1

The Damped Spring and Pendulum Problems

1.1 Derivation of the Damped Spring and Pendulum Equations

In this chapter we present some simple ordinary differential equation problems to give students a chance to become familiar with PDE2D before proceeding to more difficult problems in later chapters.

Like many second-order differential equations, the equations used to model the damped spring and pendulum are derived using Newton's second law: Mass times acceleration equals the force acting on the mass.

Suppose a weight of mass m hangs from the ceiling on a spring. We will let $y(t)$ be the height of this weight, with $y = 0$ taken as its height when stationary. Then we will consider three forces acting on this mass: The force of the spring itself will be approximately proportional to the displacement from equilibrium and in the opposite direction, $-ky$; a force of friction (perhaps due to the surrounding air or liquid, or the spring itself) approximately proportional to the velocity of the mass and in the opposite direction, $-by'$; and an additional external force, $f(t)$, which may be caused by some outside agent such as a magnetic force. Thus, according to Newton's second law

$$my'' = -ky - by' + f(t) \tag{1.1}$$

Why didn't we include the force of gravity on this mass? Well, actually we did, by taking $y = 0$ to be the height when there is no force acting on the mass *other than gravity*, see Problem 1.

Since this is a second-order equation, we will need two initial conditions, we need to specify the initial position $y(0)$ and the initial velocity $y'(0)$.

The kinetic energy associated with the spring mass is $\frac{1}{2}m(y')^2$. Its potential energy is $\frac{1}{2}ky^2$, because that is how much energy is required to move it a distance y from equilibrium, against an opposing force increasing linearly from 0 to ky, thus against an average opposing force of $\frac{1}{2}ky$. So the total (kinetic plus

Solving Partial Differential Equation Applications with PDE2D, First Edition. Granville Sewell.

potential) energy of the mass is $E(t) = \frac{1}{2}m(y')^2 + \frac{1}{2}ky^2$. Then if there is no external force adding energy to the system, $E'(t) = my'y'' + kyy' = y'(my'' + ky) = -b(y')^2$, so the total energy is constant if there is no friction and decreases if there is (until a steady state has been reached, $y' = 0$).

Now let's consider a mass m suspended by a rigid wire of length L fastened at the origin. Let $(x(t), y(t))$ be the position of the mass, and we assume that the only forces acting on this mass are the force of gravity, $(0, -mg)$, and the wire itself, which exerts a force of magnitude equal to the tension in the wire, $\lambda(t)$, in the direction of the unit vector toward the origin, $(-x(t)/L, -y(t)/L)$. Newton's second law gives us the pendulum equations:

$$\begin{aligned} mx'' &= -\lambda x/L \\ my'' &= -\lambda y/L - mg \end{aligned}$$

If we break the two second-order equations into first-order equations by defining $u \equiv x', v \equiv y'$ and add the constraint that $x^2 + y^2 = L^2$, we get five equations for the five unknown functions x, y, u, v, λ :

$$\begin{aligned} x' &= u \\ y' &= v \\ mu' &= -\lambda \frac{x}{L} \\ mv' &= -\lambda \frac{y}{L} - mg \\ 0 &= x^2 + y^2 - L^2 \end{aligned} \tag{1.2}$$

The total energy of the pendulum is the kinetic energy plus the potential energy:

$$E(t) = \frac{1}{2}m[(x')^2 + (y')^2] + mgy \tag{1.3}$$

This total energy should remain constant, since

$$\begin{aligned} E'(t) &= mx'x'' + my'y'' + mgy' = x'\left(-\lambda\frac{x}{L}\right) + y'\left(-\lambda\frac{y}{L} - mg\right) + mgy' \\ &= -\frac{\lambda}{L}(xx' + yy') = -\frac{\lambda}{2L}(x^2 + y^2)' = 0 \end{aligned}$$

Problem (1.2) is a differential-algebraic system because the last equation has no derivatives. In fact it is given as an example of an "index 3" system in the documentation for IMSL Library differential-algebraic system solver DAESL (IMSL, Inc. 2010). The documentation says it cannot be solved by DAESL until the last equation has been differentiated twice by hand. So this simple-looking system is not quite as easy to solve as it appears.

