SSJ User's Guide

Package `charts`

Charts generation

Version: November 9, 2009

This package contains utility classes to produce charts used in the Java software developed in the *simulation laboratory* of the DIRO, at the Université de Montréal.

Contents

Overview

This package provides tools for easy construction, visualization, and customization of XY plots, histograms, and empirical styled charts from a Java program. It uses and extends the free and "open source" JFreeChart tool to manage the charts. JFreeChart is distributed under the terms of the GNU Lesser General Public License (LGPL), and can be found at `http://www.jfree.org/jfreechart/`.

This package also provides facilities to export charts to PGF/TikZ source code, which can be included into LaTeX documents. TikZ is a free LaTeX package to produce pictures such as plots, and can be downloaded from `http://sourceforge.net/projects/pgf`.

The user does not need to be familiar with the JFreeChart package or the TikZ syntax to use these tools, except if customization is required. For this, one may see the API specification of JFreeChart, and the reference manual of TikZ.

The two basic abstract classes of package `charts` are `XYChart` and `CategoryChart`. All other charts inherit from one of these two. Charts are managed by the mother class which contains the data tables in a `*SeriesCollection` object, and the information about x-axis and y-axis in `Axis` instances. All these objects encapsulate JFreeChart instances along with some additional TikZ-specific attributes. The method `view` displays charts on screen while the method `toLatex` formats and returns a `String` which contains the TikZ source code that can be written to a LaTeX file.

Several chart styles are available and each one is represented by a subclass of `XYChart` or of `CategoryChart`. The `XYLineChart` uses `XYListSeriesCollection` to plot curves and lines, the `HistogramSeriesCollection` class is used by `HistogramChart` to plot histograms, and the `EmpiricalSeriesCollection` is used by `EmpiricalChart` to plot empirical style charts. It is possible to draw a scatter plot or a box plot using `ScatterChart` or `BoxChart` respectively. These concrete subclasses have similar APIs, but they are specialized for different kinds of charts.

These charts can be customized using `*SeriesCollection` subclasses and `Axis`. First, one can use methods in the `XYChart` class for setting the range values and a background grid. One can also use the method `getSeriesCollection()` for subclasses of charts to obtain the dataset of the chart, which is represented by a `*SeriesCollection` object. This dataset can be customized in the following ways by calling methods on the series collection: selecting color, changing plot style (straight lines, curves, marks only, ...), putting marks on points, setting a label and selecting the dash pattern (solid, dotted, dashed, ...). The available properties depend on the type of chart. Moreover, objects representing the axes can be retrieved with `getXAxis` for the x-axis, and `getYAxis` for the y-axis. By using methods in `Axis`, many customizations are possible: setting a label to the axis, setting ticks labels and values (auto ticks, periodical ticks or manually defined ticks) and changing the twin axis position on the current axis to select where axes must appear. These settings are independent for each axis.

Each chart object from SSJ encapsulates a JFreeChart object: a `XYChart` instance contains a `JFreeChart` instance from JFreeChart API, a `*SeriesCollection` contains a

`XYDataset` and a `XYItemRenderer`, and finally an `Axis` contains a `NumberAxis`. So any parameter proposed by JFreeChart is reachable through getter methods. However, changing the JFreeChart parameters directly may have no impact on the produced TikZ source code.

The two special classes `ContinuousDistChart` and `DiscreteDistIntChart` can be used to plot probability densities, mass functions, and cumulative probabilities for continuous or discrete distributions, which are implemented in package `probdist` of SSJ.

The package `charts` provides additional tools for formatting data plot, and creating charts with multiple datasets. The `PlotFormat` class offers basic tools to import and export data from files. Supported file formats are GNUPlot and standard CSV, which is widely used and accepted by most mathematical softwares such as MATLAB and Mathematica. Customizing file input and output format is also possible. Finally `MultipleDatasetChart` provides tools to plot with different styles on the same chart.

Examples

The following examples demonstrate how to build charts with this package.

Listing 1: A simple example of chart creation

```
import umontreal.iro.lecuyer.charts.XYLineChart;

public class NormalChart
{
   private static double[][] getPoints() {
      // The density of the standard normal probability distribution
      // points contains one array for X values and one array for Y values
      double[][] points = new double[2][25];
      final double CPI = Math.sqrt (2*Math.PI);
      for (int i = 0; i < points[0].length; ++i) {
         double x = -3.5 + i * 7.0 / (points[0].length - 1);
         points[0][i] = x;
         points[1][i] = Math.exp (-x*x/2.0) / CPI;
      }
      return points;
   }

   public static void main(String[] args) {
      double[][] points = getPoints();
      XYLineChart chart = new XYLineChart(null, "X", null, points);

      chart.setAutoRange00(true, true);      // Axes pass through (0,0)
      chart.toLatexFile("NormalChart.tex", 12, 8);
   }
}
```

The first program, displayed in Listing 1, shows the simplest way to export charts from data tables. First, a curve is defined with regularly spaced x-coordinates in method

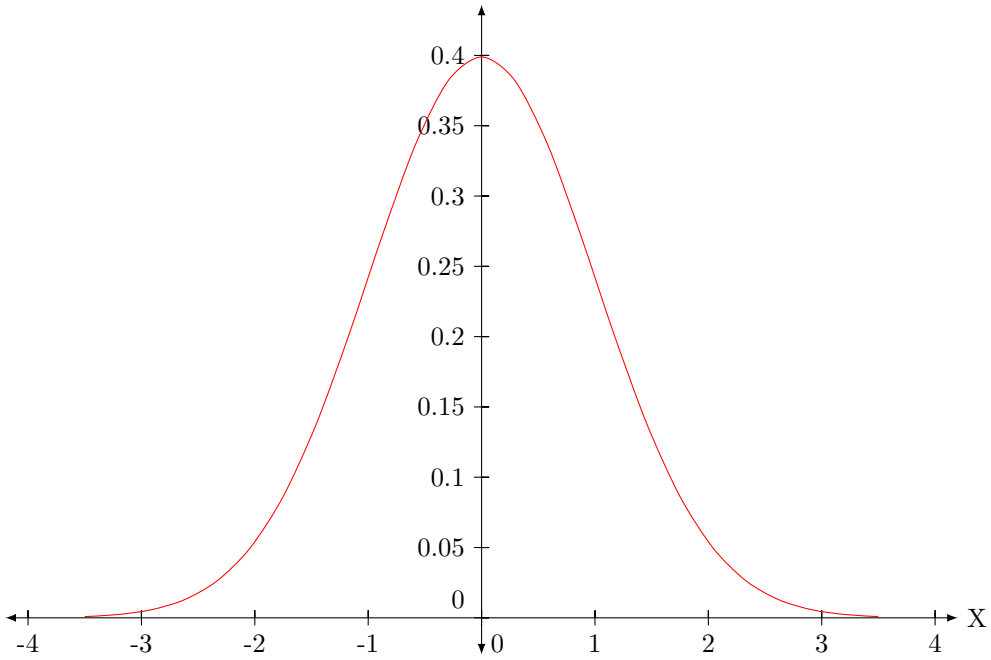

Figure 1: The density of the standard normal.

`getPoints`; it represents the curve $y = e^{-x^2/2}/\sqrt{2\pi}$, the density of the standard normal probability distribution. Arrays `points[0]` contain the x-coordinates and `points[1]` the y-coordinates of the points. Figure 1 presents the resulting chart if the produced TikZ code is added to a LaTeX document using the `tikz` package, and compiled using LaTeXor PdfLaTeX.

A simpler way to plot a probability density or a distribution function is to use the class `ContinuousDistChart`, as shown in Listing 2 where however, the normal density will be plotted directly on the screen.

Listing 2: The normal density

```
import umontreal.iro.lecuyer.probdist.*;
import umontreal.iro.lecuyer.charts.*;

public class ContDistPlot
{
   public static void main (String[] args) {
      ContinuousDistribution dist = new NormalDist();
      ContinuousDistChart plot = new ContinuousDistChart(dist, -3.5, 3.5, 1000);
      plot.viewDensity(600, 400);
   }
}
```

The next example, displayed in Listing 3, plots the probability mass function for a Poisson distribution with $\lambda = 50$ by simply creating an instance of `DiscreteDistIntChart`. Figure 2

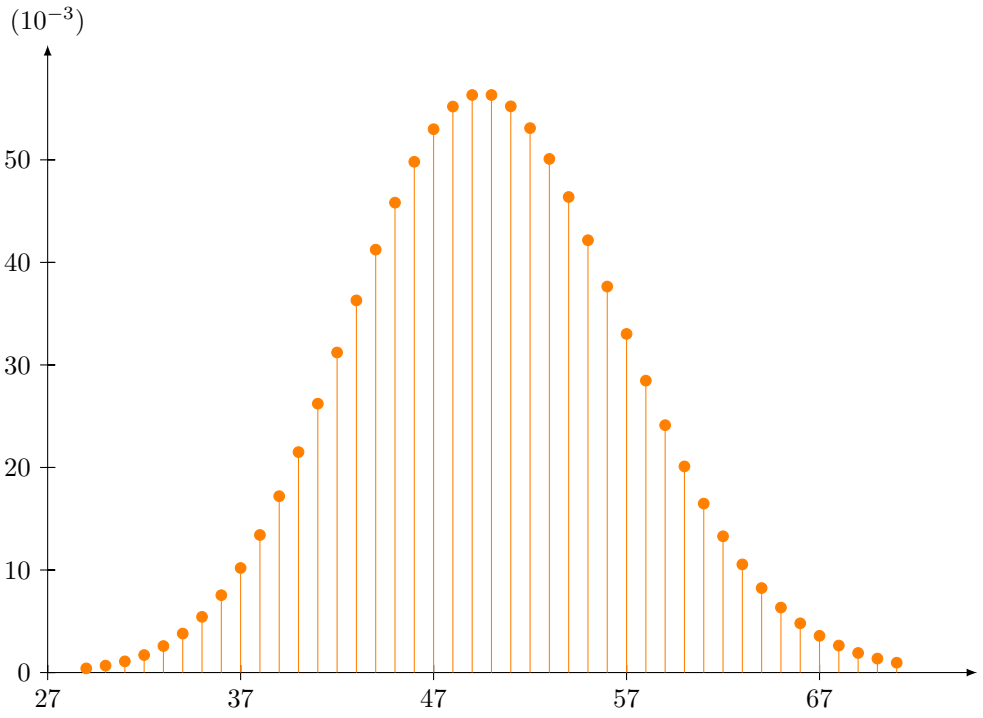

Figure 2: The probabilities of the Poisson distribution with $\lambda = 50$.

presents the resulting chart obtained from PdfLaTeX.

Listing 3: Probabilities of the Poisson distribution

```
import umontreal.iro.lecuyer.probdist.*;
import umontreal.iro.lecuyer.charts.*;
import java.io.*;

public class DistIntTest
{
   public static void main(String[] args) throws IOException {
      PoissonDist dist = new PoissonDist(50);
      DiscreteDistIntChart dic = new DiscreteDistIntChart(dist);

      // Export to Latex format
      String output = dic.toLatexProb(12, 8);  // 12cm width, 8cm height
      Writer file = new FileWriter("DistIntTest.tex");
      file.write(output);
      file.close();
   }
}
```

The next program, displayed in Listing 4, shows how to export several charts from data tables. First, three curves are defined with regularly spaced x-coordinates in methods `getPoints*`; they represent the curves $y = \sqrt{x}$, $y = \cos(x)$ and $y = x + 2$, respectively. Array `points[0]` contains the x-coordinates and `points[1]` the y-coordinates of the points. Figure 3 presents the resulting chart if the produced TikZ code is added to a LaTeX document using the `tikz` package, and compiled using LaTeXor PdfLaTeX.

