6
The perfect (sine) wave

The goals of this chapter are:

- to analyze several methods for discretizing a continuous-time system; and
- to illustrate complex poles and the significance of the unit circle.

How can you compute a sine wave if your programming language did not have a built-in sine function? You can use the coupled oscillator from the first problem set:

$$\dot{y}_1 = y_2,$$
$$\dot{y}_2 = -y_1.$$

Let's rewrite the equations to have physical meaning. Imagine y_1 as the oscillator's position x. Then y_2 is $\dot{x}$, which is the oscillator's velocity. So replace y_1 with x, and replace y_2 with v. Then $\dot{y}_2$ is the acceleration a, making the equations

$$\dot{x} = v,$$
$$a = -x.$$

The first equation is a purely mathematical definition, so it has no physical content. But the second equation describes the acceleration of an ideal spring with unit spring constant and unit mass. So the position $x(t)$ is a linear combination of $\sin t$ and $\cos t$. An accurate discrete-time simulation, if we can make one, would reproduce these sinusoidal oscillations, and this chapter shows you how.

To turn the system into a discrete-time system, let T be the time step, $x[n]$ be the discrete-time estimate for the position $x(nT)$, and $v[n]$ be the discrete-time estimate for the velocity $v(nT)$. Those changes take care of all the terms except for the derivatives. How do you translate the derivatives? Three methods are easy to use, and we try each, finding its poles and analyzing its fidelity to the original, continuous-time system.

6.1 Forward Euler

The first method for translating the derivatives is the forward-Euler approximation. It estimates the continuous-time derivatives $\dot{x}(nT)$ and $\dot{v}(nT)$ using the forward differences

$$\begin{aligned}\dot{x}(nT) &\to \frac{x[n+1]-x[n]}{T},\\ \dot{v}(nT) &\to \frac{v[n+1]-v[n]}{T}.\end{aligned}$$

Then the continuous-time system becomes

$$\begin{aligned}x[n+1]-x[n] &= Tv[n],\\ v[n+1]-v[n] &= -Tx[n].\end{aligned}$$

The new samples $x[n+1]$ and $v[n+1]$ depend only on the old samples $x[n]$ and $v[n]$. So this system provides an explicit recipe for computing later samples.

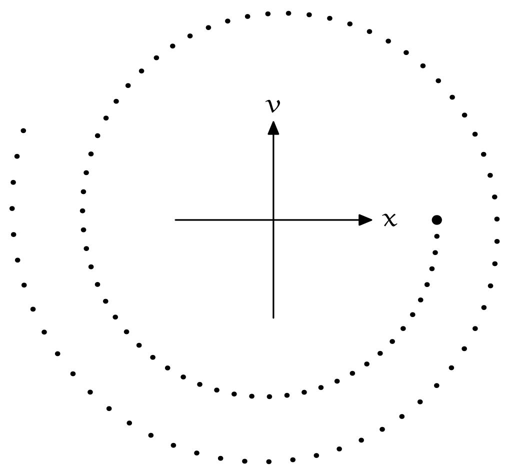
6.1.1 Simulation

Here is Python code to implement these equations. It starts the system with the initial conditions $x[0] = 1$ and $v[0] = 0$ and plots v vs x.

```
from scipy import *
import pylab as p

T = 0.1
```

```
N = int(T**-2)
x = zeros(N)
x[0] = 1
v = zeros(N)

for n in range(N-1):
    x[n+1] = x[n] + T*v[n]
    v[n+1] = v[n] - T*x[n]

p.plot(x,v,'r.')
p.show()
```

Here is the plot (generated with MetaPost rather than Python, but from the same data):



where the $n = 0$ sample is marked with the prominent dot.

> *Pause to try 26.* What would the plot look like for an exact solution of the continuous-time differential equations?

When the solution generates a true sine wave, which the continuous-time equations do, then the plot is of $v(t) = -\sin t$ versus $x(t) = \cos t$, which is a circle.

Since the discrete-time response is a growing spiral, it does not accurately represent the continuous-time system. Indeed after 50 or so time steps, the points have spiraled outward significantly. The spiraling signifies that $x[n]$ and $v[n]$ individually are not only oscillating, they are also growing.

6.1.2 Analysis using poles

To explain why the system oscillates and grows, find its poles. First, turn the two first-order equations into one second-order equation. Then find the poles of the second-order system. This method avoids having to understand what poles mean in a coupled system of equations.

