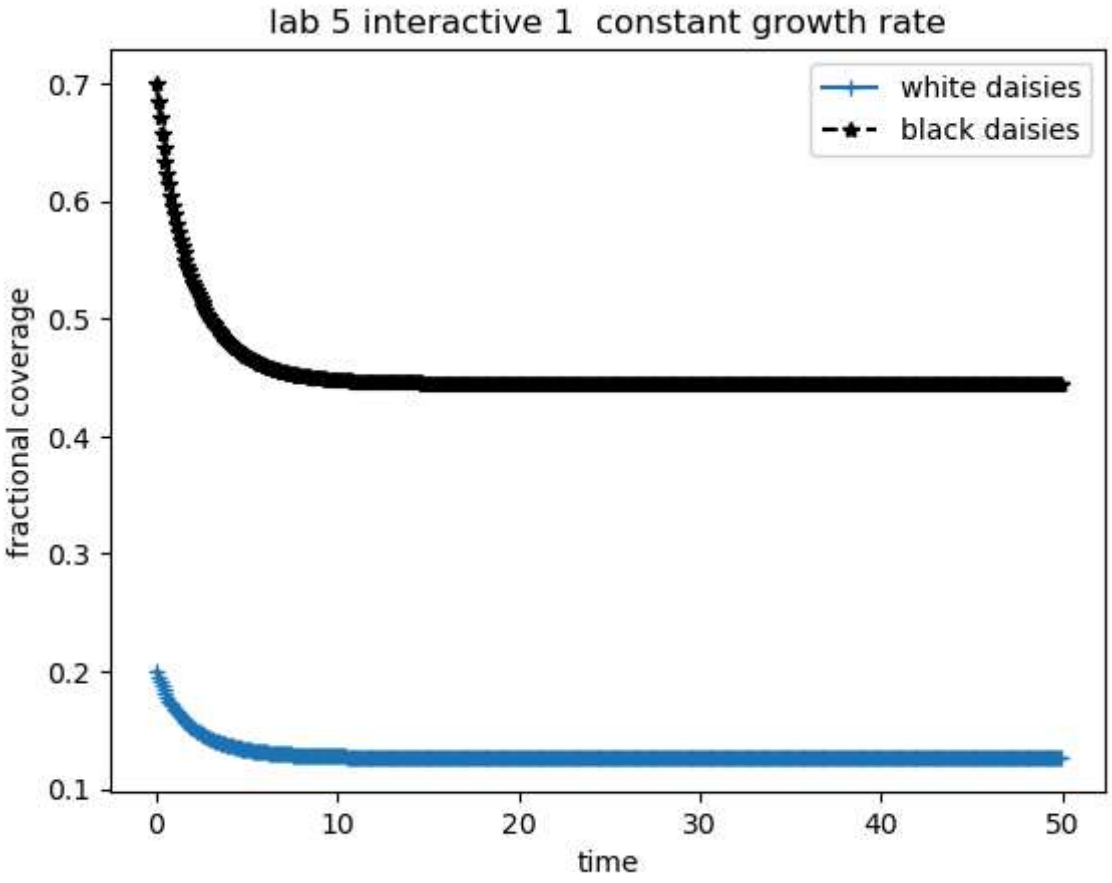
The Integrator class contains two different timeloops, both of which use embedded Runge Kutta Cash Carp code given in Lab 4 and coded here as rkckODE5. The simplest way to loop through the timesteps is just to call the integrator with a specified set of times. This is done in timeloop5fixed. Below we will describe how to use the error extimates returned by rkckODE5 to tune the size of the timesteps, which is done in timeloop5Err.

