Chapter 1

Numerical Solution of a System of ODEs

Systems of differential equations come up because nature is complex. Most systems of interest in science and engineering are made of many interacting components with multiple relationships. Each component must be represented by a variable and, in the context of differential equation, each has to be modeled by a separate equation.

In this chapter, the reader will:

- Learn about the predator-prey model, an important application of differential equations to the study of an ecological system.
- Learn how to load the Python libraries necessary to solve systems of differential equations.
- Explore the use of the `odeint` function, which is the main tool we will use to solve systems of differential equations.
- Learn how to produce different graphical displays of solutions of a system of differential equations.

The exercises at the end of the chapter present several other examples and give the reader an idea of the breadth of application areas that can be studied with systems of differential equations.

1.1 The Predator-Prey Model

This model describes two animal species in the wild, traditionally referred to as "rabbits" and "foxes". The following assumptions are made about the two populations and their interactions:

- Without the presence of foxes, the rabbit population will grow exponentially without limit. This, of course, is a simplification, since in a real population there would be environmental limits to growth.
- Without the presence of rabbits, the fox population will decay exponentially and become extinct. This is tacitly assuming that foxes prey exclusively on rabbits. Most predators will hunt more than one prey species.

- The effect of predation on the growth of each species is proportional to the product of the sizes of the rabbit and fox populations. This is also a simplification, since it essentially posits that, the capacity for hunting of the fox population is unlimited. Typically, there is a saturation point after which, no matter how large the rabbit population is, the level of predation remains constant.

Denoting by $R(t)$ and $F(t)$, respectively, the sizes of the rabbit and fox populations, the assumptions above entail the following model:

$$R' = aR - bRF \tag{1.1}$$

$$F' = -cF + dRF \tag{1.2}$$

The parameters a, b, c and d are assumed to be positive, and can be interpreted as follows:

- a is the per-capita growth rate of the rabbit population in the absence of foxes.
- b represents the effect of predation on the rabbit population.
- c is the per-capita decay rate of the fox population in the absence of rabbits.
- d represents the effect of predation on the fox population.

The equations in (1.1), (1.2) can be solved analytically to provide an implicit relationship between the variables R and F (see Exercise 1.5.1). In the next sections we will describe, instead, an experimental approach, based on numerical simulations.

1.2 Computing Solutions

We now concentrate on the task of computing solutions for the predator-prey systems. The computations presented here use numerical methods and are, thus, approximations to the actual solutions. The details of the computation (including choice of step sizes and accuracy) are hidden from the user. This suits us well right now, and readers interested in the details of the computations are referred to the documentation of the `odeint` method [1].

The following code assumes that the reader has an open Jupyter notebook, and that a standard distribution, such as Anaconda, has been installed. To solve a system of differential equations, the first step is to load the required libraries into the Jupyter workspace, which can be done by executing a code cell with the statements shown below:

```
1 %matplotlib inline
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from scipy.integrate import odeint
```

The code displayed above achieves the following:

- **Line 1**: Sets up the graphics library `matplotlib` so that graphs as displayed inline. This means that each graph appears right after the computing cell that creates it.
- **Line 2**: Import the module **numpy** with the alias **np**. **numpy** is the module that defines the **array** object, which is a Python data structure optimized for computational mathematics.

[1]http://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html#scipy.integrate.odeint

- **Line 3**: Import the module **matplotlib.pyplot** with the alias **plt**. **pyplot** is a collection of objects and functions for interactive graphics.
- **Line 4**: Import the **odeint** function from the module **scipy.integrate**. **odeint** is the solver for systems of differential equations that we will be using.

The code shown above needs to be run only once for the entire notebook. So, it is typical to put it on a cell right on top of the notebook.

