Lab 5: Daisyworld

Angelene Leow, 23162167

In [159]:
```
import context
from numlabs.lab5.lab5_funs import Integrator
from collections import namedtuple
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Problem 1 : 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. βi is note 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 βw and βb . 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).

1. For a given set of growth rates try various (non-zero) initial daisy populations.

```
initvars:
   whiteconc: 0.9
   blackconc: 0.1
```

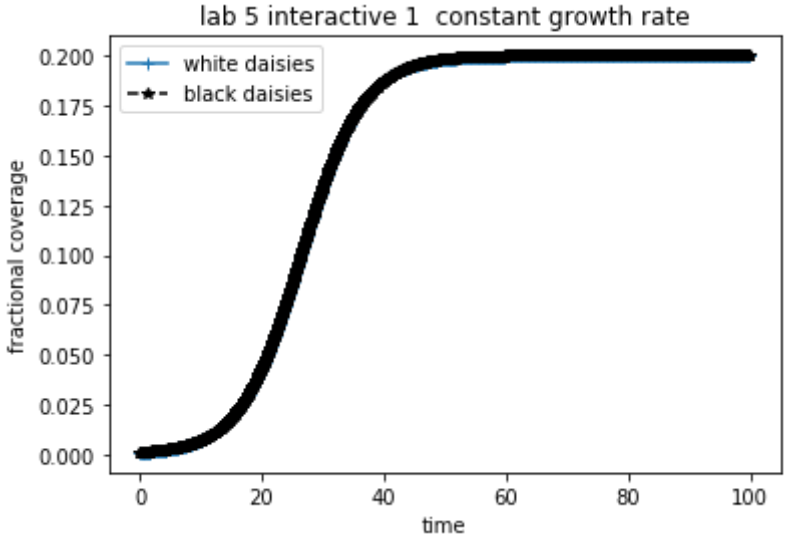
1. For a given set of initial conditions try various growth rates. In particular, try rates that are both greater than and less than the death rate.
   
2. Can you determine when non-zero steady states are achieved? Explain.

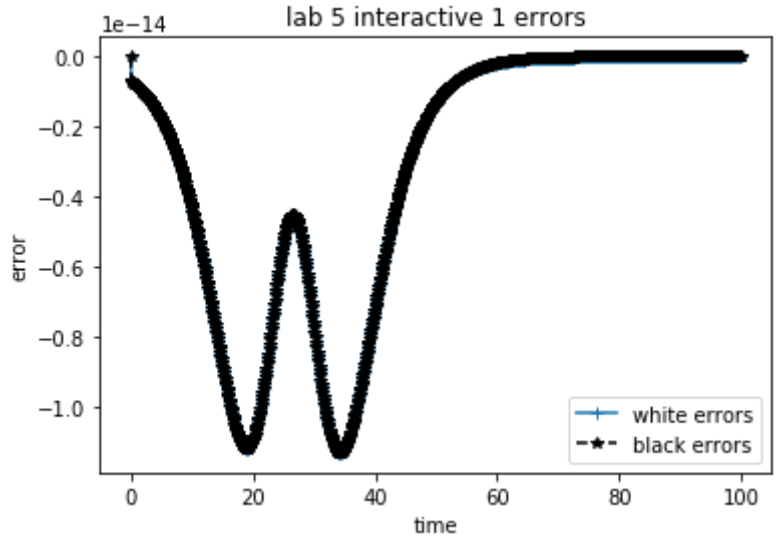
Yes. This happens when the function tends to a non zero value but stays constant with time.

$\frac{\partial A_w}{\partial t} = 0$ for white daisies and $\frac{\partial A_b}{\partial t} = 0$ for black daisies, where A_w and $A_b \neq 0$.

In [106]:
```
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 = 0.5  # 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('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='bes
t')
```





0.001 is the initial value for the stocks of both black and white daisies. Note that if we were to start with 0 in these reservoirs, the daisies would never be able to start growing since the growth flow is dependent on daisy population.

Problem 2 : Coupling

Consider daisies with the same albedo as the planet, i.e. 'grey' or neutral daisies, as specified in derivs5 routine below.

1. For the current value of L (0.2) in the file coupling.yaml, the final daisy steady state is zero. Why is it zero?

There is insufficient solar energy reaching the surface of Dasiyword to make it warm enough to allow growth for daisies. Daisy growth requires a range of temperature for optimum growth. The cooling of the atmosphere causes all currently living daisies to die off.

2. Find a value of L which leads to a non-zero steady state. $L = 0.6$

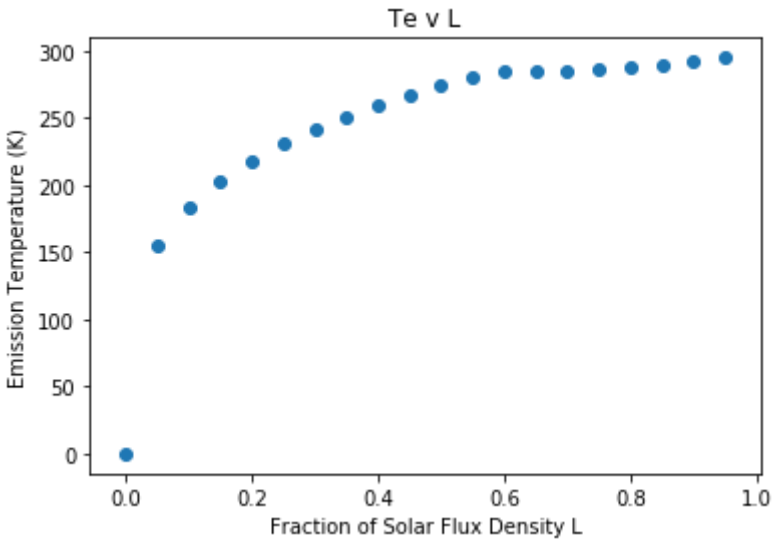
3. What happens to the emission temperature as L is varied? Make a plot of L vs. T_E for 10-15 values of L. Based on the plot in the next cell, we see there is a plateau starting around $L = 0.6$. This is where Daisyworld reaches and non-zero steady-state. When L aproaches 1 the temperature does not increase. Temperature does not increase beyond 300K as it is not feasible for $L > 1$.

In [148]:
```
class IntegCoupling(Integrator):
    """rewrite the init and derivs5 methods to
       work with a single (grey) daisy
    """
    def set_yinit(self, newL):
                #
        # change the luminocity
        #
        self.config["uservars"]["L"] = newL # change solar incidence fra
ction
        #
        # make a new namedtuple factory called uservars that includes ne
wL
        #
        uservars_fac = namedtuple('uservars', self.config['uservars'].ke
ys())
        #
        # use the factory to make the augmented uservars named tuple
        #
        self.uservars = uservars_fac(**self.config['uservars'])
        #



