

Working with Jupyter Notebooks

Andrew Binley & Guillaume Blanchy

1. What is a Jupyter Notebook?

[Jupyter notebooks](#) (.ipynb) are text files that can be viewed as interactive web page in a browser. A Jupyter notebook consists of different cells that can contain code (such as Python) or text (such as Markdown). The code cells can be run in the browser and their output directly viewed. The notebooks can be considered as scripts or macros, allowing the user to rerun with different settings, or apply to other datasets. As such they are ideal for exploratory analysis or teaching. Jupyter notebooks need to be served by a server that can be run locally (<http://localhost:8888/tree>) or remotely (<https://mybinder.org>).

2. How to install a local copy?

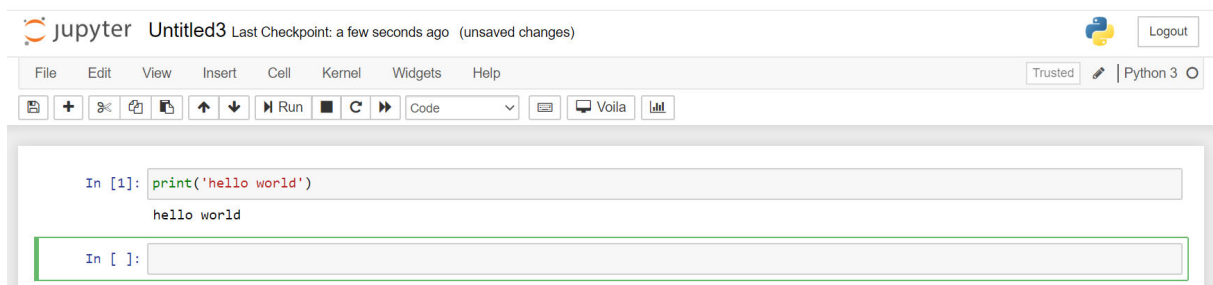
Running Jupyter notebooks locally (on your own computer) requires that you have a Python environment installed. For Windows user, we recommend the installation of the [WinPython](#) distribution (note that [anaconda](#) distribution is also an option). Once downloaded and installed, go the WinPython directory. To run the notebook just click on Jupyter Notebook .exe. The notebooks will be stored inside the “notebooks” folder.

3. Running the notebook

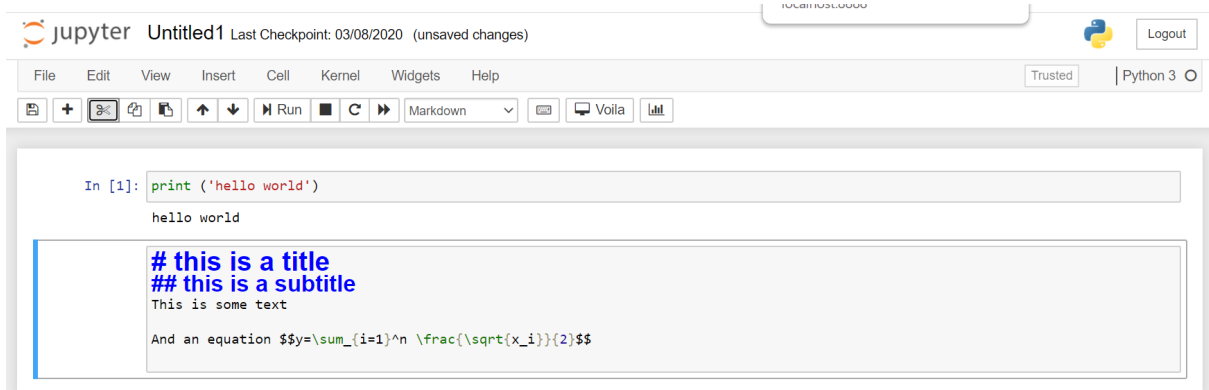
After running the Jupyter Notebook .exe file, the home webpage should open in your default browser (see figure below). From there you can create a new notebook with code and text cells or import one (upload) from your file system. To create a new notebook select New and Python 3.



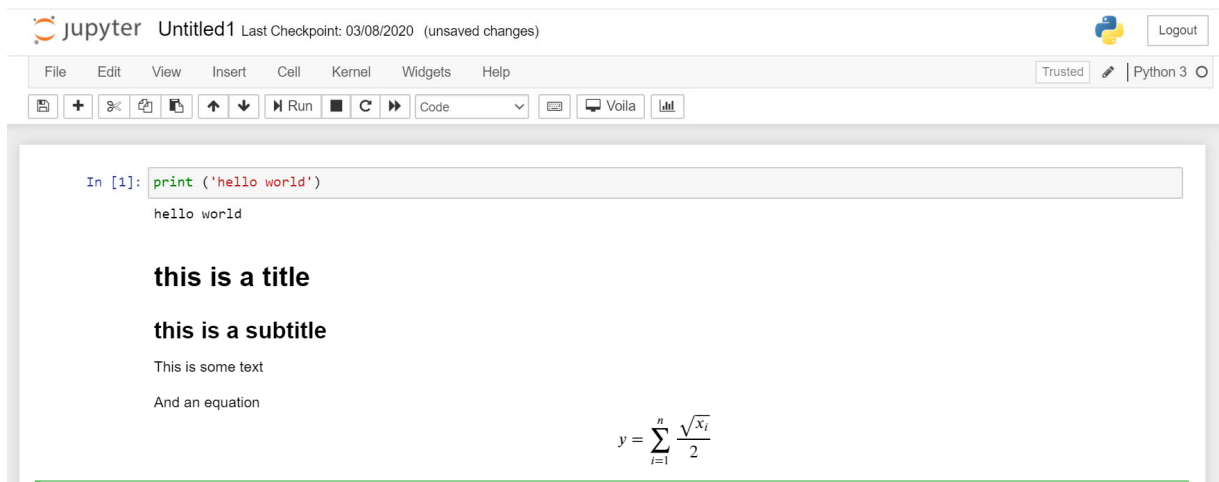
Add the text `print ('hello world')` and then Run the notebook.




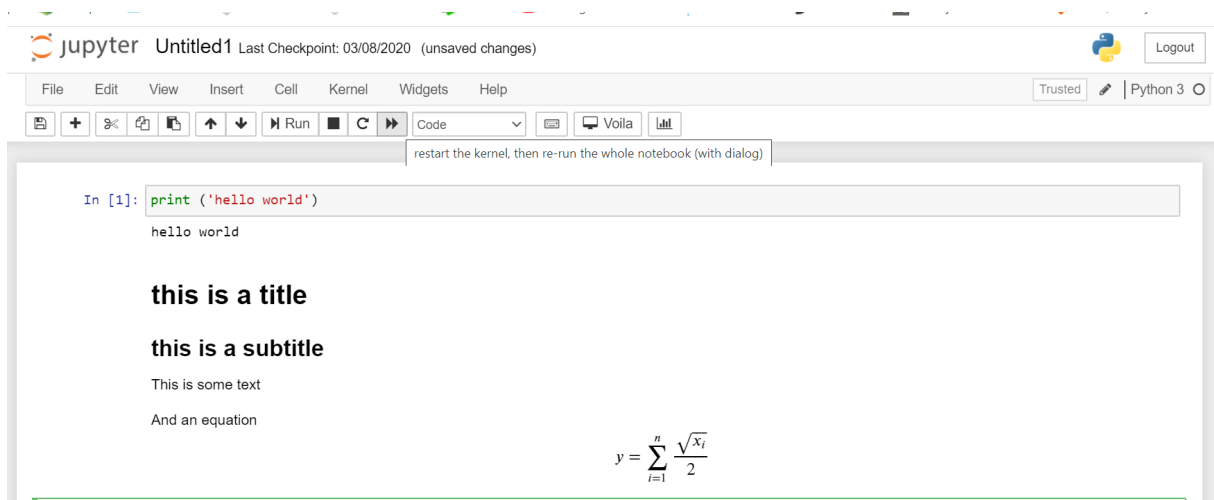
Then add the Markdown text shown in the figure below.



When you run it you should see the formatting below.



You can run each section of the notebook step by step with the Run command (or SHIFT+enter) or you can restart and run the entire notebook with the  command (see below).

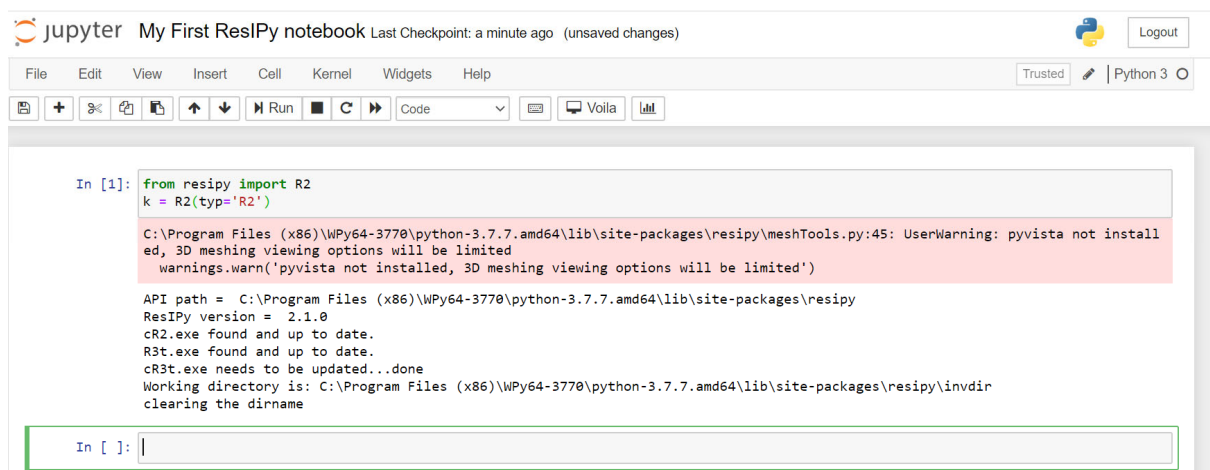


You can rename the notebook by clicking on the name (in this example Untitled1) at the top of the page. The notebook will be saved as a .ipynb file but you can also select Download from the File menu and to save in .html format, which can be useful for displaying later and/or sharing.

4. Running ResIPy in a notebook

To install the “resipy” package which contains the Python application interface (API) around the R family of codes open the WinPython Command Prompt.exe and type `pip install resipy`. If you are planning to run 3D visualization, we also recommend that you install `pyvista` (`pip install pyvista`) – this gives additional plotting options. Note that these commands only need to be run once, i.e. when you have installed WinPython but if you need to install an updated version of ResIPy then you can use `pip install -U resipy`

If you are creating a new notebook for ResIPy the first command (i.e. code section) will be from `resipy import R2` which will link to the suite of R family of codes The first thing to do is to create an instance of the R2 class (also called an ‘object’), that we will call 'k '. This object will be used throughout. Assuming we are working with R2 then we need the command `k=R2(typ='R2')` or `k=R2()` since R2 is the default type. When you run this you should see something like the screenshot below.



```
In [1]: from resipy import R2
k = R2(typ='R2')

C:\Program Files (x86)\WPY64-3770\python-3.7.7.amd64\lib\site-packages\resipy\meshTools.py:45: UserWarning: pyvista not installed, 3D meshing viewing options will be limited
  warnings.warn('pyvista not installed, 3D meshing viewing options will be limited')

API path = C:\Program Files (x86)\WPY64-3770\python-3.7.7.amd64\lib\site-packages\resipy
ResIPy version = 2.1.0
cR2.exe found and up to date.
R3t.exe found and up to date.
cR3t.exe needs to be updated...done
Working directory is: C:\Program Files (x86)\WPY64-3770\python-3.7.7.amd64\lib\site-packages\resipy\invdir
clearing the dirname

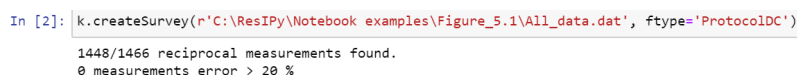
In [ ]: |
```

The notebook we will build will contain a series of methods of the form `k.text()`, for example, `k.createMesh()`. The method `k.createMesh()` is a function of a class `k`, defined above. You can get help on the attributes/parameters of each of these by typing, for example, `help(k.createMesh)`. A list of methods/functions can be found at <https://hkex.gitlab.io/pyr2/api.html>.

We will begin with reading in a dataset in the standard R2 format. Assuming a Windows user with the file `protocol.dat` in the folder `C:\ResIPy\Notebook examples\Figure_5.1\`, we can read this in with

```
k.createSurvey(r'C:\ResIPy\Notebook examples\Figure_5.1\protocol.dat',
ftype='ProtocolDC')
```

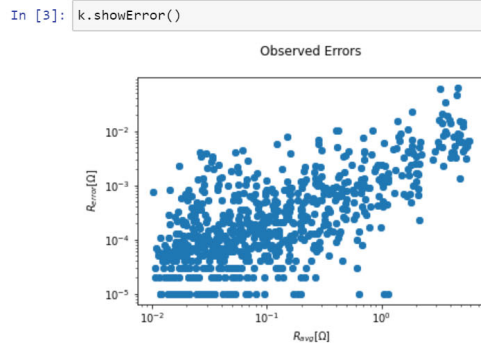
Note the `r` prefix to the file name is needed in Windows because of the `\` in the file name.



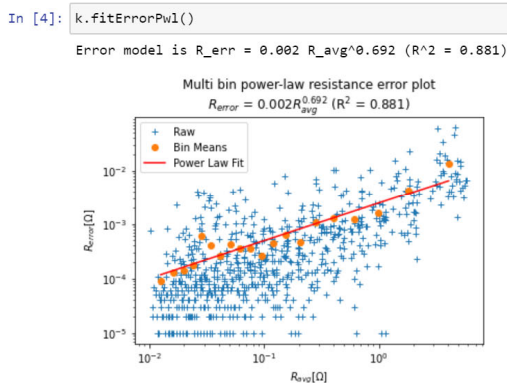
```
In [2]: k.createSurvey(r'C:\ResIPy\Notebook examples\Figure_5.1\All_data.dat', ftype='ProtocolDC')

1448/1466 reciprocal measurements found.
0 measurements error > 20 %
```

As this file contains a set of normal and reciprocal measurements we can display the reciprocal errors using `k.showError()` – this should give the following plot

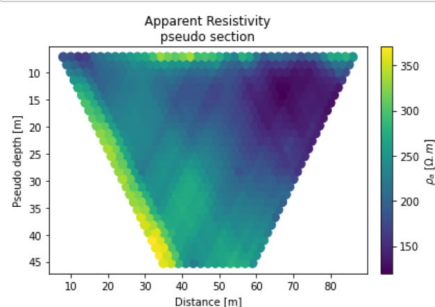


We can fit an error model to this plot using `k.fitErrorPwl()`



We now read in the electrode coordinates from the file `electrodes.csv` with `k.importElec(r'C:\ResIPy\Notebook examples\Figure_5.1\electrodes.csv')` and then plot the pseudosection with `k.showPseudo()`

