LCAV Bachelor project - year 2016 - report

June 10, 2016

Theme : **PostScript in Ipython Notebook**
Supervisor : PRANDONI Paolo - Student : FOREL Duncan

1 Description of the project

An IPython Notebook is a "web-based interactive computational environment where you can combine code execution, text, mathematics, plots and rich media into a single document'". IPython notebooks are extremely versatile: you can run them locally, and all the code samples will execute on your PC; or you can'"freeze" them and make them available as HTML pages, so that people can browse them. Finally, a server can be set up so that viewers can run the notebook remotely and combine the advantages of web-based browsing with the interactivity of the Python code.

IPython notebooks have become particularly popular because they natively integrate LaTeX and graphics support, so that they can be used to produce high-quality web pages that include math and figures. In high-quality scientific publishing, however, the gold standard for figure generation is the PostScript language. When writing in TeX or LaTeX, many packages exist (such as PsTricks) that help us design beautiful figures and plots programmatically, ie. as chunks of LaTeX and PostScript code. The document is then compiled to DVI, converted to PostScript using the dvips program and the code will be interpreted by the PostScript viewer (or printer) to render the graphics at arbitrary scale with no loss of resolution.

The goal of this project is to add PostScript functionality to IPython notebooks by allowing for PostScript-generating LaTeX/macros to execute and render within the notebook environment.

2 Workflow of the project

The project was separated in two parts : comprehension and design of the module (mostly at the beginning of the project) and then development, test and documentation (mostly at the end of the project).

2.1 Comprehension of the problem

This project can be split in two main parts : conversion of a PostScript file to a format accepted by the notebook and creating the magics with this objective.

The conversion from LaTeX to PostScript is directly handled by the LaTeX distributions, with the commands "latex" and "dvips". I had to treat the case of the PostScript files. These ones don't contain the pixels of the images, only the commands used by the PS-printer to draw the figure. Meanwhile, the notebook can natively display the following formats :

- HTML
- JSON
- PNG (the format of the files converted by this module)
- JPEG
- SVG
- LaTeX (the notebook uses MathJax to render LaTeX code, it is consequently incomplet, working mainly for the mathematical formulas).

The conversion format had to be an image, as the goal of the module is to render figures written in PostScript. I chose the PNG for the compression. Consequently, I needed a way to convert PostScript to PNG. GhostScript is a great software allowing to read and convert PS files. It contains a command-line function ps2pdf converting a PS file to a PDF file. However, the PDF aren't supported by the notebook. I had then to convert from PDF to PNG.

After some researchs, I found another free software allowing this kind of conversion : ImageMagick (website : http://www.imagemagick.org/script/index.php). Moreover, it allows to directly convert from PostScript to PNG if GhostScript is installed (ImageMagick delegates the reading of the PS files to GhostScript). With this basis, I created a script converting a given LaTeX/PostScript file to PNG. Some API exists for ImageMagick, however the software is designed for a command-line usage and these API need to be installed in complement of the software himself (in addition of packages in the case of PythonMagick, the python API for ImageMagick). I consequently chose to stay with the basic version of the software, the functionnalities provided being sufficient for this module.

To prevent the conflict between a command from the OS on Windows and ImageMagick, the installation has to be completed with the renaming of a file from the software on this OS. This forced the distinction between the Linux/MacOS and Windows.

2.2 Interrogations about internal structure

After the first step done, I had to write magic functions for the notebook. I browsed the IPython and Jupyter documentation in order to find the way to do it. The most useful page found is : http://ipython.readthedocs.io/en/stable/config/extensions/index.htm , containing almost everything on the extensions and a link on how to create magics.

The main function is load_ipython_extension, allowing to modify the behaviour of the notebook by adding shortcut, magics or variables.

A question was about overriding the file created in memory. Even if the notebook doesn't need it permanently, I didn't know if I should simply replace the files created. I understood then that doing the work this way would cause easily conflict, as I didn't delete the files used. As example, a single-page PS file would generate a single image, and then generating images from a multi-page PS file would display the same image from the previous compilation forever. This effect is still present for the commands specifying a path for the PNG files, but it is documented.

2.3 Development

The development went from the most basic functionnalities, packed in the code in "Utility functions" to the high-level functionnalities, i.e the magics.

Each level requires the lower level to work and is documented both in the code using docstrings and in the general documentation. The basic module only contained magics to display code or file and an utility magic used mainly for the help. The error management is present, even if minimal in the case of errors in the PostScript to PNG conversion. The main usage is supposed to be LaTeX code display.

Then, as suggested by Pr. Prandoni, I added the possibility to specify a path to a PNG file for each command. If an image is found at this emplacement, it is displayed, otherwise the result of the compilation is saved here. This functionnaly was accompanied by a command forcing the compilation, even if a file is present at the given emplacement. The goal was to be able to share a notebook with people who don't have LaTeX or the dependencies of this module. To store the command for the forced compilation was the most complicated part of the development. I wanted to be as simple for the user as possible and didn't want to force him to add an option every time he wanted to force the compilation, nor create a variable at the beginning of the notebook. The problem was finally solved using an option for the %PSViewer magic.