        #
        # read in 'albedo_grey chi S0 L  R albedo_ground'
        #
        uservars = namedtuple('uservars', self.config['uservars'].keys
())
        self.uservars = uservars(**self.config['uservars'])
        #
        # read in 'greyconc'
        #
        initvars = namedtuple('initvars', self.config['initvars'].keys
())
        self.initvars = initvars(**self.config['initvars'])
        self.yinit = np.array([self.initvars.greyconc])
        self.nvars = len(self.yinit)
        return None



    def __init__(self, coeffFileName, newL):
       super().__init__(coeffFileName)
       self.set_yinit(newL)

    def derivs5(self, y, t):
        """
           Make the growth rate depend on the ground temperature
           using the quadratic function of temperature

           y[0]=fraction grey daisies
           t = time
           returns f[0] = dy/dt
        """
        sigma = 5.67e-8  # Stefan Boltzman constant W/m^2/K^4
        user = self.uservars
```

```
        x = 1.0 - y[0]
        albedo_p = x * user.albedo_ground + y[0] * user.albedo_grey
        Te_4 = user.S0 / 4.0 * user.L * (1.0 - albedo_p) / sigma

        temp_e = Te_4**0.25
        self.temp_e = temp_e ## SAVES TO SELF!!!!

        eta = user.R * user.L * user.S0 / (4.0 * sigma)
        temp_y = (eta * (albedo_p - user.albedo_grey) + Te_4)**0.25

        self.temp_y = temp_y ## SAVES TO SELF!!!!

