



Jupyter and IPython
facilitating open access
and reproducible research

Min Ragan-Kelley, Simula Research Lab



How do we do

computational research?



How do we do

computational research?

1. write code



How do we do

computational research?

1. write code

2. run code



How do we do

computational research?

1. write code
2. run code
3. make figures



How do we do

computational research?

1. write code
2. run code
3. make figures
4. write paper



How do we do

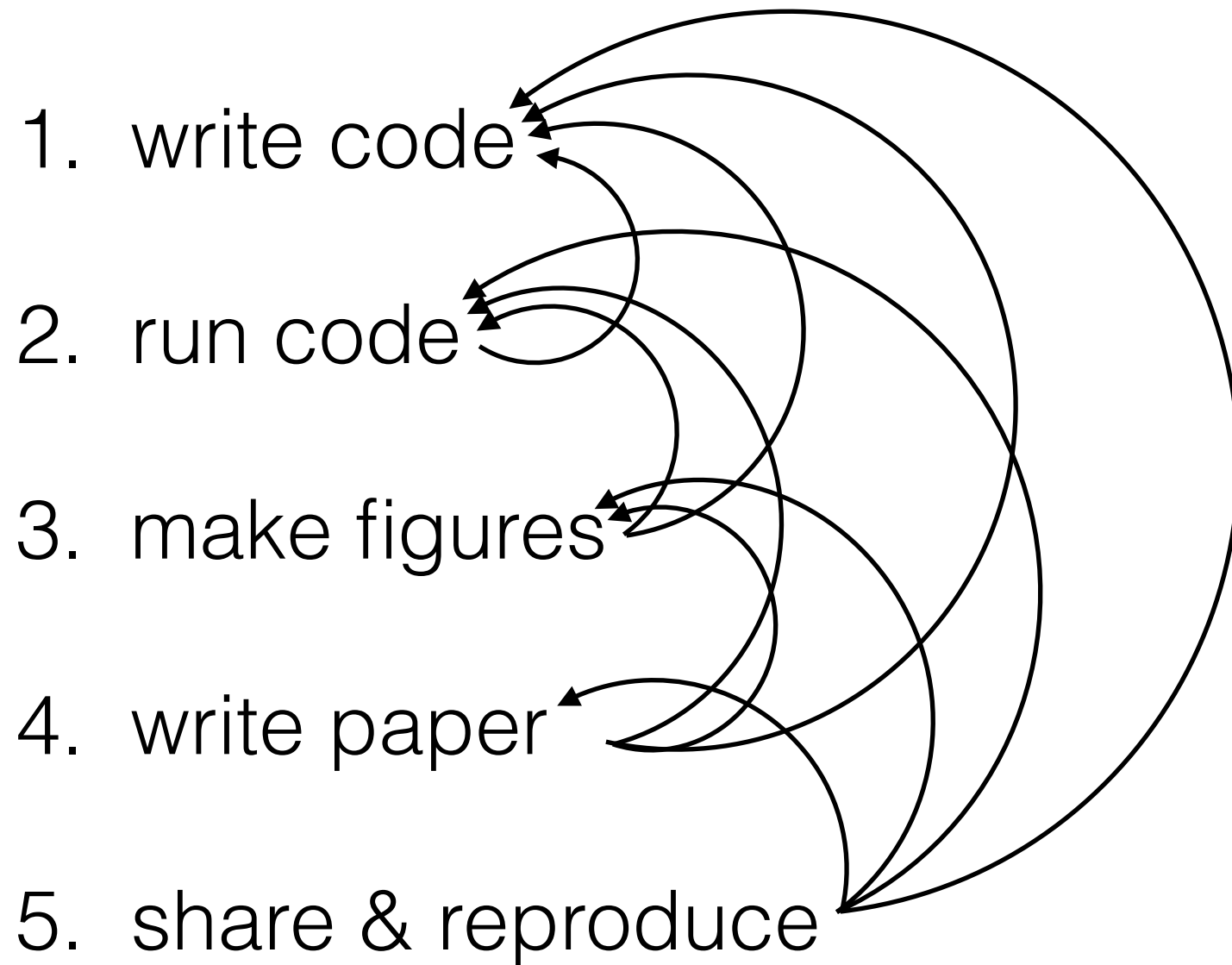
computational research?