2.4 Documentation

The documentation was built using Sphinx, a tool designed to generate high-quality documentation for programs. This documentation contains informations about how to install the module, how it works (both for the user and a programmer wanting to enhance the functionnalities), some examples and common errors.

Some of them, notably forcing the compilation of a file multiple times concluding in the display of an unexpected result, can happen easily. Therefore, it was mandatory to explain what the problem was and how to avoid it.

The documentation is available both as pdf and as html, allowing to create an online documentation. Both are joined to this report. The goal is to publish this module on github and make it available for anyone wanting to use it, contibuting in this way to the development of the ipython notebook.

2.5 Possible enhancements

- Allow the user to specify a resolution for the output file.
- Add a better management of the conversion errors.
- Allow the user to specify an output format.
- Display the options ignored by the magics (actually, an incorrect option is skipped).

3 Code of the module

The following code is included in the file PSViewer.py. A variante exists for Windows, where CONVERT_CMD as a value of "convert_img".

```
In [ ]: import sys
        import os
        import shutil

        from IPython.display import (display, display_png, Image, FileLink)


        #Constants
        VERSION_PATH = "version.txt"
        PS_FILE = "postScript.ps"
        LATEX_FILE = "LaTeX.tex"
        GENERIC_FOLDER = "Figure"
        LATEX_CMD = "latex"
        DVIPS_CMD = "dvips"
        CONVERT_CMD = "convert"
        WRAPPER = r"""
        \documentclass{minimal}
        \begin{document}
        @REPLACE@
        \end{document}
        """

        #Global Variable
        forceCompile = False


        #--------------------------
        #    Utility functions
        #--------------------------


        def clearVersion() :
            """Utility function used to clear the documents created by the compilation.
            It retrieves the current index for the compilation and remove all the previous folders.
            If the value stored for the index is incorrect, nothing is done.
```

```
    It also removes the file containing the current index.
    """
    if os.path.exists(VERSION_PATH) :
        f = open(VERSION_PATH, "r")
        value = str(f.read())
        f.close()
        if value.isdigit() :
            for i in range(int(value)) :
                shutil.rmtree("%s%i" % (GENERIC_FOLDER, i), True)
        os.remove(VERSION_PATH)


def loadVersion() :
    """Utility function loading the current index for the folder storing a compiled document.
    If the document storing the index is missing or the value contained is incorrect
    (typically, if it is a string) the index will be set to 0.
    Then, the index incremented is stored in the version file and the folder
    corresponding to the index is created.
    """
    value = ""
    if os.path.exists(VERSION_PATH) :
        f = open(VERSION_PATH, "r")
        value = f.read()
        f.close()
    else :
         f = open(VERSION_PATH, "w")
         f.write(str(0))
         f.close
    version = 0
    if value.isdigit() :
        version = int(value)
    f = open(VERSION_PATH, "w")
    f.write("%i" % (version+1))
    f.close()
    os.makedirs("%s%i" % (GENERIC_FOLDER, version))
    return version


def writeTeXFile(data, version=0) :
    """Write the data given in a TeX file corresponding to the current index.

    Keyword arguments:
    data -- the string containing the text to write in the document.
    version -- the index of the folder to store the file in (default 0).
    """
    path = os.path.join("%s%i" % (GENERIC_FOLDER, version), LATEX_FILE)
    f = open(path, 'w')
    f.write(data)
    f.close()
    return path


def writePSFile(data, version=0) :
    """Write the data given in a PS file corresponding to the current index.
```

```
    Keyword arguments:
    data -- the string containing the text to write in the document.
    version -- the index of the folder to store the file in (default 0).
    """
    path = os.path.join("%s%i" % (GENERIC_FOLDER, version), PS_FILE)
    f = open(path, 'w')
    f.write(data)
    f.close()
    return path


def convertLaTeXFile(path, doubleCompilation=False) :
    """Convert the TeX file at the specified path to a PS file,
    using the latex and dvips commands. If an error is encountered
    during the conversion, the process is stopped and an error message
    is displayed. Caution : this error message uses IPython functionalities.
    The double compilation is supported if needed. The intermediate files
    from the conversion are deleted.

    The process return the path to the PS file if the compilation was successful
    and None otherwise.

    Keyword arguments:
    path -- the path of the TeX file.
    doubleCompilation -- boolean containing if a double compilation should be done
        (default False)
    """
    here = os.getcwd()
    filename = os.path.basename(path)[:-4]
    os.chdir(os.path.dirname(path))
    ret = os.system("%s -quiet %s" % (LATEX_CMD, filename + ".tex"))
    if ret != 0 :
        os.chdir(here)
        print("An error occurred during the conversion, please check the code."
              "Details of the error in the following file :")
        display(FileLink(os.path.join(os.path.join(os.path.dirname(path), filename + ".log"))))
        return None
    if doubleCompilation :
        ret += os.system("%s -quiet %s" % (LATEX_CMD, filename + ".tex"))
    ret += os.system("%s -q %s -o %s" % (DVIPS_CMD, filename + ".dvi", filename + ".ps"))
    os.remove(filename + ".dvi")
    os.remove(filename + ".aux")
    os.remove(filename + ".log")
    os.chdir(here)
    if ret == 0 :
        return path[:-4] + ".ps"
    else :
        print("Something failed in the dvips conversion. Unexpected behaviour.")
        return None


def convertPSFile(path, pngPath=None) :
    """Convert the PS file at the specified path to one or multiple PNG files,
```

```
        one PNG image per page. The path to save the PNG can be specified, is must
        be a file and not a directory. The conversion uses a command from ImageMagick,
        convert_img (it is a renamed version of convert). If no PNG path is specified,
        the name and the emplacement of the postScript file will be used instead.
        If a path different from a PNG file is given, a missing extension will be assumed.
        This function returns the path of the PNG image (even if multiple images will have
        distinct paths).

        Keyword arguments:
        path : the path of the PS file.
        pngPath : the path where to save the PNG file(s) (default None)
        """
        if pngPath is None :
            pngPath = path[:-3] + ".png"
        else :
            if len(pngPath)<4 or (pngPath[-4:] != ".png" and pngPath[-4:] != ".PNG"):
                print("The path given was not a png file. Assuming missing extension.")
                pngPath = pngPath + ".png"
        os.system("%s %s %s" % (CONVERT_CMD, path, pngPath))
        return pngPath


    def findPNGFiles(pngPath, expectations=False) :
        """Utility function to retrieve one or multiple PNG files from an emplacement.
        The given path can design a file or the prefix of multiple PNG files
