© 2007 WILEY-VCH Verlag GmbH & Co

26
Solitons; KdeV and Sine-Gordon

This chapter examines how the inclusion of dispersion and nonlinearity affect wave behavior. Because these subjects are not often covered in traditional physics classes, we give more background materials than we normally do. We start with a linear chain of coupled pendulums, which should make it clear how the dispersion and nonlinearity arise physically, and then go to the continuum limit to obtain a differential equation of motion. Next we see how soliton *waves arise in both one and two dimensions. Although solitons were originally discovered analytically, they were rediscovered computationally in recent times, and are now an active area of research.*

26.1
Chain of Coupled Pendulums (Theory)

In 1955, Fermi, Ulam, and Pastu were investigating how a 1D chain of coupled oscillators disperses waves. Since waves of differing frequencies traveled through the chain with differing speeds, a pulse broadens as time progresses, as each of its components travel with a different speed. Surprisingly, when the oscillators were made more realistic by introducing a nonlinear term into Hooke's law

$$F(x) \simeq -k(x + \alpha x^2) \tag{26.1}$$

they found that even in the presence of dispersion, a sharp pulse in the chain would survive indefinitely. Your **problem** is to explain how this combination of dispersion and nonlinearity can combine to produce a stable pulse.

Since we have already studied nonlinear effects in a single pendulum (Chapter 19), we take as our model a 1D chain of identical pendulums connected by a torsion bar (Fig. 26.1). The angle θ_i measures the displacement of pendulum i from its equilibrium position. If all the pendulums are set off swinging together, $\theta_i \equiv \theta_j$, the coupling torques would vanish and we would have our old friend, the equation for a realistic (albeit, very thick) pendulum. We assume that three torques act on each pendulum, a gravitational torque trying to return the pendulum to its equilibrium position, and the two torques from the twisting of the bar to the right and to the left of the pendulum. The

Computationyal Physics. Problem Solving with Computers (2nd edn).
Rubin H. Landau, Manuel José Páez, Cristian C. Bordeianu
Copyright © 2007 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim
ISBN: 978-3-527-40626-5

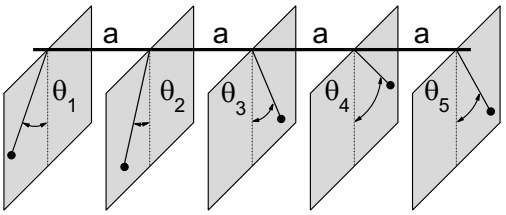

Fig. 26.1 A 1D chain of pendulums coupled with a torsion bar on top. The pendulums swing in planes perpendicular to the length of the bar.

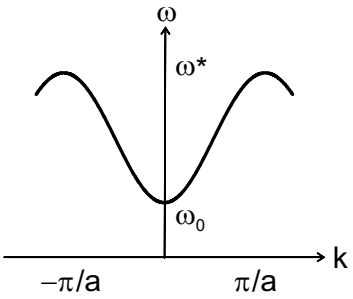

Fig. 26.2 The dispersion relation for a linearized chain of pendulums.

equation of motion for pendulum j follows from Newton's law for rotational motion:

$$\sum_{j\neq i}\tau_{ji} \;=\; I\frac{d^2\theta_j(t)}{dt^2} \tag{26.2}$$

$$-\kappa(\theta_j-\theta_{j-1})-\kappa(\theta_j-\theta_{j+1})-mgL\sin\theta_j \;=\; I\frac{d^2\theta_j(t)}{dt^2} \tag{26.3}$$

$$\Rightarrow \qquad \kappa(\theta_{j+1}-2\theta_j+\theta_{j-1})-mgL\sin\theta_j \;=\; I\frac{d^2\theta_j(t)}{dt^2} \tag{26.4}$$

where I is the moment of inertia of each pendulum, L is the length of the pendulum, and κ is the torque constant of the bar. The nonlinearity in (26.4) arises from the $\sin\theta \simeq \theta - \theta^3/6$ dependence of the gravitational torque. As it stands, (26.4) is a set of coupled nonlinear equations, with the number of equations equal to the number of oscillators.

26.2 Wave Dispersion

Consider a surfer remaining on the crest of a wave. Since she does not see the wave form change with time, her position is given by a function of the form

$f(kx - \omega t)$. Consequently, to her the wave has a constant phase

$$kx - \omega t = \text{constant} \qquad \Rightarrow \qquad x = \omega t/k = \text{constant} \tag{26.5}$$

The surfer's (phase) velocity is the rate of change of x with respect to time,

$$v_p \;=\; \frac{dx}{dt} \;=\; \frac{\omega}{k} \tag{26.6}$$

which is constant. In general, the frequency ω may be a nonlinear function of k, in which case the phase velocity varies with frequency and we have *dispersion*. If the wave contained just one frequency, then you would not observe any dispersion, but if wave was a pulse composed of many Fourier components, then it would broaden and change shape in time as each frequency moved with a differing phase velocity. There is no loss of energy due to dispersion, but the energy does disperse itself into more frequencies with time.

The functional relation between frequency ω and the wave vector k is called a *dispersion relation* (Fig. 26.2. Information in a wave is often transmitted via pulses, with each pulse containing a group of Fourier components. If the Fourier components are centered around a mean frequency ω_0, then the pulse (information) travels, not with the phase velocity, but with the *group velocity*

$$v_g \;=\; \left.\frac{\partial \omega}{\partial k}\right|_{\omega_0} \tag{26.7}$$

When there is dispersion, the group and phase velocities may differ.

