Mathematical Methods for Biology, Part 1

Dmitry Kondrashov

Table of contents

Preface

In this book you will find a collection of mathematical ideas, computational methods, and modeling tools for describing biological systems quantitatively. Biological science, like all natural sciences, is driven by experimental results. As with other sciences, there comes a point when accumulated data needs to be analyzed quantitatively, in order to formulate and test explanatory hypotheses. Biology has reached this stage, thanks to an explosion of data from molecular biology techniques, such as large-scale DNA sequencing, protein structure determination, data on gene regulatory networks, and signaling pathways. Quantitative skills have become necessary for anyone hoping to make sense of biological research.

Mathematical modeling necessarily involves making simplifying assumptions. Reality is generally too complex to be captured in a few equations, and this is especially true for living systems. Simplicity in modeling has at least two virtues: first, simple models can be grasped by our limited minds, and second, it allows for meaningful testing of the assumptions against the evidence. A complex model that fits the data may not provide any insights about how the system works, whereas a simple model which does not fit all the data can indicate where the assumptions break down. We will learn how to construct progressively more sophisticated models, beginning with the ridiculously simple.

modeling assumptions: theoretical and empirical

A mathematical model postulates a precise relationship between several quantities, attempting to mimic the behavior of a real system. All models rest on a set of assumptions, postulating how various quantities are interrelated. These assumptions generally come from two sources: a scientific theory, or experimental observations. For instance, a model of molecular motion may rest on the assumption that Newton's laws hold true. On the other hand, the observation that a drug injected into the bloodstream of a mammal is metabolized with an exponential time dependence is empirical. The benefit of models based on well-established theories, sometimes known as "first-principles models", is that they can be constructed without prior experimental knowledge of a particular system. Newton's laws apply to all sorts of classical mechanics objects, ranging in size from molecules to planets. Some prefer first-principles models, because they rely on well-established scientific principles, while others will argue that an empirical model more accurately reflects the behavior of the system at hand. From a mathematical standpoint, there is no difference between the two types of models. We will use the same tools to construct and analyze models, regardless of their origin.

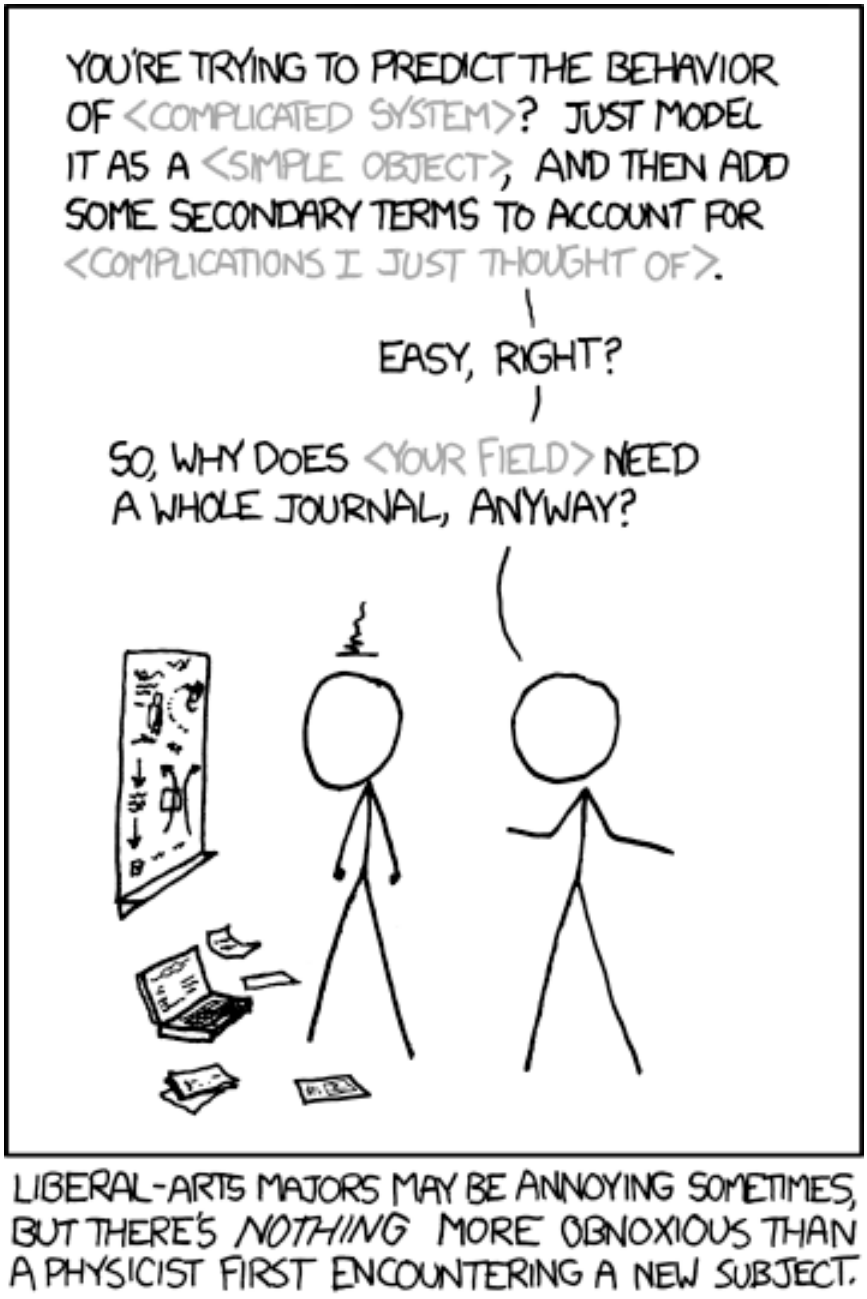


Figure 0.1: Beware: a little knowledge of mathematical modeling can lead to arrogance. <http://xkcd.com/793/

A stated assumption can be written as a mathematical relationship, usually in the form of an equation relating quantities of interest. A postulated assumption may be expressed in words as "X is proportional to Y", and can be written as the following equation: $X = aY$. Another model may postulate a relationship "X is inversely proportional to the product of Y and Z", which is expressed as $X = a/YZ$.

Suppose we want to model the relationship between the height of individuals (H) and their weight (W). Measuring those quantities in some population results in the observation that the weight is proportional to the height, with an additive correction. Then we can write the following mathematical model, based on the empirical evidence: $W = aH + c$

In electricity, Ohm's law governs the relationship between the flow of charged particles, called current (I), the electric potential (V) and the resistance of a conductor (R). This law states that the current through a conductor is proportional to the potential and inversely proportional to the resistance, and thus can be mathematically formulated:

$$I = \frac{V}{R}$$

variables and parameters

Mathematical models formulate relationships between different quantities that can be measured in real systems. There are two different types of quantities in models: *variables* and *parameters*. The same measurable quantity can be a variable or a parameter, depending on the role it plays in the model. A variable typically varies, either in time or in space, and the model tracks the changes in its value. On the other hand, a parameter typically usually stays the same for a particular manifestation of the model, e.g. an individual or a specific population. However, parameters can vary from individual to individual, or from population to population.

In the height and weight model above, the numbers H are W are the variables, which can change between different individuals. The parameters a and c can either be estimated from data for various subpopulations. Perhaps the values of the parameters are different for young people than for older people, or they are different for those who exercise regularly versus those who do not. Once the parameters have been set, one can predict W given H, or vice versa. Of course, since this is a model, it is only an approximation of reality. The deviations of predictions of the model from actual height or weight for an individual may tell us something interesting about the physiology of the individual.

There are three quantities in the equation for Ohm's law, and the distinction between variables and parameters depends on the actual system that is being modeled. In order to distinguish between the two, consider which quantity is set prior to the experiment, and which one may vary over the course of the situation we are trying to model. For instance, if voltage is being applied to a material with constant resistance, and the potential may be varied, then V is the

independent variable, I is the dependent variable, and R is a parameter. On the other hand, if the setup uses a variable resistor (known as a potentiometer or pot), and the voltage remains constant, then V is a parameter, while I and R are variables. If both the voltage V and the resistance R can vary at the same time, then all three quantities are variables.

units and dimensions

Each variable and parameter has its own *dimension*, which describes the physical or biological meaning of the quantity. Examples are time, length, number of individuals, or concentration per time. It is important to distinguish the dimension of a quantity from the *units* of measurement. The same quantity can be measured in different units: length can be in meters or feet, population size can be expressed in individuals or millions of individuals. The value of a quantity depends on the units of measurement, but its essential dimensionality does not.

There is a fundamental requirement of mathematical modeling: all the terms in an equation must agree in dimensionality; e.g. time cannot be added to number of sheep, since this sum has no biological meaning. In order to express this rule, we will write the dimension of a quantity X as $[X]$. While X refers to a numerical value, $[X]$ describes its physical meaning. Then the above statement can be illustrated by the following example:

$$aX = bY^2 \implies [aX] = [bY^2]$$

In the equation \$W = a H + c \$ all the terms must have the dimension of weight, because that is the meaning of the left hand side of the equation. Therefore, c has the dimensions of weight as well. H of course has the dimension of length, so this implies that the parameter a has dimensions of weight divided by length. This can be summed up as follows:

$$[W] = [c] = weight;\ [H] = length;\ [a] = \frac{weight}{length}$$

While the dimensions are set by the equation, the units of these quantities can vary. Weight can be expressed in pounds, kilograms, or stones, and length can be represented in inches, meters, or light years.

The dimensions of current are defined to be the amount of charge moving per unit of time, and the dimensions of voltage are energy per unit of charge. This allows us to find the dimensions of resistance by the following basic algebra:

$$[V] = \frac{energy}{charge} = \frac{[I]}{[R]} = \frac{charge/time}{[R]} \implies [R] = \frac{charge^2}{energy * time}$$

Electric potential is measured in volts, and current in amperes. The standard unit of resistance is the Ohm, which is defined as one volt per ampere. But regardless of the choice of units, the dimensions of these quantities remains.

A quantity may be made *dimensionless* by expressing it in terms of particular *scale*. For instance, we can express the height of a person as a fraction of the mean height of the population. A tall person will have height expressed as a number greater than 1, and a short one will have height less than 1. Note that this dimensionless height has no units - they have been divided out by scaling the height by the mean height. In fact, the word dimensionless is somewhat misleading: while such quantities have no scale in the context of the algebraic relationship, a quantity retains its physical significance after rescaling: height expressed as a fraction of some chosen length still represents height. Nevertheless, the accepted term in dimensionless quantity, and we will stick with this convention. Later in the book we will learn how to use the technique of rescaling to simplify and analyze dynamic models.

1 One variable in discrete time

All living things change over time, and this evolution can be quantitatively measured and analyzed. Mathematics makes use of equations to define models that change with time, known as *dynamical systems*. In this unit we will learn how to construct models that describe the time-dependent behavior of some measurable quantity in life sciences. Numerous fields of biology use such models, and in particular we will consider changes in population size, the progress of biochemical reactions, the spread of infectious disease, and the spikes of membrane potentials in neurons, as some of the main examples of biological dynamical systems.

Many processes in living things happen regularly, repeating with a fairly constant time period. One common example is the reproductive cycle in species that reproduce periodically, whether once a year, or once an hour, like certain bacteria that divide at a relatively constant rate under favorable conditions. Other periodic phenomena include circadian (daily) cycles in physiology, contractions of the heart muscle, and waves of neural activity. For these processes, theoretical biologists use models with *discrete time*, in which the time variable is restricted to the integers. For instance, it is natural to count the generations in whole numbers when modeling population growth.

This chapter is devoted to analyzing dynamical systems in which time is measured in discrete steps. We will build dynamic models, find their mathematical solutions, and then use Python to compute the solutions and plot them. In this chapter you will learn to:

- build discrete-time models of populations using rate paramaters
- define and verify mathematical solutions of these models
- use Python to compute and plot solutions

1.1 Building dynamic models

Let us construct our first models of biological systems! We will start by considering a population of some species, with the goal of tracking its growth or decay over time. The variable of interest is the number of individuals in the population, which we will call N. This is called the dependent variable, since its value changes depending on time; it would make no sense to say that time changes depending on the population size. Throughout the study of dynamical systems, we will denote the independent variable of time by t. To denote the population size at time t, we can write $N(t)$ but sometimes use N_t.

1.1.1 static population

In order to describe the dynamics, we need to write down a rule for how the population changes. Consider the simplest case, in which the population stays the same for all time. (Maybe it is a pile of rocks?) Then the following equation describes this situation:

$$N(t+1) = N(t)$$

This equation mandates that the population at the next time step be the same as at the present time t. This type of equation is generally called a *difference equation*, because it can be written as a difference between the values at the two different times:

$$N(t+1) - N(t) = 0$$

This version of the model illustrates that a difference equation at its core describes the *increments* of N from one time step to the next. In this case, the increments are always 0, which makes it plain that the population does not change from one time step to the next.

1.1.2 exponential population growth

Let us consider a more interesting situation: as a colony of dividing bacteria. such as *E. coli*, shown in {numref}`fig-cell-div`. We assume that each bacterial cell divides and produces two daughter cells at fixed intervals of time, and let us further suppose that bacteria never die. Essentially, we are assuming a population of immortal bacteria with clocks. This means that after each cell division the population size doubles. As before, we denote the number of cells in each generation by $N(t)$, and obtain the equation describing each successive generation:

$$N(t+1) = 2N(t)$$

It can also be written in the difference form, as above:

$$N(t+1) - N(t) = N(t)$$

The increment in population size is determined by the current population size, so the population in this model is forever growing. This type of behavior is termed *exponential growth* and we will see how to express the solution algebraically in the next section.

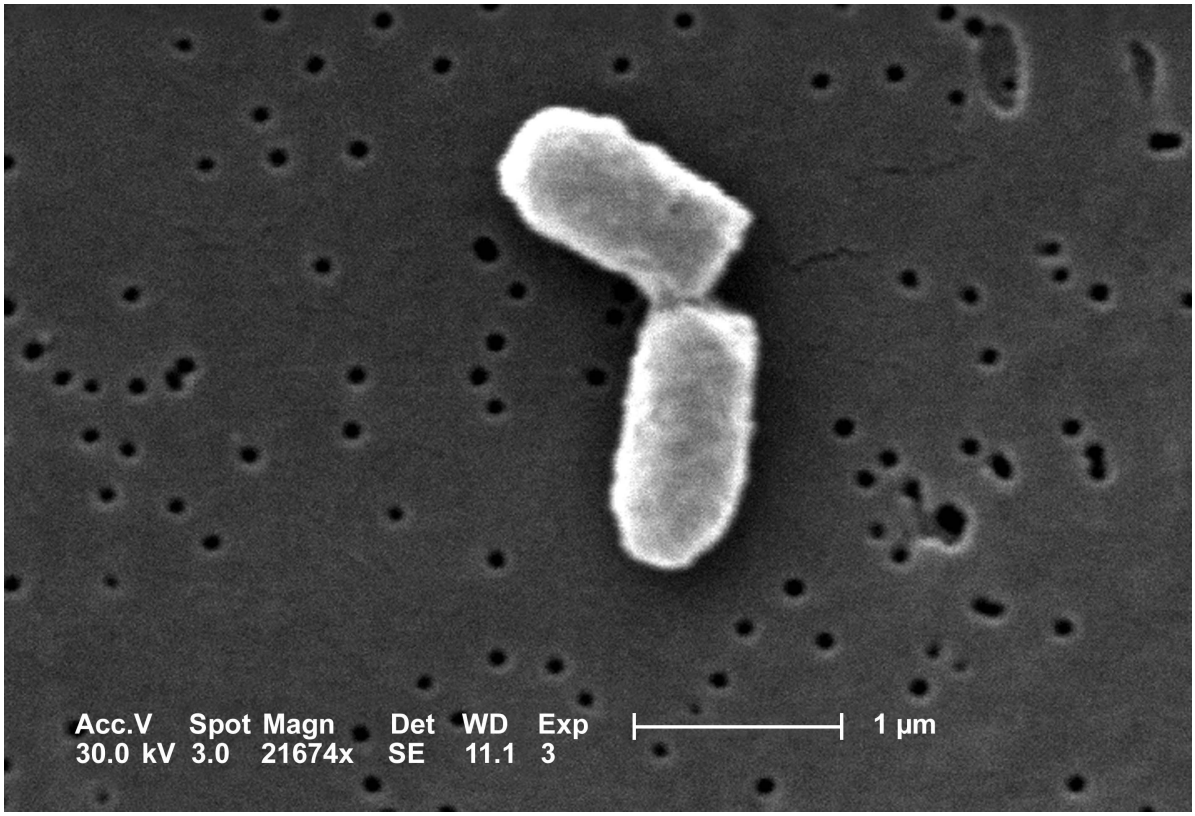


Figure 1.1: Scanning electron micrograph of a dividing *Escherichia coli* bacteria (image by Evangeline Sowers, Janice Haney Carr (CDC) in public domain via Wikimedia Commons)

1.1.3 example with birth and death

Suppose that a type of fish lives to reproduce only once after a period of maturation, after which the adults die. In this simple scenario, half of the population is female, a female always lays 1000 eggs, and of those, 1% survive to maturity and reproduce. Let us set up the model for the population growth of this idealized fish population. The general idea, as before, is to relate the population size at the next time step $N(t+1)$ to the population at the present time $N(t)$.

Let us tabulate both the increases and the decreases in the population size. We have $N(t)$ fish at the present time, but we know they all die after reproducing, so there is a decrease of $N(t)$ in the population. Since half of the population is female, the number of new offspring produced by $N(t)$ fish is $500N(t)$. Of those, only 1% survive to maturity (the next time step), and the other 99% ($495N(t)$) die. We can add all the terms together to obtain the following difference equation:

$$N(t+1) = N(t) - N(t) + 500N(t) - 495N(t) = 5N(t)$$

The number 500 in the expression is the *birth rate* of the population per individual, and the negative terms add up to the *death rate* of 496 per individual. We can re-write the equation in difference form:

$$N(t+1) - N(t) = 4N(t)$$

This expression again generates growth in the population, because the birth rate outweighs the death rate. (**allman_mathematical_2003?**)

1.1.4 dimensions of birth and death rates

What distinguishes a mathematical model from a mathematical equation is that the quantities involved have a real-world meaning. Each quantity represents a measurement, and associated with each one are the *units* of measurement, which are familiar from science courses. In addition to units, each variable and parameter has a meaning, which is called the *dimension* of the quantity. For example, any measurement of length or distance has the same dimension, although the units may vary. The value of a quantity depends on the units of measurement, but its essential dimensionality does not. One can convert a measurement in meters to that in light-years or cubits, but one cannot convert a measurement in number of sheep to seconds - that conversion has no meaning.

Thus leads us to the fundamental rule of mathematical modeling: **terms that are added or subtracted must have the same dimension**. This gives mathematical modelers a useful tool called *dimensional analysis*, which involves replacing the quantities in an equation with

their dimensions. This serves as a check that all dimensions match, as well as allowing to deduce the dimensions of any parameters for which the dimension was not specified.

In the case of population models, the birth and death rates measure the number of individuals that are born (or die) within a reproductive cycle for every individual at the present time. Their dimensions must be such that the terms in the equation all match:

$$[N(t+1) - N(t)] = [population] = [r][N(t)] = [r] * [population]$$

This implies that $[r]$ is algebraically dimensionless. However, the meaning of r is the rate of change of population over one (generation) time step. r is the birth or death rate of the population *per generation*, which is what makes is dimensionless. If the length of the generation were to change, but the reproduction and death per generation remain the same, then the paramter r would be the same, because it had been *rescaled* by the length of the generation. If they were to be reported in *absolute* units (e.g. individuals per year) then the rate would be different.

1.1.5 general demographic model

We will now write a general difference equation for any population with constant birth and death rates. This will allow us to substitute arbitrary values of the birth and death rates to model different biological situations. Suppose that a population has the birth rate of b per individual, and the death rate d per individual. Then the general model of the population size is:

$$N(t+1) = (1 + b - d)N(t)$$

(lin-pop)

The general equation also allows us to check the dimensions of birth and death rates, especially as written in the incremental form: $N(t+1) - N(t) = (b-d)N(t)$. The change in population rate over one reproductive cycle is given by the current population size multiplied by the difference of birth and death rates, which as we saw are algebraically dimensionless. The right hand side of the equation has the dimensions of population size, matching the difference on the left hand side. (**edelstein-keshet_mathematical_2005?**)

1.2 Solutions of linear difference models

We saw in the last section that we can write down equations to describe, step by step, how a variable changes over time. Let us define what the terminology of these equations:

Definition

An equation to describe a variable (e.g. N) that changes over discrete time steps described by the integer variable t is called a *difference equation* or a *discrete-time dynamic model*. These equations can be written in two ways, either in *recurrent form*:

$$N(t+1) = f(N(t))$$

(recur-eq)
or in *increment form*:

$$N(t+1) - N(t) = g(N(t))$$

(recur-eq)

1.2.1 simple linear difference models

Having set up the difference equation models, we would naturally like to solve them to find out how the dependent variable, such as population size, varies over time. A solution may be *analytic*, meaning that it can be written as a formula, or *numeric*, in which case it is generated by a computer in the form of a sequence of values of the dependent variable over a period of time. In this section, we will find some simple analytic solutions and learn to analyze the behavior of difference equations which we cannot solve exactly.

Definition

A function $N(t)$ is a *solution* of a difference equation $N(t+1) = f(N(t))$ if it satisfies that equation for all values of time t.

For instance, let us take our first model of the static population, $N(t+1) = N(t)$. Any constant function is a solution, for example, $N(t) = 0$, or $N(t) = 10$. There are actually as many solutions as there are numbers, that is, infinitely many! In order to specify exactly what happens in the model, we need to specify the size of the population at some point, usually, at the "beginning of time", $t = 0$. This is called the *initial condition* for the model, and for a well-behaved difference equation it is enough to determine a unique solution. For the static model, specifying the initial condition is the same as specifying the population size for all time.

Now let us look at the general model of population growth with constant birth and death rates. We saw in equation {eq}`lin-pop` above that these can be written in the form $N(t+1) = (1+b-d)N(t)$. To simplify, let us combine the numbers into one growth parameter $r = 1+b-d$, and write down the general equation for population growth with constant growth rate:

$$N(t+1) = rN(t)$$

(lin-pop-r)

To find the solution, consider a specific example, where we start with the initial population size $N_0 = 1$, and the growth rate $r = 2$. The sequence of population sizes is: 1, 2, 4, 8, 16, etc. This is described by the formula $N(t) = 2^t$.

In the general case, each time step the solution is multiplied by r, so the solution has the same exponential form. The initial condition N_0 is a multiplicative constant in the solution, and one can verify that when $t = 0$, the solution matches the initial value:

$$N(t) = r^t N_0$$

(lin-pop-sol)

I would like the reader to pause and consider this remarkable formula. No matter what the birth and death parameters are selected, this solution predicts the population size at any point in time t.

In order to verify that the formula for $N(t)$ is actually a solution in the meaning of definition, we need to check that it actually satisfies the difference equation for all t, not just a few time steps. This can be done algebraically by plugging in $N(t+1)$ into the left side of the dynamic model and $N(t)$ into the right side and checking whether they match. For $N(t)$ given by equation {eq}`lin-pop-sol`, $N(t+1) = r^{t+1}N_0$, and thus the dynamic model becomes:

$$r^{t+1}N_0 = r \times r^t N_0$$

Since the two sides match, this means the solution is correct.

The solutions in equation {eq}`lin-pop-sol` are exponential functions, which have a limited menu of behaviors, depending on the value of r. If $r > 1$, multiplication by r increases the size of the population, so the solution $N(t)$ will grow (see {numref}`fig-exp-growth`. If $r < 1$, multiplication by r decreases the size of the population, so the solution $N(t)$ will decay (see {numref}`fig-exp-decay`). Finally, if $r = 1$, multiplication by r leaves the population size unchanged, like in the pile of rocks model. Here is the complete classification of the behavior of population models with constant birth and death rates (assuming $r > 0$):

Classification of solutions of linear dynamic models

For a difference equation $N(t+1) = rN(t)$, solutions can behave in one of three ways:

- $|r| > 1$: $N(t)$ grows without bound
- $|r| < 1$: $N(t)$ decays to 0

- $|r| = 1$: the absolute value of $N(t)$ remains constant

Se examples of graphs of solutions of such equations with r greater than 1 in {numref}`fig-exp-growth` and solutions for r less than 1 in {numref}`fig-exp-decay`.

