

# Package ‘PAFit’

September 7, 2025

**Type** Package

**Title** Generative Mechanism Estimation in Temporal Complex Networks

**Version** 1.2.11

**Date** 2025-09-05

**Maintainer** Thong Pham <thongphamthe@gmail.com>

**Description** Statistical methods for estimating preferential attachment and node fitness generative mechanisms in temporal complex networks are provided. Thong Pham et al. (2015) <[doi:10.1371/journal.pone.0137796](https://doi.org/10.1371/journal.pone.0137796)>. Thong Pham et al. (2016) <[doi:10.1038/srep32558](https://doi.org/10.1038/srep32558)>

**URL** <https://github.com/thongphamthe/PAFit>

**BugReports** <https://github.com/thongphamthe/PAFit/issues>

**License** GPL-3

**Depends** R(>= 2.10.0)

**Imports** Rcpp (>= 0.11.3) , grDevices, graphics, stats, RColorBrewer, VGAM, MASS, magicaxis, networkDynamic, network, plyr, igraph, mapproj, knitr, methods, ggplot2

**LinkingTo** Rcpp

**LazyData** True

**Encoding** UTF-8

**NeedsCompilation** yes

**Suggests** R.rsp

**VignetteBuilder** R.rsp

**Author** Thong Pham [aut, cre],  
Paul Sheridan [aut],  
Hidetoshi Shimodaira [aut]

**Repository** CRAN

**Date/Publication** 2025-09-07 17:30:31 UTC

## Contents

PAFit-package . . . . .	3
as.PAFit_net . . . . .	5
Coauthorship network of scientists in the field of complex networks . . . . .	6
from_igraph . . . . .	7
from_networkDynamic . . . . .	8
generate_BA . . . . .	8
generate_BB . . . . .	10
generate_ER . . . . .	11
generate_fit_only . . . . .	13
generate_net . . . . .	14
generate_simulated_data_from_estimated_model . . . . .	17
get_statistics . . . . .	19
graph_from_file . . . . .	22
graph_to_file . . . . .	23
Jeong . . . . .	24
joint_estimate . . . . .	26
Newman . . . . .	30
only_A_estimate . . . . .	31
only_F_estimate . . . . .	35
PAFit_oneshot . . . . .	37
plot.Full_PAFit_result . . . . .	38
plot.PAFit_net . . . . .	40
plot.PAFit_result . . . . .	41
plot.PA_result . . . . .	43
plot_contribution . . . . .	45
print.CV_Data . . . . .	47
print.CV_Result . . . . .	48
print.Full_PAFit_result . . . . .	49
print.PAFit_data . . . . .	50
print.PAFit_net . . . . .	51
print.PAFit_result . . . . .	51
print.PA_result . . . . .	52
summary.CV_Data . . . . .	53
summary.CV_Result . . . . .	54
summary.Full_PAFit_result . . . . .	55
summary.PAFit_data . . . . .	56
summary.PAFit_net . . . . .	57
summary.PAFit_result . . . . .	58
summary.PA_result . . . . .	59
test_linear_PA . . . . .	59
to_igraph . . . . .	61
to_networkDynamic . . . . .	62

## Description

A package for estimating preferential attachment and node fitness generative mechanisms in temporal complex networks. References: Thong Pham et al. (2015) <10.1371/journal.pone.0137796>, Thong Pham et al. (2016) <doi:10.1038/srep32558>, Thong Pham et al. (2020) <doi:10.18637/jss.v092.i03>, Thong Pham et al. (2021) <doi:10.1093/comnet/cnab024>.

## Details

Package:	PAFit
Type:	Package
Version:	1.2.11
Authors:	Thong Pham, Paul Sheridan, Hidetoshi Shimodaira
Maintainer:	Thong Pham <thongphamthe@gmail.com>
Date:	2025-09-05
License:	GPL-3

The PAFit package provides a comprehensive framework to deal with growth mechanisms of temporal complex networks. In particular, it implements functions to simulate various temporal network models, gather essential network statistics from raw input data, and use these summarized statistics in the estimation of the attachment function  $A_k$  and node fitnesses  $\eta_i$ . The heavy computational parts of the package are implemented in C++ through the use of the Rcpp package. Furthermore, users with a multi-core machine can enjoy a hassle-free speed up through OpenMP parallelization mechanisms implemented in the code. Apart from the main functions, the package also includes a real-world collaboration network dataset between scientists in the field of complex networks ([coauthor.net](https://coauthor.net)). The main package functionalities are as follows.

Firstly, most well-known temporal network models based on the preferential attachment (PA) and node fitness mechanisms can be easily simulated using the package. PAFit implements `generate_BA` for the Barabási-Albert (BA) model, `generate_ER` for the growing Erdős-Rényi (ER) model, `generate_BB` for the Bianconi-Barabási (BB) model and `generate_fit_only` for the Caldarelli model. These functions have many customizable options, for example the number of new edges at each time-step are tunable stochastic variables. They are actually wrappers of the more powerful `generate_net` function, which simulates networks with more flexible attachment function and node fitness settings.

Secondly, the function `get_statistics` efficiently collects all temporal network summary statistics. We note that `get_statistics` automatically handles both directed and undirected networks. It returns a list containing many statistics that can be used to characterize the network growth process. Notable fields are `mTk` containing the number of new edges that connect to a degree- $k$  node at time-step  $t$ , and `node_degree` containing the degree sequence, i.e., the degree of each node at each time-step.

The most important functionality of the package is estimating the attachment function and node fitnesses of a temporal network. This is implemented through various methods. There are three usages: estimation of the attachment function in isolation, estimation of the node fitnesses in isolation, and the joint estimation of the attachment function and node fitnesses.

- The functions for estimating the attachment function in isolation are: `Jeong` for Jeong’s method (Ref. 1), `Newman` for Newman’s method (Ref. 2), and `only_A_estimate` for the PAFit method (Ref. 3).
- For estimation of node fitnesses in isolation, `only_F_estimate` implements a variant of the PAFit method (Ref. 4).
- For the joint estimation of the attachment function and node fitnesses, we implement the full version of the PAFit method in `joint_estimate` (Ref. 4).
- For estimating the nonparametric attachment function from a single snapshot, use `PAFit_oneshot` (Ref. 6).

Excluding `PAFit_oneshot`, the input of the remaining functions is the output object of the function `get_statistics`. The output object of these functions contains the estimation results as well as some additional information pertaining to the estimation process. The estimated attachment function and/or node fitnesses can be plotted by using the `plot` command directly on this output object. This will visualize not only the estimated results but also the remaining uncertainties when possible.

#### Author(s)

Thong Pham <thongphamthe@gmail.com>, Paul Sheridan, and Hidetoshi Shimodaira.

#### References

1. Jeong, H., Néda, Z. & Barabási, A. (2003). Measuring Preferential Attachment in Evolving Networks. *Europhysics Letters* 61(61):567-572. (doi:10.1209/epl/i2003001669).
2. Newman, M. (2001). Clustering and Preferential Attachment in Growing Networks. *Physical Review E* 64(2):025102. (doi:10.1103/PhysRevE.64.025102).
3. Pham, T., Sheridan, P. & Shimodaira, H. (2015). PAFit: A Statistical Method for Measuring Preferential Attachment in Temporal Complex Networks. *PLOS ONE* 10(9):e0137796. (doi:10.1371/journal.pone.0137796).
4. Pham, T., Sheridan, P. & Shimodaira, H. (2016). Joint Estimation of Preferential Attachment and Node Fitness in Growing Complex Networks. *Scientific Reports* 6, Article number: 32558. (doi:10.1038/srep32558).
5. Pham, T., Sheridan, P. & Shimodaira, H. (2020). PAFit: An R Package for the Non-Parametric Estimation of Preferential Attachment and Node Fitness in Temporal Complex Networks. *Journal of Statistical Software* 92 (3). (doi:10.18637/jss.v092.i03)
6. Pham, T., Sheridan, P. & Shimodaira, H. (2021). Non-parametric estimation of the preferential attachment function from one network snapshot. *Journal of Complex Networks* 9(5): cnab024. (doi:10.1093/comnet/cnab024).

#### See Also

See the accompanying vignette for a tutorial.

See also the [GitHub page](#).

**Examples**

```
## Not run:
### Jointly estimate the attachment function and node fitnesses
library("PAFit")
set.seed(1)
# a Bianconi-Barabasi network
# size of initial network = 100
# number of new nodes at each time-step = 100
# Ak = k; inverse variance of distribution of fitness: s = 10
net      <- generate_BB(N      = 1000 , m      = 10 ,
                       num_seed = 100 , multiple_node = 100,
                       s      = 10)
net_stats <- get_statistics(net)

#Joint estimation of attachment function Ak and node fitness
result    <- joint_estimate(net, net_stats)

summary(result)

# plot the estimated attachment function
plot(result, net_stats)

# true function
true_A    <- pmax(result$estimate_result$center_k,1)
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")
#plot distribution of estimated node fitnesses
plot(result, net_stats, plot = "f")

#plot the estimated node fitnesses and true node fitnesses
plot(result, net_stats, true = net$fitness, plot = "true_f")

## End(Not run)
```

as.PAFit\_net

*Converting an edgelist matrix to a PAFit\_net object***Description**

This function converts a graph stored in an edgelist matrix format to a PAFit\_net object.

**Usage**

```
as.PAFit_net(graph, type = "directed", PA = NULL, fitness = NULL)
```

**Arguments**

**graph** An edgelist matrix. Each row is assumed to be of the form (from\_node\_id to\_node\_id time\_stamp). For a directed network ,from\_node\_id is the id of

the source node and `to_node_id` is the id of the destination node. For an undirected network, the order is ignored and `from_node_id` and `to_node_id` are the ids of two ends. `time_stamp` is the arrival time of the edge. `from_node_id` and `to_node_id` are assumed to be integers that are at least 0. The whole ids need not to be contiguous.

To register a new node  $i$  at time  $t$  without any edge, add a row with format  $(i - 1 \ t)$ . This works for both undirected and directed networks.

`time_stamp` can be either numeric or string. The value of a time-stamp can be arbitrary, but we assume that a smaller `time_stamp` (regarded so by the sort function in R) represents an earlier arrival time. Examples of time-stamps that satisfy this assumption are the integer  $0:T$ , the string format 'yyyy-mm-dd', and the POSIX time.

<code>type</code>	String. Indicates whether the network is "directed" or "undirected".
<code>PA</code>	Numeric vector. Contains the PA function. Default value is NULL.
<code>fitness</code>	Numeric vector. Contains node fitnesses. Default value is NULL.

### Value

An object of class `PAFit_net`

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### Examples

```
library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50 , m = 10 , s = 10)
as.PAFit_net(net$graph)
```

---

Coauthorship network of scientists in the field of complex networks

*A collaboration network between authors of papers in the field of complex networks with article time-stamps*

---

### Description

The dataset is collaboration network of authors of network science articles with article time-stamps. An edge between two authors represents an article in common. Time stamps denote article publication dates. The network without time-stamps was compiled by Mark Newman in May 2006 from the bibliographies of two review articles on networks, M. E. J. Newman, *SIAM Review* 45, 167-256 (2003) and S. Boccaletti et al., *Physics Reports* 424, 175-308 (2006), with a few additional references added by hand. Paul Sheridan independently supplemented the network with time-stamps and some basic metadata in June 2015. The network is undirected with monthly resolution, and contains no duplicated edges. `coauthor.net` contains the network. `coauthor.true.time` contains the real times of processed time-stamps. Finally `coauthor.author_id` contains author names.

Reference: M. E. J. Newman, Finding community structure in networks using the eigenvectors of matrices, Preprint physics/0605087 (2006).

### Usage

```
data(ComplexNetCoauthor)
```

### Format

`coauthor.net` is a matrix with 2849 rows and 3 columns. Each row is an edge with the format (author id 1, author id 2, time\_stamp). `coauthor.truetime` is a two-column matrix whose each row is (time\_stamp, real time). `coauthor.author_id` is a two-column matrix whose each row is (author id, author name).

### Source

<https://www.paulsheridan.net/files/collabnet.zip>

---

from_igraph	<i>Convert an igraph object to a PAFit_net object</i>
-------------	---

---

### Description

This function converts an igraph object (of package **igraph**) to a PAFit\_net object.

### Usage

```
from_igraph(net)
```

### Arguments

`net` An object of class igraph.

### Value

The function returns a PAFit\_net object.

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### Examples

```
library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50 , m = 10 , s = 10)
igraph_graph <- to_igraph(net)
back <- from_igraph(igraph_graph)
```

---

from\_networkDynamic     *Convert a networkDynamic object to a PAFit\_net object*

---

### Description

This function converts a networkDynamic object (of package **networkDynamic**) to a PAFit\_net object.

### Usage

```
from_networkDynamic(net)
```

### Arguments

net                    An object of class networkDynamic.

### Value

The function returns a PAFit\_net object.

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### Examples

```
library("PAFit")
# a network from Bianconi-Barabasi model
net            <- generate_BB(N = 50 , m = 10 , s = 10)
nD_graph      <- to_networkDynamic(net)
back          <- from_networkDynamic(nD_graph)
```

---

generate\_BA             *Simulating networks from the generalized Barabasi-Albert model*

---

### Description

This function generates networks from the generalized Barabási-Albert model. In this model, the preferential attachment function is power-law, i.e.  $A_k = k^\alpha$ , and node fitnesses are all equal to 1. It is a wrapper of the more powerful function [generate\\_net](#).

### Usage

```
generate_BA(N                    = 1000,
            num_seed            = 2    ,
            multiple_node       = 1    ,
            m                    = 1    ,
            alpha                = 1)
```



**Arguments**

N	Integer. Total number of nodes in the network (including the nodes in the seed graph). Default value is 1000.
num_seed	Integer. The number of nodes of the seed graph (the initial state of the network). The seed graph is a cycle. Default value is 2.
multiple_node	Positive integer. The number of new nodes at each time-step. Default value is 1.
m	Positive integer. The number of edges of each new node. Default value is 1.
alpha	Numeric. This is the attachment exponent in the attachment function $A_k = k^\alpha$ .

**Value**

The output is a PAFit\_net object, which is a List contains the following four fields:

graph	a three-column matrix, where each row contains information of one edge, in the form of (from_id, to_id, time_stamp). from_id is the id of the source, to_id is the id of the destination.
type	a string indicates whether the network is "directed" or "undirected".
PA	a numeric vector contains the true PA function.
fitness	fitness values of nodes in the network. The fitnesses are all equal to 1.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**References**

1. Albert, R. & Barabási, A. (1999). Emergence of scaling in random networks. Science, 286,509–512 (<https://www.science.org/doi/10.1126/science.286.5439.509>).

**See Also**

For subsequent estimation procedures, see [get\\_statistics](#).

For other functions to generate networks, see [generate\\_net](#), [generate\\_ER](#), [generate\\_BB](#) and [generate\\_fit\\_only](#).

**Examples**

```
library("PAFit")
# generate a network from the BA model with alpha = 1, N = 100, m = 1
net <- generate_BA(N = 100)
str(net)
plot(net)
```

---

generate\_BB

*Simulating networks from the Bianconi-Barabasi model*


---

### Description

This function generates networks from the Bianconi-Barabási model. It is a ‘preferential attachment with fitness’ model. In this model, the preferential attachment function is linear, i.e.  $A_k = k$ , and node fitnesses are sampled from some probability distribution.

### Usage

```
generate_BB(N           = 1000  ,
            num_seed    = 2      ,
            multiple_node = 1     ,
            m           = 1     ,
            mode_f      = "gamma",
            s           = 10    )
```

### Arguments

The parameters can be divided into two groups.

The first group specifies basic properties of the network:

N	Integer. Total number of nodes in the network (including the nodes in the seed graph). Default value is 1000.
num_seed	Integer. The number of nodes of the seed graph (the initial state of the network). The seed graph is a cycle. Default value is 2.
multiple_node	Positive integer. The number of new nodes at each time-step. Default value is 1.
m	Positive integer. The number of edges of each new node. Default value is 1.

The final group of parameters specifies the distribution from which node fitnesses are generated:

mode_f	String. Possible values: "gamma", "log_normal" or "power_law". This parameter indicates the true distribution for node fitness. "gamma" = gamma distribution, "log_normal" = log-normal distribution. "power_law" = power-law (pareto) distribution. Default value is "gamma".
s	Non-negative numeric. The inverse variance parameter. The mean of the distribution is kept at 1 and the variance is $1/s$ (since node fitnesses are only meaningful up to scale). This is achieved by setting shape and rate parameters of the Gamma distribution to $s$ ; setting mean and standard deviation in log-scale of the log-normal distribution to $-1/2 * \log(1/s + 1)$ and $(\log(1/s + 1))^{0.5}$ ; and setting shape and scale parameters of the pareto distribution to $(s + 1)^{0.5} + 1$ and $(s + 1)^{0.5} / ((s + 1)^{0.5} + 1)$ . If $s$ is 0, all node fitnesses $\eta$ are fixed at 1 (i.e., Barabási-Albert model). The default value is 10.

**Value**

The output is a PAFit\_net object, which is a List contains the following four fields:

graph	a three-column matrix, where each row contains information of one edge, in the form of (from_id, to_id, time_stamp). from_id is the id of the source, to_id is the id of the destination.
type	a string indicates whether the network is "directed" or "undirected".
PA	a numeric vector contains the true PA function.
fitness	fitness values of nodes in the network. The name of each value is the ID of the node.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**References**

1. Bianconi, G. & Barabási, A. (2001). Competition and multiscaling in evolving networks. Europhys. Lett., 54, 436 (doi:10.1209/epl/i2001002606).

**See Also**

For subsequent estimation procedures, see [get\\_statistics](#).

For other functions to generate networks, see [generate\\_net](#), [generate\\_BA](#), [generate\\_ER](#) and [generate\\_fit\\_only](#).

**Examples**

```
library("PAFit")
# generate a network from the BB model with alpha = 1, N = 100, m = 1
# The inverse variance of the Gamma distribution of node fitnesses is s = 10
net <- generate_BB(N = 100, m = 1, mode = 1, s = 10)
str(net)
plot(net)
```

---

generate\_ER

*Simulating networks from the Erdos-Renyi model*

---

**Description**

This function generates networks from the Erdős–Rényi model. In this model, the preferential attachment function is a constant function, i.e.  $A_k = 1$ , and node fitnesses are all equal to 1. It is a wrapper of the more powerful function [generate\\_net](#).

**Usage**

```
generate_ER(N           = 1000,
            num_seed    = 2    ,
            multiple_node = 1    ,
            m           = 1)
```

**Arguments**

N	Integer. Total number of nodes in the network (including the nodes in the seed graph). Default value is 1000.
num_seed	Integer. The number of nodes of the seed graph (the initial state of the network). The seed graph is a cycle. Default value is 2.
multiple_node	Positive integer. The number of new nodes at each time-step. Default value is 1.
m	Positive integer. The number of edges of each new node. Default value is 1.

**Value**

The output is a PAFit\_net object, which is a List contains the following four fields:

graph	a three-column matrix, where each row contains information of one edge, in the form of (from_id, to_id, time_stamp). from_id is the id of the source, to_id is the id of the destination.
type	a string indicates whether the network is "directed" or "undirected".
PA	a numeric vector contains the true PA function.
fitness	fitness values of nodes in the network. The fitnesses are all equal to 1.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**References**

1. Erdős P. & Rényi A.. On random graphs. Publicationes Mathematicae Debrecen. 1959;6:290–297 (<https://snap.stanford.edu/class/cs224w-readings/erdos59random.pdf>).

**See Also**

For subsequent estimation procedures, see [get\\_statistics](#).

For other functions to generate networks, see [generate\\_net](#), [generate\\_BA](#), [generate\\_BB](#) and [generate\\_fit\\_only](#).

**Examples**

```
library("PAFit")
# generate a network from the ER model with N = 1000 nodes
net <- generate_ER(N = 1000)
str(net)
plot(net)
```

---

generate\_fit\_only      *Simulating networks from the Caldarelli model*

---

## Description

This function generates networks from the Caldarelli model. In this model, the preferential attachment function is constant, i.e.  $A_k = 1$ , and node fitnesses are sampled from some probability distribution.

## Usage

```
generate_fit_only(N           = 1000  ,
                  num_seed    = 2     ,
                  multiple_node = 1    ,
                  m            = 1     ,
                  mode_f       = "gamma",
                  s            = 10    )
```

## Arguments

The parameters can be divided into two groups.

The first group specifies basic properties of the network:

N	Integer. Total number of nodes in the network (including the nodes in the seed graph). Default value is 1000.
num_seed	Integer. The number of nodes of the seed graph (the initial state of the network). The seed graph is a cycle. Default value is 2.
multiple_node	Positive integer. The number of new nodes at each time-step. Default value is 1.
m	Positive integer. The number of edges of each new node. Default value is 1.

The final group of parameters specifies the distribution from which node fitnesses are generated:

mode_f	String. Possible values: "gamma", "log_normal" or "power_law". This parameter indicates the true distribution for node fitness. "gamma" = gamma distribution, "log_normal" = log-normal distribution. "power_law" = power-law (pareto) distribution. Default value is "gamma".
s	Non-negative numeric. The inverse variance parameter. The mean of the distribution is kept at 1 and the variance is $1/s$ (since node fitnesses are only meaningful up to scale). This is achieved by setting shape and rate parameters of the Gamma distribution to $s$ ; setting mean and standard deviation in log-scale of the log-normal distribution to $-1/2 * \log(1/s + 1)$ and $(\log(1/s + 1))^{0.5}$ ; and setting shape and scale parameters of the pareto distribution to $(s + 1)^{0.5} + 1$ and $(s + 1)^{0.5} / ((s + 1)^{0.5} + 1)$ . If $s$ is 0, all node fitnesses $\eta$ are fixed at 1 (i.e., Barabási-Albert model). The default value is 10.

**Value**

The output is a PAFit\_net object, which is a List contains the following four fields:

graph	a three-column matrix, where each row contains information of one edge, in the form of (from_id, to_id, time_stamp). from_id is the id of the source, to_id is the id of the destination.
type	a string indicates whether the network is "directed" or "undirected".
PA	a numeric vector contains the true PA function.
fitness	fitness values of nodes in the network. The name of each value is the ID of the node.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**References**

1. Caldarelli, G., Capocci, A., De Los Rios, P. & Muñoz, M.A. (2002). Scale-Free Networks from Varying Vertex Intrinsic Fitness. Phys. Rev. Lett., 89, 258702 ([doi:10.1103/PhysRevLett.89.258702](https://doi.org/10.1103/PhysRevLett.89.258702)).

**See Also**

For subsequent estimation procedures, see [get\\_statistics](#).

For other functions to generate networks, see [generate\\_net](#), [generate\\_BA](#), [generate\\_ER](#) and [generate\\_BB](#).

**Examples**

```
library("PAFit")
# generate a network from the Caldarelli model with alpha = 1, N = 100, m = 1
# the inverse variance of distribution of node fitnesses is s = 10
net <- generate_fit_only(N = 100, m = 1, mode = 1, s = 10)
str(net)
plot(net)
```

---

generate\_net

*Simulating networks from preferential attachment and fitness mechanisms*

---

**Description**

This function generates networks from the General Temporal model, a generative temporal network model that includes many well-known models such as the Erdős–Rényi model, the Barabási-Albert model or the Bianconi-Barabási model as special cases. This function also includes some flexible mechanisms to vary the number of new nodes and new edges at each time-step in order to generate realistic networks.

**Usage**

```

generate_net (N           = 1000  ,
              num_seed    = 2      ,
              multiple_node = 1     ,
              specific_start = NULL ,
              m            = 1     ,
              prob_m       = FALSE  ,
              increase     = FALSE  ,
              log          = FALSE  ,
              no_new_node_step = 0   ,
              m_no_new_node_step = m ,
              custom_PA    = NULL   ,
              mode         = 1     ,
              alpha        = 1     ,
              beta         = 2     ,
              sat_at       = 100   ,
              offset       = 1     ,
              mode_f       = "gamma",
              s            = 10    )

```

**Arguments**

The parameters can be divided into four groups.

The first group specifies basic properties of the network:

- N** Integer. Total number of nodes in the network (including the nodes in the seed graph). Default value is 1000.
- num\_seed** Integer. The number of nodes of the seed graph (the initial state of the network). The seed graph is a cycle. Default value is 2.
- multiple\_node** Positive integer. The number of new nodes at each time-step. Default value is 1.
- specific\_start** Positive Integer. If `specific_start` is specified, then all the time-steps from time-step 1 to `specific_start` are grouped to become the initial time-step in the final output. This option is usefull when we want to create a network with a large initial network that follows a scale-free degree distribution. Default value is NULL.

The second group specifies the number of new edges at each time-step:

- m** Positive integer. The number of edges of each new node. Default value is 1.
- prob\_m** Logical. Indicates whether we fix the number of edges of each new node as a constant, or let it follows a Poisson distribution. If `prob_m == TRUE`, the number of edges of each new node follows a Poisson distribution. The mean of this distribution depends on the value of `increase` and `log`. Default value is FALSE.
- increase** Logical. Indicates whether we increase the mean of the Poisson distribution over time. If `increase == FALSE`, the mean is fixed at `m`. If `increase == TRUE`, the way the mean increases depends on the value of `log`. Default value is FALSE.

log	Logical. Indicates how to increase the mean of the Poisson distribution. If log == TRUE, the mean increases logarithmically with the number of current nodes. If log == FALSE, the mean increases linearly with the number of current nodes. Default value is FALSE.
no_new_node_step	Non-negative integer. The number of time-steps in which no new node is added, while new edges are added between existing nodes. Default value is 0, i.e., new nodes are always added at each time-step.
m_no_new_node_step	Positive integer. The number of new edges in the no-new-node steps. Default value is equal to m. Note that the number of new edges in the no-new-node steps is not effected by the parameters increase or prob_m; this number is always the constant specified by m_no_new_node_step.

The third group of parameters specifies the preferential attachment function:

custom_PA	Numeric vector. This is the user-input PA function: $A_0, A_1, \dots, A_K$ . If custom_PA is specified, then mode is ignored, and we grow the network using the PA function custom_PA. Degrees greater than $K$ will have attachment value $A_K$ . Default value is NULL.
mode	Integer. Indicates the parametric attachment function to be used in generating the network. If mode == 1, the attachment function is $A_k = k^\alpha$ . If mode == 2, the attachment function is $A_k = \min(k, sat.at)^\alpha$ . If mode == 3, the attachment function is $A_k = \alpha \log(k)^\beta$ . Default value is 1.
alpha	Numeric. If mode == 1, this is the attachment exponent in the attachment function $A_k = k^\alpha$ . If mode == 2, this is the attachment exponent in the attachment function $A_k = \min(k, sat.at)^\alpha$ . If mode == 3, this is the $\alpha$ in the attachment function $A_k = \alpha \log(k)^\beta + 1$ .
beta	Numeric. This is the beta in the attachment function $A_k = \alpha \log(k)^\beta + 1$ .
sat_at	Integer. This is the saturation position <i>sat.at</i> in the attachment function $A_k = \min(k, sat.at)^\alpha$ .
offset	Numeric. The attachment value of degree 0. Default value is 1.

The final group of parameters specifies the distribution from which node fitnesses are generated:

mode_f	String. Possible values: "gamma", "log_normal" or "power_law". This parameter indicates the true distribution for node fitness. "gamma" = gamma distribution, "log_normal" = log-normal distribution. "power_law" = power-law (pareto) distribution. Default value is "gamma".
s	Non-negative numeric. The inverse variance parameter. The mean of the distribution is kept at 1 and the variance is $1/s$ (since node fitnesses are only meaningful up to scale). This is achieved by setting shape and rate parameters of the Gamma distribution to $s$ ; setting mean and standard deviation in log-scale of the log-normal distribution to $-1/2 * \log(1/s + 1)$ and $(\log(1/s + 1))^{0.5}$ ; and setting shape and scale parameters of the pareto distribution to $(s + 1)^{0.5} + 1$ and $(s + 1)^{0.5} / ((s + 1)^{0.5} + 1)$ . If $s$ is 0, all node fitnesses $\eta$ are fixed at 1 (i.e., Barabási-Albert model). The default value is 10.



**Value**

The output is a PAFit\_net object, which is a List contains the following four fields:

graph	a three-column matrix, where each row contains information of one edge, in the form of (from_id, to_id, time_stamp). from_id is the id of the source, to_id is the id of the destination.
type	a string indicates whether the network is "directed" or "undirected".
PA	a numeric vector contains the true PA function.
fitness	fitness values of nodes in the network. The name of each value is the ID of the node.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**See Also**

For subsequent estimation procedures, see [get\\_statistics](#).

For simpler functions to generate networks from well-known models, see [generate\\_BA](#), [generate\\_ER](#), [generate\\_BB](#) and [generate\\_fit\\_only](#).

**Examples**

```
library("PAFit")
#Generate a network from the original BA model with alpha = 1, N = 100, m = 1
net <- generate_net(N = 100,m = 1,mode = 1, alpha = 1, s = 0)
str(net)
plot(net)
```

---

generate\_simulated\_data\_from\_estimated\_model

*Generating simulated data from a fitted model*

---

**Description**

This function generates simulated networks from a fitted model and performs estimations on these simulated networks with the same setting used in the original estimation. Each simulated network is generated using parameters of the fitted model, while keeping other aspects of the growth process as faithfully as possible to the original observed network.

**Usage**

```
generate_simulated_data_from_estimated_model(net_object, net_stat, result, M = 5)
```

**Arguments**

net_object	an object of class PAFit_net that contains the original network.
net_stat	An object of class PAFit_data which contains summarized statistics of the original network. This object is created by the function <a href="#">get_statistics</a> .
result	An object of class Full_PAFit_result which contains the fitted model obtained by applying the function <a href="#">joint_estimate</a> .
M	integer. The number of simulated networks. Default value is 5.

**Value**

Outputs a Simulated\_Data\_From\_Fitted\_Model object, which is a list containing the following fields:

- graph\_list: a list containing M simulated graphs.
- stats\_list: a list containing M objects of class PAFit\_data, which are the results of applying [get\\_statistics](#) on the simulated graphs.
- result\_list: a list containing M objects of class Full\_PAFit\_result, which are the results of applying [joint\\_estimate](#) on the simulated graphs.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**References**

1. Pham, T., Sheridan, P. & Shimodaira, H. (2015). PAFit: A Statistical Method for Measuring Preferential Attachment in Temporal Complex Networks. PLoS ONE 10(9): e0137796. ([doi:10.1371/journal.pone.0137796](#)).
2. Pham, T., Sheridan, P. & Shimodaira, H. (2016). Joint Estimation of Preferential Attachment and Node Fitness in Growing Complex Networks. Scientific Reports 6, Article number: 32558. ([doi:10.1038/srep32558](#)).
3. Pham, T., Sheridan, P. & Shimodaira, H. (2020). PAFit: An R Package for the Non-Parametric Estimation of Preferential Attachment and Node Fitness in Temporal Complex Networks. Journal of Statistical Software 92 (3). ([doi:10.18637/jss.v092.i03](#)).
4. Inoue, M., Pham, T. & Shimodaira, H. (2020). Joint Estimation of Non-parametric Transitivity and Preferential Attachment Functions in Scientific Co-authorship Networks. Journal of Informetrics 14(3). ([doi:10.1016/j.joi.2020.101042](#)).

**See Also**

[get\\_statistics](#), [joint\\_estimate](#), [plot\\_contribution](#)

**Examples**

```
## Not run:

library("PAFit")
net_object <- generate_net(N = 500, m = 10, s = 10, alpha = 0.5)
net_stat <- get_statistics(net_object)
result <- joint_estimate(net_object, net_stat)
simulated_data <- generate_simulated_data_from_estimated_model(net_object, net_stat, result)
plot_contribution(simulated_data, result, which_plot = "PA")
plot_contribution(simulated_data, result, which_plot = "fit")

## End(Not run)
```

---

<code>get_statistics</code>	<i>Getting summarized statistics from input data</i>
-----------------------------	--

---

**Description**

The function summarizes input data into sufficient statistics for estimating the attachment function and node fitness, together with additional information about the data, such as total number of nodes, number of time-steps, maximum degree, and the final degree of the network, etc. . It also provides mechanisms to automatically deal with very large datasets by binning the degree, setting a degree threshold, or grouping time-steps.

**Usage**

```
get_statistics(net_object, only_PA = FALSE ,
              only_true_deg_matrix = FALSE ,
              binning = TRUE , g = 50 ,
              deg_threshold = 0 ,
              compress_mode = 0 , compress_ratio = 0.5 ,
              custom_time = NULL)
```

**Arguments**

The parameters can be divided into four groups. The first group specifies input data and how the data will be summarized:

<code>net_object</code>	An object of class <code>PAFit_net</code> . You can use the function <a href="#">as.PAFit_net</a> to convert from an edgelist matrix, function <a href="#">from_igraph</a> to convert from an <code>igraph</code> object, function <a href="#">from_networkDynamic</a> to convert from a <code>networkDynamic</code> object, and function <a href="#">graph_from_file</a> to read from a file.
<code>only_PA</code>	Logical. Indicates whether only the statistics for estimating $A_k$ are summarized. if TRUE, the statistics for estimating $\eta_i$ are NOT collected. This will save memory at the cost of unable to estimate node fitness). Default value is FALSE.

**only\_true\_deg\_matrix**

Logical. Return only the true degree matrix (without binning), and no other statistics is returned. The result cannot be used in `PAFit` function to estimate PA or fitness. The motivation for this option is that sometimes we only want to get a degree matrix that summarizes the growth process of a very big network for plotting etc. Default value is `FALSE`.

Second group of parameters specifies how to bin the degrees:

**binning** Logical. Indicates whether the degree should be binned together. Default value is `TRUE`.

**g** Positive integer. Number of bins. Should be at least 3. Default value is 50.

Third group contains a single parameter specifying how to reduce the number of node fitnesses:

**deg\_threshold** Integer. We only estimate the fitnesses of nodes whose number of new edges acquired is at least `deg_threshold`. The fitnesses of all other nodes are fixed at 1. Default value is 0.

Last group of parameters specifies how to group the time-stamps:

**compress\_mode** Integer. Indicates whether the timeline should be compressed. The value of `CompressMode`:

0: No compression

1: Compressed by using a subset of time-steps. The time stamps in this subset are equally spaced. The size of this subset is `CompressRatio` times the size of the set of all time stamps.

2: Compressed by only starting from the first time-step when `CompressRatio*100` percentages of the total number of edges (in the final state of the network) had already been added to the network.

3: This mode offers the most flexibility, but requires user to supply the time stamps in `CustomTime`. Only time stamps in this `CustomTime` will be used. This mode can be used, for example, when investigating the change of the attachment function or node fitness in different time intervals.

Default value is 0, i.e. no compression.

**compress\_ratio** Numeric. Indicates how much we should compress if `CompressMode` is 1 or 2. Default value is 0.5.

**custom\_time** Vector. Custom time stamps. This vector is a subset of the vector that contains all time-stamps. Only effective if `CompressMode == 3`. In that case, only these time stamps are used.

**Value**

An object of class `PAFit_data`, which is a list. Some important fields are:

**offset\_tk** A matrix where the  $(t, k+1)$  element is the number of nodes with degree  $k$  at time  $t$ , counting among all the nodes whose number of new edges acquired is less than `deg_thresh`

**n\_tk** A matrix where the  $(t, k+1)$  element is the number of nodes with degree  $k$  at time  $t$

mTk	A matrix where the $(t, k+1)$ element is the number of new edges connect to a degree- $k$ node at time $t$
sum_m_k	A vector where the $(k+1)$ -th element is the total number of edges that linked to a degree $k$ node, counting over all time steps
node_degree	A matrix recording the degree of all nodes (that satisfy degree_threshold condition) at each time step
m_t	A vector where the $t$ -th element is the number of new edges at time $t$
z_j	A vector where the $j$ -th element is the total number of edges that linked to node $j$
N	Numeric. The number of nodes in the network
T	Numeric. The number of time steps
deg_max	Numeric. The maximum degree in the final network
node_id	A vector contains the id of all nodes
final_deg	A vector contains the final degree of all nodes (including those that do not satisfy the degree_threshold condition)
deg_thresh	Integer. The specified degree threshold.
f_position	Numeric vector. The index in the node_id vector of the nodes we want to estimate (i.e. nodes whose number of new edges acquired is at least deg_thresh)
start_deg	Integer. The specified degree at which we start binning.
begin_deg	Numeric vector contains the beginning degree of each bin
end_deg	Numeric vector contains the ending degree of each bin
interval_length	Numeric vector contains the length of each bin.
binning	Logical. Indicates whether binning was applied or not.
g	Integer. Number of bins
time_compress_mode	Integer. The mode of time compression.
t_compressed	Integer. The number of time stamps actually used
compressed_unique_time	The time stamps that are actually used
compress_ratio	Numeric.
custom_time	Vector. The time stamps specified by user.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**See Also**

For creating the needed input for this function (a PAFit\_net object), see [as.PAFit\\_net](#), [from\\_igraph](#), [from\\_networkDynamic](#), and [graph\\_from\\_file](#).

For the next step, see [Newman](#), [Jeong](#) or [only\\_A\\_estimate](#) for estimating the attachment function in isolation, [only\\_F\\_estimate](#) for estimating node fitnesses in isolation, and [joint\\_estimate](#) for joint estimation of the attachment function and node fitnesses.

**Examples**

```
library("PAFit")
net      <- generate_BA(N = 100 , m = 1)
net_stats <- get_statistics(net)
summary(net_stats)
```

---

graph_from_file	<i>Read file to a PAFit_net object</i>
-----------------	--

---

**Description**

This function reads an input file to a PAFit\_net object. Accepted formats are the edgelist format or the gml format.

**Usage**

```
graph_from_file(file_name, format = "edgelist", type = "directed")
```

**Arguments**

file_name	A string indicates the file name.
format	String. Possible values are "edgelist" and "gml". If format is "edgelist", we assume the following edgelist matrix format. Each row is assumed to be of the form (from_node_id to_node_id time_stamp). from_node_id is the id of the source node. to_node_id is the id of the destination node. time_stamp is the arrival time of the edge. from_node_id and to_node_id are assumed to be integers that are at least 0. They need not to be contiguous. To register a new node $i$ at time $t$ without any edge, add a row with format (i -1 t). This works for both undirected and directed networks. time_stamp can be either numeric or string. The value of a time-stamp can be arbitrary, but we assume that a smaller time_stamp (regarded so by the sort function in R) represents an earlier arrival time. Examples of time-stamps that satisfy this assumption are the integer 0:T, the string format 'yyyy-mm-dd', and the POSIX time. If format is "gml", there must be a binary field directed indicating the type of the network (0: undirected, 1: directed). The required fields for an edge are: source, target, and time. source and target are the ID of the source node and the target node, respectively. time is the time-stamp of the edge. The required fields for a node are: id, isolated (binary) and time. The binary field isolated indicates whether this node is an isolated node when it enters the system or not. If isolated is 1, then time must contain the node's appearance time. If isolated is 0, then we can automatically infer the node's appearance time from its edges, so the field time in this case can be NULL. The assumptions on node IDs and the format of time-stamps are the same as in the case when format = "edgelist". See <a href="#">graph_to_file</a> to see detail on the format of the gml file this package outputs.

type           String. Indicates whether the network is "directed" or "undirected". This option is ignored if format is "gml", since the information is assumed to be contained in the gml file.

### Value

An object of class PAFit\_net containing the network.

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### Examples

```
library("PAFit")
# a network from Bianconi-Barabasi model
net      <- generate_BB(N = 50 , m = 10 , s = 10)

#graph_to_file(net, file_name = "test.gml", format = "gml")
#reread      <- graph_from_file(file_name = "test.gml", format = "gml")
```

---

graph\_to\_file           *Write the graph in a PAFit\_net object to file*

---

### Description

This function writes a graph in a PAFit\_net object to an output file. Accepted file formats are the edgelist format or the gml format.

### Usage

```
graph_to_file(net_object, file_name, format = "edgelist")
```

### Arguments

net\_object       An object of class PAFit\_net.

file\_name        A string indicates the file name.

format           String. Possible values are "edgelist" and "gml".  
 If format = "edgelist", we just output the edgelist matrix contained in the PAFit\_net object as it is.  
 If format = "gml", here is the specification of the gml file. There is a binary field directed indicating the type of the network (0: undirected, 1: directed). There are three attributes for an edge: source, target, and time. There are three attributes for a node: id, isolated (binary) and time. The attribute time is NULL if the attribute isolated is 0 (since this is not an isolated node, we do not need to record its first appearance time). On the other hand, time is the node's appearance time if attribute isolated is 1.

**Value**

The function writes directly to the output file.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```
library("PAFit")
# a network from Bianconi-Barabasi model
net      <- generate_BB(N = 50 , m = 10 , s = 10)
#graph_to_file(net, file_name = "test.gml", format = "gml")
```

---

Jeong

*Jeong's method for estimating the preferential attachment function*

---

**Description**

This function estimates the preferential attachment function by Jeong's method.

**Usage**

```
Jeong(net_object      ,
      net_stat = get_statistics(net_object) ,
      T_0_start = 0 ,
      T_0_end   = round(net_stat$T * 0.75) ,
      T_1_start = T_0_end + 1 ,
      T_1_end   = net_stat$T ,
      interpolate = FALSE)
```

**Arguments**

<code>net_object</code>	an object of class <code>PAFit_net</code> that contains the network.
<code>net_stat</code>	An object of class <code>PAFit_data</code> which contains summerized statistics needed in estimation. This object is created by the function <code>get_statistics</code> . Default value is <code>get_statistics(net_object)</code> .
<code>T_0_start</code>	Positive integer. The starting time-step of the <code>T_0_interval</code> . Default value is 0.
<code>T_0_end</code>	Positive integer. The ending time-step of <code>T_0_interval</code> . Default value is <code>round(net_stat\$T * 0.75)</code> .
<code>T_1_start</code>	Positive integer. The starting time-step of the <code>T_1_interval</code> . Default value is <code>T_0_end + 1</code> .
<code>T_1_end</code>	Positive integer. The ending time-step of <code>T_1_interval</code> . Default value is <code>net_stat\$T</code> .
<code>interpolate</code>	Logical. If TRUE then all the gaps in the estimated PA function are interpolated by linear interpolating in logarithm scale. Default value is FALSE.



**Value**

Outputs an `PA_result` object which contains the estimated attachment function. In particular, it contains the following field:

- `k` and `A`: a degree vector and the estimated PA function.
- `center_k` and `theta`: when we perform binning, these are the centers of the bins and the estimated PA values for those bins.
- `g`: the number of bins used.
- `alpha` and `ci`: `alpha` is the estimated attachment exponent  $\alpha$  (when assume  $A_k = k^\alpha$ ), while `ci` is the confidence interval.
- `loglinear_fit`: this is the fitting result when we estimate  $\alpha$ .

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**References**

1. Jeong, H., Néda, Z. & Barabási, A. . Measuring preferential attachment in evolving networks. *Europhysics Letters*. 2003;61(61):567–572. (doi:10.1209/epl/i2003001669).

**See Also**

See [get\\_statistics](#) for how to create summerized statistics needed in this function.

See [Newman](#) and [only\\_A\\_estimate](#) for other methods to estimate the attachment function in isolation.

**Examples**

```
library("PAFit")
net      <- generate_net(N = 1000 , m = 1 , mode = 1 , alpha = 1 , s = 0)
net_stats <- get_statistics(net)
result   <- Jeong(net, net_stats)
# true function
true_A   <- result$center_k
#plot the estimated attachment function
plot(result , net_stats)
lines(result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")
```

joint\_estimate

*Joint inference of attachment function and node fitnesses***Description**

This function jointly estimates the attachment function  $A_k$  and node fitnesses  $\eta_i$ . It first performs a cross-validation to select the optimal parameters  $r$  and  $s$ , then estimates  $A_k$  and  $\eta_i$  using that optimal pair with the full data (Ref. 2).

**Usage**

```
joint_estimate(net_object
               ,
               net_stat   = get_statistics(net_object),
               p           = 0.75
               ,
               stop_cond   = 10^-8
               ,
               mode_reg_A  = 0
               ,
               ...)
```

**Arguments**

net_object	an object of class PAFit_net that contains the network.
net_stat	An object of class PAFit_data which contains summarized statistics needed in estimation. This object is created by the function <code>get_statistics</code> . The default value is <code>get_statistics(net_object)</code> .
p	Numeric. This is the ratio of the number of new edges in the learning data to that of the full data. The data is then divided into two parts: learning data and testing data based on $p$ . The learning data is used to learn the node fitnesses and the testing data is then used in cross-validation. Default value is 0.75.
stop_cond	Numeric. The iterative algorithm stops when $\text{abs}(h(ii) - h(ii+1)) / (\text{abs}(h(ii)) + 1) < \text{stop.cond}$ where $h(ii)$ is the value of the objective function at iteration $ii$ . We recommend to choose <code>stop.cond</code> at most equal to $10^{(-\text{number of digits of } h - 2)}$ , in order to ensure that when the algorithm stops, the increase in posterior probability is less than 1% of the current posterior probability. Default is $10^{-8}$ . This threshold is good enough for most applications.
mode_reg_A	Binary. Indicates which regularization term is used for $A_k$ : <ul style="list-style-type: none"> <li>• 0: This is the regularization term used in Ref. 1 and 2. Please refer to Eq. (4) in the tutorial for the definition of the term. It approximately enforces the power-law form <math>A_k = k^\alpha</math>. This is the default value.</li> <li>• 1: Unlike the default, this regularization term exactly enforces the functional form <math>A_k = k^\alpha</math>. Please refer to Eq. (6) in the tutorial for the definition of the term. Its main drawback is it is significantly slower to converge, while its gain over the default one is marginal in most cases.</li> </ul>
...	Other arguments to pass to the underlying algorithm.

**Value**

Outputs a `Full_PAFit_result` object, which is a list containing the following fields:

- `cv_data`: a `CV_Data` object which contains the cross-validation data. This is the testing data.
- `cv_result`: a `CV_Result` object which contains the cross-validation result. Normally the user does not need to pay attention to this data.
- `estimate_result`: this is a `PAFit_result` object which contains the estimated attachment function  $A_k$ , the estimated fitnesses  $\eta_i$  and their confidence intervals. In particular, the important fields are:
  - `ratio`: this is the selected value for the hyper-parameter  $r$ .
  - `shape`: this is the selected value for the hyper-parameter  $s$ .
  - `k` and `A`: a degree vector and the estimated PA function.
  - `var_A`: the estimated variance of  $A$ .
  - `var_logA`: the estimated variance of  $\log A$ .
  - `upper_A`: the upper value of the interval of two standard deviations around  $A$ .
  - `lower_A`: the lower value of the interval of two standard deviations around  $A$ .
  - `center_k` and `theta`: when we perform binning, these are the centers of the bins and the estimated PA values for those bins. `theta` is similar to  $A$  but with duplicated values removed.
  - `var_bin`: the variance of `theta`. Same as `var_A` but with duplicated values removed.
  - `upper_bin`: the upper value of the interval of two standard deviations around `theta`. Same as `upper_A` but with duplicated values removed.
  - `lower_bin`: the lower value of the interval of two standard deviations around `theta`. Same as `lower_A` but with duplicated values removed.
  - `g`: the number of bins used.
  - `alpha` and `ci`: `alpha` is the estimated attachment exponent  $\alpha$  (when assume  $A_k = k^\alpha$ ), while `ci` is the confidence interval.
  - `loglinear_fit`: this is the fitting result when we estimate  $\alpha$ .
  - `f`: the estimated node fitnesses.
  - `var_f`: the estimated variance of  $\eta_i$ .
  - `upper_f`: the estimated upper value of the interval of two standard deviations around  $\eta_i$ .
  - `lower_f`: the estimated lower value of the interval of two standard deviations around  $\eta_i$ .
  - `objective_value`: values of the objective function over iterations in the final run with the full data.
  - `diverge_zero`: logical value indicates whether the algorithm diverged in the final run with the full data.
- `contribution`: a list containing an estimate of the contributions of preferential attachment and fitness mechanisms in the growth process of the network. The calculation adapts a quantification method proposed in Section 3 of Ref. 4, which is for preferential attachment and transitivity, to preferential attachment and fitness.
  - `PA_contribution`: an array containing the contributions of preferential attachment at each time-step
  - `fit_contribution`: an array containing the contributions of the fitness mechanism at each time-step

- mean\_PA\_contrib: the average contribution of preferential attachment through the whole growth process
- mean\_fit\_contrib: the average contribution of the fitness mechanism through the whole growth process

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### References

1. Pham, T., Sheridan, P. & Shimodaira, H. (2015). PAFit: A Statistical Method for Measuring Preferential Attachment in Temporal Complex Networks. PLoS ONE 10(9): e0137796. (doi:10.1371/journal.pone.0137796).
2. Pham, T., Sheridan, P. & Shimodaira, H. (2016). Joint Estimation of Preferential Attachment and Node Fitness in Growing Complex Networks. Scientific Reports 6, Article number: 32558. (doi:10.1038/srep32558).
3. Pham, T., Sheridan, P. & Shimodaira, H. (2020). PAFit: An R Package for the Non-Parametric Estimation of Preferential Attachment and Node Fitness in Temporal Complex Networks. Journal of Statistical Software 92 (3). (doi:10.18637/jss.v092.i03).
4. Inoue, M., Pham, T. & Shimodaira, H. (2020). Joint Estimation of Non-parametric Transitivity and Preferential Attachment Functions in Scientific Co-authorship Networks. Journal of Informetrics 14(3). (doi:10.1016/j.joi.2020.101042).

### See Also

See [get\\_statistics](#) for how to create summarized statistics needed in this function.

See [Jeong](#), [Newman](#) and [only\\_A\\_estimate](#) for functions to estimate the attachment function in isolation.

See [only\\_F\\_estimate](#) for a function to estimate node fitnesses in isolation.

### Examples

```
## Not run:

library("PAFit")
#### Example 1: a linear preferential attachment kernel, i.e.,  $A_k = k$  #####
set.seed(1)
# size of initial network = 100
# number of new nodes at each time-step = 100
#  $A_k = k$ ; inverse variance of the distribution of node fitnesses = 5
net      <- generate_BB(N      = 1000 , m      = 50 ,
                      num_seed = 100  , multiple_node = 100,
                      s        = 5)
net_stats <- get_statistics(net)

# Joint estimation of attachment function  $A_k$  and node fitness
result   <- joint_estimate(net, net_stats)
```

```

summary(result)

# plot the estimated attachment function
true_A <- pmax(result$estimate_result$center_k,1) # true function
plot(result , net_stats, max_A = max(true_A,result$estimate_result$theta))
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")

# plot the estimated node fitnesses and true node fitnesses
plot(result, net_stats, true = net$fitness, plot = "true_f")

#####
### Example 2: a non-log-linear preferential attachment kernel #####
set.seed(1)
# size of initial network = 100
# number of new nodes at each time-step = 100
# A_k = alpha* log (max(k,1))^beta + 1, with alpha = 2, and beta = 2
# inverse variance of the distribution of node fitness = 10
net <- generate_net(N = 1000 , m = 50 ,
                   num_seed = 100 , multiple_node = 100,
                   s = 10 , mode = 3, alpha = 2, beta = 2)
net_stats <- get_statistics(net)

# Joint estimation of attachment function Ak and node fitness
result <- joint_estimate(net, net_stats)

summary(result)

# plot the estimated attachment function
true_A <- 2 * log(pmax(result$estimate_result$center_k,1))^2 + 1 # true function
plot(result , net_stats, max_A = max(true_A,result$estimate_result$theta))
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")

# plot the estimated node fitnesses and true node fitnesses
plot(result, net_stats, true = net$fitness, plot = "true_f")
#####
### Example 3: another non-log-linear preferential attachment kernel #####
set.seed(1)
# size of initial network = 100
# number of new nodes at each time-step = 100
# A_k = min(max(k,1),sat_at)^alpha, with alpha = 1, and sat_at = 100
# inverse variance of the distribution of node fitness = 10
net <- generate_net(N = 1000 , m = 50 ,
                   num_seed = 100 , multiple_node = 100,
                   s = 10 , mode = 2, alpha = 1, sat_at = 100)
net_stats <- get_statistics(net)

# Joint estimation of attachment function Ak and node fitness
result <- joint_estimate(net, net_stats)

summary(result)

```

```

# plot the estimated attachment function
true_A <- pmin(pmax(result$estimate_result$center_k,1),100)^1 # true function
plot(result , net_stats, max_A = max(true_A,result$estimate_result$theta))
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")

# plot the estimated node fitnesses and true node fitnesses
plot(result, net_stats, true = net$fitness, plot = "true_f")

## End(Not run)

```

---

Newman

---

*Corrected Newman's method for estimating the preferential attachment function*


---

### Description

This function implements a correction proposed in [1] of the original Newman's method in [2] to estimate the preferential attachment function.

### Usage

```

Newman(net_object
       ,
       net_stat   = get_statistics(net_object),
       start      = 1
       ,
       interpolate = FALSE)

```

### Arguments

<code>net_object</code>	an object of class <code>PAFit_net</code> that contains the network.
<code>net_stat</code>	An object of class <code>PAFit_data</code> which contains summarized statistics needed in estimation. This object is created by the function <code>get_statistics</code> . Default value is <code>get_statistics(net_object)</code> .
<code>start</code>	Positive integer. The starting time from which the method is applied. Default value is 1.
<code>interpolate</code>	Logical. If <code>TRUE</code> then all the gaps in the estimated PA function are interpolated by linear interpolating in logarithm scale. Default value is <code>FALSE</code> .

### Value

Outputs an `PA_result` object which contains the estimated attachment function. In particular, it contains the following field:

- `k` and `A`: a degree vector and the estimated PA function.
- `center_k` and `theta`: when we perform binning, these are the centers of the bins and the estimated PA values for those bins.
- `g`: the number of bins used.

- alpha and ci: alpha is the estimated attachment exponent  $\alpha$  (when assume  $A_k = k^\alpha$ ), while ci is the mean plus/minus two-standard-deviation interval.
- loglinear\_fit: this is the fitting result when we estimate  $\alpha$ .

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### References

1. Pham, T., Sheridan, P. & Shimodaira, H. (2015). PAFit: A Statistical Method for Measuring Preferential Attachment in Temporal Complex Networks. PLoS ONE 10(9): e0137796. (doi:10.1371/journal.pone.0137796).
2. Newman, M.. Clustering and preferential attachment in growing networks. Physical Review E. 2001;64(2):025102 (doi:10.1103/PhysRevE.64.025102).

### See Also

See [get\\_statistics](#) for how to create summerized statistics needed in this function.

See [Jeong](#), [only\\_A\\_estimate](#) for other methods to estimate the attachment function in isolation.

### Examples

```
library("PAFit")
net      <- generate_net(N = 1000 , m = 1 , mode = 1 , alpha = 1 , s = 0)
net_stats <- get_statistics(net)
result   <- Newman(net, net_stats)
summary(result)
# true function
true_A   <- result$center_k
#plot the estimated attachment function
plot(result , net_stats)
lines(result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")
```

---

only\_A\_estimate

*Estimating the attachment function in isolation by PAFit method*

---

### Description

This function estimates the attachment function  $A_k$  by PAFit method. The method has a hyperparameter  $r$ . It first performs a cross-validation step to select the optimal parameter  $r$  for the regularization of  $A_k$ , then uses that  $r$  to estimate the attachment function with the full data.

**Usage**

```
only_A_estimate(net_object,
               net_stat = get_statistics(net_object),
               p         = 0.75,
               stop_cond = 10^-8,
               mode_reg_A = 0,
               MLE       = FALSE,
               ...)
```

**Arguments**

net_object	an object of class PAFit_net that contains the network.
net_stat	An object of class PAFit_data which contains summarized statistics needed in estimation. This object is created by the function <code>get_statistics</code> . The default value is <code>get_statistics(net_object)</code> .
p	Numeric. This is the ratio of the number of new edges in the learning data to that of the full data. The data is then divided into two parts: learning data and testing data based on p. The learning data is used to learn the node fitnesses and the testing data is then used in cross-validation. Default value is 0.75.
stop_cond	Numeric. The iterative algorithm stops when $\frac{abs(h(ii)-h(ii+1))}{(abs(h(ii))+1)} < stop\_cond$ where $h(ii)$ is the value of the objective function at iteration $ii$ . We recommend to choose <code>stop_cond</code> at most equal to $10^{-(number\ of\ digit\ of\ h-2)}$ , in order to ensure that when the algorithm stops, the increase in posterior probability is less than 1% of the current posterior probability. Default is $10^{-8}$ . This threshold is good enough for most applications.
mode_reg_A	Binary. Indicates which regularization term is used for $A_k$ : <ul style="list-style-type: none"> <li>• 0: This is the regularization term used in Ref. 1 and 2. Please refer to Eq. (4) in the tutorial for the definition of the term. It approximately enforces the power-law form <math>A_k = k^\alpha</math>. This is the default value.</li> <li>• 1: Unlike the default, this regularization term exactly enforces the functional form <math>A_k = k^\alpha</math>. Please refer to Eq. (6) in the tutorial for the definition of the term. Its main drawback is it is significantly slower to converge, while its gain over the default one is marginal in most cases.</li> </ul>
MLE	Logical. If TRUE, then not perform cross-validation and estimate the PA function with $r = 0$ , i.e., maximum likelihood estimation. Default is FALSE. One might want to set this option to TRUE when one believes that there are sufficient data to get a reasonable MLE result, or when one wants to compare the default, regularized result with the MLE result.
...	Other arguments to pass to the underlying algorithm.

**Value**

Outputs a `Full_PAFit_result` object, which is a list containing the following fields:

- `cv_data`: a `CV_Data` object which contains the cross-validation data. This is the final Normally the user does not need to pay attention to this data. NULL if `MLE = TRUE`.



- `cv_result`: a `CV_Result` object which contains the cross-validation result. Normally the user does not need to pay attention to this data. NULL if `MLE = TRUE`.
- `estimate_result`: this is a `PAFit_result` object which contains the estimated PA function and its confidence interval. It also includes the estimated attachment exponent  $\alpha$  (assuming the model  $A_k = k^\alpha$ ) in the field `alpha`, and the confidence interval of  $\alpha$  (in the field `ci`) when possible. In particular, the important fields are:
  - `ratio`: this is the selected value for the hyper-parameter  $r$ .
  - `k` and `A`: a degree vector and the estimated PA function.
  - `var_A`: the estimated variance of  $A$ .
  - `var_logA`: the estimated variance of  $\log A$ .
  - `upper_A`: the upper value of the interval of two standard deviations around  $A$ .
  - `lower_A`: the lower value of the interval of two standard deviations around  $A$ .
  - `center_k` and `theta`: when we perform binning, these are the centers of the bins and the estimated PA values for those bins. `theta` is similar to  $A$  but with duplicated values removed.
  - `var_bin`: the variance of `theta`. Same as `var_A` but with duplicated values removed.
  - `upper_bin`: the upper value of the interval of two standard deviations around `theta`. Same as `upper_A` but with duplicated values removed.
  - `lower_lower`: the lower value of the interval of two standard deviations around `theta`. Same as `lower_A` but with duplicated values removed.
  - `g`: the number of bins used.
  - `alpha` and `ci`: `alpha` is the estimated attachment exponent  $\alpha$  (when assume  $A_k = k^\alpha$ ), while `ci` is the confidence interval.
  - `loglinear_fit`: this is the fitting result when we estimate  $\alpha$ .
  - `objective_value`: values of the objective function over iterations in the final run with the full data.
  - `diverge_zero`: logical value indicates whether the algorithm diverged in the final run with the full data.

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### References

1. Pham, T., Sheridan, P. & Shimodaira, H. (2015). PAFit: A Statistical Method for Measuring Preferential Attachment in Temporal Complex Networks. *PLoS ONE* 10(9): e0137796. (doi:10.1371/journal.pone.0137796).
2. Pham, T., Sheridan, P. & Shimodaira, H. (2016). Joint Estimation of Preferential Attachment and Node Fitness in Growing Complex Networks. *Scientific Reports* 6, Article number: 32558. (doi:10.1038/srep32558).

### See Also

See `get_statistics` for how to create summerized statistics needed in this function.

See [Newman](#) and [Jeong](#) for other methods to estimate the attachment function  $A_k$  in isolation.

## Examples

```
## Not run:
library("PAFit")
set.seed(1)
#### Example 1: Linear preferential attachment #####
# a network from BA model
net      <- generate_net(N = 1000 , m = 50 , mode = 1, alpha = 1, s = 0)

net_stats <- get_statistics(net, only_PA = TRUE)
result    <- only_A_estimate(net, net_stats)

# plot the estimated attachment function
plot(result, net_stats)

# true function
true_A    <- result$estimate_result$center_k
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")

#### Example 2: a non-log-linear preferential attachment #####
#  $A_k = \alpha * \log(\max(k,1))^\beta + 1$ , with  $\alpha = 2$ , and  $\beta = 2$ 
set.seed(1)
net      <- generate_net(N = 1000 , m = 50 , mode = 3, alpha = 2, beta = 2, s = 0)

net_stats <- get_statistics(net, only_PA = TRUE)
result    <- only_A_estimate(net, net_stats)

# plot the estimated attachment function
plot(result, net_stats)

# true function
true_A    <- 2 * log(pmax(result$estimate_result$center_k,1))^2 + 1 # true function
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")

#####
#### Example 3: another non-log-linear preferential attachment kernel #####
set.seed(1)
#  $A_k = \min(\max(k,1), \text{sat\_at})^\alpha$ , with  $\alpha = 1$ , and  $\text{sat\_at} = 200$ 
# inverse variance of the distribution of node fitness = 10
net      <- generate_net(N = 1000 , m = 50 , mode = 2, alpha = 1, sat_at = 200, s = 0)
net_stats <- get_statistics(net, only_PA = TRUE)

result    <- only_A_estimate(net, net_stats)

# plot the estimated attachment function
true_A    <- pmin(pmax(result$estimate_result$center_k,1),200)^1 # true function
plot(result , net_stats, max_A = max(true_A,result$estimate_result$theta))
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")
```

```
## End(Not run)
```

---

only_F_estimate	<i>Estimating node fitnesses in isolation</i>
-----------------	---

---

## Description

This function estimates node fitnesses  $\eta_i$  assuming either  $A_k = k$  (i.e. linear preferential attachment) or  $A_k = 1$  (i.e. no preferential attachment). The method has a hyper-parameter  $s$ . It first performs a cross-validation to select the optimal parameter  $s$  for the prior of  $\eta_i$ , then estimates  $\eta_i$  with the full data (Ref. 1).

## Usage

```
only_F_estimate(net_object,
               net_stat = get_statistics(net_object),
               p = 0.75,
               stop_cond = 10^-8,
               model_A = "Linear",
               ...)
```

## Arguments

net_object	an object of class PAFit_net that contains the network.
net_stat	An object of class PAFit_data which contains summarized statistics needed in estimation. This object is created by the function <code>get_statistics</code> . The default value is <code>get_statistics(net_object)</code> .
p	Numeric. This is the ratio of the number of new edges in the learning data to that of the full data. The data is then divided into two parts: learning data and testing data based on p. The learning data is used to learn the node fitnesses and the testing data is then used in cross-validation. Default value is 0.75.
stop_cond	Numeric. The iterative algorithm stops when $\frac{abs(h(ii)-h(ii+1))}{(abs(h(ii))+1)} < stop\_cond$ where $h(ii)$ is the value of the objective function at iteration $ii$ . We recommend to choose <code>stop_cond</code> at most equal to $10^{-(\text{number of digits of } h - 2)}$ , in order to ensure that when the algorithm stops, the increase in posterior probability is less than 1% of the current posterior probability. Default is $10^{-8}$ . This threshold is good enough for most applications.
model_A	String. Indicates which attachment function $A_k$ we assume: <ul style="list-style-type: none"> <li>"Linear": We assume <math>A_k = k</math>, i.e. the Bianconi-Barabási model (Ref. 2).</li> <li>"Constant": We assume <math>A_k = 1</math>, i.e. the Caldarelli model (Ref. 3).</li> </ul>
...	Other arguments to pass to the underlying algorithm.

**Value**

Outputs a `Full_PAFit_result` object, which is a list containing the following fields:

- `cv_data`: a `CV_Data` object which contains the cross-validation data. Normally the user does not need to pay attention to this data.
- `cv_result`: a `CV_Result` object which contains the cross-validation result. Normally the user does not need to pay attention to this data.
- `estimate_result`: this is a `PAFit_result` object which contains the estimated node fitnesses and their confidence intervals. In particular, the important fields are:
  - `shape`: this is the selected value for the hyper-parameter  $s$ .
  - `g`: the number of bins used.
  - `f`: the estimated node fitnesses.
  - `var_f`: the estimated variance of  $\eta_i$ .
  - `upper_f`: the estimated upper value of the interval of two standard deviations around  $\eta_i$ .
  - `lower_f`: the estimated lower value of the interval of two standard deviations around  $\eta_i$ .
  - `objective_value`: values of the objective function over iterations in the final run with the full data.
  - `diverge_zero`: logical value indicates whether the algorithm diverged in the final run with the full data.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**References**

1. Pham, T., Sheridan, P. & Shimodaira, H. (2016). Joint Estimation of Preferential Attachment and Node Fitness in Growing Complex Networks. *Scientific Reports* 6, Article number: 32558. ([doi:10.1038/srep32558](https://doi.org/10.1038/srep32558)).
2. Bianconi, G. & Barabási, A. (2001). Competition and multiscaling in evolving networks. *Europhys. Lett.*, 54, 436 ([doi:10.1209/epl/i2001002606](https://doi.org/10.1209/epl/i2001002606)).
3. Caldarelli, G., Capocci, A., De Los Rios, P. & Muñoz, M.A. (2002). Scale-Free Networks from Varying Vertex Intrinsic Fitness. *Phys. Rev. Lett.*, 89, 258702 ([doi:10.1103/PhysRevLett.89.258702](https://doi.org/10.1103/PhysRevLett.89.258702)).

**See Also**

See [get\\_statistics](#) for how to create summarized statistics needed in this function.

See [joint\\_estimate](#) for the method to jointly estimate the attachment function  $A_k$  and node fitnesses  $\eta_i$ .

**Examples**

```
## Not run:
library("PAFit")
set.seed(1)
# size of initial network = 100
# number of new nodes at each time-step = 100
```

```

# Ak = k; inverse variance of the distribution of node fitnesses = 10
net      <- generate_BB(N      = 1000 , m      = 50 ,
                      num_seed = 100  , multiple_node = 100,
                      s      = 10)

net_stats <- get_statistics(net)

# estimate node fitnesses in isolation, assuming Ak = k
result   <- only_F_estimate(net, net_stats)

# plot the estimated node fitnesses and true node fitnesses
plot(result, net_stats, true = net$fitness, plot = "true_f")

## End(Not run)

```

---

PAFit_oneshot	<i>Estimating the nonparametric preferential attachment function from one single snapshot.</i>
---------------	--

---

## Description

This function estimates the attachment function  $A_k$  from one snapshot.

## Usage

```

PAFit_oneshot(net_object,
              M      = 10,
              S      = 5,
              loop   = 5,
              G      = 1000)

```

## Arguments

net_object	an object of class PAFit_net that contains the network. Any time-step information, if available, will be ignored.
M	Integer. Number of simulated networks in each iteration. Default is 10.
S	Integer. Number of iterations inside each loop. Default is 5.
loop	Integer. Number of loops of the whole process. Default is 5.
G	Integer. Number of bins for the PA function. Default is 1000.

## Value

Outputs a PAFit\_result object.

## Author(s)

Thong Pham <thongphamthe@gmail.com>

## References

1. Pham, T., Sheridan, P. & Shimodaira, H. (2021). Non-parametric estimation of the preferential attachment function from one network snapshot. *Journal of Complex Networks* 9(5): cnab024. ([doi:10.1093/comnet/cnab024](https://doi.org/10.1093/comnet/cnab024)).

## Examples

```
## Not run:
library("PAFit")
net_1 <- generate_BA(N = 10000, alpha = 1) # true attachment exponent = 1.0
result_1 <- PAFit_oneshot(net_1)
print(result_1)

net_2 <- generate_BA(N = 10000, alpha = 0.5) # true attachment exponent = 0.5
result_2 <- PAFit_oneshot(net_2)
print(result_2)

## End(Not run)
```

---

```
plot.Full_PAFit_result
```

*Plotting the estimated attachment function and node fitness*

---

## Description

This function plots the estimated attachment function  $A_k$  and node fitness  $eta_i$ , together with additional information such as their confidence intervals or the estimated attachment exponent ( $\alpha$  when assuming  $A_k = k^\alpha$ ).

## Usage

```
## S3 method for class 'Full_PAFit_result'
plot(x,
      net_stat          ,
      true_f            = NULL , plot          = "A"          , plot_bin  = TRUE ,
      line              = FALSE , confidence   = TRUE          , high_deg_A = 1  ,
      high_deg_f        = 5    ,
      shade_point       = 0.5  , col_point     = "grey25"      , pch       = 16  ,
      shade_interval    = 0.5  , col_interval  = "lightsteelblue", label_x   = NULL ,
      label_y           = NULL ,
      max_A             = NULL , min_A         = NULL          , f_min     = NULL ,
      f_max             = NULL , plot_true_degree = FALSE ,
      ...)
```

**Arguments**

x	An object of class Full_PAFit_result, containing the estimated results from <code>only_A_estimate</code> , <code>only_F_estimate</code> or <code>joint_estimate</code> .
net_stat	An object of class PAFit_data, containing the summerized statistics.
true_f	Vector. Optional parameter for the true value of node fitnesses (only available in simulated datasets). If this parameter is specified and <code>plot == "true_f"</code> , a plot of estimated $\eta$ versus true $\eta$ is produced (after a suitable rescaling of the estimated $f$ ).
plot	String. Indicates which plot is produced. <ul style="list-style-type: none"> <li>• If "A" then PA function is plotted.</li> <li>• If "f" then the histogram of estimated fitness is plotted.</li> <li>• If "true_f" then estimated fitness and true fitness are plotted together (require supplement of true fitness).</li> </ul> Default value is "A".
plot_bin	Logical. If TRUE then only the center of each bin is plotted. Default is TRUE.
line	Logical. Indicates whether to plot the line fitted from the log-linear model or not. Default value is <i>TRUE</i> .
confidence	Logical. Indicates whether to plot the confidence intervals of $A_k$ and $eta_i$ or not. If <code>confidence == TRUE</code> , a 2-sigma confidence interval will be plotted at each $A_k$ and $eta_i$ .
high_deg_A	Integer. The estimated PA function is plotted starting from high_deg_A. Default value is 1.
high_deg_f	Integer. If <code>plot == "true_f"</code> , only nodes whose number of edges acquired is not less than high_deg_f are plotted. Default value is 5.
col_point	String. The name of the color of the points. Default value is "black".
shade_point	Numeric. Value between 0 and 1. This is the transparency level of the points. Default value is 0.5.
pch	Numeric. The plot symbol. Default value is 16.
shade_interval	Numeric. Value between 0 and 1. This is the transparency level of the confidence intervals. Default value is 0.5.
max_A	Numeric. Specify the maximum of the axis of PA.
min_A	Numeric. Specify the minimum of the axis of PA.
f_min	Numeric. Specify the minimum of the axis of fitness.
f_max	Numeric. Specify the maximum of the axis of fitness.
plot_true_degree	Logical. The degree of each node is plotted or not.
label_x	String. The label of x-axis.
label_y	String. The label of y-axis.
col_interval	String. The name of the color of the confidence intervals. Default value is "lightsteelblue".
...	Other arguments to pass to the underlying plotting function.

**Value**

Outputs the desired plot.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net      <- generate_BB(N      = 1000 , m      = 50 ,
                      num_seed = 100 , multiple_node = 100,
                      s      = 10)
net_stats <- get_statistics(net)
result   <- joint_estimate(net, net_stats)
#plot A
plot(result , net_stats , plot = "A")
true_A   <- c(1,result$estimate_result$center_k[-1])
lines(result$estimate_result$center_k + 1 , true_A , col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")
#plot true_f
plot(result, net_stats , net$fitness, plot = "true_f")

## End(Not run)
```

---

plot.PAFit\_net                    *Plot a PAFit\_net object*

---

**Description**

This function plots a PAFit\_net object. There are four options of plot to specify the type of plot.

The first two concern plotting the graph in \$graph of the PAFit\_net object. Option plot = "graph" plots the graph, while plot = "degree" plots the degree distribution. Option slice allows selection of the time-step at which the temporal graph is plotted.

The last two options concern plotting the PA function and node fitnesses (if they are not NULL).

**Usage**

```
## S3 method for class 'PAFit_net'
plot(x,
     plot = "graph" ,
     slice = length(unique(x$graph[,3])) - 1,
     ...)
```



**Arguments**

x	An object of class PAFit_net.
plot	String. Possible values are "graph", "degree", "PA", and "fit". Default value is "graph".
slice	Integer. Ignored when plot is not "graph" or "degree". Specifies the time-step at which the graph is plotted. Default value is the final time-step.
...	Other arguments to pass to the underlying plotting function.

**Value**

Outputs the desired plot.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>. When plot = "graph", the function uses [plot.network.default](#) in the **network** package.

**Examples**

```
library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50 , m = 10 , s = 10)
plot(net, plot = "graph")
plot(net, plot = "degree")
plot(net, plot = "PA")
plot(net, plot = "fit")
```

---

plot.PAFit\_result      *Plotting the estimated attachment function and node fitness of a PAFit\_result object*

---

**Description**

This function plots the estimated attachment function  $A_k$  and node fitness  $eta_i$ , together with additional information such as their confidence intervals or the estimated attachment exponent ( $\alpha$  when assuming  $A_k = k^\alpha$ ) of a PAFit\_result object. This object is stored in the field \$estimate\_result of a Full\_PAFit\_result object, which in turn is the returning value of [only\\_A\\_estimate](#), [only\\_F\\_estimate](#) or [joint\\_estimate](#).

**Usage**

```
## S3 method for class 'PAFit_result'
plot(x,
     net_stat      = NULL      ,
     true_f        = NULL      , plot      = "A"          , plot_bin = TRUE ,
     line          = FALSE     , confidence = TRUE        , high_deg_A = 1   ,
     high_deg_f    = 5         ,
```

```

shade_point = 0.5 , col_point = "grey25" , pch = 16 ,
shade_interval = 0.5 , col_interval = "lightsteelblue" , label_x = NULL ,
label_y = NULL ,
max_A = NULL , min_A = NULL , f_min = NULL ,
f_max = NULL , plot_true_degree = FALSE ,
...)
```

## Arguments

x	An object of class PAFit_result.
net_stat	An object of class PAFit_data, containing the summerized statistics.
true_f	Vector. Optional parameter for the true value of node fitnesses (only available in simulated datasets). If this parameter is specified and plot == "true_f", a plot of estimated $\eta$ versus true $\eta$ is produced (after a suitable rescaling of the estimated $f$ ).
plot	String. Indicates which plot is produced. <ul style="list-style-type: none"> <li>• If "A" then PA function is plotted.</li> <li>• If "f" then the histogram of estimated fitness is plotted.</li> <li>• If "true_f" then estimated fitness and true fitness are plotted together (require supplement of true fitness).</li> </ul> Default value is "A".
plot_bin	Logical. If TRUE then only the center of each bin is plotted. Default is TRUE.
line	Logical. Indicates whether to plot the line fitted from the log-linear model or not. Default value is <i>TRUE</i> .
confidence	Logical. Indicates whether to plot the confidence intervals of $A_k$ and $eta_i$ or not. If confidence == TRUE, a 2-sigma confidence interval will be plotted at each $A_k$ and $eta_i$ .
high_deg_A	Integer. The estimated PA function is plotted starting from high_deg_A. Default value is 1.
high_deg_f	Integer. If plot == "true_f", only nodes whose number of edges acquired is not less than high_deg_f are plotted. Default value is 5.
col_point	String. The name of the color of the points. Default value is "black".
shade_point	Numeric. Value between 0 and 1. This is the transparency level of the points. Default value is 0.5.
pch	Numeric. The plot symbol. Default value is 16.
shade_interval	Numeric. Value between 0 and 1. This is the transparency level of the confidence intervals. Default value is 0.5.
max_A	Numeric. Specify the maximum of the axis of PA.
min_A	Numeric. Specify the minimum of the axis of PA.
f_min	Numeric. Specify the minimum of the axis of fitness.
f_max	Numeric. Specify the maximum of the axis of fitness.
plot_true_degree	Logical. The degree of each node is plotted or not.

label_x	String. The label of x-axis.
label_y	String. The label of y-axis.
col_interval	String. The name of the color of the confidence intervals. Default value is "lightsteelblue".
...	Other arguments to pass to the underlying plotting function.

**Value**

Outputs the desired plot.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net      <- generate_BB(N      = 1000 , m      = 50 ,
                      num_seed = 100 , multiple_node = 100,
                      s      = 10)
net_stats <- get_statistics(net)
result    <- joint_estimate(net, net_stats)
#plot A
plot(result$estimate_result , net_stats , plot = "A")
true_A    <- c(1,result$estimate_result$center_k[-1])
lines(result$estimate_result$center_k + 1 , true_A , col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")
#plot true_f
plot(result, net_stats , net$fitness, plot = "true_f")

## End(Not run)
```

---

plot.PA\_result

*Plotting the estimated attachment function*


---

**Description**

This function plots the estimated attachment function from the corrected Newman's method or the Jeong's method. Its also plots additional information such as the estimated attachment exponent ( $\alpha$  when assuming  $A_k = k^\alpha$ ).

**Usage**

```
## S3 method for class 'PA_result'
plot(x,
      net_stat = NULL,
      plot_bin = TRUE,
      high_deg = 1,
      line = FALSE,
      col_point = "black",
      shade_point = 0.5,
      pch = 16,
      max_A = NULL,
      min_A = NULL,
      label_x = NULL,
      label_y = NULL,
      ...)
```

**Arguments**

x	An object of class <code>PA_result</code> , containing the estimated attachment function and the estimated attachment exponent from either <a href="#">Newman</a> or <a href="#">Jeong</a> functions.
net_stat	An object of class <code>PA_data</code> , containing the summerized statistics. This object is created from the function <a href="#">get_statistics</a> .
plot_bin	Logical. If TRUE then only the center of each bin is plotted. Default is TRUE.
high_deg	Integer. Specifies the starting degree from which $A_k$ is plotted. If this parameter is specified, the estimated attachment function is plotted from $k = \text{high\_deg}$
line	Logical. Indicates whether to plot the line fitted from the log-linear model or not. Default value is FALSE.
col_point	String. The name of the color of the points. Default value is "black".
shade_point	Numeric. Value between 0 and 1. This is the transparency level of the points. Default value is 0.5.
pch	Numeric. The plot symbol. Default value is 16.
max_A	Numeric. Specify the maximum of the horizontal axis.
min_A	Numeric. Specify the minimum of the horizontal axis.
label_x	String. The label of x-axis. If NULL, then "Degree k" is used.
label_y	String. The label of y-axis. If NULL, then "Attachment function" is used.
...	Other arguments to pass to the underlying plotting function.

**Value**

Outputs the desired plot.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```

library("PAFit")
net      <- generate_net(N = 1000 , m = 1 , mode = 1 , alpha = 1 , s = 0)
net_stats <- get_statistics(net)
result   <- Newman(net, net_stats)
# true function
true_A   <- result$center_k
# plot the estimated attachment function
plot(result , net_stats)
lines(result$center_k, true_A, col = "red") # true attachment function
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")

```

---

plot_contribution	<i>Plotting contributions calculated from the observed data and contributions calculated from simulated data</i>
-------------------	--

---

**Description**

This function extracts from a `Simulated_Data_From_Fitted_Model` object contributions of rich-get-richer and fit-get-richer effects calculated using simulated networks and plots these contributions versus the contributions calculated from the original observed network. See [joint\\_estimate](#) for a description of how the contributions are calculated.

**Usage**

```

plot_contribution(simulated_object,
                  original_result,
                  which_plot = "PA",
                  y_label = ifelse("PA" == which_plot,
                                   "Contribution of the rich-get-richer effect",
                                   "Contribution of the fit-get-richer effect"),
                  legend_pos_x = 0.75,
                  legend_pos_y = 0.9)

```

**Arguments**

simulated_object	an object of class <code>Simulated_Data_From_Fitted_Model</code> that contains simulated data.
original_result	an object of class <code>Full_PAFit_result</code> that contains the estimation results from the original observed data.
which_plot	String. "PA": plots contributions of rich-get-richer effect, "fit": plots contribution of fit-get-richer effect. Default is "PA".
y_label	String. The label for y-axis. Default is "Contribution of rich-get-richer effect".
legend_pos_x	Numeric. The horizontal position, between (0,1), of the legend. Default value is 0.75.

legend\_pos\_y    Numeric. The vertical position, between (0,1), of the legend. Default value is 0.9.

### Value

Output a plot.

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### References

1. Pham, T., Sheridan, P. & Shimodaira, H. (2015). PAFit: A Statistical Method for Measuring Preferential Attachment in Temporal Complex Networks. PLoS ONE 10(9): e0137796. ([doi:10.1371/journal.pone.0137796](https://doi.org/10.1371/journal.pone.0137796)).
2. Pham, T., Sheridan, P. & Shimodaira, H. (2016). Joint Estimation of Preferential Attachment and Node Fitness in Growing Complex Networks. Scientific Reports 6, Article number: 32558. ([doi:10.1038/srep32558](https://doi.org/10.1038/srep32558)).
3. Pham, T., Sheridan, P. & Shimodaira, H. (2020). PAFit: An R Package for the Non-Parametric Estimation of Preferential Attachment and Node Fitness in Temporal Complex Networks. Journal of Statistical Software 92 (3). ([doi:10.18637/jss.v092.i03](https://doi.org/10.18637/jss.v092.i03)).
4. Inoue, M., Pham, T. & Shimodaira, H. (2020). Joint Estimation of Non-parametric Transitivity and Preferential Attachment Functions in Scientific Co-authorship Networks. Journal of Informetrics 14(3). ([doi:10.1016/j.joi.2020.101042](https://doi.org/10.1016/j.joi.2020.101042)).

### See Also

[joint\\_estimate](#), [plot\\_contribution](#)

### Examples

```
## Not run:

library("PAFit")
net_object      <- generate_net(N = 500, m = 10, s = 10, alpha = 0.5)
net_stat        <- get_statistics(net_object)
result          <- joint_estimate(net_object, net_stat)
simulated_data <- generate_simulated_data_from_estimated_model(net_object, net_stat, result)
plot_contribution(simulated_data, result, which_plot = "PA")
plot_contribution(simulated_data, result, which_plot = "fit")

## End(Not run)
```

---

`print.CV_Data`*Printing simple information of the cross-validation data*

---

### Description

This function prints simple information of the cross-validation data stored in a CV\_Data object. This object is the field `$cv_data` of a `Full_PAFit_result` object, which in turn is the returning value of `only_A_estimate`, `only_F_estimate` or `joint_estimate`.

### Usage

```
## S3 method for class 'CV_Data'  
print(x,...)
```

### Arguments

<code>x</code>	An object of class <code>CV_Data</code> .
<code>...</code>	Other arguments to pass.

### Value

Prints simple information of the cross-validation data.

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### Examples

```
## Since the runtime is long, we do not let this example run on CRAN  
## Not run:  
library("PAFit")  
set.seed(1)  
# a network from Bianconi-Barabasi model  
net      <- generate_BB(N      = 1000 , m      = 50 ,  
                       num_seed = 100 , multiple_node = 100,  
                       s      = 10)  
net_stats <- get_statistics(net)  
result    <- joint_estimate(net, net_stats)  
print(result$cv_data)  
  
## End(Not run)
```

---

print.CV\_Result      *Printing simple information of the cross-validation result*

---

### Description

This function prints simple information of the cross-validation result stored in a CV\_Result object. This object is the field \$cv\_result of a Full\_PAFit\_result object, which in turn is the returning value of [only\\_A\\_estimate](#), [only\\_F\\_estimate](#) or [joint\\_estimate](#).

### Usage

```
## S3 method for class 'CV_Result'  
print(x,...)
```

### Arguments

x                    An object of class CV\_Result.  
...                  Other arguments to pass.

### Value

Prints simple information of the cross-validation result.

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### Examples

```
## Since the runtime is long, we do not let this example run on CRAN  
## Not run:  
library("PAFit")  
set.seed(1)  
# a network from Bianconi-Barabasi model  
net        <- generate_BB(N            = 1000 , m            = 50 ,  
                                     num_seed = 100 , multiple_node = 100,  
                                     s        = 10)  
net_stats <- get_statistics(net)  
result    <- joint_estimate(net, net_stats)  
print(result$cv_result)  
  
## End(Not run)
```



---

```
print.Full_PAFit_result
```

*printing information on the estimation result*

---

## Description

This function outputs simple information of the estimation result.

## Usage

```
## S3 method for class 'Full_PAFit_result'  
print(x,...)
```

## Arguments

`x` An object of class `Full_PAFit_result`, containing the estimated results from `only_A_estimate`, `only_F_estimate` or `joint_estimate`.

`...` Other arguments to pass.

## Value

Outputs summary information on the estimation result.

## Author(s)

Thong Pham <thongphamthe@gmail.com>

## Examples

```
## Since the runtime is long, we do not let this example run on CRAN  
## Not run:  
library("PAFit")  
set.seed(1)  
# a network from Bianconi-Barabasi model  
net <- generate_BB(N = 1000 , m = 50 ,  
                  num_seed = 100 , multiple_node = 100,  
                  s = 10)  
net_stats <- get_statistics(net)  
result <- joint_estimate(net, net_stats)  
print(result)  
  
## End(Not run)
```

---

print.PAFit_data	<i>Printing simple information on the statistics of the network stored in a PAFit_data object</i>
------------------	---

---

## Description

This function prints simple information of the statistics stored in a PAFit\_data object. This object is the returning value of [get\\_statistics](#).

## Usage

```
## S3 method for class 'PAFit_data'  
print(x,...)
```

## Arguments

x	An object of class PAFit_data.
...	Other arguments to pass.

## Value

Prints simple information of the network statistics.

## Author(s)

Thong Pham <thongphamthe@gmail.com>

## Examples

```
## Since the runtime is long, we do not let this example run on CRAN  
## Not run:  
library("PAFit")  
set.seed(1)  
# a network from Bianconi-Barabasi model  
net <- generate_BB(N = 1000 , m = 50 ,  
                  num_seed = 100 , multiple_node = 100,  
                  s = 10)  
net_stats <- get_statistics(net)  
print(net_stats)  
  
## End(Not run)
```

---

```
print.PAFit_net      Printing simple information of a PAFit_net object
```

---

**Description**

This function outputs simple information of a PAFit\_net object.

**Usage**

```
## S3 method for class 'PAFit_net'
print(x,
      ...)
```

**Arguments**

```
x          An object of class PAFit_net.
...        Other arguments to pass.
```

**Value**

Outputs simple information of the network.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```
library("PAFit")
# a network from Bianconi-Barabasi model
net      <- generate_BB(N = 50 , m = 10 , s = 10)
print(net)
```

---

```
print.PAFit_result  printing information on the estimation result stored in a
PAFit_result object
```

---

**Description**

This function outputs simple information of the estimation result stored in a PAFit\_result object. This object is stored in the field \$estimate\_result of a Full\_PAFit\_result object, which in turn is the returning value of [only\\_A\\_estimate](#), [only\\_F\\_estimate](#) or [joint\\_estimate](#).

**Usage**

```
## S3 method for class 'PAFit_result'
print(x,...)
```

**Arguments**

x                    An object of class PAFit\_result.  
 ...                   Other arguments to pass.

**Value**

Outputs summary information on the estimation result.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net      <- generate_BB(N      = 1000 , m      = 50 ,
                      num_seed = 100  , multiple_node = 100,
                      s      = 10)
net_stats <- get_statistics(net)
result    <- joint_estimate(net, net_stats)
print(result$estimate_result)

## End(Not run)
```

---

print.PA\_result            *Printing information of the estimated attachment function*

---

**Description**

This function outputs simple information of the estimated attachment function from the corrected Newman's method or the Jeong's method.

**Usage**

```
## S3 method for class 'PA_result'
print(x,
      ...)
```

**Arguments**

x                    An object of class PA\_result, containing the estimated attachment function and the estimated attachment exponent from either [Newman](#) or [Jeong](#) functions.  
 ...                   Additional parameters to pass.

**Value**

Simple information of the estimated attachment function.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```
library("PAFit")
net      <- generate_net(N = 1000 , m = 1 , mode = 1 , alpha = 1 , s = 0)
net_stats <- get_statistics(net)
result   <- Newman(net, net_stats)
print(result)
```

---

summary.CV\_Data

*Printing summary information of the cross-validation data*

---

**Description**

This function outputs summary information of the cross-validation data stored in a CV\_Data object. This object is the field \$cv\_data of a Full\_PAFit\_result object, which in turn is the returning value of [only\\_A\\_estimate](#), [only\\_F\\_estimate](#) or [joint\\_estimate](#).

**Usage**

```
## S3 method for class 'CV_Data'
summary(object,...)
```

**Arguments**

object            An object of class CV\_Data.  
...                Other arguments to pass.

**Value**

Outputs summary information of the cross-validation data.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

## Examples

```
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net      <- generate_BB(N      = 1000 , m      = 50 ,
                       num_seed = 100 , multiple_node = 100,
                       s      = 10)

net_stats <- get_statistics(net)
result    <- joint_estimate(net, net_stats)
summary(result$cv_data)

## End(Not run)
```

---

summary.CV_Result	<i>Output summary information of the cross-validation result</i>
-------------------	--

---

## Description

This function outputs summary information of the cross-validation result stored in a `CV_Result` object. This object is the field `$cv_result` of a `Full_PAFit_result` object, which in turn is the returning value of [only\\_A\\_estimate](#), [only\\_F\\_estimate](#) or [joint\\_estimate](#).

## Usage

```
## S3 method for class 'CV_Result'
summary(object,...)
```

## Arguments

object	An object of class <code>CV_Result</code> .
...	Other arguments to pass.

## Value

Outputs summary information of the cross-validation result.

## Author(s)

Thong Pham <thongphamthe@gmail.com>

## Examples

```
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net      <- generate_BB(N      = 1000 , m      = 50 ,
                       num_seed = 100 , multiple_node = 100,
                       s      = 10)

net_stats <- get_statistics(net)
result    <- joint_estimate(net, net_stats)
summary(result$cv_result)

## End(Not run)
```

---

summary.Full\_PAFit\_result

*Summary information on the estimation result*

---

## Description

This function outputs a summary on the estimation result.

## Usage

```
## S3 method for class 'Full_PAFit_result'
summary(object,...)
```

## Arguments

object	An object of class Full_PAFit_result, containing the estimated results from <a href="#">only_A_estimate</a> , <a href="#">only_F_estimate</a> or <a href="#">joint_estimate</a> .
...	Other arguments to pass.

## Value

Outputs summary information on the estimation result.

## Author(s)

Thong Pham <thongphamthe@gmail.com>

## Examples

```
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net      <- generate_BB(N      = 1000 , m      = 50 ,
                       num_seed = 100 , multiple_node = 100,
                       s      = 10)

net_stats <- get_statistics(net)
result    <- joint_estimate(net, net_stats)
summary(result)

## End(Not run)
```

---

summary.PAFit_data	<i>Output summary information on the statistics of the network stored in a PAFit_data object</i>
--------------------	--

---

## Description

This function outputs summary information of the statistics stored in a PAFit\_data object. This object is the returning value of [get\\_statistics](#).

## Usage

```
## S3 method for class 'PAFit_data'
summary(object,...)
```

## Arguments

object	An object of class PAFit_data.
...	Other arguments to pass.

## Value

Outputs summary information of the network statistics.

## Author(s)

Thong Pham <thongphamthe@gmail.com>



**Examples**

```
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net      <- generate_BB(N      = 1000 , m      = 50 ,
                       num_seed = 100  , multiple_node = 100,
                       s      = 10)

net_stats <- get_statistics(net)
summary(net_stats)

## End(Not run)
```

---

```
summary.PAFit_net      Summary information of a PAFit_net object
```

---

**Description**

This function outputs summary information of a PAFit\_net object.

**Usage**

```
## S3 method for class 'PAFit_net'
summary(object,
         ...)
```

**Arguments**

```
object      An object of class PAFit_net.
...         Other arguments to pass.
```

**Value**

Outputs summary information of the network.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```
library("PAFit")
# a network from Bianconi-Barabasi model
net      <- generate_BB(N = 50 , m = 10 , s = 10)
summary(net)
```

---

summary.PAFit\_result *Output summary information on the estimation result stored in a PAFit\_result object*

---

## Description

This function outputs summary information of the estimation result stored in a PAFit\_result object. This object is stored in the field \$estimate\_result of a Full\_PAFit\_result object, which in turn is the returning value of [only\\_A\\_estimate](#), [only\\_F\\_estimate](#) or [joint\\_estimate](#).

## Usage

```
## S3 method for class 'PAFit_result'
summary(object,...)
```

## Arguments

object            An object of class PAFit\_result.  
 ...              Other arguments to pass.

## Value

Outputs summary information on the estimation result.

## Author(s)

Thong Pham <thongphamthe@gmail.com>

## Examples

```
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net      <- generate_BB(N      = 1000 , m      = 50 ,
                      num_seed = 100  , multiple_node = 100,
                      s      = 10)
net_stats <- get_statistics(net)
result   <- joint_estimate(net, net_stats)
summary(result$estimate_result)

## End(Not run)
```

---

summary.PA_result	<i>Summary of the estimated attachment function</i>
-------------------	---

---

### Description

This function outputs summary information of the estimated attachment function from the corrected Newman's method or the Jeong's method.

### Usage

```
## S3 method for class 'PA_result'  
summary(object,  
        ...)
```

### Arguments

object	An object of class <code>PA_result</code> , containing the estimated attachment function and the estimated attachment exponent from either <a href="#">Newman</a> or <a href="#">Jeong</a> functions.
...	Additional parameters to pass.

### Value

Summary information of the estimated attachment function.

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### Examples

```
library("PAFit")  
net      <- generate_net(N = 1000 , m = 1 , mode = 1 , alpha = 1 , s = 0)  
net_stats <- get_statistics(net)  
result   <- Newman(net, net_stats)  
summary(result)
```

---

test_linear_PA	<i>Fitting various distributions to a degree vector</i>
----------------	---

---

## Description

This function implements the method in Handcock and Jones (2004) to fit various distributions to a degree vector. The implemented distributions are Yule, Waring, Poisson, geometric and negative binomial. The Yule and Waring distributions correspond to a preferential attachment situation. In particular, the two distributions correspond to the case of  $A_k = k$  for  $k \geq 1$  and  $\eta_i = 1$  for all  $i$  (note that, the number of new edges and new nodes at each time-step are implicitly assumed to be 1).

Thus, if the best fitted distribution, which is chosen by either the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC), is NOT Yule or Waring, then the case of  $A_k = k$  for  $k \geq 1$  and  $\eta_i = 1$  for all  $i$  is NOT consistent with the observed degree vector.

The method allows the low-tail probabilities to NOT follow the parametric distribution, i.e.,  $P(K = k) = \pi_k$  for all  $k \leq k_{min}$  and  $P(K = k) = f(k, \theta)$  for all  $k > k_{min}$ . Here  $k_{min}$  is the degree threshold above which the parametric distribution holds,  $\pi_k$  are probabilities of the low-tail,  $f(\cdot, \theta)$  is the parametric distribution with parameter vector  $\theta$ .

For fixed  $k_{min}$  and  $f$ ,  $\pi_k$  and  $\theta$  can be estimated by Maximum Likelihood Estimation. We can choose the best  $k_{min}$  for each  $f$  by comparing the AIC (or BIC). More details can be founded in Handcock and Jones (2004).

## Usage

```
test_linear_PA(degree_vector)
```

## Arguments

degree\_vector a degree vector

## Value

Outputs a Linear\_PA\_test\_result object which contains the fitting of five distributions to the degree vector: Yule (yule), Waring (waring), Poisson (pois), geometric (geom) and negative binomial (nb). In particular, for each distribution, the AIC and BIC are calculated for each  $k_{min}$ .

## Author(s)

Thong Pham <thongphamthe@gmail.com>

## References

1. Handcock MS, Jones JH (2004). "Likelihood-based inference for stochastic models of sexual network formation." *Theoretical Population Biology*, 65(4), 413 – 422. ISSN 0040-5809. doi:10.1016/j.tpb.2003.09.006. *Demography in the 21st Century*, <https://www.sciencedirect.com/science/article/abs/pii/S0040580904000310>.

## Examples

```
## Not run:
library("PAFit")
set.seed(1)
net <- generate_BA(n = 1000)
```

```
stats <- get_statistics(net, only_PA = TRUE)
u      <- test_linear_PA(stats$final_deg)
print(u)

## End(Not run)
```

---

**to\_igraph***Convert a PAFit\_net object to an igraph object*

---

### Description

This function converts a PAFit\_net object to an igraph object (of package **igraph**).

### Usage

```
to_igraph(net_object)
```

### Arguments

**net\_object**      An object of class PAFit\_net.

### Value

The function returns an igraph object.

### Author(s)

Thong Pham <thongphamthe@gmail.com>

### Examples

```
library("PAFit")
# a network from Bianconi-Barabasi model
net      <- generate_BB(N = 50 , m = 10 , s = 10)
igraph_graph <- to_igraph(net)
```

---

to_networkDynamic	<i>Convert a PAFit_net object to a networkDynamic object</i>
-------------------	--

---

**Description**

This function converts a PAFit\_net object to a networkDynamic object (of package **networkDynamic**).

**Usage**

```
to_networkDynamic(net_object)
```

**Arguments**

net\_object      An object of class PAFit\_net.

**Value**

The function returns a networkDynamic object.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```
library("PAFit")
# a network from Bianconi-Barabasi model
net      <- generate_BB(N = 50 , m = 10 , s = 10)
nD_graph <- to_networkDynamic(net)
```

# Index

- \* **Barabasi-Albert model**
  - generate\_BA, 8
  - generate\_net, 14
  - PAFit-package, 3
- \* **Bianconi-Barabasi model**
  - generate\_BB, 10
  - generate\_net, 14
  - PAFit-package, 3
- \* **Corrected Newman's method**
  - PAFit-package, 3
- \* **ER model**
  - generate\_ER, 11
- \* **Jeong's method**
  - PAFit-package, 3
- \* **attachment function**
  - generate\_simulated\_data\_from\_estimated\_model, 17
  - Jeong, 24
  - joint\_estimate, 26
  - Newman, 30
  - only\_A\_estimate, 31
  - PAFit-package, 3
  - PAFit\_oneshot, 37
  - plot\_contribution, 45
- \* **fitness model**
  - generate\_fit\_only, 13
  - generate\_net, 14
  - only\_F\_estimate, 35
  - PAFit-package, 3
- \* **fitness**
  - generate\_simulated\_data\_from\_estimated\_model, 17
  - joint\_estimate, 26
  - plot\_contribution, 45
- \* **fitting degree distributions**
  - test\_linear\_PA, 59
- \* **linear preferential attachment**
  - test\_linear\_PA, 59
- \* **preferential attachment**
  - generate\_simulated\_data\_from\_estimated\_model, 17
  - Jeong, 24
  - joint\_estimate, 26
  - Newman, 30
  - only\_A\_estimate, 31
  - PAFit-package, 3
  - PAFit\_oneshot, 37
  - plot\_contribution, 45
- \* **temporal complex networks**
  - get\_statistics, 19
  - PAFit-package, 3
- as.PAFit\_net, 5, 19, 21
- coauthor.author\_id (Coauthorship network of scientists in the field of complex networks), 6
- coauthor.net, 3
- coauthor.net (Coauthorship network of scientists in the field of complex networks), 6
- coauthor.truetime (Coauthorship network of scientists in the field of complex networks), 6
- Coauthorship network of scientists in the field of complex networks, 6
- ComplexNetCoauthor (Coauthorship network of scientists in the field of complex networks), 6
- from\_igraph, 7, 19, 21
- from\_networkDynamic, 8, 19, 21
- generate\_BA, 3, 8, 11, 12, 14, 17
- generate\_BB, 3, 9, 10, 12, 14, 17
- generate\_ER, 3, 9, 11, 11, 14, 17
- generate\_fit\_only, 3, 9, 11, 12, 13, 17
- generate\_net, 3, 8, 9, 11, 12, 14, 14

`generate_simulated_data_from_estimated_model`,  
17

`get_statistics`, 3, 4, 9, 11, 12, 14, 17, 18,  
19, 24–26, 28, 30–33, 35, 36, 44, 50,  
56

`graph_from_file`, 19, 21, 22

`graph_to_file`, 22, 23

Jeong, 4, 21, 24, 28, 31, 33, 44, 52, 59

`joint_estimate`, 4, 18, 21, 26, 36, 39, 41,  
45–49, 51, 53–55, 58

Newman, 4, 21, 25, 28, 30, 33, 44, 52, 59

`only_A_estimate`, 4, 21, 25, 28, 31, 31, 39,  
41, 47–49, 51, 53–55, 58

`only_F_estimate`, 4, 21, 28, 35, 39, 41,  
47–49, 51, 53–55, 58

PAFit (PAFit-package), 3

PAFit-package, 3

`PAFit_data` (`get_statistics`), 19

`PAFit_oneshot`, 4, 37

`plot.Full_PAFit_result`, 38

`plot.network.default`, 41

`plot.PA_result`, 43

`plot.PAFit_net`, 40

`plot.PAFit_result`, 41

`plot_contribution`, 18, 45, 46

`print.CV_Data`, 47

`print.CV_Result`, 48

`print.Full_PAFit_result`, 49

`print.PA_result`, 52

`print.PAFit_data`, 50

`print.PAFit_net`, 51

`print.PAFit_result`, 51

`summary.CV_Data`, 53

`summary.CV_Result`, 54

`summary.Full_PAFit_result`, 55

`summary.PA_result`, 59

`summary.PAFit_data`, 56

`summary.PAFit_net`, 57

`summary.PAFit_result`, 58

`test_linear_PA`, 59

`to_igraph`, 61

`to_networkDynamic`, 62