To isolate the dispersive aspect of (26.4), we examine at its linear version

$$\frac{d^2\theta_j(t)}{dt^2} + \omega_0^2\theta_j(t) \;=\; \frac{\kappa}{I}(\theta_{j+1} - 2\theta_j + \theta_{j-1}) \tag{26.8}$$

where $\omega_0 = \sqrt{mgL/I}$ is the natural frequency for any one pendulum. Because we want to determine if a wave with a single frequency propagates on this chain, we test if a traveling-wave with frequency ω and wavelength λ,

$$\theta_j(t) \;=\; Ae^{i(\omega t - kx_j)} \qquad\qquad k = \frac{2\pi}{\lambda} \tag{26.9}$$

is a solution. Substitution of (26.9) into the wave equation (26.8) produces the *dispersion relation* (Fig. 26.2):

$$\omega^2 = \omega_0^2 - \frac{2\kappa}{I}(1 - \cos ka) \qquad\qquad \text{(dispersion relation)} \tag{26.10}$$

To have dispersionless propagation (all frequencies propagate with the same velocity), we need a linear relation between ω and k:

$$\lambda = c\frac{2\pi}{\omega} \qquad \Rightarrow \quad \omega = ck, \qquad\qquad \text{(dispersionless propagation)} \tag{26.11}$$

This is true for the chain only if ka is small, since then $\cos ka \simeq 1$ and $\omega \simeq \omega_0$.

Not only does the dispersion relation (26.10) change the speed of waves, it actually limits which frequencies can propagate (have real frequencies ω) on the chain. In order to have real k solutions, ω must lie in the range

$$\omega_0 \leq \omega \leq \omega^* \qquad \text{(waves propagation)} \tag{26.12}$$

The minimum frequency ω_0 and the maximum frequency ω^* are related through the limits of $\cos ka$ in (26.10),

$$(\omega^*)^2 \;=\; \omega_0^2 + \frac{4\kappa}{I} \tag{26.13}$$

Waves with $\omega < \omega_0$ do not propagate, while waves with $\omega > \omega^*$ are non-physical because they correspond to wavelengths $\lambda < 2a$, that is, oscillations where there are no particles. It should be clear that these high and low ω cutoffs will change the shape of a propagating pulse.

26.2.1 Continuum Limit, the Sine-Gordon Equation

If the wavelengths in a pulse are much longer than the repeat distance a, that is, if $ka \ll 1$, the chain can be approximated as a continuous medium. In this limit, a becomes the continuous variable x, and the system of coupled ordinary differential equations becomes a single, partial differential equation:

$$\begin{aligned}
\theta_{j+1} \;&\simeq\; \theta_j + \frac{\partial\theta}{\partial x}\Delta x \\
\Rightarrow \qquad (\theta_{j+1} - 2\theta_j + \theta_{j-1}) \;&\simeq\; \frac{\partial^2\theta}{\partial x^2}\Delta x^2 \;=\; \frac{\partial^2\theta}{\partial x^2}a^2 \\
\Rightarrow \qquad \frac{\partial^2\theta}{\partial t^2} - \frac{\kappa a^2}{I}\frac{\partial^2\theta}{\partial x^2} \;&=\; \frac{mgL}{I}\sin\theta
\end{aligned} \tag{26.14}$$

If we measure time in units of $\sqrt{I/mgL}$ and distances in units of $\sqrt{\kappa a/(mgLb)}$, we obtain the standard form of the sine-Gordon equation (SGE)[1]:

$$\frac{1}{c^2}\frac{\partial^2\theta}{\partial t^2} - \frac{\partial^2\theta}{\partial x^2} = \sin\theta \qquad \text{(Nonlinear SGE)} \tag{26.15}$$

where the $\sin\theta$ on the RHS introduces the nonlinear effects.

[1] The name "sine-Gordon" is either a reminder that the SGE is like the Klein–Gordon equation of relativistic quantum mechanics with a $\sin u$ added to the RHS, or a reminder of how clever one can be in thinking up names.

26.3 Analytic SGE Solution

The nonlinearity of the sine-Gordon equation (26.15) makes it hard to solve analytically. The trick is to guess a functional form of a traveling wave and that converts the PDE into an ODE:

$$\theta(x,t) \stackrel{?}{=} \theta(\xi = t \pm x/v) \qquad \Rightarrow \qquad \frac{d^2\theta}{d\xi^2} = \frac{v^2}{v^2-1}\sin\theta \tag{26.16}$$

You should recognize (26.16) as old friend, the equation of motion for the realistic pendulum with no driving force and no friction. The constant v is a velocity in natural units, and separates different regimes of the motion:

$$\begin{array}{lll} v<1: & \text{pendula initially down} & \downarrow\downarrow\downarrow\downarrow\downarrow \quad \text{(stable)}, \\ v>1: & \text{pendula initially up} & \uparrow\uparrow\uparrow\uparrow\uparrow \quad \text{(unstable)} \end{array} \tag{26.17}$$

Even though the equation may b familiar, which does not mean that an analytic solution exists. However, for motion along the separatrix ($E = \pm 1$) we obtain the characteristic *soliton* form,

$$\theta(x-vt) = \begin{cases} 4\tan^{-1}\left(\exp\left[+\frac{x-vt}{\sqrt{1-v^2}}\right]\right), & \text{for } E=1 \\ 4\tan^{-1}\left(\exp\left[-\frac{x-vt}{\sqrt{1-v^2}}\right]\right)+\pi, & \text{for } E=-1 \end{cases} \tag{26.18}$$

This soliton corresponds to a solitary *kink* traveling with $v = -1$ that flips the pendulums around by 2π as it moves down the chain. There is also an *antikink* in which the initial $\theta = \pi$ values are flipped to final $\theta = -\pi$.