1. write code
2. run code
3. make figures
4. write paper
5. share & reproduce



How do we do

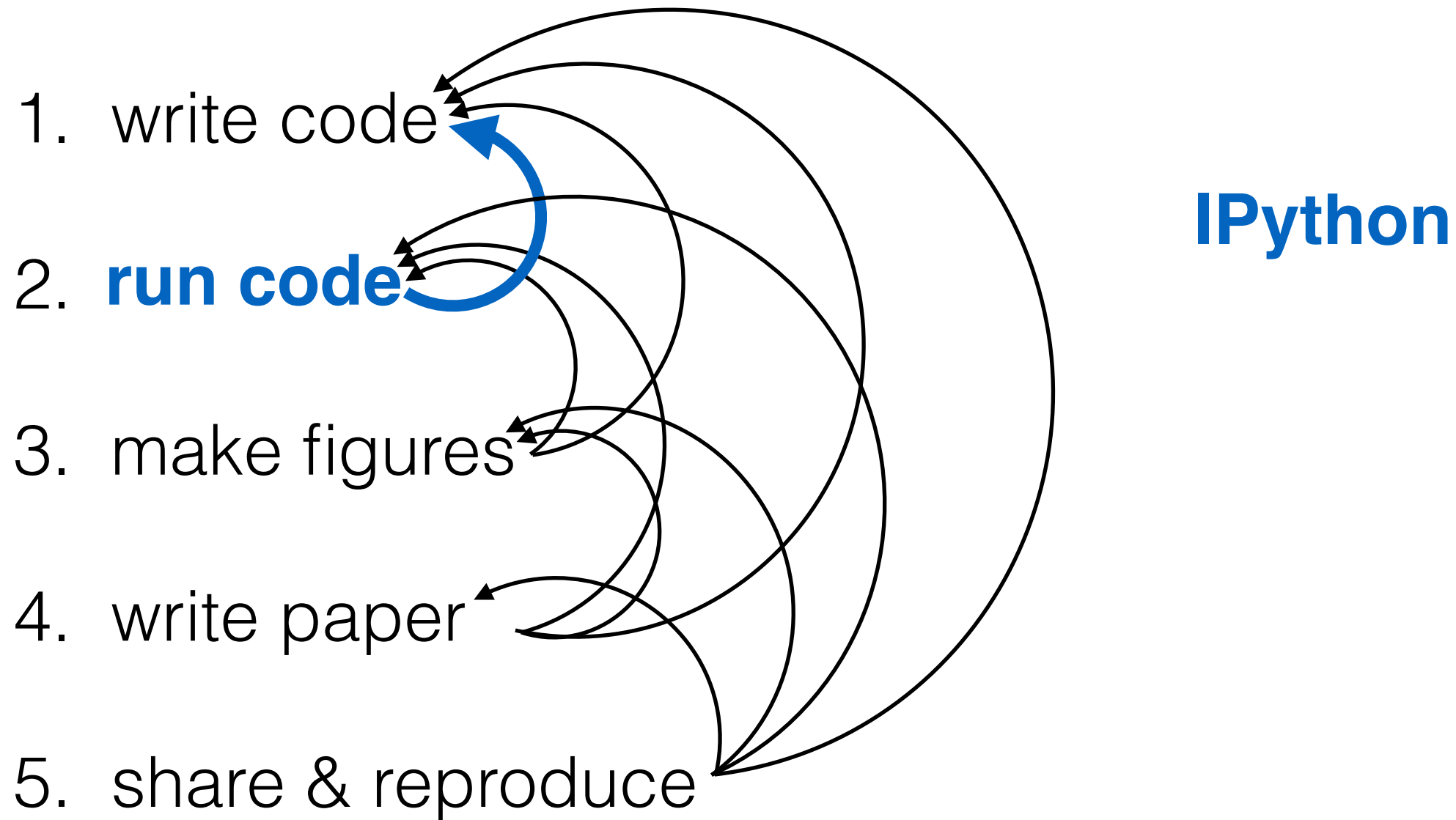
computational research?





How do we do

computational research?





IPython

Interactive Python

helps run code

```
minrk[02:13]~/Documents/Jupyter/pres/AGU-2014 $ ipython
```



IPython

Interactive Python

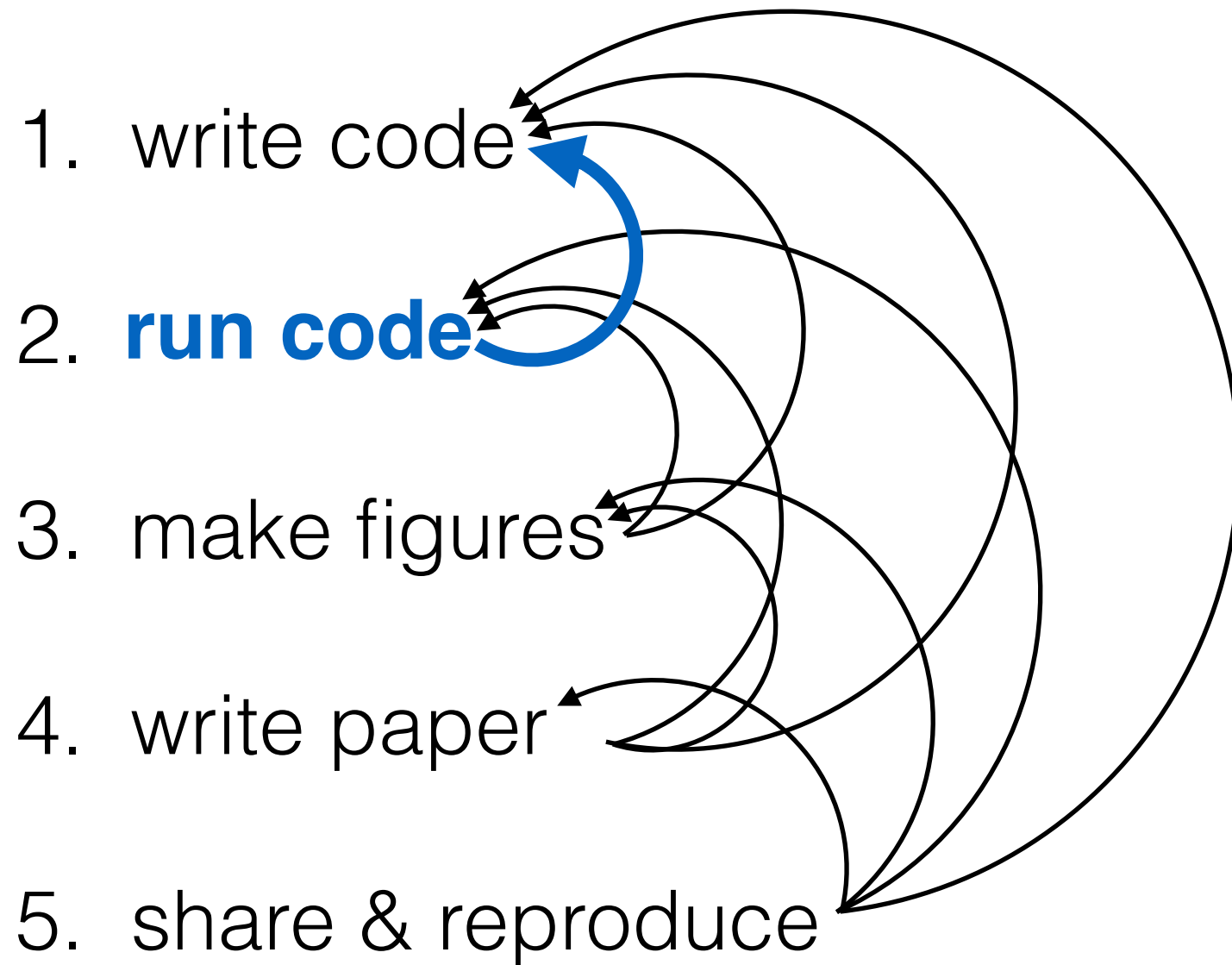
helps run code

- tab completion
- introspection
- %magics

```
minrk[02:13]~/Documents/Jupyter/pres/AGU-2014 $ ipython
```