        (in the format prefix-x.png, where x is an integer). If a file is found, the prefix case
        won't be evaluated.
        All the path found are stored in a table and returned by the function.
        If the table is empty and it was expected to find results, an error message is displayed.

        Keyword arguments:
        pngPath -- the file's emplacement or the common prefix.
        expectations -- if the function displays an error message in case of empty result
        (default False).
        """
        result = []
        if len(pngPath)<4 or pngPath[-4:] != ".png" :
            print("The path given was not a png file. Assuming missing extension.")
            pngPath = pngPath + ".png"
        if os.path.exists(pngPath) :
            result.append(pngPath)
        else :
            pngPath = pngPath[:-4] + "-"
            stillSeeking = True
            index = 0
            while(stillSeeking) :
                currentPath = "%s%i.png" % (pngPath, index)
                if os.path.exists(currentPath) :
                    result.append(currentPath)
                    index = index + 1
                else :
                    stillSeeking = False
        if expectations and len(result)<1 :
            print("Conversion in PNG file didn't work, please check the postScript"
```

```
                "code you provided.")
    return result


#--------------------------
#    Mid-level functions
#--------------------------


def displayPScode(data, pngPath=None) :
    """This function loads and displays the PNG file specified in argument,
    unless the forced compilation is enabled. If no PNG file can be displayed
    or if the compilation is forced, it writes the given text in an PS file and
    calls displayPSFile on it, transmitting the PNG path.

    Keyword arguments:
    data -- the string containing the PS code.
    pngPath -- the path where to fetch/save the PNG file (default None).
    """
    if pngPath is not None and not forceCompile :
        if checkPNGFiles(pngPath) :
            return None
    folder = 0
    if pngPath is None :
        folder = loadVersion()
    path = writePSFile(data, folder)
    displayPSFile(path, pngPath)
    return None


def displayPSFile(path, pngPath=None) :
    """This function loads and displays the PNG file specified in argument,
    unless the forced compilation is enabled. If no PNG file can be displayed
    or if the compilation is forced, it converts the PS file given in argument
    and displays the images resulting of the conversion.

    Keyword arguments:
    path -- the path of the PS file to convert.
    pngPath -- the path where to fetch/save the PNG file (default None).
    """
    if pngPath is not None and not forceCompile :
        if checkPNGFiles(pngPath) :
            return None
    result = convertPSFile(path, pngPath)
    checkPNGFiles(result, True)
    return None


def displayLatexCode(data, pngPath=None, doubleCompilation=False) :
    """This function loads and displays the PNG file specified in argument,
    unless the forced compilation is enabled. If no PNG file can be displayed
    or if the compilation is forced, it writes the given text in an TeX file and
    calls displayLatexFile on it, transmitting the PNG path and the indication for
    the double compilation.
```

```
    Keyword arguments:
    data -- the string containing the LaTeX code.
    pngPath -- the path where to fetch/save the PNG file (default None).
    doubleCompilation -- whether the compilation should be executed twice (default False)
    """
    if pngPath is not None and not forceCompile :
        if checkPNGFiles(pngPath) :
            return None
    folder = 0
    if pngPath is None :
        folder = loadVersion()
    path = writeTeXFile(data, folder)
    displayLatexFile(path, pngPath, doubleCompilation)
    return None


def displayLatexFile(path, pngPath=None, doubleCompilation=False) :
    """This function loads and displays the PNG file specified in argument,
    unless the forced compilation is enabled. If no PNG file can be displayed
    or if the compilation is forced, it converts the LaTeX file given in argument,
    using the double compilation if specified, generating a PS file. The result is then
    converted and displayed using displayPSFile.

    Keyword arguments:
    path -- the path of the LaTeX file to convert.
    pngPath -- the path where to fetch/save the PNG file (default None).
    doubleCompilation -- whether the compilation should be executed twice (default False)
    """
    if pngPath is not None and not forceCompile :
        if checkPNGFiles(pngPath) :
            return None
    result = convertLaTeXFile(path, doubleCompilation)
    if result is not None :
        displayPSFile(result, pngPath)
    return None


def checkPNGFiles(path, expectations=False) :
    """Retrieve and display the PNG images corresponding to the specified path.
    If no file is found can display an error message.
    The algorithm used for retrieving files is given in findPNGFiles.

    Keyword arguments:
    path -- the path of the PNG file or the common prefix of multiple files.
    expectations -- if the function displays an error message in case of empty result
    (default False).
    """
    result = findPNGFiles(path, expectations)
    if len(result) > 0 :
        for elem in result :
            display_png(Image(filename=elem))
        return True
    else :
```

```
        return False


#--------------------------
#    Magics functions
#--------------------------


def PSFile(line) :
    """The function defining the %PSFile magic.
    The expected arguments are the path to the PS file to display and optionally
    the path of the PNG file corresponding (if not found or compilation forced, it
    will be the converted file's path).
    In case of incorrect use, this function displays an error message.

    Keyword arguments:
    line -- the line of text after the magic's invocation, of the form path [path]
    """
    paths = line.split(" ")
    if len(paths) > 2 or len(paths) == 0 :
        print("Incorrect use of the magic. Please see '%PSViewer -help' for help")
    else :
        if len(paths) == 1 :
            displayPSFile(paths[0])
        else :
            displayPSFile(paths[0], paths[1])


def LatexFile(line) :
    """The function defining the %LatexFile magic.
    The expected arguments (the line) are the path to the Latex file to display and
    optionally the path of the PNG file corresponding (if not found or compilation
    forced, it will be the converted file's path).
    In case of incorrect use, this function displays an error message.

    Keyword arguments:
    line -- the line of text after the magic's invocation, of the form path [path]
    """
    paths = line.split(" ")
    if len(paths) > 2 or len(paths) == 0 :
        print("Incorrect use of the magic. Please see '%PSViewer -help' for help")
    else :
        if len(paths) == 1 :
            displayLatexFile(paths[0])
        else :
            displayLatexFile(paths[0], paths[1])


def PSCode(line, cell) :
    """The function defining the %%PSCode magic.
    The line argument contains the options of the magic. If a path is specified as
    first option, the magic will try to retrieve PNG images from this emplacement
    and display them without any compilation. If the compilation is forced or if no
    file are found, the following options will be evaluated and the conversion of the
```

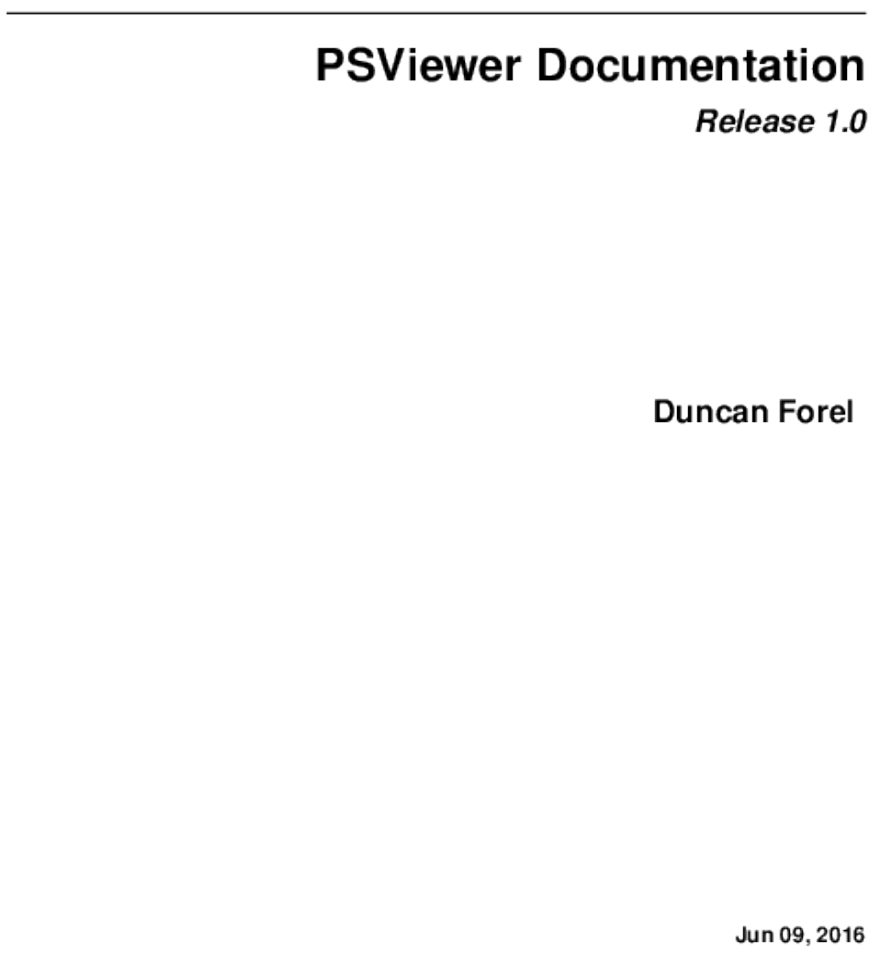
```
        cell's code will be execute.
        The current options are :
        -noSave which allow the override of the intermediate files. Caution : this can cause
        unexpected behaviour in some cases.
        Any other command will be ignored.

        Keyword arguments:
        line -- the line of text after the magic's invocation, of the form "path [option*]"
        cell -- the content of the cell, expected to be the postScript code.
        """
        if len(line) > 1 :
            commands = line.split(" ")
            path = None
            if commands[0][0] != "-" :
                path = commands[0]
            if "-noSave" in commands and pngPath is None :
                path = os.path.join(GENERIC_FOLDER + "0", "display.png")
            displayPScode(cell, path)
        else :
            displayPScode(cell)

    def LatexCode(line, cell) :
        """The function defining the %%LatexCode magic.
        The line argument contains the options of the magic. If a path is specified as
        first option, the magic will try to retrieve PNG images from this emplacement
        and display them without any compilation. If the compilation is forced or if no
        file are found, the following options will be evaluated and the conversion of the
        cell's code will be execute.
        The current options are :
        -noSave which allow the override of the intermediate files. Caution : this can cause
        unexpected behaviour in some cases.
        -noImport generates the documentclass, begin{document} and end{document} of the
        file automatically.
        -doubleCompilation indicates that a double compilation is required. This option is
        useful in case of internal links.
        Any other command will be ignored.

        Keyword arguments:
        line -- the line of text after the magic's invocation, of the form "path [option*]"
        cell -- the content of the cell, expected to be the LaTeX code.
        """
        if len(line) < 1 :
            displayLatexCode(cell)
        else :
            commands = line.split(" ")
            pngPath = None
            if commands[0][0] != "-" :
                pngPath = commands[0]
            if "-noImport" in commands :
                cell = WRAPPER.replace("@REPLACE@", cell)
            noSave = None
            doubleCompilation = False
            if "-noSave" in commands and pngPath is None :
                noSave = os.path.join(GENERIC_FOLDER + "0", "display.png")
```

```
        if "-doubleCompilation" in commands :
            doubleCompilation = True
        if noSave is not None or doubleCompilation :
            displayLatexCode(cell, noSave, doubleCompilation)
        else :
            displayLatexCode(cell)

def PSViewer(line) :
    """The function defining the %PSViewer magic.
    It mainly contains utility functionalities in order to modify slightly the
    behaviour of the module or retrieve some informations.
    The magic is expected to be followed by an option from:
    -reset -- delete the files created during the compilations, expect if they have
    been stored in a specified path.
    -path -- display the path to the folder where the intermediate files are created.
    -forceCompile -- toggle the forced compilation of the code/files given.
    -compilationState -- display if the compilation is forced or not.
    Any other option (even no option) will be interpreted as -help and will display
    the help.

    Keyword arguments:
    line -- the line of text after the magic's invocation, of the form "option"
    """
    import PSViewer
    if line == "-reset" :
        clearVersion()
        print ("Version cleared, all files deleted")
    elif line == "-path" :
        print (os.getcwd())
    elif line == "-forceCompile" :
        if PSViewer.forceCompile :
            PSViewer.forceCompile = False
            print("Toggled behaviour to displaying given image")
        else :
            PSViewer.forceCompile = True
            print("Toggled behaviour to forced compilation")
    elif line == "-compilationState" :
        if PSViewer.forceCompile :
            print("Default behaviour : forced compilation")
        else :
            print("Default behaviour : displaying given image")
    else :
        print (r"""
The following magics are defined in this module :
        (all the options must be on the same line that the magic)

%%PSCode allows to run postScript code directly in the cell.
        Giving a path as first argument will modify the behaviour of the magic.
        If one or multiple png images are found at this emplacement, they will
    be displayed instead of the compilation. In the other case or if forceCompile
    is enabled, the file will be compiled and displayed.
        the option -noSave prevent the excessive usage of memory by overriding
    the intermediate files in memory.
        Caution : this option can have unexpected result if you convert a document
```

```
    with less pages than the previous one.

%PSFile allows to view the content of a ps file in the notebook.
        The first argument is always the path to the file. A second path can be given.
    With default behaviour, the png images corresponding to the path will be loaded
    and displayed. With forceCompile or if the path is empty, the png generated by
    the magic will be stored at this emplacement.
        The element(s) must be in absolute path or in the notebook's directory.

%%LatexCode allows to run LaTeX code in the cell. Following options exist :
        Giving a path as first argument will modify the behaviour of the magic. If one
    or multiple png images are found at this emplacement, they will be displayed
    instead of the compilation. In the other case or if forceCompile is enabled,
    the file will be compiled and displayed.
        -noSave prevent the excessive usage of memory by overriding the intermediate files
    created in memory.
        Caution : this option can have unexpected result if you convert a document with
    less pages than the previous one.
        -doubleCompilation is useful if you need to compile a LaTeX document with references
    in it. You cannot compile multiple times just by reinterpreting the cell.
        -noImport allow to write a basic LaTeX document without import, without specifying
    \documentclass, \begin{document} and \end{document}

%LatexFile allows to view the content of a LaTeX file in the notebook.
        The first argument is always the path to the file. A second path can be given.
    With default behaviour, the png images corresponding to the path will be loaded
    and displayed. With forceCompile or if the path is empty, the png generated by
    the magic will be stored at this emplacement.
        The element(s) must be in absolute path or in the notebook's directory.

%PSViewer is the set of utility functions. It is mostly for help.
        -reset clear all the files created with the other magics. It can't be undone.
        -path show the path to the default folder containing the files created with
    the magics.
        -forceCompile toggle the forced compilation of the LaTeX/PS code.
        -compilationState displays what is the treatment for the path given for
    the conversion.
        Any other argument will be interpreted as -help and will display this help.""")


#--------------------------
#    Declaring magics
#--------------------------


def load_ipython_extension(ipython) :
    """The function defining the magics and which is their type."""
    ipython.register_magic_function(PSViewer, "line")
    ipython.register_magic_function(PSFile, "line")
    ipython.register_magic_function(LatexFile, "line")
    ipython.register_magic_function(PSCode, "cell")
    ipython.register_magic_function(LatexCode, "cell")
```