26.4 Numeric Solution: 2D SGE Solitons

Although we can solve the 1D SGE equation for soliton-like solutions, we will solve for 1D solitons in our study of the KdeV equation in Section 26.8, and so here solve for 2D solitons. The 2D solitons occur as solutions of the 2D generalization of the SGE equation (26.15):

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = \sin u \qquad \text{(2D SGE)} \tag{26.19}$$

Whereas the 1D SGE describes wave propagation along a chain of connected pendulums, the 2D form describes wave propagation in nonlinear elastic media. Interestingly enough, the same 2D SGE also occurs in quantum field theory, where the soliton solutions have been suggested as models for elementary particles [75–77]. The idea is that, like elementary particles, the solutions are

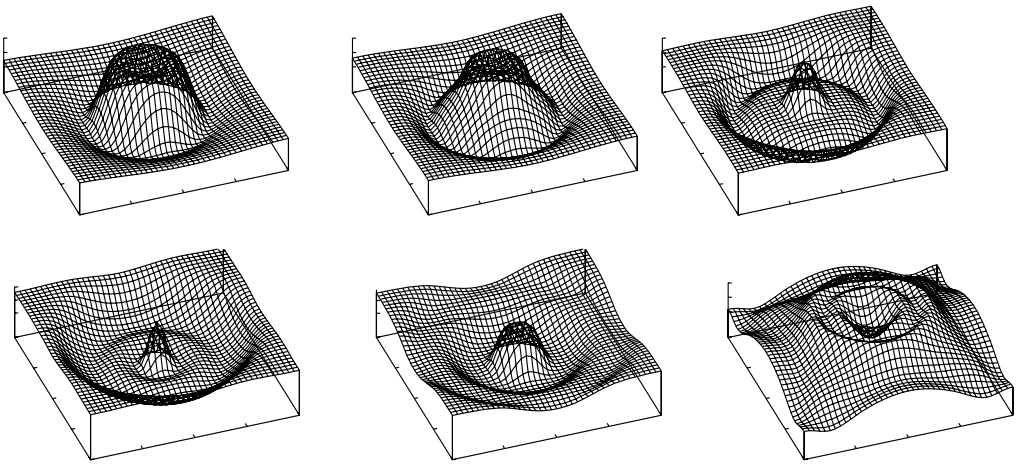

Fig. 26.3 A circular ring soliton at times 8, 20, 40, 60, 80, and 120. This has been proposed as a model for an elementary particle.

confined to a region of space for a long period of time and do not radiate away their energy.

We now have a wave equation containing nonlinear terms to solve. Although we can follow the same procedure used for the linear equation, we want to ensure that any unusual behavior we find arises from the physics and not the algorithm, and so we follow the procedure used in the research literature. We solve in a finite region of 2D space and for positive times:

$$-x_0 < x < x_0 \qquad -y_0 < y < y_0 \qquad 0 \le t \tag{26.20}$$

We take $x_0 = y_0 = 7$ and impose the *boundary conditions* that the derivative of the displacement vanishes at the ends of the region:

$$\frac{\partial u}{\partial x}(-x_0, y, t) = \frac{\partial u}{\partial x}(x_0, y, t) = \frac{\partial u}{\partial y}(x, -y_0, t) = \frac{\partial u}{\partial y}(x, y_0, t) = 0 \tag{26.21}$$

We also impose the *initial condition* that at time $t = 0$ the waveform is that of a pulse (Fig. 26.3) with its surface at rest:

$$u(x, y, t = 0) = 4\tan^{-1}(e^{3-\sqrt{x^2+y^2}}) \qquad \frac{\partial u}{\partial t}(x, y, t = 0) = 0 \tag{26.22}$$

We discretize the equation first by looking for solutions on a space-time lattice:

$$x = m\Delta x \qquad y = l\Delta x \qquad t = n\Delta t \tag{26.23}$$

$$u^n_{m,l} \overset{\text{def}}{=} u(m\Delta x,\ l\Delta x,\ n\Delta t) \tag{26.24}$$

Next we replace the derivatives in (26.19) by their finite-difference approximations to obtain the finite difference SGE:

$$u_{m,l}^{n+1} \simeq -u_{m,l}^{n-1} + 2\left[1 - 2\left(\frac{\Delta t}{\Delta x}\right)^2\right] u_{m,l}^{n} \tag{26.25}$$

$$+ \left(\frac{\Delta t}{\Delta x}\right)^2 \left(u_{m+1,l}^{n} + u_{m-1,l}^{n} + u_{m,l+1}^{n} + u_{m,l-1}^{n}\right)$$

$$- \Delta t^2 \sin\left[\tfrac{1}{4}\left(u_{m+1,l}^{n} + u_{m-1,l}^{n} + u_{m,l+1}^{n} + u_{m,l-1}^{n}\right)\right]$$

To make the algorithm simpler and ensure stability, if we make the time and space steps proportional, $\Delta t = \Delta x/\sqrt{2}$, then all the $u_{m,l}^{n}$ terms drop out:

$$u_{m,l}^{2} \simeq \tfrac{1}{2}\left(u_{m+1,l}^{1} + u_{m-1,l}^{1} + u_{m,l+1}^{1} + u_{m,l-1}^{1}\right)$$

$$- \frac{\Delta t^2}{2} \sin\left[\tfrac{1}{4}\left(u_{m+1,l}^{1} + u_{m-1,l}^{1} + u_{m,l+1}^{1} + u_{m,l-1}^{1}\right)\right] \tag{26.26}$$

Likewise, the discrete form of vanishing initial velocity (26.22) becomes

$$\partial u(x,y,0)/\partial t = 0 \qquad \Rightarrow \qquad u_{m,l}^{2} = u_{m,l}^{0} \tag{26.27}$$

This will be useful in getting the time propagation started.

The lattice points on the edges and corners cannot be obtained from these relations. They are obtained by applying the boundary conditions (26.21):

$$\frac{\partial u}{\partial z}(x_0,y,t) = \frac{u(x+\Delta x,y,t) - u(x,y,t)}{\Delta x} = 0 \tag{26.28}$$

$$\Rightarrow \quad u_{1,l}^{n} = u_{2,l}^{n} \tag{26.29}$$

Similarly, the other derivatives in (26.21) give

$$u_{N_{\max},l}^{n} = u_{N_{\max}-1,l}^{n} \quad u_{m,2}^{n} = u_{m,1}^{n} \quad u_{m,N_{\max}}^{n} = u_{m,N_{\max}-1}^{n} \tag{26.30}$$

where $N_{\max}$ is the number of grid points used for one space dimension.

26.5
2D Soliton Implementation

Listing 26.1: `TwoDsol.java` solves the 2D space plus time SGE for 2D solitons.

```
//  TwoDsol.java: solves Sine-Gordon equation for 2D soliton

import java.io.*;
import java.util.*;

public class TwoDsol {
  public static int D = 201;
  public static double u[][][][] = new double[D + 1][D + 1][4];

  public static void main(String[] argv)
                        throws IOException, FileNotFoundException {
    int nint;
        // input positive integer proportional to time of wave packet
    Scanner sc = new Scanner(System.in);  // Connect Scanner to input
    System.out.printf("Enter a pos integer from 1 (initial time)\n");
    System.out.printf("to 100 for wave packet at that time:\n");
    nint = sc.nextInt();                                  // Read int
    initial(u);                                     // Initialization
    solution(u, nint);                              // Solve equation
  }

  public static void initial(double u[][][][])  {
    double dx, dy, dt, xx, yy, dts, time, tmp;
    int i, j, k;
    dx = 14./200.;
    dy = dx;
    dt = dx/Math.sqrt(2.);
    dts = (dt/dx)*(dt/dx);
    yy = -7.;
    time = 0.;
    for ( i=0; i <= D-1; i++ )  {
      xx = -7.;
      for ( j=0; j <= D-1; j++ )  {
        tmp = 3.-Math.sqrt(xx*xx + yy*yy);
        u[i][j][0] = 4.*Math.atan(tmp);
        xx = xx + dx;
      }
      yy = yy + dy;
  }  }

  public static void solution(double u[][][][], int nint)
                          throws IOException, FileNotFoundException {
    PrintWriter w =
            new PrintWriter(new FileOutputStream("2Dsol.dat"), true);
    double dx, dy, dt, time, a2, zz, dts, a1, tmp;
    int l, m, mm, k, j, i;

    dx = 14./200.; dy = dx;
    dt = dx/Math.sqrt(2.);
    time = 0.;
    time = time + dt;
```

```
    dts = (dt/dx)*(dt/dx);
    tmp = 0.;
    for ( m=1; m <= D-2; m++ )  {
      for ( l=1; l <= D-2; l++ )  {
        a2 = u[m+1][l][0]+u[m-1][l][0] + u[m][l+1][0] + u[m][l-1][0];
        tmp = .25*a2;
        u[m][l][1] = 0.5*(dts*a2-dt*dt*Math.sin(tmp));
      }
    }
    for ( mm=1; mm <= D-2; mm++ )  {    // Borders in second iteration
      u[mm][0][1] = u[mm][1][1];
      u[mm][D-1][1] = u[mm][D-2][1];
      u[0][mm][1] = u[1][mm][1];
      u[D-1][mm][1] = u[D-2][mm][1];
    }
    u[0][0][1] = u[1][0][1];                    // Still undefined terms
    u[D-1][0][1] = u[D-2][0][1];
    u[0][D-1][1] = u[1][D-1][1];
    u[D-1][D-1][1] = u[D-2][D-1][1];
    tmp = 0.;
    for ( k=0; k <= nint; k++ )  {              // Following iterations
      for ( m=1; m <= D-2; m++ )  {
        for ( l=1; l <= D-2; l++ )  {
          a1 = u[m + 1][l][1]+u[m-1][l][1]+u[m][l+1][1]+u[m][l-1][1];
          tmp = .25*a1;
          u[m][l][2] = -u[m][l][0] + dts*a1-dt*dt*Math.sin(tmp);
          u[m][0][2] = u[m][1][2];
          u[m][D-1][2] = u[m][D-2][2];
      } }
      for ( mm=1; mm <= D-2; mm++ )  {
        u[mm][0][2] = u[mm][1][2];
        u[mm][D-1][2] = u[mm][D-2][2];
        u[0][mm][2] = u[1][mm][2];
        u[D-1][mm][2] = u[D-2][mm][2];
      }
      u[0][0][2] = u[1][0][2];
      u[D-1][0][2] = u[D-2][0][2];
      u[0][D-1][2] = u[1][D-1][2];
      u[D-1][D-1][2] = u[D-2][D-1][2];
      for ( l=0; l <= D-1; l++ )  {             // New iterations now
          old
        for ( m=0; m <= D-1; m++ )  {
          u[l][m][0] = u[l][m][1];
          u[l][m][1] = u[l][m][2];
        }
      }
      if (k==nint) {
        for ( i=0; i <= D-1; i=i + 5)  {
          for ( j=0; j <= D-1; j=j + 5)
            { w.println("" + (float)Math.sin(u[i][j][2]/2.) + ""); }
          w.println("  ");
}  }
      time = time + dt;
}  }  }
```

