*4.4 CONTINUOUS MODELS

If we want to more accurately model the situation in our fishing pond, we need to acknowledge that the size of the fish population does not really change only once a year. Like virtually all natural processes, the change happens continually or, mathematically speaking, *continuously*, over time. To more accurately model continuous natural processes, we need to update our population size more often, using smaller time steps. For example, we could update the size of the fish population every month instead of every year by replacing every annual update of

```
population = population + 0.08 * population - 1500
```

with twelve monthly updates:

```
for month in range(12):
    population = population + (0.08 / 12) * population - (1500 / 12)
```

Since both the growth rate of `0.08` and the harvest of `1500` are based on 1 year, we have divided both of them by 12.

Reflection 1 *Is the final value of* `population` *the same whether we update it annual or monthly?*

If the initial value of `population` is `12000`, the value of `population` after one annual update is `11460.0` while the final value after 12 monthly updates is `11439.753329049303`. Because the rate is "compounding" monthly, it reduces the population slightly more quickly.

This is exactly how bank loans work. The bank will quote an annual percentage rate (APR) of, say, 6% (or 0.06) but then compound interest monthly at a rate of $6/12\% = 0.5\%$ (or 0.005), which means that the actual annual rate of interest you are being charged, called the annual percentage yield (APY), is actually $(1 + 0.005)^{12} - 1 \approx 0.0617 = 6.17\%$. The APR, which is really defined to be the monthly rate times 12, is sometimes also called the "nominal rate." So we can say that our fish population is increasing at a nominal rate of 8%, but updated every month.

Difference equations

A population model like this is expressed more accurately with a ***difference equation***, also known as a ***recurrence relation***. If we let $P(t)$ represent the size of the fish population at the end of year t, then the difference equation that defines our original model is

$$P(t) = \underbrace{P(t-1)}_{\text{previous year's population}} + \underbrace{0.08 \cdot P(t-1)}_{\text{proportional increase}} - \underbrace{1500}_{\text{harvest}}$$

or, equivalently,

$$P(t) = 1.08 \cdot P(t-1) - 1500.$$

In other words, the size of the population at the end of year t is 1.08 times the size of the population at the end of the previous year $(t-1)$, minus 1500. The initial population or, more formally, the *initial condition* is $P(0) = 12{,}000$. We can find the projected population size for any given year by starting with $P(0)$, and using the difference equation to compute successive values of $P(t)$. For example, suppose we wanted to know the projected population size four years from now, represented by $P(4)$. We start with the initial condition: $P(0) = 12{,}000$. Then, we apply the difference equation to $t = 1$:

$$P(1) = 1.08 \cdot P(0) - 1500 = 1.08 \cdot 12{,}000 - 1500 = 11{,}460.$$

Now that we have $P(1)$, we can compute $P(2)$:

$$P(2) = 1.08 \cdot P(1) - 1500 = 1.08 \cdot 11{,}460 - 1500 = 10{,}876.8.$$

Continuing,

$$P(3) = 1.08 \cdot P(2) - 1500 = 1.08 \cdot 10{,}876.8 - 1500 = 10{,}246.94$$

and

$$P(4) = 1.08 \cdot P(3) - 1500 = 1.08 \cdot 10{,}246.94 - 1500 = 9{,}566.6952.$$

So this model projects that the bass population in 4 years will be 9,566. This is the same process we followed in our `for` loop in Section 4.1.

To turn this *discrete* model into a *continuous* model, we define a small update interval, which is customarily named Δt (Δ represents "change," so Δt represents "change in time"). If, for example, we want to update the size of our population every month, then we let $\Delta t = 1/12$ (1/12 of a year). Then we express our difference equation as

$$P(t) = P(t - \Delta t) + (0.08 \cdot P(t - \Delta t) - 1500) \cdot \Delta t$$

This difference equation is defining the population size at the end of year t in terms of the population size one Δt fraction of a year ago. For example, if t is 3 and Δt is 1/12, then $P(t)$ represents the size of the population at the end of year 3 and $P(t - \Delta t)$ represents the size of the population at the end of "year 2 11/12," equivalent to one month earlier. Notice that both the growth rate and the harvest number are scaled by Δt, just as we did in the `for` loop on page O4.4-1.

