Chapter 8
Modeling and Prediction for Processes on Network Graphs

Statistical Analysis of Network Data, with R - Eric D. Kolaczyk

Thu Nguyen

10 July, 2019

Contents

Libraries

```
library(igraph)
library(igraphdata)
library(sand)
```

---

1 Introduction

We study various phenomena of choice, analogous to stochasti processes defined on network graph: a collection of random variables X, indexed on $G = (V, E)$, where X can be either

- $\{X_i\}$, for $i \in V_G$, or *static processes*, or
- $\{X_i(t)\}$, where t varies in a discrete or continuous manner, or *dynamic processes*

---

2 Nearest Neighbor Methods

Let $\mathbf{X} = (X_i)$ be a collection of vertex attributes. For illustration, we use the `ppi.CC` dataset in predicting protein functions.

```
set.seed(42)
data(ppi.CC)
summary(ppi.CC)
```

```
## IGRAPH NA UN-- 134 241 --
## + attr: name (v/c), ICSC (v/n), IPR000198 (v/n), IPR000403 (v/n), IPR001806 (v/n), IPR001849
## | (v/n), IPR002041 (v/n), IPR003527 (v/n)
```

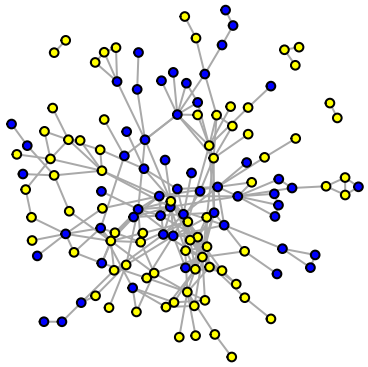
We are going to look at the binary ICSC attribute in particular:

```
V(ppi.CC)$ICSC[1:10]
```

```
##  [1] 1 1 1 1 1 0 1 1 1 1
```

A 1 for ICSC indicates cellular communication, which can be visualized below:

```
par(mar=c(0,0,0,0))
V(ppi.CC)[ICSC == 1]$color <- 'yellow'
V(ppi.CC)[ICSC == 0]$color <- 'blue'
plot(ppi.CC, vertex.size = 5, vertex.label = NA)
```



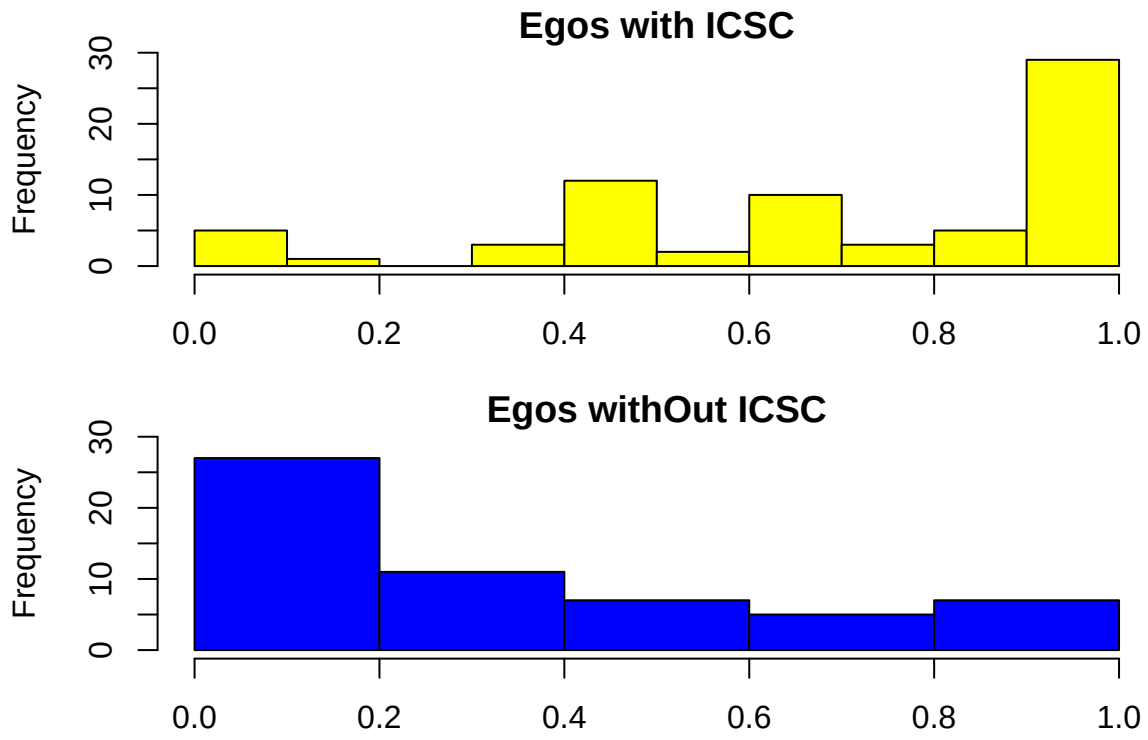
Comment: there appears a good amount of *homogeneity*: neighbors sharing the same attributes (via same colors), which supports the idea of **nearest-neighbor average**. For a given vertex i:

