1 Numerical Methods for Initial Value Problems; Harmonic Oscillators

Lab Objective: *Implement several basic numerical methods for initial value problems (IVPs), and use them to study harmonic oscillators.*

Methods for Initial Value Problems

Consider the initial value problem

$$\begin{aligned} y' &= f(x,y), \quad a \leq x \leq b, \\ y(a) &= y_0, \end{aligned} \tag{1.1}$$

where f is a continuous function. A solution of (1.1) is a continuously differentiable function $y(x)$ that satisfies the equation $y' = f(x,y)$ on the interval $[a,b]$ and for which $y(a) = y_0$. In this lab we will focus on numerical methods for approximating $y(x)$, and sidestep the important mathematical problem of verifying that (1.1) has a unique solution.

For many IVPs it is impossible to find a closed-form, analytic expression for the solution. When there is a closed-form expression for the solution, it may not be very useful. In both cases, numerical methods must be relied on to understand the solutions of (1.1).

As an example, consider the initial value problem

$$\begin{aligned} y'(x) &= \sin y(x), \\ y(0) &= y_0. \end{aligned} \tag{1.2}$$

The solution $y(x)$ is defined implicitly by

$$x = \ln \left| \frac{\cos y_0 + \cot y_0}{\csc y + \cot y} \right|.$$

This analytic expression does not provide much intuition, so we turn to a combination of qualitative and numerical methods. Since $\sin(n\pi) = 0$, this differential equation has constant solutions $y_n(x) = n\pi$, $n \in \mathbb{N}$. We can also use an IVP solver to numerically approximate solutions for several other initial values. After plotting these solutions (see Figure 1.1), it becomes obvious how solutions of (1.2) behave in general.

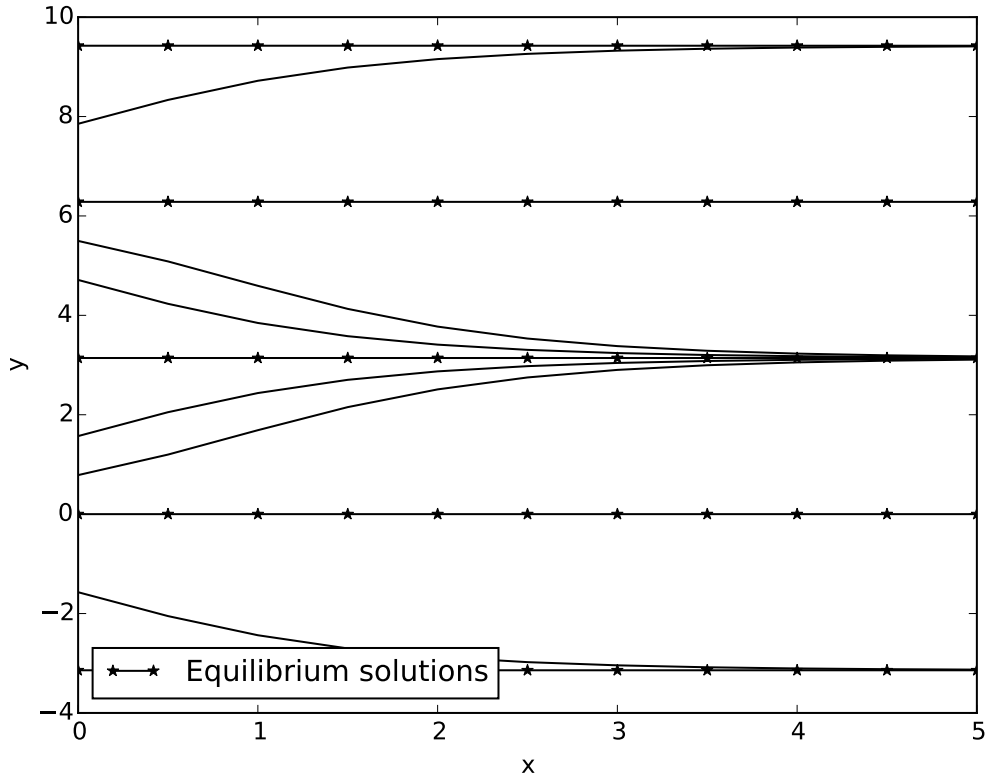

Figure 1.1: Several solutions of (1.2), using Numpy's IVP solver `dopri5`.

