3. Modify the `SimpleFFT` class to compute the inverse Fourier transform defined by (9.37). The inverse Fourier transform of a Fourier transformed data set should be the original data set.
4. Compute the CPU time as a function of N for $N = 16$, 64, 256, and 1024 for the `SimpleFFT` algorithm and the direct computation. Use the `currentTimeMillis` method in `System` class to record the time.

```
int n = 10;
long startTime = System.currentTimeMillis();
for(int i=0; i<n; i++) { // average for better results
   fft.transform();
}
long endTime = System.currentTimeMillis();
System.out.println("time/FFT = "+((endTime-startTime)/n));
```

   Verify that the dependence on N is what you expect.
5. Compare the CPU time as a function of N for the `SimpleFFT` class and the `FFT` class in the numerics package.
6. Compare the CPU time for the `FFT` class in the numerics package using two slightly different values of N such that one value is a power of two and the other is not. ■

APPENDIX 9C: PLOTTING SCALAR FIELDS

Imagine a plate that is heated at an interior point and cooled along its edges. In principle, the temperature of this plate can be measured at every point. A scalar quantity, such as temperature, pressure, or light intensity, that is defined throughout a region of space is known as a *scalar field*. The Open Source Physics library contains a number of tools that help us visualize two-dimensional scalar fields. A more complete description of two-dimensional visualization tools is available in *Open Source Physics: A User's Guide with Examples*.

An image in which pixels are color coded can be used to visualize a scalar field. The frames package defines a `RasterFrame` class that makes this process easy and efficient if the scalar field can be represented by integers from 0 to 255. The following program shows how such a `RasterFrame` is used.

Listing 9.14 A scalar field visualization using a raster frame.

```
package org.opensourcephysics.sip.ch09;
import org.opensourcephysics.frames.RasterFrame;

public class RasterFrameApp {
   public static void main(String[] args) {
      RasterFrame frame = new RasterFrame("x", "y", "Raster Frame");
      frame.setPreferredMinMax(-10, 10, -10, 10);
      // generate random data
      int nx = 256, ny = 256;
      int[][] data = new int[nx][ny];
      for(int i = 0;i<nx;i++) {
         for(int j = 0;j<ny;j++) {
            data[i][j] = (int) (255*Math.random());
         }
      }
      frame.setAll(data);
```



```
      frame.setVisible(true);
      frame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
   }
}
```

After the scalar field's values are calculated, the raster's pixels are set using the `setBlock` method. Note that the [0, 0] array element maps to the lower left hand pixel. Because the image raster's mapping has been optimized for speed, the image cannot be resized. The on-screen image size in pixels always matches the array size. Listing 9.11 uses a raster frame to display a Fraunhofer diffraction pattern.

Although the `RasterFrame` makes it easy to work with integer-based data, it lacks the flexibility for more advanced visualizations. It is unsuitable if the image is too small or if the dynamic range of the scalar field is too large. The `Scalar2DFrame` class overcomes these limitations. Using a `Scalar2DFrame` allows us to view the data using different representations, such as contour plots and three-dimensional surface plots. Listing 9.15 shows a `RasterFrame` being used to visualize the function $f(x, y) = xy$.

Listing 9.15 A scalar field test program.

```
package org.opensourcephysics.sip.ch09;
import org.opensourcephysics.frames.Scalar2DFrame;

public class Scalar2DFrameApp {
   final static int SIZE = 16; // array size

   public static void main(String[] args) {
      Scalar2DFrame frame =
           new Scalar2DFrame("x", "y", "Scalar Field");
      double[][] data = new double[16][16];
      frame.setAll(data, -10, 10, -10, 10);
      // generate sample data
      for(int i = 0;i<SIZE;i++) {
         double x = frame.indexToX(i);
         for(int j = 0;j<SIZE;j++) {
            double y = frame.indexToY(j);
            data[i][j] = x*y;
         }
      }
      frame.setAll(data);
      frame.setVisible(true);
      frame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
   }
}
```

Exercise 9.42 Scalar field visualization

Run the scalar field test program and describe the various types of visualizations available under the Tools menu. Which visualizations give respectable representations if the grid is small? What is the maximum grid size that can be used with each type of visualization and still give acceptable performance on your computer? ■