A Tool for Generating Slider Functions

File: slider-make.rev
in: /home/wiwi/pwolf/R/aplpack/sliderfns

December 1, 2009

1 The definition of `as.slider.function`

In this section we define the function `as.slider.function` which constructs a `slider.*` function on base of a function of the user. The name of the new slider function will be composed of the string `"slider."` and the name of the user's function. If you have written the function `myhist` then you have to pass `myhist` to `as.slider.function` and the new function `slider.myhist` will be created. For defining the sliders the arguments of the input function, e.g. of `myhist`, without the first one are use. The idea behind this construction is the observation that the first argument very often is a data set whereas the other arguments specify kinds of computation or some attributes of an resulting plot. Therefore, for each of all but the first argument a slider will be constructed. After calling `as.slider.function` a new window or widget is opened and the user has to add the main characteristics of the sliders interactively: minima, maxima, steps, and default values. By pressing the button `"make slider function"` the new function is printed and it is created in the global environment. The new function can be used as it is or it can be modified it as needed.

The function `as.slider.function` is separated into three parts. At first the input function is analyzed, then the widget that controls the characteristics of the sliders is constructed, and last not least is the generating function defined.

```
1    ⟨define as.slider.function 1⟩ ≡   ⊂ 2
      as.slider.function<-function(refresh.function){
        ⟨get info of input function 3⟩
        ⟨define widget for setting parameters 4⟩
        ⟨define function for button make slider function 5⟩
      }
```

Now we ready to test our new mechanism.

```
2    ⟨test as.slider.function 2⟩ ≡
      ⟨define as.slider.function 1⟩
      myhist<-function(x,breaks,col=2){
        hist(x,breaks=breaks,col=col)
      }
      as.slider.function(myhist)
```

3 ⟨*get info of input function* 3⟩ ≡ ⊂ 1, 6

```
fname<-deparse(substitute(refresh.function)); new.fname<-paste("slider.",fname,sep="")
formals.refresh.function<-formals(refresh.function)
first.arg<-names(formals.refresh.function)[1]
formals.refresh.function<-formals.refresh.function[-1]
args<-names(formals.refresh.function)
defaults<-unlist(lapply(formals.refresh.function,function(x) as.character(x)[1]))
defaults[defaults==""]<-"0"
```

4 ⟨*define widget for setting parameters* 4⟩ ≡ ⊂ 1, 6

```
## define top level widget
   top<-tktoplevel(); w<-10
   tkwm.title(top,paste('as.slider.function: define characteristics of sliders of "',
                        new.fname,'"',sep=""))
## define input field for slider characteristics
   for(i in seq(along=args)){
     frame<-tkframe(top,width=40); tkpack(frame)
     label<-tklabel(frame,text=args[i],width=15); tkpack(label,side="left")
     label.set<-c("Min","Max","Step","Default"); entry<-NULL
     for(j in seq(along=label.set)){
       label<-tklabel(frame,text=label.set[j],width=w);  tkpack(label,side="left")
       entry<-c(entry,list(tkentry(frame,width=w))); tkpack(entry[[j]],side="left")
       var.name<-paste(label.set[j],i,sep=""); value<-ifelse(j==3,"1",defaults[i])
       set.tclvalue(var.name,value); tkconfigure(entry[[j]],textvariable=var.name)
     }
   }
   frame<-tkframe(top,width=40); tkpack(frame,fill="both",expand="yes")
## define buttons "exit" and "make slider function"
   exit<-tkbutton(frame,text="exit",command=function() tkdestroy(top))
   doit<-tkbutton(frame,text=paste("make",new.fname))
   tkpack(doit, side="left"); tkpack(exit, side="right")
```

5 ⟨*define function for button make slider function* 5⟩ ≡ ⊂ 1

```
  make.slider.function<-function(environ=sys.parent()){
    # header of new slider function
    new.fun<-paste(new.fname,sep="",
                   "<-function(",first.arg,",",paste(args,collapse=","),"){")
    # header of redo function
    redo.name<-paste("redo",fname,sep=".")
    new.fun<-c(new.fun,paste("  ",redo.name," <- function(...){",sep=""))
    # request slider values
    for(i in seq(along=args)){
      new.fun<-c(new.fun,paste("    ",args[i],"<- slider(no=",i,")"))
    }
    # copy body of input function
    new.fun<-c(new.fun,paste("    ",deparse(body(refresh.function))),"  }")
    # construct call of slider for initialisation of slider widget
    new.fun<-c(new.fun,paste("  slider(sl.function =",redo.name,",",sep=""))
    # set labels of the sliders
  new.fun<-c(new.fun,paste("    sl.names   =c(",paste(paste("",args,"",sep="’"),collapse=","),"),
  # set parameters of the sliders
  param<-NULL
  for(i in seq(along=args)){
    h<-NULL
    for(j in seq(along=label.set)){
      var.name<-paste(label.set[j],i,sep=""); h<-c(h,tclvalue(var.name))
    }
    param<-rbind(param,h)
  }
  par.list<-c("sl.mins    ","sl.maxs    ","sl.deltas  ","sl.defaults")
  for(i in seq(along=par.list)){
    new.fun<-c(new.fun,paste("    ",par.list[i],"=c(",paste(param[,i],collapse=","),"),",sep=""))
  }
  # set title of slider widget
  new.fun<-c(new.fun,paste("    title      =","’",paste("slider",fname,sep="."),"’",sep=""))
  # end of slider call
  new.fun<-c(new.fun,"  )")
  # initial call of the redo function
  new.fun<-c(new.fun,paste("  ",redo.name,"()",sep=""))
  # tail of new slider function
  new.fun<-c(new.fun,"}")
  # print composed function as text
  print(noquote(cbind(new.fun)))
  A<-eval(parse(text=new.fun))
  assign(paste("slider",fname,sep="."),A,pos=1)
  cat("new slider function \"",new.fname,sep="","\" defined in .GlobalEnv\n")
  cat("try: > ",new.fname,sep="","(x)\n")
}
# add make slider button and pack it
tkconfigure(doit,command=make.slider.function)
```

2 An improved version

The function defined in the first section allows you to create very simple `slider.*` function. At once new ideas arrive and two of them are implemented in `new.slider.function`.

- Very often you have some starting code that should be evaluated before the slider panel and the refreshing function have been started. `new.slider.function` has an argument which can be used to pass initial statements. Up to now the statements have to be packed as elements of a string vector.
- More interesting is the question how we can add some buttons to a slider panel. The new argument `button.names` allows you to fix a vector of button names. For each of the names a button will be constructed. If someone pushes button *i* the `slider.*` function will save the number of the button which has already been pressed. Then it will envoke the code of `refresh.function`. To get an effect of a button press you have to some

`if`-statements in the code with a condition that integrates the value of the variable `button.pressed`. See the example append below.

6 ⟨*improved* 6⟩ ≡ ⊂ 8

```
new.slider.function<-function(refresh.function,button.names=NULL,starting.code=NULL){
  ⟨get info of input function 3⟩
  ⟨define widget for setting parameters 4⟩
  ⟨define function for button make slider function improved 7⟩
}
```

7 ⟨*define function for button make slider function improved* 7⟩ ≡ ⊂ 6

```
## print(starting.code)
h<-substitute(starting.code)
if(is.character(starting.code)){   ## print("CHAR")
  starting.code<-c("{",paste(" ",starting.code),"}")
} else {                           ## print("CALL")
  starting.code<-deparse(h)
}
## print(starting.code)
make.slider.function<-function(environ=sys.parent()){
    ### header of new slider function
      new.fun<-paste(new.fname,sep="",
                     "<-function(",first.arg,",",paste(args,collapse=","),"){")
    ### initialization of button.pressed
      set.button.pressed<-paste(" ",'slider(obj.name="button.pressed",obj.value=1)')
      new.fun<-c(new.fun,set.button.pressed)
    ### integrate starting code
      starting.code<-paste(" ",as.character(starting.code))
    new.fun<-c(new.fun,starting.code)

    ###refresh function for sliders###
      # header of refresh function
      refresh<-paste("  refresh <- function(...){",sep="")
      # request slider values
      for(i in seq(along=args)){
        refresh<-c(refresh,paste("   ",args[i],"<- slider(no=",i,")"))
      }
      # request button pressed
      refresh<-c(refresh,'    button.pressed<-slider(obj.name="button.pressed")')
    ###copy body of input function and tail
      refresh<-c(refresh,paste("   ",deparse(body(refresh.function))),"  }")
      new.fun<-c(new.fun,refresh)

    ###b.refresh functions for buttons###
      for(i in seq(along=button.names)){
        b.refresh.name<-paste("b.refresh",i,sep="")
        b.refresh<-paste("  ",b.refresh.name,"<-function(...){",sep="")
        b.refresh<-c(b.refresh,
                     paste('    slider(obj.name="button.pressed",obj.value=',i,')',sep=""),
                     "    refresh()",
                     "  }")
        new.fun<-c(new.fun,b.refresh)
      }
  ### construct call of slider for initialisation of slider widget ###
    new.fun<-c(new.fun,"  slider(","    sl.function=refresh,")
    # set labels of the sliders
    new.fun<-c(new.fun,paste("    sl.names   =c(",
                             paste(paste("",args,"",sep="’"),collapse=","),"),"))
    # set parameters of the sliders
    param<-NULL
    for(i in seq(along=args)){
      h<-NULL
      for(j in seq(along=label.set)){
        var.name<-paste(label.set[j],i,sep=""); h<-c(h,tclvalue(var.name))
      }
      param<-rbind(param,h)
    }
    par.list<-c("sl.mins    ","sl.maxs    ","sl.deltas  ","sl.defaults")
    for(i in seq(along=par.list)){
      new.fun<-c(new.fun,paste("    ",par.list[i],
                               "=c(",paste(param[,i],collapse=","),"),",sep=""))
    }

    # set buttons
    if(0<length(button.names)){ # button functions
      new.fun<-c(new.fun,paste("    but.fun   =c(",
                               paste(paste("b.refresh",seq(along=button.names),sep=""),
                                     collapse=","),
                              "),"
               ))
    }
    if(0<length(button.names)){ # button names
      new.fun<-c(new.fun,paste("    but.names =c(",
                               paste(paste("’",button.names,"’",sep=""),collapse=","),
                               "),"
               ))
    }

    # set title of slider widget
    new.fun<-c(new.fun,paste("    title      =","’",
                             paste("slider",fname,sep="."),"’",sep=""))
    # end of slider call
    new.fun<-c(new.fun,"  )")

  ### initial call of the refresh function
    new.fun<-c(new.fun,"  refresh()")
  ### tail of new slider function
    new.fun<-c(new.fun,"  ’ok’","}")
  # print composed function as text
  print(noquote(cbind(new.fun)))
  A<-eval(parse(text=new.fun))
  assign(paste("slider",fname,sep="."),A,pos=1)
  cat("new slider function \"",new.fname,sep="","\" has been defined in .GlobalEnv\n")
  cat("try now: > ",new.fname,sep="","(x)\n")
}
# add make slider button and pack it
```

```
  tkconfigure(doit,command=make.slider.function)
```

8 ⟨*TEST* `new.slider.function` 8⟩ ≡

```
⟨improved 6⟩
myhist<-function(x,breaks=5,col=2){
  hist(x,breaks=breaks,col=col,prob=TRUE)
  if(button.pressed==1){
    lines(density(x))
  }
  if(button.pressed==2){
    rug(x)
  }
  if(button.pressed==3){
    lines(density(x))
    rug(x)
  }
}
new.slider.function(myhist,
                    button.names=c("density","rug","d + r"),
                    starting.code={print("xyz")}
#                    starting.code=c('print("xyz")',1)
                   )
"ok"
```

9 ⟨* 9⟩ ≡

```
#slider.myhist(co2)
a<-cbind("slider.myhist<-"=deparse(slider.myhist))
dimnames(a)[[1]]<-rep("",length(a))
noquote(a)
```

Here is the result of an creation process.

```
Fri Nov 27 18:56:10 2009
 slider.myhist<-
 function (x, breaks, col)
 {
     slider(obj.name = "button.pressed", obj.value = 1)
     {
         print("xyz")
     }
     refresh <- function(...) {
         breaks <- slider(no = 1)
         col <- slider(no = 2)
         button.pressed <- slider(obj.name = "button.pressed")
         {
             hist(x, breaks = breaks, col = col, prob = TRUE)
             if (button.pressed == 1) {
                 lines(density(x))
             }
             if (button.pressed == 2) {
                 rug(x)
             }
             if (button.pressed == 3) {
                 lines(density(x))
                 rug(x)
             }
         }
```

```
    }
    b.refresh1 <- function(...) {
        slider(obj.name = "button.pressed", obj.value = 1)
        refresh()
    }
    b.refresh2 <- function(...) {
        slider(obj.name = "button.pressed", obj.value = 2)
        refresh()
    }
    b.refresh3 <- function(...) {
        slider(obj.name = "button.pressed", obj.value = 3)
        refresh()
    }
    slider(sl.function = refresh, sl.names = c("breaks", "col"),
        sl.mins = c(5, 2), sl.maxs = c(15, 20), sl.deltas = c(1,
            1), sl.defaults = c(5, 2), but.fun = c(b.refresh1,
            b.refresh2, b.refresh3), but.names = c("density",
            "rug", "d + r"), title = "slider.myhist")
    refresh()
    "ok"
}
```

