Let your data SPEAK

Author: Bartosz Telenczuk

Date: Kiel, September 3rd, 2012

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Contents

1 Introduction

1.1 What can visualization be used for?

1. Comparing data and showing relation ships
2. Showing links between elements
3. Showing structures
4. Showing function
5. Showing processes
6. Systematisation and ordering

Good visualizations can often lead to new scientific discoveries

2 Design

2.1 Before you start

- plan out of grid
- do not let technology impede your creativity
- you don't have to know how to draw, basic skills in handling of a pencil and paper are enough

2.2 Design Principles

1. Principle of graphical excellence:
   - Graphical excellence = substance, statistics, design
   - simplicity of design and complexity of data
2. Graphical integrity:
   - Lie factor should be close to 1:

     $$\text{Lie factor } = \frac{\text{size of effect shown in graph}}{\text{real size of effect}}$$
   - one data dimension -> one visual dimension
3. Data-ink maximization
   - above all show data
   - maximize data-ink ratio (within reason)

     $$\text{data}-\text{ink ratio } = \frac{\text{ink used to plot data}}{\text{total ink used}}$$
   - erase redundant data
   - avoid chart junk (Moire patterns, grids, graphical duck -- graphs created only for decoration)
4. Data density maximization
   - maximize data density (within reason)

     $$\text{data density } = \frac{\text{number of entries in datamatrix}}{\text{area of data graphic}}$$

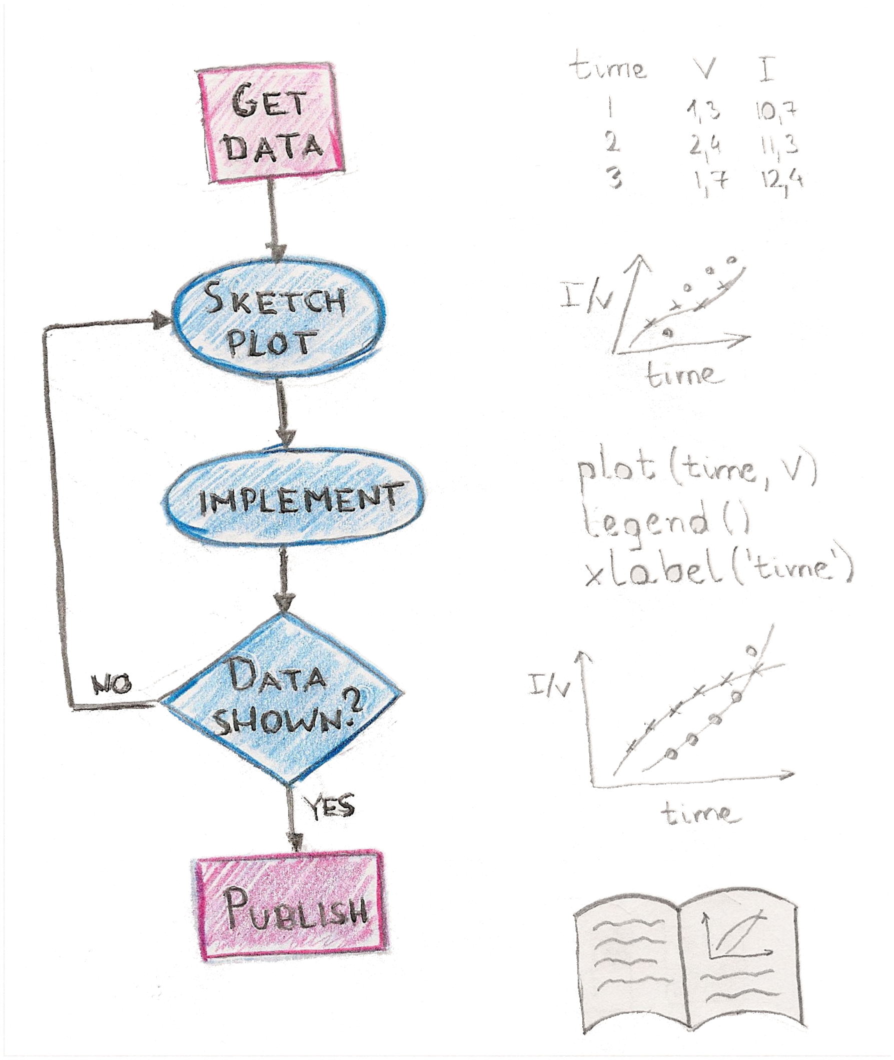
2.3 Graphical design patterns

- **multivariate (X-Y) plots** -- investigating correlations
- **multi-functioning graphical elements** -- labels as data (for example, stem and leaf plot, range frame, data-based grids)
- **small multiples**: repeat the same design several times only changing one variable (such as time)
- **map-based** plot for geographical data
- **micro-macro** design: show both the global context (big picture) and fine detail

3 Matplotlib

3.1 Visualization flowchart

- refine: usually in some DTP or vector-grahics editing software



3.2 Interactive session

An extensive tutorial on matplotlib can be found in Scientific Python [2] lecture notes by Emmanuelle Gouillart and Gae■l Varoquaux.

First open your ipython interpreter. We will use *pylab* option which opens a envrionment with matplotlib imported in the main namespace. In addition, it sets an interactive mode up, which allows one to see immediately the results of plotting commands. This mode is not recommended for production plots!

```
ipython -pylab
```

```
>>> plot([1, 2, 3])
a line plot should pop up
>>> close()
>>> bar([1, 2, 3], [1, 2, 3])
a bar plot should appear
>>> close()

evenly sampled time at 200ms intervals
>>> t = arange(0., 5., 0.2)
>>> plot(t, t, 'o')
>>> plot(t, t**2, 'x')
```

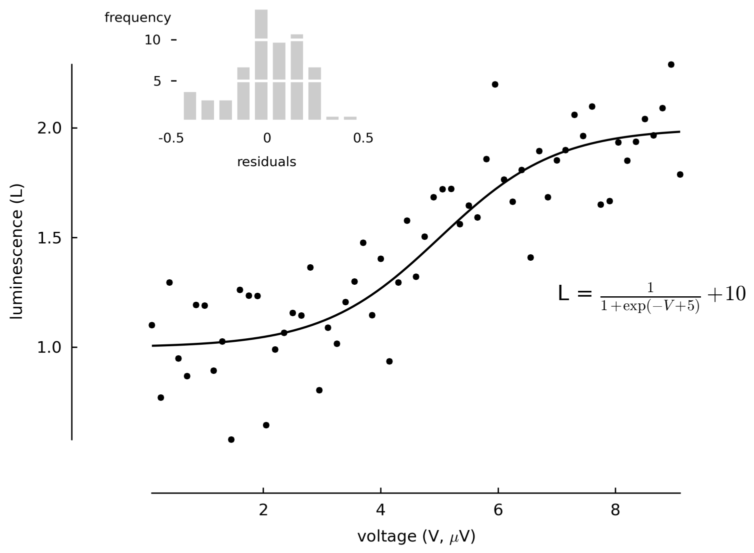
```
try adding some labels and a legend
>>> xlabel("time")
>>> legend(["t", "t^2"])

you can also produce multi-panel plot
>>> subplot(211)
>>> plot(t, t)
>>> subplot(212)
>>> plot(t, t**2)
```

- `matplotlib.pyplot` offers many interesting plot types: line/timeseries, pie chart, scatter, image, bar, vector, contour, 3d plot, map (with basemap extension). See also matplotlib gallery [1].
- most plots have high quality out of the box (good default settings, colors etc.).
- However, for publication quality plots still some customization is required.

3.3 Publication-ready figures

- matplotlib is easily cutomizable: almost each aspect of the plot can be easily chnaged via object-oriented interface: **plot size, localization of axes, fonts, colors, axes positions**
- exported figure looks like on the screen
- you can easily add LaTeX equations to your plots

Here is an example of a formatted plot with mutliple elements, labels.



Source code of the plot is available in the examples archive (`pyplot_publication.py`).

Here is a template which may be used to generate such plots:

```
from matplotlib import rcParams
params = {'backend': 'Agg',
          'figure.figsize': [5,3],
          'font.family': 'sans-serif',
          'font.size' : 8
          }
rcParams.update(params)                     # set plot attributes
import matplotlib.pyplot as plt
```

```
# [...]                                     # load data

fig = plt.figure()                          # create new figure
ax1 = plt.axes((0.18, 0.20, 0.55, 0.65))    # create new axes

plt.plot(t, y, 'k.', ms=4., clip_on=False) # plot data
ax1.set_xlim((t.min(), t.max()))
ax1.set_ylim((y.min(), y.max()))            # set axis limits

plt.xlabel(r'voltage (V, $\mu$V)')
plt.ylabel('luminescence (L)')              # set axis labels
# [...]

ax_inset = plt.axes((0.2, 0.75, 0.2, 0.2),
                    frameon=False)          # add an inset
#[...]                                      # plot data into the inset

plt.savefig('pyplot_publication.png')       #save plot
```

Things to remember:

- change the default settings (figure size, backend, fonts) in rcParams
- export to vector based graphic (such as EPS/SVG/PDF)

3.4 Matplotlib OO API

Matplotlib lets you to go even deeper: it offers a wide selection of graphical primitives with which you can build your own plots and visualizations. and remember: everything is Python so you can read the code, learn from it and modify it!

Here is an example of adding a circle artist to the axes container via OO API (*artist_example.py*):

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

font = "sans-serif"
fig = plt.figure(figsize=(5,5))          # create figure container
ax = plt.axes([0,0,1,1], frameon=False) # create axes container

art = mpatches.Circle((0.5, 0.5), 0.5,
                       ec="none")        # create an artist

ax.add_patch(art)                        # add the artist to the
                                         #   container

ax.set_xticks([])                        # remove axes ticks
ax.set_yticks([])

plt.savefig("artist_example.png")
```

3.5 Interactive plots

In matplotlib you can also build complex GUI:

- add toolbars

- use widgets and callbacks

Here is an example of handling pick event which is generated when you click on a data point (artist) (*event_example.py*):

```
import numpy
from matplotlib.pyplot import figure, show

xs, ys = numpy.random.rand(2,100)

fig = figure()
ax = fig.add_subplot(111)
line, = ax.plot(xs, ys, 'o', picker=5)        # 5 points tolerance

def onpick(event):                             # define a handler

    i = event.ind                              # indices of clicked points
    ax.plot(xs[i], ys[i], 'ro')                # plot the points in red
    fig.canvas.draw()                          # update axes

fig.canvas.mpl_connect('pick_event', onpick) # connect handler toevent
show()                                       # enter the main loop
```

3.6 Building applications with matplotlib

Matplotlib plot can be embbeded in Gtk, Qt, Wx and even html (see `mplh5canvas.py`)!

See the tutorial [3] by Gael Varoquaux: Writing a graphical application for scientific programming using TraitsUI

4 Mayavi

4.1 Mayavi2 demo

Full version of the tutorial is available at Mayavi homepage [4].

1. Parametric Surface
    - run mayavi2 from the command line
    - add a ParametricSurface source
    - add a Surface module
    - navigate in the scene
    - add an Outline
    - show only contour lines of the surface
2. Load heart.vti

    This is a simple volume of 3D data (32 x 32 x 12 points) with scalars at each point (the points are equally spaced). he data apparently represents a CT scan of a heart. I have no idea whose heart!

    - add IsoSurface
    - add ContourGridPlan
    - add ScalarCutPlan, play with 3d widgets

4.2 Mlab

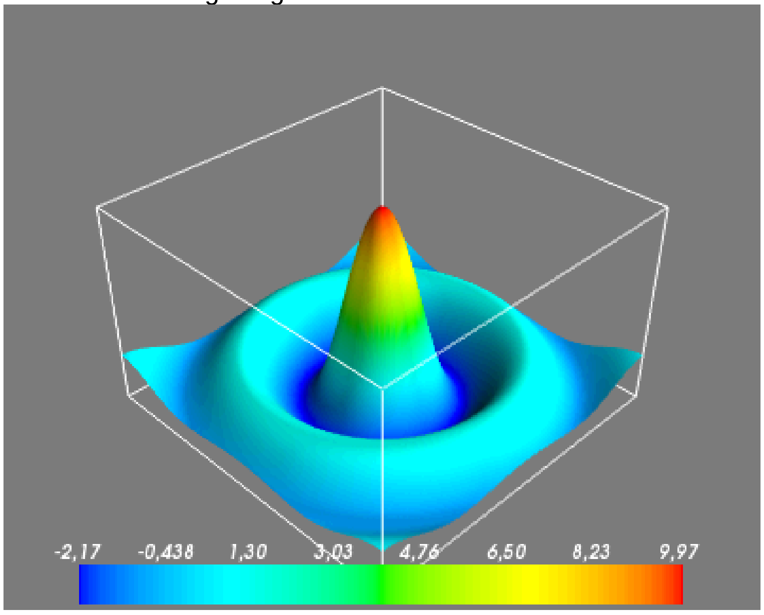
First run `ipython` with `wthread` option which sends all of the graphical interface to a separate thread which to avoid blocking the interpreter after showing the plot:

```
ipython -wthread
```

Now import `mlab` interface and plot a mexican hat:

```
>>> from enthought.mayavi import mlab
>>> import numpy as np

>>> x, y = np.ogrid[-10:10:100j, -10:10:100j]
>>> r = np.sqrt(x**2 + y**2)
>>> z = np.sin(r)/r

>>> mlab.surf(x,y, 10*z)
>>> mlab.outline()
>>> mlab.colorbar()
```

You should have obtained the following image:



Other mlab plotting functions include: `quiver3d`, `plot3d`, `point3d`, `contour3d`.

5 The End

Python is a prefect tool for data visualization:

- great 2D and 3D plotting libraries
- data parsing (for example, BeautifulSoup, see also lectures by Stefan and Francesc)
- statisitics (`numpy`, `scipy`, `scikits.learn`, MDP)
- UI (you can name plenty, check also TraitsUI)

6 Literature

1. Tufte, E. R. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, 1983.

2. Tufte, E.R. *Envisioning information* Graphics Press, Cheshire, CT, 1990.
3. Steele, J and Iliinsky, N, *Beautiful Visualization: Looking at Data through the Eyes of Experts* O'Reilly Media, 2010.
4. John Hunter and Michael Droettboom, *matplotlib* in *The Architecture of Open Source Applications. Volume II* link

---

1 http://matplotlib.sourceforge.net/gallery.html
2 https://portal.g-node.org/python-autumnschool/_media/pythonscientific.pdf
3 http://code.enthought.com/projects/traits/docs/html/tutorials/traits_ui_scientific_app.html
4 http://code.enthought.com/projects/mayavi/docs/development/html/mayavi/examples.html