A netdiffuseR Tutorial: Estimating Network Influences on Behavior within the Diffusion Paradigm

George G. Vega Yon Aníbal L. Olivera M. Thomas W. Valente

Abstract

This paper introduces diffusion network research and presents an R library, netdiffuseR, that facilitates the estimation of contagion and diffusion processes on networks. Diffusion of innovations theory explains how new ideas and practices spread within and between communities. The spreading process consists of person-to-person interactions in which adopters persuade (by whatever means) non-adopters to adopt. Estimating these effects requires network data and an indicator for when each person adopts. The analytic steps involve creating an event history version of the data set and regression estimation to determine if exposure, the adopters in one's network, is associated with individual adoption. netdiffuseR automates many of these data management and processing steps, thus facilitating the estimation of these models. The package also contains simple ways to compute various network exposure measures such as based on structural equivalence, indirect ties, homophilic ties, etc. In addition, netdiffuseR provides capabilities to conduct simulations of network diffusion processes varying network structure, seed conditions, threshold distributions, and influence mechanisms. The paper includes example code for both empirical and simulated network studies. We aim to facilitate the adoption of netdiffuseR to advance the growth and evolution of diffusion network research.

Introduction

This paper provides a general overview of the R package, netdiffuseR, a library for estimating contagion and diffusion processes on networks. We provide a brief introduction to diffusion of innovations theory, introduce netdiffuseR, provide a tutorial for empirical work, and examples of simulations. We also describe ways to analyze multiple simultaneous behavior diffusion such as adoption of complementary or competing products. Lastly we demonstrate how to model

disadoption. This paper enable researchers to use the netdiffuseR package to read, analyze, visualize, and simulate diffusion networks.

Diffusion of innovations is a theoretical approach that explains how new ideas and practices spread within and between communities. Diffusion of innovations theory has been used to study the spread of: fertility preferences and contraceptive choices (Bongaarts and Watkins 1996; Valente et al. 1997); risk behaviors such as smoking (Christakis and Fowler 2007); alcohol use (Light et al. 2013); substance use (Hoffman et al. 2006; Fujimoto and Valente 2013); physician behavior (Coleman, Katz, and Menzel 1966; Iyengar, Van Den Bulte, and Valente 2011); public health policies (Valente et al. 2015; Yamagata, Yang, and Galaskiewicz 2017); physical activity (Aral and Nicolaides 2017); among other behaviors. Thus, developing a platform to easily analyze and compare factors associated with network influences on adoption has broad applicability in the behavioral and physical sciences.

Diffusion theory states that a new idea or practice enters a community or population from some external source such as an impersonal medium, a person from another community, or a technological shift (Rogers 2003; Valente and Rogers 1995). The idea or practice then spreads from person to person through interpersonal communication. Thus, the theory and techniques of social network analysis are integral to the study of the diffusion of innovations (Valente 2010), and constitute the field of diffusion networks.

The workhorse variable used to measure network influences on adoption is network exposure which is the number or proportion of adopters in a person's immediate social network contemporaneously or lagged by one (or more) time periods (which depending on the study could be days, months, or years). The formula for the lagged exposure model is:

$$E_{it} = \frac{\sum_{i \neq j} W_{ij} a_{j(t-1)}}{\sum_{i \neq j} W_{ij}}$$

Where W_{ij} is a weight matrix indicating ties between individuals, most commonly a binary adjacency matrix such as who is friends with whom but can be any relation; a is an adoption vector indicating which of i's j alters had adopted at time $t-1$; and the denominator is the row sum, the number of outgoing ties. E_{it} in (1) measures the proportion of a person's ties at time t who have adopted the behavior at the prior time period, which is a contagion measure. As described in Valente (2024), however, there can be many extensions of the exposure calculation by estimating influence of ties based on: a count (removing the denominator), indirect connectivity, structural equivalence, strength (weak vs. strong), homophily, Simmelian relations (alter-to-alter), alter network properties such as centrality, etc. netdiffuseR enables users to calculate these alternative exposures with a parameter toggle in the exposure function.

There are other measures needed to estimate diffusion network effects. Network thresholds (Valente 1996) are the proportion or number of adopters in an individual's network needed for them to adopt. Generally, adoption thresholds coincide with their exposure at the time of adoption, nonetheless, because of timing, exposure can overestimate thresholds. Unless data

collection happens in real time, some error is expected as people may reach their adoption threshold earlier. For instance, if a person has a threshold of two connections, but three of its peers adopt simultaneously, then their exposure at the time of adoption will be three, but their threshold was two (see Berry et al. 2019 for a detailed discussion on the topic.).

Some people innovate with none or few adopters in their network adopting whereas others wait until a majority or all adopt. Network thresholds provide a means to estimate the classic two-step flow hypotheses (Katz 1957) and have been shown to moderate health promotion impacts (Valente and Saba 1998). In addition, the threshold distribution drives the diffusion rate and prevalence (Valente and Vega Yon 2020). Other diffusion network measures are Moran's I, a measure of contagion, network susceptibility, network influence (Valente et al. 2015), the hazard rate, and Bass model parameter coefficients (Bass 1969; Valente 1993).

The netdiffuseR package

The netdiffuseR R package (Valente and Vega Yon 2020; Vega Yon, Olivera Morales, and Valente 2025) was first released to CRAN on February 18, 2016. The package contains a collection of algorithms and methods first envisioned by Dr. Valente and released in his book (Valente 1995) as GAUSS code. Since its release in 2016, the package has been downloaded over 45 thousand times and has about 400 downloads a month, with an active community growing.

In a nutshell, netdiffuseR provides methods for simulating, processing, analyzing, and visualizing diffusion networks. Among its key functions are the `survey_to_diffnet()` function, `rdiffnet()` function, and the `exposure()` function. The `survey_to_diffnet()` function provides an interface to read in tabular data (in R `data.frame` format) corresponding to survey data containing network nominations and time of adoption [TOA]. The function makes it easy to process survey data, pass multiple checks, and build the adoption and cumulative adoption matrices. The `rdiffnet()` function allows simulating diffusion networks using the threshold model and, since version 1.23.0, simulating multi-diffusion and disadoption. Users can simulate diffusion processes on pre-defined networks or random graphs in a computationally efficient way. Finally, the `exposure()` function–the workhorse of the R package–provides a flexible framework for calculating various types of exposure measures as indicated above. In the following section of the paper, we will provide a demonstration of these functions.

A tutorial on netdiffuseR

This tutorial has been developed using **R** version 4.5.1 (R Core Team 2025) and the program **quarto** version 1.6.42 (Allaire et al. 2025). Readers can find the quarto document as well as all required material for reproducing the paper in https://github.com/gvegayon/netdiffuser-connections.

Analyzing existing data

First, we will show how to read survey data into netdiffuseR. We start by loading the package. If you don't have the package installed in your system, you can use the `install.packages()` function:

```
install.packages("netdiffuseR")
```

Notice that this is required to do only once. The previous code will install the package in your R system, but it will not load it. Once you are sure the package is available in your computer, you can use the function `library()` to load it:

```
# Loading the netdiffuseR
library(netdiffuseR)
```

```
Thank you for using netdiffuseR! Please consider citing it in your work.
You can find the citation information by running
  citation("netdiffuseR")
```

```
Attaching package: 'netdiffuseR'
```

```
The following object is masked from 'package:base':

    %*%
```

The package comes with a handful of datasets (the three classical diffusion datasets) that we can use as a starting point. Here, we will use the Korean Family Planning study (Rogers and Kincaid 1981). Let's take a first look at the data using the `data()`, `dim()`, and `str()` functions. The `data()` function loads datasets from R packages, the `dim()` function shows the dimmensions of it, and the `str()` function gives a glimpse of its contents:

```
# Loading the dataset and taking a look
# at the data structure
data(kfamily)
dim(kfamily) # Number of rows and columns
```

```
[1] 1047  432
```

```r
str(kfamily[,1:10]) # structure of the first 10 columns
```

```
'data.frame':   1047 obs. of  10 variables:
 $ village: num  1 1 1 1 1 1 1 1 1 1 ...
 $ id     : num  2 3 4 5 7 8 9 11 12 14 ...
 $ recno1 : int  1 1 1 1 1 1 1 1 1 1 ...
 $ studno1: int  1 1 1 1 1 1 1 1 1 1 ...
 $ area1  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ id1    : int  2 3 4 5 7 8 9 11 12 14 ...
 $ nmage1 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ nmage2 : int  1 0 2 0 0 0 0 0 0 0 ...
 $ nmage3 : int  0 0 0 0 0 1 0 1 1 0 ...
 $ nmage4 : int  0 0 0 0 0 1 0 0 1 1 ...
```

> **Tip**
>
> You can always inspect datasets that are included in an R package using the `data()` function. For example, in netdiffuseR, you can use `data(package="netdiffuseR")`.

As you can see, the dataset contains 1,047 observations with 432 columns. The dataset consists of survey data collected in 1973-1974 from women of childbearing age in 25 rural villages. For this example, we will focus on the two largest villages, namely villages 2 and 21. We can subset the data using the following code:

```r
kfamily <- subset(kfamily, village %in% c(2, 21))
```

The survey has multiple network nomination questions. Depending on the situation, you may want to aggregate them or focus on one of them. We will use the "talks to a neighbor" network nomination question, where participants nominated up to five people. Using that information, we can now use the `survey_to_diffnet` function to create a diffnet object:

```r
# Reading in the data
kfam_diffnet <- survey_to_diffnet(
  dat      = kfamily,
  idvar    = "id",
  netvars  = c("net11", "net12", "net13", "net14", "net15"),
  toavar   = "toa",
  groupvar = "village"
)
```

```
Warning in check_var_class_and_coerce(x, dat, c("numeric", "integer"),
"integer", : Coercing -id- into integer.

Warning in check_var_class_and_coerce(x, dat, c("numeric", "integer"),
"integer", : Coercing -village- into integer.

Warning in check_var_class_and_coerce(x, dat, c("numeric", "integer"),
"integer", : Coercing -toa- into integer.
```

The key things to notice from this code are: (1) We have to specify the "id" of each individual in the dataset via the `idvar` argument; (2) using the `netvars` argument, you can list all the network nomination columns in your data; (3) the `toavar` argument is the name of the variable that contains the time of adoption (year in this case); finally (4) since we have multiple communities in this dataset, we use the `groupvar` argument to tell the function there cannot be ties between villages, only within. The function has many more options and readers can learn about them in the manual `?survey_to_diffnet`.

Another thing to note are the warnings. The R programming language (like most do) distinguishes between decimal ("`numeric`") and integer numbers. In this case, the function is throwing a warning about columns beign coerced from numeric into integer types. These are mostly informative, and the user can skipping these with the `suppressWarnings()` function in R. We will see more warnings as we execute R code and will pause at some points to discuss what these mean to the user.

The `diffnet` object we just created has various inspection methods. The print method gives a glimpse of the data, indicating the number of time points, nodes, type of network, and adoption prevalence, among other things. The `summary` method provides information about the adoption dynamics; it shows the number and proportion of adopters at each time point, network density, and Moran's I (Moran 1950) indicating the level of spatial correlation. Finally, the `plot_diffnet()` function visualizes the diffusion process which colors nodes according to their adoption status, non-adopter, new adopter, and continuing adopter. The following code-chunk shows how to call these functions:

```
print(kfam_diffnet) # Using print() is not necessary, but recommended.
```

```
Dynamic network of class -diffnet-
 Name               : Diffusion Network
 Behavior           : Unknown
 # of nodes         : 113 (201, 202, 203, 204, 205, 206, 207, 209, ...)
 # of time periods  : 11 (1 - 11)
 Type               : directed
 Num of behaviors   : 1
```

```
 Final prevalence   : 1.00
 Static attributes  : village, recno1, studno1, area1, id1, nmage1, nmag... (430)
 Dynamic attributes : -
```

```
summary(kfam_diffnet)
```

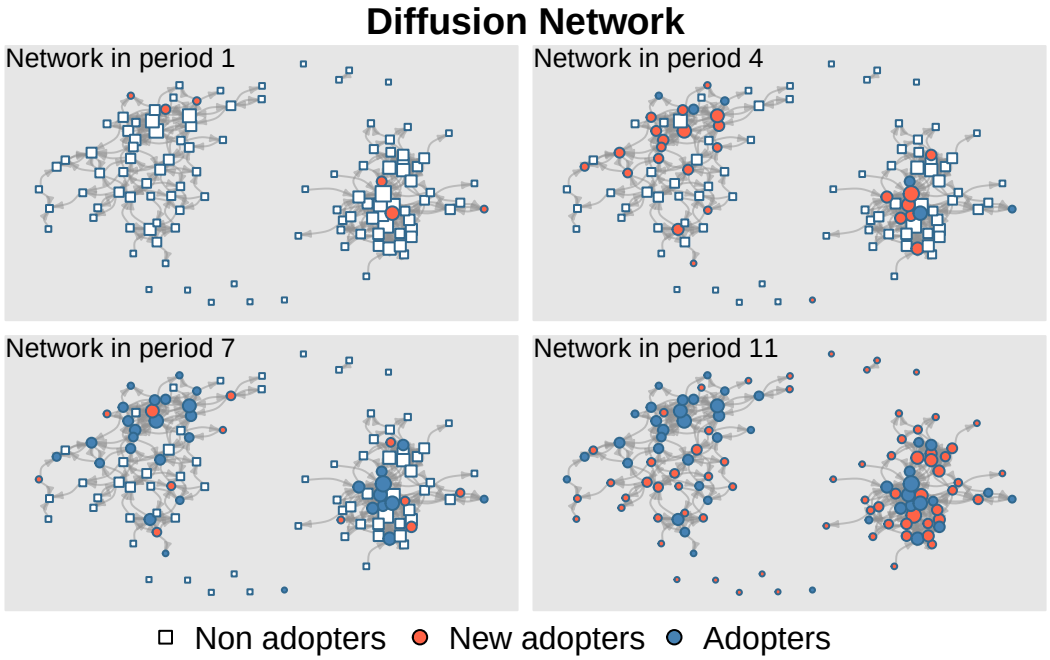
```
Diffusion network summary statistics
 Name     :  Diffusion Network
 Behavior : Unknown
------------------------------------------------------------------------
 Period   Adopters   Cum Adopt. (%)   Hazard Rate   Density   Moran's I (sd)
-------- ---------- ---------------- ------------- --------- ----------------
      1          6         6 (0.05)             -      0.02  0.01 (0.03)
      2         12        18 (0.16)          0.11      0.02  0.01 (0.03)
      3          7        25 (0.22)          0.07      0.02  0.04 (0.03)
      4          7        32 (0.28)          0.08      0.02  0.12 (0.03) ***
      5          5        37 (0.33)          0.06      0.02  0.15 (0.03) ***
      6          3        40 (0.35)          0.04      0.02  0.13 (0.03) ***
      7          4        44 (0.39)          0.05      0.02  0.13 (0.03) ***
      8          5        49 (0.43)          0.07      0.02  0.12 (0.03) ***
      9          7        56 (0.50)          0.11      0.02  0.15 (0.03) ***
     10          6        62 (0.55)          0.11      0.02  0.12 (0.03) ***
     11         51       113 (1.00)          1.00      0.02               -
------------------------------------------------------------------------
 Left censoring  : 0.05 (6)
 Right centoring : 0.00 (0)
 # of nodes      : 113

 Moran's I was computed on contemporaneous autocorrelation using 1/geodesic
 values. Significane levels  *** <= .01, ** <= .05, * <= .1.
```

> **Tip**
>
> The Moran's I statistic is computed with the inverse of the geodesic matrix (shortest path matrix) but can be computed using other statistics using the `moran()` function implemented in netdiffuseR.

In the case of the plot method, since the layout of the network is schocastic (random), we can use the function `set.seed()` to ensure reproducibility:

```
set.seed(33)
plot_diffnet(kfam_diffnet)
```



Of the observed outcomes, it is noteworthy that (a) the diffusion process is apparent in the figure, with adoptions occurring from core to periphery of the graph and (b) Moran's I shows a high-level of spatial autocorrelation between periods four and ten. Other visualization functions include `plot_diffnet2()`, `plot_threshold()`, `plot_adopters()`, and `plot_infectsuscept()`. We invite the reader to learn more about them in the manual of the netidffuseR R package.

Computing exposures

Exposures are a fundamental part of estimating diffusion network effects. Consequently, netdiffuseR provides a wide range of possibilities when it comes to calculating exposures. Using the `exposure()` function, users can calculate cohesion (direct ties) and structural equivalence exposures (Burt 1987), attribute weighted exposure, tie-weighted exposure, count exposure (instead of proportion), and many more. The following code blocks show how to calculate three different types of exposure:

Cohesive exposure. Corresponds to the simplest type of exposure. With it, we measure the proportion of neighbors who have adopted at each time point. Thus, the `exposure()` function will return a matrix of size $n \times$ the number of time points in the data:

```r
exp_cohesive <- exposure(kfam_diffnet)

# Showing the exposure for the first five agents
# and first six time points
exp_cohesive[1:5, 1:6]
```

```
    1 2 3   4   5   6
201 0 0 0 1.0 1.0 1.0
202 0 0 0 0.5 0.5 0.5
203 0 0 0 0.0 0.0 0.0
204 0 0 0 1.0 1.0 1.0
205 0 0 0 0.0 0.0 0.0
```

Structural equivalence. In a nutshell, this type of exposure is often used when influence occurrs not necessarily via direct connections, but rather through similarity of the adopters. In other words, an individual may be more likely to adopt a behavior if others with similar positions to them in the network adopt.

```r
exp_se <- exposure(kfam_diffnet, alt.graph = "se")
```

```
Warning in exposure(kfam_diffnet, alt.graph = "se"): To use alt.graph="se"
-valued- has been switched to TRUE.
```

```
'as(<dgeMatrix>, "dgCMatrix")' is deprecated.
Use 'as(., "CsparseMatrix")' instead.
See help("Deprecated") and help("Matrix-deprecated").
```

```r
# Showing the exposure for the first five agents
# and first six time points
exp_se[1:5, 1:6]
```

```
           1       2       3       4       5       6
201 0.006669 0.01595 0.02043 0.02411 0.03250 0.03614
202 0.006642 0.01583 0.02029 0.02369 0.03207 0.03571
203 0.006915 0.01742 0.02266 0.02829 0.03833 0.04234
204 0.006737 0.01580 0.02035 0.02363 0.03178 0.03547
205 0.007058 0.01807 0.02465 0.03171 0.04161 0.04553
```

The function prints a couple of messages: The first, a warning about the `value` parameter being swithced to `TRUE`, indicates that, since we are using the structural equivalence alternative graph, instead of treating the network as unweighted (0/1-valued ties), the function will compute exposures using weights. The second is a more technical internal message that is describing a change in the `Matrix` R package: users should not worry about this particular message.[1]

Attribute weighted. Finally, this type of exposure can be used when a characteristic of adopters may make them more influential in the social contagion process. For example, people may be more influenced by older individuals than younger:

```
exp_age <- exposure(kfam_diffnet, attrs = "age")

# Showing the exposure for the first five agents
# and first six time points
exp_age[1:5, 1:6]
```

```
    1 2 3      4      5      6
201 0 0 0 1.0000 1.0000 1.0000
202 0 0 0 0.4795 0.4795 0.4795
203 0 0 0 0.0000 0.0000 0.0000
204 0 0 0 1.0000 1.0000 1.0000
205 0 0 0 0.0000 0.0000 0.0000
```

We can look at the three different exposures we just calculated by combining them into a single matrix. The following code chunk takes the fifth column (exposure at the fifth time point) and shows the exposures for the first ten individuals in the network:

```
# Visualizing exposure at time 5
cbind(
  "Cohesion"       = exp_cohesive[, 5],
  "SE"             = exp_se[, 5],
  "Attr. weighted" = exp_age[, 5]
) |> head(n = 10)
```

```
     Cohesion      SE Attr. weighted
201       1.0 0.03250         1.0000
202       0.5 0.03207         0.4795
203       0.0 0.03833         0.0000
204       1.0 0.03178         1.0000
```

[1] To the date of this writing, developers of **netdiffuseR** just acknowledged the issue and are working on fix that will be reflected in the next version of the package.

```
205       0.0 0.04161         0.0000
206       0.5 0.03224         0.5000
207       0.0 0.27304         0.0000
209       0.0 0.27304         0.0000
210       1.0 0.03830         1.0000
2101      0.0 0.28641         0.0000
```

In addition to the exposure function, netdiffuseR comes with other functions that can be used in combination with its core functions. For instance, we can use the `vertex_covariate_compare()` function to calculate age homophilic exposure. Mathematically, we define it as it follows:

$$E_{it} = \frac{\sum_{i \neq j} W_{ij} a_j \|x_i - x_j\|^{-1}}{\sum_{i \neq j} W_{ij} \|x_i - x_j\|^{-1}}$$

Where $\|x_i - x_j\|$ is L-2 norm between i and j's age. The following code block illustrates how we can use `vertex_covariate_compare()` to generate an alternative graph for computing age-homophilic exposure. Since the `graph` argument must be a single adjacency matrix, we need to past the first graph in the `kfam_diffnet` object:

```
# Using homophilic distance
exp_age_homo_graph <- vertex_covariate_compare(
  # Passing the first graph
  graph   = kfam_diffnet$graph[[1]],
  # Passing the age parameter
  X       = kfam_diffnet[["age"]],
  funname = "distance"
)
```

Using `kfam_diffnet$graph[[1]]` we are accessing the first network of the diffnet object. The code `kfam_diffnet[["age"]]` extracts the `age` attribute from the diffnet object. We can take a look at the first few entries of the resulting matrix:

```
exp_age_homo_graph[1:10, 1:10]
```

```
10 x 10 sparse Matrix of class "dgCMatrix"

 [1,] . 4 . 3 . . . . . .
 [2,] 4 . . 1 . . . . . .
 [3,] . . . . . . . . . .
 [4,] . 1 . . . 1 . . . .
```

```
 [5,] . . . . . . . . . .
 [6,] 4 . . 1 . . . . . .
 [7,] . . . . . . . . . .
 [8,] . . . . . . . . . .
 [9,] . . . . . . . . . .
[10,] . . . . . . . . . .
```

Finally, with some manipulation, we can use this new matrix to compute homophilic exposure. As `exp_age_homo_graph` is an object of class `dgCMatrix` from the `Matrix` R package (a sparse matrix), we can access non-zero elements from it using the `@x` operator as it follows:

```
# Inverting the values
exp_age_homo_graph@x <- 1/exp_age_homo_graph@x

exp_age_homo <- exposure(
  kfam_diffnet,
  alt.graph = exp_age_homo_graph,
  valued = TRUE
)
```

```
Warning in exposure(kfam_diffnet, alt.graph = exp_age_homo_graph, valued =
TRUE): When -alt.graph- is static, will be repeated "t" times to fit the data.
```

The warnings in `exposure()` indicate that (a) the passed graph is going to be evaluated as a valued (weighted) network and (b) the same graph is going to be used to compute exposure in all time points (so it is static). The next code chunk gives a glimpse of the different exposure values we have just calculated:

```
# Visualizing exposure at time 5
cbind(
  "Cohesion"       = exp_cohesive[, 5],
  "SE"             = exp_se[, 5],
  "Attr. weighted" = exp_age[, 5],
  "Homophilic"     = exp_age_homo[, 5]
) |> head(10)
```

```
     Cohesion      SE Attr. weighted Homophilic
201       1.0 0.03250         1.0000     1.0000
202       0.5 0.03207         0.4795     0.8000
203       0.0 0.03833         0.0000     0.0000
204       1.0 0.03178         1.0000     1.0000
```

```
205       0.0 0.04161         0.0000     0.0000
206       0.5 0.03224         0.5000     0.5455
207       0.0 0.27304         0.0000     0.0000
209       0.0 0.27304         0.0000     0.0000
210       1.0 0.03830         1.0000     1.0000
2101      0.0 0.28641         0.0000     0.0000
```

In sum, the `exposure` function in the package provides a great deal of flexibility for calculating a wide range of exposures. Computing exposures is a fundamental task in the study of diffusion networks. A more in-depth treatment of network exposures is given in (Valente 2024).

Fitting a lagged exposure model

A natural step after computing exposures is estimating lagged exposure models. The most common approach is fitting a logistic regression that predicts the adoption of a behavior as a function of the nodes' features and lagged exposure. The netdiffuseR package has a number of tools to facilitate this process, but maybe the most useful is the `diffreg()` function. The following code shows how to fit a diffusion model using the default exposure calculation:

```
diffreg(kfam_diffnet ~ exposure) |>
 summary()
```

```
Warning: The variable `per` is not included in the model. This can bias the
`exposure` as adoption naturally increases over time.


Call:
glm(formula = Adopt ~ exposure, family = binomial(link = "logit"),
    data = dat, subset = ifelse(is.na(toa), TRUE, toa >= per))

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -2.170      0.153   -14.2  < 2e-16 ***
exposure       1.012      0.274     3.7  0.00022 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 618.02  on 760  degrees of freedom
Residual deviance: 604.76  on 759  degrees of freedom
```

```
  (113 observations deleted due to missingness)
AIC: 608.8

Number of Fisher Scoring iterations: 4
```

The function is a wrapper of the `glm()` function in R that allows estimating Generalized Linear Models. In our case, `diffreg()` estimates a logistic regression model that predicts adoption as a function of network exposure. Furthermore, as it can be seen in the output, the function automatically truncates the data up to the adoption time (like a survival analysis). Importantly, the function throws a warning message indicating that the period variable, `per`, has not been included in the model, which can lead to bias estimates.

From the output, we can see that the `exposure` term is positively associated with adoption (estimate of 1.012 and a p-value smaller than 0.01), nonetheless, the above example fits an invalid model. To address this issue, we need to make it a *lagged regression model*. In other words, to avoid violating the independent and identically distributed [iid] assumption that regular regression models have, we need to lag the exposure variable. The following code chunk does so adding more control to the `exposure()` term in the model, and more over, leverages the formula syntax to include more features from the Korean family dataset:

```
diffreg(
  kfam_diffnet ~
    # Adding age and a factor for village
    age + factor(village) + factor(per) +
    # Lagged exposure that uses the alternative graph
    exposure(
        lag = 1,
        alt.graph = exp_age_homo_graph,
        valued = TRUE
        )
  ) |>
  summary()
```

```
Call:
glm(formula = Adopt ~ exposure + age + factor(village) + factor(per),
    family = binomial(link = "logit"), data = dat, subset = ifelse(is.na(toa),
        TRUE, toa >= per))

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)        -3.5165     0.7678   -4.58  4.7e-06 ***
```

```
exposure            0.6423     0.4307    1.49    0.136
age                 0.0510     0.0202    2.53    0.011 *
factor(village)21  -0.6836     0.2929   -2.33    0.020 *
factor(per)3       -0.5536     0.5166   -1.07    0.284
factor(per)4       -0.4826     0.5283   -0.91    0.361
factor(per)5       -0.7895     0.5946   -1.33    0.184
factor(per)6       -1.2350     0.6999   -1.76    0.078 .
factor(per)7       -0.9002     0.6425   -1.40    0.161
factor(per)8       -0.6257     0.6108   -1.02    0.306
factor(per)9       -0.1541     0.5662   -0.27    0.785
factor(per)10      -0.2208     0.5932   -0.37    0.710
factor(per)11      20.7869   885.7200    0.02    0.981
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 618.02  on 760  degrees of freedom
Residual deviance: 366.81  on 748  degrees of freedom
  (113 observations deleted due to missingness)
AIC: 392.8

Number of Fisher Scoring iterations: 17
```

As you can see, we can pass arguments to the `exposure()` function on the fly, including using the R's formula interface which is commonly used with the `glm` function in the package. In this fit, we can see that the `exposure` term is no longer statistically significant, suggesting there is no evidence of network diffusion when computed weighted by age homophily. Furthermore, the output suggests that older individuals and individuals not in village 21 are more likely to adopt than younger individuals and individuals in village 21 (estimates of 0.05 and -0.68 respectively). Finally, although necessary, we do not see any significant time effect on the adoption process; only a slightly lower adoption rate at period 6 compared to the others.

More details about what happens when calling the `diffreg()` function can be found in the library's manual.

Simulating diffusion networks

Diffusion processes in networks are simulated using the `rdiffnet()` function, which allows you to simulate diffusion processes by customizing a wide range of options, like graph topology, threshold levels, the proportion of initial adopters, the centrality characteristics of those seeds, and time steps. The simulation algorithm is as follows:

1. If there is no graph data, a baseline graph is created,
2. the initial adopters are identified,
3. a threshold value is imputed for each node,
4. if required, a set of t networks is created, and
5. the new adopters are selected in time $t > 2$ based on the exposure level that the node has in time $t - 1$. If the exposure in that time is greater than its threshold, then the node adopts, otherwise, continues without change.

The function has only two mandatory arguments: the number of nodes (`n`) and the time steps (`t`). By default, the function generates a scale-free graph with random rewiring for each time step, sets the proportion of initial adopters to 0.05, and sets a random uniform threshold value for each node. This example shows the default settings simulating a diffusion process with 200 nodes and 10 time steps:

```
diffnet_1 <- rdiffnet(
  # Mandatory arguments
  n = 200,
  t = 10,
  # Optional arguments
  seed.nodes     = "random",
  seed.p.adopt   = 0.05,
  seed.graph     = "scale-free",
  rgraph.args    = list(),
  rewire         = TRUE,
  rewire.args    = list(),
  threshold.dist = runif(n),
  exposure.args  = list(),
  name           = "A diffusion network",
  behavior       = "Random contagion",
  stop.no.diff   = TRUE,
  disadopt       = NULL
)
```

which is equivalent to calling `rdiffnet(200, 10)`:

```
set.seed(331)
rdiffnet(200, 10)
```

Since the simulation process is inherently random, we use the function `set.seed()` to ensure reproducibility.

> **Tip**
>
> Although the threshold model is a deterministic model, since `rdiffnet()` calls the function `runif` to generate the threshold values, the simulation process is stochastic. If we were using a fixed threshold value, the simulation would be deterministic.

Currently, `rdiffnet()` supports thresholds specified as a fixed value for all nodes, a vector with different values for each node, or a function to be called for each node. The proportion of early adopters (`seed.p.adopt`) is used in combination with the seed nodes (`seed.nodes`) parameters. Current supported values are `"random"`, `"central"`, or `"marginal"`; or, if passed as a vector, `rdiffnet()` ignores `seed.p.adopt` and sets the initial adopters to be those nodes included in the `seed.nodes` vector.

The `rdiffnet()` function also supports the use of a pre-defined graph as input. The following code shows how to simulate a diffusion process using a Watts-Strogatz graph:

```
# Creating a graph based on watts-strogatz model
set.seed(121)
graph <- rgraph_ws(200, 10, p=.3)

# Run the simulation
diffnet_2 <- rdiffnet(
  t              = 10,
  seed.graph     = graph,
  seed.p.adopt   = 0.1,
  threshold.dist = runif(200, .3,.5)
)
```

Since we are passing an actual network to the `seed.graph` argument, the `n` argument is no longer needed.

Multiadoption

Starting with version 1.23.0, **netdiffuseR** supports simulating multi- and dis-adoption diffusion processes. To study a multi-adoption process, you can pass a `list` as the `seed.p.adopt` parameter. Here is a simple example:

```
set.seed(1231)

diffnet_3 <- rdiffnet(
 200, 10,
```

```
 seed.p.adopt = list(0.1, 0.15)
 )
```

```
Message: Multi-diffusion behavior simulation selected. Number of behaviors:  2
```

```
Message: Object -seed.nodes- converted to a -list-.All behaviors will have the same -random-
```

```
Message: Name of 1 behavior provided, but 2 are needed. Names generalized to 'behavior'_1, 'b
```

```
Warning in new_diffnet(graph = sgraph, toa = toa, self = isself, t0 = 1, :
Coercing -toa- into integer.
```

```
diffnet_3
```

```
Dynamic network of class -diffnet-
 Name               : A diffusion network
 Behavior           : Random contagion_1, Random contagion_2
 # of nodes         : 200 (1, 2, 3, 4, 5, 6, 7, 8, ...)
 # of time periods  : 10 (1 - 10)
 Type               : directed
 Num of behaviors   : 2
 Prevalence         : 0.29, 0.97
 Static attributes  : real_threshold.1, real_threshold.2 (2)
 Dynamic attributes : -
```

Inspecting the output from the print method of the `diffnet` object, we can see that the object contains two behaviors: The "Num of behaviors" entry now shows `2`, the "Behavior" entry also shows two behaviors, `"Random contagion \_1, Random contagion\_2"`, and finally, the "Prevalence" entry also shows two numbers: `0.29, 0.97`. Although we are simulating two behaviors, `rdiffnet` will simulate as many as values are in the `seed.p.adopt` list.

> **Tip**
>
> In the current implementation of `rdiffnet()`, the multi-adoption module simulates behaviors independently. That is, the code above would be equivalent to simulating the same behavior twice. More complicated models in which behaviors are interdependent are supported via the dis-adoption parameter.

`rdiffnet()`'s defaults will replicate the simulation parameters across behaviors. Nonetheless, we can use lists to specify different parameters for each behavior. For example, the following

code simulates two behaviors with different initial adopters, threshold distributions, seed nodes, and labels for the behaviors:

```
diffnet_4 <- rdiffnet(
  200, 10,
  seed.p.adopt = list(0.1, 0.15),
  threshold.dist = list(
    runif(200, .3, .5),
    runif(200, .2, .4)
  ),
  seed.nodes = list("central", "random"),
  behavior   = list("tobacco", "alcohol")
)
```

```
Message: Multi-diffusion behavior simulation selected. Number of behaviors:  2
```

```
Warning in new_diffnet(graph = sgraph, toa = toa, self = isself, t0 = 1, :
Coercing -toa- into integer.
```

```
diffnet_4
```

```
Dynamic network of class -diffnet-
 Name               : A diffusion network
 Behavior           : tobacco, alcohol
 # of nodes         : 200 (1, 2, 3, 4, 5, 6, 7, 8, ...)
 # of time periods  : 10 (1 - 10)
 Type               : directed
 Num of behaviors   : 2
 Prevalence         : 1.00, 0.70
 Static attributes  : real_threshold.1, real_threshold.2 (2)
 Dynamic attributes : -
```

In this particular example, we ran `rdiffnet()` with most of the parameters being in a `list`. The reader is invited to look at other types of possible inputs in the `rdiffnet()` documentation.

As we did in the previous section, we can give a specific network as input. Besides passing fixed networks as we did with the small-world example, the `rdiffnet()` function also supports passing `diffnet` objects as input. When doing so, the function will use the graph of the `diffnet` object as the seed graph and will take the time argument as the number of timepoints included in the graph. Here is a simple example using the Koren Family Planning network:

```r
diffnet_5 <- rdiffnet(
  seed.graph     = kfam_diffnet,
  seed.p.adopt   = list(0.1, 0.15),
  threshold.dist = runif(nvertices(kfam_diffnet), .3,.5)
  )
```

```
Message: Multi-diffusion behavior simulation selected. Number of behaviors:  2
```

```
Message: Object -seed.nodes- converted to a -list-.All behaviors will have the same -random-
```

```
Message: Name of 1 behavior provided, but 2 are needed. Names generalized to 'behavior'_1, 'b
```

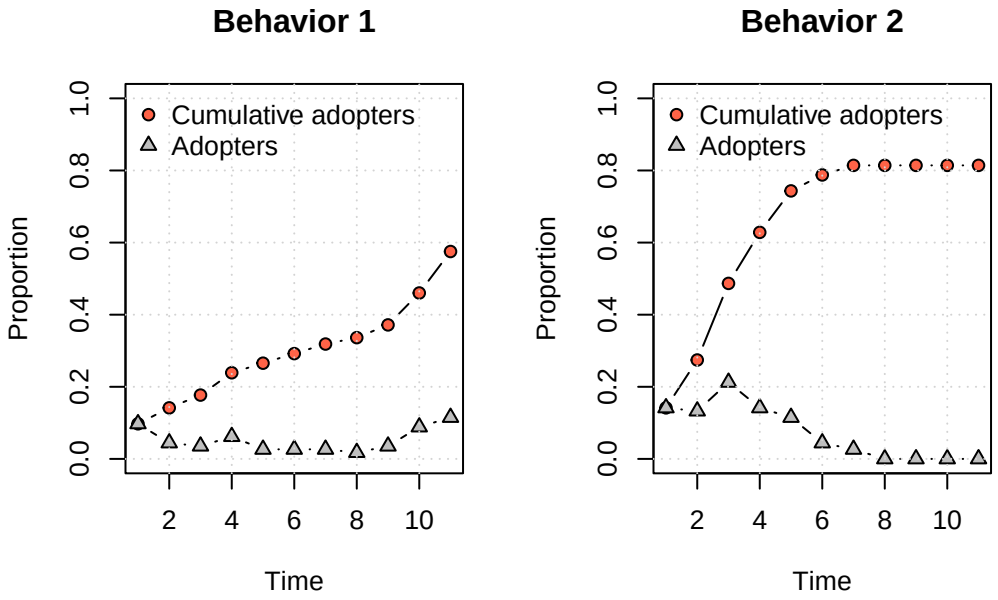
```
Warning in new_diffnet(graph = sgraph, toa = toa, self = isself, t0 = 1, :
Coercing -toa- into integer.
```

```r
diffnet_5
```

```
Dynamic network of class -diffnet-
 Name               : A diffusion network
 Behavior           : Random contagion_1, Random contagion_2
 # of nodes         : 113 (1, 2, 3, 4, 5, 6, 7, 8, ...)
 # of time periods  : 11 (1 - 11)
 Type               : directed
 Num of behaviors   : 2
 Prevalence         : 0.58, 0.81
 Static attributes  : real_threshold.1, real_threshold.2 (2)
 Dynamic attributes : -
```

To visualize the diffusion process when there's more than one behavior, we can use the `split_behaviors()` function to split the diffnet object into a list of diffnet objects, one for each behavior. Then, we can use the `plot_adopters()` function to visualize the diffusion process for each behavior; moreover, using the `par()` function in R, we can arrange both plots in a single window:

```r
diffnets_5 <- split_behaviors(diffnet_5)
op <- par(mfrow=c(1,2), cex = .8)
plot_adopters(diffnets_5[[1]], main = "Behavior 1")
plot_adopters(diffnets_5[[2]], main = "Behavior 2")
```




```
par(op)
```

Notice the `op` object that stores the original graphical parameters. We use it to reset the graphical parameters after plotting the two behaviors.

Disadoption

The disadoption feature included in **netdiffuseR** version 1.23.0 opened a new way of studying network diffusion processes. Considering the disadoption of an innovation or behavior is essential for studying significant aspects of competition between products or beliefs (Lehmann and Parker 2017). The `rdiffnet()` function includes the `disadopt` parameter to add a disadoption function, facilitating such analyses and enabling the testing of some theoretical models for disadoption (Alipour et al. 2024).

1. It must be a function that receives three arguments: `expo`, `cumadopt`, and `time`.
2. It must return a list with as many vectors as behaviors there are: It should list which nodes are dropping behaviors. For instance, if only node `5` (five) is dropping behavior one of three, it should return `list(c(5), integer(), integer())`.
3. If there are no nodes to disadopt or adopt none of n behaviors, the function must return a list with empty vectors: `replicate(n, integer())`.

A template for a disadoption function, which currently returns no disadoption, follows:

```
disadoption_function <- function(expo, cumadopt, time) {
  list(integer(), integer())
}
```

The following code shows how to build a disadoption function that randomly selects 10% of the adopters at time `t - 1`:

```
random_dis <- function(expo, cumadopt, time) {

  # Number of behaviors
  num_of_behaviors <- dim(cumadopt)[3]

  # Making room for the disadopted nodes
  list_disadopt <- list(integer(), integer())

  # We iterate through the behaviors
  for (q in 1:num_of_behaviors) {

    # Identifying the adopters at time t-1
    adopters_old <- which(cumadopt[, time - 1, q] == 1)

    if (length(adopters_old) != 0) {

      # selecting 10% of adopters to disadopt
      list_disadopt[[q]] <- sample(
        adopters_old,
        ceiling(0.10 * length(adopters_old)
        )
      )

    }

  }

  return(list_disadopt)

}
```

It is worth highlighting a few things from the code:

1. The `expo` argument is the entire exposure *array*. This means that it has three dimensions: the first dimension is the number of nodes, the second is the number of time points, and the third is the number of behaviors.
2. The `cumadopt` argument is the cumulative adoption array. It has the same dimensions as `expo`. The value of `cumadopt[i, t, q]` is 1 if node `i` has adopted behavior `q` at time `t`.
3. The `time` argument is the current time point in the simulation. This allows the function to know when the disadopt function is being called.
4. The code `which(cumadopt[, time - 1, q, drop=FALSE] == 1)` identifies which nodes had the entry `cumadopt` equal to 1 at time `t - 1`.

To simulate a diffusion process with disadoption, we can use the `rdiffnet()` function as follows:

```
diffnet_6  <- rdiffnet(
  seed.graph   = graph,
  t            = 10,
  disadopt     = random_dis,
  seed.p.adopt = 0.1
  )

diffnet_6
```

```
Dynamic network of class -diffnet-
 Name                : A diffusion network
 Behavior            : Random contagion
 # of nodes          : 200 (1, 2, 3, 4, 5, 6, 7, 8, ...)
 # of time periods   : 10 (1 - 10)
 Type                : directed
 Num of behaviors    : 1
 Final prevalence    : 0.47
 Static attributes   : real_threshold (1)
 Dynamic attributes : -
```

Using the `disadopt` function, we can build more complex models featuring competing behaviors. For instance, we can build a disadoption function that restricts nodes from adopting more than one behavior at a time, particularly, we can implement the following rule for adopters of behavior 1:

$$\text{Disadopt } 1_{it} = \begin{cases} Yes, & \text{If behavior 2 has adopted} \\ \text{No}, & \text{otherwise.} \end{cases}$$

The following code shows how to build such a function:

```
one_only <- function(expo, cumadopt, time) {

  # Id double adopters
  ids <- which(apply(cumadopt[, time,], 1, sum) == 2)

  if (length(ids) == 0)
    return(list(integer(), integer()))

  # Otherwise, make them pick one (in this case, we prefer the second)
  return(list(ids, integer()))

}
```

Of the code above, we can highlight the following:

1. We are identifying the number of behaviors adopted by each individual at time `t` using the code `apply(cumadopt[, time,], 1, sum)`. In a two behavior model, this will return a vector with values 0 (no adoption), 1 (only one behavior adopted), or 2 (both behaviors adopted).
2. The `which()` function is used to identify the nodes adopting both behaviors (after calling `apply()`).
3. If there are no double adopters, the function returns a list with empty vectors.

Let's simulate a diffusion process with the disadoption function `one_only`:

```
set.seed(331)
diffnet_7 <- rdiffnet(
  200, 10,
  disadopt = one_only,
  seed.p.adopt = list(0.1, 0.1)
)
```

```
Message: Multi-diffusion behavior simulation selected. Number of behaviors:  2

Message: Object -seed.nodes- converted to a -list-.All behaviors will have the same -random-

Message: Name of 1 behavior provided, but 2 are needed. Names generalized to 'behavior'_1, 'b
```

```
Warning in new_diffnet(graph = sgraph, toa = toa, self = isself, t0 = 1, :
Coercing -toa- into integer.
```

```
diffnet_7
```

```
Dynamic network of class -diffnet-
 Name               : A diffusion network
 Behavior           : Random contagion_1, Random contagion_2
 # of nodes         : 200 (1, 2, 3, 4, 5, 6, 7, 8, ...)
 # of time periods  : 10 (1 - 10)
 Type               : directed
 Num of behaviors   : 2
 Prevalence         : 0.28, 0.42
 Static attributes  : real_threshold.1, real_threshold.2 (2)
 Dynamic attributes : -
```

To finalize, we can demonstrate that nodes adopted a single behavior by taking the cumulative adoption matrix at the last time point and checking if there are any nodes adopting both behaviors. We will use the `toa_mat()` function which extract the cumulative adoption matrix from the model:

```
#
toas <- toa_mat(diffnet_7)

# Putting the two behaviors together
adopted <-  cbind(
  toas[[1]]$cumadopt[, 10],
  toas[[2]]$cumadopt[, 10]
)

# Looking at the first 5
head(adopted, 5)
```

```
  [,1] [,2]
1    0    0
2    0    1
3    1    0
4    1    0
5    0    1
```

```
# Tabulating side by side
table(adopted[, 1], adopted[, 2])
```

```
    0  1
0  61 84
1  55  0
```

As expected, there's no entry in the table in which both behaviors were adopted by the same node. Using the `disadopt` function, we can build (and study) increasingly complex models of network diffusion.

Conclusion

This paper has provided an introduction to the netdiffuseR R package designed to facilitate estimating contagion and diffusion processes on networks. The main goal was to provide example code and tutorial so others may use the package to explore diffusion network models either with the existing classic datasets that come with the package or using their own data. The main tutorial sections included sample code for estimating network exposure, the workhorse variable in network research. Simulation features of netdiffuseR were illustrated which provide tools for theoretical testing of diffusion processes. We also provided examples of extensions to exposure that provide a wide array of different weighting techniques, such as structural equivalence, homophilic ties, and so on.

We also provide illustrative examples of modeling multi-diffusion processes when there is more than one idea or behavior spreading through the network. For example, the Korean family planning contains data on multiple family planning choices individuals made over time. We also have example code for estimating models that include disadoption as when someone adopts a behavior and then subsequently discontinues it and then may resume at a later date, and so on. At present there are few datasets, if any, that meet these scenarios, but it is hoped that as scholars are exposed to this work they uncover such datasets.

We hope that by providing these illustrative examples, others will be able to use netdiffuseR in their work. Diffusion processes are ubiquitous in science and everyday life. The diffusion paradigm was first established by Ryan and Gross in 1943 (Valente and Rogers 1995) but it continues to grow and evolve, and our hope is that this publication will further encourage both growth and evolution.

Acknowledgements

This work was supported by the National Institute of Drug Abuse, the National Institutes of Health (NIH), grant number R01-DA051843. We would also like to thank an anonymous reviewer for their comments and suggestions that improved this paper.

References

Alipour, Fahimeh, Fedor Dokshin, Zeinab Maleki, Yunya Song, and Pouria Ramazi. 2024. "Enough but Not Too Many: A Bi-Threshold Model for Behavioral Diffusion." Edited by Javier Borge-Holthoefer. *PNAS Nexus* 3 (10): pgae428. https://doi.org/10.1093/pnasnexus/pgae428.

Allaire, J. J., Charles Teague, Carlos Scheidegger, Yihui Xie, Christophe Dervieux, and Gordon Woodhull. 2025. "Quarto." https://doi.org/10.5281/zenodo.5960048.

Aral, Sinan, and Christos Nicolaides. 2017. "Exercise Contagion in a Global Social Network." *Nature Communications* 8 (1): 14753. https://doi.org/10.1038/ncomms14753.

Bass, Frank M. 1969. "A New Product Growth for Model Consumer Durables." *Management Science* 15 (5): 215–27. https://doi.org/10.1287/mnsc.15.5.215.

Berry, George, Christopher J. Cameron, Patrick Park, and Michael Macy. 2019. "The Opacity Problem in Social Contagion." *Social Networks* 56 (January): 93–101. https://doi.org/10.1016/j.socnet.2018.09.001.

Bongaarts, John, and Susan Cotts Watkins. 1996. "Social Interactions and Contemporary Fertility Transitions." *Population and Development Review* 22 (4): 639. https://doi.org/10.2307/2137804.

Burt, Ronald S. 1987. "Social Contagion and Innovation: Cohesion Versus Structural Equivalence." *American Journal of Sociology* 92 (6): 1287–1335. https://doi.org/10.1086/228667.

Christakis, Nicholas A., and James H. Fowler. 2007. "The Spread of Obesity in a Large Social Network over 32 Years." *New England Journal of Medicine* 357 (4): 370–79. https://doi.org/10.1056/NEJMsa066082.

Coleman, James Samuel, Elihu Katz, and Herbert Menzel. 1966. *Medical Innovation; a Diffusion Study / [by] James s. Coleman, Elihu Katz [and] Herbert Menzel. Foreword by Joseph a. Precker.* Indianapolis: Bobbs-Merrill Co.

Fujimoto, Kayo, and Thomas W. Valente. 2013. "Alcohol Peer Influence of Participating in Organized School Activities: A Network Approach." *Health Psychology* 32 (10): 1084–92. https://doi.org/10.1037/a0029466.

Hoffman, Beth R., Steve Sussman, Jennifer B. Unger, and Thomas W. Valente. 2006. "Peer Influences on Adolescent Cigarette Smoking: A Theoretical Review of the Literature." *Substance Use & Misuse* 41 (1): 103–55. https://doi.org/10.1080/10826080500368892.

Iyengar, Raghuram, Christophe Van Den Bulte, and Thomas W. Valente. 2011. "Opinion Leadership and Social Contagion in New Product Diffusion." *Marketing Science* 30 (2): 195–212. https://doi.org/10.1287/mksc.1100.0566.

Katz, Elihu. 1957. "The Two-Step Flow of Communication: An Up-To-Date Report on an Hypothesis." *Public Opinion Quarterly* 21 (1, Anniversary Issue Devoted to Twenty Years of Public Opinion Research): 61. https://doi.org/10.1086/266687.

Lehmann, Donald R., and Jeffrey R. Parker. 2017. "Disadoption." *AMS Review* 7 (1-2): 36–51. https://doi.org/10.1007/s13162-017-0093-8.

Light, John M., Charlotte C. Greenan, Julie C. Rusby, Kimberley M. Nies, and Tom A. B. Snijders. 2013. "Onset to First Alcohol Use in Early Adolescence: A Network Diffusion Model." *Journal of Research on Adolescence* 23 (3): 487–99. https://doi.org/10.1111/jora.12064.

Moran, P. A. P. 1950. "Notes on Continuous Stochastic Phenomena." *Biometrika* 37 (1/2): 17. https://doi.org/10.2307/2332142.

R Core Team. 2025. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rogers, Everett M. 2003. *Diffusion of Innovations*. Fifth edition. New York London Toronto Sydney: Free Press.

Rogers, Everett M., and D. Lawrence Kincaid. 1981. *Communication Networks: Toward a New Paradigm for Research*. New York : London: Free Press ; Collier Macmillan.

Valente, Thomas W. 1993. "Diffusion of Innovations and Policy Decision-Making." *Journal of Communication* 43 (1): 30–45. https://doi.org/10.1111/j.1460-2466.1993.tb01247.x.

———. 1995. *Network Models of the Diffusion of Innovations*. 2nd ed. Hampton Press.

———. 1996. "Social Network Thresholds in the Diffusion of Innovations." *Social Networks* 18 (1): 69–89. https://doi.org/10.1016/0378-8733(95)00256-1.

———. 2010. *Social Networks and Health: Models, Methods, and Applications*. 1st ed. Oxford University PressNew York. https://doi.org/10.1093/acprof:oso/9780195301014.001.0001.

———. 2024. "Contagion and Interpersonal Influence: Distinguishing Mechanisms of Behavior Change Using Social Network Theory." *Connections* 0 (0): –. https://doi.org/doi:10.21307/connections-2019.041.

Valente, Thomas W., Stephanie R. Dyal, Kar-Hai Chu, Heather Wipfli, and Kayo Fujimoto. 2015. "Diffusion of Innovations Theory Applied to Global Tobacco Control Treaty Ratification." *Social Science & Medicine* 145 (November): 89–97. https://doi.org/10.1016/j.socscimed.2015.10.001.

Valente, Thomas W., and Everett M. Rogers. 1995. "The Origins and Development of the Diffusion of Innovations Paradigm as an Example of Scientific Growth." *Science Communication* 16 (3): 242–73. https://doi.org/10.1177/1075547095016003002.

Valente, Thomas W., and Walter P. Saba. 1998. "Mass Media and Interpersonal Influence in a Reproductive Health Communication Campaign in Bolivia." *Communication Research* 25 (1): 96–124. https://doi.org/10.1177/009365098025001004.

Valente, Thomas W., and George G. Vega Yon. 2020. "Diffusion/Contagion Processes on Social Networks." *Health Education & Behavior* 47 (2): 235–48. https://doi.org/10.1177/1090198120901497.

Valente, Thomas W., Susan C. Watkins, Miriam N. Jato, Ariane Van Der Straten, and Louis-Philippe M. Tsitsol. 1997. "Social Network Associations with Contraceptive Use Among Cameroonian Women in Voluntary Associations." *Social Science & Medicine* 45 (5): 677–

87. https://doi.org/10.1016/S0277-9536(96)00385-1.

Vega Yon, George, Anibal Olivera Morales, and Thomas Valente. 2025. *netdiffuseR: Analysis of Diffusion and Contagion Processes on Networks*. https://doi.org/10.5281/zenodo.1039317.

Yamagata, Yoshiki, Jue Yang, and Joseph Galaskiewicz. 2017. "State power and diffusion processes in the ratification of global environmental treaties, 1981–2008." *International Environmental Agreements: Politics, Law and Economics* 17 (4): 501–29. https://doi.org/10.1007/s10784-016-9332-y.