

Computo Journal Format

To be used as template for contribution to Computo

The Computo Team a friend

11/8/22

This is the abstract - Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur posuere vestibulum facilisis. Aenean pretium orci augue, quis lobortis libero accumsan eu. Nam mollis lorem sit amet pellentesque ullamcorper. Curabitur lobortis libero eget malesuada vestibulum. Nam nec nibh massa. Pellentesque porttitor cursus tellus. Mauris urna erat, rhoncus sed faucibus sit amet, venenatis eu ipsum.

Contents

About this document	2
Formatting	2
Basic markdown formatting	2
Mathematics	3
Mathematical formulae	3
Theorems and other amsthm-like environments	3
Code	4
R	4
Python	4
Figures	5
Tables	7
Markdown syntax	7
List-table filter	8
Table generated from code	8
Algorithms	9
Diagrams	10
Handling references	11
Bibliographic references	11
Other cross-references	11

To go further	12
Bibliography	12

About this document

This document provides a template based on the [quarto system](#) for contributions to **Computo** (Computo Team 2021). We show how **Python** (Perez, Granger, and Hunter 2011) or **R** (R Core Team 2020) code can be included.

Formatting

This section covers basic formatting guidelines. [Quarto](#) is a versatile formatting system for authoring HTML based on markdown, integrating \LaTeX and various code block interpreted either via Jupyter or Knitr (and thus deal with Python, R and many other languages). It relies on the [Pandoc Markdown](#) markup language.

i Note

We will only give some formatting elements. Authors can refer to the [Quarto web page](#) for a complete view of the formatting possibilities.

i Note

Quarto itself is a work-in-progress and a lot of bugs are constantly fixed or features added. As such, we recommend:

- searching any encountered issue with renders in [the upstream quarto dev repo](#)
- using a [version of quarto > 1.2](#)

To render/compile a document, run `quarto render`. A document will be generated that includes both content as well as the output of any embedded code chunks within the document:

```
quarto render content.qmd # will render to html
```

Basic markdown formatting

Bold text or *italic*

- This is a list

- With more elements
- It isn't numbered.

But we can also do a numbered list

1. This is my first item
2. This is my second item
3. This is my third item

Mathematics

Mathematical formulae

[L^AT_EX](#) code is natively supported¹, which makes it possible to use mathematical formulae: will render

$$f(x_1, \dots, x_n; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2\right)$$

It is also possible to cross-reference an equation, see Equation 1:

$$\begin{aligned} D_{x_N} &= \frac{1}{2} \begin{bmatrix} x_L^\top & x_N^\top \end{bmatrix} \begin{bmatrix} L_L & B \\ B^\top & L_N \end{bmatrix} \begin{bmatrix} x_L \\ x_N \end{bmatrix} \\ &= \frac{1}{2} (x_L^\top L_L x_L + 2x_N^\top B^\top x_L + x_N^\top L_N x_N), \end{aligned} \tag{1}$$

Theorems and other amsthm-like environments

Quarto includes a nice support for theorems, with predefined prefix labels for theorems, lemmas, proposition, etc. see [this page](#). Here is a simple example:

Theorem 0.1 (Strong law of large numbers). *The sample average converges almost surely to the expected value:*

$$\overline{X}_n \xrightarrow{a.s.} \mu \quad \text{when } n \rightarrow \infty.$$

See Theorem [0.1](#).

¹We use [katex](#) for this purpose.

Code

Quarto uses either Jupyter or knitr to render code chunks. This can be triggered in the yaml header, e.g., for Jupyter (should be installed on your computer) use

```
---
title: "My Document"
author "Jane Doe"
jupyter: python3
---
```

For knitr (R + knitr must be installed on your computer)

```
---
title: "My Document"
author "Jane Doe"
---
```

You can use Jupyter for Python code and more. And R + Knitr for if you want to mix R with Python (via the package reticulate Ushey, Allaire, and Tang (2020)).

R

R code (R Core Team 2020) chunks may be embedded as follows:

```
x <- rnorm(10)
```

Python

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length",
                 color="species",
                 marginal_y="violin", marginal_x="box",
                 trendline="ols", template="simple_white")

fig
```

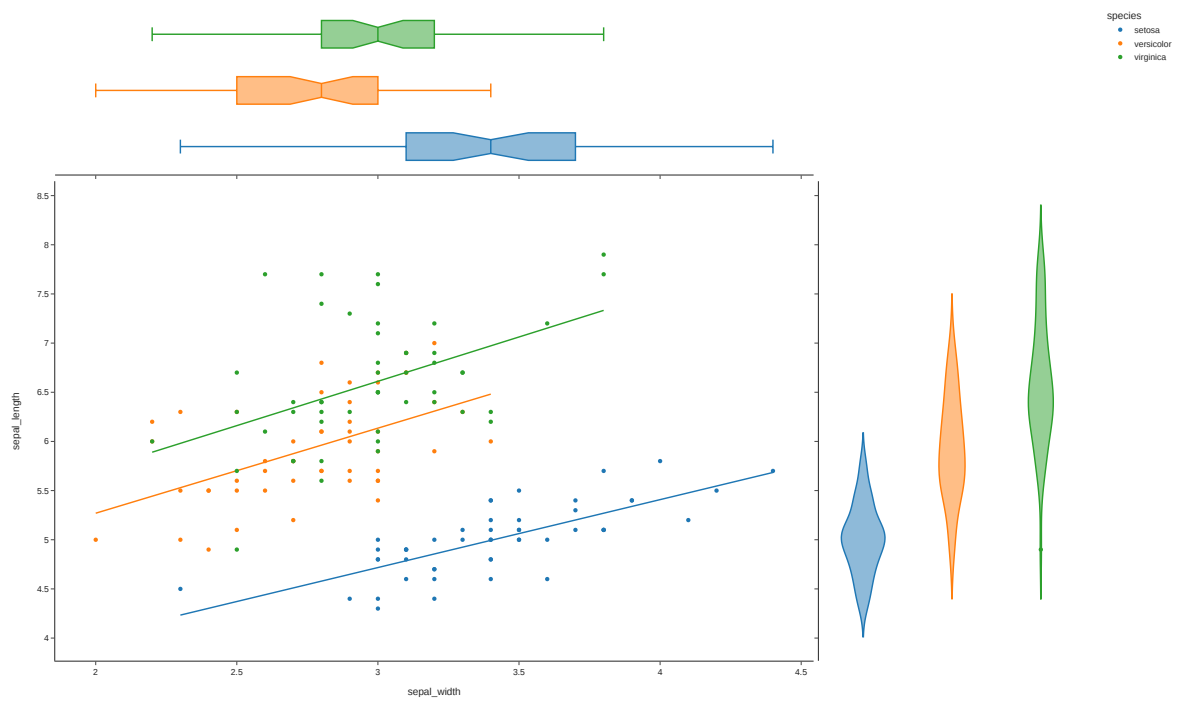


Figure 1: A simple python plotly example

Figures

Plots can be generated as follows and referenced. See plot [Figure 2](#):

```
library("ggplot2")
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth() + theme_bw()
p
```

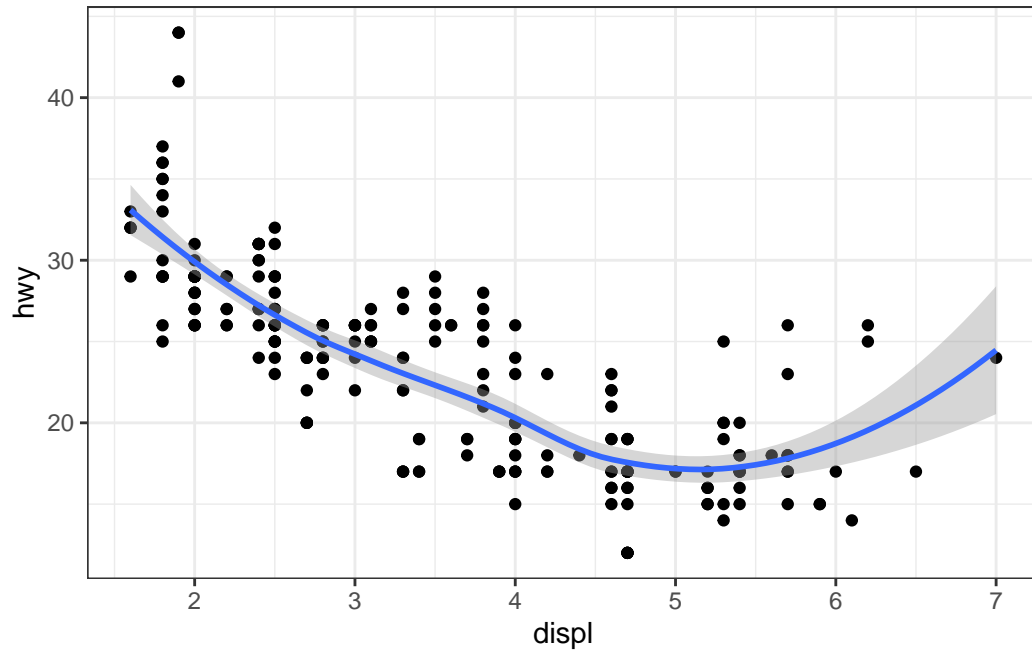


Figure 2: A simple ggplot example

Interactive plots may also be produced in the HTML output of the document²:

```
library("plotly")  
ggplotly(p)
```

²The pdfoutput is just a screenshot of the interactive plot from the html output

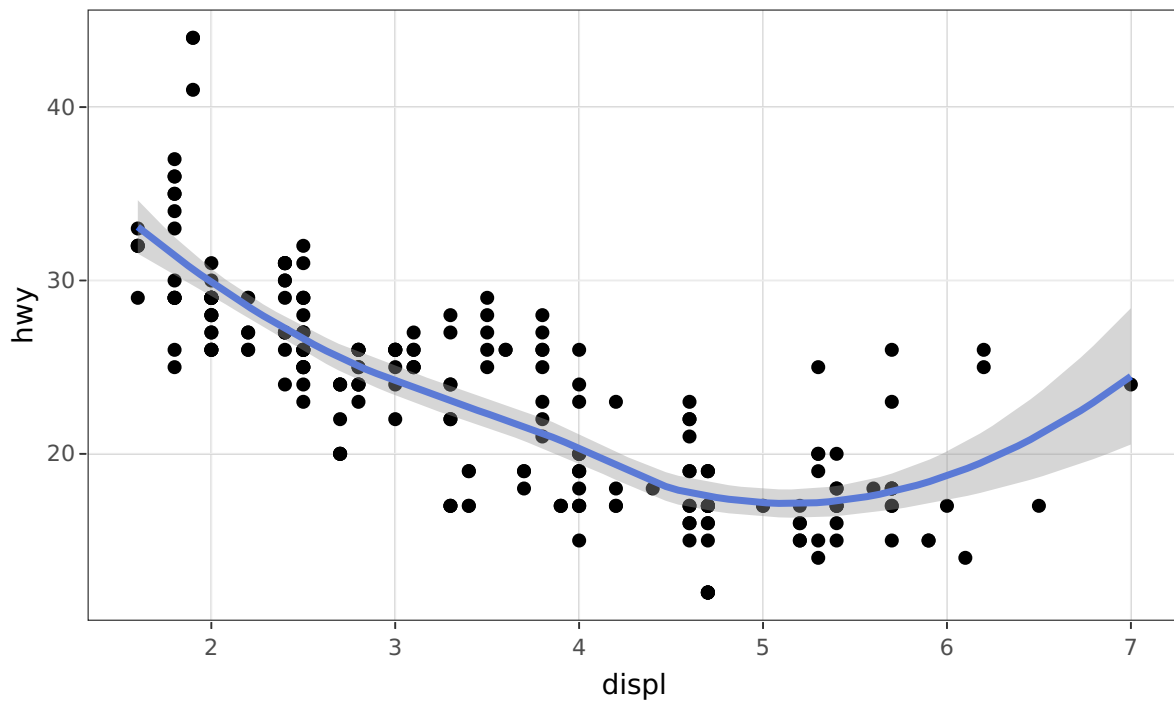


Figure 3: A simple ggplotly interactive example

It is also possible to create figures from static images:



Figure 4: SFdS logo (c.a. 2021)

Tables

Markdown syntax

Tables (with label: `@tbl-mylabel` renders Table 1) can be generated with markdown as follows

```
| Tables | Are | Cool |
|-----|:-----:|-----:|
| col 1 is | left-aligned | $1600 |
| col 2 is | centered | $12 |
```

```
| col 3 is | right-aligned |    $1 |
: my table caption {#tbl-mylabel}
```

Table 1: my table caption

Tables	Are	Cool
col 1 is	left-aligned	\$1600
col 2 is	centered	\$12
col 3 is	right-aligned	\$1

List-table filter

We also integrate the [list tables](#) filter from Pandoc, so that you may alternatively use this format , easier to write and maintain:

```
:::list-table
* - row 1, column 1
  - row 1, column 2
  - row 1, column 3

* - row 2, column 1
  -
  - row 2, column 3

* - row 3, column 1
  - row 3, column 2
:::
```

row 1, column 1	row 1, column 2	row 1, column 3
row 2, column 1		row 2, column 3
row 3, column 1	row 3, column 2	

Table generated from code

Table can also be generated by some code, for instance with `knitr` here:

```
knitr::kable(summary(cars), caption = "Table caption.")
```


Table 3: Table caption.

speed	dist
Min. : 4.0	Min. : 2.00
1st Qu.:12.0	1st Qu.: 26.00
Median :15.0	Median : 36.00
Mean :15.4	Mean : 42.98
3rd Qu.:19.0	3rd Qu.: 56.00
Max. :25.0	Max. :120.00

Algorithms

A solution to typeset pseudocode just like you would do with \LaTeX , yet with HTML output is to rely on the JavaScript [pseudocode.js](#). Your pseudocode is written inside a [Code Block](#) with the `pseudocode` class. Do not forget the class tag, that will trigger the rendering process of your pseudo-code. The result is as follows³:

```

```{.pseudocode}
% This quicksort algorithm is extracted from Chapter 7, Introduction
% to Algorithms (3rd edition)
\begin{algorithm}
\caption{Quicksort}
\begin{algorithmic}
\Procedure{Quicksort}{A, p, r}
 \If{$p < r$}
 \State $q = $ \Call{Partition}{A, p, r}
 \State \Call{Quicksort}{$A, p, q - 1$}
 \State \Call{Quicksort}{$A, q + 1, r$}
 \EndIf
\EndProcedure
\Procedure{Partition}{A, p, r}
 \State $x = A[r]$
 \State $i = p - 1$
 \For{$j = p, \dots, r - 1$}
 \If{$A[j] < x$}
 \State $i = i + 1$
 \State exchange
 $A[i]$ with $A[j]$
 \EndIf
 \EndFor
 \State exchange
 $A[i]$ with $A[r]$
\EndProcedure
\end{algorithmic}
\end{algorithm}
```

```

³For proper pdf rendering, use [Camel cased](#) names for all `algorithmic` keywords, not upper case ones like the examples in `pseudocode.js`'s documentation

```

\EndIf
\State exchange  $A[i]$  with  $A[r]$ 
\EndFor
\EndProcedure
\end{algorithmic}
\end{algorithm}
...

```

Algorithm 1 Quicksort

```

procedure QUICKSORT( $A, p, r$ )
  if  $p < r$  then
     $q = \text{PARTITION}(A, p, r)$ 
    QUICKSORT( $A, p, q - 1$ )
    QUICKSORT( $A, q + 1, r$ )
  procedure PARTITION( $A, p, r$ )
     $x = A[r]$ 
     $i = p - 1$ 
    for  $j = p, \dots, r - 1$  do
      if  $A[j] < x$  then
         $i = i + 1$ 
        exchange  $A[i]$  with  $A[j]$ 
    exchange  $A[i]$  with  $A[r]$ 

```

Diagrams

```

\usetikzlibrary{arrows}
\begin{tikzpicture}[node distance=2cm, auto,>=latex', thick, scale = 0.5]
\node (P) {$P$};
\node (B) [right of=P] {$B$};
\node (A) [below of=P] {$A$};
\node (C) [below of=B] {$C$};
\node (P1) [node distance=1.4cm, left of=P, above of=P] {$\hat{P}$};
\draw[->] (P) to node {$f$} (B);
\draw[->] (P) to node [swap] {$g$} (A);
\draw[->] (A) to node [swap] {$f$} (C);
\draw[->] (B) to node {$g$} (C);
\draw[->, bend right] (P1) to node [swap] {$\hat{g}$} (A);
\draw[->, bend left] (P1) to node {$\hat{f}$} (B);
\draw[->, dashed] (P1) to node {$k$} (P);

```

```
\end{tikzpicture}
```

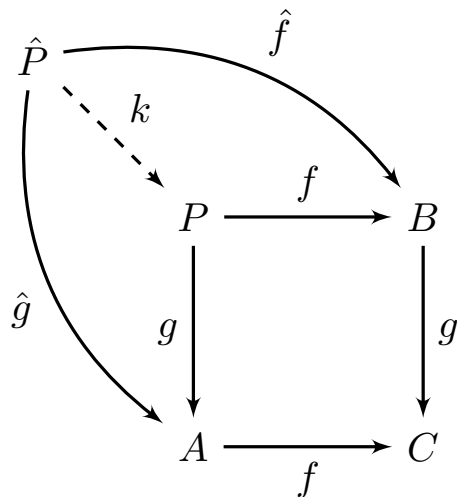


Figure 5: A simple tikz example

Handling references

Bibliographic references

References are displayed as footnotes using [BibTeX](#), e.g. `[@computo]` will be displayed as (Computo Team 2021), where `computo` is the bibtex key for this specific entry. The bibliographic information is automatically retrieved from the `.bib` file specified in the header of this document (here: `references.bib`).

Other cross-references

As already (partially) seen, Quarto includes a mechanism similar to the bibliographic references for sections, equations, theorems, figures, lists, etc. Have a look at [this page](#).

To go further

One last note

To go into more involved details, you can also simply check the source code of this document (button at the top), or have a look at the source of our [t-sne remake example](#).

Bibliography

- Computo Team. 2021. “Computo: Reproducible Computational/Algorithmic Contributions in Statistics and Machine Learning.” *Computo*.
- Perez, Fernando, Brian E Granger, and John D Hunter. 2011. “Python: An Ecosystem for Scientific Computing.” *Computing in Science & Engineering* 13 (2): 13–21.
- R Core Team. 2020. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.
- Ushey, Kevin, JJ Allaire, and Yuan Tang. 2020. *Reticulate: Interface to Python*. <https://github.com/rstudio/reticulate>.