To implement this model, we need to make some analogous changes to the algorithm from Section 4.1. First, we need to pass in the value of Δt as a parameter so that we have control over the accuracy of the approximation. Second, we need to modify the number of iterations in our loop to reflect $1/\Delta t$ update events each year; the number of iterations becomes `years` $\cdot\ (1/\Delta t)$ = `years`$/\Delta t$. Third, we need to alter how the accumulator is updated in the loop to reflect this new type of difference equation. These changes are reflected below. We use `dt` to represent Δt.

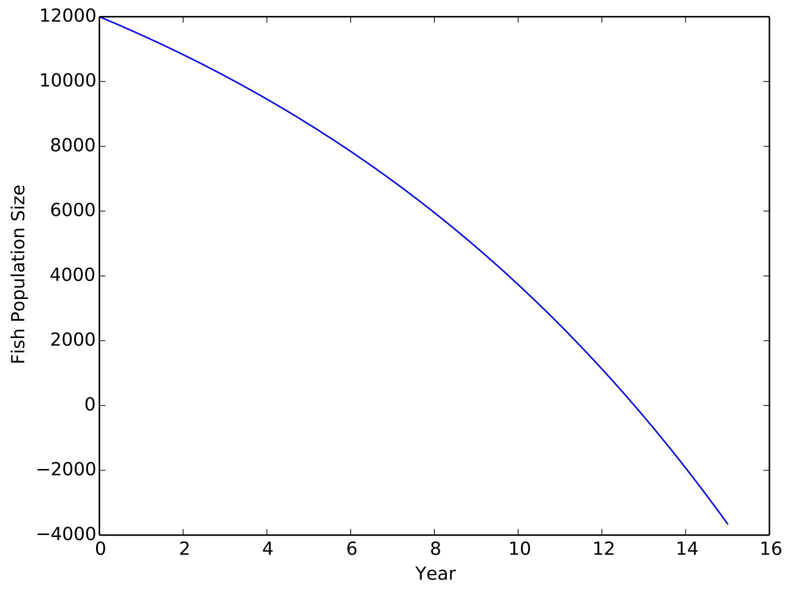


Figure 1 The plot produced by calling `pond(15, 12000, 1500, 0.01)`.

```
def pond(years, initialPopulation, harvest, dt):
    """ (docstring omitted) """

    population = initialPopulation
    for step in range(1, int(years/dt) + 1):
        population = population + (0.08 * population - harvest) * dt

    return population
```

Reflection 2 *Why do we use* `range(1, int(years/dt) + 1)` *in the* `for` *loop instead of* `range(int(years/dt))`*?*

We start the `for` loop at one instead of zero because the first iteration of the loop represents the first time step of the simulation. The initial population size assigned to `population` before the loop represents the population at time zero.

To plot the results of this simulation, we use the same technique that we used in Section 4.2. But we also use a list accumulator to create a list of time values for the x axis because the values of the index variable `step` no longer represent years. In the following function, the value of `step * dt` is assigned to the variable `t`, and then appended to a list named `timeList`.[1]

[1]Recall from Section 3.3 that this is better than adding `dt` in each iteration.

```
import matplotlib.pyplot as pyplot

def pond(years, initialPopulation, harvest, dt):
    """Simulates a fish population in a pond, and plots annual population
       size growing at a nominal annual rate of 8% with an annual harvest.

    Parameters:
        years:             number of years to simulate
        initialPopulation: the initial population size
        harvest:           the size of the annual harvest
        dt:                value of "Delta t" in the simulation
                           (fraction of a year)

    Return value: the final population size
    """

    population = initialPopulation
    populationList = [initialPopulation]
    timeList = [0]
    t = 0
    for step in range(1, int(years/dt) + 1):
        population = population + (0.08 * population - harvest) * dt
        populationList.append(population)
        t = step * dt
        timeList.append(t)

    pyplot.plot(timeList, populationList)
    pyplot.xlabel('Year')
    pyplot.ylabel('Fish Population Size')
    pyplot.show()
    return population
```

Reflection 3 *Create a trace table that shows how the values of* `step`, `t`, *and* `timeList` *change in the first few iterations of the loop when* `dt` *is 0.01.*

Figure 1 shows a plot produced by calling this function with $\Delta t = 0.01$. Compare this plot to the one in Section 4.1. To actually come close to approximating a real continuous process, we need to use very small values of Δt. But there are tradeoffs involved in doing so, which we discuss in more detail later in this section.