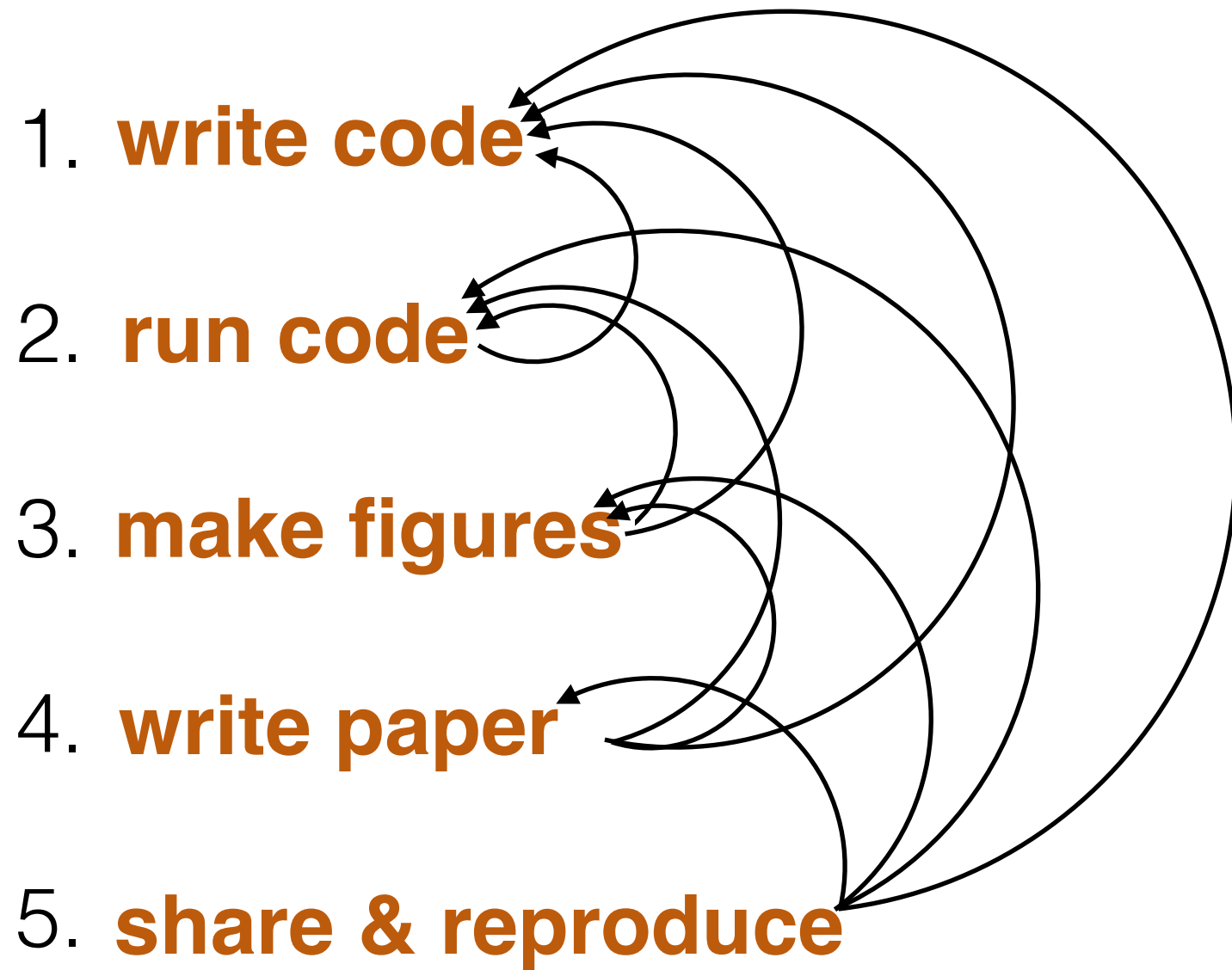
What about Jupyter?



IPython



What about Jupyter?





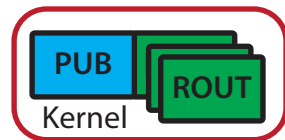
What is Jupyter?

Rich REPL Protocol



What is Jupyter?

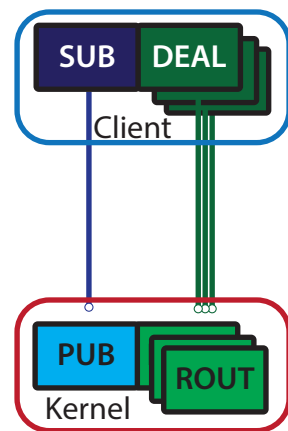
Rich REPL Protocol





What is Jupyter?

Rich REPL Protocol

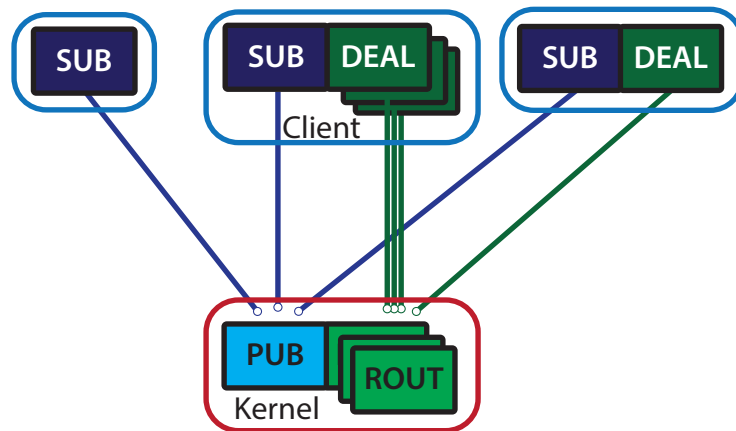


ØMQ + JSON



What is Jupyter?

Rich REPL Protocol



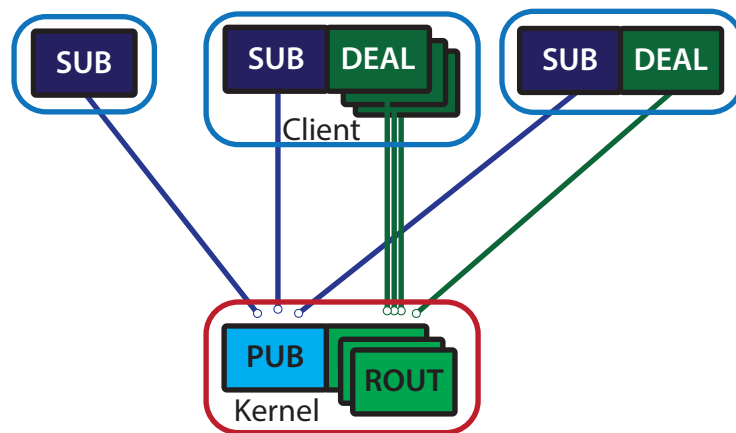
ØMQ + JSON



What is Jupyter?

Rich REPL Protocol

Document Format



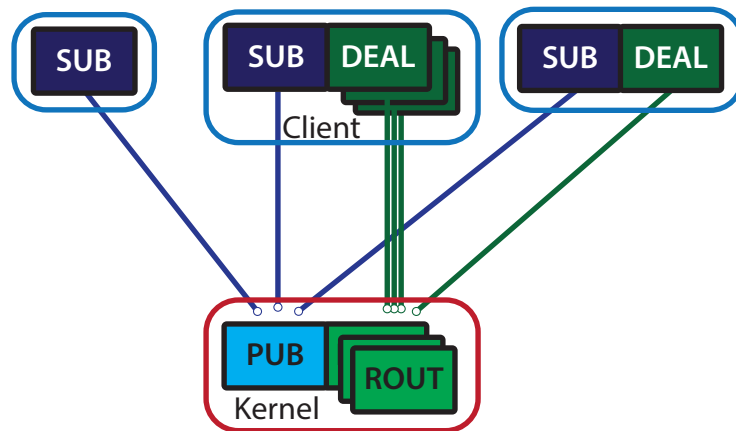
ØMQ + JSON



What is Jupyter?

Rich REPL Protocol

Document Format



ØMQ + JSON

We have already computed $P(X|A)$ above. On the other hand, $P(X| \sim A)$ is subjective: our code can pass tests but still have a bug in it, though the probability there is a bug present is reduced. Note this is dependent on the number of tests performed, the degree of complication in the tests, etc. Let's be conservative and assign $P(X| \sim A) = 0.5$. Then

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1 - p)}$$

$$= \frac{2p}{1 + p}$$

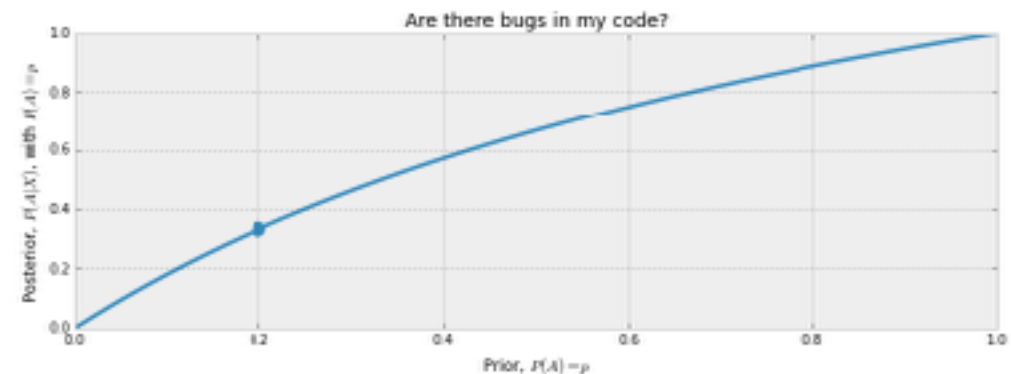
This is the posterior probability. What does it look like as a function of our prior, $p \in [0, 1]$?

```

figsize(12.5, 4)
p = np.linspace(0, 1, 50)
plt.plot(p, 2 * p / (1 + p), color="#348ABD", lw=3)
# plt.fill_between(p, 2*p/(1+p), alpha=.5, facecolor=["#A60628"])
plt.scatter(0.2, 2 * (0.2) / 1.2, s=140, c="#348ABD")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel("Prior, P(A) = p")
plt.ylabel("Posterior, P(A|X), with P(A) = p")
plt.title("Are there bugs in my code?")

```

<matplotlib.text.Text at 0x1051de650>





Jupyter Protocol

REP*L over JSON + ØMQ



Jupyter Protocol

REP*L over JSON + ØMQ

- Read

```
msg_type = 'execute_request'
content = {
    'code' : """
        import pandas as pd
        df = pd.read_csv('mydata.csv')
        """,
    ...
}
```



Jupyter Protocol

REP*L over JSON + ØMQ

- Read
- Eval

```
msg_type = 'execute_reply'  
content = {  
    'execution_count': 3,  
    'status': 'ok',  
    ...  
}
```



Jupyter Protocol

REP*L over JSON + ØMQ

- Read
- Eval
- Print*

```
msg_type = 'display_data'  
content = {  
    'data': {  
        "text/plain": "<MyDataFrame at 0x...>",  
        "text/html": "<table>...</table>",  
        ...  
    },  
    'metadata': {},  
    ...  
}
```



Jupyter Protocol

REP*L over JSON + ØMQ

- Read
- Eval
- Print*
- Loop

```
msg_type = 'display_data'
content = {
    'data': {
        "text/plain": "<MyDataFrame at 0x...>",
        "text/html": "<table>...</table>",
        ...
    },
    'metadata': {},
    ...
}
```




Jupyter Protocol

supercharge the P in REP*L

any mime-type output



Jupyter Protocol

supercharge the P in REP*L

any mime-type output

- text

```
In [5]: print(df.head())
```

	cake	lies	pie
2012-12-19	363.885981	367.826809	362.807807
2012-12-20	361.055153	368.463441	365.065045
2012-12-21	362.064454	367.768454	364.087118
2012-12-22	361.110406	368.457023	363.762849
2012-12-23	361.890903	369.800517	362.596256

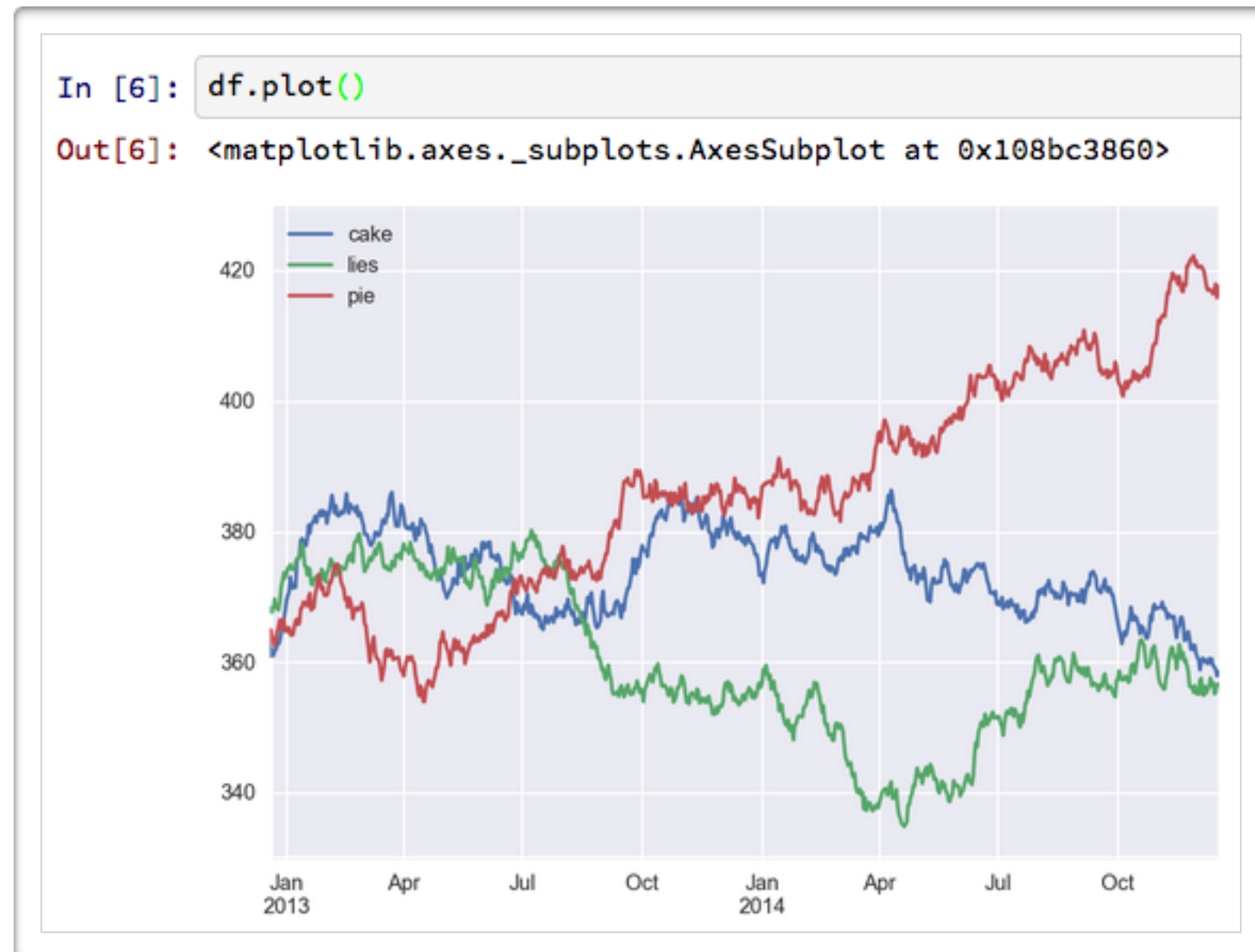


Jupyter Protocol

supercharge the P in REP*L

any mime-type output

- text
- svg, png, jpeg





Jupyter Protocol

supercharge the P in REP*L

any mime-type output

- text
- svg, png, jpeg
- latex, pdf

```
In [14]: Math(r'''f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x}, d\xi
           ''')
```

```
Out[14]: 
$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x}, d\xi$$

```



Jupyter Protocol

supercharge the P in REP*L

any mime-type output

- text
- svg, png, jpeg
- latex, pdf
- html, javascript

```
In [16]: df.tail()
```

```
Out[16]:
```

	cake	lies	pie
2014-12-14	400.537295	387.213920	371.035670
2014-12-15	402.107164	386.883925	370.902248
2014-12-16	402.548479	386.407872	369.037149
2014-12-17	403.010896	387.532590	369.017640
2014-12-18	403.191969	388.824959	369.630229



Jupyter Protocol

supercharge the P in REP*L

any mime-type output

- text
- svg, png, jpeg
- latex, pdf
- html, javascript
- interactive widgets

```
In [ ]: @interact
def factor_xn(n=5):
    display(Eq(x**n-1, factor(x**n-1)))
```



Jupyter Protocol is language agnostic





Jupyter Protocol is language agnostic



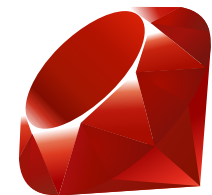
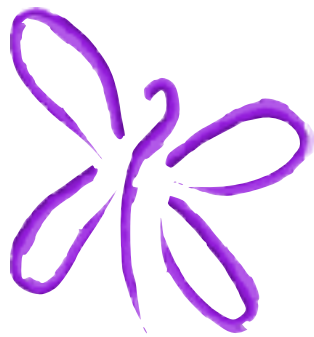


Jupyter Protocol is language agnostic





Jupyter Protocol is language agnostic





Jupyter Notebooks

- notebook = sequence of cells

The screenshot shows a Jupyter Notebook interface. The title bar reads "jupyter Sampling_Theorem (autosaved)" and "IPython (Python 3)". The menu bar includes "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". The main content area contains the following text:

Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Jupyter Notebooks

- notebook = sequence of cells
- text cell = markdown + latex)

The screenshot shows a Jupyter Notebook window titled "jupyter Sampling_Theorem (autosaved)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". The main content area contains a text cell with the following text:

Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Jupyter Notebooks

- notebook = sequence of cells
- text cell = markdown + latex)
- code cell = REP (input + output)

The screenshot shows a Jupyter Notebook window titled "jupyter Sampling_Theorem (autosaved)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". The main content area displays a text cell with the following text:

Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Jupyter Notebooks

- notebook = sequence of cells
- text cell = markdown + latex)
- code cell = REP (input + output)
- metadata everywhere

The screenshot shows a Jupyter Notebook window titled "jupyter Sampling_Theorem (autosaved)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". The main content area displays a text cell with the following text:

Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Jupyter Notebooks

- Plain Text (JSON)

The screenshot shows a Jupyter Notebook interface. The title bar reads "jupyter Sampling_Theorem (autosaved)" and includes a Python version dropdown set to "IPython (Python 3)" and a "Logout" button. A menu bar contains "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". The main content area displays the following text:

Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Jupyter Notebooks

- Plain Text (JSON)
- Publicly documented schema

The screenshot shows a Jupyter Notebook interface with the title "Sampling_Theorem (autosaved)". The notebook content includes:

Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Jupyter Notebooks

- Plain Text (JSON)
- Publicly documented schema
- Machine readable, easy to understand

The screenshot shows a Jupyter Notebook interface with the title "Sampling_Theorem (autosaved)". The notebook content includes:

Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Jupyter Notebooks

- Plain Text (JSON)
- Publicly documented schema
- Machine readable, easy to understand
- Transformable (nbconvert)

The screenshot shows a Jupyter Notebook interface with the title "Sampling_Theorem (autosaved)". The notebook content includes a section titled "Investigating the Sampling Theorem". The text in the notebook reads:

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Jupyter Notebooks

- interactive environment

The screenshot shows a Jupyter Notebook interface with the following elements:

- Header: "jupyter Sampling_Theorem (autosaved)" with a Python 3 dropdown and a "Logout" button.
- Menu: "File Edit View Insert Cell Kernel Help".
- Content: A document titled "Investigating the Sampling Theorem".

Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Jupyter Notebooks

- interactive environment
- input format

The screenshot shows a Jupyter Notebook interface with the title "Sampling_Theorem (autosaved)". The notebook content includes:

Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Jupyter Notebooks

- interactive environment
- input format
- output format

The screenshot shows a Jupyter Notebook interface with the following elements:

- Header: "jupyter Sampling_Theorem (autosaved)" with a Python 3 dropdown and a "Logout" button.
- Menu: "File Edit View Insert Cell Kernel Help".
- Content: A document titled "Investigating the Sampling Theorem".

Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

"If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart."

Since a function $x(t)$ is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic $x(t)$)

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points a_n by integrating over the **entire** function $x(t)$, not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point a_n , whereas with sampling we are just taking individual points in isolation.



Lifecycle of a Computational Idea



Lifecycle of a Computational Idea

1. Explore an idea interactively in a Notebook



Lifecycle of a Computational Idea

1. Explore an idea interactively in a Notebook
2. *Build/add to a library based on what you learn*



Lifecycle of a Computational Idea

1. Explore an idea interactively in a Notebook
2. *Build/add to a library based on what you learn*
3. Record and collaborate on analyses in Notebooks



Lifecycle of a Computational Idea

1. Explore an idea interactively in a Notebook
2. *Build/add to a library based on what you learn*
3. Record and collaborate on analyses in Notebooks
4. Document, demonstrate, and share in Notebooks



Lifecycle of a Computational Idea

1. Explore an idea interactively in a Notebook
2. *Build/add to a library based on what you learn*
3. Record and collaborate on analyses in Notebooks
4. Document, demonstrate, and share in Notebooks
5. Computational companions, reproducible papers

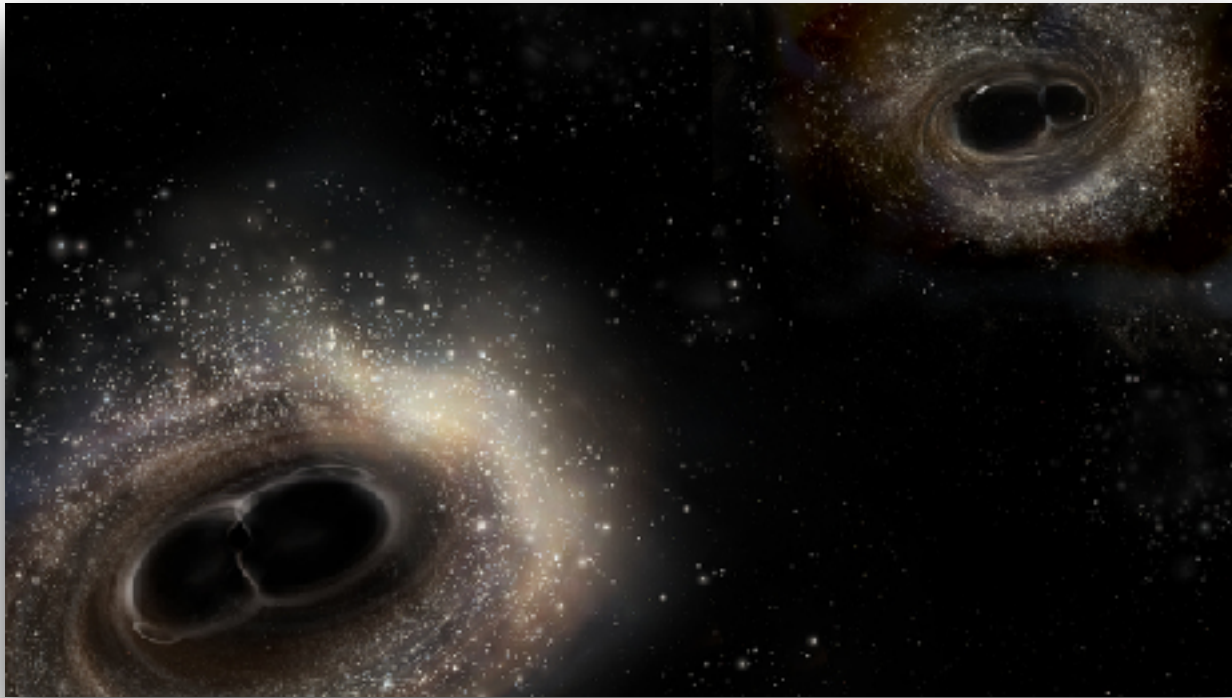


Applications of Jupyter

- **jupyterlab** - new UI, javascript APIs for notebooks
- **nbconvert** - convert notebooks to other formats (rst, html, latex/pdf, markdown, script, reveal.js slides)
- **nbviewer** - nbconvert to html on the web
- **jupyterhub** - multi-user notebook server for classes, groups
- **nbgrader** - automated grading of notebooks
- **tmpnb** - containerized (docker) transient deployments of notebooks
- **thebe** - transient kernels on the web, without notebooks
- **binder** - turn a GitHub repo of notebooks into a free executable environment
- **nbdime** - diffing and merging notebooks
- **nteract** - javascript implementation of Jupyter for electron- and react-based applications
- **dashboards** - 2D layout, hiding input for presentations of notebook results
- **dexy** - reproducible document-based workflows

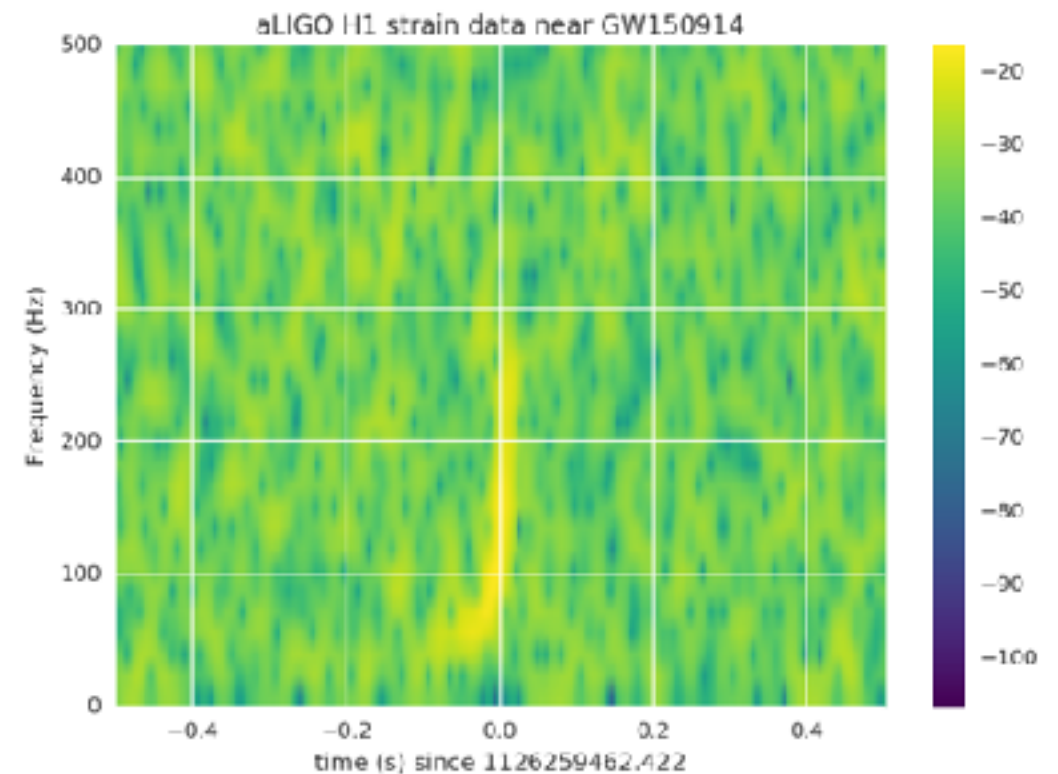


Jupyter in Science



```
# Plot the L1 whitened spectrogram around the signal
plt.figure()
spec_H1, freqs, bins, im = plt.specgram(strain_L1_whiten[indxt], NFFT=1024,
                                       noverlap=NOVL, cmap=spec_cm)

plt.xlabel('time (s) since '+str(tevent))
plt.ylabel('Frequency (Hz)')
plt.colorbar()
plt.axis([-0.5, 0.5, 0, 500]);
```



LIGO

- Direct Observation of Gravitational Waves
- Published data and analysis as Jupyter Notebooks

<https://lsc.ligo.org/events/GW150914/>

<https://github.com/minrk/ligo-binder>

image: <https://www.ligo.caltech.edu/news/ligo20160615>



Jupyter in the Classroom

Lorena Barba

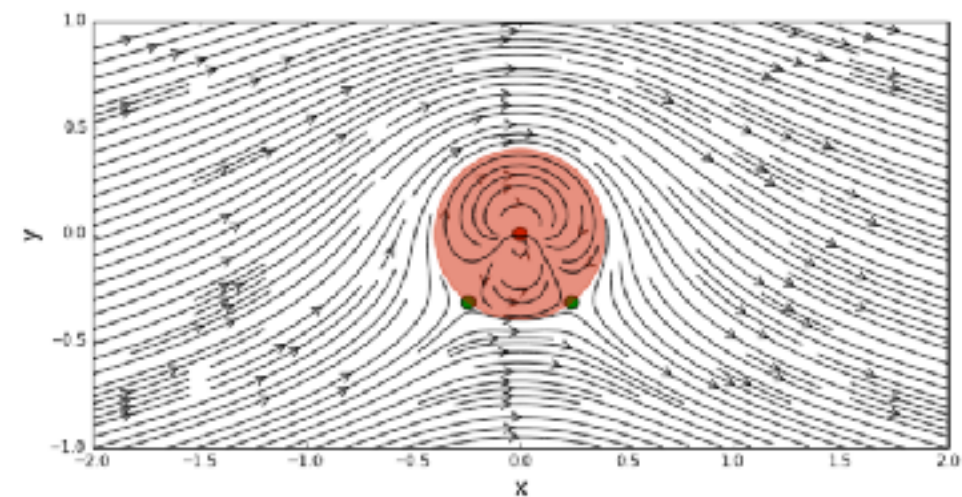
AeroPython

Aerodynamics, George Washington U

Flipped Classroom

Lorena A. Barba group

Announcing AeroPython!



Posted on 04.20.2014

<http://lorenabarba.com/blog/announcing-aeropython>



Jupyter in the Classroom

The screenshot shows a Jupyter notebook page with the following content:

- Header:** Jupyter logo and the text "Jupyter at Bryn Mawr College".
- Public notebooks:** A list of links including [/jupyter/hub/db/blank/public](#), [/CS110 Intro to Computing](#), and [/2015-Fall](#).
- Code Cell:** A code cell labeled "In [3]:" containing the following Python code:

```
fill(255, 0, 128);
beginShape();
vertex(10, 10);
vertex(70, 10);
vertex(30, 80);
vertex(40, 80);
vertex(5, 50);
vertex(10, 10);
endShape();
```
- Output:** A sketch labeled "Sketch #3:" showing a pink filled polygon on a gray background. The polygon has vertices at approximately (10, 10), (70, 10), (30, 80), (40, 80), and (5, 50).
- Status:** Below the sketch, it says "Sketch #3 state: Done."

