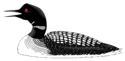
Introduction to loon

An extendible toolkit for exploratory visualization and data analysis

. . . dive in . . . the water's warm . . .

Wayne Oldford
University of Waterloo

based on joint work with
Adrian Waddell
Roche



Preliminaries

Follow installation instructions at

math.uwaterloo.ca/~rwoldfor/talks/loonTutorial/

These include the latest R and RStudio

- R from cran.r-project.org (loon requires at least R 3.4), and
- RStudio from rstudio.com/products/rstudio/download/

as well as loon from the github repository:

```
devtools::install_github("rwoldford/loon", subdir = "R")
```

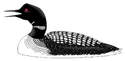
Other packages used in this tutorial:

```
install.packages(c("maps", "sp", "rworldmap", "RColorBrewer", "scales",
                   "ElemStatLearn", "zenplots", "RnavGraphImageData",
                   "gridExtra", "dimRed", "vegan", "MASS" ) )

source("https://bioconductor.org/biocLite.R")
biocLite(c('graph', 'RDRToolbox', 'Rgraphviz'),
         suppressUpdates=TRUE, suppressAutoUpdate=TRUE)
```

Preliminaries

A few packages **might** need special care. These include

- `PairViz` (`install.packages("PairViz")`) depends on some Bioconductor packages.
- `scagnostics` (`install.packages("scagnostics")`) requires `rJava`
- `dplyr` (`install.packages("dplyr")`) and `magrittr` (`install.packages("magrittr")`) will shadow %>% from one another



Data analysis

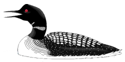
Data analysis is grounded in our understanding of context and models but is also an exploratory, open-ended, and imaginative activity.

- models are necessary but not sufficient and must never be taken too seriously
- look for interesting relationships, structure, patterns, and anomalies
- familiarity is comforting but surprise can force re-thinking
- new questions can be more important than new answers
- visualization can expose features unanticipated by models or experience
- the goal is always greater understanding

"Exposure, the effective laying open of the data to display the unanticipated, is to us a major portion of data analysis." . . . John Tukey and Martin Wilk (1966)

To be effective, exploratory visualization tools need to be simple, natural, flexible, and extendible. And . . .

. . . be integrated into a statistical analysis ecosystem like R.



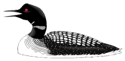
Graphics in R

Core systems:

graphics

Core graphics package in R

- simple syntax and plot model
- large suite of standard statistical plots
- well integrated into R (e.g. `plot()` is a generic function)
- easy to prototype new (or produce ad hoc) statistical graphics
- simple model for layout



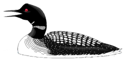
Graphics in R

Core systems:

`graphics` **`grid`**

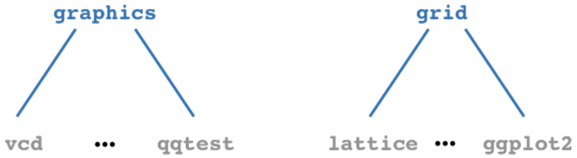
Core `grid` package in R

- classic computer graphics model (e.g. viewports, coordinate systems, `gTree`)
- adapted to data display (e.g. `dataViewport`, `points`)
- provides rich data structures that can be saved and changed
- rich, essentially unlimited, prototyping and layouts are possible
- powerful, robust, and open-ended model for graphics system developers



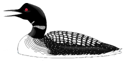
Graphics in R

Core systems:



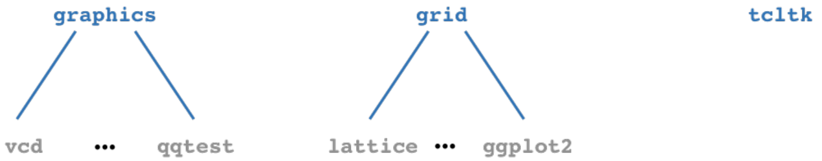
Building on core graphics or grid

- on graphics: special purpose visualizations
  - vcd ... visualizing categorical data
  - qqtest ... building in plot uncertainty
- on grid: whole visualization systems
  - lattice ... an implementation of Cleveland's "trellis" model
  - ggplot2 ... an implementation of Wilkinson's "grammar" model



Graphics in R

Core systems:

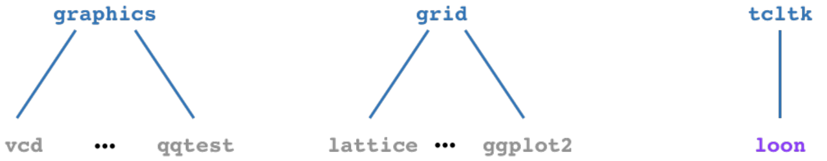


Core `tcltk` package gives a core GUI builder for R

- accesses a full programming language (`tcl`)
- Tk windowing tool kit (written in C)
- Tk allows you to create, manage, and manipulate `widgets`
    - a visual or layout component of a GUI
    - e.g. `buttons`, `scrollbars`, etc.
    - layout (geometry) management

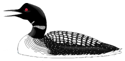
Graphics in R

Core systems:



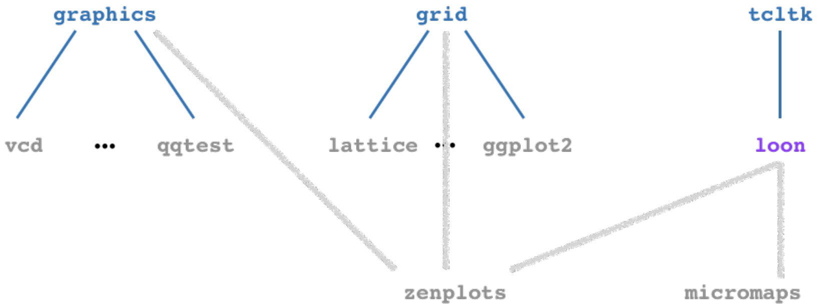
loon completes tcltk to provide a **direct manipulation** graphics system for R

- modular object-oriented design
  - e.g. separation of model, view, and controller
  - e.g. layered graphics
- integrated with R
  - R data structures snf procedures
  - programmatic interaction with displays
- built (almost) entirely in tcl/tk



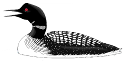
Graphics in R

Core systems:



interactive graphics can be built on loon

- ad hoc interactive graphics (e.g. teaching demonstration)
- new interactive graphics (e.g. time series decompositions)
- new interactive packages (e.g. `micromaps`)
- packages can offer choice of graphics systems (e.g. `zenplots`)



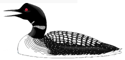
loon - dive beneath the data surface to explore its depth

Once installed:

```r
library(loon)
```

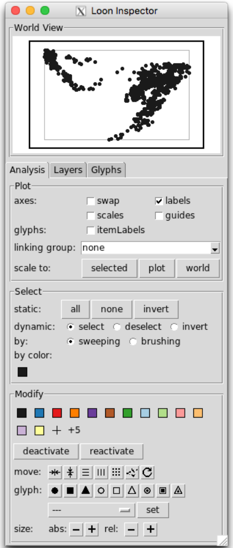
and create your first `loon` plot

```r
l_plot(x = quakes$long, y = quakes$lat,
       xlabel = "longitude", ylabel = "latitude",
       title = "Tonga trench earthquakes")
```

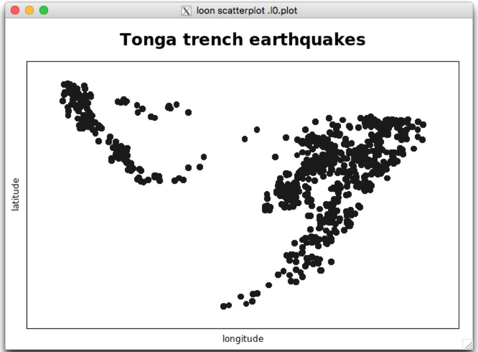


```
## [1] ".l0.plot"
## attr(,"class")
## [1] "l_plot" "loon"
```

loon - scatterplot and inspector



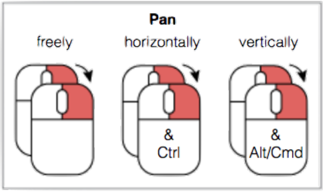
axes: [✓ labels]
linking group: none
dynamic: [✓ select] deselect invert
by: [✓ sweeping] brushing



The loon inspector is omnipresent with loon plots.

Direct manipulation – the scatterplot

Panning



Move the scatterplot

- freely about
- only horizontally
- only vertically

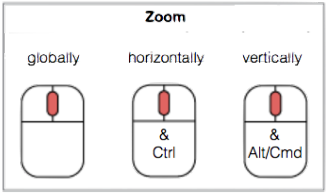
Did the inspector change in response to each movement? How?

Reproduce these movements on the inspector itself.

Does the scatterplot change?

Direct manipulation – the scatterplot

Zooming



There is a small cluster of 16 points towards the top left of the scatterplot.

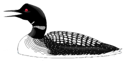
- On the scatterplot, zoom in and out and move the plot around until **only** these 16 points appear in the scatterplot
- Still on the scatterplot, use the restricted horizontal and vertical zooming so that the same 16 points occupy as much of the scatterplot's space as possible

How has the inspector changed in response?

On the inspector click on the button titled `plot`. What happened?

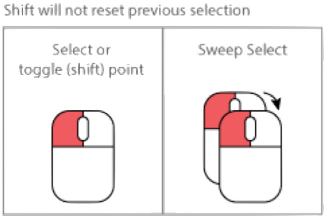
Using zooming and panning on the `World View`

- focus the scatterplot on the **same** 16 points.



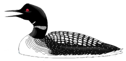
Direct manipulation – the scatterplot

selecting


Shift will not reset previous selection
Select or toggle (shift) point
Sweep Select

On the scatterplot (which should contain only 16 points)

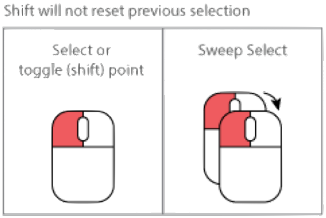
- place the mouse over the topmost point and click the left (or primary) mouse button
  - the point will be highlighted (magenta coloured)
- hold the shift key down and select the leftmost, rightmost and bottom most points
  - there should now be 4 magenta points
- colour these 4 points light blue and change their glyph to a triangle
  - click on the background of the plot to see the change in colour



How has the inspector changed?

Direct manipulation – the scatterplot

selection by sweeping

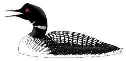

Shift will not reset previous selection
Select or toggle (shift) point
Sweep Select

On the scatterplot left-click and holding the mouse button down move the mouse. This is called **sweeping** or **sweep selection** and allows us to sweep out a contiguous area of the plot.

- use sweep select to identify, and give different colours (none black) to 4 different contiguous groups of your choice.

The 16 points should now be separated by colour into 4 groups and by glyph shape into 2 groups

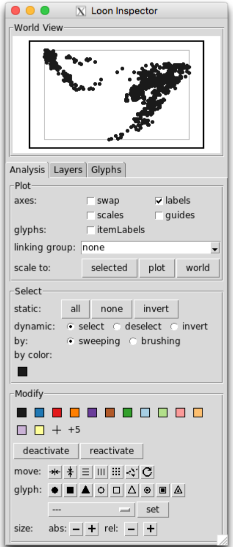
How has the inspector changed?



- **from the inspector**, select all the points of each colour in turn

The loon inspector - interacting with the plot

The inspector is used to change the loon plot. For example, checking the “guides” box places a background grid of guide lines on the loon plot.



Loon Inspector

World View

Analysis | Layers | Glyphs

Plot

axes: swap [✓ labels] scales guides

glyphs: itemLabels

linking group: none

scale to: selected plot world

Select

static: all none invert

dynamic: [✓ select] deselect invert

by: [✓ sweeping] brushing

by color:

Modify

+ +5

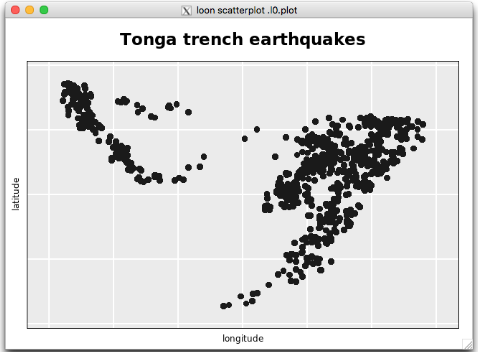
deactivate reactivate

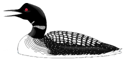
move:

glyph:

--- set

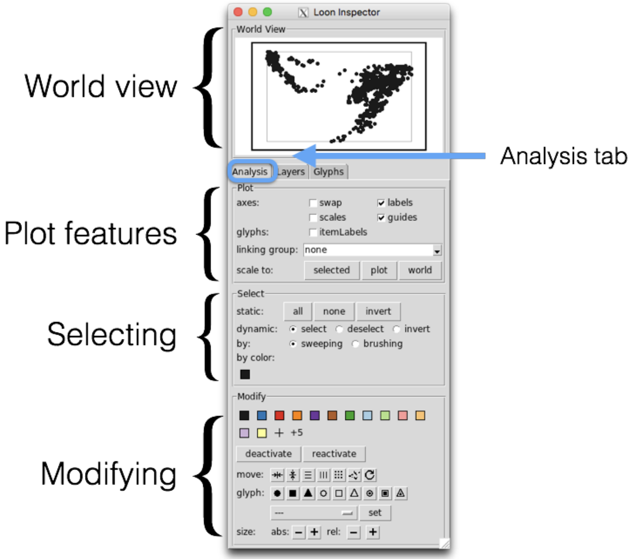
size: abs: – + rel: – +





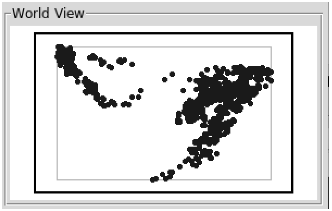
The loon inspector - components

The loon inspector separates into different panels on the analysis tab:



axes: [ ] swap [✓ labels] [ ] scales [✓ guides]
glyphs: [ ] itemLabels
linking group: none
scale to: selected plot world
static: all none invert
dynamic: [✓ select] deselect invert
by: [✓ sweeping] brushing
by color: ■
move:
glyph: ● ■ ▲ ○ □ △
size: abs: – + rel: – +

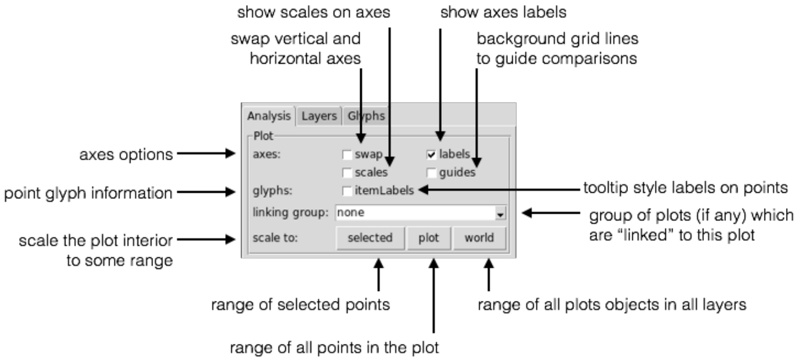
The loon inspector - the World View



The world view always shows the **whole** of the current plot, what is displayed and what is active.

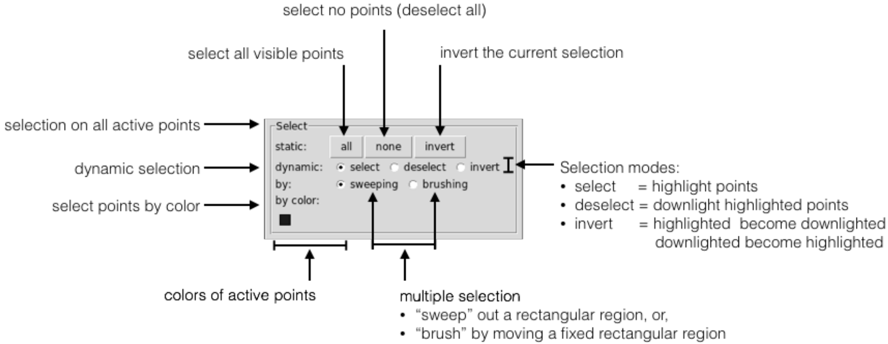
The loon inspector - the Analysis tab

Plot features:


show scales on axes
show axes labels
swap vertical and horizontal axes
background grid lines to guide comparisons
Analysis
Layers
Glyphs
Plot
axes options
axes:
swap
labels
scales
guides
point glyph information
glyphs:
itemLabels
tooltip style labels on points
linking group:
none
group of plots (if any) which are "linked" to this plot
scale the plot interior to some range
scale to:
selected
plot
world
range of selected points
range of all plots objects in all layers
range of all points in the plot

The loon inspector - the Analysis tab

Point selection:


select no points (deselect all)
select all visible points
invert the current selection
selection on all active points
dynamic selection
select points by color
colors of active points
Select
all
none
invert
by color:

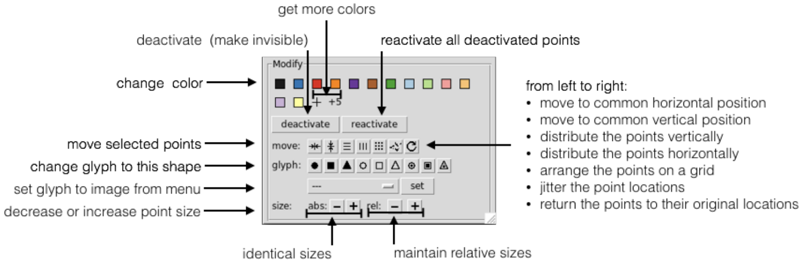
static: all none invert
dynamic: [✓ select] deselect invert
by: [✓ sweeping] brushing

Selection modes:
- select = highlight points
- deselect = downlight highlighted points
- invert = highlighted become downlighted downlighted become highlighted

multiple selection
- “sweep” out a rectangular region, or,
- “brush” by moving a fixed rectangular region

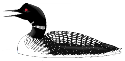
The loon inspector - the Analysis tab

Modifying selected points:


get more colors
deactivate (make invisible)
reactivate all deactivated points
change color
Modify
+5
deactivate
reactivate
move selected points
move:
change glyph to this shape
glyph:
set glyph to image from menu
---
set
decrease or increase point size
size:
abs:
rel:
identical sizes
maintain relative sizes

from left to right:

- move to common horizontal position
- move to common vertical position
- distribute the points vertically
- distribute the points horizontally
- arrange the points on a grid
- jitter the point locations
- return the points to their original locations

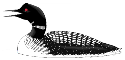


The loon inspector - interacting with the plot

Use the functionality of the loon inspector to (in order)

1. select the black points only
2. scale the plot to the selected points
3. invert the selection
4. colour the selected points red
5. select all points, and make them all have open circles as glyphs
6. select the red points and deactivate them
7. select brushing mode
8. brush select the remaining black points dividing them into two groups
9. colour one of these groups blue, the other orange
10. turn off the brush
11. re-activate the points
12. scale the scatterplot to the "world"
13. add guides (**no** scales)

You should have a scatterplot of open circles with three clusters of points in red, blue, and orange



Direct manipulation – more on brushing

You should have a scatterplot of open circles with three clusters of points in red, blue, and orange.

Use the functionality of the loon inspector to (in order)

1. Using only colour selection from the inspector, select the largest group (and no others)
2. Turn on brushing (as opposed to sweeping) mode
3. Under **dynamic** selection, choose **deselect**
4. Over the scatterplot use this brush mode to **deselect** the region nearest the highest density (shaped a bit like a question mark – ?)
5. Colour the selected points green (i.e. low density region of largest coloured group)

You should now have four different coloured groups or clusters.

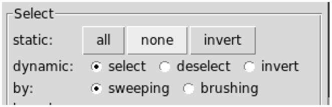
For fun, in the inspector

1. select the green group
2. under **dynamic** selection, choose **invert**
3. move the brush over the scatterplot
4. move the brush over the scatterplot while holding the `<Shift>` key

What is going on? When might this functionality be useful?

Direct manipulation – more on selection

When done, turn the selection back to the defaults:



static: none
dynamic: select
by: sweeping

Shift selection works with any mix of

- individual point selection
- static selection
- dynamic selection by sweeping or brushing
- by color

Try out different combinations of the above selections.

Direct manipulation – moving points

For a variety of reasons, we sometimes want to (temporarily) move points in a scatterplot.

From the inspector:

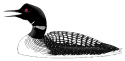
1. deactivate all but the small red group
2. select the red group and scale the plot to the selected
3. with these selected points, explore the effect of each of the `move` modifications on the configuration of the selected points
4. describe in words what each one does

From the scatterplot, selected points may be moved by hand

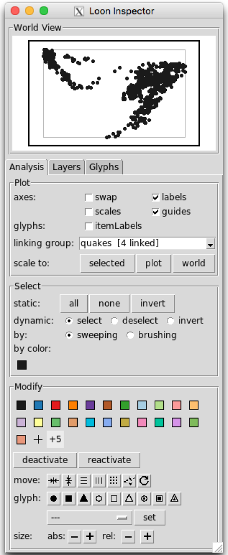
- one at a time using `<Ctrl>-<Left-button>`
- in groups using `<Shift><Ctrl><Left-button>`

For example:

1. select the red group
2. move the red group to the bottom of the plot
3. return the red group to its original location



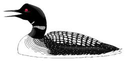
Direct manipulation – adding colours to the inspector



axes: [ ] swap [✓ labels]
[ ] scales [✓ guides]
glyphs: [ ] itemLabels
linking group: quakes [4 linked]
scale to: selected plot world
static: all none invert
dynamic: [✓ select] deselect invert
by: [✓ sweeping] brushing
by color: ■

More colours can be added directly by clicking on the '+' or the '+5' in the colour modification section of the inspector.

- clicking '+5' twice adds 10 new colours
- these colours will persist with the inspector for every loon scatterplot



Direct manipulation – the inspector palette of colours

The loon inspector comes with a predefined palette of colours.

The loon system **default** list of colours get be found programmatically:

```
l_getColorList()
```

```
##  [1] "gray10"  "#1F78B4" "#E31A1C" "#FF7F00" "#6A3D9A" "#B15928" "#33A02C"
##  [8] "#A6CEE3" "#B2DF8A" "#FB9A99" "#FDBF6F" "#CAB2D6" "#FFFF99"
```

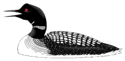
Note:

- loon colours are 6 hexadecimal digits in length and can be a named R colour
- being constrained to tcl/tk colours, no alpha level is available at this time in loon
- R has a similar set of base colours that is default for all its graphics devices:

  ```
  palette()
  ```

  ```
  ## [1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
  ## [8] "gray"
  ```

- like the R pallette, the loon inspector palette persists



Direct manipulation – changing the inspector palette of colours

(**Note** At the time of this writing (Mon Jul 23 12:44:18 2018) changing the inspector colours is recommended **only** immediately after `library(loon)`. Otherwise, there may be unintended side effects to changing the inspector colours after a plot (and hence the inspector) has been created.)

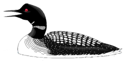
A variety of functions exist to change an inspector's palette of colours:

```
l_setColorList_baseR()  # base R palette
l_setColorList_ColorBrewer("Set2")  # colorblind friendly choice from ColorBrewer
l_setColorList_hcl(luminance = 80)  # set of hcl colours
l_setColorList_ggplot2()  # ggplot2's palette
l_setColorList_loon()  # default loon palette
l_setColorList(l_colRemoveAlpha(rainbow(5)))  # any set of colours without alpha
```

To change the palette

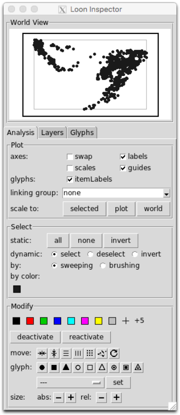
- execute one of these functions
- select the inspector
- close the inspector to refresh the colours

`l_setColorList_loon()` returns the inspector to the default loon colour palette.

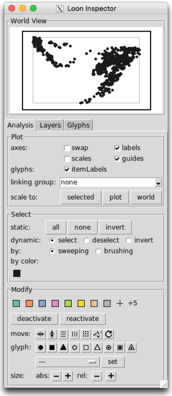


Direct manipulation – example palettes

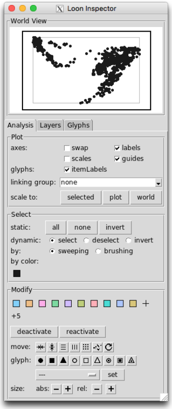
base R



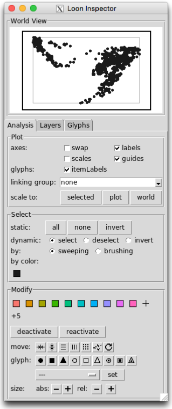
ColorBrewer



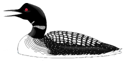
hcl



ggplot2



Again, `l_setColorList_loon()` returns the inspector to the default loon colour palette.



Programmatic manipulation - accessing the plot

The plot itself is a (tcl) data structure and can be assigned to an R variable.

Either at construction (the preferable way):

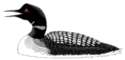
```
# loon graphics (note that the result is assigned to p)
p <- l_plot(x = quakes$long, y = quakes$lat,
            xlabel = "longitude", ylabel = "latitude",
            title = "Tonga trench earthquakes")
```

or, if (as we did) you forgot to do that or reassigned something else to that variable (more likely), by accessing the plot from its `tcl` string:

```
# accessing the plot from its string representation
p <- l_create_handle(".l0.plot")
```

Note that this string `".l0.plot"` appears on the right side of the window title of an `l_plot`.

Do this and assign your plot to `p`. You might have to look at the window to get the string if you have created more than one `l_plot()`.



Programmatic manipulation - printing the plot

Were we to now print p

```
p
```

```
## [1] ".l0.plot"
## attr(,"class")
## [1] "l_plot" "loon"
```

we get the string name of the plot, but with a class attribute. This is the **value** of p and provides us access to a very rich data structure, but one whose values reside in tcl.

For example, we can access this value and, like plots from lattice or ggplot2 produce a grid data structure that can then be displayed as a grid graphic in R.

```
library(grid)
grid.newpage()
grid.loon(p)
```

will display p as a grid graphic ... and all that entails.

Try this. Change p by direct manipulation in loon and repeat.

This allows the visual representation of a loon plot to be displayed at any time.

Connecting to grid graphics - `grid.loon()` and `loonGrob()`

Graphics in the `grid` package are built up from graphical objects or `grobs`.

In `loon`, the functions `grid.loon()` and `loonGrob()` construct `grobs` from loon plots which can then be used as any other `grob` in `grid`

For example, when transformed to `grid`, the loon plot `p` structure becomes a `gTree` in `grid`

```
library(grid)
gp <- grid.loon(p, draw = FALSE)  # produces a grob
grid.ls(gp)  # lists the contents of the grob
```

```
## GRID.gTree.2
##   l_plot
##     loon plot
##       x label
##       y label
##       title
##       bbox
##       clip
##       l_plot_layers
##         l_layer_group: root
##           GRID.points.1
##       GRID.polyline.3
```

As such, a collection of `grobs` taken from `p` at different times could be put together in a single display using tools from `grid` like `gTree()` and `grid.arrange()`.

Programmatic manipulation - getting plot contents

There are `names` associated with a loon plot indicating its contents or **states**:

```
names(p)
```

```
##  [1] "glyph"             "itemLabel"         "showItemLabels"
##  [4] "linkingGroup"      "linkingKey"        "zoomX"
##  [7] "zoomY"             "panX"              "panY"
## [10] "deltaX"            "deltaY"            "xlabel"
## [13] "ylabel"            "title"             "showLabels"
## [16] "showScales"        "swapAxes"          "showGuides"
## [19] "background"        "foreground"        "guidesBackground"
## [22] "guidelines"        "minimumMargins"    "labelMargins"
## [25] "scalesMargins"     "x"                 "y"
## [28] "xTemp"             "yTemp"             "color"
## [31] "selected"          "active"            "size"
## [34] "tag"               "useLoonInspector" "selectBy"
## [37] "selectionLogic"
```

These features of `p` can be directly accessed the `[]` function.

For example,

```
p["showGuides"]
```

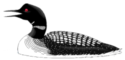
```
## [1] FALSE
```

Programmatic manipulation - getting plot states

A more meaningful example might be the following

```
getGroups <- function(loonplot){

    if (!"l_plot" %in% class(loonplot)) stop("loonplot must be an l_plot")

    lapply(unique(loonplot['color']),
           FUN = function(group){
               loonplot['color'] == group
               }
           )
}

myGroups <- getGroups(p)  # returns groups identified by unique colour in p
nGroups <- length(myGroups) # number of groups
head(quakes[myGroups[[1]],]) # Data on first few quakes in group 1.
```

```
##      lat   long depth mag stations
## 1 -20.42 181.62   562 4.8       41
## 2 -20.62 181.03   650 4.2       15
## 3 -26.00 184.10    42 5.4       43
## 4 -17.97 181.66   626 4.1       19
## 5 -20.42 181.96   649 4.0       11
## 6 -19.68 184.31   195 4.0       12
```



Programmatic manipulation - setting plot contents

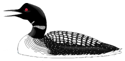
In addition to accessing the state values of a loon plot using [], its values may also be set using []. For example, try these:

```r
p["showGuides"] <- TRUE

p["size"] <- sample(1:30, size = length(p["x"]), replace = TRUE)

for (i in 1:length(myGroups))  {
    p["selected"] <- myGroups[[i]]
    Sys.sleep(1)
}
p["selected"] <- FALSE

# something a little more involved for up to 6 groups
myCols <- c("firebrick", "steelblue", "purple",
            "orange", "grey10", "grey80")
for (i in 1:length(myGroups)) {
    p["color"][myGroups[[i]]] <- myCols[i]
}

# something crazy
for (j in 1:10) {
  p["xTemp"] <- p["x"] + runif(length(p["x"]), min = -0.5, max = 0.5)
  Sys.sleep(0.1)
}

# putting locations and size back
p["xTemp"] <- p["x"]
p["size"] <- 4
```

Programmatic manipulation - setting plot contents and grid

Being able to get and set the plot contents means that it is possible to capture different plot states as grobs. Try the following:

```
library(gridExtra)
gs  <- lapply(myGroups,
              FUN = function(group) {
                  p["selected"] <- group
                  loonGrob(p)
              })
grid.arrange(grobs = gs, ncol = 2)
p["selected"] <- FALSE
```



Programmatic manipulation - adding layers

`loon` plots are actually a little richer still. The scatterplot is only one possible layer in a plot. It could have many more, each layer having different geometric objects.

For example, we can add a map to the current loon plot `p`.

First get the relevant map:

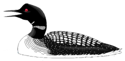
```r
library(maps)
NZFijiMap <- map("world2", regions = c("New Zealand", "Fiji"), plot = FALSE)
```

It is added as a “layer” to the loon plot

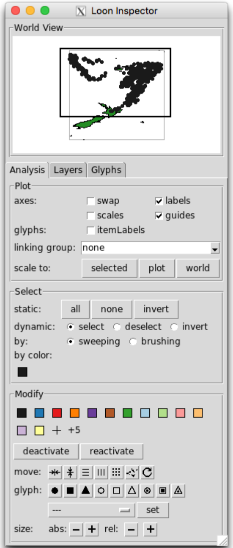
```r
l_layer(p, NZFijiMap,
        label = "New Zealand and Fiji",
        color = "forestgreen",
        index = "end")
```

```
## loon layer "New Zealand and Fiji" of type polygons of plot .l0.plot
## [1] "layer0"
```

Try this out. Now play with the scaling buttons in the loon inspector.



Adding layers - maps


Loon Inspector

World View

Analysis | Layers | Glyphs

Plot

axes: [ ] swap [✓ labels] [ ] scales [✓ guides]

glyphs: [ ] itemLabels

linking group: none

scale to: selected | plot | world

Select

static: all | none | invert

dynamic: [✓ select] deselect invert

by: [✓ sweeping] brushing

by color: ■

Modify

■ 🟦 🟥 🟧 🟪 🟫 🟩 ■ ■ ■ ■ ■ ■ + +5

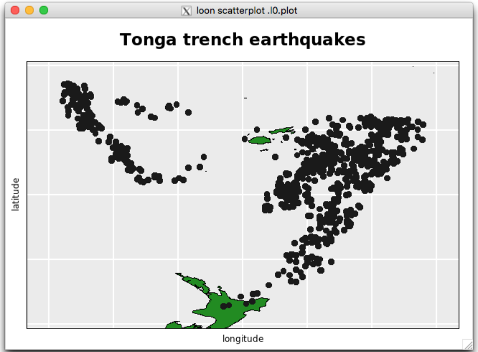
deactivate | reactivate

move:

glyph: ● ■ ▲ ○ □ △

--- set

size: abs: – + rel: – +

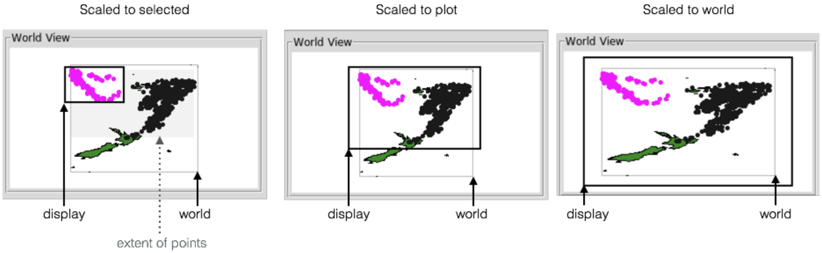


As can be seen in the world view, much of the map is outside of the plot display (shown as the black-bordered rectangle in the world view).

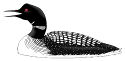
Analysis tab - scaling choices

Adding the map allows us to see the effect of all three plot scaling choices available in the inspector.

