Chapter 1

Charting, Plotting and Curving using Grapher

Graphing, plotting, and curving data points is the basis of a great deal of analyses. Making interactive graphs and composing them with other visualizations have received a strong emphasize in Roassal. Roassal provides Grapher: an API implemented as a builder dedicated to draw charts. A graph is represented as an instance of the class RTGrapher. A set of data points is represented by an instance of the class RTDataSet and RTStackedDataSet. A graph is therefore built by ading data sets in a grapher object.

We will first present the three different kinds of charts that can be built using Grapher, then discuss fine-tuning the chart, and end with a word on the implementation.

1.1 Scatterplot

A scatterplot is a collection of cartesian coordinates to display values for two variables in a data set.

Simple chart

Grapher offers a flexible way to draw scatterplots. Consider the following contrivied example, to be evaluated in a playground:

```
b := RTGrapher new.
ds := RTDataSet new.
ds points: { 4 @ 5 . 10 @ 5 . 3 @ 2 }.
ds dotShape size: 10; color: (Color blue alpha: 0.3).
```

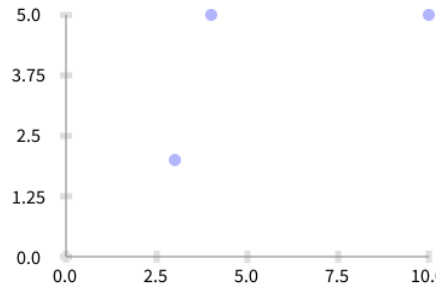

Figure 1.1: Contrivied scatterplot.

```
ds x: #x.
ds y: #y.
b add: ds.

b
```

Figure 1.1 shows the result of the script above. First, an instance of the class RTGrapher is obtained and the variable b is assigned to it. We then create a data set by instantiating RTDataSet. We give as data point two points (we recall that 4 @ 5 corresponds to the point (4, 5)). Each data point is represented as a dot, of size 10 and blue color. Each point answers to the message x and y, which is why we have locate each points along its x and y values. Finally, the data set is added in the grapher.

A data set for which each element has to be plotted using two functions, one against the X-axis and another against the Y-axis), is modelled using the class RTDataSet.

Consider this slightly more elaborated example that relate depth and seism magnitude:

```
tab := RTTabTable new input:  'http://earthquake.usgs.gov/earthquakes/feed/v0.1/
    summary/2.5_month.csv' asUrl retrieveContents usingDelimiter: $,.
tab removeFirstRow.
tab replaceEmptyValuesWith: '0' inColumns: #(4 5).
tab convertColumnsAsFloat: #(4 5).

b := RTGrapher new.
ds := RTDataSet new.
ds interaction popup.
ds points: tab values.
ds dotShape color: (Color blue alpha: 0.3).
ds x: [ :row | row at: 4 ].
ds y: [ :row | row at: 5 ].
```

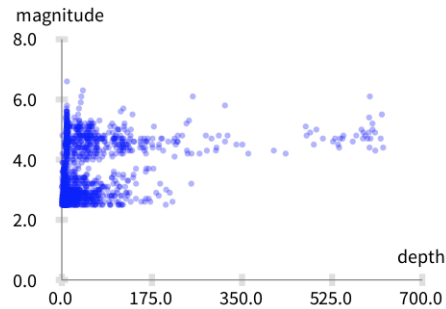

Figure 1.2: Depth and magnitude of seisms greater than 2.5 during the last 30 days.

```
b add: ds.

b maxY: 8.
b maxX: 700.

b axisX title: 'depth'.
b axisY title: 'magnitude'.
b
```

Figure 1.2 is an extension of the previous script. It simply fetches CSV data from a server. Simply put the URL in a web browser to see what the data looks like. After having created the table, the column names (*i.e.,* the first row) is removed from it. Columns 4 (depth) and 5 (magnitude) are converted as float numbers. The x value is obtained by fetching the 4th element of a CSV row, and the y value from the 5th element. Note that no seism below an intensity of 2.5 is represented.

