Lecture 4

◆ Python3 practice in Cosmos & Random Worlds

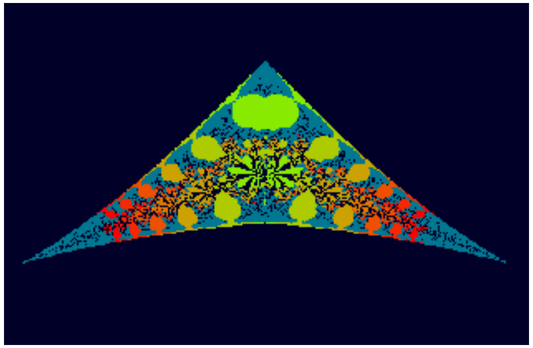
All the scripts discussed in lectures are downloadable from http://planets.utsc.utoronto.ca/~pawel/pyth . Please run & analyze them!

Simple file I/O (input/output) in Python

Computing simple statistics: mean and std deviation of data

○ **Computing and Plotting**

- Kepler problem (convergence illustrated)
- Irradiation of exoplanets K2-261b and Gj 3512b, use of simple numerical integration
- Iterated exponentiation in complex plane
- Exponential fractal



○ **Pseudorandom numbers and histograms**

- 3 kinds of randomness: nature, math, computers
- Monte Carlo casino as a randomness generator
- Generation of uniform and normal random numbers

Simple file I/O - output

Output float array in a 2-column format, cf. simple_IO.py

```
# (...)
x = np.linspace(0,10,100)
y = np.sin(x)-(x/10)**3 +3*(x/10)**4 # some math function
tab = zeros((2,100), dtype=float)

tab[0,:] = x  # will be column 1 in the file, now it's row 1
tab[1,:] = y  # will be column 2 in the file, now it's row 2

# store data in file with space-separated values (e.g., .dat)
# transpose to turn rows into columns & store
# store data in file with space-separated values (.csv)

savetxt("simple_io.dat", tab.T)

savetxt("simple_io.csv", tab.T, delimiter=",")
```

Simple file I/O - input

Input from a 2-column file a float array, cf. simple_IO.py

```
# .T = transpose, we use it to turn 2 columns back into 2 rows
# delimiters in reading function must match those in data
# .csv  - comma-separated values
# .dat  - could be space-separated values, or anything really

# read data from file of space-separated values
xx,yy = loadtxt("simple_io.dat",delimiter=" ").T
# or like this (space delimiter is a defult)
xx,yy = loadtxt("simple_io.dat").T

# this will work:
xx,yy = loadtxt("simple_io.csv",delimiter=",").T
# but this will not (because the default delimiter is space)
zz,ww = loadtxt("simple_io.csv").T
# this will fail as well (wrong delimiter for our .dat file)
aa,bb = loadtxt("simple_io.dat",delimiter=",").T
```

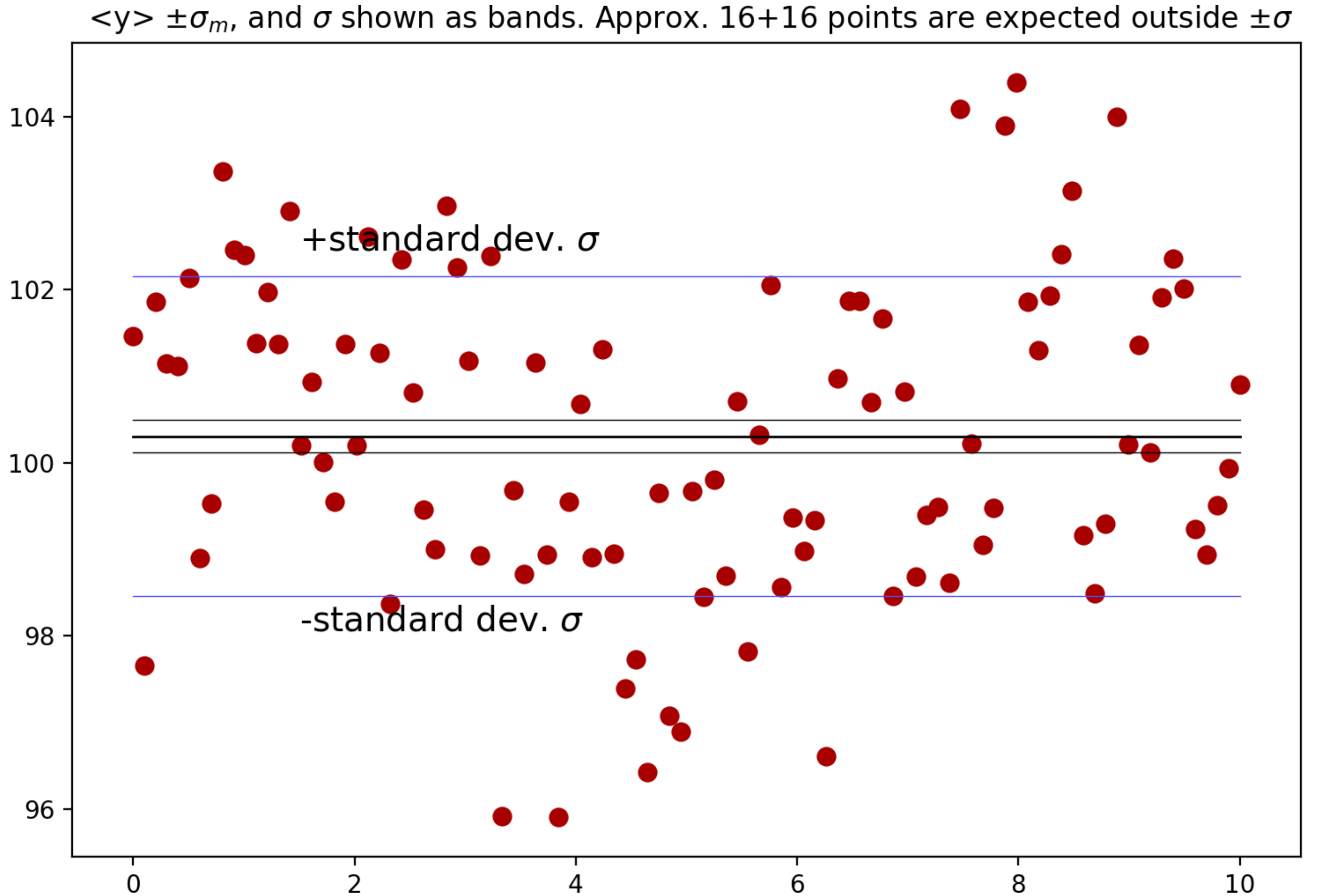
Simple statistics

Data read and analyzed in simple_stat.py

```
# Three approaches are used in the code, two hand-coded
# and this method from Numpy (yy is a NumPy array)
y_ave = yy.mean();  stdev=yy.std();  sigm_m = stdev/N**0.5
```

Standard deviations of data and of their mean are different!

Here, $\sigma \sim 1.8$, but $\langle y \rangle = 100.24 \pm 1.8/\sqrt{100} = 100.24 \pm 0.18$.



