Comparison of Graph Plotting Style/Figure in various Journals and universities. Pythonizing and meeting the recommended requirements.

Animesh Srivastava(@AnimeshSri.nith)

I. Introduction: Problem Statement

To read and learn from the different publishers and to publish a short essay about how each University and Corporation communicate visually their math-origin results. As a second stage, "pythonize" to replicate such visual standards. and... And as a final stage, create something NEW, EXCITING AND VISUALLY INVITING for Ignitus. That is, all of this relates to create a visual style for Ignitus papers. The first milestone, the graphics.

II. Detailed Information regarding Graphics

The comparison between two Publishers- IEEE Standard Format Paper and Elsevier has been carried out and related code base has been designed for a person/user to craft out a graph/figure following the norms as mentioned in the rule book of the Journals. Specific function and code required have been mentioned below, the combination of all the methods will be used to meet the requirement of a particular paper style. The standard minimum requirement for a graph/figure to be valid in IEEE Paper RuleBook

1) Resolution
   - Black and White: 300 DPI
   - Grayscale: 150 DPI
   - Black and White Photograph: 300 DPI
2) Line Weight
   - ines should be of an adequate thickness, at least 0.5 to 1.0 point thickness. Hairline rules may appear broken up in the printed document, or not show up at all.
3) Fonts to Use
   a) Times New Roman
   b) Arial

The Elsevier Publication Rulebook states :

1) Resolution
   - Black and White: 1000 DPI (minimum)(or 1200 dpi if the image contains very fine line weights)
   - Grayscale: 300 DPI
2) Line Weight
   - Line weights range from 0.10 pt to 1.5 pt
3) Fonts to Use
   a) Courier
   b) Symbol
   c) Times (or Times New Roman)
   d) Arial

III. Breaking the Graphical Representation

We focus on "pythonizing" the two important factors DPI and image format:

A. DPI

We discuss some basics about DPI and image saving, the libraries used for the Conversion of an image and saving in Python Language. The DPI values are only metadata on computer images. They give hints on how to display or print an image. Printing a 360 x 360 image with 360 DPI will result in a 11 inches printout. A simplified way to explain it: The DPI setting recommends a zoom level for the image. Saving with other DPIs will not change the content of the image. If you want a larger image use a larger size and a larger font. Image resolution, number of pixels and print size are related mathematically: Pixels = Resolution (DPI) Print size (in inches). The various famous libraries for plotting and saving images are

- Plotly
- matplotlib
- Seaborn
- ggplot
- bokeh
- pygal
- geoplotlib
- gleam
- missingno
- Leather.

The discussion regarding specifications/comparison of each library is not done in the present documented version. Let us focus on the steps required to set a particular DPI for our images:

1) How to change DPI in Matplotlib?: Matplotlib uses a default DPI of 100, to change to required DPI, you can specify the DPI when you save a plot to a file:

```
using matplotlib.pyplot as mplot
mplot.savefig("foo.png", dpi=300)
```

If you want this DPI to be used for all output by Matplotlib, then set this line in your matplotlibrc file:

```
savefig.dpi: 300
```

So matplotlib provides an easy function for setting our DPI.

A particular discussion regarding the failure of DPI Set property of Matplotlib was raised on Julialang.org , please follow the discussion on Plots, How to save figure to the file with high resolution (dpi=300)? Also, pillow image DPI setting methods is discussed How do I properly set DPI when saving a pillow image?

B. Conversion of an image from RGB to Grayscale, RGB to Black and White?

A discussion on stackoverlfow focuses on image conversion property states that numpy or matplotlib doesn't have a built-in function to convert from rgb to gray, which is the most basic operation for image processing. We now focus on a Library named as Pillow, Pillow is the friendly PIL fork by Alex Clark and Contributors. PIL is the Python Imaging Library by Fredrik Lundh and Contributors. A comparison between Pillow and matplotlib code is done below:

```
#Code 1
from PIL import Image
img = Image.open('image.png').convert('LA')
img.save('greyscale.png')
```

To save an image in the grayscale format using Matplotlib we need to understand that Conversion of an arbitrary color image to grayscale is not unique in general; different weighting of the color channels effectively represent the effect of shooting black-and-white film with different-colored photographic filters on the cameras. Please read the Colorimetric (perceptual luminance-preserving) conversion to grayscale. A general formula developed for the conversion is: For images in color spaces such as Y'UV and its relatives, which are used in standard color TV and video systems such as PAL, SECAM, and NTSC, a nonlinear luma component (Y') is calculated directly from gamma-compressed primary intensities as a weighted sum, which, although not a perfect representation of the colorimetric luminance, can be calculated more quickly without the gamma expansion and compression used in photometric/colorimetric calculations. In the Y'UV and Y'IQ models used by PAL and NTSC, the rec601 luma (Y') component is computed as

$$Y = 0.299R + 0.587G + 0.114B \tag{1}$$

$$Y' = 0.299R' + 0.587G' + 0.114B' \tag{2}$$

where we use the prime to distinguish these nonlinear values from the sRGB nonlinear values (discussed above) which use a somewhat different gamma compression formula, and from the linear RGB components. The ITU-R BT.709 standard used for HDTV developed by the ATSC uses different color coefficients, computing the luma component as

$$Y = 0.2126R + 0.7152G + 0.0722B \tag{3}$$

$$Y' = 0.2126R' + 0.7152G' + 0.0722B' \tag{4}$$

. Although these are numerically the same coefficients used in sRGB above, the effect is different because here they are being applied directly to gamma-compressed values rather than to the linearized values. The ITU-R BT.2100 standard for HDR television uses yet different coefficients, computing the luma component as

$$Y = 0.2627R + 0.6780G + 0.0593B \tag{5}$$

Normally these color spaces are transformed back to non-linear R'G'B' before rendering for viewing. To the extent that enough precision remains, they can then be rendered accurately.

```
#Code 2 using Matplotlib
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.1

img = mpimg.imread('image.png')
gray = rgb2gray(img)
plt.imshow(gray, cmap = plt.get_cmap('gray'))
plt.show()
```

IV. Conclusion

We compared two Publishers IEEE and Elsevier on basis of Graphical representation requirement in the paper. We focused on DPI and Image Conversion.