Extending ERGM Functionality within statnet: Building Custom User Terms

David R. Hunter

Steven M. Goodreau

Statnet Development Team

Sunbelt 2012

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 statistics in a standard regression)

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

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

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 $\delta(yij)$, 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)

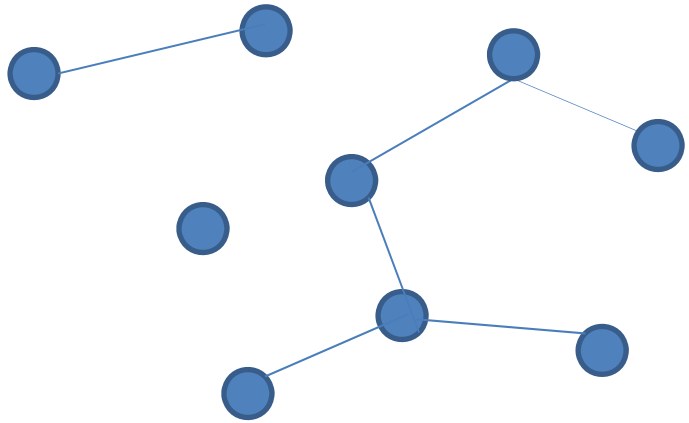
The MCMC algorithm repeatedly:

- 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 $\delta(yij)$ or $-\delta(yij)$

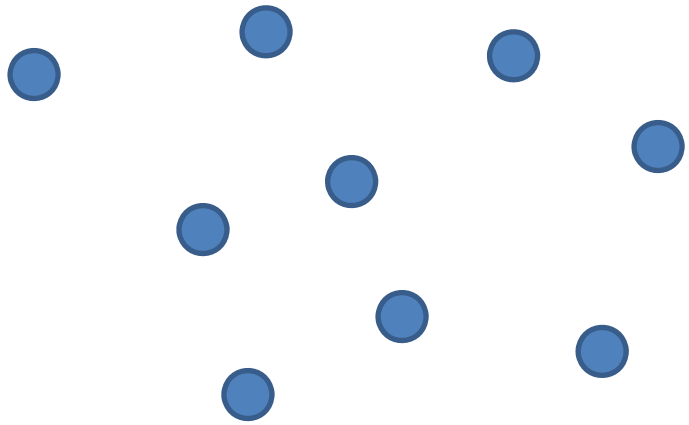
Calculating network statistics in the ergm package

