PythonTeX Examples

Alessandro Rizzoni

February 7, 2024

Introduction

These are some examples that show the capabilities of PythonTEX. These examples were written using the `\pyblock` environment - normally it is preferred to use a `\pythontexcustomcode` environment so that the settings can be applied to multiple Python evironments within the document.

Contents

1 Python Environment

The first example covers the python environment that we will use for this document. It's pretty simple! All we need for the first few examples is `numpy` and `matplotlib`. We can also use one of `matplotlib`'s dependencies, `cycler`, to save some time.

```python
import os

import numpy as np # Python numerical computing library
import matplotlib as mpl

from cycler import cycler # Property cycler utilities
from matplotlib import pyplot as plt # Pyplot API
from matplotlib import rcParams as rc # Matplotlib plot styling
from matplotlib.ticker import EngFormatter # Plot tick formatting

class Environment():

    build_directory = os.path.abspath(os.getcwd())
    output_directory = os.sep.join(build_directory.split(os.sep)[:-1])
    figures_directory = os.path.join(output_directory, 'figures')
    pgf_directory = os.path.join(figures_directory, 'pgf')
    pdf_directory = os.path.join(figures_directory, 'pdf')

    def __init__(self):
        figure_directories = [ self.figures_directory, self.pgf_directory, self.pdf_directory ]
        for directory in figure_directories:
            if not os.path.exists(directory):
                os.mkdir(directory)

environment = Environment()
```

2 Utility Functions

Example 2 contains some functions that will be useful later on. The `metallic_ratio()` function returns the "metallic ratio" of a numeric argument, and the `eng_format()` function returns an `EngFormatter` object that is used to generate the engineering format plot ticks that are used later on. Information on the metallic ratio, or mean, can be found here: `https://en.wikipedia.org/wiki/Metallic_mean`.

```python
def metallic_ratio(n):
    return 0.5 * ( n + np.sqrt(n**2 + 4) )

```

```python
def eng_format(arg: str):
    return EngFormatter(unit=arg, sep=r'\,')

def save_pythontex_figure(figure, figure_name):
    if type(figure) == mpl.figure.Figure:
        figure.savefig(os.path.join(environment.pgf_directory, f'{figure_name}.pgf'))
        figure.savefig(os.path.join(environment.pdf_directory, f'{figure_name}.pdf'))
    return figure
```

3 Geometry Class

Example 3 starts to get to the good stuff - here we are starting to interact with the Python context we generated in the preamble to pull variables from LaTeX into our Python environment. The `pytex.context` dictionary holds all of the values, etc. that we pass to Python, and we can access them as follows. Some additional sanitization and conditioning is required to give a plain string and convert from LaTeX points to inches. the three values we will start off with are the column width, the text width, and the text height. They are generally described as they are named, but the text height and the column height are not necessarily the same, such as in a multiple column document.

```python
class Geometry():

    # Convert from (LaTeX!) points to inches
    # There is ~ 0.14 micron floating point error here
    in_length = float(pytex.context['in'][:-2]) / 72.27
    cm_length = float(pytex.context['cm'][:-2]) / 72.27
    mm_length = float(pytex.context['mm'][:-2]) / 72.27
    em_length = float(pytex.context['em'][:-2]) / 72.27
    ex_length = float(pytex.context['ex'][:-2]) / 72.27
    bp_length = float(pytex.context['bp'][:-2]) / 72.27
    dd_length = float(pytex.context['dd'][:-2]) / 72.27
    pc_length = float(pytex.context['pc'][:-2]) / 72.27

    column_width = float(pytex.context['columnwidth'][:-2]) / 72.27
    text_width = float(pytex.context['textwidth'][:-2]) / 72.27
    text_height = float(pytex.context['textheight'][:-2]) / 72.27
    figure_width = column_width - 2 * em_length
    figure_height = figure_width / metallic_ratio(1) # Define figure height as a function of the figure width and the golden ratio

    axis_dimensions = (0, 0, 1, 1) # 0 lr margin, 0 tb margin, 100% figure size

geometry = Geometry()
```

