Jupyter for scientific research

March 20, 2020

Contents

1 Introduction

1.1 Motivation

In the beginning of my academical career I was aiming at an educational degree in math and physics. While studying at the University of Tübingen I also worked as a research assistant at the Institute of Physical Chemistry. In the course of my studies, I had the opportunity to spend 4 months in a school to gain first experiences in teaching physics and math to students between the ages 12 and 20. Even though it was a great experience and I enjoyed bringing new ideas and concepts to the students, I also felt a strong urge to continue learning and being able to explore. At this time the possibilities I saw in focusing on scientific research and development seemed more attractive to me. My reasoning was, that I might be able to combine my preference for teaching and research rather at the university than at school. Long story short, after some years I started my doctoral thesis in the "Institute of Theoretical and Physical Chemistry" at the University of Tübingen. I directly got the possibility to work in a project with an industrial partner. The focus was on building state of the art gas sensors. Besides the benefits of working in a great teams, having enough financial support for research activities and learning to present and report my results on a regular basis, it also imposed some new problems to be solved. One of the biggest challenges was the continuous increase of experimental data in ever shorter intervals. With the increasing industrialization and automatization of sensor production and testing, the quantity of high quality data increased dramatically.

With the increased global interest in "data science", I was not the only one facing problems with the increasing amount of data needed to be processed and analyzed. At the beginning of each analysis, the specific task was not fully defined. It was rather an exploratory research to gain an idea about possible hidden opportunities. In this case the traditional way of creating well defined software algorithms for specific tasks was not adequate.

I rather had the need for efficient tools to reduce time consuming parts regarding the data analysis, which had been performed manually until now. I was looking for a platform to combine multiple tools efficiently. So at this point of my career with my very basic knowledge in the area of programming and the need to solve urgent tasks, I was looking for efficient solutions with a steep learning curve.

Until then my standard procedure of working with data was mainly based on manual feature extraction and analysis. When Working with a limited amount of samples, manually doing these steps was acceptable and efficient. With the industrial cooperation accelerated, the number of samples increased also rapidly. Soon we reached the point where the pre-processing of the data would easily consume a large amount of time. And that without even having started with a deeper analysis. In parallel to this, a project from the Python community gained more and more interest. Articles like [Unp14] pointed me to a new way of dealing with data and indicated, that the Python programming language might be very helpful. Regarding the fact, that Python was already a well established programming language, the introduction of the "I(nteractive)Python notebook ecosystem" was making the use of Python for in an scientific work flow very attractive.

As mentioned in this article related to IPython: [Dav13]

> "… what they do offer is an environment for exploration, collaboration, and visualization."

"Python at Cern" describes for instance the use of Python at the Large Hadron Collider (LHC) at

Cern. As the article mentions, “The interesting bits of data have to be stored, analyzed, shared and published”. “The ease of use and a very low learning curve makes Python a perfect programming language for many physicists and other people without the computer science background”. Python has been used at the LHC now for more than 20 years. Now in 2020 the general technological improvements allows also smaller institutions to work in a comparable way as for instance at the LHC. High computational power combined with large data storage facilities do not require a large financial investment anymore. As a consequence, in many research fields this “data driven scientific research” gains more and more interest. Nevertheless this is a slow process, and this work intends to help improving it.

Already the initial attempts of using Python, the large potential also for my working field was clear. By learning Python I got efficient tools for calculations, analysis and representations. Additionally the new tools have been build focusing on an simple way of reporting result while including the way they have been gained. The environment around the so called “IPython notebooks” was the ideal piece, which I experienced as a missing block in the scientific work I was doing. In 2015 toolset around IPython notebooks have been unified along with other programming languages in a project called Jupyter. In the following part of this work, IPython and Jupyter will be treaded as synonyms.

Besides my work for our industrial partner I also conducted fundamental research about semiconducting metaloxide gas sensors. Based on great research before my time on gas sensors, my focus was now on numerical calculations of the gas sensors. Typically a theoretical model of a gas sensor is developed and the predictions are compared with experimental results. When publishing the results the peer review system assures, that the publications are well written and documented on a solid and proven basis. But when reading such papers, I had the experience, many of them presented excellent ideas, but unfortunately practical instructions on how to implement the presented models were often not given.

The work of rebuilding the model and recalculating the results was therefore often not possible in a reasonable amount of time. In my experience this limits the direct comparison of new model with experimental data from other sensors. It is also worth to mention, that in my experience the average experimental oriented researcher does not have the required programming knowledge to easily implement algorithm based on the presented work. But I am confident, that if an algorithm was given in an appropriate way, most research would benefit from the presented code.

With the results I gained during my PhD. thesis, I was facing the same problem. Others understood the value of my work but could not transfer it to their particular problem. At this point my graphical representation in the form of presentations, my detailed description and the corresponding algorithms of my work had been three separated parts. An appropriated way of representing those results in unified matter did not exists yet. Thus, I decided to try and combine representation, description and algorithm in one unified way.

For this I used the powerful ecosystem around the “Jupyter notebooks” to calculate, represent and describe the results for my thesis for this task. This is why my thesis is written with the large focus on suppling an introduction to this excellent toolbox.

This work will allow more people to gain insight into sensing properties of semiconducting metal oxide sensors (SMOX).Additionally this thesis is intended to be a useful introduction into Python, specially IPython notebooks, for scientific work.

These hopes are not unfounded. While lecturing at the University of Tübingen the course, there

was never enough time to use Python as a supporting programming language in e.g., the introductory course on data mining. In these cases classical tools like Excel (tm) or Origin (tm) were used to analyze example datasets for hidden facts. While most students understood the general concept of data mining, not being able to translate the general concept into machine understandable instructions stopped them from using more advanced tools.

In cases where enough time was available, a short introduction into programming with Python gained a lot of interest and was generally seen as a positive experience. With just a short introduction already many advanced tasks can be performed, which would often even not have been possible with the "traditional" tools.

This thesis is therefore structured in such a way, that I will present my research results from the past years in a condensed form. Additionally I will use this opportunity to introduce and explain the importance of applying programming tools in the common work flow of scientific work.

The potential of "outsourcing" repetitive tasks to machine executable scripts is huge. By focusing on investing more time in creative and intelligent solutions, the overall quality of the results will increase. My hope is to bring with this thesis not only a deeper insight in the understanding of SMOX based gas sensors, but also help others to start a interesting journey into the wide area of data mining and machine learning with python. This work is intentionally structured in a way, to encourage modifications of the original work and simplify this process. The supplied program code is based entirely on well establish standard tools available for all operation systems. This makes this work easily reproducible for others.

The thesis is structured in three major "work packages", each of them dealing with discrete elements of a typical scientific tasks. In a condensed form the following areas will be covered:

1. Numerical solving of differential equations and integrals
2. Using numerical models for simulation
3. Fitting numerical results with experimental data

I typically finish my introductions to Python by letting the students run their first commands. For other introductions found around the globe, this is a program which outputs "Hello World". For Python I prefer to execute other commands which in essence is a philosophical question on how instructions should be formulated.

In a "Jupyter notebook", the next cell represents a "code block" and can be executed along the text of the document. The output of the executed code is then visible along with the text of the document. The `import` statement in Python is used to add functionality or features to the current programming environment. `import this` is a special "easter egg" hidden in every Python version. It adds some Zen to the work space.

[1]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
```

```
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

1.2 Jupyter Notebooks

"The Zen of Python" might not always be the primary directive of each scientist, but the Python community consists most probably of many people who would consider the latter points as important. So did also the inventors of the IPython and Jupyter. A quick search in the world wide web will provide a detailed picture about what "IPython Notebooks" are and how Jupyter in connected with them. Here I will not try to give a general overview of this tool but rather stick to the phrase "Learning by doing". A minimal understanding of working with Python will definitely be helpful for this thesis, but is not necessarily required. I will try to explain the used features this notebooks to guide an interested reader to the point where students can:

- understand the fundamental instructions of Python
- use the basic functionality of the Notebooks
- have a understanding of SMOX based gas sensors

It is worth mentioning, that the intention of such notebooks is to merge the essential tools of scientific work flows with data acquisition, preparation, analysis, representation and documentation all available in one place. The strength of not just sharing final conclusions in a nicely formatted way, but also being able to share the full stack of steps necessary to reach the final conclusion is essentially the strength of the Jupyter notebooks. This feature is already changing the way how scientific results are shared/published and was intentionally designed this way [RPGB17].

The default format of representing anything in a notebook is based on "Markdown". Wikipedia summarizes Markdown like this:

> Markdown is a lightweight markup language with plain text formatting syntax. Wikipedia

This means a document is formatted by writing plain text and special text blocks are interpreted as formatting commands. E.g. **BOLD** letters are generated by encapsulating the text with `** Text here **`, headings are generated by starting the heading with `#`. Depending on the number of `#`, subsections are created. I will not go into detail here about the features of Markdown. Many features are used in this notebook and are directly accessible by double-clicking the text element. By doing this, the formatted text will switch back to the "plain text representation" and will reveal the way it was created. By again executing the cell with `CTRL+ENTER`, its Markdown formatted representation is again rendered.

One other handy feature of the Jupyter ecosystem I use is the ability to transform notebooks into multiple other formats. Just to name some: HTML, WORD (DOC, DOCX), Latex and PDF. The tool `nbconvert` is used internally to convert the Markdown formatted representation into other formats. For this thesis the default option "Export as PDF" under the File option generates a Latex based PDF file. Mastering-markdown is a web page, where I found some hints on how to format my notebook. This example on web page demonstrates for instance how to generate block quotes:

As Kanye West said:

> We're living the future so the present is our past.

To learn how to use notebooks it is best to use them in an interactive environment. The next section will explain how to obtain one for free!

1.2.1 Installation guide

The easiest way to get started would be to use the Anaconda distribution. Anaconda bundles multiple different tools and installs them in the operation system. Anaconda will take care of cross dependencies and handle the update process of the software. This is not the only way to get started with "Jupyter Notebooks" but surely an very fast and easy one. HERE are the slides I use for my lectures to guide students into the world of Python and here one example of it's usage.

1.2.2 Example Notebook - Sneak preview

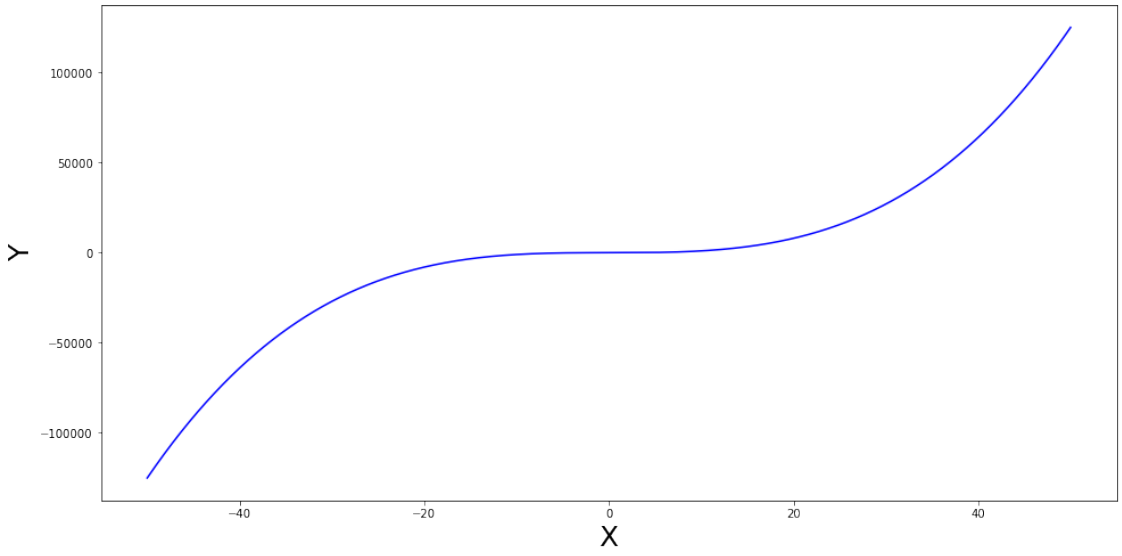
Besides the first example `import this`, here is a very basic example to demonstrate some features of the notebook.

"Simple is better than complex.": The Jupyter environment is equipped with "magic commands" which are not part of the Python programming language, but rather instructions to simplify common tasks. Magic commands always start with `%` and are followed with an instruction. I will demonstrate in this example using the `%pylab inline` instruction. This modifies the current programming space to become a lab nicely equipped for scientific work tasks. For instance, a chemistry lab is commonly equipped with a balance, a water tab and a fire extinguisher. Similarly, a "pylab" is equipped with a data handing, a plotting and a calculation tool (besides many others). The additional parameter `inline` makes sure that the figures will be along with this document. So

let's setup a "pylab" and run some lines of code. (The plotting is handled in the background by Matplotlib [Hun07])

[22]:
```
%pylab inline

#The Python way of adding a comment is by starting a line with the '#' character

#get a list of 5000 points between -50 and 50
xs = linspace(start=-50, stop=50, num=5000)


# Python way of writer 'x to the power of y' is x**y
# here I am calculating the third power of x for 5000 points between -50 and 50
ys = xs**3

#plot it
fig = figure(figsize=(16,8))
plt.plot(xs,ys,'b');

#name axes
plt.xlabel('X', fontsize = 25)
plt.ylabel('Y', fontsize = 25);
```

```
Populating the interactive namespace from numpy and matplotlib
```



The `linspace` function generates a 5000 linear distributed points in the interval [-50,50) and saves those points in the xs variable. ys is a list holding the third power of each point in xs. `plt.` is a submodule defined by the magic command `%pylab` which handels data plotting in a very simple way. `plt.plot(x,y,'r-')` for example plots x vs. y with a red line. The thesis is done in

the notebook to offer the reader the possibility to directly work with newly gained knowledge. Therefore, the upper block is a good opportunity to take the fist steps in Python. For instance, change the line format to 'b' (blue). Or to 'b-.'. Change the exponent from 3 to 3.1. To do so:

- click on the code box above
- edit the code
- Add the following line `plt.title('The power law')`
- go back with CTRL-Z to correct your mistakes
- hit CTRL-ENTER to execute the code

1.3 SMOX based gas sensors

In this section a very short summary about **S**emi conducting **M**etal **OX**ide (SMOX) based gas sensors is provided. Relevant aspects for this thesis will be presented in a general way and additional literature references will be provided. The intention is to provide the minimum amount of information needed to follow this work. The supplied references are a good staring point to gain a detailed understanding of SMOX based gas sensors.

1.3.1 SMOX material

A typical definition of a sensors describes two elements. The receptor and the transducer. The receptor interacts with the external stimulus, while the transducer translates this stimulus into a measurable response. In the case of SMOX sensors those two "parts" can't be separated directly, since the SMOX material takes over the role of both. Generally, the gas sensing with SMOX materials is based on the interaction of target molecules at its surface and a resulting change of the semiconductor itself [BW03].

Nevertheless, the chemistry at the surface, like the adsorption/desorption of molecules, may be seen as the receptor of the sensor. In the process of adsorption/desorption at the surfaces, typically charges from the inside of the material are involved. In general this results in a modification of the charge distribution inside the semiconductor. As a consequence of an increase/decrease of the free charge carrier concentration, the overall resistance of the material changes. The processes related to changes inside the semiconductor, and hence the resistance, could be associated to the "transducer" part of the sensor. A detailed description of the physical and chemical properties that make tin oxide a suitable material gas sensing is described here:"[BD05].

For most SMOX, the intrinsic electrical conductivity is very low due to a high band gap of the semiconductor. Typical preparation methods of the SMOX material generate defects inside the crystal structure due to missing oxygen atoms. For SnO_2 these defects act as additional electron donors and increase the number of free charge carriers and hence increase the conductivity.

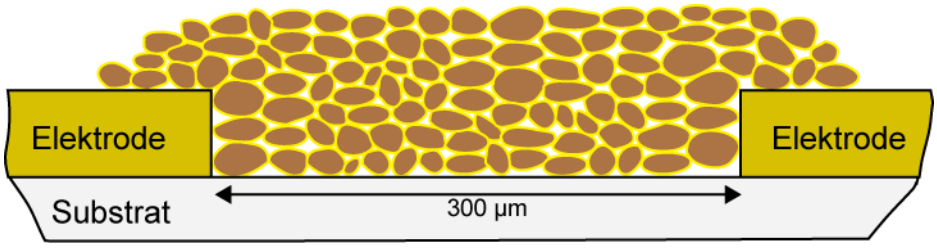
Typically, the density of additional donors N_D or acceptors N_A plays a significant role in the sensing properties. The availability of additional free charge carriers resulting from such defects depends also on the overall temperature of the sensor. Typical operation temperatures are around 300°C, but also higher and lower operations temperatures may be suitable depending on the use case. Besides effecting the semiconductors properties, the reactions at the surface of the sensor are also strongly dependent on the temperature.

1.3.2 SMOX based thick film sensors

The model of this work focuses on describing thick film SMOX gas sensors. In the case of such sensors the SMOX material is deposited as an porous thick film on top of a electric structure suitable to measure the resistance of the film. The film itself consists of multiple nano-sized grains in direct contact with each other. To measure the resistance of the film, a bias voltage is applied and the resulting current which passes through the network of multiple grains is measured. Depending on the percolation path, the total resistance of the sensor R_{Sensor} can be approximated as the product of a geometrical factor G associated to the film morphology and the resistance of one grain R_{Grain} as:

$$R_{Sensor} \propto G * R_{Grain} \qquad \text{(Resistance of sensor)}$$

The following sketch represents the typical setup of a thick film sensor:



1.3.3 Surface potential

To better understand the processes related to the changes in the free charge carrier concentration inside the semiconductor, the semiconductor is typically described in a energy bands representation. For instance for SnO_2, trapping electrons at the surface results in an upward bending of the conduction band. The position of the conduction band at the surface is typically expressed in units $E_S = -eV_S$ (e is the charge of an electron). V_S is called the surface potential. In regard to a non-bended conduction band, the number of electrons are reduced in regions, where the band is elevated.

Experimentally the Kelvin probe method can be used to measure the change in the surface potential ΔV_S. The experimental setup and the requirements to obtain the value of ΔV_S are described in detail here: [OBW09]

The relation between ΔV_S and the resulting change in the resistance of the semiconductor R is then only determined by the semiconductor. The relation between ΔV_S and the resistance R of the semiconductor is defined independently of the actual surface chemistry. The relation depends strongly on the temperature T, the defect concentration N_D and the shape of the underlaying material. This work focuses on deriving a model to predict the theoretical value of $\Delta R_{V_S} = \frac{R_{V_S}}{R_{V_S=0}}$ from a change in the surface potential ΔV_S.

With the definition of (Resistance of sensor) the value of ΔR_{V_S} of one grain can be directly calculated by the ratio of the experimentally measured value of the sensors R_{Sensor}.

$$\frac{R_{Sensor_{V_S}}}{R_{Sensor_{V_S=0}}} = \frac{G * R_{Grain_{V_S}}}{G * R_{Grain_{V_S=0}}} = \Delta R_{V_S} \qquad (1)$$

1.3.4 Numerical model

Based on the example of SnO_2 as SMOX material, a numerical model describing effects of shape, size, defect concentration and temperature is developed for spherical grains. With such a model, multiple conclusions about the dependency of these parameters on the sensitivity and performance of a sensor can be derived. Additionally, the numerical model is able to provide the number of charges involved in the sensing process. This number is an important parameter involved in the description of the chemical reaction at the surface.

When solving this problem numerically, some assumptions will need to be introduced. These assumptions simplify the problem to a level where a numerical calculation is possible, but will reduce the validity of the results only to a small subset of all possible situations. The calculations in this thesis are also packed with assumptions and boundary conditions. My intention is to supply enough information to understand the relevance of the assumption and it's implications along with the supplied code. The way of presenting this knowledge should lead/motivate others to adapt the presented work to individual other cases with different boundary conditions.

2 Summary

Since the motivation for this "interactive" thesis should be clear now, I would like to now move to my actual research topic in the next section:

"A parametrized numerical model to simulate the semiconductor influence of thick film metal oxide gas sensors".

In the next notebook I will demonstrate how theoretical numerical calculations of chemical sensors are used to better understand experimental results.

Follow this link to come to the next section.

3 About the PDF-Version of this work

This notebook was not intended to be used as a printed hard copy or as a PDF. Its main intention is to serve as an easy entry point to Python supported science. Many of the implemented features in these notebooks like interactive widgets, animated data representations and live code examples will not work in the PDF-version. Only a static snapshot of the interactive representation can be represented outside the Jupyter notebook, at best. The benefits of interacting with the presented work in a notebook should motivate the reader to use the notebook.

Nevertheless, the integrated function to export a notebook to a latex based version offers a very nice way to publish results in a printable way. So please keep in mind that the PDF-version may not be able to represent all the features of the notebook as intended and some links might not work as expected. You are strongly encouraged to switch to the Jupyter presentation of this work and experience the full potential of such notebooks. A snapshot of the current folder can be downloaded here: Num_smox_sensor.zip

3.1 Equations

In a typical scientific thesis and textbook, relevant equations are referred by numbers of identifiers. In Latex this is done by assigning a label and a tag to an equation. If this equation needs to be

referred to, a pointer to the reference is added, and the tag is used for the representation of this equation. As an example, an arbitrary equation from this thesis is used. The (internal) reference of this equation is 'example', while the printed representation (tag) is "Example Equation".

$$\frac{dV^*}{dr^{*2}} = 1 - n^*(V^*) - \frac{2}{r^*}\frac{dV^*}{dr^*} \tag{Example Equation}$$

To refer to this equation, a reference can be added which will look like this (Example Equation). The underlying mechanisms will link the reference with the equation, print the label, and add an hyperlink allowing to directly jump to the equation. This feature works either in notebook or PDF representations. One drawback of the notebook representation is, that the equation and reference need to be in the same code block/cell. Otherwise the reference is not working. Instead of linking the equation correctly, ??? will be represented. My opinion is that this will change with future versions of Jupyter. Since the PDF version is processed by a Latex interpreter in total, the "one code block" limitation does not exist there. To demonstrate this, I will try to reference the equation in the next code block.

Here the same reference to the equation: Example Equation. In the PDF-Version this will work correctly and in the Notebook-Version only ??? should be visible (until now).

3.2 Tables

With the numerical calculations performed in this thesis, data will need to be represented also. One way of doing this is by plotting the data. This is also suitable for a static PDF-Version of this thesis, which often is useful/necessary to have. This feature is well implemented and the transfer from Notebook to PDF version works well. Another way to display data are tables. Similar to the well known, omnipresent tool for working with tables, Microsoft Excel (tm), the Python universe has its own but similar tool. It's called Pandas. As a primer, I will give here a very short introduction to Pandas. What sheets are for Excel, Dataframes are for Pandas. Here a simple example how to create a Dataframe in analogy to the previous programming example:

[2]:
```
%pylab
import pandas

#create some x values
x = linspace(start=0,stop=5,num = 10)

#the y values will the the square of the x values
y = x**2

#Put them in a Dataframe and reference this new Dataframe with the variable `dF`
dF = pandas.DataFrame({'x':x, 'y':y})
```

```
Using matplotlib backend: Qt5Agg
Populating the interactive namespace from numpy and matplotlib
```

This simple example of a Dataframe should demonstrate the basic concept of Dataframes. Once data is in the `DataFrame` format, there are infinite ways to transform/slice/merge/... it, to bring

it into the desired shape. Along this thesis, some of the functionalities of Dataframes will be used and explained. Since the Jupyter environment is still "under construction", the transfer from Juypter to PDF for notebooks does not work well until now. This does not mean that is is not possible, it is just not yet implemented in the default/vanilla environment. This is why I want to highlight the small tweak that is at this point still needed to have a comparable output in Juypter notebooks and printed PDFs. To bypass this obstacle, the default behavior of pandas needs to be altered (similar to this post: Latex-Tables Monkey Patch). This following patch brings DataFrames in the appropriate shape in both representations.

3.2.1 Before the patch

[3]:
```
display(dF)
```

	x	y
0	0.000000	0.000000
1	0.555556	0.308642
2	1.111111	1.234568
3	1.666667	2.777778
4	2.222222	4.938272
5	2.777778	7.716049
6	3.333333	11.111111
7	3.888889	15.123457
8	4.444444	19.753086
9	5.000000	25.000000

3.2.2 The patch

[4]:
```
import pandas
pandas.set_option('display.notebook_repr_html', True)

def _repr_latex_(self):
    return r"""
    \begin{center}
    {%s}
    \end{center}
    """ % self.to_latex()

pandas.DataFrame._repr_latex_ = _repr_latex_  # monkey patch pandas DataFrame
```

3.2.3 Prettier tables after the patch

[5]:
```
display(dF)
```

	x	y
0	0.000000	0.000000
1	0.555556	0.308642
2	1.111111	1.234568
3	1.666667	2.777778
4	2.222222	4.938272
5	2.777778	7.716049
6	3.333333	11.111111
7	3.888889	15.123457
8	4.444444	19.753086
9	5.000000	25.000000

In Jupyter notebooks the output will look very similar. In the PDF version of this thesis, those two output will differ(, unless a newer version of Jupyter fixed this issue.) To have this thesis in a PDF printable form, I will use the presented patch to unify the output. On the other side, I suggest not to use this patch and rather work with the Jupyter notebooks and its default representation, since the patch has also its downsides. But this would go beyond the scope of this remark about the patch.

4 Bibliography section

References

[BD05] BATZILL, Matthias ; DIEBOLD, Ulrike: The surface and materials science of tin oxide. In: *Progress in Surface Science* 79 (2005), Nr. 2-4, S. 47–154. http://dx.doi.org/10.1016/j.progsurf.2005.09.002. – DOI 10.1016/j.progsurf.2005.09.002. – ISBN 0079–6816

[BW03] BÂRSAN, N. ; WEIMAR, U.: Understanding the fundamental principles of metal oxide based gas sensors; the example of CO sensing with SnO2 sensors in the presence of humidity. In: *Journal of Physics Condensed Matter* 15 (2003), Nr. 20, R813–R839. http://dx.doi.org/10.1088/0953-8984/15/20/201. – DOI 10.1088/0953–8984/15/20/201. – ISBN 0953–8984

[Dav13] DAVENPORT, Thomas H.: *The Rise of Data Discovery*. https://www.datanami.com/2016/05/04/rise-data-science-notebooks/. Version: 2013

[Hun07] HUNTER, John D.: Matplotlib: A 2D graphics environment. In: *Computing in Science and Engineering* 9 (2007), may, Nr. 3, 99–104. http://dx.doi.org/10.1109/MCSE.2007.55. – DOI 10.1109/MCSE.2007.55. – ISSN 15219615

[OBW09] OPREA, Alexandru ; BÂRSAN, Nicolae ; WEIMAR, Udo: Work function changes in gas sensitive materials: Fundamentals and applications. In: *Sensors and Actuators, B: Chemical* 142 (2009), Nr. 2, 470–493. http://dx.doi.org/10.1016/j.snb.2009.06.043. – DOI 10.1016/j.snb.2009.06.043. – ISSN 09254005

[RPGB17] RANDLES, Bernadette M. ; PASQUETTO, Irene V. ; GOLSHAN, Milena S. ; BORGMAN, Christine L.: Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study. In: *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries* (2017).

http://dx.doi.org/10.1109/JCDL.2017.7991618. – DOI 10.1109/JCDL.2017.7991618. – ISBN 9781538638613

[Unp14] UNPINGCO, José: *Python for signal processing: Featuring IPython notebooks*. Bd. 9783319013. Cham : Springer International Publishing, 2014. – 1–128 S. http://dx.doi.org/10.1007/978-3-319-01342-8. http://dx.doi.org/10.1007/978-3-319-01342-8. – ISBN 9783319013428