Assignment8

Richard McCormick

March 30th, 2022

Here we explore the stochastic transmission of a pathogen within a host meta-population, in which distinct host sub-populations are connected by migration. In a single host population, i, the model is the standard XYZ. We then add migration, where m is the per-capita migration rate, $p_{i,j}$ is the probability of hosts moving from sub-population j to i, and P is the total number of sub-populations in the meta-population:

$$\frac{dX_i}{dt} = -\frac{\beta}{N_i} X_i Y_i - mX_i + \sum_{j \neq i}^{P} p_{i,j} m X_j$$

$$\frac{dY_i}{dt} = \frac{\beta}{N_i} X_i Y_i - \gamma Y_i - mY_i + \sum_{j \neq i}^{P} p_{i,j} m Y_j$$

$$\frac{dZ_i}{dt} = \gamma Y_i - mZ_i + \sum_{j \neq i}^{P} p_{i,j} m Z_j$$

Here we will assume that the probability of hosts moving between specific sub-populations depends upon the distance between host populations. We will use a simple dispersal kernel of the form:

$$p_{i,j} = \frac{1}{\exp\left(\frac{d_{i,j}}{\theta}\right)},$$

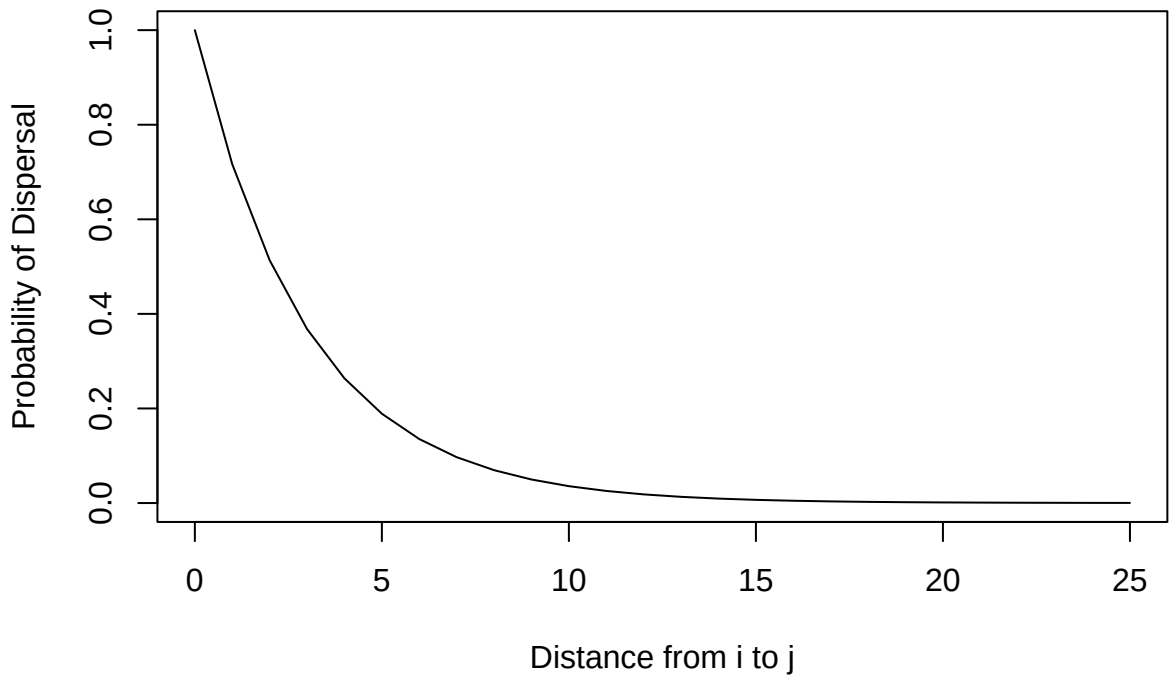
where $d_{i,j}$ is the distance between populations i and j in kilometers, and θ is a constant that determines the likelihood of long long-distance dispersal events.

Task 1 (5 points)

Create a graph that shows the probability of dispersing as a function of distance between host populations. Assume $d_{i,j}$ ranges from 0 to 25km, and $\theta = 3$.

```
d_ij = seq(0, 25, by=1)
theta = 3
p_vals = rep(0, 25)

p_func = function(d_ij, theta){
  p_ij = 1/(exp(d_ij/theta))
  return(p_ij)}

for (i in 0:length(d_ij))
{
  p_vals[i] = p_func(d_ij[i], theta)
}
```

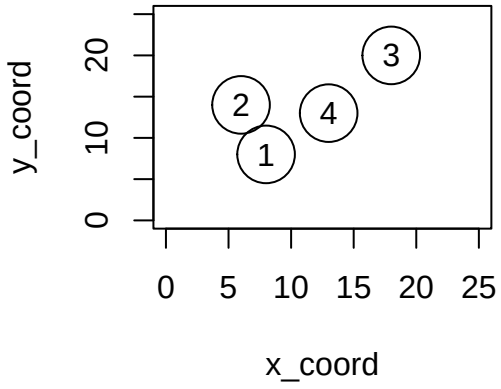
```r
plot(NA,NA,
     xlim = c(0, max(d_ij)),
     ylim = c(0, 1),
     xlab = "Distance from i to j",
     ylab = "Probability of Dispersal")

lines(p_vals ~ d_ij)
```



Setting up the host meta-population

We will simulate the location of four sub-populations and calculate the $d_{i,j}$ and $p_{i,j}$.

```
# Number of populations
n_pop = 4

# Create x-y coordinates
x_coord = c(8,6,18,13)
y_coord = c(8,14,20,13)
pop_ID = c(1:n_pop)

# Plot the 'map'
plot(y_coord ~ x_coord,
     xlim = c(0, 25), ylim = c(0, 25),
     pch = 1, cex = 4)
text(y_coord ~ x_coord, labels = pop_ID)
```




```
# Calculate pair-wise distance, based on x-y coords
# Create a distance matrix:
xy_mat = cbind(x_coord, y_coord)
dist_mat = dist(xy_mat, diag = TRUE, upper = TRUE)
dist_mat = as.matrix(dist_mat)

# Probability of movement
prob_move = matrix(0, n_pop, n_pop)
theta = 3

for(i in 1:n_pop){
  for(j in 1:n_pop){
    if(i != j){
      prob_move[i, j] = 1 / exp(dist_mat[i,j] / theta)
    }
  }
}
```

```
prob_move
```

```
##             [,1]       [,2]        [,3]       [,4]
## [1,] 0.000000000 0.12145820 0.005478998 0.09470177
## [2,] 0.121458200 0.00000000 0.011422891 0.09470177
## [3,] 0.005478998 0.01142289 0.000000000 0.05684416
## [4,] 0.094701769 0.09470177 0.056844162 0.00000000
```

Tau-leaping

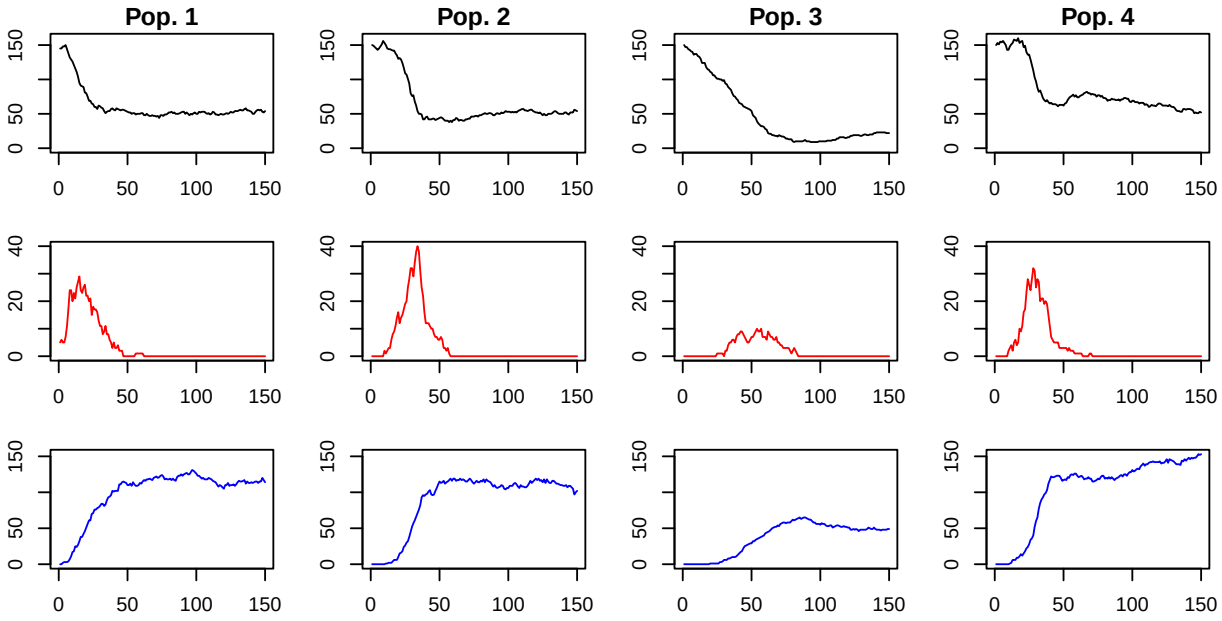
Previously, we learned about adding demographic stochasticity to the XYZ model by using Gillespie's Direct Algorithm (GDA). The GDA algorithm is great, but it becomes tedious and slow if we have more than 2 events. A more flexible and efficient algorithm is the tau-leaping algorithm. In this algorithm, for each possible event, we calculate how many events occur in a given time step, τ. In other words, we want to know how many of each event will occur in the period from t to $t + \tau$. Read this Wikipedia post for more details and to help you better understand the code.

```
#########################################################
# FUNCTION #
#########################################################

tau_leap_1step = function(counter, tau, pop_N, x_mat, y_mat, z_mat,
                          x_move, y_move, z_move, this_pop, beta, gamma, m){
  # Events: Transmission, Recovery, Migrate:X, Migrate:Y, Migrate:Z
  n_events = 5

  if(pop_N[this_pop] > 0){

    # Scale beta by population size
    beta_scaled = beta / pop_N[this_pop]

    # Event probabilities:
    event_rate = vector(mode = "numeric", length = n_events)
    ## Transmission:
    event_rate[1] = beta_scaled * x_mat[counter, this_pop] * y_mat[counter, this_pop]
    ## Recovery:
    event_rate[2] = gamma * y_mat[counter, this_pop]
    ## Migrate: X
    event_rate[3] = m * x_mat[counter, this_pop]
    ## Migrate: Y
    event_rate[4] = m * y_mat[counter, this_pop]
    ## Migrate: Z
    event_rate[5] = m * z_mat[counter, this_pop]

    # How many events of each type will occur over time period tau
    n_occur = vector(mode = "numeric", length = n_events)

    for(i in 1:n_events){
      # Poisson random variate
      n_occur[i] = rpois(1, event_rate[i] * tau)
    }

  }else{
    n_occur = rep(0, n_events)
  }

  return(n_occur)

}
```

```r
#######################################################
# SET-UP #
#######################################################

# Time
tau = 1 # leap size (arbitrary)
t_max = 150
n_times = floor(t_max / tau)
t_vec = seq(1, t_max, length.out = n_times)

# Population sizes
pop_N = vector(mode = "numeric", length = n_pop)
pop_N = rep(150, n_pop) # each pop size is equal

# Sizes of X, Y, and Z for each population
x_mat = matrix(0, nrow = n_times, ncol = n_pop)
y_mat = matrix(0, nrow = n_times, ncol = n_pop)
z_mat = matrix(0, nrow = n_times, ncol = n_pop)

# Sizes of X, Y, and Z MIGRANTS for each population
## How many move at each time step
x_move = vector(mode = "numeric", length = n_pop)
y_move = vector(mode = "numeric", length = n_pop)
z_move = vector(mode = "numeric", length = n_pop)

# INITIAL CONDITIONS
## Seed population 1 with infected individuals
y_mat[1,1] = 5
x_mat[1,] = pop_N - y_mat[1,] # this is a vectorized function

# Parameters
beta = 0.35
gamma = 1/5
m = 1/50

#######################################################
# Run one realization of the system
#######################################################

# counter
counter = 1

for(t in 1:(n_times-1)){

  for(i in 1:n_pop){

    this_pop = i
    n_occur = tau_leap_1step(counter, tau, pop_N, x_mat, y_mat, z_mat,
                             x_move, y_move, z_move, this_pop, beta, gamma, m)

    #-------------------------------------------------------
    # Update your populations:
    ## Events: Transmission, Recovery, Migrate-X, Migrate-Y, Migrate-Z
```

```
  x_mat[(counter+1), this_pop] = x_mat[counter, this_pop] -
                                    n_occur[1] - n_occur[3]
  y_mat[(counter+1), this_pop] = y_mat[counter, this_pop] +
                                    n_occur[1] - n_occur[2] - n_occur[4]
  z_mat[(counter+1), this_pop] = z_mat[counter, this_pop] +
                                    n_occur[2] - n_occur[5]

  x_move[this_pop] = n_occur[3]
  y_move[this_pop] = n_occur[4]
  z_move[this_pop] = n_occur[5]

} # end i loop

#---------------------------------------------------------
#---------------------------------------------------------
# Disperse the migrants into new populations, based on distance

# Note: the multinomial distribution helps us randomly determine
# how many individuals will move to a particular new location,
# based on the probability of moving

for(j in 1:n_pop){ # Move from j to i

  if(x_move[j] > 0){
    move_x = rmultinom(1, size = x_move[j], prob = prob_move[,j])
    for(i in 1:n_pop){
      x_mat[(counter+1),i] = x_mat[(counter+1),i] + move_x[i]
    }
  }

  if(y_move[j] > 0){
    move_y = rmultinom(1, size = y_move[j], prob = prob_move[,j])
    for(i in 1:n_pop){
      y_mat[(counter+1),i] = y_mat[(counter+1),i] + move_y[i]
    }
  }

  if(z_move[j] > 0){
    move_z = rmultinom(1, size = z_move[j], prob = prob_move[,j])
    for(i in 1:n_pop){
      z_mat[(counter+1),i] = z_mat[(counter+1),i] + move_z[i]
    }
  }

} # end j loop

# Advance the counter:
counter = counter + 1

# Check to see if any are zero
for(i in 1:n_pop){
  if(x_mat[counter,i] < 0) x_mat[counter,i] = 0
  if(y_mat[counter,i] < 0) y_mat[counter,i] = 0
```

```
    if(z_mat[counter,i] < 0) z_mat[counter,i] = 0
  }

  # Recalculate population size (because of migrants!)
  for(i in 1:n_pop){
    pop_N[i] = x_mat[counter, i] + y_mat[counter, i] + z_mat[counter, i]
  }

}

#-------------------------------------------------------
# Plot:
# This sets up a plotting grid
par(mfrow=c(3,n_pop), # 3 rows, n_pop columns
    mai = c(0.3,0.3,0.2,0.2))

for(i in 1:n_pop){
  plot(x_mat[,i] ~ t_vec, type = "l",
     xlim = c(0, t_max), ylim = c(0, max(x_mat)),
     xlab = "", ylab = "", main = paste0("Pop. ", i))
}
for(i in 1:n_pop){
  plot(y_mat[,i] ~ t_vec, type = "l", col = "red",
     xlim = c(0, t_max), ylim = c(0, max(y_mat)),
     xlab = "", ylab = "")
}
for(i in 1:n_pop){
  plot(z_mat[,i] ~ t_vec, type = "l", col = "blue",
     xlim = c(0, t_max), ylim = c(0, max(z_mat)),
     xlab = "", ylab = "")
}
```



