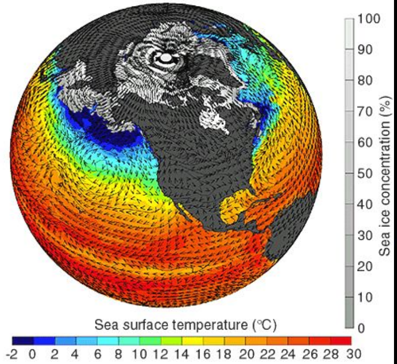
Pysindy

Yuying Liu, Yi Chu, Lianzong Wang

Background

Motivation:



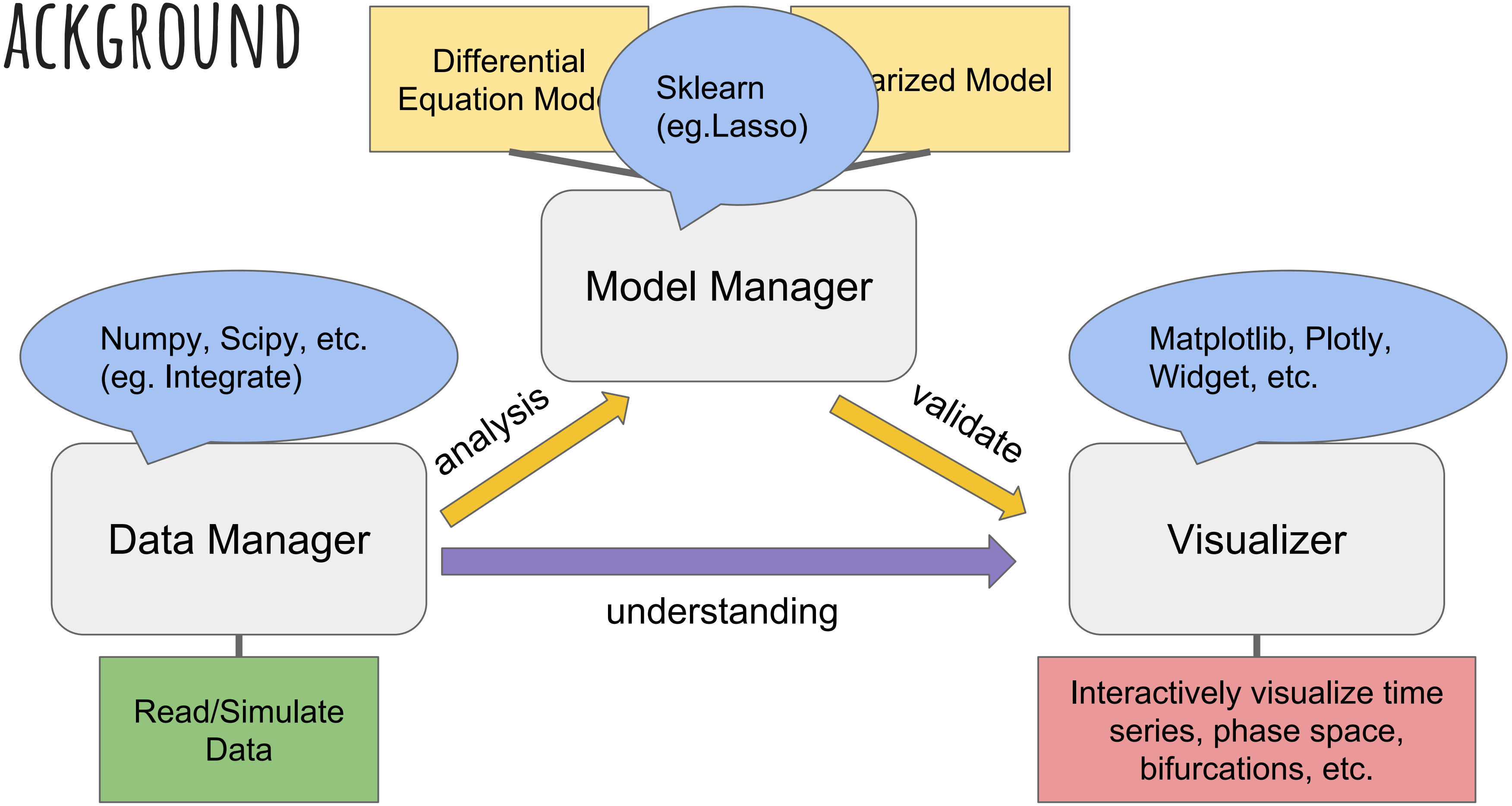
- Dynamical systems are commonly used in many fields, some are too complicated to discover. But data Science can accelerate the process & promote our understandings.
- Pioneer works arise in 2013-2018 (from UW, Princeton, Caltech, etc.) But they are pieces of code everywhere in Github, programmed in different languages.

Background

Questions:

- Given the measurements of state variables, how do we uncover the governing equations? **(Regression)**
- Given the measurements from a nonlinear system, how can we linearize it (easy to control)? **(Linearization)**
- How to get intuitive understanding & benchmark the performance of different algorithms? **(Visualization)**

Background


Differential
Equation Mod
Sklearn
(eg.Lasso)
arized Model
Model Manager
Numpy, Scipy, etc.
(eg. Integrate)
Matplotlib, Plotly,
Widget, etc.
analysis
validate
Data Manager
Visualizer
understanding
Read/Simulate
Data
Interactively visualize time
series, phase space,
bifurcations, etc.

Plotly

Pros:

Fancy

Works well with pandas

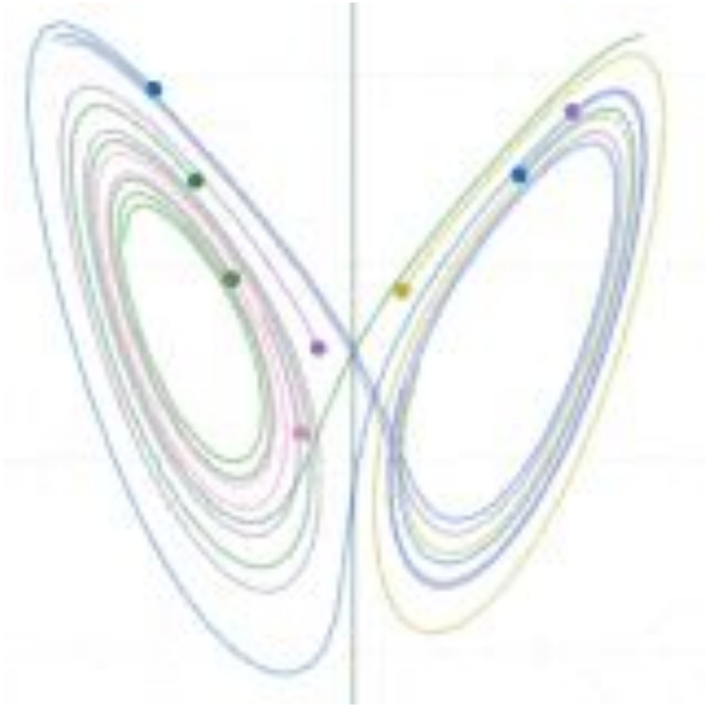
Gives you all kinds of options to manipulate your plot

Easy to implement

Cons:

Every plot you do is on web

PLOTLY-ANIMATION



Plotly

Pros:

Fancy

Works well with pandas

Gives you all kinds of options to manipulate your plot

Easy to implement

Cons:

Every plot you do is on web

Plotly (con't)

```
data = Data([
    Bar(
        x=df[".."],
        y=df[".."]
    )
])
```

Plotly

Pros:

Fancy

Works well with pandas

Gives you all kinds of options to manipulate your plot

Easy to implement

Cons:

Every plot you do is on web

Plotly

Pros:

Fancy

Works well with pandas

Gives you all kinds of options to manipulate your plot

Easy to implement

Cons:

Every plot you do is on web

MATPLOTLIB

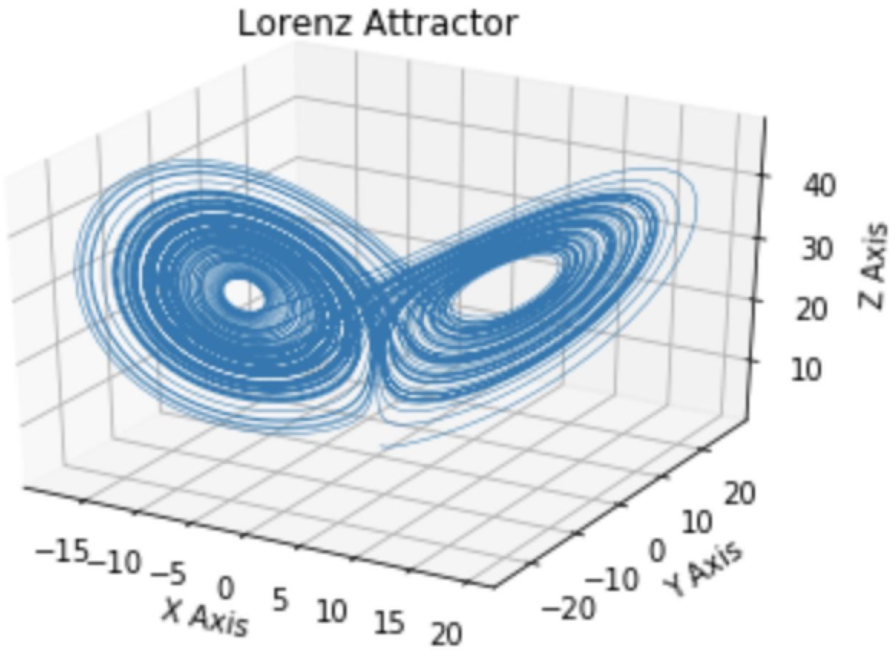
Pro(s):

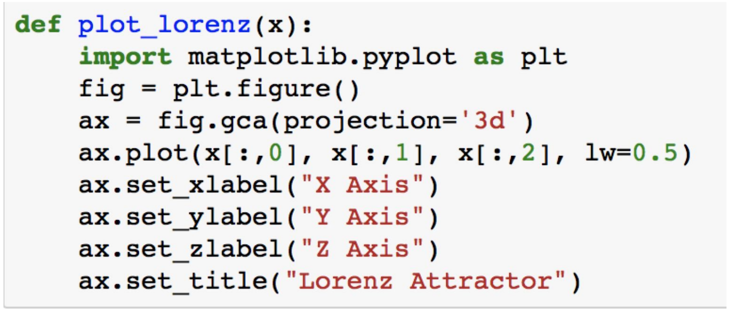
You can basically manipulate every parameter of your graph (if you know how to do it)

Con(s):

Hard to make fancy plots

Matplotlib (con't)





```
def plot_lorenz(x):
    import matplotlib.pyplot as plt
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.plot(x[:,0], x[:,1], x[:,2], lw=0.5)
    ax.set_xlabel("X Axis")
    ax.set_ylabel("Y Axis")
    ax.set_zlabel("Z Axis")
    ax.set_title("Lorenz Attractor")
```

Scipy

```
scipy.integrate.ode(f).set_integrator("..")
```

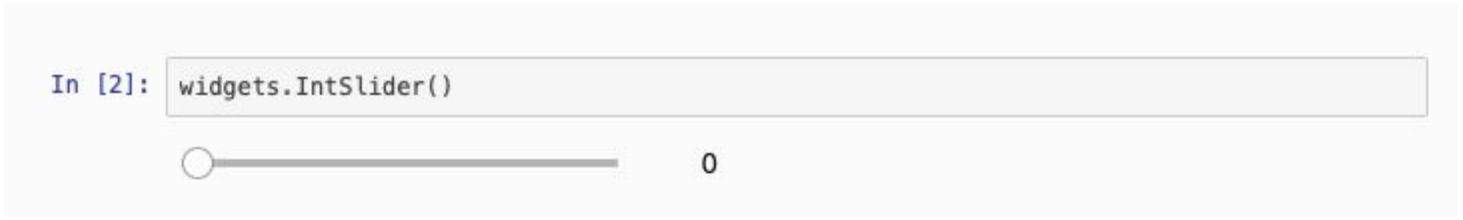
```
ODE45: “dopti5”

ODE15s: “Vode”, method = “bdf”, order = 15
```

Scipy (con't)

```
solver = integrate.ode(lorenz_eqn).set_integrator("dopri5")

solver.set_initial_value(xinit, t0)

for i in range(1, t.size):

    x[i, :] = solver.integrate(t[i])
```

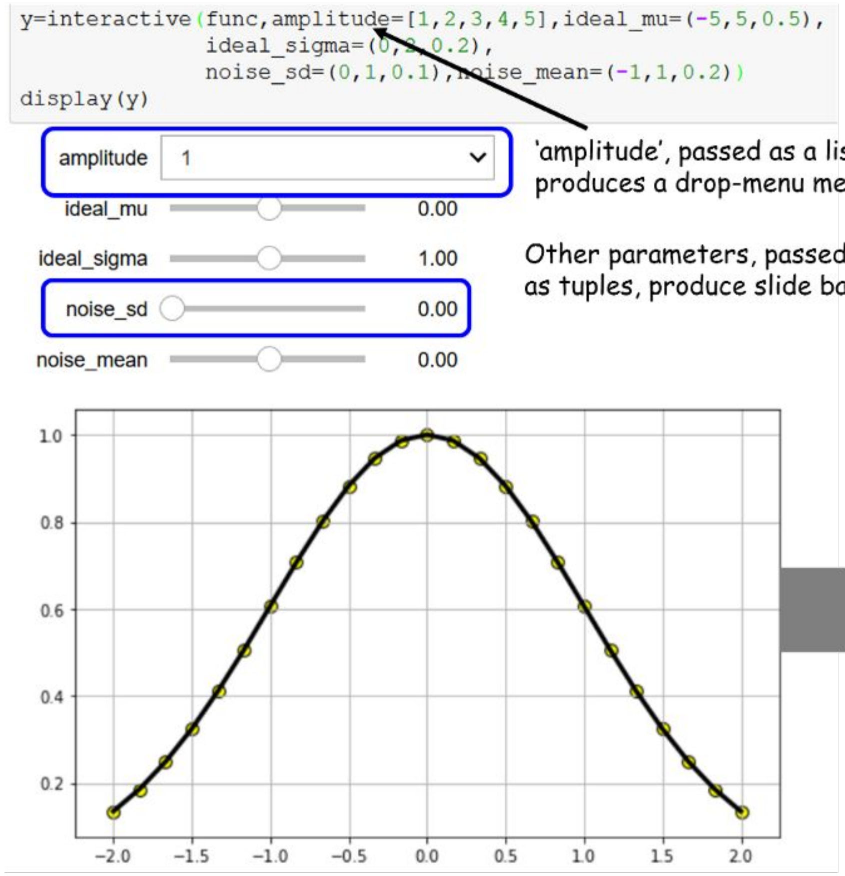
IPython Widget

Widgets are eventful python objects that have a representation in the browser, often as a control like a slider, textbox, etc.



```
In [2]: widgets.IntSlider()
```

After changing some parameters, the plot changes dynamically

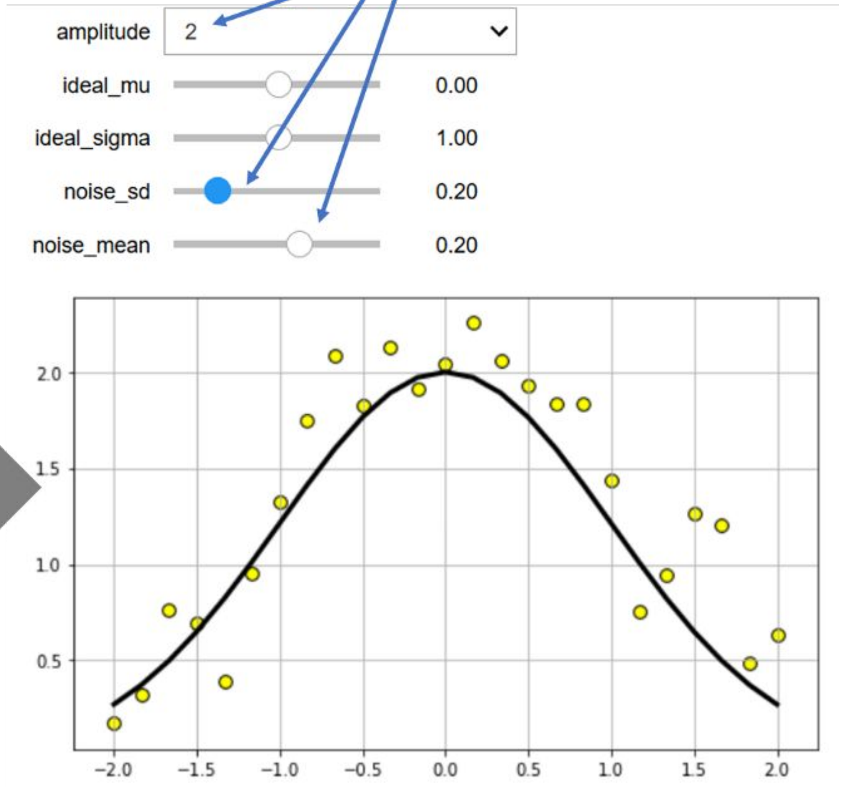


```
y=interactive(func,amplitude=[1,2,3,4,5],ideal_mu=(-5,5,0.5),
                 ideal_sigma=(0,2,0.2),
                 noise_sd=(0,1,0.1),noise_mean=(-1,1,0.2))
display(y)
```

amplitude 1
ideal_mu 0.00
ideal_sigma 1.00
noise_sd 0.00
noise_mean 0.00

'amplitude', passed as a list, produces a drop-menu menu

Other parameters, passed as tuples, produce slide bar



amplitude 2
ideal_mu 0.00
ideal_sigma 1.00
noise_sd 0.20
noise_mean 0.20

Numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.