Tutorial and Laboratories

11464 – 11524G AR/VR for Data Analysis and Communication

3D Plots in R - Week 2

Introduction

In this computer lab you will practice data visualisation using **3D plots**. In particular, you will learn about visualising data in 3D, how to create plots, their common parameters, and different examples of data visualisation that are useful in data science.

It is assumed that you are now familiar with base R to create plots, how to subset data from data frames, how to enter data or read data files, how to plot data. These topics were covered in our previous tutorials (online material). It is recommended to complete the tutorials in week 1 before attempting this tutorial.

Skills Covered in this tutorial include:

- 3D Plotting in Rstudio.
- Data visualisation using *plot3D(), threejs(), rgl(), plotly().*
- Using different plot types in 3D.

Note: Do not copy-paste the commands. As you type each line, you will make mistakes and correct them, which make you think as you go along. Remember, that the objective is that you understand the commands and master the concepts, so you can reproduce their principles on your own later (e.g., during the final assessment).

For this tutorial we will use the iris dataset in most of the examples, the dataset is available in R. This dataset was introduced by Ronald Fisher in his paper "The use of multiple measurements in taxonomic problems". It contains three iris species (setosa, virginica, versicolor) and four measures (variables) for 50 flowers from each of 3 species of iris, with all measurements given in centimetres. To load the dataset, do the following:

```
# load dataset
data("iris")
# explore dataset
View(iris)

# create vectors for plots
x <- iris$Sepal.Length
y <- iris$Petal.Length
z <- iris$Sepal.Width
```

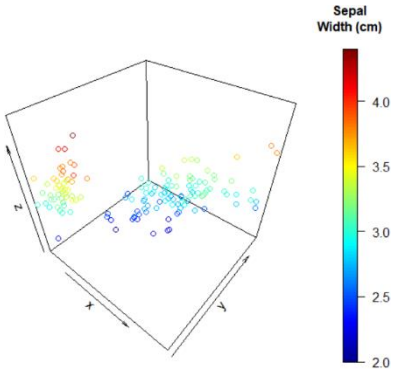
1. 3D plots with *plot3D*

The plot3D package contains many functions for 2D and 3D plotting. For example, scatter3D, points3D, lines3D, hist3D, etc. In this tutorial we will cover the scatter plot only, you can try the other types on your own. First, let's install and load the package.

```
# install packages
install.packages("plot3D")
# load package
library("plot3D")
```

Exercise 1. Generally, 3D plots are useful when we have three variables, and we want to plot their values on the x-y-z space. In this package, by default, the points are coloured automatically using the values in variable z. Let's create a simple 3D scatter plot, do the following.

```
# basic scatter plot
scatter3D(x, y, z,
clab = c("Sepal", "Width (cm)"))
```



You can avoid the colour bar using the colkey = FALSE.

```
# remove colour bar
scatter3D(x, y, z, colkey = FALSE)
```

Now, let's change some of the parameters of the previous plot. For example, colvar = NULL avoids coloring by z variable, col = "blue" changes data points to blue, pch = 19 changes the point shapes, and cx=0.5 changes the size of points. Try with different values and observe the difference, please do the following:

```
# change the style
scatter3D(x, y, z, colvar = NULL, col = "blue", pch = 19, cex = 0.9)
```

You can also change the type of box around the plot with the argument *bty*. Some of the allowed values are:

- f – full box.
- b – only the back panels are visible (default).
- b2 – black panels with visible grid lines.
- n – no box is drawn, same as setting box=FALSE.
- g – grey background and white grinds lines.
- bl – black background.
- bl2 – black background with white grid lines.
- u – user will specify the arguments col.axis, col.panel, lwd.panel, col.grid, lwd.grid manually.