Example: number of nodes of degree 2



Calculating network statistics in the ergm package

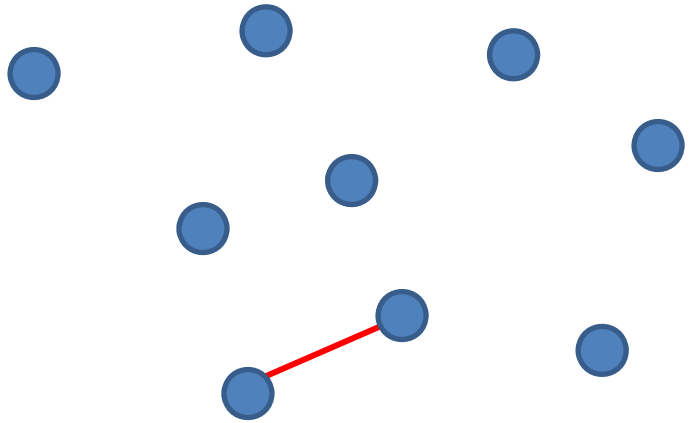
Example: number of nodes of degree 2



g(y) θ

Calculating network statistics in the ergm package

Example: number of nodes of degree 2

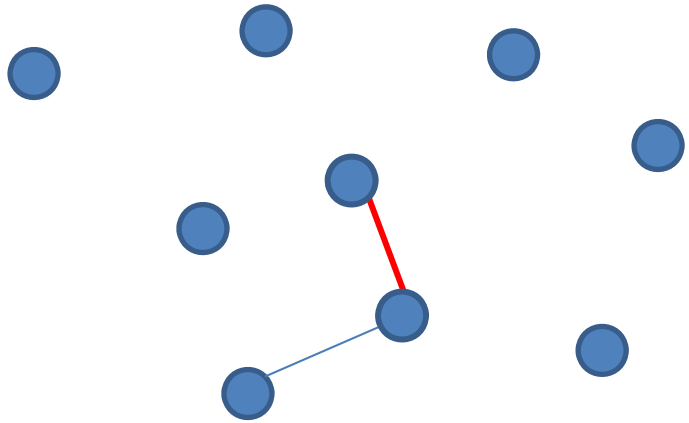


$g(y)$ θ θ

$\delta(y_{ij})$ $+\theta$

Calculating network statistics in the ergm package

Example: number of nodes of degree 2

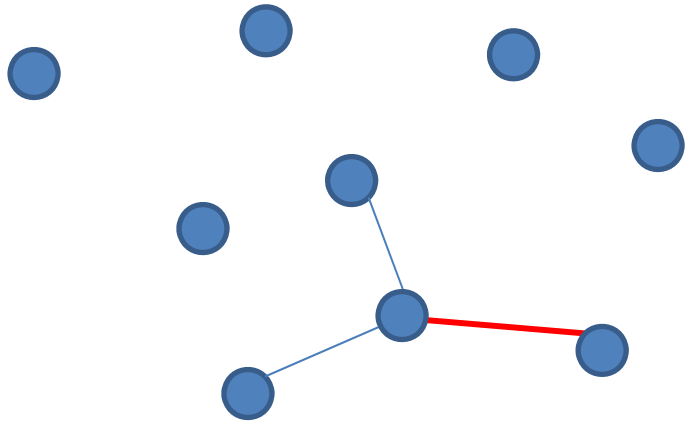


g(y) 0 0 1

$\delta(y_{ij})$ +0 +1

Calculating network statistics in the ergm package