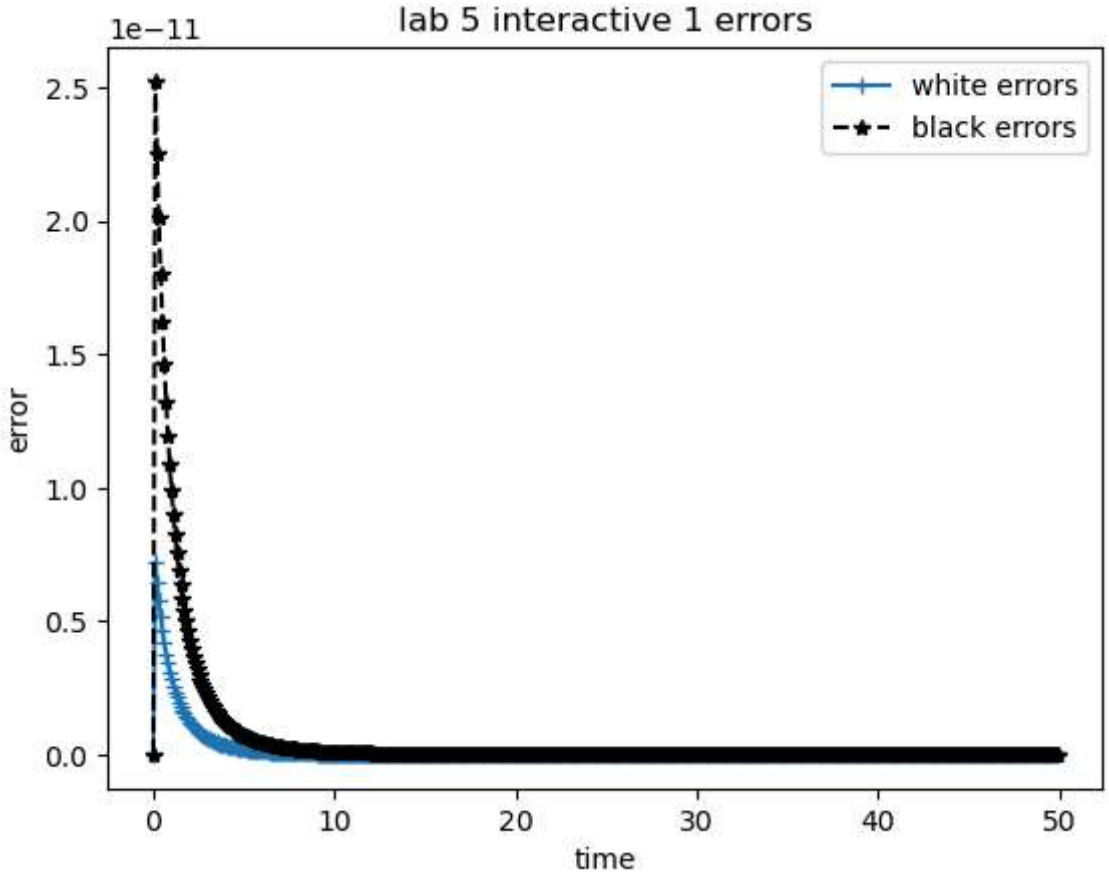
In [1]:
```
#
# 4.1  integrate constant growth rates with fixed timesteps
#
import context
from numlabs.lab5.lab5_funs import Integrator
from collections import namedtuple
import numpy as np
import matplotlib.pyplot as plt


class Integ51(Integrator):
    def set_yinit(self):
        #
        # read in 'albedo_white chi S0 L albedo_black R albedo_ground'
        #
        uservars = namedtuple('uservars', self.config['uservars'].keys())
        self.uservars = uservars(**self.config['uservars'])
        #
        # read in 'whiteconc blackconc'
        #
```

```
        initvars = namedtuple('initvars', self.config['initvars'].keys())
        self.initvars = initvars(**self.config['initvars'])
        self.yinit = np.array(
            [self.initvars.whiteconc, self.initvars.blackconc])
        self.nvars = len(self.yinit)
        return None

    #
    # Construct an Integ51 class by inheriting first intializing
    # the parent Integrator class (called super).  Then do the extra
    # initialization in the set_yint function
    #
    def __init__(self, coeffFileName):
        super().__init__(coeffFileName)
        self.set_yinit()

    def derivs5(self, y, t):
        """y[0]=fraction white daisies
           y[1]=fraction black daisies

           Constant growty rates for white
           and black daisies beta_w and beta_b

           returns dy/dt
        """
        user = self.uservars
        #
        # bare ground
        #
        x = 1.0 - y[0] - y[1]

        # growth rates don't depend on temperature
        beta_b = 0.7  # growth rate for black daisies
        beta_w = 0.7  # growth rate for white daisies

        # create a 1 x 2 element vector to hold the derivitive
        f = np.empty([self.nvars], 'float')
        f[0] = y[0] * (beta_w * x - user.chi)
        f[1] = y[1] * (beta_b * x - user.chi)
        return f


theSolver = Integ51('fixed_growth.yaml')
timeVals, yVals, errorList = theSolver.timeloop5fixed()

plt.close('all')
thefig, theAx = plt.subplots(1, 1)
theLines = theAx.plot(timeVals, yVals)
theLines[0].set_marker('+')
theLines[1].set_linestyle('--')
theLines[1].set_color('k')
theLines[1].set_marker('*')
theAx.set_title('lab 5 interactive 1  constant growth rate')
theAx.set_xlabel('time')
theAx.set_ylabel('fractional coverage')
theAx.legend(theLines, ('white daisies', 'black daisies'), loc='best')
```

```
thefig, theAx = plt.subplots(1, 1)
theLines = theAx.plot(timeVals, errorList)
theLines[0].set_marker('+')
theLines[1].set_linestyle('--')
theLines[1].set_color('k')
theLines[1].set_marker('*')
theAx.set_title('lab 5 interactive 1 errors')
theAx.set_xlabel('time')
theAx.set_ylabel('error')
out = theAx.legend(theLines, ('white errors', 'black errors'), loc='best')
```

```
*****************************
context imported. Front of path:
C:\Users\13432\Documents\ATSC_409\numeric_2024
back of path: C:\Users\13432\miniconda3\envs\numeric_2024\Lib\site-packages\Pythonwi
n
*****************************

through C:\Users\13432\Documents\ATSC_409\numeric_2024\notebooks\lab5\context.py
```