1. Define an array $u[N_{\max}][N_{\max}[3]$ with $N_{\max} = 201$ for the space slots and 3 for the time slots.
2. The solution (26.22) for the initial time $t = 0$ is placed in $u[m][l][1]$.
3. The solution for the second time Δt is placed in $u(m, l, 2)$, and the solution for the next time, $2\Delta t$, is placed in $u[m][l][3]$.
4. Assign the constants, $\Delta x = \Delta y = \frac{7}{100}$, $\Delta t = \Delta x/\sqrt{2}$, $y_0 = x_0 = 7$.
5. Start off at $t = 0$ with the initial conditions and impose the boundary conditions to this initial solution. This is the solution for the first time step, defined over the entire 201×201 grid.
6. For the second time step, increase time by Δt and use (26.26) for all points in the plane. Do not include the edge points.
7. At the edges, for $i = 1, 2, \ldots, 200$, set

$$u[i][1][2] = u[i][2][2] \qquad u[i][N_{\max}][2] = u[i][N_{\max\text{-}1}][2]$$
$$u[1][i][2] = u[2][i][2] \qquad u[N_{\max}][i][2] = u[N_{\max\text{-}1}][i][2]$$

8. To find values for the four points in the corners for the second time step, again use initial condition (26.26):

$$u[1][1][2] = u[2][1][2] \quad u[N_{\max}][N_{\max}][2] = u[N_{\max\text{-}1}][N_{\max\text{-}1}][2]$$
$$u[1][1][N_{\max}] = u[2][N_{\max}][2] \quad u[N_{\max}][1][2] = u[N_{\max\text{-}1}][1][2]$$

9. For the third time step (the future), use (26.26).
10. Continue the propagation forward in time, reassigning the future to the present, and so forth. In this way the solutions for only three time steps need to be stored.

26.6 SGE Soliton Visualization

We see in Fig. 26.3 the time evolution of a circular ring soliton for the stated initial conditions (these results are not critically dependent on the initial conditions). We note that the ring at first shrinks in size, then expands, and then shrinks back into another (but not identical) ring soliton. A small amount of the particle does radiate away, and in the last frame we can notice some interference between the radiation and the boundary conditions. An animation of this sequence can be found on the CD.

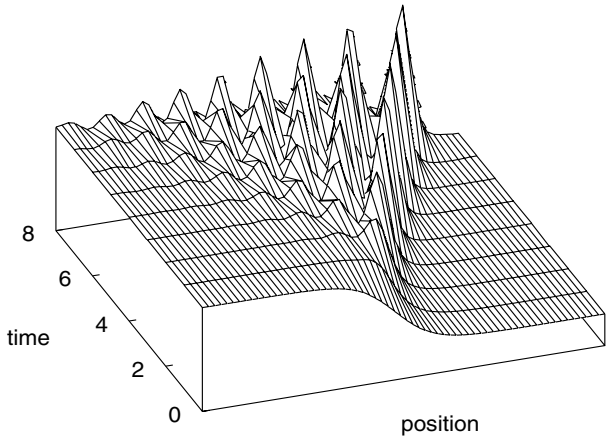

Fig. 26.4 A single two-level waveform at time zero progressively breaks up into eight solitons (labeled) as time increases. The tallest soliton (1) is narrower and faster in its motion to the right.

26.7 Shallow Water (KdeV) Solitons ⊙

In this section we look at soliton water waves. In Section 26.2.1 we looked at soliton solutions of the sine-Gordon equation. We have marked this section as optional because the material is more advanced. Nevertheless, we recommend that everyone at least read through this material because it is fascinating and because the computer has been absolutely essential in the discovery and understanding of solitons. In addition, we recommend that you look at some of the soliton animation we have placed in the Animations folder on the CD. In recognition of the possible newness of this material to many readers, we give additional background and explanatory materials.

Your **problem** is to discover whether nonlinear and dispersive systems can support waves with "particle-like" properties. In a practical sense, your problem is to determine how a tsunami can form from a sudden change in the level of the ocean floor, and then travel over long distances without dispersion or attenuation until it reeks havoc on a distant shore. While a logical response is that systems with dispersion have solutions that broaden in time and thereby lose their identity, consider Fig. 26.4 and the following experimental observation as a **problem** you need to explain. In 1834, J. Scott Russell observed a phenomenon on the Edinburgh–Glasgow canal [78]:

> *I was observing the motion of a boat which was rapidly drawn along a narrow channel by a pair of horses, when the boat suddenly stopped—not so the mass of water in the channel which it had put in motion; it accumulated round the prow of the vessel in a state of violent agitation, then suddenly leaving it behind, rolled forward with great velocity, assuming the form of a large solitary elevation, a rounded, smooth and well-defined heap of water, which*

> *continued its course along the channel apparently without change of form or diminution of speed. I followed it on horseback, and overtook it still rolling on at a rate of some eight or nine miles an hour, preserving its original figure some thirty feet long and a foot to a foot and a half in height. Its height gradually diminished, and after a chase of one or two miles I lost it in the windings of the channel. Such, in the month of August 1834, was my first chance interview with that singular and beautiful phenomenon ....*

Russell also noticed that an initial, arbitrary waveform set in motion in the channel evolves into two or more waves that move at different velocities and progressively move apart until they form individual solitary waves. In Fig. 26.4 we see a single step-like wave breaking up into approximately eight solitons (this shows why these eight solitons are considered the normal modes for this nonlinear systems).

Russell went on to produce these solitary waves in a laboratory and empirically deduced that their speed c is related to the depth h of the water in the canal and to the amplitude A of the wave by

$$c^2 = g(h + A) \tag{26.31}$$

where g is the acceleration due to the gravity. Equation (26.31) implies an effect not found for linear systems, namely, that the waves with greater amplitudes travel faster than those with smaller amplitudes. Notice that this is different from *dispersion* in which waves of different wavelengths have different velocities, but similar to what we have seen with shock waves. The former effect is illustrated in Fig. 26.5, where we see a tall soliton catching up with and passing through a short one.

26.8 Theory: The Korteweg–de Vries Equation

We want to understand these unusual water waves that occur in shallow, narrow channels such as canals [24, 39]. The analytic description of this "heap of water" was given by Korteweg and deVries (KdeV) [79] with the partial differential equation:

$$\frac{\partial u(x,t)}{\partial t} + \varepsilon u(x,t)\frac{\partial u(x,t)}{\partial x} + \mu\frac{\partial^3 u(x,t)}{\partial x^3} = 0 \tag{26.32}$$

The nonlinear term, $\varepsilon u \partial u/\partial t$ leads to a sharpening of the wave and ultimately a *shock* wave. In contrast, the $\partial^3 u/\partial x^3$ term in (26.32) produces broadening. For the proper parameters and initial conditions, the dispersive broadening exactly balances the nonlinear narrowing, and a stable wave is formed.

KdeV solved (26.32) and proved that the speed given by Russell, (26.31), is in fact correct. Seventy years after its discovery, the KdeV equation was rediscovered by Zabusky and Kruskal [80], who solved it numerically and found

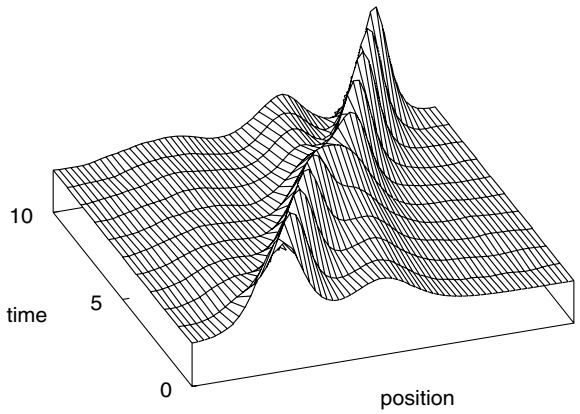

Fig. 26.5 Two shallow-water solitary waves crossing each other. The taller soliton on the left catches up with and overtakes the shorter one at $t \simeq 5$.

that a $\cos(x/L)$ initial condition broke up into eight solitary waves (Fig. 26.4). They also found that the parts of the wave with larger amplitudes move faster than those with smaller amplitudes, which is why the higher peaks tend to be on the right in Fig. 26.4. As if wonders never cease, Zabusky and Kruskal, who coined the name *soliton* for the solitary wave, also observed that the faster peaks actually passed through the slower one unscathed (Fig. 26.5).

26.8.1 Analytic Solution: KdeV Solitons

The trick in analytic approaches to these types of nonlinear equations is to substitute a guessed solution that has the form of a traveling wave,

$$u(x,t) = u(\xi = x - ct) \tag{26.33}$$

This form means that if we move with a constant speed c, we see a constant wave form (yet now the speed will depend on the magnitude of u). There is no guarantee that this form of a solution exists, but it is a lucky guess because substitution into the KdeV equation produces a solvable ODE:

$$-c\frac{\partial u}{\partial \xi} + \epsilon\, u\frac{\partial u}{\partial \xi} + \mu\frac{d^3u}{d\xi^3} = 0 \tag{26.34}$$

While you may find solving this equation for $u(\xi)$ challenging, mathematicians are good at that sort of thing and have come up with the solution

$$u(x,t) = \frac{-c}{2}\text{sech}^2\left[\frac{1}{2}\sqrt{c}(x - ct - \xi_0)\right] \tag{26.35}$$

where ξ_0 is the initial phase. We see in (26.35) an amplitude that is proportional to the wave speed c, and a sech^2 function which gives a single lump-like wave. This is a typical analytic form for a soliton.

26.8.2 Algorithm: KdeV Soliton Solution

The KdeV equation is solved numerically using a finite difference scheme with the time derivative given by a central difference centered at t:

$$\frac{\partial u(x,t)}{\partial t} \simeq \frac{u(x,t+\Delta t) - u(x,t-\Delta t)}{2\Delta t}$$

Likewise, the lowest order expansions of $u(x,t+\Delta t)$ and $u(x,t-\Delta t)$ give

$$\frac{\partial u}{\partial t} \simeq \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta t} \qquad \frac{\partial u}{\partial x} \simeq \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} \qquad x = i\Delta x,\ t = j\Delta t$$

To approximate $\partial^3 u(x,t)/\partial x^3$, we expand $u(x,t)$ to $\mathcal{O}(\Delta t)^3$ about the four points $u(x \pm 2\Delta x, t)$ and $u(x \pm \Delta x, t)$, for example,

$$u(x \pm \Delta x, t) \ \simeq\ u(x,t) \pm (\Delta x)\frac{\partial u}{\partial x} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 u}{\partial^2 x} \pm \frac{(\Delta x)^3}{3!}\frac{\partial^3 u}{\partial x^3} \tag{26.36}$$

We solve this for $\partial^3 u(x,t)/\partial x^3$. Finally, the factor $u(x,t)$ in the second term of (26.32) is taken as the average of three x values all with the same t:

$$u(x,t) \ \simeq\ \frac{u_{i+1,j} + u_{i,j} + u_{i-1,j}}{3} \tag{26.37}$$

These substitutions yield the algorithm for the KdeV equation:

$$u_{i,j+1} \simeq u_{i,j-1} - \frac{\epsilon}{3}\frac{\Delta t}{\Delta x}\left[u_{i+1,j} + u_{i,j} + u_{i-1,j}\right] \tag{26.38}$$

$$\times \left[u_{i+1,j} - u_{i-1,j}\right] - \mu\frac{\Delta t}{(\Delta x)^3}\left[u_{i+2,j} + 2u_{i-1,j} - 2u_{i+1,j} - u_{i-2,j}\right]$$

To apply this algorithm to predict future times, we need to know $u(x,t)$ at present and past times. The initial-time $u_{i,1}$ solution is known for all positions i via the initial condition. To find $u_{i,2}$, we use a forward difference scheme in which we expand $u(x,t)$, keeping only two terms for the time derivative:

$$u_{i,2} \simeq u_{i,1} - \frac{\epsilon\Delta t}{6\Delta x}\left[u_{i+1,1} + u_{i,1} + u_{i-1,1}\right]\left[u_{i+1,1} - u_{i-1,1}\right]$$

$$- \frac{\mu}{2}\frac{\Delta t}{(\Delta x)^3}\left[u_{i+2,1} + 2u_{i-1,1} - 2u_{i+1,1} - u_{i-2,1}\right] \tag{26.39}$$

The keen observer will note that there are still some undefined columns of points, namely, $u_{1,j}$, $u_{2,j}$, $u_{N_{\max}-1,j}$, and $u_{N_{\max},j}$, where $N_{\max}$ is the total number of grid points. A simple technique for determining their values is to assume that $u_{1,2} = 1$ and $u_{N_{\max},2} = 0$. To obtain $u_{2,2}$ and $u_{N_{\max}-1,2}$, assume that $u_{i+2,2} = u_{i+1,2}$ and $u_{i-2,2} = u_{i-1,2}$ (avoid $u_{i+2,2}$ for $i = N_{\max} - 1$, and $u_{i-2,2}$ for $i = 2$). To carry out these steps, approximate (26.39) so that

$$u_{i+2,2} + 2u_{i-1,2} - 2u_{i+1,2} - u_{i-2,2} \rightarrow u_{i-1,2} - u_{i+1,2}$$

The truncation error and stability condition for our algorithm are related by

$$\mathcal{E}(u) = \mathcal{O}[(\Delta t)^3] + \mathcal{O}[\Delta t(\Delta x)^2] \qquad \frac{1}{(\Delta x/\Delta t)}\left[\epsilon|u| + 4\frac{\mu}{(\Delta x)^2}\right] \leq 1 \quad (26.40)$$

The first equation shows that smaller time and space steps lead to smaller truncation error, yet because the roundoff error increases with more steps, the total error does not necessarily decrease (Chapter 3 on errors). Yet we are also limited in how small the steps can be made by the stability condition, which indicates that making Δx too small always leads to instability. Care and experimentation are clearly required to get the algorithm to work just right.

26.8.3 Implementation: KdeV Solitons

Modify or run the program `Soliton.java` that solves the KdeV equation (26.32) for the initial condition:

$$u(x, t = 0) = \frac{1}{2}\left[1 - \tanh\left(\frac{x - 25}{5}\right)\right]$$

with parameters $\epsilon = 0.2$ and $\mu = 0.1$. Start with $\Delta x = 0.4$ and $\Delta t = 0.1$. These constants are chosen to satisfy (26.40) with $|u| = 1$.

