```
      double err = control.getDouble("random y-error");
      int n = control.getInt("n"); // number of intervals
      double[] xData = new double[n];
      double[] yData = new double[n];
      double dx = (n>1) ? (b-a)/(n-1) : 0;
      Function f;
      try {
         f = new ParsedFunction(fstring);
      } catch(ParserException ex) {
         control.println(ex.getMessage());
         return;
      }
      plotFrame.clearData();
      double[] range = Util.getRange(f, a, b, 100);
      plotFrame.setPreferredMinMax(a-(b-a)/4, b+(b-a)/4, range[0],
           range[1]);
      FunctionDrawer func = new FunctionDrawer(f);
      func.color = java.awt.Color.RED;
      plotFrame.addDrawable(func);
      double x = a;
      for(int i = 0;i<n;i++) {
         xData[i] = x;
         yData[i] = f.evaluate(x)*(1+err*(-0.5+Math.random()));
         plotFrame.append(0, xData[i], yData[i]);
         x += dx;
      }
      LagrangeInterpolator interpolator =
           new LagrangeInterpolator(xData, yData);
      plotFrame.addDrawable(new FunctionDrawer(interpolator));
      double[] coef = interpolator.getCoefficients();
      for(int i = 0;i<coef.length;i++) {
         control.println("c["+i+"]="+coef[i]);
      }
   }

   public static void main(String[] args) {
      CalculationControl.createApp(new LagrangeInterpolatorApp());
   }
}
```

Problem 11.26 Lagrange interpolation
Use the `LagrangeInterpolatorApp` class to answer the following questions.

(a) How do the interpolating polynomial's coefficients compare to the series expansion coefficients of the sine and exponential functions?
(b) How well does an interpolating polynomial match a unit step function? You will need to write a step function class that implements the `Function` interface and thus contains an `evaluate` method.
(c) Do your answers depend on the number of sample points?
(d) Add random error to the sample data points for each function. How sensitive is the fit to random errors? ■

Lagrange polynomials should be used cautiously. If the degree of the polynomial is high, the distance between points is large; or if the points are subject to experimental error, the resulting polynomial can oscillate wildly. Press et al. recommend that interpolating polynomials be small. If the data is accurate but the amount of data is large, we often use a polynomial constructed from a small number of nearest neighbors. *Cubic spline* interpolation uses polynomials in this way.

A cubic spline is a third-order polynomial that is required to have a continuous second derivative with neighboring splines at its end points. Because it would be inefficient to store a large number of `Polynomial` objects, the `CubicSpline` class in the numerics package stores the coefficients for the multiple polynomials needed to fit a data set in a single array. The `CubicSplineApp` program tests this class, but it is not shown here because it is similar to `LagrangeInterpolatorApp`.

Exercise 11.27 Cubic splines
Compare the cubic spline interpolating function to the Lagrange polynomial interpolating function using the same samples as were used in Problem 11.26. ■

If the sample data is inaccurate, we often compute the coefficients for a polynomial of lower degree that passes as close as possible to the sample points. This fitting procedure is often used to construct an *ad hoc* function that describes the experimental data. The `PolynomialLeastSquareFit` class in the numerics package implements such a fitting algorithm (see Besset), and the `PolynomialFitApp` program tests this class. It is not shown because it is similar to `LagrangeInterpolatorApp`.

Exercise 11.28 Polynomial fitting
The `PolynomialFitApp` simulates experimental data from a particle trajectory near the Earth. How large a relative error in the y-values can be tolerated if we wish to determine the acceleration of gravity to within 10%? How does this answer change if the number of samples is increased by a factor of two? four? Discuss the effects of changing the degree of the fitting polynomial. ■

Suppose you are given a table of $y_i = f(x_i)$ and are asked to determine the value of x that corresponds to a given y. In other words, how do you find the inverse function $x = f^{-1}(x)$? An interpolation routine that does not require evenly spaced ordinates, such as the `CubicSpline` class, provides an easy and effective solution. The following code uses this technique to define the arcsin function.

Listing 11.7 `Arcsin` demonstrates how to use interpolation to define an inverse function.

```
package org.opensourcephysics.sip.ch11;
import org.opensourcephysics.numerics.*;

public class Arcsin {
   static Function arcsin;

   // probibit instantiation because all methods are static
   private Arcsin() {}

   static public double evaluate(double x) {
      if((x<-1)||(x>1)) {
```