Extending ERGM Functionality within statnet: Building Custom User Terms

David R. Hunter
Steven M. Goodreau
Statnet Development Team

Sunbelt 2019

Outline

- Quick review of ERGMs
- Overview of change statistics
- Overview of steps for writing one's own statistic
- Detailed look at the R code
- Detailed look at the C code
- Group exercise

ERGM basic expression

Probability of observing a network (set of relationships) *y* on a given set of actors:

$$P(Y = y) = \frac{exp\{\theta' g(y)\}}{k(\theta)}$$

where: *g(y)* = vector of network statistics (like the predictors in a standard regression)

θ = vector of model parameters (like the coefficients in a standard regression)

$$k(\theta) = \sum_{\substack{\text{all nets } y^* \\ \text{on node} \\ \text{set of } Y}} \exp\{\theta' g(y^*)\}$$

Bahadur (1961), Besag (1974), Frank (1986); Wasserman and Pattison (1996)

ERGM logit formulation

The statement about the probability of a network:

$$P(Y = y) = \frac{exp\{\theta' g(y)\}}{k(\theta)}$$

where: *g(y)* = vector of network statistics
θ = vector of model parameters
$k(\theta) = \sum \exp\{\theta' g(y^*)\}$

Is equivalent to the statement about the conditional probability of any tie in the network:

$$\ln \frac{P(Y_{ij} = 1 \mid y_{ij}^c)}{P(Y_{ij} = 0 \mid y_{ij}^c)} = \theta' \left[g(y_{ij}^+) - g(y_{ij}^-) \right]$$

where: y_{ij} is the value of the tie from *i* to *j*
y_{ij}^c is the network *y,* excluding y_{ij}
y_{ij}^+ is the network *y* with y_{ij} set to 1
y_{ij}^- is the network *y* with y_{ij} set to 0

The quantity $g(y_{ij}^+) - g(y_{ij}^-)$ is also referred to as $\delta(y)_{ij}$, the change statistics for *i,j*

ERGMs and MCMC change statistics

Given a network and a model (i.e. a set of g(y) statistics proposed to be of interest), one typically wants to find maximum likelihood estimates of the θ coefficients for that model.

- The normalizing constant *k(θ)* makes this impossible to do directly.
- Main solution: Markov Chain Monte Carlo (Geyer and Thompson 1992, Crouch et al. 1998)

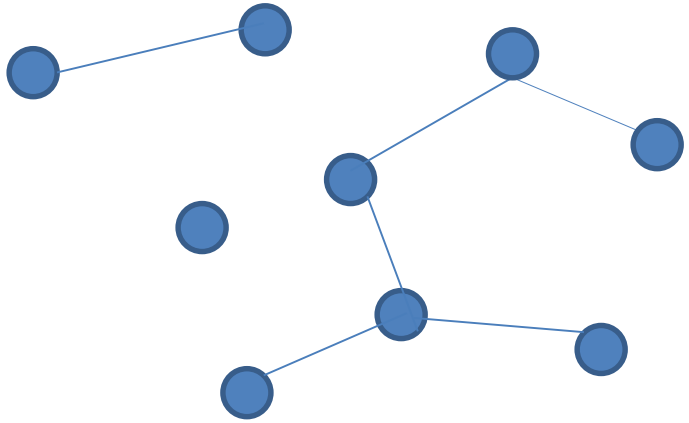
The MCMC algorithm repeatedly:

- selects an actor pair (or pairs)
- calculates the MCMC change statistics = model statistics for the network with those tie values toggled – model statistics for the current network
- uses an algorithm to decide whether or not to actually make those toggles

If one is only considering one actor pair, the MCMC change statistics must equal either $\delta(y)_{ij}$ or $-\delta(y)_{ij}$

