Diseño de Gráficos Interactivos en R

Con datos de *Varieties of Democracy*

Nicolás Schmidt[1] Mathias Silva[2]

[1]Departamento de Ciencia Política
FCS - UdelaR
nschmidt@cienciassociales.edu.uy

[2]Instituto de Economía
FCEA - UdelaR
msilva@ccee.edu.uy

Escuela de Métodos Alacip, 2017

Estructura de la presentación

1. ¿Para qué usar gráficos interactivos?

2. Datos: V-Dem

3. Gráficos animados

4. Aplicaciones Interactivas/GUI

1. ¿Para qué usar gráficos interactivos?

2. Datos: V-Dem

3. Gráficos animados

4. Aplicaciones Interactivas/GUI

Estructura de la presentación

1. ¿Para qué usar gráficos interactivos?

2. Datos: V-Dem

3. Gráficos animados

4. Aplicaciones Interactivas/GUI

¿Para qué usar gráficos animados o interactivos?

- Exploración de datos.
- Presentación de resultados (transparencia).
- Simplificar la replicabilidad de investigaciones.
- Fines didácticos y pedagógicos.

1. ¿Para qué usar gráficos interactivos?

2. Datos: V-Dem

3. Gráficos animados

4. Aplicaciones Interactivas/GUI

Estructura de la presentación

1. ¿Para qué usar gráficos interactivos?

2. Datos: V-Dem

3. Gráficos animados

4. Aplicaciones Interactivas/GUI

Varieties of Democracy

- Parte de 7 concepciones de la democracia: electoral, liberal, participativa, mayoritaria, deliberativa, igualitaria y de consenso.
- Contiene docenas de componentes de la democracia, entre los que se pueden incluir elecciones, independencia judicial, democracia directa y equidad de género, entre otros.
- Provee indicadores desagregados para cada concepción de la democracia así como para cada componente.
- Cubre todos los países del mundo (y en algunos casos a territorios dependientes) desde 1900 hasta la fecha.
- Muy importante!: Hace públicos todas las escalas y mediciones, sin costo alguno, en una interfaz de fácil acceso.

1. ¿Para qué usar gráficos interactivos?

2. Datos: V-Dem

3. Gráficos animados

4. Aplicaciones Interactivas/GUI

Estructura de la presentación

1. ¿Para qué usar gráficos interactivos?

2. Datos: V-Dem

3. Gráficos animados

4. Aplicaciones Interactivas/GUI

Gráficos animados

Paquetes que vamos a utilizar:

```
library(manipulate)
library(animation)
library(dygraphs)
library(DT)
library(ggplot2)
library(gganimate); devtools::install_github("dgrtwo/gganimate")
library(googleVis)
```

Caracteristicas de los paquetes

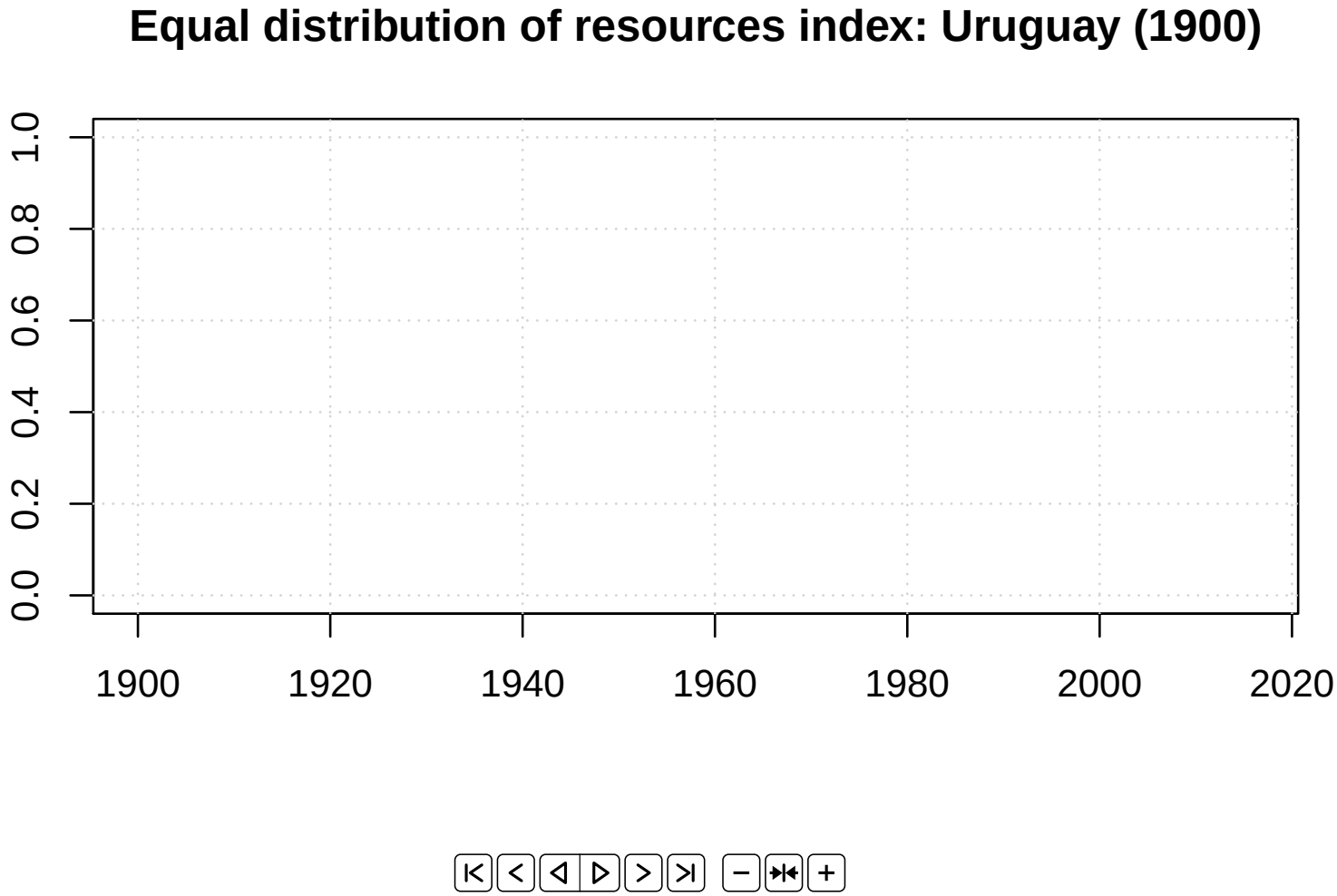
	Funciona con:		Tiene salida con:			
	R	RStudio	LaTeX	HTML	R	RStudio
manipulate	no	si	no	no	no	si
data table	no	si	no	si	no	si
dygraphs	si	si	no	si	no	si
gganimate	no	si	si	si	no	si
googleVis	si	si	no	si	no	no
animation	si	si	si	si	no	no

Con salida a LaTeX es posible hacer un gráfico animado muy simple con el paquete `base` de R:

```
nyears <- length(base2$year)
for(i in 1:nyears)
{
      pdf(paste("graph",i,".pdf",sep=""),height=4,width=6.5)
      x <- base
      if(i<nyears)
            x[,(i+1):nyears] <- NA
      {plot(x~t(years),
           ylim=c(0,1),
           xlab="",
           ylab="",
           main=paste("titulo(",years[1]-1+i,")",sep=""),
           type="l");grid()}
      dev.off()
}
```

Código LaTeX para insertar animación gráfica con controles:

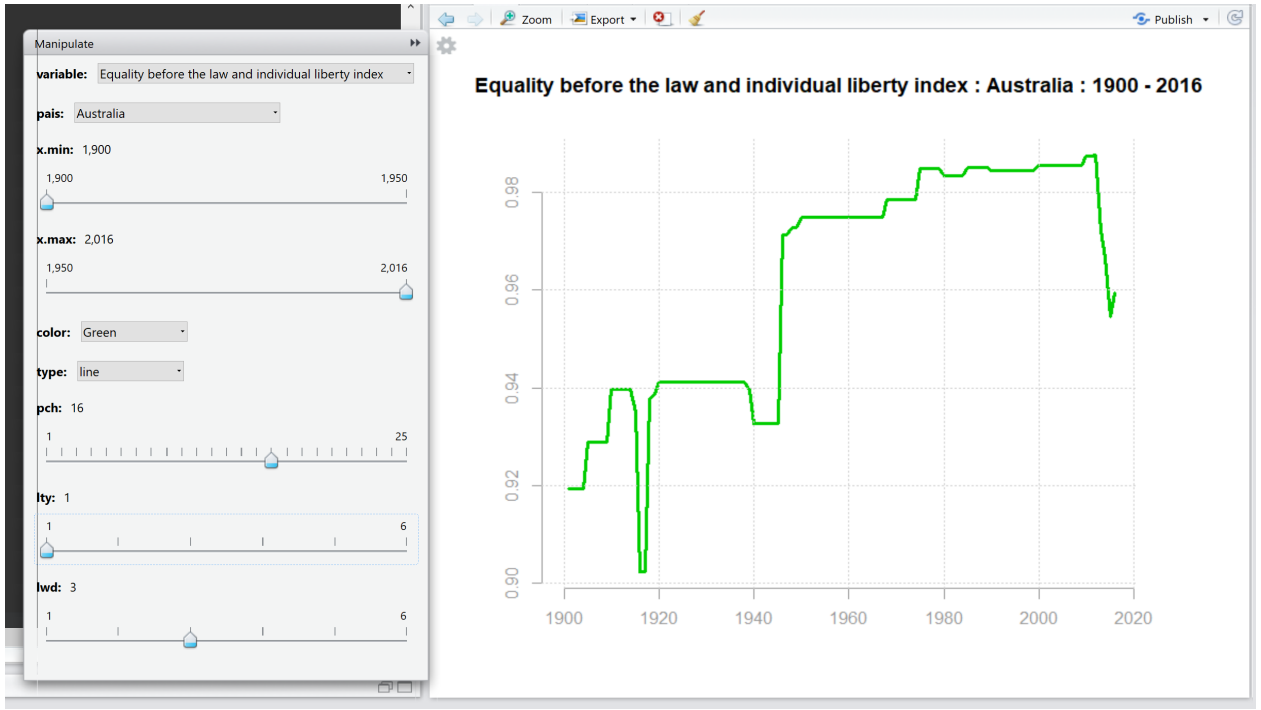
```
\documentclass[a4]{article}
\usepackage{graphicx}
\usepackage{animate}
\begin{document}
\centerline{
   \animategraphics[controls,buttonsize=0.3cm,width=12.5cm]{6}
   {graph}{1}{117}
}
\end{document}
```


Equal distribution of resources index: Uruguay (1900)
0.0
0.2
0.4
0.6
0.8
1.0
1900
1920
1940
1960
1980
2000
2020

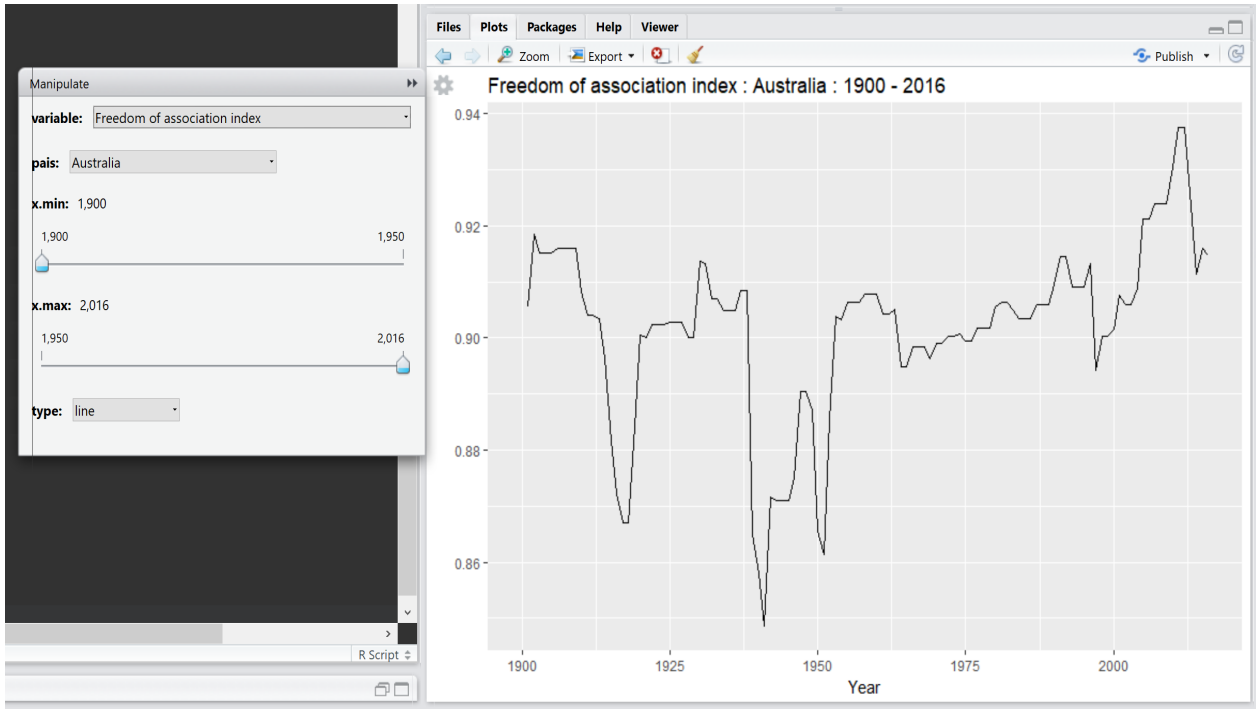
Lo mismo pero con salida HTMLcon la librería animation

```
saveHTML({
      for(i in 1:nyears)
      {
            x <- base3
            if(i<nyears)
                  x[,(i+1):nyears] <- NA
            {plot(x~t(years),
                 ylim=c(0,1),
                 xlab="",
                 ylab="",
                 col=2,
                 lwd=2,
                 main=paste("Equal␣distribution␣of␣resources␣index:␣
                      Uruguay␣(",years[1]-1+i,")",sep=""),
                 type="l");grid()}
      }
},
htmlfile = "index.html", description = "ndice..",ani.height = 500, ani.
    width = 1000,
title = "Demonstration␣of␣the␣Gradient␣Descent␣Algorithm",interval =
    0.03)
```

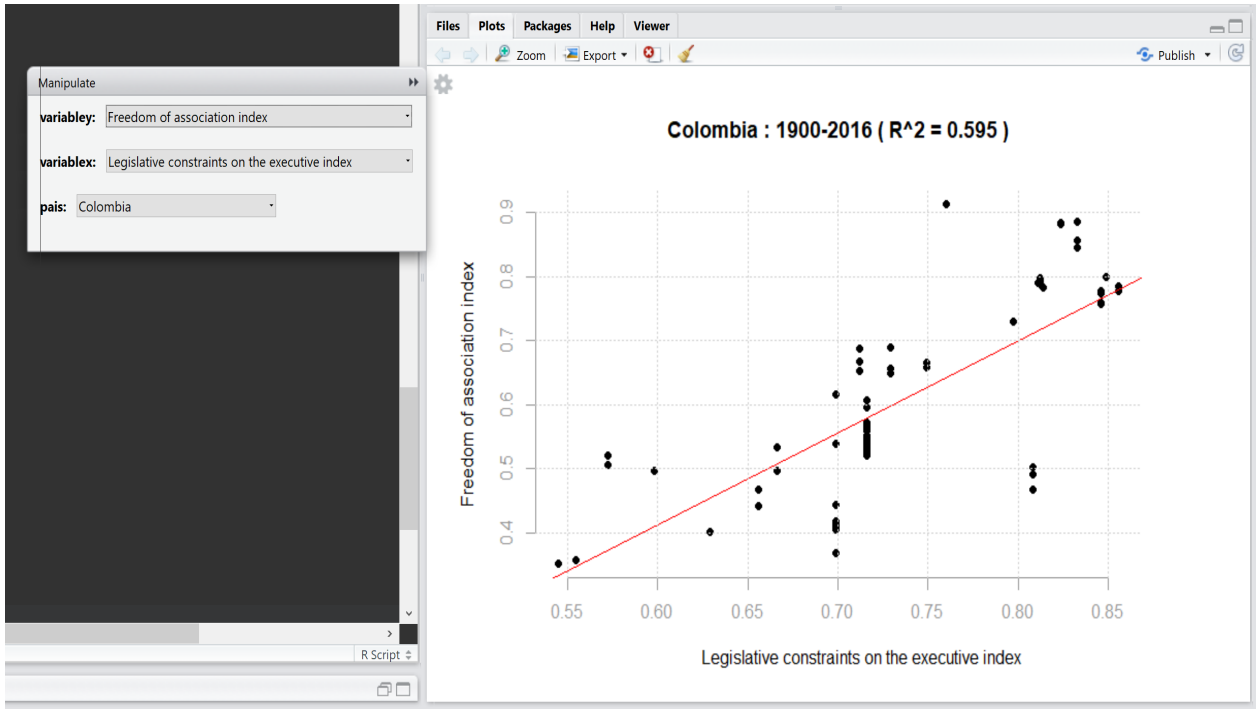
Package 'manipulate': Interactive Plots for RStudio



Package ‘manipulate’: Interactive Plots for RStudio



Package 'manipulate': Interactive Plots for RStudio

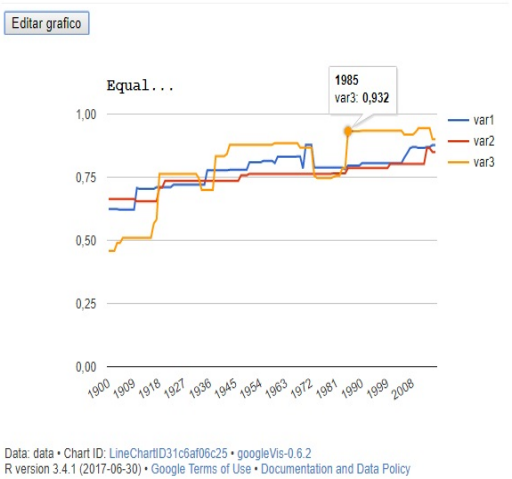


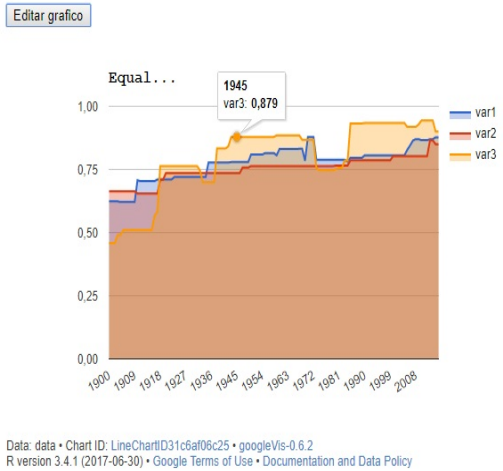
Package 'manipulate': Interactive Plots for RStudio

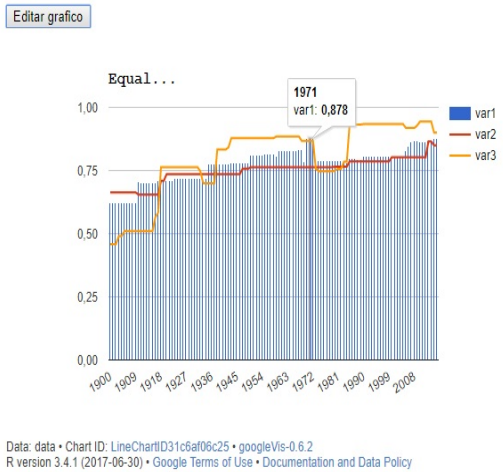
- Principales funciones:

```
manipulate() # entorno global del gráfico interactivo
picker() # desplegable de opciones
       # ejemplo:
       # variable = picker ("variable 1" = "nombre real de la var"...)
slider() # deslizador numérico
       # ejemplo:
       # x.min = slider(1900,1950, initial = 1900)
       # x.max = slider(1950,2016, initial = 1950)
       # en el plot(..., xlim = c(x.min, x.max))
checkbox() # agregar o quitar un elemento del gráfico
```

googleVis


Editar grafico
Equal...
1985
var3: 0,932
1,00
0,75
0,50
0,25
0,00
1900 1909 1918 1927 1936 1945 1954 1963 1972 1981 1990 1999 2008
var1
var2
var3
Data: data • Chart ID: LineChartID31c6af06c25 • googleVis-0.6.2
R version 3.4.1 (2017-06-30) • Google Terms of Use • Documentation and Data Policy


Editar grafico
Equal...
1945
var3: 0,879
1,00
0,75
0,50
0,25
0,00
1900 1909 1918 1927 1936 1945 1954 1963 1972 1981 1990 1999 2008
var1
var2
var3
Data: data • Chart ID: LineChartID31c6af06c25 • googleVis-0.6.2
R version 3.4.1 (2017-06-30) • Google Terms of Use • Documentation and Data Policy

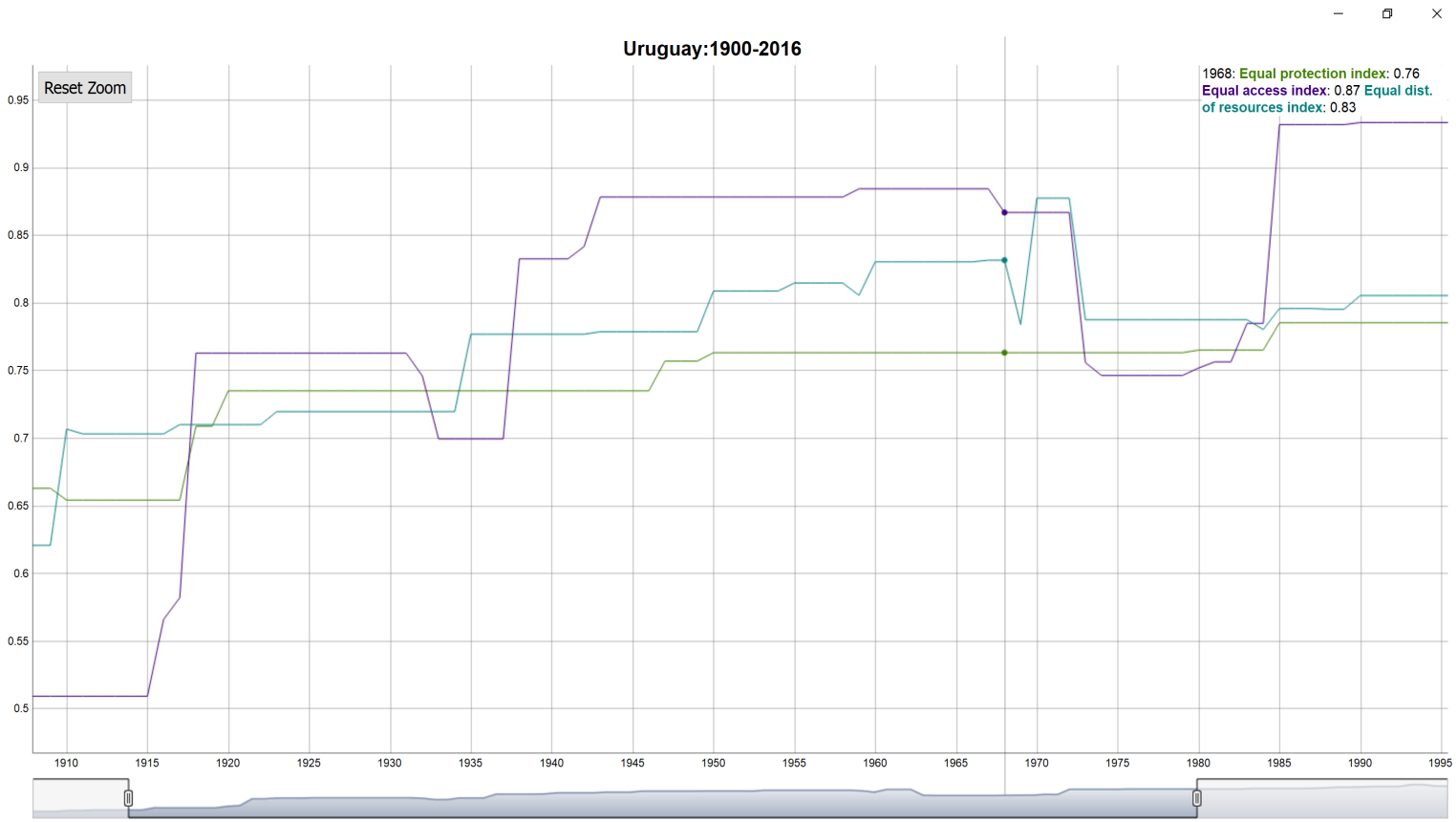

Editar grafico
Equal...
1971
var1: 0,878
1,00
0,75
0,50
0,25
0,00
1900 1909 1918 1927 1936 1945 1954 1963 1972 1981 1990 1999 2008
var1
var2
var3
Data: data • Chart ID: LineChartID31c6af06c25 • googleVis-0.6.2
R version 3.4.1 (2017-06-30) • Google Terms of Use • Documentation and Data Policy

Package 'googleVis': R Interface to Google Charts

- Código:

```
graficoGV <- gvisLineChart(basegv, "years", c("var1","var2","var3"),
                        options=list(
                               vAxes="[{title:'var1'},␣{title:'var2'},␣{
                                    title:'var3'}]",
                               width=1000,
                               height=500,
                               title="Equal...",
                               titleTextStyle="{color:'black',fontName:'
                                    Courier',fontSize:16}",
                               gvis.editor="Editar␣grafico",
                               curveType='function'
                               ))
plot(graficoGV)
```

Package 'dygraphs': Interactive Time Series Charting Library



Package 'dygraphs': Interactive Time Series Charting Library

- Código:

```
base2<-base[base$country_name=="Uruguay",]
colnames(base2)[15:17]<-c("Equal␣protection␣index",
                          "Equal␣access␣index",
                          "Equal␣dist.␣of␣resources␣index")

dygraph(base2[,c(5,15:17)], main = "Uruguay:1900-2016")%>%
      dyRangeSelector(dateWindow = c("1930", "1970"))%>%
      dyUnzoom() %>%
      dyCrosshair(direction = "vertical")
```

1. ¿Para qué usar gráficos interactivos?

2. Datos: V-Dem

3. Gráficos animados

4. Aplicaciones Interactivas/GUI

Estructura de la presentación

1. ¿Para qué usar gráficos interactivos?

2. Datos: V-Dem

3. Gráficos animados

4. Aplicaciones Interactivas/GUI

Introduciendo Tcl/Tk

- Librerías alternativas a la hora de programar una GUI (Qt (RStudio), GTK+ (GNOME), wxWidgets (BitTorrent), Tcl/Tk (RCommander), etc.)
- Tcl/Tk:
  - Tool Command Language (1988, principalmente programado en C, últ. version 8.6.6 (Julio, 2016)).
  - Toolkit (1991, últ. version con Tcl 8.6.6)
- Librería `tcltk` como interfaz para interactuar con un interpretador Tcl desde R. (más detalles aquí).
- Los contenidos de esta presentación refieren a las versiones de R 3.4.0 y Tcl 8.6.

Introduciendo Tcl/Tk

- Los pro de `tcltk`:
  - Única librería para GUI incluída en la librería `base` de R (hace ya al menos 13 años).
  - Compatibilidad cruzada con prácticamente cualquier OS.
  - Extensiones a partir de librerías de R (`tcltk2`, `rpanel`, etc.).
  - Interfaz casi directa con interpretador Tcl = Fácilmente extendible con librerías Tcl, fácil de interpretar documentación de ayuda de Tcl/Tk.
- Los contra de `tcltk`:
  - Poco documentación específica para la programación desde R.
  - Estéticamente añejo en general.
  - Difícil o imposible de utilizar desde algunas IDE (RStudio).

¿Para qué utilizar GUI?

- Presentación interactiva de resultados.
- Diseño de aplicaciones para el relevamiento y sistematizado de datos en trabajo experimental y encuestas.
- Entornos gráficos personalizados para la visualización de datos.
- Fines didácticos/pedagógicos.
- Desarrollo y/o extensión de IDE.

Elementos básicos sobre Tcl/Tk

- Cubiertos aquí:
  - Widgets, y sus relaciones de amo/esclavo (padre/hijo).
  - Geometry Managers y Windows Managers.
  - Variables Tcl.
  - Uso de librerías de Widgets externas.
- NO cubiertos aquí:
  - Event Binding.
  - Definición de funciones en Tcl.
  - Generación de gráficos en Tcl/Tk.

Un primer ejemplo...

```
rm(list=ls())
library(tcltk)
#Creando la ventana...
window0 <-tktoplevel()
#Creando un marco para poner widgets (en este caso el widget button)
button1_frame <- tkframe(parent=window0,height=50,width=100,borderwidth
    =2,relief="raised")
#Creando un widget del tipo button
button1 <- tkbutton(parent=button1_frame,text="Boton de Ejemplo")
#Empaquetado con el GM 'pack' del frame y button
tkpack(button1_frame)
tkpack(button1)
#Asignando un comando/funcion al boton
tkconfigure(button1,command=function(...){tkdestroy(window0)})
```

Una alternativa (preferida)...

```
rm(list=ls())
library(tcltk)
#Creando la ventana...
window0 <- tcl("toplevel", ".tl")
#Creando un marco para poner widgets (en este caso el widget button)
button1_frame <- tcl("frame",paste(as.character(window0),".b1f",sep=""),
    height=50,width=100,borderwidth=2,relief="raised")
#Creando un widget del tipo button
button1 <- tcl("button",paste(as.character(button1_frame),".b1",sep=""),
    text="Boton de Ejemplo",command=function(...){tcl("destroy",as.
    character(window0))})
#Empaquetado con el GM 'pack' del frame y button
tcl("pack",button1_frame)
tcl("pack",button1)
```

Reseña de los widgets básicos de Tcl/Tk

- `toplevel` - `tktoplevel()`
- `frame` - `tkframe()`
- `button` - `tkbutton()`
- `label` - `tklabel()`
- `entry` - `tkentry()`
- `listbox` - `tklistbox()`
- `radiobutton` - `tkradiobutton()`
- `checkbutton` - `tkcheckbutton()`
- `text` - `tktext()`
- `spinbox` - `tkspinbox()`
- `scale` - `tkscale()`
- `scrollbar` - `tkscrollbar()`
- `image` - `tkimage()`
- `menu` - `tkmenu()`

Uso y ubicación de un widget básico en Tcl/Tk

Salvo `image`, todos estos widgets básicos exigen la siguiente sintaxis para su uso:

```
widget pathName ?options?
```

donde `widget` es el nombre del tipo de widget que se quiere utilizar (e.g., `frame`), `pathName` es la ruta que define una relación de padres/hijos desde la ventana toplevel al widget que se está insertando, especificada como los nombres de cada uno de ellos separados por puntos (e.g., `.toplevel1.framelindo1.framechico1.button1`), y `?options?` refiere a los demás argumentos que admite cada widget. Estos argumentos pueden ser básicamente de 2 tipos: Standard Options, o Widget-Specific Options.

Uso y ubicación de un widget básico en Tcl/Tk

- Una vez definido el widget, con su ruta y opciones, resta definir dónde ubicarlo gráficamente en el espacio gráfico abarcado por su padre inmediato (e.g., “en la esquina derecha superior”, “en la parte de la derecha”, “en el punto X e Y”, etc.),. Para esto se utilizan Geometry Managers (GM).
- Existen 3 formas de GM básicas: `pack`, `grid`, y `place`. Las mismas son incompatibles entre sí dentro de un mismo widget, por lo que la forma en que se inserta el primer hijo/esclavo define el GM a usar para los restantes hijos inmediatos de ese widget.

Uso y ubicación de un widget básico en Tcl/Tk

- `pack` ubica al hijo como anclado en algúno de los 4 bordes (*sides*) de la región gráfica abarcada por el padre (top, bottom, left, o right) - `tkpack(\ldots)`
- `grid` particiona a la región gráfica del padre en una grilla cuadriculada y ubica al hijo en una región especificada de dicha grilla - `tkgrid(\ldots)`
- `place` es significativamente más complejo y flexible, al ubicar al hijo en las coordenadas relativas (x,y) y con un tamaño relativo especificado en la región gráfica abarcada por su padre - `tkplace(...)`

Uso y ubicación de un widget básico en Tcl/Tk

- En R, existen dos alternativas para definir y ubicar un widget con la librería `tcltk`:
  - Usando la función de R específica para ese tipo de widget, donde las opciones . Ejemplo:

```
primerframe <- tkframe(parent=...,...)
```

  - Usando la función general de R para interactuar con el interpretador Tcl (`tcl(...)`). Ejemplo:

```
primerframe <- tcl("frame",pathName,...)
```

- Lo mismo para usar los GM:

```
tkpack(primerframe,...)
```

```
tcl("pack",primerframe,...)
```

Otro ejemplo...

```
rm(list=ls());library(tcltk)
window0 <- tcl("toplevel", ".tl")
button1_frame <- tcl("frame",paste(as.character(window0),".b1f",sep=""),
    height=50,width=100,borderwidth=2,relief="raised")

frame2 <- tcl("frame",paste(as.character(window0),".b2f",sep=""),height
    =50,width=100,borderwidth=2,relief="raised")

button1 <- tcl("button",paste(as.character(button1_frame),".b1",sep=""),
    text="Boton de Ejemplo",command=function(...){tcl("destroy",as.
    character(window0))})

button2 <- tcl("button",paste(as.character(button1_frame),".b2",sep=""),
    text="Boton de Ejemplo 2",command=function(...){tcl("destroy",as.
    character(window0))})

label1 <- tcl("label",paste(as.character(frame2),".l1",sep=""),text="
    Presione un botón")

tcl("grid",button1_frame,column=3,row=3)
tcl("grid",frame2,column=3,row=1)
tcl("pack",button1,side="top")
tcl("pack",button2,side="bottom")
tcl("pack",label1,side="right")
```

Configuración de un widget ya creado

Una vez creado un widget podemos eventualmente querer cambiar algunos de sus argumentos, como por ejemplo cambiarle el comando que se ejecuta al interactuar con dicho widget. Para esto existen dos alternativas:

```
tkconfigure(widget,...)
```

```
tcl(widget,"configure",...)
```

Compartido de datos entre R y Tcl

Algunos widgets pueden controlar valores de una o varias variables, por lo que es necesaria una forma de llevar/traer a R los valores de esas variables que se están manipulando en el interpretador Tcl.
Para definir una variable que debe ser compartida entre R y Tcl, se puede usar el comando `tclVar()`. A su vez, para leer en R los valores de una variable Tcl se puede usar el comando `tclvalue()`.

