SEIQRHF Network Model

Luis Chaves

```
## Warning: package 'gt' was built under R version 3.6.2
```

Sourcing custom modules

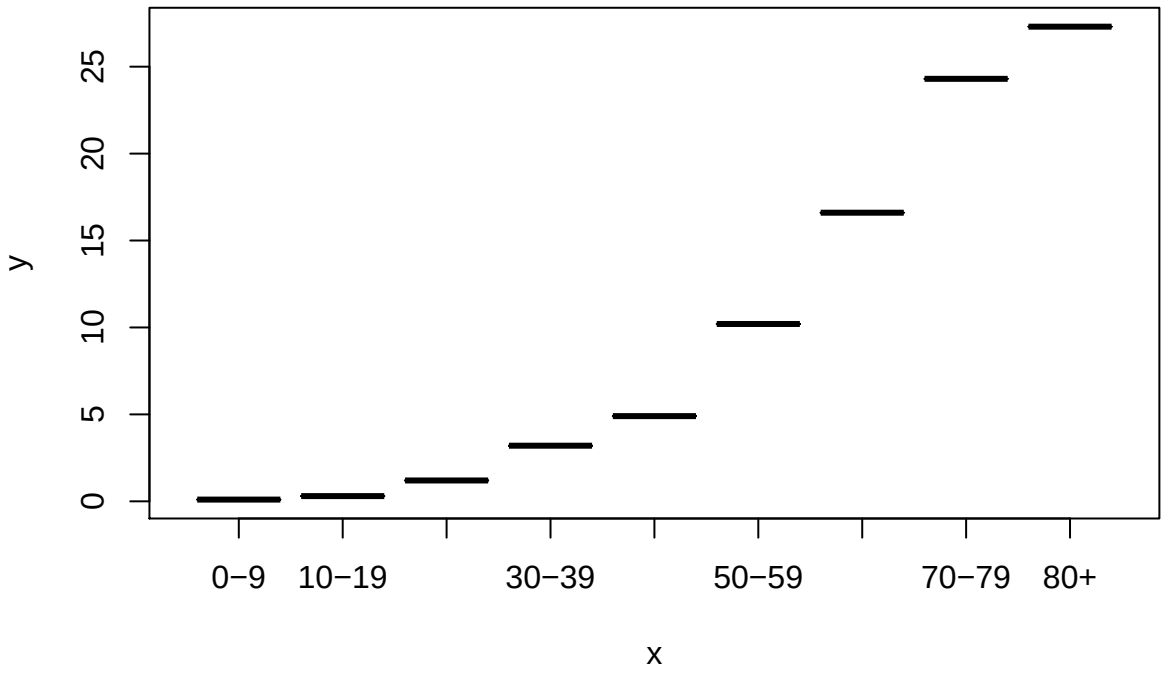
```
source("SEIQHRFNetModules.R")
```

Setting very basic network

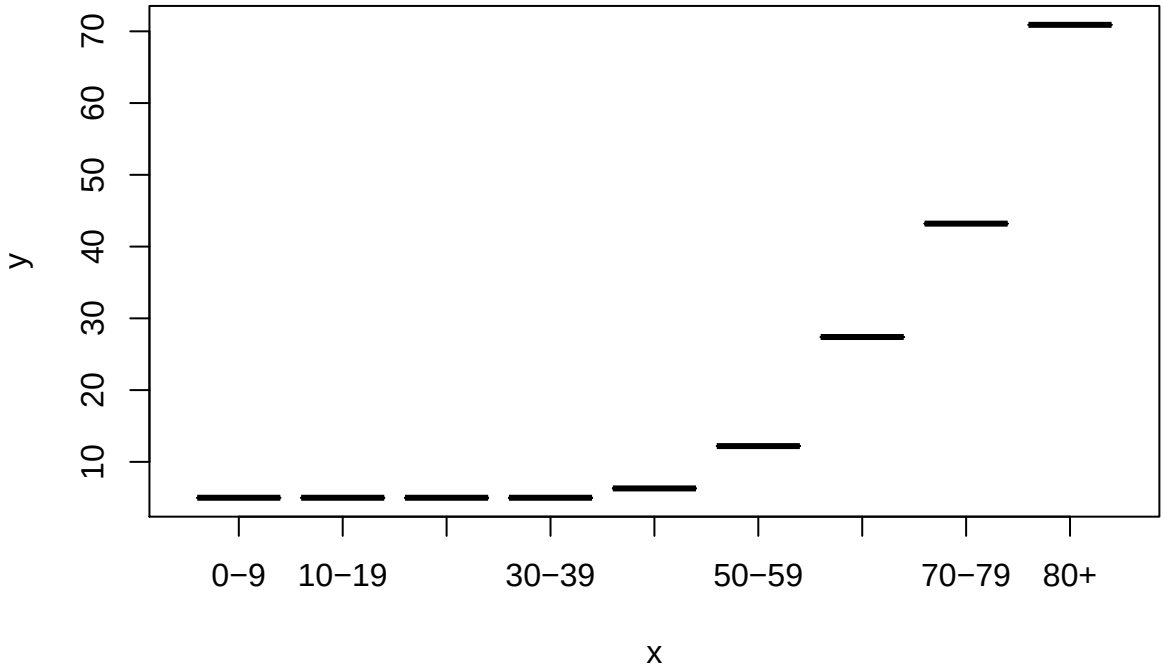
```
n = 1000
nw = network.initialize(n = n, directed = FALSE)
```

Getting nodal attributes from data

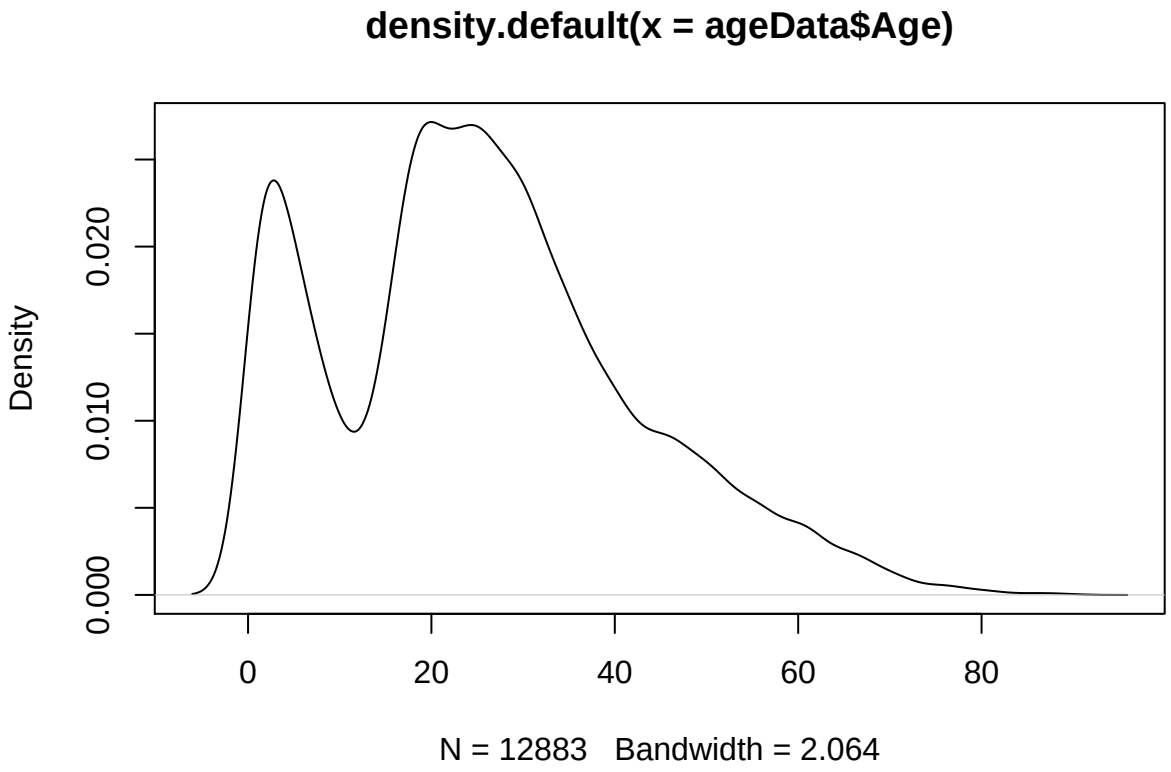
```
ageData = read.csv("age_and_sex.csv") %>% rename(Age = V1, Gender = V2, ID = X)

# I'm just taking Camp 1 data as it seems more complete
campParams = read.csv("camp_params.csv") %>% rename(Pop_Proportion = Value) %>%
  filter(Camp  == "Camp_1", Variable == "Population_structure")
campParams$Age = gdata::drop.levels(campParams$Age)
ageGroups = campParams %>%
  select(Age) %>% as.matrix()

# I'm assuming age groups to be left inclusive
# and right exclusive but probably does not matter tooo much
ageData$ageGroup = cut(ageData$Age, breaks = c(0,10,20,30,40,50,60,70,80, Inf))

plot(campParams$Age, campParams$Hosp_given_symptomatic)
```




```
plot(campParams$Age, campParams$Critical_given_hospitalised)
```




```
plot(density(ageData$Age))
```




```
paramsFromData = list()
paramsFromData$age.dist = ageData$ageGroup
# the two below are in same order as age groups
paramsFromData$rates.byAge = data.frame(AgeGroup = levels(ageData$ageGroup),
                                        hosp.rate = campParams$Hosp_given_symptomatic,
                                        fat.rate = campParams$Critical_given_hospitalised)
```

Setting network structure based on how refugees are allocated to tents

Based on Tucker model (from Manchester U.) Each individual is a member of a household that occupies either an isoboxor a tent. Isoboxes are prefabricated housing units with a mean occupancyof 10 individuals. Tents have a mean occupancy of 4 individuals. A total of 8100 individuals occupy isoboxes, and 10,600 individuals occupy tents. The exact occupancy of each isobox or tent is drawn from a Poisson distribution, and individuals are assigned to isoboxes or tents randomly without regard to sex or age. This is appropriate because many people arrive at Moria travelling alone, and thus isoboxes or tents may not represent family units

```
prop.isobox = round(8100/(8100+10600),2)
prop.tent = 1 - prop.isobox

stopifnot(round(n*prop.isobox+n*prop.tent)==n)
residence = c(rep("isobox", n*prop.isobox), rep("tent", n*prop.tent))
nw = set.vertex.attribute(nw, "residence", residence)
```

Settting nodal attributes

```
nw = set.vertex.attribute(nw, "age", sample(as.vector(paramsFromData$age.dist),n))
```

Explanation of the formation terms (documentation can be found by running `help(edge.terms)` and choosing the ergm option). We'll explain here what some basic terms are and should be:

- `edges`: This term adds one network statistic equal to the number of edges (i.e. nonzero values) in the network. For undirected networks, edges is equal to kstar(1); for directed networks, edges is equal to both ostar(1) and istar(1).
- `concurrent`: This term adds one network statistic to the model, equal to the number of nodes in the network with degree 2 or higher. The optional term attrname is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then the count is the number of nodes with ties to at least 2 other nodes with the same value for that attribute as the index node. This term can only be used with undirected networks. *`isolates`: This term adds one statistic to the model equal to the number of isolates in the network. For an undirected network, an isolate is defined to be any node with degree zero. For a directed network, an isolate is any node with both in-degree and out-degree equal to zero.
- `meandeg` — Mean vertex degree: This term adds one network statistic to the model equal to the average degree of the vertices. Note that this term is a constant multiple of both edges and density.
- `degree(d, attrname)` — Degree: The d argument is a vector of distinct integers. This term adds one network statistic to the model for each element in d; the ith such statistic equals the number of nodes in the network of degree d[i], i.e. with exactly d[i] edges. The term attrname is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then the degree count is the number of nodes with the same value of the attribute as the ego node. This term can only be used with undirected networks.
- `nodemix(attrname, base = NULL)` — **Nodal Attribute Mixing:** The attrname ar- gument is a character string giving the name of a categorical attribute in the network's vertex attribute list. **This term adds one network statistic to the model for each possible pairing of attribute values.The statistic equals the number of edges in the network in which the nodes have that pairing of values**. In other words, this term produces one statistic for every entry in the mixing matrix for the attribute. The ordering of the attribute values is alphabetical (for nominal categories) or numerical (for ordered categories). The optional base argument is a vector of integers corresponding to the pairings that should not be included. If base contains only negative integers, then these integers correspond to the only pairings that should be included. By default (i.e., with base = NULL or base = 0), all pairings are included.
- `nodematch(attrname, diff = FALSE, keep = NULL)` — **Uniform homophily and differential homophily:** The attrname argument is a character string giving the name of an attribute in the network's vertex attribute list. When diff = FALSE, this term adds one network statistic to the model, which counts the number of edges (i, j) for which attrname(i) == attrname(j). When diff = TRUE, p network statistics are added to the model, where p is the number of unique values of the attrname attribute. The **kth such statistic counts the number of edges (i, j) for which attrname(i) == attrname(j) == value (k)**, where value(k) is the kth smallest unique value of the attribute. If set to non-NULL, the optional keep argument should be a vector of integers giving the values of k that should be considered for matches; other values are ignored (this works for both diff = FALSE and diff = TRUE. For instance, to add two statistics, counting the matches for just the 2nd and 4th categories, use nodematch with diff = TRUE and keep = c(2,4).
- `density`: This term adds one network statistic equal to the density of the network. For undirected networks, density equals kstar(1) or edges divided by n(n-1)/2; for directed networks, density equals edges or istar(1) or ostar(1) divided by n(n-1).
- `nodefactor(attrname, base = 1)` — **Main effect of a factor attribute:** The attrname argument is a character string giving the name of a categorical attribute in the network's vertex attribute list. This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the attrname attribute. Each of these statistics gives the number of times a vertex with that attribute appears in an edge in the network. In particular, for edges whose endpoints both have

the same attribute value, this value is counted twice. To include all attribute values is usually not a good idea, because the sum of all such statistics equals twice the number of edges and hence a linear dependency would arise in any model also including edges. Thus, the base argument tells which value(s) (numbered in order according to the sort func- tion) should be omitted. The default value, one, means that the smallest (i.e., first in sorted order) attribute value is omitted. For example, if the "fruit" factor has levels "orange", "apple", "banana", and "pear", then to add just two terms, one for "apple" and one for "pear", set "banana" and "orange" to the base (remember to sort the values first) by using nodefactor("fruit", base = 2:3). For an analogous term for quantitative vertex attributes, see nodecov.

- `nodecov(attrname)` — **Main effect of a covariate:** The attrname argument is a character string giving the name of a quantitative (not categorical) attribute in the network's vertex attribute list. This term adds a single network statistic to the model equaling the sum of attrname(i) and attrname(j) for all edges (i, j) in the network. For categorical attributes, see node
- `sociality` - **Undirected degree:** This term adds one net- work statistic for each node equal to the number of ties of that node. The optional attrname argument is a character string giving the name of an attribute in the net- work's vertex attribute list that takes categorical values. If provided, this term only counts ties between nodes with the same value of the attribute (an actor-specific ver- sion of the nodematch term). This term can only be used with undirected networks. For directed networks, see sender and receiver. By default, base = 1 means that the statistic for the first node will be omitted, but this argument may be changed to control which statistics are included just as for the sender and receiver terms.

```
formation <- ~edges+
  concurrent+
  nodematch("residence", diff = TRUE)+
  nodemix("residence", base = c(1,3))

mean_degree = 2
concurrent_percentage = 0.5 # % of nodes (people) with a degree of 2 or larger

edges = 1200
concurrent_nodes = 600
residence.iso = 20*20
residence.tent = 15*20
residence.mixing = 6*20
target.stats = c(edges,
                 concurrent_nodes,
                 residence.iso,
                 residence.tent,
                 residence.mixing)

d.rate = 0.0001
coef.diss = dissolution_coefs(dissolution = ~offset(edges),
                              duration = 30,
                              d.rate = d.rate) # this correspond to external deaths
```

Building network and properly fitting network to stats

From `netest` documentation (help(netest)) The edges dissolution approximation method is described in Carnegie et al. This approximation requires that the dissolution coefficients are known, that the formation model is being fit to cross-sectional data conditional on those dissolution coefficients, and that the terms in the dissolution model are a subset of those in the formation model. Under certain additional conditions, the formation coefficients of a STERGM model are approximately equal to the coefficients of that same model fit

to the observed cross-sectional data as an ERGM, minus the corresponding coefficients in the dissolution model. The approximation thus estimates this ERGM (which is typically much faster than estimating a STERGM) and subtracts the dissolution coefficients.

The conditions under which this approximation best hold are when there are few relational changes from one time step to another; i.e. when either average relational durations are long, or density is low, or both. Conveniently, these are the same conditions under which STERGM estimation is slowest. Note that the same approximation is also used to obtain starting values for the STERGM estimate when the latter is being conducted. The estimation does not allow for calculation of standard errors, p-values, or likelihood for the formation model; thus, this approach is of most use when the main goal of estimation is to drive dynamic network simulations rather than to conduct inference on the formation model. The user is strongly encouraged to examine the behavior of the resulting simulations to confirm that the approximation is adequate for their purposes. For an example, see the vignette for the package tergm.

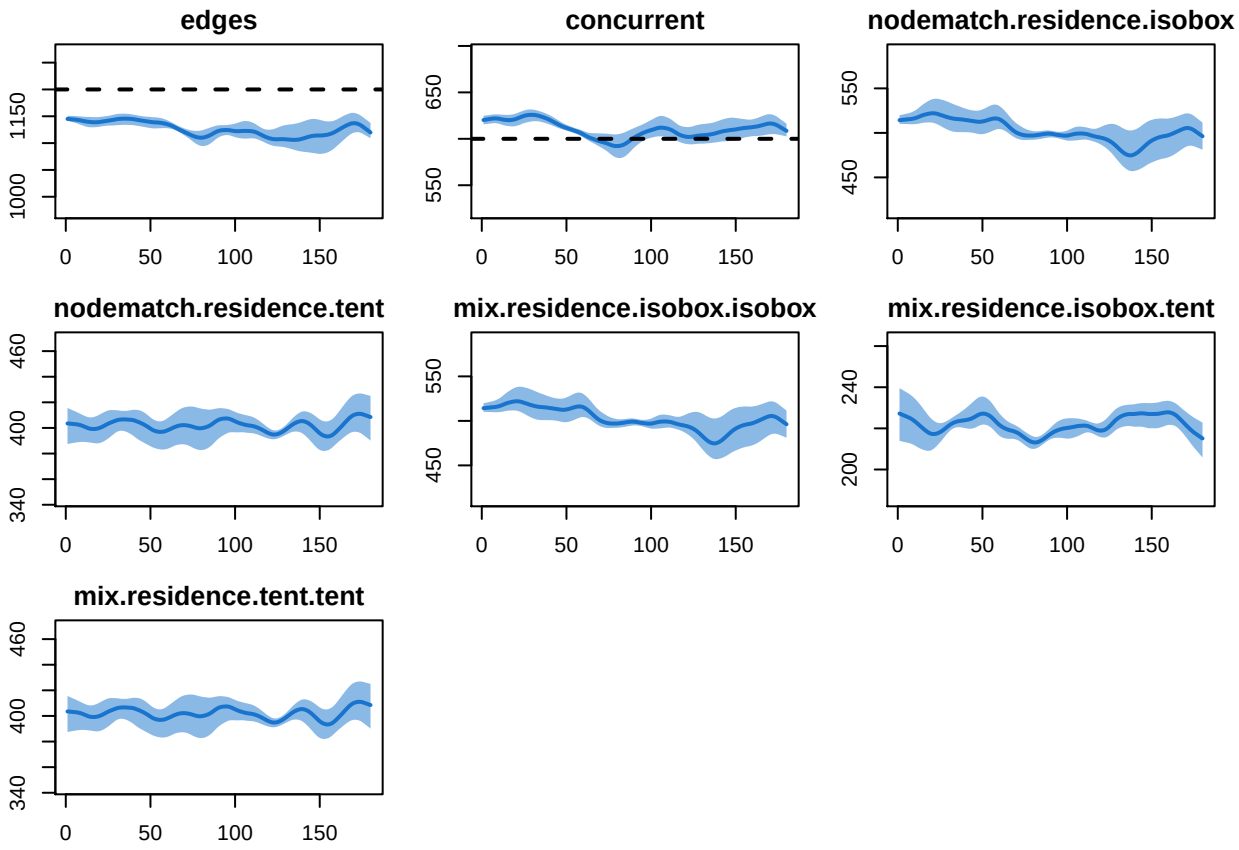
```
## Warning: `set_attrs()` is deprecated as of rlang 0.3.0
## This warning is displayed once per session.
```

Diagnostics

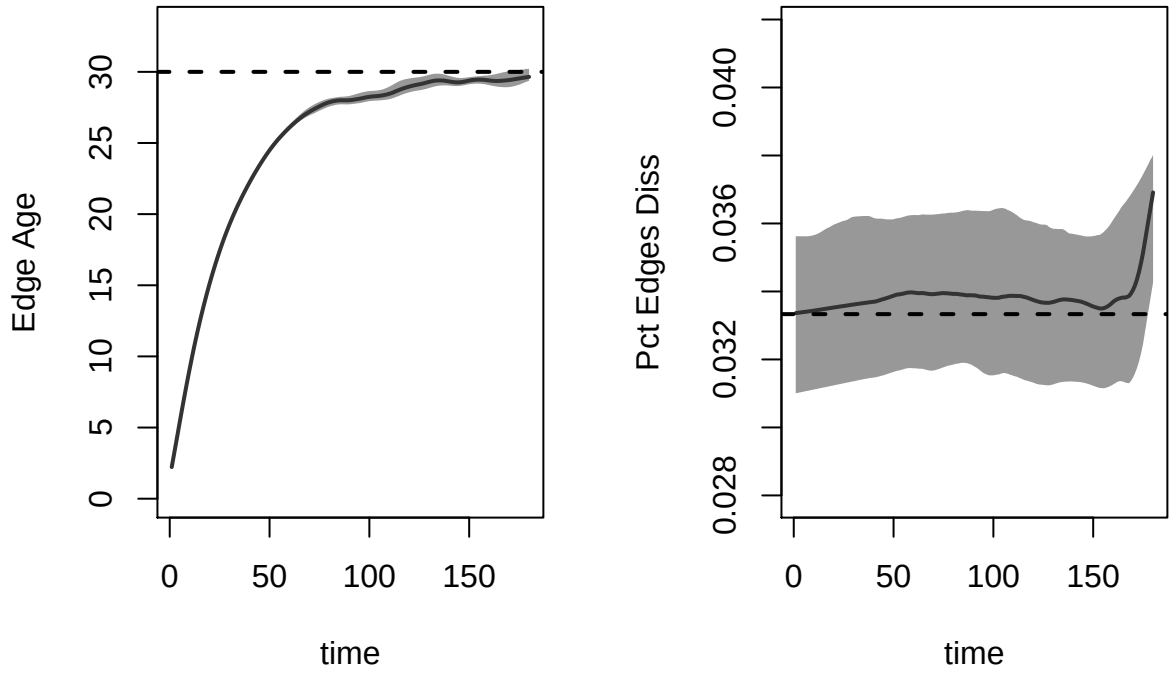
```
cores = parallel::detectCores()-1

dx = netdx(est1,
           nsims = 3,
           nsteps = 180, # simulating 6 months
           ncores = cores,
           nwstats.formula = ~edges+concurrent+nodematch("residence", diff = TRUE)+nodemix("residence"))
```

```
##
## Network Diagnostics
## -----------------------
## - Simulating 3 networks
## - Calculating formation statistics
## - Calculating duration statistics
## - Calculating dissolution statistics
##
```

```
plot(dx)
```




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



```
dx
```

```
## EpiModel Network Diagnostics
## =======================
## Diagnostic Method: Dynamic
## Simulations: 3
## Time Steps per Sim: 180
##
## Formation Diagnostics
## ----------------------
##                              Target Sim Mean Pct Diff Sim SD
## edges                          1200 1126.452   -6.129 32.289
## concurrent                      600  610.291    1.715 18.402
## nodematch.residence.isobox      400  502.596   25.649 28.596
## nodematch.residence.tent        300  402.000   34.000 20.150
## mix.residence.isobox.isobox      NA  502.596       NA 28.596
## mix.residence.isobox.tent       120  221.856   84.880 11.585
## mix.residence.tent.tent          NA  402.000       NA 20.150
##
## Dissolution Diagnostics
## ------------------------
##                 Target Sim Mean Pct Diff Sim SD
## Edge Duration   30.000   24.020  -19.934 22.308
## Pct Edges Diss   0.033    0.034    1.330  0.005
```

Running epidemic

```
param = param.net(act.rate.se = 10,
                  inf.prob.se = 0.02,
                  act.rate.si = 10,
                  inf.prob.si = 0.05,
                  act.rate.sq = 2.5,
                  inf.prob.sq = 0.02,
                  ei.rate = 1/10,
                  iq.rate = 1/30, #c(rep(1/30, 60), rep(15/30, 120)), # time varying works
                  ih.rate = 1/100,
                  qh.rate = 1/100,
                  hr.rate = 1/15,
                  qr.rate = 1/20,
                  hf.rate = 1/50,
                  hf.rate.overcap = 1/25,
                  hosp.cap = 5,
                  hosp.tcoeff = 0.5,
                  a.rate = 0,
                  di.rate = d.rate,
                  ds.rate = d.rate,
                  dr.rate = d.rate,
                  ratesbyAge = paramsFromData$rates.byAge
                  )

init = init.net(i.num = 3,
                r.num = 0,
```