The next task is to define of the system of differential equations (1.1), (1.2) in Python. The interface of the function **odeint** requires this to be done with a function with a specific signature. In our situation, we can use the code below, which should be run in a separate computing cell:

```
1 def pred_prey_system(x, t, *args):
2     a, b, c, d = args
3     R, F = x
4     return np.array([a*R - b*R*F, -c*F + d*R*F])
```

These code lines do the following:

- **Line 1:** This starts the definition the Python function `pred_prey_system`. The input arguments are:
  - `x` is a 2-component `numpy` array, representing the (R, F) pair of variables.
  - `t` is a floating point value, representing time.
  - `*args` is a tuple of extra arguments, used, in our example, to pass the parameters a, b, c and d. The `*` notation is a Python feature that allows the tuple to be passed without enclosing parenthesis.
- **Line 2:** Unwrap `args` and store its values in the variables `a`, `b`,`c` and `d`.
- **Line 3:** Unwrap `x` and store its values in the variables `R` and `F`.
- **Line 4:** Return the array that represents the right-hand side of equations in the system (1.1), (1.2) evaluated at the current values of R, F and t.

Once we have defined the system, we can compute solutions as indicated below:

```
1 a, b, c, d = 0.75, .2, 1.15, .1
2 tvalues = np.linspace(0, 10, 11)
3 init = [15, 20]
4 solution = odeint(pred_prey_system, init, tvalues, args=(a,b,c,d))
```

Notice that running this cell will not produce any output, since the result of the computation is stored in the variable `solution`. The code above solves the predator-prey system for the parameter values $a = 0.75$, $b = 0.2$, $c = 1.15$, $d = 0.1$ and the initial conditions $R(0) = 15$, $P(0) = 20$. The details of the computation are as follows:

- **Line 1:** Set the parameter values `a`, `b`, `c` and `d`.
- **Line 2:** Define the array `tvalues`. The expression `np.linspace(0, 10, 11)` generates 11 equally spaced points in the interval $[0, 10]$. Notice that both endpoints are included in the generated array.
- **Line 3:** Define the list `init`, which contains the initial values of the populations.
- **Line 4:** Call `odeint` to compute the solution and store the output in the array `solution`. The arguments to the call are:

- `pred_prey_system` is the function that specifies the system of differential equations, as defined in the previous computation cell.
- `init` is an array containing the initial condition.
- `tvalues` is an array that specifies the times at which the solution is be computed. The first value, `tvalues[0]`, is the time at which the initial condition is given.
- `args=(a,b,c,d)` sets the parameter values for this instance of the model.

We can see the results of the computation by entering the variable name `solutions` by itself in a computation cell:

```
solution
```

This produces the output:

```
array([[ 15.        ,  20.        ],
       [  1.28022691,  10.17471157],
       [  0.76907897,   3.5250347 ],
       [  1.05575964,   1.21782139],
       [  1.91904005,   0.44474622],
       [  3.83220224,   0.18548621],
       [  7.89563598,   0.10296241],
       [ 16.39501079,   0.10433694],
       [ 33.47233713,   0.36492363],
       [ 43.63108633,   8.70404658],
       [  2.98858554,  15.85969476]])
```

The solution is provided in a 2-dimensional array, with the rabbit population in the first column and the fox population in the second column. Notice that the values of the time variable are not included in the solution, since they are available in the `tvalues` array.

1.3 Generating Solution Plots

Producing solution plots is one of the most important steps in the study of a a system of differential equations. Let's first consider the problem of generating a solution for a larger set of time values, as indicated in the following code:

```
a, b, c, d = 0.75, .2, 1.15, .1
tvalues = np.linspace(0, 30, 600)
init = [15, 20]
solution = odeint(pred_prey_system, init, tvalues, args=(a,b,c,d))
```

This code is very similar to the previous computation, the only difference being on the line defining `tvalues`, where we set the maximum time to 60 and the number of time values to 600.

In order to plot the solutions, it is convenient to have the vectors representing the rabbit and fox populations in separate arrays. This can be done with the line of code below:

```
Rvalues, Fvalues = solution.transpose()
```

This code first transposes the `solution` array, so that the two populations are across the rows of the array. The rows of the array are then assigned to the variables `Rvalues` and `Fvalues`, which will be used to produce the plots.

To create a time-dependent plot of the solutions we can use the following code:

```
1 plt.plot(tvalues, Rvalues, color='green', lw=2)
2 plt.plot(tvalues, Fvalues, color='red', lw=2)
3 plt.title('Predator-prey model ' +
4     'a={:3.2f}, b={:3.2f}, c={:3.2f}, d={:3.2f}'.format(a,b,c,d))
5 plt.xlabel('$t$')
6 plt.ylabel('$R(t),F(t)$')
7 plt.legend(['$R(t)$','$F(t)$'], loc='upper left')
8 None
```

This will produce the plot shown in Figure 1.1. The plot is generated with several successive calls to functions in the module `plt`. Each call adds a component to the graph. This is called the "state machine" approach to construct a graph, which is convenient to produce graphs in an interactive fashion. The purpose of each call is discussed below:

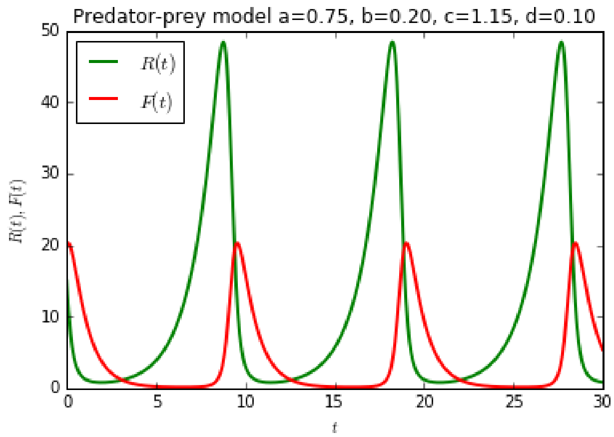
- **Lines 1 and 2:** Generate the graphs of the functions $R(t)$ and $F(t)$. In each call, the data is given in two arrays, which are the values plotted, respectively, across the horizontal and vertical axes. The remaining arguments to `plot` specify the color and line width of each graph.
- **Lines 3 and 4:** Create the title for the plot. The title is given as a formatted string, and reports the values of the parameters a, b, c and d. For purposes of display on the book page, the string was split in two pieces, but might as well be typed in a single line.
- **Lines 5 and 6:** Print the axes labels. Notice the use of $\TeX$, which is a feature available in most `matplotlib` functions.
- **Line 7:** Add a legend to the plot, placing it on the upper-left corner, so that it does not interfere with the plot.
- **Line 8:** The default behavior in an executed cell is to print the last computed value. The output of the functions in the `plt` module is irrelevant to us, and adding `None` at the end of the cell prevents its output.

Notice the apparent periodic behavior of the solution. This cyclical interplay in the population dynamics is typical of predator-prey interactions in the wild, although the data will never be as regular as the solutions of the differential equations. The periodical behavior of the solutions is even more evident in the phase diagram of the solution, which can be produced with the code below:

```
1 plt.plot(Rvalues, Fvalues, lw=2)
2 plt.title('Predator-prey model ' +
3     'a={:3.2f}, b={:3.2f}, c={:3.2f}, d={:3.2f}'.format(a,b,c,d))
4 plt.xlabel('$R$')
5 plt.ylabel('$F$')
6 None
```

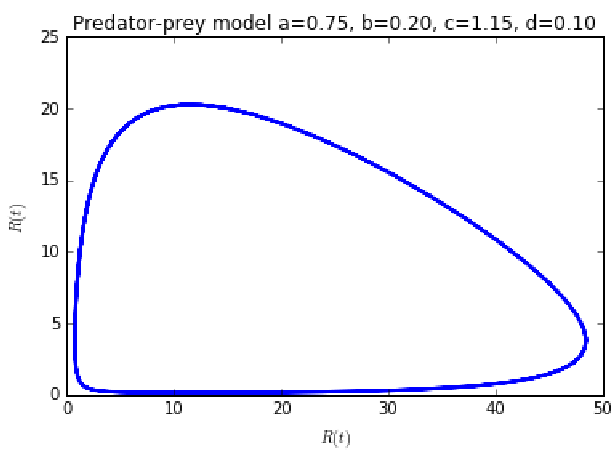
The graph produced is displayed in Figure 1.2. The code is very similar to the previous example, the main difference being that now, in Line 1, we plot `Rvalues` in the horizontal

Figure 1.1: Time-dependent plot of a solution to the predator-prey model.



axis and `Fvalues` in the vertical axis. Also notice that we omit the legend, since the meaning of the curve is clear from the axis labels.

Figure 1.2: Phase plot of a solution to the predator-prey model.



The closed curve representing the solution makes the periodicity of the solution evident. Phase plots have the advantage that solutions for several different initial conditions can be plotted simultaneously. This is illustrated by the following code:

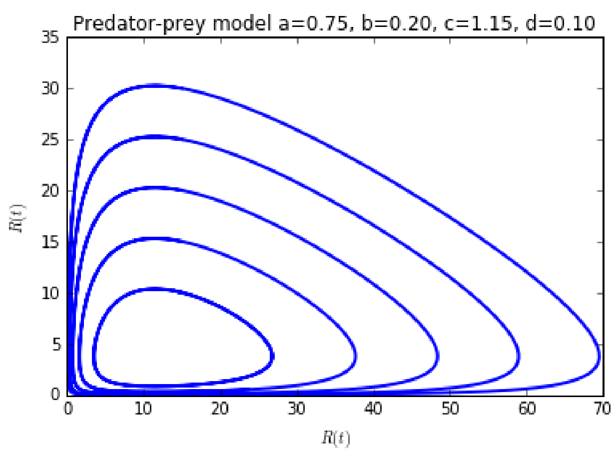
```
a, b, c, d = 0.75, .2, 1.15, .1
tvalues = np.linspace(0, 15, 600)
inits = [[15, 20], [15, 10], [15, 30],
         [15, 15], [15, 25]]
for init in inits:
    solution = odeint(pred_prey_system, init, tvalues, args=(a,b,c,d))
```

```
7      Rvalues, Fvalues = solution.transpose()
8      plt.plot(Rvalues, Fvalues, lw=2, color='blue')
9  plt.title('Predator-prey model ' +
10     'a={:3.2f}, b={:3.2f}, c={:3.2f}, d={:3.2f}'.format(a,b,c,d))
11 plt.xlabel('$R(t)$')
12 plt.ylabel('$R(t)$')
13 None
```

Please refer to Figure 1.3. The code shown above is mostly a rearrangement of code seen before. The main differences are discussed below:

- **Lines 3, 4:** Since we want to plot solutions for several different initial conditions, we define a Python list called `inits` containing all the initial conditions we want to consider.
- **Lines 5–8:** The computation and plotting of each solution are now wrapped into a `for` loop. The `for` control structure, specified in line 5, will step over the elements of the list `inits`, assigning each of the initial conditions to the variable `init` in succession. For each initial condition, we compute the solution and then plot it in the body of the loop.

Figure 1.3: Phase plot of several solutions to the predator-prey model.



It becomes evident that all solutions plotted are periodical. In the exercises, the reader is invited to explore the nature of the solutions in more detail.

1.4 Conclusions

In this chapter, the reader learned the following:

- How to represent the interaction of two species in a predator-prey situation.
- How to load the Python modules necessary to solve a system of differential equations.
- How to define a function, `pred_prey_system` that represents the system of differential equations.

- How to use the function `odeint` to solve the system, and how to access the solution.
- How to produce several different types of displays representing solutions using module `matplotlib.pyplot` (referred to as `plt` in the code).

In the next chapter we will concentrate on linear systems, and how to specialize the methods of this chapter to that important particular case.

1.5 Exercises

1.5.1. Equations (1.1), (1.2) can be written using Leibniz notation for derivatives as follows:

$$\frac{dR}{dt} = aR - bRF \tag{1.3}$$

$$\frac{dF}{dt} = -cF + dRF \tag{1.4}$$

This suggests dividing equation (1.3) by equation (1.4) to obtain:

$$\frac{dR}{dF} = \frac{aR - bRF}{-cF + dRF} \tag{1.5}$$

1. Use separation of variables on equation 1.5 to obtain an implicit formula relating the variables R and F.
2. Visit the matplotlib website (`http://matplotlib.org`) and search the documentation for a function that plots implicitly defined equations, and use it to draw a plot of R versus F, using the equation from the previous item.

1.5.2. The version of the predator-prey model described in this chapter makes the unrealistic assumption that the rabbit population can grow without limit. A more reasonable assumption is that, in the absence of predators, the rabbit population grows according to a *logistic model*:

$$R' = aR\left(1 - \frac{R}{N}\right)$$

The parameter N is a positive constant called the *carrying capacity*.

1. Modify the system (1.1), (1.2) to represent this assumption.
2. Conduct a graphical analysis of the system, by plotting solutions to various different sets of parameters and initial conditions. Describe qualitatively the solutions and interpret them in terms of a predator-prey situation.
3. Do you think, in this case, it is possible to find an implicit relationship between R and F, as was done in Exercise 1.5.1? Explain why or why not.

1.5.3. The *Van der Pol* oscillator is used to describe the dynamics of a certain electronic component, and is described by the system of differential equations:

$$x' = y$$
$$y' = \mu(1 - x^2)y - x$$

where the parameter μ is positive. Conduct a series of numerical experiments to investigate solutions of this system for different values of μ. Then, describe the dynamics of the solutions.

1.5.4. The *SIR model* for spread of an infection classifies individuals in the infected population in three *compartments*:

- $S(t)$ represents the number of individuals that are *susceptible* to be infected at time t.
- $I(t)$ represents the number of individuals that are *infected* at time t.
- $R(t)$ represents the number of individuals that are *removed* from the population, either because they acquired immunity after infection, or due to death.

Assuming that the population is fixed:

$$N = S(t) + I(t) + R(t) \tag{1.6}$$

the following system of differential equations can be used to describe the evolution of the infection:

$$S' = -\frac{\beta SI}{N} \tag{1.7}$$

$$I' = \frac{\beta SI}{N} - \gamma I \tag{1.8}$$

$$R' = \gamma I \tag{1.9}$$

1. By taking the derivative in equation 1.6 we get $S'(t) + I'(t) + R'(t) = 0$. Show that this is compatible with equations 1.7–1.9.
2. Using equation 1.6, derive a system of differential equations that involves only the variables I and R.
3. Conduct numerical experiments to investigate the dynamics of the epidemics for several different parameter values. Of special interest is to determine for which parameter values will the infection take hold and eventually infect the whole population, and for which values the infection will eventually subside.

Chapter 2

The Theory of Linear Systems of ODEs

In this chapter, we turn to the solution of linear systems of differential equations. Linear systems appear in several applications, including electrical circuits, small mechanical vibrations and continuous time, discrete space Markov processes. The important class of constant coefficient systems results for the standard linearization of an nonlinear system near an equilibrium.

2.1 The Theory of Linear Systems

In this section we will consider the system of differential equations:

$$x' = a(t)x + b(t)y + f(t) \tag{2.1}$$

$$y' = c(t)x + d(t)y + g(t) \tag{2.2}$$

It is often convenient to write this system in vector form. To this end, define:

$$\mathbf{x}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}; \quad A(t) = \begin{bmatrix} a(t) & b(t) \\ c(t) & d(t) \end{bmatrix}; \quad \mathbf{F}(t) = \begin{bmatrix} f(t) \\ g(t) \end{bmatrix}$$

We can then write the system (2.1), (2.2) as:

$$\mathbf{x}' = A(t)\mathbf{x} + \mathbf{F}(t) \tag{2.3}$$

We call $A(t)$ the *coefficient matrix* of the system, and $\mathbf{F}(t)$ the *forcing term*. If $\mathbf{F}(t) = 0$ for all t, we say that the system is *homogeneous*. Otherwise, it is *non-homogeneous*.

We remark that all the results in this chapter can be extended for the case where, in (2.3) if $\mathbf{x}(t)$ and $\mathbf{F}(t)$, n-dimensional vectors, and $\mathbf{A}(t)$ is an $n \times n$ matrix.

Example 2.1.1. The equation of a forced linear oscillator is:

$$mx'' + cx' + kx = f(t),$$

where $m > 0$, $c \geq 0$ and $k > 0$, and $x(t)$ represents the displacement of the oscillating mass from the equilibrium position. If we let $y(t) = x'(t)$, this equation can be represented as the system:

$$\begin{aligned} x' &= y \\ y' &= -\frac{k}{m}x - \frac{c}{m}y + f(t) \end{aligned}$$

It is, in general, using an analogous trick, to represent any linear differential equation of order n into a n-dimensional system of linear differential equations.

The fundamental theoretical result about linear systems is the following:

Theorem 2.1.2 (Existence and uniqueness of solutions). *Suppose that:*

1. *The functions $a(t)$, $b(t)$, $c(t)$, $d(t)$, $f(t)$ and $g(t)$ are continuous on the open interval $I \subset \mathbb{R}$.*
2. *$t_0 \in I$ and x_0, y_0 are real numbers.*

Then, there is a unique pair of functions $(x(t), y(t))$ defined for $t \in I$ such that (2.1), (2.2) are valid on I and $x(t_0) = x_0$, $y(t_0) = y_0$.

In the next sections, we will prove a series of theorems that characterize the set of solutions of a linear system of differential equations. The next section starts this study for the case where the forcing term is zero.

2.2 Homogeneous Systems

In this section we consider the system of differential equations:

$$x' = a(t)x + b(t)y \tag{2.4}$$
$$y' = c(t)x + d(t)y \tag{2.5}$$

In vector notation:

$$\mathbf{x}' = A(t)\mathbf{x} \tag{2.6}$$

where:

$$\mathbf{x}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}; \quad A(t) = \begin{bmatrix} a(t) & b(t) \\ c(t) & d(t) \end{bmatrix}.$$

We start with the following important property of solutions of the system:

Theorem 2.2.1 (Principle of superposition for homogeneous systems). *Suppose that:*

1. *$\mathbf{x}_1(t)$ and $\mathbf{x}_2(t)$ are solutions of the system (2.6) on an open interval I.*
2. *c_1 and c_2 are arbitrary real numbers.*

Then, $\mathbf{x}(t) = c_1\mathbf{x}_1(t) + c_2\mathbf{x}_2(t)$ is also a solution of (2.6) on I.

Proof. Using (2.6)

$$\mathbf{x}'(t) = \frac{d}{dt}\left[c_1\mathbf{x}_1(t) + c_2\mathbf{x}_2(t)\right] = c_1\mathbf{x}_1'(t) + c_2\mathbf{x}_2'(t) = \\ c_1A(t)\mathbf{x}_1(t) + c_2A(t)\mathbf{x}_2(t) = A(t)\left[c_1\mathbf{x}_1(t) + c_2\mathbf{x}_2(t)\right] = A(t)\mathbf{x}(t).$$

□

One of the important consequences of this theorem is a result about the structure of the set of solutions of the system (2.4), (2.5). Theorem 2.2.1 states that this set of solutions is a *vector space*. We will now show that this vector space has dimension 2. We start with a definition.

Definition 2.2.2. Suppose that:

$$\mathbf{x}_1(t) = \begin{bmatrix} x_1(t) \\ y_1(t) \end{bmatrix}, \quad \mathbf{x}_2(t) = \begin{bmatrix} x_2(t) \\ y_2(t) \end{bmatrix},$$

are two vector functions defined for t in the open interval $I \subset \mathbb{R}$. We say that $\mathbf{x}_1$ and $\mathbf{x}_2$ are *linearly independent* over I if and only if the following is true:

$$c_1\mathbf{x}_1(t) + c_2\mathbf{x}_2(t) = \mathbf{0} \text{ for all } t \in I \quad \text{implies} \quad c_1 = c_2 = 0.$$

If $\mathbf{x}_1$ and $\mathbf{x}_2$ are not linearly independent, they are said to be *linearly independent*.

The qualification "over I" is often omitted f it is clear from the context what the interval I is.

Example 2.2.3. The functions

$$\mathbf{x}_1(t) = \begin{bmatrix} 1 \\ t \end{bmatrix}, \quad \mathbf{x}_2(t) = \begin{bmatrix} t \\ t^2 \end{bmatrix},$$

are linearly independent over the interval $I = \mathbb{R}$. To see why, suppose that c_1 and c_2 are scalars such that $c_1\mathbf{x}_1 + c_2\mathbf{x}_2 = \mathbf{0}$ for all $t \in \mathbb{R}$, that is:

$$\begin{bmatrix} c_1 + c_2t \\ c_1t + c_2t^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ for all } t \in \mathbb{R}. \tag{2.7}$$

If we let $t = 0$ this implies

$$\begin{bmatrix} c_1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

so that $c_1 = 0$. Using this and plugging in $t = 1$ in (2.7) we get:

$$\begin{bmatrix} c_2 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Thus, (2.7) for all t, which implies $c_1 = c_2 = 0$. We conclude that $\mathbf{x}_1$ and $\mathbf{x}_2$ are linearly independent, as claimed.

The strategy used in the previous example (picking up values of t and plugging them in) will, usually, be enough to show linear independence of arbitrary functions. If we know the functions are solutions of a system of linear differential equations, however, we can do much better. This is shown in the next theorem:

Theorem 2.2.4. *Suppose that the assumptions of Theorem 2.1.2 are satisfied and:*

$$\mathbf{x}_1(t) = \begin{bmatrix} x_1(t) \\ y_1(t) \end{bmatrix} \text{ and } \mathbf{x}_2(t) = \begin{bmatrix} x_2(t) \\ y_2(t) \end{bmatrix}$$

are solutions of the system (2.4), (2.5) over the interval $I \subset \mathbb{R}$. Then, the following are equivalent:

1. *$\mathbf{x}_1$ and $\mathbf{x}_2$ are linearly independent over I.*
2. *The vectors $\mathbf{x}_1(t)$ and $\mathbf{x}_2(t)$ are linearly independent for all $t \in I$ (as vectors in $\mathbb{R}^2$).*
3. *There is a $t_0 \in I$ for which $\mathbf{x}_1(t_0)$ and $\mathbf{x}_2(t_0)$ are linearly independent.*
4. *For all $t \in I$,*
$$\det \begin{bmatrix} x_1(t) & x_2(t) \\ y_1(t) & y_2(t) \end{bmatrix} = 0.$$
5. *There is a $t_0 \in I$ such that:*
$$\det \begin{bmatrix} x_1(t_0) & x_2(t_0) \\ y_1(t_0) & y_2(t_0) \end{bmatrix} = 0.$$

In the statement of the theorem, we used the notion of linear independence of vectors on the plane $\mathbb{R}^2$. We say that two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ are linearly independent if $c_1\mathbf{v}_1 + c_2\mathbf{v}_2 = \mathbf{0}$ implies $c_1 = c_2 = 0$ for all scalars c_1 and c_2. On the plane, two vectors are linear independent if and only if they are not a scalar multiple of each other. In other words, the vectors must point to different directions.

The theorem says that, for solutions of a system of linear differential equations, when checking linear independence we need only check it at a fixed value t_0. Let's now turn to the proof of the theorem.

Proof. To prove Theorem 2.2.4, we start by showing that (1) implies (2). are equivalent.

Suppose that $\mathbf{x}_1$ and $\mathbf{x}_2$ are linearly independent over I, and let t_0 be an arbitrary point of I. We aim to show that the vectors $\mathbf{x}_1(t_0)$ and $\mathbf{x}_2(t_0)$ (in $\mathbb{R}^2$) are linearly independent.

Suppose that c_1 and c_2 are scalars such that

$$c_1\mathbf{x}_1(t_0) + c_2\mathbf{x}_2(t_0) = \mathbf{0}.$$

Let $\mathbf{y}(t) = c_1\mathbf{x}_1(t) + c_2\mathbf{x}_2(t)$. By the principle of superposition (Theorem 2.2.1) is a solution of the system of linear equations (2.4), (2.5), and $\mathbf{y}(t_0) = \mathbf{0}$. Then, the uniqueness part of Theorem 2.1.2 implies that $\mathbf{y}(t) = \mathbf{0}$ for all $t \in I$, that is:

$$c_1\mathbf{x}_1(t) + c_2\mathbf{x}_2(t) = \mathbf{0} \text{ for all } t \in I.$$

Since $\mathbf{x}_1$ and $\mathbf{x}_2$ are linearly independent this implies that $c_1 = c_2 = 0$, and we conclude that $\mathbf{x}_1(t_0)$ and $\mathbf{x}_2(t_0)$ are linearly independent.

The implications (2) $\implies$ (3) and (3) $\implies$ (1) follow directly from the definition of linear independence.

Thus, we have proved that (1), (2) and (3) are equivalent. The equivalence of (4) and (5), respectively, to (2) and (3) follows from the well-known characterization of linear independence in terms of determinants. □

We are now ready to prove the main result about the structure of the solution set of the system (2.4), (2.5).

Theorem 2.2.5. *Suppose that the assumptions of Theorem 2.1.2 are satisfied. Then:*

1. *There are two solutions* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *of the system* (2.4), (2.5) *that are linearly independent over* I.
2. *If* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *are linearly independent solutions of the system* (2.4), (2.5)*, then, for every other solution* $\mathbf{x}$ *of the system, there are scalars* c_1 *and* c_2 *such that* $\mathbf{x}(t) = c_1\mathbf{x}_1(t) + c_2\mathbf{x}_2(t)$ *for all* $t \in I$.

Proof. To prove part (1), fix $t_0 \in I$ and let $\mathbf{x}_1(t)$ and $\mathbf{x}_2(t)$ be solutions of the system (2.4), (2.5) that satisfy:

$$\mathbf{x}_1(t_0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } \mathbf{x}_2(t_0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

These two solutions are guaranteed to exist by Theorem 2.1.2. Since

$$\det \begin{bmatrix} \mathbf{x}_1(t_0) & \mathbf{x}_1(t_0) \end{bmatrix} = \det \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \neq 0,$$

Theorem 2.2.4 implies that $\mathbf{x}_1$ and $\mathbf{x}_2$ are linearly independent.

Let's now turn to the proof of part (2). Suppose that

$$\mathbf{x}_1(t) = \begin{bmatrix} x_1(t) \\ y_1(t) \end{bmatrix} \text{ and } \mathbf{x}_2(t) = \begin{bmatrix} x_2(t) \\ y_2(t) \end{bmatrix}$$

are linearly independent solutions of the system (2.4), (2.5) on the interval I. Let $\mathbf{x}(t)$ be any other solution. Fix $t_0 \in I$. By part (4) of Theorem 2.2.4,

$$\det \begin{bmatrix} x_1(t_0) & x_2(t_0) \\ y_1(t_0) & y_2(t_0) \end{bmatrix} \neq 0$$

This implies that there is a unique pair c_1, c_2 such that:

$$c_1 \begin{bmatrix} x_1(t_0) \\ y_1(t_0) \end{bmatrix} + c_2 \begin{bmatrix} x_2(t_0) \\ y_2(t_0) \end{bmatrix} = \mathbf{x}(t_0)$$

Let $\mathbf{y}(t) = c_1\mathbf{x}_1(t) + c_2\mathbf{x}_2(t)$. Then $\mathbf{y}(t)$ is a solution of the system of differential equations such that $\mathbf{y}(t_0) = \mathbf{x}(t_0)$. The Uniqueness Theorem then implies that $\mathbf{y}(t) = \mathbf{x}(t)$ for all $t \in I$, that is, $\mathbf{x}(t) = c_1\mathbf{x}_1(t) + c_2\mathbf{x}_2(t)$ for all $t \in I$. □

The Theorem above gives us our main method for finding the *general solution* of the system (2.4), (2.5). All we have to do is to find two linearly independent solutions $\mathbf{x}_1(t)$, $\mathbf{x}_2(t)$. The general solution is then given by:

$$\mathbf{x}(t) = c_1\mathbf{x}_1(t) + c_2\mathbf{x}_2(t),$$

where c_1 and c_2 are arbitrary scalars.

This concludes our study of the homogeneous equation. We next turn to the nonhomogeneous equation.

2.3 Non-Homogeneous Linear Systems