○ Plotting

Scientific discovery from 2018: extrasolar planet on eccentric orbit

kepl-K2-261b-0.py kepl-K2-261b.py

Core part is Kepler Equation solved by iteration

*M = {an angle from 0 to 2*pi, uniformly changing with time}*

Kepler equation

E - e sin(E) = 2 pi t/P = M

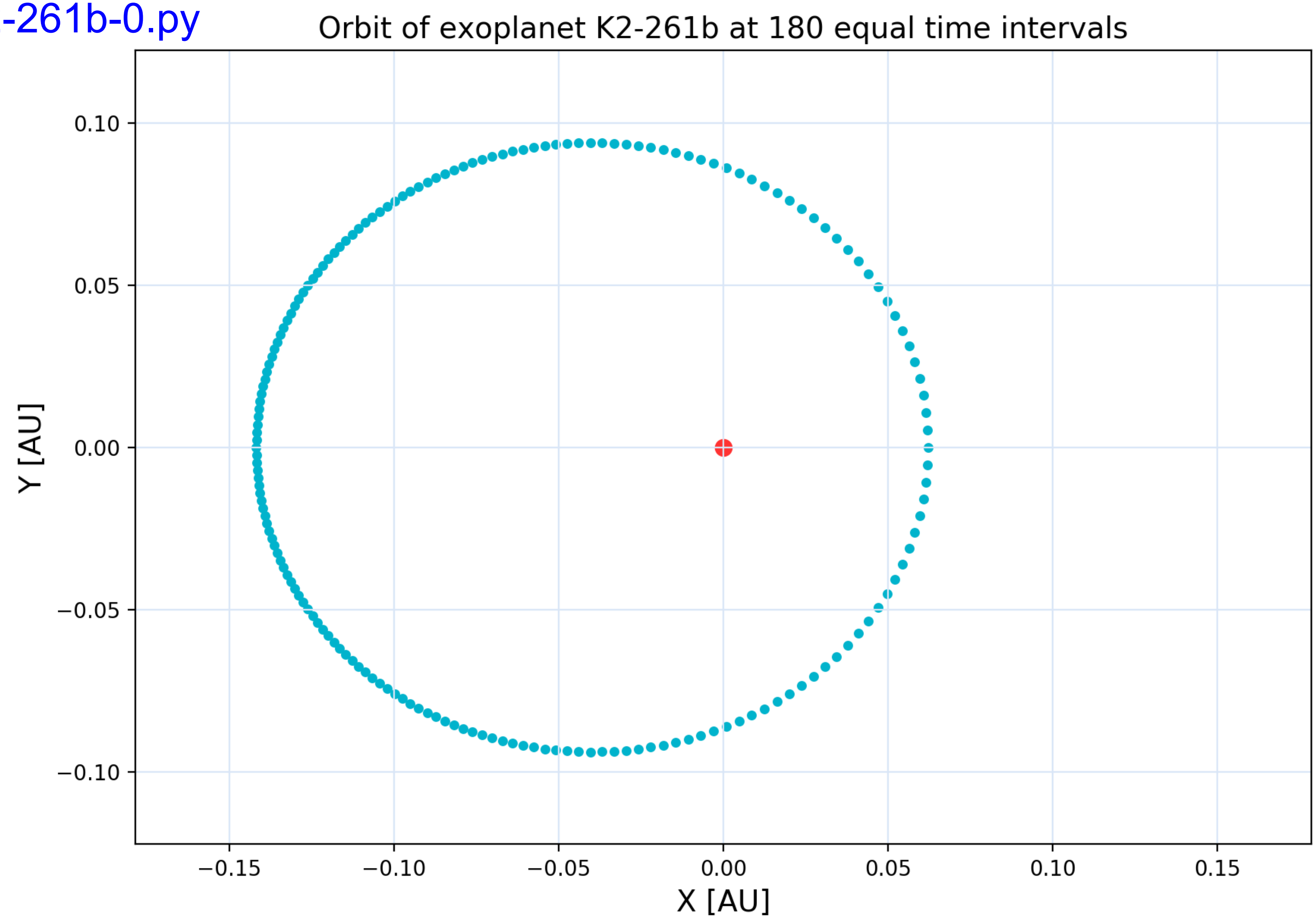
cannot be solved on paper, but can be solved iteratively

```
(...)
E = M                     # first guess to start with
for k in range(40):       # counter of iterations
    E_previous = E        # save previous E for comparison
    E = M + e*sin(E)      # Kepler equation iteration
    if (i%10==0):
        print('i',i,' k',k," E =",E)   # print every 10th
    if (abs(E-E_previous) < 1e-9):     # sufficient accuracy
        break
```

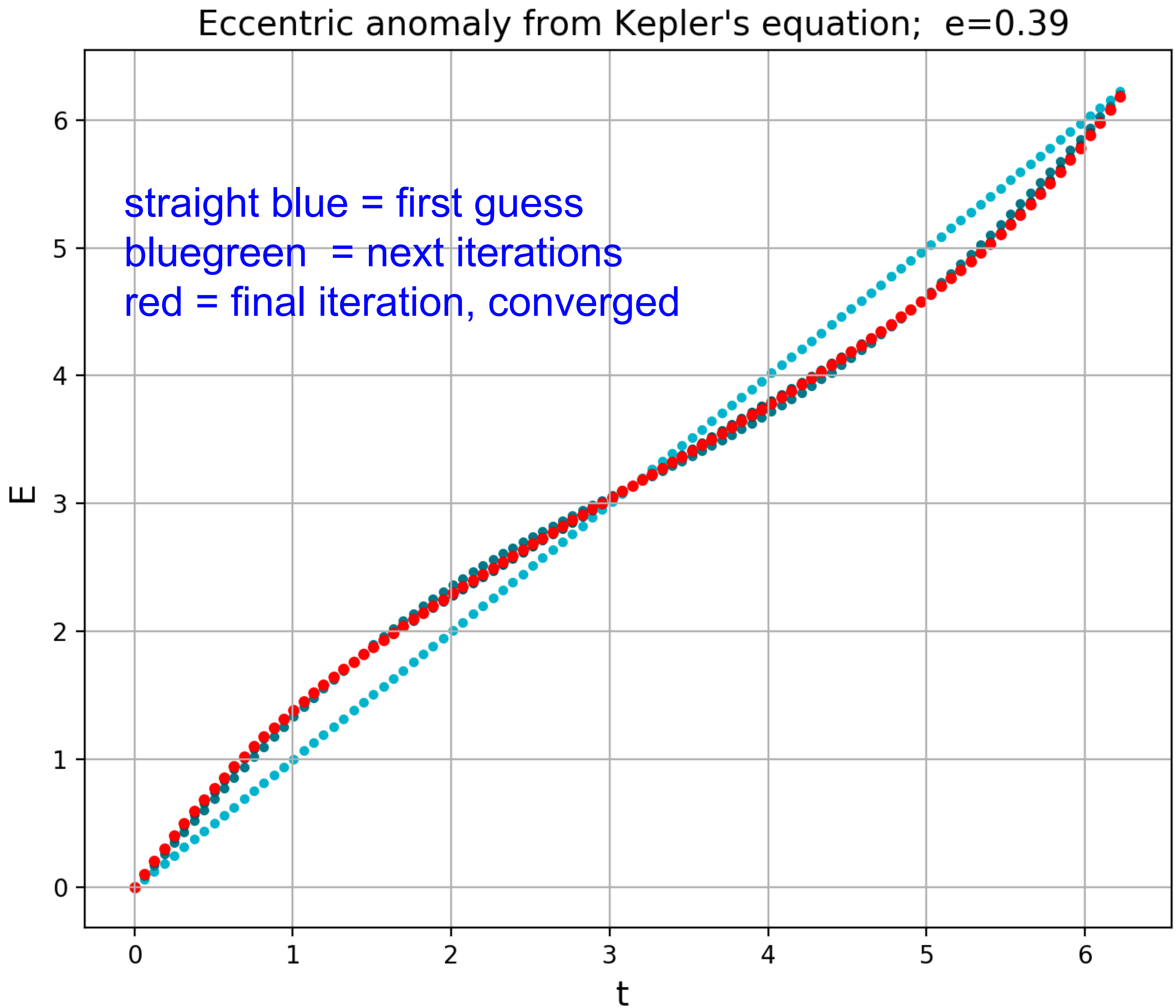
Scientific discovery from 2018: a Saturn-like extrasolar planet on a small, eccentric orbit. This program makes its uneven motion easy to understand, by covering one full period of motion with equal time intervals (big dot spacing = large speed near the star).