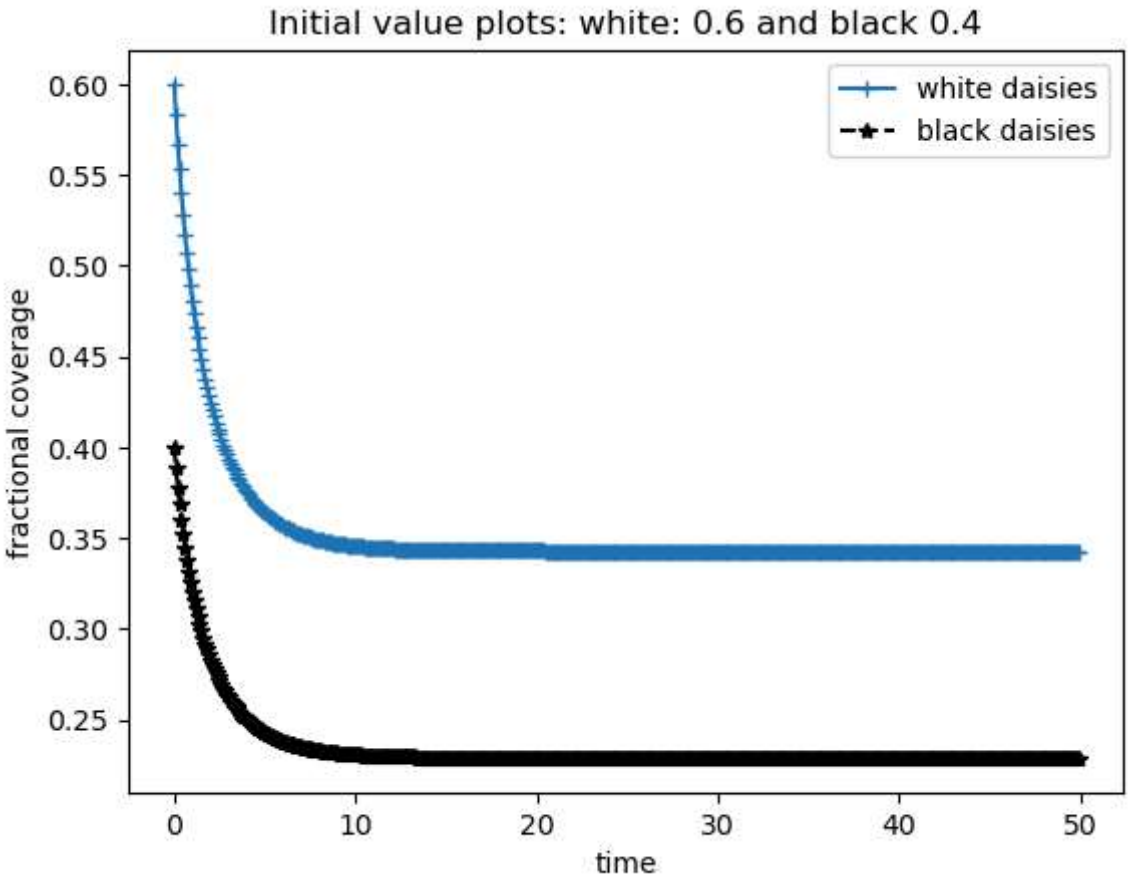


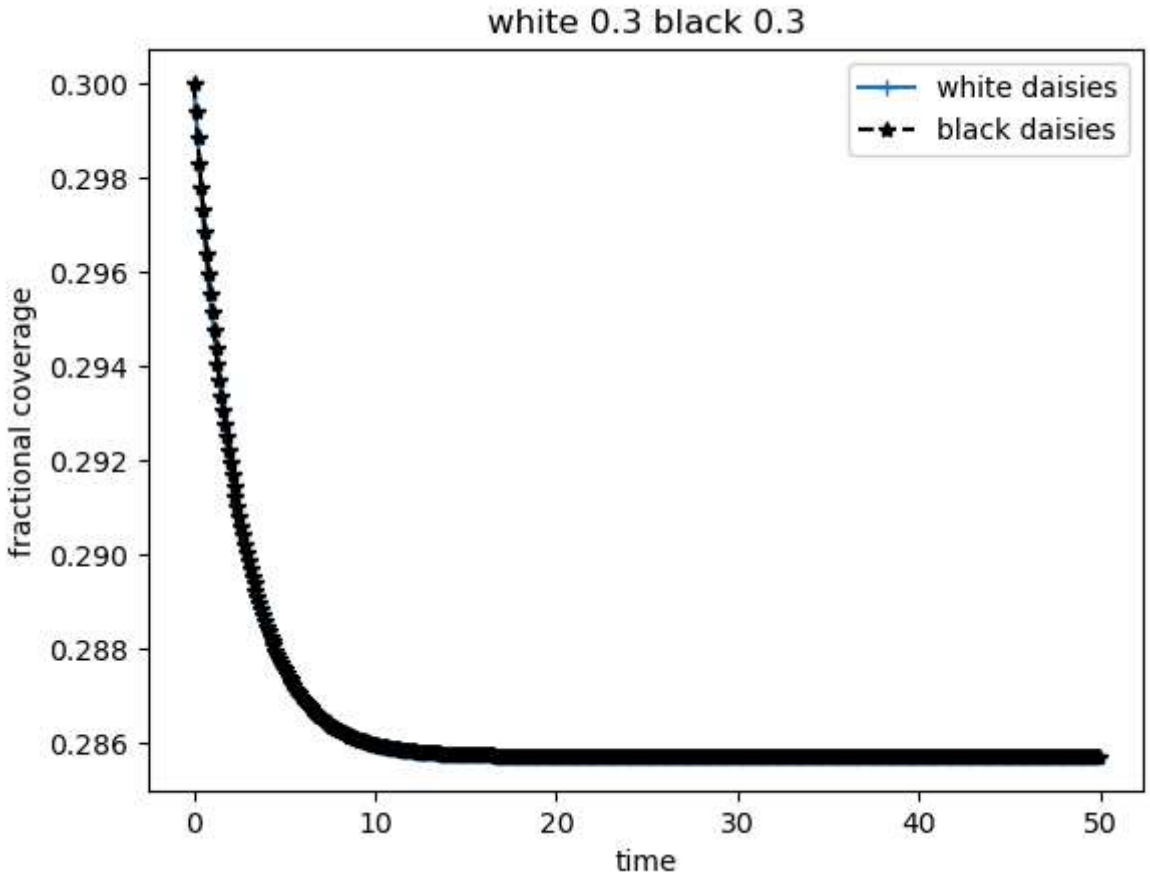
1. In the cell below, we have 3 different cases of initial conditions different from the demo one above.