1.2 Damped Spring and Pendulum Examples

Example 1.1 (Damped spring) We first solve the damped spring equation (1.1) using PDE2D, with $m = 1, b = 2, k = 101, f(t) = 0$, and initial conditions $y(0) = 0, y'(0) = 10$. This is a "0D" (no space variables), time-dependent problem, but PDE2D only handles first derivatives in time, so it must be written as a system of two first-order ordinary differential equations:

$$y' = v$$
$$mv' = -ky - bv$$

A plot of $y(t)$ is shown in Figure 1.1. Notice the solution dies out, due to the frictional force, as it oscillates. The exact solution is $y(t) = e^{-t}sin(10t)$.

Example 1.2 (Pendulum) We next solve the pendulum equations (1.2), with $m = 4, L = 1, g = 32$, and initial conditions $x(0) = L, y(0) = 0, u(0) = 0, v(0) = 0, \lambda(0) = 0$, which means we start with the pendulum horizontal, 90° to the right of its low point. This "index 3" differential-algebraic system is indeed difficult to solve for most finite difference methods: When we asked PDE2D to choose a time step adaptively (ADAPT=.TRUE.), it gave a warning every step that it had taken the minimum step size and still could not satisfy the default error tolerance, and when we requested a very small constant time step, using a second-order Adam's implicit method (CRANKN=.TRUE.), there

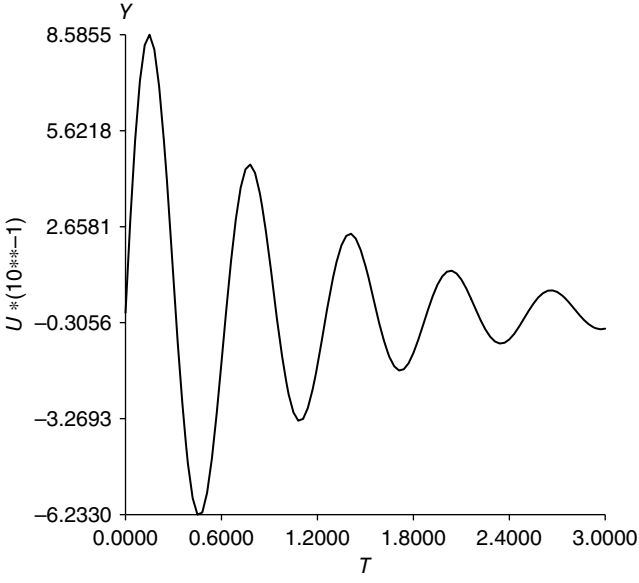

Figure 1.1 Damped spring oscillation and decay, Example 1.1.

were huge oscillations. But with ADAPT=.FALSE.,CRANKN=.FALSE., and NSTEPS=1000, which means 1000 steps are taken with a simple backward Euler method, we got the reasonable results shown in Figure 1.2. The backward Euler method is actually ideally suited for differential-algebraic systems (except of course that it is only $O(dt)$ accurate!) and should always be selected when solving mixed time-dependent and steady-state PDEs with PDE2D.[1]

Figure 1.2a shows a pendulum whose oscillations are dying out, but there was no damping included in model (1.2); the total energy in this model should be constant. The damping is due to the low-order backward Euler method's well-known tendency to dissipate energy, and rerunning with NSTEPS=100 000 produced much less damping, as seen in Figure 1.2b.

1.3 Problems

1 (Effect of gravity on spring) Suppose we add gravity as an external force to Eq. (1.1), that is, suppose $my'' = -ky - by' - mg + f(t)$. That will displace the mass downward at rest a distance mg/k (force divided by the spring constant.) Now define $z \equiv y + mg/k$, so the new rest position $y = -mg/k$ corresponds to $z = 0$. Show that Eq. (1.1) becomes $mz'' = -kz - bz' + f(t)$, so this equation really does take gravity into account, as long as $z = 0$ ($y = 0$ in the original equation) means the rest height with gravity on.