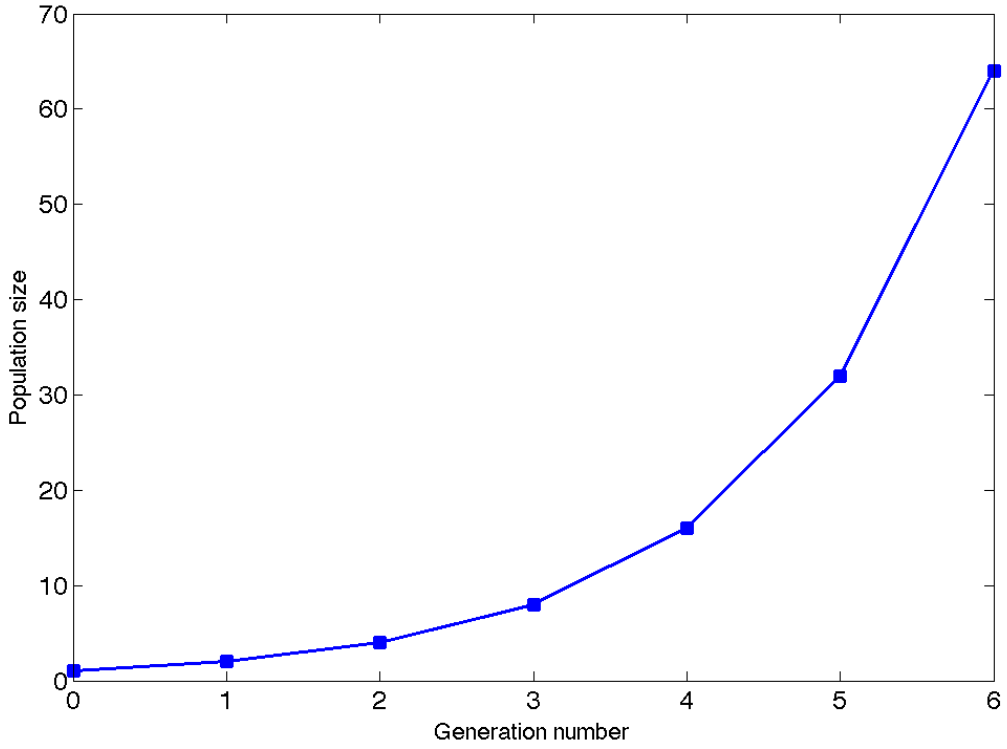


Figure 1.2: Growth of a population that doubles every generation over 6 generations

1.2.2 linear difference models with a constant term

Now let us consider a dynamic model that combines two different rates: a proportional rate (rN) and a constant rate which does not depend on the value of the variable N. We can write such a generic model as follows:

$$N(t+1) = rN(t) + a$$

The right-hand-side of this equation is a linear function of N, so this is a linear difference equation with a constant term. What function $N(t)$ satisfies it? One can quickly check that that the same solution $N(t) = r^t N_0$ does not work because of the pesky constant term a:

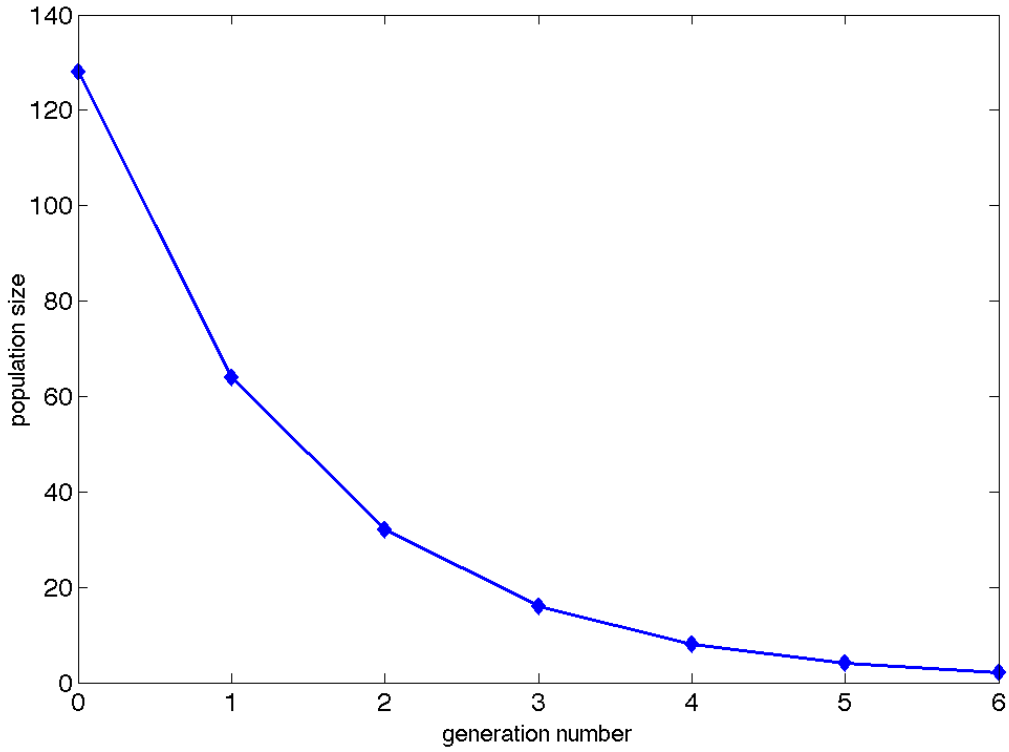


Figure 1.3: Decay of a population in which half the individuals die every time step over 6 generations

$$r^{t+1}N_0 \neq r \times r^t N_0 + a$$

To solve it, we need to try a different form: specifically, an exponential with an added constant. The exponential can be reasonably surmised to have base r as before, and then leave the two constants as unknown: $N(t) = c_1 r^t + c_2$. To figure out whether this is a solution, plug it into the linear difference equation above and check whether a choice of constants can make the two sides agree:

$$N(t+1) = c_1 r^{t+1} + c_2 = rN(t) + a = rc_1 r^t + rc_2 + a$$

This equation has the same term $c_1 r^{t+1}$ on both sides, so they can be subtracted out. The remaining equation involves only c_2, and its solution is $c_2 = a/(1-r)$. Therefore, the general solution of this linear difference equation is the following expreis which is determined from the initial value by plugging $t = 0$ and solving for c.

$$N(t) = cr^t + \frac{a}{1-r}$$

Example. Take the difference equation $N(t+1) = 0.5N(t) + 40$ with initial value $N(0) = 100$. The solution, according to our formula is $N(t) = c0.5^t + 80$. At $N(0) = 100 = c + 80$, so $c = 20$. Then the compete solution is $N(t) = 20 * 0.5^t + 80$. To check that this actually works, plug this solution back into the difference equation:

$$N(t+1) = 20 \times 0.5^{t+1} + 80 = 0.5 \times (20 \times 0.5^t + 80) + 40 = 20 \times 0.5^{t+1} + 80$$

The equation is satisfied and therefore the solution is correct.

2 Plotting in Python

You can find an introduction to the plotting library matplotlib here.

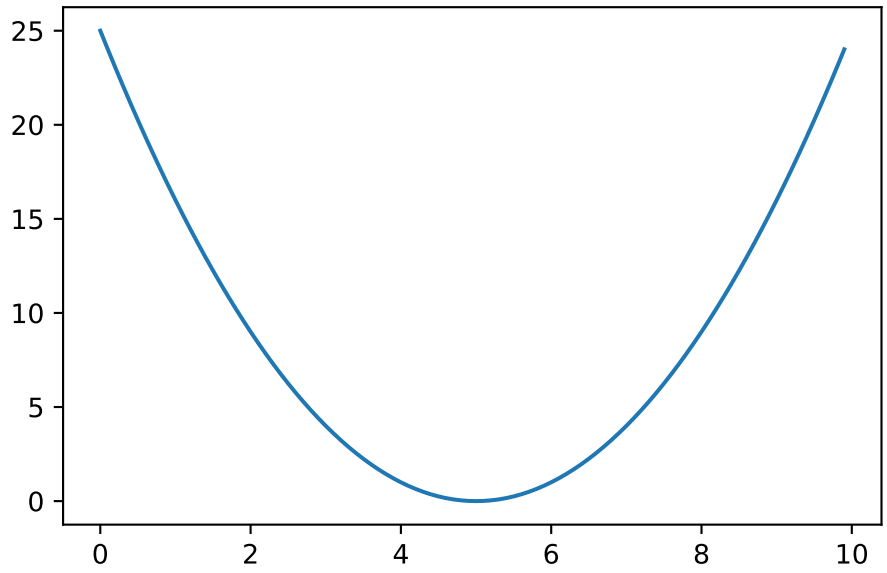
```
# Import packages
import numpy as np # package for work with arrays and matrices
import matplotlib.pyplot as plt # package with plotting capabilities
```

2.0.1 arrays and basic plotting

Here is an example of performing calculations with arrays (vectors) of values and plotting the results:

```
x = np.arange(0,10,0.1) # create an array of numbers between 0 and 10 with step 0.1
print(np.shape(x))
y = (x-5)**2 # do calculations on all the array values, call it y
plt.plot(x,y) # plot x vs y
plt.show()
```

```
(100,)
```



A two dimensional array (matrix) can be defined as follows, and the function np.shape prints out the number of rows and columns in the matrix:

```
x = np.array([[1,2,3],[4,5,6]])
print(np.shape(x))
print(x)
```

```
(2, 3)
[[1 2 3]
 [4 5 6]]
```

An example of concatenating a text string together with a numeric variable, which can then be used for labels or legends in plots:

```
prob = 0.5
string1 = 'The value of prob is ' + str(prob)
print(string1)
```

```
The value of prob is 0.5
```

2.1 Numeric solutions of discrete models

Difference equations, as we saw above, can be written in the form of $x_{t+1} = f(x_t)$. At every step, the model takes the current value of the dependent variable x_t, feeds it into the function $f(x)$, and takes the output as the next value x_{t+1}. The same process repeats every iteration, which is why difference equations written in this form are called *iterated maps*.

Computers are naturally suited for precise, repetitive operations. In our first example of a computational algorithm, we will iterate a given function to produce a sequence of values of the dependent variable x. We only need two things: to specify a computer function $f(x)$, which returns the value of the iterated map for any input value x, and the initial value x_0. Then it is a matter of repeating the operation of evaluating $f(x_t)$ and storing it as the next value x_{t+1}. Below is the pseudocode for the algorithm. Note that I will use arrows to indicated variable assignment, square brackets [] for indexing of vector, and start indexing at 0, consistent with python convention.

Iterative solution of difference equations:

- define the iterated map function $F(x)$
- set N to be the number of iterations (time steps)
- set the initial condition x_0
- initialize array x with initial value x_0
- for i from 0 to $N-1$
  - $x[i+1] \leftarrow F(x[i])$

The resulting sequence of values $x_0, x_1, x_2, ..., x_N$ is called a *numeric solution* of the given difference equation. It has two disadvantages compared to an analytic solution: first, the solution can only be obtained for a specific initial value and number of iterations, and second, any computer simulation inevitably introduces some errors, for instance from round-off. In practice, however, most complex dynamical systems have to solved numerically, as analytical solutions are difficult or impossible to find.

2.1.1 using for loops for iterative solutions of dynamic models

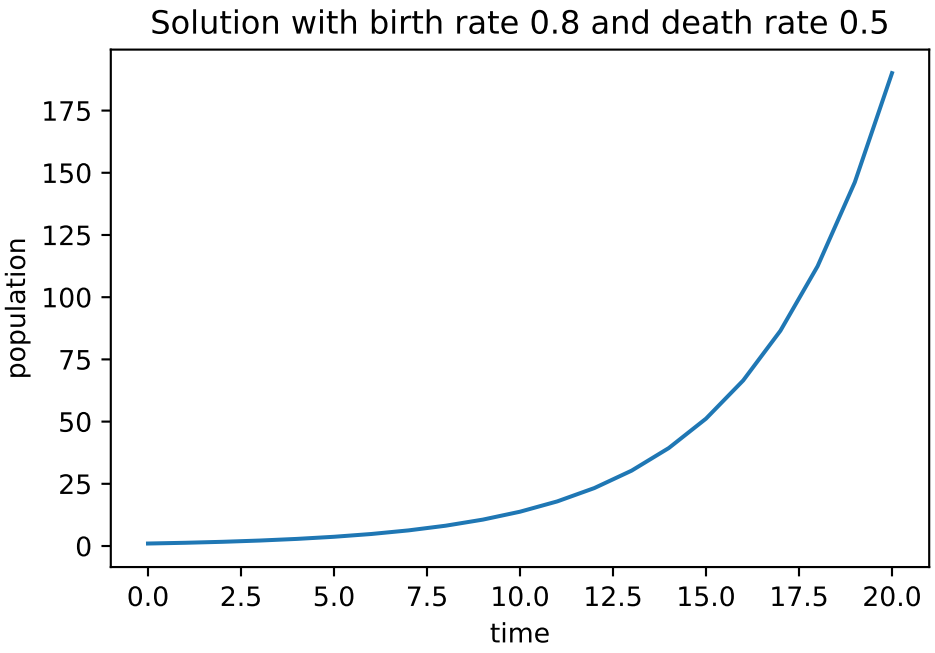
Here is a generic linear demographic model

$$x(t+1) = x(t) + bx(t) - dx(t) = rx(t)$$

Example of a script for producing a numeric solution of a discrete time dynamic model:

```
numsteps = 20 # number of iterations
birth = 0.8 # birth rate
death =  0.5 # death rate
pop = np.zeros(numsteps+1) # initialize solution array
pop[0] = 1 # initial value
t = np.arange(numsteps+1) # initialze time vector
print(t)

for i in range(numsteps):
     pop[i+1] = pop[i] + birth*pop[i] - death*pop[i]# linear demographic model

plt.plot(t, pop) # plot solution
plt.xlabel('time')
plt.ylabel('population')
title = 'Solution with birth rate ' + str(birth) + ' and death rate ' + str(death)
plt.title(title)
plt.show()
```

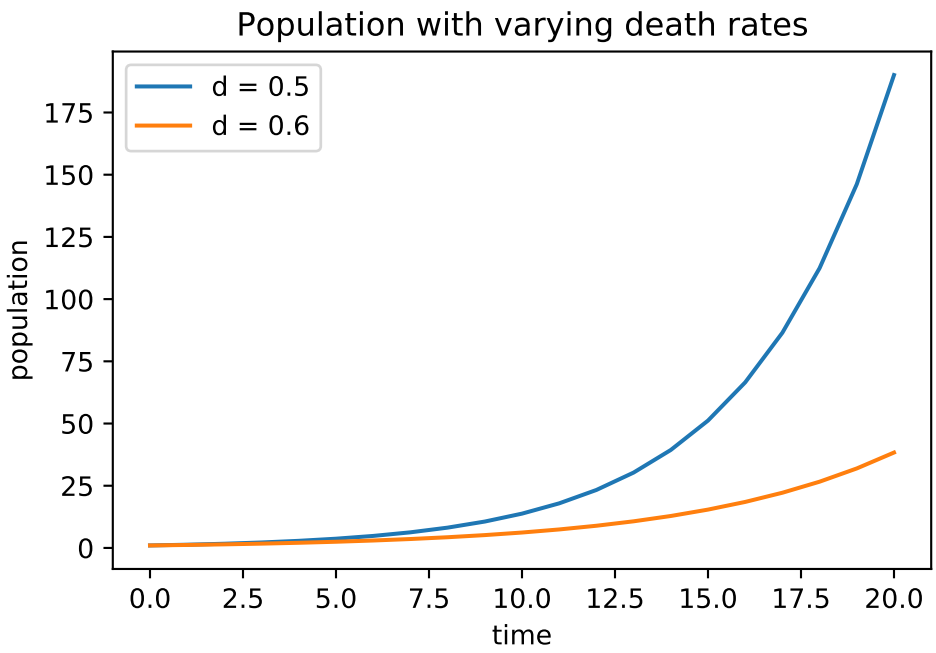
```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]
```



2.1.2 plotting multiple curves with a legend

Multiple solution plots can be overlayed on the same figure, as long as the `plt.show()` is only used once in the end. For multiple graphs it's best to use multiple colors and a legend to label different curves, using the option label in the `plt.plot` function and adding the function `plt.legend()` before producing the figure. Here's an example with solutions of the demographic model with different death rates:

```
numsteps = 20 # number of iterations
birth = 0.8 # birth rate
death =  0.5 # death rate
pop = np.zeros(numsteps+1) # initialize solution array
pop[0] = 1 # initial value
t = np.arange(numsteps+1) # initialze time vector
for i in range(numsteps):
     pop[i+1] = pop[i] + birth*pop[i] - death*pop[i]# linear demographic model

plt.plot(t, pop, label = 'd = '+str(death)) # plot solution
plt.xlabel('time')
plt.ylabel('population')
death =  0.6 # death rate
pop = np.zeros(numsteps+1) # initialize solution array
pop[0] = 1 # initial value
t = np.arange(numsteps+1) # initialze time vector
for i in range(numsteps):
     pop[i+1] = pop[i] + birth*pop[i] - death*pop[i]# linear demographic model
plt.plot(t, pop, label = 'd = '+str(death)) # plot solution

title = 'Population with varying death rates'
plt.title(title)
plt.legend()
plt.show()
```



2.1.3 random number generators

Numpy provides a variety of random number generators, and we'll use these functions in the course for many purposes. Here is an example of producing arrays of random normally distributed numbers. The function requires inputs of the mean, the standard deviation, and the number of random values (or size of the array):

```
mu = 5

sigma = 0.5

num = 30

norm_sample = np.random.normal(mu, sigma, num)

print(norm_sample)

print("The mean of the sample is " + str(np.mean(norm_sample)))

print("The standard deviation of the sample is " + str(np.std(norm_sample)))
```

```
[4.77100734 4.52994758 4.22206221 5.7529619  5.02028133 5.48352634
```

```
 5.45871784 5.54194591 4.79022161 4.93164061 5.04779049 4.6586361
 5.35748924 4.83471008 5.15979114 4.63568419 4.8095939  6.14423211
 4.69020281 4.92948335 5.41255833 4.86783332 4.61590118 5.67141568
 4.62240577 4.27958549 5.52725606 4.51262226 6.46019166 4.95665629]
The mean of the sample is 5.056545070795578
The standard deviation of the sample is 0.5200139194235976
```

3 Nonlinear discrete-time dynamic models

In this chapter we will analyze nonlinear discrete dynamical systems. Their solutions, as those of nonlinear ODEs, exhibit much more interesting behaviors than the exponential solutions of linear equations, and are typically not solvable analytically. There may be multiple fixed points, some stable and others unstable, and even crazier behaviors are possible that are not permitted in smooth-flowing ODEs. Specifically, we will see solutions that oscillate, and those that behave without any pattern at all, that are called chaotic. You will learn to do the following in this chapter:

- build the logistic population model
- find equilibrium values of nonlinear discrete-time models
- analyze the stability of equilibria based on the graph of the updating function
- write down stability conditions analytically
- use Python to make cobweb plots
- understand the term chaos

3.1 Logistic population model

Linear population growth models assume that the per capita birth and death rates are constant, that is, they stay the same regardless of population size. The solutions for these models either grow or decay exponentially, but in reality, populations cannot grow without bounds. It is generally true that the larger a population grows, the more scarce the resources, and survival becomes more difficult. For larger populations, this could lead to higher death rates, or lower birth rates, or both.

To incorporate this effect into a quantitative model we will assume there are separate birth and death rates, and that the birth rate declines as the population grows, while the death rate increases:

$$b = b_1 - b_2 N(t);\ d = d_1 + d_2 N(t)$$

To model the rate of change of the population, we need to multiply the rates b and d by the population size N, since each individual can reproduce or die. Also, since the death rate d decreases the population, we need to put a negative sign on it. The resulting model is:

$$N(t+1) - N(t) = (b-d)N(t) = [(b_1 - d_1) - (b_2 + d_2)N(t)]N(t)$$

A simpler way of writing this equation is to let $r = 1 + b_1 - d_1$ and $K = b_2 + d_2$, leading to the following iterated map:

$$N(t+1) = (r - KN(t))N(t)$$

(discr-log)

This is called the *logistic model* of population growth. As you see, it has two different parameters, r and K. If $K = 0$, the equation reduces to the old linear population model. Intuitively, K is the parameter describing the effect of increasing population on the population growth rate. Let us analyze the dimensions of the two parameters, by writing down the dimensions of the variables of the difference equation. The dimensional equation is:

$$N(t+1) = [population] = [r - KN(t)]N(t) == ([r] - [K] \times [population]) \times [population]$$

Matching the dimensions on the two sides of the equation leads us to conclude that the dimensions of r and k are different:

$$[r] = 1;\ [K] = \frac{1}{[population]}$$

The difference equation for the logistic model is *nonlinear*, because it includes a second power of the dependent variable. In general, it is difficult to solve nonlinear equations, but we can still say a lot about this model's behavior without knowing its explicit solution.

3.2 Qualitative analysis of difference equations

3.2.1 fixed points or equilibria

We have seen that the solutions of difference equations depend on the initial value of the dependent variable. In the examples we have seen so far, the long-term behavior of the solution does not depend dramatically on the initial condition. In more complex systems that we will encounter, there are special values of the dependent variable for which the dynamical system is constant, like in the pile of rocks model.

Definition

For a difference equation in recurrent form $x(t+1) = f(X(t))$, a point x^* which satisfies $f(x^*) = x^*$ is called a *fixed point* or *equilibrium*. If the initial condition is a fixed point, $x_0 = x^*$, the solution will stay at the same value for all time, $x(t) = x^*$.

The reason these special points are also known as equilibria is due to the precise balance between growth and decay that is mandated at a fixed point. In terms of population modeling, at an equilibrium the birth rates and the death rates are equal. Speaking analytically, in order to find the fixed points of a difference equation, one must solve the equation $f(x^*) = x^*$. It may have none, or one, or many solutions.

Example. The linear population models which we analyzed in the previous sections have the mathematical form $N(t+1) = rN(t)$ (where r can be any real number). Then the only fixed point of those models is $N^* = 0$, that is, a population with no individuals. If there are any individuals present, we know that the population will grow to infinity if $|r| > 1$, and decay to 0 if $|r| < 1$. This is true even for the smallest population size, as long as it is not exactly zero.

Example. Let us go back to the example of a linear difference equation with a constant term. The equation is $ N(t+1) = -0.5N(t) +10$, and we saw that the numerical solutions all converged to the same value, regardless of the initial value. Let us find the equilibrium value of this model using the definition:

$$N^* = -0.5N^* + 10 \Rightarrow 1.5N^* = 10 \Rightarrow N^* = 10/1.5 = 20/3$$

If the initial value is equal to the equilibrium, $N(0) = 20/3$, then the solution will remain constant for all time, since the next value $N(t+1) = -0.5 * 20/3 + 10 = 20/3$ remains the same.

Example: discrete logistic model. Let us use the simplified version of the logistic equation $N(t+1) = r(1 - N(t))N(t)$ and set the right-hand side function equal to the variable N to find the fixed points N^*:

$$r(1 - N^*)N^* = N^*$$

There are two solutions to this equation, $N^* = 0$ and $N^* = (r-1)/r$. These are the fixed points or the equilibrium population sizes for the model, the first being the obvious case when the population is extinct. The second equilibrium is more interesting, as it describes the *carrying capacity* of a population in a particular environment. If the initial value is equal to either of the two fixed points, the solution will remain at that same value for all time. But what happens to solutions which do not start a fixed point? Do they converge to a fixed point, and if so, to which one?

3.2.2 stability criteria for fixed points

What happens to the solution of a dynamical system if the initial condition is very close to an equilibrium, but not precisely at it? Put another way, what happens if the equilibrium is *perturbed*? To answer the question, we will no longer confine ourselves to the integers, to be interpreted as population sizes. We will instead consider, abstractly, what happens if the smallest perturbation is added to a fixed point. Will the solution tend to return to the fixed point or tend to move away from it? The answer to this question is formalized in the following definition (**strogatz_nonlinear_2001?**):

> **i** Definition
>
> For a difference equation $x(t+1) = f(x(t))$, a fixed point x^* is *stable* if for a sufficiently small number ϵ, the solution $x(t)$ with the initial condition $x_0 = x^* + \epsilon$ approaches the fixed point x^* as $t \to \infty$. If the solution $x(t)$ does not approach x^* for any nonzero ϵ, the fixed point is called *unstable*.

The notion of stability is central to the study of dynamical systems. Typically, models more complex than those we have seen cannot be solved analytically. Finding the fixed points and determining their stability can help us understand the general behavior of solutions without writing them down. For instance, we know that solutions never approach an unstable fixed point, whereas for a stable fixed point the solutions will tend to it, from some range of initial conditions.

There is a mathematical test to determine the stability of a fixed point. From standard calculus comes the Taylor expansion, which approximates the value of a function near a given point. Take a general difference equation written in terms of some function $x(t+1) = f(x(t))$. Let us define the *deviation* from the fixed point x^* at time t to be $\epsilon(t) = x_t - x^*$. Then we can use the linear (first-order) Taylor approximation at the fixed point and write down the following expression:

$$x(t+1) = f(x^*) + \epsilon(t) f'(x^*) + ...$$

The ellipsis means that the expression is approximate, with terms of order $\epsilon(t)^2$ and higher swept under the rug. Since we take $\epsilon(t)$ to be small, those terms are very small and can be neglected. Since x^* is a fixed point, $f(x^*) = x^*$. Thus, we can write the following difference equation to describe the behavior of the deviation from the fixed point X^*:

$$x(t+1) - x^* = \epsilon(t+1) = \epsilon(t) f'(x^*)$$

We see that we started out with a general function defining the difference equation and transformed it into a linear equation for the deviation $\epsilon(t)$. Note that the multiplicative constant

here is the derivative of the function at the fixed point: $f'(x^*)$. This is called the *linearization* approach, which is an approximation of a dynamical system near a fixed point with a linear equation for the small perturbation.

We found the solution to simple linear equations, which we can use describe the behavior of the perturbation to the fixed point. The behavior **depends on the value of the derivative of the updating function** $f'(X^*)$:

Important Fact

For a difference equation $x(t+1) = f(x(t))$, a fixed point x^* can be classified as follows:

- $|f'(x^*)| > 1$: the deviation $\epsilon(t)$ grows, and the solution moves away from the fixed point; fixed point is *unstable*
- $|f'(x^*)| < 1$: the deviation $\epsilon(t)$ decays, and the solution approaches the fixed point; fixed point is *stable*
- $|f'(x^*)| = 1$: the fixed point may be stable or unstable, and more information is needed

We now know how to determine the stability of a fixed point, so let us apply this method to some examples.

Example: linear difference equations. Let us analyze the stability of the fixed point of a linear difference equation, e.g. $N(t+1) = -0.5N(t) + 10$. The derivative of the updating function is equal to -0.5. Because it is less than 1 in absolute value, the fixed point is stable, so solutions converge to this equilibrium. We can state more generally that any linear difference equation of the form $N(t+1) = aN(t)+b$ has one fixed point, which is equal to $N^* = b/(1-a)$. This fixed point is stable if $|a| < 1$ and unstable if $|a| > 1$.

Example: discrete logistic model. In the last subsection we found the fixed points of the simplified logistic model. To determine what happens to the solution, we need to determine the stability of both equilibria. Since the stability of fixed points is determined by the derivative of the defining function at the fixed points, we compute the derivative of $f(N) = rN - rN^2$ to be $f'(N) = r - 2rN$, and evaluate it at the two fixed points $N^* = 0$ and $N^* = (r-1)/r$:

$$f'(0) = r; \quad f'((r-1)/r) = r - 2(r-1) = 2 - r$$

Because the intrinsic death rate cannot be greater than the birth rate, we know that $r > 0$. Therefore, we have the following *stability conditions* for the two fixed points:

- the fixed point $N^* = 0$ is stable for $r < 1$, and unstable for $r > 1$;
- the fixed point $N^* = (r-1)/r$ is stable for $1 < r < 3$, and unstable otherwise.

3.3 Analysis of logistic population model

3.3.1 rescaling the logistic model

First, let us do one more modification of the model, by taking the parameter r as the common multiple:

$$N_{t+1} = r(1 - \frac{K}{r}N_t)N_t$$

As we saw, the parameter K has dimension of inverse population size, and that the parameter r is dimensionless. We can now use rescaling of the variable N to simplify the logistic model. The goal is to reduce the number of parameters, by canceling some, and bringing the rest into one place, where they can be combined into a *dimensionless group*. Here is how this is accomplished for this model:

1. Pick a number of the same dimension as the variable, called the scale, and divide the variable by it. In this case, let the scale for population be r/K, so then the new variable is

$$\tilde{N} = \frac{NK}{r} \Longrightarrow N = \frac{\tilde{N}r}{K}$$

Since the parameter K has dimension of inverse population size, NK is in the dimensionless variable $\tilde{N}$.

2. Substitute $\tilde{N}/K$ for N in the equation:

$$\frac{\tilde{N}_{t+1}r}{K} = r\left(1 - \frac{K\tilde{N}_t r}{rK}\right)\frac{\tilde{N}_t r}{K}$$

3. Canceling all the parameters on both sides, we just have the dimensionless growth rate r, as our only parameter:

$$\tilde{N}_{t+1} = r(1 - \tilde{N}_t)\tilde{N}_t$$

On the surface, we merely used algebraic trickery to simplify the equation, but the result is actually rather deep. By changing the dimension of measurement of the population from individuals (N) to the dimensionless fraction of the carrying capacity ($\tilde{N}$) we found that there is only one parameter r that governs the behavior of this model. We will see in the next two section that varying this parameter leads to dramatic changes in the dynamics of the model population. (**edelstein-keshet_mathematical_2005?**)

3.3.2 fixed point analysis

The first step for qualitative analysis of a nonlinear model is to find the fixed points. We use the dimensionless version of the logistic equation, and the right-hand side function equal to the value of the special values N^* (fixed points):

$$r(1 - N^*)N^* = N^*$$

There are two solutions to this equation, $N^* = 0$ and $N^* = (r - 1)/r$. These are the fixed points or the equilibrium population sizes for the model, the first being the obvious case when the population is extinct. The second equilibrium is more interesting, as it describes the *carrying capacity* of a population in a particular environment. To determine what happens to the solution, we need to evaluate the stability of both equilibria.

We have seen in the analytical section that the stability of fixed points is determined by the derivative of the defining function at the fixed points. The derivative of $f(N) = rN - rN^2$ is $f'(N) = r - 2rN$, and we evaluate it at the two fixed points:

$$f'(0) = r; \quad f'((r-1)/r) = r - 2(r-1) = 2 - r$$

Because the intrinsic death rate cannot be greater than the birth rate, we know that $r > 0$. Therefore, we have the following stability conditions for the two fixed points:

- the fixed point $N^* = 0$ is stable for $r < 1$, and unstable for $r > 1$;
- the fixed point $N^* = (r - 1)/r$ is stable for $1 < r < 3$, and unstable otherwise.

We can plot the solution for the population size of the logistic model population over time. We see that, depending on the value of the parameter r (but not on k), the behavior is dramatically different:

Case 1: $r < 1$**.** The fixed point at $N^* = 0$ is stable and the fixed point is unstable $N^* = (r - 1)/r$. The solution tends to 0, or extinction, regardless of the initial condition, which is illustrated in figure [fig:sol_logistic_2] for $r = 0.8$.

Case 2: $1 < r < 3$**.** The extinction fixed point $N^* = 0$ is unstable, but the carrying capacity fixed point $N^* = (r - 1)/r$ is stable. We can conclude that the solution will approach the carrying capacity for most initial conditions. This was shown in figure [fig:sol_logistic_1] for $r = 1.5$ and is illustrated in figure [fig:sol_logistic_3] for $r = 2.8$. Notice that although the solution approaches the carrying capacity equilibrium in both cases, when $r > 2$, the solution oscillates while converging to its asymptotic value, foreshadowing the behavior when $r > 3$.

Case 3: $r > 3$**.** Strange things happen: there are no stable fixed points, so there is no value for the solution to approach. As we saw in the previous section, the solution can undergo so-called period two oscillations, which are shown in figure [fig:sol_logistic_4] with $r = 3.3$.

However, even stranger behavior is observed when the parameter r crosses the threshold of about 3.59. Figure [fig:sol_logistic_5] shows the behavior of the solution for $r = 3.6$, which is no longer periodic, and instead seems to bounce around without any discernible pattern. This dynamics is known as *chaos*.

4 Graphical analysis of difference equations

In addition to calculating numeric solutions, computers can be used to perform *graphical analysis* of discrete time models. A lot of information can be gleaned by plotting the graph of the updating function of an recurrent difference equation $x_{t+1} = f(x_t)$. Here is a summary of what we can learn from the graph of the function $f(x)$:

Information from graphs of updating functions

1. The location of the fixed points of the iterated map. Since the condition for a fixed point is $f(x) = x$, they can be found at the intersections of the graph of $y = f(x)$ and $y = x$ (the identity straight line).
2. The stability of fixed points. We learned that the derivative of $f(x)$ at a fixed point determines its stability. Graphically, this means that the slope of $f(x)$ at the point of intersection with $y = x$ can be used for this purpose; if it is steeper (in absolute value) than the straight line $y = x$, then the fixed point is unstable, but if its slope is less than one in absolute value, the equilibrium is stable.
3. Graphical iteration of the difference equation. The value of the function $f(x)$ gives the value of x at the next time step, and this fact can be used to produce a graph of successive values of the dependent variable: $x_0, x_1, x_2, ...$

Below we will demonstrate how to plot the updating function in Python and how to perform this analysis.

4.0.1 plot of the updating function

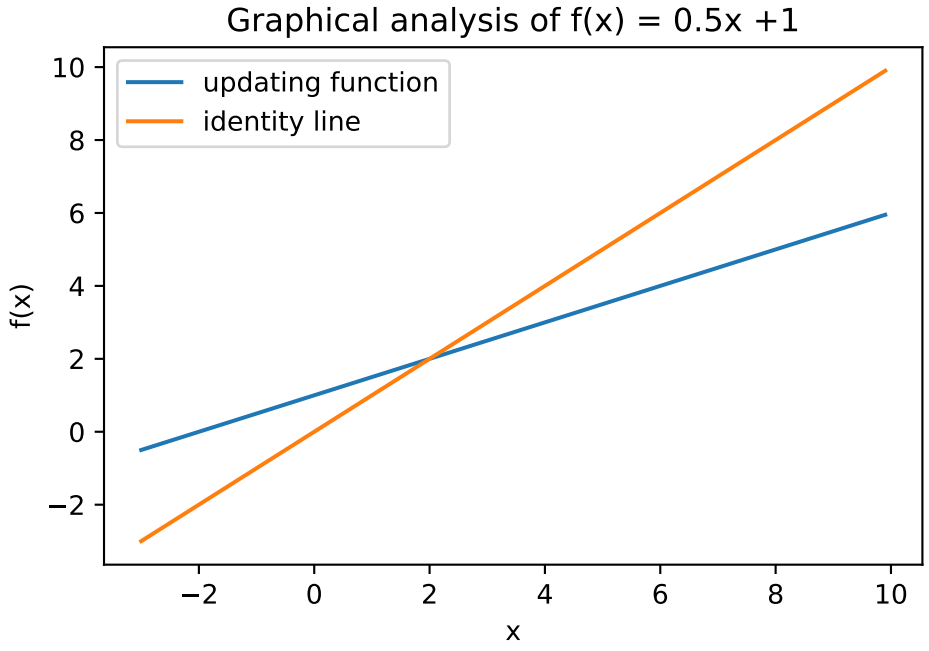
Let us consider a linear discrete-time model:

$$x_{t+1} = 5x_t - 10$$

Instead of iterating this equation starting from a particular initial value to produce a sequence of values, we will plot the function $f(x) = 0.5x - 10$ and use it to predict how solutions behave starting from *any* initial value. To do this, we will plot this function over a range of x values, along the the line $y = x$.

```
# Import packages
import numpy as np # package for work with arrays and matrices
import matplotlib.pyplot as plt # package with plotting capabilities
```

```
x = np.arange(-3,10,0.1) # range of x values
fx = 0.5*x + 1 # values of the updating function
plt.plot(x, fx, label = 'updating function') # plot the updating function
plt.plot(x, x, label = 'identity line')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Graphical analysis of f(x) = 0.5x +1')
plt.legend()
plt.show()
```



The plot show thats the updating function intersects the identity line at $x = 2$, which is the (only) fixed point of this dynamic model. But it is stable or unstable? And how will solutions behave?

4.0.2 cobweb plot

Let us exploit the idea in the third point for graphical analysis of an iterated map. Starting with some initial condition x_0, the value of x_1 is given by $f(x_0)$. To show this graphically, starting the point x_0 on the axis, draw a vertical line to $y = f(x_0)$. Next, draw a horizontal line to the graph of $y = x$. Since the y and x coordinates are equal, we now have the value of $x_1 = f(x_0)$ as the x coordinate. Then, repeat the process by drawing a vertical line to $y = f(x_1)$, and the a horizontal line $y = x$, etc. The resulting sequence of x coordinates is a quick way of assessing the dynamics of the iterated map. For instance, the values may converge to a fixed point, or grow to infinity, or bounce around without settling down. The resulting graph of alternating vertical and horizontal line segments is called a cobweb plot:

Cobweb plot pseudocode

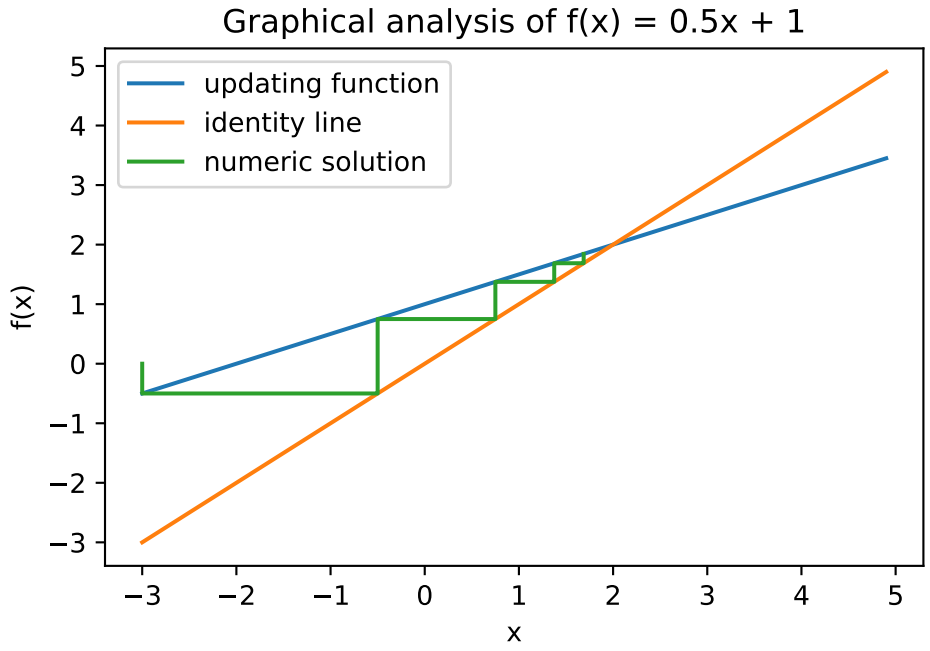
- define the updating function $f(x)$
- plot the graph of $f(x)$
- plot the identity line $y = x$
- set n to be the number of steps
- initialize an array x of length 2*n
- initialize an array y of length 2*n
- set $x[0]$ to the initial value
- set $y[0]$ to 0
- for n steps repeat (with i increasing by 2):
  - set $x[i+1] \leftarrow x[i]$
  - set $y[i+1] \leftarrow f(x[i])$
  - set $x[i+2] \leftarrow y[i+1]$
  - set $y[i+2] \leftarrow y[i+1]$
- plot the sequence of points (x, y) on the same plot

Here is an implementation of the cobweb plot for the same linear model as above.

```
x = np.arange(-3,5,0.1) # range of x values
fx = 0.5*x + 1 # values of the updating function
plt.plot(x, fx, label = 'updating function') # plot the updating function
plt.plot(x, x, label = 'identity line')

# the cobweb plot script
n = 5 # number of steps
x = np.zeros(n*2)
y = np.zeros(n*2)
x[0] = -3
```

```
print(x)
print(np.arange(0,2*n,2))
for i in np.arange(0,2*(n-1),2):
    x[i+1] = x[i] # keep the same x coordinate
    y[i+1] = 0.5*x[i] + 1 # the updating function
    x[i+2] = y[i+1] # move to the next x value
    y[i+2] = y[i+1] # keep the same y coordinate
x[2*n-1] = x[2*n-2] # finish the last half-iteration
y[2*n-1] = 0.5*x[2*n-2] + 1 # finish the last half-iteration

plt.plot(x,y, label = 'numeric solution')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Graphical analysis of f(x) = 0.5x + 1')
plt.legend()
plt.show()
```

```
[-3.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[0 2 4 6 8]
```



You can see that the solution, shown in green, start from initial value of -3 and steps upward

toward the fixed point at 2. You can experiment by trying a different initial value, say 10 (you will have to change the domain of the plotted function) but you should still observe convergence to the fixed point at 2.

This could be predicted from the analysis performed in the previous section: the slope of the updating function is 0.5 (everywhere, since it's a straight line) and since that is less than 1 in absolute value, the fixed point is stable, and solutions are expected to converge to it.

We now have at our disposal analytical, numerical, and graphical tools to analyze and predict the behavior of a dynamical system. In the next section we will use all three to analyze a more complex model of population growth.

4.1 Graphical analysis of the logistic model

As we saw, we can learn a lot about the behavior of a dynamical system from analyzing the graph of the defining function. Let us consider two quadratic functions for the logistic model: $f(N) = 2N(1 - N/2)$ and $f(N) = 4N(1 - N/4)$.

First, plotting the graphs of $y = f(N)$ and $y = N$, allows us to find the fixed points of the logistic model. Since it is a , we see that there are fixed points at $N = 0$ for both functions, and carrying capacity sizes at $N = 2$ and $N = 3$, respectively. The reader should check that this is in agreement with the analytic prediction of $N^* = (r - 1)/r$.

Second, we can obtain information about stability of the two fixed points by considering the slope of the curve $y = f(N)$ at the points where it crosses $y = N$. On the graph of the first function, the slope is clearly 0, which indicates that the fixed point is stable, in agreement with the analytical prediction. On the graph of the second function, the slope is negative and steeper than -1. This indicates that the fixed point is unstable, again consistent with our analysis above.

Third, we graph a few iterations of the cobweb plot to obtain an idea about the dynamics of the population over time. As expected, for the first function with $r = 2$, the solution quickly approaches the carrying capacity ({numref. In the second function, however, $r = 4$ and the carrying capacity is unstable. In {numref}`fig-cobweb2` we observe a wild pattern of jumps that never approach any particular value.

We have seen how graphical tools can be used to analyze and predict the behavior of a dynamical system. In the case of the logistic model, we never found the analytic solution, because it frequently does not exist as a formula. Finding the fixed points and analyzing their stability, in conjunction with looking at the behavior of a cobweb plot, allowed us to describe the dynamics of population growth in the logistic model, without doing any "mathematics". Together, the analytical and graphical analysis provide complementary tools for biological modelers.

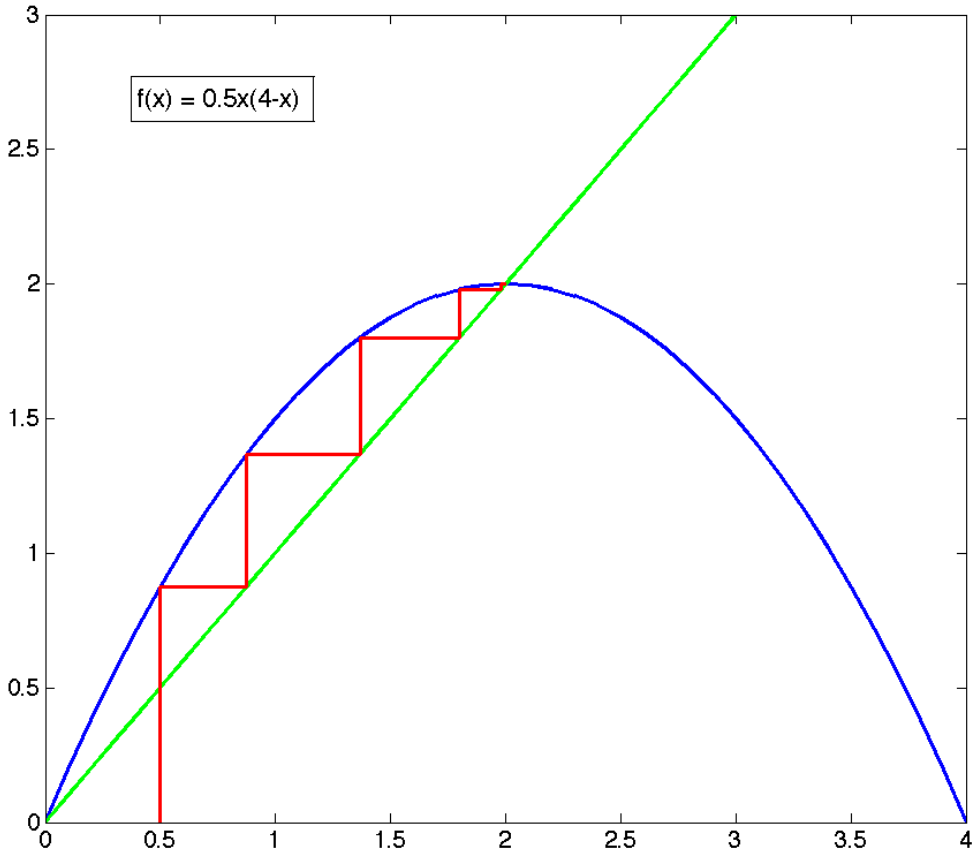


Figure 4.1: Cobweb plot of the logistic model with $r = 2$, showing a solution converging to the stable fixed point at the intersection of the graphs of the function and the identity line

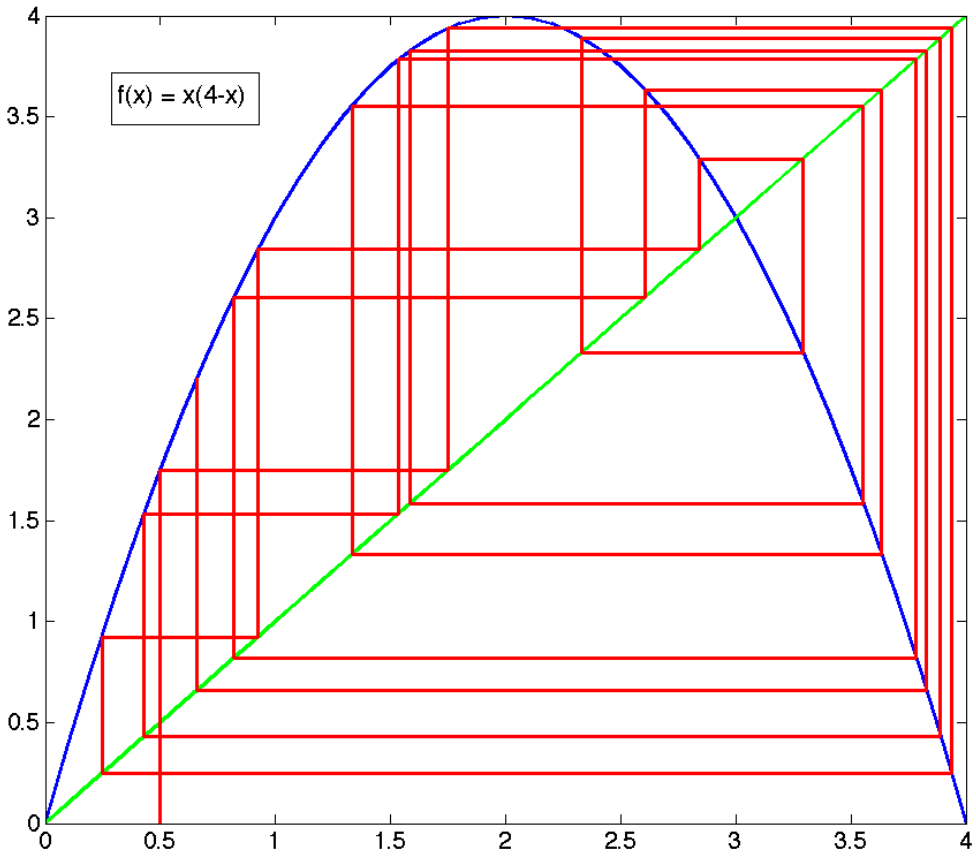


Figure 4.2: Cobweb plot of the logistic model with $r = 4$, showing a as solution bouncing around the unstable fixed point

4.1.1 chaos in discrete dynamical systems

In this chapter we learned to analyze the dynamics of solutions of nonlinear discrete-time dynamical systems without solving them on paper. In the last two sections we focused on the logistic difference equation as a simple nonlinear model with a rich array of dynamic behaviors. In this section we will summarize the analysis and draw conclusions for difference equation models in biology. This behavior was brought to the attention of biologists by John Maynard Smith (**smith_mathematical_1968?**) and Robert May (**may_bifurcations_1976?**).