Try different values and observe the result.

```
# full box
scatter3D(x, y, z, bty = "f", colkey = FALSE, main ="bty='f'")

# grey background with white grid lines and tick numbers
scatter3D(x, y, z, bty = "g", colkey = FALSE, main ="bty='g'", ticktype = "detailed")
```

Also, it you want to create your own style you need to set bty=u. Please do the following and try different values.

```
# User defined
scatter3D(x, y, z, pch = 18, bty = "u", colkey = FALSE,
      main ="bty= 'u'", col.panel ="royalblue", expand =0.4,
      col.grid = "linen")
```

You can access a full list of colours in this link: https://www.nceas.ucsb.edu/sites/default/files/2020-04/colorPaletteCheatsheet.pdf

2. 3D plots with *threejs*

The main limitation of plot3D package is its lack of interaction once the plot is obtained. Threejs package overcome this limitation, this package provides high-quality interactive 3D plots, network plots, and globes using the three.js (java script) visualisation library. First, let's install the package.

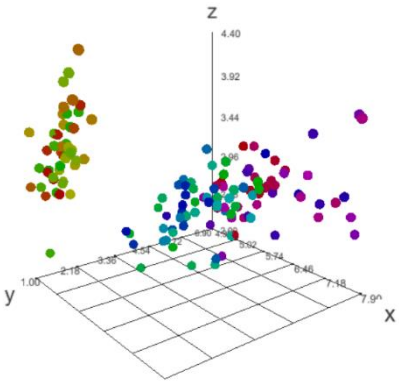
```
# install packages
install.packages("threejs")

# in case that the installation fails, install any of these and then install the threejs package again.
install.packages("dplyr")
install.packages("rlang")  # or this package, but not both.

# load libraries
library(rlang)
library(threejs)
```

Exercise 2. Let's try to generate the same scatter plot with the iris dataset, this time using the threejs package. Please, try the following.

```
# scatter plot
scatterplot3js(x, y, z,
        color=rainbow(length(z)))
```



You can interact with the plot using the mouse. Click the left button of your mouse and drag to change the orientation of your plot. Also, try to zoom in and out using the scroll wheel of your mouse.

If you want to change the colours of the data points, you can generate a vector of n contiguous colors using the functions rainbow(n), heat.colors(n), terrain.colors(n), topo.colors(n), and cm.colors(n). Try different colours and see the results, for example:

```
# change the style of data point
scatterplot3js(x, y, z,
        color=heat.colors(length(z)))

# or you can specify a colour for each type of iris
N = length(levels(iris$Species))
scatterplot3js(x, y, z,
        size = 0.6,
        color=rainbow(N)[iris$Species])
```

You can also change the points style with the *pch* parameter. There are three special cases for this parameter: "o" plotted points appear as 3D spheres, "@" plotted points appear as stroked disks (default), and "." points appear as tiny squares. Apart from these, you can also specify any character, e,g., "x", "*", "18", etc. Try some different points style and observe the results.

```
# change the style of data point
scatterplot3js(x, y, z,
        pch = "o",
        color=rainbow(length(z)))
```

This package also allows to create more complex graphs, for network analysis or population analysis using the maps package. For instance, we can plot the interaction in the ego network data using the *graphjs()* function. The data contains 4039 vertices and 88234 edges from the Stanford SNAP network repository http://snap.stanford.edu/data/facebook_combined.txt.gz.

```
# Example of network visualisation
data(ego)
graphjs(ego, bg="black")
```

Another example is using the *globejs()* function to plot the population in the 500 largest cities in the world. This example uses the *world.cities* dataset from the *maps* package. Please try the following and observe the results.

```
# Load data first
install.packages("maps")
data(world.cities, package="maps")
cities <- world.cities[order(world.cities$pop,decreasing=TRUE)[1:500],]
value  <- 100 * cities$pop / max(cities$pop)

# plot using globejs plot
globejs(bg="white",
      lat=cities$lat,
      long=cities$long,
      value=value,
      rotationlat=-0.34,
      rotationlong=-0.38,
      fov=30)
```

3. 3D plots with *rgl*

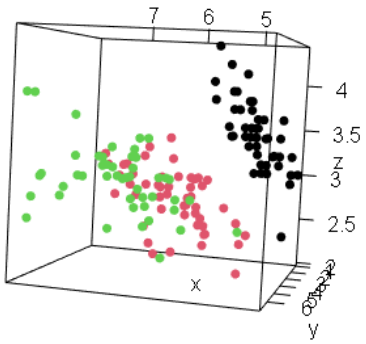
The *rgl* package is one of the best tools to work in 3D from R. This package creates interactive charts, with the ability to zoom in/out and rotate to make the plot more insightful. The visualisation is produced within a *rgl* window. First, let's install the package.

```
# install packages
install.packages("rgl")
# load package
library("rgl")
```