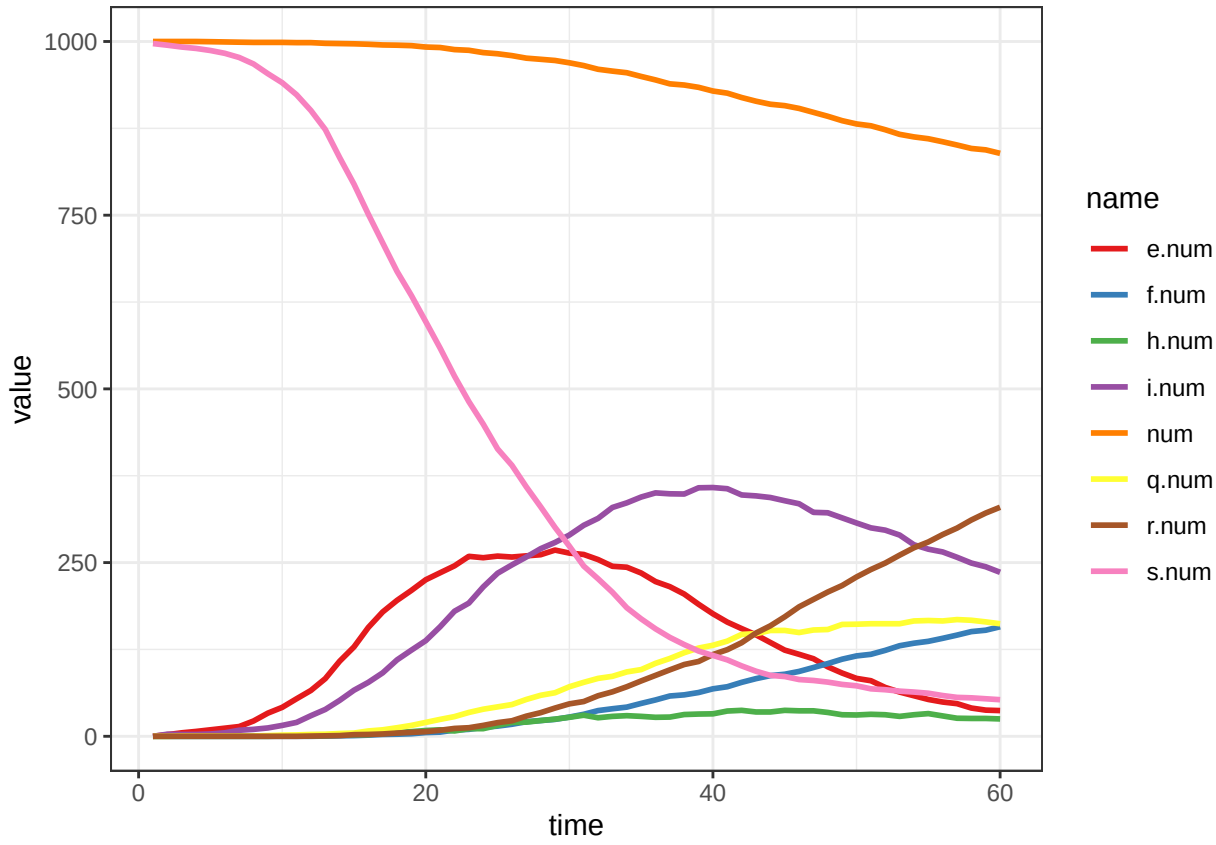
```
                e.num = 0,
                s.num = n - 3,
                f.num = 0,
                h.num = 0,
                q.num = 0
                )
```

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

```
## Time difference of 1.070401 mins
```

```
res = as.data.frame(sim1)

# The simulation time really goes up with the number of edges
```

```
res %>% select(s.num, e.num, i.num, q.num, h.num, r.num, f.num, num, time) %>%
  group_by(time) %>% summarise_all(~mean(.)) %>%
  pivot_longer(-time) %>% ggplot(aes(x = time, y = value, color = name))+
  geom_line(size = 1)+scale_color_brewer(palette = "Set1")
```

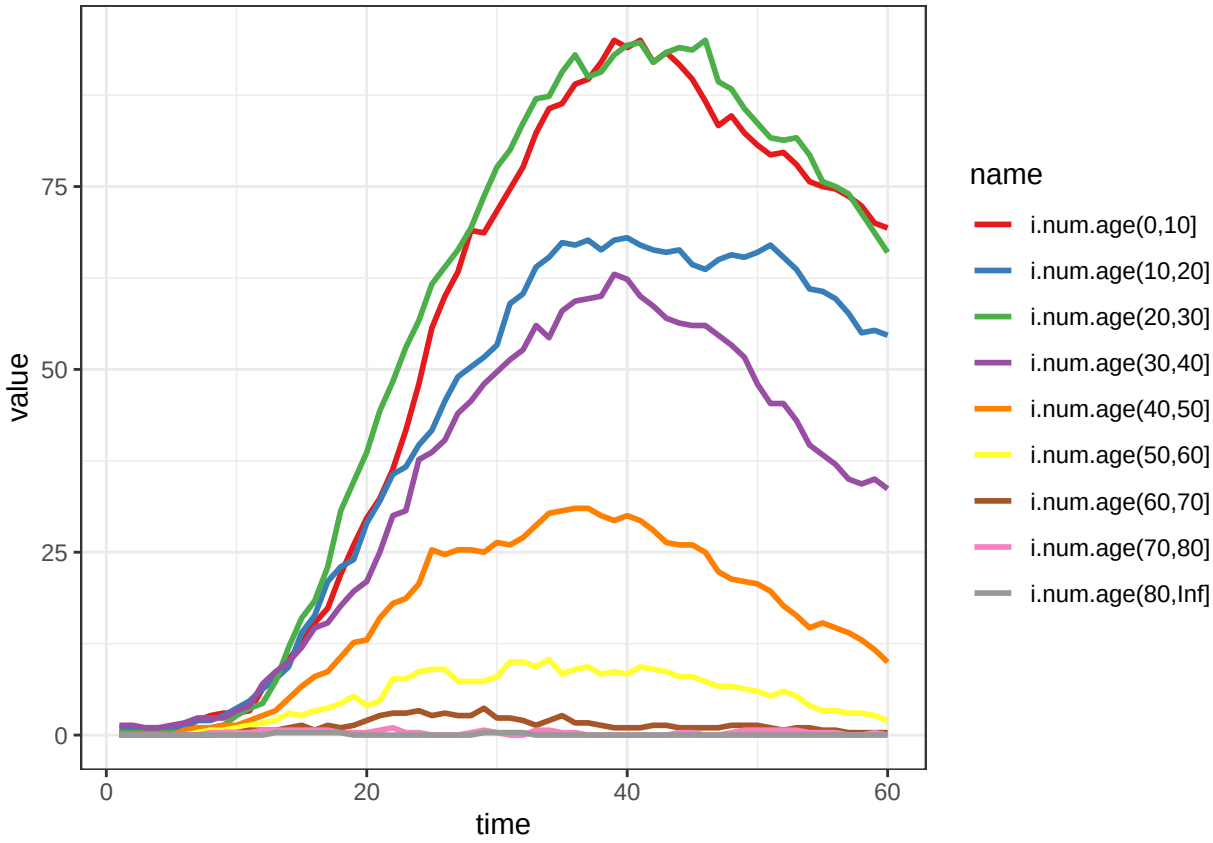



```
ggplotly(res %>% select(s.num, e.num, i.num, q.num, h.num, r.num, f.num, num, time) %>%
  group_by(time) %>% summarise_all(~mean(.)) %>%
  pivot_longer(-time) %>% ggplot(aes(x = time, y = value, color = name))+
  geom_line(size = 1)+scale_color_brewer(palette = "Set1"))
```

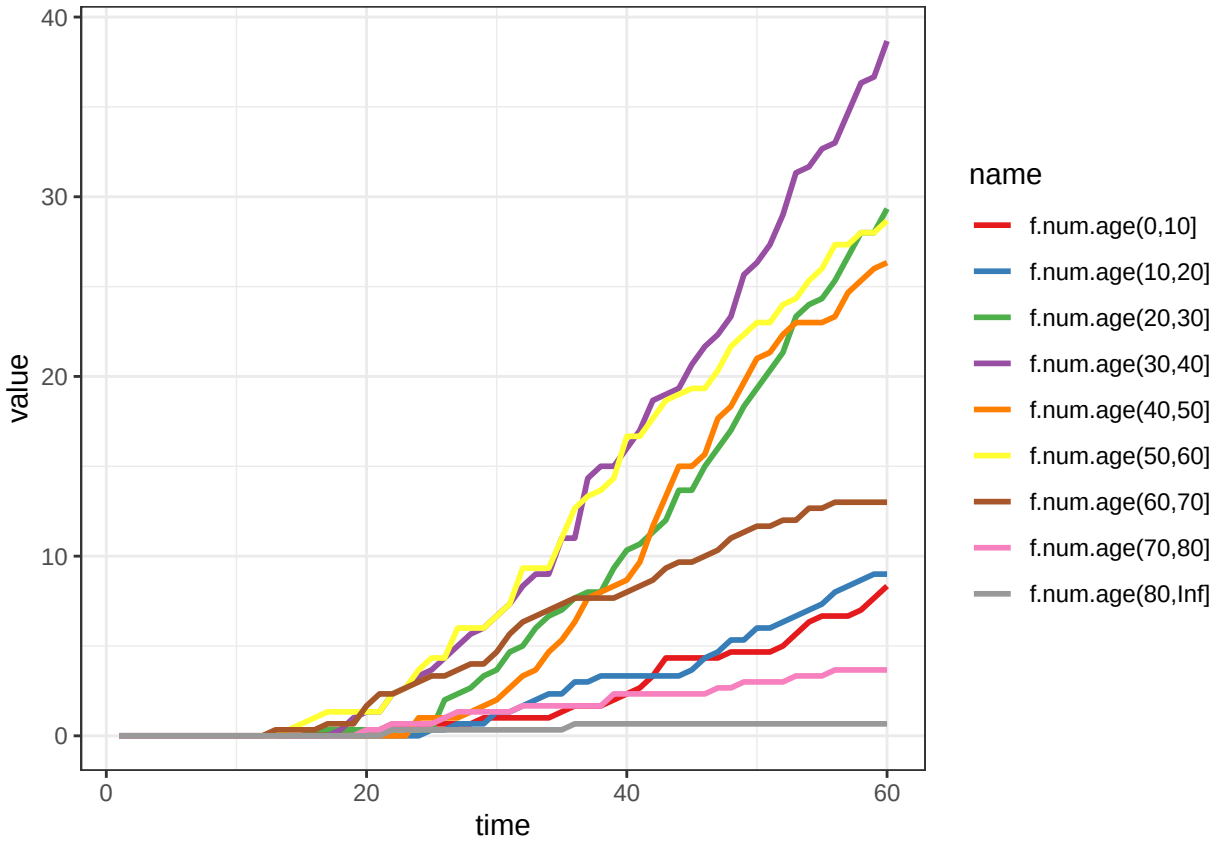
```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```

Plot by age groups

```
res %>% select(contains("i.num.age"), time) %>% group_by(time) %>% summarise_all(~mean(.)) %>%
  pivot_longer(-time) %>% ggplot(aes(x = time, y = value, color = name))+
  geom_line(size = 1)+scale_color_brewer(palette = "Set1")
```




```
res %>% select(contains("f.num.age"), time) %>% group_by(time) %>% summarise_all(~mean(.)) %>%
  pivot_longer(-time) %>% ggplot(aes(x = time, y = value, color = name))+
  geom_line(size = 1)+scale_color_brewer(palette = "Set1")
```



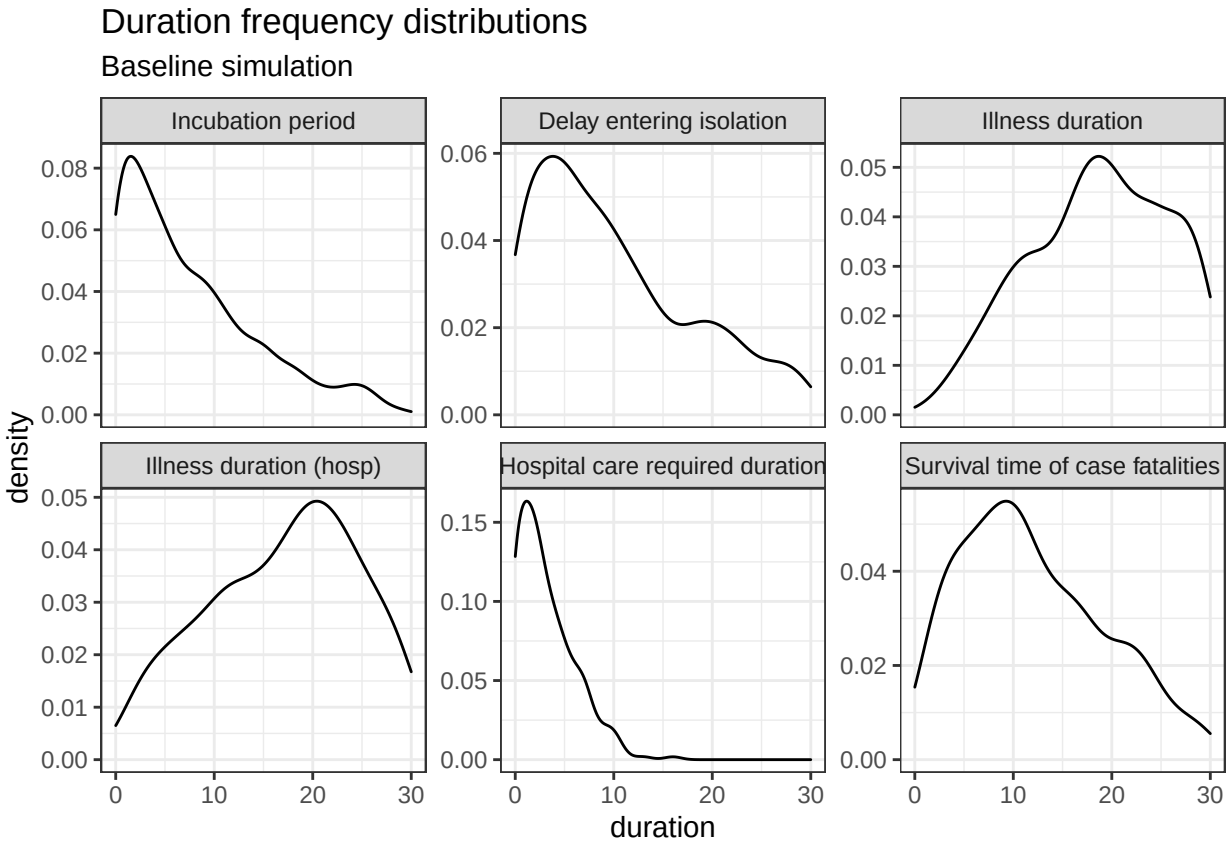

```
ggplotly(res %>% select(starts_with("num.age"), time) %>% group_by(time) %>% summarise_all(~mean(.)) %>%
  pivot_longer(-time) %>% ggplot(aes(x = time, y = value, color = name))+
  geom_line(size = 1)+scale_color_brewer(palette = "Set1"))
```

For diagnostics

```
get_times <- function(simulation.object) {

  sim <- simulation.object

  for (s in 1:sim$control$nsims) {
    if (s == 1) {
      times <- sim$times[[paste0("sim", s)]]
      times <- times %>% mutate(s = s)
    } else {
      times <- times %>% bind_rows(sim$times[[paste("sim",
                                                    s, sep = "")]] %>% mutate(s = s))
    }
  }

  times <- times %>%
    mutate(infTime = ifelse(infTime < 0, -5, infTime),
           expTime = ifelse(expTime < 0, -5, expTime)) %>%
    mutate(incubation_period = infTime - expTime,
           illness_duration = recTime - expTime,
```

```
            illness_duration_hosp = dischTime - expTime,
            hosp_los = dischTime - hospTime,
            quarantine_delay = quarTime - infTime,
            survival_time = fatTime - infTime) %>%
    select(s,
            incubation_period,
            quarantine_delay,
            illness_duration,
            illness_duration_hosp,
            hosp_los,
            survival_time) %>%
    pivot_longer(-s, names_to = "period_type", values_to = "duration") %>%
    mutate(period_type = factor(period_type,
                                levels = c("incubation_period",
                                           "quarantine_delay",
                                           "illness_duration",
                                           "illness_duration_hosp",
                                           "hosp_los",
                                           "survival_time"),
                                labels = c("Incubation period",
                                           "Delay entering isolation",
                                           "Illness duration",
                                           "Illness duration (hosp)",
                                           "Hospital care required duration",
                                           "Survival time of case fatalities"),
                                ordered = TRUE))
  return(times)
}
```

```
times = get_times(sim1)

times %>% filter(duration <= 30) %>% ggplot(aes(x = duration)) +
    geom_density() + facet_wrap(period_type ~ ., scales = "free_y") +
    labs(title = "Duration frequency distributions", subtitle = "Baseline simulation")
```


Duration frequency distributions
Baseline simulation
Incubation period
Delay entering isolation
Illness duration
Illness duration (hosp)
Hospital care required duration
Survival time of case fatalities
density
duration
0.08
0.06
0.04
0.02
0.00
0.05
0.03
0.01
0.15
0.10
0
10
20
30