RANDOM PROCESSES

Ari Maller

RANDOM NUMBERS

- Sometimes in physics we need to make use of random numbers. To do so on the computer we use a random number generator.
- It should be stressed, the computer has no means of making random numbers. To be fair, it isn't entirely clear if I say pick a number between 1 and 100 whether or not we can pick random numbers either.
- Instead the computer can create *pseudorandom* numbers, number that appear random but are generated by a completely deterministic formula.

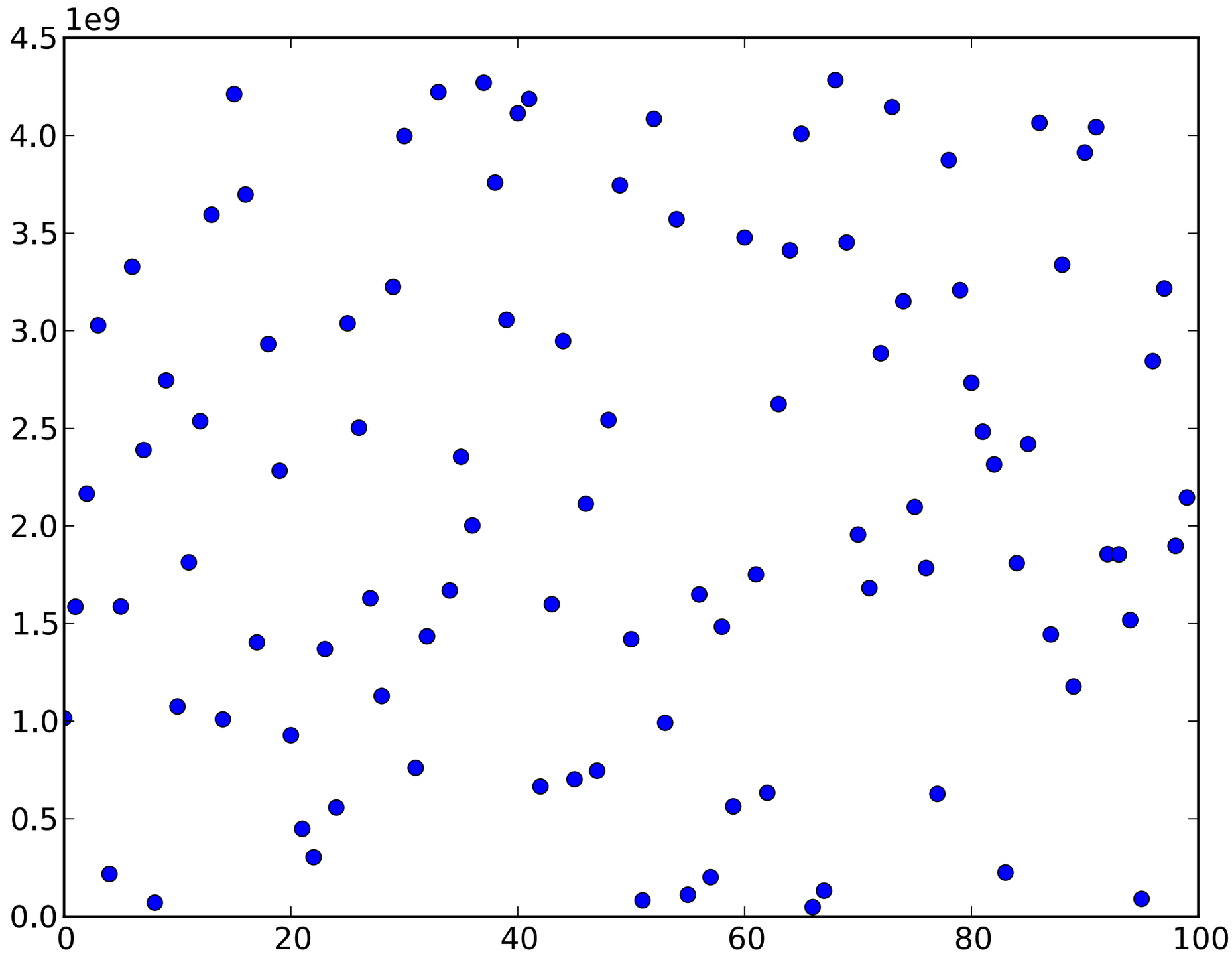
RANDOM NUMBER GENERATORS

- Consider the following equation

$$x' = (ax + c) \mod m$$

- where a, c and m are integer constants and x is an integer variable. Given a value of x the equation returns a value for x'. Now lets take that value of x' and plug it back into the equation.
- If we set a=1664525, c=1013904223 and m=4294967296 and generate values of x' say 100 times, then the resulting integers look pretty random.

The follows generated from that equation. They look pretty random.



RANDOM NUMBER GENERATORS

- This is called a *linear congruential random number generator*. It is probably the most famous of random number generators.
- It is obviously not random, since each value of x' following deterministically from the previous value of x. If you run the program twice it will produce identical results.
- This produces values in the range 0 to m-1. If we want a different range, say 0 to 1 we can divide by m. Note this produces numbers greater than or equal to 0, but always less than 1.
- While the value of a, c and m seem random, they have been chosen with great care. Other choices can produce less random numbers. For example, if c and m are both even, then the process would generate only even or odd numbers depending on the initial x.

RANDOM SEED

- The original number x used to start the process is called the random seed.
- It is important because with it you can produce the same string of random numbers, even on a different computer, if you have the same values of a, c and m.
- This is very useful for debugging. If you have an error that occurs sometimes, but not always with a random number generator, you'd like to be able to produce the same sequence of numbers in order to debug it.
- But if you forget to change the seed, then your random number generator will always produce the same sequence of numbers. That is not behave very randomly.

RANDOM NUMBER GENERATORS

- Unfortunately the random numbers created by the linear congruential random number generator are not random enough for many uses in physics.
- The flaws usually tend to be the sequence of the random numbers is not random enough. That is the ordering the random numbers occur tends to have nonrandom patterns that can introduce correlations in one's analysis.
- Luckily many other random number generators exist. In physics these days the algorithm of choice tends to be the *Mersenne twister*, which is a 'generalized feedback shift-register generator'.

RANDOM PACKAGE

- In Python one can access this algorithm from the random package numpy.random. In order to start generating random numbers one wants to create an instance of a random number generator

```
rng = default_rng(seed=1234) #creates an instance of the generator
rng.random(size=2)           #Returns 2 random float f, where 0 <= f < 1
rng.integers(low=0,high=10,size=5)  #Returns 5 integers between 0 and 10
rng.choice(a)          #returns a random choice from the array a
```

- The package also contains functions that return numbers from distributions other than the uniform distribution.
- The same seed will give the same pseudorandom order of numbers.
- There are many more methods for the generator, see here

https://numpy.org/doc/stable/reference/random/generator.html#random-generator

RANDOM NUMBERS AND CRYPTOGRAPHY

- Psuedorandom numbers play an important role in daily life through there use in cryptography.
- Imagine one of the simplest types of codes a *substation cipher*. In this code one converts all the letters to numbers and then adds a constant to each letter. Converting back to letters one has an unreadable message.
- However, if you know the constant added then you can simply subtract that number from each letter and unveil the original message.
- Such a cipher is also very easy to break. Just subtract everyone possible number between 1 and 26 from the message and one of them will be the correct unencrypted message.

RANDOM NUMBERS AND CRYPTOGRAPHY

- A much more secure cipher is to add a different number to each letter. One can choose the numbers to be added randomly, in this case trying to undo the cipher amounts to randomly changing each letter, which will just generate random messages.
- The most secure way to implement this cipher is with a *one time pad*. That is a list of random numbers that are used by the encoder and decoder only once. While very secure this is difficult to implement.
- More practical is to use a pseudorandom number generator. Then if the only the seed (called the key) is shared the message can be encoded and decoded. And this is what is essentially done for all modern cryptography.

PROBABILITY

- So now that we know how to generate pseudorandom numbers, what do we do with them?
- One use is to generate events based on the probability that such an event would occur. For example, video poker games generate a poker hand as if the cards were randomly chosen from a deck. This is done with a random number generator.
- Similarly many games have random elements, in all those cases the random events are chosen by probability and a random number generator.
- In physics too, if we know something has some probability of occurring, then we create a realization of it using random numbers.

EXAMPLE 10.1 DECAY OF AN ISOTOPE

- The radioisotope ^{208}Tl decays to stable ^{208}Pb with a half life of 3.053 minutes. Suppose we start with a sample of 1000 thallium atoms. Let us simulate the decay of these atoms over time, mimicking the randomness of that decay using random numbers.

- On average we know that the number N of atoms in our sample will fall off exponentially over time according to the standard equation of radioactive decay: $N(t) = N(0)2^{-t/\tau}$

- where τ is the half-life. Then the fraction of atoms remaining after time t is $N(t)/N(0) = 2^{-t/\tau}$ and the fraction that have decayed also equal to the probability of a single atom decaying is

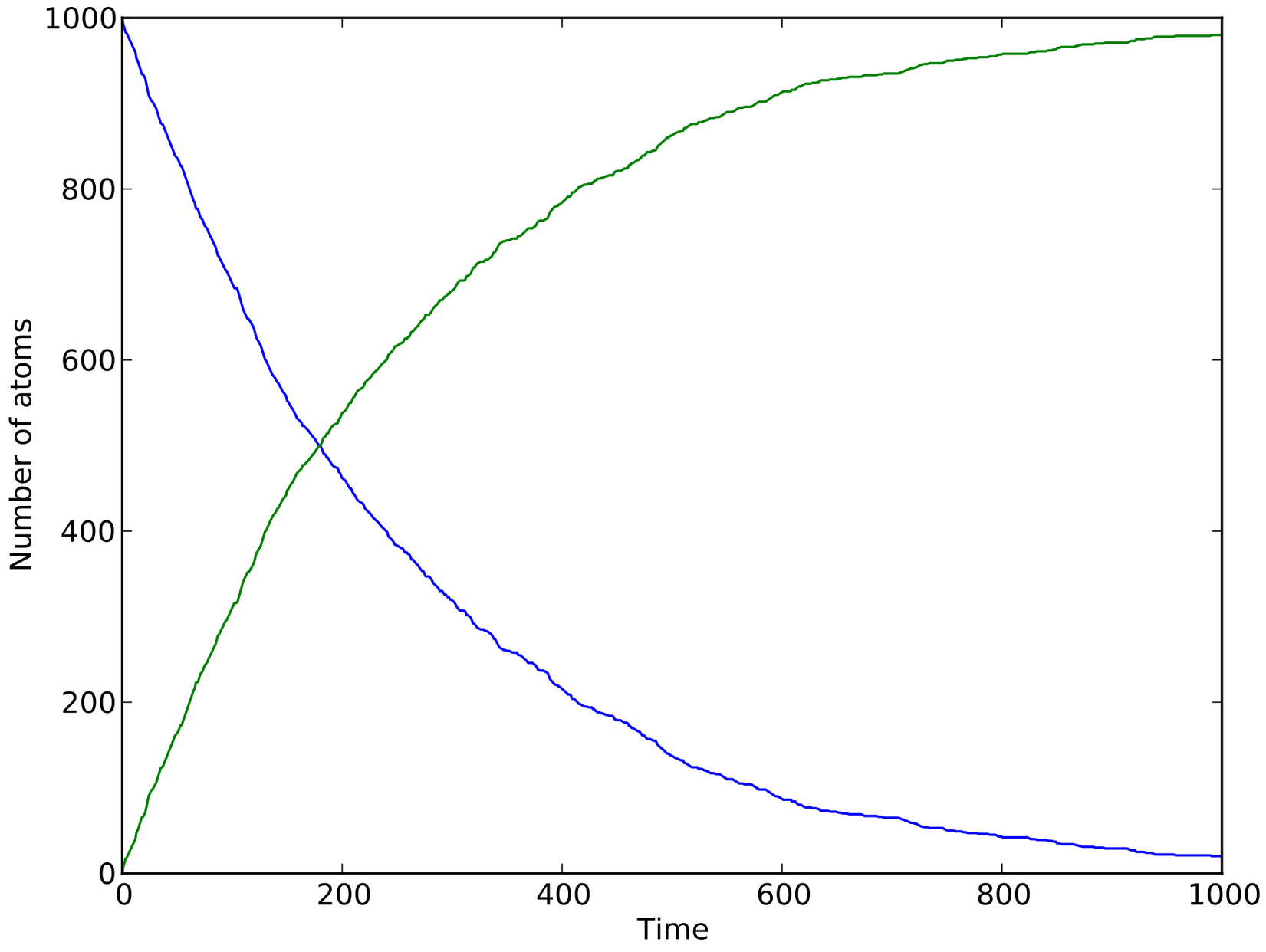
$$p(t) = 1 - 2^{-t/\tau}$$

EXAMPLE CODE (DECAY.PY)

```
from random import random
from numpy import arange
from pylab import plot,xlabel,ylabel,show

# Constants
NTl = 1000          # Number of thallium atoms
NPb = 0             # Number of lead atoms
tau = 3.053*60      # Half life of thallium in seconds
h = 1.0             # Size of time-step in seconds
p = 1 - 2**(-h/tau)   # Probability of decay in one step
tmax = 1000         # Total time

# Lists of plot points
tpoints = arange(0.0,tmax,h)
Tlpoints = []
Pbpoints = []
```

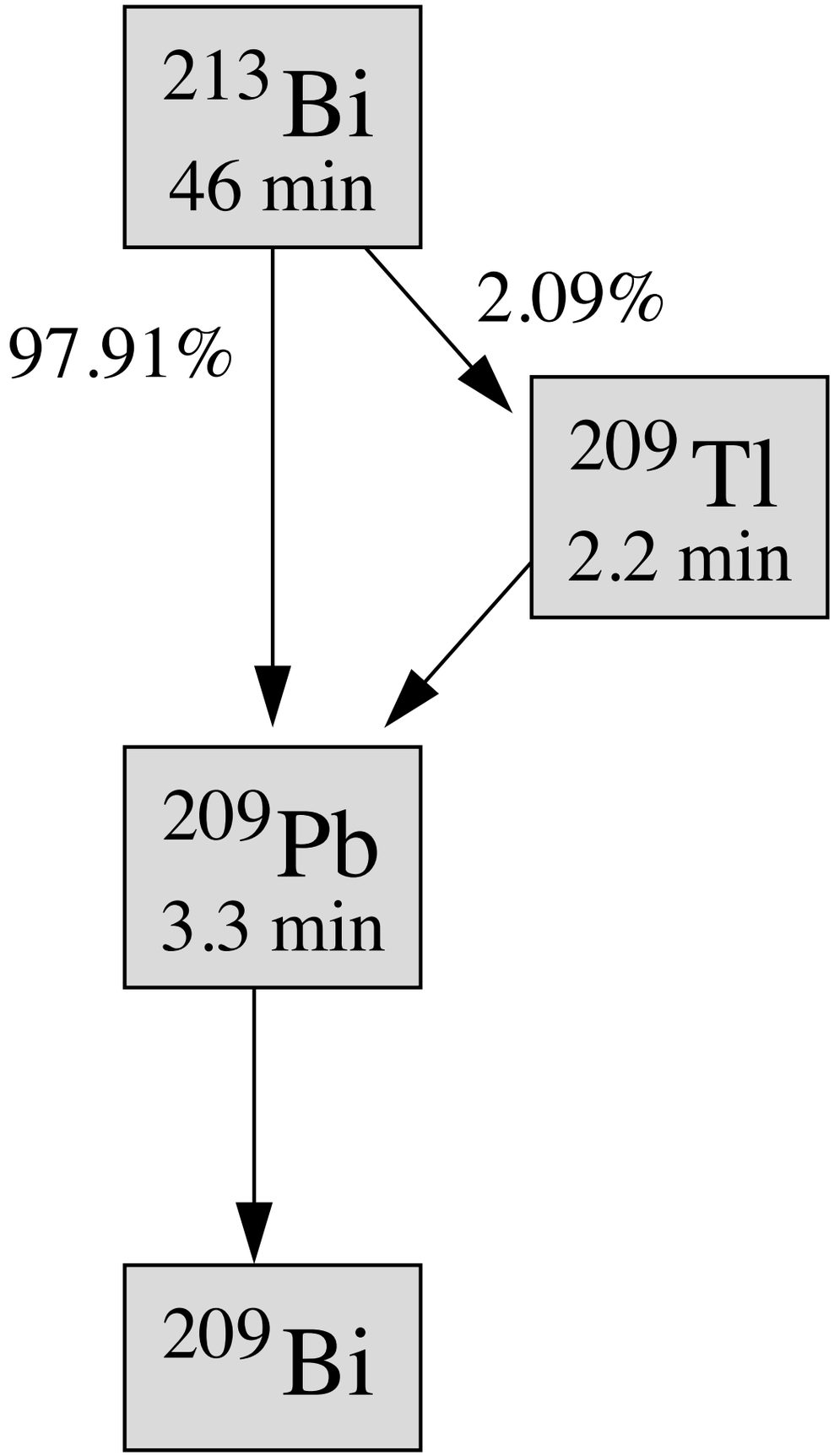
EXAMPLE CODE (DECAY.PY)

```
# Main loop
for t in tpoints:
    Tlpoints.append(NTl)
    Pbpoints.append(NPb)

    # Calculate the number of atoms that decay
    decay = 0
    for i in range(NTl):
        if random()<p:
            decay += 1
    NTl -= decay
    NPb += decay

# Make the graph
plot(tpoints,Tlpoints)
plot(tpoints,Pbpoints)
xlabel("Time")
ylabel("Number of atoms")
show()
```



NUMERICAL EXPERIMENTS

- Notice that the plot we made is not the average decay rate of the isotope. We could simply plot that since we have N(t). But the two are different.
- The plot is a random realization of a possible decay path for the isotope. Running the code a second time will give different results (since we did not set a seed).
- In some ways using random numbers can be thought of as a numerical experiment. Just like an experimental set up, it is one realization of the possible outcomes.
- Just like running many experiments will have variations around the mean value, each numerical experiment will also show variations. This can be an advantage as we will discuss later.



$$p(t) = 1 - 2^{-t/\tau}$$

EXERCISE 10.2

- The isotope ^{213}Bi decays to stable ^{209}Bi via one of two different routes, with probabilities and half-lives as seen in the figure.
- Starting with a sample consisting of 10,000 atoms of ^{213}Bi, simulate the decay of the atoms as in Example 10.1 by dividing time into slices of length $\delta t = 1$s each and on each step doing the following:

1. For each atom of ^{209}Pb in turn, decide at random, with the appropriate probability, whether it decays or not. Count the total number that decay, subtract it from the number of ^{209}Pb atoms, and add it to the number of ^{209}Bi atoms.
2. Now do the same for ^{209}Tl, except that decaying atoms are subtracted from the total for ^{209}Tl and added to the total for ^{209}Pb.
3. For ^{213}Bi the situation is more complicated: when a ^{213}Bi atom decays you have to decide at random with the appropriate probability the route by which it decays. Count the numbers that decay by each route and add and subtract accordingly.

- Note that you have to work up the chain from the bottom like this, not down from the top, to avoid inadvertently making the same atom decay twice on a single step. Keep track of the number of atoms of each of the four isotopes at all times for 20,000 seconds and make a single graph showing the four numbers as a function of time on the same axes.

NONUNIFORM RANDOM NUMBERS

- The random number generators we have discussed return uniform probability integers in a range or floats in the range of zero up to one. But many probability distributions aren't uniform.
- Consider the radioactive decay we just discussed. The probability of a decay in a small interval dt is

$$1 - 2^{-dt/\tau} = 1 - \exp\left(\frac{dt}{\tau}\ln 2\right) = \frac{\ln 2}{\tau}dt$$

- Now we can ask what is the probability that a particular atom survives until time t and then decays between t and t+dt.

$$P(t)dt = 2^{-t/\tau}\frac{\ln 2}{\tau}dt$$

NONUNIFORM RANDOM NUMBERS

- This is an example of a nonuniform probability distribution. The chance of decay is not uniform but decreases exponentially with time.
- In the example we just did we calculated the probability of an isotope decaying every time step. However, it would be much more efficient to just calculate a time at which each atom decays. But to do so we need to sample from this nonuniform probability distribution.
- Fortunately it turns out this is not too hard to do. There are a number of methods, the most common is the *transformation method*.

TRANSFORMATION METHOD

- Suppose you have a probability density q(z) and supposed you have a function x(z). If z is a random number than x(z) is also a random number, but in general it doesn't have the distribution q(z) it has a distribution p(x). Our goal is to choose x(z) so it has the distribution we want.