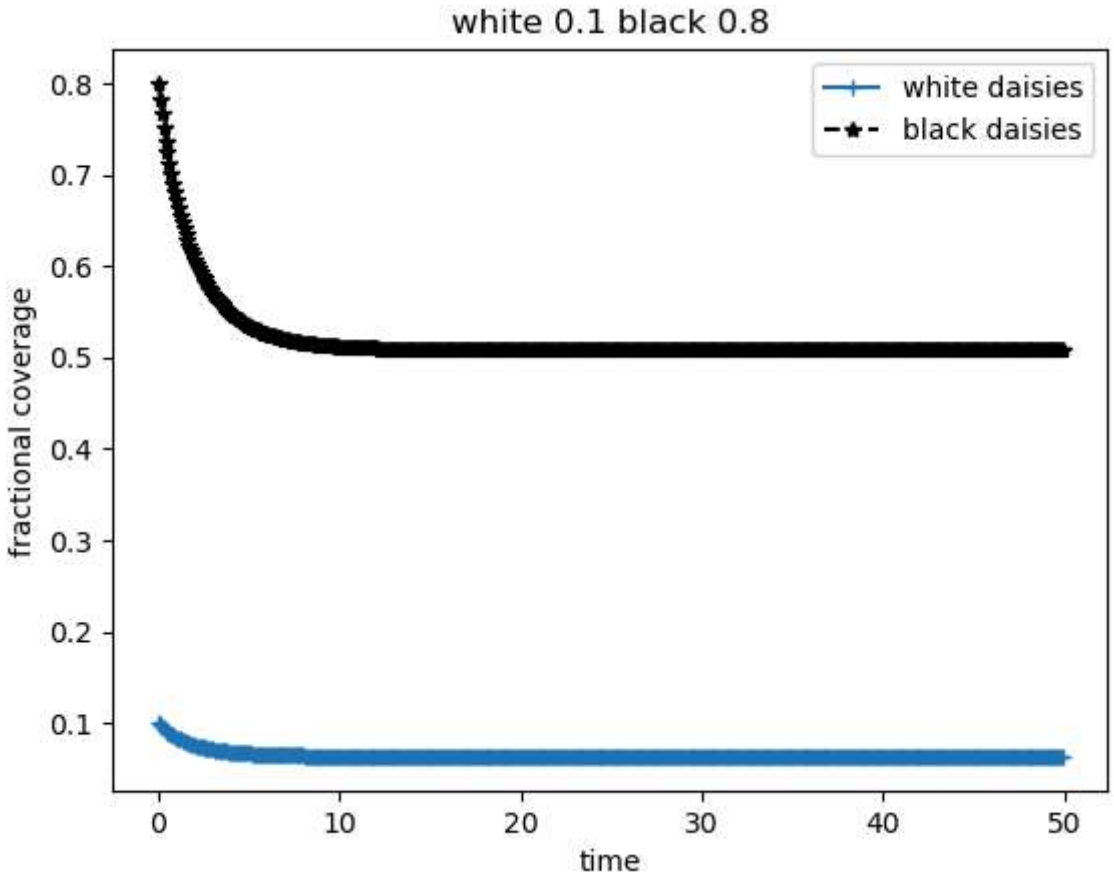
In [2]:
```
theSolver = Integ51('fixed_growth_V2.yaml')
timeVals, yVals, errorList = theSolver.timeloop5fixed()

plt.close('all')
thefig, theAx = plt.subplots(1, 1)
theLines = theAx.plot(timeVals, yVals)
theLines[0].set_marker('+')
theLines[1].set_linestyle('--')
theLines[1].set_color('k')
theLines[1].set_marker('*')
theAx.set_title('Initial value plots: white: 0.6 and black 0.4')
theAx.set_xlabel('time')
theAx.set_ylabel('fractional coverage')
theAx.legend(theLines, ('white daisies', 'black daisies'), loc='best')


theSolver = Integ51('fixed_growth_V3.yaml')
timeVals, yVals, errorList = theSolver.timeloop5fixed()

thefig, theAx = plt.subplots(1, 1)
theLines = theAx.plot(timeVals, yVals)
theLines[0].set_marker('+')
theLines[1].set_linestyle('--')
theLines[1].set_color('k')
```

```
theLines[1].set_marker('*')
theAx.set_title('white 0.3 black 0.3')
theAx.set_xlabel('time')
theAx.set_ylabel('fractional coverage')
theAx.legend(theLines, ('white daisies', 'black daisies'), loc='best')

theSolver = Integ51('fixed_growth_V4.yaml')
timeVals, yVals, errorList = theSolver.timeloop5fixed()

thefig, theAx = plt.subplots(1, 1)
theLines = theAx.plot(timeVals, yVals)
theLines[0].set_marker('+')
theLines[1].set_linestyle('--')
theLines[1].set_color('k')
theLines[1].set_marker('*')
theAx.set_title('white 0.1 black 0.8')
theAx.set_xlabel('time')
theAx.set_ylabel('fractional coverage')
theAx.legend(theLines, ('white daisies', 'black daisies'), loc='best')
```

Out[2]: <matplotlib.legend.Legend at 0x18e2b5a9610>




white 0.3 black 0.3
white daisies
black daisies
fractional coverage
0.300
0.298
0.296
0.294
0.292
0.290
0.288
0.286
0
10
20
30
40
50
time


white 0.1 black 0.8
white daisies
black daisies
fractional coverage
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
time

2. Under the cell below, we have all the plots for the different beta values

In [3]:
```
class Integ51(Integrator):
    def set_yinit(self):
        #
        # read in 'albedo_white chi S0 L albedo_black R albedo_ground'
        #
        uservars = namedtuple('uservars', self.config['uservars'].keys())
        self.uservars = uservars(**self.config['uservars'])
        #
        # read in 'whiteconc blackconc'
        #
        initvars = namedtuple('initvars', self.config['initvars'].keys())
        self.initvars = initvars(**self.config['initvars'])
        self.yinit = np.array(
            [self.initvars.whiteconc, self.initvars.blackconc])
        self.nvars = len(self.yinit)
        return None

    #
    # Construct an Integ51 class by inheriting first intializing
    # the parent Integrator class (called super).  Then do the extra
    # initialization in the set_yint function
    #
    def __init__(self, coeffFileName):
        super().__init__(coeffFileName)
        self.set_yinit()

    def derivs5(self, y, t):
        """y[0]=fraction white daisies
           y[1]=fraction black daisies

           Constant growty rates for white
           and black daisies beta_w and beta_b

           returns dy/dt
        """
        user = self.uservars
        #
        # bare ground
        #
        x = 1.0 - y[0] - y[1]

        # growth rates don't depend on temperature
        beta_b = 0.9  # growth rate for black daisies
        beta_w = 0.5  # growth rate for white daisies

        # create a 1 x 2 element vector to hold the derivitive
        f = np.empty([self.nvars], 'float')
        f[0] = y[0] * (beta_w * x - user.chi)
        f[1] = y[1] * (beta_b * x - user.chi)
        return f
```

```
theSolver = Integ51('fixed_growth.yaml')
timeVals, yVals, errorList = theSolver.timeloop5fixed()

plt.close('all')
thefig, theAx = plt.subplots(1, 1)
theLines = theAx.plot(timeVals, yVals)
theLines[0].set_marker('+')
theLines[1].set_linestyle('--')
theLines[1].set_color('k')
theLines[1].set_marker('*')
theAx.set_title('Plots for different beta terms: beta_b=0.9 beta_w=0.5')
theAx.set_xlabel('time')
theAx.set_ylabel('fractional coverage')
theAx.legend(theLines, ('white daisies', 'black daisies'), loc='best')

class Integ51(Integrator):
    def set_yinit(self):
        #
        # read in 'albedo_white chi S0 L albedo_black R albedo_ground'
        #
        uservars = namedtuple('uservars', self.config['uservars'].keys())
        self.uservars = uservars(**self.config['uservars'])
        #
        # read in 'whiteconc blackconc'
        #
        initvars = namedtuple('initvars', self.config['initvars'].keys())
        self.initvars = initvars(**self.config['initvars'])
        self.yinit = np.array(
            [self.initvars.whiteconc, self.initvars.blackconc])
        self.nvars = len(self.yinit)
        return None

    #
    # Construct an Integ51 class by inheriting first intializing
    # the parent Integrator class (called super).  Then do the extra
    # initialization in the set_yint function
    #
    def __init__(self, coeffFileName):
        super().__init__(coeffFileName)
        self.set_yinit()

    def derivs5(self, y, t):
        """y[0]=fraction white daisies
           y[1]=fraction black daisies

           Constant growty rates for white
           and black daisies beta_w and beta_b

           returns dy/dt
        """
        user = self.uservars
        #
        # bare ground
        #
        x = 1.0 - y[0] - y[1]
```

