Lecture 6

- **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.

Solution of assignments from set #2:

pharaoh-2.py loan.py scan-grid.py

Monte Carlo (continued)

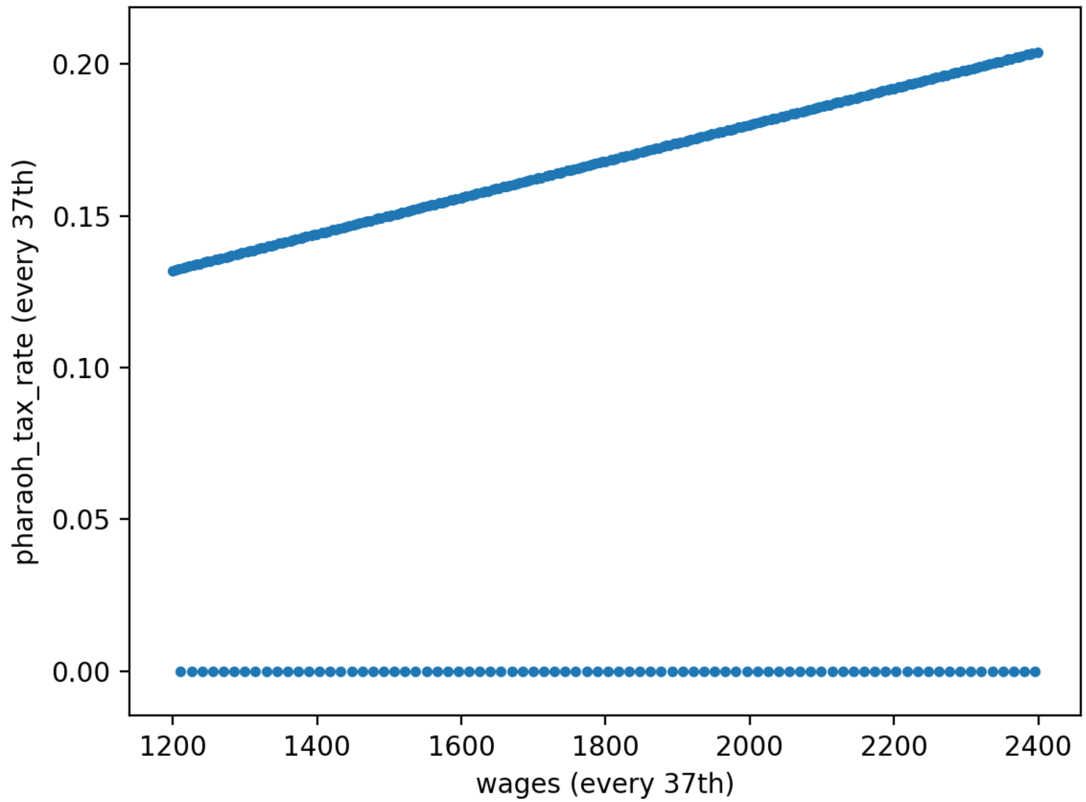
- Radiation transfer through opaque media
- Stock market as a random walker

Assignments #2

1. Egyptian taxes problem

- Imperial taxes are independent of local (tax collector's) fees [but no points subtracted if you are in tutorial group 1, and assumed otherwise]
- It does not matter much how you eliminate every 4th taxpayer from imperial taxation, {1,5,9,13,...} or {3,7,11,15,...} or N/4 random persons (but the formulation discourages you from using random numbers here). You should always compare different schemes!

- pharaoh-2.py



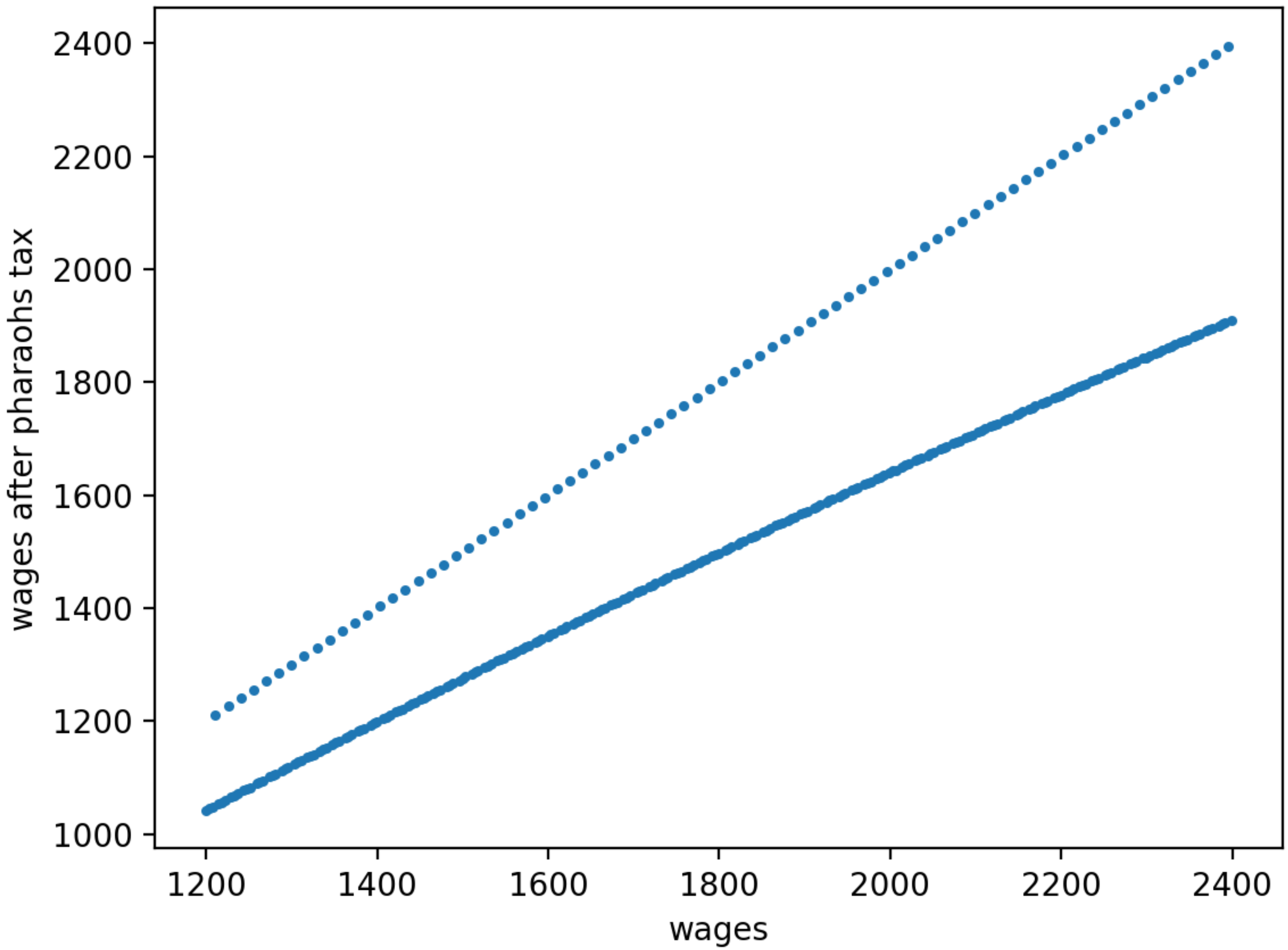
• pharaoh-2.py (fragments)

```
•   N = 12000     #  number of taxpayers
•   # wages data generation
•   wages = np.linspace(1200,2400,N)
•   print("Total wages before tax", int(sum(wages)),'=',int(sum(wages))//1000,'k')

•   # imperial tax first. Introducing so-called list comprehension
•   pharaoh_tax_rate = 0.06*(1+wages/1000)       # pharaoh's tax rate’s > 2.2*6%
•   builders_list = [i for i in range(3,N,4)]    # list comprehension,
•   pharaoh_tax_rate[builders_list] = 0.         # could be done in a loop as well
•   treasury_takes = (wages * pharaoh_tax_rate).astype(int) # rounded to int
•   pharaoh_gets = treasury_takes.sum()          # a total that pharaoh gets
•   print(" Pharaoh tax ",pharaoh_gets,' =',pharaoh_gets//1000,'k \n')
•   wages_aft_tax = wages - treasury_takes        # taxman draws from this list

(...)

•   # now the tax collector's withholdings, standard rate is 0.015
•   # 2 out of 13 people have collector's tax cut
•   tax13 = [0.015]*13                 # list multiplication
•   # arbitrary 2 of 13 get a tax break, numbers 4,10 are arbitrary, try other
•   tax13[4] = tax13[10] = 0.007
•   print(' the first 13 of tax collector rates \n', tax13)
•   # this pattern will be applied to almost all taxpayers
•   N_remaining  = N%13      # remainder of division into 13 groups
•   print("N_remaining",N_remaining)
•   # again, list multiplication (making it longer), not array multiplication!
•   taxman_rates_lst = (N//13)*tax13 + N_remaining*[0.015]
```

Assignments #2

1. Egyptian taxes:

- Similarly, it does not matter much how you reduce local taxation of a fraction = 2/13 of taxpayers, you can generate a pattern of how the first 13 taxpayers are taxed (fully or with a tax), and then repeat it 12000//13 times

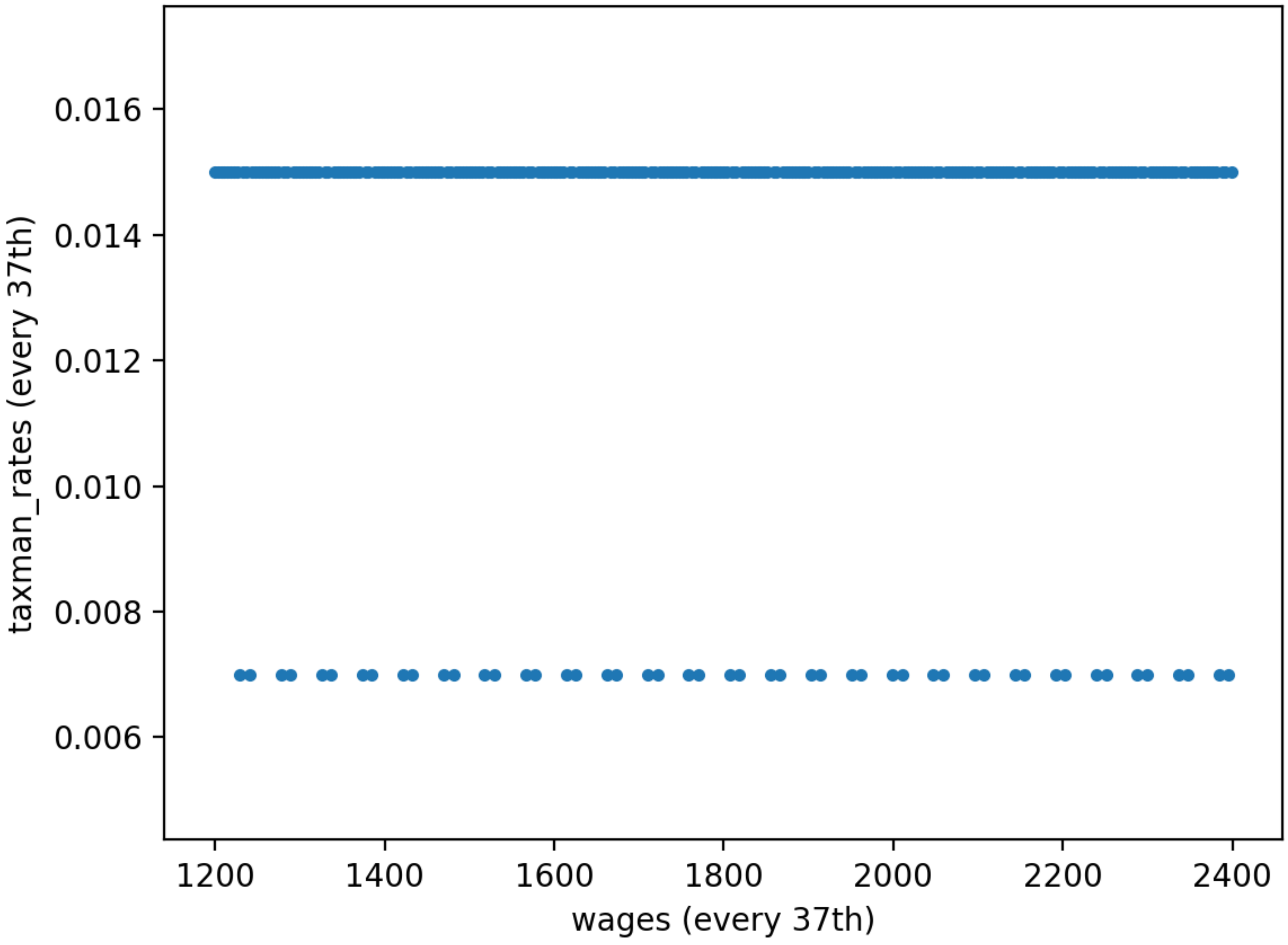


pharaoh-2.py

Assignments #2

1. Egyptian taxes:

- Similarly, it does not matter much how you reduce local taxation of a fraction = 2/13 of taxpayers, you can generate a pattern of how the first 13 taxpayers are taxed (fully or with a tax), and then repeat it 12000//13 times



1. Egyptian taxes:

Assignments #2

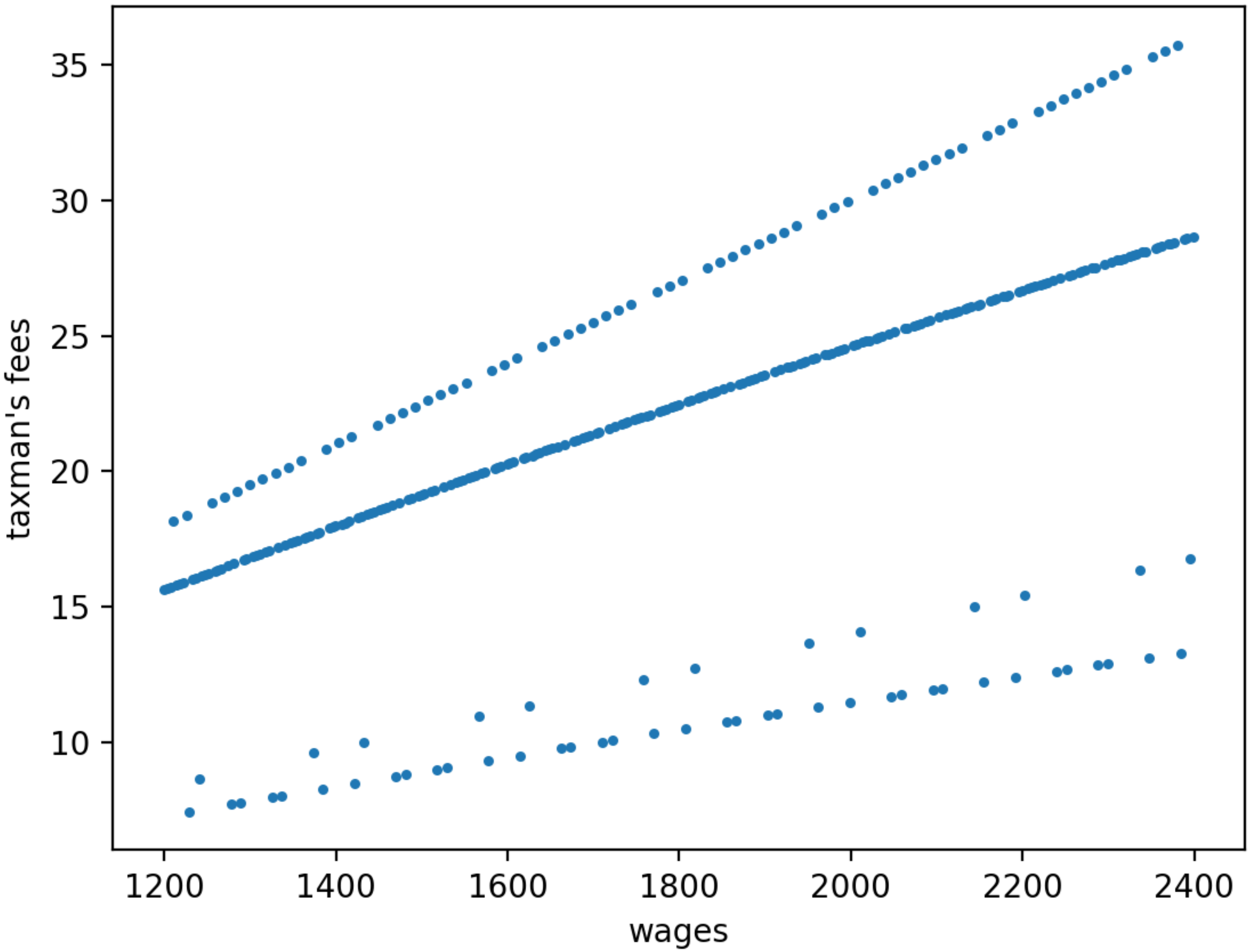
Answer: Out of

21600 k b.o.g. earnings, taxpayers pay

2782 k b.o.g. to Pharaoh's treasury, and

253 k b.o.g. to you (tax collector)

pharaoh-2.py



Assignments #2

2. Student Loan problem

- Given the specific loan conditions, and specific payment schedule,
- predict the p(t) loan reduction history and the repayment date.
- One loop suffices, but you can use two to overplot an alternative history.

```
p = 36000;      interest = 4.2e-2;      inflation = 2.6e-2
total_pay1 = 0;              equiv_pay1 = 0

for i   in range(200):                      # i is the number of the month
    pay = 300
    if (i > 2*12):                          # after 2 years apply a
        pay = 380                           # larger monthly re-payment

#  p = p*(1+interest/12) – pay              # this is the heart of the loan calculator
   p = (p-pay)*(1+interest/12)              # use this line if you pay before compound date

   total_pay1 += pay                        # this is optional, and the following even more so:
   equiv_pay1 +=  pay/(1+inflation/12)**i        # buying power [convert to init. $$]
   print (i,'yr,mo',i//12,1+i%12, ' loan =',p)
   plt.scatter (i,p,color=(0,0,1), s=9, alpha=0.6)      # blue points = main variant, interest=const.
   if (p <= 0):                                     # if loan amount drops to (or below) zero
       break                                        # then break out of the loop
```

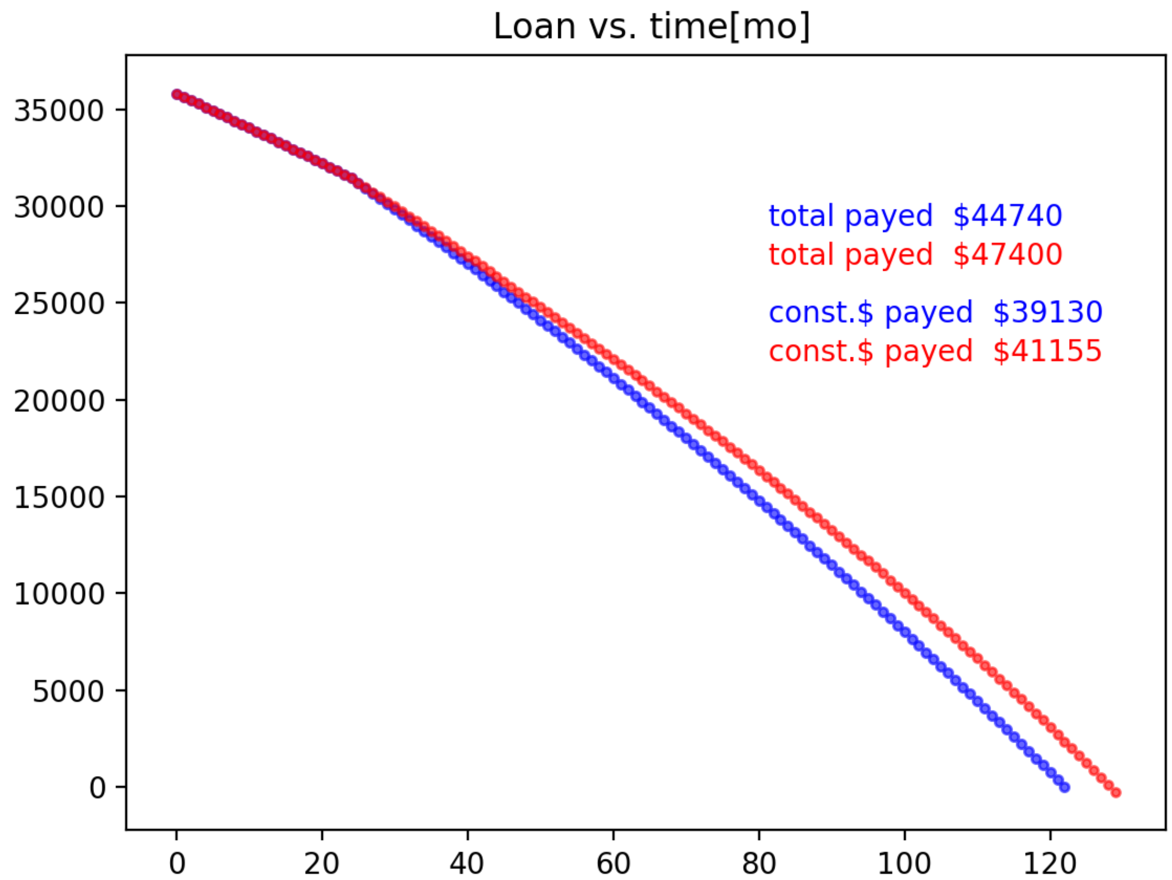
Assignments #2

2. Student Loan problem

- Answer: Loan payed after 129 mo. = 10 yr and 3 months, if bank rates constant @ 4.2%

Month:

- 0 yr,mo 0 1 loan = 35826.
- 1 yr,mo 0 2 loan = 35651.
- 2 yr,mo 0 3 loan = 35476.
- 3 yr,mo 0 4 loan = 35300.
- 4 yr,mo 0 5 loan = 35123.
- (...)
- 118 yr,mo 9 11 loan = 1483.8
- 119 yr,mo 9 12 loan = 1109.0
- 120 yr,mo 10 1 loan = 732.90
- 121 yr,mo 10 2 loan = 355.46
- 122 yr,mo 10 3 loan = -23.29
- i.e. (122 months = 10 yr + 3 mo)

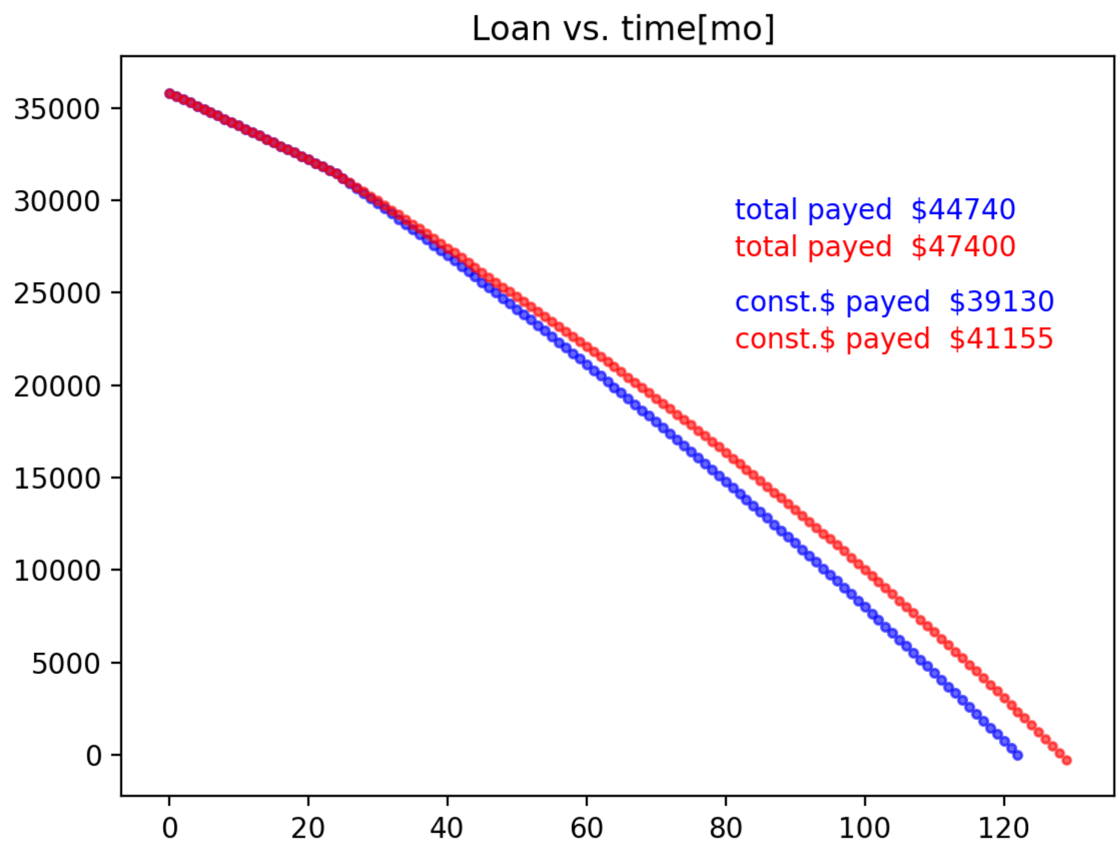


- **Total payed = $44740**
- **Total payed - loan = $8740**
- Const.$ effective interest 8.7%, assuming annual inflation rate 2.6%

Assignments #2

2. Student Loan – alternative rates case

- Answer: Loan repayed after 129 mo. = 10 yr 10 mo. if bank stepwise raises rates
- 0 yr,mo 0 1 loan = 35826.0
- 1 yr,mo 0 2 loan = 35651.3
- 2 yr,mo 0 3 loan = 35476.1
- 3 yr,mo 0 4 loan = 35300.3
- 4 yr,mo 0 5 loan = 35123.8
- 5 yr,mo 0 6 loan = 34946.8
- (...) ~10 long years pass here
- 124 yr,mo 10 5 loan = 1604.8
- 125 yr,mo 10 6 loan = 1232.4
- 126 yr,mo 10 7 loan = 858.3
- 127 yr,mo 10 8 loan = 482.4
- 128 yr,mo 10 9 loan = 104.7
- 129 yr,mo 10 10 loan = -274.8
- (129 months = 10 yr + 10 mo)



- **Total payed = $47400**
- **Total payed - loan = $11400**
- Const.$ effective interest 13.5%, assuming annual inflation rate 2.6%

Assignments #2

3. Scanning the grid

scan-grid.py

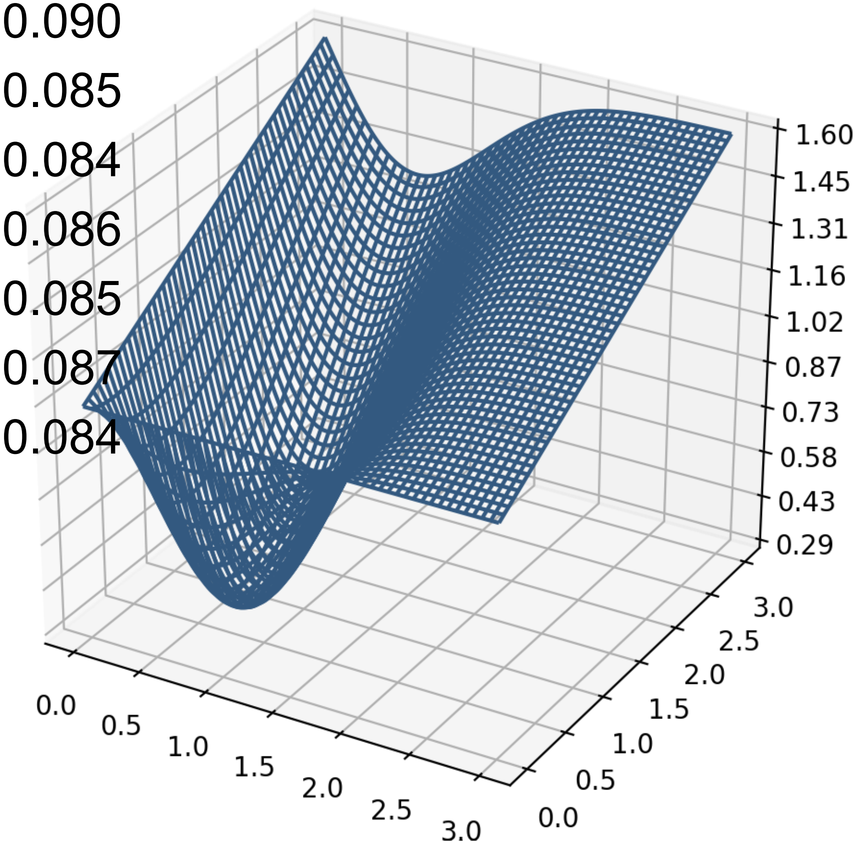
```
def f(x,y,D):
    return 1 +0.1*D*x -2.**2.5*x*y*np.exp(-x-y*y)


# main (driver) program
N = 1000
xx = yy = np.linspace(0,3,N)
X,Y = np.meshgrid(xx,yy)                # for plotting later
for D in range(10):                     # D =  last digit of student number
    A = np.empty((N,N),dtype=float)        # declare empty grid
    for ix in range(N):                    # fill the array with function values
    x = xx[ix]
    A[ix,:] = f(x,yy,D)                 # function given vector yy returns a vector
    arg = A.argmin()                    # find index in a flattened 2D array
    ix,iy = (arg//N, arg%N)             # turn arg back into 2D index
    print(...)
 #  plotting and timing
    (...)
```

