

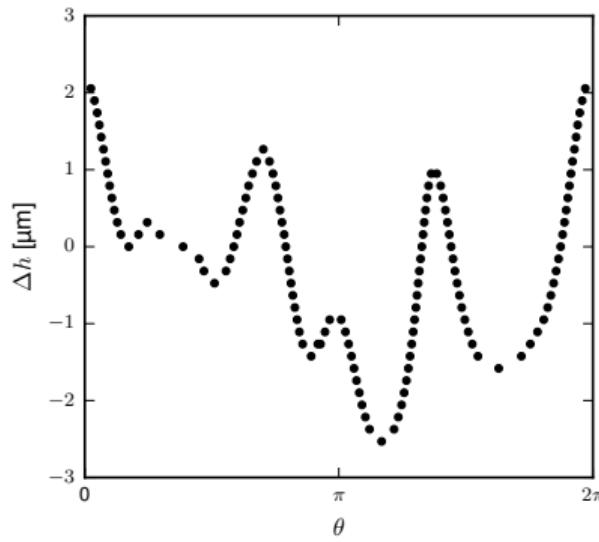
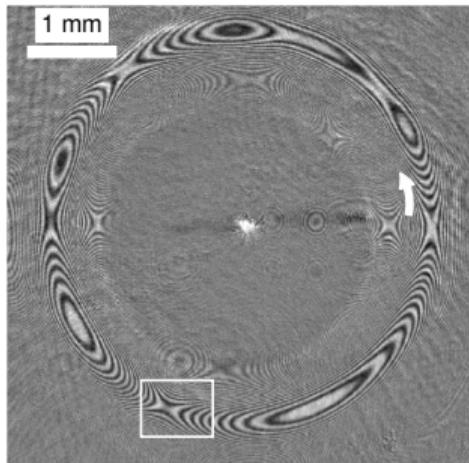
Matplotlib and Scientific Visualization

Thomas A Caswell

2021-03-17

Who am I?

- ▶ Trained as a physicist
 - ▶ Jamming + dynamics of Leidenfrost drops with Nagel and Gardel at UChicago



Who am I?

- ▶ Trained as a physicist
 - ▶ Jamming + dynamics of Leidenfrost drops with Nagel and Gardel at UChicago
- ▶ Currently in Data Science and System Integration program at NSLS-II
- ▶ Current Project Lead of Matplotlib



Acknowledgments

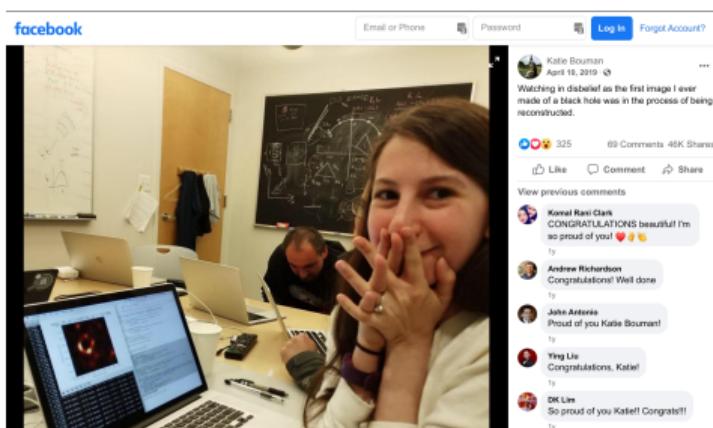
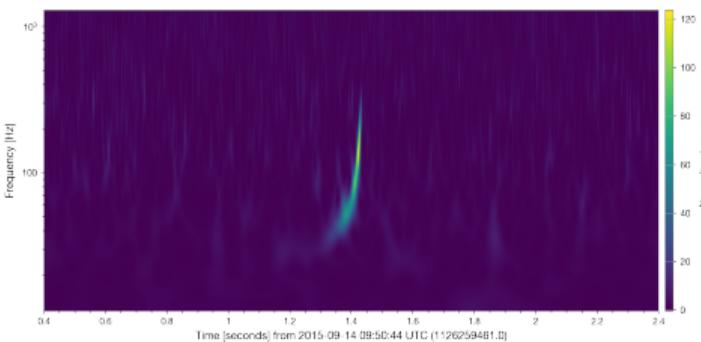
- ▶ John Hunter (1968-2012)
- ▶ Michael Droettboom
- ▶ The whole Matplotlib development team
 - ▶ Over 1,250+ have contributed code, many more in bug reports, feature requests, and user support
- ▶ Dora Caswell

This project has been made possible in part by grant number 2019-207333 from the Chan Zuckerberg Initiative DAF, an advised fund of Silicon Valley Community Foundation.

Matplotlib

... is a comprehensive library for creating static, animated, and interactive visualizations in Python.

- ▶ Widely used throughout science
 - ▶ LIGO, EHT, LSST, NSLS-II, APS, ALS, SPT, STScI, ATLAS, Mars rovers, ...
 - ▶ Single PI labs in physics, chemistry, oceanography, meteorology, biology, ...



<https://www.facebook.com/photo.php?fbid=10213321822457967>
<https://mybinder.org/v2/gh/losc-tutorial/quickview/master>
<https://github.com/losc-tutorial/quickview>

What is visualization for?

1. Exploratory data analysis

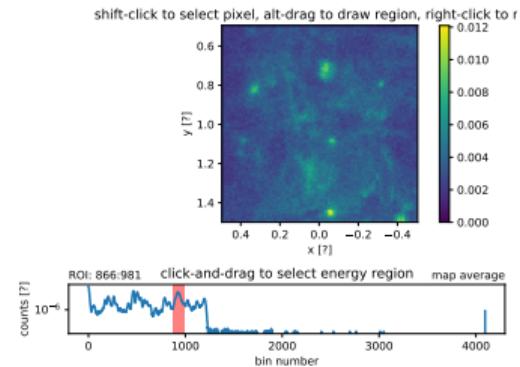
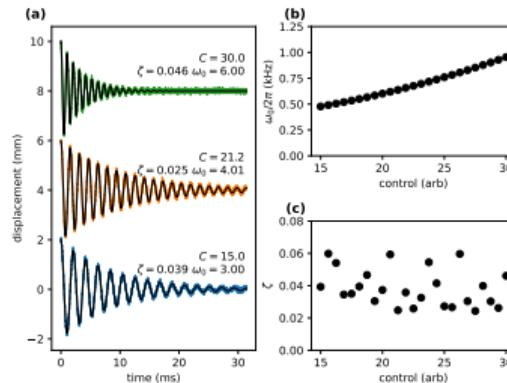
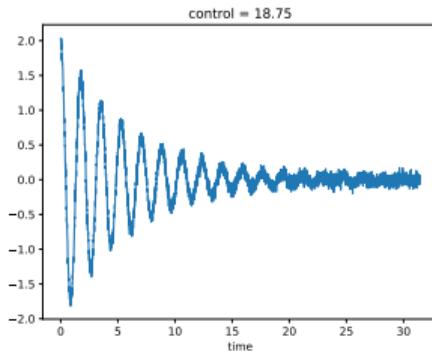
- ▶ Just get the data on the screen in a way *you* can understand as fast as you can

2. Paper figures

- ▶ Need to be *just right*

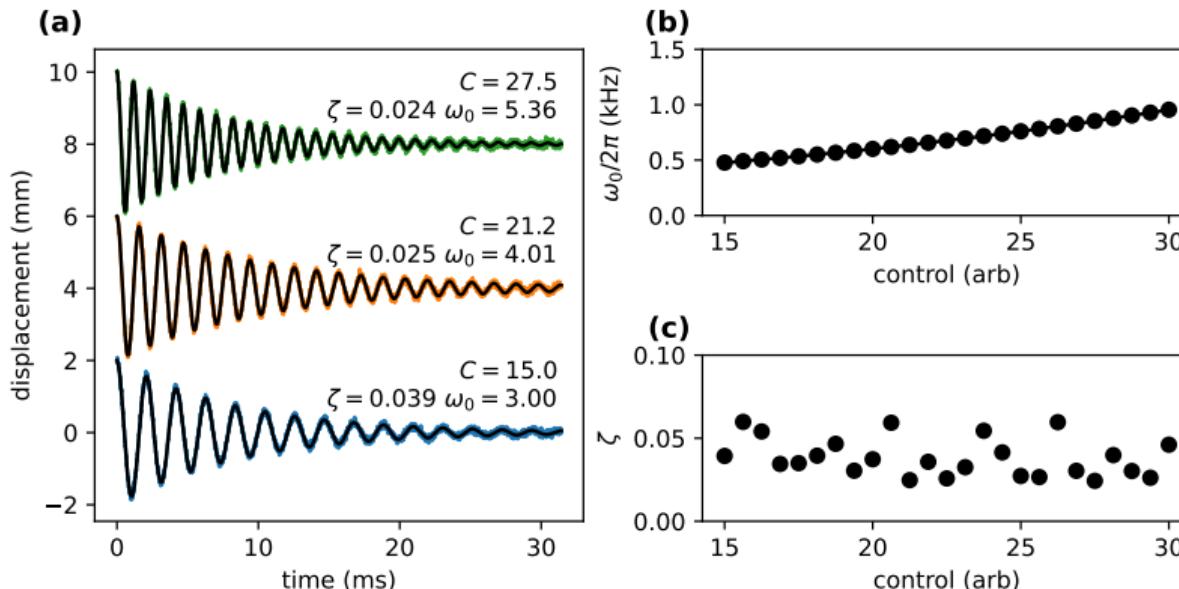
3. Part of a standard (interactive) workflow

- ▶ Repeatedly visualize and explore similar data



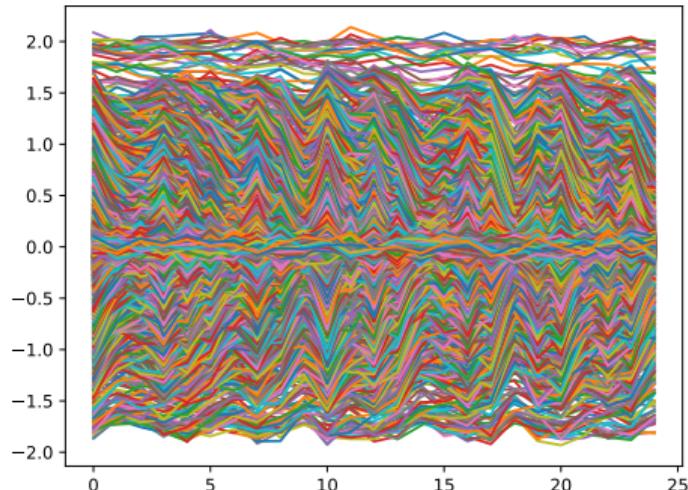
Case Study: Paper Figure

- ▶ You have 25 cantilevers, varied something (called 'control') in fabrication
- ▶ Displace away from equilibrium position and released to watch vibrations ring down
- ▶ Goals:
 - ▶ go from exploratory visualization to paper ready figures
 - ▶ write reusable helper functions



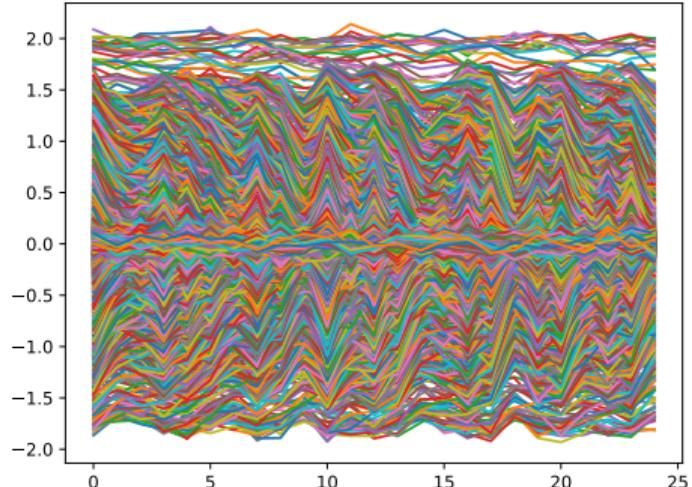
Step 0.0 ("just plot it")

```
1 from gen_data import get_data  
2 import matplotlib.pyplot as plt  
3  
4 # d is an xarray.DataArray  
5 d = get_data()  
6 plt.plot(d)
```



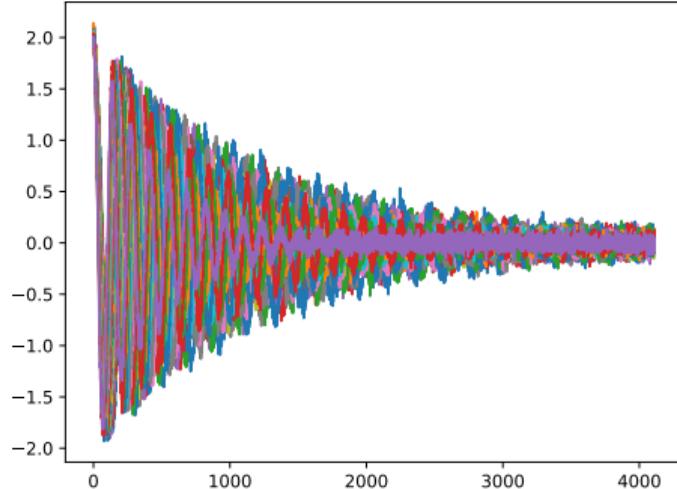
Step 0.0 ("just plot it")

```
1 from gen_data import get_data  
2 import matplotlib.pyplot as plt  
3  
4 # d is an xarray.DataArray  
5 d = get_data()  
6 plt.plot(d)  
  
▶ #accidental art
```



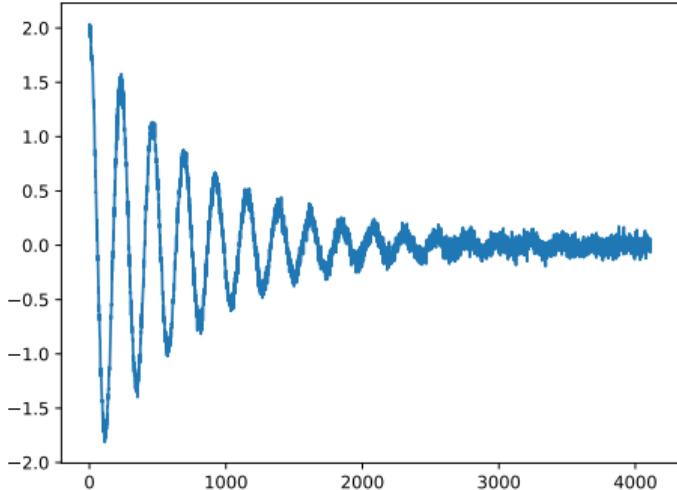
Step 0.1 ("just plot it... the right direction")

```
1 from gen_data import get_data  
2 import matplotlib.pyplot as plt  
3  
4 # d is an xarray.DataArray  
5 d = get_data()  
6 plt.plot(d.T)
```



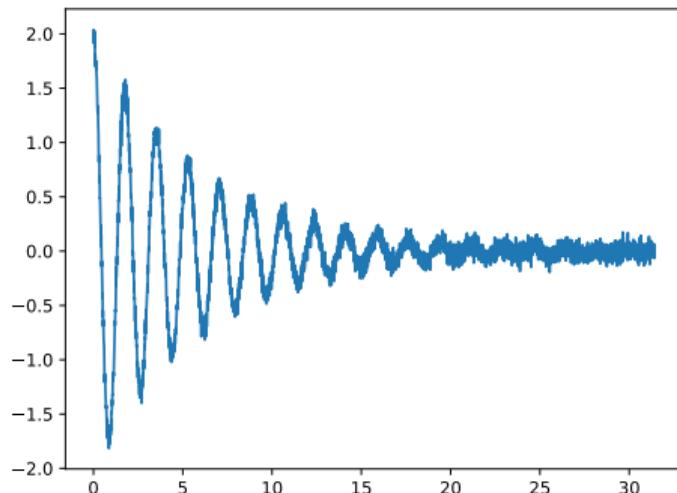
Step 1.0 (plot just one)

```
1 from gen_data import get_data  
2 import matplotlib.pyplot as plt  
3  
4 # d is an xarray.DataArray  
5 d = get_data()  
6 # pick out an arbitrary row  
7 m = d[6]  
8 plt.plot(m)
```



Step 1.1 (use meaningful x-axis, move to explicit API)

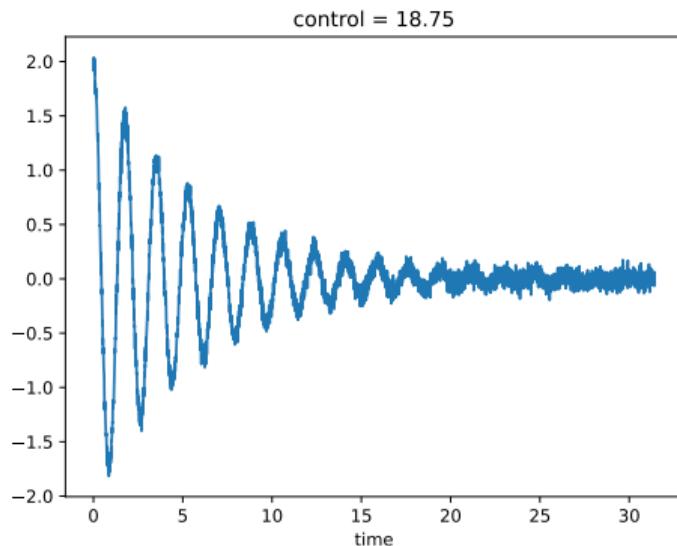
```
1 from gen_data import get_data
2 import matplotlib.pyplot as plt
3
4 # d is an xarray.DataArray
5 d = get_data()
6 # pick out an arbitrary row
7 m = d[6]
8
9 fig, ax = plt.subplots()
10 ax.plot(m.time, m)
```



- ▶ Use the explicit API to determine which Axes plots go on rather than the implicit pyplot API.

Step 1.1.1 (use xarray's plotting for a quick view)

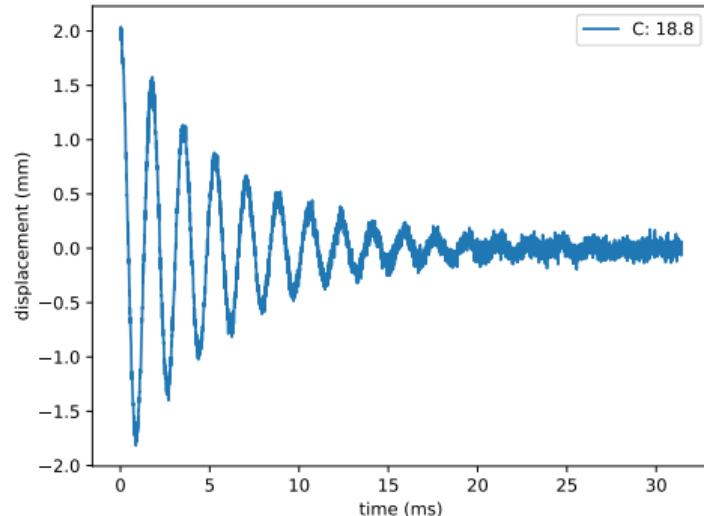
```
1 from gen_data import get_data
2 import matplotlib.pyplot as plt
3
4 # d is an xarray.DataArray
5 d = get_data()
6 # pick out an arbitrary row
7 m = d[6]
8
9 fig, ax = plt.subplots()
10 m.plot(ax=ax)
```



- ▶ The xarray.DataArray knows enough about itself to plot against time, label the x-axis, and set the title.

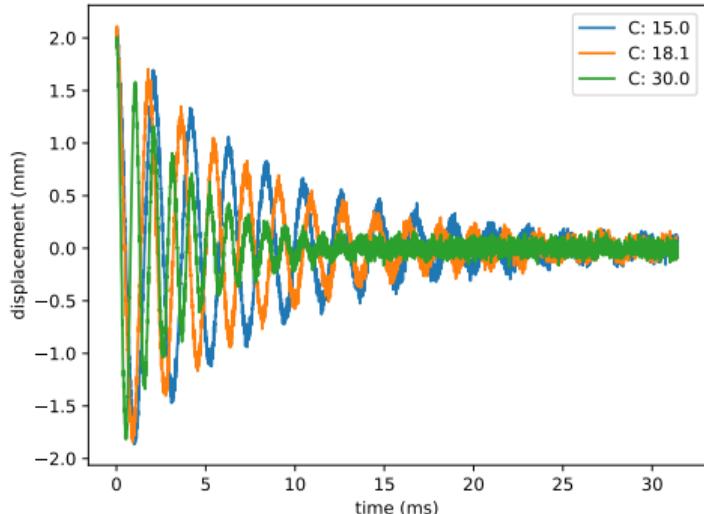
Step 1.2 (add legend and axis labels)

```
1 d = get_data()  
2 m = d[6]  
3 ctrl = float(m.control)  
4 fig, ax = plt.subplots()  
5  
6 ax.plot(  
7     m.time,  
8     m,  
9     label=f"C: {ctrl:.1f}")  
10 )  
11  
12 ax.legend()  
13 ax.set_xlabel("time (ms)")  
14 ax.set_ylabel("displacement (mm)")
```



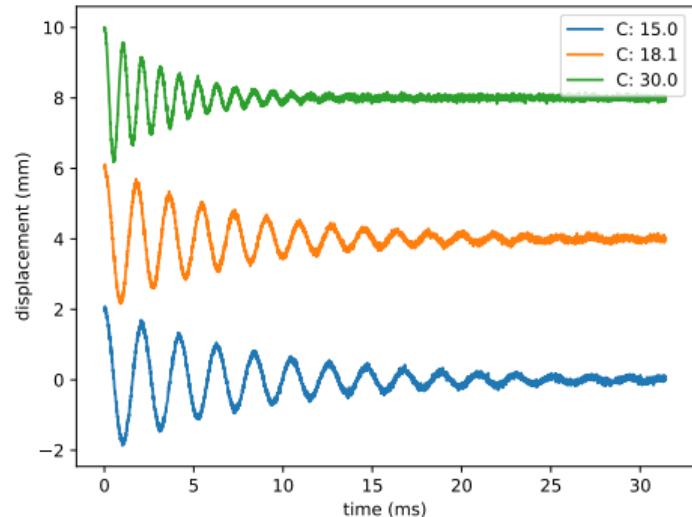
Step 1.3 (make a function)

```
1 def plot_one(ax, m):  
2     ...  
3     ln, = ax.plot(  
4         m.time, m, label=label  
5     )  
6     ...  
7  
8 fig, ax = plt.subplots()  
9 plot_one(ax, d[0])  
10 plot_one(ax, d[5])  
11 plot_one(ax, d[-1])  
12 ...
```



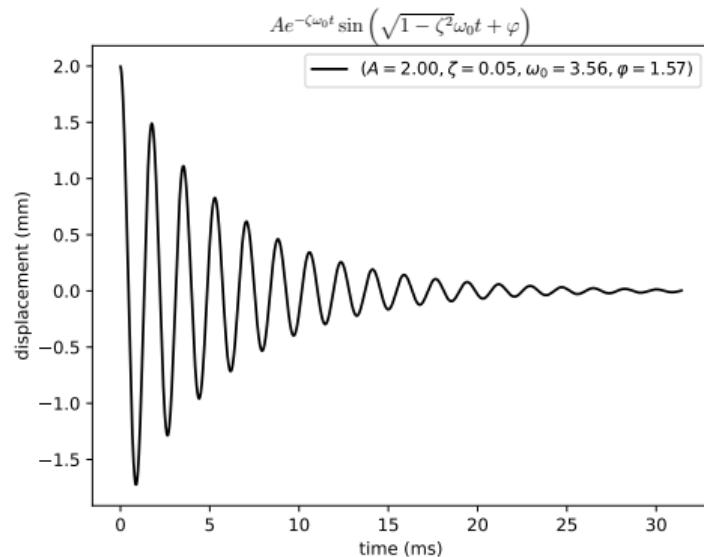
Step 1.4 (add vertical offset to your function)

```
1 def plot_one(ax, m, offset=0):  
2     ...  
3     ln, = ax.plot(  
4         m.time,  
5         m + offset,  
6         label=label  
7     )  
8     ...  
9     ...  
10    plot_one(ax, d[0], offset=0)  
11    plot_one(ax, d[5], offset=4)  
12    plot_one(ax, d[-1], offset=8)  
13    ...
```



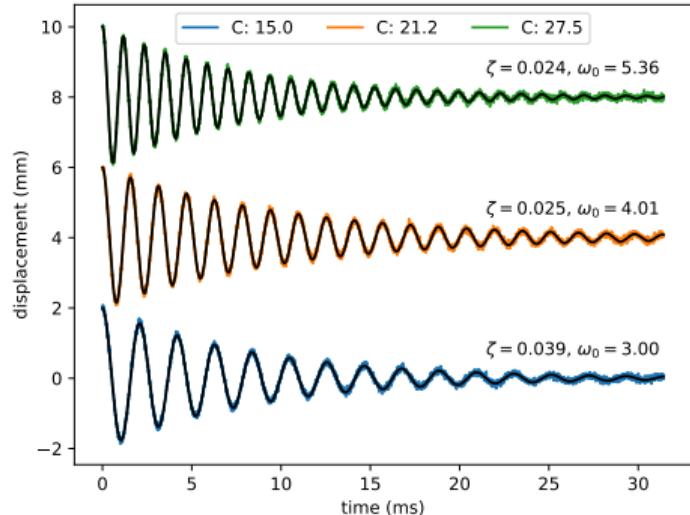
Step 2.0 (fit the data)

```
1  from gen_data import fit
2  ...
3  fit_vals = fit(m)
4  ax.plot(
5      m.time,
6      fit_vals.sample(m.time),
7      label=fit_vals._repr_latex_(),
8      color='k'
9  )
10 plt.gca().set_title(
11     r"$A e^{-\zeta\omega_0 t} \sin(\sqrt{1-\zeta^2}\omega_0 t + \varphi)$"
12     usetex=True
13 )
14 ...
```



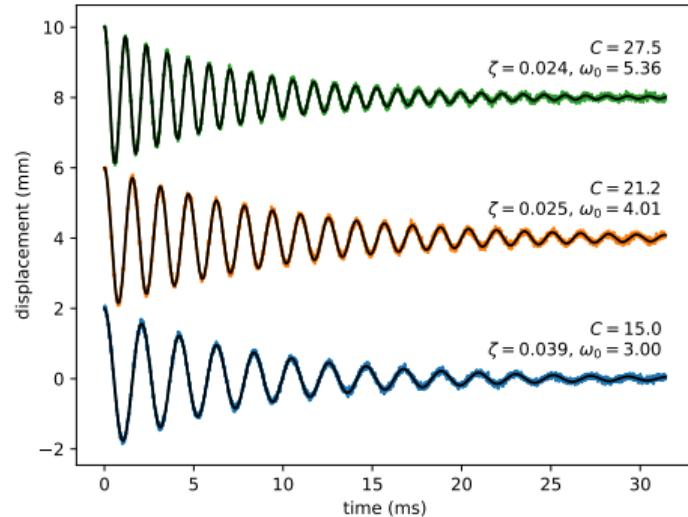
Step 2.1 (plot fit on data)

```
1 def plot_one(ax, m, fv, offset=0):
2     fit, = ax.plot(
3         t, fv.sample(t) + offset
4     )
5     ann = ax.annotate(...)
6     ...
7 ...
8 plot_one(
9     ax, d[10], fit(d[10]), offset=4
10)
11 ax.legend(n
12     ncol=3, loc="upper center"
13)
14 ...
```



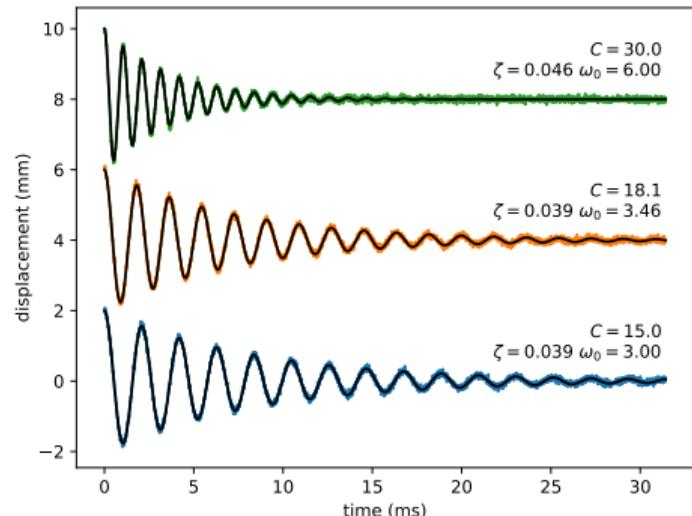
Step 2.2 (eliminate need for legend box)

```
1 def plot_one(ax, m, fv, offset=0):
2     ...
3     ann = ax.annotate(...)
4     ...
5     ...
6     ...
7     plot_one(
8         ax, d[10], fit(d[10]), offset=4
9     )
10    ...
```



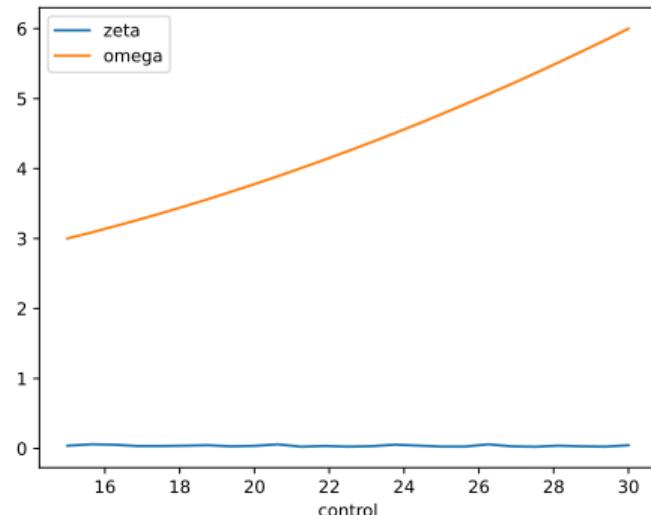
Step 2.3 (wrap plotting multiple curves with offsets and fits in a function)

```
1 def plot_several(ax, d, fits):
2     for j, (m, fv) in enumerate(
3         zip(d, fits)
4     ):
5         plot_one(ax, m, fv, 4*j)
6     ax.set_xlabel("time (ms)")
7     ax.set_ylabel("displacement (mm)")
8
9 fig, ax = plt.subplots()
10 indx = [0, 5, 24]
11 fits = [fit(d[i]) for i in indx]
12 plot_several(ax, d[indx], fits)
```



Step 3.0 (first look at ω_0 and ζ)

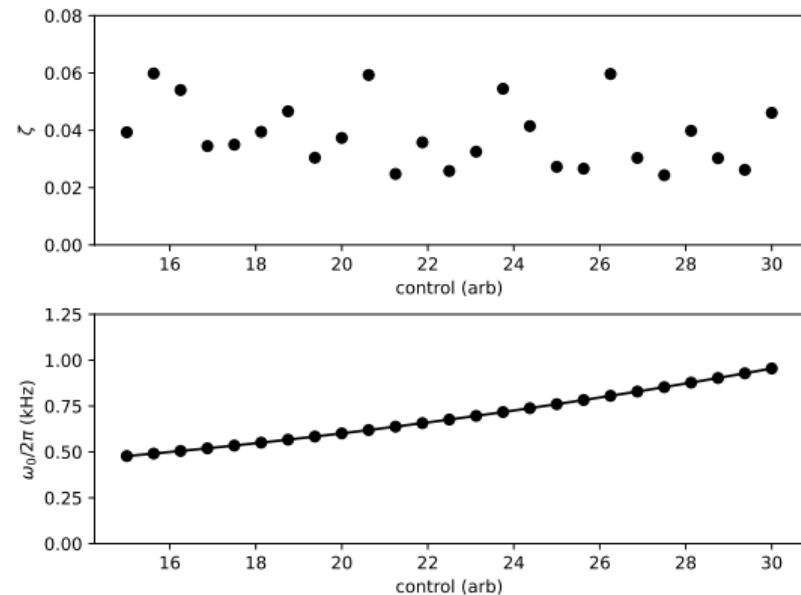
```
1 import pandas as pd  
2  
3 fits_df = pd.DataFrame(  
4     [fit(m) for m in d],  
5     index=d.coords["control"]  
6 )  
7  
8 fig, ax = plt.subplots()  
9 fits_df.plot(  
10     y=["zeta", "omega"], ax=ax  
11 )
```



- ▶ uses the column names for legend
- ▶ uses index for x-axis/label

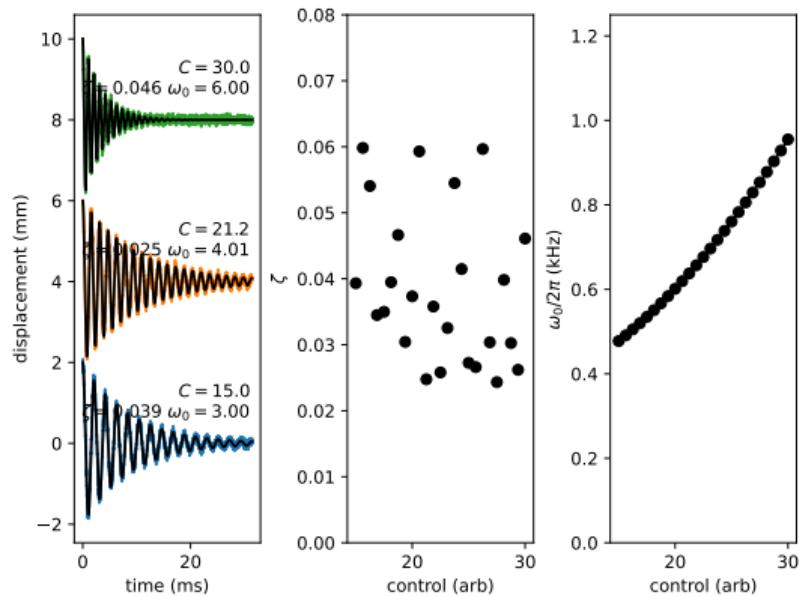
Step 3.1 (helper functions for ω_0 and ζ plots)

```
1 def plot_zeta(ax, fits_df):
2     ax.set_xlabel(...)
3     ax.set_ylabel(...)
4     return ax.plot(...)
5
6 def plot_omega(ax, fits_df):
7     ax.set_xlabel(...)
8     ax.set_ylabel(...)
9     return ax.plot(...)
10
11 fig, (ax1, ax2) = plt.subplots(2, 1)
12 plot_zeta(ax1, fits_df)
13 plot_omega(ax2, fits_df)
```



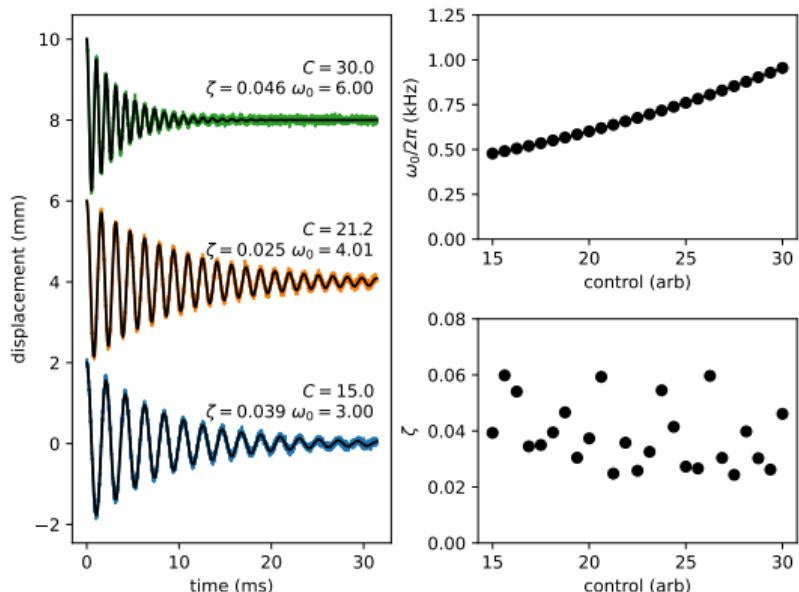
Step 4.0 (put it all together)

```
1  indx = [0, 10, 24]
2
3  fig, (ax1, ax2, ax3) = plt.subplots(
4      1, 3, constrained_layout=True
5 )
6  plot_several(
7      ax1,
8      d[indx],
9      [fits[i] for i in indx]
10 )
11 plot_zeta(ax2, fits_df)
12 plot_omega(ax3, fits_df)
```



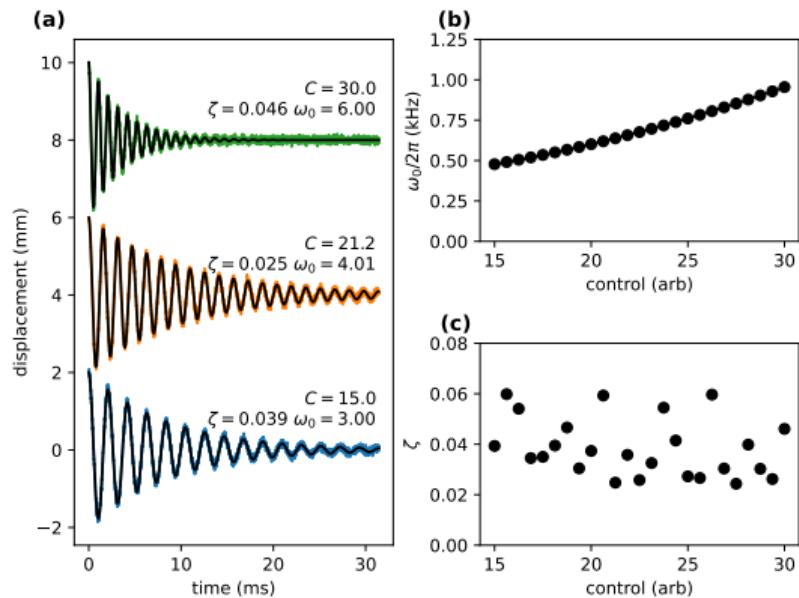
Step 4.1 (improve the layout)

```
1 fig, ad = plt.subplot_mosaic(
2     [["raw", "omega"],
3      ["raw", "zeta"]], 
4     constrained_layout=True
5 )
6 idx = [0, 10, 24]
7 plot_several(
8     ad["raw"],
9     d[idx],
10    [fits[i] for i in idx]
11 )
12 plot_zeta(ad["zeta"], fits_df)
13 plot_omega(ad["omega"], fits_df)
```



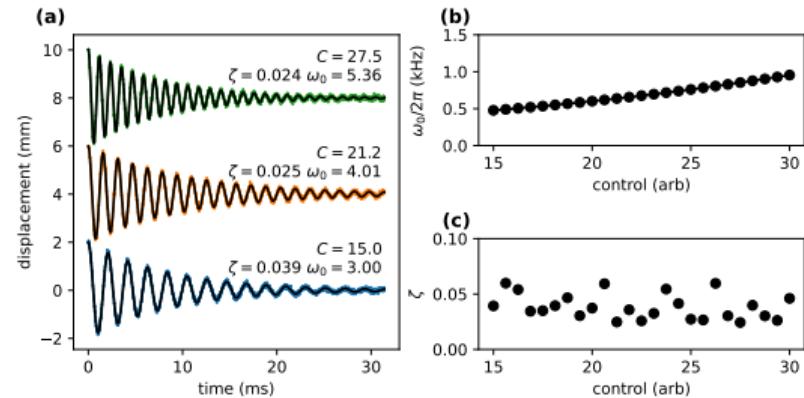
Step 4.2 (label the subplots)

```
1 def subplot_label(ax, text):
2     return ax.annotate(text, ...)
3
4 spm = {
5     "raw": "a",
6     "omega": "b",
7     "zeta": "c"
8 }
9
10 for k, v in spm.items():
11     subplot_label(ad[k], f"({v})")
```



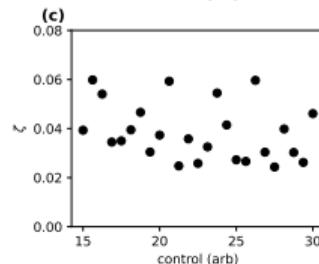
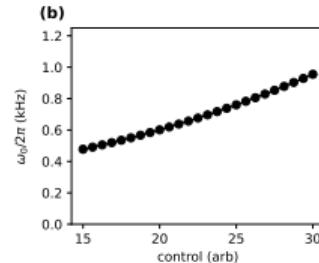
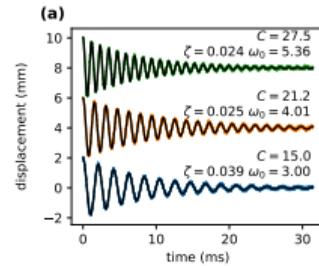
Step 4.3 (set the size to journal specifications)

```
1 def paper_figure_2(  
2     fig, layout, d, fits, plot_every  
3 ):  
4     ...  
5  
6     dcw = 17.8 / 2.54  
7     paper_figure_2(  
8         plt.figure(  
9             figsize=(dcw, dcw * 0.5)  
10            ),  
11            [["raw", "omega"], ["raw", "zeta"]],  
12            d,  
13            fits,  
14            plot_every=10,  
15        )
```



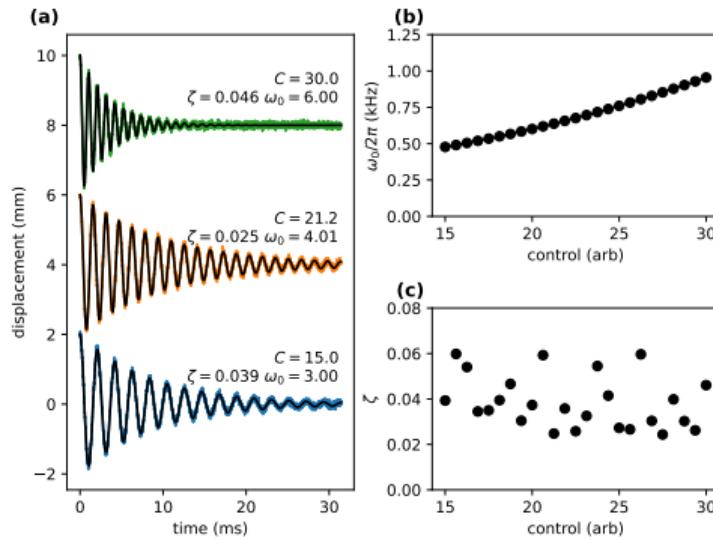
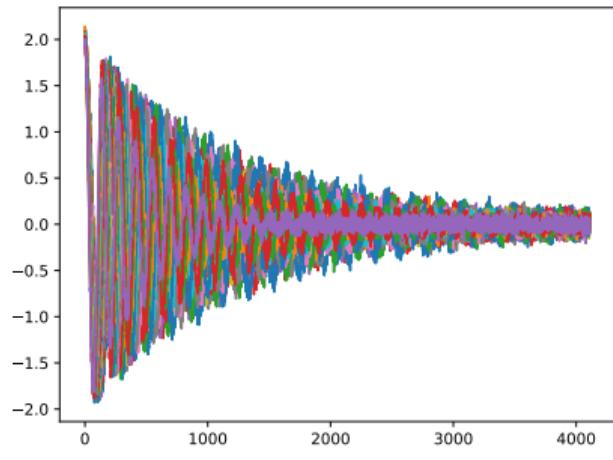
Step 4.4 (try a one column layout)

```
1 scw = 8.6 / 2.54
2 paper_figure_2(
3     plt.figure(
4         figsize=(scw, scw * 2.5)
5     ),
6     [["raw"], ["omega"], ["zeta"]],
7     d,
8     fits,
9     plot_every=10,
10 )
```



Case Study Summary

- ▶ Went from initial exploratory look at data to a paper-ready figure
- ▶ Iteratively built a mini-library for **this** experiment



Interactive Visualizations

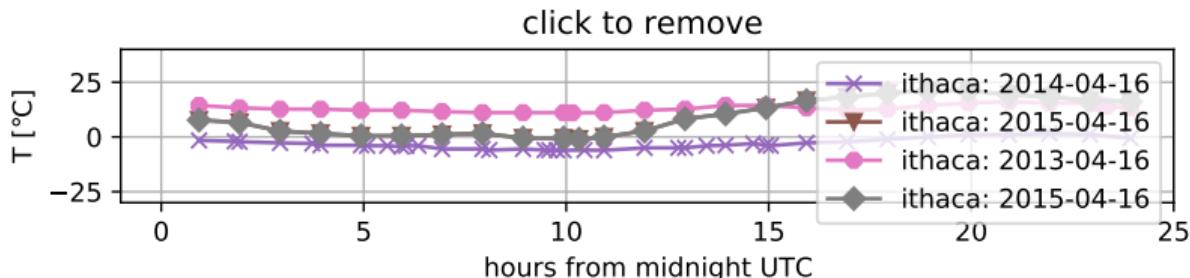
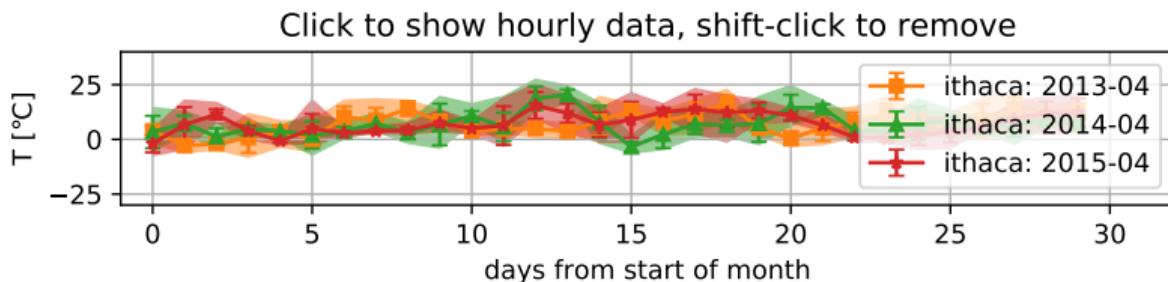
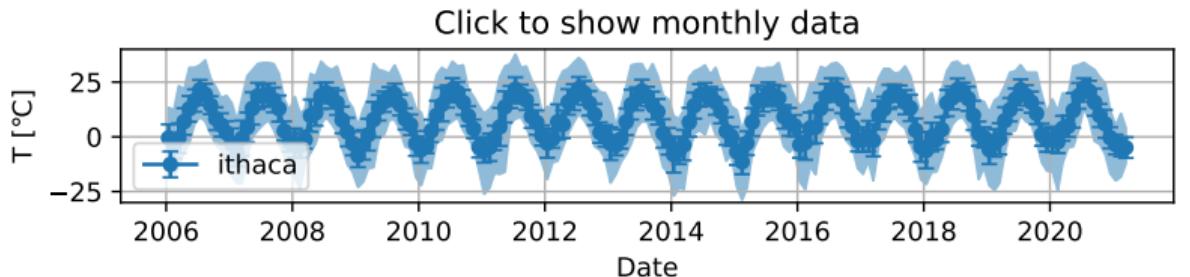
- ▶ Matplotlib has framework-agnostic UI tools to get mouse and keyboard events
- ▶ several third-party-packages with more complex interactions
 - ▶ `mplcursors` <https://mplcursors.readthedocs.io/en/stable/>
 - ▶ `MplDataCursor` <https://github.com/joferkington/mpldatacursor>
 - ▶ `mpl_interactions` <https://mpl-interactions.readthedocs.io/en/latest/>

LIVE DEMO TIME

Hello world

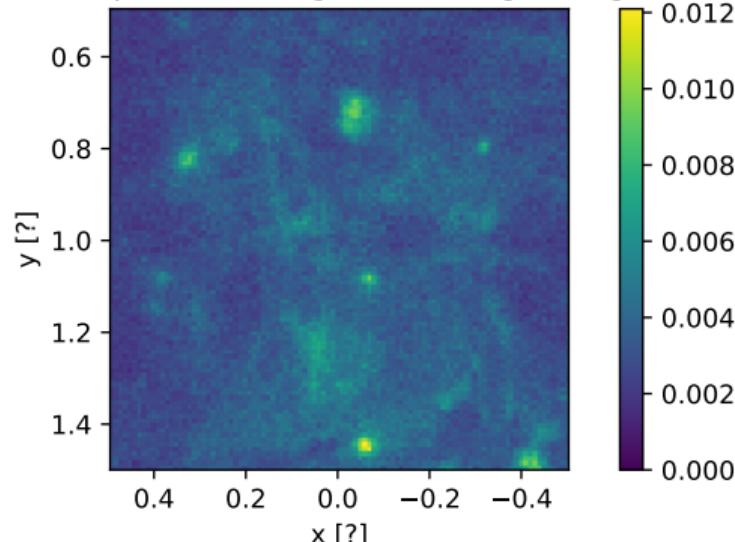
- ▶ `fig.ginput` in a terminal
- ▶ handle mouse click events in a Jupyter notebook

Interactive application (temperature)

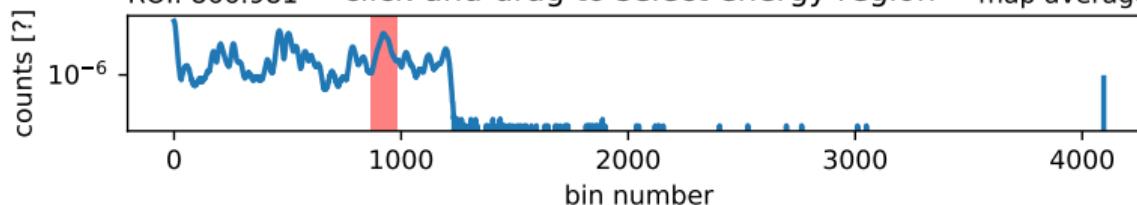


Interactive applications (x-ray fluorescence map)

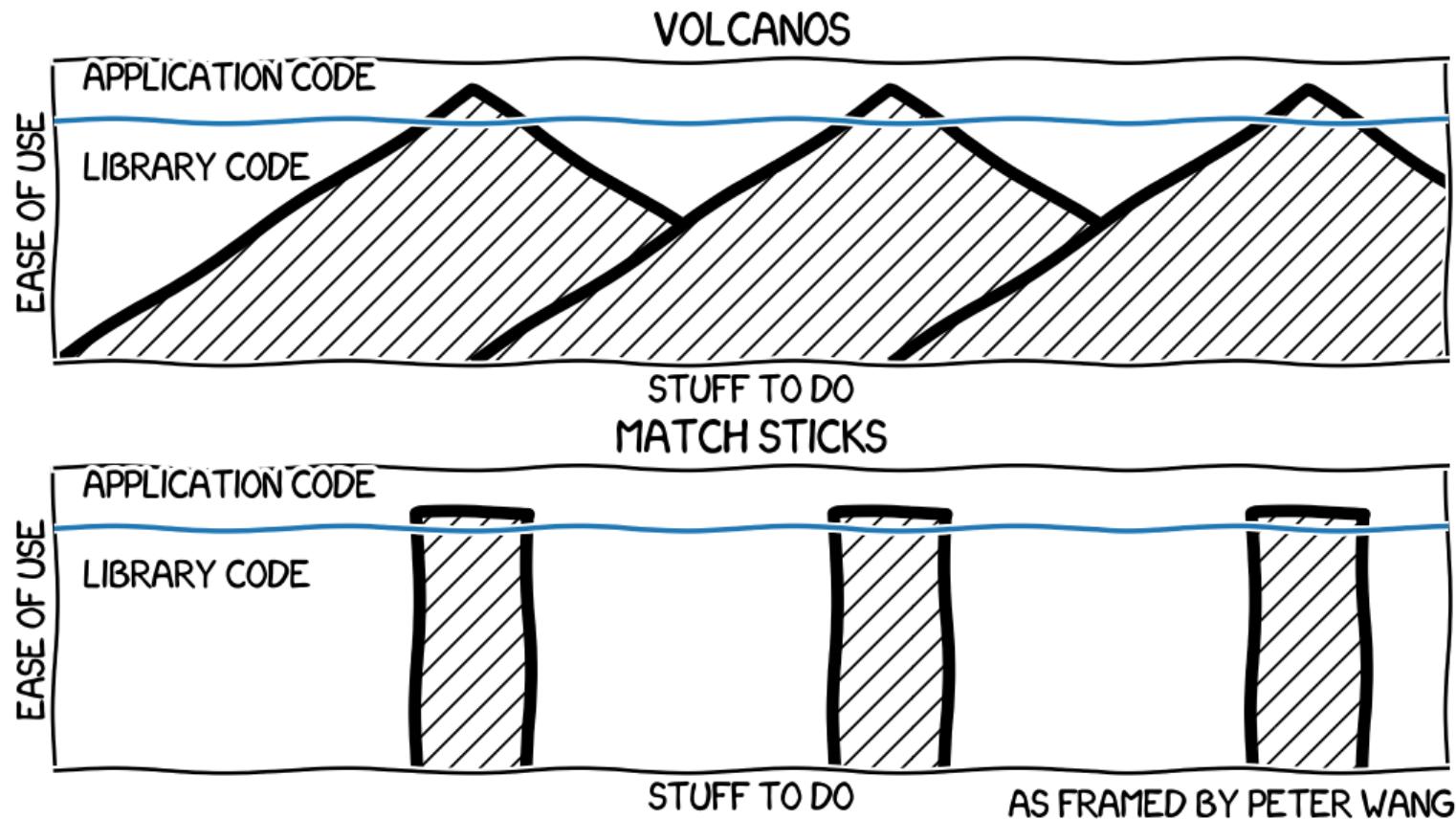
shift-click to select pixel, alt-drag to draw region, right-click to r



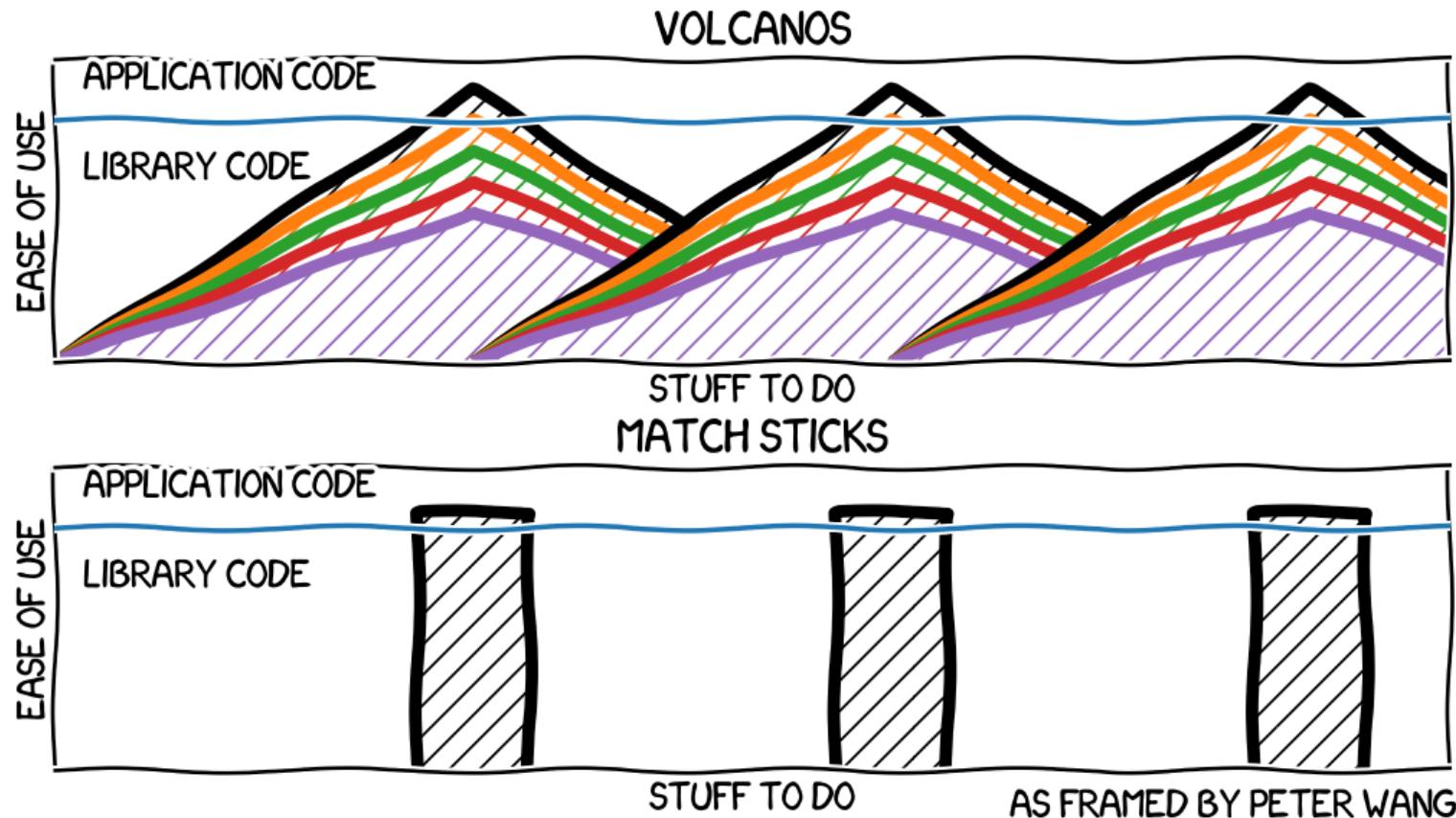
ROI: 866:981 click-and-drag to select energy region map average



Software Development Philosophy



Software Development Philosophy



Future Work

- ▶ On going incremental improvements, bug fixes, and maintenance
- ▶ Improvements to Figure and Axes layout tooling (Jody Klymak)
- ▶ Re-designing Matplotlib's internal data model (Hannah Aizenman)

Resources

This material: https://github.com/tacaswell/2021-03_APS

- ▶ docs: <https://matplotlib.org/stable>
- ▶ cheatsheets: <https://github.com/matplotlib/cheatsheets>
- ▶ chat: <https://gitter.im/matplotlib>
- ▶ forum: <https://discourse.matplotlib.org>
- ▶ tutorials:
 - ▶ https://github.com/matplotlib/interactive_tutorial,
 - ▶ <https://github.com/matplotlib/AnatomyOfMatplotlib>
 - ▶ <https://github.com/matplotlib/GettingStarted>
- ▶ Interactive Applications Using Matplotlib, Benjamin V. Root (2015)
- ▶ domain-specific libraries
- ▶ Building a maintainable plotting library (PyData NYC 2019)
<https://youtu.be/NV4Y75ZUDJA>
- ▶ Separation Of Scales (PyData Global 2020) <https://youtu.be/P85UIuMvnI>