Immigration Modelling with Differential Equations:
An incredibly unnecessary complication of an assignment
that isn't even graded

C. Cox, S. Overflow

September 27, 2021

Abstract

580 lines of code. 13 pages. 7 graphs. 24 straight hours. 31 equations. This project will go down in history as the most absurdly overdone homework assignment ever submitted. In the following pages, we delve into the topics of modelling population decay with differential equations, data creation and visualization, and why one semester of Differential Equations isn't quite enough. Using Python and its scientific computing tools, we show a visual representation of the decay of a certain population of immigrants that are unfortunately destined to leave approximately half of their pack behind at each border crossing. There were some brief asides into exponential regression, Bayesian Statistics, and binomial distribution. Our findings got quite interesting after analyzing the results with the help of Mathematics Stack Exchange, and finding a more accurate solution far beyond the scope of this class. Did I mention this assignment is not worth any credit?

The code used in this project can be viewed at:
https://github.com/isademigod/populationproblem

1 The Problem

The original given problem was to model the exponential decay of a population of immigrants as they leave approximately half of their population behind at any given border crossing. Using a linear Ordinary Differential Equation, we were to first find the theoretical decay of such a system, and then experimentally test the findings by flipping a bunch of coins or M&M's$^{\text{TM}}$ brand candy-coated chocolates. The first problem was the simplest, where the only change in population was from a number of them flipping a coin to decide to stay at the border or continue on. The second problem was more complex, adding 10 people at each border crossing, and the last, well... I ignored the instructions on that one. I was unfortunately all out of M&M's $^{\text{TM}}$, and was unable to find enough coins, so I used the full extent of my Python and Googling skills to perform the experiment virtually.

2 The Theory

2.1 Problem 1

The given problem can be simplified as follows: "A number of people are crossing a number of borders, and each one flips a coin at the crossing to determine whether they will stay in that country or continue onward. How many borders will be crossed until there are 10 (or none) left?

This situation can be modelled as:

$$\frac{\Delta N\,(x)}{\Delta x} = -0.5N, N(0) = 1000 \tag{1}$$

Where the delta fraction represents the change in number of people for each trial, the 0.5 represents the expected loss of half the population per trial, and the 1000 represents the initial population in my simulation. With some algebra and a bit of calculus we can obtain the linear ODE:

$$N' = -0.5N \tag{2}$$
$$N' + 0.5N = 0 \tag{3}$$
$$\frac{dN}{dx} + 0.5N = 0 \tag{4}$$
$$\tag{5}$$

Using $\mu = e^{0.5x}$ as our integrating factor and simplifying as the result of a product rule:

$$\frac{d}{dx}\left[e^{0.5x}N\right] = 0 \tag{6}$$
$$e^{0.5x}N = C \tag{7}$$
$$N = Ce^{-0.5x} \tag{8}$$
$$C = 1000 \tag{9}$$
$$N = 1000e^{-0.5x} \tag{10}$$

We find our solution to the differential equation, which can be interpreted as the expected chart of the population of immigrants as they pass through each border. Solving this equation for $N = 10$ we can see that:

$$x = \frac{ln\left(\frac{10}{1000}\right)}{-0.5} \approx 9.21 \tag{11}$$

So theoretically, after 9 trials, most groups starting with 1000 people will have less than 10 remaining to continue past the border. In the experimental section we will verify the accuracy of this solution.

2.2 Problem 2

Much of the theory for the second problem is the same as the first, except that for each border crossing 10 people are added to the group. This can be represented as:

$$\frac{\Delta N\,(x)}{\Delta x} = -0.5N + 10, N(0) = 1000 \tag{12}$$

As the number of people on crossing $x+1$ is equivalent to the half the number of people at crossing x, plus 10 more. This time I solved the DE as a seperable equation, because solving it as a linear got me into some algebra I didn't feel like doing:

$$\frac{dN}{dx} + 0.5N = 10 \tag{13}$$
$$\frac{dN}{dx} = 10 - 0.5N \tag{14}$$
$$dN = 10 - 0.5Ndx \tag{15}$$
$$\frac{1}{10 - 0.5N}dN = dx \tag{16}$$
$$-2\ln\left(10 - 0.5N\right) = x + c \tag{17}$$
$$\ln\left(10 - 0.5N\right) = -\frac{1}{2}x + c \tag{18}$$
$$10 - 0.5N = e^{-0.5x+c} \tag{19}$$
$$N = -2ce^{-0.5x} + 20 \tag{20}$$

Substituting for $N(0) = 1000$:

$$N = 20 + 980e^{-0.5x} \tag{21}$$

This equation asymptotically approaches N=20 as the number of trials (x values) increase. From just a quick mental check, I expect to see similar behavior from the experimental portion, as we will eventually reach a point where the number of immigrants remaining is 20, and then subtract ≈ 10 from the group, and add 10 more *ad infinitum.*

2.3 Problem 3

Yeah, yeah. If you pick a different number, the expected equation changes relative to the number of people added at the border. For example, consider 20 picked up instead of 10:

$$N' + 0.5N = 20,\ N(0) = 1000 \tag{22}$$

$$N(t) = 40 + 960e^{-0.5t} \tag{23}$$

Or 5:

$$N' + 0.5N = 5,\ N(0) = 1000 \tag{24}$$

$$N(t) = 10 + 990e^{-0.5t} \tag{25}$$

When a different number of people are picked up at the border, the population will asymptotically approach a level $2x$ the number of people picked up.

3 The Experiment

As stated previously, I was fresh out of both coins and MM's$^{\text{TM}}$, so I leveraged what little Python knowledge I have plus the generous contributions of numerous library developers and Stack Overflow answers to create an adequate simulation of the situations in each problem. Nearly 300 lines of code were written in total, but when compared to leaving the house in order to find something to flip, I consider that a more viable solution. Who am I kidding, though. I had a blast with this problem, as is likely evidenced by the length of this manuscript.

3.1 Problem 1

My solution started with a simple for loop that picked a random number of immigrants to leave behind at each border, and print a list containing the number of immigrants remaining at each border. However, I quickly decided that not only was this implementation not nearly elegant enough, it was also completely wrong. Simply choosing a random number of immigrants for each crossing resulted in erroneous results such as only one person staying behind or 999 people staying behind at the first border crossings. The results of this method over 100 trials can be seen in Figure 1.

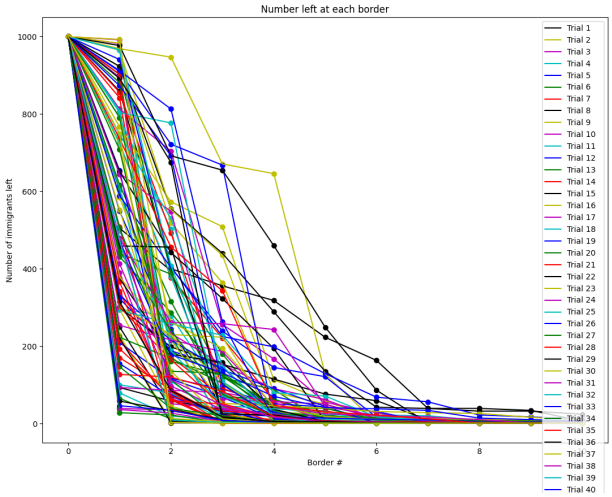

Figure 1: Random Chance Graph

For those two trials that ended with only one person on either side, the probability of that occurring in my method is, well, by definition 1 in 1000. If the simulation were accurate, the probability of flipping 1000 coins and getting one head can be represented as:

$$1 - (1 - 2^{-1000})^{10^6} < \frac{10^6}{10^{300}} = 10^{-294} \tag{26}$$

By Bernoulli's Inequality [ca17]. And while I have no possible way of conceptualizing the scale of 10 to the power of anything three digits, I think it suffices to say that it's not likely to happen in my first run of 100 trials. Clearly there is a problem here.
I asked my calculus professor, and he told me what I should have realized from the start. The fraction of people left behind at a given border crossing is best represented by a binomial distribution, which effectively means that if each person flipped a coin, the number passing through can be represented with a normal weighted random. I should have realized this before even starting the code, but you can't blame me; I never took a stats class. (Cooincidentally, this is the same reasoning I use to excuse my propensity for gambling.) Thankfully, the Numpy library in Python provides a function for binomial randoms, allowing me to recite the mantra of Python programmers: "Fear not the complex mathematics, for the library devs are with you". My code ended up looking something like this: [Cox21]

```
import numpy
data = {}
#10 total trials
for j in range(10):
    #initialize data
    num_left = [1000]
    border = [0]
    immigrants = 1000
    #10 border crossings
    for i in range(10):
        #lazily avoid division by zero
        if immigrants == 0:
            num_left.append(0)
            border.append(i+1)
```

```
        #main logic
        else:
            #number of coin flips out of number of people
            #that come up heads
            flips = sum(np.random.binomial(1, 0.5, immigrants)==0)

            normal_rand = flips/immigrants
            num_leave = math.trunc(immigrants * normal_rand)

            #subtract the number of leavers from the total remaining
            immigrants = immigrants - num_leave

            #data collection
            border.append(i+1)
            num_left.append(immigrants)
    data["Trial "+str(j+1)] = num_left
```

Ignoring about 80 lines of data collection, averaging, graphing, and finding the resulting equation of the line of best fit from the 10 trials. One may note that the tests with random distribution used 100 trials, this is to account for the random variation so that I may take an average later.

The results of this test were far more satisfying, with each test lining up beautifully with very little deviation. The resulting graph from matplotlib shows this nicely: I used an

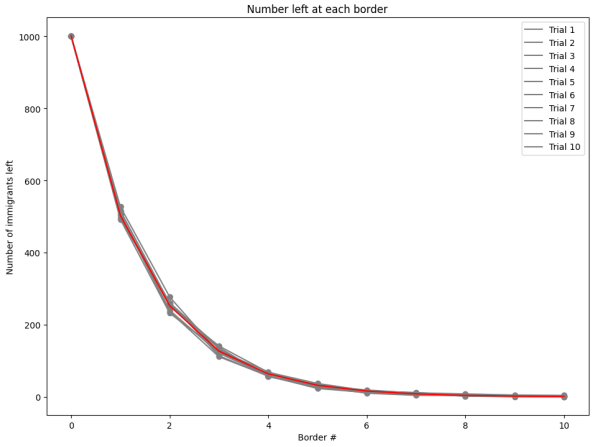

Figure 2: Binomial Chance Graph

exponential regression to find the line of best fit, in two different ways. The more hand-coded version is shown below, in which I take the natural log of the y-values, perform a linear regression, and then undo the log as a poor man's exponential regression.

```
from scipy import stats
#take an average of the returned y-values
y_lists = np.array([],int)
for l in data.values():
    l_array = np.array(l)
    y_lists = np.concatenate([y_lists,l_array],axis=0)
#reshape concatenated array into the format i need
```

```
y_lists = y_lists.reshape(20,11).astype(float)
average_y = np.average(y_lists,axis=0)

#exponential regression
A_vals = []
B_vals = []
#curve fitting using ln
x_data = np.array(border)
#take the natural log of all the y values
y_data = np.log(list(average_y.astype(float)))
#fit to a linear graph, b is the slope, A_log is intercept
b, A_log = scipy.stats.linregress(x_data, y_data)
#undo the log
a = np.exp(A_log)
#now we have our a and b
```

Later, the resulting equations became far too complex for me to want to hand-code a regression function, so I just used a lambda function to define the type of equation I was hoping to get and plugged it into SciPy's robust exponential regression feature:

```
#exponential regression
A_vals = []
B_vals = []
C_vals = []
#iterate over y-values from repeated trials to
#get an average for a and b where y=a*e^bx
for y in data.values():
    x_data = np.array(border)
    y_data = np.array(y)
    popt, pcov = scipy.optimize.curve_fit(
    lambda t,a,b,c: a*np.exp(b*t)+c,
    x_data, y_data,  p0=(1000, -0.5, 20))
    print(popt)
    a = popt[0]
    b = popt[1]
    c = popt[2]
    A_vals.append(a)
    B_vals.append(b)
    C_vals.append(c)

#get average a and b values
eqA = sum(A_vals)/len(A_vals)
eqB = sum(B_vals)/len(B_vals)
eqC = sum(C_vals)/len(C_vals)

print("the equation is y=",eqA,"*e^(",eqB,"*x)+",eqC)
```

We can simply drop those a and b values into the exponential form $y = ae^{bx}$. The resulting equation given by the program was:

$$y = 1001.2 \cdot e^{-0.689x} \tag{27}$$

Which is quite close to our solution in equation (8), although the B value is notably higher than -0.5. Interesting.[Foreshadowing...]

3.2 Problem 2

The upside of doing this problem the long way around is that it gets way easier as time goes on. I had to type a grand total of 4 characters to modify my code to test the second scenario. The main logic block turned into this:

```
for i in range(30):
    normal_rand = (sum(np.random.binomial(
        1, 0.5, immigrants)==0)
        /immigrants)
    num_leave = math.trunc(immigrants * normal_rand)

    #subtract the number of leavers, add 10
    immigrants = immigrants - num_leave + 10
    #data collection
    border.append(i+1)
    num_left.append(immigrants)
```

Can you spot the difference? I added a +10 that increases the count by 10 for each cycle. Eagle eyed readers may also notice that I changed the number of borders to cross up to 30, as I noticed in the worksheet that the number of trials recommended went up to 17. I took that as a cue that I might need a few more iterations to see the pattern. And the pattern was once again quite clear:

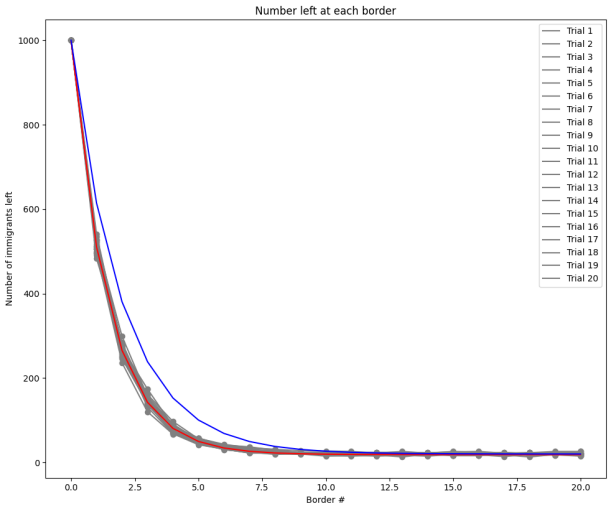

Figure 3: Problem 2 Graph

The experimental values behaved quite nicely, and produced an equation of $y = 979e^{-0.689} + 20$, with a and c values lining up well with our expected graph of $y = 980e^{-0.5x} + 20$. Although once again, the b value is considerably higher than 0.5. That is certainly quite curious, and one might start to think there may be some sort of systematic error involved.

3.3 Problem 3

We already found the solution to this one, and I could go ahead and graph it by changing 1 character in my code. That's boring, though. I already did all this work so let's play around a little bit.

Let's start by testing an edge case: What happens if $100,000$ people join their party on each border crossing?

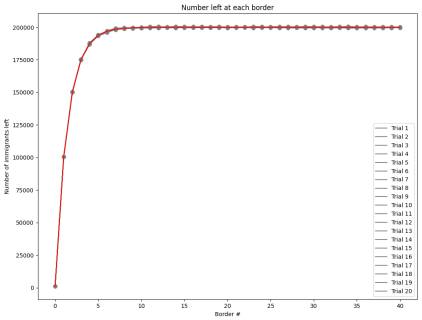

Figure 4: 100000 People Join

Neat, albeit predictable. The graph still asymptotically approaches a value $2x$ the number of people picked up at each crossing. If you're curious, the equation calculated by the regression algorithm is $y = 200,000e^{-0.69x} + 200,000$. (0.69 again.... hmmm...)
What if a random number between 0 and 200 immigrants joined their party at each border?

