Lecture 5

- **Python in the Stochastic Universe**

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

Pseudorandom numbers and histograms

- Generation of uniform and normal random numbers

What is Monte Carlo?

- Pi by MtCarlo, convergence speed and the $N^{-1/2}$ trouble
- Comparison with Riemann (structured) integration of quarter-circle
- Radioactive isotope decay
- Radiation transfer through opaque media
- Galton board
- Random walks in 1D
- Stock market as random walker
- Solution of assignment set #1: pool-table.py, beach.py, series-bare.py
- series.py

Discussion of solutions to Assignment Set #1

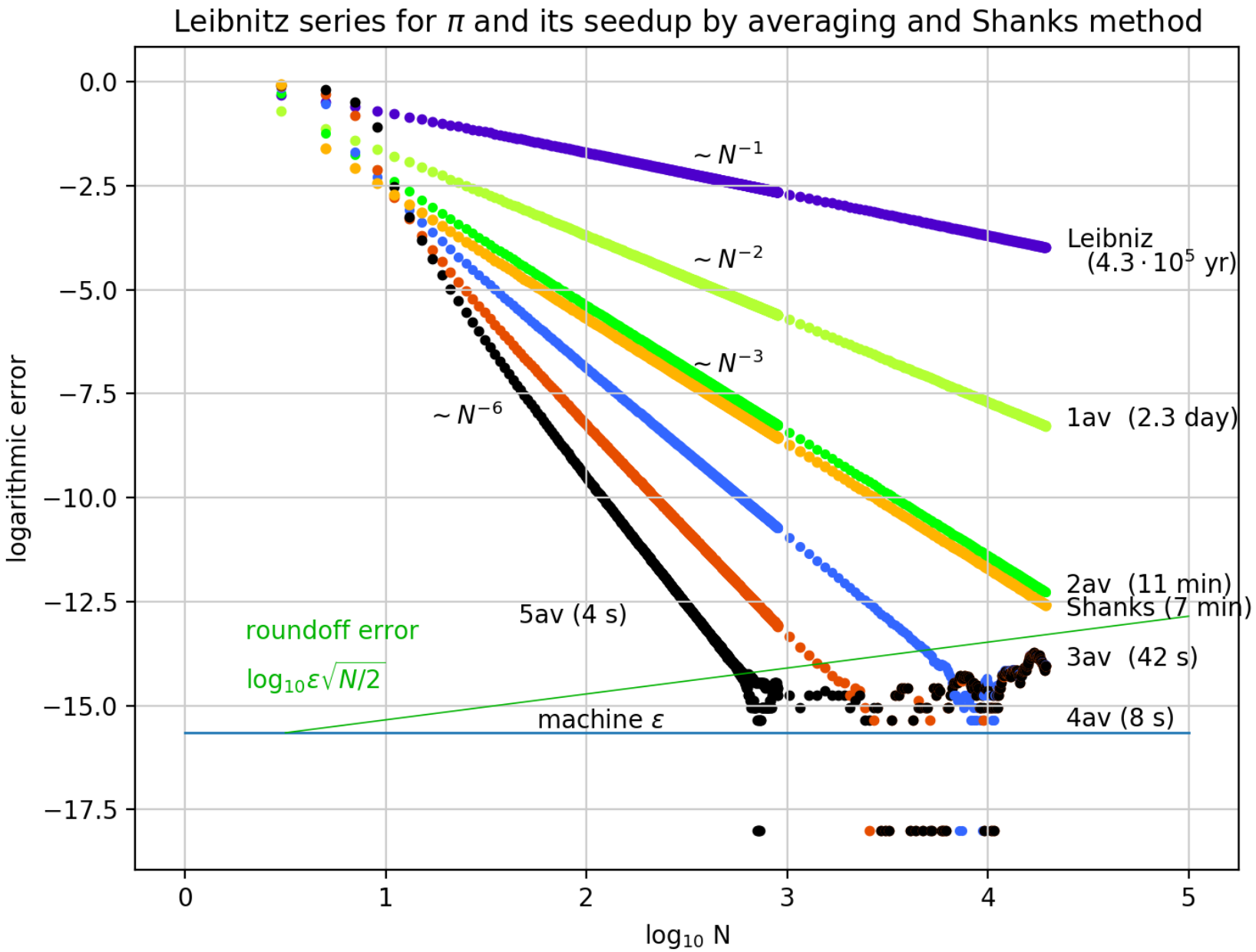
- Solutions are commented Python codes
- in our code depository:
- http://planets.utsc.utoronto.ca/~pawel/pyth
- study them line-by-line for programming style, use of graphics and numpy

series.py beach.py pool-table.py

X = 26.714m X = 5.5 m

Assignment Set #1, prob. 1.

series.py



- **Different random worlds:**
- **natural, mathematical, computational**

Monte Carlo methods

- For ex., 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 must have seen but 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 also *appear* to be truly random)

- Such numbers are uncorrelated*, but form a unique sequence that, if needed, can be repeated.
- Without repeatability, re-running MtCarlo 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 value, called seed.**

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

** - numpy,random.seed(s) function fixes the seed value s.

Random numbers

- Generation of uniform and normal pseudorandom numbers

- hist1.py use: numpy.random.rand [uniform distrib]
- histo3.py numpy.random.randn [normal distrib.]
- there's also numpy.random.choice([list]) [# of peaks]

- Two random points on 1D interval (problem 4, set #2 of assig)

Extensions:

- Three random points on a circle (what is the chance that they form a triangle that includes the center of the circle?)

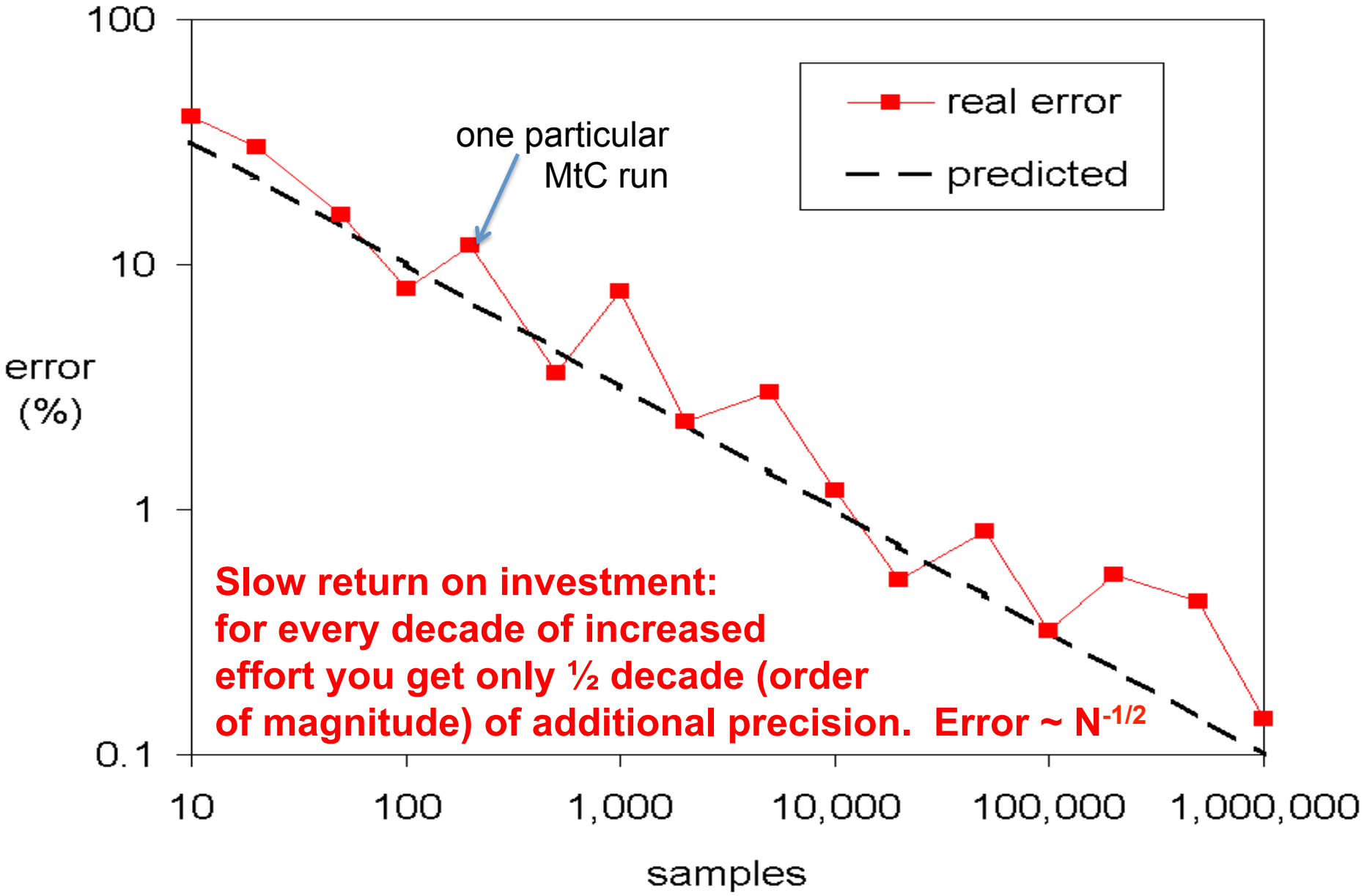
- Four random points on a sphere (what is the chance that they form a tetrahedron that includes the center of the sphere?)

