Typst-plotting

Auto generated documentation

17.6.2023

Pegacraffft **Gewi**

Abstract

Typst-plotting is a plotting library for Typst.
It supports drawing the following plots/graphs in a variety of styles.

- Scatter plots
- Line charts
- Histograms
- Bar charts
- Pie charts
- Overlaying plots/charts

More features will be added over time. If you have some feedback, let us know!

Contents

Docs were created with typst-doc

Axes

axis

This is the constructor function for creating axes. Most plots/graphs will require axes to function.

Basics

The most important parameters are `min`, `max`, `step` and `location`. These need most likely be changed for a functioning axis. If `min`, `max` and `step` are set, the `values` parameter will automatically be filled with the correct values.
Example:

```
let x_axis = axis(min: 0, max: 11, step: 2, location: "bottom")
```

will cause `values` to look like this:

```
(0, 2, 4, 6, 8, 10)
```

If you want to specify your own values, for example when using text on an axis, you need to specify `values` by yourself. Custom specified values could look like this `("", "male", "female", "divers", "unknown")` (the first empty string is not neccessary, but will make some graphs/plots look a lot better).

You can obviously do a lot more than just this, so I recommend taking a look at the examples.

Examples

An x-axis for different genders:

```
let gender_axis_x = axis(
    values: ("", "m", "w", "d"),
    location: "bottom",
    helper_lines: true,
    invert_markings: false,
    title: "Gender"
)
```

A y-axis displaying ascending numbers:

```
let y_axis_2 = axis(min: 0, max: 41, step: 10,
  location: "left", show_markings: true, helper_lines: true)
```

Parameters

```
axis(
  min: integer,
  max: integer,
  step: integer,
  values: array,
  location: string,
  show_values: boolean,
  show_markings: boolean,
  invert_markings: boolean,
  marking_offset_left: integer,
  marking_offset_right: integer,
  stroke: length color dictionary stroke,
  marking_color: color,
  value_color: color,
  helper_lines: boolean,
  helper_line_style: string,
  helper_line_color: color,
  marking_length: length,
  marking_number_distance: length,
  title: content
)
```

min `integer`

From where `values` should started generating (inclusive)

Default: `0`

max `integer`

Where `values` should stopped being generated (exclusive)

Default: `0`

step `integer`

The steps that should be taken when generating `values`

Default: `1`

values `array`

The values of the markings (exclusive with `min`, `max` and `step`)

Default: `()`

location `string`

The position of the axis. Only valid options are: `"top", "bottom", "left", "right"`

Default: `"bottom"`

show_values `boolean`

If the values should be displayed

Default: `true`

show_markings `boolean`

If the markings should be displayed

Default: `true`

invert_markings `boolean`

If the markins should point away from the data (outwards)

Default: `false`

marking_offset_left `integer`

Amount of hidden markings from the left or bottom

Default: `1`

marking_offset_right `integer`

Amount of hidden markings from the right or top

Default: `0`

stroke `length` or `color` or `dictionary` or `stroke`

The color of the baseline for the axis

Default: `black`

marking_color `color`

The color of the marking

Default: `black`

value_color `color`

The color of a value

Default: `black`

helper_lines `boolean`

If helper lines (to see better alignment of data) should be displayed

Default: `false`

helper_line_style `string`

The style of the helper lines, valid options are: `"solid", "dotted", "densely-dotted", "loosely-dotted", "dashed", "densely-dashed", "loosely-dashed", "dash-dotted", "densely-dash-dotted", "loosely-dash-dotted"`

Default: `"dotted"`

helper_line_color `color`

The color of the helper line

Default: `gray`

marking_length `length`

The length of a marking in absolute size

Default: `5pt`

marking_number_distance `length`

The distance between the marker and the number

Default: `5pt`

title `content`

The display name of the axis

Default: `[]`

Plots

plot

The constructor function for a plot. This combines the `data` with the `axes` you need to display a graph/plot. The exact structure of `axes` and `data` varies from the visual representation you choose. An exact specification of how these have to look will be found there.

Examples

This is how your plot initialisation will look most of the time:

```
let x_axis = axis(…)
let y_axis = axis(…)
let data = (…)
let pl = plot(axes: (x_axis, y_axis), data: data)
```

How your plot initialisation would look for a *pie chart*:

```
let data = (…)
let pl = plot(data: data)
```

This is a lot simpler ans a *pie chart* doesn't require any axes.

Parameters

```
plot(
  axes: axis,
  data: array
)
```

axes `axis`

A list of axes needed for drawing the plot (most likely a x- and y-axis)

Default: `()`

data `array`

The data that should be mapped onto the plot. The format depends on the plot type

Default: `()`

overlay

This function is used to overlay multiple plots. This can be used to render multiple graph lines in one plot and much more. The axes that get rendered, are the axes of the first plot inserted. Make sure all plots use the same axes as otherwise this will cause issues.

Parameters

```
overlay(
  plots: array,
  size: length array
)
```

plots `array`

An array of all the `plot` objects you want to render.

size `length` or `array`

The size as array of `(width, height)` or as a single value for both `width` and `height`

scatter_plot

This function will display a scatter plot based on the provided `plot` object.

How to create a simple scatter plot

First, we need to define the data we want to map to the scatter plot. In this case I will use some random sample data.

```
let data = ((0, 0), (1, 2), (2, 4), (3, 6), (4, 8), (5, 3), (6, 6),(7, 9),(11, 12))
```

Next, we need to define both the x and the y-axis. The x-axis location can either be `"bottom"` or `"top"`. The y-axis location can either be `"left"` or `"right"`. You can customise the look of the axes with `axis` specific parameters (here: `helper_lines: true`)

```
let x_axis = axis(min: 0, max: 11, step: 1, location: "bottom")
let y_axis = axis(min: 0, max: 13, step: 2, location: "left", helper_lines: true)
```

Now we need to create a `plot` object based on the axes and the data.

```
let pl = plot(axes: (x_axis, y_axis), data: data)
```

Last, we need to just call this function. In this case the width of the plot will be `100%` and the height will be `33%`.

```
scatter_plot(pl, (100%, 33%))
```

Parameters

```
scatter_plot(
  plot: plot,
  size: length array,
  caption: content,
  stroke: color,
  fill: color,
  render_axes: boolean
)
```

plot `plot`

The format of the plot variables are as follows:

- `axes`: Two axes are required. The first one as the x-axis, the second as the y-axis.
  Example: `(x_axis, y_axis)`
- `data`: An array of `x` and `y` pairs.
  Example: `((0, 0), (1, 2), (2, 4), …)`

size `length` or `array`

The size as array of `(width, height)` or as a single value for both `width` and `height`

caption `content`

The name of the figure

Default: `[Scatter Plot]`

stroke `color`

The stroke color of the dots

Default: `black`

fill `color`

The fill color of the dots

Default: `none`

render_axes `boolean`

If the axes should be visible or not

Default: `true`

graph_plot

This function will display a graph plot based on the provided `plot` object. It functions like the *scatter plot* but connects the dots with lines.

How to create a simple graph plot

First, we need to define the data we want to map to the graph plot. In this case I will use some random sample data.

```
let data = ((0, 0), (1, 2), (2, 4), (3, 6), (4, 8), (5, 3), (6, 6),(7, 9),(11, 12))
```

Next, we need to define both the x and the y-axis. The x-axis location can either be `"bottom"` or `"top"`. The y-axis location can either be `"left"` or `"right"`. You can customise the look of the axes with `axis` specific parameters (here: `helper_lines: true`)

```
let x_axis = axis(min: 0, max: 11, step: 1, location: "bottom")
let y_axis = axis(min: 0, max: 13, step: 2, location: "left", helper_lines: true)
```

Now we need to create a `plot` object based on the axes and the data.

```
let pl = plot(axes: (x_axis, y_axis), data: data)
```

Last, we need to just call this function. In this case the width of the plot will be `100%` and the height will be `33%`.

```
graph_plot(pl, (100%, 33%))
```

Parameters

```
graph_plot(
  plot: plot,
  size: length array,
  caption: content,
  rounding: ratio,
  stroke: color,
  fill: color,
  render_axes: boolean,
  markings: none string content
)
```

plot `plot`

The format of the plot variables are as follows:

- `axes`: Two axes are required. The first one as the x-axis, the second as the y-axis.
  Example: `(x_axis, y_axis)`
- `data`: An array of `x` and `y` pairs.
  Example: `((0, 0), (1, 2), (2, 4), …)`

size `length` or `array`

The size as array of `(width, height)` or as a single value for both `width` and `height`

caption `content`

The name of the figure

Default: `"Graph Plot"`

rounding `ratio`

The rounding of the graph, 0% means sharp edges, 100% will make it as smooth as possible (Bézier)

Default: `0%`

stroke `color`

The stroke color of the graph

Default: `black`

fill `color`

The fill color for the graph. Can be used to display the area beneath the graph.

Default: `none`

render_axes `boolean`

If the axes should be visible or not

Default: `true`

markings `none` or `string` or `content`

how the data points should be shown: “square”, “circle”, “cross”, otherwise manually specify any shape

Default: `"square"`

histogram

This function will display a histogram based on the provided `plot` object.

How to create a simple histogram

First, we need to define the data and the classes we want to map to the graph plot. In this case I will use some random sample data.
The tricky part about this is, that this data gets represented in `classes`. These are necessary to combine the data the right way, so the bars height can be displayed correctly.
Here, I will use the same class size every time but once.

Let’s create the data now:

```
let data = (
    18000, 18000, 18000, 18000, 18000, 18000, 18000, 18000, 18000, 18000,
    28000, 28000, 28000, 28000, 28000, 28000, 28000, 28000, 28000, 28000, 28000, 28000,
28000, 28000, 28000, 28000, 28000, 28000, 28000, 28000, 28000, 28000,
   35000, 46000, 75000, 95000
)
```

Now, we will define the classes. To do this we can use the `class_generator(start, end, amount)` and the `class(lower_lim, upper_lim)` function (*see `classify.typ`*)

```
let classes = class_generator(10000, 50000, 4)
classes.push(class(50000, 100000))
classes = classify(data, classes)
```

This will result in creating the following classes: `(10000 - 20000, 20000 - 30000, 30000 - 40000, 40000 - 50000, 50000 - 100000)`.

Next, we need to define both the x and the y-axis. The x-axis location can either be `"bottom"` or `"top"`. The y-axis location can either be `"left"` or `"right"`. You can customise the look of the axes with `axis` specific parameters (here: `show_markings: true` and `helper_lines: true`)

```
let x_axis = axis(min: 0, max: 100000, step: 20000, location: "bottom", show_markings:
false)
let y_axis = axis(min: 0, max: 26, step: 3, location: "left", helper_lines: true)
```

Now we need to create a `plot` object based on the axes and the data.

```
let pl = plot(axes: (x_axis, y_axis), data: data)
```

Last, we just need to call this function. Here we render the histogram with a black outline around the bars, and a gray filling of the bars.

```
histogram(pl, (100%, 20%), stroke: black, fill: gray)
```

Parameters

```
histogram(
  plot: plot,
  size: length array,
  caption: content,
  stroke: color array,
  fill: color array,
  render_axes: boolean
)
```

plot `plot`

The format of the plot variables are as follows:

- `axes`: Two axes are required. The first one as the x-axis, the second as the y-axis.
  Example: `(x_axis, y_axis)`
- `data`: An array of `x` and `y` pairs.
  Example: `((0, 0), (1, 2), (2, 4), …)`

size `length` or `array`

The size as array of `(width, height)` or as a single value for both `width` and `height`

caption `content`

The name of the figure

Default: `[Histogram]`

stroke `color` or `array`

The stroke color of a bar or an `array` of colors, where every entry stands for the stroke color of one bar

Default: `black`

fill `color` or `array`

The fill color of a bar or an `array` of colors, where every entry stands for the fill color of one bar

Default: `gray`

render_axes `boolean`

If the axes should be visible or not

Default: `true`

pie_chart

This function will display a pie chart based on the provided `plot` object.

How to create a simple pie chart

This is the easiest diagram to create. First we need to specify the data. I will use random data here.

```
let data = ((10, "Male"), (20, "Female"), (15, "Divers"), (2, "Other"))
```

Because no axes are required, we can skip this step and jump straight to creating the `plot`.

```
let p = plot(data: data)
```

Last, we just need to call this function. I will call it with all styles available.

```
pie_chart(p, (100%, 20%), display_style: "legend-inside-chart")
pie_chart(p, (100%, 20%), display_style: "hor-chart-legend")
pie_chart(p, (100%, 20%), display_style: "hor-legend-chart")
pie_chart(p, (100%, 20%), display_style: "vert-chart-legend")
pie_chart(p, (100%, 20%), display_style: "vert-legend-chart")
```

Parameters

```
pie_chart(
  plot: plot,
  size: length array,
  caption: content,
  display_style: string,
  colors: array,
  offset: length
)
```

plot `plot`

The format of the plot variables are as follows:
- `axes`: No axes are required.
- `data`: An array of single values or an array of `(amount, value)` tuples.
  Example: `((10, "Male"), (5, "Female"), (2, "Divers"), …)` or `("Male", "Male", "Male", "Female", "Female", "Divers", "Divers", …)`

size `length` or `array`

The size as array of `(width, height)` or as a single value for both `width` and `height`

caption `content`

The name of the figure

Default: `[Pie chart]`

display_style `string`

Changes the style of the pie chart. Available are: `"vert-chart-legend"`, `"hor-chart-legend"`, `"vert-legend-chart"`, `"hor-legend-chart"`, `"legend-inside-chart"`.

Default: `"hor-chart-legend"`

colors `array`

The colors used in the pie chart. If not enough colors were specified, the colors get repeated.

Default: `(red, blue, green, yellow, purple, orange)`

offset `length`

The distance from the center to the text in the pie chart (only relevant when using `"legend-inside-chart"`)

Default: `50%`

bar_chart

This function will display a bar chart based on the provided `plot` object.