Compartido de datos entre R y Tcl (Ejemplo)

```
rm(list=ls());library(tcltk)
entrada1 <- tclVar()
entrada2 <- tclVar()
ventana0 <- tcl("toplevel",".v0")
frametop <- tcl("frame",paste(as.character(ventana0),".ft",sep=""),
    height=50,width=100,borderwidth=2,relief="raised")
framemiddle <- tcl("frame",paste(as.character(ventana0),".fm",sep=""),
    height=50,width=100,borderwidth=2,relief="raised")
entrytop <- tcl("entry",paste(as.character(frametop),".entry1",sep=""),
    textvariable=entrada1)
entrymiddle <- tcl("entry",paste(as.character(framemiddle),".entry2",sep
    =""),textvariable=entrada2)
buttonmiddle <- tcl("button",paste(as.character(framemiddle),".butt1",
    sep=""))

buttonFunc <- function(...){
   print(as.integer(tclvalue(entrada1))+as.integer(tclvalue(entrada2)))
}
tcl(buttonmiddle,"configure",command=buttonFunc,text="Calcular suma")

tcl("pack",frametop,side="top")
tcl("pack",framemiddle,side="bottom")
tcl("pack",entrytop)
tcl("pack",entrymiddle)
tcl("pack",buttonmiddle,side="top")
```

Trabajo con librerías externas de Tcl/Tk

Existen librerías de Tcl/Tk con widgets distintos a los incluídos en la librería básica de Tk. La librería `tcltk` de R brinda una forma de poder utilizar estas librerías de Tcl/Tk a través de básicamente 2 funciones:

- `addTclPath()` - Para especificar la ruta en la computadora hasta el directorio donde están los scripts .tcl de la librería Tcl/Tk
- `tclRequire()` - Para cargar la librería Tcl/Tk.

Para widgets fuera de los incluídos en la librería básica de Tk no existen funciones especificas en R, por lo que solo se pueden usar a través del comando `tcl(...)`.

Ejemplo con librería BWidget

```
rm(list=ls());library(tcltk)
#addTclPath("ruta/hasta/bwidget-1.9.10")
tclRequire("BWidget")

window0 <- tcl("toplevel",".tl0",height=250,width=250)
#Usando un nuevo widget (ScrolledWindow)...
scroll_win1 <- tcl("ScrolledWindow",paste(as.character(window0),".sw1",
    sep=""))
#Otro nuevo widget (ScrollableFrame) dentro de la ScrolledWindow
scroll_frame1 <- tcl("ScrollableFrame",paste(as.character(scroll_win1)
    ,".sf1",sep=""))
#BWidget: Le damos a la ScrolledWindow el control de la ScrollableFrame
tcl(scroll_win1,"setwidget",scroll_frame1)
#Creamos una imagen .png desde R
png(filename="plot1.png",width=1000,height=1000)
plot(rnorm(100),rnorm(100))
dev.off()
#Usando el widget image...
imagen1 <- tcl("image","create","photo",file="plot1.png")
#Colocamos graficamente la imagen como un widget label
labelimage1 <- tcl("label",paste(as.character(tcl(scroll_frame1,"
    getframe")),".im1",sep=""),image=imagen1)
tcl("pack",labelimage1)
tcl("pack",scroll_win1)
```

Como utilizar la documentación de Tcl/Tk

La documentación de los comandos Tk (básica aquí) puede fácilmente adaptarse a la sintáxis que uno usa a la hora de programar GUI con la librería de R `tcltk`. La regla general para esto es:

- Si se usa una función específica de `tcltk` para el widget/comando (listado entero de funciones puede verse con `ls("package:tcltk")`) entonces los argumentos del widget/comando se definen como los argumentos de esa función. Además, generalmente se puede usar el argumento `parent=...` para especificar el objeto de R que actuará como padre inmediato del widget.

Como utilizar la documentación de Tcl/Tk

- Si se utiliza la función de interacción general con el interpretador Tcl `tcl()`, entonces se utiliza la *synopsis* definida en la documentación, separando cada argumento con una coma, y donde los primeros argumentos son los no opcionales y las opciones se especifican como argumentos nombrados (como lo es generalmente en R).

Como utilizar la documentación de Tcl/Tk (Ejemplo)

NAME

button — Create and manipulate button widgets

SYNOPSIS

button *pathName* ?*options*?

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-repeatdelay, repeatDelay, RepeatDelay
-repeatinterval, repeatInterval, RepeatInterval
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

WIDGET-SPECIFIC OPTIONS

Command-Line Name: **-command**
Database Name: **command**
Database Class: **Command**
Specifies a Tcl command to associate with the button. Thi

Command-Line Name: **-default**
Database Name: **default**
Database Class: **Default**
Specifies one of three states for the default ring: **normal**,
drawn with the platform specific appearance for a non-de
disabled state, the button is drawn with the non-default b

Command-Line Name: **-height**
Database Name: **height**
Database Class: **Height**
Specifies a desired height for the button. If an image or b
text. If this option is not specified, the button's desired he

Command-Line Name: **-overrelief**
Database Name: **overRelief**
Database Class: **OverRelief**
Specifies an alternative relief for the button, to be used w
value of this option is the empty string, then no alternativ

Command-Line Name: **-state**
Database Name: **state**
Database Class: **State**
Specifies one of three states for the button: **normal**, **acti**
the pointer is over the button. In active state the button i
bindings will refuse to activate the widget and will ignore

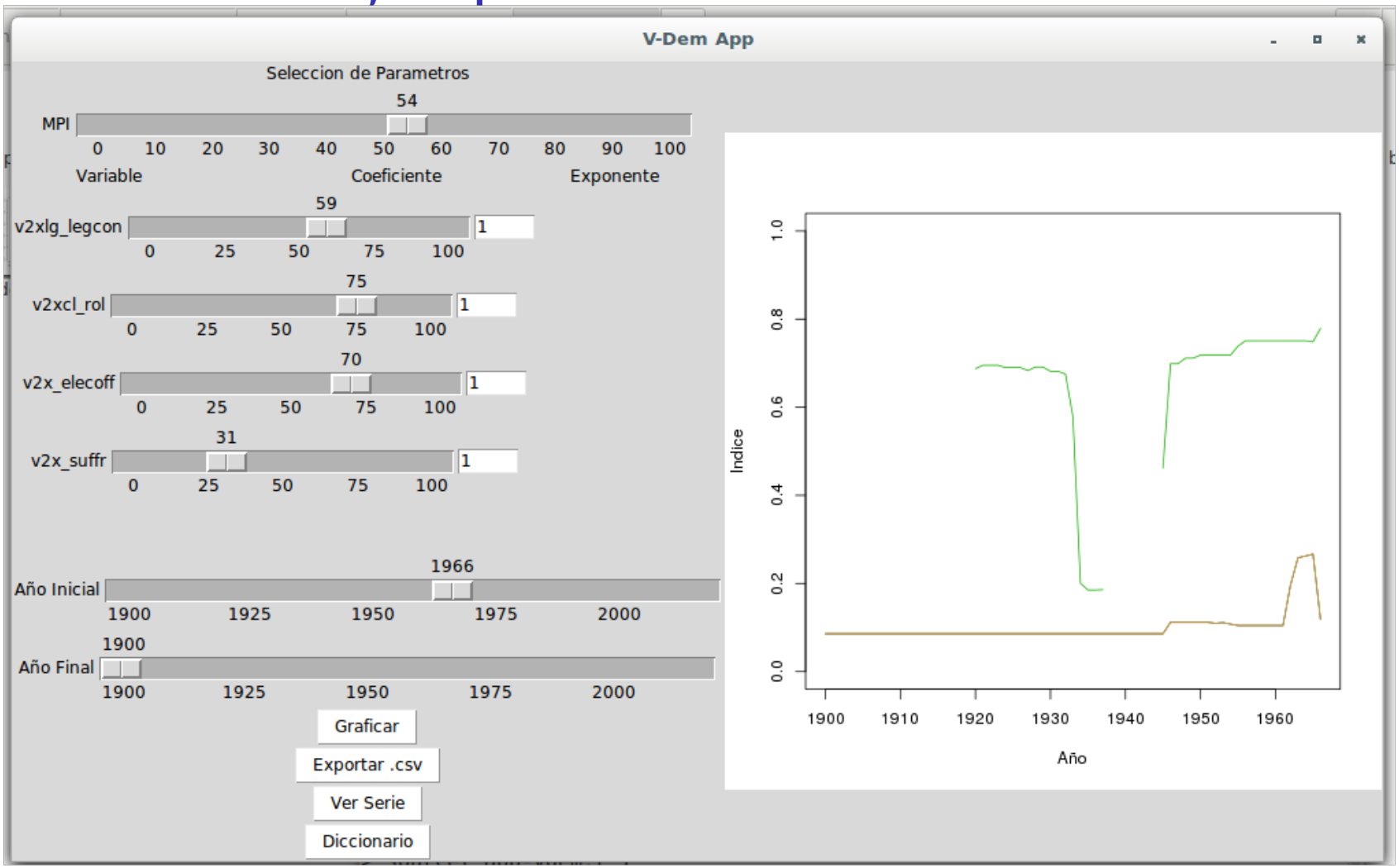
Command-Line Name: **-width**
Database Name: **width**
Database Class: **Width**
Specifies a desired width for the button. If an image or bi
image or with **-compound none**) then the width specifie
specified, the button's desired width is computed from th

Como utilizar la documentación de Tcl/Tk (Ejemplo)

```
#Asumiendo button1_frame creado con tcl()
button2 <- tcl("button",paste(as.character(button1_frame),".b2",sep=""),
text="Boton de Ejemplo 2",command=function(...){tcl("destroy",as.
character(window0))})
#Asumiendo button1_frame creado con tkframe()
button2 <- tcl("button",paste(as.character(button1_frame[["ID"]]),".b2",
    sep=""),
text="Boton de Ejemplo 2",command=function(...){tcl("destroy",as.
character(window0))})
```

```
#Asumiendo button1_frame creado con tkframe()
button2 <- tkbutton(parent=button1_frame,text="Boton de Ejemplo 2",
    command=function(...){tcl("destroy",as.character(window0))})
```

Aplicación de Ejemplo: Indices con V-Dem



Para correr la aplicación de ejemplo, solo es necesario descargar los scripts y la base de acá, y luego teniendo todos los scripts en un mismo directorio, correr en R:

```
source("app-vdem.r")
```

Algunos comentarios finales...

- Una forma de programar que puede resultar más prolija, ordenada, y que aprovecha el potencial gráfico de otras librerías de R es:
  - i Trabajar cada ventana `toplevel` en un script separado.
  - ii Exportar los gráficos e imágenes que se quiere incorporar desde R y luego importarlos a Tcl como `image`.
  - iii Tratar en script aparte los widgets o comandos más complejos.
  - iv Utilizar `frame` o widgets similares para particionar controladamente el espacio gráfico de la ventana `toplevel`, y luego incorporarle como hijos a estos los widgets que se quiera usar en su región gráfica.

Algunos comentarios finales...

- Si un widget fue creado con una función de `tcltk` especifica, entonces su *pathName* es su subelemento `$ID`. Si fue creado con la función `tcl()`, su *pathName* es el objeto R del widget como `as.character()`.
- El argumento `parent=...` en las funciones específicas para widgets solo admite objetos de R también creados con funciones específicas, por lo que un widget creado con una función específica para sí no podrá tener como padre inmediato un widget que fue creado con la función `tcl()`. El caso inverso sí es posible.
- El uso de la función `tcl()` para la definición de los widgets y comandos Tcl es ampliamente más potente que el uso de las funciones específicas para cada widget básico (como `tkframe()`). Una de las principales razones es que no existen funciones específicas para widgets 'no básicos'.
- La función `later()` de la librería `later` puede ser útil para programar eventos que se disparan con cierto retraso

Algunas Referencias

- Dalgaard , P. (2001) “A Primer on the R-Tcl/Tk Package”. R News, Vol. 1/3.
- Lawrence, M. & Verzani, J. (2014) “Programming Graphical User Interfaces with R”. Ed. Chapman & Hall.
- Welch, B. (2003) “Practical Programming in Tcl and Tk”. Ed. Prentice Hall.