2 (Resonance in spring) Solve Eq. (1.1) with $m = 1, b = 0, k = 16, f(t) = sin(\omega t), y(0) = 0, y'(0) = 0$. Solve first with $\omega = 3$, then with $\omega = 4$, and plot the solution $y(t)$ as a function of time (see Figure 1.3). Use the PDE2D GUI ("pde2d_gui *name*") to create your program and "runpde2d *name*" to run the program.

Equation (1.1), with external force $f(t) = sin(\omega t)$, can be solved analytically. If $b^2 < 4mk$, the general solution is

$$y(t) = C_1 e^{-\alpha t} sin(\beta t) + C_2 e^{-\alpha t} cos(\beta t) + \frac{sin(\omega t - \phi)}{\sqrt{(k - m\omega^2)^2 + (b\omega)^2}}$$

where $\alpha = b/(2m), \beta = \sqrt{4mk - b^2}/(2m), \phi = tan^{-1}[b\omega/(k - m\omega^2)]$. From this we see that if the frictional coefficient b is small, and ω is close to the resonant frequency $\sqrt{k/m}$ (= 4 in this problem), the solution will have an oscillating term of frequency ω, with a large amplitude. If, as in your problem, $b = 0$ and $\omega = \sqrt{k/m}$, the denominator in the analytical solution given above is zero, so it is no longer a valid solution. What is the general solution then?

1 For a system of equations $c_i u_i' = f_i(t, u_1, ..., u_N)$, backward Euler is $c_i[u_i^{n+1} - u_i^n]/dt = f_i(t^{n+1}, u_1^{n+1}, ..., u_N^{n+1})$. If the i-th equation is algebraic, $c_i = 0$, and this becomes simply $0 = f_i(t^{n+1}, u_1^{n+1}, ..., u_N^{n+1})$, which means the algebraic equation is enforced exactly every step.

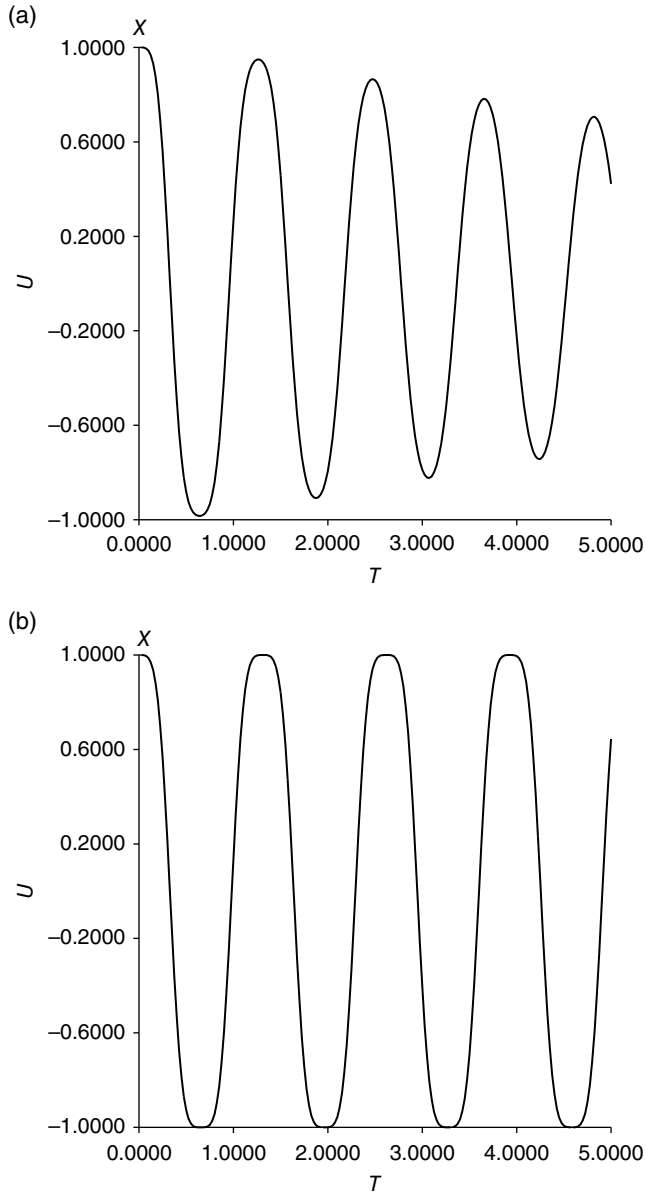

Figure 1.2 Plot of $X(t)$, pendulum Example 1.2. (a) NSTEPS=1000 and (b) NSTEPS=100 000.

