BasicNet COVID19 Refugee Camp

Luis Chaves

18/04/2020

Setting up empty network

Network basic features

First, we define our population, along with their attributes.

```
n = 1000 # individuals in camp

# initialize empty net
nw = network.initialize(n = n, directed = FALSE)

# set age attribute according to real age breakdown
#' Age breakdown within the camp (20,000 people)
#   50+ 4.7%      class 3
#   19-49 54.8%   class 2
#   0-18 40.5%    class 1
num.age_cl1 = round((40.5/100)*n)
num.age_cl2 = round((54.8/100)*n)
num.age_cl3 = round((4.7/100)*n)

stopifnot((num.age_cl1+num.age_cl2+num.age_cl3) == n)

nw = set.vertex.attribute(nw, "age_cl",
                          c(rep("young", num.age_cl1),
                            rep("adults", num.age_cl2),
                            rep("old", num.age_cl3)))
```

Network formation and dissolution parameters

Now we define the formation and dissolution terms that will be used to create out temporal network later.

In this case we'll need:

- edges:[+1 statistic] expected # of edges in average (proxy for the mean degree)
- nodefactor("age_cl", base = 1) [+2 statistics] this will ignore the stat for class with most members (adults), this is recommended because each tie will count twice once for each node. nodefactor() allows us to set the expected number of edges per class/group. base = 1, sets class 1 to the base level (which I am at this moment unsure of what it is, may be avg degree may not be)
- nodematch(age_cl, diff = FALSE):[+1 statistic] expected number of edges of individuals from the same group, the diff flag allows you to set a single statistic that applies to everyone or you could also make it per class.
- concurrency: [+1 statistic] number of nodes expected to have 2 or more edges at any given time

You can read more about the options that exist to customise network formation in this paper. as well as in the original Epimodel methods paper.

```
# formation to make network
formation <- ~edges+nodefactor("age_cl", base = 1)+nodematch("age_cl", diff = FALSE)+concurrent

mean_degree = 2 # how many connections a person has in average
stat1 = (n/3)*mean_degree #number of expected edges per age_class in general
# these (below) need to average out to the mean degree above
mean_degree_age_cl1 = 2 # young
mean_degree_age_cl2 = 3.5 # adults
mean_degree_age_cl3 = 0.5 # old
stat2 = num.age_cl3*mean_degree_age_cl3
stat3 = num.age_cl1*mean_degree_age_cl1
# same-race edge, let's say that only 50% of connections are between people of same age class
same_race_edge = 0.5
stat4 = stat1*same_race_edge # number of same class edges per person (see above)
# number of concurrent edges (concurrent degree), number of same connections at the same time
stat5 = n*0.5
target.stats = c(stat1,stat2, stat3, stat4, stat5)
```

Below `d.rate` is the homogeneous (average) departure rate for nodes in the network.

```
coef.diss = dissolution_coefs(dissolution = ~offset(edges),
                              duration = 5, # avg duration of each node -> 100 steps
                              d.rate = 0.00525)
coef.diss
```

```
## Dissolution Coefficients
## ========================
## Dissolution Model: ~offset(edges)
## Target Statistics: 5
## Crude Coefficient: 1.386294
## Mortality/Exit Rate: 0.00525
## Adjusted Coefficient: 1.440077
```

Simulate network.

```
# I found that the coefs are in a order that I was not expecting
est1 <- netest(nw,
               formation,
               target.stats,
               coef.diss,
               edapprox = T, # I am going to set edapprox to FALSE because the relationships
               # will last much longer than what would satisfy the conditions to use edapprox
               verbose = F)
```

```
## In term 'nodefactor' in package 'ergm': Argument 'base' has been superseded by 'levels', and it is r
```

```
## Warning: `set_attrs()` is deprecated as of rlang 0.3.0
## This warning is displayed once per session.
```

```
## Unable to match target stats. Using MCMLE estimation.
```

```
## Starting maximum pseudolikelihood estimation (MPLE):
```

```
## Evaluating the predictor and response matrix.
```

```
## Maximizing the pseudolikelihood.
```

```
## Finished MPLE.
## Starting Monte Carlo maximum likelihood estimation (MCMLE):
## Iteration 1 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 2.378.
## Step length converged once. Increasing MCMC sample size.
## Iteration 2 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 0.1031.
## Step length converged twice. Stopping.
## Finished MCMLE.
## This model was fit using MCMC.  To examine model diagnostics and check
## for degeneracy, use the mcmc.diagnostics() function.
```

```
summary(est1)
```

```
##
## ==========================
## Summary of model fit
## ==========================
##
## Formula:   TARGET_STATS ~ edges + nodefactor("age_cl", base = 1) + nodematch("age_cl",
##     diff = FALSE) + concurrent
## <environment: 0x7fa5f19a56c8>
##
## Iterations:  2 out of 20
##
## Monte Carlo MLE Results:
##                          Estimate Std. Error MCMC % z value Pr(>|z|)
## edges                    -8.94074    0.13779      0 -64.886  < 1e-04 ***
## nodefactor.age_cl.old    -0.53332    0.18965      0  -2.812  0.00492 **
## nodefactor.age_cl.young   1.01748    0.06987      0  14.562  < 1e-04 ***
## nodematch.age_cl         -0.04821    0.08557      0  -0.563  0.57319
## concurrent                1.96788    0.13272      0  14.827  < 1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood was not estimated for this fit.
## To get deviances, AIC, and/or BIC from fit `object$fit` run
##   > object$fit<-logLik(object$fit, add=TRUE)
## to add it to the object or rerun this function with eval.loglik=TRUE.
##
## Dissolution Coefficients
## ========================
## Dissolution Model: ~offset(edges)
## Target Statistics: 5
## Crude Coefficient: 1.386294
## Mortality/Exit Rate: 0.00525
## Adjusted Coefficient: 1.440077
```