        if (temp_y >= 277.5 and temp_y <= 312.5):
            beta_y = 1.0 - 0.003265 * (295.0 - temp_y)**2.0
        else:
            beta_y = 0.0

        # create a 1 x 1 element vector to hold the derivative
        f = np.empty([self.nvars], np.float64)
        f[0] = y[0] * (beta_y * x - user.chi)

        return f
```

In [156]:
```
## make array of fractional L values
newL = np.arange(0,1,0.05)

## Loop and append temp and emssions temp based off L valuse
temp_y_list, temp_e_list  = [], []
for L in newL:
    theSolver = IntegCoupling("coupling.yaml", L)
    timeVals, yVals, errorList = theSolver.timeloop5fixed()
    temp_y_list.append(theSolver.temp_y)
    temp_e_list.append(theSolver.temp_e)

## Plot L v Te
fig, ax = plt.subplots()
plt.title('Te v L')
ax.scatter(newL, temp_e_list)
ax.set_ylabel('Emission Temperature (K)')
ax.set_xlabel('Fraction of Solar Flux Density L')
```

Out[156]:
```
Text(0.5, 0, 'Fraction of Solar Flux Density L')
```

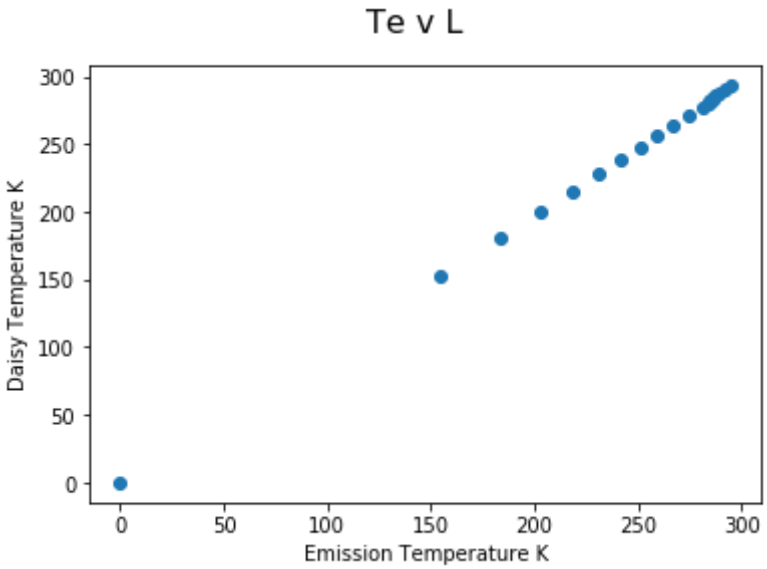


Problem 2 cont.

> **4. Do you see any difference between the daisy temperature and emission temperature? Plot both and explain.**

The average daily temperature is lower than emission temperature. This is due to a smaller planetary albedo as compared to the grey daisy albedo.

In [155]:
```
# p2q4
fig, ax = plt.subplots()
fig.suptitle('Te v L', fontsize=16)
ax.scatter(temp_e_list, temp_y_list)
ax.set_xlabel('Emission Temperature (K)')
ax.set_ylabel('Daisy Temperature (K)')
dif = temp_e_list[1]-temp_y_list[1]
```



5. How (i.e. thorugh what mechanism) does the makeup of the global daisy population affect the local temperature?

The local temperature is affected by the albedo of the grey daisies and the ground albedo. A non-steady state is observed as the value of the fractional solar flux density exceeds the albedo value of the grey daisy. Hence the grey daisies have to absorb more energy to compensate for the energy loss due to reflection and refraction into the atmosphere.

Problem 4 : Initial

> **1) Add a small initial fraction of black daisies (say, 0.01) to the value in initial.yaml and see what effect this has on the temperature and final daisy populations. Do you still have a final non-zero daisy population?**

There is no difference on the final daisy population. The temperature profile increased slightly for both black and white daisies.

> **2) Attempt to adjust the initial white daisy population to obtain a non-zero steady state. Do you have to increase or decrease the initial fraction? What is your explanation for this behavior?**

To obtain a non-zero steady state, the fractional coverage has to be above 0.2. The initial fraction has to be increased. For the white daisy to achieved non-zero steady state growth depends on temperature, and temperature is dependent on albedo, emission temperature, and solar luminosity of Daisyworld,

> **3) Experiment with other initial fractions of daisies and look for non-zero steady states.**

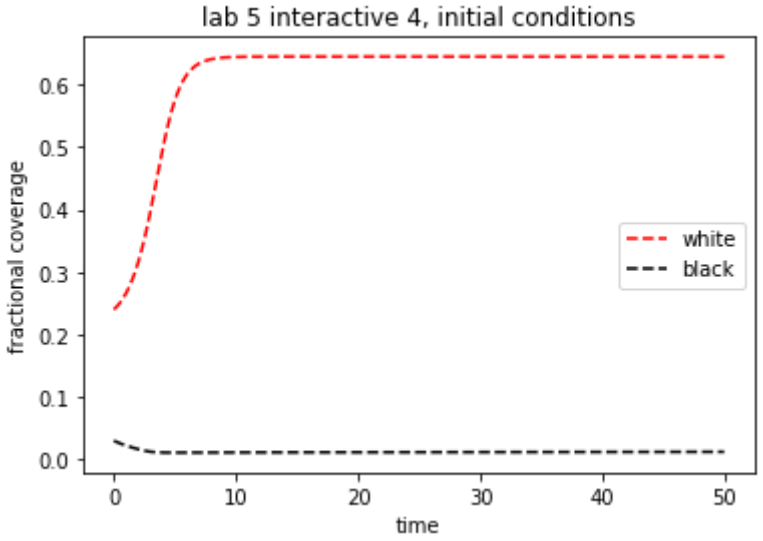
To achieve a steady state growth rate, the black daisy population should be relatively small in comparison to the white daisy population. As such, the temperature (albedo influenced) must be at an optimum temperature for daisies to survive.

In [157]:
```
from numlabs.lab5.lab5_funs import Integrator