Listing 26.2: `Soliton.java` solves the KdeV equation for 1D solitons corresponding to a "bore" initial conditions.

```
// Soliton.java: Solves Kortewg-deVries Equation

import java.io.*;

public class Soliton {
  static double ds = 0.4;                                  // Delta x
  static double dt = 0.1;                                  // Delta t
  static int max = 2000;                                   // Time steps
  static double mu = 0.1;                                  // Mu from KdeV equation
  static double eps = 0.2;                                 // Epsilon from KdeV eq

  public static void main(String[] argv)
                        throws IOException, FileNotFoundException {
```

```
    int i, j, k;
    double a1, a2, a3, fac, time;
    double u[][] = new double[131][3];

    PrintWriter w =                          // Save data in soliton.dat
          new PrintWriter(new FileOutputStream("soliton.dat"), true);
    for ( i=0;  i < 131;  i++ )                    // Initial wave form
      { u[i][0] = 0.5*(1.-((Math.exp(2*(0.2*ds*i - 5.))-1)
                          /(Math.exp(2*(0.2*ds*i - 5.)) + 1))); }
    u[0][1]   = 1.;                                          // End points
    u[0][2]   = 1.;
    u[130][1] = 0.;
    u[130][2] = 0.;
    fac = mu*dt/(ds*ds*ds);
    time = dt;
                                                        // First time step
    for ( i=1;  i < 130;  i++ ) {
      a1 = eps*dt*(u[i + 1][0] + u[i][0] + u[i-1][0]) / (ds*6.);
      if ((i>1) && (i < 129))
        {a2 = u[i + 2][0] + 2.*u[i-1][0] - 2.*u[i + 1][0]-u[i-2][0];}
      else a2 = u[i-1][0] - u[i + 1][0];
      a3 = u[i + 1][0]-u[i-1][0];
      u[i][1] = u[i][0] - a1*a3 - fac*a2/3.;
    }
                                                       // Other time steps
    for ( j=1;  j < max;  j++ )  {
      time += dt;
      for ( i=1;  i < 130;  i++ )  {
         a1 = eps*dt*(u[i + 1][1] + u[i][1] + u[i-1][1]) / (3.*ds);
         if (i>1 && i < 129) a2 = u[i+2][1] + 2.*u[i-1][1]
                                          - 2.*u[i+1][1] - u[i-2][1];
         else a2 = u[i-1][1] - u[i + 1][1];
         a3      = u[i + 1][1] - u[i-1][1];
         u[i][2] = u[i][0] - a1*a3 - 2.*fac*a2/3.;
      }
      for (k=0; k < 131; k++) {u[k][0] = u[k][1]; u[k][1] = u[k][2];}
                                  // gnuplot format, every 200th step
      if ((j%200)==0) {
        for ( k=0;  k < 131;  k += 2)  w.println("" + u[k][2] + "");
       w.println( "");                        // Empty line for gnuplot
      }
    }
    System.out.println("data stored in soliton.dat");
  }
}
```

1. Define a 2D array `u[131][3]` with the first index corresponding to the position x and the second to the time t. With our choice of parameters, the maximum value for x is $130 \times 0.4 = 52$.

2. Initialize the time to $t = 0$ and assign values to `u[i][1]`.

3. Assign values to `u[i][2]`, i=3, 4, ..., 129 corresponding to the next time interval. Use (26.39) to advance the time, but note that you cannot start

at $i = 1$ nor end at $i = 131$ because (26.39) would include `u[132][2]` and `u[-1][1]`, which are beyond the limits of the array.

4. Increment the time and assume that `u[1][2]`=1 and `u[131][2]`=0. To obtain `u[2][2]` and `u[130][2]`, assume that `u[i+2][2]`=`u[i+1][2]` and `u[i-2][2]`=`u[i-1][2]`. Avoid `u[i+2][2]` for `i=130`, and `u[i-2][2]` for `i=2`. To do this, approximate (26.39) so that (26.40) is satisfied.
5. Increment time and compute `u[i][j]` for `j=3` and for `i=3, 4,...,129`, using Eq. (26.38). Again follow the same procedures to obtain the missing array elements `u[2][j]` and `u[130][j]` (set `u[1][j]=1.` and `u[131][j]=0`). As you print out the numbers during the iterations, you will be convinced that it was a good choice.
6. Set `u[i][1] = u[i][2]` and `u[i][2]=u[i][3]` for all `i`. In this way you are ready to find the next `u[i][j]` in terms of the previous two rows.
7. Repeat the previous two steps some 2000 times. Write your solution out to a file after every ∼250 iterations.
8. Use your favorite graphics tool (we used gnuplot) to plot your results as a 3D graph of disturbance u versus position *and* versus time.
9. Observe the wave profile as a function of time and try to confirm Russell's observation that a taller soliton travels faster than a smaller one.

26.8.4 Exploration: Two KdeV Solitons Crossing

Explore what happens when a tall soliton collides with a short one. Do they bounce off each other? Do they go through each other? Do they interfere? Do they destroy each other? Does the tall soliton still move faster than the short one after collision (Fig. 26.5)? Start off by placing a tall soliton of height 0.8 at $x = 12$, and a smaller soliton in front of it at $x = 26$:

$$u(x, t = 0) = 0.8\left[1 - \tanh^2\left(\frac{3x}{12} - 3\right)\right] + 0.3\left[1 - \tanh^2\left(\frac{4.5x}{26} - 4.5\right)\right]$$

26.8.5 Phase-Space Behavior (Exploration)

Construct phase-space plots [$\dot{u}(t)$ versus $u(t)$] of the KdeV equation for various parameter values. Note that only very specific sets of parameters produce solitons. In particular, by correlating the behavior of the solutions with your phase-space plots, show that the soliton solutions correspond to the *separatrix* solutions to the KdeV equation.