How to create a simple bar chart

First we need to specify the data, we want to display. I will use some random data here.

```
let data = ((20, 2), (30, 3), (16, 4), (40, 6), (5, 7))
```

Next we need to create the axes. Keep in mind that, if you want to make the bars go from left to right, not bottom to top, you need to basically invert the x and y-axis creation. You can also customise the axes (here: `show_markings: true` and `helper_lines: true`).

```
let x_axis = axis(min: 0, max: 9, step: 1, location: "bottom")
let y_axis = axis(min: 0, max: 41, step: 10, location: "left", show_markings: true,
helper_lines: true)
```

When `rotated: true`, in other words the bars grow from left to right, the axis creation looks like this:

```
let x_axis = axis(min: 0, max: 41, step: 10, location: "bottom", show_markings: true,
helper_lines: true)
let y_axis = axis(min: 0, max: 9, step: 1, location: "left")
```

Now we need to create the `plot` object.

```
let pl = plot(axes: (x_axis, y_axis), data: data)
```

Last, we just call this function to display the chart. We specify fill colors for every single bar to make it easier to differenciate and we make the bars 30% smaller to create small gaps between bars close to each other.

```
bar_chart(pl, (100%, 120pt), fill: (purple, blue, red, green, yellow), bar_width: 70%)
```

Parameters

```
bar_chart(
  plot: plot,
  size: length array,
  caption: content,
  stroke: color array,
  fill: color array,
  centered_bars: boolean,
  bar_width: ratio,
  rotated: boolean,
  render_axes: boolean
)
```

plot `plot`

The format of the plot variables are as follows:

- `axes:` Two axes are required. The first one as the x-axis, the second as the y-axis.
  Example: `(x_axis, y_axis)`
- `data:` An array of single values or an array of `(amount, value)` tuples.
  Example: `((10, "Male"), (5, "Female"), (2, "Divers"), …)` or `("Male", "Male", "Male", "Female", "Female", "Divers", "Divers", …)`

size `length` or `array`

The size as array of `(width, height)` or as a single value for both `width` and `height`

caption `content`

The name of the figure

Default: `"Barchart"`

stroke `color` or `array`

The stroke color of a bar or an `array` of colors, where every entry stands for the stroke color of one bar

Default: `black`

fill `color` or `array`

The fill color of a bar or an `array` of colors, where every entry stands for the fill color of one bar

Default: `gray`

centered_bars `boolean`

If the bars should be on the number its corresponding to

Default: `true`

bar_width `ratio`

how thick the bars should be in percent. (default: 100%)

Default: `100%`

rotated `boolean`

If the bars should grow on the `x_axis` - this means the data gets mapped to the `y-axis`. Don't forget to create the axes accordingly.

Default: `false`

render_axes `boolean`

If the axes should be visible or not

Default: `true`

Classification

compare

This function is used to compare the data in the classifying process. In most cases you can leave it be.
If you want a different ordinality, you can overwrite this function.

Return specification

- -1 if `val1 < val2`
- 1 if `val1 > val2`
- 0 if `val1 == val2`

Parameters

```
compare(
  val1,
  val2
)
```

class

This is the constructor function for a single `class` used to classify data.
Right now, this is only used for `histograms`.

Parameters

```
class(
  lower_lim: integer,
  upper_lim: integer
)
```

lower_lim `integer`

The lower limit of the class. (Inclusive)

upper_lim `integer`

The upper limit of the class. (Exclusive)

class_generator

Generates a number of classes similarly how `axis` fills the `values` parameter on its own. It splits the area from `start` to `end` into the with `amount` specified amount of classes.
Right now, this is only used for `historams`.
Example:

```
let classes = class_generator(10000, 50000, 4)
```

This will result in creating the following classes: `(10000 - 20000, 20000 - 30000, 30000 - 40000, 40000 - 50000, 50000 - 100000)`.

Parameters

```
class_generator(
  start: integer,
  end: integer,
  amount: integer
)
```

start `integer`

The lower limit of the first generated class.

end `integer`

The upper limit of the last generated class.

amount `integer`

How many classes should be generated.

classify

Classifies the provided data into the given classes. This has to be done to create a `histogram`.

Parameters

```
classify(
  data: array,
  classes: array,
  compare: function
)
```

data `array`

The data you want to classify (needs to be comparable by the compare function). It's either an `array` of single values or an `array` of `tuples` looking like this: `(amount, value)`.

classes `array`

An array of classes the data should be mapped to (`lower_limit` and `uper_limit` need to be comparable).

compare `function`

The method used for comparing. Most of the time this doesn't need to be changed. If you want to use a different compare function, look at the specification for it (*see:* `compare(val1, val2)`).

Default: `compare`