Zubin Mishra
UID: 604644805
CEE/MAE M20
May 2, 2017

Homework 4

1 SIR Model for Infection

1.1 Introduction

The goal in this problem is to develop a program that will simulate the process of spreading disease. This is accomplished by using the forward Euler method to get the discretized governing equations. The end results are printed out to the screen and a plot is made showing the normalized population vs. time relationship. In part c, the simulation is run again, but with the introduction of a vaccine. It will be discussed whether it is better to introduce a vaccine earlier or to have a quicker acting vaccine.

1.2 Model and Methods

The script first sets up the time-stepping parameters. The time array is constructed from the parameters:

```
t = linspace(0,tFinal,tSteps+1);
```

The script then establishes coefficients and constants. In part b, this is only gamma and beta. In part c, there is also n and t_{vac}. Additionally, in part c, beta is a 1-D array, as it changes with time:

```
beta = zeros(1, tSteps+1);
for k = 1:tSteps+1
    if t(k) <= tvac
        beta(k) = 0.5;
    elseif t(k) > tvac && t(k) <= tvac+n
        beta(k) = 0.275 + 0.225*cos(pi*(t(k)-tvac)/n);
    else
        beta(k) = 0.05;
    end
end
```

Then arrays are created and initial values are established for the normalized susceptible, infected, and removed populations. The script then iterates through a for loop, calculating the next values for s, i, and r using the discretized governing equations and adding them to the 1-D arrays:

```
for k = 1:tSteps
    s(k+1) = s(k)+deltat*(-beta*i(k)*s(k));
    i(k+1) = i(k)+deltat*(beta*i(k)*s(k)-gamma*i(k));
    r(k+1) = r(k)+deltat*(gamma*i(k));
end
```

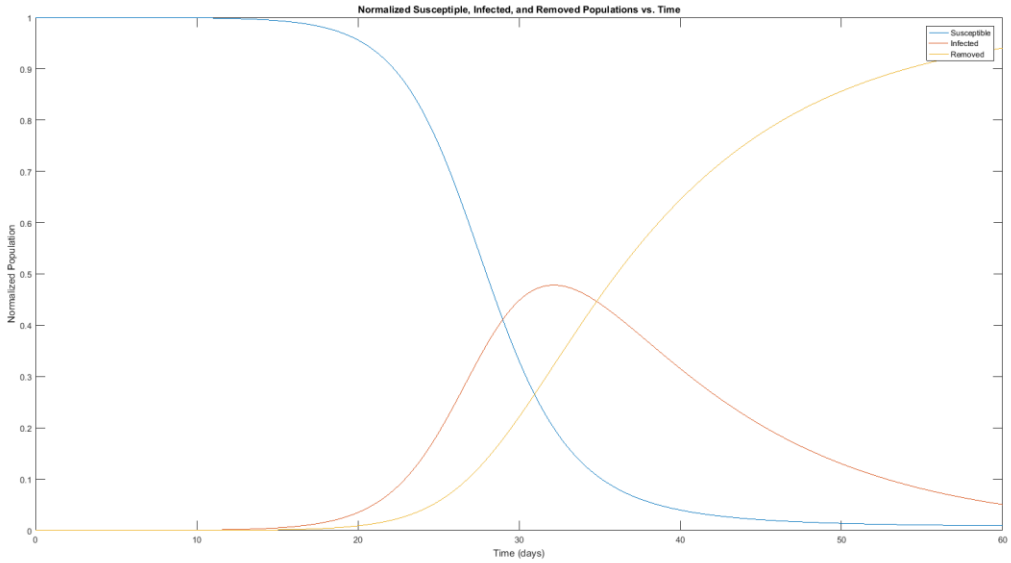
The s, i, and r arrays are then plotted against the time array, and the final non-normalized populations are printed out.

1.3 Results and Calculations

For part b, the following output is printed to the screen:

```
Final Size of Population Groups
Susceptible: 0.04 million
Infected: 0.20 million
Removed: 3.76 million
```

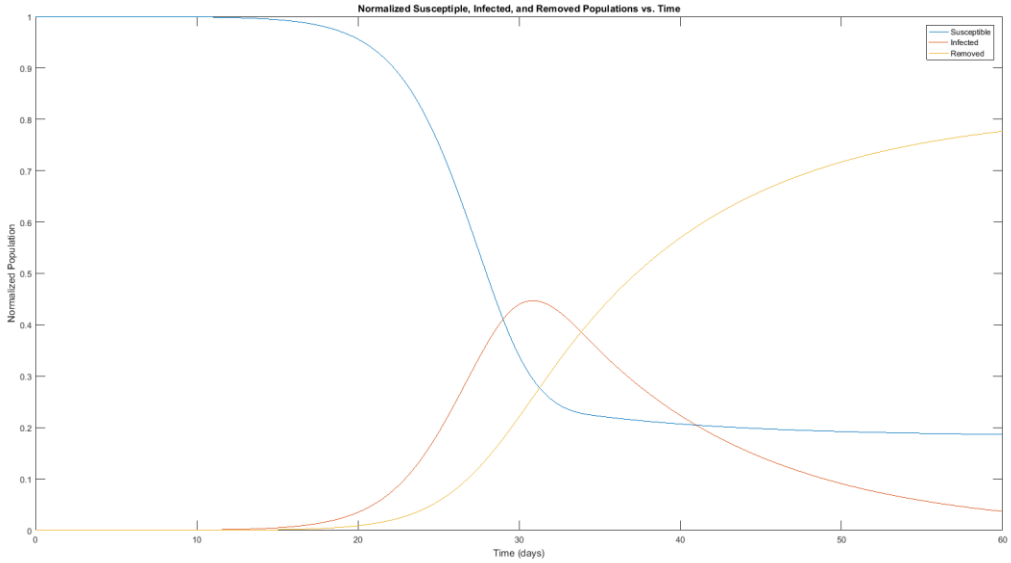
With the following plot:



For part c, the following output is printed to the screen:

```
Final Size of Population Groups
Susceptible: 0.75 million
Infected: 0.15 million
Removed: 3.11 million
```

With the following plot:



When changing n to 6, the following output is printed to the screen:

```
Final Size of Population Groups
Susceptible: 0.83 million
Infected: 0.14 million
Removed: 3.02 million
```

When changing t_{vac} to 27, the following output is printed to the screen:

```
Final Size of Population Groups
Susceptible: 0.94 million
Infected: 0.14 million
Removed: 2.92 million
```

Introducing the vaccine a day earlier led to better results, judging from the final size of the susceptible population.

When n = 1 and t_{vac} = 28, the following output is printed to the screen:

```
Final Size of Population Groups
Susceptible: 1.45 million
Infected: 0.12 million
Removed: 2.43 million
```

When n = 7 and t_{vac} = 4, the following output is printed to the screen:

```
Final Size of Population Groups
Susceptible: 4.00 million
Infected: 0.00 million
Removed: 0.00 million
```

When n = 9 and t_{vac} = 27, the following output is printed to the screen:

```
Final Size of Population Groups
Susceptible: 0.76 million
Infected: 0.15 million
Removed: 3.10 million
```

When n = 60 and t_{vac} = 1, the following output is printed to the screen:

```
Final Size of Population Groups
Susceptible: 1.40 million
Infected: 0.27 million
Removed: 2.33 million
```

The timing for parts b and c were as follows (for one run each; timings varied each run):

b) Elapsed time is 1.118576 seconds.
c) Elapsed time is 1.277267 seconds.

1.4 Discussion

In part c, the results indicate that shortening t_{vac} by one day is better than shortening n by one day. In other words, it is better to introduce a vaccine earlier than to have it act more quickly. From my test cases, the most effective treatment will minimize both t_{vac} and n, but prioritize minimizing t_{vac}.

The runtimes for parts b and c were very similar. They both ranged from ~1.1 - ~1.4 seconds. From these results, it appears the additional for loop for beta in part c does not affect the timing by much.

Across all my test cases, the number of infected increased from 0, peaked, and then decreased back towards 0. The number of susceptible decreased and then leveled out, and the number of removed increased and then leveled out. These trends were more clear when expanding the timeline past 60 days.

2 DNA Analysis

2.1 Introduction

The goal in this problem is to develop a program that takes a strand of DNA (represented as a 1-D array) and calculates the lengths of every protein-coding segment in the strand. The total number of protein-coding segments, the average coding segment length, the maximum coding segment length, and the minimum coding segment length are printed out to the screen. The frequency of different stop codons and the percentage of DNA that actually codes for proteins will be discussed.

2.2 Model and Methods

The script begins by loading the DNA array:

```
load('chr1_sect.mat');
```

The script then establishes all initial values needed. It assumes a stop has already been found so it will begin by looking for a start codon and creates an empty array to hold the protein-coding segment lengths. The script then goes through a for loop starting at 1 and going to the length of the dna array, incrementing by 3.

The for loop begins by checking if a start needs to be found and if a start is found. If true, the start position and the fact that a start was found are stored:

```
if stop_found && dna(i) == 1 && dna(i+1) == 4 && dna(i+2) == 3
        start = i;
        stop_found = false;
    end
```

The for loop then checks if a stop needs to be found and if a stop is found:

```
if ~stop_found && dna(i) == 4
      if (dna(i+1) == 1 && (dna(i+2) == 1 || dna(i+2) == 3)) || (dna(i+1)
            == 3 && dna(i+2) == 1)
```

If true, different variables are incremented to keep track of which stop codon is found. The stop position and the fact that a stop was found are then stored and the protein-coding segment length is calculated and added to the lengths array:

```
stop = i + 2;
stop_found = true;
lengths(length(lengths)+1) = stop - start + 1;
```

After the for loop completes, the results are printed out, using the length, mean, max, and min functions.

2.3 Results and Calculations

When the script is run, the following output is printed to the screen:

```
Total Protein-Coding Segments: 4366
Average Length: 84.12
Maximum Length: 1857
Minimum Length: 6
```

The stop codon counts are as follows:

taa = 1466
tag = 918
tga = 1982

The percentage of the DNA that is directly used in the protein-coding process is calculated by summing the values of the protein-coding segment lengths array and dividing by the length of the dna array:

Percentage = 29.11%

2.4 Discussion

From the results of the script, the percentage of DNA that is directly used in the protein-coding process was calculated to be 29.11%. This means that most of the DNA sequence has no role in making proteins. The most frequently used stop codon was TGA. The least frequently used stop codon was TAG.

In order to identify start and stop codons that nearly, but not exactly, match an expected pattern, I would implement a count of correct bases for each codon being examined. For example, if the codon ACG was found when looking for a start codon, the code would look at A, see that it matches ATG, and add 1 to the count of correct bases. It would then look at C (no match, count left the same) and then G (match, count incremented). If the count of correct bases is 2, then that codon is a near match for a start codon, as is the case with ACG. It would be similar in the case of stop codons, but each stop codon would have its own count of correct bases. If at least one of the three stop codon counts are 2 and none are 3, then there is a near match for a stop codon. For example TAC would have counts of 2, 2, and 1 for TAA, TAG, and TGA respectively, and so would be considered a near match.