Why does the logistic model behave so strangely in the second example above? We can use numerical simulations to plot the long-term solutions for the dependent variable for a range of parameter values, let us say between $2.5 < r < 4$. Then we plot the values to which the simulation converged (whether it is one, two, or many) on the y-axis, and the value of the parameter r on the x-axis. The resulting *bifurcation diagram* in shown in {numref}`fig-log-bifur`. The value of the parameter r is plotted on the horizontal axis, and the set of values that the dependent variable takes in the long run is shown on the vertical axis. There is only one stable fixed point for $r < 3$, then we see a 2-cycle appear for $3 < r < 3.45$. For values of r greater than about 3.45, a series of period-doubling bifurcations occur with shorter and shorter intervals of r. This is called a *period-doubling cascade*, which culminates at the value of $r \approx 3.57$, where the number of points in the cycle becomes essentially infinite. The sequence of values of r at which period-doubling occurs is approximately:

- period 2; $r_1 = 3$
- period 4; $r_2 \approx 3.449$
- period 8; $r_3 \approx 3.544$
- period 16; $r_4 \approx 3.564$
- period 32; $r_5 \approx 3.569$
- period ∞ (chaos); $r_\infty \approx 3.570$

For $r > r_\infty$, we observe a remarkable behavior found only in nonlinear dynamical systems, called *chaos*. Chaos is characterized by two qualities:

Characteristics of chaos

1. **Aperiodic behavior**: the dependent variable never repeats a value exactly, instead bouncing around an infinite set of values for all time
2. **Sensitive dependence on initial conditions**: no matter how close two initial conditions in a chaotic system, given enough time the two trajectories will diverge and lose any resemblance

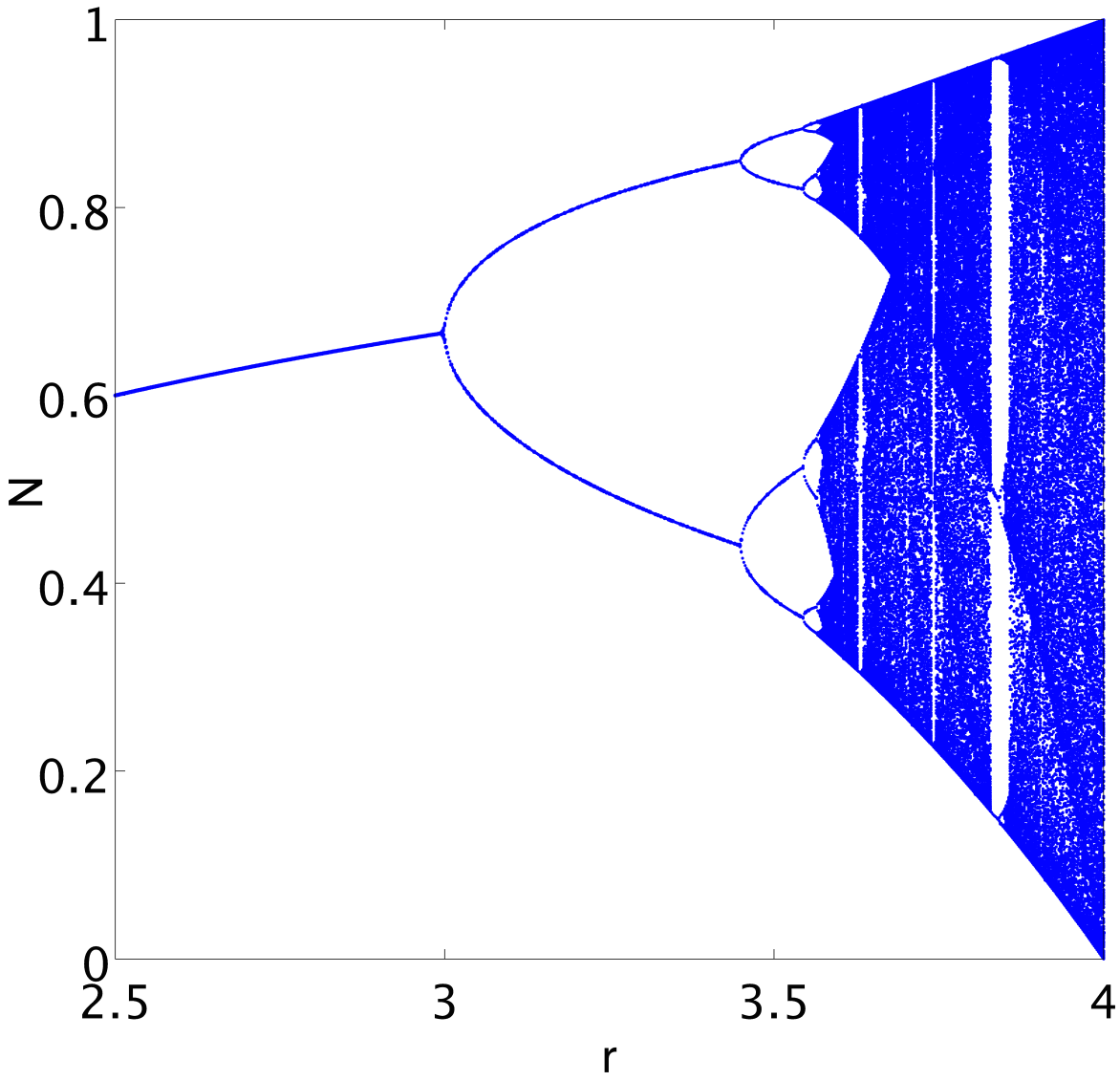


Figure 4.3: Bifurcation diagram for the logistic map $N(t+1) = r(1 - N(t))N(t)$, with the parameter r on the horizontal axis and the vertical axis showing the values of the stable fixed point (for $r < 3$), then the values of the period two oscillation, the period four oscillation, etc., and for r greater than the critical value shows some of the values the solution chaotically jumps through.

What is especially surprising about chaos is that for a given initial condition a chaotic model gives a completely predictable and reproducible sequence of values of the dependent variable. However, given finite machine precision, or any error in initial conditions, a chaotic system is practically unpredictable and irreproducible. But there is a fundamental difference between deterministic chaos and a stochastic system, e.g. the model of coin tosses where knowing the previous result of the coin flip does not allow us to predict the next result, even under ideal conditions.

Notice in figure , that for $r > r_\infty$, chaotic behavior is observed only for some values of r. As you can see in figure , there are "bands of periodicity", where the attractor is a sequence of n numbers (and periods of 3,5, etc. are observed), alternating with bands of chaos. This illustrates that even the simplest nonlinear discrete dynamical systems can have incredibly complex behavior. When these results were first published by May in the 1970s, they revolutionized both the mathematical understanding of dynamical systems, and the field of theoretical biology. In one-variable dynamic systens, chaos occurs only in discrete-time dynamical models, but for three or more variables continuous time (ODE) dynamical systems also can behavechaotically.

As a mathematical side-note, if one looks at the differences between successive values of r_n, they behave like a geometric sequence, getting smaller and smaller by a constant fraction:

$$\delta_n = \frac{r_n - r_{n-1}}{r_{n+1} - r_n}$$

It is a remarkable fact that δ_n approaches a constant value when n gets large, 4.6692..., known as the Feigenbaum constant. It can be proven that this constant is the same for other iterated maps with the same shape as the downward parabola of the logistic map (e.g. $f(x) = \sin(x)$). Explaining why this deep mathematical fact is true is far outside the bounds of this course. (**strogatz_nonlinear_2001?**)

Chaos was a popular topic back in the 1980s and 90s, and even inspired popular books (**gleick_chaos:_1988?**). It is in fact remarkable that very simple difference equations can have solutions of apparently great complexity. This is intriguing because it appeals to a fairly universal human desire for simple explanations for complicated phenomena. The popular exposure to what was dubbed "chaos theory" (which is not an actual mathematical topic) spawned some inaccurate cliches, such as "a butterfly flapping its wings in South America can cause a hurricane to form and hit Florida". The image refers to the phenomenon of sensitive dependence on initial conditions, but of course it is utterly ridiculous to draw a causal arrow between a butterfly (one of an enumerable number of things changing the "initial conditions") and large-scale atmospheric phenomena. While there is some evidence that weather patters are complex systems that exhibit chaotic behavior, we lack the ability to isolate and control all influences that may perturb it, so pinning it on a butterfly is pretty unfair.

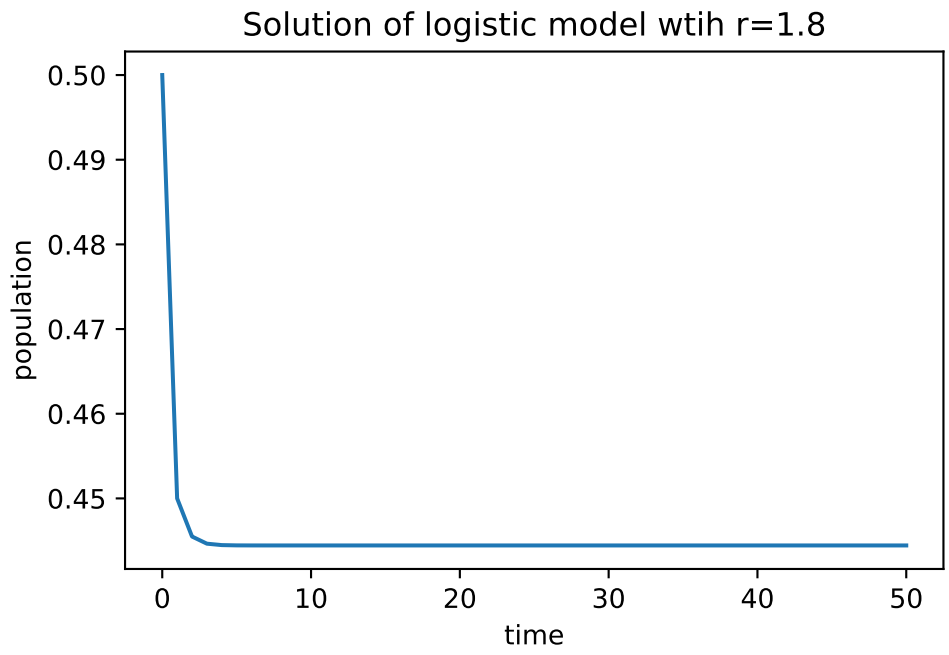
Despite the initial flurry of excitement, so-called chaos theory has failed to make a big impact on our understanding of complex biological systems. Although it is still quite fascinating

intellectually, a simple model like the logistic model is not an adequate model for any realistic population, particularly for large values of r where the chaotic behavior occurs. We now appreciate that the essential complexity of biological system requires multiple interacting variables which cannot be reduced to a single equation. However, there has been some successful observation of chaotic behavior in a population of flour beetles, which seemed to agree with predictions of a three-variable difference equation model (**costantino_chaotic_1997?**).

We have seen how graphical tools can be used to analyze and predict the behavior of a discrete-time dynamical system. We investigated the logistic model by finding the fixed points and analyzing their stability. Together with analysis of the graph of the updating function and making a cobweb plot, this allowed us to describe the dynamics of population growth in the logistic model, without doing any "math". Together, analytical and graphical analysis provide powerful tools for biological modelers.

Q3.1: For the logistic model with an initial population of 0.5 and $r = 1.1$, compute the first 50 iterations using the same for loop iteration you used above and plot the solution against time.

```
numsteps = 50 #set number of iterations
r = 1.8 #set parameter
N = np.zeros(numsteps+1) #initialize solution vector
N[0]=.5 #initial value
t = range(numsteps+1) #initialze time vector
a = -10
for i in range(numsteps):
    N[i+1] = r*N[i]*(1-N[i]) #logistic population model
plt.plot(t,N) #plot solution
plt.xlabel('time')
plt.ylabel('population')
plt.title('Solution of logistic model wtih r='+str(r))
plt.show()
```

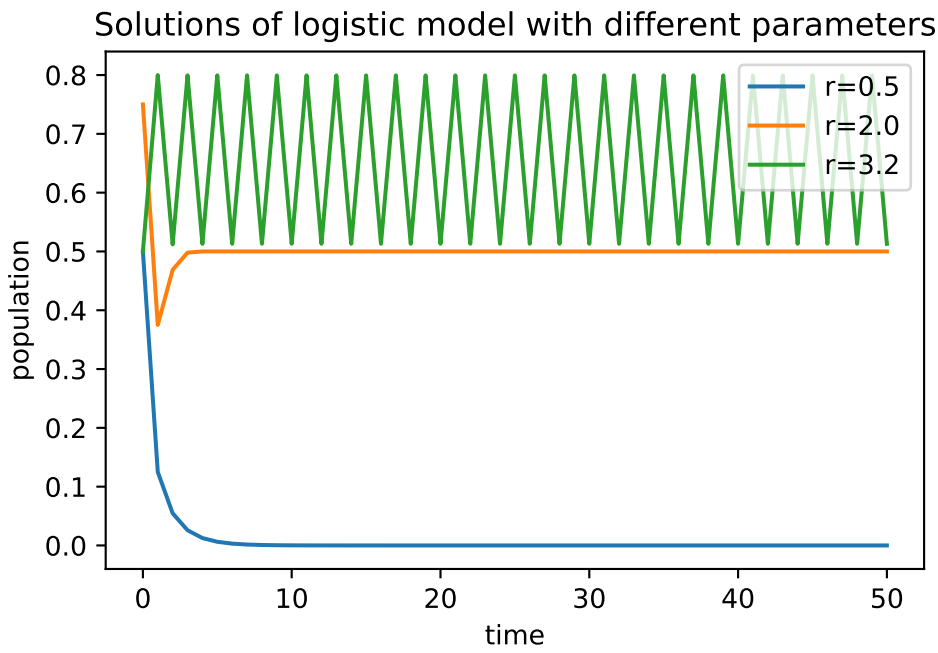


Q3.2: Change the parameter r to the following values: 0.5, 2.0, and 3.2, and in each case plot the solutions against time in separate figures. Describe each plot with a sentence.

```
numsteps = 50 #set number of iterations
r = .5 #set parameter
N = np.zeros(numsteps+1) #initialize solution vector
N[0]=.5 #initial value
t = range(numsteps+1) #initialze time vector
a = -10
for i in range(numsteps):
    N[i+1] = r*N[i]*(1-N[i]) #linear population model
plt.plot(t,N, label = 'r='+str(r)) #plot solution


r = 2.0 #set parameter
N = np.zeros(numsteps+1) #initialize solution vector
N[0]=0.75 #initial value
t = range(numsteps+1) #initialze time vector
a = -10
for i in range(numsteps):
    N[i+1] = r*N[i]*(1-N[i]) #linear population model
plt.plot(t,N, label = 'r='+str(r)) #plot solution
```

```
r = 3.2 #set parameter
N = np.zeros(numsteps+1) #initialize solution vector
N[0]=.5 #initial value
t = range(numsteps+1) #initialze time vector
a = -10
for i in range(numsteps):
    N[i+1] = r*N[i]*(1-N[i]) #linear population model
print(N)
plt.plot(t,N, label = 'r='+str(r)) #plot solution
plt.xlabel('time')
plt.ylabel('population')
plt.title('Solutions of logistic model with different parameters')
plt.legend()
plt.show()
```

```
[0.5        0.8        0.512      0.7995392  0.51288406 0.7994688
 0.51301899 0.79945762 0.51304043 0.79945583 0.51304386 0.79945554
 0.51304441 0.7994555  0.51304449 0.79945549 0.51304451 0.79945549
 0.51304451 0.79945549 0.51304451 0.79945549 0.51304451 0.79945549
 0.51304451 0.79945549 0.51304451 0.79945549 0.51304451 0.79945549
 0.51304451 0.79945549 0.51304451 0.79945549 0.51304451 0.79945549
 0.51304451 0.79945549 0.51304451 0.79945549 0.51304451 0.79945549
 0.51304451 0.79945549 0.51304451]
```

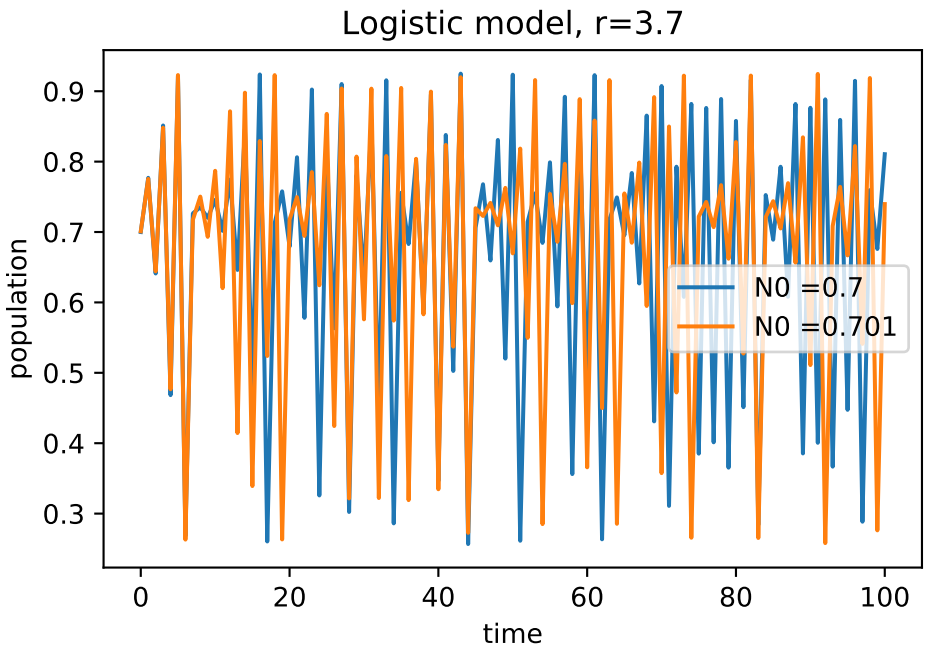


- The solution for r=0.5 decreases to zero
- The solution for r=2.0 stays at the fixed point of 0.5
- The solution for r=3.2 oscillates between two values indefinitely

Increase the parameter r further until you see strange, aperiodic behavior called chaos. Report at least one value of r at which you see chaotic dynamics.

```
numsteps = 100 #set number of iterations
r = 3.7 #set parameter
N = np.zeros(numsteps+1) #initialize solution vector
N[0]=.7 #initial value
t = range(numsteps+1) #initialze time vector
a = -10
for i in range(numsteps):
    N[i+1] = r*N[i]*(1-N[i]) #linear population model
plt.plot(t, N, label = 'N0 =' + str(N[0]))

numsteps = 100 #set number of iterations
r = 3.7 #set parameter
N = np.zeros(numsteps+1) #initialize solution vector
N[0]=.701 #initial value
t = range(numsteps+1) #initialze time vector
a = -10
```

```
for i in range(numsteps):
    N[i+1] = r*N[i]*(1-N[i]) #linear population model
plt.plot(t,N, label = 'N0 =' + str(N[0])) #plot solution

plt.xlabel('time')
plt.ylabel('population')
plt.title('Logistic model, r='+str(r))
plt.legend()
plt.show()
```



At $r = 3.7$ the solution bounces around without any apparent pattern, which is called chaos.

5 Discrete models of higher order

It is not unusual for biological systems to have multiple variables which influence each other, and thus need to be accounted in any model that aims to be useful. In this unit we will learn how to construct such models, and the methods for analyzing, solving, and numerically simulating them. We will see how models with two or more variables are used in a variety of biological fields: to describe population demographics, motility of cochlear cells, psychology of human relationships, gene regulation, and motion of molecular structures.

We will need new mathematical tools in order to analyze models with multiple variables. These methods are primarily from the realm of linear algebra. We will express multiple equations in terms of matrices and vectors, and learn how to operate on these objects. The dynamics of these models can be analyzed by doing calculations with the matrices, specifically finding special numbers and vectors known as eigenvalues and eigenvectors. These concepts, which will be introduced later, are absolutely central to all of applied mathematics, and to computational biology in particular.

In this chapter, the section on modeling is devoted to an old model of a population where individuals live for two generations, known as the Fibonacci model. We then describe how this model can be written down either as a single difference equation of second order, or as two equation of the first order, which may be represented in matrix form. We will learn to solve second order difference equations with an explicit formula, and then introduce some elementary matrix operations. In the computational section we will use the matrix notation to compute numerical solutions for higher order difference equations. Finally, in the synthesis section we will analyze two demographic population models, in which the population is broken into age groups. The matrix notation will be important for concisely representing different parameters for each age group.

In this chapter you will learn to:

- build higher order population models
- express age-structured models in matrix form
- analyze solutions of these models on paper
- use Python for matrix operations
- classify the behavior of solutions of these models

5.1 higher order difference equations

So far we have dealt with difference equations in which the value of the dependent variable at the next time step x_{t+1} depends solely on the variable at the present time x_t. These are known as *first order* difference equations because they only require one step from the present to the future. We will now examine difference equations where the future value depends not only on the present value x_t, but also on the past values: x_{t-1}, etc. The number of time steps that the equation looks into the past is the the order of the scheme.

5.1.1 the Fibonacci model and sequence

The Italian mathematician Leonardo Fibonacci, who lived in the late 12th - early 13th centuries, contributed greatly to the development of mathematics in the western world. For starters, he introduced the Hindu-Arabic numerals we use today, in place of the cumbersome Roman numerals. He also constructed an early model of population growth, which considered a population of individuals that lived for two generations. The first generation does not reproduce, but in the second generation each individual produces a single offspring (or each pair produces a new pair) and then dies. Then the total number of individuals at the next time step is the sum of the individuals in the previous two time steps ({numref}`fig-fib-rabbits`) This is described by the following second order difference equation:

$$N_{t+1} = N_t + N_{t-1}$$

(fibonacci)

The famous Fibonacci sequence is a solution of this equation. For a second-order equations, two initial conditions are required, and if we take $N_0 = 0$ and $N_1 = 1$, then the resulting sequence will look as follows:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...$$

The Fibonacci sequence is famously found in many natural phenomena, including in phyllotaxis (arrangement of parts in plants), and spirals on some mollusk shells, e.g. *Nautilus pompilius* ({numref}`fig-fib-rabbits`). It may be observed by counting the number of spirals that can be drawn between plant units (such as seeds or petals), and observing that alternating the right-handed and left-handed spirals, while moving away from the center, often results in the Fibonacci sequence. The precise reason for this is unclear, although explanations exist, for instance that this pattern provides the most efficient packing of seeds.

[The shell of the *Nautilus pompilius* mollusk has the shape of a Fibonacci spiral, shown here filled with squares of the corresponding size http://mathforum.org/mathimages] (images/fibo_nautilus.jpg)

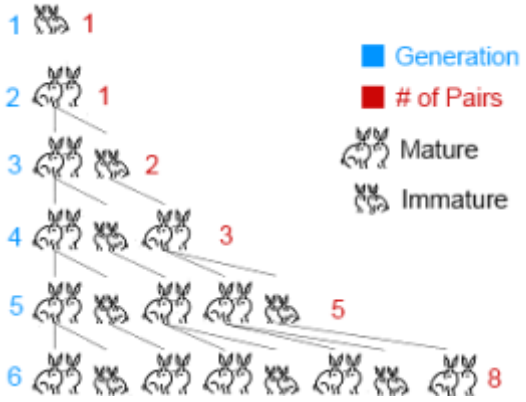


Figure 5.1: The Fibbonaci model with each pair of individuals waiting one generation before producing another pair each subsequent generation <https://artblot.wordpress.com/2013/05/10/rich-with-fibonacci-gold/

5.1.2 matrix representation of discrete time models

The Fibonacci model above can be represented by two equations instead of one, if we consider two dependent variables. Let us represent the number of rabbits in generation 1 (young) by x and in generation 2 (old) by$y$. The new generation at the next time $(t+1)$ is comprised of offspring of the young and old generations at time t, while the old generation at the next time is simply is young generation at the current time. This gives the following set of equations:

$$\begin{aligned} x_{t+1} &= x_t + x_{t-1} \\ x_t &= x_t \end{aligned}$$

The advantage of re-writing a single equation as two is that the new system is first order, that is, only relies on the values of the variables at the current time t. These equations can also be written in *matrix* form:

$$\begin{pmatrix} x_{t+1} \\ x_t \end{pmatrix} = \begin{pmatrix} x_t + x_{t-1} \\ x_t \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_t \\ x_{t-1} \end{pmatrix}$$

This representation is convenient and leads to a set of rules for matrix manipulation. We wrote the write-hand side as a product of a matrix containing the coefficients of x_t and y_t and the vector with the two variables. The product of the matrix and the vector is equal to the original vector.