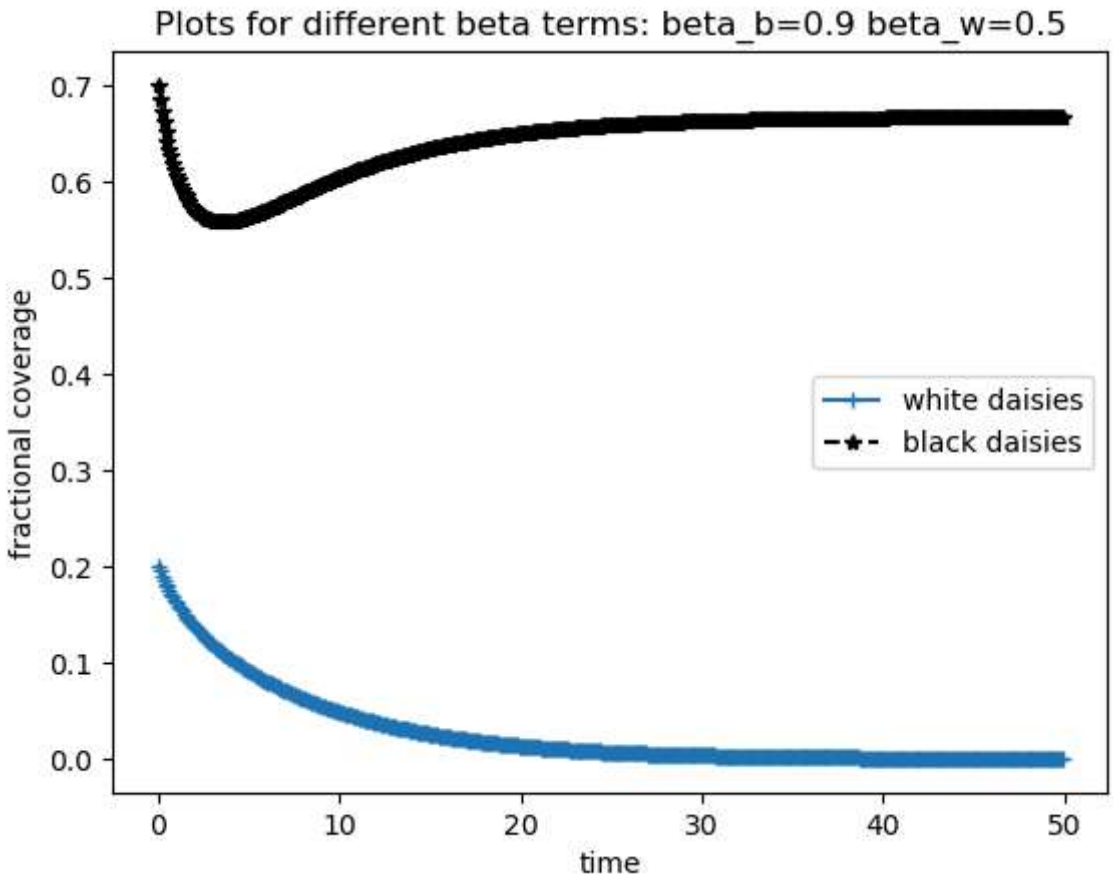
```
        # growth rates don't depend on temperature
        beta_b = 0.3  # growth rate for black daisies
        beta_w = 0.3  # growth rate for white daisies

        # create a 1 x 2 element vector to hold the derivitive
        f = np.empty([self.nvars], 'float')
        f[0] = y[0] * (beta_w * x - user.chi)
        f[1] = y[1] * (beta_b * x - user.chi)
        return f

theSolver = Integ51('fixed_growth.yaml')
timeVals, yVals, errorList = theSolver.timeloop5fixed()

thefig, theAx = plt.subplots(1, 1)
theLines = theAx.plot(timeVals, yVals)
theLines[0].set_marker('+')
theLines[1].set_linestyle('--')
theLines[1].set_color('k')
theLines[1].set_marker('*')
theAx.set_title('Plots for different beta terms: beta_b=0.3 beta_w=0.3')
theAx.set_xlabel('time')
theAx.set_ylabel('fractional coverage')
theAx.legend(theLines, ('white daisies', 'black daisies'), loc='best')

class Integ51(Integrator):
    def set_yinit(self):
        #
        # read in 'albedo_white chi S0 L albedo_black R albedo_ground'
        #
        uservars = namedtuple('uservars', self.config['uservars'].keys())
        self.uservars = uservars(**self.config['uservars'])
        #
        # read in 'whiteconc blackconc'
        #
        initvars = namedtuple('initvars', self.config['initvars'].keys())
        self.initvars = initvars(**self.config['initvars'])
        self.yinit = np.array(
            [self.initvars.whiteconc, self.initvars.blackconc])
        self.nvars = len(self.yinit)
        return None

    #
    # Construct an Integ51 class by inheriting first intializing
    # the parent Integrator class (called super).  Then do the extra
    # initialization in the set_yint function
    #
    def __init__(self, coeffFileName):
        super().__init__(coeffFileName)
        self.set_yinit()

    def derivs5(self, y, t):
        """y[0]=fraction white daisies
           y[1]=fraction black daisies

           Constant growty rates for white
           and black daisies beta_w and beta_b
```

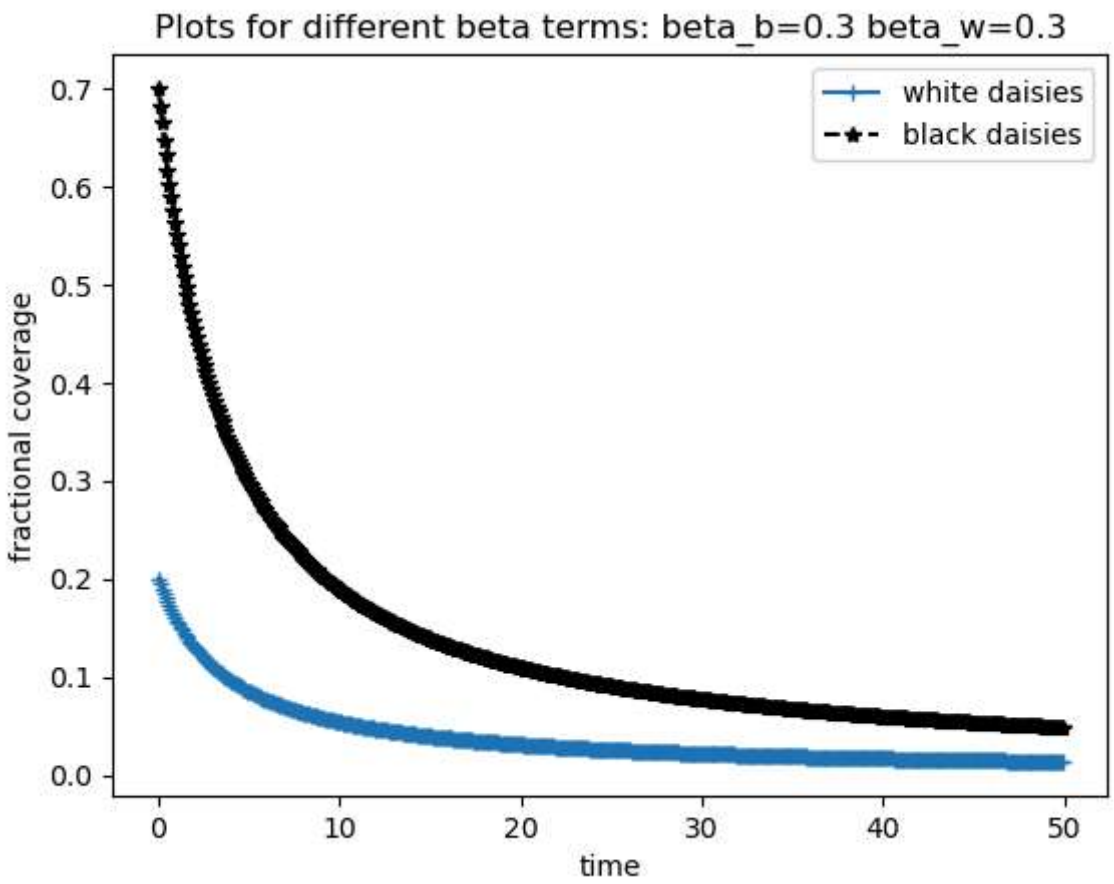
```
            returns dy/dt
        """
        user = self.uservars
        #
        # bare ground
        #
        x = 1.0 - y[0] - y[1]

        # growth rates don't depend on temperature
        beta_b = 0.6  # growth rate for black daisies
        beta_w = 0.65  # growth rate for white daisies

        # create a 1 x 2 element vector to hold the derivitive
        f = np.empty([self.nvars], 'float')
        f[0] = y[0] * (beta_w * x - user.chi)
        f[1] = y[1] * (beta_b * x - user.chi)
        return f

theSolver = Integ51('fixed_growth.yaml')
timeVals, yVals, errorList = theSolver.timeloop5fixed()

thefig, theAx = plt.subplots(1, 1)
theLines = theAx.plot(timeVals, yVals)
theLines[0].set_marker('+')
theLines[1].set_linestyle('--')
theLines[1].set_color('k')
theLines[1].set_marker('*')
theAx.set_title('Plots for different beta terms: beta_b=0.6 beta_w=0.65')
theAx.set_xlabel('time')
theAx.set_ylabel('fractional coverage')
theAx.legend(theLines, ('white daisies', 'black daisies'), loc='best')

class Integ51(Integrator):
    def set_yinit(self):
        #
        # read in 'albedo_white chi S0 L albedo_black R albedo_ground'
        #
        uservars = namedtuple('uservars', self.config['uservars'].keys())
        self.uservars = uservars(**self.config['uservars'])
        #
        # read in 'whiteconc blackconc'
        #
        initvars = namedtuple('initvars', self.config['initvars'].keys())
        self.initvars = initvars(**self.config['initvars'])
        self.yinit = np.array(
            [self.initvars.whiteconc, self.initvars.blackconc])
        self.nvars = len(self.yinit)
        return None

    #
    # Construct an Integ51 class by inheriting first intializing
    # the parent Integrator class (called super).  Then do the extra
    # initialization in the set_yint function
    #
    def __init__(self, coeffFileName):
```