4 Font Class

Here we pull the fonts from the LaTeX document so we can use them in our `matplotlib` plots and have a consistently formatted document. The font names are taken from the context dictionary and sanitized as before. After the typefaces have been transferred, the font sizes can be taken. The font sizes are taken from the default LaTeX font sizes - other sizes can be added or the base font commands can be patched if desired. The class is declared to allow for dot notation. Eventually, there may be a need for additional font configuration code, and having a class interface in place will simplify expansion later on.

```
# Determine font parameters
class Font():
    # Font faces
    roman = pytex.context['romanfont'].split('/')[1][:-3]
    bold_roman = pytex.context['romanboldfont'].split('/')[1][:-3]
    bold_italic_roman = pytex.context['romanbolditalicfont'].split('/')[1][:-3]
    italic_roman = pytex.context['romanitalicfont'].split('/')[1][:-3]
    sans = pytex.context['sansfont'].split('/')[1][:-3]
    mono = pytex.context['monofont'].split('/')[1][:-3]
    math = pytex.context['mathfont'].split('/')[1][:-3]

    # Font sizes
    tiny = pytex.context['tiny']
    script_size = pytex.context['scriptsize']
    footnote_size = pytex.context['footnotesize']
    small = pytex.context['small']
    normal_size = pytex.context['normalsize']
    large = pytex.context['large']
    llarge = pytex.context['Large']
    lllarge = pytex.context['LARGE']
    huge = pytex.context['huge']
    hhuge = pytex.context['Huge']

font = Font() # Instantiate the class for use
```

5 Plot Settings

Example 5 configures various options for the plots. The figure geometry is defined first, then the colormap is selected. The `grey` colorscheme was chosen to keep the document in black and white. Once the colormap and the number of styles are selected, a `cycler` is created to iterate over the different color and dash styles.

```
# figure settings
cmap = plt.get_cmap('grey') # Select colormap
```

```
num_plot_styles = 4 # number of colors for plotting

# Initialize empty lists for plot colors and styles
plot_colors = []
line_styles = []
for i in range(num_plot_styles): # Populate the color and style lists
    plot_colors.append(cmap(1.0 * i/num_plot_styles))
    line_styles.append((0, (i+1, i)))

# Define the main cycler with the two component lists
style_cycler = cycler(color=plot_colors, linestyle=line_styles)
```

6 Matplotlib Configuration

This example is interesting - the `matplotlib` rcparams can be modified for the active Python evironment. The more interesting feature is the PGF backend which allows for the output image to be produced as directly importable LaTeX code with the `pgf` package. The backend allows for a LaTeX preamble to be defined, and this feature is used here to create a unified plot style with the same font as the rest of the document. Other fonts are available in the Python environment - any font you define in the normal LaTeX ways, such as by importing a package, using `fontspec` or `fontsetup`, etc.

```
# Document-wide Matplotlib Configuration
rc.update({
        'backend': 'pgf',
        'lines.linewidth': 1,
        'font.family': 'serif',
        'font.size': font.footnote_size,
        'text.usetex': True,
        'axes.prop_cycle': style_cycler,
        'axes.labelsize': font.footnote_size,
        'axes.linewidth': 0.8,
        'xtick.direction': 'in',
        'xtick.top': True,
        'xtick.bottom': True,
        'xtick.minor.visible': True,
        'ytick.direction': 'in',
        'ytick.left': True,
        'ytick.right': True,
        'ytick.minor.visible': True,
        'legend.fontsize': font.footnote_size,
        'legend.fancybox': False,
        'figure.figsize': (geometry.figure_width, geometry.figure_height),
        'figure.dpi': 600,
```

```
        'figure.constrained_layout.use': True,
        'figure.constrained_layout.hspace': 0,
        'figure.constrained_layout.wspace': 0,
        'figure.constrained_layout.w_pad': 0,
        'figure.constrained_layout.h_pad': 0,
        'savefig.format': 'pgf',
        'savefig.bbox': 'tight',
        'savefig.transparent': True,
        'pgf.rcfonts': False,
        'pgf.preamble': '\n'.join([
                r'\usepackage{mathtools}',
                r'\usepackage[warnings-off={mathtools-colon, mathtools-overbracket}]{unicode-math}',
                r'\usepackage{lualatex-math}',
                r'\usepackage{siunitx}',
                r'\usepackage{fontspec}',
                r'\setmainfont{%s}[Ligatures=TeX, ItalicFont=%s, BoldFont=%s, BoldItalicFont=%s]' %(font.roman, font.italic_roman, font.bold_roman, font.bold_italic_roman),
                r'\setmathfont{%s}' %(font.math),
                r'\setsansfont{%s}[Ligatures=TeX, Scale=MatchLowercase]' %(font.sans),
                r'\setmonofont{%s}[Ligatures=TeX, Scale=MatchLowercase]' %(font.mono),
                r'\usepackage[USenglish]{babel}',
                r'\usepackage[autostyle=true]{csquotes}'
        ]),
        'pgf.texsystem': 'lualatex', # default is xetex, but lualatex is preferred
})
```

7 Example Plot

This example shows a few different things that are possible. First, a plot can be generated with plot size and other parameters passed to Python from LaTeX. The example also shows the utility provided by the use of the Matplotlib PGF backend's LaTeX preamble, which was defined in the Matplotlib configuration code. This allows the package `siunitx` to be loaded inside Matplotlib so that mathematics and physical units can be properly typeset using LaTeX.

```
fig, ax = plt.subplots()

x = np.linspace(0, 1, 1000)
for frequency in range(1, num_plot_styles + 1, 1):
    _ = ax.plot(
        x,
        np.sin(2*np.pi*frequency*x),
        label=r'\SI{%.1f}{\hertz}' % frequency
```

```
        )

ax.xaxis.set_major_formatter(
    eng_format(r'\unit{\second}')
    )

ax.yaxis.set_major_formatter(
    eng_format(r'\unit{\volt}')
    )

_ = ax.legend()

figure_name = 'example_figure'
save_pythontex_figure(fig, figure_name)
```

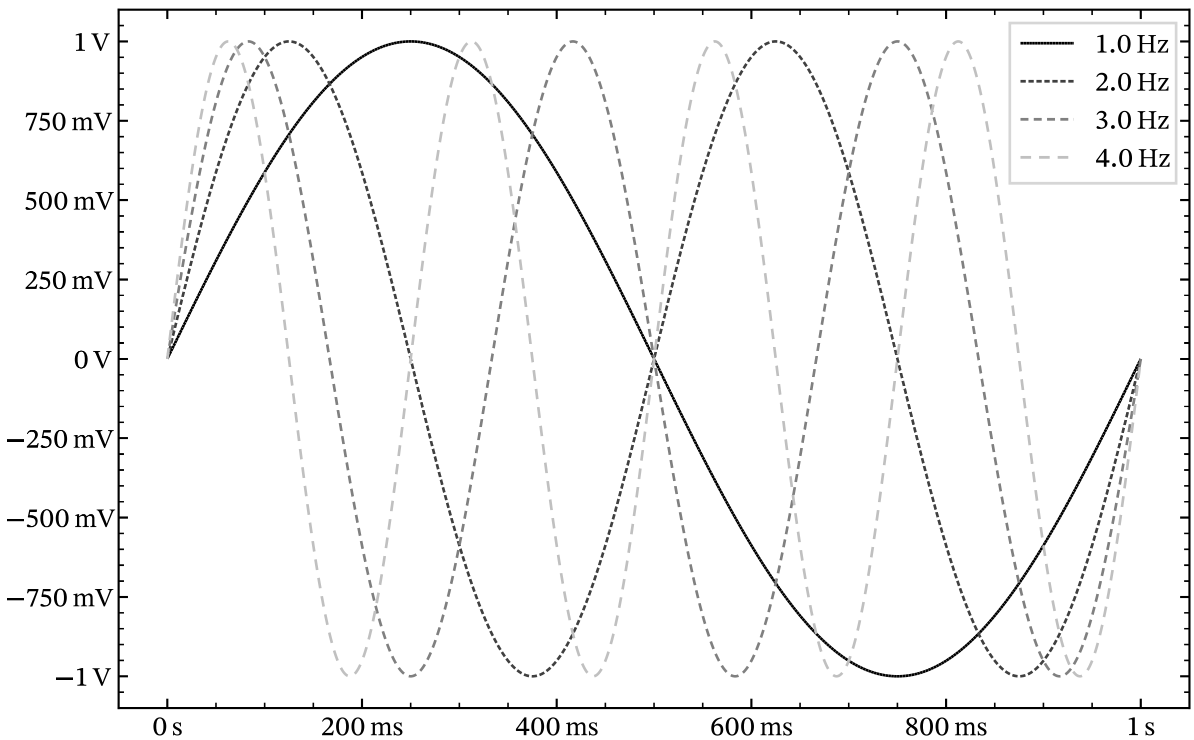

Figure 1: Example 7 Plot

8 Font Example

Example 8 shows the basics of how to use the font class.

```
print(font.roman, font.math, font.sans, font.mono)
```

```
STIXTwoText-Regular.otf STIXTwoMath-Regular.otf IosevkaAile IosevkaFixed
```

Figure 2: Example 8 Output

9 Geometry Example

In this example, we show how to interact with the geometry class.

```
print(geometry.em_length, geometry.ex_length, geometry.in_length, geometry.cm_length)
```

```
0.16604400166044003 0.07853881278538813 0.9999998616299988 0.3937005673170057
```

Figure 3: Example 9 Output