Simulating radioactivity by rolling dice

This lab uses a simple analogy to show how the exponential law arises from the (random) decay probability of individual atoms.

We use a collection of six - sided dice as our "radioactive" atoms. Here's how the analogy works:

Think of the dice as atoms of a radioactive parent element (Element 117: Wikipedium). Each roll of the dice represents an increment of time, let's say, one year. Dice that land on "6" are considered to have "decayed" (to daughter element 115: Iphonium). We remove them from the population and record the number that survived. Now we roll the remaining dice again (another "year" goes by). Again, we remove those that land on '6', note the number of surviving atoms, and roll them again. And so on ....

The probability that one of our individual "atoms" "decays" each "year" is constant and not influenced by its past history or present circumstances. In this case, the probability is 1/6 per year :

$$\lambda = 1/6 \text{ yr}^{-1} = 0.1667 \text{ yr}^{-1}$$

Mathematica simulation

Using a small number of real dice introduces a lot of statistical variability. We can use computer - generated random numbers to do a much more thorough simulation:

Here' s a Mathematica function that simulates the game of rolling dice and removing those that land on ' 6'. At its core is a ' dice' simulation which asks Mathematica for an evenly distributed random number between 0 and 1, multiplies by 6 and takes the integer above. This has an equal chance of returning each of the values {1, 2, 3, 4, 5, 6}. The function repeats this n times to build up a table of simulated dice rolls. Then it counts and returns the number of rolls NOT EQUAL to ' 6'. In the lab exercise, this is the number deemed to have ' survived' radioactive decay.

In[154]:=
```
diceroll[n_] := Length[Select[Table[Ceiling[6 * Random[]], {n}], # ≠ 6 &]]
```

In[216]:=
```
(*It's very fast. For 100000 dice throws,
my laptop takes 0.087 seconds.  NB: (1 - 1/6) = 0.8333.*)

Timing[diceroll[100 000]]
```

Out[216]=
```
{0.087454, 83 378}
```

Now we can build this into a little program that simulates the lab exercise, with as many dice, for as many "years", and for as many trials as we' d like. The 'survive' function is defined recursively, which makes it very fast. The inputs are: number of dice, number of rolls ("years") and number of times we want to simulate the experiment. The output is a list of "surviving" dice at the end of each trial.

In[217]:=
```
monte[ndice_, nrolls_, ntrials_] :=

 Do[
  out = Table[null, {ntrials}];

  Do[
   Clear[survive];
   survive[x_] := survive[x] = diceroll[survive[x - 1]];
   survive[0] = ndice;
   out[[i]] = survive[nrolls],
   {i, 1, ntrials}];

  Return[out],
  {1}]
```

Here is a simulation of the lab results - 3 trials with 200 dice for 12 years :

In[218]:=
```
monte[200, 12, 3]
```

Out[218]=
```
{17, 23, 19}
```

Here is another, and another ...

In[221]:=
```
{monte[200, 12, 3], monte[200, 12, 3], monte[200, 12, 3]}
```

Out[221]=
```
{{17, 30, 24}, {36, 22, 23}, {26, 18, 13}}
```

As we noticed in lab, the results seem to vary widely from one trial to the next. To get a good estimate of the expected outcome, we need to run many trials and take the mean :

In[226]:=
```
Mean[monte[200, 12, 10 000]] // N
```

Out[226]=
```
22.4058
```

Or we could use more dice. Notice how the mean is similar, but the results are much more tightly grouped (NB: I've normalized the answer for comparison with the 200-dice results above)

In[255]:=
```
(200 / 100 000) * monte[100 000, 12, 6] // N
```

Out[255]=
```
{22.418, 22.418, 22.452, 22.404, 22.482, 22.756}
```

In[249]:=
```
(200 / 100 000) * Mean[monte[100 000, 12, 20]] // N
```

Out[249]=
```
22.3884
```

Which approach is better? Both examples involve the same total number of dice rolls (10000 x 200 vs 20 x 100000). If this were an experiment in radioactive decay counting, we'd have counted the same number of decays either way. Looking superficially at the histograms showing the number of surviving atoms, the trials with 100,000 dice appear more precise. They have a variance of ± 0.18 compared to the more numerous trials with only 200 dice, which have a variance of ± 4.4.

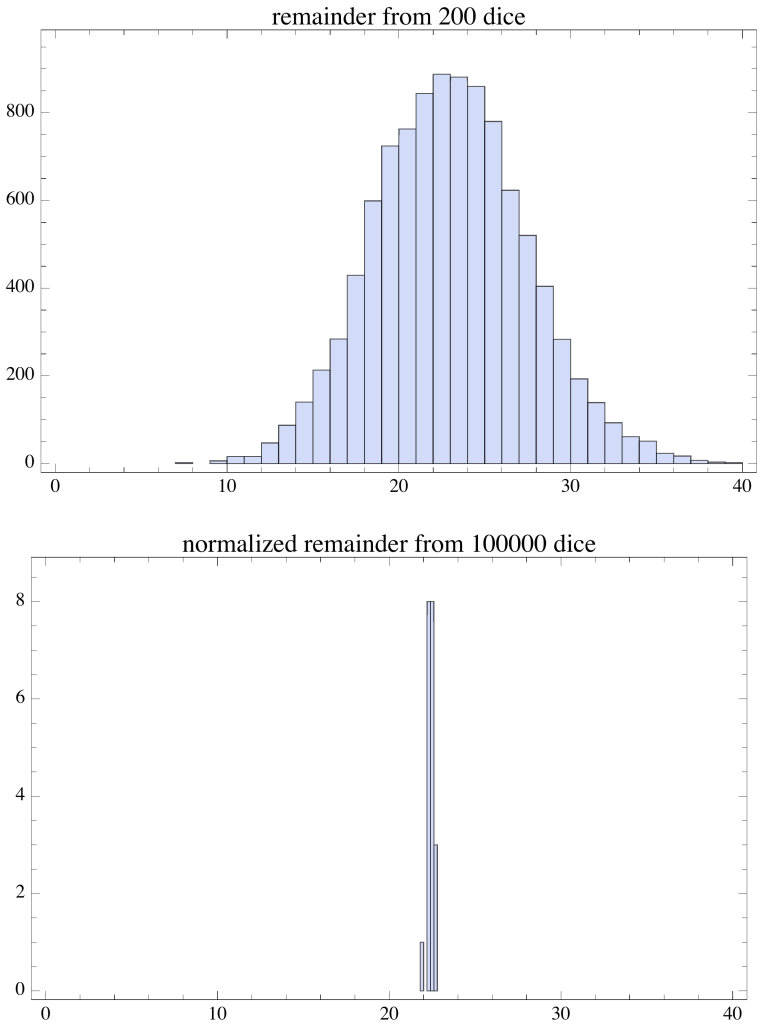
In[234]:=
```
Print[StandardDeviation[monte[200, 12, 10 000]] // N ];
Print[(200 / 100 000) * StandardDeviation[monte[100 000, 12, 20]] // N]
```

```
4.44205

0.177498
```

In[253]:=
```
GraphicsGrid[
 {{Histogram[monte[200, 12, 10 000], PlotRange → {{0, 40}, Automatic}, Frame → True,
    AxesOrigin → {0, 0}, PlotLabel → "remainder from 200 dice"]},
  {Histogram[(200 / 100 000) * monte[100 000, 12, 20], PlotRange → {{0, 40}, Automatic}, Frame →
     True, AxesOrigin → {0, 0}, PlotLabel → "normalized remainder from 100000 dice"]}}]
```

Out[253]=



Actually ... the real measure we should use to assess how well we know the mean value is the 'standard error of the sample mean', equal to the variance divided by the square root of the (number of trials -1). By this measure, the two approaches give the same confidence in our estimate of the mean value:

In[258]:=
```
Print[StandardDeviation[monte[200, 12, 10000]] / (10000 - 1)^0.5 // N];
Print[(200 / 100000) * StandardDeviation[monte[100000, 12, 20]] / (20 - 1)^0.5 // N]
```

```
0.0449145

0.0448751
```

For a nicely behaved, normally-distributed set of results like these, free of systematic errors, we can be ~68 % confident that the real answer lies within ± one standard error of our experimentally determined mean (95 % confident that it lies within ± twice the standard error).

Appendix

Did you notice that our experiment consistently returns a lower number of surviving atoms than predicted by the exponential law of radioactivity? This is due to a well - hidden flaw in the simulation : Each "year" we reduce the population in a single dice roll, by 1/6 (= 0.8333) on average. But in a large population the number of atoms declines continuously through the year, and the decay probability of 1/6 applies to fewer and fewer atoms as the year goes on. As a result, the proportion remaining after a year is actually 0.8465 ($e^{-1/6} = 0.8465$). The dice simulation causes a few too many "atoms" to decay each year. Starting with 200 radioactive atoms with decay constant $\lambda = 1/6$ yr^{-1} , the number expected after 12 years is 27 ($200\, e^{-12/6} = 27.06$). The dice simulation predicts 22.4.

To (almost) fix this, we can introduce a small fudge-factor in the math: If we apply the 1/6 cut to a population increased by (0.8465 / 0.8333) it will leave the correct proportion at the end of each year. The problem is that this produces non-integer numbers of 'surviving' atoms, which breaks the analogy. These have to be rounded to integers for input to the "diceroll" function, which produces significant errors for small numbers of surviving dice. (... In fact, it cycles between over- and undercorrecting as the number of surviving dice changes. e.g. for 32 < n < 63 it overestimates survival, for n < 32 it has no effect at all and the function reverts to underestimating survivors).

In[349]:=
```
carlomonte[ndice_, nrolls_, ntrials_] :=

 Do[
  out = Table[null, {ntrials}];

  Do[
   Clear[survive];
   survive[x_] := survive[x] = diceroll[Round[(0.8465 / 0.8333) * survive[x - 1]]];
   survive[0] = Round[(0.8465 / 0.8333) * ndice];
   out[[i]] = survive[nrolls],
   {i, 1, ntrials}];

  Return[out],
  {1}]
```

In[353]:=
```
Mean[carlomonte[200, 12, 40]] // N
```

Out[353]=
```
27.7
```