Radiocarbon dating

When archaeologists wish to know the ages of organic relics, they often turn to *radiocarbon dating*. Both carbon-12 (^{12}C) and carbon-14 (or radiocarbon, ^{14}C) are isotopes of carbon that are present in our atmosphere in a relatively constant proportion. While carbon-12 is a stable isotope, carbon-14 is unstable and decays at a known rate. All living things ingest both isotopes, and die possessing them in the same proportion as the atmosphere. Thereafter, an organism's acquired carbon-14 decays at a known rate, while its carbon-12 remains intact. By examining the current ratio of carbon-12 to carbon-14 in organic remains (up to about 60,000 years old), and comparing this ratio to the known ratio in the atmosphere, we can approximate how long ago the organism died.

The annual decay rate (more correctly, the decay constant[2]) of carbon-14 is about $k = -0.000121$. Since radioactive decay is a continuous process, $Q(t)$, the quantity of carbon-14 present at the beginning of year t, needs to be defined in terms of $Q(t - \Delta t)$, the quantity of carbon-14 a small Δt fraction of a year ago. Therefore, the difference equation modeling the decay of carbon-14 is

$$Q(t) = Q(t - \Delta t) + k \cdot Q(t - \Delta t) \cdot \Delta t\ .$$

Since the decay constant k is based on one year, we scale it down for an interval of length Δt by multiplying it by Δt. We will represent the initial condition with $Q(0) = q$, where q represents the initial quantity of carbon-14.

Although this is a completely different application, we can implement the model the same way we implemented our continuous fish population model.

```python
import matplotlib.pyplot as pyplot

def decayC14(originalAmount, years, dt):
    """Approximates the continuous decay of carbon-14.

    Parameters:
        originalAmount: the original quantity of carbon-14 (g)
        years:          number of years to simulate
        dt:             value of "Delta t" in the simulation
                        (fraction of a year)

    Return value: final quantity of carbon-14 (g)
    """

    k = -0.000121
    amount = originalAmount
    t = 0
    timeList = [0]                     # x values for plot
    amountList = [amount]              # y values for plot

    for step in range(1, int(years/dt) + 1):
        amount = amount + k * amount * dt
        t = step * dt
        timeList.append(t)
        amountList.append(amount)

    pyplot.plot(timeList, amountList)
    pyplot.xlabel('Years')
    pyplot.ylabel('Quantity of carbon-14')
    pyplot.show()

    return amount
```

Like all of our previous accumulators, this function initializes our accumulator variable, named `amount`, before the loop. Then the accumulator is updated in the body of the loop according to the difference equation above. Figure 2 shows an example plot from this function.

[2]The probability of a carbon-14 molecule decaying in a very small fraction Δt of a year is $k\Delta t$.

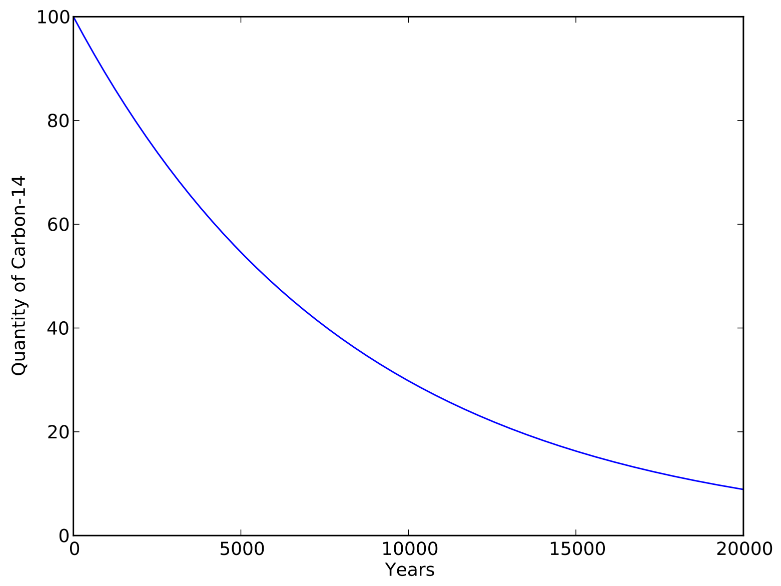


Figure 2 Plot of carbon-14 decay generated with `decayC14(100, 20000, 0.01)`.

Reflection 4 *How much of 100 g of carbon-14 remains after 5,000 years of decay? Try various Δt values ranging from 1 down to 0.001. What do you notice?*

Tradeoffs between accuracy and time

Approximations of continuous models are more accurate when the value of Δt is smaller. However, accuracy has a cost. The `decayC14` function has time complexity proportional to $y/\Delta t$, where y is the number of years we are simulating, since this is how many times the `for` loop iterates. So, if we want to improve accuracy by dividing Δt in half, this will directly result in our algorithm requiring twice as much time to run. If we want Δt to be one-tenth of its current value, our algorithm will require ten times as long to run.

To get a sense of how decreasing values of Δt affect the outcome, let's look at what happens in our `decayC14` function with `originalAmount = 1000` and `years = 2000`. The table below contains these results, with the theoretically derived answer in the last row (see Tangent 1). The error column shows the difference between the result and this value for each value of `dt`. All values are rounded to three significant digits to reflect the approximate nature of the decay constant. We can see from the table that smaller values of `dt` do indeed provide closer approximations.

Tangent 1: Differential equations

If you have taken a course in calculus, you may recognize each of these continuous models as an approximation of a *differential equation*. A differential equation is an equation that relates a function to the rate of change of that function, called the derivative. For example, the differential equation corresponding to the carbon-14 decay problem is

$$\frac{dQ}{dt} = kQ$$

where Q is the quantity of carbon-14, dQ/dt is the rate of change in the quantity of carbon-14 at time t (i.e., the derivative of Q with respect to t), and k is the decay constant. Solving this differential equation using basic calculus techniques yields

$$Q(t) = Q(0)\, e^{kt}$$

where $Q(0)$ is the original amount of carbon-14. We can use this equation to directly compute how much of 1000 g of carbon-14 would be left after 2,000 years with:

$$Q(2000) = 1000\, e^{2000k} \approx 785.0562.$$

Although simple differential equations like this are easily solved if you know calculus, most realistic differential equations encountered in the sciences are not, making approximate iterative solutions essential. The approximation technique we are using in this chapter, called *Euler's method*, is the most fundamental, and introduces error proportional to Δt. More advanced techniques seek to reduce the approximation error further.

`dt`	final `amount`	error	time
1.0	785.0447	0.0115	500 μs
0.1	785.0550	0.0012	5 ms
0.01	785.0561	0.0001	50 ms
0.001	785.0562	0.0	500 ms
0.0001	785.0562	0.0	5 sec
—	785.0562	—	—

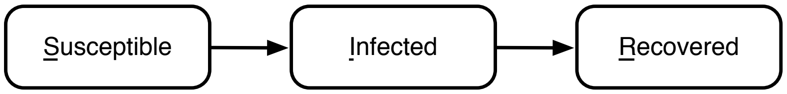
Reflection 5 *What is the relationship between the value of* `dt` *and the error? What about between the value of* `dt` *and the execution time?*

Each row in the table represents a computation that took ten times as long as that in the previous row because the value of `dt` was ten times smaller. But the error is also ten times smaller. Is this tradeoff worthwhile? The answer depends on the situation. Certainly using `dt` = 0.0001 is not worthwhile because it gives the same answer (to three significant digits) as `dt` = 0.001, but takes ten times as long.

These types of tradeoffs — quality versus cost — are common in all fields, and computer science is no exception. Building a faster memory requires more expensive technology. Ensuring more accurate data transmission over networks requires more overhead. And finding better approximate solutions to hard problems requires more time.

Simulating an epidemic

Real populations interact with each other and the environment in complex ways. Therefore, to accurately model them requires an interdependent set of difference equations, called *coupled difference equations*. In 1927, Kermack and McKendrick [29] introduced such a model for the spread of infectious disease called the *SIR model*. The "S" stands for the "susceptible" population, those who may still acquire the disease; "I" stands for the "infected" population, and "R" stands for the "recovered" population. In this model, we assume that, once recovered, an individual has built an immunity to the disease and cannot reacquire it. We also assume that the total population size is constant, and no one dies from the disease. Therefore, an individual moves from a "susceptible" state to an "infected" state to a "recovered" state, where she remains, as pictured below.



These assumptions apply well to common viral infections, like the flu and COVID-19, assuming a relatively small fraction of the infected population dies.

A virus like the flu travels through a population more quickly when there are more infected people with whom a susceptible person can come into contact. In other words, a susceptible person is more likely to become infected if there are more infected people. Also, since the total population size does not change, an increase in the number of infected people implies an identical decrease in the number who are susceptible. Similarly, a decrease in the number who are infected implies an identical increase in the number who have recovered. Like most natural processes, the spread of disease is fluid or continuous, so we will need to model these changes over small intervals of Δt, as we did in the radioactive decay model.

We need to design three difference equations that describe how the sizes of the three groups change. We will let $S(t)$ represent the number of susceptible people on day t, $I(t)$ represent the number of infected people on day t, and $R(t)$ represent the number of recovered people on day t.

The recovered population has the most straightforward difference equation. The size of the recovered group only increases; when infected people recover they move from the infected group into the recovered group. The number of people who recover at each step depends on the number of infected people at that time and the *recovery rate* r: the average fraction of people who recover each day.

Reflection 6 *What factors might affect the recovery rate in a real outbreak?*

Since we will be dividing each day into intervals of length Δt, we will need to use a scaled recovery rate of $r \cdot \Delta t$ for each interval. So the difference equation describing the size of the recovered group on day t is

$$R(t) = R(t - \Delta t) + \underbrace{r \cdot I(t - \Delta t) \cdot \Delta t}_{\text{newly recovered, from } I}$$

Since no one has yet recovered on day 0, we set the initial condition to be $R(0) = 0$.

Next, we will consider the difference equation for $S(t)$. The size of the susceptible population only decreases by the number of newly infected people. This decrease depends on the number of susceptible people, the number of infected people with whom they can make contact, and the rate at which these potential interactions produce a new infection. The number of possible interactions between susceptible and infected individuals at time $t - \Delta t$ is simply their product: $S(t-\Delta t)\cdot I(t-\Delta t)$. If we let d represent the rate at which these interactions produce an infection, then our difference equation is

$$S(t) = S(t-\Delta t) - \underbrace{d\cdot S(t-\Delta t)\cdot I(t-\Delta t)\cdot \Delta t}_{\text{newly infected, to } I}.$$

Reflection 7 *What factors might affect the infection rate in a real outbreak?*

If N is the total size of our population, then the initial condition is $S(0) = N - 1$ because we will start with one infected person, leaving $N-1$ who are susceptible.

The difference equation for the infected group depends on the number of susceptible people who have become newly infected and the number of infected people who are newly recovered. These numbers are precisely the number leaving the susceptible group and the number entering the recovered group, respectively. We can simply copy those from the equations above.

$$I(t) = I(t-\Delta t) + \underbrace{d\cdot S(t-\Delta t)\cdot I(t-\Delta t)\cdot \Delta t}_{\text{newly infected, from } S} - \underbrace{r\cdot I(t-\Delta t)\cdot \Delta t}_{\text{newly recovered, to } R}.$$

Since we are starting with one infected person, we set $I(0) = 1$.