class Integ54(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

    def __init__(self, coeff_file_name):
        super().__init__(coeff_file_name)
        self.set_yinit()
        ##########################################################
        ######################## Temp Q.1 ########################
        self.temp_w_list, self.temp_b_list, self.temp_e_list = [], [],
[]
        self.beta_b_list, self.beta_w_list = [], []
        ##########################################################


    def find_temp(self, yvals):
        """
            Calculate the temperatures over the white and black daisies
            and the planetary equilibrium temperature given the daisy fr
actions

            input:  yvals -- array of dimension [2] with the white [0] a
nd black [1]
                    daisy fractiion
            output:  white temperature (K), black temperature (K), equil
ibrium temperature (K)
        """
        sigma = 5.67e-8  # Stefan Boltzman constant W/m^2/K^4
        user = self.uservars
        bare = 1.0 - yvals[0] - yvals[1]
        albedo_p = bare * user.albedo_ground + \
            yvals[0] * user.albedo_white + yvals[1] * user.albedo_black
        Te_4 = user.S0 / 4.0 * user.L * (1.0 - albedo_p) / sigma
        temp_e = Te_4**0.25
        eta = user.R * user.L * user.S0 / (4.0 * sigma)
        temp_b = (eta * (albedo_p - user.albedo_black) + Te_4)**0.25
        temp_w = (eta * (albedo_p - user.albedo_white) + Te_4)**0.25
```

```
        return (temp_w, temp_b, temp_e)

    def derivs5(self, y, t):
        """y[0]=fraction white daisies
           y[1]=fraction black daisies
           no feedback between daisies and
           albedo_p (set to ground albedo)
        """
        temp_w, temp_b, temp_e = self.find_temp(y)

        if (temp_b >= 277.5 and temp_b <= 312.5):
            beta_b = 1.0 - 0.003265 * (295.0 - temp_b)**2.0
        else:
            beta_b = 0.0

        if (temp_w >= 277.5 and temp_w <= 312.5):
            beta_w = 1.0 - 0.003265 * (295.0 - temp_w)**2.0
        else:
            beta_w = 0.0
        ########################################################
        ###################### Temp Q.1 ########################
        self.beta_b = beta_b
        self.beta_w = beta_w
        ########################################################

        user = self.uservars
        bare = 1.0 - y[0] - y[1]
        # create a 1 x 2 element vector to hold the derivitive
        f = np.empty_like(y)
        f[0] = y[0] * (beta_w * bare - user.chi)
        f[1] = y[1] * (beta_b * bare - user.chi)