To convert to a second-order system, use the first equation to eliminate v from the second equation. First multiply the second equation by T to get

$$Tv[n+1] - Tv[n] = -T^2x[n].$$

Now express $Tv[n+1]$ using $x[n]$ and $v[n]$; and $Tv[n]$ using $x[n-1]$ and $v[n-1]$. The forward-Euler replacements are

$$\begin{aligned} Tv[n+1] &= x[n+2] - x[n+1], \\ Tv[n] &= x[n+1] - x[n]. \end{aligned}$$

Making these replacements in $Tv[n+1] - Tv[n] = -T^2x[n]$ gives

$$\underbrace{(x[n+2] - x[n+1])}_{Tv[n+1]} - \underbrace{(x[n+1] - x[n])}_{Tv[n]} = -T^2x[n].$$

Collect like terms to get:

$$x[n+2] - 2x[n+1] + (1+T^2)x[n] = 0.$$

To get a system functional, we should have included an input signal or forcing function F. Physically, the forcing function represents the force driving the spring. Here is one way to add it:

$$x[n+2] - 2x[n+1] + (1+T^2)x[n] = f[n+2],$$

The system functional is:

$$\frac{X}{F} = \frac{1}{1 - 2\mathcal{R} + (1+T^2)\mathcal{R}^2}.$$

To find the poles, factor the denominator $1 - 2\mathcal{R} + (1+T^2)\mathcal{R}^2$ into the form $(1-p_1\mathcal{R})(1-p_2\mathcal{R})$.

Pause to try 27. Where are poles p_1 and p_2?

The quadratic formula is useful in finding p_1 and p_2, but too often it substitutes for, and does not augment understanding. So here is an alternative, intuitive analysis to find the poles. First expand the generic factored form:

$$(1 - p_1\mathcal{R})(1 - p_2\mathcal{R}) = 1 - (p_1 + p_2)\mathcal{R} + p_1p_2\mathcal{R}^2.$$

Now match this form to the particular denominator $1 - 2\mathcal{R} + (1 + T^2)\mathcal{R}^2$. The result is

$$\begin{aligned} p_1 + p_2 &= 2, \\ p_1p_2 &= 1 + T^2. \end{aligned}$$

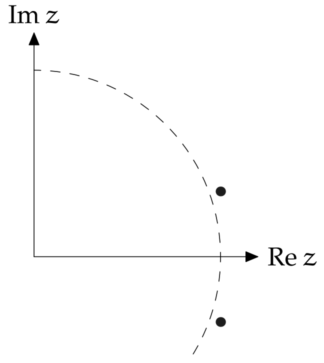
The sum of the roots is 2 while the product is greater than 1.

Pause to try 28. Show that these conditions are impossible to meet if p_1 and p_2 are real.

Let $p_1 = 1 + a$ and $p_2 = 1 - a$, which ensures that $p_1 + p_2 = 2$. Then $p_1p_2 = 1 - a^2$, which cannot be greater than 1 if a is real. So a must be imaginary. The resulting poles are:

$$p_{1,2} = 1 \pm jT$$

and are marked on the z-plane.



The poles are not on the positive real axis, which means that they produce oscillating outputs. Oscillation is desirable in a simulation of an oscillating continuous-time system. However, both poles lie outside the unit circle! Poles in that region of the z-plane produce growing outputs. So the poles of our system, which lie off the positive real axis and outside the unit circle, produce outputs that oscillate and grow, as shown in the X–V plot.

The forward-Euler method does not produce an accurate approximation to the continuous-time oscillating system.

Exercise 34. To place the poles on the unit circle, why not simulate with $T = 0$?

Exercise 35. On the z-plane, sketch how the poles move as T increases from 0 to 1.

6.2 Backward Euler

Let's find another method. Being lazy, we invent it using **symmetry**. If forward Euler is inaccurate, try backward Euler by estimating the derivatives using backward differences:

$$\begin{aligned}\dot{x}(nT) &\to \frac{x[n]-x[n-1]}{T},\\ \dot{v}(nT) &\to \frac{v[n]-v[n-1]}{T}.\end{aligned}$$

These estimates are left-shifted versions of the forward-Euler estimates.

Then the system of continuous-time equations becomes

$$\begin{aligned}x[n]-x[n-1] &= Tv[n],\\ v[n]-v[n-1] &= -Tx[n].\end{aligned}$$

The new values $x[n]$ and $v[n]$ depend on the new values themselves! This discrete-time system is an implicit recipe for computing the next samples, wherefore the backward Euler method is often called implicit Euler.

6.2.1 Finding an explicit recipe

Being a set of implicit equations, they require massaging to become an explicit recipe that we can program. You can do so with a matrix inversion, but let's do it step by step. The system of equations is

$$\begin{array}{rrcl} x[n] & -Tv[n] & = & x[n-1],\\ Tx[n] & +v[n] & = & v[n-1].\end{array}$$

Eliminate $v[n]$ to get an equation for $x[n]$ in terms of only the preceding samples $x[n-1]$ and $v[n-1]$. To eliminate $v[n]$, multiply the second equation by T then add. The result is

$$(1+T^2)x[n] = x[n-1] + Tv[n-1].$$

Similarly, eliminate $x[n]$ to get an equation for $v[n]$ in terms of only the preceding values $v[n-1]$ and $x[n-1]$. To eliminate $x[n]$, multiply the first equation by T then subtract. The result is

$$(1+T^2)v[n] = v[n-1] - Tx[n-1].$$

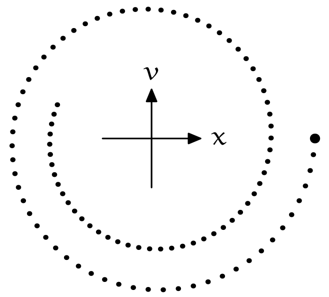
These equations are similar to the forward-Euler equations except for the factors of $1+T^2$. Those factors shrink $x[n]$ and $v[n]$, so they might control the runaway oscillations.

Pause to try 29. Modify the Python program for forward Euler to implement backward Euler, and plot the results.

To implement backward Euler, only two lines of the program need to be changed, the lines that compute the new samples. The code is

```
for n in range(N-1):
    x[n+1] = (x[n] + T*v[n])/(1+T**2)
    v[n+1] = (v[n] - T*x[n])/(1+T**2)
```

and the X–V plot is



Now the points spiral inward! The factor of $1+T^2$ is overkill.

6.2.2 Poles of backward Euler

Let's explain the inward spiral by finding the poles of the system. The first step is to convert the two first-order difference equations into one second-order equation.

Pause to try 30. Find the second-order difference equation.

To convert to a second-order difference equation, eliminate $v[n]$ and $v[n-1]$ by using

$$Tv[n] = x[n] - x[n-1]$$

and by using its counterpart shifted one sample, which is $Tv[n-1] = x[n-1] - x[n-2]$. Make these substitutions into $Tx[n] + v[n] = v[n-1]$ after multiplying both sides by $-T$. Then

$$\begin{aligned} -T^2x[n] &= \underbrace{(x[n] - x[n-1])}_{Tv[n]} - \underbrace{(x[n-1] - x[n-2])}_{Tv[n-1]} \\ &= x[n] - 2x[n-1] + x[n-2]. \end{aligned}$$

Combining the $x[n]$ terms and adding a forcing function F produces this difference equation

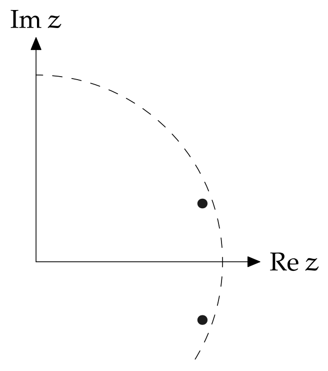
$$(1+T^2)x[n] - 2x[n-1] + x[n-2] = f[n]$$

and this system functional

$$\frac{F}{X} = \frac{1}{(1+T^2) - 2\mathcal{R} + \mathcal{R}^2}.$$

Pause to try 31. Find the poles.

Now find its poles by factoring the denominator. Avoid the quadratic formula! The denominator looks similar to the denominator in forward Euler where it was $1-2\mathcal{R}+(1+T^2)\mathcal{R}^2$, but the end coefficients 1 and $1+T^2$ are interchanged. This interchange turns roots into their reciprocals, so the poles are



$$p_1 = \frac{1}{1+jT} \qquad p_2 = \frac{1}{1-jT}.$$

These poles lie inside the unit circle, so the oscillations die out and the X–V plot spirals into the origin.

Pause to try 32. Do a cheap hack to the program make the points stay on the unit circle. Hint: Add just eight characters to the code.