Numerical Methods

Many Python integration functions are like black boxes. You plug in some functions and initial conditions, and the methods return some numbers. Numerical methods are used in these black box methods to return those values. We will consider several different methods and their uses.

Euler's Method

Numerical methods for solving initial value problems require us to approximate the solution on a set of grid points $a = x_0 < x_1 < \ldots < x_n = b$ in our interval. For simplicity we will assume that each of the n subintervals $[x_{i-1}, x_i]$ has equal length $h = (b-a)/n$. h is called the *step size*. We then look for values $y_0, y_1, \ldots, y_n$ that approximate the solution at the grid points. For each i, Taylor's theorem says that

$$y(x_{i+1}) = y(x_i) + hy'(x_i) + \frac{h^2}{2}y''(\xi_i) \text{ for some } \xi_i \in [x_i, x_{i+1}].$$

The quantity $\frac{h^2}{2}y''(\xi_i)$ is negligible for small h, and thus

$$\begin{aligned} y(x_{i+1}) &\approx y(x_i) + hy'(x_i), \\ &\approx y(x_i) + hf(x_i, y(x_i)). \end{aligned}$$

This approximation leads to a method called Euler's method: Letting $y_0 = y(a)$, y_{i+1} is given by $y_{i+1} = y_i + hf(x_i, y_i)$ for $i = 0, 1, \ldots, n-1$. Euler's method is a first order method, with error $\mathcal{O}(h^1)$.

A similar application of Taylor's theorem shows that

$$y(x_i) = y(x_{i+1}) - hy'(x_{i+1}) + \frac{h^2}{2}y''(\xi_i) \text{ for some } \xi_i \in [x_i, x_{i+1}];$$

thus for small h

$$y(x_{i+1}) \approx y(x_i) + hf(x_{i+1}, y(x_{i+1})).$$

This approximation leads to another first order method called the backwards Euler method: Letting $y_0 = y(a)$, for $i = 0, \ldots, n-1$ we solve $y_i = y_{i+1} - hf(x_{i+1}, y_{i+1})$ for y_{i+1}.

Note that for both the Euler and backwards Euler methods, only y_i, f, and other points in the interval $[x_i, x_{i+1}]$ are needed to find y_{i+1}. Because of this, they are called *one-step methods*.

Euler's method is an *explicit method*. The backwards Euler method is an *implicit method* since an equation must be solved at each step to find y_{i+1}. Explicit and implicit methods each have advantages and disadvantages. While implicit methods require an equation to be solved at each time step, they often have better stability properties than explicit methods.