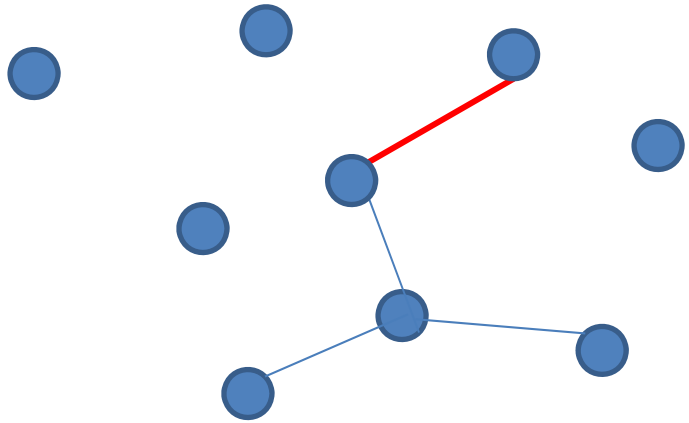
Example: number of nodes of degree 2



g(y)	0	0	1	0
$\delta(y_{ij})$	+0	+1	-1	

Calculating network statistics in the ergm package

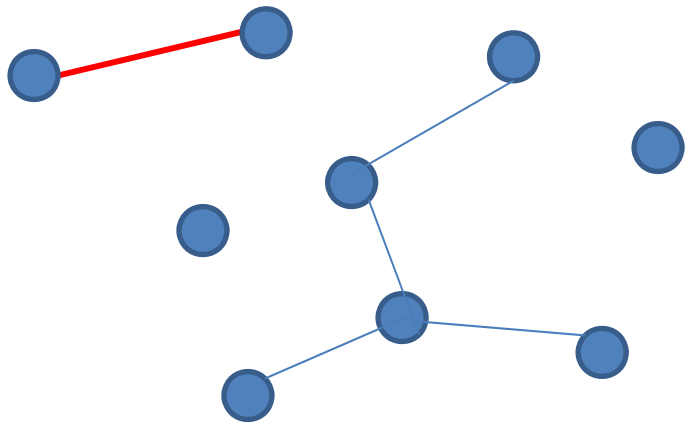
Example: number of nodes of degree 2



g(y)	0	0	1	0	1
$\delta(y_{ij})$	+0	+1	-1	+1	

Calculating network statistics in the ergm package

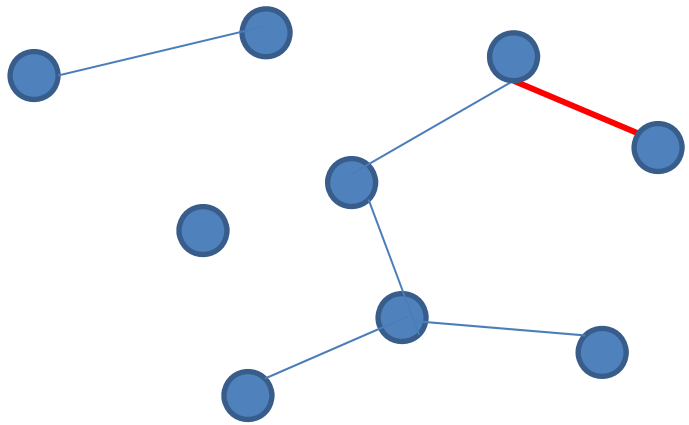
Example: number of nodes of degree 2



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



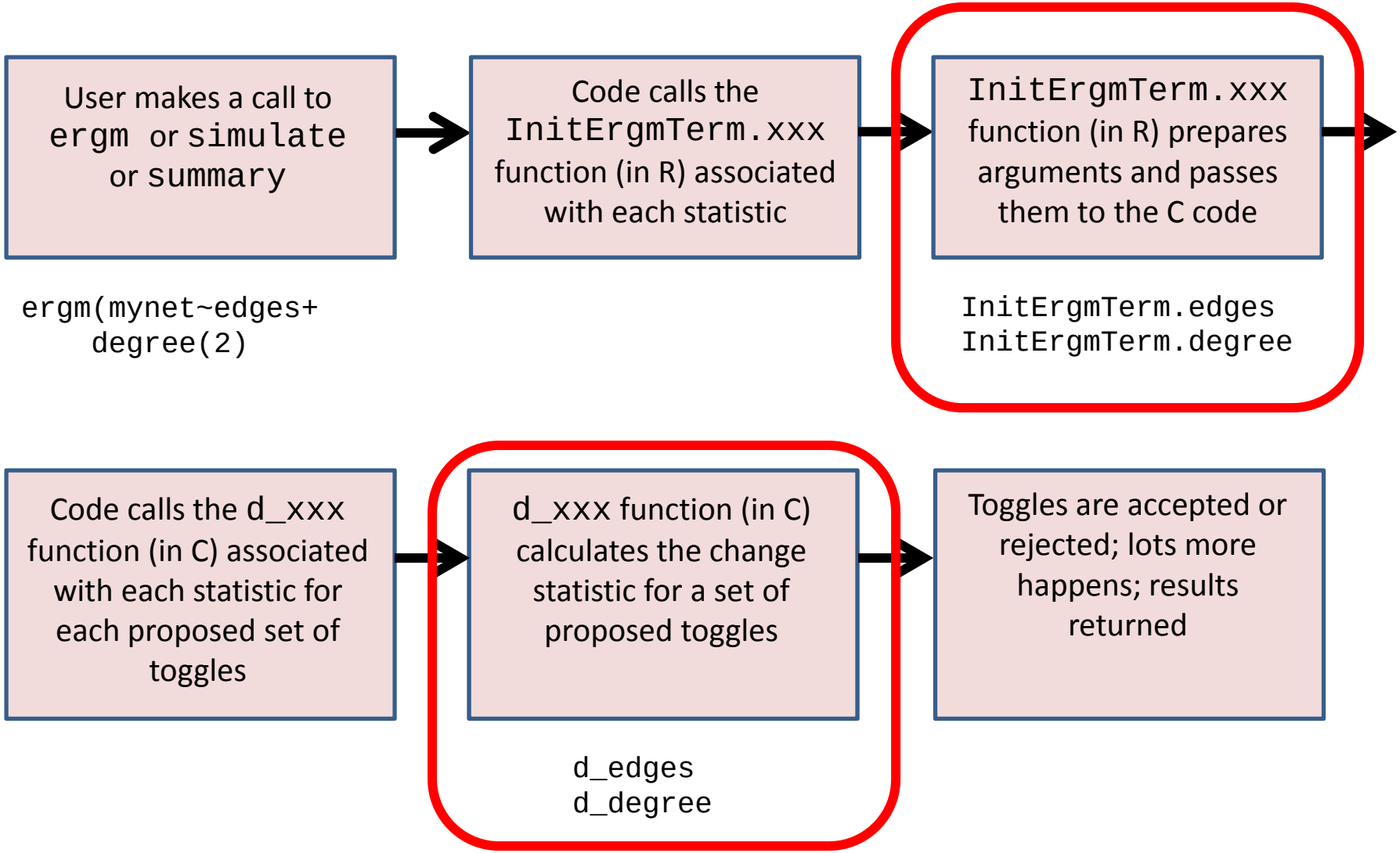
g(y)	0	0	1	0	1	1	2
$\delta(y_{ij})$		+0	+1	-1	+1	+0	+1

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
- Writing an InitErgmTerm.xxx function (in R)
- Writing a d_xxx function (in C)

The edges statistic:

InitErgmTerm.edges

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

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);
}
```

Anatomy of an InitErgmTerm function

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  ### Check the network and arguments to make sure they are appropriate.
  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))
  ### Process the arguments
  nodecov <- get.node.attr(nw, a$attrname)
  ### Construct the list to return
  list(name="absdiff",
    coef.names = paste(paste("absdiff", if(a$pow != 1) a$pow else "",
        sep = ""), a$attrname, sep = "."),
    pkgname = "ergm.userterms",
    inputs = c(a$pow, nodecov),
    dependence = FALSE )
}
```

Anatomy of an InitErgmTerm function

1. Function call

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  ### Check the network and arguments to make sure they are appropriate.
  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))
  ### Process the arguments
  nodecov <- get.node.attr(nw, a$attrname)
  ### Construct the list to return
  list(name="absdiff",
    coef.names = paste(paste("absdiff", if(a$pow != 1) a$pow else "",
        sep = ""), a$attrname, sep = "."),
    pkgname = "ergm.userterms",
    inputs = c(a$pow, nodecov),
    dependence = FALSE )
}
```

Anatomy of an InitErgmTerm function

2. Checking input

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  ### Check the network and arguments to make sure they are appropriate.
  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))
  ### Process the arguments
  nodecov <- get.node.attr(nw, a$attrname)
  ### Construct the list to return
  list(name="absdiff",
    coef.names = paste(paste("absdiff", if(a$pow != 1) a$pow else "",
        sep = ""), a$attrname, sep = "."),
    pkgname = "ergm.userterms",
    inputs = c(a$pow, nodecov),
    dependence = FALSE )
}
```

Anatomy of an InitErgmTerm function

3. Processing input (optional)

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  ### Check the network and arguments to make sure they are appropriate.
  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))
  ### Process the arguments
  nodecov <- get.node.attr(nw, a$attrname)
  ### Construct the list to return
  list(name="absdiff",
    coef.names = paste(paste("absdiff", if(a$pow != 1) a$pow else "",
        sep = ""), a$attrname, sep = "."),
    pkgname = "ergm.userterms",
    inputs = c(a$pow, nodecov),
    dependence = FALSE )
}
```

Anatomy of an InitErgmTerm function

4. Constructing output

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  ### Check the network and arguments to make sure they are appropriate.
  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))
  ### Process the arguments
  nodecov <- get.node.attr(nw, a$attrname)
  ### Construct the list to return
  list(name="absdiff",
    coef.names = paste(paste("absdiff", if(a$pow != 1) a$pow else "",
        sep = ""), a$attrname, sep = "."),
    pkgname = "ergm.userterms",
    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, ...) {
  ### Check the network and arguments to make sure they are appropriate.
  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))
  ### Process the arguments
  nodecov <- get.node.attr(nw, a$attrname)
  ### Construct the list to return
  list(name="absdiff",
    coef.names = paste(paste("absdiff", if(a$pow != 1) a$pow else "",
        sep = ""), a$attrname, sep = "."),
    pkgname = "ergm.userterms",
    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
- List arguments, and specify their type, default value and whether required

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  ### Check the network and arguments to make sure they are appropriate.
  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))
  ### Process the arguments
  nodecov <- get.node.attr(nw, a$attrname)
  ### Construct the list to return
  list(name="absdiff",
    coef.names = paste(paste("absdiff", if(a$pow != 1) a$pow else "",
        sep = ""), a$attrname, sep = "."),
    pkgname = "ergm.userterms",
    inputs = c(a$pow, nodecov),
    dependence = FALSE )
}
```

Anatomy of an InitErgmTerm function

3. Processing output

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

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  ### Check the network and arguments to make sure they are appropriate.
  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))
  ### Process the arguments
  nodecov <- get.node.attr(nw, a$attrname)
  ### Construct the list to return
  list(name="absdiff",
    coef.names = paste(paste("absdiff", if(a$pow != 1) a$pow else "",
        sep = ""), a$attrname, sep = "."),
    pkgname = "ergm.userterms",
    inputs = c(a$pow, nodecov),
    dependence = FALSE )
}
```

Anatomy of an InitErgmTerm function

4. Constructing output

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

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  ### Check the network and arguments to make sure they are appropriate.
  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))
  ### Process the arguments
  nodecov <- get.node.attr(nw, a$attrname)
  ### Construct the list to return
  list(name="absdiff",
    coef.names = paste(paste("absdiff", if(a$pow != 1) a$pow else "",
        sep = ""), a$attrname, sep = "."),
    pkgname = "ergm.userterms",
    inputs = c(a$pow, nodecov),
    dependence = FALSE )
}
```

Anatomy of a d_ function

```
CHANGESTAT_FN(d_absdiff) {
  double change, p; Vertex t, h; int i;
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    t = TAIL(i); h = HEAD(i);
    p = INPUT_PARAM[0];
    if(p==1.0){
      change = fabs(INPUT_PARAM[t] - INPUT_PARAM[h]);
    }else{
      change = pow(fabs(INPUT_PARAM[t] - INPUT_PARAM[h]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(t,h) ? -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

```
CHANGESTAT_FN(d_absdiff) {
  double change, p; Vertex t, h; int i;
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    t = TAIL(i); h = HEAD(i);
    p = INPUT_PARAM[0];
    if(p==1.0){
      change = fabs(INPUT_PARAM[t] - INPUT_PARAM[h]);
    }else{
      change = pow(fabs(INPUT_PARAM[t] - INPUT_PARAM[h]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(t,h) ? -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

```
CHANGESTAT_FN(d_absdiff) {
  double change, p; Vertex t, h; int i;
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    t = TAIL(i); h = HEAD(i);
    p = INPUT_PARAM[0];
    if(p==1.0){
      change = fabs(INPUT_PARAM[t] - INPUT_PARAM[h]);
    }else{
      change = pow(fabs(INPUT_PARAM[t] - INPUT_PARAM[h]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(t,h) ? -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

```
CHANGESTAT_FN(d_absdiff) {
  double change, p; Vertex t, h; int i;
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    t = TAIL(i); h = HEAD(i);
    p = INPUT_PARAM[0];
    if(p==1.0){
      change = fabs(INPUT_PARAM[t] - INPUT_PARAM[h]);
    }else{
      change = pow(fabs(INPUT_PARAM[t] - INPUT_PARAM[h]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(t,h) ? -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 calculation

```
CHANGESTAT_FN(d_absdiff) {
  double change, p; Vertex t, h; int i;
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    t = TAIL(i); h = HEAD(i);
    p = INPUT_PARAM[0];
    if(p==1.0){
      change = fabs(INPUT_PARAM[t] - INPUT_PARAM[h]);
    }else{
      change = pow(fabs(INPUT_PARAM[t] - INPUT_PARAM[h]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(t,h) ? -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

```
CHANGESTAT_FN(d_absdiff) {
  double change, p; Vertex t, h; int i;
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    t = TAIL(i); h = HEAD(i);
    p = INPUT_PARAM[0];
    if(p==1.0){
      change = fabs(INPUT_PARAM[t] - INPUT_PARAM[h]);
    }else{
      change = pow(fabs(INPUT_PARAM[t] - INPUT_PARAM[h]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(t,h) ? -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)
- Note the two extra variable types available: Vertex and Edge

```
CHANGESTAT_FN(d_absdiff) {
  double change, p; Vertex t, h; int i;
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    t = TAIL(i); h = HEAD(i);
    p = INPUT_PARAM[0];
    if(p==1.0){
      change = fabs(INPUT_PARAM[t] - INPUT_PARAM[h]);
    }else{
      change = pow(fabs(INPUT_PARAM[t] - INPUT_PARAM[h]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(t,h) ? -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!

```
CHANGESTAT_FN(d_absdiff) {
  double change, p; Vertex t, h; int i;
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    t = TAIL(i); h = HEAD(i);
    p = INPUT_PARAM[0];
    if(p==1.0){
      change = fabs(INPUT_PARAM[t] - INPUT_PARAM[h]);
    }else{
      change = pow(fabs(INPUT_PARAM[t] - INPUT_PARAM[h]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(t,h) ? -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 calculation

- Pull out the tail and head of the toggle
- Pull out other parameters passed, by position
- Calculate change statistic for i,j (often with conditionals)
- Common practice: flip sign for dissolution toggle

```
CHANGESTAT_FN(d_absdiff) {
  double change, p; Vertex t, h; int i;
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    t = TAIL(i); h = HEAD(i);
    p = INPUT_PARAM[0];
    if(p==1.0){
      change = fabs(INPUT_PARAM[t] - INPUT_PARAM[h]);
    }else{
      change = pow(fabs(INPUT_PARAM[t] - INPUT_PARAM[h]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(t,h) ? -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 */
}
```

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

Examples of macros

Found in changestat.h in the src directory of your ergm.userterms source code folder

- **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_UNDIRECTEDEDGE(a,b)** — **1/0 for whether edge a--b exists**

- **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** — **0 if network is directed, 1 if directed**

- **INPUT_PARAM** — **inputs passed from R**
- **N_INPUT_PARAMS** — **number of inputs passed from R**

- **STEP_THROUGH_OUTEDGES(a,e,v) : sets up loop to go through all of node a's outedges, indexed by v**

Example: mymindegree

We want a term that

- Works on undirected, non-bipartite networks only
- Refers to the number of nodes in the network of at least degree *x*
- Only needs to handle one value of *x*, not a whole vector

Example: degree2

We want a term that

- Calculates the number of nodes of degree 2

```
InitErgmTerm.absdiff <- function(nw, arglist, ...) {
  ### Check the network and arguments to make sure they are appropriate.
  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))
  ### Process the arguments
  nodecov <- get.node.attr(nw, a$attrname)
  ### Construct the list to return
  list(name="absdiff",
    coef.names = paste(paste("absdiff", if(a$pow != 1) a$pow else "",
         sep = ""), a$attrname, sep = "."),
    pkgname = "ergm.userterms",
    inputs = c(a$pow, nodecov),
    dependence = FALSE )
}
```

```
CHANGESTAT_FN(d_absdiff) {
  double change, p; Vertex t, h; int i;
  ZERO_ALL_CHANGESTATS(i);
  FOR_EACH_TOGGLE(i) {
    t = TAIL(i); h = HEAD(i);
    p = INPUT_PARAM[0];
    if(p==1.0){
      change = fabs(INPUT_PARAM[t] - INPUT_PARAM[h]);
    }else{
      change = pow(fabs(INPUT_PARAM[t] - INPUT_PARAM[h]), p);
    }
    CHANGE_STAT[0] += IS_OUTEDGE(t,h) ? -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:

- Put the R code at the end of the file “InitErgmTerm.users.R” in the R directory of your ergm.userterms source code directory (or in its own file named *.R in that directory)
- Put the C code at the end of the file “changestats.users.c” in the src directory of your ergm.userterms source code directory
- Open a cmd/terminal window.
- Navigate to your ergm.userterms source code directory
- Type: R CMD build ergm.userterms
- Type: R CMD INSTALL ergm.userterms_3.0-1.tar.gz
- Open R
- Type: library(ergm.userterms)
- Use your new term!!