- The probability of generating a value of x between x and x+dx is by definition equal to the probability of generating a value of z in the corresponding interval.

$$p(x)dx = q(z)dz$$

- Let's take q(z) to be the uniform distribution from 0 to 1.

TRANSFORMATION METHOD

- Then $$\int_{-\infty}^{x(z)} p(x')dx' = \int_0^z dz' = z$$
- if we can do this integral and solve the resulting equation for x then we will have an expression that will give us what we want. Let's try an example of an exponential pdf.

$$p(x) = \mu e^{-\mu x}$$

- where the leading factor of μ is for normalization. Then plugging into the top equation we get

$$\int_{-\infty}^{x(z)} e^{-\mu x} dx' = 1 - e^{-\mu x} = z$$

- which we can solve for x to get

$$x = -\frac{1}{\mu} \ln(1 - z)$$

EXERCISE 10.4

- Redo the calculation from Example 10.1, but this time using the faster method just described. Using the transformation method, generate 1000 random numbers from the nonuniform distribution to represent the times of decay of 1000 atoms of ^{208}Tl (which has half-life 3.053 minutes). Then make a plot showing the number of atoms that have not decayed as a function of time, i.e., a plot as a function of t showing the number of atoms whose chosen decay times are greater than t.

- Hint: You may find it useful to know that the package numpy contains a function sort that will rearrange the elements of an array in increasing order. That is, "b = sort(a)" returns a new array b containing the same numbers as a, but rearranged in order from smallest to largest.

GAUSSIAN RANDOM NUMBERS

- The most common probability distribution is probably the normal or Gaussian distribution. What if we want to get random numbers from this distribution?

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

- The transformation method fails in this case because we don't know how to do the integral. Instead imagine we have two independent random variable x and y, both drawn from the same Gaussian distribution. The probability density of the point x,y is given by

$$p(x)dxp(y)dy = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right)dx \times \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{y^2}{2\sigma^2}\right)dy$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)dxdy$$

GAUSSIAN RANDOM NUMBERS

- We can now switch to polar coordinates and our integral becomes

$$p(r,\theta)drd\theta = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\big(-\frac{r^2}{2\sigma^2}\big)rdrd\theta = \frac{1}{\sqrt{\sigma^2}}\exp\big(-\frac{r^2}{2\sigma^2}\big)rdr \times \frac{d\theta}{2\pi}$$

- which is just p(r)dr ×p(θ)dθ. The second distribution is just the uniform distribution. The p(r) part we can now solve using the transformation method.

$$\frac{1}{\sqrt{\sigma^2}}\int_0^r \exp\big(-\frac{r'^2}{2\sigma^2}\big)r'dr' = 1-\exp\big(\frac{r^2}{2\sigma^2}\big) = z$$

- which gives

$$r = \sqrt{-2\sigma^2\ln(1-z)}$$

- To get your Gaussian random variables choose z and θ from a uniform distribution, determine r from the above equation and then

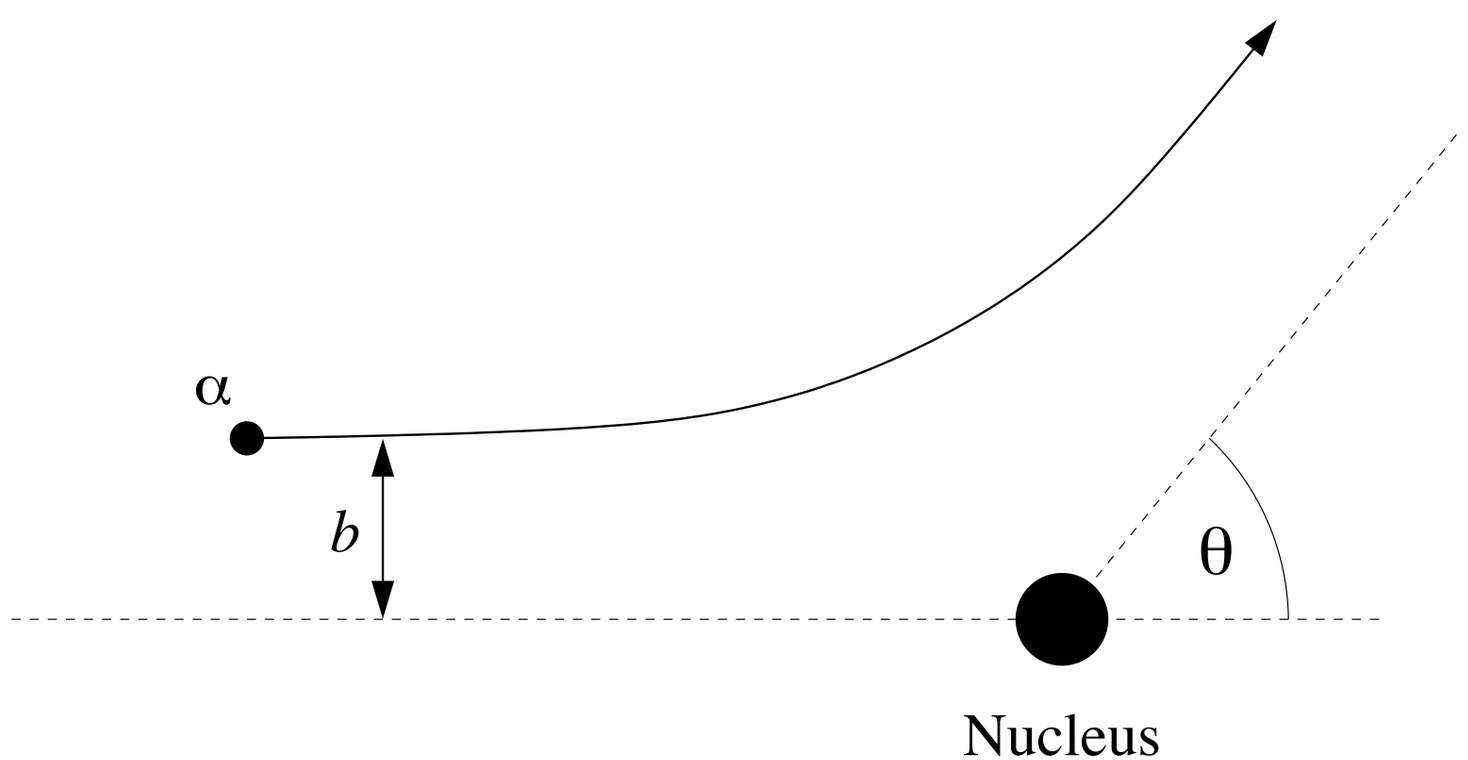
$$x = r\cos(\theta) \qquad y = r\sin(\theta)$$

EXAMPLE 10.2 RUTHERFORD SCATTERING

- At the beginning of the 20th century Rutherford and collaborators showed that a positively charged particle passing by at atom scatters by an angle θ given by

$$\tan\frac{1}{2}\theta = \frac{Ze^2}{2\pi\epsilon_0 Eb}$$

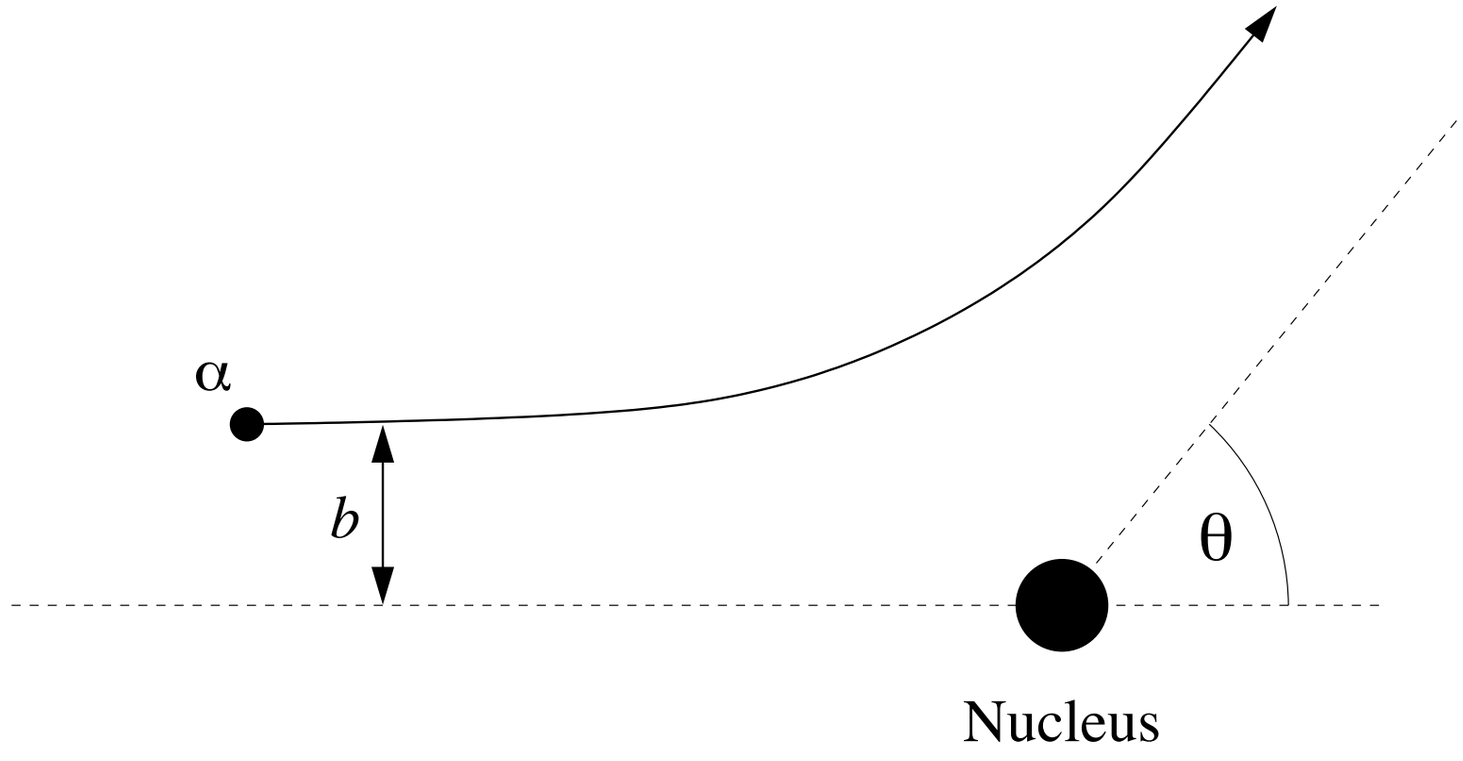
- where Z is the atomic number
- and b is the impact parameter.



EXAMPLE 10.2 RUTHERFORD SCATTERING

- Let us consider a beam of particles that has a Gaussian profile in x and y with $\sigma = a_0/100$ and a_0 is the Bohr radius.
- Let's simulate the scattering of one million particles and calculate the number that 'bounce back', have $\theta > 90$.
- For those b must be less than

$$b = \frac{Ze^2}{2\pi\epsilon_0 E}$$



```
from math import sqrt,log,cos,sin,pi
from random import random

# Constants
Z = 79
e = 1.602e-19
E = 7.7e6*e
epsilon0 = 8.854e-12
a0 = 5.292e-11
sigma = a0/100
N = 1000000

# Function to generate two Gaussian random numbers
def gaussian():
    r = sqrt(-2*sigma*sigma*log(1-random()))
    theta = 2*pi*random()
    x = r*cos(theta)
    y = r*sin(theta)
    return x,y

# Main program
count = 0
for i in range(N):
    x,y = gaussian()
    b = sqrt(x*x+y*y)
    if b<Z*e*e/(2*pi*epsilon0*E):
        count += 1

print(count,"particles were reflected out of",N)
```

When run the program will return something like

1549 particles were reflected out of 1000000

or about 0.15%. Rutherford was amazed by this result which led to the discovery of atomic nuclei.

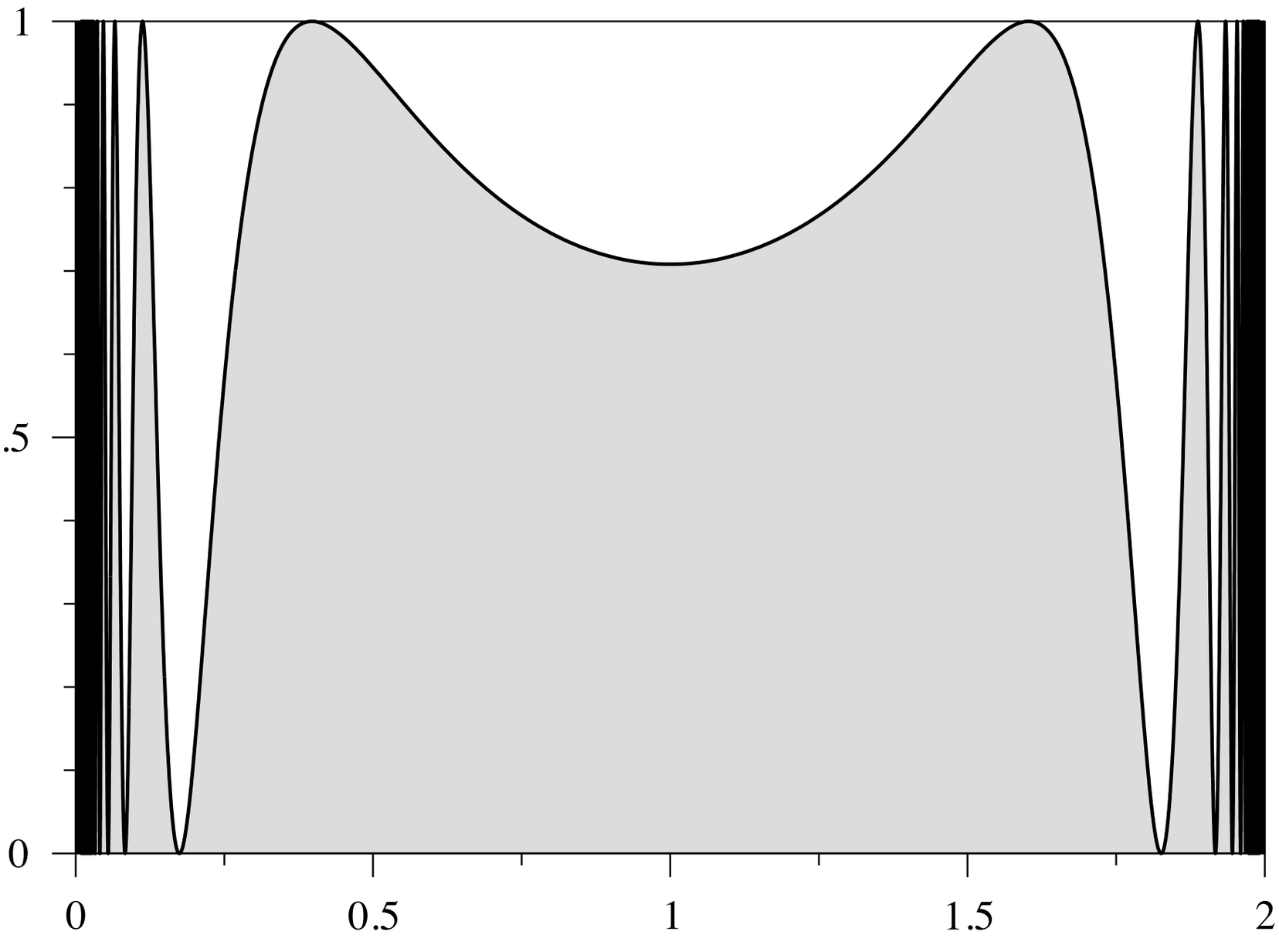
MONTE CARLO INTEGRATION

- We could have done the previous exercise analytically. Starting with the distribution being Gaussian we are just looking for those particles with r < b. So

$$\frac{1}{\sigma^2}\int_0^b \exp\left(-\frac{r^2}{2\sigma^2}\right) r dr = 1 - \exp\left(-\frac{b^2}{2\sigma^2}\right)$$

- plugging in the values from the example to get b gives a reflected percentage of 0.156% in good agreement with our calculation from random numbers.
- But now think of this the other way around. If our random calculation gives the same result of an integral, that means we can evaluate integrals via random calculations.

- Suppose we want to evaluate the integral $I = \int_0^2 \sin^2\left[\frac{1}{x(2-x)}\right] dx$
- This function is shown in the figure below.
- It is rather pathological, it varies infinitely fast at the edges, though it is well behaved in the middle.



- On the other hand since the function fits in the box we know its integral is less than 2.
- The integration methods we have learned about will perform poorly because of the edges.

MONTE CARLO INTEGRATION

- One way to evaluate this integral is to choose a large number of random points in the square. The probability that a point will fall below the function is just p=I/A.
- Thus if we generate N points and k fall below the curve we estimate the probability p=k/N, which implies the value of our integral I = kA/N.
- This process goes by the name Monte Carlo Integration, named after the casino. It uses random processes to evaluate a definite integral.
- Monte Carlo Integration is particularly useful for pathological integrands. Since we perform no extrapolation of the function, we don't get caught up in the pathological behavior.

https://commons.wikimedia.org/wiki/File:Pi_30K.gif#/media/File:Pi_30K.gif

```
from math import sin
from random import random

def f(x):
    return (sin(1/(x*(2-x))))**2

N = 10000
count = 0
for i in range(N):
    x = 2*random()
    y = random()
    if y<f(x):
        count += 1
I = 2*count/N
print(I)
```

MONTE CAROL INTEGRATION

- The main disadvantage of Monte Carlo integration is that it isn't very accurate. If we can use the trapezoid method or other standard methods they will be much more accurate.
- To see how accurate Monte Carlo integration is we can consider the chance a point is below the curve. p= I/A. The probability it falls above is then 1-p. Thus the probability we get exactly k points below the curve is

$$P(k) = \binom{N}{k} p^k (1-p)^{N-k}$$

- the standard deviation in the binomial distribution is

$$\sigma_k = \sqrt{\frac{I(A-I)}{N}}$$

- the accuracy improves as $N^{-1/2}$, which is not very fast.

MONTE CAROL INTEGRATION

- In contrast remember that the error in the trapezoid rule was of order h^2. Since $h = (b-a)/N$, this is N^{-2}. The Simpson's rule where the error went at h^4 would be N^{-4}.
- Put another way a Monte Carlo integral with 100 sample points would have an accuracy of ~10%. The same number of points in the trapezoid method would have an accuracy of 0.01%, while Simpson's method would be 0.0001%. Clearly, one should only use the Monte Carlo method as a last resort.
- However, the method we have been discussing is not a very good Monte Carlo method for performing integration. A much better method is called the *mean value method*.

MEAN VALUE METHOD

- Let us look at a general integral $$I = \int_a^b f(x)dx$$
- The average value of f(x) in that range is then
$$< f >= \frac{1}{b-a}\int_a^b f(x)dx = \frac{I}{b-a}$$
- So I = (b-a)<f>. Which means if we can estimate <f> we can evaluate I. A simple way to estimate <f> is just to measure f(x) at N points chosen uniformly at random between a and b. So we get that
$$I \simeq \frac{b-a}{N}\sum_{i=1}^{N} f(x_i)$$
- This is the fundamental formula for the mean value method.

MEAN VALUE METHOD

- How accurate is this method? We can estimate the error using standard results for the behavior of random variables.
- The variance of f, var f = <f^2> - <f>^{2}. Where

$$< f >= \frac{1}{N}\sum_{i=1}^{N} f(x_i) \qquad < f^2 >= \frac{1}{N}\sum_{i=1}^{N} |f(x_i)|^2$$

- so the standard deviation on the integral from the mean value method is

$$\sigma' = \frac{b-a}{N}\sqrt{N \operatorname{var} f} = (b-a)\sqrt{\frac{\operatorname{var} f}{N}}$$

- so the error once again goes as N$^{-1/2}$. The leading constant is smaller in this case so the error will be smaller then our previous 'hit-or-miss' method, even if it doesn't continue to become smaller with larger N.

EXERCISE 10.5

➤ Write a program to evaluate the integral below using the 'hit-or-miss' method with 10,000 points. Also evaluate the error.

$$I = \int_0^2 \sin^2 \left[\frac{1}{x(2-x)}\right] dx$$

➤ Now estimate the integral again using the mean value method. Again estimate your error.

$$\sigma_k = \sqrt{\frac{I(A-I)}{N}}$$

$$\sigma' = \frac{b-a}{N}\sqrt{N \operatorname{var} f} = (b-a)\sqrt{\frac{\operatorname{var} f}{N}}$$

INTEGRATION IN MANY DIMENSIONS

- In addition to the integration of pathological functions, Monte Carlo integration is used to evaluate high dimensional integrals.
- For our previous integration methods, performing those techniques over more dimensions required us to sample values of the integrand over a grid. If we had four dimensions and we used 100 points in each dimension that would be 100^4 or 100 million points.
- This is doable, but rather slow and higher dimensions quickly become undoable. Monte Carlo integration can give reasonably good answers with far fewer points.

INTEGRATION IN MANY DIMENSIONS

- Generalizing the mean value method is straightforward. Now we have

$$I \simeq \frac{V}{N} \sum_{i=1}^{N} f(\mathbf{r_i})$$

- where $\mathbf{r}_i$ is now a vector that is chosen randomly from the volume to be integrated over.

- One area where this method is the standard is financial mathematics. Predicting the value of a portfolio requires integrating over many variables. In order to make a quick trading decision, one doesn't want to wait a long time (seconds) to perform the calculation. Thus Monte Carlo integration is the standard for that kind of analysis.

IMPORTANCE SAMPLING

- While Monte Carlo integration is a good approach for pathological functions, there are still some functions where it doesn't preform well. Particularly, cases where the integrand contains a divergence. For example

$$I = \int_0^1 \frac{x^{-1/2}}{e^x + 1} dx$$

- which arises in the theory of Fermi gasses. While this integrand diverges at x=0, the integral is finite. But if you try to evaluate it with the mean value method you can run into problems because if your sample has an x close to zero it can contribute a very large amount.
- Thus you'll get large variations between different evaluations using this method.

IMPORTANCE SAMPLING

- We can get around this problem by drawing our x point non uniformly from the integration interval. This technique is called *importance sampling*. For any general function g(x) we can define a weighted average over the interval from a to b.

$$< g >_w = \frac{\int_a^b w(x)g(x)dx}{\int_a^b w(x)dx}$$

- if we consider g(x)=f(x)/w(x) where f(x) is the function we want to integrate then

$$\left\langle \frac{f(x)}{w(x)} \right\rangle_w = \frac{\int_a^b f(x)dx}{\int_a^b w(x)dx} = \frac{I}{\int_a^b w(x)dx}$$

- which means $$I = \left\langle \frac{f(x)}{w(x)} \right\rangle_w \int_a^b w(x)dx$$

IMPORTANCE SAMPLING

- This formula is essentially just like the formula for the mean value method, except now we have a weighted average. And how do we calculate this weighted average. Let's convert our weight into a probability.

$$p(x) = \frac{w(x)}{\int_a^b w(x)dx}$$

- Let us sample N random points from this distribution. Then we have

$$\sum_{i=1}^{N} g(x_i) \simeq \int_a^b Np(x)g(x)dx$$

- we can now write the general weighted average of g(x) as

$$\langle g \rangle_w = \frac{\int_a^b w(x)g(x)dx}{\int_a^b w(x)dx} = \int_a^b p(x)g(x)dx \simeq \frac{1}{N}\sum_{i=1}^{N} g(x_i)$$

IMPORTANCE SAMPLING

$$\langle g \rangle_w = \frac{\int_a^b w(x)g(x)dx}{\int_a^b w(x)dx} = \int_a^b p(x)g(x)dx \simeq \frac{1}{N}\sum_{i=1}^{N} g(x_i)$$

- where x_i are chosen from the p(x) distribution. Putting this together with our expression for the integral gives us

$$I \simeq \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{w(x_i)} \int_a^b w(x)dx$$

- this is the fundamental equation of importance sampling. Notice that if $w(x)=1$ this is the mean value method. So what to we gain if $w(x)$ is a different function. If properly chosen we can use $w(x)$ to remove pathologies in $f(x)$. The price we pay for this extra flexibility is that we have to chose x from a nonuniform distribution.

IMPORTANCE SAMPLING

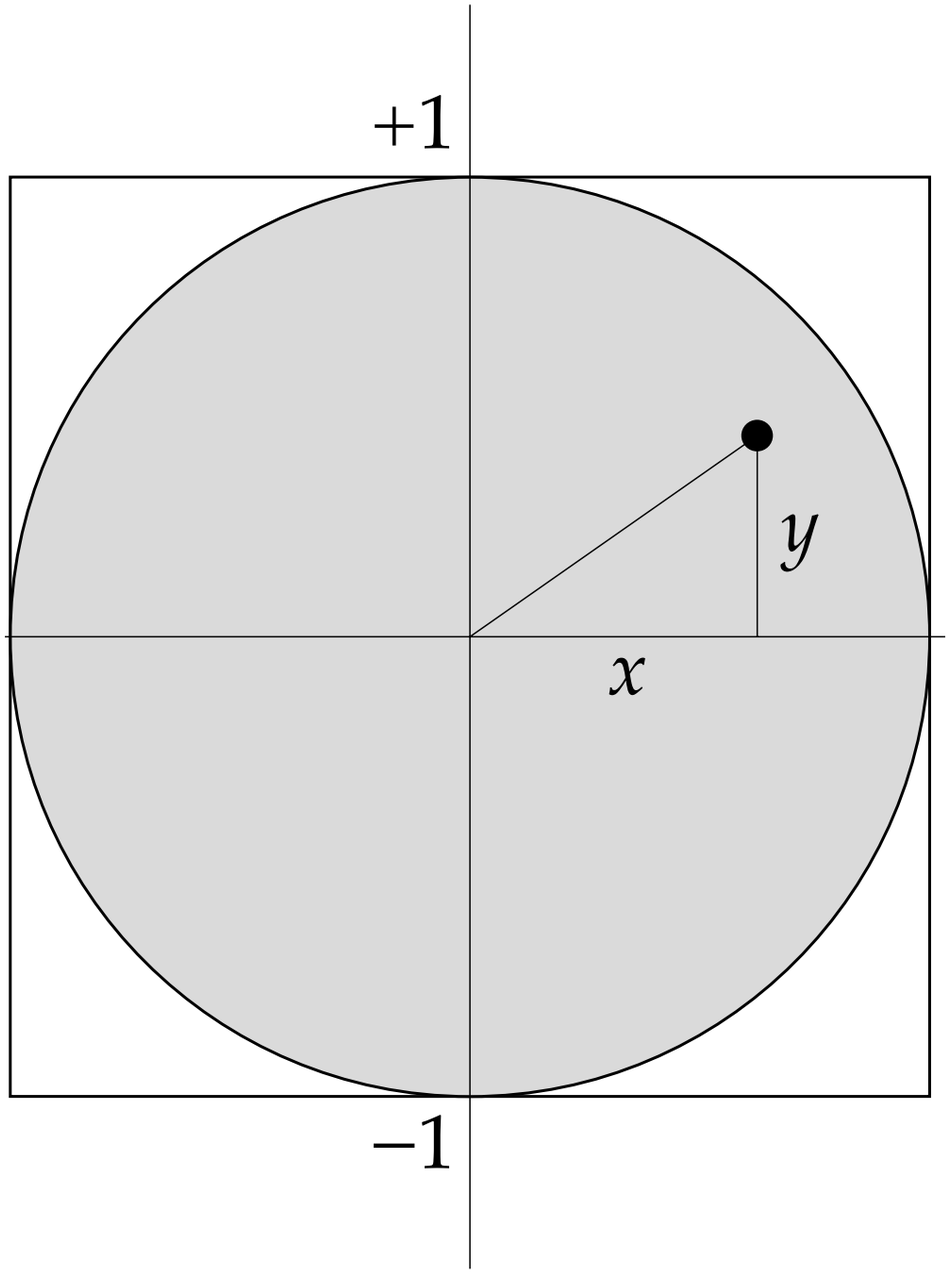
- Let's go back to our original example to see how this works in practice. Our pathological integral is

$$I = \int_0^1 \frac{x^{-1/2}}{e^x + 1} dx$$

- so we choose a weighting of $w(x) = x^{-1/2}$ to remove the divergent behavior. That gives us

$$g(x) = \frac{f(x)}{w(x)} = \frac{1}{e^x + 1} \qquad \text{and} \qquad p(x) = \frac{x^{-1/2}}{\int_0^1 x^{-1/2} dx} = \frac{1}{2\sqrt{x}}$$

- so now we can evaluate the integral taking random points from the $p(x)$ distribution and using $g(x)$ which has no divergent behavior.



$$I \simeq \frac{V}{N}\sum_{i=1}^{N} f(\mathbf{r_i})$$

EXERCISE 10.7 VOLUME OF A HYPERSPHERE

- This exercise asks you to estimate the volume of a sphere of unit radius in ten dimensions using a Monte Carlo method. Consider the equivalent problem in two dimensions, the area of a circle of unit radius: The area of the circle, the shaded area, is given by the integral

$$I = \int\int_{-1}^{1} f(x,y)dxdy$$

- where f (x, y) = 1 everywhere inside the circle and zero everywhere outside. In other words,

$$f(x,y) = 1 \;\text{ if }\; x^2 + y^2 \leq 1$$

- So if we didn't already know the area of the circle, we could calculate it by Monte Carlo integration. We would generate a set of N random points (x, y), where both x and y are in the range from -1 to 1. Then the two-dimensional version of the equation to the left for this calculation would be

$$I \simeq \frac{4}{N}\sum_{i=1}^{N} f(x_i, y_i)$$

- Generalize this method to the ten-dimensional case and write a program to perform a Monte Carlo calculation of the volume of a sphere of unit radius in ten dimensions.

- If we had to do a ten-dimensional integral the traditional way, it would take a very long time. Even with only 100 points along each axis (which wouldn't give a very accurate result) we'd still have $100^{10} = 10^{20}$ points to sample, which is impossible on any computer. But using the Monte Carlo method we can get a pretty good result with a million points or so.

STATISTICAL MECHANICS

- Importance sampling isn't just a fancy trick for Monte Carlo integration, it plays a role in many other problems.
- Statistical mechanics is the branch of physics most closely aligned with Monte Carlo techniques. This is because statistical mechanics is always looking at random processes.
- Let's consider the fundamental problem of statistical mechanics, calculating the average (or expectation) value of a quantity in a physical system in thermal equilibrium. If the systems is at temperature T then it will not be in one state, but will randomly move between different states where the probability of being in any state is given by

$$P(E_i) = \frac{e^{-\beta E_i}}{Z} \qquad Z = \sum_i e^{-\beta Ei}$$

$$P(E_i) = \frac{e^{-\beta E_i}}{Z} \qquad Z = \sum_i e^{-\beta Ei}$$

- where $\beta = 1/(k_B T)$. The average value of quantity X is then

$$\langle X \rangle = \sum_i X_i P(E_i)$$

- In some cases we can calculate this sum exactly, but often not. So this is a good problem for numerical approaches. Normally, we can't simply evaluate the sum numerically either. For example there are 10^{23} molecules in a single mole of gas. If the molecules each had only two possible quantum states (when they have many) then the number of states in this system would be 2^{10**23} which is an insanely large number. No super computer can get even close to evaluating this many states. Thus we must come up with some approximations.

STATISTICAL MECHANICS

- So let's try Monte Carlo, instead of evaluating all states, we will randomly sample N states and as N gets larger we will get closer to the correct value

$$\langle X \rangle \simeq \frac{\sum_{k=1}^{N} X_k P(E_k)}{\sum_{k=1}^{N} P(E_k)}$$

- Unfortunately in most situations this will not work. The Boltzmann probability is exponential, so the vast majority of states of exponentially small probability. Thus mosts of our randomly chosen energies contribute almost nothing to this sum.
- But we have just discussed a method to overcome this problem, important sampling. If we weight which energies we choose randomly then we can get the important ones in our sum.

IMPORTANCE SAMPLING AND STATISTICAL MECHANICS

- Here's how it works in detail. For any quantity g_i that depends on the states i, we can define the weighted average

$$\langle g \rangle_w = \frac{\sum_i w_i g_i}{\sum_i w_i}$$

- if $g_i = X_i\, P(E_i)/w_i$ then we get

$$\langle \frac{X_i P(E_i)}{w_i} \rangle_w = \frac{\sum_i w_i X_i P(E_i)/w_i}{\sum_i w_i} = \frac{\langle X \rangle}{\sum_i w_i}$$

- which gives us, just like before, that

$$\langle X \rangle = \langle \frac{X_i P(E_i)}{w_i} \rangle_w \sum_i w_i$$

- so if we use a weighted distribution of states then the average value of our quantity is just the weighted average of the quantity times the sum of the weights.

IMPORTANCE SAMPLING AND STATISTICAL MECHANICS

- Thus to calculate the average of X we can use the formula

$$\langle X \rangle \simeq \frac{1}{N} \sum_{k=1}^{N} \frac{X_k P(E_k)}{w_k} \sum_i w_i$$

- Note that the first sum is over the k states we sample, but the second is over all states. The second sum is usually done analytically if possible.

- If we choose weights that sample where the probability is large. This is easy to do. Just choose the probability as the weight, $w_i = P(E_i)$. Then $\Sigma w_i = 1$ by definition and we get

$$\langle X \rangle \simeq \frac{1}{N} \sum_{k=1}^{N} X_k$$

- where we have choses the states with the pdf of the Boltzmann's distribution.

IMPORTANCE SAMPLING AND STATISTICAL MECHANICS

- An alternative (and more physical) way of thinking about this approach is to think about the Monte Carlo as the behavior of an actual system.
- If we had some physical system it would start off in some initial state. Then it would move to another state with a probability given by Boltzmann. It would then keep changing states, each time with a probability related to its Boltzmann distribution.
- If we measured a quantity for the system we would get an average of the values that quantity has over the states the system has been in while we performed our measurement.
- Thinking of it this way importance sampling is just physically modeling what the system might do in an actual physics experiment.

MARKOV CHAIN

- Why it looks like we have a general method for statistical mechanics, there is still one problem, we have to calculate $P(E_i)$.
- The exponential part is easy, but remember the probability was also normalized by the sum of states Z, called the *partition function*. Unfortunately, it is usually not easy to calculate Z.
- Luckily there is a solution to this problem, we can make our calculation without knowing Z by a technique called a *Markov chain*.

MARKOV CHAIN

- Our goal is to have a set of states for the sum to get an average. We will do this by generating a sequence of states, that is the chain part of the Markov chain.
- Let's start with some state i. Now the next state, instead of choosing it randomly from the full ensemble, let's just make a small change to the system to get a new state j.
- The probability of going from state i to j is given by a *transition probability* T_{ij}. If we can choose our transition probability the right way then we can issue then we can insure that the chance of ending up at any one state on the chain is the Boltzmann probability.

MARKOV CHAIN

- So how do we choose T_{ij}? Well we know $\Sigma T_{ij}=1$, we have to end up at some state. Now we also want Tij to relate to the Boltzmann distribution so we have

$$\frac{T_{ij}}{T_{ji}} = \frac{P(E_j)}{P(E_i)} = \frac{e^{-\beta E_j}/Z}{e^{-\beta E_i}/Z} = e^{-\beta(E_j - E_i)}$$

- Note that Z cancels out in this equation. Now let's suppose we found the set of probabilities T_{ij} that satisfy the above equation and the probability that we are in state i on one particular step of the chain is given by $P(E_i)$. Then the probability of being in state j on the next step is the sum that we got there from any i.

$$\sum_i T_{ij} P(E_i) = \sum_i T_{ji} P(E_j) = P(E_j) \sum_i T_{ji} = P(E_j)$$

MARKOV CHAIN

- What this equation tells us is that if we have the right transition probabilities for every state on one step, then we have them for all other steps.
- We have not shown that it we start with some other distribution of states that we will converge to the Boltzmann distribution, but that is also the case (a proof is given in the appendix of the text book).
- But the important point is that if you start the system in any random state and let the chain run long enough then the distribution over states will converge to the Boltzmann distribution.

METROPOLIS ALGORITHM

- We still haven't discussed how to determine our transition probabilities T_{ij}. There is actually a lot of freedom in how they are chosen, that is many choices will still satisfy the equations we have needed to show that they are good choices.
- The most common choice is one that leads to what is called the *Metropolis algorithm*. To understand how this algorithm works note that we are allowed to visit the same state more than once in our Markov chain. That is T_{ii} can be greater than 0.
- Let's consider the change to our system to be the change of just one elements energy level chosen at random. In general this won't satisfy our previous equation but then let's accept or reject this state with a probability P_a given by

$$P_a = 1 \quad \text{if} \quad E_j \leq E_i \qquad\qquad P_a = e^{-\beta(E_j - E_i)} \quad \text{if} \quad E_j > E_i$$

METROPOLIS ALGORITHM

$$P_a = 1 \quad \text{if} \quad E_j \leq E_i \qquad\qquad P_a = e^{-\beta(E_j - E_i)} \quad \text{if} \quad E_j > E_i$$

- In other words if the change decreases the energy of our system we accept it. It if increases the energy then there is some probability P_a we accept it, otherwise we stay in the same state till the next step in our chain.

- Under this scheme the total probability T_{ij} of making a move from state i to j is the probability that we choose that move our of all possible moves, which is just 1/M if there are M possible moves, times the probability we will accept that move. So for example if $E_j > E_i$ then

$$T_{ij} = \frac{1}{M} \times e^{-\beta(E_j - E_i)}$$

$$T_{ji} = \frac{1}{M}$$

$$=> \frac{T_{ij}}{T_{ji}} = \frac{e^{-\beta(E_j - E_i)}/M}{1/M} = e^{-\beta(E_j - E_i)}$$

METROPOLIS ALGORITHM

- Likewise if $E_i >= E_j$ then $T_{ij} = \frac{1}{M}$

$$T_{ji} = \frac{1}{M} \times e^{-\beta(E_i - E_j)} \qquad => \frac{T_{ij}}{T_{ji}} = \frac{1/M}{e^{-\beta(E_i - E_j)}/M} = e^{-\beta(E_j - E_i)}$$

- In both cases we get agreement with the condition we asserted in the beginning of this discussion.
- Thus we have an method of generating probabilities that satisfy the conditions we states, which we have shown means that we get a probability of states distribution that is the same as the Boltzmann's distribution.
- The metropolis algorithm is a valid method to create a Markov chain.

METROPOLIS ALGORITHM

1. Choose a random starting state i.
2. Choose a move uniformly at random from an allowed set of moves
3. Calculate the acceptance probability P_a for the move.
4. Randomly decide if the move is accepted in which case the state becomes j or is rejected in which case the state remains i.
5. Measure the quantity of interest X for the current state and add it to a running total.
6. Repeat from step 2.

➤ When we have taken a large number of steps N, we take our running total and divide it by N to get <X>.

METROPOLIS ALGORITHM

- While the metropolis algorithm is pretty straightforward there are number of subtitles to be aware of:
  - When you reject a move that is still a step and the value of the quantity at that step must be added to your running total again.
  - It must be the case that the number of moves M be the same from i and j.
- The move set must be able to actually get you to every possible state of the system. A move set that does this is called *ergodic*, which is required for the Metropolis algorithm to work.
- Although the Markov chain always converges to the correct Boltzmann distribution, we don't know how many steps this will take and there is no general way to know.

EXAMPLE 10.3: MONTE CARLO SIMULATION OF AN IDEAL GAS

- The quantum states of an atom of mass m in a cubic box of length L have three integer quantum numbers, n_x, n_y, and $n_z = 1 \ldots \infty$ and energies by

$$E(n_x, n_y, n_z) = \frac{\pi^2 \hbar^2}{2mL^2}(n_x^2 + n_y^2 + n_z^2)$$

- An ideal gas is a gas of N such atoms that don't interact with one another. So the total energy is just the sum of energies.

$$E = \sum_{i=1}^{N} E(n_x^{(i)}, n_y^{(i)}, n_z^{(i)})$$

- So lets perform a Monte Carlo simulation for this gas. First we need our move set. Let's just let one of the atoms change one of its quantum numbers by +1 or -1.

EXAMPLE 10.3: MONTE CARLO SIMULATION OF AN IDEAL GAS

- If for example the n_x number changed by 1 the change in energy would be

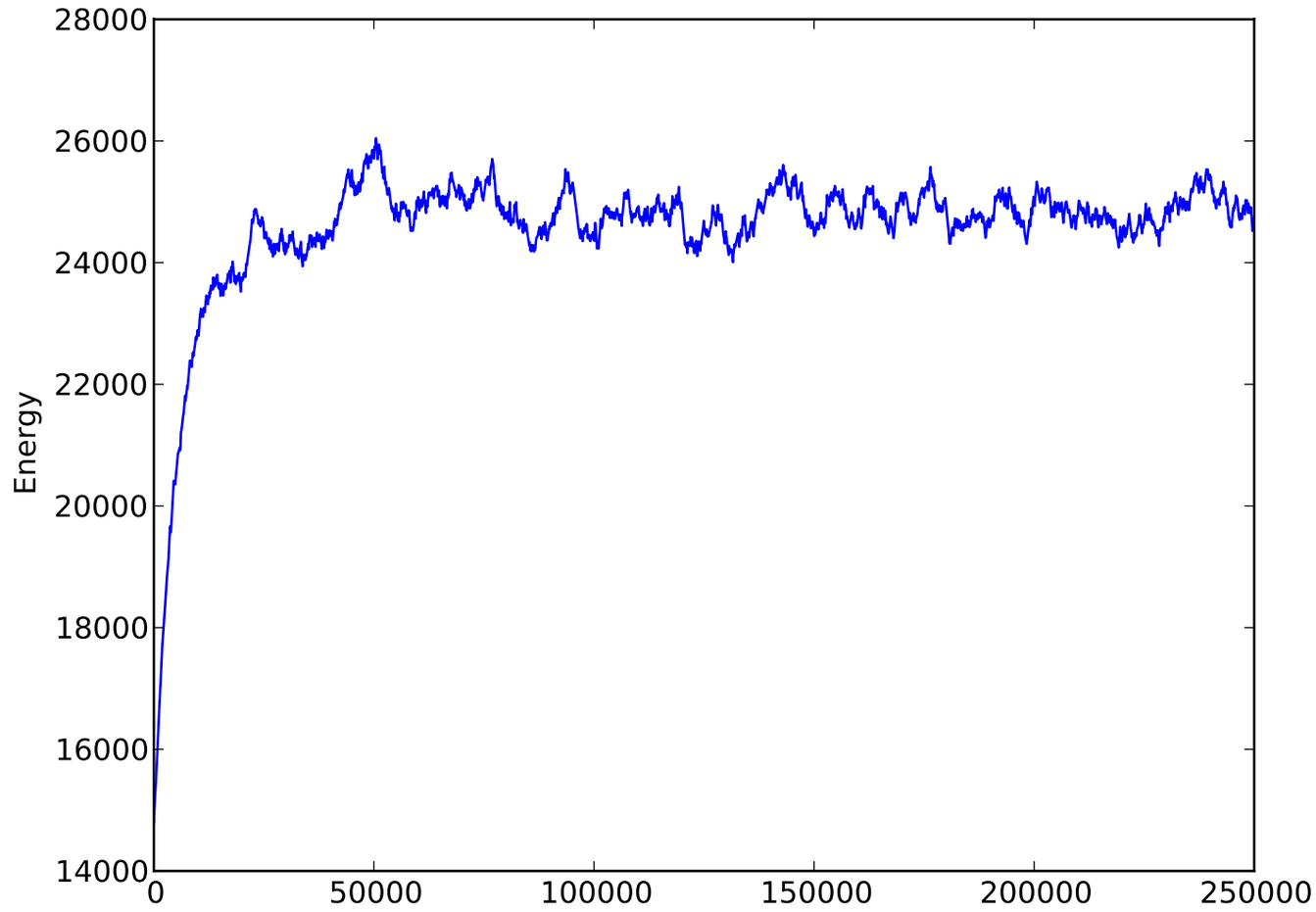
$$\Delta E = \frac{\pi^2\hbar^2}{2mL^2}[(n_x+1)^2 - n_x^2] = \frac{\pi^2\hbar^2}{2mL^2}(2n_x+1)$$

- a decrease of 1 gives

$$\Delta E = \frac{\pi^2\hbar^2}{2mL^2}[(n_x-1)^2 - n_x^2] = \frac{\pi^2\hbar^2}{2mL^2}(-2n_x+1)$$

- Let's perform our simulation for N=1000 particles at a temperature $k_BT=10$. For simplicity let's take m=h=L=1 and start in the ground state, $n_x=n_y=n_z=1$. Then we select a particle, axis and sign randomly for each move with a probability given by the above energy for any energy increases. Our code might look like this

```
T = 10.0
N = 1000
steps = 250000
# Create a 2D array to store the quantum numbers
n = ones([N,3],int)
# Main loop
eplot = []
E = 3*N*pi*pi/2
for k in range(steps):
    # Choose the particle and the move
    i = randrange(N)
    j = randrange(3)
    if random()<0.5:
        dn = 1
        dE = (2*n[i,j]+1)*pi*pi/2
    else:
        dn = -1
        dE = (-2*n[i,j]+1)*pi*pi/2

    # Decide whether to accept the move
    if n[i,j]>1 or dn==1:
        if random()<exp(-dE/T):
            n[i,j] += dn
            E += dE

    eplot.append(E)

# Make the graph
plot(eplot); ylabel("Energy"); show()
```



Notice that it takes like 50000 steps for the model to reach equilibrium. We wouldn't want to use those first steps in determining the average value of a quantity since they just have to do with where we started.

SIMULATED ANNEALING

- For many problems our goal is to find the global minimum. We discussed finding a minimum previously, but not a global minimum and mostly for one dimensional systems.
- Finding a global minimum is hard. There are many approaches to this problem, but none are guaranteed to be successful. One approach which is closely related to the Monte Carlo Markov Chain method we have been discussing is called *simulated annealing*.
- The idea behind simulated annealing is that we will look for our global minimum in a way similar to our solution of the Boltzmann problem.

SIMULATED ANNEALING

- That is we will start with a guess and then move randomly to other states. If the ‘energy’ of the other states is less we will move to it as we a looking for the lowest ‘energy’ state.
- However, just moving to lower energy states may leave us in a local, but not global minimum. Thus we also assume we have a certain temperature, so there is a probability to move to a higher energy state, just like in the metropolis algorithm.
- But if our temperature is very high, then the system won’t move to the ground state at all. So we do something that resembles the physical process of annealing.

SIMULATED ANNEALING

- Many substances, like glass, if you let them cool quickly don't end up in the ground state. Instead when glass cools defects occur, increasing the thermal stress, and making the glass more likely to break. To overcome this glass makers anneal the glass, they cool it slowly, so that defects are removed and the system cools to the global ground state instead of a local minimum.
- The goal of simulated annealing is the same. We start the metropolis algorithm with some temperature and then we slowly reduce it. In this way our algorithm is likely to jump out of local minimum, because of the temperature and is more likely to finally settle in the global minimum.

SIMULATED ANNEALING

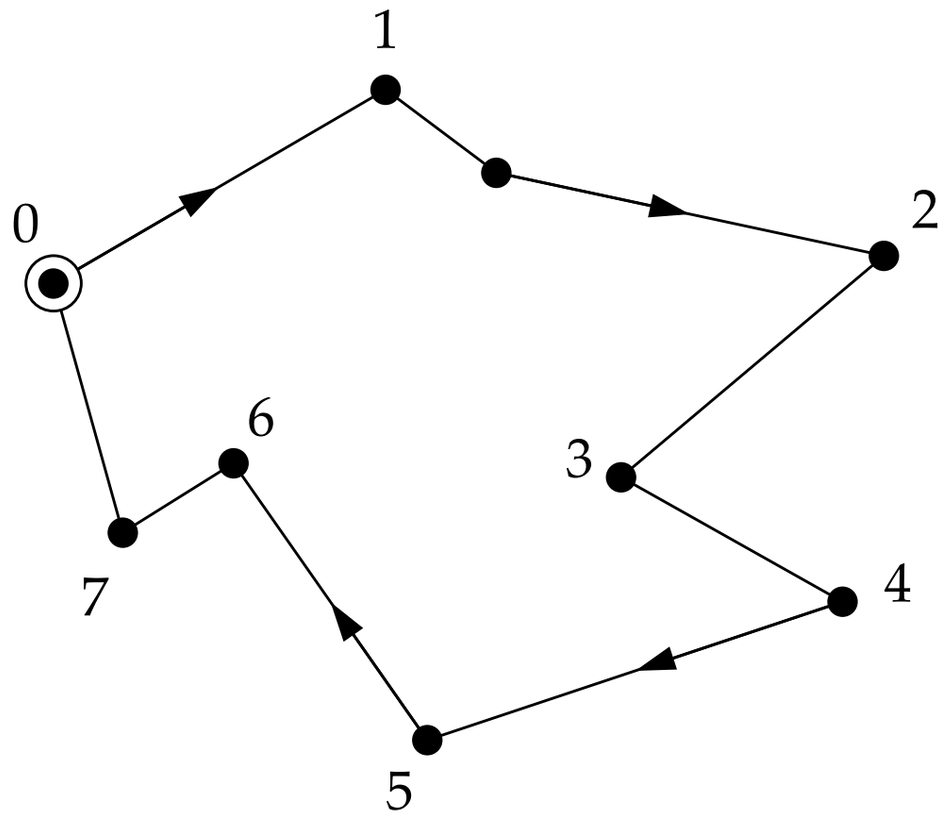
- Implementing simulated annealing is a straight forward extension of the metropolis algorithm. All that needs to be added is an initial temperature and a *cooling schedule*.
- The initial temperature should be chosen so that the Markov chain can easily move to higher energy states. In other words that the temperature is higher than the typical energies between states. This way the starting state doesn't matter as the systems will easily move to any possible random state.
- An exponential is the most common choice for the cooling schedule

$$T = T_0 e^{-t/\tau}$$

- The choice of τ usually most be determined by trial and error. The slower the cooling (larger τ) the more likely you are to find the global minimum, but also the longer the process will take.

EXAMPLE 10.4 THE TRAVELING SALESMAN

- As an example of simulating annealing let us look at the traveling salesman problem.
- The problem is we have a salesperson who needs to visit N cities on her trip and we want to figure out what is the shortest route between the N cities.
- While this might seem like a silly problem it is one that is studied a lot in computational science because it turns out to be NP-hard.



- What that means is that it is very hard to find a fast method for solving the problem.
- Note the direct approach goes as N! which is very bad scaling.

THE TRAVELING SALESPERSON

- Let us choose the locations of the N cities randomly within a unit square. Let the distance be on a flat surface. Each city is then represented by a vector $\mathbf{r}_i = (x_i, y_i)$ and $\mathbf{r}_N = \mathbf{r}_0$ since we have to end up where we begin. The distance the salesperson travels is then

$$D = \sum_{i=0}^{N-1} |\mathbf{r}_{i+1} - \mathbf{r}_i|$$

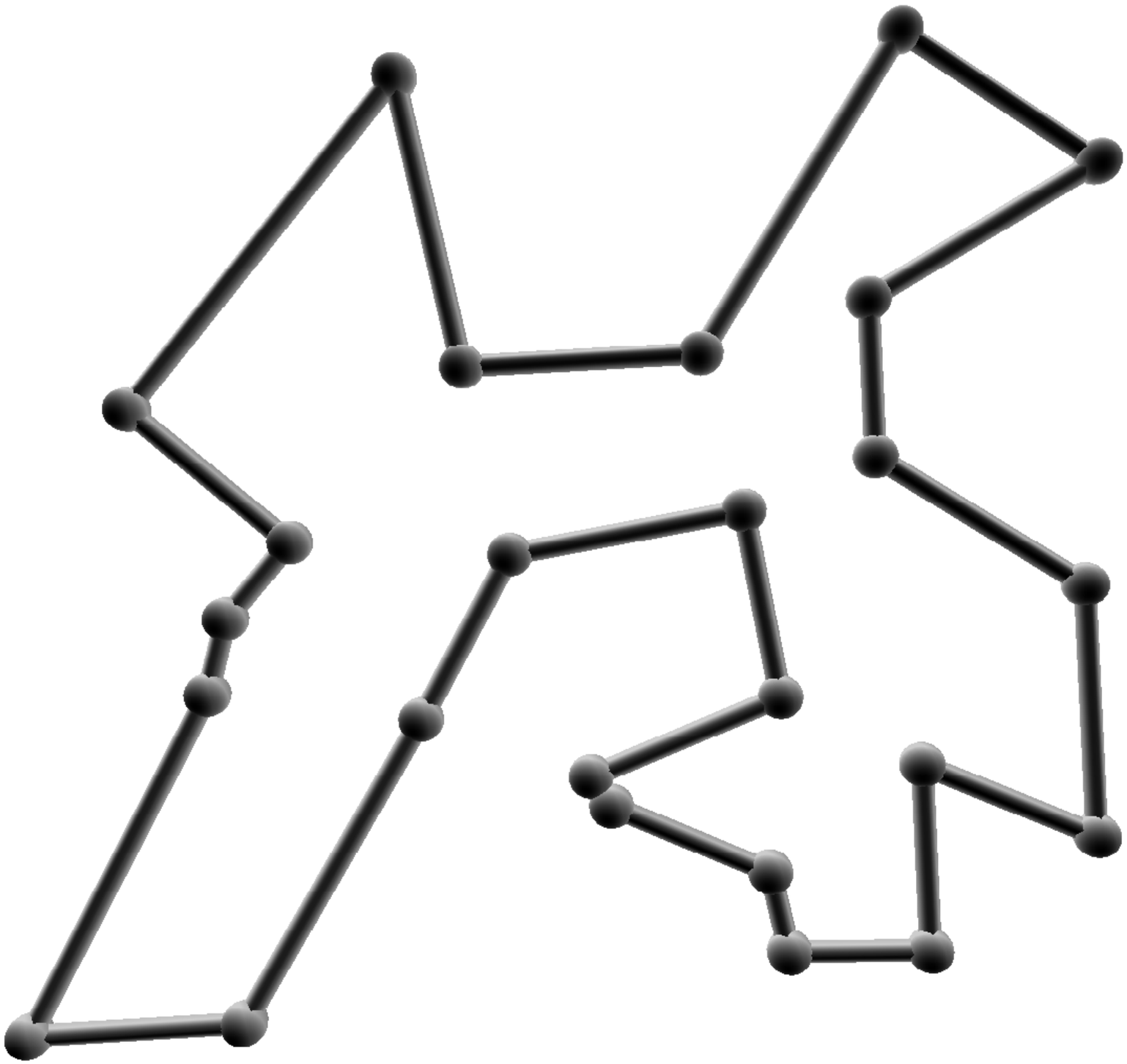
- we want to minimize D for all possible ordering of the cities. Our set of moves can be swapping two cities. So we start with an initial ordering of the cities and then we swap two and see if D becomes shorter or longer.

```
from math import sqrt,exp
from numpy import empty
from random import random,randrange
from visual import sphere,curve,display

N = 25
R = 0.02
Tmax = 10.0
Tmin = 1e-3
tau = 1e4

# Function to calculate the magnitude of a vector
def mag(x):
    return sqrt(x[0]**2+x[1]**2)

# Function to calculate the total length of the tour
def distance():
    s = 0.0
    for i in range(N):
        s += mag(r[i+1]-r[i])
    return s
```

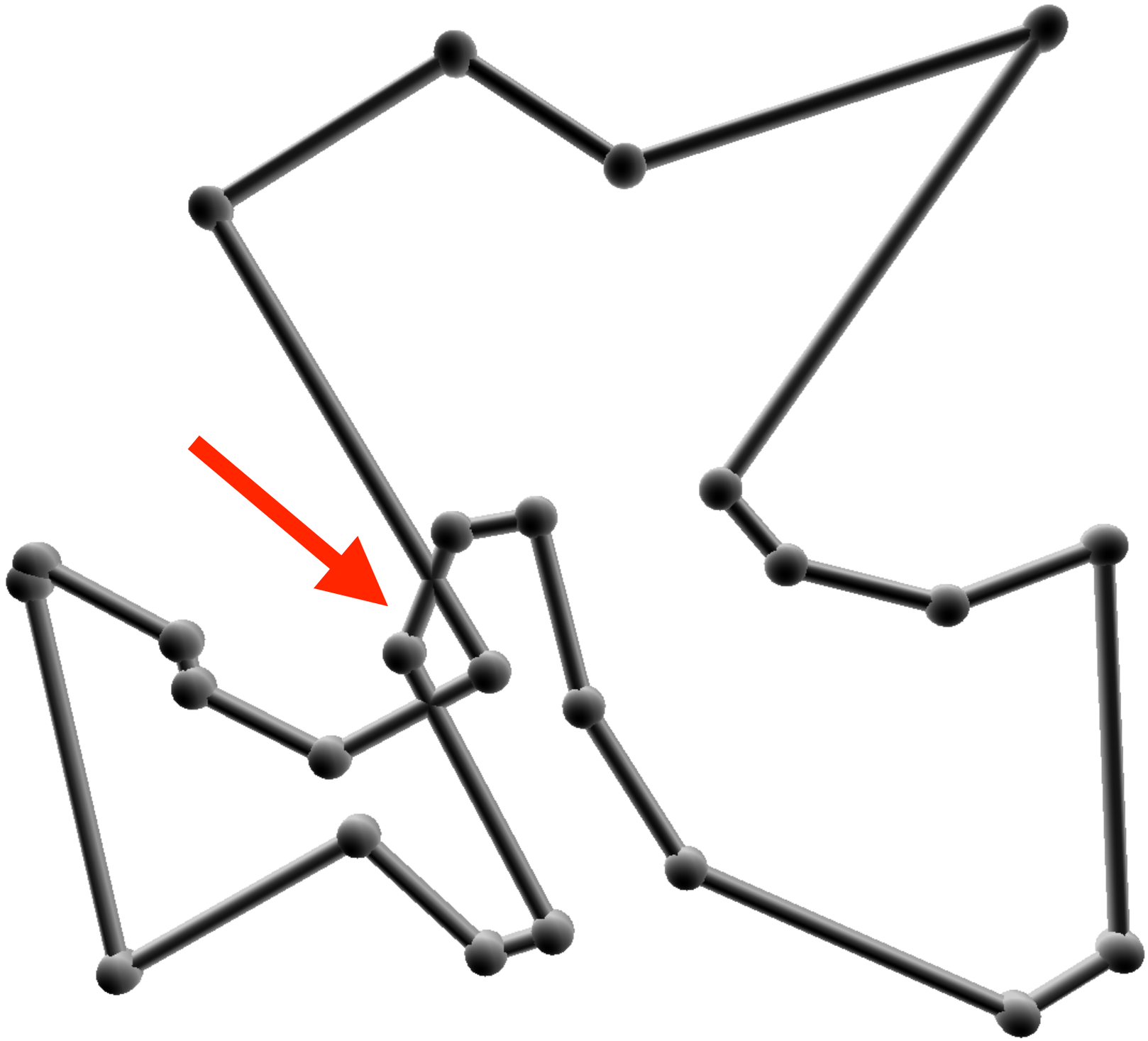
```
# Choose N city locations and calculate the initial distance
r = empty([N+1,2],float)
for i in range(N):
    r[i,0] = random()
    r[i,1] = random()
r[N] = r[0]
D = distance()


# Set up the graphics
display(center=[0.5,0.5])
for i in range(N):
    sphere(pos=r[i],radius=R)
l = curve(pos=r,radius=R/2)
```

```
# Main loop
t = 0
T = Tmax
while T>Tmin:

    # Cooling
    t += 1
    T = Tmax*exp(-t/tau)

# Update the visualization every 100 moves
if t%100==0:
    l.pos = r

# Choose two cities to swap and make sure they are distinct
i,j = randrange(1,N),randrange(1,N)
while i==j:
    i,j = randrange(1,N),randrange(1,N)
```

```
# Swap them and calculate the change in distance
oldD = D
r[i,0],r[j,0] = r[j,0],r[i,0]
r[i,1],r[j,1] = r[j,1],r[i,1]
D = distance()
deltaD = D - oldD

# If the move is rejected, swap them back again
if random()>exp(-deltaD/T):
    r[i,0],r[j,0] = r[j,0],r[i,0]
    r[i,1],r[j,1] = r[j,1],r[i,1]
    D = oldD
```



One solution found by running the program. The solution looks rather reasonable by eye. The visual package was used to make the graphic.



Another solution. However, this time we can see it is not the best solution. It would be more efficient two swap the two points where the route crosses. This is typical of simulating annealing. It will usually give you a pretty good solution, but maybe not the best possible solution.

TERMINOLOGY

- Pseudorandom Number - a completely deterministic algorithm that creates numbers that appear to be random
- Transformation Method - a way to transform uniform random numbers to other distributions
- Monte Carlo Integration - an integration method where the sampling points are chose randomly
- Importance Sampling - weighting the points when using a Monte Carlo method
- Markov Chain - a way of selecting points using previous points (the chain) so that the sample approaches a desired distribution
- Metropolis Algorithm - an algorithm that creates a Markov Chain by using a transitions probability that is accepted or rejected.
- Simulated Annealing - An algorithm for optimization where one starts with a high ‘temperature’ to explore many states and then the temperature is ‘cooled’ to find the minimum.