1 Problem Statement

From the fivethirtyeight Riddler May 17, 2019.

Consider living army vs an undead living army. They fight in duels only. If the living person wins, they continue on to the next undead. If the undead wins, they take the living and convert them into their army as one of the undead. Suppose each side has a 50% chance of winning any single battle. This continues until the living or undead army is depleted to no members. What is the chance the living army will be victorious? Which size will yield a 50% chance for either side to be victorious?

We can show this with a contour plot more effectively for these larger army sizes. Let's go up to a million, say.

2 Answer

This problem can be completed in many ways. The brute force method of simulation quickly runs into problems when the army sizes become large, but it will give you an answer. It will make it hard (besides guess and check) to see which sizes give approximately equal odds.

The more elegant method is to form $\Pr_a(a, d)$ as the probability that the winning army will win when there are a members in the alive army, and d members in the undead army. We know that $\Pr_a(1, 1) = 0.5$ because we were given this. Then the probability of the live army winnning is simply the probability that they win the next match times $\Pr_a(a, d-1)$ or the probability they lose the next match times $\Pr_a(a-1, d+1)$. This gives the recursion relation

$$\Pr_a(a, d) = p_a \Pr_a(a, d-1) + (1 - p_a) \Pr_a(a-1, d+1) \tag{1}$$

It is well-known that recursion is a great way of destroying your computer's computational capabilities, however. So actually solving the problem by using recursion is usually a bad idea. What you can do instead, is recast the problem as an iterative problem. In our case, that means, start with $\Pr_a(1, 1) = 0.5$, We also happen to know that $\Pr_a(a, 0) = 1$ and $\Pr_a(0, d) = 1$. So we can create a table where the columns correspond to the number of members in the undead army and the rows correspond to the number of members in the alive army. After filling in this table, we note that $\Pr_a(a, d)$ comes from $\Pr_a(a, d-1)$, in the column to the left of $\Pr_a(a, d)$ and from $\Pr_a(a-1, d+1)$ which is the row above and column to the right. This means we should fill in our table row by row from the left column to the right columns so that we always have the values we need.

When we do this, we find

If we create a probability table for all armies possible between 0 and 1000 for both the alive and undead armies, a different pattern emerges from the 2 to 3 times. We see that we need the alive army member number to be approximately the square of the undead army member number (in fact, we need a little more than the square). That is if there are 30 undead, we need a little more than 900 alive to have a decent chance of victory!

We can see this in the figure 1, where we see that the probability of victory for the alive army really dies off quickly. If we do a log-log plot, we can see the square root (the probability=0.5 line has a slope of nearly 2 on our plot which means that $a^2 \approx d$ is required to have even odds) quite easily.

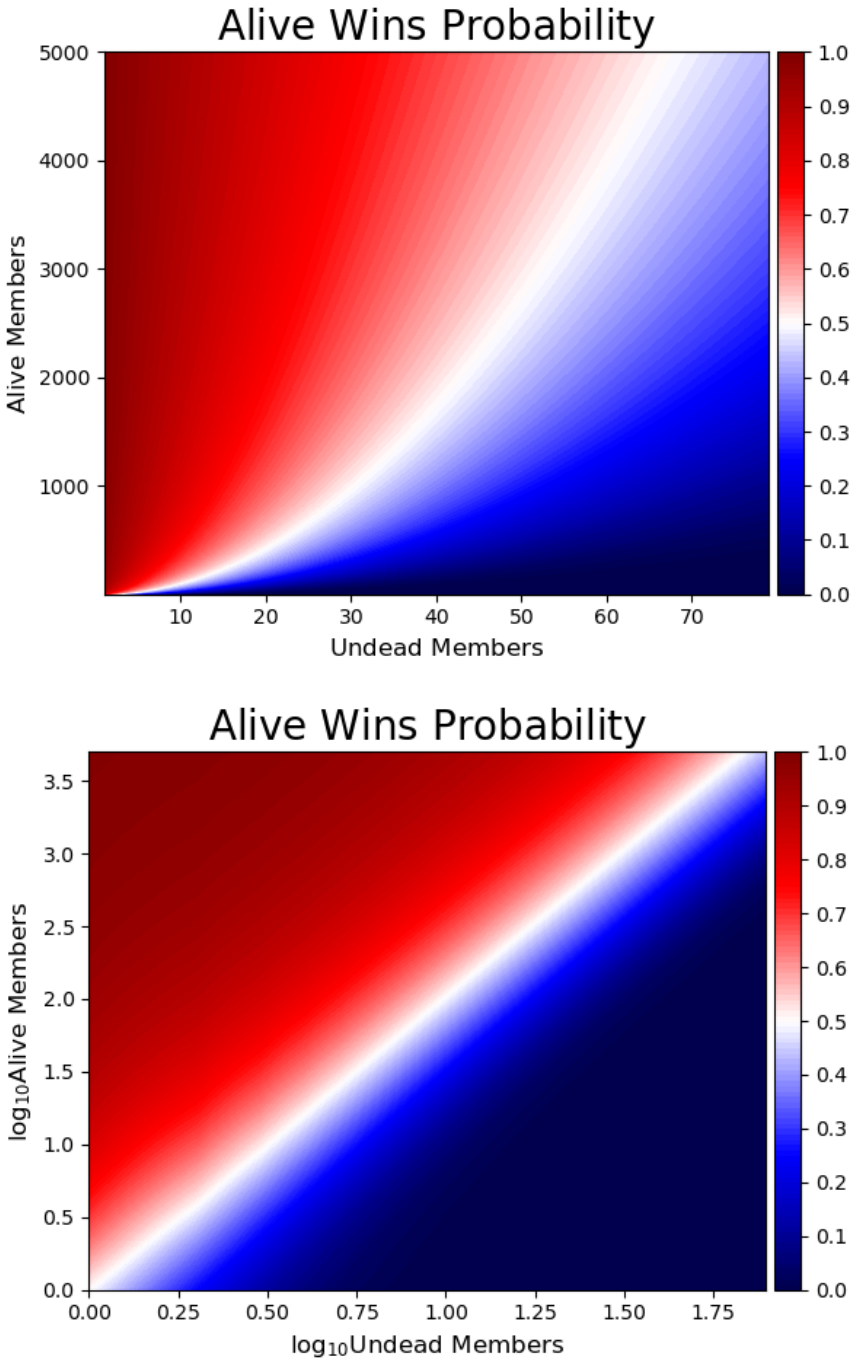

Figure 1: We see the probability as a filled in contour plot for the regular and logarithm (base 10) of army member numbers. What is interesting is that for small army sizes the alive army actually has an even bigger advantage over the undead.

$\frac{d\rightarrow}{a\downarrow}$	1	2	3	4	5	6	7	8	9	10
1	50.0	25.0	12.5	6.2	3.1	1.6	0.8	0.4	0.2	0.1
2	62.5	37.5	21.9	12.5	7.0	3.9	2.1	1.2	0.6	0.3
3	68.8	45.3	28.9	18.0	10.9	6.5	3.9	2.2	1.3	0.7
4	72.7	50.8	34.4	22.7	14.6	9.2	5.7	3.5	2.1	1.3
5	75.4	54.9	38.8	26.7	18.0	11.8	7.7	4.9	3.1	1.9
6	77.4	58.1	42.4	30.2	21.0	14.3	9.6	6.4	4.1	2.7
7	79.1	60.7	45.4	33.2	23.8	16.7	11.5	7.8	5.2	3.5
8	80.4	62.9	48.1	35.9	26.3	18.9	13.4	9.3	6.4	4.3
9	81.5	64.8	50.3	38.3	28.6	21.0	15.2	10.8	7.6	5.2
10	82.4	66.4	52.3	40.5	30.7	23.0	16.9	12.2	8.7	6.1

Table 1: This shows the probability of the alive army being victorious over the undead (all are percentages). The column gives the number in the alive army and the row gives the number of undead in the undead army. Note that only 18 entries have the alive army as more likely to win. Inspecting the chart, you might think the alive army needs to be around 2 to 3 times larger to have an equiprobable chance of victory with these (small) army sizes. Having so few entries is misleading, however.

We can also think about answering the question of how much more skilled would the alive army need to be to compensate for this. We can choose values and find that if $\Pr_a(1,1) \approx 2/3 \approx 0.667$, then we suddenly have an even match for even numbers. Thus, if the alive soldiers are about twice as skilled as the undead, they can get by with even odds when their numbers are equal. This sort of makes sense since if they are twice as skilled, then losing one member to the other side is equivalent in a match of equal skill to simply losing one from your own army. Figure 2 shows this skilled alive army case. It also shows that for small armies, the skilled alive army is even more effective, because the slope of equal probability is less than one for less than 10 undead.[1]

The code below lists a variety of ways of approching the problem. The best way is iteration, as all it requires is a large amount of memory to contain the giant table of values for large army sizes. In the code, I simply refer to the undead as the dead because it's two letters less and that's how I initially programmed it. It's more work to change than it is worth at this point since it doesn't make the problem any less clear to refer to the undead as the dead.

undead_army.py

```
#!/usr/bin/env python2
import numpy as np
from timeit import default_timer as timer
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.ticker as ticker

# Riddler May 17, 2019
# One soldier steps forward from each line and the pair duels - half the time the living soldier wins, half the time the dead soldier wins.
# If the living soldier wins, he goes to the back of his army's line, and the dead soldier is out (the living army uses dragonglass weapons, so the dead soldier is dead forever this time).
# If the dead soldier wins, he goes to the back of their army's line, but this time the (formerly) living soldier joins him there.
# (Reanimation is instantaneous for this Night King.)
# The battle continues until one army is entirely eliminated.
```

[1] This would seem to suggest that if the skilled alive people only ever met groups of 1 to 10 undead at a time, they would overestimate how well they would do against larger armies of the undead.

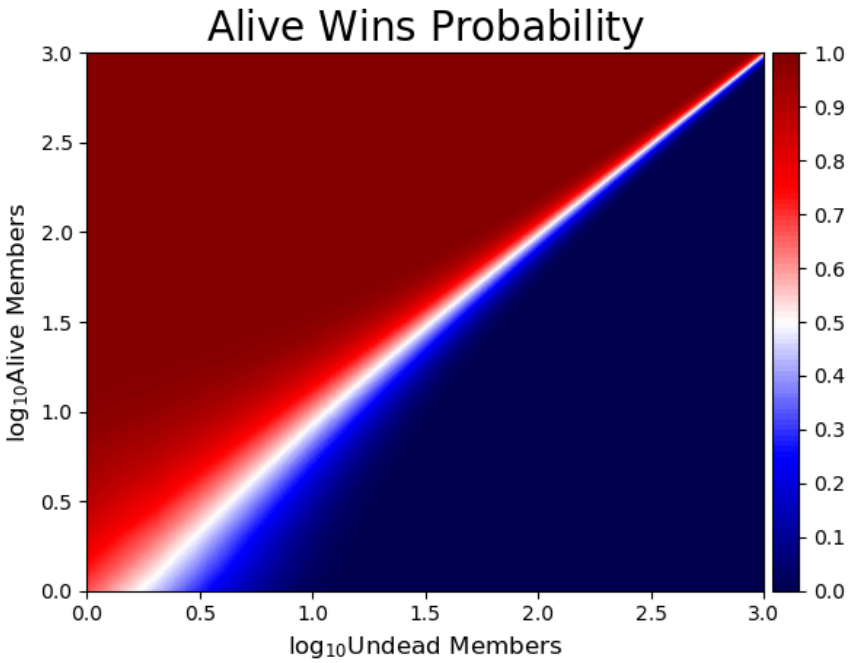

Figure 2: We see the probability as a filled in contour plot for the regular and logarithm (base 10) of army member numbers. Note how only at large equal numbers do we get equal probability. At small equal army numbers the alive actually have a slight advantage.

```
14 # What starting sizes of the armies, living and dead, give each army a 50-50 chance of winning?
15
16 #define variables
17 # a is initial number of alive warriors
18 # d is initial number of dead warriors
19 na=1000
20 nd=1000
21 p_a=0.667
22
23 #much slower Method 1, as expected
24 ##Method 1, direct brute force
25 #start=timer()
26 ## set random seed
27 #np.random.seed(1)
28 #trials=int(1e4)
29 #Awins=0
30 #for j in range(trials):
31 ##form two armies of size N1A and N2D (alive and dead)
32 #  N1A=na
33 #  N2D=nd
34 #  while (N1A*N2D>0):
35 #    aw=np.random.random(1)
36 #    if aw>=p_a:
37 #      N2D=N2D-1
38 #    else:
39 #      N1A=N1A-1
40 #      N2D=N2D+1
41 ##    print("NA",N1A)
42 ##    print("ND",N2D)
43 #  if N2D==0:
44 #    Awins=Awins+1
45 #end=timer()
46 #
47 #print("Alive wins","/","trials",Awins,"/",trials,"=",float(Awins)/float(trials))
48 #print("Method 1 time",(end-start))
```

```
49
50 ## Method 2, brute force with numpy arrays
51 #start=timer()
52 ## set random seed
53 #np.random.seed(1)
54 #trials=int(1e4)
55 ##form two armies of size N1A and N2D (alive and dead)
56 #tones=np.ones(trials)
57 #N1A=na*tones
58 #N2D=nd*tones
59 #while np.sum(N1A*N2D)!=0:
60 #  aw=np.random.random(trials)
61 #  maska=N1A>0
62 #  maskd=N2D>0
63 #  dwin=aw>(1-p_a)
64 #  awin=aw<=p_a
65 #  totmaskd=maska*maskd*dwin
66 #  totmaska=maska*maskd*awin
67 #  N2D[totmaska]=N2D[totmaska]-tones[totmaska]
68 #  N1A[totmaskd]=N1A[totmaskd]-tones[totmaskd]
69 #  N2D[totmaskd]=N2D[totmaskd]+tones[totmaskd]
70 #end=timer()
71 #Awins=np.sum(N1A>N2D)
72 #print('')
73 #print("Alive wins","/","trials",Awins,"/",trials,"=",float(Awins)/float(trials))
74 #print("Method 2 time",(end-start))
75
76 ## Method 3, brute force alternate numpy array method
77 ## Still slower than Method 2
78 #start=timer()
79 ## set random seed
80 #np.random.seed(1)
81 #trials=int(1e4)
82 ##form two armies of size N1A and N2D (alive and dead)
83 #tones=np.ones(trials)
84 #N1A=na*tones
85 #N2D=nd*tones
86 #while np.sum(N1A*N2D)!=0:
87 #  aw=np.random.random(trials)
88 #  totmaskd=(((N1A>0)&(N2D>0))&(aw>(1-p_a)))
89 ##  totmaskd=np.logical_and(np.logical_and(N1A>0,N2D>0),aw>0.5)
90 #  totmaska=(((N1A>0)&(N2D>0))&(aw<=p_a))
91 #  N2D[totmaska]=N2D[totmaska]-tones[totmaska]
92 #  N1A[totmaskd]=N1A[totmaskd]-tones[totmaskd]
93 #  N2D[totmaskd]=N2D[totmaskd]+tones[totmaskd]
94 #end=timer()
95 #Awins=np.sum(N1A>N2D)
96 #print('')
97 #print("Alive wins","/","trials",Awins,"/",trials,"=",float(Awins)/float(trials))
98 #print("Method 3 time",(end-start))
99
100 # Method 4, using math, but incredibly inefficient recursive calls
101 # use this is a recurrence relation
102 # f(a,d)=p_a*f(a,d-1)+(1-p_a)*f(a-1,d+1)
103 # where p_a is probability of a winning (representing a alive people)
104 # and (1-p_a)=p_d is probability of dead winning (reprsenting d dead people)
105 #def f(a,d,p_a=0.5):
106 #  if a==0:
107 #    return 0.
108 #  elif d==0:
109 #    return 1.
110 #  else:
111 #    return p_a*f(a,d-1)+(1-p_a)*f(a-1,d+1)
112 #start=timer()
113 #print('')
114 #print("Win percentage for living,",f(na,nd))
115 #end=timer()
116 #print("Method 4 time",(end-start))
117
118 # Method 5, using math, but iteration instead of recursion
119 # use this is a recurrence relation
```

```
120 # f(a,d)=p_a*f(a,d-1)+(1-p_a)*f(a-1,d+1)
121 # where p_a is probability of a winning (representing a alive people)
122 # and (1-p_a)=p_d is probability of dead winning (reprsenting d dead people)
123 #
124 # note idea is to make a large table and then fill in from the boundaries
125 def f_it(a,d,p_a=0.5):
126   if a<0 or d<0:
127     return "a,d>=0 required"
128   p_d=1-p_a
129   wintable=np.zeros((a+1,a+d+1))
130   wintable[:,0]=1.
131   wintable[0,:]=0.
132   for i in range(1,a+1):
133     for j in range(1,a+d):
134       wintable[i,j]=p_a*wintable[i,j-1]+p_d*wintable[i-1,j+1]
135   return wintable[1:,1:d+1]
```