4 Documentation of the module

```
In [3]: %LatexFile PSViewer.tex Images\documentation.png

        #An higher quality document is joined to this report as 'PSViewer.pdf'.
```



PSViewer Documentation

Release 1.0

Duncan Forel

Jun 09, 2016

ii

CHAPTER

ONE

INTRODUCTION TO PSVIEWER

PSViewer is an Python module designed to enhance a Jupyter notebook. It creates multiple magics to deal with the PostScript and Latex code directly in the cells (including files written in these languages).

1.1 Installation

1.1.1 Dependencies

This module use multiple external tools to convert the files to the correct format.

As the compilation of the code/file is not mandatory, the installation of the following dependencies isn't required if you project to use an already generated notebook with the associated images.

The dependencies are :

- ghostScript in order to read the PostScript files (download from the ghostScript website);
- ImageMagick in order to convert the files (download from the ImageMagick website);
- latex/dvips if you project to use the latex macros of the module (the distribution of your choice);

1.1.2 Configuration of ImageMagick on Windows

ImageMagick provides a lot of utility functions accessible by command-line invocation. However, some of them override native functionality of the OS (notably convert on Windows). In order to keep the system clear, it will be needed to rename this function.

For that, go to the installation folder of ImageMagick (usually C:\Programmes\ImageMagick-[version]) and rename the executable convert.exe by convert_img.exe. If another program needs the convert.exe, copy the file and rename the copy as convert_img.exe.

1.1.3 Loading the module

The module can be loaded in a document with the magic

```
%load_ext PSViewer
```

More details on how to load a module in a notebook can be found here .

1.2 Optional compilation and PNG

1.2.1 Compiling or loading ?

Every magic of this module (excepting %PSViewer) allow to provide a path to a PNG file as argument. The default behaviour is to check for the existence of PNG images at this emplacement using the method described below and display this image without executing the magic.

In the case of forced compilation (or if no file is found), the magic will be executed and the PNG image(s) resulting of the compilation will be saved at the given path, then displayed.

1.2.2 Format of the search for PNG files

A source of unexpected error is the way PSViewer searches for images. Multiple magics allow to specify a path which will be used to scan for PNG file and if found, display this/these files.

The precise path to a file will display the corresponding image, but if no image is found the module will scan the folder for a file in the format path-x.png (where x is an integer and path is the name of the file without extension) and display all the successive files in crescent value of x. The search starts with a x value of 0 and ends when no file of the searched format is found. Take care, files in the format path-00.png are not checked.

For example, if you give a path like myImage.png, but the folder contains only myImage-0.png, myImage-1.png and myImage-3.png, the program will display myImage-0.png and myImage-1.png.

It allows to write only one path to retrieve multiple PNG, but can induce unexpected behaviour when the page's number of a compiled document decrease (as the images from the previous compilation aren't deleted). In the same way, going from single-page document to multi-page will cause an apparent error, even if the compilation is correct.

CHAPTER

TWO

MAGICS ADDED BY PSVIEWER

PSViewer adds multiple line and cell magics to your notebook. They are a total of 5, one of them is only for configuration and utility.

2.1 %PSViewer

This is the utility/configuration magic. It contains multiple options.

It mainly contains utility functionalities in order to modify slightly the behaviour of the module or retrieve some informations. The magic is expected to be followed by an option from:

-reset : delete the files created during the compilations, expect if they have been stored in a specified path.

-path : display the path to the folder where the intermediate files are created.

-forceCompile : toggle the forced compilation of the code/files given.

-compilationState : display if the compilation is forced or not.

Any other option (even an empty line) will be interpreted as -help and will display the help.

Form of the magic : %PSViewer [option]

2.2 %PSFile

The expected arguments are the path to the PS file to display and optionally the path of the PNG file corresponding (if not found or compilation forced, it will be the converted file's path).

In case of incorrect use, this function displays an error message.

Form of the magic : %PSFile path [pngPath]

2.3 %LatexFile

The expected arguments are the path to the Latex file to display and optionally the path of the PNG file corresponding (if not found or compilation forced, it will be the converted file's path).

In case of incorrect use, this function displays an error message.

Form of the magic : %LatexFile path [pngPath]

2.4 %%PSCode

The line argument contains the options of the magic. If a path is specified as first option, the magic will try to retrieve PNG images from this emplacement and display them without any compilation. If the compilation is forced or if no file are found, the following options will be evaluated and the conversion of the cell's code will be execute.

The current options are :

-noSave which allow the override of the intermediate files. Caution : this can cause unexpected behaviour in some cases.

Any other command will be ignored.

Form of the magic : %%PSCode [pngPath] [option]*

The code should be the content of the cell, starting the line after the magic.

2.5 %%LatexCode

The line argument contains the options of the magic. If a path is specified as first option, the magic will try to retrieve PNG images from this emplacement and display them without any compilation. If the compilation is forced or if no file are found, the following options will be evaluated and the conversion of the cell's code will be execute.

The current options are :

-noSave which allow the override of the intermediate files. Caution : this can cause unexpected behaviour in some cases.

-noImport generates the documentclass, begin{document} and end{document} of the file automatically.

-doubleCompilation indicates that a double compilation is required. This option is useful in case of internal links.

Any other command will be ignored.

Form of the magic : %%LatexCode [pngPath] [option]*

The code should be the content of the cell, starting the line after the magic.

CHAPTER

THREE

NON-MAGICS FUNCTIONS

This section doesn't contain information about how to use the module. The purpose is to describe the methods actually present but not directly registered as magics.

3.1 Low-level functions

3.1.1 clearVersion

clearVersion ()

Utility function used to clear the documents created by the compilation. It retrieves the current index for the compilation and remove all the previous folders. If the value stored for the index is incorrect, nothing is done. It also removes the file containing the current index.

The default name of the file containing the index is "version.txt".

3.1.2 loadVersion

loadVersion ()

Utility function loading the current index for the folder storing a compiled document.

If the document storing the index is missing or the value contained is incorrect (typically, if it is a string) the index will be set to 0. Then, the index incremented is stored in the version file and the folder corresponding to the index is created.

The default name of the file containing the index is "version.txt".

3.1.3 writeTeXFile

writeTeXFile (*data*, *version=0*)

Write the *data* in a TeX file corresponding to the folder of index *version*.

Keyword arguments:

- data – the string containing the text to write in the document.
- version – the index of the folder to store the file in (default 0)

3.1.4 writePSFile

writePSFile (*data*, *version=0*)

Almost the same function than writeTeXFile, only the extension of the written file changes to *.ps*.

Keyword arguments:

- data – the string containing the text to write in the document.
- version – the index of the folder to store the file in (default 0)

3.1.5 convertLaTeXFile

convertLaTeXFile (*path*, *doubleCompilation=False*)

Convert the TeX file at the specified path to a PS file, using the latex and dvips commands. If an error is encountered during the conversion, the process is stopped and an error message is displayed. This function also clears the .log, .aux and .dvi files created in the process. The double compilation is supported if needed.

Caution : this error message uses IPython functionalities.

The process return the path to the PS file if the compilation was successful, and None otherwise.

Keyword arguments:

- path – the path of the TeX file.
- doubleCompilation – boolean containing if a double compilation should be done (default False)

3.1.6 convertPSFile

convertPSFile (*path*, *pngPath=None*)

Convert the PS file at the specified path to one or multiple PNG files, one PNG image per page. The path to save the PNG can be specified, is must be a file and not a directory. The conversion uses a command from ImageMagick, convert (renamed convert_img on Windows). If no PNG path is specified, the name and the emplacement of the postScript file will be used instead. If a path different from a PNG file is given, a missing extension will be assumed.

This function returns the path of the PNG image (or the path of the common prefix of the files if multiple images are created).

Keyword arguments:

- path : the path of the PS file.
- pngPath : the path where to save the PNG file(s) (default None)

3.1.7 findPNGFiles

findPNGFiles (*pngPath*, *expectations=False*)

Utility function to retrieve one or multiple PNG files from an emplacement. The given path can design a file or the prefix of multiple PNG files (in the format prefix-x.png, where x is an integer). If a file is found, the prefix case won't be evaluated. All the path found are stored in a table and returned by the function. If the table is empty and it was expected to find results, an error message is displayed.

Keyword arguments:

- pngPath – the file's emplacement or the common prefix.

- expectations – if the function displays an error message in case of empty result (default False).

3.2 Intermediate functions

For all this functions, if a pngPath is provided, the corresponding image(s) will be loaded and displayed (using checkPNGFiles). If the compilation is forced or if no file is provided/found, the magic will be executed.

3.2.1 displayPSCode

displayPSCode (*data*, *pngPath=None*)

This function writes the given text in an PS file and calls displayPSFile on it, transmitting the PNG path.

Keyword arguments:

- data – the string containing the PS code.
- pngPath – the path where to fetch/save the PNG file (default None).

3.2.2 displayPSFile

displayPSFile (*path*, *pngPath=None*)

This function converts the PS file given in argument and displays the images resulting of the conversion.

Keyword arguments:

- path – the path of the PS file to convert.
- pngPath – the path where to fetch/save the PNG file (default None).

3.2.3 displayLatexCode

displayLatexCode (*data*, *pngPath=None*, *doubleCompilation=False*)

This function writes the given text in an TeX file and calls displayLatexFile on it, transmitting the PNG path and the indication for the double compilation.

Keyword arguments:

- data – the string containing the LaTeX code.
- pngPath – the path where to fetch/save the PNG file (default None).
- doubleCompilation – whether the compilation should be executed twice (default False)

3.2.4 displayLatexFile

displayLatexFile (*path*, *pngPath=None*, *doubleCompilation=False*)

This function converts the LaTeX file given in argument, using the double compilation if specified, generating a PS file. The result is then converted and displayed using displayPSFile (with the specified pngPath transmitted).

Keyword arguments:

- path – the path of the LaTeX file to convert.

- pngPath – the path where to fetch/save the PNG file (default None).
- doubleCompilation – whether the compilation should be executed twice (default False)

3.2.5 checkPNGFiles

checkPNGFiles (*path*, *expectations=False*)

Retrieve and display the PNG images corresponding to the specified path. If no file is found can display an error message, depending on the value of *expectations*. The algorithm used for retrieving files is given in findPNGFiles. This function returns whether files were found.

Keyword arguments:

- path – the path of the PNG file or the common prefix of multiple files.
- expectations – if the function displays an error message in case of empty result (default False).

3.3 IPython utiliy

load_ipython_extension (*ipython*)

The function used by the notebook to register the magics and their types.

CHAPTER

FOUR

EXAMPLES OF THE MODULE

4.1 Displaying files

To display a file, you can use both

```
%LatexFile <your file>

%PSFile <your file>
```

depending on the type of the file. If you want to save the result to a specific emplacement, use the following expressions :

```
%LatexFile <your file> <save path>

%PSFile <your file> <save path>
```

4.2 Writing code

There is no check in the notebook, however the error in compilation will display an error message.

In the case of LaTeX code, a link to the log document will appear and you'll be able to find what error caused the compilation fail. However, for PS code no error log is displayed. It is consequently recommended to use LaTeX compilation to create PS code.

Format of writing :

```
%%LatexCode <optional save path> <option*>
<multi-line code>

%%PSCode <optional save path> <option*>
<multi-line code>
```

4.3 Generating notebook for another user

If you want to share your notebook with persons not using latex, you can ! First, enable forced compilation with

```
%PSViewer -forceCompile
```

Then, write your notebook as usual, specifying a save path for each figure using the magics. Take care at giving distinct names for each figure !

If you want to share, you just have to give the notebook with the PNG images. The receiver just has to execute the cells without activating the forced compilation and the figures will be retrieved from the already generated images.