Network diagnostics

This step is to check everything is fine with how the newtork was set up.

```
cores = detectCores()-1

t0 = Sys.time()
dx = netdx(est1,
           nsims = 5,
           nsteps = 180, # simulating 6 months
           ncores = cores)
```

```
##
## Network Diagnostics
## -----------------------
## - Simulating 5 networks
## - Calculating formation statistics
## - Calculating duration statistics
## - Calculating dissolution statistics
##
```

```
print(paste(Sys.time()-t0, "to run the diagnostic simulations."))
```

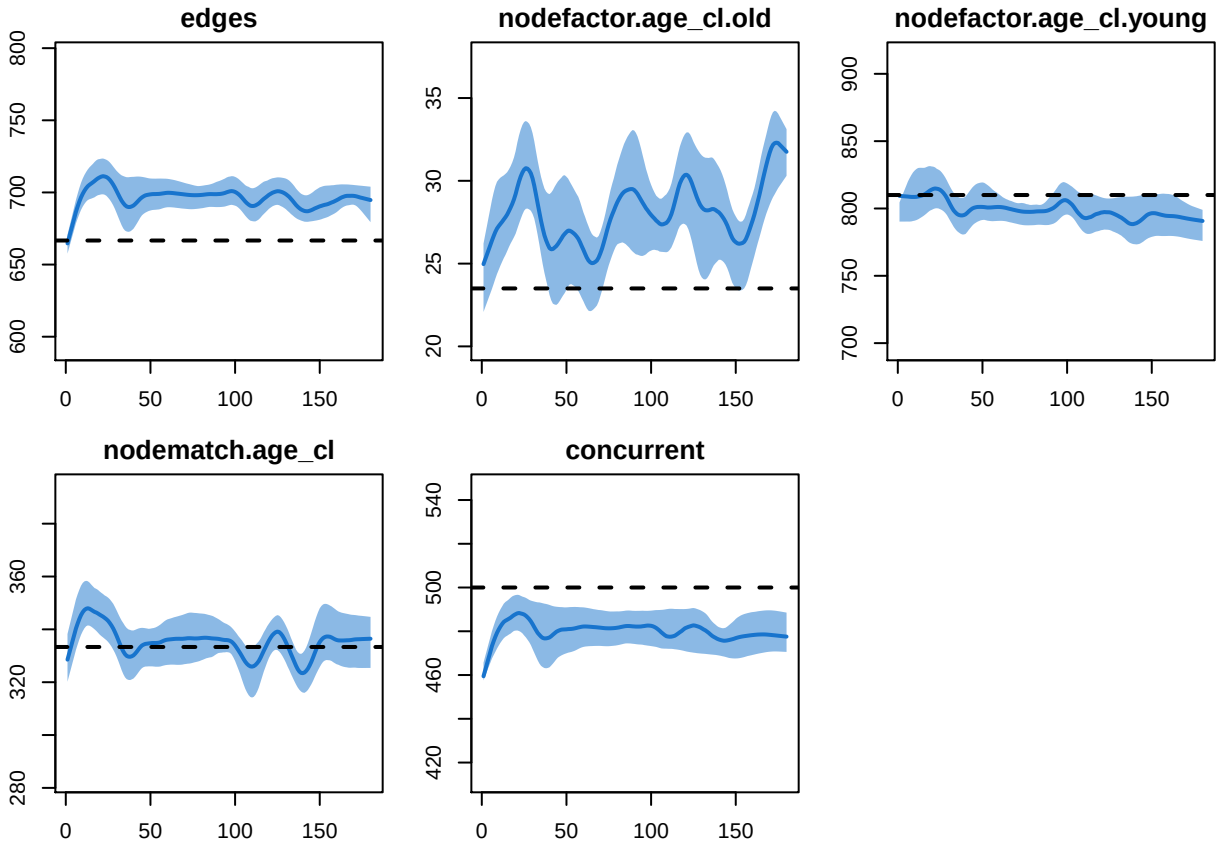
```
## [1] "28.0449020862579 to run the diagnostic simulations."
```

```
dx
```

```
## EpiModel Network Diagnostics
## ======================
## Diagnostic Method: Dynamic
## Simulations: 5
## Time Steps per Sim: 180
##
## Formation Diagnostics
## -----------------------
##                           Target Sim Mean Pct Diff Sim SD
## edges                    666.667  697.036    4.555 21.876
## nodefactor.age_cl.old     23.500   28.131   19.707  5.198
## nodefactor.age_cl.young  810.000  798.922   -1.368 26.101
## nodematch.age_cl         333.333  335.553    0.666 17.189
## concurrent               500.000  480.568   -3.886 18.318
##
## Dissolution Diagnostics
## -----------------------
##                 Target Sim Mean Pct Diff Sim SD
## Edge Duration      5.0    4.885   -2.307  4.335
## Pct Edges Diss     0.2    0.200    0.049  0.015
```

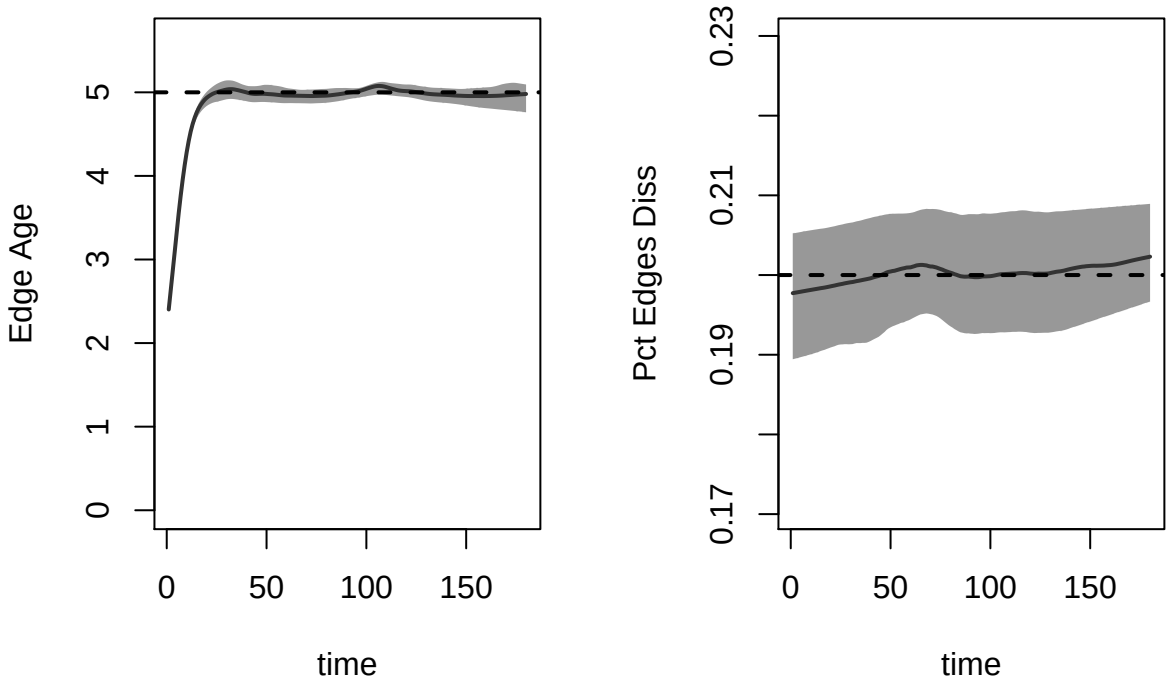
Diagnostic plots

```
plot(dx)
```



Difference between the target statistics and the estimated ones (from the simulation) may arise due to random error, in which case increasing the number of simulations will help or from systematic bias in which case the estimation approximation conditions may not hold. One reason for the estimation approximation not holding is if the dissolution coefficient is not small enough compared to the simulation length (as a rule of thumb a ratio of 1:20 between the dissolution coefficient and the simulation length should ensure a good model fit). If still a good dynamic fit is not achieved, one shoould go back to the static fit diagnostic (using `dynamic` argument in `netdx`) and if that fits then do a full STERGM estimation instead of the one using `edapprox = TRUE`.

```
par(mfrow = c(1,2))
plot(dx, "duration")
plot(dx, "dissolution")
```



This results are good as the target stastitics match pretty well, could probably do more simulation for a tigther fit.

Custom modules

We are going to make an SEIR models which is not one of the base models. Therefore we are going to need to make a new Exposed module and also change the Infection module.

Changing infection module

Code from this paper supplements (R replication code)

```
infect <- function(dat, at) {
  #' This function describes how people get infected, thus becoming
  #' exposed (in an SEIR model)

  # select those that are active (aka alive)
  active <- dat$attr$active
  # get their status (s, i, r...)
  status <- dat$attr$status
  # get the network
  nw <- dat$nw

  # get the ids of the infected
  idsInf <- which(active == 1 & status == "i")
  # get the number of active nodes
  nActive <- sum(active == 1)

  # get the number of infected (and set as eligible?)
  nElig <- length(idsInf)
  nInf <- 0
```

```
  if (nElig > 0 && nElig < nActive) {
    #discordant edgelist - a matrix of ID numbers of active dyads
    # in the network in which one member of the dyad is susceptible
    # and the other is infected
    del <- discord_edgelist(dat, at)
    if (!(is.null(del))) { # if del exists
      # set transmission probability to infection prob
      del$transProb <- dat$param$inf.prob
      # get act rate (~contact rate)
      del$actRate <- dat$param$act.rate
      # calculate prob of infection from standard formula
      del$finalProb <- 1 - (1 - del$transProb)^del$actRate
      # draw samples from binomial (Bernouilli) distribution with
      # final prob
      transmit <- rbinom(nrow(del), 1, del$finalProb)
      # select those that have transmitted according to the trial
      del <- del[which(transmit == 1), ]
      # Get new infected
      idsNewInf <- unique(del$sus)
      # update number of infected
      nInf <- length(idsNewInf)

      if (nInf > 0) {
        # set new infected as "e", for exposed
        dat$attr$status[idsNewInf] <- "e"
        # store infection time
        dat$attr$infTime[idsNewInf] <- at
      }
    }
  }

  if (at == 2) {
    # set flow of susceptiple to infected for at 1 and at 2
    # remember at 1 the model is initialised nothing actually happens
    dat$epi$se.flow <- c(0, nInf)
  }
  else {
    dat$epi$se.flow[at] <- nInf
  }
  dat$nw <- nw
  return(dat)
}
```

Construct a new disease progression module

```
## New disease progression module
progress <- function(dat, at) {
# This function dictates transition between states, in this case
# tranisiton between exposed and infected and between infected and recovered
# this function makes the recovery.net default function obsolete

  ####### get the eligible
```

```r
# active nodess
active <- dat$attr$active
# get status too (s, i, r, e)
status <- dat$attr$status

# get the ei rate and ir rate (Exposed to infected and infected to recovered)
ei.rate <- dat$param$ei.rate
ir.rate <- dat$param$ir.rate

# also gonna make de.rate and dr.rate
de.rate = dat$param$de.rate
dr.rate = dat$param$dr.rate

## E to I progression
nInf <- 0 # initialise number of infected variable
# get IDs of those that are eligible for infection (aka active nodes that are exposed)
# in this case exposure is a prerequisite to be infected
idsEligInf <- which(active == 1 & status == "e")
# get number of people that are eligible for infection
nEligInf <- length(idsEligInf)


if (nEligInf > 0) { # if anyone is eligible
  # pick those that will be infected accordging to a bernouilli trial
  vecInf <- which(rbinom(nEligInf, 1, ei.rate) == 1) # like the toss of a coin
  if (length(vecInf) > 0) { # if by the trial results somepeople were picked
    idsInf <- idsEligInf[vecInf]
    # get the id for those and transition them from "e" to "i"
    nInf <- length(idsInf)
    status[idsInf] <- "i"
  }
}

## I to R progression
# same as before but f infected to recovered transition
nRec <- 0
idsEligRec <- which(active == 1 & status == "i")
nEligRec <- length(idsEligRec)

if (nEligRec > 0) {
  vecRec <- which(rbinom(nEligRec, 1, ir.rate) == 1)
  if (length(vecRec) > 0) {
    idsRec <- idsEligRec[vecRec]
    nRec <- length(idsRec)
    status[idsRec] <- "r"
  }
}

# dat$attr$status <- status

# I should maybe modify the departure.net function insteead of this not sure

## E to D progression
```

```
# same as before but for exposed to dead transition
nDE <- 0
idsEligDE <- which(active == 1 & status == "e")
nEligDE <- length(idsEligDE)

if (nEligDE > 0) {
  vecDE <- which(rbinom(nEligDE, 1, de.rate) == 1)
  if (length(vecDE) > 0) {
    idsDE <- idsEligDE[vecDE]
    nDE <- length(idsDE)

    status[idsDE] = "d"

    dat$attr$active[idsDE] <- 0
    dat$attr$exitTime[idsDE] <- at
    dat$nw <- deactivate.vertices(dat$nw, onset = at, terminus = Inf,
                                  v = idsDE, deactivate.edges = TRUE)
  }
}

## R to D progression
# same as before but for recovered to dead transition
nDR <- 0
idsEligDR <- which(active == 1 & status == "r")
nEligDR <- length(idsEligDR)

if (nEligDR > 0) {
  vecDR <- which(rbinom(nEligDR, 1, dr.rate) == 1)
  if (length(vecDR) > 0) {
    idsDR <- idsEligDR[vecDR]
    nDR <- length(idsDR)

    status[idsDR] = "d"

    dat$attr$active[idsDR] <- 0
    dat$attr$exitTime[idsDR] <- at
    dat$nw <- deactivate.vertices(dat$nw, onset = at, terminus = Inf,
                                  v = idsDR, deactivate.edges = TRUE)
  }
}

dat$attr$status <- status

if (at == 2) {
  dat$epi$ei.flow <- c(0, nInf)
  dat$epi$ir.flow <- c(0, nRec)
  dat$epi$de.flow <- c(0, nDE)
  dat$epi$dr.flow <- c(0, nDR)
  dat$epi$e.num <- c(0, sum(active == 1 & status == "e"))
  dat$epi$r.num <- c(0, sum(active == 1 & status == "r"))
}
else {
  dat$epi$ei.flow[at] <- nInf
```

```
    dat$epi$ir.flow[at] <- nRec
    dat$epi$de.flow[at] <- nDE
    dat$epi$dr.flow[at] <- nDR
    dat$epi$e.num[at] <- sum(active == 1 & status == "e")
    dat$epi$r.num[at] <- sum(active == 1 & status == "r")
  }

  return(dat)
}
```

Custom death module - age-specific death rate

To do later once simpler models are developed.

Epidemic simulation

Now that we are satisfied with our diagnostics, we will model an epidemic on top of our dynamic network. First we need to set the parameters that define our pandemic. These are things such as the infection rate and infection probability upon contact of two individuals.

```
print(Sys.time() - t0)
```

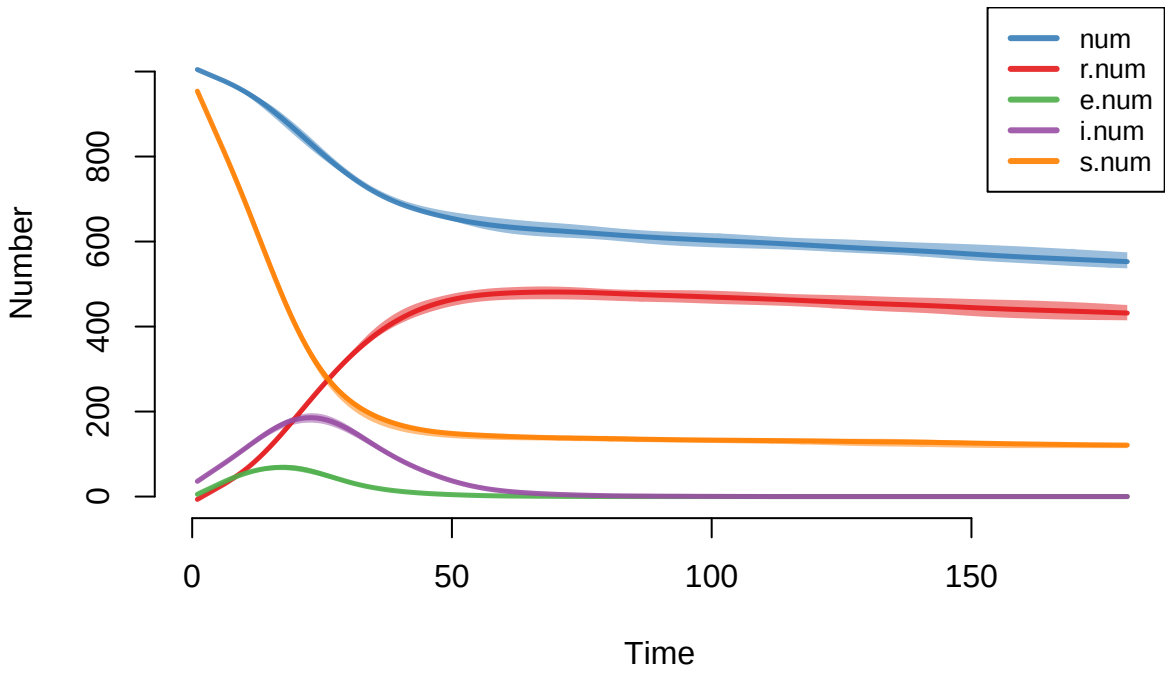
```
## Time difference of 2.284591 mins
```

```
print("to run the network simulation.")
```

```
## [1] "to run the network simulation."
```

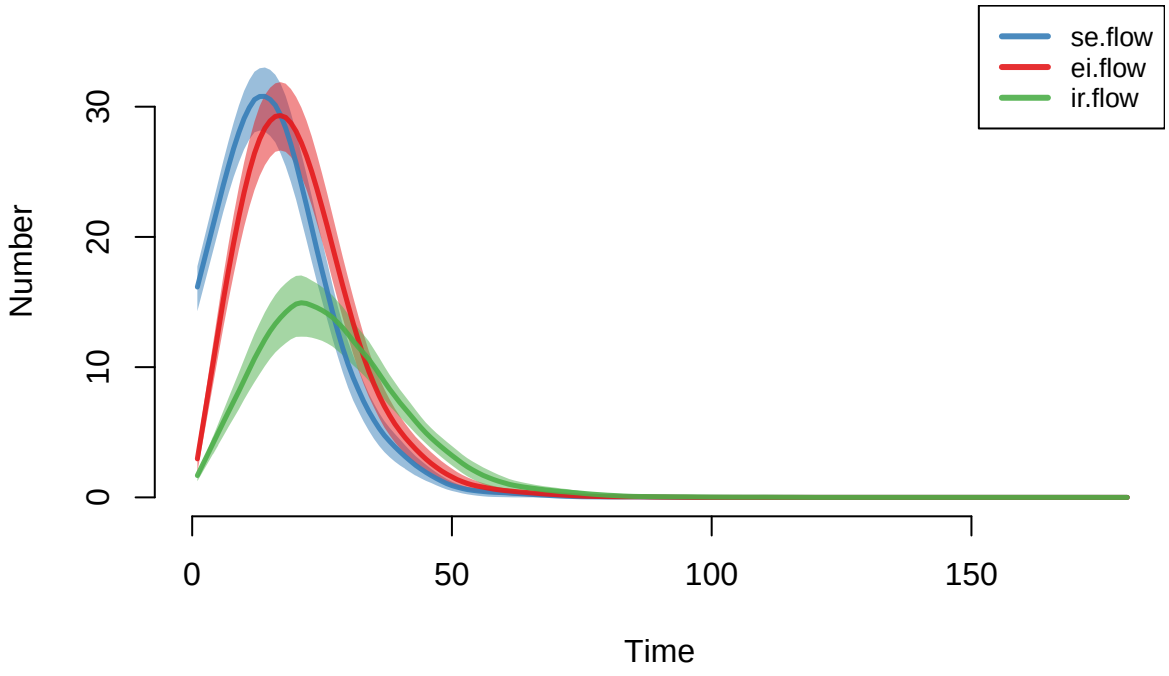
Plot absolute numbers

```
plot(sim1, y = c("num", "r.num", "e.num", "i.num", "s.num"), legend = T) # type epi is the default
```



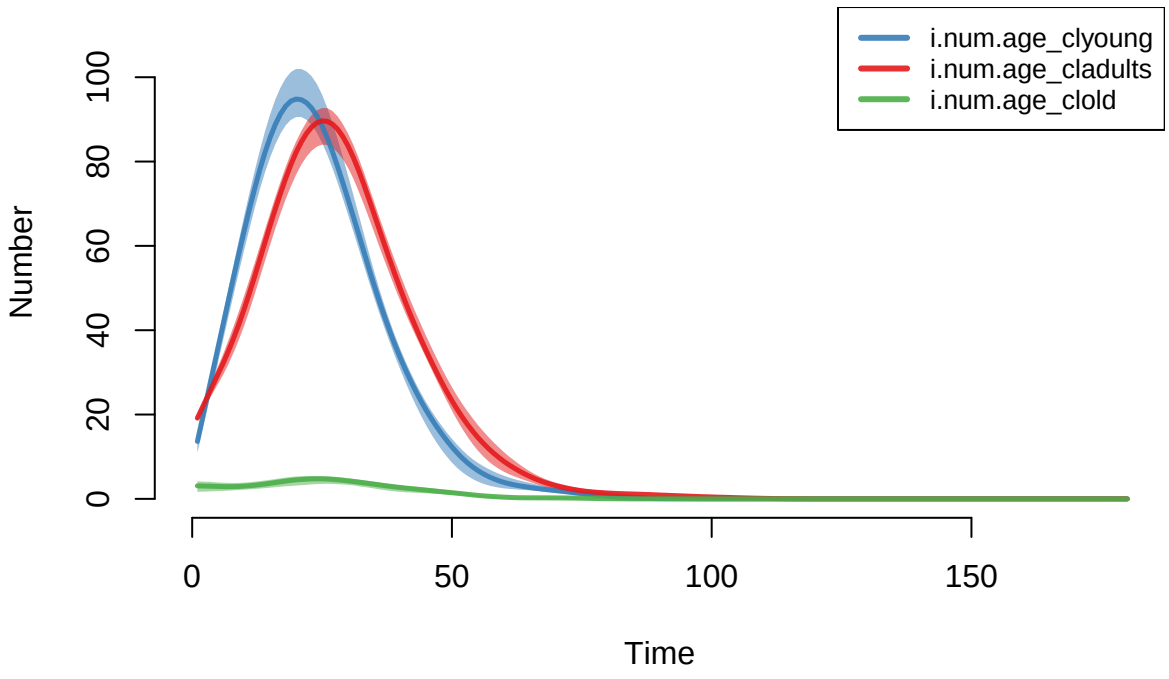
Plot flows

```
plot(sim1, y  = c("se.flow","ei.flow", "ir.flow"), legend = T)
```



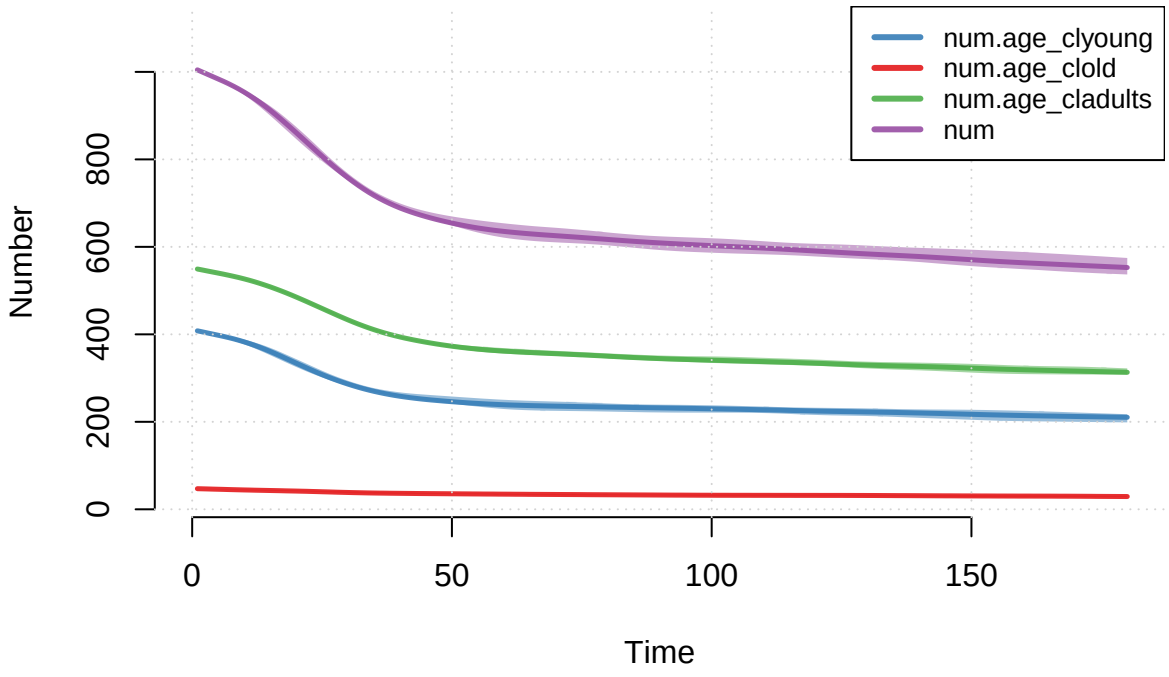
Plots by categories

```
plot(sim1, y = c("i.num.age_clyoung", "i.num.age_cladults", "i.num.age_clold"),
     legend = T)
```



Plot number by category

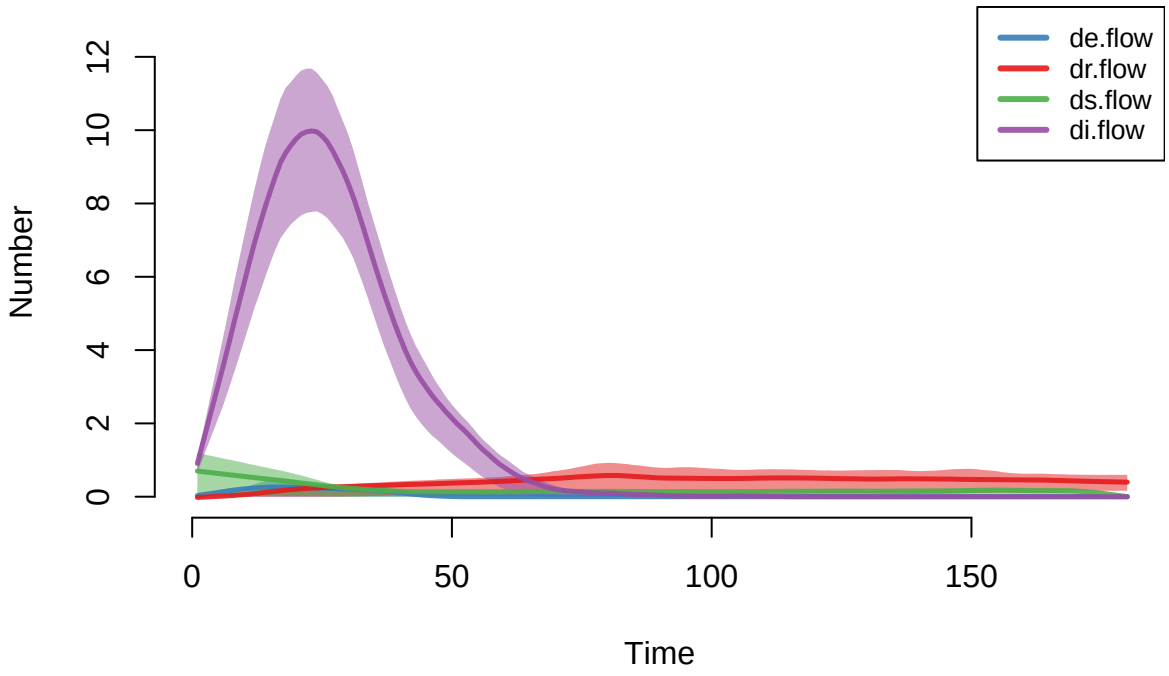
```
plot(sim1, y = c("num.age_clyoung", "num.age_clold", "num.age_cladults", "num"),
     legend = T, grid = T)
```



Plot deaths

Flows are not super useful

```
plot(sim1, y = c("de.flow", "dr.flow", "ds.flow", "di.flow"), legend = T)
```



```
# plotting options
#?plot.netsim
#plot(sim1, "network", at = 1, sims = "mean", col.status = T, legend = T)
# plot(sim1, "network", at = 500, sims = "mean", col.status = T, legend = T)
```

Retrieve the whole dataframe for further analysis.

```
head(as.data.frame(sim1))
```

```
##   sim time s.num s.num.age_clyoung s.num.age_cladults s.num.age_clold i.num
## 1   1    1   950               390                517              43    50
## 2   1    2   921               375                505              41    42
## 3   1    3   896               359                496              41    49
## 4   1    4   875               349                485              41    55
## 5   1    5   858               339                479              40    59
## 6   1    6   838               322                476              40    64
##   i.num.age_clyoung i.num.age_cladults i.num.age_clold  num num.age_clyoung
## 1                15                 31               4 1000             405
## 2                14                 26               2  998             405
## 3                21                 27               1  996             404
## 4                24                 30               1  990             402
## 5                25                 33               1  985             401
## 6                31                 31               2  982             401
##   num.age_cladults num.age_clold ei.flow ir.flow de.flow dr.flow e.num r.num
## 1              548            47       0       0       0       0     0     0
## 2              546            47       0       6       0       0     0     6
## 3              545            47      12       4       1       0    16    10
## 4              541            47      11       1       0       0    30    11
## 5              538            46      12       4       0       0    37    15
## 6              535            46      16       8       0       0    37    23
##   ds.flow di.flow a.flow se.flow
## 1       0       0      0       0
## 2       0       2      0      29
## 3       0       1      0      25
## 4       2       4      0      19
## 5       1       4      0      16
## 6       0       3      0      20
```