Open Reproducible Research in Empirical Science

Olivier Flores

UMR PVBMT, Université de La Réunion / CIRAD
1 Principles

2 Local Workflow

3 Data

4 Software integration

5 Document production
Reproducibility

One of the main principles of the scientific method, refers to the ability of a test or experiment to be accurately reproduced, or replicated, by someone else working independently.

(Wikipedia)

Notion present in *Discours de la Méthode* (Descartes, 1637)
“Generating **verifiable knowledge** has long been scientific discovery’s central goal, yet today it’s impossible to verify most of the [computational] results that scientists present at conferences and in papers.”

About scientific articles: “There is a leap of faith required by the reader; they must believe that the transformations and model fitting were done appropriately and without error.”
Wishful thinking or reality?

Essential in principle, difficult in general,

Energy- and time-saving on a daily basis: to share and communicate with collaborators, colleagues, students,...
Interests and motivation. IV  
Wishful thinking or reality?

**Objective**

Propose a simple workflow based on open tools to easily share research results, if not with the World, at least within a collaborative group.
The basis: a distributable and executable unit

Components

- **Compendium**: a special form of knowledge that “combines text, data and auxiliary software (code) into a distributable and executable unit”

- **Dynamic documents**:
  - the three unitary elements that can be “extracted and processed in various different ways by both the author and the reader”
  - Sequence of *text chunks* and *code chunks*
  - **Text** chunks: description for reading purpose
  - **Code** chunks: sequence of commands to be interpreted by general purpose software
  - **General purpose** software (R, Perl, ...) and **auxiliary** software (user’s code)
  - Relations between chunks, not necessarily linear
The box contents

Folder structure

Box = Working folder

- Three primary subfolders: code, data, text
- Simple names
- Avoid multiple versions (versioning system)
- A simple (but helpful) basis one can easily develop!
Ineracting elements
A dynamic view

- Data first!
- All elements interact
- Interface between software and languages (Python, Perl, C, ...)

Open Reproducible Research in Empirical Science
- Keep copy of the original
- Working version ready for import
- Rectangular and simple
- Keep track (metadata)
**Import working version ROpenOffice**

**Transform data**

```r
library(ROpenOffice)

tmp<-read.ods('../data/height-cone07.ods')

class(tmp)

[1] "list"

length(tmp)

[1] 6

head(tmp$Llaureolum)

species height cones
1     Ll     150    96
2     Ll     190   270
3     Ll     120    20
4     Ll     80     13
5     Ll    120     11
6     Ll    100     10
```
**Data transformation and visualization**

- Functions **melt** and **cast**: easy conversion between long and wide formats
- Data transformation: **reshape**, **plyr**
- Data mining

```r
dat <- do.call('rbind', tmp)
summary(dat)

#> species    height     cones
#> Ll:64    Min. : 20  Min. : 1.0
#> S :36    1st Qu.: 80 1st Qu.: 15.0
#> X :45    Median :120 Median : 32.5
#> R :47    Mean :119 Mean : 66.3
#> Lp:52    3rd Qu.:150 3rd Qu.: 73.8
#> C :34    Max. :400 Max. :1000.0

library(plyr)
tab <- ddply(dat, .(species), .fun = function(df) with(df, mean(cones^2/height)))
tab

#> species V1
#> 1   Ll 34.704
#> 2    S 206.706
#> 3    X 147.961
#> 4    R  81.245
#> 5   Lp  41.191
#> 6    C  3.548
```
Grammar of graphics

- **Mapping** graphical elements on variables
  - setting elements by user (shape, size, colour, ...)
- Multiple layer plots
- Flexible control on layers

The usual way

```r
plot(x=cones,y=height,data=data,
pch=as.numeric(factor(dat[, 'species'])))
```
Grammar of graphics

- **Mapping** graphical elements on variables
  - ≠ setting elements by user (shape, size, colour, ...)
- Multiple layer plots
- Flexible control on layers

```r
library(ggplot2)
qplot(data=dat, x=height, y=cones, colour=species)
```

![Diagram of a scatter plot with different colored dots representing different species.](image)
Grammar of graphics

- **Mapping** graphical elements on variables
  ≠ setting elements by user
  (shape, size, colour, ...)
- Multiple layer plots
- Flexible control on layers

Change y-scale to logarithmic:
Grammar of graphics

- **Mapping** graphical elements on variables
  - ≠ setting elements by user (shape, size, colour,...)
- Multiple layer plots
- Flexible control on layers

```r
p <- ggplot(data = dat, aes(x = height, y = cones, colour = species)) +
  geom_point() +
  geom_smooth(method = 'lm', aes(group = species), alpha = 0.5) +
  scale_y_log10(breaks = c(1,2,5,10,20,50,100,200,500,1000,2000)) +
  labs(x = 'Height (cm)', y = 'Number of cones', colour = 'Species')
```
**Grammar of graphics**

- **Mapping** graphical elements on variables
  - ≠ setting elements by user (shape, size, colour, ...)
- Multiple layer plots
- Flexible control on layers

```
p + facet_wrap(~species, ncol=2)
```
Grammar of graphics

- **Mapping** graphical elements on variables
  - ≠ setting elements by user (shape, size, colour, ...)
- Multiple layer plots
- Flexible control on layers

```r
p + facet_wrap(~species, ncol = 2, scales = 'free')
```
Integration. 1
Example in GIS world

- **Open tools**
  - Example in Geographic Information Systems (GIS)
  - QGIS, GRASS, SAGA for analysis
  - GDAL for format exchange and import
  - Interface packages with R: **rgdal**, **RSAGA**, **rGrass**,
  - Tools: **maptools**, **raster**, **rgeos**, **sp**, **spatial**,
Integration. II
Example in GIS world

```r
library(raster)

## read file names
RR<-dir(path='data/GIS/precip',
         full.names=TRUE)

## import files with GDAL and stack data
RR<-stack(RR)

## define Coordinate system
proj4string(RR)<-CRS('+init=epsg:2975')

## format data and sum
RR[RR>1e38]<-NA
RRm<-sum(RR)
```
The key to reproducibility !!

- Literate programming
  “Instead of imagining that our task is to instruct a computer what to do, let us concentrate on explaining to human beings what we want a computer to do.” (D. Knuth, 1984)
- Ordered mixture of **text** and **code** chunks
- Code chunks: produce figures and tables based on data
- Text chunks: explains the methods and results

Donald E. Knuth
**Production**

**Handling dynamic documents**

- **Weaving**: process dynamics documents to produce human-readable versions in various formats (PDF, HTML, ODF,...)

- **Tangling**: extract code chunks from dynamic document,

- Multiple alternatives

<table>
<thead>
<tr>
<th>Document format</th>
<th>(La)TeX, Markdown, ODF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weaver/Tangler</td>
<td><strong>Sweave</strong>, <strong>knitr</strong>, <strong>pander</strong>, <strong>rmarddown</strong></td>
</tr>
<tr>
<td>Software Format</td>
<td>R, Perl, Python, C</td>
</tr>
<tr>
<td>Format</td>
<td>PDF, HTML, ODF</td>
</tr>
</tbody>
</table>
The Tex approach

- Benefits from the power of TeX to produce complex documents
- Rather steep learning curve...

Text without code, may contain inline commands, to get the value of $\pi$ for instance: $\text{\textbackslash Sexpr\{pi\}}$

Code chunk producing a plot:
```r
set.seed(1213)  # for reproducibility
x <- cumsum(rnorm(100))
mean(x)  # mean of x
plot(x, type = 'l')  # Brownian motion
```

Text without code, may contain inline commands, to get the value of $\pi$ for instance: 3.1416

Code chunk producing a plot:
```r
set.seed(1213)  # for reproducibility
x <- cumsum(rnorm(100))
mean(x)  # mean of x
plot(x, type = 'l')  # Brownian motion
```
The markdown approach

- Simple mark-up language,
- Multiple output formats (PDF, HTML)
- Simple language for simple document (for now),

```r
library(pander)
pander(tab)
```

<table>
<thead>
<tr>
<th>species</th>
<th>V1</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>34.7</td>
</tr>
<tr>
<td>S</td>
<td>206.7</td>
</tr>
<tr>
<td>X</td>
<td>148</td>
</tr>
<tr>
<td>R</td>
<td>81.25</td>
</tr>
<tr>
<td>Lp</td>
<td>41.19</td>
</tr>
<tr>
<td>C</td>
<td>3.548</td>
</tr>
</tbody>
</table>
Thank you for your attention!

Questions, remarks ?