The program below is a "skeleton" for implementing this model with a recovery rate $r = 0.25$ and an infection rate $d = 0.0004$. These rates imply that the average infection lasts $1/r = 4$ days and there is a 0.04% chance that an encounter between a susceptible person and an infected person will occur and result in a new infection. The program also demonstrates how to plot and label several curves in the same figure, and display a legend. We leave the implementation of the difference equations in the loop as an exercise.

```
import matplotlib.pyplot as pyplot

def SIR(population, days, dt):
    """Simulates the SIR model of infectious disease and plots the
       population sizes over time.

    Parameters:
        population: the population size
        days:       number of days to simulate
        dt:         the value of "Delta t" in the simulation
                    (fraction of a day)

    Return value: None
    """

    susceptible = population - 1  # susceptible count = S(t)
    infected = 1.0                # infected count = I(t)
    recovered = 0.0               # recovered count = R(t)
    recRate = 0.25                # recovery rate r
    infRate = 0.0004              # infection rate d
    SList = [susceptible]
    IList = [infected]
    RList = [recovered]
    timeList = [0]

    # Loop using the difference equations goes here.
```

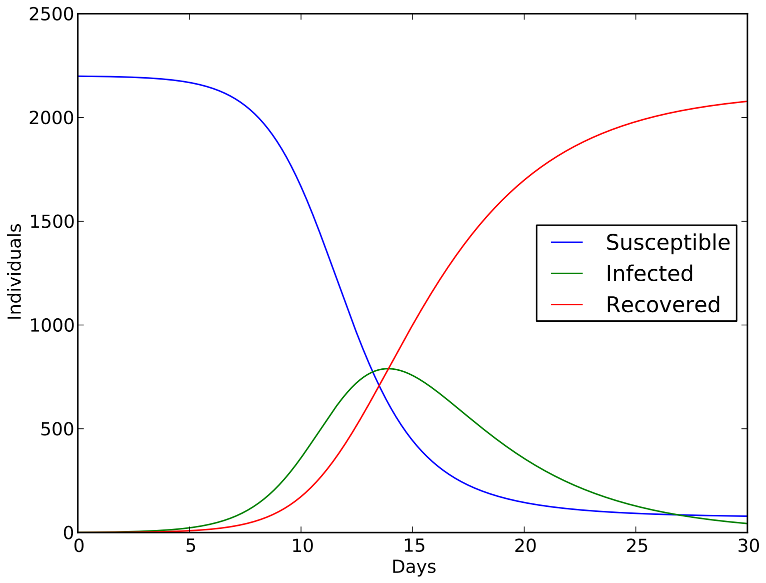


Figure 3 Output from the SIR model with 2,200 individuals over 30 days with $\Delta t = 0.01$.

```
pyplot.plot(timeList, SList, label = 'Susceptible')
pyplot.plot(timeList, IList, label = 'Infected')
pyplot.plot(timeList, RList, label = 'Recovered')
pyplot.legend(loc = 'center right')
pyplot.xlabel('Days')
pyplot.ylabel('Individuals')
pyplot.show()
```

Figure 3 shows the output of the model with 2,200 individuals (students at a small college, perhaps?) over 30 days. We can see that the infection peaks after about 2 weeks, then decreases steadily. After 30 days, the virus is just about gone, with only about 40 people still infected. We also notice that not everyone is infected after 30 days; about 80 people are still healthy.

Reflection 8 *Look at the relationships among the three population sizes over time in Figure 3. How do the sizes of the susceptible and recovered populations cause the decline of the infected population after the second week? What other relationships do you notice?*

When we are implementing a coupled model, we need to be careful to compute the change in each population size based on population sizes from the *previous* iteration rather than population sizes previously computed in *this* iteration. Therefore, we need to compute all of the changes in population sizes for time t, denoted $\Delta R(t)$, $\Delta S(t)$ and $\Delta I(t)$, based on the previous population sizes, before we update the new population sizes in each iteration. In other words, in each iteration, we first compute

$$\begin{aligned}\Delta R(t) &= r \cdot I(t-\Delta t)\cdot \Delta t,\\ \Delta S(t) &= d \cdot S(t-\Delta t)\cdot I(t-\Delta t)\cdot \Delta t, \text{ and}\\ \Delta I(t) &= \Delta S(t) - \Delta R(t).\end{aligned}$$

Then, we use these values to update the new population sizes:

$$\begin{aligned}R(t) &= R(t-\Delta t) + \Delta R(t),\\ S(t) &= S(t-\Delta t) - \Delta S(t), \text{ and}\\ I(t) &= I(t-\Delta t) + \Delta I(t).\end{aligned}$$

Interestingly, the SIR model can also be used to model the diffusion of ideas, fads, or memes in a population. A "contagious" idea starts with a small group of people and can "infect" others over time. As more people adopt the idea, it spreads more quickly because there are more potential interactions between those who have adopted the idea and those who have not. Eventually, people move on to other things and forget about the idea, thereby "recovering."

Exercises

Write a function for each of the following problems.

4.4.1* Suppose we have a pair of rabbits, one male and one female. Female rabbits are mature and can first mate when they are only one month old, and have a gestation period of one month. Suppose that every mature female rabbit mates every month and produces exactly one male and one female offspring one month later. No rabbit ever dies.

(a) Write a difference equation $R(m)$ representing the number of female rabbits at the end of each month. (Note that this is a discrete model; there are no Δt's.) Start with the first month and compute successive months until you see a pattern. Don't forget the initial condition(s). To get you started:

- $R(0) = 1$, the original pair
- $R(1) = 1$, the original pair now one month old
- $R(2) = 2$, the original pair plus a newborn pair
- $R(3) = 3$, the original pair plus a newborn pair plus a one-month-old pair
- $R(4) = 5$, the pairs from the previous generation plus two newborn pairs

  ⋮
- $R(m) = ?$

(b) Write a function

```
rabbits(months)
```

that uses your difference equation from part (a) to compute and return the number of rabbits after the given number of `months`. Your function should also plot the population growth using `matplotlib`. (The sequence of numbers generated in this problem is called the *Fibonacci sequence*.)

4.4.2. Consider Exercise 4.1.21, but now assume that a bacteria colony grows continuously with a growth rate of $0.1\Delta t$ per small fraction Δt of an hour.

(a) Write a difference equation for $B(t)$, the size of the bacteria colony, using Δt's to approximately model continuous growth.

(b) Write a function

```
bacteria(population, dt, days)
```

that uses your difference equation from part (a) to compute and return the number of bacteria in the colony after the given number of `days`. The other parameters, `population` and `dt`, are the initial size of the colony and the step size (Δt), respectively. Your function should also plot the population growth using `matplotlib`.

4.4.3* Radioactive elements like carbon-14 are normally described by their *half-life*, the time required for a quantity of the element to decay to half its original quantity. Write a function

```
halflifeC14(originalAmount, dt)
```

that returns the half-life of carbon-14. Your function will need to simulate radioactive decay until the amount is equal to half of `originalAmount`.

The known half-life of carbon-14 is $5{,}730 \pm 40$ years. How close does this approximation come? Try it with different values of `dt`.

4.4.4. Finding the half-life of carbon-14 is a special case of radiocarbon dating. When an organic artifact is dated with radiocarbon dating, the fraction of extant carbon-14 is found relative to the content in the atmosphere. Let's say that the fraction in an ancient piece of parchment is found to be 70%. Then, to find the age of the artifact, we can use a generalized version of `halflifeC14` that iterates while `amount > originalAmount * 0.7`. Show how to generalize the `halflifeC14` function in the previous exercise by writing a new function

```
carbonDate(originalAmount, fractionRemaining, dt)
```

that returns the age of an artifact with the given fraction of carbon-14 remaining. Use this function to find the approximate age of the parchment.

4.4.5* Complete the implementation of the SIR simulation. Compare your results to Figure 3 to check your work.

4.4.6. Run your implementation of the SIR model for longer periods of time than that shown in Figure 3. Given enough time, will everyone become infected?

4.4.7. Suppose that enough people have been vaccinated that the infection rate is cut in half. What effect do these vaccinations have?

4.4.8. The SIS model represents an infection that does not result in immunity. In other words, there is no "recovered" population; people who have recovered re-enter the "susceptible" population.

(a) Write difference equations for this model.

(b) Copy and then modify the `SIR` function (renaming it `SIS`) so that it implements your difference equations.

(c) Run your function with the same parameters that we used for the SIR model. What do you notice? Explain the results.