        # self.temp_e = temp_e
        # self.temp_b = temp_b
        # self.temp_w = temp_w
        return f

    def timeloop5fixed(self):
        """fixed time step with
        estimated errors
        """
        t = self.timevars
        yold = self.yinit
        yError = np.zeros_like(yold)
        yvals = [yold]
        # self.temp_w, self.temp_b, self.temp_e = [], [], []
        errorList = [yError]
        timeSteps = np.arange(t.tstart, t.tend, t.dt)
        for theTime in timeSteps[:-1]:
            yold, yError, newTime = self.rkckODE5(yold, theTime, t.dt)
            temp_wi, temp_bi, temp_ei = self.find_temp(yold)
            ########################################################
            ###################### Temp Q.1 ########################

            self.temp_w_list.append(temp_wi)
            self.temp_b_list.append(temp_bi)
```

```
            self.temp_e_list.append(temp_ei)

            self.beta_b_list.append(self.beta_b)
            self.beta_w_list.append(self.beta_w)
            ##########################################################
            yvals.append(yold)
            errorList.append(yError)

        yvals = np.array(yvals).squeeze()
        errorVals = np.array(errorList).squeeze()
        return (timeSteps, yvals, errorVals)
```

In [160]:
```
theSolver = Integ54('initial.yaml')
timevals, yvals, errorlist = theSolver.timeloop5fixed()
daisies = pd.DataFrame(yvals, columns=['white', 'black'])

thefig, theAx = plt.subplots(1, 1)
line1, = theAx.plot(timevals, daisies['white'])
line2, = theAx.plot(timevals, daisies['black'])
line1.set(linestyle='--', color='r', label='white')
line2.set(linestyle='--', color='k', label='black')
theAx.set_title('lab 5 interactive 4, initial conditions')
theAx.set_xlabel('time')
theAx.set_ylabel('fractional coverage')
out = theAx.legend(loc='center right')
```



Problem 5: Temperature

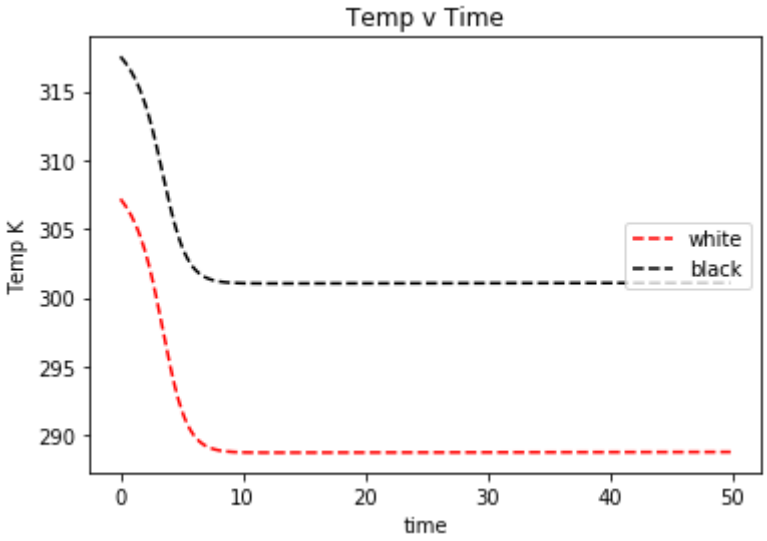
The code above adds a new method, 'find_temp' that takes the white/black daisy fractions and calculates local and planetary temperatures.

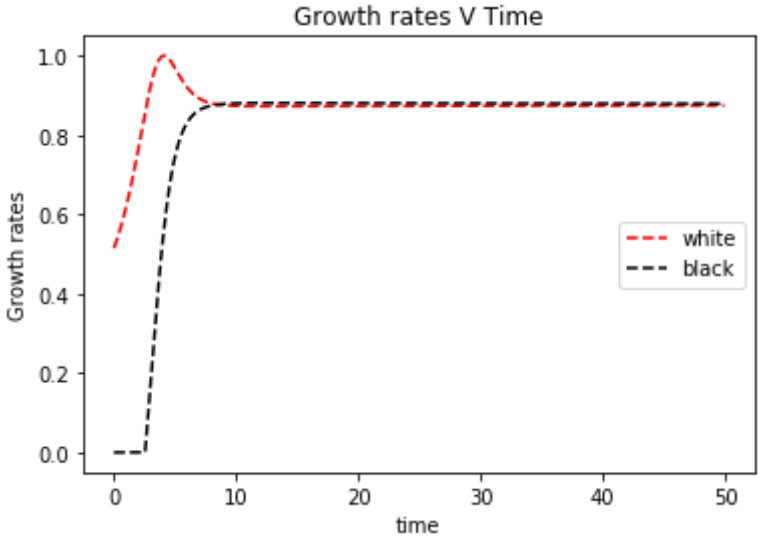
1. override 'timeloop5fixed' so that it saves these three temperatures, plus the daisy growth rates to new variables in the 'Integ54' instance.

In [162]:
```
def timeloop5fixed(self):
        """fixed time step with
        estimated errors
        """
        t = self.timevars
        yold = self.yinit
        yError = np.zeros_like(yold)
        yvals = [yold]
        # self.temp_w, self.temp_b, self.temp_e = [], [], []
        errorList = [yError]
        timeSteps = np.arange(t.tstart, t.tend, t.dt)
        for theTime in timeSteps[:-1]:
            yold, yError, newTime = self.rkckODE5(yold, theTime, t.dt)
            temp_wi, temp_bi, temp_ei = self.find_temp(yold)
            ########################################################

            self.temp_w_list.append(temp_wi)
            self.temp_b_list.append(temp_bi)
            self.temp_e_list.append(temp_ei)

            self.beta_b_list.append(self.beta_b)
            self.beta_w_list.append(self.beta_w)
            ########################################################
            yvals.append(yold)
            errorList.append(yError)

        yvals = np.array(yvals).squeeze()
        errorVals = np.array(errorList).squeeze()
        return (timeSteps, yvals, errorVals)
```

2. Make plots of (temp_w, temp_b) and (beta_w, beta_b) vs. time for a case with non-zero equilibrium concentrations of both black and white daisies.

In [161]:
```
thefig, theAx = plt.subplots(1, 1)
line1, = theAx.plot(timevals[:-1], theSolver.temp_w_list )
line2, = theAx.plot(timevals[:-1], theSolver.temp_b_list )
line1.set(linestyle='--', color='r', label='white')
line2.set(linestyle='--', color='k', label='black')
theAx.set_title('Temp v Time')
theAx.set_xlabel('time')
theAx.set_ylabel('Temp K')
out = theAx.legend(loc='center right')

thefig, theAx = plt.subplots(1, 1)
line1, = theAx.plot(timevals[:-1], theSolver.beta_w_list )
line2, = theAx.plot(timevals[:-1], theSolver.beta_b_list )
line1.set(linestyle='--', color='r', label='white')
line2.set(linestyle='--', color='k', label='black')
theAx.set_title('Growth rates V Time')
theAx.set_xlabel('time')
theAx.set_ylabel('Growth rates')
out = theAx.legend(loc='center right')
```





Problem 6: Estimate

In the demo below, compare the error estimate to the true error, on the initial value problem from ,

$$\frac{dy}{dt} = -y + t + 1, y(0) = 1$$

which has the exact solution

$$y(t) = t + e - t$$

1. Play with the time step and final time, attempting small changes at first. How reasonable is the error estimate?

The error estimate is smaller than the actual error. It is invalid when larger step size aor shorter final run time is applied.

2. Keep decreasing the time step. Does the error estimate diverge from the computed error? Why?

Yes, the error estimate diverges from the computed error at smaller time steps. The initial conditions applied to the estimate approximation cannot resolve such small time steps.

3. Keep increasing the time step. Does the error estimate diverge? What is happening with the numerical solution?

A divergence in error is observed from the fixed time step. This is due to instability when using a really large time step.

In [ ]: