RFigure Documentation

Release 3.1

Renaud Dessalles

Nov 20, 2020

Contents:

CHAPTER 1

RFigure version 3

1.1 Forewords

RFigure is a program that deals save in a specific file (whose extension is rfig3) the data and the code that is needed to produce a matplotlib figure. When creating a rfig3 file, is saved separately the data to display and the instructions executed to display.

This code is written by R. Dessalles (Grumpfou) for the most part (the only exceptions are the code in REditors that are adapted from example on the web). It is proposed under the license GNU General Public License v3.0 (unless some part of the code when specified otherwise like REditors, when written by somebody else). It is written in Python 3.6.6.

1.2 Install

1.2.1 Install on all versions

In a terminal, go in the same folder in which the sources have been installed, and use the following command lines:

```
pip install .
```

Alternately, you can explicitly save in you user directories using:

```
pip install --user .
```

1.2.2 Notes on Ubuntu

The library PyQt5

There is sometimes troubles installing PyQt5. You might have to download PtQt5 independently using the `apt-get` program before installing the RFigure library:

```
sudo apt-get install python-qt5
pip3 install .
```

The firectory of the rfig program into PATH

It is possible that by using the `--user` option, the script **rfig** is not saved in a directory which is in the environment variable "PATH". In that case, be sure to add the directory that contains *rfig** (usually `~/.local/bin/`) to "PATH" manually (see https://askubuntu.com/questions/799302/ubuntu-cant-find-an-executable-file-in-local-bin)

1.3 Usages

1.3.1 Quick Example with console

Open Python terminal and try:

```
import RFigure,numpy
X = numpy.arange(0,10,0.1)
Y = numpy.cos(X)
i = "plot(X,Y)" # the instructions
d = dict(X=X,Y=Y) # the data to display
c = "This is a test" # the commentaries associate with the figures
rf = RFigure.RFigureCore(d=d,i=i,c=c)
rf.show() # execute the instructions to be sure it works
rf.save(filepath='./Test.rfig3') # only save the rfig3 file
rf.save(filepath='./Test.rfig3',fig_type='pdf') # save the rfig3 file as well as the␣
↪pdf associated
```

Note that the data contained in `d` should be either `int`,`float`, `str`,`bool`,`numpy.ndarray`, `pandas.DataFrame` or data collections (`list`, `tuple` or `dict`) that contains supported types.

1.3.2 Example with graphical application

Once a rfig3 file is saved, one can use the graphical interface to modify the instructions using the script automatically installed **rfig**:

```
$ rfig ./Test.rfig3
```

In which case you have the following interface:

ExampleGui.png

1.3.3 Example using a magic function in Jupyter notebooks (IPython)

A Jupyter magic function exists to automatically save figures from Jupyter notebooks. It is available in the RFigure.RFigureMagics module.

To import the magic in the Jupyter notebook, use:

```
%load_ext RFigure.RFigureMagics
```

Then you can directly save RFigures using the `%%rfig` magic function: you specify the name of the file and it will create a RFigure according by taking the instructions of the cell as in the follow example:

```
In[1]:
> X = np.arange(0,10,.1)
> Y = np.cos(a)

In[2]:
> %%rfig_save "Test"
> # search the variables in the instructions, no comment and save in pdf
> plt.plot(X,Y)
```

The magic function tries to automatically detect what local variables need to be save (in the example, `X` and `Y`).

1.4 How the code is organized

- RFigureCore is a direct tool used in the scripts to create the files.
- RFigureGui is a gui interface used to modify a posteriori the instructions
- RFigureConfig: contains the header that will be execute before any instructions. It typically import numpy and matplotlib as the magic function %pyplot does in Jupyter/IPython
- RFigurePickle: handles the coding and decoding of information contained in rfig3 files. Works in the same way as the regular pickle library of python
- REditors: contains the Syntax Highlighters used to display the python code (for the instructions) and the markdown code (for the commentaries).
- RFigureMagics: contains the magic functions that can be used in Jupyter/QtConsole.
- RFigureSearchvar : contains function that analyse the code in order to determine the pertinent variable to save in the rfig file
- RFigureMisc: various functions

1.5 Miscelaneous

- By default, numpy and pyplot are already imported when executing the instructions (similar to the magic function %pyplot does in Jupyter/IPython)
- Local Header: When the program detects a file with the name `.RFigureHeaderLocal.py` in the same directory as the file, it adds it to the header (can be used to specify the font for all the rfig3 of the directory for instance).
- Format name: the RFigureCore method `formatName` (or push the button in the gui interface) will format the name of the figure as Figure_YYYYMMDD_foo.rfig3 (where YYYYMMDD stands for the current date).

