Programming as Physicists

Prakash Gautam
[प्रकाश गौतम]

February 3, 2020

Jupyter Lab

- Great tool for prototyping python
- Allows interactive sessions, useful while developing code
- Can be run in a powerful server, and viewed in local browser (not specific to jupyter only)

Caution

Don't overuse jupyter notebooks.

Running on Server

- `ssh` tunneling allows accessing web services running on a server.
- Since jupyter starts a https web service, we can use ssh to access that

```
user@local ~$ ssh username@server
username@server ~$ cd working/directory
username@server ~/working/directory $ jupyter lab --no-browser --port=8831
```

Let this shell running

```
user@local ~$ ssh username@server -NL 8831:localhost:8831
```

Access `http://localhost:8831` from local

- End result is accessing remotely running web service in local browser.

Sympy

- Mathematica/Wolframalpha, Maxima, Matlab/Octave Symbolic,
- Python has `Sympy`.
  - Supports various latex output
  - Has lot of mathematics and physics library
  - Extremely useful and easy to use

Cadabra

- Anybody can appreciate how notorious symbolic tensor analysis are
- Sympy has tensor modules, but are not very intuitive to work
- Cadabra simply blows my mind
  - It is very intuitive, minimal
  - Supports almost Latex style input
  - Can be used in jupyter notebooks

Cadabra

Blows my mind

[7]:
```
# Define the Riemann tensor
RDEF:= RT^{\alpha}_{\beta \mu \nu} = {\Gamma^{\alpha}_{\sigma \mu}*\Gamma
     -       {\Gamma^{\alpha}_{\sigma \nu}*\Gamma^{\sigma}_{\beta \mu}} +
    -\partial_{\nu}{\Gamma^{\alpha}_{\beta \mu}}
    +  \partial_{\mu}{\Gamma^{\alpha}_{\beta \nu}};
```

[7]: $RT^{\alpha}{}_{\beta\mu\nu} = \Gamma^{\alpha}{}_{\sigma\mu}\Gamma^{\sigma}{}_{\beta\nu} - \Gamma^{\alpha}{}_{\sigma\nu}\Gamma^{\sigma}{}_{\beta\mu} - \partial_{\nu}\Gamma^{\alpha}{}_{\beta\mu} + \partial_{\mu}\Gamma^{\alpha}{}_{\beta\nu}$

[8]:
```
RTUP = substitute(RDEF,CS);
```

[8]:

$$RT^{\alpha}{}_{\beta\mu\nu} = \square_{\sigma\mu}{}^{\alpha}\begin{cases} \square_{\phi r}{}^{\phi} = r^{-1} \\ \square_{\phi\theta}{}^{\phi} = (\tan\theta)^{-1} \\ \square_{\theta r}{}^{\theta} = r^{-1} \\ \square_{rr}{}^{r} = \partial_r\Lambda \\ \square_{tr}{}^{t} = \partial_r\Phi \\ \square_{r\phi}{}^{\phi} = r^{-1} \\ \square_{\theta\phi}{}^{\phi} = (\tan\theta)^{-1} \\ \square_{r\theta}{}^{\theta} = r^{-1} \\ \square_{tr}{}^{t} = \partial_r\Phi \\ \square_{\phi\phi}{}^{r} = -r\exp(-2\Lambda)(\sin\theta)^2 \\ \square_{\phi\phi}{}^{\theta} = -\frac{1}{2}\sin(2\theta) \\ \square_{\theta\theta}{}^{r} = -r\exp(-2\Lambda) \\ \square_{tt}{}^{r} = \exp(-2\Lambda+2\Phi)\partial_r\Phi \end{cases} \quad \square_{\beta\nu}{}^{\sigma}\begin{cases} \square_{\phi r}{}^{\phi} = r^{-1} \\ \square_{\phi\theta}{}^{\phi} = (\tan\theta)^{-1} \\ \square_{\theta r}{}^{\theta} = r^{-1} \\ \square_{rr}{}^{r} = \partial_r\Lambda \\ \square_{tr}{}^{t} = \partial_r\Phi \\ \square_{r\phi}{}^{\phi} = r^{-1} \\ \square_{\theta\phi}{}^{\phi} = (\tan\theta)^{-1} \\ \square_{r\theta}{}^{\theta} = r^{-1} \\ \square_{tr}{}^{t} = \partial_r\Phi \\ \square_{\phi\phi}{}^{r} = -r\exp(-2\Lambda)(\sin\theta)^2 \\ \square_{\phi\phi}{}^{\theta} = -\frac{1}{2}\sin(2\theta) \\ \square_{\theta\theta}{}^{r} = -r\exp(-2\Lambda) \\ \square_{tt}{}^{r} = \exp(-2\Lambda+2\Phi)\partial_r\Phi \end{cases}$$

Programming Paradigm

- Modular programming
- We write (several) modules/functions to accomplish task

```
def histogram(x,bins=10):
    __ = plt.hist(x,bins=bins)
```

```
x = np.random.normal(0,1,1000)
histogram(x,bins=100)
```

Programming Paradigm

- Object Oriented Programming (OOP)

```
class Histogram():
    def __init__(self,x,bins=10,weights=None):
        H,be,bv = np.histogram(x,bins=bins)
        Hc,be,bv = np.histogram(x,bins=bins,weights=None)
        self.H = H/Hc
    def plot(self,ax=None):
        if ax is None:
            fig,ax = plt.subplots()

        ax.plot(self.H,ls='steps')
```

```
x = np.random.normal(0,1,1000)
h = Histogram(x,bins=100)
h.plot()
```

Make Distributable Code

- Assume other people are going to use your code.
- Portable code is better maintainable.
- It is lot more extensible. (From my painful experience)

Python Packages

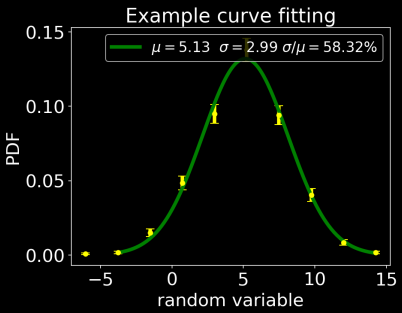
- Python modular hierarchy
  - Statements/Expression
  - functions/Modules
  - Class
  - package
  - Library
- An (possibly emmpty) `___init__.py` tells python that it is a package
- Python looks for library/package in `PYTHONPATH` environment variable

```
`-- pgsahist
    |-- __init__.py
    |-- hist.py
    |-- plot.py
    `-- utilities.py
```

```
export PYTHONPATH=package/location:$PYTHONPATH
```

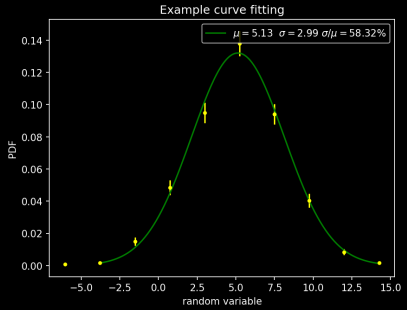
Matplotlib styling

- We can use style files to customize plots
- Comes handy when we need different version of same plot
- Requires no modification in code
- `~/.config/matplotlib/stylelib/mystyle.mplstyle` file
- In the code use `plt.style.use("mystyle")`
- S



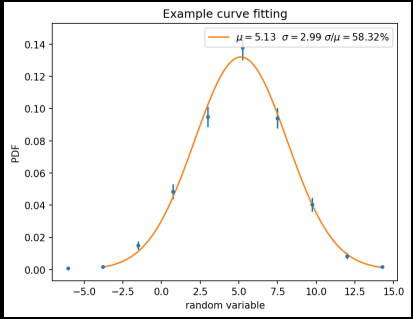
Matplotlib styling

- We can use style files to customize plots
- Comes handy when we need different version of same plot
- Requires no modification in code
- `~/.config/matplotlib/stylelib/mystyle.mplstyle` file
- In the code use `plt.style.use("mystyle")`
- S



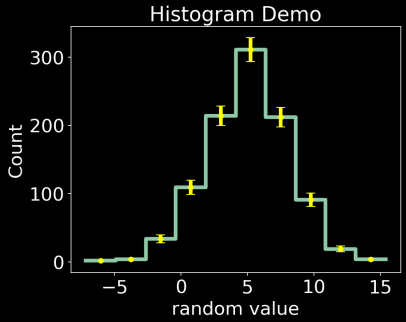
Matplotlib styling

- We can use style files to customize plots
- Comes handy when we need different version of same plot
- Requires no modification in code
- `~/.config/matplotlib/stylelib/mystyle.mplstyle` file
- In the code use `plt.style.use("mystyle")`
- S



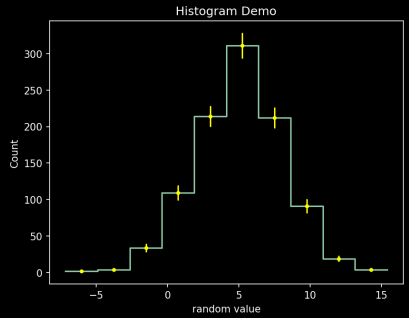
Matplotlib styling

- We can use style files to customize plots
- Comes handy when we need different version of same plot
- Requires no modification in code
- `~/.config/matplotlib/stylelib/mystyle.mplstyle` file
- In the code use `plt.style.use("mystyle")`
- S



Matplotlib styling

- We can use style files to customize plots
- Comes handy when we need different version of same plot
- Requires no modification in code
- `~/.config/matplotlib/stylelib/mystyle.mplstyle` file
- In the code use `plt.style.use("mystyle")`
- S



Matplotlib styling

- We can use style files to customize plots
- Comes handy when we need different version of same plot
- Requires no modification in code
- `~/.config/matplotlib/stylelib/mystyle.mplstyle` file
- In the code use `plt.style.use("mystyle")`
- S