Chapters 5 & 6: Integrals and their applications

Integrals in Scientific Python

There are many applications of integrals. A good example from ecology is given in Question #12 on p. 328 of S&D.

Approximation of an area under a function or set of functions

As indicated in section 5.1 of S&D, one approach to answering the question posed in #12 is to break up the area under the two curves into small rectangles and sum-up the areas of each rectangle.

For example, we could break up the interval between 2 and 3 into intervals of 0.1 and measure the height of a rectangle as the value of the functions at the midpoint of the interval:

[1] Import numpy library

In [2]:
```
import numpy as np
```

[2] Generate a vector of midpoints on the x-axis

In [13]:
```
x=np.linspace(2.05,2.95,num=10)
```

In [14]:
```
x
```

Out[14]:
```
array([ 2.05,  2.15,  2.25,  2.35,  2.45,  2.55,  2.65,  2.75,  2.85,
2.95])
```

[3] Calculate value of functions at each midpoint

In [15]:
```
def n_1(x):
    return (x-1)*(3-x)
```

In [16]:
```
def n_2(x):
    return (x-2)*(4-x)
```

In [17]:
```
y_1=n_1(x[5:10])
```

In [18]:
```
y_1
```

Out[18]:
```
array([ 0.6975,  0.5775,  0.4375,  0.2775,  0.0975])
```

In [27]:
```
y_2=n_2(x[0:5])
```

In [28]:
```
y_2
```

Out[28]:
```
array([ 0.0975,  0.2775,  0.4375,  0.5775,  0.6975])
```

[4] Calculate area of rectangles around each midpoint

In [29]:
```
tot_r_1=np.sum(y_1*0.1)
```

In [30]:
```
tot_r_1
```

Out[30]:
```
0.20874999999999999
```

In [31]:
```
tot_r_2=np.sum(y_2*0.1)
```

In [32]:
```
tot_r_2
```

Out[32]:
```
0.20874999999999999
```

[5] Add areas together to get total area

In [33]:
```
tot_r_1+tot_r_2
```

Out[33]:
```
0.41749999999999998
```

Numerical integration of an area under a function or set of functions (see section 5.2 of S&D)

The integral allows us to directly calculate the area under the set of functions given in question #12.

Numerical evaluation of integrals in Scientific Python is done using a new library, namely the "scipy" library (https://www.scipy.org/scipylib/index.html (https://www.scipy.org/scipylib/index.html)).

[1] Import the subpackage integrate from scipy

In [3]:
```
from scipy import integrate
```

[2] Calculate the integrals under each function and add together

Note: in scipy, the "quad" function computes a definite integral of a single variable (https://docs.scipy.org/doc/scipy/reference/integrate.html (https://docs.scipy.org/doc/scipy/reference/integrate.html)).

The area of niche overlap from x=2 to x=2.5 is

In [43]:
```
integrate.quad(n_2,2,2.5)
```

Out[43]:
```
(0.20833333333333334, 2.312964634635743e-15)
```

Note that the output from the integrate.quad function is the value of the integral and a measure of accuracy. To just get the value of the integral we can access the first element of the result:

In [44]:
```
integrate.quad(n_2,2,2.5)[0]
```

Out[44]:
```
0.20833333333333334
```

The total area is

In [45]:
```
integrate.quad(n_2,2,2.5)[0]+integrate.quad(n_1,2.5,3)[0]
```

Out[45]:
```
0.4166666666666667
```

Comparing this result to our approximation using rectangles indicates the inaccuracy of the rectangle approach.

Symbolic approach to answering question #12

You'll note that numerical integration is not necessarily perfectly accurate because although it is a more sophisticated version of the rectangle method using midpoints, there is error associated with it which is reflected in the accurary measure. The exact approach to calculating the area under a function or set of functions is to determine the antiderivative of a function using by evaluating an integral symbolically and then substitute values for the endpoints of the integral using the Evaluation Theorem. (See section 5.3 of S&D).