Another example that visualize methods:

```
methods := Collection withAllSubclasses flatCollect: #methods.

b := RTGrapher new.

ds := RTDataSet new.
ds dotShape circle color: (Color red alpha: 0.3).
ds points: methods.
ds x: #numberOfLinesOfCode.
ds y: [ :m | m getSource size ].
b add: ds.

b build.
```

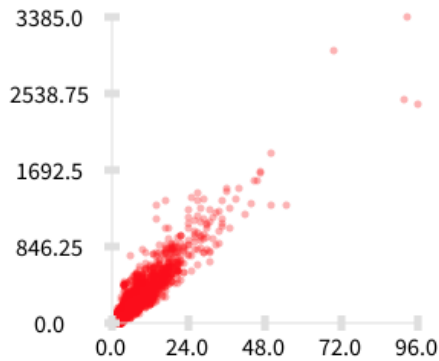

Figure 1.3: Scatterplot built by hand.

Figure 1.3 illustrates the result of the script given above. In the code, the variable methods contains all the methods of the Pharo Collection class hierarchy. Note that approximately 3,500 methods are defined in the hierarchy.

Each data point will be represented as a circle, using the default size of 5 pixels, and a translucent red. The collection of data points is specified using points:.

The x-value and y-value have to be computed for each data point. The method objects contained in the methods collection answer to the message numberOfLinesOfCode, returning the number of lines of code for the method definition. The block [:m | m getSource size] returns the number of characters that define the method.

Figure 1.3 reveals an obvious correlation between the number of lines of code and the number of characters of the method. Deviation from the diagonal indicates methods with either very long or very short lines of code.

Multiple charts

Charter supports different data set to be simultaneously displayed. Consider the following example:

```
methods := Collection withAllSubclasses flatCollect: #methods.
trachelMethods := TRObject withAllSubclasses flatCollect: #methods.

b := RTGrapher new.

"Data set 1"
ds1 := RTDataSet new.
ds1 dotShape circle color: (Color red alpha: 0.3).
```

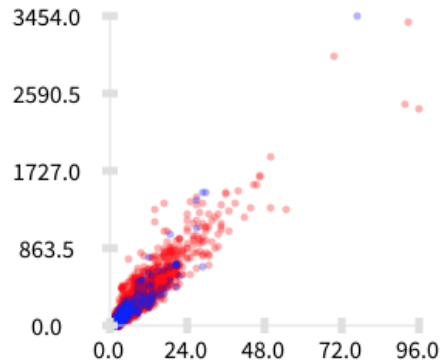

Figure 1.4: Two data set in the same chart.

```
ds1 points: methods.
ds1 x: #numberOfLinesOfCode.
ds1 y: [ :m | m getSource size ].
b add: ds1.

"Data set 2"
ds2 := RTDataSet new.
ds2 dotShape circle color: (Color blue alpha: 0.3).
ds2 points: trachelMethods.
ds2 x: #numberOfLinesOfCode.
ds2 y: [ :m | m getSource size ].
b add: ds2.

b build.
```

Figure 1.4 shows two colored data sets. The first data set, the methods of the Collection class hierarchy (methods), is colored in red. The second data set, all the methods defined in Trachel (trachelMethods), are colored in blue.

Axis configuration

Properly defining the X and Y axes is a complex task. A great deal of parameters usually have to be taken into account. By default, both axes have four ticks, and each tick is labelled with a numerical value with a precision of one decimal point. Charter however offers a number of configuration options for the configuration of Axis. For example, in our situation, both axes have to be labelled with integer values: the number of lines and the size of method definitions are integer values. Consider the example seen previously, for which only integer values with a comma as thousand separators:

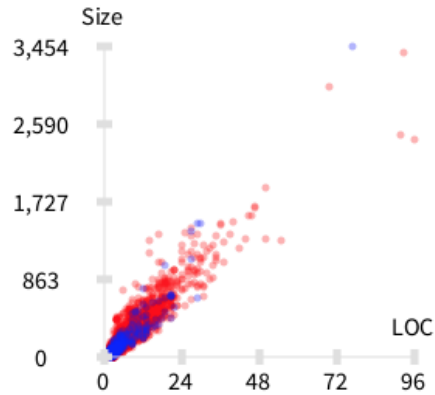

Figure 1.5: Axis configuration.

```
...
"Axis configuration"
b axisX title: 'LOC'; withThousandsSeparator.
b axisY title: 'Size'.
b build.
```

A number of options may be provided to an axis:

- title: to specificy a title to the axis.
- noDecimal to not have decimal values as label on the axis.
- noLabel to not have any label.
- twoDecimals to have two decimal numbers.
- withThousandsSeparator to have thousand separation in the labels
- rotateLabels to rotate the label by 45 degrees. Useful on the X-axis.
- decimal: to specify the number of decimal number to have.

1.2 Curve

A curve is obtained by connecting data points with a line. Curves may be obtained intensionally by giving a function or extensionally by providing the data points.

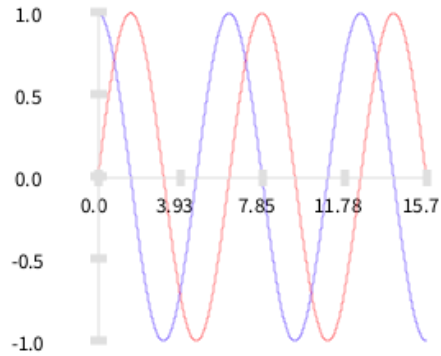

Figure 1.6: Curves defined as functions.

Function

Consider the following script:

```
b := RTGrapher new.

ds := RTDataSet new.
ds noDot.
ds points: (0 to: 3.1415 * 5 by: 0.01).
ds y: #sin.
ds x: #yourself.
ds connectColor: (Color red alpha: 0.4).
b add: ds.

ds := RTDataSet new.
ds noDot.
ds points: (0 to: 3.1415 * 5 by: 0.01).
ds y: #cos.
ds x: #yourself.
ds connectColor: (Color blue alpha: 0.4).
b add: ds.

b build.
```

Note that #sin is rigorously equivalent to [:v | v sin]. Simply shorter.

Each data point is defined as a rectangle of size 0. Which means that each datapoint will not be drawn. Instead, they are linked to each other via a connecting line, as defined by the new keyword here connectDotColor:. A Line is drawn between the dots in the order they were provided to points:.

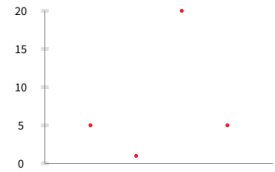

Figure 1.7: Simple stack of data points.

Stacking

Data points may be stacked, meaning that the index of the point in the collection is its X value. This is useful in case that data points are not obtained by applying a function. Consider the following example: it shows 4 data points horizontally ordered in the same order as they were provided to points:, as seen in Figure 1.7.

```
b := RTGrapher new.

ds := RTStackedDataSet new.
ds dotShape color: Color red.
ds points: #(5 1 20 5).
ds y: #yourself.
b add: ds.

b axisX noLabel; noTick.
b axisY noDecimal.
b build.
```

A slightly more elaborate example is given below. Each curve is a class contained in Roassal. Each data point is a method, sorted in a reverse alphabetical order. The Y-value of a method is its size in number of lines of code.

```
classes := RTShape withAllSubclasses.

b := RTGrapher new.
b extent: 400 @ 200.

normalizer := RTMultiLinearColorForIdentity new objects: classes.
classes do: [ :c |
   ds := RTStackedDataSet new.
   ds points: (c methods reverseSortedAs: #numberOfLinesOfCode ).
   ds interaction popup.
   ds dotShape rectangle color: (normalizer rtValue: c).
   ds connectColor: (normalizer rtValue: c).
   ds y: #numberOfLinesOfCode.
```

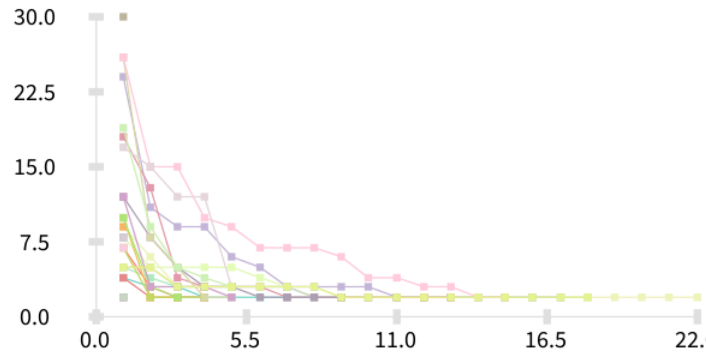

Figure 1.8: Stacking data points.

```
    b add: ds.
].

b build.
```

Figure 1.8 shows 27 different curves, each representing a subclass of the class RTShape.

Also, a distinct color is given to each curve. This is useful to be able to differentiate classes. To achieve this, a RTMultiLinearColorForIdentity object has to be initialized with the objects that will be colored. The message objects: is used for that purpose. The expression (normalizer rtValue: c) will return a color that is specific for the argument c.

Stacking or not stacking a data set

So far, a data set has been presented as a list of objects and two metrics specifying using x: and y:. The two values for each data point are then computed using the metrics.

A stacked data set is a data set for which only the list of objects and the y: metric are provided. How to determine the X value is not provided.

A scatterplot and a bar chart do not represent a data set in the same fashon. A bar chart needs a list of values to be represented (*e.g.*, 5, 6, 7). A scatterplot needs a list of two coordinates (*e.g.*, 2 @ 3, 5 @ 6).

Consider the following script:

```
b := RTGrapher new.

ds := RTDataSet new.
```

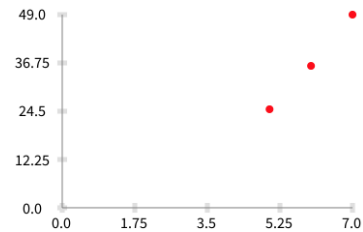

Figure 1.9: Not stacked data set.

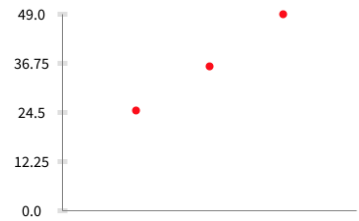

Figure 1.10: Stacked data set.

```
ds dotShape size: 8; color: Color red.
ds points: #(5 6 7).
ds x: [ :v | v ].
ds y: [ :v | v * v ].
b add: ds.

b
```

In Figure 1.9, each point has both its X and Y values that are computed.

Consider this slightly modified version:

```
b := RTGrapher new.

ds := RTStackedDataSet new.
ds dotShape size: 8; color: Color red.
ds points: #(5 6 7).
ds y: [ :v | v * v ].
b add: ds.

b axisX noTick; noLabel.
b
```

In Figure 1.10, each point has both its Y value that is computed. The X value is determined from the order of the objects provided to points:. Stacked

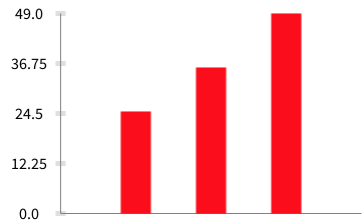

Figure 1.11: Stacked data set.

data set are useful in bar charts, as in the following example:

```
b := RTGrapher new.

ds := RTStackedDataSet new.
ds barShape size: 30; color: Color red.
ds points: #(5 6 7).
ds y: [ :v | v * v ].
b add: ds.

b axisX noTick; noLabel.
b
```

Figure 1.10 shows the result of the previous script. Bars are used instead of dots.

Stacking multiple curve

Consider two data sets #(1 1 4 6) and #(2 4 2 10 5 2) (Figure 1.12):

```
points1 := #(1 1 4 6).
points2 := #(2 4 2 10 5 2).

b := RTGrapher new.

ds := RTStackedDataSet new.
ds points: points1.
ds connectColor: Color blue.
ds y: #yourself.
b add: ds.

ds := RTStackedDataSet new.
ds points: points2.
ds connectColor: Color green.
ds y: #yourself.
b add: ds.
```

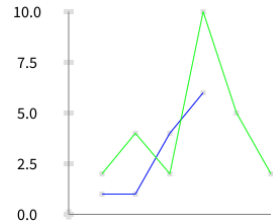

Figure 1.12: Stacking multiple curve.

```
b axisX noLabel; noTick.
b build
```

Charting these two sets requires one single call to stackX. The metric function for the Y-axis has to be specified using y: (simply #yourself in our example).

Data point aspect

> **Book Todo:**
> this section is already too big and this info is already relevant for scatterplots. Why not move it back: make the example with different colors in the Multiple charts section of scatterplot also use different shapes? (JF)

The visual aspects of data points may be customized. Consider the following example.

```
b := RTGrapher new.

ds := RTStackedDataSet new.
ds dotShape rectangle color: Color red.
ds points: (RTShape withAllSubclasses sortedAs: #ageInDays).
ds y: [ :c | c ageInDays ].
b add: ds.

ds := RTStackedDataSet new.
ds dotShape circle color: Color blue.
ds points: (TRShape withAllSubclasses sortedAs: #ageInDays).
ds y: [ :c | c ageInDays ].
b add: ds.
```

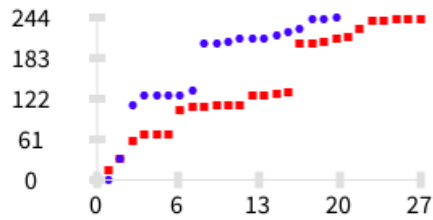

Figure 1.13: Data point aspect.

```
b axisX noDecimal.
b axisY noDecimal.

b build.
```

Figure 1.13 represents the age of shape classes contained in Trachel (*i.e.,* subclasses of the class TRShape) and Roassal (*i.e.,* subclasses of the class RTShape). Trachel shapes are older than Roassal shapes.

1.3 Bar

RTCharterBuilder supports bar charts. The following example displays a simple bar chart (Figure 1.14):

```
b := RTGrapher new.

ds := RTStackedDataSet new.
ds interaction popup.
ds points: #(10 5 4).
ds y: #yourself.
ds barShape rectangle color: Color blue; width: 15.
b add: ds.

b axisX noLabel; noTick.
b build.
```

In a bar chart, data points are typically not visible. Setting a size of 0 will make data points not apparent. The use of the interaction popup makes the bars react when the mouse is above a bar. Three numerical values are provided, and the height of a bar is given by the value itself. Bar are stacked from left to right and are colored in blue.

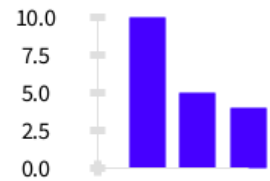

Figure 1.14: Simple bar chart.

Book Todo:
There is too much magic here. Is the histogram method responsible for the bar-chart-nature? What's with the b shape rectangle color: Color blue. then? Do I need to do this before the histogram or can I also do it after? What are the other magic messages that make a bar chart and what is their semantics? Also, the interaction stuff should be moved to the next section. (JF)

1.4 Interaction

Interactions may be defined to get particular behavior upon user actions. Typical cases is getting a value when the mouse hovers a point or a bar.

Book Todo:
This should be clear that it's for all kinds of charts. Can you add a simple example for scatterplot and for line? (JF)

Consider the following code (Figure 1.15):

```
b := RTGrapher new.

ds := RTStackedDataSet new.
ds barShape color: Color veryLightGray; width: 10.
ds points: (RTShape withAllSubclasses).
ds y: [ :c | c methods size ].
ds interaction highlightColored: Color red.

ds interaction popup
   named: [ :c | c name, ' methods']
   background: Color veryVeryLightGray
   group: [ :group :element |
```

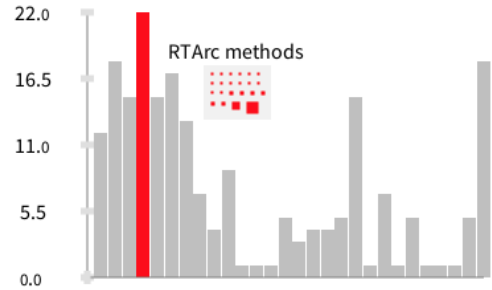

Figure 1.15: Interaction in a bar chart.

```
        | s ms |
        s := RTBox new size: #numberOfLinesOfCode; color: Color red.
        ms := element model methods sortedAs: #numberOfLinesOfCode.
        group addAll: (s elementsOn: ms).
        RTGridLayout on: group ].

    b add: ds.

    b axisX noTick; noLabel.
    b build.
```

Two interactions are defined on each bar. First, the bar on which the mouse is above is red. The original bar color is restored when the mouse leaves the bar. The second interaction is a grouped popup, which has a name, a background color, and a set of method elements. The size of a method reflects the number of lines of code defining the method.