4.4.9. Suppose there are two predator species that compete for the same prey, but do not directly harm each other. We might expect that, if the supply of prey was low and members of one species were more efficient hunters, then this would have a negative impact on the health of the other species. This can be modeled through the following pair of difference equations. $P(t)$ and $Q(t)$ represent the populations of predator species 1 and 2, respectively, at time t.

$$P(t) = P(t-\Delta t) + b_P \cdot P(t-\Delta t) \cdot \Delta t - d_P \cdot P(t-\Delta t) \cdot Q(t-\Delta t) \cdot \Delta t$$

$$Q(t) = Q(t-\Delta t) + b_Q \cdot Q(t-\Delta t) \cdot \Delta t - d_Q \cdot P(t-\Delta t) \cdot Q(t-\Delta t) \cdot \Delta t$$

The initial conditions are $P(0) = p$ and $Q(0) = q$, where p and q represent the initial population sizes of the first and second predator species, respectively. The values b_P and d_P are the birth rate and death rates (or, more formally, proportionality constants) of the first predator species, and the values b_Q and d_Q are the birth rate and death rate for the second predator species. In the first equation, the term $b_P \cdot P(t-\Delta t)$ represents the net number of births per month for the first predator species, and the term $d_P \cdot P(t-\Delta t) \cdot Q(t-\Delta t)$ represents the number of deaths per month. Notice that this term is dependent on the sizes of both populations: $P(t-\Delta t) \cdot Q(t-\Delta t)$ is the number of possible (indirect) competitions between individuals for food and d_P is the rate at which one of these competitions results in a death (from starvation) in the first predator species.

This type of model is known as *indirect, exploitative competition* because the two predator species do not directly compete with each other (i.e., eat each other), but they do compete indirectly by exploiting a common food supply.

Write a function

```
compete(pop1, pop2, birth1, birth2, death1, death2, years, dt)
```

that implements this model for a generalized indirect competition scenario, plotting the sizes of the two populations over time. Run your program using

- $p = 21$ and $q = 26$
- $b_P = 1.0$ and $d_P = 0.2$; $b_Q = 1.02$ and $d_Q = 0.25$
- $dt = 0.001$; 6 years

and explain the results. Here is a "skeleton" of the function to get you started.

```python
def compete(pop1, pop2, birth1, birth2, death1, death2, years, dt):
    """ YOUR DOCSTRING HERE """

    pop1List = [pop1]
    pop2List = [pop2]
    timeList = [0]
    for step in range(1, int(years/dt) + 1):
        # YOUR CODE GOES HERE

    pyplot.plot(timeList, pop1List, label = 'Population 1')
    pyplot.plot(timeList, pop2List, label = 'Population 2')
    pyplot.legend()
    pyplot.xlabel('Years')
    pyplot.ylabel('Individuals')
    pyplot.show()
```

Selected Exercise Solutions

4.4.1 The difference equation is

$$R(m) = R(m-1) + R(m-2)$$

where $R(m-1)$ represents the number of rabbits alive the previous month and $R(m-2)$ represents a new pair for every pair alive two months ago. The initial conditions are $R(1) = R(2) = 1$. This famous sequence is known as the Fibonacci numbers.

```
import matplotlib.pyplot as pyplot

def rabbits(months):
    population2 = 0   # R(m-2)
    population1 = 1   # R(m-1)
    populationList = [1]
    for month in range(1, months + 1):
        population = population1 + population2  # R(m)
        population2 = population1
        population1 = population
        populationList.append(population)
    pyplot.plot(range(months + 1), populationList)
    pyplot.xlabel('Month')
    pyplot.ylabel('Rabbit Population Size')
    pyplot.show()
    return population
```

4.4.3
```
def halflifeC14(originalAmount, dt):
    amount = originalAmount
    k = -0.00012096809434
    iterations = 0
    while amount > originalAmount / 2:
        amount = amount + k * amount * dt
        iterations = iterations + 1
    return iterations * dt
```

4.4.5
```
numIterations = int(days/dt) + 1
for i in range(1, numIterations):
    t = i * dt
    dR = (recRate * infected) * dt                # newly recovered
    dS = (infRate * susceptible * infected) * dt  # newly infected

    recovered = recovered + dR
    susceptible = susceptible - dS
    infected = infected + dS - dR

    timeList.append(t)
    SList.append(susceptible)
    IList.append(infected)
    RList.append(recovered)
```