Lecture 3

- **Python3 – selected concepts and grammar in practice**

  - Iteration, Formatted Printing, Timing

  - All the codes discussed today are here:
  - http://planets.utsc.utoronto.ca/~pawel/pyth
  - run them on your computer; read the code and see what every part of it does. Also, the above page contains detailed comments on codes. This is the best way to become proficient in Python.

    - Broken physical pendulum range

  - Plotting:

- MacLaurin (Taylor) series for functions $y = e^x$ & $y = cos(x)$
- Kepler problem

PYTHON ver.3 - *Nobody mastered it w/o self-study & PRACTICE ON THEIR COMPUTER*

Joshua Izaac & Jingbo Wang, "Computational Quant. Mech."
(Undergrad. Lecture Notes in Physics, Springer, 2018)

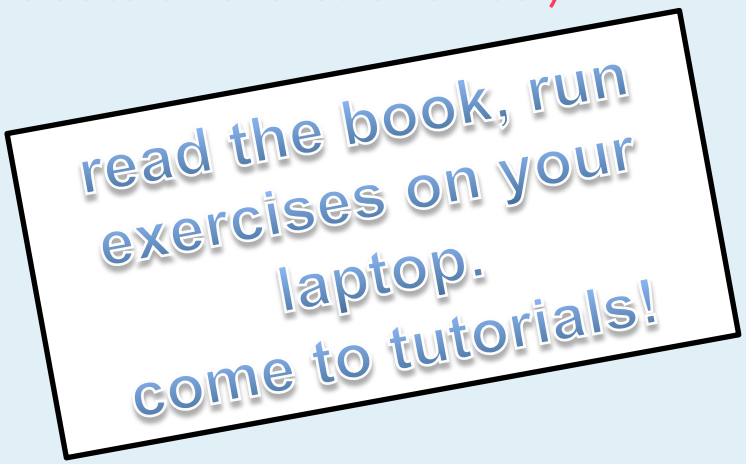
chapter 1. Numbers and precision

1.1-1.3. Fixed point (*not important*) & Floating point (*very important*)

chapter 3. Python

3.1 Indentation, case, comment
3.2 Variables and their precision
3.3 Operator precedence, conditional statements
3.4 Strings (*least important of this list*)
3.5 Data structures (*skip subsections about sets and dictionaries*!)
3.6 Loops
3.7 I/O
3.8 Functions
3.9 NumPy and arrays
3.10 Function arguments
3.11 Timing the code

-- Matplotlib – Python graphics
-- Differences between Python2 & Python3 (cf. link on course page)



All codes are available from the page
http://planets.utsc.utoronto.ca/~pawel/pyth

○ Iteration, Formatted Printing, Timing

Some people among N persons have the same birthday:
among how many people is p of that > 0.5?

birthday2-1.py overlap-birthday.py

Observational confusion in unresolved images

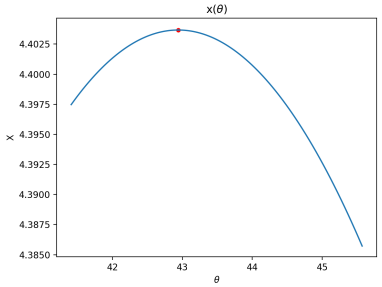
unres-observ-1.py (conditionals, break out from loop)

Simple vectors (1D arrays of numbers)

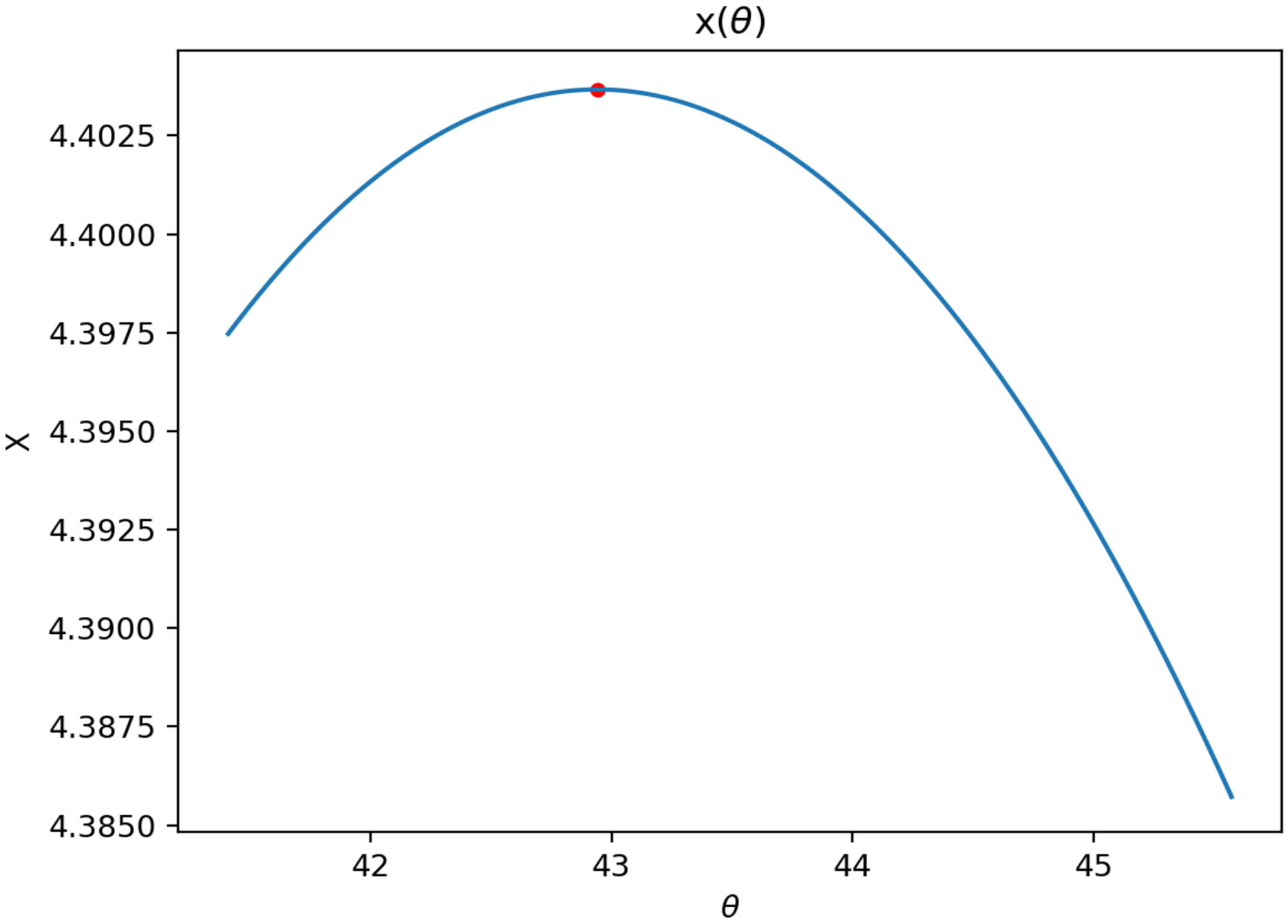
simple_vectors-01.py simple_vectors-01b.py
simple_vectors-01c.py simple_vectors-02.py simple_vectors-03.py

Broken physical pendulum - range

carousel-range-inter.py
carousel-range-0f.py
carousel-range+plot.py



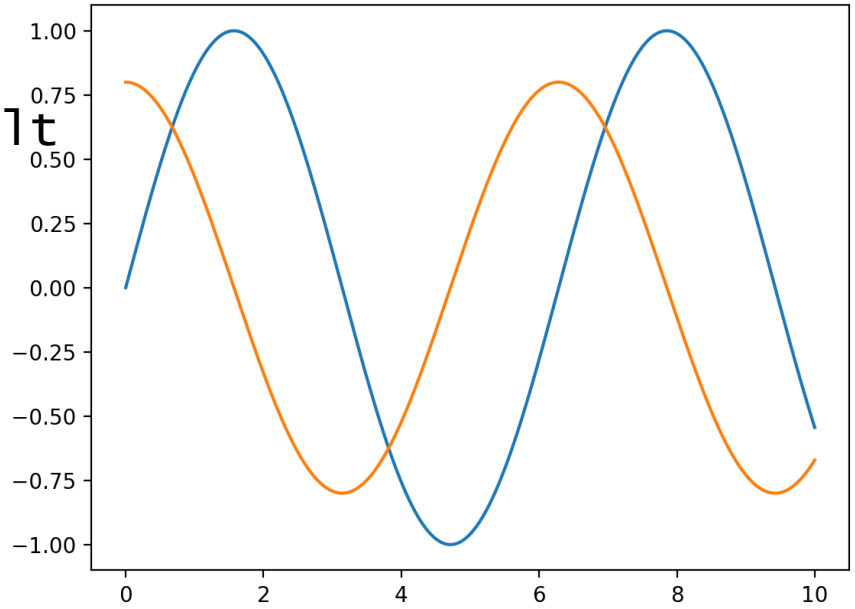
Broken physical pendulum – max. range 4.403 R achieved at 42.9 degrees angle of deflection – run carousel-range+plot.py



Plotting

Simple *interactive* plot of sin(x) function – please try to type or CTR+C/CTRL-V this on your computer after staring Python in a command window:

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.linspace(0.,10.,500)
>>> plt.plot(x, np.sin(x))
>>> plt.plot(x, 0.8*np.cos(x))
>>> plt.show()
```



As you see, Python can automate a lot of chores. On the other hand you need to *read the manuals* for how its functions work (what input they allow, what options can you use) – then you will be able to modify the default behavior of MatPlotLib. Go ahead and experiment.

Remember that in every job, an expert is someone who has done all the possible mistakes. The same holds for programming, which is a very powerful tool, but requires reading instructions and practice. And be patient, your skills will grow with time.

Plotting

- Maclaurin (Taylor around zero) series for functions $y=e^x$ & y=cos(x)

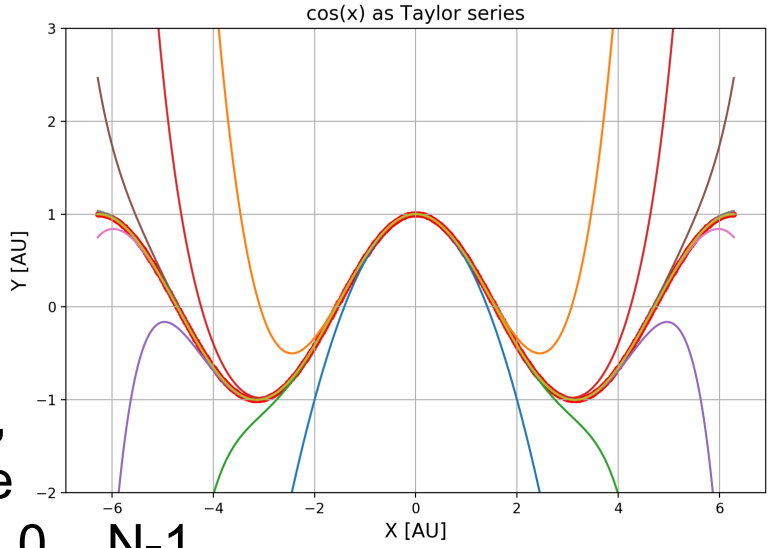
plot_cos_approx-1.py plot_exp_approx-1.py

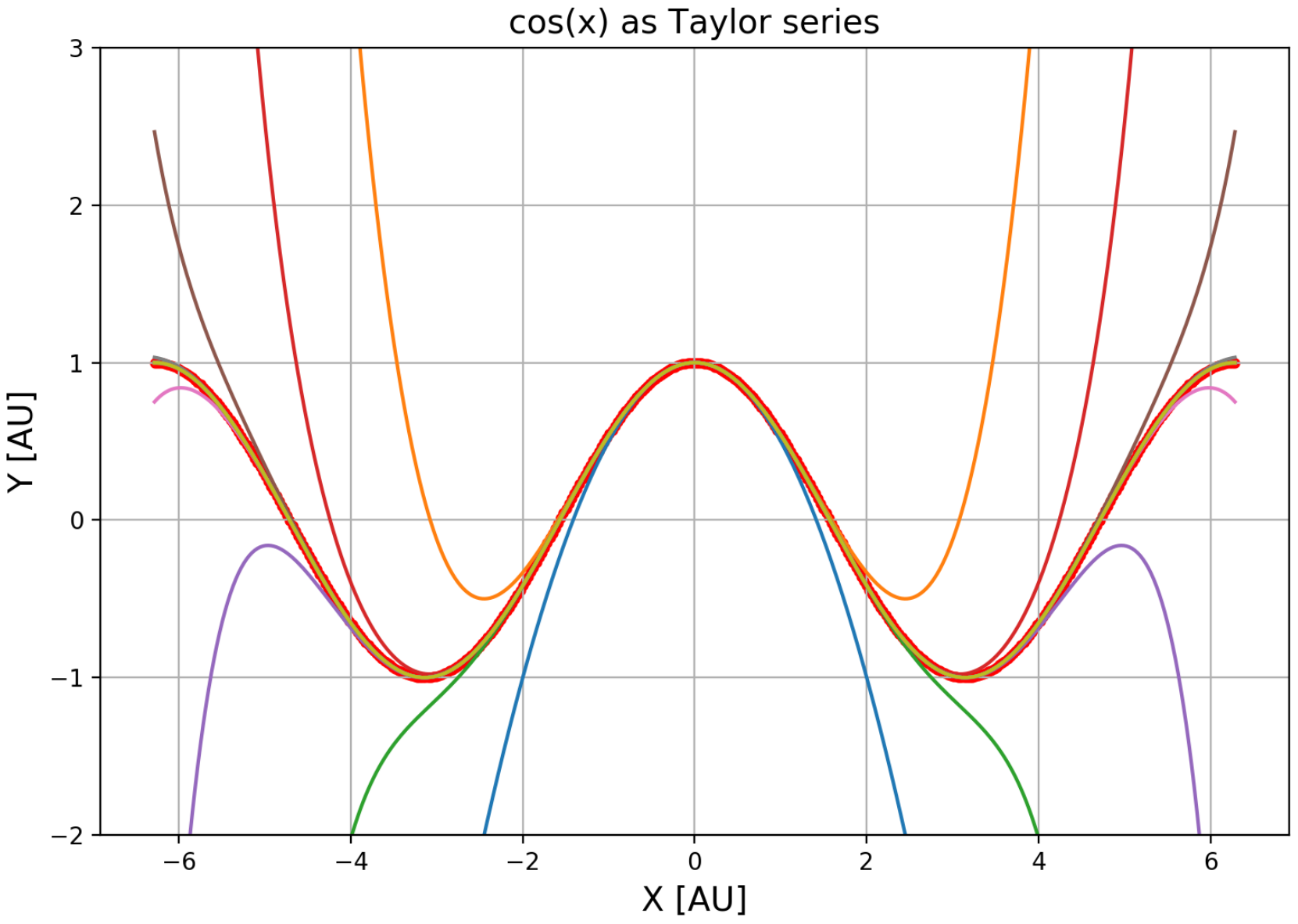
$e^x = \exp(x) = \Sigma_{k=0}^{\infty} x^k/k! = 1 + x + x^2/2 + x^3/6 + ...$

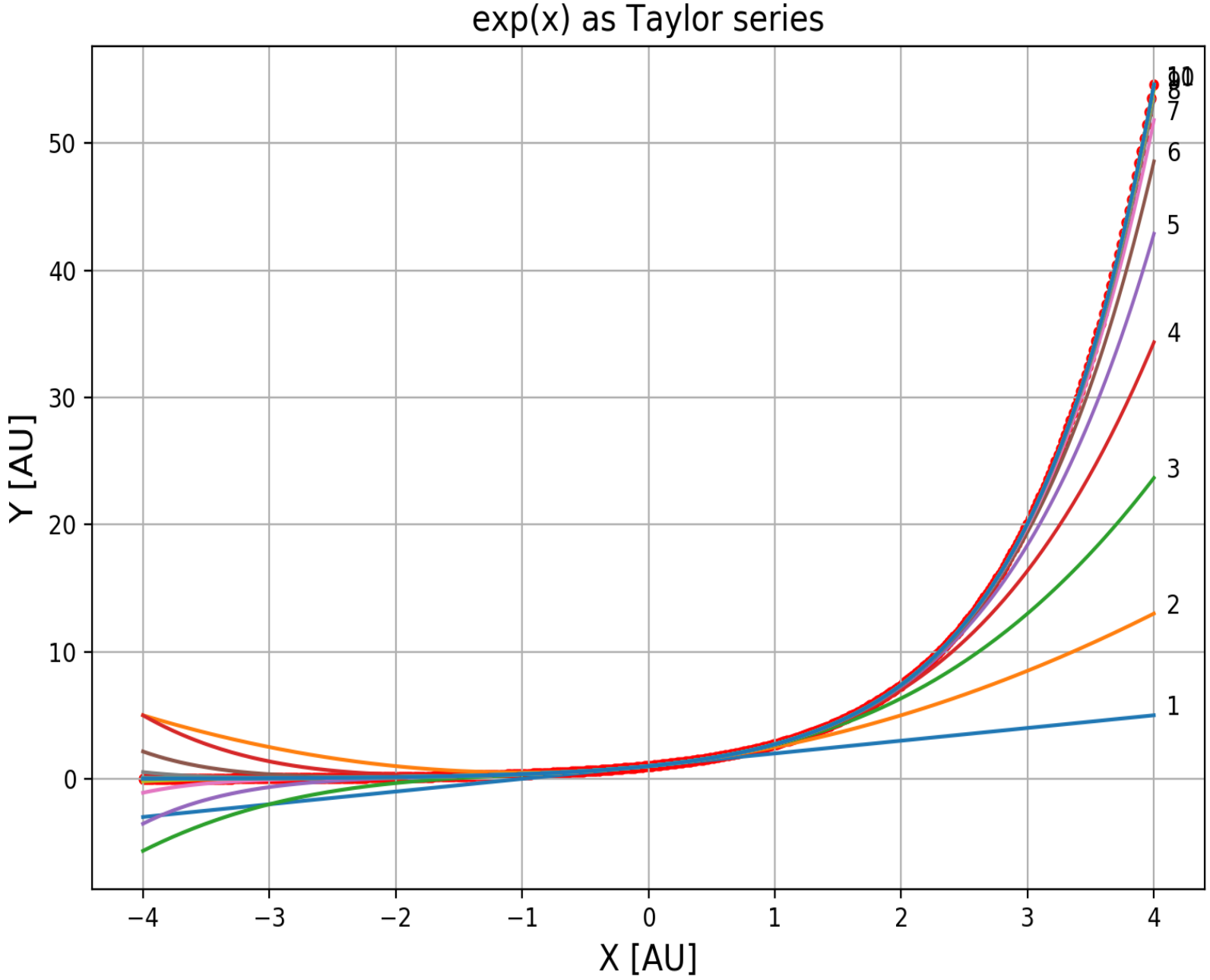
$\cos(x) = \Sigma_{k=0}^{\infty} (-1)^k x^{2k}/(2k)! = 1 - x^2/2 + x^4/24 + ...$

where the factorial is $n! = n\,(n-1)! = 1*2*3....(n-1)\,n$. This first equality is used in the loop over k in order not to start the computation of (2k)! from scratch in every loop passage. Instead we use (n-1)! already computed (n=2k).

[The scripts use the NumPy arrays, on which mathematical operations are performed in so-called vectorized form, much faster than scanning the array with a Python *for* loop & performing operations element after element. E.g., if X,Y,Z are arrays of the same length N, you can do Y = Y + Z**2*sin(X) and all those operations will be done on elements [0:N] i.e. 0....N-1.



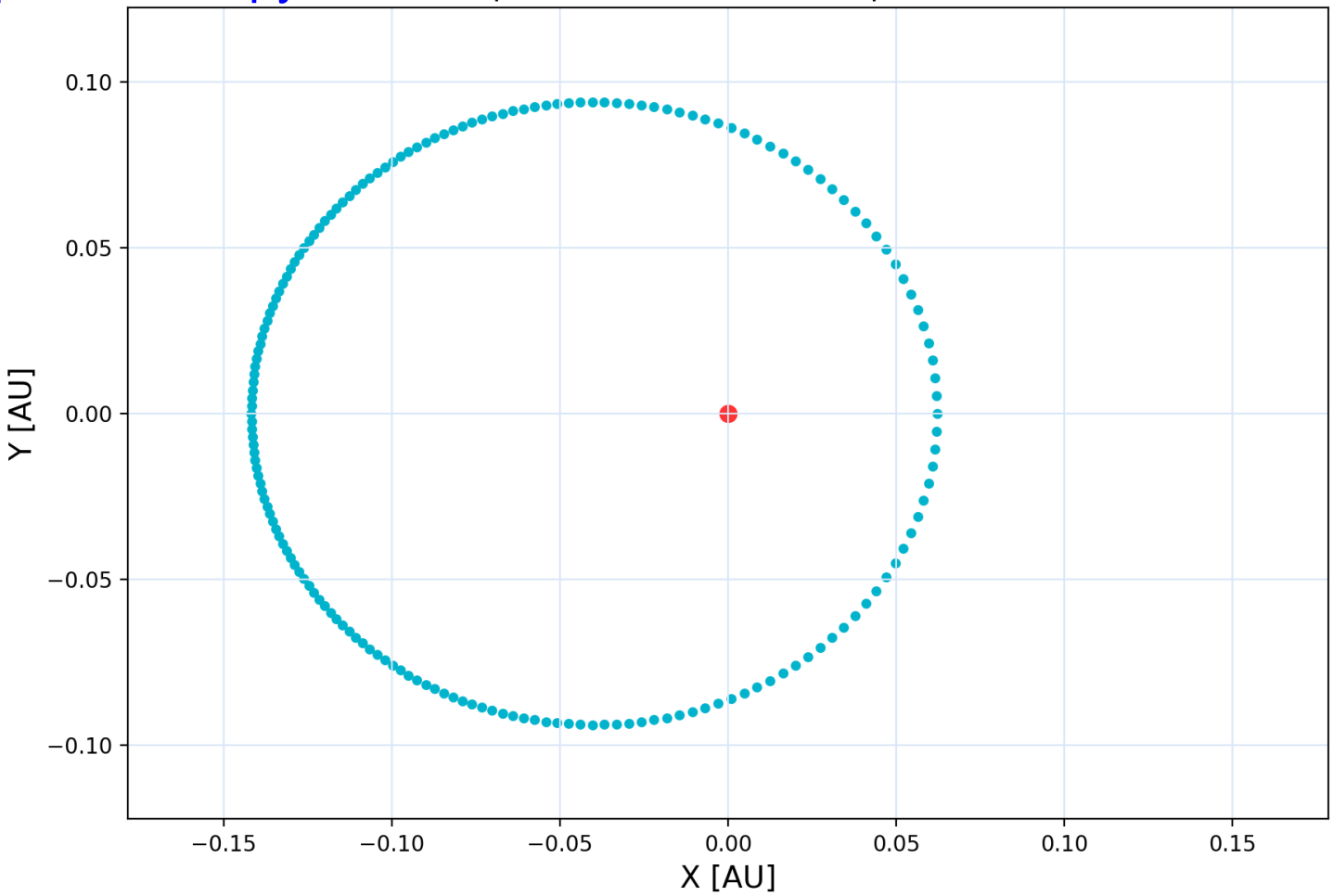

cos(x) as Taylor series
Y [AU]
X [AU]
3
2
1
0
−1
−2
−6
−4
−2
0
2
4
6


exp(x) as Taylor series
Y [AU]
X [AU]
50
40
30
20
10
0
−4
−3
−2
−1
0
1
2
3
4
7
6
5
4
3
2
1

Plotting

- Scientific discovery from 2018: an Saturn-like extrasolar planet on a small, eccentric orbit. This program makes its uneven motion easy to understand, by covering one full period of motion with equal time intervals (big dot spacing = large speed near the star).

kepl-K2-261b.py Orbit of exoplanet K2-261b at 180 equal time intervals



Other PYTHONable problems of interest

- Random walks: visualization, theory
- How likely that a random walk returns to starting point in 2N steps?
- If a stick is broken at a random point, what's the mean length of the shorter piece & the mean ratio of lengths of two pieces?
- Should I bet that someone draws his/her own ID at random from a hat?
- How to compute oscillation modes of a mechanical system?
- Given a vertical profile of a bike trip, compute total rise & total descent?
- What's the probability that two random points on a unit interval are at distance less than one-half?
- How to fit a polynomial to data? How to fit multi-parameter functions?
- How to compute trajectories of chase, and those in Newton's dynamics?
- Why and how to interpolate by splines?
- Can one compute and animate waves on the surface of water?
- How to simulate realistic airfoil's lift force by attached vortex method?
- (...)