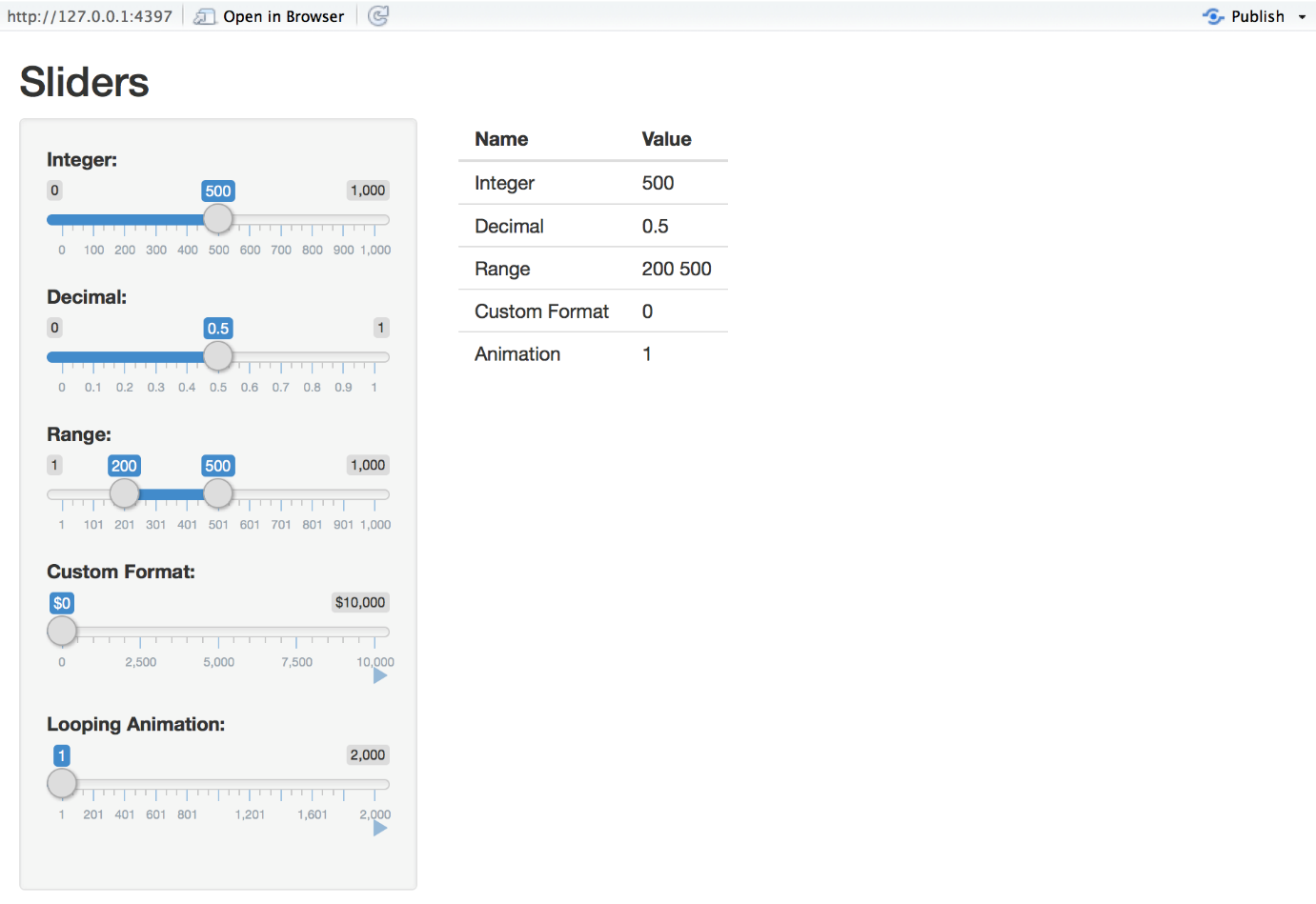
The **intrval** R package is lightweight (~11K), standalone (apart from importing from **graphics**, has exactly 0 non-**base** dependency), and it has a very narrow scope: it implements relational operators for intervals — very well aligned with the *tiny manifesto*. In this post we will explore the use of the package in two [**shiny**](https://shiny.rstudio.com/) apps with sliders.

Code Chunks - Shiny App with Sliders

The [Sliders app](https://shiny.rstudio.com/gallery/sliders.html) demonstrates the many capabilities of slider controls, including the ability to run an animation sequence.



To run the example locally type:

library(shiny)

runExample("05\_sliders")

Customizing Sliders

Shiny slider controls are extremely capable and customizable. Features supported include:

* The ability to input both single values and ranges
* Custom formats for value display (e.g for currency)
* The ability to animate the slider across a range of values

Slider controls are created by calling the sliderInput function. The UI demonstrates using sliders with a variety of options:

ui

*# Define UI for slider demo app ----*

ui <- fluidPage(

*# App title ----*

titlePanel("Sliders"),

*# Sidebar layout with input and output definitions ----*

sidebarLayout(

*# Sidebar to demonstrate various slider options ----*

sidebarPanel(

*# Input: Simple integer interval ----*

sliderInput("integer", "Integer:",

min = 0, max = 1000,

value = 500),

*# Input: Decimal interval with step value ----*

sliderInput("decimal", "Decimal:",

min = 0, max = 1,

value = 0.5, step = 0.1),

*# Input: Specification of range within an interval ----*

sliderInput("range", "Range:",

min = 1, max = 1000,

value = c(200,500)),

*# Input: Custom currency format for with basic animation ----*

sliderInput("format", "Custom Format:",

min = 0, max = 10000,

value = 0, step = 2500,

pre = "$", sep = ",",

animate = TRUE),

*# Input: Animation with custom interval (in ms) ----*

*# to control speed, plus looping*

sliderInput("animation", "Looping Animation:",

min = 1, max = 2000,

value = 1, step = 10,

animate =

animationOptions(interval = 300, loop = TRUE))

),

*# Main panel for displaying outputs ----*

mainPanel(

*# Output: Table summarizing the values entered ----*

tableOutput("values")

)

)

)

Server Script

The server side of the Slider application is very straightforward – it creates a data frame containing all of the input values and then renders it as an HTML table:

server

*# Define server logic for slider examples ----*

server <- **function**(input, output) {

*# Reactive expression to create data frame of all input values ----*

sliderValues <- reactive({

data.frame(

Name = c("Integer",

"Decimal",

"Range",

"Custom Format",

"Animation"),

Value = as.character(c(input$integer,

input$decimal,

paste(input$range, collapse = " "),

input$format,

input$animation)),

stringsAsFactors = FALSE)

})

*# Show the values in an HTML table ----*

output$values <- renderTable({

sliderValues()

})

}

The first example uses a regular slider that returns a single value. To make that an interval, we will use standard deviation (SD, *sigma*) in a quality control chart ([QCC](https://en.wikipedia.org/wiki/Control_chart)). The code is based on the pistonrings data set from the **qcc** package. The Shewhart chart sets 3-\_sigma\_ limit to indicate state of control. The slider is used to adjusts the *sigma* limit and the GIF below plays is as an animation.

library(shiny)

library(intrval)

library(qcc)

data(pistonrings)

mu <- mean(pistonrings$diameter[pistonrings$trial])

SD <- sd(pistonrings$diameter[pistonrings$trial])

x <- pistonrings$diameter[!pistonrings$trial]

## UI function

ui <- fluidPage(

plotOutput("plot"),

sliderInput("x", "x SD:",

min=0, max=5, value=0, step=0.1,

animate=animationOptions(100)

)

)

# Server logic

server <- function(input, output) {

output$plot <- renderPlot({

Main <- paste("Shewhart quality control chart",

"diameter of piston rings", sprintf("+/- %.1f SD", input$x),

sep="\n")

iv <- mu + input$x \* c(-SD, SD)

plot(x, pch = 19, col = x %)(% iv +1, type = "b",

ylim = mu + 5 \* c(-SD, SD), main = Main)

abline(h = mu)

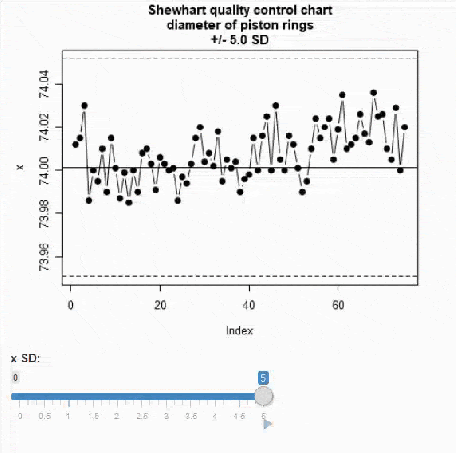
abline(h = iv, lty = 2)

})

}

## Run shiny app

if (interactive()) shinyApp(ui, server)



The second example uses range slider returning two values, which is our interval. To spice things up a bit, we combine intervals on two axes to color some random points. The next range slider defines a distance interval and colors the random points inside the ring.

library(shiny)

library(intrval)

set.seed(1)

n <- 10^4

x <- round(runif(n, -2, 2), 2)

y <- round(runif(n, -2, 2), 2)

d <- round(sqrt(x^2 + y^2), 2)

## UI function

ui <- fluidPage(

titlePanel("intrval example with shiny"),

sidebarLayout(

sidebarPanel(

sliderInput("bb\_x", "x value:",

min=min(x), max=max(x), value=range(x),

step=round(diff(range(x))/20, 1), animate=TRUE

),

sliderInput("bb\_y", "y value:",

min = min(y), max = max(y), value = range(y),

step=round(diff(range(y))/20, 1), animate=TRUE

),

sliderInput("bb\_d", "radial distance:",

min = 0, max = max(d), value = c(0, max(d)/2),

step=round(max(d)/20, 1), animate=TRUE

)

),

mainPanel(

plotOutput("plot")

)

)

)

# Server logic

server <- function(input, output) {

output$plot <- renderPlot({

iv1 <- x %[]% input$bb\_x & y %[]% input$bb\_y

iv2 <- x %[]% input$bb\_y & y %[]% input$bb\_x

iv3 <- d %()% input$bb\_d

op <- par(mfrow=c(1,2))

plot(x, y, pch = 19, cex = 0.25, col = iv1 + iv2 + 3,

main = "Intersecting bounding boxes")

plot(x, y, pch = 19, cex = 0.25, col = iv3 + 1,

main = "Deck the halls:\ndistance range from center")

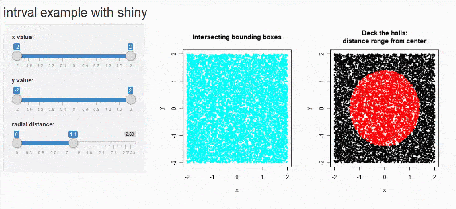
par(op)

})

}

## Run shiny app

if (interactive()) shinyApp(ui, server)



If you think there are other use cases for **intrval** in **shiny** applications, let me know in the comments section!