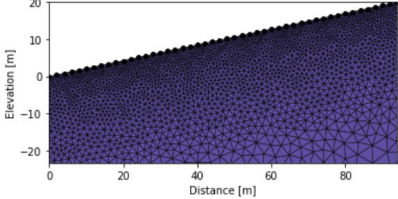
```
In [5]: k.importElec(r'C:\ResIPy\Notebook examples\Figure_5.1\electrodes.csv')
In [6]: k.showPseudo()
```



A triangular mesh is then created with `k.createMesh(typ='trian')` and we can display the mesh with `k.showMesh()`

```
In [7]: k.createMesh(typ='trian')
k.showMesh()

Creating triangular mesh...Reading mesh.msh
Gmsh version == 3.x
reading node coordinates...
Determining element type...Triangle
Reading connection matrix...
ignoring 0 elements in the mesh file, as they are not required for R2/R3t
Finished reading .msh file
ResIPy Estimated RAM usage = 0.045903 Gb
done
```



To invert the data we type `k.invert()`. Before doing this we have to specify `k.err = True` to ensure that the fitted error model is used in the inversion, otherwise default settings will be applied.

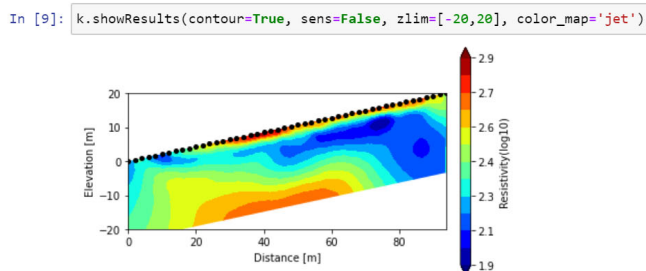
R2 will then run and display its output log.

```
In [8]: k.err = True # setting this flag is necessary for adopting the fitted error model, otherwise a default error is set
k.invert()

Writing .in file and protocol.dat... done!
----- MAIN INVERSION -----

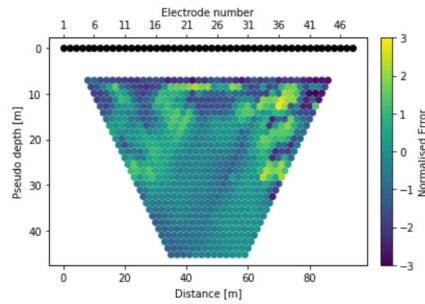
>> R 2   R e s i s t i v i t y   I n v e r s i o n   v4.0 <<
```

When R2 is complete, to display the results we can type `k.showResults(contour=True, sens=False, zlim=[-20,20], color_map='jet')`



Running `k.showPseudoInvError(vmin=-3, vmax=3)` allows us then to look at the distribution of normalized errors in a pseudosection

```
In [10]: k.showPseudoInvError(vmin=-3, vmax=3) # the range is set as -3 to 3
```



The entire notebook for the above is saved as `Figure 5.1 notebook.ipynb` in the notebook examples folder

5. Other notebooks

In the notebook examples folder the following Jupyter Notebooks are included (along with html versions that can be viewed in any browser):

`Forward modelling.ipynb` – this illustrates how to do forward modelling of a synthetic model and subsequent inversion of the forward modelled data. The notebook is setup to allow the user to explore the effect of different quadrupole geometries and data noise on the recovery of the anomalies created in the synthetic model.

`Time-lapse inversion.ipynb` – this illustrates how to time-lapse inversion using the problem illustrated in Figure 5.21.

`3D inversion.ipynb` – this illustrates a 3D inversion using the problem covered in Tutorial 06 of the ResIPy graphical interface tutorials. This example shows how to use the pyvista plotting options for 3D problems.

Many more example ResIPy Jupyter Notebooks are available at https://hkex.gitlab.io/pyr2/auto_examples/index.html

Andrew Binley & Guillaume Blanchy
Lancaster University
August 2020