3 Appendix

Code Chunk Index

Object Index

10 ⟨* 9⟩+ ≡

```
f0<-"white"; f1<-"black"; f2<-"red"; f3<-"green"; f4<-"blue"
f2<-f0; f4<-f3; #f3<-f4<-f1; f2<-f0
par(pty="s")
size1<-10  # C
lwd<-3*par()$pin[1]*size1/21
size2<-15/21*size1 # S
plot(3:15,type="n",xlim=c(3,22),ylim=c(3,22),pch=11:20,bty="n",ann=FALSE,axes=FALSE)
size.sektor<-2.2*size1/20*(w<-diff(par()$usr[1:2])/25.96)
points(5,5,cex=size1,lwd=lwd) # C
points(5,5,cex=size2,lwd=lwd,col=f3) # S
delta<-1.8*size1/21*w
segments(5-delta,5,5+delta,5,lwd=lwd,col=f4) # Querstrich

t2xy <- function(t,radius,init.angle=0) {
        t<-t/360
        t2p <- 2*pi * t + init.angle * pi/180
        list(x = radius * cos(t2p), y = radius * sin(t2p))
    }
P <- t2xy(0:45 , size.sektor,180); P$x<-P$x+5; P$y<-P$y+5
polygon(c(P$x, 5), c(P$y, 5),border = "white", col = f2)  # lu
P <- t2xy(0:45 , size.sektor*1.5,0); P$x<-P$x+5; P$y<-P$y+5  # ro
polygon(c(P$x, 5), c(P$y, 5), border = "white", col = f2)
# abline(v=5,h=5)
P <- t2xy(c(20,45,70),size.sektor*2.2,0)
segments(P$x+5,P$y+5,9*P$x+5,9*P$y+5,lwd=1.5*lwd,lty=2,xpd=NA,col=f3)
```

11 ⟨* 9⟩+ ≡

```
f0<-"white"; f1<-"black"; f2<-"red"; f3<-"green"; f4<-"blue"
f2<-f0; f4<-f3; #f3<-f4<-f1; f2<-f0
par(pty="s")
size1<-5  # C
lwd<-3*par()$pin[1]*size1/21
size2<-15/21*size1 # S
plot(3:15,type="n",xlim=c(4,8),ylim=c(4,8),pch=11:20,bty="n",ann=FALSE,axes=FALSE)
size.sektor<-2.2*size1/20*(w<-diff(par()$usr[1:2])/25.96)*2.8 # 2.8PS
points(5,5,cex=size1,lwd=lwd) # C
points(5,5,cex=size2,lwd=lwd,col=f3) # S
delta<-1.8*size1/21*w   *1.5 # 1.5 PS
segments(5-delta,5,5+delta,5,lwd=lwd,col=f4) # Querstrich

t2xy <- function(t,radius,init.angle=0) {
        t<-t/360
        t2p <- 2*pi * t + init.angle * pi/180
        list(x = radius * cos(t2p), y = radius * sin(t2p))
    }
P <- t2xy(0:60 , size.sektor,180); P$x<-P$x+5; P$y<-P$y+5
polygon(c(P$x, 5), c(P$y, 5),border = "white", col = f2)  # lu
P <- t2xy(0:60 , size.sektor*1.5,0); P$x<-P$x+5; P$y<-P$y+5  # ro
polygon(c(P$x, 5), c(P$y, 5), border = "white", col = f2)
# abline(v=5,h=5)
P <- t2xy(c(0,30,60),size.sektor*2.2,0)
segments(P$x+5,P$y+5,6*P$x+5,6*P$y+5,lwd=1.5*lwd,lty=2,xpd=NA,col=f3)
dev.copy(postscript,"logo.ps",width=4,horizontal=FALSE); dev.off()
```