```
> library(knitr)
> # set global chunk options
> knitr::opts_chunk$set(fig.path='figure/Vignette-', fig.align='center', fig.show='hold
+                       fig.width='\\linewidth',
+                       out.width='\\linewidth')
> options(formatR.arrow=TRUE,width=90)
```

Dynamic Network Regression Using R Package dnr

Abhirup Mallik

July 25, 2018

R package 'dnr' enables the user to fit dynamic network regression models for time variate network data available mostly in social sciences or social network analysis using the methodology described in [1]. In this document, we demonstrate the process of building a model to fit a dynamic network data set and using that model for prediction.

1 Analysis of Beach data

First, we consider the beach data for our demo.

```
> suppressMessages(library(dnr))
> data(beach)
> ## get the number of time points
> length(beach)

[1] 31

> ## network size (that allows NA)
> network.size.1 <- function(x){
+   if(!network::is.network(x)){
+     if(is.na(x)) return(0)
+   } else return(network::network.size(x))
+ }
> ## get the size of networks at each time point
> sapply(beach, network.size.1)

828 829 830 831 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918
 11  14  23  22  13   6  16  21  12  24  37  10   9  14  10  12  24  21  12  11  15  16
919 920 921 922 923 924 925 926 927
 10  28   0   8  10   3  10  14  34
```

The beach data is a rapidly changing data set with possible periodic effects. We visualize the adjacency matrix from four time points of the data.

```
> par(mfrow = c(2,2))
> binaryPlot(beach[[1]][, ], title = "Time point 1")
> binaryPlot(beach[[10]][, ], title = "Time point 10")
> binaryPlot(beach[[20]][, ], title = "Time point 20")
> binaryPlot(beach[[31]][, ], title = "Time point 31")
```

For vertex model, we define our own term dictionary. We use the similar approach as the edge model for specifying the time dependence of the terms using a matrix of lag terms.

Term	Index
degree (Freeman)	1
in degree	2
out degree	3
Eigen centrality	4
Between centrality	5
Info centrality	6
Closeness centrality	7
Log K cycle	8
Log size	9

Table 1: Index of the terms for specifying the vertex model

1.1 Model Fitting

We first try to build the model for vertex regression. We consider a maximum lag of 3. We need to specify the lag structure using a binary vector of size 3. We also need to specify the dependence on the vertex parameters up to 3 lags. There are 9 vertex parameters available in the current version of the library. We use a binary matrix of size 3×9 for specifying the lag dependence of the parameters.

```
> nvertexstats <- 9
> maxLag = 3
> VertexLag = rep(1, maxLag)
> VertexLagMatrix1 <- matrix(1, maxLag, nvertexstats)
> VertexLagMatrix1
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1    1    1    1    1    1    1    1    1
[2,]    1    1    1    1    1    1    1    1    1
[3,]    1    1    1    1    1    1    1    1    1
```

As for this data set there is expected seasonal effect, for example weekends would have different effect than weekdays, we would like to model that using a time variate intercept parameter. We write a function to extract the day information from the data.

```
> getWeekend <- function(z){
+     weekends <- c("Saturday", "Sunday")
+     if(!network::is.network(z)){
+         if(is.na(z)) return(NA)
+     } else {
+          zDay <- get.network.attribute(z, attrname = "day")
+          out <- ifelse(zDay %in% weekends, 1, 0)
+          return(out)
+     }
+ }
> ## for(i in 1:31) print(getWeekend(beach[[i]]))
> ## generate a vector of network level exogenous variable
> dayClass <- numeric(length(beach))
> for(i in seq_along(dayClass)) {
+     dayClass[i] <- getWeekend(beach[[i]])
+ }
```

We then use the function paramVertexOnly() to fit the model specified above to the beach data. Most of the options are kept at their default value. For a detail description of the model specification, please refer to the help pages of the function. We use the default 'bayesGLM' option for the logistic regression. We print the model object, which is an object from arm package, with its own summary method.

```
> out <- paramVertexOnly(InputNetwork = beach,
+                        maxLag = 3,
+                        VertexStatsvec = rep(1, nvertexstats),
+                        VertexLag = rep(1, maxLag),
+                        VertexLagMatrix = VertexLagMatrix1,
+                        dayClass = dayClass)
> summary(out$VertexFit$fit)

Call:
arm::bayesglm(formula = y ~ . - 1, family = binomial(link = "logit"),
    data = XYdata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0506  -1.1774  -1.0569  -0.6421   2.5063

Coefficients:
                        Estimate Std. Error z value Pr(>|z|)
lag1                    1.028333   0.769727   1.336  0.18156
lag2                    1.572503   0.892916   1.761  0.07822 .
lag3                    0.843973   0.764983   1.103  0.26992
Day                    -0.852764   0.091841  -9.285  < 2e-16 ***
DegreeLag1.             0.021577   0.235425   0.092  0.92698
```

```
InDegreeLag1.              0.043153   0.470850   0.092  0.92698
OutDegreeLag1.             0.043153   0.470850   0.092  0.92698
EigenCentralityLag1.       0.493574   0.524196   0.942  0.34641
BetweenCentralityLag1.    -0.006343   0.005935  -1.069  0.28520
InfoCentralityLag1.        0.204742   0.141982   1.442  0.14929
CloseCentralityLag1.       2.724521   0.842090   3.235  0.00121 **
LogCycleLag1.             -0.133896   0.151723  -0.883  0.37750
LogSizeLag1.              -0.572636   0.275158  -2.081  0.03742 *
DegreeLag2.               -0.002111   0.233643  -0.009  0.99279
InDegreeLag2.             -0.004221   0.467285  -0.009  0.99279
OutDegreeLag2.            -0.004221   0.467285  -0.009  0.99279
EigenCentralityLag2.       1.130413   0.578243   1.955  0.05059 .
BetweenCentralityLag2.     0.005331   0.005433   0.981  0.32647
InfoCentralityLag2.       -0.041003   0.157030  -0.261  0.79400
CloseCentralityLag2.       0.954564   0.813966   1.173  0.24090
LogCycleLag2.              0.013217   0.161119   0.082  0.93462
LogSizeLag2.              -0.992356   0.324498  -3.058  0.00223 **
DegreeLag3.               -0.002304   0.234386  -0.010  0.99216
InDegreeLag3.             -0.004608   0.468772  -0.010  0.99216
OutDegreeLag3.            -0.004608   0.468772  -0.010  0.99216
EigenCentralityLag3.       1.032205   0.569196   1.813  0.06976 .
BetweenCentralityLag3.     0.006441   0.005657   1.139  0.25490
InfoCentralityLag3.        0.457919   0.168657   2.715  0.00663 **
CloseCentralityLag3.       0.511478   0.854881   0.598  0.54964
LogCycleLag3.             -0.247933   0.167125  -1.484  0.13794
LogSizeLag3.              -0.755771   0.279837  -2.701  0.00692 **
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3555.8  on 2565  degrees of freedom
Residual deviance: 3200.3  on 2534  degrees of freedom
AIC: 3262.3

Number of Fisher Scoring iterations: 8
```

As we can see the model is hardly parsimonious. So, we decided to remove the terms that were not significant. We report the result of the refitted model.

```
> VertexLagMatrix <- matrix(0, maxLag, nvertexstats)
> VertexLagMatrix[, c(4, 7)] <- 1
> VertexLagMatrix[c(2,3),7] <- 0
> VertexLagMatrix
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    0    0    0    1    0    0    1    0    0
```

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

> out <- paramVertexOnly(InputNetwork = beach,
+                         maxLag = 3,
+                         VertexStatsvec = rep(1, nvertexstats),
+                         VertexLag = rep(1, maxLag),
+                         VertexLagMatrix = VertexLagMatrix,
+                         dayClass = dayClass)
> summary(out$VertexFit$fit)

Call:
arm::bayesglm(formula = y ~ . - 1, family = binomial(link = "logit"),
    data = XYdata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9023  -1.1774  -1.0664  -0.7149   2.6037

Coefficients:
                     Estimate Std. Error z value Pr(>|z|)
lag1                 -0.49443    0.17426  -2.837 0.004549 **
lag2                 -1.23852    0.19443  -6.370 1.89e-10 ***
lag3                 -1.23395    0.19007  -6.492 8.46e-11 ***
Day                  -0.83117    0.09079  -9.155  < 2e-16 ***
EigenCentralityLag1.  1.20172    0.28613   4.200 2.67e-05 ***
CloseCentralityLag1.  2.53228    0.73493   3.446 0.000570 ***
EigenCentralityLag2.  1.30669    0.31145   4.196 2.72e-05 ***
EigenCentralityLag3.  1.18579    0.31646   3.747 0.000179 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3555.8  on 2565  degrees of freedom
Residual deviance: 3268.0  on 2557  degrees of freedom
AIC: 3284

Number of Fisher Scoring iterations: 4
```

Now, we have a model with mostly significant parameters, so we select this model for vertex generation.

As the edge model and vertex model are separable, we can expect this model to work for the joint model as well. We use the function paramVertex() for fitting the joint vertex-edge model to the beach data.

```
> out <- paramVertex(InputNetwork = beach,
+                    maxLag = 3,
+                    VertexStatsvec = rep(1, nvertexstats),
+                    VertexModelGroup = "regular",
+                    VertexLag = rep(1, maxLag),
+                    VertexLagMatrix = VertexLagMatrix,
+                    dayClass = dayClass,
+                    EdgeModelTerms = NA,
+                    EdgeModelFormula = NA,
+                    EdgeGroup = NA,
+                    EdgeIntercept = c("edges"),
+                    EdgeNetparam = c("logSize"),
+                    EdgeExvar = NA,
+                    EdgeLag = c(1, 1, 0),
+                    paramout = TRUE)
> summary(out$VertexFit$fit)

Call:
arm::bayesglm(formula = y ~ . - 1, family = binomial(link = "logit"),
    data = XYdata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9397  -1.1774  -1.0793  -0.6249   3.6880

Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
lag1                -1.69546    0.33745  -5.024 5.05e-07 ***
lag2                -3.20981    0.59823  -5.365 8.07e-08 ***
lag3                -2.79034    0.47850  -5.831 5.49e-09 ***
Day                 -0.79949    0.09136  -8.751  < 2e-16 ***
attribLag1           1.44810    0.34010   4.258 2.06e-05 ***
attribLag2           2.23119    0.60271   3.702 0.000214 ***
attribLag3           1.79800    0.47927   3.752 0.000176 ***
EigenCentralityLag1  1.07029    0.29102   3.678 0.000235 ***
CloseCentralityLag1  2.37304    0.74049   3.205 0.001352 **
EigenCentralityLag2  1.04992    0.31895   3.292 0.000996 ***
EigenCentralityLag3  0.97343    0.32232   3.020 0.002527 **
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3555.8  on 2565  degrees of freedom
Residual deviance: 3185.3  on 2554  degrees of freedom
```

```
AIC: 3207.3

Number of Fisher Scoring iterations: 9

> summary(out$EdgeFit$fit)

Call:
arm::bayesglm(formula = y ~ . - 1, family = binomial(link = "logit"),
    data = XYdata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.3150  -0.3897  -0.3260  -0.3001   2.5369

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
edges          -0.26210    0.50209  -0.522    0.602
logCurrNetSize -0.72743    0.13676  -5.319 1.04e-07 ***
dayEffect       0.56833    0.06285   9.043  < 2e-16 ***
lag1            0.60923    0.14263   4.271 1.94e-05 ***
lag2            4.37638    0.10029  43.638  < 2e-16 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 29588  on 21343  degrees of freedom
Residual deviance: 10073  on 21338  degrees of freedom
AIC: 10083

Number of Fisher Scoring iterations: 5
```

The edge model parameters are specified using 'EdgeIntercept' term, as we are using an intercept only model. For this example, we have tried using time variate parameters, but finally decided on the intercept only model. The term 'EdgeNetParam' indicates the network level attribute. Currently the only attribute supported here is 'logSize', which is log of the network size at the present time point. The binary vector 'EdgeLag' indicates the lag dependence of the edges. The terms 'EdgeModelTerms' and 'EdgeModelFormula' has not been used for this example.

1.2 Prediction for Beach Data

As we have finalized on a model for the beach data, we can use this model to predict the future networks up to any arbitrary number of time points. As long as we do not run into the problems of degeneracy, the simulation method should be able to generate networks with this model.

```
> suppressWarnings(simResult <- engineVertex(InputNetwork = beach,
+                             numSim = 3,
+                             maxLag = 3,
+                             VertexStatsvec = rep(1, nvertexstats),
+                             VertexModelGroup = "regular",
+                             VertexAttLag = rep(1, maxLag),
+                             VertexLag = rep(1, maxLag),
+                             VertexLagMatrix = VertexLagMatrix,
+                             dayClassObserved = dayClass,
+                             dayClassFuture = c(1, 0, 0, 0, 0),
+                             EdgeModelTerms = NA,
+                             EdgeModelFormula = NA,
+                             EdgeGroup = NA,
+                             EdgeIntercept = c("edges"),
+                             EdgeNetparam = c("logSize"),
+                             EdgeExvar = NA,
+                             EdgeLag = c(0, 1, 0),
+                             paramout = TRUE
+                             ))
```

```
[1] 1
[1] 2
[1] 3
```

```
> par(mfrow = c(2,2))
> binaryPlot(beach[[31]][, ], title = "Time point 31")
> binaryPlot(simResult$SimNetwork[[1]][, ], title = "Time point 32 (simulated)")
> binaryPlot(simResult$SimNetwork[[2]][, ], title = "Time point 33 (simulated)")
> binaryPlot(simResult$SimNetwork[[3]][, ], title = "Time point 34 (simulated)")
```

2 Model for Fixed Vertex Case

Even though fixed vertex case can be considered as a special case of dynamic vertex-edge case, it is preferred that the fixed vertex case is handled in a simpler way. We have provided separate functions for this case, that we will demonstrate using the blog data set.

```
> data(rdNets)
> length(rdNets)
```

```
[1] 484
```

```
> rdNets[[1]]
```

```
 Network attributes:
  vertices = 47
```

```
  directed = TRUE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 182
    missing edges= 0
    non-missing edges= 182

 Vertex attribute names:
    dnc rnc vertex.names

 Edge attribute names:
    frequency
```

```
> plot(rdNets[[1]])
```

We use the function paramest() to fit the edge model to the blog data. The function accepts ERGM style model formulas, however we require that the terms are expanded by the user. This is required to construct the lag dependence binary matrix. The interface for this function is similar to the dynamic vertex case.

```
> input_network=rdNets[1:6]
> model.terms=c("triadcensus.003", "triadcensus.012", "triadcensus.102", "triadcensus.0
> model.formula = net~triadcensus(0:3)+gwesp(decay=0, fixed=FALSE, cutoff=30)-1;
> graph_mode='digraph';
> group='dnc';
> alpha.glmnet=1
> directed=TRUE;
> method <- 'bayesglm'
> maxlag <- 3
> lambda=NA
> intercept = c("edges")
> cdim <- length(model.terms)
> lagmat <- matrix(sample(c(0,1),(maxlag+1)*cdim,replace = TRUE),ncol = cdim)
> ylag <- rep(1,maxlag)
> exvar <- NA
> out <- suppressWarnings(paramEdge(input_network,
+                                   model.terms,
+                                   model.formula,
+                                   graph_mode='digraph',
+                                   group,intercept = c("edges"),exvar=NA,
+                                   maxlag = 3,
+                                   lagmat = matrix(sample(c(0,1),(maxlag+1)*cdim,
+                                                          replace = TRUE),ncol = cdim
+                                   ylag = rep(1,maxlag),
```

```
+                                    lambda = NA, method='bayesglm',
+                                    alpha.glmnet=1))
> out$coef

$coef
           edges     edgecov.dnc11     edgecov.dnc01     edgecov.dnc10
   -6.1490832123     -1.1271900526      0.0041958433      0.4398567614
   edgecov.dnc00    triadcensus.012            gwesp triadcensus.003.1
   -0.4810113628      0.0004529173      0.2729082692      0.0330950305
 triadcensus.012.1  triadcensus.102.1 triadcensus.021D.1          gwesp.1
    0.0117913270      0.0304005954     -0.0240556460     -0.2599131069
 triadcensus.003.2  triadcensus.012.2  triadcensus.102.2          gwesp.2
    0.0169342879      0.0162129739     -0.0098815499     -0.0558554348
 triadcensus.003.3 triadcensus.021D.3           gwesp.3              lag1
   -0.0020592027     -0.0544414053     -0.3444943234      1.1477484841
            lag2              lag3
    3.8265148583      9.2255277727

$se
           edges     edgecov.dnc11     edgecov.dnc01     edgecov.dnc10
      2.40807768        1.23650249        1.20857127        1.21305316
   edgecov.dnc00    triadcensus.012            gwesp triadcensus.003.1
      1.51080819        0.02240707        0.46772070        0.06962950
 triadcensus.012.1  triadcensus.102.1 triadcensus.021D.1          gwesp.1
      0.04363779        0.06026426        0.13238163        0.61408538
 triadcensus.003.2  triadcensus.012.2  triadcensus.102.2          gwesp.2
      0.06915443        0.04341503        0.05856483        0.65520480
 triadcensus.003.3 triadcensus.021D.3           gwesp.3              lag1
      0.03935342        0.13794127        0.63534375        1.66227152
            lag2              lag3
      1.79328785        1.21196120

$lambda
[1] NA

$fit

Call:  arm::bayesglm(formula = y ~ . - 1, family = binomial(link = "logit"),
    data = XYdata)

Coefficients:
           edges      edgecov.dnc11      edgecov.dnc01      edgecov.dnc10
      -6.1490832         -1.1271901          0.0041958          0.4398568
   edgecov.dnc00    triadcensus.012              gwesp  triadcensus.003.1
      -0.4810114          0.0004529          0.2729083          0.0330950
```

```
 triadcensus.012.1   triadcensus.102.1  triadcensus.021D.1             gwesp.1
         0.0117913           0.0304006          -0.0240556          -0.2599131
 triadcensus.003.2   triadcensus.012.2   triadcensus.102.2             gwesp.2
         0.0169343           0.0162130          -0.0098815          -0.0558554
 triadcensus.003.3  triadcensus.021D.3             gwesp.3                lag1
        -0.0020592          -0.0544414          -0.3444943           1.1477485
              lag2                lag3
         3.8265149           9.2255278

Degrees of Freedom: 6486 Total (i.e. Null);  6464 Residual
Null Deviance:          8992
Residual Deviance: 71.42        AIC: 115.4
```

Here the model formula is an ERGM formula. However, we have also provided the expansion of all the terms in the formula. For example the term 'triadcensus(0:3)' has been expanded out to respective triadcensus terms. The 'group' parameter is a categorical attribute for the vertices. This was present in the dynamic vertex case also. The specification of the intercept term is similar as well. The lag terms and lag dependcy of the parameters are represented with a binary vector or a binary matrix respectively.

We can use the model chosen to simulate the networks in future time points. Here we are simulating 10 future networks. We have kept the option of specifying the model and the initial network separate unlike the dynamic vertex case. However, using different model than the model fitted on the input network is not recommended as it is easily possible to create examples where these two inputs differ significantly, hurting the performance of the simulation.

```
> input_network=rdNets[1:6]
> model.terms=c("triadcensus.003", "triadcensus.012",
+               "triadcensus.102", "triadcensus.021D", "gwesp")
> model.formula = net~triadcensus(0:3)+gwesp(decay = 0, fixed=FALSE, cutoff=30)-1
> graph_mode='digraph'
> group='dnc'
> alpha.glmnet=1
> directed=TRUE
> method <- 'bayesglm'
> maxlag <- 3
> lambda=NA
> intercept = c("edges")
> cdim <- length(model.terms)
> lagmat <- matrix(sample(c(0,1),(maxlag+1)*cdim,replace = TRUE),ncol = cdim)
> ylag <- rep(1,maxlag)
> lagmat[1,] <- rep(0,ncol(lagmat))
> out <- suppressWarnings(paramEdge(input_network,model.terms, model.formula,
+                 graph_mode="digraph",group,intercept = c("edges"),exvar=NA,
+                 maxlag = 3,
```

```
+                 lagmat = lagmat,
+                 ylag = rep(1,maxlag),
+                 lambda = NA, method='bayesglm',
+                 alpha.glmnet=1))
> #
>
> start_network <- input_network
> inputcoeff <- out$coef$coef
> nvertex <- 47
> ns <- 10
> exvar <- NA
> input_network <- rdNets[1:6]
> maxlag <- 3
> start_network <- input_network
> inputcoeff <- out$coef$coef
> nvertex <- 47
> ns <- 10
> exvar <- NA
> tmp <- suppressWarnings(engineEdge(start_network=start_network,inputcoeff=inputcoeff,
+                      model.terms=model.terms, model.formula=model.formula,
+                      graph_mode=graph_mode,group=group,intercept=intercept,
+                      exvar=exvar,
+                      maxlag=maxlag,
+                      lagmat=lagmat,
+                      ylag=ylag,
+                      lambda = NA, method='bayesglm',
+                      alpha.glmnet=alpha.glmnet))

[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10

> par(mfrow = c(2,2))
> binaryPlot(input_network[[1]][, ], title = "Time point 6", axlabs = FALSE)
> binaryPlot(tmp$out_network[[1]][, ], title = "Time point 7 (simulated)", axlabs = FAL
> binaryPlot(tmp$out_network[[2]][, ], title = "Time point 8 (simulated)", axlabs = FAL
> binaryPlot(tmp$out_network[[3]][, ], title = "Time point 9 (simulated)", axlabs = FAL
```

2.1 Time series of parameter estimates

As all the coefficients are calculated as a part of the simulation, they are also provided along with the simulated networks. We can plot the time series of the network parameters to see the quality of the simulations.

```
> plot.ts(tmp$coefmat[, 1:10], xy.labels=FALSE,
+         main = "Estimated parameters from simulated networks", cex = 0.8)
```

2.2 Performance metrics

We also provide some functions for assesing the quality of the simulated networks and make comparisons with holdout set or some other benchmarked networks. Specifically, there are functions for number of triangles, cluster coeficient and expection of degree distribution has been implemented. We report the performance metrics for the input networks as well as the simulated networks for our example on blog data.

```
> perfMetrics <-
+     cbind(c(sapply(tmp$out_network, function(net) ntriangles(net[, ])),
+             sapply(input_network, function(net) ntriangles(net[, ]))),
+           c(sapply(tmp$out_network, function(net) clustCoef(net[, ])),
+             sapply(input_network, function(net) clustCoef(net[, ]))),
+           c(sapply(tmp$out_network, function(net) expdeg(net[, ])),
+             sapply(input_network, function(net) expdeg(net[, ]))))
> colnames(perfMetrics) <- c("Triangles", "ClustCoefs", "ExpDeg")
> perfMetrics <- data.frame(perfMetrics, row.names = NULL)
> knitr::kable(perfMetrics, digits = 3,
+              caption ="Performance metrics for input and simulated networks.")
```

```
| Triangles| ClustCoefs| ExpDeg|
|---------:|----------:|------:|
|       258|      0.445|  9.723|
|       239|      0.456|  9.085|
|       248|      0.470|  9.000|
|       246|      0.473|  8.915|
|       246|      0.464|  9.128|
|       241|      0.471|  8.872|
|       239|      0.443|  9.298|
|       250|      0.458|  9.255|
|       246|      0.473|  8.915|
|       249|      0.473|  8.957|
|       248|      0.476|  8.745|
|       257|      0.482|  9.085|
|       247|      0.475|  8.915|
|       246|      0.473|  8.915|
|       245|      0.472|  8.915|
|       248|      0.474|  9.000|
```

References

[1] A. Mallik and Z. W. Almquist. Stable Multiple Time Step Simulation/Prediction from Lagged Dynamic Network Regression Models. *ArXiv e-prints*, July 2018.