5.2 Solutions for linear higher order difference equations

5.2.1 solutions of linear difference equations

Solutions for first order linear difference equations are exponential in form. The solutions for second order linear difference equations consist of a sum of two exponentials with different bases. For the following general linear second order difference equation:

$$x_{t+1} = ax_t + bx_{t-1}$$

The solution can be written as follows:

$$x_t = A\lambda_1^t + B\lambda_2^t$$

The solution for a second order difference equation is a sum of two terms that look like solutions to first order difference equations. There are two different types of constants in the solution: the bases of the exponential λ_1, λ_2 and the multiplicative constants A and B. They are different because the exponential parameters depend on the equation itself, but not on the initial conditions, while the multiplicative constants depend only on the initial conditions. Therefore, they can be determined separately:

Outline for solving a second order linear difference equation

1. Substitute the solution $x_t = \lambda^t$ into the difference equation. For the general difference equation, we obtain a the following quadratic relation by dividing everything by λ^{t-1}:

$$\lambda^{t+1} = a\lambda^t + b\lambda^{t-1} \Rightarrow \lambda^2 = a\lambda + b$$

2. Solve the quadratic equation for values of λ which satisfy the difference equation:

$$\lambda_{1,2} = \frac{a \pm \sqrt{a^2 + 4b}}{2}$$

If $a^2 + 4b > 0$, this gives two values of λ; if $a^2 + 4b = 0$, there is a single value, and if $a^2 + 4b < 0$, then no real values of λ satisfy the difference equation.

3. Once we have found the values λ_1 and λ_2, use the initial conditions (e.g. some values x_0 and x_1) to solve for the multiplicative constants:

$$x_0 = A + B;\ x_1 = A\lambda_1 + B\lambda_2$$

Use $A = x_0 - B$ to plug into the second equation: $x_1 = (x_0 - B)\lambda_1 + B\lambda_2$

4. The general solution for A and B is the following, provided $\lambda_2 \neq \lambda_1$:

$$B = \frac{x_1 - x_0\lambda_1}{\lambda_2 - \lambda_1};\ A = \frac{x_0\lambda_2 - x_1}{\lambda_2 - \lambda_1}$$

Let us apply this approach to solving the Fibonacci difference equation {eq}`fibonacci`:

$$\lambda^2 = \lambda + 1 \implies \lambda^2 - \lambda - 1 = 0$$

We find the solutions by the quadratic formula: $\lambda_{1,2} = (1 \pm \sqrt{5})/2$.

Now let us use the initial conditions $N_0 = 0$ and $N_1 = 1$. The two multiplicative constants must then satisfy the following:

$$0 = A + B;\ 1 = A\lambda_1 + B\lambda_2$$

By the formula we found above, the initial conditions are:

$$A = \frac{-1}{\lambda_2 - \lambda_1} = \frac{1}{\sqrt{5}};\ B = \frac{1}{\lambda_2 - \lambda_1} = \frac{-1}{\sqrt{5}}$$

The complete solution, which gives the t-th number in the Fibonacci sequence is:

$$N_t = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^t - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^t$$

There are several remarkable things about this formula. First is the fact that despite the abundance of irrational numbers, for each integer t the number N_t is an integer. One can check this by programming the formula in your favorite language, and plugging in any value of t.

Second, an important feature of the Fibonacci sequence is the ratio between successive terms in the sequence. Notice that of the two terms in the formula, $(\frac{1+\sqrt{5}}{2})^t$ grows as t increases, while $(\frac{1-\sqrt{5}}{2})^t$ decreases to zero, because the first number is greater than 1, while the second is less than 1. This means that for large t, the terms in the Fibonacci sequence are approximately equal to:

$$N_t \approx \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^t$$

Since each successive term is multiplied by $(1+\sqrt{5})/2$, the ratio between successive terms, $\phi = N_{t+1}/N_t$ approaches the value $\phi = (1+\sqrt{5})/2 \approx 1.618$ for increasing t.

This number $(1+\sqrt{5})/2$ is called the *golden ratio* or *golden section*, and was known from antiquity as the most aesthetically pleasing proportion in architecture and art, when used as a ratio between the height and width of the piece of art. Algebraically, the golden ratio is defined to be the number that is both the ratio between two quantities, e.g. a and b, and also the ratio between the sum of the two quantities $(a+b)$ and the larger of the quantities e.g. b ({numref}`fig-gold-ratio`). Geometrically, the golden ratio can be constructed as the ratio between two sides of a rectangle, a and b, which are also part of the larger rectangle with sides $a+b$ and a. This construction is shown in {numref}`fig-gold-rect`.

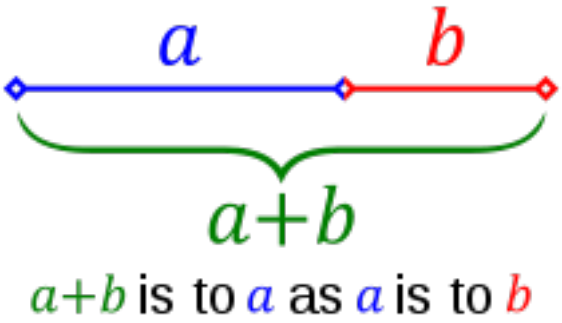


Figure 5.2: Line segments that are in golden proportion to each other http://en.wikipedia.org/wiki/Golden_ratio

To show that the geometric golden ratio is the same as the ratio that appears in the Fibonacci sequence, let us write down the algebraic condition stated above. Because we are interested in the ratio, let the smaller quantity be 1 and the larger one be ϕ; by the definition we obtain the following. $\phi = (\phi+1)/\phi$, thus $\phi^2 - \phi - 1 = 0$. This is the same quadratic equation that we derived for the exponential bases of the solution above. The solution to this equation (by the quadratic formula) is $\phi = (1 \pm \sqrt{5})/2$, and the positive solution is the golden ratio.

5.3 Matrices and vectors

One basic advantage of matrix notation is that it makes it possible to write any set of *linear equations* as a single matrix equation. By linear equations we mean those that contain only constants or first powers of the variables. The field of mathematics studying matrices and their

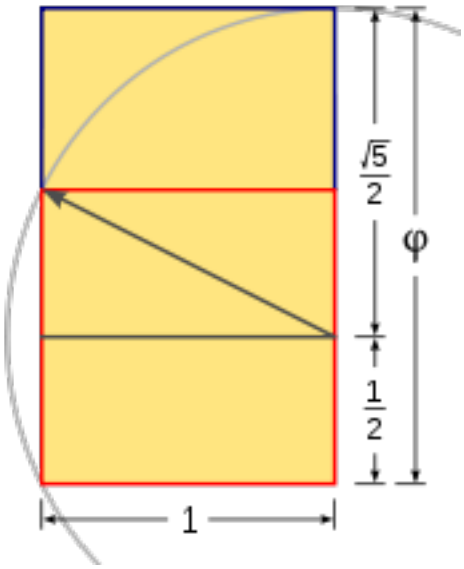


Figure 5.3: Construction of a rectangle with the golden ratio between its sides http://en.wikipedia.org/wiki/Golden_ratio

generalizations is called *linear algebra*; it is fundamental to both pure and applied mathematics. In this section we will learn some basic facts about matrices and their properties.

5.3.1 elementary matrix operations

Now is a good time to properly define what matrices are and how we can operate on them. We have already seen a matrix for the Fibonacci model, but just to make sure all of the terms are clear:

i Definition

A matrix A is a rectangular array of *elements* A_{ij}, in which i denotes the row number (index), counted from the top, and j denotes the column number (index), counted from left to right. The *dimensions* of a matrix are defined by the number of rows and columns, so an n *by* m *matrix* contains n rows and m columns.

i Definition

The elements of a matrix A which have the same row and column index, e.g. A_{33} are called the *diagonal elements*. Those which do not lie on the diagonal are called the *off-diagonal* elements.

For instance, in the 3 by 3 matrix below, the elements a, e, i are the diagonal elements:

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

Matrices can be added together if they have the same dimensions. Then matrix addition is defined simply as adding up corresponding elements, for instance the element in the second row and first column of matrix A is added with the element in the second row and first column of matrix B to give the element in the second row and first column of matrix C. Recall from the previous chapter that rows in matrices are counted from top to bottom, while the columns are counted left to right.

5.3.2 matrix multiplication

Matrices can also be multiplied, but this operation is trickier. For mathematical reasons, multiplication of matrices $A \times B$ does not mean multiplying corresponding elements. Instead, the definition seeks to capture the calculation of simultaneous equations, like the one in the previous section. Here is the definition of matrix multiplication, in words and in a formula (**strang_linear_2005?**):

The *product of matrices A and B* is defined to be a matrix C, whose element c_{ij} is the **dot product of the i-th row of A and the j-th column of B**:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + ... + a_{iN}b_{Nj} = \sum_{k=1}^{q} a_{ik}b_{kj}$$

This definition is possible only if the length of the rows of A and the length of columns of B are the same, since we cannot compute the dot product of two vectors of different lengths. Matrix multiplication is defined only if A is n by q and B is q by m, for any integers n, q, and m and the resulting matrix C is a matrix with n rows and m columns. In other words, **the inner dimensions of matrices have to match** in order for matrix multiplication to be possible. This is illustrated in {numref}`fig-mat-mult`

Example. Let us multiply two matrices to illustrate how it's done. Here both matrices are 2 by 2, so their inner dimensions match and the resulting matrix is 2 by 2 as well:

$$\begin{pmatrix} 1 & 3 \\ 6 & 1 \end{pmatrix} \times \begin{pmatrix} 4 & 1 \\ 5 & -1 \end{pmatrix} = \begin{pmatrix} 1 \times 4 + 3 \times 5 & 1 \times 1 + 3 \times (-1) \\ 6 \times 4 + 1 \times 5 & 6 \times 1 + 1 \times (-1) \end{pmatrix} = \begin{pmatrix} 19 & -2 \\ 29 & 5 \end{pmatrix}$$

One important consequence of this definition is that **matrix multiplication is not commutative**. If you switch the order, e.g. $B \times A$, the resulting multiplication requires dot products

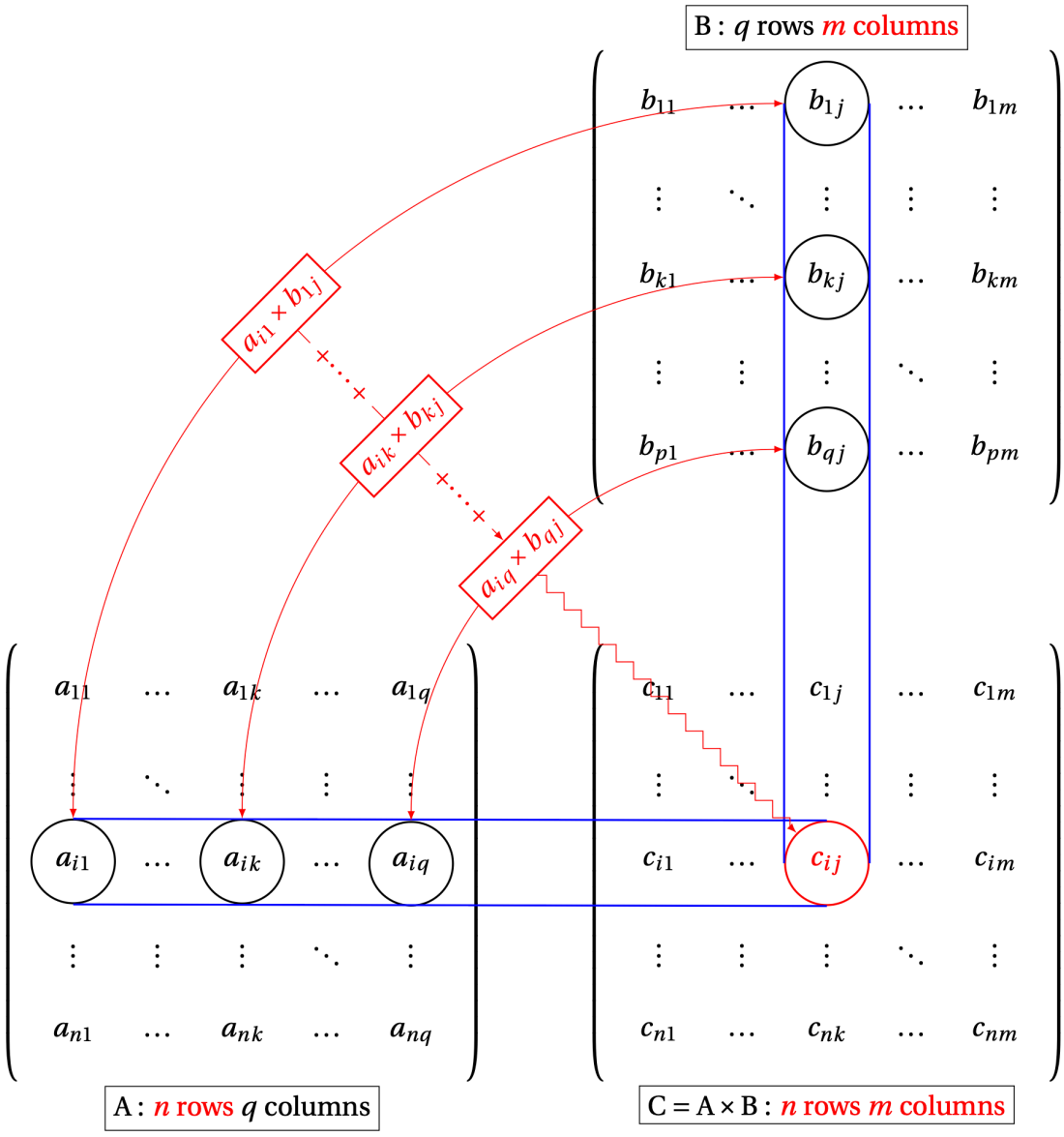


Figure 5.4: Multiplication of two matrices A and B results in a new matrix C

of the rows of B by the columns of A, and except in very special circumstances, they are not the same. In fact, unless m and n are the same integer, the product of $B \times A$ may not be defined at all.

In the example above of the matrix representation of the Fibonacci model, we implicitly used the conventional rules for multiplying matrices and vectors. Each row of the matrix

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

contains the numbers that multiply the two elements of the vector

$$\begin{pmatrix} x_t \\ x_{t-1} \end{pmatrix}$$

in order to generate the two equations $x_{t+1} = x_t + x_{t-1}$ and $x_t = x_t$.

Take the matrix equation for the Fibonacci difference equation above. Put the first two values 0 and 1 into the vector. Then perform the multiplication of the matrix and the vector:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0+1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

We can take the resulting vector and apply the matrix again, to propagate the sequence for one more step:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1+1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Multiplying matrices and vectors is a basic operation that depends on the orientation of the vector. One can only multiply a square matrix by a column vector on the left, as we saw above, not on the right. By the same token, a row vector can only multiply a matrix on the right, and not the left, because we must use the *rows* of the matrix on the left to multiply the *columns* of the matrix on the right. This underscores the important fact that matrix multiplication is not commutative.

5.3.3 matrix inverses

Above we learned the rules of matrix multiplication, and we can write $C = A \times B$, so long as the number of columns in A matches the number of rows in B. However, what if we want to reverse the process? If we know the resulting matrix C, and one of the two matrices, e.g. A, how can we find B? Naively, we would like to be able to divide both sides by the matrix A, and find $B = C/A$. However, things are more complicated for matrices.

Properly speaking, we need to introduce the *inverse* of a matrix A. If we think about inverses of real numbers, a^{-1} is a number that when it multiplies a, results in one. In order to define the equivalent for matrices, we first need to introduce the unity of matrix multiplication.

> **i** Definition
>
> The *identity* matrix is an n by n matrix that does not change another n by n matrix by multiplication:
>
> $$A \times I = I \times A = A$$
>
> The diagonal elements of the identity matrix are 1s and all off-diagonal elements are zero.

Example: Using the definition of matrix multiplication we can verify that this definition works for any 2 by 2 matrix:

$$\begin{pmatrix} -6 & -2 \\ 12 & -1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} -6 \times 1 + -2 \times 0 & -6 \times 0 + -2 \times 1 \\ 12 \times 1 - 1 \times 0 & 12 \times 0 - 1 \times 1 \end{pmatrix} = \begin{pmatrix} -6 & -2 \\ 12 & -1 \end{pmatrix}$$

Now that we have specified the identity, we can define the matrix inverse:

> **i** Definition
>
> A square matrix A has an *inverse matrix* A^{-1} if it satisfies the following:
>
> $$A^{-1} \times A = A \times A^{-1} = I$$

Finding the inverse of a matrix is not simple, and we will be content to let computers handle the dirty work. In fact, not every matrix possesses an inverse. There is a test for existence of an inverse of A, and it depends on the determinant (**strang_linear_2005?**):

A square matrix A possesses an inverse A^{-1} and is called *invertible* if and only if its determinant is not zero.

5.3.4 matrices transform vectors

In this section we will learn to characterize square matrices by finding special numbers and vectors associated with them. At the core of this analysis lies the concept of a matrix as an *operator* that transforms vectors by multiplication. To be clear, in this section we take as default that the matrices A are square, and that vectors $\vec{v}$ are column vectors, and thus will multiply the matrix on the right: $A \times \vec{v}$.

A matrix multiplied by a vector produces another vector, provided the number of columns in the matrix is the same as the number of rows in the vector. This can be interpreted as the matrix transforming the vector $\vec{v}$ into another one: $A \times \vec{v} = \vec{u}$. The resultant vector $\vec{u}$ may or may not resemble $\vec{v}$, but there are special vectors for which the transformation is very simple.

Example. Let us multiply the following matrix and vector (specially chosen to make a point):

$$\begin{pmatrix} 2 & 1 \\ 2 & 3 \end{pmatrix} \times \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 2-1 \\ 2-3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

We see that this particular vector is unchanged when multiplied by this matrix, or we can say that the matrix multiplication is equivalent to multiplication by 1. Here is another such vector for the same matrix:

$$\begin{pmatrix} 2 & 1 \\ 2 & 3 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2+2 \\ 2+6 \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \end{pmatrix}$$

In this case, the vector is changed, but only by multiplication by a constant (4). Thus the geometric direction of the vector remained unchanged.

Generally, a square matrix has an associated set of vectors for which multiplication by the matrix is equivalent to multiplication by a constant. This can be written down as a definition:

i Definition

An *eigenvector* of a square matrix A is a vector $\vec{v}$ for which matrix multiplication by A is equivalent to multiplication by a constant. This constant λ is called its *eigenvalue* of A corresponding the the eigenvector $\vec{v}$. The relationship is summarized in the following equation:

$$A \times \vec{v} = \lambda \vec{v}$$

(def-eigen)

Note that this equation combines a matrix (A), a vector ($\vec{v}$) and a scalar λ, and that both sides of the equation are column vectors. This definition is illustrated in {numref}`fig-eig-vec`, showing a vector (v) multiplied by a matrix A, and the resulting vector λv, which is in the same direction as v, due to scalar multiplying all elements of a vector, thus either stretching it if $\lambda > 1$ or compressing it if $\lambda < 1$. This assumes that λ is a real number, which is not always the case, but we will leave that complication aside for the purposes of this chapter.

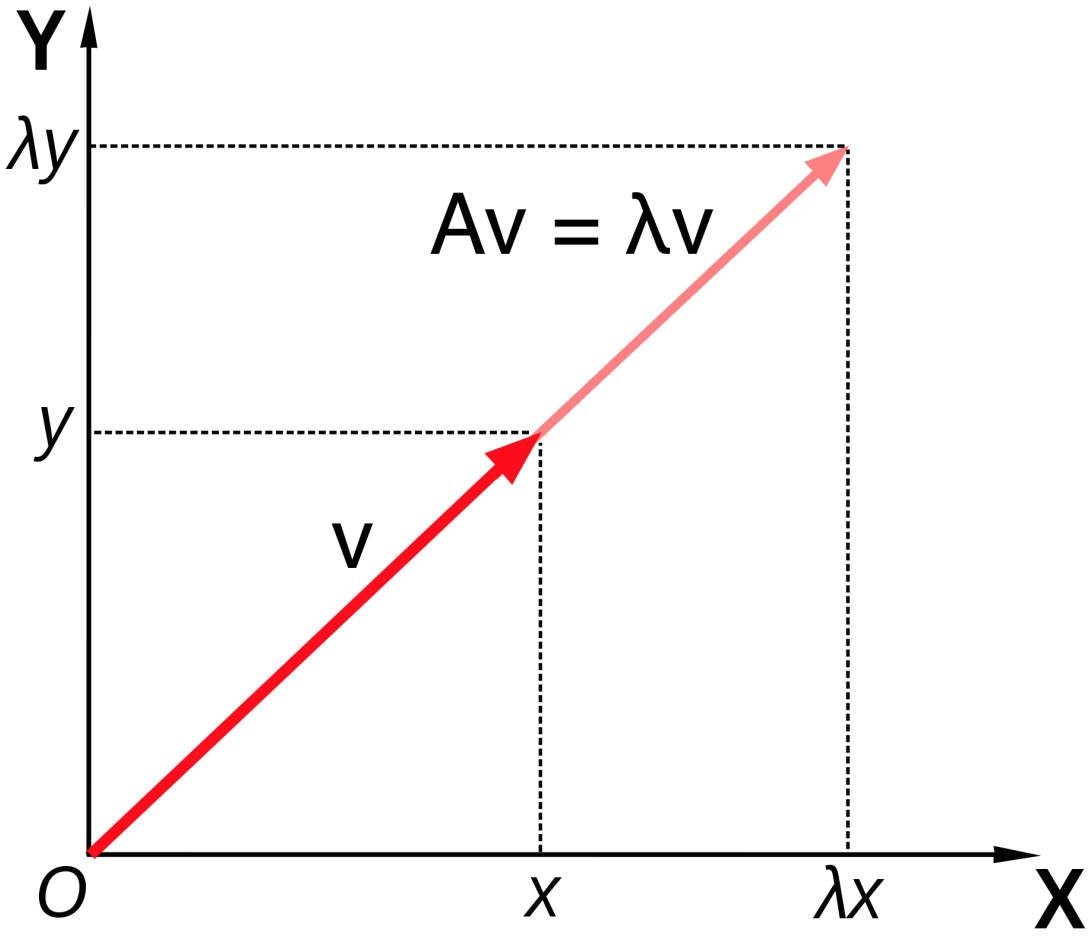


Figure 5.5: Illustration of the geometry of a matrix A multiplying its eigenvector v, resulting in a vector in the same direction λv. (Figure by Lantonov under CC BY-SA 4.0 via Wikimedia Commons)

The definition does not specify how many such eigenvectors and eigenvalues can exist for a given matrix A. There are usually as many such vectors $\vec{v}$ and corresponding numbers λ as the number of rows or columns of the square matrix A, so a 2 by 2 matrix has two eigenvectors and two eigenvalues, a 5x5 matrix has 5 of each, etc. One ironclad rule is that there cannot be more distinct eigenvalues than the matrix dimension. Some matrices possess fewer eigenvalues

than the matrix dimension, those are said to have a *degenerate* set of eigenvalues, and at least two of the eigenvectors share the same eigenvalue.

The situation with eigenvectors is trickier. There are some matrices for which any vector is an eigenvector, and others which have a limited set of eigenvectors. What is difficult about counting eigenvectors is that an eigenvector is still an eigenvector when multiplied by a constant. You can show that for any matrix, multiplication by a constant is commutative: $cA = Ac$, where A is a matrix and c is a constant. This leads us to the important result that if $\vec{v}$ is an eigenvector with eigenvalue λ, then any scalar multiple $c\vec{v}$ is also an eigenvector with the same eigenvalue. The following demonstrates this algebraically:

$$A \times (c\vec{v}) = cA \times \vec{v} = c\lambda\vec{v} = \lambda(c\vec{v})$$

This shows that when the vector $c\vec{v}$ is multiplied by the matrix A, it results in its being multiplied by the same number λ, so by definition it is an eigenvector. Therefore, an eigenvector $\vec{v}$ is not unique, as any constant multiple $c\vec{v}$ is also an eigenvector. It is more useful to think not of a single eigenvector $\vec{v}$, but of a **collection of vectors that can be interconverted by scalar multiplication** that are all essentially the same eigenvector. Another way to represent this, if the eigenvector is real, is that an eigenvector as a **direction that remains unchanged by multiplication by the matrix**, such as direction of the vector v in figure . As mentioned above, this is true only for real eigenvalues and eigenvectors, since complex eigenvectors cannot be used to define a direction in a real space.

To summarize, eigenvalues and eigenvectors of a matrix are a set of numbers and a set of vectors (up to scalar multiple) that describe the action of the matrix as a multiplicative operator on vectors. "Well-behaved" square n by n matrices have n distinct eigenvalues and n eigenvectors pointing in distinct directions. In a deep sense, the collection of eigenvectors and eigenvalues defines a matrix A, which is why an older name for them is characteristic vectors and values.

5.3.5 calculating eigenvalues

Finding the eigenvalues and eigenvectors analytically, that is on paper, is quite laborious even for 3 by 3 or 4 by 4 matrices and for larger ones there is no analytical solution. In practice, the task is outsourced to a computer. Nevertheless, it is useful to go through the process in 2 dimensions in order to gain an understanding of what is involved.

First, let us define two quantities that will be useful for this calculation:

> **i** Definition
>
> The *trace* τ of a matrix A is the sum of the diagonal elements: $\tau = \sum_i A_{ii}$

Definition

The *determinant* Δ of a 2x2 matrix A is given by the following: $\Delta = ad - bc$, where

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

For larger matrices, the determinant is defined recursively, in terms of 2x2 submatrices of the larger matrix, but we will not give the full definition here.

From the definition [def:eigen] of eigenvalues and eigenvectors, the condition can be written in terms of the four elements of a 2 by 2 matrix:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} av_1 + bv_2 \\ cv_1 + dv_2 \end{pmatrix} = \lambda \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

This is now a system of two linear algebraic equations, which we can solve by substitution. First, let us solve for v_1 in the first row, to get

$$v_1 = \frac{-bv_2}{a - \lambda}$$

Then we substitute this into the second equation and get:

$$\frac{-bcv_2}{a - \lambda} + (d - \lambda)v_2 = 0$$

Since v_2 multiplies both terms, and is not necessarily zero, we require that its multiplicative factor be zero. Doing a little algebra, we obtain the following, known as the *characteristic equation* of the matrix:

$$-bc + (a - \lambda)(d - \lambda) = \lambda^2 - (a + d)\lambda + ad - bc = 0$$

This equation can be simplified by using two quantities we defined at the beginning of the section: the sum of the diagonal elements called the trace $\tau = a + d$, and the determinant $\Delta = ad - bc$. The quadratic equation has two solutions, dependent solely on τ and Δ:

$$\lambda = \frac{\tau \pm \sqrt{\tau^2 - 4\Delta}}{2}$$

This is the general expression for a 2 by 2 matrix, showing there are two possible eigenvalues. Note that if $\tau^2 - 4\Delta > 0$, the eigenvalues are real, if $\tau^2 - 4\Delta < 0$, they are complex (have

real and imaginary parts), and if $\tau^2 - 4\Delta = 0$, there is only one eigenvalue. This situation is known as degenerate, because two eigenvectors share the same eigenvalue.

Example. Let us take the same matrix we looked at in the previous subsection:

$$A = \begin{pmatrix} 2 & 1 \\ 2 & 3 \end{pmatrix}$$

The trace of this matrix is $\tau = 2 + 3 = 5$ and the determinant is $\Delta = 6 - 2 = 4$. Then by our formula, the eigenvalues are:

$$\lambda = \frac{5 \pm \sqrt{5^2 - 4 \times 4}}{2} = \frac{5 \pm 3}{2} = 4, 1$$

These are the multiples we found in the example above, as expected.

A real matrix can have complex eigenvalues and eigenvectors, but whenever it acts on a real vector, the result is still real. This is because the complex numbers cancel each other's imaginary parts. For discrete time models, it is enough to consider the absolute value of a complex eigenvalue, which is defined as following: $|a+bi| = \sqrt{a^2 + b^2}$. As before, the eigenvalue with the largest absolute value "wins" in the long term.

5.3.6 calculation of eigenvectors on paper

The surprising fact is that, as we saw in the last subsection, the eigenvalues of a matrix can be found without knowing its eigenvectors! However, the converse is not true: to find the eigenvectors, one first needs to know the eigenvalues. Given an eigenvalue λ, let us again write down the defining equation of the eigenvector for a generic 2 by 2 matrix:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} av_1 + bv_2 \\ cv_1 + dv_2 \end{pmatrix} = \lambda \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

This vector equation is equivalent to two algebraic equations:

$$av_1 + bv_2 = \lambda v_1$$
$$cv_1 + dv_2 = \lambda v_2$$

Since we've already found λ by solving the characteristic equation, this is two linear equations with two unknowns (v_1 and v_2). You may remember from advanced algebra that such equations may either have a single solution for each unknown, but sometimes they may have none, or infinitely many solutions. Since there are unknowns on both sides of the equation, we can make both equations be equal to zero:

$$(a-\lambda)v_1 + bv_2 = 0$$
$$cv_1 + (d-\lambda)v_2 = 0$$

So the first equation yields the relationship $v_1 = -v_2 b/(a-\lambda)$ and the second equation is $v_1 = -v_2(d-\lambda)/c$, which we already obtained in the last subsection. We know that these two equations must be the same, since the ratio of v_1 and v_2 is what defines the eigenvector. So we can use either expression to find the eigenvector.

Example. Let us return to the same matrix we looked at in the previous subsection:

$$A = \begin{pmatrix} 2 & 1 \\ 2 & 3 \end{pmatrix}$$

The eigenvalues of the matrix are 1 and 4. Using our expression above, where the element $a = 2$ and $b = 1$, let us find the eigenvector corresponding to the eigenvalue 1:

$$v_1 = -v_2 \times 1/(2-1) = -v_2$$

Therefore the eigenvector is characterized by the first and second elements being negatives of each other. We already saw in the example two subsections above that the vector $(1, -1)$ is such as eigenvector, but it is also true of the vectors $(-1, 1)$, $(-\pi, \pi)$ and $(10^6, -10^6)$. This infinite collection of vectors, all along the same direction, can be described as the eigenvector (or eigendirection) corresponding to the eigenvalue 1.

Repeating this procedure for $\lambda = 4$, we obtain the linear relationship:

$$v_1 = -v_2 \times 1/(2-4) = 0.5v_2$$

Once again, the example vector we saw two subsections $(2, 1)$ is in agreement with our calculation. Other vectors that satisfy this relationship include $(10, 5)$, $(-20, -10)$, and $(-0.4, -0.2)$. This is again a collection of vectors that are all considered the same eigenvector with eigenvalue 4 which are all pointing in the same direction, with the only difference being their length.

5.4 Age-structured population models

It is often useful to divide a population into different groups by age in order to better describe the population dynamics. Typically, individuals at different life stages have distinct mortality and reproductive rates. The total population is represented as a vector, where each component denotes the size of the corresponding age group. The matrix A that multiplies this vector defines the dynamics of the higher order difference equation:

$$\vec{x}_{t+1} = A\vec{x}_t$$

We will now analyze two common *age-structured models* used by biologists and demographers.

5.4.1 Leslie models

One type of age-structured model used to describe population dynamics is called the *Leslie model* (**edelstein-keshet_mathematical_2005?**; **allman_mathematical_2003?**). In this model, there are several different age groups, and after a single time step, individuals in each one all either advance to the next oldest age group or die. This type of can be described in general using the following matrix (called a Leslie matrix):

$$L = \begin{pmatrix} f_1 & f_2 & ... & f_n \\ s_1 & 0 & ... & 0 \\ ... & ... & ... & ... \\ 0 & ... & s_{n-1} & 0 \end{pmatrix}$$

where f_i is the fecundity (number of offspring produced by an individual) of the i-th age group, and s_i is the survival rate of the i-th age group (the fraction of the group that survives and becomes the next age group). Population of the next generation is given by multiplying the age-structure vector of the previous generation: $\vec{x}_{t+1} = L\vec{x}_t$. Note that each age group proceeds straight to the next age group (multiplied by the survival rate) but no individuals stay in the same age group after one time step. Biologically, this assumes a clear, synchronized maturation of every age group in the population. Mathematically, this means that the *diagonal elements* of the matrix (those in the i-th row and i-th column) are 0.

Let us model a hypothetical population in which there are two age groups: a young age group which does not reproduce, with survival rate of 0.4 to become mature, and a mature age group which reproduces with mean fecundity of 2, and then dies. Let j_t be the population of the juveniles at time t, and m_t be the population of mature adults. Then the following Leslie matrix describes this model:

$$\begin{pmatrix} j_{t+1} \\ m_{t+1} \end{pmatrix} = \begin{pmatrix} 0 & 2 \\ 0.4 & 0 \end{pmatrix} \begin{pmatrix} j_t \\ m_t \end{pmatrix}$$

We can also express this model as a single difference equation, with the variable of total population. Because it takes two time steps for a young individual to reproduce, we need to consider the population in two previous time steps. The matrix equation above can be written as the following two equations:

$$j_{t+1} = 2m_t;\ m_{t+1} = 0.4j_t$$

This two-dimensional model can be turned into a second-order model by a simple substitution. The first equation can be written as $j_t = 2m_{t-1}$, and then substitute it into second one to, to obtain:

$$m_{t+1} = 0.8m_{t-1}$$

We can solve this equation using the tools from the analytical section. First, let us find the exponential bases λ:

$$\lambda^2 = 0.8 \Rightarrow \lambda = \pm\sqrt{0.8}$$

To solve the dynamical system completely, let us suppose we have the initial conditions m_0 and m_1. Then we have the following equations to solve:

$$A+B = m_0;\ A\sqrt{0}.8-B\sqrt{0}.8 = m_1 \Rightarrow (m_0-B)\sqrt{0}.8-B\sqrt{0}.8 = m_1 \Rightarrow B = m_0-\frac{m_1}{\sqrt{8}};\ A = \frac{m_1}{\sqrt{8}}$$

We have the following analytic solution of the difference equation:

$$m_t = \frac{m_1}{\sqrt{8}}\sqrt{8}^t - \left(m_0 - \frac{m_1}{\sqrt{8}}\right)\sqrt{8}^t = 2m_1\sqrt{8}^{t-1} - m_0\sqrt{8}^t$$

This solution can be used to predict the long-term dynamics of the population model. Since the bases of the exponentials are less than 1, the total number of individuals will decline to zero. This solution can be verified via a numerical solution of this model. {numref}`fig-leslie` shows the population over 20 time steps, starting with 10 individuals both for m_0 and m_1.

5.4.2 Usher models

Usher models are a modification of the Leslie model, where individuals are allowed to remain in the same age group after one time step. Thus, the form of an Usher matrix is:

$$U = \begin{pmatrix} f_1 & f_2 & ... & f_n \\ s_1 & r_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & ... & s_{n-1} & r_n \end{pmatrix}$$

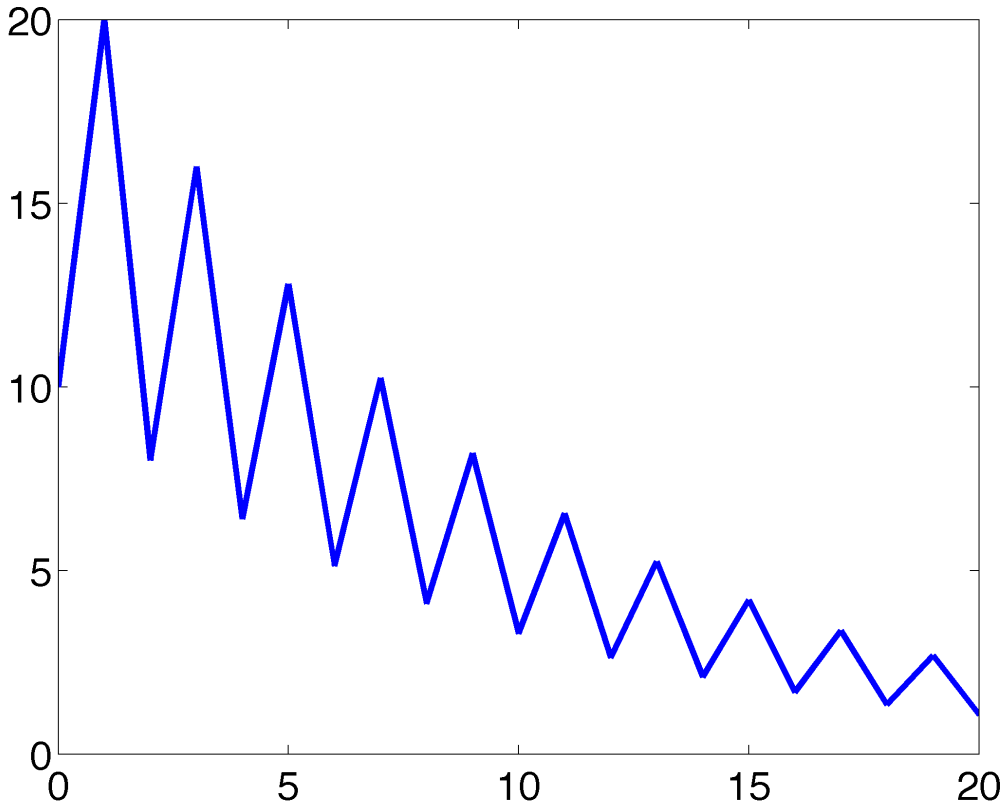


Figure 5.6: A plot of the total population in the Leslie model shown above, showing an oscillatory decay to 0

where r_i is the rate of remaining in the same age cohort.

For instance, it the population model above, we can introduce a rate of adults remaining adults (rather than dead) after a time step (let it be 0.2):

$$U = \begin{pmatrix} 0 & 2 \\ 0.4 & 0.2 \end{pmatrix}$$

$$j_{t+1} = 2m_t;\ m_{t+1} = 0.4j_t + 0.2m_t$$

Once again, we can find the solution for this model by recasting it as a single second-order equation. Let us substitute $2m_{t-1}$ for j_t to obtain the following:

$$m_{t+1} = 0.4(2m_{t-1}) + 0.2m_t$$

We can solve this second-order equation in the same fashion as above:

$$\lambda^2 = 0.2\lambda + 0.8 \Rightarrow \lambda = (0.2 \pm \sqrt{0.04 + 3.2})/2 = (0.2 \pm 1.8)/2 = 1, -0.8$$

The two exponential bases are 1 and -0.8, and therefore the solution has the general form $m_t = A + B(-0.8)^t$. The behavior of the solution over the long term is going to stabilize at some level A, determined by the initial conditions, because the term $B(-0.8)^t$, when raised to progressively larger powers, will decay to 0.

We can run a computer simulation to test this prediction, and see that the total population indeed approaches a constant. Starting with population of 10 individuals in the first two time steps, the time course of the population is plotted in {numref}`fig-usher`.

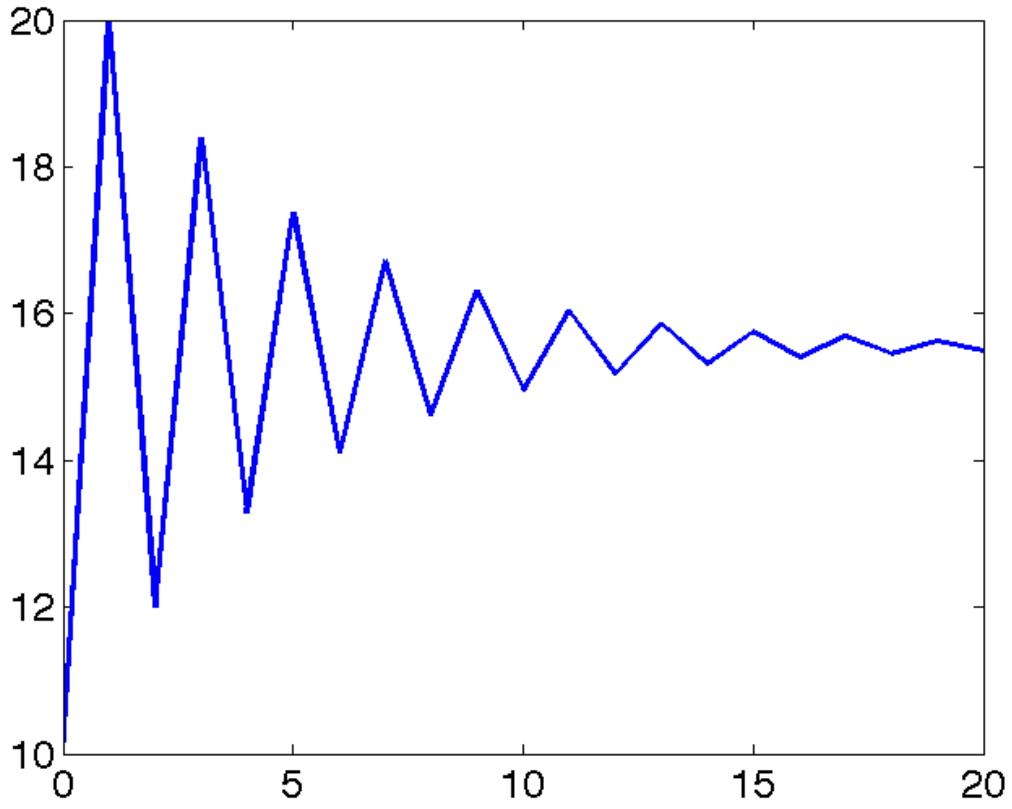


Figure 5.7: The total population of the Usher model shown above, showing oscillation and converging to a single value.

6 Matrix multiplication and population models

The solution of a difference equation can be found numerically using the matrix and vector form we introduced above. As we saw in the Fibonacci example, the next vector in the sequence can be obtained by multiplying the previous vector by the defining matrix. Let $\vec{x}_t$ be the vector containing n values of the dependent variables, and A be the defining matrix of dimension k by k. The solutions are obtained by repeated multiplication by the matrix A:

$$\vec{x}_{t+1} = A\vec{x}_t$$

We will now show how to implement this procedure in a program. For the simulation to be run, the program must set the following required components: the matrix defining the difference map, sufficient number of initial values, and the number of steps desired to iterate the solution. At each step, the current value of the vector of dependent variables is multiplied by the matrix A. In the following pseudocode I use two-dimensional arrays with two indices (row and column), and a colon in place of index indicates all of the elements in that dimension, e.g. $x[0, :]$ indicates the entire first row of array x. I assume that programming language has an operator for multiplying matrices, which is indicated by the multiplication symbol $\times$.

Solution of matrix discrete-time models

- set the number of variables k
- set age-structured matrix A
- set n to be the number of iterations (time steps)
- set the initial condition vector x_0
- initialize array x with n rows and $n + 1$ columns
- set the first column to x_0
- for i from 0 to $n - 1$
  - $x[:, i + 1] \leftarrow A \times x[:, i]$

This code produces a rectangular array x with k rows and $n + 1$ columns. The values of the variables at time j are stored in the vector $x[:, j]$. Conversely, in order to follow the dynamics of a particular variable over time, e.g. number i, through all n time steps, we can plot the vector $x[:, i]$.

Let us take the Fibonacci model again in the matrix form, with the matrix A and initial vector $\vec{x}_0$ as follows:

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix};\ \vec{x}_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

After iterating the matrix equation for 10 time steps, we obtain the following array X, with the fist and second row representing the population at the current and the previous time step, respectively, and the column representing time step:

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 5 & 8 & 13 & 21 & 34 & 55 & 89 \\ 1 & 1 & 2 & 3 & 5 & 8 & 13 & 21 & 34 & 55 \end{array}$$

```
#Necessary imports
import numpy as np #package for work with arrays and matrices -- this week including some
import matplotlib.pyplot as plt #package with plotting capabilities
```

6.1 Matrix models in Python

In this section you will use Python's linear algebra library to compute the characteristic polynomial, eigenvalues, and eigenvectors of the models.

We saw in the section above that the found the eigenvalues by rewriting the equation for λ as a kth order polynomial, then found its roots. Python has a command for that, we can construct the characteristic polynomial of a matrix using the function poly(A), where A is a matrix. More specifically, we can find the coefficients for the characteristic polynomial.

Consider the Leslie population model from the example above:

$$\begin{pmatrix} j_{t+1} \\ m_{t+1} \end{pmatrix} = \begin{pmatrix} 0 & 2 \\ 0.4 & 0 \end{pmatrix} \times \begin{pmatrix} j_t \\ m_t \end{pmatrix}$$

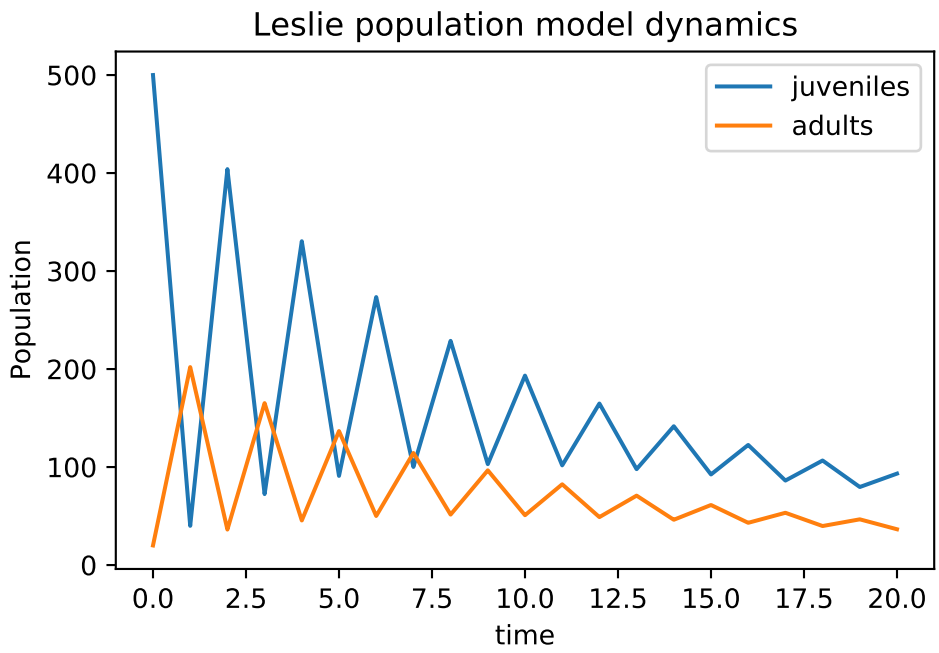
where j_t is the number of juveniles after t generations and m_t is the number of mature individuals after t generations. Propagation of this model requires multiplication of the matrix and the population vector. There is a special symbol in Python for this operation:

```
L=np.array([[0, 2], [0.4, 0]]) # define Leslie matrix
print(L)
pop = np.array([50, 10]) # define population column vector
print(pop)
new_pop = L@pop
print(new_pop)
```

```
[[0.  2. ]
 [0.4 0. ]]
[50 10]
[20. 20.]
```

This propagates the population by one time step only. To compute a numeric solution of this population over a number of time steps, use a for loop like in our last week's assignment. The only difference is that the solution is now a two-dimensional array instead of a one-dimensional one, with two rows for the two ages and numsteps+1 columns, and it needs to be pre-allocated before the for loop:

```
numsteps = 20; #number of time steps
L=np.array([[0, 2], [0.4, 0.1]]) # define Leslie matrix
pop = np.zeros([2, numsteps+1])
pop[:,0] = np.array([500,20]) #initialize the array with 50 juveniles and 10 adults
for i in range(numsteps):
    pop[:,i+1] = L@pop[:,i] #propagate the population vector for one step
plt.plot(pop[0,:],label='juveniles')
plt.plot(pop[1,:],label='adults')
plt.xlabel('time')
plt.ylabel('Population')
plt.title('Leslie population model dynamics')
plt.legend()
plt.show()
```



6.1.1 Eigenvalue and eigenvector analysis

The *eigenvalues* of the matrix L determine the dynamics of the population, the the *eigenvectors* determine the population structure. Python has a single function for finding eigenvalues and eigenvectors: `np.linalg.eig()`.

```
eVals, eVecs = np.linalg.eig(L)

print('Eigenvalues:')
print(eVals) #the order is flipped from the other method, but that's ok
print('Eigenvectors:')
print(eVecs)
```

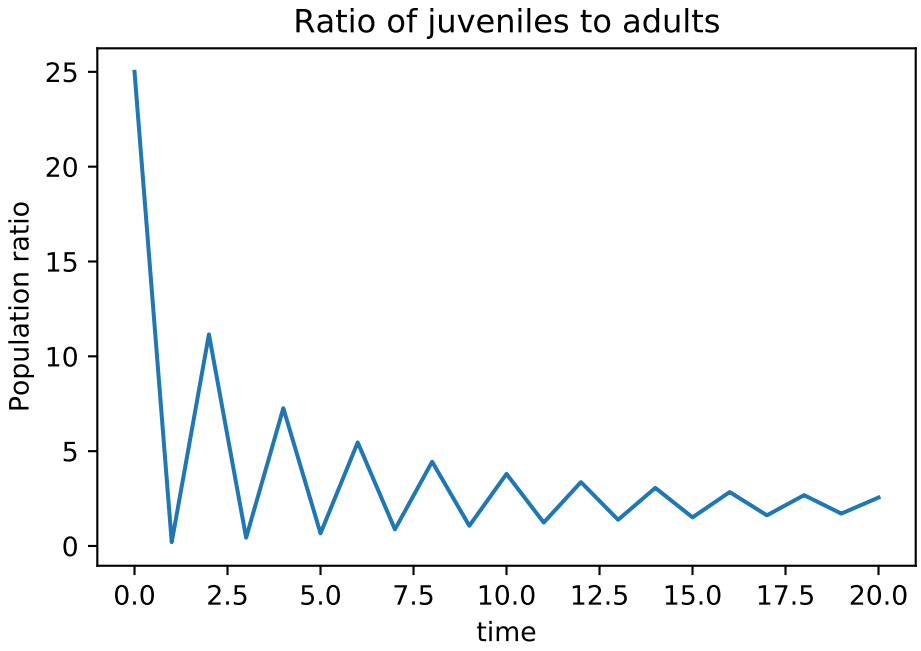
```
Eigenvalues:
[-0.84582364  0.94582364]
Eigenvectors:
[[-0.92102181 -0.90400779]
 [ 0.38951101 -0.42751597]]
```

Each column of the eVecs matrix corresponds to an eigenvaluein the eVals array (i.e. the first column of eigenvectors corresponds to the first element in eVals).

The largest (in absolute value) eigenvalue is the *dominant eigenvalue* and determines the long term behavior of the population. In this example, both eigenvalues are equal in absolute value and are less than 1, which predicts population decay.

The lack of a single dominant eigenvalue means that the population structure (ratio of juveniles and adults) does not converge to a stable fraction:

```
plt.plot(pop[0,:]/pop[1,:])
plt.xlabel('time')
plt.ylabel('Population ratio')
plt.title('Ratio of juveniles to adults')
plt.show()
```



6.1.2 Eigenvectors and population structure

Let us modify the Leslie matrix to allow the adults to survive with probability 0.2, which creates an Usher matrix with the following eigenvalues and eigenvectors. Below we also calculate the fraction of juveniles and adults in the long-term population:

```
U=np.array([[0, 2], [0.4, 0.2]]) # define Usher matrix
print(U)
```

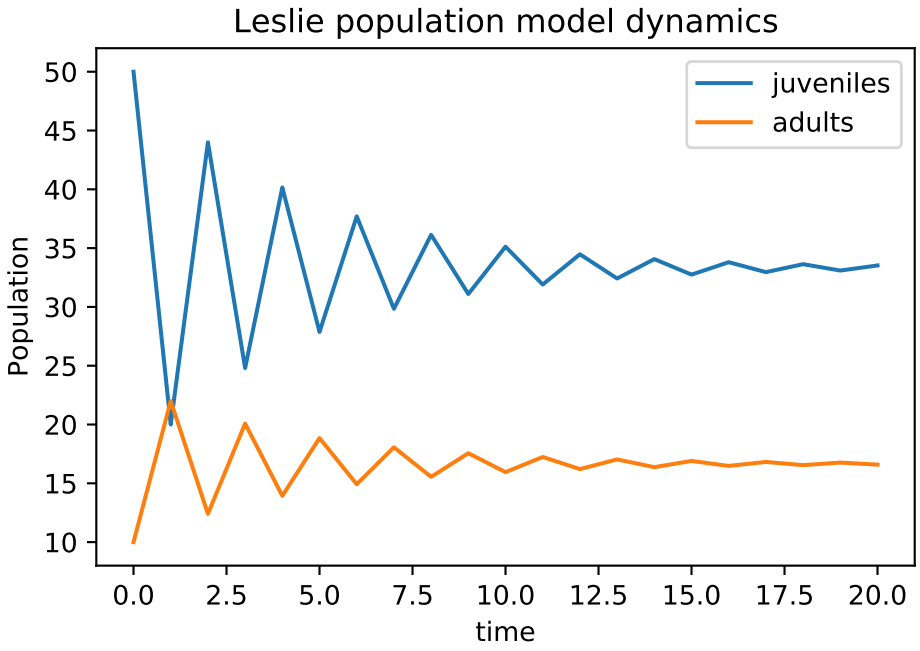
```
eVals, eVecs = np.linalg.eig(U)

print('Eigenvalues:')
print(eVals) #the order is flipped from the other method, but that's ok
print('Eigenvectors:')
print(eVecs)

print('The long term fractions of juveniles and adults are: ' + str(eVecs[:,1]/sum(eVecs[:
```

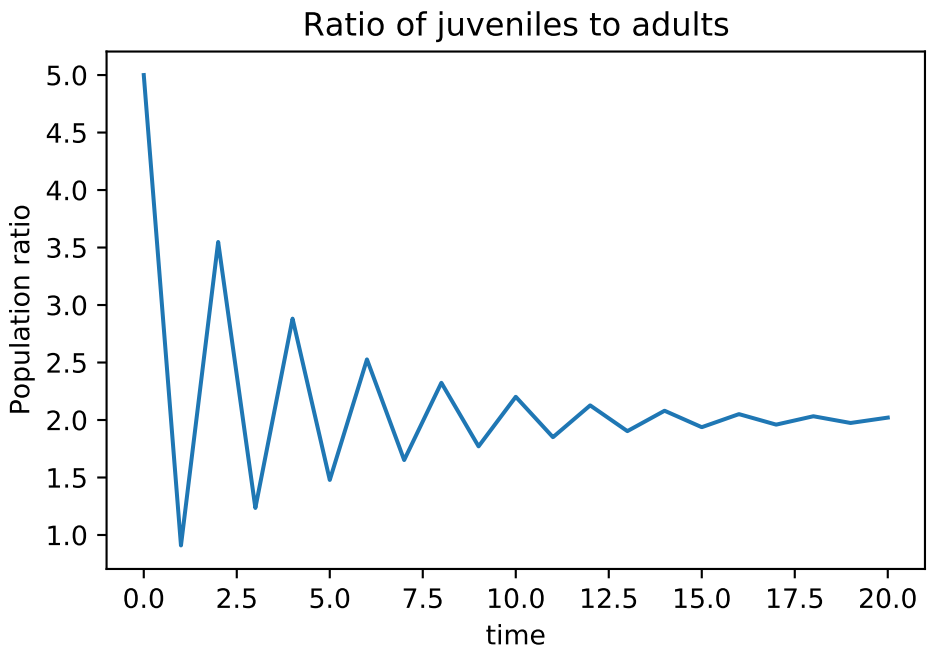
```
[[0.  2. ]
 [0.4 0.2]]
Eigenvalues:
[-0.8  1. ]
Eigenvectors:
[[-0.92847669 -0.89442719]
 [ 0.37139068 -0.4472136 ]]
The long term fractions of juveniles and adults are: [0.66666667 0.33333333]
```

Why did we use the second column (index 1)? Because it corresponds to the dominant eigenvalue 1, as you can check by looking at eVals. Notice that the population distribution remains stable in this population even as the total population declines, as can be seen by plotting the ratio of the juveniles to the adults:

```
numsteps = 20; #number of time steps
U=np.array([[0, 2], [0.4, 0.2]]) # define Leslie matrix
pop = np.zeros([2, numsteps+1])
pop[:,0] = np.array([50,10]) #initialize the array with 50 juveniles and 10 adults
for i in range(numsteps):
    pop[:,i+1] = U@pop[:,i] #propagate the population vector for one step
plt.plot(pop[0,:],label='juveniles')
plt.plot(pop[1,:],label='adults')
plt.xlabel('time')
plt.ylabel('Population')
plt.title('Leslie population model dynamics')
plt.legend()
plt.show()

plt.plot(pop[0,:]/pop[1,:])
plt.xlabel('time')
plt.ylabel('Population ratio')
plt.title('Ratio of juveniles to adults')
```

```
plt.show()
np.sqrt(3.24)
```






```
1.8
```

The juvenile/adult ratio converges to 2, as predicted by the leading eigenvector.

7 Linear ODEs with two variables

In chapter 3 we introduced and analyzed discrete-time models with multiple variables representing different demographic groups. In those models, the populations at the next time step depend on the population at the current time step in a linear fashion. More generally, any model with only linear dependencies can be represented in matrix form. In this chapter we will learn how to analyze the behavior of these models, and identify all possible classes of linear dynamical systems.

The main concept of this chapter are the special numbers and vectors associated with a matrix, called eigenvalues and eigenvectors. Any matrix can be thought of as an operator acting on vectors, and transforming them in certain ways. Loosely speaking, this transformation can be expressed in terms of special directions (eigenvectors) and special numbers that describe what happens along those special directions. Finding the eigenvalues and eigenvectors of a matrix allows us to understand the dynamics of biological models by classifying them into distinct categories.

In the modeling section, we will develop some intuition for modeling activators and inhibitors of biochemical reactions. We will then learn how to draw the flow of two-dimensional dynamical systems in the plane. In the analytical section, we will define eigenvectors and eigenvalues, and use this knowledge to find the general solution of linear multi-variable systems. In the computational section, numerical solutions of eigenvalues and eigenvectors will be applied to classifying all linear multi-dimensional systems, and to plotting the solutions, both over time and in the plane. Finally, in the synthesis section we will use a light-hearted model of relationship dynamics to illustrate how to analyze linear dynamical systems.

7.1 Flow in the phase plane

7.1.1 activators and inhibitors in biochemical reactions

Suppose two gene products (proteins) regulate each others' expression. Activator protein A binds to the promoter of the gene for I and activates its expression, while inhibitor protein I binds to the promoter of the gene for A and inhibits its expression (here the variables stand for concentrations of the two proteins in the cell):

$$\begin{aligned}\dot{A} &= -\alpha I \\ \dot{I} &= \beta A\end{aligned}$$

α and β are positive rate constants. They represent the rate of inhibition of A by I, and of activation of I by A, respectively. Let us now complicate the model by adding self-inhibition. It is common for regulatory proteins to inhibit their own production. Then, we have the following system of equations:

$$\begin{aligned}\dot{A} &= -\gamma A - \alpha I \\ \dot{I} &= \beta A - \delta I\end{aligned}$$

Here we have added two rates of self-inhibition γ and δ. This is a system of two coupled ODEs, and we will learn how to analyze these models both analytically and graphically.

7.1.2 phase plane portraits

Before we learn about the analytical tools of linear algebra, let us think intuitively about the effect of the variables on each other. The best way to describe this is through plotting the geometry of the *flow* prescribed by the differential equations. As we saw, for one-dimensional ODEs the direction of the change of the dependent variable (also known as the flow) could be shown as arrows on a line. A single variable can only increase, decrease, or stay the same (at a fixed point). In two dimensions there is more freedom. The flow is plotted on the *phase plane*, where for any combination of the two variables (say x, y) the ODE gives the derivatives of x and y. This vector gives the flow, or the rate of change at the particular point in the plane. Intuitively, the flow describes the direction in which the system is pulling the 2-dimensional solution. If we plot the progress of a solution of ODE (all the values of x and y starting with the initial condition) we will obtain a *trajectory* in the phase plane. The arrows of the flow are tangent to any trajectory curve, since they plot the derivatives of x, y.

Example: positive relationship between the variables Consider the following system of differential equations:

$$\begin{aligned}\dot{x} &= x + y \\ \dot{y} &= x + 2y\end{aligned}$$

(eq-ode1)

This is system is coupled, with x having an effect on y and vice versa. Specifically, the signs of the constants mean that positive values of x have the effect of increasing y (and vice versa), while negative values of x have the effect of decreasing y (and vice versa). For any pair of values of (x, y), there is a flow prescribed by the ODEs. E.g., when $x = 1, y = 1$, the derivatives

are $\dot{x} = 2, \dot{y} = 3$. This means that the flow at that point is given by the vector $(2, 3)$, and can be plotted in the x, y phase plane. This can be done for any pair of values of x and y, and plotted to give the phase plane portrait in figure {numref}`fig-ode1`.

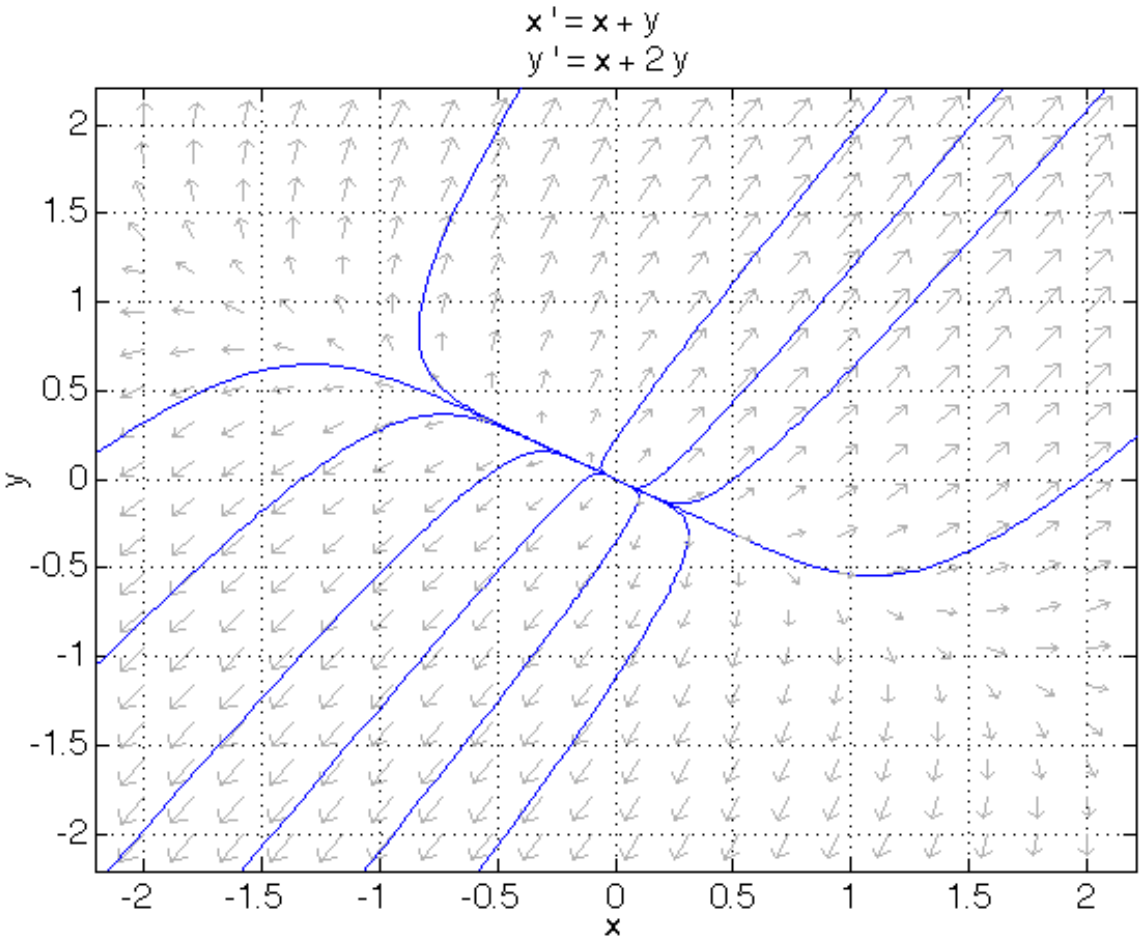


Figure 7.1: Phase plane flow for the system in {eq}`eq-ode1`

Observe that the overall dynamics of the systems are directed outward from the origin, as we expect from the ODEs. The blue lines on the plot are some sample trajectories. The solution over time for both x and y will either grow toward positive infinity, or decay to negative infinity.

Example: negative relationship between the variables Consider the following system of differential equations, where y has an effect on $\dot{x}$ opposite of its own sign. That is, negative values of y contribute to the growth of x, and vice versa.

$$\dot{x} = -y$$
$$\dot{y} = x$$

(eq-ode2)

As above, the flow at any one point is given by the ODEs. E.g. at $(0, 1)$ the two derivatives prescribe flow in the $(1, 0)$ (up) direction, while at $(1, 0)$ the flow is in the $(0, -1)$ direction. Figure {numref}`fig-ode2` shows the arrows of flow in the phase plane around the origin. Note that the arrows go around in a circular pattern around the origin - this shows oscillatory flow of solutions.

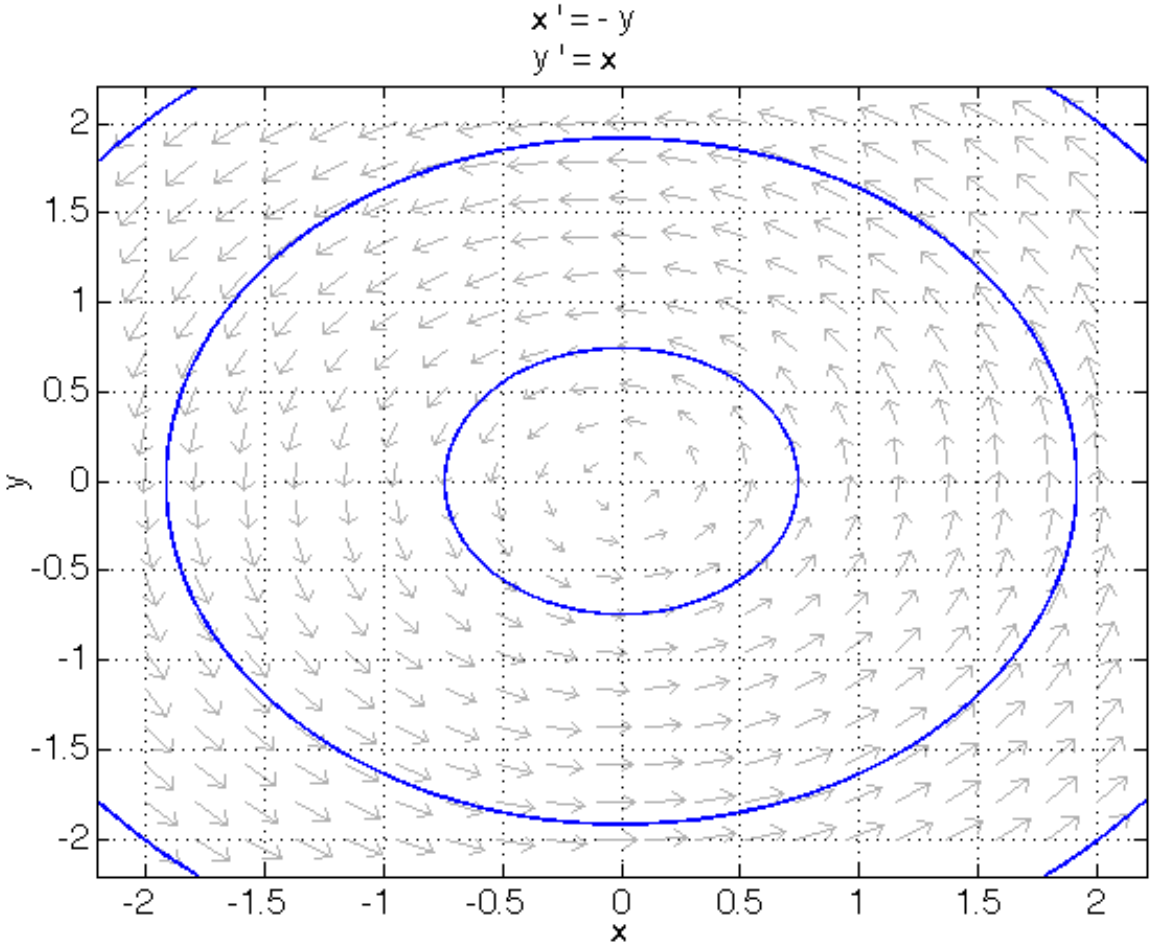


Figure 7.2: Phase plane flow for the system in {eq}`eq-ode2`

Let us consider the trajectories of x and y in time. The blue curves in the phase plane plot demonstrate the solutions go around the origin and return to the same point. This means

that the behavior of the solutions over time is *periodic*, with oscillations going from positive to negative numbers and back forever.

7.2 Solutions of linear two-variable ODEs

Let us start by considering two variable ODEs that do not affect each other:

Example: two uncoupled ODEs In general, for a two-variable system, the value of one variable affects the other. In the equations above, the terms with the constants b and c provide what is known as *coupling* between the two variables. Let us look at the primitive situation where the two variables are uncoupled, as an illustration of solving two-dimensional ODEs. If we set the coupling constants b and c to 0, we get:

$$\begin{aligned} \dot{x} = &\quad ax \\ \dot{y} = &\quad dy \end{aligned}$$

Using our knowledge of 1D linear ODE, we can solve the two equations independently to get the following: $x(t) = x_0 e^{at}$ and $y(t) = y_0 e^{dt}$. The solutions can be written in vector form:

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = x_0 e^{at} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + y_0 e^{dt} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

This is another way of writing that the dynamics of variable x is exponential growth (or decay) with rate a, and ditto y, with rate d. Given the initial conditions (x_0, y_0), we can divide the behavior of the solutions into a sum of two vectors, each growing or decaying at its own rate.

Linear algebra allows us to find the solution for two-dimensional ODEs where the variables are interdependent using the same idea. The general (homogeneous) ODE with two dependent variables can be written as follows:

$$\begin{aligned} \dot{x} = &\quad ax + by \\ \dot{y} = &\quad cx + dy \end{aligned}$$

We can write this in matrix form like this:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Let us call the matrix A, and represent the vector (x, y) as $\vec{x}$, then the general linear equation can be written like this:

$$\dot{\vec{x}} = A\vec{x}$$

(gen-lin-mult)

This notation is intended to make plain the similarity with the linear 1D ODE: $\dot{x} = ax$. This similarity is deep and substantial, in that linear equations in multiple dimensions share the same basic exponential form. In general, all solutions of linear equations can be written as a sum of exponentials multiplying different vectors:

General solution of linear 2-variable ODEs

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = C_1 e^{\lambda_1 t} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + C_2 e^{\lambda_2 t} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$$

(gen-sol-2var)
The constants C_1, C_2 are determined by the initial conditions, while the constants λ_1, λ_2 are the eigenvalues and the vectors (x_1, y_1) and (x_2, y_2) are the eigenvectors of the matrix A. We will now consider the application of this general result using computational tools.

7.3 Classification of linear systems

We have seen that linear algebra allows us to write down the solution of a multivariable dynamical system into a sum of exponential terms. In this section we use computational techniques to find the eigenvalues and eigenvectors of a system, and then produce the *phase portraits* of the linear systems. There are only a few different types of flow possible for linear systems, and we will classify them.

7.3.1 real eigenvalues

Let us consider the fixed points of the linear system: since both $\dot{x} = 0$ and $\dot{y} = 0$ must be zero, the only fixed point is the origin $(0, 0)$. We will see that the stability of the fixed point depends on the sign of the real part of the eigenvalue.

Suppose we have a positive real eigenvalue. The solution in the direction of the corresponding eigenvector is then described by $Ce^{\lambda t}$, $\lambda > 0$, which is exponential growth. The means that the solution is going to grow in the direction of the eigenvector away from the origin, and thus the origin is an unstable fixed point (in this direction). This type of fixed point is called an *unstable node*.

On the other hand, if $\lambda < 0$ for both eigenvalues, the solution decays exponentially and thus approaches the origin, so the fixed point is stable. This type of fixed point is called a *stable node*.

