Using the IPython Notebook as Lab Notebook

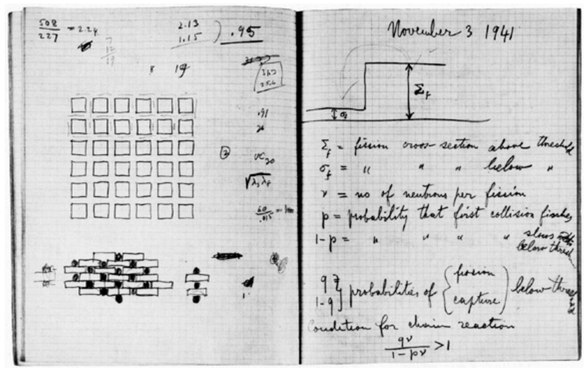
Jürgen Hasch, juergen.hasch@elbonia.de

Abstract

The IPython (Jupyter) notebook comes very close to the ideal electronic version of an engineer's lab notebook. It is easy to use, allows combining text, formulas, graphics, plots and tables, can do live calculations, and can be used to generate static documentation. Customizing the notebook with extensions significantly improves the workflow.

Motivation

Scientists and engineers keep a lab notebook to write down new ideas, do simple calculation or document measurements.
It is still the single most important tool, and often the primary means of documentation, even in the computer age. The picture shows a page of Enrico Fermi's lab notebook:



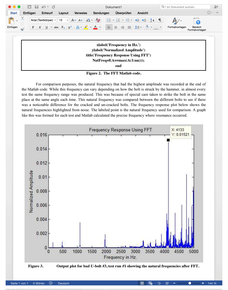
Entries in a lab notebook typically contain a heading, the current date, text, tables, math, drawings, and calculations.

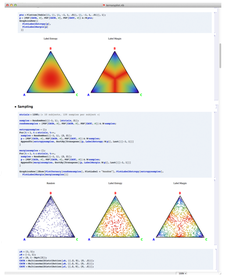
As computers have become an integral part of research and engineering, having a computer-based tool that essentially allows the same workflow as with a handwritten lab note-book becomes inevitable. The proper tool…

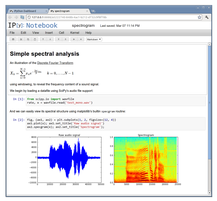
- Is easy to use (think pencil and paper)
- Can do text, formulas, graphics, plots, tables
- Can do calculations
- Can be reused
- Is useful for documentation

Computer Tools

- **Word**: The main tool being used even nowadays is Microsoft Word.
  Shocking but true.
  For calculations and generating plots, additional tools like Matlab are used.
- **Mathematica**: A powerful tool geared towards mathe-matical formulation of problems. It is not easy to use (think: pencil and paper), unless one spends considerable time learning it.
- **Mathcad:** Explicitly designed as engineering tool. Allows putting in formulas in mathematical notation and doing actual calculations.
  My experience: Limited capabilities, closed to the outside world, bugs.
- **IPython notebook**: Allows combining text, formulas, and graphics. And gives you the power if Python.







The Python Ecosystem

So what makes Python and the IPython notebook a good choice ?

- Python is a modern general-purpose language with a focus on making it fast and easy to use.
- The ecosystem: There's a package for that!
  For mathematics, plotting, optimization, data formats, internet protocols.
- The notebook frontend running in a web browser.
- The extensibility: IPython extensions allow adding functionality to the interpreter and the frontend.

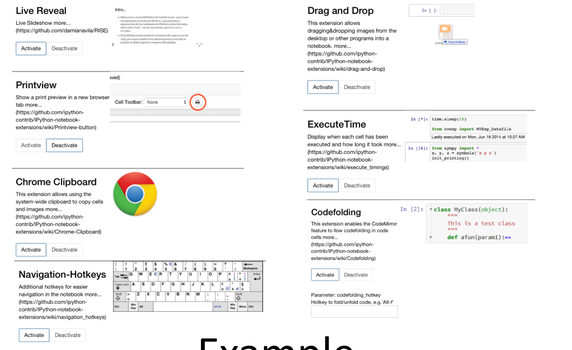
Customization

- IPython extension 'PhysicalQuantities'

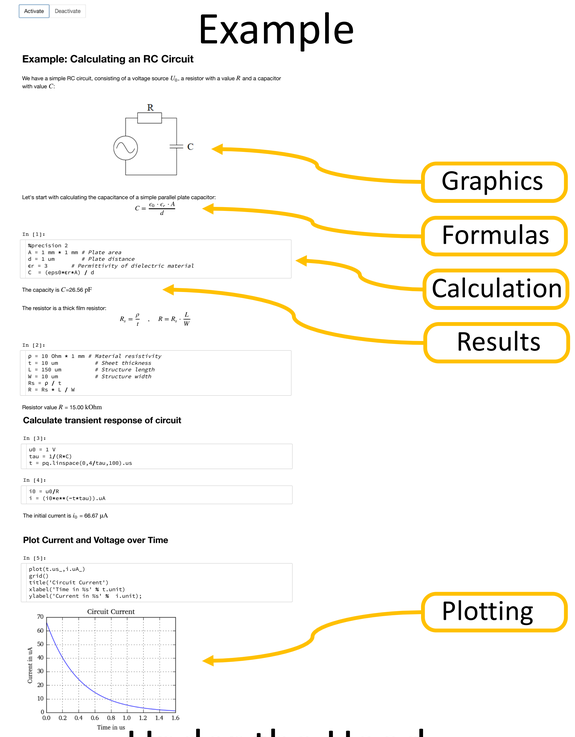
```
In [1]: 1m + 10 cm
Out[1]: 1.1 m
```

```
In [2]: a = 50 km/h
        type(a)
Out[2]: PhysicalQuantities.Quantity.PhysicalQuantity
```

- Notebook extensions



Example



Under the Hood

- Graphics

```
<img  src="/notebooks/PhysicalQuantities/images/rc.png"/>
```

- Formulas

```
Let's start with calculating the capacitance of a simple parallel plate capacitor:
$$ C = \frac{\epsilon_0 \cdot \epsilon_r \cdot A}{d}$$
```

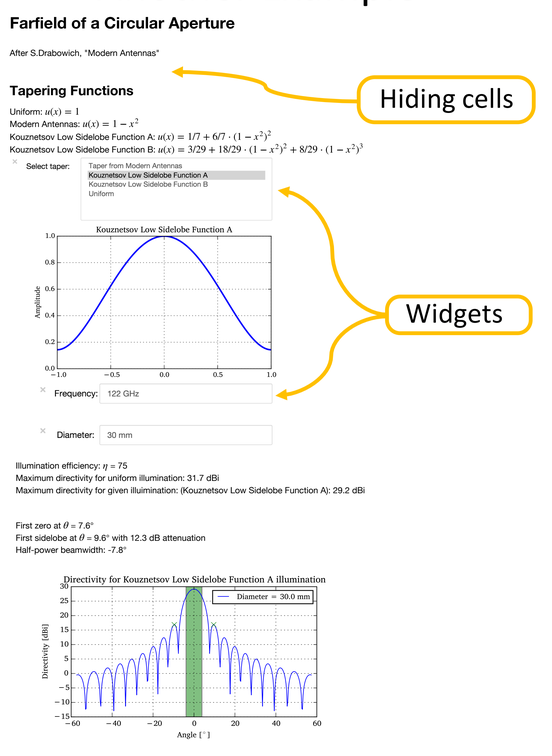
- Calculation

```
In [2]: ρ = 10 Ohm * 1 mm # Material resistivity
        t = 10 um         # Sheet thickness
        L = 150 um        # Structure length
        W = 10 um         # Structure width
        Rs = ρ / t
        R = Rs * L / W
```

- Output in Markdown

```
The capacity is $C$={{ C.pF }}
```

Another Example



Under the Hood

- Widgets

```
In [11]: from IPython.html import widgets
         from IPython.display import display
         from IPython.html.widgets import interact, interactive, fixed
```

```
In [12]: # Define tapering functions
         from __future__ import division
         tapers = {"Uniform": lambda x: 0*x+1.,
          "Taper from Modern Antennas": lambda x: 1-x**2,
          "Kouznetsov Low Sidelobe Function A": lambda x:1/7+6/7*(1-x**2)**2,
          "Kouznetsov Low Sidelobe Function B": lambda x:3/29+18/29*(1-x**2)**2+8/29*(1-x**2)**3,
          }

         taper_widget = widgets.Select(description='Select taper:',
                                       options= list(tapers.keys()) )
```

```
In [3]: # Plot tapering function
        #pyt.figsize(5,2)
        def select_tapering(taper):
            taper_function = tapers[taper]
            u=linspace(-1,1,100)
            axis([-1.1,1.1,0,1])
            plot(u, taper_function(u), lw=2)
            grid()
            title(taper)
            ylabel('Amplitude')

        interact(select_tapering, taper=taper_widget);
```

- Python in markdown

```
In [40]: # Calculate HPBW, first zero and first sidelobe
         import scipy.signal as signal
         minimas = signal.argrelmin(abs(H[0:N/2]))[0]
         maximas = signal.argrelmax(abs(H[0:N/2]))[0]
         theta_min = -theta[minimas[-1]]*180/pi
         theta_max = -theta[maximas[-1]]*180/pi
         sll =  d_max - d[maximas[-1]]
         idx = where(H > 0.5)[0]
         hpbw = 2*theta[idx[0]]*180/pi
```

```
First zero at $\theta$ = {{ '%2.1f'' % theta_min }}<br>
First sidelobe at $\theta$ = {{ '%2.1f' with %2.1f dB attenuation' % (theta_max, sll)}}<br>
Half-power beamwidth: {{ '%2.1f' % hpbw }}
```

Wish List

- Hierarchy, allow to collapse parts of a notebook
- Keep notebook and data together ('projects')
- Versioning, visual diffing
- Templates based on existing notebooks
- Easy Export into something 'looking good'