Review: ODEs in R

Like Monday — But Slower

Mathew Kiang

1/30/2018

Goals for today

1. Review model code from Monday (line-by-line)
2. Visualize an SIR
3. Modify an SIR (make SIS, add births/deaths)

Can you run this command with no errors?

```r
library(deSolve)
```

- If not, please put up a message on the discussion board or email me.

SIR Code Review

Let's go over each line

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

Recall, this was the code for SIR models.

(It is up on Canvas.)

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

You'll need to modify things that are highlighted

We will give you the rest (most of it).

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

Make a vector of `parameter = value` pairs named `parms`

Every parameter you pass will require a value

You'll need to add/remove from `parms` as your model dictates

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

Make a vector of `compartment = population` pairs named `inits`

Every compartment will need some initial value

Again, you'll need to add/remove from `inits` as your model changes

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

Make a vector of time-steps named `dt` (equivalently, `dt <- 0:300`).

`seq(start, end, step)`

Want smaller time-steps? `seq(0, 300, .001)`

However, more time-steps (and finer time-steps) requires longer computation time

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

Wrap up your model as a function.

`ode()` requires your function to take a vector of time steps (`t`), a vector of compartment values (`x`), and a named vector of parameters (`params`).

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

This `function` is named `SIR` and takes inputs `t`, `x`, `parms`

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

`t` is the vector of time-steps

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

`parms` is the vector of parameters

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

`x` is the current state of the model (required for `ode()` to work). It starts off as `inits`.

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

NOTE: Don't change the order. You can — but just don't. (Trust me.)

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

`c()` combines `parms` and `x` into one vector

`as.list()` converts that vector into a list

`with()` allows everything in the `{ }` to be referred to by shorthand

- Without `with()` you'd need to write `parms['k']`, `parms['beta']`, etc. every time

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

These lines define your compartments.

Know how to modify these.

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

Collect compartments into `der` and `return` them as a `list`.

`ode()` needs the function to return **something** as a `list`

We just make a variable called `der` for convenience. We could do `return(list(c(dS, dI, dR)))`

```r
library(deSolve)

parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                  func = SIR, parms = parms))
```

Run `ode()` on the model `SIR` with these `inits` and `parms`, for time `dt`

Then save that output as a `data.frame()` in a variable called `simulation`.

What is in simulations?

How many rows/columns?

What are they?

What is in simulations?

How many rows/columns?

What are they?

```
head(simulation, 10)
```

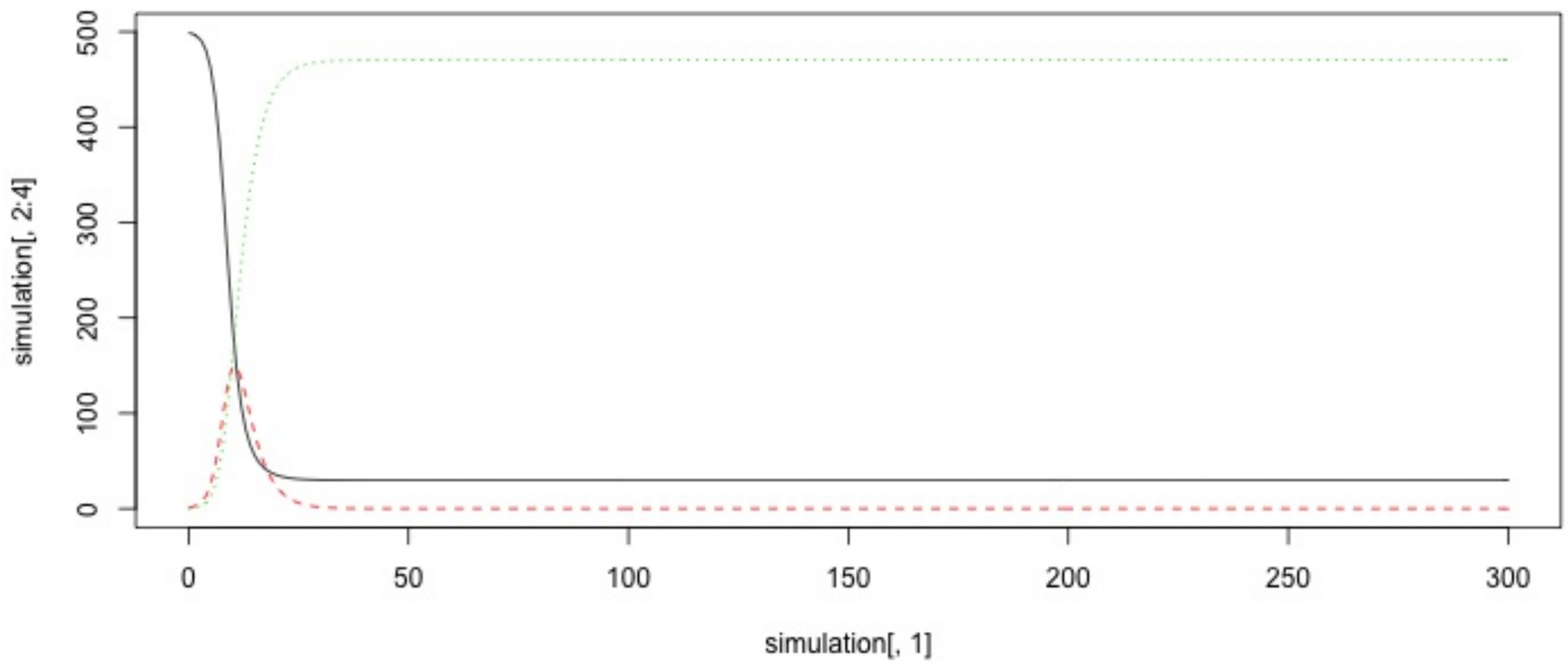
```
##    time        S          I           R
## 1     0 499.0000   1.000000   0.0000000
## 2     1 497.5874   1.940113   0.4724758
## 3     2 494.8637   3.749039   1.3873012
## 4     3 489.6624   7.189285   3.1483259
## 5     4 479.9111  13.588014   6.5008750
## 6     5 462.2423  25.004840  12.7528266
## 7     6 432.1079  43.903603  23.9884834
## 8     7 385.5990  71.432965  42.9680077
## 9     8 323.6259 104.204523  72.1695765
## 10    9 254.9172 133.138238 111.9445859
```

(Could also run `simulation[1:10, ]`)

If you still aren't clear on indexing matrices, vectors, or lists, you should definitely redo the Swirl tutorial from the beginning of the class. You'll need to do this a lot.

Visualize it

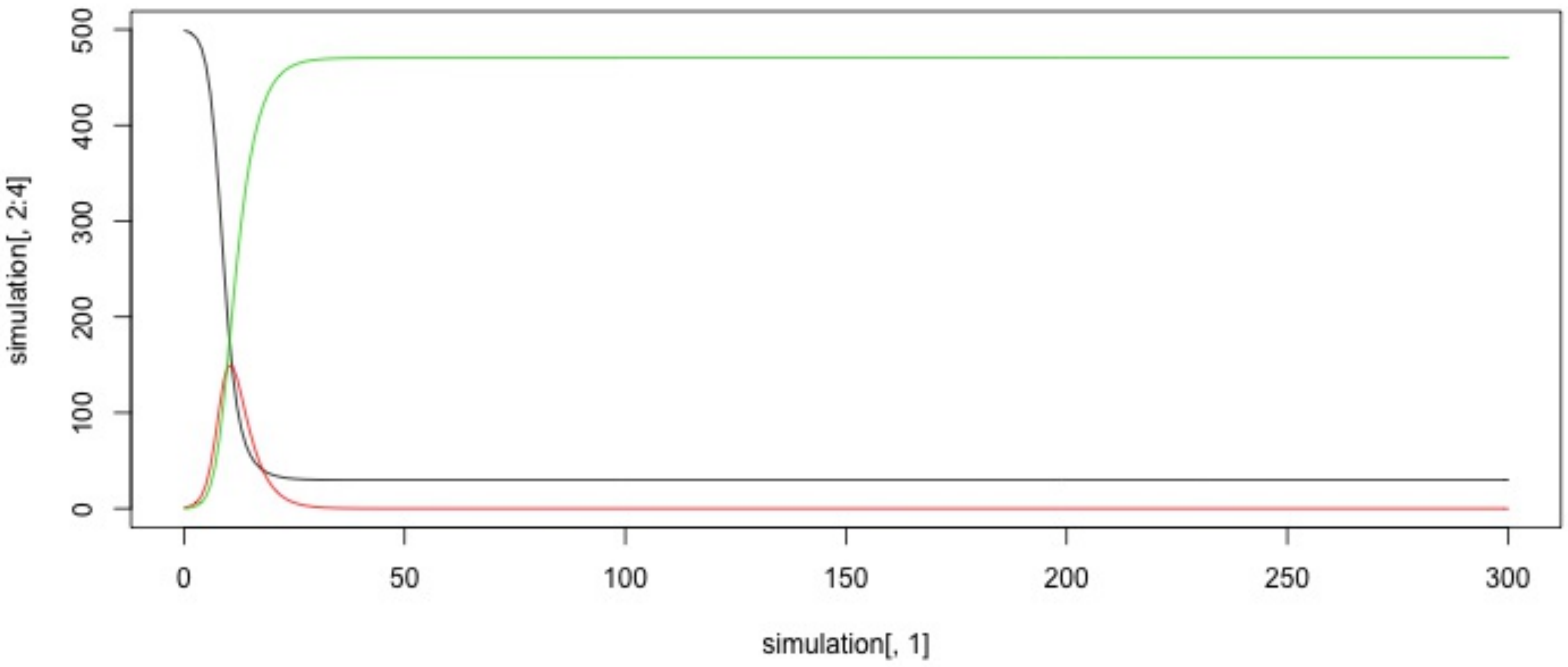
```
matplot(x = simulation[, 1], y = simulation[, 2:4], type = "l")
```



`matplot()` just plots one column (`x`) against other columns (`y`) of a matrix.

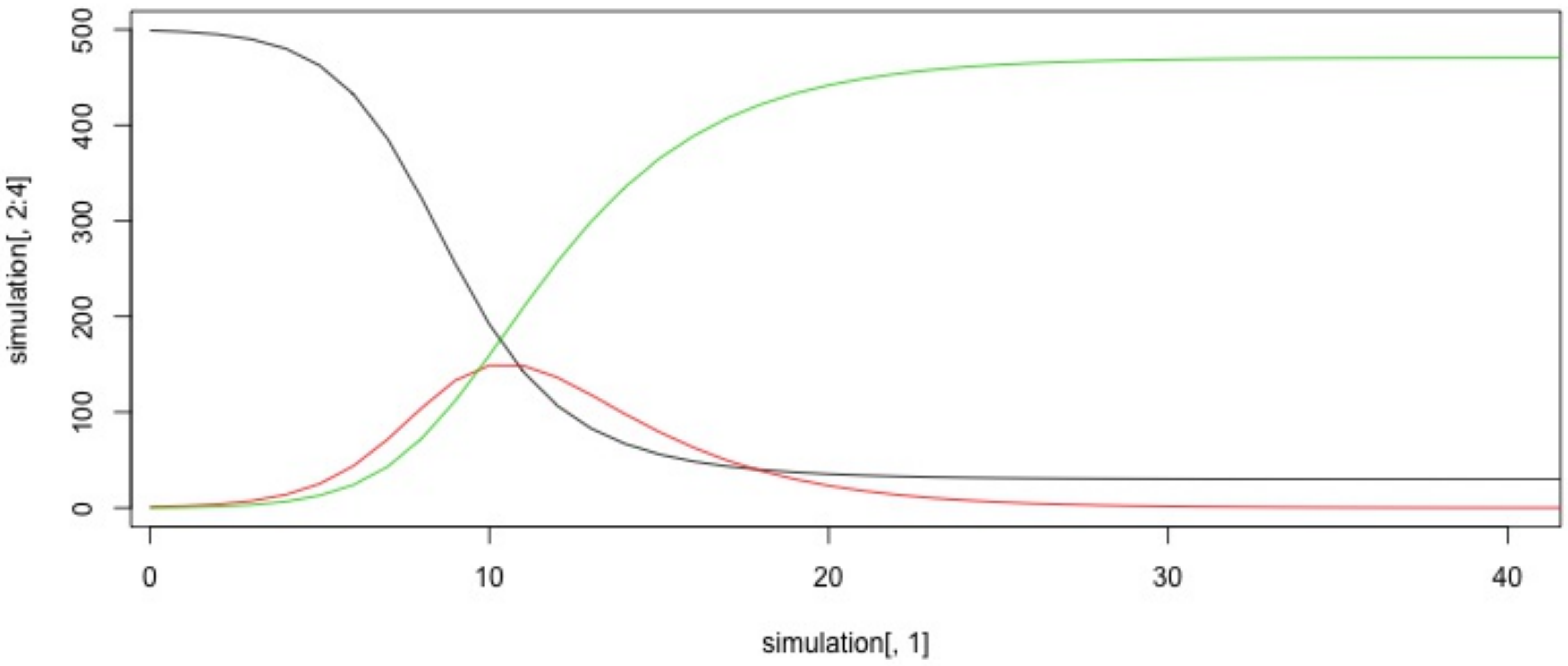
Visualize it (better)

```
matplot(x = simulation[, 1], y = simulation[, 2:4], type = "l",
        lty = 1)
```



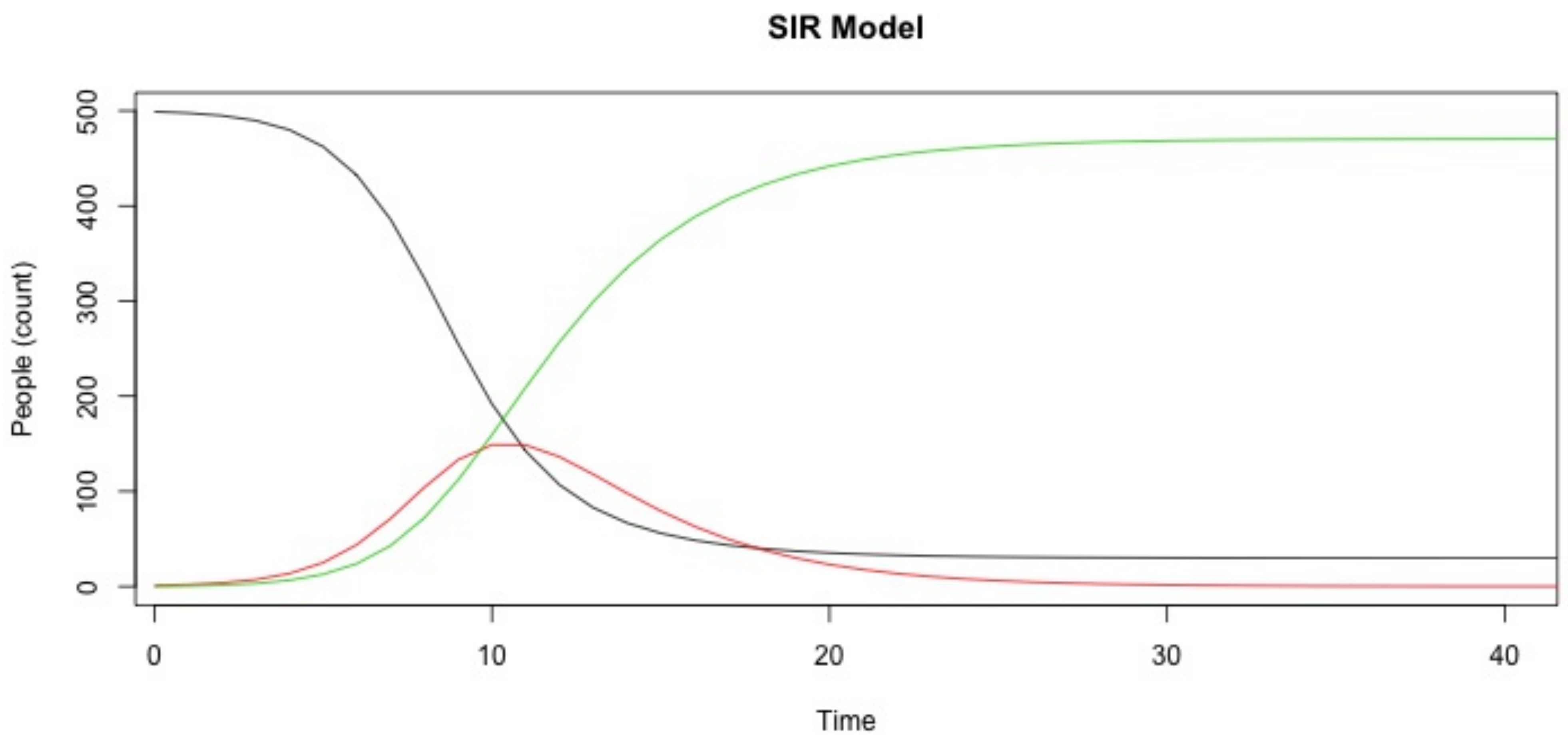
Visualize it (better-er)

```
matplot(x = simulation[, 1], y = simulation[, 2:4], type = "l",
        lty = 1, xlim = c(1, 40))
```



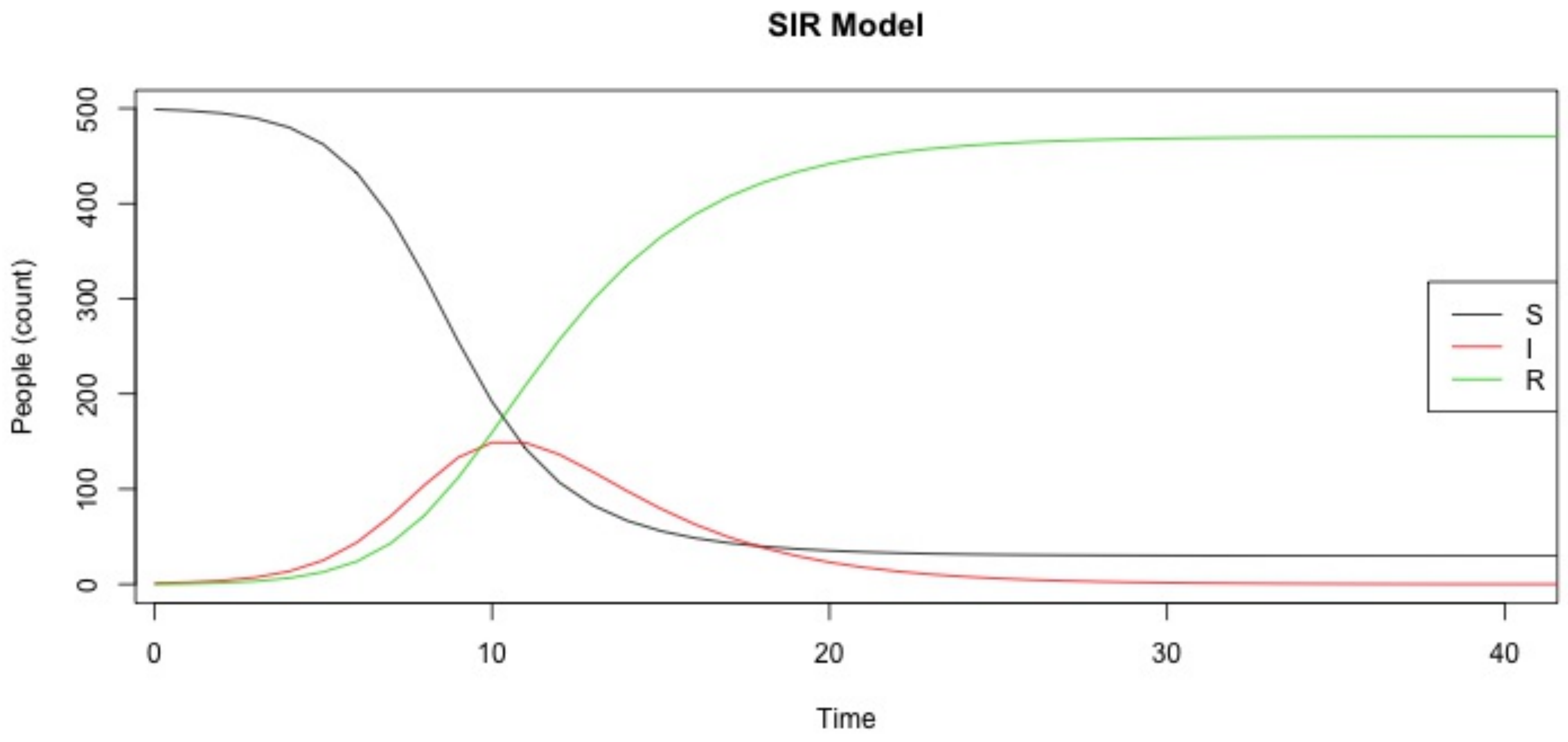
Visualize it (better-er-er)

```
matplot(x = simulation[, 1], y = simulation[, 2:4], type = "l",
        lty = 1, xlim = c(1, 40),
        xlab = "Time", ylab = "People (count)", main = "SIR Model")
```



Visualize it (best?)

```
matplot(x = simulation[, 1], y = simulation[, 2:4], type = "l",
        lty = 1, xlim = c(1, 40),
        xlab = "Time", ylab = "People (count)", main = "SIR Model")
legend(x = "right", legend = c('S', 'I', 'R'), col = 1:3, lty = 1)
```



Visualize it

- See `?matplot` and `?legend` for more.
- Also can use `plot()` and `lines()` together.

For really pretty graphs, see `ggplot2`.

- For example: https://mkiang.shinyapps.io/DiseaseDynamics/

Challenge Round

Let's modify our models

Make an SI model

```
parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

What needs to be modified?

Make an SI model

```r
parms <- c(beta = 0.333, k = 3 , r = 0.333)
inits <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

What needs to be modified?

```r
parms_si <- c(beta = 0.333, k = 3)
inits_si <- c(S = 499, I = 1)
dt <- seq(0, 300, 1)

SIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        dS <- - (beta * k * S * I) / (S + I + R)
        dI <- + (beta * k * S * I) / (S + I + R) - r * I
        dR <- r * I

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation <- as.data.frame(ode(y = inits, times = dt,
                                func = SIR, parms = parms))
```

Remove `R` from `parms` and `inits`.

Also renamed them so we don't overwrite old `parms` and `inits`.

```r
parms_si <- c(beta = 0.333, k = 3)
inits_si <- c(S = 499, I = 1)
dt <- seq(0, 300, 1)

SI <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        N <- S + I
        dS <- - (beta * k * S * I) / N
        dI <- + (beta * k * S * I) / N


        der <- c(dS, dI)

        return(list(der))
     })
}

 simulation <- as.data.frame(ode(y = inits, times = dt,
                                 func = SIR, parms = parms))
```

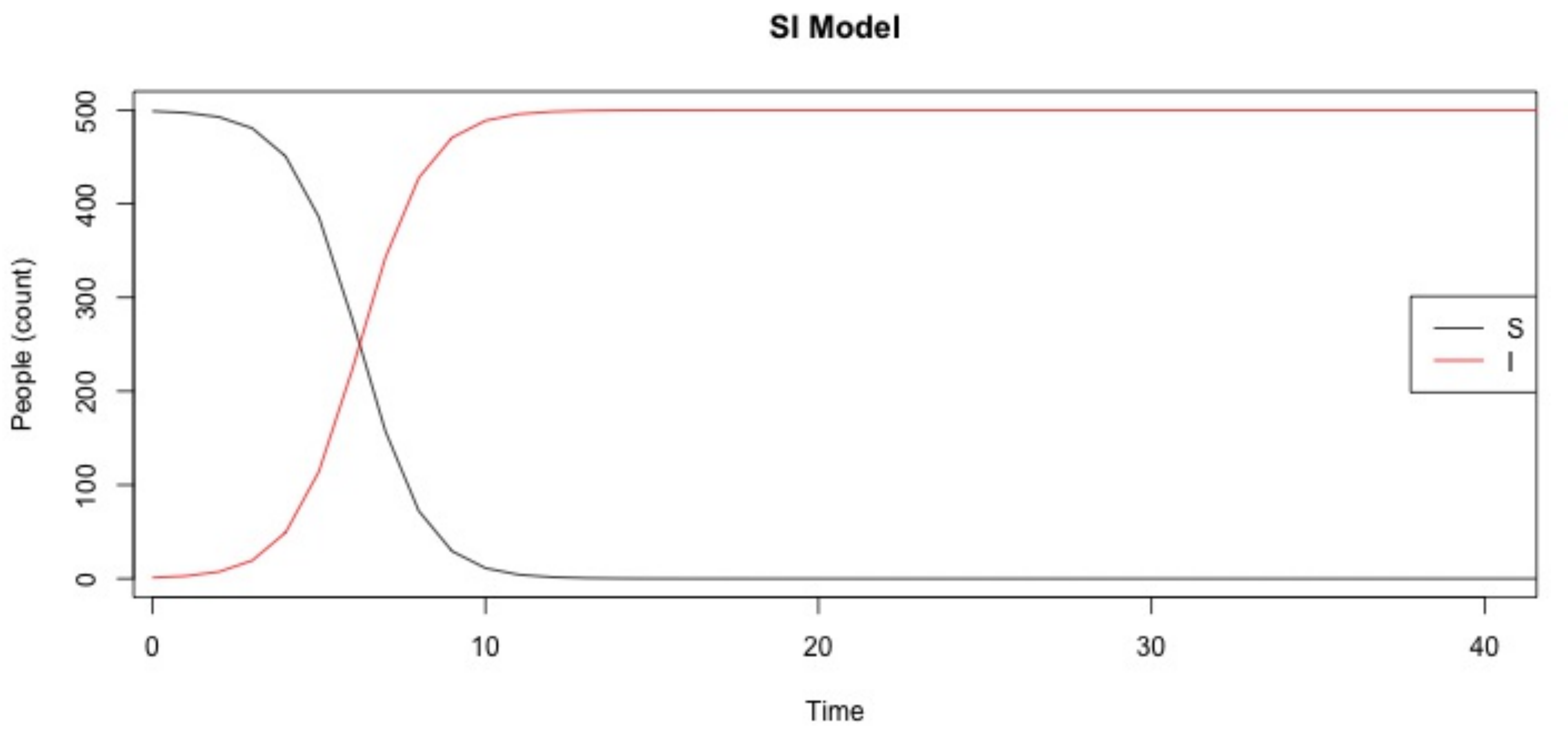
Remove `dR` from from our model.

Also define `N` in one place so we don't have to modify it multiple times.

```r
parms_si <- c(beta = 0.333, k = 3)
inits_si <- c(S = 499, I = 1)
dt <- seq(0, 300, 1)

SI <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        N <- S + I
        dS <- - (beta * k * S * I) / N
        dI <- + (beta * k * S * I) / N

        der <- c(dS, dI)

        return(list(der))
     })
}

simulation_si <- as.data.frame(ode(y = inits_si, times = dt,
                                   func = SI, parms = parms_si))
```

Save simulations in a new object and change `ode()` call to our new `parms`, `inits`, and `func`.

```r
matplot(x = simulation_si[, 1], y = simulation_si[, 2:3], type = "l",
        lty = 1, xlim = c(1, 40),
        xlab = "Time", ylab = "People (count)",
        main = "SI Model")
legend(x = "right", legend = c('S', 'I'),
       col = 1:2, lty = 1)
```



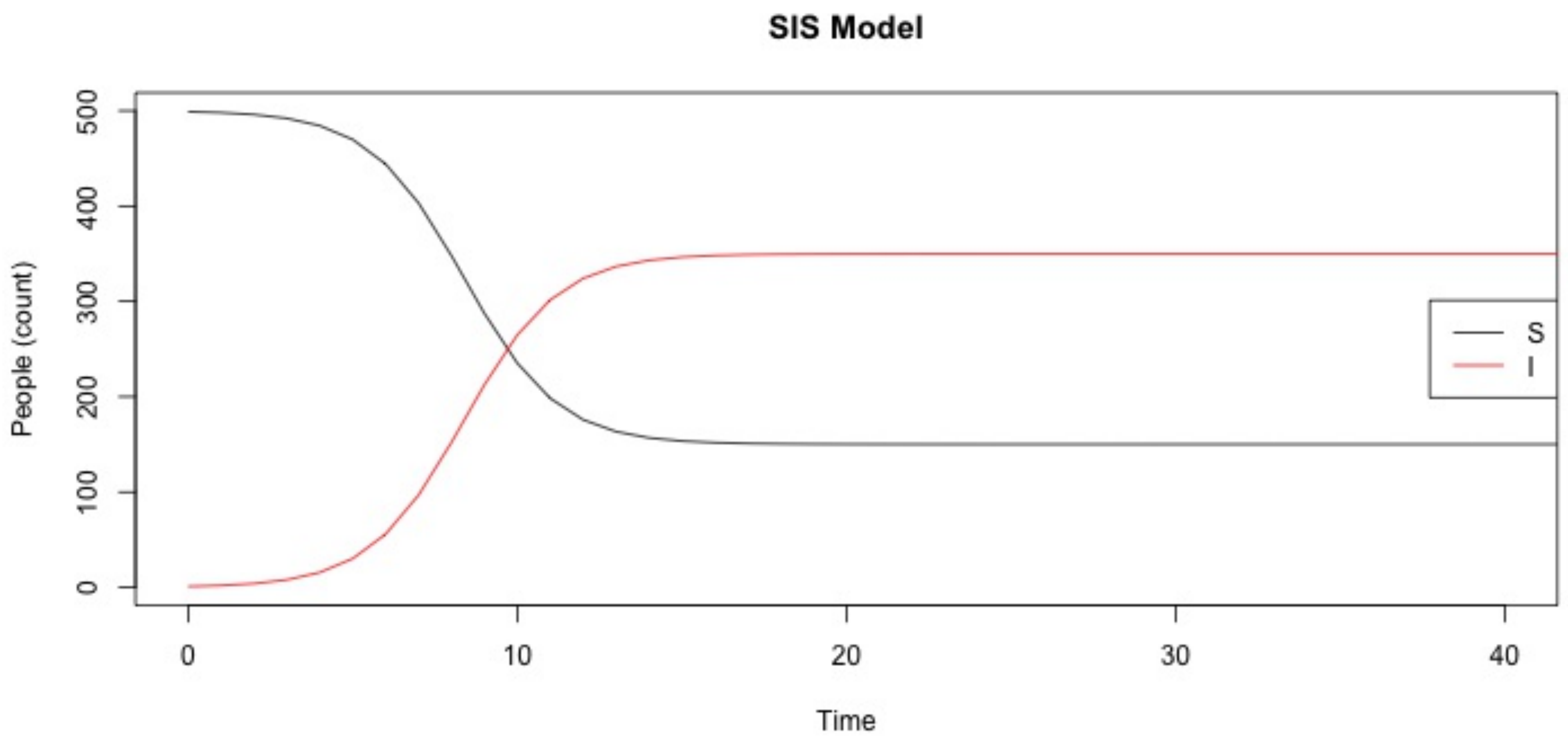
SIS

With a neighbor, make a model that allows for returning from I to S.

Example of SIS

```r
parms_sis <- c(beta = 0.333, k = 3, alpha = .3)
inits_sis <- c(S = 499, I = 1)
dt <- seq(0, 300, 1)

SIS <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        N <- S + I
        dS <- - (beta * k * S * I) / N + (alpha * I)
        dI <- + (beta * k * S * I) / N - (alpha * I)

        der <- c(dS, dI)

        return(list(der))
    })
}

simulation_sis <- as.data.frame(ode(y = inits_sis, times = dt,
                                    func = SIS, parms = parms_sis))
```

```r
matplot(x = simulation_sis[, 1], y = simulation_sis[, 2:3], type = "l",
        lty = 1, xlim = c(0, 40),
        xlab = "Time", ylab = "People (count)",
        main = "SIS Model")
legend(x = "right", legend = c('S', 'I'),
       col = 1:2, lty = 1)
```



SIR with births/deaths

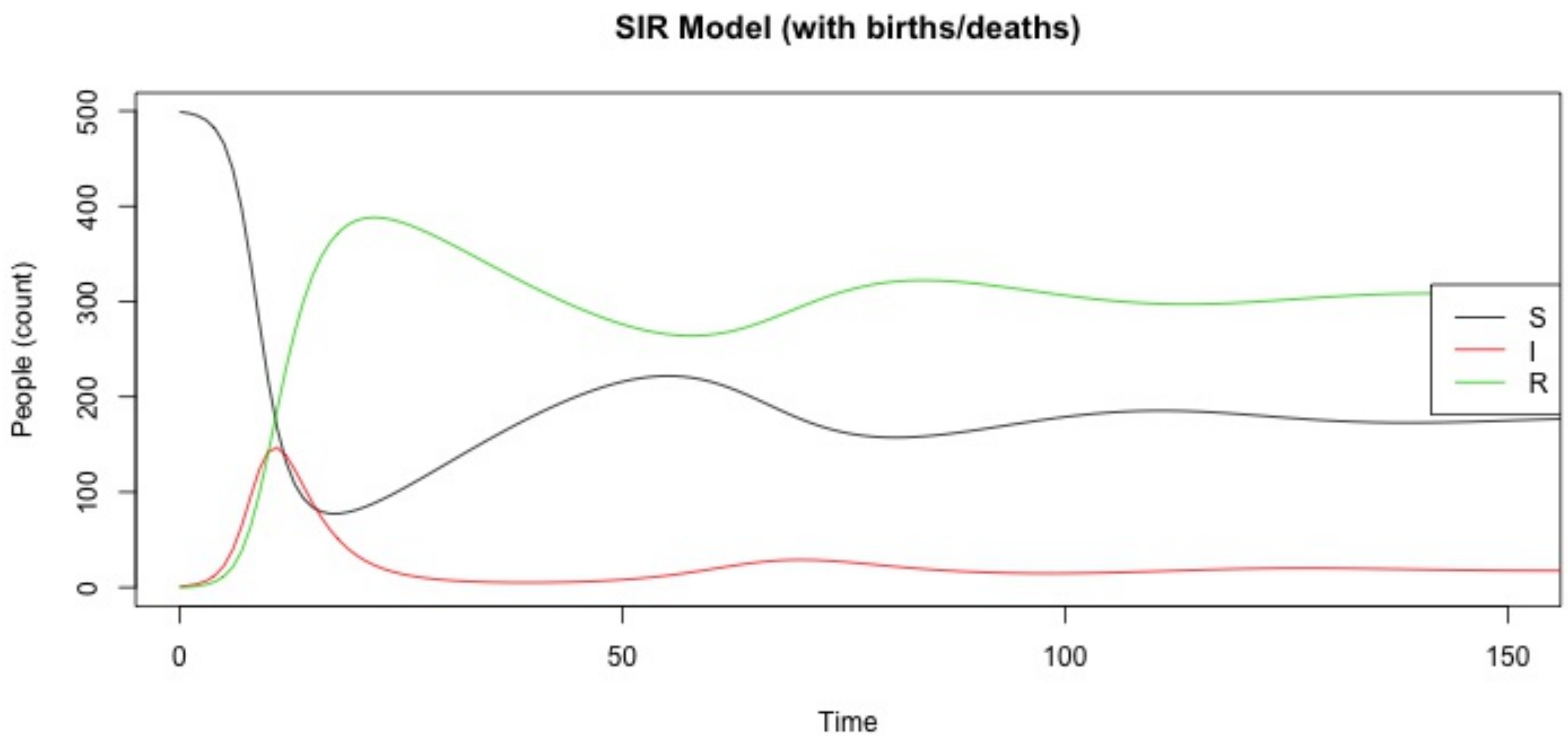
With a neighbor, make an SIR model where people can be born S and everybody dies

Keep birth rate = death rate

Example of SIR with births/deaths

```
parms_bd <- c(beta = 0.333, k = 3 , r = 0.333, birth = .02, death = .02)
inits_bd <- c(S = 499, I = 1, R = 0)
dt <- seq(0, 300, 1)

SIR_bd <- function(t, x, parms){
    with(as.list(c(parms, x)), {

        N <- S + I + R
        dS <- - (beta * k * S * I) / N - (death * S) + (birth * N)
        dI <- + (beta * k * S * I) / N - r * I - (death * I)
        dR <- r * I - (death * R)

        der <- c(dS, dI, dR)

        return(list(der))
    })
}

simulation_bd <- as.data.frame(ode(y = inits_bd, times = dt,
                                func = SIR_bd, parms = parms_bd))
```

```r
matplot(x = simulation_bd[, 1], y = simulation_bd[, 2:4], type = "l",
        lty = 1, xlim = c(1, 150),
        xlab = "Time", ylab = "People (count)",
        main = "SIR Model (with births/deaths)")
legend(x = "right", legend = c('S', 'I', 'R'),
       col = 1:3, lty = 1)
```



Conclusion

- As you can see, ODE models can get complex, very quickly
    - We could add births, return rates, seasonality, age structure, vaccination, vectors, changing human behavior, etc.
- Models will get harder than this, but you're beyond the steep learning curve at this point
- You'll need to know how to modify and build on them, but not necessarily the details of `R`
    - Point of using `R` is just to allow you to quickly see what happens to dynamics when you modify a model

That's it.

Thanks