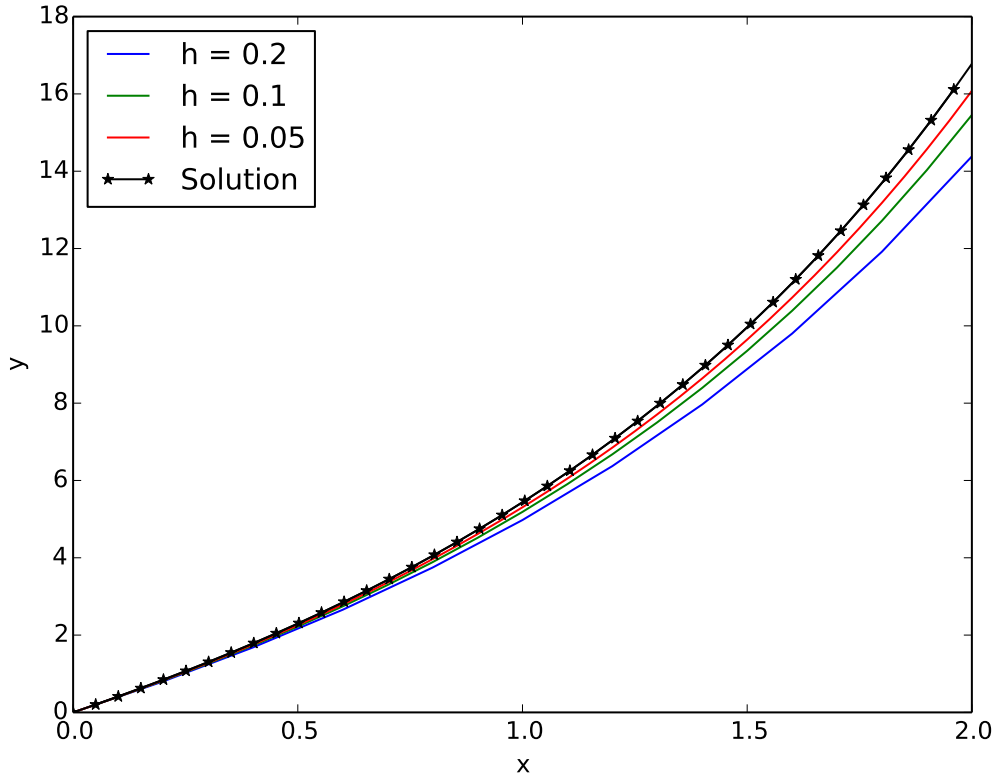

Figure 1.2: The solution of (1.3), alongside several approximations using Euler's method.

Problem 1. The solution of

$$\begin{aligned} y' &= y - 2x + 4, \quad 0 \le x \le 2, \\ y(0) &= 0, \end{aligned} \tag{1.3}$$

is given by $y(x) = -2 + 2x + 2e^x$. Use Euler's method to numerically approximate the solution with step sizes $h = 0.2, 0.1$, and 0.05. Implement the following code to initialize variables and compute all y values.

```
def initialize_all(a,b,y0,h):
    """Given an initial and final time a and b, with y(a)=y0, and step ↩
       size h,
    return several things.

    X: an aray from a to b with n elements, where n is the number of steps↩
        from a to b.
    Y: an empty array of size (n, y.size), Y[0]=y0.
    h: the step size.
    n: the number of steps to be taken.

    """
    n = int((b-a)/h+1)
    X = np.linspace(a, b, n)
    if isinstance(y0, np.ndarray):
        Y = np.empty((n, y0.size))
    else:
        Y = np.empty(n)
    Y[0] = y0
    return X, Y, h, int(n)

def euler(f,X,Y,h,n):
    """Use the Euler method to compute an approximate solution
    to the ODE y' = f(t, y) over X.

    Y[0] = y0
    f is assumed to accept two arguments.
    The first is a constant giving the value of t.
    The second is a one-dimensional numpy array of the same size as y.

    This function returns an array Y of shape (n,) if
    y is a constant or an array of size 1.
    It returns an array of shape (n, y.size) otherwise.
    In either case, Y[i] is the approximate value of y at
    the i'th value of X.
    """

    return None
```

Graph the results and check that your results match Figure 1.2.

Midpoint Method

The midpoint method is very similar to Euler's method. For small h we use the approximation

$$y(x_{i+1}) \approx y(x_i) + hf(x_i + \frac{h}{2}, y(x_i) + \frac{h}{2} f(x_i, y(x_i))).$$

Notice that in this approximation, we first evaluate $\hat{y}_i = y_i + \frac{h}{2} f(x_i, y_i)$ giving us a half step approximation. Then we evaluate f at $\hat{x}_i = x_i + h/2$ and this new approximate $\hat{y}_i$

Runge-Kutta Method

So how do we come up with numerical methods with higher order accuracy? Using Taylor's theorem (as we did for Euler's method) to create higher-order one-step methods would lead to numerically approximating derivatives of $f(t, y)$ - not very desirable.

Let us look for a second order method of the form $y_{i+1} = y_i + af(x_i + b, y_i + c)$. By expanding $af(x + b, y + c)$ with Taylor's theorem and matching constants in the equation

$$f(x, y) + \frac{h}{2} f'(x, y) = f(x, y) + \frac{h}{2} \frac{\partial f}{\partial x}(x, y) + \frac{h}{2} \frac{\partial f}{\partial y}(x, y) \cdot f(x, y),$$

we find that $a = h, b = h/2$, and $c = h/2$. This method is called the Midpoint method. IVP solvers with this general form are called *Runge-Kutta methods*.

There are many Runge-Kutta methods with varying orders of accuracy. Methods of order four or higher are most commonly used. A fourth order Runge-Kutta method iterates as follows:

$$\begin{aligned}
K_1 &= f(x_i, y_i), \\
K_2 &= f(x_i + \frac{h}{2}, y_i + \frac{h}{2} K_1), \\
K_3 &= f(x_i + \frac{h}{2}, y_i + \frac{h}{2} K_2), \\
K_4 &= f(x_{i+1}, y_i + hK_3), \\
y_{i+1} &= y_i + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4).
\end{aligned}$$

Notice that these methods are doing a type of quadrature where we are sampling the function at different points and then performing computation using the samples and some inherent weights. For example, consider a differential equation

$$y' = f(t).$$

Since the function f has no y dependence, this is a simple integration problem, and these IVP methods become well known quadrature methods. In this case, Euler's method corresponds to the left hand sum, and backward Euler's method corresponds to the right hand sum. The modified Euler and midpoint methods are second order IVP methods that correspond to the trapezoidal and midpoint rules for integration, respectively. RK4 corresponds to Simpson's rule for integration.

Advantages of Higher-Order Methods

Higher-order methods are usually much more efficient. One way to measure this efficiency is to determine how many times the right hand side of the initial value problem must be evaluated to provide a desired accuracy. As an example, consider the initial value problem

$$\begin{aligned} y' &= y\cos(x), x \in [0, 8], \\ y(0) &= 1. \end{aligned} \tag{1.4}$$

Figure 1.3 illustrates the comparative efficiency of the Euler, Midpoint, and RK4 methods. The figure also demonstrates another point: since the lower order methods require more floating point operations, floating point error limits the highest possible accuracy that can be achieved with lower order methods.

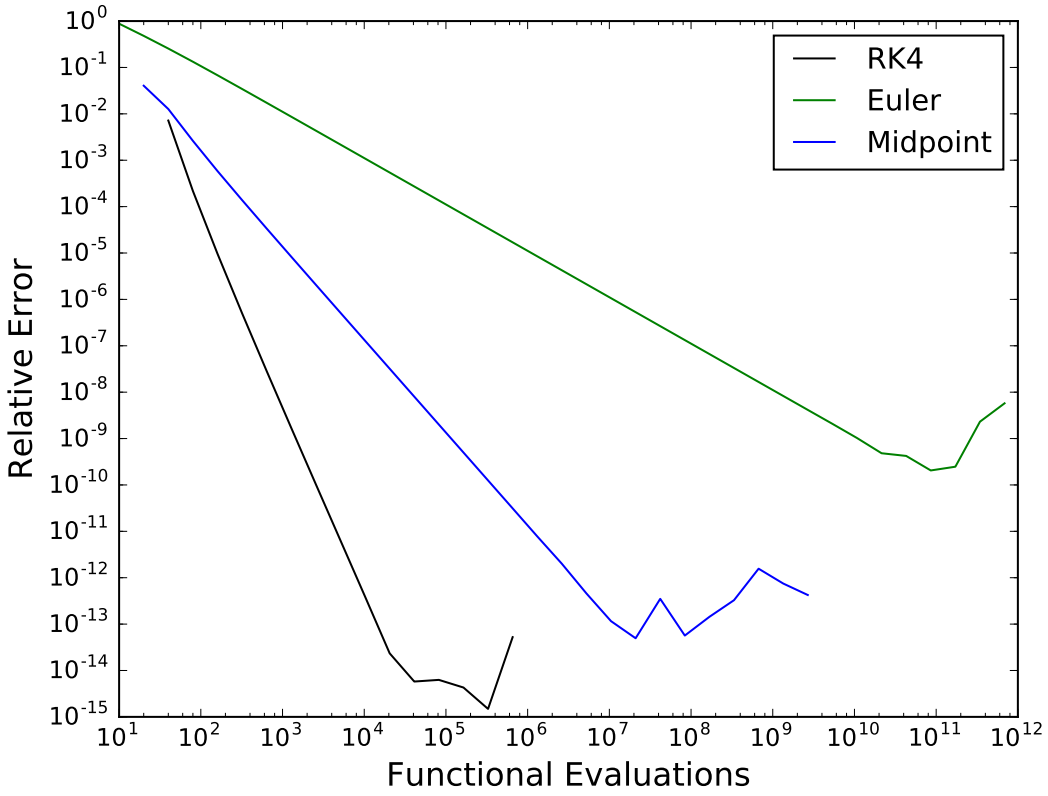

Figure 1.3: Here we graph the relative error in computing the solution of (1.4) at $x = 8$, versus the number of times the right side of (1.4) must be evaluated.

