Chapter 15

Linear Algebra

Linear algebra is one of the most important subjects in mathematics, with applications found in every branch of science. The techniques of linear algebra also form the basis of many numerical algorithms. In this chapter we will translate linear algebra problems into matrix notation and solve them using NumPy commands.

15.1 Matrix multiplication

Here is a quick review of matrix multiplication. Consider the 2×2 matrices A and B defined by

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \ , \quad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \ .$$

The following equation depicts the calculation of the product AB:

$$\begin{pmatrix} \boxed{1 \quad 2} \\ 3 \quad 4 \end{pmatrix} \begin{pmatrix} \boxed{\begin{matrix} 5 \\ 7 \end{matrix}} & \begin{matrix} 6 \\ 8 \end{matrix} \end{pmatrix} = \begin{pmatrix} \boxed{1 \cdot 5 + 2 \cdot 7} & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{pmatrix} = \begin{pmatrix} \boxed{19} & 22 \\ 43 & 50 \end{pmatrix}$$

The entry in the first row and first column of AB is the dot product of the first row of A and the first column of B. Likewise, the entry in the first row and second column of AB is the dot product of the first row of A and the second column of B. In general, the entry in the nth row and mth column of AB is the dot product of the nth row of A and the mth column of B. The same pattern holds for multiplication of any two matrices. The only restriction is that the number of entries in the rows of A must match the number of entries in the columns of B.

15.2 Linear systems

A basic problem in linear algebra is the solution of n linear algebraic equations for n unknowns. Consider the two equations

$$x + 3y = -3 \; , \tag{15.1a}$$

$$2x - 2y = 10 \; , \tag{15.1b}$$

for the two unknowns x and y. You can verify that the solution is $x = 3$, $y = -2$.

The problem above can be written in matrix notation as

$$\begin{pmatrix} 1 & 3 \\ 2 & -2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -3 \\ 10 \end{pmatrix} \; . \tag{15.2}$$

More compactly, we write this as $Av = r$ where

$$A = \begin{pmatrix} 1 & 3 \\ 2 & -2 \end{pmatrix} \; , \quad v = \begin{pmatrix} x \\ y \end{pmatrix} \; , \quad r = \begin{pmatrix} -3 \\ 10 \end{pmatrix} \; . \tag{15.3}$$

A is called the coefficient matrix. The column vector v contains the unknowns x and y. The column vector r is the right–hand side of the equation.

The problem of solving $Av = r$ for the unknowns v can now be viewed as the problem of inverting the matrix A. Let A^{-1} denote the inverse of A.

Exercise 15.2a

Verify that

$$A^{-1} = \begin{pmatrix} 1/4 & 3/8 \\ 1/4 & -1/8 \end{pmatrix}$$

is the inverse of A. That is, use matrix multiplication to show that $A^{-1}A = I$ and $AA^{-1} = I$, where

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

is the identity matrix.

Having found the inverse of A, we can now solve our problem. Multiply the equation $Av = r$ by A^{-1} to obtain $Iv = A^{-1}r$. You can verify that $Iv = v$; that's why I is called the "identity matrix." Then the solution for the unknowns x and y is $v = A^{-1}r$. Explicitly, this is

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1/4 & 3/8 \\ 1/4 & -1/8 \end{pmatrix} \begin{pmatrix} -3 \\ 10 \end{pmatrix} \; . \tag{15.4}$$

Carrying out the multiplication, we find

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ -2 \end{pmatrix} . \tag{15.5}$$

Thus, $x = 3$ and $y = -2$.

This problem is easily solved using the `matrix` data type defined by NumPy. Let's assume NumPy is loaded as `np`. The matrix A is defined by

```
A = np.matrix([[1,3],[2,-2]])
```

Note the syntax. In general, NumPy matrices are defined by

`np.matrix` ([[*first row*] , [*second row*] , [*third row*]])

The argument of `matrix()` is contained in round parentheses (). The matrix itself uses two sets of nested square brackets. The outer set of brackets contains the list of rows. Each inner set of brackets contains a list of elements for a row.

The inverse A^{-1} can be computed using

```
Ainv = np.linalg.inv(A)
```

The function `inv()` is contained in the linear algebra subpackage `linalg` within NumPy. The inverse of A can also be computed using the abbreviated notation `Ainv = A.I`.

Now define the right–hand side:

```
r = np.matrix([[-3],[10]])
```

We might refer to r as a column vector, but note that it is defined as a NumPy matrix with two rows and one column. What's the difference between `np.matrix([[-3,10]])` and `np.matrix([[-3],[10]])`?

Matrix multiplication is denoted by an asterisk $*$. Thus, the answer to our problem, $v = A^{-1}r$, is computed as

```
v = Ainv*r
print(v)
```

This yields the expected result,

```
matrix([[3],
        [-2]])
```

That is, $x = 3$ and $y = -2$.

Although the solution can be viewed as a column vector, $v = \binom{3}{-2}$, the calculation `v = Ainv*r` produces a NumPy matrix with two rows and one column. Thus, if we want to access the first element of v, which is the value of x, we must type `v[0,0]`. The second element, the value of y, is `v[1,0]`.

Exercise 15.2b

Translate the linear system

$$\begin{aligned} -2x + 7y + z &= 3 \\ 4x - 5y + 3z &= -2 \\ 2x - y + 4z &= 6 \end{aligned}$$

into the form $Av = r$. Solve for x, y and z by inverting the coefficient matrix A and computing $v = A^{-1}r$.

15.3 The `linalg.solve()` function

In general, solving a system of linear equations by inverting the coefficient matrix is a bad idea. Finding the numerical inverse of a matrix is:

- Time consuming. Other methods of solving linear systems are faster.
- Inaccurate. Inverting a matrix can lead to large machine roundoff errors.

As an example of this second point, consider the matrix

$$A = \begin{pmatrix} 4001 & 2001 \\ 8000 & 4001 \end{pmatrix} . \tag{15.6}$$

This is an "ill–conditioned" matrix that is difficult to invert numerically.

Exercise 15.3a

Use `linalg.inv()` to compute the inverse of the matrix A from Eq. (15.6), then compute $A^{-1}A$. Compare this to the identity matrix. How large are the errors? Carry out the same calculation for the matrix

$$B = \begin{pmatrix} 3001 & 2001 \\ 8000 & 4001 \end{pmatrix} ,$$

which is not ill–conditioned. How large are the errors in $B^{-1}B$?

Fortunately, linear algebra is a mature subject and the numerical routines available in the `numpy.linalg` library are well developed and reliable. In particular, we can use the `solve()` function to solve the linear system $Av = r$:

```
v = np.linalg.solve(A,r)
```

The algorithm used by `solve()` can be faster and more accurate than explicit matrix inversion.

Exercise 15.3b

Solve the linear system

$$\begin{aligned} 1.2x - 3.6y + 4.5z &= 1.3 \ , \\ -3.3x + 4.2y - 8.1z &= -2.5 \ , \\ -0.9x - 3.1y + 0.9z &= 0.1 \ , \end{aligned}$$

in two different ways. First, use `linalg.inv()` to find the inverse of the coefficient matrix A, and compute $v = A^{-1}r$. Second, use `linalg.solve()`. In each case, have your code check the result by comparing $Av - r$ to the zero vector.

From now on, you should use `solve()` rather than matrix inversion to solve linear systems of equations.

15.4 Statics

A rigid, uniform beam of length ℓ and mass m is connected to a wall at one end and supported by a cord attached to the other end. See Fig. 15.1. The cord makes an angle θ with the beam, and has tension T. The wall exerts a force on the beam with components F_x, F_y. Since the beam is in

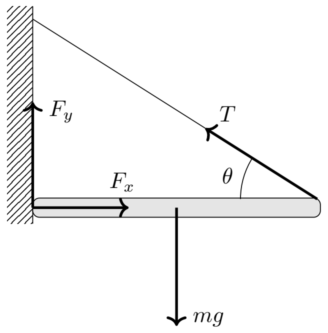

Fig. 15.1: A static beam of mass m and length ℓ is connected to a wall and a cord.

static equilibrium Newton's laws of motion tell us that the sum of forces must vanish, $\sum \vec{F} = 0$. The forces lie in the x–y plane (the plane of the diagram), so $\sum \vec{F} = 0$ yields two equations:

$$F_x - T\cos\theta = 0 \ , \tag{15.8a}$$

$$F_y + T\sin\theta - mg = 0 \ . \tag{15.8b}$$

Likewise, the sum of torques about any point $\mathcal{P}$ must vanish. This reduces to a single equation for the torque about the z–axis,

$$(T\sin\theta)(\ell/2) - F_y(\ell/2) = 0 \ , \tag{15.9}$$

where we have used the middle of the beam as the point $\mathcal{P}$.

These three equations can be written in matrix form $Av = r$:

$$\begin{pmatrix} 1 & 0 & -\cos\theta \\ 0 & 1 & \sin\theta \\ 0 & -\ell/2 & \ell\sin\theta/2 \end{pmatrix} \begin{pmatrix} F_x \\ F_y \\ T \end{pmatrix} = \begin{pmatrix} 0 \\ mg \\ 0 \end{pmatrix} \tag{15.10}$$

Assuming mg, ℓ and θ are known, we can solve this equation for the unknown forces F_x, F_y and T.

Exercise 15.4a

Solve Eq. (15.10) numerically for a beam of mass $m = 3.5\,\mathrm{kg}$ and length $\ell = 1.2\,\mathrm{m}$, and an array of cord angles θ. (The acceleration due to gravity is $g = 9.8\,\mathrm{m/s^2}$.) Have your code plot a graph of tension T versus angle θ.

Exercise 15.4b

The beam in Fig. 15.2 has length ℓ, mass m, and makes an angle θ with respect to the floor. The cord is horizontal with tension T. Derive the equations of static equilibrium for the beam. Write your equations in matrix form $Av = r$ where the unknowns are the tension in the cord and the components of force that the floor exerts on the beam. Solve the matrix equation numerically using reasonable values for mg, ℓ and θ.

15.5 Kirchoff's laws

Kirchoff's laws describe electrical circuits with steady currents. Figure 15.3 shows a simple electrical circuit consisting of two batteries, $\mathcal{E}_1$ and $\mathcal{E}_2$, and

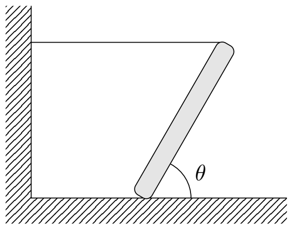

Fig. 15.2: A static beam with one end resting on the floor, the other end connected to a horizontal cord. The force that the floor exerts on the beam, the downward force due to gravity, and the tension in the cord are not shown.

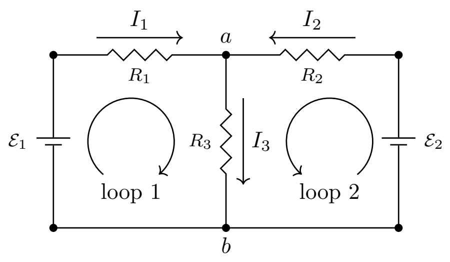

Fig. 15.3: A simple circuit with two batteries and three resistors. Letters a and b label nodes.

three resistors, R_1, R_2 and R_3. Currents I_1, I_2 and I_3 flow through the three sections of the circuit.

Kirchoff's first law states that the current entering a node is equal to the current exiting a node. At node a, the current entering is $I_1 + I_2$ and the current exiting is I_3; therefore,

$$I_1 + I_2 = I_3 \ . \tag{15.11}$$

Kirchoff's first law applied to node b yields the same relation.

Kirchoff's second law states that the sum of voltages around any closed loop must vanish. Recall that when a current I passes through a resistor R, the voltage drops by an amount IR. Kirchoff's second law applied to loop 1 of Fig. 15.3 gives

$$\mathcal{E}_1 - I_1 R_1 - I_3 R_3 = 0 \ . \tag{15.12a}$$

For loop 2, we have

$$\mathcal{E}_2 - I_2R_2 - I_3R_3 = 0 \; . \tag{15.12b}$$

We can apply Kirchoff's second law to the large loop around the outer edge of the circuit. The result is just a linear combination of the previous two equations.

Exercise 15.5a

Express Eqs. (15.11), (15.12) in matrix form, with the currents I_1, I_2, and I_3 as unknowns. Write a code to solve for the currents using the data $\mathcal{E}_1 = 12\,\mathrm{V}$, $\mathcal{E}_2 = 9\,\mathrm{V}$, $R_1 = 100\,\Omega$, $R_2 = 120\,\Omega$, $R_3 = 65\,\Omega$.

Exercise 15.5b

Use Kirchoff's laws to analyze the circuit in Fig. 15.4. The 5 resistors are identical, and the 3 batteries are identical. Set $R = 100\,\Omega$ and $\mathcal{E} = 9\,\mathrm{V}$ and solve numerically for the current in each part of the circuit. (Hint: there are 6 currents. Use 3 Kirchoff first laws and 3 Kirchoff second laws.)

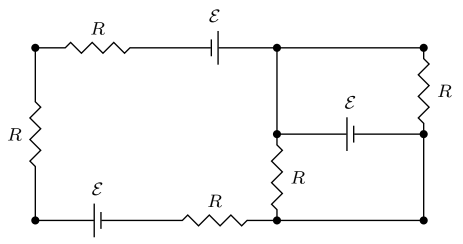

Fig. 15.4: Circuit for Exercise 15.5b.

15.6 Eigenvalues and eigenvectors

Another important class of problems in linear algebra is the eigenvalue–eigenvector problem. Given a matrix M, the goal of this problem is to find numbers λ and column vectors v such that

$$Mv = \lambda v \; . \tag{15.13}$$

The numbers λ are the *eigenvalues*, and v are the *eigenvectors*.

The eigenvalue–eigenvector equation can be written as $Mv = \lambda Iv$ where I is the identity matrix. Bringing both terms to the left–hand side, we have $(M - \lambda I)v = 0$. This equation has the form $Av = 0$ where the matrix A is defined by $A \equiv M - \lambda I$. We already know how to solve such a system: simply invert the matrix A and multiply both sides by A^{-1}. (Better yet, use the `linalg.solve()` function in NumPy.) This calculation yields the trivial solution $v = A^{-1}0 = 0$. Of course $v = 0$ is always a solution to our original problem, $Mv = \lambda v$. In fact, $v = 0$ is the only solution if the matrix $A \equiv M - \lambda I$ is invertible. But for certain special values of λ, the matrix $M - \lambda I$ will *not* be invertible. In those cases the system $Mv = \lambda v$ will have nontrivial solutions for the eigenvectors v.

Our first step in solving the eigenvalue–eigenvector problem consists in finding the eigenvalues λ. That is, we look for values of λ such that $M - \lambda I$ is not invertible.[1] Then for each eigenvalue we find a corresponding eigenvector v that satisfies $Mv = \lambda v$. Typically, for an $n \times n$ matrix M, the eigenvalue–eigenvector equation will have n solutions. That is, there will be n eigenvalues λ, and for each eigenvalue there will be a corresponding eigenvector v.

In Python, you can use the NumPy function `linalg.eig()` to solve the eigenvalue–eigenvector problem. More precisely, the command

```
eval,evec = np.linalg.eig(M)
```

will compute the eigenvalues and corresponding eigenvectors of a matrix M. The eigenvalues are placed into a NumPy array, called `eval` in this example. The eigenvectors are placed into a NumPy matrix, called `evec` in this example. The first column, `evec[:,0]`, is the eigenvector corresponding to the first eigenvalue, `eval[0]`; the second column, `evec[:,1]`, is the eigenvector corresponding to the second eigenvalue, `eval[1]`; *etc.* A colon `:` in place of a matrix index means "all elements."

Exercise 15.6

Write a code that uses `linalg.eig()` to solve the eigenvalue–eigenvector problem for

$$M = \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix} .$$

[1] A matrix A is not invertible if its determinant $\det(A)$ vanishes. Thus, the eigenvalues satisfy $\det(M - \lambda I) = 0$.

There are two eigenvalues and two corresponding eigenvectors. Verify that $Mv = \lambda v$ is satisfied for each.

Note that the eigenvector associated with a given eigenvalue is not unique. If v satisfies the equation $Mv = \lambda v$, then so does any vector proportional to v. The eigenvectors returned by the `linalg.eig()` function are normalized so that the sum of the squares of the components is unity.

15.7 Normal mode analysis

Two masses move in one dimension along a frictionless table, as shown in Fig. (15.5). The masses are connected between fixed walls by three springs. The separation between walls is D. Each spring has stiffness k and relaxed

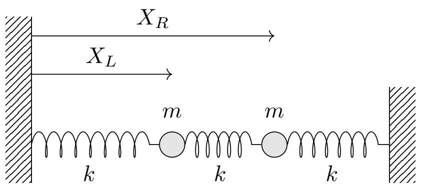

Fig. 15.5: Two masses connected to walls by three springs. The distance between walls is D.

length ℓ_0. The force that a spring exerts on the objects attached to its ends is $\pm k(\ell - \ell_0)$, where ℓ is the length of the spring.

Let X_L denote the position of the mass on the left and X_R denote the position of the mass on the right. The lengths of the three springs are (respectively, from left to right) X_L, $X_R - X_L$ and $D - X_R$. The first spring exerts a force $-k(X_L - \ell_0)$ on the left mass. The second spring exerts a force $+k(X_R - X_L - \ell_0)$ on the left mass and a force $-k(X_R - X_L - \ell_0)$ on the right mass. The third spring exerts a force $+k(D - X_R - \ell_0)$ on the right mass. Newton's second law for the two masses yields

$$m\ddot{X}_L = -k(X_L - \ell_0) + k(X_R - X_L - \ell_0) \ , \tag{15.14a}$$

$$m\ddot{X}_R = -k(X_R - X_L - \ell_0) + k(D - X_R - \ell_0) \ , \tag{15.14b}$$

where dots denote time derivatives.

The *normal mode* analysis is a powerful technique used to characterize the oscillations of a system near equilibrium. A normal mode is a pattern of oscillation in which each part of the system vibrates about its equilibrium position with a common angular frequency ω.

The equilibrium configuration for the system is found by setting $\ddot{X}_L = \ddot{X}_R = 0$ in Eqs. (15.14) and solving for X_L and X_R.

Exercise 15.7a

Verify that the equilibrium solution is $X_L = D/3$, $X_R = 2D/3$.

Let $x_L = X_L - D/3$ denote the position of the left mass relative to its equilibrium position. Likewise, $x_R = X_R - 2D/3$ denotes the position of the right mass relative to its equilibrium position. In terms of these new coordinates, the equations of motion become

$$m\ddot{x}_L = -2kx_L + kx_R \ , \tag{15.15a}$$

$$m\ddot{x}_R = kx_L - 2kx_R \ . \tag{15.15b}$$

Exercise 15.7b

Verify the results in Eqs. (15.15).

To find the normal modes for this system, we assume a solution of the form

$$x_L = a_L \cos(\omega t) \tag{15.16a}$$

$$x_R = a_R \cos(\omega t) \tag{15.16b}$$

for some amplitudes a_L, a_R and angular frequency ω. Plugging the expressions (15.16) into the differential equations (15.15) and cancelling the common factors of $\cos(\omega t)$, we find

$$-m\omega^2 a_L = -2ka_L + ka_R \ , \tag{15.17a}$$

$$-m\omega^2 a_R = ka_L - 2ka_R \ . \tag{15.17b}$$

Divide through by $-k$ and write this in matrix notation as

$$\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \begin{pmatrix} a_L \\ a_R \end{pmatrix} = (m\omega^2/k) \begin{pmatrix} a_L \\ a_R \end{pmatrix} \ . \tag{15.18}$$

This is an eigenvalue–eigenvector problem, $Mv = \lambda v$, with matrix $M = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$, eigenvalues $\lambda \equiv m\omega^2/k$, and eigenvectors $v = \begin{pmatrix} a_L \\ a_R \end{pmatrix}$.

The command `eval,evec = linalg.eig(M)` yields

```
array([3., 1.])
```

for the eigenvalues `eval` and

```
matrix([[ 0.7071068, 0.70710678],
        [-0.7071068, 0.70710678]])
```

for the eigenvectors `evec`. Since M is a 2×2 matrix, there are two eigenvalues and two eigenvectors. The first eigenvalue `eval[0]` and its corresponding eigenvector `evec[:,0]` are

$$\lambda_1 = 3.0 \ , \tag{15.19a}$$

$$v_1 = \begin{pmatrix} 0.7071068 \\ -0.7071068 \end{pmatrix} \ . \tag{15.19b}$$

The second eigenvalue `eval[1]` and its corresponding eigenvector `evec[:,1]` are

$$\lambda_2 = 1.0 \ , \tag{15.20a}$$

$$v_2 = \begin{pmatrix} 0.7071068 \\ 0.7071068 \end{pmatrix} \ . \tag{15.20b}$$

Note that `eval` is a NumPy array, whereas `evec` is a NumPy matrix.[2]

The two solutions of the eigenvalue–eigenvector problem are the *normal modes*. Let's take a close look at the first normal mode, defined by λ_1 and v_1. The amplitudes from Eqs. (15.16) are the components of v_1; thus, $a_L = 0.7071068$ and $a_R = -0.7071068$. The frequency (which we denote ω_1) is given by $\lambda_1 = m\omega_1^2/k$, so that $\omega_1 = \sqrt{k\lambda_1/m}$. To be concrete, let's assume the parameter values $m = 0.3\,\text{kg}$ and $k = 2\,\text{kg/s}^2$. Then the frequency for the first normal mode, with $\lambda_1 = 3.0$, is $\omega_1 = 4.47/\text{s}$. This corresponds to an oscillation period of $2\pi/\omega_1 = 1.40\,\text{s}$.

The second normal mode is defined by λ_2 and v_2. The amplitudes are $a_L = 0.7071068$ and $a_R = 0.7071068$ and the frequency is $\omega_2 = \sqrt{k\lambda_2/m} = 2.58/\text{s}$. The oscillation period is $2.43\,\text{s}$.

The normal modes are shown in the two graphs of Fig. 15.6. For each mode, the positions X_L and X_R are plotted versus time t, with time on the vertical axis.

The general motion for this system is a linear combination of normal modes with arbitrary phase. That is, the general solution of the differential

[2]It can be useful to define the eigenvectors as `v1 = evec[:,0]`, *etc.* This defines `v1` as a NumPy matrix with two rows and one column. In particular, `v1` is *not* a one–dimensional NumPy array. Since it takes two indices to access the elements of a matrix, the elements of `v1` are `v1[0,0]` and `v1[1,0]`.

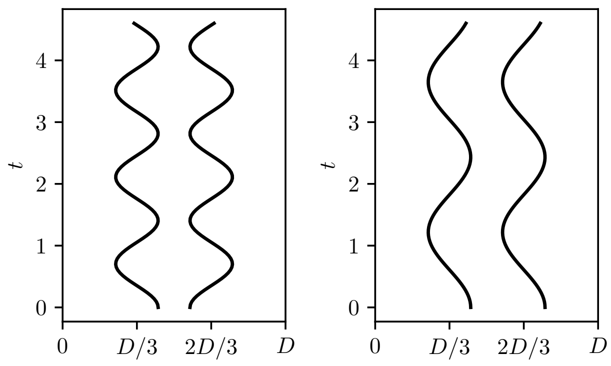

Fig. 15.6: The normal modes for the system of two masses and three springs shown in Fig. 15.5.

equations (15.14) is

$$\begin{pmatrix} X_L(t) \\ X_R(t) \end{pmatrix} = \begin{pmatrix} D/3 \\ 2D/3 \end{pmatrix} + A_1 v_1 \cos(\omega_1 t + \phi_1) + A_2 v_2 \cos(\omega_2 t + \phi_2) \ , \quad (15.21)$$

where A_1 and A_2 are constant amplitudes and ϕ_1 and ϕ_2 are constant phase angles.

Exercise 15.7c

Verify the results of this section: Write a code to compute the normal modes for the system (15.14). Have your code reproduce the plots from Fig. 15.6.

Exercise 15.7d

Two masses (both m) are suspended from a ceiling by two springs, as shown in Fig. 15.7. The springs are identical, with stiffness k and relaxed length ℓ_0. Write a code to compute and plot the normal modes.

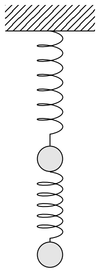
Fig. 15.7: Two masses suspended from the ceiling with springs.

Exercise 15.7e

Three masses (each of mass m) move in one dimension along a frictionless table, as shown in Fig. 15.8. The masses are connected between fixed walls by four identical springs. Find the normal modes for this system. Plot the modes as functions of time.

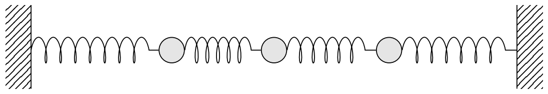
Fig. 15.8: Three masses connected between walls by four springs.