A cheap hack is to fix the problem manually. If dividing $x[n+1]$ and $v[n+1]$ by $1+T^2$ overcorrected, and dividing by 1 undercorrected, then try dividing by a compromise value $\sqrt{1+T^2}$:

```
for n in range(N-1):
    x[n+1] = (x[n] + T*v[n])/sqrt(1+T**2)
    v[n+1] = (v[n] - T*x[n])/sqrt(1+T**2)
```

However, this hack does not generalize, which is why it is a cheap hack rather than a method. In this problem we can solve the continuous-time system, so we can construct a hack to reproduce its behavior with a discrete-time system. However, for many systems we do not know the continuous-time solution, which is why we simulate. So we would like a principled method to get accurate simulations.

6.3 Leapfrog

Leapfrog, also known as the trapezoidal approximation, is a mixture of forward and backward Euler. Use forward Euler for the x derivative:

$$\dot{x}(nT) \to \frac{x[n+1]-x[n]}{T}$$

The discrete-time equation is, as in forward Euler,

$$x[n+1] - x[n] = Tv[n].$$

Then use backward Euler for the v derivative. So

$$\dot{v}(nT) \to \frac{v[n] - v[n-1]}{T}$$

and

$$v[n] - v[n-1] = -Tx[n]$$

or

$$v[n+1] - v[n] = -Tx[n+1].$$

In this mixed method, the x computation is an explicit recipe, whereas the v computation is an implicit recipe.

6.3.1 Simulation

Fortunately, this implicit recipe, unlike the full backward Euler, has a clean implementation. The system of equations is

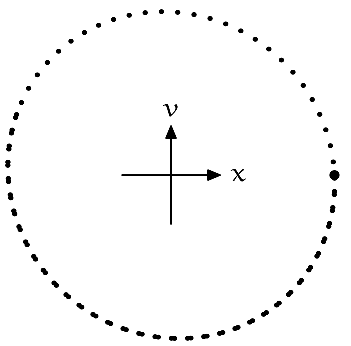
$$\begin{aligned} x[n+1] &= x[n] + Tv[n], \\ v[n+1] &= v[n] - Tx[n+1]. \end{aligned}$$

Pause to try 33. Implement leapfrog by modifying the magic two lines in the Python program.

The only change from forward Euler is in the computation of $v[n+1]$. Leapfrog uses $x[n+1]$, which is the just-computed value of x. So the code is

```
for n in range(N-1):
    x[n+1] = x[n] + T*v[n]
    v[n+1] = v[n] - T*x[n+1]
```

and the plot is



A beautiful circle without $\sqrt{1+T^2}$ hacks!

6.3.2 Analysis using poles

Let's explain that behavior by finding the poles of this system. As usual, convert the two first-order equations into one second-order equation for the position. To eliminate v, use the first equation, that $Tv[n] = x[n+1] - x[n]$. Then $v[n+1] = v[n] - Tx[n+1]$ becomes after multiplying by T:

$$\underbrace{(x[n+2]-x[n+1])}_{Tv[n+1]} - \underbrace{(x[n+1]-x[n])}_{Tv[n]} = -T^2x[n+1].$$

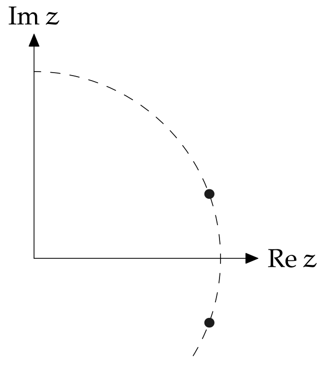
After rearranging and including a forcing function, the result is

$$x[n+2] - (2-T^2)x[n+1] + x[n] = f[n+2].$$

The system functional is

$$\frac{1}{1-(2-T^2)\mathcal{R}+\mathcal{R}^2}.$$

Again factor into the form $(1-p_1\mathcal{R})(1-p_2\mathcal{R})$. The product p_1p_2 is 1 because it is the coefficient of $\mathcal{R}^2$. The sum $p_1 + p_2$ is $2 - T^2$, which is less than 2. So the roots must be complex. A pair of complex-conjugate roots whose product is 1 lie on the unit circle. Poles on the unit circle produce oscillations that do not grow or shrink, wherefore leapfrog produces such a fine sine wave.



Exercise 36. Find the poles of the second-order equation and confirm that they lie on the unit circle.

6.4 Summary

The forward-Euler method is too aggressive. The backward-Euler method is too passive. But, at least for second-order systems, the mixed, forward–backward Euler method (leapfrog) is just right [13].