Doug Blank

CS, Brynmawr

Calysto (Multiple languages, not Python)

JupyterHub

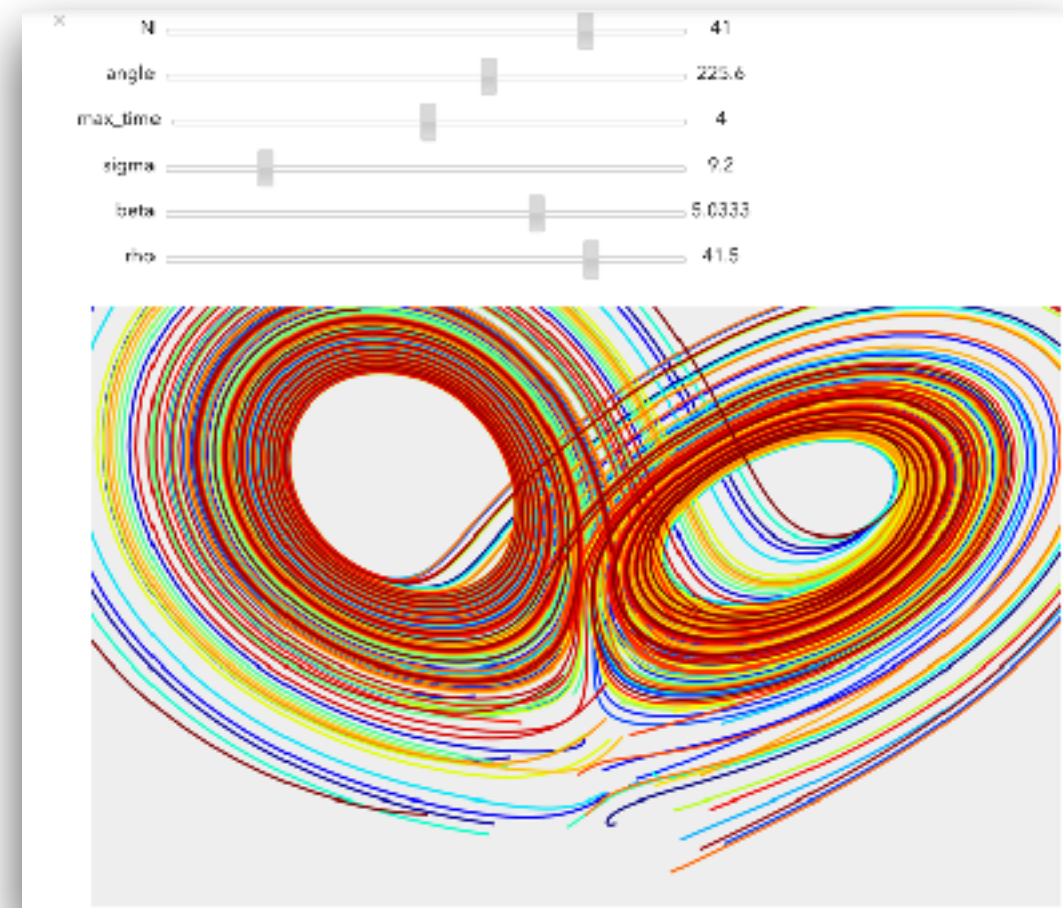


Jupyter in the Classroom

Brian Granger

Computational Physics, Cal Poly

JupyterHub



<https://github.com/ellisonbg/phys202-2015>



Jupyter in the Classroom

Eric Matthes
High School Programming
Alaska, USA

How IPython Notebook and Github have changed the way I teach Python

Posted on September 22, 2013

I teach in a small high school in southeast Alaska, and each year I teach an Introduction to Programming class. I recently learned how to use [IPython Notebook](#), and it has completely [changed the way I teach my classes](#). There is much to improve about CS education at the K-12 level in the United States, and sharing our stories and our resources will go a long way towards improving what we offer to students.





Jupyter in the Classroom

Jessica Hamrick
Computational Models of
Cognition
Psychology, UC Berkeley
220 students
JupyterHub
NBGrader

```
← Prev Assignments / ps6 / 3 - k-Means Clustering / Submission #1 Next →
```

In [5]: Student's answer

```
def distance(a, b):  
    """  
    Returns the Euclidean distance between two points,  
    a and b, in R^2.  
  
    Parameters  
    -----  
    a, b : numpy arrays of shape (2,)  
           The (x,y) coordinates for two points, a and b,  
           in R^2  
  
    Returns  
    -----  
    distance : float  
           The Euclidean distance between a and b  
    """  
    # YOUR CODE HERE  
    return np.sqrt(np.sum((a + b) ** 2))
```

You need a minus sign, not a plus sign.

In [6]: # add your own test cases here!

In [7]: test_distance Full credit No credit / 0.5

```
"""Check distance computes the correct values"""  
from numpy.testing import assert_allclose  
  
assert_allclose(distance(np.array([0.0, 0.0]), np.array([0.0, 1.0])), 1.0)  
assert_allclose(distance(np.array([3.0, 3.0]), np.array([4.3, 5.0])), 2.3853720883753127)  
assert_allclose(distance(np.array([130.0, -25.0]), np.array([0.4, 15.0])), 135.6324449614552)  
  
print("Success!")
```

```
-----  
AssertionError                                Traceback (most recent call last)  
<ipython-input-7-84d0934e10a9> in <module>()  
    3  
    4 assert_allclose(distance(np.array([0.0, 0.0]), np.array([0.0, 1.0])), 1.0)  
----> 5 assert_allclose(distance(np.array([3.0, 3.0]), np.array([4.3, 5.0])), 2.3853720883753127)  
    6 assert_allclose(distance(np.array([130.0, -25.0]), np.array([0.4, 15.0])), 135.6324449614552)  
    7
```