- Shortcurts in the graphical interface:
  - `Ctrl+S` will save the file;
  - `Ctrl+M` or `Ctrl+Enter` will show the figure.
  - `Ctrl+/` will comment/uncomment the block
  - `Ctrl+↑` will move line(s) up
  - `Ctrl+↓` will move line(s) down
  - `Ctrl+Shift+D` will duplicate the line(s)
  - `Ctrl+Shift+K` will delete the line(s)
  - `Ctrl+J` reshape the selected line(s) in one single line
  - `Ctrl+F` find and replace
  - `F3` find next
  - `Shift+F3` find previous
- `# RFIG_start_instructions` command: (deprecated, use the magic function `%%rfig_save` instead) when the instructions in input contained at some point the line ..

  # RFIG_start_instructions

the program considers the instructions begin only at this point. It is useful when using it with a Jupyter notebook:

```
In[]:
> X = numpy.arange(0,10,0.1)
> d=dict(X=X)
> rf = RFigure3.RFigureCore(i=In[-1],d=d)
> rf.save(filepath='./Test.rfig3')
> # RFIG_start_instructions
> X=arange(0,10,0.1)
> plot(X,cos(X))
```

Will save figure whose instructions are the last lines of the cell.

- The dictionary `RFIG_savefig_kargs`: by creating a dictionary `RFIG_savefig_kargs` in the instricutions, we are able to manage the arguments than will be send to the function `matplotlib.figure.Figure.savefig` when saving the file. For instance, these instructions will save the figure with a black background: .. code-block:: guess

  > gca().patch.set_alpha(0) > X=arange(0,10,0.1) > plot(X,cos(X),color='w',lw=3) > RFIG_savefig_kargs = dict(facecolor='k')

1.6 FAQ

- Why there is a 'R' in front of the modules? Because my first name is **R**enaud
- Why do not use the default pickle library of Python ? Because there is so much problem of compatibilities between the different versions of Python on this module. I better control the data with my own Pickle (the inconvenient is that it only deal with basic types)
- The licenses of the logo and icons are available on here

1.6.1 Api Docs:

RFigure.**test**()
a test docstring

Returns *Nothing*

Unless mentionned otherwise, the code is written by R. Dessalles (Grumpfou). Published under license GNU General Public License v3.0. Part of the RFigure project.

class RFigure.RFigureCore.**RFigureCore**(*d=None*, *i=None*, *c=None*, *file_split='# RFIG_start_instructions'*, *filepath=None*)
This si the RFigureCore core class.

__init__(*d=None*, *i=None*, *c=None*, *file_split='# RFIG_start_instructions'*, *filepath=None*)
This function will save the figure into a propper way, in order to open it again.

Parameters

- **d** (*dict*) – dictionary that contain the variable useful to plot the figure.
- **i** (*str or func*) – instructions, string that contains the python code to create the figure. If it is a function, takes the source of the function.
- **c** (*str*) – commentaries where the user can describe the figure.
- **file_split** (*str*) – file_split it the string that will separate the instructions. What will be bollow the first instance of will be considered as the instructions. If *file_split* string is not encountered, keeps the whole instructions
- **filepath** (*str*) – the path to the file (useful for the local header and to directly save the file)
- **frame** (*int*)

Example

```
>>> import RFigure,numpy
... X = numpy.arange(0,10,0.1)
... Y = numpy.cos(X)
... i = "plot(X,Y)" # the instrucutions
... d = dict(X=X,Y=Y) # the data to display
... c = "This is a test" # the commentaries associate with the figures
... rf = RFigure.RFigureCore(d=d,i=i,c=c)
... rf.show() # execute the instructions to be sure it works
... rf.save(filepath='./Test.rfig3') # only save the rfig3 file
... rf.save(filepath='./Test.rfig3',fig_type='pdf') # save the rfig3 file as␣
↪well as the pdf associated
```

clean_instructions()
Ensure that the instruction are idented at the the first level.

execute(*print_errors=False*)
The method executes the instructions (no *show* at the end). Proceed in four steps: 1. executes the general header file *RFigure/RFigureConfig/RFigureHeader.py* 2. executes the local header file *./.RFigureHeader-Local.py* 3. updates the locals with the rfig variables *self.dict_variables* 4. executes the rfig instructions *self.instructions*

Parameters **print_errors** (*bool*) – if True, will print the errors rather than raise them (to avoid some troubles with PyQt5)

formatExt (*filepath=None*, *ext=None*, *replace_current=True*)
Changes/adds if necessary the extension to the filepath. Update the attribute *self.filepath* accordingly

Parameters

- **filepath** (*str*) – the filepath to format.
- **ext** (*str*) – the extension (the dot needs to be included). By default, takes *self.ext*.
- **replace_current** (*bool*) – if True, then replace the current *self.filepath* by the one obtained

Returns **filepath** (*str*) – the new filepath with the updated file extension

formatName (*filepath=None*, *replace_current=True*)
Format the filename under the format: "path/Figure_YYYYMMDD_foo.rfig3" where foo is the current filename. If the filpath is already under this format, do not change it. It updates the attribute *self.filepath*.

Parameters

- **filepath** (*str*) – the filepath to format, by default takes *self.filename*
- **replace_current** (*bool*) – if True, then replace the current *self.filepath* by the one obtained

Returns **filepath** (*str*) – the new filepath with the updated file name

Example

```
>>> rf = RFigureCore(filpath='./foo/faa.rfig3')
>>> rf.formatName()
"./foo/Figure_20181201_faa.rfig3"
>>> rf.formatName(filepath='./foo/fii.rfig3')
"./foo/Figure_20181201_fii.rfig3"
>>> rf.formatName(filepath='./foo/Figure_20181201_fuu.rfig3')
"./foo/Figure_20181201_fuu.rfig3"
```

(with each time, 20181201 corresponding to the curent date)

classmethod load (*filepath*)
Return a RFigureCore instance that had openned the rfigure.

Parameters **filepath** (*str*) – the path of the rfigure to load

Returns **rfig** (*RFigureCore*) – The RFigureCore instance created.

open (*filepath*)
Open the rfig file from filepath

Parameters **filepath** (*str*) – the path of the rfigure. Set the attribute self.filepath to this value

save (*filepath=None*, *fig_type=None*, *check_ext=True*)
Will save the figure in a rfig file.

Parameters

- **filepath** (*str*) – The filepath where to save the figure. Adds the extension if necessary. If None, search the attribute self.filepath (if self.filepath also None, raise an error). Is not None, set *self.filepath* to the new file path.
- **fig_type** (*None, str or list(str)*) –

  if not None, will save the figure in the corresponding format (if it is a list, will save in several formats). Should be in ['pdf', 'eps', 'png', 'svg'].
- **check_ext** (*bool*) – if True, adds if necessary the extension to filepath

Returns paths (*list of str*) – the paths of the files created/edited (i.e. the rfig file and the pdf/png/etc. that represents the figure)

savefig(*fig_path*, *fig_type='png'*)
Method that will save the figure with the corresponding extention.

Parameters

- **fig_path** (*str*) – The path of where to save the figure the figure.
- **fig_type** (*None, str or list(str)*) –

  if not None, will save the figure in the corresponding format (if it is a list, will save in several formats). Should be in ['pdf', 'eps', 'png', 'svg'].

Returns paths (*list of str*) – the paths of the files created/edited (i.e. the rfig file and the pdf/png/etc. that represents the figure)

show(*print_errors=False*)
Method that execute the code instructions and adds the matplotlib.pyplot.show() statement at the end.

Parameters print_errors (*bool*) – if True will print the errors rather than raise them (to avoid some troubles with PyQt5)

static update(*fig_path*, *d=None*, *i=None*, *c=None*, *mode='append'*, *fig_type=None*)
Update the dict_variables of an already existing file:

Parameters

- **fig_path** (*str*) – The rfig2 file to update.
- **d** (*dict*) – The dict to update with.
- **i** (*str*) – The instricution to update with.
- **c** (*str*) – The commentary to update with.
- **mode** (*str in ['append','replace']*) –

  if mode=='append': will update the dict and add instructions and commentaries to the allready instructions and commentaries

  if mode=='replace': will replace the dict, instructions and commentaries

Returns rfig (*RFigureCore instance*) – the RFigureCore instance with updated dict_variables

class RFigure.RFigureMagics.**MyArgumentParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*)

exception ArgumentParserError

error(*message: string*)
Prints a usage message incorporating the message to stderr and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

class RFigure.RFigureMagics.**RFigureMagics**(*shell=None*, ***kwargs*)

rfig_curdate (*line*)

Magic function that assign a different current date that will be used when formatting the names. Equivalent to change the value of *RFigureCore.CURDATE*.