The effect is best seen on the world view:

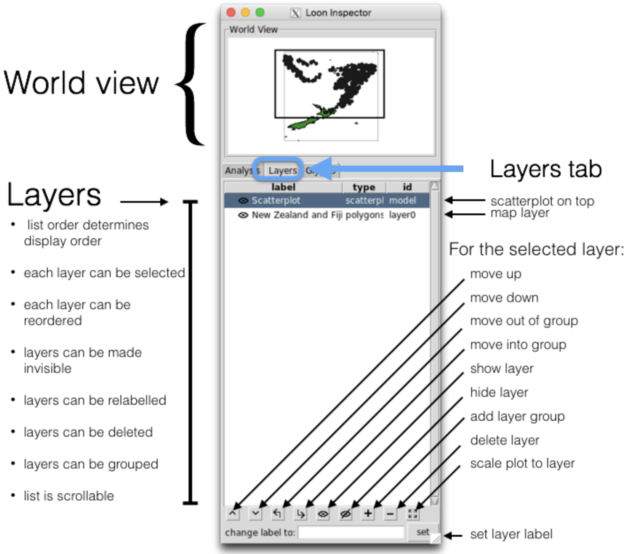


Note that the plot in the world view matches that of the actual plot. Also the aspect ratio changes with the scaling.

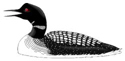


Loon inspector - Layers tab

The map is added as a layer, which can be seen by selecting the "Layers" tab in the inspector:



label	type	id
Scatterplot	scatterpl	model
New Zealand and Fiji polygons		layer0



loon plots - where are we?

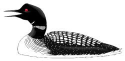
`loon`'s `l_plot()` **syntax resembles** that of `plot()` from base `graphics`:

```
# Base graphics
plot(x = quakes$long, y = quakes$lat,
     xlab = "longitude", ylab = "latitude",
     main = "Tonga trench earthquakes")

# loon graphics
l_plot(x = quakes$long, y = quakes$lat,
       xlabel = "longitude", ylabel = "latitude",
       title = "Tonga trench earthquakes")
```

The arguments to `l_plot()` provide initial values to each named state.

In result, however, loon plots are quite different

- displays **look different**
- `l_plot()` **returns a data structure**; `plot()` does not
  - a rich data structure with many states
  - can be transformed to a grob
- the `loon` plot is **highly interactive**
  - either from the inspector
  - or programmatically



But wait, there's more.

Histograms in loon - `l_hist()`

As seen earlier, the `quakes` data has some other variates in addition to the latitude and longitude of the quakes.

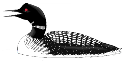
Before starting, make sure that you have the four groups coloured in the loon plot `p` as described earlier. If you don't, then colour them again now.

Let's consider a histogram of the `depth` of each earthquake.

```
h <- l_hist(quakes$depth,
            xlabel = "depth",
            title = "Tonga trench earthquakes")
```

Note that the inspector has changed. There is a different inspector for each type of base loon plot.

- in the plot section of the histogram inspector, select `yshows: density`
- on the histogram, see how individual bars can be selected
- what happens when you modify the colour?
- on the histogram, what is the (interactive) function of the small grey object?
  - how can you make it go away?
- on the histogram inspector, turn on stacked colours
- select colours on the histogram



Linking in loon - `linkingGroup`

- On the histogram inspector, where it says linking group, select the text "none" and replace it with "quakes"
- On the scatterplot inspector, if you do the same you will now be prompted to either "push" or "pull". Select "push".
- brush the scatterplot and observe the effects in the histogram
- brush the histogram and observe the effects in the scatterplot
    - shape the brush so that it is tall and thin
    - brush from left to right and right to left
- set the palette on the histogram inspector as

```
library(RColorBrewer)
cols <- brewer.pal(9, "Blues")[-c(1:4)]  # get rid of lightest
l_setColorList(cols)
```

  and colour the histogram bars so that those earthquakes that are deepest are darkest. Afterwards reset the palette to the loon colours.
- could also set colours programmatically on the histogram

```
h["color"] <- cols[cut(quakes$depth, breaks=5)]
```

Linking in loon - `linkingGroup`

Arbitrarily many plots may be created and linked in loon.

Consider, adding the two more histograms to the same `linkingGroup`

```
h_mag <- l_hist(quakes$mag, linkingGroup = "quakes",
                showStackedColors = TRUE,
                yshows = "density",
                xlabel = "magnitude")
h_stations <- l_hist(quakes$stations, linkingGroup = "quakes",
                     showStackedColors = TRUE,
                     yshows = "density",
                     xlabel = "Number of stations reporting")
```

Brush to show the relationship between magnitude of the earthquake and the number of stations reporting.

Alternatively, we could look at a linked scatterplot of these two variates and explore their relationship conditonal on `depth`

```
p_mag_stations <- l_plot(quakes$mag, quakes$stations,
                         linkingGroup = "quakes",
                         xlabel = "Magnitude",
                         ylabel = "Number of stations reporting")
```