Task 2 (15 points)

Write a brief paragraph that summarizes what this algorithm is doing. At the end of your paragraph, specifically address why we expect the total population size of population 3 to (very likely) decline over time.

This algorithm is creating an SIR model for a disease between 4 populations, and modelling migration and demography between and within those populations. For each population, the number (not fraction) of susceptible, infected, and recovered are all graphed over time. A tau-leaping algorithm is used to randomly simulate the chances of infection, recovery, and death.

Population 3's population will continue to decline over time because they are too far away from other populations to see significant immigration, and their death rate exceeds their birth rate. Ultimately, which people in Population 3 may choose to migrate elsewhere, few people elsewhere will choose to migrate to Population 3, leading to a population decline.

Task 3 (45 points)

Conduct a simulation experiment to understand how a population's distance from the epicenter population affects the timing of its outbreak. (The epicenter population is the one where the first infection occurred). I will give you some guidance here, but generally I want you to devise a way to do this.

1. In each run of your simulation, change the spatial position of population 3 so that it is more or less distant from the epicenter (population 1).
2. Each time you change the position of population 3, run the algorithm and store *when* the outbreak of population 3 reaches its maximum.
3. Create a figure that shows the effect of the distance between populations 1 and 3 on the time of population 3's peak outbreak.
4. Describe the pattern in your figure and why it does or does not make sense. (2-3 sentences)

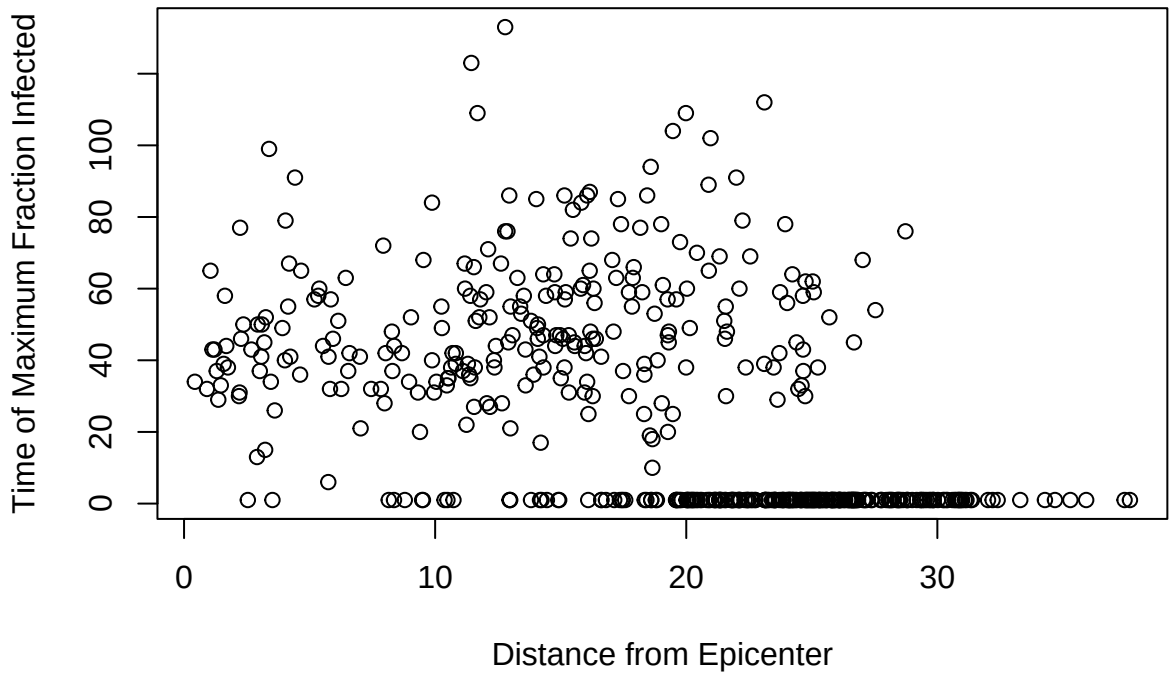
```
#####################################################
# FUNCTION #
#####################################################

tau_leap_1step = function(counter, tau, pop_N, x_mat, y_mat, z_mat,
                          x_move, y_move, z_move, this_pop, beta, gamma, m){
  # Events: Transmission, Recovery, Migrate:X, Migrate:Y, Migrate:Z
  n_events = 5

  if(pop_N[this_pop] > 0){

    # Scale beta by population size
    beta_scaled = beta / pop_N[this_pop]

    # Event probabilities:
    event_rate = vector(mode = "numeric", length = n_events)
    ## Transmission:
    event_rate[1] = beta_scaled * x_mat[counter, this_pop] * y_mat[counter, this_pop]
    ## Recovery:
    event_rate[2] = gamma * y_mat[counter, this_pop]
    ## Migrate: X
    event_rate[3] = m * x_mat[counter, this_pop]
    ## Migrate: Y
    event_rate[4] = m * y_mat[counter, this_pop]
```

```
    ## Migrate: Z
    event_rate[5] = m * z_mat[counter, this_pop]

    # How many events of each type will occur over time period tau
    n_occur = vector(mode = "numeric", length = n_events)

    for(i in 1:n_events){
      # Poisson random variate
      n_occur[i] = rpois(1, event_rate[i] * tau)
    }

  }else{
    n_occur = rep(0, n_events)
  }

  return(n_occur)

}

n_runs = 500

distance_between_pop1_pop3 = vector(mode = "double")
time_max_infected_pop3 = vector(mode = "numeric")

for (i in 1:n_runs){
  ######################################################
  # SET-UP #
  ######################################################
  # Number of populations
  n_pop = 4

  # Create x-y coordinates
  # Randomizes coordinates for population 3
  x_coord = c(8,6,runif(1, min=5, max=35),13)
  y_coord = c(8,14,runif(1, min=5, max=35),13)
  pop_ID = c(1:n_pop)

  # Calculate pair-wise distance, based on x-y coords
  # Create a distance matrix:
  xy_mat = cbind(x_coord, y_coord)
  dist_mat = dist(xy_mat, diag = TRUE, upper = TRUE)
  dist_mat = as.matrix(dist_mat)

  # Probability of movement
  prob_move = matrix(0, n_pop, n_pop)
  theta = 3

  for(i in 1:n_pop){
    for(j in 1:n_pop){
      if(i != j){
        prob_move[i, j] = 1 / exp(dist_mat[i,j] / theta)
      }
    }
```

```
}


# Time
tau = 1 # leap size (arbitrary)
t_max = 150
n_times = floor(t_max / tau)
t_vec = seq(1, t_max, length.out = n_times)

# Population sizes
pop_N = vector(mode = "numeric", length = n_pop)
pop_N = rep(150, n_pop) # each pop size is equal

# Sizes of X, Y, and Z for each population
x_mat = matrix(0, nrow = n_times, ncol = n_pop)
y_mat = matrix(0, nrow = n_times, ncol = n_pop)
z_mat = matrix(0, nrow = n_times, ncol = n_pop)

# Sizes of X, Y, and Z MIGRANTS for each population
## How many move at each time step
x_move = vector(mode = "numeric", length = n_pop)
y_move = vector(mode = "numeric", length = n_pop)
z_move = vector(mode = "numeric", length = n_pop)

# INITIAL CONDITIONS
## Seed population 1 with infected individuals
y_mat[1,1] = 5
x_mat[1,] = pop_N - y_mat[1,] # this is a vectorized function

# Parameters
beta = 0.35
gamma = 1/5
m = 1/50

#######################################################
# Run one realization of the system
#######################################################

# counter
counter = 1

for(t in 1:(n_times-1)){

  for(i in 1:n_pop){

    this_pop = i
    n_occur = tau_leap_1step(counter, tau, pop_N, x_mat, y_mat, z_mat,
                             x_move, y_move, z_move, this_pop, beta, gamma, m)

    #-------------------------------------------------------
    # Update your populations:
    ## Events: Transmission, Recovery, Migrate-X, Migrate-Y, Migrate-Z
```

```
    x_mat[(counter+1), this_pop] = x_mat[counter, this_pop] -
                                    n_occur[1] - n_occur[3]
    y_mat[(counter+1), this_pop] = y_mat[counter, this_pop] +
                                    n_occur[1] - n_occur[2] - n_occur[4]
    z_mat[(counter+1), this_pop] = z_mat[counter, this_pop] +
                                    n_occur[2] - n_occur[5]

    x_move[this_pop] = n_occur[3]
    y_move[this_pop] = n_occur[4]
    z_move[this_pop] = n_occur[5]

  } # end i loop

  #---------------------------------------------------------
  #---------------------------------------------------------
  # Disperse the migrants into new populations, based on distance

  # Note: the multinomial distribution helps us randomly determine
  # how many individuals will move to a particular new location,
  # based on the probability of moving

  for(j in 1:n_pop){ # Move from j to i

    if(x_move[j] > 0){
      move_x = rmultinom(1, size = x_move[j], prob = prob_move[,j])
      for(i in 1:n_pop){
        x_mat[(counter+1),i] = x_mat[(counter+1),i] + move_x[i]
      }
    }

    if(y_move[j] > 0){
      move_y = rmultinom(1, size = y_move[j], prob = prob_move[,j])
      for(i in 1:n_pop){
        y_mat[(counter+1),i] = y_mat[(counter+1),i] + move_y[i]
      }
    }

    if(z_move[j] > 0){
      move_z = rmultinom(1, size = z_move[j], prob = prob_move[,j])
      for(i in 1:n_pop){
        z_mat[(counter+1),i] = z_mat[(counter+1),i] + move_z[i]
      }
    }

  } # end j loop

  # Advance the counter:
  counter = counter + 1

  # Check to see if any are zero
  for(i in 1:n_pop){
    if(x_mat[counter,i] < 0) x_mat[counter,i] = 0
    if(y_mat[counter,i] < 0) y_mat[counter,i] = 0
```

```
      if(z_mat[counter,i] < 0) z_mat[counter,i] = 0
    }

    # Recalculate population size (because of migrants!)
    for(i in 1:n_pop){
      pop_N[i] = x_mat[counter, i] + y_mat[counter, i] + z_mat[counter, i]
    }

  }

  x = dist_mat[1,3]
  distance_between_pop1_pop3 <- append(distance_between_pop1_pop3, x)

  y = which.max(y_mat[,3])
  time_max_infected_pop3 <- append(time_max_infected_pop3, y)
}

plot(NA,NA,
     xlim = c(min(distance_between_pop1_pop3), max(distance_between_pop1_pop3)),
     ylim = c(min(time_max_infected_pop3), max(time_max_infected_pop3)),
     xlab = "Distance from Epicenter",
     ylab = "Time of Maximum Fraction Infected")

points(distance_between_pop1_pop3, time_max_infected_pop3)
```



This model shows that as the distance from the epicenter increases, the time of maximum

fraction infected also increases. After a certain point, the time of maximum infection drops to zero, essentially meaning that after a certain distance the virus fails to spread to Population 3 entirely. This is in line with the theory that increasing the distance between two populations will decrease the chance of trasmission between those communities.