Exercise 3. Let's try to generate the same scatter plot with the iris dataset, this time using the *rgl* package. Please, try the following.

```
# generate a scatterplot
plot3d(x,y,z)

# add colour to the data points and increase size
plot3d(x,y,z,
     size = 7,
     col=as.numeric(iris$Species))
```



You change the style of data point using the *type* parameter. Supported types are “p” for points (default), “s” for spheres, “l” for lines, “h” for segments (projections) from z=0. Try different styles and observe their results.

```
# change the style of data points
plot3d(x,y,z, type="s",
      size = 2,
      col=as.numeric(iris$Species))

# add labels
plot3d(x,y,z, type="s",
      size = 2, col=as.numeric(iris$Species),
      xlab="Sepal Length", ylab="Sepal Width", zlab="Petal Length")
```

As you have seen, if you call *plot3d()* multiple times it will overwrite the current plot. To open a new rgl graphics window, use *open3d()* and then use *plot3d()* to create your plot.

Another feature from the *rgl* package is that it allows the creation of animated plots. This is allowed with the *play3d()* function, which calls a function repeatedly for a number of seconds using the *duration* parameter. The *spin3d* function allows to spin the chart specifying the *axis*, and the speed (*rpm*) to spin. Try the following, use different values, and observe the result.

```
# create an animated 3d scatterplot chart
# Static chart
plot3d(x,y,z, col=as.numeric(iris$Species), type = "s", radius = .2 )      # DO NOT close the window

# We can indicate the axis and the rotation velocity
play3d(spin3d( axis = c(0, 1, 0), rpm = 20), duration = 10 )          # DO NOT close the window
```

If you want to save your image as a gif file and use it in a presentation or published it in a website, you can use the *movie3d()* function.

```
movie3d(movie="3dAnimatedScatterplot",
       spin3d( axis = c(0, 0, 1), rpm = 7),
       duration = 5,
       dir = getwd(),
       type = "gif",
       clean = TRUE)
```

You also need to install the magick package before running movie3d():

```
# install magick package
install.packages("magick")
library(magick)
```

4. 3D plots with *plotly*

Plotly allows to create charts using the *plot_ly()* function. This package offers publication-quality graphs online. It includes 2D and 3D plots such as line plots, scatter plots, area plots, bar chart, error bars, box plots, histograms, heatmaps, and 3D charts. First, let’s install the package.

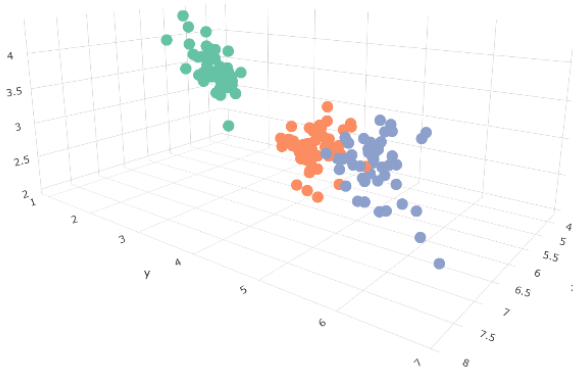
```
# install packages
install.packages("plotly")
# load package
library(plotly)
```

Exercise 4. Let's try to generate the same scatter plot with the iris dataset, this time using the *plotly* package. Once the plot is created, you can explore the plot when you hover the mouse on the data points and obtain the values for each data point. Please, try the following

```
# scatter plot
plot_ly(iris, x=~x, y=~y, z=~z)

# you can also pass the names of the variables
plot_ly(iris,
      x=~Sepal.Length,
      y=~Sepal.Width,
      z=~Petal.Length)

# add colour
plot_ly(iris, x=~x, y=~y, z=~z, color = ~Species)
```



You might a get comments/notes in the console regarding the type of plot not being specified. So, the function uses the most appropriate plot for the type of data that is supplied. One way, to avoid this message is to create a plotly object first. Try the following.

```
# scatter plot
p <- plot_ly(iris, x=~x, y=~y, z=~z)
p <- add_markers(p, color = ~Species)
p

# add labels to the plot
variables <-names(iris)
layout(p, scene = list(xaxis = list(title = variables[1]),
                       yaxis = list(title = variables[2]),
                       zaxis = list(title = variables[3])))
```

You can also use pipes:

```
# scatter plot
p <- plot_ly(iris, x=~x, y=~y, z=~z) %>%
      add_markers(color = ~Species)
p
```

For any plot in R, you can use 3 types of palettes to specify the colour. These are:

- Sequential palettes are suited to ordered data that progress from low to high (gradient). The palettes names are : Blues, BuGn, BuPu, GnBu, Greens, Greys, Oranges, OrRd, PuBu, PuBuGn, PuRd, Purples, RdPu, Reds, YlGn, YlGnBu YlOrBr, YlOrRd.
- Diverging palettes put equal emphasis on mid-range critical values and extremes at both ends of the data range. The diverging palettes are : BrBG, PiYG, PRGn, PuOr, RdBu, RdGy, RdYlBu, RdYlGn, Spectral
- Qualitative palettes are best suited to representing nominal or categorical data. They not imply magnitude differences between groups. The palettes names are : Accent, Dark2, Paired, Pastel1, Pastel2, Set1, Set2, Set3

You can try any of these palettes as follows:

```
p <- plot_ly(iris, x=~x, y=~y, z=~z, color = ~Species)
add_markers(p, color = ~Species, colors = "Accent")
add_markers(p, color = ~Species, colors = "Set1")
add_markers(p, color = ~Species, colors = "PuBu")
```

You can get more information about colour names and palettes:
http://www.sthda.com/english/wiki/colors-in-r

Another type of plot useful in data science and engineering applications is the surface plot. This type of plot is achieved by setting the type = "surface", this type of plot expects a matrix as input. The

rows and columns describe a grid, and the cell value is mapped to the surface height. You can get inside and outside the plot using the scroll wheel. Let's plot an example, try the following.

```
# surface plot
p <- plot_ly(z = volcano, type = "surface")
p
```

If you need to save the interactive plot, another way is to save it as html file. You can achieve this using the *htmlwidgets* package, you need to install the package first and then save the plot. Please do the following:

```
# Install and load package
install.packages("htmlwidgets")
library(htmlwidgets)

# save the plot in working directory
saveWidget(p, file=paste0( getwd(), "/3dSurface.html"))
```

5. Take Home Exercises

5.1 Flights dataset. This dataset contains 34,296 observations of 4 variables, origin_lat, origin_long, dest_lat, and dest_long. The data represents the origin and destination of hundreds of international flights around the world. The dataset is loaded with the *threejs* package. You can also get the complete dataset from: https://raw.githubusercontent.com/callumprentice/callumprentice.github.io/master/apps/flight_stream/js/flights_one.js

load the data set (data(flights)) and obtain the following:

1. Combine the destination coordinates with a precision of two decimals in a single variable, use a ":" (colon) as separator. Store the result in a separated variable. Hint:
   `var3 <- sprintf("%.2f:%.2f", var1, var2)`
2. Obtain the total number of flights that arrived at each airport. Make sure your variable can be counted. Hint: table()
3. Then, order the variable in descending order.
4. Obtain the top-ten (top10) destinations.
5. Now that you know what airports are the most popular, obtain the origin information (latitude and longitude) for all the flights that travelled to the top10 destinations.
6. Create a data frame that includes the origin (lat,long) and destination (lat,long) (in that order) of all the flights to the top10 destinations.
7. Plot the data to visualise the results. Hint:

```
globejs(lat=top10lat,          # vector with the latitude of the top10 destinations
        long=top10long,        # vector with the longitude of the top10 destinations
        arcs=df)   # a data frame (ori_lat, orig_long, dest_lat, dest_long) containing all the flights
                   # to the top10 destinations.
```

You can try some parameters to enhance the design, for example, *arcsHeight* controls the height of the arcs, *arcsLwd* controls the thickness of the arcs (these two parameters accept a value between 0 and 1), *arcsColor* to change the colour (e.g., arcsColor="#00FF00"), *atmosphere = TRUE* for atmosphere effect.

Example of expected output: