# Ebola Worksheet

## From Wednesday lecture – But Slower

Mathew Kiang

2/2/2018

# Goals for today

## Go over the Ebola handout*

*Again, only providing you with enough code to finish it on your own.

# Question 1*

## Make an SEIR model that incorporates case fatality ratio f

*Sort of -- Questions are unnumbered on the worksheet.

# Start with code you already have

```
SEIR <- function(t, x, parms){
    with(as.list(c(parms, x)), {
        N <- S + E + I + R
        dS <- - (beta * k * S * I) / N
        dE <- + (beta * k * S * I) / N - (a * E)
        dI <- + (a * E) - (r * I)
        dR <- r * I
        der <- c(dS, dE, dI, dR)
        return(list(der))
    })
}
```

Here is your boilerplate `SEIR` code. Incoporate $f$, which is a case fatality ratio. Recall, this is the fraction of infectious who do not recover.

- Also, change `a` to `s` ($\sigma$) and `r` to `g` ($\gamma$) to be consistent with the Althaus
- We are not going to use `b * k`, so replace that with $\beta$ as `B`

# Work with a neighbor to make this model

## Remember:

rename a to s

rename r to g

use B instead of b * k

```r
library(deSolve)

dt <- seq(0, 365, 1)
inits <- c(S = 999999,E = 0, I = 1, R = 0)
parms <- c(B = 0.45, g = 1/5.61, s = 1/5.3, f = 0.6)

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

        N <- S + E + I + R
        dS <- - (B * S * I) / N
        dE <- + (B * S * I) / N - (s * E)
        dI <- (s * E) - (g * I)
        dR <- (1 - f) * (g * I)

        der <- c(dS, dE, dI, dR)

        return(list(der))
    })
}

data_out <- as.data.frame(ode(inits, dt, SEIR_ex, parms = parms))
```

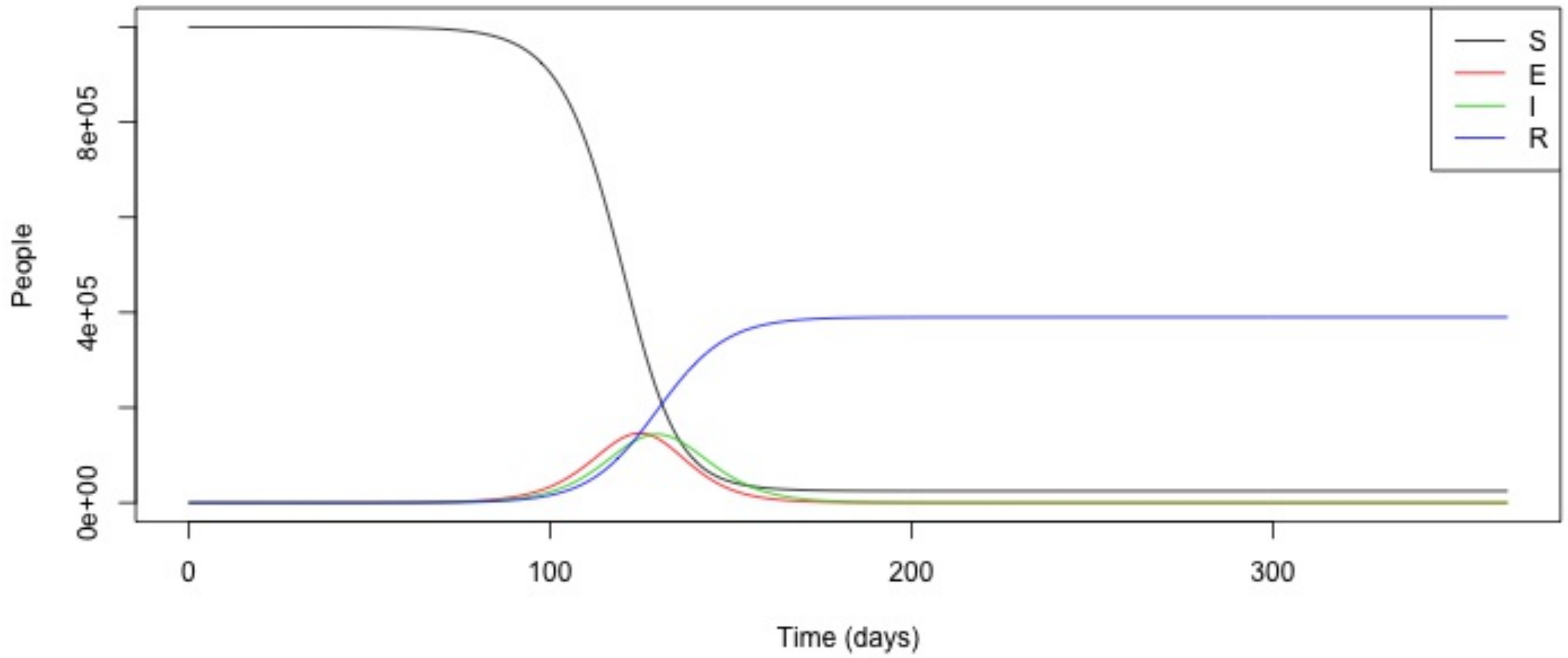
Your code should now look something like this.

- This is almost **exactly** like our boilerplate code.
- Use the `inits`, `dt`, and `parms` I specified

Should all be very familiar by now. Review previous slides if this is still unclear.

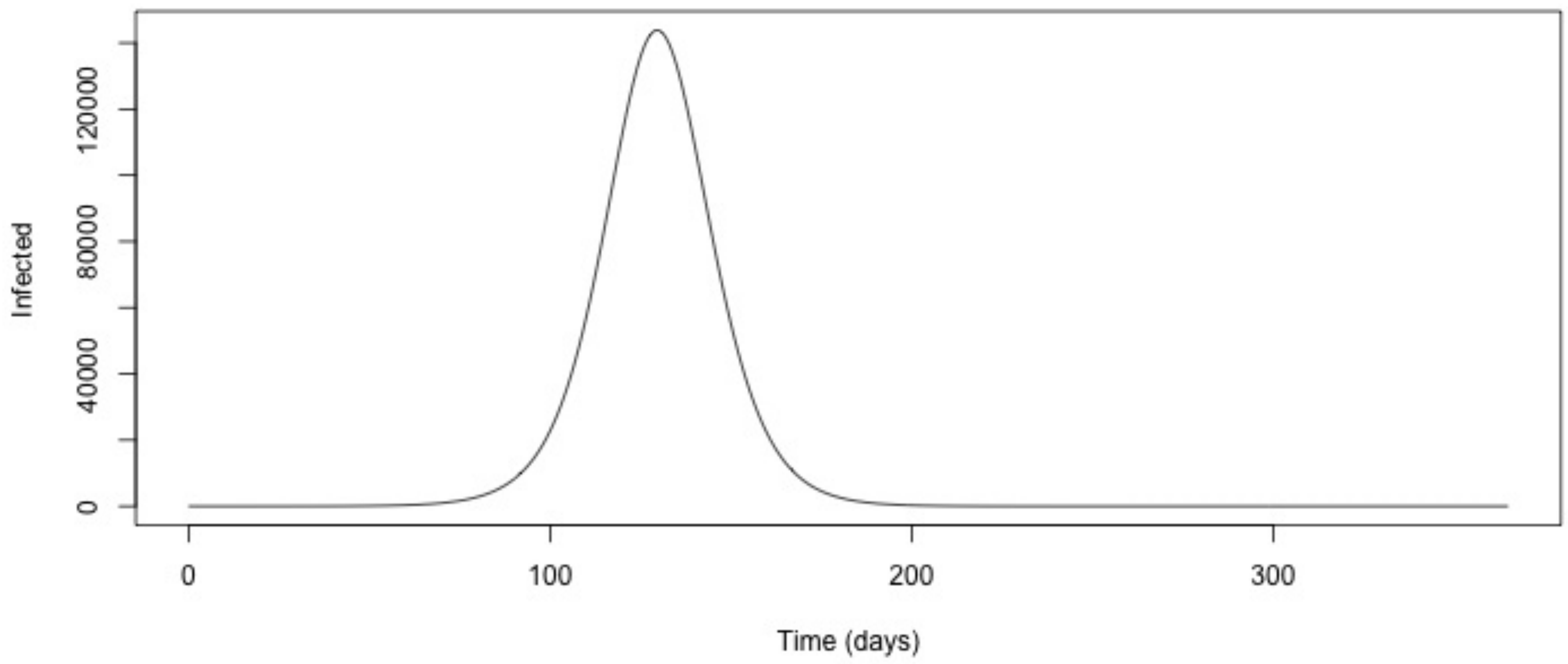
# Plot of all lines

```r
matplot(data_out[, 1], data_out[, 2:5], type = 'l',
        ylab = 'People', xlab = 'Time (days)',lty = 1)
legend(x = "topright", legend = c('S', 'E', 'I', 'R'), col = 1:4, lty = 1)
```



# Plot of infected

```r
matplot(data_out[, 1], data_out[, 4], type = 'l',
        ylab = 'Infected', xlab = 'Time (days)')
```



# Question 2

With a neighbor, add compartments C for total cases and D for total deaths

# Add new compartments

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

        N <- S + E + I + R
        dS <- - (B * S * I) / N
        dE <- + (B * S * I) / N - (s * E)
        dI <- (s * E) - (g * I)
        dR <- (1 - f) * (g * I)

        der <- c(dS, dE, dI, dR)

        return(list(der))
    })
}
```

Again, start with code you already have. Add:

- `dC` which is the cumulative cases
- `dD` which is the total number of deaths

# Add new compartments

```r
SEIR_alt1 <- function(t, x, parms) {
    with(as.list(c(parms, x)), {

        N <- S + E + I + R
        dS <- - (B * S * I) / N
        dE <- + (B * S * I) / N - (s * E)
        dI <- (s * E) - (g * I)
        dR <- (1 - f) * (g * I)

        dC <- s * E
        dD <- f * g * I

        der <- c(dS, dE, dI, dR, dC, dD)

        return(list(der))
    })
}
```

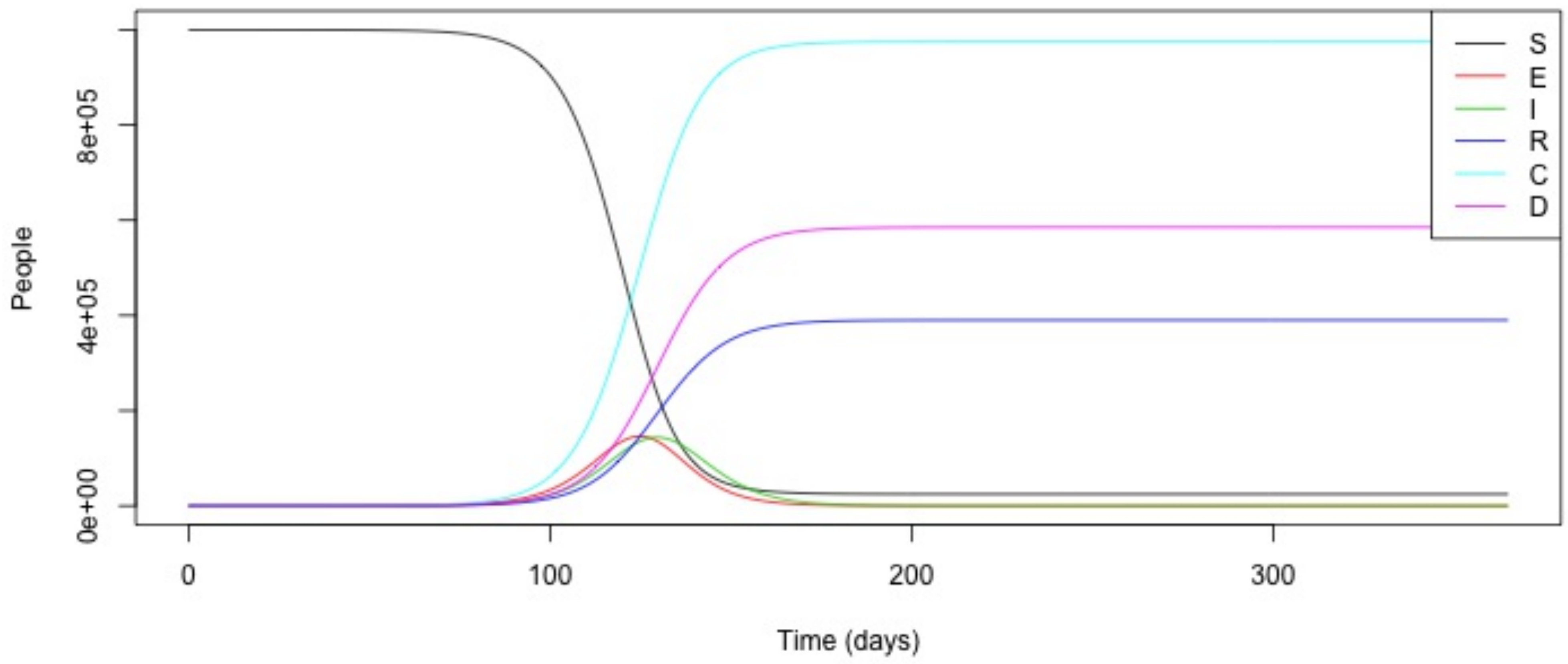
Don't forget to return `dC` and `dD` and add them in `inits`.

# Full Solution

```r
inits_alt1 <- c(S = 999999,E = 0, I = 1, R = 0, C = 0, D = 0)

SEIR_alt1 <- function(t, x, parms) {
    with(as.list(c(parms, x)), {

        N <- S + E + I + R
        dS <- - (B * S * I) / N
        dE <- + (B * S * I) / N - (s * E)
        dI <- (s * E) - (g * I)
        dR <- (1 - f) * (g * I)

        dC <- s * E
        dD <- f * g * I

        der <- c(dS, dE, dI, dR, dC, dD)

        return(list(der))
    })
}

data_alt1 <- as.data.frame(ode(inits_alt1, dt, SEIR_alt1, parms = parms))
```

# Plot of all lines

```
matplot(data_alt1[, 1], data_alt1[, 2:7], type = 'l',
        ylab = 'People', xlab = 'Time (days)',lty = 1)
legend(x = "topright", legend = c('S', 'E', 'I', 'R', 'C', 'D'),
       col = 1:6, lty = 1)
```



# Question 3

## Time-varying transmission probability

# Time-varying transmission

Althaus parameterizes transmission probability as:

$$\beta(t) = \beta e^{-k(t-\tau)}$$

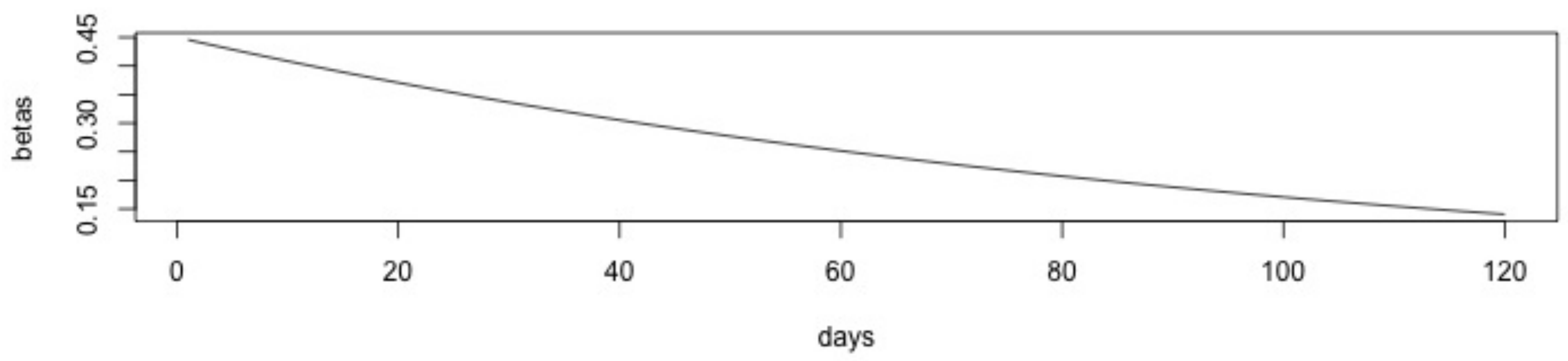
- Assume:
  - $k = 0.0097$
  - $\beta = 0.45$
  - $\tau = 0$ (immediate control measures)

With a neighbor, plot $\beta$ as a function of time from $t = 0$ to $t = 120$

- Hints:
  - Make a sequence
  - Vectorized formulas are your friend

# Solution

```r
## Set constants
beta0 <- 0.45
k <- 0.0097
tau <- 0

## Plug into formula
days <- 1:120
betas <- beta0 * exp(-k * (days - tau))

## Plot it
plot(x = days, y = betas, type = "l")
```



# Solution

```
## Set constants
beta0 <- 0.45
k <- 0.0097
tau <- 0

## Plug into formula
days <- 1:120
betas <- beta0 * exp(-k * (days - tau))

## Plot it
plot(x = days, y = betas, type = "l")
```

Set some constants. Not necessary, but makes the formula clearer.

# Solution

```r
## Set constants
beta0 <- 0.45
k <- 0.0097
tau <- 0

## Plug into formula
days <- 1:120
betas <- beta0 * exp(-k * (days - tau))

## Plot it
plot(x = days, y = betas, type = "l")
```

Set some constants. Not necessary, but makes the formula clearer.

Make a sequence of days (or `seq(0, 120, 1/24)` for calculate hourly $\beta$)

# Solution

```r
## Set constants
beta0 <- 0.45
k <- 0.0097
tau <- 0

## Plug into formula
days <- 1:120
betas <- beta0 * exp(-k * (days - tau))

## Plot it
plot(x = days, y = betas, type = "l")
```

Set some constants. Not necessary, but makes the formula clearer.

Make a sequence of days (or `seq(0, 120, 1/24)` for calculate hourly $\beta$)

Make a new vector with the formula we want. Even though `k`, `tau`, and `beta0` are scalars, `R` will automatically vectorize (perform element-wise calculations) on `days` since it has length > 1. (Try `print(betas)` if this is unclear.)

# Solution

```r
## Set constants
beta0 <- 0.45
k <- 0.0097
tau <- 0

## Plug into formula
days <- 1:120
betas <- beta0 * exp(-k * (days - tau))

## Plot it
plot(x = days, y = betas, type = "l")
```

Set some constants. Not necessary, but makes the formula clearer.

Make a sequence of days (or `seq(0, 120, 1/24)` for calculate hourly $\beta$)

Make a new vector with the formula we want. Even though `k`, `tau`, and `beta0` are scalars, `R` will automatically vectorize (perform element-wise calculations) on `days` since it has length > 1. (Try `print(betas)` if this is unclear.)

Plot it.

# Question 4

Now calculate and plot the changing R0

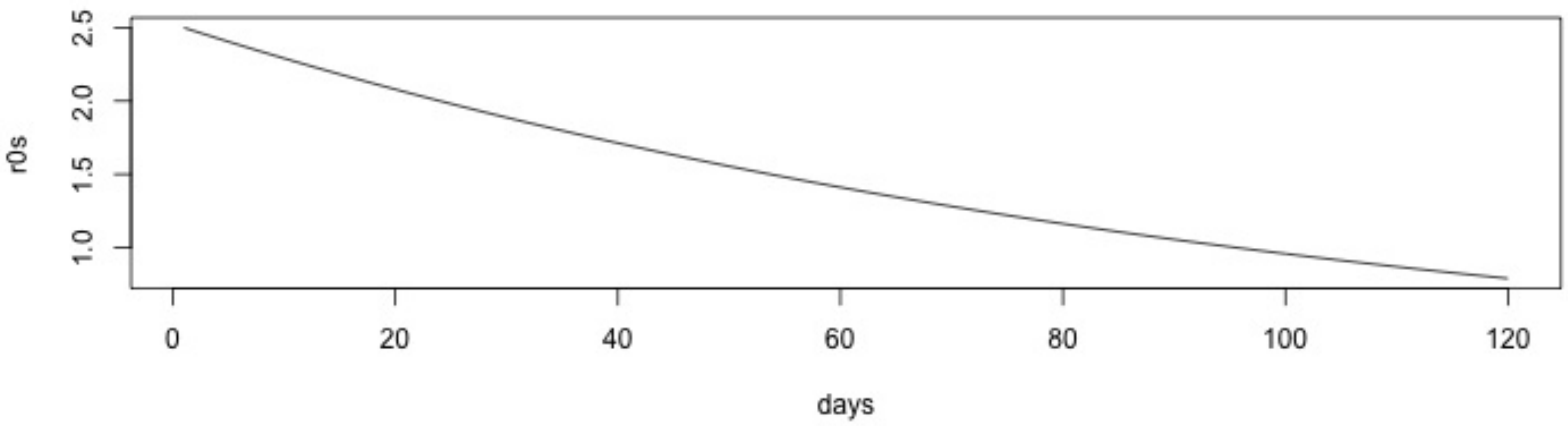
# Solution

Hint: This is (literally) one line of code to calculate and one line of code to plot.

# Solution

## Hint: This is (literally) one line of code to calculate and one line of code to plot.

```
r0s <- betas / (1/5.61)
plot(x = days, y = r0s, type = "l")
```



# On what day is R0 < 1?

# On what day is R0 < 1?

- Try `help(which)`

# On what day is R0 < 1?

- Try `help(which)`
- Combine that with indexing

# On what day is R0 < 1?

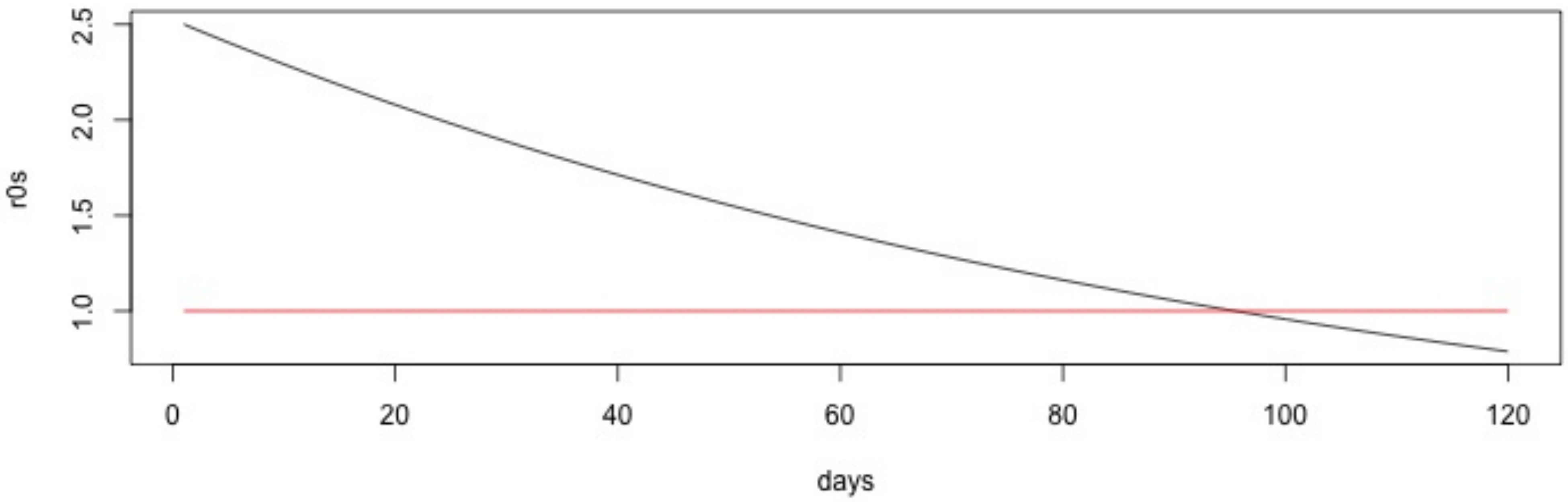
- Try `help(which)`
- Combine that with indexing

```
which(r0s <= 1)[1]
```

```
## [1] 96
```

# On what day is R0 < 1?

```
plot(x = days, y = r0s, type = "l")
lines(x = days, y = rep(1, length(days)), col = 'red')
```




Or we could do it visually.

(If this isn't clear, see `help(rep)` and consider why it is necessary.)

# With a neighbor, add the time-varying beta to SEIR model

Assume tau=0 for simplicity

# Solution

```
SEIR_alt2 <- function(t, x, parms) {
    with(as.list(c(parms, x)), {

        B  <- B_init * exp(-k * t)
        N  <- S + E + I + R
        dS <- -(B * S * I) / N
        dE <- +(B * S * I) / N - (s * E)
        dI <- (s * E) - (g * I)
        dR <- (1 - f) * (g * I)

        dC <- s * E
        dD <- f * g * I

        der <- c(dS, dE, dI, dR, dC, dD)

        return(list(der))
    })
}
```

Yes, that's it.*

* NOTE: This only works when tau=0. Need `ifelse()` if we incorporate tau.

# Examine one of the countries

(Do this on your own or with a neighbor)

# Where is the data?

Althaus's GitHub: https://github.com/calthaus/Ebola*

* NOTE: See the `Intro to R` tutorial if you don't know how to import `csv` files.

# That's it.

Thanks