Since there are two different eigenvalues, one may be positive while another is negative. In this case, the fixed point is is called a *saddle point* for geometric reasons: solutions flow toward it in one direction, like a marble along the forward-backward axis of a saddle on a horse and flow away from it along the sideways direction on a saddle. Then, the fixed point is stable when approached along one eigenvector, but unstable along the other. What happens if the initial condition is not on either eigenvector? I will use a fact of linear algebra that given any two (non-colinear) 2D vectors, any vector in the plane can represented as a sum (with some coefficients) of these two. Thus, the general solution can be written as follows:

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = C_1 e^{at} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} + C_2 e^{-bt} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

where $a, b > 0$. Then we see that the component in the direction of the first eigenvector will grow, while the component along the second eigenvector will decay. Thus, as $t \to \infty$, all solutions will approach the vector with the unstable eigenvalue, except those with initial conditions right on the eigenvector corresponding to the stable eigenvalue. This means that the fixed point is essentially unstable, because only trajectories which start exactly along the stable direction approach the fixed point in the long run, while others, may approach the fixed point for a finite time, flow away when the unstable component with the positive eigenvalue takes over, as shown in figure .

[Phase plane flow for a linear system with a saddle point] (images/week6_pp1.png)

7.3.2 complex eigenvalues

If the argument of the square root is negative, eigenvalues may be complex numbers, which we can write like this: $a + bi$. Using Euler's formula, we can write down the time-dependent part of the solutions as the following:

$$e^{(a+bi)t} = e^{at} e^{bit} = e^{at}(\cos(bt) + i \sin(bt))$$

The behavior of these solutions combines exponential growth or decay from the real part, with the oscillations produced by the imaginary part. This describes either exponentially growing or decaying oscillations, which look like decaying waves in time, or as a spiral in the phase plane:

Thus we see that the stability of the fixed point with complex eigenvalues depends on the sign of the real part. Purely imaginary eigenvalues produce periodic oscillations, which keep the same amplitude, as we saw in the example in the modeling section.

7.3.3 classification of linear systems

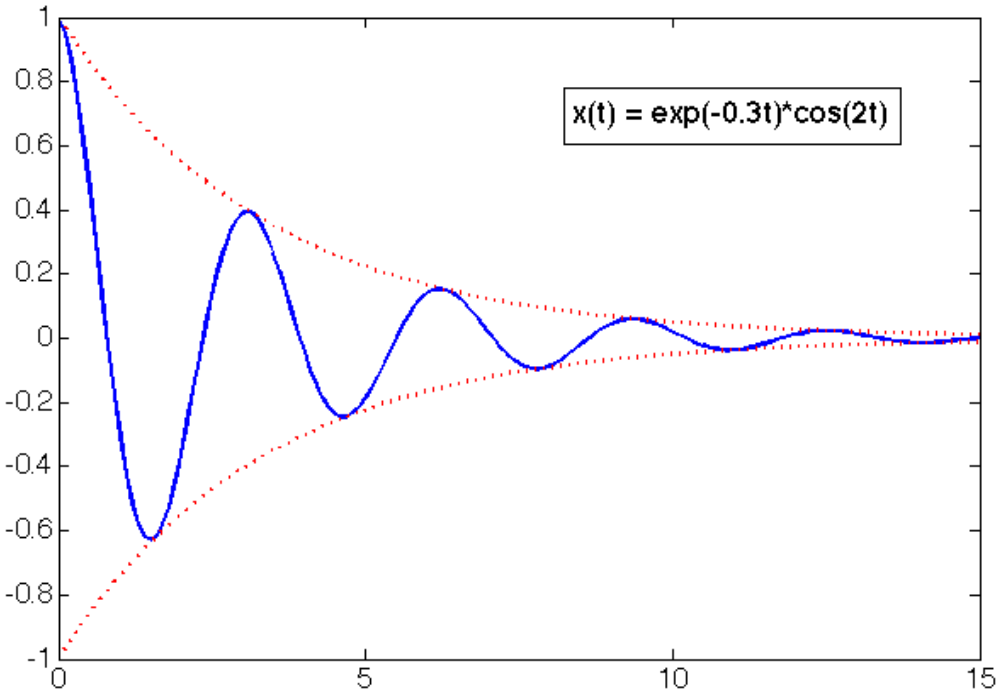


Figure 7.3: Exponentially decaying oscillations in the time plot of the solution $x(t)$

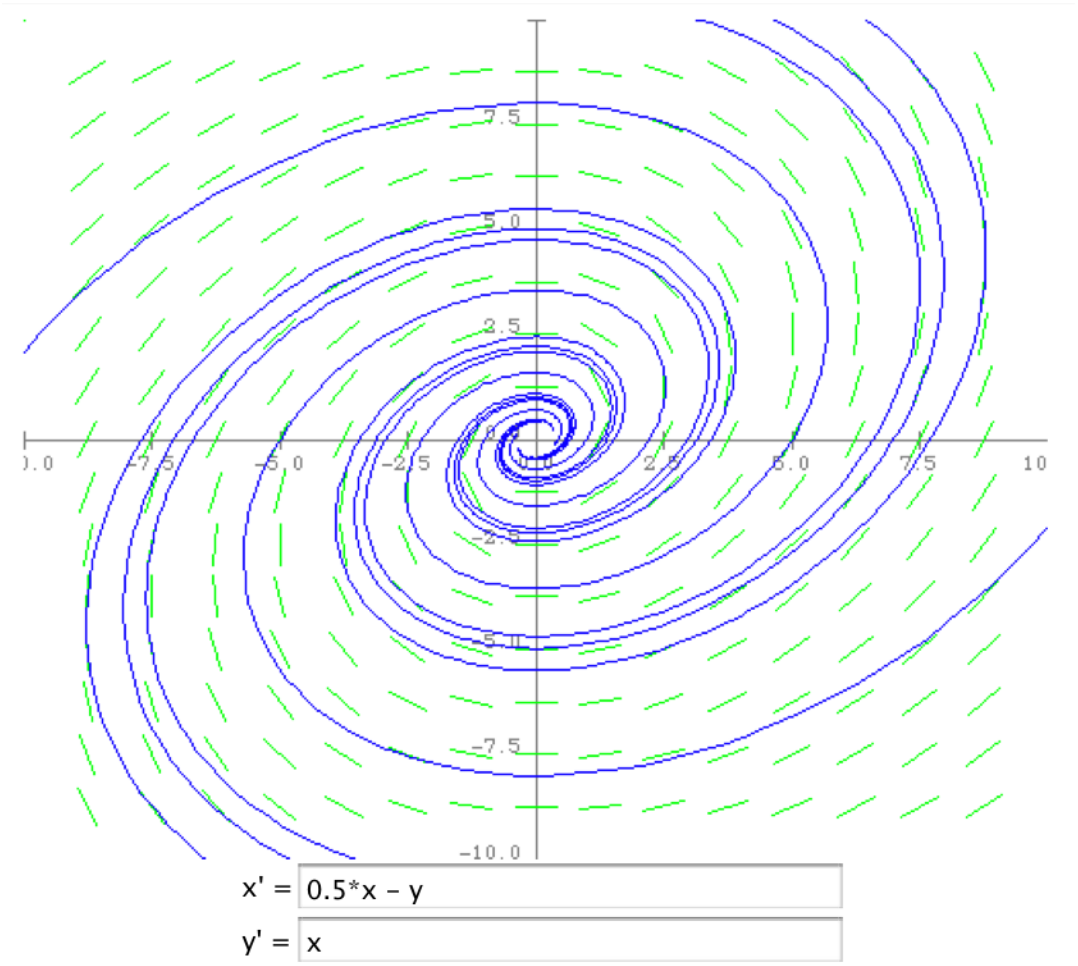


Figure 7.4: Phase plane portrait of a stable spiral

Stability	positive real parts	negative real parts	one positive, one negative	zero real part
real:	unstable node	stable node	saddle point	fixed line
complex:	unstable spiral	stable spiral	N/A	center point

Eigenvalues of linear ODEs define type of phase plane

The table above summarizes all the different types of flows in the phase plane possible for linear systems, in terms of the behavior of solutions relative to the fixed point at the origin. If the eigenvalues are real, the solutions will be exponential in nature. There are three possibilities for nonzero eigenvalues: *stable node* (both eigenvalues are negative), *unstable node* (both eigenvalues are positive), and a *saddle point* (mixed signs). If one of the eigenvalues is zero, this means that there is not flow along one direction, so there is a *line of fixed points* in the direction of the corresponding eigenvector (if both eigenvalues are zero, there is no flow at all.)

For complex eigenvalues, there are three possibilities: if the real part is positive, the solution will grow and oscillate (oscillations with exponentially increasing amplitude), if the real part is negative, the solution will decay and oscillation (oscillations with exponentially increasing amplitude), and if the real part is zero (pure imaginary eigenvalues) the solution will oscillate with constant amplitude. The first type is called an *unstable spiral*, the second a *stable spiral*, and the third a *center*. It is not possible for complex eigenvalues of two-dimensional systems to have different signs of real parts, because as the formula shows, the real part is the same for both and is equal to the trace divided by two.

7.4 Dynamics of romantic relationships

We examine a model, taken from (**strogatz_nonlinear_2001?**), that applies dynamical systems modeling to a pressing concern for many humans: the prediction of dynamics of a romantic relationship. There are several unrealistic assumptions involved in the following model: first, that love or affection can be quantified, second, that any changes in relationship depend only on the emotions of the two people involved, and third, that the rate of change of the two love variables depend linearly on each other.

If we can give those assumptions the benefit of the doubt (which is how all relationships begin), we can write down a system of ODEs to describe a romantically involved couple. Here X and Y are dynamic variables that quantify the emotional states of the two lovers:

$$\begin{aligned} \dot{X} &= aX + bY \\ \dot{Y} &= cX + dY \end{aligned}$$

Let us denote positive feelings (love) with positive values of X, Y, while negative values signify negative feelings (hate.) The significance of the parameters can be interpreted as follows: a, d describe the response of the two people to their own feelings, while b, c correspond to the effect the other person's feeling has on their own. For example, a person whose feeling grow as the other person's affection increases can be modeled with a positive value of b (or c). On the other hand, a person whose own feelings are dampened by the other one's excessively positive emotions, can be decribed by a negative value of b or c. Their own feelings can also play a role, either positive or negative, reflected in the sign of the constants a and d.

Using mathematical modeling, we can answer the following basic questions:

1. Given a set of values for parameters a, b, c, d, predict the dynamic behavior of the model relationship.
2. Find conditions for stability and existence of oscillations in the dynamical system, expressed as a function of the parameters.

To address the first question, here are some simplified scenarios for our two lovers in the model.

Detached lovers: Let the emotional state of the two lovers depend only on their own emotions, for example:

$$\begin{aligned} \dot{X} &= \quad X \\ \dot{Y} &= -Y \end{aligned}$$

To classify the behavior of the model, we find the eigenvalues of the system. In this case, they are the diagonal elements of the matrix, 1 and -1. This mean that the origin is a saddle point, and therefore it is unstable. In the X direction, the emotions are going to grow without bound, either in the love or hate direction, while in the Y direction, the emotions are going to decay to zero (indifference). This should be no surprise, that since the two equations are independent, the lovers have no emotional effect on each other.

Lovers with no self-awareness: Here is an alternate situation: suppose two lovers were not influenced by their own emotions, but were instead attuned to the emotional state of the other. Then we might have the following model, in which lover X reacts in the opposite way by emotions of lover Y, but lover Y is, contrariwise, spurred by the love or hate of X in the same direction:

$$\begin{aligned} \dot{X} &= -Y \\ \dot{Y} &= \quad X \end{aligned}$$

We find the eigenvalues of the system by using the expression in equation [eg:2D_eig]: $\lambda = (0 \pm \sqrt{-4})/2 = \pm i$. Pure imaginary eigenvalues tell us that the origin is a center point, with the

solutions periodic orbits around the origin. Psychologically, we can interpret this scenario as cycles of love and hate, never growing and never decaying. The magnitude of these oscillations depends on the initial state of the system, that is, the feelings the lovers had at the beginning of the relationship.

We can now address the second question, and find under what circumstances different types of dynamic behaviors occur. We consider the general model, and ask what kinds of eigenvalues are possible for different parameter values. First, we write down the general expression for the eigenvalues, from equation [eg:2D_eig]:

$$\lambda = \frac{a + d \pm \sqrt{(a+d)^2 - 4(ad - bc)}}{2}$$

There are two properties we are interested in: stability and existence of oscillations. Recall that stability is determined by the sign of the real part of the eigenvalues. If the square root is imaginary, then the real part is simply the trace $(a + d)$, but if the square root is real, we have to consider the whole expression to determine stability. So let us first state the condition for existence of oscillations (imaginary square root):

1. Complex eigenvalues: oscillatory solutions $4(ad-bc) > (a+d)^2$. If this expression holds, the square root is imaginary, and the stability is determined by the sign of the trace. That is, if $a + d > 0$, the system is unstable, and will grow into unbounded love or hate, but if $a + d < 0$, then the system is stable, and will spiral to indifference. The special case $a + d = 0$, such as we saw above, means that strictly periodic love/hate cycles are the solutions.

2. Real eigenvalues: exponential growth and/or decay $4(ad - bc) < (a + d)^2$. In this case, the square root is real, and no oscillatory solutions exist. In order to determine whether this implies exponential growth, decay, or a combination, we must weigh the relative sizes of $(a + d)$ and $\sqrt{(a+d)^2 - 4(ad - bc)}$. If $|a + d| > \sqrt{(a+d)^2 - 4(ad - bc)}$, then adding or subtracting the square root does not change the sign of $(a+d)$: if it is negative, both eigenvalues are negative, and the origin is a stable node, and if the trace is positive, the origin is an unstable node. However, if the absolute value of the root outweighs the absolute value of the trace $|a + d| < \sqrt{(a+d)^2 - 4(ad - bc)}$, then either adding or subtracting the root will change the sign of the eigenvalues. Therefore, one eigenvalue is positive and the other is negative, and the origin is a saddle point. The emotions will run unchecked in some preferred direction, possibly combining love and hate of the two lovers.

These conditions are not intuitive, and it took some work to express them. The benefit is that now, given any values of the self-involvement parameters a, d and the sensitivity parameters b, c we can predict the long-term dynamics of the model relationship. Whether the results have any bearing on reality, of course, depends on how well the reality is described by these primitive assumptions.

8 Phase portraits in Python

```
#Necessary imports
import numpy as np #package for work with arrays and matrices
import matplotlib.pyplot as plt #package with plotting capabilities
from scipy.integrate import odeint
```

8.0.1 phase plane plots via quiver

We are going to plot phase diagrams for linear ODEs that have the form

$$\begin{aligned} dx/dt = &\quad ax + by \\ dy/dt = &\quad cx + dy \end{aligned}$$

Python's `ax.quiver()` function allows displays vectors with arrows made of the components (u, v), which is exactly what we need. The function takes 4 inputs (x, y, u, v): x and y are the grid points and u and v are the u and v components of the vector, which are given by our ODEs.

In order to make the grid points (x, y), we will use the function `np.meshgrid()`. It's a pretty handy function that takes as input a range of x and y values and returns two matrices x, y that together give us the grid points. Here is the code to produce a grid with an x and y range from (-1.5, 1.5) with a spacing of 0.2, we could do the following:

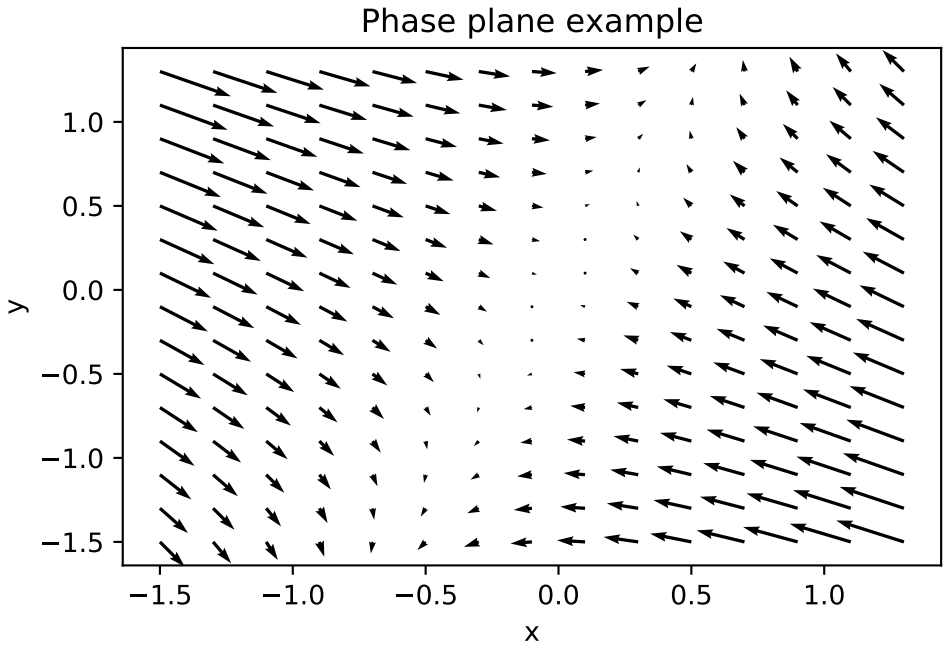
```
xmin = -1.5 #change the parameters here to control the range of the axes
xmax = 1.5
ymin = -1.5
ymax = 1.5
dx = 0.2 #set the size of the x-step on the grid
dy = 0.2 #set the size of the y-step on the grid
X = np.arange(xmin, xmax, dx)
Y = np.arange(ymin, ymax, dy)
x, y = np.meshgrid(X,Y) #create a grid
```

Define the arrays dx and dy based on the ODE in order to compute the flow vectors on that grid. Here is a linear example:

```
a = -2
b = 1
c = 1
d = 0
dx = a*x+b*y #overwrites the other dx
dy = c*x+d*y #overwrites the other dy
```

Then plot the arrows given by arrays dx,dy at points x,y:

```
fig, ax = plt.subplots()
q = ax.quiver(x, y, dx, dy)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Phase plane example')
plt.show()
```



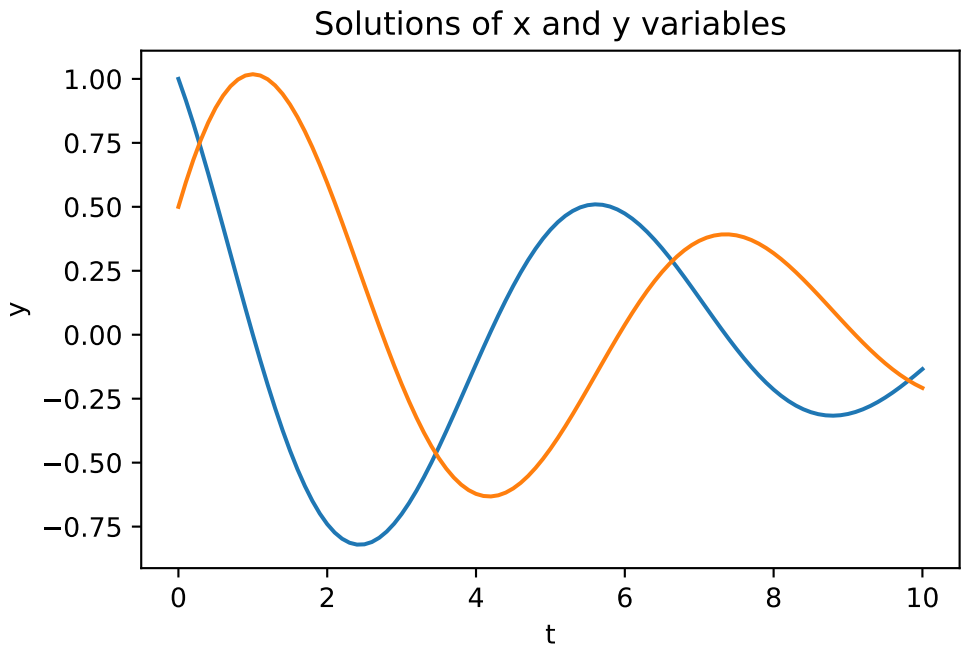
8.0.2 ODE solutions using odeint

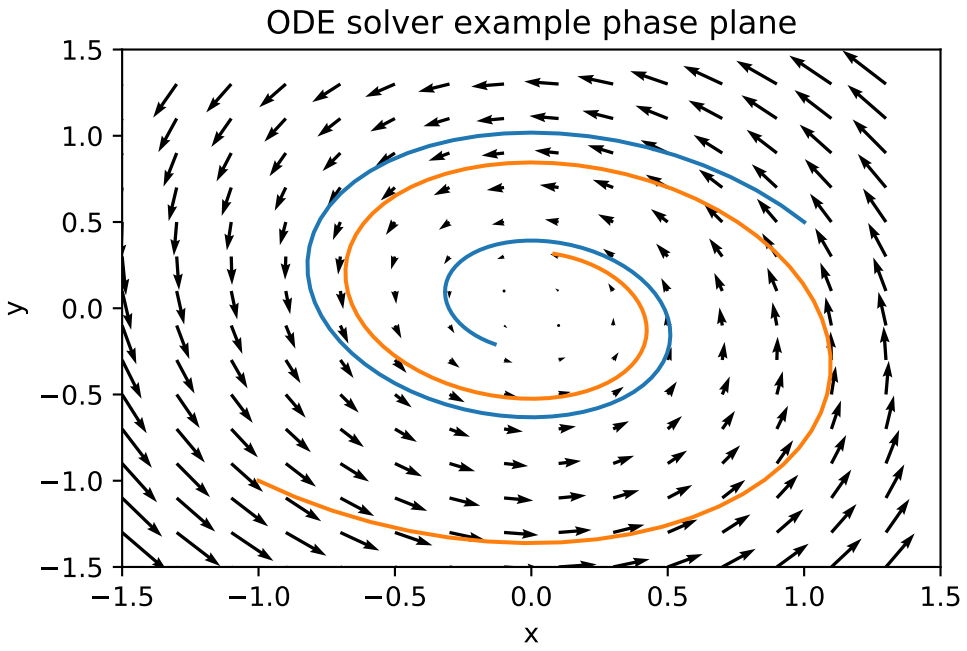
Python has an entire suite of ode solvers. We'll use the function `odeint`, with documentation provided here: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

This requires defining a function `fun` that sets the two functions for the two-variable ODE, to be called by the `odeint`, together with parameter values a, b, c, d. There are many other options that you can read about in the documentation page. Here is the sample code:

```
xmin = -1.5 #change the parameters here to control the range of the axes
xmax = 1.5
ymin = -1.5
ymax = 1.5
dx = 0.2 #set the size of the x-step on the grid
dy = 0.2 #set the size of the y-step on the grid
X = np.arange(xmin, xmax, dx)
Y = np.arange(ymin, ymax, dy)
x, y = np.meshgrid(X, Y);  #create a grid

a = -0.3
b = -1
c = 1
d = 0

dx = a*x+b*y
dy = c*x+d*y

# define the function for the ODES: note the order of inputs
def fun(xy, t, a, b, c, d):  # inputs are: variable array, time, any parameters
    newxy = [a*xy[0]+b*xy[1], c*xy[0]+d*xy[1]]
    return newxy

# Set the initial values, the vector of times, and call the ODE solver
init = [1, 0.5] #[intial x, initial y]
t = np.linspace(0, 10, 101) # create time vector
sol = odeint(fun, init, t, args=(a, b, c, d)) # calculate numeric solution of ODE defined

# Plot the solutions over time
plt.plot(t, sol)
plt.xlabel('t')
plt.ylabel('y')
```

```
plt.title('Solutions of x and y variables')
plt.show()


# plot the arrows given by arrays dx,dy at points x,y:
fig, ax = plt.subplots()
q = ax.quiver(x, y, dx, dy)
plt.xlim(-1.5,1.5)
plt.ylim(-1.5,1.5)
ax.plot(sol[:,0], sol[:,1]) # plot the x and the y variable in the phase plane

# Set different initial values, the vector of times, and call the ODE solver again
init = [-1, -1] #[intial x, initial y]
t = np.linspace(0, 10, 101) # create time vector
sol = odeint(fun, init, t, args=(a, b, c, d)) # calculate numeric solution of ODE defined

ax.plot(sol[:,0], sol[:,1]) # plot the x and the y variable in the phase plane
plt.xlabel('x') #use more informative labels for a real model
plt.ylabel('y')
plt.title('ODE solver example phase plane')
plt.show()
```




ODE solver example phase plane
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
y
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x

References