Let t^* be an approximation of some value t. The relative error of the approximation is

$$\frac{|t^* - t|}{|t|}.$$

Note that the relative error is simply the absolute error $|t^* - t|$ normalized by the size of t. A method with order p has error of the form

$$E(h) = Ch^p.$$

This means that the graph of $\log(E)$ versus $\log(h)$ has slope p. The relative error of a numerical method can be approximated and graphed to verify that pth order convergence is occurring. For example, consider the IVP

$$\begin{aligned} y' &= y - 2x + 4, \quad 0 \le x \le 2, \\ y(0) &= 0. \end{aligned} \tag{1.5}$$

The following code solves the initial value problem on several grids using the Euler method, approximates the relative error in computing $y(2)$ and creates a plot (see Figure 1.4).

```
import matplotlib.pyplot as plt

a, b, ya = 0., 2., 0.

def ode_f(x,y):
    return np.array([y - 2*x + 4.])

best_grid = 320                 #  number of subintervals in most refined grid
h = 2./best_grid
X, Y, h, n = initialize_all(a, b, ya, h)
# Requires an implementation of the euler method
best_val = euler(ode_f, X, Y, h, n)[-1]

smaller_grids = [10, 20, 40, 80]  # number of subintervals in smaller grids
h = [2./N for N in smaller_grids]

Euler_sol = [euler(ode_f, initialize_all(a, b, ya, h[i])[0],
            initialize_all(a, b, ya, h[i])[1], h[i], N+1)[-1]
            for i, N in enumerate(smaller_grids)]
Euler_error = [abs((val - best_val)/best_val) for val in Euler_sol]

plt.loglog(h, Euler_error, '-b', label="Euler method", linewidth=2.)
plt.show()
```

Problem 2. Consider the IVP (1.5). Use the Midpoint method and the fourth order Runge-Kutta method (RK4) to approximate the value of the solution at $x = 2$, with a step size of $h = 0.2$, 0.1, 0.05, 0.025, and 0.0125. Create a log-log plot of the relative error of each approximation using the `loglog` function in `matplotlib` (see Figure 1.4).

Harmonic Oscillators and Resonance

Harmonic oscillators show up often in classical mechanics. A few examples include the pendulum (with small displacement), spring-mass systems, and the flow of electric current through various types of circuits. A harmonic oscillator can be described by an initial value problem of the form

$$\begin{aligned} my'' + \gamma y' + ky &= f(t), \\ y(0) = y_0, \quad y'(0) &= y_0'. \end{aligned}$$

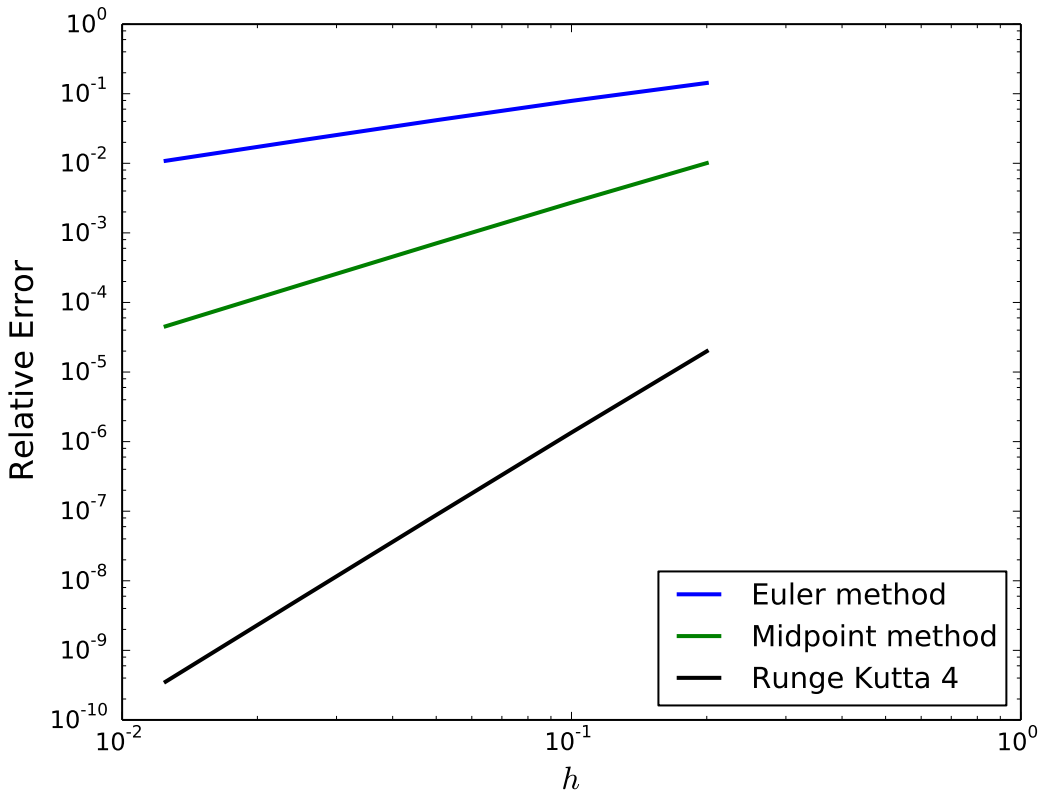

Figure 1.4: The solution of $y' - y = -2x + 4$, $y(0) = 0$, is $y(x) = -2 + 2x + 2e^x$. This loglog plot shows the relative error in numerically approximating $y(2)$, using step sizes $h = 0.2$, 0.1, 0.05, 0.025, and 0.0125. The slope of each line demonstrates the first, second, and fourth order convergence of the Euler, Midpoint, and RK4 methods, respectively.

We will describe the construction of this mathematical model in the context of a spring-mass system.

Suppose an object with mass m is placed at the end of a horizontal spring. The natural position of the object is called the *equilibrium position* for the system. If the object is displaced from its equilibrium position and given an initial velocity, it will act like a harmonic oscillator. The principal property of a harmonic oscillator $y(t)$ is that once y leaves its equilibrium value $y = 0$, it experiences a restoring force $F_r = -ky$. This force pushes y back towards its equilibrium. Hooke's law says that this holds true for a spring-mass system if the displacement y is small.

Often there is an additional damping force F_d, often due to some type of friction. This force is usually proportional to the y' (the *velocity*), is always in the opposite direction of y', and represents energy leaving the system. (You can think of it as drag.) Thus we have $F_d = -\gamma y'$, where $\gamma \geq 0$ is constant. We may also need to consider an additional external force $f(t)$, or a driving force, that is interacting with our spring-mass system.

By using Newton's law we obtain

$$\begin{aligned} ma = F &= F_r + F_d + f(t), \\ my'' &= -ky - \gamma y' + f(t). \end{aligned}$$

Simple harmonic oscillators

A simple harmonic oscillator is a harmonic oscillator that is not damped ($\gamma = 0$), and is free ($f = 0$) rather than forced ($f \neq 0$). A simple harmonic oscillator can described by the IVP

$$\begin{aligned} &my'' + ky = 0, \\ &y(0) = y_0, \quad y'(0) = y_0'. \end{aligned}$$

The solution of this IVP is $y = c_1 \cos(\omega_0 t) + c_2 \sin(\omega_0 t)$ where $\omega_0 = \sqrt{k/m}$ is the natural frequency of the oscillator and c_1 and c_2 are determined by applying the initial conditions. This in turn can be written in the form

$$y = A \sin(\omega_0 t + \delta).$$

To solve this IVP using the fourth order Runge Kutta method (RK4), we need to write this system in the form

$$z'(t) = f(t, z(t))$$

We can do this by letting $z_1 = y, z_2 = y'$. Then we have

$$z' = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}' = \begin{bmatrix} z_2 \\ \frac{-k}{m} z_1 \end{bmatrix} = f(z).$$

Problem 3. Use the RK4 method to solve for the simple harmonic oscillator satisfying

$$\begin{aligned} &my'' + ky = 0, \quad 0 \leq t \leq 20, \\ &y(0) = 2, \quad y'(0) = -1, \end{aligned} \tag{1.6}$$

for $m = 1$ and $k = 1$. Note that in your implementation of RK4, the constants K_1, K_2, K_3, and K_4 become vectors with n entries, where n is the number of equations in the first-order system.

Plot your numerical approximation of $y(t)$. Compare this with its numerical approximation when $m = 3$ and $k = 1$. Consider: Why does the difference in solutions make sense physically?

Damped free harmonic oscillators

We now consider damped free harmonic oscillators. These systems are described by the differential equation

$$my''(t) + \gamma y'(t) + ky(t) = 0.$$

For fixed values of m and k, it is interesting to study the effect of the damping coefficient γ.

The roots of the characteristic equation are

$$r_1, r_2 = \frac{-\gamma \pm \sqrt{\gamma^2 - 4km}}{2m}.$$

Note that the real parts of r_1 and r_2 are always negative, and so any solution $y(t)$ will decay over time due to a dissipation of the system energy. There are several cases to consider for the general solution of this equation:

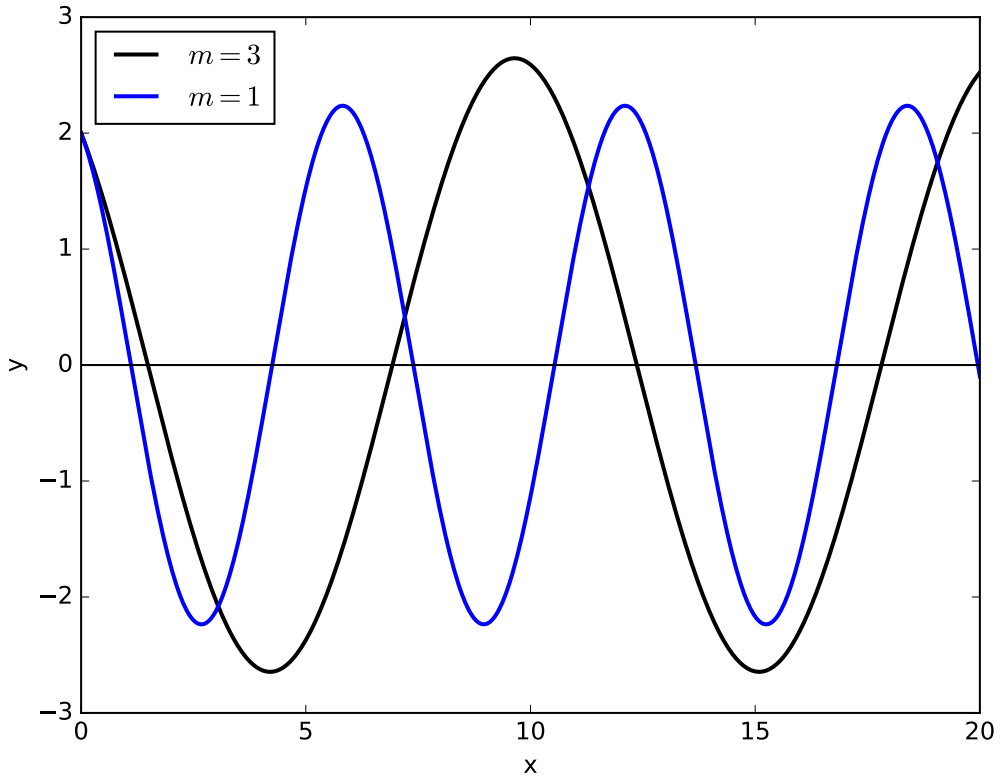

Figure 1.5: Solutions of (1.6) for several values of m.

1. If $\gamma^2 > 4km$, then the general solution is $y(t) = c_1 e^{r_1 t} + c_2 e^{r_2 t}$. Here the system is said to be *overdamped*. Notice from the general solution that there is no oscillation in this case.

2. If $\gamma^2 = 4km$, then the general solution is $y(t) = c_1 e^{\gamma t/2m} + c_2 t e^{\gamma t/2m}$. Here the system is said to be *critically damped*.

3. If $\gamma^2 < 4km$, then the general solution is

$$\begin{aligned} y(t) &= e^{-\gamma t/2m} \left[c_1 \cos(\mu t) + c_2 \sin(\mu t) \right], \\ &= R e^{-\gamma t/2m} \sin(\mu t + \delta), \end{aligned}$$

where R and δ are fixed, and $\mu = \sqrt{4km - \gamma^2}/2m$. This system does oscillate.

Problem 4. Use the RK4 method to solve for the damped free harmonic oscillator satisfying

$$\begin{aligned} &y'' + \gamma y' + y = 0, \quad 0 \le t \le 20, \\ &y(0) = 1, \quad y'(0) = -1. \end{aligned}$$

For $\gamma = 1/2$, and $\gamma = 1$, simultaneously plot your numerical approximations of y. Print $y(20)$

accurate to four significant digits, by checking that the relative error is less than 5×10^{-5}.

Forced harmonic oscillators without damping

Let's look at the systems described by the differential equation

$$my''(t) + ky(t) = F(t). \tag{1.7}$$

In many instances the external force $F(t)$ is periodic, so let us assume that $F(t) = F_0 \cos(\omega t)$. If $\omega_0 = \sqrt{k/m} \neq \omega$, then the general solution of 1.7 is given by

$$y(t) = c_1 \cos(\omega_0 t) + c_2 \sin(\omega_0 t) + \frac{F_0}{m(\omega_0^2 - \omega^2)} \cos(\omega t).$$

If $\omega_0 = \omega$, then the general solution is

$$y(t) = c_1 \cos(\omega_0 t) + c_2 \sin(\omega_0 t) + \frac{F_0}{2m\omega_0} t \sin(\omega_0 t).$$

In the case that $\omega_0 = \omega$, the solution contains a term that grows arbitrarily large as $t \to \infty$. If we included damping then the solution would be bounded, but would still be large for small γ and ω close to ω_0.

Consider a physical spring-mass system. Equation 1.7 holds only for small oscillations (this is where Hooke's law is applicable). For larger oscillations, this equation will not hold. However, the fact that the equation predicts large oscillations suggests the spring-mass system could fall apart as a result of the external force. Mechanical resonance has been known to cause failure of bridges, buildings, and airplanes.

Problem 5. Use the RK4 method to solve for the damped and forced harmonic oscillator satisfying

$$\begin{aligned} &2y'' + \gamma y' + 2y = 2\cos(\omega t), \quad 0 \leq t \leq 40, \\ &y(0) = 2, \quad y'(0) = -1. \end{aligned} \tag{1.8}$$

For the following values of γ and ω, plot your numerical approximations of y and print $y(40)$ accurate to four significant digits: $(\gamma, \omega) = (0.5, 1.5)$, $(0.1, 1.1)$, and $(0, 1)$.

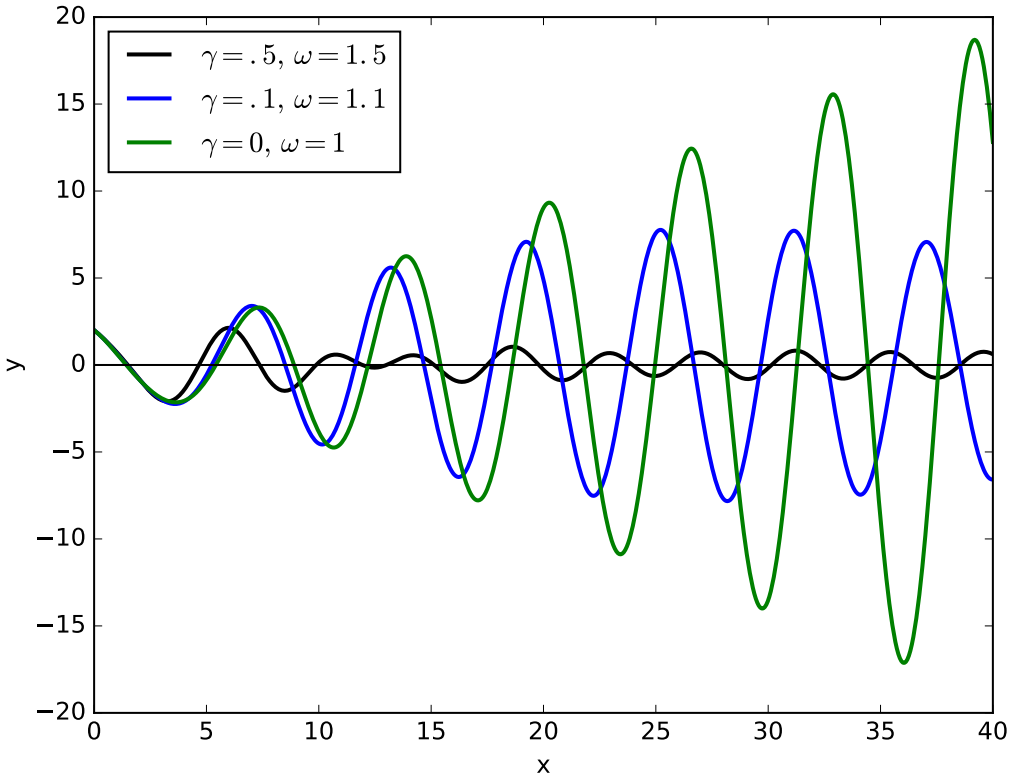

Figure 1.6: Solutions of (1.8) for several values of ω and γ.