$$\frac{\sum_{j \in N_i} x_j}{|N_i|}$$

is the attribute average among the neighbors of that vertex i, which is to be compared against some threshold. To illustrate, we are going to calculate the **nearest-neighbor average** for each vertex in `ppi.CC`. To cluster in `R`: `clusters()`, to make an induced subgraph: `induced.subgraph()`, to pick out neighbors of a vertex: `V()[nei()]`:

```
clu <- clusters(ppi.CC)
ppi.CC.gc <- induced.subgraph(ppi.CC, clu$membership == which.max(clu$csize))
nn.ave <- sapply(V(ppi.CC.gc), function(x) mean(V(ppi.CC.gc)[nei(x)]$ICSC))
```

```
par(mfrow=c(2,1), mar=c(3,4,1,0))
hist(nn.ave[V(ppi.CC.gc)$ICSC==1], col='yellow', ylim=c(0,30), main='Egos with ICSC',
     xlab='Proportion of Neighbors with ICSC')
hist(nn.ave[V(ppi.CC.gc)$ICSC==0], col='blue', ylim=c(0,30), main='Egos withOut ICSC',
     xlab='Proportion of Neighbors withOut ICSC')
```



Interpretation: *nearest-neighbor average* can predict fairly accurately:

```
nn.pred <- as.numeric(nn.ave > .5)
print(paste('Error rate at threshold of .5:', round(mean(as.numeric(nn.pred != V(ppi.CC.gc)$ICSC)), 4)))
```

```
## [1] "Error rate at threshold of .5: 0.2598"
```

We can further elaborate the illustration: thanks to the evolving nature, we can distinguish proteins between those who do not have the functions against those whose attributes are *unknown* by comparing against more recent version in the `GOstats` package from the `Bioconductor` package:

```
# source('http://bioconductor.org/biocLite.R')
# biocLite('GOstats', suppressAutoUpdate=TRUE, suppressUpdates=TRUE)
library(GOstats); library(GO.db)
# biocLite('org.Sc.sgd.db', suppressAutoUpdate=TRUE, suppressUpdates=TRUE)
library(org.Sc.sgd.db)
```

```
x <- as.list(org.Sc.sgdGO2ALLORFS)
current.icst <- x[names(x) == 'GO:0035556']
ev.code <- names(current.icst[[1]])
```

```
icst.ida <- current.icst[[1]][ev.code == 'IDA']          # ICSC from the new data
orig.icsc <- V(ppi.CC.gc)[ICSC == 1]$name                # ICSC from the original data
candidates <- intersect(icst.ida, V(ppi.CC.gc)$name)     # proteins in new & original data
new.icsc <- setdiff(candidates, orig.icsc)               # newly discovered proteins
print(cat('Newly discovered proteins not in the original dataset:', '\n', new.icsc, '\n'))
```

```
## Newly discovered proteins not in the original dataset:
##  YDL159W YDL235C YHL007C YIL033C YIL147C YLR006C YLR362W
## NULL
```

Probabily of each of those newly discovered proteins:

```
nn.ave[V(ppi.CC.gc)$name %in% new.icsc]
```

```
##   YIL033C   YLR362W   YDL159W   YLR006C   YHL007C   YDL235C   YIL147C
## 0.7500000 0.4166667 0.3333333 0.6666667 0.8750000 0.0000000 0.0000000
```

Interpretation: under the threshold of .5, 3 of them would have been positively predicted.

---

3 Markov Random Fields

A formal statistical model can allow for probabilistically rigorous predictive statements and estimation and testing of model parameters of both network (*endogenous*) and non-network (*exogenous*) effects, an example of which is **Markov Random Fields (MRF)**.

3.1 General Characterization

Let $G = (V, E)$, $\mathbf{X} = (X_1, \ldots, X_n)^T$ be a collection of discrete random variables of vertices of G. Let $\mathbf{X}_{(-i)} = (X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)^T$ and $\mathbf{X}_{\mathbf{N_i}}$ be a vector of all X_j for $j \in N_i$, then $\mathbf{X}$ is an MRF on G if:

$$\forall \mathbf{x} \in R^{(n)} : \mathbb{P}(\mathbf{X} = \mathbf{x}) > 0, \text{ and}$$
$$\mathbb{P}\Big(X_i = x_i | \mathbf{X}_{(-i)} = \mathbf{x}_{(-i)}\Big) = \mathbb{P}\Big(X_i = x_i | \mathbf{X}_{N_i} = \mathbf{x}_{N_i}\Big) \qquad (*)$$

$(*)$ implies that X_i is conditionally independent of all other X_k given the values of its neighbors. Under appropriate conditions, MRF is equivalent to *Gibbs random fields*:

$$\mathbb{X}(\mathbf{X} = \mathbf{x}) = \frac{1}{\kappa} \exp\{ U(\mathbf{x}) \}$$

where $U(\cdot)$ is the *energy function* and $\kappa = \sum_{\mathbf{x}} \exp\{ U(\mathbf{x})\}$ the *partition function*. In particular:

$$U(\mathbf{x}) = \sum_{c \in \mathcal{C}} U_c(\mathbf{x})$$

where $\mathcal{C}$ is the set of all cliques of all sizes in G.

3.2 Auto-Logitstic Models

Suppose (i) only cliques $c \in \mathcal{C}$ of size 1 or 2 have non-zero potential functions U_c, and (ii) the function $(*)$ have an exponential family form, then for some $H_i(\cdot)$ and $\{\beta_{ij}\}$, we have **auto-models**:

$$U(\mathbf{x}) = \sum_{i \in V_G} x_i H_i(x_i) + \sum_{\{i,j\} \in E_G} \beta_{ij} x_i x_j$$

Further, suppose that X_i are binary, then for some $\{\alpha_i\}$:

$$U(\mathbf{x}) = \sum_{i \in V_G} \alpha_i x_i + \sum_{\{i,j\} \in E_G} \beta_{ij} x_i x_j$$

which gives an **auto-logistic** model:

$$\mathbb{P}\Big(X_i = 1 | \mathbf{X}_{(N_i)} = \mathbf{x}_{(N_i)}\Big) = \frac{\exp\left(\alpha_i + \sum_{j \in N_i} \beta_{ij} x_j\right)}{1 + \exp\left(\alpha_i + \sum_{j \in N_i} \beta_{ij} x_j\right)}$$

Such *homogeneous auto-logistic* models are effectively probabilistic extensions of *nearest-neighbor* methods. To fit models using *auto-logistic* method, in `R`, we use the `ngspatial` package:

```
library(ngspatial)
X <- V(ppi.CC.gc)$ICSC
A <- get.adjacency(ppi.CC.gc, sparse = FALSE)
```

To specify such models, we need:

- the network process $\mathbf{X}$, as above
- the network G, as above (A)
- the set of *exogeneous* variables, such as having only an intercept or as conditioned on certain genes:

```
formula1 <- X ~ 1
gene.motifs <- cbind(V(ppi.CC.gc)$IPR000198, V(ppi.CC.gc)$IPR000403, V(ppi.CC.gc)$IPR001806,
                     V(ppi.CC.gc)$IPR001849, V(ppi.CC.gc)$IPR002041, V(ppi.CC.gc)$IPR003527)
formula2 <- X ~ gene.motifs
```

3.3 Inference and Prediction for Auto-Logistic Models

Motivation: predicting network processes $\mathbf{X}$ given parameters α and β via *Maximum Likelihood* method (or rather log likelihood). In R: the function `autologistic()` from the `ngspatial` package:

```
m1.mrf <- autologistic(formula1, A = A, control = list(confint = 'none'))
m1.mrf$coefficients
```

```
## (Intercept)         eta
##   0.2004949   1.1351942
```

Interpretation: having 1 extra neighbor increases the log-odds of having ICSC by a factor of 1.135.

```
mrf1.pred <- as.numeric((m1.mrf$fitted.values > .5))
print(paste('Error rate:', round(mean(as.numeric(mrf1.pred != V(ppi.CC.gc)$ICSC)),4)))
```

```
## [1] "Error rate: 0.2047"
```

```
m1.mrf$fitted.values[V(ppi.CC.gc)$name %in% new.icsc]
```

```
## [1] 0.7519142 0.1658647 0.2184092 0.6451897 0.9590030 0.2595863 0.3956048
```

Interpreation: * *auto-logistic* model has a slightly better error rate compared to the *nearest-neighbor* method: 20% vs. 25% * with respect to discovering new ICSC-bearing proteins, they return the same predictions

Now, we can try adding gene motif information:

```
m2.mrf <- autologistic(formula2, A = A, control = list(confint = 'none'))
m2.mrf$coefficients
```

```
##        (Intercept)       gene.motifs1       gene.motifs2       gene.motifs3       gene.motifs4
##   0.05081573292421   1.87684802831004  18.75217094733102  18.75217075179280  18.24990124753044
##       gene.motifs5       gene.motifs6                eta
##   0.00000008487244 -18.37996655553511   1.29792135705383
```

```
mrf2.pred <- as.numeric((m2.mrf$fitted.values > .5))
print(paste('Error rate:', round(mean(as.numeric(mrf2.pred != V(ppi.CC.gc)$ICSC)),4)))
```

```
## [1] "Error rate: 0.189"
```

```
m2.mrf$fitted.values[V(ppi.CC.gc)$name %in% new.icsc]
```

```
## [1] 0.7829254 0.4715219 0.4962188 0.6570828 0.7829254 0.2175373 0.3510037
```

3.4 Goodness-of-Fit

To simulate realizations of centered *auto-logistc* models in `R`: `rautologistic()` from the `ngspatial` package:

```
srand(42)
ntrials <- 100
a1.mrf <- numeric(ntrials)
a2.mrf <- numeric(ntrials)
Z1 <- rep(1, length(X))
Z2 <- cbind(Z1, gene.motifs)
for (i in 1:ntrials) {
  X1.mrf <- rautologistic(as.matrix(Z1), A=A, theta=m1.mrf$coefficients)
  X2.mrf <- rautologistic(as.matrix(Z2), A=A, theta=m2.mrf$coefficients)
  a1.mrf[i] <- assortativity(ppi.CC.gc, X1.mrf+1, directed=FALSE)
  a2.mrf[i] <- assortativity(ppi.CC.gc, X2.mrf+1, directed=FALSE)
}
(k <- assortativity(ppi.CC.gc, X+1, directed = FALSE))
```

```
## [1] 0.3739348
```

```
summary(a1.mrf)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.09479 0.20269 0.27826 0.28759 0.34924 0.53012
```

```
summary(a2.mrf)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.007286  0.232610  0.291550  0.287513  0.354014  0.492143
```

Interpretation: given the *assortativity* coefficient of 0.3739348, which lies in the upper quartile of both of 2 simulations, the GOF is not bad, but can be improved.

---

4 Kernel Methods

Kernel methods can be thought of as extension of classical regression:

(i) generalized notion of predictor variables through *kernels*

(ii) regression with penalty

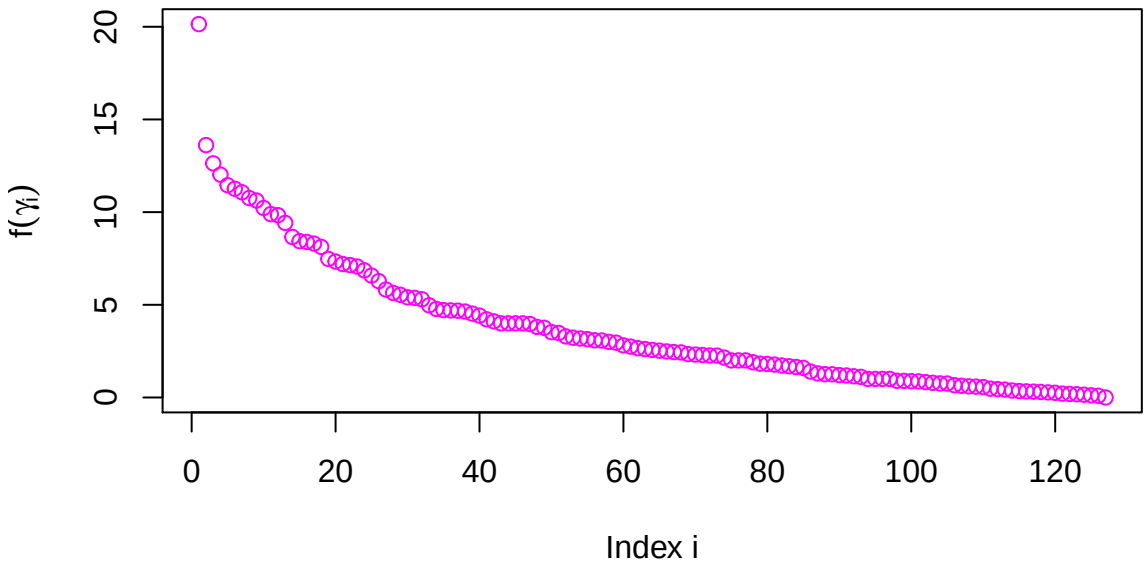
4.1 Designing Kernels on Graphs

Kernel describes the similarity among vertices of G in matrices. A (*positve semi-definite*) **kernel** is a function $K : (i, j) \to k$, where (i, j) is a vertex pair, such that $\forall m \in [n]$ and subset $\{i_1, \ldots, i_m\} : \mathbf{K}^{(m)} = [K(i_j, i_{j'})]$ is an $m \times m$ symmetric and positive semi-definite matrix.

Recall that the graph Laplacian $L := D - A$, where A is adjacency matrix and D is the vertex degree matrix. Then the *Laplacian kernel* is the *Laplacian* inverse: $K := L^{-1}$.

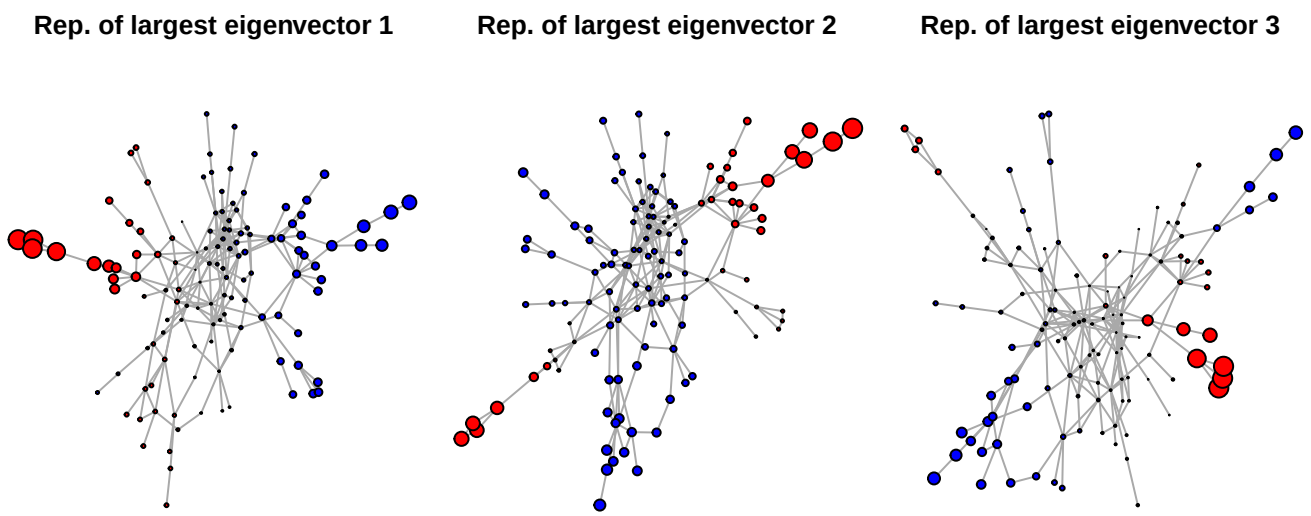
The *kernel* K thus encourages the regression to be "locally" smooth wrt the topology of G.

```
par(mar=c(4,4,.5,.5))
L <- as.matrix(graph.laplacian(ppi.CC.gc))
e.L <- eigen(L)
nv <- vcount(ppi.CC.gc)
e.vals <- e.L$values[1:(nv-1)]
f.e.vals <- c((e.vals)^(1), 0)
plot(f.e.vals, col='magenta', lwd=1, xlim=c(1,nv), xlab=c('Index i'), ylab=expression(f(gamma[i])))
```




```
par(mfrow=c(1,3), mar=c(0,0,1,0))
for (i in 1:3) {
  e.vec <- e.L$vectors[, (nv-i)]
  v.colors <- character(nv)
  v.colors[e.vec >= 0] <- 'red'
  v.colors[e.vec < 0] <- 'blue'
  v.size <- 15 * sqrt(abs(e.vec))
```

```
  l <- layout.fruchterman.reingold(ppi.CC.gc)
  plot(ppi.CC.gc, layout=l, vertex.color=v.colors, vertex.size=v.size, vertex.label=NA)
  title(paste('Rep. of largest eigenvector', i))
}
```



To apply *kernel regression*, we use the `kernlab` package:

```
library(kernlab)
K1.tmp <- e.L$vectors %*% diag(f.e.vals) %*% t(e.L$vectors)
K1 <- as.kernelMatrix(K1.tmp)
K.motifs <- gene.motifs %*% t(gene.motifs)
K2.tmp <- .5 * K1.tmp + .5 * K.motifs
K2 <- as.kernelMatrix(K2.tmp)
```

4.2 Kernel Regression on Graphs

Let $G = (V, E)$ and $X = (X_1, \ldots, X_n)$ be vertex attribute. The goal is to find a map $\hat{f} : V \to \mathbb{R}$ which best describe the characteristic of the vertices. Equivalently, given a kernel $\mathbf{K}$ with eigen-decomposition $\mathbf{K} = \phi\mathbf{\Delta}\phi^{\mathbf{T}}$ and the class of all $\mathbf{h}$:

$$\mathcal{H}_K = \{\mathbf{h} : \mathbf{h} = \phi\beta \text{ and } \beta^T\Delta^{-1}\beta < \infty\}$$

Thus, an estimate $\hat{\mathbf{h}} = \phi\hat{\beta}$ is obtained by findind $\hat{\beta}$ s.t:

$$\min\left\{ \sum_{i \in V^{obs}} C(\ x_i \mid (\phi\beta)_i\ ) + \lambda\beta^T\Delta^{-1}\lambda \right\}$$

where $C(\cdot|\cdot)$ is some convex loss function:

- the loss captured y $C(\cdot|\cdot)$ encourages GOF of model,
- the penalty $\lambda\beta^T\Delta^{-1}\lambda$ penalizes excessive complexity: eigen-vectors with small eigen-values are penalized more harshly.

```
m1.svm <- ksvm(K1, X, type = 'C-svc')           # K1 model
m1.svm.fitted <- fitted(m1.svm)
print(paste('Error rate:', round(mean(as.numeric(m1.svm.fitted != V(ppi.CC.gc)$ICSC)),4)))
```

```
## [1] "Error rate: 0.0866"
```

which is more than half that of the *Markov random field* model.

```
m2.svm <- ksvm(K2, X, type="C-svc")          # K2 model
m2.svm.fitted <- fitted(m2.svm)
print(paste('Error rate:', round(mean(as.numeric(m2.svm.fitted != V(ppi.CC.gc)$ICSC)),4)))
```

```
## [1] "Error rate: 0.0236"
```

---

5 Modeling and Prediction for Dynamic Processes

5.1 Epidemic Processes: An Illustration

Recall the Continuous-time Markov Chain process from Math 180C. Let $(\mathbf{X}(t) = (X_i(t))_{i \in V_G}$ be the continuous time-indexed process on G. Let $\mathbf{x}$ be the process's state at time t, then:

$$\mathbb{P}(\mathbf{X}(t+h) = \mathbf{x}' | \mathbf{X}(t) = \mathbf{x}) \approx \begin{cases} \beta M_i(\mathbf{x})h, & \text{if } x_i = 0, x_i' = 1 \\ \gamma h, & \text{if } x_i = 1, x_i' = 2 \\ 1 - [\beta M_i(\mathbf{x}) + \gamma]h, & \text{if } x_i = 2, x_i' = 2 \end{cases}$$

where $M_i(\mathbf{x}) = \#\{j \in N_i : x_j = 1\}$, the number of infected neighbors of i at time t. Let $(N_S(t), N_I(t), N_R(t))$ be the number of susceptible, infected, and recovered people. We are going to illustrate through different random graphs.

```
gl <- list()
gl$ba <- barabasi.game(250, m=5, directed=FALSE)
gl$er <- erdos.renyi.game(250, 1250, type=c('gnm'))
gl$ws <- watts.strogatz.game(1, 100, 12, .01)
```

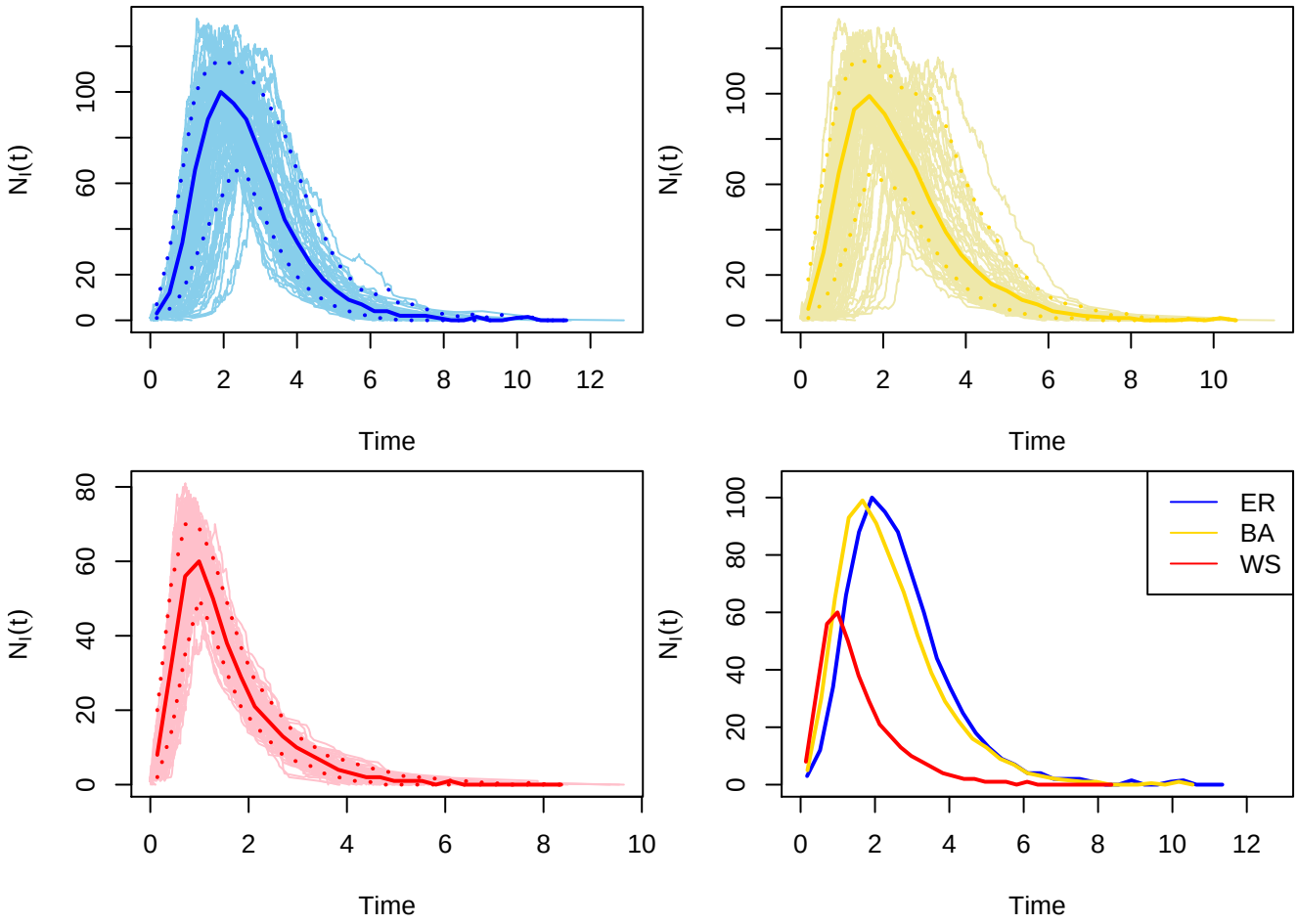
Let the infection rate $\beta = .5$, the recovery rate $\gamma = 1$:

```
beta <- .5; gamma <- 1
```

```
ntrials <- 100
sim <- lapply(gl, sir, beta=beta, gamma=gamma, no.sim=ntrials)
```

```
par(mfrow=c(2,2), mar=c(4,4,0.5,0.5))
plot(sim$er)
plot(sim$ba, color = 'palegoldenrod', median_color = 'gold', quantile_color = 'gold')
plot(sim$ws, color = 'pink', median_color = 'red', quantile_color = 'red')

x.max <- max(sapply(sapply(sim, time_bins), max))
y.max <- 1.05 * max(sapply(sapply(sim, function(x) median(x)[["NI"]]), max, na.rm=TRUE))
plot(time_bins(sim$er), median(sim$er)[['NI']], type='l', lwd=2, col='blue',
     xlim=c(0, x.max), ylim=c(0, y.max), xlab='Time', ylab=expression(N[I](t)))
lines(time_bins(sim$ba), median(sim$ba)[['NI']], lwd=2, col='gold')
lines(time_bins(sim$ws), median(sim$ws)[['NI']], lwd=2, col='red')
legend('topright', c('ER', 'BA', 'WS'), col=c('blue', 'gold', 'red'), lty=1)
```


N_I(t)
Time
ER
BA
WS