Listing 4: Three curves on the same chart

```
import umontreal.iro.lecuyer.charts.XYLineChart;

public class ChartTest1
{
   private static double[][] getPoints1() {
      double[][] points = new double[2][200];
      for (int i = 0; i < points[0].length; i++) {
         double x = i / 25.0;
         points[0][i] = x;
         points[1][i] = Math.sqrt (x);
      }
      return points;
   }

   private static double[][] getPoints2() {
      double[][] points = new double[2][21];
      for (int i = 0; i < points[0].length; i++) {
         double x = -Math.PI + 2 * i * Math.PI / (points[0].length - 1);
         points[0][i] = x;
         points[1][i] = Math.cos (x);
      }
      return points;
   }

   private static double[][] getPoints3() {
      double[][] points = new double[2][11];
      for (int i = 0; i < points[0].length; i++) {
         points[0][i] = -5 + i;
         points[1][i] = -3 + i;
      }
      return points;
   }

   public static void main(String[] args) {
      // Get data; data1 has length 2 and contains one array for
      // X-axis values, and one array for Y-axis values.
      double[][] data1 = getPoints1();
      double[][] data2 = getPoints2();
      double[][] data3 = getPoints3();

      // Create a new chart with the previous data series.
```

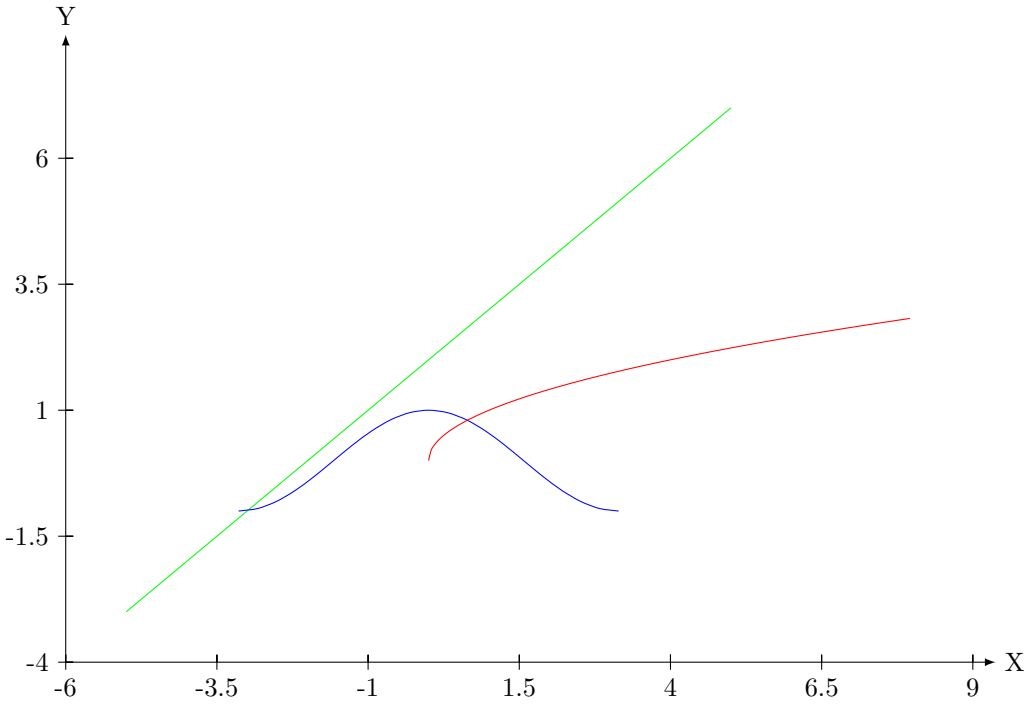

Figure 3: Results for three curves on the same chart.

```
        XYLineChart chart = new XYLineChart(null, "X", "Y", data1, data2, data3);
        chart.toLatexFile("ChartTest1.tex", 12, 8);
    }
}
```

The next example, given in Listing 5, shows how to customize a chart. First, three curves are defined as in example 4 above and the chart is created. Then the axes are customized by adding labels at chosen values on the x-axis. On the y-axis, successive labels are set regularly at points 1 unit apart. Then the data plot itself is customized. A new color is created in the RGB model for the first curve which also receives a label name and a dash plot style. Similarly, the other two curves receive their label name, plot style and color. Note that the third curve is drawn in the ORANGE color predefined in the AWT package of the standard Java toolkit. Finally, the charts are exported to a file in LaTeX format. If the file is compiled with LaTeX or PdfLaTeX, the resulting chart will appear as displayed in Figure 4.

Listing 5: Code for creating and customizing a chart

```
import umontreal.iro.lecuyer.charts.*;
import java.awt.Color;

public class ChartTest2
{
```

```
   private static double[][] getPoints1() {
      double[][] points = new double[2][40];
      for (int i = 0; i < points[0].length; i++) {
         double x = i / 4.0;
         points[0][i] = x;
         points[1][i] = Math.sqrt(x);
      }
      return points;
   }

   private static double[][] getPoints2() {
      double[][] points = new double[2][21];
      for (int i = 0; i < points[0].length; i++) {
         double x = -Math.PI + 2 * i * Math.PI / (points[0].length - 1);
         points[0][i] = x;
         points[1][i] = Math.cos(x);
      }
      return points;
   }

   private static double[][] getPoints3() {
      double[][] points = new double[2][11];
      for (int i = 0; i < points[0].length; i++) {
         points[0][i] = -5 + i;
         points[1][i] = -3 + i;
      }
      return points;
   }

   public static void main(String[] args) {
      double[][] data1 = getPoints1();
      double[][] data2 = getPoints2();
      double[][] data3 = getPoints3();

      // Create a new chart with the previous data series.
      XYLineChart chart = new XYLineChart(null, "X", "Y", data1, data2, data3);

      // Customizing axes
      Axis xaxis = chart.getXAxis();
      Axis yaxis = chart.getYAxis();
      String[] labels = { "-9", "$-\\lambda$", "$-\\sqrt{2}$",
                          "0", "$\\frac{14}{\\pi}$", "\\LaTeX" };
      double[] values = { -9, -5, -Math.sqrt(2), 0, 14.0 / Math.PI, 9 };
      xaxis.setLabels(values, labels);
      xaxis.enableCustomLabels();
      yaxis.setLabels(1);

      // Data plots customizing
      XYListSeriesCollection collec = chart.getSeriesCollection();
      collec.setColor(0, new Color(0, 64, 128));
      collec.setName(0, "$f(x) = \\sqrt(x)$");
```

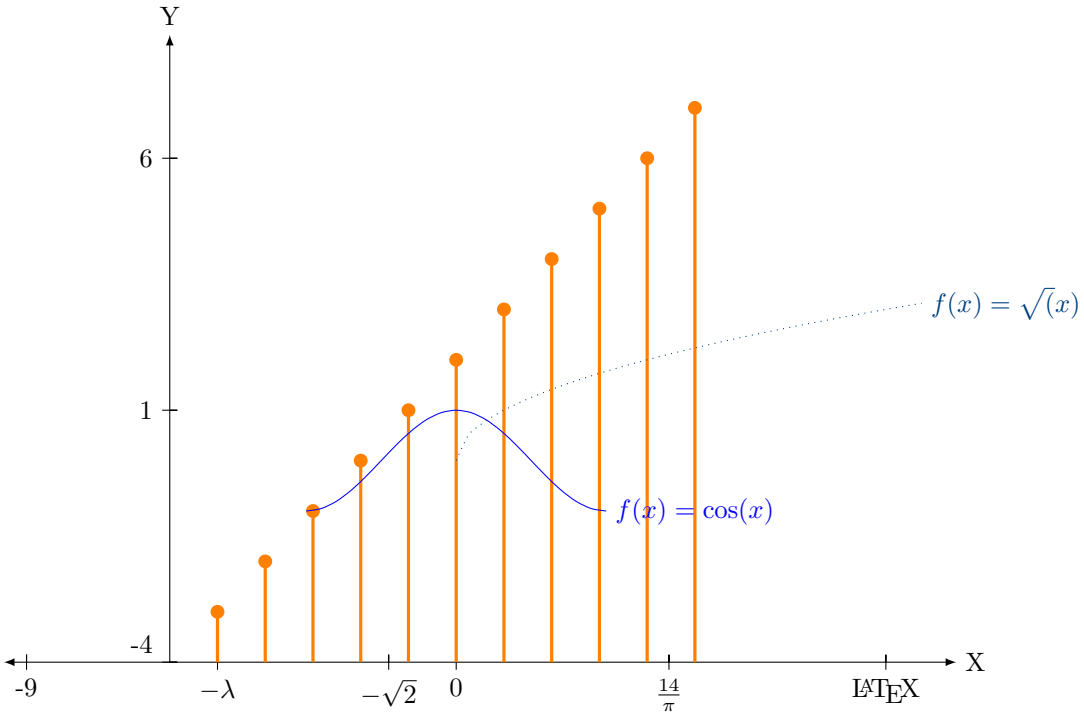

Figure 4: A customized chart.

```
        collec.setMarksType(0, "");
        collec.setDashPattern(0, "dotted");
        collec.setName(1, "$f(x) = \\cos(x)$");
        collec.setMarksType(1, "");
        collec.setColor(2, Color.ORANGE);
        collec.setPlotStyle(2, "ycomb,very thick");
        collec.setMarksType(2, "*");

        // Export to LaTex format
        chart.toLatexFile("ChartTest2.tex", 12, 8);  // 12cm width, 8cm height
    }
}
```

The next example, given in Listing 6, shows how to plot and customize empirical distributions. First, two empirical distributions are defined from a sample of points obtained from a uniform generator and a Beta$(3, 1)$ generator, both on the interval $[0, 1]$. Then an empirical chart is created to plot these two distributions. The first distribution is plotted in MAGENTA color with filled square marks from TikZ. The second distribution uses default color and plot marks. A background grid is also added with cells of size 0.1×0.1. Finally, the charts are exported to a file in LaTeX format. Figure 5 shows the resulting chart.

Listing 6: Creating and customizing empirical distribution charts

```
import umontreal.iro.lecuyer.rng.*;
import umontreal.iro.lecuyer.randvar.*;
import umontreal.iro.lecuyer.charts.*;
import java.util.Arrays;
import java.awt.Color;

public class EmpiricalChartTest
{
   private static double[] getPoints1() {
      RandomVariateGen gen = new UniformGen(new LFSR113());
      final int N = 10;
      double[] data = new double[N];
      for (int i = 0; i < N; i++)
         data[i] = gen.nextDouble();
      Arrays.sort(data);
      return data;
   }

   private static double[] getPoints2() {
      RandomVariateGen gen = new BetaGen(new LFSR113(), 3, 1);
      final int N = 20;
      double[] data = new double[N];
      for (int i = 0; i < N; i++)
         data[i] = gen.nextDouble();
      Arrays.sort(data);
      return data;
   }

   public static void main(String[] args) {
      double[] data1 = getPoints1();
      double[] data2 = getPoints2();

      // Create a new chart with the previous data series.
      EmpiricalChart chart = new EmpiricalChart(null, null, null, data1, data2);

      // Data plots customizing
      EmpiricalSeriesCollection collec = chart.getSeriesCollection();
      collec.setMarksType(0, "square*");
      collec.setColor(0, Color.MAGENTA);

      chart.enableGrid(0.1, 0.1);            // Enables grid
      chart.toLatexFile("EmpiricalChartTest.tex", 12, 8);
   }
}
```

The next example, given in Listing 7, shows how to plot a simple histogram. First, data is generated from a standard normal. Then the plot is customized: 80 bins are selected, and the range of the plot is set manually with `bounds`; the interval is $[-4, 4]$ for the x-coordinates,

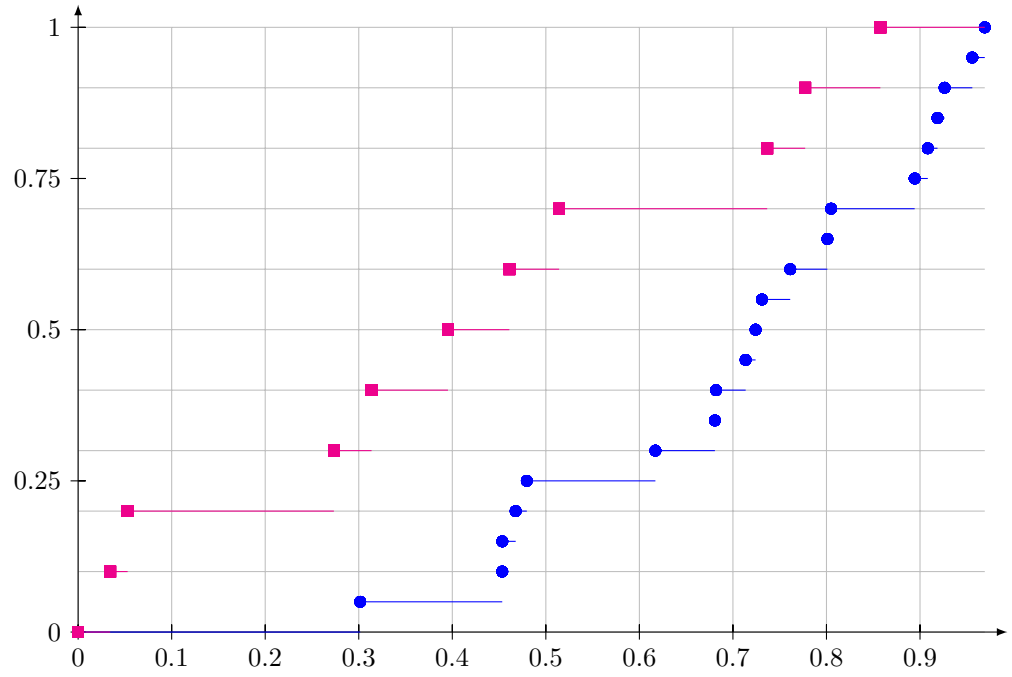

Figure 5: A customized empirical distribution chart

and $[0, 5000]$ for the y-coordinates. Finally, the histogram chart is viewed on the screen, then exported to file `HistogramTest1.tex` in LaTeX format. Figure 6 displays the resulting chart.

Listing 7: Source code creating a simple histogram

```
import umontreal.iro.lecuyer.charts.*;
import umontreal.iro.lecuyer.rng.*;
import umontreal.iro.lecuyer.randvar.*;
import java.awt.Color;

public class HistogramTest1
{
   private static double[] getData() {
      NormalGen gen = new NormalGen(new LFSR113());
      final int N = 100000;
      double[] ad = new double[N];
      for (int i = 0; i < N; i++)
         ad[i] = gen.nextDouble();
      return ad;
   }

   public static void main(String[] args) {
      double[] data = getData();

      HistogramChart chart;
      chart = new HistogramChart("Standard Normal", null, null, data);
```

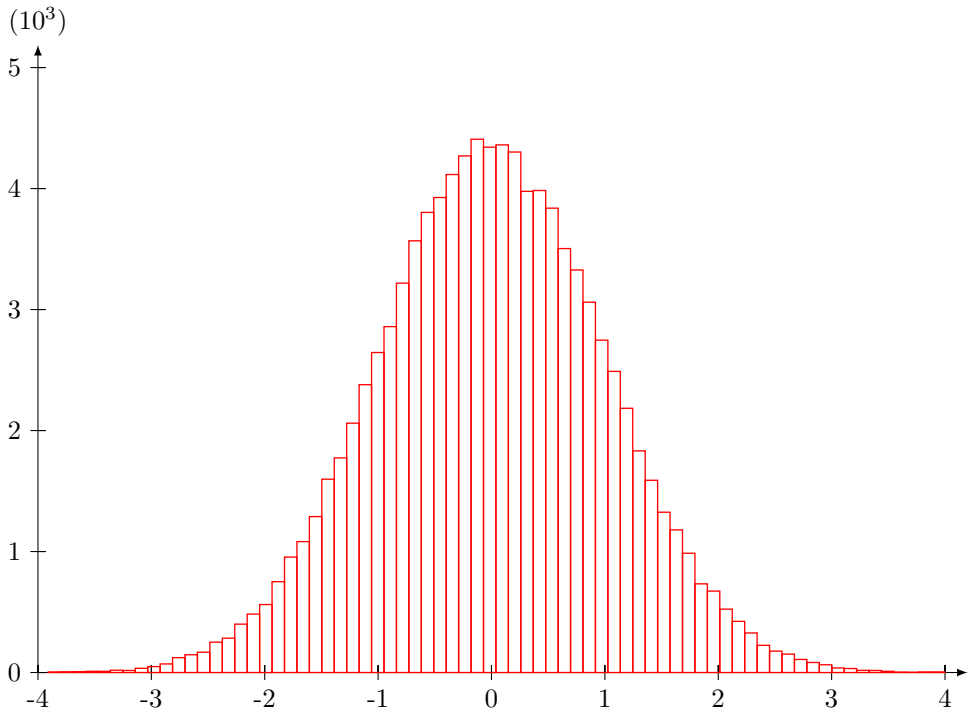

Figure 6: A simple histogram for the standard normal density

```
        // Customizes the data plot
        HistogramSeriesCollection collec = chart.getSeriesCollection();
        collec.setBins(0, 80);
        double[] bounds = { -4, 4, 0, 5000 };
        chart.setManualRange(bounds);

        chart.view(800, 500);
        chart.toLatexFile("HistogramTest1.tex", 12, 8);
    }
}
```

The next example, given in Listing 8, shows how to plot and customize histograms. First, the two histogram charts `data1` and `data2` are created. Then the data plots are customized: two colors are selected in the sRGB model from the `java.awt.Color` package. 40 bins are selected on the interval $[-6, 6]$ for the first histogram. The number of bins for the second histogram is set automatically. The range of the plot is set manually with `bounds`. Finally, the two histograms charts are exported to file `HistogramChartTest.tex` in LaTeX format. Figure 7 displays the resulting chart.

Listing 8: Source code creating and customizing histograms.

```
import umontreal.iro.lecuyer.charts.*;
import umontreal.iro.lecuyer.rng.*;
import umontreal.iro.lecuyer.randvar.*;
import java.awt.Color;

public class HistogramChartTest
{
   private static double[] getPoints1() {
      NormalGen gen = new NormalGen(new MRG32k3a(), 0, 2);
      final int N = 100000;
      double[] ad = new double[N];
      for (int i = 0; i < N; i++)
         ad[i] = gen.nextDouble();
      return ad;
   }

   private static double[] getPoints2() {
      ExponentialGen gen = new ExponentialGen(new MRG32k3a(), 1);
      final int N = 100000;
      double[] ad = new double[N];
      for (int i = 0; i < N; i++)
         ad[i] = gen.nextDouble();
      return ad;
   }

   public static void main(String[] args) {
      double[] data1 = getPoints1();
      double[] data2 = getPoints2();

      // Create a new chart with the previous data series.
      HistogramChart chart = new HistogramChart(null, null, null, data1, data2);

      // Customizes the data plots
      HistogramSeriesCollection collec = chart.getSeriesCollection();
      collec.setColor(0, new Color(255, 0, 0, 128));
      collec.setColor(1, new Color(0, 255, 0, 128));
      collec.setBins(0, 40, -6, 6);

      // Define range bounds.
      double[] bounds = { -6, 6, 0, 30000 };
      chart.setManualRange00(bounds, true, true);

      chart.toLatexFile("HistogramChartTest.tex", 12, 8);
   }
}
```

The next example, given in Listing 9, shows how to create a box-and-whisker plot. First, two series of 1000 points are obtained from a lognormal distribution and from a Poisson distribution with $\lambda = 5$. These are passed to a `BoxChart` object, which creates the boxplot,

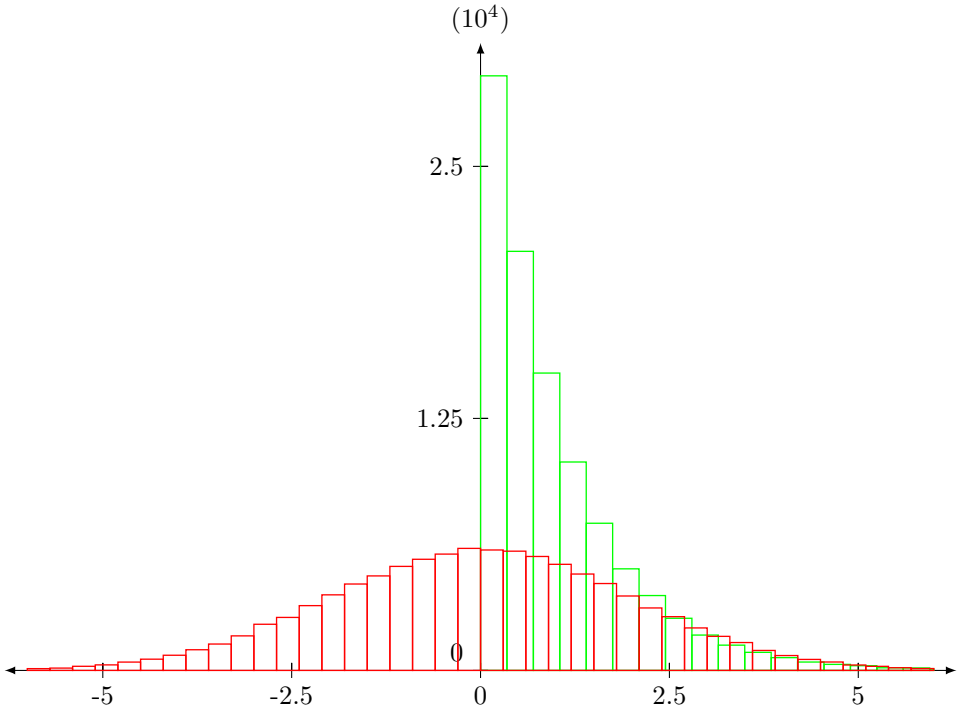

Figure 7: A customized histogram chart

which can be viewed on screen by executing the program. We find that the boxplot for the Poisson data (the box on the right of the chart) has median = 5 (the line inside the box), a mean = 5.009 (the center of the black circle) the first and the third quartiles at 3 and 6, while the lower and upper whiskers are at 0 and 10. Finally, there are outliers at 11 and 12 (the hollow circles) and extreme outliers (the triangle) at 13, outside the chart. We see that the Poisson data is skewed towards higher values. A similar description applies to the lognormal data (the box on the left of the chart), which is strongly skewed towards smaller values.

Listing 9: Source code creating a box-and-whisker plot.

```
import umontreal.iro.lecuyer.charts.*;
import umontreal.iro.lecuyer.randvar.*;
import umontreal.iro.lecuyer.rng.*;
import java.io.*;

public class BoxTest
{
   public static void main (String[] args) throws IOException {
      int count = 1000;
      double[] data1 = new double[count];
      double[] data2 = new double[count];

      RandomStream stream = new LFSR113();
```

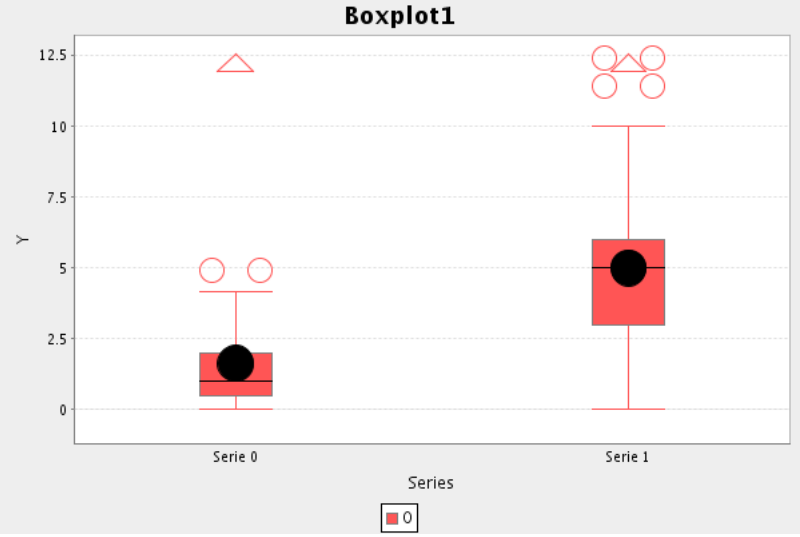

Figure 8: A box-plot

```
        RandomVariateGen log = new LognormalGen(stream);
        RandomVariateGen poi = new PoissonGen(stream, 5.0);

        for (int i = 0; i < count; i++) {
            data1[i] = log.nextDouble();
            data2[i] = poi.nextDouble();
        }

        BoxChart bc = new BoxChart("Boxplot1", "Series", "Y", data1, data2);
        bc.view(600, 400);
    }
}
```

XYChart

This class provides tools to create charts from data in a simple way. Its main feature is to produce TikZ/PGF (see WWW link `http://sourceforge.net/projects/pgf/`) compatible source code which can be included in LaTeX documents, but it can also produce charts in other formats. One can easily create a new chart, and customize its appearance using methods of this class, with the encapsulated `SSJXYSeriesCollection` object representing the data, and the two `Axis` objects representing the axes. All these classes depend on the `JFreeChart` API (see WWW link `http://www.jfree.org/jfreechart/`) which provides tools to build charts with Java, to draw them, and export them to files. However, only basic features are used here.

Moreover, `XYChart` provides methods to plot data using a MATLAB friendly syntax. None of these methods provides new features; they just propose a different syntax to create charts. Therefore some features are unavailable when using these methods only.

```
package umontreal.iro.lecuyer.charts;

public abstract class XYChart
```

Methods

`public JFreeChart getJFreeChart()`

Returns the `JFreeChart` object associated with this chart.

`public Axis getXAxis()`

Returns the chart's domain axis (x-axis) object.

`public Axis getYAxis()`

Returns the chart's range axis (y-axis) object.

`public abstract JFrame view (int width, int height);`

`public String getTitle()`

Gets the current chart title.

`public void setTitle (String title)`

Sets a title to this chart. This title will appear on the chart displayed by method `view`.

`public void setprobFlag (boolean flag)`

Must be set `true` when plotting probabilities, `false` otherwise.

`public void setAutoRange()`

The x and the y ranges of the chart are set automatically.

`public void setAutoRange (boolean right, boolean top)`

The x and the y ranges of the chart are set automatically. If `right` is `true`, the vertical axis will be on the left of the points, otherwise on the right. If `top` is `true`, the horizontal axis will be under the points, otherwise above the points.

`public void setAutoRange00 (boolean xZero, boolean yZero)`

The x and the y ranges of the chart are set automatically. If `xZero` is `true`, the vertical axis will pass through the point $(0, y)$. If `yZero` is `true`, the horizontal axis will pass through the point $(x, 0)$.

`public void setManualRange (double[] range)`

Sets the x and y ranges of the chart using the format: `range = [xmin, xmax, ymin, ymax]`.

`public void setManualRange (double[] range, boolean right, boolean top)`

Sets the x and y ranges of the chart using the format: `range = [xmin, xmax, ymin, ymax]`. If `right` is `true`, the vertical axis will be on the left of the points, otherwise on the right. If `top` is `true`, the horizontal axis will be under the points, otherwise above the points.

`public void setManualRange00 (double[] range, boolean xZero, boolean yZero)`

Sets the x and y ranges of the chart using the format: `range = [xmin, xmax, ymin, ymax]`. If `xZero` is `true`, the vertical axis will pass through the point $(0, y)$. If `yZero` is `true`, the horizontal axis will pass through the point $(x, 0)$.

`public double getChartMargin()`

Returns the chart margin, which is the fraction by which the chart is enlarged on its borders. The default value is 0.02.

`public void setChartMargin (double margin)`

Sets the chart margin to `margin`. It is the fraction by which the chart is enlarged on its borders. Restriction: `margin` ≥ 0.

`public abstract void setTicksSynchro (int s);`

Synchronizes x-axis ticks to the s-th series x-values.

Latex-specific methods

`public void enableGrid (double xstep, double ystep)`

Puts a grid on the background. It is important to note that the grid is always shifted in such a way that it contains the axes. Thus, the grid does not always have an intersection at the corner points; this occurs only if the corner points are multiples of the steps: `xstep` and `ystep` sets the step in each direction.

`public void disableGrid()`

Disables the background grid.

`public abstract String toLatex (double width, double height);`

Exports the chart to a LaTeX source code using PGF/TikZ. This method constructs and returns a string that can be written to a LaTeX document to render the plot. `width` and `height` represents the width and the height of the produced chart. These dimensions do not take into account the axes and labels extra space. The `width` and the `height` of the chart are measured in centimeters.

`public void toLatexFile (String fileName, double width, double height)`

Transforms the chart to LaTeX form and writes it in file `fileName`. The chart's width and height (in centimeters) are `width` and `height`.

`public void setLatexDocFlag (boolean flag)`

Flag to remove the `\documentclass` (and other) commands in the created LaTeX files. If `flag` is `true`, then when charts are translated into LaTeX form, it will be as a self-contained file that can be directly compiled with LaTeX. However, in this form, the file cannot be included in another LaTeX file without causing compilation errors because of the multiple instructions `\documentclass` and `\begin{document}`. By setting `flag` to `false`, these instructions will be removed from the LaTeX chart files, which can then be included in a master LaTeX file. By default, the flag is `true`.

XYLineChart

This class provides tools to create and manage curve plots. Using the `XYLineChart` class is the simplest way to produce curve plots only. Each `XYLineChart` object is linked with a `XYListSeriesCollection` data set.

```
package umontreal.iro.lecuyer.charts;

public class XYLineChart extends XYChart
```

Constructors

```
public XYLineChart()
```

Initializes a new `XYLineChart` instance with an empty data set.

```
public XYLineChart (String title, String XLabel, String YLabel,
                    double[][]... data)
```

Initializes a new `XYLineChart` instance with sets of points `data`. `title` is a title, `XLabel` is a short description of the x-axis, and `YLabel` a short description of the y-axis. The input parameter `data` represents a set of plotting data.

For example, if one n-row matrix `data1` is given as argument `data`, then the first row `data1`$[0]$ represents the x-coordinate vector, and every other row `data1`$[i], i = 1, \ldots, n-1$, represents a y-coordinate set for a curve. Therefore matrix `data1`$[i][j]$, $i = 0, \ldots, n-1$, corresponds to $n-1$ curves, all with the same x-coordinates.

However, one may want to plot several curves with different x-coordinates. In that case, one should give the curves as matrices with two rows. For examples, if the argument `data` is made of three 2-row matrices `data1`, `data2` and `data3`, then they represents three different curves, `data*`$[0]$ being the x-coordinates, and `data*`$[1]$ the y-coordinates of the curves.

```
public XYLineChart (String title, String XLabel, String YLabel,
                    double[][] data, int numPoints)
```

Initializes a new `XYLineChart` instance with sets of points `data`. `title` is a title, `XLabel` is a short description of the x-axis, and `YLabel` a short description of the y-axis. If `data` is a n-row matrix, then the first row `data`$[0]$ represents the x-coordinate vector, and every other row `data`$[i], i = 1, \ldots, n-1$, represents a y-coordinate set of points. Therefore matrix `data`$[i][\ ]$, $i = 0, \ldots, n-1$, corresponds to $n-1$ curves, all with the same x-coordinates. However, only *the first* `numPoints` of `data` will be considered to plot each curve.

```
public XYLineChart (String title, String XLabel, String YLabel,
                    double[][] data, int x, int y)
```

Initializes a new `XYLineChart` instance using subsets of `data`. `data[x][.]` will form the x-coordinates and `data[y][.]` will form the y-coordinates of the chart. `title` sets a title, `XLabel` is a short description of the x-axis, and `YLabel` is a short description of the y-axis. Warning: if the new x-axis coordinates are not monotone increasing, then they will

automatically be sorted in increasing order so the points will be reordered, but the original `data` is not changed.

```
public XYLineChart (String title, String XLabel, String YLabel,
                    DoubleArrayList... data)
```

Initializes a new `XYLineChart` instance with data `data`. The input parameter `data` represents a set of plotting data. A `DoubleArrayList` from the Colt library is used to store the data. The description is similar to the constructor `YListChart` with `double[]... data`.

```
public XYLineChart (String title, String XLabel, String YLabel,
                    XYSeriesCollection data)
```

Initializes a new `XYLineChart` instance with data `data`. The input parameter `data` represents a set of plotting data. `XYSeriesCollection` is a `JFreeChart` container class to store XY plots.

Methods

```
public int add (double[] x, double[] y, String name, String plotStyle)
```

Adds a data series into the series collection. Vector `x` represents the x-coordinates and vector `y` represents the y-coordinates of the series. `name` and `plotStyle` are the name and the plot style associated to the series.

```
public int add (double[] x, double[] y)
```

Adds a data series into the series collection. Vector `x` represents the x-coordinates and vector `y` represents the y-coordinates of the series.

```
public int add (double[] x, double[] y, int numPoints)
```

Adds a data series into the series collection. Vector `x` represents the x-coordinates and vector `y` represents the y-coordinates of the series. Only *the first* `numPoints` of `x` and `y` will be taken into account for the new series.

```
public int add (double[][] data)
```

Adds the new collection of data series `data` into the series collection. If `data` is a n-row matrix, then the first row `data`[0] represents the x-coordinate vector, and every other row `data`$[i], i = 1, \ldots, n-1$, represents a y-coordinate set of points. Therefore matrix `data`$[i][\,]$, $i = 0, \ldots, n-1$, corresponds to $n-1$ curves, all with the same x-coordinates.

```
public int add (double[][] data, int numPoints)
```

Adds the new collection of data series `data` into the series collection. If `data` is a n-row matrix, then the first row `data`[0] represents the x-coordinate vector, and every other row `data`$[i], i = 1, \ldots, n-1$, represents a y-coordinate set of points. Therefore matrix `data`$[i][\,]$, $i = 0, \ldots, n-1$, corresponds to $n-1$ curves, all with the same x-coordinates. However, only *the first* `numPoints` of `data` will be taken into account for the new series.

```
public XYListSeriesCollection getSeriesCollection()
```

Returns the chart's dataset.

`public void setSeriesCollection (XYListSeriesCollection dataset)`

Links a new dataset to the current chart.

`public void setTicksSynchro (int s)`

Synchronizes X-axis ticks to the s-th series x-values.

`public JFrame view (int width, int height)`

Displays chart on the screen using Swing. This method creates an application containing a chart panel displaying the chart. The created frame is positioned on-screen, and displayed before it is returned. The `width` and the `height` of the chart are measured in pixels.

`public JFrame viewBar (int width, int height)`

Displays bar chart on the screen using Swing. This method creates an application containing a bar chart panel displaying the chart. The created frame is positioned on-screen, and displayed before it is returned. The `width` and the `height` of the chart are measured in pixels.