See the equations outlined in the program comment section:

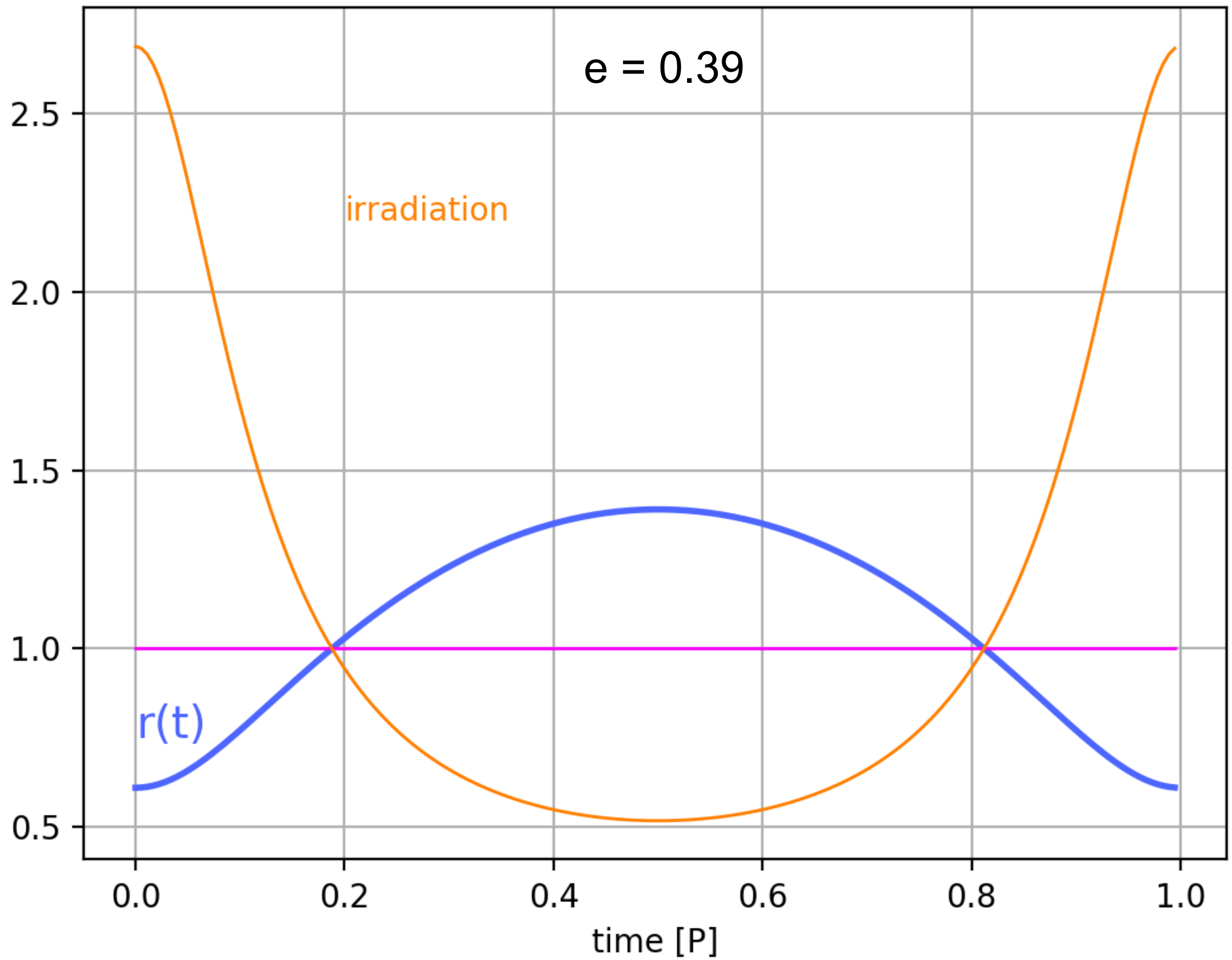
kepl-K2-261b-0.py



Convergence of Kepler Equation kepl-K2-261b-0.py



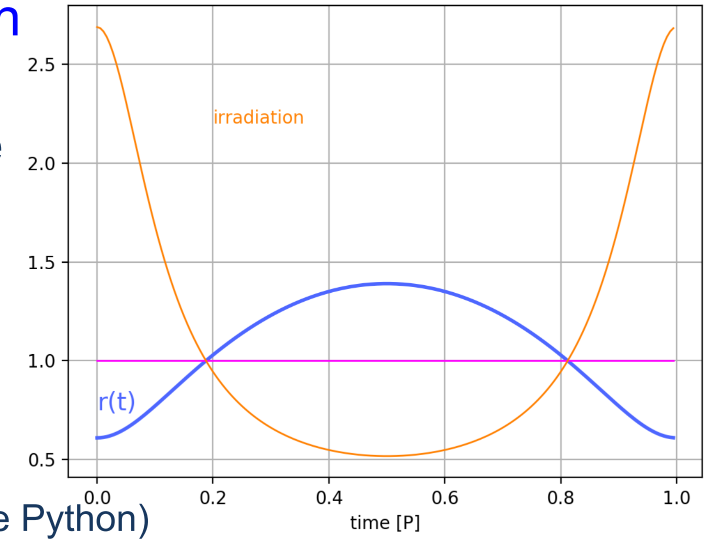
- kepl-K2-261b.py **Simple numerical integration (summation)**
- program prints: average irradiation = 1.0859946...
- The changing distance and speed partially cancel each other's influence
- on the irradiation of a planet as a whole. Over the whole period,
- planet receives ~8.6% more energy than if it were on a circular orbit.



✧ kepl-K2-261b.py output: e = 0.39, irradiation factor = 1.0859946415 6

The summation of area under the irradiation curve in this program is an example of the simplest method of integration of functions.

Planet on eccentric orbit receives 8.6% more radiation energy than on a circular orbit.



➢ Let us verify these numerical results analytically, i.e. without computer (without much help from it anyway. We'll need a calculator inside Python)

$I(\theta) = I_0\,(a/r)^2$ is an inverse r^2 scaling of stellar radiation flux; a = semi-diam.

From angular momentum conservation in orbital mechanics:

$L = v_\theta\, r = d\theta/dt\;\; r^2$ = const. specific angular momentum = $(1\text{-}e^2)^{1/2}\, 2\pi\, a^2/P$

$\Rightarrow \qquad dt = (1\text{-}e^2)^{-1/2}\;\; r^2/a^2\;\; d\theta/(2\pi)$.

Hence, when we average $I(\theta)$ over time, we do an integral over dt that converts into a trivial integral over $d\theta$ and gives a simple final result:

$$\langle I\rangle / I_0 \;=\; P^{-1}\;\{\, \text{integral}_0^{2\pi}\;\; I(\theta)/I_0\;\; dt \,\} \;\;=\; \ldots \;=\; (1\text{-}e^2)^{-1/2}\;.$$

Check: if e = 0.39, then $\langle I\rangle$=1.0859946414 9I_0, => surprising accuracy above!

News flash

https://www.space.com/gas-giant-alien-planet-red-dwarf.html

Planet catalog (1000s of planets): exoplanet.eu/catalog/gj_3512_b

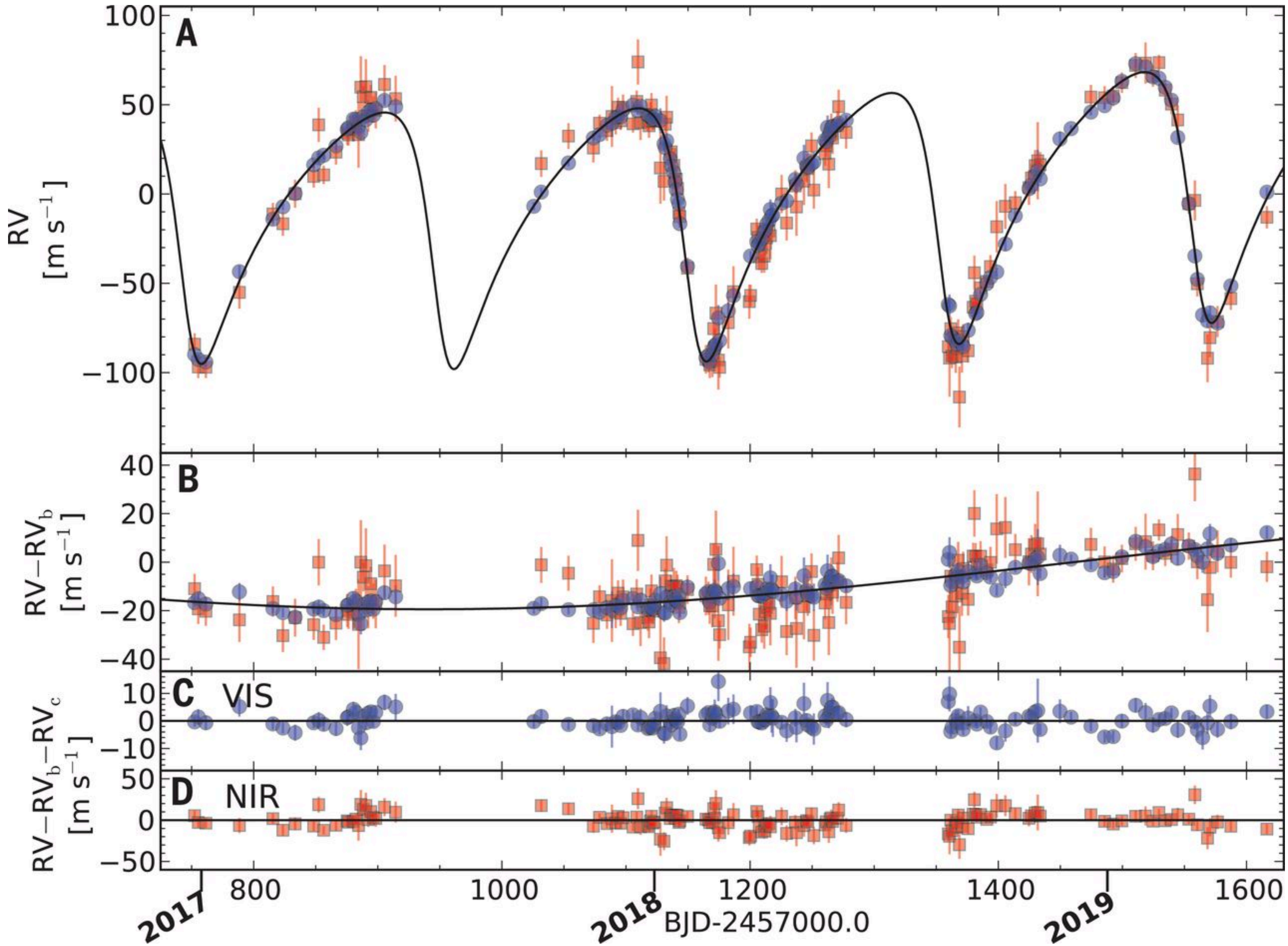
Planet somewhat more massive than Saturn, ~0.45 Jupiter masses has been announced 27 Sept. 2019. Eccentricity of orbit = 0.4356.

Circles every 203 days around a very small M5 red dwarf star GJ 3512 b located 10 pc from Earth. The star has diameter 0.14 Sun's radius, and mass only 0.12 solar masses.

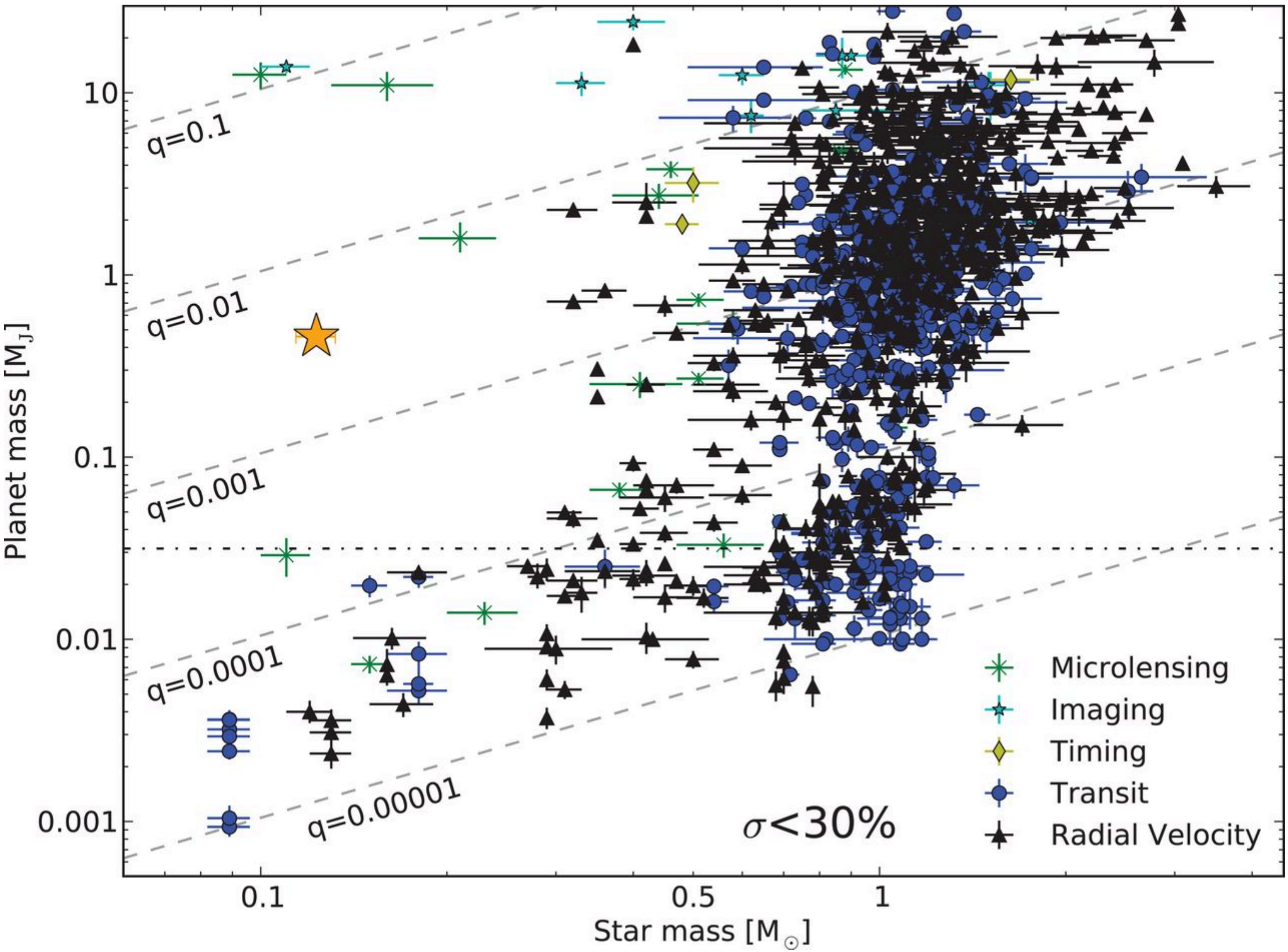
News articles claim this planet puts a question mark over planet formation theories, because there is lillte material in protoplanetary disks of small stars. (Not a very strong argument!)

Our task: Find it's mean surface temperature (under certain assumptions as to energy balance), in particular: find how big a correction one needs to apply because of the large eccentricity of orbit.

Eccentric orbit of GJ 3512b generates this Radial Velocity curve

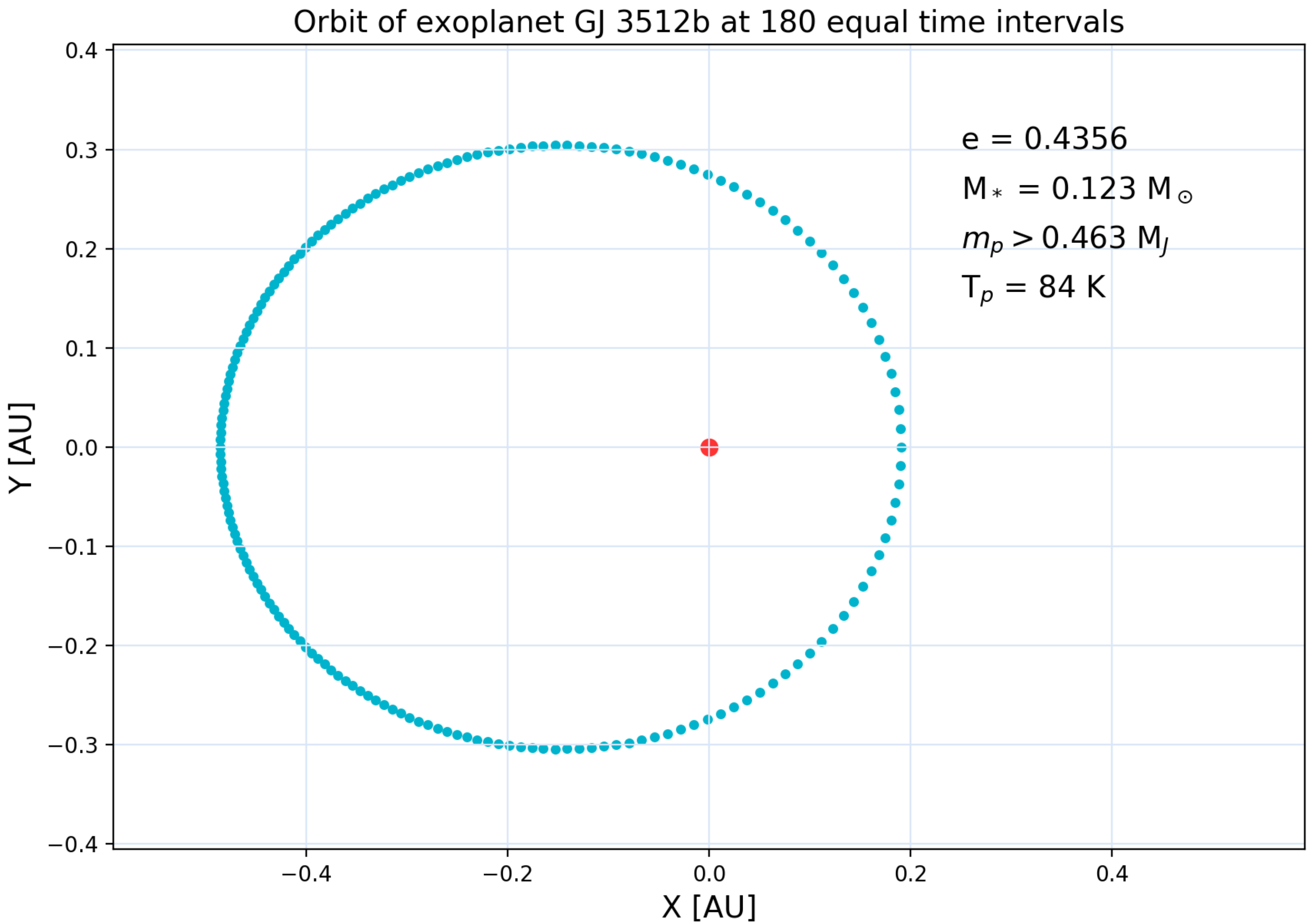


The known exoplanets (2019) on m_p – M_* diagram

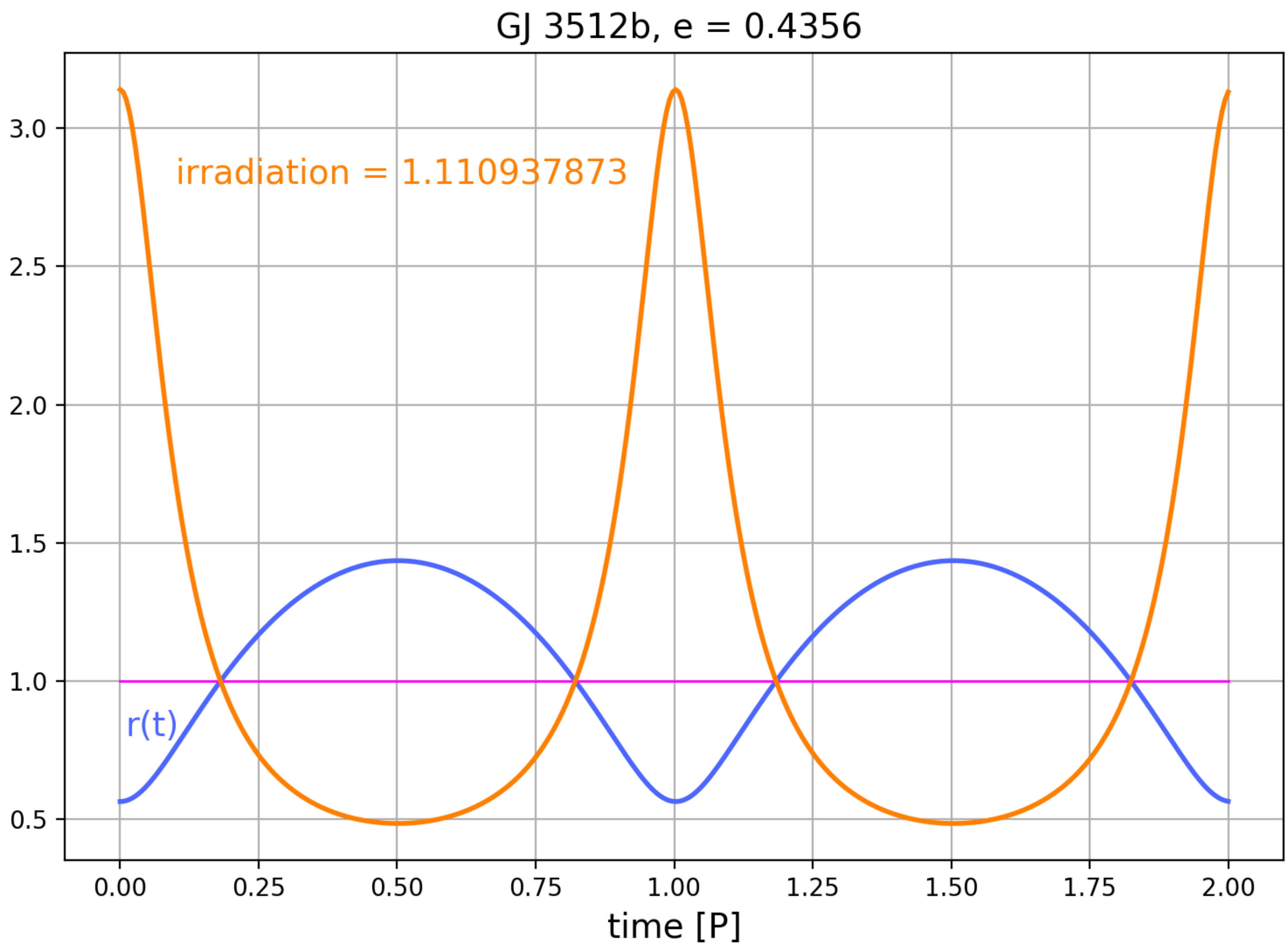


Discovery published on 27 Sept. 2019: a Saturn-class extrasolar planet on an intermediated-sized, eccentric orbit. P = (203.59+-0.14)^{d}.

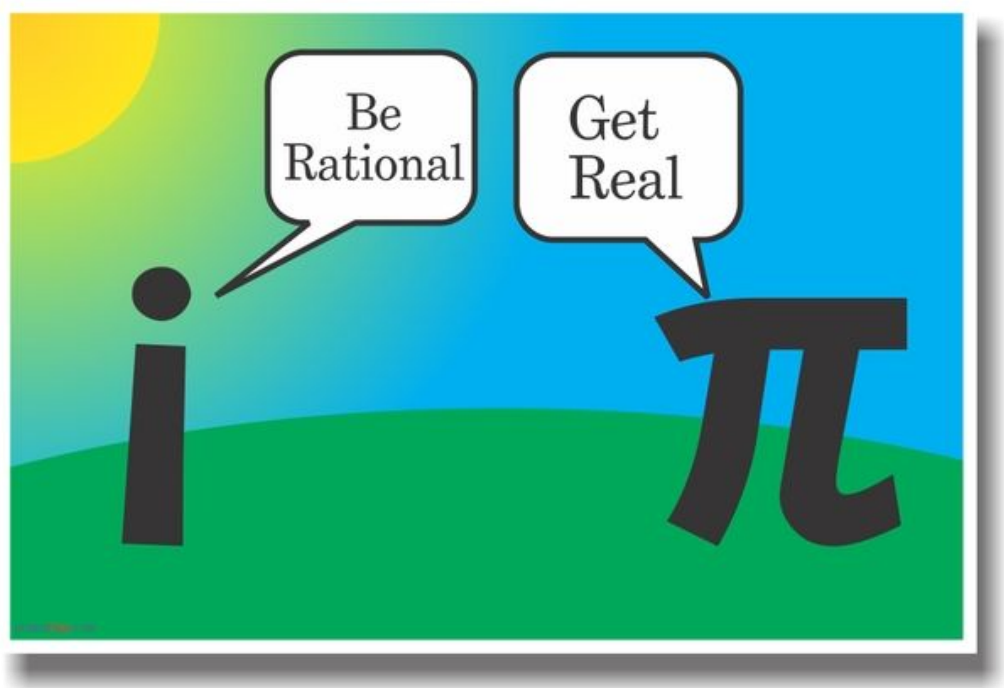
Copy, run and study Python in: kepl-GJ-3512b.py



- kepl-GJ-3512b.py **Simple numerical integration (summation)**
- integrated average irradiation = 1.1109378732817672
- theoretical average irrad. = 1.1109378731712545, difference = 1.1e-10 (!)
- planet receives ~11% more energy than if it were on a circular orbit.



- Complex worlds explored with Python



$i=\sqrt{-1}$ ... $i^{i\wedge i} = ?$

- Complex numbers are part of Python standard language, no need for special modules. Raising numbers to power works for complex numbers too, both as base and exponent. Mathematics: let x be real number,
- $e^x = \Sigma_{n=\{0,1,2,3,4,\ ...,\infty\}}\ x^n/n!$ *(Taylor at x=0; all derivatives of $e^x = e^0 = 1$)*
- $e^{ix} = \Sigma_{n=\{0,1,2,3,\ ...,\infty\}}\ i^n x^n/n!$

  where $i^n = i^{\{0,1,2,3,4,5,...\}} = \{1, i, -1, -i, 1, i, -1, ...\}$
- $\cos x := \Sigma_{n=\{0,2,4,...,\infty\}}\ (-1)^{n/2}\ x^n/n!$ *(= trig. cosine Taylor series)*

  $\sin x := \Sigma_{n=\{1,3,5,...,\ \infty\}}\ (-1)^{(n-1)/2}\ x^n/n!$ *(= trig. sine Taylor series)*
- This proves Euler's identity:

  $e^{ix} = \cos(x) + i \sin(x)$ *e.g., mysterious(?)*

  (unit circle in complex plane, if x=real) $e^{i\pi} + 1 = 0$

Notice one more curious thing. From Taylor expansion we do not immediately see that sin(x) and cos(x) are periodic functions!

- $i = exp(i\ \pi/2)$

$\ldots\ i^{\,i\wedge i} = ?$

- Let's see how i^i works
- $i^i = [\,e^{i\pi/2}]^{\,i} = e^{\,i*i\ \pi/2} = e^{\,-\pi/2} = 0.2078\ldots$
- Indeed, Python denotes i by j and does

```
>>> i = complex(0,1)   or        >>> i = 1j
>>> i**i
(0.20787957635076193+0j)

>>> i**i**i
(0.9471589980723784+0.32076444997930853j)
[Below we check whether Python does the right thing]
>>> i**(i**i)
(0.9471589980723784+0.32076444997930853j)   RIGHT
>>> (i**i)**i
(6.1232339957366e-17-1j)  ~ -j               WRONG
```

Iterated complex exponentiation of i produces lots of complex numbers, does not seem to diverge. But does it really converge, diverge, or oscillate eventually?

```
>>> z = 1
>>> for k in range(10):
...     z = i**z;        print(k,z)
...
0 1j
1 (0.20787957635076193+0j)
2 (0.9471589980723784+0.32076444997930853j)
3 (0.05009223610932118+0.6021165270360038j)
4 (0.387166181086115+0.03052708160548448j)
5 (0.7822756824339533+0.5446065576579896j)
6 (0.14256182316366683+0.4004665253370873j)
7 (0.5197863964078542+0.11838419641581431j)
8 (0.568588617271897+0.6050784067978037j)
9 (0.24236524682521116+0.30115059207131784j)
>>>
```

○ Plotting and exploring

Iterated exponentials
The use of scatter plots, color indices

$$\ldots i^{i^{\wedge} i} = ?$$

iii-00.py (print, how to use j)   iii-0.py (plot, zoom)

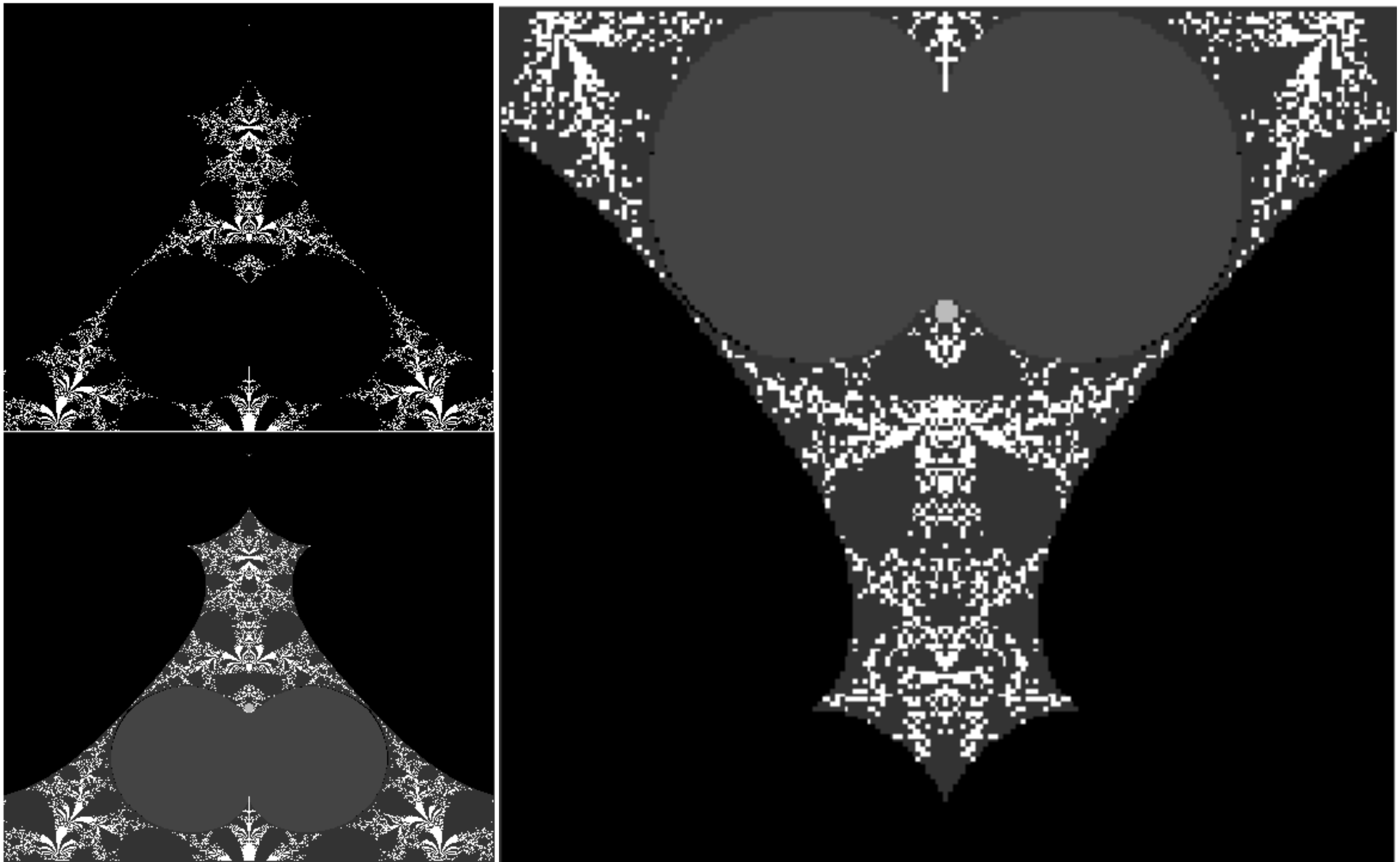
One can start with arbitrary complex number, not only *i*
iii-2.py (many plots, different z)

Exponential fractal. *convergence of* $\ldots z^{z^{\wedge} z}$ *in complex plane*

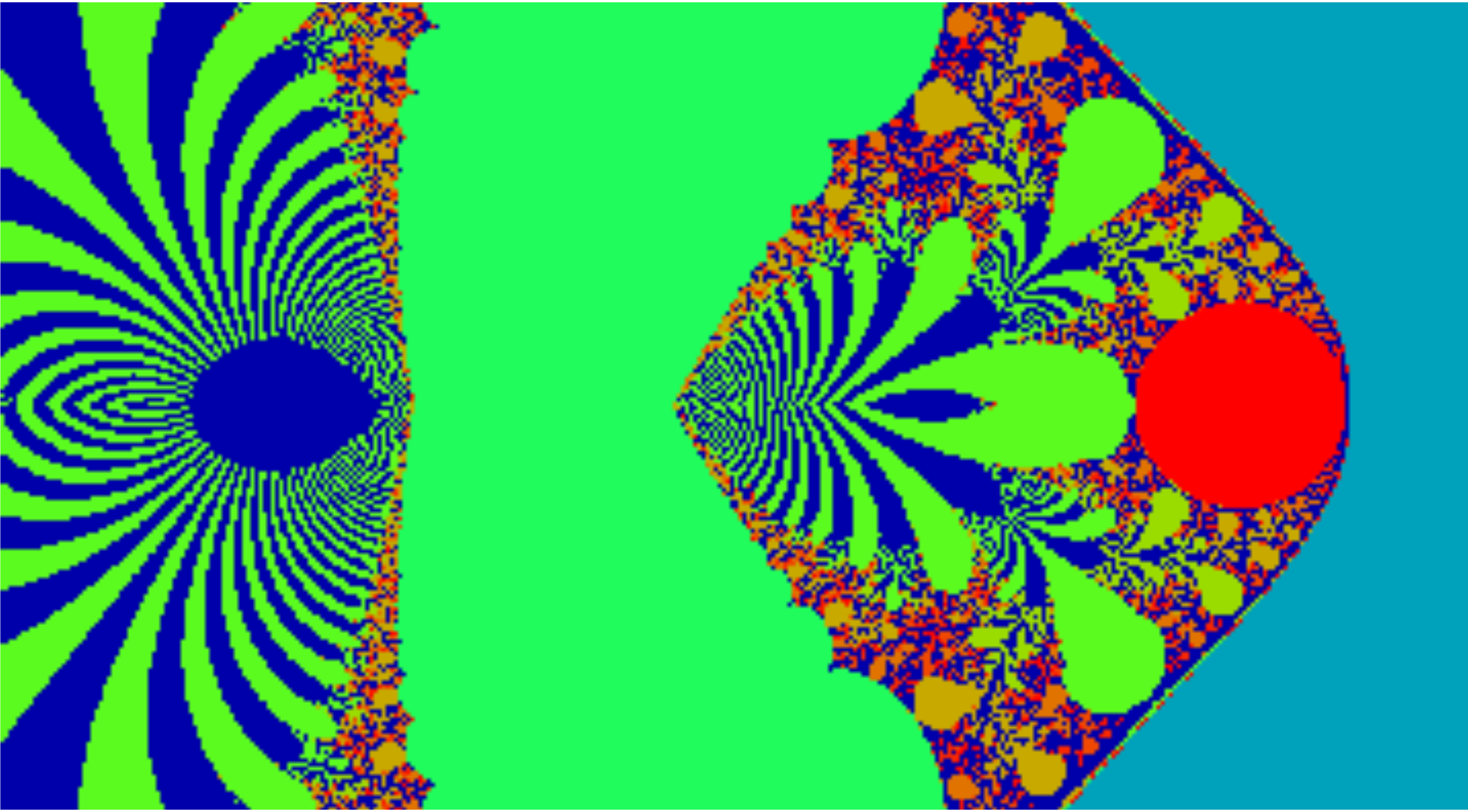
mandel_g4g.py (Mandelbrot fractal)
expfract-s1.py (Artymowicz fractal)
expfract-p1.py (different color map and region)
expfract-p100.py (scale=0.128)
expfract-p100b.py (variable scale)
Challenge: color according to periodicity not divergent iter#

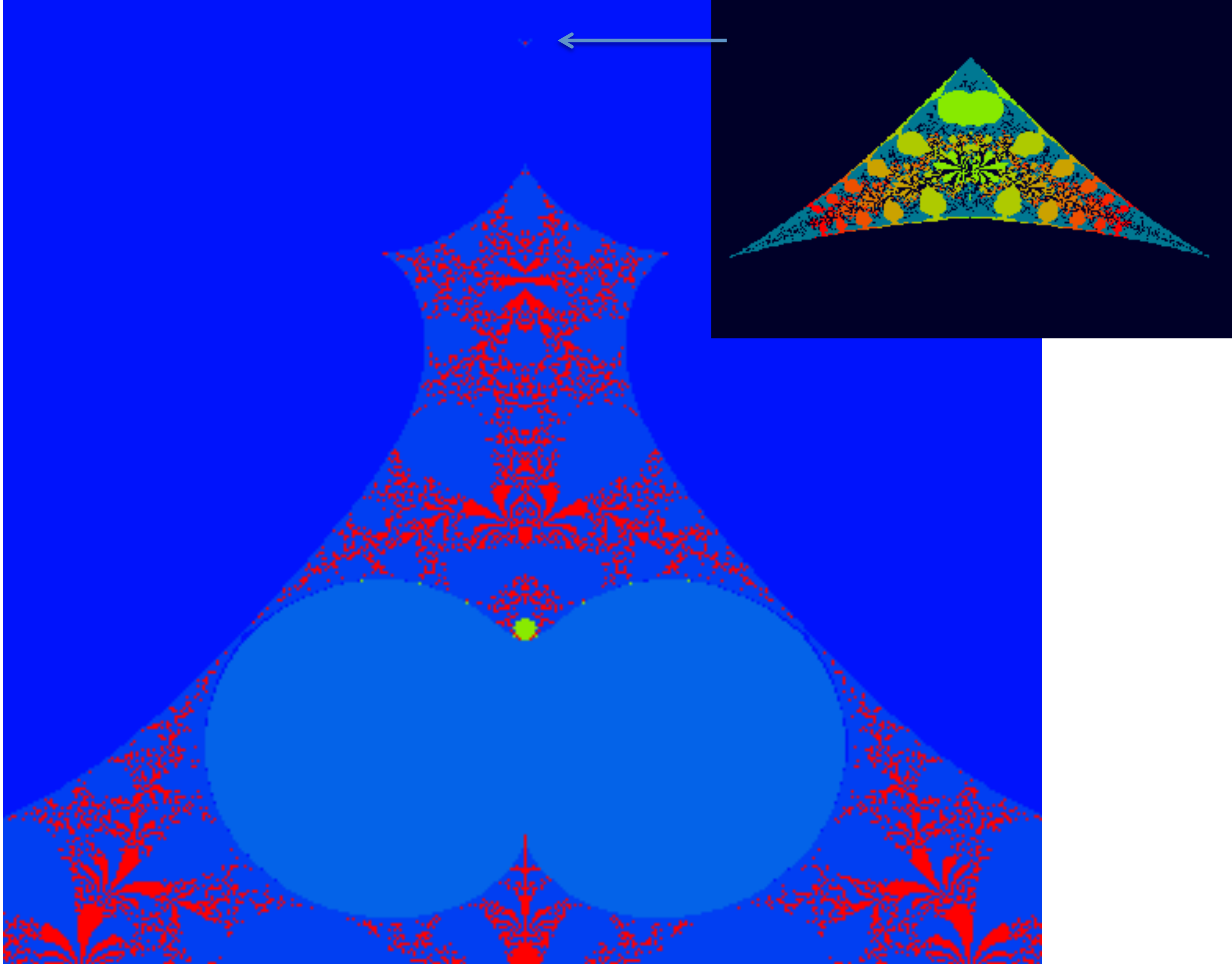
Exponential fractal

- http://planets.utsc.utoronto.ca/~pawel/iii/fractal.html

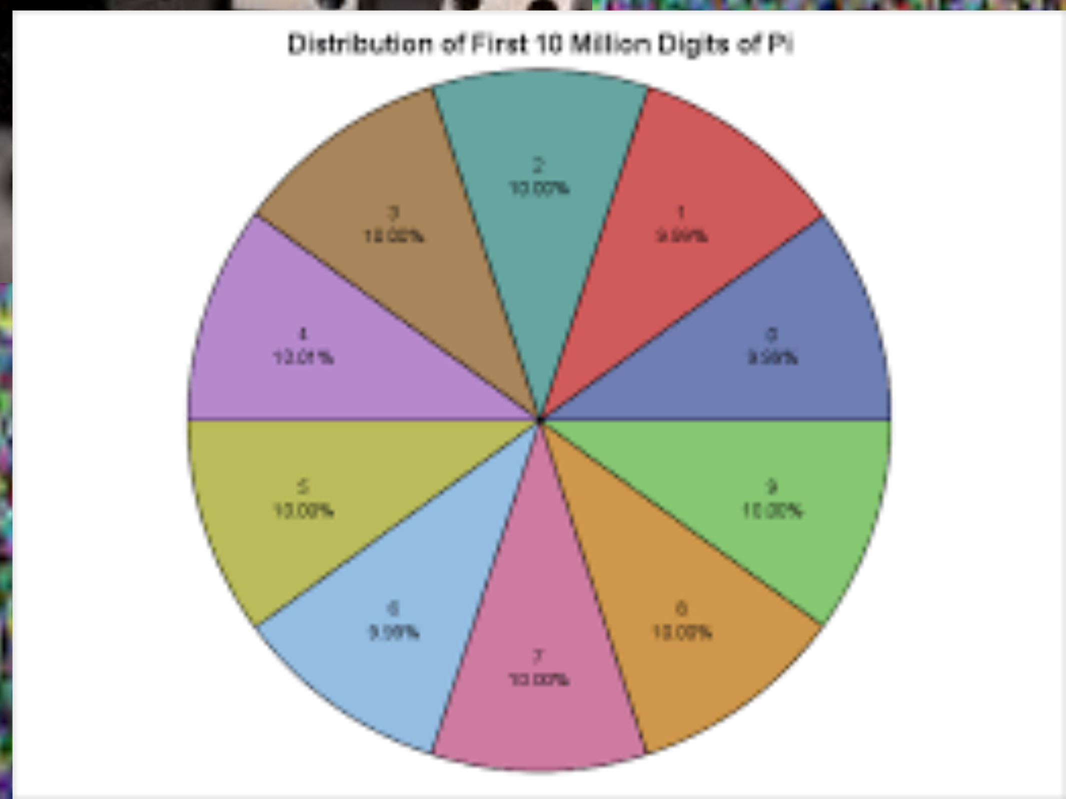


Fragment of Exponential Fractal





- **Random worlds**



Monte Carlo methods

- Statistical physics is mostly about randomness, entropy, mean values, random walks, and fluctuations.
- Frequent use of “random numbers” is not new, it started before the electronic computer era, but it became popular in computer calculations 70+ years ago, at the time when random, virtual histories of particles of radiation (n, p, e, γ) were needed to model the interaction of radiation with matter, among others, to design (thermo)nuclear bombs.
- *Casino de Monte Carlo, Monaco*

featured in James Bond 007 movies such as (I think): Golden Eye, Live and Let Die, Casino Royale, and one more I cannot recall.

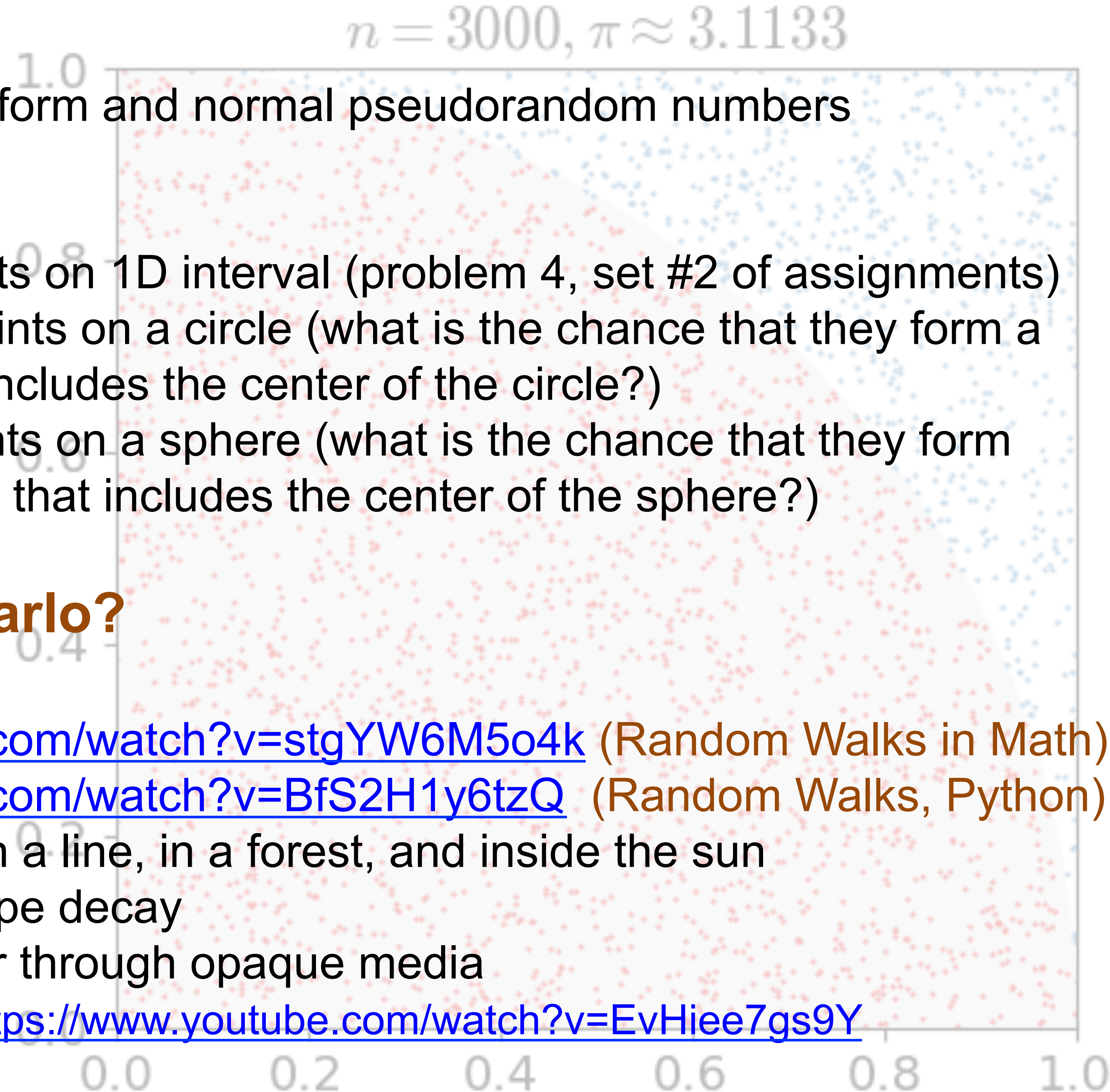
Monte Carlo, Monaco

MteCarlo methods

- We rely on *pseudo*random numbers? (*truly* random numbers are generated in nature. In math, decimal digits of π do *appear* to be truly random)
- Such numbers are uncorrelated*, but form a unique sequence that, if needed, can be repeated.
- Without repeatability, re-running MteCarlo programs to find bugs would be impossible, previous problems may disappear and new ones appear. At least during testing, we need to start from the same so-called seed value.

* - what else must be uncorrelated? (recall the story of Enigma). Mistakes were made in some old 'random' number generator algorithms that led to correlations!

Random numbers



- Generation of uniform and normal pseudorandom numbers
  - hist1.py
  - histo3.py
- Two random points on 1D interval (problem 4, set #2 of assignments)
- Three random points on a circle (what is the chance that they form a triangle that includes the center of the circle?)
- Four random points on a sphere (what is the chance that they form a tetrahedron that includes the center of the sphere?)

What is Monte Carlo?

https://www.youtube.com/watch?v=stgYW6M5o4k (Random Walks in Math)
https://www.youtube.com/watch?v=BfS2H1y6tzQ (Random Walks, Python)

- Random walks on a line, in a forest, and inside the sun
- Radioactive isotope decay
- Radiation transfer through opaque media
- Galton board https://www.youtube.com/watch?v=EvHiee7gs9Y