Examples

(in IPython/Jupyter)

```
>>> %rfig_save --format_name Test
... # Will save in the file `Figure_YYYYMMDD_Test.rfig3` where YYYYMMDD
... # stands for the current date.
... plt.plot(np.arange(10),np.arange(10))
```

```
>>> %rfig_crudate 19990909
... # set the curent date to `'19990909'`
```

```
>>> %rfig_save --format_name Test
... # Will save in the file `Figure_19990909_Test.rfig3`
... plt.plot(np.arange(10),np.arange(10))
```

rfig_format (*line*)

Magic function that assign a different format that will be used when formatting the names. Equivalent to change the value of *RFigureCore.FORMAT*.

Examples

(in IPython/Jupyter)

```
>>> %rfig_save --format_name Test
... # Will save in the file `Figure_YYYYMMDD_Test.rfig3` where YYYYMMDD
... # stands for the current date.
... plt.plot(np.arange(10),np.arange(10))
```

```
>>> %rfig_format Fig-%Y-%m-%d_%s
... # set the curent date to `'Fig-%Y-%m-%d_%s'`
```

```
>>> %rfig_save --format_name Test
... # Will save in the file `Fig_YYYY-MM-DD_Test.rfig3`
... plt.plot(np.arange(10),np.arange(10))
```

rfig_list_var (*line*, *cell*)

usage: %%rfig_list_var [–help] [dict_variable]

Detects the variables in the code of the cell.

positional arguments:

dict_variable the variable name of the dictionary in which will be stored the variables detected in the cell. If none is given, only prints its keys.

optional arguments:

--help, -h show this help message and exit

Examples

(in IPython/Jupyter)

```
>>> a = np.arange(0,10,.1)
... b = np.cos(a)
```

```
>>> %%rfig_list_var
... plot(a,b)
We determined the RFigure variables to be: `a`, `b`
```

```
>>> %%rfig_list_var a_dict
... plot(a,b)
We determined the RFigure variables to be: `a`, `b`
```

```
>>> print(a_dict.keys())
dict_keys(['a','b'])
```

rfig_load(*line*)

usage: %%rfig_load [–help] [-d D] [-i I] [-c C] [filepath]

Will load a RFigure in the Jupyter notebook.

positional arguments: filepath Path of the file to open.

optional arguments:

--help, -h	show this help message and exit
-d D	the variable name of the dictionary in which will be stored the variables of the RFigure. If none is given, import in the notebook locals.
-i I	the variable name of the string in which will be stored the instructions of the RFigure. If none is given, create a new cell filled with the instructions. Also checks if the magic *%pylab* has been executed. If not, it adds the command *%pylab* at the begining of the instructions
-c C	the variable name of the string in which will be stored the commentaries of the RFigure.

Examples

(in IPython/Jupyter)

```
>>> # Opens the RFigure Test.rfig3:
... # 1) import its variables in the notebook locals
... # 2) checks that %pylab is imported (if not addd it)
... # 3) create a new cell with the instructions
... %rfig_load "Test.rfig3"
```

```
>>> # Opens the RFigure Test.rfig3:
... # 1) stores the variables in `ddd`
... # 2) stores the instructions in `iii`
... # 3) stores the commentaries in `ccc`
... %rfig_load -i iii -c ccc -d ddd "Test.rfig3"
```

rfig_save (*line*, *cell*)

usage: %%rfig_save [–help] [–format_name] [-d D] [-c C] [–fig_type FIG_TYPE] [filepath]

Will save a RFigure, whose instructions are the code written in the remaining of the cell.

positional arguments: filepath Path of the file.

optional arguments:

--help, -h	show this help message and exit
--format_name, -fn	Format the name of the file as Figure_YYYYMMDD_foo where YYYYMMDD stands for the date. *foo* will be the file names. If the file name is already under this format, do notiong.
-d D	Dictionary of the locals in the rfigure file. If not specified, guess from the instructions.
-c C	Comments associated to the file