1.5 Distribution

1.6 Logarithmic Scale

1.7 Axis Range

1.8 Date on the axis

Dates are particular values that requires an adequate control over what is being displayed on the X-axis. This is enabled using the julianDayNumber message on a date object, converting it into a number.

Consider the following example that shows the creation of methods along time (Figure 1.16):

```
methods := RTObject withAllSubclasses flatCollect: #methods.
methods := methods reject: [ :m | m numberOfLinesOfCode > 150 ].
oldestMethod := methods minFor: #date.

b := RTGrapher new.

ds := RTDataSet new.
ds dotShape circle size: 5; color: (Color blue alpha: 0.1).
ds interaction popup.
ds points: methods.
ds y: #numberOfLinesOfCode.
ds x: [ :m | m date julianDayNumber − oldestMethod date julianDayNumber ].
b add: ds.

b axisY title: 'LOC'; noDecimal.

b axisX
   title: '';
   labelRotation: −30;
   numberOfTicks: 10;
   numberOfLabels: 10;
   labelConvertion: [ :v | (Date julianDayNumber: v + oldestMethod date
   julianDayNumber) ].
b build
```

Book Todo:
the code sais numberOfTicks: 10; numberOfLabels: 10; but this is not what's shown. (JF)

1.9 Line Shape

Book Todo:
dashed, thickness

1.10 Internal of RTCharterBuilder

Drawing charts often implies normalizing elements over a well determined range of values. Typically the size of the chart. The class RTCharterBuilder is built on the principle illustrated in the following code:

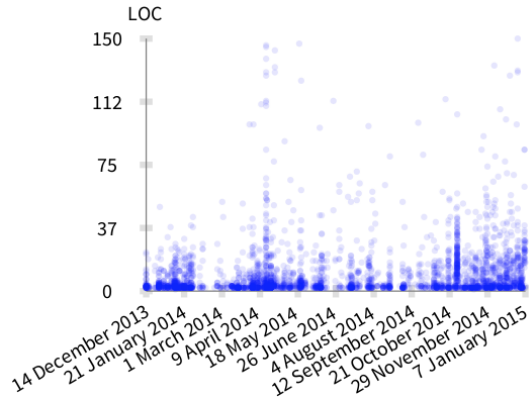

Figure 1.16: Date on the X-axis.

```
methods := Collection withAllSubclasses flatCollect: #methods.
v := RTView new.
es := (RTEllipse new color: Color red) elementsOn: methods.

v addAll: es.
RTMetricNormalizer new
    elements: es;
    alphaColor: 0.2;
    normalizeX: #numberOfLinesOfCode min: 0 max: 300;
    normalizeY: [ :m | m getSource size ] min: 0 max: -300.
v
```

Figure 1.17 illustrates the result of the script. Each position coordinate ranges over a ordered set of values.

Axis

Each axis is described by an instance of RTAxisConfiguration.

Bar chart

Rectangles are located below the data point.

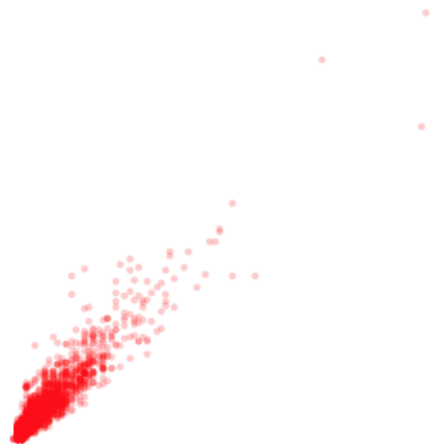

Figure 1.17: Scatterplot.