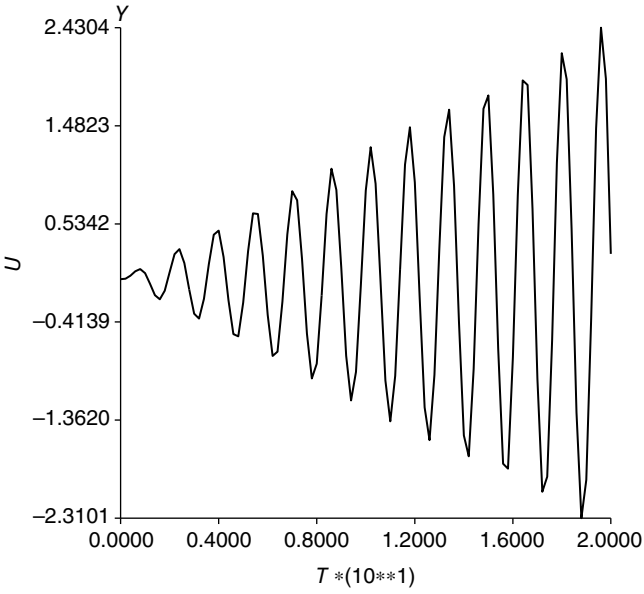

Figure 1.3 Resonance when $\omega = 4$, Problem 2.

3 (Looping pendulum) Use the GUI to recreate the pendulum Example 1.2. Do not request an adaptive time step. The GUI will set CRANKN=. TRUE. by default; you need to change this to .FALSE. with an editor (if you use the interactive driver, it will give you a choice). The GUI will set up time plots for all five variables; replace (using one of the elements of UPRINT(*) in PMOD8Z) one of these with a plot of the total energy (1.3), or else output the "integral" (for 0D problems, integral = value) of the total energy. You will see that the energy continually decreases, but with a large enough value for NSTEPS, you can slow this artificial damping.

Now instead of just dropping the pendulum from its horizontal position, give it a shove downward, that is, reset $v(0) = y'(0) = -7.5$. The pendulum will now go *past* the horizontal position on the other side and start upward, but as seen in Figure 1.4, it will not have enough energy to reach the top. (The tension can be negative because the mass is attached to a wire, not a string!) Rerun with a little stronger downward shove, and the pendulum will start looping, that is, $X(t)$ will keep increasing past zero, instead of turning back. Using the total energy formula (1.3), you can calculate exactly how big $v(0)$ needs to be for the pendulum to have enough energy to reach the top, assuming no artificial energy dissipation.

Increase $|v(0)|$ even further, until the tension remains positive even when $y > 0$ due to the centrifugal force.

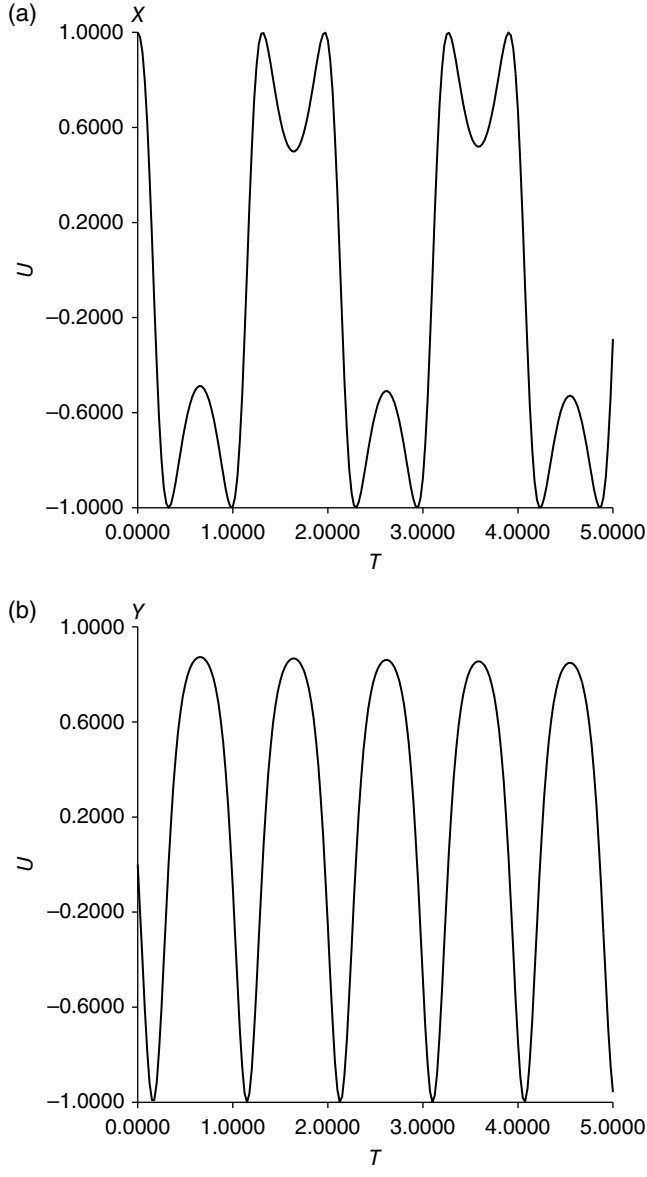

Figure 1.4 Pendulum Problem 3, $V(0) = -7.5$. (a) $X(t)$ and (b) $Y(t)$.

4 (Complex differential equation) Solve the differential equation $\frac{du}{dt} = u^{11}$, $u(0) = 1 + i$. Because of the complex initial condition, the solution will be complex, and so this must be broken into two real differential equations by defining $u = UR + i * UI$, $\frac{dUR}{dt} + i\frac{dUI}{dt} = (UR + i * UI)^{11}$, $UR(0) = 1$, $UI(0) = 1$. You could multiply out $(UR + i * UI)^{11}$ by hand and separate into real and imaginary parts, but it is much easier to let Fortran intrinsic functions do the work for you. Write the two equations as

$$\frac{dUR}{dt} = DREAL(DCMPLX(UR, UI) ** 11)$$
$$\frac{dUI}{dt} = DIMAG(DCMPLX(UR, UI) ** 11)$$

Compare your PDE2D solution with the exact solution, which is [2]

$$u = Re^{i\theta}$$
$$UR = R\ cos(\theta)$$
$$UI = R\ sin(\theta)$$
$$\text{where} \quad R = \sqrt{2}/[1 + (320t)^2]^{\frac{1}{20}}$$
$$\theta = \frac{\pi}{4} + \frac{1}{10} tan^{-1}(320t)$$

You may output the "integral" (=value) of, say, $|UR - R\ cos(\theta)| + |UI - R\ sin(\theta)|$ to measure the error.

This technique for solving complex PDEs has been used to solve some difficult complex applications with PDE2D (Alidoust, Sewell, and Linder 2012; Alidoust and Linder 2013). See also Example 7.2.

5 (Finding eigenvalues) In this section we have solved some ordinary differential equations systems, which are considered by PDE2D to be 0D time-dependent problems. PDE2D also solves 0D steady-state problems, that is, linear or nonlinear algebraic systems, and 0D eigenvalue problems, that is, matrix or generalized matrix eigenvalue problems. Use the GUI to find the eigenvalue closest to $p = 5$ and the associated eigenvector of the algebraic eigenvalue problem:

$$-5 * X1 + 7 * X2 + 3 * X3 + 4 * X4 - 8 * X5 = \lambda * X1$$
$$5 * X1 + 8 * X2 + 3 * X3 + 6 * X4 + 8 * X5 = \lambda * X2$$
$$3 * X1 - 7 * X2 + 9 * X3 - 4 * X4 + 5 * X5 = \lambda * X3$$
$$-3 * X1 + 4 * X3 + 5 * X4 + 3 * X5 = \lambda * X4$$
$$7 * X1 + 4 * X2 + 5 * X3 + 9 * X4 + 5 * X5 = \lambda * X5$$

2 Using separation of variables and the initial condition, it is easy to find that $u^{10} = 32(-320t + i)/[1 + (320t)^2]$. But since there are 10 *different* 10th roots of the complex number on the right-hand side, you have to be careful to get the one that really satisfies the initial condition!

The inverse power method will be used to find the eigenvalue closest to p (called EV0R in the PDE2D program), and the associated eigenvector can be output from POSTPR. Then reset ITYPE to 4 in the Fortran program, and PDE2D will use the shifted QR method to find all five eigenvalues without eigenvectors. (Answer: $13.1406621 \pm 4.9368807\,i$, $-4.5805667 \pm 6.9420509\,i$, 4.8798093)

6 (Boundary value problem) Consider the boundary value problem:

$$(D(x)W_x)_x = 2 + 2\delta(x-1)$$
$$W(-1) = 0$$
$$W(2) = 0$$

where $D(x)$ is discontinuous: $D(x) = 10$ for $x < 0$ and $D(x) = 1$ for $x \geq 0$, and δ is the Dirac delta function. Although this is an ordinary differential equation, it is considered by PDE2D to be a "1D" steady-state problem. Because of the discontinuous $D(x)$ and the Dirac delta, the Galerkin method must be used, so you need to create your program using the interactive driver ("pde2d *name*"). You can write Fortran functions `D(x)` and `True(x)` at the end of your PDE2D program that may look like this:

```
function D(x)
implicit double precision (a-h,o-z)
if (x < 0) then
   D = 10
else
   D = 1
endif
return
end

function True(x)
implicit double precision (a-h,o-z)
if (x < 0) then
   True = x**2/10.d0 - (5.9d0*x + 8)/21.d0
else if (x < 1) then
   True = x**2 - (59*x + 8)/21.d0
else
   True = x**2 - (17*x + 50)/21.d0
endif
return
end
```

Function "True" returns the known exact solution of this problem. You should request that the integral of "abs(W-true(x))" be calculated to

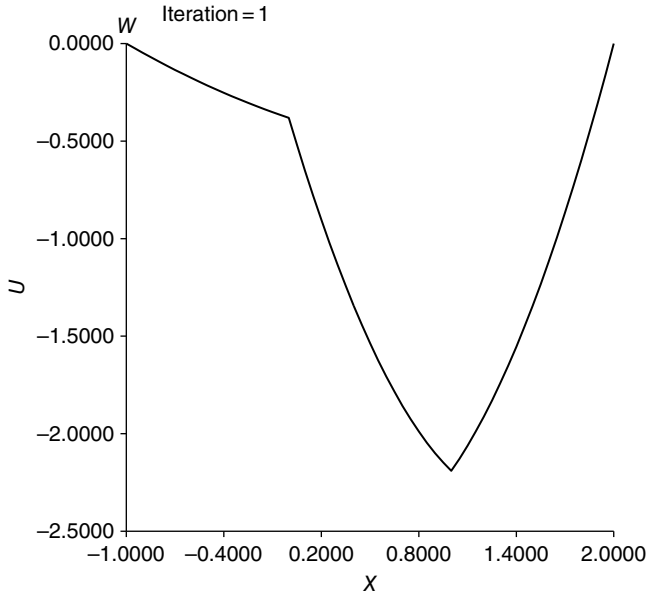

Figure 1.5 Boundary value Problem 6 solution.

measure the error. You will find that if you use second-degree elements or higher (IDEG=2,3, or 4), the error is down to roundoff error (since the solution is a piecewise quadratic polynomial) even if you only have three elements (NXGRID=4), provided you put a gridpoint at $x = 0$, where $D(x)$ and W_x are discontinuous, and another at $x = 1$, where the Dirac delta function is infinite, making W_x discontinuous there also (Figure 1.5). In Section I.5 some hints for programmers less familiar with Fortran are provided, which reference the above functions.

As an alternative to adding Fortran functions at the end of your program, you can define variables "D" and "True" by adding the above IF blocks in-line, when prompted by the interactive driver or later with an editor.