–fig_type FIG_TYPE, -ft FIG_TYPE extension of the figure, should be in ['pdf', 'eps', 'png', 'svg']

Examples

(in IPython/Jupyter)

```
>>> import numpy as np
... a = np.arange(0,10,.1)
... b = np.cos(a)
... comment = "A comment"
... diction = {'a':a,'b':1/a}
```

```
>>> %%rfig_save Test
... # search the variables in the instructions, no comment and save in pdf
... plt.plot(a,b)
```

```
>>> %%rfig_save -c comment Test
... # search the variables in the instructions, with a comment and save in pdf
... plt.plot(a,b)
```

```
>>> %%rfig_save --fig_type png Test
... # search the variables in the instructions, no comment and save in png
... plt.plot(a,b)
```

```
>>> %%rfig_save -d diction Test
... # specify other variables, no comment, save in pdf
... plt.plot(a,b)
```

```
>>> %%rfig_save --format_name Test
... # search the variables in the instructions, format the filename
... plt.plot(a,b)
```

RFigure.RFigureMisc.**RTextWrap** (*text*, *nb=79*, *sep='\n'*, *begin=''*)

Will return the text properly wrapped

Parameters

- **text** (*str*) – the textg to wrap
- **nb** (*int*) – the limit at which we wrap the text
- **sep** (*str*) – the char that seperate the paragraphs
- **begin** (*str*) – the char that should begin each line

Returns **wrapped_text** (*str*) – the wrapped version of the text

RFigure.RFigureMisc.**decoDocFormatting**(**args*)
Decorator that will format the documentation of the function by using the formatting like *doc%args*

Parameters **args** (*objects*) – the argument to put in the formatting

Returns **func** (*fonction*) – the function with the correct documentation

RFigure.RFigureMisc.**decoSetDoc**(*doc*)

Function dedicated to be used as a decorator that set the documentation *doc* to the function *func*.

Parameters

- **func** (*fonction*) – the function to set the documentation to
- **doc** (*str*) – the documentation to attach to the function

Returns **func** (*fonction*) – the function with the given documentation

RFigure.RFigurePickle.**isAuthorized**(*v*)
Function that determines if the object is authorised to be saved in a rpickle file. It checks recusively (if a list or a dict) that the object can indeed be represented in a rpickle file.

Parameters **v** (*any object*) – the object to check

Returns **res** (*bool*) – the status of the object

RFigure.RFigurePickle.**load**(*filepath*)
Load the RPickle file of filepath

Parameters **filepath** (*path to the file to load*)

Returns

- **objets** (*list or dict*) – the saved objects
- **commentaries** (*str*) – the commentaries stored in the file
- **version** (*str*) – the version string stored in the file

RFigure.RFigurePickle.**object_to_txt**(*objects*, *imports*)
Function that will transform any object in its string version

Parameters

- **objects** (*in [<class 'int'>, <class 'list'>, <class 'bool'>, <class 'dict'>, <class 'float'>, <class 'str'>, <class 'tuple'>, <class 'numpy.ndarray'>, <class 'numpy.dtype'>, <class 'numpy.int8'>, <class 'numpy.int16'>, <class 'numpy.int32'>, <class 'numpy.int64'>, <class 'numpy.float16'>, <class 'numpy.float32'>, <class 'numpy.float64'>, <class 'complex'>, <class 'numpy.complex64'>, <class 'numpy.complex128'>, <class 'numpy.datetime64'>, <class 'numpy.timedelta64'>, <class 'numpy.bool_'>]*) – the object to represent as a string
- **imports** (*list of str*) – list of all the libraries needed to define the object (will be updated insitus)

- **Returns**
- **res** (*str*) – the string representation of the object

RFigure.RFigurePickle.**save**(*objects*, *filepath*, *commentaries=''*, *version=None*, *ext='.rpk2'*)

Function that will save the given objects in a rpickle file.

Parameters

- **objects** (*list of dict*) – the object to save (can be a list or a dict if needed).
- **filepath** (*str*) – the place where to save the sile, if has no extension, it will take the one of the class.
- **commentaries** (*str*) – string that contains commentaries about the thing
- **version** (*str*) – the version of the software that should use the pickle.
- **ext** (*str*) – the file extension (the dot should be included)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

Symbols

C

D

E

F

I

L

M

O

R

S

T

U