[1] Import sympy to perform the symbolic calculations

In [4]:
```
import sympy as sp
```

[2] Symbolically evaluate integrals and substitute endpoints

In [47]:
```
x=sp.symbols('x')
```

In [48]:
```
n_1=sp.Function('n_1')
n_2=sp.Function('n_2')
```

In [49]:
```
def n_1(x):
    return (x-1)*(3-x)
```

In [50]:
```
def n_2(x):
    return (x-2)*(4-x)
```

Below are the symbolic solutions to the integrals for the two functions, using capital "N" to indicate the antiderivative.

In [59]:
```
N_1=sp.integrate(n_1(x),x)
N_1
```

Out[59]:
```
-x**3/3 + 2*x**2 - 3*x
```

In [60]:
```
N_2=sp.integrate(n_2(x),x)
N_2
```

Out[60]:
```
-x**3/3 + 3*x**2 - 8*x
```

Below the numerical value of niche overlap is calculated by substituting endpoints into each subregion of the integral

In [75]:
```
(N_1.subs(x,sp.S(3))-N_1.subs(x,sp.S(5)/2))+(N_2.subs(x,sp.S(5)/2)-N_2.su
```

Out[75]:
```
5/12
```

In [77]:
```
5/12.
```

Out[77]:
```
0.4166666666666667
```

Note that the exact answer is equal to the numerical integral for these sets of functions defining the niches of species.

Why would we evaluate an integral numerically, as opposed to symbolically given that numerical integration is not perfectly accurate?

The answer is that many functions do not have an integral that evaluates symbolically, so numerical methods are required to approximate the integral.

Further applications of integrals

Dialysis - Question 60, p. 361

Note that for this problem, no numerical values are given for the parameters in the model for the rate of removal of urea. Therefore, we will evaluate the integral symbolically.

[1] Define the function

In [78]:
```
sp.Function('c')
t,K,V,c_0=sp.symbols('t,K,V,c_0')
```

In [79]:
```
def c(t):
    return K/V*c_0*sp.exp(-K*t/V)
```

[2] Integrate the function symbolically, i.e. find the antiderivative of the function

In [83]:
```
C_t=sp.integrate(c(t),t)
C_t
```

Out[83]:
```
-c_0*exp(-K*t/V)
```

[3] Use the Evaluation Theorem to evaluate the integral across an interval

In [84]:
```
C_t.subs(t,30)-C_t.subs(t,0)
```

Out[84]:
```
c_0 - c_0*exp(-30*K/V)
```

[4] Interpretation

To help us interpret the integral, let's go back to the definition of a definite integral in Section 5.2 of S&D. A definite integral is equal to the following

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{\infty} f(x_i)\Delta x$$

in our case the function is c and it is a function of t, so let's make these substitutions

$$\int_a^b c(t)dt = \lim_{n\to\infty} \sum_{i=1}^{\infty} c(t_i)\Delta t$$

$c(t)$ is the rate urea is removed from by dialysis at time t, so $\lim_{\Delta t\to 0} c(t)\Delta t$ is the amount of urea that is removed over a very small time interval.

An integral sums these very small amounts across the interval from a to b, which corresponds to the total amount of urea that is removed from time points a to b, where in our case $a = 0$ and $b = 30$.

Air intake into lungs, Question 59, p. 361

Answer this question. As part of your answer, plot the function $f(t)$ that gives the rate of intake of air at time t from $t = 0$ to $t = 10$. In addition, plot the function $F(t)$, which is the function you derive that gives the volume of inhaled air at time t. Inspect this plot. Does it make sense? Why?