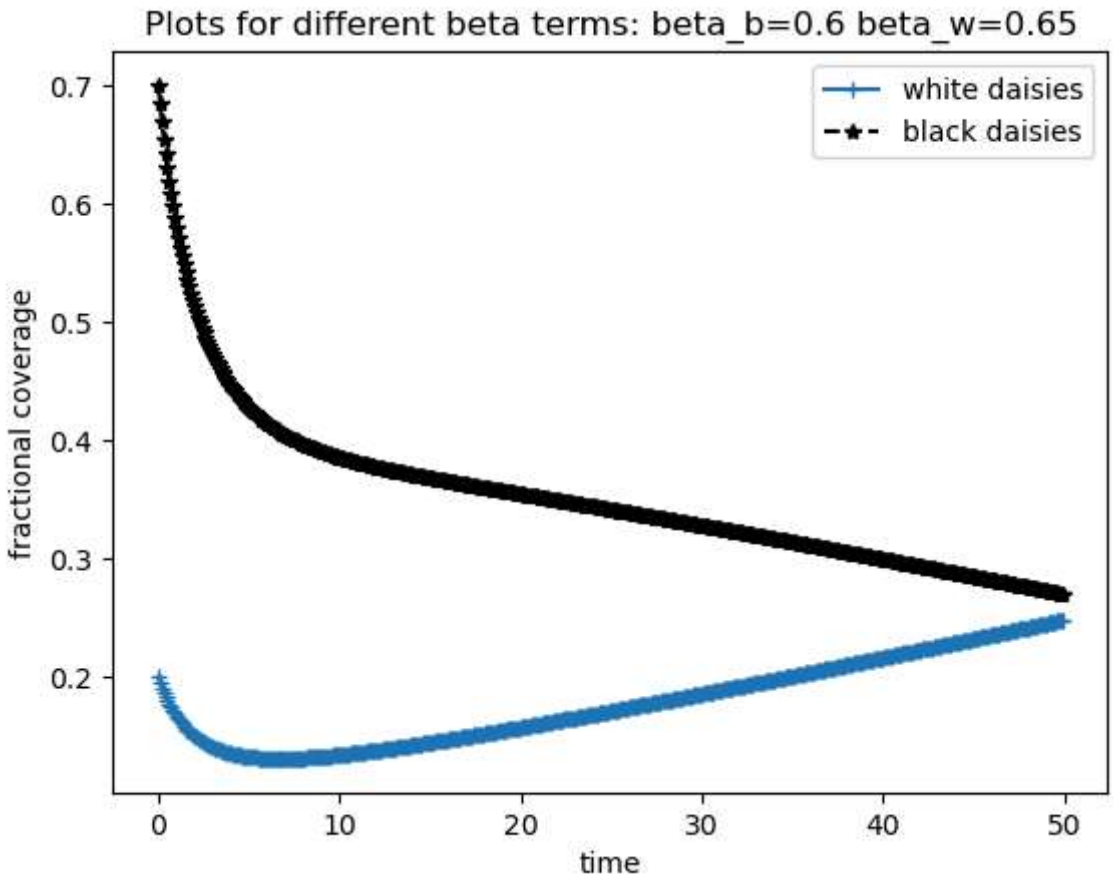
```
        super().__init__(coeffFileName)
        self.set_yinit()

    def derivs5(self, y, t):
        """y[0]=fraction white daisies
           y[1]=fraction black daisies

           Constant growty rates for white
           and black daisies beta_w and beta_b

           returns dy/dt
        """
        user = self.uservars
        #
        # bare ground
        #
        x = 1.0 - y[0] - y[1]

        # growth rates don't depend on temperature
        beta_b = 0.75  # growth rate for black daisies
        beta_w = 0.8  # growth rate for white daisies

        # create a 1 x 2 element vector to hold the derivitive
        f = np.empty([self.nvars], 'float')
        f[0] = y[0] * (beta_w * x - user.chi)
        f[1] = y[1] * (beta_b * x - user.chi)
        return f

theSolver = Integ51('fixed_growth.yaml')
timeVals, yVals, errorList = theSolver.timeloop5fixed()

thefig, theAx = plt.subplots(1, 1)
theLines = theAx.plot(timeVals, yVals)
theLines[0].set_marker('+')
theLines[1].set_linestyle('--')
theLines[1].set_color('k')
theLines[1].set_marker('*')
theAx.set_title('Plots for different beta terms: beta_b=0.75 beta_w=0.8')
theAx.set_xlabel('time')
theAx.set_ylabel('fractional coverage')
theAx.legend(theLines, ('white daisies', 'black daisies'), loc='best')

class Integ51(Integrator):
    def set_yinit(self):
        #
        # read in 'albedo_white chi S0 L albedo_black R albedo_ground'
        #
        uservars = namedtuple('uservars', self.config['uservars'].keys())
        self.uservars = uservars(**self.config['uservars'])
        #
        # read in 'whiteconc blackconc'
        #
        initvars = namedtuple('initvars', self.config['initvars'].keys())
        self.initvars = initvars(**self.config['initvars'])
        self.yinit = np.array(
            [self.initvars.whiteconc, self.initvars.blackconc])
```

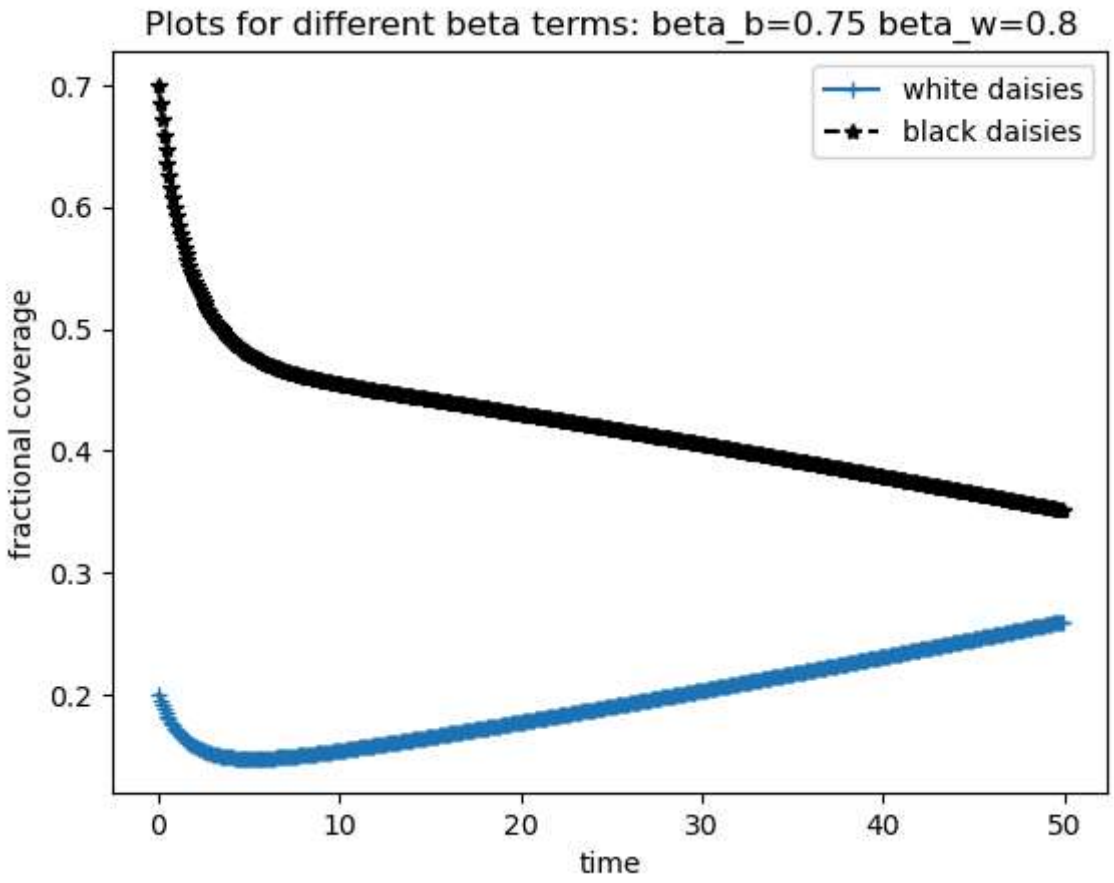
```
        self.nvars = len(self.yinit)
        return None

    #
    # Construct an Integ51 class by inheriting first intializing
    # the parent Integrator class (called super).  Then do the extra
    # initialization in the set_yint function
    #
    def __init__(self, coeffFileName):
        super().__init__(coeffFileName)
        self.set_yinit()

    def derivs5(self, y, t):
        """y[0]=fraction white daisies
           y[1]=fraction black daisies

           Constant growty rates for white
           and black daisies beta_w and beta_b

           returns dy/dt
        """
        user = self.uservars
        #
        # bare ground
        #
        x = 1.0 - y[0] - y[1]

        # growth rates don't depend on temperature
        beta_b = 0.4  # growth rate for black daisies
        beta_w = 0.4  # growth rate for white daisies

        # create a 1 x 2 element vector to hold the derivitive
        f = np.empty([self.nvars], 'float')
        f[0] = y[0] * (beta_w * x - user.chi)
        f[1] = y[1] * (beta_b * x - user.chi)
        return f

theSolver = Integ51('fixed_growth.yaml')
timeVals, yVals, errorList = theSolver.timeloop5fixed()

thefig, theAx = plt.subplots(1, 1)
theLines = theAx.plot(timeVals, yVals)
theLines[0].set_marker('+')
theLines[1].set_linestyle('--')
theLines[1].set_color('k')
theLines[1].set_marker('*')
theAx.set_title('Plots for different beta terms: beta_b=0.4 beta_w=0.4')
theAx.set_xlabel('time')
theAx.set_ylabel('fractional coverage')
theAx.legend(theLines, ('white daisies', 'black daisies'), loc='best')
```

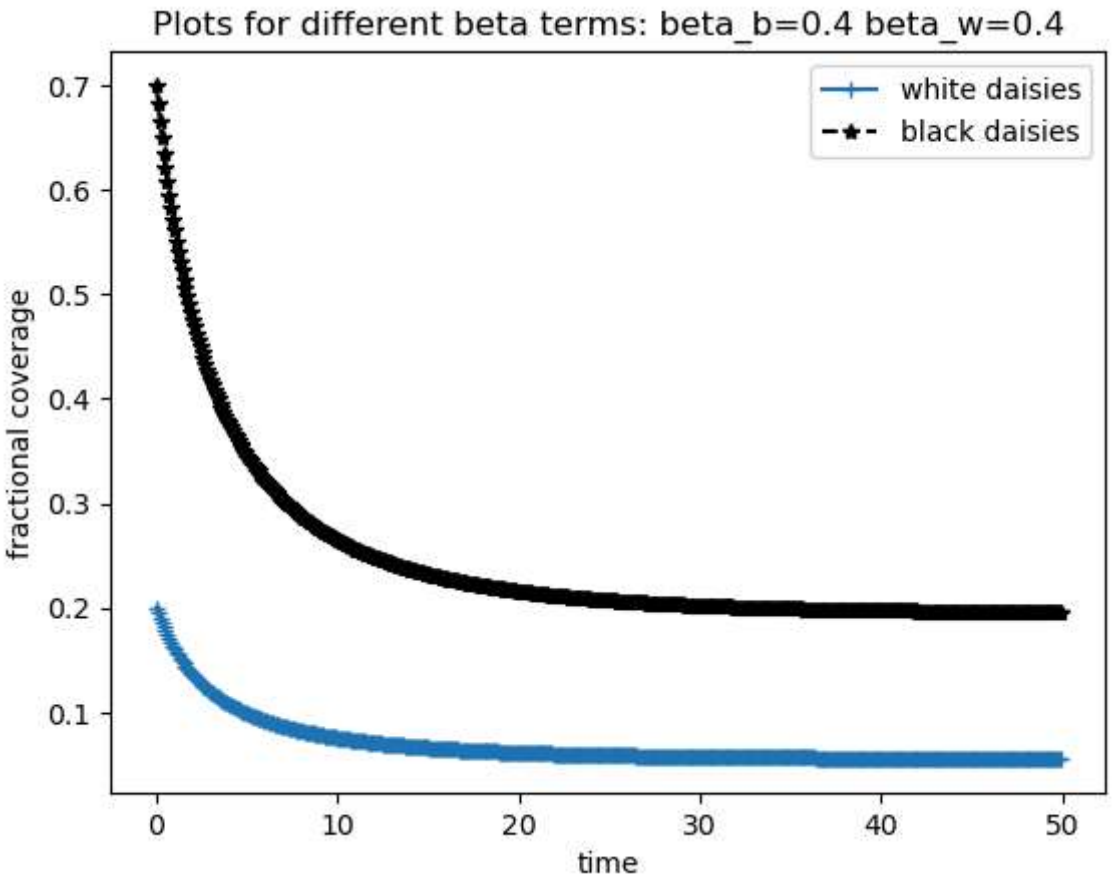
Out[3]: <matplotlib.legend.Legend at 0x18e2d243410>


Plots for different beta terms: beta_b=0.9 beta_w=0.5
fractional coverage
time
white daisies
black daisies
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50


Plots for different beta terms: beta_b=0.3 beta_w=0.3
fractional coverage
time
white daisies
black daisies
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50


Plots for different beta terms: beta_b=0.6 beta_w=0.65
white daisies
black daisies
fractional coverage
time
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50


Plots for different beta terms: beta_b=0.75 beta_w=0.8
white daisies
black daisies
fractional coverage
time
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50



3. From the plots above, it seems that non-zero solutions are obtained for the type of daisy that has a higher beta value, as the other type will go extinct. In addition, if the beta values are equal for both, then the daisy with the highest initial fraction will attain a steady state solution, or if the fractions are equal, then they will both reach a non-zero steady state. In addition to this, both the growth rates must be large enough to ensure that the system reaches a steady-state solution before a population goes extinct; This also depends on the initial conditions as a larger initial condition gives the population more wiggle room to reach a steady state. Mathematically, to get non-zero steady solutions, we need chi=beta*x. So in the case of both having the same beta values and having an initial fraction large enough, then we can achieve a non-zero steady state solution. Otherwise, in the equation, we have A*(beta*x-chi) and one of the daisy fractions A will have to go to zero in order for both equations to achieve a steady state solution.

Connecting this with the discussion at the end of the lab, we see that the black daisies are expected to go to zero with the initial conditions since L is larger than 1.3, so the only steady state solution is that of zero. As for the white daisies, they seem to be in the range where we see hysterisis, and therefore, the history of the system is important since it dictates the steady state solution depending on how the system behaves and if we increase both initial fractions or not, similar to the solar constant.