SIR Model with Demographic Stochasticity

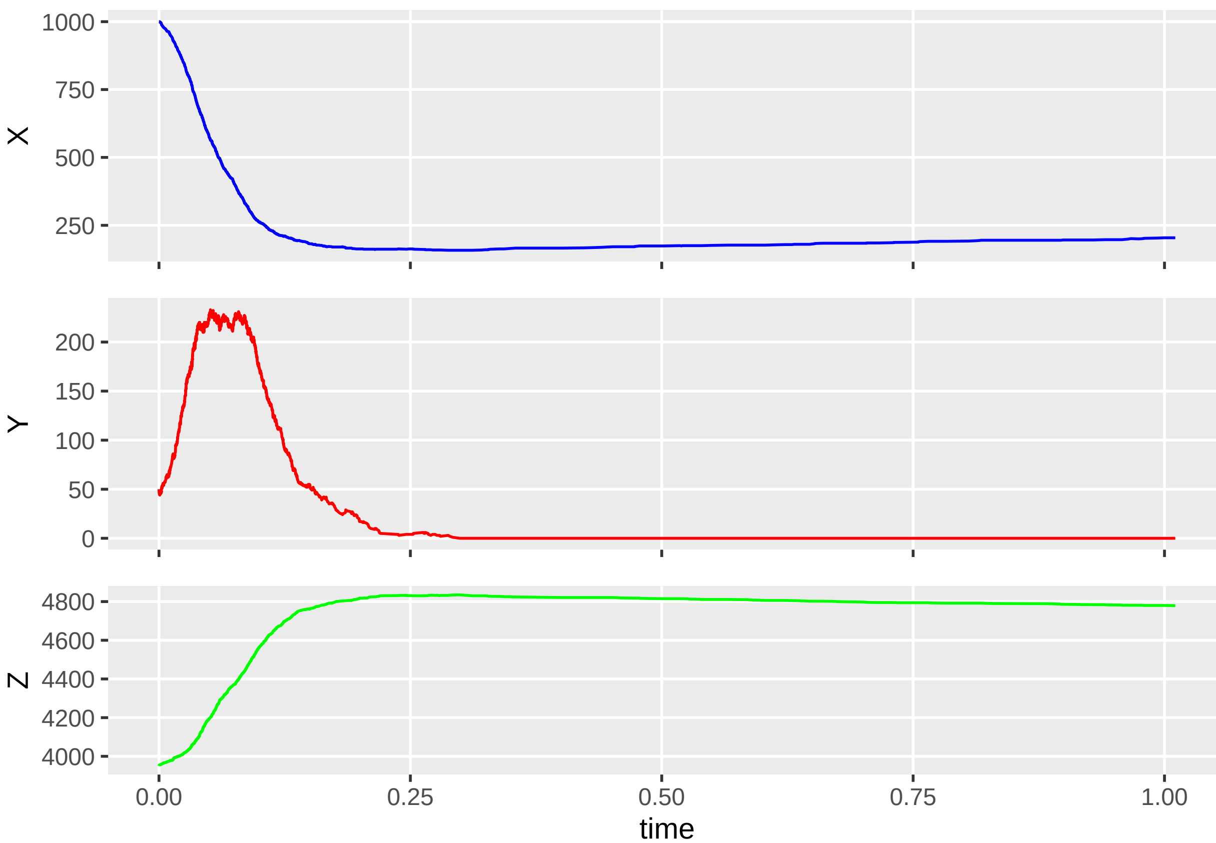
K. Bodie Weedop

10/23/2019

Using Gillespie's Direct Algorithm

```
# Parameter values
beta <- 1.0
gamma <- 0.1
mu <- 1/(70*365)
params <- c(beta, gamma, mu)

# Initial conditions
N0 <- 5000
X0 <- 1000
Y0 <- 0.01 * N0
Z0 <- N0 - X0 - Y0
vars <- c(X0, Y0, Z0)

t0 <- 0
years <- 1
tn <- years * 365

direct.results <- data.frame(time = t0,
                             X = X0,
                             Y = Y0,
                             Z = Z0)

direct.sir <- function(vars, params) {
  beta <- params[1]
  gamma <- params[2]
  mu <- params[3]

  x0 <- vars[1]
  y0 <- vars[2]
  z0 <- vars[3]
  n0 <- sum(vars)

  R.n <- rep(0, 6)
  change <- matrix(nrow = 6, ncol = 3)

  #Events
  # Transmission
  R.n[1] <- beta * x0 * y0 / n0
  change[1,] <- c(-1, 1, 0)
  # Recovery
  R.n[2] <- gamma * y0
  change[2,] <- c(0, -1, 1)
  # Birth
  R.n[3] <- mu * n0
```

```
  change[3,] <- c(1, 0, 0)
  # Death in each class
  R.n[4] <- mu * x0
  change[4,] <- c(-1, 0, 0)
  R.n[5] <- mu * y0
  change[5,] <- c(0, -1, 0)
  R.n[6] <- mu * z0
  change[6,] <- c(0, 0, -1)

  rand1 <- runif(n=1)
  rand2 <- runif(n=1)

  d.t <- (-1/sum(R.n))*log(rand1)
  p <- min(which(cumsum(R.n) >= rand2 * sum(R.n)))
  vars <- vars + change[p,]
  return(c(d.t, vars))
}

while (t0 <= tn) {
  tmp <- direct.sir(vars = vars, params = params)
  t0 <- t0 + tmp[1]
  tmp[1] <- t0
  direct.results <- rbind(direct.results, tmp)
  vars <- tmp[2:4]
}
```



Using the “τ-leap” Method

```r
# Parameter values
b <- 1.0
g <- 0.1
m <- 1/(70*365)
params <- c(b, g, m)

# Initial values for variables; endemic equilibrium
N0 <- 5000
X0 <- 1000
Y0 <- 0.01 * N0
Z0 <- N0 - X0 - Y0
init.vars <- c(X0, Y0, Z0)

# Time to run model simulation
years <- 1
tau <- 1
t_range <- seq(0, years*365, tau)

# SIR model using tau-leap method for stochastics
tau.sir <- function(vars, params) {
  beta <- params[1]
  gamma <- params[2]
  mu <- params[3]

  x0 <- vars[1]
  y0 <- vars[2]
  z0 <- vars[3]
  n0 <- sum(vars)

  # Initializing data frame for event rates and the direction in which the variables change
  rate <- rep(0, 6)
  change <- matrix(nrow = 6, ncol = 3)

  # Transmission Event
  rate[1] <- (beta * x0 * y0) / n0
  change[1,] <- c(-1, 1, 0)
  # Recovery Event
  rate[2] <- gamma * y0
  change[2,] <- c(0, -1, 1)
  # Birth Event
  rate[3] <- mu * n0
  change[3,] <- c(1, 0, 0)
  # Death Events
  rate[4] <- mu * x0
  change[4,] <- c(-1, 0, 0)
  rate[5] <- mu * y0
  change[5,] <- c(0, -1, 0)
  rate[6] <- mu * z0
  change[6,] <- c(0, 0, -1)

  # Update variables
```

```
  for (i in 1:length(rate)) {
    # Number of times events occur assumed to be Poisson
    tmp <- rpois(n=1, lambda=rate[i]*tau)
    # Ensuring that values do not go negative -- use whichever value is the minimum
    non.neg <- min(c(tmp, vars[which(change[i,] < 0)]))
    # Update using change values (e.g. [+1, -1, 0]) multiplied by the rate of that event
    vars = vars + change[i,] * non.neg
  }
  return(vars)
}

simulation.results <- data.frame(time=t_range, X=NA, Y=NA, Z=NA)
simulation.results[1, 2:ncol(simulation.results)] <- init.vars

for (t in 1:(length(t_range)-1)) {
  t.vars <- as.numeric(simulation.results[t, 2:ncol(simulation.results)])
  t <- t + 1
  tmp <- tau.sir(t.vars, params)
  simulation.results[t, 2:ncol(simulation.results)] <- tmp
}
```