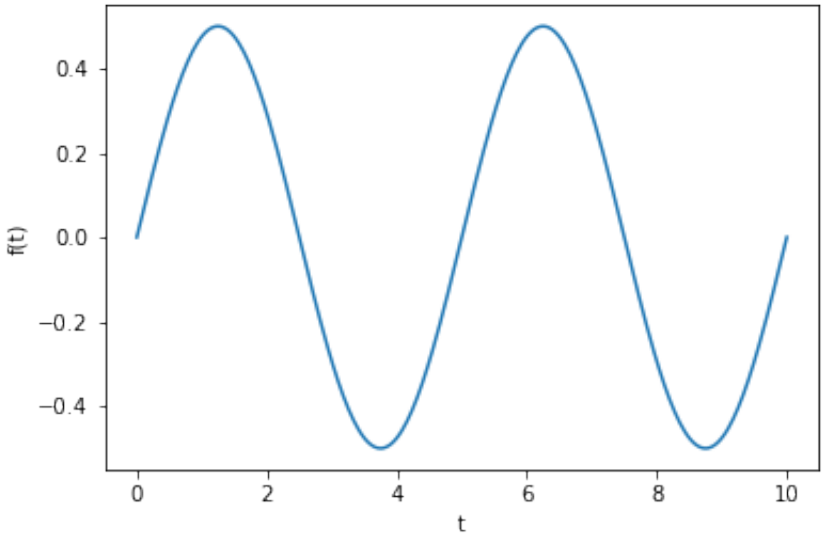
In [6]:
```
f=sp.Function('f')
t=sp.symbols('t')
```

In [7]:
```
def f(t):
    return sp.S(1)/2*sp.sin(sp.S(2)*sp.pi*t/sp.S(5))
```

In [5]:
```
import matplotlib.pyplot as plt
```

In [8]:
```
t_val=np.linspace(0,10,256,endpoint=True)
lam_f=sp.lambdify(t,f(t),"numpy")
f_val=lam_f(t_val)

plt.xlabel('t')
plt.ylabel('f(t)')
plt.show(plt.plot(t_val,f_val))
```



In [115]:
```
sp.integrate(f(t),t)
```

Out[115]:
```
-5*cos(2*pi*t/5)/(4*pi)
```

In [12]:
```
C=sp.symbols('C')
```

In [19]:
```
sp.solve((-5*sp.cos(2*sp.pi*t/5)/(4*sp.pi)).subs(t,0)+C,C)
```

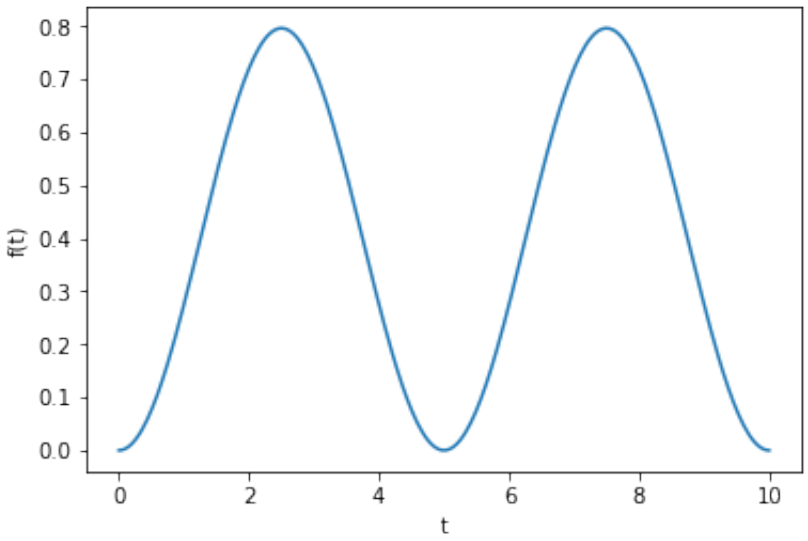
Out[19]:
```
[5/(4*pi)]
```

In [24]:
```
-5*sp.cos(2*sp.pi*t/5)/(4*sp.pi)+5/(4*sp.pi)
```

Out[24]:
```
-5*cos(2*pi*t/5)/(4*pi) + 5/(4*pi)
```

In [21]:
```
t_val=np.linspace(0,10,256,endpoint=True)
lam_F=sp.lambdify(t,-5*sp.cos(2*sp.pi*t/5)/(4*sp.pi)+5/(4*sp.pi),"numpy")
F_val=lam_F(t_val)

plt.xlabel('t')
plt.ylabel('f(t)')
plt.show(plt.plot(t_val,F_val))
```



In [29]:
```
(-5*sp.cos(2*sp.pi*t/5)/(4*sp.pi)+5/(4*sp.pi)).subs(t,2).evalf()
```

Out[29]:
```
0.719784991980041
```

In [31]:
```
(5/(4*sp.pi)*(1-sp.cos(2*sp.pi*t/5))).subs(t,2).evalf()
```

Out[31]:
```
0.719784991980041
```

Survival and renewal - Example 1, p. 401

Integrals can be used to project properties of a population into the future, such as population size, such that the number of individuals at time $tN(t)$ is

$$N(t) = N(0)S(t) + \int_0^t R(t)S(t-\tau)d\tau$$

where $S(t)$ is the probability an individual survives t time units and $R(t)$ is the rate of reproduction at the population level at time t.

In the example, they define $R(t)$ by the function $R(t) = 720e^{0.1t}$, which may seem a little strange to an ecologist.

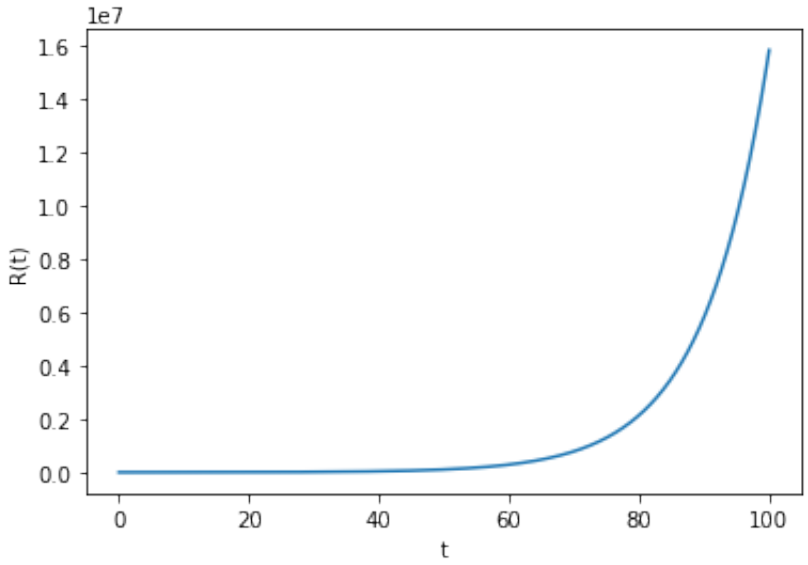
To see why it is strange, plot R(t) for t beyond 10 years, say 100 years. Given information from the plot, is it accurate to project beyond 10 years?

In [5]:
```
import matplotlib.pyplot as plt
import numpy as np
import sympy as sp
from scipy import integrate
```

In [7]:
```
t=sp.symbols('t')

t_val=np.linspace(0,100,2560,endpoint=True)
lam_R=sp.lambdify(t,720*sp.exp(0.1*t),"numpy")
R_val=lam_R(t_val)

plt.xlabel('t')
plt.ylabel('R(t)')
plt.show(plt.plot(t_val,R_val))
```



An alternate example, that may be more practical is calculating the reproductive value of an individual of age a, which is the value of an organism's future offspring relative to its own current value (Roughgarden 1979).

Define $l(x)$ to be a function that gives the probability an individual survives to age x and $m(x)$ the number of offspring at age x (assume hermaphrodism). The reproductive value of an individual is

$$v_a = \frac{e^{ra}}{l(a)} \int_a^\infty e^{-rx} l(x) m(x) dx$$

where r is the growth rate of the population.

Let's assume that $l(x) = e^{-bt}$ and $m(x) = ce^{\frac{-(x-d)^2}{g}}$.

Plot the functions $l(x)$ and $m(x)$ for r=0.01, b=0.1, c=10, d=8 and g=100 for x from 0 to 40. Do the plots make biological sense?

Treating the parameters symbolically, try to evaluate the integral

∞ _

$$\int_a^{\infty} e^{-rx} l(x) m(x) dx$$

by executing the code provided below.

When you become impatient, click the "Interrupt" option under the "Kernel" drop-down menu to stop the evaluation of the integral. There is no closed-form formula for this integral and the integral will not evaluate. Consequently, numerical analysis is required or approximations.

In [41]:
```
b,c,d,g,r,x=sp.symbols('b,c,d,g,r,x')
```

In [ ]:
```
sp.integrate(sp.exp(r*x)*sp.exp(-b*x)*c*sp.exp(-(x-d)**2/g),x)
```

Let's try numerically evaluating the integral using the parameter values from the plots above and for a=2.

In [87]:
```
integrate.quad(sp.exp(0.01*x)*sp.exp(-0.1*x)*10*sp.exp(-(x-8)**2/100),2,n
```

```
---------------------------------------------------------------------
------
ValueError                                Traceback (most recent call
last)
<ipython-input-87-1728e7ff5026> in <module>()
----> 1 integrate.quad(sp.exp(0.01*x)*sp.exp(-0.1*x)*10*sp.exp(-(x-8)
**2/100),2,np.inf)

/Users/cort/anaconda/lib/python3.6/site-packages/scipy/integrate/quad
pack.py in quad(func, a, b, args, full_output, epsabs, epsrel, limit,
points, weight, wvar, wopts, maxp1, limlst)
    321     if (weight is None):
    322         retval = _quad(func, a, b, args, full_output, epsabs,
epsrel, limit,
--> 323                        points)
    324     else:
    325         retval = _quad_weight(func, a, b, args, full_output,
epsabs, epsrel,

/Users/cort/anaconda/lib/python3.6/site-packages/scipy/integrate/quad
```

Using the code provided we got an error. We need to "lambdify" the function because it involves the sympy function exp(). Below is an approach to lambdify a function in the context of integration using scipy.

In [94]:
```
func = lambda x: sp.exp(0.01*x)*sp.exp(-0.1*x)*10*sp.exp(-(x-8)**2/100)
```

In [95]:
```
integrate.quad(func,2,np.inf)
```

Out[95]:
```
(61.69342638310371, 8.704829720595727e-08)
```

Let's imagine that we want to plot v_a for a from 0 to 30. How can we do this given that we do not have a closed-form formula for v_a. One approach is the following:

First define a function that evaluates v_a for a specific value of a.

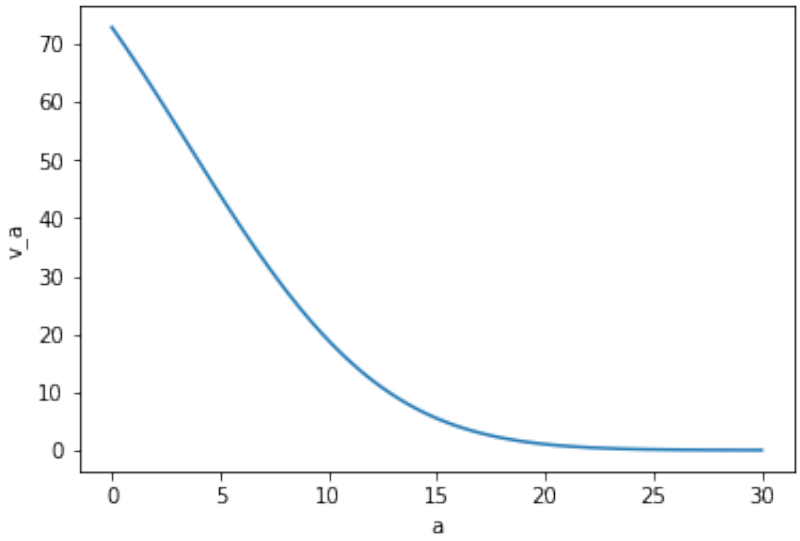
In [90]:
```
def v_a(a):
    return integrate.quad(func,a,np.inf)[0]
```

Next, plot v_a for a reasonable amount of points between 0 and 30.

In [91]:
```
a_val=np.linspace(0,30,100,endpoint=True)

v_a_val=np.empty([100])
for i in range(0,100):
    v_a_val[i]=v_a(a_val[i])

plt.xlabel('a')
plt.ylabel('v_a')
plt.show(plt.plot(a_val,v_a_val))
```



Note, we cannot use the code v_a_val = v_a(a_val) to get the values of v_a across the array a_val in the context of using integrate.quad in the definition of the function v_a. Instead, we generated an empty list of length a_val and then filled this list with values from v_a.

In [ ]: