Lab 7: Stochastic Compartmental Models and Monte Carlo Intervals

Sample SOLUTIONS

In your previous lab, you wrote a function for a stochastic SIR model, and plotted the results of two realizations of that stochastic process. While two realizations let us visualize how the stochastic nature of these models can lead to different results, in practice, we will want to (1) run more than two simulations to understand the variability in the process of interest, (2) reshape simulation results from wide form to long form more efficiently for plotting, (3) summarize that variability using some kind of confidence interval, and (4) create plots to visualize our results.

SIR model

The difference equations for the stochastic version of the SIR model are given below:

$$S_t = S_{t-1} - I_t^{(SI)}$$
$$I_t = I_{t-1} + I_t^{(SI)} - R_t^{(IR)}$$
$$R_t = R_{t-1} + R_t^{(IR)}$$

Remember now we have random variables in our difference equations:

- $I_t^{(SI)}$: number of newly infectious individuals at time t
- $R_t^{(IR)}$: number of newly recovered individuals at time t

We will use the chain binomial structure that we discussed in the previous class, which just means that

- $I_t^{(SI)} \sim Binomial(S_{t-1}, \lambda_t)$
- $R_t^{(IR)} \sim Binomial(I_{t-1}, \rho)$, where $\rho = 1 - e^{-1/D}$

Below I have included a version of the function for the stochastic SIR simulation, which should be similar to the function that you wrote in your previous lab.

```r
SIR_simulation <- function(N, S0, I0, RN, D, Time, wideform=FALSE){
  SIR_df <- data.frame(time=0:Time,
                       S=rep(NA, Time+1),
                       I=rep(NA, Time+1),
                       R=rep(NA, Time+1),
                       I_SI=rep(NA, Time+1),
                       R_IR=rep(NA, Time+1),
                       lambda_t=rep(NA, Time+1))
  SIR_df$S[1] <- S0
  SIR_df$I[1] <- I0
  SIR_df$R[1] <- N-S0-I0

  rho <- 1-exp(-1/D)

  for (t in 2:(Time+1)){
    SIR_df$lambda_t[t] <- lambda_t_fcn(RN=RN, D=D, I_i=SIR_df$I[t-1], N=N)
    SIR_df$I_SI[t] <- rbinom(n=1, size=SIR_df$S[t-1], prob=SIR_df$lambda_t[t])
    SIR_df$R_IR[t] <- rbinom(n=1, size=SIR_df$I[t-1], prob=rho)
    SIR_df$S[t] <- SIR_df$S[t-1]-SIR_df$I_SI[t]
    SIR_df$I[t] <- SIR_df$I[t-1]+SIR_df$I_SI[t]-SIR_df$R_IR[t]
    SIR_df$R[t] <- SIR_df$R[t-1]+SIR_df$R_IR[t]
  }

  if (wideform==TRUE){
    return(SIR_df)
  }
  else{
      return(data.frame(time=rep(0:Time, 3),
                        compartment=rep(c("S","I", "R"), each=(Time+1)),
                        count=c(SIR_df$S, SIR_df$I, SIR_df$R)))
  }
}
```

Wide form/long form

(a) One difference between the above function and what you wrote is the inclusion of the variable `wideform` as an argument in the function. By default, it is set to FALSE, and in this case should return a data frame with 3 columns that is hard to read, but facilitates plotting in ggplot2. Confirm that this is what it returns by running the code below.

```
SIR_sim1 <- SIR_simulation(N=1000, S0=999, I0=1, RN=4, D=3, Time=28, wideform=FALSE)
head(SIR_sim1)
```

```
##   time compartment count
## 1    0           S   999
## 2    1           S   998
## 3    2           S   996
## 4    3           S   993
## 5    4           S   988
## 6    5           S   976
```

(b) What happens when you set `wideform=TRUE` (just copy the code from lines 83-84 and change FALSE to TRUE and rename your object as SIR__sim1__wide). Use `head` to print the first 6 lines - what do they look like?

```
SIR_sim1_wide <- SIR_simulation(N=1000, S0=999, I0=1, RN=4, D=3, Time=28, wideform=TRUE)
head(SIR_sim1_wide)
```

```
##   time   S  I R I_SI R_IR    lambda_t
## 1    0 999  1 0   NA   NA          NA
## 2    1 998  1 1    1    1 0.001332445
## 3    2 997  2 1    1    0 0.001332445
## 4    3 992  7 1    5    0 0.002663114
## 5    4 987 11 2    5    1 0.009289913
## 6    5 972 23 5   15    3 0.014559635
```

(c) What happens if you omit `wideform` from the list of arguments all together? Why?

```r
SIR_sim1 <- SIR_simulation(N=1000, S0=999, I0=1, RN=4, D=3, Time=28, wideform=FALSE)
head(SIR_sim1)
```

```
##   time compartment count
## 1    0           S   999
## 2    1           S   995
## 3    2           S   993
## 4    3           S   983
## 5    4           S   964
## 6    5           S   921
```

It returns the data in long format - the default argument for `wideform` is `FALSE`, so if it is not specified, it defaults to FALSE. Failing to specify an argument only works in cases where there is a default. Otherwise you should expect an error.

You will get to explore other ways of reshaping data in your homework.

Running and storing multiple realizations (iterations) of the simulation

(d) We want run multiple simulations, perhaps hundreds. What are some ways we could think about using our existing function to do this? Let's brainstorm together and try some things out.

Solution with arrays for storage

```r
TotalSim <- 10
Time <- 28
Vars <- 7

SIR_sim_array <- array(data=NA, dim=c(Time+1, Vars, TotalSim),
                       dimnames=list(NULL,
                                     c("time", "S", "I", "R", "S_IS", "R_IR", "lambda_t"),
                                     NULL))

for (i in 1:TotalSim){
  set.seed(123+i)
  SIR_sim_df <- SIR_simulation(N=1000, S0=999, I0=1, RN=4, D=3, Time=28, wideform=TRUE)
  SIR_sim_array[,,i] <- as.matrix(SIR_sim_df) ## convert data frame to matrix
}
```

Solution with lists for storage

```r
SIR_sim_list <- list()

for (i in 1:TotalSim){
  set.seed(123+i)
  SIR_sim_df <- SIR_simulation(N=1000, S0=999, I0=1, RN=4, D=3, Time=28, wideform=TRUE)
  SIR_sim_list[[i]] <- SIR_sim_df
}
```

Monte Carlo confidence intervals

Loosely, Monte Carlo (MC) intervals can be obtained from many simulations, and can be used to obtain confidence intervals. They can be particularly useful in cases where analytic confidence intervals (like those obtained based on a t-distribution, for example), are not available, or when the assumptions for calculating such intervals are not satisfied, making them unreliable. To calculate these intervals, we will need the `quantile` function from R.

(e) Run the following example to (re)-familiarize yourself with this function.

```r
## returns 2.5 and 97.5 percentiles
quantile(x=1:10, probs=c(0.025, 0.975))
```

```
##  2.5% 97.5%
## 1.225 9.775
```

(f) Modify the code in (e) to return the 5th and 95th percentiles.

```r
## returns 5 and 95 percentiles
quantile(x=1:10, probs=c(0.05, 0.95))
```

```
##   5%  95%
## 1.45 9.55
```

(g) How can we use this on our simulation results from (d)? Let's brainstorm together and try some things out.

We know that the quantile function works with numeric vectors, and we know that we want to create MC intervals for each of the compartments, S, I, and R. To create the MC intervals, we need to have a numeric vector at each time point for each compartment. As a first attempt, let's try the following, using our **array storage option**.

Initial thought – can we pick off all the time=0 values? Note, `SIR_sim_array[this corresponds to time, this corresponds to the columns, this corresponds to the simulation number]`

```r
## using array storage
## gives a 7x10 matrix (7 for the seven columns from our array, 10 for the 10 simulations I ran)
SIR_sim_array[1,,]
```

```
##         [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## time       0    0    0    0    0    0    0    0    0     0
## S        999  999  999  999  999  999  999  999  999   999
## I          1    1    1    1    1    1    1    1    1     1
## R          0    0    0    0    0    0    0    0    0     0
## S_IS        NA    NA    NA    NA    NA    NA    NA    NA    NA     NA
## R_IR        NA    NA    NA    NA    NA    NA    NA    NA    NA     NA
## lambda_t    NA    NA    NA    NA    NA    NA    NA    NA    NA     NA
```

```
## gives a 1x10 matrix (the second row of what we have in line 170 -- all the S's at time 0)
SIR_sim_array[1,2,]
```

```
##  [1] 999 999 999 999 999 999 999 999 999 999
```

```
# equivalent to SIR_sim_array[1,,][2,]
```

This gives a matrix of all the values corresponding to time = 0 (S is the second row, I is the third, R is the fourth); let's try using this in a loop. But first, let's make some storage for this. I am going to make one data frame, but you could make one for each compartment. For the measure of center, I am going to use the median, although the mean would be another choice (you just cannot calculate it using the quantile function).

```
SIR_results_summary <- data.frame(time=0:Time,
                                  S_lower=NA,
                                  S_median=NA,
                                  S_upper=NA,
                                  I_lower=NA,
                                  I_median=NA,
                                  I_upper=NA,
                                  R_lower=NA,
                                  R_median=NA,
                                  R_upper=NA)

for (i in 1:(Time+1)){
  ## Columns 2:4 (2,3,4) are the columns in our storage corresponding to S
  SIR_results_summary[i,2:4]  <- quantile(SIR_sim_array[i,2,], probs=c(0.025, 0.5, 0.975))
  ## Columns 5:7 (5,6,7) are the columns in our storage corresponding to I
  SIR_results_summary[i,5:7]  <- quantile(SIR_sim_array[i,3,], probs=c(0.025, 0.5, 0.975))
  ## Columns 8:10 (8,9,10) are the columns in our storage corresponding to R
  SIR_results_summary[i,8:10] <- quantile(SIR_sim_array[i,4,], probs=c(0.025, 0.5, 0.975))
}

head(SIR_results_summary)
```

```
##   time S_lower S_median S_upper I_lower I_median I_upper R_lower R_median
## 1    0 999.000    999.0 999.000   1.000      1.0   1.000   0.000      0.0
## 2    1 996.225    997.5 999.000   1.000      2.0   3.775   0.000      0.0
## 3    2 992.000    995.5 998.775   1.225      3.5   7.000   0.000      0.5
## 4    3 981.225    992.0 998.550   1.450      6.5  14.550   0.000      2.0
## 5    4 962.800    981.5 997.650   2.125     16.0  29.975   0.225      3.0
## 6    5 929.575    962.5 991.850   7.700     29.0  56.550   0.450      8.5
##   R_upper
## 1   0.000
## 2   1.000
## 3   1.775
## 4   5.000
## 5   9.550
## 6  18.100
```

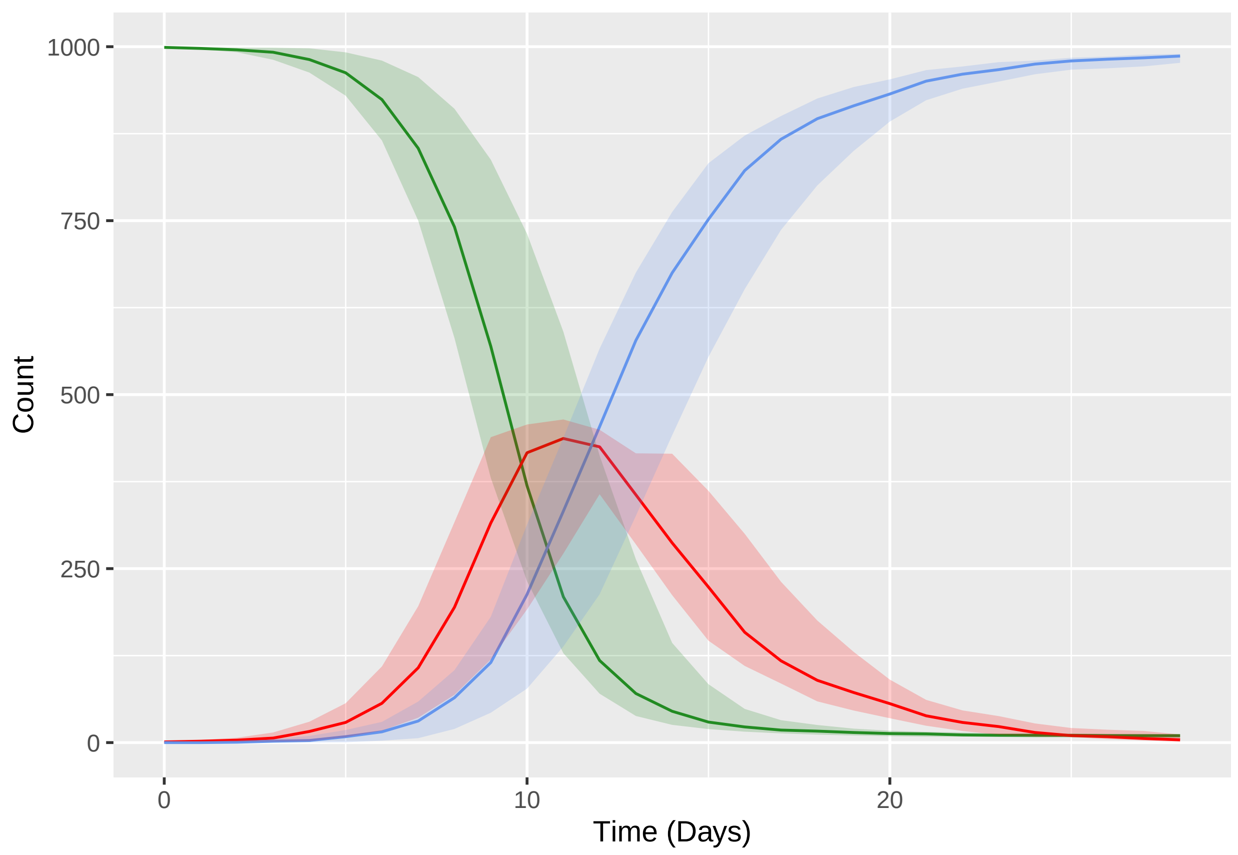
We will use this for plotting in (h) – see below.

We can do something similar for the lists – we just need to get the appropriate indexing. As with wide format/long format conversions, there are better ways to do this than by brute force coding. We will explore some of these methods for converting arrays or lists to data frames in the next homework assignment, so I am only going to use the solution with arrays for plotting below.

Plotting MC intervals

(h) Now we want to plot our results, including our MC intervals. We will need to add extra layers to our ggplot to accomplish this.

```
ggplot(data=SIR_results_summary, aes(x=time)) +
  ## add average compartments
  geom_line(aes(y=S_median), color="forestgreen") +
  geom_line(aes(y=I_median), color="red") +
  geom_line(aes(y=R_median), color="cornflowerblue") +
  ## add MC intervals to plot using geom_ribbon
  geom_ribbon(aes(ymin=S_lower, ymax=S_upper), alpha=0.2, fill="forestgreen") +
  geom_ribbon(aes(ymin=I_lower, ymax=I_upper), alpha=0.2, fill="red") +
  geom_ribbon(aes(ymin=R_lower, ymax=R_upper), alpha=0.2, fill="cornflowerblue") +
  ylab("Count") +
  xlab("Time (Days)") +
  ## add details to create legend
  scale_color_manual(values=c("forestgreen", "red", "cornflowerblue"),
                     # labels=c("S", "I", "R"),
                     name="Compartment")
```



Note, for a final plot, we will still need legends, but this gives us a visual of the MC intervals.