What is Monte Carlo?

- Random walks on a line, in a forest, and inside the sun
  (i) Explanation of the the origin of the ubiquitous $\sqrt{N}$
  (ii) Calculation of mean, and mean-square distance in 1D
- Radioactive isotope decay
- Radiation transfer through opaque media
- Galton board
  https://www.youtube.com/watch?v=EvHiee7gs9Y

https://www.youtube.com/watch?v=stgYW6M5o4k
(Random Walks in Mathematics. Recurrent vs. transient walk)
https://www.youtube.com/watch?v=BfS2H1y6tzQ
(Random Walks programmed in Python)

The main disadvantage of MtCarlo



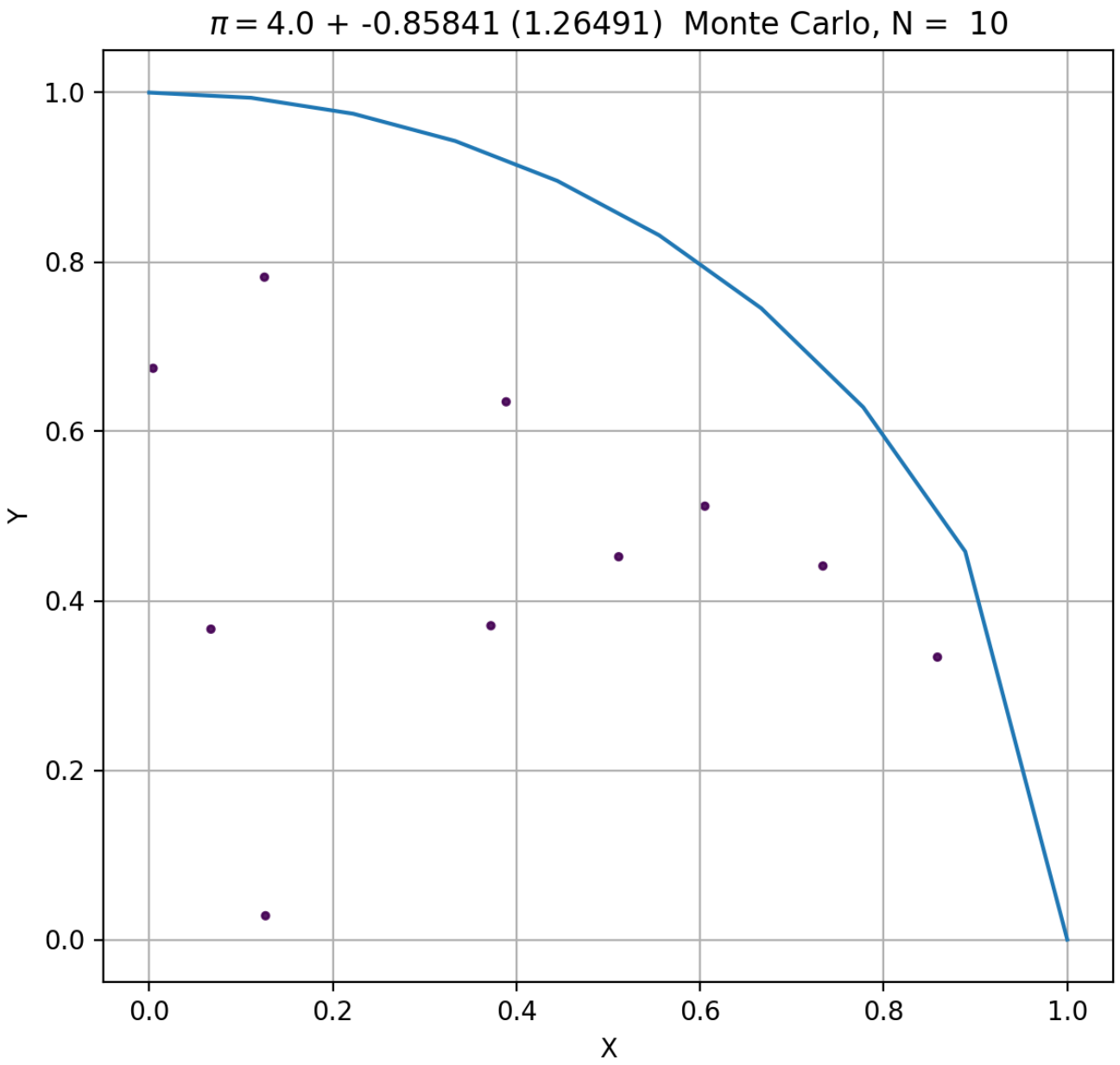
Pi by Monte Carlo method:

simple_mtc_pi.py

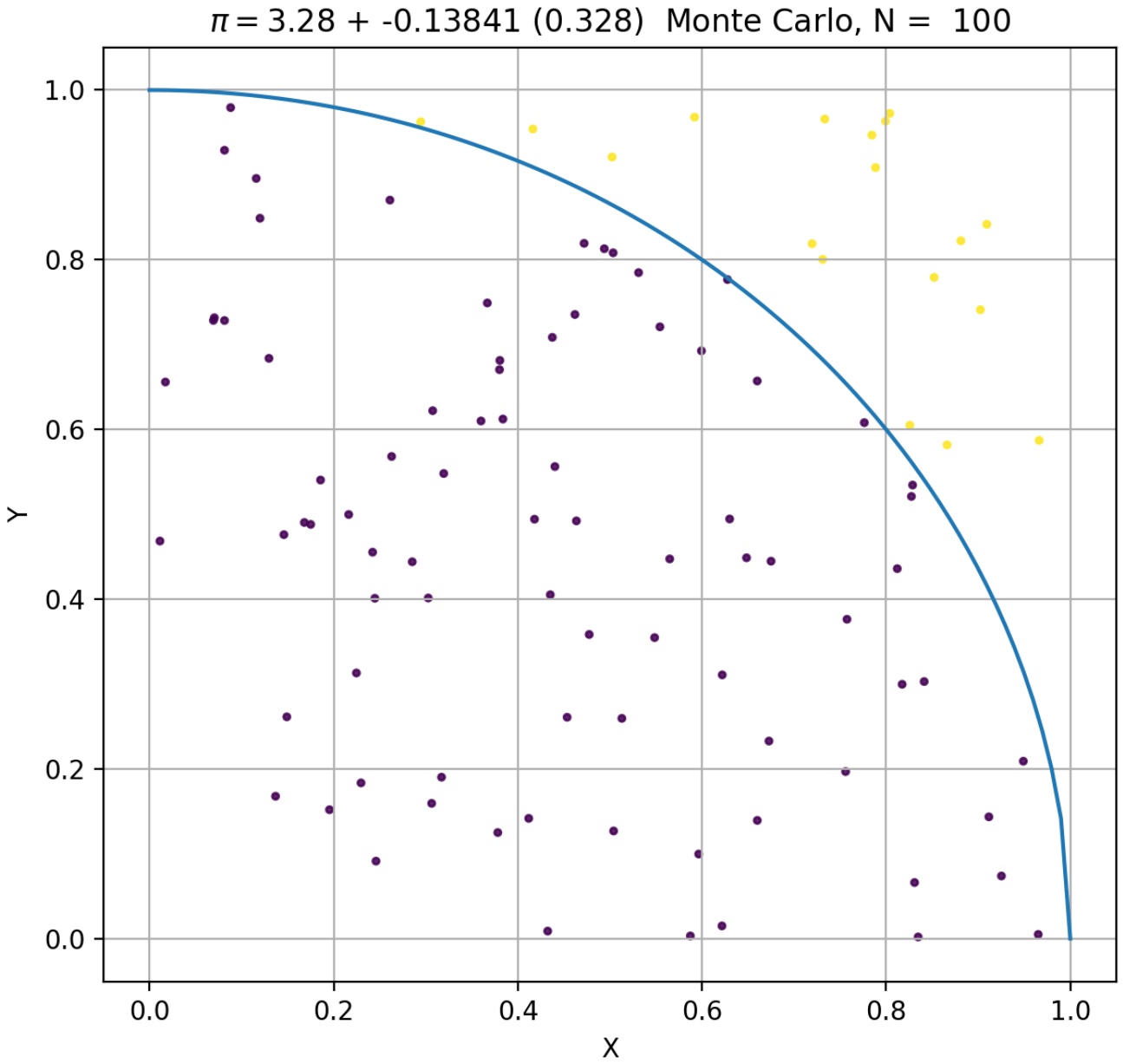
simple_mtc+int_pi.py simple_mtc+3int_pi.py

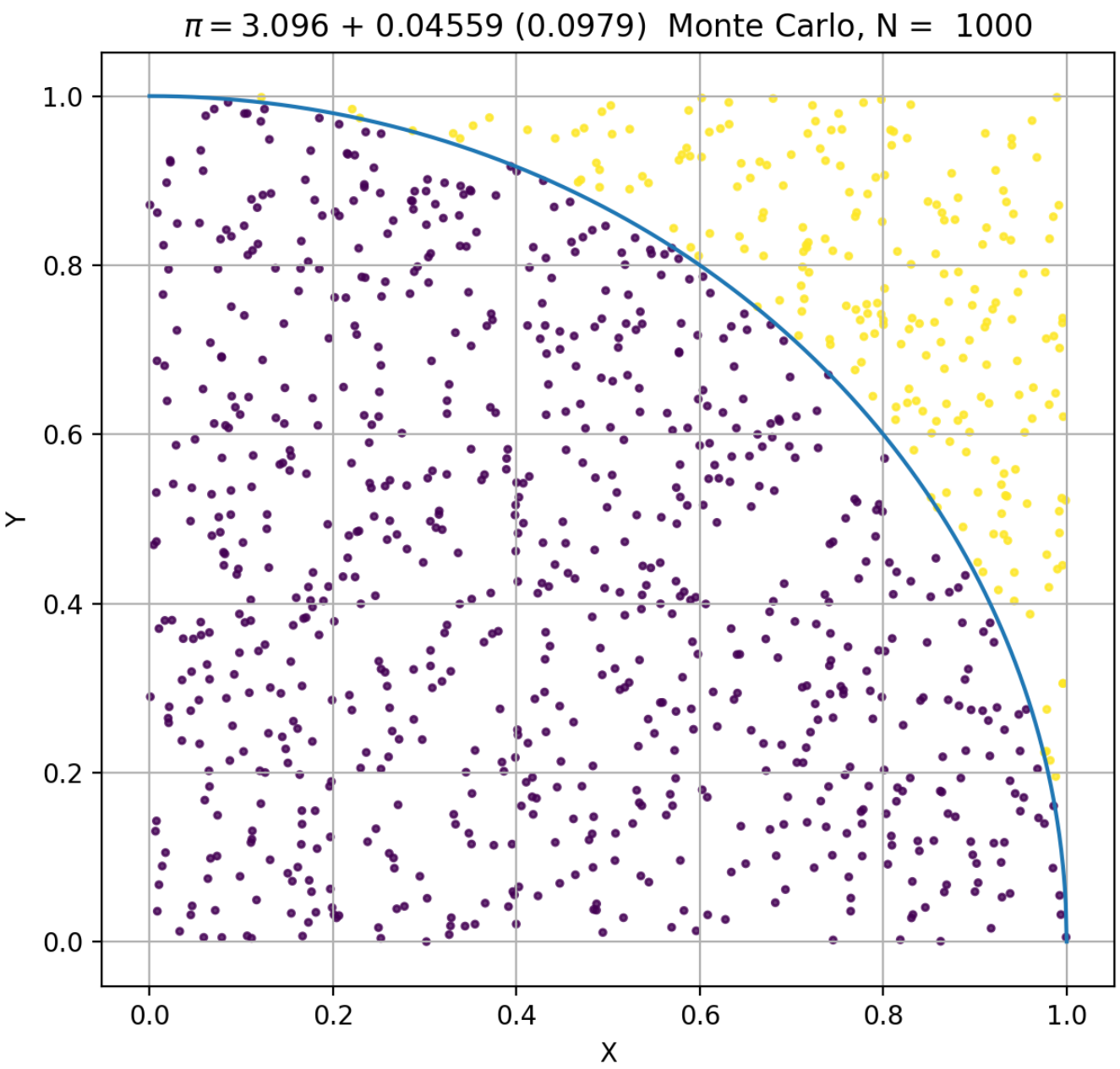
- Method: Throw many points uniformly on a unit square, and see what fraction of them fall into the ¼ circle $x^2 + y^2 < 1$
- Integral (area under curve) is equal ¼ of area $\pi*1^2 = \pi/4$,
- Therefore, estimated pi = 4*ratio of points that fall inside the quarter-circle to all points falling on a unit square.
- While finding if a point (x,y) drawn randomly is in the quarter-circle, the often seen expression $r=\text{sqrt}(x^2+y^2) < 1$ is a waste of time, one can use $x^2+y^2 < 1$ with the same effect.
- Compare `if (np.sqrt(x**2+y**2) < 1):`
  with `if ( x*x + y*y < 1):`
- Secondly, program will run *faster* if x is a big array, not a scalar (single float); for efficiency use numpy arrays.
- Compute the envelope of $y = \text{sqrt}(1-x^2)$ for plotting of simulation, and and independent numerical integration of y(x) (non-MtCarlo).

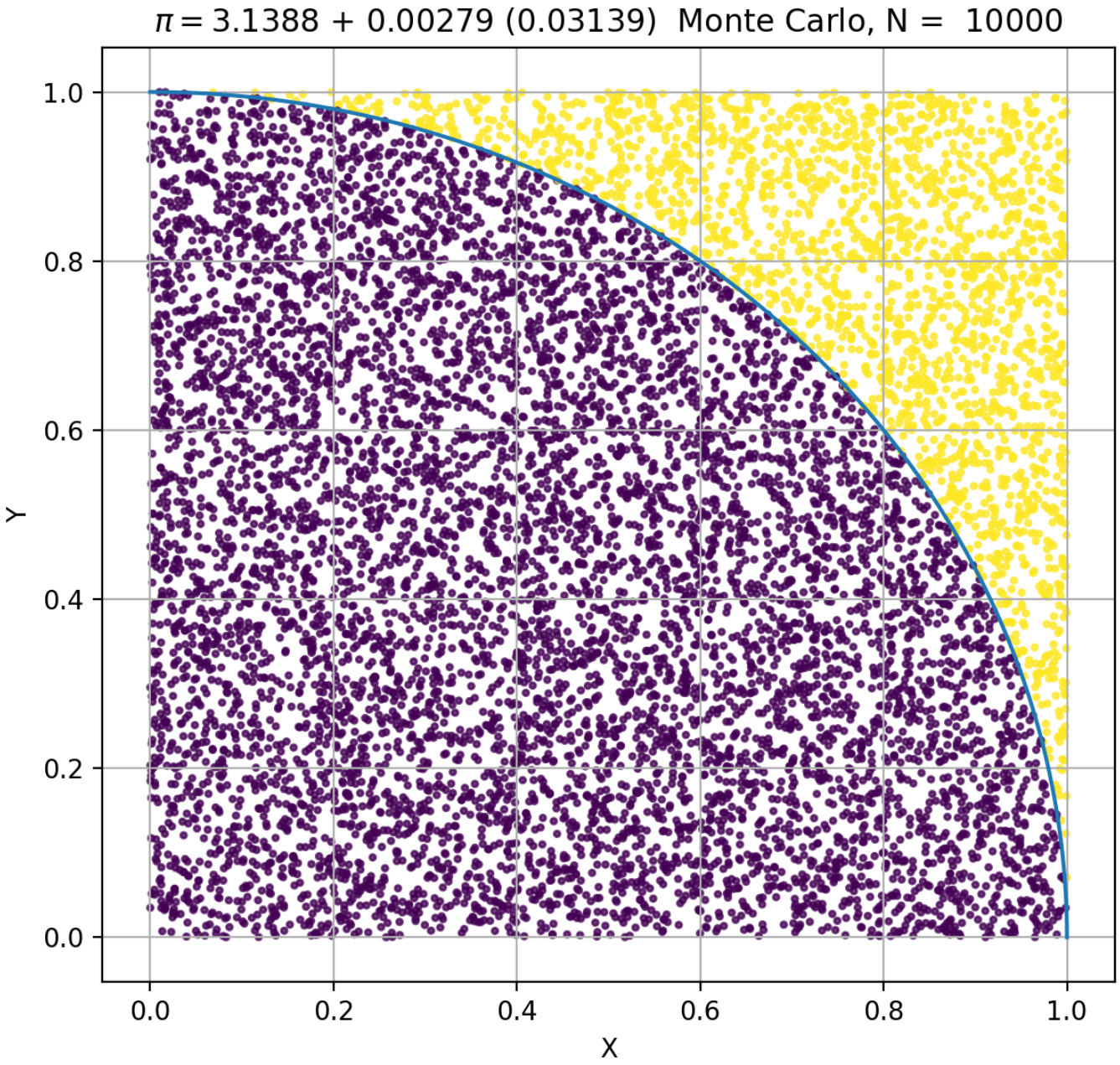
Pi by Monte Carlo method: simple_mtc+int_pi.py

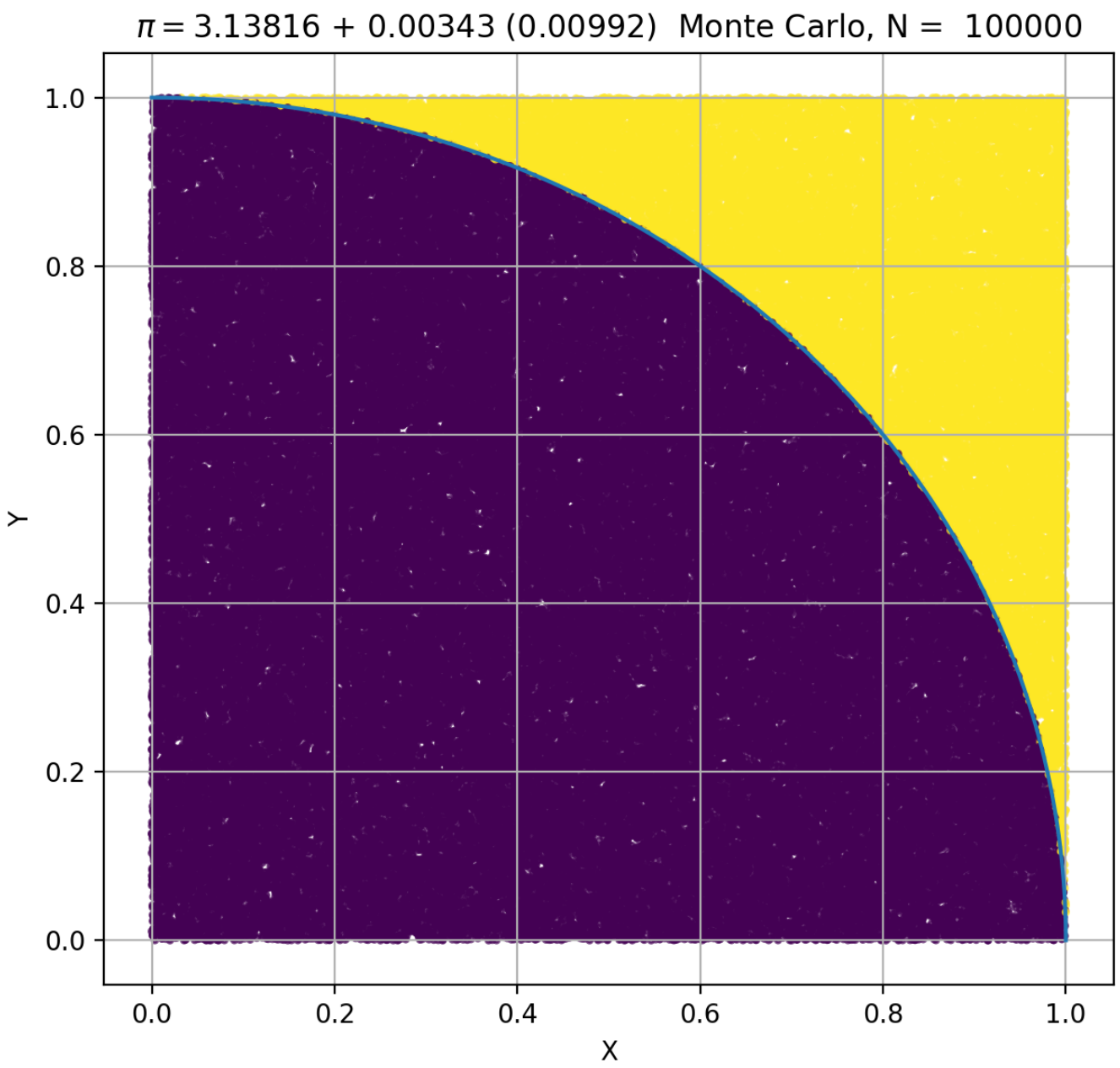


Pi by Monte Carlo method: simple_mtc+int_pi.py

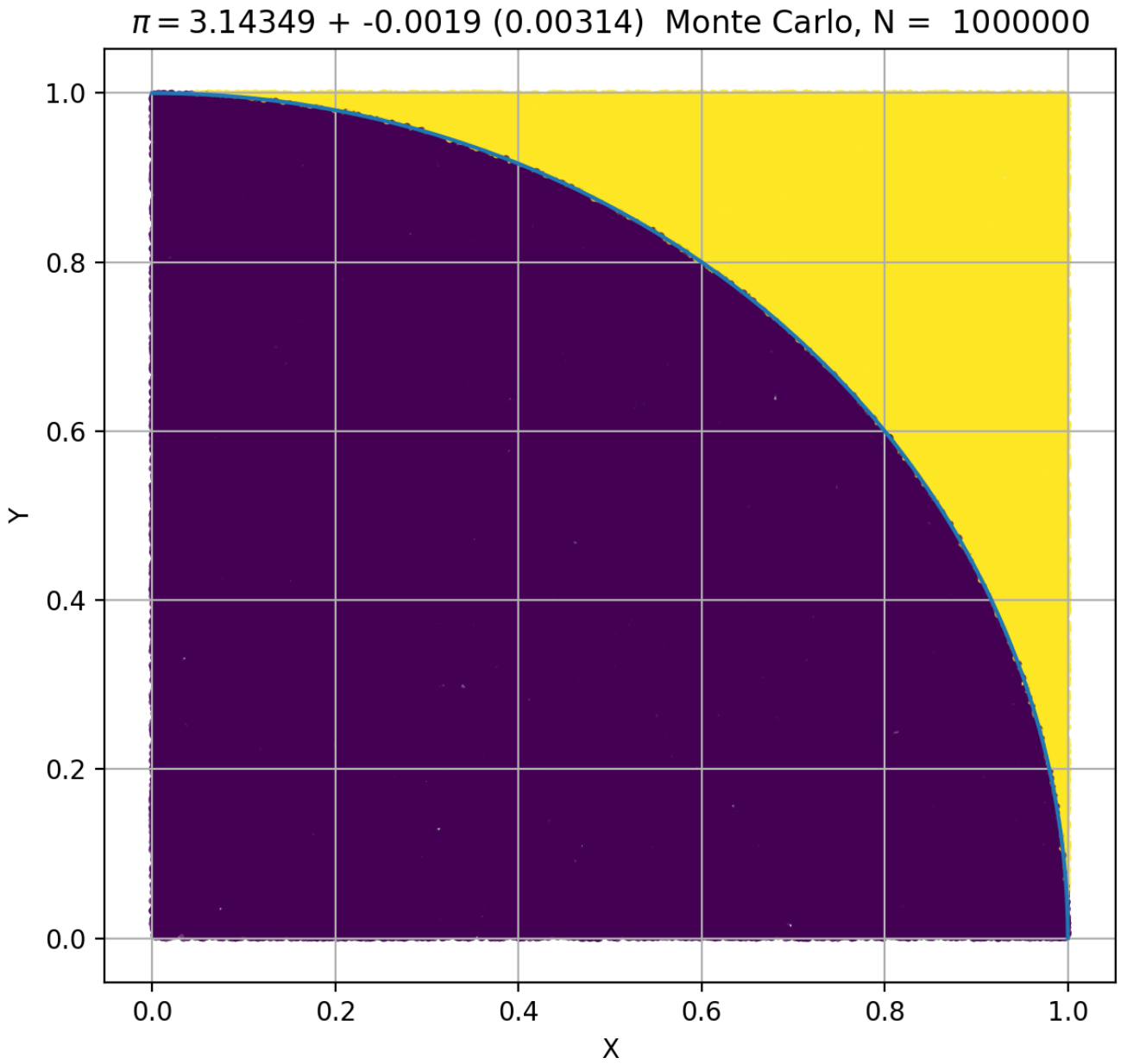



$\pi = 3.096 + 0.04559\ (0.0979)$ Monte Carlo, N = 1000
1.0
0.8
0.6
0.4
0.2
0.0
Y
0.0
0.2
0.4
0.6
0.8
1.0
X

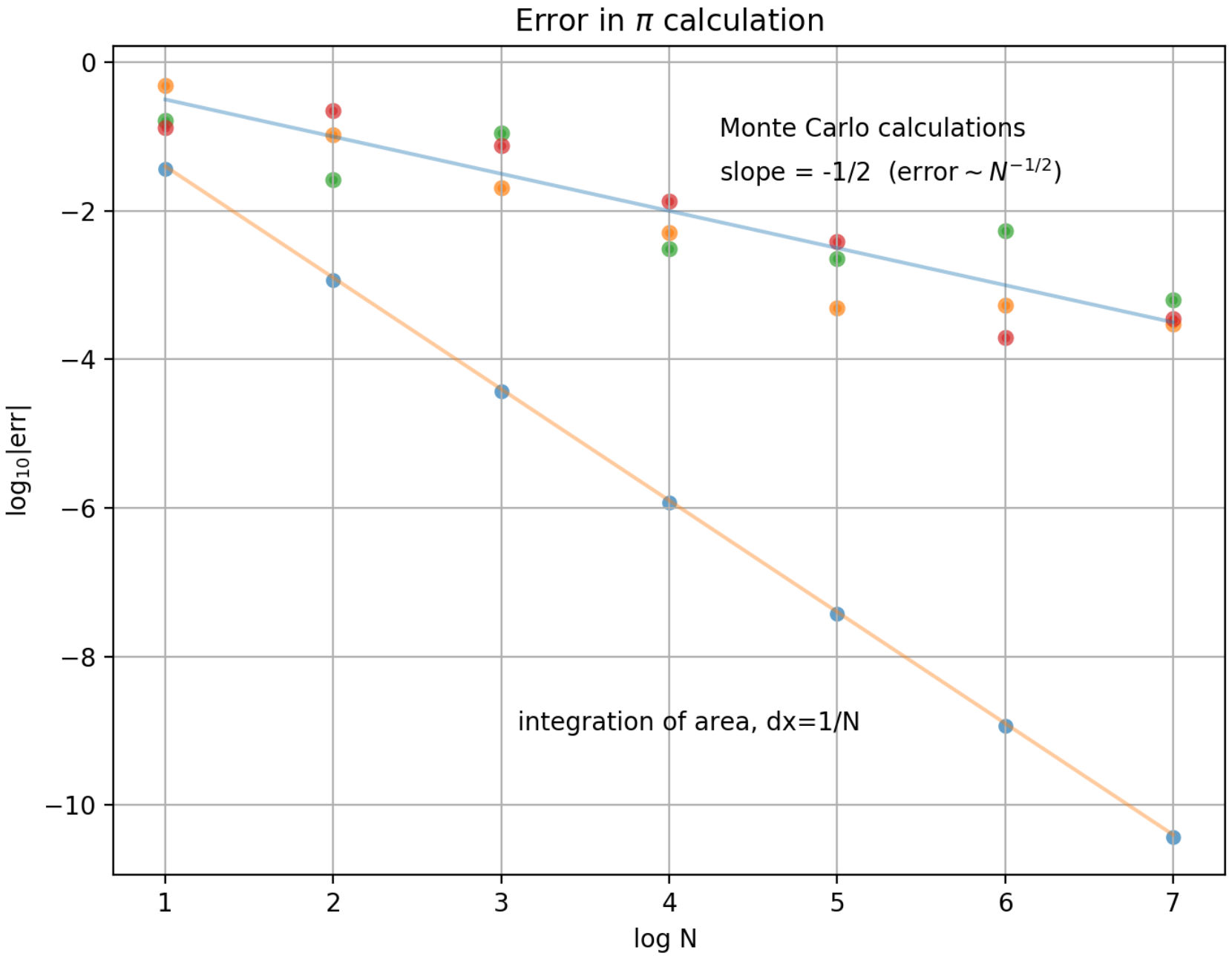

$\pi = 3.1388 + 0.00279\ (0.03139)$ Monte Carlo, N = 10000
1.0
0.8
0.6
0.4
0.2
0.0
Y
0.0
0.2
0.4
0.6
0.8
1.0
X


$\pi = 3.13816 + 0.00343\ (0.00992)$ Monte Carlo, N = 100000
1.0
0.8
0.6
0.4
0.2
0.0
Y
0.0
0.2
0.4
0.6
0.8
1.0
X

Pi by Monte Carlo method: simple_mtc+int_pi.py



simple_mtc+int_pi.py



Radioisotope decay: Uranium → Lead

Simulation using pseudorandom numbers and modeling individual nuclei.

Uranium ^{235}U (natural radioactive isotope) decays as follows:

^{235}U [*half-life=703.8 Myr*] → $^{231}Th + {}^{4}He + 2\ n$ →... → ^{207}Pb (stable)

[This is a spontaneous decay, different from the chain reaction induced by neutrons in reactors and nuclear bombs,
$^{235}U + n \rightarrow {}^{236}U \rightarrow {}^{92}Kr + {}^{141}Ba + 3\ n$.]

Let's call the initial number of uranium atoms U_0. We will explore different U_0's.

Plot the evolution of the amount of Uranium 235, U(t), for 4.5 Gyr (the age of Earth). Overplot theoretical $e^{-t*const}$ curve.

Radioisotope decay

Divide time in intervals of dt = 0.01 Gyr << half-life.

For every nucleus, with probability $P = \lambda\, dt$

remove it from a shrinking pool of uranium and add to the growing pool of stable lead nuclei.

Parameter λ is decay prob. per unit time per one nucleus.

It is related to half-life, which we denote as $t_{1/2}$. Namely,

$dU = -U \lambda\, dt \quad ==> \quad U = U_0 \exp(-\lambda t)$,

while in half-life notation we have

$$U = U_0\, (½)^{\,t/t12}$$

Comparing natural logarithms of these two equations, we get:

$$\lambda = \ln(2) / t_{1/2}.$$

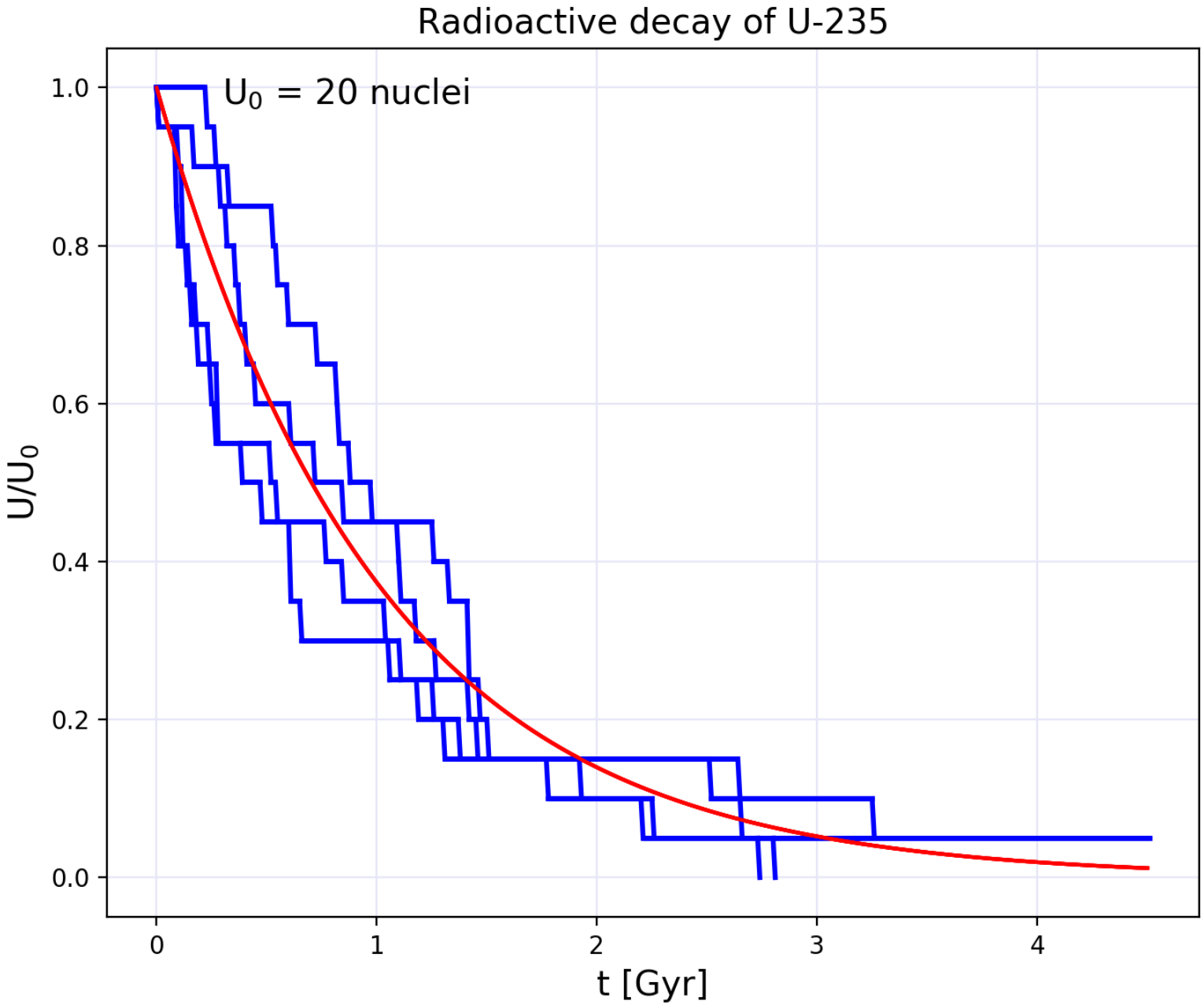
Use $P = \lambda\, dt = \ln(2)\, dt/t_{1/2}$ in every time step U times, and watch the number U shrink in time.

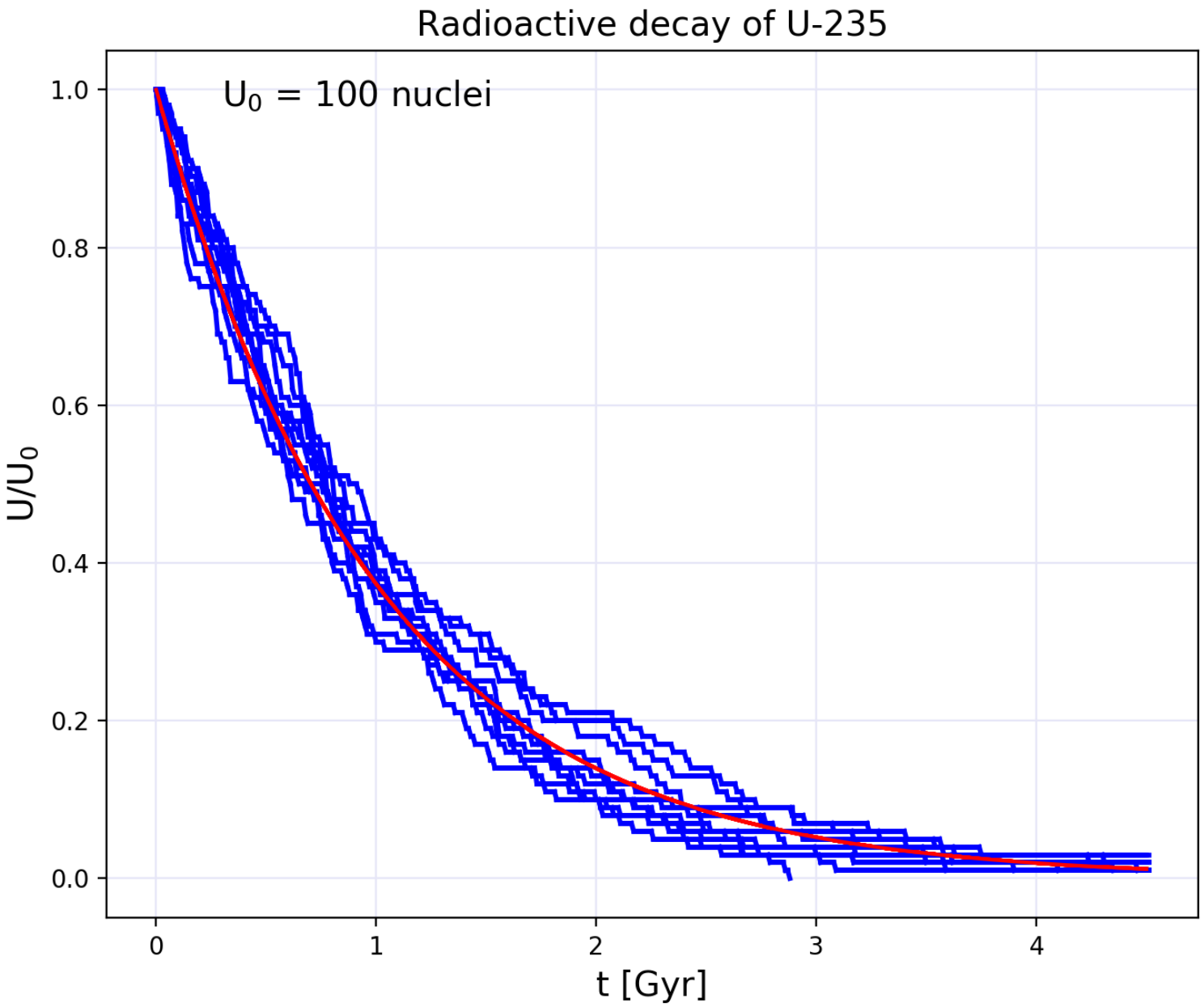
Program Radioactive.py

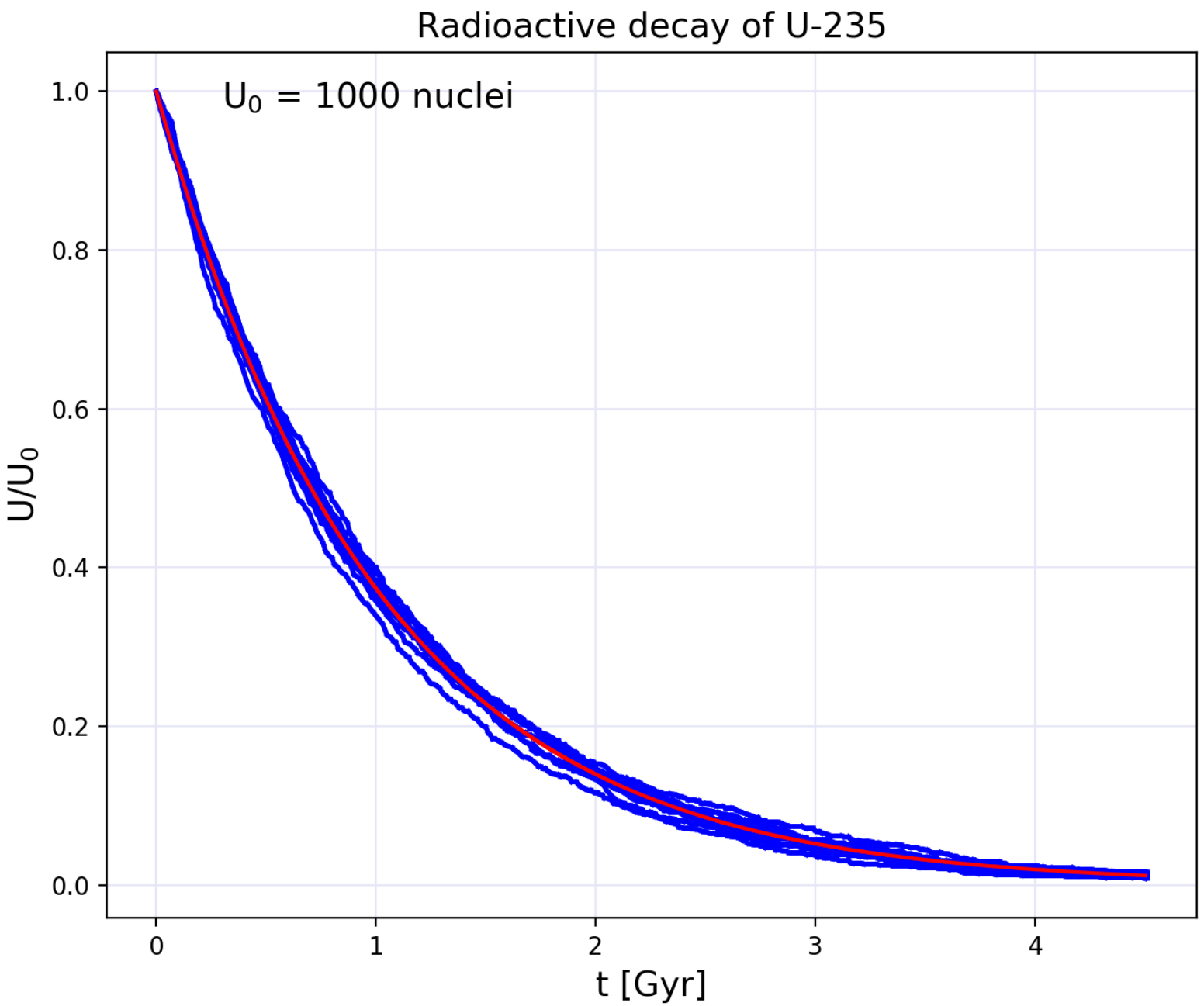
```
    (...)
    U = U0;   Pb = 0;   t = 0
    while (t < 4.5 and U > 0):
        t = t + dt
        U_previous = U
# give a chance to decay to all U nuclei: genrate U rnd numbers
        ran = np.random.random(U)
# if random numer from 0 to 1 is less than P, set decay to True
        decay = ran < P        # array of Boolean True/False values,
        N_decays = decay.sum()  # which can be summed (True=1, False=0)
        U = U - N_decays
        # You can print each simulation, good for debugging!
        # Pb = U0 - U
        # all U's decay to Pb's, so Pb+U==U0
        # print('at time t=',np.around(t,4),' U=',np.around(U,4),
        #  ' Pb/U=',np.around(Pb/U,3))
# plot 1 time segment of the decay curve
        plt.plot([t-dt,t],[U_previous/U0,U/U0],color=(0,0,1),linewidth=2)

# overplot exp(-lambd*tt)
    plt.plot(tt,theor,linewidth=1.5,color=(1,0,0))
    plt.show()
    ans = input(" one more? ")
```

Radioactive.py

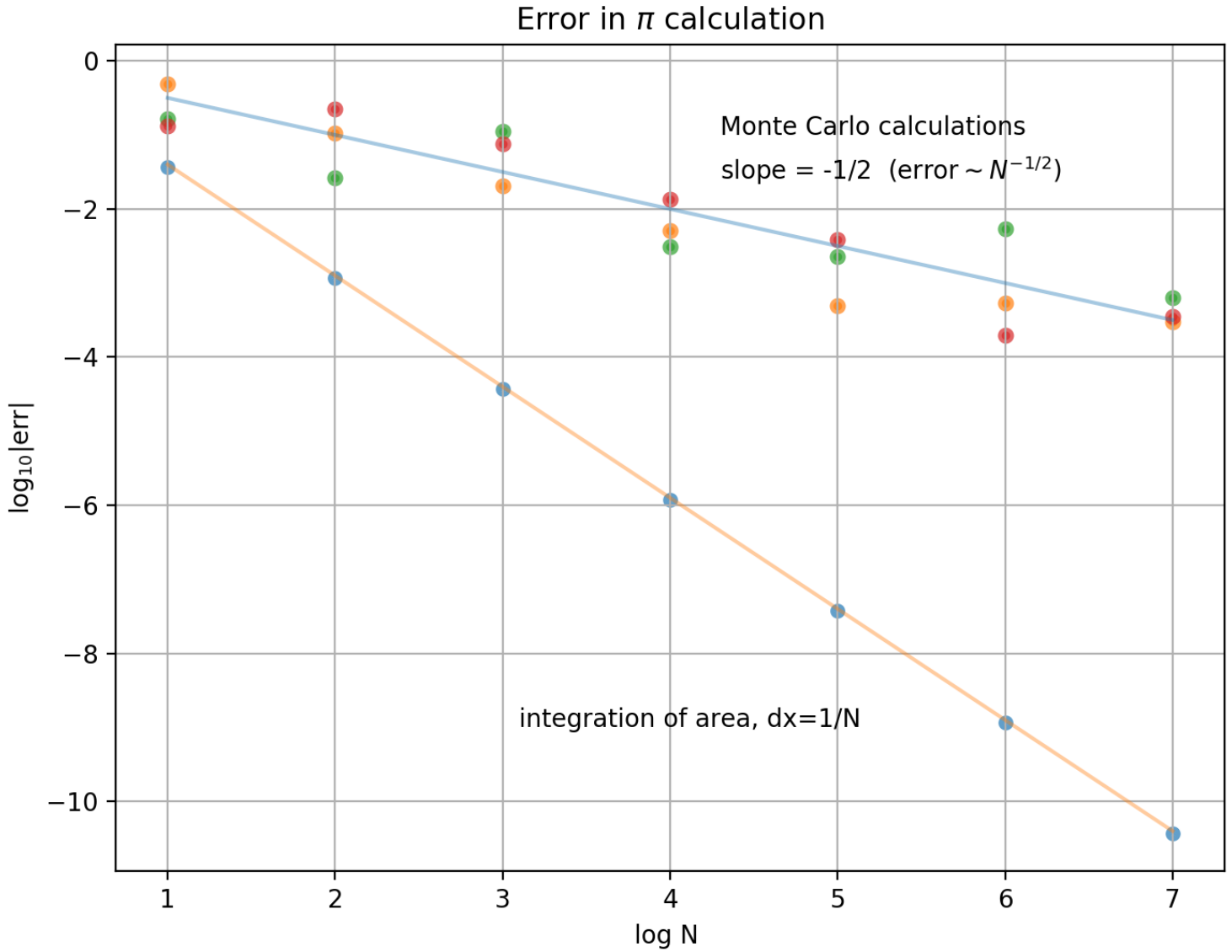



Radioactive decay of U-235
U_0 = 100 nuclei
1.0
0.8
0.6
0.4
0.2
0.0
U/U$_0$
0
1
2
3
4
t [Gyr]


Radioactive decay of U-235
U_0 = 1000 nuclei
1.0
0.8
0.6
0.4
0.2
0.0
U/U_0
0
1
2
3
4
t [Gyr]

Again, a sqrt(1/N) convergence is seen

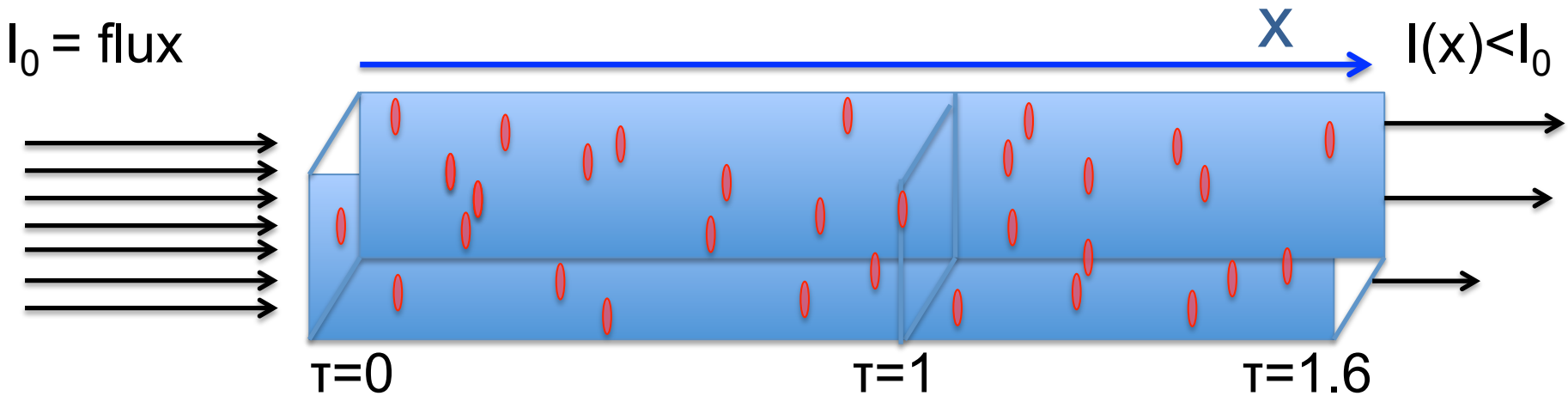
in the width of the curves as we have seen here in another context:



Radiation transfer:

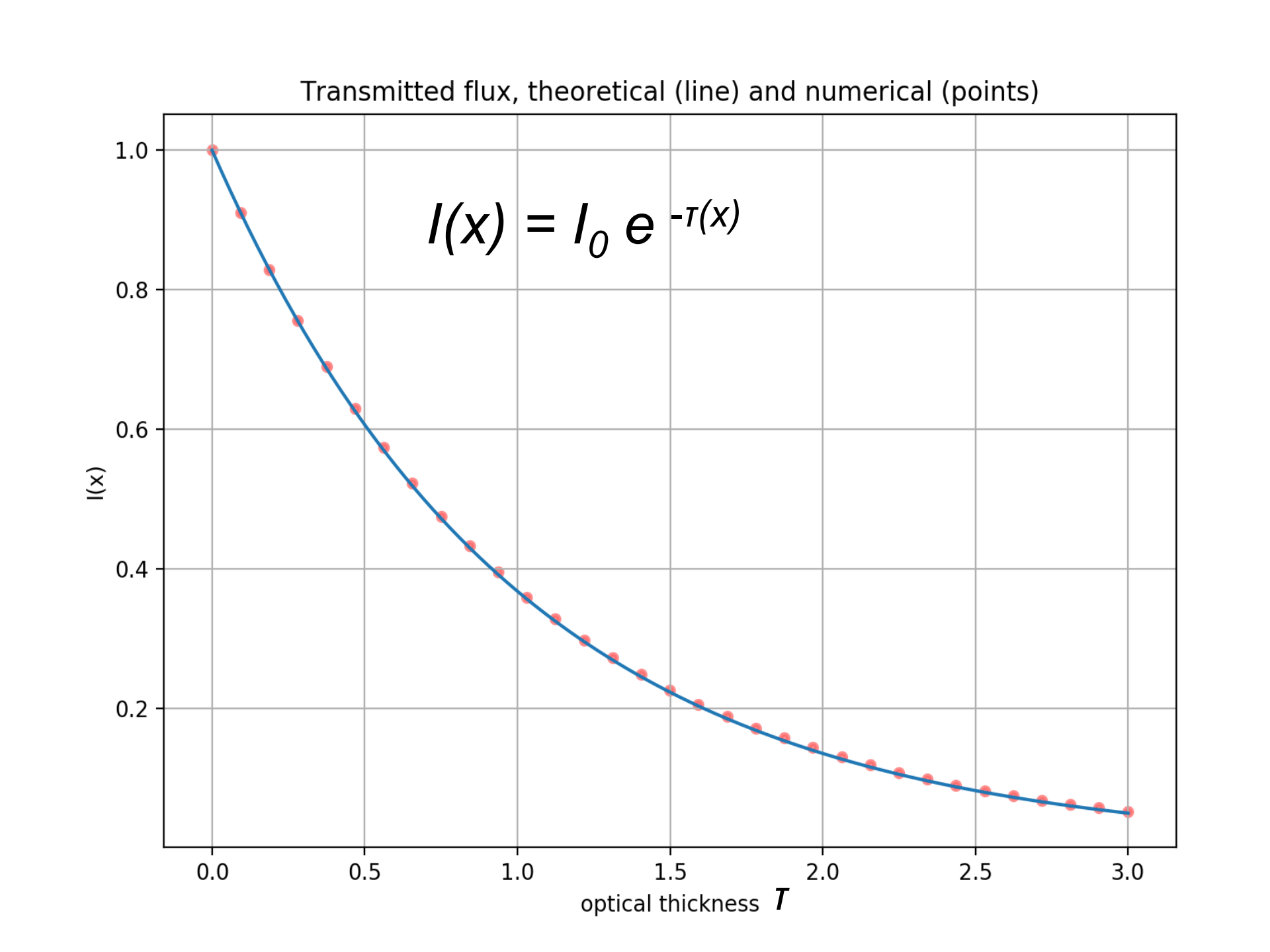
finding I(x) dusty-box-1.py

τ(x) = optical thickness = combined cross-sectional area, up to x, of absorbing and scattering particles, per unit side area of the beam. In a thin slice, dτ is a probability of photon extinction.



$d\tau := \sigma n(x)\, dx$ ➔ $\tau(x) = \text{integral}_0^x \{ \sigma n(x)\, dx \}$

If 1 particle's area σ and the number density of particles in space n(x) = const, then: τ(x) ~ x. In general $dI = I\, d\tau$ => $I = I_0\, e^{-\tau(x)}$


Transmitted flux, theoretical (line) and numerical (points)
$I(x) = I_0\, e^{-\tau(x)}$
I(x)
1.0
0.8
0.6
0.4
0.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
optical thickness τ