Visualization and optimization

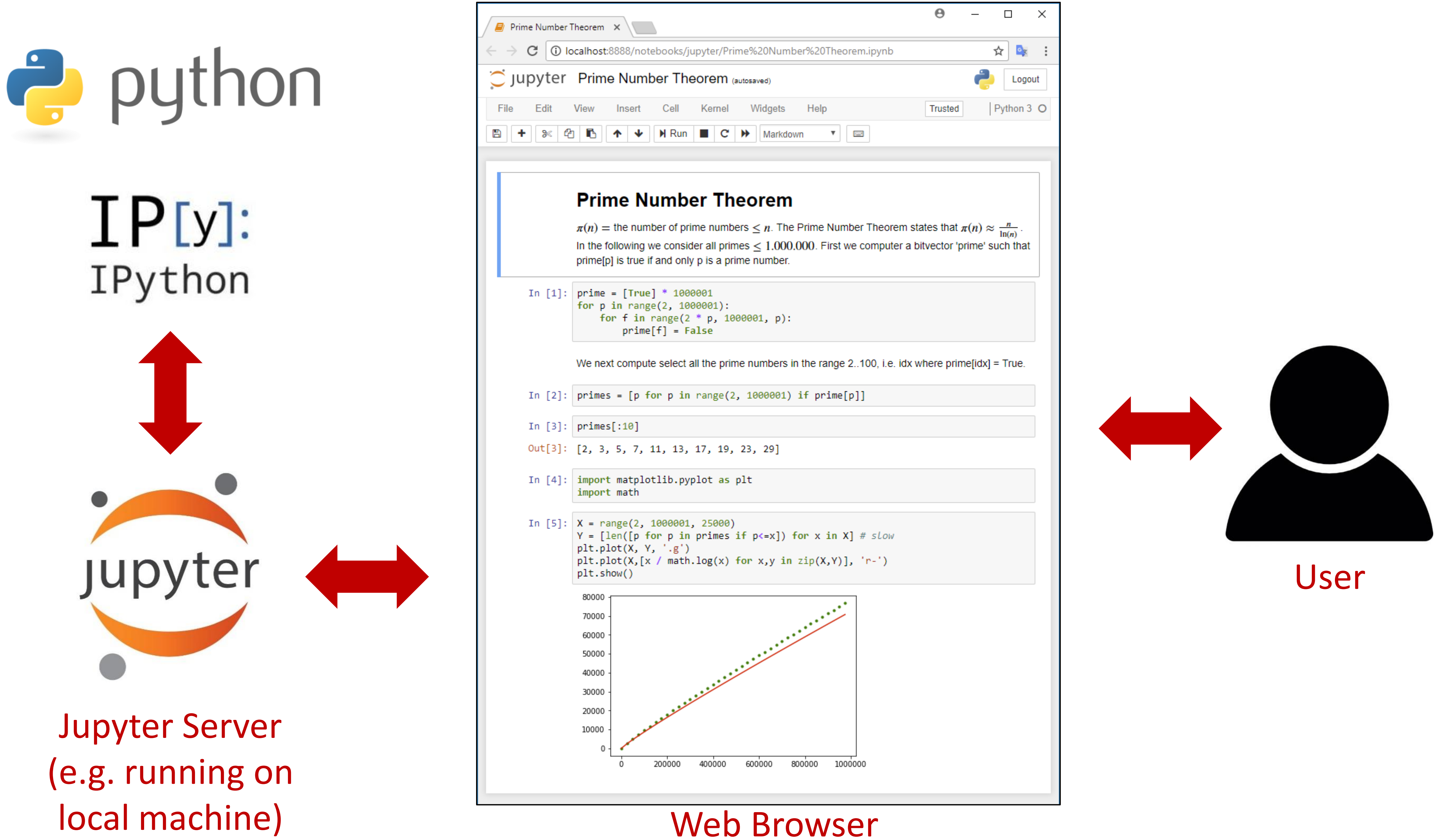
- Jupyter
- Matplotlib
- scipy.optimize.minimize

jupyter

The Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

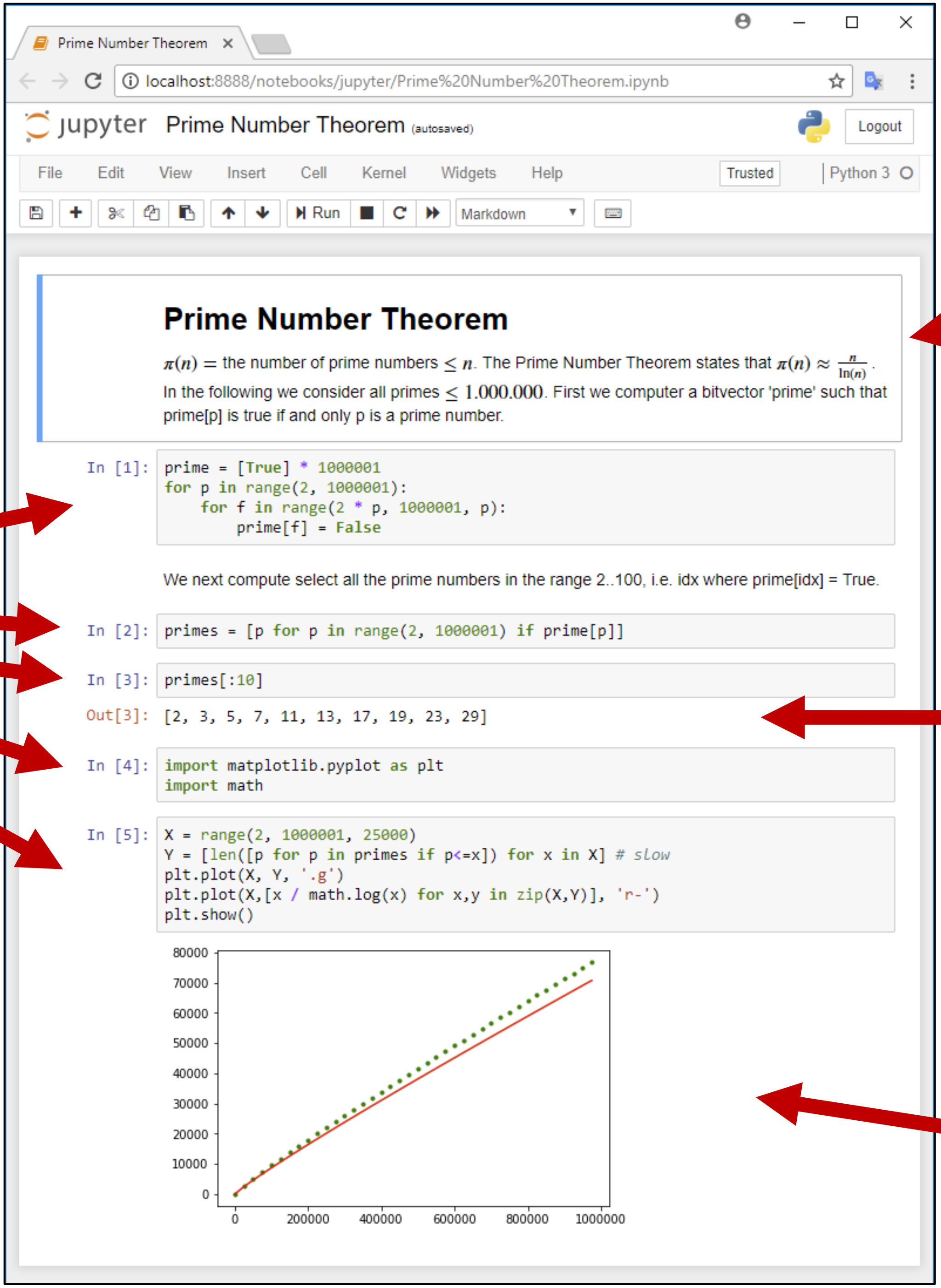
jupyter.org


python
IP[y]:
IPython
jupyter
Jupyter Server
(e.g. running on
local machine)
Prime Number Theorem
localhost:8888/notebooks/jupyter/Prime%20Number%20Theorem.ipynb
jupyter Prime Number Theorem (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
Prime Number Theorem
π(n) = the number of prime numbers ≤ n. The Prime Number Theorem states that π(n) ≈ n/ln(n).
In the following we consider all primes ≤ 1.000.000. First we computer a bitvector 'prime' such that prime[p] is true if and only p is a prime number.
In [1]:
prime = [True] * 1000001
for p in range(2, 1000001):
    for f in range(2 * p, 1000001, p):
        prime[f] = False
We next compute select all the prime numbers in the range 2..100, i.e. idx where prime[idx] = True.
In [2]:
primes = [p for p in range(2, 1000001) if prime[p]]
In [3]:
primes[:10]
Out[3]:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
In [4]:
import matplotlib.pyplot as plt
import math
In [5]:
X = range(2, 1000001, 25000)
Y = [len([p for p in primes if p<=x]) for x in X] # slow
plt.plot(X, Y, '.g')
plt.plot(X,[x / math.log(x) for x,y in zip(X,Y)], 'r-')
plt.show()
80000
70000
60000
50000
40000
30000
20000
10000
0
0
200000
400000
600000
800000
1000000
Web Browser
User

cells

formatted text: Markdown / LaTeX / HTML / ...

python code

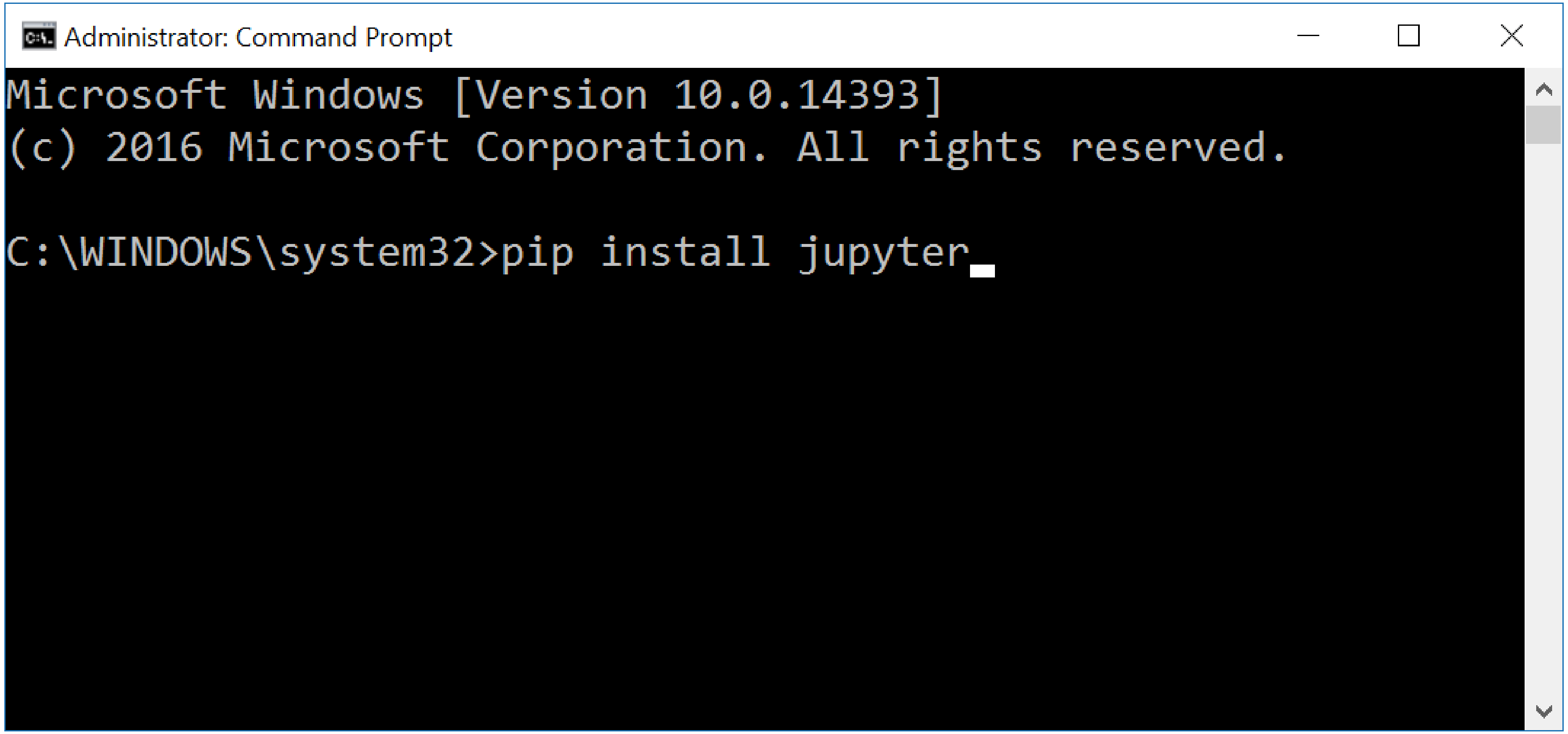


python shell output

matplotlib / numpy / ... output

Jupyter - installing

- Open a windows shell and run: `pip install jupyter`

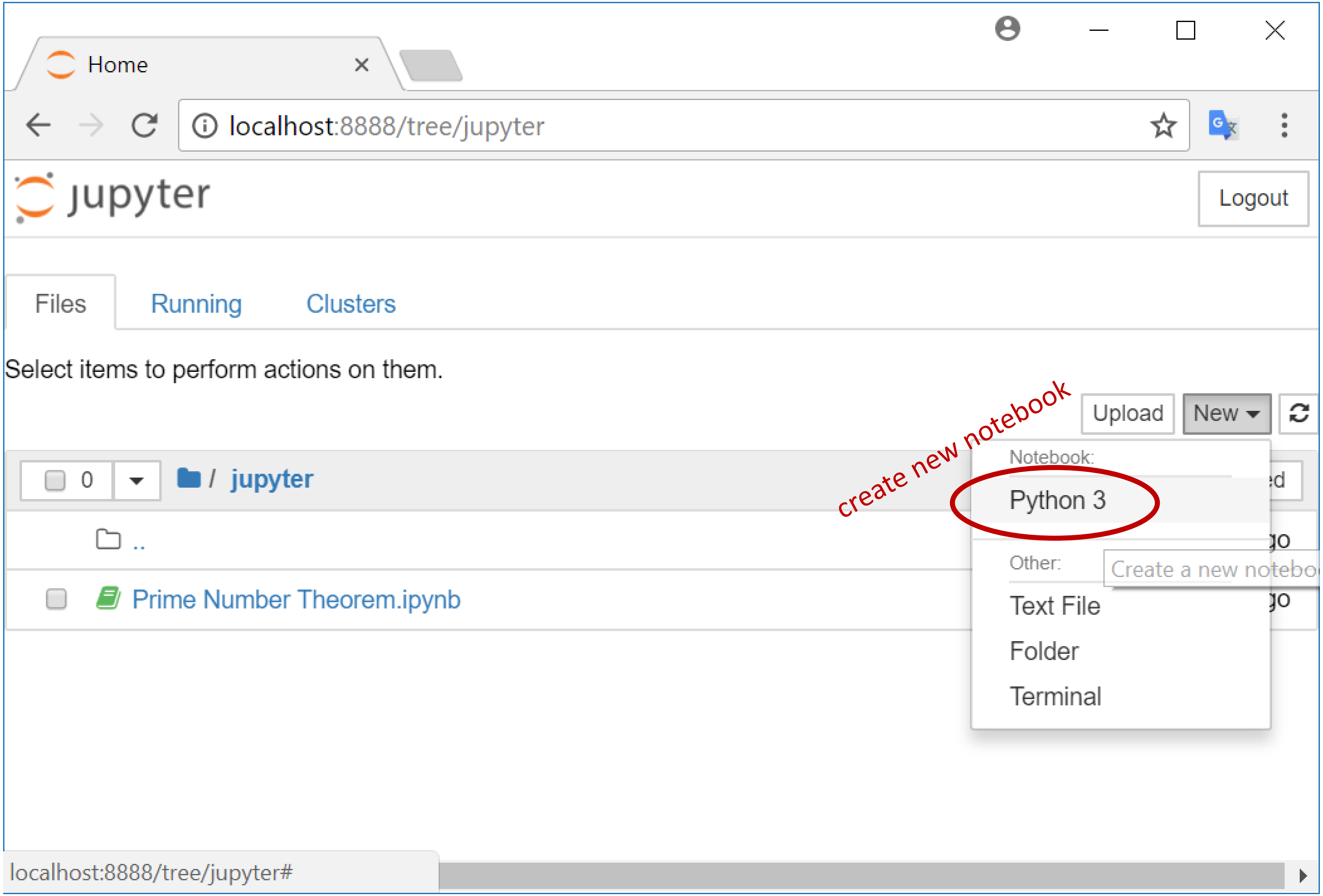


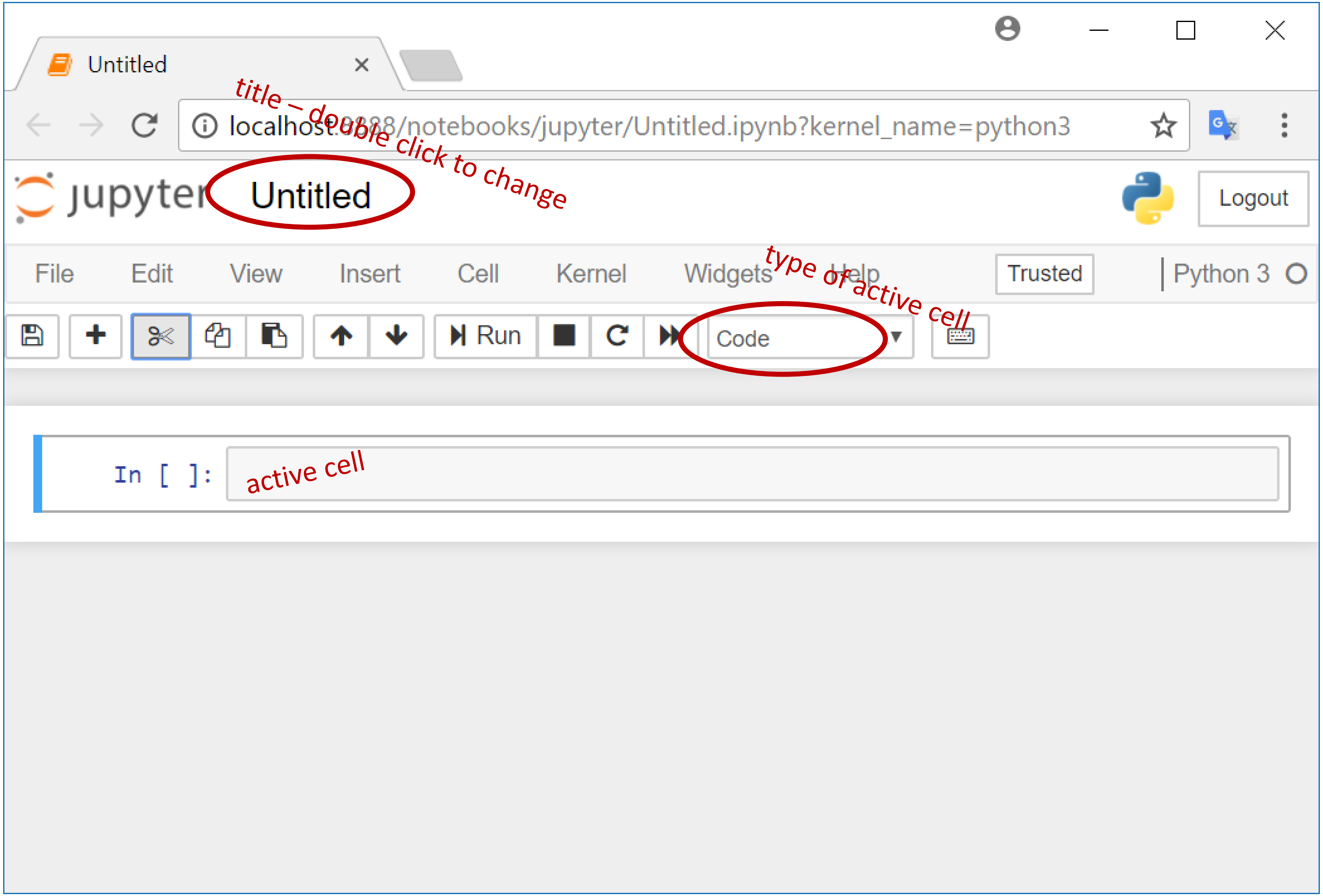
Jupyter – launching the jupyter server

- Open a windows shell and run: `jupyter notebook`

Command Prompt - jupyter notebook

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\au121>jupyter notebook
[I 17:36:08.584 NotebookApp] Serving notebooks from local directory: C:\Users\au121
[I 17:36:08.584 NotebookApp] 0 active kernels
[I 17:36:08.584 NotebookApp] The Jupyter Notebook is running at:
[I 17:36:08.584 NotebookApp] http://localhost:8888/?token=18644cfeedab986bbdcc068f61c21b06b62
548a090169463
[I 17:36:08.584 NotebookApp] Use Control-C to stop this server and shut down all kernels (twi
ce to skip confirmation).
[C 17:36:08.584 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=18644cfeedab986bbdcc068f61c21b06b62548a090169463
[I 17:36:09.154 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```


Home
localhost:8888/tree/jupyter
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
create new notebook
Upload
New
Notebook:
Python 3
Other:
Create a new notebo
Text File
Folder
Terminal
0
jupyter
..
Prime Number Theorem.ipynb
localhost:8888/tree/jupyter#

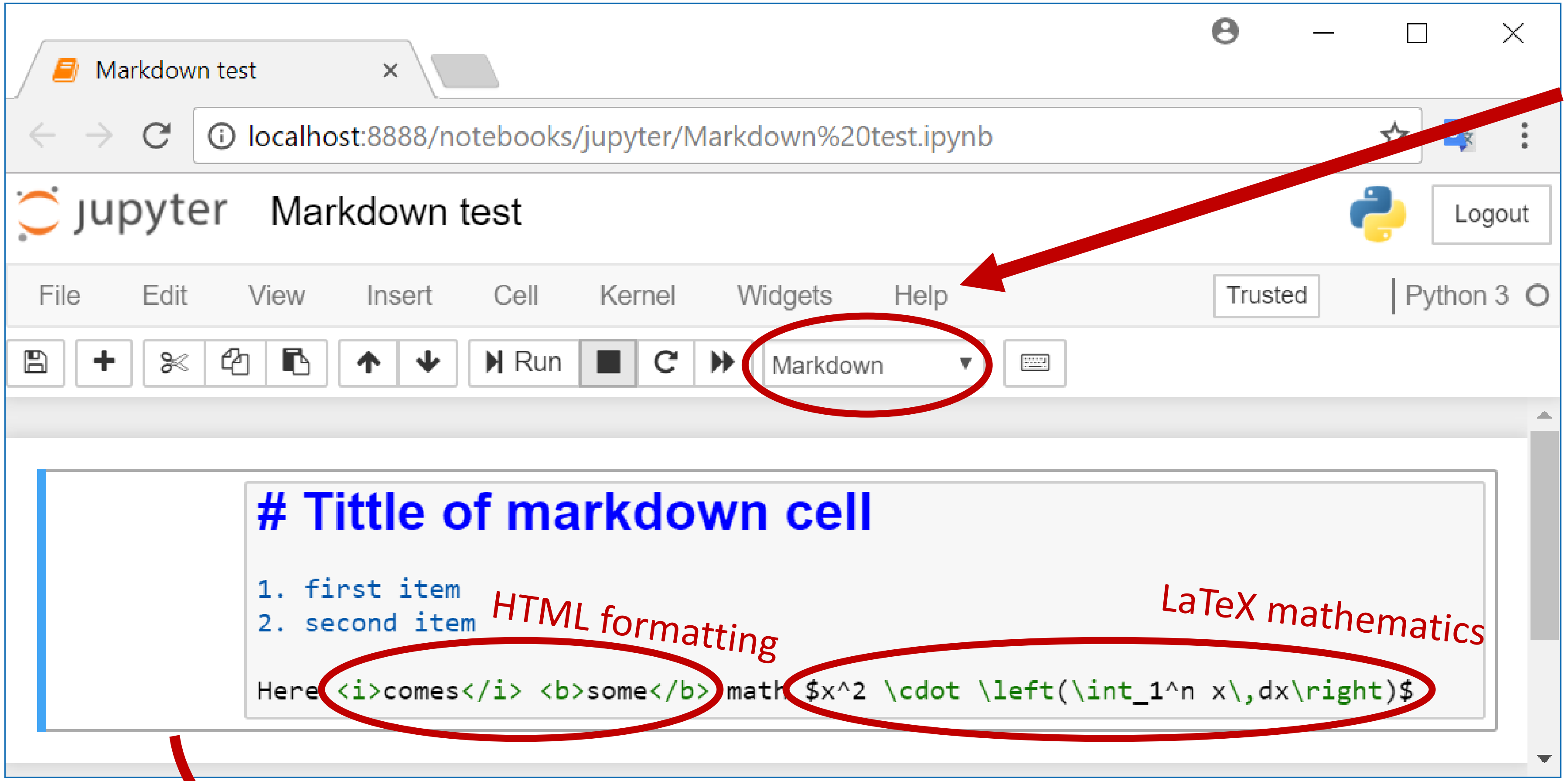


Untitled
localhost:8888/notebooks/jupyter/Untitled.ipynb?kernel_name=python3
jupyter
Untitled
title – double click to change
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
type of active cell
Run
Code
In [ ]:
active cell

Try:

Help > User Interface Tour

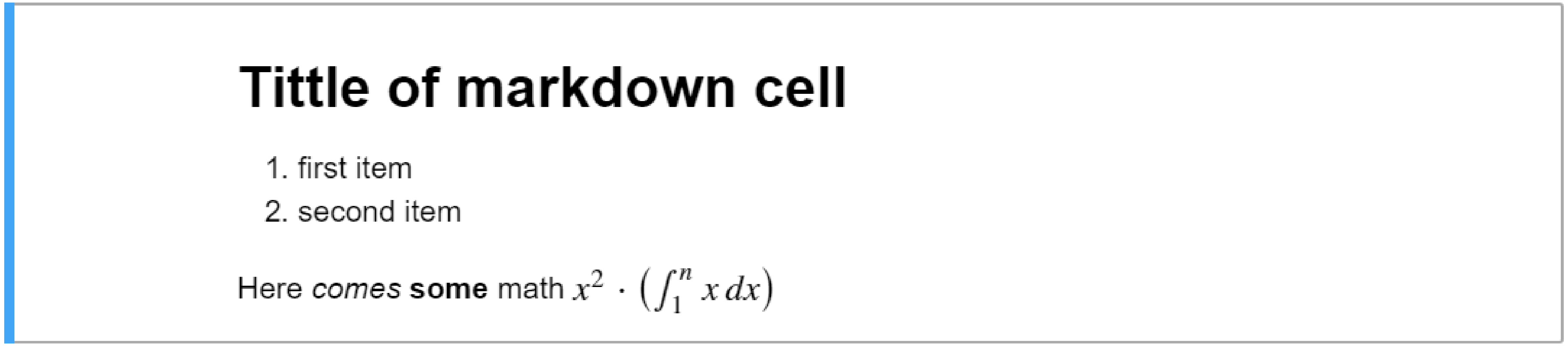
Help > Markdown



after pressing

Ctrl + Enter (evaluates)

Alt + Enter (evaluates + new cell)



github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Jupyter

- Widespread tool used for data science applications
- Documentation, code for data analysis, and resulting visualizations are stored in one common format
- Easy to update visualizations
- Works with about 100 different programming languages (not only Python 3), many special features, ...
- Easy to share data analysis

- *Many online tutorials and examples are available*

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

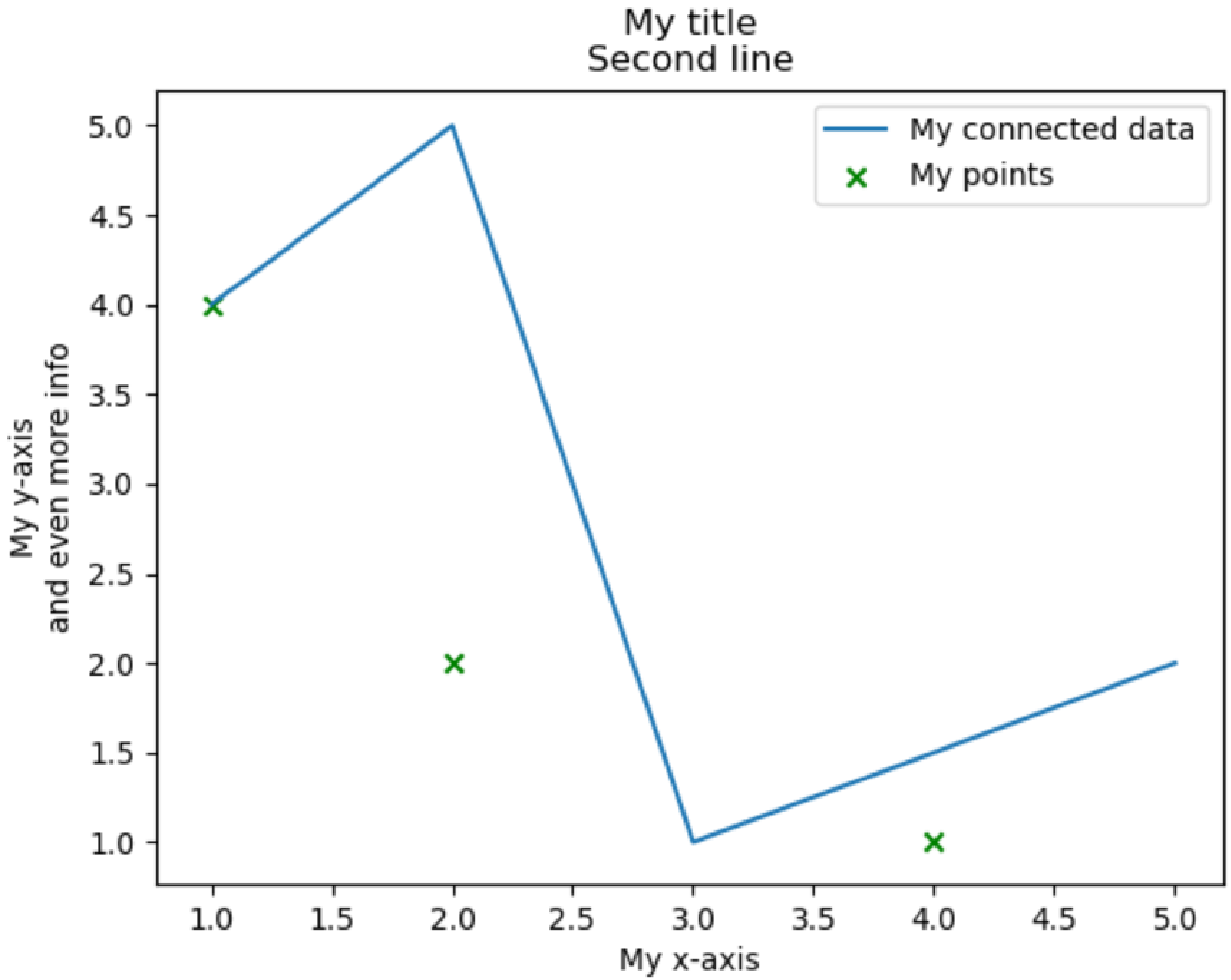
matplotlib.org

Some simple matplotlib examples

Plot and Scatter

matplotlib-plot.py

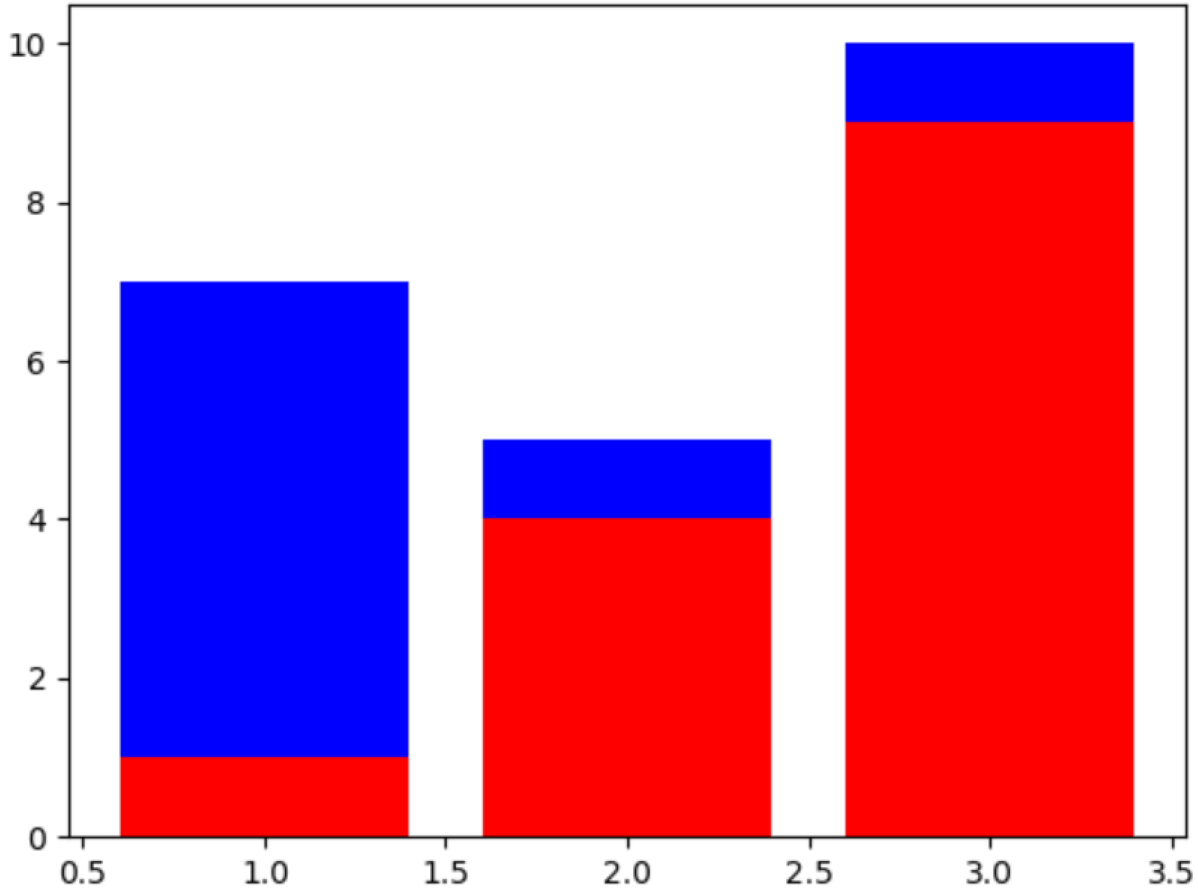
```
import matplotlib.pyplot as plt

x1 = [1, 2, 3, 5]
y1 = [4, 5, 1, 2]

x2 = [1, 2, 4]
y2 = [4, 2, 1]

plt.plot(x1, y1, label='My connected data')
plt.scatter(x2, y2, label='My points',
            marker='x', color='green')
plt.xlabel('My x-axis')
plt.ylabel('My y-axis\nand even more info')
plt.title('My title\nSecond line')
plt.legend()
plt.show()
```



Bars

matplotlib-bars.py

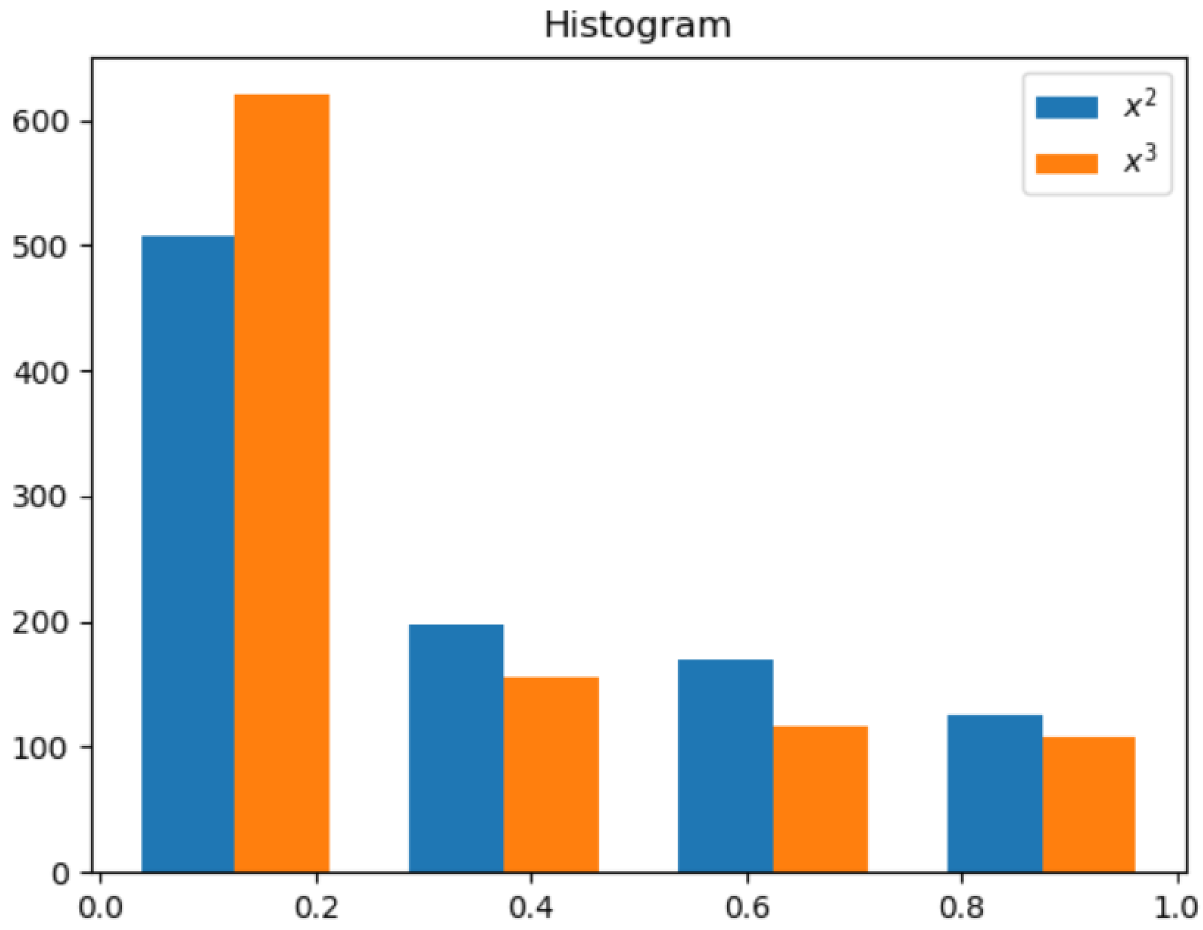
```
import matplotlib.pyplot as plt

x = [1, 2, 3]
y = [7, 5, 10]

plt.bar(x, y, color='blue')
plt.bar(x, [v**2 for v in x], color='red')
plt.show()
```



Histogram

matplotlib-histogram.py

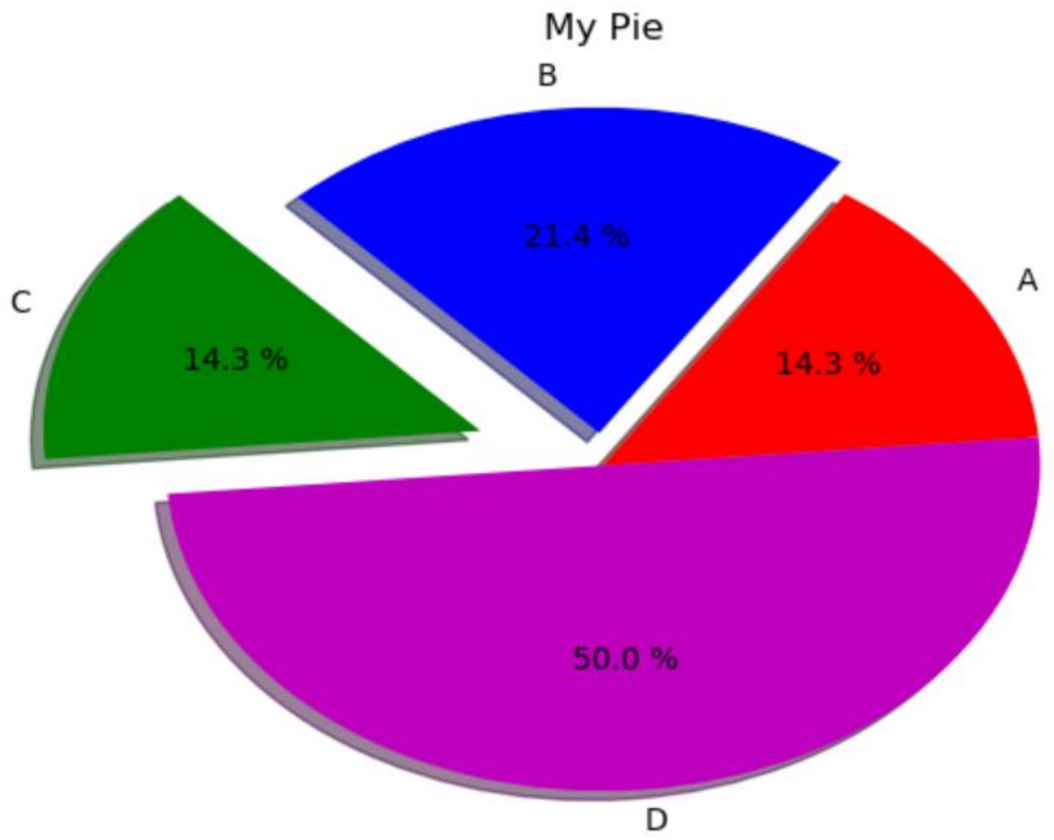
```
import matplotlib.pyplot as plt
from random import random

values1 = [random()**2 for _ in range(1000)]
values2 = [random()**3 for _ in range(1000)]

bins = [0.0, 0.25, 0.5, 0.75, 1.0]

plt.hist([values1, values2], bins, histtype='bar',
         rwidth=0.7, label=['$x^2$', '$x^3$'])
plt.title('Histogram')
plt.legend()
plt.show()
```



Pie

matplotlib-pie.py

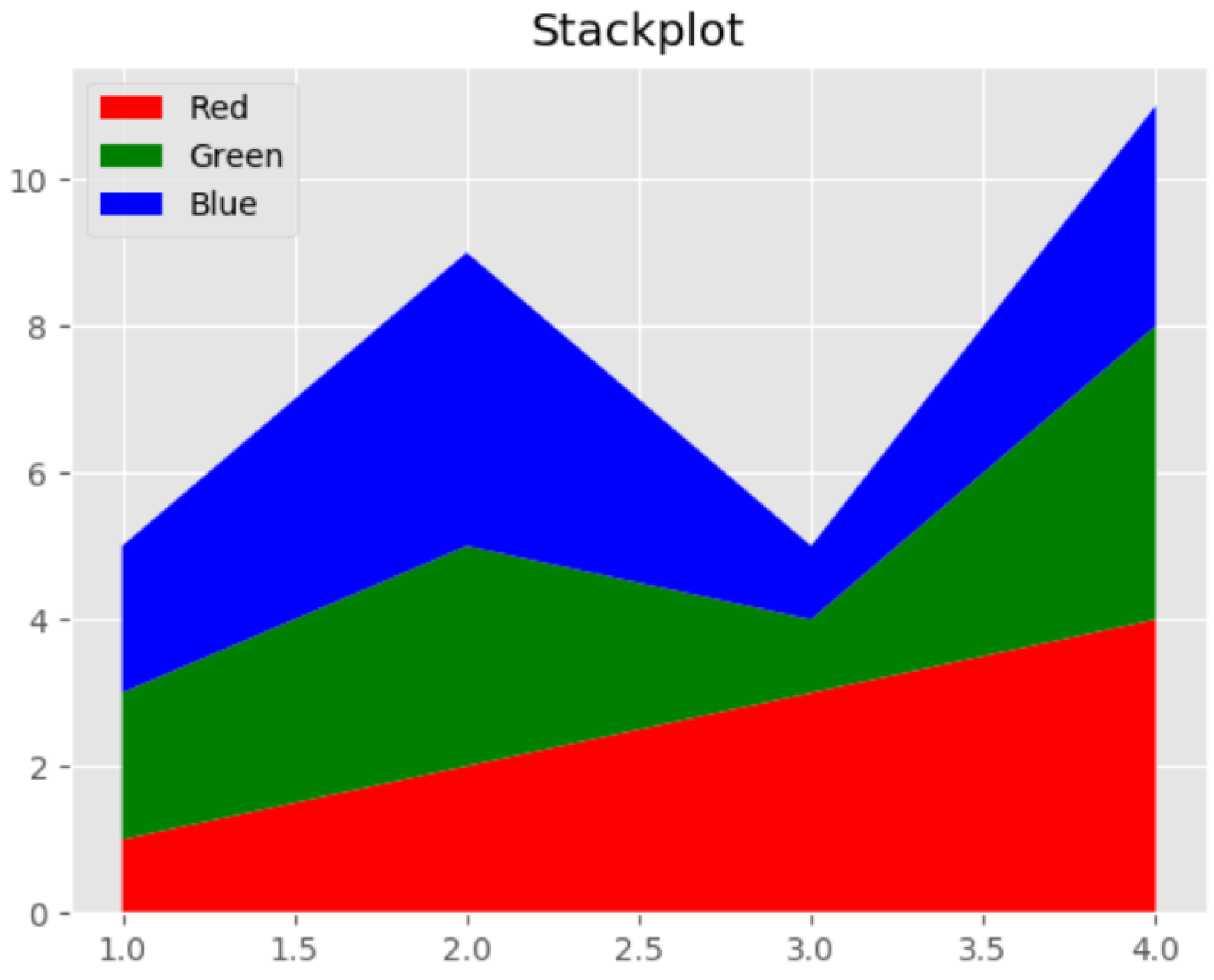
```
import matplotlib.pyplot as plt

plt.title('My Pie')
plt.pie([2, 3, 2, 7],
        labels=['A','B','C','D'],
        colors=['r', 'b', 'g', 'm'],
        startangle=5,
        shadow=True,
        explode=(0, 0.1, 0.3, 0),
        autopct='%1.1f %%'   # percent formatting
        )
plt.show()
```



Stackplot

matplotlib-stackplot.py

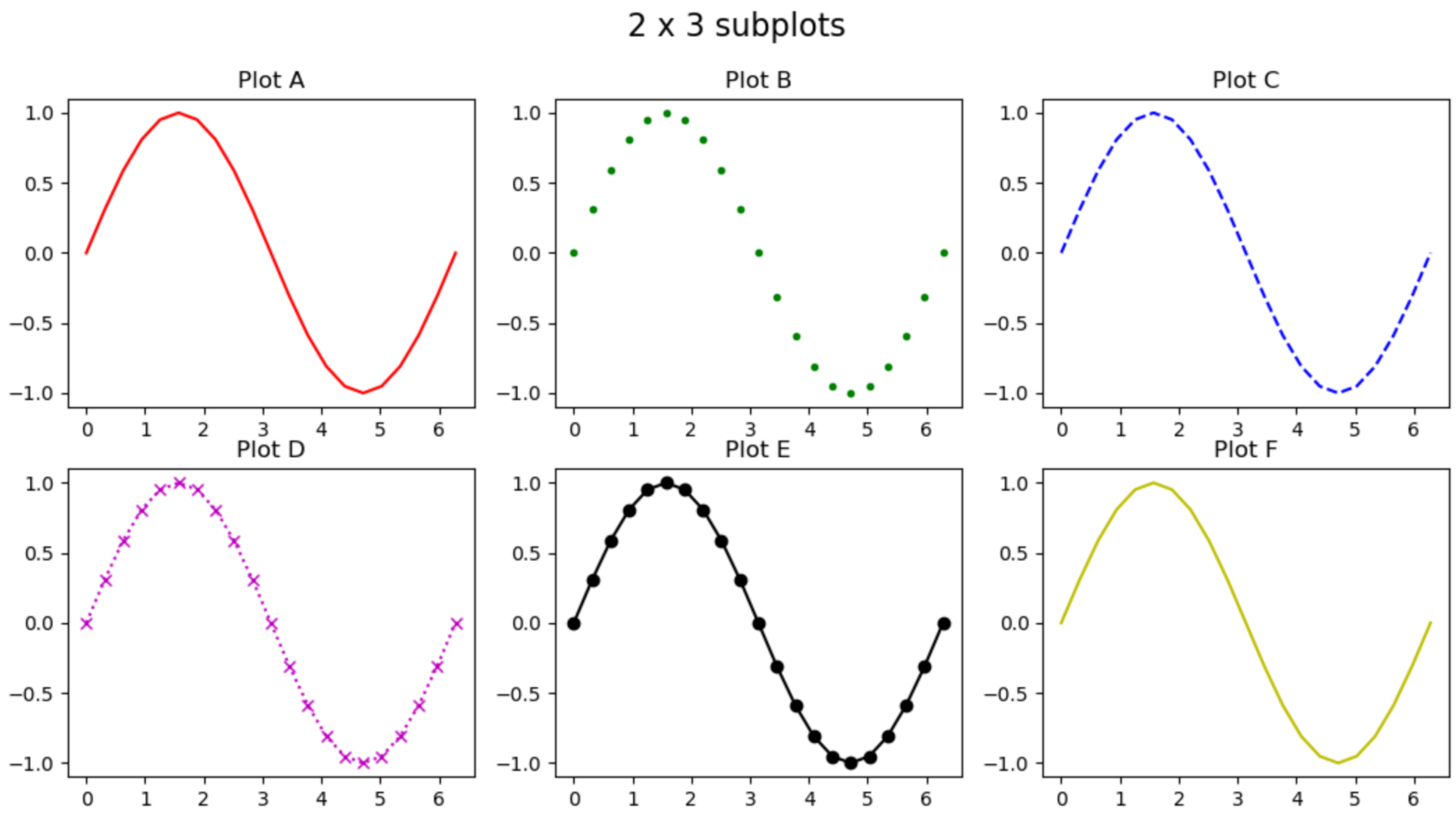
```
import matplotlib.pyplot as plt
from matplotlib import style

style.use('ggplot')

x = [1, 2, 3, 4]

y1 = [1, 2, 3, 4]
y2 = [2, 3, 1, 4]
y3 = [2, 4, 1, 3]

plt.title('Stackplot')

plt.stackplot(x, y1, y2, y3,
              colors=['r', 'g', 'b'],
              labels=["Red", "Green", "Blue"])
plt.grid(True)
plt.legend(loc=2)
plt.show()
```



Subplots (2 rows, 3 columns)

matplotlib-subplots.py

```
import matplotlib.pyplot as plt
import math

x_min, x_max, n = 0, 2 * math.pi, 20

x = [x_min + (x_max - x_min) * i / n
      for i in range(n + 1)]
y = [math.sin(v) for v in x]

plt.subplot(2, 3, 1)
plt.plot(x, y, 'r-')
plt.title('Plot A')

plt.subplot(2, 3, 2)
plt.plot(x, y, 'g.')
plt.title('Plot B')

plt.subplot(2, 3, 3)
plt.plot(x, y, 'b--')
plt.title('Plot C')

plt.subplot(2, 3, 4)
plt.plot(x, y, 'mx:')
plt.title('Plot D')

plt.subplot(2, 3, 5)
plt.plot(x, y, 'ko-')
plt.title('Plot E')

plt.subplot(2, 3, 6)
plt.plot(x, y, 'y')
plt.title('Plot F')

plt.suptitle('2 x 3 subplots', fontsize=16)

plt.show()
```

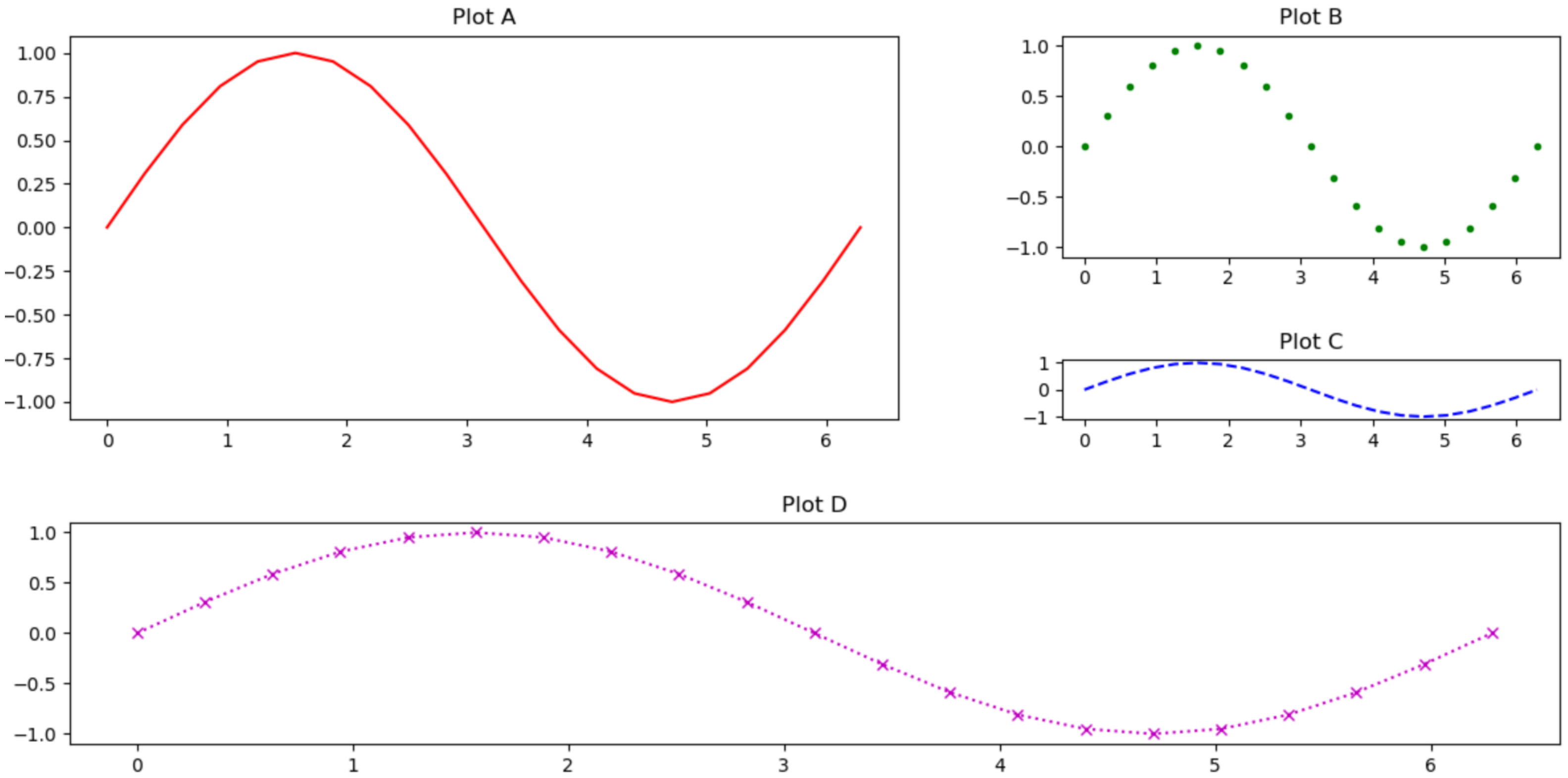


subplot2grid (5 x 5)

matplotlib-subplots.py

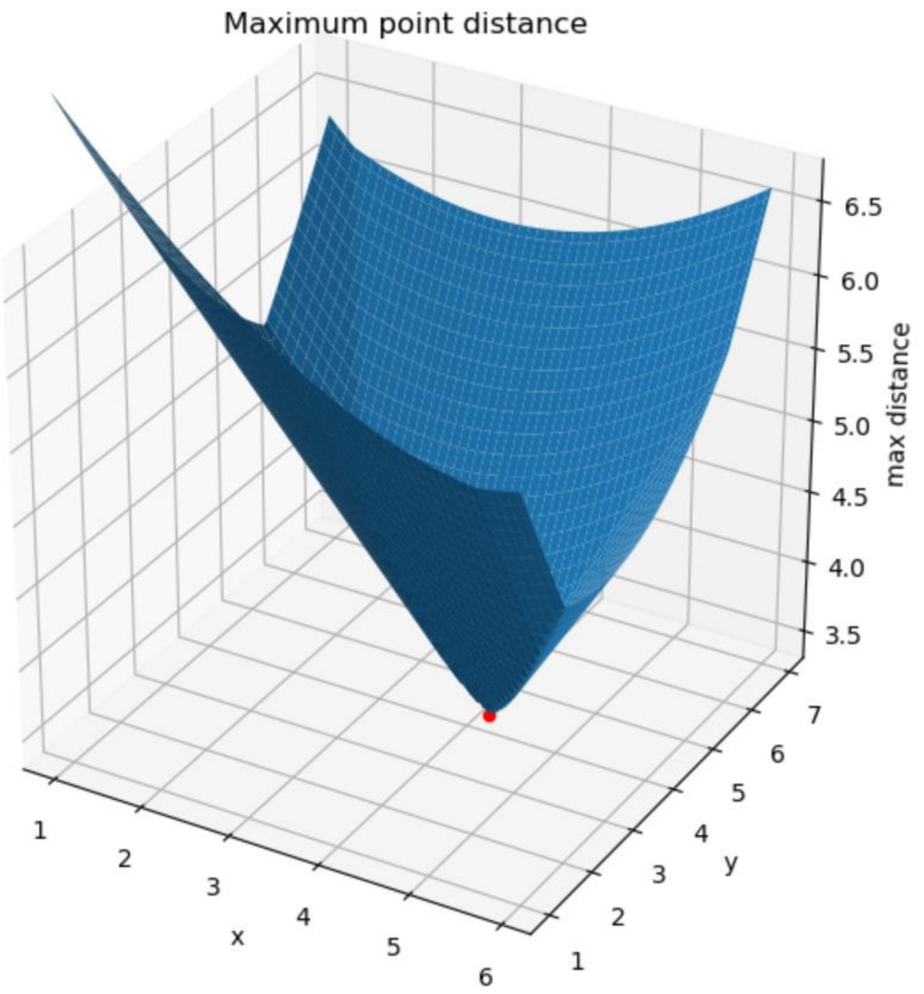
```
import matplotlib.pyplot as plt
import math

x_min, x_max, n = 0, 2 * math.pi, 20

x = [x_min + (x_max - x_min) * i / n
     for i in range(n + 1)]
y = [math.sin(v) for v in x]

plt.subplot2grid((5, 5), (0,0),
                 rowspan=3, colspan=3)
plt.plot(x, y, 'r-')
plt.title('Plot A')
plt.subplot2grid((5, 5), (0,3),
                 rowspan=2, colspan=2)
plt.plot(x, y, 'g.')
plt.title('Plot B')
plt.subplot2grid((5, 5), (2,3),
                 rowspan=1, colspan=2)
plt.plot(x, y, 'b--')
plt.title('Plot C')
plt.subplot2grid((5, 5), (3,0),
                 rowspan=2, colspan=5)
plt.plot(x, y, 'mx:')
plt.title('Plot D')
plt.tight_layout()
plt.show()
```

upper left corner



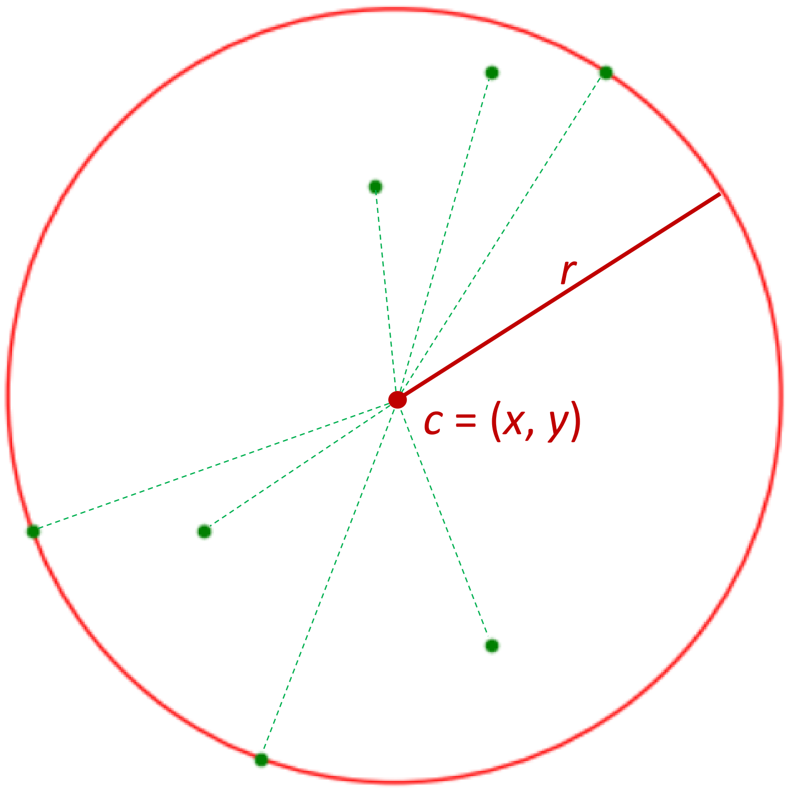
scipy.optimize.minimize

- Find point p minimizing function f
- Supports 13 algorithms – but no guarantee that result correct
- Knowledge about optimization will help you know what optimization algorithm to select and what parameters to provide for better results
- ⚠ WARNING ⚠
  Many solvers return the wrong value



https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

Example: Minimum enclosing circle

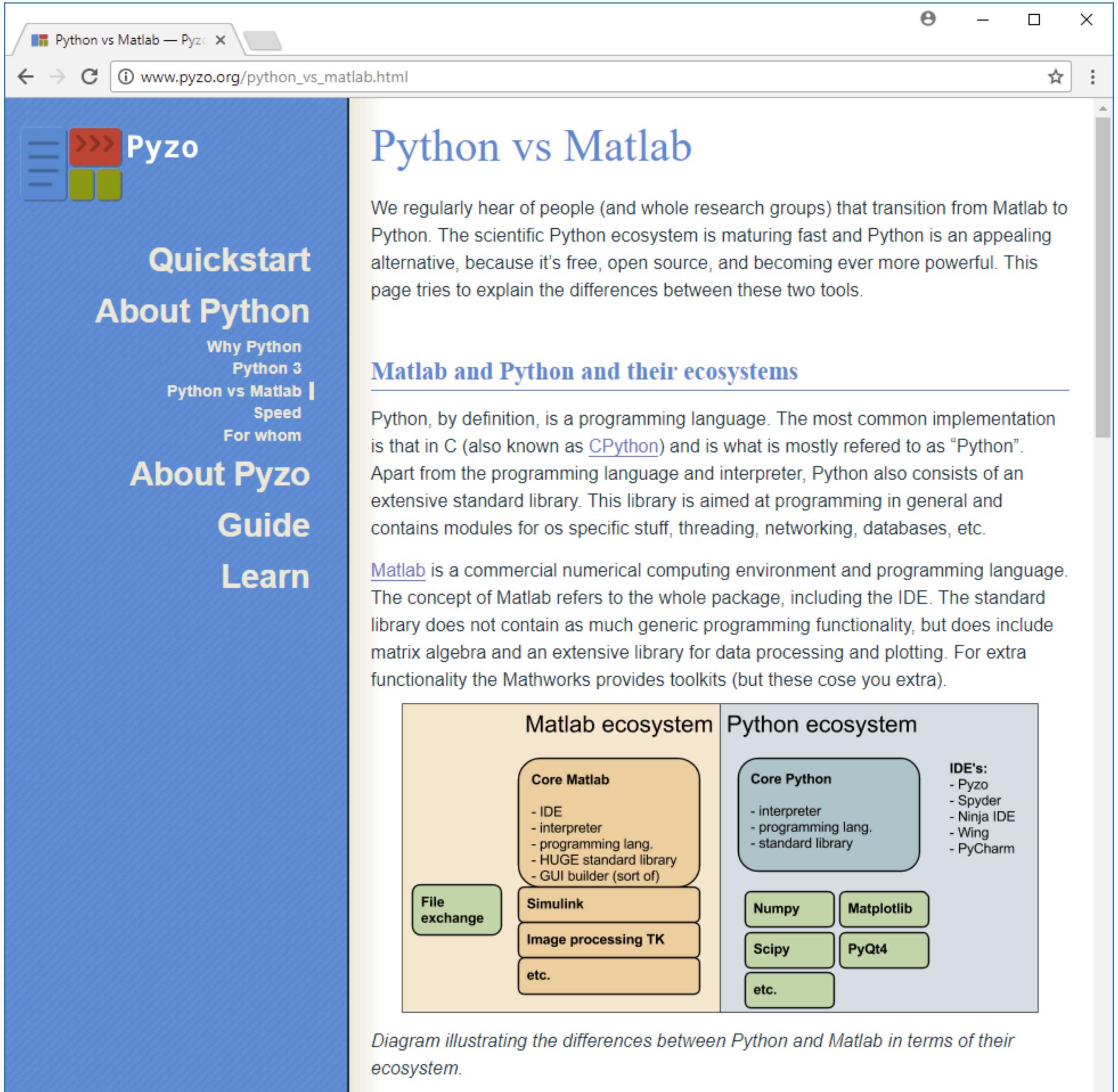


- Find c such that $r = \max_p |p - c|$ is minimized
- A solution is characterized by either 1) three points on circle, where the triangle contains the circle center 2) two opposite points on diagonal
- Try a standard numeric minimization solver
- ⚠ Computation involves max and $\sqrt{x}$, which can be hard for numeric optimization solvers

Python/scipy vs MATLAB

Some basic differences

- “**end**” closes a MATLAB block
- “**;**” at end of command avoids command output
- a(i) instead a[i]
- 1^{st} element of a list a(1)
- a(i:j) includes both a(i) and a(j)



Python vs Matlab

We regularly hear of people (and whole research groups) that transition from Matlab to Python. The scientific Python ecosystem is maturing fast and Python is an appealing alternative, because it’s free, open source, and becoming ever more powerful. This page tries to explain the differences between these two tools.

Matlab and Python and their ecosystems

Python, by definition, is a programming language. The most common implementation is that in C (also known as CPython) and is what is mostly refered to as “Python”. Apart from the programming language and interpreter, Python also consists of an extensive standard library. This library is aimed at programming in general and contains modules for os specific stuff, threading, networking, databases, etc.

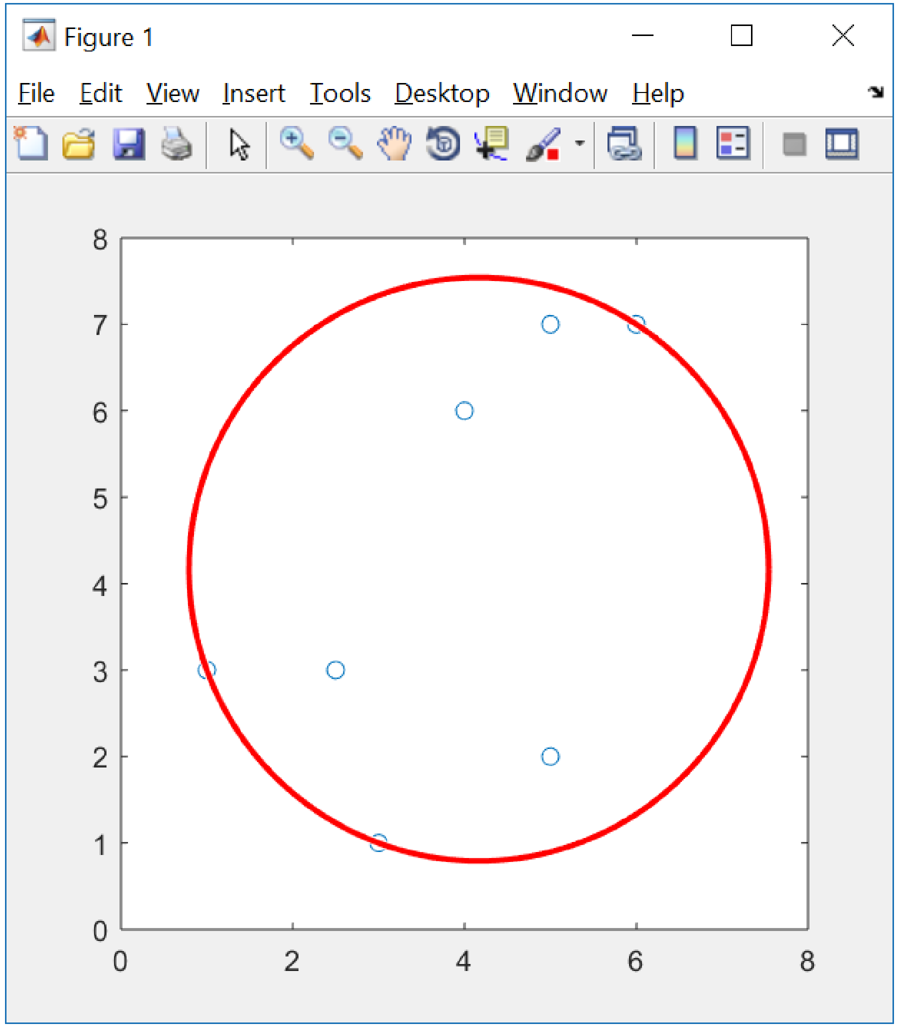
Matlab is a commercial numerical computing environment and programming language. The concept of Matlab refers to the whole package, including the IDE. The standard library does not contain as much generic programming functionality, but does include matrix algebra and an extensive library for data processing and plotting. For extra functionality the Mathworks provides toolkits (but these cose you extra).

Diagram illustrating the differences between Python and Matlab in terms of their ecosystem.

Minimum enclosing circle in MATLAB

enclosing_circle.m

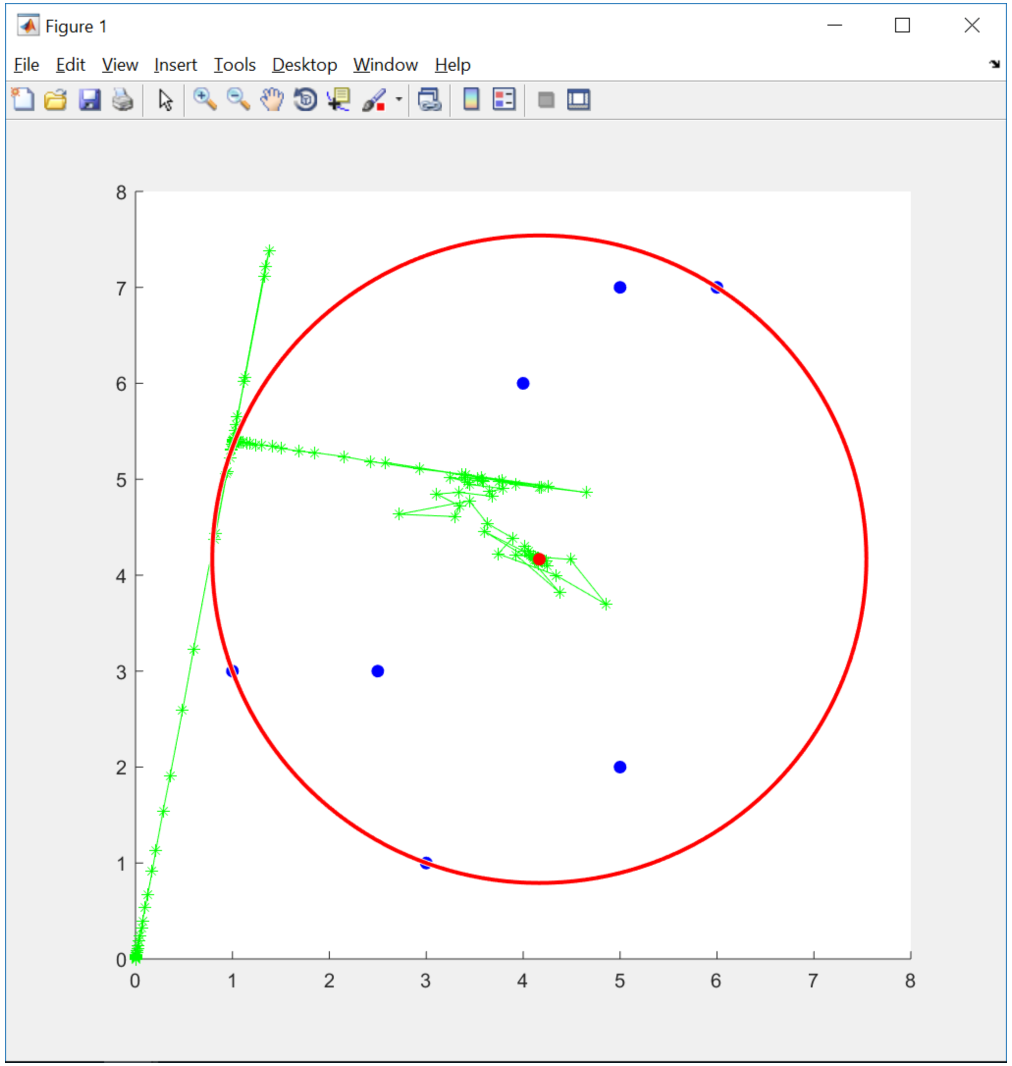
```
% Minimum enclosing circle of a point set
% fminsearch uses the Nelder-Mead algorithm

global x y
x = [1.0, 3.0, 2.5, 4.0, 5.0, 6.0, 5.0];
y = [3.0, 1.0, 3.0, 6.0, 7.0, 7.0, 2.0];
c = fminsearch(@(x) max_distance(x), [0,0]);
plot(x, y, "o");
viscircles(c, max_distance(c));

function dist = max_distance(p)
    global x y
    dist = 0.0;
    for i=1:length(x)
        dist = max(dist, pdist([p; x(i), y(i)],
                                        'euclidean'));
    end
end
```



Minimum enclosing circle in MATLAB (trace)

enclosing_circle_trace.m

```
global x y trace_x trace_y

x = [1.0, 3.0, 2.5, 4.0, 5.0, 6.0, 5.0];
y = [3.0, 1.0, 3.0, 6.0, 7.0, 7.0, 2.0];
trace_x = [];
trace_y = [];

c = fminsearch(@(x) max_distance(x), [0,0]);

hold on
plot(x, y, "o", 'color', 'b', 'MarkerFaceColor', 'b');
plot(trace_x, trace_y, "*-", "color", "g");
plot(c(1), c(2), "o", 'color', 'r', 'MarkerFaceColor', 'r');
viscircles(c, max_distance(c),"color","red");

function dist = max_distance(p)
    global x y trace_x trace_y
    trace_x = [trace_x, p(1)];
    trace_y = [trace_y, p(2)];

    dist = 0.0;
    for i=1:length(x)
        dist = max(dist, pdist([p; x(i), y(i)], 'euclidean' ));
    end
end
```

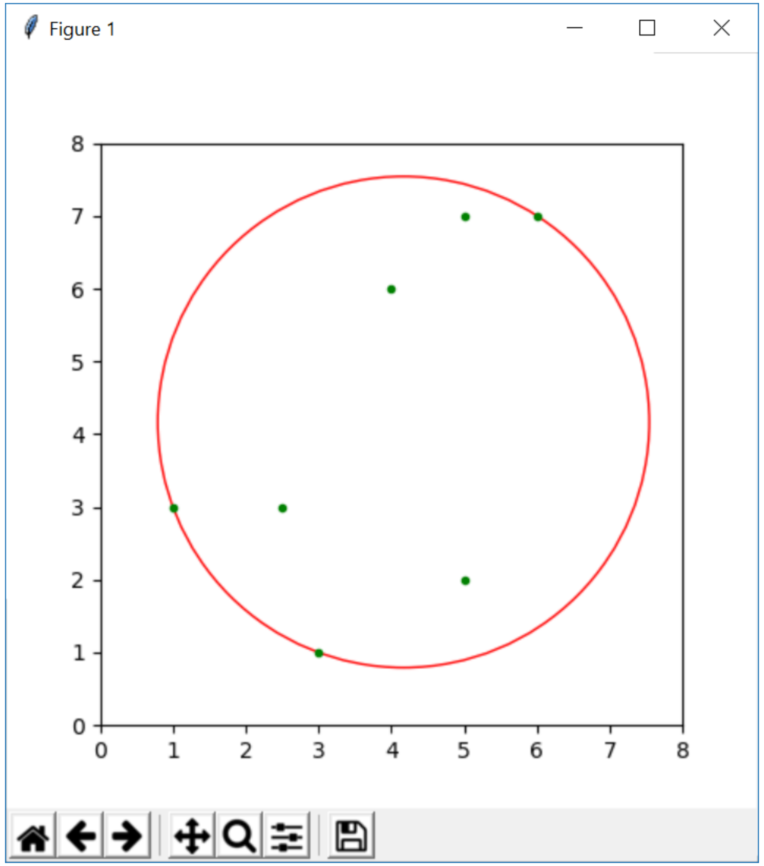


Minimum enclosing circle in Python

enclosing_circle.py

```
from math import sqrt
from scipy.optimize import minimize
import matplotlib.pyplot as plt

x = [1.0, 3.0, 2.5, 4.0, 5.0, 6.0, 5.0]
y = [3.0, 1.0, 3.0, 6.0, 7.0, 7.0, 2.0]

def dist(p, q):
    return sqrt((p[0]-q[0])**2 + (p[1]-q[1])**2)
def max_distance(c):
    return max([dist(p, c) for p in zip(x, y)])

c = minimize(max_distance, [0.0, 0.0], \
             method="nelder-mead").x
ax = plt.gca()
ax.set_xlim((0, 8))
ax.set_ylim((0, 8))
ax.set_aspect("equal")
plt.plot(x, y, "g.")
ax.add_artist(plt.Circle(c, max_distance(c), \
                         color="r", fill=False))
plt.show()
```

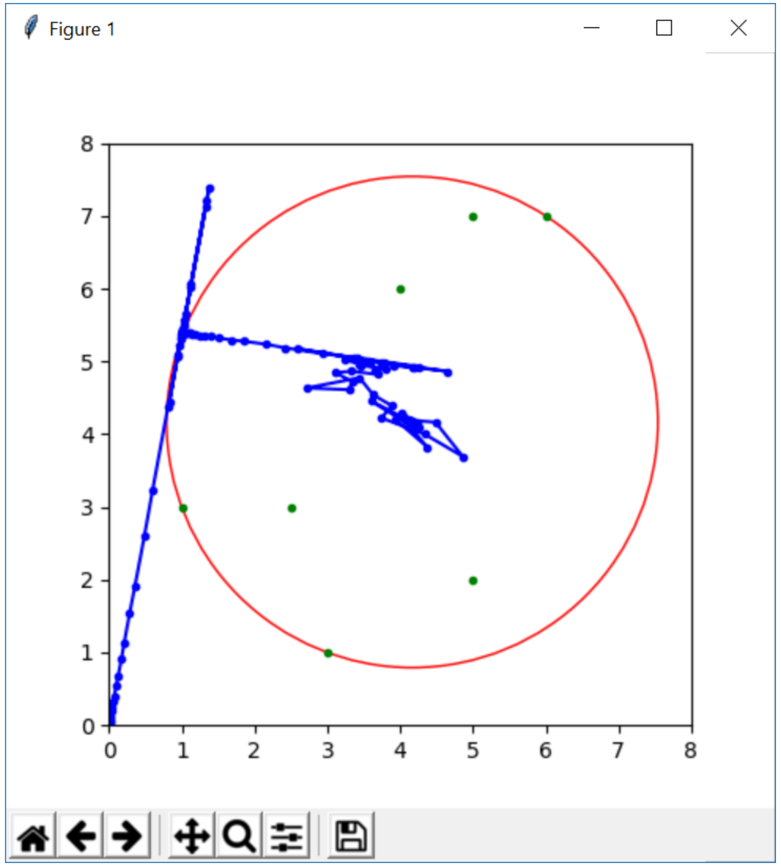
- import modules (first three lines)
- force optimization method (method="nelder-mead")
- manually set axis (force circle inside plot) (set_xlim, set_ylim, set_aspect)



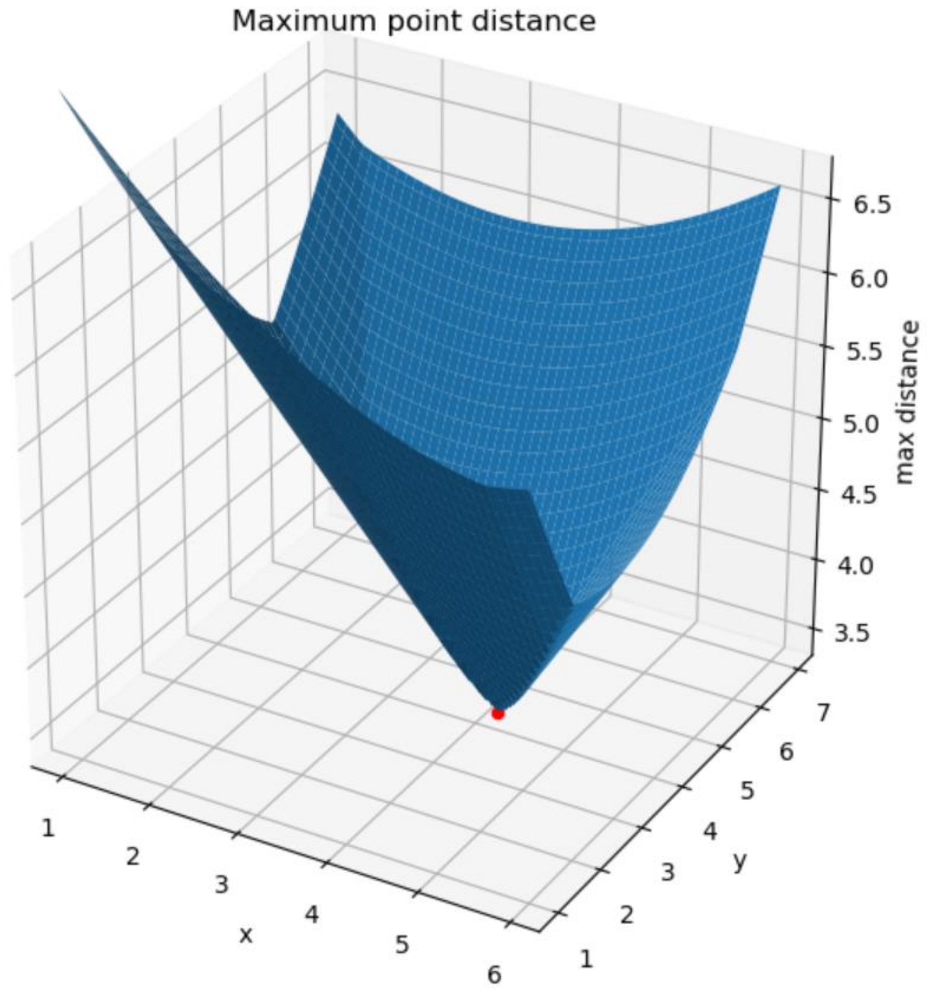
Minimum enclosing circle in Python (trace)

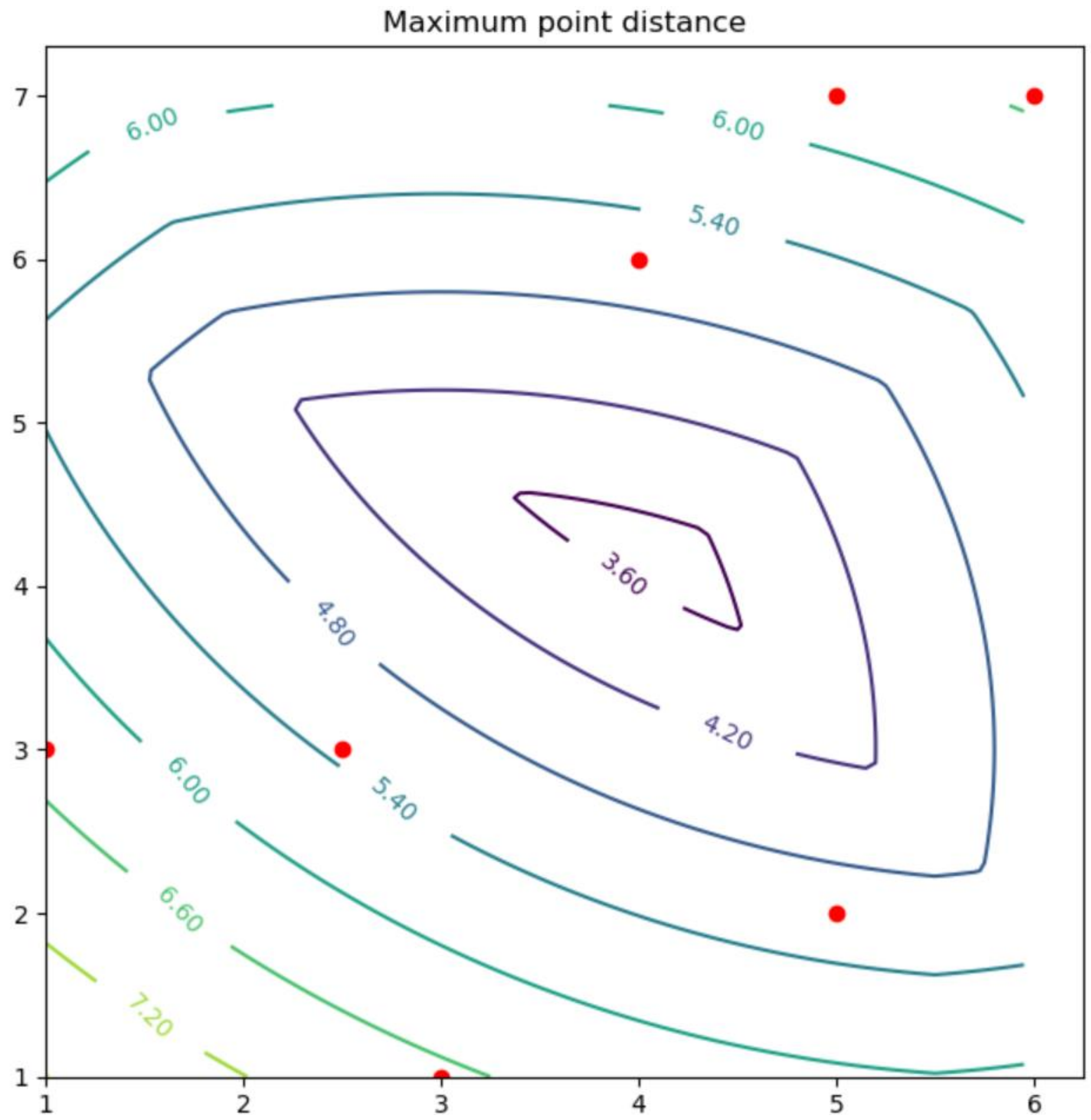
enclosing_circle_trace.py

```
from math import sqrt
from scipy.optimize import minimize
import matplotlib.pyplot as plt

x = [1.0, 3.0, 2.5, 4.0, 5.0, 6.0, 5.0]
y = [3.0, 1.0, 3.0, 6.0, 7.0, 7.0, 2.0]
trace = []

def dist(p, q):
    return sqrt((p[0]-q[0])**2 + (p[1]-q[1])**2)
def max_distance(c):
    trace.append(c)
    return max([dist(p, c) for p in zip(x, y)])

c = minimize(max_distance, [0.0, 0.0],
             method="nelder-mead").x
ax = plt.gca()
ax.set_xlim((0, 8))
ax.set_ylim((0, 8))
ax.set_aspect("equal")
plt.plot(x, y, "g.")
plt.plot(*zip(*trace), "b.-")
ax.add_artist(plt.Circle(c, max_distance(c),
                         color="r", fill=False))
plt.show()
```

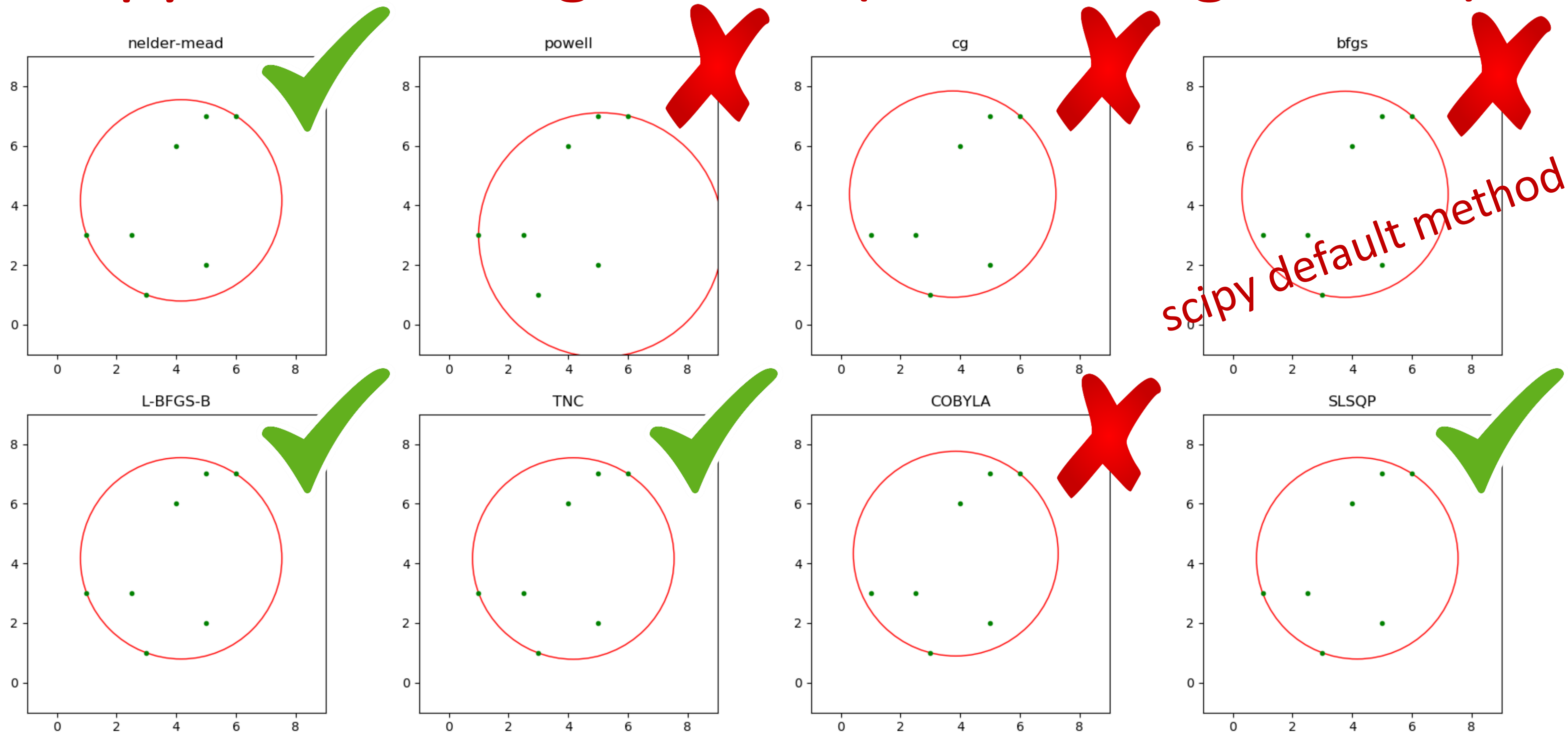


Minimum enclosing circle – search space





scipy.minimize algorithms (without arguments)



https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html