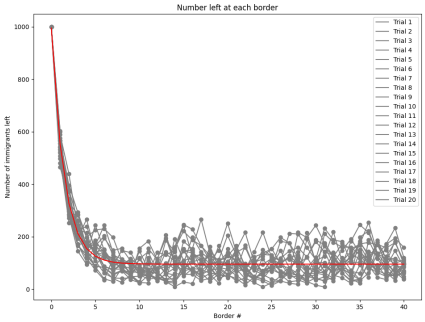

Figure 5: Random Noise

Eh, looks like just a bunch of random noise that averages out to be about the same as if the number of immigrants joining the party was 100.
How about this: Instead of flipping a coin to proceed, they play one hand of Five Card Draw poker and only proceed if they get a Royal Flush? (Odds of success = 1 in 2,598,960)

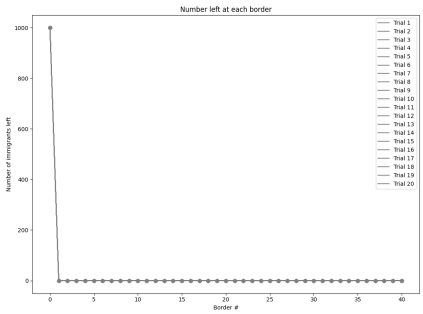

Figure 6: Royal Flush

I'm not really sure what I expected. I ran this one a couple times hoping that one person would make it through the first border, but that is once again showing my lack of understanding when it comes to statistics. Fun fact, these odds are equivalent to only crossing the border if they get struck by lightning five times before they reach it. [Com21]

For my final trick, what if we assume this takes them a long time (or they're rabbits) and we account for the population growing naturally by reproduction? Moreover, they must be immigrating for a reason, so we can assume that the standards of living increase with each country they visit, thereby increasing the number of births and lowering the number of deaths.
Let's keep the coin flip rule, but add a term that is essentially $+e^{rx}$ where r is the population growth, which starts at 0.0002 and increases proportionally to the number of borders they've crossed. I implemented this as:

```
#where i is the number of borders crossed:
r = (0.0002*i)
#num_leave is still from the coin flip
immigrants = (immigrants - num_leave) + math.e**(r*i)
```

And it would appear that this will result in an eventual explosion of population:

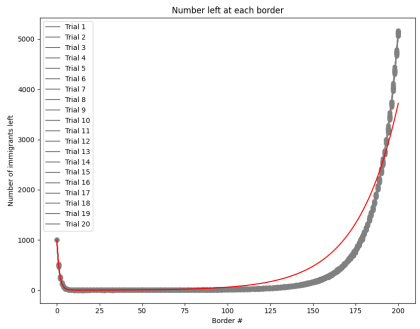

Figure 7: Population Growth

This example has a lot of potential for expansion. It would be interesting to try and work out the Differential Equation that would model this graph. Also, I'm not sure where on the globe you could go to cross 200 borders, but it might be possible to apply

some variation of the Four Color Theorem to figure out their path. These extensions are both left as an exercise to the reader. 😊

4 This Entire Paper is Wrong

But seriously, what's up with −0.69? (Nice, by the way) Why did my models fit the data so well in every way except for the exponent value being too low? My quest for the answer led me on a rabbit hole of desperation so deep that I actually ASKED a question on Stack Exchange after my Google skills failed me.
After they told me my differential equation was set up incorrectly, and I told them I did it exactly how it's supposed to be done, they mentioned a tidbit that led me to this line in the textbook:

> Dynamical systems are classified as either discrete-time systems or continuous-time systems. In this course we shall be concerned only with continuous-time systems—systems in which all variables are defined over a continuous range of time.

And there the needle drops. In our class, "A First Course in Differential Equations", we focus only on the continuous case of ODEs, but the discrete case is yet to come. Meaning that any solution we get for a problem involving variables that do not change continuously (i.e. only change when people cross a border) will only be an approximation that's easier to solve.

So what is the right way to solve this?

I must admit that the concept of discrete-case differential equations is new to me, but the nice folks at Stack Exchange provided enough of an explanation that I think I can apply it to this particular case.

So starting over, we know that "The number of people at the next border crossing is equal to the number of people at this border crossing minus half". Written in math:

$$N(x+1) = N(x) - \frac{1}{2}N(x) = \frac{1}{2}N(x) \tag{28}$$

Therefore it follows that "the number of people left on the third border crossing is the same as half of the second, which is the same as a quarter (or a two squaredth) of the first".

$$N(2) = \frac{1}{2}N(1) = \frac{1}{2^2}N(0) \tag{29}$$

So if we think about it like a series,

$$N(x) = \frac{1}{2^1}N(x-1) = \frac{1}{2^2}N(x-2) = \ldots = \frac{1}{2^n}N(0) \tag{30}$$

We can find our general solution. We can then plug in our initial value, $N(0) = 1000$ and find that:

$$N = 1000 * 2^{-x} \tag{31}$$

And it follows (I think) that $\frac{1}{2} = e^{-ln(2)}$ and $\ln(2) \approx 0.69$. [Ca21] There it is: our mystery exponent. I'm sure it won't be long until I've mastered solving that type of equation but in my ignorance I straight up didn't believe this could be right until I saw that number emerge.

5 Conclusion

Well done, Professor Allen. I've never been Nerd Sniped (see fig. 8) this hard in my entire life. This problem didn't nerd snipe me, it 360 no-scoped me [Int09] in the forehead from across the map. Obviously I had a blast, and this was the first time I've had a reason to to whip out my Python knowledge in a couple years. It was really interesting working out how to solve the problem with technology, and it was so rewarding to see the graphs pop up every time I ran the script.

I never expected to run into the issue of discrete vs. continuous case ODEs, partly because I had never heard of them before, but also because I blindly assumed that the linear DE would be an essentially perfect model. I guess the moral of the story is that there's always more math to learn, and it will always get more complex than you think it is currently.

Also, there may be another hidden moral to this story, in that at over 3300 words, this is easily the longest paper I've ever written, and it was fuelled only by a desire to play with math and computers. I, like many other people, tend to work far harder and learn more when they are invested in the outcome. Perhaps it's worth finding a way to get invested in all work, school or otherwise, enough that it feels no different from a hobby.

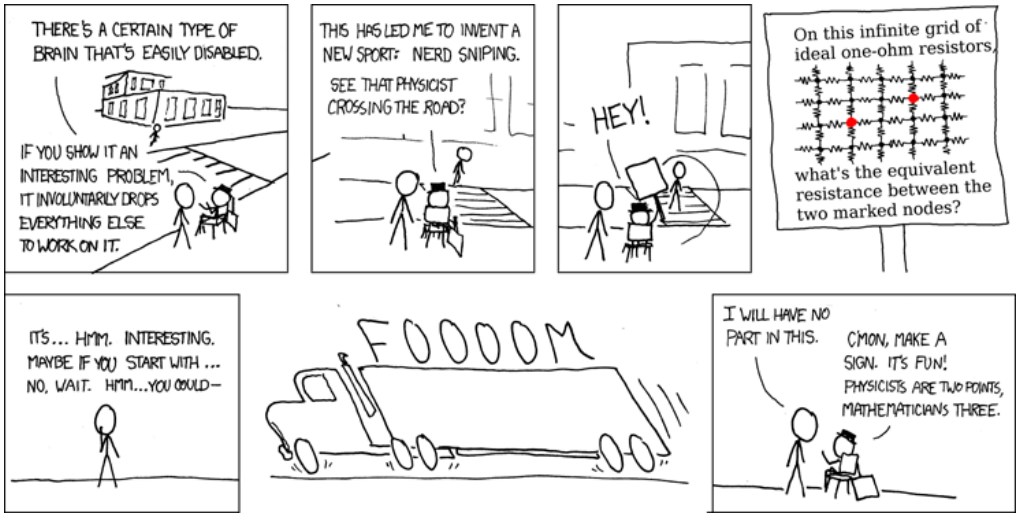

Figure 8: Nerd Sniping (credit XKCD). Caption: "I first saw this problem on the Google Labs Aptitude Test. A professor and I filled a blackboard without getting anywhere. Have fun."[Mun07]

References

[Mun07] Randall Munroe. *Nerd Sniping*. https://xkcd.com/356/. 2007.

[Int09] The Internet. *360 No-Scope*. https://knowyourmeme.com/memes/360-no-scope. 2009.

[ca17] carmichael561 and et. al. *What's the probability of getting 1000 heads in a row?* https://math.stackexchange.com/questions/2308416/. 2017.

[Com21] The United States Playing Card Company. *Beating the Odds in Poker*. https://bicyclecards.com/article the-odds-in-poker/. 2021.

[Cox21] Cory Cox. *populationproblem github repo*. https://github.com/isademigod/populationproblem. 2021.

[Ca21] Cory Cox and et. al. *Why does this data not line up with the differential equation that's supposed to model it?* https://math.stackexchange.com/questions/4258538/. 2021.