Latex-specific method

`public String toLatex (double width, double height)`

YListChart

This class extends the class `XYLineChart`. Each `YListChart` object is associated with a `YListSeriesCollection` data set. The data is given as one or more lists of y-coordinates. The x-coordinates are given by the indices + 1 of each y-coordinate in the given lists.

```
package umontreal.iro.lecuyer.charts;

public class YListChart extends XYLineChart
```

Constructor

```
public YListChart (String title, String XLabel, String YLabel,
                   double[]... data)
```

Initializes a new `YListChart` instance with set of points `data`. `title` is a title, `XLabel` is a short description of the x-axis, and `YLabel` a short description of the y-axis. The input vectors represents a set of plotting data. More specifically, each vector `data` represents a y-coordinates set. Position in the vector will form the x-coordinates. Indeed, the value `data`$[j]$ corresponds to the point $(j + 1, \texttt{data}[j])$ (but rescaled) on the chart.

```
public YListChart (String title, String XLabel, String YLabel,
                   double[] data, int numPoints)
```

Initializes a new `YListChart` instance with a set of points `data`. `title` is a title, `XLabel` is a short description of the x-axis, and `YLabel` a short description of the y-axis. The input vector represents a set of plotting data. Position in the vector gives the x-coordinates of the curve. The value `data`$[j]$ corresponds to the point $(j + 1,$ `data`$[j])$ (but rescaled on the chart) for the curve. However, only *the first* `numPoints` of `data` will be considered to plot the curve.

```
public YListChart (String title, String XLabel, String YLabel,
                   double[][] data, int numPoints)
```

Initializes a new `YListChart` instance with set of points `data`. `title` is a title, `XLabel` is a short description of the x-axis, and `YLabel` a short description of the y-axis. The input vectors represents a set of plotting data. More specifically, for a n-row matrix `data`, each row `data`$[i], i = 0, \ldots, n-1$, represents a y-coordinate set for a curve. Position in the vector gives the x-coordinates of the curves. Indeed, the value `data`$[i][j]$ corresponds to the point $(j+1,$ `data`$[i][j])$ (but rescaled on the chart) for curve i. However, only *the first* `numPoints` of each `data`$[i]$ will be considered to plot each curve.

EmpiricalChart

This class provides additional tools to create and manage empirical plots. Empirical plots are used to plot empirical distributions. The `EmpiricalChart` class is the simplest way to produce empirical plots only. Each `EmpiricalChart` object is linked with an `EmpiricalSeriesCollection` data set.

```
package umontreal.iro.lecuyer.charts;

public class EmpiricalChart extends XYChart
```

Constructors

`public EmpiricalChart()`
Initializes a new `EmpiricalChart` instance with an empty data set.

`public EmpiricalChart (String title, String XLabel, String YLabel, double[]... data)`
Initializes a new `EmpiricalChart` instance with data `data`. `title` is a title, `XLabel` is a short description of the x-axis and `YLabel` a short description of the y-axis. The input vectors `data` represents a collection of observation sets. Each vector of `data` represents a x-coordinates set. Therefore `data`$[i], i = 0, \ldots, n-1$, is used to draw the i-th plot. The values of each observation set `data`$[i]$ *must be sorted* in increasing order.

`public EmpiricalChart (String title, String XLabel, String YLabel, double[] data, int numPoints)`
Initializes a new `EmpiricalChart` instance with a set of points `data`. `title` is a title, `XLabel` is a short description of the x-axis and `YLabel` a short description of the y-axis. Vector `data` represents a x-coordinates set. The values of this observation set *must be sorted* in increasing order. Only *the first* `numPoints` of `data` will be considered to plot.

`public EmpiricalChart (String title, String XLabel, String YLabel, DoubleArrayList... data)`
Similar to the above constructor, but with `DoubleArrayList`. A `DoubleArrayList` from the Colt library is used to store data. The values of each observation set `data`$[i]$ *must be sorted* in increasing order.

`public EmpiricalChart (String title, String XLabel, String YLabel, TallyStore... tallies)`
Initializes a new `EmpiricalChart` instance with data arrays contained in each `TallyStore` object. The input parameter `tallies` represents a collection of observation sets. Therefore, the i-th `tallies` is used to draw the i-th plot.

`public EmpiricalChart (String title, String XLabel, String YLabel, XYSeriesCollection data)`
Initializes a new `EmpiricalChart` instance with data `data`. The input parameter `data` represents a set of plotting data. `XYSeriesCollection` is a `JFreeChart`-like container class used to store and manage observation sets.

Methods

`public EmpiricalSeriesCollection getSeriesCollection()`

Returns the chart's dataset.

`public void setSeriesCollection (EmpiricalSeriesCollection dataset)`

Links a new dataset to the current chart.

`public void setTicksSynchro (int s)`

Synchronizes x-axis ticks to the s-th series x-values.

`public JFrame view (int width, int height)`

Displays chart on the screen using Swing. This method creates an application containing a chart panel displaying the chart. The created frame is positioned on-screen, and displayed before it is returned. The `width` and the `height` of the chart are measured in pixels.

LaTeX-specific method

`public String toLatex (double width, double height)`

HistogramChart

This class provides tools to create and manage histograms. The `HistogramChart` class is the simplest way to produce histograms. Each `HistogramChart` object is linked with an `HistogramSeriesCollection` dataset.

```
package umontreal.iro.lecuyer.charts;

public class HistogramChart extends XYChart
```

Constructors

```
public HistogramChart ()
```

Initializes a new `HistogramChart` instance with an empty data set.

```
public HistogramChart (String title, String XLabel, String YLabel,
                       double[]... data)
```

Initializes a new `HistogramChart` instance with input `data`. `title` is a title, `XLabel` is a short description of the x-axis, and `YLabel` a short description of the y-axis. The input parameter `data` represents a collection of observation sets. Therefore `data`$[i], i = 0, \ldots, n-1$, is used to plot the ith histogram.

```
public HistogramChart (String title, String XLabel, String YLabel,
                       double[] data, int numPoints)
```

Initializes a new `HistogramChart` instance with input `data`. `title` is a title, `XLabel` is a short description of the x-axis, and `YLabel` a short description of the y-axis. The input parameter `data` represents an observation set. Only *the first* `numPoints` of `data` will be considered to plot the histogram.

```
public HistogramChart (String title, String XLabel, String YLabel,
                       DoubleArrayList... data)
```

Initializes a new `HistogramChart` instance with data `data`. Each `DoubleArrayList` input parameter represents a collection of observation sets. `DoubleArrayList` is from the Colt library and is used to store data.

```
public HistogramChart (String title, String XLabel, String YLabel,
                       TallyStore... tallies)
```

Initializes a new `HistogramChart` instance with data arrays contained in each `TallyStore` object. The input parameter `tallies` represents a collection of observation sets.

```
public HistogramChart (String title, String XLabel, String YLabel,
                       CustomHistogramDataset data)
```

Initializes a new `HistogramChart` instance with data `data`. The input parameter `data` represents a set of plotting data. `CustomHistogramDataset` is a `JFreeChart`-like container class that stores and manages observation sets.

```
public HistogramChart (String title, String XLabel, String YLabel,
                       int[] count, double[] bound)
```

Initializes a new `HistogramChart` instance with data `count` and `bound`. The adjacent categories (or bins) are specified as non-overlapping intervals: bin[j] contains the values in the interval [`bound[j]`, `bound[j+1]`], and `count[j]` is the number of such values. Thus the length of `bound` must be equal to the length of `count` plus one: the last value of `bound` is the right boundary of the last bin.

Methods

```
public HistogramSeriesCollection getSeriesCollection()
```

Returns the chart's dataset.

```
public void setSeriesCollection (HistogramSeriesCollection dataset)
```

Links a new dataset to the current chart.

```
public void setTicksSynchro (int s)
```

Synchronizes x-axis ticks to the s-th histogram bins.

```
public JFrame view (int width, int height)
```

Displays chart on the screen using Swing. This method creates an application containing a chart panel displaying the chart. The created frame is positioned on-screen, and displayed before it is returned. The `width` and the `height` of the chart are measured in pixels.

LaTeX-specific method

```
public String toLatex (double width, double height)
```

ScatterChart

This class provides tools to create and manage scatter plots. Using the `ScatterChart` class is the simplest way to produce scatter plots only. Each `ScatterChart` object is linked with a `XYListSeriesCollection` data set.

```
package umontreal.iro.lecuyer.charts;

public class ScatterChart extends XYChart
```

Constructors

```
public ScatterChart()
```

Initializes a new `ScatterChart` instance with an empty data set.

```
public ScatterChart (String title, String XLabel, String YLabel,
                     double[][]... data)
```

Initializes a new `ScatterChart` instance with data `data`. `title` is a title, `XLabel` is a short description of the x-axis and `YLabel` a short description of the y-axis. The input parameter `data` represents sets of plotting data. For example, if one n-row matrix `data1` is given as argument `data`, then the first row `data1`[0] represents the x-coordinate vector, and every other row `data1`$[i], i = 1, \ldots, n-1$, represents the y-coordinates of a set of points. Therefore matrix `data1` corresponds to $n-1$ sets of points, all with the same x-coordinates. However, one may want to plot sets of points with different x-coordinates. In that case, one should give the points as matrices with two rows. For examples, if the argument `data` is made of three 2-row matrices `data1`, `data2` and `data3`, then they represents three different sets of points, `data*`[0] giving the x-coordinates, and `data*`[1] the y-coordinates of the points.

```
public ScatterChart (String title, String XLabel, String YLabel,
                     double[][] data, int numPoints)
```

Initializes a new `ScatterChart` instance with sets of points `data`. `title` is a title, `XLabel` is a short description of the x-axis, and `YLabel` a short description of the y-axis. If `data` is a n-row matrix, then the first row `data`[0] represents the x-coordinate vector, and every other row `data`$[i], i = 1, \ldots, n-1$, represents a y-coordinate vector. Therefore matrix `data`$[i]$[ ], $i = 0, \ldots, n-1$, corresponds to $n-1$ sets of points, all with the same x-coordinates. However, only *the first* `numPoints` of each set `data`$[i]$ (i.e. the first `numPoints` columns of each row) will be plotted.

```
public ScatterChart (String title, String XLabel, String YLabel,
                     double[][] data, int x, int y)
```

Initializes a new `ScatterChart` instance using subsets of `data`. `data[x][.]` will form the x-coordinates and `data[y][.]` will form the y-coordinates of the chart. `title` sets a title, `XLabel` is a short description of the x-axis, and `YLabel` is a short description of the y-axis. Warning: if the new x-axis coordinates are not monotone increasing, then they will automatically be sorted in increasing order so the points will be reordered, but the original `data` is not changed.

```
public ScatterChart (String title, String XLabel, String YLabel,
                     DoubleArrayList... data)
```

Initializes a new `ScatterChart` instance with data `data`. The input parameter `data` represents a set of plotting data. A `DoubleArrayList` from the Colt library is used to store the data. The description is similar to the above constructor with `double[]... data`.

```
public ScatterChart (String title, String XLabel, String YLabel,
                     XYSeriesCollection data)
```

Initializes a new `ScatterChart` instance with data `data`. The input parameter `data` represents a set of plotting data. `XYSeriesCollection` is a `JFreeChart` container class to store XY plots.

Methods

```
public int add (double[] x, double[] y, String name, String plotStyle)
```

Adds a data series into the series collection. Vector `x` represents the x-coordinates and vector `y` represents the y-coordinates of the series. `name` and `plotStyle` are the name and the plot style associated to the series.

```
public int add (double[] x, double[] y)
```

Adds a data series into the series collection. Vector `x` represents the x-coordinates and vector `y` represents the y-coordinates of the series.

```
public int add (double[] x, double[] y, int numPoints)
```

Adds a data series into the series collection. Vector `x` represents the x-coordinates and vector `y` represents the y-coordinates of the series. Only *the first* `numPoints` of `x` and `y` will be taken into account for the new series.

```
public XYListSeriesCollection getSeriesCollection()
```

Returns the chart's dataset.

```
public void setSeriesCollection (XYListSeriesCollection dataset)
```

Links a new dataset to the current chart.

```
public void setTicksSynchro (int s)
```

Synchronizes X-axis ticks to the s-th series x-values.

```
public JFrame view (int width, int height)
```

Displays chart on the screen using Swing. This method creates an application containing a chart panel displaying the chart. The created frame is positioned on-screen, and displayed before it is returned. The `width` and the `height` of the chart are measured in pixels.

LaTex-specific method

```
public String toLatex (double width, double height)
```

CategoryChart

This class provides tools to create charts from data in a simple way. Its main feature is to produce TikZ/PGF (see WWW link `http://sourceforge.net/projects/pgf/`) compatible source code which can be included in LaTeX documents, but it can also produce charts in other formats. One can easily create a new chart, and customize its appearance using methods of this class, with the encapsulated `SSJCategorySeriesCollection` object representing the data, and an `Axis` object representing the axis. All these classes depend on the `JFreeChart` API (see WWW link `http://www.jfree.org/jfreechart/`) which provides tools to build charts with Java, to draw them, and export them to files. However, only basic features are used here.

Moreover, `CategoryChart` provides methods to plot data using a MATLAB friendly syntax. None of these methods provides new features; they just propose a different syntax to create charts. Therefore some features are unavailable when using these methods only.

```
package umontreal.iro.lecuyer.charts;

public abstract class CategoryChart
```

Methods

`public JFreeChart getJFreeChart()`

Returns the `JFreeChart` object associated with this chart.

`public Axis getYAxis()`

Returns the chart's range axis (y-axis) object.

`public abstract JFrame view (int width, int height);`

`public String getTitle()`

Gets the current chart title.

`public void setTitle (String title)`

Sets a title to this chart. This title will appear on the chart displayed by method `view`.

`public void setAutoRange ()`

Sets chart y range to automatic values.

`private void setManualRange (double[] range)`

Sets new y-axis bounds, using the format: `range` = [`ymin, ymax`].

`public void enableGrid (double xstep, double ystep)`

Puts a grid on the background. It is important to note that the grid is always shifted in such a way that it contains the axes. Thus, the grid does not always have an intersection at

the corner points; this occurs only if the corner points are multiples of the steps: `xstep` and `ystep` sets the step in each direction.

`public void disableGrid()`

Disables the background grid.

Latex-specific methods

`public abstract String toLatex (double width, double height);`

Transforms the chart into LaTeX form and returns it as a `String`.

`public void setLatexDocFlag (boolean flag)`

Same as in `XYChart`.

BoxChart

This class provides tools to create and manage box-and-whisker plots. Each `BoxChart` object is linked with a `BoxSeriesCollection` data set.

A boxplot is a convenient way of viewing sets of numerical data through their summaries: the smallest observation, first quartile ($Q_1 = x_{.25}$), median ($Q_2 = x_{.5}$), third quartile ($Q_3 = x_{.75}$), and largest observation. Sometimes, the mean and the outliers are also plotted.

In the charts created by this class, the box has its lower limit at Q_1 and its upper limit at Q_3. The median is indicated by the line inside the box, while the mean is at the center of the filled circle inside the box. Define the interquartile range as $(Q_3 - Q_1)$. Any data observation which is more than $1.5(Q_3 - Q_1)$ lower than the first quartile or $1.5(Q_3 - Q_1)$ higher than the third quartile is considered an outlier. The smallest and the largest values that are not outliers are connected to the box with a vertical line or "whisker" which is ended by a horizontal line. Outliers are indicated by hollow circles outside the whiskers. Triangles indicate the existence of very far outliers.

```
package umontreal.iro.lecuyer.charts;

public class BoxChart extends CategoryChart
```

Constructors

```
public BoxChart()
```

Initializes a new `BoxChart` instance with an empty data set.

```
public BoxChart (String title, String XLabel, String YLabel,
                 double[] data, int numPoints)
```

Initializes a new `BoxChart` instance with data `data`. `title` is a title, `XLabel` is a short description of the x-axis, and `YLabel` a short description of the y-axis. The input parameter `data` represents a set of plotting data. Only *the first* `numPoints` of `data` will be considered for the plot.

```
public BoxChart (String title, String XLabel, String YLabel,
                 double[]... data)
```

Initializes a new `BoxChart` instance with data `data`. `title` sets a title, `XLabel` is a short description of the x-axis, and `YLabel` is a short description of the y-axis. The input parameter `data` represents a set of plotting data.

Methods

`public int add (double[] data)`

Adds a data series into the series collection. Vector `data` represents a set of plotting data.

`public int add (double[] data, int numPoints)`

Adds a data series into the series collection. Vector `data` represents a set of plotting data. Only *the first* `numPoints` of `data` will be taken into account for the new series.

`public BoxSeriesCollection getSeriesCollection()`

Returns the chart's dataset.

`public void setSeriesCollection (BoxSeriesCollection dataset)`

Links a new dataset to the current chart.

`public void setFillBox (boolean fill)`

Sets `fill` to `true`, if the boxes are to be filled.

`public JFrame view (int width, int height)`

Displays chart on the screen using Swing. This method creates an application containing a chart panel displaying the chart. The created frame is positioned on-screen, and displayed before it is returned. The circle represents the mean, the dark line inside the box is the median, the box limits are the first and third quartiles, the lower whisker (the lower line outside the box) is the first decile, and the upper whisker is the ninth decile. The outliers, if any, are represented by empty circles, or arrows if outside the range bounds.

Latex-specific method

`public String toLatex (double width, double height)`

NOT IMPLEMENTED.

ContinuousDistChart

This class provides tools to plot the density and the cumulative probability of a continuous probability distribution.

```
package umontreal.iro.lecuyer.charts;
import umontreal.iro.lecuyer.probdist.ContinuousDistribution;


public class ContinuousDistChart
```

Constructor

`public ContinuousDistChart (ContinuousDistribution dist, double a, double b, int m)`

Constructor for a new `ContinuousDistChart` instance. It will plot the continuous distribution `dist` over the interval $[a, b]$, using $m + 1$ equidistant sample points.

Methods

`public JFrame viewCdf (int width, int height)`

Displays a chart of the cumulative distribution function (cdf) on the screen using Swing. This method creates an application containing a chart panel displaying the chart. The created frame is positioned on-screen, and displayed before it is returned. The `width` and the `height` of the chart are measured in pixels.

`public JFrame viewDensity (int width, int height)`

Similar to `viewCdf`, but for the probability density instead of the cdf.

`public String toLatexCdf (int width, int height)`

Exports a chart of the cdf to a LaTeX source code using PGF/TikZ. This method constructs and returns a string that can be written to a LaTeX document to render the plot. `width` and `height` represents the width and the height of the produced chart. These dimensions do not take into account the axes and labels extra space. The `width` and the `height` of the chart are measured in centimeters.

`public String toLatexDensity (int width, int height)`

Similar to `toLatexCdf`, but for the probability density instead of the cdf.

`public void setParam (double a, double b, int m)`

Sets the parameters a, b and m for this object.

DiscreteDistIntChart

This class provides tools to plot the mass function and the cumulative probability of a discrete probability distribution over the integers.

```
package umontreal.iro.lecuyer.charts;
import umontreal.iro.lecuyer.probdist.DiscreteDistributionInt;


public class DiscreteDistIntChart
```

Constructors

`public DiscreteDistIntChart (DiscreteDistributionInt dist)`

Constructor for a new `DiscreteDistIntChart` instance used to plot the probabilities of the discrete distribution `dist` over the integers.

`public DiscreteDistIntChart (DiscreteDistributionInt dist, int a, int b)`

Constructor for a new `DiscreteDistIntChart` instance used to plot the probabilities of the discrete distribution `dist` over the interval $[a, b]$.

Methods

`public JFrame viewCdf (int width, int height, int a, int b)`

Displays a chart of the cumulative distribution function (cdf) over the interval $[a, b]$ on the screen using Swing. This method creates an application containing a chart panel displaying the chart. The created frame is positioned on-screen, and displayed before it is returned. The `width` and the `height` of the chart are measured in pixels.

`public JFrame viewCdf (int width, int height)`

Similar to method `viewCdf` above. If the interval $[a, b]$ for the graph is not defined, it will be set automatically to $[\mu - 3\sigma, \mu + 3\sigma]$, where μ and σ are the mean and the variance of the distribution.

`public JFrame viewProb (int width, int height, int a, int b)`

Displays a chart of the probability mass function over the interval $[a, b]$ on the screen using Swing. This method creates an application containing a chart panel displaying the chart. The created frame is positioned on-screen, and displayed before it is returned. The `width` and the `height` of the chart are measured in pixels.

`public JFrame viewProb (int width, int height)`

Similar to method `viewProb` above. If the interval $[a, b]$ for the graph is not defined, it will be set automatically to $[\mu - 3\sigma, \mu + 3\sigma]$, where μ and σ are the mean and the variance of the distribution.

`public void setParam (int a, int b)`

Sets the parameters a and b for this object.

`public String toLatexCdf (int width, int height)`

Exports a chart of the cumulative probability to a LaTeX source code using PGF/TikZ. This method constructs and returns a string that can be written to a LaTeX document to render the plot. `width` and `height` represents the width and the height of the produced chart. These dimensions do not take into account the axes and labels extra space. The `width` and the `height` of the chart are measured in centimeters.

`public String toLatexProb (int width, int height)`

Similar to `toLatexCdf`, but for the probability instead of the cdf.

`public XYLineChart getCdf ()`

Returns the chart of the cdf.

`public XYLineChart getProb ()`

Returns the chart of the probability.

MultipleDatasetChart

Provides tools to plot many datasets on the same chart. This class is mainly used to draw plots with different styles. Class `XYChart` and its subclasses are to be preferred to draw simple charts with one style. Datasets are stored in an `ArrayList`. The first dataset is called as the *primary dataset*.

```
package umontreal.iro.lecuyer.charts;

public class MultipleDatasetChart
```

Constructors

`public MultipleDatasetChart()`
Initializes a new `MultipleDatasetChart`.

`public MultipleDatasetChart (String title, String XLabel, String YLabel)`
Initializes a new `MultipleDatasetChart` instance. `title` sets a title, `XLabel` is a short description of the x-axis, and `YLabel` is a short description of the y-axis.

Methods

`public JFreeChart getJFreeChart()`
Returns the `JFreeChart` variable associated with this chart.

`public Axis getXAxis()`
Returns the chart's domain axis (x-axis) object.

`public Axis getYAxis()`
Returns the chart's range axis (y-axis) object.

`public String getTitle()`
Gets the current chart title.

`public void setTitle (String title)`
Sets a title to the chart. This title will appear on the chart displayed by method `view`.

`public void setAutoRange()`
Sets chart range to automatic values.

`public void setManualRange (double[] axisRange)`
Sets new x-axis and y-axis bounds, with format: `axisRange` = [xmin, xmax, ymin, ymax].

`public int add (SSJXYSeriesCollection dataset)`
Adds a new dataset to the chart at the end of the list and returns its position.

`public SSJXYSeriesCollection get()`

Gets the primary dataset.

`public void set (SSJXYSeriesCollection dataset)`

Sets the primary dataset for the plot, replacing the existing dataset if there is one.

`public SSJXYSeriesCollection get (int datasetNum)`

Gets the element at the specified position in the dataset list.

`public void set (int datasetNum, SSJXYSeriesCollection dataset)`

Replaces the element at the specified position in the dataset list with the specified element.

`public ArrayList<SSJXYSeriesCollection> getList()`

Returns the dataset list.

`public JFrame view (int width, int height)`

Displays chart on the screen using Swing. This method creates an application containing a chart panel displaying the chart. The created frame is positioned on-screen, and displayed before it is returned. The `width` and the `height` of the chart are measured in pixels.

`public void enableGrid (double xstep, double ystep)`

Puts grid on the background. It is important to note that the grid is always placed in such a way that it contains the axes. Thus, the grid does not always have an intersection at the corner points; this occurs only if the corner points are multiples of the stepping. `xstep` and `ystep` sets the stepping in each direction.

`public void disableGrid ()`

Disables the background grid.

LaTeX-specific method

`public String toLatex (double width, double height)`

Same as in `XYChart`.

`public void setLatexDocFlag (boolean flag)`

Same as in `XYChart`.

SSJXYSeriesCollection

Stores data used in a `XYChart`. This class provides tools to manage data sets and rendering options, and modify plot color, plot style, and marks on points for each series.

```
package umontreal.iro.lecuyer.charts;

public abstract class SSJXYSeriesCollection
```

Data control methods

`public double getX (int series, int index)`

Returns the x-value at the specified index in the specified series.

`public double getY (int series, int index)`

Returns the y-value at the specified index in the specified series.

`public XYDataset getSeriesCollection()`

Returns the `XYDataset` object associated with the current object.

`public double[] getDomainBounds()`

Returns domain (x-coordinates) min and max values.

`public double[] getRangeBounds()`

Returns range (y-coordinates) min and max values.

`public String toString()`

Returns in a `String` all data contained in the current object.

Rendering methods

`public XYItemRenderer getRenderer()`

Returns the `XYItemRenderer` object associated with the current object.

`public void setRenderer (XYItemRenderer renderer)`

Sets the `XYItemRenderer` object associated with the current variable. This object determines the chart JFreeChart look, produced by method `view` in class `XYChart`.

`public Color getColor (int series)`

Gets the current plotting color of the selected series.

`public void setColor (int series, Color color)`

Sets a new plotting color to the series *series*.

```
public abstract String toLatex (double XScale, double YScale,
                                double XShift, double YShift,
                                double xmin, double xmax,
                                double ymin, double ymax);
```

Formats and returns a string containing a LaTeX-compatible source code which represents this data series collection. The original datasets are shifted and scaled with the `XShift`, `YShift`, `XScale` and `YScale` parameters. `xmin`, `xmax`, `ymin` and `ymax` represent the chart bounds.

SSJCategorySeriesCollection

Stores data used in a `CategoryChart`. This class provides tools to manage data sets and rendering options, and modify plot color, plot style, and marks on points for each series.

```
package umontreal.iro.lecuyer.charts;

public abstract class SSJCategorySeriesCollection
```

Data control methods

`public String getCategory (int series)`
Returns the category-value in the specified series.

`public double getValue (int series, int index)`
Returns the y-value at the specified index in the specified series.

`public CategoryDataset getSeriesCollection()`
Returns the `CategoryDataset` object associated with the current object.

`public abstract double[] getRangeBounds();`
Returns range (y-coordinates) min and max values.

`public abstract String toString();`
Returns in a `String` all data contained in the current object.

Rendering methods

`public CategoryItemRenderer getRenderer()`
Returns the `CategoryItemRenderer` object associated with the current object.

`public void setRenderer (CategoryItemRenderer renderer)`
Sets the `CategoryItemRenderer` object associated with the current variable. This object determines the chart JFreeChart look, produced by method `view` in class `XYChart`.

`public Color getColor (int series)`
Gets the current plotting color of the selected series.

`public void setColor (int series, Color color)`
Sets a new plotting color to the series *series*.

```
public abstract String toLatex (double YScale, double YShift,
                                double ymin, double ymax);
```

Formats and returns a string containing a LaTeX-compatible source code which represents this data series collection. The original datasets are shifted and scaled with the `YShift` and `YScale` parameters. `ymin` and `ymax` represent the chart bounds.

XYListSeriesCollection

This class extends `SSJXYSeriesCollection`. It stores data used in a `XYLineChart` or in other related charts. `XYListSeriesCollection` provides complementary tools to draw simple curves; for example, one may add or remove plots series and modify plot style. This class is linked with the JFreeChart `XYSeriesCollection` class to store data plots, and linked with the JFreeChart `XYLineAndShapeRenderer` to render the plot. Each series must contain enough points to plot a nice curve. It is recommended to use about 30 points. However, some rapidly varying functions may require many more points. This class can be used to draw scatter plots.

```
package umontreal.iro.lecuyer.charts;

public class XYListSeriesCollection  extends SSJXYSeriesCollection
```

Constructors

`public XYListSeriesCollection()`

Creates a new `XYListSeriesCollection` instance with an empty dataset.

`public XYListSeriesCollection (double[][]... data)`

Creates a new `XYListSeriesCollection` instance with default parameters and given data series. The input parameter `data` represents a set of plotting data.

For example, if one n-row matrix `data1` is given as argument, then the first row `data1`$[0]$ represents the x-coordinate vector, and every other row `data1`$[i], i = 1, \ldots, n-1$, represents a y-coordinate set for the points. Therefore matrix `data1`$[i][j]$, $i = 0, \ldots, n-1$, corresponds to $n-1$ curves, all with the same x-coordinates.

However, one may want to plot several curves with different x-coordinates. In that case, one should give the curves as matrices with two rows. For examples, if the argument `data` is made of three 2-row matrices `data1`, `data2` and `data3`, then they represents three different curves, `data*`$[0]$ being the x-coordinates, and `data*`$[1]$ the y-coordinates of the curves.

However, we may also consider the sets of points above not as part of curves, but rather as several list of points.

`public XYListSeriesCollection (double[][] data, int numPoints)`

Creates a new `XYListSeriesCollection` instance with default parameters and given points `data`. If `data` is a n-row matrix, then the first row `data`$[0]$ represents the x-coordinate vector, and every other row `data`$[i], i = 1, \ldots, n-1$, represents a y-coordinate set of points. Therefore, if the points represents curves to be plotted, `data`$[i][\,], i = 0, \ldots, n-1$, corresponds to $n-1$ curves, all with the same x-coordinates. Only the first `numPoints` of `data` will be considered for each of the set of points.

`public XYListSeriesCollection (DoubleArrayList... data)`

Creates a new `XYListSeriesCollection` instance with default parameters and given data. The input parameter represents a set of data plots, the constructor will count the occurrence

number Y of each value X in the `DoubleArrayList`, and plot the point (X, Y). Each `DoubleArrayList` variable corresponds to a curve on the chart.

`public XYListSeriesCollection (XYSeriesCollection data)`

Creates a new `XYListSeriesCollection` instance with default parameters and given data series. The input parameter represents a set of plotting data. Each series of the given collection corresponds to a curve on the plot.

Data control methods

`public int add (double[] x, double[] y)`

Adds a data series into the series collection. Vector `x` represents the x-coordinates and vector `y` represents the y-coordinates of the series.

`public int add (double[] x, double[] y, int numPoints)`

Adds a data series into the series collection. Vector `x` represents the x-coordinates and vector `y` represents the y-coordinates of the series. Only *the first* `numPoints` of `x` and `y` will be added to the new series.

`public int add (double[][] data)`

Adds a data series into the series collection. The input format of `data` is described in constructor `XYListSeriesCollection(double[][] data)`.

`public int add (double[][] data, int numPoints)`

Adds data series into the series collection. The input format of `data` is described in constructor `XYListSeriesCollection(double[][] data)`. Only *the first* `numPoints` of `data` (the first `numPoints` columns of the matrix) will be added to each new series.

`public int add (DoubleArrayList data)`

Adds a data series into the series collection. The input format of `data` is described in constructor `XYListSeriesCollection (DoubleArrayList... data)`.

`public String getName (int series)`

Gets the current name of the selected series.

`public void setName (int series, String name)`

Sets the name of the selected series.

Rendering methods

`public void enableAutoCompletion()`

Enables the auto completion option. When this parameter is enabled, straight lines are used to approximate points on the chart bounds if the method isn't able to display all points, because the user defined bounds are smaller than the most significant data point coordinate, for instance. It does not extrapolate the point sets, but simply estimates point coordinates on the curve at bound positions for a better visual rendering.

`public void disableAutoCompletion()`

Disables auto completion option. Default status is `disabled`.

`public String getMarksType (int series)`

Returns the mark type associated with the `series`th data series.

`public void setMarksType (int series, String marksType)`

Adds marks on points to a data series. It is possible to use all the marks provided by the TikZ package, some of which are `*`, `+` and `x`. A blank character, used by default, will disable marks. The PGF/TikZ documentation provides more information about placing marks on plots.

`public String getDashPattern (int series)`

Returns the dash pattern associated with the `series`th data series.

`public void setDashPattern (int series, String dashPattern)`

Selects dash pattern for a data series. It is possible to use all the dash options provided by the TikZ package: solid, dotted, densely dotted, loosely dotted, dashed, densely dashed and loosely dashed.

`public String getPlotStyle (int series)`

Gets the current plot style for the selected series.

`public void setPlotStyle (int series, String plotStyle)`

Selects the plot style for a given series. It is possible to use all the plot options provided by the TikZ package. Some of which are: `sharp plot`, which joins points with straight lines, `smooth`, which joins points with a smoothing curve, `only marks`, which does not join points, etc. The PGF/TikZ documentation provides more information about smooth plots, sharp plots and comb plots.

```
public String toLatex (double XScale, double YScale,
                       double XShift, double YShift,
                       double xmin, double xmax,
                       double ymin, double ymax)
```

YListSeriesCollection

This class extends `XYListSeriesCollection`. The data is given as lists of y-coordinates. The x-coordinates are simply the indices + 1 of each y-coordinate in the given lists.

```
package umontreal.iro.lecuyer.charts;

public class YListSeriesCollection extends XYListSeriesCollection
```

Constructors

`public YListSeriesCollection (double[]... data)`

Creates a new `YListSeriesCollection` instance with default parameters and given data series. The input vectors represent sets of plotting data. More specifically, each vector `data` represents a y-coordinates set. Position in the vector will form the x-coordinates. Indeed the value `data`$[j]$ corresponds to the point $(j + 1, \texttt{data}[j])$ on the chart.

`public YListSeriesCollection (double[] data, int numPoints)`

Creates a new `YListSeriesCollection` instance with default parameters and one data series. The vector `data` represents a set of plotting data. Position in the vector represents the x-coordinate of the points: thus the coordinates of the points are given by $(j+1, \texttt{data}[j])$. However, only *the first* `numPoints` of `data` will be considered in the series.

`public YListSeriesCollection (double[][] data, int numPoints)`

Creates a new `YListSeriesCollection` instance with default parameters and given data series. The matrix `data` represents a set of plotting data. More specifically, each row of `data` represents a y-coordinates set. Position in the vector will form the x-coordinates. Indeed, for each serie i, the value `data`$[i][j]$ corresponds to the point $(j + 1, \texttt{data}[j])$ on the chart. However, only *the first* `numPoints` of `data` will be considered for each series of points.

EmpiricalSeriesCollection

Stores data used in a `EmpiricalChart`. `EmpiricalSeriesCollection` provides complementary tools to draw empirical distribution charts, like add plots series. This class is linked with `XYSeriesCollection` class from JFreeChart to store data plots, and linked with JFreeChart `EmpiricalRenderer` to render the plot. `EmpiricalRenderer` has been developed at the Université de Montréal to extend the JFreeChart API, and is used to render charts with an empirical chart style in a JFreeChart chart.

```
package umontreal.iro.lecuyer.charts;

public class EmpiricalSeriesCollection extends SSJXYSeriesCollection
```

Constructors

`public EmpiricalSeriesCollection()`

Creates a new `EmpiricalSeriesCollection` instance with empty dataset.

`public EmpiricalSeriesCollection (double[]... data)`

Creates a new `EmpiricalSeriesCollection` instance with default parameters and given data series. Each input parameter represents an observation set. The values of this observation set *must be sorted* in increasing order.

`public EmpiricalSeriesCollection (double[] data, int numPoints)`

Creates a new `EmpiricalSeriesCollection` instance with default parameters and a given series `data`. The values of `data` *must be sorted* in increasing order. However, only *the first* `numPoints` of `data` will be considered for the series.

`public EmpiricalSeriesCollection (DoubleArrayList... data)`

Creates a new `EmpiricalSeriesCollection` instance with default parameters and given data. The input parameter represents a collection of data observation sets. Each `DoubleArrayList` input parameter represents an observation set. The values of this observation set *must be sorted in increasing order*. Each `DoubleArrayList` variable corresponds to an observation set.

`public EmpiricalSeriesCollection (TallyStore... tallies)`

Creates a new `EmpiricalSeriesCollection` instance with default parameters and given data. The input parameter represents a collection of data observation sets. Each `TallyStore` input parameter represents an observation set.

`public EmpiricalSeriesCollection (XYSeriesCollection data)`

Creates a new `EmpiricalSeriesCollection` instance with default parameters and given data series. The input parameter represents a set of plotting data. Each series of the given collection corresponds to a plot on the chart.

Data control methods

`public int add (double[] observationSet)`

Adds a data series into the series collection.

`public int add (double[] observationSet, int numPoints)`

Adds a data series into the series collection. Only *the first* `numPoints` of `observationSet` will be added to the new series.

`public int add (DoubleArrayList observationSet)`

Adds a data series into the series collection.

`public int add (TallyStore tally)`

Adds a data series into the series collection.

Rendering methods

`public String getMarksType (int series)`

Returns the mark type associated with the `series`-th data series.

`public void setMarksType (int series, String marksType)`

Adds marks on points to a data series. It is possible to use all the marks provided by the TikZ package, some of which are `*`, `+` and `x`. A blank character, used by default, will disable marks. The PGF/TikZ documentation provides more information about placing marks on plots.

`public String getDashPattern (int series)`

Returns the dash pattern associated with the `series`-th data series.

`public void setDashPattern (int series, String dashPattern)`

Selects dash pattern for a data series. It is possible to use all the dash options provided by the TikZ package: solid, dotted, densely dotted, loosely dotted, dashed, densely dashed and loosely dashed.

```
public String toLatex (double XScale, double YScale,
                       double XShift, double YShift,
                       double xmin, double xmax,
                       double ymin, double ymax)
```

HistogramSeriesCollection

Stores data used in a `HistogramChart`. `HistogramSeriesCollection` provides complementary tools to draw histograms. One may add observation sets, define histogram bins, set plot color and plot style, enable/disable filled shapes, and set margin between shapes for each series. This class is linked with class `CustomHistogramDataset` to store data plots, and linked with JFreeChart `XYBarRenderer` to render the plot. `CustomHistogramDataset` has been developed at the Université de Montréal to extend the JFreeChart API, and is used to manage histogram datasets in a JFreeChart chart.

```
package umontreal.iro.lecuyer.charts;

public class HistogramSeriesCollection extends SSJXYSeriesCollection
```

Constructors

`public HistogramSeriesCollection()`

Creates a new `HistogramSeriesCollection` instance with empty dataset.

`public HistogramSeriesCollection (double[]... data)`

Creates a new `HistogramSeriesCollection` instance with given data series. Bins the elements of data in equally spaced containers (the number of bins can be changed using the method `setBins`). Each input parameter represents a data series.

`public HistogramSeriesCollection (double[] data, int numPoints)`

Creates a new `HistogramSeriesCollection` instance with the given data series `data`. Bins the elements of data in equally spaced containers (the number of bins can be changed using the method `setBins`). Only the first `numPoints` of `data` will be taken into account.

`public HistogramSeriesCollection (DoubleArrayList... data)`

Creates a new `HistogramSeriesCollection`. Bins the elements of data in equally spaced containers (the number of bins can be changed using the method `setBins`). The input parameter represents a set of data plots. Each `DoubleArrayList` variable corresponds to a histogram on the chart.

`public HistogramSeriesCollection (TallyStore... tallies)`

Creates a new `HistogramSeriesCollection` instance with default parameters and given data. The input parameter represents a collection of data observation sets. Each `TallyStore` input parameter represents an observation set.

`public HistogramSeriesCollection (CustomHistogramDataset data)`

Creates a new `HistogramSeriesCollection` instance. The input parameter represents a set of plotting data. Each series of the given collection corresponds to a histogram.

Data control methods

`public int add (double[] data)`

Adds a data series into the series collection.

`public int add (double[] data, int numPoints)`

Adds a data series into the series collection. Only *the first* `numPoints` of `data` will be added to the new series.

`public int add (DoubleArrayList observationSet)`

Adds a data series into the series collection.

`public int add (TallyStore tally)`

Adds a data series into the series collection.

`public CustomHistogramDataset getSeriesCollection()`

Returns the `CustomHistogramDataset` object associated with the current variable.

`public List getBins (int series)`

Returns the bins for a series.

`public void setBins (int series, int bins)`

Sets `bins` periodic bins from the observation minimum values to the observation maximum value for a series.

`public void setBins (int series, int bins, double minimum, double maximum)`

Sets `bins` periodic bins from `minimum` to `maximum` for a series. Values falling on the boundary of adjacent bins will be assigned to the higher indexed bin.

`public void setBins (int series, HistogramBin[] binsTable)`

Links bins given by table `binsTable` to a series. Values falling on the boundary of adjacent bins will be assigned to the higher indexed bin.

`public List getValuesList (int series)`

Returns the values for a series.

`public double[] getValues (int series)`

Returns the values for a series.

`public void setValues (int series, List valuesList)`

Sets a new values set to a series from a `List` variable.

`public void setValues (int series, double[] values)`

Sets a new values set to a series from a table.

Rendering methods

`public boolean getFilled (int series)`

Returns the `filled` flag associated with the `series`-th data series.

`public void setFilled (int series, boolean filled)`

Sets the `filled` flag. This option fills histogram rectangular. The fill color is the current series color, alpha's color parameter will be used to draw transparency.

`public double getMargin()`

Returns the margin which is a percentage amount by which the bars are trimmed.

`public void setMargin (double margin)`

Sets the margin which is a percentage amount by which the bars are trimmed for all series.

`public double getOutlineWidth (int series)`

Returns the outline width in pt.

`public void setOutlineWidth (int series, double outline)`

Sets the outline width in pt.

```
public String toLatex (double XScale, double YScale, double XShift,
                       double YShift, double xmin, double xmax,
                       double ymin, double ymax)
```

BoxSeriesCollection

This class stores data used in a `CategoryChart`. It also provides complementary tools to draw box-and-whisker plots; for example, one may add or remove plots series and modify plot style. This class is linked with the JFreeChart `DefaultBoxAndWhiskerCategoryDataset` class to store data plots, and linked with the JFreeChart `BoxAndWhiskerRenderer` to render the plots.

```
package umontreal.iro.lecuyer.charts;

public class BoxSeriesCollection extends SSJCategorySeriesCollection
```

Constructors

`public BoxSeriesCollection ()`

Creates a new `BoxSeriesCollection` instance with an empty dataset.

`public BoxSeriesCollection (double[] data, int numPoints)`

Creates a new `BoxSeriesCollection` instance with default parameters and input series `data`. Only *the first* `numPoints` of `data` will taken into account.

`public BoxSeriesCollection (double[]... data)`

Creates a new `BoxSeriesCollection` instance with default parameters and given data series. The input parameter represents series of point sets.

`public BoxSeriesCollection (DefaultBoxAndWhiskerCategoryDataset data)`

Creates a new `BoxSeriesCollection` instance with default parameters and given data series. The input parameter represents a `DefaultBoxAndWhiskerCategoryDataset`.

Data control methods

`public int add (double[] data)`

Adds a data series into the series collection. Vector `data` represents a point set.

`public int add (double[] data, int numPoints)`

Adds a data series into the series collection. Vector `data` represents a point set. Only *the first* `numPoints` of `data` will be added to the new series.

`public String getName (int series)`

Gets the current name of the selected series.

`public double[] getRangeBounds()`

Returns the range (y-coordinates) min and max values.

```
public String toString()
```

Returns in a `String` all data contained in the current object.

```
public String toLatex (double YScale, double YShift,
                       double ymin, double ymax)
```

NOT IMPLEMENTED: To do.

Axis

Represents an axis of a chart encapsulated by an instance of `XYChart`. `Axis` uses the JFreeChart class `NumberAxis` to store some axis properties. This class represents the x-axis or the y-axis of a `XYChart` and, consequently, is drawn when calling the `toLatex` method. It provides tools to customize the axis in modifying labels and description.

```
package umontreal.iro.lecuyer.charts;


public class Axis

   public static final boolean ORIENTATION_VERTICAL = false;
   public static final boolean ORIENTATION_HORIZONTAL = true;
```

Constructor

`public Axis (NumberAxis inAxis, boolean orientation)`

Create a new `Axis` instance from an existing `NumberAxis` instance with vertical (y-axis) or horizontal (x-axis) orientation.

Methods

`protected NumberAxis getAxis()`

Returns the `NumberAxis` instance (from JFreeChart) linked with the current variable.

`public String getLabel()`

Returns the axis description.

`public void setLabel (String label)`

Sets the axis description. This description will be displayed on the chart, near the axis.

`public void setLabels (double tick)`

Sets a periodic label display. Labels will be shown every `tick` unit. This tick unit replaces the default unit even though manual labeling is disabled.

`public void setLabelsAuto()`

Calculates and sets an automatic tick unit.

LaTex-specific methods

`public void enableCustomLabels()`

Enables the use of custom labels for the current axis. The method `setLabels` must be used to set the custom labels.

`public void disableCustomLabels()`

Disables the use of custom labels for the current axis.

`public void setLabels (double[] position)`

Sets the position of each label on this axis. This method requires an array containing an increasing sequence of numbers representing the positions at which labels will appear on the axis. It is designed to export the axis to a PGF/TikZ source code; it has no effect on the chart appearance displayed with `XYChart.view()`. Method `enableCustomLabels` *must* be called before using this method.

`public void setLabels (double[] position, String[] label)`

Assigns custom labels to user-defined positions on the axis. This method requires an array of positions as well as an array of labels. The label `label[i]` will be used at position `position[i]`. It is designed to export the axis to a PGF/TikZ source code, and to use LaTeX/TikZ commands to write prettier characters; it has no effect on the chart appearance displayed with `XYChart.view()`. Method `enableCustomLabels` *must* be called before using this method.

`public double getTwinAxisPosition()`

Returns the drawing position parameter (default equals 0).

`public void setTwinAxisPosition (double position)`

Defines where the opposite axis must be drawn on the current axis, where it should appear, and on which label.

`public String toLatex (double scale)`

Formats and returns a string containing a LaTeX-compatible source code which represents this axis and its parameters.

PlotFormat

Provide tools to import and export data set tables to and from Gnuplot, MATLAB and Mathematica compatible formats, or customized format.

```
package umontreal.iro.lecuyer.charts;
```

```
public class PlotFormat
```

Data import functions

`public static double[][] fromGNUPlot (String data)`

Parses `data` according to the standard GNUPlot format and stores the extracted values in the returned table. All unrecognized characters and empty lines will be cut. Deleting comments inside the input `String` is recommended.

`public static double[][] fromCSV (String data)`

Parses `data` according to the standard CSV format and stores the extracted values in the returned table.

```
public static double[][] fromCustomizedFormat (String betweenValues,
                                               String endLine, String data)
```

Parses `data` according to a user defined format and stores the extracted values in the returned table. `betweenValues` sets characters between values on the same line and `endLine` sets characters which separates each line of the input data set. Usually, this parameter contains the classic end of line character `%n`. This method uses regular expressions, so it is possible to use regular characters as defined in the standard java API, specially in the class `Pattern` (package java.util.regex).

Data export functions

`public static String toGNUPlot (double[]... data)`

Stores data tables `data` into a `String`, with format understandable by GNUPlot.

`public static String toGNUPlot (XYSeriesCollection data)`

Stores series collection `data` into a `String`, with format understandable by GNUPlot.

`public static String toCSV (double[]...data)`

Stores data tables `data` into a `String` with format CSV (comma-separated value tabular data). The output string could be imported from a file to a matrix into Mathematica with Mathematica's function `Import["fileName", "CSV"]`, or into MATLAB with MATLAB's function `csvread('fileName')`.

`public static String toCSV (XYSeriesCollection data)`

Stores series collection `data` into a `String` with format CSV (comma-separated value tabular data).

```
public static String toCustomizedFormat (String heading, String footer,
                                         String betweenValues, String endLine,
                                         int precision, double[]...data)
```

Stores data tables `data` into a `String` with customized format. `heading` sets the head of the returned `String`, `footer` sets its end, `betweenValues` sets characters between values on the same line and finally `endLine` sets characters which separates each line of the input data set. Normally, this parameter contains the classic end of line character '%n'.

```
public static String toCustomizedFormat (String heading, String footer,
                                         String betweenValues, String endLine,
                                         int precision, XYSeriesCollection data)
```

Stores data tables `data` into a `String` with customized format from an `XYSeriesCollection` variable.