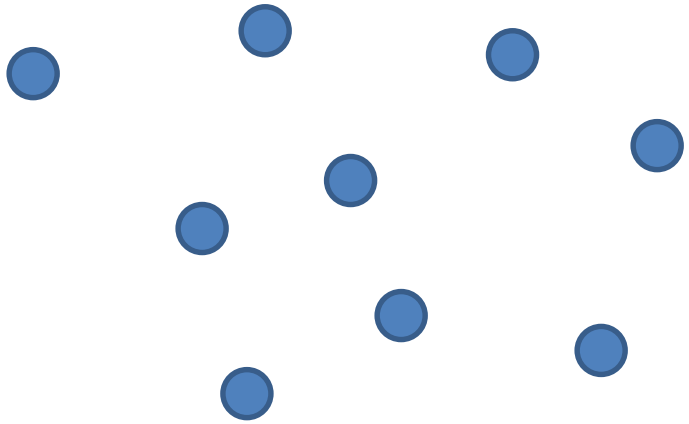
Calculating network statistics in the ergm package

Example: number of nodes of degree 2



Calculating network statistics in the ergm package

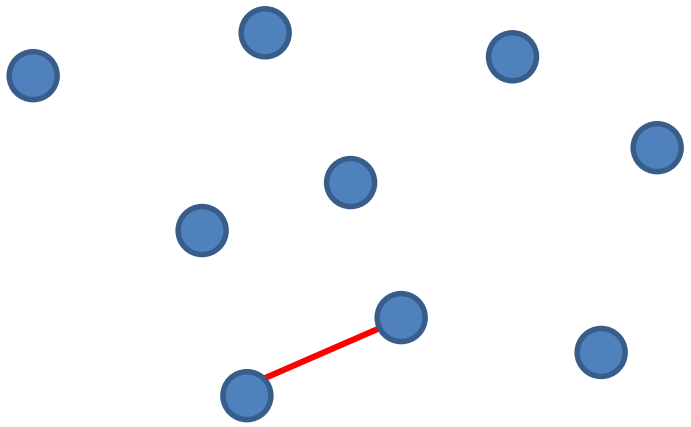
Example: number of nodes of degree 2



g(y) 0

Calculating network statistics in the ergm package

Example: number of nodes of degree 2

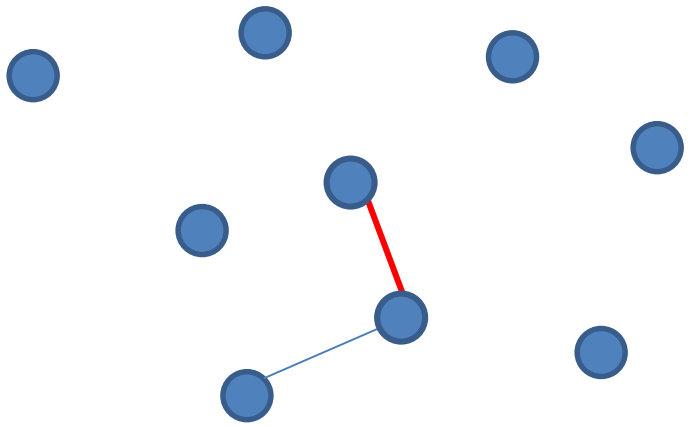


$g(y)$ 0 0

$\delta(y)_{ij}$ +0

Calculating network statistics in the ergm package

Example: number of nodes of degree 2

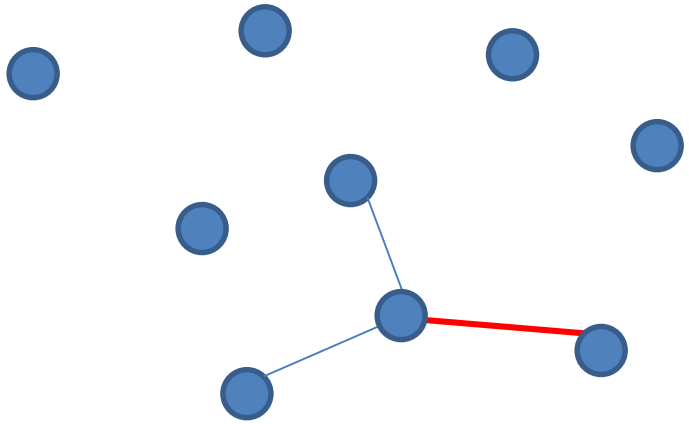


$g(y)$ 0 0 1

$\delta(y)_{ij}$ +0 +1

Calculating network statistics in the ergm package

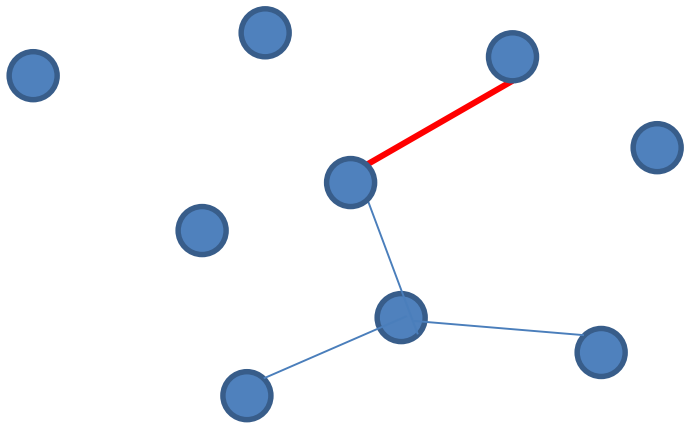
Example: number of nodes of degree 2



g(y)	0	0	1	0
$\delta(y)_{ij}$	+0	+1	−1	

Calculating network statistics in the ergm package

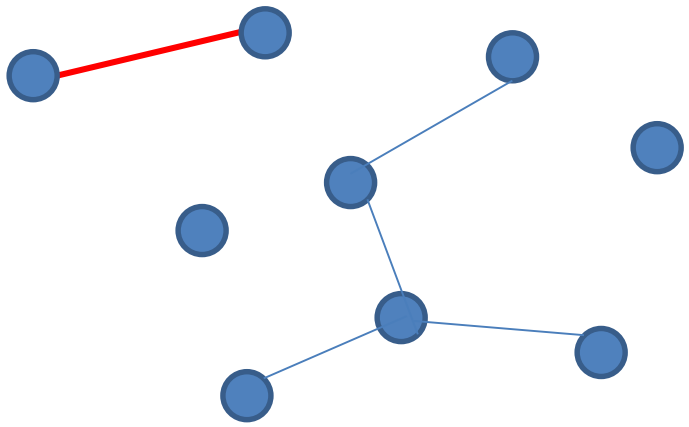
Example: number of nodes of degree 2



g(y)	0	0	1	0	1
$\delta(y)_{ij}$	+0	+1	−1	+1	

Calculating network statistics in the ergm package

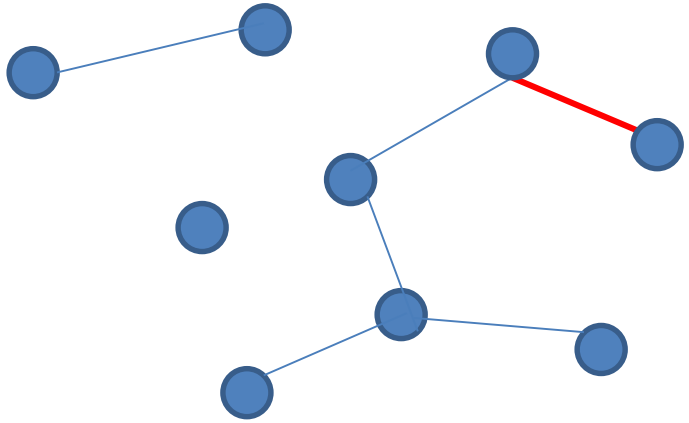
Example: number of nodes of degree 2



g(y)	0	0	1	0	1	1
$\delta(y)_{ij}$	+0	+1	−1	+1	+0	

Calculating network statistics in the ergm package

Example: number of nodes of degree 2



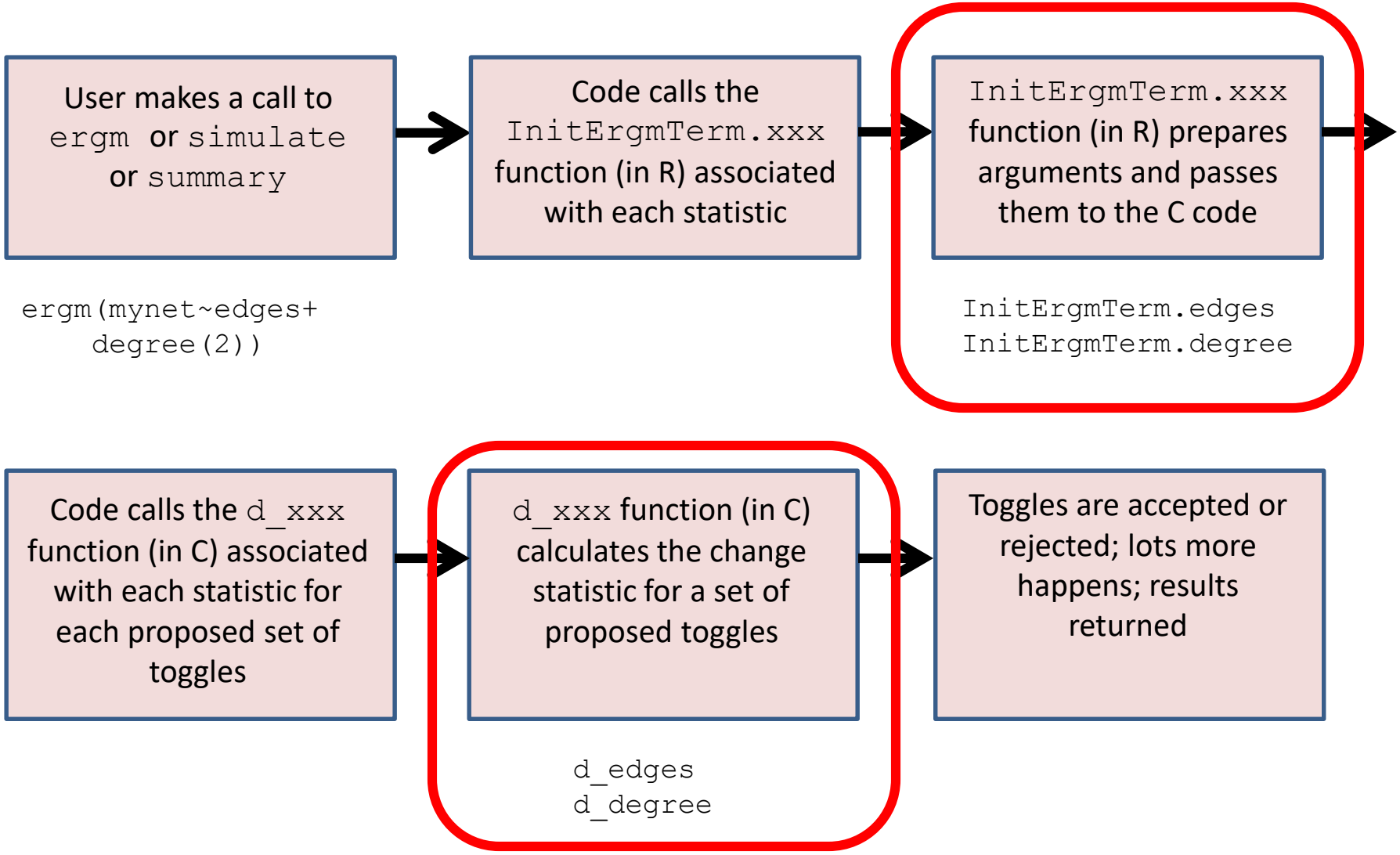
g(y)	0	0	1	0	1	1	2
$\delta(y)_{ij}$	+0	+1	−1	+1	+0	+1	

Commonly used change statistics included in ergm

```
absdiff absdiffcat adegcor altkstar asymmetric
b1concurrent b1degree b1degree.edgecov
b2degree.edgecov b1factor b1mindegree b2mindegree
b1mindegree.edgecov b2mindegree.edgecov b1star
b1starmix b1twostar b2concurrent b2degree b2factor
b2star b2starmix b2twostar balance coincidence
concurrent ctriple cycle cyclicalties degcor
degcrossprod degree density dsp dyadcov edgecov
edges esp gwb1degree gwb2degree gwdegree gwdsp
gwesp gwidegree gwnsp gwodegree hamming hammingmix
idegree indegreepopularity intransitive isolates
istar kstar localtriangle m2star meandeg mutual
nearsimmelian nodecov nodefactor nodeicov
nodeifactor nodematch nodemix nodeocov nodeofactor
nsp odegree outdegreepopularity opentriad ostar
pdegcor rdegcor receiver sender simmelian
simmelianties smalldiff sociality threepath
transitive transitiveties triadcensus triangle
tripercent ttriple twopath
```

Most of which are documented at help("ergm-terms")

General structure of an ergm call



Building your own terms in ergm requires:

- The source code for the ergm.userterms package
- The tools and knowledge needed to build R packages from source (RStudio makes this easy)
- Writing an InitErgmTerm.xxx function (in R)
- Writing a d_xxx function (in C)

Optionally:

- Download the source code for the ergm package (or use Github to look at it)

The edges statistic:

InitErgmTerm.edges

```
InitErgmTerm.edges<-function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist,
                      varnames = NULL,
                      vartypes = NULL,
                      defaultvalues = list(),
                      required = NULL)
  list(name="edges", coef.names="edges", dependence=FALSE,
     minval = 0, maxval = network.dyadcount(nw, FALSE),
     conflicts.constraints = "edges")
}
```

d_edges

```
D_CHANGESTAT_FN(d_edges) {
  int edgeflag, i;

  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    {
      edgeflag = IS_OUTEDGE(TAIL(i), HEAD(i));
      CHANGE_STAT[0] += edgeflag ? - 1 : 1;
      TOGGLE_IF_MORE_TO_COME(i);
    }
  UNDO_PREVIOUS_TOGGLES(i);
}
```

The absdiff statistic

From `?'ergm-terms'`

`absdiff(attr, pow=1) (binary) (dyad-independent) (frequently-used) (directed) (undirected) (quantitative nodal attribute),`

`absdiff(attr, pow=1, form ="sum") (valued) (dyad-independent) (directed) (undirected) (quantitative nodal attribute)`

Absolute difference: The `attr` argument specifies a quantitative attribute (see Specifying Vertex Attributes and Levels for details). This term adds one network statistic to the model equaling the sum of `abs(attr[i]-attr[j])^pow` for all edges `(i,j)` in the network.

Note that ergm versions 3.9.4 and earlier used different arguments for this term. See the above section on versioning for invoking the old behavior.

- **Binary ergms call an R function of the form InitErgmTerm.xxx**
- **Valued ergms call a separate R function called InitWtErgmTerm.xxx**
- **We will focus only the former today.**

```
> ?("ergm-attr")
```

node-attr {ergm} R Documentation

Specifying nodal attributes and their levels

Description

This document describes the ways to specify nodal attributes or functions of nodal attributes and which levels for categorical factors to include. For the helper functions to facilitate this, see `node-attr-api`.

Details

Term nodal attribute arguments, typically called `attr`, `attrs`, `by`, or `on` are interpreted as follows:

a character string

> Extract the vertex attribute with this name.

a character vector of length > 1

> Extract the vertex attributes and paste them together, separated by dots if the term expects categorical attributes and (typically) combine into a covariate matrix if it expects quantitative attributes.

a function

> The function is called on the LHS network, expected to return a vector or matrix of appropriate dimension. (Shorter vectors and matrix columns will be recycled as needed.)

a formula

> The expression on the RHS of the formula is evaluated in an environment of the vertex attributes of the network, expected to return a vector or matrix of appropriate dimension. (Shorter vectors and matrix columns will be recycled as needed.) Within this expression, the network itself accessible as either `.` or `.nw`. For example, `nodecov(~abs(Grade-mean(Grade))/network.size(.))` would return the absolute difference of each actor's "Grade" attribute from its network-wide mean, divided by the network size.

For categorical attributes, to select which levels are of interest and their ordering, use the argument `levels`. Selection of nodes (from the appropriate vector of nodal indices) is likewise handled as the selection of levels, using the argument `nodes`. These arguments are interpreted as follows:

an expression wrapped in `I()`

Anatomy of an InitErgmTerm function

```
InitErgmTerm.absdiff <- function(
                            nw, arglist, ..., version = packageVersion("ergm")) {
  if (version <= as.package_version("3.9.4")) {
    a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                        varnames = c("attrname", "pow"),
                        vartypes = c("character", "numeric"),
                        defaultvalues = list(NULL, 1),
                        required = c(TRUE, FALSE))
    nodecov <- get.node.attr(nw, a$attrname)
    covname <- a$attrname }
  else {
    a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                        varnames = c("attr", "pow"),
                        vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                        defaultvalues = list(NULL, 1),
                        required = c(TRUE, FALSE))
    nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
    covname <- attr(nodecov, "name")
  }
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

Anatomy of an InitErgmTerm function

1. Function call

```
InitErgmTerm.absdiff <- function(
                          nw, arglist, ..., version = packageVersion("ergm")) {
  if (version <= as.package_version("3.9.4")) {
    a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                        varnames = c("attrname", "pow"),
                        vartypes = c("character", "numeric"),
                        defaultvalues = list(NULL, 1),
                        required = c(TRUE, FALSE))
    nodecov <- get.node.attr(nw, a$attrname)
    covname <- a$attrname }
  else {
    a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                        varnames = c("attr", "pow"),
                        vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                        defaultvalues = list(NULL, 1),
                        required = c(TRUE, FALSE))
    nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
    covname <- attr(nodecov, "name")
  }
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

Anatomy of an InitErgmTerm function

1a. Code for backwards compatibility *

```
InitErgmTerm.absdiff <- function(
                          nw, arglist, ..., version = packageVersion("ergm")) {
  if (version <= as.package_version("3.9.4")) {
    a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                        varnames = c("attrname", "pow"),
                        vartypes = c("character", "numeric"),
                        defaultvalues = list(NULL, 1),
                        required = c(TRUE, FALSE))
    nodecov <- get.node.attr(nw, a$attrname)
    covname <- a$attrname }
  else {
    a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                        varnames = c("attr", "pow"),
                        vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                        defaultvalues = list(NULL, 1),
                        required = c(TRUE, FALSE))
    nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
    covname <- attr(nodecov, "name")
  }
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

* Not your problem when writing a new statistic

Anatomy of an InitErgmTerm function

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                      varnames = c("attr", "pow"),
                      vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                      defaultvalues = list(NULL, 1),
                      required = c(TRUE, FALSE))
  nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
  covname <- attr(nodecov, "name")
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

Anatomy of an InitErgmTerm function

2. Checking input

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                      varnames = c("attr", "pow"),
                      vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                      defaultvalues = list(NULL, 1),
                      required = c(TRUE, FALSE))
  nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
  covname <- attr(nodecov, "name")
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

Anatomy of an InitErgmTerm function

3. Processing input (optional)

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                      varnames = c("attr", "pow"),
                      vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                      defaultvalues = list(NULL, 1),
                      required = c(TRUE, FALSE))
  nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
  covname <- attr(nodecov, "name")
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

Anatomy of an InitErgmTerm function

4. Constructing output

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                      varnames = c("attr", "pow"),
                      vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                      defaultvalues = list(NULL, 1),
                      required = c(TRUE, FALSE))
  nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
  covname <- attr(nodecov, "name")
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

Anatomy of an InitErgmTerm function

1. Function call

- only thing you need to change is the name

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                      varnames = c("attr", "pow"),
                      vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                      defaultvalues = list(NULL, 1),
                      required = c(TRUE, FALSE))
  nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
  covname <- attr(nodecov, "name")
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

Anatomy of an InitErgmTerm function

2. Checking input

- Check network and argument (don't change)
- Let R know if OK for directed/undirected, bipartite/non-bipartite

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                      varnames = c("attr", "pow"),
                      vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                      defaultvalues = list(NULL, 1),
                      required = c(TRUE, FALSE))
  nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
  covname <- attr(nodecov, "name")
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

`directed=`	`TRUE`	**works for directed networks only**
	`FALSE`	**works for undirected networks only**
	`NULL`	**works for either (default)**
`bipartite=`	`TRUE`	**works for bipartite networks only**
	`FALSE`	**works for unipartite networks only**
	`NULL`	**works for either (default)**

Anatomy of an InitErgmTerm function

2. Checking input

- Check network and argument (don't change)
- Let R know if OK for directed/undirected, bipartite/non-bipartite
- List arguments, and specify their type, default value and whether required

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                      varnames = c("attr", "pow"),
                      vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                      defaultvalues = list(NULL, 1),
                      required = c(TRUE, FALSE))
  nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
  covname <- attr(nodecov, "name")
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

```
> ?("node-attr-api")
```

node-attr-api {ergm} R Documentation

Helper functions for specifying nodal attribute levels

Description

These functions are meant to be used in `InitErgmTerm` and other implementations to provide the user with a way to extract nodal attributes and select their levels in standardized and flexible ways described under `node-attr`.

`ergm_get_vattr` extracts and processes the specified nodal attribute vector. It is strongly recommended that `check.ErgmTerm()`'s corresponding `vartype="function,formula,character"` (using the `ERGM_VATTR_SPEC` constant).

`ergm_attr_levels` filters the levels of the attribute. It is strongly recommended that `check.ErgmTerm()`'s corresponding `vartype="function,formula,character,numeric,logical,AsIs,NULL"` (using the `ERGM_LEVELS_SPEC` constant).

Usage

```
ERGM_GET_VATTR_MULTIPLE_TYPES

ergm_get_vattr(object, nw, accept = "character", bip = c("n", "b1",
  "b2"), multiple = if (accept == "character") "paste" else "stop", ...)

## S3 method for class 'character'
ergm_get_vattr(object, nw, accept = "character",
```

Anatomy of an InitErgmTerm function

3. Processing input

- Pulling values of arguments out from *a*
- Processing them in any way needed for passing

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                      varnames = c("attr", "pow"),
                      vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                      defaultvalues = list(NULL, 1),
                      required = c(TRUE, FALSE))
  nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
  covname <- attr(nodecov, "name")
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

Anatomy of an InitErgmTerm function

4. Constructing output

- C function name (w/o the d_)
- Coefficient name(s)
- Inputs to pass to C function
- Whether dyadic dependent
- Empty network stat(s) (opt.)
- No need to pass the network

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                      varnames = c("attr", "pow"),
                      vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                      defaultvalues = list(NULL, 1),
                      required = c(TRUE, FALSE))
  nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
  covname <- attr(nodecov, "name")
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
         sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

Anatomy of a d_ function

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Anatomy of a d_ function

1. Function call

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Anatomy of a d_ function

2. Variable declaration

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Anatomy of a d_ function

3. Multiple toggle code *

*stay tuned for cool updates (see next-to-last slide for preview)

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Anatomy of a d_ function

4. Change statistic(s) calculation

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Anatomy of a d_ function

1. Function call

- only thing you need to change is the name

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Anatomy of a d_ function

2. Variable declaration

- Declare all variables you use (easiest to do at the end)
- We've created two extra variable types available: Vertex and Edge

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Anatomy of a d_ function

3. Multiple toggle code

- Don't change a thing! (but stay tuned for optional simplifications coming soon)

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Anatomy of a d_ function

4. Change statistic(s) calculation

- Pull out the tail and head of the toggle
- Pull out other parameters passed, by position

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Never forget:

R indexes vectors beginning with 1

C indexes vectors beginning with 0

Examples of macros

Found in ergm/inst/include/ergm_changestat.h in the source code
[NB: It's a good idea to obtain the source code for the ergm package!]

- TAIL(i)	ID of tail node in toggle i
- HEAD(i)	ID of head node in toggle i
- IS_OUTEDGE(a,b)	1/0 for whether edge a->b exists
- IS_INEDGE(a,b)	1/0 for whether edge a<-b exists
- IS_UNDIRECTED_EDGE(a,b)	1/0 for whether edge a--b exists
- INPUT_PARAM	inputs passed from R
- INPUT_ATTRIB	inputs passed from R (same as INPUT_PARAM)
- N_INPUT_PARAMS	number of inputs passed from R
- OUT_DEG[a]	outdegree of node a
- IN_DEG[a]	indegree of node a
- N_EDGES	total # of edges in the network currently
- N_NODES	total # of nodes in the network
- N_DYADS	total # of dyads in the network
- DIRECTED	1 if network is directed, 0 if directed
STEP_THROUGH_OUTEDGES(a,e,v)	sets up loop to go through all of node a's outedges, indexed by v

Network storage in ergm

Directed network

represents both: an outedge from 3 to 5
an inedge 5 to 3

```
IS_OUTEDGE(3,5)      1 (true)
IS_INEDGE(5,3)       1 (true)

IS_OUTEDGE(5,3)      0 (false)
IS_INEDGE(3,5)       0 (false)
```

Undirected network

stored as directed
tail node < head node

Anatomy of a d_ function

4. Change statistic(s) calculation

- Calculate change statistic(s) for i,j (often with conditionals)
- Common practice: flip sign(s) for dissolution toggle

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Example: mymindegree

Code up a term that that

- **Works on undirected, non-bipartite networks only**
- **Refers to this statistic: "The number of nodes in the network of at least degree *x*"**
- **Only needs to handle one value of *x* (i.e., only produces one change statistic), not a whole vector**

Example: mymindegree

Notes

- Work together, but ask us for help if needed
- If you're stuck you can also look at the mindegree term included within ergm.userterms
- Note that that term is more complex though: It takes an optional attribute
- Note also that the version in the ergm.userterms package from CRAN does not use the latest attribute-based API – that can be found at
  - https://github.com/statnet/ergm.userterms/blob/master/R/InitErgmTerm.users.R
  - https://github.com/statnet/ergm.userterms/blob/master/src/changestats.users.c

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  a <- check.ErgmTerm(nw, arglist, directed = NULL, bipartite = NULL,
                          varnames = c("attr", "pow"),
                          vartypes = c(ERGM_VATTR_SPEC, "numeric"),
                          defaultvalues = list(NULL, 1),
                          required = c(TRUE, FALSE))
  nodecov <- ergm_get_vattr(a$attr, nw, accept = "numeric")
  covname <- attr(nodecov, "name")
  list(name = "absdiff",
       coef.names = paste(paste("absdiff", if (a$pow != 1) a$pow else "",
       sep = ""), covname, sep = "."),
       inputs = c(a$pow, nodecov),
       dependence = FALSE)
}
```

```
D_CHANGESTAT_FN(d_absdiff) {
  double change, p;
  Vertex tail, head;
  int i;

  /* *** don't forget tail -> head */
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    tail = TAIL(i);
    head = HEAD(i);
    p = INPUT_ATTRIB[0];
    if(p==1.0){
      change = fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]);
    } else {
      change = pow(fabs(INPUT_ATTRIB[tail] - INPUT_ATTRIB[head]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(tail,head) ? -change : change;
    TOGGLE_IF_MORE_TO_COME(i); /* Needed in case of multiple toggles */
  }
  UNDO_PREVIOUS_TOGGLES(i); /* Needed on exit in case of multiple toggles */
}
```

Put it all together:

1. Put the R code in its own file named *.R in the R directory of your ergm.userterms source code, or at the end of the file "InitErgmTerm.users.R" in that directory

2. Put the C code at the end of the file "changestats.users.c" in the src directory of your ergm.userterms source code directory

3. In Rstudio, select Build > Build and Reload

4. Use your new term!!

Bonus tasks

- **Tweak the term**
  - **Add an attribute (either for the ego or the alters)**
  - **Have it take a vector of mindegrees**
  - **Something else creative and new**
- **Code up something else altogether!**
- **Show off the code to the crowd with a cool example**

Looking forward

- We regularly release new functionality to streamline the process
- E.g., roll-out of new macros:

```
ZERO_ALL_CHANGESTATS(i);
FOR_EACH_TOGGLE(i) {
  Code to calculate change stat
  TOGGLE_IF_MORE_TO_COME(i);
}
UNDO_PREVIOUS_TOGGLES(i);
```



```
EXEC_THROUGH_TOGGLES({
  Code to calculate change stat
})
```

Sharing new change stats with the world

- Learn basic GitHub
- Fork the ergm.terms.contrib repository (not ergm.userterms) from the statnet organization
- Write your R and C code
- Debug it thoroughly
- Have others test it for you as well if possible
- Document it well (see existing terms for guidance)
- Submit a pull request
- Advertise it on the statnet listserv once the pull request is accepted