Numerical calculations of SMOx based gas sensors with Python

November 29, 2019

Contents

1 Introduction

1.1 Motivation

When I first started working in the field of metal-oxide semiconducting gas sensors, I still had the wish to finish my time at university with a degree of eduction in math and physics. But while having spend 4 month in a school and experiencing the joy of teaching, I also felt a strong urge to continue learning and explore the possibilities science and research would allow me. Long story short, I ended up starting my Phd. in the "Institute of Theoretical and Physical Chemistry" at the University of Tübingen. Lucky as I was I directly got the possibility of working on a project with an industrial partner about gas sensors. This allowed me to investigate the sate-of-the-art of industrialized sensor production. Besides the benefits of learning to give regular presentations and reports, working in teams and funding for interesting research activities, it also imposed some new problems needed to be solved. The biggest of challenges has been the continuous increase of sensor date in always shorter intervals. Luck as I was, I was not the only one facing problems with the increasing amount of data needed to be analyzed. Aside from the conventional way of

wiring applications/programs, the need was rather on having efficient tool sets to reduce time consuming parts of the data analysis. Especially those which could theoretically be done by an algorithm. At this point my very basic knowledge in the area of programming and the need to solve urgent tasks, created the need for efficient solutions with a steep learning curve. Until then my, and of some others in the lab, standard procedure of working with our data was:

- first to extract relevant features at well defined time-points to reduce the initial amount to a reasonable quantity of data points
- in a second step a unified representation of the extracted features had to be created
- analysis of the data and taking further steps

When Working with a limited amount of samples, manually doing these steps was acceptable. With the industrial cooperation gaining speed, 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 the analysis. Luckily I was not the only one facing such problems and a project from the Python community gained more and more interest. Articles like [**?**] pointed me to a new way of dealing with data and using the Python programming language. Regarding the fact, that Python was already a well estblished programming language, the introduction of the "IPtyhon notebook ecosystem" was making the use of Python for in an scientific workflow very attractive. As mentioned in this article: [**?**]

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

I also realized the large potential for my research field. Not only that I got efficient tools for calculations, analysis and representations, but the new tools have been build with the focus of easy reporting results. Such an environment around the so called notebooks was the ideal piece, which I experienced as a missing block in the scientific work I was after. Besides my work for our industrial partner I also did fundamental research about semiconducting metaloxide gas sensors. On the basis of the great works on gas sensors, my focus was on one the numerically calculation of the gas sensors semiconducting properties. Typically a model of a gas sensor is developed and compared with experimental results. The pees review system assures, that the publications published in the journals are well written, documented the on a solid and proven basis. Unfortunately the transfer from the presented model in the publication to other model and a direct comparison of the results experimental data aired on other sensors is often not done. One of the main obstacles for the average experimental oriented researcher is the lack of the programming knowledge to implement algorithm based on the presented work.

With the results I gained while my Phd. thesis I had the experience, that others understood the value of my work but could not transfer it to their particular problem. At this point I decided to orient myself back to my educational roots and use the powerful ecosystem around the IPython notebooks and write my thesis with the large focus on suppling a introduction to this excellent toolbox.

The presented results will allow more people to gain insides in the semiconductors influence on the sensing properties. More than this I hope this thesis will also be a useful introduction into Python, specially IPython notebooks, for scientific work.

These hopes are not unfunded.In my time working at the "Eberhard Karls University of Tübingen" i was also assigned to give multiple lectures. One lecture branch has been the introduction to data mining. Often the lack of programming knowledge and the limited amount of time did not allow

a usage of Python as a supporting programming language. In these cases classical tools like Excel of Origin have been used to mine data examples for hidden facts. Most attendees have been very fast and understood the general concept of data mining. Just the lack of a the minimal requirement to translate the concept into machine understandable instructions stopped them from using them. When enough time was available, a short introduction into programming with Python gained a lot of interest and was generally seen as a positive experience. This thesis is therefore structured in such a way, that one the one side I will present my research results from the past years in a condensed form and on the other side uses this the opportunity to introduce and explain the importance of applying programming tools in the common work flow of scientific work. Since the potential of "outsourcing" repetitive tasks to machine executed scripts and invest more time in our ceativity and intelligence is huge. 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.

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

[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 SMOX based gas sensors

This section list a collection of high quality publications which will broadcast all relevant information about semiconductor based gas sensors directly into your brain! In the course of this thesis many of those aspects will be explained, analyzed in discussed in required detail at the relevant places. So reading these publications now is not necessary to follow this thesis. This section is

therefore a good point to come back if a more detailed view on the subject is desired. Especially when this thesis is used to apply the presented concepts to individual research topics.

When solving numerical problems it typical to introduction some assumptions, which are only valid under specific boundary conditions. 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. With my presented work I do not claim to calculate all aspects of SMOX gas sensors, but present a tool which helps to understand specific aspects. The way of presenting this knowledge should lead/motivate others to adapt the presented work to individual other cases with completely different boundary conditions. A general understanding of gas sensors will only be gained to a certain level, but with time, work and data this understanding will definitely grow.

1.3 Notebooks

"The Zen of Python" might not always be the primary directive of each developer, but the Python community consists most probably of many people how would consider the latter points as important. So did also the inventors of the IPython and Jupyter. A quick search will reveal multiple sources in the world wide web giving a detailed picture about what Notebooks are and how Jupyter in connected with them. Here I will not try to give an general overview about this tool and rather stick to the phrase "Learning by doing.". By explaining topic related parts this notebooks will guide an interested reader to the point where:

- understanding the fundamental instructions of Python
- using the basic functionality of the Notebooks
- fundamental understanding of SMOX based gas sensors

is gained.

It is worth mentioning, that the intention of such notebooks is to merge the essential tools of scientific work flows together. Data acquisition, preparation, analysis, representation and documentation should 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 [**?**].

The formatting used in this 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 plaintext 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. The plain text will reveal the way it was created, and the execution of the cell with `CTRL+ENTER` will reveal its Markdown formatted representation. 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. 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. For instance I gained the ability to make block quotes from this page based on this example:

As Kanye West said:

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

1.3.1 Installation guide

The easiest way to get started would be to use the Anaconda distribution. Anaconda bundles multiple different tools and installs them i nthe operation system. Anaconda will take care of cross dependecies 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 is additionally a presentation I use for my lectures to guide students into the world of Python and here one example of it's usage.

1.3.2 Example Notebook - Sneak preview

Besides the first example `import this`, here is a very basic example which should prepare exited reader on whats coming next.

"Simple is better than complex." The Jupyter environment is equipped with "magic commands", which are not part of the base programming language (i.e. Python), but rather a helper instruction to simplify common tasks. Magic commands always start with `%` and are followed with an instruction. I will demonstrate in this example the use of the `%pylab` instruction. This modifies the current programming space to be a lab nicely equipped for scientific work flows. For instance a chemistry lab is equipped with a balance, a water tab and a fire extinguisher, and in this case a "pylab" is equipped with data handing, plotting and multiple other tools. So let's setup a "pylab" and run some lines of code. (The plotting is handled in the background by Matplotlib [**?**])

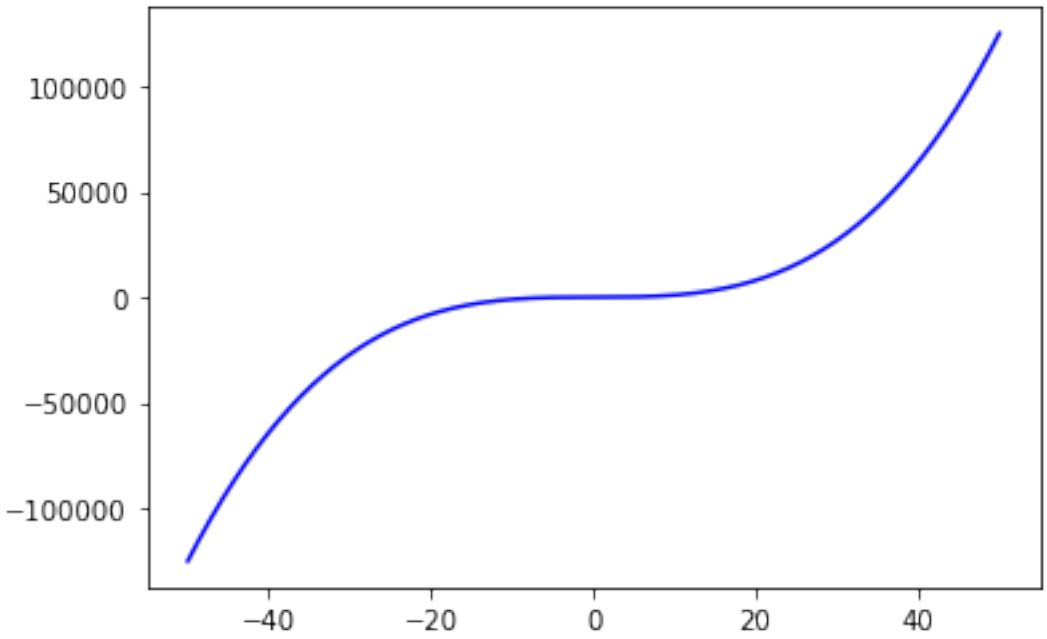
[2]:
```
%pylab inline

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

```
# Python way of adding a comment is the '#' character
# Python way of writer 'x to the power of y' is x**y
# here I am calculating thethe third power of x for 5000 points between -50 and␣
 ↪50
y = x**3

#plot it
fig = plt.plot(x,y,'b')
```

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



The additional "inline" instruction in `%pylab` configures the plotting environment to represent the graphics inside this text. The following line generates a 5000 linear distributed points in the interval [-50,50) and y is just the third power of each point. `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 do 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
- edit the field
- 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

2 Conclusion

Since the motivation for this "interactive" thesis should be clear now, I would like to come in the next section now to my actual research topic: "Numerical calculation of semiconducting metal oxide (SMOX) based gas sensors". In the next chapter I will demonstrate how theoretical numerical calculations for chemical sensors are used to better understand experimental results. In the following chapter I will demonstrate how such knowledge could be used to improve the performance of a sensor regarding it's sensitivity and selectivity.

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. The provided PDF servers just as an low level representation of the original work. It should give potentially interested readers an easy entry point to Python(any other programming language) 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 mutable representation can be represented outside the notebook. On the one side the reduction of printed pages and 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 the notebook to a Latex based PDF results in a very nice static publication. So please keep in mind, that the PDF-verison 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.

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, the 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 'second_derivative', while the printed representation (tag) is "Second derivative".

$$\frac{dV^*}{dr^{*2}} = 1 - n^*(V^*) - \frac{2}{r^*}\frac{dV^*}{dr^*} \qquad \text{(Second derivative)}$$

To refer not to this equation a reference can be added which will look like this (Second derivative). The underlying mechanisms will link the reference with the equation, print the label, and add an hyperlink, to be able to jump to the equation. This feature works either in notebook or PDF representations. One drawback of the notebook representation is, that he 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 will evolving improvements of Jupyter is is just a temporal flaw. 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 eqaution in the next code block.

Here the same reference to the equation: Second derivative

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 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, Excel, 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, are Dataframes for Pandas. Here a simple example how to create a Dataframe in analogy to the previous programming example:

[3]:
```
%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, the are infinit 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 to 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 way 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 to by nicely represented in both representations.

3.2.1 Before the patch

[4]:
```
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

[5]:
```
import pandas
pandas.set_option('display.notebook_repr_html', True)

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

[6]:
```
display(dF)
#"Test Tabel"
```

	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

[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

[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