Assignments #2

3. Scanning the grid

scan-grid.py

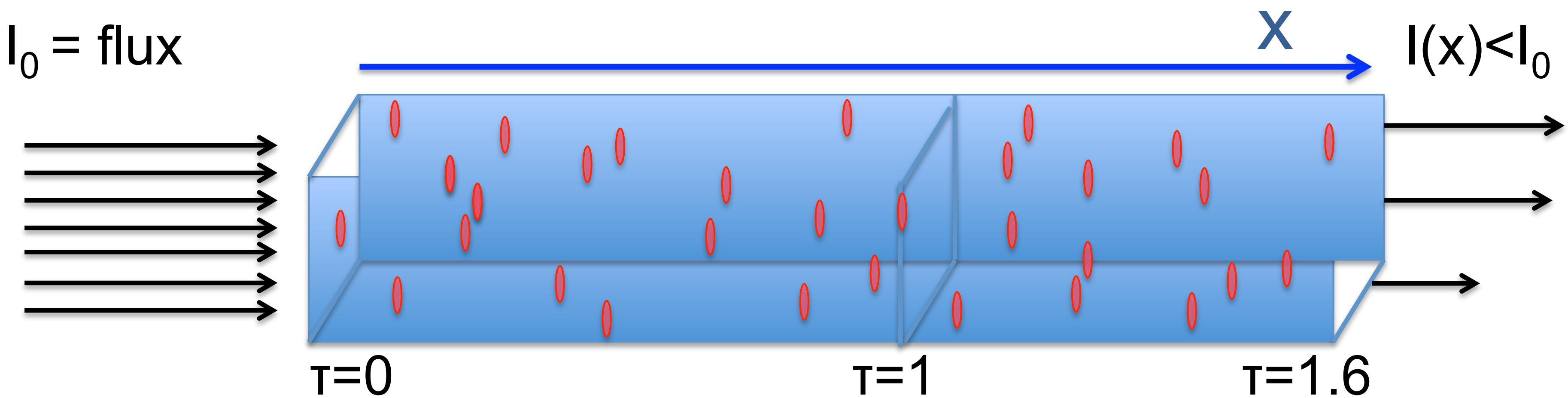
D = 0 x,y: 1.0000, 0.7057, min = 0.10748 t[s] 0.084
D = 1 x,y: 0.8979, 0.7057, min = 0.20225 t[s] 0.099
D = 2 x,y: 0.8138, 0.7057, min = 0.28778 t[s] 0.097
D = 3 x,y: 0.7417, 0.7057, min = 0.36543 t[s] 0.090
D = 4 x,y: 0.6757, 0.7057, min = 0.43619 t[s] 0.085
D = 5 x,y: 0.6186, 0.7057, min = 0.50083 t[s] 0.084
D = 6 x,y: 0.5646, 0.7057, min = 0.55992 t[s] 0.086
D = 7 x,y: 0.5165, 0.7057, min = 0.61395 t[s] 0.085
D = 8 x,y: 0.4715, 0.7057, min = 0.66332 t[s] 0.087
D = 9 x,y: 0.4294, 0.7057, min = 0.70837 t[s] 0.084



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.

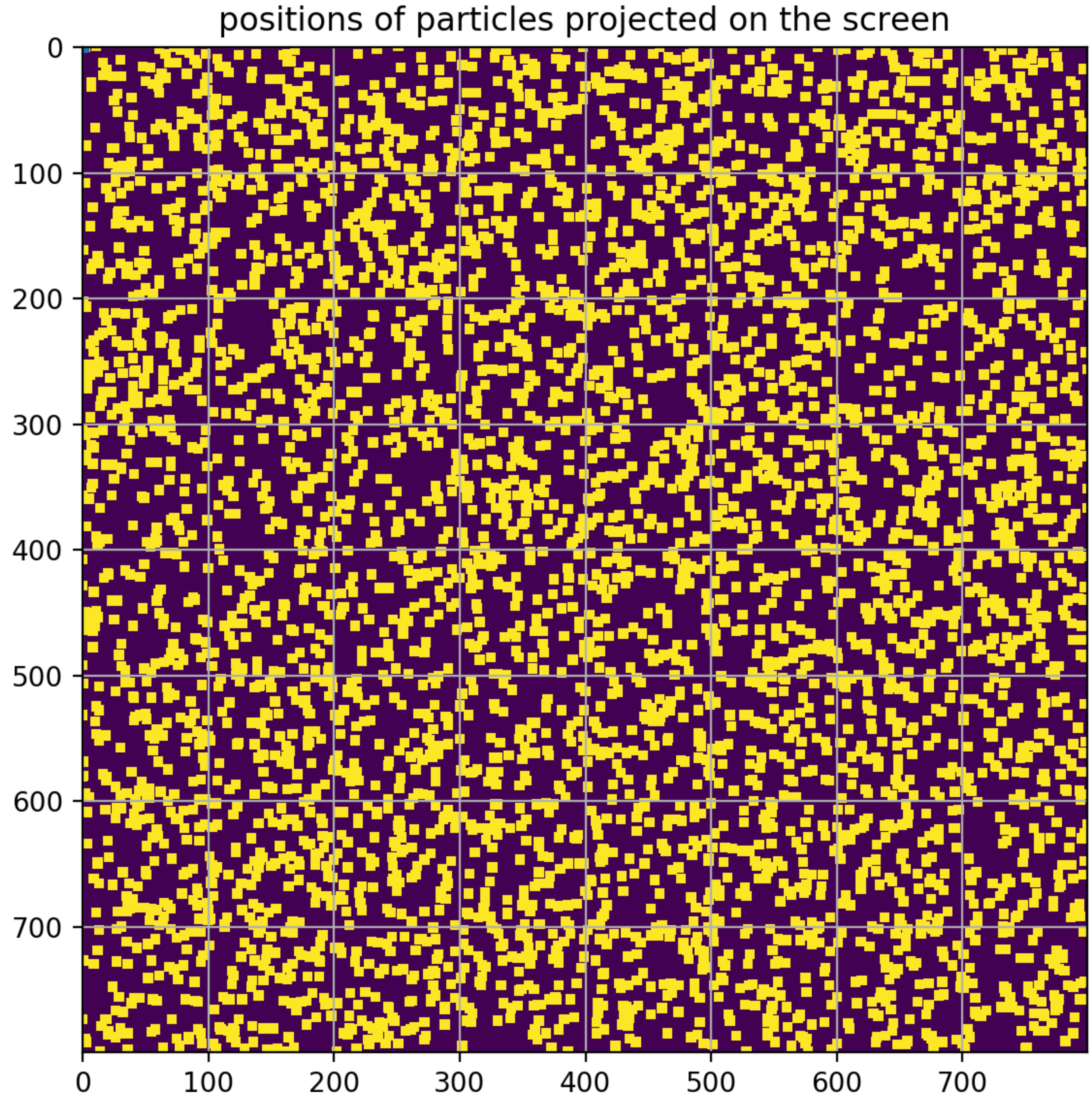


$d\tau := \sigma\, n(x)\, dx$ ➔ $\tau(x) = \text{integral}_0^x \{ \sigma\, n(x)\, dx\}$

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

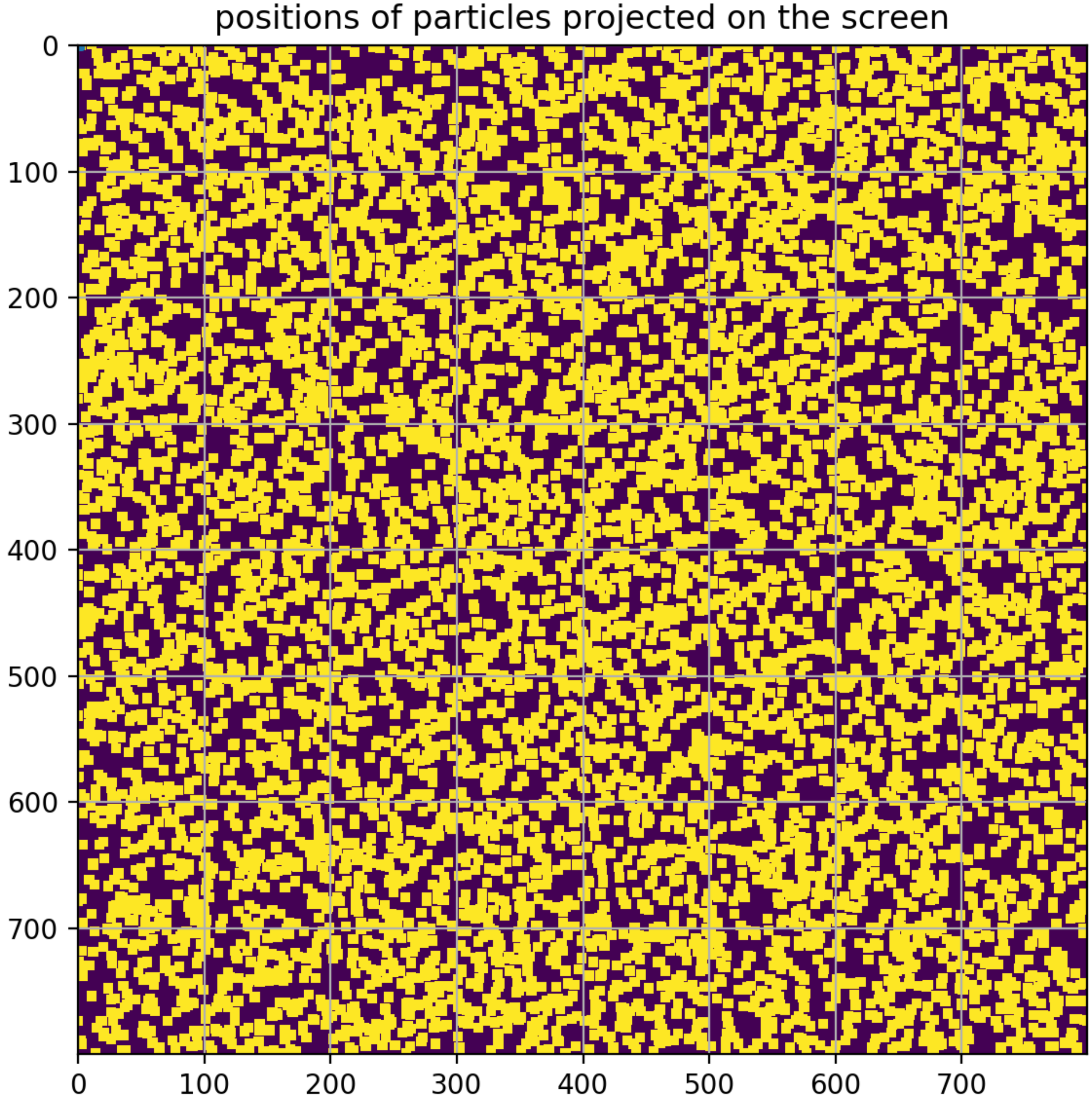
Radiation transfer: Flux vs. optical thickness

tau = 0.47



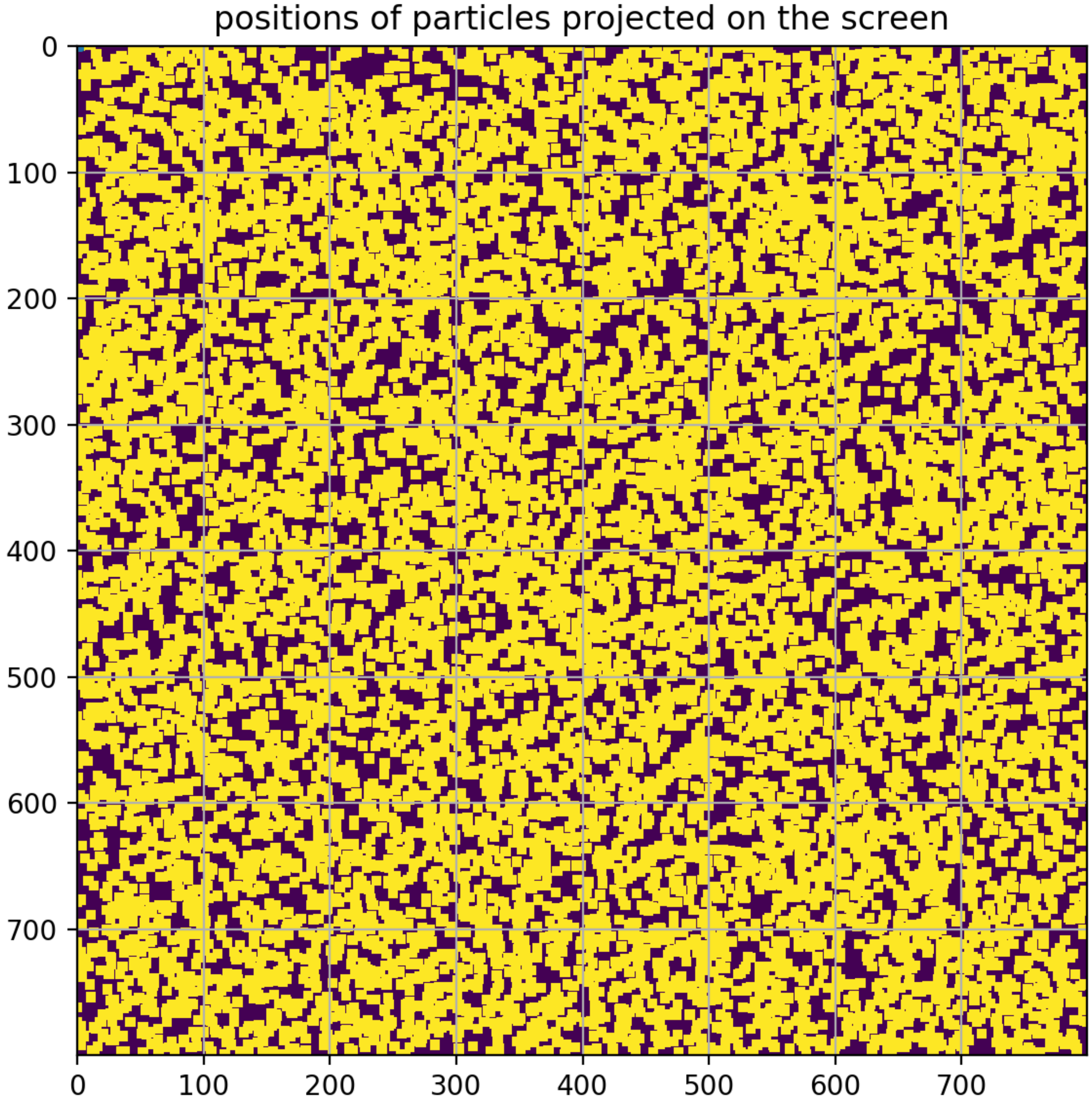
Radiation transfer: Flux vs. optical thickness

tau = 0.84



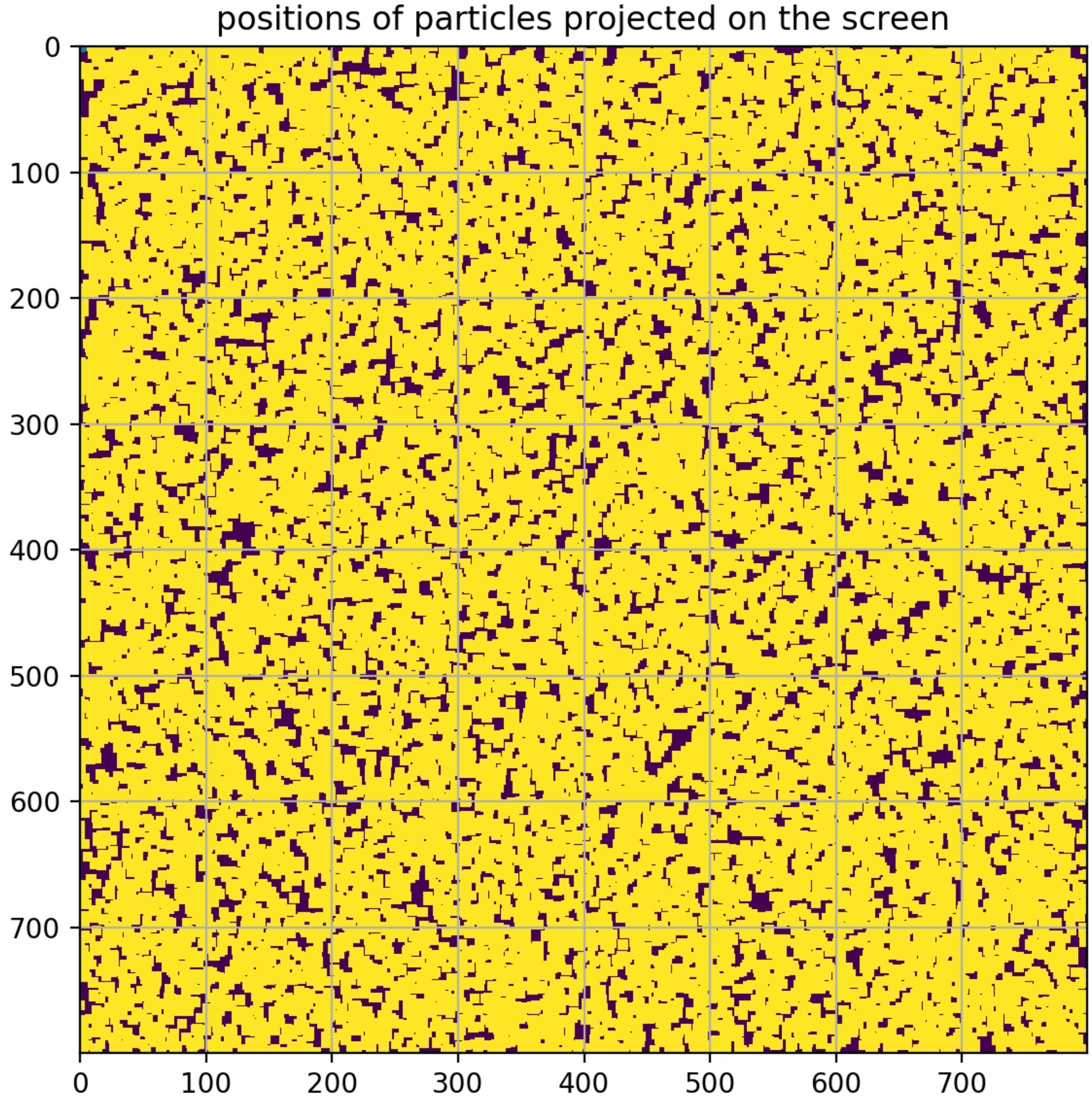
Radiation transfer: Flux vs. optical thickness

tau = 1.22



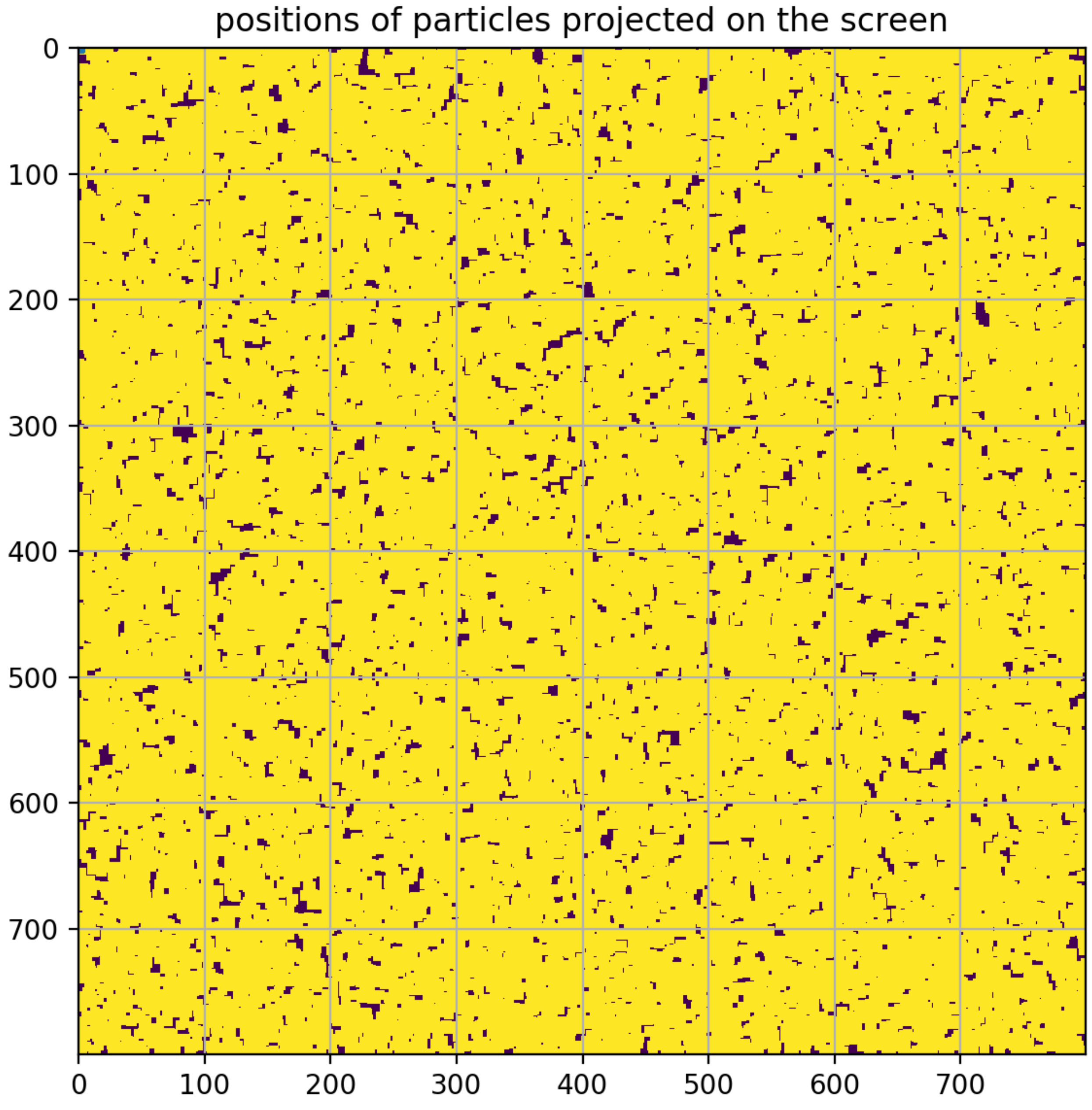
Radiation transfer: Flux vs. optical thickness

tau = 2.0

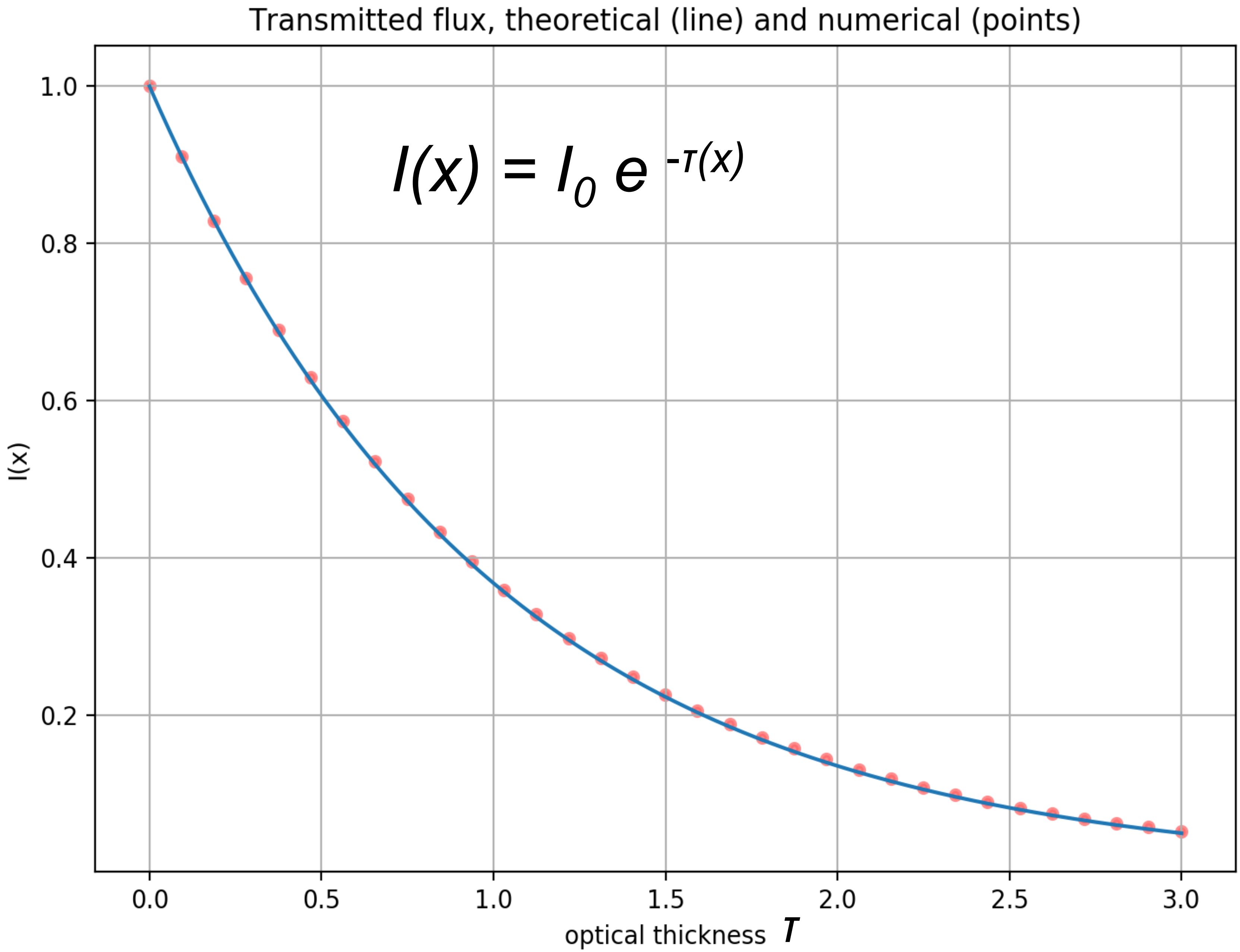


Radiation transfer: Flux vs. optical thickness

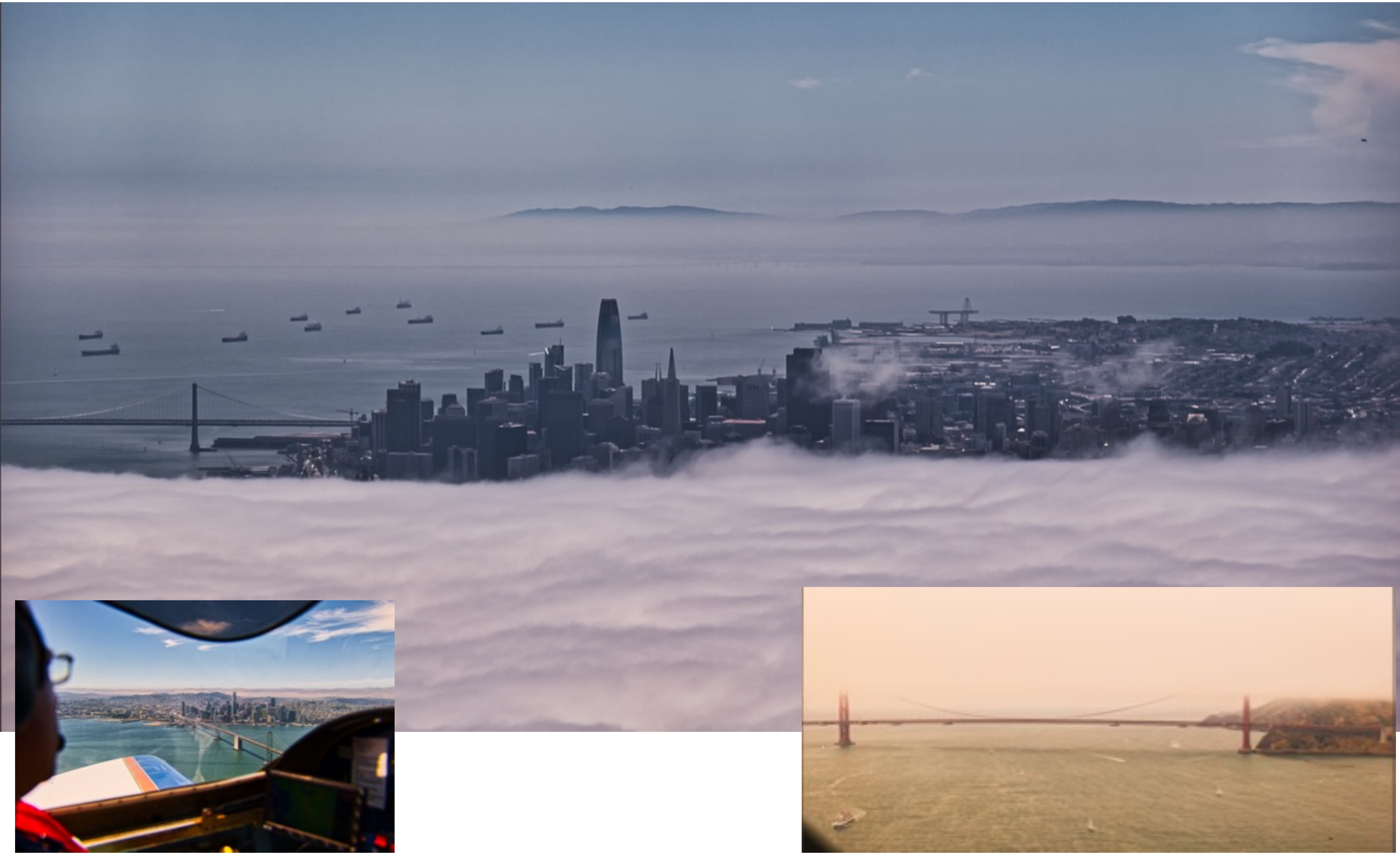
tau = 3.0



dusty-box-1.py



Application of radiation transfer equation



San Francisco (July 2019, picture taken above Alcatraz)

Golden Gate Bridge

An application of radiation transfer equation

- A cloud has geometric thickness of 100 m
- 50% of water vapor mass precipitates into
- droplets of 10 μm diameter (very fine mist)
- 100% relative humidity corresponds to 4.85 g/m^3 at 0^oC.

Find: n, σ, τ, I/I_0

Solution:

In 1 m^3, there are 4.85 g water vapor, 2.425 g in the form of water droplets. At water density 1g/cm^3, their volume is 2.425 cm^3 or

$$V = 2.425e\text{-}6 \ m^3.$$

One droplet has radius r = 5e-6 m, and occupies

$$V_1 = (4/3)\pi r^3 = 5.236e\text{-}16 \ m^3.$$

Python says there are thus

$$n = V/V_1 = 4631517045 \text{ droplets}/m^3.$$

(Q: How many accurate digits should we actually display?)

Application of radiation transfer equation

- $n = 4.63 \cdot 10^9$ droplets/m^3.
- The combined scross-sectional area in a column 100 m long

  (cross section 1 m^2) is

  $\tau = \sigma \; n \, dx = \pi(5e\text{-}6)^2 \cdot 4.63 \cdot 10^9 \cdot 100 \approx 36$

- Optical thickness is $\tau >> 1$, so we don't expect almost any direct light to pass through. Indeed,

  $I/I_0 = e^{-\tau} \approx$ np.exp(-36) $\approx 1.6 \cdot 10^{-16}$

- *You can't see even the weakest outline of the sun or moon through such a cloud!*

Cf. San Francisco shoreline below a stratus layer

- Picture taken near Meteor Crater, AZ (July 2019)
- Smoke from brush fires. Sharp edge of the sun shows that ash particles absorb more then scatter radiation. Visibility of the sun means that optical thickness did not exceed $\tau \sim 5$ to 10.

High optical thickness τ = low transparency

Golden Gate. Fuzzy haloes are a sign that water droplets scatter more photons than absorb.

Application of radiation transfer equation

- $n = V/V_1 \sim 1/r^3$ (s = size of droplet or dust grain)
- $\sigma \sim r^2$
- $\tau = \sigma\ n\ dx \sim 1/r$

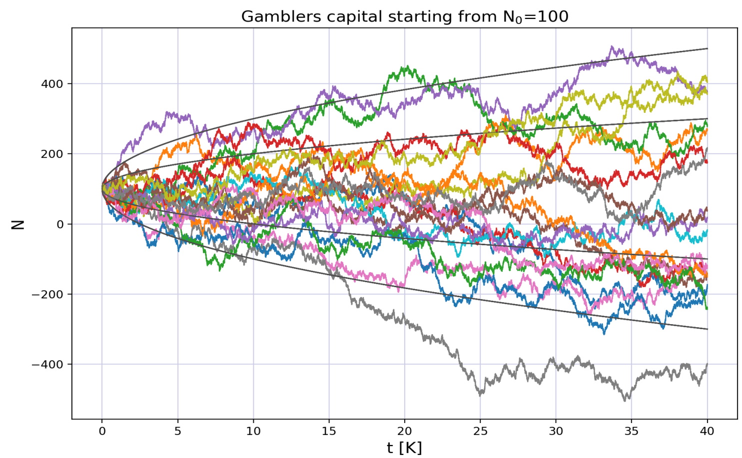
* * *

- Q: How come we sometimes see the outline of the sun through a cloud? Which parameter is different then?
- Hint: Can $\tau \approx 3.6$? When?
- Think of s
- If so, then $I/I_0 = e^{-\tau} \approx \exp(-3.6) \approx 0.026$

➔ no problem seeing direct light from the sun & sun's outline

- Random Walks... and Climate Change?

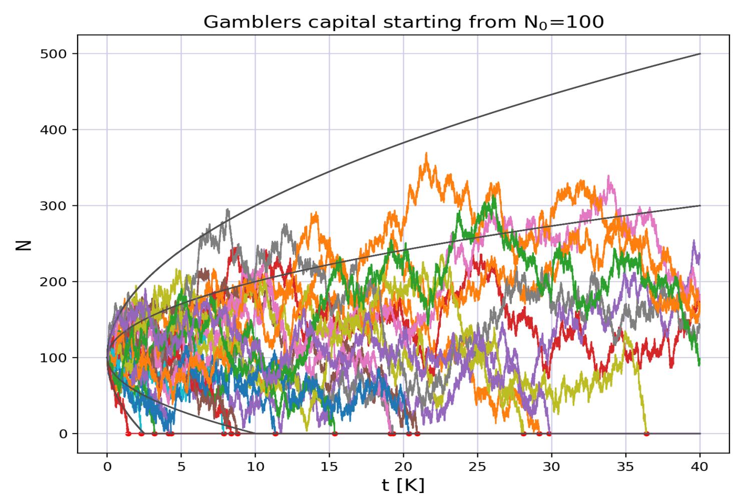
https://wattsupwiththat.com/2012/06/14/climate-models-outperformed-by-random-walks/



- Random walk in 1D

➢ without restrictions

rnd-walk-gamble-0.py

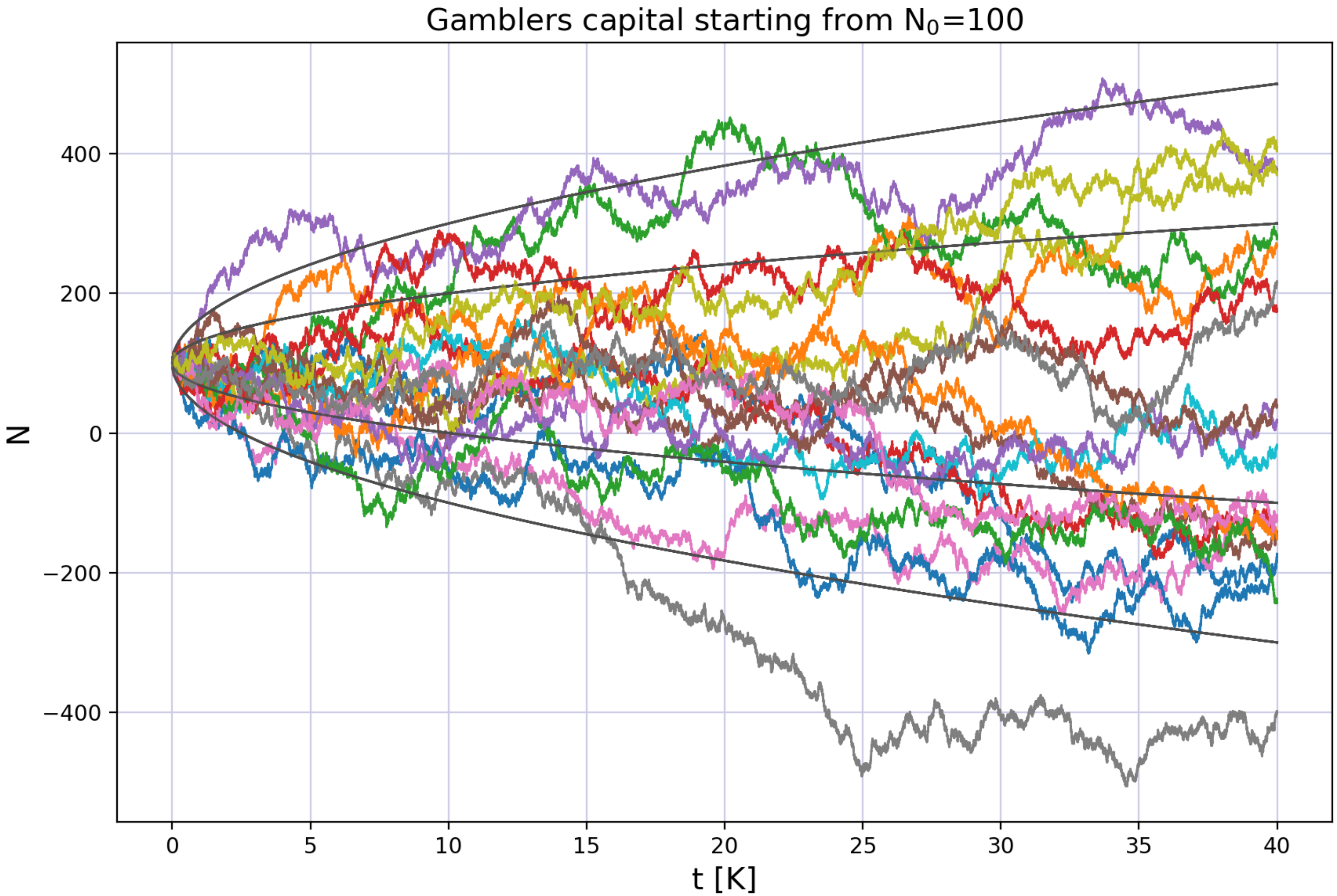


➢ with absorbing boundary(ies)

rnd-walk-gambles.py

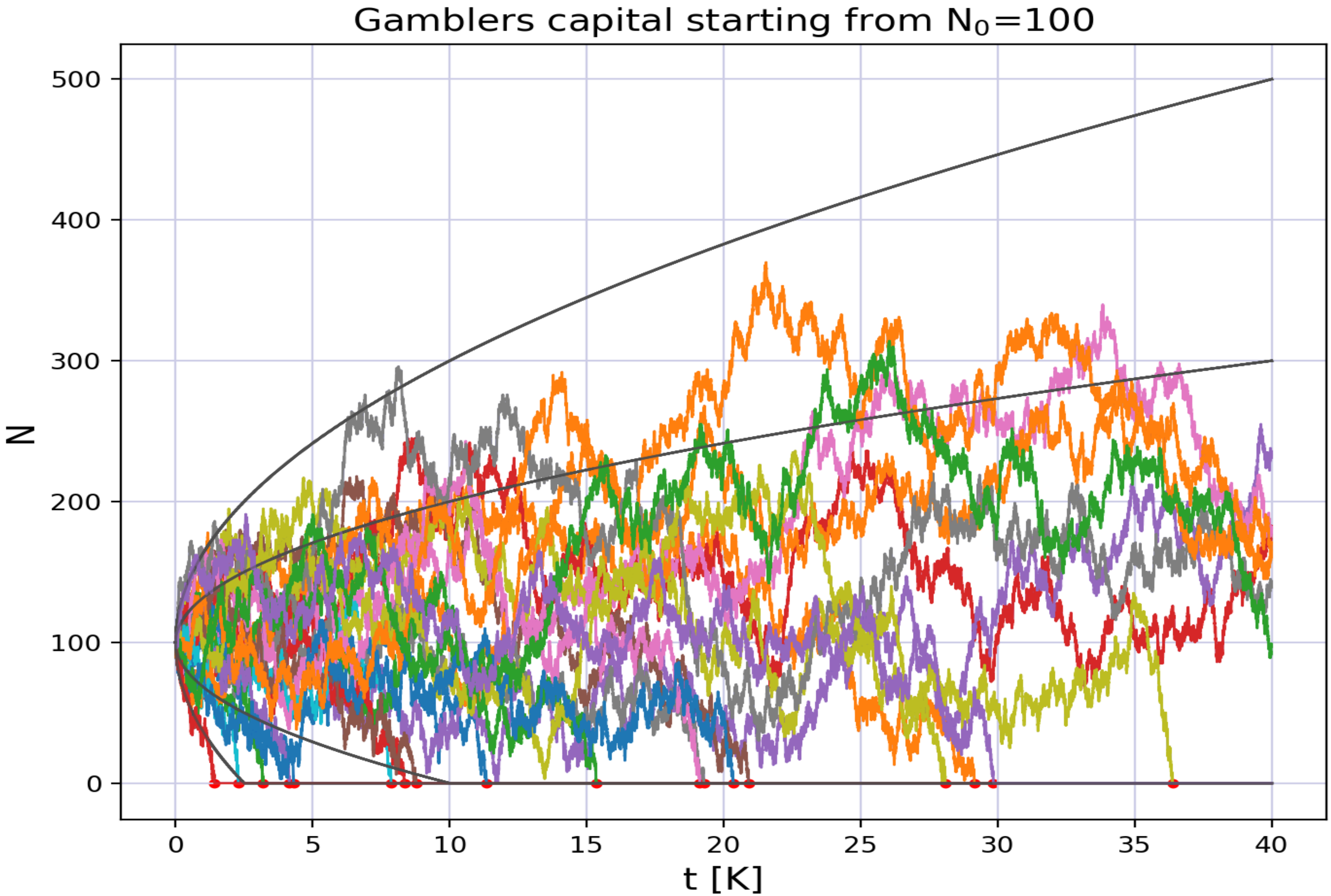
Random walk in 1D without restrictions

rnd-walk-gamble-0.py   +-$\sqrt{N}$ *and* +-2$\sqrt{N}$ *envelopes shown*



Random walk in 1D + absorbing boundary

rnd-walk-gambles.py



Gamblers ruin in 1D walk with two absorbing boundaries at n=M & n=0 can easily be solved analytically:

Let P_n be the probability of a ruin starting from value n (no limitation on number of steps)
Boundary values are: $P_0 = 1$ and $P_M = 0$. (Why?)

From value n we can either continue via n-1 with probability q, or via n+1 with probability 1-q. Therefore the probability of ruin can be expressed as:

$$P_n = q\, P_{n-1} + (1-q)\, P_{n+1}$$

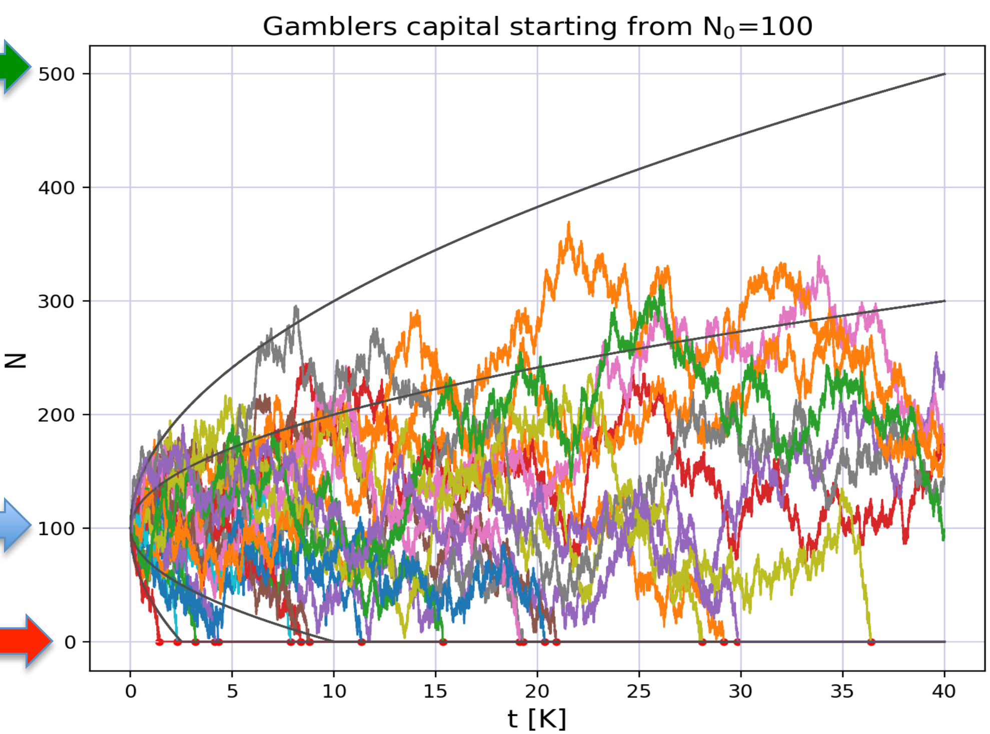
In case of q=1/2 (unbiased random walk in 1D with step +-1), we have

$$P_{n-1} - 2\, P_n + P_{n+1} = 0 \qquad \text{i.e.,} \qquad P_{n-1} - P_n = P_n - P_{n+1}\,.$$

WIN at M

start at n

RUIN



The slopes to the left and right are equal, therefore the function $P(n) = P_n$ is a straight line joining the two boundary values 0 & 1,

$$\mathbf{P_n = (M-n)/M.}$$

In our case $P_{100} = 4/5$.

See also the case M=+∞
rnd_wlk-ret-1.py
on our coding page.

Random Walks in Stock Market?

Full Randomness in Stock Market? The efficient market hypothesis says Yes, apart from long=term trends.

See N.Taleb's book "Fooled by Randomness".

Example: IBM stock price history, last two years: